1.5. Сессии и cookie в PHP

Как уже было сказано ранее, протокол HTTP не может передать состояние и параметры. Поэтому каждый отдельный HTTP-запрос независим от других, сервер обрабатывает каждый такой запрос отдельно. Частично, эту проблему решает передача методами GET и POST. Однако, при разработке веб-проектов часто возникает необходимость передать какие-то данные между файлами (скриптами). Типичная ситуация – представьте себе сайт, на котором есть форма авторизации пользователя. Естественно, вводя свои данные одни раз, пользователь не ожидает, что при переходе на другие страницы данного ресурса ему снова придется вводить логин и пароль. Однако все вышеописанные средства для решения этой задачи не подошли бы.

Опишем данную проблему более формально. Пусть имеется файл first.php, в котором мы задаем значение какой-либо переменной.

Файл **first.php**

```
<?php
  $a = "Меня задали на first.php";
?>
<html>
     <body>
          <?php
          echo $a;
          ?>
     </body>
</html>
```

И предположим, что эту переменную нам необходимо передать в файл second.php. Но файл second.php ничего не знает о значении переменной $a.

Файл **second.php**

```
<html>
     <body>
<?php
  echo $a;
?>
     </body>
</html>
```

Именно из-за отсутствия поддержки передачи информации между скриптами, файл **second.php** выведет пустое значение переменной **$a**.

Решением данной проблемы является механизм **сессий** или **сеансов**. Условно говоря, сессией является сеанс связи конкретного лица, сидящего за конкретным компьютером (или цифровым устройством), который с помощью одной и той же программы-браузера посещает ресурсы удаленного сервера. Если пользователь закрыл браузер, то сеанс его связи будет прерван. Если тот же самый пользователь, с того же самого устройства заходит на те же ресурсы, но с другого браузера, то это будет новый сеанс (сессия). Формально в начале сеанса связи сервер определяет небольшую область под хранение данных конкретного сеанса. Способ хранения этих данных (в виде файлов или в виде записей в базе данных) определяется в настройках сервера.

Любой скрипт, который будет использовать переменные (данные) из сессий, должен содержать следующую строчку:

```
session_start();
```

Эта команда говорит серверу, что данная страница нуждается во всех переменных, которые связаны с данным пользователем (браузером). Сервер берёт эти переменные (из файла либо из БД) и делает их доступными.

Очень важно открыть сессию до того, как какие-либо данные будут посылаться пользователю; на практике это значит, что функцию **session_start()** желательно вызывать в самом начале страницы (до тега <html>), например так:

```
<?php
  session_start();
?>
<html>
 <head>
 </head>
```

После того как применен вызов функции **session_start()** и установлен сеанс, можно применить простой подход к чтению/записи сессионных переменных через суперглобальный массив **$_SESSION**. Еще раз подчеркнем, что сессионные значения хранятся на стороне сервера и хранятся до тех пор, пока сеанс связи не прервется (пользователь не закроет браузер или длительность сессии можно запрограммировать). При присвоении какого-либо значения любому полю массива **$_SESSION**, переменная с таким же именем автоматически регистрируется как переменная сессии. Этот массив доступен на всех страницах, использующих сессию.

Рассмотрим, как изменится код скриптов при решении задачи передачи значения переменной **$a** между скриптами **first.php** и **second.php**.

Файл **first.php**

```
<?php
     // открываем сессию
     session_start();
     // задаём значение переменной
     $a = "Меня задали на first.php";
     // регистрируем переменную с открытой сессией
     $_SESSION["a"] = 'Меня задали на first.php'
?>
<html>
     <body>
          Произошел старт сессии
          Перейдите по ссылке для просмотра <a
href="second.php">second.php</a>
     </body>
</html>
```

Файл **second.php**

```
<?php
  // открываем сессию
  session_start();
?>
<html>
     <body>
          <?php
          echo $_SESSION["a"];
          ?>
     </body>
</html>
```

В этом случае можно убедиться, что second.php выведет значение «Меня задали на first.php». При этом сессионный механизм работает, т.к. и в файле first.php и в файле second.php произошло подключение к сеансу через session_start().

Другие функции для работы с сессиями:

- unset($_SESSION['a']) – удаление значение переменной **a** из сессии;
- session_destroy() – сессия уничтожается (например, если пользователь покинул систему, нажав кнопку «Выход»);
- session_set_cookie_params(int lifetime [, string path [, string domain]]) – с помощью этой функции можно установить, как долго будет «жить» сессия, задав unix_timestamp, определяющий время «смерти» сессии. По умолчанию сессия «живёт» до тех пор, пока клиент не закроет окно браузера.

