第 21 章 开发最佳实践

http://www.lamppr.com/node/896

在本章中，我们给出了许多的代码小提示和最佳实践，它能帮你成长为一个合格的 Drupal 开发者，并帮你摆脱电脑的折磨。我们首先学习 Drupal 的编码标准,接着学习如何为模块创建文档以方便其它开发者理解代码。还介绍了如何在 Drupal 的核心中快速的查找东西，介绍了版本控制，并详细的说明了如何维护一个第 3 方模块，最后我们讨论了如何调试和剖析代码。

编码标准

Drupal 社区认为，它的基本代码必须拥有一个标准的外观，从而提高可读性，也使得初学者更容易学习。社区也鼓励第 3 方模块的开发者采用这些标准。实际上，让我老实 的告诉你：如果你没有遵守编码标准，那么你的模块在 Drupal 社区就不会得到认真对待。我们首先学习一下具体的标准，接着介绍了一些用来检查代码的自动 工具（甚至为你纠正代码！）

行缩进
Drupal 代码缩进使用两个空格，而不是 tab 键。对于大多说编辑器，你可以设置一个首选，从而自动的将 tab 键代替为两个空格，这样你就可以继续使用 Tab 键了——如果你习惯使用了 tab 键。

PHP 开始和结束标签
包含代码的文件，比如.module 或.inc 文件，会使用 PHP 代码开始标签，如下所示：

```
1 <?php
2 ...
```

在 Drupal 中不能使用开始标签的简写形式“<?”。

结束标签“?>”不是必须的，在 Drupal 代码中也不使用它。事实上，如果使用了这个标签的话，可能会带来麻烦。对于结束标签，也有一个例 外，那就是在模板文件中，为了退出 PHP 并且回到 HTML 中，此时会使用结束标签，例如，在 themes/bluemarine /block.tpl.php 中：

```
1 <?php
2 // $Id: block.tpl.php,v 1.3 2007/08/07 08:39:36 goba Exp $
3 ?>
4 <div class="block block-<?php print $block->module; ?>"
id="block-<?php
5 print $block->module; ?>-<?php print $block->delta; ?>">
6 <h2 class="title"><?php print $block->subject; ?></h2>
7 <div class="content"><?php print $block->content; ?></div>
8 </div>
```

控制结构体是程序中用来控制执行流程的指令，比如条件语句和循环语句。条件语句有 if，else，elseif，和 switch 语句。循环语句有 while，do-while，for，和 foreach。

控制结构体在控制关键字（if， elseif， while， for， 等等）和开括号“（”之间应有一个空格，从而将其与函数调用（也使用圆括号，但是没有空格）区分开来。“{”应该与关键字位于同一行（而不是自成一行）。“}”应该自成一行。

错误的

```
1 if ($a && $b)
2 {
3 sink();
4 }
```

正确的

```
1 if ($a && $b) {
2 sink();
3 }
4 elseif ($a || $b) {
5 swim();
6 }
7 else {
8 fly();
9 }
```

花括号“{}”一般总是使用的，即便是不需要的时候，为了增强可读性并降低出错的可能，也应使用“{}”。

错误的

```
1 while ($a < 10)
2 $a++;
```

正确的

```
1 while ($a < 10) {
2 $a++;
3 }
```

切换语句的格式应该这样（注意“break;”语句在默认情况下不是必需的）：

```
01 switch ($a) {
02 case 1:
03 red();
04 break;
05
06 case 2:
07 blue();
08 break;
09
10 case 3:
11 purple();
12 // Fall through to default case.
13
14 default:
15 green();
16 }
```

当一个情况执行完以后，打算继续执行下一情况时，此时可以省略“break;”语句，注意前面代码中的注释。

函数调用

函数调用

在函数调用中，在操作符（=， <， >，等等）的两边应该各有一个空格，而在函数名和函数的开括号“（”之间则没有空格。在函数的开括号“（”和它的第一个参数之间也没有空格。中间的 函数参数使用逗号和空格分隔，在最后一个参数和闭括号“）”之间没有空格。下面的例子说明了这几点：

错误的

```
1 $var=foo ($bar,$baz);
```

正确的

```
1 $var = foo($bar, $baz);
```

这个规则也存在例外的情况。在一个包含多个相关赋值语句的区块中，如果能够提高可读性，那么可以在赋值操作符周围插入更多空格：

```
1 $a_value = foo($b);
2 $another_value = bar();
3 $third_value = baz();
```

函数声明

在函数的名字和它的开括号“（”之间没有空格。在编写函数时，如果它的有些参数需要使用默认值，那么需要把这些参数列在后面。还有，如果你的函数生成了任何有用的数据，那么你需要返回该数据，以供调用者使用。下面给出了一些函数声明的例子：

错误的

```
1 function foo ($bar = 'baz', $qux){
2 $value = $qux + some_function($bar);
3 }
```

正确的

```
1 function foo($qux, $bar = 'baz') {
2 $value = $qux + some_function($bar);
3 return $value;
4 }
```

函数名字

在 Drupal 中，函数的名字都是小写的，并基于模块的名字或者它们所属系统的名字。这个习惯避免了命名空间冲突。下划线用来分隔函数名字的描述性部分。 在模块名的后面，应该紧跟一个动词，接着是动词作用的对象：modulename_verb_object()。在下面的第一个例子中，函数名字没有正确 的使用模块前缀，并且动词和它的对象颠倒了。在接下来的例子中，很明显，修正了这些错误。

错误的

```
1 function some_text_munge() {
2 ...
3 }
```

正确的

```
1 function mymodule_munge_some_text() {
2 ...
3 }
```

私有函数与其它函数一样，遵守相同的习惯，不过它在函数名字前面加了一个下划线。

数组

对于数组，也是使用空格对它的每个元素和每个赋值操作符进行分割的。如果数组区块跨越了 80 个字符，那么每个元素都应独立成行。为了提高可读性和可维护性，最好将每个元素全部独立成行。这样你就可以方便的添加或者删除数组元素了。

错误的

```
1 $fruit['basket'] = array('apple'=>TRUE, 'orange'=>FALSE,
'banana'=>TRUE,
2 'peach'=>FALSE);
```

正确的

```
1 $fruit['basket'] = array(
2 'apple' => TRUE,
3 'orange' => FALSE,
4 'banana' => TRUE,
5 'peach' => FALSE,
6 );
```

注意 数组中最后一个元素的后面有一个逗号，这不是一个错误，PHP 允许这样。放在这里是为了防止犯错，这样开发者就可以方便的在数组列表的最后添加或者删除一个元素。这一规范是允许用的，推荐大家使用这一规范，不过它不是必须的。

在创建内部的 Drupal 数组时，比如菜单项或者表单定义，总是将每个元素单独成行：

```
1 $form['flavors'] = array(
2 '#type' => 'select',
3 '#title' => t('Flavors'),
4 '#description' => t('Choose a flavor.'),
5 '#options' => $flavors,
6 );
```

常量

常量

PHP 常量应该全部大写，并使用下划线来正确的分隔字词：

```
1 /**
2 * First bootstrap phase: initialize configuration.
3 */
4 define('DRUPAL_BOOTSTRAP_CONFIGURATION', 0);
```

常量的名字也应该使用它们的模块名作为前缀，这样就可以避免常量之间的命名冲突。例如，假定你在

编写 tiger.module，那么需要使用 TIGER_STRIPED，而不是使用 STRIPED。

全局变量
在你自己的模块中，最好不要使用全局变量。如果你必须使用全局变量，那么命名方式为：下划线，接着是你的命名空间（也就是你的模块或主题的名字），再接一个下划线，之后紧跟一个描述性的名字。

错误的

```
global $records;
```

正确的

```
global $_mymodulename_access_records;
```

模块名字
永远不要在模块名字中使用下划线。为了理解这一点，考虑以下情景：

1.一个开发者创建了 node_list.module，它包含了一个名为 node_list_all()的函数。
2.在 Drupal 的下一版本中，核心节点模块添加了一个函数 node_list_all()————命名空间冲突了！

如果开发者遵守模块的命名习惯，不使用下划线，那么前面的冲突就可以避免：nodelist_all()将永远不会与核心代码冲突。

对这一点的最简单的理解方式就是，把第一个下划线左边的任何东西看作是模块的名字，从而把“模块名＋下划线”看作是一个命名空间。例如，在 node_命名空间下的所有东西都属于核心中的节点模块。如果你编写的函数以 node_，user_，filter_,或者其它核心命名空间起头，那么你这是自寻烦恼。如果你贡献的第３方模块中存在命名空间冲突，那么就意味着你需要为其花费更多的时间，而 基于你的模块编写代码的人也需要花费更多的功夫。

译者注：这一点确实很有道理，但是在许多的第３方模块中，许多模块的名字使用了下划线，比如 advanced_forum，backup_migrate，og_forum 等等。不过在用的时候一定要小心。

文件名

文件名应该是小写的。例外情况就是文档文件，它们全部大写并使用.txt 后缀，例如：

CHANGELOG.txt
INSTALL.txt
README.txt

在给文件命名时，最好遵守核心中使用的习惯。核心中的手册模块的文件，如表 21-1 所示。

表 21-1.在手册模块和模块相关文件中使用的文件名字

文件名	描述
book.info	模块名字,描述,核心兼容性,依赖性
book.install	模式定义;包含在模块安装、卸载、启用、禁用时运行的钩子。
book.module	代码
book.admin.inc	包含了访问管理页面时所用的代码
book.pages.inc	特定于用户（很少使用）的函数的代码
book.css	手册相关的类和 ID 的默认 CSS
book-rtl.css	用于从右到左的语言的 CSS 覆写
book-all-books-block.tpl.php	默认模板文件

```
book-export-html.tpl.php          默认模板文件
book-navigation.tpl.php           默认模板文件
book-node-export-html.tpl.php     默认模板文件
```

PHP 注释

Drupal 遵循大多数的 Doxygen 注释样式指南。所有的文档区块必须使用下面的语法：

```
1 /**
2 * Documentation here.
3 */
```

除了第一行以外，其它各行在星号（*）前面必须要有一个空格。

注意 Doxygen 是一个能够友好支持 PHP 的文档生成器。它从代码中提取 PHP 注释并生成适合用户阅读的文档。更多信息，可参看 http://www.doxygen.org。

当为一个函数添加说明文档时，文档区块必须紧挨着放在函数前，中间不能存在空行。

Drupal 能够理解下面所列的 Doxygen 构造体；尽管我们接下来会介绍到其中的大多数，不过关于如何使用它们的更多信息，请参看 Doxygen 的官方站点。

・ @mainpage
• @file
• @defgroup
• @ingroup
• @addtogroup (as a synonym of @ingroup)
• @param
• @return
• @link
• @see
• @{
• @}

遵循这些标准的好处是，你可以使用第 3 方的 API 模块，为你的模块自动生成文档。API 模块实现了 Doxygen 文档生成器规范的一个子集，专门针 对 Drupal 代码的文档生成进行了优化。访问 http://api.drupal.org，你就可以看到这个模块的一个实例，而关于 API 模块的更多信 息，可参看 http://drupal.org/project/api。

