ОПЕРАТОРЫ УПРАВЛЕНИЯ

В этой главе...

Глава 3

Операторы управления позволяют выполнять блоки кодов после анализа условий и повторять блоки сценариев, что упрощает и повышает эффективность сценариев. В этой главе вы ознакомитесь с операторами принятия решений `if` и `switch`, а также с операторами организации циклов `for` и `while`.

3.1. Оператор `if`

На рис. 3.1 показана одна форма оператора `if`.

```
if (expression1)
{
    Этот блок выполняется, если выражение expression1 имеет значение "Истина".
}
elseif (expression2)
{
    Этот блок выполняется, если выражение expression2 имеет значение "Истина".
}
else
{
    Этот блок выполняется, если ни выражение expression1,
    ни выражение expression2 не имеют значение "Истина".
}
```

Рис. 3.1. Оператор `if`

Оператор `if` выполняет блок операторов, если выражение, заключенное в скобки, "истинно"; в противном случае этот блок операторов не выполняется. Это может быть один оператор с последующей точкой с запятой. Обычно этот блок представляет собой сложный оператор, ограниченный фигурными скобками. Сразу же после этого оператора может следовать оператор `else`, содержащий свой собственный оператор,

и он также может быть простым или сложным. Он выполняется только тогда, когда предыдущее выражение дает значение "ложь". Между операторами `if` и `else` может быть размещено сколько угодно операторов `elseif`. Каждый оператор `elseif` обрабатывается по очереди, и при получении ложного результата управление передается следующему такому оператору. Если оператор `elseif` дает значение `true`, то остальная часть кода большого оператора `if` не выполняется, т.е. язык PHP допускает только одно совпадение. В листинге 3.1 показана работа оператора `if-elseif-else`.

Листинг 3.1. Оператор `if-elseif-else`

```
<?php
     $name = "Леон";
     if($name == "")
     {
          print("У вас нет имени.");
     }
     elseif(($name == "леон") OR ($name == "Леон"))
     {
          print("Привет, Леон!");
     }
     else
     {
          print("Ваше имя '$name'.");
     }
?>
```

Конечно, операторы `elseif` и `else` не являются обязательными. Временами достаточно простейшего оператора `if` (листинг 3.2).

Листинг 3.2. Простейший оператор `if`

```
<?php
     if(date("D") == "Mon")
     {
          print("Необходимо выкинуть мусор.");
     }
?>
```

Оператор `if` можно использовать для создания последовательности проверок, покрывающих все возможные варианты. Для этого необходимо начать с проверки первого условия с помощью оператора `if`; а затем посредством операторов `elseif` последовательно проверить все остальные условия. Поместив `else` в конец, вы перекрываете все возможные варианты. Этот метод используется в программном коде, приведенном в листинге 3.3, для распечатки наименования дня недели на немецком языке. Сценарий сравнивает наименование дня недели, начиная с понедельника (Monday) и кончая субботой (Saturday). При отсутствии совпадений считается, что наступило воскресенье (Sunday).

Листинг 3.3. Просмотр всех вариантов с помощью оператора `if-elseif-else`

```
<?php
    /*
    ** Получить название дня недели
    */
    $englishDay = date("l");
    /*
    ** Определить название сегодняшнего дня по-немецки
    */
    if($englishDay == "Monday")
    {
        $deutschDay = "Montag";
    }
    elseif($englishDay == "Tuesday")
    {
        $deutschDay = "Dienstag";
    }
    elseif($englishDay == "Wednesday")
    {
        $deutschDay = "Mittwoch";
    }
    elseif($englishDay == "Thursday")
    {
        $deutschDay = "Donnerstag";
    }
    elseif($englishDay == "Friday")
    {
        $deutschDay = "Freitag";
    }
    elseif($englishDay == "Saturday")
    {
        $deutschDay = "Samstag";
    }
    else
    {
    // Значит, сегодня воскресенье
        $deutschDay = "Sonntag";
    }
    /*
    ** Распечатать название сегодняшнего дня по-английски и по-немецки
    */
    print("<h2>Урок немецкого: Дни недели</h2>\n" .
        "<p>\n" .
        "по-английски: <b>$englishDay</b>.<br>\n" .
        "по-немецки: <b>$deutschDay</b>\n" .
        "</p>\n");
?>
```

3.2. Оператор ?

