Формы php

```
<form action="" method="GET">
<input type="text" name="user"><br><br>
<textarea name="message"></textarea><br><br>
<input type="submit">
</form>
```

Атрибут **action** задает адрес страницы сайта, на которую будут отправляться введенные данные. Если оставить этот атрибут пустым - форма отправится на текущую страницу сайта.

Атрибут **method** задает способ отправки формы. Может принимать значение **GET** или **POST**. При отправки методом GET данные из формы будут видны в адресной строке, а при отправке методом POST - не видны.

Обработка PHP форм (метод POST)

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Когда пользователь заполняет форму выше и нажимает кнопку отправки, данные формы отправляются для обработки в PHP файл с именем «welcome.php».Формы данные передаются с помощью метода HTTP POST.

```
<html>
<body>

Welcome <?php echo $_POST["name"];
?><br>
Your email address is: <?php echo
$_POST["email"]; ?>

</body>
</html>
```

«Welcome.php» выглядит следующим образом:

$_GET, $_POST, $_REQUEST

- В **$_GET** будут лежать данные, отправленные методом GET,
- в **$_POST** будут лежать данные, отправленные методом POST,
- а **$_REQUEST** - данные, отправленные и тем, и другим методом одновременно.

Обработка PHP форм (метод GET)

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo
$_GET["email"]; ?>

</body>
</html>
```

Когда использовать GET?

- Информация , передаваемая из формы с помощью метода GET является **видимым для всех** (все имена переменных и значения отображаются в URL). GET также имеет ограничения на объем информации для отправки. Ограничение составляет около 2000 символов. Однако, поскольку переменные отображаются в URL, можно добавить в закладки страницу. Это может быть полезно в некоторых случаях.
- GET может использоваться для отправки без конфиденциальных данных.
- **Примечание:** GET никогда не должны использоваться для отправки паролей или другой конфиденциальной информации!

Когда использовать POST?

- Информация , передаваемая из формы с помощью метода POST является **невидимыой для других** (все имена / значения встроены в теле запроса HTTP) и не имеет **никаких ограничений** на объем информации для отправки.
- Кроме того POST поддерживает расширенные функциональные возможности, такие как загрузки файлов на сервер.
- Однако, поскольку переменные не отображаются в URL, то данный путь с POST запросом нельзя сохранить в закладки
- **Разработчики предпочитают POST для отправки данных формы.**

Сохраняем значения полей формы после отправки

```
<form action="" method="GET">
<input type="text" name="user" value="<?php echo
$_REQUEST['user']; ?>">
<input type="submit">
</form>
```

```
<form action="" method="GET">
<input name="user" value="<?php
if(isset($_REQUEST['user'])) echo $_REQUEST['user'];
?>">
<input type="submit">
</form>
```

PHP Form Validation

Форма PHP Validation Пример

* Обязательное поле.

Имя: *

Электронная почта: *

Веб - сайт:

Комментарий:

Пол: женский Мужской *

Отправить

Поле	Правило
Name	Обязательное, которое содержит буквы и пробелы
E-mail	Обязательное, должно содержать валидный имейл адресс
Website	Опциональное поле, должно содержать только валидный url (если содержит вообще что-то)
Comment	Опциональное поле, многострочный текст
Gender	Обязательное, выбор пола

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5"
cols="40"></textarea>
Gender:
<input type="radio" name="gender"
value="female">Female
<input type="radio" name="gender"
value="male">Male
```

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

PHP Form Validation

- Обратите внимание на то, что в начале сценария, мы проверяем форма была отправлена с помощью $ _SERVER [«REQUEST_METHOD»]. Если REQUEST_METHOD является POST, то форма была отправлена - и это должно быть подтверждено. Если метод не найден, то пропускаем проверку и отображаем пустую форму.

