Министерство науки и высшего образования РФ  
ФГАОУ ВПО

Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

**Реферат**

по дисциплине «Разработка клиент-серверных приложений»

на тему «Управление исполнением сценария php»

Выполнил:  
студент группы БПИ-20-2

Бобров М.А.

Проверил:   
Стучилин В.В.

​​​

Москва, 2022

Оглавление

[Введение: 3](#_Toc102920145)

[Команды 4](#_Toc102920146)

[Функции управления сценарием 4](#_Toc102920147)

[set\_time\_limit 4](#_Toc102920148)

[sleep 5](#_Toc102920149)

[usleep 5](#_Toc102920150)

[die 5](#_Toc102920151)

[exit 5](#_Toc102920152)

[assert 5](#_Toc102920153)

[assert\_options 6](#_Toc102920154)

[eval 7](#_Toc102920155)

[Статус подключений 8](#_Toc102920156)

[connection\_aborted 8](#_Toc102920157)

[connection\_status 8](#_Toc102920158)

[connection\_timeout 9](#_Toc102920159)

[ignore\_user\_abort 9](#_Toc102920160)

[register\_shutdown\_function 9](#_Toc102920161)

[Другие функции 9](#_Toc102920162)

[highlight\_file 10](#_Toc102920163)

[highlight\_string 11](#_Toc102920164)

[show\_source 11](#_Toc102920165)

[pack 11](#_Toc102920166)

[unpack 12](#_Toc102920167)

[iptcparse 13](#_Toc102920168)

[leak 13](#_Toc102920169)

[serialize 13](#_Toc102920170)

[unserialize 14](#_Toc102920171)

[uniqid 14](#_Toc102920172)

[Источники: 15](#_Toc102920173)

Введение:

PHP — это скриптовый, интерпретируемый язык программирования, основная задача которого — это «оживление» HTML страниц, так как обычные HTML-страницы статичны, а статичность (или неизменность) означает, что после того, как страницу создали и загрузили на сайт, при каждом обращении к этой странице браузер покажет её любому пользователю в неизменном виде. [1]

Но этого не всегда достаточно. Если использовать только HTML, то решить такие задачи не получится. Здесь-то нам и понадобится PHP. Он принимает входящий запрос от веб-сервера, выполняет сценарий и возвращает веб-серверу результат в виде готового HTML-код. Сервер отправляет этот результат в браузер пользователю, который, в свою очередь, отображает её пользователю. После этого видно свежий курс валют, погоду, и что угодно ещё. РНР позволяет изменять веб-страницу на сервере непосредственно перед тем, как она будет отправлена браузеру. Давайте разберёмся, как это работает. PHP умеет исполнять код — так называемые сценарии. В ходе исполнения PHP может изменить или динамически создать любой HTML-код, который и является результатом исполнения сценария. Затем сервер отправляет этот код браузеру. При этом браузеру не известно, как была сформирована данная страница — статично сверстана верстальщиком, или динамически создана при участии PHP. Это не важно, т. к. браузер всегда работает только с тем, что получил от сервера. Давайте запомним, что сценарий — это программа, которая находится на стороне сервера и запускается в ответ на запрос от браузера. Теперь мы можем понять, почему PHP называют препроцессором гипертекста. Когда браузер запрашивает страницу с адресом, который кончается на .php, веб-сервер обращается к PHP, и просит его выполнить сценарий из файла по этому адресу. [1]

При этом сценарии можно регулировать при помощи функций управления сценарием, которые помогают регулировать нагрузку на сервер, время их выполнения и т. д.

Команды

Функции управления сценарием

set\_time\_limit

Установка предельного времени выполнения сценария.

Синтаксис: void set\_time\_limit(int seconds).

При запуске сценария PHP запускает системный таймер, и если время (выделенное сценарию для выполнения) истекает, а сценарий еще не завершился, PHP принудительно завершает сценарий (генерируя фатальную ошибку исполнения). Это не допускает скопления большого количества сценариев, расходующих ресурсы сервера, но, по-видимому, "зависших" (например, если в них обнаружился бесконечный цикл или они пытаются дождаться подключения к не отвечающему серверу).

По умолчанию допустимое время исполнения сценария устанавливается в файле конфигурации параметром max\_execution\_time (обычно оно равно 30 с). Но для текущего сценария это время можно изменить вызовом данной функции, указав время в секундах в ее аргументе. Если указывается значение 0, то тогда временное ограничение снимается.

