Содержание

ГЛАВА 12

Отладка кода

Программы редко работают правильно с первого же раза, когда они запускаются на выполнение. В этой главе представлен ряд методик выявления и устранения ошибок в программах на PHP. Когда вы только начинаете осваивать язык PHP, ваши программы оказываются проще, чем у опытных программирующих на PHP. Но в целом устранять возникающие в них ошибки не намного проще, а для выявления и исправления этих ошибок приходится применять те же инструментальные средства и методики, которыми обычно пользуются опытные программирующие на PHP.

Управление выводом сообщений об ошибках

В вашей программе может произойти немало такого, что способно привести к генерированию интерпретатором PHP сообщения об ошибке. У вас имеется возможность выбрать место для вывода сообщений об ошибках. Такие сообщения можно, например, посылать вместе с другими выводимыми результатами выполнения программы веб-браузеру или же в журнал регистрации ошибок на веб-сервере.

Отображение сообщений об ошибках удобно сначала настроить на вывод их на экран во время разработки программы на PHP, а по окончании разработки и передачи программы в эксплуатацию — посылать сообщения об ошибках в журнал регистрации. В процессе разработки программы удобно сразу же выявлять, например, синтаксические ошибки в отдельных строках кода. Но как только программа будет признана работоспособной и пригодной для эксплуатации потребителями, появление сообщений о подобных ошибках могут их смутить.

Чтобы отображать сообщения об ошибках в окне браузера, достаточно установить значение **`On`** в директиве конфигурации `display_errors`. Если же в ней установлено значение **`Off`**, сообщения об ошибках не будут направляться браузеру для отображения в его окне. А для того чтобы ошибки в конечном итоге записывались в журнале регистрации на веб-сервере, следует установить значение **`On`** в директиве конфигурации `log_errors`.

Сообщение об ошибке, формируемое интерпретатором PHP, может быть отнесено к одной из следующих категорий.

Синтаксические ошибки. Это такие ошибки синтаксического характера в программе, как, например, отсутствие точки с запятой в конце оператора. Механизм PHP прерывает выполнение программы, когда обнаруживает синтаксическую ошибку.

Неисправимые ошибки. Это такие серьезные ошибки в программе, как, например, вызов неопределенной функции. Интерпретатор PHP прерывает выполнение программы, когда обнаруживает неисправимую ошибку.

Предупреждения. Это рекомендации, которые интерпретатор PHP дает по поводу какого-нибудь подозрительного поведения программы, хотя ее выполнение не прерывается. Предупреждение может, например, появиться при вызове функции с неверным количеством аргументов.

Замечания. Это рекомендации, которые интерпретатор PHP дает по поводу какого-нибудь нарушения принятых норм программирования. Замечание может, например, появиться в том случае, если значение переменной выводится на экран без предварительной ее инициализации.

Строгие замечания и предупреждения об употреблении устаревших языковых средств. Это предостережения интерпретатора PHP по поводу выбранного стиля программирования или применения языковых средств, которые выйдут из употребления в последующих версиях PHP.

Совсем не обязательно, что вы будете извещены об ошибках всех перечисленных выше категорий. Директива конфигурации `error_reporting` определяет категории ошибок, о которых извещает интерпретатор PHP. По умолчанию в ней устанавливается значение **`E_ALL & ~E_NOTICE & ~E_DEPRECATED`**, которое предписывает интерпретатору PHP извещать обо всех категориях ошибок, кроме замечаний и предупреждений об употреблении устаревших языковых средств. В приложении А к данной книге поясняется, что обозначают знаки **`&`** и **`~`** в значениях директив конфигурации.

В языке PHP определен ряд констант для установки такого значения в директиве конфигурации `error_reporting`, чтобы интерпретатор PHP извещал об ошибках только определенных категорий. Эти константы перечислены ниже.

- `E_ALL` (все ошибки)
- `E_PARSE` (синтаксические ошибки)
- `E_ERROR` (неисправимые ошибки)
- `E_WARNING` (предупреждения)
- `E_NOTICE` (замечания)
- `E_STRICT` (строгие замечания, до версии PHP 7.0.0)

Строгие замечания появились в версии PHP 5 и поэтому не включены в язык PHP до версии 5.4.0. Чтобы интерпретатор PHP прежних версий посчитал нечто обнаруженное в программе возможной ошибкой и соответственно отреагировал, следует установить значение **`E_ALL | E_STRICT`** в директиве конфигурации `error_reporting`.

Устранение синтаксических ошибок

На самом деле интерпретатор PHP весьма разборчив и не очень многословен. Если вы не завершите оператор точкой с запятой или начнете символьную строку одиночной кавычкой, а завершите ее двойной кавычкой, интерпретатор PHP не выполнит такой код. Вместо этого он выдаст сообщение о синтаксической ошибке, оставляя вас перед малоприятной перспективой заниматься отладкой написанного вами исходного кода.