文档例子

让我们从头到脚仔细的看看一个模块的基干，同时将不同类型的文档从中选出来进行单独说明。

模块的第 2 行（在<?php 开始标签之后），应该包含一个 CVS 标签，用来追踪文件的版本号：

```
// $Id$
```

当把代码提交到 CVS 以及使用 CVS 更新代码（CVS 的代码是最新的）时，系统将会自动对这一标签进行解析和扩展。之后，它将自动地变成下面的样子：

```
// $Id: comment.module,v 1.617.2.2 2008/04/25 20:58:46 goba Exp $
```

在本章后面，你将学到更多关于如何使用 CVS 的知识。

在声明函数以前，你需要花点功夫为模块写点文档，介绍模块能做什么，文档的格式如下所示：

```
1 /**
2 * @file
3 * One-line description/summary of what your module does goes
here.
4 *
5 * A paragraph or two in broad strokes about your module and how
it behaves.
6 */
```

文档常量

PHP 常量全部都应该大写，可以使用下划线分隔单词。当定义 PHP 常量时，最好能够解释一下它们是用来做什么的，如下面的代码片段所展示的这样：

```
1 /**
2 * Role ID for authenticated users; should match what's in the
"role" table.
3 */
4 define('DRUPAL_AUTHENTICATED_RID', 2);
```

为函数编写文档

函数文档应该使用下面的语法：

```
01 /**
02 * Short description, beginning with a verb.
03 *
04 * Longer description goes here.
05 *
06 * @param $foo
07 * A description of what $foo is.
08 * @param $bar
09 * A description of what $bar is.
10 * @return
11 * A description of what this function will return.
12 */
13 function name_of_function($foo, $bar) {
14 ...
15 return $baz;
16 }
```

简短描述的开头，应该使用现在时态的祈使动词，比如“Munge form data”（“混合标单数据”）或“Do remote address lookups”（执行一个远程地址查找）（而不是“Munges form data” 或“Does remote address lookups”）。让我们看一个来自于 Drupal 核心的一个例子，它位于 system.module 中：

```
01 /**
02 * Add default buttons to a form and set its prefix.
03 *
04 * @ingroup forms
```

```
05 * @see system_settings_form_submit()
06 * @param $form
07 * An associative array containing the structure of the form.
08 * @return
09 * The form structure.
10 */
11 function system_settings_form($form) {
12 ...
13 }
```

在前面的例子中有一些新的 Doxygen 结构体:

· @see 告诉你可参考哪些其它函数。前面的代码是一个表单定义,所以@see 指向了表单的提交处理器。当 API 模块解析这个注释来生成文档时（比如 http://api.drupal.org 中可用的文档），它将把@see 后面的函数名转换为一个可点击的链接。

· @ingroup 将一组相关的函数联系到了一起。在这个例子中，它创建了一组提供表单定义的函数。你可以创建你想要的任意的小组名字。可能的核心值 有：batch, database, file, format, forms, hooks, image, menu, node_access, node_content, schemaapi, search, themeable, 和 validation。

提示 你可以在 http://api.drupal.org 查看一个给定组的所有函数。例如，表单构建器函数位于 http://api.drupal.org /api/group/forms/6，而可主题化的函数位于 http://api.drupal.org/api/group/themeable /6。

实现普通的 Drupal 结构体的函数，比如钩子函数或表单验证/提交函数，可以完全省略@param 和 @return 语法，但是仍然应该包含一行描述函数功能的说明，如下面的例子所示：

```
1 /**
2 * Validate the book settings form.
3 *
4 * @see book_admin_settings()
5 */
6 function book_admin_settings_validate($form, &$form_state) {
7 ...
8 }
9 }
```

如果一个函数是一个菜单回调（也就是，使用 hook_menu()映射到一个 URL 上），那么最好能加以说明：

```
1 /**
2 * Menu callback; print a listing of all books.
3 */
4 function book_render() {
5 ...
6 }
```

为钩子实现编写文档

当一个函数是一个钩子实现时，此时不需要为钩子编写文档。简单的说明一下实现了哪个钩子就可以了，例如：

```
1 /**
2 * Implementation of hook_block().
3 */
4 function statistics_block($op = 'list', $delta = 0, $edit =
array() {
5 ...
6 }
```

使用 egrep 来查找代码

egrep 是一个 Unix 命令，用来在文件中搜索匹配给定正则表达式的位置。不，它不是一个鸟儿（那是 egret（白鹭））。如果你是一个 Windows 用户，并想学习一下这些例子，你可以先安装一个预编译的版本（参看 http://unxutils.sourceforge.net）或者 安装 Cygwin 环境（http://cygwin.com），这样就可以使用 egrep 了。否则，你只能使用操作系统内置的搜索功能，而不能使用 egrep。

当你需要在 Drupal 核心中查找钩子实现时，当查找错误消息生成的位置时，以及其它一些情况时，egrep 都是一个很方便的工具。让我们看一些例子，这里在 Drupal 根目录下使用 egrep：

```
$ egrep -rl 'hook_init' .
./includes/bootstrap.inc
./includes/path.inc
./modules/aggregator/aggregator.module
./modules/book/book.module
./modules/forum/forum.module
./modules/node/node.module
./modules/poll/poll.module
./modules/system/system.module
./update.php
```

在前面的情况中，我们在 Drupal 文件中从当前目录（.）递归的搜索(-r)包含 hook_init 的实例，并将匹配实例的文件名打印出来（-l）。现在看下面的例子：

```
$ egrep -rn 'hook_init' .
./includes/bootstrap.inc:1011: // Initialize $_GET['q'] prior to
loading
modules and invoking hook_init().
./includes/path.inc:9: * to use them in hook_init() or hook exit()
can make
them available, by
./modules/aggregator/aggregator.module:261: * Implementation of
hook_init().
./modules/book/book.module:164: * Implementation of hook_init().
Adds the
book module's CSS.
./modules/forum/forum.module:160: * Implementation of hook_init().
./modules/node/node.module:1596: * Implementation of hook_init().
./modules/poll/poll.module:24: * Implementation of hook_init().
./modules/system/system.module:538: * Implementation of
hook_init().
./update.php:18: * operations, such as hook_init() and hook_exit()
```

```
invokes,
css/js preprocessing
```

这里，我们在我们 Drupal 文件中，递归的(-r)搜索带有字符串“hook_init”的实例，并将它们出现的位置以及行号(-n)打印出来。 我们可以进一步的提炼搜索结果。在下面的这个例子中，我们在前面例子的搜索结果集的基础上，进一步的搜索单词“poll”出现的情况：

```
$ egrep -rn 'hook_init' . | egrep 'poll'
./modules/poll/poll.module:24: * Implementation of hook_init().
```

提炼你的搜索的另一种方式，就是为 egrep 使用-v 标记，它意味着“反转这个匹配”，也就是说，不匹配该字符串的就是所要匹配的。让我们查找单词“lock”出现的所有地方，同时排除单词 block 或 Block 的情景：

```
$ egrep -rn 'lock' . | egrep -v '[B|b]lock'
./includes/common.inc:2548: // See if the semaphore is still
locked.
./includes/database.mysql.inc:327:function db_lock_table($table) {
./includes/database.mysql.inc:332: * Unlock all locked tables.
```

利用版本控制

对于任何软件项目,版本控制都是必须的，同样 Drupal 也不例外。版本控制用来追踪 Drupal 中的每个文件上的所有变更。它保存了所有修订本的 历史以及每个修订本的作者。你可以从中得到一个逐行的报告，里面包含谁做了变更以及什么时间什么原因。版本控制也可以简化新版本的发布流程。 Drupal 社区使用久经锤炼的可靠的 CVS 软件,来维护它的修订本历史。

提示 有多种版本控制系统(Bazaar, CVS, Git, Subversion,等等),对于它们之间的优缺点的讨论,在 Drupal 开发邮件列表中会经常出现。如果你也想发篇关于这方面的文章，那么你可以先看看归档文章，熟悉一下以前的讨论。

修订本控制的好处不局限于管理 Drupal 工程。你也可以利用 Drupal 的 CVS 来帮助维护基于 Drupal 的工程，这样能够极大的降低你的维护成本。首先，你需要改变一下你的 Drupal 安装方式。

安装带有 CVS 的 Drupal

当你从 drupal.org 的下载页面，下载了 Drupal 压缩包时，代码的这份拷贝没有带有任何版本信息；版本信息可用来告诉你基准代码的当前状态。

对于版本问题，使用 CVS 的开发者可以迅速的找到答案并应用更新，而其他的开发者则仍然需要下载新版本。

注意 有两种下载 Drupal 的方式，它们唯一的可见区别是， CVS 签出的包含了一个名为“CVS”的附加文件夹，Drupal 中的所有目录里面都包含了一个这样的文件夹，它们用来存放 CVS 信息。Drupal 的.htaccess 文件包含了一组规则，如果你使用的是 Apache 的话，这些规则可用来自动屏蔽掉这些文件夹（一些 CVS 客户端，比如 TortoiseCVS，能够默认隐藏 CVS 文件夹）。

可能会有人告诉你 CVS 版的 Drupal 不安全，而且使用 CVS 获取的最新代码也不稳定。这是一个常见的误解，它混淆了两个概念。这些人在这里指的 是一个工程的 HEAD 版本；也就是，在这个 Drupal 版本中（或者使用 CVS 的其它工程），为了准备下一个发布，正在测试新特性。然而，CVS 可同时用 来

维护一个软件的 HEAD 版本和所有的稳定版本。

使用带有 CVS 的 Drupal

那么使用这个带有 CVS 的 Drupal，能够给你带来哪些好处呢？

· 你可以在正式的安全公告发布以前，对 Drupal 基准代码应用安全更新。我是不是已经提到了很容易便可实现这一点？不需要下载一个完整的 Drupal 最新版本，你只需要简单的运行单个 CVS 命令就可以了。

· 能够维护对 Drupal 代码的自定义修改。修改 Drupal 核心是最不应该做的事情，但是如果你必须修改它的话，那么修改时使用 CVS。即便是你修改了核心文件，CVS 也将聪明的尝试更新，这样在更新流程中，你就不会在不经意间将你的自定义修改覆盖掉了。

· 你也可以使用 CVS 来发现其他开发者对 Drupal 核心文件所做的修改。如果你的 Drupal 工作拷贝与 Drupal 服务器上干净的基准代码有所不同，那么你可使用一个简单的命令，将这些不同之处逐行列出。

安装一个 CVS 客户端

在命令行中运行下面的命令，来测试是否安装了一个 CVS 客户端：

$ cvs

如果你收到了一个“命令未找到”的错误，那么你可能需要安装一个 CVS 客户端。Windows 用户可以看一下 TortoiseCVS(http://tortoisecvs.sourceforge.net/)。而 Mac 用户可参看这篇文章 http://developer.apple.com/internet/opensource/cvsoverview.html。Linux 用户，你自己应该知道怎么做。

如果运行了这个 cvs 命令以后，系统为你输出了下面的 CVS 文档，那么说明你已经安装了 CVS 客户端！

```
Usage: cvs [cvs-options] command [command-options-and-arguments]
```

