

## Тема 6.6

«Использование cookies и сессий.  
Переадресация, повторные запросы  
страницы»

# Вступление

Часто в процессе выполнения тех или иных операций возникает необходимость сохранить какие-то данные для их передачи между отдельными скриптами. В этом нам помогут cookies и сессии.

Также в этой теме мы рассмотрим два простых и эффективных способа переадресации и «самообновления» (повторного запроса) страниц.

Итак...

# Cookies – определение

**Куки** (слово не склоняется; **cookie**, «печенье») – небольшой фрагмент данных, созданный веб-сервером и хранимый на компьютере пользователя в виде файла, который веб-клиент (браузер) каждый раз пересылает веб-серверу в HTTP-запросе при открытии страницы сайта, «установившего куки».

Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- **аутентификации** пользователя (Видели галочки «запомнить меня» при логине? Это оно!);
- **хранения** персональных предпочтений и настроек;
- отслеживания состояния **сессии** доступа пользователя;
- ведения **статистики** о пользователях.

Большинство современных браузеров позволяют пользователям выбрать – принимать куки или нет, но их отключение делает невозможной работу с некоторыми сайтами.

# Cookies – принцип работы

Основной принцип работы с куки легко понятен из рисунка:



# Cookies – принцип работы

Теперь чуть-чуть подробнее...

Запрашивая страницу, браузер отправляет веб-серверу короткий текст с HTTP-запросом. Например, для доступа к странице `http://www.example.org/index.html`, браузер отправляет на сервер `www.example.org` следующий запрос:

```
GET /index.html HTTP/1.1
```

```
Host: www.example.org
```

Сервер отвечает, отправляя запрашиваемую страницу вместе с текстом, содержащим HTTP-ответ. Там может содержаться указание браузеру сохранить куки:

```
HTTP/1.1 200 OK
```

```
Content-type: text/html
```

```
Set-Cookie: name=value
```

Строка Set-cookie отправляется лишь тогда, когда сервер желает, чтобы браузер сохранил куки. В этом случае, если куки поддерживаются браузером и их приём включён, браузер запоминает строку `name=value` и отправляет её обратно серверу с каждым последующим запросом.

# Cookies – принцип работы

Например, при запросе страницы <http://www.example.org/сpec.html> браузер пошлёт серверу [www.example.org](http://www.example.org) следующий запрос:

GET /сpec.html HTTP/1.1

Host: [www.example.org](http://www.example.org)

Cookie: name=value Accept: \*/\*

Таким образом, сервер узнаёт, что этот запрос связан с предыдущим. Сервер отвечает, отправляя запрашиваемую страницу и, возможно, добавив новые куки. Значение куки может быть изменено сервером путём отправления новых строк

Set-Cookie: name=newvalue.

После этого браузер заменяет старое куки с тем же name на новую строку. Строка Set-Cookie, как правило, добавляется к HTTP-ответу не самим HTTP-сервером, а CGI-программой, работающей вместе с ним (например, PHP).

Куки также могут устанавливаться программами на языках типа [JavaScript](#), встроенными в текст страниц, или аналогичными скриптами, работающими в браузере. В [JavaScript](#) для этого используется объект document.cookie. Например, document.cookie = "temperature=20" создаст куки под именем «temperature» и значением «20».

# Атрибуты cookies

Кроме пары **имя=значение**, куки может содержать срок действия, путь и доменное имя.

RFC 2965 также предусматривает, что куки должны обязательно иметь **номер версии**, но это используется редко.

Эти атрибуты должны идти после пары `name=newvalue` и разделяться точкой с запятой.

Например:

**Set-Cookie:**     **name=newvalue;**     **expires=date;**     **path=/;**  
**domain=.example.org.**

Это говорит браузеру, что куки должны быть отправлены обратно на сервер при запросах URL для указанного домена и пути. Если они не указаны, используются домен и путь запрошенной страницы.

Фактически, куки определяются тройкой параметров имя-домен-путь (оригинальная спецификация Netscape учитывала только пару имя-путь). Иными словами, куки с разными путями или доменами являются разными куки, даже если имеют одинаковые имена.

# Атрибуты cookies

Соответственно, куки меняет значение на новое, только если новое куки имеет те же имя, путь и домен.

Дата истечения указывает браузеру, когда удалить куки. Если срок истечения не указан, куки удаляется по окончании пользовательского сеанса, то есть с закрытием браузера.

Если же указана дата истечения срока хранения, куки становится постоянной до указанной даты.

Дата истечения указывается в формате  
«Нед, ДД-Мес-ГГГГ ЧЧ:ММ:СС GMT».

