Руководство по PHP ¶

от:
Mehdi Achour
Friedhelm Betz
Antony Dovgal
Nuno Lopes
Hannes Magnusson
Georg Richter
Damien Seguy
Jakub Vrana
И некоторые другие
2016-09-18
Под редакцией: Peter Cowburn
от:
Алексей Шеин
Андрей Безруков
Максим Чабан
Александр Москалёв
Борис Клименко
Дмитрий Винярчук
Борис Флейтлих
Алексей Егоров
Юрий Бабиков
Михаил Баранов
И некоторые другие
© 1997-2016 Группа документирования PHP

Авторские права

Авторские права © 1997 - 2016 принадлежат группе документирования PHP. Эти материалы могут быть распространены только на условиях и соглашениях, установленных в лицензии Creative Commons Attribution версии 3.0 или более поздней. Копия лицензииCreative Commons Attribution 3.0 распространяется вместе с руководством. Последняя на данный момент версия доступна по адресу» http://creativecommons.org/licenses/by/3.0/.

Если вы заинтересованы в распространении или публикации данного документа полностью или частично, с изменениями или без, и у вас есть вопросы, вы можете связаться с владельцами авторского права по адресу » doc-license@lists.php.net. Помните, что это адрес публичного списка рассылки.

Руководство по PHP ¶

Предисловие

PHP, расшифровывающийся как "*PHP: Hypertext Preprocessor*" - «PHP: Препроцессор Гипертекста», является распространенным интерпретируемым языком общего назначения с открытым исходным кодом. PHP создавался специально для ведения web-разработок и код на нем может внедряться непосредственно в HTML-код. Синтаксис языка берет начало из C, Java и Perl, и является легким для изучения. Основной целью PHP является предоставление web-разработчикам возможности быстрого создания динамически генерируемых web-страниц, однако область применения PHP не ограничивается только этим.

Это руководство состоит, главным образом, из справочника функций, а также содержит справочник языка, комментарии к наиболее важным из отличительных особенностей PHP, и другие дополнительные сведения.

Это руководство доступно в нескольких форматах по адресу » http://www.php.net/download-docs.php. Более подробную информацию о том, как ведется работа над руководством, вы сможете получить обратившись к приложению Об этом руководстве. Если вам интересна история PHP, обратитесь к соответствующему приложению.

Авторы и Участники

Мы публикуем имена самых активных на данный момент участников разработки документации на первой странице, но есть еще множество людей, которые помогают нам в процессе разработки или оказали неоценимую помощь в прошлом. Также, есть огромное количество людей, помогающих нам с помощью системы пользовательских замечаний, которые постоянно включаются в основную документацию. Все списки ниже отсортированы в алфавитном порядке.

Авторы и Редакторы

Следующие участники внесли значительный вклад в наполнение документации в прошлом: Bill Abt, Jouni Ahto, Alexander Aulbach, Daniel Beckham, Stig Bakken, Nilgün Belma Bugüner, Jesus M. Castagnetto, Ron Chmara, Sean Coates, John Coggeshall, Simone Cortesi, Peter Cowburn, Daniel Egeberg, Markus Fischer, Wez Furlong, Sara Golemon, Rui Hirokawa, Brad House, Pierre-Alain Joye, Etienne Kneuss, Moriyoshi Koizumi, Rasmus Lerdorf, Andrew Lindeman, Stanislav Malyshev, Justin Martin, Rafael Martinez, Rick McGuire, Moacir de Oliveira Miranda Júnior, Kalle Sommer Nielsen, Yasuo Ohgaki, Philip Olson, Richard Quadling, Derick Rethans, Rob Richards, Sander Roobol, Egon Schmid, Thomas Schoefbeck, Sascha Schumann, Dan Scott, Masahiro Takagi, Yannick Torres, Michael Wallner, Lars Torben Wilson, Jim Winstead, Jeroen van Wolffelaar и Andrei Zmievski.

Следующие участники внесли значительный вклад в редактирование и корректирование документации в прошлом: Stig Bakken, Gabor Hojtsy, Hartmut Holzgraefe, Philip Olson и Egon Schmid.

Редакторы пользовательских замечаний

Текущие наиболее активные редакторы: Daniel Brown, Nuno Lopes, Felipe Pena, Thiago Pojda и Maciek Sokolewicz.

Эти люди внесли свой вклад в управление пользовательскими замечаниями в прошлом: Mehdi Achour, Daniel Beckham, Friedhelm Betz, Victor Boivie, Jesus M. Castagnetto, Nicolas Chaillan, Ron Chmara, Sean Coates, James Cox, Vincent Gevers, Sara Golemon, Zak Greant, Szabolcs Heilig, Oliver Hinckel, Hartmut Holzgraefe, Etienne Kneuss, Rasmus Lerdorf, Matthew Li, Andrew Lindeman, Aidan Lister, Hannes Magnusson, Maxim Maletsky, Bobby Matthis, James Moore, Philip Olson, Sebastian Picklum, Derick Rethans, Sander Roobol, Damien Seguy, Jason Sheets, Tom Sommer, Jani Taskinen, Yasuo Ohgaki, Jakub Vrana, Lars Torben Wilson, Jim Winstead, Jared Wyles и Jeroen van Wolffelaar.

Другие переводчики ¶

Наиболее активные переводчики присылавшие патчи: Spbima, Тигрович, Иван Ремень, Serg, Wolg, Degit, Nooneon, StelZek, Grul, Alexandr Fedotov, Marina Lagun, Mike, HaJIuBauKa, Sunny, dba.

Мы публикуем имена самых активных на данный момент переводчиков документации, но есть еще множество людей, которые помогают нам. И каждый их перевод, поправка или совет ценен и важен.

Переводчики, ранее принимавшие участие в переводе руководства на русский язык (в алфавитном порядке): Alexander Voytsekhovskyy, Alexey Asemov, Andrey Demenev, Antony Dovgal, Boris Bezrukov, Evgeniy Syuzev, Irina Goble, Ivan Kovalenko, Jigkayev Kazbek, Kirill Barashkin, Olishuk Andrey, Veniamin Zolotukhin.

А также неизвестные переводчики со следующими никами: freespace, santiago, shafff, sveta.

Приступая к работе ¶

Введение

Содержание

Что такое PHP? ¶

PHP (рекурсивный акроним словосочетания *PHP: Hypertext Preprocessor*) - это распространенный язык программирования общего назначения с открытым исходным кодом. PHP сконструирован специально для ведения Web-разработок и его код может внедряться непосредственно в HTML.

Простой ответ, но что он может означать? Вот пример кода:

Пример #1 Пример программирования на PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>

        <?php
        echo "Привет, я - скрипт PHP!";
        ?>

    </body>
</html>
```

Вместо рутинного вывода HTML-кода командами языка (как это происходит, например, в Perl или C), скрипт PHP содержит HTML с вкраплениями кода (в нашем случае, это вывод текста "Привет, я - скрипт PHP!"). Код PHP отделяется специальными начальным и конечным тегами <?php и ?>, которые позволяют "переключаться" в "PHP-режим" и выходить из него.

PHP отличается от JavaScript тем, что PHP-скрипты выполняются на сервере и генерируют HTML, который посылается клиенту. Если бы у вас на сервере был размещен скрипт, подобный вышеприведенному, клиент получил бы только результат его выполнения, но не смог бы выяснить, какой именно код его произвел. Вы даже можете настроить свой сервер таким образом, чтобы обычные HTML-файлы обрабатывались процессором PHP, так что клиенты даже не смогут узнать, получают ли они обычный HTML-файл или результат выполнения скрипта.

PHP крайне прост для освоения, но вместе с тем способен удовлетворить запросы профессиональных программистов. Не пугайтесь длинного списка возможностей PHP. Вы можете быстро начать, и уже в течение нескольких часов сможете создавать простые PHP-скрипты.

Хотя PHP, главным образом, предназначен для работы в среде web-серверов, область его применения не ограничивается только этим. Читайте дальше и не пропустите главу Возможности PHP либо, начните непосредственно с Вводного руководства, если вас интересует исключительно веб-программирование.

Возможности PHP

PHP может все. Главная область применения PHP - написание скриптов, работающих на стороне сервера; таким образом, PHP способен выполнять все то, что выполняет любая другая программа CGI, например, обрабатывать данные форм, генерировать динамические страницы или отсылать и принимать cookies. Но PHP способен выполнять намного больше.

Существуют три основных области применения PHP.

- Создание скриптов для выполнения на стороне сервера. PHP традиционно и наиболее широко используется именно таким образом. Для этого вам будут необходимы три вещи. Интерпретатор PHP (в виде программы CGI или серверного модуля), веб-сервер и браузер. Для того чтобы можно было просматривать результаты выполнения PHP-скриптов в браузере, нужен

работающий веб-сервер и установленный PHP. Просмотреть вывод PHP-программы можно в браузере, получив PHP-страницу, сгенерированную сервером. В случае, если вы просто экспериментируете, вы вполне можете использовать свой домашний компьютер вместо сервера. За более подробными сведениями обратитесь к главе Советы по установке.

- Создание скриптов для выполнения в командной строке. Вы можете создать PHP-скрипт, способный запускаться без сервера или браузера. Все, что вам потребуется - парсер PHP. Такой способ использования PHP идеально подходит для скриптов, которые должны выполняться регулярно, например, с помощью cron (на платформах *nix или Linux) или с помощью планировщика задач (Task Scheduler) на платформах Windows. Эти скрипты также могут быть использованы в задачах простой обработки текстов. За дополнительной информацией обращайтесь к главе Использование PHP в среде командной строки.
- Создание оконных приложений, выполняющихся на стороне клиента. Возможно, PHP является не самым лучшим языком для создания подобных приложений, но, если вы очень хорошо знаете PHP и хотели бы использовать некоторые его возможности в своих клиентских приложениях, вы можете использовать PHP-GTK для создания таких приложений. Подобным образом вы можете создавать и кросс-платформенные приложения. PHP-GTK является расширением PHP и не поставляется вместе с основным дистрибутивом PHP. Если вы заинтересованы, посетите » сайт PHP-GTK.

PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix (такие как HP-UX, Solaris и OpenBSD), Microsoft Windows, Mac OS X, RISC OS, и многие другие. Также в PHP включена поддержка большинства современных веб-серверов, таких как Apache, IIS и многих других. В принципе, подойдет любой веб-сервер, способный использовать бинарный файл FastCGI PHP, например, lighttpd или nginx. PHP может работать в качестве модуля или функционировать в качестве процессора CGI.

Таким образом, выбирая PHP, вы получаете свободу выбора операционной системы и веб-сервера. Более того, у вас появляется выбор между использованием процедурного или объектно-ориентированного программирования (ООП) или же их сочетания.

PHP способен генерировать не только HTML. Доступно формирование изображений, файлов PDF и даже роликов Flash (с использованием libswf и Ming), создаваемых «на лету». PHP также способен генерировать любые текстовые данные, такие, как XHTML и другие XML-файлы. PHP может осуществлять автоматическую генерацию таких файлов и сохранять их в файловой системе вашего сервера вместо того, чтобы отдавать клиенту, организуя, таким образом, серверный кэш для вашего динамического контента.

Одним из значительных преимуществ PHP является поддержка широкого круга баз данных. Создать скрипт, использующий базы данных, - невероятно просто. Можно воспользоваться расширением, специфичным для отдельной базы данных (таким как mysql) или использовать уровень абстракции от базы данных, такой как PDO, или подсоединиться к любой базе данных, поддерживающей Открытый Стандарт Соединения Баз Данных (ODBC), с помощью одноименного расширения ODBC. Для других баз данных, таких как CouchDB, можно воспользоваться cURL или сокетами.

PHP также поддерживает «общение» с другими сервисами через такие протоколы, как LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (на платформах Windows) и многих других. Кроме того, вы получаете возможность работать с сетевыми сокетами напрямую. PHP поддерживает стандарт обмена сложными структурами данных WDDX практически между всеми языками веб-программирования. Обращая внимание на взаимодействие между различными языками, следует упомянуть о поддержке объектов Java и возможности их использования в качестве объектов PHP.

PHP имеет много возможностей по обработке текста, включая регулярные выражения Perl (PCRE) и много других расширений и инструментов для обработки и доступа к XML документам. В PHP обработка XML-документов стандартизирована и происходит на базе мощной библиотеки libxml2, расширив возможности обработки XML добавлением новых расширений SimpleXML, XMLReader иXMLWriter.

Есть еще многих других интересных расширений, которые можно просмотреть как в алфавитном порядке, так и по категориям. Есть еще много дополнительных PECL расширений, которые также могут (а может и нет) быть документированы в данном руководстве, такие как » XDebug.

Как вы видите, этой страницы не хватит для того, чтобы перечислить все, что может предложить вам PHP. Читайте следующую главу,Установка PHP и обратитесь к главе Справочник по функциям за более подробными сведениями о перечисленных выше расширениях.

Простой учебник

Содержание

В этом кратком и простом учебнике мы бы хотели показать самые основы PHP. Этот текст включает в себя только создание динамических Web-страниц с помощью PHP, однако реальная область применения PHP гораздо шире. В разделе "Что может PHP" приведена дополнительная информация.

Web-страницы, созданные с использованием PHP, обрабатываются как обычные HTML-страницы. Их можно создавать и изменять точно таким же образом, как и обычные страницы на HTML.

Что мне потребуется?

В данном руководстве мы предполагаем, что ваш сервер имеет поддержку PHP и что все файлы, заканчивающиеся на `.php`, обрабатываются PHP. В большинстве серверов обычно используется это расширение для PHP-файлов, но все-таки не лишним будет уточнить это у вашего администратора сервера. Итак, если ваш сервер поддерживает PHP, то у вас есть все, что требуется. Просто создавайте

файлы `.php` и размещайте их в вашем каталоге Web-сервера - они будут обрабатываться автоматически. Не нужно ничего компилировать, не нужно никаких дополнительных программ. Считайте файлы PHP обычными файлами HTML с набором новых "волшебных" тегов, которые могут делать кучу разных вещей.

Например, вы хотите сэкономить на интернет-канале и вести разработку локально. В этом случае вам нужно будет установить веб-сервер, такой как » Apache, и, разумеется, » PHP. Скорее всего, вы также захотите установить базу данных, например, » MySQL.

Все это может быть установлено как отдельно друг от друга, так и более простым способом. В нашем руководстве есть инструкции по установке PHP (предполагается, что вы уже установили веб-сервер). Если у вас возникли проблемы при установке PHP, мы предлагаем вам задать вопросы в нашем » списке рассылки по вопросам установки. Если же вы выбрали более простой способ, то » найдите уже настроенный пакет для вашей операционной системы, который автоматически установит все вышеперечисленное несколькими кликами мыши. Устанавливать веб-сервер с поддержкой PHP довольно легко на любой операционной системе, включая MacOSX, Linux и Windows. На Linux вам, возможно, пригодятся » rpmfind и » PBone при поиске RPM-пакетов. Можно также использовать » apt-get для поиска пакетов под Debian.

User Contributed Notes *1 note*

Первая страница на PHP

Создайте файл с именем `hello.php` в корневом каталоге веб-сервера (`DOCUMENT_ROOT`) и запишите в него следующее:

Пример #1 Первый скрипт на PHP: *hello.php*

```
<html>
 <head>
  <title>Тестируем PHP</title>
 </head>
 <body>
 <?php echo '<p>Привет, мир!</p>'; ?>
 </body>
</html>
```

Откройте данный файл в браузере, набрав имя вашего веб-сервера и */hello.php*. При локальной разработке эта ссылка может быть чем-то вроде *http://localhost/hello.php* или *http://127.0.0.1/hello.php*, но это зависит от настроек вашего сервера. Если все настроено правильно, этот файл будет обработан PHP и браузер выведет следующий текст:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
```

```
 </body>
</html>
```

Эта программа чрезвычайно проста, и для создания настолько простой странички даже необязательно использовать PHP. Все, что она делает, это вывод *Hello World*, используя инструкцию PHP echo. Заметьте, что файл *не обязан быть выполняемым* или еще как-то отличаться от других файлов. Сервер знает, что этот файл должен быть обработан PHP, так как файл обладает расширением ".php", о котором в настройках сервера сказано, что подобные файлы должны передаваться PHP. Рассматривайте его как обычный HTML-файл, которому посчастливилось заполучить набор специальных тегов (доступных также и вам), способных на кучу интересных вещей.

Если у вас этот пример не отображает ничего или выводит окно загрузки, или если вы видите весь этот файл в текстовом виде, то, скорее всего, ваш веб-сервер не имеет поддержки PHP или был сконфигурирован неправильно. Попросите вашего администратора сервера включить такую поддержку. Предложите ему инструкцию по установке: раздел Установка данной документации. Если же вы разрабатываете скрипты на PHP дома (локально), то также прочтите эту главу, чтобы убедиться, что вы все настроили верно. Убедитесь также, что вы запрашиваете файл у сервера через протокол http. Если вы просто откроете файл из вашей файловой системы, он не будет обработан PHP. Если проблемы все же остались, не стесняйтесь попросить помощи одним из » множества доступных способов получения поддержки по PHP.

Цель примера - показать формат специальных тегов PHP. В этом примере мы использовали *<?php* в качестве открывающего тега, затем шли команды PHP, завершающиеся закрывающим тегом *?>*. Таким образом можно где угодно "запрыгивать" и "выпрыгивать" из режима PHP в HTML файле. Подробнее об этом можно прочесть в разделе руководства Основной синтаксис.

Замечание: Замечание о переводах строк
Переводы строк немногое означают в HTML, однако считается хорошей идеей поддерживать HTML в удобочитаемом виде, перенося его на новую строку. PHP автоматически удаляет перевод строки, идущий сразу после закрывающего тега *?>*. Это может быть чрезвычайно полезно, если вы используете множество блоков PHP-кода или подключаете PHP-файлы, которые не должны ничего выводить. В то же время, это может приводить в недоумение. Можно поставить пробел после закрывающего тега *?>* и тогда пробел будет выведен вместе с переводом строки, или же вы можете специально добавить перевод строки в последний вызов echo/print из блока PHP-кода.
Замечание: Пара слов о текстовых редакторах
Существует множество текстовых редакторов и интегрированных сред разработки (IDE), в которых вы можете создавать и редактировать файлы PHP. Список некоторых редакторов содержится в разделе » Список редакторов PHP. Если вы хотите порекомендовать какой-либо редактор, посетите данную страницу и попросите добавить редактор в список. Использование редактора с подсветкой синтаксиса может быть очень большим подспорьем в вашей работе.
Замечание: Пара слов о текстовых процессорах
Текстовые процессоры (StarOffice Writer, Microsoft Word, Abiword и др.) в большинстве случаев не подходят для редактирования файлов PHP. Если вы все же хотите использовать какой-либо из них для тестового скрипта, убедитесь, что сохра-

няете файл как*простой текст* (plain text), иначе PHP будет не в состоянии прочесть и запустить ваш скрипт.
Замечание: Пара слов о Блокноте Windows
При написании скриптов PHP с использованием встроенного Блокнота Windows необходимо сохранять файлы с расширением `.php`. (Блокнот автоматически добавит расширение `.txt`, если вы не предпримете указанные ниже меры.) Когда во время сохранения файла вас попросят указать его имя, введите имя файла в двойных кавычках (например, "`hello.php`"). Кроме этого, можно кликнуть на выпадающее меню "Текстовые документы" в диалоговом окне сохранения файла и выбрать в нем пункт "Все файлы". После этого можно вводить имя файла без кавычек.

Теперь, когда вы успешно создали работающий PHP-скрипт, самое время создать самый знаменитый PHP-скрипт! Вызовите функциюphpinfo() и вы увидите множество полезной информации о вашей системе и настройке, такой как доступные предопределенные переменные, загруженные PHP-модули и параметры настройки. Уделите некоторое время изучению этой важной информации.

Пример #2 Получение информации о системе из PHP

```
<?php phpinfo(); ?>
```

***User Contributed Notes** 1 note*

Делаем что-нибудь полезное

Давайте сделаем что-нибудь полезное. К примеру, определим, какой браузер использует тот, кто смотрит в данный момент нашу страницу. Для этого мы проверим строку с именем браузера, посылаемую нам в HTTP-запросе. Эта информация хранится впеременной. Переменные в PHP всегда предваряются знаком доллара. Интересующая нас в данный момент переменная называется`$_SERVER['HTTP_USER_AGENT']`.

Замечание:
`$_SERVER` - специальная зарезервированная переменная PHP, которая содержит всю информацию, полученную от Web-сервера. Её также называют суперглобальной. Для более подробной информации смотрите раздел "Суперглобальные переменные". Эти специальные переменные появились в PHP, начиная с версии » 4.1.0. До этого использовались массивы $HTTP_*_VARS, такие как$HTTP_SERVER_VARS. Несмотря на то, что эти массивы уже устарели, они до сих пор существуют (см. замечания по старым программам).

Для вывода данной переменной мы сделаем так:

Пример #1 Вывод значения переменной (элемента массива)

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

Пример вывода данной программы:

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

В PHP существует огромное количество типов переменных. В предыдущем примере мы печатали элемент массива. Массивы могут быть очень полезны.

$_SERVER - это просто одна из переменных, которые предоставляются вам языком PHP. Список таких переменных можно посмотреть в разделе "Зарезервированные переменные" или просмотрев вывод функции phpinfo(), используемой в примере в предыдущем разделе.

Внутрь PHP-тегов можно помещать несколько выражений и создавать маленькие блоки кода, делающие больше, чем простой вызов echo. Например, если вы хотите добавить проверку для Internet Explorer, можно сделать так:

Пример #2 Пример использования управляющих структур и функций

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
    echo 'Вы используете Internet Explorer.<br />';
}
?>
```

Пример вывода данной программы:

```
Вы используете Internet Explorer.<br />
```

Здесь мы показали несколько новых элементов. Во-первых, здесь есть конструкция if. Если вы знакомы с основами синтаксиса языка C, то вы уже заметили что-то схожее. Если же вы не знаете C или подобного по синтаксису языка, то лучший вариант - взять какую-либо вводную книжку по PHP и прочитать первые пару глав. Другой вариант - почитать раздел "Описание языка" данного руководства.

Кроме этого, здесь присутствует вызов функции strpos(). strpos() - встроенная в PHP функция, которая ищет одну строку в другой. В данном случае мы ищем строку *'MSIE'* (так называемую "иголку" - needle) в $_SERVER['HTTP_USER_AGENT'] (в так называемом "сене" - haystack). Если "иголка" найдена внутри "сена", функция возвращает позицию "иголки" относительно начала "сена". В противном случае она возвращает **FALSE**. Если она не вернет **FALSE**, то условие в if окажется истинным (**TRUE**), и код в фигурных скобках ({ }) выполнится. В противном случае этот код не выполняется. Попробуйте создать аналогичные примеры с использованием команд if,else и других функций, таких, как strtoupper() и strlen(). Также подобные примеры содержатся во многих описаниях функций в данном руководстве. Если вы не знаете, как использовать функции, возможно, вам стоит прочесть страницу руководства о том, как читать определения функций и раздел о функциях в PHP.

Продемонстрируем, как можно входить в режим кода PHP и выходить из него даже прямо посередине блока с кодом:

Пример #3 Смешение режимов HTML и PHP

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
?>
<h3>strpos(), должно быть, вернул не false</h3>
<p>Вы используете Internet Explorer</p>
<?php
} else {
?>
<h3>strpos() вернул false</h3>
<p>Вы не используете Internet Explorer</p>
<?php
}
?>
```

Пример вывода данной программы:

```
<h3>strpos(), должно быть, вернул не false</h3>
<p>Вы используете Internet Explorer</p>
```

Вместо использования команды PHP echo для вывода, мы вышли из режима кода и просто послали содержимое HTML. Важный момент здесь то, что логическая структура кода PHP при этом не теряется. Только одна HTML-часть будет послана клиенту в зависимости от результата функции strpos() (другими словами, в зависимости от того, найдена была строка *MSIE* или нет).

User Contributed Notes *3 notes*

Работа с формами

Одно из главнейших достоинств PHP - то, как он работает с формами HTML. Здесь основным является то, что каждый элемент формы автоматически становится доступным вашим программам на PHP. Для подробной информации об использовании форм в PHP читайте раздел Переменные из внешних источников. Вот пример формы HTML:

Пример #1 Простейшая форма HTML

```
<form action="action.php" method="post">
 <p>Ваше имя: <input type="text" name="name" /></p>
 <p>Ваш возраст: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

В этой форме нет ничего особенного. Это обычная форма HTML без каких-либо специальных тегов. Когда пользователь заполнит форму и нажмет кнопку отправки, будет вызвана страница `action.php`. В этом файле может быть что-то вроде:

Пример #2 Выводим данные формы

```
Здравствуйте, <?php echo htmlspecialchars($_POST['name']); ?>.
Вам <?php echo (int)$_POST['age']; ?> лет.
```

Пример вывода данной программы:

```
Здравствуйте, Сергей.
Вам 30 лет.
```

Если не принимать во внимание куски кода с htmlspecialchars() и *(int)*, принцип работы данного кода должен быть прост и понятен.htmlspecialchars() обеспечивает правильную кодировку "особых" HTML-символов так, чтобы вредоносный HTML или Javascript не был вставлен на вашу страницу. Поле age, о котором нам известно, что оно должно быть число, мы можем просто преобразовать в integer, что автоматически избавит нас от нежелательных символов. PHP также может сделать это автоматически с помощью расширенияfilter. Переменные `$_POST['name']` и `$_POST['age']` автоматически установлены для вас средствами PHP. Ранее мы использовали суперглобальную переменную `$_SERVER`, здесь же мы точно так же используем суперглобальную переменную `$_POST`, которая содержит все POST-данные. Заметим, что *метод отправки* (method) нашей формы - POST. Если бы мы использовали метод *GET*, то информация нашей формы была бы в суперглобальной переменной `$_GET`. Кроме этого, можно использовать переменную`$_REQUEST`, если источник данных не имеет значения. Эта переменная содержит смесь данных GET, POST, COOKIE и FILE.

В PHP можно также работать и с XForms, хотя вы найдете работу с обычными HTML-формами довольно комфортной уже через некоторое время. Несмотря на то, что работа с XForms не для новичков, они могут показаться вам интересными. В разделе возможностей PHP у нас также есть короткое введение в обработку данных из XForms.

User Contributed Notes *4 notes*

Использование старых программ с новыми версиями PHP

Сейчас PHP является популярным языком сценариев(скриптов) и поэтому появляется все больше и больше распространяемых кусочков кода, которые вы можете использовать в своих скриптах. В большинстве случаев разработчики PHP старались сохранить совместимость с предыдущими версиями так, чтобы код, написанный для более старой версии, работал (в идеале) и с новыми версиями языка без каких-либо изменений. Однако случается так, что изменения все-таки необходимы.

Есть два важных изменения, которые влияют на старые программы:

- Устаревшие массивы `$HTTP_*_VARS` недоступны с версии PHP 5.4.0. Новые суперглобальные массивы были введены, начиная с PHP» 4.1.0. Это: `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_ENV`, `$_REQUEST` и `$_SESSION`.
- Внешние переменные больше не глобализуются по умолчанию. Другими словами, начиная с версии PHP » 4.2.0, директиваregister_globals в `php.ini` по умолчанию *отключена* ("off"). Рекомендуемый метод доступа к таким переменным - суперглобальные массивы, описанные выше. Более старые программы, книги и руководства могут считать, что данная директива включена ("*on*"). К примеру, если бы эта директива была включена ("*on*"), такой скрипт мог бы использовать переменную `$id`, поступившую из строки URL *http://www.example.com/foo.php?id=42*. Вне за-

висимости от значения данной директивы, переменная `$_GET['id']`будет всегда доступна.

Для дополнительной информации касательно изменений, связанных с переменными, смотрите раздел "Предопределенные переменные" и ссылки этого раздела.

Что дальше?

То, что вы узнали, поможет вам понять большую часть руководства и разобраться в большинстве приведенных примеров программ.

Если вы хотите посмотреть различные презентации и слайды, шире раскрывающие возможности PHP, вы можете посетить сайт с материалами PHP конференций: » http://talks.php.net/

Установка и настройка ¶

Общие инструкции по установке ¶

Перед началом установки вы должны знать для чего вы хотите использовать PHP. Вы можете использовать PHP для целей описанных в разделе ВозможностиPHP?

- Создавать web-сайты и web-приложения (Скрипты на стороне сервера)
- Скрипты командной строки
- GUI-приложения (Приложения с графическим интерфейсом пользователя)

Для первой и наиболее распространенной цели вам нужны три вещи: Сам PHP, веб-сервер и веб-браузер. Вероятно, вы уже имеете веб-браузер и в зависимости от настроек вашей операционной системы, вы также можете иметь и веб-сервер (например Apache в Linux и MacOS X; IIS в Windows). Также вы можете арендовать веб-сервер или некоторое дисковое пространство на веб-сервере. В этом случае вам не нужно устанавливать дополнительное программное обеспечение, только писать PHP скрипты, загружать их на сервер, и просматривать результат их работы в браузере.

В случае установки сервера и PHP самостоятельно у вас есть две варианта установки PHP. Для многих серверов PHP может быть установлен как модуль сервера. Это возможно для таких серверов, как Apache, Microsoft Internet Information Server, Netscape и iPlanet. Если PHP не поддерживает интерфейс для вашего сервера, вы всегда можете использовать его как обработчик CGI или FastCGI. Это означает, что вы должны настроить ваш сервер так, чтобы он исполнял все PHP файлы как CGI-скрипты.

Если вы также собираетесь использовать PHP в командной строке (для генерации изображений, обработки текстов и т.д.), то вам понадобится PHP CLI. Подробнее об этом можно прочитать в разделе Использование PHP в командной строке. В этом случае вам не понадобятся ни сервер ни браузер.

Вы также можете создавать приложения с графическим интерфейсом, используя при этом расширение PHP-GTK. Это требует абсолютно другого подхода, чем программирование под Веб, т.к. вы не выводите HTML или текст, а управляете окнами при помощи PHP. Для получения более полной информации о PHP-GTK, » посетите сайт посвященный этому расширению . PHP-GTK не включен в стандартную поставку PHP.

Начиная с этого момента мы будем описывать установку PHP для веб серверов на Unix и Windows, как модуля сервера и как CGI. Вы также можете найти информацию об использовании PHP в командной строке в следующих разделах.

Исходные коды и бинарные сборки для Windows можно получить здесь: » http://www.php.net/downloads.php. Мы рекомендуем использовать » зеркала для загрузки.

Установка на Unix системы ¶

Содержание ¶

Этот раздел описывает установку и настройку PHP на Unix-системах. Пожалуйста, прочитайте все разделы, касающиеся вашей платформы или веб-сервера, прежде чем вы приступите к установке.

Как сказано в разделе Общие указания по установке, в этом разделе в основном мы описываем установку PHP, ориентированную на веб, однако мы также затрагиваем установку PHP для использования в командной строке.

Есть несколько способов установки PHP на Unix платформы. Один из них - процесс конфигурирования и компиляции исходников, другой - установка из пакетов. Этот документ сфокусирован на установке из исходных кодов. Много Unix-подобных систем имеют различные системы установки программ из пакетов. Они могут помочь при установке стандартных конфигураций, но если вам необходимы другие варианты (такие как безопасный сервер или другой драйвер базы данных), вам может потребоваться скомпилировать PHP и/или ваш веб-сервер. Если вы незнакомы с компилированием собственного программного обеспечения, то, может быть, стоит поискать готовый пакет PHP с нужными вам возможностями, собранный кем-нибудь еще.

Для компиляции PHP из исходных кодов вам потребуется следующее:

- Базовые знания Unix (способность обращаться с "make" и компилятором C)
- Компилятор ANSI C
- Веб-сервер
- Любые компоненты, специфичные для отдельных модулей PHP (такие как библиотеки GD, PDF и т.д.)

При сборке непосредственно из исходников Git или после ручных изменений вам может также понадобиться:

- autoconf: 2.13+ (для PHP < 5.4.0), 2.59+ (для PHP >= 5.4.0)
- automake: 1.4+
- libtool: 1.4.x+ (кроме версии 1.4.2)
- re2c: Версия 0.13.4 или новее
- flex: Версия 2.5.4 (для PHP <= 5.2)
- bison: Версия 1.28 (предпочтительно), 1.35 или 1.75 (для PHP <= 5.4), 2.4.x <= 2.7.x (для PHP 5.5), 2.4.x <= 3.0.0 (для PHP 5.6)

Общая (или начальная) конфигурация PHP задается параметрами скрипта configure. Вы можете просмотреть список допустимых параметров вместе с их кратким пояснением при помощи команды ./configure --help. Различные опции документированы в данном руководстве отдельно, список основных параметров можно просмотреть в приложении Основные параметры конфигурации, тогда как параметры, специфичные для различных расширений, описаны на страницах документации, посвященных этим расширениям.

Когда PHP сконфигурирован, все готово к сборке модулей и/или исполняемых файлов. Об этом должна позаботиться команда make. Если что-то не получилось и вы не можете понять почему, смотрите раздел Проблемы установки.

User Contributed Notes *9 notes*

Apache 1.3.x на системах Unix ¶

Этот раздел включает инструкции по установке Apache и PHP на платформах Unix. Установка PHP с Apache2 описана всоответствующем разделе.

Вы можете менять аргументы configure на шаге 10 ниже. Полный список аргументов доступен в списке основных параметров конфигурации. А параметры, специфичные для различных расширений, описаны в разделах посвященных соответствующим расширениям. В инструкции ниже опущены номера версий - вы должны заменить 'xxx' на номер версии присутствующий в именах скачанных вами файлов.

Пример #1 Инструкция по установке PHP, как подгружаемого модуля Apache

```
1.  gunzip apache_xxx.tar.gz
2.  tar -xvf apache_xxx.tar
3.  gunzip php-xxx.tar.gz
4.  tar -xvf php-xxx.tar
5.  cd apache_xxx
6.  ./configure --prefix=/www --enable-module=so
7.  make
8.  make install
9.  cd ../php-xxx
```

10. Теперь мы сконфигурируем PHP. Здесь вы можете настроить установку PHP при помощи параметров скрипта configure. Вы можете включить или выключить некоторые расширения. Просмотрите вывод команды ./configure --help для получения полного списка параметров конфигурации. В нашем примере мы сконфигурируем PHP очень просто - с поддержкой Apache 1 и MySQL. В вашей системе путь до apxs может отличаться от указанного в примере.

```
./configure --with-mysql --with-apxs=/www/bin/apxs
```

11. `make`
12. `make install`

Если вы решите изменить параметры конфигурации после установки, три последних шага надо будет повторить. Также вам надо будет перезапустить Apache, чтобы новые модули подгрузились и начали работать. Перекомпиляция Apache для этого не потребуется.

Заметьте, что 'make install' установит также PEAR, различные инструменты PHP - такие как phpize, версию PHP для командной строки (PHP CLI) и т.д.

13. Настройте ваш php.ini:

```
cp php.ini-development /usr/local/lib/php.ini
```

Вероятно, вы захотите изменить некоторые настройки в php.ini. Если вы предпочитаете держать файл php.ini в другом месте, используйте параметр `--with-config-file-path=/some/path` в шаге 10.

Если вы используете php.ini-production, прочитайте его, чтобы знать какие изменения в поведении PHP это повлечет.

14. Отредактируйте ваш httpd.conf, чтобы Apache подгружал модуль PHP. Путь в правой части инструкции LoadModule должен указывать на модуль PHP. Команда make install может добавить эту иструкцию автоматически, но этого может и не произойти, поэтому проверьте, чтобы убедиться.

```
LoadModule php5_module libexec/libphp5.so
```

15. Также в секции AddModule файла httpd.conf, несколько ниже `ClearModuleList`, добавьте следующее:

```
AddModule mod_php5.c
```

16. Теперь следует сконфигурировать Apache, чтобы он передавал файлы с некоторыми расширениями на обработку модулю PHP. В нашем примере сделаем это для файлов с расширениями .php и .phtml. Вы можете добавить также другие расширения в конец строки, разделяя их символом пробела.

```
AddType application/x-httpd-php .php .phtml
```

Сделаем то же самое для расширения .phps - модуль PHP покажет его как исходный код с подсветкой синтаксиса, вместо того чтобы исполнять.

```
AddType application/x-httpd-php-source .phps
```

17. Используйте стандартную процедуру запуска сервера. (Нужно остановить и заново запустить сервер, процедура перезапуска с использованием сигналов HUP или USR1 в данном случае не подходит.)

В качестве альтернативы, можно установить PHP, как статический модуль Apache:

Пример #2 Инструкция по статической сборке PHP и Apache

```
1.  gunzip -c apache_1.3.x.tar.gz | tar xf -
2.  cd apache_1.3.x
3.  ./configure
4.  cd ..

5.  gunzip -c php-5.x.y.tar.gz | tar xf -
6.  cd php-5.x.y
7.  ./configure --with-mysql --with-apache=../apache_1.3.x
8.  make
9.  make install

10. cd ../apache_1.3.x

11.         ./configure         --prefix=/www         --activate-
module=src/modules/php5/libphp5.a
    (Строка выше корректна. Файл libphp5.a не существует на
    данном этапе, он будет создан.)

12. make
    (Теперь вы должны получить бинарный файл  httpd, который вы можете
скопировать  в каталог  bin  сервера Apache.  Затем вам надо выполнить "make
install",  если это ваша первая установка)

13. cd ../php-5.x.y
14. cp php.ini-development /usr/local/lib/php.ini

15.  Для  изменения  конфигурации  PHP  можно  отредактировать  файл
/usr/local/lib/php.ini.
    Отредактируйте httpd.conf или srm.conf, добавив туда:
    AddType application/x-httpd-php .php
```

В зависимости от варианта вашей установки Apache и версии Unix, возможно множество путей остановки и запуска сервера. Ниже приведены несколько распространенных команд перезапуска сервера для различных установок Apache и Unix-систем. Замените */path/to/* на реальный путь до приложений в вашей системе.

Пример #3 Примеры команд перезапуска Apache

```
1. Некоторые дистрибутивы Linux и системы SysV:
/etc/rc.d/init.d/httpd restart

2. Использование скрипта apachectl:
/path/to/apachectl stop
/path/to/apachectl start

3. httpdctl и httpsdctl (использующий OpenSSL), аналогично apachectl:
/path/to/httpsdctl stop
/path/to/httpsdctl start
```

4. При использовании mod_ssl или другого SSL сервера, можно сделать ручную остановку и пуск:

```
/path/to/apachectl stop
/path/to/apachectl startssl
```

Пути к apachectl и http(s)dctl часто различаются. Если в вашей системе есть команды *locate* или *whereis*, или *which*, они помогут вам найти программы, контролирующие сервер.

Другие варианты компиляции PHP для Apache:

```
./configure --with-apxs --with-pgsql
```

Будет создан файл подгружаемой библиотеки `libphp5.so`. Эта библиотека должна быть подгружена в Apache с использованием директивы LoadModule конфигурационного файла `httpd.conf`. В данном случае поддержка PostgreSQL будет встроена в эту библиотеку.

```
./configure --with-apxs --with-pgsql=shared
```

Будет создан файл подгружаемой библиотеки `libphp5.so` для Apache, но также будет создана подгружаемая библиотека `pgsql.so`, которая должна быть подгружена в PHP одним из двух способов: директивой extension в `php.ini` или в скрипте, использованием функции dl().

```
./configure --with-apache=/path/to/apache_source --with-pgsql
```

Будет создана библиотека `libmodphp5.a`. Файл `mod_php5.c` и несколько сопровождающих файлов будут скопированы в каталог*src/modules/php5* дерева исходных кодов Apache. Затем следует скомпилировать Apache, используя *--activate-module=src/modules/php5/libphp5.a*, система сборки Apache создаст `libphp5.a` и статически свяжет с исполняемым `httpd`. Поддержка PostgreSQL будет включена непосредственно в `httpd` и конечным результатом будет единственный исполняемый файл `httpd`, включающий все от Apache и все от PHP.

```
./configure --with-apache=/path/to/apache_source --with-
pgsql=shared
```

Так же, как и прежде, взамен поддержки PostgreSQL, встроенной непосредственно в конечный исполняемый файл `httpd`, вы получите библиотеку `pgsql.so`, которую должны будете загрузить в PHP одним из двух способов: `php.ini` или используя функцию dl().

Выбирая способ установки PHP, следует учитывать преимущества и недостатки каждого из методов. Если вы собираете PHP, как подгружаемый модуль Apache - вы сможете компилировать PHP и Apache отдельно друг от друга и вам не нужно будет перекомпилировать Apache, если вы захотите изменить конфигурацию PHP. Если вы собираете PHP статически - PHP будет работать чуть быстрее. Для получения более подробной информации посетите страницу посвященную поддержке » DSO (Dynamic Shared Object) в Apache.

Замечание:
В настоящее время файл конфигурации Apache `httpd.conf` обычно поставляется со следующими настройками:

```
User nobody
Group "#-1"
```

Пока вы не измените группу на "Group nogroup" или что-нибудь вроде "Group daemon" (такая настройка довольно распространена), PHP не сможет открывать файлы.

Замечание:
Убедитесь, что вы указываете установленную версию apxs, когда используете --with-apxs=/path/to/apxs . Вы НЕ ДОЛЖНЫ указывать версию apxs, поставляемую с исходными кодами Apache - только действительно установленную в вашу систему вместе с сервером.

Apache 2.x на Unix системах ¶

Этот раздел описывает установку PHP с Apache 2.x на Unix системах.

Внимание
Мы не рекомендуем использовать потоковый MPM в промышленной среде вместе с Apache 2. Вместо этого, используйте prefork MPM, используемый по умолчанию в Apache 2.0 и 2.2. Подробную информацию по этому вопросу вы можете найти в соответствующем разделе FAQ Apache2 и потоковый MPM

Самым авторитетным источником информации по Apache 2.x является » документация Apache. Более подробная информация о настройках при установке может быть найдена там.

Самая последняя версия Apache Http Server может быть получена на » странице загрузки Apache, а адрес соответствующей версии PHP был указан выше. Это краткое руководство описывает лишь базовую установку Apache 2.x и PHP. Для получения более детальной информации прочитайте » документацию Apache. В инструкции ниже опущены номера версий - замените 'NN' на номер, соответствующий скачанной вами версии Apache.

На данный момент есть две версии Apache 2.x - 2.0 и 2.2. Хотя для выбора каждой из них существуют отдельные доводы, 2.2 является наиболее свежей и рекомендуемой версией, если вас устраивает такой выбор. Тем не менее, данные инструкции будут работать как для 2.0, так и для 2.2.

1. Скачайте Apache HTTP server как было указано выше и распакуйте его:

   ```
   gzip -d httpd-2_x_NN.tar.gz
   tar -xf httpd-2_x_NN.tar
   ```

2. Аналогично, скачайте и распакуйте исходные коды PHP:

   ```
   gunzip php-NN.tar.gz
   tar -xf php-NN.tar
   ```

3. Скомпилируйте и установите Apache. Более подробную информацию по сборке Apache смотрите в его документации.

```
cd httpd-2_x_NN
./configure --enable-so
make
make install
```

4. Теперь ваш Apache 2.x.NN доступен как /usr/local/apache2, сконфигурирован с поддержкой подгружаемых модулей и стандартным мульти-процессным модулем (MPM) prefork. Чтобы протестировать правильность установки используйте стандартную процедуру запуска Apache, такую как:

   ```
   /usr/local/apache2/bin/apachectl start
   ```

   Затем остановите сервер, чтобы сконфигурировать и установить PHP:

   ```
   /usr/local/apache2/bin/apachectl stop
   ```

5. Теперь мы сконфигурируем и соберем PHP. Здесь вы можете настроить установку PHP с помощью различных опций, указывающих, например, какие расширения нужно включить. Просмотрите вывод команды ./configure --help для получения полного списка параметров конфигурации. В нашем примере мы сконфигурируем PHP очень просто - с поддержкой Apache и MySQL.

   Если вы собирали Apache из исходников, как было описано выше, то используйте путь до apxs как указано в следующем примере, иначе, корректируйте этот путь соответствующим вашей установке образом. Учтите также, что в некоторых дистрибутивах apxs может иметь имя apxs2.

   ```
   cd ../php-NN
   ./configure --with-apxs2=/usr/local/apache2/bin/apxs --
with-mysql
   make
   make install
   ```

   Если вы решите изменить параметры конфигурации после установки, вам надо будет повторить только три последних шага (configure, make, make install). Вам нужно будет только перезапустить Apache, чтобы новые модули подгрузились и начали работать. Перекомпиляция Apache для этого не требуется.

   Заметьте, что если не указано обратное, то 'make install' установит также PEAR, различные инструменты PHP - такие как phpize, версию PHP для командной строки (PHP CLI) и т.д.

6. Настройка вашего php.ini

   ```
   cp php.ini-development /usr/local/lib/php.ini
   ```

   Вероятно, вы захотите изменить некоторые настройки в php.ini. Если вы предпочитаете держать файл php.ini в другом месте, используйте параметр --with-config-file-path=/some/path в шаге 5.

Если же вы используетеphp.ini-production, прочитайте его, чтобы знать какие изменения в поведении PHP это повлечет.

7. Отредактируйте ваш httpd.conf, чтобы Apache загружал модуль PHP. Путь в правой части инструкции LoadModule должен указывать на модуль PHP. Команда make install может добавить эту инструкцию автоматически, но этого может и не произойти, поэтому проверьте, чтобы убедиться.

```
LoadModule php5_module modules/libphp5.so
```

8. Теперь следует сконфигурировать Apache, чтобы он передавал файлы с некоторыми расширениями на обработку модулю PHP. В нашем примере сделаем это для .php файлов. Вместо обычного использования директивы Apache AddType, мы хотим избежать интерпретации как PHP потенциально опасных загрузок и файлов наподобие exploit.php.jpg. С помощью данного примера можно указать для интерпретации PHP любые расширения, просто добавив их в конец списка. Продемонстрируем это на расширении .php.

```
<FilesMatch \.php$>
  SetHandler application/x-httpd-php
</FilesMatch>
```

Или, если мы хотим добавить расширения .php, .php2, .php3, .php4, .php5, .php6 и .phtml, это можно записать так:

```
<FilesMatch "\.ph(p[2-6]?|tml)$">
   SetHandler application/x-httpd-php
</FilesMatch>
```

Чтобы PHP отображал содержимое файлов .phps с подсветкой синтаксиса, нужно внести соответствующую директиву

```
<FilesMatch "\.phps$">
    SetHandler application/x-httpd-php-source
</FilesMatch>
```

Можно использовать mod_rewrite для отображения любого .php файла с подсветкой синтаксиса, без нужды его переименования в .phps:

```
RewriteEngine On
RewriteRule   (.*\.php)s$   $1   [H=application/x-httpd-php-source]
```

Данный фильтр, отображающий исходный код, должен быть отключен в промышленном использовании, так как он может открыть конфиденциальную или другую важную информацию, включенную в исходный код.

9. Используйте стандартную процедуру запуска Apache, например:

```
/usr/local/apache2/bin/apachectl start
```

ИЛИ

```
service httpd restart
```

Если вы следовали инструкциям выше, то на данном этапе должны иметь запущенный веб-сервер Apache2 с поддержкой PHP, как модуля *SAPI*. Конечно, для PHP и Apache доступно гораздо больше параметров конфигурации. Используйте ./configure --help в соответствующей папке с исходными кодами для получения полного списка параметров конфигурации.

Если вы хотите собрать многопоточную версию Apache, то при сборке вам следует указать модуль MPM `worker` вместо стандартного модуля MPM `prefork`. Чтобы сделать это, нужно добавить следующий аргумент к ./configure на шаге 3:

```
--with-mpm=worker
```

Не следует бездумно использовать эту настройку, используйте ее только если вы понимаете все последствия этого решения. Документация Apache по » модулям MPM значительно глубже раскрывает эту тему.

Замечание:
Если вы хотите использовать зависимый контент (content negotiation), прочитайте Apache MultiViews FAQ.
Замечание:
Для сборки многопоточной версии Apache, ваша система должна поддерживать потоки. Это также подразумевает сборку PHP с экспериментальной поддержкой Zend Thread Safety (ZTS). И, как следствие, не все расширения PHP смогут работать. Рекомендуется установка Apache с модулем MPM по умолчанию - `prefork`.

User Contributed Notes *15 notes*

Установка с интерфейсами CGI и командной строки ¶

По умолчанию, PHP собирается одновременно как CLI и CGI программа, которая может быть использована для обработки CGI-запросов. PHP как модуль сервера выигрывает в производительности, однако PHP CGI позволяет запускать PHP от пользователя, отличного от того, под которым исполняется сервер.

Внимание
Используя установку CGI, ваш сервер открыт перед несколькими возможными уязвимостями. Пожалуйста, ознакомьтесь с разделом "Безопасность CGI" чтобы узнать, как можно защитить себя от таких атак.

Тестирование ¶

Если вы собрали PHP как CGI, вы можете протестировать вашу сборку командой make test. Тестирование вашей сборки - всегда хорошая идея. Таким образом вы сможете раньше обнаружить проблемы PHP на вашей платформе, вместо того, чтобы бороться с ними позже.

Использование переменных ¶

Некоторые переменные окружения сервера не определены в текущей » спецификации CGI/1.1. Определены только следующие переменные: `AUTH_TYPE`, `CONTENT_LENGTH`, `CONTENT_TYPE`, `GATEWAY_INTERFACE`,

`PATH_INFO`, `PATH_TRANSLATED`, `QUERY_STRING`,`REMOTE_ADDR`, `REMOTE_HOST`, `REMOTE_IDENT`, `REMOTE_USER`, `REQUEST_METHOD`, `SCRIPT_NAME`, `SERVER_NAME`, `SERVER_PORT`, `SERVER_PROTOCOL`, and `SERVER_SOFTWARE`. Все остальное должно обрабатываться как дополнительные расширения (vendor extensions).

User Contributed Notes 6 notes

LINUX : Debian GNU/Linux installation notes ¶

This section contains notes and hints specific to installing PHP on » Debian GNU/Linux.

Внимание
Unofficial builds from third-parties are not supported here. Any bugs should be reported to the Debian team unless they can be reproduced using the latest builds from our » download area.

While the instructions for building PHP on Unix apply to Debian as well, this manual page contains specific information for other options, such as using either the *apt-get* or *aptitude* commands. This manual page uses these two commands interchangeably.

Using APT ¶

First, note that other related packages may be desired like *libapache2-mod-php5* to integrate with Apache 2, and *php-pear* for PEAR.

Second, before installing a package, it's wise to ensure the package list is up to date. Typically, this is done by running the command apt-get update.

Пример #1 Debian Install Example with Apache 2

```
# apt-get install php5-common libapache2-mod-php5 php5-cli
```

APT will automatically install the PHP 5 module for Apache 2 and all of its dependencies, and then activate it. Apache should be restarted in order for the changes take place. For example:

Пример #2 Stopping and starting Apache once PHP is installed

```
# /etc/init.d/apache2 stop
# /etc/init.d/apache2 start
```

Better control of configuration ¶

In the last section, PHP was installed with only core modules. It's very likely that additional modules will be desired, such as MySQL, cURL, GD, etc. These may also be installed via the *apt-get* command.

Пример #3 Methods for listing additional PHP 5 packages

```
# apt-cache search php5
# aptitude search php5
```

```
# aptitude search php5 |grep -i mysql
```

The examples will show a lot of packages including several PHP specific ones like php5-cgi, php5-cli and php5-dev. Determine which are needed and install them like any other with either *apt-get* or *aptitude*. And because Debian performs dependency checks, it'll prompt for those so for example to install MySQL and cURL:

Пример #4 Install PHP with MySQL, cURL

```
# apt-get install php5-mysql php5-curl
```

APT will automatically add the appropriate lines to the different `php.ini` related files like `/etc/php5/apache2/php.ini`,`/etc/php5/conf.d/pdo.ini`, etc. and depending on the extension will add entries similar to *extension=foo.so*. However, restarting the web server (like Apache) is required before these changes take affect.

Common Problems ¶

- If the PHP scripts are not parsing via the web server, then it's likely that PHP was not added to the web server's configuration file, which on Debian may be `/etc/apache2/apache2.conf` or similar. See the Debian manual for further details.
- If an extension was seemingly installed yet the functions are undefined, be sure that the appropriate ini file is being loaded and/or the web server was restarted after installation.
- There are two basic commands for installing packages on Debian (and other linux variants): *apt-get* and *aptitude*. However, explaining the subtle differences between these commands goes beyond the scope of this manual.

User Contributed Notes *5 notes*

Установка в системах Windows ¶

Содержание ¶

Установка PHP в современных операционных системах Microsoft Windows и рекомендуемая конфигурация с распространенными веб серверами.

Замечание:
Если вы ищете информацию о старых версиях операционных систем: Windows XP, 2003, 98 or Apache 1.x, обратитесь к разделу руководства Установка на старых версиях ОС Windows.

Официальные релизы PHP для Windows рекомендованы для продуктивного использования. Однако вы также можете собрать PHP из исходных кодов. Вам по-

требуется окружение Visual Studio. Обратитесь к разделу » Пошаговое руководство по сборке для получения более полной информации.

Использование PHP в командной строке Windows Установка PHP на Azure App Services (aka Microsoft Azure, Windows Azure, или (Windows) Azure Web Apps).

User Contributed Notes 58 notes

PHP Installer Tools on Windows ¶

PHP Install Tools

If you want to setup PHP, some common PHP applications and are using IIS, the easiest way is to use Microsoft's Web Platform Installer (WebPI).

XAMPP, WampServer and BitNami will setup PHP applications for use with Apache on Windows.

Recommended Configuration on Windows systems ¶

OpCache

Highly Recommended that you enable OpCache. This extension is included with PHP for Windows. It compiles and optimizes PHP scripts and caches them in memory so that they aren't compiled every time the page is loaded.

In your php.ini, set

Пример #1 Recommended WinCache configuration

```
zend_extension=php_opcache.dll
opcache.enable=On
opcache.cli_enable=On
```

And restart your web server. For more info, see: OpCache Configuration

WinCache

Recommended that you use WinCache if using IIS, especially if in a shared web hosting environment or using networked file storage (NAS). All PHP Applications automatically benefit from WinCache's file cache feature. File system operations are cached in memory. WinCache also can cache user objects in memory and share them between `php.exe` or `php-cgi.exe` processes (share objects between requests). Many major web applications have a plugin or extension or configuration option to make use of the WinCache user object cache. If you need high performance, you should use the object cache in your applications. See: » http://pecl.php.net/package/WinCache to download a WinCache DLL (or tgz) to your PHP extensions directory (extensions_dir in your php.ini). In your php.ini, set

Пример #2 Recommended WinCache configuration

```
extension=php_wincache.dll
wincache.fcenabled=1
```

```
wincache.ocenabled=1
```

For more info, see: » http://php.net/manual/en/wincache.configuration.php

IIS Configuration

In IIS Manager, Install FastCGI module and add a handler mapping for `.php` to the path to `PHP-CGI.exe` (not `PHP.exe`)

You may use the APPCMD command line tool to script IIS configuration.

Database

You'll probably need a Database Server. Popular databases provide PHP extensions to use them. If your web site doesn't get a lot of traffic, you can run your database server on the same server as your web server. Many popular database servers run on Windows.

PHP includes mysqli and pdo_mysql extensions. PHP 5.5 and 5.6 include mysql extension (deprecated in 7.0).

See » https://dev.mysql.com/downloads/windows/

Manual PHP Installation on Windows ¶

Choose Web Server

- IIS is builtin to Windows. On Windows Server, use Server Manager to add the IIS role. Be sure to include the CGI Role Feature. On Windows Desktop, use Control Panel's Add/Remove Programs to add IIS. See: » https://msdn.microsoft.com/en-us/library/ms181052%28v=vs.80%29.aspx?f=255&MSPPError=-2147217396 For desktop web apps and web-development, you can also use IIS/Express or PHP Desktop

  Пример #1 Command line to configure IIS and PHP

```
    @echo off

    REM download .ZIP file of PHP build from
http://windows.php.net/downloads/
    REM
    REM path to directory you decompressed PHP .ZIP file
into
set phpdir=c:\php
set phppath=php-5.6.19-nts-Win32-VC11-x86

REM Clear current PHP handlers
%windir%\system32\inetsrv\appcmd clear config
/section:system.webServer/fastCGI
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/handlers /-
[name='PHP_via_FastCGI']
```

```
REM Set up the PHP handler
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/fastCGI
/+[fullPath='%phpdir%\%phppath%\php-cgi.exe']
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/handlers
/+[name='PHP_via_FastCGI',path='*.php',verb='*',modules='Fa
stCgiModule',scriptProcessor='%phpdir%\%phppath%\php-
cgi.exe',resourceType='Unspecified']
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/handlers
/accessPolicy:Read,Script

REM Configure FastCGI Variables
%windir%\system32\inetsrv\appcmd set config -
section:system.webServer/fastCgi
/[fullPath='%phpdir%\%phppath%\php-
cgi.exe'].instanceMaxRequests:10000
%windir%\system32\inetsrv\appcmd.exe set config -
section:system.webServer/fastCgi
/+"[fullPath='%phpdir%\%phppath%\php-
cgi.exe'].environmentVariables.[name='PHP_FCGI_MAX_REQUESTS
',value='10000']"
%windir%\system32\inetsrv\appcmd.exe set config -
section:system.webServer/fastCgi
/+"[fullPath='%phpdir%\%phppath%\php-
cgi.exe'].environmentVariables.[name='PHPRC',value='%phpdir
%\%phppath%\php.ini']"
```

How to manually configure IIS

- There are several builds of Apache2 for Windows. We support ApacheLounge, but other options include XAMPP, WampServer and BitNami, which provide automatic installer tools. You may use mod_php or mod_fastcgi to load PHP on Apache. If you use mod_php, you MUST use a TS build of Apache built with same version of Visual C and same CPU (x86 or x64). How to manually configure Apache2

Choose Build

Download PHP production releases from » http://windows.php.net/downloads/release/. A lot of testing and optimization is already done on the snapshot and qa releases, but you are welcome to help us do more. There are 4 types of PHP builds:

- Thread-Safe(TS) - use for single process web serves, like Apache with mod_php
- Non-Thread-Safe(NTS) - use for IIS and other FastCGI web servers (Apache with mod_fastcgi) and recommended for command-line scripts
- x86 - production use of PHP 5.5 or 5.6 or 7.0.
- x64 - production use of PHP 7.0+ unless its a 32-bit only version of Windows. 5.5 and 5.6 x64 are expiremental.

User Contributed Notes 37 notes

Troubleshooting PHP on Windows ¶

Check Temp Directory Permissions

1. Right-click temp directory in File Explorer to get the permissions.
2. For IIS, check that user IIS_User has MODIFY permission. You can get the temporary directory from the configuration or php info.

Установка на старых версиях ОС Windows ¶

Данный раздел руководства применим к Windows 98/Me и Windows NT/2000/XP/2003. PHP не будет работать на 16 битных платформах, таких как Windows 3.1 и иногда мы ссылаемся на поддерживаемые Windows платформы как Win32.

Замечание:
Windows XP/2003 больше не поддерживаются для PHP 5.5.0.
Замечание:
Windows 98/Me/NT4/2000 больше не поддерживаются для PHP 5.3.0.
Замечание:
Windows 95 больше не поддерживается для PHP 4.3.0.

Если у вас есть окружение для разработки, такое как Microsoft Visual Studio, вы также можете собрать PHP из исходников.

Установив PHP в ОС Windows, вы также можете захотеть загрузить различные расширения для обеспечения дополнительной функциональности.

Руководство по ручной установке ¶

Этот раздел содержит инструкции для ручной установки и настройки PHP на Microsoft Windows.

Выбор и загрузка пакета дистрибутивов PHP ¶

Загрузите дистрибутив PHP в виде zip-архива с » PHP для Windows: Исполняемые файлы и исходные коды. Существует несколько различных версий zip-пакетов - выберите версию, которая подходит для используемого веб сервера:

Структура и содержание пакета PHP ¶

Распакуйте содержимое zip архива в директорию на ваш выбор, например C:\PHP\. Структура папок и файлов, извлеченных из архива, будет выглядеть следующим образом:

Пример #1 Структура пакета PHP 5

```
c:\php
    |
    +--dev
```

```
   |  |
   |  |-php5ts.lib                    -- версия php5.lib без под-
держки многопоточности
   |
   +--ext                           -- DLL расширения для PHP
   |  |
   |  |-php_bz2.dll
   |  |
   |  |-php_cpdf.dll
   |  |
   |  |-...
   |
   +--extras                        -- пустой
   |
   +--pear                          -- начальная копия PEAR
   |
   |
   |-go-pear.bat                    -- скрипт установки PEAR
   |
   |-...
   |
   |-php-cgi.exe                    -- исполняемый файл CGI
   |
   |-php-win.exe                     -- выполняет скрипты без от-
крытой консоли
   |
   |-php.exe                          -- Исполняемый файл PHP для
командной строки (CLI)
   |
   |-...
   |
   |-php.ini-development            -- настройки php.ini по умол-
чанию
   |
   |-php.ini-production               -- рекомендуемые настройки
php.ini
   |
   |-php5apache2_2.dll               -- имеется только в многопо-
точной версии
   |
   |-php5apache2_2_filter.dll       -- имеется только в многопо-
точной версии
   |
   |-...
   |
   |-php5ts.dll                      -- ядро PHP DLL ( php5.dll в
версии без поддержки многопоточности)
   |
   |-...
```

Ниже представлен список модулей и исполняемых файлов, включенных в PHP zip дистрибутив:

- `go-pear.bat` - скрипт установки PEAR. Подробнее см. » Установка (PEAR).
- `php-cgi.exe` - исполняемый файл CGI, который может быть использован во время запуска PHP на IIS через CGI или FastCGI.
- `php-win.exe` - исполняемый файл PHP для выполнения PHP скриптов без использования консоли (например, приложения PHP, использующие Windows GUI).
- `php.exe` - исполняемый файл PHP для выполнения PHP скриптов в консоли (CLI).
- `php5apache2_2.dll` - модуль Apache 2.2.X.
- `php5apache2_2_filter.dll` - фильтр Apache 2.2.X.

Изменение файла php.ini ¶

После того, как содержимое пакета php извлечено, создайте копию `php.ini-production` с именем `php.ini` в той же папке. Если необходимо, также возможно разместить `php.ini` в любом другом месте по вашему выбору, но это потребует дополнительной настройки, которая приводится в разделе Настройка PHP.

Файл `php.ini` содержит правила исполнения PHP и инструкции по работе с окружением, в котором он запускается. Ниже приводятся некоторые из настроек `php.ini`, которые могут улучшить работу PHP в Windows. Некоторые из них опциональны. Есть много других директив, которые могут быть полезны в вашем окружении - обращайтесь к списку директив php.ini за более подробной информацией.

Обязательные директивы:

- `extension_dir` = *<путь к директории расширений>* - `extension_dir` указывает директорию, где расположены расширения PHP. Путь может быть абсолютным (например "C:\PHP\ext") или относительным (например ".\ext"). Используемые в `php.ini` расширения должны быть расположены в `extension_dir`.
- `extension` = *xxxxx.dll* - Для каждого подключаемого расширения необходимо указать директиву "extension=". Расширения из`extension_dir`, отмеченные такой директивой, загружаются при старте PHP.
- `log_errors` = *On* - в PHP есть механизм ведения лога ошибок, который может использоваться для сохранения ошибок в файле или для отправки в сервис (например syslog). Механизм также использует значение директивы `error_log`. Когда PHP исполняется службой IIS, `log_errors` должен быть включен с корректным `error_log`.
- `error_log` = *<пусть к файлу лога ошибок>* - error_log нужен для обозначения абсолютного или относительного пути к файлу, в который протоколируются ошибки PHP. Этот файл должен доступным для записи веб-сервером. Самые распространенные места размещения этого файла - различные временные TEMP директории, например "C:\inetpub\temp\php-errors.log".
- `cgi.force_redirect` = *0* - Эта директива необходима для исполнения под IIS. Это механизм защиты директории, требуемый многими другими веб серверами. Однако включение его под IIS вызовет ошибки ядра PHP в Windows.

- `cgi.fix_pathinfo` = *1* - Обеспечивает поддержку PATH_INFO согласно спецификации CGI. IIS FastCGI использует эту настройку.
- `fastcgi.impersonate` = *1* - FastCGI под IIS поддерживает способность идентифицировать маркеры безопасности вызывающего клиента. Это позволяет IIS определять контекст безопасности, под которые выполняется запрос.
- `fastcgi.logging` = *0* - Запись логов FastCGI должна быть выключена в IIS. Если запись включена, тогда все сообщения любых классов распознаются FastCGI как ошибки, что приведет IIS к генерации исключения HTTP 500.

Опциональные директивы

- `max_execution_time` = ## - Эта директива указывает максимальное время выполнения любого скрипта PHP. По умолчанию равно 30 секундам. Следует увеличить это значение, если приложение PHP должно выполняться дольше.
- `memory_limit` = *###M* - Количество памяти, доступное процессу PHP, в Мб. По умолчанию 128, что достаточно для большинства PHP приложений. Некоторым сложным приложениям может потребоваться больше памяти.
- `display_errors` = *Off* - Директива определяет, какие ошибки следует возвращать веб-серверу для дальнейшего протоколирования. При значении "On" PHP сообщает обо всех видах ошибок, которые приводятся в директиве `error_reporting`. По соображениям безопасности рекомендуется установить в "Off" на рабочих серверах, чтобы исключить передачу вывода ошибок конечному пользователю, так как они могут содержат информацию, угрожающую безопасности приложения.
- `open_basedir` = *<пути к директориям, разделенные точкой с запятой>*, например openbasedir="C:\inetpub\wwwroot;C:\inetpub\temp". Эта директива указывает пути к директориям, в которых PHP разрешены операции с файловой системой. Любая операция с файлами и директориями вне указанных путей будет приводить к ошибке. Эта директива особенно полезна для предотвращения доступа к установленному PHP в окружениях разделяемых хостингов для предотвращения доступа PHP скриптов к любым файлам вне корневой директории веб сайта.
- `upload_max_filesize` = *###M* и `post_max_size` = *###M* - Максимальный разрешенный размер загруженного файла и присланных данных соответственно. Значения этих директив должны быть увеличены, если приложения PHP должны обрабатывать большие загружаемые файлы, например изображения или видеофайлы.

После установки PHP в вашей системе, следующим шагом будет выбор веб-сервера и его дальнейшая настройка для работы с PHP. Выберите конкретный веб-сервер в оглавлении к данному материалу.

Помимо запуска PHP с помощью веб-сервера, PHP может быть запущен из командной строки как *.BAT* скрипт. За более подробной информацией обращайтесь к материалу Консоль PHP на Microsoft Windows.

Microsoft IIS ¶

This section contains PHP installation instructions specific to Microsoft Internet Information Services (IIS).

- Manually installing PHP on Microsoft IIS 5.1 and IIS 6.0
- Manually installing PHP on Microsoft IIS 7.0 and later

Microsoft IIS 5.1 and IIS 6.0 ¶

This section contains instructions for manually setting up Internet Information Services (IIS) 5.1 and IIS 6.0 to work with PHP on Microsoft Windows XP and Windows Server 2003. For instructions on setting up IIS 7.0 and later versions on Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2 refer to Microsoft IIS 7.0 and later.

Configuring IIS to process PHP requests ¶

Download and install PHP in accordance to the instructions described in manual installation steps

Замечание:
Non-thread-safe build of PHP is recommended when using IIS. The non-thread-safe builds are available at » PHP for Windows: Binaries and Sources Releases.

Configure the CGI- and FastCGI-specific settings in `php.ini` file as shown below:

Пример #2 CGI and FastCGI settings in *php.ini*

```
fastcgi.impersonate = 1
fastcgi.logging = 0
cgi.fix_pathinfo=1
cgi.force_redirect = 0
```

Download and install the » Microsoft FastCGI Extension for IIS 5.1 and 6.0. The extension is available for 32-bit and 64-bit platforms - select the right download package for your platform.

Configure the FastCGI extension to handle PHP-specific requests by running the command shown below. Replace the value of the "-path" parameter with the absolute file path to the `php-cgi.exe` file.

Пример #3 Configuring FastCGI extension to handle PHP requests

```
cscript %windir%\system32\inetsrv\fcgiconfig.js -add -
section:"PHP" ^
-extension:php -path:"C:\PHP\php-cgi.exe"
```

This command will create an IIS script mapping for *.php file extension, which will result in all URLs that end with .php being handled by FastCGI extension. Also, it will configure FastCGI extension to use the executable `php-cgi.exe` to process the PHP requests.

Замечание:
At this point the required installation and configuration steps are completed. The remaining instructions below are optional but highly recommended for achieving optimal functionality and performance of PHP on IIS.

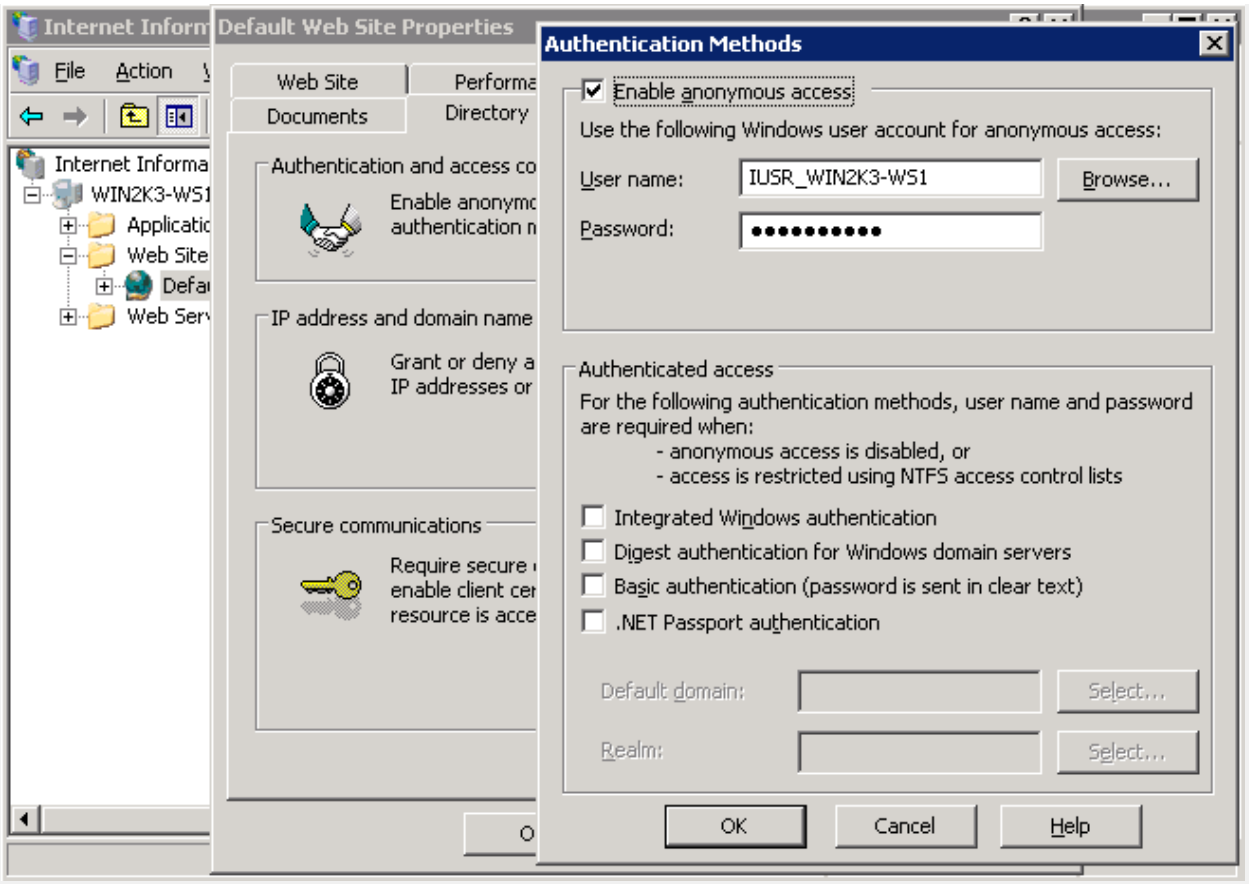
Impersonation and file system access ¶

It is recommended to enable FastCGI impersonation in PHP when using IIS. This is controlled by the `fastcgi.impersonate` directive in `php.ini` file. When impersonation is enabled, PHP will perform all the file system operations on behalf of the user account that has been determined by IIS authentication. This ensures that even if the same PHP process is shared across different IIS web sites, the PHP scripts in those web sites will not be able to access each others' files as long as different user accounts are used for IIS authentication on each web site.

For example IIS 5.1 and IIS 6.0, in its default configuration, has anonymous authentication enabled with built-in user account IUSR_<MACHINE_NAME> used as a default identity. This means that in order for IIS to execute PHP scripts, it is necessary to grant IUSR_<MACHINE_NAME> account read permission on those scripts. If PHP applications need to perform write operations on certain files or write files into some folders then IUSR_<MACHINE_NAME> account should have write permission to those.

To determine which user account is used by IIS anonymous authentication, follow these steps:

1. In the Windows Start Menu choose "Run:", type "inetmgr" and click "Ok";
2. Expand the list of web sites under the "Web Sites" node in the tree view, right-click on a web site that is being used and select "Properties";
3. Click the "Directory Security" tab;
4. Take note of a "User name:" field in the "Authentication Methods" dialog



To modify the permissions settings on files and folders, use the Windows Explorer user interface or `icacls` command.

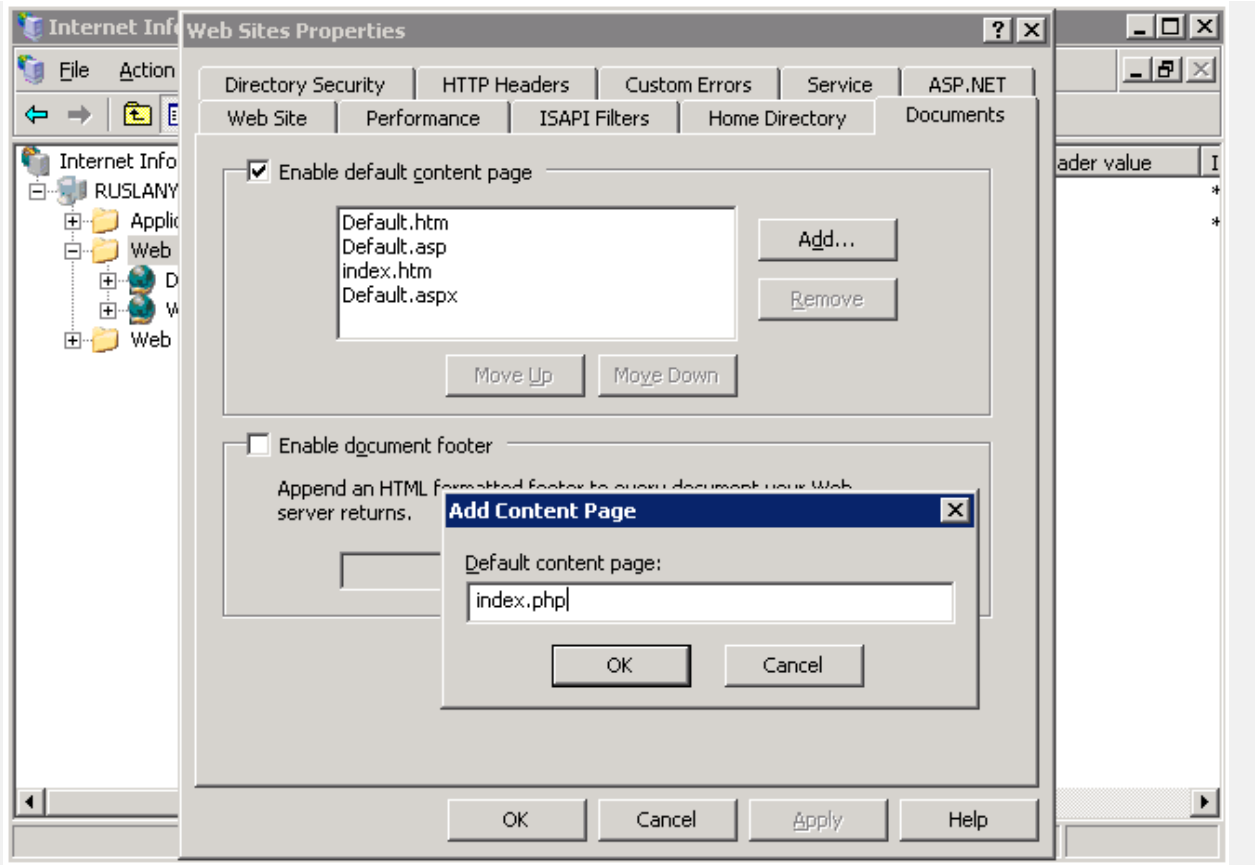
Пример #4 Configuring file access permissions

```
icacls C:\inetpub\wwwroot\upload /grant IUSR:(OI)(CI)(M)
```

Set ***index.php*** as a default document in IIS ¶

The IIS default documents are used for HTTP requests that do not specify a document name. With PHP applications, `index.php` usually acts as a default document. To add `index.php` to the list of IIS default documents, follow these steps:

1. In the Windows Start Menu choose "Run:", type "inetmgr" and click "Ok";
2. Right-click on the "Web Sites" node in the tree view and select "Properties";
3. Click the "Documents" tab;
4. Click the "Add..." button and enter "index.php" for the "Default content page:".



FastCGI and PHP Recycling configuration ¶

Configure IIS FastCGI extension settings for recycling of PHP processes by using the commands shown below. The FastCGI setting`instanceMaxRequests` controls how many requests will be processed by a single `php-cgi.exe` process before FastCGI extension shuts it down. The PHP environment variable `PHP_FCGI_MAX_REQUESTS` controls how many requests a single `php-cgi.exe` process will handle before it recycles itself. Make sure that the value specified for FastCGI `InstanceMaxRequests` setting is less than or equal to the value specified for`PHP_FCGI_MAX_REQUESTS`.

Пример #5 Configuring FastCGI and PHP recycling

```
cscript %windir%\system32\inetsrv\fcgiconfig.js -set -
section:"PHP" ^
-InstanceMaxRequests:10000

cscript %windir%\system32\inetsrv\fcgiconfig.js -set -
section:"PHP" ^
-EnvironmentVars:PHP_FCGI_MAX_REQUESTS:10000
```

Configuring FastCGI timeout settings ¶

Increase the timeout settings for FastCGI extension if there are applications that have long running PHP scripts. The two settings that control timeouts are `ActivityTimeout` and `RequestTimeout`. Refer to » Configuring FastCGI Extension for IIS 6.0 for more information about those settings.

Пример #6 Configuring FastCGI timeout settings

```
cscript     %windir%\system32\inetsrv\fcgiconfig.js     -set     -
section:"PHP" ^
-ActivityTimeout:90

cscript     %windir%\system32\inetsrv\fcgiconfig.js     -set     -
section:"PHP" ^
-RequestTimeout:90
```

Changing the Location of *php.ini* file ¶

PHP searches for `php.ini` file in several locations and it is possible to change the default locations of `php.ini` file by using `PHPRC`environment variable. To instruct PHP to load the configuration file from a custom location run the command shown below. The absolute path to the directory with `php.ini` file should be specified as a value of `PHPRC` environment variable.

Пример #7 Changing the location of *php.ini* file