Язык PHP имеет сокращенную версию оператора if, который был позаимствован из C. В нем в качестве троичного оператора используется знак вопроса (рис. 3.2).

```
conditional expression ? true expression : false expression;
```

Рис. 3.2. Формат оператора ?

Условное выражение оценивается с точки зрения его значения: дает оно значение TRUE или FALSE. При значении TRUE выполняется выражение, заключенное между знаком вопроса и двоеточием, в противном случае выполняется выражение, указанное после двоеточия. Нижеприведенный фрагмент программного кода

```
($clientQueue > 0) ? serveClients() : cleanUp();
```

работает как оператор

```
if($clientQueue > 0)
    serveClients();
else
    cleanUp();
```

Но это подобие обманчиво. Несмотря на то что сокращенная форма кажется эквивалентом использования конструкции if-else, при более детальном рассмотрении это оказывается не так. Как было сказано, оператор ? является цельным оператором, а не составной конструкцией. А это значит, что выражение всегда обрабатывается целиком и значение соответствующего выражения подставляется вместо выражения ?. Другими словами, что-то похожее на

```
print(true ? "это истинно" : "это ложно");
```

является допустимым оператором. Так как условное выражение истинно, получаем эквивалентную строку:

```
print("это истинно");
```

что нельзя сделать с помощью оператора if.

Оператор ? можно отнести к плохочитаемым, и он не является крайне необходимым. И вполне возможно, что вы им никогда так и не воспользуетесь. Однако этот оператор позволяет создавать достаточно компактный программный код.

3.3. Оператор switch

Альтернативой структурам if-elseif-else является оператор switch, работающий с допущением, что производится сравнение одного выражения и множества возможных значений. На рис. 3.3 продемонстрирована структура оператора switch.

```
switch(root-expression)
{
    case case-expression:
    default:
}
```

Рис. 3.3. Оператор switch

В операторе switch сначала задается основное выражение, а затем с помощью операторов case последовательно сравнивается его значение с возможными вариантами значений. В конце списка вариантов необходимо разместить оператор default, который работает точно так же, как оператор else, т.е. он отрабатывает, если не найдены подходящие варианты.

Обратите внимание, что операторы case не ограничиваются фигурными скобками. Это подчеркивает одно существенное различие между операторами if и switch. После обработки блока if управление передается в конец всего оператора if. В листинге 3.3, если текущий день является вторником (Tuesday), переменной deutsch_Day присваивается значение Deinstag и управление передается вниз за фигурную скобку, закрывающую блок else.

Оператор case служит стартовой точкой выполнения алгоритма. Основное выражение сравнивается с каждым выражением case до тех пор, пока не будет обнаружено совпадение. Программный код, содержащийся в этом операторе, выполняется. Оператор case, не соответствующий основному выражению, игнорируется. Иногда в этом есть смысл, но чаще всего для прекращения выполнения оператора switch используется оператор break.

Посмотрим на листинг 3.4 — здесь листинг 3.3 переписан с применением оператора switch. Самым лучшим аргументом в пользу использования оператора switch является то, что он проще и понятнее. Так как PHP позволяет сравнивать строки, оператор switch эффективнее, чем в других языках. Если у вас есть определенный опыт программирования на языке Basic, то может возникнуть естественный вопрос: "А существует ли возможность для оператора switch в PHP работать с диапазонами?" Нет, такой возможности в PHP еще не предусмотрено. Вероятно, что такую ситуацию лучше обрабатывать с помощью оператора if-elseif-else.

