Введение в куки и сессии

Термин cookies (это множественное число, произносится как «кукис» или «куки») был введён в веб-программирование благодаря компании Netscape. В единственном числе часто говорят «кука».

Cookie — это небольшая именованная порция информации, которая хранится в каталоге браузера пользователя (а не на сервере), но которую сервер (а точнее, сценарий) волен в любой момент изменить. Кстати, сценарий также получает все cookies, которые сохранены на удаленном компьютере, при каждом своем запуске, так что он может в любой момент времени узнать, что у пользователя установлено. Самым удобным в cookies является то, что они могут храниться недели и годы до тех пор, пока их не обновит сервер или же пока не истечет срок их жизни (который тоже назначается сценарием при создании cookie). Таким образом, мы можем иметь cookies, срок жизни которых как всего несколько минут (или до того момента, пока не закроют браузер), так и несколько лет.

cookie связывается с доменом и может быть установлена и прочитана с помощью JavaScript, в виде строки, доступной через document.cookie.

Эта строка состоит из пар ключ=значение, которые перечисляются через точку с запятой с пробелом "; ".

Следовательно, чтобы прочитать cookie, достаточно разбить строку по "; ", и затем найти нужный ключ. Это можно делать либо через split и работу с массивом, либо через регулярное выражение.

- установка: https://kodaktor.ru/set_cookie
- чтение: https://kodaktor.ru/get_cookie

Поскольку нет прямых методов для установки конкретного значения в конкретный ключ, это по современным меркам неудобный вариант обращения, если нужно хранить клиентские данные между уходами со страницы. Более современный вариант – localStorage.

https://kodaktor.ru/vue_simple_tasker

Это локальная БД типа «ключ:значение»

Каждый браузер хранит свои cookies отдельно. То есть cookies, установленные при пользовании Chrome, не будут видны при работе в FireFox, и наоборот.

Каждому cookie сопоставлено время его жизни, которое хранится вместе с ним. Кроме этого, имеется также информация об имени сервера, установившего этот cookie, и URL каталога, в котором находился сценарий-хозяин в момент инициализации (за некоторыми исключениями).

Если при установке куки дату не указать, то она будет считаться «сессионной» и удаляется при закрытии браузера. Если дата в прошлом, то кука будет удалена.

localStorage хранит данные без срока, очищается только с помощью JavaScript, или очисткой кэша браузера / Locally Stored Data.

sessionStorage удаляет данные при закрытии браузера (не вкладки).

Зачем нужны имя сервера и каталог? Сценарию передаются только те cookies, у которых параметры с именем сервера и каталога совпадают с хостом и каталогом сценария соответственно (на самом деле каталог не должен совпадать полностью, он может являться подкаталогом того, который создан для хранения cookies). Так что совершенно невозможно получить доступ к "чужим" cookies — браузер просто не будет посылать их серверу.

Теоретически, любой ресурс, который загружает браузер, может поставить cookie.

Например:

- Если на странице есть <img src="http://mail.ru/counter.gif">, то вместе с картинкой в ответ сервер может прислать заголовки, устанавливающие cookie.
- Если на странице есть <iframe src="http://facebook.com/button.php">, то во-первых сервер может вместе с button.php прислать cookie, а во-вторых JS-код внутри ифрейма может записать в document.cookie

При этом cookie будут принадлежать тому домену, который их поставил. То есть, на mail.ru для первого случая, и на facebook.com во втором.

Такие cookie, которые не принадлежат основной странице, называются «сторонними» (3rd party) cookies. Не все браузеры их разрешают.

Серверная сторона может приказывать браузеру сохранить куки. Мы можем послать заголовок или сгенерировать метатэг.