```
cscript     %windir%\system32\inetsrv\fcgiconfig.js     -set     -
section:"PHP" ^
-EnvironmentVars:PHPRC:"C:\Some\Directory\"
```

Microsoft IIS 7.0 и выше ¶

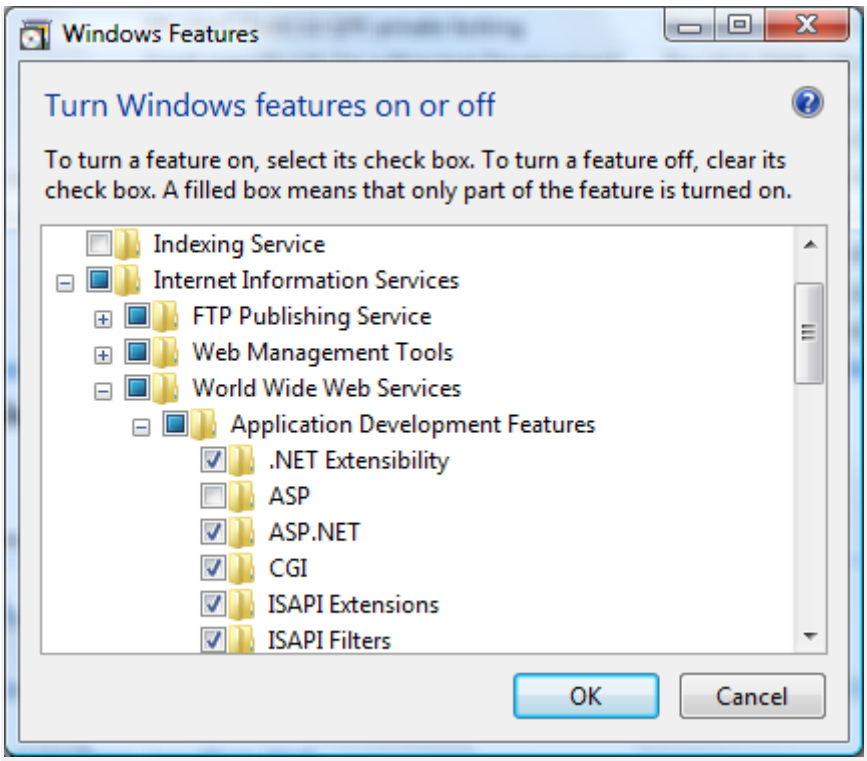
Этот раздел содержит инструкции по настройке Internet Information Services (IIS) 7.0 и более поздних версий для работы с PHP на Microsoft Windows Vista SP1, Windows 7, Windows Server 2008 и Windows Server 2008 R2. Для получения инструкций по настройке IIS 5.1 и IIS 6.0 на Windows XP и Windows Server 2003 перейдите на страницу Microsoft IIS 5.1 и IIS 6.0.

Включение поддержки FastCGI в IIS ¶

Модуль FastCGI при установке IIS по умолчанию отключён. Способы включения его различаются в зависимости от версии используемой Windows.

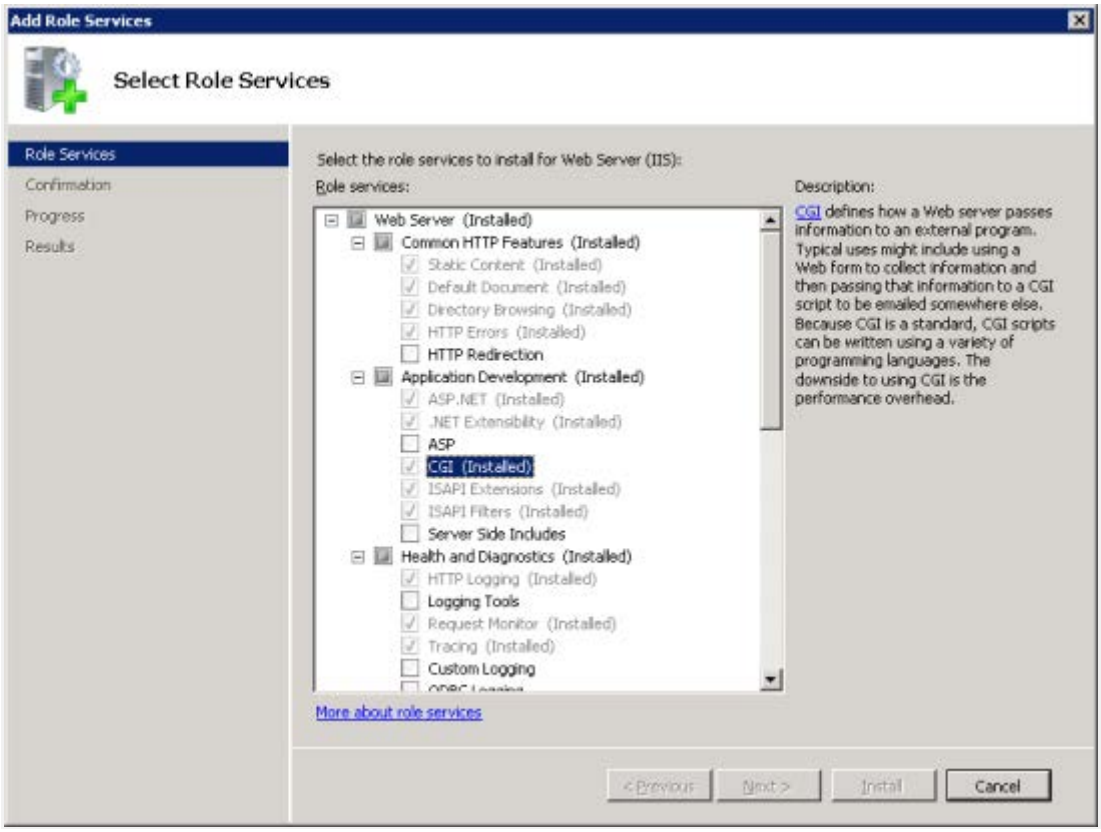
Для включения поддержки FastCGI на Windows Vista SP1 и Windows 7:

1. В меню "Пуск" выберите пункт "Выполнить", в появившемся окне введите с клавиатуры "optionalfeatures.exe" и нажмите "Ok";
2. В открывшемся окне "Компоненты Windows" раскройте папку "Службы IIS", "Службы интернета", "Компоненты разработки приложений" и установите галочку напротив "CGI";
3. Нажмите OK и ждите окончания процесса установки.



Чтобы включить поддержку FastCGI на Windows Server 2008 и Windows Server 2008 R2:

1. В Windows откройте меню Пуск выберите пункт "Выполнить:", наберите с клавиатуры "CompMgmtLauncher" и нажмите "Ok";
2. Если роль "Веб-сервер (IIS)" не представлена во вкладке "Роли", добавьте её, выбрав "Добавить роли";
3. Если роль "Веб-сервер (IIS)" присутствует, выберите "Выбор службы ролей" и установите галочку напротив "CGI" в группе "Компоненты разработки приложений";
4. Нажмите "Далее" затем "Установить" и ждите окончания процесса установки.



Настройка IIS для обработки PHP запросов ¶

Скачайте и установите PHP в соответствии с инструкциями, приведёнными в описании установки

Замечание:
При использовании IIS рекомендуется использовать потоко-небезопасную (Non-thread-safe) сборку PHP, которая доступна по ссылке » PHP для Windows: Установочные файлы и исходный код.

Измените CGI и FastCGI настройки в файле `php.ini` как показано ниже:

Пример #8 CGI и FastCGI настройки в *php.ini*

```
fastcgi.impersonate = 1
fastcgi.logging = 0
cgi.fix_pathinfo=1
cgi.force_redirect = 0
```

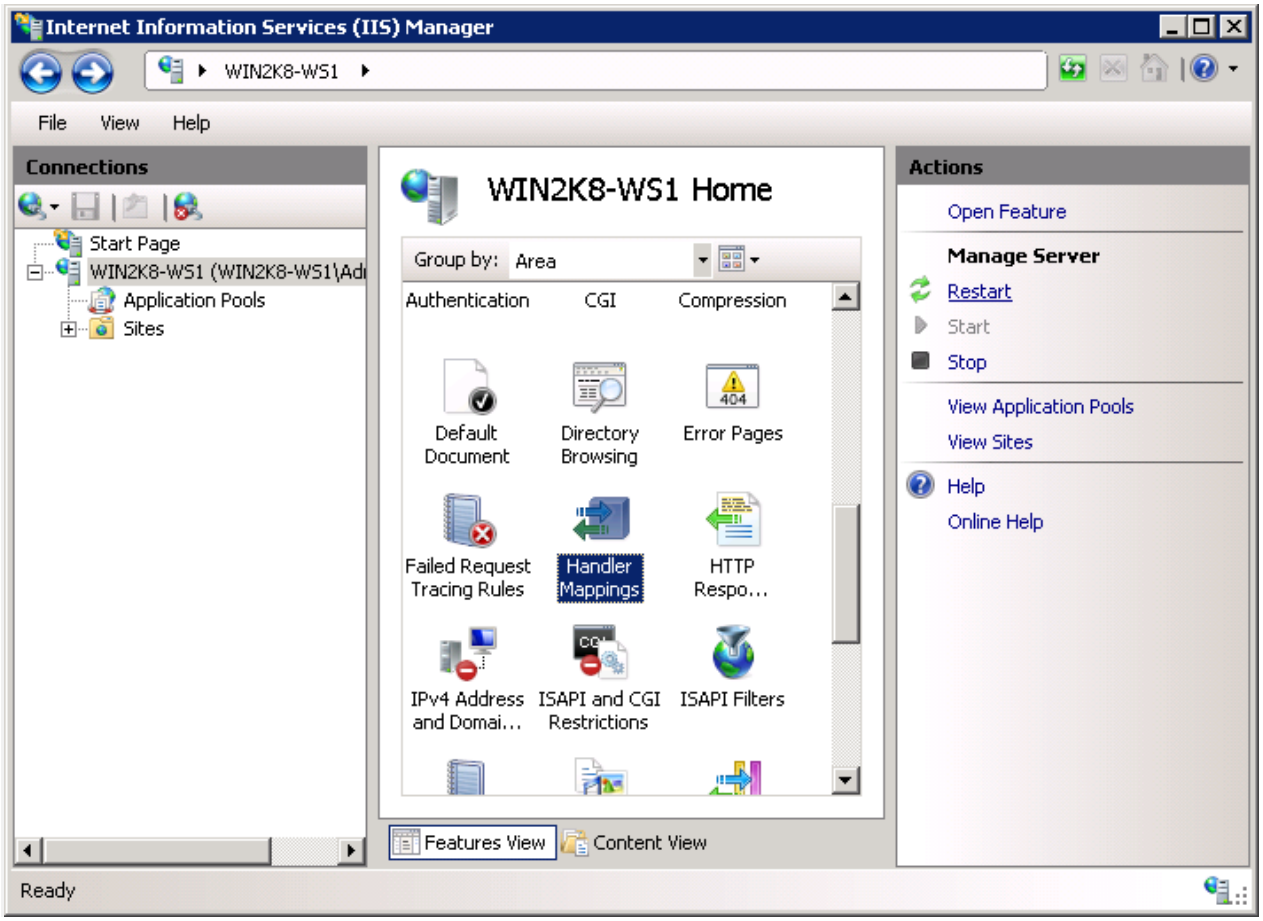
Настройте IIS обработчик для PHP, используя Интерфейс управления IIS или через командную строку.

Использование Интерфейса управления IIS для создания обработчика PHP

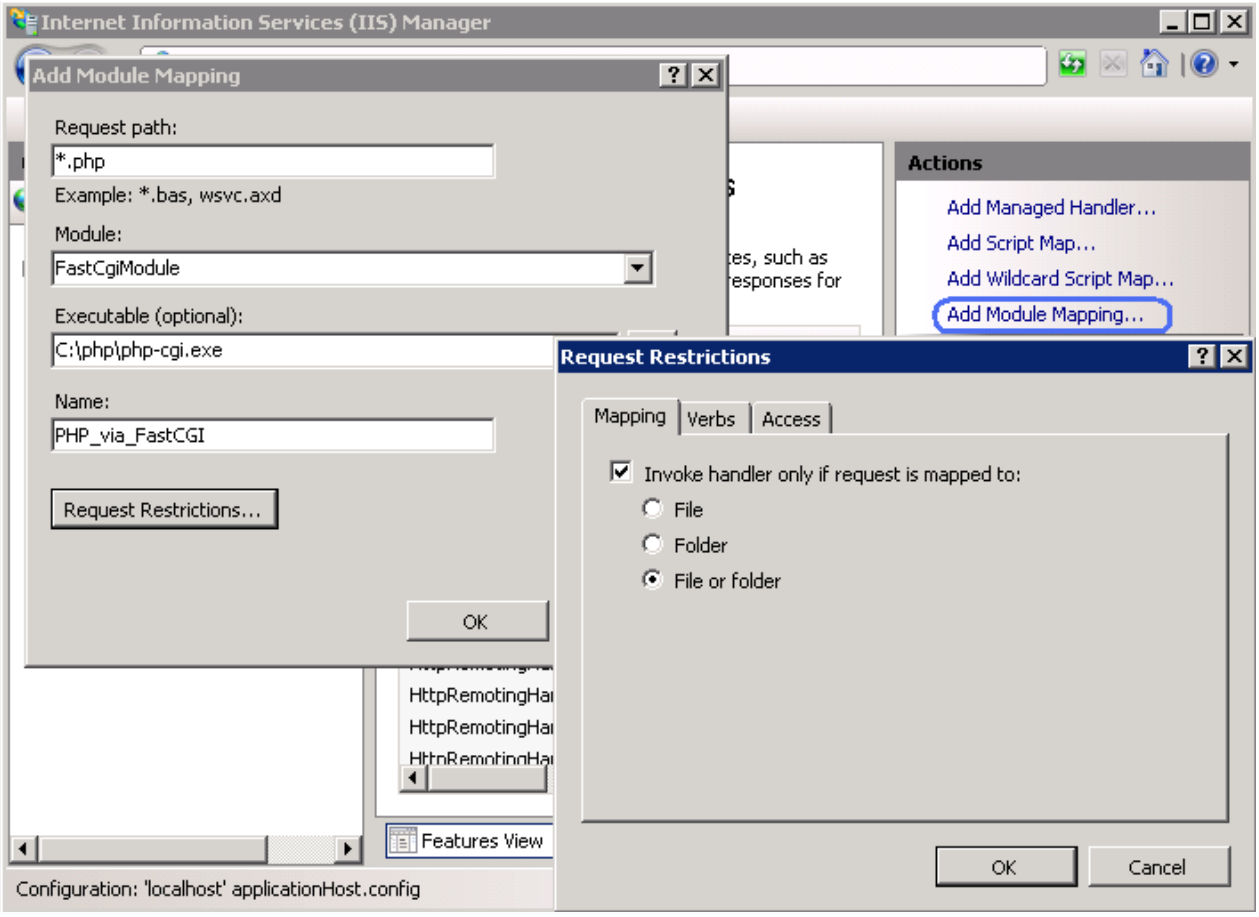
Следующие шаги позволят вам создать IIS обработчик для PHP в Интерфейсе управления IIS:

1. В Windows меню Пуск выберите команду "Выполнить:", введите в клавиатуры команду "inetmgr" и нажмите "Ok";

2. В Интерфейсе управления IIS выберите сервер в дереве "Подключения";
3. На "Начальной странице" откройте "Сопоставления обработчиков";



4. На вкладке "Действия" выберите "Добавить сопоставление модуля...";
5. В окне "Добавление сопоставления модуля" введите следующее:
   - Путь запроса: *.php
   - Модуль: FastCgiModule
   - Исполняемый файл: C:\[Path to PHP installation]\php-cgi.exe
   - Имя: PHP_via_FastCGI
6. Нажмите кнопку "Ограничения запроса" и сконфигурируйте сопоставление вызывать обработчик только при сопоставлении с файлом или каталогом;
7. Нажмите OK во всех диалогах для сохранения конфигурации.



Использование командной строки для создания сопоставления обработчика PHP

Используйте команды приведённые ниже для создания пула процессов IIS FastCGI который будет использовать `php-cgi.exe`выполняемый для PHP запросов. Замените значение параметра `fullPath` на абсолютный путь к файлу `php-cgi.exe`.

Пример #9 Создание IIS FastCGI пула процессов

```
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/fastCGI ^
/+[fullPath='c:\PHP\php-cgi.exe']
```

Настройка IIS для обработки специфических запросов PHP из командной строки показана ниже. Замените значение параметра`scriptProcessor` на абсолютный путь к файлу `php-cgi.exe`.

Пример #10 Создание сопоставления обработчика запросов PHP

```
%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/handlers ^
/+[name='PHP_via_FastCGI',
path='*.php',verb='*',modules='FastCgiModule',^
scriptProcessor='c:\PHP\php-cgi.exe',resourceType='Either']
```

Эта команда создает для IIS сопоставление обработчика для файлов с расширением *.php, который получается в результате и обрабатывается модулем FastCGI.

Замечание:
На этом шаге установка и настройка завершены. Следующие инструкции необязательны, но очень рекомендуются для достижения оптимальной функциональности и производительности PHP на IIS.

Представление и доступ к файловой системе ¶

При использовании IIS рекомендуется включить представление FastCGI в PHP. Это контролируется директивой `fastcgi.impersonate` в `php.ini` файле. Когда имперсонация включена, PHP будет выполнять все операции с файловой системой под аккаунтом, который был определён при аутентификации IIS. Это гарантирует, что при общем PHP процессе для всех сайтов IIS, PHP скрипты этих сайтов не будут иметь доступ к файлам друг друга до тех пор, пока IIS использует различные учетные записи для каждого из сайтов.

Для примера, в настройках по умолчанию IIS 7, включена анонимная аутентификация под стандартным пользователем IUSR. Это значит, что давая разрешение IIS выполнить PHP скрипт, также необходимо дать права на чтение этого скрипта аккаунту IUSR. Если PHP приложению необходимо выполнить запись в некоторые файлы или папки, тогда аккаунту IUSR следует дать права на запись в них.

Чтобы решить какой пользователь используется для идентификации в IIS 7, можно использовать следующие команды. Замените "Default Web Site" на имя IIS сайта, с которым вы работаете. На выходе, в XML конфигурации смотрите атрибут `userName`.

Пример #11 Определение аккаунта, используемого IIS при анонимной идентификации

```
%windir%\system32\inetsrv\appcmd.exe list config "Default Web
Site" ^
/section:anonymousAuthentication

<system.webServer>
  <security>
    <authentication>
      <anonymousAuthentication enabled="true" userName="IUSR" />
    </authentication>
   </security>
</system.webServer>
```

Замечание:
Если атрибут `userName` отсутствует в элементе `anonymousAuthentication`, или установлен как пустая строка, это значит, что удостоверение пула приложений используется как анонимное для этого веб-сайта.

Для изменения настроек доступа для файлов или папок, используйте интерфейс пользователя в проводнике Windows или команду `icacls`.

Пример #12 Настройка разрешения доступа к файлам

```
icacls C:\inetpub\wwwroot\upload /grant IUSR:(OI)(CI)(M)
```

Установка *index.php* как документ по умолчанию в IIS ¶

По умолчанию в IIS не установлено имя документа для обработки HTTP запросов по умолчанию. В PHP приложениях, обычно используется по умолчанию документ `index.php`. Чтобы добавить `index.php` в лист документов по умолчанию IIS, используйте такую команду:

Пример #13 Установка *index.php* как документ по умолчанию в IIS

```
%windir%\system32\inetsrv\appcmd.exe set config ^
-section:system.webServer/defaultDocument
/+"files.[value='index.php']" ^
/commit:apphost
```

FastCGI и PHP конфигурация с пересозданием процессов ¶

Настройка IIS FastCGI установок для утилизации PHP процессов с помощью команд приведена ниже. Опция FastCGI`instanceMaxRequests` устанавливает максимальное количество запросов, которое может обрабатываться одним `php-cgi.exe`процессом пока IIS не начнет их отключать. PHP переменная окружения `PHP_FCGI_MAX_REQUESTS` устанавливает как много запросов будет обрабатывать один `php-cgi.exe` процесс пока сам не начнет удалять их. Конечно, значение установленное для FastCGI`InstanceMaxRequests` меньше или равно `PHP_FCGI_MAX_REQUESTS`.

Пример #14 Настройка FastCGI и PHP пересоздаине

```
%windir%\system32\inetsrv\appcmd.exe      set      config      -
section:system.webServer/fastCgi ^
/[fullPath='c:\php\php-cgi.exe'].instanceMaxRequests:10000

%windir%\system32\inetsrv\appcmd.exe      set      config      -
section:system.webServer/fastCgi ^
/+"[fullPath='C:\{php_folder}\php-
cgi.exe'].environmentVariables.^
[name='PHP_FCGI_MAX_REQUESTS',value='10000']"
```

Установка таймаута FastCGI ¶

Увеличение параметра таймаута для FastCGI делается, если имеется долго выполняемый PHP скрипт. Два параметра контролируют таймаут, это: `activityTimeout` и `requestTimeout`. Используйте команды приведённые ниже для изменения настроек таймаута. Конечно, надо заменить значение параметра `fullPath` на полный путь к файлу `php-cgi.exe`.

Пример #15 Конфигурация установок таймаута FastCGI

```
%windir%\system32\inetsrv\appcmd.exe      set      config      -
section:system.webServer/fastCgi ^
```

```
/[fullPath='C:\php\php-
cgi.exe',arguments=''].activityTimeout:"90"  /commit:apphost

%windir%\system32\inetsrv\appcmd.exe      set      config      -
section:system.webServer/fastCgi ^
/[fullPath='C:\php\php-
cgi.exe',arguments=''].requestTimeout:"90"  /commit:apphost
```

Изменение положения файла *php.ini* ¶

PHP ищет файл `php.ini` некоторых метах и это даёт нам возможность изменить местоположения по умолчанию файла `php.ini`, используя переменную окружения `PHPRC`. Чтобы указать PHP загружать конфигурационный файл из определённого места используйте команды, приведённые ниже. Абсолютный путь до директории, где лежит файл `php.ini`, определяется как значение переменной окружения `PHPRC`.

Пример #16 Изменение положения файла *php.ini*

```
appcmd.exe set config  -section:system.webServer/fastCgi ^
/+"[fullPath='C:\php\php.exe',arguments=''].environmentVariables
.^
[name='PHPRC',value='C:\Some\Directory\']" /commit:apphost
```

Apache 1.3.x в Microsoft Windows ¶

Этот раздел содержит заметки и советы, специфичные для установки Apache 1.3.x для PHP в Microsoft Windows Также доступна отдельная страница с инструкциями и заметками по установке Apache 2.

Замечание:
Пожалуйста, для начала прочитайте руководство по установке !
Существует два способа настройки PHP для работы с Apache 1.3.x в Windows. Первый - использовать двоичный файл CGI (`php.exe` для PHP 4 и `php-cgi.exe` для PHP 5), второй - использовать Apache Module DLL. В обоих случаях необходимо отредакттировать `httpd.conf`для настройки Apache для работы с PHP и перезапустить сервер.
В настоящее время SAPI модуль более стабилен под Windows, поэтому мы рекомендуем использовать его вместо CGI так как он более прозрачен и безопасен.
Хотя существует несколько вариантов конфигурирования PHP под Apache, они достаточно просты для новичка. Пожалуйста, пользуйтесь документацией Apache для дальнейших указаний по конфигурированию.
Не забудьте перезагрузить сервер после изменения конфигурационного файла. Например, командами NET STOP APACHE и NET START APACHE, если Apache запущен как служба Windows, или с помощью обычных ярлыков.
Замечание: Помните, что при указании путей в конфигурационных файлах Apache под Windows, все обратные слеши, например,`c:\directory\file.ext` должны быть изменены на прямые: `c:/directory/file.ext`. Для путей с директориями также может понадобиться слеш в конце.

Установка PHP как модуля Apache ¶

Необходимо добавить следующие строки в файл Apache `httpd.conf`:

Пример #17 PHP как модуль Apache 1.3.x

Это предполагает, что PHP установлен в *c:\php*. Измените путь, если это не так.

Для PHP 4:

```
# Add to the end of the LoadModule section
# Don't forget to copy this file from the sapi directory!
LoadModule php4_module "C:/php/php4apache.dll"

# Add to the end of the AddModule section
AddModule mod_php4.c
```

Для PHP 5:

```
# Add to the end of the LoadModule section
LoadModule php5_module "C:/php/php5apache.dll"

# Add to the end of the AddModule section
AddModule mod_php5.c
```

Для обеих версий:

```
# Add this line inside the <IfModule mod_mime.c> conditional
brace
AddType application/x-httpd-php .php

# For syntax highlighted .phps files, also add
AddType application/x-httpd-php-source .phps
```

Установка как бинарного CGI файла ¶

Если PHP распакован в `C:\php\` как описано в разделе Manual Installation Steps, необходимо добавить следующие строки в конфигурационный файл Apache:

Пример #18 PHP и Apache 1.3.x как CGI

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php

# For PHP 4
Action application/x-httpd-php "/php/php.exe"

# For PHP 5
Action application/x-httpd-php "/php/php-cgi.exe"

# specify the directory where php.ini is
SetEnv PHPRC C:/php
```

Заметьте, что вторая строка в списке выше уже находится в *httpd.conf*, но она закомментирована. Кроме того, не забудьте заменить*c:/php/* на ваш настоящий путь к PHP.

Внимание
Используя установку CGI, ваш сервер открыт перед несколькими возможными уязвимостями. Пожалуйста, ознакомьтесь с разделом "Безопасность CGI" чтобы узнать, как можно защитить себя от таких атак.

В случае установки PHP как CGI нет такой удобной опции для подсветки синтаксиса исходников PHP как при установке в виде модуля. Если вы хотите использовать ее, необходимо использовать функцию highlight_file(). Для этого просто создайте PHP скрипт со следующим кодом: *<?php highlight_file('some_php_script.php'); ?>*.

Apache 2.x on Microsoft Windows ¶

Этот раздел содержит инструкции по установке PHP для Apache 2.x на системы Microsoft Windows. Также имеются указания для пользователей Apache 1.3.x.

Замечание:
Сначала вам необходимо прочитать пошаговое руководство по установке
Замечание: Поддержка Apache 2.2
Пользователям Apache 2.2 следует обратить внимание на то, что DLL файл для Apache 2.2 называется `php5apache2_2.dll`, а не `php5apache2.dll` и он доступен только для PHP 5.2.0 и более поздних версий.

Вам настоятельно рекомендуется ознакомиться с » Документацией по Apache, чтобы получить базовые знания о Apache 2.x Server. Также перед чтением данной справки обратите внимание на » Рекомендации для Windows по Apache 2.x.

Apache 2.x предназначен для работы в серверных версиях Windows, таких как Windows NT 4.0, Windows 2000, Windows XP или Windows 7. Хотя Apache 2.x может использоваться на Windows 9x, эти платформы не поддерживаются полностью, и некоторые функции не будут работать правильно. Исправление этой ситуации не планируется.

Скачайте наиболее актуальную версию » Apache 2.x и подходящую версию PHP. Следуйте Пошаговому руководству по установке и вернитесь для продолжения интеграции PHP и Apache.

Существует три пути установки PHP для Apache на Windows. Вы можете запустить PHP как обработчик, как CGI, или под FastCGI.

Замечание: Помните, что при указании путей в конфигурационных файлах Apache под Windows, все обратные слеши, например,`c:\directory\file.ext` должны быть изменены на прямые: `c:/directory/file.ext`. Для путей с директориями также может понадобиться слеш в конце.

Установка PHP как обработчика под Apache ¶

Вам необходимо добавить следующие строки в ваш конфигурационный файл Apache `httpd.conf` для загрузки PHP-модуля для Apache 2.x:

Пример #19 PHP как обработчик Apache 2.x

```
#
LoadModule php5_module "c:/php/php5apache2.dll"
AddHandler application/x-httpd-php .php

# конфигурирование пути к php.ini
PHPIniDir "C:/php"
```

Замечание: Не забудьте указать актуальный путь к директории PHP вместо `C:/php/` в приведенном примере. Позаботьтесь, чтобы в директиве LoadModule использовались либо `php5apache2.dll` либо `php5apache2_2.dll` и удостоверьтесь, что указанный файл фактически находится по пути, который вы указали в директиве.

Приведенная выше конфигурация позволит PHP обработать любой файл, который имеет расширение .php, даже если имеются другие расширения. К примеру, файл с именем `example.php.txt` будет запущен обработчиком PHP. Чтобы гарантировать, что только файлы, которые *имеют расширение* `.php` будут запущены, используйте следующую конфигурацию:

```
<FilesMatch \.php$>
      SetHandler application/x-httpd-php
 </FilesMatch>
```

Запуск PHP как CGI ¶

Вы должны обратиться к документации » Apache CGI для более полного понимания о запуске CGI под Apache.

Для запуска PHP как CGI, вам необходимо поместить ваши php-cgi файлы в директорию, обозначенную как директория CGI, используя директиву ScriptAlilas.

После этого вам необходимо добавить строку #! в PHP файлы, указывающую на местоположение исполняемого файла PHP.

Пример #20 PHP как CGI под Apache 2.x

```
#!C:/php/php.exe
<?php
  phpinfo();
?>
```

Внимание
Используя установку CGI, ваш сервер открыт перед несколькими возможными уязвимостями. Пожалуйста, ознакомьтесь с разделом "Безопасность CGI" чтобы узнать, как можно защитить себя от таких атак.

Запуск PHP под FastCGI ¶

Запуск PHP под FastCGI имеет ряд преимуществ по сравнению с запуском как CGI. Установка же довольно проста:

Получить mod_fcgid здесь: » http://httpd.apache.org/mod_fcgid/. исполняемые файлы под Win32 доступны для скачивания с этого сайта. Установите модуль в соответствии с прилагаемыми инструкциями.

Настроить свой веб сервер как указано ниже, позаботившись о соответствии всех путей установки на вашей конкретной системе:

Пример #21 Конфигурация Apache для запуска PHP как FastCGI

```
LoadModule fcgid_module modules/mod_fcgid.so

# Где находится ваш php.ini?
FcgidInitialEnv PHPRC        "c:/php"

AddHandler fcgid-script .php
FcgidWrapper "c:/php/php-cgi.exe" .php
```

Файлы с расширением .php в таком случае будут запускаться с помощью оболочки PHP FastCGI.

Sun, iPlanet and Netscape servers on Microsoft Windows ¶

This section contains notes and hints specific to Sun Java System Web Server, Sun ONE Web Server, iPlanet and Netscape server installs of PHP on Windows.

From PHP 4.3.3 on you can use PHP scripts with the NSAPI module to Apache compatibility are also available. For support in current web servers

CGI setup on Sun, iPlanet and Netscape servers ¶

To install PHP as a CGI handler, do the following:

- Copy `php4ts.dll` to your systemroot (the directory where you installed Windows)
- Make a file association from the command line. Type the following two lines:
- ```
  assoc .php=PHPScript
  ftype PHPScript=c:\php\php.exe %1 %*
  ```

- In the Netscape Enterprise Administration Server create a dummy shellcgi directory and remove it just after (this step creates 5 important lines in obj.conf and allow the web server to handle shellcgi scripts).
- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/shellcgi, File Suffix:php).
- Do it for each web server instance you want PHP to run

More details about setting up PHP as a CGI executable can be found here: » http://benoit.noss.free.fr/php/install-php.html

NSAPI setup on Sun, iPlanet and Netscape servers ¶

To install PHP with NSAPI, do the following:
  ```

- Copy `php4ts.dll` to your systemroot (the directory where you installed Windows)
- Make a file association from the command line. Type the following two lines:
- ```
  assoc .php=PHPScript
  ftype PHPScript=c:\php\php.exe %1 %*
  ```

- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/x-httpd-php, File Suffix: php).
- Edit `magnus.conf` (for servers >= 6) or `obj.conf` (for servers < 6) and add the following: You should place the lines after *mime types init*.
- ```
  Init fn="load-modules"
  funcs="php4_init,php4_execute,php4_auth_trans"
  shlib="c:/php/sapi/php4nsapi.dll"
  ```
- ```
  Init fn="php4_init" LateInit="yes" errorString="Failed to
  initialise PHP!" [php_ini="c:/path/to/php.ini"]
  ```

(PHP >= 4.3.3) The *php_ini* parameter is optional but with it you can place your *php.ini* in your web server configuration directory.

- Configure the default object in `obj.conf` (for virtual server classes [Sun Web Server 6.0+] in their `vserver.obj.conf`): In the <*Object name="default"*> section, place this line necessarily after all 'ObjectType' and before all 'AddLog' lines:
- ```
  Service fn="php4_execute" type="magnus-internal/x-httpd-
  php" [inikey=value inikey=value ...]
  ```

(PHP >= 4.3.3) As additional parameters you can add some special *php.ini*-values, for example you can set a *docroot="/path/to/docroot"*specific to the context *php4_execute* is called. For boolean ini-keys please use 0/1 as value, not *"On","Off",...* (this will not work correctly), e.g. *zlib.output_compression=1* instead of *zlib.output_compression="On"*

- This is only needed if you want to configure a directory that only consists of PHP scripts (same like a cgi-bin directory):
- ```
  <Object name="x-httpd-php">
  ```
- ```
  ObjectType fn="force-type" type="magnus-internal/x-httpd-
  php"
  ```
- ```
  Service fn=php4_execute [inikey=value inikey=value ...]
  ```
- ```
  </Object>
  ```

After that you can configure a directory in the Administration server and assign it the style *x-httpd-php*. All files in it will get executed as PHP. This is nice to hide PHP usage by renaming files to *.html*.

- Restart your web service and apply changes
- Do it for each web server instance you want PHP to run

Замечание:
More details about setting up PHP as an NSAPI filter can be found here: » http://benoit.noss.free.fr/php/install-php4.html
Замечание:

The stacksize that PHP uses depends on the configuration of the web server. If you get crashes with very large PHP scripts, it is recommended to raise it with the Admin Server (in the section "MAGNUS EDITOR").

CGI environment and recommended modifications in ***php.ini*** ¶

Important when writing PHP scripts is the fact that Sun JSWS/Sun ONE WS/iPlanet/Netscape is a multithreaded web server. Because of that all requests are running in the same process space (the space of the web server itself) and this space has only one environment. If you want to get CGI variables like *PATH_INFO*, *HTTP_HOST* etc. it is not the correct way to try this in the old PHP way with getenv() or a similar way (register globals to environment, *$_ENV*). You would only get the environment of the running web server without any valid CGI variables!

Замечание:
Why are there (invalid) CGI variables in the environment?

Answer: This is because you started the web server process from the admin server which runs the startup script of the web server, you wanted to start, as a CGI script (a CGI script inside of the admin server!). This is why the environment of the started web server has some CGI environment variables in it. You can test this by starting the web server not from the administration server. Use the command line as root user and start it manually - you will see there are no CGI-like environment variables.

Simply change your scripts to get CGI variables in the correct way for PHP 4.x by using the superglobal $_SERVER. If you have older scripts which use `$HTTP_HOST`, etc., you should turn on *register_globals* in `php.ini` and change the variable order too (important: remove *"E"* from it, because you do not need the environment here):

```
variables_order = "GPCS"
register_globals = On
```

Special use for error pages or self-made directory listings (PHP >= 4.3.3) ¶

You can use PHP to generate the error pages for *"404 Not Found"* or similar. Add the following line to the object in `obj.conf` for every error page you want to overwrite:

```
Error fn="php4_execute" code=XXX script="/path/to/script.php" [inikey=value inikey=value...]
```

where *XXX* is the HTTP error code. Please delete any other *Error* directives which could interfere with yours. If you want to place a page for all errors that could exist, leave the *code* parameter out. Your script can get the HTTP status code with *$_SERVER['ERROR_TYPE']*.

Another possibility is to generate self-made directory listings. Just create a PHP script which displays a directory listing and replace the corresponding default Service line for *type="magnus-internal/directory"* in `obj.conf` with the following:

```
Service fn="php4_execute" type="magnus-internal/directory" script="/path/to/script.php" [inikey=value inikey=value...]
```

For both error and directory listing pages the original URI and translated URI are in the variables *$_SERVER['PATH_INFO']* and*$_SERVER['PATH_TRANSLATED']*.

Note about **nsapi_virtual()** and subrequests (PHP >= 4.3.3) ¶

The NSAPI module now supports the nsapi_virtual() function (alias: virtual()) to make subrequests on the web server and insert the result in the web page. The problem is, that this function uses some undocumented features from the NSAPI library.

Under Unix this is not a problem, because the module automatically looks for the needed functions and uses them if available. If not,nsapi_virtual() is disabled.

Under Windows limitations in the DLL handling need the use of a automatic detection of the most recent `ns-httpdXX.dll` file. This is tested for servers till version 6.1. If a newer version of the Sun server is used, the detection fails and nsapi_virtual() is disabled.

If this is the case, try the following: Add the following parameter to *php4_init* in `magnus.conf`/`obj.conf`:

```
Init fn=php4_init ... server_lib="ns-httpdXX.dll"
```

where *XX* is the correct DLL version number. To get it, look in the server-root for the correct DLL name. The DLL with the biggest filesize is the right one.

You can check the status by using the phpinfo() function.

Замечание:
But be warned: Support for nsapi_virtual() is EXPERIMENTAL!!!

Sambar Server под Microsoft Windows ¶

Этот раздел содержит замечания и указания специфичные для » Sambar Server для Windows.

Замечание:
Пожалуйста, для начала прочитайте руководство по установке!
Этот список описывает установку ISAPI модуля для работы с сервером Sambar под Windows.

- Найдите файл, который называется `mappings.ini` (в папке config) в установочной директории Sambar.
- Откройте `mappings.ini` и добавьте следующую строку под *[ISAPI]*:

Пример #22 Конфигурация ISAPI для Sambar

```
#for PHP 4
*.php = c:\php\php4isapi.dll

#for PHP 5
*.php = c:\php\php5isapi.dll
```

(Если PHP установлен в *c:\php*.)

- Перезапустите Sambar, чтобы изменения вступили в силу.

Замечание:
Если вы хотите использовать PHP для связи с ресурсами на других компьютерах вашей сети, вам необходимо изменить учетную запись, которая используется службой сервера Sambar. По умолчанию это LocalSystem и удаленные ресурсы будут недоступны. Учетная запись может быть отредактирована с помощью опции "Службы" утилиты администрирования из панели управления Windows.

Xitami на Microsoft Windows ¶

Этот раздел содержит заметки и трюки, специфичные для » Xitami на платформе Windows.

Замечание:
Вам необходимо сначала прочитать инструкцию по установке!
Этот перечень действий описывает как установить PHP CGI библиотеку так, чтобы она работала с Xitami под Windows.
Замечание: Важно для пользователей CGI
Прочтите faq про cgi.force_redirec для получения важных подробностей. Эту директиву требуется установить в *0*. Если вы хотите использовать *$_SERVER['PHP_SELF']*, то необходимо включить cgi.fix_pathinfo директиву.
Внимание
Используя установку CGI, ваш сервер открыт перед несколькими возможными уязвимостями. Пожалуйста, ознакомьтесь с разделом "Безопасность CGI" чтобы узнать, как можно защитить себя от таких атак.

- Убедитесь, что веб-сервер работает, и ваш браузер попадет в консоль администратора Xitami. (обычно *http://127.0.0.1/admin*), и нажмите Конфигурация (Configuration).
- Перейдите в раздел "фильтры" и поместите расширение, которое должно обрабатывать PHP (то есть .php) в поле расширения файлов (.xxx).
- В фильтр команды или сценария поместите путь и имя вашего исполняемого PHP CGI файла, т.е. `C:\php\php.exe` для PHP 4 или`C:\php\php-cgi.exe` для PHP 5.
- Нажмите иконку "Сохранить".
- Перезапустите сервер, чтобы изменения вступили в силу.

Сборка из исходников ¶

В данной главе рассматривается процесс сборки PHP из исходников на Windows, с помощью инструментов сборки Microsoft. Если вы хотите скомпилировать PHP с помощью cygwin, пожалуйста, обращайтесь к главе Установка на Unix системы.

Смотрите документацию в Wiki по адресу: » http://wiki.php.net/internals/windows/stepbystepbuild

Установка расширений PHP в ОС Windows ¶

После установки PHP и веб-сервера на ОС Windows может понадобиться установить некоторые расширения для добавления функционала. Вы можете выбрать, какие расширения будут загружаться при старте PHP, модификацией вашего файла `php.ini`. Также вы можете загружать расширения динамически в ваших скриптах, используя функцию dl().

Библиотеки DLL расширений PHP имеют префикс *php_*.

Многие расширения *встроены* в Windows-версию PHP. Это значит, что дополнительные DLL-файлы и директива extension *не*используются для загрузки данных расширений. Таблица расширений PHP в Windows, содержащая список расширений, требующих (или обычно требующих) дополнительные DLL-файлы. Ниже приведен список встроенных расширений (обновлено PHP 5.0.4): BCMath,Caledar, COM, Ctype, DOM, FTP, LibXML, Iconv, ODBC, PCRE, Session, SimpleXML, SPL, SQLite, WDDX, XML и Zlib.

Место по умолчанию, в котором PHP ищет расширения - `C:\php5`. Для изменения данной настройки согласно вашей установке PHP отредактируйте файл `php.ini` следующим образом:

- измените опцию extension_dir так, чтобы она указывала на директорию, в которой расположены расширения или в которую вы поместили файлы `php_*.dll`. Например:

```
extension_dir = C:\php\extensions
```

- Включите одно или несколько расширений, которые вы хотите использовать, раскомментировав в файле `php.ini` строки вида*extension=php_*.dll*. Это делается удалением символа ";" в начале строки для каждого расширения, которое вы хотите включить.

Пример #23 Включение расширения Bzip2 в PHP-Windows

```
// измените следующую строку с ...
;extension=php_bz2.dll

// ... на
extension=php_bz2.dll
```

- Некоторые расширения для своей работы требуют дополнительные библиотеки DLL. Несколько из них находятся в распространяемом дистрибутиве PHP, в в корневой директории, но некоторые расширения, например, Oracle (`php_oci8.dll`), требуют наличия библиотек DLL, не включаемых в дистрибутив PHP. Не забудьте включить директорию `C:\php` в системную переменную PATH (данный процесс описан в отдельном разделе помощи).
- Некоторые из этих библиотек DLL не поставляются в дистрибутиве PHP. За деталями обратитесь к странице документации конкретного расширения. Также прочтите раздел руководства Установка расширений PECL, чтобы узнать дополнительную информацию о PECL. Все большее число расширений PHP можно найти в PECL, и данные расширения требуют отдельной загрузки.

Замечание: Если вы запустили PHP в качестве модуля сервера, не забудьте перезагрузить веб-сервер для применения ваших изменений в файле `php.ini`.

Следующая таблица описывает некоторые доступные расширения и требуемые дополнительные библиотеки dll.

Расширения PHP		
Расширение	**Описание**	**Примечания**
php_bz2.dll	Функции сжатия bzip2	—
php_calendar.dll	Функции преобразованиякалендарей	-
php_crack.dll	Функции Crack	—
php_ctype.dll	Семейство функций ctype	-
php_curl.dll	Функции библиотекиCURL	Требует библиотеки *libeay32.dll*, *ssleay32.dll* (в комплекте дистрибутива PHP)
php_dba.dll	DBA: функции абстрактного слоя DataBase (dbm-стиль)	—
php_dbase.dll	Функции dBase	—
php_dbx.dll	Функции dbx	—
php_exif.dll	Функции EXIF	Требуется библиотека php_mbstring.dll. Библиотека *php_exif.dll* должна загружаться *после php_mbstring.dll* в файле *php.ini*.
php_fbsql.dll	Функции FrontBase	-
php_fdf.dll	FDF: функции формата данных "Forms Data Format"	Требуется библиотека *fdftk.dll* (в комплекте дистрибутива PHP)
php_filepro.dll	Функции filePro	Доступ только для чтения
php_ftp.dll	Функции FTP	-
php_gd2.dll	Библиотека функций обработки изображенийGD	GD2
php_gettext.dll	Функции Gettext	В PHP версий <= 4.2.0 требуется библиотека *gnu_gettext.dll* (в комплекте дистрибутива PHP), в PHP версий >= 4.2.3 требуется библиотека *libintl-1.dll* и*iconv.dll* (в комплекте дистрибутива PHP).
php_hyperwave.dll	Функции HyperWave	—
php_iconv.dll	Функции конвертации кодировок ICONV	Требуется библиотека *iconv-1.3.dll* (в комплекте дистрибутива PHP), *iconv.dll*
php_ifx.dll	Функции Informix	Требуются библиотеки Informix
php_iisfunc.dll	Функции управления IIS	—
php_imap.dll	Функции IMAP POP3 и NNTP	—
php_ingres.dll	Функции Ingres	Требуются библиотеки Ingres
php_interbase.dll	Функции InterBase	Требуется библиотека *gds32.dll* (в комплекте дистрибутива PHP)
php_ldap.dll	Функции LDAP	Требует *libeay32.dll*, *ssleay32.dll* (в комплекте дистрибутива PHP)
php_mbstring.dll	Функции для работы смногобайтовыми (Multi-Byte) строками	—
php_mcrypt.dll	Функции кодированияMcrypt	Требуется библиотека *libmcrypt.dll*
php_mhash.dll	Функции Mhash	Требуется библиотека *libmhash.dll* (в ком-

Расширения PHP		
Расширение	**Описание**	**Примечания**
		плекте дистрибутива PHP)
php_mime_magic.dll	Функции Mimetype	Требуется файл *magic.mime* (в комплекте дистрибутива PHP)
php_ming.dll	Функции Ming для Flash	—
php_msql.dll	Функции mSQL	Требуется библиотека *msql.dll* (в комплекте дистрибутива PHP)
php_mssql.dll	Функции MSSQL	Требуется библиотека *ntwdblib.dll* (в комплекте дистрибутива PHP)
php_mysql.dll	Функции MySQL	Требуется библиотека *libmysql.dll* (в комплекте дистрибутива PHP)
php_mysqli.dll	Функции MySQLi	Требуется библиотека *libmysql.dll* (*libmysqli.dll* в PHP версий <= 5.0.2) (в комплекте дистрибутива PHP)
php_oci8.dll	Функции Oracle 8	Требуются клиентские библиотеки Oracle 8.1+
php_openssl.dll	Функции OpenSSL	Требуется библиотека *libeay32.dll* (в комплекте дистрибутива PHP)
php_pdf.dll	Функции PDF	—
php_pgsql.dll	Функции PostgreSQL	—
php_shmop.dll	Функции для работы с разделяемой памятью	—
php_snmp.dll	Функции для использования протокола SNMP	Только на Windows NT!
php_soap.dll	Функции SOAP	-
php_sockets.dll	Функции для работы с сокетами	—
php_sybase_ct.dll	Функции Sybase	Требуются клиентские библиотеки Sybase
php_tidy.dll	Функции Tidy	-
php_tokenizer.dll	Функции Tokenizer	-
php_w32api.dll	Функции W32api	—
php_xmlrpc.dll	Функции XML-RPC	Требуется библиотека *iconv.dll* (в комплекте дистрибутива PHP)
php_xslt.dll	Функции XSLT	Требуются библиотеки *sablot.dll*, *expat.dll*, *iconv.dll* (в комплекте дистрибутива PHP).
php_yaz.dll	Функции YAZ	Требуется библиотека *yaz.dll* (в комплекте дистрибутива PHP)
php_zip.dll	Функции для работы с файлами Zip	Доступ только для чтения
php_zlib.dll	Функции сжатия ZLib	-

Командная строка PHP в Microsoft Windows ¶

В этом разделе содержатся заметки и советы по работе PHP, запущенного из командной строки.

Замечание:
Сначала прочитайте руководство по установке!

PHP, запущенный из командной строки, может выполняться без каких-либо изменений в Windows.

```
C:\PHP5\php.exe -f "C:\PHP Scripts\script.php" -- -arg1 -arg2 -arg3
```

Но есть несколько простых шагов, которые могут упростить задачу. Некоторые из этих шагов уже могли быть выполнены, но будут приведены здесь для того, чтобы последовательность операций не была нарушена.

> Замечание:
> Как PATH, так и PATHEXT являются важными системные переменными в Windows, поэтому важно не затереть их текущее значение, а только дописать нужные данные в конец.

- Допишите расположение исполняемых файлов php (`php.exe`, `php-win.exe` или `php-cli.exe` в зависимости от ваших предпочтений и версии PHP) в конец переменной окружения PATH. О том, как добавить вашу директорию к PATH читайте в соответствующем разделе FAQ.
- Допишите *.PHP* расширение в конец переменной окружения PATHEXT. Это может быть сделано при изменении переменной PATH. Сделайте те же шаги, которые описаны в FAQ, но измените переменную `PATHEXT` вместо PATH.

> Замечание:
> Позиция, в которой вы разместите *.PHP*, будет определять, какой скрипт или программа будут запущенны для обработки файла с соответствующим расширением. Например, разместив *.PHP* перед *.BAT*, сначала будет запущен ваш скрипт, а не пакетный файл, если есть исполняемый файл с тем же именем.

- Ассоциируйте расширение *.PHP* с конкретным типом файла. Это можно сделать выполнив следующую команду:
- `assoc .php=phpfile`
- Ассоциируйте тип файла *phpfile* с соответствующим исполняемым PHP файлом. Это можно сделать выполнив команду:
- `ftype phpfile="C:\PHP5\php.exe" -f "%1" -- %~2`

Выполнение этих шагов позволит PHP скриптам выполняться из любой директории без необходимости указывать исполняемый PHP файл или расширение *.PHP*, а все параметры будут переданы в скрипт для обработки.

В примере ниже описываются некоторые изменения реестра, которые могут быть сделаны вручную.

Пример #24 Изменения реестра

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.php]
@="phpfile"
"Content Type"="application/php"
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\phpfile]
@="PHP Script"
"EditFlags"=dword:00000000
"BrowserFlags"=dword:00000008
"AlwaysShowExt"=""

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\phpfile\DefaultIcon]
@="C:\\PHP5\\php-win.exe,0"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\phpfile\shell]
@="Open"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\phpfile\shell\Open]
@="&Open"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\phpfile\shell\Open\command]
@="\"C:\\PHP5\\php.exe\" -f \"%1\" -- %~2"
```

С этими изменениями команда может быть записана как:

```
"C:\PHP Scripts\script" -arg1 -arg2 -arg3
```

или, если ваш путь в переменной окружения *"C:\PHP Scripts"* PATH:

```
script -arg1 -arg2 -arg3
```

Замечание:
Возникает небольшая проблема, если вы собираетесь использовать эту технику и используете ваши PHP-скрипты как фильтры командной строки, как в примере ниже:

```
dir | "C:\PHP Scripts\script" -arg1 -arg2 -arg3
```

или

```
dir | script -arg1 -arg2 -arg3
```

Вы можете увидеть, что ваш скрипт завис и ничего не делает. Для того чтобы оперативно получать об этом информацию, вам нужно внести в реестр еще некоторые изменений.

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer]
"InheritConsoleHandles"=dword:00000001
```

Дополнительную информацию можно найти в » базе знаний Microsoft, статья 321788.

Установка на платформах Cloud Computing ¶

Содержание ¶

PHP устанавливается на облачных платформах.

User Contributed Notes 1 note

Microsoft Azure ¶

PHP устанавливается на » облачную платформу Azure.

Смотрите также » Azure SDK для PHP.

User Contributed Notes 1 note

Amazon EC2 ¶

PHP устанавливается на » облачную платформу EC2.

Смотрите также » AWS SDK для PHP.

User Contributed Notes 2 notes

Менеджер процессов FastCGI (FPM) ¶

Содержание ¶

FPM (Менеджер процессов FastCGI) является альтернативной реализацией PHP FastCGI с несколькими дополнительными возможностями обычно используемыми для высоконагруженных сайтов.

Эти возможности включают в себя:

- продвинутое управление процессами с корректной (graceful) процедурой остановки и запуска;
- возможность запуска воркеров с различными uid/gid/chroot-окружением, а также запуска на различных портах с использованием разных php.ini (замещение safe_mode);
- логирование стандартных потоков вывода (stdout) и ошибок (stderr);
- аварийный перезапуск в случае внезапного разрушения opcode-кэша;
- поддержка ускоренной загрузки (accelerated upload);
- "slowlog" - логирование необычно медленно выполняющихся скриптов (не только их имена, но также и их трассировки. Это достигается с помощью ptrace и других подобных утилит для чтения данных исполнения удаленных процессов);
- fastcgi_finish_request() - специальная функция для завершения запроса и сброса всех буферов данных, причем процесс может продолжать выполнение каких-либо длительных действий (конвертирование видео, обработка статистики и т.п.);
- Динамическое/статическое порождение дочерних процессов;
- Базовая информация о статусе SAPI (аналогично Apache mod_status);
- Конфигурационный файл, основанный на php.ini.

User Contributed Notes *4 notes*

Установка ¶

Компиляция из исходников ¶

Для того, чтобы включить FPM при сборке PHP, добавьте строку *--enable-fpm*

Существуют также несколько других опций конфигурации FPM (все опциональны):

- *--with-fpm-user* - установить пользователя FPM (по умолчанию - nobody).
- *--with-fpm-group* - установить FPM группу (по умолчанию - nobody).
- *--with-fpm-systemd* - Включает интеграцию с systemd (по умолчанию - no).
- *--with-fpm-acl* - Использовать Список Контроля Доступа (ACL) POSIX (по умолчанию - no). С версии 5.6.5

User Contributed Notes *3 notes*

Настройка ¶

FPM использует синтаксис `php.ini` для своего файла конфигурации `php-fpm.conf` и файлов конфигурации пулов.

Список глобальных директив php-fpm.conf

`pid` string

Путь к PID файлу. Значение по умолчанию: none.

`error_log` string

Путь к файлу журнала ошибок. Значение по умолчанию: *#INSTALL_PREFIX#/log/php-fpm.log*.

`log_level` string

Уровень журналирования ошибок. Возможные значения: alert, error, warning, notice, debug. Значение по умолчанию: notice.

`emergency_restart_threshold` int

При данном числе рабочих процессов, завершенных с SIGSEGV или SIGBUS за промежуток времени, установленный*emergency_restart_interval* FPM будет перезагружен. Значение 0 означает 'Off' (отключено). Значение по умолчанию: 0 (Off).

`emergency_restart_interval` mixed

Интервал времени, используемый emergency_restart_interval, чтобы определить, когда FPM будет мягко перезагружен. Это полезно для избежания случайных повреждений общей памяти ускорителя (accelerator). Доступные единицы измерения: s(секунды), m(минуты), h(часы), или d(дни). Единица измерения по умолчанию: секунды. Значение по умолчанию: 0 (Off).

`process_control_timeout` mixed

Время, в течение которого дочерние процессы ждут ответа на сигналы мастер-процессу. Доступные единицы измерения: s(секунды), m(минуты), h(часы) или d(дни). Единица измерения по умолчанию: секунды. Значение по умолчанию: 0.

`daemonize` boolean

Запустить FPM в фоновом режиме. Установите значение 'no', чтобы запустить FPM в диспетчере для отладки. Значение по умолчанию: yes.

Список директив для пулов.

Вы можете запускать несколько FPM пулов процессов с различными настройками. Эти параметры могут быть переданы пулу.

`listen` string

Адрес, который будет принимать FastCGI-запросы. Синтаксис: 'ip.add.re.ss:port', 'port', '/path/to/unix/socket'. Эта опция обязательна для каждого пула.

`listen.backlog` int

Устанавливает listen(2) backlog. Значение '-1' означает неограниченно. Значение по умолчанию: -1.

`listen.allowed_clients` string

Список IPv4 адресов FastCGI-клиентов, которые имеют право подключения. Эквивалент переменной окружения среды FCGI_WEB_SERVER_ADDRS в оригинальном PHP FastCGI (5.2.2+). Имеет смысл только с TCP-сокетом. Каждый адрес должен быть отделен запятой. Если оставить значение пустым, то соединения будут приниматься с любого IP. Значение по умолчанию: any.

`listen.owner` string

Задает права для unix socket, если они используются. В Linux, чтобы разрешить соединения web серверу, должны быть установлены права на чтение/запись. Во многих основанных на BSD системах возможность соединения не зависит от прав доступа. Значение по умолчанию: используется пользователь и группа, от имени которого запущен сервер, установлен режим 0660.

`listen.group` string

См. *listen.owner*.

`listen.mode` string

См. *listen.owner*.

`user` string

Unix-пользователь FPM-процессов. Этот параметр является обязательным.

`group` string

Unix-группа FPM-процессов. Если не установлен, группа по умолчанию равняется имени пользователя.

`pm` string

Выбор того, как менеджер процессов будет контролировать создание дочерних процессов. Возможные значения: *static*,*ondemand*, *dynamic*. Этот параметр является обязательным.

static - фиксированное число дочерних процессов (*pm.max_children*).

ondemand - число процессов, порождающихся по требованию (когда появляются запросы, в отличии от опции dynamic, когда стартует определенное количество процессов, равное *pm.start_servers*, вместе с запуском службы.

dynamic - динамически изменяющееся число дочерних процессов, задается на основании следующих директив:*pm.max_children*, *pm.start_servers*, *pm.min_spare_servers*, *pm.max_spare_servers*.

`pm.max_children` int

Число дочерних процессов, которые будут созданы, когда *pm* установлен в *static*, или же максимальное число процессов, которые будут созданы, когда *pm* установлен в *dynamic*. Этот параметр является обязательным.

Этот параметр устанавливает ограничение на число одновременных запросов, которые будут обслуживаться. Эквивалент директивы ApacheMaxClients с mpm_prefork и переменной окружения среды `PHP_FCGI_CHILDREN` в в оригинальном PHP FastCGI.

`pm.start_servers` int

Число дочерних процессов, создаваемых при запуске. Используется, только когда *pm* установлен в *dynamic*. Значение по умолчанию: min_spare_servers + (max_spare_servers - min_spare_servers) / 2.

`pm.min_spare_servers` int

Желаемое минимальное число неактивных процессов сервера. Используется, только когда *pm* установлено в *dynamic*. Кроме того, это обязательный параметр в этом случае.

`pm.max_spare_servers` int

Желаемое максимальное число неактивных процессов сервера. Используется, только когда *pm* установлен в *dynamic*. Кроме того, это обязательный параметр в этом случае.

`pm.max_requests` int

Число запросов дочернего процесса, после которого процесс будет перезапущен. Это полезно для избежания утечек памяти при использовании сторонних библиотек. Для бесконечной обработки запросов укажите '0'. Эквивалент `PHP_FCGI_MAX_REQUESTS`. Значение по умолчанию: 0.

`pm.status_path` string

Ссылка, по которой можно посмотреть страницу состояния FPM. Если значение не установлено, то страница статуса отображаться не будет. Значение по умолчанию: none.

`ping.path` string

Ссылка на ping-страницу мониторинга FPM. Если значение не установлено, ping-страница отображаться не будет. Может быть использовано для тестирования извне, чтобы убедиться, что FPM жив и реагирует. Обратите внимание, что значение должно начинаться с косой черты (/).

`ping.response` string

Эта директива может быть использована на настройки ответа на ping-запрос. Ответ формируется как text/plain со кодом ответа 200. Значение по умолчанию: pong.

`request_terminate_timeout` mixed

Таймаут для обслуживания одного запроса, после чего рабочий процесс будет завершен. Этот вариант следует использовать, когда опция 'max_execution_time' в php.ini не останавливает выполнение скрипта по каким-то причинам. Значение '0' означает 'выключено'. Доступные единицы измерения: s(секунды), m(минуты), h(часы) или d(дни). Значение по умолчанию: 0.

`request_slowlog_timeout` mixed

Таймаут для обслуживания одного запроса, после чего PHP backtrace будет сохранен в файл 'showlog'. Значение '0' означает 'выключено'. Доступные единицы измерения: s(секунды), m(минуты), h(часы) или d(дни). Значение по умолчанию: 0.

`slowlog` string

Лог-файл для медленных запросов. Значение по умолчанию: *#INSTALL_PREFIX#/log/php-fpm.log.slow*.

`rlimit_files` int

Устанавливает лимит дескрипторов открытых файлов rlimit. Значение по умолчанию: определяется значением системы.

`rlimit_core` int

Устанавливает максимальное количество используемых ядер rlimit. Возможные значения: 'unlimited' или целое число большее или равное 0. Значение по умолчанию: определяется значением системы.

`chroot` string

Директория chroot окружения при старте. Это значение должно быть определено как абсолютный путь. Если значение не установлено, chroot не используется.

`chdir` string

Chdir изменяет текущую директорию при старте. Это значение должно быть определено как абсолютный путь. Значение по умолчанию: текущая директория или / при использовании chroot.

`catch_workers_output` boolean

Перенаправление STDOUT и STDERR рабочего процесса в главный лог ошибок. Если не установлен, STDOUT и STDERR будут перенаправлены в /dev/null в соответствии со спецификацией FastCGI. Значение по умолчанию: no.

Можно передать дополнительные переменные окружения и обновить настройки PHP для определенного пула. Для этого вам необходимо добавить следующие параметры в файл настройки пула.

Пример #1 Передача переменных окружения и настроек PHP пулу

```
env[HOSTNAME] = $HOSTNAME
env[PATH] = /usr/local/bin:/usr/bin:/bin
env[TMP] = /tmp
env[TMPDIR] = /tmp
env[TEMP] = /tmp

php_admin_value[sendmail_path] = /usr/sbin/sendmail -t -i -f
www@my.domain.com
php_flag[display_errors] = off
php_admin_value[error_log] = /var/log/fpm-php.www.log
php_admin_flag[log_errors] = on
php_admin_value[memory_limit] = 32M
```

PHP настройки, переданные через *php_value* или *php_flag* перезапишут их предыдущие значения. Пожалуйста, обратите внимание, что определения disable_functions или disable_classes не будут будут перезаписывать ранее определенные в *php.ini* значения, а добавят новые значения.

Настройки, определенные через *php_admin_value* и *php_admin_flag*, не могут быть перезаписаны через ini_set().

Начиная с версии 5.3.3 настройки PHP можно устанавливать через веб-сервер.

Пример #2 Установка настроек PHP в nginx.conf

```
set $php_value "pcre.backtrack_limit=424242";
set $php_value "$php_value \n pcre.recursion_limit=99999";
fastcgi_param  PHP_VALUE $php_value;

fastcgi_param  PHP_ADMIN_VALUE "open_basedir=/var/www/htdocs";
```

Предостережение
Так как эти настройки передаются в php-fpm как FastCGI-заголовки, php-fpm не должен быть привязан к общедоступному адресу из мира. В противном случае любой сможет изменить настройки PHP. Смотрите также listen.allowed_clients.

User Contributed Notes *5 notes*

Установка расширений PECL ¶

Содержание ¶

User Contributed Notes *2 notes*

Введение в установку PECL ¶

» PECL - это репозиторий расширений PHP, доступ к которым предоставляется через систему » PEAR. Эта часть руководства предназначена для демонстрации того, как вы можете получить и установить расширения PECL.

Эти инструкции подразумевают, что */your/phpsrcdir/* является путем к каталогу с дистрибутивом исходного кода PHP, а *extname* - это имя расширения PECL, так что вносите соответствующие коррективы. Эти инструкции также подразумевают знакомство с » командой pear. Информация в руководстве PEAR для команды *pear* также применима для команды *pecl*.

Для того, чтобы расширение можно было использовать, оно должно быть собрано, установлено и загружено. Описанные ниже методы предоставляют различные рекомендации по поводу того, как собрать и установить расширения, но без при-

менения их автоматической загрузки. Расширения могут быть загружены с помощью директивы extension в файле `php.ini`, или путем использования функции dl().

В процессе сборки модулей PHP важно иметь нужные версии необходимых утилит (autoconf, automake, libtool и т.д.). Информацию об этих утилитах и их версиях можно посмотреть в разделе "» Инструкции по осуществлению анонимного доступа к Git".

Загрузка расширений PECL ¶

Есть несколько вариантов для загрузки расширений PECL, в том числе:

- Команда *pecl install extname* автоматически скачивает код расширений, поэтому в этом случае нет нужды в отдельной скачке этих файлов.
- » http://pecl.php.net/ Веб-сайт PECL содержит информацию о различных расширениях, предоставляемых командой разработчиков PHP. Информация, доступная на этом веб-сайте, включает в себя: лог изменений, новости релизов, требования и другие подобные детали.
- *pecl download extname* Расширения PECL, имеющих опубликованные релизы на сайте PECL, доступны для скачивания и установки с помощью » команды pecl. Можно также указать отдельные ревизии для установки.
- SVN Большинство расширений PECL, также, находятся в SVN. Веб-интерфейс для просмотра доступен по адресу» http://svn.php.net/viewvc/pecl/. Для загрузки напрямую из SVN используется следующая последовательность команд:

  ```
  $ svn checkout http://svn.php.net/repository/pecl/extname/trunk extname
  ```

- Загрузка для Windows На данный момент проект PHP не компилирует бинарные файлы Windows для расширений PECL. Для компиляции PHP под Windows прочитайте соответствующий раздел.

User Contributed Notes *1 note*

Установка PHP-расширения в Windows ¶

В Windows есть два способа загрузки PHP-расширения: скомпилировать его вместе с PHP или загрузить DLL. Загрузка заранее скомпилированного расширения является наиболее простым и предпочитаемым способом.

Для загрузки расширения, оно должно присутствовать на вашей системе в виде ".dll" файла. Все расширения автоматически и периодически компилируются командой PHP (см. следующий раздел для загрузки).

За инструкциями по компиляции расширения в PHP обратитесь к разделу "Сборка из исходников".

Для компиляции отдельного расширения (или DLL-файла), обратитесь к разделу " Сборка из исходников". Если DLL-файла нет ни в стандартной поставке PHP ни в PECL, возможно, вам придется скомпилировать его вручную.

Где найти расширение? ¶

PHP-расширения обычно имеют имена вида "php_*.dll" (где звездочка обозначает имя расширения) и располагаются в папке "PHP\ext".

PHP поставляет расширения наиболее полезные большинству разработчиков. Такие расширения называются "основными" ("core").

Однако если вам требуется функционал, который не предоставляется ни одним из основных расширений, возможно, нужное вам расширение есть в PECL. Библиотека расширений сообщества PHP (The PHP Extension Community Library, PECL) является хранилищем расширений PHP, предоставляя каталог и хостинг всех известных расширений для скачки и дальнейшей разработки расширений в PHP.

Если вы разработали какое-либо расширение для собственных нужд, возможно, вы захотите хранить его в PECL, так, чтобы другие также могли воспользоваться результатами вашего труда. Хорошим побочным эффектом будет неплохой шанс получить обратную связь, благодарности (надеемся, что так и будет), сообщения об ошибках и даже исправления/патчи. Пожалуйста, прочтите» публикация PECL; перед отправкой вашего расширения в PECL.

Какое расширение нужно загрузить? ¶

Очень часто существует несколько версий расширения DLL:

- Различные номера версий (по крайней мере первые два числа должны совпадать)
- Различные настройки потокобезопасности
- Различная архитектура процессора (x86, x64, ...)
- Различные настройки отладки
- *и т.д.*

Помните, что настройки ваших расширений должны совпадать со всеми настройками используемого вами бинарного файла PHP. Следующий PHP-скрипт выведет вам *все* настройки PHP:

Пример #1 Вызов phpinfo()

```
<?php
phpinfo();
?>
```

Или запустите из командной строки:

```
drive:\\path\to\php\executable\php.exe -i
```

Загрузка расширения ¶

Наиболее распространенным способом загрузки PHP-расширения является его включение в конфигурационном файле `php.ini`. Обратите внимание, что большинство расширений уже прописаны в вашем `php.ini` и для их активации вам просто нужно удалить точку с запятой.

```
;extension=php_extname.dll
extension=php_extname.dll
```

Однако некоторые веб-серверы создают путаницу, т.к. они не используют `php.ini`, расположенный в дистрибутиве PHP, а используют свой собственный. Узнать, где находится используемый сервером `php.ini`, можно посмотрев на выводимый путь в phpinfo():

```
Configuration File (php.ini) Path  C:\WINDOWS
Loaded Configuration File   C:\Program Files\PHP\5.2\php.ini
```

После активации расширения сохраните `php.ini`, перезагрузите веб-сервер и снова проверьте phpinfo(), в нем должен появиться отдельный раздел с новым расширением.

Решение проблем ¶

Если расширение не появилось в выводе phpinfo(), проверьте лог-файлы на наличие сообщений об ошибках.

Если вы используете PHP в командной строке (CLI), ошибки загрузки расширения будут доступны сразу же на экране.

Если вы используете PHP на веб-сервере, расположение и формат лог-файлов сильно зависит от используемого вами веб-сервера. Пожалуйста, обратитесь к документации вашего веб-сервера, т.к. в данном случае эта ситуация не управляется самим PHP.

Частой проблемой является расположение DLL-файлов, значение " extension_dir" в `php.ini`, а также несовпадение настроек компиляции.

Если проблемой является несовпадение настроек компиляции, то возможно, вы скачали не тот DLL-файл. Попробуйте снова скачать расширение, на этот раз с правильными настройками. Еще раз, информация функции phpinfo() сильно помогает в этом случае.

User Contributed Notes *1 note*

Компиляция разделяемых расширений с помощью команды pecl ¶

PECL позволяет легко создавать разделяемые расширения PHP. Используя » команду pecl, выполните следующее:

```
$ pecl install extname
```

Эта команда загрузит исходный код для расширения *extname*, скомпилирует и установит `extname.so` в вашу директорию extension_dir. Файл `extname.so` может быть затем загружен в `php.ini`

По умолчанию, команда *pecl* не будет устанавливать пакеты, отмеченные состоянием *alpha* или *beta*. Если нет доступных стабильных (*stable*) версий пакетов, вы можете установить *beta*-версию пакета, используя следующую команду:

```
$ pecl install extname-beta
```

Также, вы можете установить определенную версию используя такой вариант:

```
$ pecl install extname-0.1
```

Замечание:
После подключения расширения в `php.ini` необходимо перезапустить веб-сервер для того, чтобы изменения вступили в силу.

User Contributed Notes *4 notes*

Компиляция разделяемых расширений с помощью phpize ¶

Иногда использование инсталлятора *pecl* не подходит. Это может быть связано с тем, что вы находитесь за файерволом или из-за того, что расширение, которое вы хотите установить, недоступно в PECL-совместимом пакете (к примеру, расширения из SVN, у которых еще нет релизов). Если вам необходимо собрать такое расширение, вы можете использовать низкоуровневые утилиты для выполнения сборки вручную.

Команда *phpize* используется для подготовки окружения PHP-расширения. В следующем примере директория, где находятся исходные коды расширения, называется `extname`:

```
$ cd extname
$ phpize
$ ./configure
$ make
# make install
```

В случае успешной установки будет создан файл `extname.so` и помещен в PHP директорию расширений. Вам будет необходимо добавить строку *extension=extname.so* в `php.ini` перед использованием этого расширения.

Если в системе нет *phpize*, но существует возможность установки заранее скомпилированных пакетов (типа RPM), убедитесь, что установлена соответствующая версия пакета PHP для разработчиков, так как они часто содержат команду *phpize* с подходящими файлами заголовков для сборки PHP и его расширений.

Для дополнительной информации используйте команду phpize --help.

User Contributed Notes *5 notes*

php-config ¶

php-config - это простой шелл-скрипт для получения информации о конфигурации установленного PHP.

При компиляции расширений, если у вас установлено сразу несколько версий PHP, вы должны уточнить нужную версию с помощью опции *--with-php-config* во время конфигурирования сборки, указав путь до соответствующего php-config скрипта.

Список параметров командной строки доступных для php-config скрипта всегда можно получить запустив php-config с параметром -h :

```
Usage: /usr/local/bin/php-config [OPTION]
Options:
  --prefix            [...]
  --includes          [...]
  --ldflags           [...]
  --libs              [...]
  --extension-dir     [...]
  --include-dir       [...]
  --php-binary        [...]
  --php-sapis         [...]
  --configure-options [...]
  --version           [...]
  --vernum            [...]
```

Опции скрипта

Опция	Описание
--prefix	Директория, в которой установлен PHP, например /usr/local
--includes	Список -I опций со всеми подключаемыми файлами
--ldflags	LD-флаги, с которыми был скомпилирован PHP
--libs	Внешние библиотеки, с которыми был скомпилирован PHP
--extension-dir	Директория, в которой по умолчанию ищутся расширения
--include-dir	Директория, куда по умолчанию устанавливаются заголовочные файлы
--php-binary	Полный путь до исполняемых файлов php CLI или CGI
--php-sapis	Показывает все доступные модули SAPI
--configure-options	Конфигурационные опции для воссоздания настроек текущей установки PHP.
--version	Версия PHP
--vernum	Версия PHP в виде целого числа

Компиляция расширений PECL статически в PHP ¶

Возможно, вы захотите собрать расширение PECL статично в ваш бинарный файл PHP. Для этого необходимо поместить код расширения в директорию `php-src/ext/` и вызвать перегенерацию конфигурационных скриптов через систему сборки PHP.

```
$ cd /your/phpsrcdir/ext
$ pecl download extname
$ gzip -d < extname.tgz | tar -xvf -
$ mv extname-x.x.x extname
```

В результате будет создана следующая директория:

/your/phpsrcdir/ext/extname

После этого, выполните заново сборку конфигурационного скрипта PHP и затем соберите PHP как обычно:

$ cd /your/phpsrcdir
$ rm configure
$ ./buildconf --force
$ ./configure --help
$ ./configure --with-extname --enable-someotherext --with-foobar
$ make
$ make install

Замечание: Для запуска 'buildconf' скрипта вам понадобится autoconf версии 2.13 и automake версии 1.4+ (более новые версии autoconf могут работать, но это не поддерживается).

В зависимости от расширения будет использоваться одна из двух опций *--enable-extname* или *--with-extname* . Обычно, если расширение не требует подключения внешних библиотек, используется *--enable*. Чтобы узнать это, выполните следующую команду после buildconf:

$ ./configure --help | grep extname

User Contributed Notes *1 note*

Проблемы? ¶

Содержание ¶

User Contributed Notes *1 note*

Читайте FAQ ¶

Некоторые проблемы возникают чаще других. Наиболее часто встречающиеся из них перечислены в разделе PHP FAQ этого руководства.

Другие проблемы ¶

Если вы все еще не можете установить PHP, список рассылки, возможно, сможет помочь вам. Не забудьте сначала просмотреть архив, возможно кто-либо уже имел схожую с вашей проблему и успешно её решил. Архив рассылки доступен на странице поддержки:» http://www.php.net/support.php Для подписки на список рассылки об инсталляции PHP пришлите пустое сообщение на » php-install-subscribe@lists.php.net. Адрес списка рассылки: » php-install@lists.php.net.

Если вы хотите получить помощь из списка рассылки, пожалуйста, постарайтесь точно описать детали вашего окружения (такие как операционная система, версия PHP, используемый веб-сервер, используете ли вы PHP как CGI или модулем, безопасный режим, и т.д.), а также достаточное количество кода, чтобы другие смогли воспроизвести вашу проблему и попробовать решить ее.

Bug reports ¶

Если вы думаете, что нашли ошибку в PHP, пожалуйста сообщите об этом. Возможно, разработчики PHP не знают об этой ошибке и пока вы не сообщите о ней, мало шансов что она будет исправлена. Вы можете сообщить об ошибке, используя систему отслеживания ошибок по адресу: » http://bugs.php.net/. Пожалуйста, не шлите сообщения об ошибках в список рассылки или на личные адреса разработчиков. Система отслеживания ошибок также пригодна для для запроса нововведений.

Прочитайте: » Как сообщать об ошибках перед тем как сообщать о них.

Конфигурация запуска ¶

Содержание ¶

User Contributed Notes *14 notes*

Файл конфигурации ¶

Файл конфигурации (`php.ini`) считывается при запуске PHP. Для версий серверных модулей PHP это происходит только один раз при запуске веб сервера. Для CGI и CLI версий это происходит при каждом вызове.

Поиск `php.ini` производится в следующих местах (по порядку поиска):

- По месту расположения модуля SAPI (*PHPIniDir* директива Apache 2, *-c* параметр командной строки CGI и CLI, *php_ini* параметр в NSAPI, *PHP_INI_PATH* переменная среды в THTTPD)
- Переменная среды `PHPRC`. До PHP 5.2.0 поиск по переменной производился после поиска в реестре, указанном ниже.

- Как и в PHP 5.2.0, местоположение файла *php.ini* может быть указано для различных версий PHP. Следующие ключи реестра исследуются при поиске: *[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y.z]*, *[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y]* и*[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x]*, где x, y и z подразумевают major, minor и release версии PHP. Если также имеется значение*IniFilePath* в любом из этих ключей, то местонахождение *php.ini* будет определено первым ключом по порядку. (только для Windows).
- *[HKEY_LOCAL_MACHINE\SOFTWARE\PHP]*, значение *IniFilePath* (только для Windows).
- Текущая директория (исключая CLI).
- Директория веб сервера (для SAPI модулей), или директория PHP (иначе в директории Windows).
- В директории Windows (`C:\windows` или `C:\winnt`) (для Windows), или --*with-config-file-path* с выбором при компиляции.

Если файл `php-SAPI.ini` существует (где SAPI - это тип интерфейса, который используется, например, `php-cli.ini` или`php-apache.ini`), то он используется вместо `php.ini`. Тип интерфейса между веб-сервером и PHP может быть определен с помощью функции php_sapi_name().

Замечание:
Веб сервер Apache изменяет текущую директорию на корневую при запуске, в результате чего PHP считывает `php.ini` из корневой файловой системы, если файл существует.

В `php.ini` можно использовать переменные окружения, как показано в примере ниже.

Пример #1 *php.ini* Environment Variables

```
; PHP_MEMORY_LIMIT берется из окружения
memory_limit = ${PHP_MEMORY_LIMIT}
```

Директивы `php.ini` обрабатываемые расширениями документированы на соответствующих страницах расширений. Список директив ядра имеется в приложении. Не все директивы PHP документированы в этом руководстве: для ознакомления с полным списком директив доступных в вашей версии PHP, пожалуйста, прочитайте комментарии вашего `php.ini` файла. Кроме того, вы можете найти полезной » последнюю версию `php.ini` из Git.

Пример #2 Пример *php.ini*

```
; любой текст в строке после точки с запятой (;) без кавычек игнорируется
[php] ; маркеры разделов (текст в квадратных скобках) также игнорируется
; Могут быть установлены следующие логические значения:
;    true, on, yes
; или false, off, no, none
register_globals = off
track_errors = yes

; вы можете заключать строки в двойные кавычки
```

```
include_path = ".:/usr/local/lib/php"

; обратный слеш обрабатывается так же, как любые другие символы
include_path = ".;c:\php\lib"
```

С версии PHP 5.1.0 стало возможным обращаться к существующим ini-переменным из ini-файлов. Пример: *open_basedir = ${open_basedir} ":/new/dir"*.

User Contributed Notes *6 notes*

.user.ini files ¶

С версии PHP 5.3.0 включена поддержка конфигурационных INI-файлов на уровне каталога. Эти файлы обрабатываются *только*CGI/FastCGI SAPI. Эта функция исключает расширение PECL htscanner. Если у вас Apache, пользуйтесь `.htaccess` файлами для достижения того же эффекта.

В дополнение к основному файлу`php.ini` , PHP ищет INI-файлы в каждой директории, начиная с директории запрошенного PHP-файла и продолжает поиск до корневой директории (установленной в `$_SERVER['DOCUMENT_ROOT']`). Если PHP-файл находится вне корневой директории, то сканируется только его директория.

Только INI-настройки с режимами **PHP_INI_PERDIR** и **PHP_INI_USER** будут распознаны в INI-файлах в стиле .user.ini.

Две новых INI-директивы, *user_ini.filename* и *user_ini.cache_ttl* контролируют использование пользовательских INI-файлов.

user_ini.filename устанавливает имя файла, по которому PHP производит поиск в каждой директории; если установлена пустая строка, то PHP поиск не производит. По умолчанию *.user.ini*.

user_ini.cache_ttl устанавливает насколько часто пользовательские INI-файлы будут обновляться. По умолчанию период обновления составляет 300 секунд (5 минут).

User Contributed Notes *4 notes*

Где могут быть установлены параметры конфигурации ¶

Эти режимы определяют, когда и где директива PHP может или не может быть установлена, и каждая директива в руководстве относится к одному из этих режимов. К примеру, некоторые настройки могут быть установлены с помощью PHP-скрипта, использующего ini_set(), тогда как другие могут требовать `php.ini` или `httpd.conf`.

К примеру, директива output_buffering соответствует *PHP_INI_PERDIR*, поэтому она не может быть установлена через ini_set(). Тем не менее, директива display_errors соответствует *PHP_INI_ALL*, поэтому она может быть установлена отовсюду, включая ini_set().

Определение режимов PHP_INI_*	
Режим	**Смысл**
PHP_INI_USER	Значение может быть установлено в пользовательских скриптах (с помощью ini_set()) или в реестре Windows. С версии PHP 5.3 значение может быть установлено в *.user.ini*
PHP_INI_PERDIR	Значение может быть установлено в *php.ini*, *.htaccess* или *httpd.conf* (С версии PHP 5.3)
PHP_INI_SYSTEM	Значение может быть установлено в *php.ini* или *httpd.conf*
PHP_INI_ALL	Значение может быть установлено отовсюду

Как изменить настройки конфигурации ¶

Запуск PHP как модуля Apache ¶

Когда PHP используется как модуль Apache вы также можете менять настройки конфигурации, используя директивы в файлах конфигурации Apache (например, `httpd.conf`) и файлах `.htaccess`. Для этого вам необходимы "AllowOverride Options" или "AllowOverride All" привилегии.

Есть несколько директив Apache, которые позволяют вам изменить конфигурацию PHP посредством файлов конфигурации Apache. Директивы для **`PHP_INI_ALL`**, **`PHP_INI_PERDIR`**, и **`PHP_INI_SYSTEM`**, вы можете увидеть в приложении Список директив php.ini

`php_value` `name value`

Устанавливает значение обозначенной директивы. Может использоваться только с директивами типа **`PHP_INI_ALL`** и**`PHP_INI_PERDIR`** Для очистки предыдущих установленных значений используйте значение *none*.

Замечание: Не используйте `php_value` для установки логических значений. Вместо этого необходимо использовать`php_flag`(см. ниже).

`php_flag` `name on|off`

Используется для установки директивам логических значений. Может быть использовано только с директивами типа**`PHP_INI_ALL`** и **`PHP_INI_PERDIR`**.

`php_admin_value` `name value`

Устанавливает значение обозначенной директивы. *Не может быть использовано* в файлах `.htaccess`. Директивы любого типа, установленные с помощью `php_admin_value` не могут быть переопределены через `.htaccess` или ini_set(). Чтобы очистить предыдущее значение используйте значение *none*.

`php_admin_flag` `name on|off`

Используется для установки директивам логических значений. *Не может быть использовано* в файлах `.htaccess`. Директивы любого типа, установленные с помощью `php_admin_flag` не могут быть переопределены через `.htaccess` или ini_set().

Пример #1 Пример конфигурации Apache

```
<IfModule mod_php5.c>
  php_value include_path ".:/usr/local/lib/php"
  php_admin_flag engine on
</IfModule>
<IfModule mod_php4.c>
  php_value include_path ".:/usr/local/lib/php"
  php_admin_flag engine on
</IfModule>
```

Предостережение
PHP константы не доступны вне PHP. К примеру, в `httpd.conf` вы не можете использовать константы PHP такие как **E_ALL** или **E_NOTICE** чтобы установить директиву error_reporting, так как они не будут иметь значения и будут приравниваться к *0*. Используйте вместо этого соответствующие значения типа bitmask (битовая маска). Эти константы могут быть использованы в `php.ini`

Изменение конфигурации PHP через реестр Windows ¶

При использовании PHP в Windows значения конфигурации могут быть изменены на уровне директории посредством реестра Windows. Значения конфигурации хранятся в ключе реестра *HKLM\SOFTWARE\PHP\Per Directory Values*, в подключах, включающих полный путь. К примеру, значения конфигурации для директории *c:\inetpub\wwwroot* могут храниться в ключе*HKLM\SOFTWARE\PHP\Per Directory Values\c\inetpub\wwwroot*. Настройки для директории будут действительны для любых скриптов запущенных из этой директории или её поддиректории. Значения ключа должны иметь название конфигурационной директивы PHP и строковое значение. PHP константы в значениях игнорируются. Однако только значения конфигурации, изменяемые в **PHP_INI_USER,** могут быть установлены таким образом, значения же **PHP_INI_PERDIR** не могут.

Другие интерфейсы в PHP ¶

Независимо от того, как вы запускаете PHP, вы можете изменять некоторые значения во время выполнения ваших скриптов помощьюini_set(). Для более детальной информации смотрите документацию на странице функции ini_set().

Если вам интересен полный список конфигурационных настроек вашей системы с текущими значениями, то вы можете запуститьphpinfo() функцию, и получить результирующую страницу. Вы также можете получить доступ к значениям индивидуально сконфигурированных директив в процессе выполнения, используя ini_get() или get_cfg_var().

User Contributed Notes *13 notes*

СПРАВОЧНИК ЯЗЫКА ¶

Основы синтаксиса ¶

Содержание ¶

User Contributed Notes 9 notes

Теги PHP ¶

Когда PHP обрабатывает файл, он ищет открывающие и закрывающие теги, такие как *<?php* и *?>*, которые указывают PHP, когда начинать и заканчивать обработку кода между ними. Подобный способ обработки позволяет PHP внедряться во все виды различных документов, так как всё, что находится вне пары открывающих и закрывающих тегов, будет проигнорировано парсером PHP.

PHP также допускает короткий открывающий тег *<?*, однако использовать их нежелательно, так как они доступны только если включены с помощью конфигурационной директивы `php.ini` short_open_tag, либо если PHP был сконфигурирован с опцией --enable-short-tags .

Если файл содержит только код PHP, предпочтительно опустить закрывающий тег в конце файла. Это помогает избежать добавлени случайных символов пробела или перевода строки после закрывающего тега PHP, которые могут послужить причиной нежелательных эффектов, так как PHP начинает выводить данные в буфер при отсутствии намерения у программиста выводить какие-либо данные в этой точке скрипта.

