ZOPE 2 教程

杜文山

2005-07-31

第一章 介绍 zope

Zope 是一种让具备不同技能的开发人员一起构建 Web 应用程序的框架。本章详细介绍了 Zope，以及和其它类似软件的不同之处。

1. 什么是 Web 应用程序？

网站内容需要及时的更新，尤其对于商业网站来说更是如此。网站中的网页用超级文本标记语言(HTML)编写而成。当用户访问网站的时候，实际上就是 把服务器上带有 HTML 的文本内容传送到用户的浏览器中，然后通过浏览器来解释成图文并茂的网页。当用鼠标点击链接时，就是开始传送一个新的网页。

一些网站是静态的。静态网站需要维护人员手工更新网站内容。更新内容就是手工 更新那些用 HTML 编写而成的网页文件，然后把这些文件放到服务器中。更新由静态网页组成的网站，需要编辑所有的文件，如果要更新的文件很多，更新就会很 繁琐。这样就很容易犯错误。为了提高网站内容维护的效率，就可以通过构建 Web 应用程序来解决问题。

Web 应用程序就是一种通过互联网能够让 Web 浏览器和服务器通讯的计算机程序。不 同于静态网站的，Web 应用程序动态创建页面。采用动态方式生成的 Web 站点通过 使用计算机程序来实现动态的特性。这种动态的应用程序可以用各种计算机语言来 编写。

动态构建的网站不需要维护管理人员一页一页的更新内容。动态网站可以把 HTML 部分和数据部分分离开，从而极大的提高网站维护和管理的效率。使用 Web 应用程 序的网站很多很多，比如：Google， SourceForge ，eBay，Hotmail 等等。

通常，允许人们构建 Web 应用程序的框架被称作 Web 应用服务器。Zope 就是一种 web 应用服务器，类似的竞争者比如：WebLogic ， Macromedia ColdFusion （ http://www.macromedia.com/)等等。Web 应用服务器一般通过某种计算机程序语 言来创建 web 应用程序，并且提供更多的功能，比如模板、安全模型，数据安全， 对话（session），以及其它更多的在构建 Web 应用程序时所需要的方便特性。

2. 如何通过应用服务器受益？

如果你想编写 web 应用程序，一般都需要使用应用服务器框架，除非是非常特殊的 应用程序。通过使用应用服务器框架可以充分利用已经编写好的各种服务程序， 而不需要像直接使用一种编程语言那样从头写起。许多应用服务器可以完成以下任务。

显示动态内容

你可以加入搜索特性。应用服务器可以提供动态生成内容的服务。应用服务器一般都可以个性化，并且结合数据库，以及搜索内容。

管理你的 Web 站点

应用服务器可以通过统一的方式管理站点中的数据、事务逻辑和显示。

构建一个内容管理系统

应用服务器提供构建内容管理系统的工具，从而可以让非技术编辑者可以创建和管理站点内容。

构建电子商务应用程序

应用服务器提供构建复杂的电子商务所需的框架。

安全的管理各种用户

网站中的不同用户需要不同的权限，应用服务器可以提供权限控制功能。

提供多种网络服务

支持网络服务的web站点可以处理来自其它计算机程序的请求。应用服务器正在逐步提供这样的功能。

结合多种系统

现有的内容可能来自于不同的地方，比如：关系数据库，文件，其它的站点等等。应用服务器可以把这些不同的数据整合在一起，提供统一的界面。

提供可扩展性

应用服务器可以根据服务器负载的情况来进行扩展。

Zope应用服务器可以完成上述所有功能。

3.Zope 历史

1996年，当时是Zope公司CTO和 Python领袖的Jim Fulton，为教授CGI程序起草讲 稿，尽管他的CGI编程的知识不算很多。Jim针对这门课程，以他自己的方式研究了 所有关于CGI方面的现存文档。在讲课返回的途中，Jim考虑传统的基于CGI的编程 环境中他不喜欢的方面包括：脆弱、缺乏面向对象和暴露Web服务器细节的方式。 从这些最初的沉思 开始，在返回的飞机中Jim写出了Zope的核心内容。

Zope公司（原名为Digital Creations）后来公布了三个用以支持Web出版的开放源 码软件包，分别为： Bobo、Document Template 和BoboPOS 。 这些软件包是用 Python编写的。它们发展成为Zope提供Web ORB (Object Request Broker)、DTML 脚本语言和对象数据库的核心组件。从那时起，Zope公司就开发了一套基于他们的 三个开放源码组件的商业性的应 用服务器。这个产品称为Principia 。在1998年 的11月，投资人Hadar Pedhazur 决定让Zope公司公开Principia的源码。于是就形 成了Zope，

“Zope”含义是指 Z 对象出版环境（Z Object Publishing Environment，Z没有特 别的含义）。Zope主要采用Python编写，其中与性能密切相关的部分采用C语言编 写。

4.为什么使用Zope？

比起其它 web 应用服务器，Zope 可以更好更快的创建 web 应用程序，这是因为 Zope 支 持以下特性：

Zope 是免费的，可以在开放源代码许可证条件下自由分发，不同于那些昂贵的商业应用服务器。

Zope 是一套完整的平台。它包含了开发应用程序所需的全部组件。不需要为了使用 Zope 而授权使用其它软件。并且 Zope 安装容易，轻松上手。

Zope 允许并鼓励第三方开发者打包和分发应用程序。因此，Zope 已经有了很多可以立即使用的产品组件。大多数组件都是自由并开放源代码的。Zope 拥有一大批社区开发者。

Zope 创建的应用程序可以直接通过 Zope 企业对象（ZEO）进行扩展。通过 ZEO,可以在多台计算机中部署 Zope 应用程序，而不需要修改代码。

Zope 允许开发者只使用浏览器就可以创建 web 应用程序。比如：Internet Explorer, Mozilla, Netscape, OmniWeb , Konqueror, 以及 Opera 浏览器都可以支持 Zope 的管理界面(ZMI).Zope 还可以通过使用统一的 web 界面让其他的开发者安全的同时进行开发。其它应用服务器很少支持这个特性。

Zope 提供多种和可扩展的安全框架。可以轻松结合多种权限认证系统，比如通过内置的模块可以同时支持 LDAP, Windows NT, and RADIUS。而许多其它应用服务器缺乏这些特性

Zope 可以让开发团队高效协同开发。协同环境可以让用户不会相互干扰，Zope 使用 Undo, Versions, History，以及其它工具来帮助人们一起工作，并且可以从错误中恢复过来。而其它大多数应用服务器不支持这些特性。

Zope 可以运行在大多数计算机操作系统平台中：Linux, Windows NT/2000/XP, Solaris, FreeBSD, NetBSD, OpenBSD, 和 Mac OS X.。Zope 甚至可以运行在 Windows 98/ME 中。而其它大多数应用服务器做不到这一点。

Zope 可以通过 Python 语言进行扩展。Python 很流行并且很容易学，可以促进快速开发。Python 中的许多功能库可以直接用于创建你的应用程序。而其它一些应用服务器使用不能快速开发的编译语言，比如 Java,或者使用不流行的语言。

用 Zope 创建的应用，请参考 Zope 公司的主页 Zope.com 中的案例分析页面。

5.Zope 的目标用户，以及 Zope 不适合做什么

管理大型站点的开发过程是件困难的事情。经常需要很多人一起工作来创建、部署和管理 web 应用程序。

信息架构者进行总的安排和控制

组件开发者创建可重用和分发的软件。

站点开发者结合现有的由组件开发者编写的软件，以及本地应用服务器提供的服务，构建应用程序。

站点设计者创建站点的外观和感觉

内容管理者创建和管理站点的内容

管理员维护软件系统运行

消费者使用站点来定位和使用有用的内容。

Zope 最适合组件开发者、站点管理者和站点设计者，并且这三种用户通过 Zope 提 供的服务和第三方产品可以一起协同开发应用。典型的情况是内容管理者和使用 者在系统架构者的指导下开发应用。管理员部署和维护应用程序。

Zope 是一种 web 应用构建的框架，不同水平的程序员都可以使用 Zope 来创建基于 web 的应用程序。Zope 不是一种现成的应用程序。它不是 weblog、内容管理系统或是 一种电子购物程序。

基于 Zope 的各种产品可以完成这样的功能。到目前为止，Zope.org 站点中已经有 了很多种可用于你的应用程序的产品。这些产品包括 Weblog，内容管理，以及电 子商务程序等等。

Zope 不是一种可可视化的设计工具，不同于 Macromedia Dreamweaver 或者 Adobe GoLive 这样的软件。你可以使用这些软件来管理基于 Zope 的 web 站点，但是不能用 Zope 来替代这些界面设计软件。

6. Zope 的使用条款，许可证，以及 Zope 社区

Zope 是免费的。你可以用 zope 创建和运行 web 应用程序，而不用支付费用，并且还 可以在你的产品中置入 zope 而不用给 Zope 公司支付使用费。分发 Zope 需要遵守的许可证是一种开放源码许可证，即 Zope Public License 或 ZPL。ZPL 条款中规定你可以获得和修改 Zope 的源代码。

ZPL 不同于 GNU Public License（另外一种比较流行的开放源代码许可证）。如 果你试图重新分发遵守 GPL 许可证的应用程序，并且你修改或扩展了应用程序，GPL 要求所做的贡献属于许可证颁发者。而对于遵守 ZPL 的应用程序就没有这样的要求。ZPL 已经得到开放源代码机构的认可，获得了 OSD 认证,另外还得到自由软件基金会的认 可，兼容于 GPL 许可证。

Zope 开发者社区负责维护或扩展 Zope 应用服务器。社区中的很多成员是专业咨询顾 问、专业开发者和 Web 精通者，他们使用 Zope 开发应用程序。另外，用户中还有学生以及站点开发爱好者。Zope 公司也是社区成员之一，主要负责维护 Zope 以及开发 Zope 代码。 Zope 社区通过聚会以及邮件列表和站点进行交流。在 Zope.org 的邮件列表页面，可 以找到更多的信息。

Zope 公司通过多种方式获得收入，包括为商业用户创建 web 应用程序，培训 Zope 开发 者，为使用 Zope 的公司提供技术支持，以及主机服务。Zope 公司不从 Zope 服务器 的销售中获得收益。

第二章 Zope 概念和架构

1. 基本概念

Zope 框架有一些基本概念，理解了这些概念有助于充分使用 Zope。

1.1.Zope 是一种框架

Zope 涵盖了很多 Web 应用程序开发者需要处理的底层细节，比如数据的持续性，数 据的完整性，数据访问控制等等，这样就可以让你集中精力在解决问题上。比起 其它的语言或框架，Zope 可以让你充分利用 Zope 提供的服务来更快速的构建 web 应

用程序。Zope 可以让你使用 Python 语言来编写 web 应用程序中的逻辑处理部分，当 然也可以用 Perl。Zope 还提供两种方式，就像模板一样，来处理文本、XML 和 HTML 这样的数据，一种式文本模板标记语言（DTML）和 Zope 页面模板（ZPT）。

1.2.面向对象

不同于基于文件的 Web 模板系统，比如 ASP 或 PHP，Zope 是高度面向对象的 Web 开发平 台。许多语言都支持面向对象的概念包括编写 Zope 的 Python 语言。常见的 Web 脚 本语言比如 Perl 或 PHP 部分支持面向对象的特性，通过阅读“面向对象”这一章可以 帮助深入理解这个概念，也可以通过本书中提供的例子来深入理解这个概念。

1.3.对象出版

Zope 之所以形成，其中一个基本的理念是：Web 的基础是面向对象的。指向某个 Web 资源的 URL 实际上就是对象容器中对象的路径。HTTP 协议提供了一种对象发送 消息和接收回应的方法。

Zope 的对象结构是分层次的，就是说典型的 Zope 站点是由对象组成的，对象又有 可能包含其他对象。根据对象的名称，Zope 按照层次结构通过 URL 映射到对象。比 如，URL "/Marketing/index.html"可以用来访问文件夹对象"Marketing"中的名为"index.html"的文档对象。

Zope 就是以这样一种直接的方式“出版”你所创建的对象。基本过程如下：

浏览器给 Zope 服务器发送请求。请求的 URL 格式为：protocol://host:port/path?querystring， 比如:

http://www.zope.org:8080/Resources?batch_start=100

1. Zope 把 URL 分解成："host（主机）"，"port（端口）" "path（路径）" 和 "query string（查询参数）"。(http://www.zope.org, 8080, /Resources 和 ?batch_start=100, respectively)

Zope 根据路径（path，即/Resources）在对象数据库中定位对象。

Zope 用传递过来的参数来执行这个对象。

如果对象执行的结果返回数值，那么数值就被发送回浏览器。一般是返回 HTML，文件数据或图形数据。

浏览器解释收到的数据并显示。

Zope 对象的 URL 由包含这个对象的文件夹和对象的 id 组成，用/符号分开。比如： /Uncles/Bob，就调用 Uncles 目录中的 Bob。

还比如：

/Uncles/Rick

/Uncles/Danny

/Uncles/Louis

完整的URL就可以是： http://localhost:8080/Bob 。 更为详尽的解释请参考：Zope 开发指南中的对象出版一章 Object Publishing

1.4.通过 Web 进行管理

Zope 可以通过完全通过浏览器来创建和处理各种对象。Zope 提供的管理界面就像 Windows 中的资源管理器。 对象可以按照层次放在任何地方，站点管理者通过点击对象的不同视图来管理对 象。不同的对象有不同的视图。比如“DTML Method”对象有一个标有“Edit”的视 图，其中可以编辑代码，“数据库链接对象”（Database Connection）提供修改连 接和参数的视图。所有的对象都有一个“安全”（Security）视图，用于管理访问权 限控制。

1.5.安全与授权

Zope 区别于其它应用服务器的一个显著特点是 Web 对象模型和 Web 开发模型紧密结 合。从而可以让许多不同的人都可以参与进来， Zope 允许对不同的用户进行安全 授权，从而可以让页面设计者，数据库管理员，以及内容管理员协同工作。

成功的 Web 站点需要许多人共同参与，比如应用开发者，SQL 管理员，内容管理员， 甚至是最终的用户。此时，安全问题就变得及其重要。如何控制，以及如何分配 权限？比起传统的基于文件的系统，Zope 中的对象提供了丰富得多的安全许可。不同的对象可以有不同的安全限制，比如对于“SQL Method”对象，你可以允许用户调用 它，但不能更改或查看源代码。你还可以限制用户只能创建某种类型的对象，比如 只能创建文件夹或 DTML 文档，而不能创建“SQL Method”对象。

Zope 通过“用户文件夹”（"User Folders"）来管理用户。在这个特殊的文件夹中包 含用户信息。也可以通过添加扩展包来扩展用户文件夹，从而可以通过关系型数 据库或 LDAP 目录来管理。添加新用户文件夹的权限可以分派给下级文件夹中的用 户，从而可以让你认可的用户来管理网站中的某一部分。

1.6.本地对象持续和事务（Persistence and Transactions）

Zope 对象存储在一种高性能的支持事务机制的对象数据库中，即 Zope 对象数据库 （ZODB）.对象数据库认为每个 Web 请求是个单独的事务。在 Web 请求期间，如果执 行过程中发生了错误，任何所做的更改都将被取消。对象数据库还支持多级撤销， 这样就可以让站点管理员仅仅通过点击“undo”按钮撤销更改。Zope 框架中实现对 象持续和事务的所有方式对开发者都是透明的。关系型数据库在 Zope 框架中依然有效。

1.7.获取（Acquisition ）

“获取”（Acquisition）是 Zope 中非常重要的一个概念，这个概念简单的说就是：

Zope 对象可以被包含在其它对象中（比如文件夹）。

对象可以“获取”它们的容器对象的属性和行为。

所有的 Zope 对象都支持获取，这样就提供了管理各种资源的非常强大的方式。比 如，经常使用的 SQL 查询语句或者一小段 HTML 代码，可以在某个文件夹中定义，通 过”获取“这样一种机制就可以让下级文件夹自动调用。如果这个 SQL 查询进行了修改，不用担心下级文件夹，这个修改对所有下级文件夹都有效。

因为在搜索对象的时候是按照从当前目录往上按照层次来获取的，因此很容易指定 生效的范围。比如，有一个包含与体育内容相关的文件夹“Sports”，你可以在这 个文件夹中创建新的页眉和页脚文件。这样就可以使“Sports”文件夹和下级文件夹 中调用这两个文件。而不会调用“Sports”上级文件夹中的文件。

在“获取机制”一章中将详细讲述这一概念

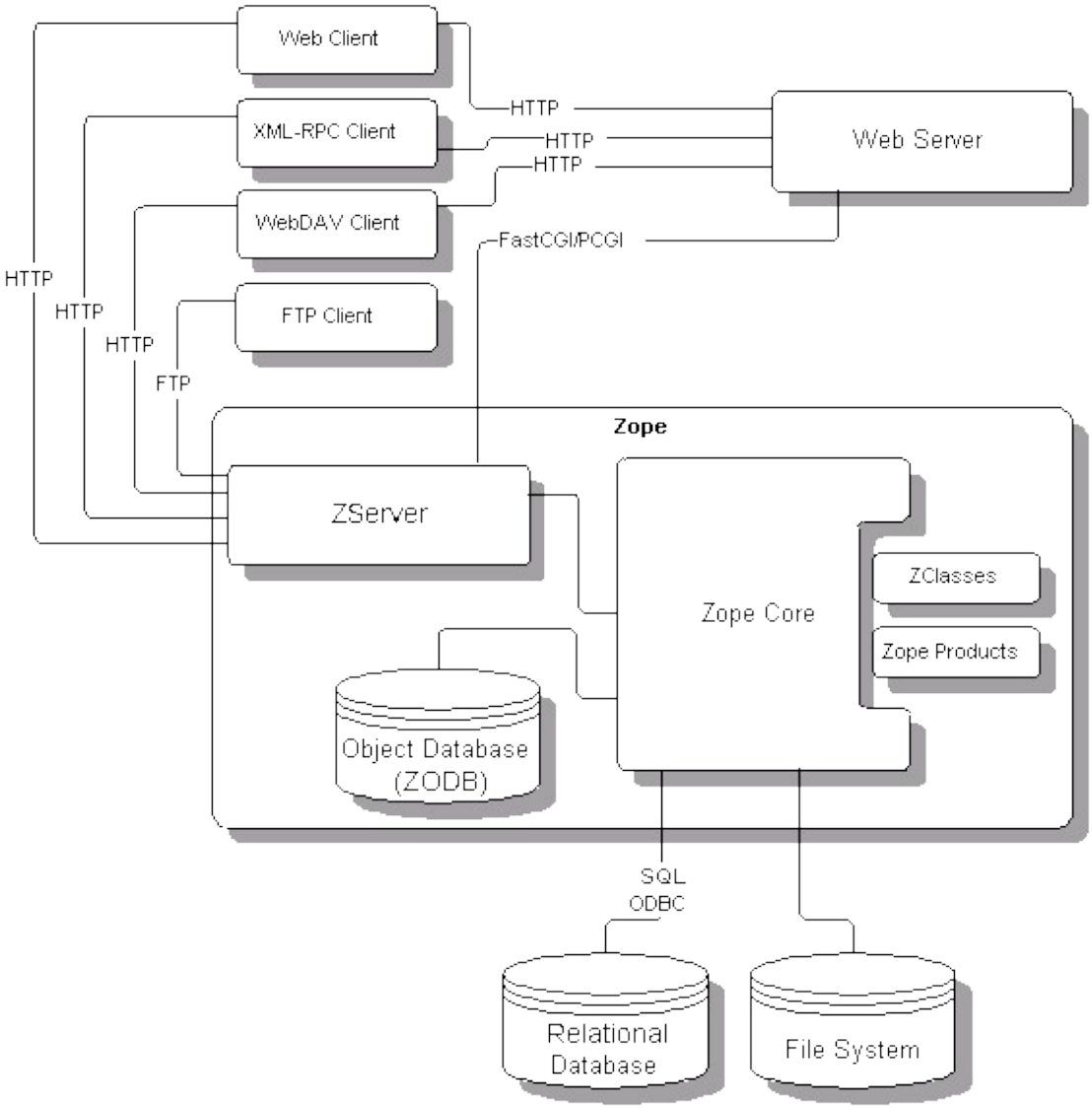
1.8.Zope 是可扩展的

Zope 是高度可扩展的，可以通过多种方法创建新的 Zope 对象，即可以通过用 Python 语言编写新的扩展模块，也可以完全通过 Web 来创建。Zope 已经包含了很多内置的 组件，这些组件可以帮助进行扩展。Zope 当中已经包含了一系列框架类，这些类在创建新 Zope 对象时用来处理细节问题。

Zope 已经有了很多扩展产品，这些扩展产品用于增强 Zope 的功能，比如添加论坛、 数据处理、XML 工具，以及电子商务程序等等。这些程序大多数是由 Zope 爱好者编 写的，并且大多数是开放源代码的。

2.基础 Zope 组件

Zope 由一些不同的组件构成，从而帮助你构建 web 应用程序。下图显示了这些基础 Zope 组件：



Zope 架构

说明：

ZServer

这是 Zope 内置的 Web 服务器，用于提供内容服务。这个 web 服务器还通过 FTP、 WebDAV 和 XML-RPC 协议提供服务。

Web Server（Web 服务器）

Zope 还可以和现有的 Web Server 结合在一起，比如 Apache 或者 Microsoft IIS，以及其它支持通用网关接口（CGI）的 Web 服务器。

Zope Core（Zope 核心）

这是 Zope 的核心引擎，它控制如何显示，以及控制管理界面和对象数据库。

Object Database（对象数据库）

使用 Zope 的时候，大多数情况下是在处理存储在 Zope 对象数据库中的对象。

Relational database（关系型数据库）

数据不一定要存储在 Zope 对象数据库中，Zope 中可以使用其它的关系型数据库，比如 Oracle, PostgreSQL, Sybase, MySQL 等等。

File System（文件系统）

Zope 还可以使用存储在服务器文件系统中的文档和其它文件。

ZClasses

可以通过使用 Web 管理界面来添加新的对象。ZClasses 就是这种对象。

Products（产品）

Zope 中还可以通过在 Zope 服务器中的文件系统里安装“产品”文件来添加新的对象。

第三章 安装和使用 Zope

Zope 可以安装在大多数的操作系统中，并且很容易，一般不会超过 10 分钟。

1. 下载 Zope

Zope 公司提供已经编译好的 Zope 二进制文件，包括 Windows、Linux 和 Solaris 操作系统。这些文件不需要编译。通常，Zope 分为两种版本，“稳定版”和“开发版”。一般使用“稳定版”就可以。 你可以通过 Zope.org 站点中的下载部分下载 Zope 安装文件。 如果没有编译好的安装文件，可以下载源代码，然后进行编译。Zope 可以在大多数 的类似 Unix 的操作系统中编译。编译需要 Python 和 C 编译器。

2. 安装 Zope

根据操作系统的不同， Zope 采用不同的安装步骤。

2.1.在 Windows 中安装 Zope

先从 Zope.org 站点的下载部分下载安装文件，名称通常为"Zope-2.X.X-win32-x86.exe"，其中 X 是指版本号。 注意，首次安装不要使用"Zope -2.X.X-to-2.X.X-win32.x86.tgz"这样的文件，这种文件是用来从已有版本升级到新版本的。

稳定版 Windows 安装文件

下载后，运行这个安装文件。它会带你完成整个安装过程。

开始安装

点击“Next”，阅读完许可证并接受后，点击“Next”，选择一个“site name”，也可以使用默认的名称“ WebSite ”，然后点击“Next”。 选择安装目录后，点击“Next”。输入初始用户的名称和密码，这个用户用于首次以管理员身份登录 zope。安装完以后，可以更改这个帐户。常见的 初始用户名是 admin。

选择站点名称

选择安装目录

输入初始用户名和密码

点击“Next”两次以后，开始安装。

安装文件

复制文件完成以后，如果你使用 Windows NT,2000 或 XP，会提示你选择是否以服务 方式运行 Zope。如果你运行 Zope 只是个人用，选择可以随意。在 Windows 95,98 或 ME 中，不能以服务方式运行。

服务选项

点击“Next”，最后点击“Finish”。安装完以后可以通过 Unwise.exe 卸载 Zope。

2.2.在 Linux 和 Solaris 系统中安装 Zope

在 Linux 和 Solaris 中安装 Zope 非常类似。安装文件是.tgz 格式，以因此必须先解压缩。 注意，首次安装不要使用″Zope-2.X.X-to-2.X.X-platform.tgz**″ 这样的文件，这种文件是用来从已有版本升级到新版本的。 在 Linux 或 Solaris 中安装 Zope 以前，需要检查一下系统是否安装了″GNUtar″ 和 ″gunzip″，如果是 Linux 系统，一般默认都已经安装好了。在 Solaris 系统中，

必须 使用GNUtar，而不是自带的tar程序。 从Zope.org中下载安装程序，名称一般为"Zope-2.X.X-solaris- sparc.tgz"（适用于 Solaris）or "Zope-2.X.X-linux2-x86.tgz"（适用于 Linux），其中的X是指 当前的版本号。

在开始安装以前，需要确定安装目录和用户。建议用普通用户（就是除了root以外 的其他用户）来解压缩和安装Zope。以下假 设在“home”下面你个人的主目录中进 行安装。 以下演示了用wget工具来下载安装文件的过程，当然也可以通过web浏览器来下载：

```
chrism@saints:~$ wget http://www.zope.org/Products/Zope/2.5.1/Zope-2.5.1-linux2-x86.tgz

--20:27:56--  http://www.zope.org:80/Products/Zope/2.5.1/Zope-2.5.1-linux2-x86.tgz

          => `Zope-2.5.1-linux2-x86.tgz.1'

Connecting to www.zope.org:80... connected!

HTTP request sent, awaiting response... 200 OK

Length: 5,979,458 [application/x-gzip]

    0K -> ......... ......... ......... ......... .........  [  0%]

   50K -> ......... ......... ......... ......... .........  [  1%]

(..and so on..)
```

Zope安装程序和其它UNIX程序有所不同，它不区分"build"目录和"install"目 录。下载解压缩之后的目录就是安装目 录。在下面的例子中，我们解压缩和安装 所使用的目录是/home/chrism/Zope-2.5.1-linux2-x86. 下载完成后，进入你的主目录，通过gunzip和GNUtar解压缩文件:

```
chrism@saints:~$ gunzip -c Zope-2.5.1-linux2-x86.tgz | tar xvf -

Zope-2.5.1-linux2-x86/

Zope-2.5.1-linux2-x86/Extensions/

Zope-2.5.1-linux2-x86/Extensions/README.txt

Zope-2.5.1-linux2-x86/LICENSE.txt

Zope-2.5.1-linux2-x86/README.txt

(.. and so on..)
```

进入这个目录，然后运行安装脚本程序：

```
chrism@saints:~$ cd Zope-2.5.1-linux2-x86

chrism@saints:~/Zope-2.5.1-linux2-x86$ ./install

------------------------------------------------------------------------

Compiling python modules

------------------------------------------------------------------------

------------------------------------------------------------------------

creating default inituser file

Note:

        The initial user name and password are 'admin'

        and 'tnLQ6imA'.

        You can change the name and password through the web

        interface or using the 'zpasswd.py' script.

chmod 0600 /home/chrism/Zope-2.5.1-linux2-x86/inituser

chmod 0711 /home/chrism/Zope-2.5.1-linux2-x86/var

------------------------------------------------------------------------

setting dir permissions

------------------------------------------------------------------------

creating default database

chmod 0600 /home/chrism/Zope-2.5.1-linux2-x86/var/Data.fs

------------------------------------------------------------------------

Writing the pcgi resource file (ie cgi script), /home/chrism/Zope-2.5.1-linux2-x86/Zope.cgi

chmod 0755 /home/chrism/Zope-2.5.1-linux2-x86/Zope.cgi
```

```
-------------------------------------------------------------------

Creating start script, start

chmod 0711 /home/chrism/Zope-2.5.1-linux2-x86/start

-------------------------------------------------------------------

Creating stop script, stop

chmod 0711 /home/chrism/Zope-2.5.1-linux2-x86/stop

-------------------------------------------------------------------

Done!

chrism@saints:~/Zope-2.5.1-linux2-x86$
```

注意，在安装过程中，将创建初始用户帐号，包括用户名和密码。这个帐号用于登 录系统。以后可以通过 zpasswd.py 程序来修改这个帐号（详细内容参见“用户和安 全”一章）。 关于安装过程中的详细说明可以参见 doc 目录中的 INSTALL.txt 文件，也可以通过 －h 参数来查看帮助信息：

```
$ ./install -h
```

2.3.通过源代码编译和安装 Zope

如果没有适合你所使用系统的编译好的二进制文件，那么也可以从源代码进行编译 安装。基本要求是：

确保系统中有“C”编译器（最好是 GNU gcc）

确保系统中有“make”工具软件（最好是 GNU make）

通过源代码安装过 Python 语言

Zope 主要采用 Python 编写而成，因此要求系统中 必须安装有 Python 语言。编译好 的二进制安装文件中已经包含了所需的 Python 文件，但源代码安装文件中不提供 Python。Zope2.5 和 2.6 版本中需要 Python2.1.3，Zope2.3 以前的版本需要 Python1.5.2。Zope2.7 开始使用 Python2.3.2。

从 Python.org 网站中可以获得安装文件。有些系统中，已经安装好了 Python，但 并不一定适合 Zope，因此如果你想从源代码编译安装 Zope，建议最好先从源代码安 装 Python。 安装好 Python 以后，用 wget 或浏览器下载 Zope 源代码安装文件，比如：

```
chrism@saints:~$ wget http://www.zope.org/Products/Zope/2.5.1/Zope-2.5.1-src.tgz
```

```
--20:49:34--  http://www.zope.org:80/Products/Zope/2.5.1/Zope-2.5.1-src.tgz
      => `Zope-2.5.1-src.tgz'
Connecting to www.zope.org:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 2,165,141 [application/x-gzip]
   0K -> ......... ......... ......... ......... ......... [  2%]
  50K -> ......... ......... ......... ......... ......... [  4%]
 100K -> ......... ......... ......... ......... ......... [  7%]
(..and so on..)
```

然后，同样，解压缩：

```
chrism@saints:~$ gunzip -c Zope-2.5.1-src.tgz | tar xvf -
```

然后执行以下命令，其中"wo_pcgi.py"中的"wo_pcgi" 指的是不支持 "PCGI"。

```
chrism@saints:~$ cd Zope-2.5.1-src
chrism@saints:~/Zope-2.5.1-src$ python2.1 wo_pcgi.py
------------------------------------------------------------------------
Deleting '.pyc' and '.pyo' files recursively under /home/chrism/Zope-2.5.1-src...
Done.
------------------------------------------------------------------------
Compiling python modules
------------------------------------------------------------------------
Building extension modules
cp ./lib/python/Setup20 ./lib/python/Setup
```

```
-----------------------------------------------

Compiling extensions in lib/python

cp /home/chrism/lib/python2.1/config/Makefile.pre.in .

make -f Makefile.pre.in boot PYTHON=

rm -f *.o *~

rm -f *.a tags TAGS config.c Makefile.pre python sedscript

rm -f *.so *.sl so_locations

VERSION=` -c "import sys; print sys.version[:3]"`; \

(..and so on until...)

-----------------------------------------------------------------

creating default inituser file

Note:

        The initial user name and password are 'admin'

        and 'w!YzlsDT'.

        You can change the name and password through the web

        interface or using the 'zpasswd.py' script.

chmod 0600 /home/chrism/Zope-2.5.1-src/inituser

-----------------------------------------------------------------

Done!
```

注意，在安装过程中，将创建初始用户帐号，包括用户名和密码。这个帐号用于登 录系统。以后可以通过 zpasswd.py 程序来修改这个帐号（详细内容参见“用户和安 全”一章）。

3.开始使用 Zope

Zope 通过 web 浏览器进行管理，Zope 内置的 ZServer 默认在 8080 端口监听 HTTP 请求。 如果 Zope 没有正常启动，有可能是因为 8080 端口已经被其它程序占用。Zope 还可 以监听其它 TCP 端口，包括 FTP(文件传输协议)，“monitor”（内部调试）,WebDAV （Web 分布式编著），以及 ICP（Internet 缓存协议）。Zope 在启动的时候还会显示 这些信息。

3.1.与现有 Web 服务器结合

Zope 可以独立提供 Web 服务，也可以和现有的 Web 服务器结合在一起，共同提供内容 服务。Zope 可以结合 Microsoft IIS 、Apache 以及其它流行的 web 服务器。并且提供 多种结合方式。请参见“虚拟主机服务”和“相关资源”一章。

3.2.在 Windows 中启动 Zope

通过运行 Zope 安装目录中的 start.bat 文件就可以启动 Zope。

如果在 Windows NT/2000/XP 系统中安装的时候选择了以“服务”的方式运行 Zope，就 可以通过 Windows 中的“服务”控制面板来控制 Zope。这种方式的好处在于通过系统 日志来跟踪 Zope 的状态。当然也可以用 start.bat 文件来手工启动。

3.3.在 UNIX 系统中启动 Zope

进入 Zope 安装目录，运行脚本程序"start"，比如：

```
chrism@saints:~$ cd Zope-2.5.1-linux2-x86

chrism@saints:~/Zope-2.5.1-linux2-x86$ ./start

------

2002-06-28T03:17:02 INFO(0) ZODB Opening database for mounting: '142168464_1025234222.179125'

------

2002-06-28T03:17:02 INFO(0) ZODB Mounted database '142168464_1025234222.179125' at /temp_folder

------

2002-06-28T03:17:17 INFO(0) Zope New disk product detected, determining if we need to fix up any ZClasses.

------

2002-06-28T03:17:17 INFO(0) ZServer HTTP server started at Thu Jun 27 23:17:17 2002

        Hostname: saints

        Port: 8080
```

```
------

2002-06-28T03:17:17 INFO(0) ZServer FTP server started at Thu Jun 27 23:17:17 2002

        Hostname: saints

        Port: 8021

------

2002-06-28T03:17:17 INFO(0) ZServer PCGI Server started at Thu Jun 27 23:17:17 2002

        Unix socket: /home/chrism/Zope-2.5.1-linux2-x86/var/pcgi.soc
```

3.4.以 Root 用户身份启动 Zope

在 POSIX 系统中，ZServer 支持 setuid()，从而只允许 root 用户使用 21(FTP)和 80 (HTTP)端口。 也就是说不一定需要以 root 身份来启动 Zope，只有要监听低端口时才需要这么做。 当需要监听低端口时，需要以 root 用户身份来启动 z2.py 文件，并且要指定用户。 方法是在 start 脚本或 z2.py 文件命令行中加入 -u 选项，并加上用户名或 UID。默 认是“nobody”，如果其它进程也在使用 nobody，有可能会影响 Zope 安全。

另外，还需要确认 var 目录的所有者是 root，命令是：chown root var，另外还需 要设置 sticky bit，命令是：chmod o+t var。一个目录设置好 sticky bit 后，任 何人都可以写入文件，但是不能变更其他人的文件。这样就避免其他用户覆盖掉 PID 文件，从而别人不会通过运行 stop 命令终止 zope 的运行。

4. 使用 Zope

要使用和管理 Zope，只需要 web 浏览器。Zope 的管理界面都用 HTML 代码组成，任何 一种支持 HTML 代码的比较新的版本的浏览器都可以对 Zope 进行管理。比如 Mozilla，或任何一种高于 3.0 版本的 Micorsoft Internet Explorer 或 Netscape Navigator 都可以。其它的还有：Opera, Galeon, Konqueror, OmniWeb , Lynx, 和 W3M。 如果 Zope 安装在本机上，可以访问： http://localhost:8080/， 然后就可以看到 ″ QuickStart ″屏幕：

Zope QuickStart

看到这个屏幕，就说明 Zope 已经成功安装。否则，请参考下边的“解决故障”一节。

4.1.登录

Zope 通过管理界面来进行管理，方法是输入管理 URL。比如，zope 安装在本机，则 管理 URL 为：http://localhost:8080/manage 然后就会弹出一个对话框，输入安装 Zope 时用到的初始用户名和密码，然后就进入 了 Zope 管理界面（ZMI）.

Zope 管理界面（ Zope management interface）

4.2.通过控制面板控制 Zope 的进程

使用 ZMI 的时候，可以通过 Zope 的控制面板（/Control Panel/）来控制 Zope 的进 程。点击*Control_Panel：*

控制面板（ Control Panel）

控制面板中会显示一些基本信息，比如版本号、安装目录等等，还有一些相关链接。 如果 Zope 在 UNIX 这样的系统中运行，或在 Windows 中以服务方式运 行，会有一个 /Restart/.按钮，用于重新启动 Zope。重新启动 Zope 一般需要几秒钟的时间，此时 不需要关闭浏览器。 终止 Zope 的运行，可以点击“Shutdown”按钮。再启动只能通过手工方式进行。如 果在这个过程中看到以下这些信息，说明终止过程正常。

```
An error was encountered while publishing this resource

exceptions.SystemExit

Zope has exited normally

( .. more output ..)
```

4.3.通过命令行控制 Zope 进程

在 Windows 系统中，可以通过"Ctrl-C"键来关闭 Zope 控制窗口，从而终止 Zope 的运 行。如果 Zope 以服务的方式来运行，可以进入 Windows 的服务控制面板，手工终止 Zope 服务。 在 Unix 系统中，通过在终端窗口中使用"Ctrl-C"键或使用 kill 加上 zope 进程 id 号来 终止运行。进程 id 号可以通过 ps 命令找到，也可以在 var/Z2.pid 文件中找到。

4.4.解决故障

如果浏览器不能在 TCP 的 8080 端口连接 Zope，说明有可能运行在其它端口，找到端 口号的方法是查看 Zope 启动时提供的信息。比如，Zope 启动的时候显示以下信息：

```
------

2000-08-07T23:00:53 INFO(0) ZServer Medusa (V1.18) started at Mon Aug  7 16:00:53 2000

        Hostname: peanut

        Port:9673

------

2000-08-07T23:00:53 INFO(0) ZServer FTP server started at Mon Aug  7 16:00:53 2000

        Authorizer:None

        Hostname: peanut

        Port: 8021

------

2000-08-07T23:00:53 INFO(0) ZServer Monitor Server (V1.9) started on port 8099
```

第一组数据显示 Web 服务所使用的端口是 9673，因此管理 URL 为： http://peanut:9673/manage/ Microsoft Internet Explorer 5.0.1 和 5.5 的某些版本可能会不能登录到管理界面，试试其他版本或升级到 IE6。 如果忘记了初始用户名和密码，可以先终止运行 Zope，然后使用 zpasswd.py 程序来 修改初始用户名和密码。详细可参见“用户和安全”一章。

4.5.Zope 的启动选项

Zope 启动脚本在 UNIX 系统中为 start，在 windows 中为 start.bat，但启动选项是 一样的。详细解释如下：

-h

显示帮助文本

-z path

指定 Zope 安装目录，默认是 z2.py 文件所在的目录

-Z path

只适用UNIX系统!

如果指定了这个选项，将创建另外一个进程，这个进程会重新启动Zope。

参数path必须指向一个记录进程id信息的pid文件。

路径可以是相对路径，相对于zope安装路径。

要不使用另外的进程，可使用空字符串：-Z=''

-t n

指定线程数，默认值是4

-i n

设定解释器检查间隔。这个整数决定了解释器如何检查间隔的事件。

比如线程切换数和信号句柄。默认值是500。适当调整这个值可增加性能。

-D

以调试模式运行Zope。这个选项可以让Zope进程不与控制终端分开。

其作用和设定环境变量 Z_DEBUG_MODE=1 一样。

-a IP地址

要监听的IP地址。如果这个IP地址为空（比如，-a ''），则对本机中所有

IP地址有效。

-d IP地址

DNS服务器的IP地址。如果为空字符串，则日志中不记录这个IP地址。如果

在本机中有DNS服务，可设定值为127.0.0.1

-u 用户名或uid号

运行Zope的用户名。可直接指定。

这个选项只在Unix中有效。如果以root身份启动Zope，这个参数是必须要指定的。

-P [IP地址:]number

以偏移量方式同时设定web、ftp和监控端口号。web端口是 number+80。

FTP端口号是number+21。监控端口号是number+99。

参数number可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听不同的地址。

也可以使用多个 -P 选项来运行多个服务。

-w port

指定Web服务（HTTP）端口号。默认为8080。如果port是"-"符号(比如： -w -)，那么HTTP服务失效。

参数可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听不同的地址。

使用多个-w选项可以运行多个服务。

-W port

指定WebDAV端口。如果port是"-"符号(比如： -W -)，那么WebDAV失效。

参数可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听不同的地址。

使用多个-W选项可以运行多个服务。

-C

--force-http-connection-close

这个选项用于关闭所有的HTTP连接。

-f port

指定FTP端口。如果port是"-"符号(比如： -f -)，那么FTP失效。默认为8021。

参数可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听不同的地址。

使用多个-f选项可以运行多个服务。

-p path

指定PCGI源文件。默认值是var/pcgi.soc，相对于Zope安装路径。如果为符号"-"，即(-p -) 或文件不存在，则PCGI无效。

-F path_or_port

为FastCGI服务指定端口号（适用于inet sockets）或路径名称（适用于unix domain sockets）。如果这个选项没有指定，则FastCGI无效。

-m port

指定安全监控服务端口号。如果为符号"—"，则监控服务失效。监控服务能够以交互方式运行ZServer。要访问这个服务，可见medusa/monitor_client.py或 medusa/monitor_client_win32.py。监控服务密码和'access'文件中指定的紧急用户密码一样。默认情况下不启动监控服务。

参数可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听不同的地址。

使用多个-m选项可以运行多个服务。

--icp port

指定ICP端口。如果使用支持ICP的前端代理服务器，比如Squid，ICP可用于在后端运行多个Zope时分布负载。

参数可以通过在IP地址后加上冒号来指定IP地址。这样可以实现不同的服务监听

不同的地址。

使用多个--icp 选项可以运行多个服务。

-l path

指定 ZServer 日志文件路径，如果为相对路径，则写入'var'目录。

默认值是'var/Z2.log'

-r

以只读模式运行 ZServer。不在磁盘中写入。

-L

打开本地支持功能。如果使用空字符串（-L ''），Zope 将使用默认的设置。

-X

用于关闭服务，这个选项可使所有默认或旧的服务设置设置失效。比如，

仅仅打开一个 Web 服务：

./start -X -w80

-M file

指定日志信息存储文件。

4.6.环境变量

Zope 在运行的时候还可以通过操作系统的环境变量的设置来调整。 在 UNIX 中，要设置操作系统环境变量，可以使用“export”命令，比如：

export |EVENT_LOG_FILE=/home/chrism/Zope/var/event.log

要在 Windows NT/2000 中设置环境变量，可以使用控制面板->系统，进行设置，或 者通过 DOS 命令“set”，比如：set EVENT_LOG_FILE=c:\chrism\Zope\var\event.log 以下是影响 Zope 运行的环境变量：

Zope 库文件路径

PYTHONPATH

Python 路径，参见"The

Python Tutorial Modules

Chapter":http://www.python.org/doc/current/tut/node8.html

INSTANCE_HOME

如果定义了 INSTANCE_HOME 变量，并有子目录'lib/python'，将被添加到 PYTHONPATH 前边。

INSTANCE_HOME 用于把核心 Zope 文件和第三方模块或产品分开。

参见: SOFTWARE_HOME

SOFTWARE_HOME

SOFTWARE_HOME 通常用于存放 Zope 核心文件安装目录

参见: INSTANCE_HOME

ZOPE_HOME

ZOPE_HOME 是 Zope 的根目录，包括 ZServer 模块、z2.py，以及默认的导入目录等等。

轮廓（Profiling）

PROFILE_PUBLISHER

如果设定了这个变量，Zope 将记录下每个 ZServer 请求。信息将存储在 PROFILE_PUBLISHER 所提供的变量值中。

访问规则与站点根目录（Access Rules and Site Roots）

SUPPRESS_ACCESSRULE

如果设定这个值，则禁止所有根目录操作

SUPPRESS_SITEROOT

如果设定这个值，则禁止所有的访问。

ZEO 相关的

CLIENT_HOME

CLIENT_HOME 允许 ZEO 客户端保持独立的 pid 和日志文件。这是一个用于试验性质的特性。

ZEO_CLIENT

如果需要持续性的客户端缓存，则需要定义环境变量 ZEO_CLIENT，

从而为这个客户端提供唯一的名称。

调试和日志

EVENT_LOG_FORMAT 或 STUPID_LOG_FORMAT

如果想规定 Zope 事件日志输出格式，则需要设定这个变量。

EVENT_LOG_FILE="path" 或 STUPID_LOG_FILE="path"

指定事件日志文件

参见：Zope 安装目录 doc 子目录中的 LOGGING.txt

EVENT_LOG_SEVERITY <number> or STUPID_LOG_SEVERITY <number>

如果设定这个变量，则只记录 serverity 高于指定数字的事件

ZSYSLOG="/dev/log"

设定这个变量，将把事件日志信息写入到 UNIX domain socket 中(一般为'/dev/log')，只适用 UNIX。

参见：LOGGING.txt

ZSYSLOG_FACILITY="facilityname"

设置这个变量可以让 Zope 使用 syslog logger，只适用 UNIX。

参见：Zope 安装目录中 doc 子目录中的 LOGGING.txt

ZSYSLOG_SERVER="machine.name:port"

这个变量用于通过 UDP socket 进行连接。

参见：doc 目录中的 LOGGING.txt。

ZSYSLOG_ACCESS="/dev/log"

ZSYSLOG_ACCESS_FACILITY="facilityname"

ZSYSLOG_ACCESS_SERVER="machine.name:port"

把基本访问信息发送到 syslog

Z_DEBUG_MODE "yes" 或 "no"

是否以调试模式运行 Zope。等同于“start”启动时的 -D 选项。

其它

Z_REALM "your realm"

设定当发送认证请求到 web 客户端时所显示的域名称。

与安全相关的

ZOPE_SECURITY_POLICY

如果这个变量设为"PYTHON"，将使用基于 python 的权限控制方式。

出于性能的考虑，默认所采用的是 cAccessControl 模块。

ZSP_OWNEROUS_SKIP

如果设定这个参数，则会省略与所有权相关的检查。

ZSP_AUTHENTICATED_SKIP

如果设定这个参数，则会省略与认证相关的检查。

ZOPE_DTML_REQUEST_AUTOQUOTE

用于设置处理 DTML 的方式

ZODB 相关

ZOPE_DATABASE_QUOTA

设定 ZODB 最大数（整数 byte）

ZOPE_READ_ONLY

设定为只读模式

Session 相关

ZSESSION_ADD_NOTIFY

设置启动时瞬时对象容器创建时对象添加所调用的对象路径

ZSESSION_DEL_NOTIFY

设置启动时瞬时对象容器创建时对象删除所调用的对象路径

ZSESSION_TIMEOUT_MINS

设置"/temp_folder/session_data"中对象失效时间长度

ZSESSION_OBJECT_LIMIT

设置"/temp_folder"中 session 数据最大条目数量值

WebDAV

WEBDAV_SOURCE_PORT_CLIENTS

这个变量能够支持通过标准的 HTTP 端口获得源文件。

结构化文本

STX_DEFAULT_LEVEL

设置这个变量可更改<Hx>元素的默认等级。默认值是 3

复杂的

Z_MAX_STACK_SIZE

这个变量可用于规定 Zope 中安全管理器所使用的堆栈的大小。（默认值为 100）

4.7.最后的办法

如果在安装过程中遇到问题，自己实在无法解决，还可以寻求别人的直接帮助，包 括使用 Zope 邮件列表和|使用#zope| IRC 频道。请参见“相关资源”一章。

第四章 面向对象技术

要充分使用 Zope 的强大功能，需要领会“面向对象”这一概念。“面向对象”是一种软 件开发模式，它被广泛的运用在许多程序语言中（C++, Java,Python, Eiffel, Modula-2,等等），以及许多模拟真实世界的计算机系统中。它要求根据对象来设 计应用程序。本章从 Zope 开发者的角度综合性的讲述面向对象的基础概念。

1. 对象（Objects）

在面向对象的系统中，比如 zope，应用程序是围绕对象来设计的。对象本身包含有 数据和逻辑。通过对比，就可以容易的看出它的特点。

在一个典型的非面向对象的应用程序中，需要考虑两件事情：

代码

例如在典型的基于 CGI 的 web 应用程序中的那些 Perl 语言编写的代码，些代码可以从数据库中搜索数据并给用户显示表格形式的数据。

数据

比如，存储在数据库 MySQL 或 Oracle 中的人员数据。这些数据供程序代码进行调用，没有了代码，这些数据几乎没有任何价值。

在一个典型的面向对象的应用程序中，只考虑一件事情就可以了：

对象

对象是代码和数据的结合体。例如，你有一个“雇员”对象，它表示一个雇员。它将包含这个雇员的数据，比如电话号码，姓名和地址，就像存储在 MySQL 或 Oracle 这样的数据库中的数据。然而，对象还包含逻辑（代码），可以控制和显示这些数据。

在一个非面向对象的应用程序中，数据和代码是分离的。但是在一个面向对象的应 用程序中，数据和代码存贮在一个或多个对象中，每个对象都代表了一个特定的 事物。对象可以表示各种事物。

在 zope 中，Control_Panel 是一种对象，文件夹是对象，根文件夹也是对象。当在 管理界面中使用“添加列表”（“add list”）来创建新的条目时，就是在创建对象。创 建 Zope 产品实际上就是定义新类型的对象，然后就可以加入到“添加列表”中从而 允许创建对象。产品作者可以定义“表单”对象或者“Weblog”对象。任何一个名词 都可以抽象成对象。

作为一种程序设计方法，面向对象技术让软件开发者根据“现实世界”来设计和创建 程序，比如：文件夹、控制面板、表单雇员等等，而不是围绕计算机概念，比 如：比特、流、整数等等这样的概念来设计程序。面向对象技术不是通过直接的使 用基础概念（比特和字节）来让计算机解决问题，而是通过更为人性化的提炼出的 概念来让计算机解决问题。面向对象的核心是让开发者尽可能在最大程度上创建基 于自然语言的真实世界系统，从而更轻松的理解和解决问题。

面向对象的思想就是对现实世界进行抽象的过程，它要求开发者尽量把大的问题分解成小而独立的子问题。从而根据这些子问题来定义解决方法。对象就是针对这些子问题而定义的解决方法。

2. 属性（Attributes）

对象的数据通过它的属性来定义。比如，雇员对象会有一个“电话号码”属性。这个 属性会包含一些表示这个雇员电话号码的字符。其它的属性还可能包括姓名、职位等等。 对象一般使用属性来存储数据，这些数据描述了这个对象。比如，通过“电话号码”、“姓名”、“职位”等等这些属性描述了一位雇员。属性可认为是一个微型的数 据库，它包含了能够表示现实世界的信息。对象的全部属性定义了对象的状态。当 一个或多个属性变了，就可以认为对象的状态发生了改变。 在 Zope 中，属性用 Properties 表示。

3. 方法（Methods）

对象的行为用“方法”来定义。对象的方法可以根据对象的属性执行某种行为或操 作。比如雇员对象的"getFirstName"方法可取得对象的"first_name"属性，而 "setFirstName"方法可以修改对象的"first_name"属性，方法"getTitle"可以取得 雇员的职位信息等等。

方法类似于一些语言（比如 C）中的函数.。方法和函数的关键不同之处在于方法是 和对象结合在一起的，方法不仅仅可以对传递过来的参数进行处理，它还可以对 所结合的对象的属性进行操作。

Zope 当中有一些对象也叫做“方法”，比如 DTML 方法（DTML Methods）,SQL 方法 （SQL Methods），以及外部方法（External Methods）。这是因为这些对象能够 起到“方法”的作用。默认情况下，在调用的时候，它们和包含其自身的文件夹对象 结合在一起，可称之为绑定。脚本（Script）对象就采用了绑定。

4. 消息（Messages）

在面向对象的系统中，一个对象需要和同一系统中的其它对象通讯才能够完成一项 工作。比如，仅仅有雇员对象是不够的，我们需要其它对象能够使用这个雇员对 象。我们可以创建名为“雇员汇总”的对象，它通过调用雇员对象，负责收集所有 雇员的姓名。

当对象和其它对象通讯的时候，就给其它对象发送消息。发送消息的方式是通过使 用对象的方法。比如“雇员汇总”对象可以通过调用雇员对象的"getFirstName"方法 来发送消息。雇员对象收到消息，并返回它的"first_name"属性。消息就是通过调用对象的方法来实现发送的。

当在 Web 浏览器中，通过 URL 来指向某个 Zope 对象的时候，实际上就是请求给 Zope 对 象发送消息，然后 Zope 对象调用 URL 中所提供的方法，把响应执行后的结果返回到 浏览器中。

5. 类和实例（Classes and Instances ）

类定义对象的行为，充当对象构造器的作用。比如，雇员对象实际上就意味着使用 雇员类构造的对象。或者是雇员类对象。大多数的对象都属于某种类。

在一个系统中，经常会发现相似的对象，这些对象只是属性的数值不同。比如， 多个雇员对象，每个都有"first_name" 和"last_name"属性，但属性数值确不相 同。这样就可以认为这些对象是同一个类的成员。

类就像是对象的建筑图。用同一张建筑图可以建造出很多房子，同样用同一个类可 以构造出很多个对象。这些对象都具有相同的行为，但属性确不尽相同。这就是 类和实例。类的实例可以是多个，并且可以通过相同的方法进行调用，实例的属性 可不同，但行为却是相同的。

通过同一个类构造的两个对象是相似的，它们都拥有相同的方法。方法一般不通 过对象本身进行定义，而是通过对象的类进行定义。比如，雇员类定义了 getFirstName 方法，所有这个类的成员都具有这个方法。类的所有方法定义了对 象的行为。

通过类构建的对象是类的实例。在 Zope 中，比如 Examples 文件夹是文件夹类的实 例。通过 Zope 管理界面所处理的对象都是某个类的实例。最典型的例子是 Zope 产 品，它们由各种类构成。

6. 继承（Inheritance）

有些时候，需要对象拥有相同的行为，但彼此是不同的。比如可能需要“正式雇员” 对象，这个对象除了拥有所有普通雇员对象的行为外，还要能够记录所得税缴纳 情况。

通过继承可以解决这个问题。继承就可以实现共享对象行为，从而具有不同于其 它对象的行为或扩展其它对象的行为。 继承是在类的层次定义。由于类定义行为，因此，要更改对象的行为，就需要更 改类。

现在假设在雇员类的基础上构造“正式雇员”对象。这个对象需要添加一个 ″getTaxIdNumber″方法和一个″tax_id_number″属性，可以说成是“正式雇员”类继承 自雇员类。按照面向对象的说法，就是“正式雇员”类是雇员类的子类，雇员类是“正式 雇员”类的超类。

当子类从其它类中继承行为时，不一定需要完全沿用超类的行为。子类可以覆盖超 类中定义的方法。

在 Zope 中，广泛使用继承。比如，“Image”类继承自“File”类，这是因为图片也是 一种文件，有很多共通之处。但“Image”类允许直接查看图片内容，而不像文件那 样直接下载。实现的方式是重新定义了“File”类中的 index_html 方法。

7. 对象存在期（Object Lifetimes）

对象实例有一定的存在期。存在期由管理员或系统进行控制。

比如，文件，文件夹，DTML 方法等等的存在期是从被用户创建时开始，直到被删 除时结束。这些对象被称作是持续对象，它们存储在对象数据库（ZODB）中。

不同的对象有不同的存在期。另外一些对象的存在期可以是短暂的，只维持一小段 时间。比如 Web 请求对象（即 REQUEST），它的存在期从服务器中接收到来自浏览 器的请求开始，直到响应发送回浏览器为止，并且这个对象会自动销毁。Zope 中的 “session data”对象的存在期更为明显，它的存在期从用户程序创建开始，到系 统自动终止这个对象时为为止。一般这个时间默认是对象不活跃后的 20 分钟。

8. 总结

Zope 是一种面向对象的开发环境，需要领会面向对象的一些基本概念，比如属性、方法、类和继承等等。如果要开发 Zope 产品，则需要充分理解面向对象的概念。

第五章 使用 Zope 管理界面

1. 介绍

当登录 Zope 以后，就进入到 Zope 管理界面（ZMI）。ZMI 是进行管理和开发的环境，可以控制 Zope，以及各种对象，或者用来开发 web 应用程序。 ZMI 按照对象的层次显示，几乎所有的链接或按钮都表示了对象的某种操作。用 Zope 构建 web 应用程序，主要都在创建和管理对象。 对象这个概念很简单，如果还没有理解也没关系。可以参考“面向对象技术”一 章。

2. Zope 管理界面如何组织对象

不同于 Apache 或 Microsoft IIS 这样的 Web 服务器的是，Zope 不是通过调用服务器硬 盘中的 HTML 文件方式来提供 web 服务。Zope 创建的对象不是象硬盘中的文件那样有 html 扩展名。Zope 中不使用操作系统中的文件层次。

Zope 是在 ZODB（Zope 对象数据库）中创建和存储对象。ZODB 通过使用"Data.fs"文件存储对象。Zope 管理界面是管理对象数据的主要方式。另外还可以通过多种方式来处理对象数据，至少还包括 FTP 和 WebDAV，这些内容在“使用其它工具管理 Zope 对象”一章中讲述。

2.1. ZMI 构成

ZMI 使用三个浏览器框架。左边的框架叫做导航框架，用于通过树形式折叠显示 Zope 对象层次结构。右边是工作框架，用于显示当前正在管理的对象的视图。上边的叫做状态框架，用于显示登录状态。

2.2. 导航框架

在这里可以看到 root 文件夹和所有的子文件夹。所有 zope 对象都存放在 root 文件夹中。

导航框架

用“＋”号显示的文件夹表示可以打开查看子文件夹。 点击导航框架中的对象图标或名称，右边的工作框架中会出现相应的对象视图。

2.3. 工作框架

工作框架中显示正在管理的对象。工作框架用于管理各种对象。

工作框架

横穿屏幕顶部的是几个标签。当前选定的标签以发亮的颜色突出显示。每一个标签带你进入当前对象的不同视图。每个视图让你执行一个有关这个对象的不同管理功能。

当你第一次登录到 Zope 时，当前的对象是根文件夹。 标签下面是当前对象的类型和 URL 的描述。左边的图标表示当前对象类型。右边是对象的 URL。“Folder at /”告诉你当前对象是一个文件夹并且它的路径是 / 。注意这个对象的 URL 是相对于 Zope 根 URL 的。因此如果 URL 或你的 Zope 站点是 http://mysite.example.com:8080，那么"Folder at /myFolder"这样的 URL 就会是 http://mysite.example.com:8080/myFolder。 工作框架中顶部所显示的链接可用于在不同对象之间进行跳转。

2.4. 状态框架

在顶部的状态框架中显示登录用户名和一个可选的下拉框。解释如下：

Set Preferences：通过这个选项可以设定 Zope 管理界面默认的参数。

Logout ：退出管理界面。

Zope Quick Start ：显示“Quick Start”页面。

状态框架

3. 创建对象

通过管理界面可以创建新对象。方法是从标有"Select type to add..."的下拉菜单中选择要创建的对象。这个下拉菜单被称作“添加列表”。 文件夹是经常用到的对象，因此可以先从添加文件夹对象开始。从添加列表中选择“Folder”，然后在出现：

添加文件夹

在 Id 字段中键入 zoo, 在 Title 字段中键入 Zope Zoo，然后单击 Add 按钮。 Zope 将在当前的文件夹里创建一个新文件夹。在根文件夹里有了一个名为 zoo 的新文件夹。 使用 Zope 创建对象的基本步骤是：

进入你想添加一个新对象的文件夹。

从下拉菜单中选择你要添加的对象类型。

填写一个添加表单并提交。

Zope 将在当前的文件夹里创建一个新对象。

注意，在添加表单中填写的 id 会用在 URL 中。比如根文件夹中有一个名为“zoo”的文件夹，那么它的 URL 为：http://your.server.name/zoo。