- Тем не менее, в приведенном выше примере, все входные поля являются необязательными. Скрипт работает отлично, даже если пользователь не вводит никаких данных.

```php
<?php
// define variables and set to empty values
$name = $email = $gender = $comment =
$website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

PHP Form Validation

- В следующем коде мы добавили некоторые новые переменные: $ nameErr, $ emailErr, $ genderErr и $ websiteErr. Эти переменные будут содержать сообщения об ошибках для требуемых полей. Мы также добавили условие «не пусто» для каждой переменной $ _POST. Если переменная пуста, сообщение об ошибке сохраняется в переменную ошибки, а если он не пустой, она посылает входные данные пользователя через функцию test_input ():

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr
= "";
$name = $email = $gender = $comment = $website
= "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
```

PHP Form Validation

Приведенный ниже код показывает простой способ как проверить, если поле имя содержит только буквы и пробелы. Если значение поля имени не является действительным, то сохранить сообщение об ошибке:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space
allowed";
}
```

Функция preg_match () ищет строку для шаблона, возвращает истину, если шаблон существует, и ложь в противном случае.

PHP Form Validation

Самый простой и безопасный способ проверить, является ли валидным адрес электронной почты это использовать функцию PHP filter_var().
В приведенном ниже коде, если адрес электронной почты не хорошо сформирован, то сохранится сообщение об ошибке:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{
  $emailErr = "Invalid email format";
}
```

PHP Form Validation

Приведенный ниже код показывает способ как проверить, является ли синтаксис URL-адрес действительным Если синтаксис URL-адрес не является действительным, то сохранится сообщение об ошибке:

```
$website = test_input($_POST["website"]);
if
(!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[
-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP Form Validation

```
Name: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo $email;?>">

Website: <input type="text" name="website" value="<?php echo $website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>

Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

- Спросите **город** пользователя с помощью формы. Результат запишите в переменную **$city**. Выведите на экран фразу **'Ваш город: %Город%'**.

```
<form action="" method="GET">
<input type="text" name="city">
<input type="submit">
</form>
```

```
<?php //Если форма была отправлена и город не
пустой: if (!empty($_REQUEST['city'])) { $city =
$_REQUEST['city']; echo 'Ваш город: '.$city; } ?>
```

- Решим предыдущую задачу так, чтобы пользователь **не мог вводить теги** и сломать нам сайт. Для этого **при записи** содержимого поля в переменную **$city** обработаем его (содержимое, то есть **$_REQUEST['city']**) функцией strip_tags, которая удалит теги (можно вместо нее обработать функцией htmlspecialchars):

```
<form action="" method="GET">
<input type="text" name="city">
<input type="submit">
</form>
```

```
<?php //Если форма была отправлена и город не
пустой: if (isset($_REQUEST['city'])) { $city =
strip_tags($_REQUEST['city']); echo 'Ваш город:
'.$city; } ?>
```

- **Скрываем форму после отправки**

```
<?php //Если город пустой - покажем форму if
(isset($_REQUEST['city'])) { ?>

<form action="" method="GET">
<input type="text" name="name">
<input type="submit">
</form> <?php } ?>

<?php //Если форма была отправлена и город не
пустой: if (isset($_REQUEST['city'])) { $city =
strip_tags($_REQUEST['age']); echo 'Ваш город:
'.$age; } ?>
```

1. Спросите **имя** пользователя с помощью формы. Результат запишите в переменную **$name**. Выведите на экран фразу **'Привет, %Имя%'**.
2. Спросите у пользователя **имя**, **возраст**, а также попросите его ввести **сообщение** (его сделайте в **textarea**). Выведите эти данные на экран в **формате**, приведенном под данной задачей. Позаботьтесь о том, чтобы пользователь **не мог вводить теги** (просто удаляйте их) и таким образом сломать сайт. (Привет, Дмитрий, 25 лет. Твое сообщение: ...)
3. Спросите **возраст** пользователя. Если форма была **отправлена** и **введен** возраст, то выведите его на экран, а форму уберите. Если же форма не была отправлена (это будет при **первом** заходе на страницу) - просто покажите ее.
4. Спросите у пользователя **логин** и **пароль** (в браузере должен быть звездочками). Сравните их с логином **$login** и паролем **$pass**, хранящихся в файле. Если все верно - выведите 'Доступ разрешен!', в противном случае - 'Доступ запрещен!'. Сделайте так, чтобы скрипт обрезал концевые пробелы в строках, которые ввел пользователь.

Атрибуты value и placeholder

1. Спросите **имя** пользователя с помощью формы. Результат запишите в переменную **$name**. Сделайте так, чтобы после отправки формы значения ее полей не пропадали.
2. Спросите у пользователя **имя**, а также попросите его ввести сообщение (**textarea**). Сделайте так, чтобы после отправки формы значения его полей не пропадали.

Многомерные массивы

Многомерный массив представляет собой массив, содержащий один или более массивов. PHP понимает многомерные массивы, которые содержат два, три, четыре, пять или более уровней вложенности. Однако массивы более трех уровней вложенности трудно понимаемы и управляемы для большинства людей.

