Примеры применения PHP

- Теневые посылки
- Работа с файлами
  - Описание примера
  - Функция `date`
  - Функции работы с файлами

Теневые посылки

- **Теневые посылки** или **Ку́ки** (от англ. *cookie* — твердое шотландское печенье) — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в виде HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

Теневые посылки

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

- Cookie является решением одной из наследственных проблем HTTP протокола (HyperText Transfer Protocol). Эта проблема заключается в непостоянстве соединения между клиентом и сервером, т.е. для каждого документа (или файла) при передаче по HTTP протоколу посылается отдельный запрос.

Теневые посылки

- Используя cookie, можно эмулировать сессию по HTTP протоколу. Коротко принцип эмуляции сессии таков: на первом запросе выдается соотвествующее значение cookie, а при каждом последующем запросе это значение читается из переменной окружения HTTP_COOKIE и соответствующим образом обрабатывается.
- Простой пример: есть форма, где пользователю предлагается указать свое имя, из нее вызывается скрипт, прописывающий значение cookie в браузер пользователя.

Теневые посылки

- При каждом последующем заходе на основе анализа значения cookie из браузера пользователя на странице появляется либо именное приветствие (если есть установленное значение cookie), либо первоначальная форма с запросом имени пользователя (если значение cookie не установлено).
- Сценарии работы с теневой посылкой (cookie) предполагают получение данных формы, проверку наличия cookie, ее считывание и запись новых данных, если они изменились, и при необходимости отображение предыдущего содержимого теневой посылки.

Теневые посылки

- Зная имя теневой посылки и имена хранящихся в ней переменных, проверяется наличие значения переменной (value) в указанной теневой посылке (visit):
- **`if (isset ($_COOKIE ["visit"])) $value = $visit;`**
- Здесь приведена сокращенная форма обращения к глобальному ассоциативному массиву $HTTP_COOKIE_VARS[]. Новое значение переменной и срок действия теневой посылки устанавливается так:

Теневые посылки

- `setcookie("visit",$_POST[value],time()+365*24*60*60);`
- `header("location:index.php");`
- В примере использованы лишь три параметра теневой посылки. Первый – имя теневой посылки, второй – ее значение, третий – срок хранения (здесь – 1 год = 365 дней по 24 часа по 60 минут и 60 секунд). Если задать 0 или пропустить этот аргумент, срок действия cookie истечет с окончанием сессии (при закрытии броузера).

Теневые посылки

- Полный прототип функции setcookie выглядит так:
- **bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]]]);**
- Здесь **$path** и **$domain** позволяют задать путь и имя домена того сценария, который может прочитать значение этого cookie.

Теневые посылки

- По умолчанию только страницы, расположенные в том же каталоге или ниже в структуре подкаталогов того сервера, который установил cookie, могут прочитать ее значение. Это делается из соображений безопасности.
- Параметры **`$secure`** и **`$httponly`** так же не обязательны. **`$secure`** требует, чтобы значение cookie передавалось только на те Web-сервера, которые испольуют безопасный протокол соединения, такой как SSL.

Теневые посылки

- Параметры **`$secure`** и **`$httponly`** так же не обязательны. **`$secure`** требует, чтобы значение cookie передавалось только на те Web-сервера, которые испольуют безопасный протокол соединения, такой как SSL.
- Если **`$httponly=TRUE`**, то cookie будут доступны только через HTTP протокол, то есть не будут доступны скриптовым языкам, вроде JavaScript. Эта возможность была предложена в качестве меры, эффективно снижающей количество краж личных данных посредством XSS атак (несмотря на то, что поддерживается не всеми браузерами).

Теневые посылки

- Стоит однако же отметить, что вокруг этой возможности часто возникают споры о ее эффективности и целесообразности. Аргумент добавлен в PHP 5.2.0.
- Удалить cookie тоже очень просто, достаточно передать функции setcookie() имя cookie и PHP сделает все остальное:
- `<?php setcookie("visit"); ?>`
- Можно также передать в качестве параметра, определяющего "время жизни" cookie любое время из прошлого.