Отсчет времени начинается от момента вызова функции. Например, если сценарий уже выполнялся в течении 15 секунд, а затем вызывается функция set\_time\_limit(20), то общее максимальное время исполнения сценария становится равным 35 секундам. Если сценарий выполняется в безопасном режиме (с установленным параметром safe mode), то тогда вызов этой функции игнорируется и используется значение из файла конфигурации. [1]

sleep

Задержка выполнения сценария.

Синтаксис: void sleep(int seconds);.

Функция sleep() выполняет задержку выполнения сценария в секундах (seconds). [2]

usleep

Задержка выполнения сценария в микросекундах.

Синтаксис: void usleep(int micro\_seconds);

Задержка выполнения сценария в микросекундах (micro\_seconds). Эта функция не работает в Windows. [2]

die

Вывод сообщения и завершение текущего сценария.

Синтаксис: void die(string message);.

Эта функция выводит сообщение и прекращает выполнение текущего скрипта. Не возвращает значение.

Пример:

<?php

$filename = '/path/to/data-file';

$file = fopen($filename, 'r') or die "unable to open file ($filename)";

?> [2]

exit

Завершает текущий сценарий.

Синтаксис: void exit(void);.

Эта функция завершает текущий сценарий. Не возвращает значение. [2]

assert

Проверка истинности значения.

Синтаксис: int assert(string|bool assertion);.

В качестве аргумента функции может быть указано значение или строка, содержащая код PHP (как в функции eval()). Функция проверяет, является ли значение (или выражение) равным false, и, если это так, выполняет определенные действия. Поведение функции определяется установками в файле конфигурации или при вызове функции assert\_options().

Обычно эта функция используется исключительно в целях отладки, для проверки тех значений, которые всегда должны быть истинны (например: подключение модуля, свободное пространство на диске и т.д.). В целом же выполнение сценария не должно зависеть от таких проверок, а использовать обычные проверки возвращаемых функциями значений.

Пример:

<?php

function handler()

{

echo "n\* Failed \* n";

}

assert("$a='1';");

echo "a: $a n";

assert(0);

// завершать сценарий

echo assert\_options(ASSERT\_BAIL, 1);

// вызвать обработчик

assert\_options(ASSERT\_CALLBACK, "handler");

// не выдавать сообщений PHP

@assert(--$a);

// эта строка не будет выполнена

echo "n ... n";

?>

Приведенный пример выведет:

a: 1Warning: Assertion failed in file.php on line 200\* Failed \* [2]

assert\_options

Определение параметров assert.

Синтаксис: mixed assert\_options(int parameter [, mixed value]).

Эта функция позволяет определить поведение конструкции assert().

Возвращается предыдущее значение параметра (или значение false при ошибке), указанного в первом аргументе одной из следующих констант:

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | ini-параметр | По умолчанию | Описание |
| ASSERT\_ACTIVE | assert.active | 1 | Разрешить указание кода в assert() |
| ASSERT\_WARNING | assert.warning | 1 | Выдавать предупреждение PHP |
| ASSERT\_BAIL | assert.bail | 0 | Завершать выполнение, если «ложь» |
| ASSERT\_QUIET\_EVAL | assert.quiet\_eval | 0 | Не выдавать сообщение |
| ASSERT\_CALLBACK | assert.callback | (null) | Установить функцию в качестве обработчика «ложных» assert() |

Если значение необходимо переопределить, его указывают во втором аргументе. [3]

eval

Производит выполнение строки содержащей PHP код.

Синтаксис: void eval(string code\_str);.

Функция eval() производит выполнение строки, заданной в code\_str содержащей PHP код. Кстати, это может пригодиться для сохранения кода в текстовом поле базы данных для более позднего выполнения. Не забывайте, что указанный в строке код должен быть синтаксически правильным (например, должны присутствовать точки с запятой после каждой команды и т.п.), в противном случае сценарий будет завершен в этой строке с ошибкой. Учитывайте также, что те значения переменных, которые будут установлены в данной строке, будут использоваться в оставшейся части сценария.

При изменении переменных значений в eval() эти переменные будут изменены и в основных данных.

Если в строке указан оператор return, то тогда выполнение указанного кода будет досрочно завершено и возвращенное значение можно будет получить как значение, возвращаемое самой функции.

<?php

$string = 'cup';

$name = 'coffee';

$str = 'This is a $string with my $name in it.';

echo $str;

eval( "$str = "$str";" );

echo $str;

?>

Результатом выполнения этого кода будет:

This is a $string with my $name in it.This is a cup with my coffee in it. [2]

Статус подключений

Внутренне PHP имеет три статуса подключения:

0 - NORMAL; 1 - ABORTED (прервано пользователем); 2 - TIMEOUT (истекло время ожидания ответа).

