

Лабораторная работа № 6 "Работа с файлами"

1. Особенности работы с файлами в PHP

Сначала разберёмся, для чего нам необходимо уметь работать с файлами. Вспомните пример организации авторизации из предыдущего урока. В сравнении с реальностью он достаточно сильно упрощён, потому что обычно вход на сайт осуществляется с помощью пары логин-пароль. И вот тут, при попытке усложнить данный пример, мы бы столкнулись с серьёзной проблемой, ведь нам нужно где-то хранить пары допустимых значений, для того чтобы сравнивать их с теми, что ввёл пользователь.

Одним из вариантов хранения информации являются файлы. Пример с парами логинов и паролей является немного оторванным от жизни, так как их обычно хранят в базе данных. Однако мы увидели, что возникают задачи, при которых нам может понадобиться хранить информацию на сервере. В реальной практике работа с файлами чаще всего осуществляется при записи логов, так как любая более-менее сложная система обязательно их делает.

Начнём с радостной новости. В PHP можно не задумываться о том, в какую сторону и сколько слешей ставить при написании пути до файла. Интерпретатор сам разберётся, в какой системе он работает, и правильно сформирует адрес.

- PHP существует два режима работы с файлами:

1. текстовый
2. бинарный

Первый используется для работы с текстовыми документами, а второй применяется для операций с байтами информации абсолютно любого файла. Однако, поскольку в PHP нет типа данных «байт», работа всегда ведётся со строковыми данными. Поэтому разница между текстовым и бинарным режимами практически отсутствует. Она заключается лишь в том, что в системах семейства Unix для перевода строки используется символ «\n», а в Windows – последовательность «\r\n». При работе в текстовом режиме PHP-интерпретатор сам определит, какой вариант нужно использовать.

Работа с любым файлом состоит из трёх составляющих:

1. открытие файла;
2. проведение операций с данными;
3. закрытие файла.

Открыть файл можно с помощью специальной функции **fopen**. Чаще всего её вызов выглядит следующим образом: `fopen($path, $mode)`, где `$path` – путь до файла, а `$mode` – режим работы с ним. Сразу оговоримся, что здесь речь идёт не о бинарном и текстовом режимах, а о тех, которые мы рассмотрим ниже.

`$mode` здесь – указание того, что мы собираемся с файлом делать. Всего существуют шесть возможных вариантов.

r+ Файл открывается одновременно на чтение и запись. Как и для режима `r`, если файла не существует, происходит регистрация ошибки.

w- Создает новый пустой файл. Если на момент вызова уже был файл с таким именем, то он уничтожается.

w+ Аналогичен `r+`, но если файла изначально не существовало, то он создаётся.

a Используется для добавления информации в конец файла.

a+ Аналогичен предыдущему, за исключением того, что если изначально файл отсутствовал, то он будет создан

В конце любой из строк `r`, `w`, `a`, `r+`, `w+` и `a+` может находиться еще один необязательный символ — `b` или `t`, который указывает на бинарный либо текстовый способ работы.

В зависимости от выбранного режима с файлом осуществляются различные операции, которые мы рассмотрим ниже. Однако в любом случае, после их завершения файл необходимо закрыть. Для этого существует специальная функция **`fclose`**, которая в качестве параметра принимает ссылку на ранее открытый файл.

В общем случае схема работы выглядит следующим образом:

```
$f = fopen($file, $mode);
```

```
//Совершаем различные операции
```

```
fclose($f);
```

В переменной `$f` находится так называемый дескриптор файла. Звучит устрашающе, но, на самом деле, это просто представление файла в удобном для PHP-виде. С дескриптором мы работаем с помощью специальных функций, примеры которых сейчас рассмотрим.

Чтение данных из файла

Рассмотрим пример посимвольного чтения данных из файла.

```
$f = fopen('file.txt', 'r'); // Открываем файл на чтение
```

```
$c = fread($f, 1);           // Считываем первый символ  
while($c != null)           // До тех пор, пока не дошли до конца файла  
{  
    echo "$c<br/>";           // Выводим текущий символ на экран  
    $c = fread($f, 1);       // И считываем следующий  
}  
fclose($f);                 // Закрываем файл
```

Функция **`fread`** в качестве параметров принимает дескриптор файла и количество символов, которое необходимо считать, и возвращает их. В самом дескрипторе при этом происходит смещение указателя текущей позиции. Это позволяет при последующем вызове **`fread`** начинать чтение с того места файла, до которого мы уже дошли.