Весь исходный код программы должен быть набран таким образом, чтобы интерпретатор PHP нормально воспринял его. И это может оказаться самой неприятной обязанностью для начинающего программировать. Упростить этот процесс помогает написание программ в редакторе, способном анализировать исходный код PHP. Если уведомить такой редактор, что в нем редактируется программа на PHP, он активизирует специальные средства, упрощающие программирование.

К числу таких специальных средств относится *выделение синтаксических конструкций*. Оно изменяет цвет отдельных частей исходного кода программы в зависимости от назначения. Например, символьные строки выделяются розовым цветом, ключевые слова вроде `if` и `while` — голубым, комментарии — серым, а переменные — черным. Выделение синтаксических конструкций упрощает обнаружение, например, отсутствующих кавычек, закрывающих символьные строки, которые продолжаются до конца файла (или кавычки, следующей далее в исходном коде программы).

Еще одним удобным средством является *автоматическое закрытие кавычек и скобок*, помогающее правильно уравновесить кавычки и скобки. Когда вы вводите закрывающую фигурную скобку `}`, редактор автоматически выделяет соответствующую открывающую фигурную скобку `{`. В разных редакторах это делается по-разному, но, как правило, курсор на позиции открывающей фигурной скобке (`{`) начинает мигать или обе фигурные скобки (`{` и `}`) выделяются полужирным. Такое поведение удобно для ввода парных знаков препинания, в том числе одиночных и двойных кавычек, ограничивающих символьные строки, а также круглых, квадратных и фигурных скобок.

Такие редакторы отображают номера строк в исходных файлах программ. Когда вы получаете от интерпретатора PHP сообщение о синтаксической ошибке, например, в строке кода 35 вашей программы, то знаете, где искать ошибку. В табл. 12.1 перечислены текстовые редакторы, поддерживающие написание программ на PHP. Цены на них указаны в долларах США на момент написания данной книги.

Таблица 12.1. Текстовые редакторы, поддерживающие написание программ на PHP

Наименование	URL	Цена в долларах
PhpStorm	`https://www.jetbrains.com/phpstorm`	89
NetBeans	`https://netbeans.org`	Бесплатно
Zend Studio	`http://www.zend.com/en/products/studio`	89
Eclipse + PDT	`http://www.eclipse.org/pdt`	Бесплатно
Sublime Text	`http://www.sublimetext.com`	70
Emacs	`http://ergoemacs.org/emacs/which_emacs.html`	Бесплатно
Vim	`http://vim.wikia.com/wiki/Where_to_download_Vim`	Бесплатно

Программные продукты PhpStorm, NetBeans, Zend Studio и Eclipse + PDT в большей степени соответствуют традиционным интегрированным средам разработки (ИСР), тогда как редакторы Sublime Text, Emacs и Vim — традиционным текстовым редакторам, хотя их нетрудно настроить с помощью подключаемых модулей, помогающих распознавать исходный код PHP. Наиболее пригодными для написания программ на PHP считаются редакторы PhpStorm и Zend Studio, тогда как остальные редакторы позволяют программировать на многих других языках. Все коммерчески доступные редакторы, перечисленные в табл. 12.1, имеют бесплатные периоды оценивания, поэтому можете опробовать их, чтобы выбрать наиболее подходящий для вас.

Синтаксические ошибки возникают в тех случаях, когда интерпретатор PHP обнаруживает в программе нечто неожиданное. Обратимся к примеру 12.1, где демонстрируется неверно написанная программа.

Пример 12.1. Синтаксическая ошибка

```
<?php
if $logged_in) {
    print "Welcome, user.";
}
?>
```

Если попытаться выполнить код из примера 12.1, интерпретатор PHP выдаст следующее сообщение об ошибке:

```
PHP Parse error: syntax error, unexpected '$logged_in' (T_VARIABLE),
expecting '(' in welcome.php on line 2

[ Ошибка синтаксического анализа кода PHP: синтаксическая ошибка,
 неожиданная переменная '$logged_in' (T_VARIABLE) вместо ожидавшейся
 круглой скобки '(' в строке кода 2 исходного файла welcome.php ]
```

Приведенное выше сообщение об ошибке означает, что в строке 2 указанного исходного файла интерпретатор PHP предполагал обнаружить открывающую круглую скобку, а выявил переменную `$logged_in`, обозначаемую как *лексема* `T_VARIABLE`. С помощью лексем в интерпретаторе PHP выражаются различные основополагающие

части программ. Когда интерпретатор PHP читает исходный текст программы, он преобразует его в перечень лексем. И там, где в исходном тексте программы введена переменная, интерпретатор PHP добавляет лексему `T_VARIABLE` в список.

Таким образом, интерпретатор PHP сообщает, что при чтении строки кода 2 была обнаружена переменная `$logged_in` там, где предполагалась открывающая скобка. Глядя на строку кода 2 из примера 12.1, нетрудно выявить причину подобной ошибки: отсутствие открывающей скобки, с которой должно начинаться проверочное выражение в условном операторе `if()`. Обнаружив ключевое слово **`if`**, интерпретатор PHP предполагал обнаружить далее открывающую скобку (**`(`**), с которой начинается проверочное выражение, а вместо нее он выявил переменную `$logged_in`.