Теневые посылки

- `setcookie('visit', $value, time()-7*24*60*60);`
- // Неделя назад. Файл cookie будет удален
- В заключение нужно сделать еще одно замечание, касающееся использования cookie. В силу того, как организована обработка cookies в протоколе HTTP, необходимо установить значения всех cookie до вывода какого-либо текста. Если сделать наоборот, PHP выдаст предупреждение и значение cookie не будет послано.

Работа с файлами

- Существуют два основных способа хранения данных: в двумерных (обычных) файлах и в базах данных. В общем случае под двумерным файлом будем понимать простой текстовый файл. В рассматриваемом ниже примере заказы клиента записываются в текстовый файл, по одному заказу в каждой строке.
- Этот способ столь же прост, сколь и ограничен. Если приходится иметь дело с большим объемом информации, лучше воспользоваться базами данных. Однако, двумерные файлы находят достаточно широкое применение, поэтому необходимо владеть технологией их применения.

Описание примера

- Пример работы с файлами: http://217.71.139.95/~gun/samples/lab2_2.
- Форма ввода данных реализована в HTML и не представляет особой сложности.
- Обработка данных формы включает получение текущей даты, формирование строки для записи, открытие файла в режиме добавления, записи данных, закрытие файла и открытие на чтение с выдачей всех записей в браузер.

Функция date

- Функция **date()** принимает два аргумента, один из которых является необязательным. Первый аргумент представляет собой строку формата, а второй, необязательный, — метку времени UNIX. Если метка времени не указана, то функция **date()** обрабатывает текущую дату и время. Она возвращает отформатированную строку, содержащую дату. Типовой вызов функции выглядит так:

```
echo date("jS  F Y");
```

- Вывод этого выражения имеет вид "31th July 2001". Коды форматирования, используемые функцией **date()**, перечислены в таблице

Функция date

Код	Описание
a	Утро или время после полудня, с двумя строчными символами, "am" или "pm"
A	Утро или время после полудня, с двумя прописными символами, "AM" или "PM".
d	День месяца в виде двузначного числа с ведущим нулем. Диапазон значений — от "01" до "31".
D	День недели в виде трехбуквенной аббревиатуры. Диапазон значений — от "Mon" (понедельник) до "Sun" (воскресенье).

Функция date

Код	Описание
F	Месяц в полнотекстовом формате. Диапазон значений — от "January" до "December".
g	Часы в 12-часовом формате без ведущих нулей. Диапазон значений — от "1" до "12".
G	Часы в 24-часовом формате без ведущих нулей. Диапазон значений — от "0" до "23".
h	Часы в 12-часовом формате с ведущими нулями. Диапазон— от "01" до "12"
H	Часы в 24-часовом формате с ведущими нулями. Диапазон— от "00" до "23"

Функция date

Код	Описание
i	Минуты с ведущими нулями. Диапазон значений — от "00" до "59".
j	День месяца в виде числа без ведущих нулей. Диапазон значений — от "1" до "31".
l	День недели в полнотекстовом формате. Диапазон значений — от "Monday" (понедельник) до "Sunday" (воскресенье).
m	Месяц в двузначном числовом формате с ведущими нулями. Диапазон значений — от "01" до "12".

Функция date

Код	Описание
M	Месяц в виде трехбуквенной аббревиатуры. Диапазон значений — от "Jan" (январь) до "Dec" (декабрь).
n	Месяц в виде числа без ведущих нулей. Диапазон значений — от "1" до "12".
s	Секунды с ведущими нулями. Диапазон значений от "00" до "59".
S	Порядковый суффикс для дат в двухбуквенном формате ("st", "nd", "rd").
t	Полное количество дней в месяце. Диапазон значений — от "28" до "31".

Функция date

Код	Описание
T	Временная зона сервера, заданная в трехбуквенном формате, например, "EST".
w	День недели в виде числа. Диапазон значений — от "0" (воскресенье) до "6" (суббота).
y	Год в двузначном формате, например, "00".
Y	Год в четырехзначном формате, например, "2000".
z	День года в виде числа. Диапазон значений — от "0" до "365".

Открытие файла

- Для открытия файла в среде PHP используется функция **fopen()**. При открытии файла необходимо указать режим его использования.

