

Drupal 专业开发指南 (中文版)

本书英文版原名为 **Pro Drupal Development**，作者 **John VanDyk** 为 **Matt Westgate**。中文版由**葛红儒**翻译。PDF 版本由 **Hezll** 制作。供大家学习交流之用，请不要私自传播，或者商业化，因为没有获得原作者授权。因私自传播造成的后果中文版作者不承担责任。

Drupal 是当今最流行的内容管理系统(CMS)之一。本书全面讲授了 **Drupal** 定制开发的方法，剖析了 **Drupal** 强大的体系结构。

- 本书由 **Drupal** 核心开发人员编写
- 全面揭示了 **Drupal** 的体系结构和行为
- 介绍了模块的编写方法
- 讲授了主题（**themes**）的制作方法

作者和审稿人

关于作者



■**JOHN VANDYK** 最初开始接触计算机，是在一个黑色的 **Bell & Howell Apple II** 上，为 **Little Brick Out**，打印和检查 **BASIC** 代码，以增加 **paddle** 宽度。在发现 **Drupal** 以前，**John** 参与了 **UserLand Frontier** 社区，并使用 **Ruby** 编写了自己的内容管理系统（和 **Matt Westgate**）。

John 在 **Iowa** 州立科技大学的昆虫系工作，他是一个系统分析员和助理教授。他的硕士论文是关于鹿鸣的耐寒性的，他的博士论文关于正在研究的使用相片来创建 **3** 维虚拟昆虫。

John 和他的妻子 **Tina** 生活在 **Ames, Iowa**。他们在家教育他们的 **6** 个孩子，他们已经习惯了在睡觉前听父亲讲段 **Drupal** 故事“邪恶的 **HTTP Request** 和英勇的 **Drupal** 会话处理器”。



■**MATT WESTGATE** 从小就喜欢动手拆解各种小玩意，这对于他后来学习计算机并开始拆解各种开源软件，应该是有因果关系的。

Matt 是 **Lullabot** 的联合创始人。**Lullabot** 是一个培训和咨询公司，致力于帮助人们学习如何构建和架构一个 **web2.0** 的网站。**Lullabot** 与 **BBC, Participant Productions, Sony, MTV**，和 **the George Lucas Educational Foundation** 有很好的合作关系。

Matt 与他的妻子和两个女儿一起生活，他正在学习如何成为一个大想法推动者（**Big Mind facilitator**）。

技术审稿人

■**STEVE POTTS** 毕业于英国的 **Manchester** 大学，获得了计算应用学士学位，接着，在 **Open** 大学取得了商业和工业计算硕士学位。

甚至在他开始高等教育以前，他在防卫工业已经努力的工作过，将大量的失去抵抗力的软件压缩到一个很小的脚印中，这样数字化监视器现在就可以找到缩影了。他的工作涉及到上百个关于防卫、手持设备、移动通信、互联网的应用。

除了这些小心翼翼的工作以外（他的朋友会用其它词语来描述这些），他还是一个熟练的技术编辑，曾参与过 **Java, XHTML, PHP, 和 wireless** 技术图书的出版，包括 **Apress** 所有的 **Building Online Communities with Drupal, phpBB, and WordPress (Douglass, Robert T., Mike Little, and Jared W. Smith. Berkeley: 2005)**

Steve 创立了自己的技术咨询设备，**Free Balloon**，并且担任 **Hawdale Associates** 的 **CTO**，一个位于英国 **Manchester** 的吸引使用和设计用户体验的公司。

■**ROBERT DOUGLASS**， **Building Online Communities with Drupal, phpBB, and WordPress** 一书的共同作者。，他是 **Drupal Association** 成员，**Lullabot** 的顾问。他还是多个 **Drupal** 模块的作者和维护者，也是 **Drupal** 内核的经常贡献者。

序言

生活中的有些事情，你没有打算，但是却发生了。你可能去逛一个杂货店，没有找到你要买的东西，但是却碰到了你的爱情。这不是你所计划的了。

当我还是一个学生的时候，我需要一个基于 **Web** 的小工具来和朋友交流。由于我不满意现有的工具，我开始自己创建一个。这个工具后来就发展成了 **Drupal** 这个内容管理系统。

以前这个项目纯属个人爱好，其代码量也很小，现在已有成千上万的站点在使用 **Drupal** 了，包括一些在世界范围内非常流行的一些站点。**Drupal** 的发展是我没有预料到的，这是一个让人难以置信的想法——这不是因为 **Drupal** 作为一个平台或者工程时的缺点。不，**Drupal** 是一个伟大的系统，它拥有一个出色的社区，在这里人们对其发展做出了卓越的贡献。每一天，**Drupal** 都在证明它是一个可靠的系统，用来构建稳定的、可扩展的、易用的强大的 **web** 应用。它非常简单，这是由于我从来没有想过 **Drupal** 能有这么成功。在 **Drupal** 发展中，有一系列的没有预料到的难以置信的事情发生，这让我感到非常惊讶。

当我开始编写 **Drupal** 时，我在计算机面前呆上了无数个日夜，努力去创建一个更好的基于 **web** 的工具。更少的代码和完美优雅的架构是我追求的目标。完全关注在 **Drupal** 的代码和架构上，我想创建一个伟大的软件，而不是一个流行的软件。这说明了，如果一件东西值得使用，那么它将吸引更多的注意并被广泛的使用。

当我毕业后，我开始努力将 **Drupal** 从一个小规模的交流工具向更广泛的方面进行扩展。我使用它构建了 **drop.org** 站点，它是一组博客，用来追踪感兴趣的 **web** 技术。除此以外，**drop.org** 还是我探索新事物的试验场，比如 **RSS** 种子，内容版本，论坛，等等。到 2000 年时，**drop.org** 吸引了一批追随者，人们对我的试验非常有兴趣；他们开始提出建议并想参与到开发过程中。不久之后，在 2000 年 1 月 15 日，我将 **Drupal** 免费开源。

从此以后，人们可以免费的下载 **Drupal** 了。**Drupal** 采用了 **GNU** 通用公共许可证，任何人都可以运行、复制、和修改 **Drupal**，甚至可以发布修改了的版本——当然其他人对此也有相同的权利。

将 **Drupal** 免费开源，是一个非常重大的决定。使用 **Drupal** 的关键好处不是它的易用性或者它的功能，尽管这些也很重要。**Drupal** 的核心价值在于这个项目是开放的，而且很容易就可获取到，而对于你使用 **Drupal** 做什么则基本没有限制。将 **Drupal** 与其它系统区别开来的是它的繁荣的社区，一个完全公开和透明的产品。**Drupal** 社区保证了 **Drupal** 的成功，

而作为一个社区，我们开发 **Drupal** 的方式就是 **Drupal** 这么成功的原因。

考虑这么一个问题——对于一个商用 **CMS** 或者一个其它的商用软件，你有多大的机会成为这方面的世界顶级的专家？除非你在拥有该软件的公司工作，并且有权访问相关文档，或者参加更高层次的公司内部会议，否则机会渺茫——因为你不能获取所有的内部信息。

将这一场景与 **Drupal** 开发相比。作为一个开发者，你可以访问 **Drupal** 的所有源代码。你可以阅读所有的关于任何设计决定的相关讨论，而且你能够接触到世界上最好的 **Drupal** 开发者。事实上，没有任何东西来阻止你成为世界顶级的 **Drupal** 开发者。唯一的限制是你有没有这个决心。

这些想法有点老生常谈了——自由软件运动已经开始了一段时间了——但是这的确解释了为什么我会对这本 **Drupal** 书籍如此兴奋。**Drupal** 专业开发指南（**Pro Drupal Development**）将帮助更多的人来学习和使用 **Drupal**。如果说 **Drupal** 社区到目前为止有任何缺失的话，那么就是缺少一本伟大的 **Drupal** 书籍，通过编写此书，**John** 和 **Matt** 为 **Drupal** 做出了传说中的贡献。这也是我预料之外的。

Dries Buytaert

Drupal 创始人和项目负责人

译者：葛红儒， **Eskalate** 科技公司

绪论

译者：葛红儒 **Eskalate** 科技公司

程序员的学习历程就是一个非常有趣的旅程。首先是，分别的去学习、摸索一个软件系统的各个独立的子模块，通过对这些模块的学习来理解整个系统。当你达到了一定的程度以后，接着你就开始研究系统的内核，尝试着编写自己的代码来操纵系统的行为。这就是我们如何学习的一一多读别人的代码、多写自己的代码。

你坚持这一模式一段时间以后，你发现自己达到了一个新的高度，你可以从头构建一个自己的系统了。例如，你自己编写了一个内容管理系统，并把它部署到多个站点上，这样是不是很酷，觉得自己撬动了地球，成为了第二个戴志康。

但是接着又会达到一个关键点，当你发现对你系统的维护比构建该特性所耗费的时间还要多时，你就到达了这一关键点。你想根据自己现在所掌握的知识来从头构建整个系统。你还发现，许多其它的系统出现了，你的系统能做的，它们能做甚至做的更好，你的系统不能做的，它们也能实现。而且它们还有一个社区，在这里，来自全球各地的开发者在一起努力的改进着该软件，这时，你终于发现，这些系统在大多数方面都优于你自己的系统。更让人难以置信的是，这个软件是免费的，并且是开源的。

这就是我们的经历，我想你可能也会遇到类似的情况。旅程的终点让人感到欣慰一一成千上万个开发者在为同一个项目而努力。在这里，你找到了朋友；你编写了自己的模块；最重要的，和你自己单打独斗的时候一样，你仍然感觉到自己在做一件有意义的事情。

这本书适用于 **3** 种读者。首先，这里有大量的插图，包括各种图表和流程图；还有许多内容摘要，这为想了解 **Drupal** 是什么，**Drupal** 能做什么的初学者提供了方便。其次，本书包含了大量的代码片段和样例模块。这适用于有些基础，想在 **Drupal** 框架之上做定制开发的读者。我们建议你，安装 **Drupal**，在阅读本书的同时动手实践这些例子（最好再有一个调试器），这样你就会熟悉 **Drupal**。最好，本书包含了大量的评论、提示、还有对代码图片的详细解释，这将整本书有机的联系到了一起。这适用于想成为 **Drupal** 高手的人。

如果你是初学者，我们建议你从头逐章学习本书，因为前面的是基础，是后面章节的预备条件。

最后，你可以从 koobe.net 下载本书，以及样例代码，从 <http://drupalbook.com> 或者 www.apress.com 你可以下载到各种截图。

致谢

首先，感谢我们的家庭，在编写过程中，他们的理解和支持，特别是真诚的承担了更多的家庭义务，都给与了我们莫大的鼓舞。

Drupal 是一个基于社区的工程。如果没有这么多人的共同努力，编写文档，提交错误报告，创建和检查改进，**Drupal** 就不会像今天这样成功，当然也就不会有这本书了。但是在这么多人当中，请允许我们感谢那些对本书做出贡献的人。

它们包括 **#drupal IRC (internet relay chat)** 栏目所有成员，他们耐心的回答各种问题，包括 **Drupal** 是怎么工作的，为什么要用特定的方式编写代码，以及为什么有些代码是好的而另一些为什么是坏的，等等。还有，很多人对我们的手稿提出了这样或者那样的改进意见，这增加本书的可读性，在这里对这些人表示感谢。这包括，**Bert Boerland, Larry Crell, Robert Douglass, Druplicon, Kevin Hemenway, Chris Johnson, Rowan Kerr, Bèr Kessels, Gerhard Killesreiter, Jonathan Lambert, Kjartan Mannes, Tim McDorman, Allie Micka, Earl Miles, David Monosov, Steven Peck, Chad Phillips, Adrian Rossouw, James Walker, Aaron Welch, Moshe Weitzman, 和 Derek Wright**。对于那些做出贡献，而在此漏掉的人表示歉意。

Ted Serbinski, Nathan Haug, Jeff Eaton, Gábor Hojtsy, 和 Neil Drumm 认真的评论了本书的全部或部分手稿，在这里，对他们表示特别的感谢。

感谢爱荷华州立大学 (**Iowa State University**) 的 **Jon Tollefson**，和 **Lullabot** 的 **Jeff Robbins**，感谢他们对本书的支持。

感谢 **Apress** 小组在样例代码需要不断修改时的容忍和理解，以及不可思议的将我们的手稿出版成书。

最后，感谢 **Dries Buytaert** 将 **Drupal** 贡献给了整个世界。

第 1 章 Drupal 工作原理 (1)

第一章 Drupal 是如何工作的

在这一章，我们为你给出一个 **Drupal** 的概貌。关于系统的每一部分如何工作的详细信息将在以后章节中介绍。在这里，我们将涉及到 **Drupal** 运行所用到的技术堆栈，构成 **Drupal** 的各个文件，和各种不同的概念术语，比如节点，钩子，区块和主题。

什么是 Drupal ?

Drupal 是用作建设网站的。它是一个高度模块化，开源的 **web** 内容管理框架，它重点建立在合作之上的。它是一个可扩展的，适应标准的，并努力保持简洁代码和较小脚本的系统。**Drupal** 发布版中包含基本的核心功能，其他的额外功能可通过安装模块来获得。**Drupal** 被设计为可被定制的，但是定制是通过覆写核心功能或者增加模块来完成的，而不是修改核心组件中的代码。它同样成功的将内容管理和内容表示两者分离。

Drupal 可以被用来建立一个 **Internet** 门户；一个个人的，部门的，或者公司的网站；一个电子商务站点；一个资源分类站点；一个在线报纸；一个图库；一个内部网，这里仅提到了一部分。它甚至可被用来教授一个远程学习课程。一个专注于安全方面的小组，通过对威胁的反应和发行安全更新来保证 **Drupal** 的安全性。还有一个繁荣的社区组织，包括用户，站点管理员，设计者，和 **web** 开发者,非常努力的工作着，以持久的改进软件。可参看 <http://drupal.org> 和 <http://groups.drupal.org>。

技术堆栈 (Technology Stack)

Drupal 的设计目标是既可以运行在廉价的 **Web** 主机上，也可以适应大量运算的分布式站点。前一目标意味着使用最流行的技术，而后者则意味着仔细的严格的编码。**Drupal** 的技术堆栈如图 1-1 所示。

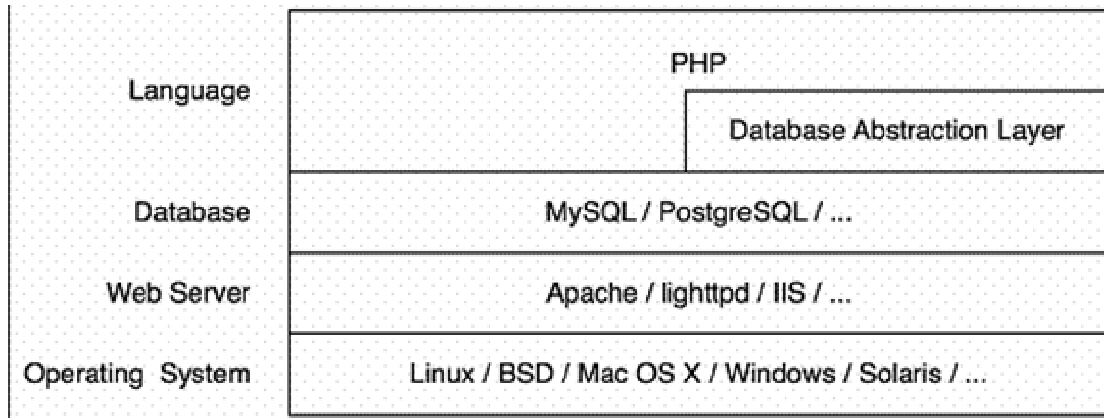


图 1-1 Drupal 的技术堆栈

操作系统位于技术堆栈的最底层，Drupal 可以不用太关注这一层。在任何支持 PHP 的操作系统上都可以成功的运行 Drupal。

Drupal 最常用的 web 服务器是 Apache，当然也可以使用其它的 web 服务器（包括微软的 IIS）。由于 Drupal 和 Apache 的这种长期的友好关系，所以在 Drupal 的根目录自带了一个 .htaccess 用来确保 Drupal 安装的安全性（如果你使用的是一个其它的 web 服务器，而不是 Apache，你一定要将 .htaccess 中的规则转化为你系统可以理解的语句）。可以使用 Apache 的 mod_rewrite 模块来实现简洁 (Clean) URLs---将 URL 中的“?”、“&”以及其它奇怪的符号清除掉，在 Drupal 中去掉的是“?q=”。这一点特别重要，当从其他的内容管理系统或者静态文件中迁移到 Drupal 上时，依照 Tim Berners-Lee

（<http://www.w3.org/Provider/Style/URI>），内容的 URL 不需要改变，而不改变的 URIs 则非常酷。

Drupal 使用一个轻量级的数据库抽象层与堆栈的下一层次（数据库层）进行交互。这一抽象层处理 SQL 查询语句的清洁工作，并使得可以使用不同厂商的数据库而不用重构你的代码。在 Drupal 中最常用的数据库是 MySQL 和 PostgreSQL。

Drupal 使用的编程语言是 PHP。PHP 的名声比较坏，这是因为 PHP 比较好学，这样大量的 PHP 代码都是由新手编写的。和许多其它的编程语言一样，PHP 也常被滥用或者用于快速实现的系统中。然而，也可以用 PHP 来编写可靠的代码。所有的 Drupal 内核代码都严格的遵守了编码规范(<http://drupal.org/nodes/318>)。

内核 (Core)

Drupal 的内核有一个轻量级的框架组成。当你从 drupal.org 下载 Drupal 时就得到了这一内核。它负责提供基本的功能用以支持系统的其它部分。

内核包括当 **Drupal** 接到请求时所要调用的系统引导指令的代码，一个 **Drupal** 常用函数库，和提供基本功能的模块比如用户管理、分类、和模板，如图 1-2 所示。

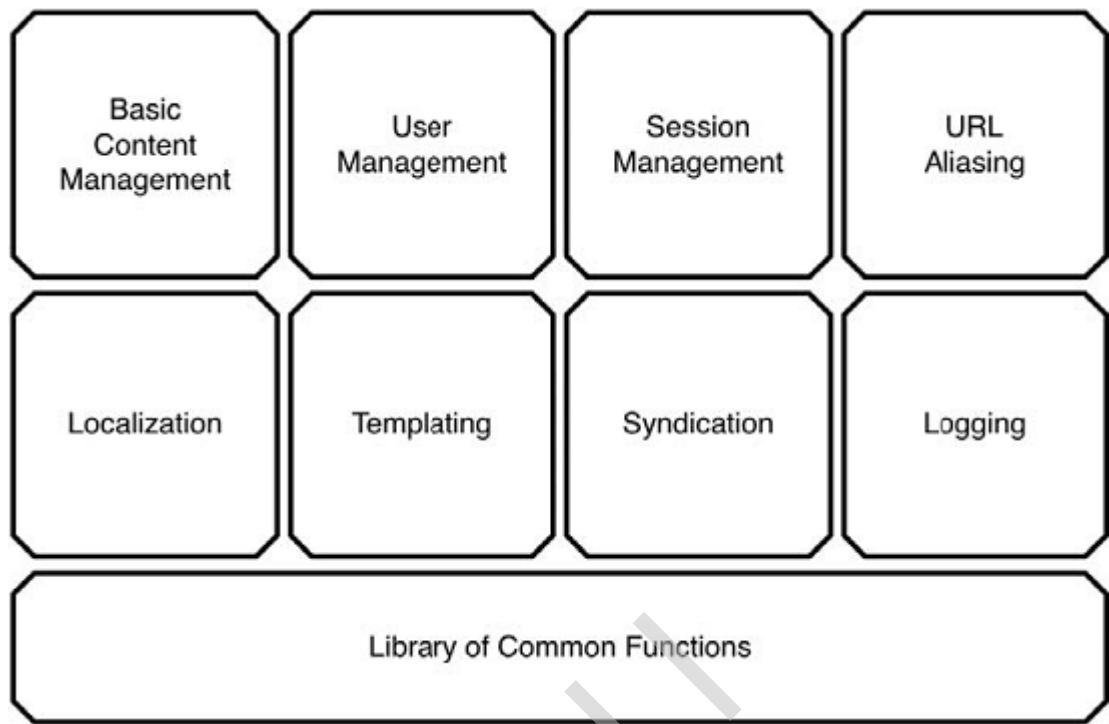


图 1-2 **Drupal** 内核的概貌（没有展示完所有的核心功能）

后台管理接口 (Administrative Interface)

Drupal 的后台管理接口与站点的其它部分紧密的集成在了一起，而且默认情况下，使用相同的主题。第一个用户，也就是用户 1，是一个对站点拥有完全权限的超级用户。以用户 1 的身份登录后，你将在你的用户区块（参看“区块”部分）中看到管理站点的一个链接。点击这一链接，你将进入到 **Drupal** 的后台管理接口页面。根据用户对站点访问权限的不同，每个用户的区块都会有一个不同的链接。

模块 (Modules)

Drupal 是一个完全模块化的框架。功能都包含在模块中，而模块可以被启用或者禁用（一些必须的模块不能被禁用）。来向 **Drupal** 站点添加特性有 3 种方式：启用已存在的模块（核心模块），安装 **Drupal** 社区成员编写的模块（第 3 方模块），编写自己的模块。这样，可以根据站点的需要来添加相应的模块，需要的功能少，所需的模块也就少，需要很多功能，就添加多一些的模块。如图 1-3 所示。

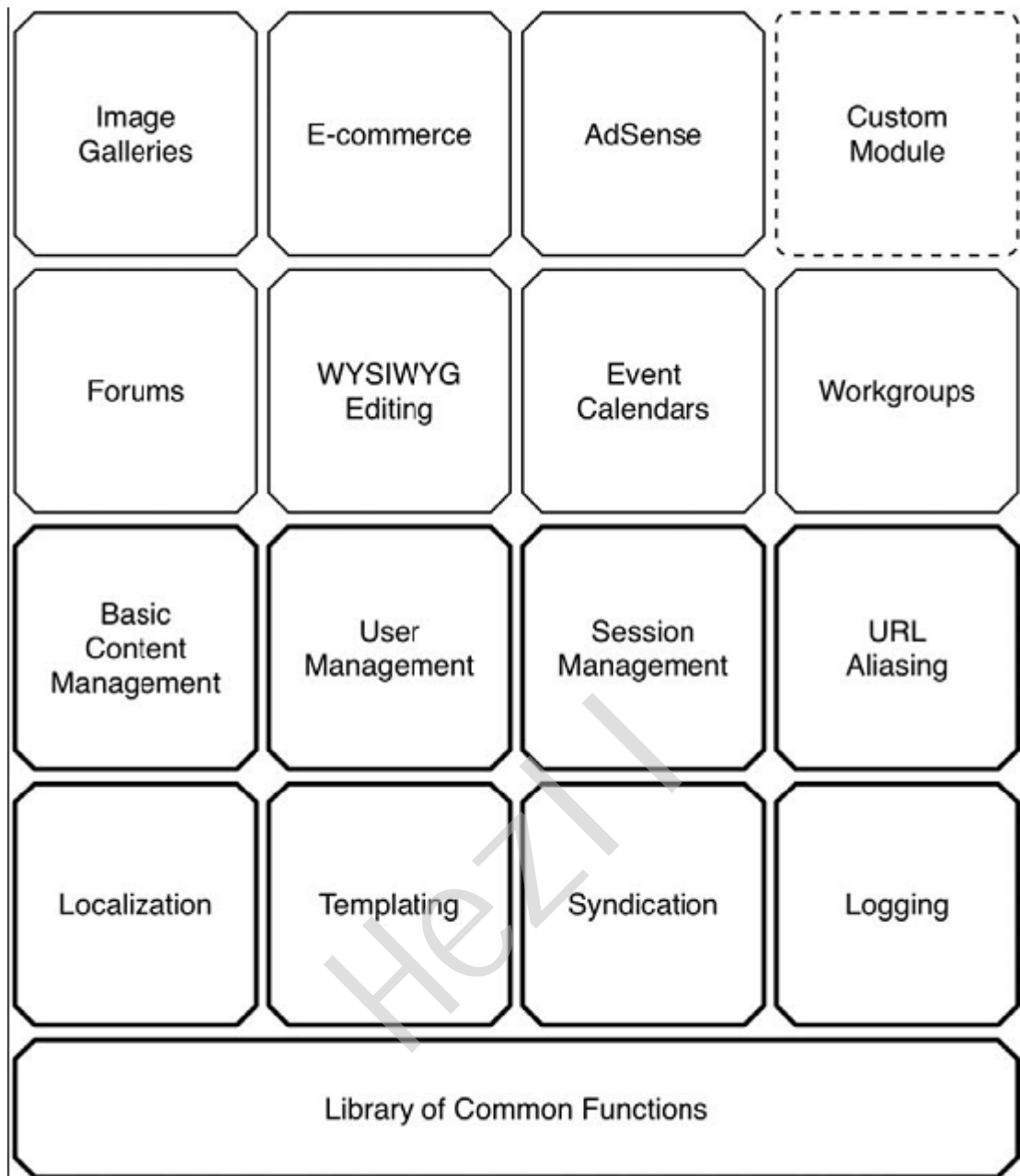


图 1-3 通过启用其它的模块来添加更多的功能

新添的内容类型比如处方、日志、或者文件，新添的行为比如 e-mail 通知、P2P 发布、和聚合，等等都是通过模块实现的。Drupal 使用了反转控制（inversion of control）设计模式，框架只在合适的时间调用相应的模块功能。这些为了模块完成它们的任务提供的机会被称为钩子。

钩子 (Hooks)

可以把钩子看做 **Drupal** 的内部事件。它们也被称为回调函数，这是由于他们是根据函数命名约定来构建的，而不是注册一个事件监听器（**listener**），它们也不是真的被回调。模块使用钩子挂在 **Drupal** 的其余部分上。

假定一个用户登录到了你的 **Drupal** 站点上。在用户登录时，**Drupal** 调用用户钩子。这意味着将调用所有的根据约定——“模块名”+“钩子名”——创建的函数都将被调用。例如，评论模块中的 `comment_user()`，本地化模块中的 `locale_user()`，节点模块中的 `node_user()`，还有任何其它具有类似名称的函数都将被调用。如果你编写了一个名为 `spammy.module` 的定制模块，其中包含一个名为 `spammy_user()` 的用来向用户发送 e-mail 的函数，那么你的这个函数也将被调用，倒霉的用户每次登录都将收到一封不请自来的 e-mail。

接近 **Drupal** 的核心功能的最常用的方式就是在模块中实现钩子。

提示 更多关于 **Drupal** 所支持的钩子的信息，参看在线文档

<http://api.drupal.org/api/5>，并查看“**Drupal** 的组成部分”（“**Components of Drupal**,”），接着“模块系统（**Drupal** 钩子）”（“**Module system (Drupal hooks)**”）。

主题（Themes）

当创建一个发送给浏览器的 **web** 页面时，实际主要考虑两点：聚集合适的数据和为 **web** 装饰这些数据。在 **Drupal** 中，主题层负责创建浏览器接收到的 **HTML**。**Drupal** 可以使用多种流行的模板方式，比如 **Smarty**，**PHP** 的模板属性语言（**Template Attribute Language for PHP (PHPTAL)**），和 **PHPTemplate**。

这里需要记住的重点是 **Drupal** 提倡将内容和显示分开。

在 **Drupal** 中可以使用多种方式来为你的网站定制外观。最简单的方式是使用 **CSS** 来覆盖 **Drupal** 内置的类和 **ID**。然而，如果你不想局限于此，并且想定制实际的 **HTML** 输出时，你将发现很容易就可以达到你的目的。**Drupal** 的模板文件有标准的 **HTML** 和 **PHP** 组成。另外，**Drupal** 页面的每个动态部分（比如盒子、列表、或者面包屑痕迹），都可以通过声明一个具有合适名字的函数来覆盖掉。接着，**Drupal** 将使用你声明的函数。

节点（Nodes）

Drupal 中的内容类型都源于一个称为节点的单独的基本类型。无论它是一篇日志、一个处方，或者甚至一个工程任务，它的底层数据结构都是相同的。这一方式背后的优势是它的

扩展性。模块开发者可以为节点添加特性比如级别、评论、文件附件、地理位置信息等等，而不用担心节点的类型，无管它是日志、处方还是其它。站点管理员可以根据内容类型混合和匹配功能；例如在日志而不是在处方上启用评论，或者仅为工程任务启用文件上传功能。

节点还包含了一个行为属性基本集，而所有其他的内容类型都继承了这一基本集。任何节点都可以被提升到首页、发布或者未发布，或者甚至被搜索。而且由于这个统一的结构，后台管理接口为节点提供了一个批量编辑的页面。

区块 (Blocks)

区块是在你网站模板的特定位置可以启用或者禁用的信息。例如，一个区块可以展示你站点当前活跃用户数。你可能使用一个区块来展示最活跃的用户，或者即将来临的事件列表。区块一般放置在模板中的左右栏、页首、或者页尾中。区块也可以用来展示特定类型的节点，一般仅用于首页，或者由于其它标准才这样做。

区块常常用于为当前用户展示定制的信息。例如，一个导航区块仅包含当前用户有权访问的链接。可以通过后台管理接口页面对区块的位置和显示进行管理。

第 1 章 Drupal 工作原理 (2) 对请求提供服务收藏

文件布局 (File Layout)

理解默认 **Drupal** 安装的目录结构, 能够帮助你调试你的站点, 并教你一些重要的最佳实践, 比如下载的模块和主题的放置位置, 如何拥有不同的 **Drupal** 轮廓 (**profile**)。一个 **Drupal** 默认安装的目录结构如图 1-4 所示。

文件夹目录中的每一元素的详解如下:

files: **Drupal** 默认不带有这个文件夹, 但是如果你想使用一个定制的 **logo**, 启用用户头像, 或者在你的站点上上传其它媒体文件时, 你需要这个文件夹。运行 **Drupal** 的 **web** 服务器需要具有对这个子目录进行读和写的权限。

includes : 包含了 **Drupal** 常用的函数库。

misc: 用来存储 **JavaScript**, 和在 **Drupal** 安装中备用的各种图标和图片。

modules: 包含所有的核心模块, 其中每个模块位于它自己的文件夹下。最好不要乱动这个文件夹下面的任何东西 (你添加的其他模块放到 **sites** 目录下)

profiles: 包含一个站点的不同安装轮廓 (**profile**)。如果在这个子目录下面, 除了默认的轮廓 (**profile**) 以外, 还有其它的轮廓 (**profiles**), 那么在你第一个安装 **Drupal** 站点时 **Drupal** 将向你询问想要安装哪一个轮廓 (**profile**)。安装 **profile** 的主要目的是自动的启用核心的或者第 3 方的模块。比如一个电子商务轮廓 (**profile**) 的示例, 它将自动把 **Drupal** 作为一个电子商务平台进行安装。

scripts: 包含了许多脚本, 这些脚本可用于检查语法, 清洁代码, 使用 **cron** 处理特定情况等等。在 **Drupal** 的请求生命周期中用不到它; 里面有一些 **shell** 和 **Perl** 的有用脚本。

sites: 包含了你对 **Drupal** 进行的修改, 包括设置、模块、主题等形式(参看图 1-5)。你从第 3 方模块库中下载的模块, 或者你自己编写的模块, 都放在 **sites/all/modules** 下面。这使得你对 **Drupal** 所进行的任何修改都保存在单个文件夹里。在目录 **sites** 下面有一个名为 **default** 的子目录用来放置你 **Drupal** 站点的默认配置文件--- **settings.php**。通常拷贝 **default** 目录并使用你站点的 **URL** 对其重命名, 这样你的设置文件就位于

sites/www.example.com/settings.php.

themes: 包含了 Drupal 的模板引擎和默认主题。

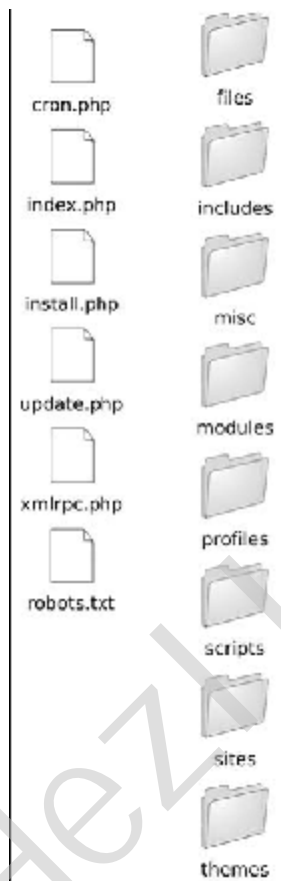


图 1-4 Drupal 安装的默认目录结构

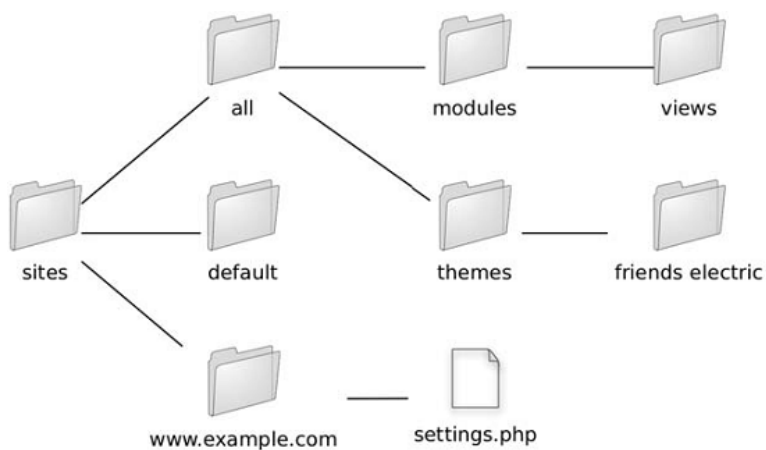


图 1-5 sites 文件夹用来存储你的所有 Drupal 修改

对一个请求提供服务 (Serving a Request)

当 **Drupal** 收到一个请求时都发生了什么，如果对此有一个概念上的框架，这对以后很有帮助，所以本部分将为这一框架提供一个快照。如果你自己也想追踪它的话，使用一个好的调试器，从 **index.php** 开始，**Drupal** 的大多数请求都从这里开始。对于展示一个简单 **web** 页面，这里所列的顺序看起来有些复杂，但这是灵活性所必需的。

Web 服务器的角色

Drupal 运行在一个 **web** 服务器上，通常是 **Apache** 上。如果 **web** 服务器识别 **Drupal** 的 **.htaccess** 文件，那么将初始化一些 **PHP** 设置，并启用简洁 (**clean**) **URL**。

注意 **Drupal** 支持简洁 (**clean**) **URL**，也就是像 <http://example.com/foo/bar> 的一样的 **URL**。**Drupal** 的 **.htaccess** 文件中的 **mod_rewrite** 规则将这一路径转换为 **index.php?q=foo/bar**。所以在内部，无论是否启用了简洁 (**clean**) **URL**，**Drupal** 总是使用相同的路径（存储在 **URL** 查询参数 **q** 中）。在这里，内部路径应该为 **foo/bar**。内部路径也被称为 **Drupal** 路径。

在备选的 **web** 服务器中，比如微软的 **IIS**，可以使用一个 **ISAPI** 模块比如 **ISAPI_Rewrite** 来实现简洁 **URL**。

引导指令流程 (The Bootstrap Process)

对于每个请求，**Drupal** 通过一系列的引导指令阶段来引导它自己。这些阶段在 **bootstrap.inc** 中定义，接下来的部分描述了处理流程。

配置 (Configuration)

在这一阶段将填充 **Drupal** 的内部配置数组，并建立站点的基础 **URL (\$base_url)**。通过 **include_once()** 来解析 **settings.php** 文件，任何已被覆盖的变量将被应用。

早期页面缓存 (Early Page Cache)

当在需要更高的性能水平时，甚至在试图建立数据库连接之前可能就需要调用缓存系统了。早期页面缓存阶段让你 (使用 **include()**) 包含一个含有名为 **page_cache_fastpath()**

的函数的 **PHP** 文件，该函数接收并返回内容给浏览器。通过将 `page_cache_fastpath` 变量设置为 **TRUE**，就可以启用早期页面缓存阶段了，而包含进来的文件是通过将 `cache_inc` 变量设置为文件的路径来定义的。

数据库 (Database)

在数据库阶段期间，将决定数据库的类型，将建立初始链接以供数据库查询使用。

访问 (Access)

Drupal 支持基于一个主机名一个 **IP** 地址来禁止主机（对站点的访问）。在访问阶段，将快速的检查请求是否来自一个被禁的主机；如果是，那么将拒绝访问。

会话 (Session)

Drupal 利用了 **PHP** 内置的会话处理，但是它使用它自己的基于数据库的会话处理器来覆盖 **PHP** 的一些处理器。在会话阶段，将初始化或者重新构建会话。

后期页面缓存 (Late Page Cache)

在后期页面缓存阶段，**Drupal** 加载足够的支持代码来决定是否需要从页面缓存中提供一个页面。这包括，把来自于数据库的设置合并到在配置阶段创建的数组中，加载或者解析模块代码。如果在会话中显示请求来自于匿名用户并且启用了页面缓存，那么将从缓存中返回页面，执行到此停止。

路径 (Path)

在路径阶段，将加载处理路径和路径别名的 (**aliasing**) 的代码。该阶段使得用户可读的 **URL** 被转化为 **Drupal** 路径，并处理内部 **Drupal** 路径的缓存和查询操作。

完整 (Full)

该阶段是引导指令的最后一个阶段，它包括加载一个通用函数库，主题支持，和支持回调映射，文件处理，Unicode，PHP 图片工具集，表单的创建和处理，自动排序的表格，和结果集的分页。在这里将设置 Drupal 定制的错误处理器，设置本地化，并加载所有启用了的模块。最后 Drupal 调用 `init` 钩子，这样在对请求正式开始处理以前，将有机会通知相应的模块。

一旦 Drupal 的整个引导指令完成了，那么框架中的所有部分现在都可以使用了。现在是时候获得浏览器的请求并将它委托给一个处理它的函数。在 URLs 和处理它们的函数之间的映射，是使用一个回调登记来完成的，这个回调登记负责 URL 映射和访问控制。模块使用菜单钩子来注册它们的回调函数（更多信息，参看第 4 章）

当 Drupal 为浏览器请求的 URL 找到一个存在的回调函数，并且用户有权访问该回调函数，那么控制权将转移给回调函数。

处理一个请求 (Processing a Request)

回调函数做了流程要求做的事情，并收集满足请求所需要的数据。例如，收到一个对内容的请求比如 <http://example.com/q=node/3>，URL 将被映射到 `node.module` 里面的函数 `node_page_view()`。进一步的处理将从数据库中取回该节点的数据并将它放到一个数组中。接着，就到了主题化的时候了。

主体化数据 (Theming the Data)

主体化涉及到将已被取回，操作或者创建的数据转化为 HTML。Drupal 将使用管理员选用的主题来为网页提供一个合适的外观，并将生成的 HTML 移交给 web 浏览器。

总结

读完本章后，你应该能大致理解 Drupal 的工作原理，并对当 Drupal 为一个请求服务时都发生了什么有个概念。组成网页服务处理流程的个部分将在后面章节中详细介绍。

第 2 章 创建一个模块 (1)

创建一个模块 (Module)

在许多开源的应用中，你可以通过修改源代码来定制你的应用。这通常是一个获得你期望的行为的一种方式，但是在 **Drupal** 中，一般是不赞成这样做的，它被是做万不得已的一个手段。修改源代码，意味着随着每次 **Drupal** 的升级，你必须要做更多的工作---你必须测试以下你的定制代码还想期望的那样工作。代替的，**Drupal** 的设计从一开始便考虑到了模块化和可扩展性。

Drupal 是个精简的并且优雅的用于开发应用的一个框架，它的默认安装通常被称作为 **Drupal** 的核心。通过启用新的模块来向核心添加功能，这些模块是一些包含 **PHP** 代码的文件，放在你的 **Drupal** 安装路径 **sites/all/modules** 的子目录下。现在看一下这个子目录，现在你可以在你的 **Drupal** 站点上导航到 **Administer->Site building->Modules**，比较一下子目录下的模块与网页上的模块列表。

在本章，我们将从头开始创建一个模块；你将学到模块必须遵守的一些标准。我们需要一个真实的目标，所以让我们考虑以下现实中的注释问题。当用户在 **Drupal** 网站上浏览内容时，用户可能对内容发表评论（如果管理员开启了评论（**comment**）模块）。但是如果是在一个网页上添加一个注释（一种仅有用户可见得节点类型），那会怎么样？这对于私密的评价内容可能非常有用（我们知道这看起来有些做作，但是现在我们来试暂时忍受一下吧）。

创建相应的文件

首先我们要做的是为模块起一个名字，名字“**annotate**”看起来还是比较合适的一简洁并且生动。现在我们需要为它找一个放置的位置。让我们把它放在目录 **sites/all/modules** 下面来使它和核心模块区分开来。我们创建一个子目录，而不仅仅是一个 **annotate.module** 文件，这是应为在我们的模块发行版本中我们还需要一些其他的文件。比如我们需要一个 **README.txt** 文件，用来向其他用户解释我们的模块是做什么的，以及如何使用它，还有一个 **annotate.info** 文件用来向 **Drupal** 提供一些关于我们模块的信息。准备好了吗？

我们的 **annotate.info** 文件内容如下：

```
; $Id$
```

```
name = Annotate
```

description = Allows users to annotate nodes.

package = Example

version = "\$Name\$"

这个文件采用.ini 格式，这是一个用于 PHP 配置文件的简单标准(参看 http://php.net/parse_ini_file)。我们从一个版本管理系统(CVS)的标识符的标签开始，并且为

Drupal 的网站上模块管理部分提供了相应的名称 (**name**) 和描述 (**description**)。模块可以按组来展示，而组的划分是由包 (**package**) 决定的；这样，如果我们有 3 个不同的模块，他们都有 **package=Example**,那么他们将被放在同一个组中。版本的值是另一个 CVS 的标示标签。如果我们想和其他用户分享这一模块，通过将它放到 Drupal 的贡献模块库中，这个值将会被自动填充。

注意：你可能在想，为什么我们需要一个单独的.info 文件。为什么不在我们的主模块中写一个函数来返回这些元数据呢？这是因为在模块管理页面加载时，它将不得不加载并解析每一个模块，不管有没有启用，这比平时需要更多的内存并且可能超出分配给 PHP 的内存上限。

通过使用.info 文件，信息可以更快速的加载并且使用更少的内存。

现在我们准备好创建一个真实的模块了。在你的子目录 **annotate** 下面创建一个名为 **annotate.module** 的文件。在文件的开始出使用 PHP 的开始标签和一个 CVS 标示符标签，并紧跟一个注释：

```
<?php
// $Id$

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

首先，让我们看一下注释的风格。我们从/**开始，在接下来的每一行中缩进一格并以*开头，最后以*/结束。标记@file意味着在接下来的一行是对这个文件要做什么的一个描述。这一行的描述将被这样使用，Drupal的自动文档提取和格式化模块api.module用它来找出这个文件是做什么的。空了一行后，我们为可能检查（并且改进）我们代码的程序员提供了一个更长的说明。注意，我们在这里故意没有使用一个结束标签?>;这对于PHP来说是可选的，如果包含了它，就可能导致文件的尾部空格问题(参看<http://drupal.org/node/545>)。

注意：为什么我们在这里这么详细的讲述每一个细节？这是因为，如果来自世界各地的成千上百的人开发同一个项目的话，如果大家采用一种标准的方式的话，将会节省大量的时间。关于Drupal的代码风格的更详细的内容可以从Drupal的用户手册的“代码标准”一节中找到(<http://drupal.org/node/318>)。

保存你的文件，访问Administer->Site building->Modules.你的模块将会出现在列表中。多么激动人心啊。

下面我们要做的就是定义一些设置，这样我们就可以使用一个基于web的表单来选择哪些节点类型我们可以添加注释。这需要两步。首先我们定义一个我们可以访问我们设置的路径。然后我们创建设置的表单。

实现一个钩子

回想一下，Drupal是建立在一个钩子的系统之上，有时候称之为回调。在执行的过程中，Drupal询问模块看它们是不是想要做些什么。举例来说，当决定那一个模块负责当前的请求时，它向所有的模块询问是否提供提供相应的路径。通过制造一个所有模块的列表，并且调用每个模块中名为：模块名+_menu的函数，来做这件事。当它遇到我们的annotate模块时，它调用函数annotate_menu(),并向它传递一个参数。这个参数意味着这个模块的相应是否可被缓存。一般情况下，菜单项可以被缓存；我们将在第4章介绍例外情况，这章讲述了Drupal的菜单/回调系统。每一个菜单项是一个联合的数组。下面就是要向我们模块中添加的东西：

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

```
function annotate_menu($may_cache) {  
  
  $items = array();  
  
  if ($may_cache) {  
  
    $items[] = array(  
  
      'path' => 'admin/settings/annotate',  
  
      'title' => t('Annotation settings'),  
  
      'description' => t('Change how annotations behave.'),  
  
      'callback' => 'drupal_get_form',  
  
      'callback arguments' => array('annotate_admin_settings'),  
  
      'access' => user_access('administer site configuration')  
  
    );  
  
  }  
  
  return $items;  
  
}
```

此时不要在细节上过于担心。这段代码说，“当用户访问页面 <http://example.com/?q=admin/settings/annotate> 时，调用函数 `drupal_get_form`，并向它传递了一个表单 ID `annotate_admin_settings`”。当 Drupal 完成了向所有的模块询问它们的菜单项时，它将会找到一个菜单以从中选择一个函数用于相应所请求的路径。

注意，所有将被展示给用户的文本被放在了函数 `t()` 中，它将会在字符串翻译时调用。通过对所有的可展示文本使用字符串的翻译函数，这使得本地化你的模块为另一个不同语言时非常方便。

注意：如果你对于钩子机制很感兴趣的话，参看文件 `includes/module.inc` 里面的函数 `module_invoke_all()`。

现在你应该清楚我们为什么叫它 `hoo_menu` 或者菜单钩子了。Drupal 的钩子通过在钩子的名称前加上你的模块名来创建。

秘密消息: Drupal 的发展很快。一个支持的钩子的完全列表和它们的使用说明可在 Drupal 的 API 文档站点(<http://api.drupal.org>)上找到。

添加特定模块的设置

Drupal 有多种不同的节点类型, 比如 **Story** 和 **Page**。我们想将评论的使用限定在特定的一些节点类型上。为了实现它, 我们需要创建一个页面, 在这里我们告诉我们的模块那些节点类型我们想评论。向 **annotate** 模块添加下面的代码:

```
/**
 * Define the settings form.
 */
function annotate_admin_settings() {
  $form['annotate_nodetypes'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these node types'),
    '#options' => node_get_types('names'),
    '#default_value' => variable_get('annotate_nodetypes', array('story')),
    '#description' => t('A text field will be available on these node types to make
user-specific notes. '),
  );
  $form['array_filter'] = array('#type' => 'hidden');
  return system_settings_form($form);
}
```

在 Drupal 中, 表单被表示为一个嵌套的树状结构; 也就是说, 一个数组的数组。这个结构向 Drupal 的表单呈现引擎 (**rendering engine**) 描述了表单是如何表示的。为了可读性,

我们将数组中的每一个元素放到单独的一行里面。每一个指示都以“#”开头，它作为数组的键。我们首先声明了表单元素的类型为复选框，这意味着通过使用一个键入的数组来构建一个多选的复选框。我们为表单元素设置一个标题，像平常的一样使用了 `t()` 函数。然后将选项 (`options`) 赋值为 `node_get_types('names')`，它方便的返回了当前 Drupal 中可用的节点类型的键值数组。函数 `node_get_types('names')` 的输出看起来像这个样子：

```
'page' => 'Page', 'story' => 'Story'
```

数组的键对应于节点类型在 Drupal 中的内部名称，而把可读性的名称（显示给用户的）放到右边。如果你启用了 `Savory Recipe` 模块，数组看起来将像这样：

```
'page' => 'Page', 'savory_recipe' => 'Savory recipe', 'story' => 'Story'
```

因此，在我们的 web 表单中，就为 `page` 和 `story` 两种节点类型生成相应的复选框。下一个指示，`#default_value`，将是这个表单元素的默认值。由于 `checkboxes` 是一个多值的表单元素（即是说，存在多于一个的复选框），所以 `#default_value` 的值将会是一个数组。

`#default_value` 的值值得讨论一下：

```
variable_get('annotate_nodetypes', array('story'))
```

Drupal 允许程序员使用特定的一对函数：`variable_get()` 和 `variable_set()` 来存储和回显任意一个值。值被存储到了数据库表 `variables` 中，并且在处理一个请求的任一时候都是可用的。由于在处理每个请求时都会回显这些值，所以不能用这个方法存储大数量的数据。但是它是个非常方便的方式用来存储模块的配置属性值。注意我们向方法 `variable_get()` 中传递了一个对于值描述的键（所以我们可以取回来它），和一个默认值。在这种情况下，默认值是一个关于那些节点类型允许注释的数组。这样默认情况下，我们允许评论节点类型 `Story`。

最后我们提供一个描述，用来告诉站点管理员关于这个域的一些更细节的信息。

现在导航到 `Administer > Settings > Annotate`，我们将看到为 `annotate.module` 所显示的表单。

Users may annotate these node types:

page

story

A text field will be available on these node types to make user-specific notes.

图 2-1，为 `annotate.module` 生成的配置表单。

定义 `$form['array_filter']` 的这行代码看起来有点神秘。不过现在，我们只需要知道，当我们使用 `settings` 钩子的存储我们的复选框值的时候，它是必须的，者就可以了。

仅用了几行代码，我们为模块提供了一个可用的配置表单，它将自动的保存和记起我们的设置。好的，代码中的一行长度适中，但是仍然，这会给你一种使用 **Drupal** 的奇妙感觉。

第 2 章 创建一个模块 (2)

添加数据填充表单

为了使用户可以进入一个 **web** 页面的笔记部分，我们需要为它提供一个位置。下面我们为笔记添加一个表单：

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

function annotate_nodeapi(&$node, $op, $teaser, $page) {

switch ($op) {

case 'view':

global $user;

// If only the node summary is being displayed, or if the
// user is an anonymous user (not logged in), abort.
if ($teaser || $user->uid == 0) {

break;

}

$types_to_annotate = variable_get('annotate_nodetypes', array('story'));

if (!in_array($node->type, $types_to_annotate)) {

break;

}

// Add our form as a content item.

$node->content['annotation_form'] = array(
```

```
'#value' => drupal_get_form('annotate_entry_form', $node),  
  
'#weight' => 10  
  
);  
  
}  
  
}
```

这个例子看起来有些复杂，所以让我们详细的分析一下。首先，注意到我们在这里实现了 **Drupal** 的另一个钩子。这次是 `_nodeapi`，在 **Drupal** 对节点进行各种处理的时候将会调用它，所以其他模块（比如我们的）在处理过程继续之前可以修改节点。我们使用变量 `$node` 来表示一个节点。注意第一个参数前面的 `&` 意味着对于 `$node` 对象这里使用参数传递，这非常棒，因为我们在这里对 `$node` 所做的任何修改将被保存下来。我们的目标是追加一个表单，所以我们非常高兴我们可以修改节点。

我们仍然需要一些在我们的代码被调用时 **Drupal** 内部将会发生什么的信息。这些信息保存在参数 `$op` 中了，它可以是 `insert`（节点被创建），`delete`（节点被删除），或者一个其它的值。当前，我们仅对当节点被准备显示出来时，修改节点感兴趣。在这种情况下，变量 `$op` 的值为 `view`。我们在这里是用了 `switch` 控制语句，这样我们可以非常容易的看到在每种情况下我们的模块将会做什么。

接下来，我们快速的检查了一些我们不想呈现注释域的情况。一种情况是当参数 `$teaser` 为真时。如果它为真，这意味着，这个节点并不是单独呈现了，而是呈现在一个列表中，比如说一个搜索引擎的结果中。在这种情况下，我们不需要添加任何东西。另一种情况是 `$user` 对象的用户 `ID` 为 `0` 时，这意味着用户没有登录。（注意，在这里我们是用了关键字 `global` 来把 `$user` 对象引进来）。我们使用 `break` 语句来跳出 `switch` 语句从而阻止修改页面。

在我们为 `web` 页面添加注释表单的以前，我们需要检查一下，将要进行显示的节点的类型是不是我们在设置页面所设置的类型中的一个，所以我们回显了在我们实现的设置钩子中所保存的节点类型，并将它保存到了具有可读性的变量 `$types_to_annotate` 中去。对于 `variable_get()` 中的第 2 个参数，我们在这里声明了一个默认数组，用于当站点管理员还没有访问我们的模块的配置页面并设置配置属性的情况。下面要做的就是检查一下，我们所要处理的节点的类型是不是确实是 `$types_to_annotate` 中的一种。同样，我们使用了 `break` 语句来放弃，如果节点类型不是我们想要注释的时候。

我们最后要做的就是创建表单，并把它添加到**\$node** 对象的内容属性中去。首先，我们需要定义一个表单，这样我们就有了添加的东西。我们将在一个单独的函数中完成这件事，它的唯一责任就是定义表单：

```
/**
 * Define the form for entering an annotation.
 */
function annotate_entry_form($node) {
  $form['annotate'] = array(
    '#type' => 'fieldset',
    '#title' => t('Annotations')
  );
  $form['annotate']['nid'] = array(
    '#type' => 'value',
    '#value' => $node->nid
  );
  $form['annotate']['note'] = array(
    '#type' => 'textarea',
    '#title' => t('Notes'),
    '#default_value' => $node->annotation,
    '#description' => t('Make your personal annotations about this content here. Only you (and the site administrator) will be able to see them.')
  );
  $form['annotate']['submit'] = array(
    '#type' => 'submit',
```

```
'#value' => t('Update')  
  
);  
  
return $form;  
  
}
```

和我们在函数 `annotate_admin_settings()` 使用的方式一样, 通过创建一个键入的数组我们创建了表单—只是在这次我们把我们的文本输入框和提交按钮放到了 `fieldset` 中去, 这样他们在 `web` 页面中就组织到了同一个组下。首先, 我们创建一个数组, 它的 `#type` 为 `fieldset`, 并为它提供了一个标题。然后我们创建文本域数组。注意, 文本域数组的数组键是 `fieldset` 数组中的一员。换句话说, 我们使用 `$form['annotate']['note']` 替换了 `$form['note']`。这样, `Drupal` 将把这个文本域元素当作 `fieldset` 元素中的一员。最后, 我们创建了提交按钮, 然后返回了定义我们表单的数组。

现在让我们回到 `annotate_nodeapi()` 上, 通过向节点的内容添加一个 `#value` 和一个 `#weight`, 我们将创建的表单追加到了页面的内容上。值包含了要展示的内容, 重量告诉 `Drupal` 把它展示到哪里。我们想把我们的注释表单放到页面的下面, 所以我们为它分配了一个相对较大的重量 `10`。我们要展示的是我们的表单, 所以我们调用函数 `drupal_get_form()` 来将我们的表单从一个描述如何创建它的数组的形式转化为最终的 `HTML` 表单。注意, 在这里我们事如何将 `$node` 对象传递到我们的表单函数中去的。我们需要这样做来得到以前的注释并把它预先填充到表单中。

使用你的 `web` 浏览器查看一个页面, 你应该可以看到这个用于追加的注释表单 (如图 2-2 所示):



The image shows a screenshot of a web form titled "Annotations". Inside the form, there is a section labeled "Notes:" followed by a large, empty text input field. Below the input field, there is a line of text that reads: "Make your personal annotations about this content here. Only you (and the site administrator) will be able to see them." At the bottom of the form, there is a button labeled "Update".

