Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Наследование исключений »
« Ошибки в PHP 7

- Руководство по PHP
- Справочник языка

Change language: Russian

Submit a Pull Request Report a Bug

Исключения

Содержание

В PHP реализована модель исключений, аналогичная тем, что используются в других языках программирования. Исключение в PHP может быть выброшено (`throw`) и поймано (`catch`). Код может быть заключён в блок `try`, чтобы облегчить обработку потенциальных исключений. У каждого блока `try` должен быть как минимум один соответствующий блок `catch` или `finally`.

Если выброшено исключение, а в текущей области видимости функции нет блока `catch`, исключение будет "подниматься" по стеку вызовов к вызывающей функции, пока не найдёт подходящий блок `catch`. Все блоки `finally`, которые встретятся на этом пути, будут выполнены. Если стек вызовов разворачивается до глобальной области видимости, не встречая подходящего блока `catch`, программа завершается с неисправимой ошибкой, если не был установлен глобальный обработчик исключений.

Выброшенный объект должен наследовать (`instanceof`) интерфейс Throwable. Попытка выбросить объект, который таковым не является, приведёт к неисправимой ошибке PHP.

Начиная с PHP 8.0.0, ключевое слово `throw` является выражением и может быть использовано в любом контексте выражения. В предыдущих версиях оно было утверждением и должно было располагаться в отдельной строке.

`catch`

Блок `catch` определяет, как реагировать на выброшенное исключение. Блок `catch` определяет один или несколько типов исключений или ошибок, которые он может обработать, и, по желанию, переменную, которой можно присвоить исключение (указание переменной было обязательно до версии PHP 8.0.0). Первый блок `catch`, с которым столкнётся выброшенное исключение или ошибка и соответствует типу выброшенного объекта, обработает объект.

Несколько блоков `catch` могут быть использованы для перехвата различных классов исключений. Нормальное выполнение (когда исключение не выброшено в блоке `try`) будет продолжаться после последнего блока `catch`, определённого в последовательности. Исключения могут быть выброшены (`throw`) (или повторно выброшены) внутри блока `catch`. В противном случае выполнение будет продолжено после блока `catch`, который был вызван.

При возникновении исключения, код, следующий за утверждением, не будет выполнен, а PHP попытается найти первый подходящий блок `catch`. Если исключение не поймано, будет выдана неисправимая ошибка PHP с сообщением `"Uncaught Exception ..."`, если только обработчик не был определён с помощью функции set_exception_handler().

Начиная с версии PHP 7.1.0, в блоке `catch` можно указывать несколько исключений, используя символ `|`. Это полезно, когда разные исключения из разных иерархий классов обрабатываются одинаково.

Начиная с версии PHP 8.0.0, имя переменной для пойманного исключения является необязательным. Если оно не указано, блок `catch` будет выполнен, но не будет иметь доступа к выброшенному объекту.

`finally`

Блок `finally` также может быть указан после или вместо блоков `catch`. Код в блоке `finally` всегда будет выполняться после блоков `try` и `catch`, независимо от того, было ли выброшено исключение и до возобновления нормального выполнения.

Одно из заметных взаимодействий происходит между блоком `finally` и оператором `return`. Если оператор `return` встречается внутри блоков `try` или `catch`, блок `finally` всё равно будет выполнен. Более того, оператор `return` выполнится, когда встретится, но результат будет возвращён после выполнения блока `finally`. Кроме того, если блок `finally` также содержит оператор `return`, возвращается значение из блока `finally`.

Глобальный обработчик исключений

Если исключению разрешено распространяться на глобальную область видимости, оно может быть перехвачено глобальным обработчиком исключений, если он установлен. Функция set_exception_handler() может задать функцию, которая будет вызвана вместо блока `catch`, если не будет вызван никакой другой блок. Эффект по сути такой же, как если бы вся программа была обёрнута в блок `try`-`catch` с этой функцией в качестве `catch`.

Примечания

> **Замечание**:
>
> Внутренние функции PHP в основном используют отчёт об ошибках, только современные объектно-ориентированные модули используют исключения. Однако ошибки можно легко перевести в исключения с помощью класса ErrorException. Однако эта техника работает только с исправляемыми ошибками.
>
> **Пример #1 Преобразование отчётов об ошибках в исключения**
>
> ```
> <?php
> function exceptions_error_handler($severity, $message, $filename, $lineno) {
> throw new ErrorException($message, 0, $severity, $filename, $lineno);
> }
>
> set_error_handler('exceptions_error_handler');
> ?>
> ```

Подсказка

Библиотека Стандартная библиотека PHP (SPL) предоставляет большое количество встроенных исключений.