4. 移动和重命名对象

大多数的计算机系统可以让你用剪切、复制和粘贴的方式在不同的目录间移动文件。Zope 有一个类似的系统，先剪切或复制对象，然后粘贴到一个新地址，通过这样一种方式 Zope 就可以让你在文件夹中移动对象。

注意：要进行移动和重命名操作，需要浏览器支持 cookies 功能。大多数浏览器默认情况下都支持。

为了验证复制和粘贴，请在根文件夹中创建一个 id 为 bears 的新文件夹。然后通过选中紧邻文件夹左边的复选框选择 bears。然后单击 Cut 按钮。剪切操作把所选的对象从文件夹中移出，并且存放到剪贴板中。对象从它的地址中消失了，直到被粘贴到其它某个地方为止。

现在通过单击它进入 zoo 文件夹，然后再通过单击进入 arctic 文件夹。你还可以通过导航栏进入到这个文件夹。现在，单击 Paste 按钮，把被剪切对象粘贴到当前文件夹。你会看到 bears 文件夹在新地址中出现了。你在根文件夹中再也找不到 bears 文件夹了，这样你就可以确信文件夹确实被移动了。 复制过程类似于剪切过程。当你粘贴被复制对象时，原始对象不变。选择你想复制的对象，然后单击 Copy 按钮。再转到另外一个文件夹，单击 Paste 按钮。

你可以剪切和复制包含有其它对象的文件夹，并且只用剪切和粘贴就可以一次移动许多对象。例如，进入 root 文件夹，复制 zoo 文件夹，再粘贴到 root 文件夹。你会在 root 文件夹中得到两个文件夹，zoo 和 copy_of_zoo。如果你把对象粘贴到同一文件夹里，即你复制它时所处的文件夹。Zope 会更改被粘贴对象的 id。这是必需的一步，这是因为你不能在同一个文件夹中拥有两个相同 id 的对象。

要想重命名 copy_of_zoo 文件夹，通过选中文件夹左边的复选框来选择这个文件夹。然后单击 Rename 按钮。这样就会出现 Rename Items 表单，如下图所示。

重命名对象

键入一个新 id：zoo2，单击 OK。id 可以由字母、数字、空格、破折号、下划线和点组成，并且它们是大小写敏感的。这是一些合法的 id：index.html、42 和 Snake-Pit。

现在，你的 root 文件夹包含了一个 zoo 文件夹和一个 zoo2 文件夹。每个文件夹中都包含一个 bears 文件夹。这是因为当我们复制 arctic 文件夹时，同时复制了其中所包含的 bears 文件夹。

如果你想删除一个对象，选中它，然后单击 Delete 按钮。不同于剪切对象，被删除的对象不存放在剪贴板中，并且不能粘贴在下一节，我们将看到我们如何通过 undo 找回已经删除的对象。

Zope 不允许你剪切、删除或重命名一些根文件夹中的特殊对象。这些对象包括 Control_Panel, browser_id_manager, 和 temp_folder.这些重要的对象是 Zope 操作所必需的。可以删除其它 root 对象，比如 index_html, session_data_manager, standard_html_header, standard_html_footer, standard_error_message, 和 standard_template.pt。但是不建议这样做。

5. 事务处理和撤销错误

Zope 中创建的对象存储在对象数据库中，因此 Zope 支持事务处理（ transactional）。 事务处理就是对一批对象进行的操作以批处理的方式执行，只要其中一个操作失败，全部操作都无效。在 Zope 中，一个 Web 请求开始一个事务处理过程，当 web 请求完成，Zope 执行事务处理，除非在处理 web 请求的过程中发生了错误。如果发生了错误，Zope 终止处理事务，所做的更改都撤销。处理 web 请求过程中发生的所有操作都属于这个事务。

事务处理过程中的任何操作都可以撤销，方法是使用“Undo”标签。通过撤销含有错误的事务就可以从错误中恢复过来。

选择刚刚创建的 zoo 文件夹，点击 Delete，删除。文件夹消失了。通过撤销删除操作可以还原这个文件夹。 点击 Undo 标签如下图：

Undo 视图

选择第一个名为/manage_delObjects 的事务，点击 Undo 按钮，就可以撤销最后一个事务。可以撤销一个撤销（换句话说就是重做）。

6. 撤销细节和注意事项

你不能撤销一个后来的事务处理所依赖的事务处理。例如，假设你把一个对象粘贴到一个文件夹，然后删除相同文件夹中的一个对象，你也许会疑惑能否撤销前边的那个粘贴操作。两个事务处理发生在同一个文件夹，因此你不能简单地撤销前一个事务处理。解决的方法是两个事务处理都要撤销。你可以一次撤销多个事务处理，方法是在 Undo 标签中选择多个事务处理，然后单击 Undo。

只有对存储在Zope里的对象所发生的更改才能够被撤销。如果你使用了关系数据库服务器中的数据，就像Oracle 或者MySQL（参见“关系数据库连通”）等，不能撤销对被存储数据所做的更改。

7. 查看修改历史

通过查看修改历史可以看到所做的修改。这个功能对于DTML Methods, DTML Documents, Zope Page Templates, 和 Script (Python)这样的对象更为有用。方法是查看对象的History视图。

历史视图

对象的History视图中可以对不同版本进行比较，跟踪变化。这个功能可以通过选择对象历史中的两个版本，然后进行相互比较。方法是先点击选中两个复选框，然后点击Compare按钮。 进行比较后的结果文件的格式被称为diff格式。diff格式中使用“＋”号表示新增加的内容，使用“－”号表示去掉的内容。

版本比较

可以通过复选框选中某个版本，然后点击“Copy to present”按钮。

8. 导入和导出对象

使用导出和导入，你可以把对象从一个 Zope 系统中转移到另外一个系统中。你能够把所有类型的 Zope 对象导出成一个文件。然后，这个文件就可以被任何其它 Zope 系统导入。 你可以把导出一个对象的过程看作是把一部分 Zope 系统克隆成文件，这样就可以在不同的计算机之间转移数据。

假设你有一个家庭作业文件夹，你想从学校 Zope 服务器中导出它，然后带回到家里的 Zope 服务器中继续完成作业。你可以在根文件夹里创建一个名为 homeWork 的文件夹。进入包含你的 homeWork 文件夹的那个文件夹。通过选择旁边的那个复选框选中 homeWork 文件夹。然后单击 Import/Export 按钮。你现在应该处于 Import/Export 文件夹视图中了，如下图所示。

导出 homeWork.zexp

这个屏幕中分成了两个部分。上半部是导出部分，下半部是导入部分。要想从这个屏幕导出对象，在 Export object id 字段中键入对象的 id。在我们的例子中，在上一屏中已经选择了 homeWork 文件夹，因此 Zope 已经为我们填写好了这个字段。

接下来的选项让你选择把导出文件下载到计算机还是把导出文件保存在服务器里，如果你选择 Download to local machine（下载到本机），单击 Export 按钮，你的 Web 浏览器会提示你下载导出文件。如果你选择 Save to file on server（在服务器上保存文件），Zope 就会在运行它的那台服务器上保存文件，你只好亲自从文件所在的那个地址中取出文件。导出文件被写入到服务器上的 Zope 的 var 目录里。默认导出文件的扩展名是.zexp。

通常，把导出文件下载到本机更为方便。有些时候则是把导出文件存储在服务器上更为便利——例如，如果你处于一个低速的连接中并且导出的文件非常大，或者你仅仅想把被导出的文件转移到相同计算机中的另外一个 Zope 例程中。

最后一个导出表单中的元素是 XML 格式复选框。选中这个复选框就会把对象按照扩展标记语言（XML）格式导出。如果没有选中，就会按照 Zope 的二进制格式导出。XML 格式的下载文件相对来说大得多，但是易于阅读理解并且可以解析。目前，唯一的可以理解这种 XML 格式的工具是 Zope，但是未来将会出现其它可以理解 Zope 的 XML 格式的工具。通常，你不用选它，除非你对 XML 格式内容非常好奇，想亲手检查它。

为了能够在家里能够看到内容，选择"download to local machine"，然后单击 Export 按钮，然后保存你导出的 homeWork.zexp 文件。

现在，假设你回到家，想把文件导入到家里的 Zope 服务器。首先，你必需把这个导出的文件复制到 Zope 安装目录中的 import 目录中。在安装 Zope 的计算机中可以找到这个目录。这个目录在不同的计算机中会不一样，因此如果你不能确认，那么就检查 Zope 的控制面板或者让你的系统管理员告诉你把导入文件放在什么地方。现在，进入你要执行导入任务的所在文件夹的 Import/Export 视图。在 Import file name 中输入以前导出的那个文件名，然后单击 Import，就把那些对象导入到 Zope 里了。在 Linux 系统中，还可以通过以下命令复制到远程计算机中:

```
chrism@saints:/tmp$ ls -al homeWork.zexp

-rw-r--r--    1 chrism   chrism        182 Jul 13 15:44 homeWork.zexp

chrism@saints:/tmp$ scp homeWork.zexp saints.homeunix.com:/home/chrism/sandboxes/ZBExample/import

chrism@saints.homeunix.com's password:

homeWork.zexp         100% |*****************************|   182       00:00

chrism@saints:/tmp$
```

在这个例子中，导出的文件从本机中的/tmp 目录复制到了远程计算机的 /home/chrism/sandboxes/ZBExample/import/homeWork.zexp 中。

现在，进入管理界面。创建一个名为 import_example 的文件夹，进入这个文件夹，然后点击 Import/Export 按钮，然后滚动屏幕到导入部分，Zope 给你提供了两个选项，Take ownership of imported object（获得导入对象的所有权）和 Retain existing ownership information（保持现有所有权信息）。在第 6 章“用户和安全”中，详细论述了所有权。现在，只需保持 Take Ownership 按钮的被选中状态就可以了。在 Import file name 中输入以前导出的那个文件名，然后单击 Import，就把那些对象导入到 Zope 里了。

导入 homeWork.zexp

导入完成后，就有了一个名为 homeWork 的新对象。

成功导入 homeWork.zexp

在导出和导入过程中，需要注意的是，在两个 Zope 中，需要安装有相同的产品。如果导入失败，说明有可能没有安装所需的产品。另外，导入的时候，对象的 ID 不能和已有的 ID 相同。

9. 使用对象属性

属性是对象结合信息的一种方式。许多 Zope 对象都支持属性。属性可用于识别内容的类型。比如，许多 Zope 内容对象都有内容类型属性。属性的另外一种用法是提供对象的元数据，比如作者，标题，状态等等。

属性可以是字符串，但要比字符串复杂。属性还可以是数字，列表或其它的数据结构。所有的属性通过 Properties 视图进行管理。比如，点击 Root 对象的 Properties 标签，就能够看到属性管理视图，如下图所示：

属性管理视图

属性可以由名称（name）、数值（value）和类型（type）组成。属性类型定义了数值的种类。

在上图中，可以看到文件夹只有一个字符串类型的属性：title，数值是：Zope。在这里可以进行更改，然后保存。在这里还可以添加新的属性，输入名称和数值，再选择完类型，点击“Add”按钮就可以了。

Zope 支持多种属性类型。每种类型适合完成不同的任务。以下，简单描述了每种属性类型的作用。

string

字符型，就是一个字符序列。字符串类型是最常用的类型。

int

整型，就是一个整数，可以是正数或负数，但不能是分数，长度可达到 32 比特。

long

长整型??类似于整形，但没有范围限制

float

浮点型，可以是小数，对于货币经常使用这个类型。

lines

多行字符型，由字符串的序列组成。

tokens

表征型，就是由多个用空格分开的单词组成的列表。

text

文本型，类似于字符型，但长度不限，并且可以有行结束符。

selection

选择型，用于实现 HTML 中的选择框。

multiple selection

多选型，用于实现 HTML 中的多选框。

通过属性可以定义对象的数据，因此是非常有用的。大多数的 Zope 对象都有属性，从而可以通过“脚本（scripts）”和“方法（methods）”这样的对象进行数据操控。

10. 使用帮助系统

Zope 内置有帮助系统。在屏幕的右上角有一个"help"按钮。这个按钮可以用来显示帮助内容。 比如，进入 root 文件夹，点击帮助按钮，就会出现下图所示内容：

帮助系统

新弹出来的窗口左边用于导航，右边用于显示内容。帮助内容随同工作框架中对象的变化而变化。

11. 浏览和搜索帮助

在帮助系统中，不仅可以看到当前对象的帮助内容，还可以浏览查看所有的帮助内容。通过点击“＋”号或“－”号就可以显示相应的目录内容。新安装的产品也可以提供帮助内容。通过点击左边的“Search”标签，然后输入关键字可以搜索内容。

12. 退出

通过选择状态框架中下拉菜单中的退出选项就可以退出系统，也可以直接关闭浏览器。

第六章 使用基本 Zope 对象

当通过 Zope 构建 web 应用时，主要都时通过调用各种对象来完成。本章讲述最基础的 Zope 对象。

1. 基本 Zope 对象

当用 Zope 构建一个 Web 应用程序时，基于各种对象构建应用程序。经过设计，不同的对象处理应用程序中的不同部分。一些对象保存内容数据，例如字处理文档、电子报表和图象这样的数据。一些对象控制动态 Web 内容的生成方式，例如如何接收处理发自 Web 表单输入信息或执行一段脚本程序。一些对象控制内容显示或表现给浏览者的方式，比如 web 页面或电子邮件。通常，Zope 对象充当以下三种类型的角色：

内容

比如象文档、图像和保存不同类型的文本和二进制数据的文件等等这样的 Zope 对象。除了 Zope 里的对象包含内容外，Zope 还可以处理外部存储的内容，例如，在一个关系数据库里的内容。

表现

你可以使用能够起到页面“模板”作用的对象来控制站点的外观和感觉。Zope 当中带有两个可以管理表现的工具：DTML(还可以处理逻辑)和 Zope 页面模板(ZPT)。DTML 可以同时处理表现和逻辑，而 ZPT 不行。

逻辑

Zope 在用脚本语言处理事物逻辑方面具有多种灵活性，Zope 允许你用 DTML、Python、Perl(需要外加)编写行为脚本。事物逻辑就是任何不涉及内容表现的程序，它更适合执行象更改对象、发送消息、测试条件和响应事件等等这样的任务。

实际上，这些对象很难严格区分成这三种。比如，对于 DTML，这三个方面的作用都能够有效，但主要是用在表现方面。还有一些不属于这三类的对象，将在“Zope 服务”一章中讲述。另外，你还可以通过安装别人开发好的对象来扩展 Zope 的能力。这些第三方提供的对象称之为“产品”。在 Zope.org 网站中可以找到。

2. 内容对象：文件夹、文件和图像

2.1. 文件夹

文件夹的主要作用是容纳其它的对象。文件夹可以容纳各种类型的对象，其中包括其它的文件夹。因此，你可以嵌套文件夹这样就形成文件夹树。这种文件夹套文件夹的排列给你的 Zope 站点提供了结构。好的结构非常重要，在 Zope 里几乎所有的方面（从安全、行为到表现）都受你的站点文件夹结构的影响。对于那些在计算机中通过文件管理程序使用过文件和文件夹的人们来说，文件夹结构应该很熟悉。例如微软的 Windows 资源管理器或者任何一种流行的 Unix 文件管理器，就像 xfm、kfm、或者 Gnome 文件管理

2.2. 文件

Zope 文件包含原始数据，就像你的计算机中的文件所起到的作用那样。大量的信息，例如软件、声音、视频和文档等，在 Internet 和世界中以文件的方式传输。你能使用文件来保存任何种类的信息 ，例如 Flash 文件、applets、压缩包等等这些 Zope 非特定支持的文件。

文件不考虑它们的内容是否属于某种特定格式、文本类型或者其它类型。文件擅长保存各种二进制内容，这些内容就是属于某种类型的原始计算机信息。

每个文件对象都有一个特殊的内容类型，它是一个标准 Internet MIME 文件类型名称。比如，"text/plain" (纯文本), "text/html" (html 文本), 和 "application/pdf" (Adobe Portable Document 格式)。当你上载文件时，Zope 试图从文件名称中猜测出文件类型。

2.2.1. 创建和编辑文件

要创建文件对象，可以从添加列表中选择 File, 先点击“Browse”按钮从本机中找到要添加的文件，比如 Word 文件或 PDF 文件等等，如下图所示：

添加 PDF 文件对象

Zope 试图通过文件名称来决定对象的 id 和 title，因此不一定需要输入 id 和 title。然后点击“Add”按钮。这样就添加好了如下图所示：

编辑 PDF 文件对象

如果添加的是 Word 文档，类型就是 application/msword。如果为 PDF 文件，则为 application/pdf。默认的类型为 application/octet-stream，也可以手工来修改类型，输入新的类型，然后保存即可。

你还可以给文件设定预处理文件（precondition ）,这个预处理文件可填写一个可执行的 Zope 对象的名称（比如 DTML 方法，脚本或外部方法等等），这个预处理文件会在当前这个文件被调用或下载之前运行。如果预处理文件运行过程中发生了错误，那么当前这个文件就无法调用。这个功能不太常用。 通过“File Data”部分，可以上载一个新文件，替换掉原有的数据。

2.2.2. 编辑文件内容

如果文件的内容只是文本，并且小于 64K，Zope 中就可以在“Edit”视图中通过文本域来编辑内容。文本文件的内容类型采用 text/ 开头，比如 text/html 或 text/plain。

2.2.3. 查看文件

在管理界面里，可以通过点击工作框架中的“View”标签来查看文件。

查看 PDF 文件对象

另外，还可以通过直接访问 URL 的方式来查看文件，比如： http://localhost:8080/Reader.pdf

2.3. 图像

图像对象包含图像文件的数据，比如 GIF, JPEG, 和 PNG 文件。

图像对象和文件对象的管理界面一样。前边所讲的与文件相关的内容同样适用于图像对象。图像多了一个预览功能。

3. 表现对象：Zope 页面模板和 DTML 对象

Zope 鼓励你通过使用不同的对象来使表现和逻辑分离。用于处理表现的对象可用于帮助确定如何显示 Web 页面以及其它可见的数据。表现对象一般用来显示 HTML(也可以是 XML 或 WML)。

Zope 有两个用于处理表现的工具：Zope 页面模板(ZPT)和文档模板标记语言(DTML)。

通过 Zope 页面模板可以动态的显示 web 页面内容。模板中的 HTML 中插入了特定的 XML 名称空间元素，这样就可以确定显示页面的方式。

文档模板标记语言（DTML）对象同样也可以显示 Web 页面内容。DTML 通过在 HTML 中插入特定的标记符号来动态控制显示页面的方式。

ZPT 和 DTML 都是服务器端运行的脚本语言，就像 SSI,PHP,embperl 或 jsp。也就是说 Zope 在服务器端运行 DTML 和 ZPT，然后把运行的结果返回到用户的浏览器。而客户端脚本语言，比如 Javascript，不是在服务器端运行，而是通过用户的浏览器来运行。

3.1. ZPT 和 DTML 的不同: 目的相同，但适用的人群不同

许多用于动态生成 HTML 内容的语言都有一个问题，那就是不能很好的把表现和逻辑分离开。比如，基于标记符号的脚本语言比如 DTML,SSI,PHP,和 JSP,都是在 HTML 中嵌入特定的标记符号，这样的话，就让那些只关注界面效果的设计者很难进行编辑，设计工具软件很难正确处理这些标记符号。 使用这些技术的时候，HTML 设计者只能先用 Macromedia Dreamweaver 或 Adobe GoLive 这样的工具设计出页面，然后交给程序开发人员。开发人员再对这些页面进行处理，用特定的符号进行处理。一旦内容发生了变化，设计人员和开发人员都需要对文件进行修改，这样就不利于高效率完成工作。

Zope 最开始采用的动态语言 DTML，但不久就发现 DTML 能够让开发人员快速生成动态的 web 页面，但是却不能让那些不懂技术的图形设计者高效率的进行处理。为了解决这个问题，就开发出了 ZPT。ZPT 是一种基于属性的表现语言，它努力适用于开发人员和非技术型的设计人员。

ZPT 和 DTML 有共通的地方，可是如何选择呢？以下是一些参考原则：

如果开发团队中有程序开发人员和非技术型设计人员，则应选择 ZPT。设计工具软件，比如 Macromedia Dreamweaver 和浏览器都可以正确显示 ZPT 文件。

当需要生成除了 XML,HTML,XHTML 等等这样的文本时，则应使用 DTML。ZPT 要求必须生成 XHTML 或 XML 兼容的页面。ZPT 不能动态处理 CSS，SQL 语句或其它非 XML 形式的文本数据。而 DTML 却可以。

对于程序开发人员来说，DTML 显得容易些。这是因为 DTML 中进行条件判断比较容易。DTML 更像是 PHP 或 ASP 这样的脚本语言。

DTML 的逻辑性比较强，它不像 ZPT 那样强调表现和逻辑分离。如果你希望表现和逻辑能够分离开，那么则使用 ZPT 。

3.2. Zope 页面模板

Zope 页面模板用于动态创建 HTML 页面。

3.2.1. 创建页面模板

创建页面模板的方法是从添加列表中选择 Page Template,然后输入 id 和 title，点击“Add”按钮。

3.2.2. 编辑页面模板

编辑页面模板的方法是点击进入页面模板的“Edit”视图。比如：

默认的页面模板内容

你可以用下面的HTML代码替换内容。

```
<html>
  <body>
    <h1>This is my first page template!</h1>
  </body>
</html>
```

然后点击“Save Changes”按钮。

3.2.3. 上载页面模板

如果有编写好的页面模板文件，可以上载到Zope中。方法是先从添加列表中选择页面模板，然后在出现的表单中，点击“Browse”按钮，选择本机中已经编写好的页面模板文件。然后点击“Add and Edit”按钮即可。

3.2.4. 观看页面模板

观看页面模板的方法是在管理界面中点击“Test”标签。也可以通过直接输入URL的方式观看结果。

观看页面模板

3.3. DTML 对象:DTML 文档和 DTML 方法

DTML 是另外一种用于处理表现的工具。它分为两种：DTML 文档和 DTML 方法。DTML 遵循 Zope 的安全策略，因此可以使用部分 Python 模块，但不能直接读取系统中的文件系统。这样的话就可以就可以让站点管理员方便的把创建 DTML 的权利授权给其他人。关于安全，请参见“用户和安全”一章。

DTML 方法用于处理动态的内容，可以被其它 DTML 和其它对象调用。比如，用于显示页面中的导航条的 DTML 方法，以及用于显示页面中的页眉的 DTML 方法。而 DTML 文档则用于存储文档类型的数据，它本身就可以直接显示。DTML 文档可以添加新的属性而 DTML 方法不行。一般来说，用 DTML 方法来处理动态内容就可以了。

3.3.1. 创建 DTML 方法

举例来说，创建一个“Sales”文件夹，然后从添加列表中选择“DTML Method”。然后输入 id： SalesStaff ，和 title：The Jungle Sales Staff，然后点击 Add 按钮。在工作区中就会增加一个 DTML 方法。

3.3.2. 编辑 DTML 方法

在 Edit 视图中可以编辑 DTML 方法，如下图所示：

编辑一个DTML方法

在这个视图中点击“Save Changes”按钮可以保存所做的修改。其中的文本框可以通过Taller、 Shorter、 Wider、和Narrower按钮调整大小。通过Upload File按钮可以上载文件。

举例来说，让我们先把默认的内容删除，然后在文本框中加入以下代码:

```
<html>
<body>
<h2>Jungle Sales Staff</h2>
<ul>
  <li>Tarzan</li>
  <li>Cheetah</li>
  <li>Jane</li>
</ul>
</body>
</html>
```

注意，上边的代码不包含任何动态的特性，它只使用了一些HTML 代码。在以后的章节中将详细讲述DTML。

编辑完成以后，点击Change 按钮就可以了。

3.3.3. 观看 DTML 方法

通过点击View 标签，可以观看DTML 方法执行以后的显示结果。比如，点击上边的那个 SalesStaff 的View 标签，就会显示如下结果：

观看 DTML 方法

另外，还可以通过直接输入URL 方式直接查看DTML 方法。

3.3.4. 上载文件

假设，你不想通过Web 浏览器来编辑HTML，或者你已经有了一个编写好的HTML 文件，那么你可以直接上载这些文件，并可以转化成DTML 方法。

比如，有一个test.html 文件，其中的内容是:

```
<html>
  <body>
    <h1>This is my first uploaded DTML Document!</h1>
  </body>
```

</html>

然后，进入 Sales 文件夹，添加一个 DTML 方法，点击 Browse 按钮。在新弹出的对话框中选择你的 test.html 文件。然后再输入 id: test，最后点击“Add and Edit”按钮。然后就会生成一个新的 DTML 方法。

4. 逻辑对象：Script(Python)对象和 External(外部)方法

逻辑对象通常用于完成那些表现对象所用到的数据处理。通常，逻辑对象在调用的时候，不返回用于表现的 HTML 或文本，而是返回能够被表现对象调用的数值。比如，一个逻辑对象可以返回由字符串组成的列表，然后，表现对象调用这个结果，通过 HTML 把这个列表显示成表格。表现对象可以确定如何显示逻辑对象生成的结果。这种方式的好处是，在调整界面的时候，不必重新编写逻辑部分。 逻辑对象也可以直接通过 URL 进行调用。一般来说，只要逻辑对象返回数据，在浏览器中就可以看到结果。

Zope 中自带有两种逻辑对象：Script(Python)对象和 External(外部)方法。通过安装扩展产品，还可以支持其它逻辑对象，比如 Perl 脚本、PHParser、 PHPObject 和 ZopeJSP 等等。 Script 对象和 Exernal 方法对象都是采用 Python 语言进行编写。Python 是一种通用型的计算机编程语言，建议你能够掌握基本的使用方法。

特别需要强调的是，Python 通过缩排的方式组织代码，而不是像其它语言那样采用成对的大括号来组织代码。如果发现你的代码不能运行，应该先检查缩排方式是否正确。

4.1. Script (Python)对象

Script 对象就是一段受到安全限制的可通过 Web 编辑的 Python 代码。Script 对象不能运行所有的 Python 代码。它受到 Zope 安全策略的限制，也就是说，出于系统安全的考虑，Script 对象不能执行某些 Python 模块，以及直接读取系统中的文件。这样就可以安全的把创建 Script 对象的权利授权给其他用户。

4.1.1. 创建一个 Script

举例来说，进入前边创建的 Sales 文件夹，从添加列表中选择 Script，输入 id： SalesScript ，然后点击 Add 按钮。

4.1.2. 编辑一个 Script

编辑 Script 的最简单方式是直接通过 Zope 管理界面进行编辑。就是进入 Script 的 Edit 视图，比如：

默认的Script内容

在"Parameter List"（参数列表）中输入：name="Chris"

然后在文本框中输入以下代码:

```
return 'Hello, %s from the %s script' % (name, script.id)
```

然后点击Save Changes按钮。

4.1.3. 测试 Script

通过点击管理界面中的Test标签，可以测试Script。进行测试之前，需要输入参数。比如对于这个 SalesScript 对象，点击Test标签之后，如下图所示：

测试 Script

在 name 参数中输入你的姓名，然后点击“Run Script”，然后就会显示:

```
Hello, [yourname] from the SalesScript script
```

如果没有输入参数，将使用默认的参数值。另外，还可以直接通过 URL 直接调用。比如，在浏览器中直接输入 SalesScript 的 URL，就会显示:

```
Hello, Chris from the SalesScript script
```

在 URL 中，还可以直接提供参数值，比如 URL 为： http://localhost:8080/SalesScript?name=Fred，就会显示:

```
Hello, Fred from the SalesScript script
```

通过这个例子可以看到，Zope 可以自动通过 URL 来传递参数。

4.1.4. 上载 Script

对于 Script 对象，Zope 同样支持上载。需要注意的是，文件中的前边通过“##”符号来表示需要提供的参数。比如对于上边的 SalesScript，它的源代码是:

```
## Script (Python) "SalesScript"

##bind container=container

##bind context=context
```

```
##bind namespace=

##bind script=script

##bind subpath=traverse_subpath

##parameters=name="Chris"

##title=

##

return 'Hello, %s from the %s script' % (name, script.id)
```

这段代码可以通过点击“view or download”链接来查看。这个链接位于大文本框的下边。

4.2. External 方法

External 方法是另外一种逻辑对象。它与 Script 对象类似，都是采用 Python 语言进行编写，它不同于 Script 之处在于：

它不通过 Zope 管理界面进行编辑，而是通过把编写好的 Python 文件放在 Zope 安装目录中的 Extensions 目录中来实现。

由于它不通过 Zope 管理界面进行编辑，因此它没有安全限制，也就是说，它可以运行所有 Python 代码，并可以直接读取文件系统。

它不支持“绑定”。（在以后的章节中将讲述绑定）。

External 方法适用于需要更多权限的情况下，当需要不受 Zope 安全策略的限制时，可以使用 External 方法。比如希望在文件系统中写入数据。

4.2.1. 创建和编辑 External 方法文件

举例来说，在 Zope 安装目录中，进入 Extensions 目录，然后编写一个名为 SalesEM.py 的文件，内容为：

也可以通过直接访问 URL 的方式来测试结果。 另外，还可以通过 URL 传递参数。比如：http://localhost:8080/Sales/SalesEM?name=Fred，结果如下：

```
Hello, Fred from the Sales external method
```

上边的例子中输出显示的 id 为文件夹的 id，而不是自身的 id，因此可以看出 External 方法与其调用的环境密切相关。External 方法或 Script 会自动判断出其调用的环境，从而显示这个环境的信息。

4.3. SQL 方法：另外一种逻辑对象

SQL 方法是用来存储和执行数据库查询的逻辑对象。SQL 方法将在“关系数据库连通”中详细讲述。

5. 使用页面模板和 Scripts 创建基本的 Zope 应用

以下是一个使用 Zope 逻辑和内容对象创建的简单实例。这个例子构造了一个在线 Web 表单，用于帮助你的用户计算他们债务的复利总数。计算过程涉及以下过程:

你需要以下信息：

你当前的帐面余额（或负债）－－称为 principal（本金）

以小数形式显示的年利率（例如 0.095）－－称为 interest_rate（利率）

年利率期间所包含的次数（通常按月）－－称为 periods（周期）

从现在起你想计算的年份数－－称为 years.(年限)

interest_rate 除以 periods（通常为 12），我们将把结果称为 i

周期和年份相乘，我们将把结果称为 n

求（1＋i）的 n 次方

本金乘以这个结果，这就是新的余额（或负债）

下面将用页面模板和 Script 对象完成这个任务。

对于这个例子，需要两个页面模板，id 分别为 interestRateForm 和 interestRateDisplay。分别用于收集和显示信息。另外还需要一个 Script，id 为 calculateCompoundingInterest，用于完成计算过程。

第一步是要创建一个文件夹，可以叫做“Interest”，即 id，将要创建的对象放在这个文件夹中。

5.1. 创建数据收集表单

在 Zope 管理界面中进入 Interest 文件夹。然后添加一个 id 为 interestRateForm 的页面模板，用于收集来自用户的"principal", "interest_rate", "periods" 和 "years"信息。代码如下:

```
<html>
  <body>
  <form action="interestRateDisplay" method="POST">
```

```
<p>Please enter the following information:</p>

Your current balance (or debt): <input name="principal:float"><br>

Your annual interest rate: <input name="interest_rate:float"><br>

Number of periods in a year: <input name="periods:int"><br>

Number of years: <input name="years:int"><br>

<input type="submit" value=" Calculate "><br>

</form>

</body>

</html>
```

这个表单负责收集信息，然后调用 interestRateDisplay 模板。

5.2. 创建计算利率的 Script

接下来，在 Interest 文件夹中创建一个 Script 对象，id 为 calculateCompoundingInterest ，它接受四个参数：principal, interest_rate, periods 和 years。代码部分为:

```
"""

Calculate compounding interest.

"""

i = interest_rate / periods

n = periods * years

return ((1 + i) ** n) * principal
```

记住：在参数列表框中输入参数名称的时候，用逗号分开。另外，在编写代码的时候，注意代码的缩排。上边的代码都靠左编写。

5.3. 创建用于显示结果的页面模板

接下来，在 Interest 文件夹中创建一个用于显示结果的页面模板，id 为 interestRateDisplay。代码如下:

```
<html>
  <body>
  Your total balance (or debt) including compounded interest over
  <span tal:define="years request/years;
                    principal request/principal;
                    interest_rate request/interest_rate;
                    periods request/periods">
    <span tal:content="years">2</span> years is:<br><br>
    <b>$
    <span tal:content="python: here.calculateCompoundingInterest(principal,
                                                                 interest_rate,
                                                                 periods,
                                                                 years)" >1.00</span>
    </b>
  </span>
  </body>
</html>
```

5.4. 处理错误

如果直接调用 interestRateDisplay，由于没有传递过来的参数，因此程序会报错。比如直接点击 Test 标签，会出现错误：

```
Zope Error
Zope has encountered an error while publishing this resource.
Error Type: KeyError
```

Error Value: years

这个错误消息显示没有找到变量 years。错误的详细内容可以查看根目录中的 error_log 对象，其中记录下了详细的错误信息。包括时间，用户，错误类型等等信息。

5.5. 使用这个应用

接下来，就可以使用这个应用了。进入 interestRateForm 的管理界面，然后点击 Text 标签。 输入一些参数，然后提交信息就会把信息发送给 interestRateDisplay，然后又调用 calculateCompoundingInterest 程序进行计算，然后显示计算的结果。运行结果如下图所示：

计算结果

6. 总结

Zope 自带有一个实例讲解程序，通过跟随这个程序进行学习，可以了解到 Zope 的一些基础概念。 要使用这个程序，可以从添加列表中选择“Zope Tutorial”对象，然后输入 id 即可。也可以直接进入帮助系统，来查看这个程序。

第七章 获取机制

获取机制(Acquisition)就是一种在一定范围内能够在 Zope 对象之间共享动态行为的技术。获取机制在 Zope 中大量使用，是 Zope 中一个很重要的概念。

1. 获取机制和继承

在“面向对象技术”一章中讲过了“继承”，使用继承，可以让对象继承某些类的特定行为，以及覆盖或添加自己特有的行为。类的行为通过方法来定义，属性也可以同时继承。

在面向对象语言中，规定了自类如何继承父类的行为。例如，在Python这种支持多继承的语言中，一个类可以有多个父类。在下面的例子中，不一定需要熟悉Python语言，但需要知道class语句用于定义类，def语句用于定义类方法。class语句后边括号中是父类的名称。

```
class SuperA:

    def amethod(self):

        print "I am the 'amethod' method of the SuperA class"

    def anothermethod(self):

        print "I am the 'anothermethod' method of the SuperA class"

class SuperB:

    def amethod(self):

        print "I am the 'amethod' method of the SuperB class"

    def anothermethod(self):

        print "I am the 'anothermethod' method of the SuperB class"

    def athirdmethod(self):

        print "I am the 'anothermethod' method of the SuperB class"

class Sub(SuperA, SuperB):

    def amethod(self):

        print "I am the 'amethod' method of the Sub class"
```

如果我们创建一个Sub类，试图调用它的方法时，如何确定调用那个方法呢？这个规则是先在当前这个类中的方法定义中寻找，如果找不到则按照继承父类的顺序在父类中进行搜索。

继承的层次是通过这个类的父类进行定义的。在这个例子中，继承的层次是：先继承SuperA，然后再继承SuperB。也就是说如果调用Sub类中的一个方法时，没有在Sub中找到，那就就先在SuperA中搜索，如果没有找到就接着在SuperB中搜索。

以下显示了在Python解释器中调用这个程序的例子：

```
>>> instance = Sub()
```

```
>>> instance.amethod()

I am the 'amethod' method of the Sub class

>>> instance.anothermethod()

I am the 'anothermethod' method of the SuperA class

>>> instance.athirdmethod()

I am the 'anothermethod' method of the SuperB class
```

这个例子显示了如何通过继承层次来确定类行为。在非 Zope 应用程序中，只能通过这样一种方式来确定对象的行为。而在 Zope 中对象通过获取机制来搜索确定行为。

2. 获取机制的核心

获取机制简单的说，就是：

对象可以位于其它对象之中。这些对象可以起到容器的作用。比如一个名为 DTML_Example 文件夹中有一个名为“amethod”的 DTML 方法，那么 DTML_Example 就是 amethod 的容器。

对象可以从它的容器对象获得行为。

继承的方式可以保证通过按照继承层次来获得对象的行为。而获取机制则是通过包含的层次来获得对象的行为。在 Zope 中，先按照对象的继承层次来搜索方法和属性，如果没有找到，就再通过获取机制按照容器的层次来搜索方法和属性。

3. 举例说明

接下来举例说明。 在 Zope 的 root 文件夹中添加 id 为 acqusition_test 的 DTML 方法:

```
<html>

<body>

  <p>

   I am being called from within the <dtml-var id> Folder!

  </p>

</body>

</html>
```

保存以后，通过点击 View 标签，可以看到如下结果：

I am being called from within the Zope Folder!

Zope 的 root 文件夹的 id 是 Zope，此时显示的是 Zope。现在，在 root 文件夹中创建一个子文件夹：AcquisitionTestFolder 。我们直接对 AcquisitionTestFolder 文件夹调用 acquisition_test 方法。假设 Zope 运行的端口为 8080，如果访问 http://localhost:8080/AcquisitionTestFolder/acquisition_test，则会显示:

I am being called from within the AcquisitionTestFolder Folder!

就会发现，即使在 AcquisitionTestFolder 文件夹中不存在 acquisition_test ，Zope 依然可以正确显示结果。它显示的结果中，不是 root 文件夹的 id，而是 AcquisitionTestFolder 的 id！这就是获取机制的作用。其核心的思想就是如果对象没有在当前的容器中找到，就沿着容器的层次搜索，直到找到为止。通过这样一种方式，可以给对象添加行为。在这个例子中，我们给 AcqusitionTestFolder 文件夹添加了一个行为（acquisition_test ）。

4. 提供服务

通过获取机制，可以按照包含的层次来获得服务。就刚才的例子来说， AcquisitionTestFolder 获得了 acquisition_test 方法提供的服务。

对象不仅可以获取服务，还可以提供服务。比如，在 AFolder 文件夹中添加一个“邮件主机”（Mail Host）对象，那么这个文件夹中的其它对象就可以发送邮件。同时也让下级文件夹中的对象可以发送邮件。

获取机制的作用体现在两个方面，一个是新创建的对象可以自动从其它对象获取服务，一个是自动给其它对象提供服务。这样就使得共享服务变得很容易。

5. 总结

通过获取机制，可以在整个系统中来分布行为。也就是不用指定对象的所有行为，只要确定其核心行为就可以了，其它的行为可以通过其它对象来提供。这样就给 Zope 应用提供了强大的灵活性。

第八章 DTML 基础

DTML(文档模板标记语言)是能够用于创建动态 HTML 和文本的模板工具。DTML 是一种服务器端运行的脚本语言。它类似于嵌入 HTML 的脚本语言，比如 JSP、PHP 或 mod_perl。

1. 何时使用 DTML

如果你想创建一些由共享的组件动态生成的 web 页面，并且不需要程序开发人员和设计人员大量沟通的情况下，比较适合使用 DTML。如果动态创建不是 HTML 类型的数据，也可以使用 DTML。

2. 何时不使用 DTML

DTML 适合用于处理页面显示，不适合进行复杂的逻辑处理和计算，也不适合进行字符串处理。比较好的一种方式是通过 Python 脚本程序来完成逻辑处理或计算，以及字符串处理，然后通过 DTML 来调用。

3. DTML 方法和 DTML 文档的区别

“DTML 方法”是方法对象。这种对象依赖于容纳他们的容器。比如有一个 AFolder 文件夹，里面有一个 AMethod：

```
AFolder/

        AMethod
```

AMethod 是 AFolder 的一个方法。这就意味着 AMethod 不需要有自己的属性，而是使用 AFolder 的属性。假设在 AMethod 中含有以下语句： -- <dtml-var id> -- 当你调用这个 AMethod 的时候，会看到显示的是 AFolder 的 id。这个 id 是 AFolder 的属性。而 DTML 文档不是方法，它们会调用自己的属性。比如，如果在 AFolder 中有一个名为 ADocument 的 DTML 文档，同样输入上边的语句，然后调用它，就会发现显示的是自己的 id，而不是文件夹的 id。在本章中，如果没有说明，则主要采用的是 DTML 方法，不是 DTML 文档！

4. DTML 标记符句法

DTML 包含两种类型的标记符，独立标记符（singleton）和块标记符（block tags）。独立标记符由一对合拢的小于号（<）和大于号（>）组成。var 标记符就是一个独立标记符的例子：

```
        <dtml-var parrot>
```

不需要用</dtml-var>结束 var 标记符。

块标记符由两个标记符组成－－开始块的标记符和关闭块的标记符，二者之间是内容：

```
<dtml-in mySequence>

  <!-- this is an HTML comment inside the in tag block -->

</dtml-in>
```

开始标记符开始块，关闭标记符结束块。关闭标记符和开始标记符有相同的名称，只是名称前面多了一个斜线。这与 HTML 和 XML 所使用的习惯相同。

5. DTML 标记符名称、目标和属性

所有的 DTML 标记符都有名称。名称就是符号“dtml-”后边的单词，比如标记符“dtml-var”中就是 var，“dtml-in”中就是 in。

多数 DTML 标记符中都有目标。目标就是名称后边的部分。比如对于“<dtml-var standard_html_header>”目标就是 standard_html_header，对于“<dtml-in foo>”就是 foo。目标的含义就是指要进行的操作所指向的对象。

所有的标记符都有属性。通过属性可以确定执行的方式。一些属性是可选的。比如对于 var 标记符，它有一个可选的默认值属性：

```
<dtml-var wingspan missing="unknown wingspan">
```

如果没有找到 wingspan 变量，就使用 missing 中指定的数值。 有些属性不一定用来指定数值。比如，下边的例子中通过 upper 属性可以把变量 exclamation 的数值转换成大写字母:

```
<dtml-var exclamation upper>
```

在 DTML 参考中详细讲述了每个标记符的属性。

6. 创建演示程序

在 root 文件夹中创建一个 id 为 DTML_Examples 的文件夹。这个文件夹用于存放本章中的例子。

6.1. 使用 DTML 完成任务

接下来，我们将演示如何用 DTML 完成三个常见任务：在页面中插入文本，通过迭代序列显示文本，处理表单结果。

6.1.1. 通过 DTML 在页面中插入文本

在 HTML 中插入变量数值，是最基本的功能。许多 DTML 标记府都支持插入变量数值。接下来，举例说明。

在刚才创建的文件夹中创建一个 id 为"Feedbags"，title 为"Bob's Fancy Feedbags"的文件夹。在 Feedbags 中创建 id 为 pricelist 的 DTML 方法，其代码如下：

```
<dtml-var standard_html_header>

<h1>Price list for <dtml-var title></h1>

<p>Hemp Bag $2.50</p>

<p>Silk Bag $5.00</p>

<dtml-var standard_html_footer>
```

当通过点击“View”标签观看这个 DTML 方法时，会把运行的结果显示出来。将显示一个完整的页面，如下图所示：

查看 pricelist 方法

如果通过浏览器查看 HTML 源代码，可以看到下面一些代码：

```
<html><head><title>Feedbags</title></head><body bgcolor="#FFFFFF">
<h1>Price list for </h1>
<p>Hemp Bag $2.50</p>
<p>Silk Bag $5.00</p>
<p><a href="http://www.zope.org/Credits" target="_top">
   <img src="http://localhost:8080/p_/ZopeButton"
        width="115" height="50" border="0" alt="Powered by Zope" />
   </a>
</p>
</body>
</html>
```

DTML 可以重复调用同一内容。在上边的例子中，调用了 standard_html_header 和 standard_html_footer 这两个位于根目录中的 DTML 方法。把运行以后的结果返回到浏览器中显示。

6.1.2. 显示序列

DTML 可以用来显示序列。序列就是条目列表，比如："Fred, Joe, Jim"。这样就可以通过表格或列表的形式来显示这些数据。接下来，举例说明。

在当前目录中创建一个 Script 对象，id 为 actors，代码为：

```
## Script (Python) "actors"

##bind container=container

##bind context=context

##bind namespace=

##bind script=script

##bind subpath=traverse_subpath

##parameters=

##title=

##

return ['Jack Lemmon', 'Ed Harris','Al Pacino', 'Kevin Spacey', 'Alan Arkin']
```

注意，编辑代码的时候都需要位于左边。这段程序返回一个字符串列表。通过使用 dtml-in 标记府可以对它进行迭代。下面创建一个 DTML 方法，id 为 showActors，代码部分为：

```
<html>

<body>

<h1>Actors in the movie Glengarry Glen Ross</h1>

<table border="1">

  <th>Name</th>

<dtml-in actors>
```

```
  <tr>
  <td><dtml-var sequence-item></td>
  </tr>
</dtml-in>
</table>
</body>
</html>
```

通过使用DTML中的in标记符，对actors脚本运行的结果进行迭代，让每条数据对应表表格中的一行。在这里使用了特殊的变量名称： sequence-item。sequence-item 是在dtml-in标记符中使用的一个特殊变量，它用于表示迭代过程中的变量。当通过管理界面中的View视图进行查看时，会显示如下结果：

```
<html>
<body>
<h1>Actors in the movie Glengarry Glen Ross</h1>
<table border="1">
  <th>Name</th>
        <tr>
  <td>Jack Lemmon</td>
  </tr>
        <tr>
  <td>Ed Harris</td>
  </tr>
        <tr>
  <td>Al Pacino</td>
```

```
  </tr>
        <tr>
  <td>Kevin Spacey</td>
  </tr>
        <tr>
  <td>Alan Arkin</td>
  </tr>
      </table>
</body>
</html>
```

注意，在程序中指需要指出要调用的对象名称就可以了。在这个例子中是actors。自动会把运行的结果数据传递到循环处理过程中。

6.1.3. 处理表单输入

通过DTML可以进行表单输入数据的处理。创建一个名为“infoForm”的DTML方法，代码如下：

```
<dtml-var standard_html_header>
<p>Please send me information on your aardvark adoption
program.</p>
<form action="infoAction">
name: <input type="text" name="user_name"><br>
email: <input type="text" name="email_addr"><br>
<input type="submit" name="submit" value=" Submit ">
</form>
<dtml-var standard_html_footer>
```

这是一个请求用户输入有关信息的表单，需要用户输入 name 和 email。要调用的是 infoAction。“action”是 HTML 中的一个属性，用于数据处理所需的方法。

接下来，继续创建一个名为 infoAction 的 DTML 方法。用于显示用户提交的信息。代码为：

```
<dtml-var standard_html_header>

<h1>Thanks <dtml-var user_name></h1>

<p>We received your request for information and will send you

email at <dtml-var email_addr> describing our aardvark adoption

program as soon as it receives final governmental approval.

</p>

<dtml-var standard_html_footer>
```

现在就可以返回到 infoForm 方法，通过 View 视图运行它。然后输入信息，提交。应该显示感谢信息，结果如下：

提交后的结果

REQUEST 也是一种 Zope 对象，它包含来自用户的 web 请求信息。当用户点击提交按钮后，就产生 REQUEST 对象。infoAction 方法通过 REQUEST 对象查找所需信息。在这个例子中，通过使用 user_name 和 email_addr 变量来确定数值。

现在通过一个例子来看看 REQUEST 中的具体内容。创建一个名为 show_request 的 DTML 方法，代码为：

<dtml-var REQUEST>

这个方法用于显示REQUEST对象。然后，修改infoForm方法，代码如下：

<dtml-var standard_html_header>

Please send me information on your aardvark adoption

program.

<form action="show_request">

name: <input type="text" name="user_name">

email: <input type="text" name="email_addr">

<input type="submit" name="submit" value=" Submit ">

</form>

<dtml-var standard_html_footer>

现在点击infoForm的View标签，然后输入一些信息，提交，就会看到类似于下面的信息：

form

 submit ' Submit '

 email_addr 'chrism@zope.com'

 user_name 'Chris'

cookies

 tree-s 'eJzTiFZ3hANPW/VYHUOALlYElA'

lazy items

 SESSION <bound method SessionDataManager.getSessionData of <SessionDataManager instance at 897d020>

other

 AUTHENTICATION_PATH ''

user_name 'Chris'

PUBLISHED <DTMLMethod instance at 8a62670>

submit ' Submit '

SERVER_URL 'http://localsaints:8084'

email_addr 'chrism@zope.com'

tree-s 'eJzTiFZ3hANPW/VYHUOAL1YElA'

URL 'http://localsaints:8084/DTML_Example/show_request'

AUTHENTICATED_USER admin

TraversalRequestNameStack []

URL0 http://localsaints:8084/DTML_Example/show_request

URL1 http://localsaints:8084/DTML_Example

URL2 http://localsaints:8084

BASE0 http://localsaints:8084

BASE1 http://localsaints:8084

BASE2 http://localsaints:8084/DTML_Example

BASE3 http://localsaints:8084/DTML_Example/show_request

environ

SCRIPT_NAME ''

HTTP_ACCEPT_ENCODING 'gzip, deflate, compress;q=0.9'

SERVER_PORT '8084'

PATH_TRANSLATED '/DTML_Example/show_request'

HTTP_ACCEPT 'text/xml...'

GATEWAY_INTERFACE 'CGI/1.1'

HTTP_COOKIE 'tree-s="eJzTiFZ3hANPW/VYHUOAL1YE1A"'

HTTP_ACCEPT_LANGUAGE 'en-us, en;q=0.50'

REMOTE_ADDR '192.168.1.3'

SERVER_NAME 'saints'

HTTP_USER_AGENT 'Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.1a+) Gecko/20020629'

HTTP_ACCEPT_CHARSET 'ISO-8859-1, utf-8;q=0.66, *;q=0.66'

CONNECTION_TYPE 'keep-alive'

channel.creation_time 1027876407

QUERY_STRING 'user_name=Chris&email_addr=chrism%40zope.com&submit=+Submit+'

SERVER_PROTOCOL 'HTTP/1.1'

HTTP_KEEP_ALIVE '300'

HTTP_HOST 'localsaints:8084'

REQUEST_METHOD 'GET'

PATH_INFO '/DTML_Example/show_request'

SERVER_SOFTWARE 'Zope/(unreleased version, python 2.1.3, linux2) ZServer/1.1b1'

HTTP_REFERER 'http://localsaints:8084/DTML_Example/infoForm'

注意，上边的信息中，每一部分（form, cookies, lazy items, other, environ）都表示了一种名称空间。Zope 通过这些名称空间搜索相应的变量数值。关于 REQUEST，还可以参考帮助系统，以及 Zope API 部分。

6.1.4. 处理错误

让我们看看如果直接调用 infoAction 方法会出现什么结果。直接点击 infoAction 的 View 标签，会发现显示了如下结果：

由于变量没有找到返回的错误信息

Zope 没有找到user_name变量。这样的信息可能会在学习Zope过程中经常出现。如果没有找到相应的变量就会出现这样的错误信息。要想仔细查看错误信息，还可以通过查看根目录中的error_log对象。 如果此时，查看error_log对象，你会发现问题出现在以下一行：

```
<h1>Thanks <dtml-var user_name missing="Anonymous User"></h1>
```

理解DTML搜索变量的方式会帮助你快速的找到出现问题的原因。

6.2. 动态获取内容

Zope如果在当前对象中没有找到变量，那么自动会在上一级目录中搜索。这样的话，就可以按照上下的层次来查找内容和行为。获取机制就起到这个作用。

在上面的例子中，可以看到，通过获取机制，就可以找到相应的 standard_html_header 变量值。即：

```
<dtml-var standard_html_header>
```

standard_html_method 对象处于什么位置并不重要，Zope会自动在数据库中向上搜索，指导搜索到root目录为止。 通过这样一种方式，就可以根据需要来安排网站页眉文件的位置。如果在下级目录中新创建了一个standard_html_header 对象，就会让这个下级目录中的对象都调用这个新的standard_html_header。

接下来，举例说明。 在“sandbox”中创建一个新的文件夹，id为“Green”。然后在这个文件夹中创建一个id为“welcome”的DTML 方法，它的代码如下：

```
<dtml-var standard_html_header>
```

```
<p>Welcome</p>

<dtml-var standard_html_footer>
```

现在观看这个 welcome 方法。结果如下：

Welcome 方法

接下来，定制这个页面的页眉。继续在文件夹 Green 中创建一个 id 为 standard_html_header 的 DTML 方法。其代码如下：

```
<html>

<head>

  <style type="text/css">

  body {color: #00FF00;}

  p {font-family: sans-serif;}

  </style>

</head>

<body>
```

这段代码只用来起到网页中的页眉的作用。它通过 CSS(层叠样式表)来调整页面的外观和感觉。

现在通过点击 View 标签来看看这个 welcome 方法。结果如下：

定制页眉以后的 Welcome 方法

可以看到生成的页面出现了新的外观。这是因为它使用了 Green 文件夹中新的页眉。这个页眉对象可以用在 Green 文件夹和下级文件夹中的所有的页面中。 也可以继续按照这样的方式来给下级文件夹中的页面定制页眉。 这样就可以快速的为站点不同部分定制出不同风格的页眉。

6.3. 使用 Python 表达式

我们已经看到了一些 DTML 标记符。比如：

<dtml-var getHippo>

它将插入变量 getHippo 的值。DTML 会自动进行处理。这种句法称为名称句法，还有一种是表达式句法。

通过使用表达式，可以传递参数，更为方便而灵活的进行调用。表达式是一种标记符属性，它是一小段 Python 代码。也可称作做 Python 表达式。

Python 表达式不同于 Python 语句。比如，python 语句“a = 1”把数值 1 分配给变量 a。DTML 中不能使用这样的语句。还比如不能使用 print "x" 这样的语句。这样的语句不是表达式。表达式必须包含数值、变量和运算符。关于 Python 表达式可以参考 Python 书籍。

表达式一般都会返回数值。比如对于表达式"a == 5"，当 a 等于 5 时返回整数 1，当 a 不等于 5 时返回 0。DTML 中调用返回的运算结果。

DTML 中如果直接调用对象名称，则会自动进行一些处理，而如果使用表达式，则不会进行自动处理。比如对于 <dtml-var standard_html_header>，会自动把运行结果插入到页面中。而如果使用 <dtml-var expr="standard_html_header"> ，则不会调用 standard_html_header，而是显示一段字符串。如果要以表达式方式调用 standard_html_header，则需要明确的提供参数。

接下来，让我们来创建一个名为 getHippo 的 Script 对象，用来通过表达式方式进行调用。它需要提供一个参数。 在 sandbox 文件夹中创建一个名为 getHippo 的 Script，代码如下：

```
## Script (Python) "getHippo"

##bind container=container

##bind context=context

##bind namespace=

##bind script=script

##bind subpath=traverse_subpath

##parameters=trap

##title=

##

return 'The hippo was captured with a %s.' % trap
```

注意这个 Script 需要提供一个参数，它不是一个可选参数。 接下来，创建一个调用 getHippo 的 DTML 方法，id 为 showHippo，代码为：

```
<dtml-var expr="getHippo('large net')">
```

在这里，就使用了一个 Python 表达式，调用 getHippo，并且传递一个字符串参数：large net。现在调用这个 DTML 方法，可以看到如下结果：

```
The hippo was captured with a large net.
```

如果我们换用以下代码：

```
<dtml-var getHippo>
```

然后运行，会出现如下的错误信息：

```
Error Type: TypeError
```

```
Error Value: getHippo() takes exactly 1 argument (0 given)
```

错误信息说明，getHippo 需要一个参数。

表达式可以增强 DTML 的功能。比如，用于条件测试：

```
<dtml-if expr="foo < bar">
  Foo is less than bar.
</dtml-if>
```

过多的使用表达式带来的一个负面影响是会使代码难以阅读。因此，如果逻辑判断过程比较多，应该考虑使用 Script 来进行逻辑处理。

6.4. DTML 表达式注意事项

使用 Python 表达式需要注意一些事情。一个经常容易范的错误是弄不清楚表达式句法和基本标记符句法的区别。例如：<dtml-var objectValues> 和 <dtml-var expr="objectValues">

最终给你两个截然不同的结果。第一个例子是 DTML var 标记符，它自动处理变量。换句化说，它试图插入你的变量，做正确的事情，而不管那个变量会是什么。通常这意味着如果变量是一个方法，调用它时要带有适当参数。 在表达式中，你拥有对变量处理方式的完全控制。在我们的例子当中，objectValues 是一个方法。因此， <dtml-var objectValues> 调用这个方法，然而 <dtml-var expr="objectValues"> 没有调用这个方法，它仅仅试图插入它。结果不是一个对象列表，而是一个字符串，例如 <Python Method object at 8681298>。如果你曾经看到这样的结果，你或许正在返回一个方法，而不是正在调用它。 要想从表达式调用一个方法，你必需使用标准的 Python 调用句法，方式是使用园括号：

```
<dtml-var expr="objectValues()">
```

如果你使用 Python 表达式，你就必需知道你正在插入什么类型的变量，并且你必需使用正确的 Python 句法来适当的处理变量。 在你离开变量表达式主题之前，我们应该提及一点，对于一个变量表达式，你可以省去 expr= 部分。但是请不要这样做。<dtml-var aName> 和 <dtml-var "aName"> 太容易混淆了。 如果你这样做了，你得到两个完全不同的结果。这些快捷方式很早以前就被加入到了 DTML 中，但是我们现在不鼓励你使用它们，除非你对于来自于微妙的语法不同而导致的混乱和调试问题做好了准备。

7. 常用的 DTML 标记符

接下来，讲述几个最常用的 DTML 标记符: var, if ,else, elif, in。

7.1. The Var Tag

var 标记符把变量插入到 DTML 方法和文档。我们已经看到许多关于 var 标记符如何被用来把字符串插入到 Web 页面中的例子。就像你已经看到的，var 标记符首先在当前对象中查找变量，然后在它的容器中，最后在 Web 请求中。 var 标记符还能使用 Python 表达式来在定位和调用变量方面提供更多的的控制。

7.1.1. var 标记符属性

你通过使用 var 标记符的属性可以控制它的行为。var 标记符有许多属性，这些属性在通常的格式化方面帮助你。以下列出了一些 var 标记符属性：

html_quote ——这个属性使得插入的值成为 HTML 引用。这意味着<、 >和 &被转义。在 Zope2.6.2 中，默认使用 html_quote。

missing ——这个属性使你能够在 Zope 不能找到变量的情况下指定使用一个默认值。例如：

```
<dtml-var bananas missing="We have no bananas">
```

fmt ——这个 fmt 属性使你能够控制 var 标记符输出的格式。存在许多格式，在“DTML 参考”中进行了详细阐述。

fmt 属性的一种用法是格式化货币值。例如，在你的根文件夹中创建一个名为 adult_rate 的浮点属性。这个属性表示了一个成年人参观动物园的价钱。给这个属性赋值为 2.2。 你可以像以下这样在一个 DTML 文档或方法中显示这个价钱：

```
One Adult pass: <dtml-var adult_rate fmt=dollars-and-cents>
```

这会正确打印“$2.20”。它能够以四舍五入的方式把十进制数字精确到美分。

7.1.2. var 标记符实体句法

Zope 只为简单的 var 标记符提供一个快捷 DTML 句法。因为 var 标记符是一种独立标记符，它能够以一个 HTML 实体的形式出现，例如：

```
&dtml-cockatiel;
```

等同于：

```
<dtml-var name="cockatiel" html_quote>
```

使用实体句法的主要原因是为了避免 HTML 标记符中含有 DTML 标记符。例如：

```
<input type="text" value="<dtml-var name="defaultValue">">
```

为了让你和文本编辑器更为容易的理解它，你可以使用实体句法来让它更为易读：

```
<input type="text" value="&dtml-defaultValue;">
```

var 标记符实体语句是非常有限的。其中你不能使用 Python 表达式或者一些标记符属性。

7.2. If 标记符

DTML 给你带来的好处之一就是能够定制 Web 页面。定制经常意味着适当的响应测试条件。if 标记符使你能够对条件求值和在此结果基础上执行各种行为。 什么是条件？一个条件或者为真（true）或者为假(false)。通常，所有对象被认为是真，除非它们是 0、None、空序列或者空字符串。 就像 var 标记符那样，你能使用名称句法和表达式句法。以下是一些 DTML 表达式形式的条件：

```
objectValues
```