```php
<?php
echo "Hello world";

// ... еще код

echo "Последнее выражение";

// Скрипт заканчивается тут без закрывающего тега PHP
```

User Contributed Notes 5 notes

Изолирование от HTML ¶

Все, что находится вне пары открывающегося и закрывающегося тегов, игнорируется интерпретатором PHP, у которого есть возможность обрабатывать файлы со смешанным содержимым. Это позволяет PHP-коду быть встроенным в документы HTML, к примеру, для создания шаблонов.

```
<p>Это будет проигнорировано PHP и отображено браузером.</p>
<?php echo 'А это будет обработано.'; ?>
<p>Это тоже будет проигнорировано PHP и отображено браузером.</p
>
```

Это работает так, как и ожидается, потому что когда интерпретатор PHP встречает закрывающие теги ?>, он просто начинает выводить все что найдет (за исключением сразу следующего символа перевода строки - смотрите раздел разделение инструкций) пока не встретит другой открывающий тег за исключением случая с содержащимся внутри кода условным оператором, в котором интерпретатор определяет результат условия перед принятием решения что пропустить. Ознакомьтесь со следующим примером.

Использование структур с условиями

Пример #1 Продвинутое изолирование с использованием условий

```
<?php if ($expression == true): ?>
  Это будет отображено, если выражение истинно.
<?php else: ?>
  В ином случае будет отображено это.
<?php endif; ?>
```

В этом примере PHP пропускает блоки, где условие не соблюдается. Даже несмотря на то, что они находятся вне пары открывающих/закрывающих тегов, PHP пропустит их в соответствии с условием, так как интерпретатор PHP будет перепрыгивать через блоки, содержащиеся внутри условия, которое не соблюдается.

При выводе больших блоков текста выход из режима синтаксического разбора PHP обычно более эффективен, чем отправка текста с помощью функций echo или print.

Существует четыре набора тегов, которые могут быть использованы для обозначения PHP-кода. Из них только два <?php ?> и <script language="php"> </script>) всегда доступны. Другими двумя являются короткие теги и теги в стиле ASP, которые могут быть включены или выключены в конфигурационном файле `php.ini`.

Хотя короткие теги и теги в стиле ASP могут быть удобны, они не так переносимы, как длинные версии, и поэтому не рекомендуются.

Замечание:
Кроме того, если вы намереваетесь вставлять PHP-код в XML или XHTML, чтобы соответствовать XML стандартам, вам следует использовать форму <?php ?>.

Пример #2 Открывающие и закрывающие теги PHP

```
1.  <?php echo 'если вы хотите хранить код PHP в документах XHTML или XML,
                 то используйте эти теги'; ?>

2.  <script language="php">
        echo 'некоторые редакторы (например, FrontPage) не
              любят инструкции обработки с этими тегами';
    </script>

3.  <? echo 'этот код с короткими тегами'; ?>
    Код с такими тегами <?= 'какой-
нибудь текст' ?> является сокращением от <? echo 'какой-
нибудь текст' ?>

4.  <% echo 'Также вы можете использовать теги в ASP стиле'; %>
    Код с такими тегами <%= $variable; %> является сокращением о
т <% echo $variable; %>
```

Несмотря на то, что теги указанные в первых двух примерах всегда доступны, наиболее широко используется (и рекомендуется) первый пример из этих двух.

Короткие теги (третий пример) доступны, только когда они включены с помощью директивы short_open_tag в конфигурационном файле `php.ini`, либо если PHP был скомпилирован с опцией --enable-short-tags .

ASP (четвертый пример) доступны, только когда они включены с помощью директивы asp_tags в конфигурационном файле `php.ini`.

Замечание:
Следует избегать использования коротких тегов при разработке приложений или библиотек, предназначенных для распространения или размещения на PHP-серверах, не находящихся под вашим контролем, так как короткие теги могут не поддерживаться на целевом сервере. Для создания переносимого, совместимого кода, не используйте короткие теги.
Замечание:
В PHP 5.2 и более ранних версиях парсер не позволял файлам содержать только один открытый тег *<?php*. Это было разрешено, начиная с версии PHP 5.3 при наличии одного или более пробела после открывающего тега.
Замечание:
Начиная с PHP 5.4 короткий тег echo *<?=* всегда распознается и действует, несмотря на значение опции short_open_tag.

User Contributed Notes 10 notes

Разделение инструкций ¶

Как в C или Perl, PHP требует окончания инструкций точкой запятой в конце каждой инструкции. Закрывающий тег блока PHP-кода автоматически применяет точку с запятой; т.е. нет необходимости ставить точку с запятой в конце последней строки блока с PHP-кодом. Закрывающий тег блока "поглотит" немедленно следующий за ним переход на новую строку, если таковой будет обнаружен.

```
<?php
    echo 'Это тест';
?>

<?php echo 'Это тест' ?>

<?php echo 'Мы опустили последний закрывающий тег';
```

Замечание:
Закрывающий тег PHP-блока в конце файла не является обязательным, и в некоторых случаях его опускание довольно полезно, например, при использовании include или require, так, что нежелательные пробелы не останутся в конце файла и вы все еще сможете добавить http-заголовки после подключения к ответу сервера. Это также удобно при использовании буферизации вывода, где также нежелательно иметь пробелы в конце частей ответа, сгенерированного подключаемыми файлами.

User Contributed Notes 3 notes

Комментарии ¶

PHP поддерживает комментарии в стиле 'C', 'C++' и оболочки Unix (стиль Perl). Например:

```
<?php
    echo "Это тест"; // Это однострочный комментарий в стиле c++
    /* Это многострочный комментарий
       еще одна строка комментария */
    echo "Это еще один тест";
    echo "Последний тест"; # Это комментарий в стиле оболочки Un
ix
?>
```

Однострочные комментарии идут только до конца строки или текущего блока PHP-кода, в зависимости от того, что идет перед ними. Это означает, что HTML-код после // ... ?> или # ... ?> БУДЕТ напечатан: ?> завершает режим PHP и возвращает режим HTML, а // или # не могут повлиять на это. Если включена директива asp_tags, то аналогичное поведение будет и с // %> и # %>. Однако тег </script> не завершает режим PHP в однострочном комментарии.

```
<h1>Это <?php # echo "простой";?> пример</h1>
<p>Заголовок вверху выведет 'Это пример'.</p>
```

'C'-комментарии заканчиваются при первой же обнаруженной последовательности */. Убедитесь, что вы не вкладываете друг в друга 'C'-комментарии. Очень легко допустить эту ошибку при комментировании большого блока кода.

```php
<?php
 /*
    echo "Это тест"; /* Этот комментарий вызовет проблему */
 */
?>
```

User Contributed Notes 14 notes

Типы ¶

Содержание ¶

Введение ¶

PHP поддерживает восемь простых типов.

Четыре скалярных типа:

- boolean
- integer
- float (число с плавающей точкой, также известное как double)
- string

Два смешанных типа:

- array
- object

И, наконец, два специальных типа:

- resource
- NULL

Для удобства понимания в этом руководстве используется также несколько псевдотипов:

- mixed
- number

- callback (он же callable)
- array|object
- void

И псевдопеременная `$...`.

Вы также можете найти несколько упоминаний типа двойной точности (double). Рассматривайте его как число с плавающей точкой, два имени существуют только по историческим причинам.

Как правило, программист не устанавливает тип переменной; обычно это делает PHP во время выполнения программы в зависимости от контекста, в котором используется переменная.

Замечание: Если вы желаете проверить тип и значение определённого выражения, используйте var_dump().
Если же вам для отладки необходимо просто удобочитаемое представление типа, используйте gettype(). Чтобы проверить на определенный тип, *не* используйте gettype(), применяйте для этого *is_type* функции. Вот несколько примеров:

```php
<?php
$a_bool = TRUE;   // логический
$a_str  = "foo";  // строковый
$a_str2 = 'foo';  // строковый
$an_int = 12;     // целочисленный

echo gettype($a_bool); // выводит: boolean
echo gettype($a_str);  // выводит: string

// Если это целое, увеличить на четыре
if (is_int($an_int)) {
    $an_int += 4;
}

// Если $a_bool - это строка, вывести ее
// (ничего не выводит)
if (is_string($a_bool)) {
    echo "Строка: $a_bool";
}
?>
```

Если вы хотите принудительно изменить тип переменной, вы можете либо привести переменную, либо использовать функциюsettype().

Обратите внимание, что переменная, в зависимости от ее типа в данный момент, в определённых ситуациях может иметь разные значения. Более подробную информацию смотрите в разделе Манипуляции с типами. Также вам, возможно, будет интересно посмотреть таблицы сравнения типов, поскольку в них приведены примеры различных сравнений, связанных с типами.

Булев ¶

Это простейший тип. boolean выражает истинность значения. Он может быть либо **TRUE**, либо **FALSE**.

Синтаксис ¶

Для указания boolean, используйте константы **TRUE** или **FALSE**. Обе они регистронезависимы.

```
<?php
$foo = True; // присвоить $foo значение TRUE
?>
```

Обычно, некоторый оператор возвращает boolean значение, которое потом передается управляющей конструкции.

```
<?php
// == это оператор, который проверяет
// эквивалентность и возвращает boolean
if ($action == "show_version") {
    echo "The version is 1.23";
}

// это необязательно...
if ($show_separators == TRUE) {
    echo "<hr>\n";
}

// ... потому что следующее имеет тот же самый смысл:
if ($show_separators) {
    echo "<hr>\n";
}
?>
```

Преобразование в булев тип ¶

Для явного преобразования в boolean, используйте *(bool)* или *(boolean)*. Однако в большинстве случаев приведение типа необязательно, так как значение будет автоматически преобразовано, если оператор, функция или управляющая конструкция требует boolean аргумент.

Смотрите также Манипуляции с типами.

При преобразовании в boolean, следующие значения рассматриваются как **FALSE**:

- само значение boolean **FALSE**
- integer 0 (ноль)
- float 0.0 (ноль)
- пустая строка, и строка "0"
- массив без элементов
- объект без полей (только для PHP 4)

- особый тип NULL (включая неустановленные переменные)
- Объекты SimpleXML, созданные из пустых тегов

Все остальные значения рассматриваются как **TRUE** (включая любой resource).

Внимание
-1 рассматривается как **TRUE**, как и любое другое ненулевое (отрицательное или положительное) число!

```
<?php
var_dump((bool) "");        // bool(false)
var_dump((bool) 1);         // bool(true)
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
?>
```

User Contributed Notes 19 notes

Целые числа ¶

Integer - это число из множества ℤ = {..., -2, -1, 0, 1, 2, ...}.

Смотрите также:

- Целые числа произвольной длины / GMP
- Числа с плавающей точкой
- Числа произвольной точности / BCMath

Синтаксис ¶

Целые числа могут быть указаны в десятичной (основание 10), шестнадцатеричной (основание 16), восьмеричной (основание 8) или двоичной (основание 2) системе счисления, с необязательным предшествующим знаком (- или +).

Двоичная запись integer доступна начиная с PHP 5.4.0.

Для записи в восьмеричной системе счисления, необходимо поставить пред числом *0* (ноль). Для записи в шестнадцатеричной системе счисления, необходимо поставить перед числом *0x*. Для записи в двоичной системе счисления, необходимо поставить перед числом *0b*

Пример #1 Целые числа

```
<?php
$a = 1234; // десятичное число
$a = -123; // отрицательное число
$a = 0123; // восьмеричное число (эквивалентно 83 в десятичной с
истеме)
$a = 0x1A; // шестнадцатеричное число (эквивалентно 26 в десятич
ной системе)
```

```
$a = 0b11111111; // двоичное число (эквивалентно 255 в десятичной системе)
?>
```

Формально, структуру целых чисел можно записать так:

```
десятичные          : [1-9][0-9]*
                    | 0

шестнадцатеричные : 0[xX][0-9a-fA-F]+

восьмеричные        : 0[0-7]+

двоичные            : 0b[01]+

целые               : [+-]?десятичные
                    | [+-]?шестнадцатеричные
                    | [+-]?восьмеричные
                    | [+-]?двоичные
```

Размер integer зависит от платформы, хотя, как правило, максимальное значение примерно равно 2 миллиардам (это 32-битное знаковое). 64-битные платформы обычно имеют максимальное значение около 9E18, кроме Windows, которая всегда 32-битная. PHP не поддерживает беззнаковые целые (integer). С версии PHP 4.4.0 и PHP 5.0.5 размер integer может быть определен с помощью константы **PHP_INT_SIZE**, а его максимальное значение - с помощью константы **PHP_INT_MAX**.

Внимание
Если в восьмеричном integer будет обнаружена неверная цифра (например, 8 или 9), оставшаяся часть числа будет проигнорирована.
Пример #2 Странности с восьмеричными числами

```
<?php
var_dump(01090); // 010 octal = 8 decimal
?>
```

Переполнение целых чисел ¶

Если PHP обнаружил, что число превышает размер типа integer, он будет интерпретировать его в качестве float. Аналогично, если результат операции лежит за границами типа integer, он будет преобразован в float.

Пример #3 Переполнение целых на 32-битных системах

```
<?php
$large_number = 2147483647;
var_dump($large_number);                     // int(2147483647)

$large_number = 2147483648;
var_dump($large_number);                     // float(2147483648
)

$million = 1000000;
```

```
$large_number =  50000 * $million;
var_dump($large_number);                    // float(50000000000)
?>
```

Пример #4 Переполнение целых на 64-битных системах

```
<?php
$large_number = 9223372036854775807;
var_dump($large_number);                    // int(9223372036854775807)

$large_number = 9223372036854775808;
var_dump($large_number);                    // float(9.223372036854E+18)

$million = 1000000;
$large_number =  50000000000000 * $million;
var_dump($large_number);                    // float(5.0E+19)
?>
```

В PHP не существует оператора деления целых чисел. Результатом *1/2* будет float *0.5*. Если привести значение к integer, оно будет округлено вниз. Для большего контроля над округлением используйте функцию round().

```
<?php
var_dump(25/7);         // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7));  // float(4)
?>
```

Преобразование в целое ¶

Для явного преобразования в integer, используйте приведение *(int)* или *(integer)*. Однако в большинстве случаев, в приведении типа нет необходимости, так как значение будет автоматически преобразовано, если оператор, функция или управляющая структура требует аргумент типа integer. Значение также может быть преобразовано в integer с помощью функции intval().

Если resource преобразуется в integer, то результатом будет уникальный номер ресурса, привязанный к resource во время исполнения PHP программы.

Смотрите также: Манипуляции с типами.

Из булевого типа ¶

FALSE преобразуется в *0* (ноль), а **TRUE** - в *1* (единицу).

Из чисел с плавающей точкой ¶

При преобразовании из float в integer, число будет округлено *в сторону нуля*.

Если число с плавающей точкой превышает размеры integer (обычно *+/- 2.15e+9 = 2^31* на 32-битных системах и *+/- 9.22e+18 = 2^63* на 64-битных системах, кроме Windows), результат будет неопределенным, так как float не имеет достаточной точности, чтобы вернуть верный результат. В этом случае не будет выведено ни предупреждения, ни даже замечания!

Внимание
Никогда не приводите неизвестную дробь к integer, так как это иногда может дать неожиданные результаты.

```
<?php
echo (int) ( (0.1+0.7) * 10 ); // выводит 7!
?>
```

Смотрите более подробно: предупреждение о точности чисел с плавающей точкой.

Из строк ¶

Смотрите Преобразование строк в числа

Из других типов ¶

Предостережение
Для других типов поведение преобразования в integer не определено. *Не полагайтесь* на любое наблюдаемое поведение, так как оно может измениться без предупреждения.

User Contributed Notes 21 notes

Числа с плавающей точкой ¶

Числа с плавающей точкой (также известные как "float", "double", или "real") могут быть определены следующими синтаксисами:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Формально:

```
LNUM          [0-9]+
DNUM          ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM [+-]?(({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

Размер числа с плавающей точкой зависит от платформы, хотя максимум, как правило составляет ~1.8e308 с точностью около 14 десятичных цифр (64-битный IEEE формат).

Внимание

Точность чисел с плавающей точкой

Числа с плавающей точкой имеют ограниченную точность. Хотя это зависит от операционной системы, в PHP обычно используется формат двойной точности IEEE 754, дающий максимальную относительную ошибку округления порядка 1.11e-16. Неэлементарные арифметические операции могут давать большие ошибки, и, разумеется, необходимо принимать во внимание распространение ошибок при совместном использовании нескольких операций.

Кроме того, рациональные числа, которые могут быть точно представлены в виде чисел с плавающей точкой с основанием 10, например, *0.1* или *0.7*, не имеют точного внутреннего представления в качестве чисел с плавающей точкой с основанием 2, вне зависимости от размера мантиссы. Поэтому они и не могут быть преобразованы в их внутреннюю двоичную форму без небольшой потери точности. Это может привести к неожиданным результатам: например, *floor((0.1+0.7)*10)* скорее всего вернет *7* вместо ожидаемого *8*, так как результат внутреннего представления будет чем-то вроде *7.9999999999999991118...*.

Так что никогда не доверяйте точности чисел с плавающей точкой до последней цифры, и не проверяйте напрямую их равенство. Если вам действительно необходима высокая точность, используйте математические функции произвольной точности и gmp-функции.

"Простое" объяснение можно найти в » руководстве по числам с плавающей точкой, которое также называется "Why don't my numbers add up?" ("Почему мои числа не складываются?")

Преобразование в число с плавающей точкой ¶

Информацию о преобразовании строк в числа с плавающей точкой смотрите в разделе Преобразование строк в числа. Для значений других типов преобразование будет сначала осуществлено в в integer и затем в число с плавающей точкой. Дополнительную информацию смотрите в разделе Преобразование к целому. Начиная с версии PHP 5, при преобразовании объекта к числу с плавающей точкой выводится замечание об ошибке.

Сравнение чисел с плавающей точкой ¶

Как указано выше, проверять числа с плавающей точкой на равенство проблематично из-за их внутреннего представления. Тем не менее, существуют способы для их сравнения, которые работают несмотря на все эти ограничения.

Для сравнения чисел с плавающей точкой используется верхняя граница относительной ошибки при округлении. Эта величина называется машинной эпсилон или единица округления(unit roundoff) и представляет собой самую маленькую допустимую разницу при расчетах.

`$a` и `$b` равны до 5-ти знаков после запятой.

```
<?php
$a = 1.23456789;
```

```php
$b = 1.23456780;
$epsilon = 0.00001;

if(abs($a-$b) < $epsilon) {
    echo "true";
}
?>
```

NaN ¶

Некоторые числовые операции могут возвращать значение, представляемое константой **NAN**. Данный результат означает неопределенное или непредставимое значение в операциях с плавающей точкой. Любое строгое или нестрогое сравнение данного значения с другим значением, включая его самого, возвратит **FALSE**.

Так как **NAN** представляет собой неограниченное количество различных значений, то **NAN** не следует сравнивать с другими значениями, включая ее саму. Вместо этого, для определения ее наличия необходимо использовать функцию is_nan().

User Contributed Notes 30 notes

Строки ¶

Строка - это набор символов, где символ - это то же самое, что и байт. Это значит, что PHP поддерживает ровно 256 различных символов, а также то, что в PHP нет встроенной поддержки Unicode. Смотрите также подробности реализации строкового типа.

Замечание: Строки (string) не могут быть размером более 2 Гб (2147483647 байт).

Синтаксис ¶

Строка может быть определена четырьмя различными способами:

- одинарными кавычками
- двойными кавычками
- heredoc-синтаксисом
- nowdoc-синтаксисом (начиная с версии PHP 5.3.0)

Одинарные кавычки ¶

Простейший способ определить строку - это заключить ее в одинарные кавычки (символ ').

Чтобы использовать одинарную кавычку внутри строки, проэкранируйте ее обратной косой чертой (\). Если необходимо написать саму обратную косую черту, продублируйте ее (\\). Все остальные случаи применения обратной косой черты будут интерпретированы как обычные символы: это означает, что если вы попытаетесь использовать другие управляющие последовательности, такие как \r или \n, они будут выведены как есть вместо какого-либо особого поведения.

Замечание: В отличие от синтаксиса двойных кавычек и heredoc, переменные и управляющие последовательности для специальных символов, заключенных в одинарные кавычки, *не* обрабатываются.

```
<?php
echo 'это простая строка';

echo 'Также вы можете вставлять в строки
символ новой строки вот так,
это нормально';

// Выводит: Однажды Арнольд сказал: "I'll be back"
echo 'Однажды Арнольд сказал: "I\'ll be back"';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\\*.*?';

// Выводит: Вы удалили C:\*.*?
echo 'Вы удалили C:\*.*?';

// Выводит: Это не будет развернуто: \n новая строка
echo 'Это не будет развернуто: \n новая строка';

// Выводит: Переменные $expand также $either не разворачиваются
echo 'Переменные $expand также $either не разворачиваются';
?>
```

Двойные кавычки ¶

Если строка заключена в двойные кавычки ("), PHP распознает большее количество управляющих последовательностей для специальных символов:

Управляющие последовательности	
Последовательность	**Значение**
\n	новая строка (LF или 0x0A (10) в ASCII)
\r	возврат каретки (CR или 0x0D (13) в ASCII)
\t	горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\v	вертикальная табуляция (VT или 0x0B (11) в ASCII) (с версии PHP 5.2.5)
\e	escape-знак (ESC или 0x1B (27) в ASCII) (с версии PHP 5.4.4)
\f	подача страницы (FF или 0x0C (12) в ASCII) (с версии PHP 5.2.5)
**	обратная косая черта
\$	знак доллара
\"	двойная кавычка
\[0-7]{1,3}	последовательность символов, соответствующая регулярному выражению символа в восьмеричной системе счисления
\x[0-9A-Fa-f]{1,2}	последовательность символов, соответствующая регулярному выражению символа в шестнадцатеричной системе счисления

Как и в строке, заключенной в одинарные кавычки, экранирование любого символа выведет также и саму обратную косую черту. До версии PHP 5.1.1, обратная косая черта в *\{$var}* не печаталась.

Но самым важным свойством строк в двойных кавычках является обработка переменных. Смотрите более подробно: обработка строк

Heredoc ¶

Третий способ определения строк - это использование heredoc-синтаксиса: <<<. После этого оператора необходимо указать идентификатор, затем перевод строки. После этого идет сама строка, а потом этот же идентификатор, закрывающий вставку.

Строка *должна* начинаться с закрывающего идентификатора, т.е. он должен стоять в первом столбце строки. Кроме того, идентификатор должен соответствовать тем же правилам именования, что и все остальные метки в PHP: содержать только буквенно-цифровые символы и знак подчеркивания, и не должен начинаться с цифры (знак подчеркивания разрешается).

Внимание

Очень важно отметить, что строка с закрывающим идентификатором не должна содержать других символов, за исключением точки с запятой (*;*). Это означает, что идентификатор *не должен вводиться с отступом* и что не может быть никаких пробелов или знаков табуляции до или после точки с запятой. Важно также понимать, что первым символом перед закрывающим идентификатором должен быть символ новой строки, определенный в вашей операционной системе. Например, в UNIX системах, включая Mac OS X, это *\n*. После закрывающего идентификатора также сразу должна начинаться новая строка.

Если это правило нарушено и закрывающий идентификатор не является "чистым", считается, что закрывающий идентификатор отсутствует и PHP продолжит его поиск дальше. Если в этом случае верный закрывающий идентификатор так и не будет найден, то это вызовет ошибку парсинга с номером строки в конце скрипта.

Heredoc не может быть использован для инициализации полей класса. Начиная с версии PHP 5.3, это ограничение распространяется только на heredoc, содержащие внутри себя переменные.

Пример #1 Неверный пример

```
<?php
class foo {
    public $bar = <<<EOT
bar
    EOT;
}
?>
```

Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что вам нет необходимости экранировать кавычки в heredoc, но вы по-прежнему можете использовать вышеперечисленные управляющие последовательности. Переменные обрабатываются, но с применением сложных переменных внутри heredoc нужно быть также внимательным, как и при работе со строками.

Пример #2 Пример определения heredoc-строки

```
<?php
$str = <<<EOD
Пример строки,
```

```
охватывающей несколько строчек,
с использованием heredoc-синтаксиса.
EOD;

/* Более сложный пример с переменными. */
class foo
{
    var $foo;
    var $bar;

    function foo()
    {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
$name = 'МоеИмя';

echo <<<EOT
Меня зовут "$name". Я печатаю $foo->foo.
Теперь я вывожу {$foo->bar[1]}.
Это должно вывести заглавную букву 'A': \x41
EOT;
?>
```

Результат выполнения данного примера:

```
Меня зовут "МоеИмя". Я печатаю Foo.
Теперь, я вывожу Bar2.
Это должно вывести заглавную букву 'A': A
```

Также возможно использовать heredoc-синтаксис для передачи данных через аргументы функции:

Пример #3 Пример применения heredoc в аргументах

```
<?php
var_dump(array(<<<EOD
foobar!
EOD
));
?>
```

Начиная с версии 5.3.0, стала возможной инициализация статических переменных и свойств/констант класса с помощью синтаксиса heredoc:

Пример #4 Использование heredoc для инициализации статических переменных

```
<?php
// Статические переменные
function foo()
```

```
{
    static $bar = <<<LABEL
Здесь ничего нет...
LABEL;
}

// Class properties/constants
class foo
{
    const BAR = <<<FOOBAR
Пример использования константы
FOOBAR;

    public $baz = <<<FOOBAR
Пример использования поля
FOOBAR;
}
?>
```

Начиная с версии PHP 5.3.0 можно также окружать идентификатор Heredoc двойными кавычками:

Пример #5 Использование двойных кавычек в heredoc

```
<?php
echo <<<"FOOBAR"
Привет, мир!
FOOBAR;
?>
```

Nowdoc ¶

Nowdoc - это то же самое для строк в одинарных кавычках, что и heredoc для строк в двойных кавычках. Nowdoc похож на heredoc, но внутри него *не осуществляется никаких подстановок*. Эта конструкция идеальна для внедрения PHP-кода или других больших блоков текста без необходимости его экранирования. В этом он немного похож на SGML-конструкцию *<![CDATA[ ]]>* тем, что объявляет блок текста, не предназначенный для обработки.

Nowdoc указывается той же последовательностью <<<, что используется в heredoc, но последующий за ней идентификатор заключается в одинарные кавычки, например, *<<<'EOT'*. Все условия, действующие для heredoc идентификаторов также действительны и для nowdoc, особенно те, что относятся к закрывающему идентификатору.

Пример #6 Пример использования nowdoc

```
<?php
$str = <<<'EOD'
Пример текста,
занимающего несколько строк,
с помощью синтаксиса nowdoc.
EOD;
```

```php
/* Более сложный пример с переменными. */
class foo
{
    public $foo;
    public $bar;

    function foo()
    {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
$name = 'МоеИмя';

echo <<<'EOT'
Меня зовут "$name". Я печатаю $foo->foo.
Теперь я печатаю {$foo->bar[1]}.
Это не должно вывести заглавную 'A': \x41
EOT;
?>
```

Результат выполнения данного примера:

```
Меня зовут "$name". Я печатаю $foo->foo.
Теперь я печатаю {$foo->bar[1]}.
Это не должно вывести заглавную 'A': \x41
```

Замечание:
В отличие от heredoc, nowdoc может быть использован в любом контексте со статическими данными. Типичный пример инициализации полей класса или констант:

Пример #7 Пример использования статичных данных

```php
<?php
class foo {
    public $bar = <<<'EOT'
bar
EOT;
}
?>
```

Замечание:
Поддержка nowdoc была добавлена в PHP 5.3.0.

Обработка переменных ¶

Если строка указывается в двойных кавычках, либо при помощи heredoc, переменные внутри нее обрабатываются.
Существует два типа синтаксиса: простой и сложный. Простой синтаксис более легок и удобен. Он дает возможность обработки переменной, значения массива (array) или свойства объекта (object) с минимумом усилий.
Сложный синтаксис может быть определен по фигурным скобкам, окружающим выражение.

Простой синтаксис

Если интерпретатор встречает знак доллара (*$*), он захватывает так много символов, сколько возможно, чтобы сформировать правильное имя переменной. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки.

```
<?php
$juice = "apple";

echo "He drank some $juice juice.".PHP_EOL;
// не работает, 's' - это верный символ для имени переменной,
// но наша переменная имеет имя $juice.
echo "He drank some juice made of $juices.";
?>
```

Результат выполнения данного примера:

```
He drank some apple juice.
He drank some juice made of .
```

Аналогично могут быть обработаны элемент массива (array) или свойство объекта (object). В индексах массива закрывающая квадратная скобка (*]*) обозначает конец определения индекса. Для свойств объекта применяются те же правила, что и для простых переменных.

Пример #8 Пример простого синтаксиса

```
<?php
$juices = array("apple", "orange", "koolaid1" => "purple");

echo "He drank some $juices[0] juice.".PHP_EOL;
echo "He drank some $juices[1] juice.".PHP_EOL;
echo "He drank some $juices[koolaid1] juice.".PHP_EOL;

class people {
    public $john = "John Smith";
    public $jane = "Jane Smith";
    public $robert = "Robert Paulsen";

    public $smith = "Smith";
}

$people = new people();

echo "$people->john drank some $juices[0] juice.".PHP_EOL;
echo "$people->john then said hello to $people->jane.".PHP_EOL;
echo "$people->john's wife greeted $people->robert.".PHP_EOL;
echo "$people->robert greeted the two $people-
>smiths."; // Won't work
?>
```

Результат выполнения данного примера:

```
He drank some apple juice.
He drank some orange juice.
```

```
He drank some purple juice.
John Smith drank some apple juice.
John Smith then said hello to Jane Smith.
John Smith's wife greeted Robert Paulsen.
Robert Paulsen greeted the two .
```
Для чего-либо более сложного, используйте сложный синтаксис.

Сложный (фигурный) синтаксис

Он называется сложным не потому, что труден в понимании, а потому что позволяет использовать сложные выражения.
Любая скалярная переменная, элемент массива или свойство объекта, отображаемое в строку, может быть представлена в строке этим синтаксисом. Просто запишите выражение так же, как и вне строки, а затем заключите его в *{* и *}*. Поскольку *{* не может быть экранирован, этот синтаксис будет распознаваться только когда *$* следует непосредственно за *{*. Используйте *{\$*, чтобы напечатать *{$*. Несколько поясняющих примеров:

```php
<?php
// Показываем все ошибки
error_reporting(E_ALL);

$great = 'здорово';

// Не работает, выводит: Это { здорово}
echo "Это { $great}";

// Работает, выводит: Это здорово
echo "Это {$great}";
echo "Это ${great}";

// Работает
echo "Этот квадрат шириной {$square->width}00 сантиметров.";

// Работает, ключи, заключенные в кавычки, работают только с син
таксисом фигурных скобок
echo "Это работает: {$arr['key']}";

// Работает
echo "Это работает: {$arr[4][3]}";

// Это неверно по той же причине, что и $foo[bar] вне
// строки. Другими словами, это по-прежнему будет работать,
// но поскольку PHP сначала ищет константу foo, это вызовет
// ошибку уровня E_NOTICE (неопределенная константа).
echo "Это неправильно: {$arr[foo][3]}";

// Работает. При использовании многомерных массивов внутри
// строк всегда используйте фигурные скобки
echo "Это работает: {$arr['foo'][3]}";

// Работает.
echo "Это работает: " . $arr['foo'][3];

echo "Это тоже работает: {$obj->values[3]->name}";
```

```php
echo "Это значение переменной по имени $name: {${$name}}";

echo "Это значение переменной по имени, которое возвращает функц
ия getName(): {${getName()}}";

echo "Это значение переменной по имени, которое возвращает \$obj
ect->getName(): {${$object->getName()}}";

// Не работает, выводит: Это то, что возвращает getName(): {getN
ame()}
echo "Это то, что возвращает getName(): {getName()}";
?>
```

С помощью этого синтаксиса также возможен доступ к свойствам объекта внутри строк.

```php
<?php
class foo {
    var $bar = 'I am bar.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo "{$foo->$bar}\n";
echo "{$foo->$baz[1]}\n";
?>
```

Результат выполнения данного примера:

```
I am bar.
I am bar.
```

Замечание:
Функции, вызовы методов, статические переменные классов, а также константы классов работает внутри *{$}*, начиная с версии PHP 5. Однако указываемое значение будет обработано как имя переменной в том же контексте, что и строка, в которой она определяется. Использование одинарных фигурных скобок (*{}*) не будет работать для доступа к значениям функций, методов, констант классов или статических переменных класса.

```php
<?php
// Показываем все ошибки
error_reporting(E_ALL);

class beers {
    const softdrink = 'rootbeer';
    public static $ale = 'ipa';
}

$rootbeer = 'A & W';
$ipa = 'Alexander Keith\'s';

// Это работает, выводит: Я бы хотел A & W
```

```php
echo "Я бы хотел {${beers::softdrink}}\n";

// Это тоже работает, выводит: Я бы хотел Alexander Keith's
echo "Я бы хотел {${beers::$ale}}\n";
?>
```

Доступ к символу в строке и его изменение ¶

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в квадратных скобках после строки, например, `$str[42]`. Думайте о строке для этой цели, как о массиве символов. Если нужно получить или заменить более 1 символа, можно использовать функции substr() и substr_replace().

Замечание: К символу в строке также можно обращаться с помощью фигурных скобок, например, `$str{42}`.

Внимание

Попытка записи в смещение за границами строки дополнит строку пробелами до этого смещения. Нецелые типы будет преобразованы в целые. Неверный тип смещения вызовет ошибку уровня **E_NOTICE**. Запись по отрицательному смещению вызовет ошибку уровня **E_NOTICE**, а при чтении вернет пустую строку. Используется только первый символ присваемой строки. Присвоение пустой строки присваивает нулевой байт (NULL).

Внимание

Строки в PHP внутренне представляют из себя массивы байт. Как результат, доступ или изменение строки по смещению небезопасно с точки зрения многобайтной кодировки, и должно выполняться только со строками в однобайтных кодировках, таких как, например, ISO-8859-1.

Пример #9 Несколько примеров строк

```php
<?php
// Получение первого символа строки
$str = 'This is a test.';
$first = $str[0];

// Получение третьего символа строки
$third = $str[2];

// Получение последнего символа строки
$str = 'This is still a test.';
$last = $str[strlen($str)-1];

// Изменение последнего символа строки
$str = 'Look at the sea';
$str[strlen($str)-1] = 'e';

?>
```

Начиная с PHP 5.4 смещение в строке должно задаваться либо целым числом либо строкой, содержащей цифры, иначе будет выдаваться предупреждение. Ранее смещение, заданное строкой вида *"foo"*, без предупреждений преобразовывалось в *0*.

Пример #10 Различия между PHP 5.3 и PHP 5.4

```php
<?php
$str = 'abc';

var_dump($str['1']);
var_dump(isset($str['1']));

var_dump($str['1.0']);
var_dump(isset($str['1.0']));

var_dump($str['x']);
var_dump(isset($str['x']));

var_dump($str['1x']);
var_dump(isset($str['1x']));
?>
```

Результат выполнения данного примера в PHP 5.3:

```
string(1) "b"
bool(true)
string(1) "b"
bool(true)
string(1) "a"
bool(true)
string(1) "b"
bool(true)
```

Результат выполнения данного примера в PHP 5.4:

```
string(1) "b"
bool(true)

Warning: Illegal string offset '1.0' in /tmp/t.php on line 7
string(1) "b"
bool(false)

Warning: Illegal string offset 'x' in /tmp/t.php on line 9
string(1) "a"
bool(false)
string(1) "b"
bool(false)
```

Замечание:
Попытка доступа к переменным других типов (исключая массивы или объекты, реализующие определенные интерфейсы) с помощью *[]* или *{}* молча вернет **NULL**.

Замечание:
В PHP 5.5 была добавлена поддержка доступа к символам в строковых литералах с помощью синтаксиса *[]* или *{}*.

Полезные функции и операторы ¶

Строки могут быть объединены при помощи оператора '.' (точка). Обратите внимание, оператор сложения '+' *здесь не работает*. Дополнительную информацию смотрите в разделе Строковые операторы.

Для модификации строк существует множество полезных функций.

Основные функции описаны в разделе строковых функций, а для расширенного поиска и замены - функции регулярных выраженийили Perl-совместимых регулярных выражений.
Также существуют функции для работы с URL, и функции шифрования/дешифрования строк (mcrypt и mhash).
Наконец, смотрите также функции символьных типов.

Преобразование в строку ¶

Значение может быть преобразовано в строку, с помощью приведения *(string)*, либо функции strval(). В выражениях, где необходима строка, преобразование происходит автоматически. Это происходит, когда вы используете функции echo или print, либо когда значение переменной сравнивается со строкой. Прочтение разделов руководства Типы и Манипуляции с типами сделает следующее более понятным. Смотрите также settype().

Значение boolean **TRUE** преобразуется в строку *"1"*, а значение **FALSE** преобразуется в *""* (пустую строку). Это позволяет преобразовывать значения в обе стороны - из булева типа в строковый и наоборот.

Целое (integer) или число с плавающей точкой (float) преобразуется в строку, представленную числом, состоящим из его цифр (включая показатель степени для чисел с плавающей точкой). Числа с плавающей точкой могут быть преобразованы с помощью экспоненциального представления (*4.1E+6*).

Замечание:
Символ десятичной точки определяется из настроек локали текущего скрипта (категория LC_NUMERIC). Смотрите также setlocale().

Массивы всегда преобразуются в строку *"Array"*, так что вы не можете отобразить содержимое массива (array), используя echo илиprint, чтобы узнать, что он содержит. Чтобы просмотреть отдельный элемент, используйте что-нибудь вроде *echo $arr['foo']*. Смотрите ниже советы о том, как отобразить/просмотреть все содержимое.

Объекты в PHP 4 всегда преобразовывались в строку *"Object"*. Если вы хотите вывести значения полей объекта (object) с целью отладки, читайте дальше. Если вы хотите получить имя класса требуемого объекта, используйте get_class(). Начиная с PHP 5, также стал доступен метод __toString.

Resource всегда всегда преобразуется в string вида *"Resource id #1"*, где *1* является номером ресурса привязанного к resource во время выполнения. И хотя не стоит точно полагаться на эту строку, которая может быть изменена в будущем, она всегда будет уникальной для текущего запуска скрипта (т.е. web-

запроса или CLI-процесса) и не может использоваться повторно для другого ресурса. Если вы хотите получить тип ресурса, используйте get_resource_type().

NULL всегда преобразуется в пустую строку.

Как вы могли видеть выше, прямое преобразование в строку массивов, объектов или ресурсов не дает никакой полезной информации о самих значениях, кроме их типов. Более подходящий способ вывода значений для отладки - использовать функцииprint_r() и var_dump().

Большинство значений в PHP может быть преобразовано в строку для постоянного хранения. Этот метод называется сериализацией и может быть выполнен при помощи функции serialize(). Кроме того, если в вашей установке PHP есть поддержка WDDX, возможна также сериализация в XML-структуру.

Преобразование строк в числа ¶

Если строка распознается как числовое значение, результирующее значение и тип определяется так, как показано далее.
Если строка не содержит какой-либо из символов '.', 'e', или 'E', и значение числа помещается в пределы целых чисел (определенных**PHP_INT_MAX**), строка будет распознана как целое число (integer). Во всех остальных случаях она считается числом с плавающей точкой (float).

Значение определяется по начальной части строки. Если строка начинается с верного числового значения, будет использовано это значение. Иначе значением будет 0 (ноль). Верное числовое значение - это одна или более цифр (могущих содержать десятичную точку), по желанию предваренных знаком, с последующим необязательным показателем степени. Показатель степени - это 'e' или 'E' с последующими одной или более цифрами.

```
<?php
$foo = 1 + "10.5";                // $foo это float (11.5)
$foo = 1 + "-1.3e3";              // $foo это float (-1299)
$foo = 1 + "bob-1.3e3";           // $foo это integer (1)
$foo = 1 + "bob3";                // $foo это integer (1)
$foo = 1 + "10 Small Pigs";       // $foo это integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo это float (14.2)
$foo = "10.0 pigs " + 1;          // $foo это float (11)
$foo = "10.0 pigs " + 1.0;        // $foo это float (11)
?>
```

Более подробную информацию об этом преобразовании смотрите в разделе о strtod(3) документации Unix.

Если вы хотите протестировать любой из примеров этого раздела, скопируйте и вставьте его и следующую строку, чтобы увидеть, что происходит:

```
<?php
echo "\$foo==$foo; тип: " . gettype ($foo) . "<br />\n";
?>
```

Не ожидайте получить код символа, преобразовав его в целое (как это делается, например, в C). Для преобразования символов в их ASCII коды и обратно используйте функции ord() и chr().

Подробности реализации строкового типа ¶

Строковый тип (string) в PHP реализован в виде массива байт и целого числа, содержащего длину буфера. Он не содержит никакой информации о способе преобразования этих байт в символы, предоставляя эту задачу программисту. Нет никаких ограничений на содержимое строки, например, байт со значением *0* ("NUL"-байт) может располагаться где угодно (однако стоит учитывать, что некоторые функции, как сказано в этом руководстве, не являются "бинарно-безопасными", т.е. они могут передавать строки библиотекам, которые игнорируют данные после NUL-байта).

Данная природа строкового типа объясняет почему в PHP нет отдельного типа “byte” - строки играют эту роль. Функции, возвращающие нетекстовые данные - например, произвольный поток данных, считываемый из сетевого сокета - тем не менее возвращают строки.

Принимая во внимание тот факт, что PHP не диктует определенную кодировку для строк, можно задать вопрос, как в таком случае кодируются строковые литералы. Например, строка *"á"* эквивалентна *"\xE1"* (ISO-8859-1), *"\xC3\xA1"* (UTF-8, форма нормализации C),*"\x61\xCC\x81"* (UTF-8, форма нормализации D) или какому-либо другому возможному представлению? Ответом является следующее: строка будет закодирована тем образом, которым она записана в файле скрипта. Таким образом, если скрипт записан в кодировке ISO-8859-1, то и строка будет закодирована в ISO-8859-1 и т.д. Однако это правило не применяется при включенном режиме Zend Multibyte: в этом случае скрипт может быть записан в любой кодировке (которая указывается ясно или определяется автоматически), а затем конвертируются в определенную внутреннюю кодировку, которая и будет впоследствии использована для строковых литералов. Учтите, что на кодировку скрипта (или на внутреннюю кодировку, если включен режим Zend Multibyte) накладываются некоторые ограничения: практически всегда данная кодировка должна быть надмножеством ASCII, например, UTF-8 или ISO-8859-1. Учтите также, что кодировки, зависящие от состояния, где одни и те же значения байт могут быть использованы в начальном и не начальном состоянии сдвига (initial and non-inital shift state), могут вызвать проблемы.

Разумеется, чтобы приносить пользу, строковые функции должны сделать некоторые предположения о кодировке строки. К несчастью, среди PHP-функций довольно большое разнообразие подходов к этому вопросу:

- Некоторые функции предполагают, что строка закодирована в какой-либо однобайтовой кодировке, однако для корректной работы им не требуется интерпретировать байты как определенные символы. Под эту категорию попадают, например, substr(),strpos(), strlen() и strcmp(). Другой способ мышления об этих функциях представляет собой оперирование буферами памяти, т.е. они работают непосредственно с байтами и их смещениями. offsets.
- Другие функции ожидают передачу кодировку в виде параметра, возможно, предполагая некоторую кодировку по умолчанию, если параметр с кодиров-

кой не был указан. Такой функцией является htmlentities() и большинство функций из расширенияmbstring.
- Другие функции используют текущие установки локали (см. setlocale()), но оперируют побайтово). В эту категорию попадаютstrcasecmp(), strtoupper() и ucfirst(). Это означает, что они могут быть использованы только с однобайтовыми кодировками, в том случае, когда кодировка совпадает с локалью. Например, *strtoupper("á")* может вернуть *"Á"*, если локаль установлена корректно и буква *á* закодирована в виде одного байта. Если она закодирована в UTF-8, будет возвращен некорректный результат, и, в зависимости от текущей локали, результирующая строка может быть (или не быть) испорчена.
- Наконец, есть функции, подразумевающие, что строка использует определенную кодировку, обычно UTF-8. Сюда попадают большинство функций из расширений intl и PCRE (в последнем случае, только при указании модификатора *u*). Хотя это и сделано специально, функция utf8_decode() подразумевает кодировку UTF-8, а utf8_encode() - ISO-8859-1.

В конечном счете, написание корректных программ, работающих с Unicode, означает осторожное избегание функций, которые не работают с Unicode и, скорее всего, испортят данные, и использование вместо них корректных функций, обычно из расширений intl иmbstring. Однако использование функций, способных работать с Unicode, является самым началом. Вне зависимости от тех функций, которые предоставляет язык, необходимо знать спецификацию самого Unicode. Например, если программа предполагает существование в языке только строчных и заглавных букв, то она делает большую ошибку.

User Contributed Notes 47 notes

Массивы ¶

На самом деле массив в PHP - это упорядоченное отображение, которое устанавливает соответствие между *значением* и *ключом*. Этот тип оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно массив, список (вектор), хэш-таблицу (являющуюся реализацией карты), словарь, коллекцию, стек, очередь и, возможно, что-то еще. Так как значением массива может быть другой массив PHP, можно также создавать деревья и многомерные массивы.

Объяснение этих структур данных выходит за рамки данного справочного руководства, но вы найдете как минимум один пример по каждой из них. За дополнительной информацией вы можете обратиться к соответствующей литературе по этой обширной теме.

Синтаксис ¶

Определение при помощи array() ¶

Массив (тип array) может быть создан языковой конструкцией array(). language construct. В качестве параметров она принимает любое количество разделенных запятыми пар *key => value* (ключ => значение).

```
array(
```

```
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

Запятая после последнего элемента массива необязательна и может быть опущена. Обычно это делается для однострочных массивов, т.е. *array(1, 2)* предпочтительней *array(1, 2,)*. Для многострочных массивов с другой стороны обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива.

Начиная с PHP 5.4 возможно использовать короткий синтаксис определения массивов, который заменяет языковую конструкцию*array()* на *[]*.

Пример #1 Простой массив

```php
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Начиная с PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

key может быть либо типа integer, либо типа string. value может быть любого типа.

Дополнительно с ключом key будут сделаны следующие преобразования:

- Строки, содержащие целое число будут преобразованы к типу integer. Например, ключ со значением *"8"* будет в действительности сохранен со значением *8*. С другой стороны, значение *"08"* не будет преобразовано, так как оно не является корректным десятичным целым.
- Числа с плавающей точкой (тип float) также будут преобразованы к типу integer, т.е. дробная часть будет отброшена. Например, ключ со значением *8.7* будет в действительности сохранен со значением *8*.
- Тип bool также преобразовываются к типу integer. Например, ключ со значением *true* будет сохранен со значением *1* и ключ со значением *false* будет сохранен со значением *0*.
- Тип null будет преобразован к пустой строке. Например, ключ со значением *null* будет в действительности сохранен со значением*""*.
- Массивы (тип array) и объекты (тип object) *не могут* использоваться в качестве ключей. При подобном использовании будет генерироваться предупреждение: *Недопустимый тип смещения (Illegal offset type)*.

Если несколько элементов в объявлении массива используют одинаковый ключ, то только последний будет использоваться, а все другие будут перезаписаны.

Пример #2 Пример преобразования типов и перезаписи элементов

```
<?php
$array = array(
    1    => "a",
    "1"  => "b",
    1.5  => "c",
    true => "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(1) {
  [1]=>
  string(1) "d"
}
```

Так как все ключи в вышеприведенном примере преобразуются к *1*, значение будет перезаписано на каждый новый элемент и останется только последнее присвоенное значение *"d"*.

Массивы в PHP могут содержать ключи типов integer и string одновременно, так как PHP не делает различия между индексированными и ассоциативными массивами.

Пример #3 Смешанные ключи типов integer и string

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Параметр key является необязательным. Если он не указан, PHP будет использовать предыдущее наибольшее значение ключа типаinteger, увеличенное на 1.

Пример #4 Индексированные массивы без ключа

```php
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hello"
  [3]=>
  string(5) "world"
}
```

Возможно указать ключ только для некоторых элементов и пропустить для других:

Пример #5 Ключи для некоторых элементов

```php
<?php
$array = array(
         "a",
         "b",
     6 => "c",
         "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Как вы видите последнее значение *"d"* было присвоено ключу *7*. Это произошло потому, что самое большое значение ключа целого типа перед этим было *6*.

Доступ к элементам массива с помощью квадратных скобок ¶

Доступ к элементам массива может быть осуществлен с помощью синтаксиса *array[key]*.

Пример #6 Доступ к элементам массива

```
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

Результат выполнения данного примера:

```
string(3) "bar"
int(24)
string(3) "foo"
```

Замечание:
Для доступа к элементам массива могут использоваться как квадратные, так и фигурные скобки (например, *$array[42]* и *$array{42}*означают одно и то же в вышеприведенном примере).

С PHP 5.4 стало возможным прямое разыменование массива, возвращаемого в качестве результата вызова функции или метода. Раньше приходилось использовать временные переменные.

С PHP 5.5 стало возможным прямое разыменование элементов у литерала массива.

Пример #7 Разыменование массива

```
<?php
function getArray() {
    return array(1, 2, 3);
}

// в PHP 5.4
$secondElement = getArray()[1];

// ранее делали так
$tmp = getArray();
$secondElement = $tmp[1];
```

```php
// или так
list(, $secondElement) = getArray();
?>
```

Замечание:
Попытка доступа к неопределенному ключу в массиве - это то же самое, что и попытка доступа к любой другой неопределенной переменной: будет сгенерирована ошибка уровня **E_NOTICE**, и результат будет **NULL**.

Создание/модификация с помощью синтаксиса квадратных скобок ¶

Существующий массив может быть изменен явной установкой значений в нем.

Это выполняется присвоением значений массиву array с указанием в скобках ключа. Кроме того, вы можете опустить ключ. В этом случае добавьте к имени переменной пустую пару скобок (*[]*).

```php
$arr[key] = value;
$arr[] = value;
// key может быть integer или string
// value может быть любым значением любого типа
```

Если массив `$arr` еще не существует, он будет создан. Таким образом, это еще один способ определить массив array. Однако такой способ применять не рекомендуется, так как если переменная `$arr` уже содержит некоторое значение (например, значение типаstring из переменной запроса), то это значение останется на месте и *[]* может на самом деле означать доступ к символу в строке. Лучше инициализировать переменную путем явного присваивания значения.

Для изменения определенного значения просто присвойте новое значение элементу, используя его ключ. Если вы хотите удалить пару ключ/значение, вам необходимо использовать функцию unset().

```php
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // В этом месте скрипта это
                // то же самое, что и $arr[13] = 56;

$arr["x"] = 42; // Это добавляет к массиву новый
                // элемент с ключом "x"

unset($arr[5]); // Это удаляет элемент из массива

unset($arr);    // Это удаляет массив полностью
?>
```

Замечание:
Как уже говорилось выше, если ключ не был указан, то будет взят максимальный из существующих целочисленных (integer) индексов, и новым ключом будет это максимальное значение (в крайнем случае 0) плюс 1. Если целочисленных индексов еще нет, то ключом будет *0* (ноль).

Учтите, что максимальное целое значение ключа *не обязательно существует в массиве в данный момент*. Оно могло просто существовать в массиве какое-то время, с тех пор как он был переиндексирован в последний раз. Следующий пример это иллюстрирует:

```php
<?php
// Создаем простой массив.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Теперь удаляем каждый элемент, но сам массив оставляем нетронутым:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Добавляем элемент (обратите внимание, что новым ключом будет 5, вместо 0).
$array[] = 6;
print_r($array);

// Переиндексация:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Полезные функции ¶

Для работы с массивами существует достаточное количество полезных функций. Смотрите раздел функции для работы с массивами.

Замечание:
Функция unset() позволяет удалять ключи массива. Обратите внимание, что массив *НЕ* будет переиндексирован. Если вы действительно хотите поведения в стиле "удалить и сдвинуть", можно переиндексировать массив используя array_values().

```
<?php
$a = array(1 => 'один', 2 => 'два', 3 => 'три');
unset($a[2]);
/* даст массив, представленный так:
   $a = array(1 => 'один', 3 => 'три');
   а НЕ так:
   $a = array(1 => 'один', 2 => 'три');
*/

$b = array_values($a);
// Теперь $b это array(0 => 'один', 1 => 'три')
?>
```

Управляющая конструкция foreach существует специально для массивов. Она предоставляет возможность легко пройтись по массиву.

Что можно и нельзя делать с массивами ¶

Почему $foo[bar] *неверно?* ¶

Всегда заключайте в кавычки строковый литерал в индексе ассоциативного массива. К примеру, пишите *$foo['bar']*, а не *$foo[bar]*. Но почему? Часто в старых скриптах можно встретить следующий синтаксис:

```
<?php
$foo[bar] = 'враг';
echo $foo[bar];
// и т.д.
?>
```

This is wrong, but it works. The reason is that this code has an undefined constant (bar) rather than a string ('bar' - notice the quotes). PHP may in the future define constants which, unfortunately for such code, have the same name. It works because PHP automatically converts a*bare string* (an unquoted string which does not correspond to any known symbol) into a string which contains the bare string. For instance, if there is no defined constant named **bar**, then PHP will substitute in the string *'bar'* and use that.

Замечание: Это не означает, что нужно *всегда* заключать ключ в кавычки. Нет необходимости заключать в кавычки константы илипеременные, поскольку это помешает PHP обрабатывать их.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
// Простой массив:
$array = array(1, 2);
$count = count($array);
for ($i = 0; $i < $count; $i++) {
```

```
    echo "\nПроверяем $i: \n";
    echo "Плохо: " . $array['$i'] . "\n";
    echo "Хорошо: " . $array[$i] . "\n";
    echo "Плохо: {$array['$i']}\n";
    echo "Хорошо: {$array[$i]}\n";
}
?>
```

Результат выполнения данного примера:

```
Проверяем 0:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 1
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 1

Проверяем 1:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 2
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 2
```

Дополнительные примеры, демонстрирующие этот факт:

```
<?php
// Показываем все ошибки
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Верно
print $arr['fruit'];  // apple
print $arr['veggie']; // carrot

// Неверно. Это работает, но из-за неопределенной константы с
// именем fruit также вызывает ошибку PHP уровня E_NOTICE
//
// Notice: Use of undefined constant fruit -
 assumed 'fruit' in...
print $arr[fruit];    // apple

// Давайте определим константу, чтобы продемонстрировать, что
// происходит. Мы присвоим константе с именем fruit значение 've
ggie'.
define('fruit', 'veggie');

// Теперь обратите внимание на разницу
print $arr['fruit'];  // apple
print $arr[fruit];    // carrot
```

```php
// Внутри строки это нормально. Внутри строк константы не
// рассматриваются, так что ошибки E_NOTICE здесь не произойдет
print "Hello $arr[fruit]";      // Hello apple

// С одним исключением: фигурные скобки вокруг массивов внутри
// строк позволяют константам там находиться
print "Hello {$arr[fruit]}";    // Hello carrot
print "Hello {$arr['fruit']}";  // Hello apple

// Это не будет работать и вызовет ошибку обработки, такую как:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE'
or T_NUM_STRING'
// Это, конечно, также действует и с суперглобальными переменным
и в строках
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";

// Еще одна возможность - конкатенация
print "Hello " . $arr['fruit']; // Hello apple
?>
```

Если вы переведете error_reporting в режим отображения ошибок уровня **E_NOTICE** (например, такой как **E_ALL**), вы сразу увидите эти ошибки. По умолчанию error_reporting установлена их не отображать.

Как указано в разделе синтаксис, внутри квадратных скобок ('*[*' и '*]*') должно быть выражение. Это означает, что можно писать вот так:

```php
<?php
echo $arr[somefunc($bar)];
?>
```

Это пример использования возвращаемого функцией значения в качестве индекса массива. PHP известны также и константы:

```php
<?php
$error_descriptions[E_ERROR]   = "Произошла фатальная ошибка";
$error_descriptions[E_WARNING] = "PHP сообщает о предупреждении"
;
$error_descriptions[E_NOTICE]  = "Это лишь неофициальное замечан
ие";
?>
```

Обратите внимание, что **E_ERROR** - это такой же верный идентификатор, как и *bar* в первом примере. Но последний пример по сути эквивалентен такой записи:

```php
<?php
$error_descriptions[1] = "Произошла фатальная ошибка";
$error_descriptions[2] = "PHP сообщает о предупреждении";
$error_descriptions[8] = "Это лишь неофициальное замечание";
?>
```

поскольку **E_ERROR** соответствует *1*, и т.д.

Так что же в этом плохого?

Когда-нибудь в будущем, команда разработчиков PHP, возможно, пожелает добавить еще одну константу или ключевое слово, либо константа из другого кода может вмешаться и тогда у вас могут возникнуть проблемы. Например, вы уже не можете использовать таким образом слова *empty* и *default*, поскольку они являются зарезервированными ключевыми словами.

Замечание: Повторим, внутри строки (string), заключенной в двойные кавычки, корректно не окружать индексы массива кавычками, поэтому *"$foo[bar]"* является верной записью. Более подробно почему - смотрите вышеприведенные примеры, а также раздел обработка переменных в строках.

Преобразование в массив ¶

Для любого из типов: integer, float, string, boolean и resource, преобразование значения в массив дает результатом массив с одним элементом (с индексом 0), являющимся скалярным значением, с которого вы начали. Другими словами, *(array)$scalarValue* - это точно то же самое, что и *array($scalarValue)*.

Если вы преобразуете в массив объект (object), вы получите в качестве элементов массива свойства (переменные-члены) этого объекта. Ключами будут имена переменных-членов, с некоторыми примечательными исключениями: целочисленные свойства станут недоступны; к закрытым полям класса (private) спереди будет дописано имя класса; к защищенным полям класса (protected) спереди будет добавлен символ '*'. Эти добавленные значения с обоих сторон также имеют нулевые байты. Это может вызвать несколько неожиданное поведение:

```
<?php

class A {
    private $A; // Это станет '\0A\0A'
}

class B extends A {
    private $A; // Это станет '\0B\0A'
    public $AA; // Это станет 'AA'
}

var_dump((array) new B());
?>
```

Вышеприведенный код покажет 2 ключа с именем 'AA', хотя один из них на самом деле имеет имя '\0A\0A'.

Если вы преобразуете в массив значение **NULL**, вы получите пустой массив.

Сравнение ¶

Массивы можно сравнивать при помощи функции array_diff() и операторов массивов.

Примеры ¶

The array type in PHP is very versatile. Here are some examples: Тип массив в PHP является очень гибким, вот несколько примеров:

```
<?php
// это
$a = array( 'color' => 'red',
            'taste' => 'sweet',
            'shape' => 'round',
            'name'  => 'apple',
           4        // ключом будет 0
         );

$b = array('a', 'b', 'c');

// . . .полностью соответствует
$a = array();
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // ключом будет 0

$b = array();
$b[] = 'a';
$b[] = 'b';
$b[] = 'c';

// после выполнения этого кода, $a будет массивом
// array('color' => 'red', 'taste' => 'sweet', 'shape' => 'round
',
// 'name' => 'apple', 0 => 4), а $b будет
// array(0 => 'a', 1 => 'b', 2 => 'c'), или просто array('a', 'b
', 'c').
?>
```

Пример #8 Использование array()

```
<?php
// Массив как карта (свойств)
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// исключительно числовые ключи
```

```
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// это то же самое, что и array(0 => 7, 1 => 8, ...)

$switching = array(         10, // ключ = 0
                     5    =>  6,
                     3    =>  7,
                     'a'  =>  4,
                              11, // ключ = 6 (максимальным числов
ым индексом было 5)
                     '8'  =>  2, // ключ = 8 (число!)
                     '02' => 77, // ключ = '02'
                     0    => 12  // значение 10 будет перезаписан
о на 12
                   );

// пустой массив
$empty = array();
?>
```

Пример #9 Коллекция

```
<?php
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo "Вам нравится $color?\n";
}

?>
```

Результат выполнения данного примера:

```
Вам нравится red?
Вам нравится blue?
Вам нравится green?
Вам нравится yellow?
```

Изменение значений массива напрямую стало возможным с версии PHP 5 путем передачи их по ссылке. До этого необходим следующий обходной прием:

Пример #10 Изменение элемента в цикле

```
<?php
// PHP 5
foreach ($colors as &$color) {
    $color = strtoupper($color);
}
unset($color); /* это нужно для того, чтобы последующие записи в
```

```
$color не меняли последний элемент массива */

// Обходной прием для старых версий
foreach ($colors as $key => $color) {
    $colors[$key] = strtoupper($color);
}

print_r($colors);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
    [3] => YELLOW
)
```

Следующий пример создает массив, начинающийся с единицы.

Пример #11 Индекс, начинающийся с единицы

```
<?php
$firstquarter  = array(1 => 'Январь', 'Февраль', 'Март');
print_r($firstquarter);
?>
```

Результат выполнения данного примера:

```
Array
(
    [1] => 'Январь'
    [2] => 'Февраль'
    [3] => 'Март'
)
```

Пример #12 Заполнение массива

```
<?php
// заполняем массив всеми элементами из директории
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
    $files[] = $file;
}
closedir($handle);
?>
```

Массивы упорядочены. Вы можете изменять порядок элементов, используя различные функции сортировки. Для дополнительной информации смотрите раздел функции для работы с массивами. Вы можете подсчитать количество элементов в массиве с помощью функции count().

Пример #13 Сортировка массива

```
<?php
sort($files);
print_r($files);
?>
```

Поскольку значение массива может быть чем угодно, им также может быть другой массив. Таким образом вы можете создавать рекурсивные и многомерные массивы.

Пример #14 Рекурсивные и многомерные массивы

```
<?php
$fruits = array ( "фрукты"  => array ( "a" => "апельсин",
                                       "b" => "банан",
                                       "c" => "яблоко"
                                     ),
                  "числа"   => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6
                                     ),
                  "дырки"   => array (      "первая",
                                       5 => "вторая",
                                            "третья"
                                     )
                );

// Несколько примеров доступа к значениям предыдущего массива
echo $fruits["дырки"][5];    // напечатает "вторая"
echo $fruits["фрукты"]["a"]; // напечатает "апельсин"
unset($fruits["дырки"][0]);  // удалит "первая"

// Создаст новый многомерный массив
$juices["apple"]["green"] = "good";
?>
```

Обратите внимание, что при присваивании массива всегда происходит копирование значения. Чтобы скопировать массив по ссылке, вам нужно использовать оператор ссылки.

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 изменился,
             // $arr1 все еще array(2, 3)

$arr3 = &$arr1;
$arr3[] = 4; // теперь $arr1 и $arr3 одинаковы
?>
```

add a note

User Contributed Notes 20 notes

Объекты ¶

Инициализация объекта ¶

Для создания нового объекта, используйте выражение *new*, создающее в переменной экземпляр класса:

```
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();
?>
```

Полное рассмотрение производится в разделе Классы и Объекты.

Преобразование в объект ¶

Если object преобразуется в object, он не изменяется. Если значение другого типа преобразуется в object, создается новый экземпляр встроенного класса *stdClass*. Если значение было **NULL**, новый экземпляр будет пустым. Массивы преобразуются в object с именами полей, названными согласно ключам массива и соответствующими им значениям, за исключением числовых ключей, которые не будут доступны пока не проитерировать объект.

```
<?php
$obj = (object) array('1' => 'foo');
var_dump(isset($obj->{'1'})); // outputs 'bool(false)'
var_dump(key($obj)); // outputs 'int(1)'
?>
```

При преобразовании любого другого значения, оно будет помещено в поле с именем соответствующему типу *scalar*.

```
<?php
$obj = (object) 'ciao';
echo $obj->scalar;  // выведет 'ciao'
?>
```

User Contributed Notes 22 notes

Ресурс ¶

Resource это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями. Полный перечень этих функций и соответствующих типов ресурсов смотрите в приложении.

Смотрите также описание функции get_resource_type().

Преобразование в ресурс ¶

Поскольку тип resource содержит специальные указатели на открытые файлы, соединения с базой данных, области изображения и тому подобное, преобразование в этот тип не имеет смысла.

Освобождение ресурсов ¶

Благодаря системе подсчета ссылок, введенной в PHP 4 Zend Engine, определение отсутствия ссылок на ресурс происходит автоматически, после чего он освобождается сборщиком мусора. Поэтому, очень редко требуется освобождать память вручную.

Замечание: Постоянные соединения с базами данных являются исключением из этого правила. Они *не* уничтожаются сборщиком мусора. Подробнее смотрите в разделе о постоянных соединениях.

User Contributed Notes 1 note

NULL ¶

Специальное значение **NULL** представляет собой переменную без значения. **NULL** - это единственно возможное значение типа null.

Переменная считается null, если:

- ей была присвоена константа **NULL**.
- ей еще не было присвоено никакого значения.
- она была удалена с помощью unset().

Синтаксис ¶

Существует только одно значение типа null - регистронезависимая константа **NULL**.

```
<?php
$var = NULL;
?>
```

Смотрите также функции is_null() и unset().

Приведение к **NULL** ¶

Приведение переменной к null с использованием *(unset) $var не* удаляет переменную и ее значение. Данное выражение только возвращает **NULL**

User Contributed Notes 8 notes

Callbacks / Callables ¶

Callbacks can be denoted by callable type hint as of PHP 5.4. This documentation used callback type information for the same purpose.

Some functions like call_user_func() or usort() accept user-defined callback functions as a parameter. Callback functions can not only be simple functions, but also object methods, including static class methods.

Passing ¶

A PHP function is passed by its name as a string. Any built-in or user-defined function can be used, except language constructs such as:array(), echo, empty(), eval(), exit(), isset(), list(), print or unset().

A method of an instantiated object is passed as an array containing an object at index 0 and the method name at index 1. Accessing protected and private methods from within a class is allowed.

Static class methods can also be passed without instantiating an object of that class by passing the class name instead of an object at index 0. As of PHP 5.2.3, it is also possible to pass *'ClassName::methodName'*.

Apart from common user-defined function, anonymous functions can also be passed to a callback parameter.

Пример #1 Callback function examples

```php
<?php

// An example callback function
function my_callback_function() {
    echo 'hello world!';
}

// An example callback method
class MyClass {
    static function myCallbackMethod() {
        echo 'Hello World!';
    }
}

// Type 1: Simple callback
call_user_func('my_callback_function');
```

```php
// Type 2: Static class method call
call_user_func(array('MyClass', 'myCallbackMethod'));

// Type 3: Object method call
$obj = new MyClass();
call_user_func(array($obj, 'myCallbackMethod'));

// Type 4: Static class method call (As of PHP 5.2.3)
call_user_func('MyClass::myCallbackMethod');

// Type 5: Relative static class method call (As of PHP 5.3.0)
class A {
    public static function who() {
        echo "A\n";
    }
}

class B extends A {
    public static function who() {
        echo "B\n";
    }
}

call_user_func(array('B', 'parent::who')); // A

// Type 6: Objects implementing __invoke can be used as callable
s (since PHP 5.3)
class C {
    public function __invoke($name) {
        echo 'Hello ', $name, "\n";
    }
}

$c = new C();
call_user_func($c, 'PHP!');
?>
```

Пример #2 Callback example using a Closure

```php
<?php
// Our closure
$double = function($a) {
    return $a * 2;
};

// This is our range of numbers
$numbers = range(1, 5);

// Use the closure as a callback here to
// double the size of each element in our
// range
$new_numbers = array_map($double, $numbers);
```

```
print implode(' ', $new_numbers);
?>
```

Результат выполнения данного примера:

```
2 4 6 8 10
```

Замечание: In PHP 4, it was necessary to use a reference to create a callback that points to the actual object, and not a copy of it. For more details, see References Explained.

Замечание:
Callback-функции, зарегистрированные такими функциями как call_user_func() и call_user_func_array(), не будут вызваны при наличии не пойманного исключения, брошенного в предыдущей callback-функции.

User Contributed Notes 10 notes

Псевдо-типы и переменные, используемые в этой документации ¶

mixed ¶

mixed говорит о том, что параметр может принимать много (но необязательно все) типов.

Например, функция gettype() принимает все типы PHP, тогда как str_replace() принимает только типы string и array.

number ¶

number говорит о том, что параметр может быть либо integer, либо float.

callback ¶

Псевдо-тип callback использовался в этой документации до того, как был введен тип callable в PHP 5.4. Он означает в точности то же самое.

array|object ¶

array|object указывает, что параметр может быть как массивом array так и объектом object.

void ¶

void в качестве типа результата означает, что возвращенное значение бесполезно. *void* в списке параметров означает, что функция не принимает параметров.

... ¶

`$...` в прототипах функции означает *and so on* (и так далее). Это имя переменной используется когда функция может принимать бесконечное количество параметров.

User Contributed Notes 9 notes

Манипуляции с типами ¶

PHP не требует (и не поддерживает) явного типа при определении переменной; тип переменной определяется по контексту, в котором она используется. То есть, если вы присвоите значение типа string переменной `$var`, то `$var` станет строкой. Если вы затем присвоите `$var` целочисленное значение, она станет целым числом.

Примером автоматического преобразования типа является оператор сложения '+'. Если какой-либо из операндов является float, то все операнды интерпретируются как float, и результатом также будет float. В противном случае операнды будут интерпретироваться как целые числа и результат также будет целочисленным. Обратите внимание, что это *НЕ* меняет типы самих операндов; меняется только то, как они вычисляются и сам тип выражения.

```
<?php
$foo = "0";  // $foo это строка (ASCII 48)
$foo += 2;   // $foo теперь целое число (2)
$foo = $foo + 1.3;  // $foo теперь число с плавающей точкой (3.3
)
$foo = 5 + "10 Little Piggies"; // $foo это целое число (15)
$foo = 5 + "10 Small Pigs";     // $foo это целое число (15)
?>
```

Если последние два примера вам непонятны, смотрите Преобразование строк в числа.

Если вы хотите, чтобы переменная принудительно вычислялась как определенный тип, смотрите раздел приведение типов. Если вы хотите изменить тип переменной, смотрите settype().

Если вы хотите протестировать любой из примеров, приведенных в данном разделе, вы можете использовать функцию var_dump().

Замечание:
Поведение автоматического преобразования в массив в настоящий момент не определено.

К тому же, так как PHP поддерживает индексирование в строках аналогично смещениям элементов массивов, следующий пример будет верен для всех версий PHP:

```
<?php
$a    = 'car'; // $a - это строка
$a[0] = 'b';   // $a все еще строка
echo $a;       // bar
?>
```

Более подробно смотрите в разделе Доступ к символу в строке.

Приведение типов ¶

Приведение типов в PHP работает так же, как и в C: имя требуемого типа записывается в круглых скобках перед приводимой переменной.

```
<?php
$foo = 10;   // $foo это целое число
$bar = (boolean) $foo;   // $bar это булев тип
?>
```

Допускаются следующие приведения типов:

- (int), (integer) - приведение к integer
- (bool), (boolean) - приведение к boolean
- (float), (double), (real) - приведение к float
- (string) - приведение к string
- (array) - приведение к array
- (object) - приведение к object
- (unset) - приведение к NULL (PHP 5)

Приведение типа (binary) и поддержка префикса b были добавлены в PHP 5.2.1

Обратите внимание, что внутри скобок допускаются пробелы и символы табуляции, поэтому следующие примеры равносильны по своему действию:

```
<?php
$foo = (int) $bar;
$foo = ( int ) $bar;
?>
```

Приведение строковых литералов и переменных к бинарным строкам:

```
<?php
$binary = (binary) $string;
$binary = b"binary string";
?>
```

Замечание:
Вместо использования приведения переменной к string, можно также заключить ее в двойные кавычки.

```
<?php
$foo = 10;            // $foo - это целое число
$str = "$foo";        // $str - это строка
$fst = (string) $foo; // $fst - это тоже строка

// Это напечатает "они одинаковы"
if ($fst === $str) {
    echo "они одинаковы";
}
?>
```

Может быть не совсем ясно, что именно происходит при приведении между типами. Для дополнительной информации смотрите разделы:

User Contributed Notes 33 notes

Переменные ¶

Содержание ¶

User Contributed Notes 12 notes

Основы ¶

Переменные в PHP представлены знаком доллара с последующим именем переменной. Имя переменной чувствительно к регистру.

Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP. Правильное имя переменной должно начинаться с буквы или символа подчеркивания и состоять из букв, цифр и символов подчеркивания в любом количестве. Это можно отобразить регулярным выражением: *'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'*

Замечание: Под буквами здесь подразумеваются символы a-z, A-Z, и байты от 127 до 255 (*0x7f-0xff*).
Замечание: *$this* - это особая переменная, которой нельзя ничего присваивать.
Подсказка
Смотрите также Руководство по именованию.

Для информации о функциях работы с переменными обращайтесь к разделу функций работы с переменными.

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // выведет "Bob, Joe"

$4site = 'not yet';     // неверно; начинается с цифры
$_4site = 'not yet';    // верно; начинается с символа подчеркив
```

```
ания
$täyte = 'mansikka';    // верно; 'ä' это (Расширенный) ASCII 22
8.
?>
```

По умолчанию, переменные всегда присваиваются по значению. То есть, когда вы присваиваете выражение переменной, все значение оригинального выражения копируется в эту переменную. Это означает, к примеру, что, после того как одной переменной присвоено значение другой, изменение одной из них не влияет на другую. Дополнительную информацию об этом способе присвоения смотрите в разделе Выражения.

PHP также предлагает иной способ присвоения значений переменным: присвоение по ссылке. Это означает, что новая переменная просто ссылается (иначе говоря, "становится псевдонимом" или "указывает") на оригинальную переменную. Изменения в новой переменной отражаются на оригинале, и наоборот.

Для присвоения по ссылке, просто добавьте амперсанд (&) к началу имени присваиваемой (исходной) переменной. Например, следующий фрагмент кода дважды выводит '*Меня зовут Боб*':

```
<?php
$foo = 'Боб';              // Присваивает $foo значение 'Боб'
$bar = &$foo;              // Ссылка на $foo через $bar.
$bar = "Меня зовут $bar"; // Изменение $bar...
echo $bar;
echo $foo;                 // меняет и $foo.
?>
```

Важно отметить, что по ссылке могут быть присвоены только именованные переменные.

```
<?php
$foo = 25;
$bar = &$foo;      // Это верное присвоение.
$bar = &(24 * 7);  // Неверно; ссылка на неименованное выражение
.

function test()
{
   return 25;
}

$bar = &test();    // Неверно.
?>
```

Хотя в PHP и нет необходимости инициализировать переменные, это считается очень хорошей практикой. Неинициализированные переменные принимают значение по умолчанию в зависимости от их типа, который определяется из контекста их первого использования: булевы принимают значение **FALSE**, целые и числа

с плавающей точкой - ноль, строки (например, при использовании в echo) - пустую строку, а массивы становятся пустыми массивами.

Пример #1 Значения по умолчанию в неинициализированных переменных

```
<?php
// Неустановленная И не имеющая ссылок (т.е. без контекста использования) переменная; выведет NULL
var_dump($unset_var);

// Булевое применение; выведет 'false' (Подробнее по этому синтаксису смотрите раздел о тернарном операторе)
echo($unset_bool ? "true\n" : "false\n");

// Строковое использование; выведет 'string(3) "abc"'
$unset_str .= 'abc';
var_dump($unset_str);

// Целочисленное использование; выведет 'int(25)'
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int);

// Использование в качестве числа с плавающей точкой (float/double); выведет 'float(1.25)'
$unset_float += 1.25;
var_dump($unset_float);

// Использование в качестве массива; выведет array(1) {  [3]=>  string(3) "def" }
$unset_arr[3] = "def"; // array() + array(3 => "def") => array(3 => "def")
var_dump($unset_arr);

// Использование в качестве объекта; создает новый объект stdClass (см. http://www.php.net/manual/en/reserved.classes.php)
// Выведет: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
$unset_obj->foo = 'bar';
var_dump($unset_obj);
?>
```

Полагаться на значения по умолчанию неинициализированных переменных довольно проблематично при включении файла в другой файл, использующий переменную с таким же именем. Это также большой риск в системе безопасности при включенной опцииregister_globals. В случае работы с неинициализированной переменной вызывается ошибка уровня E_NOTICE, за исключением случая добавления элементов в неинициализированный массив. Для обнаружения инициализации переменной может быть использована языковая конструкция isset().

User Contributed Notes 9 notes

Предопределенные переменные ¶

Любому запускаемому скрипту PHP предоставляет большое количество предопределенных переменных. Однако многие из этих переменных не могут быть полностью задокументированы, поскольку они зависят от запускающего скрипт сервера, его версии и настроек, а также других факторов. Некоторые из этих переменных недоступны, когда PHP запущен из командной строки. Перечень этих переменных смотрите в разделе Зарезервированные предопределенные переменные.

Внимание
Начиная с PHP 4.2.0, значение директивы register_globals по умолчанию установлено в *off* (отключено). Это большое изменение в PHP. Положение register_globals в *off* изменяет набор глобальных предопределенных переменных. Например, чтобы получить `DOCUMENT_ROOT`, вам необходимо будет использовать `$_SERVER['DOCUMENT_ROOT']` вместо `$DOCUMENT_ROOT`, или`$_GET['id']` из URL *http://www.example.com/test.php?id=3* вместо `$id`, или `$_ENV['HOME']` вместо `$HOME`.

Дополнительную информацию, связанную с этим изменением, вы можете получить, прочитав описание register_globals в разделе о настройках, главу о безопасности Использование Register Globals , а также сообщения о выпусках PHP » 4.1.0 и » 4.2.0.

Использование доступных зарезервированных предопределенных переменных PHP, таких как суперглобальные массивы, является предпочтительным.

Начиная с версии 4.1.0, PHP предоставляет дополнительный набор предопределенных массивов, содержащих переменные web-сервера (если они доступны), окружения и пользовательского ввода. Эти новые массивы являются особыми, поскольку они становятся глобальными автоматически - то есть, автоматически доступны в любой области видимости. По этой причине они также известны как 'автоглобальные' или 'суперглобальные' переменные. (В PHP нет механизма определяемых пользователем суперглобальных переменных.) Суперглобальные переменные перечислены ниже; однако перечисление их содержимого и дальнейшее обсуждение предопределенных переменных PHP и их сути смотрите в разделе Зарезервированные предопределенные переменные. Также вы заметите, что старые предопределенные переменные (`$HTTP_*_VARS`) всё еще существуют. Начиная с PHP 5.0.0, длинныепредопределенные переменные массивов PHP могут быть отключены директивой register_long_arrays.

