本 章将给你一个Drupal的概述。系统如何工作的每个细节将在后面的章节逐一提供。在此，将大略展示一下 drupal 的运行技术堆栈、构成 drupal 的 文件层级以及诸如节点（nodes）、钩子（hooks）、区块（blocks）、主题（themes）等 drupal 使用的一些术语。

Drupal 是什么？

Drupal 用来创建 Web 站点。它是高度模块化、强调合作的开源 web 内容管理框架。它可扩展、遵从标准并力求使用清洁的代码占用更小的空间。

Drupal 靠一些由激活的内建模块和第三方模块提供的基本核心功能和一些附加功能运行。Drupal 被设计成可定制化，但是定制是由覆写核心或附加模块的方式来完成，而不是修改核心代码。Drupal 的设计成功地分离了内容的管理和内容的表现。

Drupal 可以用来构建 internat 门户、个人网站、部门的或公司的站点；电子商务站点、资源目录；在线报纸、社交网站；相册、intranet、虚拟现实及其它你能猜想出来的 web 站点。有 专门的团队通过应对威胁并发布安全更新来努力保证 Drupal 的安全。非盈利的 Drupal Association 组织通过提高 drupal.org 基础设施、组织会议和活动来提供 Drupal 支持。有一个繁荣的网上用户社区，包括设计者、管理 员、开发者，辛苦工作，不断地提供软件，请浏览 http://drupal.org 和 http://groups.drupal.org。

技术堆栈

Drupal 设计的目标是既可用于运行于便宜的主机的小站点，也要适用于大尺寸的复杂的分布式站点。前一点目标意味着使用大量流行的技术，后一个目标意味着细致小心的、严格紧密的编码。Drupal 的技术堆栈如下图 1-1：

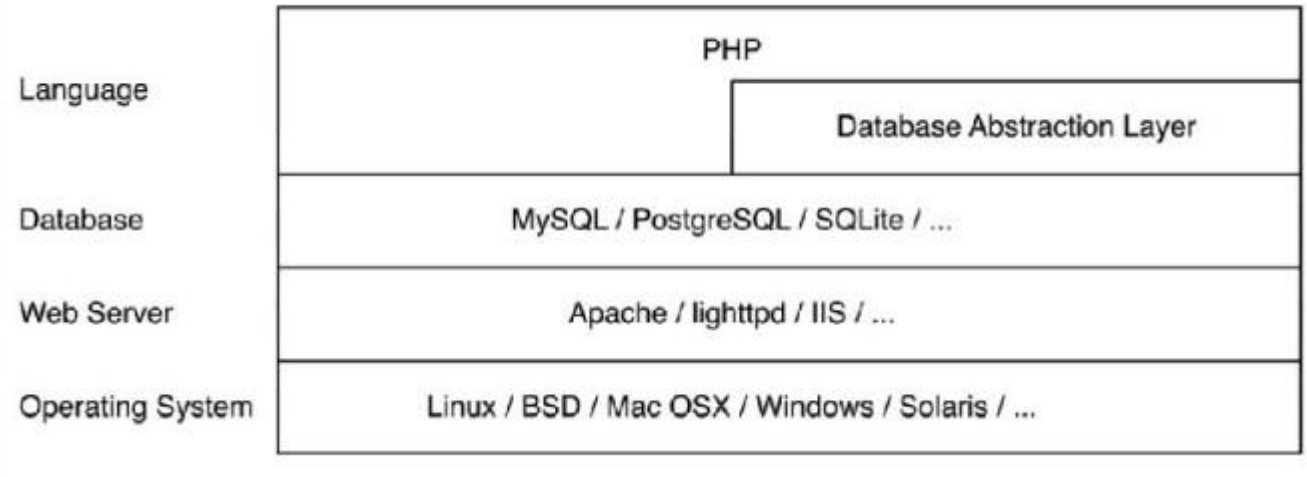

figure 1-1 Drupal's technology stack

操作系统处于堆栈的底层，Drupal 不用太关心它们，Drupal 成功地运行于任何支持 PHP 的操作系统上。

被 Drupal 使用的 web 服务器是 apache，但是其它也可以使用，包括微软的 IIS，只是因为 Drupal 在历史上长期使用 apache，Drupal 利用.htaccess 文件来安全安装 Drupal。Clean URL -- 没有 问号、'&'等其它奇怪字符的 URL 是靠一组 apache 的 mod_rewrite 规则来提供。这一点非常重要，因为当从其它内容管理系统或者从 静态文件移植过来时，需要不更改原有 URL，并且按照 Tim Berners-Lee (http://www.w3.org/Provider/Style/URI)的说法，不变的 URI 很酷。其它 web 服务器上的 Clean URL 依照这些服务器得 url 重写特性来生效。

Drupal 技术堆栈的通过数据库层（database） 有一个轻量级的数据库抽象层，Drupal7 完全重写了数据库层，此数据库接口提供一个基于 PHP 数据对象（PDO）的 API 来允许 Drupal 去支持任何支持 PHP 的数据库，包括常用的 MySql、PostgreSQL 等，Drupal7 现在也支持 SQLite。
Drupal 是用 PHP 写的，严格追寻编码标准(http://drupal.org/nodes/318)并且通过了彻底的开源检查。对于 Drupal7 简单的 PHP 学习曲线意味着可以很容易成为贡献者，他们只要着手开始，并且检查流程保证最终产品能轻易访问而不牺牲质量，初学者可以从社区获得帮助来提高他们的技能。Drupal 要求 PHP 是 5.2 版本。

核心

一个轻量级的框架组成了 Drupal 的核心（core），这就是你从 drupal.org 下载的那个，核心负责提供一些基本的功能，用来支持系统的其它部分。
核心包括允许在接收到一个请求时 Drupal 自启动的代码、drupal 频繁使用的公共函数库、以及支持诸如用户管理、分类、模板等功能的模块。如图 1-2

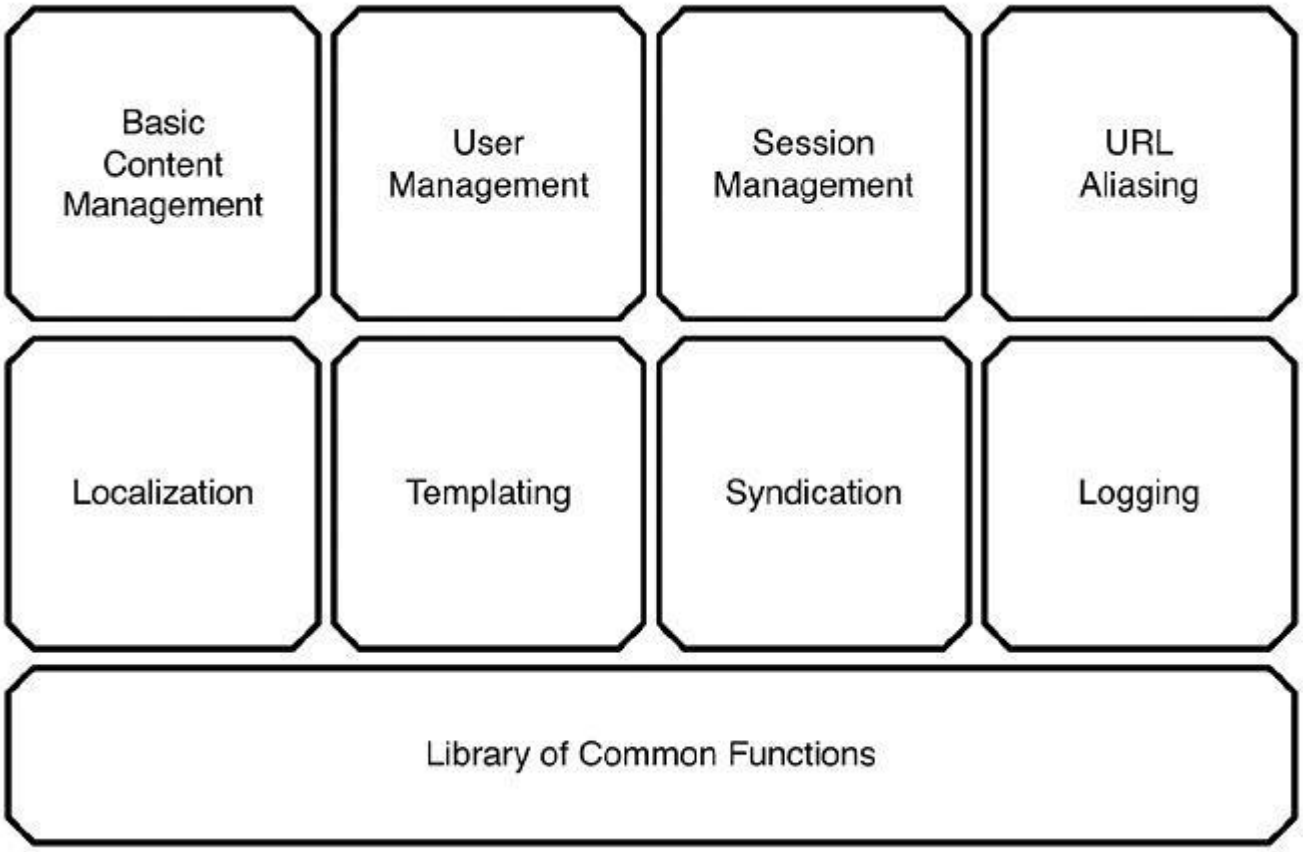

figure 1-2 An overview of the Drupal core (not all core functionality is shown)

核心还包含了建立适于多数站点的区块的功能、包含了 feed 聚合、blog、投票、论坛。

管理接口

Drupal 内的管理接口紧密地与 Drupal 的其他部分整合到了一起，所有的管理功能都可以通过管理工具条上的菜单访问到，这个工具条当你以管理员登陆后显示在窗口的顶部。

模块

Drupal 是个真正的模块化框架，功能被包含在***模块***中，可以激活或者停用，通过激活存在的模块、安装 drupal 社区成员写的模块或者自己新写的模块的方式将特性添加到 Drupal 站点。按照这种方式，Drupal 可以尽量按需使用模块来运行于精干的环境，还能达到需要的特性，需要加什么模块就加什么模块，如下图展示：

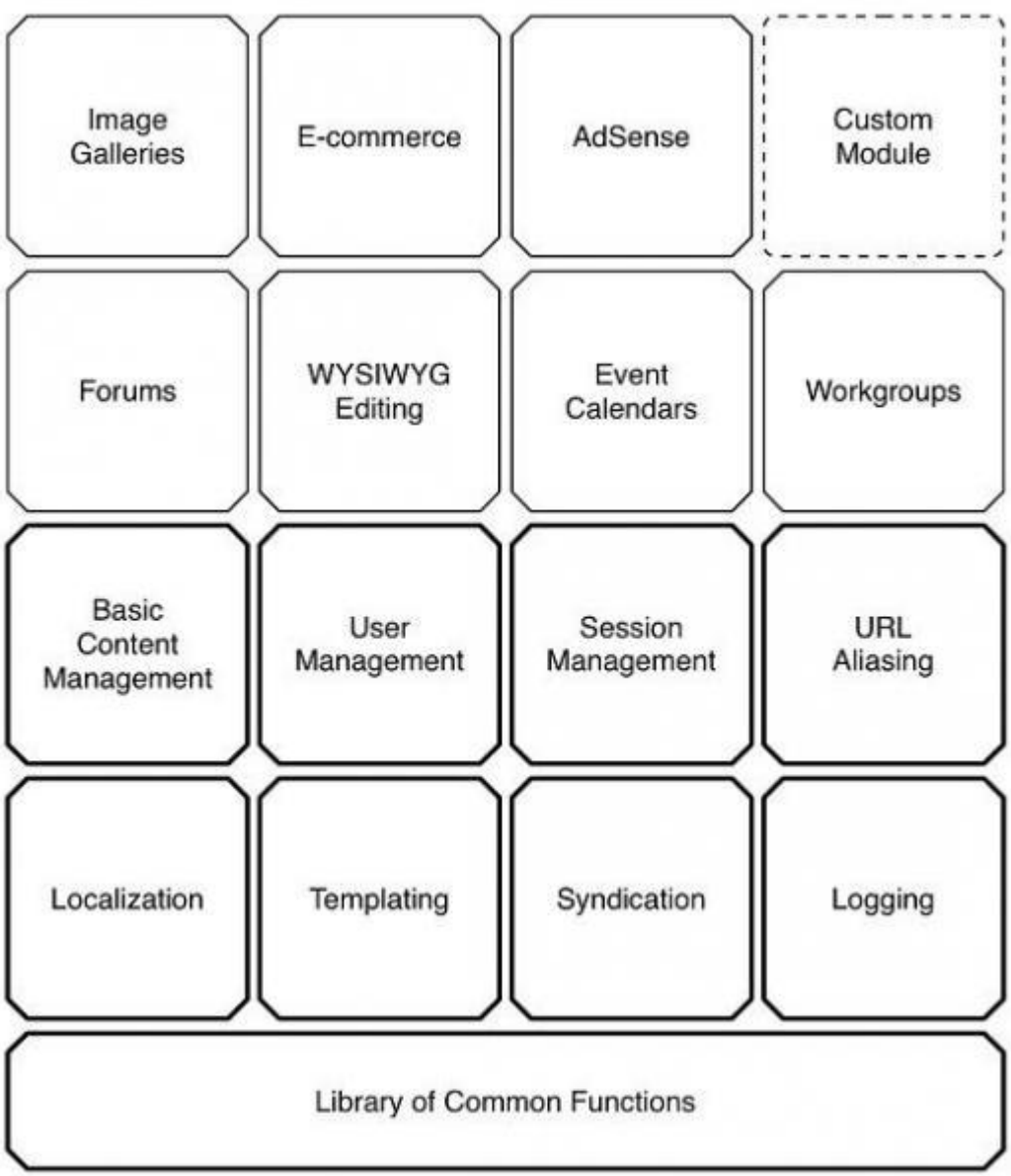

figure 1-3 Enabling additional module s gives more functionality.

模块通过添加新的内容类型来扩展 Drupal，比如食谱、博客帖子、文件，及 e-mail 通知、端对端发布、聚合器。Drupal 颠倒了控制设计模式，在恰当的时机框架将调用模块化函数，为模块做它们的某些东西的时机叫钩子。

钩子

钩子被认为是个 Drupal 内部事件，它们也被叫做回调，但是因为它们习惯上由命名函数构成，而不是通过一个监听者注册，所以不是真正的回调。钩子允许模块“钩进”drupal 接下来的发生的事情。

既 定一个用户登录进 Drupal 站点，当用户登录时，Drupal 击发 hook_name_login。这意味着任意一个按照惯例命名的函数会被调用，命名 方法是模块名+钩子名，例如 comment 模

块中是 comment_user_login()，locale 模块中叫 locale_user_login()，node 模块中叫 node_user_login()，并且任何其它相似名称的函数也会被调用，如果你写了一个 自定义的模块叫 spammy.module 并且包含一个函数叫 spammy_user_login()给用户发一个 E-mail，你的函数也会被调用，并 且不高兴的用户在每次登录后都受到一个主动发来的邮件。
大多数利用 Drupal 核心功能的通用方法是通过在模块中实现钩子完成。

tip 更多的关于 Drupal 钩子细节，请看在线文档 http://api.drupal.org/api/7 下的“Module System（Drupal hooks）”一节。

主题

当 建立一个 web 页面并发送到一个浏览器，实际上要考虑两个方面：组织恰当数据并将为 web 做标记数据。在 Drupal 中，主题层负责创建浏览器准备接收的 HTML（或 JSON、XML 等）。Drupal 使用 PHP Template 作为主要的模板引擎，或者你可以另选 Easy Template System（ETS）。大多数开发者在组织新的 Drupal 主题时倾向于标准的模板引擎。一个重要的东西需要记住，Drupal 鼓励内容和表象分离。
Drupal 允许几种方式去定制和覆写你的 web 站点的面貌和感觉。最简单的方式是使用层叠样式表 CSS 去覆写 Drupal 内建的类和 id。然而，你希望超越这个方法 去定制实际的 HTML 输出，你能发现这很容易去做，Drupal 模板文件由标准的 HTML 和 PHP 组成，加之，每个 Drupal 页面动态例如一个面包屑导 航，能简单地通过声明一个恰当名称的函数来覆写，然后 Drupal 将使用这个函数替代原来的函数去建立页面的这个部分。

节点

Drupal 的内容类型源于单一的被称为节点（node）的基础类型，不论它是一个博客条目、一个菜谱甚或一个计划任务，最根本的数据结构是相同的。天才背后的方式是 它的可扩展性。模块开发者可以附加诸如评级、评论、文件附件、地理位置信息给节点而不用担心它是博客条目、菜谱什么的，然后网站管理员依照内容类型混合、 匹配各功能，例如，管理员可以选择为博客激活评论功能但是关闭菜谱的评论功能，只为计划任务激活文件上传功能。节点还包含一个基本的行为特性，由其他内容类型继承。任何一个节点能放置到站点的首页、发布或不发布或可查找，并且由于这是统一的结构，管理接口能够在一个节点工作时提供一批编辑屏幕。

字段

Drupal 的内容由一些独立的字段构成，比如节点的标题就是一个字段，就想节点的主体一样。在 Drupal 中你可以用字段去构建任何的你可以想象出来的内容类型，例 如，一个事件。如果你想象一个事件，它典型地包含一个标题、一个描述（或叫主体）、一个开始日期、一个开始时间、一段时期、一个位置、还可能有一个注册该 事件的连接，每一个此类元素都表现为一个字段。在 Drupal 中，我们有能力用字段去创建内容类型--可以使用模块编程的方

式或者使用管理接口创建一个新 的内容类型，通过用户接口给它分配字段，方法可以二选一。使用 Field API 可以极其简单地使用少量编程来创建从简单到复杂的内容类型。

区块

一 个区块是在你的站点模板特殊位置能被激活或禁用的信息。例如，一个区块可以显示你的站点上活跃用户数量，你可以区块去包含一个站点热门内容的链接或未发生 事件的列表。区块典型地被安排在模板的边栏、头部或页脚。区块能被设置成在某些特定类型节点上显示，只能在首页上或依照其它准则。

通常，区块用来 表现定制给当前用户的信息，例如，用户区块只包含一个到当前用户有权访问的管理界面链接，诸如“My account”页面。区域（Regions）就是定义在站点主题中的一块区域，在那里区块可以表现出来，例如 header、footer、或左右边栏， 区域中的区块布局和可视化是通过 web 基础的管理接口来管理。

文件布局

理解默认的 Drupal 安装实例的目录结构将教你几个重要的实践经验，诸如下载的模块和主题应该怎么放置以及怎样去弄个不同的 Drupal 安装 profile。一个默认的 drupal 安装有下图 1-4 中的结构。

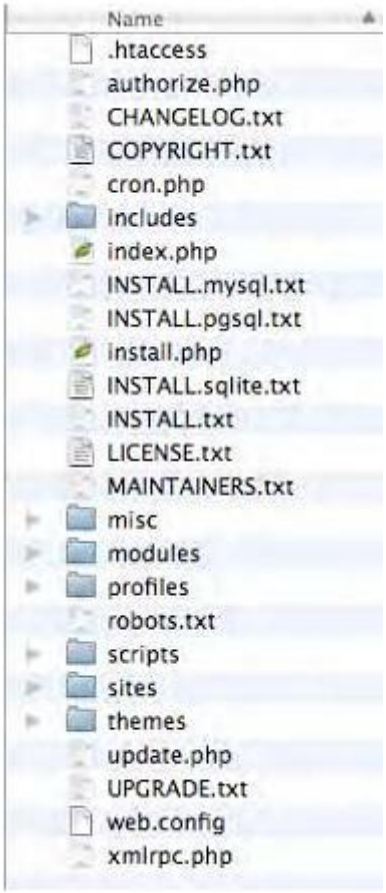

figure 1-4 The default folder structure of a Drupal installation

文件夹中每一项的细节如下：
include 目录包含 Drupal 使用的库和公共函数
misc 目录存储 Drupal 安装时使用的 javascript 和杂项的图标、图片。
modules 目录包含核心模块，里面的每个模块都有一个自己的目录。别往里加任何东西，或任何其他目录，除了 profiles 和 sites，你应该将扩展模块加到 sites 目录。
profiles 目录中包含了不同的站点安装 profile。
scripts 目录包含检查语法、清理代码、命令行运行 Drupal、处理 cron 特别情况、运行测试包（D7）等所需要的脚本。这个目录不在 drupal 请求生命周期内使用，它们是 shell 及 perl 实用脚本。
sites 目录（图 1-5）包含了你在设置表单中对设置、主题、模块的修改。当你从贡献模块仓库或写一个模块时，你应当将它们加入到-sites/all /modules 中，这是保存所有你的 Drupal 变更的单独目录。在 sites 目录中有一个名叫 default 的子目录，保存一个你的 Drupal 站点 默认的配置文件 default.settings.php，Drupal 安装器将在你提供的信息基础上修改这些初始设置并且为你的站点生成一个 settings.php 文件，默认的目录是典型地拷贝并重命名为你的站点的 URL，你最终的设置文件将是 sites/www.example.com/settings.php。
sites/default /files 目录包含 Drupal 默认基本安装信息，它需要用来存储那些上传到你的站点以备后用的任何文件。一些例子是用来存放用户 logo，激活用户头 像或其他上传的媒体文件。这些子目录需要 drupal 运行的服务器的可读可写的权限，如果有权限，Drupal 安装器将创建子目录。此外 sites/default/files，一个 sites/default/private 目录可能建立用来存储比较敏感的和不应展示的文件，除非有适当 的证明。你可以通过浏览 Configuration>FileSystem 并进入你想保存私有文件的方式来创建私有文件目录，文件路径就是标题是 Private 的文本域的文件系统路径。
themes 目录包含模板引擎和 Drupal 默认的主题，你下载的和创立的附加主题不应该放在这里，将它们放到 sites/all/themes 中。
cron.php 用来执行周期性的任务，例如修剪数据库表和计算统计等。
index.php 是服务请求的主入口点。
install.php 是 Drupal 安装器的主入口点。
update.php 在 Drupal 版本更新后更新数据库关系模式。
xmlrpc.php 接收 XML-RPC 请求，不打算使用 XML-RPC 时可以从部署中删除。
robots.txt 默认的机器人排除规则。
authorize.php 运行授权文件操作的管理脚本，例如从 drupal.org 下载并安装一个新的主题或模块。

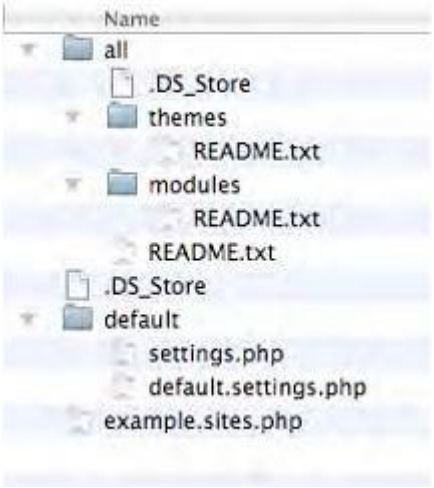

figure 1-5 The sites folder can store all yo ur Drupal modifications.
其它没有列出来的文件是文档。

服务一个请求

对于 Drupal 接收到一个请求时发生了什么有个概念上的框架非常有帮助，这节提供一个快速排练。如果你要自己跟踪，那么使用一个好的 debugger，在 index.php 处开始，它接收大多数它的请求。这个序列对于显示简单网页来说比较复杂，但它具有普遍的灵活性。

web 服务器规则

Drupal 运行于 web 服务器之上，通常是 Apache。如果 web 服务器遵循 Drupal 的。.htaccess 文件，有些 PHP 这是初始设置的，并且 URL 是被检查的。差不多所有对 Drupal 的调用都是通过 index.php，例如一个到 http://example.com/foo/bar 的调用经历下列流程：
1、Drupal 的.htaccess 文件的 mod_rewrite 规则查看送入的 URL 并从路径中分离出基本 URL，本例中，路径是 foo/bar。
2、路径被指派为 URL 请求操作符 q 的参数、
3、结果 URL 是 http://example.com/index.php?q=foo/bar。
4、Drupal 像内部路径一样处理 foo/bar 并且交由 index.php 开始处理。
如同这个处理结果，Drupal 精确地一同样方式处理 http://example.com/index.php?q=foo/bar 和 http://example.com/foo/bar，因为这两种情况内部路径相同，这可以使 Drupal 不使用怪异的 URL 字符，而使用 Clean URL。
在 作为替代的其它 web 服务器，如微软的 IIS，clean URL 可以使用 ISAPI 模块像 ISAPI 重写那样取得。IIS 7 和后面版本支持直接重写。如果你的站点运行于 IIS7 或更高版本，你应当校验用于激活 clean URL 的 web.config 文件，并且使它不能被窥探到，
像.install, .module, .test, .theme, .profile, .info 和.inc 文件一样。

启动过程

Drupal 每个请求的自启动过程都经历一系列启动阶段，这些阶段在 bootstrap.inc 中定义，如下表描述：

阶段	目标
Configuration	设置贯穿启动过程的公用变量。
Database	初始化数据库系统且注册自动装入函数。
Variables	装入系统变量和所有已激活的启动模块。
Session	初始化会话处理。
Page Header	启动 hook_boot()，初始化锁系统并且发送默认的 HTTP 头。
Language	初始化所有定义的语言类型。
Full	最终阶段：Drupal 现在完全装入，此阶段校验和修正输入数据。

处理请求

回调函数的工作是必须处理和累积要完成请求所必需的数据。例如，如果接收到一个内容请求 http://example.com/q=node/3，URL 被映射到 node.module 的函数 node_page_vies()，进一步处理是从数据库检索出这个 node 数据并将它放进一个数据结构，然后就是主题化它的时候了。

主题化数据

主题化牵扯翻译检索出的数据、操作或新建 HTML（或 XML 或其它输出格式）。Drupal 将使用管理员选择的主题给 web 页面以恰当的视觉和感觉，结果输出被送到 web 浏览器或其它 HTTP 客户端。

总结

读完这章，你能大概了解 Drupal 怎样工作并且对 Drupal 处理一个请求时发生什么有个大概的了解。一些创造 web 页面的组成部分将在下面章节详细叙述。

模块是构成 Drupal 的基础建筑构件(积木),模块也是为由现成的 Drupal 版本,又称 Drupal 核心功能提供扩展的机制，我经常对那些不熟悉 Drupal 的人说明：Drupal 的模块就像乐高积木。他们遵循事先定义的方针，完 美地融合在一起，并且，利用模块的组合，你可以创建丰富的和复杂的解决方案。
通常有两类 Drupal 模块，核心的和贡献的。核心模块是 Drupal 搭载的诸如 poll、menu、taxonomy、search、feed、聚合和论坛。贡献模块是其它所有由社区创建的，从 drupal 核心功能扩展的那些模块。有大量的贡献模块可以从 http://drupal.org/project/modules 下载，功能跨越一切，从简单地显示一个日期到复杂的电子商务前端。
在本章，我将展示如何毫无基础地创建一个自定义模块，你建筑一个模块时，你将学到模块应该遵循的标准。我需要一个显示目标，让我们聚焦真实世界问题：**评注**。当我们查看 Drupal 站点的页面时，你可能想写一个关于此页面的注解，我们可以使用 Drupal 的评论特性来完

成这些，但是评论通常对任何浏览用户都是可见的，另一方面，我只想让**注解**只有它的所有者可见。

创建文件

我们首先要做的东西就是为模块选择一个名字。“annotate”看起来不错，简短且能说明问题。下一步，我需要个地方放置模块，贡献、和自定义模块存放于/sites/all/modules 目录中，每一个模块都有一个自己的目录，和模块名相同。
注意：Drupal 核心模块存在/modules 目录中，以保护你的贡献和自定义模块在更新的时候不被覆盖和删除。

你 可能希望创建一个/sites/all/modules/custom 目录来存放你的任一个白手起家做的模块，使某些人看着更明白，理解那些模块是从 drupal.org 下载的贡献模块，那些是你自己为这个站点创建的模块。下一步我需要在/sites/all/modules/custom 目录内创建 一个 annotate 目录，来此存放与 annotate 模块关联的所有文件。
为新模块创建的第一个文件时 annotate.info 文件，drupal7 中的每一个模块都应当包含一个.info 文件，名称必须匹配模块名称。对于 annotate 模块，为了使 Drupal7 识别模块所必需的基本信息是：

```
name = Annotate
description = "Allows users to annotate nodes."
package = Pro Drupal Development
core = 7.x
files[] = annotate.module
files[] = annotate.install
files[] = annotate.admin.inc
configure = admin/config/content/annotate/settings
```

文 件的这个结构适用于所有 Drupal7 的模块，name 项用来在模块配置页面显示名字；description 项用来在模块配置页面显示模块的描 述;package 项定义模块分配到那些包或组中，在模块配置页面，模块按照包来分组显示；core 字段定义模块为哪个 Drupal 版本所写；php 项定 义这个模块需要的 php 版本；files 项是个数组，包含一些与此模块关联的文件的名字，在此，与 annotate 模块关联的文件是 annotate.module 和 annotate.install 文件。
我们能在前面列表中指派一些可选的值，这有个例子，一个模块需要 PHP5.2、安装时依赖于 forum 和 taxonomy 模块才能工作。

```
name = Forum confusion
description = Randomly reassigns replies to different discussion threads.
core = 7.x
dependencies[] = forum
dependencies[] = taxonomy
files[] = forumconfusion.module
files[] = forumconfusion.install
```

```
package = "Evil Bob's Forum BonusPak"
php = 5.2
```

现在我们准备去建立一个真正的模块，在你的子目录 sites/all/modules/custom/annotate 中创建一个 annotate.module 文件，输入 PHP 标签和一个 CVS 占位符，紧接着是一个注释：

```
<?php
/**
* @file
* Lets users add private annotations to nodes
*
* Adds a text field when a node is display
* so that authenticated users may make notes.
*/
```

首先，注意注释风格，我们用/**开始，随后的每一行，我们用一个缩进一个空格的星号开始，在注释结束时用*/。@file 符号表示接下来的行是文件做什么的描述，one-line 描述常常想 api.module 那样，Drupal 的自动文档抽取并格式它，能找到这个文件时什么的，当你在 drupal.org 浏览 http://api.drupal.org 时，在这你能找到 drupal 提供的每个 api 的细节文档。我建议你用点时间去看 drupal.org 的这部分，它对于我们这些模块开发和修改者来说是无价的资源。

后面是个空行，我们增加一个比较长的描述针对那些审查我们代码的程序员（毫无疑问）。注意我们故意不使用关闭标签?>，这在 PHP 中是可选的，如果包括，文件中尾随的空格可能引发问题（请看 http://drupal.org/coding-standards#phptags）。

注意：为什么我们对结构化如此挑剔，因为有时几百个开发者围绕着一个项目一起工作，如果每个人都按照统一的标准工作将节省时间。Drupal 编码风格细节能在 Developing for Drupal Handbook 的 Coding standard（(http://drupal.org/coding-standards）节找到。
我们工作的下一步就是定义一些东西像我们能用一个基于 web 的表单去选择那些节点可以注释。这个用两部完成，首先，我们将定义一个路径是我们能够访问我们的设置，然后，我们将建立设置表单，为制造个路径，我们需要实现一个钩子，就是 hook_menu。

实现一个钩子

Drupal 建立在钩子（有时叫回调）系统之上，在执行的过程中，Drupal 询问模块是否要做什么事情，例如，当一个节点被从数据库载入并在显示到页面之 前，Drupal 检查所有已经激活的模块，看看它们是否有实现的 hook_node_load()函数，如果有，Drupal 在节点被渲染到页面之前执行 模块的钩子。我们来看看在 annotate 模块中它是如何气作用的。
我们的第一个钩子是实现 hook_menu()函数，我们用这个函数在站点的 管理菜单增加两个项目。我们在主 admin/config 菜单增加一个项目“annotate”，并在“annotate”项下增加一个子菜单项，叫 “settings”，当我们点击它时将打开 annotate 配置页面。我们菜单项是由键和值组成的数组，值描述 Drupal 当路径被请求时应该做什么。 我们将在第 4 章

“Drupal 菜单系统”中讲述细节。我们将 hook_menu 命名为“annotate_menu” -- 用模块名替换“hook”，这种规则对所有模块来说都是一致 -- 总是用你的模块名字替换单词“hook”。

将这些加到我们的模块

```
/**
 * Implementation of hook_menu().
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Node annotation',
    'description' => 'Adjust node annotation options.',
    'position' => 'right',
    'weight' => -5,
    'page callback' => 'system_admin_menu_block_page',
    'access arguments' => array('administer site configuration'),
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
  );

  $items['admin/config/annotate/settings'] = array(
    'title' => 'Annotation Settings',
    'description' => 'Change how annotations behave.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_setings'),
    'access arguments' => array('administer site configration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
  );
  return $items;
}
```

此时不要过多担心细节，这段代码说：“当用户进入 http://example.com/?q=admin/config/annotate/settings 时， 调用函数 drupal_get_form()，并且传给它表单 ID annotate_admin_settings，寻找在文件 annotate.admin.inc 中描述的函数，只有有访问权限的用户才能看见这个菜单 项” 。当 Drupal 完成对所有模块的菜单项询问后，请求路径时将调用一个恰当的函数生成一个菜单项。

注意：如果你对钩子机制驱动函数感兴趣，请看 module_invoke_all()函数，定义在 includes/module.inc(http://api.drupal.org/api/function/module_invoke_all/7)中。你应该看到现在问什么我们叫它 hook_menu()或菜单钩子（the menu hook）。

Tip: Drupal 的钩子的变造几乎涉及各个方面，一个 Drupal 提供的钩子的完整列表和使用方法能够在 Drupal API 文档站点（http://api.drupal.org/api/group/hooks/7）找到。

增加模块特定设置

Drupal 有多种节点类型（在用户接口中叫内容类型），如 articles 和 basic pages。我们想限定注释只能在默写节点类型使用，要做到这一点，我需要创建一个页面，在这个页面我们将告诉模块那些内容类型才是我们希望可注释的，在 这个页面，我们将展现一组复选框，每个现存内容类型对应一个，这将让最终用户决定哪些内用类型可以注释（见图 2-1）。这样的页面时管理页面，组成它的代 码只有在需要时才载入和解析，因此，我们将代码放到一个单独的文件，而不放在每个 web 请求都要载入和运行的 annotate.module 文件中，随 后，我们告诉 Drupal 去在文件 annotate.admin.inc 中寻找设置表单，我们建立 sites/all/modules/annotate /annotate.admin.inc 并加入如下代码：

```
<?php

/**
* @file
* Administration page callbacks for the annotate modules
*/

/**
* Form builder. Configure annotations
*
* @ingroup forms
* @see system_settings_form()
*/
function annote_admin_settings() {
  // Get an array of node types with internal names as key adn
  // "friendly names" as values. E.g.,
  // array('page' => 'Basic Page', 'article' => 'Articles')

  $types = node_type_get_types();
  foreach ($types as $node_type) {
    $options[$node_type->type] = $node_type->name;
  }
  $form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('User may annotate these content types'),
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('A text field will be available on these content types to
make user-specific notes.'),
  );
```

```
  $form['#submit'][] = 'annotate_admin_settings_submit';
  return system_settings_form($form);
}
```

表单在 Drupal 中表现为一个嵌套的树状结构 -- 就是一个数组的数组。这个结构告诉 Drupal 表单渲染引擎是怎样的一个表单将被展现，为了可读性，我们将数组的每个元素放到一行上，每个表单属性都用井 字符号（#）标记，并且用作数组的键。开始我们声明表单元素类型为 checkboxes，意味着复选框将被使用一个键化数组建立起来，我们已经 在 $options 变量中得到了键化数组。
我们将选项设置为函数 node_type_get_types()的输出，它返回一个对象数组，输出看起来如下：

```
[article] => stdClass Object (
  [type] => article
  [name] => Article
  [base] => node_content
  [description] => Use articles for time-sensitive content like news,
press releasesor blog posts.
  [help] =>
  [has_title] => 1
  [title_label] => Title
  [has_body] => 1
  [body_label] => Body
  [custom] => 1
  [modified] => 1
  [locked] => 0
  [orig_type] => article
)
```

对象数组的键是节点类型的 Drupal 内部名称，友好的名称（那些显示给用户看的）包含在对象的那么属性中。
Drupal 表 单 API 需要#options 要设置成键=>值对数组，foreach 循环使用类型属性去创建键的部分，用名字属性去为我命名为$options 的 新数组创建值的部分，Drupal 将为 Basic page 和 article 以及其它任何你站点的内容类型生成复选框。
通过定#title 属性的值我们为这个表单元素提供标题。

注意:任何返回的文本将显示给用户(例如表单字段的#title 和#description 属性)。Drupal 提供了字符串翻译功能函数,通过为所有字符串运行翻译函数,使你的模块本地化十分简单，我们没有为菜单项使用翻译函数，是因为菜单项自动翻译。

下一个指令#default_value，我们为表单元素定义默认值，因为复选框多个表单元素，#default_value 应该是个数组。
#default_value 的值值得讨论一下：

variable_get('annotate_node_types', array('page'))

Drupal 允许程序员使用一对特殊的函数存取任何的值：variable_get()和 variable_set()。值存储在数据库表 variables 中并且可 以随时请求使用。因为这些值是在每次请求时都从数据库中抽取，用这种方式去存储巨量的数据不是一个好主意。注意我们传递给 variable_get()的 是一个描述我们的值的键（然后我们能取回）和一个默认值，在此，这个值是一个数组，就是允许注释的那个节点类型。我们是将允许注释 Basic page 内容类型作为默认。

Tip: 使用 system_settings_form()时，表单元素的名字（在此是 annotate_node_types）必须匹配 variable_get()中的键名。

我们提供了一个描述来告诉管理员一点关于应该进入这个字段的信息，我将在第 11 章论述表单的细节。
下 步我将增加代码来处理增加和删除内容类型的 annotation 字段。如果一个站点管理员选中一个内容类型，我将增加一个 annotation 字段到这个 内容类型；如果管理员决定从一个内容类型中删除 annotation 字段，我将删除这个字段，我使用 Drupal Field API 去定义此字段并将此字段关联到一个内容类型。Field API 处理所有设置一个字段的相关活动，包括在 Drupal 数据库建立一个表来存储内容作者提交的值、建立用来收集作者输入的信息的表单项、使字段和内容 类型关联、在节点编辑表单和节点页面的字段显示。我将在第 8 章论述 Field API 细节。
我要做的第一件事是建立提交例程，它将在管理员提交表 单时调用。在这个例程中，模块将检查内容类型的复选框是否被选中，如果没选中，就验证这个内容类型有没有关联的 annotation 字段，有的话，表示站 点管理员想从内容类型中删除这个字段，并且从数据库中删除存在的 annotations，如果复选框选中，模块查看这个内容类型上是否存在字段，如果没 有，则模块在此内容类型上增加一个 annotation 字段。

```
/**
 * Process annotation settings submission
 */
function annotate_admin_settings_submit($form, $form_state) {
  // Loop through each of content type checkboxes shown on the form
  foreach ($form_state['values']['annotate+node_type'] as $key => $value)  {
    // If the check box for a content type is unchecked, loop to see whether
    // this content type has the annotation field attached to it using the
    // field_info_instance function. If it does then we need to remove the
    // annotation field as the adaministrator has unchecked the box
    if (!$value) {
      $instance = field_info_instance('node', 'annotation', $key);
      if (!empty($instance)) {
        field_delete_instance($instance);
        watchdog('Annotation', 'Delete annotation field from content type: %key',
array('%key' => $key));
      }
    } else {
```

```
      // If the check box for a content type is checked, look to see whether
      // the field is associated with that content type. If not then ad the
      // annotation field to the content type.
      $instance = field_info_instance('node', 'annotation', $key);
      if (empty($instance)) {
        $instance = array(
          'field_name' => 'annotation',
          'entity_type' => 'node',
          'bundle' => $key,
          'label' => t('Annotation'),
          'widget_type' => 'text_textarea_with_summay',
          'settings' => array('display_summary' => TRUE),
          'display' => array(
            'default' => array(
              'type' => 'text_default',
            ),
            'teaser' => array(
              'type' => 'text_summary_or_trimmed',
            ),
          ),
        );
        $instance = field_create_instance($instance);
        watchdog('Annotation', 'Added annotation field t content type: %key',
array('%key' => $key));
      }
    }
  } // End foreach loop
}
```

下一步就是为模块建立.install 文件，此文件 包含一个或多个在模块安装或卸载时调用的函数。多我们这个模块而言，在它安装时我们想建立 annotation 字段，它能通过管理员与内容类型关联，如果 模块被卸载，我们想从所有内容类型中删除 annotation 字段并且从数据库中删除这个字段和它的内容。为做到这些，我们在模块目录中建一个名叫 annotate.install 的文件。
第一个调用的函数是 hook_install()，我们命名它为 annotate_install() -- 追随 Drupal 钩子函数命名约定标准，用模块的名字代替单词“hook”，在 hook_install 函数中，我们将用 Field API 检查字段是否存在，如果不存在，我们就创建 annotation 字段。

```
<?php
/**
 * Implements hook_install()
 */

function annotate_install() {
  // Check to see if annotation field exists
```

```
  $field = field_info_field('annotation');

  // if the annotation field does not exist then create it
  if (empty($field)) {
    $field = array(
      'field_name' => 'annotation',
      'type' => 'text_with_summary',
      'entity_type' => array('node'),
      'translatable' => TRUE,
    );
    $field = field_create_field($field);
  }
}
```

下一步是建立一个卸载函数，实现 hook_uninstall。我们建立一个函数 annotate_uninstall，使用 watchdog 去记录一个信息告诉管理员此模块将被卸载。 我将使用 node_get_types() API 函数去获得一个站点所有存在的内容类型的列表，通过对此类型列表的循环，检查在内容类型上是否存在 annotation 字段，如果有，就删除它，最 后，我们删除 annotation 字段本身。

```
/**
 * Implements hook_uninstall()
 */
function annotate_uninstall() {
  watchdog('Annotate Module', 'Uninstalling module and deleting field');

  $types = node_type_get_types();
  foreach ($types as $type) {
    annotate_delete_annotaion($type);
  }

  $field = field_info_field('annotation');
  if ($field) {
    field_delete_field('annotation');
  }
}

function annotate_delete_annotation($type) {
  $instance = field_info_instance('node', 'annotation', $type->type);
  if ($instance) {
    field_delete_instance($instance);
  }
}
```

处理过程的最后一步是更新.module 文件，使之 包含一个检查，看浏览节点的人是否是作者，如果不是，我们就对齐隐藏用户的注释。我采取一个简单的途径，使用 hook_node_load()，这个钩子 在节点载入是调用，在 hook_node_load()函数中，我将检查，看浏览节点的人是否是作者，如果不是作者，我通过不设置的方式来对其隐藏注释。

```
/**
 * Implements hook_node_load()
 */

function annotate_node_load($nodes, $types) {
  global $user;

  // Check to see if the person viewing the node is the author, If not then
  // hide the annotation
  foreach ($nodes as $node) {
    if ($user->uid != $node->uid) {
      unset($node->annotation);
    }
  }
}
```

保存你创建的文件（.info .install .module .admin.inc），在管理菜单进入模块页面，你的模块应该显示在标题为 Pro Drupal Development 的组中（如果没有，在你的 annotate.info 和 annotate.module 文件中双击 syntax；确保他们在 sites/all/modules/custom 目录中），激活你的模块。
现在 annotate 模块激活啦，导航到 admin/config/annotate/settings 你应该看到 annotate.module 的配置页面（图 2-1）。

Figure 2-1. The configuration form for annotate.module is generated for us.