Аналогично функции **`fread`** работает функция **`fgets`**, однако, она используется при текстовом режиме работы с файлом, а **`fread`** — при бинарном.

3. Запись данных в файл

Теперь разберём пример добавления данных в конец файла.

```
function log($msg)  
{  
    $time = date('H:i:s');  
    $f = fopen('log.txt', 'a+');  
    fputs($f, "$time: $msg \n");  
    fclose($f);  
}
```

```

}

if(здесь сложное условие в котором мы сомневаемся){ log('попали сюда');

}

else{

log('попали туда');

}

```

Рассмотрим данный код подробнее. Вверху мы объявили специальную функцию, которая в качестве параметра принимает некоторое сообщение и добавляет его в конце файла log.txt. Практическое её назначение – ведение логов. Функция **fputs** в качестве первого параметра принимает дескриптор файла, а в качестве второго – строчку, которую мы добавляем в файл. Она будет выглядеть как [текущее время: сообщение].

В реальности это можно использовать для ведения логов и для отладки скрипта. Например, представьте, что где-то есть ветвление по сложному условию, в правильности работы которого Вы сомневаетесь. Для того, чтобы прояснить ситуацию, в каждой из веток кода можно вызвать функцию log с разными сообщениями.

4. Функции для удобной работы с файлами

Возможно, пока что методы работы с файлами в PHP не показались Вам удобными. Действительно, в 99% ситуаций нет ни необходимости, ни желания считывать файл по байтам. Хочется просто сразу получить всю информацию из него. Для этого уже есть специальные функции:

file_get_contents(\$path) Возвращает все данные из файла в виде одной строки

file (\$path) Возвращает массив, составленный из строк файла

Не возник ли у Вас вопрос, а зачем мы тогда вообще рассматривали посимвольное считывание данных из файла? Ведь использовать данные функции, действительно, гораздо удобнее.

Дело в том, что файл может быть очень большого размера. В таком случае при использовании вышеуказанных функций PHP выдаст ошибку о том, что превышен лимит используемой памяти. Поэтому работа с файлом с помощью **fopen** и **fread** тоже имеет смысл, но всё-таки чаще Вы будете использовать именно **file_get_contents** и **file**.

Также существует огромное количество полезных функций и для проведения других действий над файлами.

file_put_contents(\$path, \$data) Создаёт новый файл по пути \$path, внутрь которого кладёт информацию, содержащуюся в переменной \$data

copy(\$src, \$dst) Копирует файл, лежащий по пути \$src в \$dst

rename(\$oldname, \$newname) Переименовывает либо перемещает файл, лежащий по пути \$oldname в \$newname

unlink(\$path) Удаляет файл, лежащий по пути \$path

filesize(\$path)	Возвращает размер файла, лежащего по пути \$path, в байтах
file_exists(\$path)	Возвращает true в случае, если найден файл по пути \$path, и false в противном случае. Данную функцию рекомендуется использовать как проверку перед началом чтения информации из файла
rtrim(\$string)	Данная функция не имеет прямого отношения к работе с файлами, но часто используется. Например, функция file возвращает массив строк, в которых могут присутствовать символы переноса «\n» или «\r\n» rtrim обрезает данные символы с правой стороны строки

Это далеко неполный перечень полезных функций для удобной работы с файлами. Однако рассмотренных возможностей должно хватить для решения основных задач.

5. Загрузка файлов на сервер

До этого момента мы рассматривали только работу с текстовыми файлами, в которых хранится некоторая информация. А теперь разберёмся с тем, как можно организовать загрузку файлов на сервер. Чаще всего на сервер загружаются либо картинки, либо какие-то документы.