При нормальном выполнении сценария активно состояние NORMAL. Если во время загрузки страницы пользователь нажал кнопку STOP, активным становится состояние ABORTED. Если сценарий выполняется дольше отведенного ему времени, устанавливается флаг состояния TIMEOUT. Возможно определить, как должен вести себя сценарий в зависимости от этих условий.

Если требуется, чтобы сценарий продолжал свое выполнение при разрыве соединения пользователем, нужно установить в файле конфигурации значение параметра ignore\_user\_abort = 1 (это также можно сделать в файлах конфигурации Apache). Можно также воспользоваться функцией ignore\_user\_abort(). В противном случае, сценарий завершается.

Чтобы игнорировать завершение сценария таймером, необходимо использовать функцию set\_time\_limit().

Если функцией register\_shutdown\_function() была установлена функция "запускаемая при завершении сценария", то, вне зависимости от статуса подключения, она будет исполнена перед тем, как сценарий завершится. И в "завершающей" функции можно будет выяснить (с помощью функции: connection\_aborted(), connection\_timeout() и connection\_status()), был ли сценарий завершен нормально или досрочно. [3]

connection\_aborted

Определения разрыва подключения пользователем.

Синтаксис: int connection\_aborted(void);.

Функция connection\_aborted() возвращает true, если подключение было разорвано пользователем. [2]

connection\_status

Определения статуса подключения.

Синтаксис: int connection\_status(void);.

Возвращает значение битового поля, позволяющее выяснить в "завершающей" функции, был ли сценарий завершен досрочно и причину этого. Например, если возвращается 3 (ABORTED | TIMEOUT), то это означает, что время выполнения истекло, а также то, что пользователь отказался от загрузки страницы.

Если возвращается 0 (то есть значение NORMAL), то это означает, что выполнение сценария не было прервано. [2]

connection\_timeout

Определения наступления тайм-аута.

Синтаксис: int connection\_timeout(void);.

Возвращает true, если время выполнения сценария истекло. [2]

ignore\_user\_abort

Прерывание сценария при разрыве подключения.

Синтаксис: int ignore\_user\_abort([int setting]);.

Аргументом setting можно указать, необходимо ли досрочно завершать выполнение сценария, если связь с клиентом потеряна. Если аргумент не указан, то возвращается текущая установка. [2]

register\_shutdown\_function

Устанавливает функцию, которая будет выполнена при завершении.

Синтаксис: int register\_shutdown\_function(string func);.

Регистрирует функцию с именем func в качестве функции, запускаемой после завершения сценария.

Заметьте: так как после завершения функции никакие средства вывода недоступны, это делает для функции, зарегистрированной в качестве "завершающей", недоступными обычные средства отладки, такие команды как print или echo. [2]

Другие функции

**get\_browser**

Определение возможностей браузера.

Синтаксис: object get\_browser([string user\_agent]);.

Возвращаемая информация извлекается из файла browscap.ini. Для определения браузера используется значение переменной $HTTP\_USER\_AGENT или значение, содержащееся в аргументе user\_agent. Информация возвращается в виде свойств объекта и отражает возможности клиентского браузера (например, версию, поддерживает ли он javascript или cookies).

Пример:

<?php

function list\_array($array)

{

while (list ($key, $val) == each ($array))

{

$str .= "<b>$key:</b> $val<br>n";

}

return $str;

}

echo "$HTTP\_USER\_AGENT<hr>";

$browser = get\_browser();

echo list\_array ((array) $browser);

?>

Пример содержимого возможного вывода:

Mozilla/4.5 [en] (X11: Linux 2.2.9 i586)<hr><b>browser\_name\_pattern:</b>Mozilla/4.5.\*<br><b>parent:</b>Netscape<br><b>platform:</b>Unknown<br><b>majorver:</b>4<br><b>minorver:</b>5<br><b>browser:</b>Netscape<br><b>version:</b>4<br><b>frames:</b>1<br><b>tables:</b>1<br><b>cookies:</b>1<br><b>backgroundsounds:</b> <br><b>vbscript:</b> <br><b>javascript:</b>1<br><b>javaapplets:</b>1<br><b>activexcontrols:</b> <br><b>beta:</b> <br><b>crawler:</b> <br><b>authenticodeupdate:</b> <br><b>msn:</b> <br>

Для того чтобы функция могла функционировать, следует правильно указать месторасположение файла browscan.ini в файле конфигурации. [2]

highlight\_file

Вывод содержимого файла с цветовой разметкой.