Замечание: Переменные переменных
Суперглобальные переменные не могут быть переменными переменных внутри функций или методов класса.
Замечание:
Хотя суперглобальный массив и соответствующий *HTTP_*_VARS* могут существовать одновременно, они не идентичны, поэтому изменение одного никак не повлияет на другой.

Если некоторые из переменных в variables_order не установлены, соответствующие им предопределенные массивы также останутся пустыми.

User Contributed Notes 35 notes

Область видимости переменной ¶

Область видимости переменной - это контекст, в котором эта переменная определена. В большинстве случаев все переменные PHP имеют только одну область видимости. Эта единая область видимости охватывает также включаемые (include) и требуемые (require) файлы. Например:

```
<?php
$a = 1;
include 'b.inc';
?>
```

Здесь переменная `$a` будет доступна внутри включенного скрипта `b.inc`. Однако определение (тело) пользовательской функции задает локальную область видимости данной функции. Любая используемая внутри функции переменная по умолчанию ограничена локальной областью видимости функции. Например:

```
<?php
$a = 1; /* глобальная область видимости */

function test()
{
    echo $a; /* ссылка на переменную локальной области видимости
 */
}

test();
?>
```

Этот скрипт не сгенерирует никакого вывода, поскольку выражение echo указывает на локальную версию переменной `$a`, а в пределах этой области видимости ей не было присвоено значение. Возможно вы заметили, что это немного отличается от языка C в том, что глобальные переменные в C автоматически доступны функциям, если только они не были перезаписаны локальным определением. Это может вызвать некоторые проблемы, поскольку люди могут нечаянно изменить глобальную переменную. В PHP, если глобальная переменная будет использоваться внутри функции, она должна быть объявлена глобальной внутри определения функции.

Ключевое слово *global* ¶

Сначала пример использования *global*:

Пример #1 Использование *global*

```
<?php
$a = 1;
```

```
$b = 2;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
}

Sum();
echo $b;
?>
```

Вышеприведенный скрипт выведет *3*. После определения `$a` и `$b` внутри функции как global все ссылки на любую из этих переменных будут указывать на их глобальную версию. Не существует никаких ограничений на количество глобальных переменных, которые могут обрабатываться функцией.

Второй способ доступа к переменным глобальной области видимости - использование специального, определяемого PHP массива`$GLOBALS`. Предыдущий пример может быть переписан так:

Пример #2 Использование *$GLOBALS* вместо global

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Sum();
echo $b;
?>
```

`$GLOBALS` - это ассоциативный массив, ключом которого является имя, а значением - содержимое глобальной переменной. Обратите внимание, что `$GLOBALS` существует в любой области видимости, это объясняется тем, что `$GLOBALS` является суперглобальным. Ниже приведен пример, демонстрирующий возможности суперглобальных переменных:

Пример #3 Суперглобальные переменные и область видимости

```
<?php
function test_global()
{
    // Большинство предопределенных переменных не являются
    // "супер", и чтобы быть доступными в локальной области
    // видимости, функции требуют указания 'global'.
    global $HTTP_POST_VARS;
```

```
    echo $HTTP_POST_VARS['name'];

    // Суперглобальные переменные доступны в любой области
    // видимости и не требуют указания 'global'.
    // Суперглобальные переменные доступны, начиная с PHP 4.1.0,
 а
    // использование HTTP_POST_VARS считается устаревшим.
    echo $_POST['name'];
}
?>
```

Замечание:
Использование ключевого слова *global* вне функции не является ошибкой. Оно может быть использовано в файле, которые включается внутрь функции.

Использование статических (*static*) переменных ¶

Другой важной особенностью области видимости переменной является *статическая* переменная. Статическая переменная существует только в локальной области видимости функции, но не теряет своего значения, когда выполнение программы выходит из этой области видимости. Рассмотрим следующий пример:

Пример #4 Демонстрация необходимости статических переменных

```
<?php
function test()
{
    $a = 0;
    echo $a;
    $a++;
}
?>
```

Эта функция довольно бесполезна, поскольку при каждом вызове она устанавливает `$a` в *0* и выводит *0*. Инкремент переменной`$a++` здесь не играет роли, так как при выходе из функции переменная `$a` исчезает. Чтобы написать полезную считающую функцию, которая не будет терять текущего значения счетчика, переменная `$a` объявляется как static:

Пример #5 Пример использования статических переменных

```
<?php
function test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Теперь `$a` будет проинициализирована только при первом вызове функции, а каждый вызов функции *test()* будет выводить значение `$a` и инкрементировать его.

Статические переменные также дают возможность работать с рекурсивными функциями. Рекурсивной является функция, вызывающая саму себя. При написании рекурсивной функции нужно быть внимательным, поскольку есть вероятность сделать рекурсию бесконечной. Вы должны убедиться, что существует адекватный способ завершения рекурсии. Следующая простая функция рекурсивно считает до 10, используя для определения момента остановки статическую переменную `$count`:

Пример #6 Статические переменные и рекурсивные функции

```
<?php
function test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        test();
    }
    $count--;
}
?>
```

Замечание:
Статические переменные могут быть объявлены так, как показано в предыдущем примере. Попытка присвоить этим переменным значения, являющиеся результатом выражений, вызовет ошибку обработки.

Пример #7 Объявление статических переменных

```
<?php
function foo(){
    static $int = 0;          // верно
    static $int = 1+2;        // неверно  (поскольку это выражен
ие)
    static $int = sqrt(121);  // неверно  (поскольку это тоже вы
ражение)

    $int++;
    echo $int;
}
?>
```

Замечание:
Статические объявления вычисляются во время компиляции скрипта.

Ссылки с глобальными (*global*) и статическими (*static*) переменными ¶

Движок Zend Engine 1, лежащий в основе PHP 4, оперирует модификаторами переменных static и global как ссылками. Например, реальная глобальная переменная, внедренная в область видимости функции указанием ключевого слова *global*,

в действительности создает ссылку на глобальную переменную. Это может привести к неожиданному поведению, как это показано в следующем примере:

```
<?php
function test_global_ref() {
    global $obj;
    $obj = &new stdclass;
}

function test_global_noref() {
    global $obj;
    $obj = new stdclass;
}

test_global_ref();
var_dump($obj);
test_global_noref();
var_dump($obj);
?>
```

Результат выполнения данного примера:

```
NULL
object(stdClass)(0)                                                    {
}
```

Аналогично ведет себя и выражение *static*. Ссылки не хранятся статично:

```
<?php
function &get_instance_ref() {
    static $obj;

    echo 'Статический объект: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Присвоить ссылку статической переменной
        $obj = &new stdclass;
    }
    $obj->property++;
    return $obj;
}

function &get_instance_noref() {
    static $obj;

    echo 'Статический объект: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Присвоить объект статической переменной
        $obj = new stdclass;
    }
    $obj->property++;
```

```
    return $obj;
}

$obj1 = get_instance_ref();
$still_obj1 = get_instance_ref();
echo "\n";
$obj2 = get_instance_noref();
$still_obj2 = get_instance_noref();
?>
```

Результат выполнения данного примера:

```
Статический                    объект:                    NULL
Статический                    объект:                    NULL

Статический                    объект:                    NULL
Статический          объект:          object(stdClass)(1)          {
["property"]=>
int(1)
}
```

Этот пример демонстрирует, что при присвоении ссылки статической переменной она не *запоминается*, когда вы вызываете функцию *&get_instance_ref()* во второй раз.

User Contributed Notes 65 notes

Переменные переменных ¶

Иногда бывает удобно иметь переменными имена переменных. То есть, имя переменной, которое может быть определено и изменено динамически. Обычная переменная определяется примерно таким выражением:

```
<?php
$a = 'hello';
?>
```

Переменная переменной берет значение переменной и рассматривает его как имя переменной. В вышеприведенном примере *hello*может быть использовано как имя переменной при помощи двух знаков доллара. То есть:

```
<?php
$$a = 'world';
?>
```

Теперь в дереве символов PHP определены и содержатся две переменные: `$a`, содержащая "hello", и `$hello`, содержащая "world". Таким образом, выражение

```
<?php
echo "$a ${$a}";
?>
```

выведет то же, что и

```
<?php
echo "$a $hello";
?>
```

то есть, они оба выведут: hello world.

Для того чтобы использовать переменные переменных с массивами, вы должны решить проблему двусмысленности. То есть, если вы напишете `$$a[1]`, обработчику необходимо знать, хотите ли вы использовать `$a[1]` в качестве переменной, либо вам нужна как переменная `$$a`, а затем ее индекс [1]. Синтаксис для разрешения этой двусмысленности таков: `${$a[1]}` для первого случая и`${$a}[1]` для второго.

К свойствам класса также можно получить доступ динамически. Переменное имя свойства будет разрешено в том контексте, в котором произойдет вызов к нему. Например, в случае выражения `$foo->$bar`, локальная область видимости будет просканирована на наличие переменной `$bar`, значение которой будет использовано в качестве имени свойства объекта `$foo`. Это также работает и в том случае, если `$bar` осуществляет доступ к элементу массива.

Фигурные скобки могут также использоваться, чтобы четко разграничить имя свойства. Они наиболее полезны при получении доступа к значениям внутри свойства, которое содержит массив, когда имя свойства состоит из нескольких частей, либо когда имя свойства содержит символы, которые иначе не действительны (например, из функции json_decode() или из SimpleXML).

Пример #1 Пример переменного имени свойства

```
<?php
class foo {
    var $bar = 'I am bar.';
    var $arr = array('I am A.', 'I am B.', 'I am C.');
    var $r   = 'I am r.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo $foo->$bar . "\n";
echo $foo->$baz[1] . "\n";

$start = 'b';
$end   = 'ar';
echo $foo->{$start . $end} . "\n";

$arr = 'arr';
echo $foo->$arr[1] . "\n";
echo $foo->{$arr}[1] . "\n";

?>
```

Результат выполнения данного примера:

```
I                                   am                                  bar.
I                                   am                                  bar.
I                                   am                                  bar.
I                                    am                                   r.
I am B.
```

Внимание
Пожалуйста, обратите внимание, что переменные переменных не могут использоваться с Суперглобальными массивами PHP. Переменная *$this* также является особой, на нее нельзя ссылаться динамически.

User Contributed Notes 50 notes

Переменные извне PHP ¶

HTML-формы (GET и POST) ¶

Когда происходит отправка данных формы PHP-скрипту, информация из этой формы автоматически становится доступной ему. Существует несколько способов получения этой информации, например:

Пример #1 Простая HTML-форма

```
<form action="foo.php" method="post">
    Имя:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Отправь меня!" />
</form>
```

С версии PHP 5.4.0, есть только два способы получить доступ к данным из HTML форм. Доступные сейчас способы приведены ниже:

Пример #2 Доступ к данным из простой HTML POST-формы

```
<?php
echo $_POST['username'];
echo $_REQUEST['username'];
?>
```

В старых версиях PHP также существовало несколько других способов. Они приведены ниже. Смотрите также список изменений внизу страницы.

Пример #3 Старые способы получения пользовательских данных

```
<?php
// ВНИМАНИЕ: эти методы больше НЕ поддерживаются.

// Использование import_request_variables() -
 эта функция удалена в PHP 5.4.0
   import_request_variables('p', 'p_');
   echo $p_username;
```

```
// Эти длинные предопределенные массивы удалены в PHP 5.4.0
   echo $HTTP_POST_VARS['username'];

// Использование register_globals. Эта функциональность удалена
в PHP 5.4.0
   echo $username;
?>
```

GET-форма используется аналогично, за исключением того, что вместо POST, вам нужно будет использовать соответствующую предопределенную переменную GET. GET относится также к *QUERY_STRING* (информация в URL после '?'). Так, например,*http://www.example.com/test.php?id=3* содержит GET-данные, доступные как `$_GET['id']`. Смотрите также `$_REQUEST`.

Замечание:
Точки и пробелы в именах переменных преобразуется в знаки подчеркивания. Например, *<input name="a.b" />* станет*$_REQUEST["a_b"]*.

PHP также понимает массивы в контексте переменных формы (смотрите соответствующие ЧАВО). К примеру, вы можете сгруппировать связанные переменные вместе или использовать эту возможность для получения значений списка множественного выбора select. Например, давайте отправим форму самой себе, а после отправки отобразим данные:

Пример #4 Более сложные переменные формы

```
<?php
if ($_POST) {
    echo '<pre>';
    echo htmlspecialchars(print_r($_POST, true));
    echo '</pre>';
}
?>
<form action="" method="post">
    Имя:  <input type="text" name="personal[name]" /><br />
    Email: <input type="text" name="personal[email]" /><br />
    Пиво: <br />
    <select multiple name="beer[]">
        <option value="warthog">Warthog</option>
        <option value="guinness">Guinness</option>
        <op-
tion value="stuttgarter">Stuttgarter Schwabenbräu</option>
    </select><br />
    <input type="submit" value="Отправь меня!" />
</form>
```

Имена переменных кнопки-изображения ¶

При отправке формы вместо стандартной кнопки можно использовать изображение с помощью тега такого вида:

```
<input type="image" src="image.gif" name="sub" />
```

Когда пользователь щелкнет где-нибудь на изображении, соответствующая форма будет передана на сервер с двумя дополнительными переменными - `sub_x` и `sub_y`. Они содержат координаты нажатия пользователя на изображение. Опытные программисты могут заметить, что на самом деле имена переменных, отправленных браузером, содержат точку, а не подчеркивание, но PHP автоматически конвертирует точку в подчеркивание.

HTTP Cookies ¶

PHP прозрачно поддерживает HTTP cookies как определено в » RFC 6265. Cookies - это механизм для хранения данных в удаленном браузере и отслеживания и идентификации таким образом вернувшихся пользователей. Вы можете установить cookies, используя функцию setcookie(). Cookies являются частью HTTP-заголовка, поэтому функция SetCookie должна вызываться до того, как браузеру будет отправлен какой бы то ни было вывод. Это ограничение аналогично ограничению функции header(). Данные, хранящиеся в cookie, доступны в соответствующих массивах данных cookie, таких как `$_COOKIE` и `$_REQUEST`. Подробности и примеры смотрите на странице setcookie() руководства.

Если вы хотите присвоить множество значений одной переменной cookie, вы можете присвоить их как массив. Например:

```
<?php
  setcookie("MyCookie[foo]", 'Testing 1', time()+3600);
  setcookie("MyCookie[bar]", 'Testing 2', time()+3600);
?>
```

Это создаст две разные cookie, хотя в вашем скрипте `MyCookie` будет теперь одним массивом. Если вы хотите установить именно одну cookie со множеством значений, примите во внимание сначала применение к значениям таких функций, как serialize() или explode().

Обратите внимание, что cookie заменит предыдущую cookie с тем же именем в вашем браузере, если только путь или домен не отличаются. Так, для приложения корзины покупок вы, возможно, захотите сохранить счетчик. То есть:

Пример #5 Пример использования setcookie()

```
<?php
if (isset($_COOKIE['count'])) {
    $count = $_COOKIE['count'] + 1;
} else {
    $count = 1;
}
setcookie('count', $count, time()+3600);
setcookie("Cart[$count]", $item, time()+3600);
?>
```

Точки в именах приходящих переменных ¶

Как правило, PHP не меняет передаваемых скрипту имен переменных. Однако следует отметить, что точка не является корректным символом в имени переменной PHP. Поэтому рассмотрим такую запись:

```
<?php
$varname.ext;  /* неверное имя переменной */
?>
```

В данном случае интерпретатор видит переменную *$varname*, после которой идет оператор конкатенации, а затем голая строка (то есть, не заключенная в кавычки строка, не соответствующая ни одному из ключевых или зарезервированных слов) 'ext'. Очевидно, что это не даст ожидаемого результата.

По этой причине важно заметить, что PHP будет автоматически заменять любые точки в именах приходящих переменных на символы подчеркивания.

Определение типов переменных ¶

Поскольку PHP определяет и конвертирует типы переменных (в большинстве случаев) как надо, не всегда очевидно, какой тип имеет данная переменная в конкретный момент времени. PHP содержит несколько функций, позволяющих определить тип переменной, таких как: gettype(), is_array(), is_float(), is_int(), is_object() и is_string(). Смотрите также раздел Типы.

Список изменений ¶

Версия	Описание
5.4.0	Register Globals, Magic Quotes и register_long_arrays удалены
5.3.0	Register Globals, Magic Quotes и register_long_arrays стали устаревшими
4.2.0	register_globals по умолчанию стала равна *off*.
4.1.0	Добавлены Суперглобальные массивы, такие как *$_POST* и *$_GET*

User Contributed Notes 34 notes

Константы ¶

Содержание ¶

Константы - это идентификаторы (имена) простых значений. Исходя из их названия, нетрудно понять, что их значение не может изменяться в ходе выполнения скрипта (исключения представляют "волшебные" константы, которые на самом деле не являются константами в полном смысле этого слова). Имена констант чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся в верхнем регистре.

Имя константы должно соответствовать тем же правилам, что и другие имена в PHP. Правильное имя начинается с буквы или символа подчеркивания и состоит

из букв, цифр и подчеркиваний. Регулярное выражение для проверки правильности имени константы выглядит так: *[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**

Подсказка
Смотрите также Руководство по именованию.

Пример #1 Правильные и неправильные имена констант

```
<?php

// Правильные имена констант
define("FOO",     "something");
define("FOO2",    "something else");
define("FOO_BAR", "something more");

// Неправильные имена констант
define("2FOO",    "something");

// Это корректное объявление, но лучше его не использовать:
// PHP однажды может зарегистрировать "волшебную" константу,
// которая сломает ваш скрипт
define("__FOO__", "something");

?>
```

Замечание: Понятие "буквы" здесь - это символы a-z, A-Z, и другие символы с ASCII-кодами от 127 до 255 (0x7f-0xff).

Как и superglobals, константы доступны из любой области видимости. Вы можете использовать константы в любом месте вашего скрипта, не обращая внимания на текущую область видимости. Подробную информацию об областях видимости можно найти здесь.

User Contributed Notes 11 notes

Синтаксис ¶

Вы можете определить константу с помощью функции define() или с помощью ключевого слова *const* вне объявления класса начиная с версии PHP 5.3.0. После того, как константа определена, ее значение не может быть изменено или аннулировано.

До PHP 5.6 константы могут содержать только скалярные данные (boolean, integer, float и string типов). С PHP 5.6 возможно также определять константы как скалярные выражения, и также можно определять константы с типом array. Можно определять константы с типом resource, но не рекомендуется, так как может привести к неожиданным результатам.

Получить значение константы можно, указав ее имя. В отличие от переменных, вам *не нужно* предварять имя константы символом *$*. Также вы можете использовать функцию constant() для получения значения константы, если вы формируете имя константы динамически. Используйте функцию get_defined_constants() для получения списка всех объявленных констант.

Замечание: Константы и (глобальные) переменные находятся в разных пространствах имен. Это означает, что, например, **TRUE** и$TRUE являются совершенно разными вещами.

Если вы используете неопределенную константу, PHP предполагает, что вы имеете ввиду само имя константы, как если бы вы указали литерал типа string (CONSTANT вместо "CONSTANT"). При этом будет сгенерирована ошибка уровня E_NOTICE. Смотрите также главу руководства, которая разъясняет, почему $foo[bar] - это неправильно (конечно, если вы перед этим не объявили *bar* как константу с помощью define()). Если вы просто хотите проверить, определена ли константа, используйте функцию defined().

Различия между константами и переменными:

- У констант нет приставки в виде знака доллара (*$*);
- До PHP 5.3 константы можно определить только с помощью функции define(), а не присваиванием значения;
- Константы могут быть определены и доступны в любом месте без учета области видимости;
- Константы не могут быть переопределены или аннулированы после первоначального объявления; и
- Константы могут иметь только скалярные значения, или скалярные и массивы в PHP 5.6 и новее. Вы можете использовать массивы в скалярных выражениях констант (например, *const FOO = array(1,2,3)[0];*), но результатом должно быть скалярное выражение.

Пример #1 Определение констант

```
<?php
define("CONSTANT", "Здравствуй, мир.");
echo CONSTANT; // выводит "Здравствуй, мир."
echo Constant; // выводит "Constant" и предупреждение.
?>
```

Пример #2 Определение констант с помощью ключевого слова *const*

```
<?php
// Работает, начиная с версии PHP 5.3.0
const CONSTANT = 'Здравствуй, мир.';

echo CONSTANT;

// Работает, начиная с версии PHP 5.6.0
const ANOTHER_CONST = CONSTANT.'; Прощай, мир.';

echo ANOTHER_CONST;
?>
```

Замечание:
В отличие от определения констант с помощью функции define(), константы, объявленные с помощью ключевого слова *const*должны быть объявлены в самой верхней области видимости, потому что они определяются при компилировании

скрипта. Это означает, что их нельзя объявлять внутри функций, циклов, выражений *if* и *try*/ *catch* блоков.

Смотрите также Константы классов.

"Волшебные" константы ¶

PHP предоставляет большой список предопределенных констант для каждого выполняемого скрипта. Многие из этих констант определяются различными модулями и будут присутствовать только в том случае, если эти модули доступны в результате динамической загрузки или в результате статической сборки.

Есть восемь волшебных констант, которые меняют свое значение в зависимости от контекста, в котором они используются. Например, значение **__LINE__** зависит от строки в скрипте, на которой эта константа указана. Специальные константы нечувствительны к регистру и их список приведен ниже:

Некоторые "волшебные" константы PHP

Имя	Описание
__LINE__	Текущий номер строки в файле.
__FILE__	Полный путь и имя текущего файла с развернутыми симлинками. Если используется внутри подключаемого файла, то возвращается имя данного файла.
__DIR__	Директория файла. Если используется внутри подключаемого файла, то возвращается директория этого файла. Это эквивалентно вызову *dirname(__FILE__)*. Возвращаемое имя директории не оканчивается на слеш, за исключением корневой директории.
__FUNCTION__	Имя функции.
__CLASS__	Имя класса. Это имя содержит название пространства имен, в котором класс был объявлен (например, *Foo\Bar*). Обратите внимание, что начиная с PHP5.4 __CLASS__ также работает в трейтах. При использовании в методах трейтов __CLASS__ является именем класса, в котором эти методы используется.
__TRAIT__	Имя трейта. Это имя содержит название пространства имен, в котором трейт был объявлен (например, *Foo\Bar*).
__METHOD__	Имя метода класса.
__NAMESPACE__	Имя текущего пространства имен.

Смотрите также get_class(), get_object_vars(), file_exists() и function_exists().

Список изменений ¶

Версия	Описание
5.4.0	Добавлена константа **__TRAIT__**
5.3.0	Добавлены константы **__DIR__** и **__NAMESPACE__**
5.0.0	Добавлена константа **__METHOD__**
5.0.0	До этой версии значения некоторых магических констант были всегда в нижнем регистре. Все они теперь являются регистрозависимыми (содержат имена такими, какими они были объявлены).
4.3.0	Добавлены константы **__FUNCTION__** и **__CLASS__**
4.0.2	**__FILE__** всегда содержит полный пусть с разрешенными симлинками. Раньше в

Версия	Описание
	некоторых случаях она содержала относительный путь.

User Contributed Notes 18 notes

Выражения ¶

Выражения - это самые важные строительные элементы PHP. Почти все, что вы пишете в PHP, является выражением. Самое простое и точное определение выражения - "все что угодно, имеющее значение".

Основными формами выражений являются константы и переменные. Если вы записываете "`$a` = 5", вы присваиваете '5' переменной`$a`. '5', очевидно, имеет значение 5 или, другими словами, '5' это выражение со значением 5 (в данном случае '5' - это целочисленная константа).

После этого присвоения вы ожидаете, что значением `$a` также является 5, поэтому, если вы написали `$b` = `$a`, вы полагаете, что работать это будет так же, как если бы вы написали `$b` = 5. Другими словами, `$a` это также выражение со значением 5. Если все работает верно, то именно так и произойдет.

Немного более сложными примерами выражений являются функции. Например, рассмотрим следующую функцию:

```
<?php
function foo ()
{
    return 5;
}
?>
```

Исходя из того, что вы хорошо знакомы с концепцией функций (если нет, то прочитайте главу о функциях), вы полагаете, что запись*$c = foo()* абсолютно эквивалентна записи *$c = 5*, и вы правы. Функции - это выражения, значением которых является то, что возвращает функция. Поскольку *foo()* возвращает 5, значением выражения *'foo()'* является 5. Как правило, функции возвращают не просто статическое значение, а что-то вычисляют.

Разумеется, значения в PHP не обязаны быть целочисленными, и очень часто ими не являются. PHP поддерживает четыре типа скалярных значений: целочисленные (integer), с плавающей точкой (float), строковые значения (string) и булевы (boolean) значения (скалярными являются значения, которые вы не можете 'разбить' на меньшие части, в отличие, например, от массивов). PHP поддерживает также два комбинированных (не скалярных) типа: массивы и объекты. Любое значение такого типа может присваиваться переменной или возвращаться функцией.

Однако PHP, как и многие другие языки, понимает гораздо больше выражений. PHP - это язык, ориентированный на выражения и рассматривающий почти все как выражение. Вернемся к примеру, с которым мы уже имели дело: '`$a` = 5'. Легко заметить, что здесь присутствуют два значения - значение целочисленной константы '5' и значение переменной `$a`, также принимающей значение 5. Но на са-

мом деле здесь присутствует и еще одно значение - значение самого присвоения. Само присвоение вычисляется в присвоенное значение, в данном случае - в 5. На практике это означает, что '$a = 5', независимо от того, что оно делает, является выражением со значением 5. Таким образом, запись '$b = ($a = 5)' равносильна записи '$a = 5; $b = 5;' (точка с запятой обозначает конец выражения). Поскольку операции присвоения анализируются справа налево, вы также можете написать '$b = $a = 5'.

Другой хороший пример ориентированности на выражения - префиксный и постфиксный инкремент и декремент. Пользователи PHP и многих других языков возможно уже знакомы с формой записи *variable*++ и *variable*--. Это операторы инкремента и декремента. Также как и C, PHP поддерживает два типа инкремента - префиксный и постфиксный. Они оба инкрементируют значение переменной и эффект их действия на нее одинаков. Разница состоит в значении выражения инкремента. Префиксный инкремент, записываемый как '++$variable', вычисляется в инкрементированное значение (PHP инкрементирует переменную до того как прочесть ее значение, отсюда название 'пре-инкремент'). Постфиксный инкремент, записываемый как '$variable++', вычисляется в первоначальное значение переменной $variable до ее приращения (PHP инкрементирует переменную после прочтения ее значения, отсюда название 'пост-инкремент').

Очень распространенным типом выражений являются выражения сравнения. Результатом вычислений являются **FALSE** (ложь) или**TRUE** (истина). PHP поддерживает операции сравнения > (больше), >= (больше либо равно), == (равно), != (не равно), < (меньше) и <= (меньше либо равно). Он также поддерживает операторы строгого равенства: === (равно и одного типа) и !== (не равно или не одного типа). Чаще всего эти выражения используются в операторах условного выполнения, таких как *if*.

Последний пример выражений, который мы здесь рассмотрим, это смешанные выражения операции и присвоения. Вы уже знаете, что если вы хотите увеличить $a на 1, вы можете просто написать '$a++' или '++$a'. Но что, если вы хотите прибавить больше, чем единицу,например, 3? Вы могли бы написать '$a++' много раз, однако, очевидно, это не очень рациональный и удобный способ. Гораздо более распространенной практикой является запись вида '$a = $a + 3'. '$a + 3' вычисляется в значение $a плюс 3 и снова присваивается $a, увеличивая в результате $a на 3. В PHP, как и в некоторых других языках, таких как C, вы можете записать это более коротким образом, что увеличит очевидность смысла и быстроту понимания кода по прошествии времени. Прибавить 3 к текущему значению $a можно с помощью записи '$a += 3'. Это означает дословно "взять значение $a, прибавить к нему 3 и снова присвоить его переменной $a". Кроме большей понятности и краткости, это быстрее работает. Значением '$a += 3', как и обычного присвоения, является присвоенное значение. Обратите внимание, что это НЕ 3, а суммированное значение $a плюс 3 (то, что было присвоено $a). Таким образом может использоваться любой бинарный оператор, например, '$a -= 5' (вычесть 5 из значения $a), '$b *= 7' (умножить значение $b на 7) и т.д.

Существует еще одно выражение, которое может выглядеть необычно, если вы не встречали его в других языках - тернарный условный оператор:

```php
<?php
$first ? $second : $third
?>
```

Если значением первого подвыражения является **TRUE** (не ноль), то выполняется второе подвыражение, которое и будет результатом условного выражения. В противном случае будет выполнено третье подвыражение и его значение будет результатом.

Следующий пример должен помочь вам немного улучшить понимание префиксного и постфиксного инкремента и выражений:

```php
<?php
function double($i)
{
    return $i*2;
}
$b = $a = 5;        /* присвоить значение пять переменным $a и $
b */
$c = $a++;          /* постфиксный инкремент, присвоить значение
 $a
                       (5) переменной $c */
$e = $d = ++$b;     /* префиксный инкремент, присвоить увеличенн
ое
                       значение $b (6) переменным $d и $e */

/* в этой точке и $d, и $e равны 6 */

$f = double($d++);  /* присвоить удвоенное значение $d перед
                       инкрементом (2*6 = 12) переменной $f */
$g = double(++$e);  /* присвоить удвоенное значение $e после
                       инкремента (2*7 = 14) переменной $g */
$h = $g += 10;      /* сначала переменная $g увеличивается на 10
,
                       приобре-
тая, в итоге, значение 24. Затем значение
                       присвое-
ния (24) присваивается переменной $h,
                       кото-
рая в итоге также становится равной 24. */
?>
```

Некоторые выражения могут рассматриваться как инструкции. В данном случае инструкция имеет вид '*expr ;* - выражение с последующей точкой с запятой. В записи *'$b = $a = 5;'*, *$a = 5* - это верное выражение, но само по себе не инструкция. Тогда как *'$b = $a = 5;'* является верной инструкцией.

Последнее, что стоит упомянуть, это истинность значения выражений. Во многих случаях, как правило, в условных операторах и циклах, вас может интересовать не конкретное значение выражения, а только его истинность (значение **TRUE** или **FALSE**). Константы**TRUE** и **FALSE** (регистро-независимые) - это два возможных булевых значения. При необходимости выражение автоматически

преобразуется в булев тип. Подробнее о том, как это происходит, смотрите в разделе о приведении типов.

PHP предоставляет полную и мощную реализацию выражений, и их полное документирование выходит за рамки этого руководства. Вышеприведенные примеры должны дать вам представление о том, что они из себя представляют и как вы сами можете создавать полезные выражения. Далее, для обозначения любого верного выражения PHP в этой документации мы будем использовать сокращение `expr`.

User Contributed Notes 20 notes

Операторы ¶

Содержание ¶

Оператором называется нечто, принимающее одно или более значений (или выражений, если говорить на жаргоне программирования), и вычисляющее новое значение (таким образом, вся конструкция может рассматриваться как выражение).

Операторы можно сгруппировать по количеству принимаемых ими значений. Унарные операторы принимают только одно значение, например, *!* (оператор логического отрицания) или ++ (инкремент). Бинарные операторы принимают два значения; это, например, знакомые всем арифметические операторы + (плюс) и - (минус), большинство поддерживаемых в PHP операторов входят именно в эту категорию. Ну и, наконец, есть всего один тернарный оператор, *? :*, принимающий три значения, обычно его так и называют -- "тернарный оператор" (хотя, возможно, более точным названием было бы "условный оператор").

Полный список PHP-операторов вы можете найти в разделе "Порядок выполнения операторов". В этом разделе также описан порядок выполнения операторов и их ассоциативность, которые точно определяют, как вычисляются выражения с несколькими разными операторами.

User Contributed Notes 13 notes

Приоритет оператора ¶

Приоритет оператора определяет, насколько "тесно" он связывает между собой два выражения. Например, выражение *1 + 5 * 3*вычисляется как *16*, а не *18*, поскольку оператор умножения ("*") имеет более высокий приоритет, чем оператор сложения ("+"). Круглые скобки могут использоваться для принудительного указания порядка выполнения операторов. Например, выражение *(1 + 5) * 3* вычисляется как *18*.

Если операторы имеют равный приоритет, то их группирование определяется их ассоциативностью. Например "-" имеет левую ассоциативность, поэтому *1 - 2 - 3* группируется как *(1 - 2) - 3* и равно *-4*. С другой стороны, "=" имеет правую ассоциативность, поэтому*$a = $b = $c* группируется как *$a = ($b = $c)*.

Операторы с равным приоритетом, но не имеющие ассоциативность, не могут использоваться вслед друг за другом, например *1 < 2 > 1*недопустимо в PHP. С другой стороны, выражение *1 <= 1 == 1* вполне допустимо, так как оператор == имеет меньший приоритет чем оператор <=.

Используйте скобки, даже если они не обязательны. Чаще всего это повышает читаемость кода, непосредственно определяя порядок группировки, не полагаясь на конкретные приоритет или ассоциативность оператора.

В следующей таблице приведен список операторов, отсортированный по убыванию их приоритетов. Операторы, размещенные в одной строке имеют одинаковый приоритет и их группировка при выполнении определяется исходя из их ассоциативности.

Порядок выполнения операторов		
Ассоциативность	**Оператор**	**Дополнительная информация**
неассоциативна	*clone new*	clone и new
левая	*[*	array()
правая	**	арифметика
правая	++ -- ~ *(int) (float) (string) (array) (object) (bool)* @	типы и инкремент/декремент
неассоциативна	*instanceof*	типы
правая	*!*	логические операторы
левая	* / %	арифметические операторы
левая	+ - .	арифметические операторы и строковые операторы
левая	<< >>	побитовые операторы
неассоциативна	< <= > >=	операторы сравнения
неассоциативна	== != === !== <>	операторы сравнения
левая	&	побитовые операторы и ссылки
левая	^	побитовые операторы
левая	|	побитовые операторы

Порядок выполнения операторов		
Ассоциативность	**Оператор**	**Дополнительная информация**
левая	*&&*	логические операторы
левая	*||*	логические операторы
левая	*? :*	тернарный оператор
правая	*= += -= *= **= /= .= %= &= |= ^= <<= >>= =>*	операторы присваивания
левая	*and*	логические операторы
левая	*xor*	логические операторы
левая	*or*	логические операторы
левая	*,*	множество применений

Пример #1 Ассоциативность

```
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// ассоциативность тройного оператора отличается от C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

приоритет и ассоциативность оператора определяет только группировку выражений, и не определяет порядок выполнения. PHP (в целом) не описывает в каком порядке выражение выполняется, и следует избегать код, опирающийся на порядок выполнения, так как поведение может меняться между версиями PHP в зависимости от окружающего кода.

Пример #2 Неизвестный порядок выполнения

```
<?php
$a = 1;
echo $a + $a++; // может вывести и 2 и 3

$i = 1;
$array[$i] = $i++; // может задать как индекс 1, так и 2
?>
```

Замечание:
Несмотря на то, что оператор = имеет низший приоритет, чем большинство остальных операторов, PHP все равно позволяет использовать следующую конструкцию: *if (!$a = foo())*, которая присваивает переменной $a результат выполнения функции *foo()*.

User Contributed Notes 6 notes

Арифметические операторы ¶

Помните школьные основы арифметики? Описанные ниже операторы работают так же.

Арифметические операции		
Пример	Название	Результат
-$a	Отрицание	Смена знака *$a*.
$a + $b	Сложение	Сумма *$a* и *$b*.
$a - $b	Вычитание	Разность *$a* и *$b*.
$a * $b	Умножение	Произведение *$a* и *$b*.
$a / $b	Деление	Частное от деления *$a* на *$b*.
$a % $b	Деление по модулю	Целочисленный остаток от деления *$a* на *$b*.
$a ** $b	Возведение в степень	Результат возведения *$a* в степень *$b*. Добавлен в PHP 5.6.

Операция деления ("/") возвращает число с плавающей точкой, кроме случая, когда оба значения являются целыми числами (или строками, которые преобразуются в целые числа), которые делятся нацело - в этом случае возвращается целое значение.

При делении по модулю операнды преобразуются в целые числа (удалением дробной части) до начала операции.

Результат операции остатка от деления % будет иметь тот же знак, что и делимое — то есть, результат *$a % $b* будет иметь тот же знак, что и $a. Например:

```
<?php

echo (5 % 3)."\n";          // выводит 2
echo (5 % -3)."\n";         // выводит 2
echo (-5 % 3)."\n";         // выводит -2
echo (-5 % -3)."\n";        // выводит -2

?>
```

Также вы можете ознакомиться с разделом документации Математические функции.

User Contributed Notes 11 notes

Оператор присваивания ¶

Базовый оператор присваивания обозначается как "=". На первый взгляд может показаться, что это оператор "равно". На самом деле это не так. В действительности, оператор присваивания означает, что левый операнд получает значение правого выражения, (т.е. устанавливается значением).

Результатом выполнения оператора присваивания является само присвоенное значение. Таким образом, результат выполнения "*$a = 3*" будет равен 3. Это позволяет делать трюки наподобие:

```
<?php

$a = ($b = 4) + 5; // $a теперь равно 9, а $b было присвоено 4.

?>
```

Для массивов (array), присвоение значения именованному ключу происходит с помощью оператора "=>". Приоритет этого оператора такой же, как и у остальных операторов присваивания.

В дополнение к базовому оператору присваивания имеются "комбинированные операторы" для всех бинарных арифметическихопераций, операций объединения массивов и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения. Например:

```
<?php

$a = 3;
$a += 5; // устанавливает $a в 8, как если бы мы написали: $a =
$a + 5;
$b = "Hello ";
$b .= "There!"; // устанавливает $b в "Hello There!",  как и $b
= $b . "There!";

?>
```

Обратите внимание, что присвоение копирует оригинальную переменную в новую (присвоение по значению), таким образом все последующие изменения одной из переменных никак не отразятся на другой. Это также следует учитывать, если вам надо скопировать что-то типа большого массива в длинном цикле.

Исключением из обычного для PHP способа присваивания по значению являются объекты (object), которые, начиная с версии PHP 5, присваиваются по ссылке. Принудительно скопировать объекты по значению можно с помощью специального ключевого слова clone.

Присваивание по ссылке ¶

Присваивание по ссылке также поддерживается, для него используется синтаксис $var = &$othervar;. 'Присваивание по ссылке' означает, что обе переменные указывают на одни и те же данные и никакого копирования не происходит.

Пример #1 Присваивание по ссылке

```
<?php
$a = 3;
$b = &$a; // $b - это ссылка на $a

print "$a\n"; // печатает 3
print "$b\n"; // печатает 3

$a = 4; // меняем $a

print "$a\n"; // печатает 4
print "$b\n"; // также печатает 4, так как $b является ссылкой н
а $a,
              // а значение переменной $a успело измениться
?>
```

Начиная с версии PHP 5, оператор new автоматически возвращает ссылку, поэтому присваивание результата операции new по ссылке начиная с версии PHP 5.3 генерирует ошибку уровня **E_DEPRECATED**, а в более ранних версиях - ошибку уровня **E_STRICT**.

Например, следующий код выдаст предупреждение:

```
<?php
class C {}

/* Следующая строка сгенерирует следующее сообщение об ошибке:
 * Deprecated: Assigning the return value of new by reference is
 deprecated in...
 * (Устаревший код: Присвоение результата работы new по ссылке у
старело в...)
 */
$o = &new C;
?>
```

Для получения более полной информации о ссылках и их возможностях обратитесь к разделу Подробно о ссылках.

User Contributed Notes 9 notes

Побитовые операторы ¶

Побитовые операторы позволяют считывать и устанавливать конкретные биты целых чисел.

Побитовые операторы			
Пример		**Название**	**Результат**
$a **$b**	**&**	И	Устанавливаются только те биты, которые установлены и в *$a*, и в *$b*.
$a **$b**	**|**	Или	Устанавливаются те биты, которые установлены в *$a* или в *$b*.
$a **$b**	**^**	Исключающее или	Устанавливаются только те биты, которые установлены либо только в *$a*, либо только в *$b*, но не в обоих одновременно.
~ $a		Отрицание	Устанавливаются те биты, которые не установлены в *$a*, и наоборот.
$a **$b**	**<<**	Сдвиг влево	Все биты переменной *$a* сдвигаются на *$b* позиций влево (каждая позиция подразумевает "умножение на 2")
$a **$b**	**>>**	Сдвиг вправо	Все биты переменной *$a* сдвигаются на *$b* позиций вправо (каждая позиция подразумевает "деление на 2")

Побитовый сдвиг в PHP - это арифметическая операция. Биты, сдвинутые за границы числа, отбрасываются. Сдвиг влево дополняет число нулями справа, сдвигая в то же время знаковый бит числа влево, что означает что знак операнда не сохраняется. Сдвиг вправо сохраняет копию сдвинутого знакового бита слева, что означает что знак операнда сохраняется.

Используйте скобки для обеспечения необходимого приоритета операторов. Например, *$a & $b == true* сначала проверяет на равенство, а потом выполняет

побитовое и; тогда как *($a & $b) == true* сначала выполняет побитовое и, а потом выполняет проверку на равенство.

Если оба операнда для *&*, | и ^ являются строками, то операция будет выполнена над ASCII значениями символов, составляющих эти строки, и результатом будет строка. В других случаях, оба операнда будут приведены к целому и результатом будет тоже целое число.

Если операнд для оператора ~ является строкой, то операция будет произведена над ASCII значением символов, составляющих эту строку, и результатом будет строка. Иначе, и операнд и результат будут оперироваться как целые числа.

Both operands and the result for the << and >> operators are always treated as integers.

```
        Опция настроек PHP error_reporting использует побитовые значения,
        обеспечивая реальную демонстрацию гашения значений битов.
        Чтобы показать все ошибки кроме замечаний,
        инструкции в файле php.ini предлагают использовать:
        E_ALL & ~E_NOTICE

        Начинаем со значения E_ALL:
        00000000000000000111011111111111
        Затем берем значение E_NOTICE...
        00000000000000000000000000001000
        ... и инвертируем его с помощью ~:
        11111111111111111111111111110111
        Наконец, используем побитовое И (&), чтобы установить только те биты,
        которые установлены в единицу в обоих значениях:
        00000000000000000111011111110111

        Другой способ достичь этого - использовать ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR, ^),
        чтобы получить только те биты, которые установлены в единицу
        либо только в одном, либо только в другом значении:
        E_ALL ^ E_NOTICE

        Опция error_reporting также может быть использована для демонстрации
        установки битов. Показать только ошибки и обрабатываемые ошибки можно
        следующим образом:
        E_ERROR | E_RECOVERABLE_ERROR

        Здесь мы комбинируем E_ERROR
        00000000000000000000000000000001
        и
        00000000000000000001000000000000
        с помощью оператора ИЛИ (OR, |),
```

чтобы получить биты, установленные хотя бы в одном операнде:

00000000000000000001000000000001

Пример #1 Побитовыми операции И, ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ (AND, OR и XOR) над целыми числами

```
<?php
/*
 * Не обращайте внимания на этот верхний раздел кода,
 * это просто форматирование для более ясного вывода.
 */

$format = '(%1$2d = %1$04b) = (%2$2d = %2$04b)'
        . ' %3$s (%4$2d = %4$04b)' . "\n";

echo <<<EOH
 ---------     ---------  -- ---------
 результат     значение   оп   тест
 ---------     ---------  -- ---------
EOH;


/*
 * Вот сами примеры.
 */

$values = array(0, 1, 2, 4, 8);
$test = 1 + 4;

echo "\n Побитовое И (AND) \n";
foreach ($values as $value) {
    $result = $value & $test;
    printf($format, $result, $value, '&', $test);
}

echo "\n Побитовое (включающее) ИЛИ (OR) \n";
foreach ($values as $value) {
    $result = $value | $test;
    printf($format, $result, $value, '|', $test);
}

echo "\n Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) \n";
foreach ($values as $value) {
    $result = $value ^ $test;
    printf($format, $result, $value, '^', $test);
}
?>
```

Результат выполнения данного примера:

```
 ---------     ---------  -- ---------
```

```
 результат     значение   оп   тест
 ---------     ---------  -- ---------
 Побитовое И (AND)
( 0 = 0000) = ( 0 = 0000) & ( 5 = 0101)
( 1 = 0001) = ( 1 = 0001) & ( 5 = 0101)
( 0 = 0000) = ( 2 = 0010) & ( 5 = 0101)
( 4 = 0100) = ( 4 = 0100) & ( 5 = 0101)
( 0 = 0000) = ( 8 = 1000) & ( 5 = 0101)

 Побитовое (включающее) ИЛИ (OR)
( 5 = 0101) = ( 0 = 0000) | ( 5 = 0101)
( 5 = 0101) = ( 1 = 0001) | ( 5 = 0101)
( 7 = 0111) = ( 2 = 0010) | ( 5 = 0101)
( 5 = 0101) = ( 4 = 0100) | ( 5 = 0101)
(13 = 1101) = ( 8 = 1000) | ( 5 = 0101)

 Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
( 5 = 0101) = ( 0 = 0000) ^ ( 5 = 0101)
( 4 = 0100) = ( 1 = 0001) ^ ( 5 = 0101)
( 7 = 0111) = ( 2 = 0010) ^ ( 5 = 0101)
( 1 = 0001) = ( 4 = 0100) ^ ( 5 = 0101)
(13 = 1101) = ( 8 = 1000) ^ ( 5 = 0101)
```

Пример #2 Побитовая операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) над строками

```php
<?php
echo 12 ^ 9; // Выводит '5'

echo "12" ^ "9"; // Выводит символ Backspace (ascii 8)
                 // ('1' (ascii 49)) ^ ('9' (ascii 57)) = #8

echo "hallo" ^ "hello"; // Выводит ascii значения #0 #4 #0 #0 #0
                        // 'a' ^ 'e' = #4

echo 2 ^ "3"; // Выводит 1
              // 2 ^ ((int)"3") == 1

echo "2" ^ 3; // Выводит 1
              // ((int)"2") ^ 3 == 1
?>
```

Пример #3 Побитовый сдвиг над целыми числами

```php
<?php
/*
 * Несколько примеров.
 */

echo "\n--- СДВИГ ВПРАВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
\n";

$val = 4;
$places = 1;
```

```php
$res = $val >> $places;
p($res, $val, '>>', $places, 'слева была вставлена копия знаково
го бита');

$val = 4;
$places = 2;
$res = $val >> $places;
p($res, $val, '>>', $places);

$val = 4;
$places = 3;
$res = $val >> $places;
p($res, $val, '>>', $places, 'биты были выдвинуты за правый край
');

$val = 4;
$places = 4;
$res = $val >> $places;
p($res, $val, '>>', $places, 'то же, что и выше; нельзя сдвинуть
 дальше 0');


echo "\n--- СДВИГ ВПРАВО НАД ОТРИЦАТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
\n";

$val = -4;
$places = 1;
$res = $val >> $places;
p($res, $val, '>>', $places, 'слева была вставлена копия знаково
го бита');

$val = -4;
$places = 2;
$res = $val >> $places;
p($res, $val, '>>', $places, 'биты были выдвинуты за правый край
');

$val = -4;
$places = 3;
$res = $val >> $places;
p($res, $val, '>>', $places, 'то же, что и выше; нельзя сдвинуть
 дальше -1');


echo "\n--- СДВИГ ВЛЕВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
\n";

$val = 4;
$places = 1;
$res = $val << $places;
p($res, $val, '<<', $places, 'правый край был дополнен нулями');

$val = 4;
```

```php
$places = (PHP_INT_SIZE * 8) - 4;
$res = $val << $places;
p($res, $val, '<<', $places);

$val = 4;
$places = (PHP_INT_SIZE * 8) - 3;
$res = $val << $places;
p($res, $val, '<<', $places, 'знаковые биты были выдвинуты');

$val = 4;
$places = (PHP_INT_SIZE * 8) - 2;
$res = $val << $places;
p($res, $val, '<<', $places, 'биты были выдвинуты за левый край'
);


echo "\n--- СДВИГ ВЛЕВО НАД ОТРИЦАТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
\n";

$val = -4;
$places = 1;
$res = $val << $places;
p($res, $val, '<<', $places, 'правый край был дополнен нулями');

$val = -4;
$places = (PHP_INT_SIZE * 8) - 3;
$res = $val << $places;
p($res, $val, '<<', $places);

$val = -4;
$places = (PHP_INT_SIZE * 8) - 2;
$res = $val << $places;
p($res, $val, '<<', $places, 'биты были выдвинуты за левый край,
 включая знаковый бит');


/*
 * Не обращайте внимания на этот нижний раздел кода,
 * это просто форматирование для более ясного вывода.
 */

function p($res, $val, $op, $places, $note = '') {
    $format = '%0' . (PHP_INT_SIZE * 8) . "b\n";

    printf("Выражение: %d = %d %s %d\n", $res, $val, $op, $place
s);

    echo " Десятичный вид:\n";
    printf("  val=%d\n", $val);
    printf("  res=%d\n", $res);

    echo " Двоичный вид:\n";
    printf('  val=' . $format, $val);
```

```
    printf('  res=' . $format, $res);

    if ($note) {
        echo " ЗАМЕЧАНИЕ: $note\n";
    }

    echo "\n";
}
?>
```

Результат выполнения данного примера на 32-битных машинах:

```
--- СДВИГ ВПРАВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: 2 = 4 >> 1
 Десятичный вид:
  val=4
  res=2
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000010
 ЗАМЕЧАНИЕ: слева была вставлена копия знакового бита

Выражение: 1 = 4 >> 2
 Десятичный вид:
  val=4
  res=1
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000001

Выражение: 0 = 4 >> 3
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 ЗАМЕЧАНИЕ: биты были выдвинуты за правый край

Выражение: 0 = 4 >> 4
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 ЗАМЕЧАНИЕ: то же, что и выше; нельзя сдвинуть дальше 0


--- СДВИГ ВПРАВО НА ОТРИЦАТЕЛЬНЫХ ЦЕЛЫХ ЧИСЛАХ ---
Выражение: -2 = -4 >> 1
 Десятичный вид:
```

```
  val=-4
  res=-2
 Двоичный вид:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111110
 ЗАМЕЧАНИЕ: слева была вставлена копия знакового бита

Выражение: -1 = -4 >> 2
 Десятичный вид:
  val=-4
  res=-1
 Двоичный вид:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111111
 ЗАМЕЧАНИЕ: биты были выдвинуты за правый край

Выражение: -1 = -4 >> 3
 Десятичный вид:
  val=-4
  res=-1
 Двоичный вид:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111111
 ЗАМЕЧАНИЕ: то же, что и выше; нельзя сдвинуть дальше -1


--- СДВИГ ВЛЕВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: 8 = 4 << 1
 Десятичный вид:
  val=4
  res=8
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000001000
 ЗАМЕЧАНИЕ: правый край был дополнен нулями

Выражение: 1073741824 = 4 << 28
 Десятичный вид:
  val=4
  res=1073741824
 Двоичный вид:
  val=00000000000000000000000000000100
  res=01000000000000000000000000000000

Выражение: -2147483648 = 4 << 29
 Десятичный вид:
  val=4
  res=-2147483648
 Двоичный вид:
  val=00000000000000000000000000000100
  res=10000000000000000000000000000000
 ЗАМЕЧАНИЕ: знаковые биты были выдвинуты
```

```
Выражение: 0 = 4 << 30
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 ЗАМЕЧАНИЕ: биты были выдвинуты за левый край


--- СДВИГ ВЛЕВО НАД ОТРИЦАТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: -8 = -4 << 1
 Десятичный вид:
  val=-4
  res=-8
 Двоичный вид:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111000
 ЗАМЕЧАНИЕ: правый край был дополнен нулями

Выражение: -2147483648 = -4 << 29
 Десятичный вид:
  val=-4
  res=-2147483648
 Двоичный вид:
  val=11111111111111111111111111111100
  res=10000000000000000000000000000000

Выражение: 0 = -4 << 30
 Десятичный вид:
  val=-4
  res=0
 Двоичный вид:
  val=11111111111111111111111111111100
  res=00000000000000000000000000000000
 ЗАМЕЧАНИЕ: биты были выдвинуты за левый край, включая знаковый
бит
```

Результат выполнения данного примера на 64-битных машинах:

```
--- СДВИГ ВПРАВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: 2 = 4 >> 1
 Десятичный вид:
  val=4
  res=2
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000000
0100

res=00000000000000000000000000000000000000000000000000000000000
0010
```

```
 ЗАМЕЧАНИЕ: слева была вставлена копия знакового бита

Выражение: 1 = 4 >> 2
 Десятичный вид:
  val=4
  res=1
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000000
0100

res=00000000000000000000000000000000000000000000000000000000000
0001

Выражение: 0 = 4 >> 3
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000000
0100

res=00000000000000000000000000000000000000000000000000000000000
0000
 ЗАМЕЧАНИЕ: биты были выдвинуты за правый край

Выражение: 0 = 4 >> 4
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000000
0100

res=00000000000000000000000000000000000000000000000000000000000
0000
 ЗАМЕЧАНИЕ: то же, что и выше; нельзя сдвинуть дальше 0


--- СДВИГ ВПРАВО НАД ОТРИЦАТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: -2 = -4 >> 1
 Десятичный вид:
  val=-4
  res=-2
 Двоичный вид:

val=11111111111111111111111111111111111111111111111111111111111
1100

res=11111111111111111111111111111111111111111111111111111111111
1110
```

```
 ЗАМЕЧАНИЕ: слева была вставлена копия знакового бита

Выражение: -1 = -4 >> 2
 Десятичный вид:
  val=-4
  res=-1
 Двоичный вид:

val=11111111111111111111111111111111111111111111111111111111111
1100

res=11111111111111111111111111111111111111111111111111111111111
1111
 ЗАМЕЧАНИЕ: биты были выдвинуты за правый край

Выражение: -1 = -4 >> 3
 Десятичный вид:
  val=-4
  res=-1
 Двоичный вид:

val=11111111111111111111111111111111111111111111111111111111111
1100

res=11111111111111111111111111111111111111111111111111111111111
1111
 ЗАМЕЧАНИЕ: то же, что и выше; нельзя сдвинуть дальше -1


--- СДВИГ ВЛЕВО НАД ПОЛОЖИТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: 8 = 4 << 1
 Десятичный вид:
  val=4
  res=8
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000010
0100

res=00000000000000000000000000000000000000000000000000000001000
1000
 ЗАМЕЧАНИЕ: правый край был дополнен нулями

Выражение: 4611686018427387904 = 4 << 60
 Десятичный вид:
  val=4
  res=4611686018427387904
 Двоичный вид:

val=00000000000000000000000000000000000000000000000000000000010
0100
```

```
res=0100000000000000000000000000000000000000000000000000000000000000
0000

Выражение: -9223372036854775808 = 4 << 61
 Десятичный вид:
  val=4
  res=-9223372036854775808
 Двоичный вид:

val=0000000000000000000000000000000000000000000000000000000000000000
0100

res=1000000000000000000000000000000000000000000000000000000000000000
0000
 ЗАМЕЧАНИЕ: знаковые биты были выдвинуты

Выражение: 0 = 4 << 62
 Десятичный вид:
  val=4
  res=0
 Двоичный вид:

val=0000000000000000000000000000000000000000000000000000000000000000
0100

res=0000000000000000000000000000000000000000000000000000000000000000
0000
 ЗАМЕЧАНИЕ: биты были выдвинуты за левый край


--- СДВИГ ВЛЕВО НАД ОТРИЦАТЕЛЬНЫМИ ЦЕЛЫМИ ЧИСЛАМИ ---
Выражение: -8 = -4 << 1
 Десятичный вид:
  val=-4
  res=-8
 Двоичный вид:

val=1111111111111111111111111111111111111111111111111111111111111111
1100

res=1111111111111111111111111111111111111111111111111111111111111111
1000
 ЗАМЕЧАНИЕ: правый край был дополнен нулями

Выражение: -9223372036854775808 = -4 << 61
 Десятичный вид:
  val=-4
  res=-9223372036854775808
 Двоичный вид:

val=1111111111111111111111111111111111111111111111111111111111111111
1100
```

```
res=10000000000000000000000000000000000000000000000000000000000000
0000

Выражение: 0 = -4 << 62
 Десятичный вид:
  val=-4
  res=0
 Двоичный вид:

val=11111111111111111111111111111111111111111111111111111111111111
1100

res=00000000000000000000000000000000000000000000000000000000000000
0000
 ЗАМЕЧАНИЕ: биты были выдвинуты за левый край, включая знаковый
бит
```

Внимание
Сдвиг целых чисел на значение, большее или равное максимальной ширине целых целых чисел в системе будет приводить к неопределенному поведению. Другими словами, не производите сдвиг больше чем на 31 в 32-битных платформах, и 63 в 64-битных.

Используйте функции из расширения gmp для двоичных операций над числами больше *PHP_INT_MAX*.

Смотрите также pack(), unpack(), gmp_and(), gmp_or(), gmp_xor(), gmp_testbit(), gmp_clrbit()

User Contributed Notes 45 notes

Операторы сравнения ¶

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения. Возможно вам будет интересно также ознакомиться с разделом Сравнение типов, в котором приведено большое количество соответствующих примеров.

Операторы сравнения

Пример	Название	Результат
$a == $b	Равно	**TRUE** если *$a* равно *$b* после преобразования типов.
$a === $b	Тождественно равно	**TRUE** если *$a* равно *$b* и имеет тот же тип.
$a != $b	Не равно	**TRUE** если *$a* не равно *$b* после преобразования типов.
$a <> $b	Не равно	**TRUE** если *$a* не равно *$b* после преобразования типов.
$a !== $b	Тождественно не равно	**TRUE** если *$a* не равно *$b* или они разных типов.
$a < $b	Меньше	**TRUE** если *$a* строго меньше *$b*.
$a > $b	Больше	**TRUE** если *$a* строго больше *$b*.
$a <= $b	Меньше или равно	**TRUE** если *$a* меньше или равно *$b*.
$a >= $b	Больше или равно	**TRUE** если *$a* больше или равно *$b*.

В случае, если вы сравниваете число со строкой или две строки, содержащие числа, каждая строка будет преобразована в число, и сравниваться они будут как числа. Эти правила также распространяются на оператор switch. Преобразование типов не происходит при использовании === или !== так как в этом случае кроме самих значений сравниваются еще и типы.

```
<?php
var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true
var_dump("10" == "1e1"); // 10 == 10 -> true
var_dump(100 == "1e2"); // 100 == 100 -> true

switch ("a") {
case 0:
    echo "0";
    break;
case "a": // Эта ветка никогда не будет достигнута, так как "a"
уже сопоставленно с 0
    echo "a";
    break;
}
?>
```

Для различных типов сравнение происходит в соответствии со следующей таблицей (по порядку).

Сравнение различных типов		
Тип операнда 1	**Тип операнда 2**	**Результат**
null илиstring	string	**NULL** преобразуется в "", числовое или лексическое сравнение
bool илиnull	что угодно	Оба операнда преобразуются в bool, **FALSE** < **TRUE**
object	object	Встроенные классы могут определять свои собственные правила сравнения, объекты разных классов не сравниваются, объекты одного класса - сравниваются свойства тем же способом, что и в массивах (PHP 4), в PHP 5 есть свое собственное объяснение
string,resourceилиnumber	string,resourceилиnumber	Строки и ресурсы переводятся в числа, обычная математика
array	array	Массивы с меньшим числом элементов считаются меньше, если ключ из первого операнда не найден во втором операнде - массивы не могут сравниваться, иначе идет сравнение соответствующих значений (смотри пример ниже)
object	что угодно	object всегда больше
array	что угодно	array всегда больше

Пример #1 Сравнение булево/null

```php
<?php
// Булево и null всегда сравниваются как булево значение
var_dump(1 == TRUE);  // TRUE - тоже что и (bool)1 == TRUE
var_dump(0 == FALSE); // TRUE - тоже что и (bool)0 == FALSE
var_dump(100 < TRUE); // FALSE - тоже что и (bool)100 < TRUE
var_dump(-10 < FALSE);// FALSE - тоже что и (bool)-10 < FALSE
var_dump(min(-100, -10, NULL, 10, 100)); // NULL -
 (bool)NULL < (bool)-100 , тоже что и FALSE < TRUE
?>
```

Пример #2 Алгоритм сравнения обычных массивов

```php
<?php
// Так сравниваются массивы при сравнении стандартными операторами
function standard_array_compare($op1, $op2)
{
    if (count($op1) < count($op2)) {
        return -1; // $op1 < $op2
    } elseif (count($op1) > count($op2)) {
        return 1; // $op1 > $op2
    }
    foreach ($op1 as $key => $val) {
        if (!array_key_exists($key, $op2)) {
            return null; // не могут быть сравнимы
        } elseif ($val < $op2[$key]) {
            return -1;
        } elseif ($val > $op2[$key]) {
            return 1;
        }
    }
    return 0; // $op1 == $op2
}
?>
```

Смотрите также strcasecmp(), strcmp(), операторы массивов, и раздел руководства Типы.

Внимание

Сравнение чисел с плавающей точкой

Из-за особого внутреннего представления float, не нужно проверять на равенство два float-числа.

Для более подробной информации смотрите документацию по типу float.

Тернарный оператор ¶

Еще одним условным оператором является тернарный оператор "?:".

Пример #3 Присваивание значения по умолчанию

```php
<?php
// Пример использования тернарного оператора
$action = (empty($_POST['action'])) ? 'default' : $_POST['action
'];

// Приведенный выше код аналогичен следующему блоку с использова
нием if/else
if (empty($_POST['action'])) {
    $action = 'default';
} else {
    $action = $_POST['action'];
}

?>
```

Выражение *(expr1) ? (expr2) : (expr3)* интерпретируется как expr2, если expr1 имеет значение **TRUE**, или как expr3 если expr1 имеет значение **FALSE**.

Начиная с версии PHP 5.3 также стало возможным не писать среднюю часть тернарного оператора. Выражение *expr1 ?: expr3*возвращает expr1 если expr1 имеет значение **TRUE**, и expr3 в другом случае.

Замечание: Пожалуйста учтите, что тернарный оператор является выражением и трактуется не как переменная, а как результат выражения. Это важно знать, если вы хотите вернуть переменную по ссылке. Выражение *return $var == 42 ? $a : $b;* не будет работать в функции, возвращающей значение по ссылке, а в более поздних версиях PHP также будет выдано предупреждение.
Замечание:
Рекомендуется избегать "нагромождения" тернарных выражений. Поведение PHP неочевидно при использовании нескольких тернарных операторов в одном выражении:
Пример #4 Неочевидное поведение тернарного оператора

```php
<?php
// на первый взгляд, следующий код должен вывести 'true'
echo (true?'true':false?'t':'f');

// однако он выводит 't'
// это происходит потому, что тернарные выражения вычисляются сл
ева направо

// это намного более очевидная версия вышеприведенного кода
echo ((true ? 'true' : false) ? 't' : 'f');

// здесь вы можете видеть, что первое выражение вычисляется в 't
rue', которое
// в свою очередь вычисляется в (bool)true, таким образом возвра
щая истинную ветвь
// второго тернарного выражения.
?>
```

User Contributed Notes 50 notes

Оператор управления ошибками ¶

PHP поддерживает один оператор управления ошибками: знак (@). В случае, если он предшествует какому-либо выражению в PHP-коде, любые сообщения об ошибках, генерируемые этим выражением, будут проигнорированы.

Если вы установили собственную функцию обработки ошибок с помощью set_error_handler(), то она все равно будет вызвана, однако если внутри этой функции будет вызвана функция error_reporting(), то она вернет 0, если функция, вызвавшая данную ошибку, была подавлена с помощью @.

В случае, если установлена опция track_errors , все генерируемые сообщения об ошибках будут сохраняться в переменной`$php_errormsg`. Эта переменная будет перезаписываться при каждой новой ошибке, поэтому в случае необходимости проверяйте ее сразу же.

```
<?php
// Преднамеренная ошибка при работе с файлами
$my_file = @file ('non_existent_file') or
    die ("Ошибка при открытии файла: сообщение об ошибке было та
ким: '$php_errormsg'");

// работает для любых выражений, а не только для функций
$value = @$cache[$key];
// В случае если ключа $key нет, сообщение об ошибке (notice) не
 будет отображено

?>
```

Замечание: Оператор @ работает только с выражениями. Есть простое правило: если что-то возвращает значение, значит вы можете использовать перед ним оператор @. Например, вы можете использовать @ перед именем переменной, произвольной функцией или вызовом include, константой и так далее. В то же время вы не можете использовать этот оператор перед определением функции или класса, условными конструкциями, такими как *if*, foreach и т.д.

Также ознакомьтесь с описанием функции error_reporting() и разделом руководства Обработка ошибок и функции логирования.

Внимание
На сегодняшний день оператор "@" подавляет вывод сообщений даже о критических ошибках, прерывающих работу скрипта. Помимо всего прочего, это означает, что если вы использовали "@" для подавления ошибок, возникающих при работе какой-либо функции, в случае если она недоступна или написана неправильно, дальнейшая работа скрипта будет остановлена без каких-либо уведомлений.

User Contributed Notes 19 notes

Операторы исполнения ¶

PHP поддерживает один оператор исполнения: обратные кавычки (``). Обратите внимание, что это не одинарные кавычки! PHP попытается выполнить строку, за-

ключенную в обратные кавычки, как консольную команду, и вернет полученный вывод (т.е. он не просто выводится на экран, а, например, может быть присвоен переменной). Использование обратных кавычек аналогично использованию функции shell_exec().

```
<?php
$output = `ls -al`;
echo "<pre>$output</pre>";
?>
```

Замечание:
Обратные кавычки недоступны, в случае, если включен безопасный режим или отключена функция shell_exec().
Замечание:
В отличие от некоторых других языков, обратные кавычки не будут работать внутри строк в двойных кавычках.

Ознакомьтесь также со следующими разделами документации: Функции для выполнения программ, popen() proc_open(), иИспользование PHP в командной строке.

User Contributed Notes 8 notes

Операторы инкремента и декремента ¶

PHP поддерживает префиксные и постфиксные операторы инкремента и декремента в стиле C.

Замечание: Операторы инкремента/декремента влияют только на строки и числа. Массивы, объекты и ресурсы не трогаются. Декремент **NULL** также не даст никакого эффекта, однако инкремент даст значение *1*.

Операторы инкремента и декремента

Пример	Название	Действие
++$a	Префиксный инкремент	Увеличивает *$a* на единицу, затем возвращает значение *$a*.
$a++	Постфиксный инкремент	Возвращает значение *$a*, затем увеличивает *$a* на единицу.
--$a	Префиксный декремент	Уменьшает *$a* на единицу, затем возвращает значение *$a*.
$a--	Постфиксный декремент	Возвращает значение *$a*, затем уменьшает *$a* на единицу.

Приведем пример простого скрипта:

```
<?php
echo "<h3>Постфиксный инкремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a++ . "<br />\n";
echo "Должно быть 6: " . $a . "<br />\n";

echo "<h3>Префиксный инкремент</h3>";
$a = 5;
echo "Должно быть 6: " . ++$a . "<br />\n";
```

```
echo "Должно быть 6: " . $a . "<br />\n";

echo "<h3>Постфиксный декремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a-- . "<br />\n";
echo "Должно быть 4: " . $a . "<br />\n";

echo "<h3>Префиксный декремент</h3>";
$a = 5;
echo "Должно быть 4: " . --$a . "<br />\n";
echo "Должно быть 4: " . $a . "<br />\n";
?>
```

PHP следует соглашениям Perl (в отличие от C) касательно выполнения арифметических операций с символьными переменными. Например, в PHP и Perl *$a = 'Z'; $a++;* присвоит *$a* значение *'AA'*, в то время как в C *a = 'Z'; a++;* присвоит *a* значение *'['* (ASCII значение *'Z'* равно 90, а ASCII значение *'['* равно 91). Следует учесть, что к символьным переменным можно применять операцию инкремента, в то время как операцию декремента применять нельзя, кроме того, поддерживаются только алфавит ASCII и цифры (a-z, A-Z и 0-9). Попытка инкремента/декремента других символьных переменных не будет иметь никакого эффекта, исходная строка останется неизменной.

Пример #1 Арифметические операции с символьными переменными

```
<?php
echo '== Алфавит ==' . PHP_EOL;
$s = 'W';
for ($n=0; $n<6; $n++) {
    echo ++$s . PHP_EOL;
}
// Цифры ведут себя по другому
echo '== Цифры ==' . PHP_EOL;
$d = 'A8';
for ($n=0; $n<6; $n++) {
    echo ++$d . PHP_EOL;
}
$d = 'A08';
for ($n=0; $n<6; $n++) {
    echo ++$d . PHP_EOL;
}
?>
```

Результат выполнения данного примера:

```
== Алфавит ==
X
Y
Z
AA
AB
AC
== Цифры ==
```

```
A9
B0
B1
B2
B3
B4
A09
A10
A11
A12
A13
A14
```

Инкрементирование или декрементирование булевых переменных не приводит ни к какому результату.

User Contributed Notes 11 notes

Логические операторы ¶

Логические операторы		
Пример	**Название**	**Результат**
$a and $b	И	**TRUE** если и *$a*, и *$b* **TRUE**.
$a or $b	Или	**TRUE** если или *$a*, или *$b* **TRUE**.
$a xor $b	Исключающее или	**TRUE** если *$a*, или *$b* **TRUE**, но не оба.
! $a	Отрицание	**TRUE** если *$a* не **TRUE**.
$a && $b	И	**TRUE** если и *$a*, и *$b* **TRUE**.
$a || $b	Или	**TRUE** если или *$a*, или *$b* **TRUE**.

Смысл двух разных вариантов для операторов "and" и "or" в том, что они работают с различными приоритетами (смотрите таблицуПриоритет выполнения операторов).

Пример #1 Объяснение логических операторов

```php
<?php

// --------------------
// foo() никогда не буде вызвана, так как эти операторы являются
 шунтирующими (short-circuit)

$a = (false && foo());
$b = (true  || foo());
$c = (false and foo());
$d = (true  or  foo());

// --------------------
// "||" имеет больший приоритет, чем "or"

// Результат выражения (false || true) присваивается переменной
$e
// Действует как: ($e = (false || true))
$e = false || true;
```

```php
// Константа false присваивается $f, а затем значение true игнорируется
// Действует как: (($f = false) or true)
$f = false or true;

var_dump($e, $f);

// ---------------------
// "&&" имеет больший приоритет, чем "and"

// Результат выражения (true && false) присваивается переменной $g
// Действует как: ($g = (true && false))
$g = true && false;

// Константа true присваивается $h, а затем значение false игнорируется
// Действует как: (($h = true) and false)
$h = true and false;

var_dump($g, $h);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
bool(true)
bool(false)
bool(false)
bool(true)
```

User Contributed Notes 18 notes

Строковые операторы ¶

В PHP есть два оператора для работы со строками (string). Первый - оператор конкатенации ('.'), который возвращает строку, представляющую собой соединение левого и правого аргумента. Второй - оператор присваивания с конкатенацией ('.='), который присоединяет правый аргумент к левому. Для получения более полной информации ознакомьтесь с разделом Операторы присваивания.

```php
<?php
$a = "Hello ";
$b = $a . "World!"; // $b теперь содержит строку "Hello World!"

$a = "Hello ";
$a .= "World!";     // $a теперь содержит строку "Hello World!"
?>
```

Также ознакомьтесь с разделами документации Строки и Функции для работы со строками.

User Contributed Notes 6 notes

Операторы, работающие с массивами ¶

Операторы, работающие с массивами		
Пример	**Название**	**Результат**
$a + $b	Объединение	Объединение массива *$a* и массива *$b*.
$a == $b	Равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же пары ключ/значение.
$a === $b	Тождественно равно	**TRUE** в случае, если *$a* и *$b* содержат одни и те же паты ключ/значение в том же самом порядке и того же типа.
$a != $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a <> $b	Не равно	**TRUE**, если массив *$a* не равен массиву *$b*.
$a !== $b	Тождественно не равно	**TRUE**, если массив *$a* не равен тождественно массиву *$b*.

Оператор + возвращает левый массив, к которому был присоединен правый массив. Для ключей, которые существуют в обоих массивах, будут использованы значения из левого массива, а соответствующие им элементы из правого массива будут проигнорированы.

```php
<?php
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

$c = $a + $b; // Объединение $a и $b
echo "Объединение of \$a and \$b: \n";
var_dump($c);

$c = $b + $a; // Объединение $b и $a
echo "Объединение of \$b and \$a: \n";
var_dump($c);
?>
```

После своего выполнения скрипт напечатает следующее:

```
Объединение of $a and $b:
array(3) {
  ["a"]=>
  string(5) "apple"
  ["b"]=>
  string(6) "banana"
  ["c"]=>
  string(6) "cherry"
}
Объединение of $b and $a:
array(3) {
  ["a"]=>
  string(4) "pear"
  ["b"]=>
  string(10) "strawberry"
  ["c"]=>
  string(6) "cherry"
}
```

При сравнении элементы массива считаются идентичными, если совпадает и ключ, и соответствующее ему значение.

Пример #1 Сравнение массивов

```php
<?php
$a = array("apple", "banana");
$b = array(1 => "banana", "0" => "apple");

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
?>
```

Также ознакомьтесь с разделами Массивы и Функции для работы с массивами.

User Contributed Notes 11 notes

Оператор проверки типа ¶

Оператор *instanceof* используется для определения того, является ли текущий объект экземпляром указанного класса.

Пример #1 Использование *instanceof* с классами

```php
<?php
class MyClass
{
}

class NotMyClass
{
}
$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
?>
```

Результат выполнения данного примера:

```
bool(true)
bool(false)
```

Оператор *instanceof* также может быть использован для определения, наследует ли определенный объект какому-либо классу:

Пример #2 Использование *instanceof* с наследуемыми классами

```php
<?php
class ParentClass
{
}
```

```php
class MyClass extends ParentClass
{
}

$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof ParentClass);
?>
```

Результат выполнения данного примера:

```
bool(true)
bool(true)
```

Для проверки *непринадлежности* объекта некоторому классу, используйте логический оператор *not*.

Пример #3 Использование *instanceof* для проверки того, что объект *не* является экземпляром класса

```php
<?php
class MyClass
{
}

$a = new MyClass;
var_dump(!($a instanceof stdClass));
?>
```

Результат выполнения данного примера:

```
bool(true)
```

Ну и наконец, *instanceof* может быть также использован для проверки реализации объектом некоторого интерфейса:

Пример #4 Использование *instanceof* для класса

```php
<?php
interface MyInterface
{
}

class MyClass implements MyInterface
{
}

$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof MyInterface);
?>
```

Результат выполнения данного примера:

```
bool(true)
bool(true)
```

Хотя *instanceof* обычно используется с прямо указанным именем класса, он также может быть использован с другим объектом или строковой переменной:

Пример #5 Использование *instanceof* с другими переменными

```php
<?php
interface MyInterface
{
}

class MyClass implements MyInterface
{
}

$a = new MyClass;
$b = new MyClass;
$c = 'MyClass';
$d = 'NotMyClass';

var_dump($a instanceof $b); // $b это объект класса MyClass
var_dump($a instanceof $c); // $c это строка 'MyClass'
var_dump($a instanceof $d); // $d это строка 'NotMyClass'
?>
```

Результат выполнения данного примера:

```
bool(true)
bool(true)
bool(false)
```

Оператор instanceof не генерирует никаких ошибок, если проверяемая переменная не является объектом. В этом случае он просто возвращает **FALSE**. Константы, тем не менее, не допускаются.

Пример #6 Пример использования оператора *instanceof* для проверки других переменных

```php
<?php
$a = 1;
$b = NULL;
$c = imagecreate(5, 5);
var_dump($a instanceof stdClass); // $a это целое типа integer
var_dump($b instanceof stdClass); // $b это NULL
var_dump($c instanceof stdClass); // $c это значение типа resour
ce
var_dump(FALSE instanceof stdClass);
?>
```

Результат выполнения данного примера:

```
bool(false)
bool(false)
bool(false)
PHP Fatal error:  instanceof expects an object instance, con-
stant given
```

Есть несколько подводных камней, которых следует остерегаться. До версии PHP 5.1.0, *instanceof* вызывал __autoload() если имя класса не существовало. Вдобавок, если класс не был загружен, происходила фатальная ошибка. Это можно было обойти с помощью динамической ссылки на класс или использования строковой переменной с именем класса:

Пример #7 Избежание поиска класса и фатальных ошибок с *instanceof* в PHP 5.0

```
<?php
$d = 'NotMyClass';
var_dump($a instanceof $d); // нет фатальной ошибки
?>
```

Результат выполнения данного примера:

```
bool(false)
```

Оператор *instanceof* был добавлен в PHP 5. До этого времени использовалась функция is_a(), но позже is_a() была помечена устаревшей в пользу *instanceof*. Учтите, что с версии PHP 5.3.0, is_a() больше не является устаревшей.

Ознакомьтесь также с описанием функций get_class() и is_a().

User Contributed Notes 18 notes

Управляющие конструкции ¶

Содержание ¶

- require
- include
- require_once
- include_once
- goto

User Contributed Notes 4 notes

Введение ¶

Любой сценарий PHP состоит из последовательности инструкций. Инструкцией может быть присваивание, вызов функции, повтор кода (цикл), сравнение, или даже инструкция, которая ничего не делает (пустой оператор). После инструкции обычно ставится точка с запятой. Кроме того, инструкции могут быть объединены в блоки заключением их в фигурные скобки. Блок инструкций также сам по себе является инструкцией. В этом разделе описываются различные типы инструкций.

if ¶

(PHP 4, PHP 5, PHP 7)

Конструкция *if* является одной из наиболее важных во многих языках программирования, в том числе и PHP. Она предоставляет возможность условного выполнения фрагментов кода. Структура *if* реализована в PHP по аналогии с языком C:

```
if (выражение)
  инструкция
```

Как описано в разделе про выражения, выражение вычисляется в булево значение. Если выражение принимает значение **TRUE**, PHP выполнит инструкцию, а если оно принимает значение **FALSE** - проигнорирует. Информацию о том, какие значения считаются равными значению **FALSE**, можно найти в разделе 'Преобразование в булев тип'.

Следующий пример выведет a больше b, если значение переменной `$a` больше, чем `$b`:

```
<?php
if ($a > $b)
  echo "a больше b";
?>
```

Часто необходимо, чтобы условно выполнялось более одной инструкции. Разумеется, для этого нет необходимости обворачивать каждую инструкцию в *if*. Вместо этого можно объединить несколько инструкций в блок. Например, следующий код выведет a больше b, если значение переменной `$a` больше, чем `$b`, и затем присвоит значение переменной `$a` переменной `$b`:

```
<?php
if ($a > $b) {
  echo "a больше b";
  $b = $a;
```

```
}
?>
```

Инструкции *if* могут быть бесконечно вложены в другие инструкции *if*, что даёт большую гибкость в организации условного выполнения различных частей программы.

User Contributed Notes 26 notes

else ¶
(PHP 4, PHP 5, PHP 7)

Часто необходимо выполнить одно выражение, если определенное условие верно, и другое выражение, если условие не верно. Именно для этого *else* и используется. *else* расширяет оператор *if*, чтобы выполнить выражение, в случае если условие в операторе *if*равно **FALSE**. К примеру, следующий код выведет a больше чем b, если `$a` больше, чем `$b`, и a НЕ больше, чем b в противном случае:

```
<?php
if ($a > $b) {
  echo "a больше, чем b";
} else {
  echo "a НЕ больше, чем b";
}
?>
```

Выражение *else* выполняется только, если выражение *if* эквивалентно **FALSE**, и если нет других любых выражений *elseif*, или если они все равны **FALSE** также (смотри elseif).

User Contributed Notes 8 notes

elseif/*else if* ¶
(PHP 4, PHP 5, PHP 7)

Конструкция *elseif*, как ее имя и говорит есть сочетание *if* и *else*. Аналогично *else*, она расширяет оператор *if* для выполнения различных выражений в случае, когда условие начального оператора *if* эквивалентно **FALSE**. Однако в отличие от *else* выполнение альтернативного выражения произойдет только тогда, когда условие оператора *elseif* будет являться равным **TRUE**. К примеру, следующий код может выводить a больше, чем b, a равно b or a меньше, чем b:

```
<?php
if ($a > $b) {
    echo "a больше, чем b";
} elseif ($a == $b) {
    echo "a равен b";
} else {
    echo "a меньше, чем b";
}
?>
```

Может быть несколько *elseif* в одном *if* выражении. Первое же выражение *elseif* (если будет хоть одно) равное **TRUE** будет выполнено. В PHP вы также

можете написать 'else if' (в два слова), и тогда поведение будет идентичным 'elseif' (в одно слово). Синтаксически значение немного отличается (если Вы знакомы с языком C, это тоже самое поведение), но в конечном итоге оба выражения приведут к одному и тому же результату.

Выражение *elseif* выполнится, если предшествующее выражение *if* и предшествующие выражения *elseif* эквивалентны **FALSE**, а текущий *elseif* равен **TRUE**.

Замечание: Заметьте, что *elseif* и *else if* будут равнозначны только при использовании фигурных скобок, как в примерах выше. Если используются двоеточие для определения условий *if*/*elseif*, Вы не должны разделять *else if* в два слова, иначе это вызовет фатальную ошибку в PHP.