Листинг 3.4. Обработка всех вариантов с помощью оператора switch

```
<?php
    /*
    ** Получить название дня недели
    */
    $englishDay = date("l");
    /*
    ** Определить название сегодняшнего дня по-немецки
    */
    switch($englishDay)
    {
        case "Monday":
            $deutschDay = "Montag";
            break;
        case "Tuesday":
            $deutschDay = "Dienstag";
            break;
        case "Wednesday":
            $deutschDay = "Mittwoch";
            break;
        case "Thursday":
            $deutschDay = "Donnerstag";
            break;
```

```
        case "Friday":
            $deutschDay = "Freitag";
            break;
        case "Saturday":
            $deutschDay = "Samstag";
            break;
        default:
            // Значит, сегодня воскресенье
            $deutschDay = "Sonntag";
    }
    /*
    ** Распечатать имя сегодняшнего для по-английски и по-немецки
    */
    print("<h2> Урок немецкого языка: Дни недели: Day of the
Week</h2>\n" .
    "<p>\n" .
        "по-английски: <b>$englishDay</b>.<br>\n" .
        "по-немецки: <b>$deutschDay</b>\n" .
        "</p>\n");
?>
```

3.4. Циклы

Циклы позволяют повторять строки программного кода после проверки выполнения определенного условия. Например, необходимо прочесть все строки из файла до его конца. Можно распечатать часть HTML-кода, повторив его десять раз, или запрограммировать три попытки подключения к базе данных, или прочесть данные из файла до его конца. Все эти действия программируются с помощью циклов.

Каждый цикл выполнения кода внутри цикла называется *итерацией*. Программный код внутри цикла повторяется до тех пор, пока не будет выполнено условие завершения. Язык PHP поддерживает три типа циклов, которые отличаются друг от друга тем, как они осуществляют итерации, какие действия выполняют перед началом цикла и когда осуществляется проверка условия завершения цикла, в начале или в конце цикла.

Оператор `while`

Простейшим циклом является цикл `while`. Выражение проверяется сразу же при первом удобном случае. Если это условие является ложным, программный блок просто пропускается, а если условие дает значение “истина”, программный блок выполняется, после чего управление передается обратно наверх и опять проверяется условие. На рис. 3.4 показана структура оператора `while`.

```
while(expression)
{
    Ничего или несколько операторов
}
```

Рис. 3.4. Оператор `while`

Цикл while имеет смысл в том случае, когда вы не знаете, сколько итераций должен сделать данный программный код, — так бывает, например, при чтении строк из файла или выборке из базы данных при помощи запроса. Для того чтобы узнать, как это работает, посмотрим на код, распечатывающий названия дней недели, начиная с текущего дня и заканчивая пятницей.

Цикл while (листинг 3.5) проверяет, чтобы дата, хранящаяся в переменной currentDate, не была пятницей (Friday). Если день является пятницей, выполнение цикла завершается и будет продолжено после закрывающих фигурных скобок. Но если сегодня не пятница, то распечатывается элемент списка с названием дня и к переменной currentDate прибавляется 24 часа. В этом месте достигается конец программного блока, и управление передается обратно в начало цикла.

Текущая дата снова проверяется на соответствие пятнице. Вот так постепенно будет достигнута пятница. Соответственно значение переменной currentDate станет пятницей, и цикл будет завершен. Но что будет, если запрограммировать что-то нелогичное, сравнивая, например, текущую дату со значением "Workday"? Дня недели с таким названием не существует, поэтому операция проверки всегда будет давать значение "истина". Это значит, что проверка date("l", $currentDate) != "Workday" всегда даст значение "истина". В результате получаем так называемый *вечный цикл*. Для достижения аналогичного результата лучше написать while(TRUE), что будет гораздо проще и прозрачнее.

