Пространства имён

Пространства имён введены в PHP для решения проблем в больших PHP-библиотеках. В PHP все определения классов глобальны, поэтому авторы библиотек должны выбирать уникальные имена для создаваемых ими классов.

Это делается для того, чтобы при использовании библиотеки совместно с другими библиотеками не возникало конфликтов имён. Обычно это достигается введением в имена классов префиксов. Например: если мы будем использовать класс **dataBase** - велика вероятность, что такое имя класса будет присутствовать и в других библиотеках, а при их совместном использовании возникнет ошибка. Поэтому мы вынуждены использовать для класса другое имя. Например: **ourLibraryDataBase** Такие действия приводят к чрезмерному увеличению длины имён классов.

Пространства имён позволяют разработчику управлять зонами видимости имён, что избавляет от необходимости использования префиксов и чрезмерно длинных имён. Все это служит повышению читабельности кода.

Пространства имён доступны в PHP начиная с версии 5.3.0. Данная секция экспериментальна и возможно будет подвержена изменениям.

Пространства имён

Пространство имён определяется посредством ключевого слова *namespace*, которое должно находиться в самом начале файла. Пример:

```
<?php
    namespace MyProject::DB;
    const CONNECT_OK = 1;
    class Connection { /* ... */ }
    function connect() { /* ... */  }
    ?>
```

Это пространство имён может быть использовано в разных файлах.

Пространства имён могут включать определения классов, констант и функций. Но не должны включать обычного кода.

Пространства имён

Определение пространства имён работает так:

Внутри пространства имён все имена классов, функций и констант автоматически будут префиксированы именем пространства имён. Имена классов при вызове должны быть полными, так например при вызове класса из примера выше должно использоваться следующее имя **MyProject::DB::Connection**.

Определения констант создают константы, состоящие из имени пространства имён и имени константы. Как и константы классов - константы пространства имён могут содержать данные только скалярного типа.

Поиск неквалифицированного имени класса (т.е. не содержащего **::**) осуществляется в следующей последовательности:

1. Попытка найти класс в текущем пространстве имён (т.е. префиксирование класса именем текущего пространства имён) без попытки автозагрузки (autoload).
2. Попытка найти класс в глобальном пространстве имён без попытки автозагрузки (autoload).
3. Попытка автозагрузки в текущем пространстве имён.
4. В случае неудачи предыдущего - отказ.

Поиск неквалифицированного имени функции (т.е. не включающего **::**) во время выполнения производится сначала в текущем пространстве имён, затем в глобальном пространстве имён.

Поиск неквалифицированного имени константы производится сначала в текущем пространстве имён, затем среди глобально объявленных констант.

Объектно-ориентированное программирование в PHP

Пространства имён

Использование пространств имён

На имена классов и функций, объявленных внутри пространства имён всегда можно сослаться по полному имени: **MyProject::DB::Connection** или **MyProject::DB::connect()** .

```
<?php
    require 'MyProject/Db/Connection.php';
    $x = new MyProject::DB::Connection;
    MyProject::DB::connect();
 ?>
```

Пространства имён могут быть импортированы в текущий контекст (глобальный или пространство имён) при помощи оператора *use*. Синтаксис оператора use:

```
<?php
    /* ... */
    use Some::Name as Othername;
    // Самая простая форма :
    use Foo::Bar;
    // это то же что и :
    use Foo::Bar as Bar;
?>
```

Импортированные пространства имён работают следующим образом: каждый раз, когда компилятор встречает локальное имя *Othername* (как простое имя или как префикс более длинного, разделенного *::* имени), оно заменяется на импортированное имя *Some::Name*.

Пространства имён

Оператор *use* может быть использован только в глобальном пространстве имён - не внутри класса или функции. Импортированные имена действуют с точки импортирования до конца текущего файла. Рекомендуется импортировать имена в начале файла во избежание путаницы.

```
<?php
    require 'MyProject/Db/Connection.php';
    use MyProject::DB;
    use MyProject::DB::Connection as DbConnection;

    $x = new MyProject::DB::Connection();
    $y = new DB::connection();
    $z = new DbConnection();
    DB::connect();
    ?>
```

Замечание: Операция импорта пространств имён выполняется только во время компиляции. Все локальные имена заменяются компилятором на их полные эквиваленты. Динамический импорт пространств имён невозможен.