<meta http-equiv="Set-Cookie" content="name=value; expires=дата; domain=имя_хоста; path=путь; secure"

>

Заголовок выглядит аналогично:

Set-Cookie: name=value; expires=дата; domain=имя_хоста; path=путь; secure

Получить cookies для серверного сценария легко: все они хранятся в переменной окружения HTTP_COOKIE в таком же формате, как и QUERY_STRING, только вместо символа амперсанда & используется точка с запятой ;. Например, если мы установили два cookies: cookie1=value1 и cookie2=value2, то в переменной окружения HTTP_COOKIE будет следующее:

cookie1=value1;cookie2=value2

Сценарий должен разобрать эту строку, распаковать ее и затем работать по своему усмотрению.

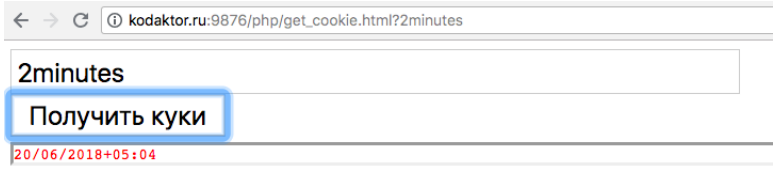
Рассмотрим серверную инструкцию, которая устанавливает куку сроком жизни в 2 минуты начиная от момента своего выполнения.

```
setcookie("2minutes", date('d/m/Y H:i'), time()+120);
```

- зайдите на http://kodaktor.ru:9876/php/get_cookie.html?2minutes
- нажмите кнопку и убедитесь что ничего не отображается
- перейдите по ссылке http://kodaktor.ru:9876/php/2minutes.php

это создаст куку со сроком жизни 2 минуты и гиперссылку на страницу для проверки (ту же самую, что выше).

- перейдите по гиперссылке незамедлительно и удостоверьтесь, что выводится дата и время.



Через 3 минуты обновите эту веб-страницу и снова нажмите кнопку. На этот раз значение куки выведено не будет...

В традиционном вебе (и PHP) переход с одной страницы на другую сопровождается запуском нового экземпляра скрипта, поэтому любые данные в «старом» сценарии, выдавшем предыдущую страницу, теряются. Сессии предоставляют механизм сохранения этих данных.

Это позволяет попросить пользователя один раз ввести логин и пароль, после чего проверить их и, если всё правильно, сохранить в куки некую информацию об успешном входе. При дальнейших попытках зайти на закрытые разделы сайта пользователю уже не нужно будет заново вводить логин и пароль, если срок жизни куки не исчерпан или не был произведён выход из сессии. Достаточно будет только проверять эту информацию.

Хотя потребность в хранении данных пользователя между запусками сценария возникает и помимо аутентификации. Например, если пользователь анонимно проходит опрос и заполняет форму, состоящую из нескольких страниц. Очевидно, то, что уже заполнено, должно сохраняться.

Как работают сессии? Для начала должен существовать механизм, который бы позволил PHP идентифицировать каждого пользователя, запустившего сценарий. То есть при следующем запуске PHP нужно однозначно определить, кто его запустил: тот же человек или другой. Делается это путем присвоения клиенту так называемого уникального идентификатора сессии, Session ID. Чтобы этот идентификатор был доступен при каждом запуске сценария, PHP помещает его в cookies браузера.

Теперь, зная идентификатор (дальше для краткости мы будем называть его SID), PHP может определить, в каком же файле на диске хранятся данные пользователя.

Как в сессии сохраняются данные? Для этого существует глобальный массив $_SESSION, который PHP обрабатывает особым образом. Поместив в него некоторые данные, мы можем быть уверены, что при следующем запуске сценария тем же пользователем массив $_SESSION получит **то же самое значение**, которое было у него при предыдущем завершении программы. Это произойдет потому, что при завершении сценария PHP автоматически сохраняет массив $_SESSION во временном хранилище, имя

которого хранится в SID.

Таким образом суперглобальный массив $_SESSION представляет собой набор данных, которые сохраняются между запусками сценария, в отличие от $_GET или $_POST.