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

Мы можем хранить данные из приведенной выше таблицы в двумерном массиве

Многомерные массивы

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

Теперь двумерный массив автомобилей содержит четыре массива, и он имеет два индекса: строки и столбца.
Чтобы получить доступ к элементам массива мы должны указать на два индексы (строки и столбцы):

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

Многомерные массивы

```php
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

PHP File Handling

Обработка файлов является важной частью любого веб-приложения. Вам часто нужно открывать и обрабатывать файл для разных задач.

PHP имеет несколько функций для создания, чтения, загрузки и редактирования файлов.

Функция readfile () считывает файл и записывает его в выходной буфер.
Предположим, у нас есть текстовый файл с именем «webdictionary.txt», который хранится на сервере, который выглядит так:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

```php
<?php
echo readfile("webdictionary.txt");
?>
```

Функция readfile () полезна, если все, что вы хотите сделать, это открыть файл и прочитать его содержимое.

PHP Open File - fopen ()

Лучшим способом открытия файлов является функция fopen (). Эта функция предоставляет больше опций, чем функция readfile ().

Первый параметр fopen () содержит имя файла, который должен быть открыт, а второй параметр указывает, в каком режиме должен быть открыт файл. В следующем примере также генерируется сообщение, если функция fopen () не может открыть указанный файл:

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("Unable
to open file!");
echo fread($myfile,filesize("webdictionary.txt"
));
fclose($myfile);
?>
```

Файл может быть открыт в одном из следующих режимов:

Modes	Description
r	**Open a file for read only**. Указатель файла начинается с начала файла.
w	**Open a file for write only**. Стирает содержимое файла или создает новый файл, если он не существует. Указатель файла начинается с начала файла.
a	**Open a file for write only**. Существующие данные в файле сохраняются. Указатель файла начинается в конце файла. Создает новый файл, если файл не существует
x	**Creates a new file for write only**. Возвращает FALSE и ошибку, если файл уже существует
r+	**Open a file for read/write**. Указатель файла начинается с начала файла.
w+	**Open a file for read/write**. Стирает содержимое файла или создает новый файл, если он не существует. Указатель файла начинается с начала файла.
a+	**Open a file for read/write**. Существующие данные в файле сохраняются. Указатель файла начинается в конце файла. Создает новый файл, если файл не существует
x+	**Creates a new file for read/write**. Возвращает FALSE и ошибку, если файл уже существует

PHP Read File - fread ()

Функция fread () читается из открытого файла.
Первый параметр fread () содержит имя файла для чтения, а второй параметр указывает максимальное количество прочитанных байтов.
Следующий код PHP читает файл «webdictionary.txt» до конца:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File– fclose ()

Для fclose () требуется имя файла (или переменная, содержащая имя файла), которую мы хотим закрыть:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets ()

Функция fgets () используется для чтения одной строки из файла.
В приведенном ниже примере выводится первая строка файла «webdictionary.txt»:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Примечание. После вызова функции fgets () указатель файла переместился на следующую строку.

PHP Check End-Of-File - feof ()

Функция feof () проверяет, достигнут ли «конец файла» (EOF).
Функция feof () полезна для обхода файла неизвестной длины.
В приведенном ниже примере файл строки "webdictionary.txt" читается строка за строкой, пока не будет достигнут конец файла:

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("Unable
to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Чтение одиночного символа - fgetc ()

Функция fgetc () используется для чтения одного символа из файла.
В приведенном ниже примере читается символ за символом в файле «webdictionary.txt» пока не будет достигнут конец файла:

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Примечание. После вызова функции fgetc () указатель файла перемещается к следующему символу.

