PHP Bugging Out and Input Validation

Fajiang Yu, fjyu@whu.edu.cn

School of Computer, Wuhan University

2017.5

Agenda

1 Bugging Out

2 Input Validation

Agenda

1 Bugging Out

2 Input Validation

Introduction

There are three basic types of runtime errors in PHP:

- Notices: These are trivial, non-critical errors that PHP encounters while executing a script – for example, accessing a variable that has not yet been defined. By default, such errors are not displayed to the user at all – although, as you will see, you can change this default behaviour.
- Warnings: These are more serious errors – for example, attempting to include() a file which does not exist. By default, these errors are displayed to the user, but they do not result in script termination.
- Fatal errors: These are critical errors – for example, instantiating an object of a non-existent class, or calling a non-existent function. These errors cause the immediate termination of the script, and PHP's default behaviour is to display them to the user when they take place.

Early Warning

```
<?php
   // initialize the $string variable
   $string = 'a string';

   // explode() a string
   // this will generate a warning or E_WARNING
   // because the number of arguments to explode() is incorrect
   explode($string);
 ?>
```

Fatal Error

```
<?php
   // call a non-existent function
   // this will generate a fatal error (E_ERROR)
   callMeJoe();
 ?>
```

Control Error Display

We can control which errors are displayed to the user, by using a built-in PHP function called `error_reporting()`.

```
<?php
 // report only fatal errors
 error_reporting(E_ERROR);

 // initialize the $string variable
 $string = 'string';
 // attempt to explode() a string
 // this will not generate a warning
 // because only fatal errors are reported
 explode($string);
 ?>
```

Control Error Display

Turn off the display of fatal errors.
The error isn't being reported doesn't mean it isn't occurring.

```
<?php
 // report no fatal errors
 error_reporting(~E_ERROR);
 // call a non-existent function
 callMeJoe();
 ?>
```

Control Error Display

- Note that there are further settings within `php.ini` that should be used on production sites. You can (and should) turn off `display_errors`, stipulate an `error_log` file and switch on `log_errors`
- It is far better – and more professional – to anticipate the likely errors ahead of time, and write defensive code that watches for them and handles them appropriately.

Rolling Your Own

There is a function called `set_error_handler()`, and it allows you to divert all PHP errors to a custom function that you've defined, instead of sending them to the default handler

```
<?php
 // define a custom error handler
 set_error_handler('oops');
 // initialize the $string variable
 $string = 'a string';
 // explode() a string
 // this will generate a warning because the number
 // of arguments to explode() is incorrect
 // the error will be caught by the custom error handler
 explode($string);
```

Rolling Your Own

```
// custom error handler
function oops($type, $msg, $file, $line, $context) {
 echo "<h1>Error!</h1>";
 echo "An error occurred while executing this script.
     Please contact the <a href=mailto:webmaster@somedomain.com>
      webmaster</a> to report this error.";
 echo "<p />";
 echo "Here is the information provided by the script:";
 echo "<hr><pre>";
 echo "Error code: $type<br />";
 echo "Error message: $msg<br />";
 echo "Script name and line number of error: $file:$line<br />";
 $variable_state = array_pop($context);
 echo "Variable state when error occurred: ";
 print_r($variable_state);
 echo "</pre><hr>";
}
?>
```

Rolling Your Own

```
<?php
 // define a custom error handler
 set_error_handler('oops');

 // initialize $string variable
 $string = 'a string';
 // this will generate a warning
 explode($string);

 // custom error handler
 function oops($type, $msg, $file, $line, $context) {
  switch ($type) {
```

Rolling Your Own

```
   // notices
   case E_NOTICE:
    // do nothing
    break;
   // warnings
   case E_WARNING:
    // report error
    print "Non-fatal error on line $line of $file: $msg <br />";
    break;
   // other
   default:
    print "Error of type $type on line $line of $file:
          $msg <br />";
    break;
  }
 }
 ?>
```

Agenda

1 Bugging Out

2 Input Validation

Empty Validation

- Submitting the form without entering any data will result in an empty record being added to the database (assuming no NOT NULL constraints on the target table). To avoid this, it's important to verify that the form does, in fact, contain valid data, and only then perform the INSERT query.
- `trim()`
- `empty()`

Empty Validation

```
trim()

if (!isset($_POST['filling']) ||
    trim($_POST['filling']) == '') {
   die("ERROR: You can't have a sandwich without a filling!");
}
else {
   $filling = mysqli_escape_string(trim($_POST['filling']));
}
```

Empty Validation

- A common mistake, especially among newbies, is to replace the `isset()` and `trim()` combination with a call to PHP's `empty()` function, which tells you if a variable is empty.
- This isn't usually a good idea, because `empty()` has a fatal flaw: it'll return true even if a variable contains the number 0.

Not My Type

```
// check if input is a number
if (!is_numeric($_POST['quantity'])) {
   die ("ERROR: Whatever you just said isn't a number!");
}

// check if input is an integer
if (intval($_POST['quantity']) != $_POST['quantity']) {
   die ("ERROR: Can't do halves, quarters or thirds...
        I'd lose my job!");
}

// check if input is in the range 1-9
if (($_POST['quantity'] < 1) || ($_POST['quantity'] > 9)) {
   die ('ERROR: I can only make between 1 and 9
        sandwiches per order!');
}
```

Not My Type

```
// check if input is of the right length
if (!(strlen($_POST['nick']) >= 6
    && strlen($_POST['nick']) <= 10)) {
   die ("ERROR: That's either too long or too short!");
}

// check if date is valid
if (!checkdate($_POST['month'], $_POST['day'],
               $_POST['year'])) {
   die("ERROR: The date $_POST['day']-$_POST['month']
       -$_POST['year'] doesn't exist!");
}
```

Not My Type

```
// check if input is of the right length
if (!(strlen($_POST['nick']) >= 6
    && strlen($_POST['nick']) <= 10)) {
   die ("ERROR: That's either too long or too short!");
}

// check if date is valid
if (!checkdate($_POST['month'], $_POST['day'],
               $_POST['year'])) {
   die("ERROR: The date $_POST['day']-$_POST['month']
       -$_POST['year'] doesn't exist!");
}

// check multi-select box
if (!is_array($_POST['toppings']) ||
    sizeof($_POST['toppings']) < 1) {
   die('You must select at least one topping for the pizza');
}
```

Thank You!
Any Questions?