只用了几行代码，我们现在有了功能齐全的模块配置页面，它将自动地保存和记住我们的设置，这让你感觉到 Drupal 手段的力量。
让我们先测试一下为所有内容类型激活 annotations，选中所有复选框，点击保存按钮，下一步建一个 basic page 节点，滚动到下方就能看见 Annotation 字段啦（图 2-2）。

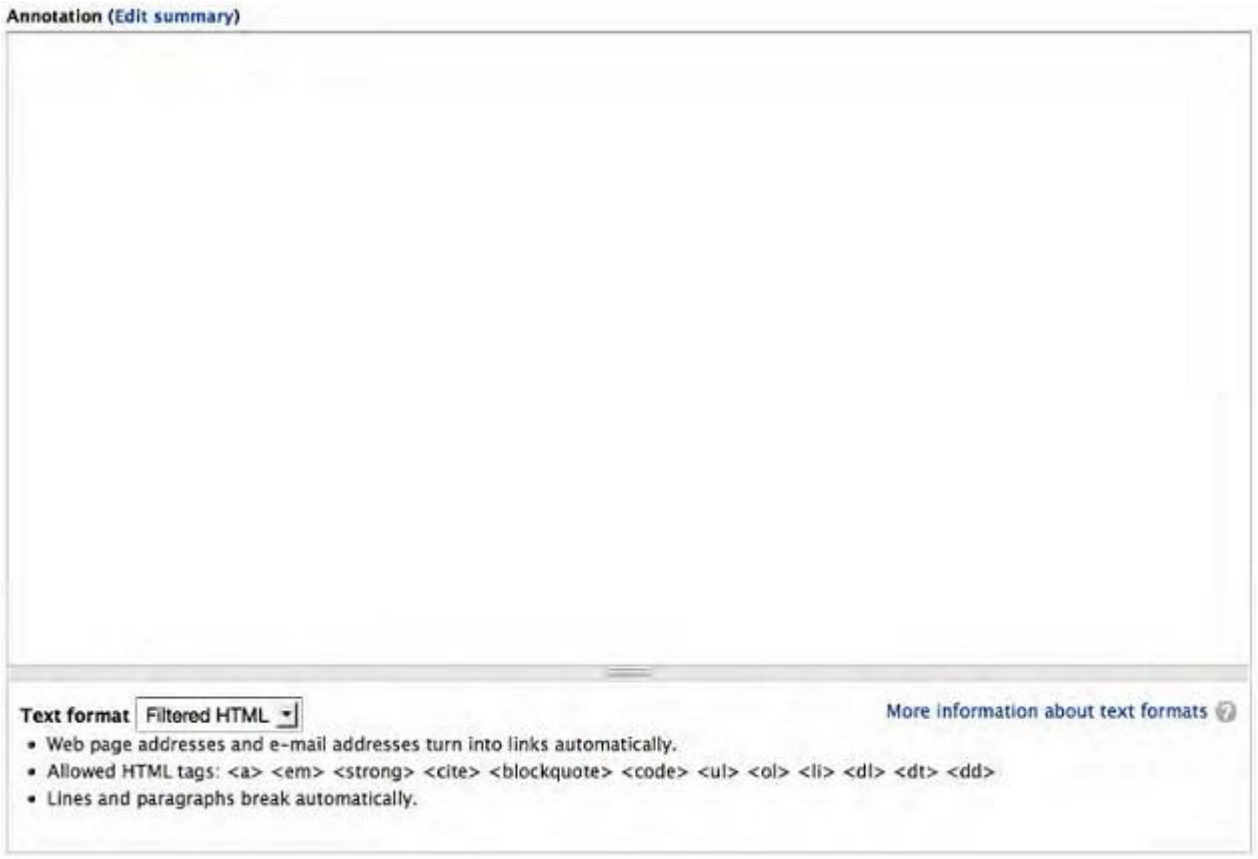

Figure 2-2. The annotation form as it appears on a Drupal web page

建一个新的节点，输入标题、内容、注释，保存后你应该看到图下图 2-3 相似的结果。

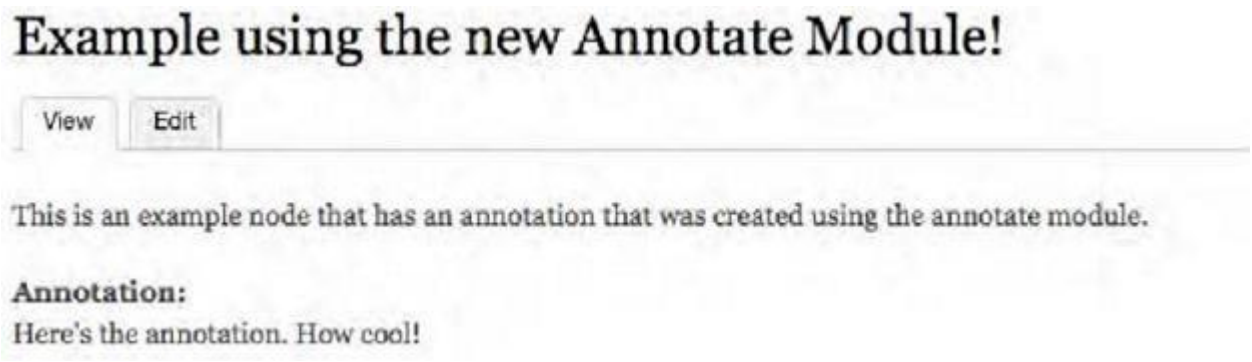

Figure 2-3. A node that has an annotation

自 此，我们都没有哪怕是含蓄地进行任何数据库操作，你可能惊奇 Drupal 在哪存储和提取 annotation 字段的值。Field API 做了幕后工作：建立了一个表以控制字段的值，在节点保存和载入时保存和提取这个值。当你调用 FIled API 函数 field_create_field()时，它使用标准的命名规则在数据库中创建一个表 field_data_<fieldname>。对于我们的 annotations 字段，表的名称为 field_data_annotations，我们将在第 4 章论述更多的细节。

创建你自己的管理节

Drupal 有许多管理设置的类别，如内容管理和用户管理，它们呈现在配置页面上，如果你需要一个自己的类别，你可以轻松地创建一个自己的类别，在这个例子中，我们建立一个新的叫“Node annotation”的类别。为此，我们使用模块的菜单钩子定义一个新的类别：

```
/**
 * Implementation of hook_menu()
 */
function annotate_menu() {
  $items['admin/config/annotate'] = array(
    'title' => 'Node annotation',
    'description' => 'Adjust node annotation options.',
    'position' => 'right',
    'weight' => -5,
    'page callback' => 'system_admin_menu_block_page',
    'access arguments' => array('administer site configuration'),
    'file' => 'system.admin.inc',
    'file path' => drupal_get_path('module', 'system'),
  );
  $items['admin/config/annotate/settings'] = array(
    'title' => 'Annotation Settings',
    'description' => 'Change how annotations behave.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('annotate_admin_setings'),
    'access arguments' => array('administer site configration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'annotate.admin.inc',
  );
  return $items;
}
```

我们的模块的类别在配置页面上看起来如同下图 2-4：

Figure 2-4. The link to the annotation module setti ngs now appears as a separate category.

如 果你修改了菜单钩子的代码，你必须清楚菜单缓存，为此，你可以 truncating 数据库表 cache_menu，或者点击模块 devel.module 提供的 Rebuild Menu 链接或者在卑职页面上点击 Clear cache data 按钮在它的页面上点击 Performance 链接。

Tip：开发模块（http://drupal.org/project/devel）是为 Drupal 开发者定制的模块，让你快速访问许多开发功能，如清除缓存、查看变量、跟踪查询等等，是每个开发者的必备。

我们用两步来建造我们的类别页，首先，我们增加一个描述类别头部的菜单项，这个菜单项有一个唯一的路径（admin/config/annotate），我们声明它应该放置在右列权重为-5，因为这个位置就在“Web Services”列别之上，如图 2-3。
第 二步告诉 Drupal 在“Node annotation”类别中建立实际的到 annotation 设置的链接，我们将原始菜单项的路径设置到admin/config/annotate /settings,Drupal 重建菜单树时，他检视路径以去建立父项和子项之间的关系并决定它们，因为 admin/config/annotate /settings 是 admin/config/annotate 的子项，它就同样显示。
Drupal 只载入需要的文件来完成一个请求，节省了内 存的使用，因为我们的页面回调指向的函数不在我们模块空间之内（system_admin_menu_block_page()在 system.module 内），我需要告诉 Drupal 载入 modules/system/system.admin.inc 而不是去试图载入 sites/all/modules/custom/annotate/system.admin.inc。我们告诉 Drupal 从 system 模块中获 取路径并将结果加到我们菜单项的文件路径键上。
当然，这只是个例子，在实际中，你要建个类别要有足够好的原因，没有那个管理员愿意面对众多的类别。

呈现设置表单

在 annotate 模块，我们使管理员有了选择哪个节点类型可以支持注释的能力，让我们研究一下它是如何工作的。
当 一个管理员想改变 annotate 模块设置时，我们想显示一个表单给管理员，使他可以选择我们提供的选项，我们设置这个页面的回调指向 drupal_get_form()函数，设置页面参数为一个包含于 annotate_admin_settings 的数组，这意味着当我们浏览 http://example.com/?q=admin/config/annotate/settings 时，对 drupal_get_form('annotate_admin_settings')的调用将被执行，它实质上告诉 Drupal 去建立有函数 annotate_admin_settings()定义的表单。
让我们看一下定义表单的函数，它为每个节点类型定义了一个复选框，现在增加两个选项，此函数在文件 sites/all/modules/custom/annotate/annotate.admin.inc 中：

```
/**
 * Form builder. Configure annotations
 *
 * @ingroup forms
 * @see system_settings_form()
 */
function annote_admin_settings() {
  // Get an array of node types with internal names as key adn
```

```
  // "friendly names" as values. E.g.,
  // array('page' => 'Basic Page', 'article' => 'Articles')

  $types = node_type_get_types();
  $foreach ($types as $node_type) {
    $options[$node_type->type] = $node_type->name;
  }
  $form['annotate_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('User may annotate these content types'),
    '#options' => $options,
    '#default_value' => variable_get('annotate_node_types', array('page')),
    '#description' => t('A text field will be available on these content types to
make user-specific notes.'),
  );

  $form['annotate_deletion'] = array(
    '#type' => 'radios',
    '#title' => t('Annotations will be deleted'),
    '#description' => t('Select a method for deleting annotations.'),
    '#options' => array(
      t('Never'),
      t('Randomly'),
      t('After 30 days')
    ),
    '#default_value' => variable_get('annotate_deletion', 0) //default to Never
  );
  $form['annotate_limit_per_node'] = array(
    '#type' => 'textfield',
    '#title' => t('Annotations per node'),
    '#description' => t('Enter the maximum number of annotations allowed per node
(0 for no limit).'),
    '#default_value' => variable_get('annotate_limit_per_node', 1),
    '#size' => 3
  );

  $form['#submit'][] = 'annotate_admin_settings_submit';
  return system_settings_form($form);
}
```

我们增加一个单选按钮去选择何时删除注释以及一个文 本域去输入一个节点允许输入几个注释，（增强功能的实现留给你做练习），不是管理我们自己的表单的处理，我们调用 system_settings_form()去让系统模块增加一些按钮到表单并校验和提交表单。图 2-5 展示了选项表单的样子：

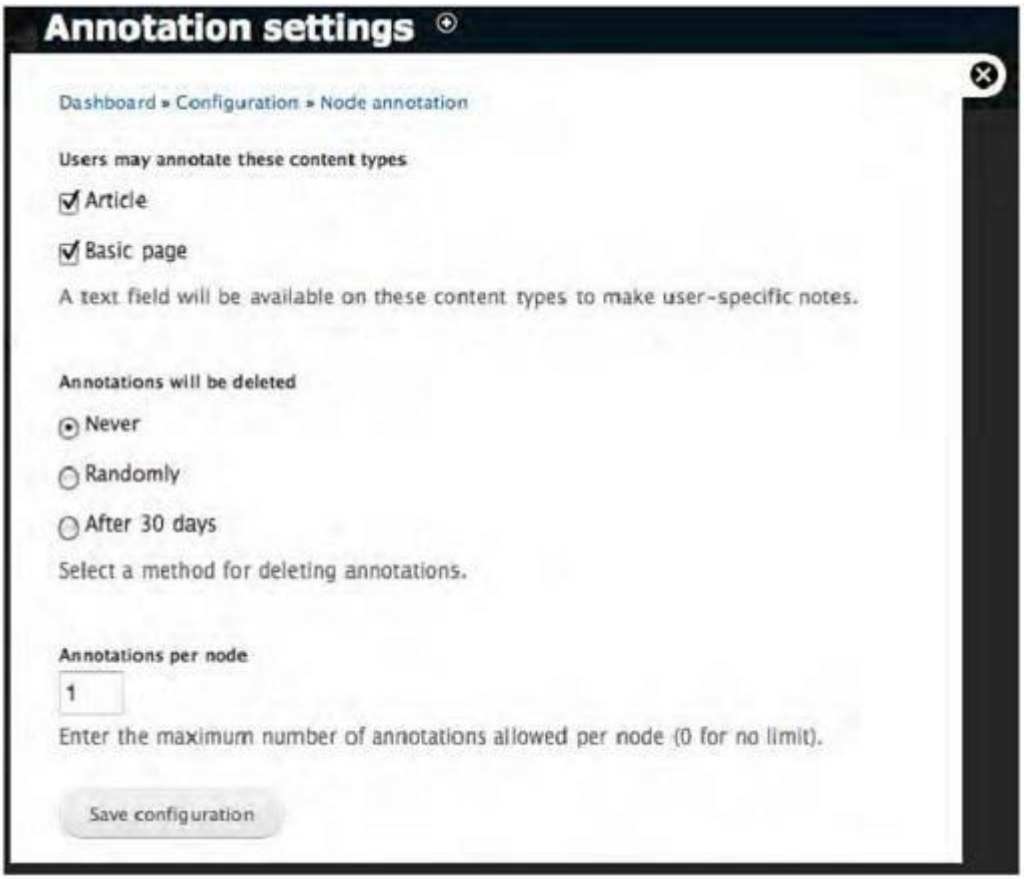

Figure 2-5. Enhanced options form using check box, radio button, and text field options

校验用户提交的设置

如 果 system_settings_form()为我们关心表单值的保存，我们怎么能检查“Annotations per node”域输入的是真正的数字？我们只需要增加一个检查看看这个值是否是数字的函数 （annotate_admin_settings_validate($form, $form_state）到 sites/all/modules/custom/annotate/annotate.admin.inc 文件中并使用它 在找到任何问题时设置一个错误。

```
/**
 * Valitate annotation settings submission.
 */
function annotate_admin_settings_validate($form, &$form_state) {
  $limit = $form_state['value']['annotate_limit_per_node'];
  if (!is_numric($limit)) {
    form_set_error('annotate_limit_per_node', t('Please enter number.'));
  }
}
```

现在,但Drupal处理表单时,它将回调到annotate_admin_settings_validate()以便校验,如果你判定输入了一个坏值,错误发生我们在出错字段的附近设置一个错误,映射到屏幕上就是一个警告信息,并且出错字段高亮显示。
Drupal怎么知道调用我们的函数?我们命名方式特殊,使用表单定义函数的名字(annotate_admin_settings)加上_validate。Drupal如何决定哪个表单校验函数被调用的全部解释请看第11章。

存储设置

前面的例子中,改变设置并按保存按钮是好使的,本节在下面描述这发生了什么。

使用Drupal variables表

首先,我们看下“Annotations_per_node”字段,它的#default_value键设成variable_get('annotate_limit_per_node', 1)。
Drupal 数据库中有个variables表,可以使用variable_set($key, $value)来保存键/值对,可以用variable_get($key, $default)抽取,那么我们意思是说:“将‘Annotations per node’字段的默认值设置为保存于variables数据库表变量annotate_limit_per_node的值,如果找不到,就设置为1。

警告:为了这些设置在variables表中存取不发生命名空间冲突,允许你给你的表单元素和变量建起个相同的名字(上例中是annotate_limit_per_node)。用你的模块名加上描述性的名字给你的表单元素/变量建起名,表单元素和变量硬都用这个名字。

“Annotations will be deleted”字段有点复杂,因为它是一个单选按钮字段,#option是下面这样:

```
'#options' => array(
  t('Never'),
  t('Randomly'),
  t('After 30 days')
)
```

PHP处理没有键的数组时,它实际暗中使用了数字键,数组实际已经如下了:

```
'#options' => array(
  [0] => t('Never'),
  [1] => t('Randomly'),
  [2] => t('After 30 days')
)
```

当我们为此字段设置默认值是,我们使用下面语句,这意味着,使用数组的0号项目就是t('Never')。

```
'#defual_value' => variable_get('annotate_deletion', 0) //default to
Never
```

用 variable_get()抽取存储的值

当你的模块抽取已经存储的设置值是，就要用到 variable_get()函数；

```
// Get stored setting of maximum number of annotations per node.
$max = variable_get('annotate_limit_per_node', 1);
```

注意这里为 variable_get()使用了一个默认值，应对没有存储的值的情况（如管理员还从未浏览过这个设置页面）。

以后的步骤

以 后我们要在开源社区共享这个模块,自然地,应该建一个 README.txt 文件,放到 annotate 目录，和 annotate.info，annotate.module，annotate.admin.inc，and annotate.install 等文件在一起，README.txt 通常包含模块开发者、怎样安装模块等信息，版权信息不需要，因为所有上传到 drupal.org 的模块都遵循 GPL，drupal.org 的打包脚本能自动生成一个 README.txt 文件，再以后你能将其上传到 drupal.org 的贡献仓库，并建立一个 project 页面来保持跟踪社区其他人的反馈。

小结

读完本章后，你能完成下列任务

+ 零基础创建一个 Drupal 模块
+ 理解怎样钩进 Drupal 代码技巧
+ 存储和抽取模块设置参数
+ 使用 Drupal 表单 API 建立和处理简单的表单
+ 在 Drupal 的主管理页面内建立新的管理类别
+ 使用复选框、文本域、单选按钮定义一个供管理员选择的表单
+ 校验设置数据并呈现错误信息
+ 理解 Drupal 怎样利用内建的稳固的变量系统存储和抽取设置值

当 用 Drupal 干活时有一个通常的考虑那就是当一个特定的事件产生时有什么事情发生，例如，站点管理员可能想在某条消息发布时接收一个 e-mail 或者一 个一个用户在评论里包含了某些单词时应该被锁定。本章论述怎样钩进 Drupal 事件以在这些事件产生时执行你自己的代码。

理解事件和触发器

Drupal 通过一连串的事件行进就像它自己的那样，这些内部事件是允许模块与 Drupal 处理过程交互的时刻，下表展示一些 Drupal 事件：

Event	type

Creation of a node Node
Deletion of a node Node
Viewing of a node Node
Creation of user account User
Updating of user profile User
Login User
Logout User

Drupal 开发者认为这些内部事件像钩子一样是因为当一个时间发生时 Drupal 允许模块在这一点上钩进执行路径，在前一章你已经见到过一些钩子，典型的开发涉及决定哪些 Drupal 事件是你想做出反应的，就是说，在你的模块中你要实现那个钩子（hook）。
假 设你有个刚刚完成的网站，你是在你的地下室计算机上管理你的网站，一旦站点火了你准备将它卖给大公司而一夜暴富，期间，你在每个用户登录时都想要得到一个 通知，你决定在每个用户登录时计算机嘟嘟叫，因为你的猫睡觉了且发现嘟嘟叫太讨厌，你决定使用一个简单 log 条目模拟一下嘟嘟叫，于是，你快速地写了一 个.info 文件，并将其放到 sites/all/modules/custom/beep/beep.info 中：

```
name = Beep
description = Simulates a system beep.
packeage = Pro Drupal Development
core = 7.x
files[] = beep.module
```

然后，写了 sites/all/modules/custom/beep/beep.module 文件：

```
<?php
/**
 * @file
 * Provide a simulated beep.
 */
function beep_beep() {
  watchdog('beep', 'Beep!');
}
```

将一条消息“Beep!”写到 Drupal 的 log 中 -- 现在就足够了，下一步，是告诉 Drupal 在用户登录那刻嘟嘟叫的时候啦，我们能用实现 hook_user_login()来容易地完成它：

```
/**
 * Implamentation of hook_user_login().
 */
function beep_user_login(&$editor, $account) {
  beep_beep();
}
```

这很简单，怎样在一个新内容被增加时嘟嘟一下呢，只需在我们的模块中实现 hook_node_insert()捕获 insert 操作：

```
/**
 * Implamentation of hook_node_insert().
 */
function beep_node_insert($node) {
  beep_beep();
}
```

那 么我们怎样才能让新增个评论也嘟嘟一下呢，好，我们能实现 hook_comment_insert()并捕获 insert 操作，但是等等，想几分钟，我们实质上一遍一遍地做同样的东西，你难道不会提供个图形用户接口吗？在那里你可以将 beep 分配给你想要的任何钩子和任何操 作类型？这些已经被 Drupal 内建的模块 trigger 做了，它允许你去为特定的事件（event）关联某些动作（action）。在这个代码中，一个 事件呗定义为唯一的 hook-operation 组合，就像“user hook,login operation”或“node hook,insert operation”，当每个这样的操作发生，trigger.module 让你触发一个动作。

为了避免混淆,让我们澄清我们的术语：

+ event: 使用在一般的编程概念中，这个术语通常被理解成从系统的一个组成部分发送到另一组成部分的一条消息。

+ hook: 这个编程技术，用在 Drupal，允许模块“钩进”执行流，可钩进（hookable）对象上的每个操作都有唯一一个钩子(如 hook_node_insert)。

+ trigger: 指的是一个钩子和一个操作的特定组合，这个操作（operation）能关联一个或多个动作（action），例如，动作 beeping 能被关联到 user hook 的 login 操作。

理解动作

一个动作就是 Drupal 做的某些东西，这有些例子：

+ 提升一个节点到首页
+ 将一个节点从未发布状态改到发布状态
+ 删除一个用户
+ 发送一个邮件

每 一个这样的情况都是一个定义清晰的任务。编程者注意到这与前面提到的 PHP 函数类似，例如你能通过调用 includes/mail.inc 中的 drupal_mail()来发送一个 e-mail。动作和函数听起来很相似，因为动作是函数，他们是能使 Drupal 宽松结合事件的函数（一会再说更 多），现在让我们审视 trigger 模块。

Trigger 用户接口

点击 Modules 链接，在模块页面激活 trigger 模块，然后点击 structure 链接，进入 Structure 页面，点击 Trigger 链接你应该能看到类似图 3-1 那样的接口：

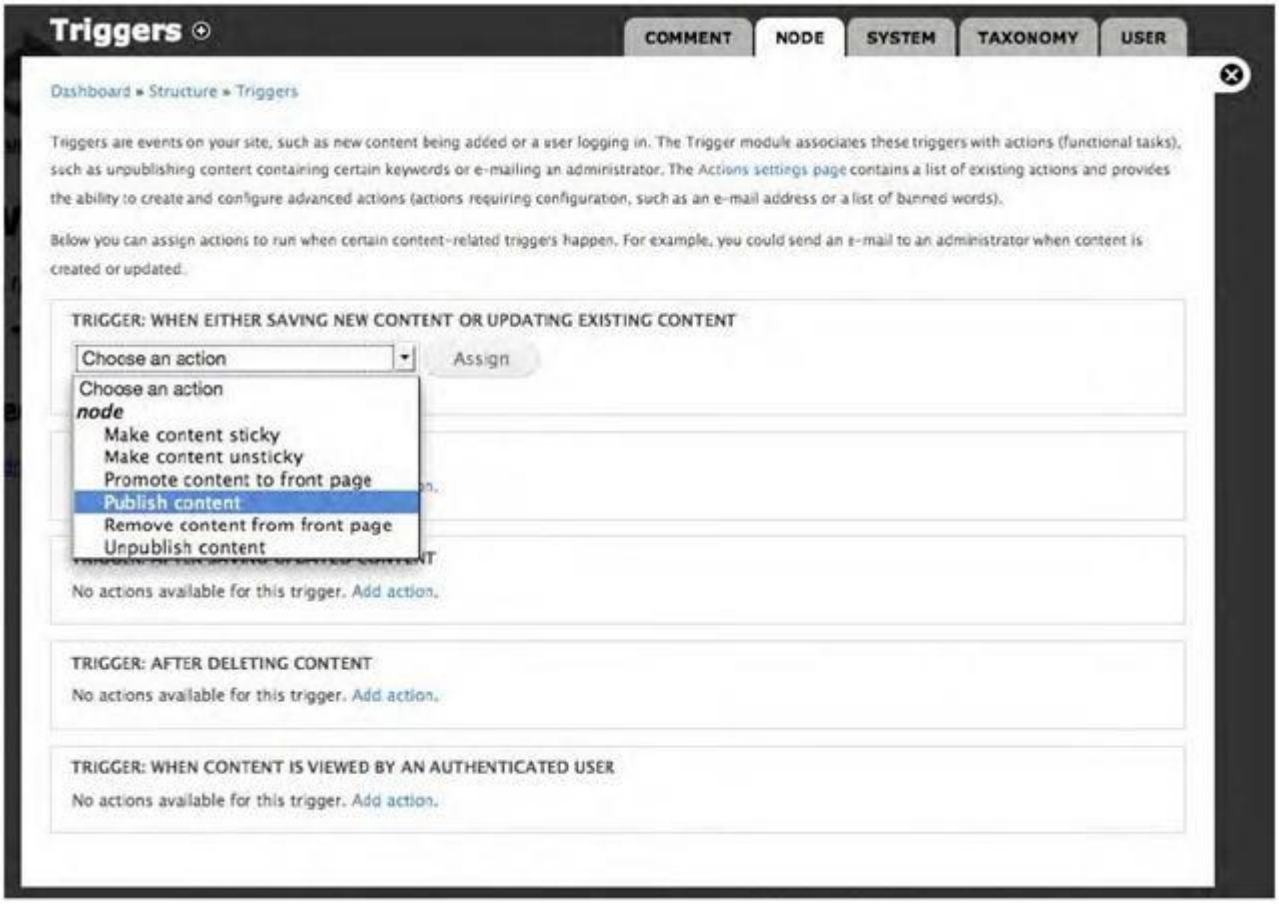

Figure 3-1. The trigger assignment interface

注意横跨顶部标签，这与 drupal 的 hooks 一致，在图 3-1 中我们能看见 node 钩子的操作，它们都有个好看的名字，如 delete 操作显示为“Trigger：After deleting content”这个标签，每个钩子的操作都展示为有能力去关联一个动作，如“Publish Content”，当操作发生时，每个可用动作都显示在“choose an action”的下拉列表里。

注意：不是所有的动作对所有触发器都可用，因为有些动作在特定的内容上没有意义，如，你在触发器”after deleting content”上不能运行“promote post to front page”动作。依赖于你的安装情况，有些触发器可能显示“No actions available for this trigger”。

有些触发器的名字和各自的钩子和操作展示在下表：

Hook	Trigger Name
comment_insert	After saving a new comment
comment_update	After saving an updated comment
comment_delete	After deleting a comment
comment_view	When a comment is being viewed by an authenticated user
cron	When cron run

node_presave When either saving a nwe post or updating an existing post
node_insert After saving a new post
node_update After saving an updated post
node_delete After deleting a post
taxonomy_term_insert After saving a new term to the database
taxonomy_term_update After saving an updated term to the database
taxonomy_term_delete After_deleting a term
user_insert After a user account has been created
user_update After a user's profile has been updated
user_delete After a user has been deleted
user_login After a user has logged in
user_logout After a user has logged out
user_view When a user's profile is being wiewd

你的第一个动作

按顺序该对我们的 beep 函数做点什么才能使它成为发育完全的动作，两步：

1、通知 Drupal 那个模块应支持这个动作

2、建立你的活动函数

第一步通过实现 hook_action_info()来完成，这里展示 beep 模块看起来应该怎样：

```
/**
 * Implementation of hook_action_info().
 */

function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'),
      'configurable' => FALSE,
      'triggers' => array('node_view', 'node_insert', 'node_update',
'node_delete'),
    ),
  );
}
```

函数的名字是 beep_action_info()因为就是钩子的实现，我们用模块名加上钩子名组成，为我们模块的每一个活动返回一个有一个条目的数组，我们只写了一个动作，还有它只有一条，将执行的动作函数名称作为键：beep_beep_action，便于度代码是知道那个函数是一个动作，所有在我们的 beep_beep()函数后面加上_action 以示为是个动作。

让我们看看数组的键的意义：

+ type：这是你写的动作的分类，Drupal 利用这个在触发器关联用户接口的下拉列表里归类动作。可用的有 system node user comment taxonomy。一个好的问题提问是决定你写的动作是什么类型"动作为哪个对象工作？"（如果回答是为大量不同的对象工作，那就把它归

为 system 类型）

+ label：动作的用户友好名字，展示在在触发器关联用户接口的下拉列表里

+ configurable：决定动作是否带任何参数

+ triggers：在这个钩子数组中，每个项目必须是枚举动作支持的操作，Drupal 使用这个信息决定在哪个适当的地方

展示可用动作列表。

我们的动作是如下这样：

```
/**
 * Simulate a beep. A Drupal action.
 */

function beep_beep_action() {
  beep_beep();
}
```

这太简单了，是不？到前面把 beep_user_login()和 beep_node_insert()删除，随后我们将使用触发器和活动去代替直接的钩子实现。

分派动作

现在，让我们点击 Structure 链接在此页点击 Triggers 链接，如果你做的每件事都正确，那么你的活动应该可用啦，就像图 3-2：

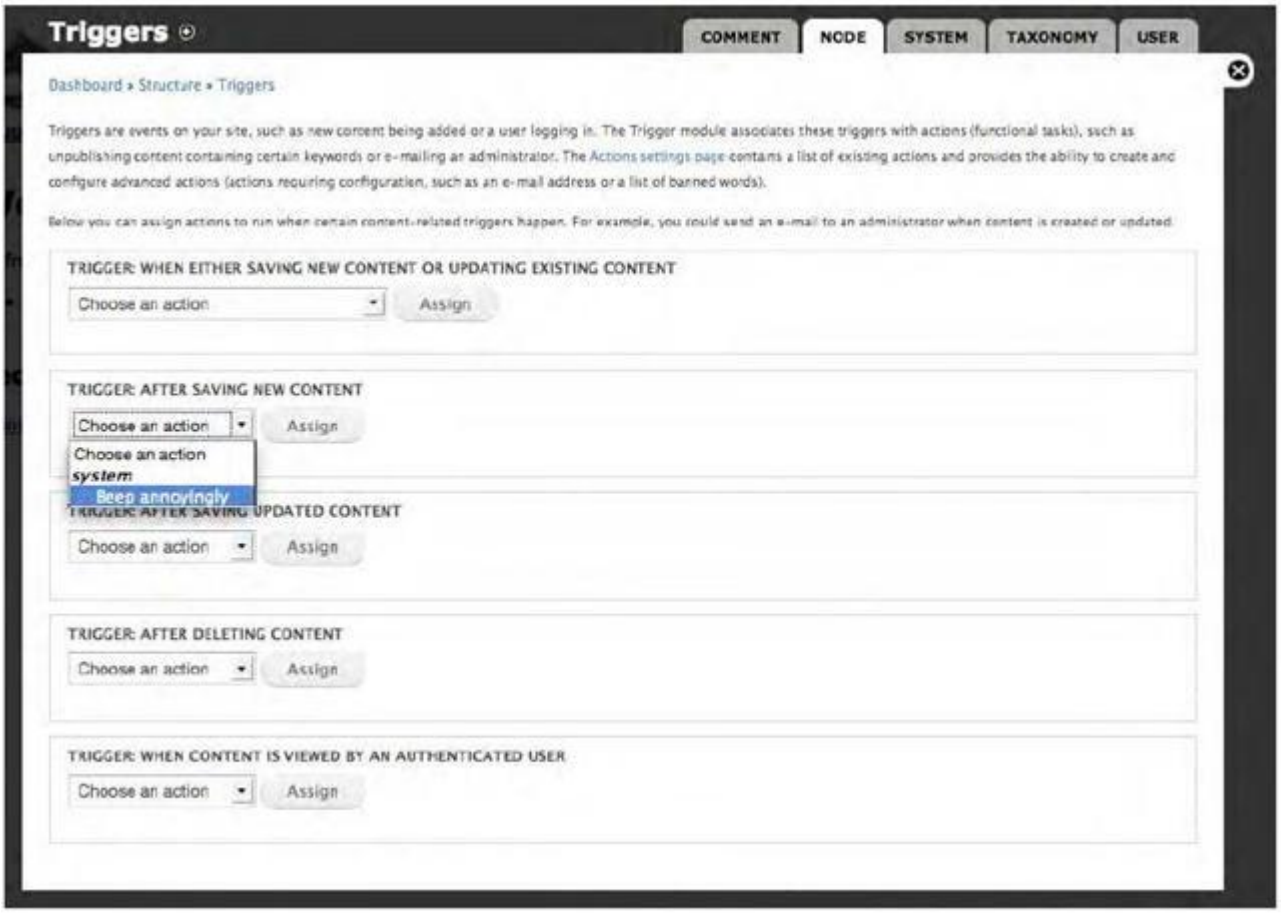

Figure 3-2. The action should be selectable in the triggers user interface.

点击“Assign”按钮、从下拉菜单中选择“Beep annoyingly”、保存新的内容，指派动作和触发器关联，下一步建立一个 Basic Page 内容项保存它，点击 Reports 链接选择 Recent log 项纪录，如果你设置动作和触发器没问题，你应当看见如图 3-3 的结果：

Figure 3-3. The results of our beep action being triggered on node save is an entry in the log file.

改变一个动作支持的触发器

如果你修改了定义动作支持哪个操作的设定值，你应该在用户接口看看可用性的变化，例如，如果你这样改变了 beep_action_info()，那么“Beep”动作就只又在“After deleting a node”操作上可用：

```
/**
 * Implementation of hook_action_info().
 */
```

```
function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'),
      'configurable' => FALSE,
      'triggers' => array('node_delete'),
    ),
  );
}
```

支持所有触发器的动作

如果你不想限制你的动作仅支持确定的触发器或一组触发器，你可以将你的动作声明为支持任何触发器：

```
/**
 * Implementation of hook_action_info().
 */

function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'),
      'configurable' => FALSE,
      'triggers' => array('any'),
    ),
  );
}
```

高级动作

有 实质上分成两类的动作（action）：带参数的动作和不带参数的动作，“Beep”动作不用带任何参数就能工作，这个动作执行时，嘟嘟一次然后就结束 了，但是当一个动作需要大量信息是就要有很多项目，比如，一个“Send e-mail”动作需要知道是谁发的 e-mail，主题和内容都是什么。一个需要配置表单进行一些配置的动作叫“advanced action”（高级动作）或“configurable action”（可配置动作）。

简 单动作不带参数、不需要配置表单、由系统自动变为可用。在模块的 hook_action_info() 的实现中，你将 configurable 键设置成 TRUE 就告诉 Drupal 你写的是高级动作，提供一个表单去配置动作、提供一个选项校验处理程序和一个请求提交处理程序来处理配置表单，高级动作和简单 动作的不同汇总在下表中：

Simple Action **Advanced Action**

parameters
No*
Required
Configuration
form No
Required
Availability Automatic
Must create instance of action using actions administration page
Value of configure key
in
hook_action_info() FALSE
TRUE

**The $object and $context parameters are available if needed.*

让我们创建一个可以多次嘟嘟的动作，用一个配置表单来指定动作的嘟嘟次数。

```
/**
 * Implementation of hook_action_info().
 */

function beep_action_info() {
  return array(
    'beep_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep annoyingly'),
      'configurable' => FALSE,
      'triggers' => array('node_view', 'node_insert', 'node_update',
'node_delete'),
    ),
    'beep_multiple_beep_action' => array(
      'type' => 'system',
      'label' => t('Beep multiple times'),
      'configurable' => TRUE,
      'triggers' => array('node_view', 'node_insert', 'node_update',
'node_delete'),
    ),
  );
}
```

如果我们实现的正确，在 Administer->Site configuration->Action，果然能在下拉列表框中可

以选择一个高级动作，如图 3-4：

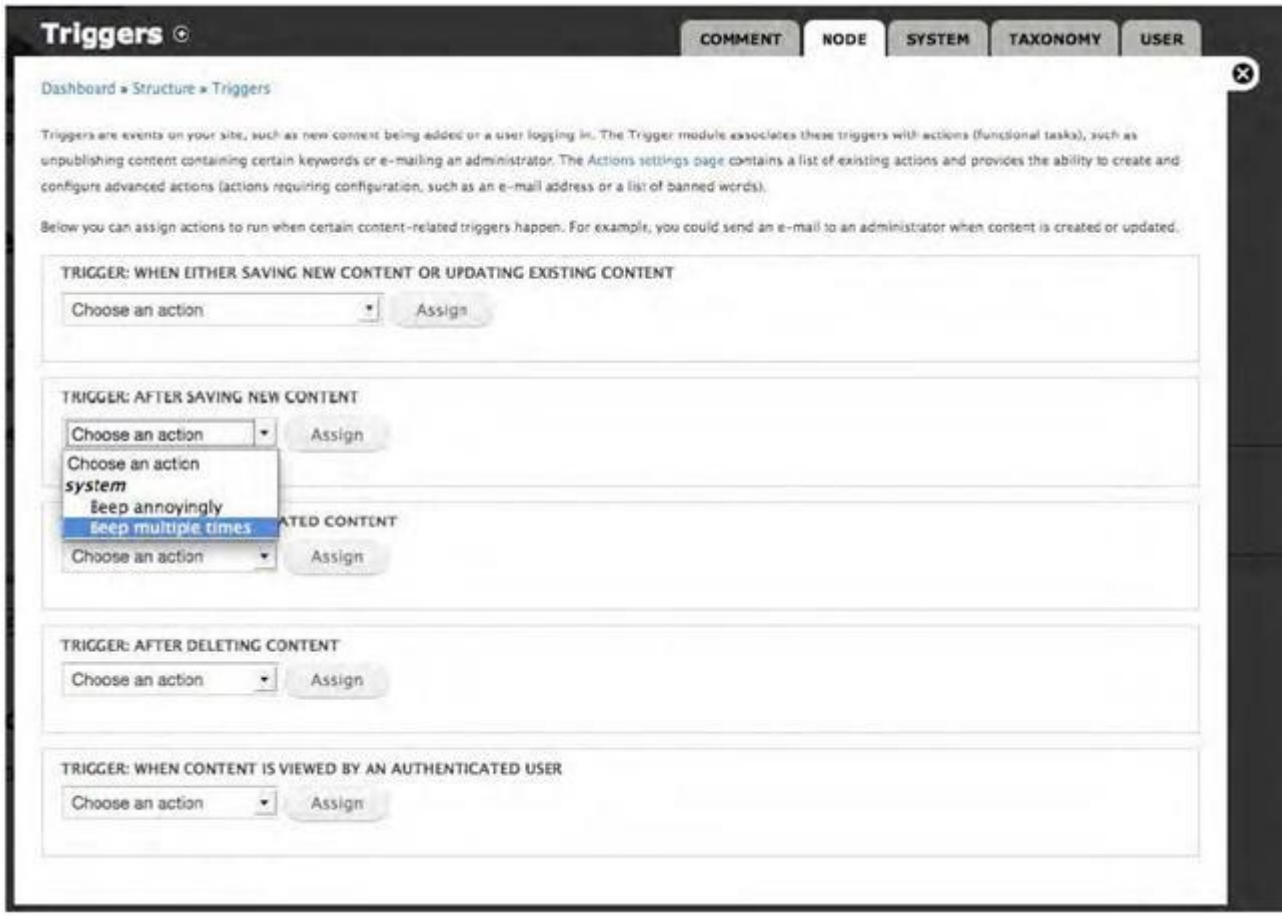

Figure 3-4. The new action appears as a choice.