Функции файловой системы PHP

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file
delete()	See unlink() or unset()
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a directory
disk_total_space()	Returns the total size of a directory
diskfreespace()	Alias of disk_free_space()
fclose()	Closes an open file
feof()	Tests for end-of-file on an open file

fflush()	Flushes buffered output to an open file
fgetc()	Returns a character from an open file
fgetcsv()	Parses a line from an open file, checking for CSV fields
fgets()	Returns a line from an open file
fgetss()	Returns a line, with HTML and PHP tags removed, from an open file
file()	Reads a file into an array
file_exists()	Checks whether or not a file or directory exists
file_get_contents()	Reads a file into a string
file_put_contents()	Writes a string to a file
fileatime()	Returns the last access time of a file
filectime()	Returns the last change time of a file
filegroup()	Returns the group ID of a file
fileinode()	Returns the inode number of a file
filemtime()	Returns the last modification time of a file
fileowner()	Returns the user ID (owner) of a file
fileperms()	Returns the permissions of a file
filesize()	Returns the file size
filetype()	Returns the file type
flock()	Locks or releases a file
fnmatch()	Matches a filename or string against a specified pattern
fopen()	Opens a file or URL

fpassthru()	Reads from an open file, until EOF, and writes the result to the output buffer
fputcsv()	Formats a line as CSV and writes it to an open file
fputs()	Alias of fwrite()
fread()	Reads from an open file
fscanf()	Parses input from an open file according to a specified format
fseek()	Seeks in an open file
fstat()	Returns information about an open file
ftell()	Returns the current position in an open file
ftruncate()	Truncates an open file to a specified length
fwrite()	Writes to an open file
glob()	Returns an array of filenames / directories matching a specified pattern
is_dir()	Checks whether a file is a directory
is_executable()	Checks whether a file is executable
is_file()	Checks whether a file is a regular file
is_link()	Checks whether a file is a link

is_readable()	Checks whether a file is readable
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST
is_writable()	Checks whether a file is writeable
is_writeable()	Alias of is_writable()
lchgrp()	Changes group ownership of symlink
lchown()	Changes user ownership of symlink
link()	Creates a hard link
linkinfo()	Returns information about a hard link
lstat()	Returns information about a file or symbolic link
mkdir()	Creates a directory
move_uploaded_file()	Moves an uploaded file to a new location
parse_ini_file()	Parses a configuration file
parse_ini_string()	Parses a configuration string
pathinfo()	Returns information about a file path
pclose()	Closes a pipe opened by popen()
popen()	Opens a pipe

readfile()	Reads a file and writes it to the output buffer
readlink()	Returns the target of a symbolic link
realpath()	Returns the absolute pathname
realpath_cache_get()	Returns realpath cache entries
realpath_cache_size()	Returns realpath cache size
rename()	Renames a file or directory
rewind()	Rewinds a file pointer
rmdir()	Removes an empty directory
set_file_buffer()	Sets the buffer size of an open file
stat()	Returns information about a file
symlink()	Creates a symbolic link
tempnam()	Creates a unique temporary file
tmpfile()	Creates a unique temporary file
touch()	Sets access and modification time of a file
umask()	Changes file permissions for files
unlink()	Deletes a file

Функции фильтра PHP

Начиная с PHP 5.2.0, функции фильтра по умолчанию включены. Для использования этих функций не требуется установка.

На поведение этих функций влияют настройки в php.ini:

Name	Description	Default	Changeable
filter.default	Filter all $_GET, $_POST, $_COOKIE, $_REQUEST and $_SERVER data by this filter. Accepts the name of the filter you like to use by default. See the filter list for the list of the filter names	"unsafe_raw"	PHP_INI_PERDIR
filter.default_flags	Default flags to apply when the default filter is set. This is set to FILTER_FLAG_NO_ENCODE_QUOTES by default for backwards compatibility reasons	NULL	PHP_INI_PERDIR

Функции фильтра PHP

Function	Description
filter_has_var()	Проверяет, существует ли переменная определенного типа ввода
filter_id()	Возвращает идентификатор фильтра по указанному имени фильтра
filter_input()	Получает внешнюю переменную (например, из ввода формы) и, опционально, ее фильтрует
filter_input_array()	Получает внешние переменные (например, из ввода формы) и, опционально, фильтрует их
filter_list()	Возвращает список всех поддерживаемых фильтров
filter_var_array()	Получает несколько переменных и фильтрует их
filter_var()	Фильтрует переменную с заданным фильтром

Функции фильтра PHP

В следующем примере используется функция filter_var (), чтобы проверить, является ли переменная как типа INT, в диапазоне между 1 и 200:

```php
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int,
FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min, "max_range"=>$max)))
=== false) {
    echo("Variable value is not within the legal range");
} else {
    echo("Variable value is within the legal range");
}
?>
```

Функции фильтра PHP

В следующем примере используется функция filter_var (), чтобы проверить, является ли переменная $ ip действительным адресом IPv6:

```php
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false) {
    echo("$ip is a valid IPv6 address");
} else {
    echo("$ip is not a valid IPv6 address");
}
?>
```

Обработка исключений PHP

С PHP 5 появился новый объектно-ориентированный способ борьбы с ошибками.