图 2-2 出现在 `drupal web` 页面上的注释表单

当我们点击 `Update` 按钮时, 将会发生什么呢? 什么都没有, 因为我们还没有为它写任何代码呢。现在就让我们添加这些代码。但是在我们出发以前, 我们不得不考虑一下我们将把用户输入的数据存储到哪里。

把数据存储到数据库表中

存储一个模块的数据的最常用的方式是为这个模块创建一个单独的数据库表。这将使得模块的数据与 **drupal** 核心数据表区分开来。当决定为你的模块创建那些字段时，你应该问自己：你需要存储哪些数据？如果我对这个表进行查询时，我将需要什么？最后，还要为我的模块考虑将来的有哪些计划？

我们需要存储的数据仅仅为注释的文本，注释所用到的节点的数字 **ID**，和填写注释的用户的用户 **ID**。保存一个时间戳也会非常有用，这样我们可以按照时间戳将最近更新的注释排成一列展示出来。最后，我们查询这张表的主要问题是，“这个用户对这个节点做了哪些注释？”我们将在 **uid** 和 **nid** 字段上创建一个复合的索引，从而使我们最常用的查询尽可能的快。用来创建我们的表 **sql** 语句如下所示：

```
CREATE TABLE annotate (  
  
uid int NOT NULL default '0',  
  
nid int NOT NULL default '0',  
  
note longtext NOT NULL,  
  
timestamp int NOT NULL default '0',  
  
PRIMARY KEY (uid, nid),  
  
);
```

我们可以仅仅把这段 **sql** 语句放到我们模块的 **README.txt** 文件中，这样其他想要安装这个模块的用户将不得不手工的将数据库表添加到他们的数据库中。另一种方式，当你启用你的模块时，**Drupal** 为你提供了自动创建相应的数据库表的工具，我们将充分利用这种方式的优势。我们创建一个特定的文件；文件名开始部分为你的模块名以后缀 **.install** 结束，所以对于 **annotate** 模块，这个文件名应为 **annotate.install**：

```
<?php  
  
// $Id$  
  
function annotate_install() {  
  
drupal_set_message(t('Beginning installation of annotate module.'));
```

```
switch ($GLOBALS['db_type']) {  
  
case 'mysql':  
  
case 'mysqli':  
  
db_query("CREATE TABLE annotations (  
  
uid int NOT NULL default 0,  
  
nid int NOT NULL default 0,  
  
note longtext NOT NULL,  
  
timestamp int NOT NULL default 0,  
  
PRIMARY KEY (uid, nid)  
  
) /*!40100 DEFAULT CHARACTER SET utf8 */;"  
  
);  
  
$success = TRUE;  
  
break;  
  
case 'pgsql':  
  
db_query("CREATE TABLE annotations (  
  
uid int NOT NULL DEFAULT 0,  
  
nid int NOT NULL DEFAULT 0,  
  
note text NOT NULL,  
  
timestamp int NOT NULL DEFAULT 0,  
  
PRIMARY KEY (uid, nid)  
  
);"  
  
);  
  
$success = TRUE;
```

```
break;

default:

drupal_set_message(t('Unsupported database.'));

}

if ($success) {

drupal_set_message(t('The module installed tables successfully.'));

}

else {

drupal_set_message(t('The installation of the annotate module

was unsuccessful.'),'error');

}

}
```

这个文件简单直接。当第一次启用 **annotate** 模块时，**drupal** 会寻找文件 **annotate.install** 并运行函数 **annotate_install()**，如果一切顺利的话，数据库表将被创建。通过禁用然后再启用这个模块，现在就可以完成它。

秘密消息：如果在你的 **.install** 文件中包含一个文字错误，或者由于其他原因执行失败，你可以使 **drupal** 忘记你的模块和它对应的表，**Administer > Site building > Modules** 禁用你的模块，然后在数据库的 **system** 表中删除对应模块的一行。

当创建了用以存储数据的表以后，我们将不得不对我们的代码做一些修改。其一，我们将需要添加一些代码，一旦用户填入注释并且点击 **Update** 按钮以后，用来解决对数据的处理流程。我们用于表单提交的函数如下：

```
/*

* Save the annotation to the database.

*/

function annotate_entry_form_submit($form_id, $form_values) {
```

```
global $user;

$nid = $form_values['nid'];

$note = $form_values['note'];

db_query("DELETE FROM {annotations} WHERE uid = %d and nid = %d",
$user->uid,
$nid);

db_query("INSERT INTO {annotations} (uid, nid, note, timestamp) VALUES
(%d, %d,
'%s', %d)", $user->uid, $nid, $note, time());

drupal_set_message(t('Your annotation was saved.'));

}
```

由于我们仅允许一个用户对一个节点只能有一个注释，所以我们可以安全的删除以前的注释（如果存在的话）然后插入我们自己的到数据库中。对于我们于数据库的交互，有几点需要注意。首先，我们不需要考虑数据库连接，这是因为在 **Drupal** 的引导程序中他已经为我们完成了这一工作。第 2，当我们是用一个数据库表时，我们需要把它放到花括号里{ }。这样可使数据库的前缀化无缝的集成（参看 <http://drupal.org/node/2622>）。第三，我们在查询语句中使用了占位符，然后为其提供相应的变量，这样 **Drupal** 内建的查询清洁机制可以启用以阻止 **SQL** 注入攻击。我们使用 %d 占位符用于数字，使用 %s 占位符用于字符串。然后，我们使用 `drupal_set_message()` 来在用户的会话中插入一消息，它将会在用户查看的下一个页面中被 **Drupal** 作为一个通知展示出来。这样，用户可以获得一些反馈信息。

最后，我们需要修改我们的 `nodeapi` 钩子代码，这样的话，如果已经存在了一个注释，它将被从数据库中取出。恰好在我们把我们的表单分配给 `$node->content` 以前，我们添加以下代码：

```
// Get previously saved note, if any.

$result = db_query("SELECT note FROM {annotations} WHERE uid = %d AND
nid = %d",
$user->uid, $node->nid);
```

```
$node->annotation = db_result($result);
```

我们首先查询数据库以取出这一用户对一节点所做的注释。接着，我们使用 `db_result` 来从结果集中取出第一行。由于我们仅允许一个用户对同一节点只能做一个注释，所以结果集中也只是一行。

测试你的模块。他现在应该能够保存和回显注释了。拍下你的背---你已经从头开始创建了一个模块了。现在你已经站在了成为 **Drupal** 核心开发者的大道上了。

更远的一些步骤

我们将与开源社区分享这一模块，这是自然的，所以需要创建一个 `README.txt` 文件，然后把它放到 `annotation` 的目录下,和 `notate.info`, `annotate.module`,`annotate.install` 文件放在一起。接下来，你可以把它上传到 `drupal.org` 上的贡献模块库中，然后创建一个项目页面来追踪社区中其他用户的反馈。

总结

当读完这一章后，你应该可以：

从头创建一个 **Drupal** 模块

了解如何钩住 **Drupal** 代码的执行

存储和回显特定模块的设定

使用 **Drupal** 的表单 **API** 来创建和处理一些简单的表单

使用数据库来存储和回显数据。

第 3 章 模块特定设置 (1)

模块特定设置

当你自己创建一个模块时,你常常想让站点管理员能够通过选择不同的模块设置属性来改变模块的行为。本章将详细讲述如何将一个模块呈现在 **Drupal** 的管理页面,如何为用户呈现一个关于设置的表单,以及存储设置属性。

把你的模块放到管理页面的模块列表中

Drupal 的管理页面为站点管理员展示了不同的站点配置选项。你想在这个配置页面找个未知把你的模块放下,这样站点管理员就可以调整你的模块的设置。现在让我们为前章所创建的注释模块添加更多的配置选项。

创建一个链接

我们需要在管理页面提供一个链接,这样站点管理员可以进入修改我们设置的页面。通过在菜单钩子(关于更多的菜单钩子的信息,参看第 4 章)上创建一个入口,我们将链接放在了站点配置设置下面。下面的就是在我们的模块中所实现的菜单钩子:

```
/**
 * Implementation of hook_menu().
 */
function annotate_menu($may_cache) {
  $items = array();
  if ($may_cache) {
    $items[] = array(
      'path' => 'admin/settings/annotate',
      'title' => t('Annotation settings'),
      'description' => t('Change how annotations behave.'),
```

```

'callback' => 'drupal_get_form',
'callback arguments' => array('annotate_admin_settings'),
'access' => user_access('administer site configuration')
);
}

return $items;
}

```

指向我们模块的链接现在出现在 **Drupal** 的管理页面的站点配置部分，如图 3-1 所示：

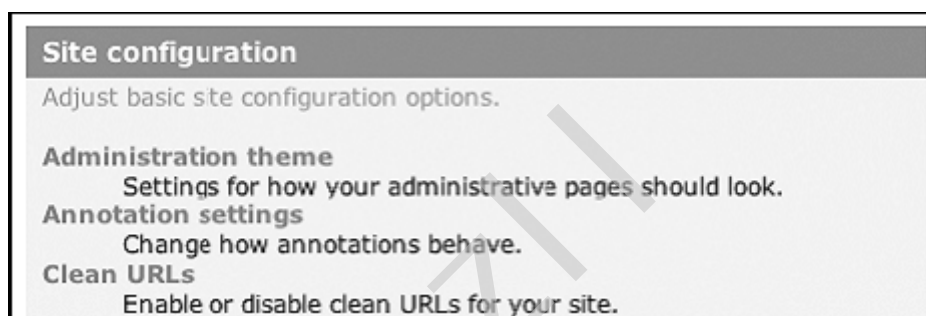


图 3-1 指向注释设置页面的链接

定义你自己的管理区域

Drupal 有多个管理设置的分类，比如内容管理和用户管理，都出现在主管理页面上。如果你的模块需要一个自己的分类，你可以非常容易得创建一个这样的分类。在这个例子中，我们创建一个名为“**Node annotation**”的新的分类。为了这样，我们修改我们的菜单钩子以定义新的分类：

```

/**
 * Implementation of hook_menu.
 */
function annotate_menu($may_cache) {
  $items = array();

```

```
if ($may_cache) {

  $items[] = array(

    'path' => 'admin/annotate',

    'title' => t('Node annotation'),

    'description' => t('Adjust node annotation options.'),

    'position' => 'right',

    'weight' => -5,

    'callback' => 'system_admin_menu_block_page',

    'access' => user_access('administer site configuration')

  );

  $items[] = array(

    'path' => 'admin/annotate/settings',

    'title' => t('Annotation settings'),

    'description' => t('Change how annotations behave.'),

    'callback' => 'drupal_get_form',

    'callback arguments' => array('annotate_admin_settings'),

    'access' => user_access('administer site configuration')

  );

}

return $items;

}
```

现在我们代码的结果改变了，我们的模块设置的链接出现在了新的分类中了，如图 3-2 所示：

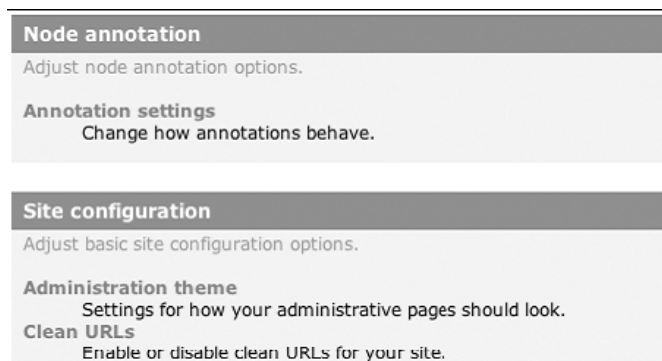


图 3-2 指向注释设置模块的链接现在作为一个单独的分类出现

如果你是一步一步跟着做的，那么你需要清除菜单缓存来查看链接的呈现。

有两种方式：截去 `cache_menu` 表或者使用 Drupal 开发专用模块所提空的清空缓存链接。

秘密消息：开发专用模块 (<http://drupal.org/project/devel>) 是为了专门支持 Drupal 开发所写的一个模块。它能帮你获取多种开发功能，比如清空缓存，查看变量，追踪查询语句，以及更多。它是专业开发的必需品。

我们可以在两步内建立我们的新分类。首先，我们添加一个描述分类头部的菜单项。这个菜单项有一个唯一的路径(`admin/annotate`)。我们生命：它应该放在右栏中，重量为-5，这是因为这个位置恰好位于站点配置分类的上面，这样方便我们截图，如图 3-2 所示的。

^_^

第 2 步是告诉 Drupal，把指向注释设置的真实链接放在分类“Node annotation”内部。我们通过修改原来菜单项的路径来这样做，以前路径为 `admin/settings/annotate`，现在替换为了 `admin/annotate/settings`。以前，菜单项是站点配置分类的路径 `admin/settings` 的孩子，，如表 3-1 所示。

当我们清空菜单缓存，Drupal 重新构造菜单树，由于 `admin/annotate/settings` 是 `admin/annotate` 的孩子，这决定了像图 3-2 所展示的这样。将模块菜单项嵌套在如表 3-1 所示的任一路径下，将使模块出现在 Drupal 管理页面的该分类下面。

当然，这是个有点做作的例子，在真实场景下，你必须有个很好的理由来创建一个新的分类，以阻止让管理员（通常是你自己）面对太多分类时所产生的困惑。

表 3-1 管理页面分类的路径

Path	Category
------	----------

admin/content	Content management
admin/build	Site building
admin/settings	Site configuration
admin/user	User management
admin/logs	Logs

为用户呈现一个设置表单

当一个站点管理员想要修改注释模块的设置时,我们需要展示一个表单以使管理员可以从中进行选择。在我们的菜单项中,我们把回调函数设置为 `drupal_get_form()`,把回调参数设置为 `annotate_admin_settings`。这意味着,当你访问

<http://example.com/?q=admin/>

`annotate/settings` 时,命令 `drupal_get_form('annotate_admin_settings')` 将被执行,它主要告诉 Drupal 构造由函数 `annotate_admin_settings()` 所定义的表单。

下面让我们看一下定义表单的函数,它定义了一个关于节点类型的复选框(参看图 2-1),并且增加了另外两个选项:

```
/**
 * Define the settings form.
 */
function annotate_admin_settings() {
  $form['annotate_nodetypes'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these node types'),
    '#options' => node_get_types('names'),
    '#default_value' => variable_get('annotate_nodetypes', array('story')),
    '#description' => t('A text field will be available on these node 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

);

```

Table 3-1. Paths to Administrative Categories

Path Category

admin/content Content management

admin/build Site building

admin/settings Site configuration

admin/user User management

admin/logs Logs

```

$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

);

return system_settings_form($form);

}

```

我们添加了一个单选按钮来选择什么时候应该删除注释，添加了一文本输入框来限制一个节点所允许的注释数量（对此模块增强特性的实现留给读者作为练习）。在这里，我们没有自己管理处理我们自己的表单的流程，而是使用了函数 `system_settings_form()` 来让系统模块为表单添加一些按钮，并让它管理表单的验证和提交。图 3-3 显示的当前表单的选项的样子。

Annotation settings

Users may annotate these node types:

page

story

A text field will be available on these node types to make user-specific notes.

Annotations will be deleted:

Never

Randomly

After 30 days

Select a method for deleting annotations.

Annotations per node:

Enter the maximum number of annotations allowed per node (0 for no limit).

图 3-3 使用了复选框，单选按钮，文本输入框的增强的表单



Hezi

第 3 章 模块特定设置 (2)

验证用户提交的设置

如果由函数 `system_settings_form()` 为我们管理表单的话，那么我们如何才能检查是否在每一节点多少个注释的输入框内输入的是一个真实的数字？我们可以钩住表单提交的处理过程么？以及怎么钩住？当然我们可以。我们仅需在我们的表单定义中定义一个验证函数，然后创建这个验证函数。

```
/**
 * Define the settings form.
 */

function annotate_admin_settings() {
  $form['annotate_nodetypes'] = array(
    28 CHAPTER 3 ■ MODULE-SPECIFIC SETTINGS
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these node types'),
    '#options' => node_get_types('names'),
    '#default_value' => variable_get('annotate_nodetypes', array('story')),
    '#description' => t('A text field will be available on these node 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
);

// Define a validation function.

$form['#validate'] = array(
    'annotate_admin_settings_validate' => array()
);

return system_settings_form($form);
}

// Validate the settings form.
```

```
function annotate_admin_settings_validate($form_id, $form_values) {  
  
if (!is_numeric($form_values['annotation_limit_per_node'])) {  
  
form_set_error('annotate_limit_per_node', t('Please enter a number.));  
  
}  
  
}
```

现在当 Drupal 处理这个表单时，它将回调 `annotate_admin_settings_validate()` 来进行验证。如果我们检测到输入的数据是坏的，我们将在错误出现的字段上设置一个错误信息，这反映为在页面上方出现一个警告信息，并将该字段的值转化为红色，如图 3-4 所示：

Annotation settings

Please enter a number.

Users may annotate these node types:

page

story

A text field will be available on these node types to make user-specific notes.

Annotations will be deleted:

Never

Randomly

After 30 days

Select a method for deleting annotations.

Annotations per node:

Enter the maximum number of annotations allowed per node (0 for no limit).

图 3-4 验证脚本设置了一个错误信息

存储设置

在前面的例子中，改变设置然后点击“**Save configuration**”按钮，可以正常工作。如果点击了“**Reset to defaults**”按钮，各字段值将重置为它们的默认值。下面部分将描述这是如何工作的。

使用 Drupal 的变量数据库表

首先，让我们看一下字段“Annotations per node”。它的#default_value 这样设置：

```
variable_get('annotate_limit_per_node', 1)
```

Drupal 在数据库中有一个名为 `variables` 的表，并且键值对可以使用方法 `variable_set($key,$value)` 来存储，使用方法 `variable_get($key,$default)` 来回显。所以我们要说的就是，“将字段‘Annotations per node’的默认值设置为数据库表 `variables` 所存储的变量 `annotate_limit_per_node` 的值，如果不存在的话，使用 `1` 作为默认值”。所以当点击“Reset to defaults”按钮时，Drupal 将使用默认值 `1`。

警告：为了使在 `variables` 表中存储和回显的设置没有命名空间的冲突，你应该是你的表单字段名和变量的键（如上例中的 `annotate_limit_per_node`）名称相同。命名方式为你的模块名加上一个描述性的名称。

由于“Annotations will be deleted”字段是一个单选按钮，它看起来复杂一点。这个字段的#option 如下所示：

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

当 PHP 遇到一个没有键的数组时，它默认的为其插入数字键，所以这个数组的真实表示如下所示：

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

当我们为这个字段设置默认值时，我们使用：

```
'#default_value' => variable_get('annotate_deletion', 0) //默认为 Never
```

它意味着，当有效时，默认为数组的第键值为 0 的元素，即是 t("Never")。

使用 `variable_get()` 来回显存储的值

当你的模块回显存储的设置时，应该采用

```
// Get stored setting of maximum number of annotations per node.
```

```
$max = variable_get('annotate_limit_per_node', 1);
```

注意，在这里 `variable_get()` 的默认值，也是在没有存储值可用的情况使用（可能管理员还没有访问设置页面）。

总结

当读完本章后，你应该可以：

创建一个链接，在 **Drupal** 的主配置页面，来指向你的模块的特定的配置设置页面。

在 **Drupal** 的主管理页面创建一个新的管理分类

定义一个表单，使得管理员可以为你的模块选择特定的选项

验证选项并且如果验证失败的话返回一个错误信息反馈。

理解 **Drupal** 如何使用自己内建的持久化变量系统来存储和回显模块的设置

第 4 章 Drupal 菜单 (menu) 系统 (1)

Drupal 的菜单 (menu) 系统是一个黑暗的地方之一，很少有人有勇气去探索它。披上你的盔甲---我们马上就进入了。

单词“菜单系统”有些用词不当。如果将菜单系统作为一个拥有三个主要责任的话，可能会更恰当一些：1，回调映射，2，访问控制，3，菜单定制。关于菜单系统的主要代码位于 `includes/menu.inc` 里，而比如启用定制菜单这些特性的可选代码位于 `menu.module`。

在本章，我们将探索一下什么是回调映射以及它是如何工作的，看一下如何通过访问控制来保护菜单项，并逐条列出了所有的各种内建的菜单项。最后通过检验如何覆写，添加，和删除存在的菜单项来结束本章，这样你就可以随心所欲的定制 Drupal 了。

回调映射 (Callback Mapping)

当一个 Web 浏览器向 Drupal 发送一个请求时，它向 Drupal 传递一个 URL。通过这一信息，Drupal 必须指出要运行哪段代码以及如何处理这一请求。这通常称为分发。Drupal 截取 URL 的基础部分并使用后面的部分，称之为路径。举例来说，如果 URL 是 <http://example.com/?q=node/3>，则 Drupal 路径为 `node/3`。

将 URLs 映射为函数

通常采用的方式如下所示：Drupal 向所有提供了一个菜单项数组的模块 (module) 查询，这些菜单项就是路径和关于路径的一些信息。一个模块 (module) 必须提供的一段信息就是回调 (callback)。在这里回调就是一个 PHP 函数的名称，当一个浏览器请求一个特定的路径时它就会运行。当一个请求到达的时候 Drupal 将执行以下步骤：

- 1，如果路径是一个真实路径的别名，Drupal 找出真实路径并使用它来代替别名。比如，如果管理员使用别名 <http://example.com/?q=cats> 来代替 <http://example.com/?q=node/3>，Drupal 使用 `node/3` 作为路径
- 2，执行钩子方法 `hook_menu`，这样所有的模块 (module) 都可以提供他们的回调。
- 3，创建一个从路径 (比如 `node/add`) 到回调 (php 函数 比如 `node_page()`) 的一个映射

4, 如果启用了 **menu.module** 模块, 将应用站点管理员对映射所做的任何修改和添加 (比如覆写一个菜单项的标题)。

5, 使用映射来查找请求 **URL** 对应的回调函数, 并调用它。如果同时声明了回调参数的话, **Drupal** 将其一同传送。

6, 返回函数的执行结果, 如果用户没有对这个 **URL** 的访问权限的话返回一个“拒绝访问”信息, 如果路径没有映射到一个函数的话返回一个 **404** 相应。

图 4-1 和 4-2 图形表示了这一流程。

Hezi

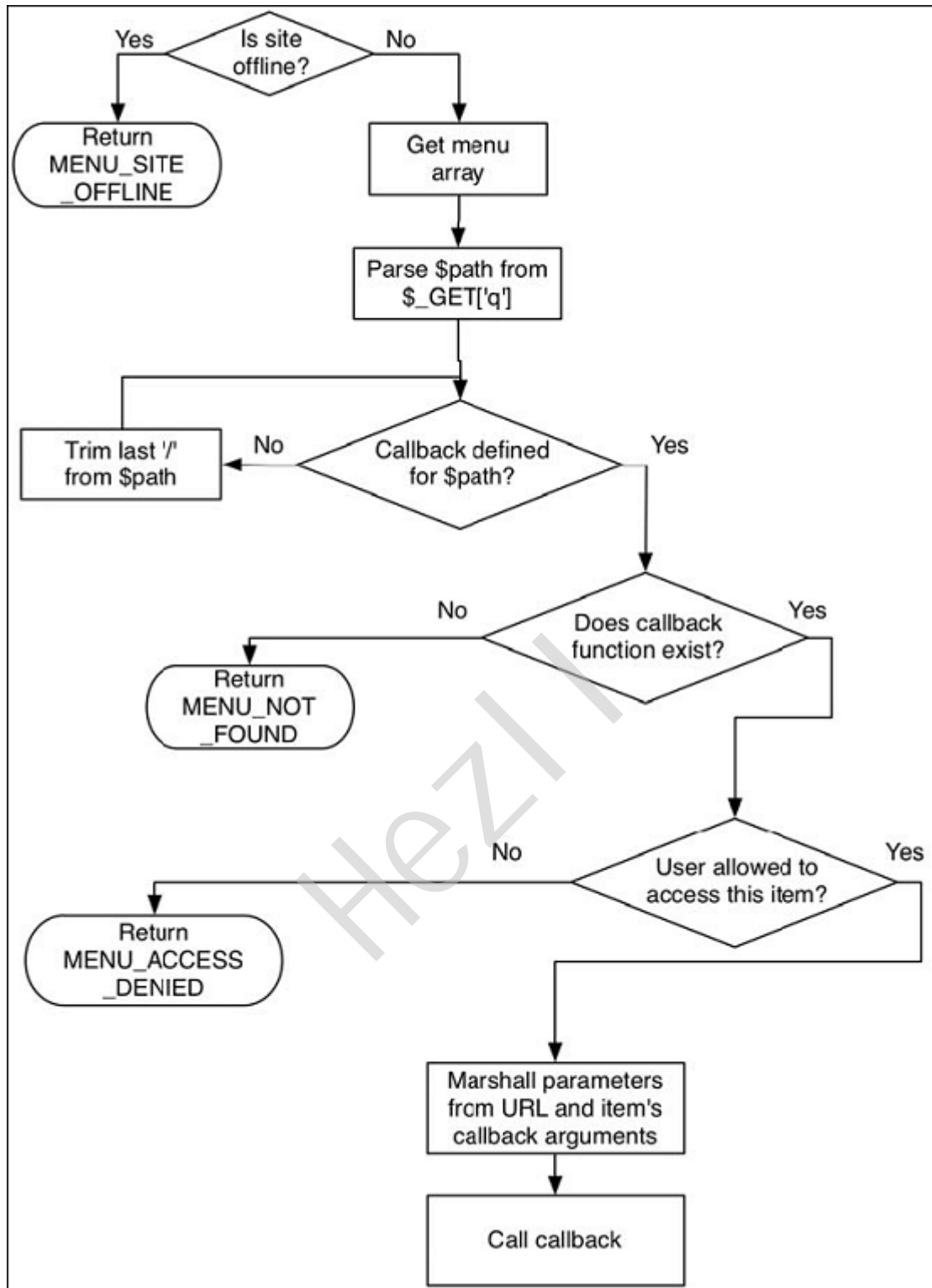


图 4-1 菜单分发流程概貌

注意图 4-1 中的获取菜单数组（Get menu array）这一部分。在这一流程中 Drupal 构造了数组 `$menu`，它包含了每一菜单项的详细信息，比如菜单项的路径，允许那些人访问，子菜单项，等等。图 4-2 展示了 Drupal 如何构造菜单数组的概貌。

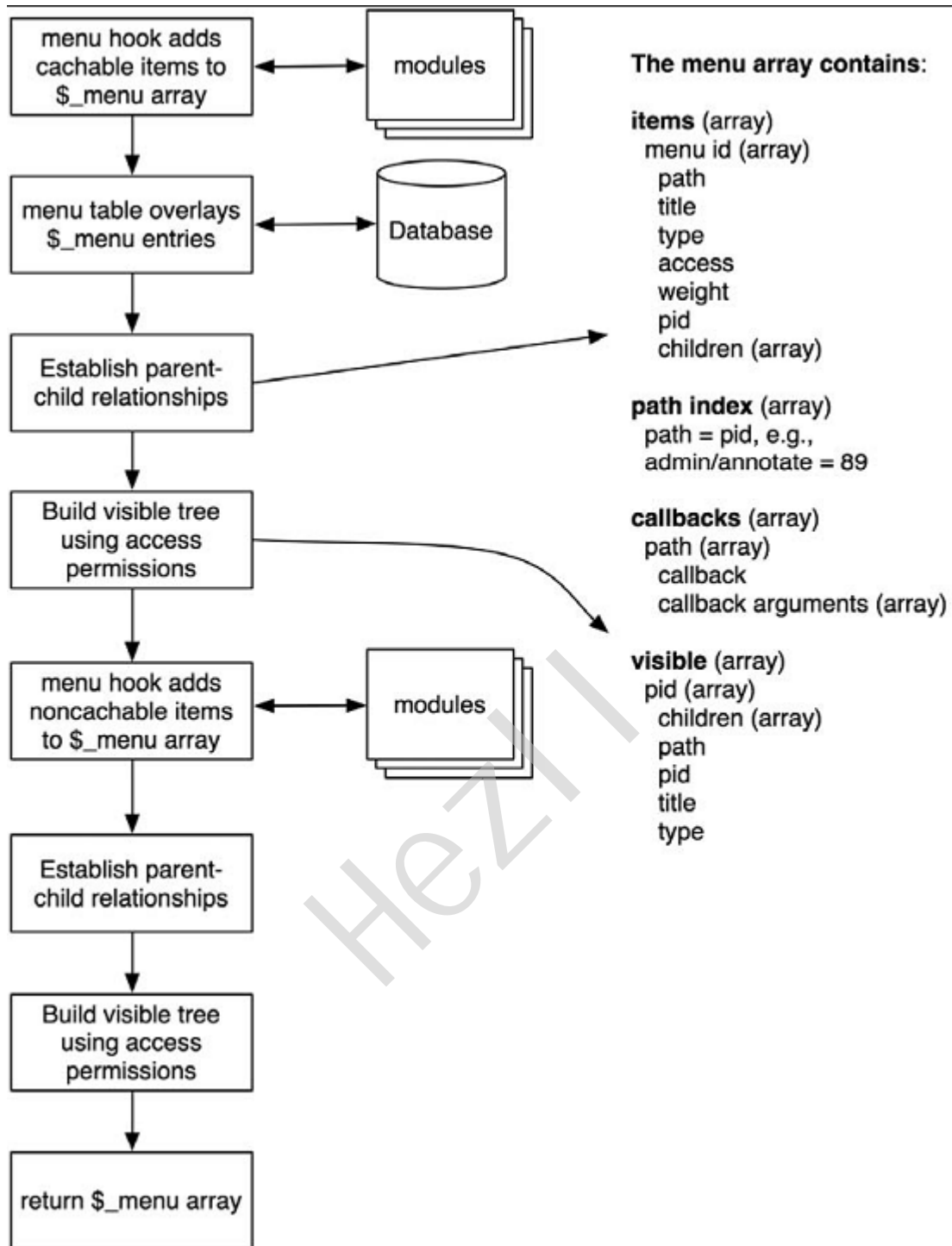


图 4-2 菜单数组构造流程概貌

通过在你的模块中使用菜单钩子来为这一流程设置钩子。这允许你定义可被包含到菜单树中的菜单项，让我们看一个名为 `mymenu.module` 的样例模块，它在 `Drupal` 的默认导航菜单中添加了一个菜单项。我们映射 `Drupal` 路径 `mymenu` 到 `php` 函数 `mymenu_hello()`。

首先，我们需要一个文件 `mymenu.info`：

```
; $Id $
```

```
name = "Mymenu Module"
```

```
description = "Adds a menu to the navigation block."
```

```
version = "$Name$"
```

然后我们需要文件 `mymenu.module`:

```
<?php
```

```
// $Id$
```

```
/**
```

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

```
*/
```

```
function mymenu_menu($may_cache) {
```

```
// Create an array to hold the menu items we'll define.
```

```
$items = array();
```

```
if ($may_cache) {
```

```
// Define a static menu item.
```

```
$items[] = array(
```

```
'title' => t('Greeting'),
```

```
'path' => 'mymenu',
```

```
'callback' => 'mymenu_hello',
```

```
'access' => TRUE
```

```
);
```

```
}
```

```
return $items;
```

```
}  
  
function mymenu_hello() {  
  
return t('Hello!');  
  
}
```

创建子目录 `sites/all/modules/mymenu`，然后将 `mymenu.info` 和 `mymenu.module` 文件放到里面，在管理员页面 **Administer** > **Site building** > **Modules** 启用这一模块，这样在导航区块(**block**)中就会出现这一菜单项，如图 4-3 所示。

需要注意的最重要的事是我们定义了一个路径并将其映射到一个函数上。路径是一个 **Drupal** 路径；也就是说，它是相对于你的 **Drupal** 的安装的基础 **URL** 而言的。但是在这里，还有一些其他的有趣事情发生。我们为我们的菜单项给出了一个标题，当页面在浏览器中显示时它会作为页面标题被自动使用（如果你想在以后的代码执行中覆写页面标题，你可以通过使用 `drupal_set_title()` 来设置它）。

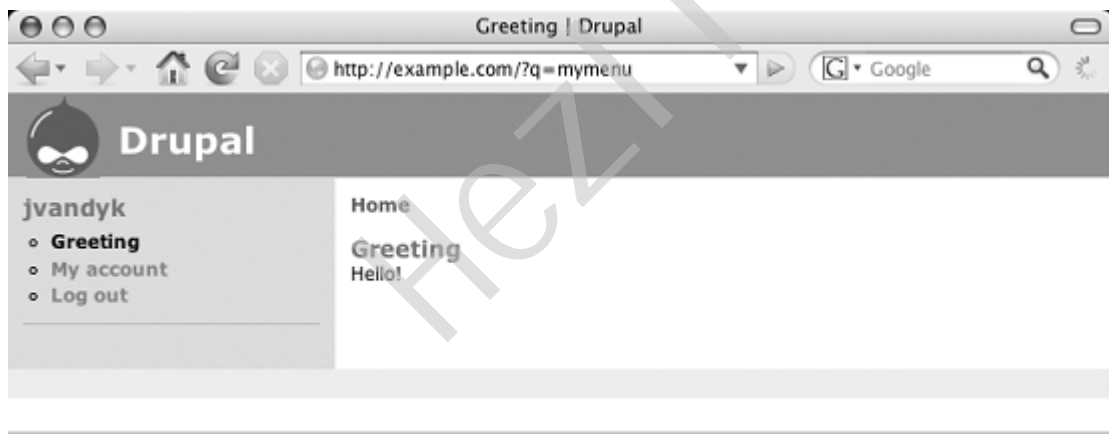


图 4-3 菜单项（Greeting）出现在导航区块(**block**)中。

菜单钩子实际被调用了两次：一次将 `$may_cache` 设置为 **TRUE**，一次将 `$may_cache` 设置为 **FALSE**。在本例中我们创建的菜单项是一个静态的菜单项。它不可改变，因此它是可缓存的。一旦为一给定用户构造了整个菜单树，**Drupal** 将该树缓存为一个序列化的数组放到表 `cache_menu` 中去。在接下来的请求中，该树将被回显和反序列化，而不是重新构造。

如果我们想创建一个动态的菜单项（比如，一个使用当前时间作为菜单项标题的），我们添加一个 `else` 语句：

```
<?php
```

```
// $Id$

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

function mymenu_menu($may_cache) {

// Create an array to hold the menu items we'll define.

$item = array();

if ($may_cache) {

// Define a static menu item.

$item[] = array(

'title' => t('Greeting'),

'path' => 'mymenu',

'callback' => 'mymenu_hello',

'access' => TRUE

);

}

else {

// Define a dynamic menu item.

$timestamp = format_date(time(), 'small');

$item[] = array(

'title' => t('Stock quote at @time', array('@time' => $timestamp)),

'path' => 'stockquote',

'callback' => 'mymenu_stock_quote',
```

```
'access' => TRUE

);

}

return $items;

}

function mymenu_hello() {

return t('Hello!');

}
```

现在菜单项在请求时被创建，如图 4-4 所示。

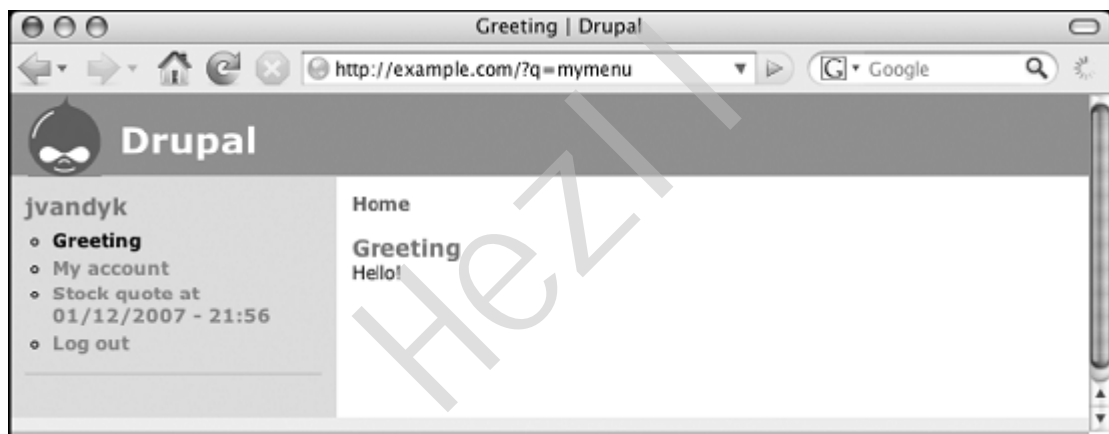


图 4-4 动态创建菜单项

动态菜单项应该尽可能的禁止使用，因为在每次 HTTP 请求中它们必须被检查和添加上去，而不是从一个缓存中回显出来。如果你有一个经常被网络爬虫访问的繁忙的站点或者一个大站点，这将添加大量的处理。

秘密消息：当你开发你的模块时，你将需要安装 **devel.module** 模块，因为它使你快速和方便的清楚菜单缓存，这样，你就可以立即看到代码修改后的结果。如果没有安装 **devel.module** 模块，你可以通过写出用于截去表 **cache_menu** 的 **sql** 来手工清理缓存（比如 **TRUNCATE TABLE 'cache_menu'**）。另一种方式是开发一个动态菜单项 (**!\$may_cache**)，当开发完成时将其更改为一个静态菜单项 (**\$may_cache**)。

第 4 章 Drupal 菜单 (menu) 系统 (2)

回调参数 (Callback Arguments)

有时你可能希望为映射到这一路径的函数提供更多的信息。首先，路径的任何额外部分都会被一同传送。让我们把函数改成如下所示：

```
function mymenu_hello($name = NULL) {  
  
  if (!isset($name)) {  
  
    $name = t('good looking!');  
  
  }  
  
  // Sanitize the user submitted name.  
  
  return t('Hello @name!', array('@name' => $name));  
  
}
```

现在如果我们访问页面 <http://example.com/?q=mymenu>，我们将得到：

Hello, good looking!