```
<?php

/* Некорректный способ: */
if($a > $b):
    echo $a." больше, чем ".$b;
else if($a == $b): // Не скомпилируется.
    echo "Строка выше вызывает фатальную ошибку.";
endif;


/* Корректный способ: */
if($a > $b):
    echo $a." больше, чем ".$b;
elseif($a == $b): // Заметьте, тут одно слово.
    echo $a." равно ".$b;
else:
    echo $a." не больше и не равно ".$b;
endif;

?>
```

User Contributed Notes 5 notes

Альтернативный синтаксис управляющих структур ¶

(PHP 4, PHP 5, PHP 7)

PHP предлагает альтернативный синтаксис для некоторых его управляющих структур, а именно: *if*, *while*, *for*, *foreach* и *switch*. В каждом случае основной формой альтернативного синтаксиса является изменение открывающей фигурной скобки на двоеточие (:), а закрывающей скобки на *endif;*, *endwhile;*, *endfor;*, *endforeach;* или *endswitch;* соответственно.

```
<?php if ($a == 5): ?>
A равно 5
<?php endif; ?>
```

В приведенном выше примере, блок HTML "A равно 5" вложен внутрь структуры *if* написанной с альтернативным синтаксисом. HTML блок будет показан только если переменная `$a` равна 5.

Альтернативный синтаксис также применяется и к *else* и *elseif*. Ниже приведена структура *if* с *elseif* и *else* в альтернативном формате:

```
<?php
if ($a == 5):
    echo "a равно 5";
    echo "...";
elseif ($a == 6):
    echo "a равно 6";
    echo "!!!";
else:
    echo "a не равно ни 5 ни 6";
endif;
?>
```

Замечание:
Смешивание синтаксиса в одном и том же блоке управления не поддерживается.

Внимание
Любой вывод (включая пробельные символы) между выражением *switch* и первым *case* приведут к синтаксической ошибке. Например этот код не будет работать:

```
<?php switch ($foo): ?>
    <?php case 1: ?>
    ...
<?php endswitch ?>
```

В то же время следующий пример будет работать, так как завершающий перевод строки после выражения *switch* считается частью закрывающего *?>* и следовательно ничего не выводится между *switch* и *case*:

```
<?php switch ($foo): ?>
<?php case 1: ?>
    ...
<?php endswitch ?>
```

Смотри также while, for, и if.

User Contributed Notes 7 notes

while ¶
(PHP 4, PHP 5, PHP 7)

Циклы *while* являются простейшим видом циклов в PHP. Они ведут себя так же, как и их коллеги из языка C. Простейшей формой цикла *while* является следующее выражение:

```
while (expr)
    statement
```

Смысл выражения *while* очень прост. Оно указывает PHP выполнять вложенные выражения повторно до тех пор, пока выражение в самом *while* является **TRUE**. Значение выражения *expr* проверяется каждый раз перед началом цикла, поэтому даже если значение выражения изменится в процессе выполнения вложенных выражений в цикле, выполнение не прекратится до конца итерации (каждый раз,

когда PHP выполняет выражения в цикле - это одна итерация). В том случае, если выражение *while* равно **FALSE** с самого начала, вложенные выражения ни разу не будут выполнены.

Также, как и с оператором *if*, вы можете группировать несколько выражений внутри одного цикла *while*, заключая эти выражения между фигурными скобками или используя альтернативный синтаксис:

```
while (expr):
    statement
    ...
endwhile;
```

Следующие примеры идентичны, и оба выведут числа от 1 до 10:

```
<?php
/* пример 1 */

$i = 1;
while ($i <= 10) {
    echo $i++;  /* выводится будет значение переменной
                   $i перед её увеличением
                   (post-increment) */
}

/* пример 2 */

$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

User Contributed Notes 20 notes

do-while ¶

(PHP 4, PHP 5, PHP 7)

Цикл *do-while* очень похож на цикл *while*, с тем отличием, что истинность выражения проверяется в конце итерации, а не в начале. Главное отличие от обычного цикла *while* в том, что первая итерация цикла *do-while* гарантированно выполнится (истинность выражения проверяется в конце итерации), тогда как она может не выполниться в обычном цикле *while* (истинность выражения которого проверяется в начале выполнения каждой итерации, и если изначально имеет значение **FALSE**, то выполнение цикла будет прервано сразу).

Есть только один вариант синтаксиса цикла *do-while*:

```
<?php
$i = 0;
do {
    echo $i;
```

```
} while ($i > 0);
?>
```

В примере цикл будет выполнен ровно один раз, так как после первой итерации, когда проверяется истинность выражения, она будет вычислена как **FALSE** (`$i` не больше 0) и выполнение цикла прекратится.

Опытные пользователи C могут быть знакомы с другим использованием цикла *do-while*, которое позволяет остановить выполнение хода программы в середине блока, для этого нужно обернуть нужный блок кода вызовом *do-while* (0) и использовать *break*. Следующий фрагмент кода демонстрирует этот подход:

```
<?php
do {
    if ($i < 5) {
        echo "i еще недостаточно велико";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
   echo "значение i уже подходит";

    /* обработка i */

} while (0);
?>
```

Не беспокойтесь, если вы не понимаете это сразу или вообще. Вы можете писать скрипты и даже мощные программы без использования этой 'возможности'. Начиная с версии PHP 5.3.0, стало возможным использовать оператор *goto* вместо подобного "хака".

User Contributed Notes 10 notes

for ¶

(PHP 4, PHP 5, PHP 7)

Цикл *for* самый сложный цикл в PHP. Он ведет себя так же как его аналог в языке C. Синтаксис цикла *for* следующий:

```
for (expr1; expr2; expr3)
    statement
```

Первое выражение (`expr1`) всегда вычисляется (выполняется) только один раз в начале цикла.

В начале каждой итерации оценивается выражение `expr2`. Если оно принимает значение **TRUE**, то цикл продолжается, и вложенные операторы будут выполнены. Если оно принимает значение **FALSE**, выполнение цикла заканчивается.

В конце каждой итерации выражение `expr3` вычисляется (выполняется).

Каждое из выражений может быть пустым или содержать несколько выражений, разделенных запятыми. В `expr2` все выражения, разделенные запятыми, вычисляются, но результат берется из последнего. Если выражение `expr2` отсутствует, это означает, что цикл будет выполняться бесконечно. (PHP неявно воспринимает это значение как **TRUE**, также, как в языке C). Это может быть не столь бесполезно, сколь вы могли подумать, так как часто необходимо прервать цикл, используя условный оператор *break* вместо использования выражения в цикле *for*, которое принимает истинное значение.

Рассмотрим следующие примеры. Все из них отображают числа от 1 до 10:

```
<?php
/* пример 1 */

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* пример 2 */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* пример 3 */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* пример 4 */

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

Конечно, первый пример кажется самым хорошим (или, возможно, четвертый), но вы можете обнаружить, что возможность использовать пустые выражения в циклах *for* может стать удобной во многих случаях.

PHP также поддерживает альтернативный синтаксис с двоеточием для циклов *for*.

```
for (expr1; expr2; expr3):
```

```
    statement
    ...
endfor;
```

Перебор массивов как показано ниже - это обычное дело для многих пользователей.

```
<?php
/*
 * Это массив с некоторыми данными, которые мы хотим изменить
 * при работе цикла.
 */
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0; $i < count($people); ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

Вышеприведенный код может работать медленно, так как размер массива вычисляется в каждой итерации. Поскольку размер не меняется, цикл может быть легко оптимизирован с помощью промежуточной переменной, в которую будет записан размер массива, вместо повторяющихся вызовов функции count():

```
<?php
$people = array(
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);

for($i = 0, $size = count($people); $i < $size; ++$i) {
    $people[$i]['salt'] = mt_rand(000000, 999999);
}
?>
```

User Contributed Notes 17 notes

foreach ¶

(PHP 4, PHP 5, PHP 7)

Конструкция *foreach* предоставляет простой способ перебора массивов. *Foreach* работает только с массивами и объектами, и будет генерировать ошибку при попытке использования с переменными других типов или неинициализированными переменными. Существует два вида синтаксиса:

```
foreach (array_expression as $value)
    statement
foreach (array_expression as $key => $value)
    statement
```

Первый цикл перебирает массив, задаваемый с помощью *array_expression*. На каждой итерации значение текущего элемента присваивается переменной *$value* и внутренний указатель массива увеличивается на единицу (таким образом, на следующей итерации цикла работа будет происходить со следующим элементом).

Второй цикл будет дополнительно соотносить ключ текущего элемента с переменной *$key* на каждой итерации.

Возможно настроить итераторы объектов.

Замечание:
Когда оператор *foreach* начинает исполнение, внутренний указатель массива автоматически устанавливается на первый его элемент Это означает, что нет необходимости вызывать функцию reset() перед использованием цикла *foreach*.

Так как оператор *foreach* опирается на внутренний указатель массива, его изменение внутри цикла может привести к непредсказуемому поведению.

Для того, чтобы напрямую изменять элементы массива внутри цикла, переменной *$value* должен предшествовать знак &. В этом случае значение будет присвоено по ссылке.

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
    $value = $value * 2;
}
// массив $arr сейчас таков: array(2, 4, 6, 8)
unset($value); // разорвать ссылку на последний элемент
?>
```

Указатель на *$value* возможен, только если на перебираемый массив можно ссылаться (т.е. если он является переменной). Следующий код не будет работать:

```
<?php
foreach (array(1, 2, 3, 4) as &$value) {
    $value = $value * 2;
}
?>
```

Внимание
Ссылка *$value* на последний элемент массива остается даже после того, как оператор *foreach* завершил работу. Рекомендуется уничтожить ее с помощью функции unset().

Замечание:
Оператор *foreach* не поддерживает возможность подавления сообщений об ошибках с помощью префикса '@'.

Вы могли заметить, что следующие конструкции функционально идентичны:

```
<?php
$arr = array("one", "two", "three");
```

```
reset($arr);
while (list(, $value) = each($arr)) {
    echo "Значение: $value<br />\n";
}

foreach ($arr as $value) {
    echo "Значение: $value<br />\n";
}
?>
```

Следующие конструкции также функционально идентичны:

```
<?php
$arr = array("one", "two", "three");
reset($arr);
while (list($key, $value) = each($arr)) {
    echo "Ключ: $key; Значение: $value<br />\n";
}

foreach ($arr as $key => $value) {
    echo "Ключ: $key; Значение: $value<br />\n";
}
?>
```

Вот еще несколько примеров, демонстрирующие использование оператора:

```
<?php
/* Пример 1: только значение */

$a = array(1, 2, 3, 17);

foreach ($a as $v) {
    echo "Текущее значение переменной \$a: $v.\n";
}

/* Пример 2: значение (для иллюстрации массив выводится в виде з
начения с ключом) */

$a = array(1, 2, 3, 17);

$i = 0; /* только для пояснения */

foreach ($a as $v) {
    echo "\$a[$i] => $v.\n";
    $i++;
}

/* Пример 3: ключ и значение */

$a = array(
    "one" => 1,
    "two" => 2,
    "three" => 3,
```

```
    "seventeen" => 17
);

foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}

/* Пример 4: многомерные массивы */
$a = array();
$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";

foreach ($a as $v1) {
    foreach ($v1 as $v2) {
        echo "$v2\n";
    }
}

/* Пример 5: динамические массивы */

foreach (array(1, 2, 3, 4, 5) as $v) {
    echo "$v\n";
}
?>
```

Распаковка вложенных массивов с помощью list() ¶
(PHP 5 >= 5.5.0, PHP 7)

В PHP 5.5 была добавлена возможность обхода массива массивов с распаковкой вложенного массива в переменные цикла, передавlist() в качестве значения.

Например:

```
<?php
$array = [
    [1, 2],
    [3, 4],
];

foreach ($array as list($a, $b)) {
    // $a содержит первый элемент вложенного массива,
    // а $b содержит второй элемент.
    echo "A: $a; B: $b\n";
}
?>
```

Результат выполнения данного примера:

```
A: 1; B: 2
A: 3; B: 4
```

Можно передавать меньшее количество элементов в list(), чем находится во вложенном массиве, в этом случае оставшиеся значения массива будут проигнорированы:

```
<?php
$array = [
    [1, 2],
    [3, 4],
];

foreach ($array as list($a)) {
    // Обратите внимание на отсутствие $b.
    echo "$a\n";
}
?>
```

Результат выполнения данного примера:

```
1
3
```

Если массив содержит недостаточно элементов для заполнения всех переменных из list(), то будет сгенерировано замечание об ошибке:

```
<?php
$array = [
    [1, 2],
    [3, 4],
];

foreach ($array as list($a, $b, $c)) {
    echo "A: $a; B: $b; C: $c\n";
}
?>
```

Результат выполнения данного примера:

```
Notice: Undefined offset: 2 in example.php on line 7
A: 1; B: 2; C:

Notice: Undefined offset: 2 in example.php on line 7
A: 3; B: 4; C:
```

User Contributed Notes 24 notes

break ¶

(PHP 4, PHP 5, PHP 7)

break прерывает выполнение текущей структуры *for*, *foreach*, *while*, *do-while* или *switch*.

break принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать.

```
<?php
$arr = array('один', 'два', 'три', 'четыре', 'стоп', 'пять');
while (list(, $val) = each($arr)) {
    if ($val == 'стоп') {
        break;    /* Тут можно было написать 'break 1;'. */
    }
    echo "$val<br />\n";
}

/* Использование дополнительного аргумента. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "Итерация 5<br />\n";
        break 1;  /* Выйти только из конструкции switch. */
    case 10:
        echo "Итерация 10; выходим<br />\n";
        break 2;  /* Выходим из конструкции switch и из цикла wh
ile. */
    default:
        break;
    }
}
?>
```

История изменений *break*	
Версия	**Описание**
5.4.0	*break 0;* больше не допускается. В предыдущих версиях это воспринималось точно также как и *break 1;*.
5.4.0	Удалена возможность передачи переменных (например, *$num = 2; break $num;*) в виде числового аргумента.

User Contributed Notes 17 notes

continue ¶

(PHP 4, PHP 5, PHP 7)

continue используется внутри циклических структур для пропуска оставшейся части текущей итерации цикла и, при соблюдении условий, начала следующей итерации.

Замечание: Заметим, что в PHP структура switch считается циклической, и внутри нее может использоваться *continue*.

continue принимает необязательный числовой аргумент, который указывает на скольких уровнях вложенных циклов будет пропущена оставшаяся часть итерации. Значением по умолчанию является *1*, при которой пропускается оставшаяся часть текущего цикла.

```php
<?php
while (list($key, $value) = each($arr)) {
    if (!($key % 2)) { // пропуск нечетных чисел
        continue;
    }
    do_something_odd($value);
}

$i = 0;
while ($i++ < 5) {
    echo "Снаружи<br />\n";
    while (1) {
        echo "В середине<br />\n";
        while (1) {
            echo "Внутри<br />\n";
            continue 3;
        }
        echo "Это никогда не будет выведено.<br />\n";
    }
    echo "Это тоже.<br />\n";
}
?>
```

Пропуск точки запятой после *continue* может привести к путанице. Пример как не надо делать.

```php
<?php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2)
        continue
    print "$i\n";
}
?>
```

Ожидается, что результат будет такой:

```
0
1
3
4
```

Но, в PHP до версии 5.4.0, этот скрипт выведет следующее:

```
2
```

Потому что выражение *continue print "$i\n";* воспринимается как единое выражение, и print вызывается только тогда, когда выражение *$i == 2* истинно. (Возвращаемое значение от *print* передается к *continue* как числовой аргумент.)

Замечание:
Начиная с PHP 5.4.0, вышеприведенный пример вызовет ошибку **E_COMPILE_ERROR**.

Изменения, касающиеся оператора *continue*	
Версия	**Описание**
5.4.0	*continue 0;* больше не допускается. В предыдущих версиях это воспринималось точно также как и *continue 1;*.
5.4.0	Убрана возможность задавать переменную (например, *$num = 2; continue $num;*) в качестве числового аргумента.

User Contributed Notes 17 notes

switch ¶

(PHP 4, PHP 5, PHP 7)

Оператор *switch* подобен серии операторов IF с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же переменную (или выражение) с множеством различных значений, и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Это именно тот случай, для которого удобен оператор *switch*.

Замечание: Обратите внимание, что в отличие от некоторых других языков, оператор continue применяется в конструкциях switch и действует подобно оператору *break*. Если у вас конструкция switch находится внутри цикла, и вам необходимо перейти к следующей итерации цикла, используйте *continue 2*.
Замечание:
Заметьте, что конструкция swich/case использует неточное сравнение (==).

Следующие два примера иллюстрируют два различных способа написать то же самое. Один использует серию операторов *if* и *elseif*, а другой -- оператор *switch*:

Пример #1 Оператор *switch*

```
<?php
if ($i == 0) {
    echo "i равно 0";
} elseif ($i == 1) {
    echo "i равно 1";
} elseif ($i == 2) {
    echo "i равно 2";
}

switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
}
?>
```

Пример #2 Оператор *switch* допускает сравнение со строками

```php
<?php
switch ($i) {
    case "яблоко":
        echo "i это яблоко";
        break;
    case "шоколадка":
        echo "i это шоколадка";
        break;
    case "пирог":
        echo "i это пирог";
        break;
}
?>
```

Важно понять, как оператор *switch* выполняется, чтобы избежать ошибок. Оператор *switch* исполняет строчка за строчкой (на самом деле выражение за выражением). В начале никакой код не исполняется. Только в случае нахождения оператора *case*, значение которого совпадает со значением выражения в операторе *switch*, PHP начинает исполнять операторы. PHP продолжает исполнять операторы до конца блока *switch* либо до тех пор, пока не встретит оператор *break*. Если вы не напишете оператор *break* в конце секции case, PHP будет продолжать исполнять команды следующей секции case. Например :

```php
<?php
switch ($i) {
    case 0:
        echo "i равно 0";
    case 1:
        echo "i равно 1";
    case 2:
        echo "i равно 2";
}
?>
```

В этом примере, если `$i` равно 0, то PHP исполнит все операторы echo! Если `$i` равно 1, PHP исполнит два последних оператора echo. Вы получите ожидаемое поведение оператора ('i равно 2' будет отображено) только, если `$i` будет равно 2. Таким образом, важно не забывать об операторах *break* (даже если вы, возможно, хотите избежать его использования по назначению при определенных обстоятельствах).

В операторе *switch* выражение вычисляется один раз и этот результат сравнивается с каждым оператором *case*. В выражении *elseif*, выражение вычисляется снова. Если ваше условие более сложное, чем простое сравнение и/или находится в цикле, конструкция*switch* может работать быстрее.

Список операторов для исполнения в секции case также может быть пустым, что просто передает управление списку операторов в следующей секции case.

```php
<?php
switch ($i) {
case 0:
```

```
case 1:
case 2:
    echo "i меньше чем 3, но неотрицательно";
    break;
case 3:
    echo "i равно 3";
}
?>
```

Специальный вид конструкции case -- *default*. Сюда управление попадает тогда, когда не сработал ни один из других операторов case. Например:

```
<?php
switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
       echo "i не равно 0, 1 или 2";
}
?>
```

Выражением в операторе *case* может быть любое выражение, которое приводится в простой тип, то есть в тип integer, или в тип с плавающей точкой (float), или строку. Массивы или объекты не могут быть здесь использованы до тех пор, пока они не будут разыменованы до простого типа.

Возможен альтернативный синтаксис для управляющей структуры switch. Для более детальной информации, см. Альтернативный синтаксис для управляющих структур.

```
<?php
switch ($i):
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
        echo "i не равно to 0, 1 или 2";
endswitch;
?>
```

Возможно использование точки с запятой вместо двоеточия после оператора case. К примеру :

```
<?php
switch($beer)
{
    case 'tuborg';
    case 'carlsberg';
    case 'heineken';
        echo 'Хороший выбор';
    break;
    default;
        echo 'Пожалуйста, сделайте новый выбор...';
    break;
}
?>
```

User Contributed Notes 59 notes

declare ¶

(PHP 4, PHP 5, PHP 7)

Конструкция *declare* используется для установки директив исполнения для блока кода. Синтаксис *declare* схож с синтаксисом других конструкций управления исполнением:

```
declare (directive)
    statement
```

Секция *directive* позволяет установить поведение блока *declare*. В настоящее время распознаются только две директивы: директива*ticks* (Дополнительная информация о директиве ticks доступна ниже) и директива *encoding* (Дополнительная информация о директиве encoding доступна ниже).

Замечание: Директива encoding была добавлена в PHP 5.3.0

Так как директива обрабатывается при компиляции файла, то только символьные данные могут использоваться как значения директивы. Нельзя использовать переменные и константы. Пример:

```
<?php
// Правильно:
declare(ticks=1);

// Недопустимо:
const TICK_VALUE = 1;
declare(ticks=TICK_VALUE);
?>
```

Часть *statement* блока *declare* будет исполнена - как исполняется и какие сторонние эффекты возникают в процессе исполнения может зависеть от набора директив в блоке *directive*.

Конструкция *declare* может также быть использована в глобальной области видимости, влияя на весь следующий за ней код (однако если файл с *declare* был включен, тогда он не имеет воздействия на родительский файл).

```
<?php
// выполняется одинаково:

// можно так:
declare(ticks=1) {
    // прочие действия
}

// или так:
declare(ticks=1);
// прочие действия
?>
```

Тики ¶

Тик - это событие, которое случается каждые `N` низкоуровневых операций, выполненных парсером внутри блока *declare*. Значение`N` задается, используя `ticks=N` внутри секции *directive* блока *declare*.

Не все выражения подсчитываются. Обычно, условные выражения и выражения аргументов не подсчитываются.

Событие (или несколько событий), которое возникает на каждом тике определяется, используя register_tick_function(). Смотрите пример ниже для дополнительной информации. Имейте в виду, что для одного тика может возникать несколько событий.

Пример #1 Пример использования тика

```
<?php

declare(ticks=1);

// Функция, исполняемая при каждом тике
function tick_handler()
{
    echo "tick_handler() выполнено\n";
}

register_tick_function('tick_handler');

$a = 1;

if ($a > 0) {
    $a += 2;
    print($a);
}

?>
```

Пример #2 Пример использования тиков

```
<?php

function tick_handler()
{
  echo "tick_handler() called\n";
}

$a = 1;
tick_handler();

if ($a > 0) {
    $a += 2;
    tick_handler();
    print($a);
    tick_handler();
}
tick_handler();

?>
```

См. также register_tick_function() и unregister_tick_function().

Кодировка ¶

Кодировка скрипта может быть указана для каждого скрипта используя директиву encoding.

Пример #3 Определение кодировки для скрипта.

```
<?php
declare(encoding='ISO-8859-1');
// прочий код
?>
```

Предостережение
Скомбинированный с пространством имен, единственно допустимый синтаксис для declare является*declare(encoding='...');* где *...* это значение кодировки. *declare(encoding='...') {}* приведет к ошибке парсера, если используется вместе с пространством имен.

Значение кодировки в declare игнорируется в PHP 5.3 если php не скомпилирован с *--enable-zend-multibyte*.

Имейте в виду, что PHP не может показать был ли он скомпилирован с *--enable-zend-multibyte* иначе как с помощью phpinfo().

Смотрите также zend.script_encoding.

User Contributed Notes 20 notes

return ¶
(PHP 4, PHP 5, PHP 7)

return возвращает управление программой в вызывавший модуль. Выполнение возвращается в выражение, следующее после вызова текущего модуля.

Если вызвано из функции, выражение *return* немедленно прекращает выполнение текущей функции и возвращает свой аргумент как значение данной функции. *return* также завершает выполнение выражения eval() или всего файла скрипта.

Если вызывается из глобальной области видимости, выполнение текущего файла скрипта прекращается. Если текущий файл скрипта был подключен с помощью функций include или require, тогда управление возвращается к файлу, который вызывал текущий. Более того, если текущий файл скрипта был подключен с помощью include, тогда значение переданное *return* будет возвращено в качестве значения вызова include. Если *return* вызывается из главного файла скрипта, тогда выполнение скрипта прекращается. Если текущий файл скрипта был вызван конфигурационными опциями auto_prepend_file или auto_append_file из файла настроек `php.ini`, тогда выполнение этого скрипта прекращается.

Для более детальной информации смотрите раздел Возвращаемые значения.

Замечание: Заметьте, что так как *return* является языковой конструкцией, а не функцией, круглые скобки, окружающие аргументы, не являются необходимостью. Общепринято не использовать их в данном случае и, в принципе, так и надо делать, т.к. PHP будет меньше работы по синтаксическому разбору файла в данном случае.
Замечание: Если параметры не указаны, тогда круглые скобки должны быть опущены, и вернется значение **`NULL`**. Вызов *return* со скобками, но без аргументов вызовет синтаксическую ошибку.
Замечание: Вы *не должны* использовать скобки при возврате переменной, если возвращаете по ссылке, так как это не сработает. Вы можете вернуть только переменную по ссылке, а не результат выражения. Если вы используете *return ($a);*, тогда вы вернете не переменную, а результат выражения *($a)* (который, естественно, будет являться значением переменной $a).

User Contributed Notes 9 notes

require ¶
(PHP 4, PHP 5, PHP 7)

require идентично include за исключением того, что при ошибке оно также выдаст фатальную ошибку уровня **`E_COMPILE_ERROR`**. Другими словами, она остановит выполнение скрипта, тогда как include только выдала бы предупреждение **`E_WARNING`**, которое позволило бы скрипту продолжить выполнение.

Смотрите документацию по include, чтобы узнать как это работает.

User Contributed Notes 25 notes

67
chris at chrisstockton dot org ¶

9 years ago

```
Remember, when using require that it is a statement, not a func-
tion. It's not necessary to write:
```

```
<?php
require('somefile.php');
?>

The following:
<?php
require 'somefile.php';
?>

Is preferred, it will prevent your peers from giving you a hard
time and a trivial conversation about what require really is.
```

include ¶

(PHP 4, PHP 5, PHP 7)

Выражение *include* включает и выполняет указанный файл.

Документация ниже также относится к выражению require.

Файлы включаются исходя из пути указанного файла, или, если путь не указан, используется путь, указанный в директивеinclude_path. Если файл не найден в include_path, *include* попытается проверить директорию, в которой находится текущий включающий скрипт и текущую рабочую директорию перед тем, как выдать ошибку. Конструкция *include* выдаст warning, если не сможет найти файл; поведение отлично от require, который выдаст фатальную ошибку.

Если путь указан — не важно, абсолютный (начинающийся с буквы диска или с \ в Windows или с / в Unix/Linux системах) или относительно текущей директории (начинающийся с *.* или *..*) — директива include_path будет проигнорирована в любом случае. К примеру, если начинается с *../*, парсер проверит родительскую директорию на наличие запрошенного файла.

Для большей информации о том, как PHP обрабатывает включаемые файлы и включаемые пути, смотрите документацию для директивы include_path.

Когда файл включается, его код наследует ту же область видимости переменных, что и строка, на которой произошло включение. Все переменные, доступные на этой строке во включающем файле будут также доступны во включаемом файле. Однако все функции и классы, объявленные во включаемом файле, будут доступны в глобальной области видимости.

Пример #1 Простой пример *include*

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>

test.php
<?php
```

```
echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

Если включение происходит внутри функции включающего файла, тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции. То есть, он будет в той же области видимости переменных этой функции. Исключением к этому правилу являются магические константы, которые выполняются парсером перед тем, как происходит включение.

Пример #2 Включение внутри функции

```
<?php

function foo()
{
    global $color;

    include 'vars.php';

    echo "A $color $fruit";
}

/* vars.php в той же области видимости, что и foo(),   *
* поэтому $fruit НЕ доступен снаружи этой области     *
* $color доступен, поскольку мы переменную глобальной */

foo();                    // A green apple
echo "A $color $fruit";   // A green

?>
```

Когда файл включается, парсинг в режиме PHP кода прекращается и переключается в режим HTML в начале указанного файла и продолжается снова в конце. По этой причине любой код внутри включаемого файла, который должен быть выполнен как PHP код, должен быть заключен в верные теги начала и конца PHP кода.

Если "обертки URL include" включены в PHP, вы можете также указать файл для включения через URL (с помощью HTTP или других поддерживающихся обработчиков - смотри Поддерживаемые протоколы и обработчики (wrappers) для списка протоколов) вместо локального пути. Если целевой сервер интерпретирует указанный файл как PHP код, могут быть переданы переменные во включаемый файл с помощью строки URL-запроса, как если бы использовался метод HTTP GET. Это, строго говоря, не тоже самое, что включение файла и наследование родительской области видимости; скрипт выполняется на удаленном сервере и результат выполнения включается в локальный скрипт.

Внимание
Версии PHP для Windows до PHP 4.3.0 не поддерживают возможность использования удаленных файлов этой функцией даже в том случае, если включена опция allow_url_fopen.

Пример #3 Пример *include* через HTTP

```
<?php

/* В этом примере предполагается, что www.example.com настроен н
а обработку .php
* файлов, но не .txt. Также, 'Works' обозначает, что переменные
* $foo и $bar доступны внутри включаемого файла. */

// Не сработает; file.txt не обрабатывается www.example.com как
PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Не сработает; будет искаать файл 'file.php?foo=1&bar=2' в
// локальной файловой системе.
include 'file.php?foo=1&bar=2';

// Сработает.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt';  // Сработает.
include 'file.php';  // Сработает.

?>
```

Внимание

Предупреждение безопасности

Удаленные файлы могут быть обработаны на удаленной стороне (в зависимости от расширения файла и того, что удаленный сервер выполняет скрипты PHP или нет), но это все равно должно производить валидный PHP скрипт, потому что он будет затем обработан уже на локальном сервере. Если файл с удаленного сервера должен быть обработан там и выведен его результат, предпочтительнее воспользоваться функцией readfile() В противном случае, должны быть предусмотрены дополнительные меры, чтобы обезопасить удаленный скрипт от ошибок и нежелательного кода.

Смотрите также раздел Удаленные файлы, функции fopen() и file() для дополнительной информации.

Обработка возвращаемых значений: оператор *include* возвращает значение *FALSE* при ошибке и выдает предупреждение. Успешные включения, пока это не переопределено во включаемом файле, возвращают значение *1*. Возможно выполнить выражение returnвнутри включаемого файла, чтобы завершить процесс выполнения в этом файле и вернуться к выполнению включающего файла. Также, возможно вернуть значение из включаемых файлов. Вы можете получить

значение включения как если бы вы вызвали обычную функцию. Хотя это не возможно при включении удаленного файла, только если вывод удаленного файла не содержитправильные теги начала и конца PHP кода (так же, как и локальный файл). Вы можете определить необходимые переменные внутри этих тегов и они будут представлены в том месте, где файл был включен.

Так как *include* - это специальная языковая конструкция, круглые скобки не обязательны для заключения аргумента. Будьте осторожны при сравнении возвращаемого значения.

Пример #4 Сравнение возвращаемого значения при include

```
<?php
// не сработает, интерпретируется как include(('vars.php') == 'O
K'), т.е. include('')
if (include('vars.php') == 'OK') {
    echo 'OK';
}

// сработает
if ((include 'vars.php') == 'OK') {
    echo 'OK';
}
?>
```

Пример #5 Выражения *include* и return

```
return.php
<?php

$var = 'PHP';

return $var;

?>

noreturn.php
<?php

$var = 'PHP';

?>

testreturns.php
<?php

$foo = include 'return.php';

echo $foo; // выведет 'PHP'

$bar = include 'noreturn.php';

echo $bar; // выведет 1
```

```
?>
```

$bar имеет значение *1*, т.к. включение файла произошло успешно. Заметьте разницу между примерами сверху. Первый используетreturn внутри включаемого файла, тогда как второй не использует. Если файл не может быть включен, возвращается **FALSE** и возникает **E_WARNING**.

Если во включаемом файле определены функции, они могут быть использованы в главном файле вне зависимости от того, были ли они объявлены до return или после. Если файл включается дважды, PHP 5 выдаст фатальную ошибку, потому что функции уже были определены, в то время как PHP 4 не обратил бы внимания на функции, определенные после return. Рекомендуется использоватьinclude_once вместо того, чтобы проверять был ли уже файл включен ранее.

Другой путь "включить" PHP файл в переменную - это захватить вывод используя Функции контроля вывода с *include*. К примеру:

Пример #6 Использование буферизированного вывода для включения PHP файла в строку

```
<?php
$string = get_include_contents('somefile.php');

function get_include_contents($filename) {
    if (is_file($filename)) {
        ob_start();
        include $filename;
        return ob_get_clean();
    }
    return false;
}

?>
```

Для того, чтобы включать файлы автоматически в скрипты, обратите внимание на конфигурационные директивы auto_prepend_file иauto_append_file в `php.ini`.

Замечание: Поскольку это языковая конструкция, а не функция, она не может вызываться при помощи переменных функций.

Смотрите
также require, require_once, include_once, get_included_files(), readfile(), virtual() и include_path.

User Contributed Notes 53 notes

require_once ¶
(PHP 4, PHP 5, PHP 7)

Выражение *require_once* идентично require за исключением того, что PHP проверит, включался ли уже данный файл, и, если да, не будет включать его еще раз.

Смотри документацию include_once для информации по поведению *_once*, и чем она отличается от своих не-*_once* родственников.

User Contributed Notes 25 notes
bimal at sanjaal dot com ¶

5 years ago

```
If your code is running on multiple servers with different envi-
ronments (locations from where your scripts run) the following
idea may be useful to you:

a. Do not give absolute path to include files on your server.
b. Dynamically calculate the full path (absolute path)

Hints:
Use a combination of dirname(__FILE__) and subsequent calls to
itself until you reach to the home of your '/index.php'. Then,
attach this variable (that contains the path) to your included
files.

One of my typical example is:
```

```
<?php
define('__ROOT__', dirname(dirname(__FILE__)));
require_once(__ROOT__.'/config.php');
?>
```

```
instead of:
<?php require_once('/var/www/public_html/config.php'); ?>

After this, if you copy paste your codes to another servers, it
will still run, without requiring any further re-
configurations.

[EDIT BY danbrown AT php DOT net: Contains a typofix (missing
')') provided by 'JoeB' on 09-JUN-2011.]
```

include_once ¶

(PHP 4, PHP 5, PHP 7)

Выражение *include_once* включает и выполняет указанный файл во время выполнения скрипта. Его поведение идентично выражениюinclude, с той лишь разницей, что если код из файла уже один раз был включен, он не будет включен и выполнен повторно. Как видно из имени, он включит файл только один раз (include once).

include_once может использоваться в тех случаях, когда один и тот же файл может быть включен и выполнен более одного раза во время выполнения скрипта, в данном случае это поможет избежать проблем с переопределением функций, переменных и т.д.

Смотри документацию по include для информации как эта функция работает.

Замечание:

В PHP 4, функциональность *_once* отличалась в регистронезависимых операционных системах (таких как Windows), к примеру:
Пример #1 Пример *include_once* в регистронезависимых ОС для PHP 4

```
<?php
include_once "a.php"; // это подключит a.php
include_once "A.php"; // это подключит a.php снова! (только в PHP 4)
?>
```

Это поведение изменилось в PHP 5. К примеру, пути в Windows в начале нормализуются так, чтобы `C:\PROGRA~1\A.php` обозначало тоже самое, что и `C:\Program Files\a.php`, и файл подключался лишь один раз.

User Contributed Notes 6 notes

goto ¶

(PHP 5 >= 5.3.0, PHP 7)

Изображение предоставлено » xkcd

Оператор *goto* используется для перехода в другую часть программы. Место, куда необходимо перейти указывается с помощью метки, за которой ставится двоеточие, после оператора *goto* указывается желаемая метка для перехода. Оператор не является неограниченным "*goto*". Целевая метка должна находиться в том же файле, в том же контексте. Имеется ввиду, что вы не можете ни перейти за границы функции или метода, ни перейти внутрь одной из них. Вы также не можете перейти внутрь любой циклической структуры или оператора switch. Но вы можете выйти из них, и обычным применением оператора *goto* является использование его вместо многоуровневых *break*.

Пример #1 Пример использования *goto*

```
<?php
goto a;
echo 'Foo';

a:
echo 'Bar';
?>
```

Результат выполнения данного примера:

```
Bar
```

Пример #2 Пример использования *goto* в цикле

```
<?php
for($i=0,$j=50; $i<100; $i++) {
  while($j--) {
    if($j==17) goto end;
  }
}
echo "i = $i";
end:
echo 'j hit 17';
?>
```

Результат выполнения данного примера:

```
j hit 17
```

Пример #3 Следующий пример работать не будет

```
<?php
goto loop;
for($i=0,$j=50; $i<100; $i++) {
  while($j--) {
    loop:
  }
}
echo "$i = $i";
?>
```

Результат выполнения данного примера:

```
Fatal error: 'goto' into loop or switch statement is disallowed
in
script on line 2
```

Замечание:
Оператор *goto* доступен в PHP начиная с версии 5.3.

User Contributed Notes 11 notes

chrisstocktonaz at gmail dot com ¶

7 years ago

Remember if you are not a fan of wild labels hanging around you are free to use braces in this construct creating a slightly cleaner look. Labels also are always executed and do not need to be called to have their associated code block ran. A purposeless example is below.

```
<?php

$headers = Ar-
ray('subject', 'bcc', 'to', 'cc', 'date', 'sender');
$position = 0;

hIterator: {

    $c = 0;
```

```
    echo $headers[$position] . PHP_EOL;

    cIterator: {
        echo ' ' . $headers[$position][$c] . PHP_EOL;

        if(!isset($headers[$position][++$c])) {
            goto cIteratorExit;
        }
        goto cIterator;
    }

    cIteratorExit: {
        if(isset($headers[++$position])) {
            goto hIterator;
        }
    }
}
?>
```

Функции ¶

Содержание ¶

User Contributed Notes 1 note

m_ilyas at outlook dot com ¶

6 months ago

```
\\Here is simplest php function example
<?php
    function swap($a,$b){
        print nl2br("Before swapping\n");
        print nl2br("a = $a and b = $b \n\n\n");
        $a = $a + $b; // 30 = 10 + 20
        $b = $a - $b;
        $a = $a - $b;
        print nl2br("After swapping\n");
        print nl2br("a = $a and b = $b \n");
    }
    swap(10,20);

?>
```

Функции, определяемые пользователем ¶

Приведем пример синтаксиса, используемого для описания функций:

Пример #1 Псевдокод для демонстрации использования функций

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Внутри функции можно использовать любой корректный PHP-код, в том числе другие функции и даже объявления классов.

Имена функций следуют тем же правилам, что и другие метки в PHP. Корректное имя функции начинается с буквы или знака подчеркивания, за которым следует любое количество букв, цифр или знаков подчеркивания. В качестве регулярного выражения оно может быть выражено так: *[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**.

Подсказка
Смотрите также Руководство по именованию.

Функции не обязаны быть определены до их использования, *исключая* тот случай, когда функции определяются условно, как это показано в двух последующих примерах.

В случае, когда функция определяется в зависимости от какого-либо условия, например, как это показано в двух приведенных ниже примерах, обработка описания функции должна *предшествовать* ее вызову.

Пример #2 Функции, зависящие от условий

```
<?php

$makefoo = true;

/* Мы не можем вызвать функцию foo() в этом месте,
   поскольку она еще не определена, но мы можем
   обратиться к bar() */

bar();

if ($makefoo) {
  function foo()
  {
    echo "Я не существую до тех пор, пока выполнение программы м
еня не достигнет.\n";
  }
}

/* Теперь мы благополучно можем вызывать foo(),
   поскольку $makefoo была интерпретирована как true */

if ($makefoo) foo();
```

```php
function bar()
{
  echo "Я существую сразу с начала старта программы.\n";
}

?>
```

Пример #3 Вложенные функции

```php
<?php
function foo()
{
  function bar()
  {
    echo "Я не существую пока не будет вызвана foo().\n";
  }
}

/* Мы пока не можем обратиться к bar(),
   поскольку она еще не определена. */

foo();

/* Теперь мы можем вызвать функцию bar(),
   обработка foo() сделала ее доступной. */

bar();

?>
```

Все функции и классы PHP имеют глобальную область видимости - они могут быть вызваны вне функции, даже если были определены внутри и наоборот.

PHP не поддерживает перегрузку функции, также отсутствует возможность переопределить или удалить объявленную ранее функцию.

Замечание: Имена функций регистронезависимы, тем не менее, более предпочтительно вызывать функции так, как они были объявлены.

Функции PHP поддерживают как списки аргументов переменной длины, так и значения аргументов по умолчанию. Смотрите также описания функций func_num_args(), func_get_arg(), и func_get_args() для более детальной информации.

Можно вызывать функции PHP рекурсивно.

Пример #4 Рекурсивные функции

```php
<?php
function recursion($a)
{
    if ($a < 20) {
```

```
        echo "$a\n";
        recursion($a + 1);
    }
}
?>
```

Замечание: Рекурсивный вызов методов/процедур с глубиной более 100-200 уровней рекурсии может вызвать переполнение стека и привести к аварийному завершению скрипта. В частности, бесконечная рекурсия будет считаться программной ошибкой.

User Contributed Notes 10 notes

Аргументы функции ¶

Функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми выражений. Аргументы вычисляются слева направо.

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию.Списки аргументов переменной длины также поддерживаются, смотрите также описания функций func_num_args(), func_get_arg() иfunc_get_args() для более детальной информации.

Пример #1 Передача массива в функцию

```
<?php
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

Передача аргументов по ссылке ¶

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

Если вы хотите, что бы аргумент всегда передавался по ссылке, вы можете указать амперсанд (&) перед именем аргумента в описании функции:

Пример #2 Передача аргументов по ссылке

```
<?php
function add_some_extra(&$string)
{
    $string .= 'и кое-что еще.';
}
$str = 'Это строка, ';
add_some_extra($str);
```

```
echo $str;     // выведет 'Это строка, и кое-что еще.'
?>
```

Значения аргументов по умолчанию ¶

Функция может определять значения по умолчанию в стиле C++ для скалярных аргументов, например:

Пример #3 Использование значений по умолчанию в определении функции

```
<?php
function makecoffee($type = "капуччино")
{
    return "Готовим чашку $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("эспрессо");
?>
```

Результат выполнения данного примера:

```
Готовим чашку капуччино.
Готовим чашку .
Готовим чашку эспрессо.
```

PHP также позволяет использовать массивы (array) и специальный тип **NULL** в качестве значений по умолчанию, например:

Пример #4 Использование нескалярных типов в качестве значений по умолчанию

```
<?php
func-
tion makecoffee($types = array("капуччино"), $coffeeMaker = NULL
)
{
    $device = is_null($coffeeMaker) ? "вручную" : $coffeeMaker;
    return "Готовлю чашку ".join(", ", $types)." $device.\n";
}
echo makecoffee();
echo makecoffee(array("капуччино", "лавацца"), "в чайнике");
?>
```

Значение по умолчанию должно быть константным выражением, а не (к примеру) переменной или вызовом функции/метода класса.

Обратите внимание, что все аргументы, для которых установлены значения по умолчанию, должны находиться правее аргументов, для которых значения по умолчанию не заданы, в противном случае ваш код может работать не так, как вы этого ожидаете. Рассмотрим следующий пример:

Пример #5 Некорректное использование значений по умолчанию

```php
<?php
function makeyogurt($type = "ацидофил", $flavour)
{
    return "Готовим чашку из бактерий $type со вкусом $flavour.\
n";
}

echo makeyogurt("малины");   // Не будет работать так, как мы мо
гли бы ожидать
?>
```

Результат выполнения данного примера:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/phptest/functest.html on line 41
Готовим чашку из бактерий малины со вкусом .
```

Теперь сравним его со следующим примером:

Пример #6 Корректное использование значений по умолчанию

```php
<?php
function makeyogurt($flavour, $type = "ацидофил")
{
    return "Готовим чашку из бактерий $type со вкусом $flavour.\
n";
}

echo makeyogurt("малины");   // отрабатывает правильно
?>
```

Результат выполнения данного примера:

```
Готовим чашку из бактерий ацидофил со вкусом малины.
```

Замечание: Начиная с PHP 5, значения по умолчанию могут быть переданны по ссылке.

Объявление типов ¶

Замечание:
Объявление типов также известно, как подсказки для типов в PHP 5.

Объявления типов позволяют функциям строго задавать тип передаваемых параметров. Передача в функцию значений несоответствующего типа будет приводить к ошибке: в PHP 5 это будет обрабатываемая фатальная ошибка, а в PHP 7 будет выбрасываться исключение TypeError.

Чтобы объявить тип агрумента, необходимо перед его именем добавить имя требуемого типа. Также можно объявить тип **NULL**, чтобы указать, что значением по умолчанию аргумента является **NULL**.

Valid types ¶

Тип	Описание	Минимальная версия PHP

Тип	Описание	Минимальная версия PHP
Имя класса/интерфейса	Аргумент должен быть *instanceof*, что и имя класса или интерфейса.	PHP 5.0.0
array	Аргумент должен быть типа array.	PHP 5.1.0
callable	Аргумент должен быть корректным callable типом.	PHP 5.4.0
bool	Аргумент должен быть типа boolean.	PHP 7.0.0
float	Аргумент должен быть float типа.	PHP 7.0.0
int	Аргумент должен быть типа integer.	PHP 7.0.0
string	Аргумент должен иметь тип string.	PHP 7.0.0

Примеры ¶

Пример #7 Основные объявления типов-классов

```
<?php
class C {}
class D extends C {}

// Это не является расширением класса C.
class E {}

function f(C $c) {
    echo get_class($c)."\n";
}

f(new C);
f(new D);
f(new E);
?>
```

Результат выполнения данного примера:

```
C
D

Fatal error: Uncaught TypeError: Argument 1 passed to f() must
be an instance of C, instance of E given, called in - on line 14
and defined in -:8
Stack trace:
#0 -(14): f(Object(E))
#1 {main}
  thrown in - on line 8
```

Пример #8 Основные объявления типов-интерфейсов

```
<?php
interface I { public function f(); }
class C implements I { public function f() {} }

// Это не реализует интерфейс I.
```

```php
class E {}

function f(I $i) {
    echo get_class($i)."\n";
}

f(new C);
f(new E);
?>
```

Результат выполнения данного примера:

```
C

Fatal error: Uncaught TypeError: Argument 1 passed to f() must implement interface I, instance of E given, called in - on line 13 and defined in -:8
Stack trace:
#0 -(13): f(Object(E))
#1 {main}
  thrown in - on line 8
```

Пример #9 Объявление типа Null

```php
<?php
class C {}

function f(C $c = null) {
    var_dump($c);
}

f(new C);
f(null);
?>
```

Результат выполнения данного примера:

```
object(C)#1 (0) {
}
NULL
```

Строгая типизация ¶

По умолчанию, PHP будет пытаться привести значения несоответствующих типов к скалярному типу, если это возможно. Например, если в функцию передается integer, а тип аргумента объявлен string, в итоге функция получит преобразованное string значение.

Для отдельных файлов можно включать режим строгой типизации. В этом режиме в функцию можно передавать значения только тех типов, которые объявлены для аргументов. В противном случае будет выбрасываться исключение TypeError. Есть лишь одно исключение - integer можно передать в функцию, которая ожидает значение типа float.

Для влючения режима строгой типизации используется выражение *declare* в объявлении *strict_types*:

Предостережение
Включение режима строгой типизации также повлияет на объявления типов возвращаемых значений.

Замечание:
Режим строгой типизации распространяется на вызовы функций совершенные *из* файла, в котором этот режим включен, а не на функции, которые в этом файле объявлены. Если файл без строгой типизации вызывает функцию, которая объявлена в файле с включенным режимом, значения аргументов будут приведены к нужным типам и ошибок не последует.

Замечание:
Строгая типизация применима только к скалярным типам и работает только в PHP 7.0.0 и выше. Равно как и сами объявления скалярных типов добавлены в этой версии.

Пример #10 Строгая типизация

```php
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1.5, 2.5));
?>
```

Результат выполнения данного примера:

```
int(3)

Fatal error: Uncaught TypeError: Argument 1 passed to sum() must be of the type integer, float given, called in - on line 9 and defined in -:4
Stack trace:
#0 -(9): sum(1.5, 2.5)
#1 {main}
  thrown in - on line 4
```

Пример #11 Слабая типизация

```php
<?php
function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));

// These will be coerced to integers: note the output below!
```

```
var_dump(sum(1.5, 2.5));
?>
```

Результат выполнения данного примера:

```
int(3)
int(3)
```

Пример #12 Обработка исключения TypeError

```
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

try {
    var_dump(sum(1, 2));
    var_dump(sum(1.5, 2.5));
} catch (TypeError $e) {
    echo 'Error: '.$e->getMessage();
}
?>
```

Результат выполнения данного примера:

```
int(3)
Error: Argument 1 passed to sum() must be of the type integer,
float given, called in - on line 10
```

Списки аргументов переменной длины ¶

PHP поддерживает списки аргументов переменной длины для функций, определяемых пользователем. Для версий PHP 5.6 и выше это делается добавлением многоточия (...). Для версий 5.5 и старше используются функции func_num_args(), func_get_arg() иfunc_get_args().

... *в PHP 5.6+* ¶

В версиях PHP 5.6 и выше список аргументов может содержать многоточие ..., чтобы показать, что функция принимает переменное количество аргументов. Аргументы в этом случае будут переданы в виде массива. Например:

Пример #13 Использование ... для доступа к аргументам

```
<?php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
```

```
}

echo sum(1, 2, 3, 4);
?>
```

Результат выполнения данного примера:

```
10
```

Многоточие (...) можно использовать при вызове функции, чтобы распаковать массив (array) или Traversable переменную в список аргументов:

Пример #14 Использование ... для передачи аргументов

```
<?php
function add($a, $b) {
    return $a + $b;
}

echo add(...[1, 2])."\n";

$a = [1, 2];
echo add(...$a);
?>
```

Результат выполнения данного примера:

```
3
3
```

Можно задать несколько аргументов в привычном виде, а затем добавить .... В этом случае ... поместит в массив только те аргументы, которые не нашли соответствия указанным в объявлении функции.

Также можно добавить подсказку типа перед .... В этом случае PHP будет следить, чтобы все аргументы обработанные многоточием (...) были того же типа, что указан в подсказке.

Пример #15 Аргументы с подсказкой типов

```
<?php
function total_intervals($unit, DateInterval ...$intervals) {
    $time = 0;
    foreach ($intervals as $interval) {
        $time += $interval->$unit;
    }
    return $time;
}

$a = new DateInterval('P1D');
$b = new DateInterval('P2D');
echo total_intervals('d', $a, $b).' days';
```

```
// This will fail, since null isn't a DateInterval object.
echo total_intervals('d', null);
?>
```

Результат выполнения данного примера:

```
3 days
Catchable fatal error: Argument 2 passed to total_intervals()
must be an instance of DateInterval, null given, called in - on
line 14 and defined in - on line 2
```

В конце концов, можно передавать аргументы по ссылке. Для этого перед ... нужно поставить амперсанд (&).

Предыдущие версии PHP ¶

Для указания того, что функция принимает переменное число аргументов, никакой специальный синтаксис не используется. Для доступа к аргументам необходимо использовать функции func_num_args(), func_get_arg() и func_get_args().

В первом примере выше было показано, как задать список аргументов переменной длины для версий PHP 5.5 и более ранних:

Пример #16 Доступ к аргументам в PHP 5.5 и ранних версий

```
<?php
function sum() {
    $acc = 0;
    foreach (func_get_args() as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);
?>
```

Результат выполнения данного примера:

```
10
```

User Contributed Notes 39 notes

Возврат значений ¶

Значения возвращаются при помощи необязательного оператора возврата. Возвращаемые значения могут быть любого типа, в том числе это могут быть массивы и объекты. Возврат приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой данная функция была вызвана. Для получения более детальной информации ознакомьтесь с описаниемreturn.

Замечание:
Если конструкция return не указана, то функция вернет значение **NULL**.

Использование выражения return

Пример #1 Использование конструкции return

```
<?php
function square($num)
{
    return $num * $num;
}
echo square(4);   // выводит '16'.
?>
```

Функция не может возвращать несколько значений, но аналогичного результата можно добиться, возвращая массив.

Пример #2 Возврат нескольких значений в виде массива

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

Для того, чтобы функция возвращала результат по ссылке, вам необходимо использовать оператор & и при описании функции, и при присвоении переменной возвращаемого значения:

Пример #3 Возврат результата по ссылке

```
<?php
function &returns_reference()
{
    return $someref;
}

$newref =& returns_reference();
?>
```

Для получения более детальной информации о ссылках обратитесь к разделу документации Подробно о ссылках.

Объявление типов возвращаемых значений ¶

В PHP 7 добавлена возможность объявлять тип возвращаемого значения. Аналогично объявлению типов аргументов можно задать тип значения, которое будет возвращаться функцией. Типы, которые можно объявить для возвращаемых значений те же, что и для аргументов фукнций.

Режим строгой типизации также работает для объявлении типа возвращаемого значения. В обычном режиме слабой типизации возвращаемое из функции значе-

ние приводится к корректному типу. При строгой типизации возвращаемое значение должно быть заданного типа, иначе будет выброшено исключение TypeError.

Замечание:
Если переопределяется родительский метод, возвращаемое значение дочернего метода должно быть того же типа, что и родительского. Если в родительском методе не задан тип возвращаемого значения, то и дочерний метод этот тип может не объявлять.

Примеры ¶

Пример #4 Обычное объявление типа возвращаемого значения

```
<?php
function sum($a, $b): float {
    return $a + $b;
}

// Будет возвращаться значение типа float.
var_dump(sum(1, 2));
?>
```

Результат выполнения данного примера:

```
float(3)
```

Пример #5 То же в режиме строгой типизации

```
<?php
declare(strict_types=1);

function sum($a, $b): int {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1, 2.5));
?>
```

Результат выполнения данного примера:

```
int(3)

Fatal error: Uncaught TypeError: Return value of sum() must be
of the type integer, float returned in - on line 5 in -:5
Stack trace:
#0 -(9): sum(1, 2.5)
#1 {main}
  thrown in - on line 5
```

Пример #6 Возврат объектов

```php
<?php
class C {}

function getC(): C {
    return new C;
}

var_dump(getC());
?>
```

Результат выполнения данного примера:

```
object(C)#1 (0) {
}
```

User Contributed Notes 9 notes

Обращение к функциям через переменные ¶

PHP поддерживает концепцию переменных функций. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной, и пытается ее выполнить. Эту возможность можно использовать для реализации обратных вызовов, таблиц функций и множества других вещей.

Переменные функции не будут работать с такими языковыми конструкциями как echo, print, unset(), isset(), empty(), include, require и другими подобными им операторами. Вам необходимо реализовывать свою функцию-обертку (wrapper) для того, чтобы приведенные выше конструкции могли работать с переменными функциями.

Пример #1 Работа с функциями посредством переменных

```php
<?php
function foo() {
    echo "In foo()<br />\n";
}

function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br />\n";
}

// Функция-обертка для echo
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // Вызывает функцию foo()

$func = 'bar';
```

```
$func('test');  // Вызывает функцию bar()

$func = 'echoit';
$func('test');  // Вызывает функцию echoit()
?>
```

Вы также можете вызвать методы объекта, используя возможности PHP для работы с переменными функциями.

Пример #2 Обращение к методам класса посредством переменных

```
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // Вызываем метод Bar()
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // Обращаемся к $foo->Variable()

?>
```

При вызове статических методов, вызов функции "сильнее" чем оператор доступа к статическому свойству:

Пример #3 Пример вызова переменного метода со статическим свойством

```
<?php
class Foo
{
    static $variable = 'static property';
    static function Variable()
    {
        echo 'Method Variable called';
    }
}

echo Foo::$variable; // Это выведет 'static property'. Переменна
я $variable будет разрешена в нужной области видимости.
$variable = "Variable";
Foo::$variable();  // Это вызовет $foo-
>Variable(), прочитав $variable из этой области видимости.
```

```
?>
```

С версии PHP 5.4.0, можно вызывать callable функцию помещенную в переменную.

Пример #4 Callable-фукнции

```
class Foo
{
    static function bar()
    {
        echo "bar\n";
    }
    function baz()
    {
        echo "baz\n";
    }
}

$func = array("Foo", "bar");
$func(); // prints "bar"
$f = array(new Foo, "baz");
$func(); // prints "baz"
$f = "Foo::bar";
$func(); // выведет "bar" в PHP 7.0.0 и выше; в предыдущих версиях это приведет к фатальной ошибке
```

Смотрите также is_callable(), call_user_func(), Переменные переменные и function_exists().

Список изменений

Версия	Описание
7.0.0	'ClassName::methodName' доступна как функция-переменная.
5.4.0	Массивы, являющиеся корректными callable-методами, доступны как функции-переменные.

User Contributed Notes 11 notes

Встроенные функции ¶

В самом PHP содержится достаточно большое количество встроенных функций и языковых конструкций. Также есть функции, которые требуют, чтобы PHP был собран со специфическими расширениями, в противном случае вы получите сообщение о фатальной ошибке, вызванной использованием неизвестной функции. Например, для того чтобы использовать функции для работы с изображениями, например, imagecreatetruecolor(), вам необходимо собрать PHP с поддержкой GD. Или же для того, чтобы воспользоваться функцией mysql_connect(), вам необходима поддержка модуля MySQL. Тем не менее, есть много встроенных функций, которые доступны всегда: например функции обработки строк и функции для работы с переменными. Вызвав phpinfo() илиget_loaded_extensions(), вы можете узнать, поддержка каких модулей есть в используемом вами PHP. Также следует учесть, что поддержка некоторых дополнительных расширений включена по

умолчанию, и что сама документация к PHP разбита по расширениям. Ознакомьтесь с разделами Конфигурация, Установка, а также с документацией непосредственно к дополнительным расширениям для получения более детальной информации о том, как настроить ваш PHP.

Более подробную информацию о том, как следует читать и интерпретировать прототипы функций, вы можете найти в разделе Как правильно читать описания функций. Очень важно понимать, что возвращает функция, или как именно она модифицирует передаваемые аргументы. Например, функция str_replace() возвращает модифицированную строку, в то время как функция usort()работает с фактически переданной переменной. Каждая страница документации также содержит информацию, которая специфична для данной функции, например, информацию о передаваемых параметрах, изменениях в поведении, возвращаемых значениях в случае как удачного, так и неудачного выполнения, доступности функции в различных версиях. Знание и применение этих (порой даже незаметных) нюансов очень важно для написания корректного PHP-кода.

Замечание: Если в функцию передаются не те аргументы, которые она ожидает, например, массив (array) вместо строки (string), возвращаемое значение функции не определено. Скорее всего в этом случае будет возвращен **NULL**, но это просто соглашение, на него нельзя полагаться.

Ознакомьтесь также с описанием функции function_exists(), справочником функций и функциями get_extension_funcs() и dl().

Анонимные функции ¶

Анонимные функции, также известные как замыкания (*closures*), позволяют создавать функции, не имеющие определенных имен. Они наиболее полезны в качестве значений callback-параметров, но также могут иметь и множество других применений.

Пример #1 Пример анонимной функции

```
<?php
echo preg_replace_callback('~-([a-z])~', function ($match) {
    return strtoupper($match[1]);
}, 'hello-world');
// выведет helloWorld
?>
```

Замыкания также могут быть использованы в качестве значений переменных; PHP автоматически преобразует такие выражения в экземпляры внутреннего класса Closure. Присвоение замыкания переменной использует тот же синтаксис, что и для любого другого присвоения, включая завершающую точку с запятой:

Пример #2 Пример присвоения анонимной функции переменной

```
<?php
$greet = function($name)
{
    printf("Hello %s\r\n", $name);
```

```
};

$greet('World');
$greet('PHP');
?>
```

Замыкания могут также наследовать переменные из родительской области видимости. Любая подобная переменная должна быть объявлена в конструкции *use*.

Пример #3 Наследование переменных из родительской области видимости

```
<?php
$message = 'hello';

// Без "use"
$example = function () {
    var_dump($message);
};
echo $example();

// Наследуем $message
$example = function () use ($message) {
    var_dump($message);
};
echo $example();

// Значение унаследованной переменной задано там, где функция определена,
// но не там, где вызвана
$message = 'world';
echo $example();

// Сбросим message
$message = 'hello';

// Наследование по ссылке
$example = function () use (&$message) {
    var_dump($message);
};
echo $example();

// Измененное в родительской области видимости значение
// остается тем же внутри вызова функции
$message = 'world';
echo $example();

// Замыкания могут принимать обычные аргументы
$example = function ($arg) use ($message) {
    var_dump($arg . ' ' . $message);
};
$example("hello");
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Notice: Undefined variable: message in /example.php on line 6
NULL
string(5) "hello"
string(5) "hello"
string(5) "hello"
string(5) "world"
string(11) "hello world"
```

Наследование переменных из родительской области видимости *не* то же самое, что использование глобальных переменных. Глобальные переменные существуют в глобальной области видимости, которая не меняется, вне зависимости от того, какая функция выполняется в данный момент. Родительская область видимости - это функция, в которой было объявлено замыкание (не обязательно та же самая, из которой оно было вызвано). Смотрите следующий пример:

Пример #4 Замыкания и область видимости

```php
<?php
// Базовая корзина покупок, содержащая список добавленных
// продуктов и количество каждого продукта. Включает метод,
// вычисляющий общую цену элементов корзины с помощью
// callback-замыкания.
class Cart
{
    const PRICE_BUTTER  = 1.00;
    const PRICE_MILK    = 3.00;
    const PRICE_EGGS    = 6.95;

    protected $products = array();

    public function add($product, $quantity)
    {
        $this->products[$product] = $quantity;
    }

    public function getQuantity($product)
    {
        return isset($this->products[$product]) ? $this-
>products[$product] :
               FALSE;
    }

    public function getTotal($tax)
    {
        $total = 0.00;

        $callback =
            function ($quantity, $product) use ($tax, &$total)
            {
                $pricePerItem = constant(__CLASS__ . "::PRICE_"
.
```

```
                    strtoupper($product));
                $to-
tal += ($pricePerItem * $quantity) * ($tax + 1.0);
            };

        array_walk($this->products, $callback);
        return round($total, 2);
    }
}

$my_cart = new Cart;

// Добавляем несколько элементов в корзину
$my_cart->add('butter', 1);
$my_cart->add('milk', 3);
$my_cart->add('eggs', 6);

// Выводим общую сумму с 5% налогом на продажу.
print $my_cart->getTotal(0.05) . "\n";
// Результатом будет 54.29
?>
```

Анонимные функции реализованы с помощью класса Closure.

Список изменений

Версия	Описание
5.4.0	Стало возможным использовать *$this* в анонимных функциях.
5.3.0	Появление анонимных функций.

Примечания

Замечание: Совместно с замыканиями можно использовать функции func_num_args(), func_get_arg() и func_get_args().

User Contributed Notes 48 notes

Классы и объекты ¶

Содержание ¶

- Anonymous classes
- Перегрузка
- Итераторы объектов
- Магические методы
- Ключевое слово "final"
- Клонирование объектов
- Сравнение объектов
- Контроль типа
- Позднее статическое связывание
- Объекты и ссылки
- Сериализация объектов
- Журнал изменений ООП

User Contributed Notes 9 notes

Jason ¶

8 years ago

For real quick and dirty one-liner anonymous objects, just cast an associative array:

```
<?php

$obj = (object) array('foo' => 'bar', 'property' => 'value');

echo $obj->foo; // prints 'bar'
echo $obj->property; // prints 'value'

?>
```

... no need to create a new class or function to accomplish it.

Введение ¶

Начиная с версии PHP 5 объектная модель была полностью переписана, она стала более производительной и функциональной. Это было главным изменением с версии PHP 4. В PHP 5 теперь полная объектная модель.

Среди добавленных возможностей в PHP 5 можно найти видимость, абстрактные и ненаследуемые (final) классы и методы, а такжемагические методы, интерфейсы, клонирование и контроль типов (typehinting).

PHP работает с объектами так же, как с ссылками или дескрипторами, это означает что каждая переменная содержит ссылку на объект, а не его копию. Более подробную информацию см. в разделе Объекты и ссылки.

Подсказка
Смотрите также Руководство по именованию.

Основы ¶

class ¶

Каждое определение класса начинается с ключевого слова *class*, затем следует имя класса, и далее пара фигурных скобок, которые заключают в себе определение свойств и методов этого класса.

Именем класса может быть любое слово, при условии, что оно не входит в список зарезервированных слов PHP, начинается с буквы или символа подчеркивания и за которым следует любое количество букв, цифр или символов подчеркивания. Если задать эти правила в виде регулярного выражения, то получится следующее выражение: *^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$*.

Класс может содержать собственные константы, переменные (называемые свойствами) и функции (называемые методами).

Пример #1 Простое определение класса

```
<?php
class SimpleClass
{
    // объявление свойства
    public $var = 'значение по умолчанию';

    // объявление метода
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Псевдо-переменная `$this` доступна в том случае, если метод был вызван в контексте объекта. `$this` является ссылкой на вызываемый объект. Обычно это тот объект, которому принадлежит вызванный метод, но может быть и другой объект, если метод был вызван статически из контекста другого объекта. Это показано на следующих примерах:

Пример #2 Переменная *$this*

```
<?php
class A
{
    function foo()
    {
        if (isset($this)) {
            echo '$this определена (';
            echo get_class($this);
            echo ")\n";
        } else {
            echo "\$this не определена.\n";
        }
    }
```

```
}

class B
{
    function bar()
    {
        // Замечание: следующая строка вызовет предупреждение, е
сли включен параметр E_STRICT.
        A::foo();
    }
}

$a = new A();
$a->foo();

// Замечание: следующая строка вызовет предупреждение, если вклю
чен параметр E_STRICT.
A::foo();
$b = new B();
$b->bar();

// Замечание: следующая строка вызовет предупреждение, если вклю
чен параметр E_STRICT.
B::bar();
?>
```

Результат выполнения данного примера:

```
$this определена (A)
$this не определена.
$this определена (B)
$this не определена.
```

new ¶

Для создания экземпляра класса используется директива *new*. Новый объект всегда будет создан, за исключением случаев, когда он содержит конструктор, в котором определен вызов исключения в случае ошибки. Рекомендуется определять классы до создания их экземпляров (в некоторых случаях это обязательно).

Если с директивой *new* используется строка (string), содержащая имя класса, то будет создан новый экземпляр этого класса. Если имя находится в пространстве имен, то оно должно быть задано полностью.

Пример #3 Создание экземпляра класса

```
<?php
$instance = new SimpleClass();

// Это же можно сделать с помощью переменной:
$className = 'Foo';
$instance = new $className(); // Foo()
?>
```

В контексте класса можно создать новый объект через *new self* и *new parent*.

Когда происходит присвоение уже существующего экземпляра класса новой переменной, то эта переменная будет указывать на этот же экземпляр класса. Тоже самое происходит и при передаче экземпляра класса в функцию. Копию уже созданного объекта можно создать через ее клонирование.

Пример #4 Присваивание объекта

```
<?php

$instance = new SimpleClass();

$assigned   =  $instance;
$reference  =& $instance;

$instance->var = '$assigned будет иметь это значение';

$instance = null; // $instance и $reference становятся null

var_dump($instance);
var_dump($reference);
var_dump($assigned);
?>
```

Результат выполнения данного примера:

```
NULL
NULL
object(SimpleClass)#1 (1) {
   ["var"]=>
     string(30) "$assigned будет иметь это значение"
}
```

В PHP 5.3.0 введены несколько новых методов создания экземпляров объекта:

Пример #5 Создание новых объектов

```
<?php
class Test
{
    static public function getNew()
    {
        return new static;
    }
}

class Child extends Test
{}

$obj1 = new Test();
$obj2 = new $obj1;
var_dump($obj1 !== $obj2);
```

```php
$obj3 = Test::getNew();
var_dump($obj3 instanceof Test);

$obj4 = Child::getNew();
var_dump($obj4 instanceof Child);
?>
```

Результат выполнения данного примера:

```
bool(true)
bool(true)
bool(true)
```

extends ¶

Класс может наследовать методы и свойства другого класса используя ключевое слово *extends* при его описании. Невозможно наследовать несколько классов, один класс может наследовать только один базовый класс.

Наследуемые методы и свойства могут быть переопределены (за исключением случаев, когда метод класса объявлен как final) путем объявления их с теми же именами, как и в родительском классе. Существует возможность доступа к переопределенным методам или статическим методам путем обращения к ним через parent::

Когда переопределяются методы число и типы аргументов должны оставаться такими же как и были, иначе PHP вызовет ошибку уровня **E_STRICT**. Это не относится к конструктору, который можно переопределить с другими параметрами.

Пример #6 Простое наследование классов

```php
<?php
class ExtendClass extends SimpleClass
{
    // Переопределение метода родителя
    function displayVar()
    {
        echo "Расширенный класс\n";
        parent::displayVar();
    }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
```

Результат выполнения данного примера:

```
Расширенный класс
значение по умолчанию
```

::class ¶

Начиная с версии PHP 5.5 можно использовать ключевое слово *class* для разрешения имени класса. С помощью конструкции*ClassName::class* можно получить строку с абсолютным именем класса *ClassName*. Обычно это довольно полезно при работе с классами, использующими пространства имен.

Пример #7 Разрешение имени класса

```
<?php
namespace NS {
    class ClassName {
    }

    echo ClassName::class;
}
?>
```

Результат выполнения данного примера:

```
NS\ClassName
```

User Contributed Notes 21 notes

Свойства ¶

Переменные, которые являются членами класса, называются "свойства". Также их называют, используя другие термины, такие как "атрибуты" или "поля", но в рамках этой документации, мы будем называть их свойствами. Они определяются с помощью ключевых слов *public*, *protected*, или *private*, следуя правилам правильного описания переменных. Это описание может содержать инициализацию, но инициализация должна применяться для константных значений - то есть, переменные должны быть вычислены во время компиляции и не должны зависеть от информации программы во время выполнения для их вычисления.

Смотри Область видимости для получения информации о применении *public*, *protected*, и *private*.

Замечание:
Для того, чтобы поддерживать обратную совместимость с PHP 4, PHP 5 по-прежнему позволяет использовать ключевое слово *var*при определении свойств вместо (или в дополнении к) *public*, *protected*, или *private*. Однако *var* больше не требуется. В версиях PHP с 5.0 по 5.1.3, использование *var* считалось устаревшим вызывало **E_STRICT** предупреждение, но с PHP 5.1.3 больше не считается устаревшим и не выдает предупреждения.

Если, для определения свойства, вы используете *var* вместо одного из: *public*, *protected*, или *private*, тогда PHP 5 будет определять свойство как *public*.

В пределах методов класса доступ к нестатическим свойствам может быть получен с помощью -> (объектного оператора):`$this->property` (где *property* - имя свойства). Доступ к статическим свойствам может быть получен с помо-

щью :: (двойного двоеточия): `self::$property`. Подробнее о различиях между статическими и нестатическими свойствами смотрите в разделе "Ключевое слово Static" для получения большей информации.

Псевдо-переменная `$this` доступна внутри любого метода класса, когда этот метод вызывается в пределах объекта. `$this` - это ссылка на вызываемый объект (обычно, объект, которому принадлежит метод, но возможно и другого объекта, если метод вызванстатически из контекста второго объекта).

Пример #1 Определение свойств

```
<?php
class SimpleClass
{
   // неправильное определение свойств:
   public $var1 = 'hello ' . 'world';
   public $var2 = <<<EOD
hello world
EOD;
   public $var3 = 1+2;
   public $var4 = self::myStaticMethod();
   public $var5 = $myVar;

   // правильное определение свойств:
   public $var6 = myConstant;
   public $var7 = array(true, false);

   // Это разрешено только в PHP 5.3.0 и более поздних версиях.
   public $var8 = <<<'EOD'
hello world
EOD;
}
?>
```

Замечание:
Существуют несколько интересных функций для обработки классов и объектов. Вы можете их увидеть тут Class/Object Functions.

В отличии от heredocs, nowdocs может быть использованы в любом статическом контексте данных, включая определение свойств.

Пример #2 Пример использования nowdoc для инициализации свойств

```
<?php
class foo {
   // As of PHP 5.3.0
   public $bar = <<<'EOT'
bar
EOT;
}
?>
```

Замечание:
Поддержка nowdoc была добавлена в PHP 5.3.0.

User Contributed Notes 12 notes

Константы классов ¶

Константы также могут быть объявлены и в пределах одного класса. Отличие переменных и констант состоит в том, что при объявлении последних или при обращении к ним не используется символ $.

Значение должно быть неизменяемым выражением, не (к примеру) переменной, свойством, результатом математической операции или вызовом функции.

Интерфейсы также могут содержать константы (*constants*). За примерами обращайтесь к разделу об интерфейсах.

Начиная с версии PHP 5.3.0, стало возможным обратиться к классу с помощью переменной. Значение переменной не должно быть ключевым словом (например, *self*, *parent* или *static*).

Пример #1 Объявление и использование константы

```
<?php
class MyClass
{
    const CONSTANT = 'значение константы';

    function showConstant() {
        echo  self::CONSTANT . "\n";
    }
}

echo MyClass::CONSTANT . "\n";

$classname = "MyClass";
echo $classname::CONSTANT . "\n"; // начиная с версии PHP 5.3.0

$class = new MyClass();
$class->showConstant();

echo $class::CONSTANT."\n"; // начиная с версии PHP 5.3.0
?>
```

Пример #2 Пример со статичными данными

```
<?php
class foo {
    // Начиная с версии PHP 5.3.0
    const BAR = <<<'EOT'
bar
EOT;
}
?>
```

В отличие от heredoc, nowdoc может быть использован в любом контексте статичных данных.

Замечание:
Поддержка nowdoc была добавлена в версии PHP 5.3.0.

User Contributed Notes 14 notes

Автоматическая загрузка классов ¶

Большинство разработчиков объектно-ориентированных приложений используют такое соглашение именования файлов, в котором каждый класс хранится в отдельно созданном для него файле. Одной из наиболее при этом досаждающих деталей является необходимость писать в начале каждого скрипта длинный список подгружаемых файлов.

В PHP 5 это делать не обязательно. Можно определить функцию __autoload(), которая будет автоматически вызвана при использовании ранее неопределенного класса или интерфейса. Вызов этой функции - последний шанс для интерпретатора загрузить класс прежде, чем он закончит выполнение скрипта с ошибкой.

Подсказка
spl_autoload_register() предоставляет более гибкую альтернативу для автоматической загрузки классов. По этой причине использовать __autoload() не рекомендуется, а сама функция в будущем может перестать поддерживаться или быть удалена.

Замечание:
До версии 5.3.0, исключения, вызванные в функции __autoload, не могли быть перехвачены в блоке catch и завершались с неисправимой ошибкой. Начиная с версии 5.3.0 эти исключения можно перехватывать в ближайшем блоке catch. Если бросить определенное пользователем исключение, то класс этого исключения должен быть доступен. Функция __autoload также может использоваться рекурсивно для автоматической загрузки пользовательских классов исключений.

Замечание:
Автоматическая загрузка недоступна в случае использования PHP в командной строке в интерактивном режиме.

Замечание:
Если имя класса используется, например, для вызова через call_user_func(), то оно может содержать некоторые опасные символы, такие как ../. Поэтому, рекомендуется не использовать данные от пользователей в таких функциях или же, как минимум, проверять значения в __autoload().

Пример #1 Пример автоматической загрузки

В этом примере функция пытается загрузить классы *MyClass1* и *MyClass2* из файлов *MyClass1.php* и *MyClass2.php* соответственно.

```
<?php
function __autoload($class_name) {
    include $class_name . '.php';
}

$obj  = new MyClass1();
```

```php
$obj2 = new MyClass2();
?>
```

Пример #2 Еще один пример автоматической загрузки

В этом примере представлена попытка загрузки интерфейса *ITest*.

```php
<?php

function __autoload($name) {
    var_dump($name);
}

class Foo implements ITest {
}

/*
string(5) "ITest"

Fatal error: Interface 'ITest' not found in ...
*/
?>
```

Пример #3 Автоматическая загрузка с перехватом исключения в версиях 5.3.0+

В данном примере вызывается исключение и отлавливается блоком try/catch.

```php
<?php
function __autoload($name) {
    echo "Want to load $name.\n";
    throw new Exception("Unable to load $name.");
}

try {
    $obj = new NonLoadableClass();
} catch (Exception $e) {
    echo $e->getMessage(), "\n";
}
?>
```

Результат выполнения данного примера:

```
Want to load NonLoadableClass.
Unable to load NonLoadableClass.
```

Пример #4 Автоматическая загрузка с перехватом исключения в версиях 5.3.0+ - Класс пользовательского исключения не подгружен

В данном примере вызывается недоступное исключение.

```php
<?php
function __autoload($name) {
    echo "Want to load $name.\n";
```

```
    throw new MissingException("Unable to load $name.");
}

try {
    $obj = new NonLoadableClass();
} catch (Exception $e) {
    echo $e->getMessage(), "\n";
}
?>
```

Результат выполнения данного примера:

```
Want to load NonLoadableClass.
Want to load MissingException.

Fatal error: Class 'MissingException' not found in test-
MissingException.php on line 4
```

Смотрите также

- unserialize()
- unserialize_callback_func
- spl_autoload()
- spl_autoload_register()

User Contributed Notes 52 notes

Конструкторы и деструкторы ¶

Конструктор ¶

```
void __construct ([ mixed $args = "" [, $... ]] )
```

PHP 5 позволяет объявлять методы-конструкторы. Классы, в которых объявлен метод-конструктор, будут вызывать этот метод при каждом создании нового объекта, так что это может оказаться полезным, например, для инициализации какого-либо состояния объекта перед его использованием.

Замечание: Конструкторы в классах-родителях не вызываются автоматически, если класс-потомок определяет собственный конструктор. Чтобы вызвать конструктор, объявленный в родительском классе, следует обратиться к методу parent::__construct()внутри конструктора класса-потомка. Если в классе-потомке не определен конструктор, то он может наследоваться от родительского класса как обычный метод (если он не определен как приватный).

Пример #1 Использование унифицированных конструкторов

```
<?php
class BaseClass {
   function __construct() {
       print "Конструктор класса BaseClass\n";
   }
}
```

```php
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
       print "Конструктор класса SubClass\n";
   }
}

class OtherSubClass extends BaseClass {
    // inherits BaseClass's constructor
}

// In BaseClass constructor
$obj = new BaseClass();

// In BaseClass constructor
// In SubClass constructor
$obj = new SubClass();

// In BaseClass constructor
$obj = new OtherSubClass();
?>
```

В целях обратной совместимости, если PHP 5 не может обнаружить объявленный метод __construct() и этот метод не наследуется от родительских классов, то вызов конструктора произойдет по устаревшей схеме, через обращение к методу, имя которого соответствует имени класса. Может возникнуть только одна проблема совместимости старого кода, если в нём присутствуют классы с методами __construct(), использующиеся для других целей.

В отличие от других методов, PHP не будет генерировать ошибку уровня **E_STRICT**, если __construct() будет перекрыт методом с другими параметрами, отличными от тех, которые находятся в родительском __construct().

Начиная с версии PHP 5.3.3, методы с именами, совпадающими с последним элементом имени класса, находящимся в пространстве имен, больше не будут считаться конструкторами. Это изменение не влияет на классы, не находящиеся в пространстве имен.

Пример #2 Конструкторы в классах, находящихся в пространстве имен

```php
<?php
namespace Foo;
class Bar {
    public function Bar() {
        // конструктор в версиях PHP 5.3.0-5.3.2
        // обычный метод, начиная с версии PHP 5.3.3
    }
}
?>
```

Деструкторы ¶

```
void __destruct ( void )
```

PHP 5 предоставляет концепцию деструкторов, сходную с теми, что применяются в других ОО языках, таких, как C++. Деструктор будет вызван при освобождении всех ссылок на определенный объект или при завершении скрипта (порядок выполнения деструкторов не гарантируется).

Пример #3 Пример использования деструктора

```
<?php
class MyDestructableClass {
   function __construct() {
       print "Конструктор\n";
       $this->name = "MyDestructableClass";
   }

   function __destruct() {
       print "Уничтожается " . $this->name . "\n";
   }
}

$obj = new MyDestructableClass();
?>
```

Как и в случае с конструкторами, деструкторы, объявленные в родительском классе, не будут вызваны автоматически. Для вызова деструктора, объявленном в классе-родителе, следует обратиться к методу parent::__destruct() в теле деструктора-потомка. Также класс-потомок может унаследовать деструктор из родительского класса, если он не определен в нем.

Деструктор будет вызван даже в том случае, если скрипт был остановлен с помощью функции exit(). Вызов exit() в деструкторе предотвратит запуск всех последующих функций завершения.

Замечание:
Деструкторы, вызываемые при завершении скрипта, вызываются после отправки HTTP-заголовков. Рабочая директория во время фазы завершения скрипта может отличаться в некоторых SAPI (например, в Apache).
Замечание:
Попытка бросить исключение в деструкторе (вызванного во время завершения скрипта) влечет за собой фатальную ошибку.