Глобальное пространство имён

Без определения какого-либо пространства имён, все определения классов и функций оказываются в глобальном пространстве - как это было в PHP до появления поддержки пространств имён. Префиксирование имени знаками *::* указывает, что имя из глобального пространства имён - работает даже в контексте пространства имён.

```
<?php
    namespace A::B::C;

    /* This function is A::B::C::fopen */
    function fopen() {
    /* ... */
    $f = ::fopen(...); // вызывает глобальную fopen
    return $f;
    }
?>
```

Пространства имён

__NAMESPACE__

Константа времени компиляции **__NAMESPACE__** определяет текущее пространство имён. Вне пространства имён эта константа имеет значение пустой строки. Эта константа используется когда требуется сформировать полное имя класса или функции из текущего пространства имён.

```
<?php
    namespace A::B::C;

    function foo() {
    // do stuff
    }

    set_error_handler(__NAMESPACE__ . "::foo");
?>
```

Правила разбора имён

Разбор имён происходит по следующим правилам:

1. Все квалифицированные имена транслируются во время компиляции в соответствии с текущими импортированными пространствами имён. К примеру, если импортировано пространство имён A::B::C, вызов C::D::e() будет транслирован как A::B::C::D::e().
2. Неквалифицированные имена классов транслируются во время компиляции в соответствии с текущими импортированными пространствами имён (полные имена заменяют короткие импортированные имена). К примеру, если пространство имён *A::B::C* импортировано, new C() будет транслировано как new A::B::C().
3. Внутри пространства имён вызов неквалифицированных функций, определенных в этом же пространстве имён интерпретируется как вызов в данном пространстве имён во время компиляции.
4. Внутри пространства имён (например *A::B*) вызов неквалифицированных функций, не определенных в этом пространстве имён будет разрешаться во время выполнения. Вызов функции *foo()* будет разрешаться следующим образом:
    1. Поиск в текущем пространстве имён: *A::B::foo()*.
    2. Поиск ***внутренней*** PHP функции *foo()*.

В случае неудачи всех предыдущих попыток будет использован вызов определенной в глобальном пространстве имён функции *::foo()*.

Правила разбора имён

5. Внутри пространства имён (например *A::B*), вызов неквалифицированных классов разрешается во время выполнения. Например вызов new C() будет разрешаться следующим образом:
   1. Поиск класса в текущем пространстве имён: *A::B::C*.
   2. Попытка вызова ***внутреннего*** PHP-класса *C*.
   3. Попытка автозагрузки *A::B::C*.

В случае неудачи всех предыдущих, будет использован вызов new ::C().

6. Вызов квалифицированных функций разрешается во время выполнения. Например вызов *A::B::foo()* будет разрешаться следующим образом :
   1. Поиск функции *foo()* в пространстве имён *A::B*.
   2. Поиск класса *A::B* и вызов его статического метода *foo()*. Будет сделана автозагрузка класса, если необходимо.

7. Квалифицированные имена классов разрешаются во время компиляции, как классы соответствующего пространства имён. К примеру new A::B::C() будет ссылаться на класс **C** пространства имён *A::B*.