现 在，我们需要提供一个表单时管理员选择他需要嘟嘟几次，我们使用 Field API 定义一个或多个字段来完成此项工作，我们还要写一些表单校验和提交的函数。函数名称基于在 hook_action_info()中定义的动作的 ID，动作 ID 当前组成是 beep_multiple_beep_action，那么惯例规定我们 add_form 到表单定义函数名称就是 beep_multiple_beep_action_form，Drupal 预计校验函数名称是动作 ID 加 _validate（beep_multiple_beep_action_validate）及提交函数的名称是动作 ID 加 _submit（beep_miltiple_beep_action_submit）。

```
/**
 * Form for configurable Drupal action to beep multiple times.
 */

function beep_multiple_beep_action_form($content) {
  $form['beeps'] = array(
    '#type' => 'textfield',
    '#title' => t('Number of beeps'),
    '#description' => t('Enter the number of times to beep when action executes'),
    '#default_value' => isset($content['beeps']) ? $content['beeps'] : '1',
```

```
    '#required' => TRUE,
  );
  return $form;
}
function beep_multiple_beep_action_validate($form, $form_state) {
  $beeps = $form_state['values']['beeps'];
  if (!is_int($beeps)) {
    form_set_error('beeps', t('Please enter a whole number between 0 and 10.'));
  }
  else if ((int) $beeps > 10) {
    form_set_error('beeps', t('That would be too abboying. Please choose fewer than
10 beeps.'));
  } else if ((int) $beeps < 0) {
    fomr_set_error('beeps', t('That would likely create a block hole! Beeps must
a positive integer.'));
  }
}

functionbeep_multiple_beep_action_submit($form, $form_state) {
  return array(
    'beeps' => (int)$form_state['values']['beeps']
  );
}
```

第 一个函数描述一个表单，只有一个字段允许管理员输入嘟嘟的次数。要访问高级动作表单，路径为：Configuration->Action，在这个 动作页面，滚动到下部有一个高级动作选择列表，点击“Beep multiple times”项，然后选择这项，Drupal 将显示一个如图 3-5 的高级动作表单。

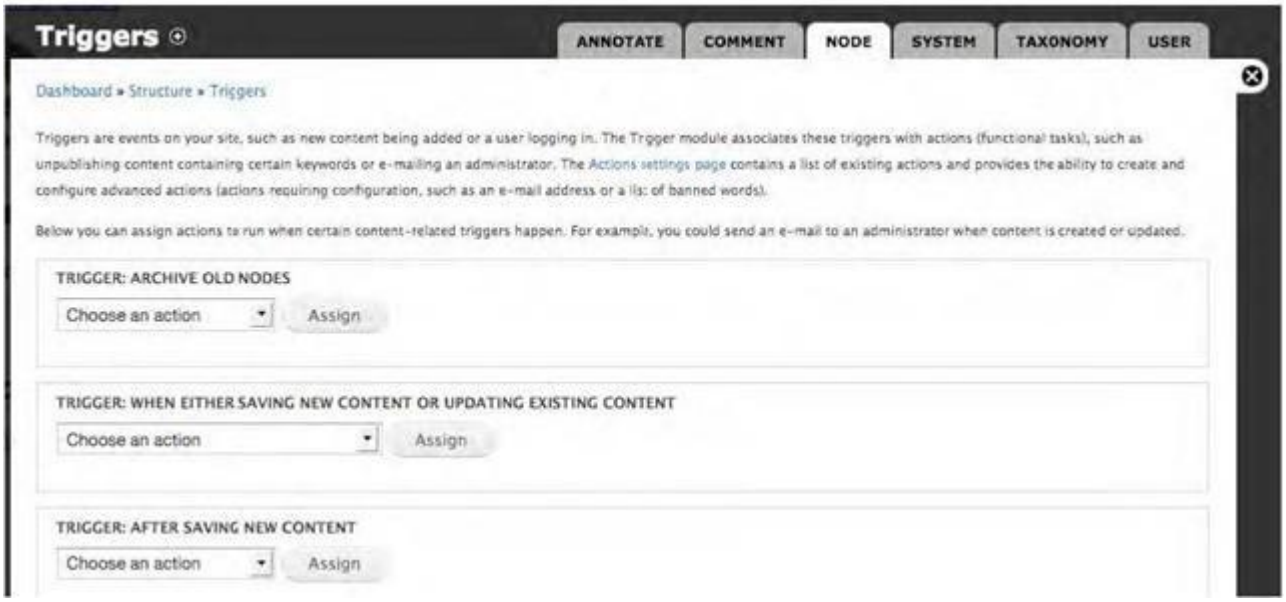

Figure 3-5. The action configuration form for the “Beep multiple times”

action

Drupal 有一个附加的动作配置表单描述字段,这个字段的值是可编辑的用来替代 action_info 钩子中定义的那个。至于这种方式，因为我们能建立一个高级动作 来嘟嘟两次并给它一个描述“Beep two times”而其它那些嘟嘟五次的则带有“Beep five times”，这种方式，我们为一个触发器指派动作时能告诉两个高级动作之间的不同。高级主题因此能以此方式制造管理员感觉。

tip：这两个动作，“Beep two times”和“Beep five times”能同样被称为“Beep multiple times”动作的实例。

校验函数就如同其它的表单校验函数（更多请看第 11 章）。在此我们检查并确信用户有效输入了一个没超过范围的数字。
提交函数返回一个动作配置表单指定的值，它应该是一个我们感兴趣的字段的键化数组，数组的值应该是在动作运行时称为可用的，描述是自动处理的，我们只需要去返回我们提供的字段，在这里是嘟嘟的次数。
最终现在是写高级动作自己的时候了：

```
/**
 * COnfigurable action. Beeps a specified number of times.
 */

function beep_multiple_beep_action($object, $context) {
  for ($i = 0; $i < $context['beeps']; $i++) {
    beep_beep();
  }
}
```

你注意到这个动作接收两个参数 $object 和$context，这是参照早先写的没带参数的简单动作。

注 意：简单动作也可以想可配置动作那样带同样的参数，因为 PHP 忽略传递过来但是在函数体声明中不出现的参数。如果我们想了解关于当前上下文的一些东西，我 们能简单改变简单动作的函数声明，将 beep_beep_action()改为 beep_beep_action($object, $context)就行，所有动作都带有$object 和$context 参数运行。

使用上下文动作

我们明确了一个动作的函数声明是 exmple_action($object, $context)，让我们审查一下每个参数的细节：

+ $object: 许多动作作用与一个 Drupal 内建的对象之上：节点、用户、taxonomy 等等。当 trigger.module 执行一个动作时，动作作用之上的对象 也在参数$object 中传递给动作。例如如果一个动作设置成在一个新节点建立后执行，$object 参数将包含节点对象。

+ $context: 一个动作能在不同的上下文中北调用。动作通过在 hook_action_info()中定义 hooks 键来声明支持哪个触发器，但是，支持多触发器的动作 需要通过确定哪个触发器

上下文的方式来运行，这种方式向，动作依赖不同的上下文有不同的活动。

触发器模块如何布置上下文

让我们设计一个场景,假设你有个网站提供了一个争端,这有一个商业模块:用户付款注册，许多只是在站点上留下一条评论，一旦他们贴了一条评论，就锁定他们，只有再次付款才能解锁。忽略经济前景，我们只把焦点放到我们怎么能用触发器和动作实现实现这些。
我们需要一个锁定当前用户的动作，仔细查看 user.module 就发现 Drupal 已经给我们提供了一个：

```
/**
 * Implements hook_action_info().
 */

function user_action_info() {
  return array(
    'user_block_user_action' => array(
      'label' => t('Block current user'),
      'type' => 'user',
      'configurable' => FALSE,
      'triggers' => array(),
    ),
  );
}
```

这个动作不会通过触发器页面显示出来，因为它声明为不支持任何钩子，triggers 键只是一个空数组，只要我们能改变它，我们真能。

使用 action_info_alter()改变存在的动作

当 Drupal 运行每个模块用来声明它们提供的动作的 action_info 钩子时，Drupal 还给模块提供一个修改信息的机会 -- 包括其他模块提供的信息，这里展示我们怎么样使“Block current user”动作在 comment insert 触发器中可用：

```
/**
 * Implementation of hook_drupal_alter(). Called by Drupal after
 * hook_action_info() so modules may modify the action_info array.
 *
 * @param array $info
 *   The relust of calling hook_action_info() on modules.
 */

function beep_action_info_alter(&$info) {
  // Make the "Block current user" action available to the
  // comment insert trigger.
```

```
  if (!in_array("comment_insert", $info['user_block_user_action']['triggers']))
{
    $info['user_block_user_action']['triggers'][] = 'comment_insert';
  }
}
```

最后结果是“Block current user action”现在可以指派，就像图 3-6 那样。

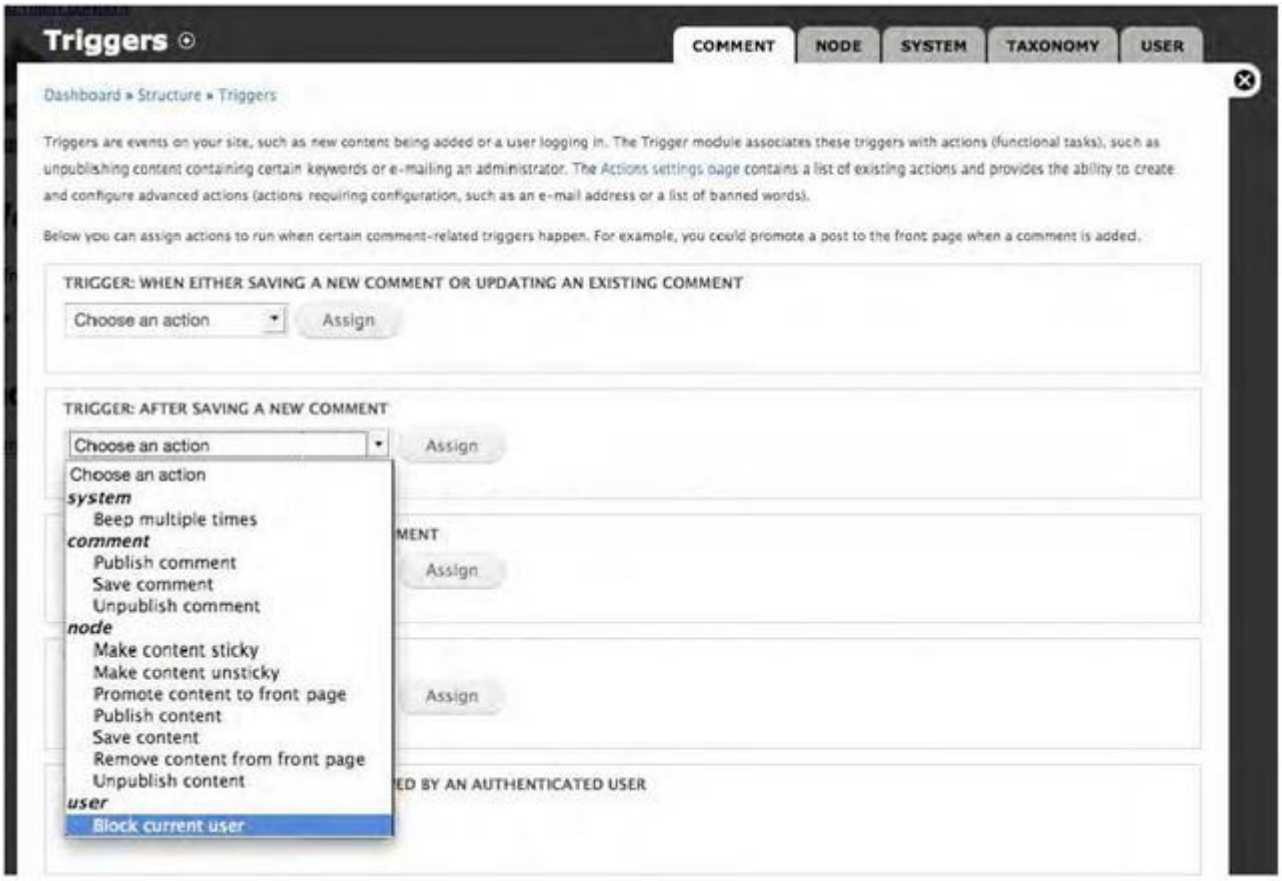

Figure 3-6. Assigning the “Block current user” action to the comment insert trigger

建立上下文

因 为我们指派的动作，当一个能的评论发布时，当前用户将被锁定。让我们看看其中发生了什么。我们已经知道 Drupal 通知模块的方式，某些事件发生将引发一 个钩子，在此情形下，这是一个 comment 钩子，特定的操作发生，这里是 insert 操作，随后一个新的评论增加了，trigger 模块实现 comment 钩子，在此钩子中，它询问数据库是否给此特定的触发器指派了动作，数据库给了它关于我们指派的“Block current user”动作的信息，现在 trigger 模块准备好去执行这个动作，这个动作有标准的动作函数声明 example_action($object, $context)。但 是我们有一个问题，这个活动作为一个 user 类型的动作执行的，不是 comment 类型，它期望它接收的对象是 user 对象！但是在这，一个 user 动作 是在 comment 钩子的上下文中调用的，关于 comment 的信息而不是关于 user 的信息将传递给钩子，我们应该怎么做？究竟发生了什么使 trigger 模块确定我们的动作是 user 动作并且载入 user 动作期望的$user 对

象？这是 modules/trigger /trigger.module 中的一块代码，给我么展示发生了什么：

```
/**
 * Loads associated objects for comment triggers.
 *
 * When an action is called in a context that does not match its type, the
 * object that the action expects must be retrieved. For example, when an action
 * that works on nodes is called during the comment hook, the node object is not
 * available since the comment hook doesn't pass it. So here we load the object
 * the action expects.
 *
 * @param $type
 *   The type of action that is about to be called.
 * @param $comment
 *   The comment that was passed via the comment hook.
 *
 * @return
 *   The object expected by the action that is about to be called.
 */

function _trigger_normalize_comment_context($type, $comment) {
  switch ($type) {
    // An action that works with nodes is being called in a comment context.
    case 'node':
      return node_load(is_array($comment) ? $comment['nid'] : $comment->nid);
    case 'user':
      return user_load(is_array($comment) ? $comment['uid'] : $comment->uid);
  }
}
```

当 上面的代码为我们的 user 动作执行时，第二种情况匹配，user 对象被载入然后我们的用户动作被执行。comment 钩子知道的信息（如 comment 的主题）存于$context 参数值传递给动作。注意动作寻找用户 ID，首先在对象中，然后在上下文，最终会退到全局变量$user：

```
/**
 * Blocks the current user
 *
 * @ingroup actions
 */

function user_block_user_action(&$entity, $context = array()) {
  if (isset($entity->uid)) {
    $uid = $entity->uid;
  }
  elseif (isset($context['uid'])) {
    $uid = $context['uid'];
```

```
  }
  else {
    global $user;
    $uid = $user->uid;
  }

  db_update('user')
    ->field(array('status' => 0))
    ->condition('uid', $uid)
    ->execute();
  drupal_session_destroy_uid($uid);
  watchdog('action', 'Blocked user %name.', array('%name' => $user->name));
}
```

动 作应该有点智能，因为它们不知道多少有关它被调用时发生什么，这就是为什么最好的动作候选人是直截了当的、原子的。tigger 模块总是在上下文中传递当 前钩子和操作，这些值存在$context['hook']和$context['op']中，这种技术提供了一个标准化的方式去为一个动作提供信息。

动作的存储

动 作是在给定的时间运行的函数。简单动作没有配置参数，例如我们建立的简单的“Beep”动作，它不需要其它信息（当然，如果需要，$object 和$context 也可用）。对比这个动作，我们建立的“Beep multiple times”动作需要知道嘟嘟多少次，其它高级动作，诸如“Send E-mail”，可能需要更多的信息：发给谁，e-mail 的主题是什么等等，这些参数应存到数据库中。

actions 表

当一个高级动作实例被管理员建立后，他在配置表单输入的信息被序列化并存入 actions 表的 parameters 字段。一个简单动作“Beep”的记录看起来这样：

```
aid: 2
type: 'system'
callback: 'beep_beep_action'
parameters: (serialized array containing the beeps parameter with its value, i.e.,
the number of times to beep)
label: Beep three times
```

只 有在一个高级动作被执行前，parameters 字段的内容才被反序列化且包含进$context 参数传递给动作，在我们“Beep multiple times”动作实例中的嘟嘟次数将作为$context['beeps']可以在 beep_multiple_beep_action()中使用。

动作 ID

注意前一节两个表内的动作 ID 内的不同，简单动作的 iD 是实际函数名，但是我们明显我们不能用函数名作为高级动作的标识符，因为相同动作的多个实例都存储了，所以用一个数字动作 ID 来代替。
动作引擎决定是否去为一个动作处理和抽取参数基于动作 ID 是否是数字。如果不是数字，动作就简单执行，不询问数据库，这是一个极快速决定，Drupal 在 index.php 中以相同的情形从菜单常量中去内容。

直接用 actions_do()调用一个动作

trigger 模块只有一种方式去调用动作，你可能想写一个独立的模块，它调用动作并自己准备参数，如果这样，使用 actions_do()是被推荐的调用动作的方式。函数声明如下：

```
actions_do($action_ids, $object = NULL, $context = NULL, $a1 = NULL, $a2 =NULL)
```

让我们检验这些参数

+ $action_ids： 要执行的动作，可以是单个动作 ID 或一个动作 ID 的数组
+ $object： 动作作用其上的对象：一个 node、user 或 comment 或 any
+ $context： 包含动作希望使用信息的关联数组，为高级数组包含配置参数
+ $a1 和$a2： 可选的附加参数，如果传给 actions_do()，那么将直接传给动作

这里演示怎样用 actions_do()调用简单动作“Beep”：

$object = NULL; // $objects a required parameter but unsed in the case $object 是必须得参数，但这种情况不设置它

```
actions_do('beep_beep_action', $object);
```

这里演示怎样用 actions_do()调用高级动作“Beep multiple times”：

```
$object = NULL;
actions_do(2, $object);
```

或者我们能绕开抽取参数来调用它:

```
$object = NULL;
$context['beeps'] = 5;
action_do('beep_multiple_beep_action', $object, $context);
```

注 意：硬编码的 PHP 开发者可能迷惑，“为什么全都使用动作？为什么不只是直接调用函数，或只是实现一个钩子？为什么为上下文藏匿参数而费心，为什么不用传 统的 PHP 参数代替检索它们？”答案是通过一个十分标准的声明格式写出动作，代码重用能委托给管理员，站点管理员可能不知道 PHP，不能调用一个 PHP 开 发者设置的功能去在新节点增加后发送 E-mail，管理员简单地挂上“Send mail”动作到一个触发器，然后当一个新节点增加后触发，但不调用任何东西。

使用 hook_trigger_info()定义你自己的触发器

Drupal 怎样知道那些触发器可用来显示在触发器用户接口页面？典型方式是，它让模块定义钩子声明那些触发器模块实现。例如，这是一个 triggers 模块自己的 hook_trigger_info()的实现，定义了所有安装了 drupal7 后可用的标准触发器：

```
/**
 * Implements hook_trigger_info().
 *
 * Defines all the triggers that this module implements triggers for.
 */

function trigger_trigger_info() {
   return array(
     'node' => array(
       'node_presave' => array(
         'label' => t('When either saving new content or updating existing
content'),
       ),
       'node_insert' => array(
         'label' => t('After saving new content'),
       ),
       'node_update' => array(
         'label' => t('After saving updated content'),
       ),
       'node_delete' => array(
         'label' => t('After deleting content'),
       ),
       'node_view' => array(
         'label' => t('When content is viewed by an authenticated user'),
       ),
     ),
     'comment' => array(
       'comment_presave' => array(
         'label' => t('When either saving a new comment or updating an existing
comment'),
       ),
       'comment_insert' => array(
         'label' => t('After saving a new comment'),
       ),
       'comment_update' => array(
         'label' => t('After saving an updated comment'),
       ),
       'comment_delete' => array(
         'label' => t('After deleting a comment'),
       ),
       'comment_view' => array(
```

```
        'label' => t('When a comment is being viewed by an authenticated user'),
      ),
    ),
    'taxonomy' => array(
      'taxonomy_term_insert' => array(
        'label' => t('After saving a new term to the database'),
      ),
      'taxonomy_term_update' => array(
        'label' => t('After saving an updated term to the database'),
      ),
      'taxonomy_term_delete' => array(
        'label' => t('After deleting a term'),
      ),
    ),
    'system' => array(
      'cron' => array(
        'label' => t('When cron runs'),
       ),
    ),
    'user' => array(
      'user_presave' => array(
        'label' => t('When either creating a new user account or updating an
existing'),
      ),
      'user_insert' => array(
        'label' => t('After creating a new user account'),
      ),
      'user_update' => array(
        'label' => t('After updating a user account'),
      ),
      'user_delete' => array(
        'label' => t('After a user has been deleted'),
      ),
      'user_login' => array(
        'label' => t('After a user has logged in'),
      ),
      'user_logout' => array(
        'label' => t('After a user has logged out'),
      ),
      'user_view' => array(
        'label' => t("When a user's profile is being viewed"),
      ),
    ),
  );
```

```
}
```
正 如你在这个函数结构中看见，每一个 trigger 的分类（如 node、comment、system、和 user）返回一个选项数组，呈现在触发器配置 页。在每个分类数组内部，你能定义 trigger 和呈现在触发器配置页面上的标签，就像下面，在这 node_insert 是触发器的名字并且 label 元 素的值是呈现在触发器配置页面上的。

```
'node_insert' => array(
  'label' => t('After saving new content'),
)
```
如果我们更新第二章的 annotate 模块去包含钩子，那些钩子看起来这样：

```
/**
 * Implementation of hook_trigger_info().
 */
function annotate_trigger_info() {
  return array(
    'annotate' => array(
      'annotate_insert' => array(
        'label' => t('After Saving new aanotations'),
      ),
      'annotate_update' => array(
        'label' => t('After saving update annotation'),
      ),
      'annotate_dalete' => array(
        'label' => t('After deleting annotation'),
      ),
      'annotate_view' => array(
        'label' => t('When annotation is viewed by an authenticated user'),
      ),
    ),
  );
}
```

在清除缓存之后，Drupal 应该应该获得新的 hook_trigger_info()的新实例并且修改触发器页面，包含一个新的 Annotation 钩子的选项卡，就像图 3-7 那样。当然，模块还要 能回应用 module_invoke()或 module_invoke_all()触发的钩子和动作，例如，模块应该需要调用 module_invoke_all('annotate_insert', 'annotate_update', 'annotate_delete', 'annotate_view')，然后，它还需要实现 hook_annotate_insert、hook_annotate_update、 hook_annotate_delete、和 hook_annotate_view 并使用 actions_do()来触发动作。

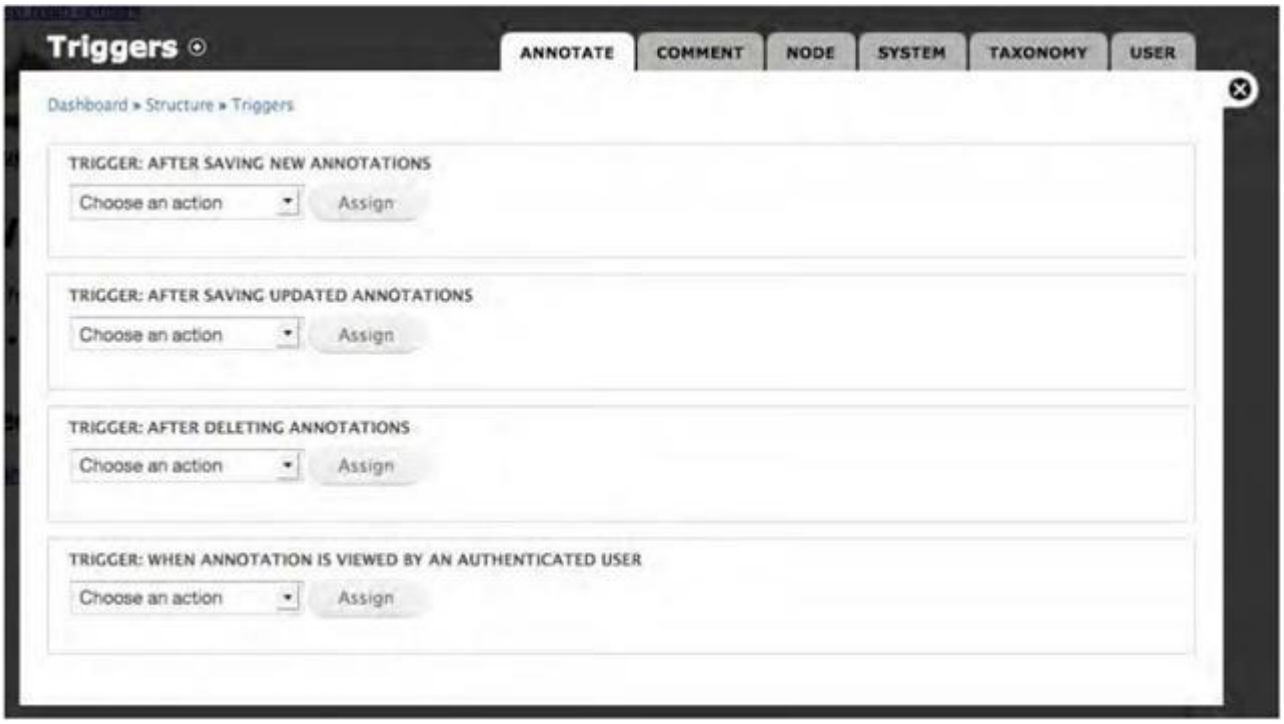

Figure 3-7. The newly defined trigger appears as a tab in the triggers user interface.

为存在的钩子增加触发器

有 时你的代码增加了一个新的操作，你可能想为一个存在的钩子增加触发器，例如，你可能想增加一个由 hook_node_archive 调用的钩子。假设你写 了一个 archive 模块它获取旧的节点并将其移到一个数据仓库，为此你能定义一个全新的钩子，这完全恰当。但是这操作是基于一个节点，你应该想去触发 hook_node_archive 代替触发器接口相同选项卡下所有呈现内容上的所有触发器，假设你的模块叫“archive”，下列代码会增加一个附加 触发器：

```
/**
 * Implementation of hook_trigger_info()
 */
function archive_trigger_info() {
  return array(
    'node' => array(
      'archive_nodes' => array(
        'label' => t('Archive old nodes'),
      )
    )
  );
}
```

新的触发器将展现在触发器管理页面触发器列表的末尾，就想图 3-8 展示的。

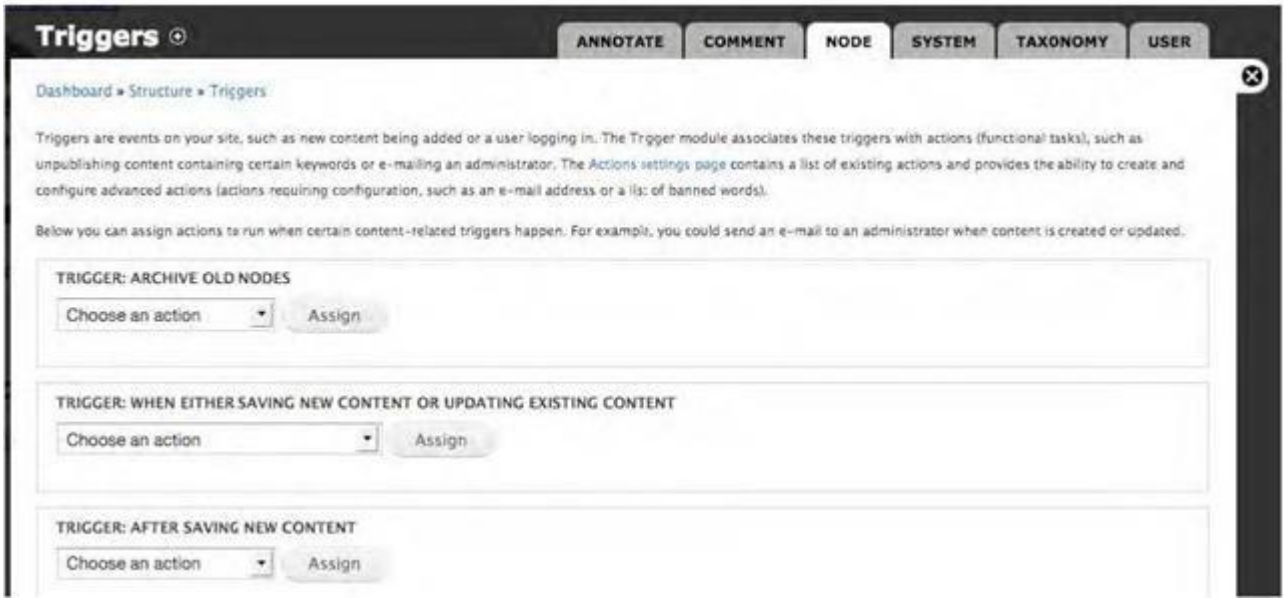

Figure 3-8. The additional trigger (“When the post is about to be archived”) appears in the user interface.

小节

阅读完本章，你应该能够：

+ 理解怎样去为一个触发器指派一个动作
+ 写一个简单的动作
+ 写一个高级动作和它的分配配置页面
+ 使用动作管理页面创建和重命名高级动作实例
+ 理解上下文是什么
+ 理解动作怎样能使用上下文来改变它们的行为
+ 理解动作怎样存储、抽取和执行
+ 定义你自己的钩子和使它们像钩子一样显示

Drupal 的菜单系统复杂但是强大，术语“菜单系统”（menu system）有些取名不当，最好认为菜单系统有三个主要职责：回调映射、访问控制、菜单定制。菜单系统基本的代码在 includes/menu.inc 中，那些可选代码包含一些定制菜单时激活的特性则在 modules/menu 中。

在本章，我们将探索回调映射怎样工作，看看怎样生成带访问控制的菜

单项、总结菜单项的不同内建类型。最后以练习如何去覆写、增加、删除存在的菜单项来结束本章，你就能尽可能无干扰地定制 Drupal 了。

回调映射

当 一个 web 浏览器对 Drupal 发出一个请求，它给 Drupal 一个 URL，从这个信息，Drupal 可以算出什么代码要运行和怎样处理这个请求，这是通 常都知道的路由和调度。Drupal 修剪掉 URL 的基本部分，使用后面的部分，叫路径（path），例如，如果 URL 是 http://example.com?q=node/3，Drupal 路径是 node/3。如果你是使用 Drupal 的 cleanURL 特性，在你的浏览器中，此 URL 就显示为 http://example.com/node/3，但是你的 web 服务器在 Drupal 看到它之前悄悄重写 URL 到 http://example.com?q=node/3，所以 Drupal 总是处理相同的 Drupal 路径，在前面的例子中，Drupal 路径不论是否是 CleanURL 都是 node/3，如何工作的更多细节请看第一章“The Web Server's Role”节。

映射 URL 到函数

通 常的途径是这样：Drupal 询问所有激活的模块去提供一个菜单项的数组，每个菜单项包含一个以一个路径为键及有关路径某些信息的数组，一个模块应提供的 这一块信息是一个页面回调（page callback），在这个上下文中，一个回调简单点就是浏览器请求特定路径时应该调用的 PHP 函数的名字。一个请求到达时，Drupal 将经历如下步 骤：

1. 构建 Drupal 路径。如果 Drupal 路径是一个到真实路径的别名，

Drupal 找到真实路径并用它来替代。例如，如果管理员用 http://example.com/?q=about 代替 http://example.com?q=node/3，Drupal 使用 node/3 作为路径。

2. Drupal 在数据库表 menu_router 中保持跟踪哪个路径映射到哪个回调，并在数据库表 menu_likes 中保持跟踪哪个菜单项是链接。有一个检查去看 menu_router 和 menu_links 表是否需要重建，在 Drupal 安装或更新后罕有发生。

3. 算出 menu_router 表中的哪一项符合这个 Drupal 路径并创建一个路径项描述要调用的回调。

4. 载入任何需要传递给回调的对象。

5. 检查用户是否有访问这个回调的权限，如果没有，返回一个“访问拒绝”消息。

6、对菜单标题和描述本地化。

7. 载入任何需要包含的文件。

8. 调用回调并返回结果，那个 index.php 传递到 theme_page()，结果是最终页面。

这个流程如图 4-1 和 4-2：

Figure 4-1. Overview of the menu dispatching process

Figure 4-2. Overview of the router and link building process

创建一个菜单项

为创建一个菜单项，我们使用 hook_menu()，hook_menu()持有一个由准备附加到一个菜单的项目组成的数组，每一个项目自己就是一个键值对组成的数组，描述菜单项的属性，下表描述了菜单项数组各键的细节：

Key	Value
title	一个必须字段，菜单项未翻译 title
title callback	一个生成 title 的函数，默认是 t()，因为这个原因，我们不把前面的 title 放在 t()函数中，如果你不想翻译，就简单设置此项为 FALSE。
description	菜单项的未翻译描述
page callback	当用户浏览此路径时调用的函数，用来显示 web 页面
page arguments	要传递给页面回调函数的参数的数组；整型值传递匹配的 URL 成分
access callback	一个函数，返回一个布尔值，判定用户是否有访问该菜单项的权限；默认为 user_access()，除非从父菜单项继承一个值
access arguments	传递给访问回调函数的参数数组；整型值传递匹配的 URL 成分
file	回调访问之前将要包含的文件；它允许回调函数处于一个单独的文件中。文件应该相对于实现模块路径，除非用“file path”选项另外指定其他值

file path 指向包含 file 指定的文件所在文件夹的路径，默认为实现 hook 的模块路径

weight 一个整数值，决定项目在菜单中的相对位置；值高的项目在下面，默认为 0

menu_name 可选，设置一个自定义菜单，如果你的项目不想放到导航菜单上

type 一个标志的位掩码，描述菜单项的特性，值可以使下面这些：

MENU_NORMAL_ITEM 正常菜单项，可以在菜单树上展示，并能被管理员移动/隐藏

MENU_CALLBACK 回调简单地注册一个路径，URL 访问时恰当的函数将被触发

MENU_SUGGESTED_ITEM 模块可以“启发”菜单项，管理员可以激活

MENU_LOCAL_TASK 本地任务，默认渲染成选项卡

MENU_DEFAULT_LOCAL_TASK 每一个本地任务集合都要提供个默认的任务，点击它是就像点击父项连接到相同路径

钩 进处理过程的最好地方是在你的模块中使用菜单钩子，它允许你去定义菜单项，这些菜单项将被包含进路由表。让我们创建一个简单的叫 menufun.module 的模块，实验菜单系统，我们映射 Drupal 路径 menufun

到我们要写的名叫 menufun_hello()的 PHP 函 数，首先，我们需要在 sites/all/modules/custom/menufun 中增加 menufun.info：

```
name = Menu Fun
description = Learning about the menu system.
package = Pro Drupal Development
core = 7.x
files[] = menufun.module
```

然后我们需要建立 sites/all/modules/custom/menufun/menufun.module 文件，它包含我们实现的菜单钩子和我们要执行的函数：

```
<?php
/**
 * @file
 * Use this module to learn about Drupal's menu system.
 */

/**
 * Implementation of hook_menu()
 */
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'page callback' => 'menufun_hello',
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

在前面的代码中，你可以看见我们用 3 个键值对数组作为项建立我的菜单（$items['menufun']）：

“title”：必需，定义未翻译的菜单 title

“page callback”：用户访问菜单路径是要调用的函数

“access callback”：典型地包含一个返回布尔值的函数

```
/**
 * Page callback
 */
function menufun_hello() {
  return t('Hello!');
}
```

激活这个模块，菜单项将被插入路由表，那么当访问 http://example.com?q=menufun 时，Drupal 就找到并允许我们的函数，展示如图 4-3。

需要注意的重要的东西是我们是定义了一个路径并映射它到一个函数，此路径是 Drupal 路径。我们将路径定义为我们的$items 数组的键值，我们使用的路径的名字与我们的模块名相同，这个实用技术保证了原始 URL 的命名空间，当然，你能定义任意路径。

Figure 4-3. The menu item has enabled Drupal to find and run the menufun_hello() function.

Page Callback 参数

有时，你可能希望提供更多的信息到映射路径的页面回调函数，首先，路径任何附加的部分都是自动向前传递，让我们像下面一样改变我们的函数：

```
function menufun_hello($first_name = '', $last_name = '') {
  return t('Hello @first_name @last_name', array('@first_name' =>
$first_name, '@last_name' => $last_name));
}
```

如果现在访问 http://example.com/?q=menufun/John/Doe，将展示如图 4-4 的情形：

Figure 4-4. Parts of the path are passed along to the callback function.

注意每一个 URL 扩展成分是怎样作为参数传递给我们的回调函数的。

你还能在菜单钩子中通过给$items 数组附加一个可选的 page arguments 键来定义页面回调参数，定义页面参数比较常用是因为它允许你在传递给回调函数的参数上增加更多控制。

例如，让我们更新我们的 menufun 模块，为我们的菜单项增加 page arguments：

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'page callback' => 'menufun_hello',
    'page arguments' => array('Jane', 'Doe'),
    'access callback => TRUE,
    'type' => NEMU_CALLBACK,
  );

  return $times;
}
```

而 后 Drupal 有明确给出页面参数给出的指令，任何剩余的路径参数是传递给回调函数但没获得解释，作为额外的参数，使用 PHP 参数重写机制。从 URL 来的 参数还是可用的，为访问它们，你应当重写你的回

调函数声明来增加 URL 参数。你修改了菜单使下列函数声明$first_name 结果是 Jane（第一个参 数）$last_name 是 Doe（第二个参数）。

```
function menufun_hello($first_name = '', $last_name = '') { ... }
```

让我们测试在页面参数和 URL 中输入 Jane Doe 的情形，进入 http://example.com/?q=menufun/John/Doe 将展现如图 4-5 的情形（如果不是这样，你就是忘了重建菜单）

Figure 4-5. Passing and displaying arguments to the callback function

如果你想使用 URL 传来的值，你要用下面的值更新页面回调函数：

```
function menufun_hello($first_name = '', $last_name = '') {

  return t('Hello @first_name @last_name from @from_first_name
@from_last_name', array('@first_name' => $first_name, '@last_name =>
$last_name));

}
```

更新你的版本，清空缓存，使用 http://example.com/?menufun 访问来看看情况。

在另外的文件中页面回调

如 果你没有指明另外情况，Drupal 假设你的页面回调能在你的.module 文件中找到，许多模块都是被分割成多个文件，以使在每个页面请求时

按条件载入 不常用的文件，菜单项 file 键（如'file => 'menu_geetings.inc'）用来指明包含回调函数的文件名。

作 为例子，我要更新 menufun.module hook_menu()函数去包含里面有回调函数的文件的名字。下列代码增加了'file' => 'menufun_greetings'到菜单项数组，我还改变页面回调到 menufun_greeting，只是演示，此回调不使用已经存在于 menufun.module 中的函数。

```
/**
 * Implementation of hook_menu().
 */

function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

下一步我们将在 menufun 目录建立名为 menufun_greeting.inc，代码是：

```
<?php

function menufun_greeting($first_name = '', $last_name = '',
$from_first_name='', $from_last_name='') {
  return t('Hello @first_name @last_name from @from_first_name
@from_last_name', array('@first_name' => $first_name, '@last_name' =>
$last_name, '@from_first_name' => $from_first_name, '@from_last_name' =>
$from_last_name));
}
```

保存这两个文件，清除缓存，测试改变后的情况，你应该能得到相同的结果，只是执行时间上扩大了调用.module 的时间。

在导航区块增加一个连接

在菜单例子中，我们声明菜单项的类型是 MENU_CALLBACK，现在改成 MENU_NORMAL_ITEM，我们指明我们不想简单地将路径映射到回调函数，我们想让 Drupal 去将它包含进一个菜单。

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
  );

  return $items;
}
```

现在菜单项目应该展示到了导航区块上，如图 4-6：

Figure 4-6. The menu item appears in the navigation block.