User Contributed Notes 50 notes

Область видимости ¶

Область видимости свойства или метода может быть определена путем использования следующих ключевых слов в объявлении:*public*, *protected* или *private*. Доступ к свойствам и методам класса, объявленным как public (общедоступный), разрешен отовсюду. Модификатор protected (защищенный) разрешает доступ наследуемым и родительским классам. Модификатор private (закрытый) ограничивает область видимости так, что только класс, где объявлен сам элемент, имеет к нему доступ.

Область видимости свойства ¶

Свойства класса должны быть определены через модификаторы public, private, или protected. Если же свойство определено с помощью *var*, то оно будет доступно как public свойство.

Пример #1 Объявление свойства класса

```php
<?php
/**
 * Определение MyClass
 */
class MyClass
{
    public $public = 'Общий';
    protected $protected = 'Защищенный';
    private $private = 'Закрытый';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Работает
echo $obj->protected; // Неисправимая ошибка
echo $obj->private; // Неисправимая ошибка
$obj->printHello(); // Выводит Общий, Защищенный и Закрытый


/**
 * Определение MyClass2
 */
class MyClass2 extends MyClass
{
    // Мы можем переопределить public и protected методы, но не
private
    protected $protected = 'Защищенный2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Работает
echo $obj2->protected; // Неисправимая ошибка
```

```
echo $obj2->private; // Неопределен
$obj2->printHello(); // Выводит Общий, Защищенный2 и Неопределен

?>
```

Замечание: Метод объявления переменной через ключевое слово *var*, принятый в PHP 4, до сих пор поддерживается в целях совместимости (как синоним ключевого слова public). В версиях PHP 5 ниже 5.1.3 такое использование выводит предупреждение**E_STRICT**.

Область видимости метода ¶

Методы класса должны быть определены через модификаторы public, private, или protected. Методы, где определение модификатора отсутствует, определяются как public.

Пример #2 Объявление метода

```
<?php
/**
 * Определение MyClass
 */
class MyClass
{
    // Объявление общедоступного конструктора
    public function __construct() { }

    // Объявление общедоступного метода
    public function MyPublic() { }

    // Объявление защищенного метода
    protected function MyProtected() { }

    // Объявление закрытого метода
    private function MyPrivate() { }

    // Это общедоступный метод
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Работает
$myclass->MyProtected(); // Неисправимая ошибка
$myclass->MyPrivate(); // Неисправимая ошибка
$myclass->Foo(); // Работает общий, защищенный и закрытый


/**
 * Определение MyClass2
```

```
 */
class MyClass2 extends MyClass
{
    // Это общедоступный метод
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Неисправимая ошибка
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Работает
$myclass2-
>Foo2(); // Работает общий и защищенный, закрытый не работает

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new foo();
$myFoo->test(); // Bar::testPrivate
                // Foo::testPublic
?>
```

Видимость из других объектов ¶

Объекты одного типа имеют доступ к элементам с модификаторами private и protected друг друга, даже если не являются одним и тем же экземпляром. Это

объясняется тем, что реализация видимости элементов известна внутри этих объектов.

Пример #3 Доступ к элементам с модификатором private из объектов одного типа

```php
<?php
class Test
{
    private $foo;

    public function __construct($foo)
    {
        $this->foo = $foo;
    }

    private function bar()
    {
        echo 'Доступ к закрытому методу.';
    }

    public function baz(Test $other)
    {
        // Мы можем изменить закрытое свойство:
        $other->foo = 'hello';
        var_dump($other->foo);

        // Мы также можем вызвать закрытый метод:
        $other->bar();
    }
}

$test = new Test('test');

$test->baz(new Test('other'));
?>
```

Результат выполнения данного примера:

```
string(5) "hello"
Доступ к закрытому методу.
```

User Contributed Notes 24 notes

Наследование ¶

Наследование — это хорошо зарекомендовавший себя принцип программирования. PHP использует этот принцип в своей объектной модели. Этот принцип будет распространяться на то, каким образом множество классов и объектов относятся друг к другу.

Например, когда вы расширяете класс, дочерний класс наследует все публичные и защищенные методы из родительского класса. До тех пор пока не будут эти методы переопределены, они будут сохранять свою исходную функциональность.

Это полезно для определения и абстрагирования функциональности и позволяет реализовать дополнительную функциональность в похожих объектах без необходимости реализовывать всю общую функциональность.

Замечание:
Пока не используется автозагрузка, классы должны быть объявлены до того, как их будут использовать. Если класс расширяет другой, то родительский класс должен быть объявлен до наследующего класса. Это правило применяется к классам, которые наследуют другие классы или интерфейсы.

Пример #1 Пример наследования

```php
<?php

class Foo
{
    public function printItem($string)
    {
        echo 'Foo: ' . $string . PHP_EOL;
    }

    public function printPHP()
    {
        echo 'PHP is great.' . PHP_EOL;
    }
}

class Bar extends Foo
{
    public function printItem($string)
    {
        echo 'Bar: ' . $string . PHP_EOL;
    }
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Выведет: 'Foo: baz'
$foo->printPHP();       // Выведет: 'PHP is great'
$bar->printItem('baz'); // Выведет: 'Bar: baz'
$bar->printPHP();       // Выведет: 'PHP is great'

?>
```

User Contributed Notes 8 notes

Оператор разрешения области видимости (::) ¶

Оператор разрешения области видимости (также называемый "Paamayim Nekudotayim") или просто "двойное двоеточие" - это лексема, позволяющая обращаться к статическим свойствам, константам и перегруженным свойствам или методам класса.

При обращении к этим элементам извне класса, необходимо использовать имя этого класса.

Начиная с версии PHP 5.3.0, стало возможным обратиться к классу с помощью переменной. Значение переменной не должно быть ключевым словом (например, *self*, *parent* или *static*).

"Paamayim Nekudotayim" на первый взгляд может показаться странным словосочетанием для обозначения двойного двоеточия. Однако во время создания Zend Engine версии 0.5 (который входил в PHP3), команда Zend выбрала именно это обозначение. Оно на самом деле значит "двойное двоеточие" - на иврите!

Пример #1 Использование :: вне объявления класса

```php
<?php
class MyClass {
    const CONST_VALUE = 'Значение константы';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // Начиная с версии PHP 5.3.0

echo MyClass::CONST_VALUE;
?>
```

Для обращения к свойствам и методам внутри самого класса используются ключевые слова `self`, `parent` и `static`.

Пример #2 Использование :: внутри объявления класса

```php
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'статическая переменная';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

$classname = 'OtherClass';
echo $classname::doubleColon(); // Начиная с версии PHP 5.3.0

OtherClass::doubleColon();
?>
```

Когда дочерний класс перегружает методы, объявленные в классе-родителе, PHP не будет осуществлять автоматический вызов методов, принадлежащих классу-родителю. Этот функционал возлагается на метод, перегружаемый в дочернем классе. Данное правило распространяется на конструкторы и деструкторы, перегруженные и "магические" методы.

Пример #3 Обращение к методу в родительском классе

```php
<?php
class MyClass
{
    protected function myFunc() {
        echo "MyClass::myFunc()\n";
    }
}

class OtherClass extends MyClass
{
    // Перекрываем родительское определение
    public function myFunc()
    {
        // Но все еще вызываем родительскую функцию
        parent::myFunc();
        echo "OtherClass::myFunc()\n";
    }
}

$class = new OtherClass();
$class->myFunc();
?>
```

См. также некоторые примеры статических вызовов.

User Contributed Notes 15 notes

Ключевое слово "static" ¶

Подсказка
Эта страница описывает использование ключевого слова *static* для определения статических методов и свойств. *static*также может использоваться для определения статических переменных и позднего статического связывания. Для получения информации о таком применении ключевого слова *static* пользуйтесь вышеуказанными страницами.

Объявление свойств и методов класса статическими позволяет обращаться к ним без создания экземпляра класса. Атрибут класса, объявленный статическим, не может быть доступен посредством экземпляра класса (но статический метод может быть вызван).

В целях совместимости с PHP 4, сделано так, что если не использовалось определение области видимости, то член или метод будет рассматриваться, как если бы он был объявлен как *public*.

Так как статические методы вызываются без создания экземпляра класса, то псевдо-переменная `$this` не доступна внутри метода, объявленного статическим.

Доступ к статическим свойствам класса не может быть получен через оператор ->.

При попытке вызова нестатических методов статически выводится предупреждение уровня **E_STRICT**.

Как и любая другая статическая переменная PHP, статические свойства могут инициализироваться только используя литерал или константу, выражения же недопустимы. Таким образом вы можете инициализировать статическое свойство например целым числом или массивом, но не сможете указать другую переменную, результат вызова функции или объект.

Начиная с версии PHP 5.3.0 существует возможность ссылаться на класс используя переменную. Поэтому значение переменной в таком случае не может быть ключевым словом (например, *self*, *parent* и *static*).

Пример #1 Пример статического свойства

```
<?php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}

class Bar extends Foo
{
    public function fooStatic() {
        return parent::$my_static;
    }
}


print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo-
>my_static . "\n";      // Не определено свойство my_static

print $foo::$my_static . "\n"; // Начиная с PHP 5.3.0
$classname = 'Foo';
print $classname::$my_static . "\n"; // Начиная с PHP 5.3.0

print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Пример #2 Пример статического метода

```
<?php
class Foo {
```

```
    public static function aStaticMethod() {
        // ...
    }
}

Foo::aStaticMethod();
$classname = 'Foo';
$classname::aStaticMethod(); // Начиная с PHP 5.3.0
?>
```

User Contributed Notes 45 notes

Абстрактные классы ¶

PHP 5 поддерживает определение абстрактных классов и методов. Класс, который содержит по крайней мере один абстрактный метод, должен быть определен как абстрактный. Следует помнить, что нельзя создать экземпляр абстрактного класса. Методы, объявленные абстрактными, несут, по существу, лишь описательный смысл и не могут включать реализации.

При наследовании от абстрактного класса, все методы, помеченные абстрактными в родительском классе, должны быть определены в классе-потомке; кроме того, область видимости этих методов должна совпадать (или быть менее строгой). Например, если абстрактный метод объявлен как protected, то реализация этого метода должна быть либо protected либо public, но никак не private. Более того, сигнатуры методов должны совпадать, т.е. контроль типов (type hint) и количество обязательных аргументов должно быть одинаковым. К примеру, если в дочернем классе указан необязательный параметр, которого нет в сигнатуре абстрактного класса, то в данном случае конфликта сигнатур не будет. Это правило также применяется к конструкторам начиная с версии PHP 5.4, ранее сигнатуры конструкторов могли отличаться.

Пример #1 Пример абстрактного класса

```
<?php
abstract class AbstractClass
{
   /* Данный метод должен быть определён в дочернем классе */
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

   /* Общий метод */
    public function printOut() {
        print $this->getValue() . "\n";
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }
```

```
    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') ."\n";
?>
```

Результат выполнения данного примера:

```
ConcreteClass1
FOO_ConcreteClass1
ConcreteClass2
FOO_ConcreteClass2
```

Пример #2 Пример абстрактного класса

```
<?php
abstract class AbstractClass
{
    // Наш абстрактный метод должен определять только необходимы
е аргументы
    abstract protected function prefixName($name);

}

class ConcreteClass extends AbstractClass
{

    // Наш дочерний класс может также определять необязательные
аргументы, не указанные в сигнатуре родительского метода
    public function prefixName($name, $separator = ".") {
        if ($name == "Pacman") {
            $prefix = "Mr";
        } elseif ($name == "Pacwoman") {
            $prefix = "Mrs";
```

```
        } else {
            $prefix = "";
        }
        return "{$prefix}{$separator} {$name}";
    }
}

$class = new ConcreteClass;
echo $class->prefixName("Pacman"), "\n";
echo $class->prefixName("Pacwoman"), "\n";
?>
```

Результат выполнения данного примера:

```
Mr. Pacman
Mrs. Pacwoman
```

Код, предназначенный для прежних версий PHP, должен работать без изменений, если в нём отсутствуют классы или функции, именованные 'abstract'.

User Contributed Notes 21 notes

Интерфейсы объектов ¶

Интерфейсы объектов позволяют создавать код, который указывает, какие методы должен реализовать класс, без необходимости описывания их функционала.

Интерфейсы объявляются так же, как и обычные классы, но с использованием ключевого слова *interface*. Тела методов интерфейсов должны быть пустыми.

Все методы, определенные в интерфейсы должны быть публичными, что следует из самой природы интерфейса.

implements ¶

Для реализации интерфейса используется оператор *implements*. Класс должен реализовать все методы, описанные в интерфейсе; иначе произойдет фатальная ошибка. При желании классы могут реализовывать более одного интерфейса за раз, реализуемые интерфейсы должны разделяться запятой.

Замечание:
Класс не может реализовать два интерфейса, содержащих одноименную функцию, так как это повлечет за собой неоднозначность.
Замечание:
Интерфейсы могут быть унаследованы друг от друга, так же, как и классы, с помощью оператора extends.
Замечание:
Сигнатуры методов в классе, реализующем интерфейс, должны точно совпадать с сигнатурами, используемыми в интерфейсе, в противном случае будет вызвана фатальная ошибка.

Константы (*Constants*) ¶

Интерфейсы могут содержать константы. Константы интерфейсов работают точно так же, как и константы классов, за исключением того, что они не могут быть перекрыты наследующим классом или интерфейсом.

Примеры ¶

Пример #1 Пример интерфейса

```
<?php

// Объявим интерфейс 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Реализуем интерфейс
// Это сработает нормально
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $tem-
plate = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}

// Это не будет работать
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (iTemplate::getHtml)
// (Фатальная ошибка: Класс BadTemplate содержит 1 абстрактный м
етод
// и поэтому должнен быть объявлен абстрактным (iTemplate::getHt
ml))
class BadTemplate implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
```

```
    {
        $this->vars[$name] = $var;
    }
}
?>
```

Пример #2 Расширяемые интерфейсы

```
<?php
interface a
{
    public function foo();
}

interface b extends a
{
    public function baz(Baz $baz);
}

// Это сработает
class c implements b
{
    public function foo()
    {
    }

    public function baz(Baz $baz)
    {
    }
}

// Это не сработает и выдаст фатальную ошибку
class d implements b
{
    public function foo()
    {
    }

    public function baz(Foo $foo)
    {
    }
}
?>
```

Пример #3 Множественное наследование интерфейсов

```
<?php
interface a
{
    public function foo();
}

interface b
```

```
{
    public function bar();
}

interface c extends a, b
{
    public function baz();
}

class d implements c
{
    public function foo()
    {
    }

    public function bar()
    {
    }

    public function baz()
    {
    }
}
?>
```

Пример #4 Интерфейсы с константами

```
<?php
interface a
{
    const b = 'Константа интерфейса';
}

// Выведет: Константа интерфейса
echo a::b;


// Вот это, однако, не будет работать, так как
// константы перекрывать нельзя.
class b implements a
{
    const b = 'Class constant';
}
?>
```

Интерфейс, совместно с контролем типов, предоставляет отличный способ проверки того, что определенный объект содержит определенный набор методов. Смотрите также оператор instanceof и контроль типов.

User Contributed Notes 49 notes

Трейты ¶

Начиная с версии 5.4.0 PHP вводит инструментарий для повторного использования кода, называемый трейтом.

Трейты (англ. traits) - это механизм обеспечения повторного использования кода в языках с поддержкой единого наследования, таких как PHP. Трейты предназначены для уменьшения некоторых ограничений единого наследования, позволяя разработчику повторно использовать наборы методов свободно, в нескольких независимых классах и реализованных с использованием разных архитектур построения классов. Семантика комбинации трейтов и классов определена таким образом, чтобы снизить уровень сложности, а также избежать типичных проблем, связанных с множественным наследованием и с т.н. mixins.

Трейт очень похож на класс, но предназначен для группирования функционала хорошо структурированным и последовательным образом. Невозможно создать самостоятельный экземпляр трейта. Это дополнение к обычному наследованию и позволяет сделать горизонтальную композицию поведения, то есть применение членов класса без необходимости наследования.

Пример #1 Пример использования трейта

```
<?php
trait ezcReflectionReturnInfo {
    function getReturnType() { /*1*/ }
    function getReturnDescription() { /*2*/ }
}

class ezcReflectionMethod extends ReflectionMethod {
    use ezcReflectionReturnInfo;
    /* ... */
}

class ezcReflectionFunction extends ReflectionFunction {
    use ezcReflectionReturnInfo;
    /* ... */
}
?>
```

Приоритет ¶

Наследуемый член из базового класса переопределяется членом, находящимся в трейте. Порядок приоритета следующий: члены из текущего класса переопределяют методы в трейте, которые в свою очередь переопределяют унаследованные методы.

Пример #2 Пример приоритета старшинства

Наследуемый метод от базового класса переопределяется методом, вставленным в MyHelloWorld из трейта Trait. Поведение такое же как и для методов, определенных в классе MyHelloWorld. Порядок приоритета такой: методы из текущего

класса переопределяют методы трейта, которые в свою очередь переопределяют методы из базового класса.

```php
<?php
class Base {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait SayWorld {
    public function sayHello() {
        parent::sayHello();
        echo 'World!';
    }
}

class MyHelloWorld extends Base {
    use SayWorld;
}

$o = new MyHelloWorld();
$o->sayHello();
?>
```

Результат выполнения данного примера:

```
Hello World!
```

Пример #3 Пример альтернативного порядка приоритета

```php
<?php
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

class TheWorldIsNotEnough {
    use HelloWorld;
    public function sayHello() {
        echo 'Hello Universe!';
    }
}

$o = new TheWorldIsNotEnough();
$o->sayHello();
?>
```

Результат выполнения данного примера:

```
Hello Universe!
```

Несколько трейтов ¶

Несколько трейтов могут быть вставлены в класс путем их перечисления в директиве use, разделяя запятыми.

Пример #4 Пример использования нескольких трейтов

```
<?php
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait World {
    public function sayWorld() {
        echo 'World';
    }
}

class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
?>
```

Результат выполнения данного примера:

```
Hello World!
```

Разрешение конфликтов ¶

Если два трейта вставляют метод с одним и тем же именем, это приводит к фатальной ошибке в случае, если конфликт явно не разрешен.

Для разрешения конфликтов именования между трейтами, используемыми в одном и том же классе, необходимо использовать оператор *insteadof* для того, чтобы точно выбрать один из конфликтных методов.

Так как предыдущий оператор позволяет только исключать методы, оператор *as* может быть использован для включения одного из конфликтующих методов под другим именем.

Пример #5 Пример разрешения конфликтов

В этом примере Talker использует трейты A и B. Так как в A и B есть конфликтные методы, он определяет использовать вариант smallTalk из трейта B, и вариант bigTalk из трейта A.

Класс Aliased_Talker применяет оператор *as* чтобы получить возможность использовать имплементацию bigTalk из B под дополнительным псевдонимом *talk*.

```
<?php
trait A {
    public function smallTalk() {
        echo 'a';
    }
    public function bigTalk() {
        echo 'A';
    }
}

trait B {
    public function smallTalk() {
        echo 'b';
    }
    public function bigTalk() {
        echo 'B';
    }
}

class Talker {
    use A, B {
        B::smallTalk insteadof A;
        A::bigTalk insteadof B;
    }
}

class Aliased_Talker {
    use A, B {
        B::smallTalk insteadof A;
        A::bigTalk insteadof B;
        B::bigTalk as talk;
    }
}
?>
```

Изменение видимости метода ¶

Используя синтаксис оператора *as* можно также настроить видимость метода в выставке класса.

Пример #6 Пример изменения видимости метода

```
<?php
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
```

```
    }
}

// Изменение видимости класса sayHello
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// Создание псевдонима метода с измененной видимостью
// видимость sayHello не изменилась
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}
?>
```

Трейты, скомпонованные из трейтов ¶

Аналогично тому, как классы могут использовать трейты, также могут и трейты использовать другие трейты. Используя один или более трейтов в определении другого трейта, он может частично или полностью состоять из членов, описанных в этих трейтах.

Пример #7 Пример трейтов, скомпонованных из трейтов

```
<?php
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait World {
    public function sayWorld() {
        echo 'World!';
    }
}

trait HelloWorld {
    use Hello, World;
}

class MyHelloWorld {
    use HelloWorld;
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
?>
```

Результат выполнения данного примера:

```
Hello World!
```

Абстрактные члены трейтов ¶

Трейты поддерживают использование абстрактных методов для того, чтобы устанавливать требования при выставке класса.

Пример #8 Экпресс требования с абстрактными методами

```
<?php
trait Hello {
    public function sayHelloWorld() {
        echo 'Hello'.$this->getWorld();
    }
    abstract public function getWorld();
}

class MyHelloWorld {
    private $world;
    use Hello;
    public function getWorld() {
        return $this->world;
    }
    public function setWorld($val) {
        $this->world = $val;
    }
}
?>
```

Статические члены трейта ¶

Трейты могут определять и статические свойства и статические методы.

Пример #9 Статические переменные

```
<?php
trait Counter {
    public function inc() {
        static $c = 0;
        $c = $c + 1;
        echo "$c\n";
    }
}

class C1 {
    use Counter;
}

class C2 {
    use Counter;
}

$o = new C1(); $o->inc(); // echo 1
$p = new C2(); $p->inc(); // echo 1
?>
```

Пример #10 Статические методы

```
<?php
trait StaticExample {
    public static function doSomething() {
        return 'Что-либо делаем';
    }
}

class Example {
    use StaticExample;
}

Example::doSomething();
?>
```

Свойства ¶

Трейты могут также определять свойства.

Пример #11 Определение свойств

```
<?php
trait PropertiesTrait {
    public $x = 1;
}

class PropertiesExample {
    use PropertiesTrait;
}

$example = new PropertiesExample;
$example->x;
?>
```

Если трейт определяет свойство, то класс не может определить свойство с таким же именем, иначе будет сгенерирована ошибка. Это будет ошибка **E_STRICT**, если определение класса совместимо (такая же область видимости и начальные значения) или фатальная ошибка в ином случае.

Пример #12 Разрешение конфликтов

```
<?php
trait PropertiesTrait {
    public $same = true;
    public $different = false;
}

class PropertiesExample {
    use PropertiesTrait;
    public $same = true; // Строгое следование стандартам
    public $different = true; // Фатальная ошибка
```

```
}
?>
```

User Contributed Notes 34 notes

Anonymous classes ¶

Support for anonymous classes was added in PHP 7. Anonymous classes are useful when simple, one-off objects need to be created.

```
<?php

// Pre PHP 7 code
class Logger
{
    public function log($msg)
    {
        echo $msg;
    }
}

$util->setLogger(new Logger());

// PHP 7+ code
$util->setLogger(new class {
    public function log($msg)
    {
        echo $msg;
    }
});
```

They can pass arguments through to their constructors, extend other classes, implement interfaces, and use traits just like a normal class can:

```
<?php

class SomeClass {}
interface SomeInterface {}
trait SomeTrait {}

var_dump(new class(10) extends SomeClass implements SomeInterfac
e {
    private $num;

    public function __construct($num)
    {
        $this->num = $num;
    }

    use SomeTrait;
});
```

Результат выполнения данного примера:

```
object(class@anonymous)#1 (1) {
  ["Command line code0x104c5b612":"class@anonymous":private]=>
  int(10)
}
```

Nesting an anonymous class within another class does not give it access to any private or protected methods or properties of that outer class. In order to use the outer class' protected properties or methods, the anonymous class can extend the outer class. To use the private properties of the outer class in the anonymous class, they must be passed through its constructor:

```
<?php

class Outer
{
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1()
    {
        return 3;
    }

    public function func2()
    {
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop)
            {
                $this->prop3 = $prop;
            }

            public function func3()
            {
                return $this->prop2 + $this->prop3 + $this-
>func1();
            }
        };
    }
}

echo (new Outer)->func2()->func3();
```

Результат выполнения данного примера:

```
6
```

All objects created by the same anonymous class declaration are instances of that very class.

```
<?php
function anonymous_class()
```

```
{
    return new class {};
}

if (get_class(anonymous_class()) === get_class(anonymous_class()
)) {
    echo 'same class';
} else {
    echo 'different class';
}
```

Результат выполнения данного примера:

```
same class
```

Замечание:
Note that anonymous classes are assigned a name by the engine, as demonstrated in the following example. This name has to be regarded an implementation detail, which should not be relied upon.

```
<?php
echo get_class(new class {});
```

Результатом выполнения данного примера будет что-то подобное:

```
class@anonymous/in/oNi1A0x7f8636ad2021
```

add a note

User Contributed Notes 1 note

up
down

3
Anonymous ¶

5 months ago

Below three examples describe anonymous class with very simple and basic but quite understandable example

```
<?php
// First way - anonymous class assigned directly to variable
$ano_class_obj = new class{
    public $prop1 = 'hello';
    public $prop2 = 754;
    const SETT = 'some config';

    public function getValue()
    {
        // do some operation
        return 'some returned value';
    }

    public function getValueWithArgu($str)
    {
        // do some operation
        return 'returned value is '.$str;
    }
};
```

```php
echo "\n";

var_dump($ano_class_obj);
echo "\n";

echo $ano_class_obj->prop1;
echo "\n";

echo $ano_class_obj->prop2;
echo "\n";

echo $ano_class_obj::SETT;
echo "\n";

echo $ano_class_obj->getValue();
echo "\n";

echo $ano_class_obj->getValueWithArgu('OOP');
echo "\n";

echo "\n";

// Second way - anonymous class assigned to variable via defined function
$ano_class_obj_with_func = ano_func();

function ano_func()
{
    return new class {
        public $prop1 = 'hello';
        public $prop2 = 754;
        const SETT = 'some config';

        public function getValue()
        {
            // do some operation
            return 'some returned value';
        }

        public function getValueWithArgu($str)
        {
            // do some operation
            return 'returned value is '.$str;
        }
    };
}

echo "\n";

var_dump($ano_class_obj_with_func);
echo "\n";
```

```php
echo $ano_class_obj_with_func->prop1;
echo "\n";

echo $ano_class_obj_with_func->prop2;
echo "\n";

echo $ano_class_obj_with_func::SETT;
echo "\n";

echo $ano_class_obj_with_func->getValue();
echo "\n";

echo $ano_class_obj_with_func->getValueWithArgu('OOP');
echo "\n";

echo "\n";

// Third way - passing argument to anonymous class via construc-
tors
$arg = 1; // we got it by some operation
$config = [2, false]; // we got it by some operation
$ano_class_obj_with_arg = ano_func_with_arg($arg, $config);

function ano_func_with_arg($arg, $config)
{
    return new class($arg, $config) {
        public $prop1 = 'hello';
        public $prop2 = 754;
        public $prop3, $config;
        const SETT = 'some config';

        public function __construct($arg, $config)
        {
            $this->prop3 = $arg;
            $this->config =$config;
        }

        public function getValue()
        {
            // do some operation
            return 'some returned value';
        }

        public function getValueWithArgu($str)
        {
            // do some operation
            return 'returned value is '.$str;
        }
    };
}
```

```
echo "\n";

var_dump($ano_class_obj_with_arg);
echo "\n";

echo $ano_class_obj_with_arg->prop1;
echo "\n";

echo $ano_class_obj_with_arg->prop2;
echo "\n";

echo $ano_class_obj_with_arg::SETT;
echo "\n";

echo $ano_class_obj_with_arg->getValue();
echo "\n";

echo $ano_class_obj_with_arg->getValueWithArgu('OOP');
echo "\n";

echo "\n";
```

Перегрузка ¶

Перегрузка в PHP означает возможность динамически "создавать" свойства и методы. Эти динамические сущности обрабатываются с помощью "волшебных" методов, которые можно создать в классе для различных видов действий.

Методы перегрузки вызываются при взаимодействии с теми свойствами или методами, которые не были объявлены или не видны в текущей области видимости. Далее в этом разделе мы будем использовать термины "недоступные свойства" или "недоступные методы" для отражения этой комбинации объявления и области видимости.

Все методы перегрузки должны быть объявлены как *public*.

Замечание:
Ни один аргумент не может быть передан по ссылке в эти "волшебные" методы.
Замечание:
Интерпретация "перегрузки" в PHP отличается от остальных объектно-ориентированных языков. Традиционно перегрузка означает возможность иметь множество одноименных методов с разным количеством или различными типами аргументов.

Список изменений ¶

Версия	Описание
5.3.0	Добавлен метод __callStatic(). Добавлено предупреждение об усилении публичной видимости и не-статичном объявлении.
5.1.0	Добавлены методы __isset() и __unset(). Добавлена поддержка для перегрузки приватных свойств с помощью __get().
5.0.0	Добавлен метод __get().

Перегрузка свойств ¶

```
public void __set ( string $name , mixed $value )
public mixed __get ( string $name )
public bool __isset ( string $name )
public void __unset ( string $name )
```

Метод __set() будет выполнен при записи данных в недоступные свойства.

Метод __get() будет выполнен при чтении данных из недоступных свойств.

Метод __isset() будет выполнен при использовании isset() или empty() на недоступных свойствах.

Метод __unset() будет выполнен при вызове unset() на недоступном свойстве.

Аргумент `$name` представляет собой имя вызываемого свойства. Метод __set() содержит аргумент `$value`, представляющий собой значение, которое будет записано в свойство с именем `$name`.

Перегрузка свойств работает только в контексте объекта. Данные магические методы не будут вызваны в статическом контексте. Поэтому данные методы не должны объявляться статичными. Начиная с версии PHP 5.3.0, при объявлении "волшебного" метода в качестве *static* будет показано предупреждение.

Замечание:
Возвращаемое значение метода __set() будет проигнорировано из-за способа обработки в PHP оператора присваивания. Аналогично, __get() никогда не вызовется при цепных присваиваниях, например, таких:

```
$a = $obj->b = 8;
```

Пример #1 Перегрузка свойств с помощью методов __get(), __set(), __isset() и __unset()

```
<?php
class PropertyTest
{
    /**  Место хранения перегружаемых данных.  */
    private $data = array();

    /**  Перегрузка не применяется к объявленным свойствам.  */
    public $declared = 1;

    /**  Здесь перегрузка будет использована только при доступе
вне класса.  */
    private $hidden = 2;

    public function __set($name, $value)
    {
        echo "Установка '$name' в '$value'\n";
        $this->data[$name] = $value;
    }
```

```
    public function __get($name)
    {
        echo "Получение '$name'\n";
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        }

        $trace = debug_backtrace();
        trigger_error(
            'Неопределенное свойство в __get(): ' . $name .
            ' в файле ' . $trace[0]['file'] .
            ' на строке ' . $trace[0]['line'],
            E_USER_NOTICE);
        return null;
    }

    /**  Начиная с версии PHP 5.1.0  */
    public function __isset($name)
    {
        echo "Установлено ли '$name'?\n";
        return isset($this->data[$name]);
    }

    /**  Начиная с версии PHP 5.1.0  */
    public function __unset($name)
    {
        echo "Уничтожение '$name'\n";
        unset($this->data[$name]);
    }

    /**  Не "волшебный" метод, просто для примера. */
    public function getHidden()
    {
        return $this->hidden;
    }
}


echo "<pre>\n";

$obj = new PropertyTest;

$obj->a = 1;
echo $obj->a . "\n\n";

var_dump(isset($obj->a));
unset($obj->a);
var_dump(isset($obj->a));
echo "\n";

echo $obj->declared . "\n\n";

echo "Давайте поэкспериментируем с private свойством 'hidden':\n
```

```
";
echo "Private свойства видны внутри класса, поэтому __get() не и
спользуется...\n";
echo $obj->getHidden() . "\n";
echo "Private свойства не видны вне класса, поэтому __get() испо
льзуется...\n";
echo $obj->hidden . "\n";
?>
```

Результат выполнения данного примера:

```
Установка 'a' в '1'
Получение 'a'
1

Установлено ли 'a'?
bool(true)
Уничтожение 'a'
Установлено ли 'a'?
bool(false)

1

Давайте поэкспериментируем с private свойством 'hidden':
Private свойства видны внутри класса, поэтому __get() не исполь-
зуется...
2
Private свойства не видны вне класса, поэтому __get() использу-
ется...
Получение 'hidden'


Notice:    Неопределенное свойство в __get(): hidden в файле
<file> на строке 70 в <file> на строке 29
```

Перегрузка методов ¶

```
public mixed __call ( string $name , array $arguments )
public                                                  stat-
ic mixed __callStatic ( string $name , array $arguments )
```

В контексте объекта при вызове недоступных методов вызывается метод __call().

В статическом контексте при вызове недоступных методов вызывается метод __callStatic().

Аргумент `$name` представляет собой имя вызываемого метода. Аргумент `$arguments` представляет собой числовой массив, содержащий параметры, переданные в вызываемый метод `$name`.

Пример #2 Перегрузка методов с помощью методов __call() и __callStatic()

```
<?php
class MethodTest {
```

```
    public function __call($name, $arguments) {
        // Замечание: значение $name регистрозависимо.
        echo "Вызов метода '$name' "
             . implode(', ', $arguments). "\n";
    }

    /**  Начиная с версии PHP 5.3.0  */
    public static function __callStatic($name, $arguments) {
        // Замечание: значение $name регистрозависимо.
        echo "Вызов статического метода '$name' "
             . implode(', ', $arguments). "\n";
    }
}

$obj = new MethodTest;
$obj->runTest('в контексте объекта');

MethodTest::runTest('в статическом контексте');  // Начиная с ве
рсии PHP 5.3.0
?>
```

Результат выполнения данного примера:

```
Вызов метода 'runTest' в контексте объекта
Вызов статического метода 'runTest' в статическом контексте
```

User Contributed Notes 67 notes

Итераторы объектов ¶

PHP 5 предоставляет такой способ объявления объектов, который дает возможность пройти по списку элементов данного объекта, например, с помощью оператора foreach. По умолчанию, в этом обходе (итерации) будут участвовать все видимые свойства объекта.

Пример #1 Итерация простого объекта

```
<?php
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';

    protected $protected = 'protected var';
    private   $private   = 'private var';

    function iterateVisible() {
       echo "MyClass::iterateVisible:\n";
       foreach($this as $key => $value) {
           print "$key => $value\n";
       }
    }
```

```
}

$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
echo "\n";


$class->iterateVisible();

?>
```

Результат выполнения данного примера:

```
var1 => value 1
var2 => value 2
var3 => value 3

MyClass::iterateVisible:
var1 => value 1
var2 => value 2
var3 => value 3
protected => protected var
private => private var
```

Как показывает результат, foreach проитерировал все доступные и принадлежащие объекту видимые свойства.

Кроме того, вы можете развить эту концепцию и реализовать встроенный в PHP 5 интерфейс Iterator. Это позволит самому объекту решать как он будет итерироваться и какие данные будут доступны на каждой итерации.

Пример #2 Объект Iteration, реализующий интерфейс Iterator

```
<?php
class MyIterator implements Iterator
{
    private $var = array();

    public function __construct($array)
    {
        if (is_array($array)) {
            $this->var = $array;
        }
    }

    public function rewind()
    {
        echo "перемотка в начало\n";
        reset($this->var);
    }
```

```
    public function current()
    {
        $var = current($this->var);
        echo "текущий: $var\n";
        return $var;
    }

    public function key()
    {
        $var = key($this->var);
        echo "ключ: $var\n";
        return $var;
    }

    public function next()
    {
        $var = next($this->var);
        echo "следующий: $var\n";
        return $var;
    }

    public function valid()
    {
        $key = key($this->var);
        $var = ($key !== NULL && $key !== FALSE);
        echo "верный: $var\n";
        return $var;
    }

}

$values = array(1,2,3);
$it = new MyIterator($values);

foreach ($it as $a => $b) {
    print "$a: $b\n";
}
?>
```

Результат выполнения данного примера:

```
перемотка в начало
верный: 1
текущий: 1
ключ: 0
0: 1
следующий: 2
верный: 1
текущий: 2
ключ: 1
1: 2
следующий: 3
```

```
верный: 1
текущий: 3
ключ: 2
2: 3
следующий:
верный:
```

Интерфейс IteratorAggregate может быть использован как альтернатива реализации всех методов интерфейса Iterator.IteratorAggregate требует чтобы был реализован только один метода - IteratorAggregate::getIterator(), который должен возвращать экземпляр класса, реализующий интерфейс Iterator.

Пример #3 Объект Iteration, реализующий интерфейс IteratorAggregate

```php
<?php
class MyCollection implements IteratorAggregate
{
    private $items = array();
    private $count = 0;

    // Требование интерфейса IteratorAggregate
    public function getIterator() {
        return new MyIterator($this->items);
    }

    public function add($value) {
        $this->items[$this->count++] = $value;
    }
}

$coll = new MyCollection();
$coll->add('value 1');
$coll->add('value 2');
$coll->add('value 3');

foreach ($coll as $key => $val) {
    echo "ключ/значение: [$key -> $val]\n\n";
}
?>
```

Результат выполнения данного примера:

```
перемотка в начало
текущий: value 1
верный: 1
текущий: value 1
ключ: 0
ключ/значение: [0 -> value 1]

следующий: value 2
текущий: value 2
верный: 1
текущий: value 2
```

```
ключ: 1
ключ/значение: [1 -> value 2]

следующий: value 3
текущий: value 3
верный: 1
текущий: value 3
ключ: 2
ключ/значение: [2 -> value 3]

следующий:
текущий:
верный:
```

Замечание:
Больше примеров итераторов можно найти в расширении SPL.
Замечание:
Пользователи PHP 5.5 и более поздних версий также могут изучить генераторы, представляющие собой альтернативный способ определения итераторов.

User Contributed Notes 16 notes

Магические методы ¶

Имена мето-
мето-
дов __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(),__invoke(), __set_state(), __clone() и __debugInfo() зарезервированы для "магических" методов в PHP. Не стоит называть свои методы этими именами, если вы не хотите использовать их "магическую" функциональность.

Предостережение
PHP оставляет за собой право все методы, начинающиеся с __, считать "магическими". Не рекомендуется использовать имена методов с __ в PHP, если вы не желаете использовать соответствующий "магический" функционал.

__sleep() и __wakeup() ¶

```
public array __sleep ( void )
void __wakeup ( void )
```

Функция serialize() проверяет, присутствует ли в вашем классе метод с "магическим" именем __sleep(). Если это так, то этот метод выполняется прежде любой операции сериализации. Он может очистить объект и предполагается, что будет возвращен массив с именами всех переменных объекта, который должен быть сериализован. Если метод ничего не возвращает кроме **NULL**, то это значит, что объект сериализован и выдается предупреждение **E_NOTICE**.

Замечание:
Недопустимо возвращать в __sleep() имена приватных свойств объекта в родительский класс. Это приведет к предупреждению**E_NOTICE**. Вместо этого вы можете использовать интерфейс Serializable.

Рекомендованное использование __sleep() состоит в завершении работы над данными, ждущими обработки или других подобных задач очистки. Кроме того, этот метод можно выполнять в тех случаях, когда нет необходимости сохранять полностью очень большие объекты.

С другой стороны, функция unserialize() проверяет наличие метода с "магическим" именем __wakeup(). Если такой имеется, то он может воссоздать все ресурсы объекта, принадлежавшие ему.

Обычно __wakeup() используется для восстановления любых соединений с базой данных, которые могли быть потеряны во время операции сериализации и выполнения других операций повторной инициализации.

Пример #1 Sleep и wakeup

```
<?php
class Connection
{
    protected $link;
    private $dsn, $username, $password;

    public function __construct($dsn, $username, $password)
    {
        $this->dsn = $dsn;
        $this->username = $username;
        $this->password = $password;
        $this->connect();
    }

    private function connect()
    {
        $this->link = new PDO($this->dsn, $this-
>username, $this->password);
    }

    public function __sleep()
    {
        return array('dsn', 'username', 'password');
    }

    public function __wakeup()
    {
        $this->connect();
    }
}?>
```

__toString() ¶

```
public string __toString ( void )
```

Метод __toString() позволяет классу решать самостоятельно, как он должен реагировать при преобразовании в строку. Например, что напечатает *echo $obj;*. Этот метод должен возвращать строку, иначе выдастся неисправимая ошибка **E_RECOVERABLE_ERROR**.

Внимание
Нельзя бросить исключение из метода __toString(). Попытка это сделать закончится фатальной ошибкой.

Пример #2 Простой пример

```php
<?php
// Объявление простого класса
class TestClass
{
    public $foo;

    public function __construct($foo)
    {
        $this->foo = $foo;
    }

    public function __toString()
    {
        return $this->foo;
    }
}

$class = new TestClass('Привет');
echo $class;
?>
```

Результат выполнения данного примера:

```
Привет
```

Ранее, до PHP 5.2.0, метод __toString() вызывался только непосредственно в сочетании с функциями echo или print. Начиная с PHP 5.2.0, он вызывается в любом строчном контексте (например, в printf() с модификатором *%s*), но не в контекстах других типов (например, с *%d* модификатором). Начиная с PHP 5.2.0, преобразование объекта в строку при отсутствии метода __toString() вызывает ошибку **E_RECOVERABLE_ERROR**.

__invoke() ¶

```
mixed __invoke ([ $... ] )
```

Метод __invoke() вызывается, когда скрипт пытается выполнить объект как функцию.

Замечание:
Данный метод доступен начиная с PHP 5.3.0.

Пример #3 Использование __invoke()

```php
<?php
class CallableClass
{
```

```
    public function __invoke($x)
    {
        var_dump($x);
    }
}
$obj = new CallableClass;
$obj(5);
var_dump(is_callable($obj));
?>
```

Результат выполнения данного примера:

```
int(5)
bool(true)
```

__set_state() ¶

```
static object __set_state ( array $properties )
```

Этот статический метод вызывается для тех классов, которые экспортируются функцией var_export() начиная с PHP 5.1.0.

Параметр этого метода должен содержать массив, состоящий из экспортируемых свойств в виде *array('property' => value, ...)*.

Пример #4 Использование __set_state() (начиная с PHP 5.1.0)

```
<?php

class A
{
    public $var1;
    public $var2;

    public static function __set_state($an_array) // С PHP 5.1.0
    {
        $obj = new A;
        $obj->var1 = $an_array['var1'];
        $obj->var2 = $an_array['var2'];
        return $obj;
    }
}

$a = new A;
$a->var1 = 5;
$a->var2 = 'foo';

eval('$b = ' . var_export($a, true) . ';'); // $b = A::__set_sta
te(array(
                                            //    'var1' => 5,
                                            //    'var2' => 'foo
',
                                            // ));
var_dump($b);
```

```
?>
```

Результат выполнения данного примера:

```
object(A)#2 (2) {
  ["var1"]=>
  int(5)
  ["var2"]=>
  string(3) "foo"
}
```

__debugInfo() ¶

```
array __debugInfo ( void )
```

Этот метод вызывается функцией var_dump(), когда необходимо вывести список свойств объекта. Если этот метод не определен, тогда будут выведены все public, protected и private свойства объекта.

Этот метод был добавлен в PHP 5.6.0.

Пример #5 Использование __debugInfo()

```
<?php
class C {
    private $prop;

    public function __construct($val) {
        $this->prop = $val;
    }

    public function __debugInfo() {
        return [
            'propSquared' => $this->prop ** 2,
        ];
    }
}

var_dump(new C(42));
?>
```

Результат выполнения данного примера:

```
object(C)#1 (1) {
  ["propSquared"]=>
  int(1764)
}
```

User Contributed Notes 45 notes

Ключевое слово "final" ¶

PHP 5 представляет ключевое слово *final*, разместив которое перед объявлениями методов класса, можно предотвратить их переопределение в дочерних классах. Если же сам класс определяется с этим ключевым словом, то он не сможет быть унаследован.

Пример #1 Пример окончательных (final) методов

```
<?php
class BaseClass {
   public function test() {
       echo "Вызван метод BaseClass::test()\n";
   }

   final public function moreTesting() {
       echo "Вызван метод BaseClass::moreTesting()\n";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "Вызван метод ChildClass::moreTesting()\n";
   }
}
// Выполнение заканчивается фатальной ошибкой: Cannot override f
inal method BaseClass::moreTesting()
// (Метод BaseClass::moretesting() не может быть переопределён)
?>
```

Пример #2 Пример окончательного (final) класса

```
<?php
final class BaseClass {
   public function test() {
       echo "Вызван метод BaseClass::test()\n";
   }

   // В данном случае неважно, укажете ли вы этот метод как fina
l или нет
   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
}
// Выполнение заканчивается фатальной ошибкой: Class ChildClass
may not inherit from final class (BaseClass)
// (Класс ChildClass не может быть унаследован от окончательного
```

```
 класса (BaseClass))
?>
```

Замечание: Свойства не могут быть объявлены окончательными, только классы и методы.

User Contributed Notes 13 notes

Клонирование объектов ¶

Создание копии объекта с абсолютно идентичными свойствами не всегда является приемлемым вариантом. Хорошим примером необходимости копирования конструкторов может послужить ситуация, когда у вас есть объект, представляющий собой окно GTK и содержащий ресурс-идентификатор этого окна; когда вы создаете копию этого объекта, вам может понадобиться, чтобы копия объекта содержала ресурс-идентификатор нового окна. Другим примером может послужить ситуация, когда ваш объект содержит ссылку на какой-либо другой используемый объект и, когда вы создаёте копию родительского объекта, вам нужно также создать новый экземпляр этого другого объекта, так, чтобы копия объекта-контейнера содержала собственный отдельный экземпляр содержащегося объекта.

Копия объекта создается с использованием ключевого слова *clone* (который вызывает метод __clone() объекта, если это возможно). Вызов метода __clone() не может быть осуществлён непосредственно.

```
$copy_of_object = clone $object;
```

При клонировании объекта, PHP 5 выполняет неполную копию всех свойств объекта. Любые свойства, являющиеся ссылками на другие переменные, останутся ссылками.

```
void __clone ( void )
```

По завершении клонирования, если у класса был определен метод __clone(), то этот метод __clone() вызывается у свежесозданной копии объекта, для возможного изменения всех необходимых свойств.

Пример #1 Клонирование объекта

```
<?php
class SubObject
{
    static $instances = 0;
    public $instance;

    public function __construct() {
        $this->instance = ++self::$instances;
    }

    public function __clone() {
        $this->instance = ++self::$instances;
    }
}
```

```php
class MyCloneable
{
    public $object1;
    public $object2;

    function __clone()
    {
        // Принудительно копируем this->object, иначе
        // он будет указывать на один и тот же объект.
        $this->object1 = clone $this->object1;
    }
}

$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();

$obj2 = clone $obj;


print("Оригинальный объект:\n");
print_r($obj);

print("Клонированный объект:\n");
print_r($obj2);

?>
```

Результат выполнения данного примера:

```
Оригинальный объект:
MyCloneable Object
(
    [object1] => SubObject Object
        (
            [instance] => 1
        )

    [object2] => SubObject Object
        (
            [instance] => 2
        )

)
Клонированный объект:
MyCloneable Object
(
    [object1] => SubObject Object
        (
            [instance] => 3
        )
```

```
    [object2] => SubObject Object
        (
            [instance] => 2
        )

)
```

User Contributed Notes 14 notes

Сравнение объектов ¶

При использовании оператора сравнения (==), свойства объектов просто сравниваются друг с другом, а именно: два объекта равны, если они содержат одинаковые свойства и одинаковые значения, и являются экземплярами одного и того же класса.

При использовании оператора идентичности (===), переменные объектов считаются идентичными тогда и только тогда, когда они ссылаются на один и тот же экземпляр одного и того же класса.

Следующий пример внесёт ясность.

Пример #1 Пример сравнения объектов в PHP 5

```php
<?php
function bool2str($bool)
{
    if ($bool === false) {
        return 'FALSE';
    } else {
        return 'TRUE';
    }
}

function compareObjects(&$o1, &$o2)
{
    echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "\n";
    echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "\n";
    echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "\n";
    echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "\n";
}

class Flag
{
    public $flag;

    function Flag($flag = true) {
        $this->flag = $flag;
    }
}

class OtherFlag
{
```

```
    public $flag;

    function OtherFlag($flag = true) {
        $this->flag = $flag;
    }
}

$o = new Flag();
$p = new Flag();
$q = $o;
$r = new OtherFlag();

echo "Два экземпляра одного и того же класса\n";
compareObjects($o, $p);

echo "\nДве ссылки на один и тот же экземпляр\n";
compareObjects($o, $q);

echo "\nЭкземпляры двух разных классов\n";
compareObjects($o, $r);
?>
```

Результат выполнения данного примера:

```
Два экземпляра одного и того же класса
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : FALSE
o1 !== o2 : TRUE

Две ссылки на один и тот же экземпляр
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

Экземпляры двух разных классов
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Замечание:
Расширения могут определять собственные правила для сравнения своих объектов (==).

User Contributed Notes 11 notes

Контроль типа ¶

PHP 5 предоставляет возможность использовать контроль типов. На данный момент функции имеют возможность заставлять параметры быть либо объектами (путем указания имени класса в прототипе функции), либо интерфейсами, либо массивами (начиная с PHP 5.1), или колбеком с типом callable (начиная с PHP

5.4). Однако если NULL использовался как значение параметра по умолчанию, то это будет также допустимо в качестве аргумента для последующего вызова.

Если класс или интерфейс указан для контроля типа, то все его потомки или реализации также допустимы.

Контроль типа не может быть использован со скалярными типами, такими как int или string. Ресурсы и Трейты также недопустимы.

Пример #1 Пример контроля типов

```php
<?php
// Тестовый класс
class MyClass
{
    /**
     * Тестовая функция
     *
     * Первый параметр должен быть объектом типа OtherClass
     */
    public function test(OtherClass $otherclass) {
        echo $otherclass->var;
    }


    /**
     * Другая тестовая функция
     *
     * Первый параметр должен быть массивом
     */
    public function test_array(array $input_array) {
        print_r($input_array);
    }

    /**
     * Первый параметр должен быть итератором
     */
    public function test_interface(Traversable $iterator) {
        echo get_class($iterator);
    }

    /**
     * Первый параметр должен быть типа callable
     */
    public function test_callable(callable $callback, $data) {
        call_user_func($callback, $data);
    }
}

// Другой тестовый класс
class OtherClass {
    public $var = 'Hello World';
```

```
}
?>
```

В случае передачи аргумента неправильного типа результатом будет фатальная ошибка.

```
<?php
// Экземпляры каждого класса
$myclass = new MyClass;
$otherclass = new OtherClass;

// Ошибка: Аргумент 1 должен быть экземпляром класса OtherClass
$myclass->test('hello');

// Ошибка: Аргумент 1 должен быть экземпляром класса OtherClass
$foo = new stdClass;
$myclass->test($foo);

// Ошибка: Аргумент 1 не должен быть null
$myclass->test(null);

// Работает: Выводит Hello World
$myclass->test($otherclass);

// Ошибка: Аргумент 1 должен быть массив
$myclass->test_array('a string');

// Работает: Выводит массив
$myclass->test_array(array('a', 'b', 'c'));

// Работает: Выводит ArrayObject
$myclass->test_interface(new ArrayObject(array()));

// Работает: Выводит int(1)
$myclass->test_callable('var_dump', 1);
?>
```

Также, контроль типов работает и с функциями:

```
<?php
// Пример класса
class MyClass {
    public $var = 'Hello World';
}

/**
 * Тестовая функция
 *
 * Первый параметр должен быть объект класса MyClass
 */
function myFunction(MyClass $foo) {
    echo $foo->var;
}
```

```php
// Это работает
$myclass = new MyClass;
myFunction($myclass);
?>
```

Контроль типов допускает значения NULL:

```php
<?php

/* Прием значения NULL */
function test(stdClass $obj = NULL) {

}

test(NULL);
test(new stdClass);

?>
```

User Contributed Notes 30 notes

Позднее статическое связывание ¶

Начиная с версии PHP 5.3.0 появилась особенность, называемая позднее статическое связывание, которая может быть использована для того чтобы получить ссылку на вызываемый класс в контексте статического наследования.

Если говорить более точно, позднее статическое связывание сохраняет имя класса указанного в последнем "не перенаправленном вызове". В случае статических вызовов это явно указанный класс (обычно слева от оператора *::*); в случае не статических вызовов это класс объекта. "Перенаправленный вызов" - это статический вызов, начинающийся с *self::*, *parent::*, *static::*, или, если двигаться вверх по иерархии классов, forward_static_call(). Функция get_called_class() может быть использована чтобы получить строку с именем вызванного класса, и *static::* представляет ее область действия.

Само название "позднее статическое связывание" отражает в себе внутреннюю реализацию этой особенности. "Позднее связывание" отражает тот факт, что обращения через *static::* не будут вычисляться по отношению к классу, в котором вызываемый метод определен, а будут вычисляться на основе информации в ходе исполнения. Также эта особенность была названа "статическое связывание" потому, что она может быть использована (но не обязательно) в статических методах.

Ограничения *self::* ¶

Статические ссылки на текущий класс, наподобие *self::* или *__CLASS__*, вычисляются используя класс, к которому они принадлежат, как к тому, в котором они были определены.

Пример #1 Использование *self::*

```php
<?php
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        self::who();
    }
}

class B extends A {
    public static function who() {
        echo __CLASS__;
    }
}

B::test();
?>
```

Результат выполнения данного примера:

```
A
```

Использование позднего статического связывания ¶

Позднее статическое связывание пытается устранить это ограничение предоставляя ключевое слово, которое ссылается на класс, вызванный непосредственно в ходе выполнения. Попросту говоря, ключевое слово, которое позволит вам ссылаться на *B* из *test()* в предыдущем примере. Было решено не вводить новое ключевое слово, а использовать *static*, которое уже зарезервировано.

Пример #2 Простое использование*static::*

```php
<?php
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        static::who(); // Здесь действует позднее статическое св
язывание
    }
}

class B extends A {
    public static function who() {
        echo __CLASS__;
    }
}

B::test();
?>
```

Результат выполнения данного примера:

```
B
```

Замечание:
В нестатическом контексте вызванным классом будет тот, к которому относится экземпляр объекта. Поскольку *$this->* будет пытаться вызывать закрытые методы из той же области действия, использование *static::* может дать разные результаты. Другое отличие в том, что *static::* может ссылаться только на статические поля класса.

Пример #3 Использование *static::* в нестатическом контексте

```php
<?php
class A {
    private function foo() {
        echo "success!\n";
    }
    public function test() {
        $this->foo();
        static::foo();
    }
}

class B extends A {
   /* foo() будет скопирован в B, следовательно его область дейс
твия по прежнему A,
      и вызов будет успешен*/
}

class C extends A {
    private function foo() {
        /* исходный метод заменен; область действия нового метод
а C */
    }
}

$b = new B();
$b->test();
$c = new C();
$c->test();   //не верно
?>
```

Результат выполнения данного примера:

```
success!
success!
success!


Fatal error:  Call to private method C::foo() from context 'A'
in /tmp/test.php on line 9
```

Замечание:

Разрешающая область позднего статического связывания будет фиксирована вычисляющем ее статическим вызовом. С другой стороны, статические вызовы с использованием таких директив как *parent::* или *self::* перенаправляют информацию вызова.

Пример #4 Перенаправленные и не перенаправленные вызовы

```
<?php
class A {
    public static function foo() {
        static::who();
    }

    public static function who() {
        echo __CLASS__."\n";
    }
}

class B extends A {
    public static function test() {
        A::foo();
        parent::foo();
        self::foo();
    }

    public static function who() {
        echo __CLASS__."\n";
    }
}
class C extends B {
    public static function who() {
        echo __CLASS__."\n";
    }
}

C::test();
?>
```

Результат выполнения данного примера:

```
A
C
C
```

User Contributed Notes 26 notes

Объекты и ссылки ¶

Одним из ключевых моментов объектно-ориентированной парадигмы PHP 5, которой часто обсуждается, является "передача объектов по ссылке по умолчанию". Это не совсем верно. Этот раздел уточняет это понятие используя некоторые примеры.

Ссылка в PHP это псевдоним (алиас), который позволяет присвоить двум переменным одинаковое значение. Начиная с PHP 5 объектная переменная больше не содержит сам объект как значение. Такая переменная содержит только идентификатор объекта, который позволяет найти конкретный объект при обращении к

нему. Когда объект передается как аргумент функции, возвращается или присваивается другой переменной, то эти разные переменные не являются псевдонимами (алиасами): они содержат копию идентификатора, который указывает на один и тот же объект.

Пример #1 Ссылки и объекты

```
<?php
class A {
    public $foo = 1;
}

$a = new A;
$b = $a;     // $a и $b копии одного идентификатора
             // ($a) = ($b) = <id>
$b->foo = 2;
echo $a->foo."\n";


$c = new A;
$d = &$c;    // $c и $d ссылки
             // ($c,$d) = <id>

$d->foo = 2;
echo $c->foo."\n";


$e = new A;

function foo($obj) {
    // ($obj) = ($e) = <id>
    $obj->foo = 2;
}

foo($e);
echo $e->foo."\n";

?>
```

Результат выполнения данного примера:

```
2
2
2
```

User Contributed Notes 15 notes

Сериализация объектов ¶

Сериализация объектов - сохранение объектов между сессиями ¶

Функция serialize() возвращает строковое представление любого значения, которое может быть сохранено в PHP. Функцияunserialize() использует эту строку для

восстановления исходного значения переменной. Использование serialize для сериализации объекта сохранит имя класса и все его свойства, однако методы не сохраняются.

Для того, чтобы иметь возможность сделать unserialize() для объекта нужно чтобы класс этого объекта был определен заранее. То есть, если у вас есть экземпляр класса A, и вы сделаете его сериализацию, вы получите его строковое представление, которое содержит значение всех переменных описанных в нем. Для того, чтобы восстановить объект из строки в другом PHP файле класс A должен быть определен заранее. Это можно сделать сохранив определение класса A в отдельный файл и подключить этот файл или использовать функцию spl_autoload_register() для автоматического подключения.

```
<?php
// classa.inc:

  class A {
      public $one = 1;

      public function show_one() {
          echo $this->one;
      }
  }

// page1.php:

  include("classa.inc");

  $a = new A;
  $s = serialize($a);
  // сохраняем $s где-
нибудь, откуда page2.php сможет его получить.
  file_put_contents('store', $s);

// page2.php:

  // это нужно для того, чтобы функция unserialize работала прав
ильно.
  include("classa.inc");

  $s = file_get_contents('store');
  $a = unserialize($s);

  // теперь можно использовать метод show_one() объекта $a.
  $a->show_one();
?>
```

Если приложение использует сессии и функцию session_register() для регистрации объектов, эти объекты сериализуются автоматически в конце исполнения каждой страницы PHP, и десериализуются автоматически в начале исполнения каждой из следующих страниц. Это означает, что эти объекты могут появиться на любой из страниц приложения, став однажды частью сессии. Тем не менее, функция session_register() удалена в PHP 5.4.0.

Если в приложении производится сериализация объектов для последующего использования, настоятельно рекомендуется подключать определение класса для этого объекта во всем приложении. Невыполнение этого требования может привести к тому, что объект будет десериализован в отсутствие определения класса, что в свою очередь приведет к тому, что PHP будет использовать для этого объекта класс __PHP_Incomplete_Class_Name, который не имеет методов и сделает объект бесполезным.

В приведенном выше примере `$a` стало частью сессии после запуска *session_register("a")*, вы должны подключать файл *classa.inc* на всех ваших страницах, а не только `page1.php` и `page2.php`.

Обратите внимание, что, кроме вышеприведенного совета, можно вмешаться в процесс сериализации и десериализации объекта с помощью методов __sleep() и __wakeup(). Метод __sleep() также позволяет сериализовать лишь некоторое подмножество свойств объекта.

User Contributed Notes 4 notes

Журнал изменений ООП ¶

Изменения модели ООП в PHP 5 описаны в этом разделе. Описания и другие примечания, касающиеся этих возможностей, описаны в рамках документации ООП PHP5

Версия	Описание
5.5.0	Добавлено: finally в обработчик исключений.
5.4.0	Добавлено: трейты.
5.4.0	Изменено: Если абстрактный класс определяет сигнатуру для конструктора, то она будет принудительно применяться.
5.3.3	Изменено: Методы с тем же именем, что и последний элемент пространства имен класса больше не будет рассматриваться как конструктор. Это изменение не будет влиять на классы, не входящие в пространства имен.
5.3.0	Изменено: Больше не требуется, чтобы классы, реализующие интерфейсы с методами, которые имеют значения по умолчанию в прототипе, соответствовали значениям по умолчанию в интерфейсе.
5.3.0	Изменено: Теперь стало возможным ссылаться на класс, используя переменную (например, *echo $classname::constant;*). Значение переменной не может быть ключевым словом (например, *self*, *parent* или *static*).
5.3.0	Изменено: Ошибка **E_WARNING** происходит, если магические перегруженные методы задекларированы как статические. Это также усиливает требование, что эти методы должны быть публичными.
5.3.0	Изменено: До 5.3.0, исключения в функции __autoload() не могли быть перехвачены в блоке catch и приводили к фатальной ошибке. Сейчас исключения в функции __autoload могут быть перехвачены в блоке catch, но с одной оговоркой. Если перехватывается пользовательское исключение, то класс, обрабатывающий это исключение, должен быть доступен. Функция __autoload может быть использована рекурсивно для автозагрузки пользовательского класса обработки исключения.
5.3.0	Добавлено: Метод __callStatic.
5.3.0	Добавлено: Поддержка heredoc и nowdoc для *констант* и определений свойств класса. Примечание: Значение heredoc должны следовать тем же правилам, что и

Версия	Описание
	строки в двойных кавычках (например, без переменных внутри).
5.3.0	Добавлено: Позднее статическое связывание.
5.3.0	Добавлено: метод __invoke().
5.2.0	Изменено: Метод __toString() вызывался только когда он напрямую комбинировался с echo или print. Сейчас он вызывается в любом контексте строки (например, в printf() с модификатором *%s*), но не в других типах контекста (например, с модификатором *%d*). С PHP 5.2.0, преобразование объектов без метода *__toString* в строку выдает ошибку уровня **E_RECOVERABLE_ERROR**.
5.1.3	Изменено: В предыдущих версиях PHP 5 использование *var* считалось устаревшим и выдавало ошибку **E_STRICT**. Сейчас это не считается устаревшим, поэтому ошибка больше не выдается.
5.1.0	Изменено: Статический метод __set_state() теперь вызывается для классов, экспортируемых функцией var_export().
5.1.0	Добавлены: методы __isset() и __unset().

User Contributed Notes 2 notes

Пространства имен ¶

Содержание ¶

User Contributed Notes 6 notes

Anonymous ¶

5 years ago

```
The keyword 'use' has two different applications, but the re-
served word table links to here.

It can apply to namespace constucts:

file1:
<?php namespace foo;
  class Cat {
    static function says() {echo 'meoow';}  } ?>

file2:
<?php namespace bar;
```

```
  class Dog {
    static function says() {echo 'ruff';}  } ?>
```

file3:

```
<?php namespace animate;
  class Animal {
    static function breathes() {echo 'air';}  } ?>
```

file4:

```
<?php namespace fub;
  include 'file1.php';
  include 'file2.php';
  include 'file3.php';
  use foo as feline;
  use bar as canine;
  use animate;
  echo \feline\Cat::says(), "<br />\n";
  echo \canine\Dog::says(), "<br />\n";
  echo \animate\Animal::breathes(), "<br />\n";  ?>
```

Note that
felineCat::says()
should be
\feline\Cat::says()
(and similar for the others)
but this comment form deletes the backslash (why???)

The 'use' keyword also applies to closure constructs:

```
<?php function getTotal($products_costs, $tax)
    {
        $total = 0.00;

        $callback =
            function ($pricePerItem) use ($tax, &$total)
            {

                $total += $pricePerItem * ($tax + 1.0);
            };

        array_walk($products_costs, $callback);
        return round($total, 2);
    }
?>
```

Обзор пространств имен ¶

(PHP 5 >= 5.3.0, PHP 7)

Что такое пространства имен? В широком смысле - это один из способов инкапсуляции элементов. Такое абстрактное понятие можно увидеть во многих местах. Например, в любой операционной системе директории служат для группировки связанных файлов и выступают в качестве пространства имен для находящихся в них файлов. В качестве конкретного примера файл *foo.txt* может находиться сразу

в обоих директориях: */home/greg* и */home/other*, но две копии *foo.txt* не могут существовать в одной директории. Кроме того, для доступа к *foo.txt* извне директории */home/greg*, мы должны добавить имя директории перед именем файла используя разделитель, чтобы получить */home/greg/foo.txt*. Этот же принцип распространяется и на пространства имен в программировании.

В PHP пространства имен используются для решения двух проблем, с которыми сталкиваются авторы библиотек и приложений при создании повторно используемых элементов кода, таких как классы и функции:

1. Конфликт имен между вашим кодом и внутренними классами/функциями/константами PHP или сторонними.
2. Возможность создавать псевдонимы (или сокращения) для Ну_Очень_Длинных_Имен, чтобы облегчить первую проблему и улучшить читаемость исходного кода.

Пространства имен PHP предоставляют возможность группировать логически связанные классы, интерфейсы, функции и константы.

Пример #1 Пример синтаксиса, использующего пространство имен

```
<?php
namespace my\name; // см. раздел "Определение пространств имен"

class MyClass {}
function myfunction() {}
const MYCONST = 1;

$a = new MyClass;
$c = new \my\name\MyClass; // см. раздел "Глобальная область вид
имости"

$a = strlen('hi'); // см. раздел "Использование пространств имен
: возврат
                   // к глобальной функции/константе"

$d = namespace\MYCONST; // см. раздел "оператор пространства име
н и
                        // константа __NAMESPACE__"
$d = __NAMESPACE__ . '\MYCONST';
echo constant($d); // см. раздел "Пространства имен и динамическ
ие особенности языка"
?>
```

Замечание:
Названия пространств имен *PHP* и *php*, и составные названия, начинающиеся с этих (такие как *PHP\Classes*), являются зарезервированными для нужд языка и их не следует использовать в пользовательском коде.

User Contributed Notes 5 notes

Определение пространств имен ¶
(PHP 5 >= 5.3.0, PHP 7)

Хотя любой исправный PHP-код может находиться внутри пространства имен, только классы (включая абстрактные и трейты), интерфейсы, функции и константы зависят от него.

Пространства имен объявляются с помощью зарезервированного слова *namespace*. Файл, содержащий пространство имен, должен содержать его объявление в начале перед любым другим кодом, кроме зарезервированного слова declare.

Пример #1 Объявление единого пространства имен

```
<?php
namespace MyProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

?>
```

Только выражение *declare* может находиться перед объявлением пространства имен для указания кодировки файла. Кроме того, объявлению пространства имен не должен предшествовать не PHP-код, в том числе лишние пробелы:

Пример #2 Объявление простого пространства имен

```
<html>
<?php
namespace MyProject; // fatal error -
 объявле-
ние пространства имен должно быть первым выражением в скрипте
?>
```

Кроме того, в отличии от любой другой конструкции PHP, одно и тоже пространство имен можно определять в нескольких файлах, что позволяет распределять находящееся в них по файловой системе.

User Contributed Notes 9 notes

Определение подпространств имен ¶

(PHP 5 >= 5.3.0, PHP 7)

Так же, как файлы и каталоги, пространства имен PHP позволяют создавать иерархию имен. Таким образом, имя пространства может быть определено с подуровнями:

Пример #1 Определение пространства имен с иерархией

```
<?php
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
```

```
?>
```
Вышеприведенный пример создает константу *MyProject\Sub\Level\CONNECT_OK*, класс *MyProject\Sub\Level\Connection* и функцию*MyProject\Sub\Level\connect*.

Описание нескольких пространств имен в одном файле ¶
(PHP 5 >= 5.3.0, PHP 7)

Несколько пространств имен также можно описать в одном файле с помощью двух допустимых синтаксических конструкций.

Пример #1 Описание нескольких пространств имен, простой синтаксис

```
<?php
namespace MyProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

namespace AnotherProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
?>
```

Данный синтаксис не рекомендуется для комбинирования пространств имен в одном файле. Вместо этого рекомендуется использовать альтернативный синтаксис со скобками.

Пример #2 Описание нескольких пространств имен, синтаксис со скобками

```
<?php
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace AnotherProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}
?>
```

Настоятельно не рекомендуется при программировании комбинировать несколько пространств имен в один файл. Основным применением этому может быть объединение нескольких PHP файлов в один файл.

Для объединения кода в глобальном пространстве имен с кодом в других пространствах имен, используется только синтаксис со скобками. Глобальный код должен быть помещен в конструкцию описания пространства имен без указания имени:

Пример #3 Описание глобального и обычного пространства имен в одном файле

```
<?php
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace {        // глобальный код
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
}
?>
```

PHP-код не может находиться вне скобок конструкции пространства имен, кроме начального выражения declare.

Пример #4 Описание глобального и обычного пространства имен в одном файле

```
<?php
declare(encoding='UTF-8');
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace {      // глобальный код
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
}
?>
```

User Contributed Notes 8 notes

Использование пространства имен: основы ¶

(PHP 5 >= 5.3.0, PHP 7)

До обсуждения использования пространств имен важно понять как PHP узнает какие элементы из пространства имен запрашиваются в вашем коде. Можно провести аналогию между пространствами имен PHP и файловой системой. Есть три способа обратиться к файлу в файловой системе:

1. Относительное имя файла, такое как *foo.txt*, преобразуемое в *currentdirectory/foo.txt*, где currentdirectory текущая директория, в которой мы находимся. Тогда, если текущая директория */home/foo*, то имя преобразуется в */home/foo/foo.txt*.
2. Относительное имя пути, такое как *subdirectory/foo.txt*, преобразуется в *currentdirectory/subdirectory/foo.txt*.
3. Абсолютное имя пути, такое как */main/foo.txt*, которое остается таким же: */main/foo.txt*.

Тот же принцип применим и к элементам из пространств имен PHP. Для примера, имя класса может быть указано тремя способами:

1. Неполные имена (имена классов без префикса), такие как *$a = new foo();* или *foo::staticmethod();*. Если текущее пространство имен*currentnamespace*, то эти имена преобразуются в *currentnamespace\foo*. Если код находится в глобальном пространстве имен, то имена остаются такими же: *foo*. Предупреждение: неполные имена для функций и констант будут определяться в глобальном пространстве имен, если они не определены в текущем пространстве имен. Подробнее в Использование пространств имен: доступ к глобальным функциям и классам.
2. Полные имена (имена классов с префиксами), такие как *$a = new subnamespace\foo();* или *subnamespace\foo::staticmethod();*. Если текущее пространство имен *currentnamespace*, то эти имена преобразуются в *currentnamespace\subnamespace\foo*. Если код находится в глобальном пространстве имен, то имена преобразуются в *subnamespace\foo*.
3. Абсолютные имена или имена с предшествующим префиксом, обозначающим глобальное пространство. *$a = new \currentnamespace\foo();* или *\currentnamespace\foo::staticmethod();*. *И*мена всегда определяются также как и записаны:*currentnamespace\foo*.

Ниже приведен пример трех вариантов синтаксиса в реальном коде:

file1.php
```php
<?php
namespace Foo\Bar\subnamespace;

const FOO = 1;
function foo() {}
class foo
{
    static function staticmethod() {}
}
?>
```
file2.php
```php
<?php
namespace Foo\Bar;
include 'file1.php';

const FOO = 2;
function foo() {}
class foo
{
    static function staticmethod() {}
```

```
}

/* Неполные имена */
foo(); // определяется как функция Foo\Bar\foo
foo::staticmethod(); // определяется как класс Foo\Bar\foo с мет
одом staticmethod
echo FOO; // определяется как константа Foo\Bar\FOO

/* Полные имена */
sub-
namespace\foo(); // определяется как функция Foo\Bar\subnamespac
e\foo
sub-
namespace\foo::staticmethod(); // определяется как класс Foo\Bar
\subnamespace\foo
                                    // с методом staticmethod
echo subnamespace\FOO; // определяется как константа Foo\Bar\sub
namespace\FOO

/* Абсолютные имена */
\Foo\Bar\foo(); // определяется как функция Foo\Bar\foo
\Foo\Bar\foo::staticmethod(); // определяется как класс Foo\Bar\
foo с методом staticmethod
echo \Foo\Bar\FOO; // определяется как константа Foo\Bar\FOO
?>
```

Обратите внимание, что для доступа к любым глобальным классам, функциям или константам, может использоваться абсолютное имя, такое как \strlen(), или \Exception, или *\INI_ALL*.

Пример #1 Доступ к глобальным классам, функциям и константам из пространства имен

```
<?php
namespace Foo;

function strlen() {}
const INI_ALL = 3;
class Exception {}

$a = \strlen('hi'); // вызывает глобальную функцию strlen
$b = \INI_ALL; // получает доступ к глобальной константе INI_ALL
$c = new \Exception('error'); // Создает экземпляр глобального к
ласса Exception
?>
```

User Contributed Notes 8 notes

Пространства имен и динамические особенности языка ¶

(PHP 5 >= 5.3.0, PHP 7)

На реализацию пространств имен в PHP повлияли и динамические особенности языка. Преобразуем нижеследующий код для использования пространств имен:

Пример #1 Динамически доступные элементы

example1.php:

```php
<?php
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "global";

$a = 'classname';
$obj = new $a; // выводит classname::__construct
$b = 'funcname';
$b(); // выводит funcname
echo constant('constname'), "\n"; // выводит global
?>
```

Необходимо использовать абсолютное имя (имя класса с префиксом пространства имен). Обратите внимание, что нет никакой разницы между полным именем и абсолютным внутри динамического имени класса, функции или константы. Начальный обратный слеш не является необходимым.

Пример #2 Динамически доступные элементы пространства имен

```php
<?php
namespace namespacename;
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "namespaced";

include 'example1.php';

$a = 'classname';
$obj = new $a; // выводит classname::__construct
$b = 'funcname';
$b(); // выводит funcname
echo constant('constname'), "\n"; // выводит global
```

```php
/* обратите внимание, что при использовании двойных кавычек симв
ол обратного слеша должен быть заэкранирован. Например, "\\names
pacename\\classname" */
$a = '\namespacename\classname';
$obj = new $a; // выводит namespacename\classname::__construct
$a = 'namespacename\classname';
$obj = new $a; // также выводит namespacename\classname::__const
ruct
$b = 'namespacename\funcname';
$b(); // выводит namespacename\funcname
$b = '\namespacename\funcname';
$b(); // также выводит namespacename\funcname
echo constant('\namespacename\constname'), "\n"; // выводит name
spaced
echo constant('namespacename\constname'), "\n"; // также выводит
 namespaced
?>
```

Обязательно прочитайте примечание об экранировании имен пространства имен в строках.

User Contributed Notes 4 notes

Ключевое слово namespace и константа __NAMESPACE__ ¶

(PHP 5 >= 5.3.0, PHP 7)

PHP поддерживает два способа к абстрактно доступным элементам в текущем пространстве имен таким, как магическая константа__**NAMESPACE**__ и ключевое слово *namespace*.

Значение константы __**NAMESPACE**__ - это строка, которая содержит имя текущего пространства имен. В глобальном пространстве, вне пространства имен, она содержит пустую строку.

Пример #1 Пример использование константы __NAMESPACE__ в коде с пространством имен

```php
<?php
namespace MyProject;

echo '"', __NAMESPACE__, '"'; // выводит "MyProject"
?>
```

Пример #2 Пример использование константы __NAMESPACE__ в глобальном пространстве

```php
<?php

echo '"', __NAMESPACE__, '"'; // выводит ""
?>
```

Константа __**NAMESPACE**__ полезна для динамически конструируемых имен, например:

Пример #3 использование константы __NAMESPACE__ для динамического конструирования имени

```
<?php
namespace MyProject;

function get($classname)
{
    $a = __NAMESPACE__ . '\\' . $classname;
    return new $a;
}
?>
```

Ключевое слово *namespace* может быть использовано для явного запроса элемента из текущего пространства имен или из подпространства. Это эквивалент оператора *self* для классов в пространстве имен.

Пример #4 Оператор namespace, внутри пространства имен

```
<?php
namespace MyProject;

use blah\blah as mine; // см. "Использование пространств имен: импорт/создание псевдонима имени"

blah\mine(); // вызывает функцию MyProject\blah\mine()
namespace\blah\mine(); // вызывает функцию MyProject\blah\mine()

namespace\func(); // вызывает функцию MyProject\func()
namespace\sub\func(); // вызывает функцию MyProject\sub\func()
namespace\cname::method(); // вызывает статический метод "method" класса MyProject\cname
$a = new namespace\sub\cname(); // Создает экземпляр класса MyProject\sub\cname
$b = namespace\CONSTANT; // присваивает значение константы MyProject\CONSTANT переменной $b
?>
```

Пример #5 Оператор namespace в глобальном коде

```
<?php

namespace\func(); // вызывает функцию  func()
namespace\sub\func(); // вызывает функцию  sub\func()
namespace\cname::method(); // вызывает статический метод "method" класса cname
$a = new namespace\sub\cname(); // Создает экземпляр класса sub\cname
$b = namespace\CONSTANT; // присваивает значение константы CONSTANT переменной $b
?>
```

User Contributed Notes 1 note

Использование пространств имен: импорт/создание псевдонима имени ¶

(PHP 5 >= 5.3.0, PHP 7)

Возможность ссылаться на внешнее абсолютное имя по псевдониму или импортирование - это важная особенность пространств имен. Это похоже на возможность файловых систем unix создавать символические ссылки на файл или директорию.

Все версии PHP, поддерживающие пространства имен, поддерживают три вида создания псевдонима имени или импорта: создание псевдонима для имени класса, создание псевдонима для имени интерфейса и для имени пространства имен. PHP 5.6+ также поддерживает импорт функций и имен констант.

В PHP создание псевдонима имени выполняется с помощью оператора *use*. Вот пример, показывающий 5 типов импорта:

Пример #1 импорт/создание псевдонима имени с помощью оператора use

```
<?php
namespace foo;
use My\Full\Classname as Another;

// это тоже самое, что и использование My\Full\NSname как NSname
use My\Full\NSname;

// импортирование глобального класса
use ArrayObject;

// импортирование функции (PHP 5.6+)
use function My\Full\functionName;

// псевдоним функции (PHP 5.6+)
use function My\Full\functionName as func;

// импортирование константы (PHP 5.6+)
use const My\Full\CONSTANT;

$obj = new namespace\Another; // создает экземпляр класса foo\An
other
$obj = new Another; // создает объект класса My\Full\Classname
NSname\subns\func(); // вызывает функцию My\Full\NSname\subns\fu
nc
$a = new ArrayObject(array(1)); // создает объект класса ArrayOb
ject
// без выражения "use ArrayObject" мы создадим объект класса foo
\ArrayObject
func(); // вызывает функцию My\Full\functionName
echo CONSTANT; // выводит содержимое константы My\Full\CONSTANT
?>
```

Обратите внимание, что для имен в пространстве имен (абсолютные имена, содержащие разделитель пространств имен, такие как*Foo\Bar*, в отличие от гло-

бальных имен, которые его не содержат, такие как *FooBar*) нет необходимости в начальном обратном слеше (\) и его присутствие там не рекомендуется, так как импортируемые имена должны быть абсолютными и не обрабатываются относительно текущего пространства имен.

PHP дополнительно поддерживает удобное сокращение для задания нескольких операторов use в одной и той же строке

Пример #2 импорт/создание псевдонима имени с помощью оператора use, комбинирование нескольких операторов use

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // создает объект класса My\Full\Classname
NSname\subns\func(); // вызывает функцию My\Full\NSname\subns\fu
nc
?>
```

Импорт выполняется во время компиляции, и не влияет на имена динамических классов, функций или констант.

Пример #3 Импорт и динамические имена

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // создает объект класса My\Full\Classname
$a = 'Another';
$obj = new $a;      // создает объект класса Another
?>
```

В дополнение, импорт распространяется только на неполные и полные имена. Абсолютные имена не затрагиваются операцией импорта.