Синтаксис: boolean highlight\_file(string filename);.

Имя или путь файла указывается в аргументе. Цвета выделения синтаксиса определяются в файле конфигурации PHP. Возвращает true или false при ошибке.

Например, чтобы заставить сервер Apache при получении запроса с URL, содержащего значение вида "http://имя.сервера/source/путь/к/файлу.php", выводит листинг файла "http://имя.сервера/source/путь/к/файлу.php", сделайте следующее. Добавьте в файл httpd.conf следующий фрагмент:

# Используем директиву "ForceType" чтобы указать,# что значение source в URL - не каталог, а имя сценария PHP<Location /source> ForceType application/x-httpd-php</Location>

Создайте в корневом web-каталоге следующий файл с именем source:

<HTML><HEAD>

<TITLE>Source Display</TITLE>

</HEAD>

<BODY bgcolor=#FFEEDD>

<?php

$script = getenv ("PATH\_TRANSLATED");

if(!$script) {

echo "<BR><B>ERROR: Укажите имя сценария</B><BR>";

} else {

if(ereg("(.php|.inc)$",$script)) {

echo "<H!>Листинг файла: $PATH\_INFO</H1>n<hr>n";

if(!@highlight\_file($script))

echo "Ошибка вывода файла";

} else {

echo "<H1>ERROR: Показываются только листинги PHP файлов </H1>";

}

}

echo "<HR>Распечатано: ".date("Y/M/d H:i:s",time());

?>

</BODY>

</HTML> [2]

highlight\_string

Выделение строки цветом.

Синтаксис: void highlight\_string(string str);

Функция действует подобно highlight\_file(), но использует не содержимое файла, а указанной строки. [2]

show\_source

Синоним функции highlight\_file.

Синтаксис: boolean show\_source(string str); [2]

pack

Пакетирование данных в двоичную строку.

Синтаксис: string pack(string format [,mixed $args, ...]);

Функция pack() упаковывает заданные аргументы в бинарную строку, которая затем и возвращается. Формат параметров, а также их количество, задается при помощи строки $format, которая представляет собой набор однобуквенных спецификаторов форматирования - наподобие тех, которые указываются в sprintf(), но только без знака %. После каждого спецификатора может стоять число, которое отмечает, сколько информации будет обработано данным спецификатором. А именно, для форматов a,A,h и H число задает, какое количество символов будет помещено в бинарную строку из тех, что находится в очередном параметре-строке при вызове функции (то есть, определяет размер поля для вывода строки). В случае @ оно определяет абсолютную позицию, в которую будут помещены следующие данные. Для всех остальных спецификаторов следующие за ними числа задают количество аргументов, на которые распространяется действие данного формата. Вместо числа можно указать \*, в этом случае подразумевается, что спецификатор действует на все оставшиеся данные.

Вот полный список спецификаторов формата:

a - строка, свободные места в поле заполняются символом с кодом 0;

A - строка, свободные места заполняются пробелами;

h - шестнадцатеричная строка, младшие разряды в начале;

H - шестнадцатеричная строка, старшие разряды в начале;

c - знаковый байт (символ);

C - беззнаковый байт;

s - знаковое короткое целое (16 битов порядок байтов определяется архитектурой процессора);

S - беззнаковое короткое число;

n - беззнаковое целое (16 битов, старшие разряды в конце);

v - беззнаковое целое (16 битов, младшие разряды в конце);

i - знаковое целое (размер и порядок байтов определяется архитектурой);

I - беззнаковое целое;

l - знаковое длинное целое (32 бита, порядок знаков определяется архитектурой);

L - беззнаковое длинное целое;

N - беззнаковое длинное целое (32 бита, старшие разряды в конце);

V - беззнаковое целое (32 бита, младшие разряды в конце);

f - число с плавающей точкой (зависит от архитектуры);

d - число с плавающей точкой двойной точности (зависит от архитектуры);

x - символ с нулевым кодом;

X - возврат назад на 1 байт;

@ - заполнение нулевым кодом до заданной абсолютной позиции.

// Целое, целое, все остальное - символы

$bindata = pack("nvc\*", 0x1234, 0x5678, 65, 66);

После выполнения приведенного кода в строке $bindata будет содержаться 6 байтов в такой последовательности:0x12, 0x34, 0x78, 0x56, 0x41, 0x42 (в шестнадцатеричной системе счисления). [3]

unpack

Распаковывает данные из двоичной строки.