如果你不想将它放到那个地方，你可以增加或减少它的 weight 值，weight 是菜单定义项目中的另一个键。

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
    'weight' => -1,
    'type' => MENU_NORMAL_ITEM,
  );

  return $items;
}
```

效果如同图 4-7，菜单项可以不用改变代码而使用管理工具重排位置，Structure->Menus（菜单模块应该激活）。

Figure 4-7. Heavier menu items sink down in the navigation block.

菜单嵌套

目前为止，我们只定义了一个单个静态的菜单项，让我们用下面代码增加第二个和其它的回调：

```
function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => TRUE,
```

```
    'type' => MENU_NORMAL_ITEM,
    'weight' => '-1',
  );

  $items['menufun/farewell'] = array(
    'title' => 'Farewell',
    'page callback' => 'menufun_farewell',
    'file' => 'menufun_greeting.inc',
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
  );

  return $items;
}
```

下一步，在 menufun_greeting.inc 文件中增加如下的页面回调函数 menufun_farewell：

```
function menufun_farewell() {
  return t('Goodbye');
}
```

更新模块后，别忘记清除缓存。

Drupal 将通知第二个菜单项（menufun/farewell）路径是第一个菜单项（menufun）的子路径。因此，当渲染此菜单页面时，Drupal 将指向 第二个菜单，就像图 4-8，它也明确地设置页面顶部的面包屑踪迹来指示菜单嵌套，当然，一个主题可以将菜单和面包屑渲染成设计者需要的样式。

Figure 4-8. Nested menu

访问控制

到 目前为止，我们简单地将 access callback 键设置成 TRUE，这意味着任何人都能访问我们的菜单，通常菜单访问是通过在模块内使用 hook_permission()来定义权限来 控制的，并使用一个函数来测试他们，函数名用于定义菜单项的 access callback 键，典型是 user_access。让我们定义一个权限叫做 receive greeting，如果用户的规则不允许这个权限，那么他将在访问 http://example.com/?q=menufun 时收到“access denied”消息。

```
/**
 * Implementation of hook_permission()
 */

function menufun_permission() {
  return array(
    'receive greeting' => array(
      'title' => t('Receive a greeting'),
      'description' => t('Allow users receive a greeting message'),
    ),
  );
}

/**
 * Implementation of hook_menu().
 */

function menufun_menu() {
  $items['menufun'] = array(
    'title' => 'Menu Fun',
```

```
    'page callback' => 'menufun_greeting',
    'file' => 'menufun_greeting.inc',
    'page arguments' => array('Jane', 'Doe'),
    'access callback' => 'user_access',
    'access arguments' => array('receive greeting'),
    'type' => MENU_NORMAL_ITEM,
    'weight' => '-1',
  );
  $items['menufun/farewell'] = array(
    'title' => 'Farewell',
    'page callback' => 'menufun_farewell',
    'file' => 'menufun_greeting.inc',
    'access callback' => 'user_access',
    'access arguments' => array('receive greeting'),
    'type' => MENU_NORMAL_ITEM,
  );
  return $items;
}
```

在前面的代码中，范文将被调用 user_access('receive greeting')的结果决定，在这种方式下，菜单系统服务就像个看门人，基于用户规则决定哪个路径可以访问、哪个拒绝访问。

TIP：user_access()函数是默认的访问回调，如果你不定义访问回调函数，你的访问参数将有菜单系统传递给 user_access()函数。

子 菜单项不从它的父菜单项继承访问回调和访问参数。access argument 键应该在每个菜单项都定义，access callback 键只在其与 user_access 不同的时候定义，例外情况是当菜单项的类型是 MENU_DEFAULT_LOCAL_TASK 的时候， 它将继承父项的 access

callback 和 access arguments，所以为清晰，最好明确地定义这些键，即使是默认本地任务。

本地化和定制标题

菜单 title 有两种，静态和动态，静态标题有分配 title 键的值建立的，动态标题通过一个 title 回调来建立的。Drupal 自动翻译静态标题，所以你不要把它封装到 t()函数中，如果你用动态标题，你必须负责在你的回调中做翻译工作。

```
'title' => t('Greeting') // 不要这样做。
```

注意：description 永远都是静态的，值是由 description 键设定，由 Drupal 自动翻译。

定义标题回调

标题可以在运行时用标题回调动态创建，下面例子演示使用标题回调将标题设置成当前日期和时间，然后我用一个标题回调，此函数负责在返回值之前执行翻译，为执行翻译，我们将要返回的值封装在 t()中。

```
functionmenufun_menu() {
  $items['menufun'] = array(
    'title' => 'Greeting',
    'title_calback' => 'menufun_title',
    'description' => 'A salutation.',
```

```
    'page callback' => menufun_hello',
    'access callback' => TRUE,
  );
  return $items;
}

/**
 * Page callback.
 */
function menufun_hello() {
  return t('Hello!');
}
/**
 * Title callback.
 */
function menufun_title() {
  $now = format_date(time());
  return t('It is now @time', array('@time' => $now));
}
```

情形就如同图 4-9，菜单项标题的设置可以在运行时通过使用一个定制的标题回调来完成。

Figure 4-9. Title callback setting the title of a menu item

但是你想分隔菜单项标题和页面标题怎么办？简单 -- 你可用 drupal_set_title()来设置页面标题：

```
function menufun_title() {
  drupal_set_title(t('The page title'));
  $now = format_date(time());
  return t('It is now @time', array('@time' => $now));
```

```
}
```

结果是页面一个标题，而菜单项是另外一个标题，如图 4-10。

Figure 4-10. Separate titles for the menu item and the page

菜单项通配符

到目前为止，我们在我们的菜单项中始终使用正规的 Drupal 路径名称，名称如 menufun 和 menufun/farewell，但是 Drupal 经常使用 user/4/track 或 node/15/edit 等等，路径的某部分是动态的，让我们看看它是怎样工作的。

基本通配符

字符 % 是 Drupal 菜单项中的通配符，意味着这个值是在运行时由在 URL 的通配符位置找到的值决定，这是一个使用通配符的菜单项：

```
function menufun_menu() {
  $items['menufun/%'] = array(
    'title' => 'Hi',
    'page callback' => 'menufun_hello',
    'page arguments' => array(1),
    'access callback' => TRUE,
  );
  return $items;
}
```

此 菜单项对于 Drupal 路径 menufun/hi menufun/foo/bar menufun/123 和 menufun/file.html 都可以工作，但是它对路径 menufun 无效。菜单的路径应该分隔开来写，使它只包含路径的 一部分，通配符只匹配两部分之中的一个字符串。注意尽管%通常设计成匹配一个数字（如 user/%/edit 匹配 user/2375/edit），但实际 上它匹配位置上的任何文本。

注意：带 有通配符的菜单项将不再显示在导航菜单中，即使是菜单项 type 设置成为 MENU_NORMAL_ITEM，原因显而易见，因为带有通配符的路径结构 Drupal 不知道怎样为这个链接构建 URL，但是请看后面章节“Building Paths from Wildcards Using to_arg()函数”找到怎样告诉 Drupal 来使用 URL。

通配符和页面回调参数

在菜单路径结尾处的通配符不影响传递给页面回调的 URL 的附加部分，因为通配符只是匹配到下一个斜杠为止，继续我们的例子路径 menufun/%，URL http://example.com/?q=menufun/foo/Fred 只有字符串 foo 匹配通配符，路径的下一个位置（Fred）将被作为一个参数传递给页面回调。

使用通配符的值

要使用路径匹配的部分，需指定在 page arguments 键指定路径部分的序号：

```
function menufun_menu() {
  $items['menufun/%/bar/baz'] = array(
    'title' => 'Hi';
    'page callback' => 'menufun_hello',
    'page arguments' => array(1), // The matched wildcard.
    'access callback' => TRUE,
  );
  return $items;
}

/**
 * Page Callback.
 */
function menufun_hello($name = NULL) {
  return t('Hello. $name is @name', array('@name' => $name));
}
```

页面回调函数 menufun_hello()接收的参数展示在图 4-11 中：

Figure 4-11. The first parameter is from the matched wildcard.

第 一个参数$name 将被传给页面回调，array(1)意味着“请将路径的部分 1 传递，不管它是什么”，我们从 0 开始算，部分 0 是 menufun、部分 1 是通配符匹配的、部分 2 是 bar 等等，第二个参数$b 也将被传递，因为 Drupal 传递路径的组成部分优先于 Drupal 传递路径参数（请看 Page Callback Argument）。

通配符和参数替换

在 实际应用中，Drupal 路径成分通常用来去查看和改变一个诸如节点或用户等对象，例如，路径 node/%/edit 用来去编辑 node，路径 user /% 用来去查看关于此用户 ID 的信息。让我们注视一下后面的菜单项，那个能在 modules/user/user.module 中找到 hook_menu()的实现，这路径相应的 URL 匹配的应该像 http://example.com/?q=user/2375 这样的，那是你可能在 Drupal 页面点击“我的账户”时的 URL：

```
$items['user/%user_uid_only_optional'] = array(
  'title' => 'My account',
  'title callbcak' => 'user_page_title',
  'title arguments' => array(1),
  'page callback' => 'user_view_page',
  'page arguments' => array(1),
  'access callback' => 'user_view_access',
  'access argument' => array(1),
  'weight' => -10,
  'menu_name' => 'user-menu',
);
```

当 Drupal 用 user/%user_uid_only_optional 创建菜单，它用下列描述的程序替换%user_uid_only_optional：

1. 在第二段，匹配%后面和下一个可能的/之前的字符串，在此，字符串为 user_uid_optional。

2. 附加一个_load 到字符串尾部生成函数名，在此，函数名为 user_uid_optional_load。

3. 调用这个函数并传递一个参数，Drupal 路径中的通配符的值。如 URL 是 http://example.com/?q=user=2375，

Drupal 路径是 user/2375，通配符匹配第二个段就是 2375，一个就是 user_uid_optional_load('2375')。

4. 调用的结果然后用在通配符位置，然后标题回调使用 array(1)作为参数来调用，

代替 Drupal（2375）路径的部分 1，我们传递 user_uid_optional_load('2375')的调用结果，它是个

用户对象，想象一下它是个 Drupal 路径的部分将用它描述的对象来代替。

5. 注意页面和访问回调也要使用替换对象。那么在前面的菜单项，user_view_access()将为访问调用

user_view()将被调用生成页面内容，并且两个都要被传递用户 2375 的对象。

TIP：思考关于在一个 Drupal 路径如 node/%node/edit 中对象替换是容易的，如果年想象%node 成一个带有注释的通配符，就是说 node/%node/edit 就是 node/%/edit，带有暗示的就够来在通配符匹配上运行 node_load()。

给载入函数传递附加参数

如果有附加的参数要传递给载入函数，它们可以被定义在 load arguments 键中。下面是一个节点例子，这个菜单项查看一个节点版本，节点 ID 和节点版本 ID 都要传递给载入函数 node_load()。

```
$items['node/%node/revisions/%/view'] = array(
  'title' => 'Revisions',
  'load arguments' => array(3),
  'page callback' => 'node_show',
  'page arguments' => array(1, TRUE),
```

```
  'access callback' => '_node_revision_access',
  'access arguments' => array(1),
);
```

菜 单项指定 array(3)给 load arguments 键，这意味着为附加 nodeID 附加通配符的值，它自动传到给载入函数，就像先前的轮廓，一个附加的参数呗传到载入函数，而后 array(3)成为一个成员，在此，就是数字 3，就像你在“Using the Value of a Wildcard”节你看到的，这意味着路径的第三位置部分将被使用，此位置和路径参数例子

URL http://example.com/?q=node/56/revision/4/view 在下表中展示：

Position	Argument	Value from URL
0	node	node
1	%node	56
2	revisions	revisions
3	%	4
4	view	view

因此，定义 load argument 键意味着调用 node_load('56', '4')将替代 node_node('56')。

当页面回调运行时，载入函数将用在如的节点对象替换 56，因此页面回调是 node_show($node, NULL, TRUE)。

特殊、预定义载入参数：%map 和%index

这 有两个特殊的载入函数，%map 标志以数组形式传递 Drupal 当前路径，在前面的例子中，如果%map 作为载入参数传递，它的值应该是 array('node', '56', 'revisions', '4', 'view')。如果载入函数声明参数是一个引

用，那么 map 值可以被它操作；前面的例子标志%index 的值是 1，因为通配符在位置 1 。

使用 to_arg()函数从通配符创建路径

前 面说过，Drupal 不能从一个由通配符构成的 Drupal 路径中构建有效的链接，例如 user/%（毕竟 Drupal 不知道怎样替换这个%），但这么说不完全严密，我们能定义一个 helper 函数能为通配符生成一个替换，而后 Drupal 能用啦构建链接。在“My account”菜单项，“My account”链接的路径是有下几部产生的：

1. Drupal 路径本来是 user/%user_uid_optional。

2. 当构建链接是，Drupal 查看名为 user_uid_optional_to_arg()的函数，如果没定义 Drupal 不知道怎样绘出建立路径并且不会显示这个链接。

3. 如果函数找到了，Drupal 使用函数结果作为链接中通配符的替换，user_uid_optional_to_arg()函数返回当前用户 ID，例子如果你是用户 4，Drupal 链接“My account”到 http://example.com/?q=user/4。

函数 to_arg()的使用不是针对给定的页面执行的，就是说，to_arg()函数在任何页面的链接创立是运行的，不只是那些匹配菜单项的 Drupal 路径的页面，“My account”链接将在所有页面上显示，而不只在浏览 http://example.com.com/?q=user/3 页面时。

通配符和 to_arg()函数的特殊情形

当 Drupal 从基于 Drupal 路径的通配符跟随字符串的菜单项创建链接是要查看 to_arg()函数，这可能是任意字符串，如下例：

```
/**
 * Implementation of hook_menu().
 */
function menufun_menu() {
  $items['menufun/%a_zoo_animal'] = array(
    'title' => 'Hi',
 'page callback' => 'menufun_hello',
 'page arguments' => array(1),
 'access callback' => TRUE,
 'type' => MENU_NORMAL_ITEM,
 'weight' => -10,
  );
  return $items;
}

function menufun_hello($animal) {
  return t("Hello $anima");
}

function a_zoo_animal_to_arg($arg) {
  // $arg is '%' since it is a wildcard
  // let't replace it with a zoo animal.
  return 'tiger';
}
```

这使得链接“Hi”呈现在导航区块中，链接的 URL 是 http://example.com/?q=menufun/tiger。通常你不能用一个静态的字符串替换通配符，如此例。当然啦，to_arg()函数能提供一些动态的东西，像当前用户的 uid 和当前节点的 nid。

从其它模块变更菜单项

当 Drupal 重建 menu_router 表和更新 menu_link 表时（比如当一个新的模块被激活），模块有机会去改变一个菜单项，通过实现 hook_menu_alter()。例如，“Log off”菜单项使用 user_logout()签退当前用户，终止用户会话，然后重定向用户到站点首页。user_logout()函数在 modules/user/user.pages.inc 中，那么此 drupal 路径的菜单项有一个 file 键定义，正常情况下，当用户点击导航区块上的 “Log off”链接，Drupal 装入 modules/user/user.pages.inc 文件并运行 user_logout()，让我们改变这些重定向签 退用户到 drupal.org。

```
/**
 * Implementation of hook_menu_alter().
 *
 * @param array $items
 * Menu items keyed bu path.
 */
function menufun_menu_alter(&$items) {
  // Replace the page callback to 'user_logout' with a call to
  // our own page callback.
  $items['logout']['page callback'] = 'menufun_user_logout';
  $items['logout']['access callback'] = 'user_is_logged_in';
  // Drupal no longer has to load the user.pages.inc file
  // since it will be calling our menufun_user_logout(), which
  // is in our module -- and that's already in scope.
  unset($items['logout']['file']);
}

/**
 * Menu callback; logs the current user out, and redirects to drupal.org
 * This is a modified version of user_logout().
 */
function menufun_user_logout() {
  global $user;

  watchdog('menufun', 'Session closed for %name.', array('%name' =>
$user->name));
```

```
  // Destroy the current session.
  session_destroy();
  // Run the 'logout' operation of the user hook so modules can respond
  // to the logout if they want to.
  module_invoke_all('user', 'logout', NULL, $user);

  // Load the anonymous user so the global $user object will be correct
  // on any hook_exit() implementation.
  $user = drupal_anonymous_user();

  drupal_goto('http://drupal.org/');
}
```

在运行 hook_menu_alter()实现之前，logout 路径的菜单项看起来这样：

```
array(
  'access callback' => 'user_is_logged_in',
  'file'            => 'user.pages.inc',
  'module'          => 'user',
  'page callback'   => 'user_logout',
  'title'           => 'Log out',
  'weight'          => 10,
)
```

当我们变更之后，page callback 被设置到 menufun_user_logout：

```
array(
  'access callback' => 'user_is_logged_in',
  'module'          => 'user',
  'page callback'   => 'menufun_user_logout',
  'title'           => 'Log out',
  'weight'          => 10,
)
```

从其它模块变更菜单链接

当 Drupal 保存一个菜单项到 menu_link 表，将给模块一个改变此链接的机会，利用实现 hook_menu_link_alter()。这就是“Log out”菜单项能改变标题到“Sign off”。

```
/**
 * Implements hook_menu_link_alter().
 *
 * @param $item
 * Associative array defining a menu link as passed into menu_link_save()
 */
function menufun_menu_link_alter(&$item) {
  if ($item['link_path'] == 'user/logout') {
    $item['link_title'] = 'Sign off';
  }
}
```

这个钩子可以用来修改链接的标题或权重。如果你需要修改菜单项的其它参数诸如 access callback，请用 hook_menu_alter()。

注意：在 hook_menu_link_alter()中改变并制造一个菜单项不能重写由用户接口提供的菜单项，就是 menu.module 提供的 Administer->Site building->Menus 接口。

菜单项类别

当你在菜单钩子中增加一个菜单项，一个你可能使用的键是 type，如果你没定义一个 type，默认的 MENU_NORMAL_ITEM 将被使用。Drupal 区别对待我们分配的菜单项类型，每一个菜单项类型都是有一串 flags 或属性组成，下表列出菜单项类型 flags：

Binary Constant	Hexadecimal	Decimal	Description
000000000001	0x0001	1	MENU_IS_ROOT 项是菜单树的根
000000000010	0x0002	2	MENU_VISIBLE_IN_TREE

项在菜单树可见

000000000100 0x0004 4 MENU_VISIBLE_IN_BREADCR UMB 项在面包屑可见

000000001000 0x0008 8 MENU_LINKS_TO_PARENT 项连接到父页面

000000100000 0x0020 32 MENU_MODIFIED_BY_ADMIN 项可由管理员修改

000010000000 0x0080 128 MENU_IS_LOCAL_TASK 项是本地任务

000100000000 0x0100 256 MENU_IS_LOCAL_ACTION 项是本地动作

例 如，常数 MENU_NORMAL_ITEM（define('MENU_NORMAL_ITEM', MENU_VISIBLE_IN_TREE | MENU_VISIBLE_IN_BREADCRUMB)就有 MENU_VISIBLE_IN_TREE、 MENU_VISIBLE_IN_BREADCRUMB 标记，如下

面展示：

Binary	Constant
000000000010	MENU_VISIBLE_IN_TREE
000000000100	MENU_VISIBLE_IN_BREADCRUMB
000000000110	MENU_NORMAL_ITEM

因此，MENU_NORMAL_ITEM 有下面的标志 000000000110，下表列出可用的菜单项类型和他们体现的 flags：

Menu Flags	Menu Type Constants				
	MENU_NORMAL_ITEM	MENU_CALLBACK	MENU_SUGGESTED_ITEM*	MENU_LOCAL_TASK	MENU_DEFAULT_LOCAL_TASK
MENU_IS_ROOT					
MENU_VISIBLE_IN_TREE	X				
MENU_VISIBLE_IN_BREADCRUMB	XX		X		
MENU_LINKS_TO_PARENT				X	
MENU_MODIFIED_BY_ADMIN					
MENU_CREATED_BY_ADMIN					
MENU_IS_LOCAL_TASK			X	X	

那么哪些常数是你在定义自己的菜单项时使用的，看一下上表看看你想激活哪些标志并使用常数包含这些标志，每个常数的细节说明都在 includes/menu.inc 的注释中，大多数通常使用 MENU_CALLBACK, MENU_LOCAL_TASK, and MENU_DEFAULT_LOCAL_TASK，读一读细节。

常见任务

本节我们展示一些典型的方法去面对在使用菜单工作时开发者面对的一些问题。

给菜单指定一个回调但不指定连接

常常我们可能想映射一个 URL 到一个函数而不建个可视的菜单项，例如，可能你有个 javascript 函数在 web 表单中需要去从 drupal 中获得州

的列 表，那么你需要封装一个 URL 到一个 PHP 函数但是不需要在任何导航菜单包含这些，你能通过将 MENU_CALLBACK 指派给你的菜单项类型来做到这 些，就像本章第一个例子。

显示菜单项为选项卡

一 个显示为选项卡那样的回调一看就知是个本地任务（local task）并且有类型 MENU_LOCAL_TASK 或 MENU_DEFAULT_LOCAL_TASK。本地任务的标题应该是短动词，例如“add”或 “list”等。本地任务通常活动在特定的对象上，例如一个节点、用户。你可以将本地任务看成是一个菜单项的语义声明，它正常渲染为选项卡 -- 类似是个语义声明，通常渲染成粗体。

本地任务在渲染选项卡序列中必须有一个父项目，一个常用的惯例是分配一个回调到 一个像 milkshake 一样的根路径，然后分配本地任务到这个路径的扩展部分，如 milkshake/prepare、milkshake /drink，等等。Drupal 内建的主题支持两级本地任务选项卡（附加的层级也是潜在支持的，但是你的主题必须提供显示附加层级的功能）。选项卡渲染 的顺序是依靠菜单项标题的 alpha 排序决定的，如果不是你想要的，你可以增加一个 weight 键到你的菜单项，那就按权重排序了。下面的例子展示结果为两个主选项卡，并在默认本地任务下有两个子选项卡，建立 sites/all/modules/custom/milkshake /milkshake.info 输入：

```
name = Milkshake
description = Demostrates menu local tasks.
package = Pro Drupal Development
core = 7.x
```

```
files[] = milkshake.module
```

然后在 sites/all/modules/custom/milkshake/milkshake.module 中输入：

```
<?php

/**
 * @file
 * Use this module to learn about Drupla's menu system.
 * specifically how local tasks work.
 */

/**
 * Implements hook_menu().
 */
function milkshake_menu() {
  $items['milkshake'] = array(
    'title' => 'Milkshake flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_overview',
    'type' => MENU_NORMAL_ITEM,
  );
  $items['milkshake/list'] = array(
    'title' => 'List flavors',
    'access arguments' => TRUE,
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'weight' => 0,
  );
  $items['milkshake/add'] = array(
    'title' => 'Add flavor',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_add',
    'type' => MENU_LOCAL_TASK,
    'weight' => 1,
  );
  $items['milkshake/list/fruity'] = array(
    'title' => 'Fruity flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_list',
    'page arguments' => array(2), // pass 'fruity'.
    'type' => MENU_LOCAL_TASK,
  );
  $items['milkshake/list/candy'] = array(
```

```
    'title' => 'Candy flavors',
    'access arguments' => TRUE,
    'page callback' => 'milkshake_list',
    'page arguments' => array(3), // pass 'Candy'.
    'type' => MENU_LOCAL_TASK,
  );

  return $items;
}

function milkshake_overview() {
  $output = t('The following flavors are available...');
  // ... more code here
  return $output;
}

function milkshake_add() {
  return t('A handy form to add flavors might go here...');
}
fucntion milkshake_list($type) {
  return t('List @type flavors', array('@type' => $type));
}
```

隐藏存在的菜单项

存在的菜单项可以通过改变链接项目的 hidden 属性来隐藏，假设你由于某种原因想移除“Create content”菜单项，使用我们的老朋友 hook_menu_link_alter()：

```
/**
 * Implements hook_menu_link_alter().
 */
function milkshake_menu_link_alter(&$item) {
  // Hide the Create content link.
  if ($item['link_path'] == 'node/add') {
    $item['hidden'] = 1;
  }
}
```

使用 menu.module

激 活 Drupal 的 menu 模块提供了一个便利的用户接口，使站点管理员可以定制存在的菜单，如导航和主菜单，可以新增加一个菜单。当 menu_rebuild()函数运行时，菜单树呈现的数据结构就保存进数据库，这发生在你激活或停用模块或者其它搞混了菜单树结构的情况。数据保存进 menu_router 表，关于链接的信息保存进 menu_links 表。

当处理建立一个页面的链接时，Drupal 首先建立基于从模块的菜单钩子的实现接收到的路径信息的树，并存储进 menu_router 表，然后它按照从数据库来的菜单信息布局。这种行允许你使用 menu.module 去改变菜单树的 parent、标题、路径和描述 -- 你并没有真正改变潜在的树，当然啦，你是在它上边建立可以覆盖它的数据。

注意：菜单项类型，如 MENU_CALLBACK 或 DEFAULT_LOCAL_TASK 在数据库中用相等的十进制表示。

常见错误

你已经实现了菜单钩子，但是你的回调没有激发、你的菜单没显示出来或者就是不工作，请你检查下面列出的情况：

+ 你是否将 access callback 键射成了一个返回 FALSE 的函数？
+ 你是否忘记在菜单钩子函数结尾处增加 return $items;？
+ 你是否将 access arguments 或 page arguments 的值意外地设成了字符串而不是数组？

+ 你清空你的菜单缓存并重建菜单了吗？

+ 你要讲菜单显示为选项卡，你给它指派一个有页面回调的父项了吗？

+ 如果你使用本地任务，你做到在一个页面上最少两个选项卡了吗（这是显示它们所必须的）？

小结

读完这章，你也该可以：

+ 映射 URL 到你的模块、其它模块或.inc 文件中的函数
+ 理解访问控制如何工作
+ 理解路径中的通配符如何工作
+ 建立带映射到函数的选项卡（本地任务）的页面
+ 程序化修改存在的菜单项和链接

为加深阅读，menu.inc 的注释已经抽出，见 http://api.drupal.org/?q=api/group/menu/7

Drupal 正确运行取决于数据库。内容、评论、分类、菜单、用户、角色、权限几乎每一样东西都存储进数据库，并且作为 Drupal 用来渲染你的站点内容的必须的信息 的来源，同时控制那些用户可以访问它。Drupal 内部在你的代码和数据库之间有一个轻量级的数据库抽象层。抽象层移除了绝大多数数据库复杂的接口和不同 数据库引擎对 Drupal 的屏蔽。在本章，你将学习数据库抽象层如何工作及怎样使用它，你将看

到模块怎样能修改查询，然后，你将看到怎样去连接一个附加的 数据库（诸如精灵数据库），最后，你将练习怎样在你的模块的.install 文件中包含必要的查询来建立和更新数据库表。

定义数据库参数

Drupal 依靠你的站点的 settings.php 文件知道用什么用户名和密码来连接哪个数据库。此文件一般是 sites/axample.com /settings.php 或 sites/default/settings.php。默认的数据库连接代码看起来就像下面：

```
$database = array(
  'default' => array(
    'default' = array(
      'driver' => 'mysql',
      'database' => 'databasename',
      'username' => 'username',
      'password' => 'password',
      'host' => 'localhost',
      'port' => '',
      'prefix' => '',
    ),
  ),
);
```

这 个例子连接到 Mysql 数据库。PostgreSQL 用户的连接字符串前缀应该用 pgsqldaitimysql。显然，此处使用的数据库名、用户名、密 码对于你的数据库应该有效，它们是数据库要求的，不是 Drupal 要求的，并且它们是在你设置你数据库使用的连接账号时建立的。Drupal 安装者询问在 你的 settings.php 文件中能建立$database 数组的用户名、密码。

如果你的站点数据库是使用 sqlite，设置就相对简单，驱动应该设置到 sqlite 并且数据库应该是设置到包含数据库名称的路径。

```
$detabase['default']['default'] = array(
  'driver' => 'sqlite',
  'database' => '/path/to/databasefilename',
);
```

理解数据库抽象层

数 据库抽象层 API 的工作你不能充分理解，直到你努力多做几次。是否你有个项目，在那里你需要改变数据库系统并且耗尽了多天以详察你的代码去改变针对特定数 据库的函数和调用？当有一个数据库抽象层，只要你的查询是 ANSI-SQL，你将不用为不同的数据库写不同的查询，例如你不用调用 mysql_query()或 pg_query()，Drupal 使用 db_query()，它保证了数据库无关的商业逻辑。

Drupal 7 数据库抽象层基于 PHP's Data Object（PDO）库，用于两个打算，第一个是保证你的代码可用于任何数据库，第二个是预防用户提交数据放到查询中引发的注入攻击。抽象层建立的原则是：写一个 SQL 要比学习一个新的抽象层语言方便得多。

Drupal 还提供了一个模式 API，它允许你去描述你的数据库模式（就是你要用哪些表和字段）到一个 Drupal 通用的方式并且 Drupal 翻译它们到你使用的数据库的细节，我们将在谈到.install 文件时再说它们。

Drupal 使用检查 settings.php 文件的$databse 数组的方法来决定连接的数据库类型，例如，如果$database['default'] ['default']['driver']设置成 mysql，那么 Drupal 将包含 includes/database.mysql.ic，如果设置 等于 pgsql，Drupal 将包含 includes/database.pgsql.inc，如果设置等于 sqlite，Drupal 将包含 includes/database.sqlite.inc，这个机制见图 5-1.

Figure 5-1. Drupal determines which database file to include by examining $databases.

如果你看见一个数据库 Drupal 还不支持，你能为你的数据库自己写个封装函数的实现。更多信息参见本章最后的“Writing Your Own Database Driver”节。

连接数据库

Drupal 自动将连接数据库作为它自启动处理的一个部分，你不用担心它做了什么。

如果你是 Drupal 自身以外工作（如你写了一个 PHP 脚本或一个存在的 Drupal 自身之外的需访问 Drupal 数据库的 PHP 代码），你要使用下面的途径。

```
// Make Drupal PHP's current durectory.
chdir('/full/path/to/your/drupal/installation');

// Bootstrap Drupal up through the database phase.
include_once(('./includes/bootstrap.inc');
drupal_bootstrap(DRUPAL_BOOTSTRAP_DATABASE);

// Now you can run queries using db_query().
$result = db_query('SELECT title FROM {node}');
...
```

警 告：Drupal 经常配置为在 sites 目录中有多个文件夹，所以站点能从展现阶段移到产品阶段而不改变数据库认证，例如，你有带有测试数据库服务器认证 的 sites/staging.example.com/settings.php 和带有产品数据

库服务器认证的 sites/www.exapmle.com/settings.php，当监理一个数据库连接时，Drupal 允许你使用 sites/default/settings.php，因为只不是一个 http 请求。

执行简单查询

Drupal 的 db_query()函数常用来在活动连接上执行一个 SELECT 查询。还有其他函数执行 INSERT、UPDATE、DELETE，我也要讲一讲这些，但是首先让我们看看从数据库中提取信息。

这 里有些当写一些 SQL 语句时你需要知道的 Drupal 特有的语法，首先，表名要封装到大括号中，如果必要，你可以加上前缀，这能给它们以唯一的名字，这个 惯例允许主机提供商限制的用户在他们能建立的数据库数量中去安装已有一个数据库的 Drupal 同时能利用在 settings.php 中指定数据库前缀来避 免表名的冲突。这是个例子提取角色 2 的名字：

```
$result = db_query('SELECT name FROM {role} WHERE rid = :rid',
array(':rid', => 2));
```

注意是用:rid 作为占位符。在 Drupal 中，查询总是用占位符来写，实际分配的值就像一个键=>值对，:rid 占位符自动由数组分配给:rid 的值替换，数组用来定义所有的分配给查询占位符的值，如下例，增加占位符意味着增加参数：

```
db_query('SELECT name FROM {role} WHERE rid > :rid AND rid < :max_rid',
array(':rid' => 0, ':max_rid' => 3);
```

上面这行当由数据库有效执行时将变成下面这行：

```
SELECT name FROM role WHERE rid > 0  AND rid < 3
```

用户提交的数据必须总是在分隔开的参数中传递，这样值就能无害化以防止 SQL 注入攻击。

db_query()的第一个参数总是查询自身，剩下的参数是动态值，校验并插入查询字符串，它们的值作为键=>值对传递。

我们应该注意使用此语法是时 TRUE、FALSE、NULL 要转为他们等量的十进制值 1 或 0，这应该没问题。

让我们看一下某些例子。我们使用一个叫 joke 的数据库表，它包含 3 个字段，一个节点 ID（整型）、一个版本 ID（整型）和一个文本字段包含一个妙语。

让我们卡是一个简单的查询，从表 joke 中获取所有行的所有字段，条件是字段 vid（整型值）与$node->vid 相同：

```
db_query('SELECT * FROM {joke} WHERE vid = :vid', array(':vid' =>
$node->vid));
```

下 面，让我们使用 db_insert()在 joke 表中插入新行，我们使用->fields 和使用键是字段名，值是分配给此行内此字段键的值的键=& gt;值对数组来定义要插入的字段，另外注意语句结尾处的->execute()，它文如其名，就是执行对数据库的插入。

```
$nid = db_insert('joke')
  ->fields(array(
    'nid' => '4',
    'vid' => 1,
    'punchline' => 'And the pig said oink!',
))
->execute();
```

下 面让我们更新 joke 表中的所有字段，设置 punchline 等于“Take my wife，please!”，条件是 nid 大于或等于 3。我要传递一个字段和值的数组去用->fields 更新，并且我还要使用 ->condition 设置条件。在这个例子中，我要更新 joke 表中任何 nid 大于等于 3 的记录的 punchline 字段：

```
$num_updated = db_update('joke')
  ->fields(array(
    'punchline' => 'Take my wife please!',
  ))
  ->condition('nid', 3, '>=')
  ->execute();
```

如果我们想看更新影响到了多少行，我能使用更新执行后分配的值 $num_updated。

最终，让我们从 joke 表中删除所有 punchline 等于“Take my wife please!”的所有行，我将用 db_delete 函数和条件->修饰符去指定条件一从表中删除记录。

```
$num_deleted = db_delete('joke')
  ->condition('punchline', 'Take my wife please!')
  ->execute();
```

检索查询结果

有不同的方式去检索查询的结果，这依赖于你是否需要单个的行，或者全部结果集，或者你计划获得获得某一范围的结果，以便在分页显示的内部使用。

获得单个值

如果你需要从数据库获得一个单个的值，你可以使用->fetchField()方法去检索这个值，这是一个例子获取 joke 表的记录数：

```
$nbr_record = db_query("SELECT count(nid) FROM {joke}")->fetchField();
```

获得多行

大多数情况，你想从数据库返回多于一个字段的值，这是典型的迭代方式逐步访问结果集：

```
$type = 'page';
$status = 1;

$result = db_query("SELECT nid, title FROM {node} WHERE type = :type AND
status =:status",
  array(
    ':type' => $type,
    ':status' => 1,
  ));
foreach ($result as $row) {
  echo $row->title . '<br />';
}
```

上 面的代码片段将打印出所有的已发布的类型为 page 的节点的标题（status 字段为 0 表示未发布的，为 1 表示已发布），调用 db_query() 返回一 个结果数组，数组的每个元素是表中符合查询条件的一行，使用 foreach 循环访问结果集数组，并分行打印出每个节点的标题。

使用查询生成器和查询对象

Drupal7 的一个新特性是提供一个能力去用一个查询生成器来构建查询对象。在前面的例子中，我们的查询相对简单，但是如果一些复杂的查

询怎么写？手头正好有查询对象使用的查询生成器，让我们展示一个例子，然后在这个概念上将演示去创建复杂的多的查询。

在较早例子中，我建立一个查询从 role 表中选择值，条件是 role ID 大于或等于 2，如下：

```
$result = db_query('SELECT name FROM {role} WHERE rid = :rid',
array(':rid' => 2));
```

我将使用查询对象和查询生成器写出相同的查询，首先我将用要 select 的表及分配给它一个的表标识（这里是 r）来创建一个查询对象，然后，我就可以从这个表中引用字段。

```
$query = db_select('role', 'r')1
```

下一步，我将展开查询去包含一个条件就是 rid=2 的哪些字段是我想要从表中返回的。

```
$query
  ->condition('rid', 2)
  ->fields('r', array(name));
```

最终，我执行查询，将结果集指派给$result。

```
$result = $query->execute();
```

循环访问查询返回数组来打印输出。

```
foreach ($result as $row) {
  echo $row -> name . '<br />';
}
```

使用查询对象和查询生成器使构建复杂的数据库查询简单化，如何使用查询生成器的演示将在下面的例子中。

获取有限范围的结果

执 行可能返回几百甚至几千条记录的查询是一种风险，这是你写查询时需要思考的。有一个机制使这种风险最小化，就是使用范围修饰符去约束查询返回记录的最大数 量。一个例子是返回所有类型为“page”的节点的查询，如果站点有超过 1000 的节点，这种查询如果执行了，用户可能就会被信息淹没，不知所措，你能使 用范围修饰符去限制你的查询返回的行数，这可以缓解潜在的运行时间太长或信息太多的压力。下面的查询增加了范围修饰符，设置了开始位置和最大返回数量。

```
$query = db_select('node', 'n');

$query
  ->condition('type', 'page')
  ->fields('n', array('title'))
  -range(0, 100);
$result = $query->execute();

foreach($result as $row) {
  echo $row->title. '<br />';
}
```

分页显示结果

如 果你的查询范围大量的行，你可能考虑使用分页器，一个分页程序在一个页面上显示有限的行数，并提供一个导航元素允许浏览者导航。一个例子是一个 100 行的 查询，你能配置查询某一时间显示 10 行，可以点击“next”去显示下 10 行，“previous”显示前 10 行，“first”显示头 10 行，“last”显示最后 10 行，或者点击页面号码直接跳到指定页的结果（例如点击 5 将获得 51 行到 60 行）。

为演示分页程序，我将创建一个查询，它将返回 node 表中的所有节点，每页显示 10 行，下部有个分页器。

首先我建立查询对象，指令 Drupal 建立一个使用分页器的查询对象来扩展查询对象。

```
$query = db_select('node', 'n')->extend('PagerDefault');
```

下一步我将增加条件、字段、和同时显示行数的数量，使用 limit 修饰符。

```
$query
  ->condition('type', 'page')
  ->fields('n', array('title'))
  ->limit(10);
```

下一步执行查询，遍历结果集，将每一行都增加给一个输出遍历，命名为$output

```
$output = '';
foreach ($relult as $row) {
  $output .= $row->title . '<br />';
}
```

下一步调用 themeing 函数并应用分页器到我的输出，输出结果是同时显示 10 个项目，下部带有分页器（见图 5-2），并显示结果。分页器怎样处理数据库结果的细节和主题层渲染分页结果的细节见 includes/pager.inc。

```
$output .= theme('pager');
print $output;
```

Figure 5-2. Drupal’s pager gives built-in navigation through a result set.

其它常用查询

Drupal'7 数据库层提供了另外几个你可能想使用的常用查询功能。第一个例子是排序结果集。使用 orderBy 方法允许你去给结果集排序，此例显示按标题的字母顺序排序结果集：

```
$query
  ->condition('type', 'page')
  ->fields('n', array('title'))
  ->orderBy('title', 'ASC');
```

下一个例子是反序

```
$query
  ->condition('type', 'page')
  ->fields('n', array('title', 'changed'))
  ->orderBy('changed', 'DESC')
  ->orderBy('title', 'ASC');
```

这是一个可能产生重复结果的查询，这种情况，可以使用 distinct 方法过滤掉重复的记录。

```
$query
  ->condition('type', 'page')
  ->fields('n', array('title', 'changed'))
  ->orderBy('changed', 'DESC')
  ->orderBy('title', 'ASC')
  ->distinct();
```

更多例子和细节见 http://drupal.org/node/310069.

使用 druapl_write_record()插入和更新

一个常见的问题是插入一个新的数据库行及更新一个存在的行。代码典型地是测试是插入还是更新，然后执行适当的操作。

因为 Drupal 的每个表都是用关系模式描述的，Drupal 知道一个表中有什么字段及知道字段有什么默认值，通过传递一个键化的字段和值的数组给 drupal_write_record()，你能让 Drupal 生成和执行 SQL 以代替你手工写它们。

假设你有个表，保持跟踪你的大量小兔子集合，你模块中描述这个表的图表钩子应该这样：

```
/**
 * Implements hook_schema().
 */
function bunny_schema() {
  $schema['bunnies'] = array(
    'description' => t('Store infomation abuot giant rabbits.'),
    'fields' => array(
      'bid' => array(
        'type' => 'serial',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t('Primary key: Aunique ID for each bunny.'),
      ),
      'name' => array(
        'type' => 'varchar',
        'length' => 64,
        'not null' => TRUE,
        'description' => t('Each bunny gets a name.'),
      ),
      'tons' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t('The weight of the bunny to the nearest ton.'),
      ),
    ),
    'primary key' => array('bid'),
```

```
    'indexes' => array(
      'tons' => array('tons'),
    ),
  );
  return $schema;
}
```

插入一个新纪录就如同更新一个记录一样简单：

```
$table = 'bunnies';
$record = new stdClass();
$record->name = t('Bortha');
$record->tons = 2;
drupal_write_record($table, $record);

// The new bunny ID, $record->bid, was st by drupal_write_record()
// since $record is passed by refrence.
watchdog('bunny', 'Added bunny with id %id.', array('%id' =>
$record->bid));
// Change our mind about the name.
$record->name = t('Bertha');
// Now update the record in the database.
// For updates we pass in the name of the table's primary key.
drupal_write_record($table, $record, 'bid');

watchdog('bunny', 'Updated bunny with id %id.', array('%id" =>
$record->bid));
```

数组语法也是支持的，虽然假如$record 是个数组，drupal_write_record()

也将在内部将其转换为对象。

Schema API

Drupal 通过数据库抽象层支持多种数据库（Mysql、PostreSQL、SQLite

等等）。每一个想拥有数据库表的模块都使用一个关系模式定义来为

Drupal 描述这个表，Drupal 随后将这个定义翻译成数据库能懂得语法。

使用模块的.install 文件

如同第 2 章展示的，当你写一个模块它需要一个或多个数据库表来存储东西，在模块发布的.install 文件中可以含有指令去建立和维护表结构。

创建表

在安装一个新模块时，Drupal 自动检查是否在模块的.install 文件中有模式定义（图 5-3），如果模式定义存在，Drupal 就建立模式定义的数据库表，下面演示一个模式定义的通常结构：

```
$schema['tablename'] = array(
  // table description
  'description' => t('Description of what the table is used for.'),
  'fields' => array(
    // Fields definition.
    'field1' => array(
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
      'default' => 0,
      'description' => t('Description of what this field is used for.'),
    ),
  ),
  // Index declarations.
  'primary key' => array('field1'),
);
```

Figure 5-3. The schema definition is used to create the database tables.

让我们看一下 modules/book/book.install 中的 Drupal book 表的模式定义。

```
/**
 * Implements hook_schema().
 */
function book_schema() {
  $schema['book'] = array(
```