从 CVS 签出 Drupal

我们将学习如何在命令行中使用 CVS。现在有许多图形化的 CVS 客户端，一旦学会了这些基本的命令以后，你应该能够很容易的实用这些图形化的 CVS 客户端了。Windows 用户通过安装 Cygwin 环境（参看 http://drupal.org/node/150036），就可以使用 CVS 命令行 了。如果你使用的是 CVS 命令行客户端，那么这会便于你与社区中的其它开发者交流 CVS 问题。
用 CVS 行话来说，你将需要从官方的 CVS 资源库中签出一个 Drupal 的工作拷贝。这可能有点唠叨，但是使用正确的术语是非常重要的。下面是一个 CVS 命令，它用来从 CVS 服务器上获取 Drupal 6.2：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

让我们来分解一下这个命令。cvs 执行一个 CVS 客户端；也就是说，它在你的计算机上运行一个名为 cvs 的程序：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

cvs 中的选项–d，代表“目录”，它用来指定 CVS 资源库的位置：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

一个资源库，用 CVS 的话来说，就是一个使用 CVS 维护的文件树所在的位置。现在，如果资源库也位于同一台计算机上，那么 – d 选项就可以简单许 多：cvs -d /usr/local/myrepository。然而，Drupal 的资源库是位于远程服务器上的，所以我们需要指定更多的连接参数。让我们更具体的分 析一下这个命令。

-d 选项的每个参数都使用冒号进行了隔离。pserver 代表“密码验证服务器”，它是用来连接到 Drupal 资源库的连接方法。然而，CVS 也可以通过其它协议进行连接，比如 SSH。

接着，指定了用户名和密码。对于 Drupal CVS 资源库，这两者是一样的：anonymous。跟在@符号后面的是要连接的主机名：cvs.drupal.org。最后，我们需要指定资源库在远程主机上所处的位置：/cvs/drupal。

注意 当你通过了 CVS 服务器的一次认证以后，下次你就不需要再次认证了，因为在你的根目录下将创建一个名为.cvspass 的文件，它存储了登录信息。接下来，应用于这个资源库的 CVS 命令，就不再需要使用这个全局的-d 参数了。

现在，连接参数已经建立，我们可以指定 cvs 实际执行的命令了；在这里，我们使用 checkout 命令从 Drupal 资源库中获取一个工作拷贝：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

不要将下面的-d 与前面的全局选项-d 混淆了：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

这个-d 用来，将资源库的一个工作拷贝存放在你计算机上的 drupal6 目录中，这个目录位于根目录下面的 www 目录的下面。这是一个可选的参数， 如果没有这个参数，那么就使用资源库的名字在本地创建一个文件夹，用来保存工作拷贝。由于在这种情况下，资源库的名字为 drupal，所以它将创建一个名 为 drupal 的文件夹，来保存资源库的工作拷贝。

-r 参数代表“修订本”。一般来说，它应该是一个标签或者分支。我们将在接下来讨论什么是标签和分支。在前面的命令中，我们请求名为 DRUPAL- 6-2 的修订本，这里的 DRUPAL-6-2 是一个标签，对应于 Drupal 6.2 发布。你可以将它替换为 Drupal 当前版本的标签。

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

Drupal 核心的所有标签和分支的列表，可参看 http://drupal.org/node/93997。

最后，drupal 是要签出的资源库的名字。

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout -d
~/www/drupal6 -r DRUPAL-6-2 drupal
```

标签和分支

打标签和做分支是许多修订本控制系统的标准练习。我们将学习一下，如何在 Drupal 核心和贡献的模块中使用这些概念。花点时间好好的理解一下这些概念，可帮你节省不少的时间，并减轻不少的烦恼。

Drupal 核心中的分支

当发布一个新的 Drupal 版本时，,维护者将在 CVS 中创建一个分支，它实际就是当前 HEAD 基准代码的一个克隆。这既允许了在代码的原有主干上继续开 发新特性，而同时也允许了社区来完善一个新的稳定版本。Drupal 6 就是这样创建的。实际的规范分支名有 DRUPAL-4-6-0, DRUPAL-4-7-0, DRUPAL-5，和 DRUPAL-6（注意，在 Drupal5 中，命名规范改变了；删除了第 3 个数字）。

让我们看看它是怎么工作的。在下面的系列图示中，注意时间位于竖向轴上。随着 Drupal 开发的持续进行，bug 修正和新特性被提交到了基准代码中；开发的最前沿（前线）叫做 HEAD，如图 21-1 所示。

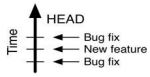

图 21-1. Drupal 开发时间线

当代码足够成熟，值得为其创建一个分支时，Drupal 核心的负责人就会为给定发布在树上创建一个稳定的分支。此时，新建分支和 HEAD 中的代码是 相同的。接着，继续向树的 HEAD 添加新特性和 bug 修正，向稳定的分支添加 bug 修正，如图 21-2 所示。稳定的分支，作为一个规则，它只接收 bug 修 正；而新特性只能添加到 HEAD 中。它们之所以叫做“稳定的”分支，就是因为保证了它们不会被突然改变。

图 21-2.创建了一个分支。

当向稳定分支提交了足够的 bug 修正以后，核心的负责人决定应该再创建一个 Drupal 正式的发布，这样又创建了一个发布。但是，这里的新发布是使用标签创建的，而不是使用分支，下面让我们学习一下标签。

Drupal 核心中的标签

标签就是一个特定分支在某个时间的快照。在 Drupal 世界中，标签用来标记 beta，bug-fix，和安全发布。这样就得到了更小的版本，比如 Drupal 6.1 和 6.2。规范的标签名有 DRUPAL-4-7-1,DRUPAL-4-7-2, DRUPAL-5-7, DRUPAL-6-0, DRUPAL-6-1, 和 DRUPAL-6-2（再次注意，在 Drupal5 中，命名规范改变了）。Drupal 核心使用的标签名字的完整列表，可参看 http://drupal.org/node/93997。

在 Drupal6 还处于正在开发的时候，核心负责人想创建一个 beta 发布，这样就可以方便人们测试代码了。所以他们创建了一个标签，DRUPAL-6-0-BETA-1，如图 21-3 所示。

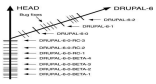

图 21-3.创建了一个标签 DRUPAL-6-0-BETA-1。

标签 DRUPAL-6-0-BETA-1 指的是处于特定状态的代码；也就是，在某个时间点上，代码的一个准确快照。如果你现在真的需要的话，那么你仍然可以使用 CVS 下载 Drupal 的 beta 1 发布。

随着越来越多的 bug 被修正，那么将会使用标签，比如 DRUPAL-6-0-RC-1 和 DRUPAL-6-0-RC-2，来创建一个或多个候选发 布。当代码足够成熟时，将会从基准代码开发中创建一个分支 DRUPAL-6，而 HEAD 则将用于 Drupal7 的开发。这样，值得庆祝的一天到来 了，DRUPAL-6-0 标签创建了；关于 Drupal6 的技术文章、博客，都会疯狂的增加；Drupal6 也将应用于各种实际的网站之中。而在 DRUPAL-6 分支背后，许多 Drupal 程序员将继续为其修正 bug，这样就有了标签 DRUPAL-6-1,

DRUPAL-6-2,以此类推。

-dev 后缀
同时，HEAD 上的开发仍在继续。不过在 Drupal 社区中，一般不把它称作 HEAD，而是把它看作是 Drupal 的下一版本，因为这就是正在开发的真正东西。在图 21-4 中，你可以看到，7.x-dev 就是开发 Drupal 7 的地方。

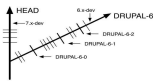

图 21-4.–dev 快照指的是开发的前沿。

当 Drupal 7 正式发布时，核心负责人将为 Drupal7 添加一个稳定的分支，在该分支上将会继续添加新的标签。注意，因此，7.x-dev 不是一个标签！这意味着它 指的不是处于给定状态的代码。更准确的说，它指的是一个分支上的继续开发。每天，drupal.org 上的打包脚本都会从分支上获得一个快照，将它作为一 个“开发快照”提供给大家下载，如图 21-5 所示。不过，这样做只是为了方便；它不是 CVS 的一个特性。

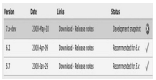

图 21-5.在 http://drupal.org/download 上，你可以下载 Drupal 下一个版本的开发快照。

同样，在稳定的分支上，仍会进行 bug 的修正工作。看一下图 21-4 中的 DRUPAL-6 分支。从图中我们可以看出，在创建了 DRUPAL-6-2 标签以后，已经又修正了一个 bug，但是现在还没有创建新的标签。并不是每次修正一个 bug 后，都创建一个标签的；只有在修正了足够多的 bug 后，核心维 护者判定应该批准一个新发布时，才创建一个标签（例外情况就是，安全性修正，它通常会导致一个立即发布）。当时间到了以后，核心维护者就会创建 DRUPAL- 6-3 标签，这样就创建了一个新发布。

所以，让我们再次回到图 21-4 上；DRUPAL-6 分支上的最新代码，它超过了 DRUPAL-6-2 发布，里面包含了一个在发布以后的 bug 修 正，我们把它称为 6.x-dev。这意味着它是 Drupal 6.3 的开发版本；在核心维护者创建 DRUPAL-6-3 标签时，这一代码将成为 Drupal 6.3。在此以后，分支末端的代码还是 6.x-dev，因为该代码将成为 Drupal 6.4。

提示 当开发者提到分支末端的代码时，他们一般不会停下来检查哪个才是实际的下一版本（它是 6.1? 6.2? 6.3?）。在这种情况下，他们使用一个“x”，来替代“6.1”中的“1”或者“6.2”中的“2”，这样就可以简单把它称为 6.x-dev ----- 也就是，“x”指的是将会成为 Drupal 6 的下一版本的代码，而不管具体的版本号是多少。

现在，你应该能够理解标签和分支的不同了，以及标签是如何关联到核心发布上的。这一信息总结在了表 21-2 中。

表 21-2.标签、分支、发布、Tarball（沓包）之间的关系

标签	出现的分支	发布	Tarball 沓包
DRUPAL-5-7	DRUPAL-5	Drupal 5.7	drupal-5.7.tar.gz
DRUPAL-6-0	DRUPAL-6	Drupal 6.0	drupal-6.0.tar.gz
DRUPAL-6-1	DRUPAL-6	Drupal 6.1	drupal-6.1.tar.gz
DRUPAL-6-2	DRUPAL-6	Drupal 6.2	drupal-6.2.tar.gz
HEAD	None	7.x-dev	drupal-7.x-dev.tar.gz

使用一个标签或分支的名字签出 Drupal

我们在前面的“从 CVS 签出 Drupal”一节中，已经学到了如何根据 DRUPAL-6 分支上的一个标签来获取

代码的一个版本。我们以此为基础，做些变通，来取回各种标签和分支下的代码，并把它们放在当前目录中的 drupal 文件夹下。
签出 DRUPAL-6 分支的一个拷贝，代码与它的第一个 beta 发布完全相同：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout
-r DRUPAL-6-0-BETA-1 drupal
```

签出 Drupal 6.2 的一个拷贝：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout
-r DRUPAL-6-2 drupal
```

签出 6.x-dev 的一个拷贝（也就是 DRUPAL-6 分支上的最近代码，包括最近发布以后的任何 bug 修正）：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout
-r DRUPAL-6 drupal
```

签出 HEAD 的最近版本的一个拷贝（也就是 7.x-dev）。注意，在这种情况下不需要指定分支：

```
cvs -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal checkout
drupal
```