如果我们访问页面 <http://example.com/?q=mymenu/Fred>，我们得到：

Hello, Fred;

你也可以在菜单钩子内部定义回调参数，通过项数组 `$items` 添加可选的键 `callback arguments`。你在这里定义的回调参数将比从路径中生成的参数靠前。这非常有用，因为你可以从不同的菜单项来调用同一个回调函数并为它们提供一些隐藏的上下文信息。

```
function mymenu_menu($may_cache) {  
  
  $items = array();  
  
  if ($may_cache) {  
  
    $items[] = array(  

```

```
'title' => t('Greeting'),

'path' => 'mymenu',

'callback' => 'mymenu_hello',

'callback arguments' => array(t('Hi!'), t('Ho!')),

'access' => TRUE

);

}

return $items;

}

function mymenu_hello($first, $second, $name = NULL) {

// We just want to see what $first and $second are.

drupal_set_message(t('First is %first', array('%first' => $first)));

drupal_set_message(t('Second is %second', array('%second' => $second)));

if (!isset($name)) {

$name = t('good looking');

}

return t('Hello @name!', array('@name' => $name));

}
```

现在访问页面 <http://example.com/?q=mymenu/Fred>,我们得到如图 4-5 所示的结果。

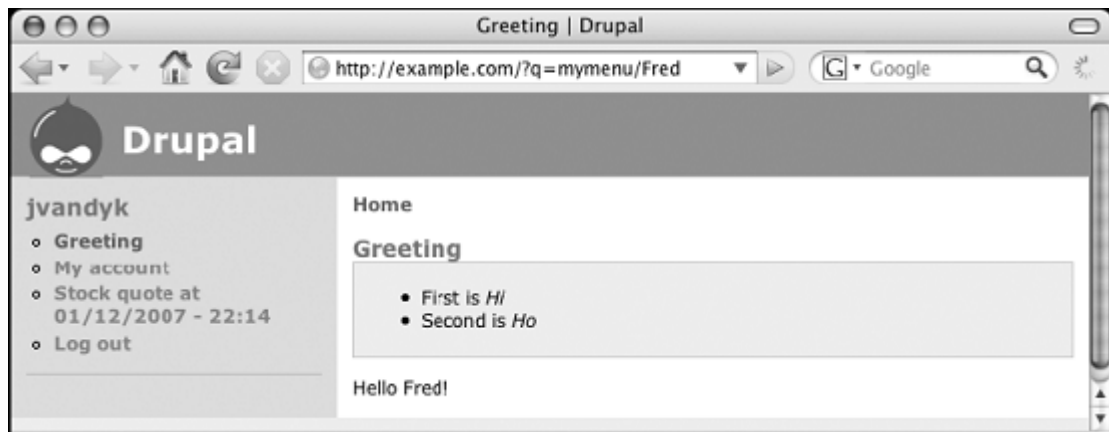


图 4-5 传递并展示回调参数

在回调参数的对应数组中，键将被忽略，所以你不能使用键来映射函数参数；只有顺序才是重要的。回调参数通常是变量，并且经常用在动态菜单项中（比如当 `$may_cache` 为 `FALSE` 时添加他们）。

菜单嵌套：

到目前为止我们仅定义了一个单独的静态菜单项。让我们添加第 2 个：

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

function mymenu_menu($may_cache) {

  $items = array();

  if ($may_cache) {

    $items[] = array(

      'title' => t('Greeting'),

      'path' => 'mymenu',

      'callback' => 'mymenu_hello',
```

```
'access' => TRUE

);

$items[] = array(

'title' => t('Farewell'),

'path' => 'mymenu/goodbye',

'callback' => 'mymenu_goodbye',

'access' => TRUE

);

}

return $items;

}
```

Drupal 将发现第 2 个菜单项(**mymenu/goodbye**)的路径是第一个菜单项路径 (**mymenu**) 的孩子。因此，当渲染菜单时（转化为 HTML 格式），Drupal 将对第 2 个菜单进行缩进，如图 4-6 所示。

当然，一个主题 (**theme**) 可以渲染设计者希望的任何菜单样式。

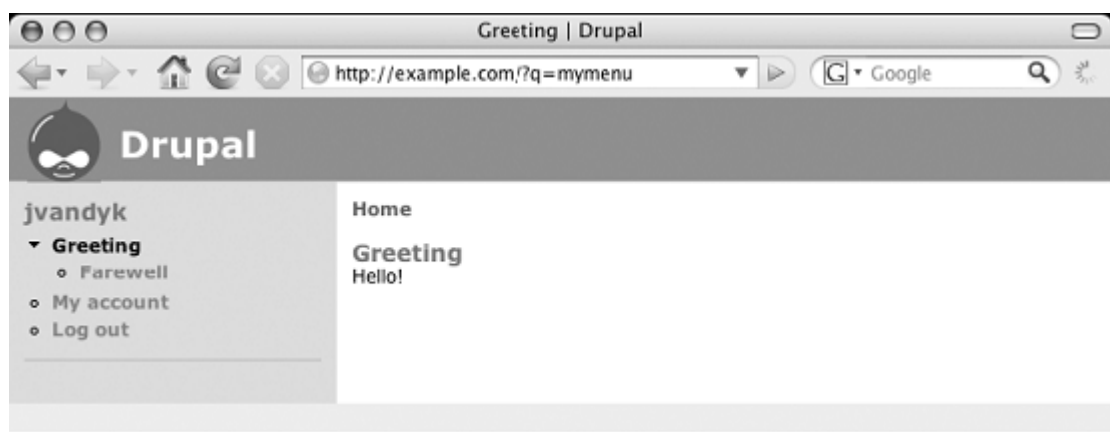


图 4-6 嵌套菜单 (Nested menu)

访问控制

到目前为止，在我们的例子中，我们简单的把菜单项的键 `access` 设置为 `TRUE`，这意味着任何用户都可以访问我们的菜单。通常菜单访问是通过在模块内部使用 `hook_perm()` 定义权限来控制的，通过使用 `user_access()` 来测试这些权限。让我们定义一个名为“`receive greeting`”的权限，如果一个用户没拥有一个获取这一权限的角色，如果他（或她）想访问页面 <http://example.com/?q=mymenu> 时，他将收到一个“拒绝访问”信息。

```
/**
 * Implementation of hook_perm().
 */
function mymenu_perm() {
  return array('receive greeting');
}
/**
 * Implementation of hook_menu().
 */
function mymenu_menu($may_cache) {
  $items = array();
  if ($may_cache) {
    $items[] = array(
      'title' => t('Greeting'),
      'path' => 'mymenu',
      'callback' => 'mymenu_hello',
      'access' => user_access('receive greeting') // Returns TRUE or FALSE.
    );
  }
}
```

```

}

return $items;

}

```

这样，菜单系统可被看作是个守门员，他根据用户的角色来决定那些路径可以访问，那些将被拒绝访问。

菜单项可以嵌套。举例来说，一个路径为 **foo/bar** 的菜单项是路径为 **foo** 的菜单项的孩子。当决定对一个菜单项的访问时，**Drupal** 将查找菜单项的全路径对应的键 **access** 并使用它。如果键 **access** 为 **True**，访问将被允许，即便它的父菜单的键 **access** 是 **FALSE**。如果没有为一个菜单项设置 **access**，将会默认使用它父菜单的键 **access**。如果父菜单也没有设置 **access**，**Drupal** 将依次向上迭代直到找到一个 **access** 键（菜单树的根的键 **access** 为 **TRUE**）。本地任务为简单的嵌套菜单项。表 4-1 展示了基于一个菜单项的访问设置和它父菜单的访问控制来决定一个用户的访问被允许还是被拒绝。

表 4-1 访问设置和用户访问结果

Parent	Child	User Access
FALSE	FALSE	Denied
TRUE	FALSE	Denied
FALSE	TRUE	Allowed
TRUE	TRUE	Allowed
FALSE	Undefined	Denied
TRUE	Undefined	Allowed

各种菜单项

当你在菜单钩子中添加一个菜单项时，一个你可能用到的键就是类型（**type**）。如果你们有定义类型，将会使用默认类型 **MENU_NORMAL_ITEM**。**Drupal** 将根据你分配的类型不同来区别对待相应的菜单。每一个菜单项类型都有一系列的标记或者属性组成。表 4-2 列出了菜单项类型的标记。

表 4-2 菜单项类型标记

Binary	Hexadecimal	Decimal	Constant
000000000001	0x0001	1	MENU_IS_ROOT
000000000010	0x0002	2	MENU_VISIBLE_IN_TREE
000000000100	0x0004	4	MENU_VISIBLE_IN_BREADCRUMB
000000001000	0x0008	8	MENU_VISIBLE_IF_HAS_CHILDREN
000000010000	0x0010	16	MENU_MODIFIABLE_BY_ADMIN
000000100000	0x0020	32	MENU_MODIFIED_BY_ADMIN
000001000000	0x0040	64	MENU_CREATED_BY_ADMIN
000010000000	0x0080	128	MENU_IS_LOCAL_TASK
000100000000	0x0100	256	MENU_EXPANDED
001000000000	0x0200	512	MENU_LINKS_TO_PARENT

举例来说，常量 `MENU_NORMAL_ITEM` 拥有标记 `MENU_VISIBLE_IN_TREE`, `MENU_VISIBLE_IN_BREADCRUMB`, 和 `MENU_MODIFIABLE_BY_ADMIN`, 如表 4-3 所示。看一下不同的标记在同一个单独的常量中是如何表示的。

表 4-3 菜单项类型 `MENU_NORMAL_ITEM` 的标记

Binary	Constant
000000000010	MENU_VISIBLE_IN_TREE
000000000100	MENU_VISIBLE_IN_BREADCRUMB

000000010000

MENU_MODIFIABLE_BY_ADMIN

000000010110

MENU_NORMAL_ITEM

因此 MENU_NORMAL_ITEM 拥有下列标记 000000010110, MENU_NORMAL_ITEM

表 4-4 展示了可用的菜单项类型和他们所表达的标记

Constant	MENU_IS_ROOT	MENU_VISIBLE_IN_TREE	MENU_VISIBLE_IN_BREADCRUMBS	MENU_VISIBLE_IF_HAS_CHILDREN	MENU_MODIFIABLE_BY_ADMIN	MENU_CREATED_BY_ADMIN	MENU_IS_LOCAL_TASK	MENU_EXPANDED	MENU_LINKS_TO_PARENT
MENU_NORMAL_ITEM	x	x	x						
MENU_ITEM_GROUPING		x	x	x					
MENU_CALLBACK		x							
MENU_DYNAMIC_ITEM	x	x							
MENU_SUGGESTED_ITEM		x		x					
MENU_LOCAL_TASK							x		
MENU_DEFAULT_LOCAL_TASK							x		x
MENU_CUSTOM_ITEM	x	x		x	x				
MENU_CUSTOM_MENU	x	x		x	x				

表 4-4 菜单项类型所表达的标记

当你定义你的菜单项类型时,你应该使用那一个常量呢? 查看表 4-4, 看一下你想使用那些标记, 然后使用包含这些标记的常量。对于每一个常量的详细描述, 参看 includes/menu 里面的注释。最常用的为 MENU_CALLBACK, MENU_LOCAL_TASK, 和 MENU_DEFAULT_LOCAL_TASK。仔细阅读可获更详细信息。

第 4 章 Drupal 菜单 (menu) 系统 (3)

常用任务

在这一部分，我们展示了程序员使用菜单面临的常见问题的一些典型解决办法。

分配回调而不向菜单添加一个链接

你可能经常会遇到将一个 URL 映射到一个函数，同时不需要创建一个可见的菜单项的情况。你可以通过为你的菜单项指定类型为 `MENU_CALLBACK` 来完成这一任务，下面是从 `node.module` 中取出来的例子：

```
$items[] = array(
  'path' => 'rss.xml',
  'title' => t('RSS feed'),
  'callback' => 'node_feed',
  'access' => user_access('access content'),
  'type' => MENU_CALLBACK
);
```

将菜单项展示为标签

用 Drupal 的公认地晦涩的行话来说，一个展示为标签的回调被看作是一个本地任务 (`local task`) 并拥有类型 `MENU_LOCAL_TASK` 或者 `MENU_DEFAULT_LOCAL_TASK`。本地任务的标题应该是一个简短的动词，比如“添加”或者“列出”。本地任务通常使用在特定的一些对象上，比如节点 (`node`)，用户 (`user`)，或者工作流 (`workflow`)。

为了使标签可以被展示，本地任务必须有一个父菜单项。一个常用的实践是将一个回调指定到一个根路径比如 `milkshake` 上，然后将本地任务指定到扩展了这一路径的子路径，比如 `milkshake/prepare`，`milkshake/drink`，等等。Drupal 内建支持了两级可标签化的本地任务。

标签的展示顺序由菜单项标题的值的字母顺序来决定。如果你不喜欢这种顺序，你可以为你的菜单项添加一个键 **weight(重量)**，然后它们将按重量排序。下例展示了使用默认本地任务的代码生成了两个主标签和两个次标签：

```
/**
 * Implementation of hook_menu().
 */
function milkshake_menu($may_cache) {
  $items = array();
  if ($may_cache) {
    $items[] = array(
      'path' => 'milkshake',
      'title' => t('Milkshake flavors'),
      'callback' => 'milkshake_overview',
      'type' => MENU_CALLBACK
    );
    $items[] = array(
      'path' => 'milkshake/list',
      'title' => t('List flavors'),
      'type' => MENU_DEFAULT_LOCAL_TASK,
      'access' => user_access('list flavors'),
      'weight' => 0
    );
    $items[] = array(
      'path' => 'milkshake/add',
```

```
'title' => t('Add flavor'),

'callback' => 'milkshake_add',

'type' => MENU_LOCAL_TASK,

'access' => user_access('add flavor'),

'weight' => 1

);

$items[] = array(

'path' => 'milkshake/list/fruity',

'title' => t('Fruity flavors'),

'callback' => 'milkshake_list',

'type' => MENU_LOCAL_TASK,

'access' => user_access('list flavors'),

);

$items[] = array(

'path' => 'milkshake/list/candy',

'title' => t('Candy flavors'),

'callback' => 'milkshake_list',

'type' => MENU_LOCAL_TASK,

'access' => user_access('list flavors'),

);

}

return $items;

}
```

```
function milkshake_overview() {

$output = t('The following flavors are available...');

// ... more code here

return $output;

}
```

图 4-7 展示了在 Drupal 的 Bluemaringe 主题 (theme) 下的结果。



图 4-7 本地任务和可标签化的菜单

注意页面的标题来自于父回调，而不是来自默认本地任务。如果你想使用一个不同的标题，你可以使用 `drupal_set_title()` 来设置它。

手工修改已存在的菜单

当你在你的模块中实现菜单钩子的时候，你完全可以为其他模块的路径添加入口，或者甚至可以覆盖它们。这通常可以使用 `menu.module` 提供的方便的 `web` 接口来完成，`menu.module` 是 Drupal 自带的一部分，但是你可能也有需要使用编程来完成的情况。

包裹指向菜单项的调用

例如，`devel.module` (如果你想做一些严谨的 Drupal 开发的话，你可能会用到) 拥有一个菜单项用来清楚 Drupal 的缓存表。现在让我们包裹这一函数从而首先调用我们的函数。首先，我们通过在我们的菜单钩子内部声明具有相同路径的菜单项来覆写 `devel.module` 的菜单项。

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

```
function mymodule_menu($may_cache) {

  $items = array();

  if (!$may_cache && module_exists('devel')) {

    // Make sure devel.module is enabled.

    $items[] = array(

      'path' => 'devel/cache/clear', // Same path that devel.module uses.

      'title' => t('Wrap cache clear'),

      'callback' => 'mymodule_clear_cache',

      'type' => MENU_CALLBACK,

      'access' => user_access('access devel information')

    // Same as devel.module.

    );

  }

}

function mymodule_clear_cache() {

  drupal_set_message('We got called first!');

  // Wrap the devel function normally called.

  devel_cache_clear();

}
```

现在当我们进入 <http://example.com/?q=devel/cache/clear> 时,我们的模块将先被调用, 然后它将调用原本应该调用的函数。下面是执行结果:

We got called first!

Cache cleared.

当你想修改 **Drupal** 的默认行为而不改变底层代码，这是一个非常有用的技术。

注意：由于我们模块的菜单钩子调用位于 **devel.module** 的之后，所以在本部分展示的技术才正常工作。模块被调用的顺序由它们位于表 **system** 中的重量决定。当覆写路径时，当 **\$may_cache** 为 **FALSE** 时通常容易添加你的菜单项，这是因为这些菜单项在最后时刻被添加，而此时大多数路径都已到位（因此可被覆写）。

删除已存在的菜单

使用在“包裹指向菜单项的调用”一节中所展示的方式，你可以通过覆写它们的路径来删除已存在的菜单项。比如由于一些原因，你想删除菜单项“**create content**”以及添加内容的能力：

```
$items[] = array(
  'path' => 'node/add',
  'title' => t('This should not show up'),
  'callback' => 'drupal_not_found',
  'type' => MENU_CALLBACK
);
```

注意：在这种情况下，你不能在保留创建特定内容类型的同时删除“**create content**”页面，这是因为他们是同一概念：这一菜单项所映射函数 **node_add**，它使用了 **Drupal** 内建的根据路径来传递参数，所以 **node/add** 和 **node/add/story** 调用了同一个函数。在后一种情况，“**stary**”作为一个参数被传递。一个保留创建内容类型的可选的方式是使用 **menu.module** 的接口来禁用菜单项“**create content**”，这将使低于“**create content**”的菜

单项可被访问。

向已有的菜单中添加菜单项

你可以通过使用一个聪明的路径来将你的菜单项添加到已存在的菜单中去。例如，假如你感觉非常暴躁，想为用户管理接口添加一个标签用来删除所有的用户。通过检查 `user.module` 中的菜单钩子，你决定把路径 `admin/user` 作为你的基础路径。下面是你从 `eradicateusers`。

Module 中返回的菜单项：

```
$items[] = array(  
  
  'path' => 'admin/user/eradicate',  
  
  'title' => t('Eradicate all users'),  
  
  'callback' => 'mymodule_eradicate_users',  
  
  'type' => MENU_LOCAL_TASK,  
  
  'access' => user_access('eradicate users')  
  
);
```

这将菜单项作为本地任务来添加，如图 4-8 所示：

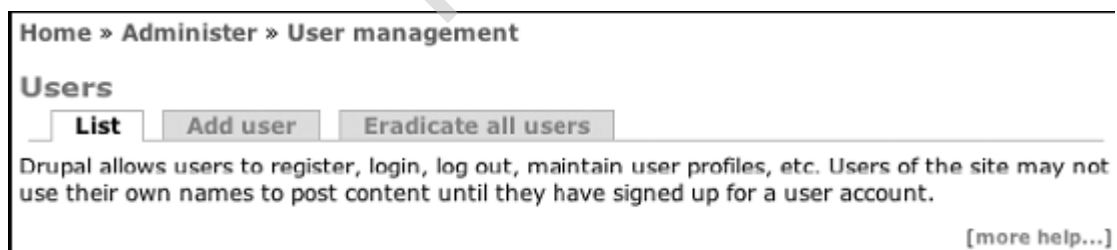


图 4-8 向另一模块的菜单添加本地任务

如果你想在管理菜单区块中展示这一菜单项，那么你需要使用类型 `MENU_NORMAL_ITEM` 来代替 `MENU_LOCAL_TASK`。如果你想在两个地方同时显示，你需要这样：

```
'type' => MENU_NORMAL_ITEM | MENU_LOCAL_TASK
```

这样菜单项就同时拥有了两个菜单项类型属性。

使用 `menu.module`

当位于 `includes/menu.inc` 中的函数 `menu_rebuild` 运行时，代表着菜单树的数据结构将被镜像到数据库中。它发生在你启用或者禁用模块的时候，或者其他的影响菜单树的组成的一些事件时。数据被保存到了数据库表 `menu` 中，该表的结构是这样的(来自于 `modules/system/system.install`):

```
CREATE TABLE {menu} (  
  
  mid int unsigned NOT NULL default '0',  
  
  pid int unsigned NOT NULL default '0',  
  
  path varchar(255) NOT NULL default '',  
  
  title varchar(255) NOT NULL default '',  
  
  description varchar(255) NOT NULL default '',  
  
  weight tinyint NOT NULL default '0',  
  
  type int unsigned NOT NULL default '0',  
  
  PRIMARY KEY (mid)  
  
) /*!40100 DEFAULT CHARACTER SET UTF8 */
```

注意访问信息没有保存到数据库中。在为每次请求构建菜单树的流程中，**Drupal** 首先根据从模块的菜单钩子中收到的信息来构造树，然后使用从数据库中取出来的信息来覆盖这些信息。这一行为使得你可以使用 `menu.module` 来改变菜单树的属性：父亲，路径，标题以及描述---你没有改变底层的菜单树，实际上，实际上创建了覆盖在它上面的数据。

注意 菜单项类型，比如 `MENU_CALLBACK` 或者 `DEFAULT_LOCAL_TASK`，在数据库中是以其数字代码来表示的。

`Menu.module` 也为节点表单添加了一节，用来将当前发布的内容作为一个菜单项使用。

常见错误

现在你刚刚在你的模块中实现了菜单钩子，但是你的回调没有被触发，你的菜单没有被显示，或者其他不能正常工作的事情。下面是一些通常需要检查的地方：

你为一个函数设置的键 `access` 的返回是否为 `FALSE`?

在你的菜单钩子的结尾你是不是忘记了添加 `return $items`?

你清空了你的菜单缓存了吗?

如果你使用表达式来指定键 `path`，那么这个表达式是不是解析为一个合法的路径?

如果你通过指定类型为 `MENU_LOCAL_TASK` 从而将菜单项展示为标签的话，你制定了一个带有回调函数的父菜单项了吗?

如果你使用了本地任务，那么在一个页面你至少拥有两个标签了吗(为了展现这是必须的)?

如果你想修改/覆盖/删除一个存在的路径时，你能确保你的模块中菜单钩子的调用位于定义你所覆写路径的菜单钩子之后么? 当 `$may_cache` 为 `FALSE` 时试着返回你的菜单项，以使你的菜单项定义位于流程的后面。

小结

当读完这一章后，你应该可以：

将 `URL` 映射到函数上

为菜单树添加入口

创建带有映射到函数上的标签（本地任务）的页面

理解访问控制是如何工作的

通过代码来添加，修改，删除已存在的菜单项

注意：更多的阅读，可参看 `menu.inc` 的注释，它值得一读。当然，也可参看 <http://api.drupal.org/api/5/group/menu>。

第 5 章 Drupal 数据库层 (1)

Drupal 的数据库层

Drupal 的正常工作依赖于数据库。在 Drupal 内部，在你的代码与数据库之间存在着一个轻量级的数据库抽象层。在本章，你将学习这一数据库抽象层是如何工作的，如何使用它，甚至如何写一个自己的驱动。你将看到查询语句如何被模块自动的修改以限制这些查询的范围。然后你将看到如何链接额外的数据库（比如一个合法的数据库）。最后，当一个模块独立的安装，升级，或者禁用时，你将测试如何创建，转移，甚至删除一个表。

定义数据库参数

在建立数据库连接时，通过查看你站点的 `settings.php` 文件，Drupal 将知道需要连接哪一个数据库和使用相应的用户名和密码。这个文件一般位于 `sites/example.com/settings.php` 或者 `sites/default/settings.php`。定义数据库连接的代码行如下所示：

```
$db_url = 'mysql://username:password@localhost/databasename';
```

这个例子使用的是 MySQL 数据库。使用 PostgreSQL 的用户需要将前缀“mysql”替换为“pgsql”。显然，这里使用的用户名和密码对于你的数据库来说必须是合法的。他们是数据库的机密，而不是 Drupal 的，当你使用你的数据库工具建立数据库帐号是建立他们(用户名和密码)。

理解数据库抽象层

使用一个数据库抽象层 API，直到有一天你试着不再使用它的时候，你才能发现他的全部优点。你是否曾经遇到过需要改变数据库系统的项目，你花费大量的时间仔细的审查每段代码来将它改为特定数据库的函数和查询？使用数据库抽象层以后，你将不再考虑各个不同数据库之间函数名称的细微差别，由于你使用的是符合 ANSI SQL 的语句，你不再需要写单独的查询语句了。举例来说，没有使用 `mysql_query()` 或者 `pg_query()`，Drupal 使用的是 `db_query()`，这使得业务层和数据库层相互隔离。

Drupal 的数据库层是轻量级的，它主要用于两个目的。第一个目的是使你的代码不与特定的数据库绑定。第二个目的是清洁用户提交的用于查询语句的数据以阻止 SQL 注入攻击。这一层是建立在使用 Sql 比重新学习一个新的抽象层语言更方便的原理之上的。通过检查

你的文件 `settings.php` 内部的变量 `$db_url`，Drupal 决定要连接的数据库的类型。例如，如果 `$db_url` 开始部分为“mysql”，那么 Drupal 将包含进 `includes/database.mysql.inc`。如果 `$db_url` 开始部分为“pgsql”，那么 Drupal 将包含进 `includes/database.pgsql.inc`。图 5-1 展示这一机制。

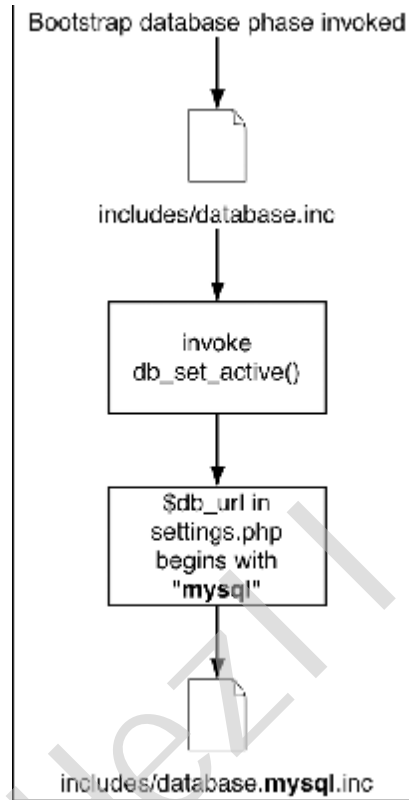


图 5-1 通过检查变量 `$db_url`，Drupal 决定需要引入哪一个数据库文件。

作为一个例子，让我们比较一下 `db_fetch_object()` 在 MySQL 和 PostgreSQL 抽象层的不同之处：

```
// From database.mysql.inc.  
  
function db_fetch_object($result) {  
  
if ($result) {  
  
return mysql_fetch_object($result);  
  
}  
  
}  
  
// From database.pgsql.inc.
```

```
function db_fetch_object($result) {  
  
if ($result) {  
  
return pg_fetch_object($result);  
  
}  
  
}
```

如果你使用的数据库还未被支持,你可以通过为你的数据库实现相应的包裹函数来建立你自己的数据库抽象层。更多信息,参看本章最后部分的“建立你自己的数据库抽象层”。

建立数据库连接

作为 **Drupal** 的通常的引导指令(**bootstrap**)流程的一部分,它将自动的建立数据库连接,所以你不需要为此担心。

-----注意:当你遇到需要写一个单独的 **PHP** 脚本或者有段处于 **Drupal** 之外的 **PHP** 代码的情况时,你需要访问 **Drupal** 的数据库,你可以使用 `include_once('includes/bootstrap.inc')`,然后调用 `drupal_bootstrap(DRUPAL_BOOTSTRAP_DATABASE)`来建立一个数据库连接。然后你就可以使用 `db_query()`了,正如下节所讲。

执行简单的查询

Drupal 的函数 `db_query()`用来为已建立的数据库连接执行查询语句。这些查询语句包括 **SELECT**, **INSERT**, **UPDATE**, 和 **DELETE**。让我们看一些例子。

从名为 **joke** 的表中取出所有字段的所有行,条件为字段 **vid** 的整数值与 `$node->nid` 的值相等:

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

向名为 **joke** 的表中插入一行。这一行中将包含两个整数和一个字符串值(注意字符串值的占位符位于单引号中,这将帮助阻止 **SQL** 注入攻击):

```
db_query("INSERT INTO {joke} (nid, vid, punchline) VALUES (%d, %d, '%s')",  
$node->nid, $node->vid, $node->punchline);
```

修改名为 `joke` 的表中的所有行，条件为字段 `vid` 的整数值与 `$node->nid` 的值相等。通过设置字段 `puchline` 等于 `$node->punchline` 包含的值来修改所有的这些行：

```
db_query("UPDATE {joke} SET punchline = '%s' WHERE vid = %d",
        $node->punchline,
        $node->vid);
```

从名为 `joke` 的表中删除所有行，条件为字段 `vid` 的整数值与 `$node->nid` 的值相等：

```
db_query('DELETE FROM {joke} WHERE nid = %d', $node->nid);
```

当你写 **SQL** 语句的时候你需要知道，这里有一些 **Drupal** 特定的语法。首先，注意到表名位于花括号之间。这样做是为了表名可以前缀化这样可以保证它们名称的唯一性。这一习惯可以使用户在已存在的数据库中安装 **Drupal** 而避免表名的冲突，因为用户的主机托管提供者限制了他们可以建立的数据库的个数。

下一个不同寻常的地方是 `%d` 占位符。在 **Drupal** 中，查询语句通常使用占位符，真实值作为参数跟在后面。占位符 `%d` 将自动的被后面参数的值替换掉—在此为 `$node->vid`。更多的占位符意味着更多的参数：

```
db_query('SELECT FROM {joke} WHERE nid > %d AND nid != %d', 5, 7);
```

在数据库中执行时，他将转化为如下形式：

```
SELECT FROM joke WHERE nid > 5 and nid != 7
```

用户提交的数据应该作为单独的参数传入，这样这些值可以被清洗从而阻止 **SQL** 注入攻击。**Drupal** 使用 `printf` 语义(参看 <http://php.net/printf>)来作为占位符在查询语句中的值。根据用户提交数据的类型，这里有不同的`%`修饰符。

表 5-1 列出了数据库查询占位符和他们的含义。

表 5-1 数据库查询占位符和他们的含义

Placeholder	Meaning
-------------	---------

<code>%s</code>	String
-----------------	--------

<code>%d</code>	Integer
-----------------	---------

%f **Float**

%b **Binary data; do not enclose in ''**

%% Inserts a literal % sign (e.g., **SELECT * FROM {users} WHERE name LIKE '%%s%%'**)

db_query()的第一个参数总为查询语句本身。剩下的参数都是用于验证和插入到查询字符串中去的动态值。它可以是一个值的数组，或者每个值都作为一个独立的参数。后者更常用一些。

我们应该注意到使用这一语法我们将 **NULL, TRUE, 和 FALSE** 自动类型转换到了相应的数字等价形式(**0** 或 **1**)。大多数情况它是对的。

回显查询结果

有多种方式用于回显查询结果，这依赖于你的需求，你是需要一个单独的一行还是需要整个结果集，或者为了内部使用你打算得到结果的一部份，或者你想分页显示。

取回单个的值

如果你需要的仅是单个的一个值，那么你可以使用 **db_result()** 来回显该值。下面是一个例子，用来回显已发布博客日志的总数：

```
$sql = "SELECT COUNT(*) FROM {node} WHERE type = 'blog' AND status = 1";
```

```
$total = db_result(db_query($sql));
```

取回多行

大多数情况，你需要从数据库中返回的都多于一个字段。下面是一个典型的迭代方式用于遍历整个结果集：

```
$sql = "SELECT * FROM {node} WHERE type = 'blog' AND status = 1";
```

```
$result = db_query(db_rewrite_sql($sql));
```

```
while ($data = db_fetch_object($result)) {
```

```
  $node = node_load($data->nid);
```

```
print node_view($node, TRUE);  
  
}
```

上面的代码片段将输出所有的已发布的类型为 **blog** 的节点（表 **node** 中的字段 **status** 的值为 **0** 时，意味着未发布，为 **1** 时意味着已发布）。我们在接下来会讲解 **db_rewrite_sql**（）。函数 **db_fetch_object**（）从结果集中取出一行作为一个对象。将取出的结果作为一个数组可以使用 **db_fetch_array**（）。前者更为常用，这是因为大多数开发者觉得它更简洁一些。

取回限制范围的结果

你可能会觉得，在一个比如说有 **10,000** 个日志的站点上运行上面的代码将会是一个危险的想法。我们将限制这个语句的结果，**10** 个最新发布的日志：

```
$sql = "SELECT * FROM {node} n WHERE type = 'blog' AND status = 1 ORDER  
BY  
n.created DESC";
```

```
$result = db_query_range(db_rewrite_sql($sql), 0, 10);
```

我们没有将语句传递到 **db_query**（）和使用 **LIMIT** 条件语句，我们使用了函数 **db_query_range**（）。为什么？因为并非所有的数据库都支持 **LIMIT** 语句，所以我们需要使用 **db_query_range**（）作为包裹函数。

如果你还有一些没有硬编码的参数，你需要将这些用来填充占位符的变量放到范围的前面（所以 **type** 和 **status** 位于 **0, 10** 之前，如下面的例子所示）：

```
$type = 'blog';
```

```
$status = 1;
```

```
$sql = "SELECT * FROM {node} n WHERE type = '%s' AND status = %d ORDER  
BY  
n.created DESC";
```

```
$result = db_query_range(db_rewrite_sql($sql), $type, $status, 0, 10);
```

将结果分页显示

我们可以使用一个更好的方式展示这些日志：分页显示。我们可以使用 **Drupal** 的分页器 (**pager**)来这样做。让我们在此取回所有的日志，而这次我们将分页显示它们，使用指向更多结果页面的链接和位于最下面的“第一页和最后一页”的链接。

```
$sql = "SELECT * FROM {node} n WHERE type = 'blog' AND status = 1 ORDER BY
```

```
n.created DESC"
```

```
$result = pager_query(db_rewrite_sql($sql), 0, 10);
```

```
while ($data = db_fetch_object($result)) {
```

```
$node = node_load($data->nid);
```

```
print node_view($node, TRUE);
```

```
}
```

```
// Add links to remaining pages of results.
```

```
print theme('pager', NULL, 10);
```

虽然 `pager_query()` 不是数据库抽象层的真正的一部份，当你需要创建一个分页显示的结果时，你需要知道它。最后一行的调用 `theme('pager')`将展示导航到其他页面的链接，你不需要向 `theme('pager')`传递结果的总数，因为结果总数在调用 `pager_query()` 时已被记下。

使用临时表

如果你需要做很多的处理，在请求流程中你可能需要创建一个临时表。你可以通过调用函数 `db_query_temporary()`来完成它，如下面的格式：

```
$result = db_query_temporary($sql, $arguments, $temporary_table_name);
```

接下来你可以使用临时表的名称来对临时表进行查询。例如，在 **Drupal** 的搜索模块 (`search module`)中的函数 `do_search()`，搜索是在多个阶段中完成的，它使用了临时表来保存中间信息。下面是常用方式，这里被简化了（学习 `search.module` 可了解全部实现，它可能是令人头痛的事）：

```
// Select initial search results into temporary table named 'temp_search_sids'.
```

```
$result = db_query_temporary("

SELECT i.type, i.sid, SUM(i.score * t.count) AS relevance, COUNT(*) AS
matches

FROM {search_index} i

INNER JOIN {search_total} t ON i.word = t.word $join1

WHERE $conditions

GROUP BY i.type, i.sid

HAVING COUNT(*) >= %d",

$args, 'temp_search_sids');

...

// Later: calculate maximum relevance, to normalize it, using temporary table.

$normalize = db_result(db_query('SELECT MAX(relevance) FROM
temp_search_sids'));

...

// Still later: create a temporary search results table named
'temp_search_results'.

$result = db_query_temporary("

SELECT i.type, i.sid, $select2

FROM temp_search_sids i

INNER JOIN {search_dataset} d

ON i.sid = d.sid AND i.type = d.type $join2

WHERE $conditions $sort_parameters",

$args, 'temp_search_results');

...
```

```
// Finally: do actual search query.
```

```
$result = pager_query("SELECT * FROM temp_search_results", 10, 0,  
$count_query);
```

注意，临时表不需要使用花括号以对表进行前缀化，这是因为临时表的存在是暂时的，他不能出发表的前缀化流程。与之相对应的，永久的表的名称位于花括号内部以支持表的前缀化。

Hezi

第 5 章 Drupal 数据库层 (2)

使用 `hook_db_rewrite_sql()` 将查询暴露给其它模块

这个钩子用来修改 Drupal 中任何地方的查询，这样你就不用直接修改相关模块了。如果你将一个查询传递给 `db_query()`，而且你相信其他人可能想修改它，那么你就需要把它包装到 `db_rewrite_sql()` 里面，这样其他的开发者就可以访问它了。当执行一个这样的查询时，它首先检查所有实现了 `db_rewrite_sql` 钩子的模块，给它们一个修改查询的机会。例如，节点模块修改了节点列表查询，从而将受到节点访问规则保护的节点排除在外。

警告 如果你执行一个节点列表查询（例如，你直接对 `node` 表查询来获取所有节点的一个子集），但是你没有使用 `db_rewrite_sql()` 来包装你的查询，那么节点访问规则将被忽略，这是由于节点模块没有机会修改你的查询来排除受保护的节点。

如果发出查询的人，但是你想在你的模块中有机会修改他人的查询，那么需要在你的模块中实现这个钩子。

表 5-2 使用 `db_rewrite_sql()` 的两种方式的总结

表 5-2. 什么时候使用 `db_rewrite_sql()` 函数 VS 使用 `db_rewrite_sql()` 钩子

名称	什么时候使用
<code>db_rewrite_sql()</code>	当编写节点列表查询或者其他查询时，你想让别人能够修改它的时候

<code>hook_db_rewrite_sql()</code>	当你修改其它模块中的查询时
------------------------------------	---------------

包装查询

下面是函数签名：

```
function hook_db_rewrite_sql($query, $primary_table = 'n', $primary_field = 'nid',
```

```
$args = array())
```

参数如下：

- `$query`: 可被覆写的 SQL 查询。

-
- **\$primary_table**: 在该查询中, 包含主键字段的表的别名。例如, 其值可为 **n** 或者 **c** (例如., 对于 `SELECT nid FROM {node} n`, 该值应为 **n**)。
 - **\$primary_field**: 在该查询中主键字段的名称。它的值可为 **nid, tid, vid, cid**, 等等。(例如, 如果你的查询要得到一系列节点 ID, 那么主字段应该为 **nid**)。
 - **\$args**: 传递给 `hook_db_rewrite_sql()` 实现的一个包含参数的数组。

修改其它模块的查询

让我们看一个该钩子的一个实现。下面的例子利用了节点表中 **moderate** (适度的) 列来覆写节点查询。在我们修改了查询以后, 处于 **moderated** 状态的节点 (例如, **moderate** 列为 **1**), 对于不具有“**administer content**”权限的用户将被隐藏起来。

```
/**
 * Implementation of hook_db_rewrite_sql().
 */
function moderate_db_rewrite_sql($query, $primary_table, $primary_field,
$args) {
  switch ($primary_field) {
    case 'nid':
      // Run only if the user does not already have full access.
      if (!user_access('administer content')) {
        $array = array();
        if ($primary_table == 'n') {
          // Node table is already present;
          // just add a WHERE to hide moderated nodes.
          $array['where'] = "(n.moderate = 0)";
        }
      }
    }
}
```

```
// Test if node table is present but alias is not 'n'.

elseif (preg_match('@{node} ([A-Za-z_]+)@', $query, $match)) {

    $node_table_alias = $match[1];

    // Add a JOIN so that the moderate column will be available.

    $array['join'] = "LEFT JOIN {node} n ON $node_table_alias.nid = n.nid";

    // Add a WHERE to hide moderated nodes.

    $array['where'] = "($node_table_alias.moderate = 0)";

}

return $array;

}

}

}
```

注意我们检查任何主键为 **nid** 的查询，并向这些查询中插入一些额外信息。让我们看一下实际效果。

下面是最初的查询，在 `moderate_db_rewrite_sql()` 处理以前的：

```
SELECT * FROM {node} n WHERE n.type = 'blog' and n.status = 1
```

下面是 `moderate_db_rewrite_sql()` 处理过后的查询：

```
SELECT * FROM {node} n WHERE n.type = 'blog' and n.status = 1 AND
n.moderate = 0
```

调用 `moderate_db_rewrite_sql()` 后，它向输入的查询中追加了 `AND n.moderate = 0`。这个钩子通常用来限制对查看节点、词汇表、词语、或者评论的访问。`db_rewrite_sql()` 仅限于它能够理解的 **SQL** 语法。当你需要使用 **JOIN** 语法时，千万不要用 **FROM** 语句来代替。

下面的不正确：

```
SELECT * FROM node AS n, comment AS c WHERE n.nid = c.nid
```

这个正确：

```
SELECT * FROM node n INNER JOIN comment c on n.nid = c.nid
```

在 Drupal 中连接多个数据库

数据库抽象层使得函数名更容易记忆，它同时还在查询中添加了内置的安全特性。有时，我们需要连接到第 3 方或者遗留的数据库上，如果 **Drupal** 数据库 **API** 能满足这些需要并同时提供安全特性的话，那该多好啊。幸好，我们可以！

在 `settings.php` 文件中，`$db_url` 既可以是一个字符串（通常是这样的）也可以是包含多个数据库连接字符串的数组。下面是默认的语法，声明了一个单独的连接字符串：

```
$db_url = 'mysql://username:password@localhost/databasename';
```

当使用一个数组时，它的键是一个在激活数据库连接时所引用的简洁名称，而它的值就是连接字符串本身。下面是一个例子，在这里我们声明了两个连接字符串，默认的（**default**）和遗留的（**legacy**）：

```
$db_url['default'] = 'mysql://user:password@localhost/drupal5';
```

```
$db_url['legacy'] = 'mysql://user:password@localhost/legacydatabase';
```

注意 **Drupal** 本身使用的数据库一定要以 **default** 为键。

当你需要连接到 **Drupal** 中其它的数据库上时，你首先使用它的键名激活该连接，当你使用完连接时，将它切换回到默认的连接上。

```
// Get some information from a non-Drupal database.

db_set_active('legacy');

$result = db_query("SELECT * FROM ldap_user WHERE uid = %d",
$user->uid);

// Switch back to the default connection when finished.

db_set_active('default');
```

注意 切记一定要切换回到默认的连接上，这样 **Drupal** 可以干净的完成整个请求生命周期并将它写入到自己的表中。

由于数据库抽象层设计的是每个数据库使用同样的函数名，所以多个数据库后台（比如，**MySQL** 和 **PostgreSQL**）不能够同时使用。然而，在同一个站点如何同时使用 **MySQL** 和 **PostgreSQL** 连接呢，相关信息请参看 <http://drupal.org/node/19522>。

使用模块的 `.install` 文件

在第 2 章我们已经看到了，当我们编写一个模块，它需要创建一个或多个数据库表来存储信息时，创建和维护表结构的 **SQL** 放在 `.install` 文件中，该文件与模块一同发布。通常包括的 **SQL** 主要是针对最常用的数据库系统的（**MySQL** 和 **PostgreSQL**）。

创建表

一个名为 `$db_type` 的全局变量决定了当前使用的数据库类型。在下面的例子中，一个 `hook_install` 函数分别为 MySQL 和 PostgreSQL 包含了相应的 `CREATE TABLE` 语句。下面的例子来自于：

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

function book_install() {
  switch ($GLOBALS['db_type']) {
    case 'mysql': // Use same as mysqli.
    case 'mysqli':
      db_query("CREATE TABLE {book} (
        vid int unsigned NOT NULL default '0',
        nid int unsigned NOT NULL default '0',
        PRIMARY KEY (vid),
        KEY nid (nid),
      ) /*!40100 DEFAULT CHARACTER SET UTF8 */");
      break;
    case 'pgsql':
      db_query("CREATE TABLE {book} (
        vid int_unsigned NOT NULL default '0',
        nid int_unsigned NOT NULL default '0',
```

PRIMARY KEY (vid)

");

db_query("CREATE INDEX {book}_nid_idx ON {book} (nid)");

break;

}

}

注意前面用于 **MySQL** 的表创建语句中下面的有点古怪的代码:

```
/*!40100 DEFAULT CHARACTER SET UTF8 */
```

*/**意味着一个内嵌评论的开始，它在**/*处结束（这是标准的评论语法，可用于许多语言，包括 C 和 PHP）。这意味着，当使用的是一个不同的数据库时，注释定界符内部的代码将被忽略。如果开定界（*/**）符后面紧跟了一个感叹号（**!**），那么 **MySQL** 将尝试解析并执行评论定界符内部的代码。如果感叹号（**!**）后面紧跟了一个 **MySQL** 版本号的话，只有当 **MySQL** 的版本等于或者高于给定的版本时，才会执行里面的代码。所以，前面的代码所表示的实际意思就是，“如果执行 **CREATE TABLE** 语句时，所用到的 **MySQL** 数据库的版本等于或者高于 **4.1**，为该表使用 **UTF-8** 作为默认的文本编码”。

Maintaining Tables

When you create a new version of a module, you might have to change the database schema.