```
    'description' => 'Stores book outline information. Uniquely connects
each node in the outline to a link in {menu_links}',
    'fields' => array(
      'mlid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {menu_links}.mlid.",
      ),
      'nid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book page's {node}.nid.",
      ),
      'bid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => "The book ID is the {book}.nid of the top-level
page.",
      ),
    ),
    'primary key' => array('mlid'),
    'unique keys' => array(
       'nid' => array('nid'),
    ),
    'indexes' => array(
      'bid' => array('bid'),
    ),
  );
  return $schema;
}
```

这个定义描述了 book 表，有三个 int 型字段，还有一个主键、一个唯一的索引（意味着此字段的所有条目都是唯一的）、一个常规索引。注意当一个字段在它的描述中引用其它的表时，要加花括号。然后激活模式模块去建立到表描述的方便的超链接。

使用模式模块

此时此刻，你可能想“太折磨人啦，建立巨大的描述数组去告诉 Drupal 我的表时什么样的真是苦差事”，但是，不要烦恼，只需简单下载 schema 模块并激活它（http://drupal.org/project/schema），进入 Structure->Schema 将给你能力通过点击选项卡去看数据库的任何一个模式定义，如果你使用 SQL 建立你的表，你能使用 schema 模块获得模式定义，然后拷贝粘贴进你的.install 文件。

Tip：你可能很少去白手起家写一个模式，作为替代，使用你已经存在的表和 schema 模块的选项卡去建立你自己的模式，你还能使用其它的工具，比如 Table Wizard 模块去将 Drupal 的任何表的细节显示给 Views 模块。

schema 模块还允许你去任何模块的模式，例如图 5-4 展示 book 模块模式在 schema 模块中的现实情况，注意表及字段描述中的花括号内的表名是怎样变成有用的链接的。

Figure 5-4. The schema module displays the schema of the book module

模式字段类型映射到数据库

在 模式中声明的字段类型要映射到数据库中的实际类型。例如，一个以 tiny 尺寸声明的 integer 字段将成为 Mysql 的 TINYINT 字段或

PostgreSQL 的 small int 字段，实际映射能通过 getFieldTypeMap()函数来打印出来便于查看，或者查看表 5-1 来查看。

+ Textual

Textual 字段包含文本。

+ Varchar

varchar 或者可变字符字段，是频繁使用的字段类型来存储长度小于 256 个字符的文本。字符串最大长度是有键 length 定义的，Mysql 的 varchar 字段长 度为 0-255（5.0.2 及以前版本）和 0-65535（5.0.3 及以后） PostgreSQL 的 varchar 字段可能更长。

```
$field['fieldname'] = array(
  'type' => 'varchar', // Required
  'length' => 255, // Required
  'not null' => TRUE, // Default to FALSE.
  'default' => 'chocolate', // See below.
  'description' => t('Always state the purpose of your field.'),
);
```

如果 default 键没有设置同时 not null 键设置为 FALSE，默认值将设置为 NULL。

+ Char

char 字段是定长的字符字段，它的字符串长度由 length 键定义，MySQL 的 char 字段长度为 0-255 个字符。

```
$field['fieldname'] = array(
```

```
  'type' => 'char',
  'length' => 64,
  'not null' => TRUE,
  'default' => 'strawberry',
  'description' => t('Always state the purpose of your field.'),
);
```

如果 default 键没有设置同时 not null 键设置为 FALSE，默认值将设置为 NULL。

+ Text

Text 字段为相当长的文本数据使用的，例如 node_revisions 表中的 body 字段（body 文本在这里存储），text 字段可以不使用默认值。

```
$field['filename'] = array(
  'type' => 'text',
  'size' => 'small',
  'not null' => TRUE,
  'description' => t('Alway state the purpose of your field.'),
);
```

+ Numerial

Numerical 字段用于存储数字诸如整数、序列号、浮点数、和数值型。

+ Integer

这个字段类型用来存储整数例如节点 ID，如果 unsigned 键为 TRUE，那么就不允许负数。

```
$field['fieldname'] = array(
  'type' => 'int',  // Required
  'unsigned' => TRUE, // Defaults to FALSE.
  'size' => 'small', // tiny | small | medium | normal | big
```

```
  'not null' => TRUE, // Defaults to FALSE.
  'description' => t('Always state the purpose of you field.'),
);
```

+ Serial

serial 字段保存增量字段，例如，一个节点增加了，node 表的 nid 字段将增长。这是由插入一行并调用 db_last_insert_id()完成的，如果一行 的增加是和其它行的增加交织在一起的，并返回了最后的 ID，正确的 ID 还是被返回，因为它是基于“每次连接”的，一个 serial 字段应该被索引，它通常 用来索引成主键。

```
$field['fieldname'] = array(
  'type' => 'serial', // Required
  'unsigned' => TRUE, // Defaults to FALSE Serial number are usually
positive.
  'size' => 'small', // tiny | small | medium | normal | big
  'not null' => TRUE,
  'description' => t('Always state the purpose of you field.'),
);
```

+ Float

float 字段存储浮点数，这与 tiny,small,medium,normal 明显不同，与之对比，尺寸指明了双精度字段。

```
$field['fieldname'] = array(
  'type' => 'float', // Required
  'unsigned' => TRUE,
  'size' => 'normal', // tiny | small | medium | normal | big
  'not null' => TRUE,
  'description' => t('Always state the purpose of you field.'),
);
```

+ Numeric

Numeric 数据类型允许你去指定一个数字的精度和小数位，精度是这个数字有效位的总数，小数位是小数点右边位数的总数，例如 123.45 的精度是 5，小数位是 2。写的时候不再使用 size 键了，numeric 字段在 Drupal 的核心中不常用。

```
$field['fieldname'] = array(
  'type' => 'numeric', // Required
  'unsigned' => TRUE,
  'precision' => 5, // Significant digits.
  'scale' => 2, // Digita to the right of the decimal.
  'not null' => TRUE,
  'description' => t('Always state the purpose of you field.'),
);
```

+ 日期和时间：Datetime

Drupal 核心不使用这个数据类型，更偏爱在整数字段中使用 Unix 时间戳，datetime 格式是包含日期和时间的组合格式。

```
$field['fieldname'] = array(
  'type' => 'datetime',
  'not null' => TRUE,
  'description' => t('Always state the purpose of you field.'),
);
```

+ 二进制：Blob

二进制大对象数据（blob）类型用来存储二进制数据（例如，Drupal 的缓存表去存储缓存数据），二进制数据可以包含音乐、图片、或视频，可用两个尺寸，normal 和 big。

```
$field['fieldname'] = array(
  'type' => 'blob',
  'size' => 'normal',
  'not null' => TRUE,
  'description' => t('Always state the purpose of you field.'),
);
```

用 mysql_type 声明特别的列类型

如 果你知道精确的数据库引擎列类型，在模式定义中你能设置 mysql_type（或 pgsql_type）键，这将覆盖 type 和 size。例如 MySQL 有一个字段类型叫 TINYBLOB 对应小的二进制大对象，要指定 Drupal 在 MySQL 上运行时应该使用 TINYBLOB，如果在另外不同的数据库引擎运行则使用正规的 BLOB 类型，你应该这样写模式定义：

```
$field['fieldname'] = array(
  'mysql_type' => 'TINYBLOB', // MySQL will use this.
  'type' => 'blob', // Other databases will use this.
  'size' => 'normal', // Other databases will use this.
  'not null' => TRUE,
  'description' => t('Wee little blobs.')
);
```

Mysql 和 PostgreSQL 的真实类型如下表：

模式定义使用的类型 数据库实际使用的类型

Type size ____MySQL____ ___PostgerSQL___ ___SQLite____

---

varchar normal VARCHAR varchar VARCHAR

char normal CHAR character VARCHAR

text tiny TINYTEXT text TEXT

text small TINYTEXT text TEXT

text medium MEDIUMTEXT text TEXT

text	big	LONGTEXT	text	TEXT
text	normal	TEXT	text	TEXT
serial	tiny	TINYINT	serial	INTEGER
serial	small	SMALLINT	serial	INTEGER
serial	medium	MEDIUMINT	serial	INTEGER
serial	big	BIGINT	bigserial	INTEGER
serial	normal	INT	serial	INTEGER
int	tiny	TINYINT	smallint	INTEGER
int	small	SMALLINT	smallint	INTEGER
int	medium	MEDIUMINT	int	INTEGER
int	big	BIGINT	bigint	INTEGER
int	normal	INT	int	INTEGER
float	tiny	FLOAT	real	FLOAT
float	small	FLOAT	real	FLOAT
float	medium	FLOAT	real	FLOAT
float	big	DOUBLE precision	double	FLOAT
float	normal	FLOAT	real	FLOAT
numeric	normal	DECIMAL	numeric	NUMERIC
blob	big	LONGBLOB	bytea	BLOB
blob	normal	BLOB	bytea	BLOB
datetime	normal	DATETIME	timestamp	TIMESTAMP

维护表

当你创建一个模块的新版本时，你可能改变数据库模式，也行增加一个新的列或在某一个列上增加所有，你不能简单地删除和重建表，因为它包含了数据。这展现了怎样平滑地改变数据库。

1. 更新你的.install 文件中的 hook_schema()，那么新用户安装新版的模块就可以安装新的模式了。你的.install 文件中的模式定义要永远是最后的表和字段的模式定义。

2. 通过写一个更新函数给老用户一个更新方法。更新函数写法特殊，以基于 Drupal 版本号的数字开始，例如版本 Drupal7 的模块第一次更新函数叫 modulename_update_7000()，第二次的就叫 modulename_update_7001()。这有个 modules/comment/comment.install 的例子，展示表名有 comments 改为 comment：

```
/**
* Rename {comments} table to {comment}.
*/
function comment_update_7002() {
  db_rename_table('comments', 'comment');
}
```

3. 这个函数将在更新模块后运行 http://exmple.com/update.php 时运行。

警告：因为你每次添加一个表、字段、或者索引时，都会修改 hook_schema()实现中的模式定义，所以你的更新函数千万不要使用这里的模式定义。你可以把 hook_schema()实现中的模式定义看成是当前的，而把更新函数中的模式看成是过去的。参看 http://drupal.org/node/150220。

用来处理模式的函数的完整列表，可参看 http://api.drupal.org/api/group/schemaapi/7

TIP：Drupal 会追踪一个模块当前所用的模式版本。这一信息存储在 system 表中。在运行完本节所示的更新以后，评论模块对应记录中的 schema_version 的 值就变成了 6003。为了让 Drupal 忘记该项，可以使用 devel 模块中的“Reinstall Modules”（重装模块）选项，或者从 system 表中删除该模块的记录。

在卸载时删除表

当一个模块被禁用后，这个模块存储在数据库中的任何数据都剩下来不可触及，以防管理员有心要重用这个模块。模块页面有一个卸载选项卡它可以自动从数据库删除数据，同时你可能想删除你定义的任何变量。这有个第 2 章我们写的 annotation 模块的例子：

```
/**
 * Implements hook_uninstall().
 */
function annotate_uninstall() {
  // Clean up our entry in the variables table.
  variable_del('annotate_nodetypes');
}
```

用 hool_schema_alter()改变存在的模式

通 常模块创建和使用它自己的表，但是如果你的模块想修改一个存在的表的情况呢？假设你的模块绝对地增加一个列到 node 表，一种简单的方法是到你的数据库并 增加一列，但是然后那个应该映射到实际数据库表的 Drupal 模式定义将会变得不一致；这有个最好的方式：hook_schema_alter()。

假 设你有个模块它以某种方式标记节点，并且为效率的原因，你完全地在存在的 node 表上设置来代替你自己的关联到节点 ID 的表，你的模块需要做两个东西：在 你的模块安装期间修改 node 表并且修改要映射到实际数据库表的模式定义。功能由 hokk_install()完成，后来又由 hook_schema_alter()完成。假定你的模块叫 markednode.module，你的 markednode.install 文件应该包 含下面的函数：

```
/**
 * Implements hook_install().
 */
function markednode_install() {
  $field = array(
    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'default' => 0,
    'initial' => 0, // Sets initial value for preexisting nodes.
    'description' => t('Whether the node has been marked by the markednode
module.'),
  );

  // Create a regular index called 'marked' on the field named 'marked'.
  $keys['indexes'] = array(
    'marked' => array('marked')
  );
  db_add_field('node', 'marked', $field, $keys);
}

/**
 * Implements hook_schema_alter(). We alter $schema by reference.
 *
 * @param $schema
 * The system-wide schema collected by drupal_get_schema().
 */
function markednode_schema_alter(&$schema) {
  // Add field to existing schema.
  $schema['node']['fields']['marked'] = array(
    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'default' => 0,
    'description' => t('Whether the node has been marked by the markednode
module.'),
  );
}
```

用 hook_query_alter()修改其它模块的查询

这 个钩子用于修改 Drupal 别的地方的你不能直接修改的模块的查询。所有动态选择的查询对象都通过 execute()方法在查询字符串完成之前立即传递给 hook_query_alter()，给模块修改查询的机会是非常希望的。hook_query_alter()接受单个参数:select 查询对象本 身。

hook_query_alter()怎样工作有个例子，一个模块叫 dbtest 利用 hook_query_alter()去修改两个查询， 第一个修改发生在找到一个标记“db_test_alter_add_range”时，如果找到这个标记，查询就做了修改，增加 range(0,2)到查 询，第二个修改当查询带有标记“db_test_alter_add_join”时发生，这时，一个关联增加到表 test 和 people 之间。

注意：Tags 是标记一个查询的字符串，一个查询可以有任何数量的标记。标记常用来标记一个查询使修改钩子决定是否执行动作。标记必须全部小写且只包含字母、数子、下划线、一字母开头，就是说，它们追随通用 PHP 标识符的规则。

```
function dbtest_query_alter(SelectQuery $query) {
  // you might add a range
  if ($query->hasTag('db_test_alter_add_range')) {
    $query->range(0, 2),
  }

  // or add a join
  if ($query.hasTag('db_test_alter_add_join')) {
    $people_alias = $query->join('test', 'people',
"test_task.pid=people.id");
    $name_field = $query->addField('name', 'people', 'name');
    $query->condition($people_alias . '.id', 2);
  }
 ...
```

```
?>
```

Drupal 中连接多个数据库

虽 然数据库抽象层使记住函数名变得简单，它还增加了内建的查询安全机制，但有时，我们需要连接第三方数据库或传统数据库，并且它同样需要大量使用 Drupal 数据库 API 及获得安全上的好处，好消息是我们能做到，例如，你的模块能打开一个到非 Drupal 数据库的连接并抽取数据。

在 settings.php 文件中，$databases 是一个由多个数据库连接字符串组成的数组，这是默认语法，描述一个单个的连接：

```
array(
  'driver' => 'mysql',
  'database' => 'databasename',
  'username' => 'username',
  'password' => 'password',
  'host' => 'localhost',
  'port' => 3306,
  'prefix' => 'myprefix_',
);
```

作为一个例子，你可以有两个数据库，默认数据库和一个传统数据库：

```
$databases = array(
  'default' =>array(
    'default' =>array(
      'driver' => 'mysql',
      'database' => 'd7',
      'username' => 'username',
      'password' => 'userpassword',
      'host' => 'localhost',
      'port' => '',
      'prefix' => '',
    ),
  ),
```

```
  'legacy' =>array(
    'default' =>array(
      'driver' => 'mysql',
      'database' => 'legacydatabase',
      'username' => 'legacyusername',
      'password' => 'legacyuserpassword',
      'host' => '122.185.22.1',
      'port' => '6060',
    ),
  ),
);
```

注意：你 Drupal 站点使用的数据库应该总是用 default 键定义。

如果你要在 Drupal 中连接一个其他的数据库，你要用它的键名激活它并在完成后切换回默认连接。

```
// Get some infomation from a non-Drupal database.
db_set_active('legacy');
$result = db_query("SELECT * FROM lpad_user WHERE uid = :uid",
array(':uid' => $user->uid));

// Switch back to default connection when finished.
db_set_active('default');
```

警告：如 果你切换到不同的数据库连接，然后试图做某些东西，如 t("text")，它将发生一个错误。t()函数需要数据库活动并且数据库连接保持切换，甚至在你 切换它的代码空间之外，因此，为应对这个问题总是要小心地将数据库连接切换回 default，并且在实践中你要小心不要调用能转换为数据库请求的代码。

因为数据库抽象层被设计成为每个数据库都使用标识的函数名，多种后端数据库不能同时使用，然而，你可以在 http://drupal.org/node/19522 查

看更多信息，看怎样在同一个站点上允许 Mysql 和 postgreSQL 两种链接。

使用一个临时表

如果你进行大量的处理，你可能需要在请求发生时去建立一个临时表，你能使用下面格式来调用 db_query_temporary()：

```
$tablename = db_query_temporary($query, $arguments, $options);
```

+ $query 是准备好了的要执行的查询语句
+ $arguments 是要代入查询的值的数组
+ $options 控制怎样查询操作的选项数组

返回值是临时表的名字

然后你就能用临时表的名字查询临时表

```
$final_result = db_query('SELECT foo FROM ' . $tablename);
```

注意临时表的表名不需要花括号，临时表是暂时性的，不需要通过表前缀处理。与此相反，永久表总是通过用花括号包围来支持表前缀。

注意: Drupal 核心不使用临时表，并且使用这个到数据库连接的 Drupal 数据库用户也可能没有建立临时表的权限，所以模块开发者不要以为运行 Drupal 的每个人都有这个权限。

自己的数据库驱动

假设我们想为一个新的、未来的使用分子计算来提高效率的名叫 DNAbase 的数据库系统写一个数据库抽象层，优于白手起家，我们将拷贝一个存在的抽象层来修改它，我们使用 PostgreSQL 的实现。

首 先，我们拷贝 includes/database/pgsql/database.inc 到 includes/database/dnabase /database.inc，然后，改变逻辑，内部的每一个封装器函数映射到 DNAbase 功能，替换掉原来的 PostgreSQL 功能。

我们在 Drupal 中测试连接到 DNAbase 数据库，在 settings.php 中使用 $databases。

查看 http://drupal.org/node/310087 获取更多的创建自己的数据库驱动的细节。

小结

读完这一章，你应该可以

+ 理解 Drupal 的数据库抽象层
+ 执行基本查询
+ 从数据库中获取单一和多个结果
+ 获取有限范围的结果
+ 使用分页器
+ 理解 Drupal 模式 API
+ 写出其它开发者能够修改的查询
+ 干净地从其它模块修改查询

+ 连接到多个数据库，包括原始数据库

+ 写一个抽象层驱动

用 户是使用 Drupal 的原因，Drupal 能帮助用户创建、合作、沟通和塑造一个在线社区。在本章回顾场景并看一下用户怎样授权、登录及内部表现。我们开 始一个练习，$user 对象是什么、它的结构是怎样的，然后我们演练用户注册、登录、用户授权的过程。最终我们练习 Drupal 绑定外部授权系统如 LDAP 和 Pubcookie。

$user 对象

为了用户登录，Drupal 需要用户激活 cookie，关掉 cookie 的用户在 Drupal 中还可作为匿名用户（anonymous）。
当系统自举处理的会话阶段，Drupal 建立了一个全局$user 对象，它表示当前用户身份。如果用户没登录（并且没有会话 cookie），那么他或她被当做一个匿名用户。这段代码建立一个匿名用户看起来如下（在 includes/bootstrap.inc 中）：

```
function drupal_anonymous_user($session = '') {
  $user = new stdClass();
  $user->id = 0;
  $user->hostname = ip_address();
  $user->roles = array();
  $user->roles[DRUPAL_ANONYMOUS_RID] = 'anonymous user';
  $user->session = $session;
  $user->cache = 0;
  return $user;
}
```

其它情况，如果用户当前是登录的，$user 对象则是通过用户 ID 连接 user 表、roles 和 sessions 表来建立的两个表的所有字段值都放置到$user 对象中。

注意：用户 ID 是个整数，在用户注册或通过管理员创建用户时分配，ID 是 user 表的主键。

$user 对象可以简单通过在 index.php 中用 global $user; print_r($user)来查看。下面是一个登录用户的$user 对象的情形：

```
stdClass Object (
  [uid] => 1
  [name] => admin
  [pass] => $S$CnUvfOYdoxl/Usy.X/Y9/SCmOLLY6QldrzjF7EOWOfR4LG7rCAmR
  [mail] => joe@example.com
  [theme] =>
  [signature] =>
```

```
  [signature_format] => 0
  [created] => 1277957059
  [access] => 1278254230
  [login] => 1277990573
  [status] => 1
  [timezone] =>
  [language] =>
  [picture] => 0
  [init] => joe@example.com
  [data] =>
  [sid] => 8cnG9e0jsCC7I7IYwfWB0rmRozIbaLlk35IQGN5fz9k
  [ssid] =>
  [hostname] => ::1
  [timestamp] => 1278254231
  [cache] => 0
  [session] => batches|a:1:{i:3;b:1;}
  [roles] => Array (
    [2] => authenticated user
    [3] => administrator )
}
```

下面是$user 对象的成分:

成分	说明
user 表提供的	
uid	用户的 ID，user 表的主键，在一个 Drupal 安装中是唯一的
name	用户的用户名，用户登录时输入的
pass	用户密码 安全为 sha512 哈希表
mail	用户当前 email 地址
theme	此字段已弃用，为兼容而保留
signature	用户签名，用在用户评论时，只有当评论模块激活后才可以浏览
signature format	用户签名格式，如 filtered text full text
created	用户账号建立的时间-unix 时间戳
access	用户最后访问时间-unix 时间戳
login	用户最后成功登陆的时间-unix 时间戳
status	用户被锁定为 0，良好为 1
timezone	用户时区与 GMT 相距的秒数
language	用户默认语言，为空，除非多语言呗激活
picture	用户账号图片的路径
init	用户注册时提供的初始 email 地址
data	能被模块存储到这的任何数据（见下节）

成分　　说明

---

user_roles 表提供的

---

roles　　当前分配给此用户的角色

sessions 表提供的

---

sid　　由 PHP 分配给这个用户会话的会话 ID
Ssid　　由 PHP 分配给这个用户会话的安全会话 ID
hostname　　用户浏览当前页面时从哪个 ip 地址来的
timestamp　　一个 unix 时间戳，表现用户浏览器最后就收到一个完整页面的时间
cache　　每个用户缓存的时间戳
session　　在用户的会话期间可以由模块存到这里打任意的、暂短的数据

检测用户是否登录

在一个请求之间，检测用户是否登录的标准方式是测试$user->uid 是不是 0。为此目的，Drupal 有个便利的函数叫 user_is_logged_in()，这是一个相当的 user_is_anonymous()函数：

```
if (user_is_logged_in()) {
  $output = t('User is logged in.');
else {
  $output = t('User is an anonymous user.');
}
```

用户钩子介绍

用户钩子的实现给你的模块一个机会来对执行在用户账号上的不同操作做出反应及修改 $user 对象，这是一些 hook_user 的变形，每个变形执行不同的动作：

Hook function　　目的

---

hook_username_alter(&$name, $account)　　改变用户显示的用户名
hook_user_cancel($edit, $account, $method)　　用户账号取消时的动作
hook_user_cancel_methods_alter(&methods)　　修改账号取消时的方法
hook_user_categories()　　检索用户设置或 profile 信息变化列表
hook_user_delete($account)　　响应用户删除
hook_user_insert(&$edit, $account, $category)　　用户账号建立时
hook_user_load($users)　　当从数据库中载入 user 对象时的动作
hook_user_login(&$edit, $account)　　用户登录

hook_user_logout($account)　　用户登出
hook_user_operations()　　增加一块用户操作
hook_user_role_delete($role)　　通知其它模块用户角色已删除
hook_user_role_insert($role)　　通知其它模块用户角色已增加
hook_user_role_update($role)　　通知其它模块用户角色已更新
hook_user_update(&$edit, $account, $category)　　一个用户账号更新时
hook_user_view($account, $viewmode)　　用户账号信息显示时
hook_user_view_alter(&$build)　　用户创建时，模块可以修改结果信息

警告：不要混淆许多用户钩子中的参数$account 和全局$user 对象，$account 参数是当前操作的用户对象，而全局$user 对象是指当前登录的用户，二者通常，但不总是一样。

理解 hook_user_view($account, $view_mode)

hook_user_view()通常用来在用户 profile 页面增加显示信息（例如你在 http://example.com/?q=user/1 看到的，见图 6-1）。

Figure 6-1. The user profile page, with the blog module and the user module implementing hook_user_view() to add additional information

让我们练习 blog 模块怎样在页面上用 hook_user_view()增加信息：

```
/**
 * Implements hook_user_view().
 */
function blog_user_view($account) {
  if (user_access('create blog content', $account)) {
    $account->content['summary']['blog'] = array(
      '#type' => 'user_profile_item',
      '#title' => t('Blog'),
```

```
      '#markup' => l(t('View recent blog entries'), "blog/$account->uid",
array('attributes' = array('title' => t("Read !username's lastest blog entries.",
array('!username' => format_username($account))))),
      '#attributes' => array('class' => array('blog')),
    )
  }
}
```

view 函数存储一些信息到$user->content，用户 profile 信息组织进 categories，每个分类呈现一个关于用户信息的页面，在图 6-1，这只有一个分类，叫 History。外面的数组应该按分类名称键化，在前面的例子，键名是 summary，对应于 History 分类（十分明确， 它应该制造出键名和分类是相同东西的感觉）。里面的一些数组应该有一个唯一的文本键（这里是 blog）并且要 有#type、#title、#markup、#attributes 元素。类型 user_user_profile_item 指示 Drupal 主题布局 到 modules/user/profile-item.tpl.php。比较代码片段和图 6-1，你能看见这些元素是怎么渲染的，列印图 6-1 的$user->content 数组的内容，它使页面展示成图 6-1 那样：

```
Array
(
    [#pre_render] => Array
        (
            [0] => _field_extra_fields_pre_render
        )
    [#entity_type] => user
    [#bundle] => user
    [#attached] => Array
        (
            [css] => Array
                (
                    [0] => modules/field/theme/field.css
                )
        )
    [summary] => Array
        (
            [blog] => Array
                (
                    [#type] => user_profile_item
                    [#title] => Blog
                    [#markup] => View recent blog entries
                    [#attributes] => Array
                        (
                            [class] => Array
                                (
                                    [0] => blog
                                )
                        )
```

```
            )
        [#type] => user_profile_category
        [#attributes] => Array
            (
                [class] => Array
                    (
                        [0] => user-member
                    )
            )
        [#weight] => 5
        [#title] => History
        [member_for] => Array
            (
                [#type] => user_profile_item
                [#title] => Member for
                [#markup] => 3 days 11 hours
            )
    )
[user_picture] => Array
    (
        [#markup] =>
        [#weight] => -10
    )
)
```

你 的模块还可以实现 hook_user_view()来在$user->content 送到主题之前操作$user->content 的 profile 项目。下面的例子简单从用户 profile 页面删除 blog profile 项目，假设这个模块叫 ide.module

```
/**
 * Implements hook_user_view().
 */
function hide_user_view($account, $view_mode = 'full') {
  unset($account->content['summary']['blog']);
}
```

用户注册流程

依 照默认，用户在 Drupal 站点上注册除了一个用户名和一个有效的 e-mail 地址外不需要更多的信息。通过实现几个用户钩子模块能增加它们自己的字段到 用户注册表单，让我们写一个模块 legalagree.module，它提供一个快速的方式来使你的站点适应今天的喜好诉讼的社会。

首先，建立目录 sites/all/modules/custom/legalagree，增加下面的文件到这个 legalagree 目录，然后通过 Administer->Site building->Modules 激活这个模块。

legalagree.info

```
name = Legal Agreement
description = Display a dubuious legal agreement during user registration.
package = Pro Drupal Development
core = 7.x
files[] = legalagree.module
```

legalagree.module

```
<?php
/**
 * @file
 * Support for dubious legal agreement druring user registration.
 */

/**
 * Implements hook_form_alter().
 */
function legalagree_form_alter(&$form, &$form_state, $form_id) {
  // check to see if the form is the user registration or user profile form
  // if not then return and don't do anything
  if (!($form_id == 'user_register_form' || $form_id == 'user_profile_form')) {
    return;
  }

  // add a new validate function to the user form to handle the legal agreement
  $form['#validate'][] = 'legalagree_user_form_validate';

  // add a field set to wrap the legal agreement
  $form['account']['legal_agreement'] = array(
    '#type' => 'fieldset',
    '#title' => t('Legal agreement')
  );

  // add the legal agreement radio buttons
  $form['account']['legal_agreement']['decision'] = array(
    '#type' => 'radios',
    '#description' => t('By registering at %site-name, you agree that at any time,
we (or our surly, brutish henchmen) may enter your place of residence and smash your
belongings with a ball-peen hammer.', array('%site-name' =>
variable_get('site_name', 'drupal'))),
    '#default_value' => 0,
    '#options' => array(t('I disagree'), t('I agree'))
  );
```

```
}

/**
 * Form validation handler for the current password on the user_account_form()
 *
 * @see user_account_form()
 */
function legalagree_user_form_validate($form, &$form_state) {
  global $user;

  // Did user agree?
  if ($form_state['input']['decision'] <> 1) {
    form_set_error('decision', t('You must agree to the Legal Agreement before
registration can be completed.'));
  } else {
    watchdog('user', t('User %user agreed to legal terms', array('%user' =>
$user->name)));
  }
}
```

用户钩子在注册表单建立期间、表单校验期间及用户记录插入到数据库之后得到调用，我们简单的模块的结果在一个注册表单中看起来应该像图 6-2。

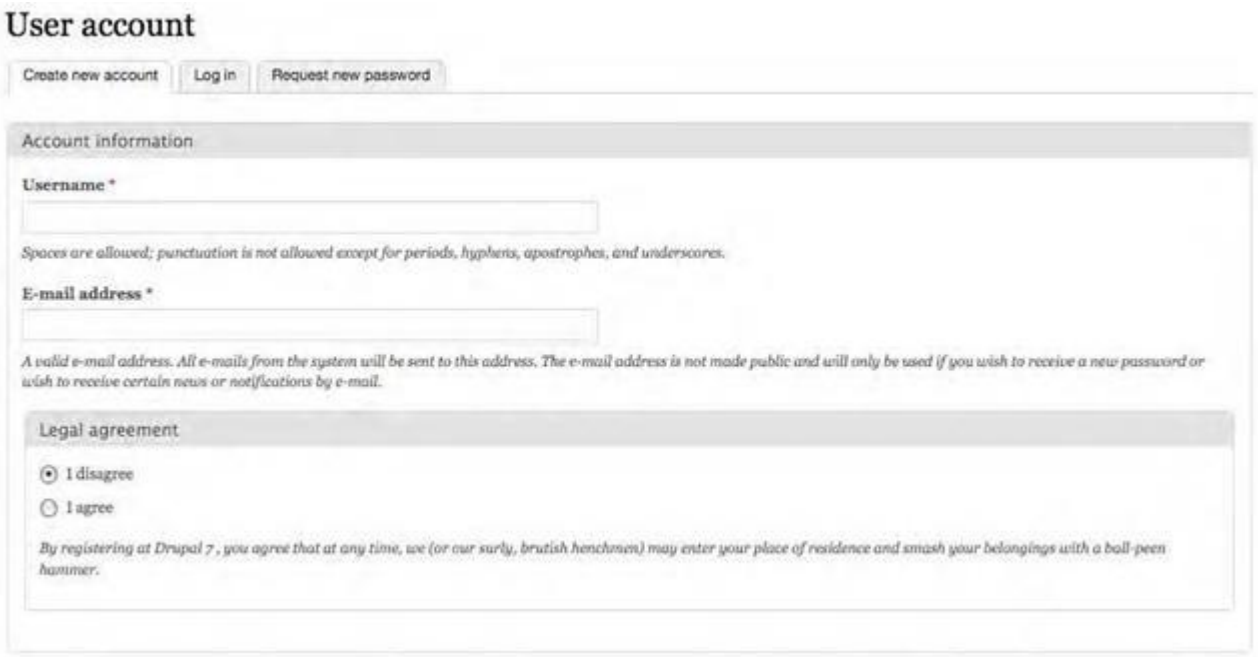

Figure 6-2. A modified user registration form

使用 profile 模块收集用户信息

如 果你计划扩展用户注册表单来收集关于用户的信息，你应该在写自己的模块之前试一试 profile.module 模块，它允许你建立任意的表单去收集数据， 定义用户注册表单哪些信息是否必须收集，设计哪些信息是公用还是私有，此外它允许管理员去定义页面，那样用户就能选择一个 URL 来访问它们的 profile 了，URL 结构 site URL 加 profile/加 profile 字段名字加值。
例如，如果你定义一个文本 profile 字段叫 profile_color，你能浏览喜欢黑色的那些用户 http://example.com/?q=profile/profile_color/black，或者假设你建立一个会议站点，负责出席者的晚餐计划，你能定义一个复选框 profile 字段名叫 profile_vegetarian，你就能通过羡慕的 URL 来看全部的素食者 http://example.com/?q=profile/profile_vagetarian（注意是复选框字段，值是暗示的因此是忽略的；所以不像上课例子的 black，这里没有值附加到 URL 末尾）。
一个真实世界的例子，参加 2010 旧金山 Drupal 大会的用户列表能用 profile/conference-sf-2010 访问到（这里，字段的名字没有前缀 profile_）。

TIP：profile 的概要页面的自动创建只有在 profile 设置表单填写了 page title 字段后才起作用，并且不可用于文本、URL、日期字段。

登录流程

登录处理开始于用户填上登录表单（典型的是 http://example.com/?q=user 或显示在一个区块中）并且点击了“Log in”按钮。
表单的校验例程检查用户名是否锁定、是否访问角色不允许访问、是否用户输入的用户名或密码无效，有这些状况时，用户都能得到及时通知。

注意：Drupal 有本地和外部授权，外部授权例子有 OpenID、LDAP、Pubcookie 及其它。

Drupal 通过在 users 表中查找出一行匹配用户名和密码 hash 的记录来实现登录本地用户。一个成功的登录结果引发两个用户钩子（login 和 load），你的模块可以实现他们如图 6-3。

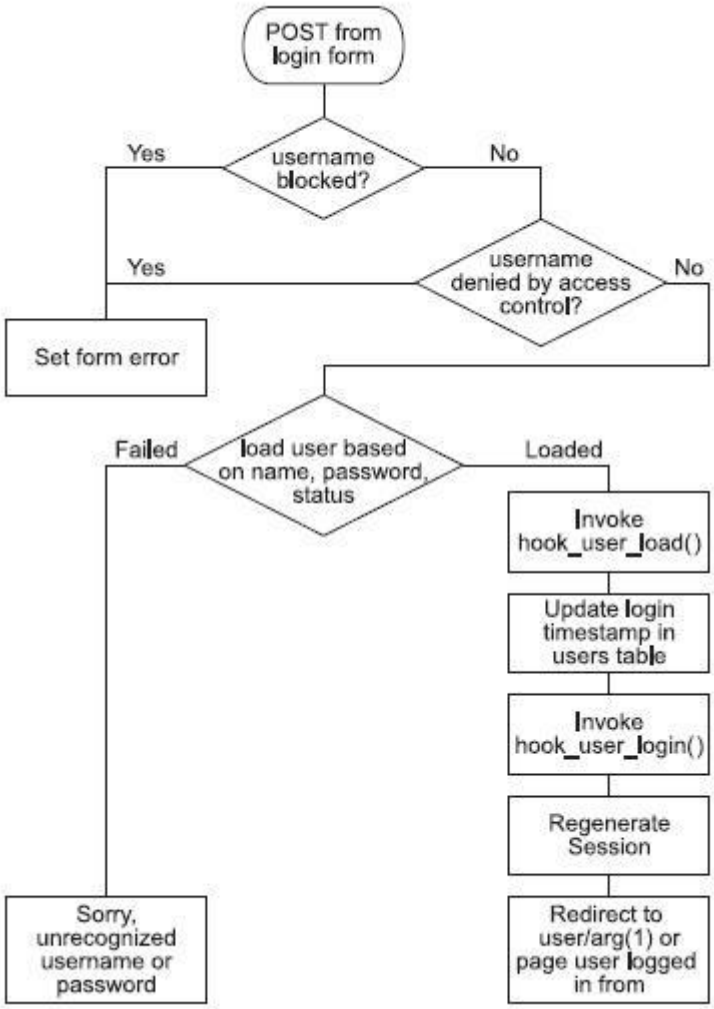

Figure 6-3. Path of execution for a local user login

在载入时附加数据到$user

用户钩子的 load 操作是在$user 对象成功从数据库载入时在响应调用 user_load()内触发，这发生在一个用户登录、为一个节点抽取授权信息及其它几个点。

注意：因为调用用户钩子是昂贵的，user_load()在当前$user 对象为一个请求实例化时是不调用的（查看$user 节），如果你写你自己的模块，你要永远在调用一个期望完全载入$user 对象的函数之前调用 user_load()，除非你确认这些已经发生。

让我们写一个模块叫 loginhistory 来保存用户登录历史，我们将在用户的“My account”页面显示用户登录过的次数。在 sites/all/modules/custom/创建目录 loginhistory，并且增加下面几个文件：

loginhistory.info

```
name = Login History
```

```
description = Keeps track of user logins
package = Pro Drupal Development
core = 7.x
files[] = loginhistory.install
files[] = loginhistory.module
```

我们需要一个.install 文件来建立数据库表存储登录信息：

loginhistory.install

```
<?php
/**
 * Implements hook_schema().
 */
function loginhistory_schema() {
  $schema['login_history'] = array(
    'description' => 'Stores infomation about user logins.',
    'fields' => array(
      'uid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => 'The {user}.uid of the user logging in.',
      ),
      'login' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => 'Unix timestamp denoting time of login.',
      ),
    ),
    'indexes' => array('uid' => array('uid')),
  );
  return $schema;
}
```

loginhistory.module

```
<?php
/**
 * @file
 * Keep track of user logins.
 */

/**
```

```
 * Implement hook_user_login
 */
function loginhistory_user_login(&$edit, $account) {
  // insert a new record each time the user log in
  $nid = db_insert('login_history')->fields(array(
        'uid' => $account->uid,
        'login' => $account->login
  ))->execute();
}

/**
 * Implements hook_user_view_alter
 */
function loginhistory_user_view_alter(&$build) {
  global $user;

  // count the number of logins for the user.
  $login_count = db_query("SELECT count(*) FROM {login_history} where uid = :uid",
array(':uid' => $user->uid))->fetchField();
  // update the user page by adding the number of logins to the page
  $build['summary']['login_history'] = array(
    '#type' => 'user_profile_item',
    '#title' => t('Number of logins'),
    '#markup' => $login_count,
    '#weight' => 10,
  );
}
```

然 后安装这个模块，每个成功的用户登录都会引发 hook_user_login 的 login 操作，作为响应，模块在数据库的 login_history 表中 插入一条新记录，用户访问“My account”页面，当$user 对象在 hook_user_view 期间装入，hook_user_view_alter 函数将触发，模块将在页面上 增加用户的登录成功次数如图 6-4。

Figure 6-4. Login history tracking user logins

提供用户信息分类

如 果你在 drupal.org 上有一个账号，你能在 My account 页面上看到提供的用户信息分类的效果了，然后选择 Edit 选项卡，去编辑你的账号信息，诸如密码，也可以提供关于你自己的几个其它分类信息 如 Drupal involvement、个人信息、工作信息及是否接收新闻。

外部登录

有时，你可能不想使用 Drupal 本地 users 表，例如你可能有一个用户的表在其它数据库或在 LDAP 内，Drupal 很容易将外部授权整合进登录流程。
让 我们实现一个是分简单的外部授权模块来描画外部收取如何工作。假设你的公司只聘任一个人叫 Dave，并且用户名是基于 first name 和 last name 分配的，这个模块授权任何以 dave 开头的字符串为用户名的用户登录，如 davebrown、davesmith、davejones 将成功登 录。我们的途径将用 form_alter()去改变用户登录校验例程，变为运行我们自己的校验例程。这是 sites/all/modules /custom/authdave/authdave.info：

```
name = Authenticate Daves
description = External authentication for all Daves.
package = Pro Drupal Development
core = 7.x
files[] = authdave.module
```

这是 authdave.module

```
<?php

/**
```

```
 * Implements hook_form_alter().
 * We replace the local login validation handler with our own.
 */
function authdave_form_alter(&$form, &$form_state, $form_id) {
  // In this simple example we authenticate on username to see whether starts with
dave
  if ($form_id == 'user_login' || $form_id == 'user_login_block') {
    $form['#validate'][] = 'authdave_user_form_validate';
  }
}

/**
 * Custom form validate function
 */
function authdave_user_form_validate($form, &$form_state) {
  if (!authdave_authenticate($form_state)) {
    form_set_error('name', t('Unrecognized username.'));
  }
}

/**
 * Custom user authentication function
 */
function authdave_authenticate($form_state) {
  // get the first four characters of the user name

  $username = $form_state['input']['name'];
  $testname = drupal_substr(drupal_strtolower($username), 0, 4);

