Содержание

20

Интернационализация и локализация

В этой главе мы обсудим основы интернационализации веб-приложений в плане подготовки к локализации их содержимого. Создавать интернационализированные веб-приложения с помощью PHP несложно, а преимуществом интернациональной аудитории является ее многочисленность. Начните с осознания того, что интернационализация и локализация — разные, но связанные концепции, и вы будете готовы обрести по-настоящему всемирную аудиторию.

В главе рассматриваются следующие основные темы.

- Подготовка к разным наборам символов.
- Структурирование приложения для генерации локализованного содержимого.
- Использование `gettext()` для интернационализации и локализации.

Локализация — это больше, чем перевод

Распространенное заблуждение заключается в том, что локализацию веб-сайта, веб-приложения или вообще чего угодно сводят просто к переводу содержимого на язык, применяемый в целевом местоположении. Однако важно понимать, что ни интернационализация, ни локализация не являются тем же самым, чем является перевод содержимого. В действительности вы можете иметь веб-сайт или веб-приложение, наполненное содержимым, которое переведено, скажем, на немецкий, английский или японский язык, и все же такой веб-сайт или веб-приложение вообще может не считаться интернационализированным или даже локализованным. Это будет просто переведенный веб-сайт или веб-приложение, но не более того.

Чтобы создать локализованное программное обеспечение (охватывающее веб-сайты и веб-приложения, а также все другие типы), вы должны сначала его интернационализировать. Базовые аспекты интернационализированного ПО включают:

- вынесенные вовне строки, значки и графика;
- изменяемые способы отображения результатов, возвращаемых функциями форматирования (дат, денежных значений, чисел и т.д.).

Процесс локализации имеет смысл начинать только после того, как ПО сконструировано так, что строки вынесены вовне (т.е. все строки, используемые в функциях, классах и других местах кода, поддерживаются в одном месте и включаются либо по-другому задействуются в виде константных переменных), а функции форматирования можно изменять в зависимости от локали. Перевод содержимого является частью локализации, или нацеливанию ПО на конкретную локаль.

Локаль — это место, где что-то установлено, такое как место проживания людей, которые разговаривают на американском английском и пишут слово "color" без буквы "u"; на эту локаль можно ссылаться как на "США". Но в мире вычислений понятие локали относится к набору параметров, идентифицирующих язык пользователя, географический регион и любые основанные на местоположении предпочтения, которые могут быть важны для представления внутри пользовательского интерфейса.

Стандартные идентификаторы локали состоят из индикаторов языка и региона, например, `en_US` для "английского языка в США" и `en_GB` для "английского языка в Великобритании".

Вас может интересовать, зачем нужно устанавливать региональные различия, но подумайте всего лишь об отличиях в орфографии американского и британского английского, не говоря уже о контекстных отличиях, которые могут быть применены ко всему ПО для пользователей из США, но не для пользователей из Великобритании. В качестве другого примера представьте, что вы поддерживаете веб-сайт с текстом на немецком языке, который ориентирован на использование только посетителями из Германии; это будет локаль `de_DE` ("немецкий язык в Германии"). Хотя австрийцы, разговаривающие на немецком, прекрасно смогут работать с веб-сайтом в том виде, как он есть, если только вы не локализовали его для локали `de_AT` ("немецкий язык в Австрии"), этот веб-сайт никогда не будет для таких пользователей полностью правильным.

Наборы символов

Наборы символов обычно бывают *однобайтовыми* или *многобайтовыми*, что имеет отношение к количеству байтов, которое необходимо для представления символа в языке. Английский, немецкий и французский языки (среди многих других) относятся к однобайтовым наборам символов, потому что для представления символа, такого как буква *a* или цифра *9*, достаточно только одного байта. Однобайтовые кодовые наборы имеют максимум 256 символов, включая полный набор символов ASCII, символы с ударением и другие символы, необходимые для форматирования.

Многобайтовые наборы символов имеют более 256 символов, в том числе однобайтовые символы как подмножество. Многобайтовые языки среди прочих включают традиционный и упрощенный китайский, японский, корейский, тайский, арабский и иврит. Для представления символа такие языки требуют более одного байта. Хорошим примером может служить слово *Tokyo* (Токио). В английском языке оно записывается с помощью четырех разных символов, используя всего 5 байтов. Тем не менее, в японском языке это слово представляется двумя слогами, *tou* и *kyou*, каждое из которых занимает 2 байта, в итоге давая 4 байта.