Perhaps you've added a column to support page ranking in the book module, and you have an

installed base of users. Here's how their databases will be updated:

当你创建一个模块的新的版本时，你可能想要修改数据库的 **schema**。可能你在你 **book**（书）模块中添加了一列来支持页面等级，而且有许多用户使用了该模块。下面就是如何对他们的数据库进行更新：

1. 更新 `install` 钩子中的 `CREATE TABLE` 语句，这样你模块的最新用户将使用最新的 `schema` 来安装模块：

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

function book_install() {
  switch ($GLOBALS['db_type']) {
    case 'mysql': // use same as mysql

    case 'mysqli':
      db_query("CREATE TABLE {book} (
        vid int unsigned NOT NULL default '0',
        nid int unsigned NOT NULL default '0',
        rank int unsigned NOT NULL default '0',
        PRIMARY KEY (vid),
        KEY nid (nid),
      ) /*!40100 DEFAULT CHARACTER SET UTF8 */");
      break;
    case 'pgsql':
      db_query("CREATE TABLE {book} (
        vid int_unsigned NOT NULL default '0',
        nid int_unsigned NOT NULL default '0',
```

```
rank int_unsigned NOT NULL default '0',  
  
PRIMARY KEY (vid)  
  
)");  
  
db_query("CREATE INDEX {book}_nid_idx ON {book} (nid)");  
  
break;  
  
}  
  
}
```

2. 通过编写一个更新函数为已有用户提供一个更新的方法。更新函数使用序号来命名，从 1 开始：

```
function book_update_1() {  
  
$items = array();  
  
$items[] = update_sql("ALTER TABLE {book} ADD COLUMN rank int_unsigned  
NOT NULL default '0'");  
  
}
```

在升级了该模块以后，当用户运行 <http://example.com/update.php> 调用这个函数。

提示 **Drupal** 追踪了一个模块当前使用的 **schema** 版本。该信息存放在 **system** 表中。为了让 **Drupal** 忘记它，可以使用 **devel** 模块的 **Reinstall Modules** 选项，或者直接从 **system** 表中删除该模块的记录。

在 Uninstall 上删除数据库表

在 **Administer > Modules** 页面有一个 **Uninstall** 标签，它不仅允许禁用模块，还能够从数据库中将它们的数据删除掉。如果你想要在这个页面为你模块的数据库表启用删除选项，那么你需要在你的模块的 `.install` 文件中实现 `uninstall` 钩子。你可能想同时删除你定义的所有变量。下面是我们在第 2 章编写的 `annotation` 模块中的例子：

```
function annotate_uninstall() {  
  
  db_query("DROP TABLE {annotations}");  
  
  variable_del('annotate_nodetypes');  
  
}
```

Writing Your Own Database Abstraction Layer

编写你自己的数据库抽象层

假定你想为一个新的将来的名为 `DNAbase` 的数据库编写一个数据库抽象层，该数据库使用分子计算来提升性能。我们没有从头开始，而是将复制一份已存在的抽象层，接着修改它。我们将使用 `MySQL` 的实现，这是由于 `MySQL` 是 `Drupal` 中最常用的数据库。

首先，我们复制一份 `includes/database.mysql.inc` 并将其重命名为 `includes/database.dnabase.inc`。接着我们修改每个包装函数内部的逻辑，使用 `DNAbase` 的功能来代替 `MySQL` 的功能。当我们完成了所有的这些修改以后，

那么在我们的文件中声明了下面的函数：

```
_db_query($query, $debug = 0)  
  
db_affected_rows()
```

db_connect(\$url)

db_decode_blob(\$data)

db_distinct_field(\$table, \$field, \$query)

db_encode_blob(\$data)

db_error()

db_escape_string(\$text)

db_fetch_array(\$result)

db_fetch_object(\$result)

db_lock_table(\$table)

db_next_id(\$name)

db_num_rows(\$result)

db_query_range(\$query)

db_query_temporary(\$query)

db_result(\$result, \$row = 0)

db_status_report(\$phase)

db_table_exists(\$table)

db_unlock_tables()

db_version()

通过更新 `settings.php` 中的 `$db_url`, 我们在 `Drupal` 中连接到 `DNABase` 数据库来测试该系统。它看起来这样:

```
$db_url = 'dnabase://john:secret@localhost/mydnadatabase';
```

其中 **john** 是用户名, **secret** 是密码, 而 **mydnadatabase** 是我们将要连接的数据库名你可能还想创建一个测试模块来调用直接这些模块以保证它们正常工作。

总结

读完本章后, 你应该能够

- 理解 **Drupal** 的数据库抽象层
- 进行基本的查询
- 从数据库中获取单个或者多个结果
- 获取一个限定范围内的结果
- 使用分页器
- 编写其它开发者可以修改的查询
- 干净的修改其它模块中的查询
- 连接多个数据库, 包括遗留的数据库
- 编写一个抽象层代码库

第 6 章 Drupal 用户 (1)

用户是使用 **Drupal** 的重要原因。**Drupal** 可以帮助用户创建，协作，交流，并形成一個在线社区。在本章，我们将解开用户的内幕，看一下如何验证用户，用户登陆，还有用户的内部表示。我们从检查对象 `$user` 是什么以及它是如何构造的。然后我们进一步讲述了用户注册，用户登录，用户验证的流程。最后我们讲述了如何将一个现有的验证系统例如 **LDAP**（轻量级目录访问协议）和 **Pubcookie** 等等与 **Drupal** 集成。

对象 `$user`

用户为了登录，必须启用 **cookies**。一个关闭了 **cookie** 的用户仍然可以以匿名的身份与 **Drupal** 交互。

在引导指令流程的会话阶段，**Drupal** 创建了一个 `$object` 对象，用来作为当前用户的标识。如果用户没有登录（这样就没有一个会话 **cookie**），那么它将被看作匿名用户对待。创建匿名用户的代码如下所示（位于 `bootstrap.inc`）：

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

另一方面，如果用户当前登录了，通过使用用户的 **ID** 来关联表 `user` 和表 `session` 来创建对象 `$object`。两个表中的所有字段都没放到了对象 `$user` 中。

注意：用户的 **ID** 是在用户注册时或者管理员创建用户时所分配的一个整数。最近使用 **ID** 存储在表 `sequences` 中。

通过项 `index.php` 中添加代码 `global $user; print_r($user);` 可以很容易的查看 `$user` 对象。下面是一个登录用户对象的通常结构：

`stdClass Object (`

`[uid] => 2`

`[name] => Joe Example`

`[pass] => 7701e9e11ac326e98a3191cd386a114b`

`[mail] => joe@example.com`

`[mode] => 0`

`[sort] => 0`

`[threshold] => 0`

`[theme] => chameleon`

`[signature] => Drupal rocks!`

`[created] => 1161112061`

`[access] => 1161113476`

`[login] => 1161112317`

`[status] => 1`

`[timezone] => -18000`

`[language] => en`

`[picture] => files/pictures/picture-2.jpg`

`[init] => joe@example.com`

`[data] =>`

`[roles] => Array ([2] => authenticated user)`

`[sid] => fq5vvn5ajvj4sihli314ltsqe4`

```

[hostname] => 127.0.0.1

[timestamp] => 1161113476

[cache] => 0

[session] => user_overview_filter|a:0:{}

)

```

在上面展示的 **\$object** 对象中，斜体字段意味着数据来自于 **session** 表。表 6-1 解释了 **\$user** 对象的组成部分：

表 6-1 **\$object** 的组成部分

组成	描述
来自于表 user	
uid	用户的 ID.它是表 users 的主键并在 Drupal 安装中是唯一的。
Name	用户的用户名，当用户登录时输入
Pass	用户的 MD5 哈希化的密码，当用户登录时进行对比。由于没有保存用户的原始真实密码，所以密码只能被重置，不能被恢复。
Mail	用户当前的 email 地址
Mode,sort 和 threshold	特定于用户的评论浏览喜好
theme	如果启用了多个主题，这个代表用户选择的主题。如果用户主题未安装， Drupal 将其转到站点的默认主题。
Signature	用户进入他/她的帐户页面时所使用的签名。当用户添加一个评论时使用。仅当评论模块(comment module)启用时可见
Created	用户帐号创建时的一个 Unix 时间戳
Access	用户最近一次访问的 Unix 时间戳
Login	用户最近一次登录的 Unix 时间戳

Status	1 表示良好, 0 表示被拒绝访问的用户
Timezone	用户时区与 GMT 之间的差异, 以秒为单位
Language	用户的默认语言, 通过 <code>common.inc</code> 中的 <code>local_initialize()</code> 设置
Picture	与用户帐号相联系的图像文件的路径
Init	用户注册时提供的初始 <code>email</code> 地址
Data	由模块存储的任何数据都可放置在这里 (参看下一节, “向 <code>\$user</code> 对象存储数据”)
来自于表 <code>user_role</code>	
roles	分配给当前用户的角色
来自于表 <code>session</code>	
sid	通过 PHP 分配给当前用户会话的会话 ID
hostname	用户浏览当前页面时所使用的 IP 地址
timestamp	一个 Unix 时间戳, 表示用户的浏览器最后一次收到一个完整页面的时间
cache	一个用于 <code>per-user caching</code> (参看 <code>cache.inc</code>) 的时间戳
session	在用户会话期间, 模块可以向这里存储任意的数据。

向对象 `$user` 中存储数据

表 `users` 包含了一个名为 `data` 的一列用于存储保存在序列化数组中的额外信息。如果你向对象 `$user` 中添加你自己的数据, 它将会通过 `user_save()` 存储在这一列上。

```
// Add user's disposition.
```

```
global $user;
```

```
$extra_data = array('disposition' => t('Grumpy'));
```

```
user_save($user, $extra_data);
```

现在对象 `$user` 拥有了一个永久属性:

```
global $user;
```

```
print $user->disposition;
```

Grumpy

尽管这种方式很方便，但是当用户登录和初始化对象 `$user` 时，由于以这种方式存储的数据需要反序列化，这样会增加更多的开销。因此，不加考虑就向 `$user` 中放入大量的数据将会引起一个性能瓶颈。一个可选的并且是更好的在 `$user` 加载时向它添加属性方法，将会在接下来讨论。

测试用户是否登录了

测试用户是否登录的标准方式是看一下 `$user->uid` 是否为 0:

```
global $user;
```

```
if ($user->uid) {
```

```
    $output = t('User is logged in!');
```

```
else {
```

```
    $output = t('User is an anonymous user.');
```

```
}
```

当定义一个 **PHP** 类型的区块，以向登录用户展示特定信息，向匿名用户展示其他信息时，常常使用这种方式。

```
<?php  
  
global $user;  
  
if ($user->uid) {  
  
return t('You are currently logged in!');  
  
}  
  
else {  
  
return t('You are not currently logged in.');
```

```
?>
```

HEZIL

第 6 章 Drupal 用户 (2) hook_user, 用户注册和登录流程

hook_user() 入门 (Introduction to hook_user())

在你的模块中实现钩子 `hook_user()` 使你能够对用户帐号进行不同的操作，以及修改 `$user` 对象。着我们看一下这个函数的签名：

```
function hook_user($op, &$amp;edit, &$amp;user, $category = NULL)
```

参数 `$op` 用来描述对用户帐号所进行的当前操作，它可以有多个不同的值：

- **after_update**: 在 `$user` 被保存到数据库中以后调用。
- **categories**: 返回一个关于分类的数组，当用户编辑用户帐号时，这些分类被当作 **Drupal** 菜单本地任务 (**menu local tasks**)。参看 `profile.module` 中 `profile_user()` 的一个实现。
- **delete**: 刚刚从数据库中删除了一个用户。这给了模块一个机会，用来从数据库中删除与该用户相关的信息。
- **form**: 在被展示的用户编辑表单中插入一个额外的表单字段元素。
- **insert**: 一个新的用户帐号将要被创建并保存到数据库中。
- **login**: 用户已经成功登录。
- **logout**: 用户刚刚退出登录，他/她的会话已被销毁。
- **load**: 已成功加载用户帐号。模块可以向 `$user` 对象添加额外的信息。
- **register**: 用户帐号注册表单将被展示。模块可以向表单中添加额外的表单元素。
- **submit**: 用户编辑表单已被提交。在帐号信息发送给 `user_save()` 以前可对其进行修改。
- **update**: 已存在的用户帐号 (修改后) 将要被保存到数据库中。
- **validate**: 用户帐号已被修改。模块应该验证它的定制数据并生成任何必要的错误消息。
- **view**: 正在展示用户的帐号信息。模块应该把它要添加到展示页面中的定制信息放到一个数组中并将数组返回。查看操作最后调用 `theme_user_profile` 来生成用户个人资料页面。马上就可以看到更详细的解释了。

参数 `$edit` 是一个数组，当用户帐号正被创建或者更新时提交的表单数值组成了这一数组。注意它是通过引用传递的，所以你对它所做的任何修改都将实际的改变表单数值。

`$user` 对象也是通过引用传递的，所以你对它所做的任何修改都将实际的改变 `$user` 信息。

参数 `$category` 是正被编辑的当前用户帐号的分类。

警告 不要混淆了 `hook_user()` 中的参数 `$user` 和全局变量 `$user` 对象。参数 `$user` 是当前正被操作的帐号的用户对象。而全局变量 `$user` 对象是当前登录的用户。

理解 `hook_user('view')` (Understanding `hook_user('view')`)

模块可以使用 `hook_user('view')` 来向用户个人资料页面添加信息 (例如，你在 <http://example.com/?q=user/1> 看到的；参看图 6-1)

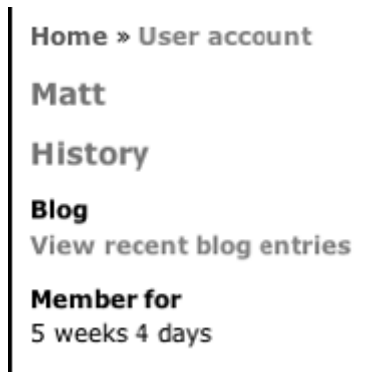


图 6-1 用户个人资料页面，这里日志模块和用户模块使用 `hook_user('view')` 添加了额外的信息

让我们看一下日志模块是如何向这一页面添加它的信息的：

```
function blog_user($op, &$edit, &$user) {  
  
  if ($op == 'view') {  
  
    $items['blog'] = array(  
  
      'title' => t('Blog'),  
  
      'value' => l(t('View recent blog entries'), "blog/$user->uid"),  
  
      'class' => 'blog', // CSS selector class to add.  
  
    );  
  
    return array(t('History') => $items);  
  
  }  
}
```

```
}
```

查看操作返回了一个关联数组的关联数组。外面的数组使用分类名称作为键。在前面的例子中它是 **History**（历史）。内部的数组应该有一个唯一的文本键（在这里为 **blog**）并带有 **title**, **value**, 和 **class** 元素。比较该代码片段和图 6-1，你将看到这些元素是如何被显示的。

在调用 **user.module** 中的 **theme_user_profile()** 以前，你的模块也可以通过实现 **hook_profile_alter()** 来操作个人资料项目，。下面这个的例子，简单的从你的用户个人资料页面删除了日志项目：

```
/**
 * Implementation of hook_profile_alter().
 */
function hide_profile_alter(&$account, &$fields) {
  unset($fields['History']['blog']);
}
```

用户注册流程（The User Registration Process）

默认情况下，**Drupal** 站点的用户注册只需要一个用户名和一个有效的 **e-mail** 地址就可以了。模块可以通过实现用户钩子来向用户注册表单添加它们自己的字段。让我们编写一个名为 **legalagree.module** 的模块，它提供了一个快速的方式来使你的站点适应今天这个好诉讼的社会。

首先在 **sites/all/modules/custom** 下面创建一个名为 **legalagree** 的文件夹，并向 **legalagree** 目录添加下面的文件（参看列表 6-1 和 6-2）。接着通过 **Administer > Site building > Modules** 来启用模块。

列表 6-1. **legalagree.info**

```
; $Id$

name = Legal Agree

description = Displays a dubious legal agreement during user registration.

version = "$Name$"
```

列表 6-2. **legalagree.module**

```
<?php
// $id$

/**
 * @file
 * Support for dubious legal agreement during user registration.
 */

/**
 * Implementation of hook_user().
 */
function legalagree_user($op, &$edit, &$user, $category = NULL) {
  switch($op) {
    // User is registering.
    case 'register':
      // Add a fieldset containing radio buttons to the
      // user registration form.
      $fields['legal_agreement'] = array(
        '#type' => 'fieldset',
        '#title' => t('Legal Agreement')
      );
      $fields['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'))
);

return $fields;

// Field values for registration are being checked.

// (Also called when user edits his/her 'my account' page, but
// $edit['decision'] is not set in that case.)

case 'validate':

// Make sure the user selected radio button 1 ('I agree').

// the validate op is reused when a user updates information on
// The 'my account' page, so we use isset() to test whether we are
// on the registration page where the decision field is present.
if (isset($edit['decision']) && $edit['decision'] != '1') {
form_set_error('decision', t('You must agree to the Legal Agreement
before registration can be completed.));
}

return;

// New user has just been inserted into the database.

case 'insert':

// Record information for future lawsuit.

watchdog('user', t('User %user agreed to legal terms',
array('%user' => $user->name)));

return;
}
```


}

在创建注册表单期间，在表单验证期间，还有在用户记录被插入到数据库中以后，都要调用用户钩子。我们这个简洁的模块所导致的注册表单类似于图 6-2 所示的。

Account information

Username: *
Joe Example
Your full name or your preferred username: only letters, numbers and spaces are allowed.

E-mail address: *
joe@example.com
A valid e-mail address. All e-mails from the system will be sent to this address. The e-mail address is not made public and will only be used if you wish to receive a new password or wish to receive certain news or notifications by e-mail.

Legal Agreement

I disagree
 I agree

By registering at *Betty's Golden Earplugs*, you agree that at any time, we (or our surly, british henchmen) may enter your place of residence and smash your belongs with a ball-peen hammer.

Create new account

图 6-2 一个修改了的用户注册表单

使用 profile.module 来收集用户信息

如果你想扩展用户注册表单来收集用户信息，在你打算编写自己的模块以前，你可以先试用一下 **profile.module**。它允许你创建任意的表单来收集数据，在用户注册表单上定义信息是否是必须的（或者收集的），指定信息是公开的还是私有的。例外，它允许管理员定义页面，这样就可以根据用户的个人资料选项，使用一个由“站点 URL” + “profile/” + “个人资料字段的名称” + “值”（参看图 6-3）构建的 URL，来查看用户了。

Profiles

Here you can define custom fields that users can fill in in their user profile (such as *country*, *real name*, *age*, ...).

Title	Name	Type	Category	<small>[more help...]</small> Operations
Vegetarian	profile_is_veggie	checkbox	2007 Conference	edit delete
First name	profile_first	single-line textfield	Personal information	edit delete
Last name	profile_last	single-line textfield	Personal information	edit delete

Add new field

- single-line textfield
- multi-line textfield
- checkbox
- list selection
- freeform list
- URL
- date

图 6-3 用来创建额外用户个人资料字段的页面

例如，如果你定义了一个名为 `profile_color` 的文本字段，你可以使用 http://example.com/?q=profile/profile_color/black 来查看所有的选择了黑色作为他们喜欢的颜色的用户。或者假定你正在创建一个会议网站，并负责为参加者计划宴会。你可以定义一个名为 `profile_vegetarian` 的复选框作为个人资料字段，并可在 http://example.com/?q=profile/profile_vegetarian（注意，对于复选框，已隐含了值，因此这里忽略了它）查看所有的素食用户。

在 Drupal 官方网站 <http://drupal.org> 上可以找到一个实际中的例子，2006 年参加加拿大范库弗峰地区 Drupal 会议的用户列表，可以在地址 <http://drupal.org/profile/conference-vancouver-2006> 中看到（这里，字段名前面没有加前缀“`profile_`”）。

提示 只有在个人资料字段设置中填充了字段 `Page` 的标题时，自动根据个人资料创建总结页面才正常工作，但它不适用于 `textarea`，`URL`，或者日期字段。

登录流程 (The Login Process)

当用户填完登录表单(一般位于 <http://example.com/?q=user> 或者在一个区块中) 并点击登录按钮时, 登录流程开始。

登录表单的验证程序检查用户名是否被封锁了, 无论是根据访问规则拒绝访问, 还是由于用户输入了一个错误的密码。如果任何一种情况发生了, 都会及时的通知用户。

在 **Drupal** 中可以使用本地和外部认证两种方式。外部认证系统的; 例子包括 **LDAP**, **Pubcookie**, **Sxip**, 以及其它。一种外部认证类型是分布式认证, 在这里来自于一个 **Drupal** 站点的用户可以登录到另一个 **Drupal** 站点 (参看 **Drupal** 内核中的 **drupal.module**)。

Drupal 将首先尝试从本地登录, 在 **users** 表中查找是否存在匹配用户名和密码哈希值的记录。如果不成功, **Drupal** 将尝试外部认证 (参看图 6-4)。成功的本地登录将导致调用两个用户钩子 (**load** 和 **login**), 在你的模块中可以实现这些钩子。

HEZIL

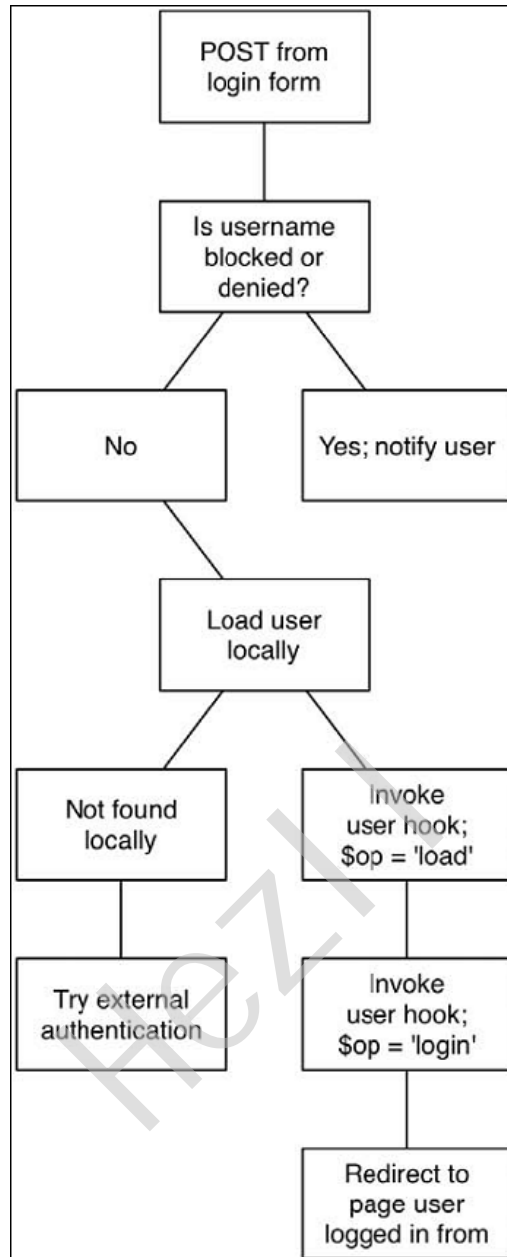


图 6-4 用户登录的执行路径

向\$user 对象添加数据

通过调用 `user_load()`，从数据库中成功的加载一个 `$user` 对象时，将调用用户钩子的加载（load）操作。当一个用户登录（或退出）时，当从一个节点取回作者信息时，或者一些其它情况时，都会调用这一加载（load）操作。

注意 为了性能优化，当为一个请求实例化当前 `$user` 对象时（参看前面的“`$user` 对象”部分），没有调用 `user_load()`。如果你正在编写你自己的模块，在调用一个需要用到完整

加载了的 `$user` 对象的函数以前，总要调用 `user_load()`，除非你能保证已经完整加载了 `$user` 对象。

让我们创建一个名为“`loginhistory`”的模块，用来保存用户登陆的历史记录。在 `sites/all/modules/custom/` 下面创建一个名为 `loginhistory` 的文件夹，并添加以下文件（参看列表 6-3 到 6-5）。首先是 `loginhistory.module`。

列表 6-3. `loginhistory.module`

```
<?php

// $Id$

/**
 * @file
 * Keeps track of user logins.
 */

/**
 * Implementation of hook_user().
 */

function loginhistory_user($op, &$edit, &$account, $category = NULL) {
  switch($op) {

    // Successful login.

    case 'login':

      // Record timestamp in database.

      db_query("INSERT INTO {login_history} (uid, timestamp) VALUES (%d, %d)",
        $account->uid, $account->login);

      break;

    // $user object has been created and is given to us as $account parameter.

    case 'load':

      // Add the number of times user has logged in.
```

```

$account->loginhistory_count = db_result(db_query("SELECT
COUNT(timestamp) AS

count FROM {login_history} WHERE uid = %d", $account->uid));

break;

// 'My account' page is being created.

case 'view':

// Add a field displaying number of logins.

$items['login_history'] = array(

'title' => t('Number of Logins'),

'value' => $account->loginhistory_count,

'class' => 'member'

);

return array(t('History') => $items);

}

}

```

我们需要一个 `.install` 文件来创建数据库表来存储登录信息，所以说我们创建 `loginhistory.install`。

列表 6-4. `loginhistory.install`

```

<?php

// $Id$

/**

 * Implementation of hook_install().

 */

function loginhistory_install() {

switch ($GLOBALS['db_type']) {

case 'mysql':

```

```
case 'mysql':

db_query("CREATE TABLE {login_history} (

uid int NOT NULL default '0',

timestamp int NOT NULL default '0',

KEY (uid)

) /*!40100 DEFAULT CHARACTER SET UTF8 */");

break;

case 'pgsql':

db_query("CREATE TABLE {login_history} (

uid int_unsigned default '0',

timestamp int_unsigned NOT NULL default '0',

KEY (uid)

)");

break;

}

}

/**

* Implementation of hook_uninstall().

*/

function loginhistory_uninstall() {

db_query("DROP TABLE {login_history}");

}
```

下面是 loginhistory.info 文件

列表 6-5. loginhistory.info

```
; $Id$
```

```
name = Login History
```

```
description = Keeps track of user logins.
```

```
version = "$Name$"
```

在安装这个模块以后，每次成功的用户登录都将调用用户登录钩子，在钩子中模块将向数据库表 `login_history` 中插入一条记录。当加载 `$user` 对象时，将调用用户加载钩子，此时模块将把用户的当前的登录次数添加 `$user->loginhistory_count`。当用户查看“my account”（“我的帐号”）页面时，登录次数将被展示出来如图 6-5 所示。

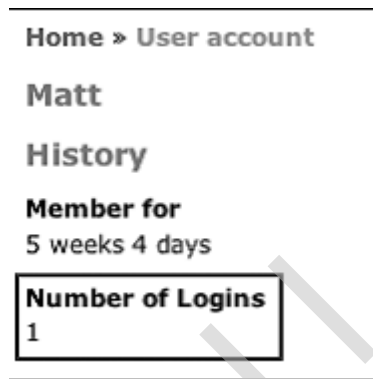


图 6-5 追踪用户的登录历史

注意 当你在你的模块中为对象 `$user` 或 `$node` 添加变量时，在变量名前面最好加上前缀，以避免命名空间的冲突。这就是为什么这里使用 `loginhistory_count` 来代替 `count` 的原因。

尽管在“my account”页面，我们展示了我们添加到 `$user` 上的额外的信息，记住由于 `$user` 对象是全局变量，其它模块也能访问它。我们留给读者一个非常有用的联系，为了安全性（“喂！我今天上午 3:00 没有登录”），修改前面的模块，在左（或右）栏的区块中来提供一个美观的历史登陆列表。

第 6 章 Drupal 用户 (3)

提供用户信息分类

如果你在 <http://drupal.org> 拥有一个帐号，通过登录并点击“my account”链接，接着选择编辑标签（edit tab），你就可以看到提供关于用户信息的分类的效果。除了编辑你的帐号信息比如你的密码以外，你可以在其它分类中你还可以提供你的其它个人信息。在编写此书时，<http://drupal.org> 支持编辑 CVS 信息、Drupal 相关信息、个人信息、工作信息、以及接收新闻通讯的偏好。

通过使用 `profile.module` 或者用户钩子的分类操作，你可以添加像这些分类一样的信息分类；参看 `profile.module` 中的实现。

外部登陆

Drupal 已经内置了对外部认证的支持，通过在一个模块中实现相应的钩子，就可以简单的将外部认证插入到 Drupal 中。Drupal 进行外部认证时所走的流程概貌如图 6-6 所示。

如果没有启用提供外部认证（就是说，实现 `auth` 钩子）的模块，Drupal 将把所有的用户名当作本地的用户名进行处理。所以 `joe` 和 `joe@example.com` 都被看做简单的字符串，而没有任何其它特殊含义。然而，如果启用了提供一个提供外部认证的模块，那么这两个用户名的处理流程就会非常不同了。

注意 Drupal 在尝试外部认证以前，总是首先尝试以本地用户进行登录。

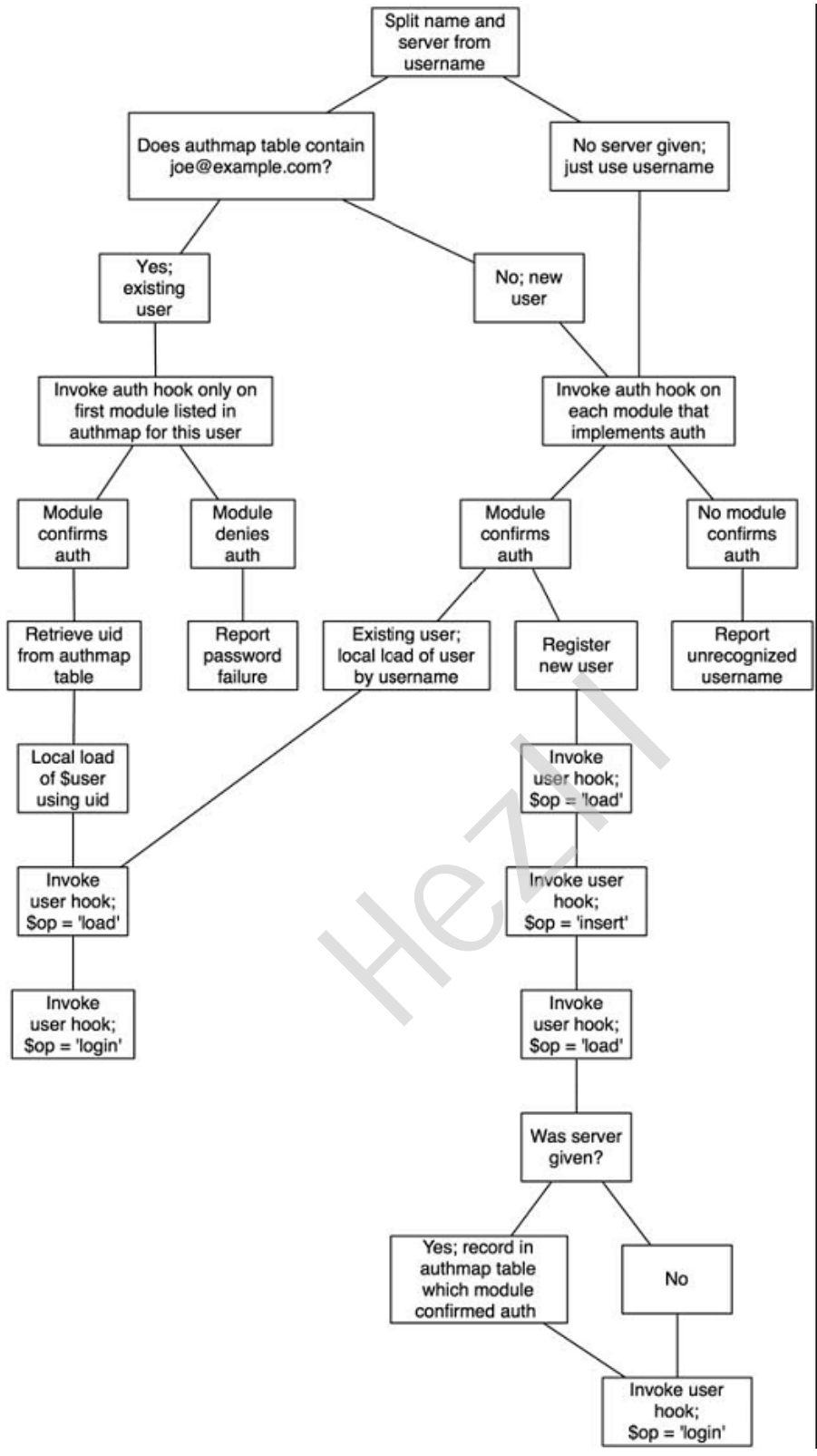


图 6-6 Drupal 的外部登陆流程

简单的外部认证

让我们实现一个非常简单的外部认证模块，一个使用简单用户名的公司使用这个模块。假定你的公司只雇用名为 **Dave** 的员工，而用户名基于第一个和最后一个名字（姓和名）指定。对于任何以 **dave** 开头的用户名都将通过该模块的认证，所以用户 **davebrown**, **davesmith**, 和 **davejones** 都将能够成功的登录。

```
<?php

// $Id$

/**

 * Implementation of hook_auth()

 */

function authdave_auth($username, $pass, $server) {

// Does username begin with 'dave'?

if (substr(drupal_strtolower($username, 0, 4 )) == 'dave') {

// Make a global variable to note that we did the authentication.

global $authdave_authenticated;

$authdave_authenticated = TRUE;

return TRUE;

}

else {

return FALSE;

}

}

}
```

如果一个用户在 **users** 表中不存在对应的记录，那么将会为其创建一个记录。然而，在登录过程中创建的用户没有为其提供 **e-mail** 地址，在 **Drupal** 默认的本地用户注册时则提供了 **e-mail** 地址，如果你的站点需要发送 **e-mail**，那么这样简单的模块就不是一个可行的解决方案。你需要设置 **users** 表的 **mail** 列，这样你就有了一个与用户相关联的 **e-mail** 地址。为了做到这一点，你可以实现用户钩子（**hook_user**），给出插入操作时的逻辑，这样当插入一个新的用户时就会调用相应的逻辑：

```
/**
```

```
* Implementation of hook_user()

*/

function authdave_user($op, &$edit, &$account, $category = NULL) {

switch($op) {

case 'insert':

// New user was just added; if we did authentication,

// look up email address of user in a legacy database.

global $authdave_authenticated;

if ($authdave_authenticated) {

$email = mycompany_email_lookup($account->name);

// Set email address in the user table for this user.

db_query("UPDATE {users} SET mail = '%s' WHERE uid = %d", $email,

$account->uid);

}

break;

}

}

}
```

聪明的读者将会注意到，如果同时启用了 **Drupal** 的本地认证和我们的外部认证，那么就没有一种方式来让插入操作下面的代码告诉我们，用户是通过本地认证的还是通过外部认证的；所以我们在这里使用全局变量以指示我们的模块进行了认证。

使用提供的服务器进行外部认证

当一个用户使用 joe@example.com 格式的用户名开始登录时，我们需要依照更多的信息进行处理。**Drupal** 内核包含了 `drupal.module`，它提供了一个 **XML-RPC** 客户端，可用于连接到其它服务器上请求认证。例如，在网站 <http://groups.drupal.org> 上，你可以使用你在 <http://drupal.org> 上的用户名和密码进行登录。下面是我第一次登录时所发生的事情：

- 1.我使用用户名 jvandyk@drupal.org 和我的密码在 groups.drupal.org 上登录。、

2. `groups.drupal.org` 检查本地用户数据库并且没有找到我。

3. `groups.drupal.org` 检查 `authmap`，也没能找到我。

4. 由于在 `groups.drupal.org` 上启用了 `drupal.module`，调用它的 `auth` 钩子。

5. `drupal.module` 向 <http://drupal.org> 发送一个 XML-RPC 请求，并且询问，“在你这里是不是有一个名为 `jvandyk` 并且使用这个密码的用户？”

6. `drupal.org` 回答道，“是的，它是一个热心的用户”。

7. `groups.drupal.org` 为我在 `users` 表中添加一条记录（包括一个本地用户 ID），在 `authmap` 表中也添加一条记录，这样当我下次登录时，只需要运行步骤 1 和步骤 3 就可以了。

当提供了一个服务器时，外部登陆的关键点在于 `authmap` 表。这个表包含了 3 个很重要的列：用户 ID，外部的用户名，处理认证的模块的名字。在前面的例子中，我的用户 ID 可能是 334，用户名是 jvandyk@drupal.org，模块列的值为 `drupal`，这是因为 `drupal module` 对我进行了认证，当我下次登录时仍然由它负责认证我。

注意 在这里，`drupal.org` 为 `groups.drupal.org` 对我进行了认证。但是 `drupal.org` 没有将我的 e-mail 地址提供给 `groups.drupal.org`。和我们本节中的简单外部认证例子一样，如果 <http://groups.drupal.org> 的维护者认为 `users` 表的 `mail` 列都有值的话，那么这将是一个非常愚蠢的想法。这里使用了一个随机生成的密码作为 `password` 列的值。

下面是 `drupal.module` 中 `auth` 钩子实现的简化版本，用来说明前面场景所用到的代码：

```
/**
 * Implementation of hook_auth().
 */
function drupal_auth($username, $password, $server = FALSE) {
  if (!empty($server)) {
    // Ask remote server to attempt login for this username and password.
    $result = xmlrpc("http://$server/xmlrpc.php", 'drupal.login', $username,
    $password);
    if ($result === FALSE) { // Authentication failed.
      drupal_set_message(t('Error %code: %message', array(
        '%code' => xmlrpc_errno(),
```

```
'%message' => xmlrpc_error_msg()), 'error');

return FALSE;

}

else {

return $result;

}

}

}

}
```

在认证服务器上（在前面的例子中就是 <http://drupal.org>），为了响应服务器端 `drupal_auth()` 发出的 XML-RPC 请求，运行以下代码：

```
/**
 * Callback function from drupal_xmlrpc() for authenticating remote clients.
 *
 * Remote clients are usually other Drupal instances.
 */

function drupal_login($username, $password) {

if (variable_get('drupal_authentication_service', 0)) {

if ($user = user_load(array(

'name' => $username,

'pass' => $password,

'status' => 1))) {

// Found an unblocked user so return user ID.

return $user->uid;
```

```
}  
else {  
return 0;  
}  
}  
}
```

info 钩子

如果你的模块实现了外部认证（也就是说，使用了 **auth** 钩子），那么你也应该实现 **info** 钩子。这个钩子提供了你模块的名字以及它的验证方法，当其它模块想知道有哪些验证方法可用时使用这一信息。例如，在 **user.module** 就用它构建用户登录页面所支持的认证方法列表：

```
/**  
 * Implementation of hook_info().  
 */  
function drupal_info($field = 0) {  
  $info['name'] = 'Drupal';  
  $info['protocol'] = 'XML-RPC';  
  if ($field) {  
    return $info[$field];  
  }  
  else {  
    return $info;  
  }  
}
```

总结

读完本章后，你应该能够

- 理解用户在 **Drupal** 内部是如何表示的
- 理解如何使用不同的方式来存储与用户相关的信息
- 使用用户注册过程中的钩子，来获取一个正在注册的用户的信息。
- 使用用户登录过程中的钩子，在用户登录时运行你自己的代码
- 理解两种不同的外部认证方式的工作原理
- 实现你自己的外部认证模块

注意 更多关于外部认证的信息，参看 `ldap_integration.module`, `pubcookie.module`, 和 `sxip.module`。

HEZI

第 7 章 Drupal 节点 (Drupal node) (1)

Drupal 节点

在本章中，我们将介绍节点和节点类型。我们将向你展示创建一个节点类型的两种不同方式。首先，我们将向你展示程序解决方案，使用 **Drupal** 钩子函数编写模块来创建节点类型。在定义节点可以做什么不可以做什么的时候，该方式具有更高的自由度和灵活性。接着，我们将向你展示如何通过 **Drupal** 后台管理接口来创建一个节点类型，并简单的讨论了内容创建工具模块 (**CCK**)，**Drupal** 社区正在慢慢的将 **CCK** 的方式添加到 **Drupal** 核心中去。最后我们将研究一下 **Drupal** 的节点访问控制机制。

那么什么才是一个节点呢？

对于刚刚接触 **Drupal** 开发的新手来说，最先遇到的问题之一就是，什么是节点？一个节点是一内容片段。**Drupal** 为每一片内容指定一个名为“节点 ID”（在代码中接卸为 **\$nid**）的数字 ID。一个每个节点还拥有一个标题，从而允许管理员通过标题查看的内容。

注意：如果你熟悉面向对象的话，那么你可以把每个节点类型看做一个对象，把每个节点看做一个对象实例。然而，**Drupal** 的代码不是 100%面向对象的，为什么这样呢？下面是一个很好的解释。（参看

<http://api.drupal.org/api/HEAD/file/developer/topics/oop.html>）。

有许多不同的节点或节点类型。常见的节点类型有“**blog entry**”，“**poll**”，和“**book page**”。一般情况下（在本书中），术语“内容类型”和“节点类型”是同义的，尽管节点类型是一个更抽象的概念并且你可以把它看做容器，如图 7-1 所展示的。

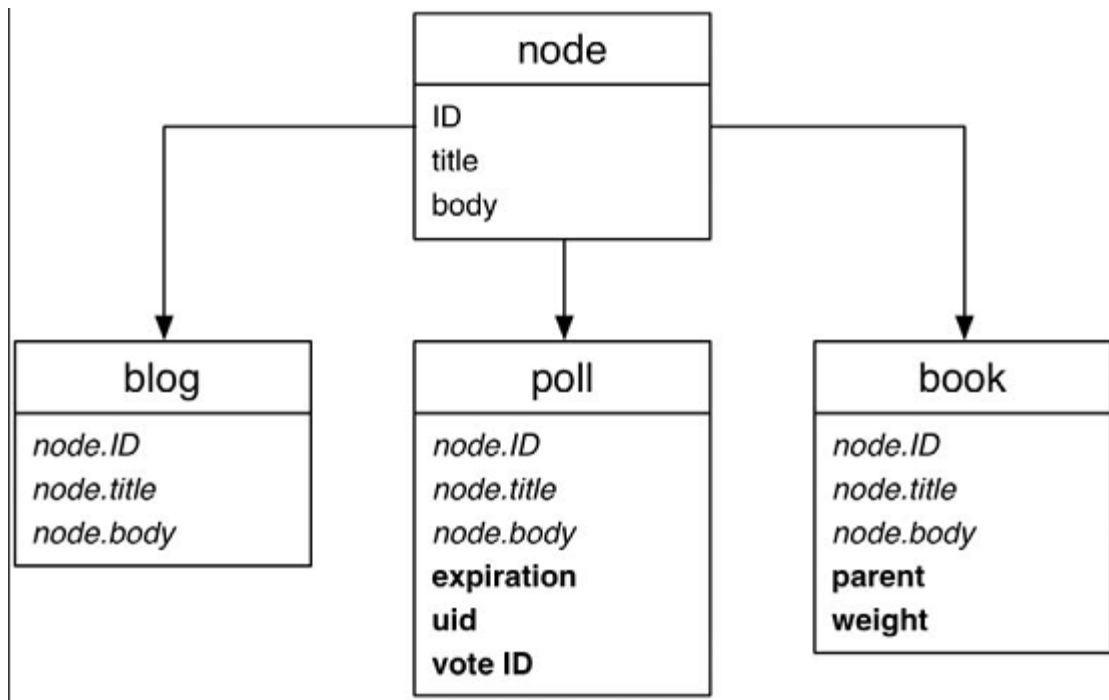


图 7-1 基于基本节点的节点类型和可能添加的字段

把所有内容类型当作节点的好处是，这样它们就可以基于相同的底层数据结构了。对于开发者来说，这意味着你可以对所有的内容以同样的代码方式进行许多操作。可以在节点上非常容易的进行一组操作，并且你也可以为你自己的节点类型添加许多额外的功能。由于所有的内容都是节点，所以 **Drupal** 内置的支持了对内容的搜索、创建、编辑和管理等操作。该一致性对于终端用户也同样明显。由于创建、编辑和删除节点的表单拥有一个类似的外观，这样就保持了一致性和更好的接口易用性。

节点类型通常通过为它们添加自己的属性来扩展基本节点。节点类型 **poll** 存储了投票相关条目，如投票的过期日期和投票人。节点类型 **book** 为每个节点存储了父节点 **ID**，这样它就可以知道自己在表 **book** 的记录中的位置。节点类型 **blog**，则与前面二者不同，它没有添加任何的其它的数据；替代的，通过为每个用户创建日志和为每个日志创建 **RSS** 种子，从而为数据添加了不同的试图。所有的节点都包括存储在表 **node** 和 **node_revisions** 中的下列属性：

- **nid**: 节点的唯一标识 **ID**。
- **vid**: 节点的唯一修订本 **ID**，由于 **Drupal** 需要为每个节点存储可修订的内容，所以该字段是必须的。在所有的节点和节点修订本中，**vid** 是唯一的。
- **type**: 每一个节点都有一个节点类型；例如，**blog**, **story**, **article**, **image** 等等。
- **title**: 节点的标题，短的 **128** 位字符的字符串。如果通过代码将表 **node_type** 中的字段 **has_title** 设置为 **0** 的话，那么就没有标题了。

-
- **uid:** 作者的用户 ID。默认情况，每个节点都有一个唯一的作者。
 - **status:** 0 表示未发布；就是说，不具有权限“administer nodes”的用户看不到内容。1 意味着已发布，并且内容对于具有“access content”权限的用户是可见的。Drupal 的节点级别的访问控制机制（参看 `hook_access()`）可以禁止已发布节点的展示，如果启用了搜索模块，那么可以使用搜索模块来对内容建立索引。
 - **created:** 节点创建时的 Unix 时间戳。
 - **changed:** 节点最后被修改的 Unix 时间戳。如果你是用了节点修订系统，那么它的值与表 `node_revisions` 中字段 `timestamp` 的值相同。
 - **comment:** 一个整数字段用以描述节点的评论状态，拥有 3 个可能值：
 - 0: 对当前节点禁用了评论。这是评论模块启用时的默认值。
 - 1: 不能再向当前节点添加评论了。
 - 2: 可以查看评论，并且用户可以创建新的评论。评论模块负责控制着谁可以创建评论以及评论展示的外观。
 - **promote:** 另一个整数字段，用来决定是否将节点发布到首页上，有两个值可用：
 - 1: 发布节点的同时将节点展示到你站点的首页。这里需要注意的是，由于你可以将首页改为你想要的那个页面，这里可能有点用词不当。更准确一点的说，页面 <http://example.com/?q=node> 将包含所有的字段 `promote` 为 1 的节点，而该页面在默认情况下为站点的首页。
 - 0: 不将节点展示在首页 <http://example.com/?q=node>。
 - **moderate:** 一个整数字段，其中 0 表示仅用了 `moderation`，1 表示启用了 `moderation`。下面是该字段的警告说明。在核心 Drupal 安装中没有为该字段留下接口。换句话说就是，你可以反复的改变该字段的值，而它默认情况下不起任何作用。所以开发者不要使用该字段来开发任何功能。在以前的 Drupal 版本中，该字段在核心代码中起到很大作用。
 - **sticky:** 当 Drupal 在一个页面展示一系列节点时，默认情况是将标记为 `sticky` 的节点列在前面，接着按照创建日期列出剩下的“`unsticky`”节点。换句话说就是“`sticky`”节点位于节点列表的顶部。1 表示 `sticky`，0 表示 `unsticky`。你可以在同一列表中包含多个 `sticky` 节点。

如果你是用了 Drupal 的修订系统，Drupal 将创建一个内容的修订本来追踪谁最后修改了节点。

不是所有的东西都是节点

用户、区块和评论不是节点。这些特定的数据结构中的每一个都拥有它自己的钩子系统以适应它的特定目的。节点一般有“标题”和“主体”两部分，而代表用户的数据结构中则不需要这

些。用户需要的是，**e-mail** 地址、用户名称、一种安全的存储密码的方式。当要存储的内容片段更小一些时，比如存的时导航菜单、搜索框、罪行评论列表等等，我们此时使用轻量级的存储解决方案---区块。评论也不使用节点来保存它们轻量级的内容。一个页面可能会有 **100** 或者更多的评论，试想，如果所有的这些评论都使用节点钩子系统的话，那么会给系统带来多大的负担呢。

在过去，经常争论，用户或评论到底应不应该归结为节点。如果现在再提起这样的问题的话，就好比在编程风格上高呼“**Emacs** 更好”一样。（译者注：我不知道 **Emacs** 什么意思 ^_^）。

创建一个节点模块

传统上，当你在 **Drupal** 中创建一个新的内容类型时，你应该编写一个节点模块来负责提供你的内容类型所需的新的且有趣的东西。我们说这是传统方式，这是因为 **Drupal** 框架最近常用的方式是允许你通过后台管理接口来创建内容类型，使用第 **3** 方模块来扩展这些内容类型的功能，而不是从头开始编写一个节点模块。在本章中，我们将讨论这两种方式。

让我们编写一个节点模块，从而让用户可以为站点添加笑话。每一个笑话都包括一个标题，笑话本身，接着是一个笑话妙语（**punchline**）。你应该可以非常容易的使用内置的节点属性 **title** 来存储笑话的标题，用节点属性 **body** 来存储笑话内容，但是你还需要创建一个新的数据库表来存储笑话妙语。下面是数据库元数据：

```
CREATE TABLE joke (  
  
nid int unsigned NOT NULL default '0',  
  
vid int unsigned NOT NULL default '0',  
  
punchline text NOT NULL,  
  
PRIMARY KEY (nid,vid),  
  
UNIQUE KEY vid (vid),  
  
KEY nid (nid)  
);
```

你存储了节点的 **ID**，这样你就可以引用保存了标题和主体的 **node_revisions** 表中的对应节点了。这里使用了列 **vid**，这样你就可以使用 **Drupal** 内置的节点修订控制系统了。当你编写对数据库进行更新操作的代码时，你就会看到如何使用它了。

首先，让我们在目录 **sites/all/modules/custom** 下面创建名为 **joke** 的文件夹。

创建 **.install** 文件

由于你已经知道了数据库的元数据，让我们继续，创建 `joke.install` 文件并将其放到目录 `sites/all/modules/custom/joke` 下面。参看第 2 章以获取创建安装文件的更多信息。

```
<?php
// $Id$

/**
 * Implementation of hook_install().
 */

function joke_install() {
  switch ($GLOBALS['db_type']) {
    case 'mysql':
    case 'mysqli':
      db_query("CREATE TABLE {joke} (
        nid int unsigned NOT NULL default '0',
        vid int unsigned NOT NULL default '0',
        punchline text NOT NULL,
        PRIMARY KEY (nid,vid),
        UNIQUE KEY vid (vid),
        KEY nid (nid)
      ) /*!40100 DEFAULT CHARACTER SET UTF8 */");
      break;
    case 'pgsql':
      db_query("CREATE TABLE {joke} (
        nid int unsigned NOT NULL default '0',
        vid int unsigned NOT NULL default '0',
        punchline text NOT NULL,
        PRIMARY KEY (nid,vid),
```

```
UNIQUE KEY vid (vid),  
KEY nid (nid)  
);  
break;  
}  
}  
/**  
 * Implementation of hook_uninstall().  
 */  
function joke_uninstall() {  
db_query('DROP TABLE {joke}');  
}
```

创建.info 文件

让我们在创建一个 **joke.info** 文件并将其添加到 **joke** 文件夹下。

```
; $Id$  
name = Joke  
description = Provides a joke node type with a punchline.  
version = "$Name$"
```

创建.module 文件

最后，你需要创建模块文件本身。创建一个名为 **joke.module** 的文件并将其放到 **sites/all/modules/custom/joke** 下面。接着，你可以在模块列表页面（**Administer > Site building > Modules**）启用这一模块。你首先添加如下代码：

```
<?php
```

// \$Id\$

/**

* @file

* Provides a "joke" node type.

*/

Hezi

第 7 章 Drupal 节点 (Drupal node) (2)

注意：在前面列表中提到的内部名称字段，是用来构造“创建内容(create content)”链接的 URL 的。例如，我们使用“joke”作为我们节点类型的内部名称（它是我们返回的数组的键），那么要创建一个新的笑话的话，用户要访问页面

<http://example.com/?q=node/add/joke>。通常你不需要对此作出修改。内部名称存储在表 `node` 和 `node_revisions` 的“type”列中。

定义一个菜单回调函数

现在你已经定义了基本的节点属性，让我们为路径 `node/add/joke` 创建一个菜单回调函数，这样你就可以创建一个笑话表单并定义一些权限了。

```
/**
 * Implementation of hook_menu().
 */
function joke_menu($may_cache) {
  $items = array();

  // Do not cache this menu item during the development of this module.
  if (!$may_cache) {
    $items[] = array(
      'path' => 'node/add/joke',
      'title' => t('Joke'),
      'access' => user_access('create joke'),
    );
  }

  return $items;
}
```

由于菜单系统的层次性，在这里“callback”参数是可选的。你可以不对其进行设置，替代的你使用父路径 `node/add` 的回调函数，可以在 `node.module` 中的菜单钩子中找到该路

径。在那里，它映射到了构建节点表单的函数 `node_add()` 上。通过使用已经定义好的回调函数，由于已经将它映射到了核心的节点验证和提交程序上了，所以你可以节省大量的工作。换句话说就是，现在你只需要关注于你自己定制的数据就可以了（在这里，你的模块将需要处理笑话妙语（`punchline`）字段），而其他的用户提交的数据比如标题和主体都不用你操心。这是由于 `node.module` 将会负责标题和主体以及其它一些核心节点属性的创建、验证、处理等操作。

使用 `hook_perm()` 定义特定于节点类型的权限

在你的菜单项目中你也添加了对“create joke”权限的检查，但是你尚未在模块中定义该权限。让我们使用 `hook_perm()` 来创建该权限：

```
/**
 * Implementation of hook_perm().
 */
function joke_perm() {
  return array('create joke', 'edit own joke');
}
```

现在你可以导航到 **Administer > User management > Access control**，你就可以看到你上面定义的权限了，并且可以将它分配给用户角色了。

使用 `hook_access()` 来限制对一个节点类型的访问

节点模块可以使用 `hook_access()` 来限制对它们定义的节点类型的访问。超级用户（用户 ID 1）将绕过所有的访问权限检查，所以此时不会调用该钩子函数。如果没有为你的节点类型定义该钩子函数，那么所有的访问权限检查都会失败，这样就只有超级用户和具有“`administer nodes`”权限的用户才能够访问该类型的内容。

```
/**
 * Implementation of hook_access().
 */
function joke_access($op, $node) {
  global $user;

  if ($op == 'create') {
```

```

return (user_access('create joke'));

}

if ($op == 'update' || $op == 'delete') {

return (user_access('edit own joke') && ($user->uid == $node->uid));

}

}

```

前面的函数允许具有“create joke”权限的用户创建一个笑话节点。如果用户还具有“edit own joke”权限并且它们是节点作者的话，那么他们还可以更新或者删除一个笑话。传递到钩子函数 `hook_access()` 的 `$op` 中的另一个值是“view”（查看），它允许你控制谁可以查看该节点。然而我们需要提醒一下：当查看的页面仅有一个节点时，才调用钩子 `hook_access()`。当用户查看的页面是多节点列表页面，节点此时处于 `teaser` 状态，那么这种情况下就不能使用 `hook_access()` 来阻止用户对该页面的访问。你可以创建一些其它的钩子函数并直接操纵 `$node->teaser` 的值来控制对它的访问，但是这有点麻烦。一个好的解决方案是使用我们将在后面简短讨论到的函数 `hook_node_grants()` 和 `hook_db_rewrite_sql()`。

为我们的节点类型定制节点表单

到目前为止，你已经为你的新节点类型定义了元数据、菜单回调函数和访问控制权限。接着，你需要构建节点表单，这样用户就可以输入笑话了。你可以通过实现 `hook_form()` 来完成这一工作：

```

/**
 * Implementation of hook_form().
 */

function joke_form($node) {

// Get metadata for this node type

// (we use it for labeling title and body fields).