Обработка исключений используется для изменения нормального потока выполнения кода, если возникает указанное условие (исключительное). Это условие называется исключением.

Это то, что обычно происходит при срабатывании исключения:

- Текущее состояние кода сохраняется
- Выполнение кода переключится на предопределенную (настраиваемую) функцию обработчика исключений
- В зависимости от ситуации обработчик может затем возобновить выполнение из состояния сохраненного кода, завершить выполнение сценария или продолжить сценарий из другого места в коде

Использование:

- Основное использование исключений
- Создание настраиваемого обработчика исключений
- Несколько исключений
- Повторное бросание исключения
- Установка обработчика исключений верхнего уровня

Примечание. Исключения должны использоваться только с условиями ошибки и не должны использоваться для перехода в другое место в коде в указанной точке.

Обработка исключений PHP

```php
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or
below");
  }
  return true;
}

//trigger exception
checkNum(2);
?>
```

```
Fatal error: Uncaught exception
'Exception'
with message 'Value must be 1 or
below' in C:\webfolder\test.php:6
Stack trace: #0
C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown
in C:\webfolder\test.php on line 6
```

Обработка исключений PHP

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}

//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

В приведенном коде появится сообщение об ошибке:

```
Message: Value must be 1 or below
```

Установка обработчика исключений верхнего уровня

Функция set_exception_handler () задает определяемую пользователем функцию для обработки всех неперехваченных исключений.

```
<?php
function myException($exception) {
  echo "<b>Exception:</b> " .
$exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception
occurred');
?>
```

Вывод кода должен быть примерно таким:
Exception: Uncaught Exception occurred
В приведенном коде не было блока «catch». Вместо этого запускается обработчик исключений верхнего уровня. Эта функция должна использоваться для захвата исключений, которые не были исключены.

Правила для исключений

- Код может быть окружен в блоке try, чтобы помочь выявить потенциальные исключения
- Каждый блок try или «throw» должен иметь по крайней мере один соответствующий блок catch
- Множество блоков catch можно использовать для обнаружения разных классов исключений
- Исключения могут быть выброшены (или повторно выбраны) в блоке catch в блоке try
- Простое правило: если вы что-то бросаете, вы должны поймать его.

Обработка ошибок PHP

- При создании сценариев и веб-приложений важна обработка ошибок. Если в коде отсутствует код проверки ошибок, ваша программа может выглядеть очень непрофессионально, и вы можете быть уязвимы в безопасности.

Методы обработки ошибок:

- Простые выражения «die ()»
- Пользовательские ошибки и триггеры ошибок
- Отчет об ошибках

Основная обработка ошибок: использование функции die ()

```
<?php
$file=fopen("welcome.txt","r");
?>
```

Если файл не существует, вы можете получить такую ошибку:

```
Warning: fopen(welcome.txt) [function.fopen]:
failed to open stream:
No such file or directory
in C:\webfolder\test.php on line 2
```

Чтобы пользователь не получал сообщение об ошибке мы проверяем, существует ли файл до того, как мы попытаемся получить к нему доступ:

```
<?php
if(!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
?>
```

Теперь, если файл не существует, вы получите такую ошибку:

```
File not found
```

Создание настраиваемого обработчика ошибок

```
error_function(error_level,error_message,
error_file,error_line,error_context)
```

- error_level - Обязательный. Задает уровень отчета об ошибке для пользовательской ошибки. Значение должно быть числом.
- error_message - Обязательный. Задает сообщение об ошибке для пользовательской ошибки
- error_file - Необязательный. Указывает имя файла, в котором произошла ошибка
- error_line - Необязательный. Указывает номер строки, в которой произошла ошибка
- error_context - Необязательный. Определяет массив, содержащий каждую переменную, и их значения, используемые при возникновении ошибки

Запуск ошибки

В сценарии, где пользователи могут вводить данные, полезно запускать ошибки при возникновении незаконного ввода. В PHP это выполняется функцией trigger_error ().

```
<?php
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below");
}
?>
```

Вывод кода выше должен быть примерно таким:

```
Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6
```

Ошибка может быть вызвана в любом месте в сценарии, и, добавив второй параметр, вы можете указать, какой уровень ошибок запускается.
Возможные типы ошибок:

- E_USER_ERROR - Неустранимая пользовательская ошибка во время выполнения. Ошибки, от которых невозможно восстановить. Выполнение сценария останавливается
- E_USER_WARNING - Нефатальное пользовательское предупреждение во время выполнения. Выполнение скрипта не останавливается
- E_USER_NOTICE - По умолчанию. Пользовательское уведомление во время выполнения. Скрипт нашел что-то, что может быть ошибкой, но может также произойти при обычном сценарии

В этом примере E_USER_WARNING возникает, если переменная «test» больше, чем «1». Если возникает E_USER_WARNING, мы будем использовать наш собственный обработчик ошибок и завершить скрипт:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}

//set error handler
set_error_handler("customError",E_USER_WARNING)
;

//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or
below",E_USER_WARNING);
}
?>
```

Вывод кода должен быть примерно таким:

Error: [512] Value must be 1 or below
Ending Script

Регистрация ошибок

- По умолчанию PHP отправляет журнал ошибок в систему регистрации или файл сервера, в зависимости от того, как в файле php.ini задана настройка error_log. С помощью функции error_log () вы можете отправлять журналы ошибок в указанный файл или удаленное место назначения.
- Отправка сообщений об ошибках самому себе по электронной почте может быть хорошим способом получения уведомления об определенных ошибках.

Отправить сообщение об ошибке по E-Mail

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From:
webmaster@example.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING)
;

//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or
below",E_USER_WARNING);
}
?>
```

Вывод кода должен быть примерно таким:

```
Error: [512] Value must be 1 or
below
Webmaster has been notified
```

И почта, полученная из вышеприведенного кода, выглядит так:

```
Error: [512] Value must be 1 or
below
```

Это не должно использоваться со всеми ошибками. Регулярные ошибки должны регистрироваться на сервере, используя стандартную систему регистрации PHP.

Работа с базой данных

Создание базы данных MySQL

- ```
  <?php
  $servername = "localhost";
  $username = "username";
  $password = "password";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password);
  // Check connection
  if (!$conn) {
      die("Connection failed: " . mysqli_connect_error());
  }

  // Create database
  $sql = "CREATE DATABASE myDB";
  if (mysqli_query($conn, $sql)) {
      echo "Database created successfully";
  } else {
      echo "Error creating database: " . mysqli_error($conn);
  }

  mysqli_close($conn);
  ?>
  ```
  ```

Создание таблицы MySQL

- ```
  <?php
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password, $dbname);
  // Check connection
  if (!$conn) {
     die("Connection failed: " . mysqli_connect_error());
  }
  // sql to create table
  $sql = "CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP
  )";
  if (mysqli_query($conn, $sql)) {
     echo "Table MyGuests created successfully";
  } else {
     echo "Error creating table: " . mysqli_error($conn);
  }
  mysqli_close($conn);
  ?>
  ```
  ```

Вставка данных в MySQL

- ```
  <?php
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password, $dbname);
  // Check connection
  if (!$conn) {
      die("Connection failed: " . mysqli_connect_error());
  }

  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";

  if (mysqli_query($conn, $sql)) {
      echo "New record created successfully";
  } else {
      echo "Error: " . $sql . "<br>" . mysqli_error($conn);
  }

  mysqli_close($conn);
  ?>
  ```
  ```

Получить идентификатор последней введенной записи

- ```
  <?php
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password, $dbname);
  // Check connection
  if (!$conn) {
      die("Connection failed: " . mysqli_connect_error());
  }

  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";

  if (mysqli_query($conn, $sql)) {
      $last_id = mysqli_insert_id($conn);
      echo "New record created successfully. Last inserted ID is: " . $last_id;
  } else {
      echo "Error: " . $sql . "<br>" . mysqli_error($conn);
  }

  mysqli_close($conn);
  ?>
  ```
  ```

Вставка множества записей в MySQL

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Prepared Statements

Подготовленный оператор - это функция, используемая для повторения одних и тех же (или подобных) SQL-предложений с высокой эффективностью.