Создание единой точки входа

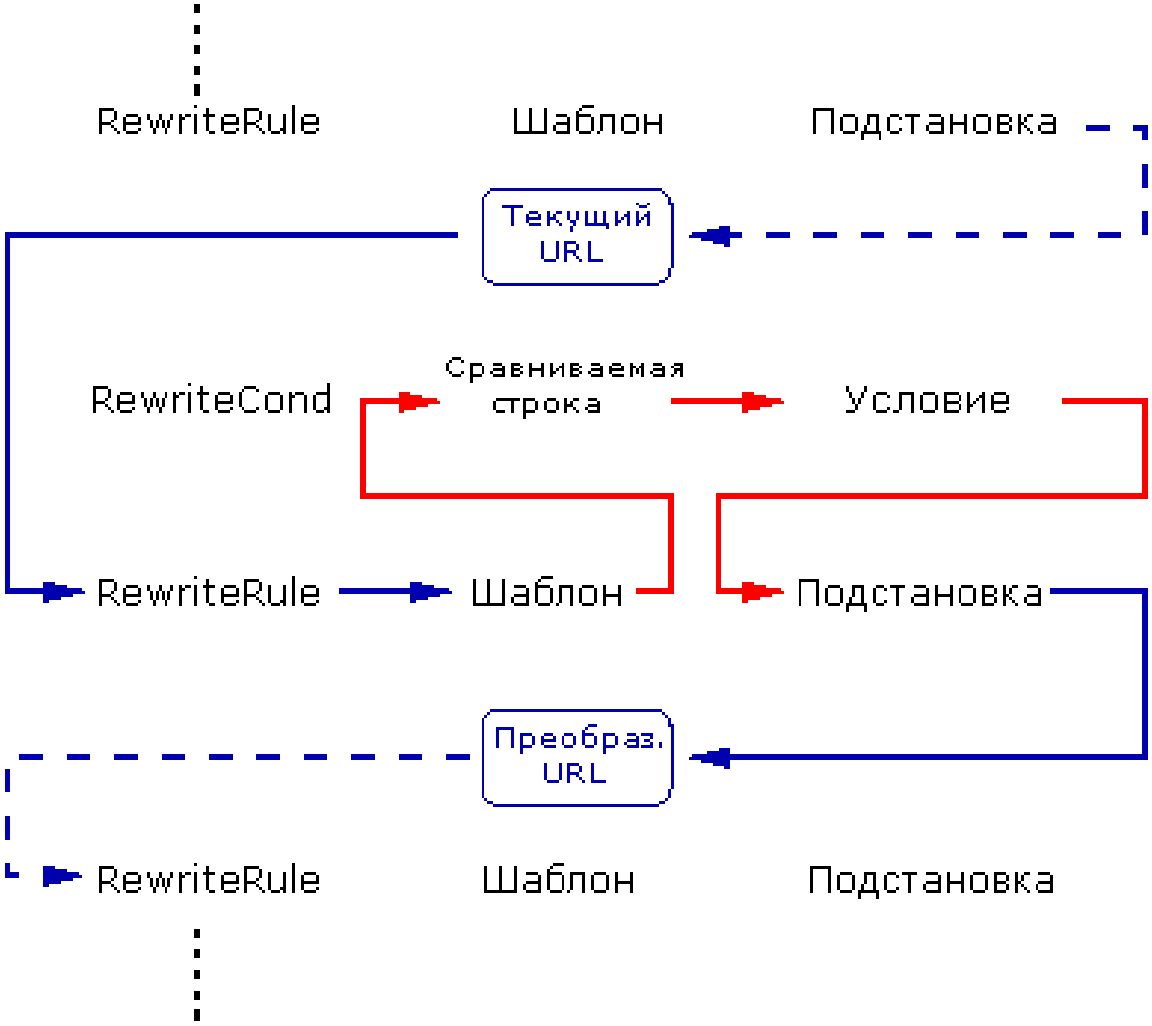
Apache mod_rewrite

Модуль **mod_rewrite** является **программным модулем** веб сервера **Apache**.

Основная функция mod_rewrite — **манипуляция с URL**.

Модуль оперирует с полными URL (включая path-info) и в контексте сервера (конфигурация задается в файле настроек Apache - **httpd.conf**) и в контексте каталога (конфигурация задается в файле **.htaccess**, расположенном в данном каталоге) и даже может генерировать части строки запроса в качестве результата. Преобразованный результат может приводить к внутренней обработке, внешнему перенаправлению запроса или даже к прохождению через внутренний прокси модуль Apache.

Apache mod_rewrite



Директивы mod_rewrite

- RewriteEngine
- RewriteRule
- RewriteCond
- RewriteBase
- RewriteOptions
- RewriteLog
- RewriteLogLevel
- RewriteLock
- RewriteMap

Самые важные директивы mod_rewrite:

- RewriteEngine: Включает/выключает механизм mod_rewrite для текущего запроса.
- RewriteCond: определяет условия для какого-либо правила. Перед директивой RewriteRule располагаются одна или несколько директив RewriteCond. Следующее за ними правило преобразования(RewriteRule) рассматривается только тогда, когда URI соответствует условиям этих директив(RewriteCond).
- RewriteRule: Описывает правило изменения адреса URL.

Apache mod_rewrite

Синтаксис директивы RewriteEngine имеет вид:

RewriteEngine on|off

Директива RewriteEngine включает или выключает работу механизма преобразований URL.

Если она установлена в положение off модуль RewriteRule не работает.

Эту директиву целесообразно использовать для выключения модуля mod_rewrite вместо простого комментирования директив RewriteRule.

Синтаксис директивы RewriteCond имеет вид:

RewriteCond *СравниваемаяСтрока Условие*

Директива RewriteCond определяет условия работы для какого-либо правила, идущего следом.