Примеры

Пример #2 Выбрасывание исключения

```
<?php
function inverse($x) {
if (!$x) {
throw new Exception('Деление на ноль.');
}
return 1/$x;
}

try {
echo inverse(5) . "\n";
echo inverse(0) . "\n";
} catch (Exception $e) {
echo 'Выброшено исключение: ', $e->getMessage(), "\n";
}

// Продолжение выполнения
echo "Привет, мир\n";
?>
```

Результат выполнения приведённого примера:

```
0.2
Выброшено исключение: Деление на ноль.
Привет, мир
```

Пример #3 Обработка исключений с помощью блока `finally`

```
<?php
function inverse($x) {
if (!$x) {
throw new Exception('Деление на ноль.');
}
return 1/$x;
}

try {
echo inverse(5) . "\n";
} catch (Exception $e) {
echo 'Поймано исключение: ', $e->getMessage(), "\n";
```

```
} finally {
echo "Первый блок finally.\n";
}

try {
echo inverse(0) . "\n";
} catch (Exception $e) {
echo 'Поймано исключение: ', $e->getMessage(), "\n";
} finally {
echo "Второй блок finally.\n";
}

// Продолжение нормального выполнения
echo "Привет, мир\n";
?>
```

Результат выполнения приведённого примера:

```
0.2
Первый блок finally.
Поймано исключение: Деление на ноль.
Второй блок finally.
Привет, мир
```

Пример #4 Взаимодействие между блоками `finally` и `return`

```
<?php

function test() {
try {
throw new Exception('foo');
} catch (Exception $e) {
return 'catch';
} finally {
return 'finally';
}
}

echo test();
?>
```

Результат выполнения приведённого примера:

```
finally
```

Пример #5 Вложенные исключения

```
<?php

class MyException extends Exception { }

class Test {
public function testing() {
try {
try {
throw new MyException('foo!');
} catch (MyException $e) {
// повторный выброс исключения
throw $e;
}
} catch (Exception $e) {
var_dump($e->getMessage());
}
}
}
```

```
$foo = new Test;
$foo->testing();

?>
```

Результат выполнения приведённого примера:

```
string(4) "foo!"
```

Пример #6 Обработка нескольких исключений в одном блоке catch

```
<?php

class MyException extends Exception { }

class MyOtherException extends Exception { }

class Test {
public function testing() {
try {
throw new MyException();
} catch (MyException | MyOtherException $e) {
var_dump(get_class($e));
}
}
}

$foo = new Test;
$foo->testing();

?>
```

Результат выполнения приведённого примера:

```
string(11) "MyException"
```

Пример #7 Пример блока `catch` без указания переменной

Допустимо начиная с PHP 8.0.0

```
<?php

class SpecificException extends Exception {}

function test() {
throw new SpecificException('Ой!');
}

try {
test();
} catch (SpecificException) {
print "Было поймано исключение SpecificException, но нам безразлично, что у него внутри.";
}
?>
```

Пример #8 Throw как выражение

Допустимо начиная с PHP 8.0.0

```
<?php

function test() {
do_something_risky() or throw new Exception('Всё сломалось');
}
```

```
try {
test();
} catch (Exception $e) {
print $e->getMessage();
}
?>
```

＋add a note

User Contributed Notes 15 notes

up
down
118
ask at nilpo dot com ¶
14 years ago

If you intend on creating a lot of custom exceptions, you may find this code useful. I've created an interface and an abstract exception class that ensures that all parts of the built-in Exception class are preserved in child classes. It also properly pushes all information back to the parent constructor ensuring that nothing is lost. This allows you to quickly create new exceptions on the fly. It also overrides the default __toString method with a more thorough one.

```
<?php
interface IException
{
/* Protected methods inherited from Exception class */
public function getMessage(); // Exception message
public function getCode(); // User-defined Exception code
public function getFile(); // Source filename
public function getLine(); // Source line
public function getTrace(); // An array of the backtrace()
public function getTraceAsString(); // Formated string of trace

/* Overrideable methods inherited from Exception class */
public function __toString(); // formated string for display
public function __construct($message = null, $code = 0);
}

abstract class CustomException extends Exception implements IException
{
protected $message = 'Unknown exception'; // Exception message
private $string; // Unknown
protected $code = 0; // User-defined exception code
protected $file; // Source filename of exception
protected $line; // Source line of exception
private $trace; // Unknown

public function __construct($message = null, $code = 0)
{
if (!$message) {
throw new $this('Unknown '. get_class($this));
}
parent::__construct($message, $code);
}

public function __toString()
{
return get_class($this) . " '{$this->message}' in {$this->file}({$this->line})\n"
. "{$this->getTraceAsString()}";
}
}
?>
```