Чтобы правильно интерпретировать и отображать текст веб-страниц на языке, для которого они предназначены, вы должны сообщить веб-браузеру, какой набор символов необходимо применять. Цель достигается отправкой перед всем содержимым подходящих заголовков.

Такими заголовками являются `Content-type` и `Content-language`; они также могут быть установлены как атрибуты дескрипторов HTML5. Поскольку PHP снабжает вас возможностью создания динамической среды, охватите все свое приложение за счет отправки соответствующих заголовков перед текстом и вывода корректных атрибутов в дескрипторах HTML5 внутри выходного документа.

В следующем примере функция `header()` выводит нужные заголовки для сайта на английском языке:

```
header("Content-Type: text/html;charset=ISO-8859-1");
header("Content-Language: en");
```

Дескрипторы HTML5, соответствующие показанным выше заголовкам, выглядят так:

```
<html lang="en">
<meta charset="ISO-8859-1">
```

Сайт на японском языке использует другой набор символов и другой код языка:

```
header("Content-Type: text/html;charset=UTF-8");
header("Content-Language: ja");
```

Вот дескрипторы HTML5, которые соответствуют этим заголовкам:

```
<html lang="ja">
<meta charset="UTF-8">
```

Важно правильно устанавливать заголовки. Например, если есть множество страниц, которые содержат текст на японском языке, и вы не отправили подходящие заголовки относительно языка и набора символов, тогда такие страницы будут визуализироваться некорректно в браузерах, где в качестве главного языка установлен не японский. Другими словами, из-за того, что не была включена информация о наборе символов, браузер предположит, что текст должен визуализироваться с применением стандартного набора символов.

Аналогично, если ваши страницы на японском языке используют набор символов UTF-8, а браузер настроен на ISO-8859-1, то браузер будет пытаться визуализировать японский текст с применением однобайтового набора символов ISO-8859-1. Браузер потерпит неудачу, если только заголовки не предупредят его об использовании UTF-8 *и* в системе не будут установлены необходимые библиотеки и языковые пакеты для отображения текста так, как было задумано.

Последствия для безопасности от наборов символов

На протяжении всего руководства по PHP (особенно в разделах, которые сосредоточены на взаимодействии с базами данных вроде MySQL) вы встретите предупреждения о последствиях для безопасности от наборов символов. Речь не идет о том, что наборы символов в своей основе небезопасны. Предупреждения предназначены для того, чтобы заставить разработчиков чуть больше вникать в сущность применяемых наборов символов. Разработчики должны быть осведомлены о том, что могут возникнуть проблемы с безопасностью, если не предпринять основные меры предосторожности при работе со строками, которые содержат такие символы — например, в операторах SQL.

Классическим примером проблем с безопасностью, связанных с наборами символов, является несоответствие кодировки между сервером, PHP и MySQL, особенно в случае многобайтовых языков. Давайте предположим, что PHP считает, что отправляет чистый текст ASCII в MySQL, а стандартный набор символов базы данных установлен в Big5. В такой ситуации при использовании функции `mysql_real_escape_string()` (или ее объектно-ориентированного эквивалента) для очистки строк от управляющих символов перед их отправкой в базу данных интерпретатор PHP потеряет замыкающий символ двухбайтового набора символов, поскольку ему попросту не известно о том, что его нужно искать.

Описанное недоразумение будет причиной того, что закодированный символ станет по существу ненужными данными, а это само по себе уже достаточно плохо, если вы намереваетесь отображать его пользователю. Однако гораздо худший исход произойдет, когда кто-то воспользуется таким несоответствием в своих интересах и внедрит код SQL в то, что теперь является открытой строкой, стремясь добраться до вашей базы данных с целью возможного выполнения в ней вредоносного кода.

Использование функций PHP для работы с многобайтовыми строками

Упомянув ранее в главе о многобайтовых схемах кодирования символов, было бы упущением не указать, что в состав PHP входит набор функций, специально предназначенных для манипулирования многобайтовыми строками. Если вы попытаетесь манипулировать многобайтовой строкой с помощью строковой функции, не осведомленной о многобайтовых схемах кодирования символов, то такая функция, скорее всего, не сможет корректно проанализировать строку. Вообще говоря, это имеет смысл, т.к. функция, которая ожидает однобайтовые символы, и не должна знать, что делать (или привносить беспорядок), когда она сталкивается с многобайтовыми строками.