Список всех лексем, употребляемых интерпретатором PHP, можно найти в оперативно доступном руководстве по языку PHP (`http://www.php.net/tokens`). Эти лексемы могут появляться в сообщениях об ошибках, выдаваемых интерпретатором PHP.

Но коварство синтаксических ошибок состоит в том, что номер строки кода, упоминаемый в сообщении об ошибке, зачастую не соответствует тому месту, где фактически возникает ошибка. Именно такой случай возникновения ошибки в коде и приведен в примере 12.2.

Пример 12.2. Труднообъяснимая ошибка

```
<?php
$first_name = "David';
if ($logged_in) {
    print "Welcome, $first_name";
} else {
    print "Howdy, Stranger.";
}
?>
```

При попытке выполнить код из примера 12.2 интерпретатор PHP выдаст следующее сообщение об ошибке:

```
PHP Parse error: syntax error, unexpected 'Welcome' (T_STRING)
in trickier.php on line 4

[ Ошибка синтаксического анализа кода PHP: синтаксическая ошибка,
 неожиданная символьная строка 'Welcome' (T_STRING)
 в строке кода 4 исходного файла trickier.php ]
```

В этом сообщении ошибка указана там, где ее на самом деле нет. Как бы тщательно вы ни пытались обнаружить указанную ошибку (наличие символьной строки `'Welcome'`) в строке кода 4, вам так и не удастся этого сделать. Строка кода `print "Welcome, $first_name";` составлена совершенно правильно, т.е. указанная в ней символьная строка заключена в двойные кавыки, а завершается эта строка кода точкой с запятой.

На самом деле искомая ошибка присутствует в строке кода 2 из примера 12.2. В частности, символьная строка, присваиваемая переменной `$first_name`, начинается с двойной кавычки, а оканчивается одиночной кавычкой. Читая строку кода 2, интерпретатор PHP обнаруживает двойную кавычку и считает, что далее следует символьная строка. И поэтому он воспринимает весь остальной исходный код как содержимое символьной строки до тех пор, пока не встретится следующая (неэкранированная) двойная кавычка. Следовательно, интерпретатор PHP пропускает одиночную кавычку в строке кода 2, продолжая читать исходный код до тех пор, пока не встретится первая же двойная кавычка в строке 4. Обнаружив эту двойную кавычку, он интерпретирует ее как завершение символьной строки. Таким образом, все, что следует после этой двойной кавычки, он посчитает новой командой или оператором. Но ведь после этой двойной кавычки следует исходный код `Welcome, $first_name";`, который не имеет никакого смысла для интерпретатора PHP. Ведь он ожидает встретить точку с запятой сразу в конце оператора или же точку для сцепления только что определенной символьной строки со следующей символьной строкой. Но исходный код `Welcome, $first_name";` похож на неограниченную символьную строку, находящуюся не на своем месте, и поэтому интерпретатор PHP выдает сообщение о синтаксической ошибке в строке кода 4.

Представьте, что вы несетесь по улицам Манхеттена со сверхзвуковой скоростью. Тротуар на 35-й улице имеет выбоины, о которые вы спотыкаетесь. Но поскольку вы движетесь так быстро, то приземляетесь на 39-й улице, разбившись до крови о мостовую и выпустив на нее свои кишки. Подойдя к вам, полицейский из службы безопасности дорожного движения воскликнет: “Эй! Происшествие на 39-й улице! Кто-то испачкал тротуар своими внутренностями!”

Аналогичным образом поступает в данном случае интерпретатор PHP. Номер строки кода, в которой он обнаруживает нечто неожиданное, не всегда отвечает именно той строке, где фактически возникает ошибка.

Получив от интерпретатора PHP сообщение о синтаксической ошибке, проанализируйте сначала строку кода, указанную в данном сообщении. Проверьте соблюдение самых основных правил синтаксиса, в том числе наличие точки с запятой в конце оператора. Если эта строка кода написана правильно, проанализируйте исходный код программы на несколько строк вперед и назад в поисках фактической ошибки. Обратите особое внимание на те знаки препинания, которые должны следовать парами: одиночные или двойные кавычки, в которые заключаются символьные строки; круглые скобки в вызовах функций или проверочных выражениях; квадратные скобки в элементах массивов; а также фигурные скобки в блоках кода. При этом количество открывающих (`(`, `[` и `{`) и закрывающих (`)`, `]` и `}`) знаков препинания должно совпадать.

В подобных случаях настоящую помощь оказывает редактор, поддерживающий написание программ на PHP. Выделяя синтаксис и автоматически закрывая кавычки и скобки, такой редактор может подсказать, где искать ошибки в набираемом исходном коде программы, облегчая их выявление.

Проверка данных в программе

Избавившись от синтаксических ошибок, возможно, придется потрудиться еще немного, прежде чем завершить отладку исходного кода программы. Ведь программа может быть написана синтаксически безупречно, но логически неверно. Как и предложение, составленное грамматически верно, но не имеющее никакого логического смысла, программа, в которой интерпретатор PHP не выявит никаких ошибок, не делает то, что он нее требуется.