如果变量 objectValues 存在，值为真。也就是说，找到 objectValues，并且其值不是 0，None，空序列或空字符串。

对于 var 标记符，可以使用名称句法和表达式句法。以下是一些表达式例子：

expr="1" －－ 总为真。

expr="rhino" －－ 如果 rhino 变量为真，此表达式为真。

expr="x<5" －－ 如果 x 小于 5，则此表达式为真。

expr="objectValues('File')" －－ 如果通过带有一个 File 参数的方式调用 objectValues 方法返回为真值，则此表达式为真。这个方法在后部分章节有更为详细的介绍。

if 标记符是一种块标记符。如果条件为真，就执行 if 标记符内部的块。 以下给你显示了如何通过 if 标记符使用一个变量表达式来测试一个条件：

```
<p>How many monkeys are there?</p>

<dtml-if expr="monkeys > monkey_limit">

  <p>There are too many monkeys!</p>

</dtml-if>
```

在以上的例子中，如果 Python 表达式 monkeys > monkey_limit 为真，那么你将看到第一个和第二个 HTML 段落。如果测试条件为假，你将只能看到第一个段落。 if 标记符可以嵌套至任何深度，例如：

```
<p>Are there too many blue monkeys?</p>

<dtml-if "monkeys.color == 'blue'">

  <dtml-if expr="monkeys > monkey_limit">

    <p>There are too many blue monkeys!</p>
```

```
  </dtml-if>

</dtml-if>
```

嵌套的 if 标记符首先测试第一个条件，然后如果那个条件为真，就测试第二个条件。通常，DTML 的 if 标记符的工作方式非常像 Python 的 if 语句。

7.2.1. 名称句法和表达式句法的不同

名称句法检测是否存在一个名称，以及它的值。例如：

```
<dtml-if monkey_house>

  <p>There <em>is</em> a monkey house Mom!</p>

</dtml-if>
```

如果 monkey_house 变量不存在，那么这个条件为假。如果有一个 monkey_house 变量，但是它为假，那么这个条件也为假。如果存在一个 monkey_house 变量并且它不是 0、None、空序列或空字符串，此时条件才为真。

Python 表达式句法不检测变量是否存在。这是因为表达式必须为有效 Python 语句。例如：

```
<dtml-if expr="monkey_house">

  <p>There <em>is</em> a monkey house, Mom!</p>

</dtml-if>
```

只要 monkey_house 存在，它就会象被希望的那样工作。如果 monkey_house 变量不存在，当 DTML 试图找到这个变量时，Zope 引发一个 KeyError 例外。

7.3. else 和 elif 标记符

if 标记符仅让你能够在条件为真的情况下才执行一个行为。你也许需要在条件为假的情况下执行不同的行为。这可以通过 DTML 中的 else 标记符实现。if 块也可以包含一个 else 独立标记符。例如：

```
<dtml-if expr="monkeys > monkey_limit">

  <p>There are too many monkeys!</p>

<dtml-else>

  <p>The monkeys are happy!</p>
```

```
</dtml-if>
```

else 标记符把 if 标记符块分割成两个块，如果条件为真就执行第一个块，如果条件不为真就执行第二个块。

一个 if 标记符块还可以包含一个 elif 独立标记符。elif 标记符指定另外一个条件，就像一个额外的 if 标记符。这样可以使你在一个块中指定多个条件：

```
<dtml-if expr="monkeys > monkey_limit">
  <p>There are too many monkeys!</p>
<dtml-elif expr="monkeys < minimum_monkeys">
  <p>There aren't enough monkeys!</p>
<dtml-else>
  <p>There are just enough monkeys.</p>
</dtml-if>
```

一个 if 标记符块可以包含任意多个 elif 标记符，但是只有一个 else 标记符。else 标记符通常必须在 elif 标记符后出现。elif 标记符可以测试使用名称句法或者表达式句法的条件。

7.4. 通过 if 标记符使用 Cookie

让我们看一个具体一些的 if 标记符例子。当访问者来到你的站点，你经常需要给他们一个 cookie，用以通过某种特殊的数值识别他们。在 Internet 里经常使用 cookie，并且当它们被正确的实现时，它们就会非常有用。

假设我们想区分新访问者和已访问者。当一个用户访问这个站点，我们可以设置 cookie。然后我们可以在显示页面时测试 cookie。如果用户已经到过这个站点，他们有 cookie。如果他们还没有 cookie，意味着他们是新用户。

假设我们正在运行一个特例。对于动物园首次访问者实行半价。以下是一个 DTML 片断，它使用 hasVisitedZoo 变量测试 cookie，然后根据访问者是否为新访问者显示价格。

```
<dtml-if hasVisitedZoo>
  <p>Zoo admission <dtml-var adult_rate fmt="dollars-and-cents">.</p>
<dtml-else>
  <b>Zoo admission for first time visitors
      <dtml-var expr="adult_rate/2" fmt="dollars-and-cents"></p>
```

```
</dtml-if>
```

这个片断测试hasVisitedZoo变量。如果用户以前已经访问过这个动物园，它就显示出正常价格。如果访问者首次访问这里，他们得到半价优惠。

为了完整的显示这个例子，以下是一个用基于Python的脚本实现的hasVisitedZoo方法：

```
## Script(Python) "hasVisitedZoo"

##

Returns true if the user has previously visited

the Zoo. Uses cookies to keep track of zoo visits.

request = context.REQUEST

response = request.RESPONSE

if request.has_key('zooVisitCookie'):

    return 1

else:

    response.setCookie('zooVisitCookie', '1')

    return 0
```

在“高级Zope脚本”一章中，我们更为详细的论述如何用Python编写事物逻辑脚本。对于目前来说，足以显示出搜索cookie的方式，它根据是否找到cookie而返回真值或假值。注意Python是如何使用和DTML中的if和else标记符类似的if和else语句的。DTML的if和else标记符是基于Python的。事实上，就像DTML ，Python也有一个elif语句。

7.5. In标记符

DTML的in标记符迭代一个对象序列，为每个序列项完成一个执行块。在编程中，这经常被称为迭代（iteration）或循环（looping）。

in 标记符就像if标记符一样是一种块标记符。对应于在in标记符循环中的每次迭代，in标记符块中的内容被执行一次：

```
<dtml-in todo_list>

  <p><dtml-var description></p>
```

```
</dtml-in>
```

这个例子遍历一个名为 todo_list 的对象列表。它为列表中的每个数据项插入一段描述该数据项的 HTML 段落。

迭代在许多 Web 任务中非常有用。设想一个显示待售房屋的站点。用户通过你的站点搜索符合某个标准的房屋。你需要在页面中按照一致的方式格式化所有的搜索结果。因此，你需要迭代每个结果，一次一条，并为每个搜索结果提供一个类似的 HTML 块。

从某种角度说，in 标记符块中的内容是一种模板，为序列中的每个数据项应用一次这个模板。

7.5.1. 迭代文件夹内容

这里是一个如何迭代文件夹内容的例子。这个 DTML 遍历文件夹中的所有文件并为每个文件显示一个链接。这个例子给你说明如何显示文件夹中的所有文件，因此要运行这个例子，就像在“使用基本 Zope 对象”一章中讲述的那样，你需要上载一些文件到 Zope 里：

```
<dtml-var standard_html_header>

<ul>

<dtml-in expr="objectValues('File')">

  <li><a href="&dtml-absolute_url;"><dtml-var title_or_id></a></li>

</dtml-in>

</ul>

<dtml-var standard_html_footer>
```

以上的代码显示以下的文件清单，如下图所示：

迭代一个文件列表

让我们一步一步的看看这个DTML例子。首先，使用var标记符在这个文档中插入你的普通页眉。然后，指出让浏览器显示一个HTML圆点列表，你用了HTML中的ul标记符。

接下来，有一个in标记符。这个标记符有一个表达式，它调用名为objectValues的Zope API方法。这个方法返回一个当前文件夹里符合给定标准的对象序列。在这个例子中，对象必须是文件。这个方法调用返回当前文件夹里的文件列表。

in标记符遍历这个序列中的每个数据项。如果在当前文件夹中存在四个文件对象，那么in标记符执行它的块中的代码四次——为对象序列中的每个对象执行一次。

在每次迭代期间，in标记符首先在当前对象中查找变量。在“高级DTML”一章中，我们将仔细的看看DTML如何查找变量。

如以下例子所示，这个in标记符迭代一个文件对象集合，并且使用var标记符在每个文件中查找变量：

```
<dtml-in expr="objectValues('File')">

  <li><a href="&dtml-absolute_url;"><dtml-var title_or_id></a></li>

</dtml-in>
```

第一个var标记符是一个实体，第二个是一个普通DTML中的var标记符。当in标记符遍历第一个对象，它把absolute_url和title_or_id变量插入到第一个圆点列表项中：

```
<ul>

  <li><a href="[http://localhost:8080/FirstFile]">FirstFile</a></li>
```

第二次迭代期间，把第二个对象的absolute_url 和title_or_id 变量插入到输出中：

```
<ul>
  <li><a href="[http://localhost:8080/FirstFile]">FirstFile</a></li>
  <li><a href="[http://localhost:8080/SecondFile]">SecondFile</a></li>
```

这个过程持续到in标记符迭代完当前文件夹中每个文件。当in标记符迭代完毕，最后用</url> HTML标记符结束你的HTML圆点列表，然后插入standard_html_footer，结束文档。

7.5.2. in标记符特殊变量

in标记符给你提供了一些有用的信息，在你迭代一个序列的同时，它们可以使你定制HTML。例如，你可以让你的文件库更容易阅读，方法是通过把它放入一个HTML表格中，并且隔行换用颜色，如下图5所示。

使用交替颜色显示的文件清单

in 标记符使这种工作简单了。对你的文件库方法略做如下修改：

```
<dtml-var standard_html_header>
<table>
<dtml-in expr="objectValues('File')">
  <dtml-if sequence-even>
```

```
    <tr bgcolor="grey">
  <dtml-else>
    <tr>
  </dtml-if>
  <td>
  <a href="&dtml-absolute_url;"><dtml-var title_or_id></a>
  </td></tr>
</dtml-in>
</table>
<dtml-var standard_html_footer>
```

在上面的例子中，使用了一个 if 标记符来测试名为 sequence-even 的特殊变量。in 标记符在每次循环中赋予这个变量真值或假值。如果当前的迭代数字为偶数，那么值为真。如果迭代数字是奇数，它就为假。

这个测试的结果是针对序列中的间隔对象把一个 tr 标记符（不管背景是灰色还是无色）插入到了文档中。如你所预期的，奇序列总是具有与偶序列相反的的值。

in 标记符为你定义了许多特殊变量。以下是最普通的和最常用的变量列表：

- sequence-item —— 这个特殊变量是迭代里的当前数据项。在文件库例子中，每次循环，当前迭代文件被分配给序列项。它经常用来得到当前迭代对象的引用。 - sequence-index —— 目前为止已经完成迭代的当前数字，从 0 开始。如果这个号码是偶数，sequence-even 为真并且 sequence-odd 为假。 - sequence-number —— 目前为止已经完成迭代的当前数字，从 1 开始。它可以被看成是循环中当前对象的基本位置（第一，第二，第三等等）。如果这个号码是偶数，sequence-even 为假并且 sequence-odd 为真。 - sequence-start —— 只对第一次迭代，这个变量为真。 - sequence-end —— 只对最后一次迭代，这个变量为真。

这些特殊的变量在参考文档中有详细阐述。

DTM 是一种适合创建动态内容的强大工具。它使你能够执行相当复杂的计算。在“高级 DTML”一章里，你会找到更多的 DTML 标记符，以及找到已经见到的这些标记符的强大的使用方法。不管它的能力有多么强大，对于复杂的脚本，你应该尽量避免使用 DTML。在“高级 Zope 脚本”一章中，你会发现如何使用 Python 编写事物逻辑脚本。

第九章 使用 Zope 页面模板

页面模板是一种 web 页面生成工具。它能帮助编程人员和设计人员协作生成适用于 Zope web 应用程序的动态 web 页面。设计人员可以利用它们来维护页面，而不用放弃他们的工具，同时保留了把这些页面嵌入到一个应用程序里所必需的功能。在本章，你将学习有关页面模板的基础，讲述如何在你的 web 站点里使用页面模板轻松的创建动态 web 页面。在“高级页面模板”一章里，将学习相关的高级特性。

页面模板的目标是要让设计人员和编程人员轻松的一起工作。设计人员可以使用所见即所得（WYSIWYG）的 HTML 编辑器来创建一个模板，然后编程人员就可以在编辑处理后融入到应用程序里。如果需要的话，设计人员还可以重新把模板调入编辑器，进一步修改它的结构和样子。只需要一些必要的步骤就可以保留编程人员所做的修改，而设计人员不会影响应用程序。

页面模板的核心思想是：

可以很好的用编辑工具处理。

所见非常相似所得。

除了结构逻辑部分，代码和模板分离。

页面模板就像要生成的页面的模型，而从某种程度上来讲，它又是一个有效的 HTML 页面。

1. Zope 页面模板和 DTML 的对比

Zope 已经有了 DTML,为什么还要另外一种模板语言？首先，DTML 不是针对 HTML 设计人员的。DTML 页面一旦转化为模板，它就不是有效的 HTML，难以脱离 Zope 进行修改。其次，困扰 DTML 的是难以分离表现、逻辑和内容（数据）。这样就降低了内容管理和 web 站点开发的可扩展性。最后，DTML 中自动处理的部分比较多，这样就不容易进行控制。

DTML 可以完成页面模板不能完成的工作，比如动态生成电子邮件信息（页面模板只能生成 HTML 和 XML）。因此，DTML 不会失去作用。但我们确实会看到页面模板几乎可以完成 Zope 里所有的 HTML/XML 表现工作。

2. 页面模板如何工作

页面模板使用模板属性语言(Template Attribute Language (TAL))。TAL 由一些特殊的标记符组成。例如，一个动态的页面标题可以是这样：

```
<title tal:content="here/title">Page Title</title>
```

tal:content 属性就是一种 TAL 语句。由于它有 XML 名称空间（tal:部分），大多数的编辑工具都可以支持它，而不会把它删去。当 WYSIWYG 编辑器或 web 浏览器处理它时，不会更改模板的结构或样子。名称 content 表示它将设置 title 标记符中的内容，值"here/title"是一种表达式，提供了要插入到标记符里的文字。

所有的 TAL 语句由开头为 tal:的标记符属性和与数值组成。一个 TAL 语句的值显示在引号里边。更多相关信息请参见“Zope 页面模板参考”部分。

对于使用WYSIWYG工具的设计人员来说，上边的例子是完全有效的HTML，在编辑器里显示正常。换句话说，页面模板能够很好的支持编辑工具。 这个例子还说明了“所见非常相似所得”原则。当你在编辑器里观看这个模板，标题部分会正常显示。模板提供了一个要被生成的文档的例子。 当这个模板在Zope里保存以后，用户观看时，Zope会把模板的内容变为动态的内容，从而用"here/title"所代表的内容替换"Page Title"。在这个例子里，"here/title"对应将要生成的页面的标题。替换过程是在调用模板时是自动完成的。

有的模板语句可以用来替换整个标记符、内容或部分属性。你可以重复显示标记符多次，或者忽略它。你可以把几个模板的某些部分连接在一起，以及指定简单的错误处理。这些功能可用来生成文档结构。除了这些功能，页面模板不能用来创建继承或类或者执行复杂的流程控制，以及完成复杂的运算。要完成这些复杂的任务，你应当使用基于Python的脚本或应用程序组件。

页面模板语言有自己的用途，它的作用并不侧重强大和通用的特性。这也就意味着它用在一种框架里边（比如Zope），用框架当中的另外一些与页面表现无关的对象来完成逻辑处理和计算任务。

例如，模板语言适合于处理发票单，每条数据生成对应的一行，然后在每行里显示描述、数量、价格等等这样的文字。它不适合用来在数据库里创建发票记录，或者处理信用卡信息。

3. 创建一个页面模板

设计页面的时候，你可能会使用FTP或WebDAV，而不用Zope管理界面(ZMI)来创建和编辑页面模板。有关远程编辑页面模板方面的信息请参见本章后面的“使用FTP和WebDAV”部分。对于本章里的小例子，使用ZMI就够了。

首先通过web浏览器以管理员身份登录进入Zope管理界面。选择一个工作目录（根目录就可以），然后从下拉添加列表中选择"Page Template"。在添加表单里的Id字段中键入"simple_page"，然后按"Add and Edit"按钮。

现在，可以看到这个页面模板的主编辑页面。标题是空的，内容的类型是text/html，默认的模板文字在编辑区里边。

现在让我们创建简单的动态页面。在Title字段里键入单词"a Simple Page"。然后，按照以下内容编辑模板文字：

```
<html>
 <body>
   <p>
     This is <b tal:replace="template/title">the Title</b>.
   </p>
 </body>
</html>
```

现在，按Save Changes按钮。Zope就会显示一条消息，表示你所做的修改已经生效了。

如果出现了以 "Page Template Diagnostics" 起始的HTML注释语句，就请你检查一下，确认是否正确输入了代码，然后再保存一次。这个注释是一条错误消息，告诉你有错误。不需要你去删除它，一旦错误修正了，它自动会消失。

点击Test标签，你会看到一个顶部显示"This is a Simple Page."的页面。注意，文字是纯文本，没有粗体样式。这是因为tal:replace 语句替换了整个标记符。

返回上一步，点击content-type字段下边的 Browse HTML source链接。这样就会给你显示未处理的模板源码。你会看到"This is the Title."。粗体的文字充当了动态标题文本的占位符。为了进一步编辑例子，再一次返回上一步。

Content-Type字段允许你指定页面的内容类型。通常，对于HTML，你将使用text/html内容类型，对于XML,使用text/xml内容类型。

如果你把内容类型设定为text/html，那么Zope使用兼容HTML模式解析你的模板，这种模式允许使用HTML标记符。如果你把内容类型设定为其他的类型，那么Zope假定你的模板是组织良好的XML。为了处理XML，Zope还要求显式声名TAL和METALXML名称空间。比如，如果你想使用XHTML，应该在html标记符中进行以下声明：

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"

  xmlns:metal="http://xml.zope.org/namespaces/metal">
```

在后面的例子中，使用普通的HTML，所以内容类型设定为text/html就够了。 "Expand macros with editing" 选项将在“高级页面模板”一章中阐述。

4. 简单表达式

在页面模板里边的表达式"template/title"是一种路径表达式。这是一种最普通的表达式类型。TAL表达式语法定义中还定义了其他几种表达式类型。详细请参考“Zope页面模板参考”。

"template/title"路径表达式用于提取模板的 title 属性。以下是其它一些常用的路径表达式：

request/URL: 当前web请求的URL。

user/getUserName: 已经授权的用户登录名。

container/objectIds: 同一文件夹中的其他模板对象列表。

每个路径都以一个变量名开始。如果这个变量包含你所需的值，使用这个变量就可以了。否则，你就需要添加一个”/”号，以及下级对象或属性的名称。你可能需要经过多个下级对象来得到要找的值。

Zope定义了一个小型内建变量集，其中比如request和user，这些将在第九章“高级页面模板”里讲述。在这一章里，你还会学习到如何定义你自己的变量。

4.0.1. 插入文字

在上边的例子里，对一个粗体标记符使用了 tal:replace 语句。当你测试时，Zope 用模板的标题替换整个标记符。当你浏览时，你看到了以粗体显示的模板文字。我们使用一个粗体标记符是为了加亮显示不同之处。

为了把动态的文字放置到其他文本里，通常使用 span 标记符而不使用 bold 标记符调用 tal:replace。例如，在上面的例子中加入以下几行：

```
<br>

The URL is <span tal:replace="request/URL">http://www.example.com</span>.
```

Span 标记符不会显示出来，因此当你在编辑器或浏览器里观看源码时，这会显示为"The URL is http://www.example.com."。当你观看运行结果时，会看到：

```
The URL is http://localhost:8080/template_test/simple_page.
```

如果你想把文字插入到一个标记符里，但不影响标记符本身，可以使用 tal:content 语句。要把例子页面的标题设置为模板的 title 属性，可以在 html 和 body 标记符之间加入以下几行：

```
<head>

  <title tal:content="template/title">The Title</title>

</head>
```

如果你在一个新浏览器窗口里打开"Test"标签，这个窗口的标题会成为"a Simple Page"。如果你查看页面的源文件，你会看到以下代码：

```
<html>

  <head>

    <title>a Simple Page</title>

  </head>

...
```

Zope 把模板的标题插入到了 title 标记符里边。

4.0.2. 重复结构

接下来，让我们给 simple_page 模板添加一些内容，要求是显示和模板并列的同一文件夹里的对象列表。需要做一个表，这个表针对每个对象对应一行，并包含几列，分别对应 id,meta-type 和 title。在模板的底部继续添加以下几行：

```
<table border="1" width="100%">
  <tr>
    <th>Number</th>
    <th>Id</th>
    <th>Meta-Type</th>
    <th>Title</th>
  </tr>
  <tr tal:repeat="item container/objectValues">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Meta-Type</td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

其中的 tal:repeat 语句的含义是：针对容器对象值列表里的每个条目重复这一行。Repeat 语句每次处理对象列表中的一个条目，即为 item 变量（称之为重复变量），并且对应这些条目生成表格中的一行。每一行里的"item/getId"变量值是 item 的 Id，"item/meta_type" 和 "item/title"也是这样。

你可以使用任何你喜欢的名称命名重复变量（"item"只是一个例子）,只要它以字母开头，并且只包含字母、数字和下划线('_')。重复变量只在 repeat 标记符里定义。如果你试图在 tr 标记符上边或下边使用它，会显示错误提示。

你还可以使用重复变量来得到当前重复的相关信息。通过把它放置在内建变量 repeat 路径后边，你就可以访问重复的序号，可以是从 0 (index)开始,从 1(number)开始,从 A(Letter)开始，或者以其他一些方式。因此表达式 repeat/item/number 对应第一行为 1，对应第二行为 2，以此类似。

因为一个 tal:repeat 循环可以放置在另外一个循环里，同一时间里可以有多个循环起作用。这就是为什么必须写 repeat/item/number，而不是仅仅写 repeat/number。你必须通过指定循环的名称来区分不同的循环。

现在，观看页面，注意它是如何列出同一文件夹里的所有对象的。试试添加或删除文件夹里的对象，注意页面是如何反映这些变化的。

4.0.3. condition 元素

使用页面模板，你可以动态查询环境变量，并且有选择的插入文本。例如，针对一个 cookie 你可以指定特殊的信息：

```
<p tal:condition="request/cookies/verbose | nothing">

  Here's the extra information you requested.

</p>
```

这段代码的含义是：只有设置了名为 verbose 的 cookie 时，才会在输出中显示这个段落。只有 cookie 中有 verbose 这个变量，表达式"request/cookies/verbose | nothing"才为真。在“高级页面模板”中，将学习更多的内容。

使用 tal:condition 语句，可以检查所有类型的条件。如果表达式为真值，tal:condition 语句就保留这些内容，如果值为假，则是删去整个语句标记符，包括它的内容。Zope 认为数字 0、空字符串、空列表和内建变量 nothing 为假值。几乎其他任何值都为真，包括非零数字，以及包含任何内容的字符串（即使是空格）。

condition 语句的另外一种常见用途是在循环以前检测序列是否为空。就像刚才的例子中讲到的如何根据对象集合绘制表格。以下这个例子显示如果对象列表为空就不绘制表格。继续在 simple_page 模板中添加以下代码，。

```
<table tal:condition="container/objectValues"

       border="1" width="100%">

  <tr>

    <th>Number</th>

    <th>Id</th>

    <th>Meta-Type</th>

    <th>Title</th>

  </tr>

  <tr tal:repeat="item container/objectValues">
```

```
        <td tal:content="repeat/item/number">#</td>

        <td tal:content="item/getId">Id</td>

        <td tal:content="item/meta_type">Meta-Type</td>

        <td tal:content="item/title">Title</td>

      </tr>

    </table>
```

让我们在 template_test 文件夹中添加三个子文件夹，分别是"1", "2" 和 "3"。然后重新调用 simple_page 模板，会看到显示如下表格：

Number	Id	Meta-Type	Title
1	simple_page	Page Template	
2	1	Folder	
3	2	Folder	
4	3	Folder	

如果表达式 container/objectValues 值为假，那么整个表格就被忽略，不显示。

4.1. 更改属性

大多数情况下，模板要列出的对象有一个 icon 属性，这个属性包含了指向表示这种对象的图标的路径。为了在表格的列中显示这个图标，需要把这个路径插入到 img 标记符的 src 属性里边。按照以下代码编辑表格：

```
    <td><img src="/misc_/OFSP/Folder_icon.gif"

             tal:attributes="src item/icon">

        <span tal:replace="item/meta_type">Meta-Type</span>

    </td>
```

tal:attributes 语句用 item/icon 的值替换 img 标记符的 src 属性。模板里边的 src="/misc_/OFSP/Folder_icon.gif"属性充当一个替代物.

注意，我们已经在表格中使用 tal:content 语句，在 span 标记符中使用了 tal:replace 语句。这些可以在表格单元里放入图象和文本。

5. 用页面模板创建一个文件库

以下是一个在 Zope 里边使用页面模板创建简单文件库的例子，它使用一个模板，一些 Python 代码，以及一些文件。

首先，使用 HTML 编辑器创建一个模拟的文件库页面。比如使用 Macromedia Dreamweaver, Adobe GoLive , and Netscape Composer。 这个页面不需要精心制作，它仅显示一些粗略的信息。下边是仅包含一个文件的文件库页面：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

                      "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

  <title>File Library</title>

  <style type="text/css">

  .header {

    font-weight: bold;

    font-family: helvetica;

    background: #DDDDDD;

  }

  h1 {

    font-family: helvetica;

  }

  .filename {

    font-family: courier

  }

  </style>

  <meta name="GENERATOR" content="amaya 5.1">
```

```
</head>

<body>

<h1>File Library</h1>

<p>Click on a file below to download it.</p>

<table border="1" cellpadding="5" cellspacing="0">

  <tbody>

    <tr>

      <td class="header">Name</td>

      <td class="header">Type</td>

      <td class="header">Size</td>

      <td class="header">Last Modified</td>

    </tr>

    <tr>

      <td><a href="Sample.tgz" class="filename">Sample.tgz</a></td>

      <td>application/x-gzip-compressed</td>

      <td>22 K</td>

      <td>2001/09/17</td>

    </tr>

  </tbody>

</table>

</body>

</html>
```

现在，通过浏览器登录进入你的 Zope 管理界面，创建一个名为 FileLib 的文件夹。在这个文件夹里，通过从添加菜单里选择 Page Template 创建一个名为页面模板 index_html，指定 Id 为 index_html，然后点击 Add。关于用其它方式创建页面模板的方法，可参见“使用外部工具”一章。

现在，用你的 HTML 编辑器，通过 FTP，WebDAV 或 HTTP PUT，保存以上代码，存入到 index_html 中。比如 URL 为：http://localhost:8080/FileLib/index_html。不同的编辑工具采用不同的方法来完成这个任务。详细请参见“使用外部工具”一章。

现在，你已经保存了模板，就可以返回 Zope，点击 index_html，然后点击 Test 标签来观看这个模板。它看起来就像模拟页面一样，一切正常。

现在，让我们给上边的 HTML 添加一些动态特性。首先，我们想让模板的标题是动态的。在 Zope 里，你会注意到页面模板有一个可填写的 title 表单字段。我们想让 Zope 动态的把模板标题插入到处理后的模板内容里。代码如下：

```
<head>

  ...

  <title tal:content="template/title">File Library</title>

  ...

<body>

<h1 tal:content="template/title">File Library</h1>

...
```

现在，更改 index_html 页面模板的标题。保存以后，点击 Test 标签，你就会看到页面模板在输出里动态的插入了模板的标题。

注意这里的 content 标记符属性。这个属性的含义是：用'template/title'变量替换这个标记符的内容。在这个例子里，template/title 是 index_html 页面模板的标题。

下一步是要创建一个动态的文件列表，用于显示 FileLib 文件夹里的所有文件。

开始前，你需要写一行 Python 代码。进入 FileLib 文件夹，创建一个 Script (Python)。让 id 为 files，然后点击 Add and Edit，按照以下内容编辑代码：

```
## Script (Python) "files"

##

return container.objectValues('File')
```

这样就会返回 FileLib 文件夹里的全部文件对象列表。现在，编辑你的 index_html 页面模板，添加一些新的 tal 属性：

```
...
<tr tal:repeat="item container/files">
  <td><a href="Sample.tgz" class="filename"
         tal:attributes="href item/getId"
         tal:content="item/getId">Sample.tgz</a></td>
  <td tal:content="item/content_type">application/x-gzip-compressed</td>
  <td tal:content="item/getSize">22 K</td>
  <td tal:content="item/bobobase_modification_time">2001/09/17</td>
</tr>
...
```

值得注意的是 HTML 标记符 tr 里边的 tal:repeat 属性。这个属性告诉模板根据"container/files"（就是刚才创建的 Script）返回的值进行循环，并对每一个文件创建新的表行。每一次循环，当前文件对象被分配名称为 item.

表格中每一行的单元格都有 tal:content 属性，它描述了应该插入到单元格里的数据。每次循环，用文件的 id, content type,size, modification times 替换相应的内容。另外，链接是通过使用 tal:attributes 来动态指向当前文件并覆盖 href 属性。

数据来自于 item 对象，方式是通过对文件对象调用 Zope API 方法。方法 item/getId, item/content_type, item/getSize, item/bobobase_modification_time 都是标准的 API 函数，这些函数在 Zope 的在线帮助系统里都有说明。

进入 Zope，先把一些文件上载到 FileLib 文件夹里，测试这个脚本。方法是通过从添加菜单里选择 File，点击 upload 按钮。上载完毕后，你只需要点击 Add 就可以了。如果你不指定 id,那么就会使用你上载的文件名。

当上载一些文件后，进入 index_html 页面模板，然后点击 Test 标签。现在，你会看到这个页面模板已经显示为一个非常简单的文件库，其中的一些 HTML 标记符属性已经更改了。

但是目前还存在一些小的问题，文件大小和日期显示不是非常好，并且不匹配。你可能喜欢以 K 或 MB 方式而不是 bytes 方式显示文件大小。以下是一个用于解决这个问题的脚本：

```
## Script (Python) "file_size"
##
```

```
"""

Return a string describing the size of a file.

"""

bytes=context.getSize()

k=bytes/1024.0

mb=bytes/1048576.0

if mb > 1:

    return "%.2f MB" % mb

if k > 1:

    return "%d K" % k

return "%d bytes" % bytes
```

用 file_size 作为 id，在 FileLib 文件夹里创建这个脚本。它以 kilobytes 和 megabytes 方式计算文件大小，并返回一个适当的描述文件大小的字符串。现在，你就可以在 item/getSize 表达式里使用这个脚本了：

```
...

<td tal:content="item/file_size">22 K</td>

...
```

你还可以使用 Python 修正日期格式问题。在 FileLib 文件夹里创建一个名为 file_date 的脚本。

```
## Script (Python) "file_date"

##

"""

Return modification date as string YYYY/MM/DD

"""

date=context.bobobase_modification_time()
```

```
return "%s/%s/%s" % (date.year(), date.mm(), date.day())
```

现在，把"item/bobobase_modification_time"表达式换成这个脚本：

```
...

<td tal:content="item/file_date">2001/09/17</td>

...
```

祝贺你，你已经成功创建了动态页面脚本。这个例子说明了页面模板如何在应用程序里边起到良好的表现层作用。页面模板为应用程序逻辑（即基于 Python 的脚本）提供界面，而应用程序逻辑处理站点里的数据（即那些文件）。

6. 使用 FTP 和 WebDAV 进行远程编辑

你可以使用 FTP 和 WebDAV 远程编辑页面模板，就像 HTTP PUT 方式发布一样。使用这些方法，你就可以使用页面模板，而不用离开像 DreamWeaver 这样的 WYSIWYG 编辑器。

上边的部分给你显示了如何远程使用 HTTP PUT 方式上载页面。你也可以使用 FTP 和 WebDAV 经过相同的步骤完成同样的任务。

在 Zope 管理界面里创建一个页面模板。你可以用任何文件扩展名命名文件。有些人，倾向于.html，有些人倾向于.zpt。注意，在 Zope 里边，一些像 index_html 这样的名称具有特殊的含义。

用你的编辑器编辑文件并保存它。当你保存时，应该使用与上一步相同的源码 URL。

当你编辑完后，可以重新载入你的页面，检查错误注释。关于调试，请参见下一节。

当然，你也可以不使用 Zope 管理界面来创建页面模板。详细请见“使用外部工具”一章。

7. 调试和测试

Zope 能够帮助你查找和修改页面模板里的错误。Zope 进行两次错误检查：一次是在编辑页面模板时，一次是在观看页面模板时。在编辑状态时，Zope 可以捕捉到更多类型的错误。

你可能已经熟悉了 Zope 插入错误注释的捕捉错误的方式。这些注释给你提示出 Zope 在页面模板里发现的问题。这些问题大多数发生在 tal 语句里。例如：

```
<!-- Page Template Diagnostics

 Compilation failed

 TAL.TALDefs.TALError: bad TAL attribute: 'contents', at line 10, column 1

-->
```

这条消息显示，你在模板的第10行错误的使用了tal:contents，而不是tal:content。其他的消息告诉你有关模板表达式和宏方面的问题。

当你使用Zope管理界面来编辑页面模板时，可以轻松显示这些消息。可是，如果你使用WebDAV或FTP，这些消息很容易被忽略掉。例如，如果你通过FTP保存了一个模板，你不会得到一个揭示问题的FTP错误，你不得不从Zope重新载入这个模板来查看错误消息。在使用FTP和WebDAV时，编辑完成后，在Zope里重新载入一次是一种好方法，这样就可以确信是否包含错误消息。

如果你没有注意到错误消息，试图执行带有问题的模板，你会看到类似于这样的错误消息：

```
Error Type: RuntimeError

Error Value: Page Template hello.html has errors.
```

这就告诉你有错误，应该重新载入模板，检查错误消息。

除了当编辑时会遇到这种错误消息外，当你观看页面模板时偶尔还会遇到常见的Zope错误。这些问题往往是由于模板表达式不正确造成的。例如，如果表达式不能定位变量，你就会遇到下边的错误：

```
Error Type: Undefined

Error Value: "unicorn" not found in "here/unicorn"
```

这条错误消息告诉你，它不能找到表达式here/unicorn里引用的变量unicorn。为了帮助你指出出现了什么错误，Zope在回溯里包含了环境信息。如果你处于调试模式里边，这条信息将显示在错误页面的底部。另外，观看错误页面的源文件来查看回溯。回溯将包含环境变量方面的信息。

```
...

'here': <Application instance at 01736F78>,

'modules': <Products.PageTemplates.ZRPythonExpr._SecureModuleImporter instance at 016E77FC>,

'nothing': None,

'options': {'args': ()},

'request': ...

'root': <Application instance at 01736F78>,

'template': <ZopePageTemplate instance at 01732978>,

'traverse_subpath': [],
```

```
'user': amos})

...
```

这些信息有些复杂，在这个例子里，它告诉我们，here 变量为一个应用程序实例。这也就意味着它是顶级 Zope 文件夹（注意，根变量和"Application instance"相同）。出现问题的原因可能是由于对一个有 unicorn 属性的文件夹应用模板，但是这个文件夹没有这个属性。这个回溯没有提供很多的帮助，但是确实可以帮助你。

8. XML 模板

页面模板的灵活性还表现在另外一方便，那就是他们能够象 HTML 那样动态处理 XML。例如，对于以下 XML:

```
<guestbook>
  <entry>
    <comments>My comments</comments>
  </entry>
  <entry>
    <comments>I like your web page</comments>
  </entry>
  <entry>
    <comments>Please no blink tags</comments>
  </entry>
</guestbook>
```

这段 XML 的创建方式是对某个文件夹里的所有 DTML 文档进行循环，然后把它们的源码部分插入到 comment 元素里边。在这一节，我们将给你显示如何使用页面模板来生成相同的 XML。

再比如，创建一个名为"entries.xml"的页面模板，其内容如下：

```
<guestbook xmlns:tal="http://xml.zope.org/namespaces/tal">
  <entry tal:repeat="entry python:here.objectValues('DTML Document')">
    <comments tal:content="entry/document_src">Comment goes here...</comments>
```

```
    </entry>

</guestbook>
```

确认你把内容类型设置成 text/xml，点击 Save Changes，然后点击 Test 标签。如果你正在使用 Netscape，它将提示你下载一个 XML 文档，如果你正在使用 MSIE 5 或更高版本，你可以在浏览器里直接查看 XML 文档。

注意 tal:repeat 是如何对所有的 DTML 文档进行循环的。tal:content 语句把每个文档的源码插入到 comments 元素里边。xmlns:tal 属性是一个 XML 名称空间声明。它告诉 Zope，以 tal 开头的名称是页面模板命令。有关 TAL 和 TALES XML 名称空间方面的更多信息，请参见“Zope 页面模板参考”

使用页面模板创建 XML 几乎完全类似于创建 HTML。最大的不同在于你必须使用显式 XML 名称空间声明。另外一个不同是你应该把内容类型设置为 text/xml 或者任何你的 XML 应该设置的类型。最后一个不同是你通过 source.xml 而不是 source.html 来浏览 XML 模板。

9. 使用带有内容的模板

通常，Zope 支持内容，表现和逻辑组件。页面模板是一种表现组件，因而可以用来显示内容组件。

Zope 带有几种内容组件：ZSQL Methods, Files, 和 Images。DTML 文档和方法不是纯粹的内容组件，这是因为他们可以容纳内容并且执行 DTML 代码。你可以使用 Files 来处理文本内容，如果小于 64K 并包含文本，你可以编辑文件内容。File 对象是相当基本的，不一定提供所有你所需的特性。

Zope 的内容管理框架(CMF)通过提供一套丰富的内容组件集来解决这个问题。CMF 是一种内容管理扩展件。它包括各种加强功能，包括工作流，界面和内容对象等。CMF 利用了很多页面模板。在以后的版本里，Zope 可能会包含 CMF。

10. 结论

Zope 页面模板帮助你为 web 应用程序构建 web 页面。模板使你轻松使用普通的 HTML 工具和技术来构建 web 页面。它们还提供了融进现有应用程序的便利性。页面模板帮助设计人员和编程人员一起工作来生成 web 应用程序。在“高级页面模板”里，你将学习更强大的模板技术，比如：Python 表达式和宏。

第十章 创建基本应用程序

在本章，将一步一步的讲解如何构建简单的 Web 应用程序。在这个过程中，将涉及到常用的 Zope 核心概念。使用到的对象包括：文件夹、Script(Python)和页面模板。我们将通过使用这几种对象创建一个简单的网站，名称是：Zope 动物园。

我们将以“实例空间”的方式开发这个网站。本章的后边将讲解“实例空间”，但目前，只要知道它是一种最容易最快速的方式就可以了，这是因为全部的例子都可以通过 web 浏览器来完成。

1. Zope 动物园网站的目标

首先我们必须清楚要完成的目标。这个应用程序的主要目标是为 Zope 动物园创建一个 Web 网站，并且网站要容易使用和管理基本的要求是：

要让 Web 用户能够轻松的周游站点，就好像他们正在步行走过一个真实的动物园。

外观要容易调整和管理，比如对于层叠样式表（CSS）。

网站要进行良好的设计和组织，以便于将来进行更改。

要利用 Zope 的动态特性，实现内容的自动更新。

2. 从文件夹开始

Zope 当中的文件夹对象提供了一种很直观的内容组织方式。我们可以从创建文件夹开始。这些文件夹用来存放各种对象。

举个例子，我们可以创建一个名为“Invoices”的文件夹，用来存放订单。这个文件夹中可以包含用于处理订单的逻辑对象或方法，也可以存放订单数据。整个目录就形成了一个简单的应用程序。

下面我们将为动物园网站创建文件夹，用于存放所有相关的对象。

2.1. 第一步：创建文件夹

首先启动 Zope，通过浏览器登录进入管理界面，然后：

进入根目录

通过添加列表创建一个新文件夹

1. 指定 id 为 ZopeZoo

选中“Create public interface”

点击 Add 按钮。

3. 特殊的文件夹对象：index_html

因为我们选择了“Create public interface”选项，因此 Zope 还会在创建文件夹的同时创建一个页面模板对象：index_html。

index_html 是 ZopeZoo 文件夹中默认对象，当 Web 用户访问这个目录时就调用 index_html。这就如同是 Apache 服务器中默认的文件是 index.html 一样。

index_html 是文件夹对象的默认视图。我们可以通过多种方式来查看 ZopeZoo 文件夹：

点击 index_html 对象的“Test”标签

1. 点击文件夹 ZopeZoo 的“View”标签
2. 浏览器中直接输入网址： http://localhost:8080/ZopeZoo/index_html
3. 浏览器中直接输入网址： http://localhost:8080/ZopeZoo/

对象 index_html 可以是页面模板、DTML 方法、Script 对象或者其它任何一种可以通过 URL 可以访问的对象。这个对象应当返回 HTML 或其它数据，比如：XML 或文本等等。从面向对象的角度来说，index_html 不仅是文件夹中的一个对象，同时还是文件夹的一个方法或属性。

4. 设计可导航的动物园

为了实现能够轻松浏览网站的目的，需要在网站的页面中加入导航的功能。换句话说，就是在网站中的每个页面里都要显示一段相似的链接，帮助用户访问站点中的每个部分。

并且还要确保导航链接都是正确的，而不管站点结构如何变化。解决的方法是设计合理的网站结构，然后创建用来在导航链接中显示结构的 Zope 方法。

首先让我们定义站点的结构。假设动物园中有三类动物。通过添加文件夹来组织站点。

4.1. 第二步：创建网站结构

(注意，不要在这一步中添加 index_html 对象，即不用选择“Create public interface”)

1. 进入 ZopeZoo 文件夹，创建三个子文件夹，id 分别为：Reptiles, Mammals and Fish。

在 Mammals 文件夹中添加一个名为 Whales 的文件夹。

在 Reptiles 中创建两个文件夹：Lizards 和 Snakes。

通过管理界面中的导航框架，可以看到文件夹 ZopeZoo 的结构。如果看不到，刷新页面。通过点击“+”号图标可以打开文件夹。如下图所示：

动物园文件夹结构

现在，我们假设，要浏览动物园网站，永辉会从默认的视图（index_html）开始。可以称这个页面为首页。通过首页，用户应该可以选择一个链接，进入下级文件夹，从而访问想看到的动物。

5. 网站主导航栏

首先，让我们创建导航方法，用来显示网站的主导航栏，也就是那些可以访问站点主栏目(Fish, Mammals, and Reptiles)的链接。站点中的每个页面都调用这个方法。并且这个方法会动态创建下级文件夹的链接。

导航的方法采用页面模板（ZPT）。让我们来创建这个模板，并仔细看看 TAL 的使用方式。

5.1. 第三步：创建网站主导航方法

1. 进入 ZopeZoo 文件夹。

从添加列表中选择页面模板

输入 id: nav_main

点击 Add and Edit

编辑代码如下：

http://down.baow.org

此时会发现，导航的作用生效了，但同时发现，index_html 和 nav_main 也列了出来。 我们不想显示所有的对象，只希望显示文件夹对象。因此还需要对代码进行修改。

5.2. 第四步：排除非文件夹对象

返回 nav_main 代码部分，然后找到 span 部分，把代码改成：

- <span tal:repeat="foldy
- python:container.objectValues(['Folder'])" tal:omit-tag="">
-

点击 Save Changes.

然后再进行测试，会发现只显示文件夹对象了。这样就形成了主导航栏。 如果文件夹中没有 index_html，则会通过获取机制来取得上级目录中的 index_html 对象。

6. 获取机制

获取机制可以用来共享属性。比如，我们这个例子中 ZopeZoo 文件夹的 index_html 属性。 关于获取机制，核心的思想就是如果某个对象缺少某个属性，Zope 会自动向上级目录中搜索这个属性，直到找到为止。 在我们这个例子中，如果 animal 目录中不能找到 index_html，Zope 会自动尝试向上级目录中查找。直到搜索到 ZopeZoo 文件夹。 假设 ZopeZoo 中也没有 index_html，那么就继续往上查找，直到根文件夹，如果还没有就会提示错误信息。如果站点中存在多个 index_html，则会采用最近的一个。 比如，对于 Web 请求： http://localhost:8080/ZopeZoo/Mammals/，Zope 会先搜索 Mammals 文件夹中的 index_html，如果找到了就返回这个 index_html。

7. 结合组件

我们将继续创建两个新的 Zope 页面模板。第一个是页眉模板，它用来处理导航，放在每个页面的上部。然后创建一个与正文相关的页面模板，用来显示正文内容。 然后，把这些组件结合进入 index_html，形成基本的网站框架。

7.1. 第五步：创建几个组件

1. 返回到 ZopeZoo 文件夹。

添加两个页面模板，id 分别为 header 和 body_content。

编辑 header 代码如下：

-
-
- MAIN NAVIGATION
-

-
-

点击 Save Changes

符号中通过使用 id 属性，可以使用层叠样式表来更改界面。点击 Test 标签，可以看到以下一些 HTML 代码：

页面的 HTML 代码

很明显，这不是我们想要的结果。Zope 会自动把 HTML 代码转化成可显示的字符。但我们需要的是能够在其它页面中调用并显示的 HTML。让我们继续解决这个问题。

7.2. 第六步：指定 HTML 的构成，而不是显示 HTML 代码

返回到 header 模板的代码部分，在 tal:content 语句中插入 structure 关键字，代码如下：

-
-
- MAIN NAVIGATION
-
-
-

保存更改

然后，进行测试，就会发现结果正常了，显示的内容和直接调用 nav_main 模板差不多。如下图所示：

新的页眉

接下来，让我们来编辑 body_content 模板，需要让 body_content 成为 index_html 的一部分。

7.3. 第七步：准备 body_content 模板

1. 返回到 ZopeZoo 文件夹，点击 body_content 模板。

编辑代码部分，只保留<body>和</body>之间的代码（不包括<body>和</body>）。

保存

接下来，让我们把这些组件组织到 index_html 中，这样就可以创建默认的视图。

7.4. 第八步：组织默认的视图

1. 返回到 ZopeZoo 文件夹，点击 index_html

用以下代码替换原有的代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title tal:content="here/title_or_id">TITLE</title>
</head>
<body>
  <span tal:replace="structure here/header">DUMMY HEADER</span>
```

```
    <span tal:replace="structure here/body_content">BODY</span>

  </body>

  </html>
```

点击 Save Changes

可以看到，这个 index_html 是通过指定两个组件来构成的，即 header 和 body_content。通过在浏览器中输入以下网址，可以进行测试：

```
http://localhost:8080/ZopeZoo/
```

(如果 Zope 没有安装在本机中，需要 localhost 换成网址)可以看到如下结果：

默认的视图

现在可以看到一个比较完整的网站框架，它有导航栏，并且可以动态的创建链接。

点击链接时，通过 http://localhost:8080/ZopeZoo/Reptiles/ ，Web 服务器接收到请求。Zope 返回 Reptiles 文件夹默认的视图，即 index_html。

由于 Reptiles 文件夹中没有 index_html，于是向上搜索 index_html。然后调用 header 方法。header 又调用 nav_main。最后 index_html 又调用 body_content，生成完整的 index_html 页面。

注意，尽管 index_html 位于 ZopeZoo 文件夹中，但是却能够针对 Reptiles 对象返回相应的信息。这是因为 index_html 是针对 Reptiles 对象进行调用的。

8. 针对文件夹调用方法

Zope 中的获取机制与对象调用时所处的环境密切相关，是 Zope 中的一个强大特性。因此通过 URL 加上要调用的方法的 id，就可以对任何对象调用方法。在上面的例子中，还没有充分利用这一特性，我们只是调用了默认的方法 index_html。

比如，对于前边提过的 URL： http://localhost:8080/Invoices/addInvoice ，其中 Invoices 是一个文件夹对象。这个 URL 表示针对 Invoices 调用 addInvoice 方法。

这个功能是 Zope 中一个通用的特性。实际上，除了文件夹对象，还可以对很多对象调用方法。详细方法可参考“高级 Zope 脚本”一章。

另外，还可以通过 URL 传递参数。例如： http://localhost:8080/Invoices/editInvoice?invoice_number=42。这个 URL 把参数 invoice_number=42 传递给 Invoices 文件夹的 editInvoice 方法。通过这种方式就可以编辑序号为 42 的订单。

9. 创建与环境相关的子文件夹导航栏

接下来，我们将加强导航栏。我们将在页眉部分增加一个子文件夹导航部分，放在主栏目导航链接的后边。

在 nav_main 对象中定义的主栏目导航栏保持不变。我们将创建一个导航组件，它会根据用户所处的位置进行变化，显示出所有的子文件夹，从而可以充分显示网站的逻辑结构。

9.1. 第九步：创建子文件夹导航方法

1. 进入 ZopeZoo 文件夹

加一个 id 为 nav_subfolder 的页面模板

点击 Add and Edit

用以下代码替换原来的代码：

```
<ul>
        <li tal:repeat="foldo python:here.objectValues('Folder')">
          <a href="HREF" tal:attributes="href foldo/absolute_url"
             tal:content="foldo/title_or_id">TITLE OR ID</a>
        </li>
      </ul>
```

保存

点击 Test 标签，会看到列出了三个链接。如下所示：

子文件夹导航

其中的代码和以前看到的很相似，不同之处在于 tal:repeat 语句中使用了"here"，而不是"container"。这是关键之处，体现了获取机制。TAL 中的 container 表示的是模板所处的文件夹，而 here 表示的是模板所应用的对象。使用 Test 标签测试这段代码时，结果是一样的，这是因为 container 和 here 都是同一个对象： ZopeZoo 文件夹。 要看到 here 所起的作用，我们需要针对另外一个对象调用这个模板。比如，打开一个新的浏览器窗口，然后访问以下 URL：

http://localhost:8080/ZopeZoo/Reptiles/nav_subfolder

你会看到文件夹 Reptiles 中的子文件夹列表，如下图所示：

对 Reptiles 调用 nav_subfolder

同样，如果我们针对 Mammals 文件夹调用 nav_subfolder，就会显示其子文件夹的链接。这个方法可以针对任何对象进行调用（只有文件夹能够显示结果）。由于 nav_subfolder 位于站点的 ZopeZoo 文件夹中，因此所有子文件夹都可以调用它。

10. 给 Header 添加新元素

为了能够在站点中的每个页面中都可以调用 nav_subfolder，需要在 header 中调用它。当然也可以直接在 index_html 中调用它，但是放在 header 中会更好些。

当站点中有多个 index_html 时，这种方式特别有用，这是因为 header 是一样的，修改了 header，站点中所有页面都会自动变化。当然，你完全可以根据需要进行调整。

10.1. 第十步：结合子文件夹导航栏

1. 进入 ZopeZoo 文件夹，点击 header 页面模板。

修改代码如下：

```
<div id="header">
  <div id="nav-main" tal:content="structure here/nav_main">
    MAIN NAVIGATION
  </div>
  <div id="nav-subfolder" tal:content="structure here/nav_subfolder">
    SUBFOLDER NAVIGATION
  </div>
</div>
```

保存

在浏览器中输入 http://localhost:8080/ZopeZoo/ ，进行测试，就会看到新出现的子文件夹导航栏。

11. 创建子文件夹的默认视图

我们可以看到，文件夹对象可以从上级文件夹中获取 index_html。目前为止，调用的都是 ZopeZoo 文件夹中的 index_html。如果某些文件夹需要调用不同的默认视图，只需要在这些文件夹中创建新的 index_html 就可以了。比如，可以在 Reptiles 文件夹中创建一个新的 index_html。新的默认视图对所有的子文件夹（Snakes 和 Lizards）都会生效。 我们继续使用 ZopeZoo 文件夹中的 index_html，不创建新的 index_html。我们通过使用 body_content 来为每个文件夹创建不同的内容。 通过这种方式可以实现内容与界面部分的分离。我们只需要在每个文件夹中添加一个 body_content 就可以了。

11.1. 第十一步：定制默认视图

通过管理界面，进入 Reptiles 文件夹。

添加一个 id 为 body_content 的页面模板

输入以下代码：

```
<h1>The Reptile House</h1>
<p>We are open from 6pm to midnight Monday through Friday.</p>
```

```
<h2>Interesting reptile fact:</h2>

<p>The shape of a reptile's pupil indicates whether the animal

   is active at night or during the day. Most reptiles active at

   night have slit-like pupils that can be closed almost

   completely in bright light. Reptiles active in daytime have

   round pupils.

</p>

<p>For more interesting facts, visit

<a href="http://www2.worldbook.com/features/reptiles/html/facts.html">

this World Book site</a>.

</p>
```

保存

如果点击 Test 标签，就会看到以上的页面。要完整的看结果，可以输入网址：http://localhost:8080/ZopeZoo/Reptiles/ ，就会看到完整的结果。 通过这样一种方式，就可以实现调用同一个 index_html，但显示的是对应文件夹中的数据内容。 如果点击 Snakes 链接，显示的页面内容是和 Reptiles 一样的。这是因为 Snakes 位于 Reptiles 文件夹中，使用的是 Reptiles 中的 body_content。 另外，还会发现没有了子文件夹导航栏。这是因为 Snakes 目录中没有子文件夹。

12. 加强网站的功能

现在，我们的导航系统已经可以工作了，但是它有一个问题。用户要是进入站点栏目，比如从 Mammals 进入 Whales，则需要使用浏览器的“返回”按钮来返回到上一级。我们的导航栏中还没有提供返回到上级的链接。

接下来，让我们给 nav_subfolder 添加一个“返回到上级”的链接。

12.1. 第十二步：添加一个“Return to Parent”链接

1. 进入 ZopeZoo 文件夹的 nav_subfolder 模板

在代码的开始部分加入新的一行后，如下：

```
<p><a href="..">Return to parent</a></p>
```

```
<ul>
  <li tal:repeat="foldo python:here.objectValues('Folder')">
    <a href="HREF" tal:attributes="href foldo/absolute_url"
       tal:content="foldo/title_or_id">TITLE OR ID</a>
  </li>
</ul>
```

保存

这样编辑完以后，当再次访问站点时，就会很方便的返回上级文件夹。 但是，又发现另外一个问题，就是当用户位于站点顶级文件夹（就是 ZopeZoo ）时，仍然存在返回上级的链接。这个链接是不需要的。我们可以通过添加一个判断语句来弥补这个缺点。

12.2. 第十三步：修改“Parent Link”模板代码

在 Zope 管理界面中，进入到 nav_subfolder 页面模板

修改代码如下：

```
<p tal:condition="python:here.id != 'ZopeZoo'">
  <a href="..">Return to parent</a></p>
<ul>
  <li tal:repeat="foldo python:here.objectValues('Folder')">
    <a href="HREF" tal:attributes="href foldo/absolute_url"
       tal:content="foldo/title_or_id">TITLE OR ID</a>
  </li>
</ul>
```

保存

以上代码中是通过在 TAL 中提供一个 python 判断来实现的。如果 here 对象的 id 不是" ZopeZoo "，则 Python 表达式的值为真，则显示链接。如果值为假，则忽略显示链接。 在这个例子中，如过 here.id 等于" ZopeZoo "，Python 表达式的值为假，那么就会把链接去掉。

13. 我在哪里？

在我们的例子中，还存在一个不方便的地方，就是当在 Reptiles 中添加 body_content 以后，失去了一些信息。比如当用户浏览 Lizards Snakes 文件夹时，很难判断出所处的位置。

位于 ZopeZoo 文件夹中的 body_content 方法包含了显示用户当前位置的代码，即用 here/title_or_id 显示当前对象的标题或 id。 接下来，我们把这个功能放入到 header 对象中，这样就可以让所有页面都具有这个功能。

13.1. 第十四步：添加“You Are Here”元素

1. 进入 ZopeZoo 文件夹中的 header 页面模板

添加<h2>元素，代码修改成：

```
<div id="header">
  <div id="nav-main" tal:content="structure here/nav_main">
    MAIN NAVIGATION
  </div>
  <div id="nav-subfolder" tal:content="structure
    here/nav_subfolder">
    SUBFOLDER NAVIGATION
  </div>
<h2>You are in the
  <span tal:replace="here/title_or_id">TITLE_ID</span>
  section.
</h2>
</div>
```

保存

现在我们的站点就可以显示所处位置信息了。

14. 提炼样式表

Zope 中鼓励把可以重复使用的部分提炼出来。前面已经看到，我们已经把导航部分提炼了出来，形成了 header。通过这样一种方式就可以实现共享。 在 HTML 中，可以通过层叠样式表(CSS)规定网页外观，比如颜色、字体等等。CSS 可以很方便的调整页面的外观。 所以，提炼出样式表信息，就可以帮助我们方便的管理网站的外观。通过在不同的文件夹中编排 CSS 文件，就可以对网站中不同的部分实现不同的外观。 CSS 文件是一种文本文件。下面，通过 Zope 文件对象给我们的网站添加样式表。

14.1. 第十五步：创建网站样式表

1. 进入 ZopeZoo 文件夹。

添加一个 id 为 style_sheet 的 File 对象。

点击 style_sheet，进入编辑状态。

修改文件类型为：text/css

编辑代码如下：

```
body {
  font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
  background-color: #F0FFF0;
}
h2 {
  color: Green;
}
#header {
  padding: 1em;
  background-color: #90EE90;
  font-size: smaller;
```

```
}

#header h2 {

  font-style: italic;

  font-size: 12pt;

}

#nav-main {

  margin-bottom: 1em;

}

#nav-subfolder {

  font-weight: bold;

}
```