Now you can create new exceptions in one line:

```
<?php
class TestException extends CustomException {}
?>
```

Here's a test that shows that all information is properly preserved throughout the backtrace.

```
<?php
function exceptionTest()
{
try {
throw new TestException();
}
catch (TestException $e) {
echo "Caught TestException ('{$e->getMessage()}')\n{$e}\n";
}
catch (Exception $e) {
echo "Caught Exception ('{$e->getMessage()}')\n{$e}\n";
}
}

echo '<pre>' . exceptionTest() . '</pre>';
?>
```

Here's a sample output:

```
Caught TestException ('Unknown TestException')
TestException 'Unknown TestException' in C:\xampp\htdocs\CustomException\CustomException.php(31)
#0 C:\xampp\htdocs\CustomException\ExceptionTest.php(19): CustomException->__construct()
#1 C:\xampp\htdocs\CustomException\ExceptionTest.php(43): exceptionTest()
#2 {main}
```

up
down
6

tianyiw at vip dot qq dot com ¶

5 months ago

Easy to understand `finally`.

```
<?php
try {
try {
echo "before\n";
1 / 0;
echo "after\n";
} finally {
echo "finally\n";
}
} catch (\Throwable) {
echo "exception\n";
}
?>
# Print:
before
finally
exception
```

up
down
81

Johan ¶

12 years ago

Custom error handling on entire pages can avoid half rendered pages for the users:

```
<?php
```

```
ob_start();
try {
/*contains all page logic
and throws error if needed*/
...
} catch (Exception $e) {
ob_end_clean();
displayErrorPage($e->getMessage());
}
?>
```

up
down
22

Shot (Piotr Szotkowski) ¶

15 years ago

'Normal execution (when no exception is thrown within the try block, *or when a catch matching the thrown exception's class is not present*) will continue after that last catch block defined in sequence.'

'If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception …" message, unless a handler has been defined with set_exception_handler().'

These two sentences seem a bit contradicting about what happens 'when a catch matching the thrown exception's class is not present' (and the second sentence is actually correct).

up
down
12

daviddlowe dot flimm at gmail dot com ¶

6 years ago

Starting in PHP 7, the classes Exception and Error both implement the Throwable interface. This means, if you want to catch both Error instances and Exception instances, you should catch Throwable objects, like this:

```
<?php

try {
throw new Error( "foobar" );
// or:
// throw new Exception( "foobar" );
}
catch (Throwable $e) {
var_export( $e );
}

?>
```

up
down
12

christof+php[AT]insypro.com ¶

6 years ago

In case your E_WARNING type of errors aren't catchable with try/catch you can change them to another type of error like this:

```
<?php
set_error_handler(function($errno, $errstr, $errfile, $errline){
if($errno === E_WARNING){
// make it more serious than a warning so it can be caught
trigger_error($errstr, E_ERROR);
return true;
} else {
// fallback to default php error handler
return false;
}
});
```

```
try {
// code that might result in a E_WARNING
} catch(Exception $e){
// code to handle the E_WARNING (it's actually changed to E_ERROR at this point)
} finally {
restore_error_handler();
}
?>
```

up
down
21
Edu ¶
10 years ago

The "finally" block can change the exception that has been throw by the catch block.

```
<?php
try{
try {
throw new \Exception("Hello");
} catch(\Exception $e) {
echo $e->getMessage()." catch in\n";
throw $e;
} finally {
echo $e->getMessage()." finally \n";
throw new \Exception("Bye");
}
} catch (\Exception $e) {
echo $e->getMessage()." catch out\n";
}
?>
```

The output is:

```
Hello catch in
Hello finally
Bye catch out
```

up
down
8
mlaopane at gmail dot com ¶
5 years ago

```
<?php

/**
* You can catch exceptions thrown in a deep level function
*/

function employee()
{
throw new \Exception("I am just an employee !");
}

function manager()
{
employee();
}

function boss()
{
try {
manager();
```

```
} catch (\Exception $e) {
echo $e->getMessage();
}
}

boss(); // output: "I am just an employee !"
```