使用 CVS 更新代码

如果你想将你的站点更新到最新的 Drupal 代码上，甚至更新到下一个新版本上，那么你可以使用 cvs update 命令来实现这一点。首先测试一下 cvs update 命令将做出哪些变更，运行下面的命令：

cvs -n update -dP

这将为你显示将被修改的东西，而实际上并未修改。要执行实际的更新，使用这个命令：

cvs update –dP

这将使得你的 Drupal 工作拷贝，与当前分支的最近变更保持同步。CVS 通过查看存储在 CVS 文件夹里的 CVS 元数据来获知当前分支，在你初次执行签出 时，信息就被保存到了那里，所以你不需要每次都指定它。当资源库中已有目录在你的工作拷贝中不存在时，-d 选项将为你创建相应得目录。-P 将清除掉所有无 关的空目录。

注意 在运行任何可能修改文件的 CVS 命令以前，一定要先备份你的数据。另一种最佳实践是，在将这些变更放到实际站点以前，先在一个实验站点上执行 CVS 更新，并解决任何潜在的文件冲突，然后再将这些变更放到实际站点上。

向一个更高的 Drupal 主版本的升级，实际上就是 CVS update 命令的一个变体。我们假定你现在用的是 Drupal 5.7，并想把它升级到 6.2。再提醒一次，在运行下面的命令以前，一定要确定你位于 Drupal 的根目录下。
在当前分支上，更新到最近的正式发布，我们在这个例子中假定它就是 Drupal 5.7。在下面的命令中，实际上并不需要指定 DRUPAL-5-7（因为 cvs 知道你的当前分支），不过罗嗦一点也有好处，它能让你确保所做的变更就是你想要的：

cvs update -dP -r DRUPAL-5-7

警告 如果你正在将 Drupal 更新到一个新版本上，那么你首先应该禁用所有的非核心模块和主题，然后再运行 cvs update 命令来更新 Drupal 核心。更详细的操作指南，可参看 http://drupal.org/upgrade。

接着，将核心代码升级到 Drupal 6。让我们假定 Drupal 6.2 就是最新的版本。这样，下面的命令就会从 CVS 树的 DRUPAL-6 分支上，获取标签为 DRUPAL-6-2 的代码：

cvs update -dP -r DRUPAL-6-2

对于标准更新流程的其余部分，比如更新第 3 方的模块和主题、以及通过访问 update.php 更新数据库，你仍然需要执行，但是现在你不需要下载核心的新版本并覆写你的核心文件了。

追踪 Drupal 代码变更

想检查开发小组中是否有人修改了核心文件？对于核心文件上所做的任何变更，想为其生成一个报告？cvs diff 命令，根据代码的不同之处（也就是更新和修改），为用户生成一个逐行的输出。

注意 在 Unix 命令行上，diff 命令（不是 cvs diff 命令）比较两个文件并为你显示变更。通过键入 diff file1 file2，你就可以使用它了。cvs diff 命令，比较的是本地文件与资源库中的文件，而不是比较两个本地文件。

下面是运行 cvs diff 命令所得到的示例输出，这里使用了 cvs diff –up：

```
Index: includes/mail.inc
===================================================================
=
RCS file: /cvs/drupal/drupal/includes/mail.inc,v
retrieving revision 1.8.2.2
diff -u -p -r1.8.2.2 mail.inc
--- includes/mail.inc 2 Apr 2008 08:41:30 -0000 1.8.2.2
+++ includes/mail.inc 15 May 2008 23:56:40 -0000
@@ -272,8 +272,8 @@ function drupal_html_to_text($string, $a
$string = _filter_htmlcorrector(filter_xss($string,
$allowed_tags));
// Apply inline styles.
- $string = preg_replace('!</?(em|i)>!i', '/', $string);
- $string = preg_replace('!</?(strong|b)>!i', '*', $string);
+ $string = preg_replace('!</?(em|i)((?> +)[^>]*)?>!i', '/',
$string);
+ $string = preg_replace('!</?(strong|b)((?> +)[^>]*)?>!i', '*',
$string);
```

// Replace inline <a> tags with the text of link and a footnote.
// 'See the Drupal site' becomes

以单个加号（+）开头的行是添加进来的，而以单个减号（-）开头的行是被删除了的。它看起来像是有人修改了 drupal_html_to_text()函数中的正则表达式。

Drupal 使用了 unified diffs ，这由-u 选项指出。这里也是用–p 选项；这将在变更总结的后面输出函数的名字。当阅读这个输出时，它能帮你快速判定代码所在的函数，因为并不是所有的 Drupal 开发者都能够记住该行代码位于哪个函数中的。下面的这一行，取自前面的 cvs diff 输出，显示了受影响的函数：

@@ -272,8 +272,8 @@ function drupal_html_to_text($string, $a

解决 CVS 冲突

如果你修改了 Drupal 的核心代码，那么当你执行 CVS 更新时就可能出现冲突。运行完 cvs update 命令以后，对于那些带有冲突的文件将会使用一个“C”将其标出，由于这些冲突的存在（CVS 插入的用来标记冲突的文本，不是有效的 PHP），所以你的站点也将不再工作。CVS 尝试着合并文件的新版本和旧版本，但是它没有成功，所以现在需要人工干预，来手工的检查冲突文件。发生冲突时，包含冲突 的文件就像下面的这样：

<<<<<<< (filename)
your custom changes here
=======
the new changes from the repository
>>>>>>> (latest revision number in the repository)

你需要删除你不想要的那些行，并删除冲突指示字符，来保持代码的整洁。

干净的修改核心代码

你应该尽可能的不去修改核心代码。但是有时候，你可能不得不修改。如果你需要修改，那么要确保能有一种方式让你准确的追踪变更。让我们看一个简单的 例子；我们将编辑 sites/default/default.settings.php 文件。在 143 行，你将看到下面一行代码：

```
ini_set('session.cookie_lifetime',555-5555);
```

这个值用来控制 cookies 的存活时间（单位为秒）。让我们假定数据库中的 sessions 表膨胀的太快了，所以我们需要降低这些会话的有效期。 我们可以直接修改这个值，但是如果资源库中的这一行也被修改了，那么在接下来的 CVS 更新中，我们将得到一个冲突并需要手工的解决这一问题。

一个干净的解决方案，就是将我们想要修改的那行代码注释掉，复制该行代码并将其放在文件中原有代码的下面，然后再做修改：

```
1 /* Original value - Changed to reduce cookie lifetime
2 ini_set('session.cookie_lifetime',555-5555);
3 */
4 ini_set('session.cookie_lifetime',555-5555); // We added this.
```

由于原始的那行代码未被修改，所以运行 CVS 时就不会出现冲突了。

创建和应用补丁

如果你想修正 bug，或者想测试一下他人的潜在的 bug 修正，或者由于这个或者那个原因需要修改核心代码，那么此时你就开始需要创建或者应用一个补 丁了。一个补丁就是用户和计算机都可读的一个文本文件，它根据对 Drupal 代码资源库所做修改，为用户显示逐行的报告。补丁是由 diff(或 cvs diff)程序生成的，我们在前面的“追踪 Drupal 代码变更”一节中，已经看到了一个例子。

创建一个补丁
下面是补丁的示例，它用来整理 includes/common.inc 中 t()函数的文档：

```
Index: includes/common.inc
===================================================================
RCS file: /cvs/drupal/drupal/includes/common.inc,v
retrieving revision 1.591
```

```
diff -u -r1.591 common.inc
--- includes/common.inc 28 Mar 2007 07:03:33 -0000 1.591
+++ includes/common.inc 28 Mar 2007 18:43:18 -0000
@@ -639,7 +639,7 @@
*
* Special variables called "placeholders" are used to signal
dynamic
* information in a string, which should not be translated.
Placeholders
- * can also be used for text that that may change from time to
time
+ * can also be used for text that may change from time to time
* (such as link paths) to be changed without requiring updates to
translations.
*
* For example:
```

在对 includes/common.inc 文件做了修改以后，开发者在 Drupal 根路径下运行下面的命令：

```
cvs diff -up > common.inc_50.patch
```

这个命令获得 cvs diff 的输出，并将其放到名为 common.inc_50.patch 的新文件中。接着开发者访问 drupal.org，并在这里报告 bug：http://drupal.org/node/100232。

应用一个补丁

补丁是根据 cvs diff 或者 diff 命令的输出所创建的文件。在你创建或者下载了一个补丁以后，导航到 Drupal 的根目录并运行下面的命令：

```
patch -p0 < path/to/patchfile/patchfile.patch
```

如果补丁是在 Drupal 安装的根目录下创建的，而你也是在你的 Drupal 安装的根目录应用的，那么路径应该是相同的，所以-p0（这里是一个零）标记用来告诉补丁程序使用在补丁文件中找到的路径（也就是，从路径前缀中去掉 0 部分）。

如果你在应用补丁时遇到了问题，那么可以在 http://drupal.org/node/60116 寻找相关帮助。

有时，为了提升速度或者添加缺失的功能，你可能想为你的实际站点应用一个补丁。这件事的最佳实践是，创建一个 patchescvs diff –up”命令来重新创建补丁。在同一个文件夹下，你还需要创建一个文本文件用来说明每个补丁的应用原因。你可以使用一个命名习惯，使得名字能够提供一些清 晰的上下文信息，例如：文件夹，用来在每个补丁应用后存放它的拷贝。如果你还没有这样做，你可以对文件运行“

```
modulename-description-of-problem-NODEID-COMMENTNUM.patch
```

假定你使用了工作流和令牌（token）模块，但是这两个模块之间整合的不大好。有人提交了一个补丁，修正了这个问题，但是模块的开发者还没有把补丁合并到一个新发布中，而你的站点明天就要上线，你现在就需要它。你应该这样为补丁命名：

```
workflow-conflict-with-token-api-12345-67.patch
```

这样，当需要更新站点时，负责更新的任何人现在都可以断定以下几点：

- 这个安装的哪些部分被修改了？
- 为什么做这些修改？
- 这个补丁现在是不是被合并到了主流代码中？
- 如果没有的话，是不是有人也遇到了同样的问题并提供了更好的解决方案？

维护一个模块

在本节中，我们将详细的学习一下，如何在 drupal.org 上创建和维护一个模块。我们将覆盖大多数的常见任务。

获得一个 Drupal CVS 帐号

Drupal 有两个 CVS 资源库:一个 Drupal 核心资源库,只有很少的开发者能够对它提交代码;一个贡献资源库,用来保存所有贡献的模块、翻 译、主题，这些资源你都可在 drupal.org 上找到，另外还保存了一些文档和方便开发者存储代码片断的沙箱文件夹。如果你有一个模块、主题、或者翻 译，你想把它贡献出来，那么你可以申请一个 CVS 帐号，获取对 Drupal CVS 贡献资源库的写访问权，这样你就可以分享你的代码并向社区回馈你的贡献了。