保存更改

样式表规定了如何显示<body>和<h2>元素，以及模板中的 部分。现在需要让 web 页面引用这个样式表。HTML 中，在<head>部分调用样式表。

14.2. 第十六步：调用样式表

1. 进入 ZopeZoo 文件夹，点击 index_html。

在<head>部分插入样式表，即修改代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

  <title tal:content="here/title_or_id">TITLE</title>

  <style type="text/css">

   <!--
```

```
        @import url(style_sheet);
        -->
      </style>
    </head>
    <body>
      <span tal:replace="structure here/header">DUMMY HEADER</span>
      <span tal:replace="structure here/body_content">BODY</span>
    </body>
    </html>
```

保存修改

点击 index_html 的 Test 标签进行测试，就会看到网站外观变得好看多了。如下图所示：

美化后的动物园

如果要更改网站中其它部分的外观，可以利用获取机制，在文件夹中增加一个 style_sheet 就可以了。这个文件夹和子文件夹中都会调用这个 style_sheet。

15. 在 Zope“实例空间”中构建应用程序

要开发一个 Web 应用程序，在 Zope 中有多种方式。最简单快速的方式就是我们在上面采用的方式，即在实例空间中构建。要理解“实例空间”这个概念，需要再次提及面向对象的概念。 实例空间相当于 Zope 的根文件夹。使用 web 浏览器，我们就可以在

实例空间中通过管理界面来查看和控制对象。实例空间中所有的对象组件都是 Zope 类的实例。我们通过添加各种实例对象，就可以简单的构建应用程序。 如下图所示，显示了一个管理界面，并且选择了添加列表。

Zope 管理界面中的添加列表

添加列表中的每个对象都表示了 Zope 中已经定义好的类。当我们选择了一个类，就会提示我们输入创建类的实例所需的信息然后，Zope 就会使用输入的信息在当前的位置创建新的实例。这个实例直到被删除才会消失。 比如，文件夹（Folder）类就是由 Zope 公司的工程师编写的。所有在管理界面中显示的文件夹对象都是文件夹类的实例。由于每个对象都是某种类的实例，因此我们使用“实例空间”这个概念来表示 ZODB 中所有的对象。 要在实例空间中构建应用程序，就需要通过管理界面创建 Zope 类的实例。我们通过调整实例，就可以让它们完成特定的功能，然后把它们结合到一起，就创建了 Zope 应用程序。

15.1. 实例空间应用程序与产品

除了通过在实例空间中构建 Zope 应用程序，还可以通过开发“产品”来构建。通过产品的方式，可以在添加列表中显示出来，从而可以通过新对象来扩展 Zope。另外，产品应用程序容易分发给其他人，并且可以创建更为实用的对象。 在“扩展 Zope”一章中，集中讲解了如何创建产品。

第十一章 用户和安全

1. 简介

Zope 是一个多用户的系统。Zope 不通过操作系统中的用户进行管理，而是通过自己提供的系统管理用户。 Zope 中的用户不同于操作系统中的用户，因此不能直接对操作系统中的文件进行操作。Zope 用户可以根据不同的权限对内容进行调用和操作。Zope 中的用户根据安全策略拥有不同的权限。只有 Zope 管理员才可以更改安全策略。 并且，通过安全策略可以给网站中的不同部分分配权限。安全授权是 Zope 的一个重要特性。通过授权可以根据需要把部分权限分配给特定用户，而不会影响系统的安全。在本章，我们详细讲解管理员用户、角色、角色权限分配、创建安全策略。

2. 最常用的安全策略

Zope 中最常用的两种用户是：管理员和匿名用户。在以前的章节中，已经讲过了使用初始帐号，即“admin”可以登录进入管理界面。用户“admin”是一个具有管理员角色的用户，管理员可以进行几乎所有的操作。

初始默认的情况下，管理员可以登录进入管理界面并修改对象，而匿名用户只能观看运行后的内容。这种方式对于许多简单的网站已经够用了，特别是那些不需要登录功能或编辑内容的网站。

3. 验证和授权

当一个用户访问被保护的资源时，比如试图访问一个被保护的 DTML 方法，Zope 会弹出一个对话框，要求你输入用户信息。只有填写并提交以后，Zope 才会根据用户的权限提供不同的内容。

Zope 通过输入的用户名和密码来识别用户。如果在用户数据库中找到了对应的用户，才会得到认可。 只有通过验证以后的用户，才不会再次弹出对话框。

对于匿名用户，Zope 不进行验证。如果不访问受保护的资源，Zope 会默认用户为匿名用户。 安全策略决定了访问资源时是否进行用户验证。比如，如果用户试图访问一个方法，如果这个方法已经受到安全策略的保护，就会弹出验证对话框。我们已经看到了，如果要进入 Zope 管理界面，就需要进行用户验证。

如果没有通过验证，Zope 会再次弹出对话框。如果通过了验证，就会根据用户的权限显示相应的内容。 通常，如果只访问公开的资源，则不需要用户登录。

4. 授权，用户和许可

一旦用户通过了验证，Zope 会判断用户是否具有访问这个资源的权限。这个过程称之为授权。 授权的过程涉及用户和被保护资源之间的两个媒介层：角色和许可。

用户拥有角色，角色描述了用户能够做什么，例如作者、管理员和编辑员。这些角色由 Zope 中的系统管理员控制。用户可以有多个角色，也可以在不同的环境中拥有不同的角色。Zope 对象拥有许可，许可描述了对这些对象都能够做什么，例如观看、删除对象和管理属性。这些许可通过 Zope 本身或产品设定，每个对象都可以设定自己的许可。

Zope 中的每个可以通过 Web 界面访问的对象都可以关联安全策略。并且，安全策略支持获取，也就是说，在一个文件夹中定义了安全策略，这个文件夹中的对象默认采用同一安全策略。这样就可以轻松的维护安全策略。 安全策略把角色映射到许可。换句话说，安全策略指明“谁”在“哪里”能够做“什么”。比如，一个文件夹的安全策略可以把“删除对象”的权限关联给“管理员”。这样就可以让管理员能够在这个文件夹中删除对象。如果这个对象没有覆盖上级的安全策略，就自动获取。

5. 管理用户

在安装 Zope 的时候，会生成一个初始帐号，用户名是 admin，这是一个管理员帐号，可以管理 Zope 中的对象。如果要让别人也可以登录进入 Zope，则需要创建新的用户。

5.1. 在用户文件夹中创建用户

Zope 用户对象定义用户帐号。一个用户有的属性包括用户名、密码、一个或多个角色等等。通过角色可以方便的定义用户的权限范围。

在用户文件夹中可以创建用户。用户文件夹中包含用户帐号。Zope 中的不同地方可以有多个文件夹，同一文件夹中只能有一个用户文件夹。

下面举例说明。要创建新的用户帐号，进入根文件夹。点击对象 acl_users。再点击 Add 按钮，创建一个新用户，如下图所示

在用户文件夹中添加一个用户

以上表单定义了用户。在 Name 字段中输入一个用户名，比如“bob”。用户名可以包含字母、空格和数字。用户名区分大小写。然后在“Password”中输入密码，并确认密码。 “Domains”字段可以限制能够登录的域。比如，输入“myjob.com”，那么只有属于这个域中的用户可以登录。这个字段中可以输入多个域，用空格分开即可。比如：“myjob.com myhome.net”。另外，也可以设置 IP 地址，比如“209.67.167.*”，即所有来自以“209.67.167”开始的 IP 地址，并且用户名和密码都匹配的登录才能够通过验证。

角色选择中可以选择多个角色。默认的角色包括 Manager 和 Owner。通常，能够登陆进入到管理界面的用户需要使用 Manager 角色。用户对于自己创建的对象一般都有 Owner 角色。Owner 不是很常用。角色可以进行定制，比如增加“编辑者”和“审核者”角色。 要创建用户，点击 Add 按钮即可。

Zope 中自带的用户帐号不支持更多的属性，比如“电子邮件地址”和“联系电话”等等。要支持这些功能，需要单独安装用户产品，比如 CMF 成员组件或 exUserFolder。

用户对象不支持复制与粘贴。

5.2. 编辑用户

通过单击管理界面中的用户对象，可以编辑现有的用户属性，但不能更改用户名。如果要更改用户名，可以先把这个用户删除，然后再添加新的用户。在管理界面中不能显示用户的密码信息。

和其它对象一样，安全策略控制一个用户是否具有编辑其它用户的权限。只有拥有管理员角色的用户才能够编辑用户信息。很多情况下，我们希望用户能够只能修改自己的信息，但不能修改其它人的信息，这时需要使用“Proxy Roles”，在本章的后面将会讲到解决的方法。

用户文件夹就像其它文件夹一样，可以创建、编辑和删除对象。然而对于用户文件夹，不能剪切和粘贴用户，除了用户对象不能再创建其它对象。 要删除现有的用户，可以选中以后，点击 Delete 按钮。

6. 定义用户位置

在 Zope 中可以定义多个用户文件夹。Zope 用户不能登录进入定义这个用户的文件夹以上的上级文件夹。用户帐号在 Zope 中所在的位置决定了用户的权限范围。

如果一个管理员帐号定义在根文件夹中，那么这个用户可以管理根文件夹中的内容。在任何文件夹中都可以创建用户文件夹如果一个管理员户帐号在一个子文件夹中定义，这个用户则只能管理这个子文件夹中的内容。

假设一个用户文件夹位于 /BeautySchool /Hair/acl_users ，其中有一个用户 Ralph Scissorhands。Ralph 不能进入 /BeautySchool /Hair 以上的文件夹。如果 Ralph 具有管理员角色，他就可以通过 http://zopeserver/BeautySchool/Hair/manage 管理 Hair 目录中的各种对象。

通过这种方式，可以把网站的管理员权限分配给其他用户。分配权限的时候只要在相应的目录中创建用户文件夹，然后添加用户就可以了。这样就可以进行分布式管理。

7. 使用其它用户文件夹

如果你觉得 Zope 中自带的用户文件夹不能满足你的要求，那么可以使用其它的替代品。这些替代品可以在 Zope.org 的产品区中找到。以下是常用的几种：

Extensible User Folder（可扩展用户文件夹）

可扩展用户文件夹可以使用其它的数据源，用户的数据可以存储在 Postgresql、RADIUS 和 SMB 等等这样类似于 ZODB 的数据库中。

etcUserFolder

这种用户文件夹使用标准的 Unix 系统中的 /etc/password 文件来完成验证。

LDAP User Folder

这种用户文件夹可以通过 LDAP 服务器进行用户验证。

NT UserFolder

通过使用NT中的用户帐号进行验证，它只有当使用Windows NT或Windows 2000 操作系统时才有效。

MySQL UserFolder

通过使用MySQL数据库中的数据进行验证。

一些用户文件夹在验证的时候使用Web页面的形式进行登录，而不是弹出一个对话框。所起到的作用是一样的。 大多数的用户是通过用户文件夹进行管理的，但是有一些特殊帐号不是通过用户文件夹进行管理的。

8. 特殊的用户帐号

Zope中有三个特殊的帐号不通过用户文件夹进行定义：匿名用户、紧急用户和初始管理员帐号。匿名用户使用最频繁，紧急用户和初始管理员帐号使用比较少。

8.1. 匿名用户

匿名用户是一种不需要验证的用户帐号。如果在Zope中没有设置用户，那么默认为匿名用户。匿名用户的角色就是匿名角色Zope的安全策略中，对于公开的资源，默认使用匿名角色，在大多数情况下是足够的。

8.2. 紧急用户

紧急用户是一种特殊的帐号，它不受安全设置的限制。紧急用户除了能够创建其它的用户帐号外，不能创建任何对象。 紧急用户适用于两种情况：修正用户系统和创建用户帐号。你可以通过紧急用户帐号创建或修改其它的用户帐号。一般常见的用途是定义管理员帐号或者更改已经具有管理员角色的帐号的密码。当你丢失了管理员用户或密码的时候，就可以使用紧急帐号。通过使用紧急帐号登录，添加新的管理员帐号，然后就可以使用新的管理员帐号进行管理。 使用紧急用户的另外一种情况是用来修正用户系统。如果你无法登录进入Zope，则可以用紧急帐号进行恢复。 紧急帐号不能创建新的内容、逻辑或表现对象。

8.2.1. 创建一个紧急用户

创建紧急用户帐号是通过调用文件系统中的一个文件来完成的。在Zope的安装目录中有一个名为zpasswd.py的文件，比如，在Unix系统中可以这样进行调用：

```
$ cd (... where your ZOPE_HOME is... )

$ python zpasswd.py access

Username: superuser

Password:

Verify password:
```

Please choose a format from:

SHA - SHA-1 hashed password

CRYPT - UNIX-style crypt password

CLEARTEXT - no protection.

Encoding: SHA

Domain restrictions:

在Windows系统中，可以这样进行调用：

> cd (... where your ZOPE_HOME is ...)

> cd bin

> python ..\zpasswd.py ..\access

运行这个文件会一步一步的引导你创建一个紧急用户帐号。创建完成后，需要重新启动Zope。

8.3. 初始管理员

实际上，在安装Zope的时候，就会自动创建一个初始管理员帐号，在Unix中安装的时候，会出现以下一些信息：

creating default inituser file

Note:

The initial user name and password are 'admin'

and 'IVX3kAwU'.

You can change the name and password through the web

interface or using the 'zpasswd.py' script.

上面显示的信息中就包含了一个初始管理员帐号。

创建初始管理员帐号的方式和创建紧急用户的方式是类似的，还是通过zpasswd.py文件进行创建。比如：

$ cd (... were your ZOPE_HOME is ...)

```
$ python zpasswd.py inituser

Username: bob

Password:

Verify password:

Please choose a format from:

SHA - SHA-1 hashed password

CRYPT - UNIX-style crypt password

CLEARTEXT - no protection.

Encoding: SHA

Domain restrictions:
```

初始管理员帐号位于根文件夹中，

9. 保护密码

普通的登录验证过程没有对密码进行加密处理。如果你担心用户名和密码的安全问题，或者希望以一种更为安全的方式管理站点，那么可以通过SSL(安全套接字层)连接来实现。最简单的实现方式是使用Apache和支持SSL的浏览器。在Zope.org中可以找到相关的文章。

10. 定制安全策略

通过安全策略可以控制用户的权限。“角色”用于标注用户分类，而“许可”用于保护对象。安全策略规定了具有某种角色的用户对于某种对象可以进行那些操作。 对于站点中的某一部分，Zope中可以针对不同的用户设定不同的操作权限。通过这样一种方式，使得安全策略简单而强大。当然，也可以针对特定的对象设定权限。

10.1. 处理角色

Zope用户拥有多种角色，这些角色定义用户能够执行什么类型的行为。角色定义用户分类，例如管理员、匿名用户、已验证用户。 角色类似于UNIX中组的概念。就像UNIX组，Zope用户能够拥有多种角色。 角色使得管理安全更为容易。你可以为不同的用户角色设置几个不同的安全策略，而不是定义每个单独的用户可以做什么。 Zope内置有四种角色：

管理员（Manager）

这个用户适用于执行各种管理功能，比如创建和编辑文件夹和文档等等。

匿名（Anonymous）

匿名用户拥有匿名角色。这个角色可以查看公开的资源。通常，拥有这个角色的用户不能修改对象。

所有者（Owner）

这个角色自动分配给创建对象的用户。本章后边将详细讲述。

已验证用户（Authenticated）

这个角色自动分配给已经通过验证的用户。通过这个角色可以获知用户是否已经通过验证。当用户登录以后，就属于已验证用户。

对于简单的站点，一般只用管理员和匿名角色就够了。对于更为复杂的站点可以创建新的角色，从而根据角色对用户进行分类。

10.2. 定义全局角色

一个“全局”角色可以显示在对象的 Security 视图中。要创建全局角色，可点击对象的 Security 标签，向下滚动屏幕。在 User defined role 字段中输入新角色的名称，然后点击 Add Role 按钮。角色名称应该简短，使用一个或两个单词描述一种类型用户，例如 Author（作者）、SiteArchitect （站点建筑师）或者 Designer（设计师）。应该选择与应用程序有关系的角色名称。

添加完新角色后，会发现新增加了一列。也可以删除新增加的角色。

应该注意的是在对象层次中，新定义的角色的有效范围是从定义这个角色的地方开始的。比如，如果在 Examples 文件夹中定义了一个角色，那么这个角色只能在这个文件夹中使用。如果要让一个角色适用于所有站点，那么应该在根文件夹中创建。

通常，角色应该是针对站点中的大范围来应用的。如果你发现自己正在创建用于限制访问站点中的一小部分的角色，就有可能采用一种更好的方法来完成相同的任务。例如，对于你需要保护的文件夹的角色，你只需简单的更改安全设置就可以了。或者你可以在对象更深层次中定义用户，用以限制他们的访问。

10.3. 理解本地角色

本地角色是一种高级 Zope 安全特性。通过使用本地角色，可以在处理某一对象时，给特定的用户赋予多个角色。如果一个对象拥有与某个用户相关联的本地角色，当这个用户处理那个对象时就得到那些附加角色。

例如，一个用户通过 Zope 管理界面创建了一个对象，那么这个对象对于这个用户就赋予了 Owner 角色。通常，如果用户对于某个对象没有 Owner 角色，并且没有通过全局角色设定权限，那么就不能编辑这个对象。如果用户对这个对象有 Owner 角色，那么就可以对这个对象进行编辑。

本地角色是一种不是很常用的安全控制方法。通过这种方法可以对特定的对象指定用户。

10.4. 理解许可

对于某种对象，许可定义了一个可执行的操作。比如许多 Zope 对象，包括 DTML 方法和 DTML 文档，允许被观看。这个操作受到 View 许可的保护。Zope 产品的开发人员在开发的时候定义对象的各种许可。

一些许可只与某一种类型的对象有关。例如，Change DTML Methods（更改 DTML 方法）许可只保护 DTML 方法。其它许可保护不同种类的对象，例如 FTP 访问许可或 WebDAV 访问许可，这些许可控制用户是否能够通过 FTP 或 WebDAV 访问对象。 通过进入对象的安全管理视图，可以看到对象所有的许可。

邮件主机对象的安全设置

上图中列出了邮件主机对象的许可。

10.5. 定义安全策略

安全策略中包含角色和许可。安全策略定义某人在站点给定部分能够做什么。 几乎任何对象都可以设置安全策略。通过对象的 Security 标签可以设置安全策略。比如，点击根文件夹的 Security 标签，如下图所示：

根文件夹的安全策略

在上图中，显示了一列复选框。竖列表示角色，横行表示许可。选中一个复选框，就表示了属于一个角色的用户能够对这个对象执行选中的许可。 比如，你希望不让匿名用户访问站点，那么你可以通过调整根文件夹中的安全策略来实现。另外，还可以通过定制子文件夹的安全策略来限制子文件夹的访问。

10.6. 安全策略获取

不同的安全策略如何互相影响？我们已经看到了你能够对不同对象创建安全策略，但是，是什么决定了哪一种策略控制哪一种对象呢？回答是：如果对象有自己的安全策略就使用这个安全策略。另外，还通过获取机制（acquisition）继承上级的安全策略。通过获取机制可以在不同的文件夹之间共享信息。安全策略也可以从上级文件夹中获取。在 Security 视图中可以控制安全策略的获取机制。其中，左边的复选框用来设置获取机制。对于根文件夹，由于没有上级对象，因此不会显示出左边的复选框。

举个例子来说，假设你想让这个文件夹私有。就像我们以前看到的，这仅仅需要拒绝匿名角色的观看许可（View permission）。但是，就像你在这个屏幕中看到的，匿名角色没有 View permission，这个文件夹却不是私有的。这是为什么？答案是 View permission 的 Acquire permission settings 设置选项被选中了。这就意味着当前的设置被这个文件夹的双亲的安全策略扩展了。比这个文件夹更高的某个位置，匿名角色一定被赋予了 View permission。通过检查文件夹的双亲的安全策略就能够验证这一点。要使文件夹私有，我们必须不选中 Acquire permission settings 选项。这样就确保只有那些在这个安全策略中明确设定的设置才是有效的。

通常，你应该尽量获取安全设置，除非你有一个不这样做的明确原因。这样做使你管理安全设置变得更为容易，这是因为大量的工作可以在根文件夹完成。

11. 安全使用模式

Zope 的安全概念可以简单的描述为：通过角色和许可的互相结合创建出安全策略。用户按照角色进行组织。用户可以执行的操作受到所属角色的限制。这些简单工具能够以多种方式结合在一起。下面让我们看一些安全管理模式，这对于如何创建有效而易于管理的安全体系提供了好例子。

11.1. 安全原则概要

以下是几个原则，用于指导安全管理。以下的安全模式提供了比较明确的原则，当你面对未知的情况时，这些原则会给你一些帮助。

在用户的最高控制层次定义用户，除非没有更高层次。

应该把由同一人管理的对象放在一起。

保持简单

原则一和原则二是紧密联系的，都是很通用的原则。通常应该尽量把相关联的资源和用户放在一起。不要以一种生硬的方式组织资源和用户。编排成文件夹和下级文件夹的形式会明显的起到帮助作用。 除了注意站点结构外，尽量让事情保持简单。安全设置复杂，就越是难以理解、管理和保持有效。例如，限制创建新角色的数量，设法使用安全策略获取来代替明确定义安全设置。如果你发现安全策略、用户和角色正在变成一堆乱草，你就应该再想想你正在做什么，也许有更简单的方法。

11.2. 全局和本地策略

进行安全管理的时候最普通的一种方式是在根目录中定义全局安全策略，然后通过获取机制来使用。可以在其它文件夹中添加新的策略来替代全局策略。但是应该尽量降低替换全局策略的次数。如果发现自己在多个不同的地方设置相同的安全策略，那么最好把这些分离的资源放在一起，然后建立一个安全策略，集中进行管理。这样会更方便些。 通常，对于下级文件夹应该采用获取的方式来设置安全策略。如果需要设置不同的安全策略，则需要在“安全”设置中不选择“获取”这个选项。 这是一种很常用的安全管理方式。

11.3. 把控制权委派给本地管理员

这是 Zope 中一种很重要的模式。Zope 鼓励你把各种资源聚集到文件夹里，然后在这些文件夹里通过创建用户帐号来管理。

比方说，你想把 Sales（销售）文件夹的管理委派给新的销售 Web 管理员——Steve。除了 Sales 文件夹以外，你不想让 Steve 弄乱其它的任何内容，因此你不需要把他添加到根文件夹中的 acl_users 文件夹里，而是代之以在 Sales 文件夹中创建一个新的用户文件夹。

现在，你可以把 Steve 添加到 Sales 中的用户文件夹里，并且给他赋予管理员角色。现在，Steve 通过浏览器访问 http://www.zopezoo.org/Sales/manage，就可以直接登录到 Sales 文件夹来管理他的区域。

管理 Sales 文件夹

你会发现在上图中，靠左边的导航树显示 Sales 为根文件夹。在这个文件夹中被定义的本地管理员永远没有能力登录到比 Sales 更高层次的任何文件夹，因此 Sales 被显示为顶级文件夹。

这种模式是非常强大的，这是因为它可以被重复应用。例如，Steve 可以创建一个下级文件夹。然后，在这个文件夹中创建一个用户文件夹，这样就能够把控制权委派给其他的市场销售管理员，以此类推。对于由上千人管理的大型 Web 站点来说，这是一种解决方法。这种模式的美妙之处在于高层管理员不需要过分关注他们的下属的所作所为。如果他们愿意，管理员可以查看，但是他们安全可以不用担心，这是因为他们的授权可以保证其他下级用户在不能在其它区域做任何更改，并且下属的安全设置是通过获取的方式得到的。

11.4. 通过角色访问不同的层次

本地管理员模式是强大且易于扩展的，但是它采用了一种相当粗略的管理方式。要么你有权访问，要么没权访问。有时你需要更为精细的控制。在许多情况下，资源需要被不同类型的人来使用。角色提供了一种解决这种问题的方法。角色允许你定义用户类型，并且设置相应的安全策略。

在创建新的角色之前，需要先看看是否真的需要它们。假设你有一个出版文章的 Web 站点。公众能够阅读文章，管理员可以编辑和公布文章，但却可能存在第三种类型的用户，他们可以撰写文章，但是不能公布或编辑文章。

一种解决方法是创建一个作者文件夹，在其中创建作者帐号并赋予管理员角色。这个文件夹应该为私有的，这样它就只能由管理员来查看。文章可以在这个文件夹中编写，然后管理员通过把文章移出这个文件夹来公布它们。这是一种可行的解决方法，但是它要求作者只在站点的某一部分里工作，并且还需要管理员把文章移出编写文件夹。还有，当一个作者想更新一篇已经移出编写文件夹的文章时也会出现问题。

更好的解决方法是添加一个 Author（作者）角色。添加角色会帮助我们，这是因为它允许使用那些不是基于位置的访问控制因此，在我们的例子中，通过添加一个作者角色，使得我们可以在站点任何地方编写、编辑和公布文章。我们可以设置一个全局安全策略，这个策略赋予作者以创建和编写文章的能力，但是不给他们公布或编辑文章的许可。

角色允许你控制权限，它基于用户是谁，而不是仅基于用户被定义的位置。

11.5. 使用角色控制访问权限

角色能够帮助你克服采用本地管理员模式所带来的难题。这个难题是本地管理员模式需要一种严格的控制层级，它在一组用户不是另外一组用户的管理员情况下不允许这两个不同用户组的人们访问相同的资源。换句话说，在站点某个部分中定义的用户不能管理站点另外一部分中的资源。

让我们通过一个例子来说明这个问题。假设你运行着一个医药公司的大型站点。你有两种类型的用户——科学家和销售人员。通常，科学家和销售人员管理不同的 Web 资源。然而，假设存在一些两种类型的人们都需要管理的事情，例如包含复杂科学警示的广告。如果我们在 Science 文件夹中定义科学家，并且在 Sales 文件夹中定义销售人员，我们应该在哪里放置 AdsWithComplexWarnings （带有复杂警示的广告）文件夹呢？除非 Science 文件夹在 Sales 文件夹中，或者相反，这两种方式都没有可以让科学家和销售人员共同管理的 AdsWithComplexWarnings 文件夹的位置。怎么办？

解决的方法是使用角色。在比 Science 和 Sales 文件夹更高的层级上可以创建两个角色，比方说，Scientist 和 SalesPerson 。然后，不是在它们自己的文件夹中定义科学家和销售人员，而是在对象层级的更高层级中定义它们，这样他们就有权访问 AdsWithComplexWarnings 文件夹。

当你在这个更高层级创建用户时，你不用赋予他们管理员角色，代之以相应的 Scientist 或 SalesPerson 。然后你应该设置安全策略。在 Science 文件夹中，Scientist 角色应该等同于管理员角色。在 Sales 文件夹中， SalesPerson 角色应该像管理员一样具有相同的许可。最终，在 AdsWithComplexWarnings 文件夹中，你应当给予 Scientist 和 SalesPerson 角色适当的许可按照这种方式，角色不是按照权限层级来控制访问权限，而是基于你是谁来控制访问权限。

可能使用这种模式的另外一个常见情形是当你不能定义本地管理员的时候。例如，你也许正在使用一种其它类型的用户文件夹，它要求在根文件夹里定义所有的用户。在这种情况下，就需要使用基于角色的方式来控制访问权限。 以上是安全模式的讨论至此，你应该对如何使用用户文件夹、角色和安全策略来为你的应用程序制定安全架构有了全面了解。接下来，我们讲述两个高级安全专题：如何执行安全检查和保护可执行内容。

12. 执行安全检查

大多数情况下，你不需要执行任何“手工”式的安全检查。如果一个用户试图执行一个被保护的操作，Zope 会提示他们登录如果用户没有足够的许可来访问受保护的资源，Zope 拒绝他们访问。然而，有些时候你希望手工执行安全检查。这样做的主要原因是要限制给已授权用户提供过多的选择。这样做虽然不会防止卑鄙的用户试图访问受保护的操作，但是它通过不给用户提供无效选项，大大减少用户受挫的机会。

最普通的安全询问是当前的用户是否拥有给定的许可。我们通过使用 Zope API 中的 checkPermission 来完成这个功能。例如假设你的应用程序允许一些用户上载文件。这种操作会受到“Add Documents, Images, and Files”许可的保护。你可以在 DTML 中测试当前用户是否拥有这些许可：

```
<dtml-if expr="_.SecurityCheckPermission(

             'Add Documents, Images, and Files', this())">

  <form action="upload">
```

```
    ...

  </form>

</dtml-if>
```

SecurityCheckPermission 函数带有两个参数，一个许可名称和一个对象。在这个例子中，我们通过传递 this()引用当前对象。

对于页面模板，语句略有不同，但效果一样：

```
<form action="upload"

  tal:condition="python: modules['AccessControl'].getSecurityManager().checkPermission('Add Documents, Images, and Files', here)">

...

</form>
```

也可以通过一段 Python 脚本来完成这个任务。下面的例子中，我们使用 check_security 脚本来完成这个任务，然后在页面模板中进行调用，以下是页面模板代码：

```
<form action="upload"

      tal:condition="python: here.check_security('Add Documents, Images and Files', here)">
```

以下是页面模板中调用的脚本代码：

```
## Script (Python) "check_security"

##bind container=container

##bind context=context

##bind namespace=

##bind script=script

##bind subpath=traverse_subpath

##parameters=permission, object

##title=Checks security on behalf of a caller
```

```
from AccessControl import getSecurityManager

sec_mgr = getSecurityManager()

return sec_mgr.checkPermission(permission, object)
```

可以看到，可以通过多种对象来完成许可检查。Zope API 中还有许多其它的用于安全检查的函数，checkPermission 是很常用的一个。 通过把当前对象传递给 checkPermission，我们就可以在测试当前用户是否拥有给定的许可时确信调用了本地角色。通过访问用户对象就可以找到当前用户。当前用户就像其它对象一样可以调用 Zope API 中定义的各种方法。 假设你想在一个 Web 页面上显示当前用户名。用 DTML 可以轻松实现这一点：

```
<dtml-var expr="_.SecurityGetUser().getUserName()">
```

使用 DTML 的 SecurityGetUser 函数或页面模板中的 user，你可以取得当前已经登录的用户。这个 DTML 片断通过对当前用户对象调用 getUserName 方法来检测当前用户。如果用户没有登录，你得到匿名用户名，即：Anonymous User。页面模板中实现的方式如下：

```
<p tal:content="user/getUserName">username</p>
```

适用于脚本的 Zope 安全 API 可以在“API 参考”中找到。适用于 DTML 的 Zope 安全 API 可以在“DTML 参考”中找到。适用于页面模板的 Zope 安全 API 可以在“Zope 页面模板参考”中找到。也可以参考帮助部分。

13. 高级安全专题：所有权和可执行内容

你现在已经学习了所有 Zope 安全方面的基础知识。剩下的是有关所有权和可执行内容方面的高级概念。Zope 使用所有权（ownership）在对象和创建它们的用户之间建立关联，可执行内容（executable content）是指那些执行用户代码的对象，比如脚本、DTML 方法和文档。

对于小型的具有可信赖户的站点，你确实可以忽略这些高级专题。然而，对于一个允许非信任用户创建和管理 Zope 对象的大型站点来说，理解所有权和安全执行内容是重要的。

13.1. 问题：特洛伊木马攻击

涉及到所有权和可执行内容控制的一种情况是特洛伊木马攻击。特洛伊木马是一种对系统的攻击，它诱骗一个用户执行一种潜在的有害行为。一个典型的特洛伊木马化妆成为友善的程序，而一旦你无意中运行它，它就会引起破坏。 所有电脑系统都容易遭受这种攻击。对于基于 Web 的平台系统，这种攻击所必需的是要诱骗某人访问一个执行破坏行为的 URL。 这种攻击非常难以防范。你能够轻而易举的诱骗某人单击一个链接，或者你可以使用更为高级的技巧（例如 Javascript）来引起用户访问怀有恶意的 URL。

Zope 提供了一些避免特洛伊木马的保护方法。Zope 帮助你保护站点避免服务器端特洛伊攻击的方式是通过限制编写 Web 资源的权限来实现的。如果一个不被信任者编写一个 Web 页面，那么，通过这样一种方式，Web 页面对不信任的访问者造成破坏的能力就受到限制。例如，假设不被信任的用户创建一个删除你的站点中所有页面的 DTML 文档或 Python 脚本。如果他们试图观看这个页面，就会失败，这是因为他们没有足够的许可。如果一个管理员观看这个页面，同样会失败，即使管理员员拥有执行这个危险行为的许可。 Zope 使用所有权信息和可执行内容控制来提供这种有限的保护。

13.2. 管理所有权

当用户创建一个 Zope 对象，他就拥有那个对象。没有所有者的对象被认为是无归属。所有权信息存储在对象本身中，这类似于 UNIX 如何跟踪文件的所有者。 通过观看 Ownership 管理标签，你会找到一个对象所有权情况，如下图所示。

管理所有权设置

这个屏幕告诉你对象是否有所有者以及所有者是谁。如果对象被其他的某个人所拥有，并且你拥有接管所有权许可（Take ownership permission），你就能接管对象的所有权。通过选中所有下级对象复选框中的 Take ownership，可以接管所有下级对象的所有权。如果所有者帐号已经被删除，或者如果对象反过来让你继续管理，此时接管所有权非常有用。

如我们前边所提到的，所有权影响安全策略，这是因为一个用户对于他们所拥有的对象有本地角色 Owner。并且，所有权还影响安全，这是因为它控制角色的可执行内容。

注意，具有 Owner 角色的用户并不一定是对象的实际拥有者。

13.3. 可执行内容的角色

由于 DTML 文档、DTML 方法、SQL 方法、基于 Python 的脚本和基于 Perl 的脚本能够动态生成内容，因此被认为是可执行的。它们的内容同样也可以通过 Web 进行编辑。

当你通过 URL、DTML 或脚本访问可执行对象时，Zope 运行对象的可执行内容。对象行为受限于对象所有者的角色和你的角色。换句话说，一个可执行对象只能够执行那些同时授权给所有者和观看者的行为。这样就避免无特权用户编写破坏性脚本以及诱骗一个权限大的用户执行这个脚本。你不能欺骗其他某个人来执行一个你自己没有被授权执行的行为。这就是 Zope 如何使用所有权避免服务器端特洛伊木马攻击的。

需要强调的是，一般来说，“不拥有”的对象不能够执行。如果在运行可执行对象时出现了问题，有可能是由于所有权设置不正确。

13.4. 代理角色

有时 Zope 系统中的对可执行对象的访问限制并不能完全满足你的需要。对于可执行对象，有时你也许想取消安全－－不管所有权是什么。还有的时候你也许想提供一种可执行对象，它具有一种允许无特权的观看者执行被保护行为的额外权限。代理角色给你提供了一种调整可执行对象角色的方法。

假设你想创建一个邮件表单，它允许匿名用户给站点的 Web 管理员发送电子邮件。发送电子邮件被 Use mailhost services 许可保护。匿名用户通常没有必要拥有这个许可。这是因为你不需要让一些人用你的 Zope 服务器匿名发送电子邮件。

这样安排你的许可所产生的问题是那个发送电子邮件的 DTML 方法无法处理匿名用户。如何克服这个问题？答案是把发送电子邮件的 DTML 方法设置成代理角色，这样一来，当它执行时就拥有了管理员角色。查看你的 DTML 方法的 Proxy 管理标签，如下图所示：

代理角色的管理

选择 Manager，然后单击 Change 按钮。这样就把邮件发送方法的代理角色设置成管理员。注意，你自己必须拥有管理员角色才能把它设置为代理角色。现在，当任何人－－匿名或不是匿名－－都可以运行你的邮件发送方法了，它按照管理员角色执行，并且有发送电子邮件的授权。

代理角色融入了一些可执行的许可。这样一来，你就能够使用它们来限制安全。例如，如果你把一个脚本的代理角色设置成匿名角色，然后除了匿名用户以外，脚本永远不执行任何其他角色－－不管所有者（owner）角色和观看者（viewer）角色。

要小心使用代理角色，这是因为他们可以被用于避开默认的安全限制。

14. 总结

安全由两个过程组成－－验证和授权。用户文件夹控制验证，安全策略控制授权。Zope 安全与位置概念密切相关。用户有位置，安全策略有位置，甚至角色也能有位置。创建一种有效的安全架构需要关注位置。当遇到疑惑时，请参考本章所讨论的安全模式。

第十二章 高级 DTML

DTML 是一种“做你想做的”的语言。当它做你真正想让它做的事情的时候，它是好的，但是当它做一些你不想让它做的事情的时候，它就不好玩了。本章给你讲解如何让 DTML 做你真正想做的事情。读完本章，将学会完成以下任务：

调用和修改 REQUEST 对象

Modify the current namespace 修改当前的名称空间

通过 DTML 调用其它脚本

发送可以带附件的电子邮件

通过 DTML 处理例外

开始学习前的建议：

在深入学习 DTML 之前，最好先了解 Python 语言。

理解获取机制是如何工作的。

如果想用 DTML 编写复杂的功能，那么应该考虑使用 Python 脚本。维护更为方便。

在构建大型站点之前要理解 DTML 文档和 DTML 方法之间的区别。

DTML 相对来说是有些复杂的。而对于完成页面表现这样的任务是非常轻松的。

以下的例子显示了一个容易犯的错误。设想你有一个名为 zooName 的 DTML 文档。这个文档包含一个 HTML 表单，如下：

```
<dtml-var standard_html_header>
  <dtml-if zooName>
    <p><dtml-var zooName></p>
  <dtml-else>
    <form action="<dtml-var URL>" method="GET">
      <input name="zooName">
      <input type="submit" value="What is zooName?">
    </form>
```

```
  </dtml-if>

<dtml-var standard_html_footer>
```

这些代码看起来很简单。这是一个调用它自己的 HTML 页面。HTML 的 action 是 URL 变量，这个 URL 变量就是 DTML 文档的 URL。如果存在一个 zooName 变量，那么页面会打印它。如果不存在 zooName 变量，页面显示一个请求输入这个变量的表单。当你单击 Submit，你输入的数据使得 if 求值为真，这段代码应该打印出表单中所输入的内容。

不幸的是，这就属于那些 DTML 不做你想做的事情的情况之一，这是因为包含这个 DTML 的 DTML 文档的名称也是 zooName，并且 DTML 没有使用请求（request）以外的变量，它使用它自己的变量，这样就引起了调用自己，然后再调用自己——无限——直到溢出递归错误。因此，不是做你真正想做的，DTML 却给你发送一条错误信息。这就是初学者混淆的地方。在以下的几节，我们给你显示如何修正这个例子，让 DTML 做你想做的事情。

1. 如何搜索变量

在 zooName 文档中的 DTML 错误能够用两种方法修正。第一种方法是重命名文档，比如 zopeNameFormOrReply。然而，如果你采用这种方法，你必需始终记着这个特殊的例外（文档的名称和变量的名称相同）。第二种修正错误的方法是理解如何搜索名称以及明确指出名称来自于名称空间的何处。

DTML 名称空间是以堆栈形式排列的对象集合。堆栈是一个对象列表，这个列表通过把对象推进和溢出堆栈来操纵对象。 当运行一个 DTML 文档或方法时，Zope 通过创建一个 DTML 名称空间来查找 DTML 变量名。理解 DTML 名称空间的工作方式是重要的，这样你就能精确的预知 DTML 如何定位变量。一些容易遇到的 DTML 问题可以通过理解 DTML 名称空间来解决。

当 Zope 在 DTML 名称空间堆栈里查找名称，它先查看堆栈中最顶部的对象。如果找不到名称，就向下查看下一个条目。DTML 沿着堆栈向下查找，依次检测每个对象，直到它发现要找的名称。

如果直到堆栈的底部，检测了所有情况，还是没有找到要找的名称，那么就生成一个错误。例如，试图查找不存在的名称 unicorn：

```
<dtml-var unicorn>
```

只要不存在名称 unicorn，观看这个 DTML 就会返回一个错误，如下图所示。

不能找到变量时提示的错误消息

注意，DTML 堆栈不包含所有的名称，这是因为 DTML 不从空堆栈开始。甚至在开始执行 DTML 以前，许多对象被推进名称空间堆栈是可能的。

2. DTML 名称空间

对于 Zope 中的每个请求，会动态建立 DTML 名称空间。当你通过 Web 调用一个 DTML 方法或 DTML 文档时，DTML 名称空间都从两个堆栈元素开始－－客户对象（client object）和请求（request），如下图所示。

初始 DTML 名称空间堆栈

客户对象是位于 DTML 名称堆栈顶部的第一个对象。客户对象是什么取决于你是否在执行一个 DTML 方法或一个 DTML 文档。在我们前边的例子中，客户对象被命名为 zooName，这就是它不能正常运行的原因。我们真正想要的表单输入来自于 Web 请求，但是首先查找客户对象。

请求名称空间通常在 DTML 名称空间堆栈底部，因此，是最后进行查找名称的名称空间。这就意味着我们必需在例子中明确的声明我们想要的名称空间。我们可以这样处理：

```
<dtml-var standard_html_header>
```

```
  <dtml-with REQUEST only>
    <dtml-if zooName>
      <p><dtml-var zooName></p>
    <dtml-else>
      <form action="<dtml-var URL>" method="GET">
        <input name="zooName">
        <input type="submit" value="What is zooName?">
      </form>
    </dtml-if>
  </dtml-with>
<dtml-var standard_html_footer>
```

这里，with 标记符表示了只在 REQUEST 名称空间里查找 zooName。

3. DTML 客户对象

DTML 中的客户对象取决于是否在执行 DTML 方法还是 DTML 文档。在文档的情况中，客户对象通常是文档本身，或者，换句话说，一个 DTML 文档是它本身的客户对象。

然而，DTML 方法能够拥有不同种类的客户对象，这取决于它是如何被调用的。比如，假设你有一个显示文件夹中所有内容的 DTML 方法，那么客户对象就是文件夹。这个客户对象可以变化，它取决于当前方法正在显示的那个文件夹。例如，看看以下这个位于根文件夹中的名为 list 的 DTML 方法。

```
<dtml-var standard_html_header>
<ul>
<dtml-in objectValues>
  <li><dtml-var title_or_id></li>
</dtml-in>
</ul>
```

<dtml-var standard_html_footer>

这个方法显示什么取决于调用的方式。如果你通过URL http://localhost:8080/Reptiles/list 对Reptiles文件夹调用这个方法，就会得到如下图所示的内容。

- Snakes
- Lizards

对Reptiles文件夹应用list方法

然而，如果你要用URL http://localhost:8080/Birds/list 对Birds文件夹调用这个方法，你就会得到不同的内容，只会显示Parrot 和 Raptors。 相同的DTML方法，产生不同的结果。在第一个例子中，list方法的客户对象是Reptiles文件夹，在第二个例子中，客户对象是Birds文件夹。当Zope在第一个例子中查找objectValues变量时，它调用Reptiles 文件夹的objectValues方法。在第二个例子中，它调用Birds文件夹的objectValues方法。 换句话说，客户对象是首次查找方法和属性这样的变量时所用到的对象。

对于刚刚接触Zope的人来说，经常弄不清楚如何选择DTML方法还是DTML文档。如何区分呢？可以参考以下原则：

对象有自己的属性吗？如果有，应该使用DTML文档，而DTML方法不能继承属性。

对象需要象一个页面那样进行调用吗？如果是这样，则应该使用DTML文档。这是由于通过DTML文档的属性，比如title属性，可以更容易进行控制。

对象需要与环境交互吗？如果这样，则应该使用DTML方法。这是由于DTML方法可以对其它对象进行调用。

4. DTML请求对象

请求（request）对象位于DTML名称空间堆栈的底部。这个请求对象包含所有与当前Web请求相关的信息。 就像客户对象使用获取（acquisition）可以在多个地方查找变量一样，请求对象同样在多个地方查找变量。当这个请求查找一个变量时，它按照以下顺序在这些来源中查找变量：

CGI环境。公用网关接口或CGI接口定义了一组标准的环境变量，这些变量可供动态Web脚本调用。这些变量在REQUEST名称空间里提供。

表单数据。如果当前请求来自于表单，那么带有请求的任何由表单提交输入的数据可以在REQUEST对象中找到。

Cookies. 如果当前客户的请求含有Cookies，可以在当前REQUEST对象的Cookies中找到它们。

它变量。REQUEST名称空间给你提供许多其它有用的信息，例如当前对象的URL和它的所有上级对象。

请求名称空间在Zope中非常有用，这是因为它是客户（即浏览器）通过提供表单数据、cookies和其它有关本身的信息与Zope进行交流的基本方式。有关请求对象的更多信息，可参考“API 参考”。 一个简单明了的例子是在DTML中输出REQUEST：

```
<dtml-var standard_html_header>

<dtml-var REQUEST>

<dtml-var standard_html_footer>
```

结果如下图所示：

显示 request

因为请求是在客户对象之后，如果名称在请求和客户对象中都存在，DTML 总是首先在客户对象中查找它们。这会是一个问题接下来，让我们通过直接控制 DTML 查找变量的方式来看一些解决问题的方法。

4.1. 调用变量

当你使用 var 标记符插入一个变量时，Zope 首先使用 DTML 名称空间查找这个变量。然后，Zope 调用它并且插入结果。调用意味着把一个对象或数值转换成适合显示的字符串。Zope 通过使用 Python 当中的把对象强制转换成字符串的标准方法来处理简单变量。对于复杂的对象，例如 DTML 方法和 SQL 方法，Zope 调用并执行这种对象，而不是仅仅试图把它转换成字符串。这使你能够把 DTML 方法插入到其它 DTML 方法里。

通常，Zope 按照你所希望的方式调用变量。只有当你开始使用更高级的技巧时才需要知道调用的过程。在本章的后边，我们会看一些关于如何使用 DTML 的 getitem 效用函数控制调用过程的例子（见“DTML 名称空间效用函数”）。

4.2. 修改 DTML 名称空间

现在，你已经看到了 DTML 名称空间是一种堆栈，你可能会觉得疑惑，新对象是如何或者为什么被推进堆栈中。 一些 DTML 标记符在它们执行的时候可以修改 DTML 名称空间。一个标记符在执行的期间可以把一些对象推进名称空间堆栈。这些标记符包括 in 标记符，with 标记符和 let 标记符。

4.2.1. 修改 in 标记符名称空间

当 in 标记符对一个序列进行迭代时，它把当前的序列中的数据项推进到名称空间堆栈的顶部：

```
<dtml-var getId> <!-- This is the id of the client object -->

<dtml-in objectValues>

  <dtml-var getId> <!-- this is the id of the current item in the

                        objectValues sequence -->

</dtml-in>
```

你已经在本书的例子中多次看到这种情况了。当 in 标记符对一个序列进行迭代的时候，在 in 标记符块内容的持续期内，每个数据项都被推进到名称空间堆栈。当每次执行完时，序列中的当前数据项溢出 DTML 名称空间堆栈，同时序列中的下一个数据项被推进堆栈。

4.2.1.1. 注意事项

更准确的说，in 标记符可以在名称空间堆栈中加入条目。此时常用的变量有：

sequence-item: 当前正在迭代的条目

sequence-start: 如果当前条目是序列中的第一个，那么这个变量值为真。

sequence-end: 如果当前条目是序列中的最后一个，那么这个变量值为真。

sequence-length: 序列的长度。

previous-sequence: 如果当前的一批数据不是第一组，那么这个变量值为真。

next-sequence: 如果当前的一批数据不是最后一组，那么这个变量值为真。

in 标记符还包含一些变量，可参见“DTML 参考”。

4.2.2. with 标记符

with 标记符在 with 块的持续期内把一个指定的对象推进到名称空间堆栈的顶部。这使你能够指定第一次查找变量时的位置。当 with 块结束时，对象溢出名称空间堆栈。

考虑一个含有方法和属性的文件夹，你可以用以下的 Python 表达式访问这些名称：

```
<dtml-var standard_html_header>
```

```
<dtml-var expr="Reptiles.getReptileInfo()">

<dtml-var expr="Reptiles.reptileHouseMaintainer">

<dtml-in expr="Reptiles.getReptiles()">

  <dtml-var species>

</dtml-in>

<dtml-var standard_html_footer>
```

注意，为了取得Reptiles文件夹的有关数据，这段代码添加了许多复杂功能。使用with标记符，你可以使这个例子更为清晰：

```
<dtml-var standard_html_header>

<dtml-with Reptiles>

  <dtml-var getReptileInfo>

  <dtml-var reptileHouseMaintainer>

  <dtml-in getReptiles>

    <dtml-var species>

  </dtml-in>

</dtml-with>

<dtml-var standard_html_footer>
```

使用with标记符的另外一个情况是需要把请求或请求中的一部分放置到名称空间堆栈顶部。例如，假设你有一个表单，它包含一个名为id的输入变量。如果你试图像以下这样通过查找id变量来处理这个表单：

```
<dtml-var id>
```

这种方式不会得到表单的id变量，得到的是客户对象的id。一种解决方法是使用with标记符把Web请求中的表单推进到DTML名称空间堆栈的顶部：

```
<dtml-with expr="REQUEST.form">

  <dtml-var id>
```

```
</dtml-with>
```

这样就确保你首先得到表单里的 id。请参考 API 文档。

如果你提交的表单没有为 id 变量提供值，位于名称空间堆栈顶部的表单无效，这是因为表单不包含一个 id 变量。你最终还是得到客户对象的 id，这是因为当 DTML 在表单中查找 id 变量失败以后，它搜索客户对象。with 标记符有一个属性，这个属性使你能够调整 DTML 名称空间，使之仅包括你指定的对象：

```
<dtml-with expr="REQUEST.form" only>
  <dtml-if id>
    <dtml-var id>
  <dtml-else>
    <p>The form didn't contain an "id" variable.</p>
  </dtml-if>
</dtml-with>
```

使用 only 属性使你能够确定查找变量的位置。

4.2.3. let 标记符

let 标记符使你能够把一个新名称空间推进到名称空间堆栈中。这个名称空间通过 let 标记符的属性定义：

```
<dtml-let person="'Bob'" relation="'uncle'">
  <p><dtml-var person>'s your <dtml-var relation>.</p>
</dtml-let>
```

显示结果为：

```
<p>Bob's your uncle.</p>
```

let 标记符的作用在很多方面是和 with 标记符一样的。let 标记符主要的好处在于你能够用它定义多个在块中使用的变量。let 标记符创建一个或多个新变量和对应值，并且还把一个含有那些变量和对应值的名称空间对象推进到 DTML 名称空间堆栈中。通常，with 标记符常用于把现有的对象推进到名称空间堆栈中，而 let 标记符更适用于为块定义新变量。

当你发现自己正在编写复杂的使用新变量的 DTML 时，用 Python 或 Perl 处理相同的事情可能会是一种更好的解决方法。高级脚本在“高级 Zope 脚本”中讲述。

DTML 名称空间比较复杂，这种复杂经过长期演变而来。它不仅可以用来帮助理解名称来自何处，但更有益于明确指定名称的位置。With 和 let 标记符使你能够改变名称空间，从而可以引用你所需要的对象。

4.3. DTML 名称空间效用函数

在 Zope 里，DTML 名称空间也是一种对象。它能够使用 DTML 里的 _（下划线）对象直接访问。_名称空间可以认为是辅助名称空间（under namespace）。 针对某些程序任务，辅助名称空间给你提供许多有用的方法。让我们看其中的几个。

比方说，你想打印你的名字三次。这可以通过 in 标记符完成，但是你如何明确的告诉 in 标记符循环三次呢？只需要传递一个带有三个条目的序列就可以了：

```
<dtml-var standard_html_header>

<ul>

<dtml-in expr="_.range(3)">

  <li><dtml-var sequence-item>: My name is Bob.</li>

</dtml-in>

</ul>

<dtml-var standard_html_footer>
```

_.range(3) 这个 Python 表达式返回一个序列，它包含三个整数：0、1 和 2。range 函数是标准的 Python 内建函数，并且许多 Python 内建函数可以通过 _ 名称空间访问，包括：

'range([start,], stop, [step])' -- 返回整数列表，从 start 到 stop，步长为 step，start 默认为 0，step 默认为 1。例如'_range(3,10,2)'结果为'[3,5,7,9]'。

'_.len(sequence)' -- 'len' 返回序列大小。

许多名称来自于 Python 语言，这种语言包含了一套被称为内建函数的特殊函数集。Python 的思想是要拥有一套小型的内建名称集。Zope 的思想可以被认为是拥有一套大型的复杂的内建名称数组。

辅助名称空间还可以用于控制所要查找的变量。这是一种非常普通的用法。你已经看到 in 标记符定义了许多特殊变量，例如 sequence-item 和 sequence-key，你可以在循环中使用这些变量来帮助显示和控制循环。如果你想在 Python 表达式中使用这些变量中的一个，那该怎么办？

```
<dtml-var standard_html_header>

<h1>The squares of the first three integers:</h1>
```

```
<ul>
<dtml-in expr="_.range(3)">
  <li>The square of <dtml-var sequence-item> is:
    <dtml-var expr="sequence-item * sequence-item">
  </li>
</dtml-in>
</ul>
<dtml-var standard_html_footer>
```

试试这个，管用吗？不！为什么？问题存在于这个 var 标记符：

```
<dtml-var expr="sequence-item * sequence-item">
```

记住，在 Python 表达式属性中的任何内容必须是一个有效的 Python 表达式。在 DTML 中，sequence-item 是一个变量名称，但是在 Python 中，它意味着“对象 sequence 减去对象 item”。这不是你想要的。你真正想要的是变量 sequence-item。一种解决方法是使用 prefix 属性，比如：

```
<dtml-var standard_html_header>
<h1>The squares of the first three integers:</h1>
<ul>
<dtml-in prefix="loop" expr="_.range(3)">
  <li>The square of <dtml-var loop_item> is:
    <dtml-var expr="loop_item * loop_item">
  </li>
</dtml-in>
</ul>
<dtml-var standard_html_footer>
```

prefix 属性使得 in 标记符变量重新命名，它用 prefix 前缀加上下划线，而不是“sequence-”。所以上边的例子中，"sequence-item" 变成了 "loop_item"。

另外一种方法是使用 getitem 效用函数来明确的指定要查找的变量：

```
The square of <dtml-var sequence-item> is:

<dtml-var expr="_.getitem('sequence-item') *

                _.getitem('sequence-item')">
```

getitem 函数采用第一个参数来查找名称。现在，这个 DTML 方法就可以正确显示最初 3 个整数的积。getitem 方法接受一个可选的第二个参数，它指定是否呈递这个变量。以呈递 DTML 变量方式调用意味着把变量转换为一个字符串。默认的情况下，getitem 函数不呈递一个变量。 以下代码显示了如何插入一个名为 myDoc 的呈递过的变量：

```
<dtml-var expr="_.getitem('myDoc', 1)">
```

这个例子从某种角度来说是没有意义的，这是因为它等同于：

```
<dtml-var myDoc>
```

可是，假设你有一个表单。通过这个表单，一个用户可以从选择列表中选择他们想看的文档。假设表单有一个名为 selectedDoc 的输入项，这个输入项包含文档的名称。你就可以象以下这样显示被呈递的文档：

```
<dtml-var expr="_.getitem(selectedDoc, 1)">
```

注意，在以上的例子中，selectedDoc 没有在引号中。我们不想插入字符“selectedDoc”，而是想插入变量 selectedDoc 的值。例如，selectedDoc 的值可能会是 chapterOne。使用这种方式，你就能够通过使用动态值而不是静态文字来查找一个条目。

如果你是一个 Python 程序员，并且开始使用 DTML 完成复杂的任务，请考虑用 Python 脚本完成大部分原本用 DTML 处理的工作。这些将在“高级 Zope 脚本”中详细阐述。使用 Python 可以回避 DTML 中的许多难题。

5. DTML 安全

Zope 能够被许多不同类型的用户使用。例如，Zope 站点，Zope.org（ http://www.zope.org/），在撰写本书的时候成员已经超过 11,000 个。每个成员都可以登录 Zope，添加对象和新闻条目，以及管理他们自己的个人区域。

因为 DTML 是一种脚本语言，在处理对象和属性方面非常灵活。如果安全系统没有囊括 DTML，那么一个用户可能会潜在的创建怀有恶意的或侵犯他人隐私的代码。

DTML 受到标准的 Zope 安全设置的限制。因此，如果不允许你通过指向对象的 URL 方式来访问对象，你就不能通过 DTML 访问这个对象。你不能使用 DTML 欺骗 Zope 安全系统。

例如，假设你有一个名为 Diary 的 DTML 文档，它是私有的。匿名用户不能通过 Web 访问你的 Diary。如果一个匿名用户观看 DTML 并且试图访问你的 Diary，它们就会被拒绝：

```
<dtml-var Diary>
```

DTML 验证当前用户是否已授权可以访问所有的 DTML 变量。如果用户没有授权，那么安全系统引发一个 Unauthorized 错误，并且要求用户提供更多的特权验证信任信息。

在“用户和安全”一章中，你可以阅读与可执行内容有关的安全规则方面的内容。有多种方法可以用来调整 DTML 文档或方法的角色，从而可以在不考虑角色的情况下访问被限制的变量。

5.1. 安全脚本限制

DTML 不允许消耗大量内存或执行无穷循环和递归。对循环和内存的限制是相当严格的，DTML 不适合处理复杂、消耗多的程序逻辑。例如，你不能用_.range 效用函数创建巨大的列表。不能通过 DTML 直接访问文件系统。

记住，这些安全限制是简单的，并且有可能被一些用户突破。通常，让任何你不信任的人在站点中编写 DTML 代码是不明智的

6. 高级 DTML 标记符

在本章的剩下部分，我们讲讲其它的 DTML 标记符。在“DTML 参考”中总结了这些标记符。DTML 有一套内置的标记符，可以完成常见的任务。当然，也可以添加新的标记符。在 Zopr.org 可以找到相关的文章和新的 DTML 标记符。

这一节讲述 Zope 各种标记符所具有的用途。除了用于处理例外的 DTML 标记符外，这些标记符很难进行分类。

6.1. call 标记符

var 标记符调用方法，但是它还插入返回值。使用 call 标记符，你能调用方法而不用在输出中插入它们的返回值。这种方式在你对调用一个方法的效果更感兴趣而不是对它所返回的值更感兴趣的时候是有用的。

例如，当你想更改属性 animalName 的值，你会对调用 manage_changeProperties 方法的效果更感兴趣，而不是返回值。这有一个例子：

```
<dtml-if expr="REQUEST.has_key('animalName')">
  <dtml-call expr="manage_changeProperties(animalName=REQUEST['animalName'])">
  <h1>The property 'animalName' has changed</h1>
<dtml-else>
  <h1>No properties were changed</h1>
```

```
</dtml-if>
```

在这个例子中，根据是否存给定的名称，页面更改一个属性。manage_changeProperties 方法的结果不一定需要显示出来。

call 标记符另外一个常见用法是调用影响客户行为的方法，例如 RESPONSE.redirect 方法。在这个例子中，你让客户端程序重定向到一个不同的页面，更改被重定向的页面，并且更改在 let 标记符中定义的 target 变量值：

```
<dtml-var standard_html_header>

<dtml-let target="'http://example.com/new_location.html'">

  <h1>This page has moved, you will now be redirected to the

  correct location.  If your browser does not redirect, click <a

  href="<dtml-var target>"><dtml-var target></a>.</h1>

  <dtml-call expr="RESPONSE.redirect(target)">

</dtml-let>

<dtml-var standard_html_footer>
```

简而言之，call 标记符和 var 标记符非常类似，区别在于它不插入调用变量的结果。

call 标记符还可以用来调用 ZSQL 方法或处理 REQUEST。以下是两个调用 ZSQL 方法的例子：

```
<dtml-call "insertLogEntry(REQUEST)">
```

或:

```
<dtml-call "insertLogEntry(logInfo=REQUEST.get('URL0'), severity=1)">
```

调用一个处理 REQUEST 的 Python 脚本：

```
<dtml-call "preprocess(REQUEST)">
```

6.2. comment 标记符

使用 comment 标记符，DTML 可以用注释进行说明

```
<dtml-var standard_html_header>

<dtml-comment>
```

```
  This is a DTML comment and will be removed from the DTML code

  before it is returned to the client.  This is useful for

  documenting DTML code.  Unlike HTML comments, DTML comments

  are NEVER sent to the client.

</dtml-comment>

<!--

  This is an HTML comment, this is NOT DTML and will be treated

  as HTML and like any other HTML code will get sent to the

  client.  Although it is customary for an HTML browser to hide

  these comments from the end user, they still get sent to the

  client and can be easily seen by 'Viewing the Source' of a

  document.