// We defined this in joke_node_info().

$type = node_get_types('type', $node);

$form['title'] = array(

```

```
'#type' => 'textfield',

'#title' => check_plain($type->title_label),

'#required' => TRUE,

'#default_value' => $node->title,

'#weight' => -5

);

$form['body_filter']['body'] = array(

'#type' => 'textarea',

'#title' => check_plain($type->body_label),

'#default_value' => $node->body,

'#rows' => 7,

'#required' => TRUE

);

$form['body_filter']['filter'] = filter_form($node->format);

$form['punchline'] = array(

'#type' => 'textfield',

'#title' => t('Punchline'),

'#required' => TRUE,

'#default_value' => $node->punchline,

'#weight' => 5

);

return $form;

}
```

注意：如果你不熟悉表单 API 的话，请参看第 10 章。

作为站点管理员，你现在可以导航到 **Create content > Joke** 并查看新创建的表单了。在前面函数中的第一行代码返回了该节点类型的元数据信息。`node_get_types()`将检查 `$node->type` 以决定要返回哪种节点类型的元数据（在我们的例子中，`$node->type` 的值将为“joke”）。这里再强调一遍，在钩子 `hook_node_info()`中设置节点元数据，你已经在前面的 `joke_node_info()`中设置了它。函数的其余部分包含了三个表单字段，用来收集标题、主题、笑话妙语。这里有一个重点，就是如何实现标题和主体的`#title` 键的动态化的。它们的值来源于 `hook_node_info()`，如果在 `hook_node_info()`中“locked”属性为 `FALSE` 的话，站点管理员也可以在 <http://example.com/?q=admin/content/types/joke> 修改这些值。

添加过滤器格式支持

由于主体字段是一个 `textarea`，而且节点主体字段可以使用过滤器格式，那么你可以使用下面的代码来包含 **Drupal** 的标准过滤器选择器（使用过滤器的更多信息,参看第 11 章）：

```
$form['body_filter']['filter'] = filter_form($node->format);
```

如果你想让笑话妙语字段也可以使用过滤器格式，你需要在你的数据库表“joke”中再添加一列以为每个笑话妙语存储输入过滤器格式，如下所示：

```
ALTER TABLE 'joke' ADD 'punchline_format' INT UNSIGNED NOT NULL;
```

接着你要将你的最后一个表单字段定义修改成如下所示的形式：

```
$form['punchline']['field'] = array(  
  
  '#type' => 'textarea',  
  
  '#title' => t('Punchline'),  
  
  '#required' => TRUE,  
  
  '#default_value' => $node->punchline,  
  
  '#weight' => 5  
);
```

```
$form['punchline']['filter'] = filter_form($node->punchline_format);
```

一般情况下，在 **Drupal** 中构建表单时的最后一行代码应该是如下的格式：

```
return drupal_get_form('joke_node_form', $node);
```

然而，由于你使用的是一个节点表单而不是一个普通的表单，**node.module** 将处理大部分额外的工作。它负责验证和存储在表单中它能识别的所有默认字段，并为你（开发者）提供了验证和存储你定制数据的钩子。

接下来我们将讨论这些钩子函数。

使用 `hook_validate()` 来验证字段

当你提交一个你的节点类型的节点时，将会调用你模块中的钩子 `hook_validate()` 以验证你定制字段中的输入数据了。在验证中，你也可以使用 `form_set_value()` 做些修改。可以使用 `form_set_error()` 来设置错误消息，如下所示：

```
/**
 * Implementation of hook_validate().
 */
function joke_validate($node) {
  // Enforce a minimum word length of 3.
  if (isset($node->punchline) && str_word_count($node->punchline) <= 3) {
    $type = node_get_types('type', $node);
    form_set_error('punchline', t('The punchline of your @type is too short. You
    need at least three words.', array('@type' => $type->name)));
  }
}
```

注意你已经在 `hook_node_info()` 中为主体字段定义了最小单词数目，而 **Drupal** 将自动对此进行验证。然而，笑话妙语字段是你添加到该节点类型表单的一个定制字段，所以你需要负责它的验证（加载、保存）。

使用 `hook_insert()` 来存储我们的数据

当保存一个新的节点时将会调用钩子 `insert()`，在该钩子中你可以将定制的数据存储到相关的表中。只有节点为当前节点类型时才调用这一钩子。例如，如果节点类型是“joke”，那么将会调用 `joke_insert()`。如果新添加了一个“book”类型的节点，那么就不会调用 `joke_insert()`（在这里将调用 `book_insert()`）。

注意：如果你在插入一个不同类型的节点时对其做些操作的话，你需要把它当作一个普通的节点提交，使用钩子 `hook_nodeapi()` 插入一些操作。参看“使用 `hook_nodeapi()` 操纵其它类型的节点”。

下面是为 `joke.module` 编写的 `hook_insert()` 函数：

```
/**
 * Implementation of hook_insert().
 */
function joke_insert($node) {
  db_query("INSERT INTO {joke} (nid, vid, punchline) VALUES (%d, %d, '%s')",
    $node->nid, $node->vid, $node->punchline);
}
```

使用 `hook_update()` 更新定制数据

当编辑完一个节点并且核心节点数据已被写入到数据库中时，将调用钩子 `update()`。再改钩子中编写对相关表的更新操作代码。而钩子 `hook_insert()`，只有在节点为当前节点类型时才调用。例如，如果节点类型为“joke”，那么将调用 `joke_update()`。

```
/**
 * Implementation of hook_update().
 */
function joke_update($node) {
```

```
if ($node->revision) {  
  
joke_insert($node);  
  
}  
  
else {  
  
db_query("UPDATE {joke} SET punchline = '%s' WHERE vid = %d",  
  
$node->punchline, $node->vid);  
  
}  
  
}
```

在这里，你首先检查是否设置了节点修订标记，如果已经设置了，你创建一个笑话妙语的副本以保留旧的版本。（译者注：这句是对 `joke_insert($node);` 的解释，我没有看懂）。

使用 `hook_delete()` 清理数据

在从数据库中山出一个节点之后，**Drupal** 将会立即调用所有实现了钩子 `hook_delete()` 的模块中的该钩子方法。该钩子一般用来从数据库中删除相关的信息。只有在删除当前节点类型的节点时，才调用本钩子。如果节点类型为“joke”，将会调用 `joke_delete()`。

```
/**  
  
 * Implementation of hook_delete().  
  
 */  
  
function joke_delete(&$node) {  
  
 // Delete the related information we were saving for this node.  
  
 db_query('DELETE FROM {joke} WHERE nid = %d', $node->nid);  
  
}
```

注意：当要删除的是一个修订本而不是整个节点时，**Drupal** 将调用钩子 `hook_nodeapi()`，其中将 `$op` 设为“delete revision”并将整个节点对象传进来。接着你的模块可以使用 `$node->vid` 作为键来删除它在修订本中的数据。

使用 `hook_load()` 来修改节点对象

在你的基本的 `joke` 模块中你最后一个需要实现的节点相关的钩子就是 `hook_load()`，它可以在构建节点对象时向对象中添加你定制的节点属性。我们需要把笑话妙语字段注入到节点加载流程中，这样就可以在其他模块以及主题层使用它了。在构建完核心节点对象以后，且节点为当前节点类型时，将会立即调用该钩子。如果节点类型为“`joke`”，那么就调用 `joke_load()`。

```
/**
 * Implementation of hook_load().
 */
function joke_load($node) {
  return db_fetch_object(db_query('SELECT punchline FROM {joke} WHERE vid
  = %d',
  $node->vid));
}
```

使用 `hook_view()` 显示笑话妙语

现在你有了一个完整的系统，可以输入和编辑笑话。然而，尽管可以在节点提交表单中输入笑话妙语，但在查看笑话时，你却没有提供显示笑话妙语（包袱）字段的方式，你的用户将会对此感到很困惑。让我们使用 `hook_view()` 来显示笑话妙语（包袱）：

```
/**
 * Implementation of hook_view().
 */
function joke_view($node, $teaser = FALSE, $page = FALSE) {
  if (!$teaser) {
    // Use Drupal's default node view.
    $node = node_prepare($node, $teaser);
    // Add a random number of Ha's to simulate a laugh track.
    $node->guffaw = str_repeat(t('Ha!'), mt_rand(0, 10));
    // Now add the punchline.
```

```
$node->content['punchline'] = array(
  '#value' => theme('joke_punchline', $node),
  '#weight' => 2
);
}

if ($teaser) {
  // Use Drupal's default node view.

  $node = node_prepare($node, $teaser);
}

return $node;
}
```

你首先需要保证节点不是 **teaser** 形式显示的（也就是说，**\$teaser** 应该为 **FALSE**，这样你就可以继续了）。你已将笑话秒的显示分解为一个单独的主题函数，这样就可以非常容易的覆写它了。如果你个站点管理员想使用你的模块但又想定制外观的话，这将会非常方便。你为主题函数提供了一个默认实现：

```
function theme_joke_punchline($node) {
  $output = '<div class="joke-punchline">'.
  check_plain($node->punchline). '</div><br />';
  $output .= '<div class="joke-guffaw">'.
  check_plain($node->guffaw). '</div>';
  return $output;
}
```

你现在应该有了一个可以完全工作的笑话输入和查看系统。继续前进，输入一些笑话测试一下。你现在应该可以看到你的笑话了，它看起来有点朴素和简单，如图 7-2 所示：



图 7-2 简单主题下的笑话节点

尽管这也可以工作，但还是存在一个更好的方式让用户立即看到笑话妙语。我们想要的是使用一个可伸缩的字段集，当用户点击时再展示笑话妙语。在 **Drupal** 内部已经存在了一个可伸缩的字段集功能，所以你只需要使用现有的就可以了而不是创建你自己的 **Javascript** 文件。把这一交互放到你的站点主题的模板文件中比放到主题函数中更好一些，因为它依赖于标识字体和 **CSS** 类。你的设计者喜欢你这样做，因为如果要修改笑话节点的外观的话，只需要简单的编辑模板文件就可以了。你需要创建一个名为 **node-joke.tpl.php** 的模板文件，并将其放到你当前使用的主题的目录下面，下面是该文件中的内容。如果你使用的主题为 **bluemarine**，那么 **node-joke.tpl.php** 将被放到 **themes/bluemarine** 下面。由于我们将使用一个模板文件，那么就不再需要或者调用函数 **theme_joke_punchline()** 了，所以我们就可以把该函数注释掉了。

注意：主题系统将会自动发现 **node-joke.tpl.php**，**Drupal** 将使用该模板文件来修改笑话的外观，而不是默认的节点模板文件 **node.tpl.php**。更多关于主题系统的知识，请参看第 8 章。

```
<div class="node<?php if ($sticky) { print " sticky"; } ?>
<?php if (!$status) { print " node-unpublished"; } ?>">
<?php if ($picture) {
print $picture;
}?>
<?php if ($page == 0) { ?><h2 class="title"><a href="<?php
print $node_url?>"><?php print $title?></a></h2><?php }; ?>
<span class="submitted"><?php print $submitted?></span>
<span class="taxonomy"><?php print $terms?></span>
<div class="content">
<?php print $content?>
```

```

<fieldset class="collapsible collapsed">

<legend>Punchline</legend>

<div class="form-item">

<label><?php print check_markup($node->punchline)?></label>

<label><?php print $node->guffaw?></label>

</div>

</legend>

</fieldset>

</div>

<?php if ($links) { ?><div class="links">&raquo; <?php print $links?></div>

<?php }; ?>

</div>

```

最后，你需要加载负责可伸缩字段集的 JavaScript 文件。这需对 `joke_load()` 做一简单修改：

```

function joke_load($node) {

drupal_add_js('misc/collapse.js');

return db_fetch_object(db_query('SELECT * FROM {joke} WHERE vid = %d',

$node->vid));

}

```

如果用户查看的是节点列表页面，那么每个页面将会多次调用 `drupal_add_js()`，但是函数 `drupal_add_js()` 实际上会阻止重复加载文件。所以在这里加载 JavaScript 文件能够保证只有当需要它时（页面中包含笑话节点）才加载它，而不是将它放到菜单钩子里面对于所有页面都加载它。`collapsible.js` 中的 JavaScript 将为字段集寻找可伸缩的 CSS 选择器，并且在找到以后知道如何处理它，如图 7-3 所示。这样，在 `node-joke.tpl.php` 中它将找到下面代码并激活它：

```

<fieldset class="collapsible collapsed">

```

这将得到我们想要的笑话的交互体验：



图 7-3 使用 Drupal 内置的可伸缩 CSS 支持来隐藏笑话妙语

使用 hook_nodeapi () 操纵其它类型的节点

前面的钩子只有在基于节点类型时才调用。当 Drupal 查看一个 **blog** (日志) 节点时, 调用 **blog_load()**。如果你想为每个节点都添加一些信息, 不管它是什么类型, 那该怎么办呢? 我们到目前为止看到的钩子函数都做不到这一点; 对于这一工作我们需要一个更强大的钩子: **hook_nodeapi()**。

尽管创建节点不需要该钩子, 但是还是值得在此对它进行说明, 因为对于任何节点的生命周期期间, 该钩子提供了一个使用模块对其进行修改的机会。一般在 **node.module** 调用完所有的特定节点类型的钩子函数以后调用钩子 **nodeapi()**。下面是函数的签名:

```
hook_nodeapi(&$node, $op, $a3 = NULL, $a4 = NULL)
```

因为节点对象 **\$node** 是通过引用传递的, 所以对它的任何修改都将改变真正的节点。参数 **\$op** 用来描述当前对节点所进行的操作, 它可以有多种不同的值:

- **delete**: 删除节点时。
- **insert**: 刚刚将新节点插入到数据库中。
- **load**: 从数据库中加载了基本的节点对象, 再加上有节点类型设置的额外节点属性 (将会运行负责该项工作的 **hook_load()** 函数; 参看本章前面的“使用 **hook_load()** 来修改节点对象”)。此时你可以添加新的属性或者操纵已有的节点属性。
- **view**: 正准备将节点展示给用户。将在钩子 **hook_view()** 之后调用该动作, 所以模块可以假定节点已被过滤并且现在包含的为 **HTML**。
- **update**: 节点刚被更新到数据库中。
- **validate**: 用户刚完成对节点的编辑并试图预览或者查看它。你可以使用该钩子来检查甚至修改数据, 尽管在验证钩子中修改数据被认为是坏的开发实践。
- **submit**: 节点已通过了验证并将被保存到数据库中。

- **prepare:** 将要展示节点表单。它应用于节点的“Add”（添加）和“Edit”（编辑）表单时。
- **print:** 为打印准备一个节点视图。用在 `book.module` 中的打印机专用视图中。
- **search result:** 节点将作为一个项目展示在搜索结果中时。
- **update index:** 节点正在被搜索模块索引化。如果你想对使用 `nodeapi` 的“view”操作不能显示的额外信息进行索引的话，你可以在这里完成它（参看第 12 章）。
- **rss item:** 节点将被作为 RSS 种子的一部分包含进来。

最后两个参数是根据当前操作可以改变其值的变量。当展示一个节点并且 `$op` 为 `view` 时，`$a3` 将为 `$teaser`，而 `$a4` 将为 `$page`（参看 `node.module` 中的 `node_view()`）。参数的总结可参看表 7-1。

表 7-1. 当 `$op` 为 `view` 时，`hook_nodeapi()` 中参数 `$a3` 和 `$a4` 的含义

参数	含义
<code>\$teaser</code>	是否要仅仅展示 <code>teaser</code> ，比如在主页上。
<code>\$page</code>	如果一个页面仅有一个节点时， <code>\$page</code> 为 <code>TRUE</code>

当节点正被验证时，`$a3` 参数为节点对象，而 `$a4` 参数为 `node_validate()` 中的 `$form` 参数（也就是，表单定义）。

如何存储节点

节点在数据库中被分成 3 个部分。表 `node` 包含了描述节点的大部分元数据。表 `node_revisions` 包含了节点的主体和 `teaser`，以及一些修订特有的信息。正如你在 `joke.module` 例子中看到的，其他节点类型可以自由的在加载节点时向节点添加数据，同时可以向它们自己的表中存储他们想要的数据库。

图 7-4 展示了一个包含了大部分常用属性的节点对象。注意你创建的用以存储笑话妙语的表被用来填充节点。根据启用模块的不同，在你的 `Drupal` 安装里面的节点对象包含的属性可能会有或多或少的变化。

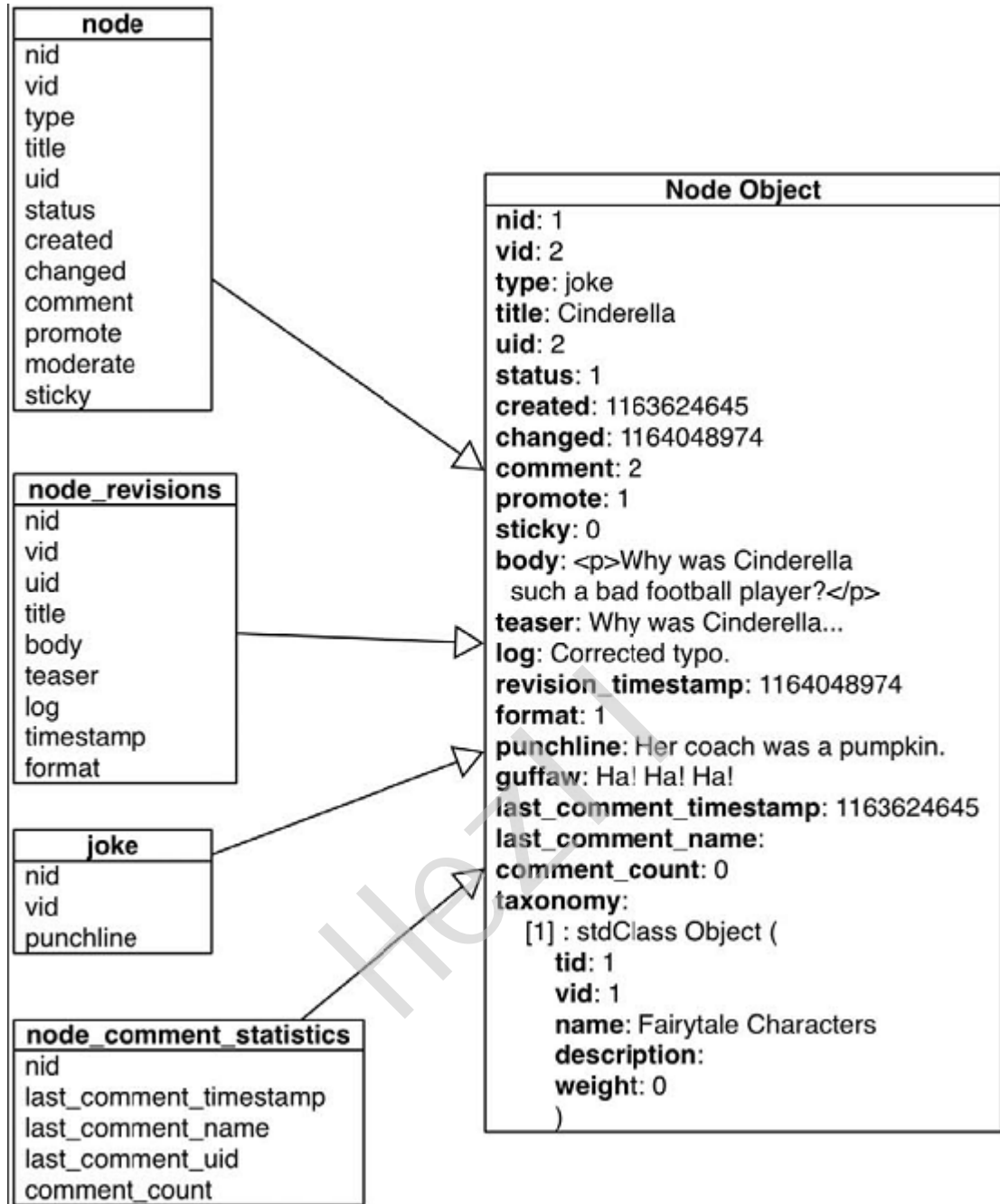


图 7-4 节点对象

第 7 章 Drupal 节点 (Drupal node) (3)

使用 CCK 创建节点类型

使用前面你在 `joke.module` 中所用的方式创建一个节点类型，尽管可以完全的进行控制并具有较高的性能，但是这种方式有点枯燥无味。如果不做任何编程工作就可以组装一个新的节点类型的话，难道这种方式不会更好么？这就是 **CCK** 提供的方式。

■ 注意：更多关于 **CCK** 的信息，访问 **CCK** 工程地址

<http://drupal.org/project/cck>。

CCK 中的部分已被发布到 **Drupal 5** 中。你现在可以导航到 **Administer > Content management > Content types**，通过后台管理接口页面添加一个新的内容类型（比如一个笑话 (`joke`) 内容类型）。如果你已经启用了 `joke.module` 的话，你要为节点类型起一个不同的名字以避免命名冲突。**CCK** 中的其它部分尚未被加入到 **Drupal** 核心中，比如为新节点类型添加除标题和主体以外的其它字段的能力。在 `joke.module` 例子中，你需要三个字段：标题，笑话本身，和笑话妙语（包袱）。你使用 **Drupal** 的 `hook_node_info()` 将主体 (`body`) 字段改名为笑话 (`Joke`)；通过实现一些钩子函数，并为笑话妙语的存储创建你自己的数据库表，从而提供了笑话妙语 (`punchline`) 字段。在 **CCK** 中，你可以简单的创建一个名为 `punchline` 的文本输入字段，并将其添加到你的内容类型中。**CCK** 替你负责数据的存储、取回、和删除。

注意：**Drupal** 第 3 方模块库中包含了大量的 **CCK** 字段模块，用来添加图像、日期、`e-mail`、地址等等字段。**CCK** 相关第 3 方模块位于：

<http://drupal.org/project/Modules/category/88>。

由于在编写本书时，**CCK** 的许多地方还在开发和完善中，所以我们在这里不讨论更多细节了。然而，可以清晰的看到，在将来使用编写模块的方式创建一个新的节点类型会越来越来少，而使用 **CCK** 的方式通过管理接口页面组装一个新的节点类型会越来越来流行。

限制为节点的访问

有多种方式用于限制对节点的访问。你已经看到了，如何使用 `hook_access()` 来限制对以节点类型的访问，以及使用 `hook_perm()` 定义权限。但是 **Drupal** 提供了控制访问的更丰富的工具集：使用表 `node_access` 以及两个访问钩子函数 `hook_node_grants()` 和 `hook_node_access_records()`。

一般情况下，**Drupal** 将拒绝对节点的访问，除非一个节点访问模块已向表 `node_access` 插入了定义如何访问的一行记录。

定义节点认可 (Grants)

对应于节点之上的三种操作：查看、更新、删除，有三个基本的权限。当这些操作中的一个将要发生时，将首先使用实现节点类型的模块里面的函数 `node_access()`。如果该模块没有定义是否允许访问的话，**Drupal** 将向所有用于节点访问控制的模块询问—该操作是否应该被允许。通过使用 `hook_node_grants()` 为每个领域 (`realm`) 每个用户得到一个许可 (`grant`) ID 列表来完成这一工作。

什么是领域 (Realm) ?

一个领域是一个任意字符串，它用于允许多个节点访问控制模块共享数据库表 `node_access`。例如，`acl.module` 是一个使用访问控制列表 (ACLs) 来管理节点访问的第三方模块，它的领域就是 `acl`。`taxonomy_access.module` 是另一第 3 方模块，它基于分类词语来限制对节点的访问，它使用 `term_access` 作为领域。所以，领域就是在表 `node_access` 中标识你的模块空间的东西；他有点像命名空间。当要你的模块返回许可 ID 时，你将从你模块定义的领域中返回它。

什么是许可 (Grant) ID

一个许可 ID 是一个标识符，用于为一个给定领域提供节点访问控制权限的信息。例如，一个节点访问控制模块——比如使用用户角色管理对论坛节点类型访问的 `forum_access.module`——可以使用角色 ID 作为许可 ID。一个使用 **US ZIP** 代码管理对节点类型访问的模块可以使用 **ZIP** 代码作为许可 ID。在每种情况下，都是与用户有关的东西作为许可 ID，比如该用户拥有这个角色么？这个用户的 **ZIP** 是 **12345** 么？用户是在这个访问控制列表中么？或者，这个用户的已订阅时间超过 **1** 年了么？

Although each grant ID means something special to the node access module that provides

grant IDs for the realm containing the grant ID, the mere presence of a row containing the grant

ID in the node_access table enables access, with the type of access being determined by the

presence of a 1 in the grant_view, grant_update, or grant_delete column. (不知道怎么翻译这一段， ^<^)

在正在保存节点时会将许可 ID 插入到表 `node_access` 中。向每个实现了钩子 `hook_node_access_records()` 的模块传递该节点对象。模块将检查节点，或者简单的返回（如果它没有为该节点处理访问控制），或者返回一个用于插入到表 `node_access` 中的许可的数组。使用 `node_access_acquire_grants()` 可以批量的插入许可。下面是一个来自于 `forum_access.module` 的例子。

```
/**
 * Implementation of hook_node_access_records().
 *
 * Returns a list of grant records for the passed in node object.
 */
function forum_access_node_access_records($node) {
  ...
  if ($node->type == 'forum') {
    $result = db_query('SELECT * FROM {forum_access} WHERE tid = %d',
      $node->tid);
    while ($grant = db_fetch_object($result)) {
      $grants[] = array(
        'realm' => 'forum_access',
        'gid' => $grant->rid,
        'grant_view' => $grant->grant_view,
        'grant_update' => $grant->grant_update,
        'grant_delete' => $grant->grant_delete
      );
    }
    return $grants;
  }
}
```

}

节点访问流程

当要对节点进行某一操作时，**Drupal** 将进入如图 7-5 所示的流程。

图 7-5 决定是否允许对给定节点的访问

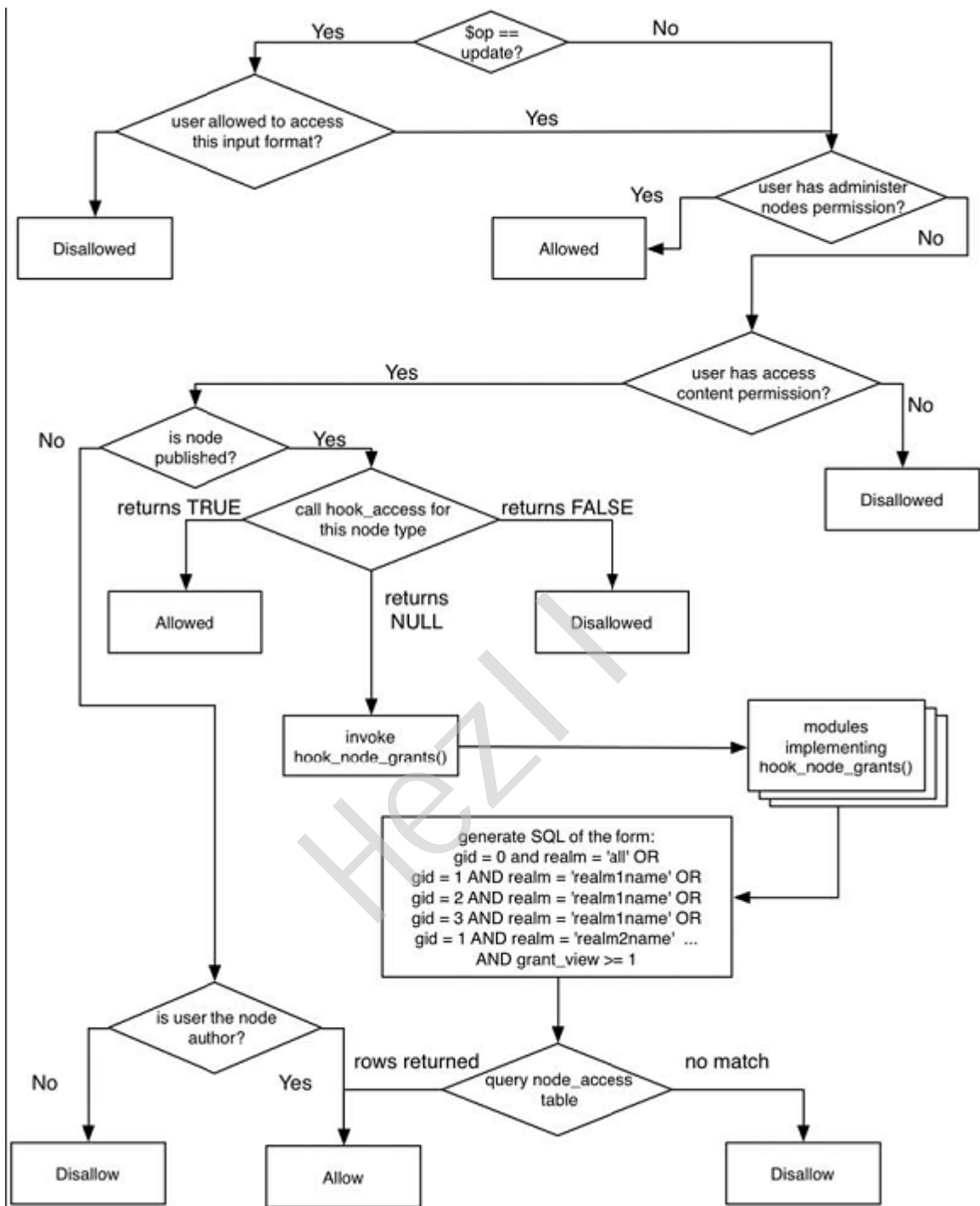
总结

读完本章后，你应该可以

- 理解什么是节点，什么是节点类型
- 编写模块来创建节点类型
- 理解如何在节点创建、保存、加载等等时插入钩子函数

HEZI

- 理解如何决定对节点的访问控制



第 8 章 Drupal 主题系统 (Drupal theme)

(1)

修改 **Drupal** 生成的 **HTML** 或者其他标识字体，你需要深入的了解主题系统的各个组成部分。主题系统是个优雅的架构，它可使你绕过核心代码，但是它有一个很长的学习曲线，特别是在你想使你的站点于其他 **drupal** 站点看起来不同时。我们将向你讲述主题系统是如何工作的，以及想你展示隐藏在 **Drupal** 核心本后的一些最佳实践。首先要记住的是：不要通过编辑模块文件内部的 **HTML** 来改变你站点的外观。如果这样做了，你仅仅创建了一个对你个人适用的内容管理系统，这样你就会失去开源软件系统最大的优势之一--社区的支持。覆盖，而不是修改。

主题系统的组成

主题系统有多个抽象层次组成：模板语言（**template language**），主题引擎（**theme engines**）和主题

模板语言和主题引擎

主题系统可以使用多个模板语言。**Smarty**,**PHPTAL**,和 **XTemplate** 可以与 **Drupal** 集成，用来向模板文件中添加动态数据。为了使用这些语言，需要一个叫做主题引擎的包装器，用来在模板语言和 **Drupal** 之间交互。你可以在

<http://drupal.org/project/Theme+engines> 找到对应的模板语言的主题引擎。你可以通过将相应主题引擎的目录放置到你站点的主题引擎目录下面来安装主题引擎。如果仅用于单个站点，使用目录 **sites/sitename/themes/engine**，如果用于多个 **Drupal** 站点，则使用目录 **sites/all/themes/engine**，如图 8-1 所示。

Drupal 社区创建了一个自己的引擎，专门对 **Drupal** 进行了优化。它叫做 **PHPTemplate**，它使用 **php** 函数来作为模板语言，这消除了其他模板语言常常使用的中间层的解析环节。这是 **Drupal** 最长用的的模板引擎，它被默认安装了。它位于 **themes/engine/phptemplate**,如图 8-2 所示：

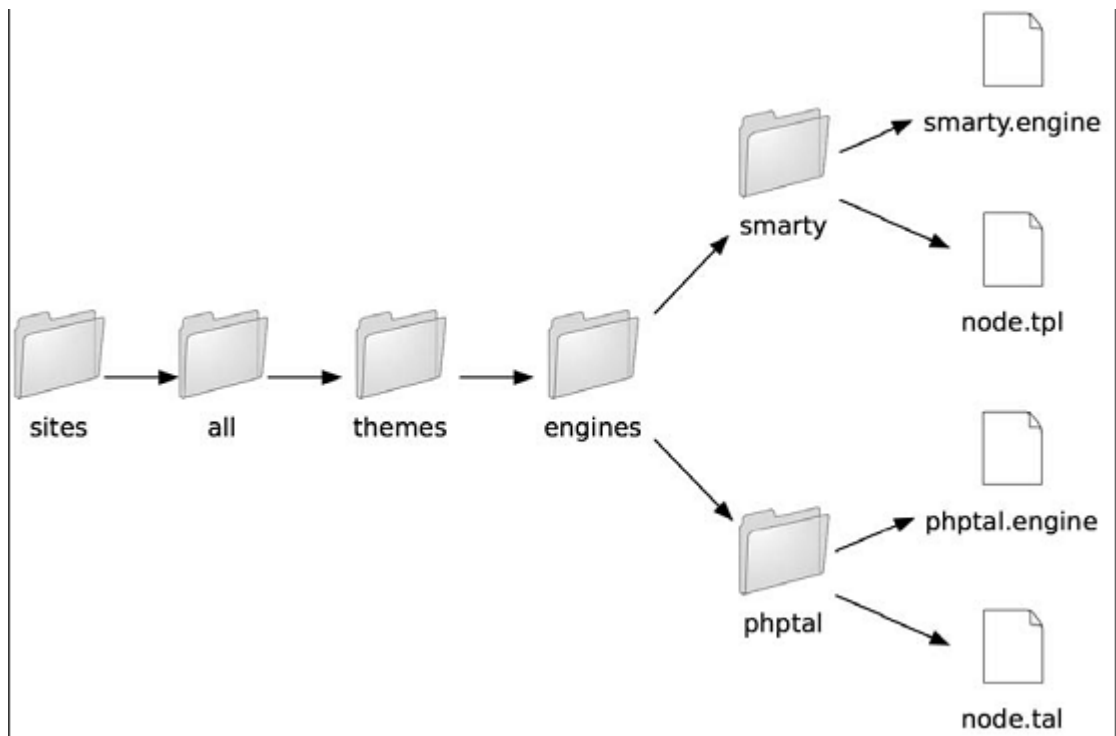


图 8-1 为 Drupal 添加定制主题引擎的目录结构

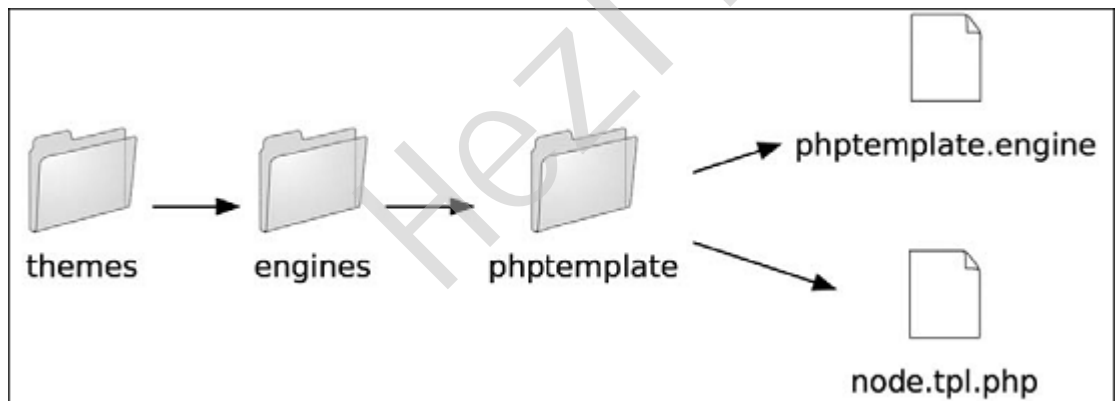


图 8-2 为 Drupal 核心主题引擎的目录结构。这个位置专门用于放置核心主题引擎。

注意：注意完全可以不使用模板语言而简单的使用 **php** 模板文件。如果你是速度的狂热分子，或者可能仅仅是想折磨一下你的设计者，你甚至可以不使用主题引擎而仅仅使用裸 **php** 函数来实现你的整个主题。例如一个基于 **php** 的主题，参看 **themes/chameleon/chameleon.theme**。

当你安装好一个主题引擎后，你不会看到你的站点的任何改变。这是因为，主题引擎仅仅是一个接口库，在主题引擎被使用以前，你仍然需要安装一个依赖于该主题引擎的 **Drupal** 主题。

要使用哪一个模板语言呢？如果你正在转换一个遗留站点，可能使用以前的模板语言更方便一些，也许你的设计团队更倾向于使用所见即所得的编辑器，这样 **PHPTAL** 应该是个更

好的选择，因为它可以阻止这些编辑器对模板的破坏。你可以看到关于 **PHPTemplate** 最多的文档和支持，如果你是从新开始建立一个站点的话，如果从长期的维护和社区支持这两个角度来看，它应该是最好的选择了。

主题 (themes)

用 **Drupal** 的话来说，主题就是一组展示你站点外观的文件。你可以从 <http://drupal.org/project/themes> 下载定已制好的主题，或者你可以自己动手。这正是你在本章将要学习的。作为一个 **web** 设计者，主题有你所期望的大部分内容组成：样式，图像，**JAVAscript** 文件，等等。你将发现，在 **Drupal** 主题和纯 **HTML** 站点之间的区别就是模板文件。这些文件一般都包含大段的静态 **HTML** 和一些小段的用来插入动态内容的代码。模板文件的语义依赖于他所支持的主题引擎。例如，列表 **8-1**，**8-2**，**8-3** 列出了 **3** 段模板文件代码片段，它们输出同样的内容但是包含完全不同的模板文件内容。

Listing 8-1. Smarty

```
<div id="top-nav">
  {if count($secondary_links)}
  <ul id="secondary">
    <li>
      {theme function='links' data=$secondary_links delimiter="</li>\n
<li>"}
    </li>
  </ul>
  {/if}
  {if count($primary_links)}
  <ul id="primary">
    <li>
      {theme function='links' data=$primary_links delimiter="</li>\n
<li>"}
    </li>
  </ul>
  {/if}
</div>
```

Listing 8-2. PHPTAL

```

<div id="top-nav">
  <ul tal:condition="php:is_array(secondary_links)" id="secondary">
    <li tal:repeat="link secondary_links" tal:content="link">secondary
link</li>
  </ul>
  <ul tal:condition="php:is_array(primary_links)" id="primary">
    <li tal:repeat="link primary_links" tal:content="link">primary
link</li>
  </ul>
</div>

```

-

Listing 8-3. PHPTemplate

```

<div id="top-nav">
  <?php if (count($secondary_links)) : ?>
  <ul id="secondary">
    <?php foreach ($secondary_links as $link): ?>
    <li><?php print $link?></li>
    <?php endforeach; ?>
  </ul>
  <?php endif; ?>
  <?php if (count($primary_links)) : ?>
  <ul id="primary">
    <?php foreach ($primary_links as $link): ?>
    <li><?php print $link?></li>
    <?php endforeach; ?>
  </ul>
  <?php endif; ?>
</div>

```

每一个模板文件由于它所使用的模板语言的不同看起来也不同。模板文件的后缀决定于它所使用的模板语言，也就是它所依赖的主题引擎（参看表 8-1）

表 8-1 模板文件的扩展名意味着它所依赖的模板语言。

Template File	Extension	Theme Engine
theme		PHP
.tpl.php		PHPTemplate*
.tal		PHPTAL
.tpl		Smarty

第 8 章 Drupal 主题系统 (Drupal theme)

(2) 安装主题

安装主题

为了使在 **Drupal** 管理界面能够呈现一个新的主题，你需要把它放到 **sites/sitename/themes** 下面。如果你想在一个多站点 **Drupal** 系统中使所有的站点都可以使用它，那么你需要把它放到 **sites/all/themes** 下面。你可以在你的站点安装任意多个主题，而主题的安装和模块的安装基本上相同。将主题文件放到相应的位置后，导航到管理页面 **Administer > Site building > Themes**。你可以一次安装和启用多个主题。这意味着什么？通过启用多个主题，用户可以在他们的个人首页上选择启用主题中的任何一个，在用户访问站点时，就使用所选的主题了。

当下载或者创建一个新的主题时，将新主题和核心主题以及贡献主题区分开来是个很好的习惯。我们推荐在你的 **themes** 文件夹下面建立一个另一个层次文件夹。将定制的主题放到文件夹 **custom**，而将从 **drupal.Org** 下载下来的社区贡献的主题放到 **drupal-contrib**。

建造一个 PHPTemplate 主题

有多种基本方式可以创建一个主题，这依赖于你开始所拥有的材料。将如你的设计者已经为你的提供了站点的 **HTML** 和 **CSS** 文件。那么将设计者的设计转化为一个主题有多么简单呢？它确实不是很难，你可能已经完成了工作的 **80%**。那么剩下的 **20%**--最后的小饮和甜点—它将 **Drupal** 的主题花与静态 **HTML** 区分开了（英文意思为将男仆与日本武士区分开来）。首先让我们从简单的开始。

注意：如果你从零开始设计你的主题的话，在开放源代码 **WEB** 设计站点 <http://www.oswd.org> 里面有很多非常棒的设计可供借鉴（注意他们是 **HTML** 和 **CSS** 设计，而不是 **Drupal** 主题）

我们假设给了你列表 **8-4** 和 **8-5** 中所示的 **HTML** 页面和 **CSS** 样式，让你将他们转化为一个 **DRUPAL** 主题。显然在一个真实的项目中，你所受到的文件应该比这些更细节，但是从这里你可以学到方法。

Listing 8-4. page.html

```
<html>
  <head>
    <title>Page Title</title>
    <link rel="stylesheet" href="global.css" type="text/css" />
  </head>
  <body>
```

```
<div id="container">
  <div id="header">
<h1>Header</h1>
  </div>
  <div id="sidebar-left">
<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam
  nonummy nibh euismod tincidunt ut.
</p>
  </div>
  <div id="main">
<h2>Subheading</h2>
<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam
  nonummy nibh euismod tincidunt ut.
</p>
  </div>
  <div id="footer">
Footer
  </div>
</div>
</body>
</html>
```

Listing 8-5. global.css

```
#container {
  width: 90%;
  margin: 10px auto;
  background-color: #fff;
  color: #333;
  border: 1px solid gray;
  line-height: 130%;
}
#header {
  padding: .5em;
  background-color: #ddd;
  border-bottom: 1px solid gray;
}
#header h1 {
  padding: 0;
  margin: 0;
```

```

}
#sidebar-left {
  float: left;
  width: 160px;
  margin: 0;
  padding: 1em;
}
#main {
  margin-left: 200px;
  border-left: 1px solid gray;
  padding: 1em;
  max-width: 36em;
}
#footer {
  clear: both;
  margin: 0;
  padding: .5em;
  color: #333;
  background-color: #ddd;
  border-top: 1px solid gray;
}
#sidebar-left p { margin: 0 0 1em 0; }
#main h2 { margin: 0 0 .5em 0; }

```

该设计如图 8-3 所示

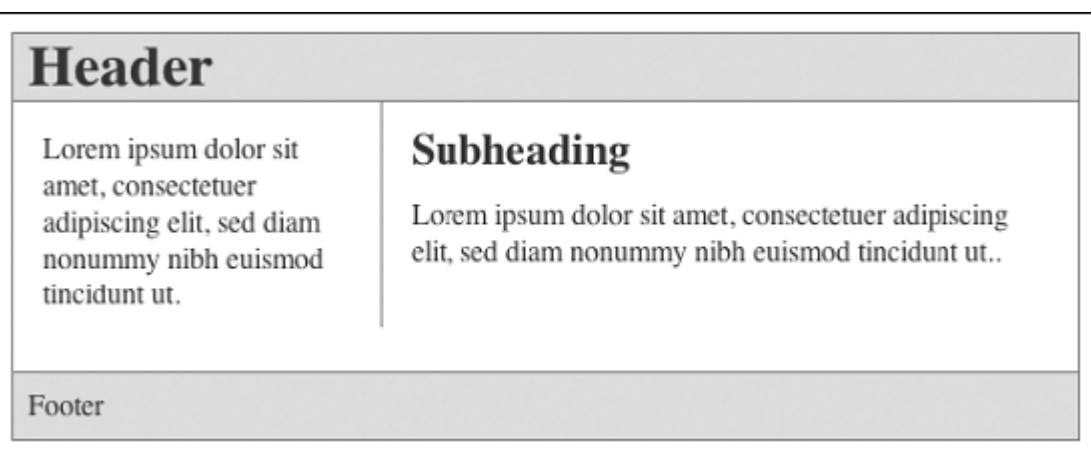


图 8-3 在转化为 Drupal 主题以前的设计

让我们将新主题叫做 greyscale,所以在文件夹 sites/all/themes/custom 下面创建一个子文件夹 greyscale。如果不存在 sites/all/themes/custom 的话你需要创建一个。将

`page.html` 和 `global.css` 复制到 `greyscale` 文件夹下面。接下来，将 `page.html` 重命名为 `page.tpl.php`，这样他将作为为每个 `Drupal` 页面服务的一个新的页面模板。由于主题 `greyscale` 有了一个 `page.tpl.php` 文件，现在你就可以在管理页面中激活它。到 **Administer > Site building > Themes** 下面，将它设置为默认主题。

恭喜!现在你应该可以实际看到你的新设计了。外部的样式还没有加载进来（我们将在后面讨论它），而在你的站点中对任何页面的导航都显示这个同一个页面，尽管如此，这也是一个了不起的开始。由于在你的站点中到任何页面的导航都显示 `page.tpl.php` 中的静态 `HTML` 内容，所以现在你没有办法进入管理页面了。我们将你关到了 `Drupal` 的门外面。哎哟。一不小心被关到了门外面，下面我们将向你讲述如何解套。一种办法是对刚才启用的主题进行重命名。在这种情况下，你可以简单的将 `greyscale` 改为 `greyscale_` 这样你就可以重新返回站点里面了。那是一个快速解决办法，由于你知道问题的真正所在，另一种方式，你可以向 `page.tpl.php` 中添加适当的变量，从而展示 `Drupal` 的动态内容而不是上面的静态内容。

每一个 `PHPTemplate` 模板文件---无论 `page.tpl.php`，`node.tpl.php`，`block.tpl.php` 还是其它---都有一组动态内容变量提供给他们使用。打开 `page.tpl.php` 将相应的静态内容替换为相应的 `Drupal` 变量。不要担心，我们很快后面讲述这些变量。

```
<html >
  <head>
    <title><?php print $head_title ?></title>
    <link rel="stylesheet" href="global.css" type="text/css" />
  </head>
  <body>
    <div id="container">
      <div id="header">
<h1><?php print $site_name ?></h1>
      </div>
      <?php if ($sidebar_left): ?>
      <div id="sidebar-left">
<?php print $sidebar_left ?>
      </div>
      <?php endif; ?>
      <div id="main">
<h2><?php print $title ?></h2>
<?php print $content ?>
      </div>
      <div id="footer">
<?php print $footer_message ?>
      </div>
    </div>
```

```
</body>
</html >
```

重新加载页面，你将发现，变量被 **Drupal** 的相应内容所替换。你将注意到 **global.css** 样式单没有被加载，这是因为指向该文件的路径不正确。你可以手工的调整其路径，或者你可以以 **Drupal** 的方式来完成它以获得更好的灵活性和其它好处。

首先将 **global.css** 重命名为 **style.css**。根据习惯，**Drupal** 将自动的查找每个主题下面的 **style.css** 文件，它将该信息添加到变量 **\$styles** 里面，从而被传递到 **page.tpl.php**。让我们使用这一信息更新 **page.tpl.php**。