Чтобы можно было применять функции PHP для работы с многобайтовыми строками, их потребуется включить во время процесса конфигурирования, используя параметр `--enable-mbstring` при настройке и сборке PHP. Пользователям Windows понадобится включить расширение `php_mbstring.dll` в файле `php.ini`. После успешного конфигурирования вы можете применять любую из свыше 40 функций, относящихся к `mbstring`, для обработки многобайтового ввода в PHP.

Исчерпывающие сведения о функциях, работающих с многобайтовыми строками, приведены в руководстве по PHP: `http://php.net/manual/ru/book.mbstring.php`. В качестве общего правила при работе с многобайтовыми строками ищите функцию с именем, похожим на имя однобайтового эквивалента (например, когда нужно найти первое вхождение строки внутри другой строки, ищите функцию `mb_stripos()` вместо просто `strpos()`).

Создание базовой страничной структуры, допускающей локализацию

Теперь, когда вы ознакомились с основными сведениями об интернационализации, локализации и наборах символов, рассмотрим процесс создания для вебсайта базовой страничной структуры, допускающей локализацию. Показанные здесь фрагменты позволяют пользователю выбрать целевой язык и затем получить приглашающее сообщение на выбранном языке.

Цель настоящего раздела — предоставить простой пример вынесения вовне строк (одна из характеристик интернационализации) и отображения локализованного текста на основе пользовательских предпочтений. Рабочий поток этого набора сценариев таков, что пользователь оказывается на вашем русскоязычном веб-сайте, но также имеет возможность просматривать его внутри выбранной локали (в данном примере только русской или японской).

В процесс вовлечены три действия.

- Создание и использование мастер-файла для отправки заголовка, специфичного к локали.
- Создание и применение мастер-файла для отображения информации на основе выбранной локали.
- Использование самого сценария.

В листинге 20.1 приведено содержимое мастер-файла, применяемого для отправки заголовка, специфичного к локали.

Листинг 20.1. `define_lang.php` — файл определения языка

```
<?php
if ((!isset($_SESSION['lang'])) || (!isset($_GET['lang']))) {
    $_SESSION['lang'] = "ru";
    $currLang = "ru";
} else {
    $currLang = $_GET['lang'];
    $_SESSION['lang'] = $currLang;
}
switch($currLang) {
    case "ru":
       define("CHARSET","ISO-8859-1");
       define("LANGCODE", "ru");
     break;
    case "ja":
       define("CHARSET","UTF-8");
       define("LANGCODE", "ja");
     break;
```

```
      default:
         define("CHARSET","ISO-8859-1");
         define("LANGCODE", "ru");
      break;
}
header("Content-Type: text/html;charset=".CHARSET);
header("Content-Language: ".LANGCODE);
?>
```

В листинге 20.1 можно заметить, что если переменная сеанса не существует, тогда используются настройки русской локали. Если бы сайт был по умолчанию японским, то данный файл понадобилось бы изменить для применения японской локали как стандартной. Этот сценарий рассчитан на использование вместе со сценарием из листинга 20.2, который содержит механизм выбора языка, устанавливая значение `$currLang` согласно результату выбора в строке 6 листинга 20.1.

Оператор `switch`, начинающийся в строке 10 листинга 20.1, имеет несколько операторов `case`, которые предназначены для присваивания подходящих значений константным переменным `CHARSET` и `LANGCODE`. В строках 27 и 28 листинга 20.1 эти переменные применяются в первый раз, когда динамически создаются и отправляются заголовки `Content-type` и `Content-language`.