Пример #4 Импортирование и абсолютные имена

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // создает объект класса My\Full\Classname
$obj = new \Another; // создает объект класса Another
$obj = new Another\thing; // создает объект класса My\Full\Class
name\thing
$obj = new \Another\thing; // создает объект класса Another\thin
g
?>
```

Scoping rules for importing ¶

Ключевое слово *use* должно быть указано в самом начале файла (в глобальной области) или внутри объявления пространства имен. Это необходимо потому, что импорт выполняется во время компиляции, а не во время исполнения, поэтому

оно не может быть заключено в блок. Следующий пример показывает недопустимое применение ключевого слова *use*:

Пример #5 Недопустимое правило импорта

```
<?php
namespace Languages;

class Greenlandic
{
    use Languages\Danish;

    ...
}
?>
```

Замечание:
Правила импорта задаются на каждый файл отдельно. Это означает, что присоединяемые файлы *НЕ* будут наследовать правила импорта из родительского файла.

User Contributed Notes 14 notes

Глобальное пространство ¶
(PHP 5 >= 5.3.0, PHP 7)

Без определения пространства имен, определения всех классов и функций находятся в глобальном пространстве - также как это было в PHP до введения пространств имен. Добавление префикса \ к именам означает, что это имя должно находиться в глобальном пространстве, даже если вы находитесь в контексте определенного пространства имен.

Пример #1 Использование глобального пространства и его задание

```
<?php
namespace A\B\C;

/* Эта функция является A\B\C\fopen */
function fopen() {
     /* ... */
     $f = \fopen(...); // вызов глобальной функции fopen
     return $f;
}
?>
```

User Contributed Notes 4 notes

Использование пространств имен: переход к глобальной функции/константе ¶
(PHP 5 >= 5.3.0, PHP 7)

Внутри пространства имен, когда PHP встречает неполное имя класса, функции или константы, он преобразует эти имена с разными приоритетами. Имена классов всегда преобразуются к текущему имени пространства имен. Таким образом,

чтобы получить доступ ко внутреннему классу или пользовательскому классу вне пространства имен, необходимо ссылаться по их абсолютному имени. Например:

Пример #1 Доступ к глобальным классам внутри пространства имен

```
<?php
namespace A\B\C;
class Exception extends \Exception {}

$a = new Exception('hi'); // $a -
 это объект класса A\B\C\Exception
$b = new \Exception('hi'); // $b - это объект класса Exception

$c = new ArrayObject; // фатальная ошибка, класс A\B\C\ArrayObje
ct не найден
?>
```

Для функций и констант, PHP будет прибегать к глобальным функциям или константам, если функция или константа не существует в пространстве имен.

Пример #2 Необходимость прибегнуть к глобальным функциям/константам внутри пространства имен

```
<?php
namespace A\B\C;

const E_ERROR = 45;
function strlen($str)
{
    return \strlen($str) - 1;
}

echo E_ERROR, "\n"; // выводит "45"
echo INI_ALL, "\n"; // выводит "7" -
 прибегнет к глобальной INI_ALL

echo strlen('hi'), "\n"; // выводит "1"
if (is_array('hi')) { // выводит строку "это не массив"
    echo "это массив\n";
} else {
    echo "это не массив\n";
}
?>
```

User Contributed Notes 1 note

Правила разрешения имен ¶

(PHP 5 >= 5.3.0, PHP 7)

Для этих правил здесь приведены несколько важных определений:

Определения имени пространства имен
Неполное имя

Это идентификатор без разделителя пространств имен, например, *Foo*

Полное имя

Это идентификатор с разделителем пространств имен, например, *Foo\Bar*

Абсолютное имя

Это идентификатор с разделителем пространств имен, который начинается с разделителя пространств имен, например,*\Foo\Bar*. Пространство имен *\Foo* также является абсолютным именем.

Имена разрешаются согласно следующим правилам:

1. Вызовы абсолютных функций, классов или констант разрешаются во время компиляции. Например, *new \A\B* разрешается в класс*A\B*.
2. Все неполные и полные имена (не абсолютные) переводятся в процессе компиляции в соответствии с текущими правилами импорта. Например, если пространство имен *A\B\C* импортировано как *C*, вызов *C\D\e()* преобразуется к *A\B\C\D\e()*.
3. Внутри пространства имен все полные имена, не переведенные в соответствии с правилами импорта, получают префикс текущего пространства имен. Например, если происходит вызов *C\D\e()*, он преобразуется внутри пространства имен *A\B* к *A\B\C\D\e()*.
4. Неполные имена классов преобразуются в процессе компиляции в соответствии с текущими правилами импорта (полное имя заменено на короткое импортируемое имя). Например, если пространство имен *A\B\C* импортировано как C, выражение *new C()*преобразовывается к выражению *new A\B\C()*.
5. Внутри пространства имен (скажем, A\B), вызовы к неполным именам функций преобразуются во время исполнения. Вот, к примеру, как преобразуется вызов функции *foo()*:
    1. Производится поиск функции из текущего пространства имен: *A\B\foo()*.
    2. PHP пытается найти и вызвать функцию *глобального пространства foo()*.
6. Внутри пространства имен (скажем, *A\B*), вызовы к неполным или полным именам классов (неабсолютным) преобразуются во время исполнения. Вот как выражение *new C()* или *new D\E()* преобразуется. Для *new C()*:
    1. Ищется класс из текущего пространства имен: *A\B\C*.
    2. Производится попытка автозагрузки *A\B\C*.

    Для *new D\E()*:

    3. Ищется класс с помощью префиксации текущего пространства имен: *A\B\D\E*.
    4. Производится попытка автозагрузки *A\B\D\E*.

    Для обращения к любому глобальному классу в глобальном пространстве, должно использоваться его абсолютное имя *new \C()*.

Пример #1 Примеры разрешения имен

```php
<?php
namespace A;
use B\D, C\E as F;

// вызовы функций

foo();      // сперва пытается вызвать "foo", определенную в про
странстве имен "A",
            // затем вызывает глобальную функцию "foo"

\foo();     // вызывает функцию "foo", определенную в глобальном
 пространстве

my\foo();   // вызывает функцию "foo", определенную в пространст
ве "A\my"

F();        // сперва пытается вызвать "F", определенную в прост
ранстве имен "A",
            // затем вызывает глобальную функцию "F"

// ссылки на классы

new B();    // создает объект класса "B", определенного в простр
анстве имен "A".
            // если не найден, то пытается сделать автозагрузку
класса "A\B"

new D();    // используя правила импорта, создает объект класса
"D", определенного в пространстве имен "B"
            // если не найден, то пытается сделать автозагрузку
класса "B\D"

new F();    // используя правила импорта, создает объект класса
"E", определенного в пространстве имен "C"
            // если не найден, то пытается сделать автозагрузку
класса "C\E"

new \B();   // создает объект класса "B", определенного в глобал
ьном пространстве,
            // если не найден, то пытается сделать автозагрузку
класса "B"

new \D();   // создает объект класса "D", определенного в глобал
ьном пространстве,
            // если не найден, то пытается сделать автозагрузку
класса "D"

new \F();   // создает объект класса "F", определенного в глобал
ьном пространстве,
            // если не найден, то пытается сделать автозагрузку
класса "F"

// статические методы/функции пространства имен из другого прост
```

```
ранства имен

B\foo();    // вызывает функцию "foo" из пространства имен "A\B"

B::foo();   // вызывает метод "foo" из класса "B", определенного
 в пространстве имен "A"
            // если класс "A\B" не найден, то пытается сделать а
втозагрузку класса "A\B"

D::foo();   // используя правила импорта, вызывает метод "foo" к
ласса "D", определенного в пространстве имен "B"
            // если класс "B\D" не найден, то пытается сделать а
втозагрузку класса "B\D"

\B\foo();   // вызывает функцию "foo" из пространства имен "B"

\B::foo();  // вызывает метод "foo" класса "B" из глобального пр
остранства
            // если класс "B" не найден, то пытается сделать авт
озагрузку класса "B"

// статические методы/функции пространства имен из текущего прос
транства имен

A\B::foo();   // вызывает метод "foo" класса "B" из пространства
 имен "A\A"
              // если класс "A\A\B" не найден, то пытается сдела
ть автозагрузку класса "A\A\B"

\A\B::foo();  // вызывает метод "foo" класса "B" из пространства
 имен "A"
              // если класс "A\B" не найден, то пытается сделать
 автозагрузку класса "A\B"
?>
```

User Contributed Notes 10 notes

Часто задаваемые вопросы (FAQ): вещи, которые вам необходимо знать о пространствах имен ¶

(PHP 5 >= 5.3.0, PHP 7)

Этот список вопросов разделен на две части: общие вопросы и некоторые особенности реализации, которые полезны для более полного понимания.

Сперва, общие вопросы.

1. Если я не использую пространства имен, следует ли считать что-либо из этого важным ?
2. Как мне использовать внутренние или глобальные классы в пространстве имен ?
3. Как мне использовать функции классов в пространствах имен, или константы в их собственном пространстве имен ?
4. Как такое имя как *\my\name* или *\name* преобразуется ?

5. Как такое имя, как *my\name* преобразуется ?
6. Как неполное имя класса такое как *name* преобразуется ?
7. Как неполное имя функции или неполное имя константы такое как *name* преобразуется ?

Некоторые детали реализации пространств имен, которые полезно понимать.

1. Импортируемые имена не могут конфликтовать с классами, определенными в том же файле.
2. Вложенные пространства имен недопустимы.
3. Ни функции, ни константы не могут быть импортированы с помощью оператора *use*.
4. Динамические имена пространств имен (идентификаторы, взятые в кавычки) должны экранировать символ обратного слеша.
5. Ссылаться на неопределенные константы, используя обратный слеш, нельзя. Выводится фатальная ошибка
6. Невозможно переопределить специальные константы, такие как NULL, TRUE, FALSE, ZEND_THREAD_SAFE или ZEND_DEBUG_BUILD

Если я не использую пространства имен, следует ли считать что-либо из этого важным ? ¶

Нет. Пространства имен не оказывают никакого влияния ни на какой существующий код ни в каком виде или на любой написанный код, который не содержит пространств имен. Вы можете написать такой код, если желаете:

Пример #1 Доступ к глобальным классам вне пространства имен

```
<?php
$a = new \stdClass;
?>
```

Это функционально эквивалентно следующему:

Пример #2 Доступ к глобальным классам вне пространства имен

```
<?php
$a = new stdClass;
?>
```

Как мне использовать внутренние или глобальные классы в пространстве имен ? ¶

Пример #3 Доступ ко внутренним классам в пространствах имен

```
<?php
namespace foo;
$a = new \stdClass;

function test(\ArrayObject $typehintexample = null) {}

$a = \DirectoryIterator::CURRENT_AS_FILEINFO;
```

```php
// расширение внутреннего или глобального класса
class MyException extends \Exception {}
?>
```

Как мне использовать функции классов в пространствах имен, или константы в их собственном пространстве имен ? ¶

Пример #4 Доступ ко внутренним классам, функциям или константам в пространствах имен

```php
<?php
namespace foo;

class MyClass {}

// использование класса из текущего пространства имен
function test(MyClass $typehintexample = null) {}
// другой способ использовать класс из текущего пространства име
н
function test(\foo\MyClass $typehintexample = null) {}

// расширение класса из текущего пространства имен
class Extended extends MyClass {}

// доступ к глобальной функции
$a = \globalfunc();

// доступ к глобальной константе
$b = \INI_ALL;
?>
```

Как такое имя как *\my\name* или *\name* преобразуется ? ¶

Имена, которые начинаются с \ всегда преобразуются к тому как они выглядят, т.е. *\my\name* - это на самом деле *my\name*, и*\Exception* - это *Exception*.

Пример #5 Абсолютные имена

```php
<?php
namespace foo;
$a = new \my\name(); // создает экземпляр класса "my\name"
echo \strlen('hi'); // вызывает функцию "strlen"
$a = \INI_ALL; // переменной $a присваивается значение константы
 "INI_ALL"
?>
```

Как такое имя, как *my\name* преобразуется ? ¶

Имена, которые содержат обратный слеш, но не начинаются с него, такие как *my\name* могут быть преобразованы двумя различными способами.

Если присутствует импортирующее выражение, которое создает синоним *my* другого имени, то этот синоним применяется к *my* в*my\name*.

В ином случае, текущее имя пространства имен становится префиксом к *my\name*.

Пример #6 Полные имена

```
<?php
namespace foo;
use blah\blah as foo;

$a = new my\name(); // создает экземпляр класса "foo\my\name"
foo\bar::name(); // вызывает статический метод "name" в классе "
blah\blah\bar"
my\bar(); // вызывает функцию "foo\my\bar"
$a = my\BAR; // присваивает переменной $a значение константы "fo
o\my\BAR"
?>
```

Как неполное имя класса такое как *name* преобразуется ? ¶

Имена классов, которые не содержат обратный слеш, такие как *name* могут быть преобразованы двумя различными способами.

Если присутствует импортирующее выражение, которое создает синоним *name* другого имени, то применяется этот синоним.

В ином случае, текущее имя пространства имен становится префиксом к *my\name*.

Пример #7 Неполные имена классов

```
<?php
namespace foo;
use blah\blah as foo;

$a = new name(); // создает экземпляр класса "foo\name"
foo::name(); // вызывает статический метод "name" в классе "blah
\blah"
?>
```

Как неполное имя функции или неполное имя константы такое как *name* преобразуется ? ¶

Имена функций или констант, которые не содержат обратного слеша, такие как *name* могут быть преобразованы двумя различными способами.

Сперва, текущее имя пространства имен становится префиксом к *name*.

Затем, если константа или функция *name* не существует в текущем пространстве имен, используется глобальная константа или функция *name*, если она существует.

Пример #8 Неполные имена функций или констант

```
<?php
namespace foo;
```

```php
use blah\blah as foo;

const FOO = 1;

function my() {}
function foo() {}
function sort(&$a)
{
    \sort($a); // вызывает глобальную функцию "sort"
    $a = array_flip($a);
    return $a;
}

my(); // вызывает "foo\my"
$a = strlen('hi'); // вызывает глобальную функцию "strlen", пото
му что "foo\strlen" не существует
$arr = array(1,3,2);
$b = sort($arr); // вызывает функцию "foo\sort"
$c = foo(); // вызывает функцию "foo\foo" -
 импорт не применяется

$a = FOO; // присваивает переменной $a значение константы "foo\F
OO" - импорт не применяется
$b = INI_ALL; // присваивает переменной $b значение глобальной к
онстанты "INI_ALL"
?>
```

Импортируемые имена не могут конфликтовать с классами, определенными в том же файле. ¶

Следующие комбинации скриптов допустимы:

file1.php
```php
<?php
namespace my\stuff;
class MyClass {}
?>
```
another.php
```php
<?php
namespace another;
class thing {}
?>
```
file2.php
```php
<?php
namespace my\stuff;
include 'file1.php';
include 'another.php';

use another\thing as MyClass;
$a = new MyClass; // создает экземпляр класса "thing" из простра
нства имен "another"
?>
```

Конфликт имен отсутствует даже несмотря на то, что класс *MyClass* существует внутри пространства имен *my\stuff*, потому что определение MyClass находится в отдельном файле. Однако следующий пример приводит к фатальной ошибке с конфликтом имен, потому что класс MyClass определен в том же файле, где находится оператор use.

```
<?php
namespace my\stuff;
use another\thing as MyClass;
class MyClass {} // фатальная ошибка: MyClass конфликтует с выра
жением импорта
$a = new MyClass;
?>
```

Вложенные пространства имен недопустимы. ¶

PHP не позволяет вложение пространств имен одно в другое

```
<?php
namespace my\stuff {
    namespace nested {
        class foo {}
    }
}
?>
```

Однако сымитировать вложенные пространства имен так:

```
<?php
namespace my\stuff\nested {
    class foo {}
}
?>
```

Ни функции, ни константы не могут быть заимпортированы с помощью оператора *use*. ¶

Элементы, которые подвержены действию оператора *use* - это пространства имен и имена классов. Для сокращения длинных имен констант или функций, заимпортируйте их содержимое в пространство имен.

```
<?php
namespace mine;
use ultra\long\ns\name;

$a = name\CONSTANT;
name\func();
?>
```

Динамические имена пространств имен (идентификаторы, взятые в кавычки) должны экранировать символ обратного слеша. ¶

Очень важно представлять это, потому что обратный слеш используется как экранирующий символ внутри строк. Он всегда должен быть продублирован, когда ис-

пользуется внутри строки, иначе появляется риск возникновения неумышленных последствий:

Пример #9 Подводные камни при использовании имени пространства имен внутри строки с двойными кавычками

```
<?php
$a = "dangerous\name"; // \n -
 это переход на новую строку внутри строки с двойными кавычками!
$obj = new $a;

$a = 'not\at\all\dangerous'; // а тут нет проблем.
$obj = new $a;
?>
```

Внутри строк, заключенных в одинарные кавычки, обратный слеш в качестве разделителя более безопасен, но по-прежнему рекомендуемая практика экранирования обратного слеша во всех строках является наилучшим вариантом.

Ссылаться на неопределенные константы, используя обратный слеш, нельзя. Выводится фатальная ошибка ¶

Любая неопределенная константа, являющаяся неполным именем, как *FOO*, будет приводить к выводу сообщения о том, что PHP предположил, что *FOO* было значение константы. Любая константа, с полным именем или абсолютным, которая содержит символ обратного слеша будет приводить к фатальной ошибке, если не будет найдена.

Пример #10 Неопределенные константы

```
<?php
namespace bar;
$a = FOO; // выводит предупреждение: undefined constants "FOO" a
ssumed "FOO";
$a = \FOO; // фатальная ошибка: undefined namespace constant FOO
$a = Bar\FOO; // фатальная ошибка: undefined namespace constant
bar\Bar\FOO
$a = \Bar\FOO; // фатальная ошибка: undefined namespace constant
 Bar\FOO
?>
```

Невозможно переопределить специальные константы, такие как NULL, TRUE, FALSE, ZEND_THREAD_SAFE или ZEND_DEBUG_BUILD ¶

Любая попытка определить константу пространства имен, которая совпадает с названиями специальных встроенных констант, приведет к фатальной ошибке.

Пример #11 Неопределенные константы

```
<?php
namespace bar;
const NULL = 0; // Фатальная ошибка;
const true = 'stupid'; // также фатальная ошибка;
```

```
// и т.д.
?>
```

User Contributed Notes 4 notes

Errors ¶

Содержание ¶

Introduction ¶

Sadly, no matter how careful we are when writing our code, errors are a fact of life. PHP will report errors, warnings and notices for many common coding and runtime problems, and knowing how to detect and handle these errors will make debugging much easier.

Basics ¶

PHP reports errors in response to a number of internal error conditions. These may be used to signal a number of different conditions, and can be displayed and/or logged as required.

Every error that PHP generates includes a type. A list of these types is available, along with a short description of their behaviour and how they can be caused.

Handling errors with PHP ¶

If no error handler is set, then PHP will handle any errors that occur according to its configuration. Which errors are reported and which are ignored is controlled by the `error_reporting` php.ini directive, or at runtime by calling error_reporting(). It is strongly recommended that the configuration directive be set, however, as some errors can occur before execution of your script begins.

In a development environment, you should always set `error_reporting` to **`E_ALL`**, as you need to be aware of and fix the issues raised by PHP. In production, you may wish to set this to a less verbose level such as `E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED`, but in many cases **`E_ALL`** is also appropriate, as it may provide early warning of potential issues.

What PHP does with these errors depends on two further php.ini directives. `display_errors` controls whether the error is shown as part of the script's output. This should always be disabled in a production environment, as it can include confidential information such as database passwords, but is often useful to enable in development, as it ensures immediate reporting of issues.

In addition to displaying errors, PHP can log errors when the `log_errors` directive is enabled. This will log any errors to the file or syslog defined by `error_log`. This can be extremely useful in a production environment, as you can log errors that occur and then generate reports based on those errors.

User error handlers ¶

If PHP's default error handling is inadequate, you can also handle many types of error with your own custom error handler by installing it with set_error_handler(). While some error types cannot be handled this way, those that can be handled can then be handled in the way that your script sees fit: for example, this can be used to show a custom error page to the user and then report more directly than via a log, such as by sending an e-mail.

Errors in PHP 7 ¶

PHP 7 changes how most errors are reported by PHP. Instead of reporting errors through the traditional error reporting mechanism used by PHP 5, most errors are now reported by throwing Error exceptions.

As with normal exceptions, these Error exceptions will bubble up until they reach the first matching *catch* block. If there are no matching blocks, then any default exception handler installed with set_exception_handler() will be called, and if there is no default exception handler, then the exception will be converted to a fatal error and will be handled like a traditional error.

As the Error hierarchy does not inherit from Exception, code that uses `catch (Exception $e) { ... }` blocks to handle uncaught exceptions in PHP 5 will find that these Errors are not caught by these blocks. Either a `catch (Error $e) { ... }` block or a set_exception_handler()handler is required.

Error hierarchy ¶

- Throwable
  - Error
    - ArithmeticError
      - DivisionByZeroError
    - AssertionError
    - ParseError
    - TypeError
  - Exception
    - ...

User Contributed Notes 2 notes

Исключения ¶

Содержание ¶

Модель исключений (exceptions) в PHP 5 схожа с используемыми в других языках программирования. Исключение можно сгенерировать (как говорят, "выбросить") при помощи оператора *throw*, и можно перехватить (или, как говорят, "поймать") оператором *catch*. Код генерирующий исключение, должен быть окружен блоком *try*, для того чтобы можно было перехватить исключение. Каждый

блок *try* должен иметь как минимум один соответствующий ему блок *catch* или *finally*.

Генерируемый объект должен принадлежать классу Exception или наследоваться от Exception. Попытка сгенерировать исключение другого класса приведет к неисправимой ошибке.

catch

Можно использовать несколько блоков *catch*, перехватывающих различные классы исключений. Нормальное выполнение (когда не генерируются исключения в блоках *try* или когда класс сгенерированного исключения не совпадает с классами, объявленными в соответствующих блоках *catch*) будет продолжено за последним блоком *catch*. Исключения также могут быть сгенерированы (или вызваны еще раз) оператором *throw* внутри блока *catch*.

При генерации исключения код следующий после описываемого выражения исполнен не будет, а PHP предпримет попытку найти первый блок *catch*, перехватывающий исключение данного класса. Если исключение не будет перехвачено, PHP выдаст сообщение об ошибке: "*Uncaught Exception ...*" (Неперехваченное исключение), если не был определен обработчик ошибок при помощи функцииset_exception_handler().

finally

В PHP 5.5 и более поздних версиях также можно использовать блок *finally* после или вместо блока *catch*. Код в блоке *finally* всегда будет выполняться после кода в блоках *try* и *catch*, вне зависимости было ли брошено исключение или нет, перед тем как продолжится нормальное выполнение кода. whether an exception has been thrown, and before normal execution resumes.

Примечания

Замечание:
Внутренние функции PHP в основном используют сообщения об ошибках, и только новые объектно-ориентированные расширения используют исключения. Однако ошибки можно легко преобразовать в исключения с помощью класса ErrorException.

Подсказка
Стандартная библиотека PHP (SPL) предоставляет хороший набор встроенных классов исключений.

Примеры

Пример #3 Выброс исключений

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Деление на ноль.');
    }
    return 1/$x;
}
```

```
try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Выброшено исключение: ',  $e->getMessage(), "\n";
}

// Продолжение выполнения
echo "Hello World\n";
?>
```

Результат выполнения данного примера:

```
0.2
Выброшено исключение: Деление на ноль.
Hello World
```

Пример #4 Вложенные исключения

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Деление на ноль.');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Поймано исключение: ',  $e->getMessage(), "\n";
} finally {
    echo "Первое finally.\n";
}

try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Поймано исключение: ',  $e->getMessage(), "\n";
} finally {
    echo "Второе finally.\n";
}

// Продолжение нормального выполнения
echo "Hello World\n";
?>
```

Результат выполнения данного примера:

```
0.2
Первое finally.
Поймано исключение: Деление на ноль.
Второе finally.
```

```
Hello World
```

Пример #5 Вложенные исключения

```php
<?php

class MyException extends Exception { }

class Test {
    public function testing() {
        try {
            try {
                throw new MyException('foo!');
            } catch (MyException $e) {
                // повторный выброс исключения
                throw $e;
            }
        } catch (Exception $e) {
            var_dump($e->getMessage());
        }
    }
}

$foo = new Test;
$foo->testing();

?>
```

Результат выполнения данного примера:

```
string(4) "foo!"
```

User Contributed Notes 26 notes

Наследование исключений ¶

Определенный пользователем класс исключения должен быть определен, как класс расширяющий (наследующий) встроенный класс Exception. Ниже приведены методы и свойства класса Exception, доступные дочерним классам.

Пример #1 Встроенный класс Exception

```php
<?php
class Exception
{
    protected $message = 'Unknown exception';   // Сообщение
    private   $string;                          // Свойство для
__toString
    protected $code = 0;                        // Код исключени
я,
                                                // определяемый
пользователем
    protected $file;                            // Файл в которо
```

```
м было
                                                // выброшено иск
лючение
    protected $line;                            // Строка в кото
рой было
                                                // выброшено иск
лючение
    private   $trace;                           // Трассировка в
ызовов методов и функций
    private   $previous;                        // Предыдущее ис
ключение, для
                                                // вложенных бло
ков try

    public function __construct($message = null, $code = 0, Exce
ption $previous = null);

    final private function __clone();           // Запрещает кло
нировать исключения

    final public  function getMessage();        // Возвращает со
общение исключения
    final public  function getCode();           // Код исключени
я
    final public  function getFile();           // Файл, где выб
рошено исключение
    final public  function getLine();           // Строка, выбро
сившая исключени
    final public  function getTrace();          // Массив backtr
ace()
    final public  function getPrevious();       // Предыдущее ис
ключение
    final public  function getTraceAsString();  // Трассировка в
ызовов как строка

    // Переопределяемое
    public function __toString();               // форматированн
ая строка для отображения
}
?>
```

Если класс, наследуемый от Exception переопределяет конструктор, необходимо вызвать в конструкторе parent::__construct(), чтобы быть уверенным, что все данные будут доступны. Метод __toString() может быть переопределен, что бы обеспечить нужный вывод, когда объект преобразуется в строку.

Замечание:
Исключения нельзя клонировать. Попытка клонировать исключение приведет к неисправимой ошибке **E_ERROR**.

Пример #2 Наследование класса Exception (PHP 5.3.0+)

```php
<?php
/**
 * Определим свой класс исключения
 */
class MyException extends Exception
{
    // Переопределим исключение так, что параметр message станет
 обязательным
    public function __construct($message, $code = 0, Exception $
previous = null) {
        // некоторый код

        // убедитесь, что все передаваемые параметры верны
        parent::__construct($message, $code, $previous);
    }

    // Переопределим строковое представление объекта.
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this-
>message}\n";
    }

    public function customFunction() {
        echo "Мы можем определять новые методы в наследуемом кла
сce\n";
    }
}


/**
 * Создадим класс для тестирования исключения
 */
class TestException
{
    public $var;

    const THROW_NONE    = 0;
    const THROW_CUSTOM  = 1;
    const THROW_DEFAULT = 2;

    function __construct($avalue = self::THROW_NONE) {

        switch ($avalue) {
            case self::THROW_CUSTOM:
                // Бросаем собственное исключение
                throw new MyException('1 -
 неправильный параметр', 5);
                break;

            case self::THROW_DEFAULT:
                // Бросаем встроеное исключение
                throw new Exception('2 -
 недопустимый параметр', 6);
```

```
                break;

            default:
                // Никаких исключений, объект будет создан.
                $this->var = $avalue;
                break;
        }
    }
}


// Example 1
try {
    $o = new TestException(TestException::THROW_CUSTOM);
} catch (MyException $e) {      // Will be caught
    echo "Поймано собственное переопределенное исключение\n", $e
;
    $e->customFunction();
} catch (Exception $e) {        // Будет пропущено.
    echo "Поймано встроенное исключение\n", $e;
}

// Отсюда будет продолжено выполнение программы
var_dump($o); // Null
echo "\n\n";


// Example 2
try {
    $o = new TestException(TestException::THROW_DEFAULT);
} catch (MyException $e) {      // Тип исключения не совпадет
    echo "Поймано переопределенное исключение\n", $e;
    $e->customFunction();
} catch (Exception $e) {        // Будет перехвачено
    echo "Перехвачено встроенное исключение\n", $e;
}

// Отсюда будет продолжено выполнение программы
var_dump($o); // Null
echo "\n\n";


// Example 3
try {
    $o = new TestException(TestException::THROW_CUSTOM);
} catch (Exception $e) {        // Будет перехвачено.
    echo "Поймано встроенное исключение\n", $e;
}

// Продолжение исполнения программы
var_dump($o); // Null
echo "\n\n";
```

```php
// Example 4
try {
    $o = new TestException();
} catch (Exception $e) {        // Будет пропущено, т.к. исключение не выбрасывается
    echo "Поймано встроенное исключение\n", $e;
}

// Продолжение выполнения программы
var_dump($o); // TestException
echo "\n\n";
?>
```

Замечание:
PHP 5 до версии 5.3.0 не поддерживает вложенные исключения. Если есть необходимость запустить вышеприведенный пример в указанных версиях PHP, то нужно заменить класс MyException.

```php
<?php
/**
 * Определим свой класс исключения
 */
class MyException extends Exception
{
    // Переопределим исключение так, что параметр message станет обязательным
    public function __construct($message, $code = 0) {
        // некоторый код

        // убедитесь, что все передаваемые параметры верны
        parent::__construct($message, $code);
    }

    // Переопределим строковое представление объекта.
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }

    public function customFunction() {
        echo "Мы можем определять новые методы в наследуемом классе\n";
    }
}
?>
```

User Contributed Notes 8 notes

Generators ¶

Содержание ¶

Generators overview ¶

(PHP 5 >= 5.5.0, PHP 7)

Generators provide an easy way to implement simple iterators without the overhead or complexity of implementing a class that implements the Iterator interface.

A generator allows you to write code that uses foreach to iterate over a set of data without needing to build an array in memory, which may cause you to exceed a memory limit, or require a considerable amount of processing time to generate. Instead, you can write a generator function, which is the same as a normal function, except that instead of returning once, a generator can yield as many times as it needs to in order to provide the values to be iterated over.

A simple example of this is to reimplement the range() function as a generator. The standard range() function has to generate an array with every value in it and return it, which can result in large arrays: for example, calling range(0, 1000000) will result in well over 100 MB of memory being used.

As an alternative, we can implement an *xrange()* generator, which will only ever need enough memory to create an Iterator object and track the current state of the generator internally, which turns out to be less than 1 kilobyte.

Пример #1 Implementing range() as a generator

```
<?php
function xrange($start, $limit, $step = 1) {
    if ($start < $limit) {
        if ($step <= 0) {
            throw new LogicException('Step must be +ve');
        }

        for ($i = $start; $i <= $limit; $i += $step) {
            yield $i;
        }
    } else {
        if ($step >= 0) {
            throw new LogicException('Step must be -ve');
        }

        for ($i = $start; $i >= $limit; $i += $step) {
            yield $i;
        }
    }
}

/*
 * Note that both range() and xrange() result in the same
 * output below.
 */
```

```
echo 'Single digit odd numbers from range():  ';
foreach (range(1, 9, 2) as $number) {
    echo "$number ";
}
echo "\n";

echo 'Single digit odd numbers from xrange(): ';
foreach (xrange(1, 9, 2) as $number) {
    echo "$number ";
}
?>
```

Результат выполнения данного примера:

```
Single digit odd numbers from range():  1 3 5 7 9
Single digit odd numbers from xrange(): 1 3 5 7 9
```

Generator objects ¶

When a generator function is called for the first time, an object of the internal Generator class is returned. This object implements theIterator interface in much the same way as a forward-only iterator object would, and provides methods that can be called to manipulate the state of the generator, including sending values to and returning values from it.

User Contributed Notes 6 notes

Generator syntax ¶

A generator function looks just like a normal function, except that instead of returning a value, a generator yields as many values as it needs to.

When a generator function is called, it returns an object that can be iterated over. When you iterate over that object (for instance, via aforeach loop), PHP will call the generator function each time it needs a value, then saves the state of the generator when the generator yields a value so that it can be resumed when the next value is required.

Once there are no more values to be yielded, then the generator function can simply exit, and the calling code continues just as if an array has run out of values.

Замечание:
A generator cannot return a value: doing so will result in a compile error. An empty return statement is valid syntax within a generator and it will terminate the generator.

yield keyword ¶

The heart of a generator function is the yield keyword. In its simplest form, a yield statement looks much like a return statement, except that instead of stopping execution of the function and returning, yield instead provides a value to the code looping over the generator and pauses execution of the generator function.

Пример #1 A simple example of yielding values

```php
<?php
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}

$generator = gen_one_to_three();
foreach ($generator as $value) {
    echo "$value\n";
}
?>
```

Результат выполнения данного примера:

```
1
2
3
```

Замечание:
Internally, sequential integer keys will be paired with the yielded values, just as with a non-associative array.

Предостережение
If you use yield in an expression context (for example, on the right hand side of an assignment), you must surround the yield statement with parentheses in PHP 5. For example, this is valid:

```
$data = (yield $value);
```

But this is not, and will result in a parse error in PHP 5:

```
$data = yield $value;
```

The parenthetical restrictions do not apply in PHP 7.

This syntax may be used in conjunction with the Generator::send() method.

Yielding values with keys ¶

PHP also supports associative arrays, and generators are no different. In addition to yielding simple values, as shown above, you can also yield a key at the same time.

The syntax for yielding a key/value pair is very similar to that used to define an associative array, as shown below.

Пример #2 Yielding a key/value pair

```php
<?php
/*
 * The input is semi-colon separated fields, with the first
 * field being an ID to use as a key.
 */

$input = <<<'EOF'
```

```
1;PHP;Likes dollar signs
2;Python;Likes whitespace
3;Ruby;Likes blocks
EOF;

function input_parser($input) {
    foreach (explode("\n", $input) as $line) {
        $fields = explode(';', $line);
        $id = array_shift($fields);

        yield $id => $fields;
    }
}

foreach (input_parser($input) as $id => $fields) {
    echo "$id:\n";
    echo "    $fields[0]\n";
    echo "    $fields[1]\n";
}
?>
```

Результат выполнения данного примера:

```
1:
    PHP
    Likes dollar signs
2:
    Python
    Likes whitespace
3:
    Ruby
    Likes blocks
```

Предостережение

As with the simple value yields shown earlier, yielding a key/value pair in an expression context requires the yield statement to be parenthesised:

```
$data = (yield $key => $value);
```

Yielding null values ¶

Yield can be called without an argument to yield a **NULL** value with an automatic key.

Пример #3 Yielding **NULL**s

```
<?php
function gen_three_nulls() {
    foreach (range(1, 3) as $i) {
        yield;
    }
}

var_dump(iterator_to_array(gen_three_nulls()));
?>
```

Результат выполнения данного примера:

```
array(3) {
  [0]=>
  NULL
  [1]=>
  NULL
  [2]=>
  NULL
}
```

Yielding by reference ¶

Generator functions are able to yield values by reference as well as by value. This is done in the same way as returning references from functions: by prepending an ampersand to the function name.

Пример #4 Yielding values by reference

```
<?php
function &gen_reference() {
    $value = 3;

    while ($value > 0) {
        yield $value;
    }
}

/*
 * Note that we can change $number within the loop, and
 * because the generator is yielding references, $value
 * within gen_reference() changes.
 */
foreach (gen_reference() as &$number) {
    echo (--$number).'... ';
}
?>
```

Результат выполнения данного примера:

```
2... 1... 0...
```

*Generator delegation via **yield from*** ¶

In PHP 7, generator delegation allows you to yield values from another generator, Traversable object, or array by using the yield from keyword. The outer generator will then yield all values from the inner generator, object, or array until that is no longer valid, after which execution will continue in the outer generator.

If a generator is used with yield from, the yield from expression will also return any value returned by the inner generator.

Предостережение

Storing into an array (e.g. with iterator_to_array())

yield from does not reset the keys. It preserves the keys returned by the Traversable object, or array. Thus some values may share a common key with another yield or yield from, which, upon insertion into an array, will overwrite former values with that key.

A common case where this matters is iterator_to_array() returning a keyed array by default, leading to possibly unexpected results.iterator_to_array() has a second parameter `use_keys` which can be set to **FALSE** to collect all the values while ignoring the keys returned by the Generator.

Пример #5 yield from with iterator_to_array()

```
<?php
function from() {
    yield 1; // key 0
    yield 2; // key 1
    yield 3; // key 2
}
function gen() {
    yield 0; // key 0
    yield from from(); // keys 0-2
    yield 4; // key 1
}
// pass false as second parameter to get an array [0, 1, 2, 3, 4
]
var_dump(iterator_to_array(gen()));
?>
```

Результат выполнения данного примера:

```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(4)
  [2]=>
  int(3)
}
```

Пример #6 Basic use of yield from

```
<?php
function count_to_ten() {
    yield 1;
    yield 2;
    yield from [3, 4];
    yield from new ArrayIterator([5, 6]);
    yield from seven_eight();
    yield 9;
    yield 10;
}

function seven_eight() {
    yield 7;
```

```
    yield from eight();
}

function eight() {
    yield 8;
}

foreach (count_to_ten() as $num) {
    echo "$num ";
}
?>
```

Результат выполнения данного примера:

```
1 2 3 4 5 6 7 8 9 10
```

Пример #7 yield from and return values

```
<?php
function count_to_ten() {
    yield 1;
    yield 2;
    yield from [3, 4];
    yield from new ArrayIterator([5, 6]);
    yield from seven_eight();
    return yield from nine_ten();
}

function seven_eight() {
    yield 7;
    yield from eight();
}

function eight() {
    yield 8;
}

function nine_ten() {
    yield 9;
    return 10;
}

$gen = count_to_ten();
foreach ($gen as $num) {
    echo "$num ";
}
echo $gen->getReturn();
?>
```

Результат выполнения данного примера:

```
1 2 3 4 5 6 7 8 9 10
```

User Contributed Notes 9 notes

Comparing generators with Iterator objects ¶

The primary advantage of generators is their simplicity. Much less boilerplate code has to be written compared to implementing an Iteratorclass, and the code is generally much more readable. For example, the following function and class are equivalent:

```
<?php
function getLinesFromFile($fileName) {
    if (!$fileHandle = fopen($fileName, 'r')) {
        return;
    }

    while (false !== $line = fgets($fileHandle)) {
        yield $line;
    }

    fclose($fileHandle);
}

// versus...

class LineIterator implements Iterator {
    protected $fileHandle;

    protected $line;
    protected $i;

    public function __construct($fileName) {
        if (!$this->fileHandle = fopen($fileName, 'r')) {
            throw new RuntimeException('Couldn\'t open file "' .
 $fileName . '"');
        }
    }

    public function rewind() {
        fseek($this->fileHandle, 0);
        $this->line = fgets($this->fileHandle);
        $this->i = 0;
    }

    public function valid() {
        return false !== $this->line;
    }

    public function current() {
        return $this->line;
    }

    public function key() {
        return $this->i;
    }
```

```
    public function next() {
        if (false !== $this->line) {
            $this->line = fgets($this->fileHandle);
            $this->i++;
        }
    }

    public function __destruct() {
        fclose($this->fileHandle);
    }
}
?>
```

This flexibility does come at a cost, however: generators are forward-only iterators, and cannot be rewound once iteration has started. This also means that the same generator can't be iterated over multiple times: the generator will need to be rebuilt by calling the generator function again.

User Contributed Notes 3 notes

Ссылки. Разъяснения ¶

Содержание ¶

User Contributed Notes 48 notes

Dave at SymmetricDesigns dot com ¶

8 years ago

```
Another example of something to watch out for when using refer-
ences with arrays.  It seems that even an ususued reference to an
array cell modifies the *source* of the reference.  Strange be-
havior for an assignment statement (is this why I've seen it
written as an =& operator?  - although this doesn't happen with
regular variables).
<?php
    $array1 = array(1,2);
    $x = &$array1[1];   // Unused reference
    $array2 = $array1;  // reference now also applies to $array2
!
    $array2[1]=22;      // (changing [0] will not affect $ar-
ray1)
    print_r($array1);
?>
Produces:
```

```
    Array
    (
    [0] => 1
    [1] => 22    // var_dump() will show the & here
    )

I fixed my bug by rewriting the code without references, but it
can also be fixed with the unset() function:
<?php
    $array1 = array(1,2);
    $x = &$array1[1];
    $array2 = $array1;
    unset($x); // Array copy is now unaffected by above refer-
ence
    $array2[1]=22;
    print_r($array1);
?>
Produces:
    Array
    (
    [0] => 1
    [1] => 2
    )
```

Что такое ссылки ¶

Ссылки в PHP - это средство доступа к содержимому одной переменной под разными именами. Они не похожи на указатели C; например, вы не можете делать вычисления над указателями, они не являются реальными адресами в памяти и пр. Подробнее в Чем ссылки не являются. Вместо этого указатели в PHP - это псевдонимы в таблице имен переменных. В PHP имя переменной и её содержимое - это разные вещи, поэтому одно содержимое может иметь разные имена. Ближайшая аналогия - имена файлов Unix и файлы - имена переменных являются элементами каталогов, а содержимое переменных - это сами файлы. Ссылки в PHP - аналог жёстких ссылок (hardlinks) в файловых системах Unix.

User Contributed Notes 1 note

stilezy ¶

5 months ago

```
One subtle effect of PHP's assign-by-reference is that operators
which might be expected to work with args that are references
usually don't.  For example:

$a = ($b ? &$c : &$d);

fails (parser error) but the logically identical

if ($b)
   $a =& $c;
else
   $a =& $cd;

works. It's not always obvious why seemingly identical code
throws an error in the first case. This is discussed on a PHP
```

bug report (https://bugs.php.net/bug.php?id=54740). TL;DR version, it acts more like an assignment term ($var1) "=&" ($var2) than a function/operator ($var1) "=" (&$var2).

Что делают ссылки ¶

Есть три основных операции с использованием ссылок: присвоение по ссылке, передача по ссылке, и возврат по ссылке. Данный раздел познакомит вас с этими операциями и предоставит ссылки для дальнейшего изучения.

Присвоение по ссылке ¶

Для начала, ссылки PHP позволяют создать две переменные указывающие на одно и тоже значение. Таким образом, когда выполняется следующее:

```
<?php
$a =& $b;
?>
```

то *$a* указывает на то же значение что и *$b*.

Замечание:
$a и $b здесь абсолютно эквивалентны, но это не означает, что $a указывает на $b или наоборот. Это означает, что $a и$b указывают на одно и тоже значение.

Замечание:
При присвоении, передаче или возврате неинициализированной переменной по ссылке, происходит ее создание.

Пример #1 Использование ссылок с неинициализированными переменными

```
<?php
function foo(&$var) { }

foo($a); // $a создана и равна null

$b = array();
foo($b['b']);
var_dump(array_key_exists('b', $b)); // bool(true)

$c = new StdClass;
foo($c->d);
var_dump(property_exists($c, 'd')); // bool(true)
?>
```

Такой же синтаксис можно использовать и в функциях, возвращая ссылки, а также в операторе *new* (начиная с PHP 4.0.4 и до PHP 5.0.0):

```
<?php
$foo =& find_var($bar);
?>
```

Начиная с PHP 5, new автоматически возвращает объекты по ссылке, поэтому использовать =& в этом случае нерекомендуется и вызывает сообщение **E_DEPRECATED** в PHP 5.3 и последующих версиях, а также сообщение **E_STRICT** в более ранних версиях. (Технически разница в том, что в PHP 5 переменные объектов, так же как и ресурсы, являются просто указателями на ре-

альные данные объекта, поэтому ссылки на объекты не те же самые "ссылки", о которых говорилось ранее (псевдонимы). Подробнее вОбъекты и ссылки.)

Внимание

Если переменной, объявленной внутри функции как *global*, будет присвоена ссылка, она будет видна только в функции. Чтобы избежать этого, используйте массив $GLOBALS.

Пример #2 Присвоение ссылок глобальным переменным внутри функции

```
<?php
$var1 = "Example variable";
$var2 = "";

function global_references($use_globals)
{
    global $var1, $var2;
    if (!$use_globals) {
        $var2 =& $var1; // только локально
    } else {
        $GLOBALS["var2"] =& $var1; // глобально
    }
}

global_references(false);
echo "значение var2: '$var2'\n"; // значение var2: ''
global_references(true);
echo "значение var2: '$var2'\n"; // значение var2: 'Example vari
able'
?>
```

Думайте о *global $var;* как о сокращении от *$var =& $GLOBALS['var'];*. Таким образом, присвоение *$var* другой ссылки влияет лишь на локальную переменную.

Замечание:

При использовании переменной-ссылки в foreach, изменяется содержание, на которое она ссылается.

Пример #3 Ссылки и foreach

```
<?php
$ref = 0;
$row =& $ref;
foreach (array(1, 2, 3) as $row) {
    // do something
}
echo $ref; // 3 - последнее значение, используемое в цикле
?>
```

Хотя, стого говоря, в выражениях, создаваемых с помощью конструкции *array()*, нет присвоения, но тем не менее можно указать префикс *&* для элементов массива. Пример:

```
<?php
$a = 1;
$b = array(2, 3);
$arr = array(&$a, &$b[0], &$b[1]);
$arr[0]++; $arr[1]++; $arr[2]++;
/* $a == 2, $b == array(3, 4); */
?>
```

Однако следует отметить, что ссылки в массивах являются потенциально опасными. При обычном (не по ссылке) присвоении массива, ссылки внутри этого массива сохраняются. Это также относится и к вызовам функций, когда массив передается по значению. Пример:

```
<?php
/* Присвоение скалярных переменных */
$a = 1;
$b =& $a;
$c = $b;
$c = 7; //$c не ссылка и не изменяет значений $a и $b

/* Присвоение массивов */
$arr = array(1);
$a =& $arr[0]; //$a и $arr[0] ссылаются на одно значение
$arr2 = $arr; //присвоение не по ссылке!
$arr2[0]++;
/* $a == 2, $arr == array(2) */
/* Содержимое $arr изменилось, хотя было присвоено не по ссылке!
 */
?>
```

Иными словами, поведение отдельных элементов массива не зависит от типа присвоения этого массива.

Pass By Reference ¶

Второе, что делают ссылки - передача параметров по ссылке. При этом локальная переменная в функции и переменная в вызывающей области видимости ссылаются на одно и то же содержимое. Пример:

```
<?php
function foo(&$var)
{
    $var++;
}

$a=5;
foo($a);
?>
```

Этот код присвоит *$a* значение 6. Это происходит, потому что в функции *foo* переменная *$var* ссылается на то же содержимое, что и переменная *$a*. См. также детальное объяснение передачи по ссылке.

Возврат по ссылке ¶

Третье, что могут делать ссылки - это возврат по ссылке.

User Contributed Notes 22 notes

Чем ссылки не являются ¶

Как уже было сказано, ссылки не являются указателями. Это означает, что следующая конструкция не будет делать то, что вы ожидаете:

```php
<?php
function foo(&$var)
{
    $var =& $GLOBALS["baz"];
}
foo($bar);
?>
```

Переменная `$var` в функции `foo` будет связана с `$bar` в вызывателе, но затем она будет перепривязана к `$GLOBALS["baz"]`. Нет способа связать `$bar` в области видимости вызывателя с чем-либо ещё путём использования механизма ссылок, поскольку `$bar` не доступна в функции `foo` (доступно лишь её значение через `$var`, но `$var` имеет только значение переменной и не имеет связи имя-значение в таблице имен переменных). Вы можете воспользоваться возвращением ссылок из функции для привязки внешней переменной к другому значению.

User Contributed Notes 10 notes

Передача по ссылке ¶

Вы можете передавать переменные в функцию по ссылке, и функция сможет изменять свои аргументы. Синтаксис таков:

```php
<?php
function foo(&$var)
{
    $var++;
}

$a=5;
foo($a);
// $a здесь равно 6
?>
```

Замечание: В вызове функции отсутствует знак ссылки - он есть только в определении функции. Этого достаточно для корректной передачи аргументов по ссылке. Начиная с PHP 5.3.0, вы можете получить предупреждение о том, что передача переменной по ссылке устарела, если используете & в *foo(&$a);*. Начиная с PHP 5.4.0 передача переменной по ссылке стала невозможна, поэтому использование этого приема приведет к фатальной ошибке.

По ссылке можно передавать:

- Переменные, например *foo($a)*
- Оператор new, например *foo(new foobar())*
- Ссылки, возвращаемые функцией, например:

```php
<?php
function foo(&$var)
{
    $var++;
}
```

```
function &bar()
{
    $a = 5;
    return $a;
}
foo(bar());
?>
```

См. также объяснение возвращения по ссылке.

Любое другое выражение не должно передаваться по ссылке, так как результат не определён. Например, следующая передача по ссылке является неправильной:

```
<?php
function foo(&$var)
{
    $var++;
}
function bar() // Операция & отсутствует
{
    $a = 5;
    return $a;
}
foo(bar()); // Вызывает неисправимую ошибку начиная с PHP 5.0.5

foo($a = 5); // Выражение, а не переменная
foo(5); // Константа, а не переменная
?>
```

Эти требования для PHP 4.0.4 и позже.

User Contributed Notes 9 notes

Возвращение по ссылке ¶

Возвращение по ссылке используется в тех случаях, когда вы хотите использовать функцию для выбора переменной, с которой должна быть связана данная ссылка. *Не* используйте возврат по ссылке для увеличения производительности. Ядро PHP само занимается оптимизацией. Применяйте возврат по ссылке только имея технические причины на это. При возвращении по ссылке используйте такой синтаксис:

```
<?php
class foo {
    public $value = 42;

    public function &getValue() {
        return $this->value;
    }
}

$obj = new foo;
$myValue = &$obj->getValue(); // $myValue указывает на $obj-
>value, равное 42.
```

```
$obj->value = 2;
echo $myValue;                  // отобразит новое значение $obj-
>value, т.е. 2.
?>
```

В этом примере устанавливается свойство объекта, возвращённого функцией *getValue*, а не его копии, как было бы без использования ссылок.

Замечание: В отличие от передачи параметров по ссылке, *&* здесь нужно использовать в обоих местах - для указания на то, что вы возвращаете ссылку, а не копию, как обычно, и для указания того, что происходит связывание по ссылке, а не обычное присвоение для `$myValue`.

Замечание: Если вы возвращаете ссылку из функции используя следующий синтаксис: *return ($this->value);*, это *не* будет работать, так как вы возвращаете по ссылке результат *выражения*, а не переменную. По ссылке можно возвращать только переменные и ничего больше. Начиная с PHP 4.4.0 и PHP 5.1.0, если код пытается вернуть по ссылке динамическое выражение или результат оператора *new*, будет выброшено предупреждение **E_NOTICE**.

Для использования возвращаемой ссылки вы должны применять присвоение по ссылке:

```
<?php
function &collector() {
  static $collection = array();
  return $collection;
}
$collection = &collector();
$collection[] = 'foo';
?>
```

Для передачи возвращаемой ссылки в другую функцию, принимающую ссылку, вы можете использовать следующий синтаксис:

```
<?php
function &collector() {
  static $collection = array();
  return $collection;
}
array_push(collector(), 'foo');
?>
```

Замечание: Заметим, что *array_push(&collector(), 'foo'); не* сработает, а приведет к неисправимой ошибке.

User Contributed Notes 17 notes

Сброс переменных-ссылок ¶

При сбросе ссылки, просто разрывается связь имени и содержимого переменной. Это не означает, что ,содержимое переменной будет разрушено. Например:

```
<?php
$a = 1;
$b =& $a;
unset($a);
?>
```

Этот код не сбросит *$b*, а только *$a*.

Опять же, можно провести аналогию с вызовом unlink (в Unix).

User Contributed Notes 5 notes

Неявное использование механизма ссылок ¶

Многие синтаксические конструкции PHP реализованы через механизм ссылок, поэтому всё сказанное выше о ссылочном связывании применимо также и к этим конструкциям. Некоторые конструкции, вроде передающих и возвращающих по ссылке, рассмотрены ранее. Другие конструкции, использующие ссылки:

Ссылки global ¶

Если вы объявляете переменную как global $var, вы фактически создаёте ссылку на глобальную переменную. Это означает то же самое, что и:

```
<?php
$var =& $GLOBALS["var"];
?>
```

Это значит, например, что сброс (unset) `$var` не приведёт к сбросу глобальной переменной.

$this ¶

В методах объекта, `$this` всегда является ссылкой на вызывающий объект.

User Contributed Notes 8 notes

Предопределённые переменные ¶

PHP предоставляет всем скриптам большое количество предопределённых переменных. Эти переменные содержат всё, от внешних данных до переменных среды окружения, от текста сообщений об ошибках до последних полученных заголовков.

См. также ЧАВО под названием "Как register_globals касаются меня?"

Содержание ¶

- $_COOKIE — HTTP Куки
- $php_errormsg — Предыдущее сообщение об ошибке
- $HTTP_RAW_POST_DATA — Необработанные POST-данные
- $http_response_header — Заголовки ответов HTTP
- $argc — Количество аргументов переданных скрипту
- $argv — Массив переданных скрипту аргументов

User Contributed Notes 43 notes

Суперглобальные переменные

Суперглобальные переменные — Суперглобальные переменные - это встроенные переменные, которые всегда доступны во всех областях видимости

Описание ¶

Некоторые предопределённые переменные в PHP являются "суперглобальными", что означает, что они доступны в любом месте скрипта. Нет необходимости использовать синтаксис global $variable; для доступа к ним в функциях и методах.

Суперглобальными переменными являются:

- *$GLOBALS*
- *$_SERVER*
- *$_GET*
- *$_POST*
- *$_FILES*
- *$_COOKIE*
- *$_SESSION*
- *$_REQUEST*
- *$_ENV*

Список изменений ¶

Версия	Описание
4.1.0	Суперглобальные переменные впервые появились в PHP.

Примечания ¶

Замечание: Доступность переменных
По умолчанию все суперглобальные переменные доступны всегда, однако существуют настройки, которые могут на это влиять. За дальнейшей информацией обращайтесь к описанию директивы variables_order.
Замечание: Работа с register_globals
В случае, если устаревшая директива register_globals включена (*on*), содержание суперглобальных массивов становится доступно в виде отдельных переменных. Например, `$_POST['foo']` будет также доступна в виде `$foo`.

За дополнительной информацией обращайтесь к ЧАВО под названием "Как register_globals касаются меня?"

Замечание: Переменные переменных
Суперглобальные переменные не могут быть использованы в качестве переменных переменных внутри функций и методов.

Смотрите также ¶

- variable scope
- Директива variables_order
- "Функции фильтрации данных"

User Contributed Notes 4 notes

$GLOBALS
(PHP 4, PHP 5, PHP 7)

$GLOBALS — Ссылки на все переменные глобальной области видимости

Описание ¶

Ассоциативный массив (array), содержащий ссылки на все переменные глобальной области видимости скрипта, определенные в данный момент. Имена переменных являются ключами массива.

Примеры ¶

Пример #1 Пример *$GLOBALS*

```
<?php
function test() {
    $foo = "local variable";

    echo '$foo in global scope: ' . $GLOBALS["foo"] . "\n";
    echo '$foo in current scope: ' . $foo . "\n";
}

$foo = "Example content";
test();
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
$foo in global scope: Example content
$foo in current scope: local variable
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Замечание: Доступность переменной
В отличии от всех остальных суперглобальных переменных, `$GLOBALS` всегда доступна в PHP.

User Contributed Notes 6 notes

$_SERVER

$HTTP_SERVER_VARS [удалено]
(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_SERVER -- $HTTP_SERVER_VARS [удалено] — Информация о сервере и среде исполнения

Описание ¶

Переменная `$_SERVER` - это массив, содержащий информацию, такую как заголовки, пути и местоположения скриптов. Записи в этом массиве создаются веб-сервером. Нет гарантии, что каждый веб-сервер предоставит любую из них; сервер может опустить некоторые из них или предоставить другие, не указанные здесь. Тем не менее, многие эти переменные присутствуют в » спецификации CGI/1.1, так что вы можете их ожидать их реализации и в конкретном веб-сервере.

Переменная `$HTTP_SERVER_VARS` содержит ту же начальную информацию, но она не суперглобальная. (Заметьте, что`$HTTP_SERVER_VARS` и `$_SERVER` являются разными переменными, так что PHP обрабатывает их соответственно). Также учтите, что "длинные массивы" были удалены в версии PHP 5.4.0, поэтому `$HTTP_SERVER_VARS` больше не существует.

Индексы ¶

Вы можете найти (а можете и не найти) любой из следующих элементов в массиве `$_SERVER`. Заметьте, что немногие элементы, если вообще такие найдутся, будут доступны (или действительно будут иметь значение), если PHP запущен в командной строке.

'*PHP_SELF*'
: Имя файла скрипта, который сейчас выполняется, относительно корня документов. Например,`$_SERVER['PHP_SELF']` в скрипте по адресу `http://example.com/foo/bar.php` будет `/foo/bar.php`. Константа __FILE__ содержит полный путь и имя файла текущего (то есть подключенного) файла. Если PHP запущен в командной строке, эта переменная содержит имя скрипта, начиная с PHP 4.3.0. Раньше она была недоступна.

'argv'
: Массив аргументов, переданных скрипту. Когда скрипт запущен в командой строке, это дает C-подобный доступ к параметрам командной строки. Когда вызывается через метод GET, этот массив будет содержать строку запроса.

'argc'
: Содержит количество параметров, переданных скрипту (если запуск произведен в командной строке).

'*GATEWAY_INTERFACE*'
: Содержит используемую сервером версию спецификации CGI; к примеру'*CGI/1.1*'.

'*SERVER_ADDR*'
: IP адрес сервера, на котором выполняется текущий скрипт.

'*SERVER_NAME*'
Имя хоста, на котором выполняется текущий скрипт. Если скрипт выполняется на виртуальном хосте, здесь будет содержатся имя, определенное для этого виртуального хоста.
'*SERVER_SOFTWARE*'
Строка идентификации сервера, указанная в заголовках, когда происходит ответ на запрос.
'*SERVER_PROTOCOL*'
Имя и версия информационного протокола, через который была запрошена страница; к примеру '*HTTP/1.0*';
'*REQUEST_METHOD*'
Какой метод был использован для запроса страницы; к примеру '*GET*', '*HEAD*', '*POST*', '*PUT*'.
Замечание:
PHP скрипт завершается после посылки заголовков (то есть после того, как осуществляет любой вывод без буферизации вывода), если запрос был осуществлен методом *HEAD*.
'*REQUEST_TIME*'
Временная метка начала запроса. Доступна, начиная с PHP 5.1.0.
'*REQUEST_TIME_FLOAT*'
Временная метка начала запроса с точностью до микросекунд. Доступна, начиная с PHP 5.4.0.
'*QUERY_STRING*'
Строка запросов, если есть, с помощью которой была получена страница.
'*DOCUMENT_ROOT*'
Директория корня документов, в которой выполняется текущий скрипт, в точности та, которая указана в конфигурационном файле сервера.
'*HTTP_ACCEPT*'
Содержимое заголовка *Accept:* из текущего запроса, если он есть.
'*HTTP_ACCEPT_CHARSET*'
Содержимое заголовка *Accept-Charset:* из текущего запроса, если он есть. Например: '*iso-8859-1,*,utf-8*'.
'*HTTP_ACCEPT_ENCODING*'
Содержимое заголовка *Accept-Encoding:* из текущего запроса, если он есть. Например: '*gzip*'.
'*HTTP_ACCEPT_LANGUAGE*'
Содержимое заголовка *Accept-Language:* из текущего запроса, если он есть. Например: '*en*'.
'*HTTP_CONNECTION*'
Содержимое заголовка *Connection:* из текущего запроса, если он есть. Например: '*Keep-Alive*'.
'*HTTP_HOST*'
Содержимое заголовка *Host:* из текущего запроса, если он есть.
'*HTTP_REFERER*'
Адрес страницы (если есть), которая привела браузер пользователя на эту страницу. Этот заголовок устанавливается веб-браузером пользователя. Не все браузеры устанавливают его и некоторые в качестве дополнительной возможности позволяют изменять содержимое заголовка `HTTP_REFERER`. Одним словом, в самом деле ему нельзя доверять.
'*HTTP_USER_AGENT*'
Содержимое заголовка *User-Agent:* из текущего запроса, если он есть. Эта строка содержит обозначение браузера, которым пользователь запросил

данную страницу. Типичным примером является строка: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Среди прочего, вы можете использовать это значение с функцией get_browser() чтобы адаптировать вывод вашей страницы к возможностям браузера пользователя

'*HTTPS*'

Принимает непустое значение, если запрос был произведен через протокол HTTPS.

Замечание: Обратите внимание, что при использовании ISAPI с IIS значение будет *off*, если запрос не был произведен через протокол HTTPS.

'*REMOTE_ADDR*'

IP-адрес, с которого пользователь просматривает текущую страницу.

'*REMOTE_HOST*'

Удаленный хост, с которого пользователь просматривает текущую страницу. Обратный просмотр DNS базируется на значении переменной `REMOTE_ADDR`.

Замечание: Ваш веб-сервер должен быть настроен, чтобы создавать эту переменную. Для примера, в Apache вам необходимо присутствие директивы *HostnameLookups On* в файле `httpd.conf`, чтобы эта переменная создавалась. См. такжеgethostbyaddr().

'*REMOTE_PORT*'

Порт на удаленной машине, который используется для связи с веб-сервером.

'*REMOTE_USER*'

Аутентифицированный пользователь.

'*REDIRECT_REMOTE_USER*'

Аутентифицированный пользователь, если запрос был перенаправлен изнутри.

'*SCRIPT_FILENAME*'

Абсолютный путь к скрипту, который в данный момент исполняется.

Замечание:
Если скрипт запускается в командной строке (CLI), используя относительный путь, такой как `file.php` или `../file.php`, переменная `$_SERVER['SCRIPT_FILENAME']` будет содержать относительный путь, указанный пользователем.

'*SERVER_ADMIN*'

Эта переменная получает свое значение (для Apache) из директивы конфигурационного файла сервера. Если скрипт запущен на виртуальном хосте, это будет значение, определенное для данного виртуального хоста.

'*SERVER_PORT*'

Порт на компьютере сервера, используемый веб-сервером для соединения. Для установок по умолчанию, значение будет '*80*'; используя SLL, например, это значение будет таким, какое сконфигурировано для соединений безопасного HTTP.

Замечание: Чтобы получить физический (реальный) порт в Apache 2, необходимо установить *UseCanonicalName = On* и*UseCanonicalPhysicalPort = On*, иначе это значение может быть подменено и не вернуть реальной значение физического порта. Полагаться на это значение небезопасно в контексте приложений, требующих усиленной безопасности.

'*SERVER_SIGNATURE*'

Строка, содержащая версию сервера и имя виртуального хоста, которые добавляются к генерируемым сервером страницам, если включено.

'*PATH_TRANSLATED*'

Filesystem- (not document root-) based path to the current script, after the server has done any virtual-to-real mapping.

Замечание: Начиная с PHP 4.3.2, переменная PATH_TRANSLATED больше не устанавливается неявно в Apache 2 SAPI, по сравнению с Apache версии 1, где она устанавливается в то же самое значение, что и переменная SCRIPT_FILENAME, когда она не используется Apache. Это изменение было сделано для соответствия спецификации CGI, где переменная PATH_TRANSLATEDдолжна существовать только тогда, когда PATH_INFO определена. Пользователи Apache 2 могут использовать директиву*AcceptPathInfo* = *On* в конфигурационном файле `httpd.conf` для задания переменной PATH_INFO.

'*SCRIPT_NAME*'

Содержит путь, к текущему исполняемому скрипту. Это полезно для страниц, которые должны указывать на самих себя. Константа __FILE__ содержит полный путь и имя текущего (т.е. включаемого) файла.

'*REQUEST_URI*'

URI, который был передан для того, чтобы получить доступ к этой странице. Например, '*/index.html*'.

'*PHP_AUTH_DIGEST*'

При выполнении HTTP Digest аутентификации, этой переменной присваивается заголовок 'Authorization', который присылается клиентом (его необходимо потом использовать для соответствующей валидации).

'*PHP_AUTH_USER*'

Когда выполняется HTTP-аутентификация, этой переменной присваивается имя пользователя, предоставленное пользователем.

'*PHP_AUTH_PW*'

Когда выполняется HTTP-аутентификация, этой переменной присваивается пароль, предоставленный пользователем.

'*AUTH_TYPE*'

Когда выполняется HTTP-аутентификация, этой переменной присваивается тип аутентификации, который используется.

'*PATH_INFO*'

Содержит любой предоставленный пользователем путь, содержащийся после имени скрипта, но до строки запроса, если доступно. Например, если текущий скрипт запрошен по URL `http://www.example.com/php/path_info.php/some/stuff?foo=bar`, то переменная `$_SERVER['PATH_INFO']` будет содержать */some/stuff*.

'*ORIG_PATH_INFO*'

Исходное значение переменной '`PATH_INFO`' до начала обработки PHP.

Список изменений ¶

Версия	Описание
5.4.0	Массив *$HTTP_SERVER_VARS* больше не доступен в связи с удалением "длинных массивов".
5.3.0	Директива register_long_arrays, которая приводила к заполнению *$HTTP_SERVER_VARS* помечена как устаревшая.
4.1.0	Введена переменная *$_SERVER* вместо старой *$HTTP_SERVER_VARS*.

Примеры ¶

Пример #1 Пример использования *$_SERVER*

```
<?php
echo $_SERVER['SERVER_NAME'];
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
www.example.com
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- Расширение filter

User Contributed Notes 55 notes

$_GET

$HTTP_GET_VARS [deprecated]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_GET -- $HTTP_GET_VARS [deprecated] — GET-переменные HTTP

Описание ¶

Ассоциативный массив параметров, переданных скрипту через URL.

`$HTTP_GET_VARS` содержит аналогичный набор данных, но не является суперглобальным. (Заметьте, что `$HTTP_GET_VARS` и `$_GET`являются разными переменными и обрабатываются PHP независимо друг от друга)

Список изменений ¶

Версия	Описание
4.1.0	Добавлена *$_GET*, заменяющая *$HTTP_GET_VARS*.

Примеры ¶

Пример #1 Пример использования *$_GET*

```
<?php
echo 'Привет ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

Подразумевается, что пользователь ввел в браузере адрес http://example.com/?name=Hannes

Результатом выполнения данного примера будет что-то подобное:

```
Привет Hannes!
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.
Замечание:
Параметры GET обрабатываются urldecode().

Смотрите также ¶

- Переменные извне PHP
- Функции фильтрации данных

User Contributed Notes 13 notes

$_POST

$HTTP_POST_VARS [deprecated]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_POST -- $HTTP_POST_VARS [deprecated] — HTTP POST variables

Описание ¶

Ассоциативный массив данных, переданных скрипту через HTTP метод POST.

`$HTTP_POST_VARS` содержит аналогичный набор данных, но не является суперглобальным. (`$HTTP_POST_VARS` и `$_POST` являются разными переменными и обрабатываются PHP независимо друг от друга)

Список изменений ¶

Версия	Описание
4.1.0	Введена переменная *$_POST*, заменяющая *$HTTP_POST_VARS*.

Примеры ¶

Пример #1 Пример использования *$_POST*

```php
<?php
echo 'Привет ' . htmlspecialchars($_POST["name"]) . '!';
?>
```

Подразумевается, что пользователь отправил через POST name=Hannes

Результатом выполнения данного примера будет что-то подобное:

```
Привет Hannes!
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- Обработка внешних переменных
- Функции фильтрации данных

User Contributed Notes 12 notes

$_FILES

$HTTP_POST_FILES [устаревшее]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_FILES -- $HTTP_POST_FILES [устаревшее] — Переменные файлов, загруженных по HTTP

Описание ¶

Ассоциативный массив (array) элементов, загруженных в текущий скрипт через метод HTTP POST.

`$HTTP_POST_FILES` содержит ту же начальную информацию, но не является суперглобальным. (Заметьте, что `$HTTP_POST_FILES` и`$_FILES` являются разными переменными, так что PHP обрабатывает их соответственно)

Список изменений ¶

Версия	Описание
4.1.0	Введен массив *$_FILES*, который должен заменить *$HTTP_POST_FILES*.

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- move_uploaded_file() - Перемещает загруженный файл в новое место
- Обработка загрузок файлов

User Contributed Notes 26 notes

$_REQUEST

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_REQUEST — Переменные HTTP-запроса

Описание ¶

Ассоциативный массив (array), который по умолчанию содержит данные переменных $_GET, $_POST и $_COOKIE.

Список изменений ¶

Версия	Описание
5.3.0	Добавлена директива request_order. Она позволяет контролировать содержимое переменной *$_REQUEST*.
4.3.0	Данные переменной *$_FILES* более не включаются в *$_REQUEST*.
4.1.0	Добавлена переменная *$_REQUEST*.

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.
Замечание:
При работе в командной строке переменные argv и argc *не* включаются в данный массив - они присутствуют в массиве $_SERVER.
Замечание:
Переменные в массиве $_REQUEST передаются в скрипт посредством методов GET, POST или COOKIE, поэтому им нельзя доверять, т.к. они могли быть изменены удаленным пользователем. Их наличие и порядок добавления данных в соответствующие массивы определяется директивой variables_order.

Смотрите также ¶

- import_request_variables() - Импортирует переменные GET/POST/Cookie в глобальную область видимости
- "Работа с внешними данными"
- "Функции фильтрации данных"

User Contributed Notes 5 notes

$_SESSION

$HTTP_SESSION_VARS [устаревшее]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_SESSION -- $HTTP_SESSION_VARS [устаревшее] — Переменные сессии

Описание ¶

Ассоциативный массив, содержащий переменные сессии, которые доступны для текущего скрипта. Смотрите документацию пофункциям сессии для получения дополнительной информации.

$HTTP_SESSION_VARS первоначально содержит ту же информацию, но она не является суперглобальной переменной. (Обратите внимание, что $HTTP_SESSION_VARS и $_SESSION являются различными переменными и в таковом качестве обрабатываются PHP).

Список изменений ¶

Версия	Описание
4.1.0	Добавлена *$_SESSION*, которая превратила *$HTTP_SESSION_VARS* в устаревшую.

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- session_start() - Start new or resume existing session

User Contributed Notes 11 notes

$_ENV

$HTTP_ENV_VARS [устаревшее]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_ENV -- $HTTP_ENV_VARS [устаревшее] — Переменные окружения

Описание ¶

Ассоциативный массив (array) значений, переданных скрипту через переменные окружения.

Эти значения импортируются в глобальное пространство имен PHP из системных переменных окружения, в котором запущен парсер PHP. Большинство значений передаётся из командной оболочки, под которой PHP запущен, и различных системных приложений, полного и точного списка не существует. Пожалуйста, изучите документацию к вашей командной оболочке для получения списка переменных окружения.

Некоторые переменные окружения включают в себя CGI переменные, причем их наличие не зависит от того, запущен PHP как модуль HTTP сервера или как CGI-модуль.

`$HTTP_ENV_VARS` содержит те же данные, но не является суперглобальной переменной. (Следует отметить, что `$HTTP_ENV_VARS` и`$_ENV` - различные переменные и обрабатываются по-разному)

Список изменений ¶

Версия	Описание
4.1.0	Введена *$_ENV* в замен устаревшей *$HTTP_ENV_VARS*.

Примеры ¶

Пример #1 Пример использования *$_ENV*

```php
<?php
echo 'Мое имя пользователя: ' .$_ENV["USER"] . '!';
?>
```

Допустим, скрипт запустил "bjori"

Результатом выполнения данного примера будет что-то подобное:

```
Мое имя пользователя: bjori!
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- getenv() - Получение значения переменной окружения
- Расширение фильтр

User Contributed Notes 6 notes

$_COOKIE

$HTTP_COOKIE_VARS [устаревшее]

(PHP 4 >= 4.1.0, PHP 5, PHP 7)

$_COOKIE -- $HTTP_COOKIE_VARS [устаревшее] — HTTP Куки

Описание ¶

Ассоциативный массив (array) значений, переданных скрипту через HTTP Куки.

`$HTTP_COOKIE_VARS` содержит те же данные, но не является суперглобальной переменной. (Следует отметить, что `$HTTP_COOKIE_VARS` и `$_COOKIE` - различные переменные и PHP обрабатывает их по-разному)

Список изменений ¶

Версия	Описание
4.1.0	Введена *$_COOKIE* для замены устаревшей *$HTTP_COOKIE_VARS*.

Примеры ¶

Пример #1 Пример использования *$_COOKIE*

```php
<?php
echo 'Привет, ' . htmlspecialchars($_COOKIE["name"]) . '!';
?>
```

Положим, что значение куки с именем "name" было установлено равным "Ханнес".

Результатом выполнения данного примера будет что-то подобное:

```
Привет, Ханнес!
```

Примечания ¶

Замечание:
Это 'суперглобальная' или автоматическая глобальная переменная. Это просто означает что она доступна во всех контекстах скрипта. Нет необходимости выполнять global $variable; для доступа к ней внутри метода или функции.

Смотрите также ¶

- setcookie() - Посылает cookie
- Обработка внешних переменных
- Расширение фильтр

User Contributed Notes 2 notes

$php_errormsg

(PHP 4, PHP 5, PHP 7)

$php_errormsg — Предыдущее сообщение об ошибке

Описание ¶

`$php_errormsg` является переменной, содержащей текст последней ошибки, сгенерированной PHP. Эта переменная будет доступна только в блоке кода, в котором случилась ошибка, и только если включена конфигурационная опция track_errors (по умолчанию отключена).

Внимание
Если настроен пользовательский обработчик ошибок (set_error_handler()), значение `$php_errormsg` будет установлено только, если обработчик ошибки вернет **FALSE**.

Примеры ¶

Пример #1 Пример использования *$php_errormsg*

```php
<?php
@strpos();
echo $php_errormsg;
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Wrong parameter count for strpos()
```

User Contributed Notes 3 notes

$HTTP_RAW_POST_DATA

(PHP 4, PHP 5)

$HTTP_RAW_POST_DATA — Необработанные POST-данные

Описание ¶

Внимание
Данная возможность была xmlns *УСТАРЕВШЕЙ* начиная с версии PHP 5.6.0. Крайне не рекомендуется полагаться на эту возможность в будущем.

`$HTTP_RAW_POST_DATA` содержит необработанные POST-данные. Также смотрите always_populate_raw_post_data.

В большинстве случаев нужно использовать *php://input* вместо `$HTTP_RAW_POST_DATA`.

User Contributed Notes 3 notes

Ray dot Paseur at GMail dot com ¶

6 years ago

To get the Raw Post Data:

```
<?php $postdata = file_get_contents("php://input"); ?>
```

Please see the notes here:
http://us.php.net/manual/en/wrappers.php.php

$http_response_header

(PHP 4 >= 4.0.4, PHP 5, PHP 7)

$http_response_header — Заголовки ответов HTTP

Описание ¶

Массив (array) `$http_response_header` похож на функцию get_headers(). Когда Вы используете обертку HTTP, `$http_response_header`будет заполнен заголовками ответов HTTP. `$http_response_header` будет создан в локальной области видимости.

Примеры ¶

Пример #1 Пример *$http_response_header*

```
<?php
function get_contents() {
  file_get_contents("http://example.com");
  var_dump($http_response_header);
}
get_contents();
var_dump($http_response_header);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
array(9) {
  [0]=>
  string(15) "HTTP/1.1 200 OK"
  [1]=>
```

```
  string(35) "Date: Sat, 12 Apr 2008 17:30:38 GMT"
  [2]=>
  string(29) "Server: Apache/2.2.3 (CentOS)"
  [3]=>
  string(44) "Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT"
  [4]=>
  string(27) "ETag: "280100-1b6-80bfd280""
  [5]=>
  string(20) "Accept-Ranges: bytes"
  [6]=>
  string(19) "Content-Length: 438"
  [7]=>
  string(17) "Connection: close"
  [8]=>
  string(38) "Content-Type: text/html; charset=UTF-8"
}
NULL
```

User Contributed Notes 2 notes

$argc

(PHP 4, PHP 5, PHP 7)

$argc — Количество аргументов переданных скрипту

Описание ¶

Содержит количество аргументов переданных скрипту при запуске из командной строки.

Замечание: Имя файла скрипта всегда передаётся в качестве первого аргумента, таким образом минимальное значение `$argc` равно *1*.
Замечание: Эта переменная не доступна когда register_argc_argv установлен в disabled.

Примеры ¶

Пример #1 Пример использования *$argc*

```php
<?php
var_dump($argc);
?>
```

Запустим пример в командной строке: php script.php arg1 arg2 arg3

Результатом выполнения данного примера будет что-то подобное:

```
int(4)
```

User Contributed Notes 5 notes

Tejesember ¶

5 years ago

To find out are you in CLI or not, this is much better in my opinion:

```
<?php
if (PHP_SAPI != "cli") {
    exit;
}
?>
```

$argv

(PHP 4, PHP 5, PHP 7)

$argv — Массив переданных скрипту аргументов

Описание ¶

Содержит массив array из всех аргументов переданных скрипту при запуске из командной строки.

Замечание: Первый аргумент `$argv[0]` всегда содержит имя файла запущенного скрипта.
Замечание: Эта переменная недоступна если register_argc_argv установлен в disabled.

Примеры ¶

Пример #1 Пример использования*$argv*

```
<?php
var_dump($argv);
?>
```

Запустим пример в командной строке: php script.php arg1 arg2 arg3

Результатом выполнения данного примера будет что-то подобное:

```
array(4) {
  [0]=>
  string(10) "script.php"
  [1]=>
  string(4) "arg1"
  [2]=>
  string(4) "arg2"
  [3]=>
  string(4) "arg3"
}
```

Смотрите также ¶

- getopt() - Извлечение параметров из списка аргументов командной строки

User Contributed Notes 7 notes

Предопределённые исключения ¶

Содержание ¶

Смотрите также "Исключения SPL"

Exception ¶

(PHP 5 >= 5.1.0, PHP 7)

Введение ¶

Exception - это базовый класс для всех исключений.

Обзор классов ¶

```
Exception {
/* Свойства */
protected string $message ;
protected int $code ;
protected string $file ;
protected int $line ;
/* Методы */
public __construct ([ string $message =          "" [, int $code =
0 [, Exception $previous = NULL ]]] )
final public string getMessage ( void )
final public Exception getPrevious ( void )
final public mixed getCode ( void )
final public string getFile ( void )
final public int getLine ( void )
final public array getTrace ( void )
final public string getTraceAsString ( void )
public string __toString ( void )
final private void __clone ( void )
}
```

Свойства ¶

message

Текст исключения

code

Код исключения

file

Имя файла, в котором было вызвано исключение

line

Номер строки, в которой было вызвано исключение

Содержание ¶

User Contributed Notes 1 note

cHao ¶

1 year ago

```
Note that an exception's properties are populated when the ex-
ception is *created*, not when it is thrown.  Throwing the ex-
ception does not seem to modify them.

Among other things, this means:

* The exception will blame the line that created it, not the
line that threw it.

* Unlike in some other languages, rethrowing an exception
doesn't muck up the trace.

* A thrown exception and an unthrown one look basically identi-
cal.  On my machine, the only visible difference is that a
thrown exception has an `xdebug_message` property while an un-
thrown one doesn't.  Of course, if you don't have xdebug in-
stalled, you won't even get that.
```

ErrorException ¶

(PHP 5 >= 5.1.0, PHP 7)

Введение ¶

Исключение в случае ошибки.

Обзор классов ¶

```
ErrorException extends Exception {
/* Свойства */
protected int $severity ;
/* Наследуемые свойства */
protected string $message ;
protected int $code ;
protected string $file ;
```

```
protected int $line ;
/* Методы */
public __construct ([ string $message =          "" [, int $code =
0 [, int $severity =                    1 [, string $filename =
__FILE__ [,int $lineno =
__LINE__ [, Exception $previous = NULL ]]]]]] )
final public int getSeverity ( void )
/* Наследуемые методы */
final public string Exception::getMessage ( void )
final public Exception Exception::getPrevious ( void )
final public mixed Exception::getCode ( void )
final public string Exception::getFile ( void )
final public int Exception::getLine ( void )
final public array Exception::getTrace ( void )
final public string Exception::getTraceAsString ( void )
public string Exception::__toString ( void )
final private void Exception::__clone ( void )
}
```

Свойства ¶

severity

Важность ошибки

Примеры ¶

Пример #1 Использование set_error_handler() для превращения сообщений об ошибках в ErrorException.

```
<?php
func-
tion exception_error_handler($errno, $errstr, $errfile, $errline
 ) {
    if (!(error_reporting() & $errno)) {
        // Этот код ошибки не входит в error_reporting
        return;
    }
    throw new ErrorException($errstr, 0, $errno, $errfile, $errl
ine);
}
set_error_handler("exception_error_handler");

/* вызываем исключение */
strpos();
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Fatal error: Uncaught exception 'ErrorException' with message
'strpos() expects at least 2 parameters, 0 given' in
/home/bjori/tmp/ex.php:12
Stack trace:
```

```
#0 [internal function]: exception_error_handler(2, 'strpos() ex-
pect...', '/home/bjori/php...', 12, Array)
#1 /home/bjori/php/cleandocs/test.php(12): strpos()
#2 {main}
  thrown in /home/bjori/tmp/ex.php on line 12
```

Содержание ¶

- ErrorException::__construct — Создание исключения
- ErrorException::getSeverity — Возвращает критичность исключения

User Contributed Notes 4 notes

Встроенные интерфейсы и классы ¶

Содержание ¶

- Traversable
- Iterator
- IteratorAggregate
- ArrayAccess
- Serializable
- Closure
- Generator

См. также Интерфейсы SPL и Предопределенные классы.

User Contributed Notes 1 note

articice at ua dot fm ¶

1 year ago

```
Note that PHP extensions also provide interfaces you can imple-
ment, for example JsonSerializable.
Here is the full
list http://php.net/manual/en/refs.basic.other.php
```

Интерфейс Traversable ¶

(PHP 5 >= 5.0.0, PHP 7)

Введение ¶

Интерфейс, определяющий, является ли класс обходимым (traversable) используя foreach.

Абстрактный базовый интерфейс, который не может быть реализован сам по себе. Вместо этого он должен быть реализован используя IteratorAggregate или Iterator.

Замечание:
Внутренние (встроенные) классы, которые реализуют этот интерфейс, могут быть использованы в конструкции foreach и не обязаны реализовывать IteratorAggregate или Iterator.
Замечание:

Это внутренний интерфейс, который не может быть реализован в скрипте PHP. Вместо него нужно использовать либоIteratorAggregate, либо Iterator. При реализации интерфейса, наследующего от Traversable, убедитесь, что в секции implements перед его именем стоит IteratorAggregate или Iterator.

Обзор интерфейсов ¶

```
Traversable {
}
```

Этот интерфейс не имеет методов, его единственная цель - быть базовым интерфейсом для всех обходимых классов.

User Contributed Notes 3 notes

Интерфейс Iterator ¶

(PHP 5 >= 5.0.0, PHP 7)

Введение ¶

Интерфейс для внешних итераторов или объектов, которые могут повторять себя изнутри.

Обзор интерфейсов ¶

```
Iterator extends Traversable {
/* Методы */
abstract public mixed current ( void )
abstract public scalar key ( void )
abstract public void next ( void )
abstract public void rewind ( void )
abstract public boolean valid ( void )
}
```

Предопределенные итераторы ¶

PHP уже предоставляет некоторые итераторы для многих ежедневных задач. См. список итераторов SPL для более детальной информации.

Примеры ¶

Пример #1 Основы использования

Этот пример демонстрирует в каком порядке вызываются методы, когда используется с итератором оператор foreach.