```
<html >
  <head>
    <title><?php print $head_title ?></title>
    <?php print $styles ?>
  </head>
  ...
```

保存你的修改并重新加载页面。瞧。如果你查看页面的源代码的话，你将注意到，其他启用的模块所带有的样式单也被加载了进来，这些都是通过变量 **\$styles** 完成的。通过将你的 **Css** 文件命名为 **style.css**，这允许 **Drupal** 使用它的 **css** 预处理引擎来处理它，从而消除 **CSS** 文件中所有的空白和换行，替换的，**drupal** 没有使用多个样式单，而是将它们合并到了一起，作为一个文件提供。关于这一特性的更多细节，参看第 **22** 章。

这里有更多的可添加到 **page.tpl.php** 和其它模板文件的变量。让我们逐一考察一下。

第 8 章 Drupal 主题系统 (Drupal theme)

(3) 模板文件

理解模板文件

一些主题包含所有的各种模板文件，而其他仅包含 **page.tpl.php**。所以你如何知道你可以创建那些模板文件以及哪些可被 **Drupal** 识别？创建模板文件时所遵循的命名习惯有哪些？在接下来的部分，你将学习到使用模板文件的里里外外的各种知识。

模板文件 page. tpl . php

page.tpl.php 是所有其它模板文件的祖宗，并为站点提供了整体的外观，其它模板文件被插入到了 **page.tpl.php**，如图 8-4 所说明的。

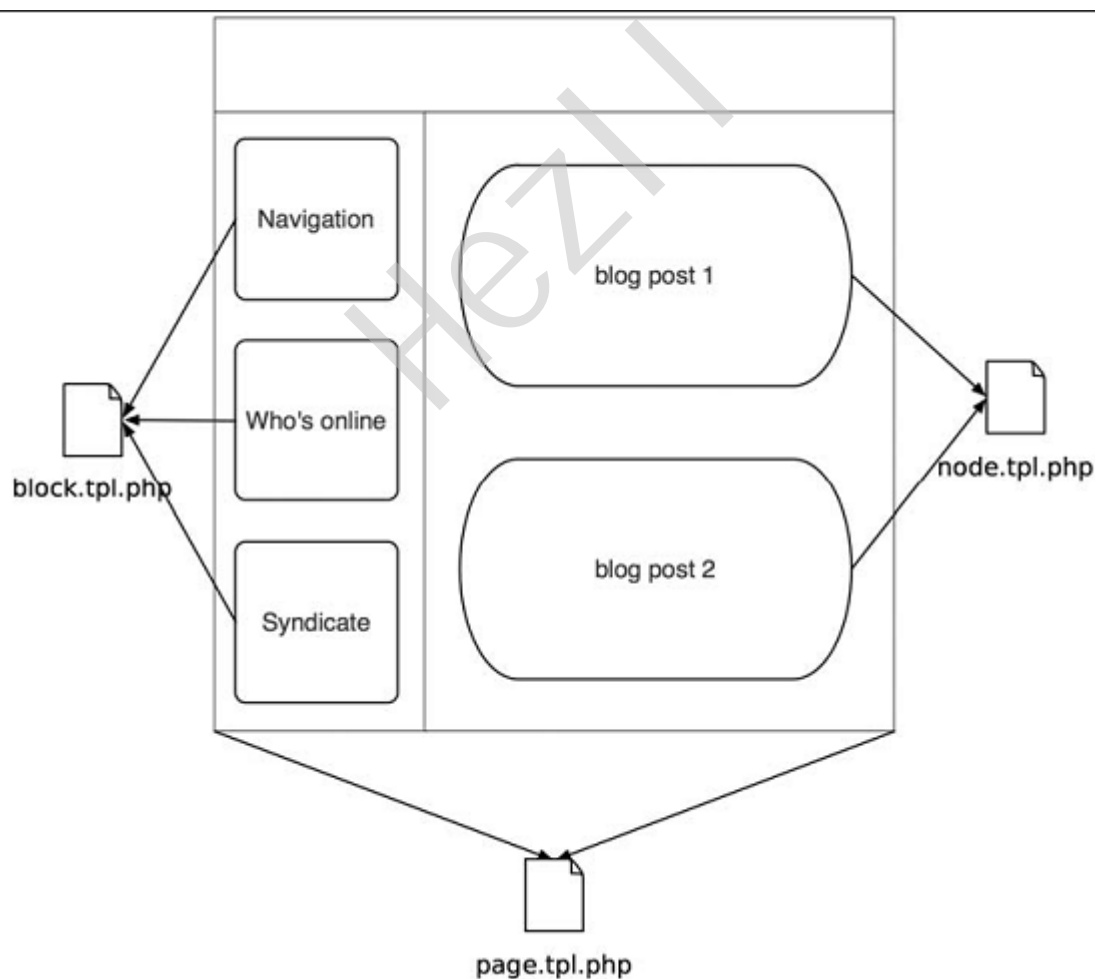


图 8-4 其它的模板被插入到了 **page.tpl.php** 文件中

在图 8-4 中 `block.tpl.php` 和 `node.tpl.php` 的插入由主题系统自动完成。记住在前面的例子中什么时候你创建了 `page.tpl.php` 文件？好的，变量 `$content` 包含了对 `node.tpl.php` 输出的调用，而 `$sidebar_left` 包含了对 `block.tpl.php` 输出的调用。

当你想为其它的页面创建不同的外观时，而一个简单的页面外观不再适用了，你想怎么办？有两种好的方式可以创建更多的页面模板。

首先，你可以通过基于站点当前系统 URL 来创建额外的页面模板文件。例如，如果你想访问 <http://example.com/?q=user/1>，那么 PHPTemplate 将以下面的顺序来查找页面模板：

`page-user-1.tpl.php`

`page-user.tpl.php`

`page.tpl.php`

PHPTemplate 一旦找到一个要包含的模板文件将会停止继续寻找。`page-user.tpl.php` 可被所有的用户页面使用，而 `page-user-1` 仅可以在 URL 为 `user/1`，`user/1/edit` 时执行，等等。

注意：在这里 Drupal 使用的是内部的 URL，所以如果你是用了允许别名化 URL 的模块 `path` 或者 `pathauto` 时，你仍然需要引用 Drupal 的内部 URL，而不是使用别名。

让我们使用节点编辑页面 <http://example.com/?q=node/1/edit> 作为一个例子。下面是 PHPTemplate 查找页面模板文件的顺序：

`page-node-edit.tpl.php`

`page-node-1.tpl.php`

`page-node.tpl.php`

`page.tpl.php`

警告：如果你没有在 `page.tpl.php` 中使用区域变量（`$header`，`$footer`，`$sidebar_left`，`$sidebar_right`），但是它们仍然被创建了。这是一个性能问题，因为 Drupal 构建的所有区块，而对于一个特定的页面视图却把它们扔到了一遍。如果定制页面视图不需要区块，一个比从模板文件中排除该变量的更好的方式是到区块管理页面中去，当展示定制页面时，将这些区块禁用掉。参看第 9 章，查看关于在特定页面禁用区块的更多信息。

细节：为你的站点的首页创建一个定制的页面模板，简单的创建一个名为 `page-front.tpl.php` 的模板文件即可

如果你需要创建一个定制的页面模板，你可以从复制你当前的 `page.tpl.php` 的开始，然后将它修改为你所需要的。下面是在页面模板中可用到的变量：

`$base_path`: Drupal 安装的基本路径。如果安装在根目录下，这是最简单的，他将默认为根目录。

- **`$breadcrumb`**: Returns the HTML for displaying the navigational breadcrumbs on the page.

- **`$closure`**: 返回 `hook_footer()` 的输出，它常在页面的底部使用。

- **`$css`**: 返回一个所有 `css` 文件组成的数组结构，以添加到页面中去。使用 `$styles` 来返回 `$css` 数组的 HTML 版本。

- **`$content`**: 返回将要展示的 HTML 内容。例如，它可以包含一个节点，一组节点，管理接口的内容，等等。

- **`$directory`**: 主题所在的相对路径。例如 `themes/bluemarine` 或者 `sites/all/themes/custom/mytheme`。你通常联合使用该变量和 `$base_path` 来构建你的站点主题的绝对路径：`<?php print $base_path . $directory ?>`

- **`$feed_icons`**: 返回该页面的 RSS 种子链接

- **`$footer_message`**: 返回页脚信息文本，在下面页面输入它：

Administer > Site configuration > Site information.

- **`$head`**: 返回放置在 `<head></head>` 部分的 HTML。模块可以通过调用 `drupal_set_html_head()` 来向 `$head` 添加额外的比如 RSS 种子的纯文本。

- **`$head_title`**: 在页面标题中展示的文本，放在 HTML `<title></title>` 标签中。

- **`$help`**: 帮助文本，大多数用于管理页面。模块可以通过实现 `hook_help()` 来提供该变量。

- **`$is_front`**: 如果当前展示的为首页的话为真 `TRUE`。

- **`$language`**: 站点展示时所使用的语言

- **`$layout`**: 这一变量允许你定义外观的不同类型的风格，而变量 `$layout` 的值依赖于启用的工具条 (`sidebars`) 的数量。可能的值包括: `none, left, right, and both`。

- **`$logo`**: 指向 `logo` 图像的路径，在启用主题的主题配置页面定义。通常这样使用：

``

- **\$messages**: 为表单或者其他信息返回的验证错误和正确的提示信息的 HTML。它通常显示在页面的头部。

- **\$mission**: 返回站点使命文本, 在 **Administer > Site configuration > Site information** 中输入. 只有当 **\$is_front** 为 **TRUE** 时才可以使⤵用。

- **\$node**: 整个节点对象, 当察看一个单独节点页面时可用。

- **\$primary_links**: 一个包含了一级链接的数组。在 **Administer > Site building > Menus** 中定义它们。通常 **\$secondary_links** 通过函数 **theme('links')** 来定制输出的样式, 如下所示:

```
<?php print theme('links', $primary_links) ?>
```

- **\$scripts**: 返回向页面的 **<script>** 标签中所添加的 HTML。这也是关于 **jQuery** 如何被加载的 (参看第 17 章查看关于 **jQuery** 的更多信息)

- **\$search_box**: 返回搜索表单的 HTML。当管理员在启用的主题中的主题配置页面禁止展示搜索时, 或者搜索模块禁用时, **\$site_slogan** 为空。

- **\$secondary_links**: 一个包含了二级链接的数组。在 **Administer > Site building > Menus** 中定义它们。通常 **\$secondary_links** 通过函数 **theme('links')** 来定制输出的样式, 如下所示:

```
<?php print theme('links', $secondary_links) ?>
```

- **\$sidebar_left**: 返回左边工具条的 HTML, 包含了属于该区域的所有区块的 HTML。

- **\$sidebar_right**: 返回右边工具条的 HTML, 包含了属于该区域的所有区块的 HTML。

- **\$site_name**: 站点的名称。在 **Administer > Site configuration > Site information** 中设置。当管理员在启用的主题中的主题配置页面禁止展示标语时, **\$site_name** 为空。

- **\$site_slogan**: 站点的标语。在 **Administer > Site configuration > Site information** 中设置。当管理员在启用的主题中的主题配置页面禁止展示标语时, **\$site_slogan** 为空。

- **\$styles**: 返回连接到页面需要的 CSS 文件的 HTML。CSS 文件通过 **drupal_add_css()** 添加到 **\$styles** 中去。

- **\$tabs**: 返回用于为节点展示诸如 **View/Edit** 的标签的 HTML。

- **\$title**: 主内容的标题，与**\$head_title**不同。当察看一个单独节点视图页面时，**\$title**就是节点的标题。当常看 **Drupal** 的管理员页面时，**\$title** 通常有菜单项来设置，菜单项对应于查看的页面。（参看第 4 章，关于菜单项的更多信息）。

模板文件 node.tpl.php

节点模板负责控制一个页面内部的一片单独的内容的展示。而不是影响整个页面，节点模板仅影响 **page.tpl.php** 中的变量**\$content**。他们负责节点以 **teaser** 视图的方式展示（当多个节点在同一个页面列出时），或者以 **body** 视图的方式（当节点填充 **page.tpl.php** 中的整个变量**\$content** 并单独出现在他自己的页面）。节点文件中的变量**\$page**，当为 **body** 视图方式时，它为真，当为 **teaser** 视图方式时，它为假。

模板文件 **node.tpl.php** 是一个处理所有节点视图的一般模板。如果你想要一个不同的模板，比如说，日志模板而不是论坛节点模板？你如何才能为特定节点类型创建一个专有的模板而不是全部都使用一个通用的模板？

幸好节点模板提供了一个令人喜欢的粒度而没有超出盒子的范围。简单的将 **node.tpl.php** 复制一份并重命名为 **node-nodetype.tpl.php**，这样 **PHPTemplate** 就可以选择这一个模板而不是通用模板。所以为 **blog** 实体定制输出样式时，仅需要简单的使用 **node-blog.tpl.php** 即可。你通过 **Administer > Content management > Content types** 创建的任何节点类型都可以以同样的方式拥有一个单独的模板文件。在节点模板中，你可以使用下面的变量：

- **\$content**: 节点的主体部分，如果是一个分页显示的结果时，它为 **teaser**。
- **\$date**: 节点被创建的格式化日期。
- **\$links**: 与节点相关的链接，比如“**read more**” 或者“**add comment.**”模块通过实现 **hook_link()** 来添加额外的链接。
- **\$name**: 创建该页面的用户名，连接到他的个人主页。
- **\$node**: 整个节点对象和它的所有属性。
- **\$node_url**: 该节点的持久化的 **URI**。
- **\$page**: ， 当为 **body** 视图方式时，它为真，当为 **teaser** 视图方式时，它为假。
- **\$taxonomy**. 由节点的分类词语构成的一个数组

-
- **\$teaser**: 布尔值，用来决定是否展示 **teaser**。当它为假时，意味着节点采用 **body** 方式展示，为真时，表示以 **teaser** 方式展示。
 - **\$terms**: 与该节点相关的分类单词的 **HTML**。每一个单词都指向他自己的分类单词页面。
 - **\$title**: 节点的标题。当在多个节点列表的页面这里有个链接指向该节点的主体视图。
 - **\$submitted**: “Submitted by”文本。管理员可以配置这一信息的展示，在一个基于单个节点类型的主题配置页面配置。
 - **\$picture**: 用户图像的 **HTML**，如果启用了图像并且设置了用户图像。

通常节点模板文件中的变量 **\$content** 并不会像你期望的方式构建数据。当使用了一个扩展了节点属性的贡献模块比如 **CCK** 等字段相关的模块时，这尤其正确。

幸运的是，**PHPTemplate** 将整个节点对象传递给了节点模板文件。如果你在你的模板文件中的头部使用下面的调试语句并加载包含节点的页面，你将发现组成节点的所有属性。如果查看你浏览页面的源代码的话，读起来可能更容易一些。

```
<pre>
<?php print_r($node) ?>
</pre>
```

现在你可以看到组成节点的所有部分了，直接访问他们的属性，像期望的那样标识它们，而不是使用聚合的变量 **\$content**。

警告：当直接格式化一个节点对象时，你也必须负责你站点的安全。请参看第 20 章来学习如何使用适当的函数来过滤用户提交的数据以阻止 **XSS** 攻击

模板文件 block.tpl.php

区块在 **Administer > Site building > Blocks** 中列出，并且由 **block.tpl.php** 提供的标识包装。如果你对区块不熟悉的话，可以参看第 9 章以获得更多信息。像页面模板和节点模板文件一样，区块系统使用了一个建议的层次来查找包裹了区块的模板文件。该层次如下所示：

block-modulename-delta.tpl.php

block-modulename.tpl.php

block-region.tpl.php

block.tpl.php

在上面的序列中, **modulename** 是实现了该区块的模块的名称.例如, 区块“Who’s Online”由模块 **user.module** 实现。由站点管理员创建的区块都指向区块模块。如果你不知道那个模块实现了给定的区块, 通过使用一些 **PHP** 调试, 你可以找到所有的原始信息.通过在你的 **block.tpl.php** 的头部键入下面一行代码, 你将打印出在当前页面启用的每个区块的整个区块对象。

```
<pre>
<?php print_r($block); ?>
</pre>
```

如果你查看浏览页面的源代码的话, 这会更容易读些。下面是区块“Who’s Online”所展示的:

```
stdClass Object
(
  [module] => user
  [delta] => 3
  [theme] => bluemarine
  [status] => 1
  [weight] => 0
  [region] => footer
  [custom] => 0
  [throttle] => 0
  [visibility] => 0
  [pages] =>
  [title] =>
  [subject] => Who's online
  [content] => There are currently ...
)
```

现在你得到了该区块的所有的细节了, 你可以非常容易的构建一个或多个如下所示的区块模板文件这依赖于你所想要指向的范围:

`block-user-3.tpl.php` // Target just the Who's Online block.

`block-user.tpl.php` // Target all block output by user module.

`block-footer.tpl.php` // Target all blocks in the footer region.

`block.tpl.php` // Target all blocks on any page.

下面是在区块模板文件中你可以使用的变量：

- **\$block**: 整个区块对象。
- **\$block_id**: 一个整数，当生成一个区块时，递增，同时区块模板文件被调用。
- **\$block_zebra**: 当**\$block_id** 递增时，该变量不断的在“odd” and “even.”之间变换。

模板文件 `comment.tpl.php`

模板文件 `comment.tpl.php` 为评论添加修饰。它不像节点那样可以容易的将日志评论和论坛评论区分开来。可以这样做，但是必须使用代码完成，并且好好的钻研一下函数 `phptemplate_variables()`。下面是使用评论模板文件时可使用的变量：

- **\$author**: Hyperlink author name to the author's profile page, if he or she has one.
- **\$comment**: Comment object containing all comment attributes.
- **\$content**: The body of the comment.
- **\$date**: Formatted creation date of the post.
- **\$links**: Contextual links related to the comment such as “edit, “reply,” and “delete.”
- **\$new**: Returns “new” for a comment yet to be viewed by the currently logged in user and “updated” for an updated comment. You can change the text returned from `$new` by overriding `theme_mark()` in `includes/theme.inc`. Drupal doesn't track which comments

have been read or updated for anonymous users.

- **\$picture**: HTML for the user picture. You must enable picture support at Administer > User management > User settings, and you must check “User

pictures in comments” on each theme’s configuration page for enabled themes. Finally, either the site administrator must provide a default picture or the user must upload a picture so there is an image to display.

- **\$submitted**: “Submitted by” string with username and date.
- **\$title**: Hyperlink title to this comment.

模板文件 `box.tpl.php`

模板文件 `box.tpl.php` 是 **Drupal** 内部最容易引起歧义的模板文件了。他在 **Drupal** 核心中使用，用来包裹评论的提交表单和查询结果。除了这些，很少用的到它。它对区块不起作用，这可能是个一出错的地方（这是因为由管理员创建的区块在数据库中存贮使用的表名为 **boxes**）。在盒子模板中你可使用的变量如下：

- **\$content**: 盒子的内容
- **\$region**: 盒子应该被展示的区域。例如包括 `header`, `sidebar-left`, 和 `main`
- **\$title**: 盒子的标题。

第 8 章 Drupal 主题系统 (Drupal theme)

(4) 高级特性--1, 覆写主题

Drupal 主题高级特性

在前面的部分，你学到了 **Drupal** 使用的各种不同模板文件，当 **Drupal** 要将你的主题合并到一起时就会查找这些模板。你学到了如何创建页面模板文件，如何创建特定节点类型的节点模板文件，甚至特定区块的区块模板文件。换句话说，对于定制主题的知识，你已经掌握了 **80%**。

那么剩下的 **20%**呢？如何主题化 **Drupal** 的表单？如何修改一些简单的东西比如面包屑的外观？在这一部分，我们将回答这些问题并帮你成为 **Drupal** 主题化的高手。你将从主题化高手的基本武器之一 **template.php** 文件开始学起。

template.php 文件是整合定制的模板文件的地方：定义新的区块区域，覆写 **Drupal** 的默认主题函数，拦截传递给模板文件的变量并创建定制的变量。

覆写主题函数

Drupal 的主题系统背后的核心哲理和钩子系统的类似。通过遵循命名规范，就可以标识出哪些函数是主题相关的函数，它们负责格式化并返回你站点的内容。主题函数由它们函数名的前缀“**theme_**”来标识。这一命名规范使得 **Drupal** 能够为所有的主题函数创建一个函数覆写机制。设计者就可以命令 **Drupal** 执行一个具有更高优先级的定制的函数，从而代替开发者在模块中给出的默认的主题函数。例如，让我们检查一下当构建站点的面包屑时该流程是怎么工作的。

打开 **includes/theme.inc** 并检查文件中的函数。几乎所有的函数都以 **theme_**开头，这就告诉人们它是可以被覆写的。特别的，我们检查 **theme_breadcrumb()**:

```
/**
 * Return a themed breadcrumb trail.
 *
 * @param $breadcrumb
 *   An array containing the breadcrumb links.
 * @return a string containing the breadcrumb output.
 */
function theme_breadcrumb($breadcrumb) {
  if (!empty($breadcrumb)) {
```

```
        return '<div class="breadcrumb">'. implode(' è ',
$breadcrumb) . '</div>';
    }
}
```

这个函数控制着 **Drupal** 中面包屑导航条的 **HTML** 输出。当前，它在面包屑的每一项之间添加了一个指向右边的双箭头分隔符 (>>)。假定你想将 **div** 标签改为 **span** 标签，并使用星号 (*) 来代替双箭头 (>>)。那么你该怎么办呢？一种方式是在 **theme.inc** 中修改这个函数，保存，并调用。这样也能达到目的。（不！不！千万不要这样做！）。我们有一些更好的方式。

你见过 **Drupal** 内核中是怎么调用这些主题函数么？你永远都不会看到直接调用 **theme_breadcrumb()** 的情况。替代的，它通常包装在助手函数 **theme()** 中。

你期望函数这样调用：

```
theme_breadcrumb($breadcrumb)
```

但实际不是这样。替代的，你将看到开发者这样调用：

```
theme('breadcrumb', $breadcrumb);
```

这个通用的 **theme()** 函数负责初始化主题层并将函数调用分发到合适的位置，这使得我们能够以更优雅的方案来解决我们的问题。图 8-5 展示了通过调用 **theme()**，命令 **Drupal** 按照下面的次序查找相应的面包屑函数。

假定你使用的主题为 **bluemarine**，它是基于 **PHPTemplate** 的主题，那么 **Drupal** 将会查找下面的函数：

```
bluemarine_breadcrumb()
```

```
phptemplate_breadcrumb()
```

```
theme_breadcrumb()
```

为了修改 **Drupal** 的面包屑，在当前主题的文件夹下创建一个空的 **template.php** 文件，并将 **theme.inc** 中的 **theme_breadcrumb()** 函数复制并粘贴到该文件里面。记住要包含 **<?php** 标签。还有对函数进行重命名，将 **theme_breadcrumb** 改为 **mytheme_breadcrumb**。

```

<?php
/**
 * Return a themed breadcrumb trail.
 *
 * @param $breadcrumb
 * An array containing the breadcrumb links.
 * @return a string containing the breadcrumb output.
 */
function mytheme_breadcrumb($breadcrumb) {
  if (!empty($breadcrumb)) {
    return '<span class="breadcrumb">'. implode(' * ',
$breadcrumb) .'</span>';
  }
}

```

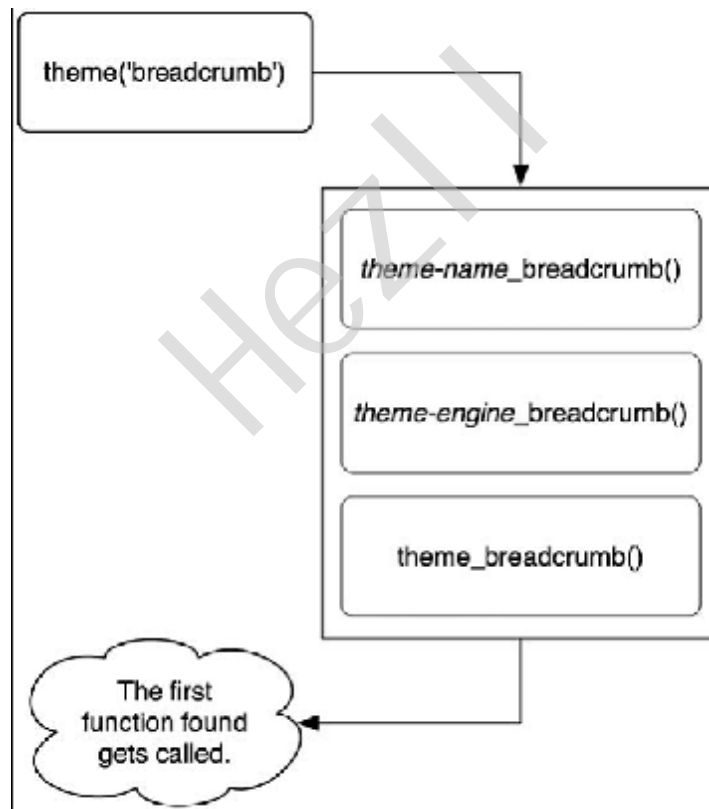


图 8-5 覆写主题的工作原理

当下一次 Drupal 需要生成面包屑时，它就会先找到你的函数并使用它来代替默认的 `theme_breadcrumb()` 函数，这样面包屑中就会包含你的星号了，而不再包含默认的双箭头了。很漂亮，不是吗？你没有修改一行核心代码。通过 `theme()` 函数来管理所有的主题函数调用，如果当前主题覆写了任何一个 `theme_` 函数，Drupal 都可以检查到这些覆写的函数并使用它们来代替默认的主题函数。开发者，请注意：在你的模块中任何需要输出

HTML 或者 XML 的部分都应该使用以“`theme_`”开头的主题函数，这样 s 设计者就可以覆写它们了。拿 `user.module` 来说（参看图 8-6）

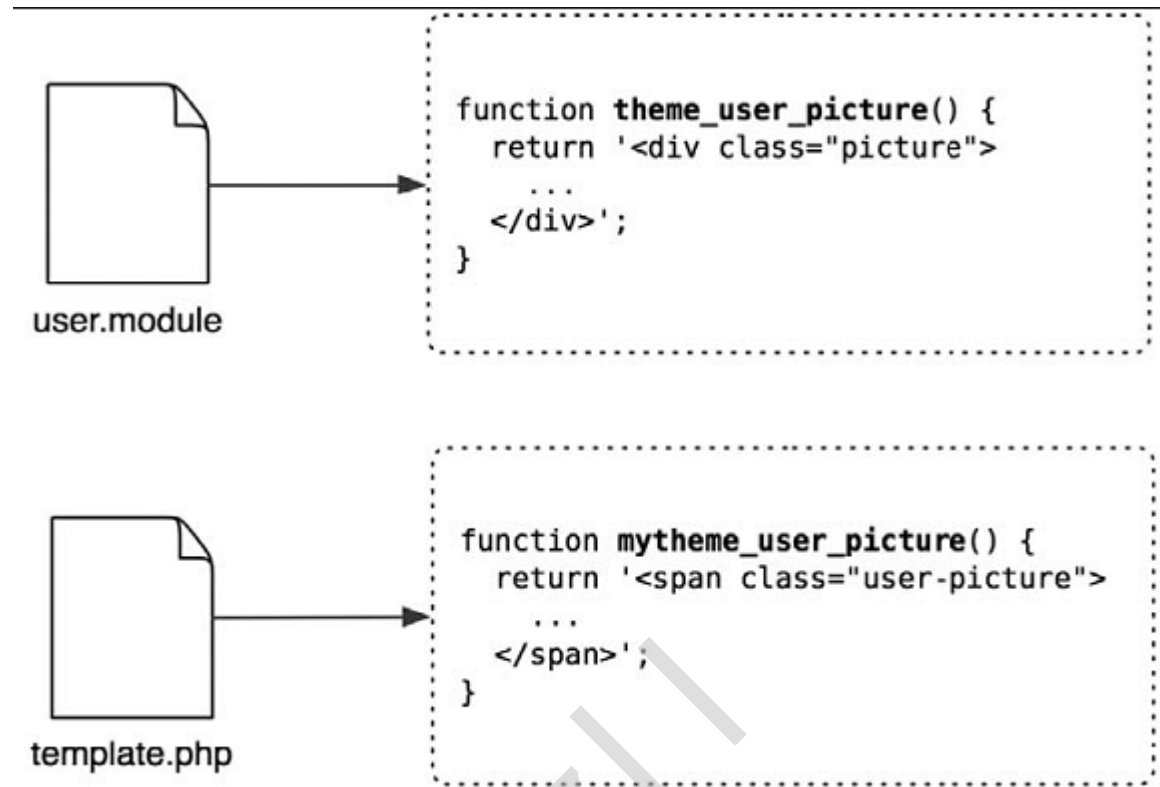


图 8-6 可以在主题层覆写由模块暴露的主题函数。定制的主题函数 `mytheme_user_picture()` 覆写了默认的 `theme_user_picture()`，接下来将会使用 `mytheme_user_picture()` 代替 `theme_user_picture()`。

第 8 章 Drupal 主题系统 (Drupal theme)

(4) 高级特性--2, 添加和操纵模板变量

定义额外的模板文件

如果你和一个设计者一同工作，你想让他/她从代码中找到主题函数并覆写它，这有点难为人的。幸运的是，有另一种方法，使得设计者更容易的修改外观。你可以将匹配的主题函数替换为它们自己的模板文件。我们将通过大家熟悉的面包屑的例子来说明这一点。

首先，在你的主题目录下创建一个名为 `breadcrumb.tpl.php` 的文件。这是用于面包屑的新的模板文件。因为我们想将 `<div>` 标签替换为 `` 标签，继续前进，向该文件中添加以下内容：

```
<span class="breadcrumb"><?php print $breadcrumb ?></span>
```

现在设计者就很容易编辑了。现在你需要在 **Drupal** 需要显示面包屑时告诉它调用这个模板文件。在 `template.php` 里，像你前面做的那样覆写 `theme_breadcrumb()`，但是这次在函数内部你将使用模板文件来代替相应的功能：

```
function mytheme_breadcrumb($breadcrumb) {
    if (!empty($breadcrumb)) {
        $variables = array(
            'breadcrumb' => implode(' * ', $breadcrumb)
        );
        return _phptemplate_callback('breadcrumb', $variables);
    }
}
```

这个函数的关键点在于 `_phptemplate_callback()`。它的第一个参数是要查找的模板文件的名称，第二个参数是传递给模板文件的变量数组。你的模板文件需要多少变量，你就可以创建并传递多少。

添加和操纵模板变量

问题又来了：如果你可以创建你自己的模板文件并控制传递给它们的参数，那么你怎么操纵或者添加传递给页面和节点模板文件的变量呢？

每次加载一个模板文件都需要调用 `_phptemplate_callback()` 函数，我们在前面已经介绍了该函数。这个函数负责收集变量并将其传递给合适的模板文件中，而这些变量可来自于不同的位置：

- 作为 `_phptemplate_callback()` 的第 2 个参数传递过来的 `$variables` 数组。
- 通过 `phptemplate.engine` 中的 `_phptemplate_default_variables()` 为每个模板文件追加的默认变量。
- 从 `_phptemplate_variables()` 返回的变量。这个函数在 **Drupal** 中，默认情况下是不存在的，在变量发送给相应的模板文件以前，你可以在你的主题内使用这个函数来操纵变量。

图 8-7 展示了如何将其与更高的主题系统绑定到一起

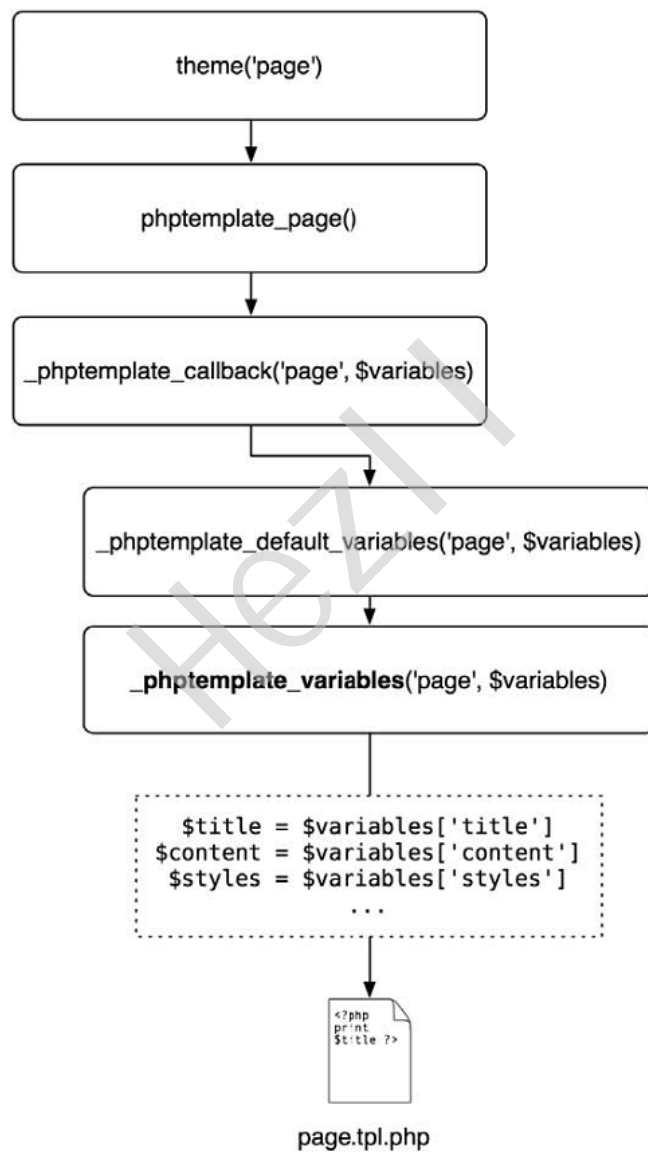


图 8-7 开发者可以使用 `_phptemplate_variables()` 函数来拦截并操纵发送给模板文件的变量

`_phptemplate_variables()` 的一个通常的用法是，当有人访问站点而且已经登录时，为其设置一个变量。向你的 `template.php` 文件中添加以下变量：

```

/**
 * Intercept template variables.
 *
 * @param $hook
 * The name of the theme function being executed.
 * @param $vars
 * An array of variables passed to the template file.
 */
function _phptemplate_variables($hook, $vars = array()) {
    switch ($hook) {
// Send a new variable, $logged_in, to page.tpl.php to tell us if the
current
// user is logged in or out.
        case 'page':
// Get the currently logged in user.
            global $user;
// An anonymous user has a user ID of zero.
            if ($user->uid > 0) {
// The user is logged in.
                $vars['logged_in'] = TRUE;
            }
            else {
// The user is not logged in.
                $vars['logged_in'] = FALSE;
            }
            break;
        }
    }
    return $vars;
}

```

在前面的代码中，由于这样或者那样的原因，你为页面主题钩子或者 **page.tpl.php** 创建了一个新的变量。当用户已登录时，**\$logged_in** 为 **TRUE**；当匿名用户访问站点时，**\$logged_in** 为 **FALSE**。另一个通常需要设置变量的情况是，当一个作者创建一个评论时，检查他是不是被评论的文章的作者，这样就可以使用不同的样式了。下面是如何实现的：

```

function _phptemplate_variables($hook, $vars = array()) {
    switch ($hook) {
// Send a new variable, $logged_in, to page.tpl.php to tell us if the
current
// user is logged in or not.
        case 'page':
// Get the currently logged in user.

```

```

        global $user;
// An anonymous user has a user id of zero.
        if ($user->uid > 0) {
// The user is logged in.
            $vars['logged_in'] = TRUE;
        }
        else {
// The user is not logged in.
            $vars['logged_in'] = FALSE;
        }
        break;
    case 'comment':
// We load the node object to which the current comment is attached.
        $node = node_load($vars['comment']->nid);
// If the author of this comment is equal to the author of the node,
// we set a variable; then in our theme we can theme this comment
// differently so it stands out.
        $vars['author_comment'] = $vars['comment']->uid ==
$node->uid ? TRUE
            : FALSE;
        break;
    }
    return $vars;
}

```

现在在 `comment.tpl.php` 中你可以检查 `$author_comment` 的值，根据它的值的不同可设置不同的 **CSS** 类。

注意 在 `_phptemplate_variables()` 中你还可以修改的一个变量是 `$vars['template_file']`，它是 **Drupal** 将要调用的模板文件的名称。如果你在一个更复杂的条件下需要加载另外一个模板文件，你可以在这里修改这个变量。

第 8 章 Drupal 主题系统 (Drupal theme)

(4) 高级特性--3, 定义新的区块区域

定义新的区块区域

区域是在 Drupal 的主题中放置区块的地方。通过 Drupal 的后台管理接口 **Administer > Site building > Blocks**, 你可以将区块指定到区域中并管理它们。

尽管你可以创建任何你想要的区域, 但是在主题中默认的区域有左栏、右栏、页首、和页尾。一旦声明了区域以后, 就可以在你的页面模板文件中 (例如, **page.tpl.php**) 将其作为变量使用它们了。例如, 对于页首区域, 可以使用 `<?php print $header ?>`。通过在你的 **template.php** 文件中使用 **name-of-your-theme_regions()** 函数, 你可以创建其它的区域。

```
/*
 * Declare the available regions implemented by this engine.
 *
 * @return
 * An array of regions. Each array element takes the format:
 * variable_name => t('human readable name')
 */
function mytheme_regions() {
  return array(
    'left' => t('left sidebar'),
    'right' => t('right sidebar'),
    'content_top' => t('content top'),
    'content_bottom' => t('content bottom'),
    'header' => t('header'),
    'footer' => t('footer')
  );
}
```

为了在你的页面模板中输出“content_top”区域, 可使用 `<?php print $content_top ?>`。

Drupal 表单的主体化

修改 Drupal 表单的外观不像创建一个模板文件那样容易, 因为 Drupal 中的表单依赖于它们自己的 API。关于表单主体化的具体细节, 可参看第 10 章。

总结

读完本章以后，你应该可以

- 理解主题引擎和主题是什么
- 理解 Drupal 中 PHPTemplate 的工作原理
- 创建模板文件
- 覆写主题函数
- 操纵模板变量
- 为区块创建新的页面区域

Hezi

第 9 章 Drupal 区块 (Drupal block) (1)

区块是文本或者功能的一个片段，它通常位于一个网站的主内容区域之外，比如左栏，右栏，页首，页尾，等等。如果你曾经登录过一个 **Drupal** 站点，或者访问过一个 **Drupal** 的管理界面，那么你是用过区块。区块的访问权限和放置位置通过管理接口控制，它简化了开发者创建区块时的工作量，区块配置页面位于 **Administer > Site building > Blocks** (<http://example.com/?q=admin/build/block>)..

什么时候使用区块？

区块包含一个标题和一个描述，大多数用于代码片段和状态指示器，而不是用于包含各种信息的内容。因此，区块不是节点，它与节点有着不同的规则。节点有修改控制，完善的访问权限，附带评论的能力，**RSS** 种子和分类词语，它们通常用于一个站点的主要内容部分，区块拥有选项用于控制谁可以访问它们以及它们出现在站点的哪些页面。如果启用了 **throttle**（节流阀）模块，当访问量超过一定阈值时，一些不重要的区块，能够被自动关闭。区块列表页面如图 9-1 所示。

Block	Region	Weight	Throttle	Operations
Left sidebar				
Navigation	left sidebar	0	<input type="checkbox"/>	configure
User login	left sidebar	0	<input type="checkbox"/>	configure
Disabled				
Primary links	<none>	0	<input type="checkbox"/>	configure
Recent comments	<none>	0	<input type="checkbox"/>	configure
Syndicate	<none>	0	<input type="checkbox"/>	configure
Who's new	<none>	0	<input type="checkbox"/>	configure
Who's online	<none>	0	<input type="checkbox"/>	configure
<input type="button" value="Save blocks"/>				

图 9-1，当节流阀（throttle）模块启用时，区块列表页面显示了节流阀（throttle）选项

可以通过 **Drupal** 管理接口创建区块（称作定制区块），也可以通过区块 **API** 用代码创建区块（称作模块区块）。当你创建区块时，该选用哪种方法呢？简单的区块，比如一个用于订阅邮件列表的表单，与站点相关的文本都适用于定制区块。区块如果与你写的模块相关，或者包含了大量的 **php** 代码，则使用区块 **API** 通过模块创建。由于代码存放在数据库中比写在模块中更难以维护，所以尽量不要在定制区块中使用 **php** 代码，一个站点的编辑可能对此并不了解，它可能会偶然的不经意间将大量工作轻易的删掉。进一步讲，如果使用模块创

建区块过于笨重，而又不得不使用 **php** 代码时，仅仅在区块中调用一个定制函数，而将函数存放在别处即可。

注意，对于特定站点的区块和其它组件的一个常用经验，是创建一个站点特定的模块，将站点的定制函数放在里面

尽管区块 **API** 很简单，并且有一个单一钩子函数 **hook_block()** 驱动，但是在这一框架下，你可以做到你可能想不到的事情，区块可以展示你所想要的任何事情（这是因为，由于它们是使用 **php** 实现的，所以在功能上不受限制）。尽管如此，它们通常扮演一个对主内容进行支撑的角色。比如，你可以为每一个用户角色创建一个定制的导航区块或者你可以使用一个列出赞同评论的区块。

区块的配置选项

开发人员通常不需要担心区块的可视性，因为可以通过区块管理接口页面来处理大多数的情况。使用如图 9-2 所示的接口，在这些页面内，你可以控制下列选项：

区域位置：区域是区块在站点中放置的位置。通过主题来创建和显示区域而不是通过区块 **API**，没有指定区域的区块不被显示。

特定用户的可视化设定：管理员可以允许个人用户在他们的帐号设置中定制特定区块的可视性。用户可以通过单击链接“我的帐号”来修改区块的可视性。

特定角色的可视化设定：管理员可以选择区块仅对具有特定角色的用户可见。

特定页面的可视化设定：管理员可以选择区块仅对具有特定页面或者范围内的页面的用户可见或者隐藏，或者当你的 **php** 代码返回值为真的情况下可见。

▼ **Block specific settings**

Block title:

Override the default title for the block. Use `<none>` to display no title, or leave blank to use the default block title.

▼ **User specific visibility settings**

Custom visibility settings:

Users cannot control whether or not they see this block.

Show this block by default, but let individual users hide it.

Hide this block by default but let individual users show it.

Allow individual users to customize the visibility of this block in their account settings.

▼ **Role specific visibility settings**

Show block for specific roles:

anonymous user

authenticated user

Show this block only for the selected role(s). If you select no roles, the block will be visible to all users.

▼ **Page specific visibility settings**

Show block on specific pages:

Show on every page except the listed pages.

Show on only the listed pages.

Show if the following PHP code returns `TRUE` (PHP-mode, experts only).

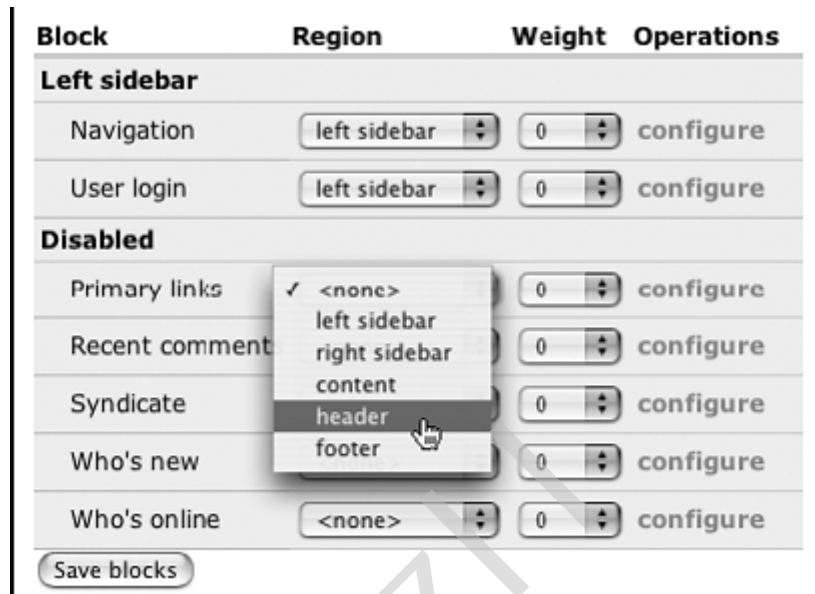
Pages:

Enter one page per line as Drupal paths. The '*' character is a wildcard. Example paths are `blog` for the blog page and `blog/*` for every personal blog. `<front>` is the front page. If the PHP-mode is chosen, enter PHP code between `<?php ?>`. Note that executing incorrect PHP-code can break your Drupal site.

图 9-2 管理接口页面区块的配置选项截图

区块位置

我们前面提到，在 **Drupal** 的区块管理页面给站点管理员提供了区块出现在哪里的选项。在同一页面的同一区域内，他们还可以选择区块的出现次序，如图 9-3 所示。区域是通过主题层使用方法 `hook_regions()` 来定义的，而不是通过区块 API，而且不同的区块可以有不同的区域，参看第 8 章可以获得关于创建区域的更多信息。



The screenshot shows the Drupal block management interface. It features a table with columns for Block, Region, Weight, and Operations. The table is divided into two sections: 'Left sidebar' and 'Disabled'. In the 'Disabled' section, a dropdown menu is open for the 'Primary links' block, showing options: '<none>', 'left sidebar', 'right sidebar', 'content', 'header', and 'footer'. The 'header' option is currently selected. A 'Save blocks' button is visible at the bottom left of the table.

Block	Region	Weight	Operations
Left sidebar			
Navigation	left sidebar	0	configure
User login	left sidebar	0	configure
Disabled			
Primary links	<none>	0	configure
Recent comment	left sidebar	0	configure
Syndicate	right sidebar	0	configure
Who's new	content	0	configure
Who's online	header	0	configure
	footer	0	configure

图 9-3 一个区块可以放置的区域依赖于站点主题提供的区域。

定义区块

可以通过在模块中使用钩子方法 `hook_block()` 来定义区块，而且一个模块可以在同意钩子方法中实现多个区块。一旦定义好了一个区块，那么它将会出现在区块管理页面。另外，站点管理员可以通过后台接口手工的定义定制区块。在本节，我们主要关注通过代码创建区块。让我们看一下区块所涉及到的数据库表元数据，如图 9-4 所示。

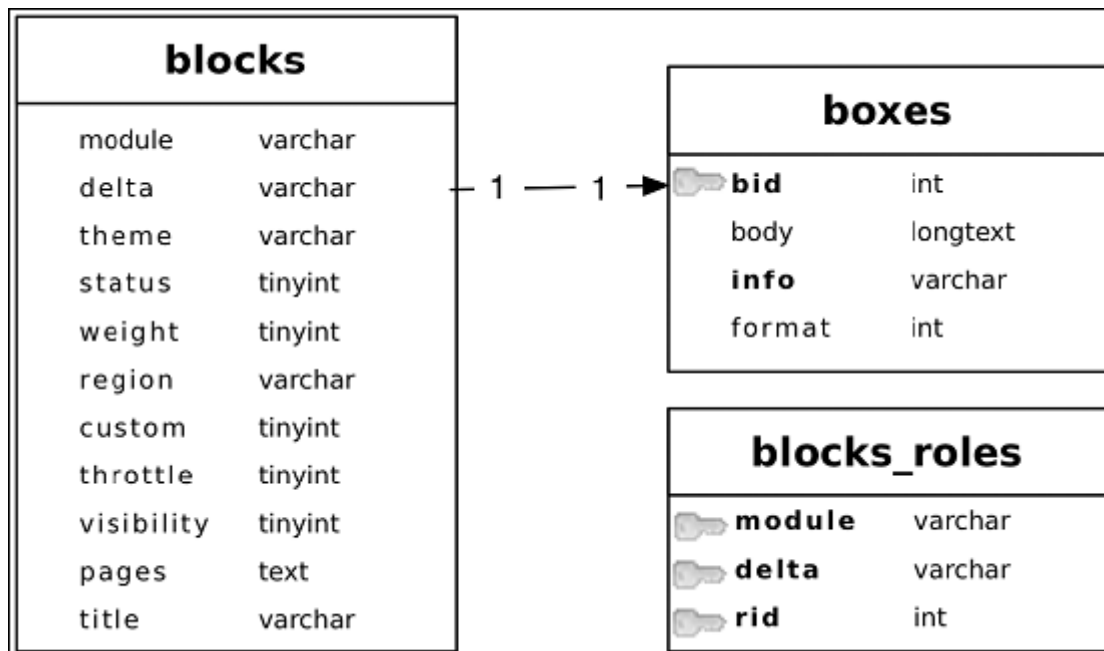


图 9-4 区块的数据库元数据

每个区块的通用属性存储在表 **blocks** 里面，通过后台接口创建的区块的其它数据，比如它们的内容还有输入格式类型存放在表 **boxes** 里面。最后，表 **blocks_roles** 存放每一个区块的基于角色的访问控制权。以下属性在表 **blocks** 里面定义：

module: 这一字段存放的是定义区块的模块的名称。比如，用户登录区块在用户模块中定义，等等。对于通过后台接口创建的区块，在这里都使用 **block**（区块）模块，它们由区块模块创建。

delta: 由于一个模块可以在钩子方法 **hook_block()** 中定义多个区块，那么 **delta** 存放了每个区块的键值。它们在钩子方法范围内是唯一的，但对于整个站点的所有区块则不一定唯一。通过后台接口创建的区块的 **delta** 字段，同时作为表 **blocks** 的 **bid** 字段的主键。**delta** 可以是整数，也可以是字符串。

theme: 由于区块可以为多个主题定义，**Drupal** 需要存放启用该区块的主题的名称。启用该区块的每一个主题在数据库中都有一行相应的记录。配置选项在主题间不共享。

status: 它用来标识区块是否被启用。**1** 意味着启用，**0** 意味着禁用。如果一个区块。没有为其分配一个区域，那么 **Drupal** 将会将其状态设置为 **0**。

throttle: 当 **throttle**（节流阀）模块启用时，该字段用于追踪哪些区块被节流阀控制。**0** 意味着禁用，**1** 意味着启用。节流阀模块能够自动的探测访问流量，当流量达到一定的阈值时，能临时的禁用一些区块（参看第 22 章，更多详情）。

理解区块的主题化

在一个页面请求周期内，主题系统向区块系统请求以返回每一区域内的一系列区块，当为页面模板文件(通常为 `page.tpl.php`)变量赋值时，完成这一步骤。为了聚合左栏和右栏的主体化的区块，**Drupal** 执行以下代码：