СравниваемаяСтрока - строка которая может содержать следующие дополнительные конструкции в дополнении к простому тексту:

- **RewriteRule обратные_связи**: Это обратные связи вида
  $N
  (0 <= N <= 9) предоставляющие доступ к сгруппированным частям (в круглых скобках!) шаблона из соответствующей директивы RewriteRule (единственной, следующей сразу за текущим набором директив RewriteCond).
- **RewriteCond обратные_связи**: Это обратные связи вида
  %N
  (1 <= N <= 9) предоставляющие доступ к сгруппированным частям (в круглых скобках!) шаблона из соответствующей директивы RewriteCond в текущем наборе условий.
- **RewriteMap расширения**: Это расширения вида
  ${mapname:key|default}
- **Переменные сервера**: Это переменные вида
  %{NAME_OF_VARIABLE}

Паттерн Model View Controller

Apache mod_rewrite

Переменные сервера

HTTP Headers (заголовки)

- %{HTTP_USER_AGENT}
- %{HTTP_REFERER}
- %{HTTP_COOKIE}
- %{HTTP_FORWARDED}
- %{HTTP_HOST}
- %{HTTP_PROXY_CONNECTION}
- %{HTTP_ACCEPT}

Request (переменные запроса)

- %{REMOTE_ADDR}
- %{REMOTE_HOST}
- %{REMOTE_USER}
- %{REMOTE_IDENT}
- %{REQUEST_METHOD}
- %{SCRIPT_FILENAME}
- %{PATH_INFO}
- %{QUERY_STRING}
- %{AUTH_TYPE}

Server (переменные сервера)

- %{DOCUMENT_ROOT}
- %{SERVER_ADMIN}
- %{SERVER_NAME}
- %{SERVER_ADDR}
- %{SERVER_PORT}
- %{SERVER_PROTOCOL}
- %{SERVER_SOFTWARE}

Время

- %{TIME_YEAR}
- %{TIME_MON}
- %{TIME_DAY}
- %{TIME_HOUR}
- %{TIME_MIN}
- %{TIME_SEC}
- %{TIME_WDAY}
- %{TIME}

Специальные

- %{API_VERSION}
- %{THE_REQUEST}
- %{REQUEST_URI}
- %{REQUEST_FILENAME}
- %{IS_SUBREQ}

Условие - это шаблон условия, *т.е.*, какое-либо **регулярное выражение** применяемое к текущему экземпляру *СравниваемойСтроки*, *т.е.*, *СравниваемаяСтрока* просматривается на поиск соответствия *Условию*.

Есть некоторые специальные варианты *Условий*. Вместо обычных строк с регулярными выражениями можно также использовать один из следующих вариантов:

- '**<Условие**' (лексически меньше)

  Условие считается простой строкой и лексически сравнивается с *СравниваемаяСтрока*. Истинно если *СравниваемаяСтрока* лексически меньше чем *Условие*.
- '**>Условие**' (лексически больше)

  Условие считается простой строкой и лексически сравнивается с *СравниваемаяСтрока*. Истинно если *СравниваемаяСтрока* лексически больше чем *Условие*.
- '**=Условие**' (лексически равно)

  Условие считается простой строкой и лексически сравнивается с *СравниваемаяСтрока*. Истинно если *СравниваемаяСтрока* лексически равно *Условие*, т.е. эти две строки полностью одинаковы (символ в символ). Если *Условие* имеет вид "" (две кавычки идущих подряд) - сравнивает *СравниваемаяСтрока* с пустой строкой.
- '**-d**' (является ли каталогом)

  СравниваемаяСтрока считается путем, проверяется существование этого пути и то что этот путь является каталогом.
- '**-f**' (является ли обычным файлом)

  СравниваемаяСтрока считается путем, проверяется существование этого пути и то что этот путь является обычным файлом.

- '**-s**' (является ли обычным файлом с ненулевым размером)

СравниваемаяСтрока считается путем, проверяется существование этого пути и то что этот путь является обычным файлом, размер которого больше нуля.

- '**-l**' (является ли символической ссылкой)

СравниваемаяСтрока считается путем, проверяется существование этого пути и то что этот путь является символической ссылкой.

- '**-F**' (проверка существования файла через подзапрос)

Проверяет через все списки контроля доступа сервера, существующие в настоящий момент, является ли *СравниваемаяСтрока* существующим файлом, доступным по этому пути. Для этой проверки используется внутренний подзапрос, поэтому используйте эту опцию с осторожностью — это отрицательно сказывается на производительности сервера!

- '**-U**' (проверка существования URL через подзапрос)

Проверяет через все списки контроля доступа сервера, существующие в настоящий момент, является ли *СравниваемаяСтрока* существующим URL, доступным по этому пути. Для этой проверки используется внутренний подзапрос, поэтому используйте эту опцию с осторожностью — это отрицательно сказывается на производительности сервера!

Apache mod_rewrite

Дополнительно возможно устанавливать специальные **флаги** для *Условия* добавляя
[*flags*]
третьим аргументом в директиву RewriteCond.