Синтаксис: array unpack(string format, string data);

Распаковывает данные из двоичной строки в массив согласно формату. Возвращает массив, содержащий распакованные элементы. $array = unpack("c2chars/nint", $binarydata); Возникающий в результате массив будет содержать "chars1", "chars2" и "int". [2]

iptcparse

Анализирует двоичный IPTC блок на одиночные тэги.

Синтаксис: array iptcparse(string iptcblock);

Эта функция анализирует двоичный блок IPTC на одиночные теги. Возвращает массив, использующий tagmarker как индекс и значение как значение. Возвращает false при ошибке или если никаких IPTC данных не было найдено. [2]

leak

Имитация утечки памяти.

Синтаксис: void leak(int bytes);

leak() отсекает определенный объем памяти.

Это полезно при отладке диспетчера памяти, который автоматически очищает "отсеченную" память при выполнении запроса. Размер блока памяти указывается в байтах аргументом bytes. [2]

serialize

Генерирует удобохраниемое представление значения.

Синтаксис: string serialize(mixed value);

serialize() возвращает строку состоящую из потока байтов при представлении значения value, которое может где-нибудь сохранено. Это полезно для сохранения или передачи значений PHP без потери их типа и структуры.

Пример:

//$session\_data содержит многомерный массив с информацией о сессии

//текущего пользователя. Мы используем serialize() для сохранения

//этого в базе данных в конце запроса.

$conn = odbc\_connect("webdb", "php", "chicken");

$stmt = odbc\_prepare($conn,

"UPDATE sessions SET data = ? WHERE id = ?");

$sqldata = array(serialize($session\_data),

$PHP\_AUTH\_USER);

if (!odbc\_execute($stmt, &$sqldata)) {

$stmt = odbc\_prepare($conn, INSERT INTO sessions (id, data) VALUES(?, ?)");

if (!odbc\_execute($stmt, &$sqldata)) {

/\* Что-то сделано неправильно. \*/

}

} [2]

unserialize

Создает PHP значение из сохраненного представления.

Синтаксис: mixed unserialize(string str);

unserialize() берет одно сохраненное значение и преобразует обратно в PHP значение. Возвращает преобразованное значение, и может иметь тип: integer, double, string, array или object. Если был преобразован object, то методы не восстановятся.

Пример:

// Здесь мы используем unserialize() для загрузки данных о сессии из базы данных

// в $session\_data. Этот пример дополняет описанный в месте

// с serialize() .

$conn = odbc\_connect("webdb", "php", "chicken");

$stmt = odbc\_prepare($conn, "SELECT data FROM sessions WHERE id = ?");

$sqldata = array($PHP\_AUTH\_USER);

if (!odbc\_execute($stmt, &$sqldata) || !odbc\_fetch\_into($stmt, &$tmp)) {

// Если сбой запуска или выборки ,

// то инициализируем массив

$session\_data = array();

} else {

// Мы должны иметь представление в $tmp[0].

$session\_data = unserialize($tmp[0]);

if (!is\_array($session\_data)) {

// Что-то неправильно, инициализируем массив

$session\_data = array();

}

} [2]

uniqid

Генерирует уникальный идентификатор.

Синтаксис: int uniqid(string prefix [, boolean lcg]);

Функция uniqid() возвращает уникальный идентификатор, основанный на текущем времени в микросекундах и имеющий префикс prefix.Префикс может быть полезен, например, если Вы генерируете идентификаторы одновременно на отдельных хостах, которые, могли бы случилось, генерировали идентификатор в одной и той же микросекунде. Префикс может быть длиной до 114 символов.Если в качестве его значения передается пустая строка, то длина сгенерированного идентификатора будет 13 символов (при lcg=true - 23 символа). Если указан необязательный аргумент lcg со значением true, к концу идентификатора будет добавляться "комбинированный хеш энтропии LCG", делающий его значение более уникальным. Принято также обрабатывать полученное значение криптографическими методами (например, это часто делается в идентификаторах сессий).

// без случайной части

$token = md5(uniqid(""));

// посложнее

$better\_token = md5(uniqid(rnad()));

Эти строки генерируют 32 байта (128-битное шестнадцатеричное число): они обладают максимальной уникальностью, которая только может потребоваться. [2]

Источники:

1: <https://htmlacademy.ru/tutorial/php/basics>

2: <http://programming-lang.com/ru/comp_programming/cat/1/j150.html>

3: <http://php.info-kna.ru/part/12/>