-->

<dtml-var standard_html_footer>
```

注释块将在DTML输出中删除。 除了用于说明DTML，你可以使用comment标记符临时注释掉其它DTML标记符。稍后，你可以删除comment标记符，使你的DTML重新生效。

6.3. tree标记符

tree标记符使你能够轻松的用HTML构建动态树，用以显示分等级的数据。树是一种用图形表示数据的方法，它起始于“根”对象，“根”对象下面又有对象，它们通常被称为“分支”。分支可以拥有它们自己的分支，就像真的树一样。这个概念对于那些使用过文件管理程序（例如Microsoft Windows Explorer，用于浏览文件系统）的人们来说应该是熟悉的。Zope管理界面中左边的导航视图就是使用tree标记符创建的。 For example here's a tree that represents a collection of folders and sub-folders. 例如，下图显示了一个树，它显示了一个文件夹和下级文件夹的集合。

tree 标记符生成的树结构

生成这个树的 DTML 显示如下：

```
<dtml-var standard_html_header>

<dtml-tree>

  <dtml-var getId>

</dtml-tree>

<dtml-var standard_html_footer>
```

tree 标记符查询对象，用以找到它们的下级对象，并且精细地把结果以树的形式显示出来。tree 标记符块作为模板来显示树的节点。

因为基本的 Web 协议（HTTP）是没有状态的，每次你查看一个页面时，你需要记住树处于什么状态。要实现这个要求，Zope 把树的状态存储在一个 cookie 中。因为这个树的状态被存储在一个 cookie 中，所以一次只能在一个页面中显示一个树，否则将会错误的调用相同的 cookie。

你可以用相当多的 tree 标记符属性和特殊变量来调整 tree 标记符的行为。tree 标记符属性举例如下：

branches

用于查找下级文件夹的方法名称。这个方法默认为 tpValues，它是由一些标准 Zope 对象定义的方法。

leaves

用于显示没有下级分支对象的方法名称。

nowrap

0或1，如果为0，分支文字没有约束，相反，文字可能被缩短。默认值为0。

sort

文字被插入以前，对分支进行排序。属性值是将要被排序的数据项属性的名称。

assume_children

0或1，如果为1，那么所有对象被假设有下级对象，因此，当它们合拢时总是在前面有一个加号。只有当一个条目被展开时才会查找下级对象。在重获下级对象是个消耗多的过程时，这就会是一个好的选项。默认值为0。

single

0或1，如果为1，则只有树中的一个分支能够扩展。当一个新分支展开时，任何已经展开的分支合拢。默认值为0。

skip_unauthorized

0或1，如果为1，当试图显示用户没有足够权限的下级对象时，不引发错误。被保护的下级对象不被显示。默认值为0

假设你想使用tree标记符创建一个动态站点地图。你不想把每个页面都在站点地图中显示出来。让我们假定，你对要在站点地图中显示的文件夹和文档赋予一个属性。

让我们首先定义一个id为publicObjects的脚本：

```
## Script (Python) "publicObjects"

## Returns sub-folders and DTML documents that have a true 'siteMap' property.

results=[]

for object in context.objectValues(['Folder', 'DTML Document']):

    if object.hasProperty('siteMap') and object.siteMap:

        results.append(object)

return results
```

现在我们就能够创建一个使用tree标记符和这个脚本的DTML方法，画一个站点地图：

```
<dtml-var standard_html_header>

<h1>Site Map</h1>

<p><a href="&dtml-URL0;?expand_all=1">Expand All</a> |

   <a href="&dtml-URL0;?collapse_all=1">Collapse All</a>

</p>

<dtml-tree branches="publicObjects" skip_unauthorized="1">

  <a href="&dtml-absolute_url;"><dtml-var title_or_id></a>

</dtml-tree>

<dtml-var standard_html_footer>
```

这个 DTML 方法画出了所有公开的资源的链接，并且用树显示它们。下图显示了站点地图的样子。

使用 tree 标记符动态生成的站点地图

有关 tree 标记符参数和特殊变量的总结，请参见“DTML 参考”。

6.4. return 标记符

通常，DTML 创建文本输出。你还可以让 DTML 返回除了文本以外的其它值。使用 return 标记符，你能够让 DTML 方法返回任意值，就像一个基于 Python 或 Perl 的脚本

举例如下：

```
<p>This text is ignored.</p>

<dtml-return expr="42">
```

这个 DTML 方法返回数字 42。需要注意的是在 return 标记符以后 DTML 执行结束。对于 return 标记符，大部分功能都可以用 Python 脚本代替。

6.5. sendmail 标记符

sendmail 标记符可用于格式化和发送电子邮件信息。你可以使用 sendmail 标记符连接一个现有的邮件主机，或者手工指定 SMTP 主机。以下是一个关于如何用 sendmail 标记符发送电子邮件消息的例子：

```
<dtml-sendmail>

To: <dtml-var recipient>

From: <dtml-var sender>

Subject: Make Money Fast!!!!

Take advantage of our exciting offer now! Using our exclusive method

you can build unimaginable wealth very quickly. Act now!

</dtml-sendmail>
```

注意，一个额外的空行把邮件头和消息正文分开了。

sendmail 标记符普遍用法是发送一个由反馈表单生成的邮件消息。sendmail 标记符可以包含任何你需要的 DTML 标记符，因此使用表单数据调整你的消息是很容易的。

6.6. mime 标记符

mime 标记符允许你使用 Multipurpose Internet Mail Extensions（多目标 Internet 邮件扩展(MIME)）来格式化数据。MIME 是一种用于在邮件消息中对数据进行编码的 Internet 标准。通过使用 mime 标记符，你能使用 Zope 发送带有附件的电子邮件。 假设你想把你的简历上载到 Zope，然后让 Zope 把这个文件发送给潜在雇主列表。 以下是上载表单：

```
<dtml-var standard_html_header>

<p>Send you resume to potential employers</p>

<form method=post action="sendresume" ENCTYPE="multipart/form-data">

<p>Resume file: <input type="file" name="resume_file"></p>

<p>Send to:</p>

<p>
```

```
<input type="checkbox" name="send_to:list" value="[jobs@yahoo.com]">
  Yahoo<br>
<input type="checkbox" name="send_to:list" value="[jobs@microsoft.com]">
  Microsoft<br>
<input type="checkbox" name="send_to:list" value="[jobs@mcdonalds.com]">
  McDonalds</p>
<input type=submit value="Send Resume">
</form>
<dtml-var standard_html_footer>
```

注意，添加在 input 中的文字“:list”是用来指定字段数据类型的。比如，如果选择了前两个复选框，那么 REQUEST 中 send_to 的值为为：[jobs@yahoo.com, jobs@microsoft.com]

创建另外一个名为 sendresume 的 DTML 方法来处理表单并发送简历文件：

```
<dtml-var standard_html_header>
<dtml-if send_to>
  <dtml-in send_to>
    <dtml-sendmail smtphost="my.mailserver.com">
    To: <dtml-var sequence-item>
    Subject: Resume
    <dtml-mime type=text/plain encode=7bit>
    Hi, please take a look at my resume.
    <dtml-boundary type=application/octet-stream disposition=attachment
    encode=base64><dtml-var expr="resume_file.read()"></dtml-mime>
    </dtml-sendmail>
```

```
  </dtml-in>

  <p>Your resume was sent.</p>

<dtml-else>

  <p>You didn't select any recipients.</p>

</dtml-if>

<dtml-var standard_html_footer>
```

这个方法对 sendto 变量进行迭代，并且为每个条目发送一个电子邮件。注意，在 To: header 和 mime 标记符开始之间没有空行。如果一个空行被插入到它们之间，那么消息就不被接收邮件阅读器译成多部件（multipart）消息。

注意，在 boundary 标记符和 var 标记符之间或 var 标记符结束和 mime 标记符关闭之间不存在新行。这是重要的，如果你用新行中断了标记符，那么它们将被编码，并且包括 MIME 部分，这可能不是你所希望的。

需要说明的是，mime 标记符可以任意签套在 mime 标记符中。

6.7. unless 标记符

unless 标记符执行代码块，除非给定条件为真。unless 标记符与 if 标记符相反。看看以下 DTML 代码：

```
<dtml-if expr="not butter">

  I can't believe it's not butter.

</dtml-if>
```

等同于：

```
<dtml-unless expr="butter">

  I can't believe it's not butter.

</dtml-unless>
```

unless 标记符的目的是什么？它只是一个方便的标记符。 unless 标记符比 if 标记符更为有限，这是因为它不能包含一个 else 或 elif 标记符。

就像 if 标记符一样，可以按照名称方式调用 unless 标记符，因此：

```
<dtml-unless the_easter_bunny>
```

```
  The Easter Bunny does not exist or is not true.

</dtml-unless>
```

检查 the_easter_bunny 是否存在，同时检查它的真假值。然而，以下例子只检查 the_easter_bunny 的真假值。

```
<dtml-unless expr="the_easter_bunny">

  The Easter Bunny is not true.

</dtml-unless>
```

如果 the_easter_bunny 不存在，这个例子引发一个例外。通过 unless 标记符可以实现的任何事情用 if 标记符也可以实现。这样一来，它的使用是完全可选择的，并且是一种编程风格。

6.8. 用 in 标记符进行成批处理

你经常需要呈现一个大的信息列表，但是一次只能给用户显示一屏。例如，如果一个用户查询你的数据库并且得到 120 个结果，你可能只想给用户显示一小批——比如说每个页面是 10 或 20 个结果。把大的列表分解成多个部分被称为成批化（batching）。成批化有许多好处。

- 用户只需下载一个适当大小的文档，这就胜于一个潜在巨大的文档。这使得页面装载迅速，因为它们比较小。 - 因为使用较小的结果批块（batch），会消耗较少的内存。 - 后一级和前一级导航界面使得显示大量数据相对容易。

in 标记符提供了几个用于帮助进行成批处理的变量。以下的例子说明了如何按照一次批块（batch）包含 10 个条目的方法显示 100 个条目。

```
<dtml-var standard_html_header>

  <dtml-in expr="_.range(100)" size=10 start=query_start>

    <dtml-if sequence-start>

      <dtml-if previous-sequence>

        <a href="<dtml-var URL><dtml-var sequence-query

           >query_start=<dtml-var previous-sequence-start-number>">

           (Previous <dtml-var previous-sequence-size> results)

        </a>

      </dtml-if>
```

```
      <h1>These words are displayed at the top of a batch:</h1>
      <ul>
    </dtml-if>
      <li>Iteration number: <dtml-var sequence-item></li>
    <dtml-if sequence-end>
      </ul>
      <h4>These words are displayed at the bottom of a batch.</h4>
      <dtml-if next-sequence>
        <a href="<dtml-var URL><dtml-var sequence-query
          >query_start=<dtml-var
          next-sequence-start-number>">
        (Next <dtml-var next-sequence-size> results)
        </a>
      </dtml-if>
    </dtml-if>
  </dtml-in>
<dtml-var standard_html_footer>
```

让我们具体看一下DTML如何执行的。首先，我们用了一个in标记符，它迭代100个由range效用函数生成的数字。属性size告诉in标记符一次只显示10个条目。属性start告诉in标记符第一次显示什么数据项序号。

在in标记符内部是两个if标记符。第一个检测专用变量sequence-start。这个变量只有当最初的一组数据经过in块时才为真。因此，这个if标记符只在循环开始时才执行一次。第二个if标记符检测专用变量sequence-end。这个变量只有当最后的一组数据经过in标记符时才为真。因此，第二个if块只在结束时才执行一次。位于if标记符之间的段落每次循环都被执行。

在每个if标记符的内部是另外一个if标记符，它检测专用变量previous-sequence 和 next-sequence。在当前的批块有前一个或后一个批块时变量为真，各个批块都是这样。换句话说，除了第一个批块以外，previous-sequence对于其它所有批块都

为真，同时，除了最后一个批块外，next-sequence 对于其它所有批块都为真。因此，DTML 检测相关的批块是否存在，如果存在，它就显示导航链接。

批块导航由链接组成，它们通过设置 query_start 变量指向所对应的文档，这个 query_start 变量指出 in 标记符在显示批块时应该从哪里开始。要想更好的熟悉这些是如何工作的，单击几次前一级和后一级链接，看看导航链接的 URL 是如何变化的。

最后一点，它使用 next-sequence-size 和 previous-sequence-size 专用变量显示前一级和后一级批块的统计数字。所有的这些最终生成以下 HTML 代码：

```
<html><head><title>Zope</title></head><body bgcolor="#FFFFFF">
  <h1>These words are displayed at the top of a batch:</h1>
  <ul>
    <li>Iteration number: 0</li>
    <li>Iteration number: 1</li>
    <li>Iteration number: 2</li>
    <li>Iteration number: 3</li>
    <li>Iteration number: 4</li>
    <li>Iteration number: 5</li>
    <li>Iteration number: 6</li>
    <li>Iteration number: 7</li>
    <li>Iteration number: 8</li>
    <li>Iteration number: 9</li>
  </ul>
  <h4>These words are displayed at the bottom of a batch.</h4>
     <a href="[http://pdx:8090/batch?query_start=11]">
       (Next 10 results)
     </a>
```

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

另外一个例子通过显示页码方式提供导航栏：

```
<dtml-in "_.range(1,101) "size=10 start=start>
          <dtml-if sequence-start>
            <p>Pages:
            <dtml-call "REQUEST.set('actual_page',1)">
            <dtml-in previous-batches mapping>
              <a href="<dtml-var URL><dtml-var sequence-query>start=<dtml-var "_['batch-start-index']+1">">
              <dtml-var sequence-number></a> 
              <dtml-call "REQUEST.set('actual_page',_['sequence-number']+1)">
            </dtml-in>
            <b><dtml-var "_['actual_page']"></b>
          </dtml-if>
          <dtml-if sequence-end>
            <dtml-in next-batches mapping> 
               <a href="<dtml-var URL><dtml-var sequence-query>start=<dtml-var "_['batch-start-
index']+1">">
               <dtml-var "_['sequence-number']+_['actual_page']"></a>
             </dtml-in>
          </dtml-if>
    </dtml-in>
    <dtml-in "_.range(1,101) "size=10 start=start>
           <br><dtml-var sequence-item>
```

```
</dtml-in>
```

成批处理可以很复杂。一个处理批块的好方法是使用 Searchable Interface 对象为你创建一个批搜索报告单。你就可以修改 DTML 来满足你的需要。

6.9. 处理例外的标记符

Zope 处理例外非常方便。用 raise 和 try 标记符，你就可以获得这种便利。

6.9.1. raise 标记符

你可以用 raise 标记符引发例外。引发例外的一个原因是要显示一个错误。例如，你可以用 if 标记符检测问题，以及出现错误的情况，你就可以用 raise 标记符汇报错误。

raise 标记符有一个用于指定错误类型的 type 属性。错误类型是对错误的简要描述。另外，还有一些标准的错误类型，例如返回 HTTP 错误的 Unauthorized 和 Redirect。Unauthorized 错误导致在用户的浏览器中显示登录提示。你可以引发 HTTP 错误来使 Zope 发送一个 HTTP 错误。例如：

```
<dtml-raise type="404">Not Found</dtml-raise>
```

这会引发一个 HTTP 404 (Not Found)错误。Zope 把 HTTP 404 错误发送给客户端的浏览器。

raise 标记符是一种块标记符。合拢的 raise 标记符块呈递后创建错误消息。如果呈递的文本包含任何 HTML 标记，那么 Zope 把错误消息文本显示在浏览器中，否则，显示一个普通的错误消息。

以下是一个 raise 标记符的例子：

```
<dtml-if expr="balance >= debit_amount">
  <dtml-call expr="debitAccount(account, debit_amount)">
  <p><dtml-var debit_amount> has been deducted from your
  account <dtml-var account>.</p>
<dtml-else>
  <dtml-raise type="Insufficient funds">
    <p>There is not enough money in account <dtml-account>
    to cover the requested debit amount.</p>
  </dtml-raise>
```

```
</dtml-if>
```

引发例外所产生的一个重要作用是引起当前的事务处理被撤销。这意味着任何由 Web 请求所做的修改都被忽略掉了。因此，除了报告错误，例外允许你在突发问题时取消所做的修改。

6.10. try 标记符

如果通过手工方式用 raise 标记符引发一个例外，或者是由于 Zope 遇到错误而造成的结果引发一个例外，在这两种情况下你都可以用 try 标记符捕获例外。例外是 Zope 在执行一个 DTML 文档或方法的过程中遇到的意外错误。当一个例外被检测到以后，正常的 DTML 停止执行。考虑以下的例子：

```
Cost per unit: <dtml-var

                    expr="_.float(total_cost/total_units)"

                    fmt=dollars-and-cents>
```

如果 total_units 不为 0，这个 DTML 工作正常。然而，一旦 total_units 为 0， ZeroDivisionError 例外就被引发，指出一个不合法操作。因此，只要运行这个 DTML，就会返回一个错误消息。

你可以使用 try 标记符来处理这些类型的问题。用 try 标记符，你自己就可以预期和处理错误，这胜于只要发生例外就得到一个 Zope 错误消息。

try 标记符有两个功能。第一个，如果引发一个例外，try 标记符获得执行控制，并且适当处理例外，这样就避免返回一个 Zope 错误消息。第二个，try 标记符允许任何后边的 DTML 继续进行。

在 try 标记符内部，有一个或多个识别和处理不同例外的 except 标记符。当一个例外被引发，依次检测每个 except 标记符，判断它是否匹配例外类型。第一个匹配的 except 标记符处理这个例外。如果在 except 标记符中没有给出例外，那么 except 标记符匹配所有的例外。 以下显示如何使用 try 标记符避免可能在上一个例子中发生的错误：

```
<dtml-try>

  Cost per unit: <dtml-var

                    expr="_.float(total_cost/total_units)"

                    fmt="dollars-and-cents">

<dtml-except ZeroDivisionError>

  Cost per unit: N/A

</dtml-try>
```

如果 ZeroDivisionError 被引发，就执行 except 标记符，Cost per unit: N/A 被呈递。当 except 标记符块完成，DTML 继续执行 try 块后边的语句。

DTML 的 except 标记符采用 Python 的基于类的例外。除了按照名称匹配例外，except 标记符还匹配名为 exception 的任何子类。例如，如果 ArithmeticError 在 except 标记符中被指定，这个标记符就可以处理所有 ArithmeticError 子类，包括 ZeroDivisionError 。对于 Python 例外类和它们的子类列表，请见 Python 参考，例如在线 Python 库参考。一个 except 标记符能够捕获多个例外，方法是把它们都列在同一标记符以内。在一个 except 标记符内部，你可以通过几个专用变量访问被处理的例外的相关信息。

: error_type

被处理的例外的类型。

: error_value

被处理的例外的值。

: error_tb

被处理的例外的回溯(traceback)。

你可以使用这些变量来给用户提供错误消息或者采取不同的行为，例如给 Web 管理员或者错误日志发送电子邮件，这取决于错误的类型。

6.10.1. try 标记符的可选项：else 块

try 标记符有一个可选择的 else 块，如果一个例外没有发生，这个块被呈递。以下是一个如何在 try 标记符内使用 else 标记符的例子：

```
<dtml-try>
  <dtml-call feedAlligators>
<dtml-except NotEnoughFood WrongKindOfFood>
  <p>Make sure you have enough alligator food first.</p>
<dtml-except NotHungry>
  <p>The alligators aren't hungry yet.</p>
<dtml-except>
  <p>There was some problem trying to feed the alligators.<p>
```

```
  <p>Error type: <dtml-var error_type></p>

  <p>Error value: <dtml-var error_value></p>

<dtml-else>

  <p>The alligator were successfully fed.</p>

</dtml-try>
```

第一个用于匹配被引发的错误的类型的 except 块被呈递。如果一个 except 块没有名称，那么它匹配所有被引发的错误。在 try 块里没有发生错误时，可选择的 else 块被呈递。在 else 块中的例外不被前边的 except 块处理。

6.10.2. try 标记符可选项：finally 块

你还能够以另一种略微不同的方式使用 try 标记符。不是象处理例外那样，使用 try 标记符来诱捕例外，而是在例外发生以后清除它们。在 try 标记符内部的 finally 标记符指定了一个要被呈递的清除块，即使当例外发生时都会执行。如果你需要清除一些不能通过撤销事务处理方式清除的内容，finally 块才会有用。 finally 块将总被调用，不管是否存在例外，以及是否使用了 return 标记符。如果你在 try 标记符中使用 return 标记符，finally 块的任何输出被忽略。以下是一个有关如何使用 finally 标记符的例子。

```
<dtml-call acquireLock>

<dtml-try>

    <dtml-call useLockedResource>

<dtml-finally>

    <!-- this always gets done even if an exception is raised -->

    <dtml-call releaseLock>

</dtml-try>
```

在这个例子中，你首先获得一个资源锁，然后试图对被锁定的资源执行一些行为。如果一个例外被引发，你不处理它，但是你确信在把控制权传递给例外处理者之前释放锁。如果所有都正常并且没有引发例外，你仍然要通过执行 finally 块在 try 块的结尾释放锁。 try/finally 形式的 try 标记符很少在 Zope 中使用。这种复杂的程序控制采用 Python 可以更好的完成。

7. 其它有用的例子

在这里提供一些有用的例子。尽管使用 Python 脚本可能可以更好的解决这些问题，但是知道如何通过 DTML 来实现也是非常值得的。

7.1. 转发 REQUEST

我们在前边已经讲过网页的重定向问题。有些情况下，需要把 REQUEST 从一个 DTML 方法转发给另外一个 DTML 方法。在下面这个例子中，从 REQUEST 中取得“type”变量。然后，根据这个值搜索 lookup 变量中对应的 DTML 方法，如果找到了就把 REQUEST 传递给这个 DTML 方法。代码如下：

```
<dtml-let lookup="{'a' : 'form15', 'b' : 'form75', 'c' : 'form88'}">

      <dtml-return "_[lookup[REQUEST.get('type')]]">

</dtml-let>
```

当然，这个例子也可以继续添加例外处理部分。

7.2. 使用<dtml-in>标记符排序

在很多情况下都需要数据排序。DTML 中支持多种排序方式。在下面的例子中，通过调用一个 ZSQL 方法，从 log 表中返回 logTime, logType 和 userName。同时提供不同排序方式的链接。

```
<dtml-comment>

The sorting is accomplished by looking up a sort type

variable in the REQUEST that is comprised of two parts.  All

but the last character indicate the name of the column on

which to sort.  The last character of the sort type indicates

whether the sort should be ascending or descending.

</dtml-comment>

<table>

  <tr>

    <td>Time <a href="<dtml-var URL>?st=logTimea">A</a> <a href="<dtml-var
URL>?st=logTimed">D</a></td>

    <td>Type <a href="<dtml-var URL>?st=logTypea">A</a> <a href="<dtml-var
URL>?st=logTyped">D</a></td>
```

```
        <td>User <a href="<dtml-var URL>?st=userNamea">A</a> <a href="<dtml-var
URL>?st=userNamed">D</a></td>
      </tr>
      <dtml-comment>The line below sets the default sort</dtml-comment>
      <dtml-if "REQUEST.get('st')==None"><dtml-call "REQUEST.set('st', 'logTimed')"></dtml-if>
      <dtml-in getLogData sort_expr="REQUEST.get('st')[0:-1]" reverse_expr="REQUEST.get('st')[-1]=='d'">
        <tr>
          <td><dtml-var logTime></td>
          <td><dtml-var logType></td>
          <td><dtml-var userName></td>
        </tr>
      </dtml-in>
    </table>
```

从Python 脚本中调用DTML对象。

有时需要从Python 脚本程序中调用DTML，示例如下：

```
dtmlMethodName = 'index_html'
return context[dtmlMethodName](container, container.REQUEST)
```

通过这种方式可以把REQUEST传递给指定的DTML。

7.3. 直接搜索

有些时候，需要在定位属性的时候不使用获取机制。在下面的例子中，使用了包含子文件夹的文件夹。每个子文件夹中包含一些图片。每个文件夹和图片都有一个名为desc的属性。

如果你需要调用图片的desc属性，此时若当前的图片没有desc属性，就会调用上级文件夹的desc属性。大多数情况下是我们需要的，可我们有些时候确实只需要这个图片的属性，需要显示出这个图片没有这个属性。这时可以通过aq_explicit来实现。

文件夹结构如下：

```
Folder
  |
  |- Folder1 (desc='Folder one')
  |- Folder2 (desc='Folder two')
       |- Image1 (desc='Photo one')
       |- Image2
       |- Image3 (desc='Photo three')
```

如果按照下面的方式调用 Image2 的 desc 属性，会返回 Folder2 的 desc 属性。

```
<dtml-var "Image2.desc">
```

通过使用 aq_explicit，就可以解决问题：

```
<dtml-var "Image2.aq_explicit.desc">
```

此时，由于找不到 desc 属性，会引发一个错误。更好的方式如下：

```
<dtml-if "_.hasattr(Image2.aq_explicit, 'desc')">
  <dtml-var "Image2.aq_explicit.desc">
<dtml-else>
  No desc property.
</dtml-if>
```

这种方式很实用。

第十三章 高级页面模板

在前面的章节中，你学习了页面模板的基础知识。在本章，你将学习高级特性，包括新的表达式类型和宏。

1. 高级 TAL

你已经学习了一些TAL语句，在这一节，我们将讲述所有的TAL语句，以及它们的各种选项。注意，更为详细的材料请见“Zope页面模板参考”。

1.1. 高级内容插入

你在第五章已经看到了tal:content和tal:replace是如何工作的。在本节，你将学习内容插入方面的一些高级的技巧。

1.1.1. 插入结构

通常，tal:replace 和 tal:content语句会把文本中HTML标记符转化成可显示的字符。比如，把文本里的尖括号<转化为<。如果你想插入原始的未经处理的文本，你需要使用带有structure关键字的表达式。例如：

```
<p replace="structure here/story">
  the <b>story</b>
</p>
```

当你要插入一段存储在一个属性里边或由其他Zope对象生成的HTML或XML时，这个特性是很有用的。例如，你可能有一些新闻条目，这些条目里包含了简单的HTML标记，例如那些显示粗体和斜体的标记。当把它们插入到"Top News"页面里边时，你想保留这些，你就可以写成：

```
<p tal:repeat="newsItem here/topNews"
   tal:content="structure newsItem">
  A news item with<code>HTML</code> markup.
</p>
```

这样就会把带有HTML标记符的新闻条目插入到段落里边。变量here指的是这个模板所在的文件夹。可参考后边的“表达式”部分。通过topNews找到新闻条目，形成一个列表。然后循环处理每个新闻条目，并保持原有的HTML符号。

1.1.2. 虚设元素（Dummy Elements）

通过使用内建的变量nothing，你可以包含模板里可见的页面元素，而在生成的文本里却不可见。比如：

```
<tr tal:replace="nothing">
  <td>10213</td><td>Example Item</td><td>$15.34</td>
</tr>
```

这个功能用于填充最终要被动态内容替换掉的部分。例如，一个具有 10 行的表格在模板里通常只有 1 行。通过添加 9 行虚设行，模板的样子就更像最终的结果了。

并不总是需要在你的页面模板里使用 tal:replace="nothing"机制来加入虚设的内容。例如，你已经看到了一些 tal:content 或 tal:replace 元素内的内容往往在执行后被删掉。在这种情况下，你无须进行任何特定功能来确信虚设内容已经删除掉了。

1.1.3. 默认内容

通过在 tal:content 或 tal:replace 里使用 default 表达式，你可以提供默认内容。例如：

```
<p tal:content="default">Spam<p>
```

这也就意味着：

```
<p>Spam</p>
```

大多数的时候，你需要有选择性的提供默认内容，而不总是使用它。例如：

```
<p tal:content="python:here.getFood() or default">Spam</p>
```

注意：Python 表达式在本章的后边阐述。如果 getFood 方法返回一个真值，那么就把结果插入到段落里，否则使用默认的 Spam。

1.2. 高级循环

你已经在第五章看到了使用 tal:repeat 语句通常情况下可以完成什么任务。本节侧重于讲述一些这个语句的高级特性。

1.2.1. 重复变量

特别值得一提的是重复变量。重复变量提供了当前循环的相关信息。重复变量有以下属性：

index - 重复的序号，从 0 开始

number - 重复的序号，从 1 开始

even - 对于偶数序号(0, 2, 4, ...)为真。

odd - 对于奇数序号(1, 3, 5, ...)为真。

start - 对于起始重复为真(index 0)。

end - 对于结尾或最终的重复为真

length - 序列长度，就是重复总次数

letter - 用小写字母计次，"a" - "z", "aa" - "az", "ba" - "bz", ..., "za" - "zz", "aaa" - "aaz"等等。

Letter - 用大写字母计次。

你可以使用路径表达式或Python表达式来访问重复变量的内容。在路径表达式里，由三部分组成，即repeat名称,语句变量名称和你要的信息名称，例如repeat/item/start。在Python表达式里，使用通常的字典方式就可以得到重复变量，然后就可以访问信息属性，例如'python:repeat['item'].start'

1.2.2. 小技巧

这里是一些有用的小技巧。有些时候，你希望重复一个标记符，但想显示标记符。例如，你可能想重复几个段落标记符，但不需要把它们放在另外一个标记符里。你可以通过使用tal:omit-tab语句实现这个功能。

```
<div tal:repeat="quote here/getQuotes"

    tal:omit-tag="">

  <p tal:content="quote">quotation</p>

</div>
```

上边的代码在执行的时候，不会显示 div 标记府。tal:omit-tag语句在本章后边进行描述。 你可以嵌套tal:zepeat语句。每个tal:repeat语句必须有一个不同的重复变量名。以下是一个显示数学乘方表的例子：

```
<table border="1">

  <tr tal:repeat="x python:range(1, 13)">

    <td tal:repeat="y python:range(1, 13)"

        tal:content="python:'%d x %d = %d' % (x, y, x*y)">

        X x Y = Z

    </td>

  </tr>

</table>
```

这个例子使用了Python 表达式，在本章后边将进行进一步讲述。

如果你已经用过很多次DTML里边的dtml-in重复语句，你可能使用过了批块化。批块化就是把一个大的列表分隔为多个小列表的过程。典型的是用它来在一个web页面里显示一个大列表里的小列表项目。就比如搜索引擎如何批块化显示搜索结果。tal:repeat语句不支持批块化，但是Zope带有一个批块化工具。见本章后边的“批块化”部分。

tal:repeat另外一个没有提供的有用的特性是排序。如果你想对一个列表排序，你或者编写自己的排序脚本（在Python里是相当容易的）,或者你可以使用sequence.sort函数。以下是一个如何按照标题对一个列表排序，然后按照修改日期排序的例子

```
<table tal:define="objects here/objectValues;
                   sort_on python:(('title', 'nocase', 'asc'),
                                   ('bobobase_modification_time', 'cmp', 'desc'));
                   sorted_objects python:sequence.sort(objects, sort_on)">
  <tr tal:repeat="item sorted_objects">
    <td tal:content="item/title">title</td>
    <td tal:content="item/bobobase_modification_time">
      modification date</td>
  </tr>
</table>
```

这个例子通过在sort函数外边定义sort参数。sequence.sort函数的参数是一个序列和排序方式。在这个例子里，排序方式是在sort_on变量里定义的。关于sequence.sort函数的更多信息请参见“API参考”。

1.3. 高级属性控制

你已经用到了tal:attributes语句。你可以用它来动态替换标记符属性，例如，对于a 元素的href属性。通过用分号分隔开属性，可以对一个标记符替换多个属性：

```
<a href="link"
  tal:attributes="href here/getLink;
                  class here/getClass">link</a>
```

你还可以用XML名称空间定义属性，例如：

```
<Description
    dc:Creator="creator name"
    tal:attributes="dc:Creator here/owner/getUserName">
  Description</Description>
```

简单的把 XML 名称空间前缀放在属性名称前面，你可以用 XML 名称空间创建属性。

1.4. 定义变量

通过使用 tal:define 属性，你可以定义变量。这样做有几个原因。一个原因是要避免在模板里重复编写长的表达式。另外一个原因就是要避免重复的调用复杂的方法。一旦你定义了变量，你就可以在一个模板里多次使用。例如，下边是一个定义了变量的列表，后边进行判断，再对它进行重复。

```
<ul tal:define="items container/objectIds"
    tal:condition="items">
  <li tal:repeat="item items">
    <p tal:content="item">id</p>
  </li>
</ul>
```

tal:define 语句创建了变量 items，你就可以在 ul 标记符中的任何地方使用它。还要注意的是，在同一 ul 标记符里是如何使用两个 TAL 语句的。有关如何对一个标记符使用多个语句方面的信息，请见本章后边的“TAL 语句之间的交互”部分。在这个例子里，第一个语句分配变量 items，然后第二个语句判断值的真假。如果 items 为假值，那么 ul 标记符不显示。 现在，假设当没有条目时并不是简单的删去列表，而是显示一条消息。要实现这一点，在列表前加入以下代码：

```
<h4 tal:condition="not:container/objectIds">There
Are No Items</h4>
```

当 container/objectIds 为假时，表达式 not:container/objectIds 为真，依次类推。参见本章后边的“Not 表达式”部分。此时你还不能使用变量 items，因为他还没有定义。如果你把 items 定义移到 h4 标记符中，那么你不能在 ul 标记符里使用，这是因为它变成了 h4 标记符的本地变量。你可以把定义放置在包括 h4 和 ul 标记符的其他标记符里，但是有一种简单的解决方法。通过在变量名称前放置关键词 global,你就可以在定义它的 h4 标记符到模板底部之间使用它。

```
<h4 tal:define="global items container/objectIds"
```

```
            tal:condition="not:items">There Are No Items</h4>
```

使用 tal:define 定义多个变量的方法是用分号分隔开，。例如：

```
        <p tal:define="ids container/objectIds;

                       title container/title">
```

你可以定义任意多个变量。每个变量可以有它自己的全局或本地范围。你还可以在后边定义里引用前边定义的变量。例如：

```
<p tal:define="title template/title;

               global untitled not:title;

               tlen python:len(title);">
```

其中 title 和 tlen 是本地变量，而 untitled 是全局变量。通过使用 tal:define，你可以增进模板的效率和可读性。

1.5. 忽略标记符

你可以删除带有 tal:omit-tag 语句的标记符。你会很少使用这个语句，但是有时候还是有用的。omit-tag 属性删除一个标记符，但不影响标记符内容。例如：

```
<b tal:omit-tag=""><i>this</i> stays</b>
```

执行后为：

```
<i>this</i> stays
```

这种用法，tal:omit-tag 操作就像是 tal:replace="default"。并且，tal:omit-tag 也可以与 true/false 表达式一起使用。只有表达式为真时才删除标记府。比如：

```
Friends: <span tal:repeat="friend friends">

  <b tal:omit-tag="not:friend/best"

     tal:content="friend/name">Fred</b>

</span>
```

这样就会产生一个列表，其中 best 用粗体显示。

1.6. 错误处理

如果在你的页面模板里发生了错误，你可以捕捉这个错误，并给用户显示一条有用的消息。例如，假设你的模板定义了一个使用表单数据的变量。

```
...
<span tal:define="global prefs request/form/prefs"
      tal:omit-tag="" />
...
```

如果 Zope 遇到了一个错误，比如在表单数据里不能找到 prefs 变量，整个页面将终止；而你将得到一个错误页面。值得庆幸的是，通过使用 tal:on-error 语句以及错误处理机制，你就可以避免这种事情。

```
...
<span tal:define="global prefs here/scriptToGetPreferences"
      tal:omit-tag=""
      tal:on-error="string:An error occurred">
...
```

当执行模板时引发了一个错误，那么 Zope 就会查找 tal:on-error 语句来处理这个错误。它先在当前的标记符里查找，然后在合拢的标记符里，就这样一直到顶级的标记符。当它找到一个错误处理器，它用错误处理表达式替换标记符内容。在这个例子里，span 标记符将包含一个错误消息。

一般情况下，你将对一个标记符定义错误处理器，其中包含逻辑页面元素，例如表格。如果一个错误影响了绘制表格，那么错误处理器可以从页面里忽略表格，或者用某种错误消息替换它。 对于更为灵活的错误处理，你可以调用脚本。例如：

```
<div tal:on-error="structure here/handleError">
...
</div>
```

任何发生在 div 里的错误将调用 handleError 脚本。注意 structure 选项允许脚本返回 HTML。你的错误处理脚本可以检测错误并且根据错误的类型采取不同的处理方法。方式是通过名称空间调用 error 变量。例如：

```
## Script (Python) "handleError"
```

```
##bind namespace=_

##

error=_['error']

if error.type==ZeroDivisionError:

    return "<p>Can't divide by zero.</p>"

else

    return """<p>An error occurred.</p>

              <p>Error type: %s</p>

              <p>Error value: %s</p>""" % (error.type,

                                           error.value)
```

你的错误处理脚本可以采取各种处理方法，例如，它可以通过发送邮件记录错误。 tal:on-error 语句不适合处理通常意义上的例外。例如，你不能用来验证表单输入。你应该使用脚本，这是因为脚本允许你完成强大的例外处理。tal:on-error 语句适合于处理执行模板时所发生的错误。

1.7. 在 TAL 语句之间交互

当每个元素中只有一个 TAL 语句时，执行的顺序是简单的。从 root 元素开始，执行每个的元素语句，然后访问每个下级元素按照这个顺序，执行他们的语句，依次类推。 可是，存在相同的元素拥有多个 TAL 语句的情况。除了 tal:content 和 tal:replace 语句不能结合在一起外，任何语句的结合都可能出现在相同的元素里边。 当一个元素有多个语句时，他们的执行顺序如下：

1. define
2. condition
3. repeat
4. content 或 replace
5. attributes
6. omit-tag

由于 tal:on-error 语句只有当发生错误时才出现，因此，它没有列出来。 采用这种顺序的原因是：因为一般先要设置其他语句里使用的变量，因此 define 放在第一位。接下来要做的事情是决定是否显示这个元素，因此 condition 放在其次，并且还由于 condition 可能依赖于刚才设置的变量，因此，它放在 define 后边。能够用每次循环的值替换元素的各个部分是很有价值的，因此 repeat 放在 content replace 和 attributes 前面。Content 和 replace 不能同时对同一元素应用，因此它们处于同一位置。Omit-tag 位于最后，这是因为没有其他的语句依赖于它，因此它应该位于 define 和 repeat 后边。 以下是一个包含多个 TAL 语句的例子：

```
<p tal:define="x /root/a/long/path/x | nothing"
   tal:condition="x"
   tal:content="x/txt"
   tal:attributes="class x/class">Ex Text</p>
```

注意 tal:define 语句是如何先被执行的，其他的语句依赖于它的结果。 当对元素结合 TAL 语句时，有三个应该知道的限制

对于单一的标记符，每一种语句只能应用一次。这是因为 HTML 不允许相同名称的属性出现多次出现。例如，你不能在同一标记符中出现两个 tal:define。

tal:content 和 tal:replace 不能同时在同一标记符中出现，这是因为它们的功能是相反的。

标记符中编写 TAL 属性的顺序不影响他们执行的顺序。不管你如何排列它们，TAL 语句执行总是按照上边所描述的固定顺序执行。

如果你打破 TAL 语句的顺序，你必须放在另外一个元素里。例如，假设你想对一个项目序列进行循环，但要跳过一些。下边的例子试图编写一个模板，它从 0 循环到 9，并跳过 3：

```
<!-- broken template -->
<ul>
  <li tal:repeat="n python:range(10)"
      tal:condition="python:n != 3"
      tal:content="n">
    1
  </li>
</ul>
```

这个模板不会工作，这是因为在执行重复以前先检测条件。以下是解决这个问题的一种方式：

```
<ul>
  <div tal:repeat="n python:range(10)"
       tal:omit-tag="">
    <li tal:condition="python:n != 3"
        tal:content="n">
      1
    </li>
  </div>
</ul>
```

实现方式是在 div 标记符中定义 n 变量。注意，由于存在 tal:omit-tag 语句，div 标记符不会在输出里显示。在 HTML 中可以使用 div 或 span 标记符，而在 XML 中，没有对应的标记符。此时，在 XML 中可以任意定义新的 tal 标记符。比如：

```
<tal:series define="items here/getItems">
  <tal:items repeat="item items">
  <tal:parts repeat="part item">
    <part tal:content="part">Part</part>
  </tal:parts>
  </tal:items>
  <noparts tal:condition="not:items" />
</tal:series>
```

这段代码中的 tal:series, tal:items, and tal:parts 标记符就是新定义的，可用来处理 XML 名称空间。和 div 比较，它有两个好处，第一，不通过 tal:omit-tag 就可以忽略这些标记符。第二，属性中不需要加上 tal 字样。

2. 表单处理

在DTML里，你可以使用一组“form/action”来处理表单。一组form/action由两个DTML方法或DTML文档组成：一个用于提供收集用户输入信息表单，另外一个包含可以处理输入并返回结果的行为。表单调用行为。 Zope页面模板对不是非常适合form/action模式，这是由于这种模式假设输入处理和回应是由同一种对象来完成的。对于页面模板，最好使用form/action/response模式。Form和response应该为页面模板，action应该为一个脚本。表单模板收集输入，然后调用回应脚本。回应脚本应该处理输入，然后返回一个回应模板。这种模式比form/action模式更为灵活，这是由于脚本能够返回任意数量不同类型的回应对象。

例如，以下是一个表单模板的一部分：

```
...
<form action="action">
  <input type="text" name="name">
  <input type="text" name="age:int">
  <input type="submit">
</form>
...
```

这个表单调用以下脚本：

```
## Script (Python) "action"
##parameters=name, age
##
container.addPerson(name, age)
return container.responseTemplate()
```

这个脚本调用一个方法来处理输入，然后返回另外一个模板。你可以通过Python调用模板来执行一个页面模板。回应模板一般应包含正确处理表单后的确认信息。 行为脚本可以完成各种任务。它可以检验输入，处理错误，发送邮件，以及其它任务等。以下是一个用脚本确认输入的例子：

```
## Script (Python) "action"
##
if not context.validateData(request):
```

```
            # if there's a problem return the form page template
            # along with an error message
            return context.formTemplate(error_message='Invalid data')
        # otherwise return the thanks page
        return context.responseTemplate()
```

这个脚本检验表单输入，并返回表单模板，如果存在问题就带有一个错误消息。你可以用关键字参数给页面模板传递额外的信息。关键字参数以内建变量选项的形式存在。因此这个例子里的表单模板可能会包括这样一部分：

```
<span tal:condition="options/error_message | nothing">
Error: <b tal:content="options/error_message">
  Error message goes here.
</b></span>
```

这个例子显示了如何通过关键字参数给模板传递一个要显示的错误消息。如果没有传递错误消息，就显示为空。

根据你的应用程序的不同，你可以选择让用户指向一个回应页面模板，而不是直接返回它。这样就进行了两次调用，但它的用途在于更改了显示在用户的浏览器中的URL,而不是回应脚本的URL。如果你坚持要采用原来的方式，你可以用页面模板创建一组form-action。但只有当你不关心错误处理时，才能这样做，并且回应应该总是一样，不管用户提交了什么。由于页面模板没有象dtml-call那样的等同语句，你可以使用任何一种方式来调用输入处理方法，且不插入结果。例如：

```
        <span tal:define="unused here/processInputs"
              tal:omit-tag=""/>
```

这个例子调用了processInputs方法，并且把结果分配给unused变量：

3. 表达式

你已经用过了页面模板表达式。表达式给模板语句提供值。例如，<td tal:content="request/form/age">Age</td>，其中的语句表达式是 request/form/age。这是一种路径表达式。在本节，将学习各种类型的表达式和变量。

3.1. 内建变量

变量就是你在表达式里可以用的名称。你已经在一些例子里看到内建变量，比如template, user, repeat, request。下边列出了是其他内建变量和用途，注意这些变量只适用于页面模板：

nothing

一个假值，类似于一个空字符串，你可以在 tal:replace 或者 tal:content 里用来删掉一个标记符或它的内容。如果你把一个属性设置为 nothing，这个属性就会从标记符中删除（或不插入），而不象空字符串。

default

当在 tal:replace, tal:content, 或者 tal:attributes 里使用时，是一个指定的值。它保持文本不变。

options

关键词参数，如果有的话，会传递给模板。注意：options 只有当从 python 里边调用模板时才存在。当模板从 web 执行时，没有 options。

attrs

模板里当前标记符的属性字典。键名为属性名称，键值为属性在模板里最初的值。这个变量很少用。

root

Root 对象。使用这个对象从某个固定的位置得到 Zope 对象，不管模板被放置在什么地方，或在什么地方调用。

here

模板被调用时的对象。这经常等同于 container，但如果你使用获取，则会不同。要在不同的地方找到你所要的 Zope 对象，这取决于模板是如何被调用的。Here 变量类似于基于 Python 的脚本的 context 变量。

container

保存模板的容器（一般是一个文件夹）。使用这个对象可以通过相对主目录的方式得到 Zope 对象。当从普通位置调用模板时，Container 和 Here 变量指的是同一对象。然而，当对另外一个对象应用模板时（例如，一个 ZSQL 方法）,这两个变量不再指同一对象。

modules

适用于模板的 Python 模块集。参见编写 Python 表达式部分。

在本章，你会看到使用这些变量的例子。

3.2. 字符串表达式

字符串表达式可以混合路径表达式和文本。所有以字符“string:”开始的表达式是字符串表达式。其中的每个路径表达式必须用一个'$'开头。下边是几个例子：

```
"string:Just text. There's no path here."

"string:copyright $year by Fred Flintstone."
```

如果路径表达式有多个部分，或者需要和文本部分分开，它必须用大括号('{}')括起来。例如：

```
"string:Three ${vegetable}s, please."

"string:Your name is ${user/getUserName}!"
```

注意，上边的例子是如何表达的。你需要用大括号把 vegetable 路径括起来，这样 Zope 就不会弄错。

由于文本是位于属性值里边，通过使用实体句法"，你才能在里边包括双引号。由于美元符号已经用来标志路径表达式因此要显示美元符号，就需要再附加两个美元符号('$$')。例如：

```
"string:Please pay $$$dollars_owed"

"string:She said, &amp;quot;Hello world.&amp;quot;"
```

一些复杂的字符串格式操作（比如搜索和替换或更改大小写），字符串表达市很难完成。对于这种情况，你应该使用 Python 表达式或脚本。

3.3. 路径表达式(Path Expressions)

路径表达式通过类似于 URL 的路径指向对象。所有的路径从已知对象开始（比如内建变量，repeat 变量，或者用户定义的变量），直到想要的对象。以下是一些例子。

```
template/title

container/files/objectValues

user/getUserName

container/master.html/macros/header

request/form/address

root/standard_look_and_feel.html
```

通过使用路径表达式，你可以访问对象和下级对象，以及它们的属性和方法。还可以在路径表达式里使用获取。请参见“高级 Zope 脚本”一章里边的“从 Web 调用脚本”部分。在路径表达式里，Zope 采用 URL 的方式访问对象。当然，访问对象的时候需要足够的权限。

3.3.1. 替代路径

当每次使用模板时，路径表达式 template/title 都会存在，尽管它可能是一个空字符串。一些路径，比如 request/form/x，在执行模板的过程中可能不存在。这样就会在确定路径表达式的值时引起错误。

当路径不存在，可使用替代路径或值。例如，如果 request/form/x 不存在，你可能想使用 here/x。实现这个功能，方式是按照引用的顺序列出来，并用束线符号('|')分开：

```
<h4 tal:content="request/form/x | here/x">Header</h4>
```

有两个变量作为替代变量时很有用，即 nothing 和 default。例如，default 告诉 tal:content 保持现有内容。不同的 TAL 语句对 default 和 nothing 有不同的解释。请参见“Zope 页面模板参考”部分。

在替代路径表达式里，你还可以使用非路径表达式，例如：

```
<p tal:content="request/form/age|python:18">age</p>
```

在这个例子里，如果 request/form/age 路径不存在，那么值就是 18。这种形式允许你指定默认值。注意，只能使用非路径表达式作为替代表达式。

你还可以用存在表达式直接检测路径是否存在。参见下边的“存在表达式”部分。

3.4. Not 表达式（Not Expressions）

Not 表达市可以对表达式求反，例如：

```
<p tal:condition="not:here/objectIds">

  There are no contained objects.

</p>
```

当表达式为假时，Not 表达式返回真，否则相反。在 Zope 里，不存在的变量，0，空字符串，空序列，nothing 和 None 被认为是假，其他的为真。不存在的路径即不是真也不是假，不能使用 not 表达式。对于 Python 表达式，不一定需要使用 Not 表达式使用 Python 中的 not 关键字就可以了。

3.5. Nocall 表达式

通常，路径表达式会执行表达式里所包含的对象。这就意味着，如果对象是一个函数，脚本，方法，或一些其他可执行的对象，那么表达式会调用这个对象，求得结果。一般来说，需要这样，但并总需要这样。例如，如果你想把一个 DTML 文档放入到一个变量里边，从而为了引用它的属性，你不能使用通常的路径表达式，这是因为它会把文档执行成字符串。

如果你在路径前边加上前缀 nocall:，就会阻止执行，只返回对象。例如：

```
<span tal:define="doc nocall:here/aDoc"
      tal:content="string:${doc/getId}: ${doc/title}">
Id: Title</span>
```

这种表达式类型另外一种用途是用在Python表达式里边，即在需要定义一个变量来处理函数或模块类的时候。

式除了对象，Nocall表达还可以是函数：

```
<p tal:define="join nocall:modules/string/join">
```

这个表达式把变量join变量定义为函数（'string.join'），而不是调用函数的结果。

3.6. Exists表达式

如果表达式的路径存在，则exists表达式为真，否则为假。例如，下面的代码，如果没有给request传递c参数，就显示显示错误：

```
<h4 tal:define="err request/form/errmsg | nothing"
    tal:condition="err"
    tal:content="err">Error!</h4>
```

你可以用exists表达式完成同样的功能：

```
<h4 tal:condition="exists:request/form/errmsg"
    tal:content="request/form/errmsg">Error!</h4>
```

你可以结合exists表达式和not表达式，例如：

```
<p tal:condition="not:exists:request/form/number">Please enter
a number between 0 and 5</p>
```

注意，在这个例子里，你不能使用表达式："not:request/form/number"，这是由于如果变量number变量存在并且为0，那么表达式将为真。

3.7. Python 表达式

Python 编程语言简单而富有表现力。如果你以前从没有用过，可以读一读 Python 站点中的介绍或说明。 Python 表达式可以包括任何 Python 语言认为是表达式的内容。你不能使用象 if 和 while 这样的语句。此外，Zope 还对访问受保护的信息、更改安全数据和创建无限循环这样的错误进行一些安全限制。更多信息，请参见“高级 Zope 脚本”里有关 Python 安全限制部分。

3.7.1. 比较

Python 表达式特别有用的一种场合是在 tal:condition 语句里。在你需要比较两个字符串或数字时，只能用 Python 表达式来完成。你可以使用比较操作符<（小于）、>（大于）、==（等于）和!=（不等于）。还可以使用布尔操作符 and, not 和 or。例如：

```
<p tal:repeat="widget widgets">
  <span tal:condition="python:widget.type == 'gear'>
  Gear #<span tal:replace="repeat/widget/number>1</span>:
  <span tal:replace="widget/name">Name</span>
  </span>
</p>
```

这个例子对一个对象集合进行循环，测试每个对象的 type 属性。 有些时候，你需要基于一个或多个条件在一条语句里选择不同的值。你可以通过 test 函数来完成这个功能，就象这样：

```
You <span tal:define="name user/getUserName"
     tal:replace="python:test(name=='Anonymous User',
                              'need to log in', default)">
      are logged in as
      <span tal:replace="name">Name</span>
    </span>
```

Test 函数就象 if/then/else 语句。关于 test 函数方面更多信息，请参见“DTML 参考”。这里是另外一个使用 test 函数的例子：

```
<tr tal:define="oddrow repeat/item/odd"
    tal:attributes="class python:test(oddrow, 'oddclass',
```

'evenclass')">

没有 test 函数，你只能写两个适合不同条件的 tr 元素，一个用于偶数行，另外一个用于奇数行。

3.7.2. 使用其它的表达式类型

你可以使用 Python 表达式里边的其它表达式类型。每个表达式类型都有对应的同名函数，包括：path(), string(), exists()和 nocall()。这样就允许你编写下边的表达式：

```
"python:path('here/%s/thing' % foldername)"

"python:path(string('here/$foldername/thing'))"

"python:path('request/form/x') or default"
```

比起 path 表达式"request/form/x | default"，最后一个例子有一些略微不同的含义，这是由于如果"request/form/x"不存在或为假时，将使用默认的文字。

4. 使用 Zope 对象

Zope 之所以强大，相当多的表现在可以结合多种特殊的对象。你的页面模板可以使用 Scripts, SQL Methods, Catalogs,以及定制的内容对象。为了使用这些对象，你必须知道如何在页面模板里访问它们。

对象属性即通常的属性，因此你可以用表达式"template.title"得到模板的标题。大多数 Zope 对象支持获取机制，获取机制允许你从父对象得到属性。这就意味着 Python 表达式"here.Control_Panel"将从 root 文件夹获取 Control_Panel 对象。对象方法是属性，就像"here.objectIds" 和 "request.set"。文件夹里的对象能够以文件夹属性的方式访问，但是它们的 id 常常不是有效的 Pytho 标识，你不能使用通常的表示法。例如，你不能使用以下的表达式：

```
"python:here.penguin.gif"'.
```

你必须写为：

```
"python:getattr(here, 'penguin.gif')"
```

这是由于 Python 不支持用点号隔开的属性名称。 一些对象，比如 request, modules, 以及 Zope 文件夹支持 Python 访问。例如：

```
request['URL']

modules['math']

here['thing']
```

当你访问文件夹中的条目，它并不试图获取名称，它只有在文件夹里确实存在具有这个 Id 的对象时才会成功。

如前几章所显示的，path 表达式允许你忽略一些细节。Zope 试着先访问属性，然后是访问条目。你可以写成：

```
"here/images/penguin.gif"
```

而不是：

```
"python:getattr(here.images, 'penguin.gif')"
```

又如：

```
"request/form/x"
```

而不是：

```
"python:request.form['x']"
```

Path 表达式不允许你指定这些细节。例如，如果你一个名为 get 的表单变量，你必须写成：

```
"python:request.form['get']"
```

这是由于这个 path 表达式：

```
"request/form/get"
```

将对表单字典求 get 方法的值。如果确实想这样的话，你可以在 Python 表达式里通过 path()函数使用上边所述的 path 表达式。

5. 使用脚本

脚本对象经常用来处理事务逻辑和复杂的数据控制。当你发现编写 TAL 语句的时候采用了复杂的表达式时，就应该考虑是否用脚本来完成工作。如果你觉得难以理解你的模板语句和表达式，最好就用脚本来简化页面模板。

每个脚本都有一个参数列表，当调用脚本时应该提供这些参数。如果这个列表为空，那么你可以通过编写 path 表达式来使用这个脚本。否则，为了提供参数，就需要使用 Python 表达式，象这样：

```
"python:here.myscript(1, 2)"
```

```
"python:here.myscript('arg', foo=request.form['x'])"
```

如果你想通过脚本给页面模板返回多个数据，一种好的方法是以字典的形式返回。这样，你可以定义变量来存储所有的数据用 path 表达式来引用每个条目。例如：假设 getPerson 脚本返回一个含有 name 和 age 键的字典：

```
<span tal:define="person here/getPerson"
```

```
                tal:replace="string:${person/name} is ${person/age}">

            Name is 30</span> years old.
```

当然，还可以返回 Zope 对象和 Python 列表。

6. 调用 DTML

不象脚本，DTML 方法和文档没有明显的参数列表声明。但是，它们希望传递 client，映射，以及关键字参数。它们使用这些参数来构建名称空间。

当 Zope 通过 Web 公布一个 DTML 对象时，它把对象相关参数作为 client，REQUEST 作为映射。当一个 DTML 对象调用另外一个它把自己的名称空间作为映射传递，没有 client。

如果你使用 path 表达式来调用一个 DTML 对象，它将传递一个带有 request，here 和模板变量的名称空间。这就意味着 DTML 对象如果发布在和模板一样的地方，则使用相同的各种名称，以及模板里定义的变量名称。比如，下面的模板使用 DTML 生成 JavaScript ：

```
<head tal:define="items here/getItems.sql">

  <title tal:content="template/title">Title</title>

  <script tal:content="structure here/jsItems"></script>

</head>

...etc...
```

DTML 方法 jsItems 为：

```
<dtml-let prefix="template.id">

<dtml-in items>

&dtml-prefix;_&dtml-name; = &dtml-value; ;

</dtml-in>

</dtml-let>
```

这段 DTML 使用了模板的 id 和 items 变量。

7. Python 模块

Python 语言带有大量的模块，这些模块提供了很多功能。每个模块都是 Python 函数、数据和具有某一用途的类的集合，比如数学计算或规则表达式。

其中的一些模块，包括"math" 和 "string"，在 Python 表达式里默认存在。例如，你可以从 math 模块里得到 pi 的值，写成"python:math.pi"。要通过 path 表达式访问它，你就需要使用模块变量，写成"modules/math/pi"。

String 模块在 Python 表达式里是隐藏的，因此需要通过模块变量进行访问。你可以直接用在表达式里，或者定义一个全局变量，就像这样：

```
tal:define="global mstring modules/string"

tal:replace="python:mstring.join(slist, ':')"
```

实际上，很少需要这样做，这是因为大部分时候都可以直接使用 string 模块，而不必依赖 string 模块里的函数。

模块可以组合成包，这是一种组织和命名相关模块的简单方式。例如，Zope 里边基于 Python 的脚本是通过一组模块提供的，这组模块位于 Products 包里的 PythonScripts 子包。通常，这组模块里边的"standard"模块提供一些有用的格式函数，就像在 DTML 里的"var"标记符那样。这个模块的完整名称是"Products. PythonScripts .standard"，因此你可以使用以下任何一种语句访问它：

```
tal:define="global pps modules/Products/PythonScripts/standard"

tal:define="global pps python:modules['Products.PythonScripts.standard']"
```

许多 Python 模块不能从页面模板，DTML 或脚本进行访问，除非你给它们添加了 Zope 安全权限。这个过程超出了本书的范围请参见《Zope 开发指南》。

8. Macros(宏)

到此为止，你已经看到了页面模板如何给独立的 web 页面加入动态的行为。页面模板的另外一个特性是许多页面可以重复使用外观和风格元素。 例如，使用页面模板，网站就可以有一致的外观和风格。不管页面的内容是什么，都将有一致的页眉，按钮条，页脚，以及其它的页面元素。对于 web 站点来说，这是一种非常普遍的要求。

通过使用 macros，你可以在多个页面里重复使用表现元素。Macros 定义了多个页面之间共性的部分。一个 macro 可以为整个页面，或者仅为页面的一部分，比如页眉或页脚。当你在一个页面模板里边定义一个或多个 macro 以后，就可以在其他页面模板里边使用它们。

8.1. 使用 macro

你可以通过类似于 TAL 语句的标记符属性来定义 macro。Macro 标记符属性被称为 macro 扩展标记符属性语句（Macro Expansion Tag Attribute Language (METAL)）。以下是一个定义 macro 的例子：

```
<p metal:define-macro="copyright">
```

```
  Copyright 2001, <em>Foo, Bar, and Associates</em> Inc.

</p>
```

其中的metal:define-macro语句定义了一个名为"copyright"的macro。这个macro由p 和内容（包括所有被标记符括起来的部分）组成。

在页面模板里定义的macro存储在模板的macro属性里边。通过指向在其他模板里定义的macro属性，你可以调用macro。例如，假设copyright这个macro位于一个名为"master_page"的页面模板里边，以下显示了如何在另外一个页面模板里调用这个macro：

```
<hr>

<b metal:use-macro="container/master_page/macros/copyright">

  Macro goes here

</b>
```

在这个页面模板里，当Zope执行这个页面时，b标记符将完全用macro替换：

```
<hr>

<p>

  Copyright 2001, <em>Foo, Bar, and Associates</em> Inc.