Выявление и исправление логических ошибок в тех частях программы, которые ведут себя не так, как предполагалось, является немалой долей программирования в целом. Конкретные подходы к диагностике и анализу отдельных ситуаций сильно зависят от характера логических ошибок, которые требуется устранить. В этом разделе демонстрируются две методики исследования происходящего в программе на PHP. Первая из них более простая и состоит в добавлении операторов вывода отладочной информации, хотя для этого придется видоизменить исходный код программы, что неприемлемо для условий эксплуатации, где обычные пользователи не должны видеть результаты отладки программы. А вторая методика состоит в применении отладчика, который требует дополнительных затрат труда для своей настройки, но в то же время обеспечивает бо́льшую гибкость для проверки программы во время ее выполнения.

Добавление операторов вывода отладочной информации

Если программа ведет себя странно, в нее можно ввести контрольные точки для вывода значений переменных на экран. Это даст возможность выяснить места, где поведение программы отличается от предполагаемого. В примере 12.3 демонстрируется программа, неправильно пытающаяся вычислить общую стоимость нескольких товаров.

Пример 12.3. Логически неверно написанная программа

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    $total_price = $price * $tax_rate;
}

printf('Total price (with tax): $%.2f', $total_price);
```

Код программы из примера 12.3 действует неверно, выводя на экран следующий результат:

```
Total price (with tax): $13.50
```

Общая стоимость товаров должна быть не менее 20 долларов. Что же не так в программе из примера 12.3? Чтобы выяснить это, можно, например, ввести в цикл `foreach()` строки кода, где значение переменной `$total_price` выводится до и после его изменения. Это даст возможность выяснить причину ошибки в математических расчетах. В примере 12.4 исходный код из примера 12.3 аннотирован рядом диагностических операторов `print`.

Пример 12.4. Логически неверно написанная программа с выводом отладочной информации на экран

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    print "[before: $total_price]";
    $total_price = $price * $tax_rate;
    print "[after: $total_price]";
}

printf('Total price (with tax): $%.2f', $total_price);
```

При выполнении кода из примера 12.4 на экран выводится следующий результат:

```
[before: 0][after: 6.426][before: 6.426][after: 3.24][before: 3.24]
[after: 13.5]Total price (with tax): $13.50
```

Проанализировав результаты вывода отладочной информации в программе из примера 12.4, можно обнаружить, что значение переменной `$total_price` не увеличивается на каждом шаге цикла `foreach()`. Более тщательный анализ кода этой программы позволяет сделать вывод, что строку кода

```
$total_price = $price * $tax_rate;
```

следует заменить на такую:

```
$total_price += $price * $tax_rate;
```

Это означает, что вместо операции присваивания (**=**) в данной строке кода следует использовать операцию инкрементирования с присваиванием (**+=**).

Чтобы включить выводимую отладочную информацию в массив, достаточно воспользоваться функцией `var_dump()`, которая выводит на экран все элементы данного массива. Результат, выводимый из функции `var_dump()`, следует заключить в дескрипторы `<pre>` и `</pre>`, чтобы правильно представить его в формате HTML для отображения в окне браузера. Так, в примере 12.5 содержимое всех параметров переданной на обработку форму выводится с помощью функции `var_dump()`.

Редактирование нужного исходного файла

Если вы вносите изменения в программу во время ее отладки, но не видите, как эти изменения отражаются на ее поведении при повторной загрузке страницы в веб-браузер, отредактируйте нужный исходный файл. Поработав с локальной копией программы, но собираясь загрузить ее в браузер из удаленного сервера, скопируйте измененный исходный файл на этот сервер, прежде чем перезагружать страницу.

Чтобы согласовать редактируемый исходный файл со страницей, отображаемой в окне браузера, можно, например, временно ввести в самом начале исходного кода программы следующую строку кода с вызовом функции **`die()`**:

```
die('This is: ' . __FILE__);
```

Специальная константа `__FILE__` содержит имя исполняемого файла. Таким образом, при загрузке PHP-страницы, в самом начале которой находится приведенная выше строка кода, по URL вроде следующего:

```
http://www.example.com/catalog.php
```

в окне браузера появится лишь такая строка:

```
This is: /usr/local/htdocs/catalog.php
```

По результатам выполнения функции **`die()`** в окне браузера можно выяснить, какой именно исходный файл редактируется в настоящий момент. По окончании редактирования удалите вызов функции **`die()`** из исходного файла программы и продолжите ее отладку.

Пример 12.5. Вывод всех параметров переданной на обработку формы с помощью функции `var_dump()`

```
print '<pre>';
var_dump($_POST);
print '</pre>';
```

Отладочные сообщения, безусловно, информативны, но они могут нарушать и отвлекать от нормального вывода информации на обычной странице. Чтобы направить отладочные сообщения в журнал регистрации ошибок на сервере, следует воспользоваться функцией `error_log()` вместо оператора `print`. В примере 12.6 приведена та же самая программа, что и в примере 12.4, но в ней для отправки диагностических сообщений в журнал регистрации ошибок на сервере применяется функция `error_log()`.