Листинг 20.2. `lang_strings.php` — файл определения строк

```
<?php
function defineStrings() {
   switch($_SESSION['lang']) {
        case "ru":
           define("WELCOME_TXT","Добро пожаловать!");
           define("CHOOSE_TXT","Выберите язык");
        break;
        case "ja":
           define("WELCOME_TXT","ようこそ！");
           define("CHOOSE_TXT","言語を選択");
        break;
        default:
           define("WELCOME_TXT","Добро пожаловать!");
           define("CHOOSE_TXT","Выберите язык");
        break;
   }
}
?>
```

Внутри блоков `case` оператора `switch` в листинге 20.1 определяются две константы для каждого языка. Константами являются `CHARSET` и `LANGCODE`, которые соответствуют набору символов и коду языка для каждой локали. Эти константы используются в сценарии отображения из листинга 20.3 при создании подходящих дескрипторов `META` для набора символов и кода языка.

В листинге 20.2 создается функция, которая будет применяться в листинге 20.3 для передачи локализованных строк браузеру. Подобно листингу 20.1 в данном коде с использованием оператора `switch` определяются строки, применяемые для двух констант, `WELCOME_TXT` и `CHOOSE_TXT`, которые будут отображаться в сценарии, показанном в листинге 20.3.

Последний фрагмент головоломки связан с самим сценарием отображения, приведенным в листинге 20.3. Сценарий просто начинает сеанс, так что он может прочитать переменную сеанса, которая хранит результат выбора пользователем языка (и устанавливается через щелчок на ссылке, отображаемой на странице), и затем заполняет пустые места с помощью констант, определенных для выбранного языка.

Листинг 20.3. `lang_selector.php` — сценарий выбора и отображения языка

```
<?php
session_start();
include 'define_lang.php';
include 'lang_strings.php';
defineStrings();
?>
<!DOCTYPE html>
<html lang="<?php echo LANGCODE; ?>">
<title><?php echo WELCOME_TXT; ?></title>
<meta charset="<?php echo CHARSET; ?>" />
<body>
  <h1><?php echo WELCOME_TXT; ?></h1>
  <h2><?php echo CHOOSE_TXT; ?></h2>
  <ul>
    <li><a href="<?php echo $_SERVER['PHP_SELF']."?lang=ru"; ?>">ru</a>
    </li>
    <li><a href="<?php echo $_SERVER['PHP_SELF']."?lang=ja"; ?>">ja</a>
    </li>
  </ul>
</body>
</html>
```

Когда страница выбора языка (`lang_selector.php`) посещается впервые, она должна выглядеть примерно так, как показано на рис. 20.1, т.к. выбор языка не делался и потому по умолчанию отображается текст на русском языке.

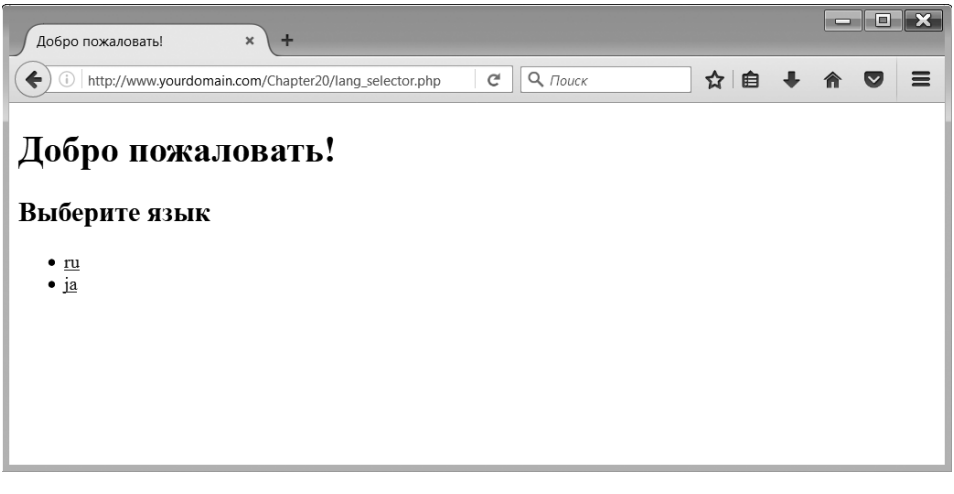

Рис. 20.1. *По умолчанию отображается текст на русском языке*

Тем не менее, после выбора другой локали (японской в данном примере) отображение изменится для вывода локализованных строк (рис. 20.2).