Flags список следующих флагов разделенных запятыми:

- '**nocase|NC**' (регистронезависимость)
  Этот флаг эффективен только для сравнений между *СравниваемаяСтрока* и *Условие*. Он не работает при проверках в файловой системе в подзапросах.
- '**ornext|OR**' (ИЛИ следующее условие)

Пример:

```
RewriteCond %{REMOTE_HOST}  ^host1.*  [OR]
RewriteCond %{REMOTE_HOST}  ^host2.*  [OR]
RewriteCond %{REMOTE_HOST}  ^host3.*
RewriteRule ...правило RewriteRule...
```

Apache mod_rewrite

Обобщенный синтаксис директивы RewriteRule имеет вид:

RewriteRule Pattern Substitution [Optional Flags]

* Pattern - регулярное выражение шаблона. Если URL соответствует шаблону, то правило выполняется. Иначе правило пропускается.
* Substitution - новый URL, который будет использоваться вместо соответствующего шаблону адреса.
* [Optional Flags] - один или несколько флагов, которые определяют поведение правила.

Возможно добавить в файл .htaccess столько правил RewriteRule, сколько нужно. Модуль mod_rewrite проходит все правила каждый раз при запросе, обрабатывая соответствующие адресу URL.

Если правило изменяет запрашиваемый URL на новый адрес, то новый URL используется дальше при проходе по файлу .htaccess, и может соответствовать другому правилу RewriteRule, размещающемуся далее в файле. (Если нужно изменить такое поведение, то надо использовать флаг L ("последнее правило").)

Синтаксис регулярных выражений:

^ - начало строки
$ - конец строки
. - (точка) любой символ
(a|b) - a или b
(...) - выбор группы
[abc] - любой символ из диапазона (a или b или c)
[^abc] - ни один символ из диапазона (ни a или b или c)
a? - символ a 1 или 0 раз
a* - символ a 0 или более раз
a+ - символ a 1 или более раз
a{3} - символ a точно 3 раза
a{3,} - символ a более 3 раз
a{3,6} - символ a от 3 до 6 раз
!(pattern) ! - отрицание

Флаги RewriteRule

R[=code] Перенаправление на новый URL по заданному коду
F Forbidden (отправляет заголовок 403)
G Больше не существует (Gone)
P Прокси (Proxy)
L Последнее правило
N Следующий
C Chain
T=mime-type Установка mime-type
NS Skip if internal sub-request
NC Не зависимый от регистра символов
QSA Append query string (Прибавляет строку запроса)
NE Не отменяет результат
PT Через
S=x Пропустить следующие x правил
E=var:value Устанавливает переменную окружения "var" в "value".

Коды ответа сервера:

301 Moved permanently (Перемещен постоянно)
302 Moved temporarily (Перемещен временно)
403 Forbidden (Запрещено)
404 Not found (Файл не файден)
410 Gone (Больше не существует)

Пример 1

```
RewriteEngine on

RewriteRule ^dummy\.html$ http://www.google.com/ [R=301]
```

В данном примере реализовано следующее:
RewriteEngine on - включаем механизм mod_rewrite
RewriteRule ^dummy\.html$ http://www.google.com/ [R=301] - перенаправялем запросы к странице dummy.html на сайт Google, используя код HTTP ответа – 301(Moved Permanently (Перемещено окончательно)).

Если теперь открыть веб-браузер и посетить страницу dummy.html на данном сайте, произойдет перенаправление на сайт http://www.google.com.

Пример 2

```
RewriteEngine on

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

RewriteRule ^(.*)$ index.php?route=$1 [L,QSA]
```

Пример 3

```
RewriteEngine on
RewriteBase /img/
RewriteCond %{HTTP_REFERER}   !^$
RewriteCond %{HTTP_REFERER}   !^http://www\.yourdomain\.com.*
[NC]
RewriteRule .* - [F]
```

```
RewriteEngine on
RewriteBase /
RewriteCond %{HTTP_REFERER} !^http://www\.yourdomain\.com.* [NC]
RewriteCond %{HTTP_REFERER} !^$
RewriteRule \.(jpe?g|gif|png|css|swf)$ - [F]
```

Пример 4

```
RewriteEngine on

RewriteRule ^([a-z]+)/([0-9]*)/([0-9]*)/([0-9]*)/$
                                    /index.php?show=$1&year=$2&month=$3&day=$4
```

Пример 5

```
RewriteEngine on

RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3.*
RewriteRule ^foo\.html$ foo.NS.html [L]
RewriteCond %{HTTP_USER_AGENT} ^Lynx/.* [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/[12].*
RewriteRule ^foo\.html$ foo.20.html [L]
RewriteRule ^foo\.html$ foo.32.html [L]
```