Рассмотрим классический пример применения сессии при авторизации пользователя. Простая авторизация предназначена для выяснения доступа пользователя к некоторым защищенным (засекреченным) страницам. Чаще всего для проверки пользователя используется пара «логин – пароль». Пусть есть пользователь с логином `maria` и паролем `123456` (не самая надежная пара, но для примера подойдет). На блок-схеме показано (рисунок 16), как работает алгоритм.

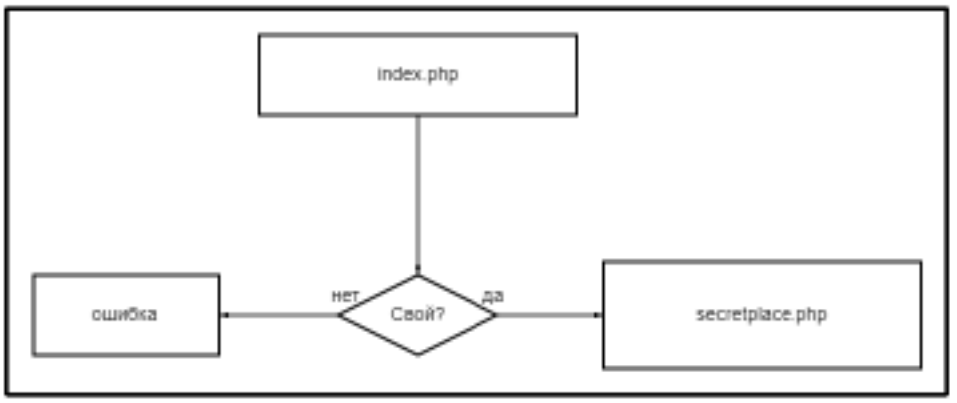

Рис. 16. Блок-схема авторизации пользователя

Файл **index.php**

```
<html>
     <head>
          <title>Введите пароль</title>
     </head>
     <body>
          <form action="authorize.php" method="post">
               Логин:<input type="text" name="username"><br>
               Пароль:<input type="password"
name="password"><br>
               <input type="submit" name="Submit">
          </form>
     </body>
</html>
```

Файл **authorize.php**

```
<?php
  // открываем сессию
  session_start();
  //получаем данные с формы
  $Submit      = $_POST["Submit"];
  $username    = $_POST["username"];
  $password    = $_POST["password"];
// данные были отправлены формой?
  if($Submit){
    // проверяем данные на правильность...
     if(($username=="maria")&&($password=="123456")){
          // запоминаем имя пользователя
          $_SESSION["logged_user"] = $username;
          // и переправляем его на <секретную> страницу...
          header("Location: secretplace.php");
          exit;
    }
  }
  // если что-то было не так, то пользователь получит сообще-
ние об ошибке.
?>
<html><body>
Вы ввели неверный пароль!
</body></html>
```

Рассмотрим код файла **secretplace.php**. Просто зайти на эту страницу нельзя: если имя пользователя не зарегистрировано, то перенаправляем его на страницу index.php для ввода логина и пароля.

Файл **secretplace.php**

```
<?php
  // открываем сессию
  session_start();
  $logged_user = $_SESSION["logged_user"];
  if(!isset($logged_user)){
    header("Location: index.php");
    exit;
  }
?>
<html>
 <body>
 Здравствуйте, <?php echo $logged_user; ?>, вы на секретной
странице<br>
 <a href="logout.php">Выйти</a>
 </body>
</html>
```

Также предусмотрим файл **logout.php**, который позволит выйти из сеанса текущего пользователя (сбросить авторизацию). При этом для чистоты эксперимента после уничтожения сессии перенаправим пользователя снова на секретную страницу. Если пользователь не авторизован, то уже скрипт **secretplace.php** перенаправит пользователя на **index.php**.

Файл **logout.php**

```
<?php
  session_start();
  //уничтожаем сессию
  session_destroy();
  //перенаправляем на секретную страницу,
  //если сессия сброшена, то произойдет перенаправление на
  //index.php
  header("location: secretplace.php");
 ?>
```

Увидеть файлы с данными сессии можно на локальном сервере в директории WebServers/tmp. Все файлы в этой папке имеют специальные идентификаторы, но если их отсортировать по дате, то можно легко найти последние изменённые файлы (рисунок 17).

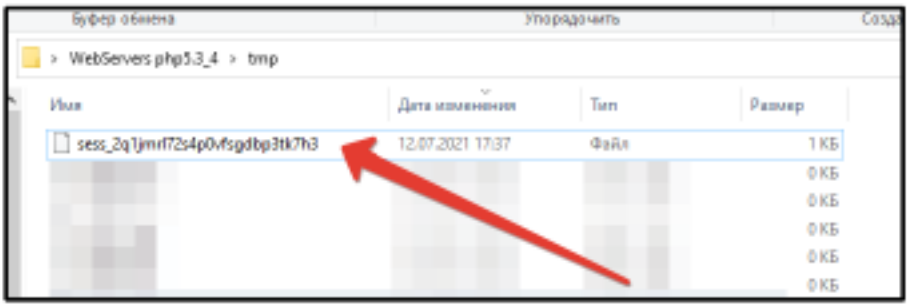

Рис. 17. Список файлов сессий

Содержимое этого файла можно легко посмотреть с помощью текстового редактора (рисунок 18).