CVS 帐号不是人人都可以拥有的。你需要向管理员说明你确实需要一个帐号。你需要提供获取帐号的具体动机。如果你打算贡献一个模块的话，你需要提供 模块的一个拷贝以供审查，并证明它与已有的模块存在重大的不同。（你可以在 drupal.org 上花点时间，使用搜索表单搜索一下，确保你的模块是新的并且是不同的。在高级搜索表单上，选中工程复选框，这样你就只在贡献的资源中进行搜索了。）还有，确保你可以接受 GNU GPL 许可，因为贡献资源库中的所有代码都必须采用 GPL 许可。

关于如何申请的更多详细，可参看 http://drupal.org/node/59。提交代码和维护你自己贡献的模块的更多详细，可参看 Drupal 站点的 http://drupal.org/handbook/cvs/quickstart；另外，我们接下来将会详细的介绍大多数的常见 任务。

另外，还有许多其它的方式来为 Drupal 社区贡献力量，比如编写文档和参与论坛讨论；具体可参看 http://drupal.org/node/22286。

签出贡献资源库

我前面提到了，drupal.org 有两个资源库，一个用于核心代码，一个用于贡献的代码包括模块和主题。对于前者，只有很少的人能够访问；而对于 后者，许多开发者都可以访问。你可以以匿名或者登录用户的身份，来签出贡献资源库。如果你是为了一个站点从贡献资源库签出代码的话（例如，你只想使用 CVS 获取一个模块的拷贝，这样你就可以运行它了），那么最好使用匿名用户的身份进行签出。否则，当下一个人来维护你创建的 Drupal 站点时，他想从 CVS 上更新模块代码，而系统则提示需要输入你设置的密码，那么此时他会晕死的！
你可以签出整个资源库：

```
cvs -z6 -d:pserver:anonymous:xxxxx@xxxxx.com:/cvs/drupal-contrib
checkout contributions
```

然而，并不鼓励你这样做，因为下载的太多了，会加重服务器的负担。最好只下载自己需要的。假定你创建了一个模块，并想将它贡献给 Drupal 社区。 这意味着你只需要资源库中的 modules 子目录。如果你想把代码提交到资源库，那么你需要登录进来（你需要 CVS 帐号和密码；参看“获得一个 Drupal CVS 帐号”）。假定你的 CVS 用户名为 sproinx，下面给出了如何登录进来：

```
cvs -d:pserver:xxxxx@xxxxx.com:/cvs/drupal-contrib login
```

系统会提示你输入密码，就是你申请 CVS 帐号时提供的。这个密码与你的 drupal.org 密码可以不一样。

提示 登录到 drupal.org，点击“我的帐户”，点击编辑链接，并点击 CVS 标签，这样你就可以修改你的 CVS 帐号的密码了。

接下来，你可以签出贡献资源库（就是 drupal-contrib 资源库）的 modules 子目录。你可以签出 modules 子目录所包含的所有模块，不过很少这样做，除非你想在一个长途飞行期间细读所有模块的拷贝：

```
cvs -z6 -d:pserver:xxxxx@xxxxx.com:/cvs/drupal-contrib checkout
contributions/modules
```

这将在你的本地计算机上创建一个 modules 子目录的拷贝。它应该看起来这样的：

```
contributions/
  CVS/
  modules/
    a_sync/
    aapi/
    about_this_node/
    abuse/
    ...
```

或者，你可以只签出 modules 子目录，而不带有它所包含的模块，大多数的开发者都这样做：

```
cvs -d:pserver:xxxxx@xxxxx.com:/cvs/drupal-contrib checkout
-l contributions/modules
```

注意 在编写本书时，modules 子目录已经包含了 300MB 的数据。这就是为什么在签出该子目录时，在 CVS 命令中使用-z6 标记（-z6 在将数据通过网络传输前，会先对其进行压缩），或者使用-l 标记，省略对所有模块的签出。

注意在前面的 CVS 命令中，参数-d:pserver:xxxxx@xxxxx.com:/cvs/drupal-contrib 是重复的。由于每次都输入一遍这个参数，这样会很不方便，所以聪明的开发者可以把它放在 CVSROOT 环境变量中：

```
export CVSROOT=:pserver:xxxxx@xxxxx.com:/cvs/drupal-contrib
```

从现在起，CVS 命令就会简短很多。设置了 CVSROOT 以后，前面的命令将会变成这样：

```
cvs login
cvs -z6 checkout contributions/modules
cvs checkout -l contributions/modules
```

从现在起，我假定已经设置了 CVSROOT 环境变量。

将你的模块添加到资源库

现在你有了一份贡献资源库中 modules 子目录的拷贝，你可能想现在就可以将你的模块其它上千个模块

放到一起了。我们先不要急！首先，花点时间调查一下资源库中是不是已经有一个模块解决了你的问题。下面是一些资源，可帮你确定这一点：

・允许你根据类别，名字，或日期浏览模块，还可以使用主发布兼容性（Drupal 6, Drupal 5,等等）过滤模块。
•http://drupal.org/node/23789 概括了一些基本的方式，如何在他人的基础之上贡献自己的力量。
•http://drupalmodules.com 可方便的用来搜索第 3 方模块，同时还可以对模块进行评价和打分。

如果你觉得你的模块值得编写，现在就可以开发它了。让我们创建一个模块。

下面是.info 文件：

```
1 // $Id$
2 name = Foo
3 core = 6.x
```

而下面是模块本身：

```
1 <?php
2 // $Id$
3
4 /**
5 * @file
6 * The greatest module ever made.
7 */
```

现在，模块的目录就包含了前面的两个文件，它看起来应该这个样子：

```
foo/
  foo.info
  foo.module
```

继续前进，将新模块复制到你新签出的贡献资源库中：

```
cp -R foo /path/to/local/copy/of/contributions/modules
```

接着，将新目录告诉 CVS：

```
cd /path/to/local/copy/of/contributions
cvs add modules/foo
```

把目录中的文件也添加进来：

```
cvs add modules/foo/*
```

CVS 将提醒你，尽管这些文件的添加已被预订了，但是你还需要提交它们：

```
cvs add: 使用`cvs commit'来永久的添加这些文件
```

如果你的模块包含子目录，由于 CVS 不能递归的添加，所以你也需要添加这些子目录：

```
cvs add modules/foo/subdir1
cvs add modules/foo/subdir1/*
```

初始提交

现在到了关键时刻了。是时候将你的文件提交到资源库中了！是不是有点紧张。检查/path/to/local/copy/of /contributions/modules/foo，看看它是不是包含了所有的文件并且里面包含了你想要提交的代码。接着，输入决定性的命令。使用一 个简洁的句子来描述你模块的功能，接着继续前进：

```
cvs commit -m "Initial commit of foo module. This module sends
badgers to those who use it."
```

-m 标记意味着，后面引号内中的内容是一个消息，它将和代码提交被一同记录下来。在你的消息中，你应该提供一些有用的信息。如果你想输入多行的文本，而你 安装的 CVS 自动打开了一个文本编辑器，那么省略-m 标记可能会有所用处。在我的 OS X 机子上，它打开了一个 vim 编辑器，给我显示了一个如下所示的界面：

```
CVS:
----------------------------------------------------------------------
----
CVS: Enter Log. Lines beginning with `CVS:' are removed
automatically
CVS:
CVS: Committing in .
CVS:
CVS: Added Files:
CVS: foo.info
CVS: foo.module
CVS:
----------------------------------------------------------------------
----
~
~
```

如果你以前从没有用过 vim，这可能会有点恐怖。使用向下箭头键导航到以“CVS:”开头的最后一行，接着按下“o”键（就是“oh boy!”中的“o”）。接着键入你的长一点的提交消息，完成以后，按下 Esc 键，接着键入:wq 来退出程序。通过设置 CVSEDITOR 环境变量，你就 可以使用自己喜欢的编辑器了；例如，对于 emacs，可以这样设置变量：

```
export CVSEDITOR=emacs
or like this for Textmate:
export CVSEDITOR="mate -w"
```

签出你的模块

现在，你的模块已被提交到资源库中了，这和其它的第 3 方模块一样，你可以将它从 CVS 签出，并放在你的 Drupal 本地开发拷贝中（你可能首先需要创建 modules 和 contrib 目录）：

```
cd /path/to/drupal
cd /sites/all/modules/contrib
cvs checkout -d foo contributions/modules/foo
```

如果你收到了一个如下所示的错误，那么你应该没有设置你的 CVSROOT 环境变量（参看“签出贡献资源库”）。

```
cvs checkout: No CVSROOT specified! Please use the `-d' option
cvs [checkout aborted]: or set the CVSROOT environment variable.
```

提示 如果你维护的 Drupal 站点使用了从 CVS 签出的模块，那么你可以研究一下 CVS 部署模块，位于 http://drupal.org/project /cvs_deploy。它将模块的 CVS 信息和 Drupal 内置的更新状态模块整合在了一起，后者用来报告哪些模块需要更新。

在 drupal.org 上创建一个工程

由于你把你的模块贡献给了社区，如果能够使用一种结构化的方式，让模块的其它用户能与你进行交互，这应该是再好不过了。这样，你就不会经常收到哪些 不期而遇的电子邮件了，而且还有一种标准的方式用来追踪请求的特性、bug 修正、等等。登录到 drupal.org 以后，访问 http://drupal.org/node/add/project，或者使用站点导航菜单导航到“创建内容➤工程”，然后填充表单；你需要格外注意 一下“完整描述”字段，在这里你可以描述你的模块（或者主题）。填完表单以后，你就可以访问你的工程了，工程地址为：http://drupal.org /project/yourprojectname。

警告 在你执行任何标签化或分支化以前，一定要创建一个工程。

提交一个 bug 修正

如果你的签出正常工作了，那么在 sites/all/modules/contrib 下，应该包含以下内容：

```
1       foo/
2           CVS/
3           foo.info
4           foo.module
```

我们刚刚分享了我们的代码，就已经有人在我们的问题列表 http://drupal.org/project/yourprojectname /issues 中，创建问题了。drupal.org 上的用户 flyingpizza 在一篇文章 http://drupal.org/node /555-5555 中指出，我们忘记在我们的.info 文件中添加一个描述了！现在让我们添加一个描述：

```
1 // $Id: foo.info,v 1.1 2008/05/22 14:15:21 jvandyk Exp $
2 name = Foo
3 description = Sends badgers to those who use it.
4 core = 6.x
```

注意，文件中的第一行，已经被 CVS 从“// Id”修改为文件的实际标识信息了。如果你看到的仍然是“// Id”，那么你使用的版本应该不是从 CVS 中签出的。

在我们提交这个变更以前，通过运行 cvs diff 命令，让我们预览一下我们的变更：

cvs diff –up

输出如下所示：

```
01 RCS file: /cvs/drupal-
contrib/contributions/modules/foo/foo.info,v
02 retrieving revision 1.1
03 diff -u -u -p -r1.1 foo.info
04 --- foo.info 22 May 2008 14:15:21 -0000 1.1
05 +++ foo.info 22 May 2008 14:21:54 -0000
06 @@ -1,3 +1,4 @@
07 // $Id: foo.info,v 1.1 2008/05/22 14:15:21 jvandyk Exp $
08 name = Foo
09 +description = Sends badgers to those who use it.
10 core = 6.x
```

注意，在输出中，我们新加的一行的前面有个+字符。现在让我们继续前进，提交这个变更：

```
1 cvs commit -m "#555-5555 by flyingpizza: Added missing
description line."
```