  // check to see if the person is a dave
  if ($testname == 'dave') {
    // if it's a dave then user the external login_register function
    // to either log the person in or create a new account if that
    // person doesn't exist as a Drupal user
    user_external_login_register($username, 'authdave');
    return TRUE;
  } else {
    return FALSE;
  }
}
```

在这个 authdave 模块（见图 6-5）我们简单地将第二个校验例程换为我们的，比较图 6-5 和图 6-3，图 6-3 处理的是本地登录流程。

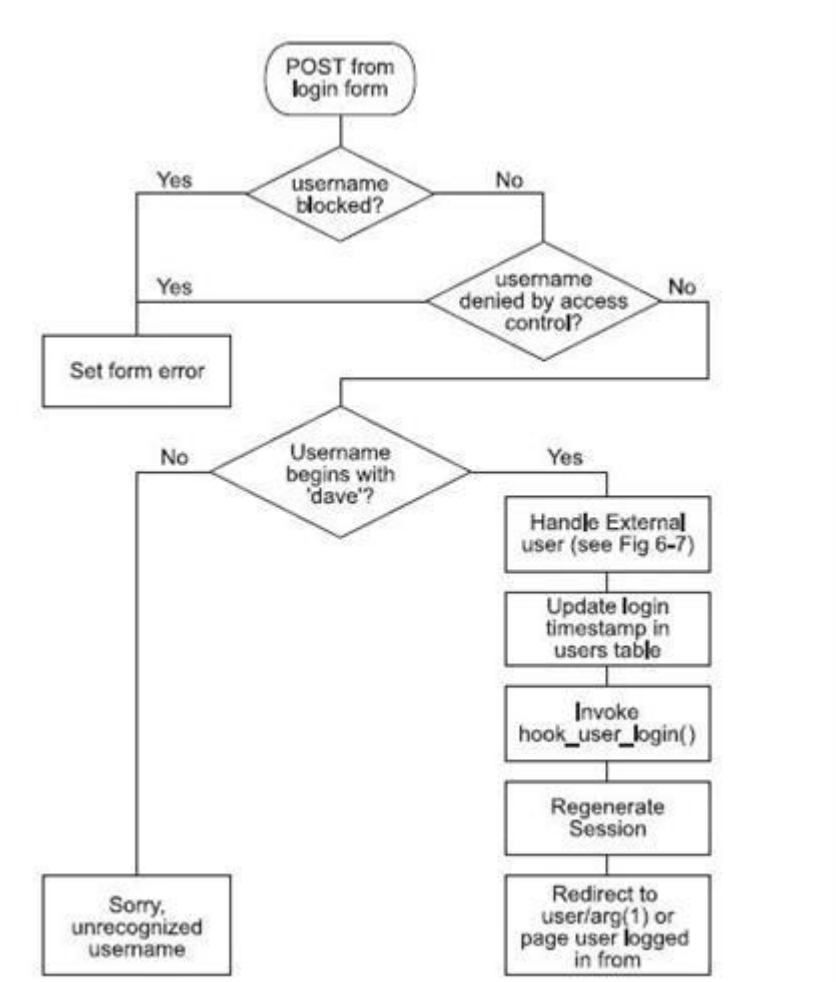

Figure 6-5. Path of execution for external login with a second validation handler provided by the authdave module (compare with Figure 6-3)

函数 user_external_login_register()是帮助函数，在用户第一次注册这个用户并记录他的登录。用户 davejones 的假象登录路径显示为图 6-6。

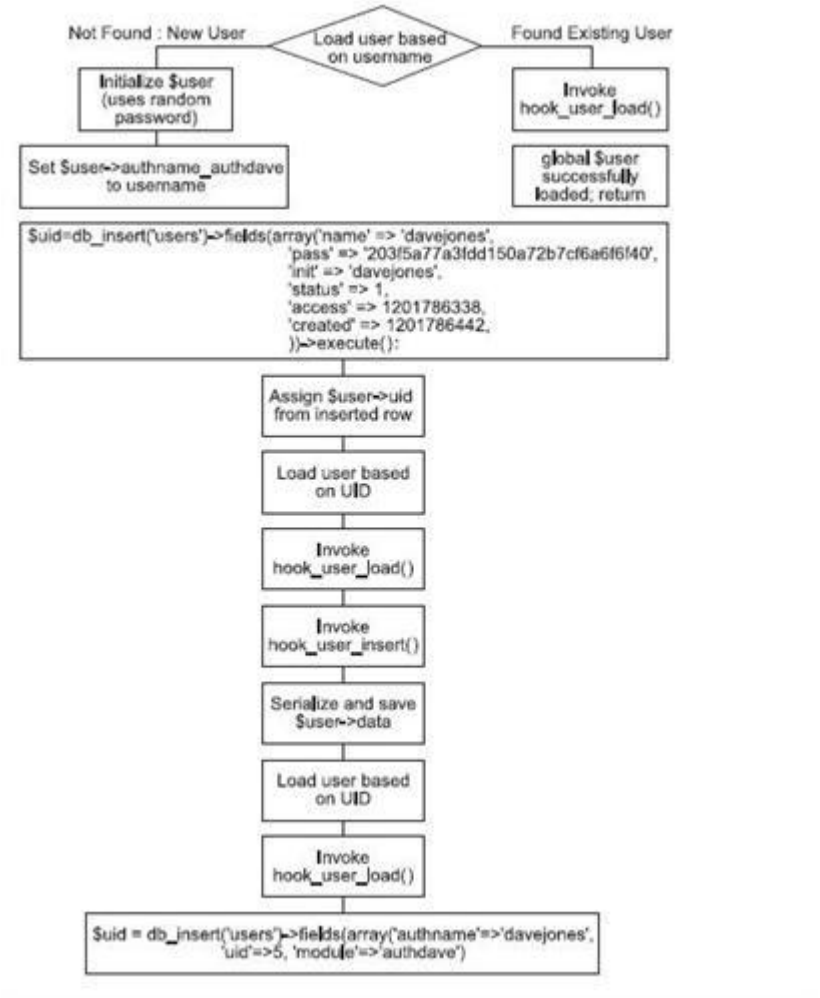

Figure 6-6. Detail of the external user login/registration process

如 果用户名以 dave 开始，并且用户头一次登录，user 表中没有这个用户的记录，这个记录将要增加。然而，这里不像 Drupal 默认的本地用户注册那样提 供有效的 e-mail 地址，如果你的站点真的要给用户发邮件，这样的一个模块只是简单的而非真的解决方案。你可能想设置 users 表的 email 列，那样 就将有一个与用户关联的 e-mail 地址，要做到这些，你能有自己的模块响应用户钩子的 insert 操作它将在一个新用户插入时触发：

```
/**
 * Implements hook_user_insert().
 */
function authdave_user_insert(&$edit, &$account, $category = NULL) {
  global $authdave_authenticated;
  if ($authdave_authenticated) {
    $email = mycompany_email_lookup($account->name);
    // set e-mail address in the users table for this user.
    db_update('users')
      ->fields(
        array(
          'mail' => $email,
```

```
        )
      )
      ->condition('uid', $account->uid)
      ->execute();
  }
}
```

已经理解的读者可能注意到：这里没有办法告诉用户是否是本地用户还是外部授权，我们聪明地保存一个全局变量包含我们模块做过的授权，我们能用如下方式查询 authmap 表：
db_query("SELECT uid FROM {authmap} WHERE uid = :uid AND module = :module",
array(':uid' => $account->uid, ':module' => 'authdave'));
所有通过外部授权的用户在 authmap 表中都有一个记录，就像 users 表，然而，在这种情形授权和 hook_user_insert 运行于相同的请求，一个全局变量相较于数据库查询是个好的选择。

小结

读完本章，你能够去：

+ 理解在 Drupal 内部用户是如何呈现的
+ 理解怎样用几种方法存储用户相关数据
+ 钩进用户注册流程来从注册中的用户获取更多信息
+ 钩进用户登录流程以在用户登录时运行自己的代码
+ 理解外部授权用户怎样工作
+ 实现你自己的外部授权模块

更多的外部授权信息情况 openid.module 模块（Drupal 核心一部分）或贡献模块 pubcookie.module。

本 章将介绍节点和节点类型，我将展示怎样用两种不同方式创建一个节点类型，首先展示用 Drupal 钩子通过写一个模块来程序化地创建一个节点类型，这种情况 在决定一个节点能干和不能干什么的时候有更大程度上的控制权和柔性。然后我将展示怎么样从 Drupal 管理员接口创建一个节点类型，最后我们将研究 Drupal 节点访问控制机制。

TIP：开发者经常使用术语 node 和 node type，在 Drupal 用户接口中，它们分别对应 posts 和 content type，在术语的使用上我们尽量同管理员保存一致。

节点具体是什么

Drupal 的开发新手问的首要问题就是：”node 是什么？“，一个节点是一个内容片，Drupal 为每个内容片指定一个 ID 号码叫节点 ID（代码中简写为$nid），通常每个节点还有个标题，允许管理员查看一个标题的列表。

注意：如果你熟悉对象导向，可以将节点类型想象为一个类，而个体节点想象为一个对象的实例，尽管 Drupal 不是 100%的对象导向，一个好的原因在这里（http://api.drupal.org/api/HEAD/file/development/topics/oop.html）。

有 许多不同 j 节点类别，就是节点类型。一些常用的节点类型是”blog entry“ ”poll“和”forum“。有时术语内容类型（content type）与节点类型（node content）是同义词，尽管一个节点类型是真正的抽象概念并且被认为源自节点，就像图 7-1 的描述。

Figure 7-1. Node types are derived from a basic node and may add fields.

所 有的内容类型作为节点的最美好的地方是它们基于相同下层数据结构。对于开发者，这意味着你能让许多操作程序化地对待所有内容，它能简单地在节点上执行批量 操作并且你还能创造行动获得大量自定义内容类型的功能。因为下层节点数据结构和行为，Drupal 天生支持检索、创建、编辑和管理内容，这些也一致地呈献 给终端用户。创建、编辑和删除节点的表单看起来和感觉上非常相似，成为一个一致的因而更加易用的接口。

节 点类型通常通过增加它自己的数据属性来扩展基本节点。节点类型 poll 存储投票的选项，不管当前投票是否可用，也不管是否允许用户投票；forum 节点类 型为每个节点载入分类，那样它就知道它适合放在管理员定义的论坛的哪里啦；blog 节点例外，不增加任何其他的数据，它们只通过为每个用户建立 blogs 和为每个 blog 建立 Rss 来增加不同的视图到数据。所有节点都有下列存储在 node 和 node_revisions 表中的属性：

+ nid：节点的唯一的 ID

+ vid：节点的唯一的版本 ID，需要它是因为 Drupal 能为每个节点存储内容版本

+ type：每个节点都有一个节点类型——例如，blog、story、article、image 等等

+ language：节点的语言，开箱即用这个列是空的，包括语言中立的节点

+ title：短于 255 个字符的字符串，作为节点的标题，除非节点类型

声明节点没有标题，

通过将 node_type 的 has_title 字段设成 0

+ uid：作者的用户 ID，默认节点有一个单个的作者

+ created：节点建立时的 Unix 时间戳

+ changed：节点最后修改时的 Unix 时间戳，如果你使用节点版本系统，

node_revisions 表中的 timestamp 也用相同的值

+ comment：一个整数字段，描述节点的评论状态，有 3 个可能的值：

+ 0：节点不允许评论 在 comment 模块禁用是，这是默认值

+ 1：当前节点不允许有更多的评论，在用户接口的“Comment settings”表单中

相当于“read only”

+ 2：评论可见、用户可以添加新的评论，控制评论权限和是否可见是在 comment 模块中，在

在用户接口的“Comment settings”表单中相当于“read/write”

+ promote：一个整型字段决定是否在 front 页面上展示这个节点，有两个值：

+ 1：提升到 front 页面，这个节点将提升到你的站点的默认首页，正常页面也展示它

例如 http://example.com/?q=node/3，首页这个叫法不恰当，实际应该是

http://example.com/?q=node 页面，它包含所有 promote 为 1 的节点，这个页面是

默认的首页

+ 0：节点不显示在 http://example.com/?q=node 上

+ sticky：当 Drupal 在一个页面上显示一个节点列表时，默认是先显示标记为 sticky 的那些节点，

然后是按照建立的时间顺序显示节点，1 代表 sticky，同一个列表中可以有多个标记

为 sticky 的节点

+ tnid：当一个节点是作为另一个节点的翻译版本的时候，源节点的 nid 就存储在这，例如

节点 3 是英语、节点 5 是与节点 3 同内容的瑞典语，节点 5 的 tnid 字段就是 3

+ translate：值是 1 指示翻译需要更新，0 表示谁翻译是新的

如果你使用节点版本系统，Drupal 将建立内容版本可以跟踪谁最后编辑

不是所有东西都是节点

用 户、区块 、评论不是节点，它们有自己特殊的数据结构，有自己的钩子系统来实现自己的目的。节点（通常）有标题和主体内容，表现一个用户的数据结构不需要这些，相 反，它需要一个 e-mail 地址、一个用户名、一个安全存储的密码；区块是一个为一些诸如菜单导航、搜索框、最新评论列表等小片内容的轻量级存储解决方 案；评论也不是节

点，使它们保持轻量级是正确的，每页上完全可能有 100 个或更多的评论，如果这些评论中的每个评论在载入时都执行一遍节点钩子系统，那么 这将是对性能的巨大打击。

过去，对用户和评论是否应该是节点存在大量争论，并有贡献模块真正实现了这个，警告不要过度关注这个论点，这就像在一个编程集会上大喊“Emacs 是最好的”一样。（Emacs 和 Vi 都行，个人习惯，不是核心问题，反而容易引起不必要的争论，要关心真正的核心问题。）

创建一个节点模块

传 统上，当你想建立一个新的节点类型是你要写一个节点模块，它承担提供你的节点类型需要的新的感兴趣的东西，我们说“传统上”是因为最近 Drupal 框架允 许你在管理员接口建立新的节点类型并利用贡献模块扩展它们的功能，这比白手起家写代码容易，本章我们将涵盖这两种解决方案。

我要写个模块，让用户 增加一个工作帖子到一个站点，一个工作帖子节点应该包括一个标题、一个含有工作细节的主体和一个能输入公司名字的字段。对于标题和主体，我将使用内建的节 点 title 和 body，也是 Drupal 所有节点的标准，我需要为公司名称增加一个新的自定义字段。开始时在 sites/all/modules/custom 目录建一个文件夹 job_post。

创建.install 文件

job post 模块的 install 文件执行所有的 setup 操作，像定义节点类型、建立组成我们节点类型的字段、处理卸载过程等。

```
<?php
/**
 * @file
 * Install file for Job Post module
 */

/**
 * Implements hook_install().
 * - Add the body field
 * - Configure the body field
 * - create the company name field
 */
function job_post_install() {
  node_type_rebuild();
  $types = node_type_get_types();
  // add the body field to the node type
  node_add_body_field($types['job_post']);
  // load the instance definition for our content type's body
  $body_instance = field_info_instance('node', 'body', 'job_post');
  // Configure the body field
  $body_instance['type'] = 'text_summary_or_trimmed';

  // save our changes to the body field instance.
  field_update_instance($body_instance);

  // create all the fields we are adding to our content type.
  foreach (_job_post_installed_fields() as $field) {
    field_create_field($field);
  }

  // Create all the instance for our fields.
  foreach (_job_post_installed_instances() as $instance) {
    $instance['entity_type'] = 'node';
    $instance['bundle'] = 'job_post';
    field_create_instance($instance);
  }
}

/**
```

```
 * Return a structured array defining the fields create by content type.
 * For the job post module there is only one additional field - the company
name
 * Other fields could be added by defining them in this function as
additional elements
 * in the array below
 */
function _job_post_installed_fields() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'label' => $t('Company posting the job listing'),
      'type' => 'text',
    ),
  );
}

/**
 * Return a structured array defining the field instances associated with
this content type.
 */
function _job_post_installed_instances() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'type' => 'text',
      'label' => $t('Company posting the job listing'),
      'widget' => array(
        'type' => 'text_textfield',
      ),
      'display' => array(
        'example_node_list' => array(
          'label' => $t('Company posting the job listing'),
          'type' => 'text',
        ),
      ),
    ),
  );
}

/**
 * Implements hook_uninstall()
```

```
 */
function job_post_uninstall() {
  // Gather all the example content that might have created while this
  // module was enabled.
  $sql = 'SELECT nid FROM {node} n WHERE n.type = :type';
  $result = db_query($sql, array(':type' => 'job_post'));
  $nids = array();
  foreach ($result as $row) {
    $nids[] = $row->nid;
  }

  // Delete all the nodes at once
  node_delete_multiple($nids);

  // Loop over each of the fields defined by this module and delete
  // all instance of the field, their data, and the field itself.
  foreach (array_keys(_job_post_installed_fields()) as $field) {
    field_delete_field($field);
  }

  // Loop over any remaining field instance attached to the job_post
  // content type (such as the body field) and delete them individually.
  $instances = field_info_instances('node', 'job_post');
  foreach ($instances as $instance_name => $instance) {
    field_delete_instance($instance);
  }

  // Delete our content type
  node_type_delete('job_post');

  // Purge all field infromation
  field_purge_bacth(1000);
}
```

创建.info 文件

让我们还建立一个.info 文件放到 job_post 目录

```
name = Job Post
description = A job posting content type
package = Pro Drupal Development
core = 7.x
files[] = job_post.install
```

```
files[] = job_post.module
```

创建.module 文件

最后，我们需要模块文件自己，在 sites/all/modules/custom/job_posting 目录中建一个叫 job_post.module 的文件，在你完成这个模块后，你可以在 Modules 页面激活它，开始你应该在文件中写下 PHP 的 tag：

```
<?php
/**
 * @file
 * This module provides a node type called job post
 */
```

提供关于我们节点类型的信息

现在你准备增加一些钩子到 job_post.module，第一个钩子我们想实现的是 hook_node_info()，Drupal 在它搜寻哪些类型是激活的时执行这个钩子，你需要为你的自定义节点提供某些元数据。

```
/**
 * Implements hook_node_info() to provide our job_post type.
 */
function job_port_node_info() {
  return array(
    'job_post' => array(
      'name' => t('Job Post'),
      'base' => 'job_post',
      'description' => t('Use this content type to post a job.'),
      'has_title' => TRUE,
      'title_label' => t('Job Title'),
      'help' => t('Enter the job title,
                   job description, and the name of the company that
posted the job'),
    ),
  );
}
```

单个一个模块可以定义多个节点类型，返回值应该是一个数组。这是可以在 node_info()钩子中支持的分解的元数据值：

+ name：节点类型的人类可读名字，必需

+ base：base 字符串用于构造符合节点类型的回调（例如，如果 base 是 example_foo，那么 example_foo_insert 将在

插入这个类型的一个节点时调用）这个值通常但不总是模块名，必需

+ description：节点类型的简要描述，必需

+ help：当建立一个此类型的节点是显示的帮助信息，可选

+ has_title：布尔值，指示这个节点类型是否有标题字段，默认为 TRUE，可选

+ title_label：标题字段的标签，默认为 Title，可选

+ locked：布尔值，指示管理员是否可以此类型的机器名，FALSE 可以改变（未锁）TRUE 不可改变，默认为 TRUE，可选

注意：在前面的列表 base 中提到的内部的 name 字段，用于构建“Create content”连接的 URL，例如，我们用 job_post 做我们节点类型的内部名（它是我们返回数组的键），那么去建立一个新的 job_post，用户将转到 http://example.com/?q=node/add_job_post。通常将 locked 设成 FALSE 使内部名可修改不是个好主意。内部名存储在 node 和 node_revirsions 表的 type 列。

修改菜单回调

要在“Create content”页面上有一个连接，不需要事先 hook_menu()，Drupal 自动搜寻新内容类型并将其加到到 http://example.com/?q=node/add 页面，如图 7-2 所示，一个到节点提交表单的连接将出现在 http://example.com/?q=node/add/job_post 中，名称和描述将从你定义的 job_post_node_info()中取值。

如果你不希望增加这个字节连接，你能用 hook_menu_alter()删除它，例如，下面代码能为所有那些没有“administer nodes”权限的人删除这个页面。

```
/**
 * Implements hook_menu_alter()
 */
function job_post_menu_alter(&$callbacks) {
  // If the user does not have 'administer nodes' permisssion,
  // disable the job_post menu item by setting its access callback to
FALSE.
  if (!user_access('administer nodes')) {
    $callback['node/add/job_post']['access callback'] = FALSE;
    // Must unset access arguments or Drupal will use user_access()
    // as a default access callback.
    unset($callback['node/add/job_post']['access arguments']);
  }
}
```

用 hook_permission()定义权限

典 型的模块定义节点类型的权限包括建立此类型节点、修改你建立的节点、修改任何人建立的这个类型节点的能力，这是在 hook_permission()中作 为 create job_post、edit own job_post、edit any job_post 等等定义

的，你可以在你的模块中定义这些权限，让我们现在使用 hook_permission()建立这些权限：

```
/**
 * Implements hook_permission().
 */
function job_post_permission() {
  return array(
    'create job_post' => array(
      'title' => t('Create a job post'),
      'description' => t('Create a job post'),
    ),
    'edit own job post' => array(
      'title' => t('Edit own job post'),
      'description' => t('Edit own job post'),
    ),
    'edit any job post' => array(
      'title' => t('Edit any job post'),
      'description' => t('Edit any job post'),
    ),
    'delete own job post' => array(
      'title' => t('Delete own job post'),
      'description' => t('Delete own job post'),
    ),
    'delete any job post' => array(
      'title' => t('Delete any job post'),
      'description' => t('Delete any job post'),
    ),
  );
}
```

现在，如果你导航到 People 点击 Permission 选项卡，你定义的新的权限就在那里并准备分配给用户角色。

用 hook_node_access()限制节点类型的访问

你 在 hook_permission()中定义了权限，但是它们是怎么实施的呢？节点模块能使用 hook_node_access()来限制访问它们定义的 节点类型。超

级用户（ID 为 1）允许通过任何访问检查，这种情况下钩子不调用。如果没有为你的节点类型定义钩子，那么所有的访问检查将失败，那么将只有有 “administer nodes”权限的超级用户才可以建立、编辑、删除这个类型的内容。

```
/**
 * Implements hook_node_access().
 */

function job_post_node_access($op, $node, $account) {
  $is_author = $account->uid == $node->uid;
  switch($op) {
    case 'create':
      // Allow if user's role has 'creat joke' permission.
      if (user_access('create job', $account)) {
        return NODE_ACCESS_ALLOW;
      }
    case 'update':
      // Allow if user's role has 'edit own joke' permission and user is
      // the author; or if the user's role has 'edit any joke' permission.
      if (user_access('edit own job', $account) && $is_author ||
user_access('edit any job', $account)) {
        return NODE_ACCESS_ALLOW;
      }
    case 'delete':
      // Allow if user's role has 'delete own joke' permission and user
is
      // the author; or if the user's role has 'delete any joke'
permission.
      if (user_access('delete own job', $account) && $is_author ||
user_access('delete any job', $account)) {
        return NODE_ACCESS_ALLOW;
      }
  }
}
```

上 面的函数允许用户去建立一个 job post，如果他的角色中含有“create job post”权限，如果角色含有“edit own job post”权限且是此帖子的作者

或者他们有“edit any job post”权限，他们还能更新一个 job post；如果他们有“delete own job post”权限则可以删除自己的帖子，如果有“delete any job post”权限，则可以删除任意 job post 类型的节点。

我 们传给 hook_node_access()的参数$op 的另一个值是 view，允许你去控制谁可以看到这个节点，作为警 告，hook_node_access()只是为单独的节点视图页面调用，hook_node_access()将不能阻止某些人从强制节点视图查看这个 节点，比如一个多节点列表页面。你能通过创造性地使用其它钩子和操作$node->teaser 值来直接克服这些，但是这只是一个小技巧，最好解决 方案是使用 hook_node_grants()，我们过会再说。

为我们的节点类型自定义表单

到 此为止，我们为我们的新节点类型定义了元数据定义了访问权限，下一步，你需要建立节点表单以使用户能输入一个 job。你可以通过实现 hook_form()来做到这些，Drupal 提供一个包含标题、主体、和其它任何你定义的可选字段的标准节点表单，对于 Job post 内容类型，标准表单相当合适，那么我们用它来渲染 add/edit 表单。

```
/**
 * Implement hook_form() with the standard default form.
 */
function job_post_form($node, $form_state) {
  return node_content_form($node, $form_state);
}
```

注意：如果你不熟悉 form API，请看第 11 章。

作为站点管理员，如果你激活你的模块，你能通过浏览 Add content -> Job Post 来看见刚建立的表单（图 7-3）

当你使用一个节点表单而不是一个通用表单工作时，node 模块处理校验和保存所有它知道的节点表单的默认字段（诸如标题、主体字段），也提供钩子给开发者来校验和保存你的自定义字段，下步我们将说明这些。

用 hook_validate()校验字段

当 你的节点类型的一个节点被提交，你的模块将通过 hook_validate() 调用，因此，当用户提交表单来建立一个 job post，函数 job_post_validate() 使你能校验在你自定义字段的输入，你能在提交后用 form_set_value() 来改变数据，错 误由 form_set_error()来设置，见下面：

```
/**
 * Implements hook_validate().
 */
function job_post_validate($node) {
  // Enforce a minimum character count of 2 on company names.
  if (isset($node->job_post_company) &&
    strlen($node->job_post_company['und'][0]['value']) < 2) {
    form_set_error('job_post_company',
                   t('The company name is too short. It must be alteast
2 char.'),
                   $limit_validation_error = NULL);
  }
}
```

注意你已经在 hook_node_info()中定义了主题最少字符数，Drupal 将自动校验，尽管如此，punchline 字段是一个你增加的扩展字段，那么你需要响应它的校验和载入、存储。

我 将赞助人的信息拆分进独立的主题函数中，这个主题函数可以轻易被覆写，这是对过度工作的管理员的一种礼貌态度，因为他可能使用你的模块，但是他不想定制输 出的外观和感觉。为激活这个特性，我将建立一个 hook_theme()函数，它将定义模块怎样处理新的赞助人字段，在 hook_thmem()函数中， 我将定义分配给赞助人字段的变量和模板文件，此文件将用来定义怎样将赞助人信息作为节点的一部分渲染。

```
/**
 * Implements hook_theme().
 */
function job_post_theme() {
  // define the variables and template associated with the sponsor field
  // The sponsor will contain the name of the sponsor anr the sponsor_id
  // will be used to create a unique CSS ID
  return array(
    'sponsor' => array(
      'variables' => array('sponsor' => NULL, 'sponsor_id' => NULL),
      'template' => 'sponsor',
    ),
  );
}
```

流程的最后一步是建立为赞助商信息建立模板文件，在 hook_theme()函

数中我分配一个值 sponsor 给 template 属性--然后在我的模块目录建立一个 sponsor.tpl.php 文件，文件内容入下：

```
<?php

/**
 * @file
 * Default theme implementation for rendering job post sponsor infomation
 *
 * Available variables:
 * - $sponsor_id: the node ID associated with the job posting
 * - $sponsor:    the name of the job post sponsor
 */
?>
  <div id="sponsor-<?php print $sponsor_id; ?>" class="sponsor">
    <div class="sponsor-title">
      <h2>Sponsored by</h2>
    </div>
    <div class="sponsored-by-message">
      Thie job posting was sponsored by: <?php print $sponsor; ?>
    </div>
  </div>
```

你 需要去清理主题注册缓存来使 Drupal 看起了是你主题钩子的样子，你可以是我弄个 devel.module 或浏览 Modules 页面来清理缓存。你现在拥有了全功能的 job post 条目和展示系统，进到以前的某些 jobpost 试试有什么东西，你能看到你的帖子有了些格式，如同图 7-4 和 7-5。

用 hook_node_xxxxx()来操作非我们类型的节点

前 面的钩子只是基于模块实现的 hook_node_info()的 base 键才调用，当 Drupal 看到一个 blog 节点类型，blog_load()就被 调用。如果你想去增加某些信息到每个节点，不管它是什么类型，该如何？到目前为止，我们学的钩子都不能胜任这项工作，为此，我们需要一个特别的强大的钩子 集合。

node_xxxx 钩子为模块创造一个机会，使之在任意一个节点的的生命周期对不同操作做出反应，node_xxxx 钩子通常由 node.module 只在 node-type-specific 回调调用之后才调用，这里是主要的 node_xxxx 钩子列表：

+ hook_node_insert($node)：响应节点的建立。

+ hook_node_load($node,$type)：起作用在一个节点从数据库载入的时候。$node 是键化数组，$type 是载入的类型的键化数组。

+ hook_node_update($node)：响应一个节点的更新

+ hook_node_delete($node)：响应一个节点的删除

+ hook_node_view($node,$view_mode)：起作用在节点开始渲染的时候，$view_mode 是显示模式，例如 full 活 teaser

+ hook_node_prepare($node)：起作用在节点刚要显示在 add/edit 表单时

+ hook_node_presave($node)：起作用在节点开始插入或更新

+ hook_node_access($node, $op, $account)：控制节点的访问，$op 是将要执行的操作类型（如 insert、update、view、delete）

$account 是执行这些操作的用户的账号

+ hook_node_grants_alter(&$grants, $account, $op)：在试图查看、编辑或删除一个节点时，改变用户访问授权

在显示一个节点页面如 http://example.com/?q=node/3 时，钩子的触发顺序如图 7-6.

节点如何存储

在 数据库中节点是分开存储的，node 表包含大多数描述节点的元数据。node_revisions 表包含节点的主体和摘要，连同版本信息。并且你可以在 job_post.module 看到，在节点载入时，其它节点可以自由地向节点中追加数据而且可以将任何想要的数据存储在自己的表中。

如图 7-7，一个节点对象包含大多数公共属性，注意有 fieldAPI 建立的用于存储 job post 公司的表是与主要节点表分开的，依赖于什么模块被激活，你的 Drupal 节点对象可能包含或多或少的属性。

通过管理员界面来创建节点类型

尽管使用模块来创建一个节点类型的方式提供了更多的控制与效能，但是它太复杂了，难道就没有一个比编程创建节点类型更好的方式了吗，Drupal 为你提供了核心的自定义节点类型功能。

通 过管理接口你能增加一个节点类型（就像你的模块 job_post 一样），在 Structure->Content types，如果你的模块 job_post 激活了，你要确保节点类型有个不同的名字，避免命名空间冲突。在 job_post.module 例子中，你需要 三个字段：job title、job description（节点主体）和发布帖子的公司名字 name。在 job_post.module 中，你不得不手工增加 body 字段和 name of company 字段，使用核心自定义节点类型功能，通过用户接口你能排除所有编程任务，Drupal 核心处理一切建表任务及插入、更新、删除、访问控制和 查看节点。

限制节点访问

有几种方式来限制对节点的访问。你已经看到怎样用 hook_access()来控制访问节点及使用 hook_permission()来定义权限，但是 Drupal 用 node_access 表提供了一个更加丰富的访问控制集，还有两个访问钩子: hook_node_grants()，hook_access_records()。

当 Drupal 最初安装时，一个单个记录写进了 node_access 表，作用为关闭 Drupal 节点访问机制，只有当一个使用节点访问机制的模块激活 时，Drupal 这部分才开始生效，在 modules/node/node.module 中的函数 node_access_rebuild()保持对哪个 节点访问模块激活的跟踪，如果这些

模块禁用，那么这个函数恢复默认记录，表 7-2：

Table 7-2. The Default Record for the node_access Table

nid	gid	realm	grant_view	grant_update	grant_delete
0	0	all	1	0	0

通常，一个节点访问模块被使用（就是修改 node_acccess 表的那个），Drupal 将允许访问节点，除非节点访问模块将一个记录插入 node_access 表，来定义怎样对待访问控制。

定义节点授权

节 点上的操作有 3 个基本权限：view、update 和 delete，当这些操作中的某个将要发生，提供节点类型的模块首先通过它的 hook_access()的实现说话，如果模块没有为访问允许提供位置（就是那个地方为 NULL，而不是 TRUE 或 FALSE），Drupal 询问所有 对节点访问感兴趣的模块来回应是否操作必须被允许的问题，它们靠响应 hook_node_grants()来做这 些，hook_node_grants()带有一个当前用户每个领域的授权 ID 列表。

什么是领域（Realm）

Realm 是一个独特的字符串，它允许多个节点访问模块去共享 node_access 表，例如，acl.module 是一个贡献模块，利用访问控制列表来控制节点访 问，它的 realm 是 acl。另外的贡献模块还有

taxonom_access.module，它基于 taxonomy 分类来约束对节点的访问，它是用 term_access 作为 realm。那么，realm 就是一些在 node_access 表中指示你的模块空间的东西，它有点像命名空间，当你的模块返 回授权 ID，你要为你的模块定义 realm。

什么是授权 ID

一 个授权 ID 是一个标示符，为一个给定的领域（realm）提供关于节点访问权限信息，例如一个节点访问模块 forum_access.module，它通 过用户规则来管理到 forum 节点的访问——可以使用用户规则 ID 作为授权 ID，一个通过美国邮政编码来管理节点访问的模块可以使用邮政编码作为授权 ID。在每一种情况中，它都得确定一些用户的东西：他是否被分配了这个用户规则或者他的邮编是否是 12345？或者用户是否在访问控制列表中，或者用户是 否已经订阅了一年以上？

尽 管每个授权 ID 对节点访问模块（这些模块从包含这个授权 ID 的领域来获得授权 ID）来说意味着特别的东西，这只不过表现为 node_access 表中有一 行包含这个授权 ID 来许可访问，带有访问类型，此类型有 grant_view、grant_update、grant_delete 列的值是不是 1 来决 定。
当一个节点保存时，授权 ID 就被插入 node_access 表。每一个实现 hook_node_access_records()的模块都传 递一个节点对象。模块预计测试一下节点并简单返回（如果它不为节点处理访问）或者为 node_access 表的插入返回一个授权数组。下面是 node_access_example.module 的一

个例子，在这个模块中，hook_node_access_records()检视节点是否是 私有——如果是，那么授权设置成只能查看，第二个授权检视用户是否为作者——如果是，那么设置所有授权（查看、更新、删除）。

```
function hook_node_access_records($node) {
  // We only care about the node if it has been marked private. If not,
it is
  // treated just like any other node and we completely ignore it.
  if ($node->private) {
    $grants = array();
    $grants[] = array(
      'realm' => 'example',
      'gid' => 1,
      'grant_view' => 1,
      'grant_update' => 0,
      'grant_delete' => 0,
      'priority' => 0,
    );
    // For the example_author array, the GID is equivalent to a UID, which
    // means there are many many groups of just 1 user
    $grants[] = array(
      'realm' = 'example_author',
      'gid' => $node->uid,
      'grant_view' => 1,
      'grant_update' => 1,
      'grant_dalete' => 1,
      'priority' => 0,
    );
    return $grants;
  }
}
```

节点处理流程

当一个操作要在一个节点上执行，Drupal 按照下面的流程轮廓顺序执行，

如图 7-8：

小结

读完本章，你可以：

+ 理解什么是节点什么是节点类型
+ 写建立节点类型的模块
+ 理解怎样钩入节点的建立、保存、载入等等
+ 理解怎样访问节点的判定

一 个 field 通常是 Drupal 中的一个成分，用来存储一个值，记录用户登录或一个模块建立。字段的信息在数据库中校验、存储，可以从数据库中抽取并显示 在网站上。字段的例子包括用户名、街道地址、电话号码、价格、一个或两个描述事件的段落、一个日期或其他任何你想象的信息片段。
在 Drupal 7 中，建立字段有了人人瞩目的改变——Field API 加到了核心里。过去为了校验、存储、抽取、显示字段层信息而定义表、写代码是个乏味冗长的任务，现在处理这些通过一个 Field API 集合。许多字段层特性都加入了 Drupal7 的核心，来自 Drupal6 的 CCK 模块，CCK 模块提供一个 UI 层接口来在 Drupal 先前版本建立 字段。
在本章中，我将展示标准的建立到 Drupal7 核心的类 CCK 功能，怎样通过增加一个新的字段类型来扩展功能，任何一个站点管理员都可以讲这个新的字段类型附加到内容类型中，最后展示怎样在一个模块中用 Field API 通过几个不同的字段类型来建立一个新的内容类型。

创建内容类型

Drupal 有个杀手级应用，就是可以创建自定义内容类型，自定义内容类型被定义为创建一个节点的框架，内容类型典型地包含一些字段，最少一个 title 字段、一个 body 字段和几个其它用来获取结构化信息的字段。例如一个 event 自定义内容类型，event 有一些字段来获取、存储、显示信息，诸如事件名字、事件 描述（body）、事件的日期和时间、事件的位置。让我们通过后台来建立一个新的 event 内容类型：Structure->Content Type 点击 Add content type 连接。建立一个新的内容类型相对简单——输入内容类型要呈现的名称的值（此

例中是 Event）几个简短的描述，以及可选的覆盖 Event 节点标题 的标签，此例中我们将标签从 Title 改到 Event Title（见图 8-1）。

Figure 8-1. Main page for creating a content type

伴 随“Submission form settings”值定义完，下一步是修改发布选项去给你的站点的请求加地址。点击“Publishing options”选项卡来 check/uncheck 选项，我们不选“Promoted to front page”选项，默认是选择。
在“Display settings”表单，我们不选择在 Event 显示时是否显示作者信息。我们不需要看见是谁发布的事件。
“Comment settings”表单控制一个内容类型的评论如何显示，在 Evenet 这个情况，不需要评论，我们设置“Default comment setting for new content”为 Hidden。
在“menu settings”表单，我们将“Main Menu”复选框选掉，不在页面上显示。
随 着内容类型的大部分配置选项的设置，我现在已经准备好保存 Event 内容类型并且进行到处理流程的下一步——增加字段（见图 8-2）。在你点击“Save and add fields”按钮后，Drupal 带你到这个页面，在此你可以为 Event 内容类型增加新的字段。

Figure 8-2. The page for adding fields
在增加字段之前，我们要确定事件时间是否用纯文本形式，还是用作者建立事件时可用的弹出式日历形式，在大多数情况下，你可能想最后做这些，就像日期通常有其它的使用目的，像确定在日历上哪里放置一个内容项目、以用户所在地的格式来格式化日期或进行日期计算。Date 模块（http://drupal.org/project/date）提供一个可以在我们的 Event 内容类型中使用的字段，它可以弹出一个日历让你选择日期。那么，在我们增加字段处理之前，按标准安装方式安装 Date 模块。

为一个内容类型增加字段

两个为 Event 内容类型增加的字段是事件的定位和事件的日期/时间。我们从定位字段开始，在标签那输入 Event Location，字段名称那里输入 event_location，我们选择“text field”组件来作为这个字段的存储的数据类型（如图 8-3）

Figure 8-3. Adding the Event Location field to the Event content type
点击 Save 按钮来时事件定位字段的文本字段的长度设定出现，我们保留默认值 255 个字节，然后点击“Save field settings”按钮，下一个表单（图 8-4）显示事件定位字段配置选项的细节。下 一个值的设置允许你去覆写我们用图 8-3 输入的标签的值，通过点击选中为必须域，意味着作者建立新的事件时必须在这个字段输入值，设置文本字段的物理长 度，就是显示在屏幕上的长度，是否作者有权限改变输入过滤，帮助文本将显示在输入字段的下方，默认值是当在屏幕上渲染这个表单时分配给该字段的值。

Figure 8-4. Field settings for the Event Location field

最后设置的值是字段值的数量或基数以及事件定位字段能输入的最大字符数（见图 8-5），我们保留默认值为 1，就是事件只能有一个定位，并且我保留可输入最大长度为 255 个字符，点击“Save Settings”按钮返回如图 8-3 事件定位字段增加到字段列表的表单。

Figure 8-5. Setting the cardinality and maximum number of characters for the field

下 一步是增加 Event 日期字段。我将使用同建立 Event Location 字段一样的处理程序。将 Event Date and Time 作为标签，字段名为 event_date_time，选择 Datetime 作为数据存储类型，选择带有弹出式日历的 Text Field 作为在屏幕上显示的样式， 点击 Save 按钮引出 date 字段的字段设置页面，如图 8-6，除了“Time zone handling”我们都使用默认值，我们将“Time zone handling”设置为“No time zone conversion”，由作者输入的时间就是我想在站点上显示的时间。

Figure 8-6. The Field Settings page for the Event Date and Time field

然 后你点击“Save field settings”按钮， Event 设置页面显示出来以设置日期字段，在这个页面上，我可以覆写我先前输入的标签，选择此字段是否是必须（日期和时间在我 的站点上是事件必须的），帮助文本将显示在此字段的饿下面，默认显示的是在日期模块中设置的日期格式，如果作者没有选择一个值，则使用默认值，输入格式定 义在输入表单中日期的输入顺序和格式，前面的年份和后面的年份将显示在弹出日历中，以及分钟的递增值（事件发生间隔一小时、半小时、一刻钟，你将这个值分 别改成 60,30 或 15），你也有能力去设置你能建立的值的数量——我们只需要调用一个值，来确定起止时间，就是日期和时间的间隔尺度，你还可以覆写 Drupal 是否可以转换输入的时间，当它基于不同的选项显示的时候。对于 Event 日期，我将保留所有的值为默认值，然后点击“Save settings”按钮，现在我有了 Event 内容类型所有的字段，并且开始授权 Events 使用节点创建表单，如图 8-7。

Figure 8-7. Creating a new Event

注意：在本书写成时，Date 模块通过修订直接加到 Drupal 核心，我建议你查看 http://drupal.org/project/date 来看看配置一个日期字段的途径或表单的更新。

创建一个自定义字段

Drupal7 核心自带的几个通用字段类型能满足大多数不同的目的，你能用前面的字段类型为广大不同的目的去捕获、存储和显示值，但是，这有许多可能是标准的字段类型不 是你需要的，这些可以使 Field API 起作用、激活自定义字段的建立，它用来在你的站点上建立任何的字段类型。

Table 8-1. Standard Field Types in Drupal 7 Core

field type	usage
Boolean	用于在复选框或单选框中收集真假值
Decimal	用于收集包含小数点的十进制数字
File	提供一个文件上传字段允许作者附加一个文件到你的内容类型的一个实例
Image	提供一个图片上传字段允许作者附加一个图片到你的内容类型的一个实例
Integer	提供一个文本字段使作者能在此输入一个整数
List	提供一个建立选择列表（弹出、或 select 值列表）的
List（numeric）	或复选框/单选框按钮使作者可以选择一个或多个预先
List（text）	提供的值
Long text	提供一个多行文本区使作者可以在此输入信息
Long text and summary	
Term reference	提供选择 taxonomy term 的能力
Text	一个简单的文本框

借助 Field API，你有能力去构建一个自定义字段满足几乎任何类型的你能想到的数据输入。一个例子，我们将使用 Drupal.org 上的字段类型例子（http://api.drupal.org/api/drupal/developer--examples--color_example--col...），它定义一个自定义字段，在字段类型设置字段时，将文本渲染成颜色。我们使用这个字段类型来捕获和显示事件参与者出席未来事件应该穿什么颜色。

第一步在 sites/all/modules/custom 中建立一个目录叫 color_example，在此目录中建立新文件 color_example.info，输入下面内容：

```
name = Color Example

description = "Create a custom field for inputting and displaying text in a colorful fashion"

package = Pro Drupal Development

core = 7.x

files[] = color_example.module

php = 5.2
```

下一步建立其它文件名叫 color_example.module，放入下面内容

```
/**
 * @file
 * An example field using the Field API
 *
 */
```

保存文件，激活这个模块。我们已经准备好构造新的 RGB 字段类型细节了。
第一步调用 hook_field_info()，它定义我们的新字段的基本属性，我们定义字段为 color_example_rgb()并且分配一个标签、描述、默认组件和默认格式到新的字段类型。

```
/**
 * Implements hook_field_info().
 *
 * Provides the description of the filed
 */
function color_example_field_info() {
  return array(
    'color_example_rgb' => array(
      'lable' => t('Example Color RGB'),
      'description' => t('Demonstrates a field composed of an RGB color.'),
      'default_widget' => 'color_example_3text',
      'default_formatter' => 'color_example_simple_text',
    ),
  );
```

```
}
```

下 一步定义在此字段收集的数据怎样在 Drupal 数据库中存储，在 D7 和 Field API 之前，我们要在模块中自己定义表和图表来定义我们的内容类型和内容类型中用到的自定义字段，在 D7 中，利用 Field API，这个任务由 hook_field_schema()为我们处理，例如，我们存储一个用于在屏幕上渲染文本的描述 HTMLhex 颜色代码的 7 字符字 段——例如使用#FF0000 渲染红色文本。在此例中，我们建立一个单个的 column 存贮由站点管理员输入的 RGB 的值，假如他把这个字段分配给一个内 容类型的话。

```
/**
 * Implements hook_field_schema().
 */
function color_example_field_schema($field) {
  $columns = array(
    'rgb' => array('type' => 'varchar', 'length' => 7, 'not null' => FALSE),
  );
  $indexes => array(
    'rgb' => array('rgb'),
  );
  return array(
    'columns' => $columns,
    'indexes' => $indexes,
  );
}
```

下 一步是用 hook_field_validate()校验用户的输入，我要告诉 Drupal 用户输入的值匹配一个典型 HTML 颜色代码模式，我将检视第一 个字符是不是“#”并且下面跟随的是不是数字或一个在 a-f 之间的字符，如果输入的值不匹配这个模式，我将显示一个错误。

```
/**
```

```
 * Implements hook_field_validate().