up
down
11
Simo ¶
8 years ago
#3 is not a good example. inverse("0a") would not be caught since (bool) "0a" returns true, yet 1/"0a" casts the string to integer zero and attempts to perform the calculation.
up
down
9
telefoontoestel at nospam dot org ¶
9 years ago
When using finally keep in mind that when a exit/die statement is used in the catch block it will NOT go through the finally block.

```
<?php
try {
echo "try block<br />";
throw new Exception("test");
} catch (Exception $ex) {
echo "catch block<br />";
} finally {
echo "finally block<br />";
}

// try block
// catch block
// finally block
?>

<?php
try {
echo "try block<br />";
throw new Exception("test");
} catch (Exception $ex) {
echo "catch block<br />";
exit(1);
} finally {
echo "finally block<br />";
}

// try block
// catch block
?>
```

up
down
6
Tom Polomsk ¶
9 years ago
Contrary to the documentation it is possible in PHP 5.5 and higher use only try-finally blocks without any catch block.
up
down
6
Sawsan ¶
12 years ago
the following is an example of a re-thrown exception and the using of getPrevious function:

```
<?php

$name = "Name";

//check if the name contains only letters, and does not contain the word name

try
{
try
{
if (preg_match('/[^a-z]/i', $name))
{
throw new Exception("$name contains character other than a-z A-Z");
}
if(strpos(strtolower($name), 'name') !== FALSE)
{
throw new Exception("$name contains the word name");
}
echo "The Name is valid";
}
catch(Exception $e)
{
throw new Exception("insert name again",0,$e);
}
}

catch (Exception $e)
{
if ($e->getPrevious())
{
echo "The Previous Exception is: ".$e->getPrevious()->getMessage()."<br/>";
}
echo "The Exception is: ".$e->getMessage()."<br/>";
}

?>
```

up
down
1

jlherren ¶

25 days ago

As noted elsewhere, throwing an exception from the `finally` block will replace a previously thrown exception. But the original exception is magically available from the new exception's `getPrevious()`.

```
<?php
try {
try {
throw new RuntimeException('Exception A');
} finally {
throw new RuntimeException('Exception B');
}
}
catch (Throwable $exception) {
echo $exception->getMessage(), "\n";
// 'previous' is magically available!
echo $exception->getPrevious()->getMessage(), "\n";
}
?>
```

Will print:

```
Exception B
Exception A
```
up
down
-1
lscorionjs at gmail dot com ¶
1 year ago
```
<?php

try {
$str = 'hi';
throw new Exception();
} catch (Exception) {
var_dump($str);
} finally {
var_dump($str);
}

?>

Output:
string(2) "hi"
string(2) "hi"
```
up
down
-2
ilia-yats at ukr dot net ¶
1 year ago
Note some undocumented details about exceptions thrown from 'finally' blocks.

When exception is thrown from 'finally' block, it overrides the original not-caught (or re-thrown) exception. So the behavior is similar to 'return': value returned from 'finally' overrides the one returned earlier. And the original exception is automatically appended to the exceptions chain, i.e. becomes 'previous' for the new one. Example:

```
<?php
try {
try {
throw new Exception('thrown from try');
} finally {
throw new Exception('thrown from finally');
}
} catch(Exception $e) {
echo $e->getMessage();
echo PHP_EOL;
echo $e->getPrevious()->getMessage();
}

// will output:
// thrown from finally
// thrown from try
?>

Example with re-throwing:
<?php
try {
try {
throw new Exception('thrown from try');
} catch (Exception $e) {
throw new Exception('thrown from catch');
} finally {
throw new Exception('thrown from finally');
}
} catch(Exception $e) {
```

```
echo $e->getMessage();
echo PHP_EOL;
echo $e->getPrevious()->getMessage();
}

// will output:
// thrown from finally
// thrown from catch
?>
```

The same happens even if explicitly pass null as previous exception:

```
<?php
try {
try {
throw new Exception('thrown from try');
} finally {
throw new Exception('thrown from finally', null, null);
}
} catch(Exception $e) {
echo $e->getMessage();
echo PHP_EOL;
echo $e->getPrevious()->getMessage();
}

// will output:
// thrown from finally
// thrown from try
?>
```

Also it is possible to pass previous exception explicitly, the 'original' one will be still appended to the chain, e.g.:

```
<?php
try {
try {
throw new Exception('thrown from try');
} finally {
throw new Exception(
'thrown from finally',
null,
new Exception('Explicitly set previous!')
);
}
} catch(Exception $e) {
echo $e->getMessage();
echo PHP_EOL;
echo $e->getPrevious()->getMessage();
echo PHP_EOL;
echo $e->getPrevious()->getPrevious()->getMessage();
}

// will output:
// thrown from finally
// Explicitly set previous!
// thrown from try
?>
```

This seems to be true for versions 5.6-8.2.