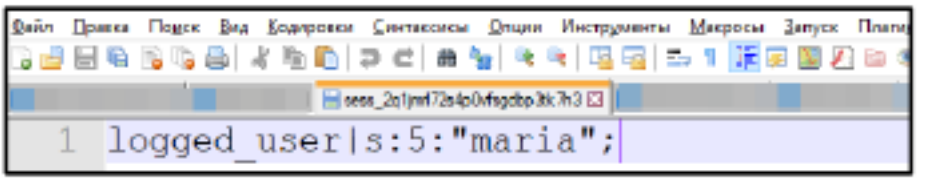

Рис. 18. Содержимое сессионного файла

Для отслеживания сессий часто применяются специальные **cookie-файлы**. Они хранятся на компьютере пользователя в виде небольшой текстовой информации. В каждом браузере своя директория для хранения cookie-файлов. В отличие от файлов сессий, cookie могут храниться дольше закрытия браузера. Более того, можно определять время хранения файлов cookie программно. Cookie-файл хранит данные в виде пар «имя – значение».

Cookie применяются для контроля за некоторыми фрагментами информации, к которой обращается пользователь при переходе от страницы к странице. При этом предполагается, что эта информация хранится и после того, как пользователь прервал сеанс связи с сайтом. Примером использования cookie может служить алгоритм голосования – с помощью cookie можно запретить пользователю принимать участие в голосовании более одного раза. Чуть позже мы разберем подробнее этот пример.

Для установки cookie-файлов используется функция setcookie().

```
Setcookie (имя_cookie, значение_cookie, вре-
мя_жизни_cookie_в_секундах);
```

Как и любой другой заголовок cookie должны передаваться **до того,** как будут выведены какие-либо другие данные скрипта (это ограничение протокола). Это значит, что в скрипте вызовы этой функции должны располагаться **до остального вывода**, включая вывод тегов <html> и <head>, а также пустые строки и пробельные символы.

Пример

```
setcookie ('my_cookie', 'Понедельник', time()+60*60*24*3);
```

Функция **time()** – возвращает количество секунд, прошедших с 1 января 1970 года. Обратите внимание, что использование записи 60*60*24*3 в качестве добавления секунд жизни cookie гораздо информативнее, чем вычислить это значение и записать 259 200. Так сразу понятно, что время жизни cookie трое суток.

Считывание переменных cookie может происходить с помощью суперглобального массива $_COOKIE (по аналогии с суперглобальным массивом $_SESSION).

Вернемся к примеру с голосованием. Одной из наиболее частных проблем при сборе сведений через формы голосования является то, что один и тот же пользователь может многократно проголосовать. В некоторых случаях это существенно влияет на результаты голосования, поэтому разработчику нужен механизм запрета голосования для пользователя, который уже ранее голосовал. Механизм сессий здесь не подходит, так как сессия уничтожается, как только пользователь закрыл браузер, и с точки зрения сервера – это будет новый респондент. А вот использование cookie здесь как раз пригодится. После того как пользователь первый раз проголосовал, в его cookie сохраняется небольшая информация. При попытке в следующий раз проголосовать скрипт сначала посмотрит, не сохранено ли что-то насчет голосования в cookie, если нет, то позволит проголосовать, если есть, то не позволит. Конечно, cookie-файл привязан к браузеру, и ничто не мешает этому же пользователю проголосовать через другой браузер. Но пользователь о таком скорее всего не знает, и в большинстве своем метод сработает.

Фрагмент алгоритма голосования

```
if (!(isset($_COOKIE['golos']))) {
  setcookie('golos', 'yes', time()+60*5);
  }
 else { echo "Вы уже голосовали";  }
```

В данном случае пользователь не может проголосовать в течение пяти минут.

Таким образом, сессии и cookie позволяют отслеживать и запоминать действия пользователей при взаимодействии с сайтом. Если нужно менять информацию пользователю в зависимости от его выбора или ранее сделанных действий (посещенных страниц, авторизации, голосования, отбора продуктов в корзину, заполнении форм обратной связи и др.), то можно применять сессии и cookie. Надо понимать, что по-

мимо удобства для пользователя, эти механизмы должны быть надежно защищены от несанкционированного доступа посторонних лиц.