 *

 * Verifies that the RGB field as combined is valid

 * (6 hex digits with a # at the beginning).

 */

function color_example_field_validate($entity_type, $entity, $field, $instance,
$langcode, $items, &$errors) {

  foreach ($items as $delta => $item) {

    if (!empty($item['rgb'])) {

      if (!preg_match('@^#[0-9a-f]{6}$@', $item['rgb'])) {

        $errors[$field['field_name']][$langcode][$delta][] = array(

          'error' => 'color_exmple_invalid',

          'message' => t('Color must be in the HTML format #abcdef.'),

        );

      }

    }

  }

}
```

下一个函数定义什么构成了这个类型的空字段，在这种情况，我们使用 PHP 空函数返回 TRUE 或 FALSE 二者之一，这依赖于此字段是否为空。

```
/**

 * Implements hook_field_is_empty().

 */
```

```
function color_example_field_is_empty($item, $field) {
  return empty($item['rgb']);
}
```

1.字段格式器是一个函数，定义字段的内容是怎样显示的，钩子 hook_field_formatter_info() 函数标识格式器的类型，在我们的例子中她用于显示文本和背景。

```
/**
 * Implements hook_field_formatter_info().
 */
function color_example_filed_formatter_info() {
  return array(
    // this formatter just display the hex value in the color indicated.
    'color_example_simple_text' => array(
      'label' => t('Simple text-based formatter'),
      'field types' => array('color_example_rgb').
    ),
    // This formatter changes the background color of the content region.
    'color_example_color_background' => array(
      'lable' => t('Change the background of the output text'),
      'field types' => array('color_example_rgb'),
    ),
  );
}
```

下面我将为这两个格式器建立可渲染输出，两个格式器是这样定义的：

1. color_example_simple_text 只输出标记指示颜色，此颜色是输入的并使用 inline 风格来设置文本颜色为此值。

2. color_example_color_background 作用相同但是改变 div.region-content 的背景颜色。

```
/**

 * Implements hook_field_formatter_view().

 */

function color_example_filed_formatter_view($entity_type, $entity, $field, $instance, $langcode, $items, $display) {

  $element = array();

  witch ($display['type']) {

    // This formatter simply output the field as text and with a color.

    case 'color_example_simple_text':

      foreach ($items as $delta => $item) {

        $element[$delta]['#markup'] = '<p style="color:' . $item['rgb'] . '">' . t('The color for event is @code', array('@code' => $item['rgb'])) . '</p>';

      }

      break;

    // This formatter adds css to the page changing the '.region-content' area's

    // background color. If there are many fields, the last one will win.

    case 'color_exmple_color_background':

      foreach ($items as $delta => $item) {

        drupal_add_css('div.region-content { background-color:' . $item['rgb'] . ';}', array('type' => 'inline'));

        $element[$delta]['#markup'] = '<p>' . t('The color for this event has been changed to @code', array('@code' => $item['rgb'])) . '</p>';
```

```
      }
      break;
  }
  return $element;
}
```

下一个函数定义校验用户的输入数据

```
/**
 * Validate the individual fields and the convert them into a single HTML RGB
 * value as text.
 */
function color_example_text_validate($element, &$form_state) {
  $delta = $element['#delta'];
  $field =
$form_state['field'][$element['#field_name']][$element['#language']]['field'];
  $field_name = $field['field_name'];
  if (isset($form_state['values'][$field_name][$element['#language']][$delta]))
{
    $value = $form_state['values'][$field_name][$element['#language']][$delta];
    foreach (array('r', 'g', 'b') as $colorfield) {
      $val = hexdec($values[$colorfield]);
      // If they left any empty, we'll set the value empty and quit.
      if (strlen($values[$colorfield) == 0) {
        form_set_value($element, array('rgb' => NULL), $form_state);
```

```
        return;
      }
      // If they gave us anything that's not hex reject it.
      if ((strlen($values[$colorfield]) != 2) || $val <0 || val > 255) {
        form_error($element[$colorfield], t('Saturation value must be a 2-digit hex value between 00 and ff.'));
      }
    }
    $value = sprintf('#%02s%02s%02s', $values['r'], $values['g'], &values['b']);
    form_set_value($element, array('rgb' => $value), $form_state);
  }
}
```

最后，我用 hook_field_error()来在用户输入无效时显示错误信息。

```
/**
 * Implements hook_field_error().
 */
function color_example_field_widget_error($element, $error, $form, $form_state) {
  switch ($error['error']) {
    case 'color_example_invalid':
      form_error($element, $error['message']);
      break;
  }
```

```
}
```

下一个文件是 Javascript 文件，提供一个 farbtastic 颜色选择器，在这个目录中建立 color_example.js，包含下列代码：

```
/**
 * @file
 * Javascript for Color Example.
 */
/**
 * Provide a farbtastic colorpicker for the fancier widget.
 */
(function($) {
  Drupal.behaviors.color_example_colorpicker = {
    attach: function(context) {
      $(".edit-field-example-colorpicker").live("focus", function(event) {
        var edit_field = this;
        var picker = $(this).closest('tr').find(".field-example-colorpicker");
        // Hide all color picker except this one.
        $(".field-example-colorpicker").hide();
        $(picker).show();
        $.farbtastic(picker, function(color) {
          edit_field.value = color;
        }).setColor(edit_field.value);
      });
```

```
    }

  }

})(jQuery);
```

最后一个文件是本模块必须的 CSS 文件，建立一个叫 color_example.css 的新文件包含下面 CSS 代码：

```
/**

 * @file

 * CSS for Color Example

 */

div.form-item table .form-type-textfield,

div.form-item table .form-type-textfield * {

  display: inline-block;

}
```

保存完这个模块之后，字段已经准备好去加到一个内容类型里啦，通过 Structure->Content Types 来为 Event 增加 color 字段，点击 Event 内容类型的 Manage Fields 选项卡将展示如图 8-8 情形：

Figure 8-8. Adding the new Event Color field

然 后点击 Save 按钮，带到下一个字段设置页面，它给我们展示并没有任何一个字段设置分配给 Event COlor 字段（例如 maximum length），点击“Save setting”按钮到 Event 的可覆写设置页面（如图 8-9）。在这个表单中，我将输入帮助文本，显示在表单的该字段下面，输入 color 字段的默认 值，点击“Save Setting”完成 Event 节点编辑表单的增加字段处理流程。

Figure 8-9. Setting the configuration options for the new Event Color field

随着字段增加到 Event 内容类型，我已经准备去测试它的输出。导航到 Add Content->Event 保留新的 Event 的 RGB color 字段的默认值（如图 8-10）。

Figure 8-10. The new Event RGB Color field on the node edit form

然后保存 Event，新的字段显示的格式如同定义在模块的格式器函数的那样，图 8-11 用节点建立时定义的颜色显示文本，这个例子中我输入的是红色。

Figure 8-11. The new field is displayed in the color defined by the author.

目前为止，我展示了怎样使用标准的 Drupal 字段类型去建立一个新内容类型和怎样去建立一个能够加到内容类型的新字段类型，下面我将展示怎样在一个模块中使用 Field API 去程序化地增加一个字段。

程序化增加字段

Field API 能被用来程序化地增加字段到一个内容类型或节点类型，下面的例子演示使用 Field API 来增加一个新字段到一个由模块创建的内容类型。Job Post 模块创建内容类型，它扩展自常规节点（title 和 body），增加了一个新的字段，用来存储和显示发布工作信息的公司的名称，增加字段的程序发 生在模块的.install 文件的 hook_install()和 hook_update()函数中。

在 hook_install()函数中第一步是增加 body 字段到我们的新的 Job Post 内容类型，默认情况下，通过模块建立的内容类型只包含 title 字段，你必须隐含的增加 body 字段。node.module 定义了一个名叫 node_add_body_field()的函数，用它增加标准的 body 字段到我们的 Job Post 内容类型。下一步我将遍历的是增加所有要增加到 Job Post 内容类型中的字段定义，本例子的情况是，只有一个叫 job_post_companies 的字段需要增加，如果我想增加多个字段，我能通过在 _job_post_installed_fields()函数中增加字段定义来完成。定义新字段简单的事情，给定一个名字、一个标签、和一个字段类型。 建立字段的最后的步骤是使用 field_create_instance()函数实例化字段，这个函数做完所有的幕后工作，诸如在数据库中创建存储机制来 获取用户输入的值，当你看见这个情景，这里不需要再数据库中定义表——Field API 为你做了所有的工作。在你安装模块后，这个字段就在你的节点编辑表单里啦，已准备好为你的模块使用。另外关于怎样使用和主题化自定义字段的输出请看 第 7 章。

```
<?php

/**

 * @file

 * Install file for Job Post module

 */


/**
```

```
 * Implement hook_install().
 *
 * - Add the body field.
 * - Configure the body field.
 * - Create the company name field.
 */
function job_post_install() {
  node_types_rebuild();
  $types = node_type_types();
  node_add_body_field($types['job_post']);

  // Load the instance definition for our content type's body.
  $body_instance = field_info_instance('node', 'body', 'job_post');

  // Add our job_post_list view mode to the body instance display by.
  $body_instance['type'] = 'text_summary_or_trimmed';

  // Save our changes to the body field instance.
  field_update_instance($body_instance);

  // Create all the fields we are adding to our content type.
  foreach (_job_post_installed_field() as $field) {
    field_create_field($field);
  }
```

```
  // Create all the instances for our fields.
  foreach (_job_post_installed_instances() as $instance) {
    $instance['entity_type'] = 'node';
    $instance['bundle'] = 'job_post';
    field_create_instance($instance);
  }
}

/**
 * Return a structured array defining the fields create by this content type.
 */
function _job_post_installed_fields() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'label' => $t('Company posting the job listing'),
      'type' => 'text',
    ),
  );
}
```

```
/**
 * Return a structtured array defining the instances for this content type.
 */
function _job_post_installed_instances() {
  $t = get_t();
  return array(
    'job_post_company' => array(
      'field_name' => 'job_post_company',
      'type' => 'text',
      'label' => $t('Company posting the job listing'),
      'widget' => array(
        'type' => 'text_textfield',
      ),
      'display' => array(
        'job_post_list' => array(
          'label' => $t('Company posting the job listing'),
          'type' => text,
        ),
      ),
    ),
  );
}
```

```
/**
 * Implements hook_uninstall().
 */
function job_post_uninstall() {
  // Gather all the example content that might have been created while this
  // module was enable.
  $sql = 'SELECT nid FROM {node} n WHERE n.type = :type';
  $result = db_query($sql, array(':type' => 'job_post'));
  $nids = array();
  foreach ($result as $row) {
    $nids[] = $row->nid;
  }

  // Delete all the nodes at once
  node_delete_multiple($nids);

  // Loop over each of the fields defined by this module and delete
  // all instances of the field, their data, and the field itself.
  foreach (array_keys(_job_post_installed_fields()) as $field) {
    field_delete_field($field);
  }

  // Loop over any remaining field instances attached to the job_post
```

```
  // content type (such as body field) and delete them individually.
  $instances = field_info_instances('node', 'job_post');
  foreach ($instances as $instance_name => $instance) {
    field_delete_instance($instance);
  }

  // Delete our content type.
  node_type_delete('job_post');

  // Purge all field information
  field_purge_batch(1000);
}
```

小结

在本章中，我涵盖了用 D7 核心功能去建立包含 title、body 之外的附加字段的自定义内容类型、怎样建立一个自定义的字段类型及怎样去程序化地增加一个新字段到一个模块等等内容。在下一章，我们将进入主题化章节，学习如何将可视化风格应用到 Drupal 渲染内容上。

改 变 Drupal 生成的 HTML 或其它标签需要那些组成主题系统层次的知识。在本章中，我将教你主题系统怎样工作并且透露一些隐藏在 Drupal 核心中的最 好的实践。首要问题是你在模块文件中不需要（或称不应该）编辑 HTML 来改变你的站点的视觉外观，要是那样做，你只是建立你自己私有的内容管理系统，并且 因此失去了有社区支持的开源软件系统的内含的最大优势。一定记住：要覆写，而不是修改。

主题

用 Drupal 的话来说，主题就是制造出站点外观和感觉的文件的集合。你可以从 http://drupal.org/project/themes 下 载预先构建好的主题，或者你学完这章后自己建一个。主题由下面一些东西组成，你期望看起来是 web 设计者：样式表、图片、javascript 脚本等等，你需要注意 Drupal 主题和纯的 HTML 站点的不同是针对模板文件，模板文件典型包含大的 HTML 段和由动态内容如 Drupal 构造页面替换的特定的小 片，你可以创建针对性模板文件

来针对每一个 Drupal 内容的容器——诸如 overall page、regions、blocks、nodes、comments 甚至 field。一会我们将遍历结构层模板文件的建立过程，但是先让我们从安装现 成的 Drupal.org 的模板开始并测试构成主体的组成成分。

安装现成的主题

有成百上千的 Drupal 主题可用，如果你寻求简单、快速地布置并运行一个站点，你可以考虑通过 www.drupal.org/project/themes 来浏览主题，确定选择了 7.x，来只显示支持 D7 的主题。

注意：你必须挑选 Drupal7 版本的主题，因为主题结构的变化，D6 及以前版本的主题不能运行于 D7 站点上。

在 你浏览主题时，你通常能遇到所谓的“启动器主题”，启动器主题致力于提供构建一个新主题的实体基础。典型上，启动器主题包含一个丰富的在线文档和功能帮 助。启动器主题的好处在于它提供了实体基础，可以在其上放置图片元素和颜色，而不用从一个空白页面开始。没有归类为启动器主题的主题已经是包含有图形效果 （如图片、颜色、字体等等）应用并且可以通过少量修改来适应你的需要。

作为演示，我们将安装 Pixture Reloaded 主题，除了它已经可以在 D7 上运行之外，此主题没什么重要作用，浏览 drupal.org 的主题页面，复制 pixture_reloaded 主题下载连接的 URL，返回你的站点，在页面顶部点击 Appearance 连接，在外观页面点击 Install new theme 连接，在上传新主题的表单上粘贴下载 URL 到标签为“Install form a URL”的文本框，然后点击 Install 按钮，Drupal 将下载并保存主题到你的 sites/all/themes 目录，然后你可以通过浏览 Appearance 页面并点击“set default”连接激活主题并作为默认。

从 drupal.org 安装主题简单快速，你能通过上述方式下载任意数量的主题并在站点上测试他们，但是很可能有时你想建立自己所有的定制的主题，在下面的章节中，我将展示你怎样通过全新设计和白手起家来创建一个新主题。

创建新主题

有几种方式来创建一个新主题，依赖于你开始的材料，假设你的设计师已经给你了站点的 HTML 和 CSS，讲设计师的 HTML 和 CSS 转换到一个 Drupal 主题，相对来说是比较简单的。通常建立一个新的 Drupal 主题有下列步骤：

+ 1、建立或修改站点的一个 HTML 文件
+ 2、建立或修改站点的一个 CSS 文件
+ 3、建立一个.info 文件告诉 Drupal 你的新主题
+ 4、依据 Drupal 希望那样来标准化文件名
+ 5、向你的模板中插入可用变量
+ 6、给节点类型、区块等等分别建立附件的文件

我 们以确定名字叫“Grayscale”开始来构建新主题——然后用相同的名字在 sites/all/themes

中建立目录（sites/all /themes/grayscale），下一步我们为此主题在目录中建立一个.info 文件，我将建立 grayscale.info 文件，初始时包含合并 到 Drupal 主题注册器中所需要的基本信息

```
name = Grayscale
core = 7.x
engine = phptemplate
```

一 旦你保存了 grayscale.info 文件，你就能激活这个主题，通过点击页面顶部的 Appearance 连接，向下滚动，直到看见 Grayscale 主题，点击“enable and set default“连接将主题应用为站点默认主题，点击 Home 按钮浏览站点主页，你就有个新的 Drupal 主题（见图 9-1），并且你所作的全部工作就是 在.info 文件中输入了 3 行。

Figure 9-1. The site rendered in the Grayscale theme

显 然它不能够赢得任何创新奖，只是遍历了一下看看白手起家创建一个 Drupal 主题是多么简单。让我们给网站扩展一点东西，应用一些 CSS 重排和风格的东 西，第一步在我们的主题 Grayscale 目录中建一个叫 css 的目录，虽然将所有 css 文件放到一个子目录中的做法不是必需的，但这样的好处在于别人不 必挖掘你的主题目录来定位 CSS 文件。在 css 目录，建立一个 style.css 的文件，名字完全随意，但是多数 Druapl 主题用 style.css 来 作为指定给主题的主.css 文件名。
下面我们需要通知我们的主题应用 style.css，要做到这些，我们更新 grayscale.info 文件，加入下面一行：

```
stylesheets[all][] = css/style.css
```

这个指定了站点在所有媒体（screen、projector、print）上显示都要应用 style.css，你还可以讲样式表应用到指定的媒体，比如 print，使用下面一行：

```
stylesheets[print][] = css/print.css
```

或者为 screen 和 projector 使用同一个样式表：

```
stylesheets[screen, projector] = theScreenProjectorStyle.css
```

我们的目的是将其应用到所有媒体。
下 面我们实验 Drupal 用来渲染页面的结构，页面里我们能指定 CSS 的 id 和 class 来应用样式。如果你用 firefox，我建议你下载并安装 firebug，如果你用 IE，我建议你下载并安装 IE Developers Toolbar，或者如果你使用 safari，请使用内建的 web inspector，这 3 个工具都提供跟踪站点结构、简单标示 CSS 的 id 和 class 应用风格的能力，图 9-2 展示了 firefox 的 firebug 跟 踪一个页面时显示的信息。

给你点时间下载这些工具，如果你还没有它们，一旦安装后，使用这个跟踪器选项去测试我们用 Drupal 生成的 HTML 和 DIV。
下 一步是为 CSS 的 ID 和 class 定义风格，开始这步之前，让我们查看站点的页面源代码来检视 Drupal 为我们新的站点的主页生成的 HTML，焦点放到 DIV 标签的结构上，为简洁起见我们省略了 DIV 标签之间的 HTML 代码，如果你想去看页面的细节，你去浏览器窗口点击右键选择查看源文件。这里只展 示<body></body>之间的 DIV 结构。

```
<body class="html front logged-in one-sidebar sidebar-first page-node toolbar
tollbar-drawer">
  <div id="skip-link"> ... </div>
  <div class="region redion-page-top">..</div>
  <div id="page-wrapper">
    <div id="header">
      <div class="section clearfix">
        <a id="logo" ... ></a>
        <div id="name-and-slogan"></div>
      </div>
    </div>
    <div id="navigation">
      <div class="section">
        <ul id="main-menu"> ... </ul>
        <ul id="seconary-menu"> ... </ul>
      </div>
    </div>
    <div id="main-wraper">
      <div id="main">
        <div id="content"></div>
        <div id="sidebar-first" class="column sidebar"></div>
      </div>
    </div>
    <div id="footer"></div>
  </div>
</body>
```

可以注意到 Div 标签之间的更多内容，但是我们联系的重点是在于理解 div 结构使我们能定义风格到 css/style.css。下面是为如图 9-3 那样的显示而定义的 CSS。

```
body {
  background-color: #c6c6c6;
}
#page {
  background-color: #c6c6c6;
}
```

```
#skip-link {
   width: 960px;
   margin-right: auto;
   margin-left: auto;
   background-color: #c6c6c6;
}

#header {
   width: 960px;
   background-color: #ffffff;
   margin-right: auto;
   margin-left: auto;
   margin-top: 10px;
   height: 40px;
   padding-top: 10px;
   border-top: 3px solid #000;
   border-bottom: 3px solid #0
}

#logo {
   float: left;
   margin-left: 20px;
}

a#logo {
   text-decoration: none;
}

#name-and-slogan {
  float: left;
  margin-left: 100px;
}

#site-name a {
  text-decoration: none;
}

#navigation {
  width: 960px;
  margin-right: auto;
  margin-left: auto;
  background-color: #c6c6c6;
  height: 45px;
}
```

```
#navigation h2 {
  display: none;
}
ul#main-menu {
  background-color: #EEE;
  height: 25px;
}

ul#main-menu {
  text-decoration: none;
  padding-top: 5px;
}

ul#main-menu li a {
  text-decoration: none;
  padding-right: 10px;
}

ul#secondary-menu {
  background-color: #333;
  height: 25px;
}

ul#secondary-menu li a {
  text-decoration: none;
  color: #fff;
  padding-right: 10px;
  height: 25px;
  border-right: 1px solid #fff;
}

ul#secondary-menu a:hover {
  color: #ff0000;
}

#main-wrapper {
  clear: both;
  background-color: #ffffff;
  width: 960px;
  margin-right: auto;
  margin-left: auto;
}
```

```
#main {
  width: 960px;
  margin: 5px auto;
}

#content {
  width: 775px;
  float: right;
  padding-left: 15px;
}
#sidebar-first {
  float: left;
  width: 130px;
  margin:0;
  padding: 20px;
  background-col
}

#footer {
  width: 920px;
  padding: 20px;
  margin-right:
  margin-left: a
  clear: both;
  min-height: 10
  background-col
  color: #fff;
}

#footer a {
  color: #fff;
}
```

Figure 9-3. The site after applying the style sheet additions

保存完 style.css 之后，重新浏览站点主页，你看见的应该如图 9-3。
只有 4 行的.info 文件和几行的 CSS，我们已经是有能力建立全新的 Drupal 主题，我们还没建立模板文件、HTML 或碰到一行 PHP 代码，展示出来 Drupal 主题是如何的简单而强大。建立 Grayscale 主题之所以如此简单是因为 Drupal 有一个预定义的模板文件集合，当一个主题自己没有提供文件作为自己的一部分是，它们就被应用到主题上。下一节中，我们说一说几个模板文件的细节。

.info 文件

Grayscale 主题的.info 文件只有注册主题并使之在 Appearance 页面成为可选择所而必须的最少的信息，在大多数情况，你可能想定义你自己的的区域、包含附 加的样式表、包含 javascript 文件来作为主题的一部分。让我们看一下怎样扩展.info 文件来增加每一样属性。

为主题增加区域

一 个区域（region）本质上来说是站点页面上的一段。在我们构建 Grayscale 主题时，我们使用了 Drupal 自动为我们建立的标准区 域：sidebar first,sidebar second, content, header,footer,highlighted, help, page_bottom, page_top。有许多你想划分主题到附加的区域情况，我们通过在.info 文件中包含区域描述及在 page.tpl.php 中包含区域的结合来做到这 些。在.info 中定义一个新的区域的语法是：

```
regions[alerts] = Alerts
regions[feature] = Featture Articles
regions[socialnetworks] = Social Networks
```

你 可以在.info 文件中定义许多你想要的区域，但是你应该在你的.info 文件中包含 page_botton、page_top、help 以及 content，作为核心需求区域来发挥正常功能。下一步要更新你的 page.tpl.php 文件来增加你新的区域，在页面上显示这个区域的流程如下：

```
<div id="alerts">
  <?php print render($page['alerts']); ?>
</div><!-- /alerts -->
```

为主题增加 CSS 文件

当 我们建立 Grayscale 主题时，我们增加了一个单个的 CSS 文件，它组合了我们完成我们设计目标所需要的所有风格。可能有这样情形，需要你组合多于一 个的样式表，或你希望样式表基于浏览站点的设备，这两种方式都要通过下面的语法来完成（假设你所有的样式表都放在主题目录的 css 子目录中）。

```
;// add a style sheet that deals with colors for all mediums
stylesheets[all][] = css/colors.css
;// add a style sheet just for printing
stylesheets[print][] = css/print.css
;// add a style sheet just for projecting
stylesheets[projector][] = css/showtime.css
;// add a style sheet for screen
stylesheets[screen][] = css/style.css
;// add a style sheet for screen and projector
stylesheets[screen, projector] = css/showit.css
```

为主题增加 JavaScript 文件

如果你的主题使用 javascript，最好的实践是将 javascript 代码存储到外部文件中，要包含这些文件需要在你主题的.info 文件中列出每个 javascript 文件，假设你把所有你的 javascript 文件放到主题的 js 子目录中，包含文件的语法如下：

```
scripts[] = js/jcarousel.js
```

为主题增加设置

可能有这样情形，你希望你的主题不用动模板文件或 CSS 就能配置，如，你可能希望为站点管理员提供一个改变字体尺寸和样式的能力，我们通过提供设置来做到这些，要定义一个设置，我们组合这些定义到.info 文件中，如下：

```
settings[font_family] = 'ff-sss'
settings[font_size] = 'fs-12'
```

用上面的设置更新你的 grayscale.info 文件，随后我们实现那些允许站点管理员能设置且你站点要使用该值的魔术片段。
下 一步是提供站点管理员改变该值的手段，要做到这些，我们在主题目录中建立一个 theme-settings.php 的文件，并建立需要收集的字体族和字体 大小的值的表单元素，在 theme-settings.php 中，我们使用 hook_form_system_theme_settings_alter()函数去增加字段来设置字体族和字体大小。插入下面代码：

```
<?php

function grayscale_form_system_theme_settings_alter(&$form, &$form_state) {
  $form['style'] = array(
    '#type' => 'fieldset',
    '#title' => t('Style settings'),
    '#collapsible' => FALSE,
    '#collapsed' => FALSE,
  );

  $form['style']['font'] = array(
    '#type' => 'fieldset',
    '#title' => t('Font settings'),
    '#collapsible' => TRUE,
    '#collapsed' => TRUE,
  );
  $form['style']['font']['font_family'] = array(
    '#type' => 'select',
    '#title' => 't('Font family'),
    '#default_value' => theme_get_setting('font_family'),
    '#options' => array(
```

```
      'ff-sss' => t('Helvetica Nueue, Trebuchet MS, Arial, Nimbus Sans L, FreeSans,
sans-serif'),
      'ff-ssl' => t('Verdana, Geneva, Arial, Helvetica, sans-serif'),
      'ff-a'   => t('Arial, Helvetica, sans-serif'),
      'ff-ss'  => t('Garamond, Perpetua, Nimbus Roman No9 L, Times New Roman,
serif'),
      'ff-sl'  => t('Baskerville, Georgia, Palatino, Palatino Linotype, Book
Antiqua, URW Palladio L, serif'),
      'ff-m'   => t('Myriad Pro, Myriad, Arial, Helvetica, sans-serif'),
      'ff-l'   => t('Lucida Sans, Lucida Grande, Lucida Sans Unicode, Verdana,
Geneva, sans-serif'),
    ),
  );
  $fomr['style']['font']['font_size'] = array(
    '#type' => 'select',
    '#title' => t('Font size'),
    '#default_value => theme_get_settings('font_size'),
    '#description' => t('Font sizes are always set in relative units - the sizes
shown are the pixel value equivalent.'),
    '#options' => array(
      'fs-10' => t('10px'),
      'fs-11' => t('11px'),
      'fs-12' => t('12px'),
      'fs-13' => t('13px'),
      'fs-14' => t('14px'),
      'fs-15' => t('15px'),
      'fs-16' => t('16px'),
    ),
  );
}
```

保存文件之后浏览 Appearance 页面点击 Grayscale 主题的 Settings 连接。你应该能看见风格设置特性我们只加到了表单的底部，点击 Font Settings 连接展开这个表单，如图 9-4.

Figure 9-4. The font settings options

流 程的最终步骤是在主题中应用管理员选择了的值,我们通过在我们主题的$classes 变量中增加字体族和字体大小的设置来完成这些。要增加这些值，我们需 要建立 template.php 文件，这个文件用来做各式各样的主题处理，这个文件的细节我们在本章后面在细看，现在，我们在 Grayscale 目录中建 立 template.php 文件，并加一个 hook_process_HOOK()函数来将其参数增加到$classes 变量。 hook_process_HOOK()函数名叫 grayscale_process_html()，这

里 grayscale 是主题名，html 是我们想 覆写的.tpl.php 文件的名字，我们还能用 hook_process_HOOK()覆写任何其它的主题文件。

```
<?php

/**
 * Override or insert variables into the html template.
 */
function grayscale_process_html(&#vars) {
  // Add classes for the font styles.
  $classes = explode(' ', $vars['classes']);
  $classes[] = theme_get_setting('font_family');
  $classes[] = theme_get_setting('font_size');
  $vars['classes'] = trim(implode(' ', $classes));
}
```

设 置完变量之后，它们将被应用到 html.tpl.php 文件中并通过$calsses 变量应用到 body 标签上，html.tpl.php 文件在 modules/system 目录中，是核心的一部分，本章后面我将给你展示怎样覆写核心模板包括这个 html.tpl.php 文件。

```
<body class="<?php print $classes; ?>" <?php print $attributes; ?>>
```

如果你列印了$calsses 的值，你应该看到，如果你没改变字体族和字体大小的默认值，下面的值：
html front logged-in one-sidebar sidebar-first page-node toolbar toolbar-drawer ff-sss fs-12 你能看到 ff-sss 和 fs-12 已经加到了$classes 变量的结尾。剩下唯一的东西是为每个选项建立 CSS，我们更新 css/style.css 文 件来包含我们在先前建立的 theme-settings.php 文件中定义的每个选项的风格。

```
/* font family */
.ff-sss {font-family: "Helvetica Nueue", "Trebuchet MS", Arial, "Nimbus Sans L", FreeSans, sans-serif;}
.ff-ssl {font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;}
.ff-a {font-family: Arial, Helvetica, sans-serif;}
.ff-ss {font-family: Garamond, Perpetua, "Nimbus Roman No9 L", "Times New Roman", serif;}
.ff-sl {font-family: Baskerville, Georgia, Palatino, "Palatino Linotype", "Book Antiqua", "URW Palladio L", serif;}
.ff-m {font-family: "Myriad Pro", Myriad, Arial, Helvetica, sans-serif;}
.ff-l {font-family: "Lucida Sans", "Lucida Grande", "Lucida Sans Unicode", Verdana, Geneva, sans-serif;}

/* Base fontsize */
.fs-10 {font-size:0.833em}
.fs-11 {font-size:0.917em}
```

```
.fs-12 {font-size:1em}
.fs-13 {font-size:1.083em}
.fs-14 {font-size:1.167em}
.fs-15 {font-size:1.25em}
.fs-16 {font-size:1.333em}
```

试着访问 Appearance 页面并点击 Grayscale 主题的 Settings 连接，改变一下设置，看看变化。

理解模板文件

一 些主题包含所有类型的模板文件，而另一些，就像我们的 Grayscale 主题，只有.info 和.css 文件，主题所需要的模板文件数量依赖于你想定制多 少对应于标准 Drupal 模板的多样的组成部分。当你审视不同的模板文件，记住，如果你的主题没有提供一个这里所说的模板文件，Drupal 自己讲使用包 含在 Drupal 核心中的为每个内容（如 page node comment field）分配的模板文件，我将在下面的章节讲述这些文件。

重点

与 给定的 Drupal 主题有联系的是几个 tpl.php 文件，一些主题只提供基本的 html.tpl.php 文件，你可以将它想象为包含所有传统的 HTML 站点的<body>之上的所有东西，而 page.tpl.php 你可以认为它包含着传统的基于 HTML 站点的<body> 与</body>之间的每一样东西。有些主题利用 Drupal 的高级功能通过提供单独的主题文件来主题化独立的页面成分，例如下面

+ node.tpl.php：此文件定义节点是如何在页面上渲染
+ field.tpl.php：此文件定义字段是如何在页面上渲染
+ block.tpl.php：此主题文件定义区块如何在页面上渲染

html.tpl.php 是所有模板文件的祖父，为站点提供全部的页面布局，其它的模板文件嵌入 page.tpl.php，如下面图 9-5：

Figure 9-5. Other templates are inserted within the encompassing page.tpl.php file.

html.tpl.php、page.tpl.php、block.tpl.php、node.tpl.php 和 field.tpl.php 的嵌入在页面构建时由主题系统自动发生，如果你的主题没有包含一个或所有的上述文件，Drupal 使用下表内的内核自带的模板文件：

Template file	location	Description
html.tpl.php	modules/system	站点主模板文件，包含所有<head></head>之间的元素

page.tpl.php	modules/system	定义包含在<body></body>之中，包括二者的所有东西， 当同主页面布局全部结构工作时要修改这个文件
region.tpl.php	dules/system	定义站点上区域怎样布局和渲染
node.tpl.php	modules/node	定义站点上节点如何布局和渲染
block.tpl.php	modules/block	定义站点上区块如何布局和渲染
field.tpl.php	modules/field/theme	定义站点上的字段如何布局和渲染

在为你的 Grayscale 主题建立这些模板文件的自定义版本之前，让我们简要介绍一下列出来的核心模板文件的结构和内容。

html.tpl.php 文件

这 是在站点上显示基本的 HTML 结构的默认模板，这个主题文件的焦点在<html>标签和<body>标签之间的元素，在下面的代 码中，你能看见 html.tpl.php 文件提供的元素，如 DOCTYPE 定义、RDF 定义、HTML、少量 DIV 标签以及 PHP 代码片段，它打印分配给 不同变量的内容，它们定义在表 9-2 中，虽然模板文件相对简单，但是它演示了 Drupal 主题的力量，和通过在运行时设置各种变量的值来显示动态内容及主 题引擎用内容替换这些值的能力。这些变量的值可以通过内容如 URL 参数来设置，无论用户是匿名的还是已经登录的，用户规则是如果他登录进站点，其它内容， 就是定义为应显示在站点上的帮助。我们将看见审视其它的模板文件，但是重要的是理解那一小片 PHP 代码的意义，你可以在这看到 html.tpl.php 的 例子：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php print
$language->language; ?>"
  version="XHTML+RDFa 1.0" dir="<?php print $language->dir; ?>"<?php print
$rdf_namespaces; ?>>

<head profile="<?php print $grddl_profile; ?>">
  <?php print $head; ?>
  <title><?php print $head_title; ?></title>
  <?php print $styles; ?>
  <?php print $scripts; ?>
</head>
<body class="<?php print $classes; ?>" <?php print $attaibutes; ?>>
  <div id="skip-link">
    <a href="#main-content"><?php print t('Skip to main content'); ?></a>
  </div>
  <?php print $page_top; ?>
  <?php print $page; ?>
  <?php print $page_bottom; ?>
```

</body>
</html>

表 9-2 列出了所有的通过不同模板的处理器和预处理器函数而可用于 html.tpl.php 文件的变量。如$css 变量包含当前页面所有 CSS 文件的列表，这些 CSS 文件定义在.info 文件中，用的是 stylesheets[all]。

注意：为每个模板文件列出的变量表示这些变量可用于这个模板，你不需要在你的模板文件中使用全部的变量，
而只使用你的函数和技术需求所必需的，而在 html.tpl.php 中使用的变量是例外。

表 9-2 Variables That Are Available for Use Within the html.tpl.php File

---

Variable	Description of contents
$css	当前页面的 CSS 文件的数组
$language	（对象）站点开始时使用的语言
$language->language	包含语言的文本化描述
$language->div	语言的方向，它是“ltr”或“rtl”之一
$rdf_namescapces	在此 HTML 文档中使用的所有 RDF 名称空间前缀
$grddl_profile	一个 GRDDL 的 profile，允许代理扩展 RDF 数据
$head_title	可修改的页面抬头，用于 title 标签
$head	构造 head 段，包括 meta 标签、keyword 标签等等
$styles	style 标签，页面导入所有必需的 CSS 文件
$scripts	script 标签，页面要加载的 javascript 文件和设置
$page_top	初始化从任何改变页面的模块来的 makeup，这个变量永远要首先输出，在所有动态内容之前
$page	渲染页面内容；Drupal 用 page.tpl.php 的内容来替换$page
$page_bottom	最终关闭从任何改变页面的模块来的 makeup，这个变量永远要最后输出，在所有动态内容之后
$classes	CSS 类字符串，默认的$classes 字符串包含下面这些：html front logged-in one-sidebar sidebar-first page-node toolbar toolbar-drawer，其中的每一个都能用来作为 DIV id 和类的后缀

---

page.tpl.php 文件

下 一个模板文件我们将讲 page.tpl.php 文件。这个模板文件注重的是那些显示在<body>和</body>标签之间元素 和包含页面的 HTML 结构，包括 DIV 标签及 php 代码片段。如果我们回头看看 html.tpl.php 模板，你将看到<?php print $page; ?>这里的$page 值就是 page.tpl.php 的内容。

这个模板文件定义显示给用户的页面结构，如果你回头看这本章开 头的例子，所有我们在 Grayscale 主题中为其加风格的元素都是用 page.tpl.php 模板渲染的元素。在下面的代码中，你将看到这个页面上渲染 的所有元素以及确定什么可以在页面上显示与页面怎样结构化的条件逻辑。就像 html.tpl.php 文件一样，page.tpl.php 文件由 HTML 和 php 片段混合而成，这里的 php 片段包括条件逻辑来确定一个值是否设定和显示几个变量分配的值，这里的变量可以包含任何东西，从简单如“42”到复杂的 HTML 结构包括 Javascript 代码。一个例子是：page.tpl.php 文件包含的第一个变量是$logo，这个值使 HTML 必须显示站点的 logo，如果有，这就是为什么条件逻辑去测试是否提供了 logo。

/modules/system 中的 page.tpl.php 文件结构如下：

```
<div id="page-wrapper"><div id="page">
  <div id="page-header"><div class="section clearfix">
    <?php if ($logo): ?>
      <a href="<?php print $front_page; ?>" title="<?php print t('Home'); ?>"
rel="home" id="logo">
        <img src="<?php print $logo; ?>" alt="<?php print t('Home'); ?>" />
      </a>
    <?php endif; ?>

    <?php if ($site_name || $site_slogan): ?>
      <div id="name-and-slogan">
        <?php if ($site_name): ?>
          <?php if ($title): ?>
            <div id="site-name"><strong>
              <a href="<?php print $front_page; ?>" title="<?php print
t('Home'); ?>" rel="home"><span><?php print $site_name; ?></span></a>
            <strong></div>
          <?php else: ?>
            <h1 id="site-name">
              <a href="<?php print $front_page; ?>" title="<?php print
t('Home'); ?>" rel="home"><span><?php print $site_name; ?></span></a>
            </h1>
          <?php endif; ?>
        <?php endif; ?>
        <?php if ($site_slogan): ?>
          <div id="site-slogan"><?php print $site_slogan; ?></div>
        <?php endif; ?>
      </div> <!-- #name-and-slogan -->
    <?php endif; ?>
    <?php print reader($page['header']); ?>
```

```
  </div></div> <!-- /.section, /#header -->

  <?php if ($main_menu || $secondary_menu): ?>
    <div id="navigation"><div class="section">
      <?php print theme('link__system_main_menu', array('links' => $main_menu,
      'attaibutes' => array('id' => 'main-menu', 'class' => array('links',
'clearfix')), 'heading' => t('Main_menu'))); ?>
      <?php print theme('link__system_secondary_menu', array('links' =>
$secondary_menu,
      'attributes' => array('id' => 'secondary-menu', 'class' => array('links',
'clearfix')), 'heading' => t('Secondary_menu'))); ?>
    </div> <!-- /.section, /#navigation -->
  <?php endif; ?>

  <?php if ($breadcrumb): ?>
    <div id="breadcrumb"><?php print $breadcrumb; ?></div>
  <?php endif; ?>

  <div id="main-wrapper"><div id="main" class="clearfix">
    <div id="content" class="column"><div class="section">
      <?php if ($page['highlighted']): ?><div id="highlighted">
        <?php print render($page['highlighted']); ?></div>
      <?php endif; ?>
      <a id="main-content"></a>
      <?php print render($title_prefix); ?>
      <?php if ($title): ?><h1 class="title" id="page-title"><?php print
$title; ?></h1><?php endif; ?>
      <?php print render($title_suffix); ?>
      <?php if ($tabs): ?><div class="tabs"><?php print
render($tabs); ?></div><?php endif; ?>
      <?php print render($page['help']); ?>
      <?php if ($action_links): ?><ul class="action-links"><?php print
render($action_links); ?></ul><?php endif; ?>
      <?php print render($page['content']); ?>
      <?php print $feed_icons; ?>
    </div></div> <!-- /.section, /#content -->

    <?php if ($page['sidebar_first']): ?>
      <div id="sidebar-first" class="column sidebar"><div class="section">
        <?php print render($page['sidebar_first']); ?>
      </div></div> <!-- /.section /#sidebar-first -->
    <?php endif; ?>
    <?php if ($page['sidebar_secondary']): ?>
      <div id="sidebar-secondary" class="column sidebar"><div class="section">
```

```
        <?php print render($page['sidebar_secondary']); ?>
      </div></div> <!-- /.section /#sidebar-secondary -->
    <?php endif; ?>
  </div></div> <!-- /#main /#main-wrapper -->

  <div id="footer"><div class="section">
    <?php print render($page['footer']); ?>
  </div></div> <!-- /.section, /#footer -->
</div></div> <!-- /#page, /#page-wrapper -->
```

page.tpl.php 文件中可用的默认变量见表 9-3：

Variable	Description of contents
$base_path	Drupal 安装的基本路径，最近这个路径总是默认为/
$directory	模板所在路径，例如 modules/system 或 theme/bartik
$is_front	如果当前页面是首页则为 TRUE
$logged_in	如果用户注册并登录则为 TRUE
$is_admin	如果用户有权限访问管理页面则为 TRUE
$front_page	首页的 URL，连接到首页时用它替代$base_path，它包括语言域或前缀
$logo	logo 图片的路径
$site_name	站点名称，如果主题禁用则为空
$site_slogan	站点口号，如果主题禁用则为空
$main_menu(array)	包含站点主菜单连接的数组
$secondary_menu(array)	
$breadcrumb	
$title_prefix	包含模块的附加输出数组，准备去显示在呈现在模板中的主标题标签之前
$title	页面标题
$title_suffix	包含模块的附加输出数组，准备去显示在呈现在模板中的主标题标签之后
$message	状态和错误信息
$tabs(array)	选项卡连接到当前页面下的任一子页面
$action_links(array)	例如管理员界面的“Add menu”
$feed_icons	当前页面所有供稿图标的字符串
$node	节点对象，如果这是自动载入的页面分配的节点，并且节点 ID 是页面路径的第二个参数（如 node/123 和 node/123/revisions 但不是 comment/reply/123）