Пример 12.6. Логически неверно написанная программа с выводом отладочной информации в журнал регистрации ошибок

```
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    error_log("[before: $total_price]");
    $total_price = $price * $tax_rate;
    error_log("[after: $total_price]");
}

printf('Total price (with tax): $%.2f', $total_price);
```

При выполнении кода из примера 12.6 на экран выводится только общая стоимость товаров, как показано ниже.

```
Total price (with tax): $13.50
```

В то же время следующие строки с отладочной информацией направляются в журнал регистрации ошибок на сервере:

```
[before: 0]
[after: 6.426]
[before: 6.426]
[after: 3.24]
[before: 3.24]
[after: 13.5]
```

Конкретное местоположение журнала регистрации ошибок на веб-сервере зависит от его конфигурации. Так, если применяется веб-сервер Apache, местоположение журнала регистрации ошибок определяется в директиве `ErrorLog` конфигурации этого веб-сервера

Функция `var_dump()` сама выводит информацию, поэтому следует принять дополнительные меры, чтобы направить выводимую отладочную информацию в журнал регистрации ошибок аналогично буферизации вывода, обсуждавшейся в конце раздела "Причины для размещения вызовов функций `setcookie()` и `session_start()` вначале страницы" главы 10. Функцию `var_dump()` следует вызывать в промежутке между вызовами функций, временно приостанавливающих и возобновляющих вывод информации, как показано в примере 12.7.

Пример 12.7. Отправка всех параметров из переданной на обработку формы в журнал регистрации ошибок с помощью функции `var_dump()`

```
// задержать вывод информации
ob_start();
// вызвать функцию var_dump(), как обычно
var_dump($_POST);
```

```
// сохранить в переменной $output всю выводимую информацию,
// сформированную с момента вызова функции ob_start()
$output = ob_get_contents();
// вернуться к обычному выводу информации
ob_end_clean();
// отправить содержимое переменной $output
// в журнал регистрации ошибок
error_log($output);
```

Функции `ob_start()`, `ob_get_contents()` и `ob_end_clean()` определяют в примере 12.7 порядок формирования интерпретатором PHP выводимой информации. В частности, функция `ob_start()` дает интерпретатору PHP команду не выводить пока что никакой информации, а накапливать во внутреннем буфере все, что предназначается для вывода. А когда вызывается функция `var_dump()`, интерпретатор PHP, выполняя команду функции `ob_start()`, направляет выводимую информацию во внутренний буфер. Функция `ob_get_contents()` возвращает содержимое внутреннего буфера. А поскольку с момента вызова функции `ob_start()` информацию выводила только функция `var_dump()`, то эта информация размещается в переменой `$output`. Далее функция `ob_end_clean()` отменяет команду, которая была дана интерпретатору PHP функцией `ob_start()`, предписывая ему вернуться в нормальный режим вывода информации. И, наконец, функция `error_log()` направляет содержимое переменной `$output` (в данном случае — отладочную информацию, выводимую функцией `var_dump()`) в журнал регистрации ошибок на веб-сервере.

Применение отладчика

Методика вывода и регистрации отладочной информации, описанная в предыдущем разделе, проста в употреблении. Но поскольку она требует видоизменения исходного кода программы, то ее нельзя применять в условиях эксплуатации, где обычные пользователи не должны видеть выводимую отладочную информацию. Кроме того, прежде чем запускать программу на выполнение, необходимо решить, какую именно информацию следует выводить на экран или в журнал регистрации ошибок. Чтобы вывести значение интересующей переменной, приходится снова видоизменять программу, вводя дополнительный код, а затем перезапускать ее.

Все эти затруднения устраняются, если исследовать работу программы с помощью специального отладчика. Отладчик позволяет анализировать работу программы во время ее выполнения, просматривая значения переменных и порядок вызова функций. Для этого не нужно вносить никаких изменений в исходный код программы, а только отдельно настроить режим ее отладки.

Для проверки работоспособности программ на PHP имеется несколько отладчиков, а многие редакторы, перечисленные в табл. 12.1, содержат встроенные отладчики или вполне совместимы с внешними отладчиками для проверки работоспособности

программы на PHP, которую можно выполнять непосредственно в редакторе. В этом разделе рассматривается методика проверки работоспособности программ с помощью отладчика phpdbg, входящего в состав PHP.

Отладчик phpdbg входит в состав PHP, начиная с версии 5.6, но ваша установка интерпретатора PHP может быть не настроена на включение в его состав данного отладчика. Если в вашей системе отсутствует программа `phpdbg`, проверьте (или обратитесь к своему системному администратору за помощью проверить), что ваша установка PHP была создана с параметром **`--enable-phpdbg`**.

Отладчик `Xdebug` (`https://xdebug.org/`) является эффективным и полноценным диагностическим средством. Он может взаимодействовать с редакторами и ИСР по определенному протоколу, но не содержит свой собственный, удобный в употреблении клиент. Отладчик `Xdebug` доступен бесплатно.