提交消息中的#555-5555，将被自动修改为一个超链接 http://drupal.org/node/555-5555，显示在 drupal.org 的提交日志中（例如，位于 http://drupal.org/cvs）。

提示 提交消息应该简洁并具有描述性，同时应该指出哪些人为这个提交做出了贡献。里面应该包括问题的节点号，提交补丁的用户的名字，以及为这个提交做出其它贡献 的用户的名字。这样，我们就可以从 CVS 提交消息中方便的链接到 drupal.org 上的节点 555-5555 的讨论了。如果你在怀疑是否应该在提交消息中包含某人的用户名，那么还是慷慨一点， 把他人的贡献也列出来。通过指出贡献者的工作，比如修正了 bug，这样可以给贡献者增加信誉，对于你自己所做的贡献，在提交时，也要恰如其分的指出。把 Drupal 社区的团结之爱传遍整个地球！

漂亮。图 21-6 给出了我们到目前为止的开发工作。

图 21-6. foo 模块的开发

查看一个文件的历史

你可以使用 cvs log 命令来查看一个文件的历史。让我们看看 foo.info 文件的两次提交：

cvs log foo.info

```
----------------------------
revision 1.2
date: 2008-05-22 09:28:25 -0500; author: jvandyk; state: Exp;
lines: +2 -1;
commitid: LYpsSr1ZkEut7Y3t;
"#555-5555 by flyingpizza: Added missing description line.
----------------------------
revision 1.1
date: 2008-05-22 09:15:21 -0500; author: jvandyk; state: Exp;
commitid: wcK48PdiM0yZ2Y3t;
Initial commit of foo.module. This module sends badgers to those
who use it.
```

创建一个分支

我们知道，还有许多懒惰的用户还在使用 Drupal5，现在让我们看看如何为他们创建一个分支。

警告 只有在 drupal.org 上为你的模块创建了一个工程以后，才能为你的模块创建一个分支。

首先，你需要确定你使用的是 HEAD 的最近版本：

cvs update –dP

另一种检查的方式是，使用 CVS 状态命令。让我们检查一下 foo.info 文件的状态：

cvs status foo.info

```
===================================================================
File: foo.info Status: Up-to-date
Working revision: 1.2 2008-05-22 09:28:25 -0500
Repository revision: 1.2 / cvs/drupal-contrib/
contributions/modules/foo/foo.info,v
Commit Identifier: LYpsSr1ZkEut7Y3t
Sticky Tag: (none)
Sticky Date: (none)
Sticky Options: (none)
```

注意上面所列的状态为 Up-to-date（最近的）。这意味着你的本地文件与资源库中的文件完全相同。如果你修改了你的本地文件却没有提交到资源 库，或者你忘记删除一些调试代码了，那么状态将变为 Locally Modified（本地已修改）。还有 Sticky Tag 字段的值为(none)，这意味着你在使用 HEAD。

创建一个兼容 Drupal5 的分支

让我们继续前进，来创建分支：

cvs tag -b DRUPAL-5

不要被这里的单词 tag 搞晕了。由于我们加了-b 选项，所以这里创建的是一个分支，而不是一个标签（更准确的说，一个分支就是一个标签，一个特殊的 标签，不过我们这里将其简单化一点）。执行了上面的命令以后，模块的开发历史就带有了新的 DRUPAL-5 分支了，如图 21-7 所示。

图 21-7.带有 Drupal5 分支的模块开发历史

注意，由于我们还没有做任何修改，所以此时代码在两个分支中还是完全相同的。

现在让我们看一个现实的问题。我们的模块依赖于 badger（徽章）模块！而我们尚未在.info 文件中对此做出声明。此外，Drupal5 描述依 赖关系的语法与 Drupal6 及以后版本的语法不一样。所以，让我们为 DRUPAL-5 分支添加 Drupal5 版本。但是如何判定本地工作空间包含了什么 呢？本地空间中的那些文件是来自于 DRUPAL-5 分支，还是来自于 HEAD？让我们声明我们需要 DRUPAL-5 分支的文件：

cvs update -dP -r DRUPAL-5

这个命令的意思是说，“获取 DRUPAL-5 分支的文件，创建任何需要的新目录，删除任何不再需要的空目录”。现在让我们修改.info 文件：

```
// $Id: foo.info,v 1.2 2008/05/22 14:28:25 jvandyk Exp $
name = Foo
```

```
description = Sends badgers to those who use it.
dependencies = badger
```

让我们查看这些变更：

cvs diff -up foo.info

```
===================================================================
=
RCS file: / cvs/drupal-
contrib/contributions/modules/foo/foo.info,v
retrieving revision 1.2
diff -u -u -p -r1.2 foo.info
--- foo.info 22 May 2008 14:28:25 -0000 1.2
+++ foo.info 22 May 2008 16:40:53 -0000
@@ -1,4 +1,4 @@
// $Id: foo.info,v 1.2 2008/05/22 14:28:25 jvandyk Exp $
name = Foo
description = Sends badgers to those who use it.
-core = 6.x
+dependencies = badger
```

注意我们删除了“core = 6.x”（因为这是个 Drupal6 特性，而现在是在 DRUPAL-5 分支上），让我们查看一下状态：

cvs status foo.info

```
===================================================================
=
File: foo.info Status: Locally Modified
Working revision: 1.2 2008-05-22 09:28:25 -0500
Repository revision: 1.2 / cvs/drupal-contrib/
contributions/modules/foo/foo.info,v
Commit Identifier: LYpsSr1ZkEut7Y3t
Sticky Tag: DRUPAL-5 (branch: 1.2.2)
Sticky Date: (none)
Sticky Options: (none)
```

注意，状态现在为 Locally Modified（本地已修改），而 Sticky Tag 字段指示我们使用的是 DRUPAL-5 分支。

最后，让我们提交变更：

cvs commit -m "Drupal-5-compatible dependency on badger module."

图 21-8 给出了现在的模块开发历史。

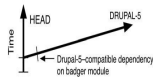

图 21-8.向 DRUPAL-5 分支提交代码后，模块的开发历史

标签化和创建一个发布

现在模块已经可用于 Drupal 5 了。让我们继续前进，来创建一个发布。我们将通过创建一个标签来实现这一点。

注意 一个标签（tag）就是给处于特定状态的文件贴个标签（label）。当用户下载加了标签的代码时，他/她得到的文件与贴标签时的状态完全一样。这就是为什么标签化在创建发布时非常有用。

记住，一个标签就意味着一个发布。由于这是 DRUPAL-5 分支上我们代码的第一个发布，所以我们知道该标签应该为 DRUPAL-5--1-0。图 21-9 说明了该标签的实际含义。

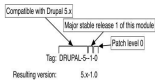

图 21-9.在标签名字和生成的模块版本之间的关系

在贴标签以前，最好先使用 cvs status 看看状态，以确定当前的文件就是你想要的。接着继续前进，来创建标签：

cvs tag DRUPAL-5--1-0

回顾一下我们模块的开发历史，如图 21-10 所示。

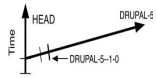

图 21-10.开发历史显示了 DRUPAL-5 分支上的标签

创建一个兼容 Drupal6 的分支

我们已经为 Drupal5 创建了一个分支，并在该分支上创建了一个标签。现在让我们把精力主要放在 Drupal 6 上，来添加对 badger（徽章）模块的依赖关系。但是首先，我们需要做出一个决定。我们是应该立即创建一个分支呢？还是应该简单的使用 HEAD？由于 我们可以在任何想要的地方创建标签，所以这个问题就是，又没有必要创建一个 DRUPAL-6 分支？让我们检查一下这两种不同的方式。

为发布使用 HEAD
用于新发布的一种方式是，编辑 HEAD 上的 foo.info 文件来添加依赖关系。首先，由于我们刚才使用的是 DRUPAL-5 分支下的文件，所以我们需要从 HEAD 获取文件，并将其放入到本地工作空间中。你可能会认为使用下面的命令就可以了：

cvs update -dP -r HEAD

然而，这将在你的本地工作空间生成一个粘性标签，如果你想使用设置为 HEAD 的粘性标签执行提交的话，那么你将收到一个如下所示的错误：

```
cvs commit: sticky tag `HEAD' for file `foo.info' is not a branch
cvs [commit aborted]: correct above errors first!
```

解决方案是使用下面的命令，它将重置粘性标签：

cvs update –A

现在，让我们添加依赖信息，这里使用了 Drupal6 所用的方括号格式：

```
// $Id: foo.info,v 1.2 2008/05/22 14:28:25 jvandyk Exp $
name = Foo
description = Sends badgers to those who use it.
dependencies[] = badger
core = 6.x
```

你可以使用 cvs diff 和 cvs status 来检查变更。接着提交变更：

cvs commit -m "Drupal-6-compatible dependency on badger module."

图 21-11 显示了我们最近的变更。

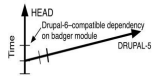

图 21-11.开发历史显示 HEAD 上有一提交

由于现在我们还不为 Drupal 7 做任何开发工作，所以我们可以使用 HEAD 来放置新的开发工作。这意味着少加了一个分支，也就少添了一些麻烦。分支少，就意味着提交 bug 修正的地方少 了。让我们继续前进，为 Drupal6 创建第一个标签。由于它是兼容 Drupal 核心 6.x 系列的第一个发布，所以我们把它叫做 DRUPAL- 6--1-0：

cvs tag DRUPAL-6--1-0

这个标签是创建在 HEAD 上的，如图 21-12 所示。

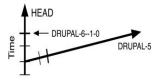

图 21-12.一个应用于 CVS 树的主干的标签

假定我们继续更新模块，又创建了多个提交和发布。那么我们的开发历史，将很快变成图 21-13 所示的样子。

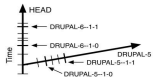

图 21-13.为 Drupal6 发布使用 HEAD 所做的开发

创建一个 Drupal6 分支

当 Drupal7 出来以后,我们想继续开发 Drupal6 下的模块。而现在, DRUPAL-5 分支的开发工作基本上停止不前了。但是我们也不能在 HEAD 上同时开发 Drupal 7 和 Drupal 6 下的版本啊？现在需要为 Drupal 6 创建一个分支，将特定于 Drupal 6 的开发放在那里进行。首先，我们需要确定我们使用的是 HEAD 的最近版本。接着，为 Drupal 6 创建分支。

```
cvs update -A
cvs tag -b DRUPAL-6--1
```

现在，我们的开发历史就如图 21-14 所示了。

图 21-14.为 Drupal 6 创建一个分支。

稍等一下！为什么对于 DRUPAL-6 分支，我们使用了 DRUPAL-6--1，而没有使用 DRUPAL-6？答案很简单：从 Drupal 6 开始，分支的名字能够更具体的描述它们的含义了（更多详细可参看 http://drupal.org/node/147493）。你为 DRUPAL-6 分支将要创建的标签是用于发布的 6.x-1.x 系列的。这意味着这些发布与 Drupal 6 的任何版本都兼容，并且属于发布的第一个系列。图 21-15 显示了对应于你模块的 6.x-1.2 和 6.x-1.3 发布的标签；图 21-16 显示了对应于标签的发布。