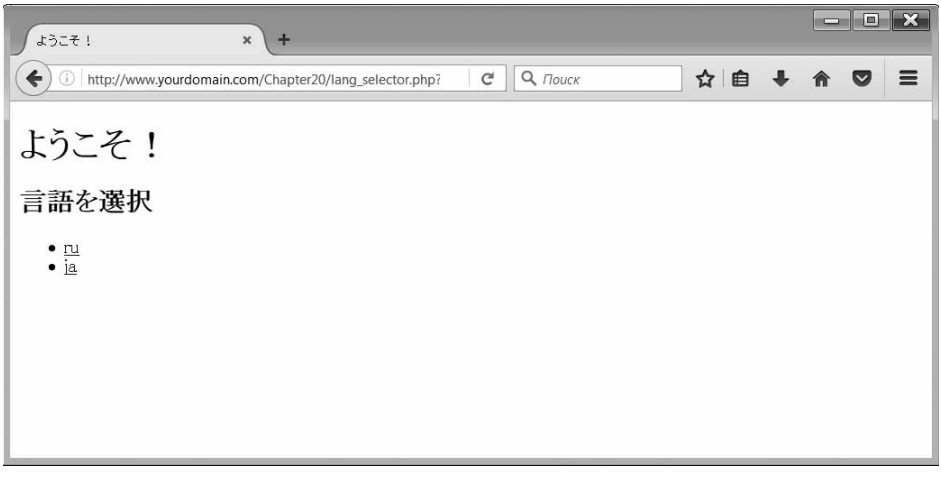

Рис. 20.2. *При выборе японской локали отображается текст на японском языке*

Использование `gettext()` в интернационализированном приложении

В предыдущем разделе был представлен базовый подход к интернационализации и локализации набора веб-страниц. Более передовой подход, применяемый для крупных сайтов с большим объемом содержимого или взаимодействия с пользователем, предусматривает использование встроенной функции PHP по имени `gettext()`, которая предлагает уровень API-интерфейса к GNU-пакету `gettext`.

Несмотря на то что применение функции `gettext()` вполне очевидно специфично для PHP, многие другие языки программирования обладают встроенной поддержкой GNU-пакета `gettext`. Концептуальное понимание того, что делает GNU-пакет `gettext`, важно для общего понимания интернационализации и локализации независимо от используемого языка программирования. В своей основе GNU-пакет `gettext` ищет специфически представленную строку, указанную в определенной локали, и заменяет такую строку в подходящем месте. Это очень похоже на процесс, который применялся для изменения строк, ассоциированных с константами, в листинге 20.2, но в более широких масштабах.

Конфигурирование системы для использования `gettext()`

Конфигурирование PHP для применения `gettext()` и связанных функций требует установки GNU-пакета `gettext`, изменения конфигурации PHP и привязки к некоторому каталогу внутри дерева документов веб-сервера.

Если вы используете сервер Linux или Mac OS X для разработки или развертывания, тогда с высокой вероятностью имеете установленный GNU-пакет `gettext`. В случае применения для разработки или развертывания сервера Windows, скорее всего, это не так.

Чтобы установить GNU-пакет gettext в любой системе, перейдите в раздел Downloading gettext (Загрузить gettext) страницы http://www.gnu.org/software/gettext/ и загрузите файл, подходящий для вашей системы. После установки GNU-пакета gettext вы можете сконфигурировать PHP для его распознавания и использования.

Включение gettext() и связанных функций в PHP после установки GNU-пакета gettext на сервере требует изменения конфигурации и повторной компиляции PHP в системе Linux или Mac OS X и разрешения готового расширения в системе Windows. В случае конфигурирования PHP для компиляции в системе Linux или Mac OS X добавьте следующий флаг компиляции:

```
--with-gettext
```

Добавив такой флаг компиляции, продолжайте компилировать и устанавливать PHP как обычно. Не забудьте перезапустить Apache после установки новой сборки модуля PHP.

В системе Windows отредактируйте файл php.ini, чтобы включить php_gettext.dll, удалив символ точки с запятой в начале строки вида:

```
;extension=php_gettext.dll
```

Внеся такое изменение и сохранив файл, перезапустите веб-сервер Apache. Благодаря описанным выше изменениям вы должны увидеть в выводе функции phpinfo() раздел, который указывает, что поддержка GNU-пакета gettext включена.

