PHP Exception Handling

In this tutorial you will learn how to throw and catch exceptions in PHP.

What is an Exception

An exception is a signal that indicates some sort of exceptional event or error has occurred. Exceptions can be caused due to various reasons, for example, database connection or query fails, file that you're trying to access doesn't exist, and so on.

PHP provides a powerful exception handling mechanism that allows you to handle exceptions in a graceful way. As opposed to PHP's traditional error-handling system, exception handling is the object-oriented method for handling errors, which provides more controlled and flexible form of error reporting. Exception model was first introduced in PHP 5.

Using Throw and Try...Catch Statements

In exception-based approach, program code is written in a `try` block, an exception can be thrown using the `throw` statement when an exceptional event occurs during the execution of code in a `try` block. It is then caught and resolved by one or more `catch` blocks.

The following example demonstrates how exception handling works:

Example **Download**

```php
<?php
function division($dividend, $divisor){
    // Throw exception if divisor is zero
    if($divisor == 0){
        throw new Exception('Division by zero.');
    } else{
        $quotient = $dividend / $divisor;
        echo "<p>$dividend / $divisor = $quotient</p>";
    }
}

try{
    division(10, 2);
    division(30, -4);
    division(15, 0);

    // If exception is thrown following line won't execute
```

```
    echo '<p>All divisions performed successfully.</p>';
} catch(Exception $e){
    // Handle the exception
    echo "<p>Caught exception: " . $e->getMessage() . "</p>";
}

// Continue execution
echo "<p>Hello World!</p>";
?>
```

You might be wondering what this code was all about. Well, let's go through each part of this code one by one for a better understanding.

Explanation of Code

The PHP's exception handling system has basically four parts: `try`, `throw`, `catch`, and the Exception class. The following list describes how each part exactly works.

- The `division()` function in the example above checks if a divisor is equal to zero. If it is, an exception is thrown via PHP's `throw` statement. Otherwise this function perform the division using given numbers and display the result.
- Later, the `division()` function is called within a `try` block with different arguments. If an exception is generated while executing the code within the `try` block, PHP stops execution at that point and attempt to find the corresponding `catch` block. If it is found, the code within that `catch` block is executed, if not, a fatal error is generated.
- The `catch` block typically catch the exception thrown within the `try` block and creates an object (`$e`) containing the exception information. The error message from this object can be retrieved using the Exception's `getMessage()` method.

The PHP's Exception class also provides `getCode()`, `getFile()`, `getLine()` and `getTraceAsString()` methods that can be used to generate detailed debugging information.

Example **Download**

```
<?php
// Turn off default error reporting
error_reporting(0);

try{
    $file = "somefile.txt";

    // Attempt to open the file
    $handle = fopen($file, "r");
    if(!$handle){
```

```
        throw new Exception("Cannot open the file!", 5);
    }

    // Attempt to read the file contents
    $content = fread($handle, filesize($file));
    if(!$content){
        throw new Exception("Could not read file!", 10);
    }

    // Closing the file handle
    fclose($handle);

    // Display file contents
    echo $content;
} catch(Exception $e){
    echo "<h3>Caught Exception!</h3>";
    echo "<p>Error message: " . $e->getMessage() . "</p>";
    echo "<p>File: " . $e->getFile() . "</p>";
    echo "<p>Line: " . $e->getLine() . "</p>";
    echo "<p>Error code: " . $e->getCode() . "</p>";
    echo "<p>Trace: " . $e->getTraceAsString() . "</p>";
}
?>
```

The Exception's constructor optionally takes an exception message and an exception code. While the exception message is typically used to display generic information on what went wrong, the exception code can be used to categorize the errors. The exception code provided can be retrieved later via Exception's `getCode()` method.

Tip: Exception should only be used to denote exceptional conditions; they should not be used to control normal application flow e.g., jump to another place in the script at a particular point. Doing that would adversely affect your application's performance.

Defining Custom Exceptions

You can even define your own custom exception handlers to treat different types of exceptions in a different way. It allows you to use a separate `catch` block for each exception type.

You can define a custom exception by extending the Exception class, because Exception is the base class for all exceptions. The custom exception class inherits all the properties and methods from PHP's Exception class. You can also add your custom methods to the custom exception class. Let's check out the following example:

Example | Download

```php
<?php
// Extending the Exception class
class EmptyEmailException extends Exception {}
class InvalidEmailException extends Exception {}

$email = "someuser@example..com";

try{
    // Throw exception if email is empty
    if($email == ""){
        throw new EmptyEmailException("<p>Please enter your E-mail address!</p>");
    }

    // Throw exception if email is not valid
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        throw new InvalidEmailException("<p><b>$email</b> is not a valid E-mail address!</p>");
    }

    // Display success message if email is valid
    echo "<p>SUCCESS: Email validation successful.</p>";
} catch(EmptyEmailException $e){
    echo $e->getMessage();
} catch(InvalidEmailException $e){
    echo $e->getMessage();
}
?>
```

In the above example we've derived two new exception classes: **EmptyEmailException**, and **InvalidEmailException** from the Exception base class. Multiple `catch` blocks are used to display different error messages, depending on the type of exception generated.

Since these custom exception classes inherits the properties and methods from the Exception class, so we can use the Exception's class methods like `getMessage()`, `getLine()`, `getFile()`, etc. to retrieve error information from the exception object.

Setting a Global Exception Handler

As we've discussed earlier in this chapter if an exception is not caught, PHP generates a Fatal Error with an "Uncaught Exception ..." message. This error message may contain sensitive information like file name and line number where the problem occurs. If you don't want to expose such information to the user, you can create a custom function and register it with the `set_exception_handler()` function to handle all uncaught exceptions.

Example **Download**

```php
<?php
function handleUncaughtException($e){
    // Display generic error message to the user
    echo "Opps! Something went wrong. Please try again, or contact us if the problem persists.";

    // Construct the error string
    $error = "Uncaught Exception: " . $message = date("Y-m-d H:i:s - ");
    $error .= $e->getMessage() . " in file " . $e->getFile() . " on line " . $e->getLine() . "\n";

    // Log details of error in a file
    error_log($error, 3, "var/log/exceptionLog.log");
}

// Register custom exception handler
set_exception_handler("handleUncaughtException");

// Throw an exception
throw new Exception("Testing Exception!");
?>
```

Note: An uncaught exception will always result in script termination. So if you want the script to continue executing beyond the point where the exception occurred, you must have have at least one corresponding `catch` block for each `try` block.