Листинг 3.5. Использование цикла while для печати названий дней недели

```
<?php
    // Присвоить переменной $currentDate значение текущей даты в секундах
    $currentDate = time();
    // Напечатать какой-нибудь поясняющий текст
    print("До пятницы осталось (дней):\n");
    print("<ol>\n");
    while(date("l", $currentDate) != "Friday")
    {
        // Распечатать название дня недели
        print("<li>" . date("l", $currentDate) . "</li>\n");
        // Прибавить к переменной $currentDate 24 часа
        $currentDate += (60 * 60 * 24);
    }
    print("</ol>\n");
?>
```

Вечный цикл также называется *неопределенным*. Если вы вдруг обнаружите, что ваша страница загружается очень медленно, то вполне вероятно, что виной тому будет где-то запрограммированный вами бесконечный цикл. К счастью, машина PHP по умолчанию останавливает любой сценарий, если он занимает больше 30 секунд процессорного времени. Этот тайм-аут можно изменить с помощью функции set_time_limit. Иногда неопределенный цикл может создаваться преднамеренно, и его можно прервать где-то посередине выполнения. Для этих целей вполне может пригодиться оператор break.

Оператор `break`

Оператор `break` позволяет выйти из самого внутреннего цикла или оператора `switch`. Читатель может уже убедиться в том, что это позволяет повысить удобство использования операторов `switch`. И для циклов этот оператор тоже находит свое применение. Может так случиться, что потребуется выйти из цикла где-то посередине его выполнения, — в листинге 3.6 демонстрируется как раз такой случай.

Листинг 3.6. Выход из цикла с помощью оператора `break`

```
<?php
    while(TRUE)
    {
        print("Эта строка печатается.");
        break;
        print("Эта строка не будет распечатана никогда.");
    }
?>
```

Оператор `break` также позволяет выходить из вложенных циклов, для этого после самого оператора ставится определенная цифра. В листинге 3.7 показан выход из двух вложенных циклов.

Листинг 3.7. Выход из вложенных циклов

```
<?php
    while(TRUE)
    {
        while(TRUE)
        {
            print("Эта строка печатается.");
            break 2;
        }
        print("Эта строка не будет распечатана никогда.");
    }
?>
```

Оператор `continue`

Оператор `continue` по своему действию во многом аналогичен оператору `break`, за исключением того, что он не завершает полностью работу цикла, а только лишь текущую итерацию. Управление передается закрывающей фигурной скобке, и цикл продолжает выполняться дальше. Внутри циклов `for`, которые описываются ниже, приращение индекса цикла осуществляется так, как если бы цикл доработал нормально до конца.

Как можно себе представить, этот оператор используется для пропуска части цикла, когда выполняется определенное условие (листинг 3.8). Случайные числа генерируются внутри цикла, пока не будет сгенерировано десять чисел, каждое из которых

будет больше предыдущего. Бо́льшую часть само тело цикла не будет работать из-за работы оператора `if`, который инициирует работу оператора `continue`.

Подобно оператору `break`, можно задавать оператор `continue` с последующим целым числом. В этом случае управление передается вверх на указанное число уровней.

Листинг 3.8. Оператор `continue`

```
<?php
    /*
    ** Генерировать десять случайных чисел,
    ** каждое из которых больше предыдущего
    */
    // Инициализация переменных
    $count = 0;
    $max = 0;
    // Получить десять случайных чисел
    while($count < 10)
    {
        $value = rand(1,100);
        // Продолжить, если значение $value слишком мало
        if($value < $max)
        {
            continue;
        }
        $count++;
        $max = $value;
        print("$value <br>\n");
    }
?>
```

Оператор `do...while`

Принятие решения о продолжении выполнения цикла можно отложить на конец цикла, воспользовавшись оператором `do...while`. В листинге 3.9 повторяется алгоритм из листинга 3.5. Вы не заметите никакой разницы в работе программного кода, приведенного в этих листингах, до тех пор, пока дело не дойдет до пятницы. В этом случае программный код из листинга 3.5 ничего не выведет на печать. Новая же версия добавит в список дней пятницу, так как тело цикла выполняется до выполнения проверки значения переменной `currentDate`. Переход на цикл `do...while` позволит вывести на печать перечень дней до следующей пятницы.