图 21-15.显示标签名的 6.x-1.x 发布系列

图 21-16.显示发布版本号的 6.x-1.x 发布系列

当 Drupal7 出来以后,我们想继续开发 Drupal6 下的模块。而现在, DRUPAL-5 分支的开发工作基本上停止不前了。但是我们也不能在 HEAD 上同时开发 Drupal 7 和 Drupal 6 下的版本啊？现在需要为 Drupal 6 创建一个分支，将特定于 Drupal 6 的开发放在那里进行。首先，我们需要确定我们使用的是 HEAD 的最近版本。接着，为 Drupal 6 创建分支。

标签和版本号

在分支名字比如 DRUPAL-6—1 和标签名字比如 DRUPAL-6--1-3 中，我们看到有两个连字符是连着的。如果你把紧挨着 6 的连字符看作是 Drupal 的一个发布的通配符，那么就不难理解了。也就是说，DRUPAL-6--1-3 标签，对应于你模块的 6.x-1.3 发布，它与 Drupal 6 的任意发布都兼容（Drupal 6.1, Drupal 6.2, Drupal 6.3,等等）。把标签名中主版本号后面的连字符，想象成可以翻译为发布号的 x，如下所示：

译者注： 假定我们创建了如下的标签名，DRUPAL- 6-1-1-3，DRUPAL-6-2-1-3，DRUPAL-6-3-1-3，这样这里面就有了 4 个数字，它们使用连字符分割，第一个数字就是 Drupal 的主版本号，第 2 个数字就是 Drupal 在主版本下的具体发布号，第 3 个数字就是模块的主版本号，第 4 个数字就是模块在主版本下的具体发布 号，这样以来，就很好理解 DRUPAL-6--1-3，在这里我们可以把它看成第 2 个数字为空的情况，而这种情况在默认情况下，匹配 Drupal6 下的所 有具体发布。也就是说，省略第 2 个数字，表示模块兼容于 Drupal 主版本下的所有具体发布。我觉得这样解释，比上面的更容易理解一点。

现在，我们为 Drupal 6 创建了一个分支，我们可以继续使用 HEAD 来开发 Drupal7 下的版本----直到 Drupal8 问世，那时我们需要为 Drupal7 创建一个分支。 图 21-17 显示了 Drupal 7 开发的样子。这种方式与前面我们创建 DRUPAL-5 分支后开始 Drupal 6 开发的方式完全一样。

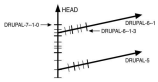

图 21-17.Drupal6 开发位于自己的分支下，而 Drupal7 开发则使用 HEAD

通过程序来检查你的编码风格

有两种主要的方式，可用来检查你的编码风格是否符合 Drupal 的编码标准：一种方式是使用一个 Perl 脚本，另一种方式是使用一个第 3 方模块。

使用 code-style.pl
在你的 Drupal 根目录下的 scripts 目录中，你可以找到一个名为 code-style.pl 的 Perl 脚本，它可用来检查你的 Drupal 编码风格。下面讲述如何使用这个脚本。

首先，修改文件的权限从而让它可执行；否则，你将得到一个“权限被拒绝”错误。使用 chmod，通过命令行就可以实现这一点，如下所示：

```
$ cd scripts
$ ls -l | grep code-style
-rw-r--r-- 1 jvandyk jvandyk 4946 Feb 15 2007 code-style.pl

$ chmod u+x code-style.pl
$ ls -l | grep code-style
-rwxr--r-- 1 jvandyk jvandyk 4946 Feb 15 2007 code-style.pl
```

Windows 用户不用修改文件的权限，但是在你运行 code-style.pl 以前，一定要确保 Perl 已经安装了。关于 Perl 的更多详细，可参看 http://www.perl.org。

现在，只要把将被评价的模块或者其它文件的位置传递给 code-style.pl，你就可以执行这个脚本了。下面的例子说明了执行脚本的命令：

$ ./code-style.pl ../modules/node/node.module

程序的输出通常采用下面的格式：

line number : 'error' -> 'correction' : content of line

例如，下面的脚本告诉我们，我们需要在 foo.module 的 30 行的赋值操作符（=）周围添加空格，现在包含的代码为“$a=1;”：

foo.module30: '=' -> ' = ' : $a=1;

注意 一定要同时了解优点和不足。这一脚本做的很不错，但它还没有十全十美，所以你需要细心的评估每一个报告。

高级分支化

在前面的例子中，我们假定在一个 Drupal 主版本下只存在模块的一个主版本，但是这也有例外的情况。例如，假定我们发布了 foo 模块的 6.x- 1.3 版本。接着，灵感爆发了。我们想到了另外的一种实现方式，只需要一半的代码量，就可以实现同样的功能，而且跑得更快。不过，这需要修改 API，而与 foo 模块相关的一切将全被打乱。解决的方案是使用新 API 发布一个 2.0 版本。由于模块仍然兼容于 Drupal 6，所以我们使用 DRUPAL-6--2-0 作为标签名，而对应的发布号就是 6.x-2.0。

我们可以把代码提交到 DRUPAL-6—1 分支中，然后告诉他人 6.x-1.3 版是我们模块的 1.x 系列的最终发布了。但是如果安全小组在我们的模 块中找到了一个安全漏洞，我们不得不发布 6.x-1.4 版，那该怎么办呢？所以，对于我们重写的模块，不能为它使用 DRUPAL-6—1 分支。

那么解决方案呢？创建一个新分支，在那里我们发布模块的 2.0 版。我们把新分支命名为 DRUPAL-6--2，在已有的 DRUPAL-6--1 分支上做出 DRUPAL-6--2 分支。首先，确定我们得到了 DRUPAL-6--1 分支下文件的最近版本。接着创建新分支：

```
cvs update -dP -r DRUPAL-6--1
cvs tag -b DRUPAL-6--2
```

我们的开发历史，现在变成了图 21-18 所示的样子。

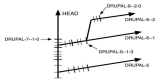

图 21-18.在已有的稳定分支上创建一个新分支

另一种可选的方式是，如果我们还没有为 Drupal 7 做任何开发，那么对于模块的 2.0 发布的开发，我们可以为其使用 HEAD,最后，我们再从 HEAD 上为 2.0 系列的发布创建一个分支（这和 DRUPAL-6—1 分支的创建方式一样）。这种方式如图 21-19 所示。

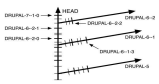

图 21-19.从 HEAD 上为模块的第 2 个稳定分支创建一个分叉，模块的这个主版本也支持 Drupal 的同一个主版本(Drupal 6)

这两种方式的具体选择，取决于实际情况。一般来讲，尽可能的不去创建新分支，直到必须创建的时候再为其创建。在图 21-19 中，创建 DRUPAL- 6--2 分支的决定因素是开始为 Drupal 7 开发了模块了。

如果没有 DRUPAL-6--2 分支的话，那么模块的 2.0 系列的开发将还会占据着 HEAD，这样在 CVS 树中，就没有地方进行 Drupal 7 下的开发了。

创建一个发布节点

为了使那些不熟悉 CVS 的人也可以下载你的模块，你应该在 drupal.org 上创建一个发布节点。一个发布节点提供了给定发布标签的相关信息，而 drupal.org 上的打包脚本能够自动的为发布标签指示的文件构建一个 tarball（沓包）。例如，你能为你模块的 DRUPAL- 6--1-3 标签创建一个发布节点。打包脚本从 DRUPAL-6—1 分支取出 DRUPAL-6--1-3 标签所标示的文件，然后为其创建一个 tarball（沓包）和一个链接，这样 drupal.org 上的访问者就可以下载这个 tarball（沓包）了。而 tarball（沓包）的名字则应该 为 foo-6.x-1.3.tar.gz。

为了创建一个发布节点，我们导航到我们在 http://drupal.org 创建的工程页面（参看“在 drupal.org 上创建一个工程”），并 点击“添加发布”链接。接着选择这个发布所代表的 CVS 标签，并指出这个发布中的变更是安全更新呢，还是 bug 修正，还是新特性。

在发布节点的正文中，应该列出这个特定发布的新功能。可以把这些看作是发布笔记。我们还应该列出都解决了哪些问题，最好能够给出指向问题节点的链 接。图 21-20 给出了一个典型的发布节点。有一个方便的脚本，位于 http://cvs.drupal.org/viewvc.py/drupal /contributions/tricks/cvs-release-notes，它能帮我们自动生成一列修正的问题。

图 21-20.漂亮菜单（nice menus）模块的 6.x-1.1 版的发布节点

一旦创建了发布节点并运行了打包脚本以后，发布节点将被添加到工程页面，任何人都可以在这里下载 tarball（沓包），或者阅读你输入的发布笔记，如图 21-21 所示。

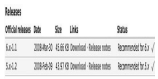

图 21-21.在工程页面可下载已发布的沓包。

在工程管理中混合使用 SVN 和 CVS

Drupal 的核心代码使用了 CVS 的，然而你工程的其余部分，可能完全没有使用版本控制，也可能使用了一个不同的版本控制系统。

通常的实践是，使用另外一个的不冲突的版本控制系统比如 Subversion(SVN)，来将整个工程（包括 Drupal 和它的 CVS 元数据）存放 在它自己的资源库中。这一想法是，我们首先对 Drupal 核心执行 CVS 更新操作（从 cvs.drupal.org 获取变更），接着切换到 SVN，使用 SVN 提交这些变更（这将它们放进你的 SVN 资源库中）。你可以使用这个 SVN 资源库，来存放任何自定义的模块、主题、图片、甚至你的工程的数据库模式。

注意 关于 Subversion 的更多详细，可参看 http://subversion.tigris.org。

测试和开发代码

软件测试就是将一个程序隔离成不同的部分，来判定它们的行为和预期的是否一致。在 Drupal 的接下来的版本中，测试将会是一个主要的目标。事实上，在 Drupal 7 中，测试将成为核心的组成部分。测试的好处包含以下几点：

・如果代码的变更（例如，代码重构）破坏了软件，那么这将被即刻发现。

•将检查代码错误的流程自动化。自动测试位于 http://testing.drupal.org，它用来为核心检查引入的补丁。

•确保新代码正常工作。

关于测试的更多信息，参看 http://drupal.org/simpletest。你还可以参与测试小组，它位于 http://groups.drupal.org/unit-testing。

`devel` 模块

Devel 模块是个大杂烩，里面包含了许多实用功能，开发者可用它来调试和检查代码的各种细节。

你可以从 http://drupal.org/project/devel 下载该模块（或者使用 CVS 签出并获得更酷的效果）。安装了 devel 模块后，一定要启用 devel 区块。下面是 devel 区块中一些含义比较模糊的链接，这里给出了明确的解释：

· Empty cache（清空缓存）:这个将执行 includes/common.inc 中的 drupal_flush_all_caches()函数。这与你导航到 “管理➤站点配置➤性能”点击“清除缓存数据”的效果是一样的。也就是说，CSS 和 JavaScript 缓存被清空了；新压缩的 CSS 和 JavaScript 文件被重新命名，这样就强制客户下载新文件了；主题注册表被重新构建；菜单被 重构；node_type 表被更新；数据库中的缓存表，用来存储页面、菜单、节点、区块、过滤器、和变量缓存的表，也全被清空了。具体一点，清空的缓存表 有 cache, cache_block, cache_filter, cache_menu, 和 cache_page。第 3 方模块的自定义缓存表，如果模块实现了 hook_flush_caches()（它返回一个数组，里面包含了要被清空的缓存 表的名字），那么这些缓存表也被清空。