Где же находится то промежуточное хранилище, которое использует PHP? Вообще говоря, это можно задать, написав соответствующие функции и зарегистрировав их как обработчики сессии. Но в PHP уже существуют обработчики по умолчанию, которые хранят данные в файлах (в системах UNIX для этого обычно используется каталог /tmp).

Прежде, чем работать с сессией, ее необходимо инициализировать. Делается это путем вызова специальной функции session_start().

Эта функция инициализирует механизм сессий для текущего пользователя, запустившего сценарий. По ходу инициализации она выполняет ряд действий:

- если посетитель запускает программу впервые, у него устанавливается cookies с уникальным идентификатором и создается временное хранилище, ассоциирован- ное с этим идентификатором;
- определяется, какое хранилище связано с текущим идентификатором пользователя;
- если в хранилище имеются какие-то данные, они помещаются в массив $_SESSION.

Функция возвращает true, если сессия успешно инициализирована, и false в противном случае. Начиная с PHP 7, функция принимает необязательный параметр $options, который содержит ассоциативный массив с параметрами механизма сессий. Если раньше их можно было установить только через php.ini, то сейчас большинство можно задать динамически.

```php
<?php ## Пример работы с сессиями.
  session_start();
  // Если на сайт только-только зашли, обнуляем счетчик.
  if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
  // Увеличиваем счетчик в сессии.
  $_SESSION['count'] = $_SESSION['count'] + 1;
?>
<h2>Счетчик</h2>
В текущей сессии работы с браузером Вы открыли эту страницу
<?= $_SESSION['count'] ?> раз(а).<br />
Закройте браузер, чтобы обнулить счетчик.<br />
<a href="<?= $_SERVER['SCRIPT_NAME'] ?>"
target="_blank">Открыть дочернее окно браузера</a>.
```

http://kodaktor.ru:9876/php/session_count.php

Откройте этот адрес в отдельном браузере и несколько раз обновите страницу. Счётчик будет увеличиваться. Можно закрыть эту вкладку и открыть другую, значение сохранится. После закрытия браузера (т.е. выхода из него как из программы, запущенной в данной операционной системе) счётчик будет сброшен. Проверьте это!

https://www.youtube.com/watch?v=LeOodGbJnKg

Запустите и проверьте работу приложения, состоящего из частей, приведённых в репозитории

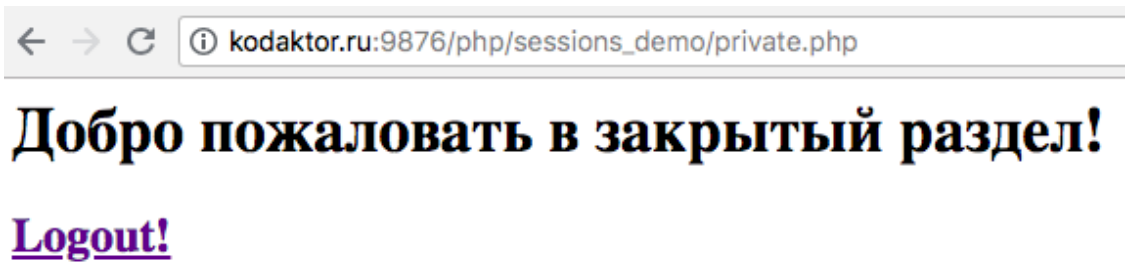
https://github.com/GossJS/php_starters1/tree/sessions_demo

Вот действующая демонстрация:

http://kodaktor.ru:9876/php/sessions_demo/private.php

Авторизоваться!

Введите пароль и сможете снова перейти на указанный выше адрес, но на этот раз содержимое приватного раздела окажется другим:



Совершенно аналогично работает приложение, подготовленное на платформе Node.js: https://lr.kodaktor.ru/public

Добавьте хранения логина и пусть в вашем случае хранятся два логина и два пароля в файле JSON. Можно также использовать http://kodaktor.ru/j/users как простейшую базу данных логинов/паролей. Пусть также будут два закрытых раздела со ссылками друг на друга и на логаут.