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "w");
```

- **fopen** ожидает двух или трех входных параметров. Обычно используются два.
- Первым параметром должен быть файл, который необходимо открыть. При этом можно указать путь к файлу. Указанный путь называется относительным, поскольку он описывает позицию в файловой системе относительно $DOCUMENT_ROOT.

Открытие файла

- В среде UNIX в качестве разделителя каталогов используется символ прямой слэш (/). На платформах Windows можно применять символы прямой или обратный слэш . При использовании символа обратный слэш он должен быть помечен как специальный, чтобы функция **fopen** смогла их корректно интерпретировать. Для этого помещают дополнительный символ обратный слэш :

```
$fp = fopen("..\\..\\orders\\orders.txt", "w") ;
```

- Второй параметр функции fopen() — это режим открытия файла, который должен иметь строковый тип. Режимы файла перечислены в таблице.

Открытие файла

Режим	Значение
r	Режим чтения, начиная с начала файла.
r+	Режим чтения и записи, начиная с начала файла.
w	Режим записи, начиная с начала файла. Если файл существует, его содержимое удаляется. Если файл не существует, файл создается.
w+	Режим записи и чтения, начиная с начала файла. Если файл уже существует, его содержимое удаляется. Если файл не существует, он создается.

Открытие файла

Режим	Значение
a	Режим добавления (записи), начиная с конца существующего содержимого, если оно имеется. Если файл не существует, файл создается.
a+	Режим добавления (записи) и чтения, начиная с конца существующего содержимого, если оно имеется. Если файл не существует, он создается.
b	Двоичный режим (в сочетании с одним из остальных режимов). Windows различает эти файлы, а UNIX – нет.

Открытие файла

- Третий параметр функции **`fopen()`** не является обязательным. Его можно использовать, если файл необходимо искать в пути **`include_path`** (определенном в конфигурации PHP). Если это требуется, установите параметр равным 1 и не задавайте имя каталога или путь:

```
$fp = fopen("orders.txt", "a", 1);
```

- В случае успешного открытия файла функция **`fopen()`** возвращает указатель на файл и сохраняет его в переменой, в данном случае **`$fp`**. Эта переменная будет использоваться для доступа к файлу, когда потребуется выполнить считывание либо запись в него.

Открытие файла

- Используя функцию **`fopen()`**, можно открывать для чтения или записи не только локальные, но и удаленные файлы с использованием протоколов FTP и HTTP. Если используемое имя файла начинается с ftp://, открывается FTP-соединение с указанным сервером в пассивном режиме и возвращается указатель на начало файла. Если используемое имя файла начинается с http://, открывается HTTP-соединение с указанным сервером и возвращается указатель на ответ от сервера. При этом обязательно следует указывать завершающие символы косой черты в именах каталогов, как показано в следующем примере:

```
http://www.server.com/
```

Открытие файла

- Обычная ошибка, связанная с открытием файла — попытка открыть файл, для которого отсутствуют права на чтение или запись. В этом случае PHP выводит соответствующее предупреждение.
- В случае получения подобного сообщения об ошибке необходимо убедиться, что пользователь, выполняющий сценарий, обладает правами доступа к файлу.
- Если сценарий находится в UNIX в каталоге **`~/public_html/chapter2/`**, общедоступный для записи каталог можно было бы создать, набрав следующие команды:

```
mkdir ~/orders chmod 777 -/orders
```

Открытие файла

- Если обращение к функции **fopen()** оказывается безуспешным, она возвращает значение false. Обработку ошибок можно сделать более удобной, подавив сообщение об ошибке от PHP, и реализовав собственное:
- `@$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "a", 1);`
- `if (!$fp) {echo "<p><strong> Your order could not be processed.</strong></p></body></html>"; exit;}`
- Символ @ перед обращением к функции указывает PHP на необходимость подавления любых сообщений об ошибках, генерируемых после вызова функции. Символ @ должен располагаться в самом начале строки.

Запись в файл

- Для этого можно воспользоваться функцией **`fwrite()`** или **`fputs()`**. Функцию **`fwrite()`** можно вызвать так:

```
fwrite($fp, $outputstring);
```

- Это запись строки из переменной **`$outputstring`** в файл, указанный **`$fp`**.
- Прототип fwrite() имеет вид