</p>
```

如果你更改了macro（例如，名称变了），那么使用了这个macro的页面模板都会自动反映出这种变化。 通过使用metal:use-macro语句，在path表达式里调用了Macro。metal:use-macro语句用指定的macro替换内容。

8.2. Macro细节

metal:define-macro 和 metal:use-macro语句还是相当易用的，但是有一些注意事项：

Macro的名称在其被定义的页面模板里边必须是唯一的。你可以在一个模板里定义多个macro，但他们的名字不能相同。

一般通过在路径表达式里用metal:use-macro语句来调用一个macro。然而，只要表达式返回一个macro, 就可以使用任何类型的表达式。比如：

```
<p metal:use-macro="python:here.getMacro()">

  Replaced with a dynamically determined macro,
```

```
  which is located by the getMacro script.
</p>
```

使用 Python 表达式来定位 macro，可以让你动态的确定模板使用那一个 macro。

metal:use-macro 语句中可使用 default 变量：

```
<p metal:use-macro="default">
  This content remains - no macro is used
</p>
```

这个结果等同于在 tal:content 和 tal:replace 语句中使用 default。

如果在 metal:use-macro 中使用 nothing 变量，会得到一个错误，这是由于 nothing 不是一个 macro。如果你想使用 nothing，应该用 tal:condition 语句。

Zope 执行模板时会先处理 macros，然后对 TAL 表达式求值。例如，看以下的这个 macro:

```
<p metal:define-macro="title"
   tal:content="template/title">
  template's title
</p>
```

当你使用这个 macro,它将插入使用 macro 的那个模板的标题，而不是定义 macro 的模板的标题。换句话说，当你使用一个 macro，就像是把 macro 的文字复制到模板里，然后执行你的模板。

如果你选中了页面模板在 Edit 视图里的 Expand macros when editing 选项，那么你使用的任何 macro 都将在模板源文件里展开。当在管理界面中编辑时，而不是使用 WYSIWYG 编辑工具，不选择这个选项会更方便。新创建的页面模板默认不使用这个特性

8.3. 使用 slot(内容块)

当你使用 macro 时如果能够定制其中的某一部分，macro 就更显得有用了。实现这个功能，可以通过在 macro 里定义 slots 的方式实现，这样当你使用模板时就可以填充它。例如，考虑一个栏目条 macro：

```
<p metal:define-macro="sidebar">
  Links
```

```
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/products">Products</a></li>
      <li><a href="/support">Support</a></li>
      <li><a href="/contact">Contact Us</a></li>
    </ul>
  </p>
```

这个 macro 不错，但假设你希望在某些页面里的栏目条里加入一些附加信息。实现这个功能的一种方式是：

```
  <p metal:define-macro="sidebar">
    Links
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/products">Products</a></li>
      <li><a href="/support">Support</a></li>
      <li><a href="/contact">Contact Us</a></li>
    </ul>
    <span metal:define-slot="additional_info"></span>
  </p>
```

当你使用这个 macro，你可以这样来填充 slot:

```
  <p metal:fill-slot="container/master.html/macros/sidebar">
    <b metal:fill-slot="additional_info">
      Make sure to check out our <a href="/specials">specials</a>.
```

```
    </b>
  </p>
```

当你执行这个模板，栏目条会包含新加的信息：

```
  <p>
    Links
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/products">Products</a></li>
      <li><a href="/support">Support</a></li>
      <li><a href="/contact">Contact Us</a></li>
    </ul>
    <b>
      Make sure to check out our <a href="/specials">specials</a>.
    </b>
  </p>
```

注意定义 slot 的 span 元素是如何填充 b 元素的。

9. 定制默认的外观

Slot 的常见用途是为可以定制的页面提供默认的外观。在上一部分里边的 slot 例子里，slot 定义仅是一个空的 span 元素。然而，你可以在 slot 定义里提供默认的外观。例如，看以下这个修改过的栏目条 macro:

```
  <div metal:define-macro="sidebar">
    <p metal:define-slot="links">
    Links
    <ul>
```

```
      <li><a href="/">Home</a></li>
      <li><a href="/products">Products</a></li>
      <li><a href="/support">Support</a></li>
      <li><a href="/contact">Contact Us</a></li>
    </ul>
    </p>
    <span metal:define-slot="additional_info"></span>
  </div>
```

现在，这个栏目条可以充分定制。你可以填充 links slot 来重新定义工具条链接。然而，如果你不填充 links slot，那么，你将得到默认的链接。 你甚至可以通过在 slots 里定义 slots，进一步使用这种技巧。这样就可以让你覆盖默认的外观。以下是一个在 slot 内部定义 slot 的栏目条：

```
  <div metal:define-macro="sidebar">
    <p metal:define-slot="links">
    Links
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/products">Products</a></li>
      <li><a href="/support">Support</a></li>
      <li><a href="/contact">Contact Us</a></li>
      <span metal:define-slot="additional_links"></span>
    </ul>
    </p>
    <span metal:define-slot="additional_info"></span>
```

```
</div>
```

如果你想定制栏目条链接，你可以填充 links slot 来完全覆盖现有的链接，还可以填充 additional_links slot 来在默认链接后边插入一些额外的链接。你可以任意嵌套 slot。

10. 混合 METAL 和 TAL

在相同的元素里可以同时使用 METAL 和 TAL。例如：

```
<ul metal:define-macro="links"
    tal:repeat="link here/getLinks">
  <li>
    <a href="link url"
       tal:attributes="url link/url"
       tal:content="link/name">link name</a>
  </li>
</ul>
```

这是由于 METAL 语句在 TAL 语句前执行，不会冲突。并且这个例子没有使用 slot 来定义 macro。Macro 调用 getLinks 脚本来决定 links。这样你就可以在站点里不同部分重新定义 getLinks 脚本来定制站点的链接。 有些时候要想采取最好的方式来定制站点里不同部分的外观和感觉并不总是件容易的事情。通常，你应该使用 slots 来覆盖外观元素，并且使用脚本动态提供内容。在上边的例子里，需要确定链接是内容还是外观。通常脚本提供了一种更为灵活的解决方式，特别是如果你的站点包含链接内容对象

11. 全页面 Macros

使用 macros 不适合在页面之间共享外观元素，你却可以使用 macros 来定义整个页面。Slots 可以帮助实现这个功能。以下是一个定义整个页面的 macro：

```
<html metal:define-macro="page">
  <head>
    <title tal:content="here/title">The title</title>
  </head>
```

```
 <body>
   <h1 metal:define-slot="headline"
       tal:content="here/title">title</h1>
   <p metal:define-slot="body">
     This is the body.
   </p>
   <span metal:define-slot="footer">
     <p>Copyright 2001 Fluffy Enterprises</p>
   </span>
 </body>
</html>
```

这个macro定义一个带有三个slots的页面，即headline，body，和 footer。注意headline slot是如何通过一个TAL语句来动态确定headline内容。你可以在页面里使用这个macro来表现不同类型或地方的内容。例如，以下是一个模板演示了如何使用这种macro来表现新闻条目：

```
<html metal:use-macro="container/master.html/macros/page">
  <h1 metal:fill-slot="headline">
    Press Release:
    <span tal:replace="here/getHeadline">Headline</span>
  </h1>
  <p metal:fill-slot="body"
     tal:content="here/getBody">
    News item body goes here
  </p>
```

```
</html>
```

这个模板重新定义了 headline slot，从而包含了单词"Press Release"，并且对当前对象调用 getHeadline 方法。他还重新定义了 body slot ，从而对当前对象调用 getBody 方法。 这种方法强大的一方面在于你可以更改页面 macro，这样 press release 模板就会自动更新。例如，你可以把页面正文放入表格里，并在左边加入一个栏目条，press release 模板就会自动使用这些新的外观元素。 比起 DTML 里边的 standard_html_header 和 standard_html_footer 方法，这是一种更为灵活的解决方式。事实上，Zope 在 root 文件夹里带有一个名为 standard_template.pt 的页面模板，它包含一个带有 head 和 body slot 的页面 macro。以下显示了如何在模板里使用这个 macro：

```
<html metal:use-macro="here/standard_template.pt/macros/page">
  <div metal:fill-slot="body">
    <h1 tal:content="here/title">Title</h1>
    <p tal:content="here/getBody">Body text goes here</p>
  </div>
</html>
```

使用 standard_template.pt macro 非常类似于使用其它的全页面 macros。唯一需要说明的是用来定位 macro 的路径。在这个例子里，路径用 here 开始。这样，Zope 就会使用获取机制来搜索 standard_template.pt 对象，从应用模板的那个对象所处位置开始。这样允许你通过在各个位置创建特定的 standard_template.pt 对象来定制外观和感觉。这种方法就像覆盖站点不同位置的 standard_html_header 和 standard_html_footer 来定制外观和感觉。然而，使用 standard_template.pt，你可以有更多的选择你可以像采用 here 那样，对 root 或 container 对这个 macro 起用路径。如果路径由 root 开始，那么你将总是得到位于 root 文件夹里边的标准模板。如果路径由 container 开始，那么 Zope 将通过使用获取机制搜索标准模板，开始的文件夹是定义模板的文件夹。这样允许你定制模板的外观和感觉，但不允许你定制不同位置的对象的外观和感觉。

12. 缓存模板

通常执行页面模板是相当快的，有时不是足够快。对于经常访问的页面或需要长时间执行的页面，你需要为提高速度牺牲掉一些动态行为。缓存可以帮助你实现这些。

你可以像缓存其他对象一样使用缓存管理器来缓存页面模板。要缓存页面模板，你必须用一个缓存管理器关联它。实现这点你可以进入模板的 Cache 视图，并选择缓存管理器，还可以进入缓存管理器的 Associate 视图，然后选择你的模板。 以下是一个如何缓存页面模板的例子。首先，创建一个基于 Python 的脚本 long.py ，内容如下：

```
## Script (Python) "long.py"
##
for i in range(500):
```

```
    for j in range(500):
      for k in range(5):
        pass
  return 'Done'
```

这个脚本的目的是要让人感觉到执行时间。现在，创建一个使用这个脚本的页面模板，例如：

```
<html>
  <body>
    <p tal:content="here/long.py">results</p>
  </body>
</html>
```

现在观看这个页面，注意它花费了一些执行时间。现在，让我们用缓存来提高执行速度。先创建一个 Ram Cache Manager。并确定它创建在与页面模板相同的文件夹里，或者在更高一级的目录里。现在观看 Cache 视图。选择刚才创建的 Ram Cache Manager，然后点击 Save Changes。单击 Cache Settings 可查看配置情况。默认的情况下，缓存内的对象存储时间为一个小时(3600 秒)。根据你的应用程序的需要，你可以调整这个数字。现在返回你的页面模板，再次观看它。这个过程会花费点时间来处理现在重新调用这个页面，就会发现立刻就出来了。你可以一次又一次的调用这个页面，它总是迅速的出来，这就是缓存的作用。

如果你更改页面模板，它就会从缓存里删除。因此，下一次观看它时，就会花费点时间来处理。当它再次被存入缓存后，就会快了。 缓存是一种简单而非常强大的增进性能的技术。使用缓存很容易，并且可以明显的提高速度。那些对性能要求高的应用程序，使用缓存是非常值得的。

13. 页面模板工具

Zope 页面模板强大而简单。它不象 DTML，页面模板不提供一些便利的特性，比如批块处理，绘制树结构，排序等等。页面模板的创建者想让它简单。这样的话，就可能失去一些 DTML 提供的内建特性。为了满足这些需要，Zope 提供了一些用于加强页面模板的工具。

13.1. 批块化处理巨大的信息集合

当一个用户查询数据库，得到上百个结果，比起在一个页面里显示这些结果，有一种通常更好的显示方式，那就是采用多个每页只显示 20 个结果的页面。把巨大的列表分割为多个小的列表就被称为批块化。 页面模板不象 DTML 那样把批块化功能内建到语句里边，页面模板支持批块化是通过使用一种特殊的批块对象，这种对象由 ZTUtils 工具模块提供。有关这个模块的更多信息请参见“API 参考”一章。 以下是一个简单的例子，显示了如何创建一个批块对象（Batch object）：

```
<ul tal:define="lots python:range(100);
```

```
                    batch python:modules['ZTUtils'].Batch(lots,
                                                          size=10,
                                                          start=0)">
      <li tal:repeat="num batch"
          tal:content="num">0
      </li>
    </ul>
```

这个例子按照每次 10 个条目来处理列表（数字从 0 到 9）。批块对象把一个长的列表转化为群组或批块。在这个例子里，是把含有一百个条目的列表分成多个包含 10 个条目的批块。 你可以通过传递一个不同的起始数字来显示不同的批块：

```
    <ul tal:define="lots python:range(100);
                    batch python:modules['ZTUtils'].Batch(lots,
                                                          size=10,
                                                          start=13)">
```

这个批块从第 14 个条目开始，并以第 23 个条目结束。换句话说，它显示的数字是从 13 到 22。需要注意的是批块的 start 参数是第一个条目的索引（index）。索引从 0 开始计算，而不是从 1。所以，索引 13 代表的是序列里边第 14 个条目。Python 使用索引来指向列表条目。 通常，当你使用批块时，你需要使用导航元素来让用户在批块之间进行跳转。以下是一个说明如何在多个批块之间进行导航的例子：

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
  </head&
```

第十四章 高级 Zope 脚本

Zope 通过对象管理表现（presentation）、逻辑（logic）和数据（data）。目前为止，你已经看到了如何通过文件和图像管理数据，以及使用 DTML 和页面模板管理表现。本章给你显示如何添加脚本对象，使你能够通过 Web 浏览器用 Python(http://www.python.org)编写脚本。

什么是逻辑，并且它如何区别于表现？逻辑可用于处理修改对象、发送消息、测试条件和事件反应。而表现则用来格式化信息和提供报告，并显示它们。典型的情况是，使用 DTML 或页面模板处理表现，用 Python 编写 Zope 脚本来处理逻辑。

1. Zope 脚本

Zope 脚本对象包含有用某种程序语言编写的代码。当前，Zope 主要提供基于 Python 语言的脚本，它用 Python 语言编写，在 Zope 里是编写程序逻辑的首选方式。

以下是 Zope 脚本介绍：

基于 Python 的脚本

你可以使用 Python 语言（一种通用脚本语言）控制 Zope 对象和执行其它任务。

外部方法

采用 Python 编写，但是代码存储在文件系统中。外部方法可以完成脚本不能完成的一些功能，比如脚本中那些受到安全限制的功能。

基于 Perl 的脚本

你可以使用 Perl（一种强大的文本处理语言）来编写 Zope 对象脚本，并访问 Perl 程序库。Perl 脚本提供与基于 Python 的脚本相类似的好处。Perl 对于那些知道 Perl 而不知道 Python 的人们来说更具有吸引力——或者是对于那些因没有相应的 Python 程序库而想使用 Perl 程序库的人们。要使用 Perl，需要安装相应的扩展模块。

你可以象其它任何对象那样给你的 Zope 应用程序添加这些脚本。

2. 调用脚本

Zope 脚本可以通过 Web 或其它脚本或对象调用。几乎任何类型的脚本都可以被其它对象调用。你可以通过一个 DTML 方法调用基于 Python 的脚本或通过基于 Perl 的脚本调用一个内建方法。事实上，脚本可以调用脚本，而这个脚本又调用其它脚本，如此等等。就像你在以前章节中看到的，你可以直接地把一个脚本替换成一个用其它语言编写的脚本。例如，如果你正在使用 Perl 执行一个任务，但是后来觉得用 Python 完成会好些，你就可以用一个具有相同 id 的基于 Python 的脚本替换这个脚本。

2.1. 环境（Context）

当你调用脚本，一般都需要指出相关的其它对象。这些对象能够提供所需的信息或脚本要操作这些对象。按照面向对象的说法，是以这些对象的方法的形式调用脚本。但在传统的面向对象编程中，是执行在类中定义的方法。脚本如何能够担当许多对象的方法，并且不是在对象类中进行定义的？

在“获取机制”一章中，我们已经学习了 Zope 可以通过获取机制从上级容器中找到所需的对象。获取机制可以让我们在当前对象的环境中以对象的一个方法的形式调用脚本，只需要构建合适的 URL。调用脚本时所针对的对象给脚本提供了一个执行环境。简单的可以说成是针对当前对象调用脚本。也可以说，环境(context)就是脚本执行时所处的环境，从这个环境中脚本才能够找到所需信息。

也可以这样来理解脚本的环境，那就是把脚本看作是编程语言中的函数，把环境看作是这个函数的隐形参数。 通过两种方式可以调用脚本并提供环境：通过访问 URL 和通过其它脚本或模板调用脚本。

2.1.1. 通过 Web 调用脚本

你可以在 Web 浏览器中通过访问脚本的 URL 来直接调用脚本。你可以通过使用不同的 URL 对不同的对象调用同一脚本。之所以可以这样做是因为使用不同的 URL，可以决定脚本环境。这是一个强大的特性，它使得你能够在对象上应用逻辑，例如文档或者文件夹这样的对象，而无需让对象嵌入实际的代码。

要通过 Web 对一个对象调用一个脚本，只需简单的访问 URL（对象名称后跟随脚本名称的 URL）。这样就把脚本与对象关联在一起。例如，假设你有一些对象和脚本，如下图所示：

一些对象和脚本

要对 hippo 对象调用 feed 脚本，你访问 URL：Zoo/ LargeAnimals /hippo/feed。要对 kargarooMouse 对象调用 feed 脚本，你访问 URL：Zoo/ SmallAnimals /kargarooMouse/feed。这些 URL 把 feed 脚本放在 hippo 和 kargarooMouse 各自对象环境中。

Zope 把 URL 看成是一种映射，从而确定对象和脚本。

2.1.2. URL 漫游和获取

Zope 分解 URL 并把它和对象层级进行比较，向上查找，直至找到每个部分所对应的对象。这个过程被称为 URL 漫游。例如，当你使用 URL：Zoo/ LargeAnimals /hippo/feed，它从根文件夹开始并查找一个名为 Zoo 的对象。然后，它移到 Zoo 文件夹并查找名为 LargeAnimals 的对象。再移到 LargeAnimals 文件夹并查找名为 hippo 的对象。再移到 hippo 对象并查找名为 feed 的对象。feed 脚本不能在 hippo 对象中被找到，因此它通过获取（Acquisition）的过程在 Zoo 文件夹中找到它。它试图在当前对象容器中查找对象。如果那样不行，它就沿着 URL 路径向上返回并再试一次。在这个例子中，Zope 首先在 hippo 对象中查找 feed 对象于是它进入第一个容器 LargeAnimals ，然后进入下一个容器 Zoo，最终找到 feed。

现在 Zope 已经到达了 URL 末尾，并且每个名称都找到了对应的对象。Zope 找到了最后的 feed，并发现它可以调用，于是针对 hippo 对象进行操作。这就是 feed 脚本如何对 hippo 对象进行调用的。

同样，你可以对 hippo 通过 URL zoo/ LargeAnimals /hippo/wash 调用 wash 方法。在这种情况下，Zope 从 LargeAnimals 文件夹获取 wash 方法。

2.1.3. 通过 HTTP 查询字符串传递参数

通过 URL 可以传递参数，比如：

```
http://my-zope-site:8080/Zoo/LargeAnimals/hippo/wash?soap=lye
```

3. 通过其它对象调用脚本

你可以通过其它对象调用脚本，这些对象可以是 DTML 对象，页面模板或脚本等等。

3.1. 通过 DTML 调用脚本

就像在“高级 DTML”中看到的，你可以使用 DTML 中的 call 标记符调用脚本，比如：

```
<dtml-call updateInfo>
```

DTML 将调用 updateInfo 脚本，不管脚本是用什么语言实现的。

如果 updateInfo 脚本需要参数，你可以为 DTML 名称空间绑定（见本章后边的“绑定变量”部分）选择一个名称，这样就会在名称空间中查找参数。你还可以在一个表达式中传递参数，就像这样：

```
<dtml-call expr="updateInfo(color='brown', pattern='spotted')">
```

你还可以传递在 DTML 名称空间中有效的任何变量。例如，假设使用 dtml-let 语句定义了 newColor 和 newPattern 变量，你就可以这样传递参数：

```
<dtml-call expr="updateInfo(color=newColor, pattern=newPattern)">
```

你还可以传递 dtml 标记符中已经定义好的变量，比如对于 dtml-in：

```
<dtml-in all_animals prefix="seq">

  <dtml-call expr="feed(animal=seq_item)">

</dtml-in>
```

上边的代码中，假设feed是脚本，并且有一个变量animal。通过加上“seq_”前缀，引用了dtml-in标记符中的sequence-item变量。

3.2. 通过其它脚本调用脚本

就像DTML一样，通过其它Python脚本或Perl脚本也可以调用脚本对象。此时必须传递参数。例如，以下通过Python脚本调用updateInfo脚本：

```
new_color='brown'

context.updateInfo(color=new_color,

                   pattern="spotted")
```

其中，通过使用context变量，告诉Zope使用获取机制查找updateInfo。

使用Perl时，可以这样：

```
$new_color = 'brown';

$self->updateInfo(color => $new_color,

                  pattern => "spotted");
```

这里可以看到self用来表示当前的context。

Zope使用获取来定位脚本。这种获取和通过Web调用脚本时所采用的获取相同。返回我们前面一节中的hippo feed例子。下图显示了一个简略的对象层级，它包含两个脚本vaccinateHippo.py 和 vaccinateHippo.pl。

一些对象和脚本

假设 vaccinateHippo.py 是一个 Python 脚本。以下显示了 vaccinateHippo.py 脚本中如何对 hippo 对象调用 vaccinate 脚本

```
context.Vet.LargeAnimals.hippo.vaccinate()
```

实际上，它所使用的获取路径和通过 Web 调用它时所使用的获取路径相同。结果等同于访问 Zoo/Vet/ LargeAnimals /hippo/vaccinate 。注意，这个例子中，不需要在 Vet 前边指出 Zoo。这是因为所有的对象都位于 Zoo 文件夹。

同样，使用 Perl 时，可以这样：

```
$self->Vet->LargeAnimals->hippo->vaccinate();
```

3.3. 通过页面模板调用脚本

通过页面模板调用脚本非常类似于 URL 方式或 Python 方式。只要使用 TALES 路径表达式就可以了，比如：

```
<div tal:replace="here/hippo/feed" />
```

可以参考“高级页面模板”一章。

页面模板中没有和 DTML 中对应的 call 标记符，因此需要不把结果插入到页面里时，可以使用 define 语句，比如：

```
<div tal:define="dummy here/hippo/feed" />
```

在页面模板中，here 表示当前环境，类似于脚本对象中的 context 变量。

如果需要传递参数，则需要使用 python 表达式，比如：

```
<div tal:replace="python:here.hippo.feed(food='spam')" />
```

上边的Python表达式就像一行Python脚本对象中的一行代码。不同之处在于它使用here变量来表示context，作用是一样的。在将来的版本中，有可能会把名称统一成context。

3.4. 调用脚本：总结和比较

让我们总结以下调用脚本时用到的不同形式：

通过 URL:

```
http://my-zope-server.com:8080/foo/updateInfo?amount=lots
```

通过 Python 脚本:

```
context.foo.updateInfo(amount="lots")
```

通过 Perl 脚本:

```
$self->foo->updateInfo(amount="lots");
```

通过页面模板:

```
<span tal:content="dummy here/foo/updateInfo"/>
```

通过页面模板，并传递参数:

```
<span tal:content="python:here.foo.updateInfo(amount='lots')"/>
```

通过 DTML:

```
<dtml-with foo >
 <dtml-var updateInfo>
</dtml-with>
```

通过 DTML, 并传递参数:

```
<dtml-with foo>
 <dtml-var expr="updateInfo(amount='lots')">
</dtml-with>
```

DTML 中的另外一种变形：

```
<dtml-var expr="_['foo'].updateInfo()">
```

如果 Zope 找不到所调用的脚本，就会引发一个错误。那么则需要使用以下的方式：

```
updateInfo = getattr(context, "updateInfo", None)
if updateInfo is not None:
    updateInfo(color="brown", pattern="spotted")
else:
    # complain about missing script
```

getattr 函数是 Python 内置的函数。第一个参数指定一个对象，第二个参数指定属性名称。getattr 函数会返回指定的属性，如果找不到就返回第三个参数。

4. 使用基于 Python 的脚本

上边已经看到了一些例子，接下来让我们仔细进行讲解。

4.1. Python 语言

Python（ http://www.python.org/）是一种高级的面向对象的脚本语言。大部分 Zope 都是用 Python 编写的。许多人喜欢 Python 是因为 Python 的清楚明了、简单易用和可扩展至大型项目的能力。

有许多学习 Python 的资源。Python.org Web 站点有许多 Python 文档，其中包含有一个由 Python 的创建者 Guido van Rossum 编写的指南。

Python 自带有丰富的模块和软件包。你可以在 Python.org 站点找到更多的有关 Python 标准程序库方面的信息。

4.1.1. 创建基于 Python 的脚本

要创建一个基于 Python 的脚本，从产品添加列表中选取 Script (Python)。给脚本命名为 hello，然后单击 Add and Edit 按钮。你现在应该看到脚本的 Edit 视图。

这个屏幕允许你控制脚本的参数和正文。你可以在 Parameter List 字段中输入脚本的参数。在屏幕底部的文本区中键入脚本正文。

在 Parameter List 字段中输入 name="World"，并在脚本正文中键入

```
return "Hello %s." % name
```

上边的例子等同于以下标准Python语句：

```
def hello(name="World"):
    return "Hello %s." % name
```

编辑时如下图所示：

脚本编辑视图

通过Test标签测试这个脚本，如下图所示：

测试脚本

保持 name 字段为空并单击 Run Script 按钮。Zope 应该返回 Hello World。现在返回并试试在 value 字段中输入你的姓名，然后单击 Run Script 按钮。

因为脚本是针对 Zope 对象调用的，你就可以通过 context 变量访问相应的 Zope 对象。例如，以下的脚本返回一个指定的 Zope 对象所包含的对象数量：

```
## Script (Python) "numberOfObjects"

##

return len(context.objectIds())
```

这个脚本调用 context.objectIds()来查找所包含的对象数量。当你对一个给定的 Zope 对象调用这个脚本，环境变量被绑定给 context 对象。因此，如果你通过访问 URL: FolderA/FolderB/_numberOfObjects 调用这个脚本，context 参数将指向 FolderB 对象。

当用 Python 编写逻辑时，典型的情况是你想查询 Zope 对象、调用其它脚本和返回报告。例如假设你想实现一个简单的工作流系统，其中各种对象用象征它们状态的属性相连。你可能想生成报表，这些报表总结了什么对象处于什么状态。你可以使用 Python 来查询对象，并检测它们的属性。例如，以下是个带有参数 status 的名为 objectsForStatus 的脚本：

```
## Script (Python) "objectsForStatus"

##parameters=status

## Returns all sub-objects that have a given status
```

```
property.

results=[]

for object in context.objectValues():

    if object.getProperty('status') == status:

        results.append(object)

return results
```

这个脚本对一个对象的下级对象进行循环并返回所有的具有 status 属性（带有给定值）的下级对象。代码中，“##”是通过 FTP 编辑脚本时自动生成的，在这里可以指定参数。请参见后边的“绑定变量”部分。 你可以使用以下的 DTML 脚本发送电子邮件报表。例如：

```
<dtml-sendmail>

To: <dtml-var ResponsiblePerson>

Subject: Pending Objects

These objects are pending and need attention.

<dtml-in expr="objectsForStatus('Pending')">

<dtml-var title_or_id> (<dtml-var absolute_url>)

</dtml-in>

</dtml-sendmail>
```

这个例子给你显示了你如何使用 DTML 处理表现或者报表格式，同时，Python 处理逻辑。这是一种非常重要的模式，你将一次又一次的在 Zope 里看到它。

4.1.2. 绑定变量

只要调用一个基于 Python 的脚本就会创建一套专用变量。这些变量（在 Bindings 视图中定义）用于让脚本访问其它 Zope 对象和脚本。其它对象中不存在这些变量。ZPT 中有类似的变量。

这些绑定变量都有默认值，一般不需要修改。在这里解释它们是为了让你知道每个专用变量如何工作，以及在脚本中如何使用这些变量。

Context

名称默认为“context”。这个变量是指脚本被调用时的目标对象。

Container

名称默认为“container”。这个变量指脚本所在的文件夹。

Script

名称默认为“script”。这个变量指脚本对象本身。

Namespace

默认为空。如果你是通过DTML方法调用脚本的，并且你已经为这个绑定选择了一个名称，那么被指定的变量包含“高级DTML”一章里所讲述的DTML名称空间。还有，如果设置了这个绑定，并且DTML以不直接传递参数的方式调用这个脚本，那么这个脚本就在DTML名称空间中搜索参数。

Subpath

名称默认为“traverse_subpath”，这是一个高级变量，对于本书中的任何一个例子，你都不需要。如果你的脚本被漫游（traverse），意即在一个URL中其它路径元素跟随在脚本名称之后，于是那些路径元素被放置在一个列表中，从左到右，即在这个变量里。

如果你通过FTP编辑脚本，将会发现这些绑定被罗列在你的脚本文件顶部的注释中。例如：

```
## Script (Python) "example"

##bind container=container

##bind context=context

##bind namespace=

##bind script=script

##bind subpath=traverse_subpath

##parameters=name, age

##title=

##

return "Hello %s you are %d years old." % (name, age)
```

你可以通过更改这些注释来更改脚本的绑定，然后上载你的脚本。注意这些注释语句有特定的含义。

这些绑定可用来控制定位对象的方式。比如，对于前边的例子，feed 脚本可以包含以下一行：

```
animal_id = context.getId()
```

我们可以得到“hippo”，这是因为 context 是 hippo，它的 id 是“hippo”。getId()很常用，比如：

```
folder_id = container.getId()

script_id = script.getId()
```

4.1.3. 访问 HTTP 请求

我们如何取得用户发送的请求信息？我们可以在 REQUEST 对象中找到这些信息。REQUEST 表示了 web 请求信息。例如，我们通过 URL Zoo/ LargeAnimals /hippo/feed?food_type=spam 调用 feed，我们可以通过 context.REQUEST.food_type 来调用 food_type 变量。这种方式也适用于表单。

另外一种访问 REQUEST 的方法是把它传递给脚本。然后可以通过 REQUEST.food_type 进行调用。

4.1.4. 字符串处理

脚本常用来处理字符串。Python 有一些标准字符串处理模块。由于安全的限制，你不能在基于 Python 的脚本中执行规则表达式（regular expression）处理，但可以通过外部方法对象来完成。你可以直接调用 string 模块。你还可以从 DTML 访问字符串模块，但通过 Python 要容易得多。假设你需要替换 DTML 文档中的某个单词。以下是一个脚本，replaceWord，它接受两个参数，word 和 replacement。这个脚本可进行替换操作：

```
## Script (Python) "replaceWord"

##parameters=word, replacement

##

Replaces all the occurrences of a word with a

replacement word in the source text of a DTML

Document. Call this script on a DTML Document to use

it.

Note: you will need permission to edit a document in order

to call this script on the document.
```

```
This script assumes that the context is a DTML document,

which provides the document_src() and manage_edit() methods

described in Appendix B (API Reference).

import string

text=context.document_src()

text=string.replace(text, word, replacement)

context.manage_edit(text, context.title)
```

你还可以直接使用 replace 方法：

```
## Script (Python) "replaceWord"

##parameters=word, replacement

##

text=context.document_src()

text=text.replace(word, replacement)

context.manage_edit(text, context.title)
```

你可以通过 Web 对一个 DTML 文档调用这个脚本来更改文档的源内容。例如，URL：Swamp/replaceWord?word=Alligator&replacement=Crocodile 就对 Swamp 文档调用脚本并用单词“Crocodile”替换单词所有的“Alligator”。

当然，也可以对其它对象调用这个脚本，比如 DTML 方法、页面模板、其它脚本等等。

通过这种方式也可以完成文字搜索的功能，但 Zope 中有一个专门用于搜索的工具，就是目录册（Catalogs）。可参见“内容搜索与分类”一章。

4.1.5. 处理数学

脚本的另外一个常见用途是进行数学计算，用 DTML 或 ZPT 完成这种任务将是笨拙的。数学和随机模块让你可以用 Python 访问许多数学函数。这些模块就像在 Python.org Web 站点上描述的那样，是标准的 Python 服务。

```
: math
```

数学函数，例如 sin 和 cos

: random

伪随机数字生成函数

random 模块中一个有趣的函数是 choice 函数，它从一个对象序列中返回一个随机选项。以下这个 randomImage 脚本显示了如何使用这个函数：

```
## Script (Python) "randomImage"

##

When called on a Folder that contains Image objects this

script returns a random image.

import random

return random.choice(context.objectValues('Image'))
```

假设你有一个名为 Images 的文件夹，它包含有几个图片。你可以用 DTML 随机显示文件夹中的图片，就像这样：

```
<dtml-with Images>

  <dtml-var randomImage>

</dtml-with>
```

这段 DTML 对 Images 文件夹调用 randomImage 脚本。

ZPT 中相同的用法为：

```
<span tal:replace="here/Images/randomImage"/>
```

4.1.6. 打印语句支持

基于 Python 的脚本可以非常方便的完成打印信息的功能。通常，被打印的数据被发送到标准输出并且显示在控制台中。对于像 Zope 这样的服务器应用程序，这是不实用的，这是因为大多数时间你不能访问服务器的控制台。脚本能够打印并且用特殊变量 printed 返回打印的内容。例如：

```
## Script (Python) "printExample"

##
```

```
for word in ('Zope', 'on', 'a', 'rope'):
    print word
return printed
```

结果为：

```
Zope
on
a
rope
```

上边的脚本在执行的时候，会自动以换行的形式显示出来。

你可以使用 print 语句在脚本中执行简单的调试。这里的 Print 语句不适合完成复杂的输出控制。应该使用 DTML 或 ZPT 来完成页面显示。

4.1.7. 内建函数

基于 Python 的脚本对象中所支持的内建函数略微不同于通常的 Python 语言中的内建函数。之所以不同是为了避免用户执行不安全的操作。比如：不存在 open 函数，这样就阻止用户访问文件系统。为了弥补部分缺少的内建函数，提供了一些额外的函数。

以下是通用的内建函数列表，它们的作用和标准的 Python 内建函数一样：None, abs, apply, callable, chr, cmp, complex, delattr, divmod, filter, float, getattr, hash, hex, int, isinstance, issubclass, list, len, long, map, max, min, oct, ord, repr, round, setattr, str, tuple。有关这些内建函数作用方面的更多信息，请见在线 Python 文档。

对于在 range 和 pow 函数，也可以使用，使用方式和标准 Python 中的一样；然而它们不能生成非常大的数字和序列。这个限制有利于防止以前描述过的“拒绝服务”攻击。

另外，存在这些 DTML 效用函数： DateTime , test, namespace, 和 render。有关这些函数更多的信息请见“DTML 参考”。

最后，为了补偿缺少 type 函数，可使用 same_type 函数比较两个或更多个对象类型，如果它们是同一类型就返回真。因此，不是使用以下语句：

```
if type(foo) == type([]):
    return "foo is a list"
```

要检测 foo 是否为一个列表，你应该这样使用 same_type 函数：

```
if same_type(foo, []):

    return "foo is a list"
```

现在让我们看看外部方法对象，比起基于Python的脚本，它提供了更强大的能力和较少的限制。

4.1.8. 使用外部方法(External Methods)

有些时候脚本、DTML和ZPT所具有的安全约束会很不方便。例如，你也许希望从磁盘读取文件、访问网络或者为某事使用一些高级库函数，就像规则表达式或者图像处理。在这些情况下，你需要使用外部方法。

要创建和编辑外部方法，需要访问文件系统。由于不能在Web浏览器中直接编辑脚本，会有些不方便。申请访问服务器的文件系统提供了一个重要的安全控制方式。如果用户拥有访问服务器文件系统的权限，他们同时就拥有了破坏Zope的能力。因此，通过在文件系统中编辑脚本，就可以增强安全性。

Zope服务器上的Extensions目录专门用来创建和编辑外部方法文件。这个目录位于顶级Zope目录。另外，还可以在Zope安装目录中的product目录中的Extensions目录中创建和编辑外部方法文件。INSTANCE_HOME目录中，也可以存放外部方法文件。关于INSTANCE_HOME目录可参考安装Zope部分。

让我们举例说明。在服务器上的Zope的Extensions目录中创建一个名为Example.py的文件。在Example.py文件中，输入以下代码：

```
def hello(name="World"):

    return "Hello %s." % name
```

现在已经创建了一个Python函数，接下来，如何使用呢？

进入管理界面，从产品添加列表中选择External Method。然后输入id，比如“hello”，然后在Function name中输入“hello”，在Module name中输入“Example”。最后点击Add按钮。这样就添加了外部方法。如果再点击这个外部方法，则会显示它的属性，如下图所示：

外部方法的属性视图

注意，如果你想创建多个相关的外部方法，不需要在文件系统创建多个模块文件。你可以把这些方法函数放在一个模块中，然后在 Zope 中为每个函数添加一个外部方法，其中模块名称是一样的，但函数名称不同。

现在，通过进入 Test 视图测试新脚本。你会看见一条问候语。你可以通过在 URL 中指定名称给脚本传递参数。例如，hello?name=Spanish+Inquisition。

这个例子和以前用过的基于 Python 的脚本对象中的“hello world”例子完全相同。事实上，对于象这样简单的字符串处理任务，脚本对象就已经足够了。

之所有需要使用外部方法是为了访问文件系统或网络，或者调用那些受到限制的 Python 模块。

例如，脚本对象不能调用系统中的环境变量。而通过外部方法可以这样来调用：

```
def instance_home():
    import os
    return os.environ.get('INSTANCE_HOME')
```

规则表达式是另外一个受到限制的有用工具。让我们看一个例子，通过以下这个外部方法，可以取得 HTML 页面中的正文部分即取得<body>和</body>之间的部分。

```
import re
pattern = r"<\s*body.*?>(.*?)</body>"
```

```
regexp = re.compile(pattern, re.IGNORECASE + re.DOTALL)

def extract_body(htmlstring):

    """

    If htmlstring is a complete HTML page, return the string

    between (the first) <body> ... </body> tags

    """

    matched = regexp.search(htmlpage)

    if matched is None: return "No match found"

    body = matched.group(1)

    return body
```

这个例子中，我们先导入 re 模块，然后使用 extract_body 函数对 html 数据进行处理。规则表达式会在第一次调用的时候编译一次，而不是每次调用都编译，这样效率就得以提高。

接下来，把这些代码放在 my_extensions.py 文件中，再添加一个 id 为 body_external_m 的外部方法，指定 my_extensions 为模块名，extract_body 为函数名。

你就可以在脚本中这样来调用这个外部方法：

```
## Script (Python) "store_html"

##

# code to get 'htmlpage' goes here...

htmlpage = "some string, perhaps from an uploaded file"

# now extract the body

body = context.body_external_m(htmlpage)

# now do something with 'body' ...
```

你可以在这个例子中添加一个 HTML 页面，试试效果如何。

以下这个例子，使用Python图像软件库（Python Imaging Library (PIL)）创建文件夹里的现有图像对象的缩小版本。在位于Extensions目录中的名为Thumbnail.py的文件中输入以下代码：

```
def makeThumbnail(self, original_id, size=200):
    """
    Makes a thumbnail image given an image Id when called on a Zope
    folder.
    The thumbnail is a Zope image object that is a small JPG
    representation of the original image. The thumbnail has an
    'original_id' property set to the id of the full size image
    object.
    """
    import PIL
    from StringIO import StringIO
    import os.path
    # none of the above imports would be allowed in Script (Python)!
    # Note that PIL.Image objects expect to get and save data
    # from the filesystem; so do Zope Images. We can get around
    # this and do everything in memory by using StringIO.
    # Get the original image data in memory.
    original_image=getattr(self, original_id)
    original_file=StringIO(str(original_image.data))
    # create the thumbnail data in a new PIL Image.
    image=PIL.Image.open(original_file)
```

```
    image=image.convert('RGB')
    image.thumbnail((size,size))
    # get the thumbnail data in memory.
    thumbnail_file=StringIO()
    image.save(thumbnail_file, "JPEG")
    thumbnail_file.seek(0)
    # create an id for the thumbnail
    path, ext=os.path.splitext(original_id)
    thumbnail_id=path + '.thumb.jpg'
    # if there's an old thumbnail, delete it
    if thumbnail_id in self.objectIds():
        self.manage_delObjects([thumbnail_id])
    # create the Zope image object for the new thumbnail
    self.manage_addProduct['OFSP'].manage_addImage(thumbnail_id,
                                                   thumbnail_file,
                                                   'thumbnail image')
    # now find the new zope object so we can modify
    # its properties.
    thumbnail_image=getattr(self, thumbnail_id)
    thumbnail_image.manage_addProperty('original_id', original_id, 'string')
```

注意，上边代码中的第一个参数是 self，是可选的。它相当于 context 变量。

要执行这段代码你必须为这个例子安装有 PIL。关于 PIL 的更多信息请见 PythonWorks Web 站点(http://www.pythonworks.com/products/pil)。 接下来创建一个名为 makeThumbnail 的外部方法，它调用 Thumbnail 模块中的

makeThumbnail 函数。 现在你有了一个创建缩小图像的方法。你可以用 URL 对一个文件夹调用这个方法，例如 ImageFolder_/makeThumbnail?original_id=Horse.gif。这会创建一个名为 Horse.thumb.jpg 的缩小图像。 你可以使用一个脚本访问文件夹中的所有的图像并且为它们创建缩小图像。比如，创建一个名为 makeThumbnails 的脚本：

```
## Script (Python) "makeThumbnails"

##

for image_id in context.objectIds('Image'):

    context.makeThumbnail(image_id)
```

这样就会访问文件夹中的所有图像并且为每个图像创建一个缩小图。

现在对一个含有图像的文件夹调用这个脚本。它为其中的每个图像创建一个缩小图像。试试再一次对这个文件夹调用 makeThumbnails 脚本，你会发现它创建了你的缩小图像的缩小图像。这没有意义。你需要更改 makeThumbnails 脚本，让它识别出现有的缩小图像并且不生成它们的缩小图像。因为所有的缩小图像都有一个 original_id 属性，你可以检测那个属性，把它作为区分缩小图像和正常图像的方法。

```
## Script (Python) "makeThumbnails"

##

for image in context.objectValues('Image'):

    if not image.hasProperty('original_id'):

        context.makeThumbnail(image.getId())
```

删除所有文件夹里的缩小图像，然后试着对文件夹调用新的 makeThumbnails 脚本。它现在看起来工作正确了。现在使用一点 DTML，你可以把脚本和外部方法粘合在一起。创建一个名为 displayThumbnails 的 DTML 方法：

```
<dtml-var standard_html_header>

<dtml-if updateThumbnails>

  <dtml-call makeThumbnails>

</dtml-if>

<h2>Thumbnails</h2>

<table><tr valign="top">
```

```
<dtml-in expr="objectValues('Image')">
  <dtml-if original_id>
    <td>
      <a href="&dtml-original_id;"><dtml-var sequence-item></a><br>
      <dtml-var original_id>
    </td>
  </dtml-if>
</dtml-in>
</tr></table>
<form>
<input type="submit" name="updateThumbnails" value="Update Thumbnails">
</form>
<dtml-var standard_html_footer>
```

当你对一个文件夹调用这个DTML方法，它循环访问文件夹里的所有图像，显示所有的缩小图像，并且把它们和原始图像链接起来，如下图所示。

显示缩小的图片

这段 DTML 方法还包含了一个表单，它使你能够更新缩小图像。如果你添加、删除或更改你的文件夹中的图像，你可以使用这个表单来更新缩小图像。

这个例子显示了如何一起使用脚本、外部方法和 DTML。Python 关心逻辑，而 DTML 处理表现。你的外部方法处理外部模块包，比如 PIL，而脚本进行简单 Zope 对象处理。当然，也可以用 ZPT 替换 DTML。

4.1.9. 用外部方法处理 XML

使用外部方法几乎可以处理任何事情。一件有趣的事情是处理 XML。你可以用外部方法生成和处理 XML。 Zope 已经可以处理某些类型的 XML 消息，例如 XML-RPC 和 WebDAV。只要你创建与其它系统通信的 Web 应用程序，就可能需要接收 XML 消息的能力。你可以用几种方式接收 XML：你可以从文件系统或通过网络读取 XML 文件，或者你可以定义远程系统能够调用的且能够接收 XML 参数的脚本。

当收到一个 XML 消息，必须处理 XML，从而找出它的含义并且判断如何对它进行反应。让我们快速的看一下如何使用 Python 手工解析 XML。假设你想把 Web 应用程序连接到 Jabber 聊天服务器（ http://www.jabber.com/）。你需要允许用户通知你并且接收基于 Web 应用程序状态的动态响应。例如，假设你想让用户使用即时消息来检查动物的状态。你的应用程序应当这样对 XML 即时消息作出相应：

```
<message to="[cage_monitor@zopezoo.org]" from="[user@host.com]">
  <body>monkey food status</body>
</message>
```

你应该扫描消息正文，找到命令，调用脚本，返回响应。比如：

```
<message to="[user@host.com]" from="[cage_monitor@zopezoo.org]">
  <body>Monkeys were last fed at 3:15</body>
</message>
```

以下是一个通过外部方法处理这个 XML 消息的程序：

```
# Uses Python 2.x standard xml processing packages.  See
# [http://www.python.org/doc/current/lib/module-xml.sax.html] for
# information about Python's SAX (Simple API for XML) support If
# you are using Python 1.5.2 you can get the PyXML package. See
# [http://pyxml.sourceforge.net] for more information about PyXML.
from xml.sax import parseString
from xml.sax.handler import ContentHandler
class MessageHandler(ContentHandler):
    """
    SAX message handler class
    Extracts a message's to, from, and body
    """
    inbody=0
    body=""
    def startElement(self, name, attrs):
        if name=="message":
            self.recipient=attrs['to']
            self.sender=attrs['from']
```

```
        elif name=="body":
            self.inbody=1
    def endElement(self, name):
        if name=="body":
            self.inbody=0
    def characters(self, content):
        if self.inbody:
            self.body=self.body + content
def receiveMessage(self, message):
    """
    Called by a Jabber server
    """
    handler=MessageHandler()
    parseString(message, handler)
    # call a script that returns a response string
    # given a message body string
    response_body=self.getResponse(handler.body)
    # create a response XML message
    response_message="""
      <message to="%s" from="%s">
        <body>%s</body>
      </message>""" % (handler.sender, handler.recipient, response_body)
```

```
    # return it to the server

    return response_message
```

这个 receiveMessage 外部方法使用 Python 的 SAX (Simple API for XML)模块解析 XML 消息。 MessageHandler 类用于接收回叫信号。Handler 保存信息。这个外部方法先创建 handler 类实例，然后把它传递给 parseString 函数。然后通过调用 getResponse 生成一个响应消息。getResponse 脚本（在此未显示）扫描正文，查找命令，查询 Web 应用程序状态，然后返回一些响应。receiveMessage 方法使用这些响应和发送者信息创建一条 XML 消息并返回它。

远程服务器使用这个外部方法的方式是通过使用标准的 HTTP POST 命令调用 receiveMessage 方法。太棒了，你已经实现了一个通过 HTTP 运行的定制的 XML 聊天服务器

4.1.10. 外部方法注意事项

尽管外部方法不受到限制，但仍然有一些事情难以处理。 只要遵守 Zope 中的规则，Python 代码能够各种任务。虽然难以在此全面讲述使用 Zope 框架编写程序，但有些事情需要注意。

如果你把自己的类实例交给 Zope 并希望它们象 Zope 对象那样工作，此时可能会发生问题。例如，你不能在一个外部方法脚本文件中定义一个类并且把它作为 Zope 对象属性。这对于 Zope 的持续机制会引发问题。你不能随便把自己的类实例交给 DTML 或脚本。问题在于你的实例没有 Zope 安全信息。你可以在模块内部定义和使用自己的类和实例，不要期望 Zope 直接使用它们。对于返回的信息应该使用简单的 Python 数据结构，例如字符串、字典和列表或 Zope 对象。

如果你希望创建新类型的对象，应该试试编写 Zope 产品。这个内容可参考 Zope 开发指南。

5. 使用基于 Perl 的脚本

基于 Perl 的脚本允许你通过 Perl 语言编写脚本对象。

5.1. Perl 语言

Perl 就像 Python 一样是一种高级脚本语言。从整体上来看，Perl 和 Python 是非常相似的语言。它们都有相似的数据构造，并使用相似的程序构造。对于 Internet 脚本编程，Perl 是一种流行的语言。在 CGI 脚本编程的早期，Perl 和 CGI 几乎是等同的。Perl 一直是主要的 Internet 脚本编程语言。

Perl 拥有非常丰富的模块集，它们囊括了几乎任何计算任务。CPAN (Comprehensive Perl Archive Network)是 Perl 资源方面的权威指南。

用于创建 Zope 脚本的 Perl 软件可以从 ActiveState 下载。默认的 Zope 安装中不支持 Perl 脚本，因此要求你安装有 Perl 和一些其它的软件包。在 Zope.org 中可以找到更多的相关信息。

5.1.1. 创建基于 Perl 的脚本

基于 Perl 的脚本非常类似于基于 Python 的脚本。两者都可以访问 Zope 对象并且通过类似的方式调用。以下是 Perl 编写的 hello world 程序：

```
my $name=shift;

return "Hello $name.";
```

让我们看一个由 Monty Taylor 编写的比较复杂的例子脚本。它通过使用 LWP:: UserAgent 软件包从网络中得到“每日 Dilbert 滑稽演员”的 URL。创建一个名为 get_dilbert_url 的脚本，代码如下：

```
use LWP::UserAgent;

my $ua = LWP::UserAgent->new;

# retrieve the Dilbert page

my $request = HTTP::Request->new('GET','[http://www.dilbert.com]');

my $response = $ua->request($request);

# look for the image URL in the HTML

my $content = $response->content;

$content =~ m,(/comics/dilbert/archive/images/[^"]*),s;

# return the URL

return $content
```

通过用 DTML 在 HTML 的 IMG 标记符中调用这个脚本，可以显示“每日 Dilbert 滑稽演员”：

```
<img src="&dtml-get_dilbert_url;">
```

然而，这段代码存在一个问题。每次你显示这个卡通片，Zope 都要进行一次网络连接。效率低并且浪费时间。每天最好连接一次 Dilbert 的 URL。

下边的这个脚本 cached_dilbert_url 用一个 dilbert_url_date 属性跟踪最后一次取得 Dilbert 的 URL 的时间：

```
my $context=shift;

my $date=$context->getProperty('dilbert_url_date');

if ($date==null or $now-$date > 1){

    my $url=$context->get_dilbert_url();
```

```
    $context->manage_changeProperties(
      dilbert_url => $url
      dilbert_url_time => $now
    );
  }
  return $context->getProperty('dilbert_url');
```

这段脚本使用两个属性，dilbert_url 和 dilbert_url_date。如果 URL 太旧，就取得新的 URL。你可以通过 DTML 使用这个脚本比如：

```
<img src="&dtml-cached_dilbert_url;">
```

Perl 和 DTML 可以结合在一起使用。

5.2. 基于 Perl 的脚本安全

就像 DTML 和基于 Python 的脚本，基于 Perl 的脚本同样受到 Zope 安全系统的限制。脚本安全在两种语言中类似，但是存在一些特定的约束。

首先，安全系统不允许使用 eval 语句对表达式求值。例如，下面的脚本：

```
my $context = shift;
my $input = shift;
eval $input
```

这段代码带有一个参数并在 Perl 中对它求值。这意味着你可以从，比如说一个 HTML 表单中调用这个脚本，然后对表单元素中的某个元素求值。这是不允许的，因为表单元素可能包含恶意的代码。

另外，基于 Perl 的脚本除了用 my 声明的本地变量外不能给其它对象分配新变量。

6. 高级获取机制

在“获取机制”一章中，我们讲过了获取机制。这种方式可以称之为容器获取。在这部分讲深入讲解获取机制的高级特性。对于初学 Zope 的人士，以下内容可以跳过。

让我们先重新看看以前的那个 Zoo 例子。

Zope 动物园站点结构

我们已经看到了 Zope 如何使用 URL 漫游和获取在上级容器中找到对象。可以使用更复杂一些的编排。比如，假设你想对 hippo 对象调用 vaccinate 脚本。那么如何使用 URL？如果你按照 Zoo/ LargeAnimals /hippo/vaccinate 来调用，将找不到 vaccinate 脚本，这是因为任何 hippo 的容器中都没有 vaccinate。

解决方法是在 URL 中提供更多的路径信息。为了能够提供更多的可能性，Zope 可以把多个 URL 结合成一个。通过获取机制，Zope 会沿着 URL 搜索这个脚本。刚才的问题可以使用 Zoo/Vet/ LargeAnimals /hippo/vaccinate。同样，如果想对 kargarooMouse 对象调用 vaccinate 脚本，则应使用 Zoo/Vet/ SmallAnimals /kargarooMouse/vaccinate。

让我们看看“Zoo/Vet/ LargeAnimals /hippo/vaccinate”的定位过程。Zope 首先进入根目录查找 Zoo。然后进入 Zoo 文件夹搜索 Vet。然后进入 Vet 搜索 LargeAnimals 。Vet 中没有这个对象，但是通过获取机制，找到 LargeAnimals ，然后查找 hippo。然后进入 hippo，查找 vaccinate。由于不能找到 vaccinate，于是继续沿着 URL 往上返回。首先返回到 LargeAnimals ，但仍不能找到 vaccinate。继续到 Vet，找到了 vaccinate。于是结束搜索，调用 vaccinate 脚本。

注意，我们也可以调整 URL。比如 Zoo/ LargeAnimals /Vet/hippo/vaccinate 也是有效的。不同之处在于搜索的顺序是不一样的。

当 Zope 在 URL 中漫游时，先搜索当前对象的子对象。如果不能找到，就搜索当前对象的容器。如果还找不到，就沿着 URL 路径向上搜索。直到找到为止，或找不到引发一个错误。如果在 URL 中指定了多个环境文件夹，就按照从左到右的顺序搜索。

这种环境获取机制是一种非常有用的机制，它可以让 URL 有更多的表现力。搜索对象时用到的 URL 决定了如何找到所需的对象。

6.0.1. 环境获取机制注意事项

6.0.1.1. 先容器后环境

注意上边谈到的环境获取机制是对容器获取机制的补充，而不是替代。

6.0.1.2. 一次搜索一个

另外一个需要明确的是，搜索时一次搜索一个，只要找到对象，就不再继续搜索。举例来说，假设文件夹结构如下：

文件夹结构

现在假设页面 about_penguins 中有一个指向 Images/penguins.png 的链接。对于/Content/Images/penguins.png 这样的 URL 是不起作用的。URL 漫游的时候是从左往右进行的。一次只搜索一个。首先找到 Content，然后是 Images，没有找到 penguins.png。所指定的上级容器都已经搜索完了，此时停止，引发一个错误。它不会搜索根目录中的 Images 目录。

6.0.1.3. 可读性

环境获取机制有可能会使代码难以理解。

6.0.1.4. 脆弱性

过渡使用环境获取机制容易导致脆弱性。这种方式容易使得网站变得松散。有可能会影响脚本的执行。比如，一个脚本通过复杂的路径调用另外一个脚本。就容易产生错误。还以 Zoo 例子说明。存在几种不能运行 feed 脚本的情况。首先是插入另外一个具有相同名称的对象的时候，其次是调用不合适的路径。

7. 通过脚本调用 DTML

有些时候，你需要通过脚本调用 DTML 方法或文档。比如，HTML 表单中调用一个脚本，这个脚本处理用户的输入，然后返回输出页面。

脚本擅长逻辑计算，但不适合生成 HTML。因此需要把返回的信息传递给 DTML，通过脚本调用 DTML。

假设，我们已经有了一个 a_dtml_method，我们可以这样调用它：

```
# grab the method and the REQUEST from the context

dtml_method = context.a_dtml_method
```

```
REQUEST = context.REQUEST

# call the dtml method, for parameters see below

s = dtml_method(client=context, REQUEST=REQUEST, foo='bar')

# s now holds the rendered html

return s
```

注意，DTML 方法和文档需要 client 和 REQUEST 参数。如果给 DTML 方法传递了 client，这个方法以 client 属性的方式搜索名称。这个例子中还传递了 REQUEST 对象和关键字参数。这个 DTML 方法先在关键字参数中搜索变量，然后是在名称空间，最后是在 REQUEST 对象中搜索变量。

7.1. 通过脚本调用 ZPT

和 DTML 一样，有时我们需要通过脚本调用 ZPT。假设我们有以下模板：

```
Hello <span tal:replace="options/name | default">

World

</span>
```

调用方法如下：

```
pt = context.hello_world_pt

s = pt(name="John Doe")

return s
```

参数 name 的值通过 options/name 在页面中显示出来。通过这种方式可以传递多个值。比如可以构建一个列表，然后把这个列表传递给页面模板。列表可以通过外部方法来构建。在 Extensions 目录中添加 my_extensions.py，代码如下：

```
class Giraffe:

    __allow_access_to_unprotected_subobjects__ = 1

    def __init__(self, name, neck_length=None):

        self.name = name

        self.n_length=neck_length
```

```
    def neck_length(self):
        n = self.n_length
        if not n: return "unspecified"
        if type(n) == type(0.0):
            return "%s meters" % n
        return n
def giraffes(self):
    # make a list of giraffes
    glist = []
    for name, neck in (('Guido', 1.2), ('Jim', 'long'), ('Barry', None)):
        g = Giraffe(name, neck_length=neck)
        glist.append(g)
    # display the lot of them
    pt = self.display_giraffes
    return pt(giraffes=glist)
```

添加一个外部方法giraffes，module为my_extensions，function为giraffes。然后再添加页面模板display_giraffes，代码为：

```
<table border="1">
  <tr>
    <th>Name</th>
    <th>Neck length</th>
  </tr>
  <tr tal:repeat="giraffe options/giraffes">
```

```
    <td tal:content="giraffe/name">name</td>

    <td tal:content="giraffe/neck_length">neck_length</td>

  </tr>

</table>
```

如果通过外部方法 giraffes 的 Test 标签，可以看到类似于下图中的结果：

Giraffe 表格

在 my_extensions 模块（即 my_extensions.py 文件）中，我们定义了类 Giraffes。语句 allow_access_to_unprotected_subobjects = 1 的作用是告诉 Zope 任何人都可以访问 giraffes。

然后在 giraffes 函数中，我们先创建了一些 giraffes 对象，然后交给页面模板显示结果。

在页面模板中，我们再次使用 options 访问 giraffe 对象，比如 giraffe/name。在这个过程中自动调用 neck_length 方法。如果你需要给 giraffe 对象传递参数，应该使用 python 表达式，比如：giraffe.neck_length()。

8. 给脚本传递参数

可以给脚本传递参数。当你通过 Web 调用脚本，Zope 会在请求中自动查找参数并传递给脚本。比如，如果你有一个带有参数 dolphin 的脚本，Zope 会在 REQUEST 中搜索 dolphin。在实际中，常常用这种方式处理表单，比如：

```
<form action="form_action">

Name of Hippo <input type="text" name="name"><br>

Age of Hippo <input type="text" name="age"><br>

<input type="submit">

</form>
```

你可以通过脚本 form_action 来处理表单，这个脚本包含参数 name 和 age：

```
## Script (Python) "form_action"

##parameters=name, age

##

"Process form"

age=int(age)

message= 'This hippo is called %s and is %d years old' % (name, age)

if age < 18:

    message += '\n %s is not old enough to drive!' % name

return message
```

除了表单变量，你还可以指定任何请求变量作为脚本的参数。例如，要访问请求和响应对象，只需要在参数列表中加入 REQUEST 和 RESPONSE。

在上面的脚本中，可能存在一个小问题。你也许希望 age 是一个整数，而不是字符串，但所有的表单变量都是字符串。Perl 可以自动处理，但 Python 不行。需要使用转换函数进行转换：

```
age=int(age)
```

但这个过程显得不太方便。为此，Zope 提供了在表单中指定输入数据的类型的功能。方式如下：

```
Age <input type="text" name="age:int">
```

在名称当中附加了:int，这样就告诉 Zope 自动把输入的信息转换成整数。这个过程被称为编集。如果不能进行转换，就会报错。如下图所示：

参数转换错误

Zope 可以进行许多种参数转换。以下是一个简要的列表：

: boolean（布尔类型）

把变量转换成 true 或 false。0，None, 空字符串，或者空序列都为假，其它都为真。

: int

把变量转换成整数。

: long

把变量转换成长整数。

: float

把变量转换成浮点数。

: string

把变量转换成字符串。由于原来已经是字符串了，所以这个转换比较少用。

: text

把变量转换成带有行结束符的字符串。

: list

把变量转换成Python列表。

: tuple

把变量转换成Python元组。元组类似于列表，但不能更改。

: tokens

用空格把字符串分隔成列表。

: lines

用行结束符把字符串转换成列表。

: date

- 把字符串转换成 DateTime 对象，可使用的格式很多，比如：10/16/2000, 12:01:13 pm

: required

如果变量不存在则引发错误。

: ignore_empty

如果变量为空字符串，则从request中排除。

list和tuple可以和其它种类的转换结合在一起使用。这样就可以对列表或元组中的每个数据进行转换，比如：

```
<form action="processTimes">
<p>I would prefer not to be disturbed at the following
times:</p>
<input type="checkbox" name="disturb_times:list:date"
value="12:00 AM"> Midnight<br>
<input type="checkbox" name="disturb_times:list:date"
value="01:00 AM"> 1:00 AM<br>
```

```
<input type="checkbox" name="disturb_times:list:date"

value="02:00 AM"> 2:00 AM<br>

<input type="checkbox" name="disturb_times:list:date"

value="03:00 AM"> 3:00 AM<br>

<input type="checkbox" name="disturb_times:list:date"

value="04:00 AM"> 4:00 AM<br>

<input type="submit">

</form>
```