Например:

Set-Cookie: RMID=732423sdfs73242; expires=Fri, 31-Dec-2010  
23:59:59 GMT; path=/; domain=.example.net



# «Срок хранения» cookies

Срок хранения куки **истекает** в следующих случаях:

- В **конце сессии** (например, когда браузер закрывается), если куки не являются постоянными.
- Дата истечения была указана и **срок хранения вышел**.
- Дата истечения срока хранения изменена сервером или скриптом на **уже прошедшую дату** (позволяет серверу или скрипту удалённо стирать куки).
- Браузер **удалил куки** по запросу пользователя.

Заметим, что сервер не может узнать, когда истекают сроки хранения куки, поскольку браузер не отправляет на сервер эту информацию.

# Аутентификация через cookies

Куки могут использоваться сервером для опознавания ранее аутентифицированных пользователей. Это происходит следующим образом:

1. Пользователь вводит имя пользователя и пароль в текстовых полях страницы входа и отправляет их на сервер.
2. Сервер получает имя пользователя и пароль, проверяет их и, при их правильности, отправляет страницу успешного входа, прикрепив куки с неким идентификатором.
3. Эта куки может быть действительна как для текущей сессии браузера, так и быть настроена на длительное хранение.
4. Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет куки с идентификатором серверу.
5. Сервер проверяет идентификатор по своей базе идентификаторов и, при наличии в базе такого идентификатора, «узнаёт» пользователя.

Недостаток этого метода заключается в том, что злоумышленник может украсть куки и сервер опознает его как легального пользователя. Для усложнения такой операции многие сервера проверяют также ip (или его часть) и информацию о пользовательском клиенте.

# Работа с cookies в PHP

`bool setcookie ( string $name [, string $value [, int $expire=0 [, string $path [, string $domain [, bool $secure=false [, bool $httponly=false ]]]]] )` – устанавливает куки.

Поскольку данная функция работает с заголовками HTTP-протокола, её вызов должен **предшествовать выводу чего бы то ни было в выходной поток** (это ограничение можно обойти с помощью буферизации вывода данных, но лучше не рисковать).

После того, как куки были установлены (**ЕСЛИ** они были установлены, т.е. это разрешено на стороне клиента), они доступны через суперглобальный массив `$_COOKIE`.

**Внимание! Куки доступны при СЛЕДУЮЩЕМ запросе от клиента к серверу, т.е. если вызвать setcookie(), а потом в процессе этого же выполнения скрипта пытаться получить к ним доступ через \$\_COOKIE – их в этом массиве НЕ БУДЕТ.**

# Работа с cookies в PHP

Все параметры `setcookie()` за исключением имени куки — необязательны.

`name` — имя переменной-куки, именно оно потом будет ключом элемента массива `$_COOKIE`.

`expire` — «срок годности», т.е. **дата-вермя**, после которых cookie становится недействительным. Пример установки куки на 30 дней: `time()+60*60*24*30`. Значение 0 в этом параметре функции `setcookie()` означает, что куки действительны только на время данной сессии. **Внимание!** «Срок годности» сравнивается со временем на компьютере пользователя, а не со временем на сервере. Учитывайте смещение часовых поясов!

`path` — путь, для которого куки актуальны. Установка этого параметра в «/» распространяет действие куки целиком на весь установивший их домен. Если указать здесь «/каталог/», действие куки будет ограничено этим каталогом и его подкаталогами. По умолчанию используется текущий путь, т.е. если куки установил скрипт `/dir1/1.php`, то значением `path` будет «/dir1/». **Внимание!** Если вы используете эмуляцию путей через `mod_rewrite`, значением `path` будет реальный путь к скрипту на сервере, поэтому и рекомендуется с помощью `mod_rewrite` делать «замену URL» на `/index.php` — тогда значение `path` будет просто «/».

# Работа с cookies в PHP

**domain** – доменное имя, при запросе страниц с которого куки будут отправляться на сервер. Чтобы сделать куки также доступными и на **поддоменах** этого домена, следует установить значение параметра **domain** равным «**.example.com**», т.е. с точкой в начале имени.

**secure** – указывает, что куки следует передавать **ТОЛЬКО** по HTTPS соединению. По умолчанию значение этого параметра – **FALSE**.

**httponly** – использовать куки только для передачи по **HTTP**, т.е. сделать их недоступными скриптовым языкам, выполняющимся на стороне клиента (JavaScript и т.п.). По умолчанию – **FALSE**.

Значение переменной-куки будет автоматически «**urlencoded**» (представлено в виде, допустимом для передачи через URL).