```
$sidebar_left = theme('blocks', 'left');
$sidebar_right = theme('blocks', 'right');
// And any other regions exposed by hook_regions().
```

你可能还记得第 8 章里面的 `theme("block")`，它用于调用方法 `theme_blocks()`，下面展示了 `theme_blocks()` 做了什么：

```
function theme_blocks($region) {
  $output = '';
  if ($list = block_list($region)) {
    foreach ($list as $key => $block) {
      $output .= theme('block', $block);
    }
  }
  return $output;
}
```

在前面的代码中，我们为每个给定区域对其区块进行迭代，并为每个区块执行一个主题函数。最后，我们为调用此函数的代码返回该区域内所有主体化的区块。

第 9 章 Drupal 区块(Drupal block)(2) 区块钩子方法

使用区块钩子

在用代码创建区块时，我们在钩子方法 `hook_block()` 中处理所有的逻辑。通过这个钩子，你可以创建一个单独的区块或者一组区块。任何模块都可以通过其钩子方法创建区块。让我们看下该函数的方法签名：

```
function hook_block($op = 'list', $delta = 0, $edit = array())
```

参数列表

区块钩子中使用的参数在接下来的部分讨论。

\$op: 这一参数用于定义一个区块在传递过来时所处的阶段。通过参数 `$op` 来定义一个操作阶段，这一模型在 **Drupal** 中是常用的框架---例如 `hook_nodeapi()` 和 `hook_user()` 中都用到。

\$op 的可能值如下：

list: 返回包含模块定义的所有区块的数组。数组的键值是 **delta**(在本模块定义的区块范围内，它是唯一标识符)。那么每个数据元素的值是一个提供了区块重要数据的一个数组，**list** 的可能值和默认值，如下：

info: 一个可国际化的字符串（例如，通过 `t()` 包装），为站点管理员提供了一个合适的描述。

status: 区块启用为 **True**，禁用为 **FALSE**，默认为 **FALSE**。

region: 默认区域为左栏。

weight: 它控制着区块在它的区域内的放置次序。重量越小，位置越靠前，水平方向是靠左方，垂直方向是靠上方。重量越大，越靠后。默认值为 **0**。

pages: 定义节点所在的默认页面。默认是一个空字符串。**Pages** 的值是通过换行分隔的 **Drupal** 路径，***** 为通配符。例如，路径 **blog** 为日志首页，而 **blog/*** 则为每个个人日志页面。**<front>** 代表首页。

custom: **TRUE** 代表着这个区块是通过后台接口创建的，而 **FALSE** 则代表着它是通过模块实现的区块。

title: 区块的默认标题。

configure: 返回一个用于区块特定设定的表单字段的数组。它整合了默认后台接口的表单数组，从而使你能够扩展区块的配置方式。如果你实现了它，你同时还需要实现保存(**save**)

操作**\$op**(参看下面)。

save: 当配置表单提交时被调用。当你的模块可以保存你在配置操作**\$op**中收集的定制化的区块的配置信息时，使用该操作。你想保存的操作包含在变量**\$edit**中。

view: 区块被显示。返回一个包含区块标题和内容的数组。

\$delta: 这是返回的区块 ID。这里你可以使用一个整数或者一个字符串，注意当操作**\$op**的状态为 **list** 时，**\$delta** 被忽略。

\$edit: 当操作为保存时，**\$edit** 包含了从区块配置表单提交过来的表单数据。

Hezi

第 9 章 Drupal 区块 (Drupal block) (3)

创建区块

创建一个区块:

在本例中, 你将创建两个区块, 它们使得内容修改更易于管理。首先, 你将创建一个区块用于列出等待批准的评论, 然后你将创建一个区块以列出未发布的节点。两个区块都提供了用于修改相应内容的编辑表单的链接。

让我们创建一个名为 `approval.module` 的模块。它将包含我们的区块代码。在路径 `sites/all/modules/custom` 下面创建一个名为 `approval` 的文件夹 (如果 `modules` 和 `custom` 不存在的话, 你需要创建它们)。

接下来, 向文件中添加 `approval.info` 文件:

```
; $Id$
name = Approval
description = Blocks for facilitating pending content workflow.
version = "$name$"
```

然后, 在添加 `approval.module` 文件:

```
<?php
// $Id$
/**
 * @file
 * Implements various blocks to improve pending content workflow.
 */
```

当你创建好这些文件后, 在 **Administer > Site building > Modules** 下面启用该模块。你将继续使用 `approval.module`, 所以不要关闭文本编辑器。

接下来我们添加区块钩子方法并实现 `list` 操作, 我们的区块将出现在区块管理页面的区块列表中 (参看图 9-5)。

Block	Region	Weight	Operations
Left sidebar			
Navigation	left sidebar	0	configure
User login	left sidebar	0	configure
Disabled			
Pending comments	<none>	0	configure
Primary links	<none>	0	configure

图 9-5 在区块列表中，你可以看到你创建的区块了。

注意数组的键 **info** 不是区块启用时所展示给用户的区块标题。而是一个仅出现在管理员可以配置的区块列表页面的描述。你将在接下来的查看 (**view**) 情景中实现真正的区块标题。首先，你需要创建额外的配置选项，为了实现这一点，使用下面的代码来实现配置 (**configure**) 情景，你创建了一个新的表单字段，当你点击区块列表页面区块右边的配置链接时，即可看到它，如图 9-6 所示：

Home » Administer » Site building » Blocks

'Pending comments' block

▼ Block specific settings

Block title:

Override the default title for the block. Use <none> to display no title, or leave blank to use the default block title.

Number of pending comments to display:

图 9-6 带有区块定制字段的区块配置表单

当如图 9-6 所示的区块配置表单被提交后，它将触发下一步操作，这里是保存，你将使用它来保存表单字段值。

```
function approval_block($op = 'list', $delta = 0, $edit = array()) {
  switch ($op) {
    case 'list':
      $blocks[0]['info'] = t('Pending comments');
      return $blocks;
    case 'configure':
      $form['approval_block_num_posts'] = array(
        '#type' => 'textfield',
        '#title' => t('Number of pending comments to display'),
      );
  }
}
```

```

    '#default_value' => variable_get('approval_block_num_posts', 5),
  );
  return $form;
case 'save':
  variable_set('approval_block_num_posts',
    (int) $edit['approval_block_num_posts']);
  break;
}
}

```

通过使用 **Drupal** 自带的变量系统 `variable_set()`，你将区块所展示的评论的数目保存了下来。注意这里使用了类型转换，将其转换为整数，目的是对数据进行清洁检查，最后添加查看操作，当区块显示时，返回一个待定评论的列表。

```

function approval_block($op = 'list', $delta = 0, $edit = array()) {
  switch ($op) {
    case 'list':
      $blocks[0]['info'] = t('Pending comments');
      return $blocks;

    case 'configure':
      $form['approval_block_num_posts'] = array(
        '#type' => 'textfield',
        '#title' => t('Number of pending comments to display'),
        '#default_value' => variable_get('approval_block_num_posts', 5),
      );
      return $form;
    case 'save':
      variable_set('approval_block_num_posts', (int)
        $edit['approval_block_num_posts']);
      break;
    case 'view':
      if (user_access('administer comments')) {
        // Retrieve the number of pending comments to display that
        // we saved earlier in the 'save' op, defaulting to 5.
        $num_posts = variable_get('approval_block_num_posts', 5);
        // Query the database for unpublished comments.
        $result = db_query_range('SELECT c.* FROM {comments} c WHERE
          c.status = %d ORDER BY c.timestamp', COMMENT_NOT_PUBLISHED, 0,
          $num_posts);
        // Preserve our current location so user can return after editing.
        $destination = drupal_get_destination());

```

```

    $items = array();
    while ($comment = db_fetch_object($result)) {
        $items[] = l($comment->subject, 'node/'. $comment->nid, array(),
            NULL, 'comment-'. $comment->cid). ' '.
            l(t('[edit]'), 'comment/edit/'. $comment->cid, array(),
                $destination);
    }
    $block['subject'] = t('Pending comments');
// We theme our array of links as an unordered list.
    $block['content'] = theme('item_list', $items);
}
return $block;
}
}

```

这里我们通过对数据库进行查询来获得待定的评论，将评论的标题展示为链接，同时为每一个评论追加一个编辑链接，如图 9-7 所示

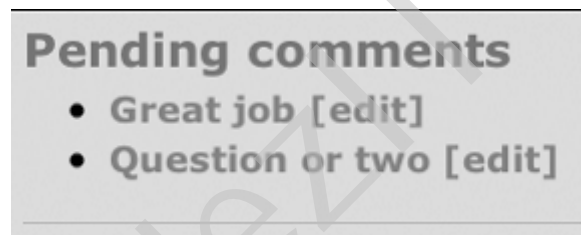


图 9-7 “待定评论”列表区块在它启用后的情况。它展示了两个待定评论

在前面的代码中，注意我们是如何使用方法 `drupal_get_destination()` 的，这个方法将记住在你提交表单以前你所在的页面，所以当你更新一个评论以后（或者发布，或者删除），它将自动重定向到你原来所在的页面。

使用下面的代码，设置了区块的标题：

```
$block['subject'] = t('Pending comments');
```

现在待定区块已经完成，让我们在 `approval_block()` 钩子函数中定义另一个区块---它列出了所有未发布的节点，并提供了指向它们的编辑页面的链接。

```

function approval_block($op = 'list', $delta = 0, $edit = array()) {
    switch ($op) {
        case 'list':
            $blocks[0]['info'] = t('Pending comments');
            $blocks[1]['info'] = t('Unpublished nodes');
            return $blocks;
    }
}

```

```
}  
}
```

注意这里是如何为每一个区块分配一个键值的（`$blocks[0]`, `$blocks[1]`, ... `$blocks[n]`）。区块模块将最终使用这些键值作为 `$delta` 参数。这里我们将“待定评论”区块的 `$delta ID` 定义为 0, “未发布节点”区块的 `$delta ID` 定义为 1。在这里也可以使用“待定”和“未发布”作为键值。有程序员的习惯决定使用哪种键值，而键值不一定是数字形式。

下面是完整的例子，我们的新区块如图 9-8 所示：

```
function approval_block($op = 'list', $delta = 0, $edit = array()) {  
  switch ($op) {  
    case 'list':  
      $blocks[0]['info'] = t('Pending comments');  
      $blocks[1]['info'] = t('Unpublished nodes');  
      return $blocks;  
    case 'configure':  
      // Only in block 0 (the Pending comments block) can one  
      // set the number of comments to display.  
      if ($delta == 0) {  
        $form['approval_block_num_posts'] = array(  
          '#type' => 'textfield',  
          '#title' => t('Number of pending comments to display'),  
          '#default_value' => variable_get('approval_block_num_posts', 5),  
        );  
      }  
      return $form;  
    case 'save':  
      if ($delta == 0) {  
        variable_set('approval_block_num_posts', (int)  
          $edit['approval_block_num_posts']);  
      }  
      break;  
    case 'view':  
      if ($delta == 0 && user_access('administer comments')) {  
        // Retrieve the number of pending comments to display that  
        // we saved earlier in the 'save' op, defaulting to 5.  
        $num_posts = variable_get('approval_block_num_posts', 5);  
        // Query the database for unpublished comments.  
        $result = db_query_range('SELECT c.* FROM {comments} c WHERE  
c.status = %d  
ORDER BY c.timestamp', COMMENT_NOT_PUBLISHED, 0, $num_posts);
```

```

$destination = drupal_get_destination();
$items = array();
while ($comment = db_fetch_object($result)) {
  $items[] = l($comment->subject, 'node/'. $comment->nid, array(),
    NULL, 'comment-'. $comment->cid). ' ';
  l(t('[edit]'), 'comment/edit/'. $comment->cid, array(),
    $destination);
}
$block['subject'] = t('Pending Comments');
// We theme our array of links as an unordered list.
$block['content'] = theme('item_list', $items);
}
elseif ($delta == 1 && user_access('administer nodes')) {
// Query the database for the 5 most recent unpublished nodes.
// Unpublished nodes have their status column set to 0.
$result = db_query_range('SELECT title, nid FROM {node} WHERE
status = 0 ORDER BY changed DESC', 0, 5);
$destination = drupal_get_destination();
while ($node = db_fetch_object($result)) {
  $items[] = l($node->title, 'node/'. $node->nid). ' ';
  l(t('[edit]'), 'node/'. $node->nid .'/edit', array(),
    $destination);
}
$block['subject'] = t('Unpublished nodes');
// We theme our array of links as an unordered list.
$block['content'] = theme('item_list', $items);
}
return $block;
}
}

```

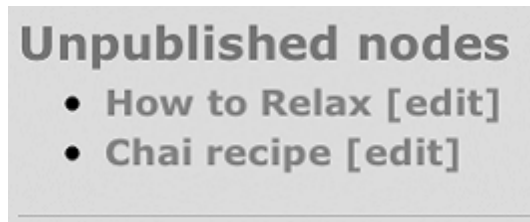


图 9-8， 区块启用后，列出了未发布节点列表

由于你有多个区块，你查看操作下，你使用了 `if...elseif` 来构建。在每一种情况下，你检查被查看区块的 `$delta` 以决定是否该运行该段代码。在脚本形式里，它看起来如下所示：

```

if ($delta == 0) {
// Do something to block 0
}
elseif ($delta == 1) {
// Do something to block 1
}
elseif ($delta == 2) {
// Do something to block 2
}
}

```

在“未发布节点”区块启用后，区块的最终结果如图 9-8 所示。

额外例子：添加一个“待定用户”区块

如果你想扩展 `approval.module`，你可以添加另一个区块，以展示等待站点管理员批准的用户帐号。这将作为作业留给读者自己动手将其放到 `approval.module` 模块中去。这里展示了一个在假定的 `userapproval.module` 模块中的类似的例子：

```

function userapproval_block($op = 'list', $delta = 0, $edit = array()) {
  switch ($op) {
    case 'list':
      $blocks[0]['info'] = t('Pending users');
      return $blocks;
    case 'view':
      if (user_access('administer users')) {
        $result = db_query_range('SELECT uid, name, created FROM {users}
WHERE uid != 0 AND status = 0 ORDER BY created DESC', 0, 5);
        $destination = drupal_get_destination();
        // Defensive coding: we use $u instead of $user to avoid potential namespace
        // collision with global $user variable should this code be added to later.
        while ($u = db_fetch_object($result)) {
          $items[] = theme('username', $u). ' '.
            l('[edit]', 'user/'. $u->uid. '/edit', array(), $destination);
        }
        $block['subject'] = t('Pending users');
        $block['content'] = theme('item_list', $items);
      }
      return $block;
    }
  }
}

```

在安装模块时，启用一个区块

有时，你想在安装模块时，将一个区块自动展示出来。这非常直接，通过查询语句直接将区块的设置信息直接插入到 **blocks** 表中即可。查询放在钩子方法 **hook_install()** 中，钩子方法位于模块的 **.install** 文件中。下面是一个例子，当 **Drupal** 被安装时，用户模块启用了用户区块（参看 **modules/system/system.install**）：

```
db_query("INSERT INTO {blocks} (module, delta, theme, status) VALUES ('user', 0, '%s', 1)", variable_get('theme_default', 'garland'));
```

上面的数据库查询语句将区块插入到了区块表中，并将它的状态设置为 **1**，所以它被启用了。

区块可视化例子

在区块管理接口里面，你可以在区块配置页面的“页面可视化配置”里面加入 **php** 代码片段。当一个页面被构建时，**Drupal** 将运行 **php** 代码片段来决定区块是否被显示。一些常用的代码片段例子如下所示。每一段代码返回 **TRUE** 或 **FALSE** 来指示区块对于特定请求是否可见。

将区块仅展示给登录用户

当 **\$user->id** 不为 **0** 时，返回 **TRUE**

```
<?php
global $user;
return (bool) $user->uid;
?>
```

将区块仅展示给匿名用户

当 **\$user->id** 为 **0** 时，返回 **TRUE**

```
<?php
global $user;
return !(bool) $user->uid;
?>
```

总结

在本章，你学到了以下几点：

- 区块是什么以及它们与节点的区别
- 区块的可视化和位置配置是如何设定的
- 如何定义一个或多个区块
- 如何在默认情况下启用区块

第 14 章 在 Drupal 中使用分类 (Drupal taxonomy) (1)

第 14 章 Drupal 的分类法 (Taxonomy)

分类是对事物的划分归类。**Drupal** 自带了一个分类模块 (**Taxonomy module**)，它允许你对节点 (就是所谓的“事物”) 进行分类。在本章中，你将看到 **Drupal** 支持的分类的不同形式。你也将看到数据是如何存储的，如何对分类 (**taxonomy**) 数据库表写出查询语句，以及在你自己的模块中使用这些语句。最后，你将会看到，当分类 (**Taxonomy**) 改变时，你的模块如何受到改变的通知，还有我们将介绍一些常用的分类 (**Taxonomy**) 相关的任务。

什么是分类

分类涉及到对事物进行归类。你将会在 **Administer > Content Management > Categories** 下面看到 **Drupal** 的对分类的支持 (如果在这里没有的话，请确认启用了分类模块)。当涉及到 **Drupal** 的分类系统时，用词的准确性是非常重要的。让我们看一下你会遇到的常用词。

词语 (terms)

词语是即将应用到节点上实际标签。例如，假定你有一个包含产品评论的网站。你可以在每一个评论上使用词语“坏的”，“可以”，“优秀”来进行标记。词语有时也称为标签，将一个词语指定到一个对象 (比如说一个产品评论节点) 上的行为称之为标签化。

抽象层次 (A level of Abstraction)

当你查看数据结构时，你会立即发现，**Drupal** 对所有你输入的词语添加一个层次上的抽象，在内部对它们的引用是通过 **ID** 来完成的，而不是通过名字。例如，如果你在前面输入了一个词语，但是你的经理觉得单词“**Poor**”比“**Bad**”更好一些，这时没有任何难的。你简单的编辑这个 **id** 为 **1** 词语，将它从“**Bad**”改为“**Poor**”。在 **Drupal** 内部一切工作正常，这是因为 **Drupal** 在内部把它当作了 **ID** 为 **1** 的词语来进行使用。

同义词 (Synonyms)

当你定义一个词语时，你可以输入该词的同义词；一个同义词是一个具有同样含义的单词。**Drupal** 包含的分类功能允许你输入同义词，并提供了数据库表对它们进行存储，以及一些有用的函数比如 **taxonomy_get_synonyms(\$tid)** and **taxonomy_get_synonym_root(\$synonym)**，但是对这些函数的用户接口的实现留给

了贡献模块，比如术语表模块（**glossary module**）
（<http://drupal.org/project/glossary>）。

词汇表（vocabularies）

一个词汇表包含了一组词语。**Drupal** 允许你将一个词汇表与一个或多个节点类型相联系。当跨节点类型进行分类时，这种弱联系非常有用。例如，如果你有一个站点，允许用户可以提交关于旅游的故事和图片；这将非常容易的让你看到标记为 **Belgium** 的所有故事和图片。词汇表接口编辑页面如图 14-1 所示。

必须的词汇表

词汇表可以是必须的，也可以不必须。如果一个词汇表是必须的，那么用户在提交节点表单以前必须为节点选择一个词语。如果不是必须的，那么用户提交表单时，可以使用默认词语 **none**。

受控的词汇表

当一个词汇表有一个有限的词语时（也就是说，用户不可以添加新的词语），被称为受控词汇表。对于一个受控词汇表，词汇一般都存在于下拉选择按钮中。当然，管理员，或者拥有管理分类权限的用户可以添加，删除，或者修改词语。

自由化标签

自由化标签与受控词汇表相对立。当用户提交一个节点时，可以输入他们自己的词语。如果词语还不是词汇表的一部份的话，它将被添加。当启用自由化标签时，词汇表的用户接口将会以一个文本输入框出现（使用 **Javascript** 自动完成），而不是受控词汇表所使用的下拉选择按钮。

单独 VS 多个词语

Drupal 允许你设置对于一个给定的节点，是使用单独的一个词语还是使用多个词语进行标签化。选择后者，将使得用户接口的节点提交表单中的单选下拉按钮变为多选下拉按钮。

提示：本选项仅适用于受控词汇表，对于自由化标签不使用

相关词语

如果一个词汇表允许相关词语，那么当你定义一个新的词语的时候，将会出现一个多选下拉菜单，这样你就可以从已经存在的词语中间选出相关的词语。

重量

每一个词汇表都有一个重量，从-10到10（如图14-1所示）。这用来控制用户的节点提交表单中的词汇表的布局。词汇表的重量越轻，对应的分类字段集位置越靠上边。重量越大，位置越靠下面。

每一个词语也都有一个重量。词语的在用户的下拉选择按钮中的位置决定于它的重量。这一优先级同样展现在 **Administer > Content management > Categories > List terms.**

Hezi

Vocabulary name: *

The name for this vocabulary. Example: "Topic".

Description:

Description of the vocabulary; can be used by modules.

Help text:

Instructions to present to the user when choosing a term.

Types: *

- Forum topic
- Page
- Story

A list of node types you want to associate with this vocabulary.

Hierarchy:

- Disabled
- Single
- Multiple

Allows a tree-like hierarchy between terms of this vocabulary.

- Related terms

Allows related terms in this vocabulary.

- Free tagging

Content is categorized by typing terms instead of choosing from a list.

- Multiple select

Allows nodes to have more than one term from this vocabulary (always true for free tagging).

- Required

If enabled, every node **must** have at least one term in this vocabulary.

Weight:

In listings, the heavier vocabularies will sink and the lighter vocabularies will be positioned nearer the top.

图 14-1 添加词汇表的表单

分类的种类

分类有多个种类。最简单的仅有一列词语，而最复杂的则有复杂的层次结构关系。另外，词语可以有同义词或者与它相关的词语。让我们从最简单的开始。

扁平结构

一个仅包含了一列词语的词汇表是非常简单的。表 14-1 展示了我们如何在一个叫做编程语言的简单的扁平的词汇表中对编程语言进行分类。

表 14-1 词汇表中的简单词语

Term ID	Term Name
1	C
2	C++
3	Cobol

层次结构

现在，让我们引入一个概念“层次”，在这里每一个词语都与另一个存在一种关系。

表 14-2 词汇表中词语的层次化结构（子词语位于父亲的下面并缩进了）

Term ID	Term Name
1	Object-Oriented
2	C++
3	Smalltalk
4	Procedural
5	C
6	Cobol

图 14-2 明确的展示了这种层次关系。在本例中，Procedural 是父亲而 Cobol 是孩子。注意，每一个词语都有一个 ID，这与它是父亲还是孩子没有关系。

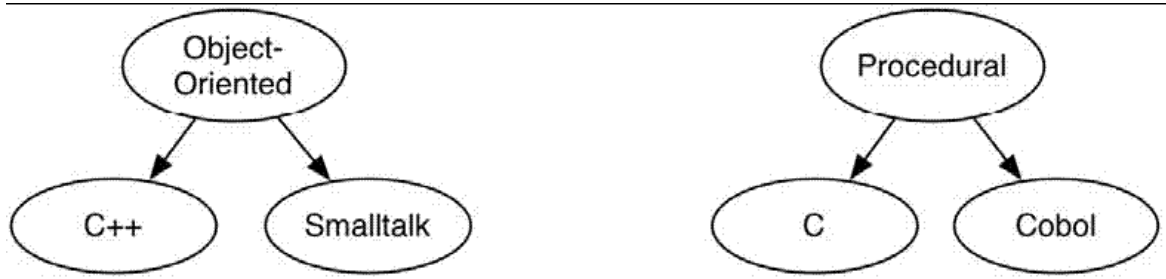


图 14-2 一个层次化的词汇表在词语之间具有父子关系

多层次关系

一个词汇表可以有多层次关系而不仅仅是单层次关系。这简单的意味着一个词语可以有多个的父亲。例如，假定你往编程语言词汇表中添加 **PHP**。**PHP** 可以使用过程化的方式编码，但是在最近的版本中，面向对象的编程能力也被引入了。我们把它放到面向对象还是过程化的下面？在多层次化关系中，你可以把它放在两者的下面，如图 14-3 所示

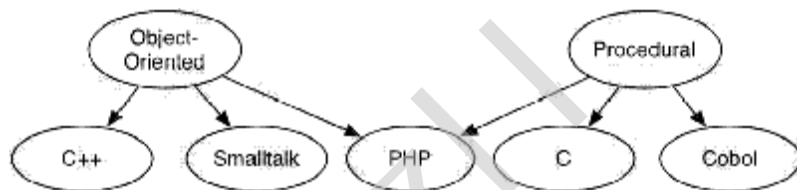


图 14-3 一个多层次化的词汇表中，词语可以有多个的父亲

在建立网站的计划阶段，你值得花费大量的时间用来好好的认真考虑一下分类的使用情况，以决定你需要使用哪些分类。

第 15 章 Drupal 缓存 (cache) (1)

缓存

为动态网站建造页面需要对数据库进行大量的数据读取，以获取诸如保存的内容、站点设置、当前用户等等诸多信息。将这些耗费资源的操作结果保存起来以备后用，这是从应用层提高一个反应缓慢站点的最简单的途径之一。**Drupal** 内置的缓存 **API** 对大部分核心数据进行了自动缓存，并为 **Drupal** 开发者提供了一组工具以进行量身定做。

缓存是如何工作的

模块开发者可以将他们要缓存的数据保存到 **Drupal** 数据库中专门用于缓存的数据库表中，或者它们也可以为缓存的存储创建一个新的数据库表。当下次用到这些缓存过的信息时，可以通过一个简单的查询快速的获取这些信息，而不是在使用笨重的数据操作了。

在你的模块中，你可以放置缓存信息的默认数据库表是 **cache**。当缓存信息的数据量不大时，最好使用该表。你过你想为每个节点、菜单、用户缓存信息，你将需要为你的模块创建独有的缓存数据库表，这样就可以减小 **Drupal** 的 **cache** 表的大小，从而提高新能。当要为你的模块创建一个新的缓存数据库表时，该表的数据结构一定要与 **cache** 表完全相同，不同的仅仅是表名。为了一致性，在表明前面加上前缀 **cache_**是个不错的注意。让我们看一下 **cache** 表的结构；参看表格 15-1。

注意；当为你的模块创建一个新的缓存数据库表时，该表的数据结构一定要与 **cache** 表完全相同

表 15-1 表 **cache** 的元数据

Field	Type	Null	Index
cid	varchar(255)	NO	PRIMARY
data	longblob	YES	
expire	int	NO	MULTIPLE
created	int	NO	
headers	text	YES	

列 **cid** 存放的是用于快速取回缓存信息的主键。在 **Drupal** 核心中使用缓存 **ID** 的例子有，用于页面缓存的页面的 **URL**（比如，<http://example.com/?q=taxonomy/term/1>），用于用户菜单缓存的用户 **ID** 和本地区域（比如，**1:en**），或者甚至可以使用规则的字符串（比如，表 **variables** 的内

容的缓存，它将缓存 ID 设置为了 **variables**（译者注，这里将的缓存是将表 **variables** 的所有内容缓存在了一起））。

列 **data** 用于存放你想要缓存的信息。对于复杂的数据类型比如数组或者对象，需要使用 PHP 的 **serialize()** 函数将其序列化后从而将其数据结构也保存到数据库中。当然，这意味着当你从数据库中取回这些缓存数据时，你需要使用 PHP 的 **unserialize()** 函数对其反序列化从而得到相应的数组或者对象。

列 **expire** 采用下面的 3 种值：

CACHE_PERMANENT：这意味只有当针对给定永久项目的缓存 ID 使用 **cache_clear_all()**，才能删除该项目。

CACHE_TEMPORARY：这意味着当下一次不带参数调用 **cache_clear_all()** 时，就会删除该项目，而没有最小时间限制。标记为 **CACHE_PERMANENT** 的项目此时不被删除。

一个 **Unix** 时间戳：该时间戳指的是该项目的最小存在时间，在最小时间内，就不能删除它，当过了这个时间，它就和标记为 **CACHE_TEMPORARY** 的项目性质一样了。

列 **created** 是缓存项目创建的时间，它不用来决定缓存的存在周期。

列 **headers** 用来存放 HTTP 的回应头部，在缓存的数据是整个请求的 **Drupal** 页面时使用。大多数时候，你不使用该列，这是因为你要缓存的是页面的一部份而不是整个页面，也就是说要缓存的数据不依赖于头部信息。记住，尽管如此，你定制的缓存数据库表的结构仍然需要和默认的 **cache** 表完全相同，所以尽管不使用它也要保存它。

知道什么时候使用缓存

要记住一点，使用缓存是要考虑平衡的。对大量的数据进行缓存可以对性能有很大提高，这是不假，但是这是有前提的，前提就是缓存的数据需要在接下来被多次使用。这就是为什么页面缓存仅用于匿名用户：已注册的用户看到的通常是页面的定制版本，这样缓存效果就不太明显了。对小量的数据进行缓存（比如当天的流行文章列表）尽管对你的网站性能的提高不大，但也会有所改善的。

另外要的是，要缓存的数据最好不要经常修改。比如，每周最佳故事列表，就比较适用。如果对于一个繁忙的论坛，缓存最近 5 个评论，此时效果就不太明显，因为要缓存的数据很快就会过期，在它需要被更新以前，很少有用户使用到它。在最坏的情况，一个坏的缓存策略（对改动过于频繁的数据进行缓存）可能会增加网站负担而不是提高性能。

在 Drupal 核心中是如何使用缓存的

Drupal 自带了 4 个缓存数据库表：**cache_menu**,**cache_filter**,**cache**,**cache_page**。

`cache_menu` 为每个用户 ID 存放导航菜单的缓存副本；`cache_filter` 为每个被过滤器系统解析过的每个节点的内容存储缓存副本；`cache` 存储模块设置，当你调用 `cache_set()` 时，它是默认的缓存表；`cache_page` 存储匿名页面的缓存副本。我们在接下来的部分中会逐一的讲解每一个缓存。需要注意的是，在后台 **Administer > Site configuration > Performance** 中的页面缓存设置仅用于页面缓存，对于 **Drupal** 中的其它缓存不起作用。换句话说，过滤器，菜单，模块设置总是被缓存的。

菜单系统

所有由菜单模块创建的菜单都被缓存，无论 **Drupal** 的页面缓存是否被启用。关于菜单的例子有 **Drupal** 的一级和二级链接菜单，以及用户导航区块。菜单基于单个用户、单个区域进行缓存。关于菜单系统的更多信息参看第 4 章。

过滤的输入格式

当创建或者编辑一个节点时，它的内容将通过与其输入格式相关的各种过滤器。例如，**HTML** 过滤器格式将换行装化为 **HTML** `<p>` 和 `
` 标签，同时过滤掉恶意的 **HTML**。如果每次简单的查看该节点时都进行过滤，那么这是很费资源的。因此，只有在创建或者编辑节点内容完以后才应用过滤器（其他时候不用）并将过滤后的内容缓存到数据库中，这同样与 **Drupal** 的页面缓存是否启用没有关系。关于输入格式的更多信息参看第 11 章。

提示 当你使用后台管理接口改变节点摘要（**node teaser**）的默认长度时，只有当你重新保存每个节点以后才会生效，原因就是过滤器缓存。解决该问题的简便方法是清空表 `cache_filter`，这样每个节点将被过滤器重新解析。

用于后台管理的变量和模块设置

Drupal 将大多数的管理设置存储在表 `variables` 里，并将该表的所有数据缓存到表 `cache` 里以加快对配置数据的查看速度。这些变量的例子包括你站点的名称、评论和用户的设置、以及文件目录的位置。所有的这些变量被缓存到表 `cache` 中的一行记录当中，这样就可以快速的取回它们，而不是在需要每一个变量时都进行一次数据库查询。它们作为 **PHP** 数组进行存储，所以需要缓存的数据进行序列化从而保存它的结构。任何使用 `variable_set()` 和 `variable_get()` 作为设置器（**setter**）和读取器（**getter**）函数的变量都将以这种方式存储和缓存。

第 15 章 Drupal 缓存 (cache) (2) 页面缓存

页面

我们前面讨论了很多，都是关于对站点的耗费资源的成分进行缓存的，但是 **Drupal** 最有效的缓存优化是对整个页面视图进行缓存。对于匿名用户，很容易做到这一点，这是因为所有页面对于所有匿名用户都是相同的。然而，对于登录用户，每个页面都是针对用户量身定做的。因此需要采用不同的缓存策略来处理这种情况。

对于匿名用户，**Drupal** 可以使用一个简单的查询来取回缓存的页面内容，当然它需要一些其它查询以加载 **Drupal** 本身。对于匿名用户页面缓存，有两种缓存策略供你选择：普通模式和激进模式。当然你也可以禁用页面缓存。这些设置可以在 **Drupal** 后台管理接口 **Administer > Site configuration > Performance** 中找到。在接下来的部分中，让我们看一下每种设置。

禁用模式

这将完全禁用页面缓存。通常在开发网站的时候使用。一般情况下，你需要启用页面缓存。

注意 即使禁用了页面缓存，**Drupal** 仍对用户菜单、过滤后的内容、系统变量进行缓存。这些部件级别的缓存不能被禁用。

普通模式

与完全不使用缓存相比，普通模式对性能有巨大提升，因此它是对一个运行缓慢的 **Drupal** 站点提升速度的最简单方式之一。让我们仔细的看一下，当缓存系统的普通模式启用时，请求的生命周期。

为了理解页面缓存的普通模式，你首先需要了解 **Drupal** 的引导指令流程。引导指令流程由在 **Drupal** 称之为阶段的小的独立的步骤组成。在安装和更新流程中，**Drupal** 就用到了将引导指令系统划分为多个阶段的优点，在这些流程中只需要加载需要的代码即可。对当前讨论，我们需要知道的是，系统在提供缓存页面时，只需要加载他所需要的代码和数据库链接即可。

图 15-1 详细展示了为匿名用户请求提供缓存页面的流程。

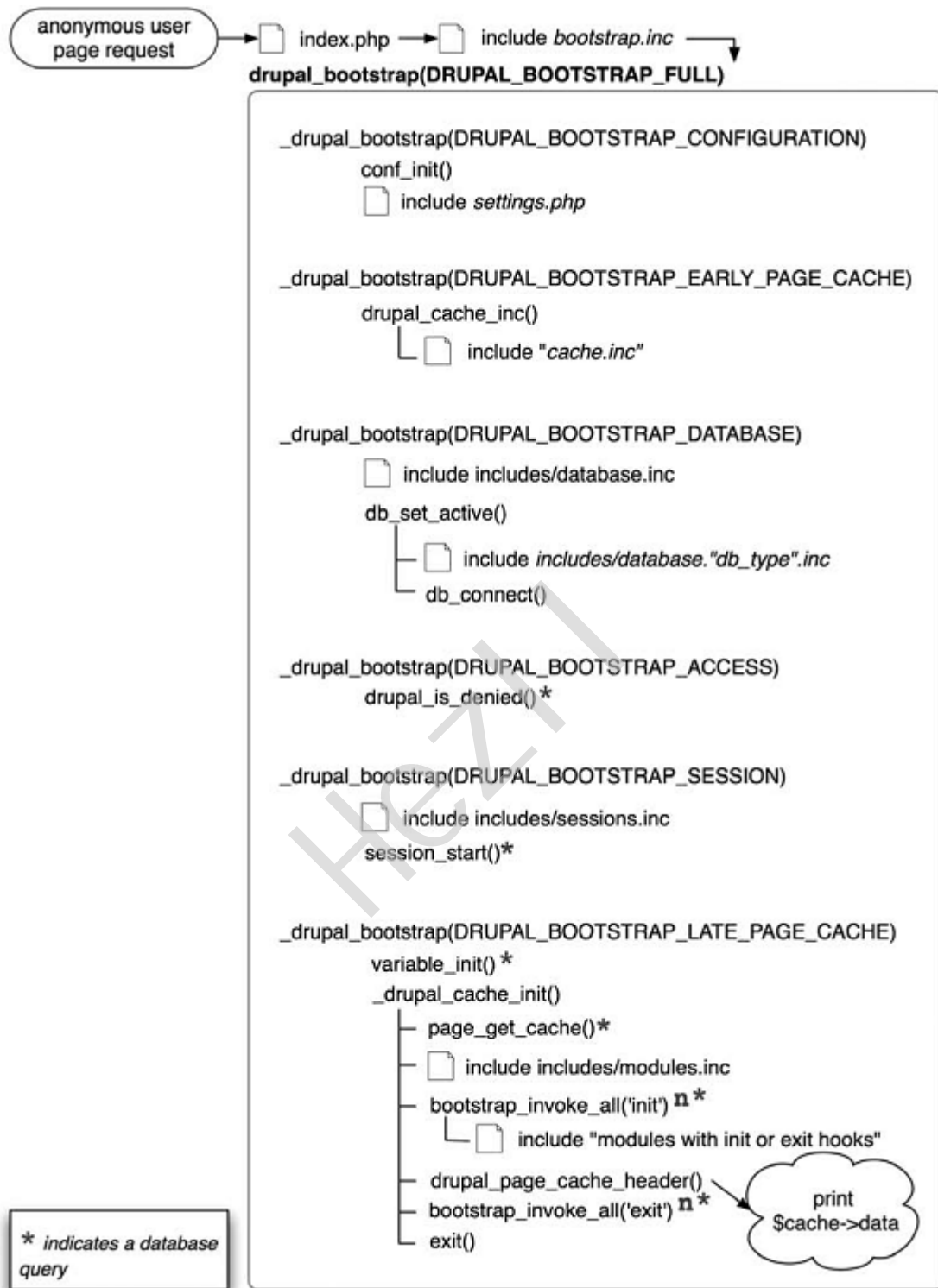


图 15-1 本图展示了当 Drupal 处于页面缓存普通模式下为匿名用户请求提供缓存页面的生命周期。引导指令流程的前面 5 个阶段与缓存无关，用在这里仅仅是为了保持完整性。n* 意味着在该处数据库查询次数是未知的。

在最开始，请求使得 Web 服务器执行 `index.php`，`index.php` 中的第一行 PHP 代码是用来包含文件 `includes/bootstrap.inc`，该文件含有加载引导指令的核心函数。接着，`index.php` 调用函数 `drupal_bootstrap()`。

`drupal_bootstrap()` 负责执行每一个引导指令阶段。对于普通模式缓存，我们只需要关心引导指令阶段 `DRUPAL_BOOTSTRAP_LATE_PAGE_CACHE` 就可以了。在该阶段，首先从数据库中取回系统变量。假定缓存策略是普通模式，接下来就是包含文件 `includes/module.inc` 了。`module.inc` 内部的函数用来允许 Drupal 将模块系统放到线上。Drupal 接着将初始化实现了钩子函数 `hook_init()` 或者 `hook_exit()` 的模块。通过分别调用 `bootstrap_invoke_all('init')` 和 `bootstrap_invoke_all('exit')` 来激活这些钩子函数。例如，统计模块，使用 `statistics_init()` 函数来追踪页面访问。节流阀（`throttle`）模块使用 `throttle_exit()` 函数来根据当前访问量改变节流阀值。

注意 在一个模块中使用 `hook_init()` 或者 `hook_exit()` 函数会为整个站点的性能带来负担，这是由于对于提供给访问者的每个缓存页面都需要加载你的模块。当你实现这些钩子函数时，可用的函数也受到限制，这是由于没有加载 `includes/common.inc`。通用函数比如 `t()`、`l()` 和 `pager_query()` 此时不可用。

`Drupal_page_cache_header()` 通过设置 HTTP 头部来准备缓存数据。这场情况下，Drupal 将设置 `Etag` 和 `304` 头部信息，这样浏览器可以使用它们自己内部的缓存机制，在应用时阻止不必要的 HTTP 循环请求。如果浏览器发送的头部信息已被请求过，那么就将缓存的数据方法送给浏览器。（译者注，我也没看懂这个函数是干什么的^^）。

激进模式

激进模式完全绕开了对所有模块的加载；如图 15-2 所示。这意味着，此时对于缓存页面不再调用 `init` 和 `exit` 钩子函数了。由于不用加载模块，最终的结果是需要解析的 PHP 代码少了，需要执行的数据库查询也少了。如果你启用了使用这些钩子的模块（比如统计模块和节流阀模块），那么在激进模式环境下，它们可能不会像你预期的那样工作了。在后台管理接口页面 **Administer > Site configuration > Performance**，Drupal 会给出警告，指出哪些模块可能受到影响。

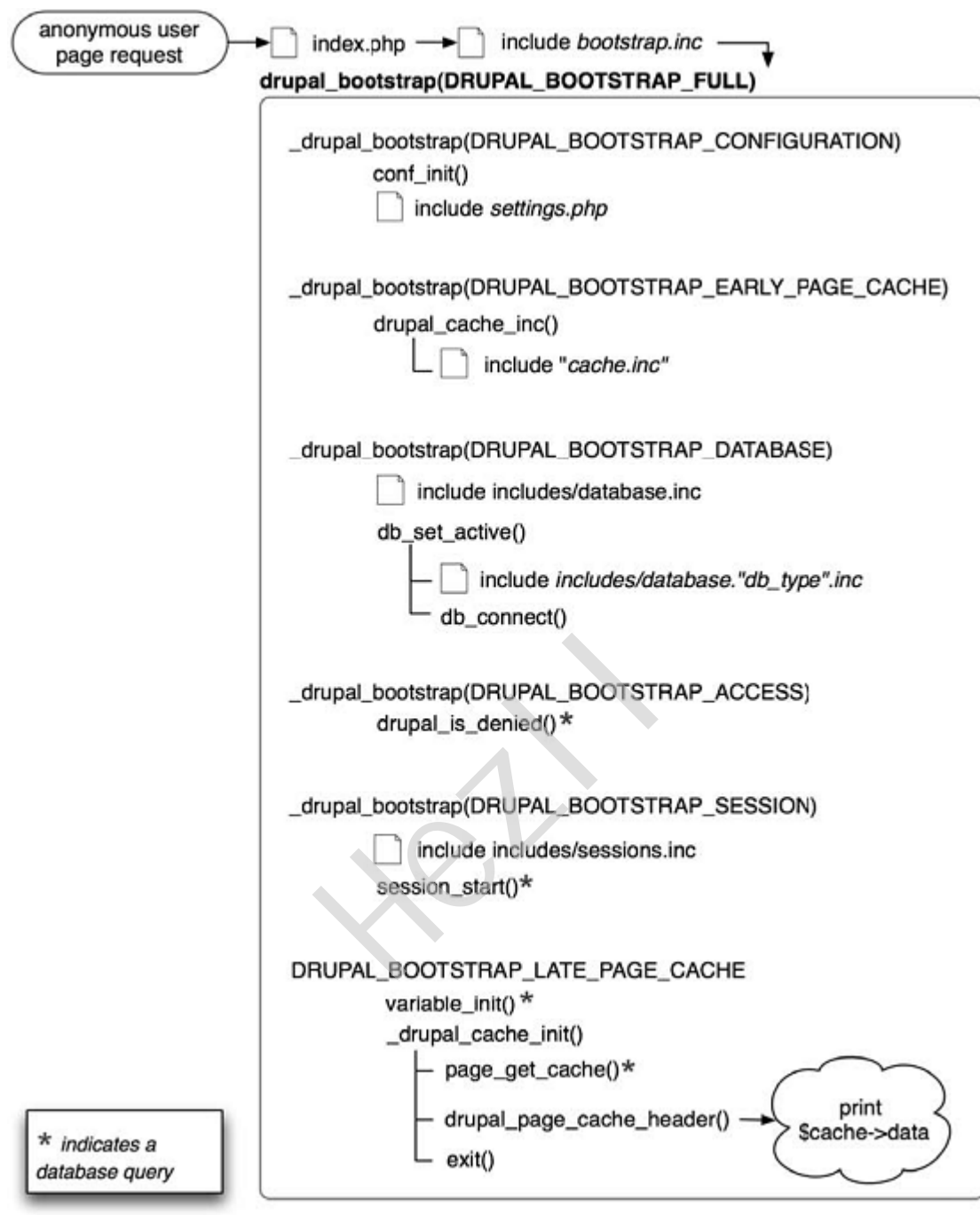


图 15-2 当 Drupal 处于页面缓存激进模式下为匿名用户请求提供缓存页面的生命周期

最小缓存周期

该设置用来控制你的站点上的缓存内容最小缓存生命周期。当一个用户提交了新的内容时，他/她将立即看到变化；然而，其他用户只有在过了最小缓存生命周期以后，才能看到新的内容。当然，如果将最小缓存周期设置为“none”，每个人都能立即看到新的内容。

Fastpath: 隐藏的缓存设置

缓存设置 **fastpath** 不能通过 **Drupal** 后台管理接口进行配置，这是因为它的高级特性；**fastpath** 使得程序员能够绕过 **Drupal** 实现一个高度定制化的缓存方案，比如内存或者基于文件的缓存；参看图 15-3。

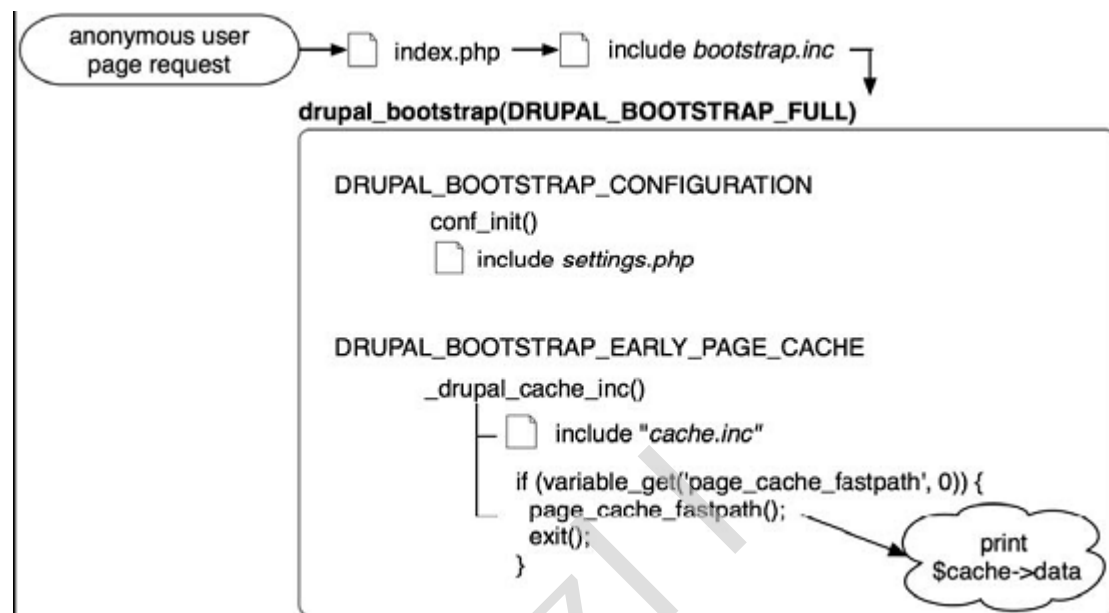


图 15-3 当 **Drupal** 处于页面缓存 **fastpath** 模式下为匿名用户请求提供缓存页面的生命周期

memcache 模块 (<http://drupal.org/project/memcache>) 是一个基于内存缓存的例子，而 **fastpath_fscache** 模块

(http://drupal.org/project/fastpath_fscache 在) 则是基于文件的。在 **sites/all/modules** 里面安装 **fastpath_fscache.module** 以后，我们将向你展示如何启用 **fastpath** 模式。

由于 **fastpath** 在默认情况下不需要数据库连接，所以要将所有的配置选项放在你的 **settings.php** 文件中：