```
Int fputs(int fp,string str,int [len]);
```

- Третий необязательный параметр **`len`** - максимальное количество байтов, которые требуется записать. Функция записывать строку str в файл, пока не встретит конец строки или не запишет len байтов, в зависимости от того, что произойдет раньше.

Запись в файл

- Создадим строку, которая представляет одну запись в файле данных:
- `$output = $date. "\t". $HTTP_POST_VARS["name"]. "\t". $_POST["textarea"]."\t".join($checkboxes,",")."\t".$_POST["list"]."\n";`
- В этом примере каждая запись заказа сохраняется в отдельной строке файла.
- После сохранения нескольких записей содержимое файла будет выглядеть так: http://217.71.139.95/~gun/samples/lab2_2/1c3.txt.

Закрытие файла

- По завершении использования файла его следует закрыть при помощи функции **`fclose()`**, как показано ниже:
- **`fclose($fp);`**
- Эта функция возвращает значение **`true`** в случае успешного закрытия файла и **`false`**, если файл не был закрыт. Ошибка при этом значительно менее вероятна, чем при открытии файла, поэтому в данном случае проверка выполнения функции не выполняется.

Считывание из файла

- В примере используется цикл **`while`** для считывания из файла конца файла. Проверка производится функцией **`feof()`**:
- **`while (!feof($fp))`**
- Для считывания из файла используется функция **`fgets()`**:
- **`$order= fgets($fp, 100);`**
- Считывание будет выполняться, пока не встретится символ новой строки (\n), **EOF** или из файла не будут прочитаны 99 байт.

Считывание из файла

- Интересна функция **`fgetss()`**:
- **`string fgetss (int fp, int length, string [allowable_tags] ) ;`**
- Она подобна функции **`fgets()`**, но будет избавляться от любых дескрипторов **PHP** и **HTML**, найденных в строке. Если в файле необходимо оставить конкретные дескрипторы, они должны быть включены в строку **allowable_tags.** Функцию **`fgetss()`** следует использовать для обеспечения безопасности при считывании файла, содержащего данные, введенные пользователем.

Считывание из файла

- Функция **`fgetcsv()`** — вариация **`fgets()`**:
- **`array fgetcsv(int fp, int length, string [delimiter]);`**
- Используется для разделения строк файлов при использовании в качестве разделительного символа табуляции или запятой. Если требуется восстановить переменные в строке отдельно одна от другой, следует прибегнуть к функции fgetcsv(). Ей необходимо передать разделитель полей. Например:
- **`$order = fgetcsv($fp, 100, "\t");`**
- Параметр length должен быть больше длины самой длинной строки считываемого файла, выраженной в символах.

Считывание из файла

- Вместо считывания по одной строке из файла за один проход можно считывать весь файл. Существуют три различных способа.
- readfile(). Открывает файл, повторяет его содержимое в стандартном выводе (окне браузера), а затем закрывает файл. Прототип этой функции имеет вид
- **`int readfile (string имя_файла, int [use_include_path]) ;`**
- Необязательный второй параметр указывает, должен ли PHP искать файл в пути use_include_path, и действует так же, как в функции **`fopen()`**. Функция возвращает общее количество байтов, считанных из файла.

Считывание из файла

- Во-вторых, можно использовать функцию **`fpassthru()`**. Вначале необходимо открыть файл с помощью функции **`fopen()`**. Затем указатель файла можно передать в функцию **`fpassthru()`**, которая загрузит содержимое файла, начиная с позиции, заданной указателем, в стандартный вывод. По завершении функция закрывает файл:
- `$fp = fopen("orders.txt", "r");`
- `fpassthru($fp);`
- Функция **`fpassthru()`** возвращает значение **true**, если считывание было выполнено успешно, и **false** — в противном случае.

Считывание из файла

- Третья возможность считывания всего файла — использование функции **`file()`**. Эта Функция идентична функции **`readfile()`** за исключением того, что вместо повторения файла в стандартном выводе она преобразует его в массив. Ее вызов выглядит так:
- **`$filearray = file($fp);`**
- Эта строка приведет к считыванию всего файла в массив, названный **`$filearray`**. каждая строка файла сохраняется в отдельном элементе массива.