При включенной поддержке GNU-пакета gettext далее нужно создать в корне документов веб-сервера каталоги для содержимого, специфичного для локали. Первым делом вам понадобится создать в корне документов каталог, который будет содержать все каталоги локали (для всех поддерживаемых локалей).

Каталоги локали внутри родительского каталога должны иметь имена, которые содержат аббревиатуру из двух букв нижнего регистра для языка согласно спецификации ISO-639-1 (например, “en”, “ja”, “ru”), символ подчеркивания и код страны из двух букв верхнего регистра в соответствие со спецификацией ISO-3166-1 (скажем, “US”, “JP”, “RU”). Внутри каталога, специфичного для локали, должен присутствовать каталог по имени LC_MESSAGES.

Таким образом, для поддержки на веб-сайте трех локалей с применением GNU-пакета gettext и PHP-функции gettext() структура каталогов может выглядеть примерно так:

```
/htdocs
  /locale
    /ru_RU
      /LC_MESSAGES
    /en_US
      /LC_MESSAGES
    /ja_JP
      /LC_MESSAGES
```

Далее вы узнаете, какие виды файлов помещаются в указанные каталоги — это не файлы PHP, а файлы, которые содержат переведенные строки, предназначенные для использования по всему локализованному веб-сайту.

Создание файлов перевода

Файлы перевода, хранящиеся в файловой системе, как указано выше, и применяемые GNU-пакетом `gettext`, являются специфическим файловым типом, который называется файлом переносимого объекта (Portable Object — PO). Такие файлы содержат простой текст, но имеют расширение `*.po`. Вообще говоря, для создания файлов PO какой-то специальный редактор не требуется — в конце концов, в них находится всего лишь простой текст. Однако многие обнаруживают, что использование определенного редактора либо инструмента управления содержимым значительно сокращает накладные расходы, связанные с сопровождением файлов подобного рода (которые включают сотрудничество с переводчиками и другими создателями содержимого).

Мы рекомендуем опробовать для создания и сопровождения файлов локализованного содержимого инструменты вроде Poedit (`https://poedit.net/`) или POEditor (`https://poeditor.com/`), но пока просто продемонстрируем примеры файлов PO, введенные как простой текст. Для каждой локали необходим один файл PO по имени `messages.po`.

Файлы PO начинаются с идентифицирующей заголовочной информации и продолжаются списком идентификаторов и строк сообщений. В листинге 20.4 приведен пример содержимого файла PO, устанавливающего две строки для применения в локали `ru_RU`.

Листинг 20.4. `messages.po` — файл PO для локали `ru_RU`

```
# требуются пустые msgid и msgstr
msgid ""
msgstr ""

"Project-Id-Version: 0.1\n"
"POT-Creation-Date: 2017-04-05 14:00+0500\n"
"Last-Translator: Иван Петров <ivan@petrov.com>\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Language: ru_RU\n"

# приветственное сообщение
msgid "WELCOME_TEXT"
msgstr "Добро пожаловать!"

# приглашение выбрать язык
msgid "CHOOSE_LANGUAGE"
msgstr "Выберите язык"
```

В начале файла указан пустой идентификатор сообщения (`msgid`) и пустая строка сообщения (`msgstr`). За ними следует идентифицирующая информация о файле и его создателях: файл имеет версию 0.1, создан 5 апреля 2017 года Иваном Петровым и закодирован в наборе символов UTF-8 для предполагаемого использования с локалью `ru_RU`.

После всей заголовочной информации приводятся сообщения и их переводы. В этом примере имеются два сообщения, идентифицируемые собственными ключами: `WELCOME_TEXT` и `CHOOSE_LANGUAGE`. Ключи сообщений похожи на константы, которые применялись в простой версии локализации ранее в главе, но именованы по-другому, чтобы не путаться в двух примерах.

За каждым ключом сообщения следует строка сообщения для использования вместо этого ключа. Комментарии поясняют назначение каждого набора ключа и строки, а между парами строк предусмотрена одна пустая строка.

Выглядит просто, что и есть на самом деле, но с созданием и сопровождением файлов PO также связана скрытая сложность: длинный список доступных для применения параметров, которые перечислены в спецификации по адресу `http://www.gnu.org/software/gettext/manual/gettext.html#PO-Files`.