· Enable Theme developer（启用主题开发者）：这个链接用来启用主题开发者模块，启用这个模块后，将鼠标指向一个页面元素，就会为你显示出用来生成这个页面元素的模板或主题函数了（参看第 8 章）。

· Function reference（函数引用）：这个函数提供了一列用户函数，这些函数是在这个请求期间使用 PHP 的 get_defined_functions()定义的。点击一个函数的名字，可以查看它的文档。

· Hook_elements(): 这个链接使用一种便于阅读的格式来显示 hook_elements()调用的结果，在使用表单 API 时，它会非常有用。

· Rebuild menus（重构菜单）：这个调用 menu_rebuild()，它将清空并重新构建 menu_router 表，同时更新 menu_links 表（参看第 4 章）。

· Reinstall modules（重装模块）：通过运行 hook_uninstall()和 hook_install()来重新安装一个模块。模式版本号将被设置成最近更新的号码。在重新安装模块以 前，对于任何已有的数据库表，如果其对应模块没有实现 hook_uninstall()，那么一定要手动的将其清除。

· Session viewer（会话查看器）：使用这个链接来显示你的$_SESSION 变量的内容。

· Variable editor（变量编辑器）:这个链接列出当前存储在 variables 表中以及你的 settings.php 文件的$conf 数组中的所有变量及它们的 值，它还允许你对这些变量进行编辑。一般可以使用 variable_get()和 variable_set()来访问这些变量。

显示查询

打开页面 http://example.com/?q=admin/settings/devel（如果你启用了开发区块的话，那么还可点击 “Devel 设置”链接），选中“Collect query info”（收集查询信息） 和 “Display query log”（显示查询日志）旁的复选框。

一旦你保存了这些设置，你将会看到，在每个页面的最底部，都有一列查询，这些查询就是为生成当前页面所用到的所有查询语句！此外，列表还会告诉你生成查询的函数，该查询所耗费的时间，以及查询的调用次数。

你可以在许多方式中使用这一信息。例如，如果同一查询在单个页面中被调用了 40 次，那么你就需要检查一下，你的代码中是否存在一个坏的控制结构循 环。如果确实如此的话，那么你可以考虑使用一个静态变量，在请求期间，来保存数据库查询结果。下面是一个例子，它给出了这个设计模式的大致样子（来自于 modules/taxonomy/taxonomy.module）：

```
01 function taxonomy_get_term($tid) {
02 // Define a static variable to hold data during this page
request.
03 static $terms = array();
04
05 // Look in the static variable and only hit the database if the
data
06 // for this term ID has not already been retrieved.
07 if (!isset($terms[$tid])) {
08 $terms[$tid] = db_fetch_object(db_query('SELECT * FROM
{term_data} WHERE tid = %d', $tid));
09 }
10
11 return $terms[$tid];
12 }
```

我们创建了一个静态数组来保存结果集，这样，如果查询已被执行过了，那么我们就已经获得了该值，这样就可以直接将其返回，而不需要再次查询数据库了。

处理耗费时间的查询

下面是一个例子，如何使用 devel 模块识别缓慢的查询，从来提高站点的性能。假定我们已经编写了一个自定义节点模块 task（任务），而且使用 hook_load()来向节点对象添加关于任务的附加信息。表的模式如下：

```
1 CREATE TABLE task (
2 nid int,
3 vid int,
4 percent_done int,
5 PRIMARY KEY (nid,vid),
6 KEY nid (nid)
7 );
```

在运行了 devel.module 和查看了查询日志以后，我们注意到对前面这个表进行的查询拖累了站点性能！注意超过 5 毫秒的查询，就被默认为缓慢的（导航到“管理➤站点配置➤Devel 设置”，你可以修改这个值）。

毫秒 函数 查询
27.16 task_load SELECT * FROM task WHERE vid = 3

那么，为什么这个查询这么耗费时间呢？如果它是一个使用多表关联的复杂查询，那么我们将考虑使用

更好的方式来组织数据，但是在这里它是一个非常简单 的查询。首先，我们使用 SQL 的 EXPLAIN 语法，来查看数据库是怎么解释这个查询的。当我们在一个 SELECT 语句前面添加一个关键字 EXPLAIN 时，数据库将返回这个查询执行计划的相关信息：

EXPLAIN SELECT * FROM task WHERE vid = 3

MySQL 给出了下面的报告：

Id select_type table type possible_keys key key_len ref rows Extra
1 SIMPLE task system NULL NULL NULL NULL 1

这里最重要的一列就是 key 列，它现在为 NULL。这告诉我们，MySQL 在取回结果集时没有使用任何主键、唯一键、或者索引键；它需要逐行进行查找。所以提升这个查询的速度的最好方式，就是向 vid 列添加一个唯一键。

ALTER TABLE task ADD UNIQUE (vid);

关于 MySQL 的 EXPLAIN 的更多信息，可参看 http://dev.mysql.com/doc/refman/5.0/en/explain.html。

Devel 模块的其它用途

Devel 模块还有一些其它的方便函数，它们能帮你提升开发效率。

例如，你可以实时的切换当前查看 Drupal 页面的用户。这在技术支持和调试其它角色时，会非常有用。为了切换到另一个用户，导航到 URL http://example.com/?q=devel/switch/$uid，其中$uid 是你想切换到的用户的 ID。还有一种方式是，启用 “Switch users”（切换用户）区块，它提供了一组链接，能够用来实现相同功能。

Devel 模块提供了一个名为 Execute PHP（执行 PHP）的附加区块，可以用来方便的输入和执行简短的代码片断（这也是为什么在实际的站点中，一定要禁用 devel 模块的一个原因！）。

你可以使用 dsm()， dvm()，dpr()，和 dvr()函数来输出调试消息，这些消息对于其它用户是不可见的。

· dpm()向页面的消息区域输出一个简单变量（比如，一个字符串或者一个整数）。可把它理解为“调试输出消息”（“debug print message”）的意思。

· dvm()向页面的消息区域输出一个 var_dump()。对于复杂的变量比如数组或者对象，可以为其使用这个函数。可把它理解为“调试变量消息”（“debug variable message”）的意思。

· dpr()使用一个特殊的递归函数(dprint_r())，在页面顶部输出一个复杂的变量（比如一个数组或者对象），这里输出结果的格式也是很漂亮的。

· dvr()在页面顶部输出一个格式良好的 var_dump()。

这些函数的输出，对于那些没有“访问 devel 信息”权限的用户，是不可见的，这在实时调试时非常方便。

下面给出了一个例子：

```
1 dpr(node_load(5)); // Display the data structure of node 5.
2 dvr($user); // Display the $user variable.
```

模块构建器模块

http://drupal.org/project/module_builder 中，有一个很好的模块，它能帮你方便构建出模块的骨架。它

向你询问你想要创建的钩子，并帮你创建它们，而且还带有示例代码。接着，你就可以在它的基础上开始工作了！

应用剖析和调试

下面的 PHP 调试器和集成开发环境（IDE），提供了一些强大工具，能够帮你快速找到 Drupal 的瓶颈所在；它们也能够帮你找出模块中的低效算法：

- Zend Studio IDE: http://www.zend.com/
- Komodo IDE: http://www.activestate.com/Products/komodo_ide/
- Eclipse IDE: http://www.eclipse.org/
- Xdebug PHP Extension: http://www.xdebug.org/

在下面的图中，我们使用了 Zend Studio（拥有最漂亮的图形输出，这存在争议）的截图，而其它的 IDE 也能够生成类似的输出。图 21-22 显示了一个图形输出，它是使用应用剖析器 （application profiler）追踪 Drupal 请求得到的。结果显示了每个文件里面的函数的运行所占用的相对时间。在这个情况下，Drupal 看起来在 includes/bootstrap.inc 中花费了将近一半的时间。

图 21-22.在 Zend IDE 中，一个 Drupal 请求的时间饼图

在图 21-23 和 21-24 中，我们向下钻取，来查看哪些函数在一个请求期间耗费了相对较多的处理器时间。这一特性，能方便的帮你判定哪些地方需要多花一点工夫进行优化。

图 21-23.在 Zend IDE 中，一个 Drupal 请求的调用踪迹

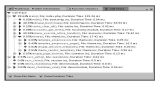

图 21-24.在 Zend IDE 中，一个 Drupal 请求的函数统计

实时调试是一个 PHP 的特性，而不是 Drupal 的特性，但是它值得在这里讨论一下，如果你的笔记本上装了一个实时调试器，那么别人就会把你当作 Drupal 高手。

使用一个 PHP 调试器，可以让你在运行时暂停 PHP 代码的执行（比如，设置一个断点），并逐步的检查发生了什么。熟悉一个 PHP 调试器，是你最应该 掌握的一门技术之一。一帧一帧的追踪代码执行，就像电影中的慢动作一样，这是调试的最好方式，它能帮你慢慢的熟悉 Drupal 这样一个庞然大物。

Drupal 初学者的成人礼，就是端上一杯茶水，启动调试器，花一些时间逐步的追踪一个标准的 Drupal 请求，这能够帮你获得 Drupal 工作原理的第一手资料。

总结

读完本章后，你应该能够：

- 按照 Drupal 的编码规范编写代码。
- 为你的代码添加注释，这样可以使用 API 模块来将你的注释转化为文档。
- 使用 egrep 来搜索 Drupal 的基准代码。
- 使用版本控制下载和更新 Drupal。
- 干净的修改 Drupal 核心。

- 使用统一的 diff 格式生成显示代码变更的补丁。
- 使用别人创建的补丁。
- 使用标签和分支来维护一个第 3 方模块。
- 使用 devel.module 来提高你的开发效率。
- 通过开发者的最佳实践来识别 Drupal 编码高手。

使用编码器模块

在 http://drupal.org/project/coder，你将找到一个宝贝，它能帮你节省不少的时间，减轻你的烦恼。这就是编码器模块：一个用来评估其它模块代码的模块。

下载最新的版本，将它放到 sites/all/modules/contrib/下，接着在“管理➤站点构建➤模块”启用它，这和其它模块的安装一样。

为了使用编码器模块评估你的模块，点击你站点导航中的新链接“代码评估”，选择你想要的评估类型，以及选择你想要评估的模块或主题。或者导航到“管理➤站点构建➤模块”，在模块列表中，这个模块提供了一个方便的“代码评估”链接供你使用。

提示 如果你想快速的适应 Drupal 的编码习惯，那么编码器模块应该是你的必选模块之一。

你甚至可以更进一步，那就是使用编码器模块自带的 coder_format.php 脚本。这个脚本事实上可以修正你的代码格式错误。下面是如何使用 coder_format.php 来检查我们在第 2 章编写的注释模块：

```
$ cd sites/all/modules
$ php contrib/coder/scripts/coder_format/coder_format.php \
custom/annotate/annotate.module
```

脚本就地修改了 annotate.module 文件，并将原始文件保存为了 annotate.module.coder.orig。为了看到脚本都做了什么，可以使用 diff：

```
$ diff custom/annotate/annotate.module
custom/annotate/annotate.module.coder.orig
```