Считывание из файла

- Еще одна возможность обработки файлов — считывание из файла по одному символу с помощью функции **fgetc()**:
- **while (!feof($fp)) {**
- **$char = fgetc($fp);**
- **if (!feof($fp))**
- **echo ($char=="\n" ? "
": $char); }**
- Этот код сохраняет его в переменной $char, пока не будет достигнут конец файла. Затем выполняется дополнительная обработка с целью замещения текстовых символов конца строки \n HTML-разделителями строк
.

Считывание из файла

- Побочный эффект использования функции **`fgetc()`**: она будет возвращать символ **EOF**, в то время как **`fgets()`** не делает этого. После считывания символа приходится снова выполнять проверку с помощью функции **`feof()`**, поскольку символ **EOF** не должен отображаться в окне браузера.
- Последний способ — использование функции **`fread()`** для считывания из файла произвольного количества байтов:
- **`string fread(int fp, int length);`**
- Функция считывает **length** байтов или все байты до конца файла, в зависимости от того, что произойдет раньше.

Другие файловые функции

- **Проверка существования файла.** Можно проверить файл на предмет существования без его открытия, воспользоваться функцией **`file_exists()`**:
- `If file_exists ("orders.txt"))`
- `  echo "There are orders waiting.";`
- `Else echo "There are no orders.";`
- **Выяснение размера файла.** Размер файла можно проверить с помощью функции **`filesize()`**. Она возвращает размер файла, выраженный в байтах:
- `echo filesize("orders.txt");`

Другие файловые функции

- **Удаление файла unlink()**:
- `unlink("orders.txt");`
- Эта функция возвращает значение **false**, если файл не может быть удален. Это будет происходить при недостаточном уровне прав доступа к файлу или если файл не существует.
- **Перемещение внутри файла.** Выяснять позицию указателя файла внутри файла и изменять ее можно с помощью функций **`rewind()`, `fseek()` и `ftell()`.**
- Функция **`rewind()`** переустанавливает указатель файла на начало файла. Функция **`ftell()`** сообщает в байтах позицию указателя относительно начала файла.

Другие файловые функции

- Примеры:
- `echo "position is ".(ftell($fp));`
- `rewind($fp);`
- `echo "Position is".(ftell ($fp));`
- Функция **`fseek()`** может использоваться для установки указателя файла внутри файла:
- `int fseek(int fp, int offset);`
- При вызове **`fseek()`** указатель файла **fp** устанавливается в точку файла, имеющую смещение **offset** байтов относительно начала файла. Вызов функции **`rewind()`** эквивалентен вызову функции **`fseek()`** со смещением, равным нулю.

Блокирование файлов

- Во избежание одновременного доступа используется блокирование файлов с помощью функции **`flock()`**. Эта функция должна вызываться после открытия файла, но перед считыванием данных из файла или их записью:
- **`bool flock(int fp, int operation);`**
- В функцию необходимо передать указатель на открытый файл и число, представляющее вид требуемой блокировки. Функция возвращает значение **true**, если блокировка была успешно выполнена, и **false** — в противном случае.
- Возможные значения параметра operation перечислены в таблице.

Блокирование файлов

Значение	Описание
1	Блокировка чтения. Файл может использоваться совместно с другими читающими приложениями.
2	Блокировка записи. Это монопольный режим. Файл не доступен для совместного использования.
3	Снятие существующей блокировки.
+4	Добавление 4 к текущему значению параметра предотвращает другие попытки блокирования во время выполнения текущего блокирования.

Проблемы

- При работе с двумерными файлами возникает ряд проблем:
  - ✓ Когда двумерные файлы становятся большими, работа с ними существенно замедляется.
  - ✓ Поиск конкретной записи или группы записей в файле затруднен.
  - ✓ Конкурирующий доступ может порождать проблемы.
  - ✓ При необходимости вставить записи или удалить их из середины файла (т.е. при необходимости произвольного доступа), это может оказаться затруднительным.
  - ✓ Кроме ограничений, налагаемых правами доступа к файлам, не существует никакого способа обеспечения различных уровней доступа к данным.