Спецификацию рекомендуется прочитать. Редактор файлов PO используется еще и потому, что он берет на себя заботу об еще одном действии — преобразовании файлов PO в файлы MO. Файл MO представляет собой файл машинного объекта (Machine Object), или файл, который содержит двоичный объект, в итоге читаемый GNU-пакетом `gettext`. Легкие для восприятия и сопровождения файлы PO не применяются GNU-пакетом `gettext` напрямую, а взамен используются файлы MO.

Благодаря GNU-пакету `gettext` и связанным с ним инструментам, установленным в системе, файлы PO можно преобразовывать в файлы MO с помощью служебной программы. В среде Linux соответствующая команда будет такой:

```
msgfmt messages.po -o messages.mo
```

Команда создает файл MO по имени `messages.mo` из содержащего простой текст файла PO по имени `messages.po`. Теперь все готово к применению PHP для выполнения остальной работы.

Реализация локализованного содержимого в PHP с использованием `gettext()`

После всех объяснений способа установки и применения GNU-пакета `gettext` реализация в PHP может выглядеть слегка разочаровывающей. Ниже перечислены основные шаги реализации.

- Использовать `putenv()` для установки переменной среды `LC_ALL` для локали.
- Применить `setlocale()` для установки значения `LC_ALL`.
- Использовать `bindtextdomain()` для установки местоположения каталога переводов для заданной области (область в этом случае означает имя, идентифицирующее файл, который хранит строки сообщений, а не домен, подобный `www.mydomain.com`).
- Применить `textdomain()` для установки стандартной области, подлежащей использованию с `gettext()`.
- Применить `gettext("какой-то msgid")` или `_("какой-то msgid")` для обращения к переводу GNU-пакетом `gettext`, который относится к указанному идентификатору сообщения.

В результате сбора всех фрагментов вместе получится сценарий, похожий на приведенный в листинге 20.5.

Листинг 20.5. `use_gettext.php` — чтение файлов MO с помощью PHP

```
<?php
$locale="ru_RU";
putenv("LC_ALL=".$locale);
setlocale(LC_ALL, $locale);
```

```
$domain='messages';
bindtextdomain($domain, "./locale");
textdomain($domain);
?>
<!DOCTYPE html>
<html>
<title><?php echo gettext("WELCOME_TEXT"); ?></title>
<body>
  <h1><?php echo gettext("WELCOME_TEXT"); ?></h1>
  <h2><?php echo gettext("CHOOSE_LANGUAGE"); ?></h2>
  <ul>
    <li><a href="<?php echo $_SERVER['PHP_SELF']."?lang=ru_RU";?>">ru_RU
    </a></li>
    <li><a href="<?php echo $_SERVER['PHP_SELF']."?lang =ja_JP";?>">ja_JP
    </a></li>
  </ul>
</body>
</html>
```

Если после освоения основ применения интернационализации и локализации вы собираетесь заняться разработкой приложения, используемого носителями множества разных языков, тогда мы рекомендуем обратить внимание на инфраструктуру локализации, базирующуюся на GNU-пакете `gettext`, и краудсорсинговые службы перевода для поддержки создания файлов PO (на тот случай, если вы не располагаете группой носителей языка или же не готовы тратить крупные суммы на услуги переводчиков).

Дополнительные источники информации

Интернационализация и локализация — это обширные темы, и в настоящей главе были раскрыты только самые основные концепции. Например, мы не обсуждали локализацию чисел, дат и денежных значений с применением PHP. Тем не менее, все упомянутые задачи можно решить с использованием встроенной функциональности, такой как функция `strftime()` для отображения времени с учетом локали, или за счет расширения функциональности для создания вспомогательных классов, которые удовлетворят имеющиеся потребности. Можете также взглянуть на инфраструктуры PHP, которые помогают все это обработать автоматически, либо с целью оценки их пригодности к вашему проекту, либо для ознакомления с тем, как разработчики построили такую функциональность. Обилие примеров можно найти на веб-сайтах Zend Framework (`http://framework.zend.com/manual/current/en/modules/zend.i18n.translating.html`) и Symfony (`http://symfony.com/doc/current/book/translation.html`).

Что дальше

Одним из многочисленных полезных действий, которые можно выполнять с помощью PHP, является создание изображений на лету. В следующей главе будет показано, как применять функции библиотеки обработки изображений для достижения ряда интересных и полезных эффектов.