Листинг 3.9. Использование цикла `do...while` для печати названий дней недели

```
<?php
    /*
    ** Присвоить переменной $currentDate значение текущей даты в секундах
    */
    $currentDate = time();
    // Напечатать какой-нибудь поясняющий текст
    print("До следующей пятницы осталось (дней):\n");
```

```
    print("<ol>\n");
    do
    {
        /*
        ** Распечатать название дня недели
        */
        print("<li>" . date("l", $currentDate) . "</li>\n");
        /*
        ** Прибавить к переменной $currentDate 24 часа
        */
        $currentDate += (60 * 60 * 24);
    }
    while(date("l", $currentDate) != "Friday");
    print("</ol>\n");
?>
```

Оператор `for`

Строго говоря, в цикле `for` нет особой необходимости. Любой цикл `for` можно создать с помощью цикла `while`. То, что может предложить цикл `for`, не несет в себе какой-то принципиально новой функциональности. Это просто традиционная структура построения циклов. Большинство циклов использует в работе переменную счетчика, который считает обрабатываемые итерации до тех пор, пока не будет достигнут его максимум.

Представим, что вам требуется пересчитать числа с 1 и до 10. При использовании оператора `while` некой переменной необходимо присвоить значение 1. После этого необходимо задать цикл `while`, который бы проверял значение счетчика на равенство или меньше 10. В конце программного блока этот счетчик должен увеличиваться на единицу. Проблема здесь заключается в том, что очень просто забыть это сделать. И в результате получим бесконечный цикл. Оператор `for` позволяет выполнить все эти действия в одной строке. В операторе `for` выполняются три операции: инициализация выражения, логическое выражение продолжения и выражение инкремента. На рис. 3.5 изображен цикл `for`.

```
for(initialization; continue; increment)
{
   Ничего или несколько операторов
}
```

Рис. 3.5. Оператор `for`

При первом прохождении выполняется выражение инициализации. Обычно это присвоение переменной значения 0 или 1. При всех последующих прохождениях, подобно тому, как это делается с оператором `while`, выполняется логическая проверка. И если эта проверка дает значение `FALSE`, управление сразу переходит за пределы программного блока, в противном случае блок выполняется. Перед следующей проверкой условия выполняется выражение инкремента. Таким образом, вся информация, необходимая для управления работой цикла, компактно располагается в одной строке, что позволяет хорошо проанализировать все итерации.

Листинг 3.10. Типичный цикл `for`

```
<?php
    for($counter = 1; $counter <= 10; $counter++)
    {
        print("Счетчик равен $counter<br>\n");
    }
?>
```

В листинге 3.10 представлен простейший цикл `for`. Большинство циклов `for` выглядит аналогично листингу 3.10. В них используются счетчики, инкрементируемые с шагом 1. Но в принципе по необходимости оператор `for` позволяет использовать и более сложные выражения. Инициализирующая часть оператора позволяет задавать список значений, разделенный запятыми. Это значит, что он может использоваться для задания одной или нескольких переменных. Эта часть может быть вообще пропущена. В листинге 3.11 приведен код из листинга 3.5, переделанный для работы с циклом `for`. Для удобства чтения первая строка оператора `for` разбита на подстроки, и это позволит лучше увидеть три части. Несмотря на то что оператор `for` удлинился и выглядит сложнее, на самом деле он не очень отличается от такого же оператора, представленного в листинге 3.9. В этом случае переменной `currentDate` присваивается некое начальное значение, которое проверяется на соответствие условию завершения цикла, после чего к значению добавляется приращение, равное не просто 1, а количеству секунд в дне.