Функция **setcookie()** возвращает **TRUE**, если ей удалось модифицировать заголовки HTTP-ответа с целью установки куки и **FALSE** – в противном случае.

# Работа с cookies в PHP

Примеры установки куки:

```
setcookie("TestCookie", $value);  
setcookie("TestCookie", $value, time()+3600);  
setcookie("TestCookie", $value, time()+3600, "/dir1/",  
".example.com", 1);
```

Примеры удаления куки:

```
setcookie ("TestCookie", "", time() - 3600);  
setcookie ("TestCookie", "", time() - 3600, "/dir1/",  
".example.com", 1);
```

# Работа с cookies в PHP

`bool setrawcookie ( string name [, string value [, int expire [, string path [, string domain [, bool secure [, bool httponly]]]] ] )` – делает всё то же самое, что и `setcookie()`, но не производит автоматическое urlenco'дирование переменных.

`object http_parse_cookie ( [string cookie [, int flags [, array allowed_extras]] ] )` – разбирает строку заголовка HTTP-запроса и возвращает объект, в свойствах которого содержатся значения параметров куки:

Пример:

```
print_r(http_parse_cookie("foo=bar;          bar=baz;          path=/;
domain=example.com; comment=; secure", 0, array("comment")));
// stdClass Object ( [cookies] => Array ( [foo] => bar [bar]
=> baz ) [extras] => Array ( [comment] => ) [flags] => 16
[expires] => 0 [path] => / [domain] => example.com )
```

`string http_build_cookie ( array cookie )` – выполняет обратное преобразование: из объекта, возвращаемого `http_parse_cookie()`, формирует строку для передачи в заголовке HTTP-запроса.

# Сессии – определение

**Сессии** – «идентифицированные соединения», «сеансы пользователя», т.е. весь процесс работы пользователя с некоторым сайтом, во время которого РНР может определять, что запросы продолжают поступать от «того же самого запомненного пользователя».

Этот механизм необходим потому, что протокол HTTP сам по себе не является «**СЕССИОННЫМ**»: после выполнения пары запрос-ответ, сервер закрывает соединение и навсегда забывает о пославшем запрос клиенте. Механизм сессий позволяет «**НАПОМНИТЬ**» серверу, что это – «тот же самый пользователь».

Использование сессий очень удобно для **хранения данных** и передачи их **между разными скриптами** без передачи их через формы или ссылки, т.е. данные **не будут передаваться по каналам связи**. Фактически, мы можем использовать сессии для всего того же, что и куки, учитывая один важный момент: данные в сессии, как правило, хранятся очень недолго (по умолчанию – не более часа) в отличие от данных куки, которые могут храниться месяцами.



# Сессии – настройки PHP

Здесь мы рассмотрим самые важные параметры настройки PHP, относящиеся к работе с сессиями:

```
session.save_path = "/tmp"
```

Указывает каталог для хранения файлов сессии.

```
session.use_cookies = 1
```

Использовать куки для передачи идентификатора сессии.

```
session.name = PHPSESSID
```

Имя переменной в куки или GET-запросе, содержащей идентификатор сессии.

```
session.auto_start = 0
```

Автоматически запускать механизм сессий для всех скриптов.

# Сессии – настройки PHP

`session.cookie_lifetime = 0`

Время жизни идентификатора сессии в куки.

`session.cookie_path = /`

Путь, для которого актуален идентификатор сессии в куки.

`session.cookie_domain =`

Домен, для которого актуален идентификатор сессии в куки.

`session.cookie_httponly = 0`

Устанавливать ли параметр `httponly` для идентификатора сессии в куки.

`session.bug_compat_42 = 0`

`session.bug_compat_warn = 1`

Запретить использовать устаревший механизм `session_register()` и вывести предупреждение.

# Сессии – настройки PHP

```
session.use_trans_sid = 0
```

Использовать ли модификацию URL и форм в отправляемом файле с целью обеспечения работоспособности сессий при отключённых у пользователя куки.

```
url_rewriter.tags =
```

```
"a=href,area=href,frame=src,input=src,form=fakeentry"
```

Если предыдущий параметр выставлен в 1, какие теги модифицировать. В формы будет добавлено скрытое поле, а в URL'ы будет добавлена передаваемая методом GET переменная с именем session.name.

Для управления настройками сессий относительно их взаимодействия с куки, можно использовать функции

```
session_get_cookie_params ( void )
```

и

```
void session_set_cookie_params ( int lifetime [, string path [,  
string domain [, bool secure [, bool httponly]]]] )
```

# Сессии – использование в PHP

`string session_name ( [string name] )` – возвращает или изменяет (если задан параметр) для данной сессии значение `session.name` (см. предыдущие слайды). Для изменения (с параметром) должна быть вызвана **ДО** вызова `session_start()`. Не работает на изменение (вызов с параметром), если сессии начинаются автоматически.