Подготовленные заявления в основном работают следующим образом:

1. Подготовьте: шаблон базы данных SQL создается и отправляется в базу данных. Определенные значения остаются неуказанными, называемыми параметрами (помечены как «?»). Пример: INSERT INTO MyGuests VALUES (?,?,?)
2. База данных анализирует, компилирует и выполняет оптимизацию запросов в шаблоне оператора SQL и сохраняет результат без его выполнения.
3. Выполнение: позднее приложение привязывает значения к параметрам, а база данных выполняет инструкцию. Приложение может выполнять оператор столько раз, сколько требуется с разными значениями

По сравнению с выполнением SQL-запросов напрямую подготовленные операторы имеют три основных преимущества:

- Подготовленные утверждения сокращают время синтаксического анализа, поскольку подготовка по запросу выполняется только один раз (хотя оператор выполняется несколько раз)
- Связанные параметры минимизируют пропускную способность сервера, так как вам нужно отправлять только параметры каждый раз, а не весь запрос
- Подготовленные утверждения очень полезны для инъекций SQL, потому что значения параметров, которые передаются позже с использованием другого протокола, не обязательно должны быть экранированы. Если исходный шаблон инструкции не получен из внешнего ввода, SQL-инъекция не может произойти.

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

Строка кода для объяснения из приведенного выше примера:

```
"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)«
```

В нашем SQL мы вставляем вопросительный знак (?) там, где мы хотим подставить целочисленное, строковое, двойное или blob значение.

Затем взгляните на функцию bind_param ():

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

Эта функция связывает параметры с SQL-запросом и сообщает базе данных, что значат параметры. Параметр «sss» перечисляет типы данных, которые являются параметрами. Символ s сообщает mysql, что параметр является строкой.
Аргумент может быть одним из четырех типов:

- i - целое число
- d - двойной
- s - строка
- b - BLOB

Мы должны установить один из них для каждого параметра.
Говоря mysql, какой тип данных ожидать, мы минимизируем риск инъекций SQL.

Выбор данных из MySQL

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. "
" . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
?>
```

Оператор SELECT используется для выбора данных из одной или нескольких таблиц:
`SELECT column_name(s) FROM table_name`
или мы можем использовать символ *, чтобы выбрать ВСЕ столбцы из таблицы:
`SELECT * FROM table_name`

Сначала мы настраиваем SQL-запрос, который выбирает столбцы id, firstname и lastname из таблицы MyGuests. Следующая строка кода запускает запрос и помещает полученные данные в переменную с именем $ result.
Затем функция num_rows () проверяет количество строк и если возвращено больше чем 0, то функция fetch_assoc () помещает все результаты в ассоциативный массив, который мы можем обойти. Цикл while () проходит через результирующий набор и выводит данные из столбцов id, firstname и lastname.

Удалить данные из MySQL

- ?php
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password, $dbname);
  // Check connection
  if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
  }

  // sql to delete a record
  $sql = "DELETE FROM MyGuests WHERE id=3";

  if (mysqli_query($conn, $sql)) {
  echo "Record deleted successfully";
  } else {
  echo "Error deleting record: " . mysqli_error($conn);
  }

  mysqli_close($conn);
  ?>

Обновление данных в MySQL

- ```
  <?php
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  // Create connection
  $conn = mysqli_connect($servername, $username, $password, $dbname);
  // Check connection
  if (!$conn) {
      die("Connection failed: " . mysqli_connect_error());
  }

  $sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

  if (mysqli_query($conn, $sql)) {
      echo "Record updated successfully";
  } else {
      echo "Error updating record: " . mysqli_error($conn);
  }

  mysqli_close($conn);
  ?>
  ```
  ```

Ограничение выбора данных из MySQL

MySQL предоставляет предложение LIMIT, которое используется для указания количества возвращаемых записей.
Предложение LIMIT позволяет легко кодировать многостраничные результаты или разбиение на страницы с помощью SQL и очень полезно для больших таблиц. Возвращение большого количества записей может повлиять на производительность.
Предположим, мы хотим выбрать все записи из 1 - 30 (включительно) из таблицы под названием «Заказы». Тогда SQL-запрос будет выглядеть так:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

Когда SQL-запрос выше будет запущен, он вернет первые 30 записей.
Что делать, если мы хотим выбрать записи 16 - 25 (включительно)?
Mysql также предоставляет способ справиться с этим: используя OFFSET.

Ограничение выбора данных из MySQL

В приведенном ниже SQL-запросе говорится «вернуть только 10 записей, начать запись 16 (OFFSET 15)»:

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

Вы также можете использовать более короткий синтаксис для достижения того же результата:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Обратите внимание, что при использовании запятой цифры меняются на противоположные.