В корне сайта создайте папку img. Затем создайте страничку index.php и поместите в неё следующий код:

```
<?php
```

```
function upload_file($file)
```

```
{
if ($file['name'] == "")
{
echo 'Файл не выбран!';
return;
}
```

```
if(copy($file['tmp_name'], 'img/'.$file['name']))
```

```
echo 'Файл успешно загружен';
else
echo 'Ошибка загрузки файла';
}
?>
```

```
<!DOCTYPE html>
```

```

<html>
<head>
<meta charset="cp1251"/>
<title>Загрузка файла на сервер</title>
</head>
<body>
<h1>Пример загрузки файла на сервер</h1> <?php
if (isset($_FILES['file']))
{
    upload_file($_FILES['file']);
}
?>

<form method="post" enctype="multipart/form-data"> <input type="file" name="file" />

<input type="submit" value="Загрузить файл!" /> </form>

</body>
</html>

```

Для загрузки файла используется элемент HTML input с типом «файл»:

```
<input type="file" name="text" />
```

При этом важно указать у формы атрибут enctype:

```
<form method="post" enctype="multipart/form-data">
```

Если это не сделать, файл не будет загружен на сервер.

При обработке отправки формы информацию о загруженных файлах можно найти в системном массиве \$_FILES по ключу, указанному у элемента input на форме. В нашем примере данные будут храниться в \$_FILES['file'].

Информация о файле включает в себя:

name: имя файла (как он называется у пользователя);

tmp_name: путь к временному файлу (на сервере);

size: размер файла (в байтах);

type: тип выбранного файла (например, «image/jpeg»);

error: код ошибки (возникает в том случае, если попытка загрузки была неудачной).

Когда браузер отправляет файл на сервер, PHP сохраняет его во временной директории.

Добраться до файла помогает свойство tmp_name, в котором указан путь до неё.

В итоге самой операцией загрузки является строка

```
<copy($file['tmp_name'], 'img/'.$file['name'])
```

которая копирует файл из временной директории в папку с картинками.

6. Листинг директории

Работа с файлами тесно пересекается с работой с папками. Например, нам могут понадобиться имена всех картинок, которые есть у нас в директории `img`. Рассмотрим основные функции для работы с каталогами:

mkdir (\$path) Данная функция может принимать большее количество параметров. Подробное описание смотрите в справочнике	Создаёт новую директорию по пути <code>\$path</code>
rmdir (\$path)	Удаляет директорию, находящуюся по пути <code>\$path</code>
opendir(\$path)	Открывает директорию, находящуюся по пути <code>\$path</code>
readdir(\$dir)	Считывает имя следующего файла, лежащего в директории. <code>\$dir</code> – дескриптор папки, который вернула функция <code>opendir</code>
closedir(\$dir)	Завершает работу с директорией. <code>\$dir</code> – дескриптор папки, который вернула функция <code>opendir</code>
scandir(\$path)	Самая удобная функция. Возвращает массив с именами файлов и подпапок, лежащих в директории по пути <code>\$path</code>

Контроль

1. Имеет ли значение, какие слэши ставятся при написании пути до файла
2. Какие два режима работы с файлами существуют в PHP
3. Каков порядок работы с файлами
4. С помощью какой функции можно прочитать несколько символов из файла
5. С помощью какой функции можно записать символы в файл
6. Какие функции для удобной работы с файлами Вы знаете
7. Зачем нужна низкоуровневая работа с файлами, если есть удобные функции
8. Как загрузить изображение на сервер
9. В каком массиве хранится информация об изображении, загружаемом на сервер
10. Как получить список всех файлов в директории

Задание

Создайте галерею фотографий. Она должна состоять всего из одной странички, на которой пользователь видит все картинки в уменьшенном виде и форму для загрузки нового изображения. При клике на фотографию она должна открыться в браузере в новой вкладке. Размер картинок можно ограничивать с помощью свойства `width`.

При загрузке изображения необходимо делать проверку на тип и размер файла. При загрузке изображения на сервер должна создаваться его уменьшенная копия. А на странице `index.php` должны выводиться именно копии. На реальных сайтах это активно используется для экономии трафика. При клике на уменьшенное изображение в браузере в новой вкладке должен открываться оригинал изображения.