Отладчик Zend Debugger входит в состав ИСР Zend Studio (`http://www.zend.com/en/products/studio`). В нем применяется собственный протокол для взаимодействия с ИСР Zend Studio, но с ним могут взаимодействовать и другие ИСР, например PhpStorm.

Чтобы начать сеанс отладки в отладчике phpdbg, достаточно запустить проверяемую программу на выполнение с параметром **`-e`**, обозначающим отлаживаемую программу, как показано ниже.

```
phpdbg -e broken.php
```

В ответ на эту команду отладчик выдаст следующее сообщение:

```
Welcome to phpdbg, the interactive PHP debugger, v0.4.0]
To get help using phpdbg type "help" and press enter
[Please report bugs to <http://github.com/krakjoe/phpdbg/issues>]
[Successful compilation of broken.php]

[ Добро пожаловать в phpdbg — диалоговый отладчик кода PHP,
    версия v0.4.0]
  Для получения справки введите команду "help" и нажмите
    клавишу <enter>
  [О выявленных программных ошибках сообщайте по следующему адресу:
    <http://github.com/krakjoe/phpdbg/issues>]
  [Исходный файл broken.php успешно скомпилирован broken.php] ]
```

Это означает, что отладчик phpdbg прочитал исходный файл `broken.php`, выбрал из него команды и готов выполнить их автоматически. Прежде всего необходимо установить *точку прерывания* в отлаживаемой программе. Этим отладчику phpdbg предписывается остановить выполнение программы в определенном ее

месте. Как только отладчик phpdbg остановит выполнение программы в точке прерывания, можно приступить к анализу ее внутреннего состояния. В строке кода 7 из рассматриваемого здесь примера неверно написанной программы переменная `$total_price` получает свое значение в теле цикла, поэтому выполнение данной программы целесообразно прервать именно в этой строке кода, введя следующую команду:

```
prompt> break 7
```

В приведенной выше командной строке следует ввести лишь то, что выделено курсивом и полужирным, т.е. непосредственно команду, а не подсказку `prompt>`. Команда **`break 7`** сообщает отладчику phpdbg приостановить выполнение программы, как только будет достигнута строка кода 7. В ответ отладчик phpdbg выдаст следующее сообщение:

```
[Breakpoint #0 added at broken.php:7]

[ Точка прерывания #0 введена в строке
  кода 7 исходного файла broken.php ]
```

Чтобы начать отладку программы, необходимо дать отладчику phpdbg следующую команду на ее выполнение:

```
prompt> run
```

Отладчик начинает выполнение программы построчно от первой до седьмой строки кода, где установлена точка прерывания. В этой точке отладчик phpdbg выдаст следующее сообщение:

```
[Breakpoint #0 at broken.php:7, hits: 1]
>00007:     $total_price = $price * $tax_rate;
 00008: }
 00009:
```

А теперь можно ввести *контрольную точку* для проверки переменной `$total_price`. Этим отладчику phpdbg предписывается приостанавливать выполнение программы всякий раз, когда требуется проверить изменение значения в переменной `$total_price`. Именно это и требуется для диагностики логической ошибки, поскольку в переменной `$total_price` устанавливается не то значение, которое предполагалось. Контрольная точка вводится по команде **`watch`** следующим образом:

```
prompt> watch $total_price
```

Отладчик phpdbg отвечает следующим сообщением:

```
[Set watchpoint on $total_price]

[ Установлена контрольная точка для переменной $total_price ]
```

А теперь, когда контрольная точка установлена, точка прерывания в строке кода 7 больше не требуется. Поэтому ее можно удалить по команде **`break del`** следующим образом:

```
prompt> break del 0
```

Этим отладчику phpdbg предписывается удалить установленную первую точку прерывания (аналогично индексированию элементов массива в PHP, отладчик phpdbg начинает нумерацию элементов отладки с **`0`**, а не с **`1`**). Отладчик phpdbg подтверждает удаление точки прерывания следующим сообщением:

```
[Deleted breakpoint #0]

[ Удалена точка прерывания #0 ]
```

Итак, все готово для того, чтобы продолжить выполнение программы и приостановить его всякий раз, когда изменяется значение переменной `$total_price`. Приведенная ниже команда **`continue`** предписывает отладчику phpdbg продолжить выполнение программы.

```
prompt> continue
```

Отладчик phpdbg начинает выполнение программы. Первыми выполняются команды из строки кода 7, где изменяется значение в переменной `$total_price`. После этого выполнение программы немедленно останавливается, и отладчик phpdbg выдает следующее сообщение:

```
[Breaking on watchpoint $total_price]
Old value: 0
New value: 6.426
>00007:      $total_price = $price * $tax_rate;
 00008: }
 00009:

[ Прерывание в контрольной точке $total_price]
  Прежнее значение: 0
  Новое значение: 6.426 ]
```