Листинг 3.11. Использование цикла `for` для печати названий дней недели

```
<?php
    /*
    ** Напечатать какой-нибудь поясняющий текст
    */
    print("До пятницы осталось (дней):\n");
    print("<ol>\n");
    for($currentDate = date("U");
        date("l", $currentDate) != "Friday";
        $currentDate += (60 * 60 * 24))
    {
        /*
        ** Распечатать название дня недели
        */
        print("<li>" . date("l", $currentDate) . "</li>\n");
    }
    print("</ol>\n");
?>
```

Оператор `foreach`

Оператор `foreach`, имеющийся в арсенале PHP, реализует формализованный подход обеспечения итераций с применением массивов. О нем речь пойдет подробнее в главе 5, “Массивы”. Массив — это набор значений, на которые можно ссылаться

с помощью ключей. Оператор foreach делает выборку значений из массива. Как и любая другая циклическая структура, оператор foreach может иметь простой или сложный оператор, выполняющийся при каждой итерации цикла (рис. 3.6).

```
foreach(array as key=>value)
{
   Ничего или несколько операторов
}
```

Рис. 3.6. Оператор foreach

Оператор foreach ожидает массив, ключевое слово as и определение переменных, которые присваиваются каждому элементу. Если после ключевого слова as следует одно значение, как, например, foreach($array as $value), то на каждой итерации цикла переменной value будет присвоено значение последующего элемента массива. При создании оператора foreach можно также использовать индекс элемента массива: foreach($array as $key=>$value). К этому оператору мы еще вернемся в главе 5, "Массивы".

3.5. Операторы exit, die и return

Подобно оператору break, оператор exit обеспечивает возможность прервать выполнение программы, но, в отличие от оператора break, он завершает выполнение программы вообще. При этом в браузер не отсылается никакого диагностического текста. Такой прием полезен в случае возникновения ошибки, когда продолжение выполнения программы грозит гораздо более серьезными последствиями. Это довольно традиционный метод программирования при работе с запросами к базам данных. В этом случае, если оператор SQL не проходит простой синтаксической проверки, нет никакого смысла продолжать его выполнение.

Оператор die аналогичен оператору exit за исключением того, что он снабжается выражением, которое отсылается в браузер перед аварийным завершением работы сценария. Пользуясь тем, что подвыражения обрабатываются в соответствии с приоритетностью и ассоциативностью, и учитывая природу логических операторов, выражение из листинга 3.12 будет допустимо.

Листинг 3.12. Идиома использования оператора die

```
$fp = fopen("somefile.txt", "r") OR die("Невозможно открыть файл");
```

Приоритет оператора OR в листинге 3.12 имеет существенное значение, т.е. его приоритет меньше приоритета оператора присвоения, что позволяет PHP присвоить значение, возвращаемое функцией fopen, переменной fp, а затем обработать выражение OR. Оператор ||, функционально тождественный оператору OR, имеет более высокий приоритет, чем оператор присвоения. В случае применения оператора || в такой ситуации PHP обработает его первым и завершит работу сценария.

В главе 4, "Функции", обсуждается традиционное использование оператора return, но в PHP существует и вариант нетрадиционного использования данного оператора.

Это происходит при использовании оператора include, описанного в главе 7, "Операции ввода-вывода и доступ к диску". Если его вызов производится вне функции, оператор return останавливает выполнение текущего сценария и возвращает управление сценарию, который вызвал include, т.е. когда в сценарии используется функция include, вложенный сценарий может завершаться преждевременно. При использовании команды return в сценарии, который не был вызван с помощью оператора include, сценарий просто завершит свое выполнение, как это бывает при выполнении оператора exit.

Конечно, можно согласиться с тем, что это довольно странная концепция, но и такой оператор вполне заслуживает своего собственного имени. Но, с другой стороны, в особых случаях он позволяет создавать довольно приличный код.

3.6. Исключительные события

При возникновении ошибки PHP отсылает сообщение об ошибке в браузер. Некоторые ошибки прерывают выполнение сценария. Для обработки ошибок, которые не останавливают выполнение сценариев, их необходимо задавать с помощью функции, указанной с помощью функции set_error_handler. Эта функция подробно описывается в главе 15, "Настройка". Кроме того, можно воспользоваться возможностями функции trigger_error (о ней речь пойдет в главе 9, "Операционная система"), которая позволяет генерировать новые ошибки.