```
$conf = array(
  'fastpath' => 1,
  'cache_inc' => 'sites/all/modules/fastpath_fscache/cache.fs.inc'
);
```

数组中的第一项用于启用 **fastpath** 模式。启用 **fastpath** 模式仅需这一行代码！第 2 项声明了 **fastpath_fscache** 将使用的定制缓存库函数。由于你是加载自己定制的缓存库函数来代替 **Drupal** 默认使用的 **includes/cache.inc** 库，所以你将需要编写你自己的 **cache_set()**, **cache_get()** 和 **cache_clear_all()** 函数。一旦启用了 **fastpath**，它将覆盖由 **Drupal** 后台管理接口页面所做的任何缓存设置。

Hezi

第 15 章 Drupal 缓存 (cache) (3) 缓存 API

使用缓存 API

对于模块开发者来说,如果他想使用缓存 API 的话,那么就需要掌握两个函数:`cache_set()` 和 `cache_get()`。

使用 `cache_set()` 缓存数据

`cache_set()` 用来将数据写入到缓存中。函数签名如下:

```
cache_set($cid, $table = 'cache', $data, $expire = CACHE_PERMANENT,
$headers = NULL)
```

函数参数有:

\$cid: 唯一的缓存 ID, 为一字符串, 作为缓存数据的键。

\$table: 用来存储数据的表的名称。你可以创建你自己的表, 或者使用 `cache`, `cache_filter`, `cache_menu`, `cache_page`。默认使用 `cache` 表。

\$data: 存储在缓存中的数据。记住复杂 PHP 数据类型必须先序列化。

\$expire: 缓存数据的有效期时间长度。可能值有 `CACHE_PERMANENT`, `CACHE_TEMPORARY`, 或者一个 Unix 时间戳。

\$headers: 对于缓存页面, 传给浏览器的 HTTP 头部字符串。

一个 `cache_set()` 的通用迭代模式可在 `filter.module` 找到。

```
// Store in cache with a minimum expiration time of 1 day.
if ($cache) {
cache_set($cid, 'cache_filter', $text, time() + (60 * 60 * 24));
}
```

使用 `cache_get()` 取回缓存过的数据

`cache_get()` 用来取回缓存过的数据。函数签名如下:

```
cache_get($cid, $table = 'cache')
```

函数参数有:

\$cid: 用于取回数据的缓存 ID。

\$table: 用来取回缓存数据的表的名称。你可以创建你自己的表，或者使用 `cache`, `cache_filter`, `cache_menu`, `cache_page`。默认使用 `cache` 表。

一个 `cache_set()` 的通用迭代模式可在 `filter.module` 找到。

```
// Check for a cached version of this piece of text.  
if ($cached = cache_get($cid, 'cache_filter')) {  
  return $cached->data;  
}
```

总结

在本章，你学到:

Drupal 提供的各种缓存类型: 页面、菜单、变量、过滤器缓存

页面缓存系统如何工作

普通模式，激进模式和 **fastpath** 模式之间的不同之处

缓存 API 函数

第 17 章 在 Drupal 中使用 jQuery (1)

在 Drupal 中使用 jQuery

JavaScript 无处不在。每一个流行的 Web 浏览器都带有一个 JavaScript 解释器。Apple 的主面板的窗口小部件是用 JavaScript 写的。Mozilla Firefox 使用 JavaScript 来实现它的用户接口。Adobe Photoshop 里面可以使用 JavaScript。JavaScript 存在于每个角落。

以前，大量的 JavaScript 代码使人头痛。如果你曾经有过这样痛苦的经历，现在是时候换一种方式了，过去的就都让它过去吧，现在我们开始使用 jQuery。jQuery 使得编写 JavaScript 代码更直观有趣，它已经被内置到 Drupal5 里面！在本章，你将学到什么是 jQuery 以及在 Drupal 中如何使用它。最后你将看到一个实际的例子。

什么是 jQuery?

jQuery，由 John Resig 创建，主要用于解决开发者在使用 JavaScript 遇到的常见的困惑和限制。编写 JavaScript 代码是件麻烦和让人头痛的事，通常查找你想操作的特定的 HTML 或者 CSS 元素是很困难的。jQuery 为你提供了在你的文档中查找这些元素的一种简单且快捷的方法。

查找一个对象的技术名字是 DOM traversal(往返移动)。DOM 是 Document Object Model (文档对象模型)的简称。该模型提供了一种树状方式通过标签访问页面元素，以及通过 JavaScript 访问其他元素，如图 17-1 所示。

注意，你可以从 jQuery 的官方网站 <http://jquery.com> 和 <http://www.visualjquery.com> 学到更多的相关知识。

当编写 JavaScript 代码时，你常常花费大量的时间来处理浏览器和操作系统的兼容问题。jQuery 替你处理了这些工作。还有，JavaScript 中不存在更高层次的函数。常见的任务比如对页面特定部分的动画特效，四处拖动，或者对于元素的排序，这些在 JavaScript 中都不存在相应的函数。而 jQuery 也解决了这些限制。

和 Drupal 一样，jQuery 的代码很少且很有效，仅有 19kb。jQuery 的核心是一个可扩展的框架，JavaScript 程序员可以编写相应的钩子函数，你可以在 <http://docs.jquery.com/Plugins>

找到数百个 jQuery 插件。

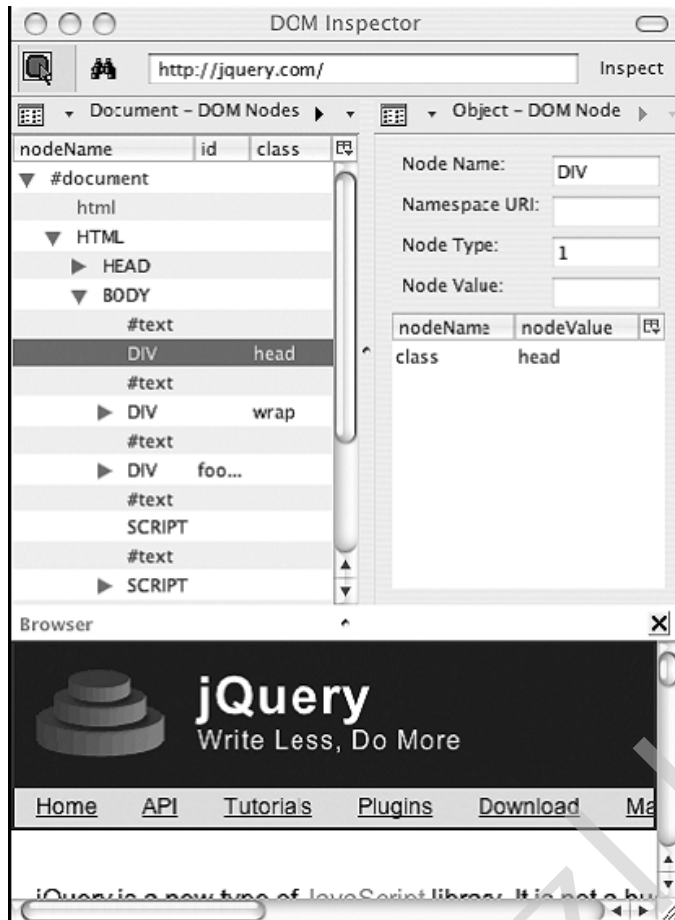


图 17-1 使用 Firefox 浏览器中的 Mozilla DOM 查看工具看到的 <http://jquery.com/> 页面的 DOM 结构图

旧方式

让我们首先快速的回顾一下纯 JavaScript 方式的 DOM 查找。下面的代码展示了 Drupal 是如何找到一个页面内所有的可伸缩的字段集的，接着就是 jQuery 的对应例子。

```
var fieldsets = document.getElementsByTagName('fieldset');  
  
var legend, fieldset;  
  
for (var i = 0; fieldset = fieldsets[i]; i++) {  
  if (!hasClass(fieldset, 'collapsible')) {  
    continue;  
  }  
  
  legend = fieldset.getElementsByTagName('legend');
```

```
if (legend.length == 0) {  
  
    continue;  
  
}  
  
legend = legend[0];  
  
...  
  
}
```

而下面则是使用 **jQuery** 后的代码：

```
$('#fieldset.collapsible > legend').each(function() {...});
```

你可以看到，正如 **jQuery** 的口号所说“写得少，做得多”。**jQuery** 处理了使用 **JavaScript** 操作 **DOM** 的常见的重复的任务，将其封装成了一种简洁且直观的语法。最终的代码简短，灵活，易读，其简单性有种 **Ruby** 血统。

jQuery 是如何工作的

jQuery 是一个在结构化文档中查找东西的工具。而 **CSS** 和 **XPath** 同样是此类工具。**CSS** 用于 (X) **HTML** 文档中，而 **XPath** 用于 **XML** 文档中。与其在 **JavaScript** 中实现另外的依照查找东西的方法，不如使用已有的，**jQuery** 同时实现了 **CSS** 和 **XPath** 查询语法，这就省去了让程序员学习另一种语法的成本。**jQuery** 的 **DOM** 操作非常直观，对 **CSS1-3** 全部支持，并支持基本的 **XPath** 表达式。

使用一个 CSS ID 选择器

让我们快速的回顾一下基本的 **CSS** 语法。

假定你要操作的 **HTML** 如下所示：

```
<p id="#intro">Welcome to the World of Widgets</p>
```

如果你想将段落的背景颜色设置为蓝色，你使用 **CSS** 在你的样式表中找到特定的段落，这里是用了 **#intro ID** 选择器，它应该在一个给定页面是唯一的。

```
#intro {  
  
background-color: blue;  
  
}
```

使用 **jQuery** 你也可以完成同样的工作。但是，首先，在这里先简单介绍一下 **jQuery** 的语法。为了代码保持简洁，**jQuery** 将 **jQuery** 命名空间映射为美元符号，代码如下：

```
//将 jQuery 变量定义为一个函数

var jQuery = function(a,c) {...}

//将 jQuery 映射为 '$'。

var $ = jQuery;
```

注意 如果你对 **jQuery** 引擎如何工作的感兴趣的话，你可以从 <http://jquery.com> 下载整个的 **jQuery** 的 **JavaScript** 文件。**Drupal5** 中包含的是一个压缩的版本，这使得浏览器加载它的速度更快一些。

下面是使用 **jQuery** 方式将你的段落的背景颜色设置为蓝色的代码：

```
$("#intro").css("background-color", "blue");
```

你甚至可以加点 **jQuery** 特效，慢慢的显示段落文本：

```
$("#intro").css("background-color", "blue").fadeIn("slow");
```

使用一个 CSS 类 选择器

下面是使用 **CSS** 类 选择器来代替 **CSS ID** 选择器(上节中我们做的)的类似的例子。**HTML** 如下所示：

```
<p class="intro">Welcome to the World of Widgets</p>
```

我们的 **CSS** 如下所示：

```
.intro { background-color: blue; }
```

下面的也可以工作，限定范围更窄一些：

```
p.intro { background-color: blue; }
```

下面是对应的 **jQuery** 代码：

```
$(".intro").css("background-color", "blue").fadeIn("slow");
```

或

```
$("#p.intro").css("background-color", "blue").fadeIn("slow");
```

在第一个例子中，你让 **jQuery** 查找 **info** 类的所有 **HTML** 元素，而第二个例子则有一点细小的区别。这里你查找 **info** 类的所有段落元素。注意后者的速度更快一些，这是因为查找的范围小了很多，使用 **p.intro** 将查找的范围限制在了段落元素里面。

提示 在 **CSS** 中，“.”类选择器在同一文档中可以重复出现，而“#”ID 选择器则在同一页面只能出现一次。

Hezi

第 17 章 在 Drupal 中使用 jQuery (3) 编写一个使用 jQuery 的 Drupal 模块

创建一个 jQuery 的投票小部件

让我们编写一个基于 jQuery 的 Drupal 模块。我们将建立一个如图 17-2 所示的 Ajax 的投票小组件，它可以让用户为喜欢的文章添加一分。我们使用 jQuery 来处理投票和总分的变化，而不用重新加载整个页面。我们还添加一个基于角色的授权，这样只有具有“rate 内容”授权的用户才允许投票。由于每个用户的每次投票只能增加一份，让我们将模块的名称命名为“plus1”。

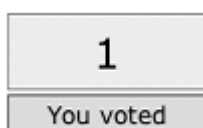


图 17-2 投票组件

在我们接触到 plus1 的 jQuery 部分以前，首先我们需要构建模块所需的基本一些代码。如果你以前从来没有创建过模块，请参看第 2 章。如果有经验的话，现在就开始了。

在 sites/all/modules/custom 下面创建一个名为 plus1 的目录（你可能需要创建这个目录如果它不存在的话）。在目录 plus1 下面，创建文件 plus1.info，它包含下面的代码：

```
name = Plus 1  
  
description = "A +1 voting widget for nodes. "  
  
version = "$Name$"
```

该文件将模块注册到 Drupal 中，这样可以通过管理页面启用或者禁用它。

接着，你将创建 plus1.install 文件。这个 PHP 文件里面的函数将在模块启用或者禁用的时候调用，一般用来创建或者删除数据库表。在这里我们想用它来追踪谁为哪个节点投了票。

```
<?php  
  
// $Id$  
  
/**
```

```
* Implementation of hook_install().
*/
function plus1_install() {
switch ($GLOBALS['db_type']) {
case 'mysql':
case 'mysqli':
db_query("CREATE TABLE {plus1_vote} (
uid int NOT NULL default '0',
nid int NOT NULL default '0',
vote tinyint NOT NULL default '0',
created int NOT NULL default '0',
PRIMARY KEY (uid,nid),
KEY score (vote),
KEY nid (nid),
KEY uid (uid)
) /*!40100 DEFAULT CHARACTER SET UTF8 */");
break;
case 'pgsql':
db_query("CREATE TABLE {plus1_vote} (
uid int NOT NULL default '0',
nid int NOT NULL default '0',
vote tinyint NOT NULL default '0',
created int NOT NULL default '0',
PRIMARY KEY (uid,nid)
);");
```

```

db_query("CREATE INDEX {plus1_vote}_score_idx ON {plus1_vote}
(vote);");

db_query("CREATE INDEX {plus1_vote}_nid_idx ON {plus1_vote} (nid);");

db_query("CREATE INDEX {plus1_vote}_uid_idx ON {plus1_vote} (uid);");

break;

}

}

/**
 * Implementation of hook_uninstall().
 */

function plus1_uninstall() {
db_query('DROP TABLE {plus1_vote}');
}

```

还有就是添加 **plus1.css** 文件。这个文件不是必需的，但它可以使投票组件看起来更美观，如图 17-3 所示。



图 17-3 带有 CSS 和不带有的对比

向 **plus1.css** 添加如下内容：

```

div.plus1-widget {
width: 100px;
margin-bottom: 5px;
text-align: center;
}

```

```
div.plus1-widget .score {  
  
padding: 10px;  
  
border: 1px solid #999;  
  
background-color: #eee;  
  
font-size: 175%;  
  
}  
  
div.plus1-widget .vote {  
  
padding: 1px 5px;  
  
margin-top: 2px;  
  
border: 1px solid #666;  
  
background-color: #ddd;  
  
}
```

现在你创建了起支撑作用的文件,现在让我们将注意力集中到 **jQuery** 的 **JavaScript** 文件,还有模块文件本身。创建两个空文件,其中一个命名为 **jquery.plus1.js**,另一个命名为 **plus1.module**,并将它们放到 **plus1** 文件夹下。在接下来的步骤中,你将逐步的像这两个文件添加代码。总结一下,你创建了以下文件:

sites/

all/

modules/

plus1/

jquery.plus1.js

plus1.css

plus1.info

plus1.install

plus1.module

创建模块

在一个文本编辑器中打开空的 `plus1.module` 文件，并向其中添加标准的 **Drupal** 头部文档：

```
<?php
// $Id$

/**
 * @file
 *
 * A simple +1 voting widget.
 */
```

接下来你要一个一个的添加你要用到的 **Drupal** 钩子函数。其中一个比较简单的是 `hook_perm()` 的使用，它让你为 **Drupal** 的基于角色的访问控制页面添加一个“rate 内容”的授权，你将使用这一授权来阻止匿名用户在未创建帐号或者登陆的时候投票。

```
/**
 * Implementation of hook_perm().
 */
function plus1_perm() {
  return array('rate content');
}
```

现在你将开始实现一些 **Ajax** 功能。**jQuery** 的一个重要特性提交它自己的 **HTTP GET** 或 **POST** 请求，这将使你投票提交给 **Drupal** 而不用刷新整个页面。**jQuery** 将拦截到对投票链接的点击，然后向 **Drupal** 发送一个请求，**Drupal** 将保存投票并返回分数。**jQuery** 将使用新的分数来更新页面上的分数。图 17-4 展示了我们要做的场景图。

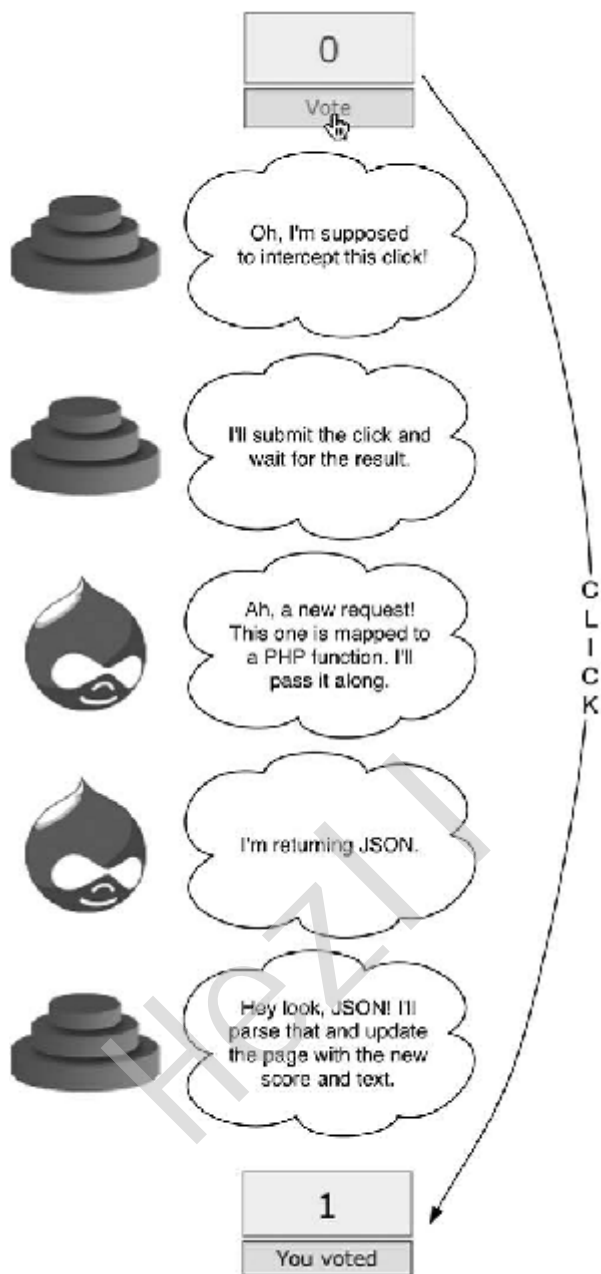


图 17-4 投票更新流程的概貌图

一旦 jQuery 拦截了对投票链接的点击事件，它需要能够将 URL 传递给 Drupal 来提交投票。我们使用钩子函数 `hook_menu` 来讲由 jQuery 提交的投票 URL 映射到一个 Drupal 的 PHP 函数。该 PHP 函数将投票保存到数据库中，并返回一个分数给 jQuery（以 JSON 的形式，即 JavaScript Object Notation）。

```
/**
```

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

```
*/
```

```

function plus1_menu($may_cache) {

  $items = array();

  if ($may_cache) {

    $items[] = array(

      'path' => 'plus1/vote',

      'callback' => 'plus1_vote',

      'type' => MENU_CALLBACK,

      'access' => user_access('rate content'),

    );

  }

}

```

return \$items;在前面的函数中，当路径为 **plus1/vote** 的请求进来以后，如果请求该路径的用户拥有“rate content”授权，那么将有函数 **plus1_vote()**处理它。路径 **plus1/vote/3** 翻译为 PHP 函数就是调用 **plus1_vote(3)**（参看第 4 章，关于 Drupal 的菜单/回调系统的更详细信息）。

```

/**
 * Called by jQuery.
 * This submits the vote request and returns JSON to be parsed by jQuery.
 */

function plus1_vote($nid) {

  global $user;

  // Authors may not vote on their own posts.

  $is_author = db_result(db_query('SELECT uid FROM {node} WHERE nid = %d
  AND
  uid = %d', $nid, $user->uid));

  // Before processing the vote we check that the user is logged in,
  // we have a node ID, and the user is not the author of the node.

```

```
if ($user->uid && ($nid > 0) && !$is_author) {  
  
$vote = plus1_get_vote($nid, $user->uid);  
  
if (!$vote) {  
  
$values = array(  
  
'uid' => $user->uid,  
  
'nid' => $nid,  
  
'vote' => 1,  
  
);  
  
plus1_vote_save($values);  
  
watchdog('plus1', t('Vote by @user accepted', array('@user' =>  
$user->name)));  
  
$score = plus1_get_score($nid);  
  
// This print statement will return results to jQuery's request.  
  
print drupal_to_js(array(  
  
'score' => $score,  
  
'voted' => t(' You voted ' )  
  
)  
  
);  
  
}  
  
}  
  
exit();  
  
}
```

上面的函数 `plus1_vote()` 保存了当前的投票并向 `jQuery` 返回信息，信息的格式是包含了新分数和字符串 `You voted` 的关联数组，该字符串用于替换投票组件下面的“Vote”文本，我们使用了 `t(' You voted ')` 而不是在 `jQuery` 中直接创建它，这样就可以将它翻译成其它语言。这个数组传给了 `drupal_to_js()`，`drupal_to_js()` 将 PHP 变量转化为 JavaScript 的等价形式，在这里将一个 PHP 关联数组转化为了一个 JavaScript 关联数组。Drupal 将 JavaScript 数组序列化为 JSON 格式（关于 JSON 的更多信息，参看

<http://en.wikipedia.org/wiki/JSON>)。在上面的代码中我们创建了几个基本函数，现在让我们创建这些函数：

```
/**
 * Return the number of votes for a given node ID/user ID pair.
 *
 * @param $nid
 * A node ID.
 * @param $uid
 * A user ID.
 * @return Integer
 * Number of votes the user has cast on this node.
 */
function plus1_get_vote($nid, $uid) {
return (int) db_result(db_query('SELECT vote FROM {plus1_vote} WHERE nid
= %d
AND uid = %d', $nid, $uid));
}
/**
 * Return the total score of a node.
 *
 * @param $nid
 * A node ID.
 * @return Integer
 * The score.
 */
function plus1_get_score($nid) {
```

```

return (int) db_result(db_query('SELECT SUM(vote) FROM {plus1_vote}
WHERE

nid = %d', $nid));

}

/**

* Save the vote.

*

* @param $values

* An array of the values to save to the database.

*/

function plus1_vote_save($values) {

db_query('DELETE FROM {plus1_vote} WHERE uid = %d AND nid = %d',
$values['uid'],
$values['nid']);

db_query('INSERT INTO {plus1_vote} (uid, nid, vote, created) VALUES (%d,
%d, %d,
%d)', $values['uid'], $values['nid'], $values['vote'], time());

}

```

现在，基本的 **getter** 和 **setter** 已经写好，现在让我们关注投票小部件生成代码：

```

/**

* Create voting widget to display on the webpage.

*/

function plus1_jquery_widget($nid) {

// Load the JavaScript and CSS files.

drupal_add_js(drupal_get_path('module', 'plus1') .'/jquery.plus1.js');

drupal_add_css(drupal_get_path('module', 'plus1') .'/plus1.css');

```

```
global $user;

$score = plus1_get_score($nid);

$is_author = db_result(db_query('SELECT uid FROM {node} WHERE nid = %d
AND uid = %d', $nid, $user->uid));

$voted = plus1_get_vote($nid, $user->uid);

return theme('plus1_widget', $nid, $score, $is_author, $voted);

}

/**
 * Theme for the voting widget.
 */

function theme_plus1_widget($nid, $score, $is_author, $voted) {

$output = '<div class="plus1-widget">';

$output .= '<div class="score">';

$output .= $score;

$output .= '</div>';

$output .= '<div class="vote">';

if ($is_author) { // User is author; not allowed to vote.

$output .= t('Votes');

}

elseif ($voted) { // User already voted.

$output .= t('You voted');

}

else { // User is eligible to vote.

// The class plus1-link is what we will search for in our jQuery later.

$output .= l(t('Vote'), "plus1/vote/$nid", array('class' => 'plus1-link'));
```



```

}

$output .= '</div>';

$output .= '</div>';

return $output;

}

```

在前面的方法 `plus1_jquery_widget()` 中，我们首先加载相应的 CSS 和 Javascript 文件，然后将小部件的主体化委托给了我们自己创建的定制函数 `theme_plus1_widget()`。记住 `theme('plus1_widget')` 实际上调用的就是 `theme_plus1_widget()`（参看第 8 章关于它是如何工作的）。创建一个独立的主题函数，而不是将 HTML 代码放到方法 `plus1_jquery_widget()` 中，这将允许设计者在它们想改变外观时能够覆写该函数。我们的主题函数 `theme_plus1_widget()`，为关键的 HTML 元素创建 CSS 类选择器，这使得 jQuery 能够非常方便的访问到它们。还有，让我们看一下链接的 URL，它指向 `plus1/vote/$nid`，其中 `$nid` 是当前已发布节点的 ID。当用户点击链接时，jQuery 将代替 Drupal 对它进行拦截并处理。我们是通过使用 jQuery 监控该链接上的 `onClick` 事件来完成拦截的。看一下在我们创建链接时是如何定义 CSS 类选择器 `plus1-link`。看一下在我们后面的 Javascript 代码中选择器的使用 `a.plus1-link`。它由 `<a>` 元素和 CSS 类选择器 `plus1-link` 组成。

函数 `plus1_jquery_widget()` 生成发送给浏览器的小部件。你想让它出现在节点的查看页面中，这样当用户查看节点时，就可以对它们投票了。你能猜一下使用哪一个 Drupal 钩子函数实现这一点吗？他就是我们的老朋友 `hook_nodeapi()`，它允许我们可以像创建节点一样任意的修改节点。

```

/**
 * Implementation of hook_nodeapi().
 */

function plus1_nodeapi(&$node, $op, $teaser, $page) {

  switch ($op) {

    case 'view':

      // Show the widget, but only if the full node is being displayed.

      if (!$teaser) {

        $node->content['plus1_widget'] = array(

          '#value' => plus1_jquery_widget($node->nid),

```

```
'#weight' => 100,
);
}
break;
case 'delete':
db_query('DELETE FROM {plus1_vote} WHERE nid = %d', $node->nid);
break;
}
}
```

我们将重量（**weight**）元素的值设为一个大点的值 **100**，这样就确保了小部件出现在节点的底部而不是顶部。我们在删除（**delete**）情况下，也放置了一段代码，这样当节点被删除时与其相关的投票记录也将一同被删除。

上面就是 **plus1.module** 的全部内容了，离我们整个模块的完成现在就剩下编写 **jquery.plus1.js** 了，它不足 **15** 行代码：

```
// $Id$
// Global killswitch: only run if we are in a supported browser.
if (Drupal.jsEnabled) {
$(document).ready(function(){
$('a.plus1-link').click(function(){
var voteSaved = function (data) {
var result = Drupal.parseJson(data);
$('div.score').fadeIn('slow').html(result['score']);
$('div.vote').html(result['voted']);
}
$.get(this.href, null, voteSaved);
return false;
}
```

```
});  
});  
}
```

你应该将你的 **jQuery** 代码封装在 **Drupal.jsEnabled** 测试中，该测试确保了当前浏览器对特定 **DOM** 方法的支持（如果不支持的话，我们的 **Javascript** 代码就不被执行）。

该 **Javascript** 代码向 **a.plus1-link** 添加了一个事件侦听器（还记得我们如何将 **.plus1-link** 定义为 **CSS** 类选择器？），这样当用户点击链接时，它将触发一个 **HTTP GET** 请求，发送到它指向的 **URL**。当请求完成后，返回值（从 **Drupal** 中返回）作为 **data** 参数传递到匿名函数中，用该函数给变量 **voteSaved** 赋值。返回值是一个序列化为 **JSON** 格式的 **Javascript** 数组，所以你要使用 **Drupal.parseJson()** 对其反序列化。通过关联数组的键来引用数组，数组键在 **Drupal** 内部的 **plus1_vote()** 函数中指定。最后 **Javascript** 更新了分数并将文本“**Vote**”改为“**You Voted**”。为了阻止加载整个页面（因为这是 **Ajax** 要求的），我们在 **Javascript** 的 **jQuery** 函数中将返回值置为 **false**。

提示：如果你在计算机上自己动手实践，但是小部件却不能正常工作。你需要检查一下，你是不是以内容创建者的身份登录了（这是因为用户不能对自己创建的内容投票），并检查一下登录用户是否拥有权限“**rate content**”（“评论内容”）。一个好用的 **Ajax** 请求测试工具是名为 **Firbug** 的 **FireFox** 插件，你可以从 <http://getfirebug.com/> 下载到它。

扩展该模块的方式

对本模块的一个很好的扩展，是允许站点管理员对特定节点类型启用投票小部件。你可以使用我们在第 2 章节点注释模块所用到的方法来完成它。然后，你需要在

```
hook_nodeapi('view')
```

内部检查给定节点类型是否启用了投票功能。

兼容性

jQuery 的兼容性，作为 **jQuery** 的重要信息，你可以在 <http://docs.jquery.com> 找到。总之，**jQuery** 支持下面的浏览器：

IE6.0 及更高版本

Mozilla Firefox 1.5 及更高版本

Apple Safari 2.0 及更高版本

Opera 9.0 及更高版本

小结

在本章，你学到了：

什么是 **jQuery**

jQuery 如何工作的等一般概念

jQuery 和 **Drupal** 是如何交互的，请求和数值在前后之间的传递

如何构建一个简单的投票小部件

HEZIL

第 23 章 Drupal 安装过程 profile

安装过程 Profile

当你安装 **Drupal** 时，一些模块被启用，一些特定的配置被选择，但是这些默认的可能并不是你所需要的。**Drupal** 安装器使用了一个默认的安装过程 **profile**，用来决定所有的这些配置。通过创建你自己的安装过程 **profile**，你可以定制 **Drupal** 的初始安装，从而使你的站点带有你想要的模块和设置。可能你在为每一个高校工作，你想创建一个安装过程 **profile**，从而能够启用一个与你高校的单点登录系统相绑定的定制模块，能够为站点管理员创建一个新的角色，能够在安装完成时向你发送 **e-mail**。**Drupal** 的安装器系统，允许你通过创建一个安装过程 **profile** 来定制安装时的各种操作。在本章你将学到如何做到这一点。

Profile 的存放位置

你的 **Drupal** 站点已经包含了一个安装过程 **profile**。它是 **Drupal** 自带的默认的安装过程 **profile**，位于 `profiles/default/default.profile`。我们想创建一个新的名为“**university**”（大学）的 **profile**，所以我们将先在 `profiles/university/university.profile` 创建一个新的文件。先在，我们将向这个文件仅添加一个独立的函数：

```
<?php
// $Id$
/**
 * Return a description of the profile for the initial installation
 screen.
 *
 * @return
 * An array with keys 'name' and 'description' describing this profile.
 */
function university_profile_details() {
  return array(
    'name' => st('Drupal (Customized for Iowa State University)'),
    'description' => st('Select this profile to enable settings typical
for a departmental website.')
  );
}
```

注意，这里文件的名称与 **profile** 目录的名称相同，在文件名的后面使用了 **.profile** 后缀，文件 **university.profile** 中的所有函数都以前缀 **university_** 开头。我们还是用了函数 **st**

()，而不是通常用的 t ()，这是因为在安装器运行这段代码时，**Drupal** 还没有运行完一个完整的引导指令，所以 t () 不可用。

安装过程 Profile 的工作原理

当 **Drupal** 的安装器启动时，它扫描 **profiles** 目录以查看有多少个可用的 **profile**。如果他发现多于一个时，它将向用户提供所有的 **profile** 以备用户选择。例如，创建了我们的 **university.profile** 文件并向其添加了 **university_profile_details()** 函数以后，访问 <http://example.com/install.php>，将会产生一个如图 23-1 所示的截图。

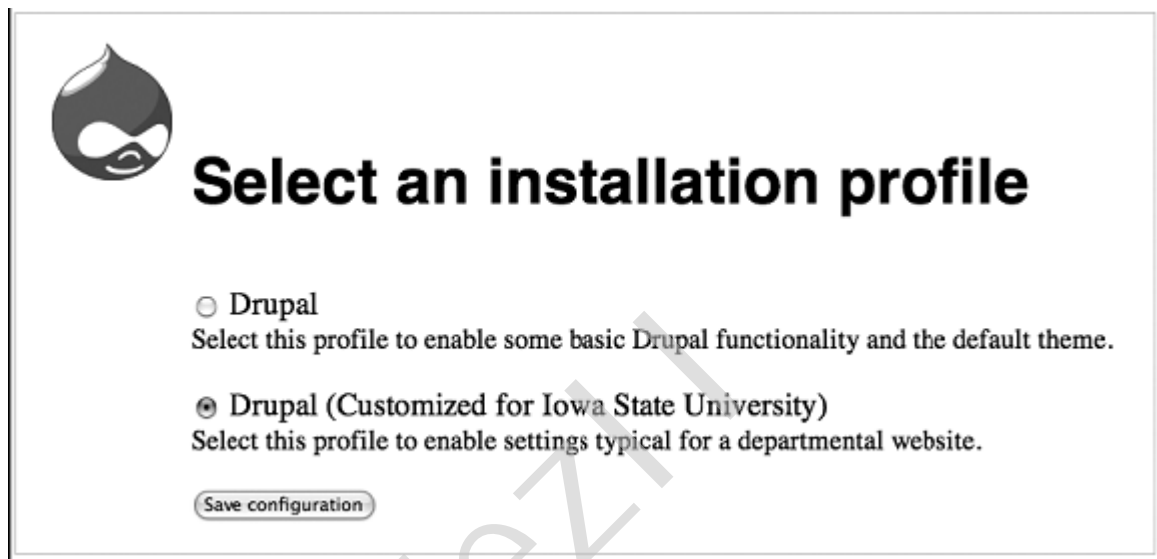


图 23-1 Drupal 展示了一个有哪些 profile 可用的选项

Drupal 的安装器接下来将再度回到安装过程 **profile** 上。它将找出该 **profile** 想要启用哪些模块，在安装过程的最后，当安装器将执行交给安装过程 **profile** 时，它又回来了一次。在后面的之一阶段，更多的 **Drupal** 定制化完成了。流程的概貌如图 23-2 所示。

指示启用哪些模块

通过添加函数 **university_profile_modules()**，我们告诉 **Drupal** 我们的安装过程 **profile** 想启用哪些模块（还有，我们知道这个函数的名称应该由我们的 **profile** 的名称加上 **_profile_modules** 合成）。

```
/**  
 * Return an array of the modules to be enabled when this profile is  
 installed.  
 */
```

```
* @return
* An array of modules to be enabled.
*/
function university_profile_modules() {
  return array(
    // Enable required core modules.
    'block', 'filter', 'help', 'node', 'system', 'user', 'watchdog',
    // Enable optional core modules.
    'color', 'help', 'taxonomy', 'throttle', 'search', 'statistics',
    // Enable single signon by enabling a contributed module.
    'pubcookie',
  );
}
```

启用这些模块以前，安装器会询问每一个模块，以查看正被安装的 **Drupal** 系统是否提供了该模块所有的必要条件。通过为每个模块调用 `hook_requirements('install')` 来完成检查。如果要求没有被满足，安装器将会失败，并报告缺少了哪些条件。

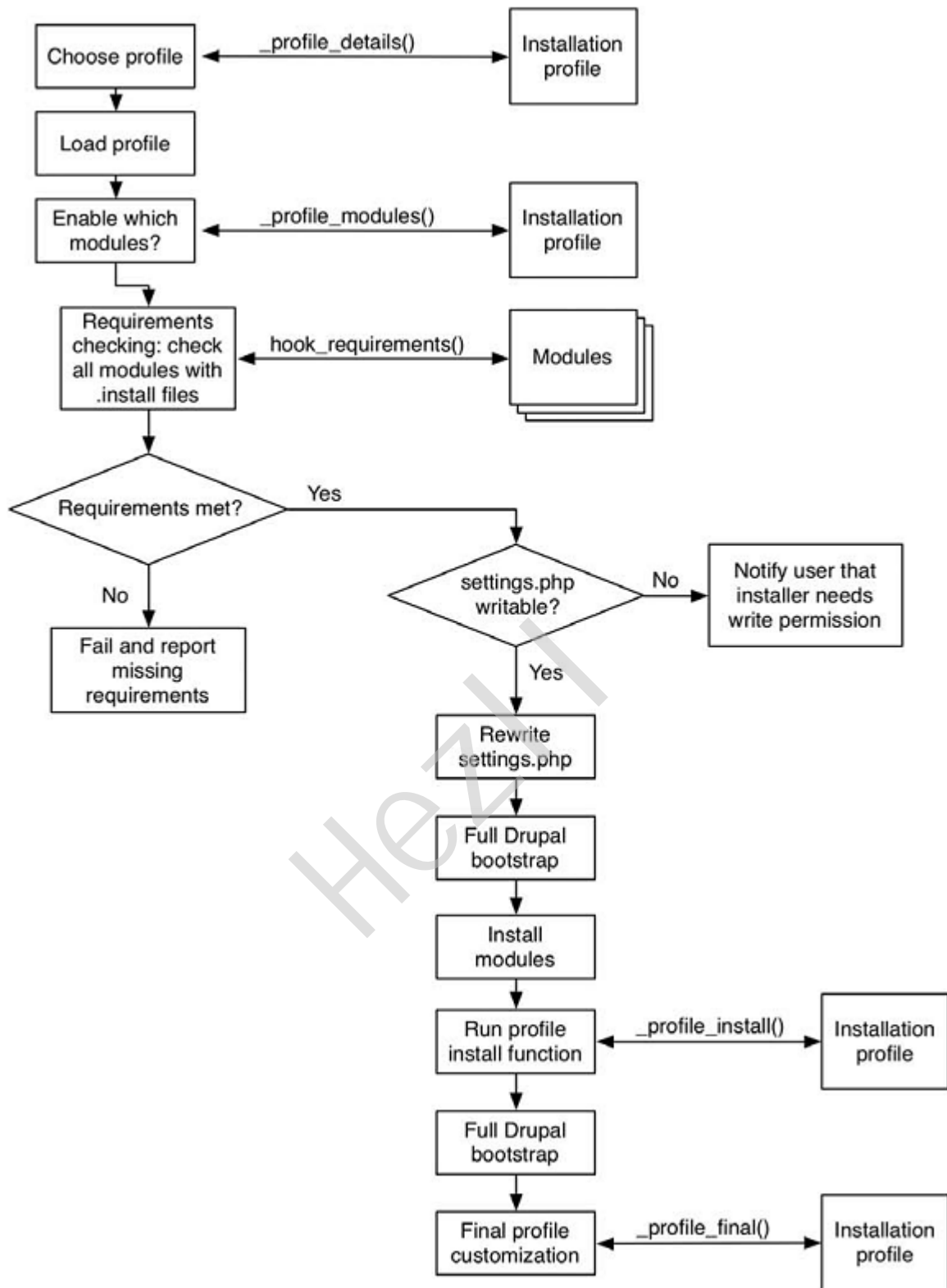


图 23-2 安装器是如何与安装过程 profile 交互的

注意 必要条件钩子是一个可选的钩子，它允许模块在在安装以前测试所需环境是否准备好了。关于该钩子的更多信息，参看

http://api.drupal.org/api/5/function/hook_requirements。

安装器在启用模块以前确保模块是存在的。它会在多个地方查找，这些位置在表 23-1 中列出来了。由于我们要启用 **pubcookie** 模块（不包含在 **Drupal** 内核中的模块），在运行我们的安装过程 **profile** 以前，我们需要确保能够在这些所列的目录中找到它。

Table 23-1. Drupal 模块可以存放的目录

目录	存放的模块
modules	Drupal 核心模块
sites/all/modules	第 3 方模块(适用于所有站点)
profiles/profilename/modules	位于安装过程 profile 中的模块
sites/*/modules	与你的 settings.php 文件位于同一目录的模块

安装器还会在放置你站点 **settings.php** 文件的目录下查找模块。如果 **settings.php** 位于 **sites/default**，那么 **Drupal** 会在 **sites/default/modules** 下面找。类似的，如果 **settings.php** 位于 **sites/example.com**，那么 **Drupal** 将在 **sites/example.com/modules** 下面寻找模块。

最后设置

安装系统将按照我们列出模块的次序来启用我们所需要的模块，然后再调用安装过程 **profile**。这一次，安装系统将查找一个名为 **university_install()** 的函数。在我们的例子中，我们没有实现这个函数，这是因为我们想在函数 **university_profile_final()** 中完成所有的事情。但是在引导指令结束以前调用 **install()** 钩子，如果需要的话，使你能够在最后时刻修改所要设置的东西。

注意 在 **Drupal** 的下一个版本中，安装过程 **profile** 中的 **install()** 函数将被删除，如果兼容性对你很重要的话，推荐你完全彻底 **profile_final()** 钩子。

最后，安装器调用 **university_profile_final()**。

```
/**
 * Perform final installation tasks for this installation profile.
 */
function university_profile_final () {
// Define a node type, 'page'.
  $node_type = array(
    'type' => 'page',
    'name' => st('Page'),
```

```

    'module' => 'node',
    'description' => st('A standard web page.'),
    'custom' => TRUE,
    'modified' => TRUE,
    'locked' => FALSE,
    'has_title' => TRUE,
    'has_body' => TRUE,
    'orig_type' => 'page',
    'is_new' => TRUE,
  );
  node_type_save((object) $node_type);

// Page node types should be published and create new revisions by default.
  variable_set('node_options_page', array('status', 'revision'));

// If the administrator enables the comment module, we want
// to have comments disabled for pages.
  variable_set('comment_page', COMMENT_NODE_DISABLED);

// Define a node type, 'newsitem'.
  $node_type = array(
    'type' => 'news',
    'name' => st('News Item'),
    'module' => 'node',
    'description' => st('A news item for the front page.'),
    'custom' => TRUE,
    'modified' => TRUE,
    'locked' => FALSE,
    'has_title' => TRUE,
    'has_body' => TRUE,
    'orig_type' => 'news',
    'is_new' => TRUE,
  );
  node_type_save((object) $node_type);

// News items should be published and promoted to front page by default.
// News items should create new revisions by default.
  variable_set('node_options_news', array('status', 'revision',
    'promote'));

// If the administrator enables the comment module, we want
// to have comments enabled for news items.

```

```

variable_set('comment_news', COMMENT_NODE_READ_WRITE);
// Create a taxonomy so news can be classified.
$vocabulary = array(
  'name' => t('News Categories'),
  'description' => st('Select the appropriate audience for your news
item.'),
  'help' => st('You may select multiple audiences.'),
  'nodes' => array('news' => st('News Item')),
  'hierarchy' => 0,
  'relations' => 0,
  'tags' => 0,
  'multiple' => 1,
  'required' => 0,
);
taxonomy_save_vocabulary($vocabulary);

// Define some terms to categorize news items.
$terms = array(
  st('Departmental News'),
  st('Faculty News'),
  st('Staff News'),
  st('Student News'),
);
// Submit the "Add term" form programmatically for each term.
foreach ($terms as $name) {
  drupal_execute('taxonomy_form_term', array('name' => $name), $vid);
}
// Add a role.
db_query("INSERT INTO {role} (name) VALUES ('%s')", 'site
administrator');
// Configure the pubcookie module.
variable_set('pubcookie_login_dir', 'login');
variable_set('pubcookie_id_is_email', 1);
// ...other settings go here
// Report by email that a new Drupal site has been installed.
$to = 'administrator@example.com';
$from = ini_get('sendmail_from');
$subject = st('New Drupal site created!');
$body = st('A new Drupal site was created: @site', array('@site' =>
base_path()));
drupal_mail('university-profile', $to, $subject, $body, $from);

```

```
}
```

如前面的例子所示，安装过程 `profile` 需要完成多个常用任务。安装器在调用 `profile_final()` 钩子以前，先完成一个完整的 **Drupal** 引导指令，所以在该钩子中可以使用所有的 **Drupal** 函数了。

注意 在安装过程 `profile` 中，我们使用 `st()` 来代替 `t()`，从而使得整个安装过程 `profile` 可被翻译，如果有任何翻译的话，将被存放在一个安装过程 `profile` 的翻译文件中。这是一个与安装过程 `profile` 位于同一个目录下的 `.po` 文件。更多关于 `.po` 文件的信息，参看第 18 章。

设置 Drupal 变量

通过简单的调用 `variable_set()` 来设置 **Drupal** 变量：

```
variable_set('pubcookie_login_dir', 'login');
```

创建初始节点类型

如果你需要使用 **Drupal** 内置的内容类型系统创建新的节点类型的话，你只需要创建一个节点类型定义对象并将其传递给 `node_type_save()` 就可以了。在前面的 `profile` 例子中，我们创建了两个节点类型：“`page`”用于普通网页，而“`news`”用于新闻项。我们接着使用了 `variable_set()` 来设置节点的默认选项，这样发布的新闻项将会出现在首页，而“`page`”则不。

如果你启用了提供节点类型的模块，根据这些模块中的 `node_info()` 钩子，**Drupal** 就可以调用里面的节点类型了。

将信息保存到数据库中

一个安装过程 `profile` 可能想修改一些数据库设置。由于现在数据库连接已经可用，可以使用 `db_query()` 来修改数据库。在我们的例子中，我们为该 **Drupal** 站点添加了一个角色。在你的 `profile` 中，你可做更多修改，比如向

`permission` 表中插入一些权限。

一个简单的获取合适的查询的方法是，先按照默认设置安装好 **Drupal**，接着完全按照你想要的方式来配置它。这甚至可以包括创建一些节点，完成 `URL` 别名的设置。本章例子中的安装过程 `profile` 可能想要一个 **About** 页面，一个 **Courses Taught** 页面，等等。当完成了这一配置过程以后，你可以使用你的数据库工具来将你站点的数据库完全导入到一个

SQL 脚本中。然后在 SQL 脚本中挑出你想要的 **INSERT SQL** 命令，并将它们包含到你的安装过程 **profile** 中。

提交表单

由于 **Drupal** 支持通过程序的方式提交表单，如果你要与网站交互的话，你可以使用 **drupal_execute()** 来提交表单。在前面的例子中，我们使用这种方式来为站点添加分类词语。关于 **drupal_execute()** 的更多信息，参看第 10 章。

总结

在本章，你学到了以下几点：

- 什么是安装过程 **profile**
- 安装过程 **profile** 的存放位置
- 如何创建一个基本的安装过程 **profile**
- 在最后安装阶段如何操纵 **Drupal**