Это очень удобно, поскольку можно ясно видеть, как значение переменной `$total_price` изменяется в коде с **`0`** на **`6.426`**. Чтобы выяснить, что произойдет дальше, достаточно ввести команду **`continue`** снова, как показано ниже.

```
prompt> continue
```

После этого выполнение программы остановится снова, а отладчик phpdbg выдаст следующее сообщение:

```
[Breaking on watchpoint $total_price]
Old value: 6.426
New value: 3.24
```

```
>00007:     $total_price = $price * $tax_rate;
 00008: }
 00009:
```

При выполнении цикла в строке кода 7 значение переменной `$total_price` изменяется с **`6.426`** на **`3.24`**. Но ведь это совершенно неверно, поскольку значение переменной `$total_price` должно увеличиться! Выполнение программы можно продолжить дальше по следующей команде:

```
prompt> continue
```

И в последний раз значение переменной `$total_price` изменяется следующим образом:

```
[Breaking on watchpoint $total_price]
Old value: 3.24
New value: 13.5
>00007:     $total_price = $price * $tax_rate;
 00008: }
 00009:
```

На этот раз значение данной переменной увеличивается до **`13.5`**. И в последний раз команда **`continue`** выдается для завершения программы, как показано ниже.

```
prompt> continue
```

Отладчик phpdbg продолжает выполнение программы и фактически выдает окончательный результат ее выполнения:

```
Total price (with tax): $13.50
[$total_price was removed, removing watchpoint]
[Script ended normally]

[ Общая стоимость (с учетом налога): $13.50
  [Переменная $total_price удалена, удаляется
    контрольная точка]
  [Сценарий завершился нормально] ]
```

Когда отладчик phpdbg приостанавливает выполнение программы в контрольной точке во второй раз, становится очевидно, что логическая ошибка кроется в порядке расчета значения переменной `total_price`. К такому же выводу удалось прийти в результате анализа отладочной информации, выводимой по методике, описанной в предыдущем разделе.

Особенности синтаксиса вводимых команд (или выбираемых элементов ГПИ) могут отличаться в разных отладчиках или ИСР, но основной принцип действия остается прежним: отладчик выполняет программу под особым контролем. Выполнение программы можно приостанавливать в разных выбираемых для этой цели местах, анализируя внутреннее состояние программы в момент приостановки ее выполнения.

Обработка неперехватываемых исключений

В разделе "Индикация ошибок с помощью исключений" главы 6 пояснялись основы обработки исключений в PHP, а в примере 6.8 было демонстрировано, что произойдет, если исключение будет сгенерировано, но не перехвачено. В этом случае выполнение программы на PHP прервется, а интерпретатор PHP выведет сведения об ошибке и результат трассировки стека (т.е. перечень функций, которые по очереди вызывались в тот момент, когда было прервано выполнение программы).

Несмотря на то что любой код, который способен сгенерировать исключение, следует всегда заключать в блоки операторов `try/catch`, на практике это не всегда удается сделать, чтобы идеально удовлетворить целям обработки исключений. Можно, например, воспользоваться сторонней библиотекой, даже не подозревая об исключениях, которые она генерирует, или же совершить ошибку и забыть о ситуации, в которой прикладной код может сгенерировать исключение. Для подобных случаев в PHP предоставляется возможность указать специальный обработчик исключений, который будет вызван, если исключение не обрабатывается в прикладном коде. Такой обработчик исключений служит удобным местом для регистрации сведений об исключении и предоставления пользователю программы более удобной информации, чем результат трассировки стека.

Чтобы воспользоваться специальным обработчиком тех исключений, которые иначе не обрабатываются, необходимо сделать следующее.

1. Написать функцию, которая обработает исключение. Эта функция принимает в качестве единственного аргумента исключение, передаваемое ей на обработку.
2. Вызвать функцию `set_exception_handler()`, чтобы сообщить интерпретатору PHP о функции, обрабатывающей исключение.

В примере 12.8 демонстрируется установка обработчика исключений, выводящего удобное для восприятия пользователем сообщение об ошибке и регистрирующего дополнительные сведения о возникшем исключении.

Пример 12.8. Установка специализированного обработчика исключений

```
function niceExceptionHandler($ex) {
    // выдать удобное для восприятия пользователем
    // сообщение об ошибке
    print "Sorry! Something unexpected happened.
           Please try again later.";
    // зарегистрировать дополнительные сведения об исключении
    // для последующего анализа системным администратором
    error_log("{$ex->getMessage()} in {$ex->getFile()}
               @ {$ex->getLine()}");
    error_log($ex->getTraceAsString());
}
```

```
set_exception_handler('niceExceptionHandler');

print "I'm about to connect to a made up, pretend,
       broken database!\n";

// Имя источника данных, предоставленное конструктору
// класса PDO, не обозначает достоверную базу данных или
// параметры подключения к ней, и поэтому конструктор данного
// класса сгенерирует исключение
$db = new PDO('garbage:this is obviously not going to work!');

print "This is not going to get printed.";
```

В функции `niceExceptionHandler()` из примера 12.8 оператор `print` служит для вывода простого сообщения об ошибке, удобного для восприятия пользователем, а функция `error_log()` вызывается вместе с методами для объекта типа `Exception` с целью зарегистрировать дополнительные сведения об исключении для последующего их анализа. Если вызвать функцию `set_exception_handler()` со строковым значением **`niceExceptionHandler`** в качестве аргумента, то интерпретатору PHP будет предписано направлять неперехватываемые исключения на обработку функции `niceExceptionHandler()`.