С другой стороны, для решения возникающих проблем можно использовать исключительные события (рис. 3.7). Исключительные события представляют собой объектно-ориентированные ошибочные условия и возникают в контексте оператора try. Для инициирования исключительного события используется оператор throw, затем управление передается блоку catch, который получает копию возникшего исключительного события. Для каждого исключительного события, которое будет перехватываться, необходимо добавить свой собственный блок catch или воспользоваться встроенным классом Exception PHP. Встроенный класс Exception включает два метода: getFile, возвращающий путь к файлу, генерировавшему исключительное событие, и getLine, возвращающий номер строки из этого файла.

При работе с объектно-ориентированными языками (например, Java) концепция исключительного события аналогична. Если же вы предпочитаете процедурный тип программирования, такой подход работать не будет. В листинге 3.13 продемонстрировано использование исключительных событий. Объекты детально обсуждаются в главе 6, "Классы и объекты". Если у вас нет никакого опыта работы с объектами, вернитесь к данному материалу после подробного изучения главы 6, "Классы и объекты".

```
try
{
   Ничего или несколько операторов
   throw Exception
   Ничего или несколько операторов
}
catch(class $variable)
{
   Ничего или несколько операторов
{
```

Рис. 3.7. Оператор try

Листинг 3.13. Использование оператора `try`

```
<?php
    // Вывести математическое выражение из основного класса
    class mathException extends Exception
    {
        public $type;
        public function __construct($type)
        {
            // Принять имя файла и номер строки
            parent::Exception();
            $this->type = $type;
        }
    }
    // Попробовать операцию деления
    $numerator = 1;
    $denominator = 0;
    try
    {
        // Запустить исключительное событие "деление на нуль"
        if($denominator == 0)
        {
            throw new mathException("Division by zero");
        }
        print($numerator/$denominator);
    }
    catch(mathException $e)
    {
        // Мы перехватили математическое исключительное событие
        print("Перехват математического исключительного события
($e->type) in " .
        "$e->file on line $e->line<br>\n");
    }
    catch(Exception $e)
    {
        // Мы перехватили исключительное событие другого типа
        print("Перехват исключительного события в " .
        $e->file() . " on line " .
        $e->line() . "<br>\n");
    }
?>
```

3.7. Оператор `Declare`

Оператор `declare` используется для отметки программного блока, предназначенного для выполнения при возникновении определенных условий. На рис. 3.8 представлена форма оператора `declare`.

```
declare(directive)
{
   Ничего или несколько операторов
}
```

Рис. 3.8. Оператор `declare`

В то время, когда писалась данная книга, PHP принимал и понимал только одну директиву — ticks. Директива ticks работает в паре с выражением register_tick_function, которое приводит к периодической остановке процессора PHP для выполнения функции. Каждый такой tick представляет собой событие низкого уровня, определенное анализатором. Эта функциональность не предназначалась для широкого использования при программировании, и процессор PHP не гарантирует никакого соответствия количества тиков и числа операторов, заключенных в блоке declare. В листинге 3.14 приводится пример такой функции, использующей директиву tick.

Листинг 3.14. Использование оператора `declare`

```
<?php
    // Определение tick-функции
    function logTick($part)
    {
        static $n = 0;
        print("Tick $n $part " . microtime() . "<br>\n");
        $n++;
    }
    print("Запуск " . microtime() . "<br>\n");
    // Зарегистрировать tick-функцию
    register_tick_function("logTick", "вычисляет квадратные корни ");
    // Запустить код, расположенный внутри блока declare
    declare(ticks=1)
    {
        1;1;1;
    }
    // Отменить регистрацию tick-функции
    unregister_tick_function("logTick");
    print("Завершено " . microtime() . "<br>\n");
?>
```

В будущем, вероятно, оператор declare сможет принимать и другие директивы. Так как директива ticks используется довольно редко, операторы declare можно проигнорировать.