通过一起使用 list 和 date，Zope 将先把选中的时间转换成 date，然后把所有选中的时间组合成列表 disturb_times。

一种更复杂的转换是把一序列输入转换成 record（记录）。

Record 是一种具有属性的数据结构。通过使用 Record，可以把几个输入变量组合成一个带有属性的变量。现有的 Record 转换有：

: record

把变量转换成 record 的属性。

: records

把变量转换成一个结构对象列表的属性。

: default

如果变量为空，则提供一个默认值组结构对象属性。

: ignore_empty

如果变量为空，忽略一个结构对象属性。

这里有一个例子：

```
<form action="processPerson">

  First Name <input type="text" name="person.fname:record" /><br />
```

```
  Last Name <input type="text" name="person.lname:record" /><br />
  Age <input type="text" name="person.age:record:int" /><br />
  <input type="submit" />
</form>
```

该表单会调用processPerson脚本，并传递一个person参数给脚本，这个参数有三个属性，分别是fname、lname和age。在脚本中使用person参数的方法如下：

```
## Script (Python) "processPerson"
##parameters=person
##
"Process a person record"
full_name="%s %s" % (person.fname, person.lname)
if person.age < 21:
    return "Sorry, %s. You are not old enough to adopt an aardvark." % full_name
return "Thanks, %s. Your aardvark is on its way." % full_name
```

records与record类似，但它会生成一个结构对象列表，如：

```
<form action="processPeople">
  <p>Please, enter information about one or more of your next of
  kin.</p>
  <p>
    First Name <input type="text" name="people.fname:records" />
    Last Name <input type="text" name="people.lname:records" />
  </p>
  <p>
```

```
    First Name <input type="text" name="people.fname:records" />
    Last Name <input type="text" name="people.lname:records" />
  </p>
  <p>
    First Name <input type="text" name="people.fname:records" />
    Last Name <input type="text" name="people.lname:records" />
  </p>
  <input type="submit" />
</form>
```

变量 people 是一个 records 列表，每个 record 都有一个 fname 和 lname 属性。

list:int 和 int:list 是一样的，与位置无关。

另一个有用的转换器是 action，它可从新定义 form 的行为。这样我们可在 form 中根据不同条件选择不同的脚本。action 转换器的描述如下：

: action

为原始的 form 添加新的选择功能，在有两个提交按钮的情况下，我们可根据不同的提交按钮选择不同的脚本。action 的我们也理解为 method。

: default_action

当没有 action 转换器时的默认选择。

这里是一个使用 action 的例子：

```
<form action="employeeHandlers">
  <p>Select one or more employees</p>
  <input type="checkbox" name="employees:list" value="Larry" /> Larry<br />
  <input type="checkbox" name="employees:list" value="Simon" /> Simon<br />
```

```
  <input type="checkbox" name="employees:list" value="Rene" /> Rene<br />

  <input type="submit" name="fireEmployees:action" value="Fire!" /><br />

  <input type="submit" name="promoteEmployees:action" value="Promote!" />

</form>
```

我们假设一个名为 employeeHandlers 的文件夹下有两个脚本，分别 fireEmployees 和 promoteEmployees。表单将根据你选择的提交按钮执行其中的一个脚本。

Form 转换器在一发 Zope 应用时是很有用的，我们要好好学会使用它。

16.5. 安全的脚本

在 符合 Zope 安全策略的前提下，所有的脚本都能通过 Web 来编辑。Zope 考虑到本身系统的安全性，Python Script 的一些功能是受到约束的。例如，如果脚本要直接访问文件系统，则只能通过外部方法来实现。本章将介绍在 Zope 的安全架构如何确保脚本安全运 行。

16.5.1. Python Script 的安全约束

如 果没有约束，Python Script 可访问 Zope 的私有对象，修改 Zope 对象和影响 Zope 进程，直接访问运行 Zope 的服务器等。这些都会使 Zope 服务器运行出错，造成 Down 机。约束 Python Script 的目的是防止它危害到 Zope 系统的正常运行。这些约束分别有：

循环限制

> Script 不能创建一个无限循环。如果 Script 运行一个大的循环，Zope 会引发一个异常。这些循环包括 for 和 while 循环。

导入限制

> Script 不能随意地导入模块和包，你只能导入 Products.PythonScript.standard 工具模块、AccessControl 模块。如果你要导入其它包和模块，请用外部方法来实现。

访问限制

> 当 你访问 Zope 对象时会受到 Zope 标准的安全策略的限制。换句话说，当用户用 Script 去访问对象时，是要被授权的。在调用脚本时，可以使用代理角 色，它可改变一个角色的授权。另外，你不能访问用下横线到头的对象，因为 Zope 把这个对象当成私有对象。在脚本中不要定义类，因为在 zope 中，你被限 制访问这些类的属性，因为你不能访问类的__init__方法。如果你要定义类，你应该使用外部方法或 Zope 产品。

写入限制

一般地，你不能直接改变Zope对象的属性，你应该调用Zope API提供的方法。

尽管有这些约束，但用户使用python Script还是有机会耗用大量的CPU和Memory资源。所以一些恶意的脚本会造成Dos攻击，耗用大量的系统资源。这是一个很难解决的问题，DTML同样也有这样的问题，所以我们应控制好可执行脚本人员的权限。

16.6. DTML VS Python VS Page Templates

Zope为我们提供了多种开发的工具，编写短小的代码可使用python script脚本实现。DTML和ZPT可用以设计表现层。大型的逻辑处理可用python script或外部方法。下面用一个例子演示一下三种工具的不同实现过程。

使用DTML

```
<dtml-in objectValues>
  <dtml-var getId>: <dtml-var sequence-item>
</dtml-in>
done
```

使用ZPT

```
<div tal:repeat="item here/objectValues"
     tal:replace="python:'%s: %s\n' % (item.getId(), str(item))" />
```

使用python

```
for item in context.objectValues():
    print "%s: %s" % (item.getId(), item)
print "done"
return printed
```

16.7. 远程调用Zope脚本

使用XML-RPC

```
import xmlrpclib
server = xmlrpclib.Server('http://www.zopezoo.org/')
```

```
for employee in server.JanitorialDepartment.personnel():
    server.fireEmployee(employee)
```

通过HTTP客户端，可以远程调用Zope的脚本，下面以wget这个程序为例介绍一下。假设Zope中的Lions目录下有一个叫feeds的脚本，如果我们想每天早上运行：

```
$ wget --spider http://www.zopezope.org/Lions/feed
```

--spider选项告诉wget，不保存文件，直接执行。如果要该脚本需授权访问，则可这样写：

```
$ wget --spider --http-user=ZooKeeper \
    --http-passwd=SecretPhrase \
    http://www.zopezope.org/Lions/feed
```

设置cron，每天早上8点运行该命令。

```
$ crontab -e
0 8 * * * wget -nv --spider --http_user=ZooKeeper \
  --http_pass=SecretPhrase http://www.zopezoo.org/Lions/feed
```

16.8. 结论

使用脚本，你可控制Zope对象，粘合应用程序的逻辑层、数据层和表现层。使用Zope API，你可在Zope文件夹内管理对象。你还可通过编程，实现图像处理和XML解析。

第十五章 Zope服务

Zope中有些服务对象。这些服务对象完成一些基础性的功能。

1. 访问规则服务

通过访问规则服务使得用户访问Zope站点时能够“漫游”文件夹。当用户通过浏览器提交一个URL请求，Zope搜索URL中指定的文件夹，这个搜索的过程被称为漫游。访问规则决定了搜索定位文件夹的方式。以下通过例子进行解释。

在Zope中创建名为accessrule_test的文件夹，然后在这个文件夹中创建名为access_rule的脚本，它有两个参数：container和request。代码如下：

```
useragent = request.get('HTTP_USER_AGENT', '')
```

```
if useragent.find('Windows') != -1:
    request.set('OS', 'Windows')
elif useragent.find('Linux') != -1:
    request.set('OS', 'Linux')
else:
    request.set('OS', 'Non-Windows, Non-Linux')
```

这个脚本根据用户所使用的操作系统的不同，能够在REQUEST中加入一个新变量“OS”，数值可以是“Windows”、“Linux”或“Non-Windows, Non-Linux”。

保存这个脚本，然后点击access_test文件夹的Contents视图，从添加列表中选择“Set Access Rule”，在id字段中输入access_rule。然后点击 Set Rule按钮。然后，会显示“access_rule is now the Access Rule for this object”，然后点击OK。注意，access_rule脚本的图标发生了变化，表示已经应用了这个规则。

还是在这个文件夹中，创建一个名为test的DTML方法，代码如下：

```
<dtml-var standard_html_header>
<dtml-var REQUEST>
<dtml-var standard_html_footer>
```

保存以后，点击View视图。会看到所有REQUEST中的变量。在“other category”中，会有一个新变量，即OS，显示了用户的操作系统。

重新进入accessrule_test文件夹，再次从添加列表中选择Set Access Rule，点击 No Access Rule 按钮，会提示取消了规则。 如果再次调用test，会发现OS变量没有了。

访问规则的用法很多。访问规则不一定是脚本对象，也可以是DTML方法或外部方法。

2. 临时存储服务

临时文件夹（Temporary Folder）是一种用来临时存储数据的文件夹。临时文件夹和普通的文件夹的主要区别如下：

当重新启动Zope时，存储在临时文件夹中的内容会消失。

临时文件夹中存储的对象不支持撤销功能。

默认的情况下，在根目录中有一个名为 temp_folder 的临时文件夹，title 属性为“Session Data Container”。Zope 中默认的 session 系统会调用这个文件夹。

临时文件夹的内容存储在 RAM 内存中，而不是存在 Zope 数据库中。这样就适合存储经常进行读写操作的小型对象，比如 session 数据。临时文件夹不适合存储大的对象。

3. 版本服务

版本对象可以用来帮助多人协同工作。通常，当你编辑一个文档时，其他人可能同时会在编辑另外一个文档。在一个大型 Zope 站点中，成百甚至上千的人们能够同时使用 Zope。对于其中的大部分内容来说会运转良好，但是可能会发生问题。例如，两个人可能会同时编辑同一文档。当第一个人完成修改，这些修改在 Zope 中被保存。当第二个人完成修改，他覆盖了第一个人的修改。对于这样的问题，当然可以通过使用撤销和历史来对付这个问题，但是它始终会存在问题。使用版本（Version）对象就可以彻底解决这个问题。

另外一个你可能遇到的问题是，你可能想做一些修改，但是你想把它们完成之后才公布出来。例如假设你想更改站点的菜单结构。当人们正在使用站点时，你不想让用户看到这些修改，这是因为当你修改时可能会临时中断导航系统。

在 Zope 里，版本是一种实现私有修改的方法。你可以修改许多不同的文档，而其他人看不到其中的变化。当你断定已经完成时，可以公布所做的修改，或者放弃这些修改。只要你愿意你就可以在一个版本中工作。例如，你可能花费了一周的时间把链接点放到新菜单系统中，一旦完成，就可以提交这个版本，使所有修改立即生效。

注意，通过 Zope 管理界面使用版本的时候，需要浏览器支持 cookies 功能。大多数浏览器默认情况下都支持这个功能。

举例说明。从产品添加列表中选择 Version。会进入到一个添加表单。输入 id 为 MyChanges ，然后单击 Add 按钮。现在，已经创建了一个版本，但是你还不能使用它。要想使用这个版本，先单击它，进入到这个版本的 Join/Leave（加入/离开）视图，如下图所示。

加入版本

这个版本显示出当前没有使用它。单击 Start working in MyChanges （开始在 MyChanges 中工作）按钮。现在，Zope 应该显示出你正在一个版本中工作。现在返回到根文件夹。注意，每到一个地方，你都会在屏幕的顶部看到一条短消息，它显示为 You are currently working in Version /MyChanges （你当前正在版本 /MyChanges 中工作）。这条消息让你知道在此处所做的任何修改不会被公布，但是会存储在你的版本中。例如，创建一个名为 new 的新 DTML 文档。注意，在它的 id 后边会有一个小的红色钻石图标。现在编辑你的 standard_html_header 方法。给它添加一行，如下：

```
<HTML metal:use-macro="container/aq_parent/template/macros/page">
  <HEAD>
<link rel="stylesheet" href="css.css" type="text/css">
    <TITLE><dtml-var title_or_id></TITLE>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
  <H1>Changed in a Version</H1>
```

当你工作在一个版本中时，你创建或编辑的任何对象都会标记有一个红色钻石图标。现在返回到你的版本，然后单击 Quit working in MyChanges （从 MyChanges 中退出工作）按钮。现在，返回新文档试试。就会发现那个在你的版本中创建的文档消失了。其它你在版本中所做的任何修改也不见了。你会注意到 standard_html_header 方法现在有了一个小红钻石并且紧跟一个锁符号。它就显示出这个对象在一个版本中被修改了。在一个版本中修改一个对象就会锁定它，因此其他的人不可以修改它，直到你提交或放弃修改。锁定确保那些当你工作于一个版本中时所做的修改不会覆盖其他人所做的修改。因此，例如，如果你想确保在

特定的时间只有你在处理一个对象，你就可以在版本中修改它。尽管在版本中工作能够保护文档不会被意外的修改，但是如果你想编辑被其他某个人锁定的对象，它就会显得不方便。为了避免把其他人锁定在外以致他们不能修改对象，限定你的版本使用范围是一个好想法。

现在返回到你的版本，然后单击Start working in MyChanges（开始在 MyChanges 中工作）按钮。它又恢复到你离开版本时的状态。现在，需要让你所做的修改永久生效。进入到Save/Discard（保存/放弃）视图，如下图所示。

提交版本更改

在注释字段中输入一个注释，例如This is a test，然后单击Save按钮。现在，你的修改就被公布了，并且所有在你的版本中修改过的对象被解锁。注意，你仍然在你的版本中工作。进入到Join/Leave（参加/离开）视图，单击Quit working in MyChanges 按钮。现在可以验证在你的版本中创建的文档是可见的。你对standard_html_header所做的修改也是可见的。就像Zope中的其它任何事情一样，如果你愿意，你可以选择撤销修改。进入到撤销视图，你会发现针对所有你在版本中的修改你只有一个事务处理，而不是许多事务处理（每个修改对应一个）。如果你想撤销这个事务处理，所有你在版本中所做的修改都会被撤销。

版本对于群组协作是一个强大的工具。通过版本，要进行测试工作，不必运行一个实际服务器和一个测试服务器。这是因为通过版本能够完成测试和运行的工作，并且在完成时公布内容。每个人都可以有自己的版本。也可以是许多人在同一版本中工作。通过这种方式，你就能基于版本共同合作修改，在公布内容的同时隐藏正在进行修改的内容。

3.1. 提示：版本和ZCatalog

ZCatalog是Zope中的索引和搜索引擎。在后面的章节中会讲到。版本不能和ZCatalog一起工作。这是因为版本会锁定对象，组织版本外的更改。ZCatalog会自动跟踪变化。版本不能和ZCatalog配合使用。

4. 缓存服务

缓存可用于临时存储经常访问的信息。使用缓存的原因是为了提高速度。任何类型的动态内容，例如一个 DTML 页面或者一个 Python 脚本，每次调用时都要运行。对于简单页面或者快速脚本，通常这不成问题。对于非常复杂的 DTML 页面或者脚本，这样就会进行大量计算或者调用远程服务器，访问这样的页面或脚本会多花费时间。DTML 和 Python 都可以达到这样的复杂程度，特别是如果你使用许多循环（例如<dtml-in>标记符或者 Python 循环语句），或者如果你调用含有许多相互调用的脚本等等类似的情况。花费大量的时间进行计算会很耗费资源。

缓存可以明显的提供站点的访问速度。第一次访问页面时，和普通的访问一样，会相对慢一些，然后页面的结果就会存储在缓存中，以后的访问会访问缓存中的内容，而不用再次进行运算。这样，通过缓存就会明显提高服务器的响应速度。

使用缓存的时候需要注意的是：

缓存寿命

如果页面缓存时间过长，它们也许不能及时反映出站点里的最新信息。如果站点内容更新很快，可能会因为缓存中包含旧的信息这样使得缓存隐藏了新信息，导致用户看不到这些新信息。运行结果在缓存中的时间长短被称为信息的缓存寿命。

个人信息

许多 Web 页面可能是为某个特殊用户而个性化的。很明显，缓存这个信息，然后显示给其他用户会很不好，这是因为涉及个人秘密，还因为其他用户不能得到与自己相关的信息，他将得到其他人的信息。因此，缓存经常从不适用于个性化信息。

通过设置缓存策略，可以解决这些问题。缓存策略允许你控制缓存内容的方式。缓存策略由缓存管理器对象控制。

4.1. 添加缓存管理器

可以像任何其它 Zope 对象那样添加缓存管理器，Zope 自带有两种类型的缓存管理器：

HTTP 加速缓存管理器（HTTP Accelerated Cache Manager）

- HTTP 加速缓存管理器允许你控制一个 Zope 外部 HTTP 缓存服务器，例如 Squid（http://www.squid-cache.org/）。HTTP 加速缓存管理器本身不执行缓存任务，而是设置特殊的 HTTP 头信息，这些信息告诉一个外部的缓存服务器缓存什么。安装一个外部缓存服务器，例如 Squid，超出了本书范围。请查看 Squid 站点更多相关信息。

(RAM) 缓存管理器

RAM 缓存管理器是一种计算机内存中缓存内容的 Zope 缓存管理器。这使得缓存非常快速，但是会引起 Zope 消耗更多的计算机内存。RAM 缓存管理器工作时不需要任何其它外部资源，例如 Squid 服务器。

让我们举例说明，在根文件夹中创建一个名为 CacheManager 的 RAM 缓存管理器。这个缓存管理器适用于整个站点。 现在，单击 CacheManager ，查看它的配置屏幕。在屏幕上给出了一些元素，如下所示：

Title（标题）

缓存管理器的标题。这是可选项。

REQUEST Variables（REQUEST 变量）

这个信息用于存储页面的缓存复本。这是一个高级特性。目前，你保持默认设置 AUTHENTICATED_USER。

Threshold Entries（极限个数）

缓存管理器一次缓存的对象的数量。

Cleanup Interval（清除间隔）

缓存结果的寿命。

目前，保持所有这些条目不动，一般使用默认值就可以了。对于设置缓存管理器已经足够了！

缓存管理器中还有两个视图比较有用，它们是 Statistics（统计）视图和 Associate（关联）视图。Statistics 视图给你显示出缓使用次数，告诉你缓存效率如何。

另外一个视图是 Associate 视图，用于把一个或多个指定类型的 Zope 对象和指定的缓存管理器建立关联。例如，你也许只想让你的缓存管理器缓存 DTML 文档。你可以在 Associate 视图中更改这些设置。

至此，还没有缓存任何东西，你只创建了一个缓存管理器。下一节说明如何缓存实际文档内容。

4.2. 缓存对象

要缓存一个文档，在根文件夹中创建一个名为 Weather 的新 DTML 文档。这个对象将包含一些天气信息。代码如下：

```
<dtml-var standard_html_header>

  <p>Yesterday it rained.</p>

<dtml-var standard_html_footer>
```

现在，单击这个名为 Weather 的 DTML 文档，单击它的 Cache 视图。这个视图让你把这个文档和一个缓存管理器建立起关联。如果你下拉位于视图顶部的选择框，你会看到在前一节所创建的名为 CacheManager 的缓存管理器。选择它作为 Weather 的缓存管理器。

现在，任何人在任何时候访问这个 Weather 文档，得到将是已被缓存的复本。对于一个象 Weather 这样简单的文档来说，没有充分显示出缓存的好处。但是设想一下，Weather 中包含数据库查询。例如：

```
<dtml-var standard_html_header>

  <p>Yesterday's weather was <dtml-var yesterdayQuery> </p>
```

```
<p>The current temperature is <dtml-var currentTempQuery></p>

<dtml-var standard_html_footer>
```

让我们假设 yesterdayQuery 和 currentTempQuery 是 SQL 方法，为了得到昨天的天气预报和当前温度，这两个 SQL 方法各自查询一个数据库。（关于 SQL 方法的更多信息，请见章“关系数据库连通”）让我们假设数据库中的信息每小时更新一次。

现在，没有缓存，每次查看 Weather 文档时都会查询数据库。如果 Weather 文档一个小时以内被查看几百次，那么所有这些几百次的查询结果总包含相同的信息。

如果缓存这个文档，那么只有当缓存过期时才会再次执行查询。默认的缓存时间是 300 秒（5 分钟），因此，执行查询的数量只有往常的十二分之一，这样设置这个文档缓存就节省 91%数据库查询量。这个方法的不好一面是数据可能会落后 5 分钟，但是它通常是可以接受的。

5. 邮件服务

Zope 带有一个用于发送电子邮件的对象，这就是邮件主机对象。通常用在 DTML 中的 sendmail 标记符中。

邮件主机对象可以通过 Python 或 DTML 发送电子邮件。每个邮件主机对象代表了一个邮件服务器。比如，可以把 SMTP 邮件服务器设置成 yourmail.yourdomain.com，这样就可以通过这个服务器发送电子邮件。接下来，举例说明。

要创建邮件主机对象，可以从添加列表中选择 MailHost 。其中默认的设置是，id 为 MailHost ，SMTP 服务器和端口是 localhost 和 25。 然后创建一个 DTML 方法，代码如下：

```
<dtml-sendmail>

From: me@nowhere.com

To: you@nowhere.com

Subject: Stop the madness!

Take a day off, you need it.

</dtml-sendmail>
```

这样就可以通过刚才创建的邮件主机对象发送邮件了。

6. 错误日志服务

错误日志(Error Log)对象通常位于根目录中，id 为 error_log，它提供了调试和错误日志信息。当用户访问站点时，如果出现了错误，就会在这个对象中记录下来。 这个对象包括的选项为：

Number of exceptions to keep(保留例外数量)

默认保留 20 条例外信息，旧的信息会自动删掉。

Copy exceptions to the event log

如果选择了这个选项，会把错误信息复制到系统中 EVENT_LOG_FILE 变量所指定的文件中。

7. 虚拟主机服务

关于虚拟主机服务的详细信息，请查看“虚拟主机服务”一章。

8. 搜索和索引服务

参见“内容搜索与分类”部分。

9. Session 服务

参见“任务（Sessions）处理”部分

第十六章 内容搜索和分类

ZCatalog（目录册）是 zope 内置的搜索引擎，用来对各种 zope 对象进行分类和搜索。你也可以用它搜索外部数据，例如关系型数据、文件和远程网页等。除了搜索，你还可以用 ZCatalog 组织对象。ZCatalog 支持丰富的查询接口，你可以完成全文搜索，也可以一次搜索多个索引。另外，ZCatalog 还可以处理索引对象的元数据（meta-data ）。以下是两个最为普遍的使用模式：

群组目录化

把大量的对象一次性编进目录

自动目录化

当创建对象时把自动编进目录，同时跟踪变化。

1. 群组目录化起步

下面，让我们看看如何使用 ZCatalog 搜索文档。一次性把多个对象编进目录被称为群组目录化。群组目录化包含几个步骤：

创建一个 ZCatalog

创建索引

查找对象并把它们编进目录

创建 Web 搜索界面

1.1. 创建一个 ZCatalog

进入根目录中的 Zoo 文件夹，然后从产品添加列表中选择 ZCatalog。显示 Zcatalog 添加表单，如下图所示。

ZCatalog 添加表单

在表单中输入 Id 为 AnimalCatalog ，然后单击 Add 按钮，就创建了新的 ZCatalog。它的图标看起来象一个上面带有小放大镜的文件夹。通过选中 AnimalCatalog 图标查看 ZCatalog 的 Contents 视图。

ZCatalog 看起来很类似文件夹，但是它多了几个标签。在 Zcatalog 中有六个标签和文件夹是相同的。ZCatalog 有以下视图 Contents、Catalog、Properties、Indexes、 MetaData 、Find Objects、Advanced、Undo、Security 和 Ownership 。当你单击一个 ZCatalog 时，默认显示 Contents 视图。这里，可以添加新对象，和文件夹一样。但并不意味着其中的对象就可以搜索。ZCatalog 可以对任何层次中的对象编制目录，因此需要指定具体的层次。

1.2. 创建索引

为了确定把什么信息编进目录，以及存储信息的位置，我们需要创建 Lexicon（词典）和 Index（索引）。为了实现全文搜索通过 Lexicon 来提供单词存储服务，Index 是用于实现对象的快速搜索。

在刚才创建的 AnimalCatalog 的 Contents 视图中，从添加列表中选择 ZC TextIndex Lexicon，输入 id 为 zooLexicon，如下图所示：

ZCTextIndex Lexicon 添加表单

接下来，我们创建索引，这些索引用来在 ZCatalog 中记录我们所需的信息。点击 ZCatalog 的 Indexes 标签。然后从下拉菜单中选择 ZC TextIndex，然后在新显示的添加表单中输入 id 为 zooTextIdx，在“Field name”字段中输入 PrincipiaSearchSource，它的作用是告诉 ZC TextIndex 对 DTML 文档的正文部分建立索引（ PrincipiaSearchSource 是 DTML 文档和方法对象的一个 API 函数）。下边的 Lexicon 菜单中应该显示 zooLexicon。如下图所示：

ZCTextIndex 添加表单

最后点击 Add 按钮。关于这些选项在后边的章节中将详细讲述。

另外，我们还需要指出 ZCatalog 直接存储对象的那些属性。这些属性被称为元数据（Metadata）。现在，进入 ZCatalog 的 Metadata 标签，添加“id”和“title”两个元数据。

1.3. 搜索并目录化对象

现在已经创建了 ZCatalog 和索引，就可以进入下一步：搜索对象并对它们编制目录。要完成这个例子，需要再添加两个 DTML 文档，这两个文档放在和 AnimalCatalog 对象同一级的文件夹中。假设这两个文档中包含了关于爬行动物和两西牺动物的信息。

第一个文档的 id 为“chilean_frog”，title 为“Chilean four-eyed frog”，正文部分为：

```
The Chilean four-eyed frog has a bright

pair of spots on its rump that look like enormous eyes. When

seated, the frog's thighs conceal these eyespots. When

predators approach, the frog lowers its head and lifts its

rump, creating a much larger and more intimidating head.

Frogs are amphibians.
```

第二个文档的 id 为“carpet_python”，title 为“Carpet Python”，正文部分为：

```
*Morelia spilotes variegata* averages 2.4 meters in length.  It

is a medium-sized python with black-to-gray patterns of

blotches, crossbands, stripes, or a combination of these

markings on a light yellowish-to-dark brown background.  Snakes

are reptiles.
```

网站的访问者应该能够搜索 Zoo 中存储的动物信息。因此应该能够让用于输入关键字以后进行搜索，然后把所有满足要求的文档显示出来。搜索是最有用、最普通的网站功能之一。

刚才创建的那个 AnimalCatalog 能够对所有文档编制目录，从而让用户可以搜索特定的单词。要对文档编制目录，先进入 AnimalCatalog ，然后点击它的 Find Objects 标签。在这个视图中，可以指对象类型。从“Find objects of type”多选框中选择“DTML Document”，然后点击“Find and Catalog”。 ZCatalog 会从当前的位置开始搜索所有的 DTML 文档，包括所有的下级文件夹。例如，如果这个 ZCatalog 位于 /Zoo/ZCatalog，那么文件夹 Zoo 和所有的下级子文件夹都会进行搜索。

如果要搜索的对象非常多，那么所需要的搜索时间会比较多，需要一点耐心。 搜索完成以后，会自动返回到 ZCatalog 视图中，并显示完成情况。 其中会列出已经目录化完毕的对象。也可以点击这些对象的链接进行查看。 现在数据已经存储在了 ZCatalog 的数据库中，接下来需要创建搜索页面。

1.4. 搜索和汇报表表单

要想创建搜索和汇报表单，请确认是在 AnimalCatalog 中，然后从添加列表中选取 *Z Search Interface* 。选取 AnimalCatalog 作为可搜索对象，如下图所示。

创建一个搜索表单

把 Report Id 命名为“ SearchResults ” 并把 “Search Input Id”命名为“ SearchForm ” ，选择“Generate Page Templates”，单击 Add。这将创建两个新的页面模板，名称分别为 SeachForm 和 SearchResults 。 虽然这些对象位于 ZCatalog 中，但是并不会被 Zcatalog 编进目录。AnimalCatalog 只把 DTML 文档编进目录。搜索表单和汇报模板只是 ZCatalog 中用于搜索动物文档的用户界面。 要想搜索 AnimalCatalog ZCatalog，选中 SearchForm ，单击它的 Test 标签。 然后在 ZooTextIdx 中输入一些单词就可以搜索文档。比如，输入单词“Reptiles”。AnimalCatalog 会进行搜索并返回一个结果表格。你还可以进行多个搜索，比如“reptiles OR amphibians”。

2. 配置目录册

比起刚才完成的任务，ZCatalogs 能够胜任远为强大和复杂的搜索任务。让我们看看目录册如何存储信息。这会帮助你调整 ZCatalog，提供你所需要搜索。

2.1. 定义索引

ZCatalog 把与对象相关的信息和内容存储在名为索引的快速数据库中。索引可以非常快速的存储和取出大量的信息。你可以创建不同类型的索引，用于记录对象的不同类型的信息。例如，你可以用一个索引记录 DTML 文档的文本内容，用另外一个索引记录含有指定属性的对象。

当搜索一个 ZCatalog 时，不是在对每个对象逐个搜索。你要是有许多对象，那样就会花费相当多的时间。在你搜索一个 Zcatalog 以前，它先查看对象，并且记录所指定的信息。这个过程被称为索引化。这样就能按照某个标准进行搜索，ZCatalog 就会返回符合标准的对象。 索引类似于书籍中的索引。比如，一本书中的索引，你可以这样查找单词 Python：

Python: 23, 67, 227

单词 Python 在三个页面中出现。zope 索引的工作方式就像这样，区别在于 zope 映射搜索关键字，在这个例子中，对应于单词 Python，列出的将是所有包含这个单词的对象，而不是书中的页码。 在 Zope 2.6 中，可以通过一种新的索引接口添加和删除索引，如下图所示：

管理索引

每个索引都有一个名称，比如 PrincipiaSearchSource，还有类型，比如 ZC TextIndex。

当你把对象编进目录时，ZCatalog 使用每个索引来检测对象。ZCatalog 根据对象的属性和方法，为每个索引找到对象中的值例如，上边例子中的 PrincipiaSearchSource 索引，ZCatalog 调用每个文档的 PrincipiaSearchSource 方法，并且在 PrincipiaSearchSource 索引中记录结果。如果 ZCatalog 不能为索引找到相应的属性或方法，那么就忽略掉。有八种类型的索引

- ZC TextIndex

用于搜索文本。当需要全文搜索的时候，使用这种索引。

FieldIndex

用于搜索含有特定值的对象。当需要搜索日期、数字或特定字符串时使用这种索引。

KeywordIndex

- 用于搜索多个特定值。这种索引类似于 FieldIndex，但可以搜索多个值，而不仅仅是一个。

PathIndex

用于搜索含有指定的URL路径元素的对象。比如，可以搜索所有路径中以 /Zoo/Animals 开始的对象。

TopicIndex

- 用于在已过滤集合（ FilteredSet ）中搜索，每个集合中包含文档的ID， 并且这些ID匹配集合的过滤表达式。使用这种索引可以优化经常访问的搜索。

DateIndex

- 这是 FieldIndex 的子类，专用于搜索日期时间值。

DateRangeIndex

用于搜索属于某一时间范围的对象。

TextIndex

- 旧版本的全文索引，为了解决兼容问题。应使用 ZC TextIndex 。

在本章的后边，我们将更为详细的讲解这些索引。通过 ZCatalog 的 Indexes 视图可以创建新索引。其中，你可以为新索引输入名称和指定类型。这会在 Zcatalog 中创建一个新的空索引。要想在索引中加入新的信息，需要进入 Advanced 视图，然后单击 Update Catalog 按钮。如果对象比较多，这个过程会花费一些时间。

要想从 ZCatalog 中删除一个索引，先选择索引，然后单击 Delete 按钮。这样就会删除索引以及其中所有的内容。这种操作可以撤销。

2.2. 定义元数据（Meta Data）

ZCatalog 不仅可以建立索引信息，还可以在一种被称为元数据表的表格数据库中存储对象信息。元数据表类似于关系型数据库中的表，由一列或多列组成，定义了表结构。表格中填充以信息行，这些信息与被编进目录的对象相关。根据你的需要，这些行可以包含指定的对象信息。你的元数据列不需要和 ZCatalog 的索引匹配。索引用来进行搜索；元数据用来显示搜索结果。

元数据表用于生成搜索汇报，它可以在汇报表格中显示对象信息。例如，如果你创建了一个名为 Title 的元数据列，那么就可以在汇报表格中显示搜索结果对象的 Title 信息，而不是直接调用对象的 Title 信息。

要想增加一个新的元数据表列，方法是先进入 Metadata 视图，输入列名，点击 Add。要想从元数据表中删除一列，选中列的复选框，然后单击 Delete 按钮。这样就会删除列和每行中对应的所有内容。这种操作是可以撤销的。下一步，让我们更为仔细的看看如何搜索一个 ZCatalog。

3. 搜索目录册

搜索 ZCatalog 的方式是把搜索关键字传递给 ZCatalog。这些搜索关键字描述了在索引中要查找什么。ZCatalog 能够从 Web 请求中收集这些信息，或者也可以通过 DTML 或 Python 明确传递信息。在搜索请求的响应中，ZCatalog 返回搜索结果列表。

3.1. 用表单搜索

在前面的例子中，你使用 Z Search Interface（Z 搜索接口）自动构建了一对 Form/Action 来查询 ZCatalogs。Z Search Interface 构建了一个非常简单的表单和报表。这两个方法有利理解 ZCatalog 如何进行搜索，以及如何定制和扩展搜索接口。

假设你有一个 ZCatalog，用来存储新闻条目。每个新闻条目的属性包括内容、作者和日期。ZCatalog 中相应有三个索引，它们分别对应于这这些属性，即"contentTextIdx", "author" 和 "date"。内容索引是个 ZC TextIndex，作者和日期索引分别是 FieldIndex 和 DateIndex。对于 ZC TextIndex，需要使用一个 ZC TextIndexLexicon。为了显示搜索结果，应该在 Metadata 中加入 author，date 和 absolute_url。以下是用来搜索这个 ZCatalog 的表单：

```
<html><body>

<form action="Report" method="get">

<h2 tal:content="template/title_or_id">Title</h2>

Enter query parameters:<br><table>

<tr><th>Author</th>

<td><input name="author" width=30 value=""></td></tr>

<tr><th>Content</th>

<td><input name="contentTextIdx" width=30 value=""></td></tr>

<tr><th>Date</th>

<td><input name="date" width=30 value=""></td></tr>

<tr><td colspan=2 align=center>

<input type="SUBMIT" name="SUBMIT" value="Submit Query">

</td></tr>

</table>

</form>

</body></html>
```

这个表单由三个名为 contentTextIdx、author 和 date 的输入框组成。这些名称必须和 ZCatalog 中相应的索引名称一致。以下是对应的搜索结果表格：

```
<html>
<body tal:define="searchResults here/NewsCatalog;">
<table border>
  <tr>
    <th>Item no.</th>
    <th>Author</th>
    <th>Absolute url</th>
    <th>Date</th>
  </tr>
  <div tal:repeat="item searchResults">
  <tr>
    <td>
      <a href="link to object" tal:attributes="href item/absolute_url">
        #<span tal:replace="repeat/item/number">
          search item number goes here
        </span>
      </a>
    </td>
    <td><span tal:replace="item/author">author goes here</span></td>
    <td><span tal:replace="item/date">date goes here</span></td>
  </tr>
  </div>
```

```
</table>

</body></html>
```

这个例子的核心是 searchResults 变量：

```
<body tal:define="searchResults here/NewsCatalog;">
```

其中调用 NewsCatalog ，此时不用明确指定 contentTextIdx 、author 和 date。这几个值会自动从搜索表单的 REQUEST 中提供，然后传递给 ZCatalog。 ZCatalog 返回一个搜索结果列表。搜索结果包含 Metadata 表中所设定的信息。

3.2. 通过 Python 进行搜索

用页面模板搜索 ZCatalog 时很简单，只需要注意正确传递参数就可以了。 有些时候，你需要不通过 web 表单来搜索 ZCatalog。例如，假设你想给 Zope Zoo 添加一个工具条，它只显示与当前正在查看的那一部分相关的新闻条目。Zope Zoo 站点是通过文件夹来组织的，不同的文件夹表示不同的动物。每个文件夹的 id 表示了一个指定的群组的名称或一类动物的名称。假设你想让工具条显示当前区域相关的所有新闻条目。以下的 relevantSectionNews 脚本通过使用当前文件夹的 id 来查询 ZCatalog：

```
## Script (Python) "relevantSectionNews"

##

Returns news relevant to the current folder's id """

id=context.getId()

return context.NewsCatalog({'contentTextIdx' : id})
```

这段脚本就像调用方法一样搜索 NewsCatalog 。此时 ZCatalog 需要传递一个参数。这个参数映射索引和搜索条件。在这个例子中，contentTextIdx 索引用来查询包含当前文件夹名称的新闻条目。要在工具条中使用它，只需编辑 Zope Zoo 的主页面模板，调用这段脚本：

```
...

<ul>

  <li tal:repeat="item here/relevantSectionNews">

    <a href="news link" tal:attributes="href item/absolute_url">

      <span tal:replace="item/title">news title</span>

    </a>
```

```
  </li>
</ul>
...
```

在使用这个模板之前，先要确认在 NewsCatalog 的 Metadata 中已经定义了 absolute_url 和 title。

4. 搜索和索引的细节

前边已经看到了 ZCatalog 支持 8 种索引类型。让我们仔细看看这几种索引。

4.1. 搜索 ZCTextIndexe

ZC TextIndex 可用于对文本建立索引。建立索引以后，可以进行全文搜索。ZC TextIndex 支持多种搜索方式。

4.1.1. 布尔表达式

通过布尔表达式搜索，比如：

```
word1 AND word2
```

通过这个表达式可以搜索所有包含“word1”和“word2”的对象。有效的布尔操作符包括 AND、OR 和 NOt。其中 NOT 还可以用“-”替代，比如：

```
word1 -word2
```

这个表达式会搜索包含“word1”但不包含“word2”的文档。没有操作符的单词列表等同于 AND，比如“carpet python snakes”等同于“carpet AND python AND snakes”。

4.1.2. 括号

可用括号控制搜索顺序，比如：

```
(word1 AND word2) OR word3)
```

这个表达式不仅返回包含“word1”和“word2”的对象，还返回只包含“word3”的对象。

4.1.3. 通配符

可以这样使用通配符 *：

```
Z*
```

它表示了所有以“Z”开始的单词，或者：

Zop?

它表示以“Zope”开始但后边是一个任意字母的单词。注意，通配符不能放在最前边，比如“?ope”是不支持的，将被忽略掉。

4.1.4. 词组搜索

使用引号表示词组搜索，比如：

"carpet python" OR frogs

将搜索出包含词组“carpet python”或者单词“frogs”的对象。 上边的这些可以混合使用，比如：

((bob AND uncle) NOT Zoo*)

搜索出的对象将包含单词“bob”和“uncle”，但不包含起始以“Zoo”的单词，比如“Zoologist” 、“Zoology” 、“Zoo”等等。 类似的，还比如：

snakes OR frogs -"carpet python"

搜索出的对象将包含单词“snakes”或“frogs”，但不包含词组“carpet python”。

在Python脚本中，搜索的方式和表单中是类似的，比如对于前边的relevantSectionNews脚本，假设我们想搜索出包含单词“catastrophic”的新闻条目，可以是这样：

```
## Script (Python) "relevantSectionNews"

##

Returns relevant, non-catastropic news """

id=context.getId()

return context.NewsCatalog(

         {'contentTextIdx' : id + ' -catastrophic'}

        )
```

ZC TextIndexe 很强大。如果再和自动目录化模式一起使用，就可以实现更为完整的全文搜索。

4.2. 词典（Lexicons）

ZC TextIndexe 中使用词典。词典可以处理和存储文本中的单词，用来进行全文搜索。 词典的作用包括：

不区分大小写

一般情况下搜索条件不需要区分单词大小写，比如，搜索"python", "Python" 和 "pYTHON"应该是一样的。词典就可以实现这个功能。

删除结束单词

在有些语言中有结束单词（Stop words），应该从索引中删除。

把文本分隔成单词

通过分隔器（splitter）可以把文本分隔成单词。不同的文本使用不同的分隔方法，如果处理 HTML 文档，则可能需要能够识别 HTML 代码的分隔器，从而可以把 HTML 代码删除掉。可有时还需要保留这些 HTML 代码。

要创建 ZC TextIndexe ，需要先创建 Lexicon 对象。多个 ZC TextIndexe 可以共享一个 Lexicon。

4.3. 搜索字段索引

FieldIndexe 略微不同于 ZC TextIndex 。ZC TextIndex 会对文本进行处理，它会把文本分隔成单词，然后对单词建立索引 而 FieldIndex 不会对所找到的值进行处理，而是把所找到的值看成是一个整体。这在搜索那些含有固定值的对象时很有用。在上边的新闻条目例子中，author 就使用了 FieldIndex 。在这个例子中需要输入作者姓名。如果在作者这个地方显示成一个作者下拉列表，则会更方便。 ZCatalog 中有一个 uniqueValuesFor 方法，可以列出索引中全部唯一值。让我们把代码修改成如下代码：

```
<html><body>

<form action="Report" method="get">

<h2 tal:content="template/title_or_id">Title</h2>

Enter query parameters:<br><table>

<tr><th>Author</th>

<td>

  <select name="author:list" size="6" multiple>

    <option
```

```
      tal:repeat="item python:here.NewsCatalog.uniqueValuesFor('author')"
      tal:content="item"
      value="opt value">
    </option>
  </select>
</td></tr>
<tr><th>Content</th>
<td><input name="content_index" width=30 value=""></td></tr>
<tr><th>Date</th>
<td><input name="date_index" width=30 value=""></td></tr>
<tr><td colspan=2 align=center>
<input type="SUBMIT" name="SUBMIT" value="Submit Query">
</td></tr>
</table>
</form>
</body></html>
```

其中，用于列出索引唯一值的代码为：

```
<select name="author:list" size="6" multiple>
  <option
    tal:repeat="item python:here.NewsCatalog.uniqueValuesFor('author')"
    tal:content="item"
    value="opt value">
```

```
  </option>
</select>
```

这段代码把可以把所有的作者姓名列出来。如下图所示：

搜索一个作者

4.4. 搜索关键字索引

KeywordIndex 可以建立起一种关键字序列索引。只要对象包含一个或多个关键字就可以找到这个对象。 假设你有许多图像对象，它们有一个 keywords 属性。keywords 属性是一种多行（lines）属性，这个属性列出了图片相关的关键字，例如，单词 "Portraits", "19th Century", 和 "Women"是图片维多利亚女王的关键字。 关键字提供了一种图片分类的方式。根据 keywords 属性的不同，这些图片属于不同的类别。 你可以使用关键字索引搜索 keywords 属性。在 ZCatalog 中添加一个名为 keywords 关键字索引。然后重新对图片编制目录。然后就可以搜索了。 通过这种方式一个图文可以位于多个类别中，比起 FieldIndex ，这样就增大了搜索和分类对象时的灵活性。 你还可以对关键字索引的建立列表。比如，在这个例子中，可使用 uniqueValuesFor 方法创建定制的搜索表单：

```
<select name="keywords:list" multiple>
  <option
    tal:repeat="item python:here.uniqueValuesFor('keywords')"
    tal:content="item">
      opt value goes here
  </option>
</select>
```

这样就会把现有的关键字列出来，然后你可以从多选框中选择一个或多个关键字进行搜索。

4.5. 搜索路径索引

路径索引用于通过对象的路径进行搜索。假设一个对象的路径是：/zoo/animals/Africa/tiger.doc，你可以通过路径：/zoo 或 /zoo/animals 或 /zoo/animals/Africa 搜索到这个对象。也就是说，通过路径索引可以搜索指定路径中的对象。 如果你把相关的对象放在同一文件夹中，你就可以通过路径索引快速找到这些对象，比如：

```
<h2>Lizard Pictures</h2>

<p tal:repeat="item

    python:here.AnimalCatalog(pathindex='/Zoo/Lizards',

    meta_type='Image')">

  <a href="url" tal:attributes="href item/absolute_url" tal:content="item/title">

    document title

  </a>

</p>
```

这个搜索返回所有位于/Zoo/Lizards 文件夹中的图片文件。注意这个例子中创建了链接，因此需要创建一个名为 meta_type 的 FieldIndex，并且 Metadata 中需要添加 absolute_url 和 title。

4.6. 搜索日期索引（DateIndex）

DateIndexe 就像 FieldIndex，但是只适用于处理日期时间数值。用法和 FieldIndex 一样。

4.7. 搜索日期范围索引（DateRangeIndexe）

DateRangeIndex 适用于搜索属于一定时间范围的对象。就比如新闻条目对象，它有两个 DateTime 属性，即有效时间和失效时间。

4.8. 搜索主题索引（TopicIndex）

TopicIndex 是一种适用于已过滤集合（ FilteredSet ）的容器。 FilteredSet 包含一个表达式和一组内部 ZCatalog 文档 id，这些 id 是为了提高性能而提供的已经完成的搜索结果。这种方式不是执行多次 ZCatalog 搜索，因此速度会快很多。

当编制对象目录或撤销目录时会生成 FilteredSet 。建立索引的时候会检查是否和 FilteredSet 中的表达式匹配。如果匹配则把这个对象添加到 FilteredSet 中。如果目录册发生变化，使得不再匹配 FilteredSet 的表达式，则会从 FilteredSet

中删除。FilteredSet 中已经内置有 PythonFilteredSet ，当然也可以定制新的类型。PythonFilteredSet 使用 eval()函数进行检测。要被放入索引中的对象的引用方法是在表达式中使用"o."，以下是一些例子。 对 DTML 方法建立索引：

```
o.meta_type=='DTML Method'
```

对文件夹并且 title 非空的对象建立索引：

```
o.isPrincipiaFolderish and o.title
```

搜索 TopicIndex 的方法类似于其它索引。比如，通过 Python 搜索 folders_with_titles：

```
zcat = context.AnimalCatalog

results = zcat(topicindex='folders_with_titles')
```

5. 使用 Record 进行高级搜索

从 2.4 版本开始，Zope 支持使用 Record 对象完成更为精确的搜索。Record 对象包含检索方式的信息。Record 是带有属性或映射的 Python 对象。不同的索引支持不同的 Record 属性。

5.1. 关键字索引(KeywordIndex) Record 属性

query

单词序列或一个单词

operator

指定匹配一个还是所有的关键字。允许值：and，or。可选，默认为 or。

例如：

```
# big or shiny

results=ZCatalog(categories=['big, 'shiny'])

# big and shiny

results=ZCatalog(categories={'query':['big','shiny'],

                             'operator':'and'})
```

第二个搜索匹配的对象中包含关键字"big" 和 "shiny"。如果没有 Record，则返回"big"或"shiny"的对象。

5.2. 字段索引(FieldIndex) Record 属性

query

对象序列或一个值

range

定义搜索的范围（可选，默认：没有） 允许值：

min

搜索指定数值大于 min 的对象

max

搜索指定数值小于 max 的对象

minmax

搜索指定数值大于 min 并且小于 max 的对象

例如：

```
# animals with population count greater than 5
zcat = context.AnimalCatalog
results=zcat(population_count={
                'query' : 5,
                'range': 'min'}
            )
```

这个搜索匹配 population_count 属性大于 5 的对象。其中 FieldIndex 为 population_count，对象应该有 population_count 属性。

5.3. 路径索引(PathIndex) Record 属性

query

- 字符串形式的路径，比如"/Zoo/Birds"或 Zoo", "Birds 。

level

路径开始搜索的层次，默认为 0，即从根文件夹开始。-1 表示从路径中的任何地方开始都可以。

假设对象的路径为：

1. /aa/bb/aa
2. /aa/bb/bb
3. /aa/bb/cc
4. /bb/bb/aa
5. /bb/bb/bb
6. /bb/bb/cc
7. /cc/bb/aa
8. /cc/bb/bb
9. /cc/bb/cc

以下显示了一些搜索方式，以及 level 的作用：

query="/aa/bb", level=0

结果和以前的例子一样，即从根文件夹开始搜索，结果如下：

* /aa/bb/aa
* /aa/bb/bb
* /aa/bb/cc

query="/bb/bb", level=0

搜索结果为：

* /bb/bb/aa
* /bb/bb/bb

* /bb/bb/cc

query="/bb/bb", level=1

搜索结果中的对象的路径从向下一层开始匹配：

* /aa/bb/bb

* /bb/bb/bb

* /cc/bb/bb

query="/bb/bb", level=-1

搜索结果中的对象的路径不限制位置:

* /aa/bb/bb

* /bb/bb/aa

* /bb/bb/bb

* /bb/bb/cc

* /cc/bb/bb

query="/xx", level=-1

返回 None

你可以使用 level 属性灵活定制搜索路径。 对于 Zope 2.4.1，使用元组，可以直接包含 level 信息。比如：("/aa/bb", 1)。

5.4. 日期索引(DateIndex) Record 属性

日期索引（ DateIndex ）支持的 Record 属性，包括：

- query
    - 对象序列或一个值

range

- 定义搜索的范围（可选，默认：没有）. 允许值：

- min
    - 搜索指定数值大于 min 的对象

max
    - 搜索指定数值小于 max 的对象

minmax
    - 搜索指定数值大于 min 并且小于 max 的对象

还以我们前边的新闻条目为例，如果搜索指定日期的新闻条目，必须准确输入日期。通过使用 range 属性，可以搜索日期属于一定范围的对象。比如：

```
# return NewsItems newer than a week

zcat = context.NewsCatalog

results = zcat( date={'query' : ZopeTime() - 7,

                      'range' : 'min'

              })
```

5.5. 日期范围索引(DateRangeIndex) Record 属性

DateRangeIndex 只支持 query 属性。

5.6. 主题索引(TopicIndex) Record 属性

就像 KeywordIndex ， TopicIndex 支持 operator 属性：

- operator
    - 指定是否匹配所有的 FieldSet 还是匹配一个。允许值：and，or。（可选，默认为 or）

5.7. ZCTextIndex Record 属性

由于 ZC TextIndex 操作符已经可以嵌入到搜索字符串中，因此 ZC TextIndex 没有 Record 属性。

5.8. 用 HTML 创建 Record

通过使用 HTML 表单可以进行 Record 查询。比如：

```
<form action="Report" method="get">

<table>

<tr><th>Search Terms (must match all terms)</th>

    <td><input name="content.query:record" width=30 value=""></td></tr>

    <input type="hidden" name="content.operator:record" value="and">

<tr><td colspan=2 align=center>

<input type="SUBMIT" value="Submit Query">

</td></tr>

</table>

</form>
```

6. 自动目录化

当创建、更改或删除对象时，通过自动目录化技术可以实现自动更新 ZCatalog 中的数据。此时，ZCatalog 可以自动跟踪对象的变化。

和群组目录化相比，自动目录化在很多情况下显得更方便。尽管群组目录化很简单，但它有缺点。在一个事务处理中所能编制索引的内容总数等于现存的留给 Zope 过程的自由虚拟内存数加上系统拥有的临时存储空间数。换句话说，你一次想编制索引的内容越多，对计算机硬件要求就越高。群组索引化对于编制多达几千个对象的索引工作还不错，但是超过那个数量，自动目录化工作得更好。

自动目录化的另外一个主要优点是可以处理变化的对象。随同对象的改变，索引信息总是最新的，即使对于迅速变化的信息也是如此，例如留言板。

在这一节，我们将展示一个创建“新闻”条目的例子，人们可以把它添加到你的站点中。这些条目自动被编进目录。这个例子由两步组成：

- - 创建一个新对象类型。 - 创建用来处理新类型对象的 ZCatalog

对于这个例子首先要定义新对象类型。一种方式是通过定义一个 ZClass 来完成。 ZClass 是一种 Zope 对象，它定义新的对象类型。从某种角度来说，ZClass 就像描述如何构建对象的蓝图。就比如我们前边讲到的新闻条目。新闻条目不仅具有内容，它们还有特有的属性。你要构建的新闻站点能够在线收集新闻条目，然后进行审核，再把它们发布到 Web 站点中去。

在这种系统中，需要创建名为News Item的新对象类型。通过这种方式，当你需要给你的站点添加一个新的News Item时，只要从产品添加列表中选取它就可以。如果你把这种对象设计成自动目录化，那么你可以非常方便的搜索新闻内容。对于这个例子，我们用到一点ZClass知识，有关Zclass将在“扩展Zope”中有详细讲述。

在控制面板的产品部分定义新对象类型。先单击控制面板，然后单击产品管理，就可以到达这个部分。产品包含新类型的ZClass。在这个屏幕中，单击Add来添加一个新产品，将出现新产品的添加表单。

给新产品命名为 NewsItem ，然后单击Generate，会返回到产品管理视图，在此可以看见新添加的产品。

通过单击选中 NewsItem 产品。这个新产品看起来非常象一个文件夹。它包含一个Help的对象，还有一个Add菜单以及横贯顶部的类似于通常的文件的标签。要添加一个新ZClass，下拉Add菜单选择ZClass。然后会出现ZClass 添加表单，如下图所示。

-

ZClass 添加表单

这是一个有点复杂的表单，详细内容将在“扩展Zope”一章中阐述。目前，创建新的ZClass只需做三件事：

- - 指定Id为“ NewsItem ”，这是新的ZClass的名称。 - 指定Meta_Type为“News Item”，这用于为新对象类型在添加菜单中创建条目。 - 从左边的Base Classes框中选择 CatalogPathAware，然后单击指向右边Base Classes框的箭头按钮。这样就会让 CatalogPathAware在右边窗口中显示出来。注意，如果要继承多个类，需要把CatalogPathAware放在第一位，特别是要放在ObjectManager前面。

完成以后，不要修改表单中的其它任何设置，要创建新的ZClass，继续单击Add。然后会返回到 NewsItem 产品。注意，一个名为 NewsItem 的新对象出现了，同时还有几个其它对象。 NewsItem 对象是新的ZClass。其它对象是辅助对象，这些内容将在“扩展Zope”一章里进行详尽的讲述。

选中 NewsItem ZClass对象。结果如下图所示。

-

ZClass Methods 视图

这是ZClass的Methods视图。这里，你可以添加方法对象。例如，你可以创建页面模板或脚本，这些对象即成为 NewsItem 的方法。在创建任何方法以前，让我们回顾一下新的 NewsItem 对象的要求：

- 新闻内容
    - 新闻条目中包含新闻内容，这是它主要的目的。这些内容可以是任何类型的纯文本或者有标记的内容例如HTML或XML。

作者版权

- 新闻条目应该提供作者版权信息。

日期

- 新闻条目是有时间性的，因此创建日期是必须要有的。

以上这些仅是建议，你也许还需要其它属性。要添加新属性，单击Property Sheets标签。这会进入到Property Sheets视图

属性单（property sheets）用来给新对象类型按组别添加属性。由于你的对象没有定义属性单，目前这个视图为空。要添加一个新属性单，单击Add Common Instance property sheet（添加普通实例属性单），并给它命名为News。现在单击Add。这样就给对象添加一个名为News的新属性单。单击这个属性单会进入到Properties视图，如下图所示。

-

属性单的属性屏幕

这个视图几乎等同于在文件夹和其它对象中所看到的Properties视图。这里，可以创建新闻条目对象的属性。继续在这个表单里创建以下三个新属性：

- content（内容）
    - 这个属性类型应该为文本型（text）。每个新创建的新闻条目都包含content属性。

author（作者）

- 这个属性类型应该为字符型（string）。它用来显示新闻的作者。

date（日期）

- 这个属性类型应该为日期型（date）。它用来显示更新的时间和日期。date属性必须输入一个值，目前，你可以输入字符串“01/01/2000”。

好了！现在已经创建了一个属性单，它描述了新闻条目，以及包含什么类型的信息。属性可看作是对象包含的数据。现在我们设置好了数据。接下来，需要给你的新对象类型创建界面。这通过为你的对象创建新视图来实现，即通过添加一对Form/Action页面来实现。通过这样的页面就可以编辑属性。我们需要为对象创建自己编辑表单。

首先，我们需要创建用来显示和编辑属性的表单。点击Methods标签。从添加下拉列表中选择“Page Template”，命名为editPropertiesForm，然后输入：

```
<html><head>
```

```
<title tal:content="here/title_or_id">title</title>
<link rel="stylesheet" type="text/css" href="/manage_page_style.css">
</head>
<body bgcolor="#FFFFFF" link="#000099" vlink="#555555">
<span
  tal:define="manage_tabs_message options/manage_tabs_message | nothing"
  tal:replace="structure here/manage_tabs">
  prefab management tabs
</span>
<form action="manage_editNewsProps" method="get">
<table>
<tr>
  <th valign="top">Content</th>
  <td>
    <textarea
      name="content:text" rows="6" cols="35"
      tal:content="here/content">content text</textarea>
  </td>
</tr>
<tr>
  <th>Author</th>
  <td>
```

```
    <input name="author:string"

           value="author string"

           tal:attributes="value here/author">

  </td>

</tr>

<tr>

  <th>Date</th>

  <td>

    <input name="date:date"

           value="the date"

           tal:attributes="value here/date">

  </td>

</tr>

<tr><td></td><td>

<input type="submit">

</td></tr>

</form>

</body>

</html>
```

这样就创建了Form/Action中的Form部分。注意，其中的manage_tabs是用来显示管理界面中的标签的。

接下来完成Action部分。添加一个脚本对象，id为manage_editNewsProps，代码为：

```
# first get the request
```

```
req = context.REQUEST

# change the properties in the zclass' propertysheet

context.propertysheets.News.manage_editProperties(req)

# signal the change to the zcatalog

context.reindex_object()

# now return a message

form = context.editPropertiesForm

return form(REQUEST=req,

            manage_tabs_message="Saved changes.",

            )
```

创建完成。接下来，我们定义视图。点击 Views 标签，进入 Views 视图。 这里，默认已经创建了三个视图。这些视图将在“扩展 Zope”一章中详细讲述。 要创建新的视图，可以使用 Views 视图中底部的表单。其中输入名称“News”，然后选择“editPropertiesForm”，再点击 Add 按钮。如下图所示：

-

Views 视图

我们需要让这个视图成为新闻条目的第一个视图。因此需要调整它的顺序。选择新创建的News 视图，然后点击First 按钮。这样就会排到第一个位置。

最后一步则是定义一个用来显示的方法。单击Methods 标签，从添加列表中选择页面模板，然后添加 id 为 index_html 的页面模板。这是新闻条目的默认视图。添加以下代码：

```
<html><head>

<title tal:content="template/title">The title</title>

</head><body>

<h1>News Flash</h1>

<p tal:content="here/date">

  date goes here

</p>

<p tal:content="here/author">

  author goes here

</p>

<p tal:content="here/content">

  content goes here

</p>

</body></html>
```

最后，我们将这这个显示方法添加一个新的管理标签。点击Views 标签，创建一个名为“View”的新视图，并分配给index_html。然后让它排在News 后面。

好了，目前已经创建了一个名为News Item 的新对象类型。当重新进入根目录，在添加列表中会看到这个条目。

然而，现在还不能添加任何新的News Item，这是因为在这个练习中的下一步还需要要创建一个 ZCatalog，用它把新闻条目编进目录。进入根文件夹，创建一个 id 为 Catalog 的新目录册。ZClass 通过获取机制搜索名为 Catalog 的 ZCatalog，因此这个ZCatalog 应该放在所有的新闻条目都能找到的地方。

就像前边的两个使用 Zcatalog 的例子，你需要创建索引和一个元数据表，用它们感知对象。首先创建以下索引：

- content（内容）
    - 应该使用 ZC TextIndex 。它对新闻条目内容建立索引。

author（作者）

- 应该使用 FieldIndex 。它对新闻条目的作者建立索引。

date（日期）

- 应该使用 DateIndex 。它对新闻条目的日期建立索引。

当创建完这些索引以后，添加以下 Metadata 列：

- - author - date - absolute_url

创建完成以后，最后一步则需要为这个 ZCatalog 创建搜索界面。方法是使用 Z Search Interface 工具，这些内容在前边讲述过。

现在，我们就可以测试一下了。先添加一些新闻条目进来。进入到任何一个文件夹中，从添加列表中选择 News Item，应该出现添加表单。 输入 News Item 的 id 为“ KoalaGivesBirth ”，然后单击 Add。这样就创建了一个新 News Item。选择新 News Item。 注意它有四个标签，分别匹配 ZClass 中的四个视图。第一个视图是 News；这个视图对应于你在 News Item ZClass 中所创建的 News Property Sheet。 在 contents 框中输入新闻：

Today, Bob the Koala bear gave birth to little baby Jimbo.