При выполнении кода из примера 12.8 на экран выводится следующий результат:

```
I'm about to connect to a made up, pretend, broken database!
Sorry! Something unexpected happened. Please try again later.

[ Попытка подключиться к вымышленной, ненастоящей, нарушенной
    базе данных!
 Извините, но произошло нечто непредвиденное. Попробуйте
 сделать еще одну попытку! ]
```

А зарегистрированы будут следующие дополнительные сведения:

```
could not find driver in exception-handler.php @ 17
#0 exception-handler.php(17):
    PDO->__construct('garbage:this is...')
#1 {main}

[ не удалось найти драйвер в исходном файле exception-handler.php @ 17 ]
```

Благодаря этому исключается возможная утечка секретных сведений (например, учетных данных или путей к файлам базы данных), которые пользователь мог бы почерпнуть из технических подробностей возникшей ошибки, если бы они были выведены на экран. Вместо этого они сохраняются в журнале регистрации ошибок для последующего анализа.

Специальный обработчик исключений не препятствует прерыванию программы после обработки исключения. Как только обработчик исключений завершит свое выполнение, завершится и сама программа. Именно поэтому в примере 12.8 строка кода с сообщением `"This is not going to get printed."` (Это не предполагается для вывода) так и не выполняется.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Настройка отображения в окне веб-браузера, вывода в журнал регистрации ошибок или того и другого.
- Конфигурирование интерпретатора PHP на уровне сообщения об ошибках.
- Извлечение выгод из текстового редактора, поддерживающего написание программ на PHP.
- Расшифровка сообщений о синтаксических ошибках.
- Выявление и устранение синтаксических ошибок.
- Вывод отладочной информации с помощью оператора `print` и функций `var_dump()` и `error_log()`.
- Отправка отладочной информации, выводимой функцией `var_dump()`, в журнал регистрации ошибок с помощью функций буферизации.
- Проверка работоспособности программы с помощью отладчика во время ее выполнения.
- Обработка исключений, которые нельзя перехватить в каком-нибудь другом коде.

Упражнения

1. В следующей программе содержится синтаксическая ошибка:

```
<?php
$name = 'Umberto';
function say_hello() {
    print 'Hello, ';
    print global $name;
}
say_hello();
?>
```

Не выполняя программу в интерпретаторе PHP, выясните, о какой ошибке должен сообщить этот интерпретатор при попытке выполнить программу. Какие изменения следует внести в данную программу, чтобы она выполнялась правильно и выводила на экран сообщение "Hello, Umberto"?

2. Видоизмените ответ на задание написать функцию validate_form() в упражнении 3 из главы 7 таким образом, чтобы она выводила имена и значения всех параметров из переданной на обработку формы в журнал регистрации ошибок на веб-сервере.

3. Видоизмените ответ на задание в упражнении 4 из главы 8 таким образом, чтобы воспользоваться специальной функцией обработки ошибок обращения к базе данных, выводящей разные сообщения в окно веб-браузера и журнал регистрации ошибок на веб-сервере. Эта функция обработки ошибок должна совершать выход из программы после вывода сообщений об ошибках.

4. Приведенная ниже программа предназначена для вывода в алфавитном порядке списка всех посетителей ресторана из таблицы по заданию в упражнении 4 из главы 8. Выявите и устраните имеющиеся в ней ошибки.

```
<?php
// подключиться к базе данных
try {
    $db = new PDO('sqlite::/tmp/restaurant.db');
} catch ($e) {
    die("Can't connect: " . $e->getMessage());
}
// организовать обработку исключений
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// установить режим извлечения строк из таблицы в виде массивов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
// получить массив наименований блюд из базы данных
$dish_names = array();
$res = $db->query('SELECT dish_id,dish_name FROM dishes');
foreach ($res->fetchAll() as $row) {
    $dish_names[ $row['dish_id']]] = $row['dish_name'];
}
$res = $db->query('SELECT ** FROM customers ORDER BY phone DESC');
$customers = $res->fetchAll();
if (count($customers) = 0) {
    print "No customers.";
} else {
    print '<table>';
    print '<tr><th>ID</th><th>Name</th><th>Phone</th>
           <th>Favorite Dish</th></tr>';
    foreach ($customers as $customer) {
        printf("<tr><td>%d</td><td>%s</td>
```

```
                    <td>%f</td><td>%s</td>
                 </tr>\n",
                $customer['customer_id'],
                htmlentities($customer['customer_name']),
                $customer['phone'],
                $customer['favorite_dish_id']);
    }
    print '</table>';
?>
```