```
<?php
class myIterator implements Iterator {
    private $position = 0;
    private $array = array(
        "firstelement",
        "secondelement",
        "lastelement",
    );
```

```
    public function __construct() {
        $this->position = 0;
    }

    function rewind() {
        var_dump(__METHOD__);
        $this->position = 0;
    }

    function current() {
        var_dump(__METHOD__);
        return $this->array[$this->position];
    }

    function key() {
        var_dump(__METHOD__);
        return $this->position;
    }

    function next() {
        var_dump(__METHOD__);
        ++$this->position;
    }

    function valid() {
        var_dump(__METHOD__);
        return isset($this->array[$this->position]);
    }
}

$it = new myIterator;

foreach($it as $key => $value) {
    var_dump($key, $value);
    echo "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
string(18) "myIterator::rewind"
string(17) "myIterator::valid"
string(19) "myIterator::current"
string(15) "myIterator::key"
int(0)
string(12) "firstelement"

string(16) "myIterator::next"
string(17) "myIterator::valid"
string(19) "myIterator::current"
string(15) "myIterator::key"
int(1)
string(13) "secondelement"
```

```
string(16) "myIterator::next"
string(17) "myIterator::valid"
string(19) "myIterator::current"
string(15) "myIterator::key"
int(2)
string(11) "lastelement"

string(16) "myIterator::next"
string(17) "myIterator::valid"
```

Содержание ¶

User Contributed Notes 13 notes

Интерфейс IteratorAggregate ¶

(PHP 5 >= 5.0.0, PHP 7)

Введение ¶

Интерфейс для создания внешнего итератора.

Обзор интерфейсов ¶

```
IteratorAggregate extends Traversable {
/* Методы */
abstract public Traversable getIterator ( void )
}
```

Пример #1 Основы использования

```
<?php
class myData implements IteratorAggregate {
    public $property1 = "Public property one";
    public $property2 = "Public property two";
    public $property3 = "Public property three";

    public function __construct() {
        $this->property4 = "last property";
    }

    public function getIterator() {
        return new ArrayIterator($this);
    }
}

$obj = new myData;
```

```php
foreach($obj as $key => $value) {
    var_dump($key, $value);
    echo "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
string(9) "property1"
string(19) "Public property one"

string(9) "property2"
string(19) "Public property two"

string(9) "property3"
string(21) "Public property three"

string(9) "property4"
string(13) "last property"
```

Содержание ¶

User Contributed Notes 3 notes

Интерфейс ArrayAccess ¶

(PHP 5 >= 5.0.0, PHP 7)

Введение ¶

Интерфейс обеспечивает доступ к объектам как к массиву.

Обзор интерфейсов ¶

```php
ArrayAccess {
/* Методы */
abstract public boolean offsetExists ( mixed $offset )
abstract public mixed offsetGet ( mixed $offset )
abstract public void offsetSet ( mixed $offset , mixed $value )
abstract public void offsetUnset ( mixed $offset )
}
```

Пример #1 Основы использования

```php
<?php
class obj implements ArrayAccess {
    private $container = array();

    public function __construct() {
        $this->container = array(
            "one"   => 1,
            "two"   => 2,
```

```
            "three" => 3,
        );
    }

    public function offsetSet($offset, $value) {
        if (is_null($offset)) {
            $this->container[] = $value;
        } else {
            $this->container[$offset] = $value;
        }
    }

    public function offsetExists($offset) {
        return isset($this->container[$offset]);
    }

    public function offsetUnset($offset) {
        unset($this->container[$offset]);
    }

    public function offsetGet($offset) {
        return isset($this->container[$offset]) ? $this-
>container[$offset] : null;
    }
}

$obj = new obj;

var_dump(isset($obj["two"]));
var_dump($obj["two"]);
unset($obj["two"]);
var_dump(isset($obj["two"]));
$obj["two"] = "A value";
var_dump($obj["two"]);
$obj[] = 'Append 1';
$obj[] = 'Append 2';
$obj[] = 'Append 3';
print_r($obj);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
bool(true)
int(2)
bool(false)
string(7) "A value"
obj Object
(
    [container:obj:private] => Array
        (
            [one] => 1
            [three] => 3
            [two] => A value
```

```
            [0] => Append 1
            [1] => Append 2
            [2] => Append 3
        )

)
```

Содержание ¶

User Contributed Notes 11 notes

ArrayAccess::offsetExists

(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetExists — Определяет, существует ли заданное смещение (ключ)

Описание ¶

```
abstract public boolean ArrayAccess::offsetExists ( mixed $offset )
```

Определяет, существует или нет данное смещение (ключ).

Данный метод исполняется, когда используется функция isset() или функция empty() для объекта, реализующего интерфейсArrayAccess.

Замечание:
Когда используется функция empty(), метод ArrayAccess::offsetGet() вызывается и результат проверяется только в случае, если методArrayAccess::offsetExists() возвращает **TRUE**.

Список параметров ¶

offset

Смещение (ключ) для проверки.

Возвращаемые значения ¶

Возвращает **TRUE** в случае успешного завершения или **FALSE** в случае возникновения ошибки.

Замечание:
Возвращаемое значение будет приведено к логическому типу, если возвращаемое значение не является логическим.

Примеры ¶

Пример #1 Пример использования ArrayAccess::offsetExists()

```
<?php
class obj implements arrayaccess {
    public function offsetSet($offset, $value) {
        var_dump(__METHOD__);
    }
    public function offsetExists($var) {
        var_dump(__METHOD__);
        if ($var == "foobar") {
            return true;
        }
        return false;
    }
    public function offsetUnset($var) {
        var_dump(__METHOD__);
    }
    public function offsetGet($var) {
        var_dump(__METHOD__);
        return "value";
    }
}

$obj = new obj;

echo "Выполняется obj::offsetExists()\n";
var_dump(isset($obj["foobar"]));

echo "\nВыполняется obj::offsetExists() и obj::offsetGet()\n";
var_dump(empty($obj["foobar"]));

echo "\nВыполняется obj::offsetExists(), но *не* obj:offsetGet()
 поскольку нечего возвращать\n";
var_dump(empty($obj["foobaz"]));
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Выполняется obj::offsetExists()
string(17) "obj::offsetExists"
bool(true)

Выполняется obj::offsetExists() и obj::offsetGet()
string(17) "obj::offsetExists"
string(14) "obj::offsetGet"
bool(false)

Выполняется obj::offsetExists(), но *не* obj:offsetGet() по-
скольку нечего возвращать
string(17) "obj::offsetExists"
bool(true)
```

User Contributed Notes *1 note*

ArrayAccess::offsetGet
(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetGet — Возвращает заданное смещение (ключ)

Описание ¶
```
abstract public mixed ArrayAccess::offsetGet ( mixed $offset )
```

Возвращает заданное смещение (ключ).

Данный метод исполняется, когда проверяется смещение (ключ) на пустоту с помощью функции empty().

Список параметров ¶
offset

Смещение (ключ) для возврата.

Примечания ¶
Замечание:
Начиная с PHP 5.3.4, смягчена проверка для прототипа метода, и возможна реализация данного метода с возвращением по ссылке. Это делает возможным косвенную модификацию для величин массива перегруженного объекта ArrayAccess.

Явная модификация - это такая модификация, при которой полностью заменяется значение величины массива, как в случае*$obj[6] = 7*. С другой стороны, при косвенной модификации заменяется только часть величины, или происходит попытка присвоения величины по ссылке другой переменной, как в случае *$obj[6][7] = 7* или *$var =& $obj[6]*. Увеличение с использованием оператора ++ и уменьшение с использованием оператора -- также реализуются с помощью способа, который требует косвенную модификацию.

В то время как, явная модификация запускает вызов метода ArrayAccess::offsetSet(), косвенная модификация запускает вызов метода ArrayAccess::offsetGet(). В таком случае, реализация метода ArrayAccess::offsetGet() должна быть в состоянии возвращать результат по ссылке, в противном случае будет поднято сообщение об ошибке уровня **E_NOTICE**.

Возвращаемые значения ¶

Может возвращать значение любого типа.

Смотрите также ¶

- ArrayAccess::offsetExists() - Определяет, существует ли заданное смещение (ключ)

ArrayAccess::offsetSet
(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetSet — Устанавливает заданное смещение (ключ)

Описание ¶

```
abstract public void ArrayAccess::offsetSet ( mixed $offset , mixed $value )
```

Присваивает значение указанному смещению (ключу).

Список параметров ¶

offset

Смещение (ключ), которому будет присваиваться значение.

value

Значение для присвоения.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

Примечания ¶

Замечание:

Параметр **offset** будет установлен в **NULL** если иное значение недоступно, как показано в следующем примере.

```
<?php
$arrayaccess[] = "first value";
$arrayaccess[] = "second value";
print_r($arrayaccess);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => first value
    [1] => second value
)
```

Замечание:

Данный метод не вызывается при присвоениях по ссылки и других косвенных изменений величин массива перегруженного объекта ArrayAccess (косвенные в том смысле, что они произведены не прямой заменой величины, а путем изменения часть элемента или свойства элемента массива, или присвоением элемента массива по ссылке другой переменой). Вместо этого, вызывается метод ArrayAccess::offsetGet(). Данная операция будет успешна только в том случае, если метод возвращает по ссылке, что доступно только начиная с PHP 5.3.4.

ArrayAccess::offsetUnset

(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetUnset — Удаляет смещение

Описание ¶

```
abstract public void ArrayAccess::offsetUnset ( mixed $offset )
```

Удаляет смещение.

Замечание:
Этот метод *не* будет вызван при приведении типа к (unset)

Список параметров ¶
offset

Смещение для удаления.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

Интерфейс ArrayAccess ¶
(PHP 5 >= 5.0.0, PHP 7)

Введение ¶

Интерфейс обеспечивает доступ к объектам как к массиву.

Обзор интерфейсов ¶
```
ArrayAccess {
/* Методы */
abstract public boolean offsetExists ( mixed $offset )
abstract public mixed offsetGet ( mixed $offset )
abstract public void offsetSet ( mixed $offset , mixed $value )
abstract public void offsetUnset ( mixed $offset )
}
```

Пример #1 Основы использования

```
<?php
class obj implements ArrayAccess {
    private $container = array();

    public function __construct() {
        $this->container = array(
            "one"   => 1,
            "two"   => 2,
            "three" => 3,
        );
    }

    public function offsetSet($offset, $value) {
        if (is_null($offset)) {
            $this->container[] = $value;
        } else {
            $this->container[$offset] = $value;
        }
    }

    public function offsetExists($offset) {
```

```
        return isset($this->container[$offset]);
    }

    public function offsetUnset($offset) {
        unset($this->container[$offset]);
    }

    public function offsetGet($offset) {
        return isset($this->container[$offset]) ? $this-
>container[$offset] : null;
    }
}

$obj = new obj;

var_dump(isset($obj["two"]));
var_dump($obj["two"]);
unset($obj["two"]);
var_dump(isset($obj["two"]));
$obj["two"] = "A value";
var_dump($obj["two"]);
$obj[] = 'Append 1';
$obj[] = 'Append 2';
$obj[] = 'Append 3';
print_r($obj);
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
bool(true)
int(2)
bool(false)
string(7) "A value"
obj Object
(
    [container:obj:private] => Array
        (
            [one] => 1
            [three] => 3
            [two] => A value
            [0] => Append 1
            [1] => Append 2
            [2] => Append 3
        )

)
```

Содержание ¶

- ArrayAccess::offsetExists — Определяет, существует ли заданное смещение (ключ)
- ArrayAccess::offsetGet — Возвращает заданное смещение (ключ)
- ArrayAccess::offsetSet — Устанавливает заданное смещение (ключ)

- ArrayAccess::offsetUnset — Удаляет смещение

User Contributed Notes *11 notes*

ArrayAccess::offsetExists
(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetExists — Определяет, существует ли заданное смещение (ключ)

Описание ¶
```
abstract public boolean ArrayAccess::offsetExists ( mixed $offset )
```

Определяет, существует или нет данное смещение (ключ).

Данный метод исполняется, когда используется функция isset() или функция empty() для объекта, реализующего интерфейсArrayAccess.

Замечание:
Когда используется функция empty(), метод ArrayAccess::offsetGet() вызывается и результат проверяется только в случае, если методArrayAccess::offsetExists() возвращает **TRUE**.

Список параметров ¶
offset

Смещение (ключ) для проверки.

Возвращаемые значения ¶

Возвращает **TRUE** в случае успешного завершения или **FALSE** в случае возникновения ошибки.

Замечание:
Возвращаемое значение будет приведено к логическому типу, если возвращаемое значение не является логическим.

Примеры ¶

Пример #1 Пример использования ArrayAccess::offsetExists()

```
<?php
class obj implements arrayaccess {
    public function offsetSet($offset, $value) {
        var_dump(__METHOD__);
    }
    public function offsetExists($var) {
        var_dump(__METHOD__);
        if ($var == "foobar") {
            return true;
        }
        return false;
```

```
    }
    public function offsetUnset($var) {
        var_dump(__METHOD__);
    }
    public function offsetGet($var) {
        var_dump(__METHOD__);
        return "value";
    }
}

$obj = new obj;

echo "Выполняется obj::offsetExists()\n";
var_dump(isset($obj["foobar"]));

echo "\nВыполняется obj::offsetExists() и obj::offsetGet()\n";
var_dump(empty($obj["foobar"]));

echo "\nВыполняется obj::offsetExists(), но *не* obj:offsetGet()
 поскольку нечего возвращать\n";
var_dump(empty($obj["foobaz"]));
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
Выполняется obj::offsetExists()
string(17) "obj::offsetExists"
bool(true)

Выполняется obj::offsetExists() и obj::offsetGet()
string(17) "obj::offsetExists"
string(14) "obj::offsetGet"
bool(false)

Выполняется obj::offsetExists(), но *не* obj:offsetGet() по-
скольку нечего возвращать
string(17) "obj::offsetExists"
bool(true)
```

User Contributed Notes *1 note*

ArrayAccess::offsetGet

(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetGet — Возвращает заданное смещение (ключ)

Описание ¶

```
abstract public mixed ArrayAccess::offsetGet ( mixed $offset )
```

Возвращает заданное смещение (ключ).

Данный метод исполняется, когда проверяется смещение (ключ) на пустоту с помощью функции empty().

Список параметров ¶

offset

Смещение (ключ) для возврата.

Примечания ¶

Замечание:
Начиная с PHP 5.3.4, смягчена проверка для прототипа метода, и возможна реализация данного метода с возвращением по ссылке. Это делает возможным косвенную модификацию для величин массива перегруженного объекта ArrayAccess.

Явная модификация - это такая модификация, при которой полностью заменяется значение величины массива, как в случае*$obj[6] = 7*. С другой стороны, при косвенной модификации заменяется только часть величины, или происходит попытка присвоения величины по ссылке другой переменной, как в случае *$obj[6][7] = 7* или *$var =& $obj[6]*. Увеличение с использованием оператора ++ и уменьшение с использованием оператора -- также реализуются с помощью способа, который требует косвенную модификацию.

В то время как, явная модификация запускает вызов метода ArrayAccess::offsetSet(), косвенная модификация запускает вызов метода ArrayAccess::offsetGet(). В таком случае, реализация метода ArrayAccess::offsetGet() должна быть в состоянии возвращать результат по ссылке, в противном случае будет поднято сообщение об ошибке уровня **E_NOTICE**.

Возвращаемые значения ¶

Может возвращать значение любого типа.

Смотрите также ¶

- ArrayAccess::offsetExists() - Определяет, существует ли заданное смещение (ключ)

ArrayAccess::offsetSet

(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetSet — Устанавливает заданное смещение (ключ)

Описание ¶

```
abstract public void ArrayAccess::offsetSet ( mixed $offset , mixed $value )
```

Присваивает значение указанному смещению (ключу).

Список параметров ¶

offset

Смещение (ключ), которому будет присваиваться значение.

`value`

Значение для присвоения.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

Примечания ¶

Замечание:
Параметр **`offset`** будет установлен в **`NULL`** если иное значение недоступно, как показано в следующем примере.

```
<?php
$arrayaccess[] = "first value";
$arrayaccess[] = "second value";
print_r($arrayaccess);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => first value
    [1] => second value
)
```

Замечание:
Данный метод не вызывается при присвоениях по ссылки и других косвенных изменений величин массива перегруженного объекта ArrayAccess (косвенные в том смысле, что они произведены не прямой заменой величины, а путем изменения часть элемента или свойства элемента массива, или присвоением элемента массива по ссылке другой переменой). Вместо этого, вызывается метод ArrayAccess::offsetGet(). Данная операция будет успешна только в том случае, если метод возвращает по ссылке, что доступно только начиная с PHP 5.3.4.

- Встроенные интерфейсы и классы

- ArrayAccess

ArrayAccess::offsetUnset
(PHP 5 >= 5.0.0, PHP 7)

ArrayAccess::offsetUnset — Удаляет смещение

Описание ¶
```
abstract public void ArrayAccess::offsetUnset ( mixed $offset )
```

Удаляет смещение.

Замечание:
Этот метод *не* будет вызван при приведении типа к (unset)

Список параметров ¶
offset

Смещение для удаления.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

Интерфейс Serializable ¶
(PHP 5 >= 5.1.0, PHP 7)

Введение ¶

Интерфейс для индивидуальной сериализации.

Классы, которые реализуют этот интерфейс не поддерживают больше __sleep() и __wakeup(). Метод serialize вызывается всякий раз, когда необходима сериализация экземпляру класса. Этот метод не вызывает __destruct() и не имеет никаких побочных действий кроме тех, которые запрограммированы внутри него. Когда данные десериализованы, класс известен и соответствующий метод unserialize() вызывается как конструктор вместо вызова __construct(). Если вам необходимо вызвать стандартный конструктор, вы можете это сделать в этом методе.

Обзор интерфейсов ¶
```
Serializable {
/* Методы */
abstract public string serialize ( void )
abstract public void unserialize ( string $serialized )
}
```

Пример #1 Основы использования

```php
<?php
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser);

$newobj = unserialize($ser);

var_dump($newobj->getData());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
string(38) "C:3:"obj":23:{s:15:"My private data";}"
string(15) "My private data"
```

Содержание ¶

User Contributed Notes 5 notes

Serializable::serialize

(PHP 5 >= 5.1.0, PHP 7)

Serializable::serialize — Представляет объект в виде строки

Описание ¶

```
abstract public string Serializable::serialize ( void )
```

Возвращает строковое представление объекта.

Замечание:
Этот метод действует как деструктор объекта. Метод __destruct() *не* вызывается после этого метода.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Возвращает строковое представление объекта или **NULL**

Ошибки ¶

Вызывает Exception при возвращении типов, отличных от string или **NULL**

Смотрите также ¶

- __sleep()

User Contributed Notes *1 note*

Serializable::unserialize
(PHP 5 >= 5.1.0, PHP 7)

Serializable::unserialize — Создает объект

Описание ¶

```
abstract public void Serializable::unserialize ( string $serialized )
```

Вызывается во время десериализации объекта.

Замечание:
Этот метод действует как конструктор объекта. Метод __construct() *не* вызывается после этого метода.

Список параметров ¶

serialized

Строковое представление объекта.

Возвращаемые значения ¶

Возвращаемое значение этого метода игнорируется.

Смотрите также ¶

- __wakeup()

Интерфейс Serializable ¶
(PHP 5 >= 5.1.0, PHP 7)

Введение ¶

Интерфейс для индивидуальной сериализации.

Классы, которые реализуют этот интерфейс не поддерживают больше __sleep() и __wakeup(). Метод serialize вызывается всякий раз, когда необходима сериализация экземпляру класса. Этот метод не вызывает __destruct() и не имеет никаких побочных действий кроме тех, которые запрограммированы внутри него. Когда данные десериализованы, класс известен и соответствующий метод unserialize() вызывается как конструктор вместо вызова __construct(). Если вам необходимо вызвать стандартный конструктор, вы можете это сделать в этом методе.

Обзор интерфейсов ¶

```
Serializable {
/* Методы */
abstract public string serialize ( void )
abstract public void unserialize ( string $serialized )
}
```

Пример #1 Основы использования

```
<?php
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser);

$newobj = unserialize($ser);

var_dump($newobj->getData());
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
string(38) "C:3:"obj":23:{s:15:"My private data";}"
string(15) "My private data"
```

Содержание ¶

- Serializable::serialize — Представляет объект в виде строки

- Serializable::unserialize — Создает объект

User Contributed Notes *5 notes*

Serializable::serialize
(PHP 5 >= 5.1.0, PHP 7)

Serializable::serialize — Представляет объект в виде строки

Описание ¶
```
abstract public string Serializable::serialize ( void )
```

Возвращает строковое представление объекта.

Замечание:
Этот метод действует как деструктор объекта. Метод __destruct() *не* вызывается после этого метода.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Возвращает строковое представление объекта или **NULL**

Ошибки ¶

Вызывает Exception при возвращении типов, отличных от string или **NULL**

Смотрите также ¶

- __sleep()

User Contributed Notes *1 note*

Serializable::unserialize
(PHP 5 >= 5.1.0, PHP 7)

Serializable::unserialize — Создает объект

Описание ¶
```
ab-
stract public void Serializable::unserialize ( string $serialize
d )
```

Вызывается во время десериализации объекта.

Замечание:
Этот метод действует как конструктор объекта. Метод __construct() *не* вызывается после этого метода.

Список параметров ¶

serialized

Строковое представление объекта.

Возвращаемые значения ¶

Возвращаемое значение этого метода игнорируется.

Смотрите также ¶

- __wakeup()

Класс Closure ¶

(PHP 5 >= 5.3.0, PHP 7)

Введение ¶

Класс используемый для создания анонимных функций.

Анонимные функции, появившиеся в PHP 5.3, являются объектами данного класса. Ранее это считалось только деталью реализации, но начиная с PHP 5.4, этот класс получил методы, позволяющие контролировать анонимные функции после их создания.

Кроме методов, описанных здесь, этот класс также имеет метод *__invoke*. Данный метод необходим только для совместимости с другими классами, в которых реализован магический вызов, так как этот метод не используется при вызове функции.

Обзор классов ¶

```
Closure {
/* Методы */
private __construct ( void )
pub-
lic static Closure bind ( Closure $closure , object $newthis [,
mixed $newscope = "static" ] )
public Closure bindTo ( object $newthis [, mixed $newscope = "stat-
ic" ] )
}
```

Содержание ¶

- Closure::__construct — Конструктор запрещающий создавать новые объекты
- Closure::bind — Дублирует замыкание с указанием связанного объекта и области видимости класса
- Closure::bindTo — Дублирует замыкание с указанием связанного объекта и области видимости класса

User Contributed Notes 3 notes

Closure::__construct

(PHP 5 >= 5.3.0, PHP 7)

Closure::__construct — Конструктор запрещающий создавать новые объекты

Описание ¶

```
private Closure::__construct ( void )
```

Этот метод существует только, чтобы запретить создание новых объектов класса Closure. Объекты этого класса создаются способом, описанным в разделе анонимные функции.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Этот метод не возвращает никаких значений. Он просто выбрасывает ошибку типа **E_RECOVERABLE_ERROR**.

Смотрите также ¶

- Анонимные функции

Closure::bind

(PHP 5 >= 5.4.0, PHP 7)

Closure::bind — Дублирует замыкание с указанием связанного объекта и области видимости класса

Описание ¶

```
pub-
lic static Closure Closure::bind ( Closure $closure , object $ne
wthis [, mixed $newscope = "static" ] )
```

Этот метод является статическим вариантом Closure::bindTo(). Смотрите документацию к указанному методу для подробной информации.

Список параметров ¶

closure

Анонимная функция для привязывания к объекту

newthis

Объект, к которому будет привязана переданная функция, или **NULL** для отсоединения функции от ее текущего объекта.

newscope

Область видимости класса, с которой ассоциируется замыкание, или 'static' для сохранения текущей области видимости. Если передан объект, то будет использован его класс. Этот параметр определяет видимость protected (защищенных) и private (закрытых) методов привязанного объекта.

Возвращаемые значения ¶

Возвращает новый объект Closure или **FALSE** в случае возникновения ошибки

Примеры ¶

Пример #1 Пример Closure::bind()

```
<?php
class A {
    private static $sfoo = 1;
    private $ifoo = 2;
}
$cl1 = static function() {
    return A::$sfoo;
};
$cl2 = function() {
    return $this->ifoo;
};

$bcl1 = Closure::bind($cl1, null, 'A');
$bcl2 = Closure::bind($cl2, new A(), 'A');
echo $bcl1(), "\n";
echo $bcl2(), "\n";
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
1
2
```

Смотрите также ¶

- Анонимные функции
- Closure::bindTo() - Дублирует замыкание с указанием связанного объекта и области видимости класса

User Contributed Notes 2 notes

Closure::bindTo

(PHP 5 >= 5.4.0, PHP 7)

Closure::bindTo — Дублирует замыкание с указанием связанного объекта и области видимости класса

Описание ¶

```
public Closure Closure::bindTo ( object $newthis [, mixed $newscope = "static" ] )
```

Создает и возвращает новую анонимную функцию с тем же телом функции и связанными переменными, но с другим связанным объектом или новой областью видимости класса.

"Привязанный объект" определяет значение *$this*, которое будет доступно в теле функции, а "область видимости класса" представляет собой класс, который определяет к каким protected (защищенным) и private (закрытым) элементам этого объекта будет иметь доступ анонимная функция. Если точнее, то это те элементы, как если бы анонимная функция была бы методом класса, переданного в параметре **newscope**.

Статические замыкания не могут иметь привязанный объект (значение параметра **newthis** должно быть равно **NULL**), но эта функция может все равно использоваться для изменения его области видимости класса.

Данный метод гарантирует, что у нестатического замыкания с привязанным объектом будет задана область видимости и наоборот. Для выполнения этого условия применяются следующие правила: Для нестатического замыкания, с указанной областью видимости и с**NULL** вместо объекта, будет создано статическое замыкание. Для нестатического замыкания с незаданной областью видимости, но с указанием объекта, создается замыкание с неуказанной областью видимости.

Замечание:
Если вам необходимо только дублировать анонимную функцию, то вы можете вместо данного метода использовать клонирование.

Список параметров ¶

newthis

Объект, к которому будет привязана переданная функция, или **NULL** для отсоединения функции от ее текущего объекта.

newscope

Область видимости класса, с которой ассоциируется замыкание, или 'static' для сохранения текущей области видимости. Если передан объект, то будет использован его класс. Этот параметр определяет видимость protected (защищенных) и private (закрытых) методов привязанного объекта.

Возвращаемые значения ¶

Возвращает новый объект Closure или **FALSE** в случае возникновения ошибки

Примеры ¶

Пример #1 Пример Closure::bindTo()

```
<?php

class A {
    function __construct($val) {
        $this->val = $val;
    }
    function getClosure() {
        //Возвращает замыкание, связанное с текущими объектом и
областью видимости
```

```
        return function() { return $this->val; };
    }
}

$ob1 = new A(1);
$ob2 = new A(2);

$cl = $ob1->getClosure();
echo $cl(), "\n";
$cl = $cl->bindTo($ob2);
echo $cl(), "\n";
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
1
2
```

Смотрите также ¶

- Анонимные функции
- Closure::bind() - Дублирует замыкание с указанием связанного объекта и области видимости класса

User Contributed Notes *6 notes*

The Generator class ¶

(PHP 5 >= 5.5.0, PHP 7)

Введение ¶

Generator objects are returned from generators.

Предостережение
Generator objects cannot be instantiated via new.

Обзор классов ¶

```
Generator implements Iterator {
/* Методы */
public mixed current ( void )
public mixed getReturn ( void )
public mixed key ( void )
public void next ( void )
public void rewind ( void )
public mixed send ( mixed $value )
public mixed throw ( Exception $exception )
public bool valid ( void )
public void __wakeup ( void )
}
```

Содержание ¶

- Generator::current — Get the yielded value
- Generator::getReturn — Get the return value of a generator

- Generator::key — Get the yielded key
- Generator::next — Resume execution of the generator
- Generator::rewind — Rewind the iterator
- Generator::send — Send a value to the generator
- Generator::throw — Throw an exception into the generator
- Generator::valid — Check if the iterator has been closed
- Generator::__wakeup — Serialize callback

User Contributed Notes 1 note

Pistachio ¶

7 months ago

Unlike return, yield can be used anywhere within a function so logic can flow more naturally. Take for example the following Fibonacci generator:

```
<?php
function fib($n)
{
    $cur = 1;
    $prev = 0;
    for ($i = 0; $i < $n; $i++) {
        yield $cur;

        $temp = $cur;
        $cur = $prev + $cur;
        $prev = $temp;
    }
}

$fibs = fib(9);
foreach ($fibs as $fib) {
    echo "" . $fib;
}

// prints: 1 1 2 3 5 8 13 21 34
```

Generator::current

(PHP 5 >= 5.5.0, PHP 7)

Generator::current — Get the yielded value

Описание ¶

```
public mixed Generator::current ( void )
```

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Returns the yielded value.

Generator::getReturn
(PHP 7)

Generator::getReturn — Get the return value of a generator

Описание ¶
```
public mixed Generator::getReturn ( void )
```

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Returns the generator's return value once it has finished executing.

Примеры ¶

Пример #1 Generator::getReturn() example

```
<?php

$gen = (function() {
    yield 1;
    yield 2;

    return 3;
})();

foreach ($gen as $val) {
    echo $val, PHP_EOL;
}

echo $gen->getReturn(), PHP_EOL;
```

Результат выполнения данного примера:

```
1
2
3
```

User Contributed Notes *1 note*

Generator::key
(PHP 5 >= 5.5.0, PHP 7)

Generator::key — Get the yielded key

Описание ¶
```
public mixed Generator::key ( void )
```

Gets the key of the yielded value.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Returns the yielded key.

Примеры ¶

Пример #1 Generator::key() example

```
<?php

function Gen()
{
    yield 'key' => 'value';
}

$gen = Gen();

echo "{$gen->key()} => {$gen->current()}";
```

Результат выполнения данного примера:

```
key => value
```

Generator::next
(PHP 5 >= 5.5.0, PHP 7)

Generator::next — Resume execution of the generator

Описание ¶

```
public void Generator::next ( void )
```

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

User Contributed Notes *1 note*

Generator::rewind
(PHP 5 >= 5.5.0, PHP 7)

Generator::rewind — Rewind the iterator

Описание ¶

```
public void Generator::rewind ( void )
```

If iteration has already begun, this will throw an exception.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

User Contributed Notes 2 notes

Generator::send
(PHP 5 >= 5.5.0, PHP 7)

Generator::send — Send a value to the generator

Описание ¶
```
public mixed Generator::send ( mixed $value )
```

Sends the given value to the generator as the result of the current yield expression and resumes execution of the generator.

If the generator is not at a yield expression when this method is called, it will first be let to advance to the first yield expression before sending the value. As such it is not necessary to "prime" PHP generators with a Generator::next() call (like it is done in Python).

Список параметров ¶
value

> Value to send into the generator. This value will be the return value of the yield expression the generator is currently at.

Возвращаемые значения ¶

Returns the yielded value.

Примеры ¶

Пример #1 Using Generator::send() to inject values

```
<?php
function printer() {
    while (true) {
        $string = yield;
        echo $string;
    }
}

$printer = printer();
$printer->send('Hello world!');
```

```
$printer->send('Bye world!');
?>
```

Результат выполнения данного примера:

```
Hello world!
Bye world!
```

User Contributed Notes 3 notes

sfroelich01 at sp dot gm dot ail dot am dot com ¶

3 years ago

Reading the example, it is a bit difficult to understand what exactly to do with this. The example below is a simple example of what you can do this.

```
<?php
function nums() {
    for ($i = 0; $i < 5; ++$i) {
                //get a value from the caller
        $cmd = (yield $i);

        if($cmd == 'stop')
            return;//exit the function
        }
}

$gen = nums();
foreach($gen as $v)
{
    if($v == 3)//we are satisfied
        $gen->send('stop');

    echo "{$v}\n";
}

//Output
0
1
2
3
?>
```

Generator::throw

(PHP 5 >= 5.5.0, PHP 7)

Generator::throw — Throw an exception into the generator

Описание ¶

```
public mixed Generator::throw ( Exception $exception )
```

Throws an exception into the generator and resumes execution of the generator. The behavior will be the same as if the current yieldexpression was replaced with a *throw $exception* statement.

If the generator is already closed when this method is invoked, the exception will be thrown in the caller's context instead.

Список параметров ¶

exception

Exception to throw into the generator.

Возвращаемые значения ¶

Returns the yielded value.

Примеры ¶

Пример #1 Throwing an exception into a generator

```
<?php
function gen() {
    echo "Foo\n";
    try {
        yield;
    } catch (Exception $e) {
        echo "Exception: {$e->getMessage()}\n";
    }
    echo "Bar\n";
}

$gen = gen();
$gen->rewind();
$gen->throw(new Exception('Test'));
?>
```

Результат выполнения данного примера:

```
Foo
Exception: Test
Bar
```

User Contributed Notes *1 note*

Generator::valid

(PHP 5 >= 5.5.0, PHP 7)

Generator::valid — Check if the iterator has been closed

Описание ¶

```
public bool Generator::valid ( void )
```

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Returns **FALSE** if the iterator has been closed. Otherwise returns **TRUE**.

Generator::__wakeup
(PHP 5 >= 5.5.0, PHP 7)

Generator::__wakeup — Serialize callback

Описание ¶
```
public void Generator::__wakeup ( void )
```

Throws an exception as generators can't be serialized.

Список параметров ¶

У этой функции нет параметров.

Возвращаемые значения ¶

Эта функция не возвращает значения после выполнения.

Контекстные опции и параметры ¶

PHP предлагает различные контекстные опции и параметры, которые могут быть использованы со всеми файловыми системами и обработчиками потоков (stream wrappers). Контекст создается с помощью функции stream_context_create(). Опции устанавливаются путем вызова stream_context_set_option(), а параметры -- путем вызова stream_context_set_params().

Содержание ¶

Контекстные опции сокета

Контекстные опции сокета — Список контекстных опций сокета

Описание ¶

Контекстные опции доступны для всех оберток, которые работают через сокеты, такие как *tcp*, *http* и *ftp*.

Опции ¶

bindto

Используется для указания IP адреса (IPv4 или IPv6) и\или номера порта, которые PHP будет использовать для подключения к сети. Синтаксис выглядит следующим образом: *ip:port* для адреса IPv4, и *[ip]:port* для адреса IPv6. Установка IP и\или порта в *0*позволит системе самой выбрать нужный IP и\или порт.

Замечание:
Так как во время обычной работы FTP создает 2 соединения с сокетами, номер порта не может быть задан с помощью данной опции.

backlog

Используется для ограничения исходящих соединений, в очереди соединений сокета.

Замечание:
Используется только для stream_socket_server().

Список изменений ¶

Версия	Описание
5.3.3	Добавлено *backlog*.
5.1.0	Добавлено *bindto*.

Примеры ¶

Пример #1 Пример использования **bindto**

```
<?php
// Соединение с сетью, используя IP '192.168.0.100'
$opts = array(
    'socket' => array(
        'bindto' => '192.168.0.100:0',
    ),
);


// Соединение с сетью, используя IP '192.168.0.100' и порт '7000
'
$opts = array(
    'socket' => array(
        'bindto' => '192.168.0.100:7000',
    ),
);


// Соединение с сетью, используя IPv6 адрес '2001:db8::1'
```

```php
// и порт '7000'
$opts = array(
    'socket' => array(
        'bindto' => '[2001:db8::1]:7000',
    ),
);


// Соединение с сетью, используя порт '7000'
$opts = array(
    'socket' => array(
        'bindto' => '0:7000',
    ),
);


// Создаем контекст...
$context = stream_context_create($opts);

// ...и используем его для получения данных
echo file_get_contents('http://www.example.com', false, $context
);

?>
```

User Contributed Notes 1 note

Опции контекста HTTP

Опции контекста HTTP — Список опций контекста HTTP

Описание ¶

Опции контекста для транспортных протоколов *http://* и *https://*.

Опции ¶

`method` string

> **GET**, **POST**, или любой другой метод HTTP, поддерживаемый удаленным сервером.
>
> По умолчанию - **GET**.

`header` string

> Дополнительные заголовки для отправки вместе с запросом. Значения в этой опции будут переопределять другие значения (такие как *User-agent:*, *Host:*, и *Authentication:*).

`user_agent` string

Значение для отправки вместе с заголовком *User-Agent:*. Это значение будет использоваться, если заголовок user-agent *не* был указан в опции контекста *header* выше.

По умолчанию используется значение директивы user_agent из файла `php.ini`.

`content` string

Дополнительные данные для отправки после заголовков. Обычно используется с запросами POST и PUT.

`proxy` string

URI, указывающий адрес прокси-сервера. (Например, *tcp://proxy.example.com:5100*).

`request_fulluri` boolean

Когда установлено в **TRUE**, весь URI будет использован при построении запроса. (Например, *GET http://www.example.com/path/to/file.html HTTP/1.0*). Хотя это нестандартный формат запроса, некоторые прокси-серверы требуют его.

По умолчанию значение **FALSE**.

`follow_location` integer

Следовать переадресациям заголовка *Location*. Для отключения установите в значение *0*.

По умолчанию *1*.

`max_redirects` integer

Максимальное количество переадресаций, которым можно следовать. Значение *1* или меньше означает, что никаких переадресаций не будет произведено.

По умолчанию *20*.

`protocol_version` float

Версия протокола HTTP.

По умолчанию *1.0*.

Замечание:
В PHP до версии 5.3.0 не было реализовано декодирование фрагментированных передач. Если это значение установлено в*1.1*, то это ваша ответственность, чтобы оно соответствовало версии *1.1*.

timeout float

Тайм-аут на чтение в секундах, указанный с помощью типа float (например, *10.5*).

По умолчанию используется значение директивы default_socket_timeout из файла `php.ini`.

ignore_errors boolean

Извлечь содержимое даже в случае, если присутствует код статуса неуспешного завершения

По умолчанию **FALSE**.

Список изменений ¶

Версия	Описание
5.3.4	Добавлен параметр **follow_location**.
5.3.0	Параметр **protocol_version** поддерживает декодирование фрагментированной передачи, когда установлен в *1.1*.
5.2.10	Добавлен параметр **ignore_errors**.
5.2.10	Параметр **header** теперь может быть типа array с числовыми индексами.
5.2.1	Добавлен параметр **timeout**.
5.1.0	Добавлено проксирование протокола HTTPS через HTTP-прокси.
5.1.0	Добавлен параметр **max_redirects**.
5.1.0	Добавлен параметр **protocol_version**.

Примеры ¶

Пример #1 Извлечь страницу и отправить данные методом POST

```
<?php

$postdata = http_build_query(
    array(
        'var1' => 'некоторое содержимое',
        'var2' => 'doh'
    )
);

$opts = array('http' =>
    array(
        'method'  => 'POST',
        'header'  => 'Content-type: application/x-www-form-
urlencoded',
        'content' => $postdata
    )
);

$context = stream_context_create($opts);
```

```php
$result = file_get_contents('http://example.com/submit.php', fal
se, $context);

?>
```

Пример #2 Игнорировать переадресации, но извлечь заголовки и содержимое

```php
<?php

$url = "http://www.example.org/header.php";

$opts = array('http' =>
    array(
        'method' => 'GET',
        'max_redirects' => '0',
        'ignore_errors' => '1'
    )
);

$context = stream_context_create($opts);
$stream = fopen($url, 'r', false, $context);

// информация о заголовках, а также
// мета-данные о потоке
var_dump(stream_get_meta_data($stream));

// актуальная информация по ссылке $url
var_dump(stream_get_contents($stream));
fclose($stream);
?>
```

Примечания ¶

Замечание: Опции контекста низлежащего потока в сокете
Дополнительные опции контекста могут поддерживаться низлежащим транспортным протоколом. Для потоков *http://*, это относится к опциям контекста для транспортного протокола *tcp://*. Для потоков *https://*, это относится к опциям контекста для транспортного протокола *ssl://*.

Замечание: Строка статуса HTTP
Когда эта обертка потока следует по переадресации, *wrapper_data*, возвращаемый функцией stream_get_meta_data(), необязательно содержит строку статуса HTTP, которая на самом деле относится к содержанию данных по индексу *0*.

```
array (
  'wrapper_data' =>
  array (
    0 => 'HTTP/1.0 301 Moved Permantenly',
    1 => 'Cache-Control: no-cache',
    2 => 'Connection: close',
    3 => 'Location: http://example.com/foo.jpg',
    4 => 'HTTP/1.1 200 OK',
    ...
```

Первый запрос вернул код *301* (постоянное перенаправление), так что обертка потока автоматически последовала этому перенаправлению, чтобы получить ответ *200* (индекс = *4*).

Смотрите также ¶

- http://
- Контекстные опции сокета
- Опции контекста SSL

User Contributed Notes 6 notes

Параметры контекста FTP

Параметры контекста FTP — Список параметров контекста FTP

Описание ¶

Параметры контекста для транспортных протоколов *ftp://* и *ftps://*

Опции ¶

`overwrite` boolean

Разрешает перезаписывать существующие файлы на удаленном сервере. Работает только в режиме записи (upload).

По умолчанию **FALSE**.

`resume_pos` integer

Смещение в файле, с которого начинается передача. Работает только в режиме чтения (download).

По умолчанию *0* (Начало файла).

`proxy` string

FTP-запрос через прокси-сервер HTTP. Применяется только при операции чтения файла. Пример: *tcp://squid.example.com:8000*.

Список изменений ¶

Версия	Описание
5.1.0	Добавлен параметр **`proxy`**.
5.0.0	Добавлены параметры **`overwrite`** и **`resume_pos`**.

Примечания ¶

Замечание: Опции контекста низлежащего потока в сокете Дополнительные опции контекста могут поддерживаться низлежащим транспортным протоколом. Для потоков *ftp://*, это относится к опциям контекста для транспортного протокола *tcp://*. Для потоков *ftps://*, это относится к опциям контекста для транспортного протокола *ssl://*.

Смотрите также ¶

- ftp://
- Контекстные опции сокета

- Опции контекста SSL

User Contributed Notes 1 note

Опции контекста SSL

Опции контекста SSL — Список опций контекста SSL

Описание ¶

Опции контекста для протоколов *ssl://* и *tls://*

Опции ¶

`peer_name` string

Имя узла. Если его значение не задано, тогда имя подставляется основываясь на имени хоста, использованного при открытии потока.

`verify_peer` boolean

Требовать проверки используемого SSL-сертификата.

По умолчанию **TRUE**.

`verify_peer_name` boolean

Требовать проверки имени узла.

По умолчанию **TRUE**.

`allow_self_signed` boolean

Разрешить самоподписанные сертификаты. Требует **`verify_peer`**.

По умолчанию **FALSE**

`cafile` string

Расположение файла сертификата в локальной файловой системе, который следует использовать с опцией контекста*verify_peer* для проверки подлинности удалённого узла.

`capath` string

Если параметр *cafile* не определён или сертификат не найден, осуществляется поиск в директории, указанной в *capath*. Путь*capath* должен быть к корректной директории, содержащей сертификаты, имена которых являются хешем от поля subject, указанного в сертификате.

`local_cert` string

Путь к локальному сертификату в файловой системе. Это должен быть файл, закодированный PEM, который содержит ваш сертификат и приватный ключ. Он дополнительно может содержать публичный ключ эмитента.

passphrase string

Идентификационная фраза, с которой ваш файл *local_cert* был закодирован.

CN_match string

Имя хоста (Common Name), которое мы ожидаем. PHP будет производить ограниченное сопоставление, учитывающее символы подстановки (*,?). Если имя хоста не совпадает со строкой, соединение будет сброшено.

Замечание: Эта опция устарела. Начиная с PHP 5.6.0 используйте **peer_name**.

verify_depth integer

Прервать, если цепочка сертификата слишком длинная.

По умолчанию проверка отсутствует.

ciphers string

Устанавливает список доступных алгоритмов шифрования. Формат этой строки описан в разделе » шифры(1).

По умолчанию принимает значение *DEFAULT*.

capture_peer_cert boolean

Если установлено в **TRUE**, то будет создана опция контекста *peer_certificate*, содержащая сертификат удаленного узла.

capture_peer_cert_chain boolean

Если установлено в **TRUE**, то будет создана опция контекста *peer_certificate_chain*, содержащая цепочку сертификатов.

SNI_enabled boolean

Если установлено в **TRUE**, то будет включено указание имени сервера. Включение SNI позволяет использовать разные сертификаты на одном и том же IP-адресе.

SNI_server_name string

Если эта опция установлена, то это значение будет использоваться для указания имени сервера. Если это значение не установлено, то имя сервера определяется из имени хоста, использовавшегося при открытии потока.

Замечание: Эта опция устарела. Начиная с PHP 5.6.0 используйте **peer_name**.

disable_compression boolean

Отключает сжатие TLS, что помогает предотвратить атаки типа CRIME.

peer_fingerprint string | array

Прерваться, если дайджест сообщения не совпадает с указанным хэшом.

Если указана строка (string), то её длина определяет какой алгоритм хэширования будет использован: "md5" (32) или "sha1" (40).

Если указан массив (array), то ключи определяют алгоритм хэширования, а каждое соответствующее значение является требуемым хэшом.

Список изменений ¶

Версия	Описание
5.6.0	Добавлены **peer_fingerprint** и **verify_peer_name**. Значение по умолчанию для **verify_peer** изменено на **TRUE**.
5.4.13	Добавлен **disable_compression**. Требуется OpenSSL >= 1.0.0.
5.3.2	Добавлены параметры **SNI_enabled** и **SNI_server_name**.
5.0.0	Добавлены параметры **capture_peer_cert**, **capture_peer_chain** и **ciphers**.

Примечания ¶

Замечание: Так как *ssl://* - это низлежащий транспортный протокол для оберток *https://* и *ftps://*, то любые опции контекста, которые применяются к *ssl://* будут также применяться к *https://* и *ftps://*.

Замечание: Чтобы была доступна возможность указания имени сервера (SNI, Server Name Indication), PHP должен быть скомпилирован с OpenSSL 0.9.8j или более поздней. Используйте константу **OPENSSL_TLSEXT_SERVER_NAME** чтобы определить, поддерживается ли SNI.

Смотрите также ¶

- Контекстные опции сокета

User Contributed Notes 4 notes

- Руководство по PHP
- Справочник языка
- Контекстные опции и параметры

Change language:
Russian
Edit Report a Bug

Опции контекста CURL

Опции контекста CURL — Список опций контекста CURL

Описание ¶

Опции контекста CURL доступны в том случае, если расширение CURL скомпилировано, используя конфигурационную опцию --with-curlwrappers .

Опции ¶

`method` string

> **GET**, **POST**, или любой другой HTTP-метод, поддерживаемый удаленным сервером.
>
> По умолчанию - **GET**.

`header` string

> Дополнительные заголовки для отправки вместе с запросом. Значения в этой опции будут переопределять другие значения (такие как *User-agent:*, *Host:*, и *Authentication:*).

`user_agent` string

> Значение для отправки вместе с заголовком User-Agent.
>
> По умолчанию используется значение директивы user_agent из файла `php.ini`.

`content` string

> Дополнительные данные для отправки после заголовков. Эта опция не используется для запросов **GET** или **HEAD**.

`proxy` string

> URI, указывающий адрес прокси-сервера. (Например, *tcp://proxy.example.com:5100*).

`max_redirects` integer

> Максимальное количество переадресаций, которые можно сделать. Значение *1* или меньше означает, что никаких переадресаций не будет произведено.

По умолчанию *20*.

curl_verify_ssl_host boolean

Проверить хост.

По умолчанию **FALSE**

Замечание:
Эта опция доступна для двух оберток протоколов: http и ftp.

curl_verify_ssl_peer boolean

Требовать проверку используемого SSL-сертификата.

По умолчанию **FALSE**

Замечание:
Эта опция доступна для двух оберток протоколов: http и ftp.

Примеры ¶

Пример #1 Получает страницу и посылает POST-запрос

```
<?php

$postdata = http_build_query(
    array(
        'var1' => 'некоторое содержимое',
        'var2' => 'doh'
    )
);

$opts = array('http' =>
    array(
        'method'  => 'POST',
        'header'  => 'Content-type: application/x-www-form-
urlencoded',
        'content' => $postdata
    )
);

$context = stream_context_create($opts);

$result = file_get_contents('http://example.com/submit.php', fal
se, $context);

?>
```

Смотрите также ¶

- Контекстные опции сокета

Контекстные опции Phar

Контекстные опции Phar — Список контекстных опций Phar

Описание ¶

Контекстные опции для обёртки *phar://*.

Опции ¶

`compress` int

Одна из констант Phar сжатия.

`metadata` mixed

Phar метаданные. См. Phar::setMetadata().

Смотрите также ¶

- phar://
- потоки Phar

MongoDB context options

MongoDB context options — MongoDB context option listing

Описание ¶

Context options for *mongodb://* transports.

Опции ¶

`log_cmd_insert` callable

A callback function called when inserting a document, see log_cmd_insert().

`log_cmd_delete` callable

A callback function called when deleting a document, see log_cmd_delete().

`log_cmd_update` callable

A callback function called when updating a document, see log_cmd_update().

`log_write_batch` callable

A callback function called when executing a Write Batch, see log_write_batch().

`log_reply` callable

A callback function called when reading a reply from MongoDB, see log_reply().

log_getmore callable

A callback function called when retrieving more results from a MongoDB cursor, see log_getmore().

log_killcursor callable

A callback function called executing a killcursor opcode, see log_killcursor().

Список изменений ¶

Версия	Описание
pecl/mongo 1.5.0	Added Write API Context options

Смотрите также ¶

- Контекстные опции сокета
- Опции контекста SSL

User Contributed Notes

Параметры контекста

Параметры контекста — Список параметров контекста

Описание ¶

Данные параметры (*parameters*) могут быть заданы для контекста (*context*) с помощью функции stream_context_set_params().

Список параметров ¶

notification callable

Функция типа callable, вызываемая при наступлении события в потоке.

За подробностями обращайтесь к документации функции stream_notification_callback.

Поддерживаемые протоколы и обработчики (wrappers) ¶

PHP поставляется с множеством встроенных обработчиков для различных URL-протоколов для использования с функциями файловой системы, таких как fopen(), copy(), file_exists() и filesize(). В дополнение к этим обработчикам, можно регистрировать собственные обработчики, используя функцию stream_wrapper_register().

Замечание: URL синтаксис, используемый для описания обработчика, может быть только вида *scheme://...*. Варианты синтаксиса*scheme:/* и *scheme:* не поддерживаются.

Содержание ¶

User Contributed Notes 31 notes

gjaman at gmail dot com ¶

8 years ago

You can decompress (gzip) a input stream by combining wrappers:

eg: $x = file_get_contents("compress.zlib://php://input");

I used this method to decompress a gzip stream that was pushed to my webserver

file://

file:// — Доступ к локальной файловой системе

Описание ¶

Файловая система - это стандартная обертка для PHP, представляющая файловую систему на локальном компьютере. Когда задан относительный путь (путь, который не начинается с символов "/", "\", "\\" или с буквы жесткого диска в Windows), он будет применен к текущей рабочей директории. В большинстве случаев это директория, в которой находится сценарий, если она не была изменена. При использовании CLI SAPI директорией по умолчанию будет та, из которой вызвано исполнение сценария.

With some functions, such as fopen() and file_get_contents(), *include_path* may be optionally searched for relative paths as well.

Использование ¶

- `/path/to/file.ext`
- `relative/path/to/file.ext`
- `fileInCwd.ext`
- `C:/path/to/winfile.ext`
- `C:\path\to\winfile.ext`
- `\\smbserver\share\path\to\winfile.ext`
- `file:///path/to/file.ext`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Чтение	Да
Запись	Да
Добавление	Да
Одновременное чтение и запись	Да
Поддержка stat()	Да
Поддержка unlink()	Да
Поддержка rename()	Да
Поддержка mkdir()	Да
Поддержка rmdir()	Да

Список изменений ¶

Версия	Описание
5.0.0	Добавлено *file://*.

http://

https://

http:// -- https:// — Доступ к URL-адресам по протоколу HTTP(s)

Описание ¶

Предоставляет доступ только для чтения файлов/ресурсов через HTTP 1.0, используя метод HTTP GET. Для поддержки name-based виртуальных хостов вместе с запросом посылается заголовок *Host:*. Если вы сконфигурировали строку user_agent, используя ваш файл `php.ini` или контекст потока, то она также будет включена в запрос.

Этот поток также позволяет получить доступ к *содержимому* ресурса; заголовки сохраняются в переменной `$http_response_header`.

Если важно знать URL, с которого был получен документ (после всех переадресаций, которые были произведены), то вам необходимо обработать серию заголовков ответов, возвращаемых потоком.

INI-директива from будет использоваться для заголовка *From:*, если установлена и не переопределена в контексте Контекстные опции и параметры.

Использование ¶

- `http://example.com`
- `http://example.com/file.php?var1=val1&var2=val2`
- `http://user:password@example.com`
- `https://example.com`
- `https://example.com/file.php?var1=val1&var2=val2`
- `https://user:password@example.com`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Да
Чтение	Да
Запись	Нет
Добавление	Нет
Одновременное чтение и запись	Недоступно
Поддержка stat()	Нет
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

Список изменений ¶

Версия	Описание
4.3.7	Определение сбойных IIS-серверов для избежания ошибок типа *"SSL: Fatal Protocol Error"*.
4.3.0	Добавлен протокол *https://*.
4.0.5	Добавлена поддержка переадресаций.

Примеры ¶

Пример #1 Определение URL, с которого был забран документ после переадресаций

```
<?php
$url = 'http://www.example.com/redirecting_page.php';

$fp = fopen($url, 'r');

$meta_data = stream_get_meta_data($fp);
foreach ($meta_data['wrapper_data'] as $response) {

    /* Были ли мы переадресованы? */
    if (strtolower(substr($response, 0, 10)) == 'location: ') {

        /* Сохранить в $url адрес, куда нас переадресовали */
        $url = substr($response, 10);
    }

}

?>
```

Примечания ¶

Замечание: Протокол HTTPS поддерживается только когда расширение openssl включено.

Соединения HTTP предназначены только для чтения; запись данных или копирование файлов в HTTP-ресурс не поддерживается.

Отправка запросов *POST* и *PUT*, например, может быть выполнена с помощью HTTP-контекста.

Смотрите также ¶

- Опции контекста HTTP
- *$http_response_header*
- stream_get_meta_data() - Извлекает заголовок/метаданные из потоков/файловых указателей

User Contributed Notes 4 notes

ftp://

ftps://

ftp:// -- ftps:// — Доступ к URL-адресам по протоколу FTP(s)

Описание ¶

Позволяет читать существующие файлы и создавать новые файлы через FTP. Если сервер FTP не поддерживает режим passive mode, соединение будет невозможно.

Вы можете открыть файл либо для чтения, либо для записи, но не одновременно для того и другого. Если файл на сервере FTP уже существует, и вы пытаетесь открыть его для записи, но не указали опцию контекста *overwrite*, соединение будет невозможно. Если вам необходимо перезаписать существующие файлы на FTP, укажите опцию *overwrite* в контексте и откройте файл для записи. Кроме того, вы можете использовать FTP расширение.

Если вы установили директиву from в файле `php.ini`, то это значение будет отправлено как пароль при анонимном подключении к FTP.

Использование ¶

- `ftp://example.com/pub/file.txt`
- `ftp://user:password@example.com/pub/file.txt`
- `ftps://example.com/pub/file.txt`
- `ftps://user:password@example.com/pub/file.txt`

Опции ¶

Основная информация		
Атрибут	**PHP 4**	**PHP 5**
Ограничение поallow_url_fopen	Да	Да
Чтение	Да	Да
Запись	Да (только новые файлы)	Да (новые файлы / существующие файлы с опцией **`overwrite`**)
Добавление	Нет	Да
Одновременное чте-	Нет	Нет

Основная информация		
Атрибут	**PHP 4**	**PHP 5**
ние и запись		
Поддержка stat()	Нет	Начиная с PHP 5.0.0 доступны только: filesize(), filetype(), file_exists(), is_file(), и is_dir(). Начиная с PHP 5.1.0 доступны: filemtime().
Поддержка unlink()	Нет	Да
Поддержка rename()	Нет	Да
Поддержка mkdir()	Нет	Да
Поддержка rmdir()	Нет	Да

Список изменений ¶

Версия	Описание
4.3.0	Добавлена поддержка *ftps://*.

Примечания ¶

Замечание:
FTPS поддерживается только когда включена поддержка расширения OpenSSL. Если сервер не поддерживает SSL, то соединение переключается обратно на обычный нешифрованный протокол FTP.
Замечание: Дополнение
Начиная с PHP 5.0.0 файлы могут быть дописаны с помощью *ftp://* URL-обертки. В предыдущих версиях попытка дописать файл через *ftp://* приводила к ошибке.

Смотрите также ¶

- Параметры контекста FTP

User Contributed Notes 3 notes

php://

php:// — Доступ к различным потокам ввода-вывода

Описание ¶

PHP предоставляет несколько разнообразных потоков ввода-вывода, которые позволяют получить доступ к собственным потокам ввода-вывода PHP,к дескрипторам стандартного ввода, вывода и потока ошибок, к временным файловым потокам в памяти и на диске, и фильтрам, которые могут манипулировать другими файловыми ресурсами по мере их считывания или записи.

php://stdin, php://stdout и php://stderr ¶

`php://stdin`, `php://stdout` и `php://stderr` позволяют получить прямой доступ к соответствующим потокам ввода или вывода процесса PHP. Поток указывает на копию файлового дескриптора, таким образом, если вы откроете `php://stdin` и потом закроете его, вы закроете только вашу копию дескриптора. Актуальный поток, на который ссылается **`STDIN`** остается неизменным. Обратите внимание, что PHP демонстрировал ошибочное поведение в этом отношении до версии PHP 5.2.1. Рекомендуется просто использовать констан-

ты **STDIN**, **STDOUT** и **STDERR** вместо ручного открытия потоков, используя эти обертки.

Поток `php://stdin` предназначен только для чтения, тогда как `php://stdout` и `php://stderr` предназначены только для записи.

php://input ¶

`php://input` является потоком только для чтения, который позволяет вам читать необработанные данные из тела запроса. В случае POST-запросов предпочтительней использовать `php://input` вместо `$HTTP_RAW_POST_DATA`, так как этот метод не зависит от специальных `php.ini` директив. Кроме того, в тех случаях, где `$HTTP_RAW_POST_DATA` не заполняется по умолчанию, это потенциально менее затратно для памяти, чем активация директивы always_populate_raw_post_data. `php://input` не доступен с типом содержимого*enctype="multipart/form-data"*.

Замечание: До версии PHP 5.6, поток, открытый с `php://input` может быть прочтен только один раз. Поток не поддерживает операции поиска. Тем не менее, в зависимости от реализации SAPI интерфейса, может быть возможно открыть другой поток`php://input` и повторить чтение. Это возможно только если тело запроса заранее сохраняется. Это типично для случая с POST-запросом, но не для других методов запросов, таких как PUT или PROPFIND.

php://output ¶

`php://output` является потоком только для записи, который позволяет вам записать данные в выходной буфер аналогично как это делают функции print и echo.

php://fd ¶

`php://fd` предоставляет прямой доступ к указанному файловому дескриптору. Например, `php://fd/3` относится к файловому дескриптору 3.

php://memory и php://temp ¶

`php://memory` и `php://temp` являются потоками для чтения/записи и позволяют сохранять временные данные в файлоподобной обертке. Единственная разница между ними заключается в том, что `php://memory` будет всегда хранить данные в оперативной памяти, тогда как `php://temp` будет использовать временный файл в том случае, когда объем хранимой информации достигнет заданный лимит (по умолчанию составляет 2 Мб). Расположение этого временного файла определяется аналогично функции sys_get_temp_dir().

Размер лимита для `php://temp` может устанавливаться путем добавления */maxmemory:NN*, где *NN* - это максимальный размер данных в байтах для хранения в памяти перед использованием временного файла.

php://filter ¶

`php://filter` - это вид мета-обертки, предназначенный для разрешения применения фильтров к потоку во время открытия. Это полезно для функционально полных файловых функций таких, как readfile(), file(), и file_get_contents() там, где иначе не было возможности применить фильтр к потоку до того, как содержимое будет прочитано.

Поток `php://filter` принимает следующие параметры как часть своего пути. В одном пути можно указать несколько цепочек фильтров. Пожалуйста, ознакомьтесь с примерами и особенностями при использовании этих параметров.

Параметры для php://filter	
Название	**Описание**
resource=<поток для фильтрации>	Этот параметр является необходимым. Он указывает потоку, что его необходимо отфильтровать.
read=<список фильтров для применения к цепочке чтения>	Этот параметр является необязательным. Одно или более имен фильтров может быть указано здесь, разделенные вертикальной чертой (|).
write=<список фильтров для применения к цепочке записи>	Этот параметр является необязательным. Одно или более имен фильтров может быть указано здесь, разделенные вертикальной чертой (|).
<список фильтров для применения к обеим цепочкам чтения и записи>	Любой список фильтров, которые используются без префиксов *read=* или *write=*, будет применен к обоим потокам на чтение и на запись при необходимости.

Опции ¶

Основная информация (для *php://filter* смотрите информацию по обертке, которая подвергается фильтрации)	
Атрибут	**Поддержка**
Ограничение поallow_url_fopen	Нет
Ограничение поallow_url_include	только *php://input*, *php://stdin*, *php://memory* и *php://temp*.
Чтение	только *php://stdin*, *php://input*, *php://fd*, *php://memory* и *php://temp*.
Запись	только *php://stdout*, *php://stderr*, *php://output*, *php://fd*, *php://memory* и *php://temp*.
Добавление	только *php://stdout*, *php://stderr*, *php://output*, *php://fd*, *php://memory* и *php://temp*. (Эквивалентно записи)
Одновременное чтение и запись	только *php://fd*, *php://memory* и *php://temp*.
Поддержка stat()	только *php://memory* и *php://temp*.
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет
Поддержка stream_select()	только *php://stdin*, *php://stdout*, *php://stderr*, *php://fd* и *php://temp*.

Список изменений ¶

Версия	Описание
5.6.0	Стало возможным использовать повторно *php://input*.
5.3.6	Был добавлен *php://fd*.
5.1.0	Были добавлены *php://memory* и *php://temp*.
5.0.0	Был добавлен *php://filter*.

Примеры ¶

Пример #1 php://temp/maxmemory

Этот необязательный параметр позволяет установить лимит памяти до того, как *php://temp* начнет использовать временный файл.

```
<?php
// Установка предела в 5 MB.
$fiveMBs = 5 * 1024 * 1024;
$fp = fopen("php://temp/maxmemory:$fiveMBs", 'r+');

fputs($fp, "hello\n");

// Читаем то, что мы записали.
rewind($fp);
echo stream_get_contents($fp);
?>
```

Пример #2 php://filter/resource=<поток для фильтрации>

Этот параметр должен быть расположен в конце вашей спецификации *php://filter* и должен указывать на поток, который вы хотите фильтровать.

```
<?php
/* Это просто эквивалентно:
  readfile("http://www.example.com");
  так как на самом деле фильтры не указаны */

readfile("php://filter/resource=http://www.example.com");
?>
```

Пример #3 php://filter/read=<список фильтров для применения к цепочке чтения>

Этот параметр принимает один или более имен фильтров, разделенных вертикальной чертой |.

```
<?php
/* Этот скрипт выведет содержимое
  www.example.com полностью в верхнем регистре */
readfile("php://filter/read=string.toupper/resource=http://www.example.com");

/* Этот скрипт делает тоже самое, что вверхний, но
  будет также кодировать алгоритмом ROT13 */
```

```
readfile("php://filter/read=string.toupper|string.rot13/resource
=http://www.example.com");
?>
```

Пример #4 php://filter/write=<список фильтров для применения к цепочке записи>

Этот параметр принимает один или более имен фильтров, разделенных вертикальной чертой |.

```
<?php
/* Этот скрипт будет фильтровать строку "Hello World"
  через фильтр rot13, затем записывать результат в
  файл example.txt в текущей директории */
file_put_contents("php://filter/write=string.rot13/resource=exam
ple.txt","Hello World");
?>
```

zlib:// bzip2:// zip://

zlib:// -- bzip2:// -- zip:// — Сжатые потоки

Описание ¶

`zlib:` PHP 4.0.4 - PHP 4.2.3 (только на системах с fopencookie)

`compress.zlib://` и `compress.bzip2://` PHP 4.3.0 и выше

`zlib:` работает как gzopen() за исключением того, что этот поток может использоваться функцией fread() и другими функциями, работающими с файловой системой. Устарела начиная с PHP 4.3.0 ввиду неоднозначности при наличии файлов, содержащих ':'; используйте взамен `compress.zlib://`.

`compress.zlib://` и `compress.bzip2://` соответствуют gzopen() и bzopen() со ответственно и работают даже в системах, не поддерживающих fopencookie.

ZIP-модуль добавляет обертку `zip:`.

Использование ¶

- `compress.zlib://file.gz`
- `compress.bzip2://file.bz2`
- `zip://archive.zip#dir/file.txt`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Чтение	Да
Запись	Да (кроме *zip://*)
Добавление	Да (кроме *zip://*)

Основная информация	
Атрибут	**Поддержка**
Чтение и запись одновременно	Нет
Поддержка stat()	Нет, используйте стандартную обертку *file://* для получения информации по сжатым файлам.
Поддержка unlink()	Нет, используйте стандартную обертку *file://* для удаления сжатых файлов.
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

User Contributed Notes 4 notes

data://

data:// — Схема Data (RFC 2397)

Описание ¶

`data:` (» RFC 2397) - это обертка потоков, доступная с PHP 5.2.0.

Использование ¶

- `data://text/plain;base64,`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Ограничение по allow_url_include	Да
Чтение	Да
Запись	Нет
Добавление	Нет
Чтение и запись одновременно	Нет
Поддержка stat()	Нет
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

Примеры ¶

Пример #1 Вывод содержимого data://

```
<?php
// выводит "I love PHP"
echo file_get_contents('data://text/plain;base64,SSBsb3ZlIFBIUAo
=');
?>
```

Пример #2 Получение типа потока

```php
<?php
$fp   = fopen('data://text/plain;base64,', 'r');
$meta = stream_get_meta_data($fp);

// выводит "text/plain"
echo $meta['mediatype'];
?>
```

User Contributed Notes 3 notes

glob://

glob:// — Нахождение путей, соответствующих шаблону

Описание ¶

Обертка потока `glob:` доступна с версии PHP 5.3.0.

Использование ¶

- `glob://`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Ограничение по allow_url_include	Нет
Чтение	Нет
Запись	Нет
Добавление	Нет
Одновременное чтение и запись	Нет
Поддержка stat()	Нет
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

Примеры ¶

Пример #1 Основы использования

```php
<?php
// Просмотреть все файлы *.php в директории ext/spl/examples/
// и напечатать имена файлов и их размеры
$it = new DirectoryIterator("glob://ext/spl/examples/*.php");
foreach($it as $f) {
    printf("%s: %.1FK\n", $f->getFilename(), $f-
>getSize()/1024);
}
?>
```

```
tree.php: 1.0K
findregex.php: 0.6K
findfile.php: 0.7K
dba_dump.php: 0.9K
nocvsdir.php: 1.1K
phar_from_dir.php: 1.0K
ini_groups.php: 0.9K
directorytree.php: 0.9K
dba_array.php: 1.1K
class_tree.php: 1.8K
```

phar://

phar:// — PHP архив

Описание ¶

Обертка потока `phar://` доступна начиная с версии PHP 5.3.0. Смотрите раздел обертка потока Phar для более детального описания.

Использование ¶

- `phar://`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Ограничение по allow_url_include	Нет
Чтение	Да
Запись	Да
Добавление	Нет
Одновременное чтение и запись	Да
Поддержка stat()	Да
Поддержка unlink()	Да
Поддержка rename()	Да
Поддержка mkdir()	Да
Поддержка rmdir()	Да

Смотрите также ¶

- Контекстные опции Phar

ssh2://

ssh2:// — Secure Shell 2

Описание ¶

`ssh2.shell://` `ssh2.exec://` `ssh2.tunnel://` `ssh2.sftp://` `ssh2.scp://`
PHP 4.3.0 и более поздние (PECL)

Замечание: Эта обертка не включена по умолчанию Для того, чтобы использовать обертки `ssh2.*://` вам необходимо установить расширение » SSH2, доступное в » PECL.
В дополнение для принятия традиционных аутентификационных данных URI, обертки ssh2 будут также повторно использовать открытые соединения, передавая аутентификационную информацию в хост-части URL.

Использование ¶

- `ssh2.shell://user:pass@example.com:22/xterm`
- `ssh2.exec://user:pass@example.com:22/usr/local/bin/somecmd`
- `ssh2.tunnel://user:pass@example.com:22/192.168.0.1:14`
- `ssh2.sftp://user:pass@example.com:22/path/to/filename`

Опции ¶

Основная информация					
Атрибут	**ssh2.shell**	**ssh2.exec**	**ssh2.tunnel**	**ssh2.sftp**	**ssh2.scp**
Ограничение по allow_url_fopen	Да	Да	Да	Да	Да
Чтение	Да	Да	Да	Да	Да
Запись	Да	Да	Да	Да	Нет
Добавление	Нет	Нет	Нет	Да (когда поддерживается сервером)	Нет
Одновременная чтение и запись	Да	Да	Да	Да	Нет
Поддержка stat()	Нет	Нет	Нет	Да	Нет
Поддержка unlink()	Нет	Нет	Нет	Да	Нет
Поддержка rename()	Нет	Нет	Нет	Да	Нет
Поддержка mkdir()	Нет	Нет	Нет	Да	Нет
Поддержка rmdir()	Нет	Нет	Нет	Да	Нет

Опции контекста		
Имя	**Использование**	**По умолчанию**
session	Предварительно соединенный ресурс ssh2 для повторного использования	
sftp	Предварительно выделенный ресурс sftp для повторного использования	
methods	Обмен ключами, ключ хоста, шифр, компрессия, и MAC методы для использования	
callbacks		
username	Имя пользователя для соединения	
password	Пароль для аутентификации	
pubkey_file	Имя файла, в котором находится публичный ключ для аутентификации	
privkey_file	Имя файла, в котором находится приватный ключ для аутентификации	
env	Ассоциативный массив с переменными окружения, которые необходимо установить	
term	Тип эмуляции терминала для запроса, когда выделяется pty	
term_width	Ширина терминала, запрашивается когда выделя-	

Опции контекста		
Имя	**Использование**	**По умолчанию**
	ется pty	
term_height	Высота терминала, запрашивается когда выделяется pty	
term_units	Единицы, в которых измеряются term_width и term_height	**SSH2_TERM_UNIT_CHARS**

Примеры ¶

Пример #1 Открытие потока из активного соединения

```
<?php
$session = ssh2_connect('example.com', 22);
ssh2_auth_pubkey_file($session, 'username', '/home/username/.ssh
/id_rsa.pub',
                                            '/home/username/.ssh
/id_rsa', 'secret');
$stream = fopen("ssh2.tunnel://$session/remote.example.com:1234"
, 'r');
?>
```

Пример #2 Переменная *$session* должна быть доступна!

Если вы хотите использовать какую-либо из оберток *ssh2.*://$session*, необходимо сохранить доступным ресурс, хранящийся в переменной *$session*. Следующий код не будет иметь желаемого эффекта:

```
<?php
$session = ssh2_connect('example.com', 22);
ssh2_auth_pubkey_file($session, 'username', '/home/username/.ssh
/id_rsa.pub',
                                            '/home/username/.ssh
/id_rsa', 'secret');
$connection_string = "ssh2.sftp://$session/";
unset($session);
$stream = fopen($connection_string . "path/to/file", 'r');
?>
```

unset() закрывает сессию, потому что *$connection_string* не является ссылкой на переменную *$session*, а только ее текстовым представлением. Это также происходит и в случае неявного вызова unset() при выходе из области видимости (например, из функции).

User Contributed Notes 2 notes

rar://

rar:// — RAR

Описание ¶

Эта обертка принимает URL-кодированный путь к RAR-архиву (относительный или абсолютный), необязательный символ звездочки (*), необязательный символ решетки (#) и необязательное URL-кодированное имя такое, как хранится в архиве. Для указания имени содержимого требуется символ решетки, начальный обратный слеш в названии содержимого необязателен.

Эта обертка может открывать файлы и директории. Когда открываются директории, знак звездочки требует, чтобы имена объектов директории были закодированы unencode. Если такой знак не указан, они будут возвращены в URL-кодировке. Смысл этого в том, чтобы позволить обертке корректно использовать встроенную функциональность, такую как RecursiveDirectoryIterator когда присутствуют имена файлов, которые кажутся как url-закодированные данные.

Если символ решетки и часть имени записи не включена, будет отображен корень архива. Это отличается от обычных директорий тем, что результирующий поток не будет содержать такую информацию, как время модификации, так как корневая директория не сохраняется как отдельная запись в архиве. Использование обертки с RecursiveDirectoryIterator требует, чтобы символ решетки был включен в URL, когда происходит доступ к корню, так чтобы URL потомков мог быть сконструирован правильно.

Замечание: Эта обертка не включена по умолчанию
Для того, чтобы использовать обертку `rar://`, вам необходимо установить расширение » rar, доступное из репозитория » PECL.

`rar://` Доступно начиная с PECL rar 3.0.0

Использование ¶

- `rar://<url encoded archive name>[*][#[<url encoded entry name>]]`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Ограничение по allow_url_include	Нет
Чтение	Да
Запись	Нет
Добавление	Нет
Одновременное чтение и запись	Нет
Поддержка stat()	Да
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет
Опции контекста	

Название	Использование	По умолчанию
open_password	Пароль используется для шифрования заголовков архива, если таковые есть. WinRAR будет шифровать все файлы с таким же паролем, как и пароль заголовков, когда последний присутствует. Таким образом, для архивов с зашифрованными заголовками опция *file_password* будет игнорироваться.	
file_password	Пароль, используемый для шифрования файла, если таковой имеется. Если заголовки также зашифрованы, эта опция будет игнорирована в пользу *open_password*. Причина этого в том, что нет смысла в использовании одновременно двух разных паролей для шифрования отдельно заголовков и отдельно файлов. Нет таких архивов, где бы это пригодилось. Заметим, что если у архива отсутствуют зашифрованные заголовки, то опция *open_password* будет игнорирована и эта опция должна быть использована вместо нее.	
volume_callback	Обратный вызов для определения пути недостающих томов архива. Смотрите RarArchive::open() для более детальной информации.	

Примеры ¶

Пример #1 Обход RAR-архива

```
<?php

class MyRecDirIt extends RecursiveDirectoryIterator {
    function current() {
        return rawurldecode($this->getSubPathName()) .
            (is_dir(parent::current())?" [DIR]":"");
    }
}

$f = "rar://" . rawurlencode(dirname(__FILE__)) .
    DIRECTORY_SEPARATOR . 'dirs_and_extra_headers.rar#';

$it = new RecursiveTreeIterator(new MyRecDirIt($f));

foreach ($it as $s) {
    echo $s, "\n";
}
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
|-allow_everyone_ni [DIR]
|-file1.txt
|-file2_ℵ.txt
|-with_streams.txt
\-ℵ [DIR]
  |-ℵ\%2Fempty%2E [DIR]
  | \-ℵ\%2Fempty%2E\file7.txt
```

```
  |-ж\empty [DIR]
  |-ж\file3.txt
  |-ж\file4_ж.txt
  \-ж\ж_2 [DIR]
    |-ж\ж_2\file5.txt
    \-ж\ж_2\file6_ж.txt
```

Пример #2 Открытие зашифрованного файла (шифрование заголовка)

```php
<?php
$stream = fopen("rar://" .
    rawurlencode(dirname(__FILE__)) . DIRECTORY_SEPARATOR .
    'encrypted_headers.rar' . '#encfile1.txt', "r", false,
    stream_context_create(
        array(
            'rar' =>
                array(
                    'open_password' => 'samplepassword'
                )
            )
        )
    );
var_dump(stream_get_contents($stream));
/* дата создания и дата последнего доступа включается опциональн
о в WinRAR, поэтому у
 * большинства файлов их нет */
var_dump(fstat($stream));
?>
```

Результатом выполнения данного примера будет что-то подобное:

```
string(26) "Encrypted file 1 contents."
Array
(
    [0] => 0
    [1] => 0
    [2] => 33206
    [3] => 1
    [4] => 0
    [5] => 0
    [6] => 0
    [7] => 26
    [8] => 0
    [9] => 1259550052
    [10] => 0
    [11] => -1
    [12] => -1
    [dev] => 0
    [ino] => 0
    [mode] => 33206
    [nlink] => 1
    [uid] => 0
    [gid] => 0
```

```
    [rdev] => 0
    [size] => 26
    [atime] => 0
    [mtime] => 1259550052
    [ctime] => 0
    [blksize] => -1
    [blocks] => -1
)
```

ogg://

ogg:// — Аудио потоки

Описание ¶

Файлы, открываемые для чтения с использованием обертки `ogg://`, рассматриваются как сжатый аудио поток, кодируемый с использованием кодека *OGG/Vorbis*. Аналогично, файлы открытые для записи или добавления через обертку `ogg://` записываются как сжатые звуковые данные. Функция stream_get_meta_data(), когда используется с файлами *OGG/Vorbis* открытыми для чтения, будет возвращать разнообразную информацию о потоке, включая тэг производителя **`vendor`**, комментарии **`comments`**, число каналов**`channels`**, частоту дискретизации **`rate`**, и диапазон частоты кодирования, описываемый: **`bitrate_lower`**, **`bitrate_upper`**,**`bitrate_nominal`**, и **`bitrate_window`**.

`ogg://` PHP 4.3.0 и старше (PECL)

Замечание: Данная обертка не доступна по умолчанию Для того чтобы использовать обертку `ogg://` вы должны установить расширение » OGG/Vorbis доступное в » PECL.

Использование ¶

- `ogg://soundfile.ogg`
- `ogg:///path/to/soundfile.ogg`
- `ogg://http://www.example.com/path/to/soundstream.ogg`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет
Чтение	Да
Запись	Да
Добавление	Да
Одновременные чтение и запись	Нет
Поддержка stat()	Нет
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

Установки для контекста			
Название	**Использование**	**Значение по умолчанию**	**Режим**
pcm_mode	Опция PCM кодирования применяемая во время чтение, одна из:**OGGVORBIS_PCM_U8**, **OGGVORBIS_PCM_S8**, **OGGVORBIS_PCM_U16_BE**,**OGGVORBIS_PCM_S16_BE**, **OGGVORBIS_PCM_U16_LE**, и **OGGVORBIS_PCM_S16_LE**. (8 или 16 битное, со знаком или без, прямой или обратный порядок байтов)	OGGVORBIS_PCM_S16_LE	Чтение
rate	Частота дискретизации входных данных, выраженная в Гц	44100	Запись/Добавление
bitrate	Когда дано целое число, постоянный битрейт при котором кодировать. (от 16000 до 131072) Когда дано вещественное число, качество переменного битрейта для использования. (от -1.0 до 1.0)	128000	Запись/Добавление
channels	Количество аудио каналов для кодирования, обычно 1 (Моно), или 2 (Стерео). Может варьироваться вплоть до 16.	2	Запись/Добавление
comments	Массив строк для запись в заголовок трека.		Запись/Добавление

Примеры ¶

User Contributed Notes 1 note

expect://

expect:// — Потоки для взаимодействия с процессами

Описание ¶

Потоки, открытые с помощью обертки `expect://`, предоставляют доступ к stdio, stdout и stderr процессов через PTY.

Замечание: Эта обертка отключена по умолчанию Для того, чтобы использовать обертку `expect://`, необходимо установить модуль » Expect, доступный в » PECL.

`expect://` PHP 4.3.0 и выше (PECL)

Использование ¶

- `expect://command`

Опции ¶

Основная информация	
Атрибут	**Поддержка**
Ограничение по allow_url_fopen	Нет

Основная информация	
Атрибут	**Поддержка**
Чтение	Да
Запись	Да
Добавление	Да
Чтение и запись одновременно	Нет
Поддержка stat()	Нет
Поддержка unlink()	Нет
Поддержка rename()	Нет
Поддержка mkdir()	Нет
Поддержка rmdir()	Нет

Примеры ¶