变 量
$page['help'],$page['highlighted'],$page['content'],$page['sidebar_first'],$page['sidebar_second'],$page['header'],$page['footer'] 呈现为页面区域，一个区域呈现为一个物理容器，站点管理员能分配任何一个区块级元素到那里（例如 logon 表单、搜索区块、一个节点、一个视图或一个菜 单）。如果在你的主题.info 文件中没有特别指定任一个区域，你只能获得默认的区域，在后面的章节我将讲述建立附加的区域及怎样构建你的主题.info 文件。

region.tpl.php 文件

此模板文件关注你的站点怎样显示区域（region），默认的 region.tpl.php 文件十分简单——基本上就是显示分配给一个区域的内容。

```
<?php if ($content): ?>
  <div class="<?php print $classes; ?>
    <?php print $content; ?>
  </div>
<?php endif; ?>
```

这个模板文件默认的可用变量见表 9-4：

Variable	Description of contents
$content	区域的内容，典型的是区块
$classes	可以用 CSS 进行风格化内容的类字符串，它能被预处理函数通过 $classes 变量操作 典型的值是下面一个或多个： regions 当前模板类型如"theming hook" region-[name]:就是将下划线的区域名称换成带横杠的，如 page_top 区域的类是 region-page-top
$region	region 变量的名字，就是定义在.info 文件中的
$classes_array	HTML 类属性的数组，它被平整化到变量$classes 字符串中
$is_admin	
$is_front	
$logged_in	

---

node.tpl.php 文件

这 个模板文件定义个体的节点如何在站点上显示，默认的 node.tpl.php 文件在目录 modules/node 中。在下面的列表中，你能看见元素类似前 面讨论过的其它模板文件，明显

地 HTML 和 php 片段执行条件逻辑并打印分配给变量的值，在 node.tpl.php 中你还能看到模板打印出分配给变 量$content 的值——这种情况下是独立的节点，你还能注意到模板使用“hide”来移除两个正常本应该在节点渲染时显示在页面上的元素——就是评论 和分配给节点的连接，节点通过 hide($content['comments'])和 hide($content['links'])来完成这些，这么 做的原因是我们通常想在节点显示时控制评论和连接在哪里渲染，通过先隐藏他们，然后再显示它们（使用 print render($content['comments'])和 print render(content['links'])）。你能使用这种途径去隐藏任何本应该显示的东西的元素，使用 hide()函数，如果你想在以后显示这 些元素，你可以使用 print render()函数。

```
<div id="node-<?php print $node->nid; ?>" class="<?php print $classes; ?>
clearfix"<?php print $attributes; ?>>
  <?php print $user_picture; ?>
  <?php print rebder($title_prefix); ?>
  <?php if (!page): ?>
    <h2<?php print $title_attributes; ?>><a href="<?php print $node_url; ?>"><?php
print $title; ?></a></h2>
  <?php endif; ?>
  <?php print render($title_suffix); ?>

  <?php if ($display_submitted): ?>
    <div class="submitted">
      <?php
        print t('Submitted by !username on !datetime, array('!username' => $name,
'!datetime' => $date));
      ?>
    </div>
  <?php endif; ?>

  <div class="content"<?php print $content_attributes; ?>>
    <?php
      // We hide thw comments and links now so that we can render them later.
      hide($content['comments']);
      hide($content['links']);
      print render($content);
    ?>
  </div>

  <?php print render($content['links']); ?>
  <?php print render($content['comments']; ?>
</div>
```

在模板文件 node.tpl.php 中可用的变量见表 9-5

Variable	Description of contents
$title	标题的（消毒化）版本
$content(array)	节点开始显示时产生的元素的数组，如果你要显示全部的节点内容 请使用 render($content)，或者像先前解释的用 hide()和 show()函数来显示节点对象的单独元素
$user_picture	节点作者的图片，来自 user-picture-tpl.php
$date	格式化的创建时间，预处理函数可以用 format_date()带$created 参数来 重新格式化
$name	由 theme_username()主题化的所有者名称
$node_url	当前节点的 URL
$display_submitted	
$classes	CSS 类字符串，它能在预处理函数中通过$classes_array 来操作，默认的值 可以是下面一个或多个 node:当前模板类型，如“theming hook” node-[type]:当前节点类型，例如，如果节点是 Blog entry，它就应该是“node-blog” node-teaser:节点是预告表单 node-preview:酒店节点是预览模式 下面这些是通过节点发布选项控制 node-promoted:节点生到首页 node-sticky:在预告列表中节点始终在首位 node-unpubkished:未发布节点版本，只给管理员
$title_prefix(array)	
$title_suffix(array)	
$node	完整的 node 对象
$type	节点的类型，如：story,page,blog 等
$comment_count	一个节点评论的数量
$uid	节点作者的用户 UID
$created	节点发布时的 UNIX 时间戳
$classes_array	HTML 类属性值的数组，它平整化进变量$classes 字符串中
$zebra	输出奇偶，用于预告列表的条纹化输出
$id	节点的位置，每次输出的增量
$view_mode	查看模式，如 full 或 teaser
$page	全页状态标志（TRUE 或 FALSE）
$promote	提升到主页的状态标志(TRUE 或 FALSE）
$sticky	
$status	发布状态标志
$comment	节点的评论设置状态
$readmode	标志，如果预告内容不能显示完全部内容则设为 TRUE，就是

readmore
$is_front
$logged_in
$is_admin

---

field.tpl.php 文件

这个模板文件用来主题化字段，不像前面讲的模板，当渲染字段时它不被 Drupal 自动调用，你要想使用这个模板，就要将它从/module/fields/templates 拷贝到你的主题目录下。

```
<div class="<?php print $classes; ?> clearfix"<?php print $attributes; ?>>
  <?php if (!$label_hidden) : ?>
    <div class="field-label"<?php print $title_attributes; ?>>
      <?php print $label ?>: </div>
  <?php endif; ?>
  <div class="field-items"<?php print $content_attributes; ?>>
    <?php foreach ($items as $delta => $item) : ?>
      <div class="field-item <?php print $delta % 2 ? 'odd' : 'even'; ?>" <?php print
$item_attributes[$delta]; ?>>
        <?php print render($item); ?>
      </div>
    <?php endforech; ?>
  </div>
</div>
```

模板文件 field.tpl.php 中可用的默认变量如下表：

Variable	Description of contents
$items	字段值数组；使用 render()去输出他们
$label	项目标签
$label_hidden	一个标志，用于设置是否显示标签
$classes	CSS 类字符串，能通过 CSS 操作，能通过预处理函数的变量$classes 数组操作 默认的值是以下一个或多个值 field:当前模板类型，如“theming hook” field-name-[filed_name]:当前字段名。例如字段名是 field_description，那么他就 应该是 field-name-field-description field-type-[field_type]:当前字段类型，如如果字段类型是 text，那么它就应该是

field-type-text

field-label-[field_display]:当前标签位置，例如，如果标签位置是 above，它就应该是

field-label-above

$element['#object'] 字段所附加到的实体

$element['#view_mode'] 字段附加到的实体的查看模式，如 full 或 teaser

$element['#field_name'] 字段名称

$element['#field_type'] 字段类型

$element['#field_language'] 字段语言

$element['#field_translatable'] 字段是否可翻译

$element['#label_display'] 标签显示的位置：inline,above,hidden

$field_name_css CSS 兼容的字段名

$field_type_css CSS 兼容的字段类型

$classes_array HTML 类属性值的数组，它平整化进变量$classes 字符串中

---

block.tpl.php 文件

block 层主题模板，block.tpl.php 位于 modules/block 目录中，依据这一点，你应该看到一个明确的模式构建的模板文件

```
<div id="<?php print $block_html_id; ?>" class="<?php print $classes; ?>"<?php print $attributes; ?>>
  <?php print render($title_prefix); ?>
  <?php if ($block->subject) : ?>
    <h2<?php print $title_attributes; ?>><?php print $block->sybject ?></h2>
  <?php endif; ?>
  <?php print render($title_suffix); ?>
  <div class="content"<?php print $content_attributes; ?>>
    <?php print $content; ?>
  </div>
</div>
```

模板文件 block.tpl.php 中可用的默认变量见表 9-7

Variable Description of contents

$block->subject 区块标题

$content 区块的内容

$block->module 生成区块的模块

$block->delta　　block 的 ID，每个模块中唯一
$block->region　　封装当前区块的区域
$classes　　CSS 类字符串，能通过 CSS 操作，能通过预处理函数的变量$classes 数组操作
默认的值是以下一个或多个值
block:当前模板类型，如"theming hook"
block-[module]:生成区块的模块，例如，用户模块提供处理默认的用户导航区块，
这种情形下，它应该是 block-user
$title_prefix(array)
$title_suffix(array)
$classes_array(array)　　HTML 类属性值的数组，它平整化进变量$classes 字符串中
$block_zebra
$blcok_id　　依赖于每个区块的区域
$is_front
$logged_in
$is_admin
$block_html_id　　生成的唯一的有效的 HTML 的 ID

覆写模板文件

可 能看到有时你需要修改 page.tpl.php，node.tpl.php 或其它任何一个标准模板文件来决定怎样在你的站点上显示元素，让我们回过头看一 下 Grayscale 主题，自定义 page.tpl.php 文件使之在用户在主页浏览时显示一条欢迎信息，如果用户不在主页上，就不显示欢迎信息。开始的 处理是将 page.tpl.php 文件从 module/system 目录拷贝到 sites/all/themes/grayscale /templates 目录，通过拷贝它到我们主题的目录，Drupal 将使用我们版本的 page.tpl.php，而不是 modules/system 下 的那个。
修改起来相对简单，我们使用$is_front 变量送给模板文件，通过条件解析，检查浏览者是不是在主页上，如果在，就显示“Welcome to My Site”，打开文件，找到下面一行：

```
<div id="content" class="column"><div class="section">
```

直接到这行后面，插入下面代码，它用变量$is_front 检查浏览者是否在主页，如果是，则打印出一条欢迎信息：

```
<?php
  if ($is_front) : ?><div id="welcome_message">
    <?php print "Welcome to My Site"; ?></div>
<?php endif; ?>
```

结果是“Welcome to My Site”打印在第二个菜单之下，没什么激动人心的，但是它演示了标准 page.tpl.php 文件和自定义的杠杆作用概念。

注意：如果你自定义的 page.tpl.php 文件没有效果，导航到
admin/config/development/preformace 并清除缓存

你 还能为站点特定的页面创建自定义的.tpl 文件，例如，你能拷贝 page.tpl.php 并建立一个 page--front.tpl.php 文件，这个新 模板只能应用到你站点的主页。你还能用 node.tpl.php 做同样的事情，比如你要给 article 节点做个与其它节点不同的主题形式，你可以从 modules/node 拷贝 node.tpl.php 文件到你的主题目录并重命名为 node--article.tpl.php，如果节点是 article，那么这个新的模板文件将覆写 node.tpl.php，对于另外的细节，请浏览 http://drupal.org/documentation/theme 上的主题指南。

注意：这里在 node 和 article 中间是两个横杠，Drupal 要求两个横杠。

其它模板文件

通 过浏览站点的模块和主题目录，你可以看到几个其它的模板文件，比如，comment 模块使用 comment.tpl.php 来渲染评论，comment 模 块创建了几个变量，并将这些变量送进 comment.tpl.php 文件。comment.tpl.php 文件的指定就如同评论的模板文件是通过以'template' => 'comment'为参数调用 hook_theme()生成的一样，'template' => 'comment'是作为数组中的一个值（见下面代码），这里不需要指定.tpl.php 扩展名，如同 Drupal 猜中你的意思一样。我将讲一些怎样去建 立变量并将变量送到你的模板文件的具体细节。

```
/**
 * Implements hook_theme().
 */
function comment_theme() {
  return array(
    'comment_block' => array(
      'variables' => array(),
    ),
    'comment_preview' => array(
      'variables' => array('comment' => NULL),
    'comment' => array(
      'template' => 'comment',
      'render element' => 'elements',
    ),
    'comment_post_forbidden' => array(
      'variables' => array('node' => NULL),
```

```
    ),
    'comment_wrapper' => array(
      'template' => 'comment-wrapper',
      'render element' => 'content',
    ),
  );
}
```

theme()函数简介

当 Drupal 想为一个可主题化项目（如一个节点、一个区块、一个面包屑踪迹、一个评论、或一个用户签名）生成某些 HTML 输出，它将查找将要为此项目生成 输出的主题函数或模板文件。Drupal 差不多所有部分都是可主题化的，这意味着你能你能覆写为此项目生成的实际的 HTML，一会我们看看例子。

Tip：Drupal 的所有可主题化项目列表，请见 http://api.drupal.org/group/themeable/7
---

theme()工作流概览

这是一个简单的节点页面，如 http://example.com/?q=node/3 显示的时候发生的情况的高层概览：

1. Drupal 的菜单系统接收请求并将控制放手给 node 模块
2. 在建立了节点数据结构之后，theme('node', $wariables)被调用。它找到恰当的主题函数或模板文件

   定义模板可用的变量，并且应用这个模板，结果是完成了节点的 HTML（如果有多个节点要显示，那么这个

   处理过程为每个节点来一次）
3. 一个 HTML 结构返回（你能在 index.php 中看到它是作为$return 变量）并且再次传递给 theme()函数，就像

   theme('page', $return)
4. 在处理页面模板前，Drupal 做一些预处理，诸如发现哪些区域是可用的及每个区域应该显示哪些区块，每

   每个区块是通过调用 theme('blocks', $region)来转换进 HTML 的，它定义变量并应用一个区块模板，你可

   以在这里看见一些模式
5. 最终，Drupal 为页面模板定义变量并应用页面模板

你 可以从 theme()处理流程看出，theme()函数对于 Drupal 是多么重要，它负责运行预处理函数来设置将在模板中应用的变量，并且安排一个主题 调用恰当的函数或找出一个恰当的模板文件，结果是 HTML。我们将在后面深层次看一下这个函数怎样工作，现在已足够理

解当 Drupal 想去把节点转换进 HTML 时发生的事情：theme('node', $variables = array())被调用。依赖于是哪个主题被激活，theme_node()生成 HTML 或者用名字叫 node.tpl.php 的模板文件完成这些。
这 个处理可以在许多层次被覆写，例如，主题能覆写内建的主题函数，那样的话，当 theme('node', $variables = array())被调用，一个叫 grayscale_node()的函数可以代替 theme_node()来处理它。模板文件命名规则我们稍后再看，一个 node--story.tpl.php 的模板文件只能影响 story 类型的节点。

覆写可主题化项目

正如你所见，可主题化项目是由它们的函数名来识别的，它们都以 theme_开头，或者项目有一个模板文件（标准的可主题化项目见 http://api.drupal.org/api/group/themeable/7），命名规则使 Drupal 有能力为所有可主题化函数建立一个函数-覆写机制。设计者能指示 Drupal 去执行一个函数，优先于模块开发者指定的主题函数，或者覆盖 Drupal 的默认模板。让我们看看当构建站点面包屑时处理工作是怎样的。
打开 includes/theme.inc，检查文件中的函数，这里许多函数都是以 theme_开头，明显它们能被覆写，让我们检查下 theme_breadcrumb()函数：

```
/**
 * Returns HTML for a breadcrumb trail.
 *
 * @param $variables
 *   An associative array containing:
 *   - breadcrumb: A array containing the breadcrumb links.
 */
function theme_breadcrumb($variables) {
  $breadcrumb = $variables['beradcrumb'];

  if (!empty($breadcrumb)) {
    // Provide a navigational heading to give context for breadcrumb links to
    // screen-reader users. Make the heading invisible with .element-invisible.
    $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';

    $output .='<div class="breadcrumb">' . implode(' >> ', $breadcrumb) . '</div>';
  }
}
```

这 个函数控制站点面包屑导航的 HTML，一般来讲，它在每个面包屑踪迹之间增加一个右向的双箭头，假设你想把 标签改 为标签，同时用一个星号代替双箭头。该怎么做呢？有一种解决方案，编辑 theme.inc 中的这个函数，保存它，调用它。（不！ 不！，绝不要怎么做）这有更好的方案。
你曾经见过在核心中如何调用，你绝没有见过直接调用 theme_breadcrumb()，作为替代，它总是封装进 theme()助手函数，你希望函数如下面方式调用：
theme_breadcrumb($variables)
其实不然，你将见到开发者这样调用

theme('breadcrumb', $variables);
通用的 theme()函数为初始化主题层负责，并且在恰当的地方安排函数调用，给我们带来优雅的解决问题的方案。调用 theme()函数使 Drupal 按下面的顺序查找 breadcrumb 函数。
假设你使用的主题是 Grayscale，是以 PHPTemplate 为基础的主题，Drupal 应按下面顺序查找（我们先忽略 breadcrumb.tpl.php 一下）：

grayscale_breadcrumb()
sites/all/themes/grayscale/breadcrumb.tpl.php
them_breadcrumb()

你主题的 template.php 文件在这覆写 Drupal 默认的主题函数，并且拦截和建立自定义变量传递给模板文件。

注意：在做这个例子时，不要使 Bartik 作为活动主题，因为 Bartik 已经有一个 template.php 文件

要 调整 Drupal 面包屑，建立（或更新，如果你在前面的例子中建了一个的话）一个 sites/all/themes/grayscale /template.php 文件，拷贝粘贴 theme.inc 中的 theme_breadcrumb()函数，确保包括<?php 标签，还有将 theme_breadcrumb 改为 grayscale_breadcrumb，下一步，点击菜单的 Modules 连接，Drupal 就会重建主题注 册，发现你的新函数。

```
<?php

/**
 * Returns HTML for a breadcrumb trail.
 *
 * @param $variables
 *   An associative array containing:
 *   - breadcrumb: A array containing the breadcrumb links.
 */
function grayscale_breadcrumb($variables) {
  $breadcrumb = $variables['beradcrumb'];

  if (!empty($breadcrumb)) {
    // Provide a navigational heading to give context for breadcrumb links to
    // screen-reader users. Make the heading invisible with .element-invisible.
    $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';

    $output .='<div class="breadcrumb">' . implode(' >> ', $breadcrumb) . '</div>';
  }
}
```

下面，如果你用 Grayscale，在 css/style.css 中加入：

```
.breadcrumb {
  margin-top: 10px;
  clear: both;
  height: 15px;
  backgound-color: #fff;
  width: 960px;
  margin-right: auto;
  margin-left: auto;
}
```

下 一时间 Drupal 去格式化面包屑踪迹，它首先找到你的函数并用它老代替默认的 theme_breadcrumb()函数，面包屑将用星号来代替 Drupal 默认的双箭头。通过给 theme()传递所有主题函数，Drupal 总是检查当前的主题是否覆写任何一个 theme_函数，有就调用它们。

注意：你主题中任何输出 HTML 或 XML 的部分，都应包含进 theme 函数中，这样它们能被做主题的访问到来覆写它们。

用 Template 文件覆写

如果你和一个设计师工作，告诉他“只要进入代码并找到可主题化函数并覆写”，这不是问题，幸运的是，有另外的方式来得到更容易的设计师方式，你能将可主题化项目放进他们自己的模板文件，我将利用手头的面包屑例子来做演示。

在 我们开始之前，确认没有主题函数覆写了 theme_breadcrumb()，如果我们在前面章节中在主题的 template.php 文件里建立了 grayscale_breadcrumb()函数，请注释掉它，建立文件 sites/all/themes/grayscale /breadcrumb.tpl.php，这是一个新的模板文件，因为我们想将 标签改成标签，在文件中输 入：

```
<?php if (!empty($breadcrumb)) : ?>
  <span class="breadcrumb"><?php print implode('!', $breadcrumb) ?></span>
<?php endif; ?>
```

这对设计师来说就足够简单了，现在你需要让 Drupal 知道在渲染 breadcrumb 时找到这个模板文件，为此，需要清空缓存，Drupal 将发现你的 breadcrumb.tpl.php 文件，并且映射面包屑可主题化项目到这个模板文件。
现在你知道怎样在 Drupal 中以设计师的喜好来覆写任意的可主题化项目。

增加和操作模板变量

在这个例子中，我们看一下操作或增加传递进页面或节点模板的变量，让我们继续前面的例子。修改 sites/all/themes/grayscal/breadcrumb.tpl.php，用一个叫$breadcrumb_delimiter 的变量来为面包屑定义分隔符。

```
<?php if (!empty($breadcrumb)) : ?>
  <span class="breadcrumb">
    <?php print implode(' ', $breadcrumb_delimiter . ' ' $breadcrumb); ?>
  </span>
<?php endif; ?>
```

要设置这个$breadcrumb_delimiter 变量，一个选择是在模块中，我们建立一个 sites/all/modules/crumbpicker.info：

```
name = Breadcrumb Picker
description = Provide a character for the breadcrumb trail delimiter.
package = Pro Drupal Development
core = 7.x
```

crumbpicker.module 应该很小：

```
<?php
/**
 * @file
 * Provide a character for the breadcrumb trail delimiter.
 */

/**
 * Implements $modulename_preprocess_$hook().
 */
function crumbpicker_preprocess_breadcrumb(&$variables) {
  $variables['breadcrumb_delimiter'] = '/';
}
```

在激活这个模块后，你可以看到面包屑踪迹的分隔符变成了/。
这个预处理例子刻画出一个模块设置一个模板文件使用的变量,但是应该有一个比建立模块更简单的方式来每次设置一个变量，果然，template.php 文件前来救驾，让我们写一个函数来设置面包屑分隔符，写到主题的 template.php 文件里：

```
/**
 * Implements $themeenginename_preprocess_$hook().
 * Variables we set here will be available to the breadcrumb template file.
 */
function grayscale_precess_breadcrumb(&$variables) {
  $variables['breadcrumb_delimiter'] = '#';
}
```

这比建立模块简单、直白，模块方法通常最适合使存在的模块去提供模板变量；模块通常不是单单为此目的而写。现在，我们有一个模块提供一个变量、template.php 中函数提供一个变量，通常使用的是哪个？
通 常预处理函数层级运行于一个特定的循序，它们每一个都潜在地覆写前一个预处理函数已经定义的变量。在上面的例子中，面包屑分隔符应该是#，因为 phptemplate_preprocess_breadcrumb()将在 crumbpicker_preprocess_breadcrumb() 之后执行，因此它分配的变量$breadcrumb_delimiter 将覆写原先分配的变量。
对于使用 Grayscale 主题来主题化面包屑，实际的预处理顺序（第一个调用和最后一个调用）将是这样：

template_preprocess()
template_preprocess_breadcrumb()
crumbpicker_preprocess()
crumbpicker_preprocess_breadcrumb()
phptemplate_preprocess()
phptemplate_preprocess_breadcrumb()
grayscale_preprocess()
grayscale_preprocess_breadcrumb()
template_process()

因 此，grayscale_preprocess_breadcrumb()能覆写任何一设置的变量，它在变量呗模板文件处理前最后一个调用。当这些函数只 有部分实现时就调用所有这些函数在你看来是浪费时间，那你是正确的，在主题注册时，Drupal 就确定了那些函数已实现了，并只调用它们。

使用主题开发模块

一个 Drupal 主题开发的宝贵资源是 developer 模块，它是 devel.module 的一部分，并且可以从 http://drupal.org/project/devel_themer 下载，主题开发者模块在页面上指定一个元素，并发现那个模板或主题函数是用来建立这个元素的以及此元素可用的变量（和它的值）。

小结

读完本章之后，你应该可以：

+ 建立模板文件
+ 覆写主题函数
+ 操作模板变量
+ 为区块建立新的页面区域

Drupal 7 主题其它的细节请查看 http://drupal.org/documentation/theme 上的主题手册页面。

区 块是一片文本或功能区域，能放置在主题定义的区域中。模块可以是下面任何的东西：一个节点、一个节点列表、一个视频、一个表单、一个在线投票、一个对话窗 口、一个 fb 的状态更新、或你任何能想象出来的东西。当我同客户谈及区块，他们经常这么回答“哦，

你的意思区块就是一个控件”，通常这个术语用于 Drupal 范围之外那些表现为区块的东西，我将展示如何建立和使用区块。

区块是什么

一个区块本质上是一个独立的容器，它可以容纳你能想象出来任何虚拟东西。通过对几个例子的考察可能更容易理解什么是区块（表 10-1），下表程序 Drupal 带的一些标准区块。

Block	Description
Login Form	允许用户登录、注册、重置密码的表单
Who's online	在此列出所有在站点登录的用户
Who's new	显示站点最新用户列表的区块
Search form	搜索表单时包含在 block 里面的
recent comments	
Main menu and secondary menu	
Recent content	
Most recent poll	
Active forum topics	

几个贡献模块，包括 block 就像一个他们传递的功能解决方案的组成，Ubercart 提供了几个区块，用于显示访问者的购物车状态信息。
使用 Block API 和区块管理接口，你可以为你虚拟的有目的的的东西建立自定义区块，例如我将在下面建立的一些区块：

Block	Description
Recent Bloggers	显示最后发贴的作者头像相册的区块
slideshow of upcoming events	显示未来节点预告的区块，像一个幻灯片
A chat form	
A Donate now feature	
A list of new books added to a library's collection	
A contact us form	
A list of postings on multiple social networking sites	
A Google map showing recent postings	

区 块可以通过 Drupal 的管理界面或 Block API（blocks 模块提供）程序的随便一种方式定义，在创建一个区块时，你怎样知道是用的哪种方法？一个一次性区块如与站点有关的一小片静态 HTML 是自定义区块的一个好的候选者。那些关联到你的模块的天然是动态的区块，

或者由大量 PHP 代码组成的是使用 Block API 的极好的候选者并且在模块内实现。试图在自定义区块中避免 PHP 代码，正如数据库中的代码维护要难于模块中的代码。站点的编辑会进步并意外地删除所 有那些困难工作太容易了。如果在模块层建立一个区块没有什么意义，只要调用一个这个区块内的自定义函数并存储 PHP 代码到别处。

Tip:一个通常的有关区块和其它独特成分的实践是去建立一个站点特有的模块并将站点特定的功能放置到这个模块中，例如一个站点的开发者 Jones Pies 和 Soda Company 可以建立一个 jonespiessoda 模块【特有功能分割原理】

尽 管 block API 相对简单，但是不要忽视框架内你做东西得复杂性。区块能显示你想出来任何东西（它们是用 PHP 写的，因此你能做的是无限的）但是它们通常履行对站点 主要内用支持规则。例如，你能为每个用户规则建立一个自定义导航区块或者你能依赖于批准去暴露出一个评论列表。

区块配置选项

一个通用的配置选项你应该熟悉，设置一个区块的可视化配置页。区块的可视化定义当一个区块在一个页面上是否可见是基于你使用图 10-2 接口所指定的条件，用用户登录区块作为例子，你通过下面选项控制区块是否可见：

+ page_specific visibility settings：管理员可以选择在特定页面或一定范围的页面上区块是显示还是隐藏，或者
你自定义 PHP 代码决定条件的真假

+ content types visibility settings：管理员能选择区块只能显示在一个特定内容的页面，如，只在论坛主题页显示

+ Role-specific visibility settings：管理员可以选择只给特定规则的用户显示区块

+ User-specific visibility settings：管理员能允许单独的用户去自定义给定区块的可视性，用户点击”My account”
来修改区块的可视性

定义一个区块

在 模块中区块是用 hook_block_info()定义的，并且一个模块可以用一个该函数定义多个区块，一旦区块定义了，它就将显示在区块管理页面。此 外，站点管理员能通过 web 接口手工建立自定义模块。在本章，我们的焦点在程序化建立模块，让我们看看区块的数据库定义，如图 10-3：

每个区块的所有属性都存储在 block 表中。那些由配置接口建立的区块的附加数据则存储在其它的提供的表中，如图 10-3 所示。
下面的属性定义在 block 表的列中：

+ bid：每个区块的唯一 ID

+ module：这个列包含定义此区块的模块的名字，用户登录区块由 user 模块创建，等等。在后台手工建的区块呗认为是 block 模块建立的。

+ delta：因为模块在 hook_block_info()中能定义多个区块，这个 delta 列里存储的是在同一个 hook_block_info()中定义的区块的唯一键，delta 应该是一个字符串。

+ theme：区块可以为多个主题定义，因此 Drupal 需要存储激活此区块的主题的名字，每一个激活此区块的主题在数据库中拥有自己的行，配置选项不跨主题共享。

+ status：跟踪区块是否被激活，1 为激活，0 为未激活，如果一个区块没有分配到一个区域，Drupal 将 status 置为 0.

+ weight：区块的宽度决定在同一区域内与其它区块的相关位置。

+ region：封装这个区块的区域名称如 footer 等。

+ custom：用户指定的可视性设置，为 0 意味着用户不能控制区块的可视性，为 1 意味着区块默认显示，用户不可以隐藏它，为 2 意味着区块默认隐藏，但用户能选择显示它。

+ visibility：确定区块怎样呈现，0 表示在所有页面展示，除了列出的页面，1 意味着只在列出的页面展示，2 表示 Drupal 将执行自定义的 PHP 代码决定可视性。

+ pages：本字段的内容依赖于 visibility 字段的设置，visibility 字段为 0 或 1，则此字段包含页面的列表，visibility 字段为 2 则此字段包含自定义的 PHP 代码来在显示此区块时执行。

+ title：区块的自定义标题。为空则区块使用默认标题（有创建区块的模块提供）如果为<none>，这不显示任何标题。

+ cache：确定 Drupal 怎样缓存此区块，-1 不缓存，1 为每个规则缓存，这是默认设置，2 则为每个用户缓存，4 为每个页面缓存，8 则缓存，不管是规则、用户、页面都使用相同方式。

使用区块钩子

区 块钩子——hook_block_info()，hook_block_configure()，hook_block_save()和 hook_block_view()处理所有程序化创建区块的逻辑，使用这些钩子，你能声明一个或多个区块，任何模块都可以通过实现这些钩子来建立区块， 让我们看看这些钩子：

+ hook_block_info()：定义所有模块提供的区块。

+ hook_block_configure($delta = '')：区块的配置表单，$delta 参数是区块返回的 ID，你能用为这个$delta 参数使用一个整数或字符串，相同的参数也应用在 hook_block_save 和 hook_block_view 钩子中。

+ hook_block_save($delta = '', $edit = array())：为区块保存配置选项，$edit 参数包含从区块配置表单提交的表单数据。

+ hook_block_view($delta = '')：处理区块在一个区域内激活来显示它的内容。

创建一个区块

在本例中，你建立两个区块，内容适度简单便于管理，首先，建立一个区块，列出待审批的评论，然后建立一个区块列出未发布的节点。两个区块都有连接到适度的配置表单。
让我们建立一个新的 appval.module 模块来承载我们的区块代码。
approval.info

```
name = Approval
description = Blocks for facilitating pending content workflow.
package = Pro Drupal Development
core = 7.x
version = VERSION
files[] = approval.module
```

approval.module

```
<?php

/**
 * @file
 * Implements various blocks to improve pending content workflow.
 */
```

一旦这个文件建立了，通过 Modules 页面激活这个模块，你能继续在 approval.module 文件内工作，继续编辑你的文本。
让我们增加 hook_block_info()，那样我们的区块就呈现在区块管理页面的区块列表中了（见图 10-4）。我将在 info 属性定义它显示的标题，status 定义为 True 那样它就自动激活，region 设置为 sidebar_first，weight 设置为 0，visibility 设置为 1（可见）。

```
/**
 * Implements hook_block_info().
 */
function approval_block_info() {
  $blocks['pending_comments'] = array(
    'info' => t('Pending Comments'),
    'status' => TRUE,
    'region' => 'sidebar_first',
    'weight' => 0,
    'visibility' => 1,
  );
  return $blocks;
}
```

注 意 info 的值不是区块展示给用户的标题，info 是描述区块可以在区块列表里呈现，能由管理员配置，你能在以后通过 hook_block_view() 实现实际的区块标题。然而你首先要设置附加的配置选项，为此，需实现 hook_block_configure 钩子，你建立一个新的表单字段，它在点击 configure 连接后可以显示，如图 10-5：

```
/**
 * Implements hook_block_configure().
 */
function approval_block_configure($delta) {
  $form = array();
  switch($delta) {
    case 'pending_comments' :
      $form['pending_comment_count'] = array(
        '#type' => 'textfield',
        '#title' => t('Configure Number of Comments to Display.'),
        '#size' => 6,
        '#description' => t('Enter the number of pending comments that will paaear
in the block.'),
        '#default_value' => variable_get('pending_comment_count', 5),
      );
      break;
  }
  return $form;
}
```

当图 10-5 这个区块的配置表单提交时，将触发 hook_block_save()，你将下面的程序保存这个表单字段：

```
/**
 * Implements hook_block_save().
 */
function approval_block_save($delta = '', $edit = array()) {
  switch($delta) {
    case 'pending_comments' :
      variable_set('pending_comment_count',
(int)$edit['pending_comment_count']);
      break;
  }
```

```
  return;
}
```

你使用 Drupal 内置的变量系统 variable_set()来保存未审评论数量，注意我们将类型转换为整型，最后，使用 hook_block_view()和一个区块显示时返回一个未决评论列表的自定义函数来增加视觉操作：

```
/**
 * Implements hook_block_view().
 */
function approval_block_view(delta = '') {
  switch($delta) {
    case 'pending_comments' :
      $block['subject'] = t('Pending Comments');
      $block['content'] = approval_block_content($delta);
      return $block;
      break;
  }
}

/**
 * A module-defined blcok content function.
 */
function approval_block_contents($delta) {
  switch ($delta) {
    case 'pending_comments' :
      if (user_access('administer comments')) {
        $nbr_comments = variable_get('pending_comment_count');
        $result = db_query('SELECT cid, subject FROM {comment} WHERE status = 0
limit $nbr_comments");
        $items = array();
        foreach ($result as $row) {
          $items[] = l($row->subject, 'comment/' . $row->cid . '/edit');
        }
        return array('#markup' => theme('item_list', array('item' => $items)));
      }
      break;
  }
}
```

这里我们查询数据库来获得需要审批的评论，将这些评论的标题显示为连接，如图 10-6
你还设置用下面的行设置了区块的标题：
$blockp['subject'] = t('Pending comments');

现在，Pending comments 区块完成了，让我们定义另一个区块，这里含有 approval_block_info()函数——它列出所有为发布节点并提供一个连接到它的编辑页：

```
/**
 * Implements hook_block_info().
 */
function approval_block_info() {
  $blocks['pending_comments'] = array(
    'info' => t('Pending Comments'),
    'status' => TRUE,
    'region' => 'sidebar_first',
    'weight' => 0,
    'visibility' => 1,
  );
  $block['unpublished_nodes'] = array(
    'info' => t('Unpublished nodes');
    'status' => TRUE,
    'region' => 'sidebar_first',
    'weight' => 0,
    'visibility' => 1,
  );
  return $block;
}
```

注意，每个区块都分配一个键（$blcok['pending_comments'],$block['unpublished_nodes']..... $block['xxxxx'])。block 模块随后将这些键作为$delta 参数使用。
我将更新 hook_block_configure()和 hook_block_save()函数，增加设置显示的节点数量的表单保存由管理员在这个表单输入的值。

```
/**
 * Implements hook_block_configure().
 */
function approval_block_configure($delta) {
  $form = array();
  switch($delta) {
    case 'pending_comments' :
      $form['pending_comment_count'] = array(
        '#type' => 'textfield',
        '#title' => t('Configure Number of Comments to Display.'),
        '#size' => 6,
```

```
        '#description' => t('Enter the number of pending comments that will paaear
in the block.'),
        '#default_value' => variable_get('pending_comment_count', 5),
      );
      break;
    case 'unpublished_nodes' :
      $form['unpublished_node_count'] = array(
        '#type' => 'textfield',
        '#title' => t('Configure Number of Nodes to Display'),
        '#size' => 6,
        '#description' => t('Enter the number of number of unpublished nodes that
will appear in the block.');
        '#default_value' => variable_get('unpublished_node_count', 5),
      );
      break;
  }
  return $form;
}

/**
 * Implements hook_block_save().
 */
function approval_block_save($delta = '', $edit = array()) {
  switch($delta) {
    case 'pending_comments' :
      variable_set('pending_comment_count',
(int)$edit['pending_comment_count']);
      break;
    case 'unpublished_nodes' :
      variable_set('unpublished_node_count',
(int)$edit['unpublished_node_count']);
      break;
  }
  return;
}
```

我还将更新 hook_block_view 和 approval_block_content 函数来增加未发布节点的显示。

```
/**
 * Implements hook_block_view().
 */
function approval_block_view(delta = '') {
  switch($delta) {
    case 'pending_comments' :
      $block['subject'] = t('Pending Comments');
```

```
      $block['content'] = approval_block_content($delta);
      return $block;
      break;
    case 'unpublished_nodes' :
      $block['subject'] = t('Unpublished Nodes');
      $block['content'] = approval_block_content($delta);
      return $block;
      break;
  }
}

/**
 * A module-defined blcok content function.
 */
function approval_block_contents($delta) {
  switch ($delta) {
    case 'pending_comments' :
      if (user_access('administer comments')) {
        $nbr_comments = variable_get('pending_comment_count');
        $result = db_query('SELECT cid, subject FROM {comment} WHERE status = 0
limit $nbr_comments");
        $items = array();
        foreach ($result as $row) {
          $items[] = l($row->subject, 'comment/' . $row->cid . '/edit');
        }
        return array('#markup' => theme('item_list', array('item' => $items)));
      }
      break;
    case 'unpublished_nodes' :
      if (user_access('adminsier nodes')) {
        $nbr_nodes = variable_get('unpublished_node_count', 5);
        $result = db_query_range('SELECT nid, titlem FROM {node} WHERE status = 0',
0, $nbr_nodes);
        $items = array();
        foreach ($result as $row) {
          $items[] = l($row->title, 'node/' . $row->nid . '/edit');
        }
        return array('#markup' => theme('item_list', array('items' => $items)));
      }
      break;
  }
}
```

你的新未发布节点区块看起来像图 10-7

在模块安装时激活模块

在 approval.module，我们自动激活区块并分配他们到一个区域。例如我们建立了 Pending Comments 区块自动激活它（status = TRUE）并分配它到 sidebar_first 区域。

```
$blocks['pending_comments'] = array(
    'info' => t('Pending Comments'),
    'status' => TRUE,
    'region' => 'sidebar_first',
    'weight' => 0,
    'visibility' => 1,
  );
```

相同的情况，你可以希望让管理员来决定是否激活区块并决定将它分配给某个区域，为此，将 status 属性设置为 FALSE 并不给它指定区域，下面例子演示建立一个新区块 Pending Users 不自动激活也没给它指定区域，

```
$block['pending_users'] = array(
  'info' => t('Pending Users'),
  'status' => FALSE,
  'weight' => 0,
);
```

区块可见性例子

在 区块管理员接口中，你能在区块配置页面的“Page visibility settings”节输入 PHP 代码片段，当页面建立时，Drupal 将允许这个 PHP 片段来决定区块是否显示；下面是一些常用片段例子，每一个片段应该 返回 TRUE 或 FALSE 来指示区块为特别需求显示。

只为登录用户显示

当$user->uid 不为 0 时返回 TRUE

```
<?php
  global $user;
  return (bool) $user->uid;
?>
```

只为匿名用户显示

```
<?php
  global $user;
```

```
  return !(bool) $user->uid;
?>
```

小结

在本章你学到了下面知识：

+ 区块是什么和节点有什么不同
+ 怎样可视化和布置区块
+ 怎样定义一个或多个区块
+ 怎样默认激活一个区块