`session_start()` – запускает механизм сессий для скрипта, в котором вызвана. Как и `setcookie()` – должна быть вызвана **ДО** вывода чего бы то ни было в выходной поток. Это значит, что в вызывающем эту функцию скрипте также ничего не должно быть **ДО** начала PHP-кода.

`session_register($var1 [, $var2], ...)` – **НЕ ИСПОЛЬЗОВАТЬ!**  
Является deprecated с PHP 5.3 и будет УБРАНА в PHP 6.

# Сессии – использование в PHP

`bool session_set_save_handler ( callback open, callback close, callback read, callback write, callback destroy, callback gc )` – позволяет указать свои собственные функции-обработчики для всех событий, которые могут произойти с сессиями (открытие, закрытие, сохранение переменных и т.д.)

`bool session_destroy ( void )` – закрывает (завершает) сессию. Однако, не удаляет данные из массива `$_SESSION` и из куки (если это важно – нужно удалить эти данные вручную).

`string session_id ( [string id] )` – устанавливает (если передан параметр) или возвращает идентификатор сессии.

`bool session_regenerate_id ( [bool delete_old_session] )` – генерирует новый идентификатор сессии и уничтожает старую сессию, если передан параметр `delete_old_session` со значением **TRUE**.

# Сессии – пример

Самый простой пример, на основе которого, тем не менее, строятся и все сложные, таков.

В файле s1.php пишем код:

```
session_start();  
$a=25;  
$_SESSION["zzz"]=$a;  
$a=99;
```

Запускаем скрипт s1.php, а затем – s2.php

В файле s2.php пишем код:

```
session_start();  
echo $_SESSION["zzz"]; // 25
```

Обратите внимание, что просто изменение значения переменной `$a` не меняет её значения в массиве `$_SESSION`.

# Сессии – аутентификация

Классическим примером работы с сессиями является их использование для аутентификации пользователя.

Алгоритма работы таков:

1. Если мы не получили данные из формы, проверить наличие в сессии информации о том, что пользователь залогинен. Если да – продолжить работу, если нет – попросить залогиниться.
2. Если мы получили данные из формы, проверить пару логин/пароль и если они верны – занести в данные сессии информацию о том, что пользователь залогинен (а также, возможно, его ФИО и т.д.). Если логин/пароль неверен – попросить ввести их заново.
3. Если пользователь «разлогинивается» – закрыть сессию и удалить у него куки с информацией о сессии, а у себя удалить всё из массива `$_SESSION`.

И всё! ☺

# Переадресация и повторные запросы

Иногда в процессе «залогинивания/разлогинивания» пользователя имеет смысл перенаправить его на ту или иную страницу.

Это можно сделать двумя способами:

1. С использованием функции

`void header ( string string [, bool replace [, int http_response_code]] )` – устанавливает (или заменяет, если указан параметр `replace`) значение http-заголовка. Третий параметр позволяет указать «код ответа HTTP» (200, 404, 403 и т.п.)

Пример переадресации:

```
header("Location: http://www.google.com");
```

Переадресацию можно выполнять на любой произвольный URL.



# Переадресация и повторные запросы

2. С использованием тега meta refresh в заголовке HTML-документа. Пример:

```
<meta http-equiv="refresh" content="5;url=http://example.com/" />
```

Здесь содержимое content – это время в секундах, по истечении которого происходит переадресация, и URL, на которой должен перейти браузер.

Первый способ срабатывает **ЗНАЧИТЕЛЬНО** быстрее, даже если во втором способе поставить время задержки равное 0. Однако, в первом способе время задержки указать вообще нельзя, что иногда бывает неудобно, если надо показать пользователю какой-то текст, прежде чем перенаправить его на другую страницу.

# Переадресация и повторные запросы

Оба способа позволяют указать в качестве URL для переадресации **саму же страницу**, с которой пришёл такой заголовок. В первом случае это бессмысленно, т.к. приведёт к вечному циклу повторных мгновенных переадресаций (многие браузеры это отслеживают и блокируют), а во втором случае так удобно делать страницы, которые должны с той или иной целью самопроизвольно обновляться раз в столько-то секунд.

Подробное описание заголовков запроса и ответа по протоколу HTTP, а также кодов состояния см. здесь:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html> – заголовки

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> – коды состояния

[http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_headers](http://en.wikipedia.org/wiki/List_of_HTTP_headers) – кратко в Википедии