在 Author 框中输入你的名字，在 Date 框中输入今天的日期。 单击 Change，现在你的 News Item 就应该包含一些新闻了。因为 News Item 对象是 CatalogPathAware ，所以当它变改或添加时，Catalog 中的数据随之变化。通过查看 Catalog 对象的 Cataloged Objects 标签可以验证这一点。 刚才添加的 News Item 是唯一被目录化的对象。当添加更多的 News Items 以后，它们在此被自动的目录化。你可以多添加几个条目，然后搜索 Zcatalog 试试。例如，如果你搜索 Koala，你应该找到新闻条目 KoalaGivesBirth 。

到此，你可能需要使用一些本章前边讲过的更高级的搜索表单。例如，在搜索表单中，作者输入框变成作者选择列表。

7. 结论

通过使用 ZCatalog 可以非常方便的完成对象搜索功能。特别对于那些内容多的站点显得更为实用。ZCatalog 搜索非常类似于搜索关系型数据库，区别在于这种方式更加面向对象。在很多情况下使用 ZCatalog 就可以了，但是有些时候需要使用关系型数据库。下一章“关系数据库连通”详细讲述 Zope 如何处理关系数据库和如何把关系数据当作对象来使用。

第十七章 关系数据库连通

Zope 使用对象数据库来存储各种页面、文件和其它对象。它很快并且几乎不需要额外的设置和维护。和文件系统一样，它特别适用于存储像图片这样的中等规模的数据。而关系型数据库则使用另外一种方式。它们基于数据表格，比如：

Row	First Name	Last Name	Age
1	Bob	McBob	42
2	John	Johnson	24
3	Steve	Smith	38

表格中的信息按行储存。表格的列布局称为结构。在关系数据库中查询和更改表格使用结构化查询语言（SQL）。本章需要了解基本的 SQL 知识，如果你还不清楚 SQL，请先看看相关的书籍。

关系型数据库和对象数据库各自都有优点和缺点。Zope 对这两种方式都支持，从而可以让你灵活的选择合适的存储方式。使用关系型数据库的原因是为了访问现有的数据库或为了和其它应用程序共享数据。很多编程语言和软件都支持关系型数据库。尽管通过其它语言或应用程序也可以访问 ZODB，但相对来说，要麻烦一点。

通过 Zope 来使用关系型数据库，可以保留 Zope 的所有好处，包括安全、动态表现、网络功能等等。你可以使用 Zope 动态控制数据访问、数据表现和数据管理。

1. 常用的关系型数据库

有许多种关系型数据库系统。以下是常见的几种：

- Oracle
    - Oracle 是一种强大而流行的商业关系数据库。但是，相当昂贵复杂。Oracle 可以通过 Oracle Web 站点（ http://www.oracle.com/)进行购买或试用。

DB2

- IBM 的 DB2 是 Oracle 的主要商业竞争者，功能类似，昂贵复杂。更多信息可参考 http://www.ibm.com/software/data/db2/

PostgreSQL

- PostgreSQL 是领先的开放源码的支持 SQL 标准的关系型数据库。在 PostgreSQL 站点可以找到更多信息。

MySQL

- MySQL 是一种快速的开放源码的关系型数据库。在 MySQL 站点可以找到更多信息。

SAP DB

- 由 SAP 开发的开放源码的数据库。提供兼容 Oracle7 模式。可参考 http://www.sapdb.org/。

Sybase

- Sybase 是另外一个流行的商业关系型数据库。可参考 Sybase 的网站。

SQL Sever

- Microsoft 提供的在 Windows 操作系统种运行的 SQL Server。可参考 http://www.microsoft.com/sql/。

Interbase

- Interbase 是由 Borland/Inprise 提供的开放源码的关系型数据库。在 Borland 站点可找到更多信息。另外，还可以使用 FireBird 。

Gadfly

- Gadfly 是用 Python 编写的关系型数据库。Zope 中已经包含了 Gadfly，适用于演示和存储小型数据。Gadfly 很快，但由于它把整个数据存在内存中，因此不适合处理大量数据。更多信息可查看 http://gadfly.sourceforge.net/.

关于这些数据库的安装、设置和使用请参考相关站点或书籍。Zope 可以连接使用所有的这些数据库系统。在连接以前，请先确认数据库是否已经启动。Gadfly 不需要安装和设置。

2. 数据库适配器

要想使用数据库，必须要有相应的数据库适配器。数据库适配器可以从 Zope.org 的产品部分下载。Zope 中已经带有 Gadfly 适配器。目前经常使用的几种数据库的适配器如下：

- Oracle
  - DCOracle2 模块由 Zope 公司提供，其中包括 ZoracleDA。

DB2

- ZDB2DA ，由 Blue Dynamics 提供。

PostgreSQL

- 最新的是 ZPsycopgDA，包含在 psycopg 模块中

MySQL

- ZMySQLDA

SAP DB

- ZsapdbDA 由 Ulrich Eck 提供

Sybase

- SybaseDA 由 Zope 公司提供。

SQLServer

- ZODBC DA 由 Zope 公司编写。适用于 Windows 系统，以及支持 ODBC 的数据库。

Interbase/Firebird

- 现有几个 DA ，包括 isectZope, kinterbasdbDA 和 gvibDA

Informix

- ZinformixDA

Gadfly

- Zope 中已经包含了 Gadfly 适配器

如果你需要连接多个数据库或者希望让不同的用户连接同一数据库，那么可以建立多个数据库连接对象。

3. 设置数据库连接

数据库适配器下载并安装以后，通过从添加列表中选择数据库连接对象来连接数据库。所有的数据库连接管理界面都非常类似。通过使用数据库连接对象，可以建立和管理数据库连接。由于数据库在 Zope 外部运行，因此需要指定必要的连接信息，即连接字符串。每种数据库的连接字符串都会有所不同。比如，下图中显示了 PostgreSQL 数据库连接表单。 数据库连接对象必须在定义数据库方法以前建立。而且，每个 Z SQL 方法必须和一个数据库连接相关联。现有许多针对不同数据库的数据库适配器（简写为 DA）：

-

PostgreSQL 数据库连接

本章中的例子将使用Gadfly数据库。不管使用什么数据库，基本的使用方法是相同的，大多数都有“Test”和“Browse”标签，分别用于进行连接测试和显示数据库结构。

从添加列表中选择“Z Gadfly Database Connection”，会出现Gadlfy数据库连接添加表单。选择并添加一个连接。注意此时，并不需要指定连接字符串，这是因为Gadfly在Zope内部运行。

此时还要选择Demo作为数据源，id指定为Gadfly_database_connection，然后点击Add按钮。这样就会创建一个新的Gadfly数据库连接。建立好以后，点击这个连接。

出现Gadfly数据库连接的Status视图。这个视图告诉你是否已经连接了数据库，并且提供了一个用于连接或断开的按钮。通常，Zope会自动控制数据库连接，因此，一般不需要手工控制连接。对于Gadfly，连接和断开连接是无意义的。然而，对于外部数据库，你就可能需要手工连接或断开连接来维护数据库。

下一个视图是Properties视图。这个视图显示了数据源和其它一些属性。通过修改这些属性，就可以切换数据源。下图显示了Properties视图。

-

Properties 视图

通过 Test 视图，可以测试数据库连接。在这个视图中，可以直接键入并运行 SQL 代码。这个视图用于测试数据库，以及执行一次性 SQL 命令（例如创建表格）。这里不是输入大量 SQL 代码的地方。SQL 命令一般位于 Z SQL 方法中，在后边章节中将详细讲述。

为了完成本章的例子，让我们在数据库中创建一个表格。通过 Test 视图可用来直接把 SQL 语句发送给数据库。通过在 Test 视图中直接键入 SQL 代码可以创建表格，不需要使用 SQL 方法来创建表格。用以下 SQL 代码创建一个名为 employees（雇员）的表格

```
CREATE TABLE employees
(
emp_id integer,
first varchar,
last varchar,
salary float
)
```

单击 Submit Query 按钮，运行这个 SQL 命令。Zope 应该返回一个确认屏幕，它告诉你运行了什么 SQL 代码以及可能会产生的运行结果。

根据数据库的不同，这里所使用的 SQL 可能不同。有关创建数据库表格的详细方法，请查看相关数据库的说明文档。

上边的 SQL 在 Gadfly 数据库中创建名为 employees 表格。这个表格有四列：emp_id、first、last 和 salary。第一列是雇员的 id，它是唯一的数字，用以区分雇员。接下来的两列属于 varchar 类型，这种类型类似于字符串型。salary 列属于 float 类型用于存储浮点数字。每种数据库支持不同的类型，因此，请参考相关数据库的使用文档，明确数据库所支持的类型。

要测试这个表格，进入 Browse 视图。这个视图使你能够观看数据库的表格，以及每个表格的结构。这里，你可以看到有一个 employees 表，如果单击加号，表格展开显示四个列：emp_id、 first、 last 和 salary，如下图所示。

-

浏览数据库连接

当使用许多大表格创建复杂的 SQL 应用程序时，这个信息非常有用，这是因为它能够展现表格的结构。不是所有的数据库都支持表格浏览。

现在，你创建了一个数据库连接对象，并且定义了一个表格，接下来就可以通过创建 Z SQL 方法来操作数据库。

4. 使用 Z SQL 方法

ZSQL 方法是通过数据库连接对象执行 SQL 代码的对象。所有的 Z SQL 方法必须关联一个数据库连接对象。Z SQL 方法可以查询数据库和更改数据。Z SQL 方法可以包含多个 SQL 命令。

ZSQL 方法有两个功能，一个是生成 SQL，一个是把数据库的反映转换成对象。它的好处在于：

- 生成的 SQL 将处理特殊的字符，这些字符需要加上引号或删除。这样就提高了代码编写速度。
- 如果后台的数据库更换了，比如从 PostgreSQL 换成了 Oracle，那么生成的 SQL 会自动调整，这样可以让应用程序更容易移植。

- 查询的结果被打包成容易处理的对象，这样就可以轻松的显示和处理结果。
- 可充分使用事务处理。本章后边将详细讲述事务处理。

4.1. Z SQL 方法举例

例如，我们要创建一个新的名为 hire_employee 的 Z SQL 方法，它在 employees 表格中插入一条记录。当雇佣一个新雇员时调用这个方法，并在 employees 表格中插入一条包含新雇员信息的记录。从添加列表中选择 Z SQL Method，然后会出现 Z SQL 方法的添加表单，如下图所示。

-

添加表单

除了要为 Z SQL 方法指定 id 和 title 外，还需要选择数据库连接对象。这个新方法的 id 命名为 hire_employee，然后选择在前边创建的 Gadfly_database_connection。

接下来，你可以给这个 Z SQL 方法指定参数。就像脚本一样，Z SQL 方法可以带有参数。参数可用于构造 SQL 语句。在这个例子中，需要四个参数：雇员 id 号、名字、姓氏和雇员的工资。在参数框中键入“emp_id first last salary”。你可以让每个参数占据一行，或者把多个参数放在同一行中，其间用空格分开。你还可以为参数提供默认值，就像 Python 脚本那样。例如，empid=100，它赋予参数 empid 默认值 100 。

在最后一个 Query 模板输入以下代码：

```
insert into employees (emp_id, first, last, salary) values

(<dtml-sqlvar emp_id type="int">,

 <dtml-sqlvar first type="string">,
```

```
 <dtml-sqlvar last type="string">,

 <dtml-sqlvar salary type="float">

)
```

注意这段代码还包含有DTML。其中的DTML代码可以把参数值插入到SQL代码中，而在数据库中执行生成的SQL代码。如果参数emp_id的值为42，first参数的值为Bob，last参数的值为Uncle，salary参数的值为50000.00，那么查询模板将创建以下SQL代码：

```
insert into employees (emp_id, first, last, salary) values

(42,

 'Bob',

 'Uncle',

 50000.00

)
```

查询模板和SQL特定的DTML标记符在下一节中进行详细讲述。

视图中下边的三个按钮提供了几种添加方式。其中，Add按钮可创建方法并返回到包含新方法的文件夹。Add and Edit按钮可创建方法并显示编辑视图。Add and Test按钮创建方法并返回到Test视图，从而可以测试这个新方法。要添加新Z SQL方法，单击Add按钮。

现在你有了一个在employees表格中插入新雇员的Z SQL方法。你还需要另外一个Z SQL方法，用来在表格中查询雇员。创建一个id为list_all_employees的Z SQL方法。它不用参数，SQL代码为：

```
select * from employees
```

这个简单的SQL代码能够从employees表格中选择所有的行。现在你有了两个Z SQL方法，一个用于插入新雇员，一个用于观看数据库中所有的雇员。让我们通过在employees表格中插入一些新雇员并列出他们来测试这两个新方法。要进行这个测试，单击hire_employee方法，然后单击Test标签。这将返回到Test视图，如下图所示。

-

hire_employee Test 视图

这里，你看到一个带有四个输入框的表单，它们对应于hire_employee方法的每个参数。根据Z SQL 方法的参数，Zope自动生成这个表单。因为hire_employee有四个参数，Zope自动创建这个带有四个输入框的表单。你可以测试这个方法，方式是为新雇员输入一个雇员号、名字、姓氏和工资。雇员id中输入42，在名字中输入Bob，在姓氏中输入 McBob ，在工资中输入50000.00，然后单击Test按钮。就会显示测试结果。

屏幕显示：This statement returned no results（这个语句没有返回结果）。这是因为hire_employee方法只在表格中插入新记录，它不从表格中选择任何信息，因此没有返回记录。这个屏幕还显示了查询模板如何生成SQL。DTML标记符sqlvar把这四个参数转换成有效的数据库中可执行的SQL代码。你可以通过随意添加一些雇员来测试这个方法。

要检验添加的信息是否插入到了表格中，选择list_all_employees方法，然后单击它的Test标签。 这个视图显示：This query requires no input（这个查询不需要输入），说明list_all_employees没有任何参数。单击Submit查询按钮，测试这个方法。

list_all_employees方法返回employees表格的内容。你可以看到所有雇员。Zope自动为你生成这个结果表格。接下来，我们给你展示如何创建WEb站点中的用户界面。

4.2. 通过Z SQL方法显示结果

查询关系数据库返回结果序列。序列中的数据项称为结果行。SQL查询结果通常是一个序列。如果SQL查询只返回一行，结果列表中就只有一个数据项。

由于Zope是面向对象的，所以Z SQL方法返回的是一个结果对象。所有返回的行被打包成一个对象。可以这样来理解，即结果对象可看成是数据库中已经转化成对象的行。这些对象具有与结果数据库结构相匹配的属性。

DTML中通过调用一个Z SQL方法来显示搜索结果。例如，给站点添加一个名为listEmployees的DTML方法，其代码如下：

```
<dtml-var standard_html_header>
  <ul>
  <dtml-in list_all_employees>
    <li><dtml-var emp_id>: <dtml-var last>, <dtml-var first>
      makes <dtml-var salary fmt=dollars-and-cents> a year.
    </li>
  </dtml-in>
  </ul>
<dtml-var standard_html_footer>
```

这个方法调用 list_all_employee 方法。in 标记符用于迭代由 list_all_employees 方法返回的结果对象。Z SQL 方法总是返回对象列表，因此可以通过 DTML in 标记符调用它们。

in 标记符中间部分决定了如何何显示每个搜索结果。比如，在那个只有三个雇员的表格的例子里，listEmployees 可返回这样的 HTML：

```
<html>
  <body>
  <ul>
    <li>42: Roberts, Bob
      makes $50,000 a year.
    </li>
    <li>101: leCat, Cheeta
      makes $100,000 a year.
    </li>
    <li>99: Junglewoman, Jane
```

```
        makes $100,001 a year.

      </li>

    </ul>

    </body>

</html>
```

通过 in 标记符，用 HTML 列表显示 list_all_employees 返回的结果。

不同的数据库适配器处理不同的数据库类型时会有些不同。由于 SQL 中的类型比 Python 要多，因此返回的数据类型只能尽量接近 SQL 类型。比如，对于数据库中的日期类型，通常返回 Zope 中的 DateTime 对象，而数据库中的 char,varchar 和 text 类型都返回 strings 类型。

结果对象和其它 Zope 对象之间的区别在于结果对象不需要创建，以及持久的保存。结果对象不能长时间存在。只要搜索完毕结果对象就消失。下次搜索时则会建立新的结果对象。

接下来，让我们看看如何创建用户界面，从而可以收集数据并把它传递给 Z SQL 方法。

4.3. 给 Z SQL 方法提供参数

目前为止，使用 listEmployees DTML 方法可以显示雇员，这个 DTML 方法调用 list_all_employees 方法。现在，让我们看看如何为 Z SQL 方法 hire_employee 构建一个用户界面。hire_employee 接受四个参数：emp_id、first、last 和 salary。hire_employee 方法中的 Test 标签使你能够调用这个方法，但是还不能把它结合进一个 Web 应用程序中。你需要为 Z SQL 方法创建你自己的输入表单或者从应用程序中调用它。

Z Search Interface（Z 搜索界面）可以自动创建一个输入表单。在“内容搜索和分类”中，你使用 Z Search Interface 构建了一个 form/action 方法对，它自动生成一个用于查询 ZCatalog 的 HTML 搜索表单和返回结果的报表屏幕。Z Search Interface 还可以配合 Z SQL 方法，从而可以构建一套类似的 search/result 屏幕。

从添加列表中选择 Z Search Interface，然后指定 hire_employee 作为 Searchable object。在 Report Id 中输入值 hireEmployee，在 Search Id 中输入 hireEmployeeForm，然后单击 Add。

单击新创建的 hireEmployeeForm，然后单击 View 视图。为每个新雇员输入一个 employee_id、名字、姓氏和工资。Zope 返回：There was no data matching this query（没有匹配这个查询的数据）。这是因为由 Z Search Interface 生成的报告表单的意思是要显示一个 Z SQL 方法的结果，而 hire_employee 方法不返回任何结果，它只在表格中插入一个新行。让我们编辑一下 hireEmployeeReport ,让它更有用。选择 hireEmployeeReport 方法。修改代码如下：

```
<dtml-var standard_html_header>

<dtml-in hire_employee size=50 start=query_start>
```

```
<dtml-if sequence-start>
   <dtml-if previous-sequence>
     <a href="<dtml-var URL><dtml-var sequence-query
              >query_start=<dtml-var
              previous-sequence-start-number>">
     (Previous <dtml-var previous-sequence-size> results)
     </a>
   </dtml-if previous-sequence>
   <table border>
     <tr>
     </tr>
</dtml-if sequence-start>
     <tr>
     </tr>
<dtml-if sequence-end>
   </table>
   <dtml-if next-sequence>
      <a href="<dtml-var URL><dtml-var sequence-query
        >query_start=<dtml-var
         next-sequence-start-number>">
      (Next <dtml-var next-sequence-size> results)
      </a>
```

```
        </dtml-if next-sequence>

      </dtml-if sequence-end>

  <dtml-else>

    There was no data matching this <dtml-var title_or_id> query.

  </dtml-in>

  <dtml-var standard_html_footer>
```

这是一段相当长的DTML！所有的DTML的意思是要动态构建面向批块（batch-oriented）的表格结果表单。因为我们不需要这些，让我们把hireEmployeeReport 方法修改得简单些：

```
<dtml-var standard_html_header>

<dtml-call hire_employee>

<h1>Employee <dtml-var first> <dtml-var last> was Hired!</h1>

<p><a href="listEmployees">List Employees</a></p>

<p><a href="hireEmployeeForm">Back to hiring</a></p>

<dtml-var standard_html_footer>
```

现在观看hireEmployeeForm，并输入另外一个新雇员。注意hire_employee方法是如何通过DTML call 标记符调用的。这是因为我们知道hire_employee方法没有输出。因为没有结果需要迭代，就不需要用in标记符来调用这个方法。只要简单的用call标记符调用就可以了。

现在你有了完整的用于雇佣新雇员的用户界面。使用Zope的安全系统，可以限制访问这个方法，只让某一群你赋予权限的用户来雇佣新雇员。记住，由Z Search Interface生成的搜索和报表屏幕只是简单的框架，你可以根据需要进行定制。

接下来，我们将看看如何精确的控制SQL查询。

5. 动态SQL查询

Z SQL方法中的查询模板中可以包含DTML，当调用这个方法时执行这些DTML代码。通过使用DTML可以调整SQL代码。主要使用sqlvar, sqltest 和 sqlgroup 标记符。

5.0.1. 使用 Sqlvar 标记符插入参数

特别需要注意的是，在数据库的列里应当插入数据类型一致的数据。如果使用字符型的“12”，而本应该为整数型 12，数据库会报错。SQL 要求不同的数据类型使用不同的引用方式，并且不同的数据库采用不同的引用规则。除了避免错误，SQL 引用对于安全也很重要。假设你有一个执行选择的查询：

```
select * from employees

  where emp_id=<dtml-var emp_id>
```

这个查询是不安全的，这是因为有可能因为类型不匹配导致 SQL 代码无效，使表格 employees 报错。要避免这种问题，你需要确信正确引用了变量。sqlvar 标记符就起到这个作用。把上边的代码换成：

```
select * from employees

  where emp_id=<dtml-sqlvar emp_id type=int>
```

sqlvar 标记符类似于通常的 var 标记符。然而，它有一些特定属性，适合处理 SQL 类型引用或空值。sqlvar 标记符可接受的参数包括：

name

- name 参数等同于 var 标记符中的 name 参数。这是一个 Zope 变量或 Z SQL 方法参数。变量或参数的值被插入到 SQL 查询模板。name 参数是必需的，但是前缀“name=”可以忽略。

type

- type 参数决定将要插入到查询模板中的变量值或参数的格式。有效的类型有 string、int、 float 或者 nb。nb 指非空（non-blank），即要求一个字符串至少要有一个字符。sqlvar 标记符的 type 参数是必需的。

optional

- optional 参数用来表示变量或参数可以空缺或为空值。如果变量或参数不存在或为一个空值，sqlvar 标记符不处理它。optional 参数是可选的。

type 参数是 sqlvar 标记符的关键特性。它负责正确的引用要插入的变量。对于 sqlvar 标记符的全部说明请参见“DTML 参考”。当把变量插入到一段 SQL 代码中时，应该尽量使用 sqlvar 标记符而不是 var 标记符，这是因为它能正确的引用变量并使 SQL 代码保持安全。

5.0.2. 用 sqltest 进行等式比较

许多 SQL 查询涉及等式比较操作。这些查询搜索符合某种等式关系的值。例如，你可能需要查询 employees 表格中工资大于某个数值的雇员。要完成这个功能，需要创建一个 Z SQL 方法，命名为 employees_paid_more_than。参数为 salary，SQL 模板如下

```
select * from employees

  where <dtml-sqltest salary op=gt type=float>
```

现在，单击Add and Test。op标记符属性设置为gt，它是指大于。这个Z SQL方法返回工资大于指定数值的雇员记录。sqltest构建安全的在输入值和表格列之间进行比较所必需的SQL语句。在工资输入框中输入“10000”，然后单击Test按钮。你可以看到，sqltest标记符运行以下SQL代码：

```
select * from employees

  where salary > 10000
```

sqltest标记符根据指定的类型把这些比较解释为相应的SQL。sqltest标记符接受以下标记符参数：

name

- 要插入的变量名称。

type

- 插入值的数据类型。这个属性是必需的，可以是：string、int、float或 nb。nb数据类型指非空的字符串，并且长度必须大于0。

column

- SQL列的名称。

multiple

- 表示是否可以提供多个值。它用来测试列是否属于变量集合。例如，当name是一个字符串列表，例如“Bob”，“Billy”，那么 <dtml-sqltest name type="string" multiple>，对应的SQL为：name in ("Bob", "Billy")。

optional

- 标明这个测试是否为可选。如果测试是可选的并且没有提供对应的变量值，那么就不会插入文本。如果值是空字符串，那么只有在type为nb时不插入文本。

op

- 用来提供比较操作符。包括：eq（等于）、gt（大于）、lt（小于）、ge（大于或等于）、le（小于或等于）和ne（不等于）。

有关 sqltest 标记符的更多信息请见“DTML 参考”。如果你的数据库支持其它的比较操作符，例如 like，你可以通过 sqlvar 使用它们。例如，如果 name 是字符"Mc%"，以下 SQL 代码：

```
<dtml-sqltest name type="string" op="like">
```

将解释成：

```
name like 'Mc%'
```

sqltest 标记符可以帮助你构建正确的 SQL 查询。通常，如果你使用 sqltest 而不是手工编写比较代码，你的查询会更为灵活并且很好的处理不同类型的输入和不同的数据库。

5.0.3. 用 sqlgroup 标记符创建复杂的查询

sqlgroup 标记符使你能够创建支持多组参数的 SQL 查询。根据不同的参数，可以准确定制 SQL 查询的范围。 以下是一个没有限制的 SQL 查询例子：

```
select * from employees
```

以下是一个限定工资数额的 SQL 查询例子：

```
select * from employees
where(
  salary > 100000.00
)
```

以下是一个限定工资和名字的 SQL 查询：

```
select * from employees
where(
  salary > 100000.00
  and
  first in ('Jane', 'Cheetah', 'Guido')
)
```

以下是一个限定名字和姓氏的 SQL 查询的例子：

```
select * from employees
where(
  first = 'Old'
  and
  last = 'McDonald'
)
```

这几个查询可以用一个 Z SQL 方法完成，这个方法中指定的参数越多就越精确。以下 SQL 模板可以构建以上这三个查询：

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqltest salary op=gt type=float optional>
<dtml-and>
  <dtml-sqltest first op=eq type=nb multiple optional>
<dtml-and>
  <dtml-sqltest last  op=eq type=nb multiple optional>
</dtml-sqlgroup>
```

如果 sqlgroup 标记符中包含任何文本，就解释 where，并把这些语句构建成查询。如果没有提供参数，这个 sqlgrou 标记符不解释 where 子句。

sqlgroup 标记符由三个被 and 标记符分隔开的块组成。如果合拢的块含有数值，这些标记符插入字符 and。通过这种方式，查询当中就有了几个 and。指定的参数越多，被加入的限定语句就越多。在这个例子中，通过使用 and 语句限制搜索的范围。如果使用 or 标记符则会扩展搜索的范围。

这个例子同时展示了 sqltest 标记符中的 multiple 属性。如果 first 或 last 的值是一个列表，那么 SQL 中就显示成多个值而不是一个值。

你还可以嵌套 sqlgroup 标记符。例如：

```
select * from employees
```

```
<dtml-sqlgroup where>
  <dtml-sqlgroup>
     <dtml-sqltest first op=like type=nb>
  <dtml-and>
     <dtml-sqltest last op=like type=nb>
  </dtml-sqlgroup>
<dtml-or>
  <dtml-sqltest salary op=gt type=float>
</dtml-sqlgroup>
```

如果提供一些参数，这个模板可生成下面的SQL：

```
select * from employees
where
( (first like 'A%'
   and
   last like 'Smith'
  )
  or
  salary > 20000.0
)
```

你可以用sqlgroup标记符构造非常复杂的SQL语句。对于简单的SQL代码，不需要使用sqlgroup标记符。然而，如果你发现你自己正在创建多个不同但是相关联的Z SQL方法，你应该用一个使用sqlgroup标记符的方法看看能否完成相同的任务。

6. 高级技巧

目前为止，你已经看到了如何连接一个关系数据库，进行查询和发送命令，并且创建了用户界面。这些是 Zope 中基本的使用关系数据库的方法。

在以下部分，你将学习如何用 Zope 更为紧密的结合关系数据库查询并且提高性能。我们先从如何通过直接指定参数和用获取的方式传递参数开始。然后，学习如何直接通过 URL 调用 Z SQL 方法访问结果对象。接下来，学习如何通过把结果对象绑定给类，从而使结果对象变得更为强大。最后，我们看看使用缓存提高性能和如何处理数据库事务处理（transactions）。

6.1. 直接指定 Z SQL 方法的参数

如果你用 DTML 调用一个不带参数的 Z SQL 方法，自动从 REQUEST 中收集参数。当从一个搜索表单中查询一个数据库时，它工作得很好，但是有些时候你需要手工或用程序来查询数据库。Zope 可以用 DTML 或 Python 直接指定 Z SQL 方法的参数。例如，对于 employee_by_id，可以使用以下 DTML：

```
<dtml-var standard_html_header>

  <dtml-in expr="employee_by_id(emp_id=42)">

    <h1><dtml-var last>, <dtml-var first></h1>

    <p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-var

    first> makes <dtml-var salary fmt=dollars-and-cents> per year.</p>

  </dtml-in>

<dtml-var standard_html_footer>
```

注意，employee_by_id 方法只返回一个记录，因此 in 标记符中的内容只执行一次。在这个例子中，你象其它任何方法那样调用 Z SQL 方法，并且给它传递一个关键字参数 emp_id。以下例子显示如何使用 Python 轻松地实现相同的任务：

```
## Script (Python) "join_name"

##parameters=id

##

for result in context.employee_by_id(emp_id=id):

    return result.last + ', ' + result.first
```

这段脚本接受一个 id 参数并把它作为 emp_id 传递给 employee_by_id。然后迭代搜索结果并用逗号把名字和姓氏连接起来。

通过直接指定参数，Z SQL 方法可以对关系数据提供更多的控制。用 DTML 和 Python 处理这样的任务非常轻松，就像调用其它 Zope 方法一样。

6.2. 从其它对象获取参数

Z SQL 可以从其它对象获取信息并修改SQL 查询。看看下图 ，它显示了某个组织机构的Web 站点中的几个文件夹：

-

某个组织机构的Web 站点的文件夹结构

假设每个部门的文件夹都有一个字符型的department_id 属性，用来标识相应部门的会计分类帐 id。这个属性将用于通过共享的Z SQL 方法查询相应部门的信息。现举例说明，创建嵌套的带有不同department_id 属性的文件夹。然后在根文件夹中创建一个id 为requisition_something 的Z SQL 方法，它带有三个参数：description、 quantity 和 unit_cost，输入以下查询模板：

```
INSERT INTO requisitions
  (
    department_id, description, quantity, unit_cost
  )
VALUES
  (
    <dtml-sqlvar department_id type=string>,
    <dtml-sqlvar description type=string>,
    <dtml-sqlvar quantity type=int>,
    <dtml-sqlvar unit_cost type=float>
  )
```

现在创建一个Search id 为requisitionSomethingForm ， Report id 为requisitionSomething 的Z Search Interface。选择requisition_something 作为可搜索对象，然后单击 Add。

编辑requisitionSomethingForm，删去department_id 字段的第一个输入框。我们不需要从表单中得到department_id 值，我们需要它来自于一个获取的属性。 现在，能够访问URL，例如：：

http://example.org/Departments/Support/requisitionSomethingForm

来为 Support 部门申请一些吊袋（punching bags）。你还可以通过：

http://example.org/Departments/Sales/requisitionSomethingForm

来为 Sales 部门申请一些带有图标的湿的橡胶链（tacky rubber key-chains）。 使用在“用户和安全”中所描述的 Zope 安全系统，你现在可以限制访问这些表单，从而使得职员只可以获取本部门信息。

这个例子中有意思的是 department_id 不是直接指定的，而是从被访问的 Z SQL 方法所在的文件夹中获得这个值。在前边的 URL 例子中，Z SQL 方法 requisition_something 从 Sales 和 Support 文件夹中获得值。这使你能够灵活调整 SQL 查询。所有的部门可以共享一个查询，但是它是为每个部门定制的。

通过使用获取机制和直接指定参数，就可以根据 Web 应用程序灵活调整 SQL 查询。

6.3. 直接访问结果对象

至此，你已经采用 Web 表单、直接指定参数和获取机制这几种方式给 Z SQL 方法提供参数。你也可以通过 Web 使用特殊的 URL 来给 Z SQL 方法提供参数。这种方式被称为直接访问结果对象。使用这个技术，你可以通过 URL 直接访问结果对象。

要用 URL 直接访问结果对象，必须确保给 SQL 方法指定一个参数只返回一个结果对象。例如，创建一个新的名为 employee_by_id 的 Z SQL 方法，它使用一个参数，emp_id，以下为 SQL 模板：

```
select * from employees where

  <dtml-sqltest emp_id op=eq type=int>
```

这个方法基于雇员的 id 从 employees 表格中选择一个雇员。因为每个雇员有一个唯一的 id，因此只返回一条记录。

Zope 支持一种特定的 URL 格式，从而可以访问只返回一个结果的 Z SQL 方法。方式是在 SQL 方法的后边加上参数。比如，http://localhost:8080/employee_by_id/emp_id/42。这个 URL 返回一个结果对象，就如同使用 DTML 传递一个参数一样。

你用这个 URL 得到的结果看起来比较单调。它没有用 HTML 显示结果。仍然需要对结果对象进行显示处理。要实现这一点，你可以对结果对象调用一个 DTML 方法。方法可以使用以前在“高级 Zope 脚本”中讲过的 URL 获取来实现。例如，以下 URL：

http://localhost:8080/employee_by_id/emp_id/42/viewEmployee

这里，我们看见通过 URL 给 employee_by_id 传递了参数 emp_id。然后对结果对象调用 viewEmployee。让我们创建 DTML 方法 viewEmployee 试一试。创建一个名为 viewEmployee 的 DTML 方法，代码如下：

```
<dtml-var standard_html_header>

  <h1><dtml-var last>, <dtml-var first></h1>
```

```
<p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-var

first> makes <dtml-var salary fmt=dollars-and-cents> per year.</p>

<dtml-var standard_html_footer>
```

现在当你访问 URL ： http://localhost:8080/employee_by_id/emp_id/42/viewEmployee 时，viewEmployee 方法绑定给 employee_by_id 返回的结果对象。 ViewEmployee 方法可以作为通用的模板，其它的 Z SQL 方法也可以调用它。

由于 employee_by_id 方法只接受一个参数，因此甚至可以不用在 URL 中指定 emp_id 参数。如果 Z SQL 方法只有一个参数，那么你可以配置 Z SQL 方法只接受一个额外路径元素参数而不用一对参数。对于这个例子，先选择 employee_by_id 方法，然后单击 Advanced 标签。这里，你可以看见复选框： Allow "Simple" Direct Traversal。选中它，然后单击 Change。现在就可以用简化的 URL 浏览雇员记录，例如 http://localhost:8080/employee_by_id/42/viewEmployee。此时，emp_id 没有在 URL 中出现。

6.4. 其它结果对象方法

结果对象还支持其它一些方法，在一些情况中，使用这些方法会更方便。这些方法可以通过脚本、页面模板和 DTML 进行调用比如，通过脚本：

```
result=context.list_all_employees()

return len(result)
```

通过 DTML：

```
<dtml-var "_.len(list_all_employees())">
```

假定 result 为结果对象，我们可以调用以下方法：

```
len(result)
```

- 显示返回的行数。上边的例子中为 3。

```
result.names()
```

- 返回一个表头列表。上边的例子返回的列表包括 emp_id, first, last 和 salary。

```
result.tuples()
```

- 返回一个元组列表，对于上边的例子，返回：

```
[(43, 'Bob', 'Roberts', 50000),

 (101, 'Cheeta', 'leCat', 100000),
```

(99, 'Jane', 'Junglewoman', 100001)]

result.dictionaries()

- 返回一个词典列表，对于上边的例子，返回：

[{'emp_id': 42, 'first': 'Bob','last': 'Roberts', 'salary': 50000},

{'emp_id': 101, 'first: 'Cheeta', 'last': 'leCat', 'salary': 100000},

{'emp_id': 99, 'first': 'Jane', 'last': 'Junglewoman', 'salary': 100001}]

result.data_dictionary()

- 返回一个词典，它描述了结果表的结构。字典中的键包括 name, type, null 和 width。其中，如果字段中包含空值，那么 null 为 true，width 为字段中字符的宽度。有一些数据库适配器不支持 null 和 width。

result.asRDB()

- 采用和关系数据库类似的方式显示结果。比如通过 DTML 显示结果：

<pre>

<dtml-var "list_all_employees().asRDB()">

</pre>

... 显示结果为 ...

emp_id first last salary

42 Bob Roberts 50000

101 Cheeta leCat 100000

99 Jane Junglewoman 100001

result[0][1]

- 返回第一行第一列中的数据。在这个例子中为 bob。使用时需要仔细注意。

6.5. 给结果对象绑定类

结果对象对于结果行中的每列都有属性。我们还可以编写定制的方法，然后把这些方法添加给结果对象。

结果对象可以通过两种方式绑定方法。就像你在前边一节中所看到的，你可以给 Z SQL 方法的结果对象绑定 DTML 和其它方法方式是通过直接访问结果对象，同时结合普通的基于 URL 的获取绑定机制，这些内容在“高级 Zope 脚本”中讲述过。另外一种方式是通过定义一个 Python 类，这个类中混合有结果对象类。这些类可以在文件系统中与外部方法相同的地址中定义，即 Zope 扩展目录中。Python 类由多个方法和属性组成。通过把结果对象和类关联，你就可以使结果对象拥有丰富的 API 和用户界面。

用来给结果类绑定方法和其它类属性的类被称为 Pluggable Brains 或者 Brains。看看下面这个 Python 类：

```
class Employee:

  def fullName(self):

    """ The full name in the form 'John Doe' """

    return self.first + ' ' + self.last
```

作为 Z SQL 方法查询的结果出现的，并绑定这个 Brains 类的结果对象将拥有 Employee 基础类。这意味着记录对象拥有所有的在 Employee 中定义的方法、行为和数据。

要使用这个类，先在 Extensions 目录中的 Employee.py 文件中输入上面的代码。然后进入 employee_by_id 方法的 Advanced 标签，然后在 Class Name 字段中输入 Employee，在 Class File 字段中输入 Employee，然后单击 Save Changes。然后，编辑 DTML 方法 employeeView ，使之包含以下内容：

```
<dtml-var standard_html_header>

  <h1><dtml-var fullName></h1>

  <p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-var

  first> makes <dtml-var salary fmt=dollars-and-cents> per year.</p>

<dtml-var standard_html_footer>
```

现在当你访问 URL： http://localhost:8080/employee_by_id/42/viewEmployee 时，viewEmployee 方法调用 fullName 方法。fullName 方法在 Employee 模块中的 Employee 类中定义，并且被绑定给由 employee_by_id 返回的结果对象。

Brains 提供了一种非常强大的工具，它使你能够以一种更为面向对象的方式处理关系型数据库中的数据。例如，你不仅可以使用直接访问机制访问 fullName 方法，而且可以在任何地方使用它来处理结果对象。例如：

```
<dtml-in employee_by_id>

  <dtml-var fullName>

</dtml-in>
```

Z SQL 方法返回一个灵活的对象序列，而不仅是数据。

这个例子只显示了用 Brains 所能实现的一部分功能。通过 Python 程序，你还可以创建访问网络资源的 Brains 类，调用其它 Z SQL 方法，以及执行各种类型的事物逻辑。Python 编程已经超出了本书的范围，在此只能提供一个例子。这个例子更为强大。除了 employees 表格，假设你还有 managers 表格。同样假设你有一个 Z SQL 方法 manager_by_id，它根据指定的 emp_id 参数，返回相应的管理员 id：

```
select manager_id from managers where

  <dtml-sqltest emp_id type=int op=eq>
```

你可以在 Brains 类中这样使用这个 Z SQL 方法：

```
class Employee:

    def manager(self):

        """

        Returns this employee's manager or None if the

        employee does not have a manager.

        """

        # Calls the manager_by_id Z SQL Method.

        records=self.manager_by_id(emp_id=self.emp_id)

        if records:

            manager_id=records[0].manager_id

            # Return an employee object by calling the

            # employee_by_id Z SQL Method with the manager's emp_id

            return self.employee_by_id(emp_id=manager_id)[0]
```

这个 Employee 类显示了类方法如何使用其它 Zope 对象来把关系数据编织在一起，使之感觉起来象是对象集合。Manager 方法调用两个 Z SQL 方法，一个用于指出雇员的管理者的 emp_id，另外一个用于返回一个用来表示管理者的结果对象。此时，可以把 employee 对象看作引用了管理者对象。例如，你可以象以下这样使用 DTML 方法 viewEmployee：

```
<dtml-if manager>
```

```
  <dtml-with manager>
    <p> My manager is <dtml-var first> <dtml-var last>.</p>
  </dtml-with>
</dtml-if>
```

你可以看到，Brains 可以变得既复杂又强大。当设计关系数据库应用程序时，你应该让事情保持简单，并且降低复杂性，确保 Brains 类没有添加许多不需要的高级特性。

7. 缓存结果

你可以通过缓存来提高 SQL 查询的性能。缓存存储 Z SQL 方法的结果，因此如果你经常采用相同的参数调用相同的方法，你就不必每次都连接数据库。根据实际应用的不同，缓存可以不同程度的提高性能。

要控制缓存，进入一个 SQL 方法的 Advanced 标签。有三种缓存控制方式，如下图所示。

-

缓存控制方式

Maximum rows to retrieve 字段控制每个查询缓存多少数据。Maximum results to cache 字段控制要缓存多少查询。Maximum time (sec) to cache 控制被缓存的查询保存多长时间。通常，你把这些数值设置得越大，性能提高得越多。同时 Zope 将消耗更多的内存。通过调节性能，应该为应用程序调试出最优的设置。

通常，需要把 Maximum results to cache 设置得足够高，同时把 Maximum time to cache 设置得足够长。对于一个较少点击的站点，缓存结果的时间可以长一些，而对于有许多点击的站点来说，应该把缓存结果的时间设置得短一些。对于内存大的机器

应该增加要缓存的数量。要不使缓存生效，可把缓存时间设置成 0 秒。对于大多数查询，把 Maximum number of rows retrieved 设置成默认值 1000 是比较合适的。对于极为大量的查询，应该增大这个数字。

8. 事务处理（transaction）

事务处理是一组可以被一次性全部撤销的操作。前边已经讲过，对 Zope 所做的所有操作都支持事务处理。事务处理可以确保数据的完整性。当使用一个不支持事务处理的系统时，修改了 10 个对象而当修改第 11 个时失败了，这样数据就是不完整的。事务处理使你能够放弃错误发生时一个请求期间内所做的所有修改。

假设你有一个 Web 页面，它根据顾客购买的商品给顾客开帐单。这个页面首先从存货中扣除商品，然后计算顾客帐目中的总数。如果第 2 个操作由于某种原因失败了，你就需要确信对存货的修改没有生效。

大多数商业和开放源码的关系数据库都支持事务处理。如果关系数据库支持事务处理，则会和 Zope 中的事务处理结合在一起这就确保了 Zope 数据和关系数据库中的数据的完整性。

在这个例子中，事务处理将从顾客开始提交表单开始，直到显示完结果页面为止。此过程中混合了 ZODB 和关系型数据库，Zope 可以确保数据的完整性。

9. 总结

Zope 使你能够用关系数据库构建 Web 应用程序。不象其它的许多 Web 应用服务器，Zope 拥有自己的对象数据库，不是必须使用关系数据库存储信息。 Zope 使你能够象使用其它 Zope 对象那样使用关系数据。你可以用脚本和 Brains 连接关系数据来处理事物逻辑，你可以用 Z SQL 方法和表现工具查询关系数据，例如 DTML，你甚至可以在使用关系数据的同时使用高级 Zope 特性，例如 URL 直接访问、获取、撤销和安全特性。

第十八章 虚拟主机服务

Zope 中使用两种对象来完成虚拟主机功能，即 SiteRoot 和 Virtual Host Monster。 SiteRoots 实际上已经不建议使用，在以前的 Zope 版本中常使用 SiteRoot 。现在，使用 Virtual Host Monster，它可以完成 SiteRoot 的所有功能，并且比较安全。如果你想使用 Zope 的虚拟主机服务，应当使用 Virtual Host Monster 。

1. Virtual Host Monster（VHM）

有些时候，需要 Zope 生成对象的 URL。比如，调用对象的 absolute_url 方法时，需要返回 URL。这个 URL 通常包括主机名称端口，和路径。如果是默认安装，可能不需要虚拟主机服务。而要使用一个 Zope 给多个网站提供服务的时候，就需要使用虚拟主机服务了。在这种情况下，每个网站都有独立的域名，或者与 apache 这样的服务器结合。此时就需要根据设置生成对象的不同 URL。

通过使用 Virtual Host Monster 就可以变更 Zope 对象生成的 URL，这样就可以定制 URL，允许以不同的 URL 方式访问对象。这个功能很有用。例如，你想公布一个 Zope 文件夹中的内容，比如 /FooFolder ，但是要求不包含这个文件夹的名称（比如主机名称 http://www.foofolder.com/）。

Virtual Host Monster 就可以完成这样的功能，方式是通过在 URL 中加入特殊的路径元素，从而可以截取或译出传递给 Zope 的信息。如果 URL 中没有提供特殊的路径元素，那么不进行任何工作。如果提供了，那么 Virtual Host Monster 就解析这些路径元素，并让 Zope 对象生成 URL。

Zope 中受到 Virtual Host Monster 影响的变量包括 REQUEST 中名称以 URL 或 BASE 开始的变量（比如 URL1,BASE2，URLPATH0），以及对象的 absolute_url()方法。

配置 Virtual Host Monster 有点复杂，这是因为它需要以 Zope 的方式重写 URL。为了能够在发送 Zope 的 REQUEST 中的 URL 里加入特殊的路径元素，需要使用一种“重写”（rewrite）工具。Virtual Host Monster 带有一个简单的重写工具，即在它的 Mappings 视图中提供。你也可以使用 Apache 或其它服务器中的重写工具。

2. 把 Virtual Host Monster 放在何处以及如何给它命名

在根文件夹中放置一个 Virtual Host Monster 就可以处理所有的虚拟主机服务。它与 id 无关。

3. 特殊的路径元素 VirtualHostBase 和 VirtualHostRoot

Virtual Host Monster 只有在 URL 中遇到以下特殊的路径元素时才会生效：

- VirtualHostBase
    - 如果 Virtual Host Monster 在传入 URL 中看到了这个名称，就会使得对象生成相应的 URL，包括不同的协议，不同的主机名称和不同的端口号。

VirtualHostRoot

- 如果 Virtual Host Monster 在传入 URL 中看到了这个名称，就会使得对象生成不同根路径的 URL。

VirtualHostBase

- VirtualHostBase 声明一般位于传入 URL 的开始部分。Virtual Host Monster 会截取这个名称后边的两个路径元素组成一个新的协议，主机名称和端口号。

在 VirtualHostBase 声明后边必须是协议和主机名称:端口号，用"/"分开。端口号部分可选。如果没有提供端口号，则不变
例如： 如果在根文件夹中安装了一个 VHM，那么传入 Zope 的 URL：

'http://zopeserver:8080/VirtualHostBase/http/www.buystuff.com'

将生成：

'http://buystuff.com:8080'.

如果在根文件夹中安装了一个 VHM，那么传入 Zope 的 URL：

'http://zopeserver:8080/VirtualHostBase/http/www.buystuff.com:80'

将生成' http://buystuff.com'，其中端口号 80 默认不显示。

如果在根文件夹中安装了一个 VHM，那么传入 Zope 的 URL：

'http://zopeserver:8080/VirtualHostBase/https/www.buystuff.com:443'

将生成' https://buystuff.com/'，其中端口号 443 默认不显示。

需要说明的是如果 Zope 运行在 8080 这样的端口，你想生成不包含这个号码的 URL，则必须在 VirtualHostBase 声明中包含默认的 80，比如 /VirtualHostBase /http/ www.buystuff.com:80。如果不指定 80，则会使用现有的 HTTP 端口号。

4. VirtualHostRoot

VirtualHostRoot 一般位于传入 URL 的结束部分。VHM 会收集 VirtualHostRoot 前边和后边的路径元素，然后生成新的 URL。

举例来说，比如对于/a/b/c/ VirtualHostRoot /d，将去掉 a/b/c，新生成的 URL 中带有 /d。 举例：

如果在根文件夹中安装了一个 VHM，那么传入的 URL：

http://zopeserver:8080/Folder/VirtualHostRoot/

新生成的 URL 将去掉 Folder，并以' http://zopeserver:8080/'开始。访问这个网址的时候即是相对于 Folder 的文件夹。

如果在根文件夹中安装了一个 VHM，那么传入的 URL：

'http://zopeserver:8080/HomeFolder/VirtualHostRoot/Chris

新生成的 URL 将去掉 HomeFolder，并以' http://zopeserver:8080/Chris'开始。访问的时候即相对于' /HomeFolder /Chris'。

5. 一起使用 VirtualHostRoot 和 VirtualHostBase

虚拟主机服务中一种常见的形式是在 Zope 根文件夹中为每个域名创建一个文件夹。比如 http://www.buystuff.com 通过根文件夹中的/buystuff 来提供服务， 而 http://www.mycause.org 通过/mycause 提供服务。为了实现这个功能，需要共同使用 VirtualHostBase 和 VirtualHostRoot 。

要以 http://www.mycause.org/形式访问 /mycause，则应该通过以下 URL：

/VirtualHostBase/http/www.mycause.org:80/mycause/VirtualHostRoot/

要以 http://www.buystuff.com/形式访问/buystuff，则应该通过以下 URL：

/VirtualHostBase/http/www.buystuff.com:80/buystuff/VirtualHostRoot/

6. 测试Virtual Host Monster

假设安装好的Zope的HTTP在8080。进入根文件夹，从添加列表中选择Virtual Host Monster，输入id为VHM，然后单击Add。然后创建一个名为vhm_test的文件夹。在这个文件夹中创建一个名为index_html的DTML方法，修改代码为：

```
<html>
<body>
<table border="1">
  <tr>
    <td>Absolute URL</td>
    <td><dtml-var absolute_url></td>
  </tr>
  <tr>
    <td>URL0</td>
    <td><dtml-var URL0></td>
  </tr>
  <tr>
    <td>URL1</td>
    <td><dtml-var URL1></td>
  </tr>
</table>
</body>
</html>
```

点击View标签，可以看到以下结果：

Absolute URL http://localhost:8080/vhm_test

URL0 http://localhost:8080/vhm_test/index_html

URL1 http://localhost:8080/vhm_test

现在如果访问 http://localhost:8080/vhm_test，可以看到相同的结果。

现在访问 http://localhost:8080/VirtualHostBase/http/zope.com:80/vhm_test，你会看到：

Absolute URL http://zope.com/vhm_test

URL0 http://zope.com/vhm_test/index_html

URL1 http://zope.com/vhm_test

你会发现生成的URL已经变了。其中使用了默认的80端口。

现在访问 http://localhost:8080/VirtualHostBase/http/zope.com:80/vhm_test/VirtualHostRoot/，你会看到：

Absolute URL http://zope.com

URL0 http://zope.com/index_html

URL1 http://zope.com

你会发现vhm_test成为了zope.com的默认根文件夹。

7. 重写传入的URL

至此，你可能还不清楚这些方法有什么好处，因为总不能让用户访问http://yourserver.com//VirtualHostBase/http/zope.com/vhm_test/VirtualHostRoot/这样的URL。答案是：不是让人而是让计算机来完成这个功能。主要有两种方式，一种是通过VHM的Mappings标签来完成，另外一种是通过Apache服务器的"rewrite rules"(或其它服务器中类似的功能)。最好只使用其中的一种方式。下面分别举例说明。

8. Virtual Host Monster Mappings 标签

使用这个功能的前提是：

- 只使用Zope，没有使用Apache这样的前端服务器。
- 你有一个或多个文件夹需要以" http://some.hostname.com/"这样的形式访问，而不是"http://hostname.com/a/folder"。

在 Mappings 中要以表格形式输入信息。 让我们举例说明。假设 Zope 运行在 localhost 的 8080 端口，并在根文件夹中创建了一个 Virtual Host Monster 对象。还需要创建的是在本机系统中的 host 文件中添加一个 alias，在 Unix 中这个文件是 /etc/hosts，在 Windows 中是 c:\WINNT\system32\drivers\etc\hosts，比如：

127.0.0.1 www.example.com

这样就可以在访问主机 www.example.com 时访问本机中的 Zope。

添加完成以后，进入 VHM 的 Mappings 标签，输入一行：

www.example.com:8080/vhm_test

这样就可以通过 http://www.example.com:8080 访问 vhm_test 文件夹。如果访问 http://www.example.com:8080，会出现:

Absolute URL http://www.example.com:8081

URL0 http://www.example.com:8080/index_html

URL1 http://www.example.com:8080

如果在实际的情况中，就可以以 http://www.example.com:8080 形式公布 vhm_test 文件夹了。

你还可以加入多个子域名，比如".buystuff.com" 将匹配 "my.buystuff.com", "zoom.buystuff.com" 等等。

8.1. Apache Rewrite Rules

如果你还同时使用 Apache，就不能使用 Mappings 标签，你应该使用 Apache 的重写规则功能来指向 Zope。这种方式简单的说即 Apache 监听 80 端口，而同时 Zope 的 Web 服务器监听其它端口（比如 8080）。通过在 Apache 的配置文件中进行配置，告诉 Apache 使用重写规则功能，此时需要 Apache 打开 mod_rewrite 模块。方法是在 Apache 的配置文件中加入 --enable-module=rewrite 。

当在 Apache 中配置好 mod_rewrite 模块以后，才能继续完成这个例子。假设 Zope 运行在 localhost 的 8080 端口，并在根文件夹中创建了一个 Virtual Host Monster 对象。还需要创建的是在本机系统中的 host 文件中添加一个 alias，在 Unix 中这个文件是/etc/hosts，在 Windows 中是 c:\WINNT\system32\drivers\etc\hosts，比如：

127.0.0.1 www.example.com

这样就可以在访问主机 www.example.com 时访问本机中的 Zope。

现在假设 Apache 运行在 80 端口，Zope 运行在 8080 端口，你想以 www.example.com 形式访问 vhm_test 文件夹，需要在 Apache 的 httpd.conf 中加入以下代码，然后重新启动 Apache。

NameVirtualHost *

<VirtualHost *>

ServerName www.example.com

RewriteEngine On

RewriteRule ^/(.*)
http://127.0.0.1:8080/VirtualHostBase/http/www.example.com:80/vhm_test/VirtualHostRoot/$1 [L,P]

</VirtualHost>

现在访问 http://www.example.com，你会看到：

Absolute URL http://www.example.com

URL0 http://www.example.com/index_html

URL1 http://www.example.com

这个页面由 Apache 提供，但结果来自于 Zope 。上边的配置使得 Apache 能够把 URL 重新处理，指向到 Zope。

关于 Apache，请参考相关文档。

9. "Inside-Out" Virtual Hosting

虚拟主机的另外一种用途是让 Zope 只控制站点中的一部分，剩下的由其它的服务器来完成。比如，让 Zope 只负责 http://www.mycause.org/dynamic_stuff，让 Apache 负责 http://www.mycause.org/中的其余部分。此时需要给 Zope 生成的 URL 中加入"dynamic_stuff"部分。

如果在插入 VirtualHostRoot 时，后边的路径元素名称用 _vh_ 开始，那么这些元素将忽略掉，只留下剩余的部分。比如 /a/ VirtualHostRoot /_vh_z/，将去掉 a，生成的 URL 用/z 开始。

在我们这个例子中，应该发送请求至 http://www.mycause.org/dynamic_stuff/anything，那么配置文件中则应该写成 /VirtualHostRoot/_vh_dynamic_stuff/anything 虚拟主机服务".

第十九章 任务（Sessions）处理

Sessions 可用来为站点用户保持 HTTP 请求之间的状态，从而实现类似于购物程序中所需要的与某个用户相关的状态信息。这种状态一般需要维持一段时间，一般都要大于一个请求所能维持的时间。Zope 2.5.0 以后的版本都内建了 Sessions。

1. 介绍

Sessions 可以用来跟踪站点访问者的状态。Web 浏览器通过 HTTP 协议与 Zope 交换数据，而 HTTP 不能识别后续的请求是否来自于同一用户。这是因为 HTTP 认为每个请求都是完全独立的。

Sessions 就可以帮助解决这种问题。Session 这个词就就意味着一段时间内源自于相同客户端的多个相关联的 HTTP 请求。Zope 的 sessions 使用 cookies、HTTP 表单元素、或部分 URL 在后台跟踪用户的 sessions。Zope 的 session 系统可以不通过手工方式管理用户的 session。匿名用户和已登录用户都可以使用 session 来跟踪。

session 中使用数据被称为“session 数据”。session 数据只在一定的持续期内有效，这段持续期可以进行配置。session 数据可用来记录用户访问站点时用到的数据，比如用户在购物框中存放的商品信息。

值得注意的是把敏感的数据存放在 session 数据对象中是不安全的，除非浏览器和 Zope 之间的连接进行了加密处理。在不确定的情况下，不要把与用户密切相关的信息放在 session 中，比如电话号码、住址、帐号、信用卡号码或其它的个人信息。

另外，sessions 也适用于那些对系统性能影响比较大的页面。影响的程度依据使用方式和配置的不同而不同，一般来说，使用 session 和不使用 session 时执行速度相差 5% - 10%，则应该使用 session。

2. session 配置

Zope 从 2.5 以后自带有配置好的 session 环境，因此一般不需要更改这些对象，除非你非常想知道如何设置 session。 Zope 使用几种对象来管理 session 数据，简要介绍如下：

- Browser ID Manager
    - 这个对象用来管理如何识别访问者的浏览器，方式包括 cookies、表单变量或 url 路径元素或者是这几种的结合。Zope 中默认提供的是 browser_id_manager。

Transient Object Container

- 这个对象用于存储 session 数据。它可以设置 session 的持续时间。Zope 默认提供的是 /temp_folder/session_data。其中的数据每次重新启动 Zope 都会消失。

Session Data Manager

- 这个对象用于连接 browser id 和 session 数据信息。Zope 中默认提供的是/session_data_manager。

3. 使用 Session 数据

一般通过 REQUEST 的 SESSION 属性可以访问 session 数据。 以下是一个用 Python 脚本处理 session 的例子：

```
## Script (Python) "sessionTest"

secs_per_day=24*60*60

session=context.REQUEST.SESSION
```

```
if session.has_key('last view'):
    # The script has been viewed before, since the 'last view'
    # has been previously set in the session.
    then=session['last view']
    now=context.ZopeTime()
    session['last view']=now # reset last view to now
    return 'Seconds since last view %.2f' % ((now - then) * secs_per_day)
# The script hasn't been viewed before, since there's no 'last
# view' in the session data.
session['last view']=context.ZopeTime()
return 'This is your first view'
```

注意，这只是一个非常简单的例子，不适合实际中应用。

使用以上代码在根文件夹中创建名为 sessionTest 的脚本，然后通过 Test 标签测试这个脚本。反复调用这个页面多次，这段脚本会跟踪上次的访问时间并计算时间间隔。如果关闭了浏览器，重新查看这个页面，上次的记录就没有了。如果你先访问其它页面，然后在 20 分钟以内返回到这个页面，你的信息仍然存在。

这个例子显示了 session 数据的基本特性，session 数据对象有点类似于 Python 中的字典。session 数据对象可以存储任何对象，经常遇到的是存储普通的对象，比如列表、字典、字符串和数字等等。

需要注意的是，当用 session 处理字典或列表这样的数据时，需要通过分配数值来存储 session 数据，比如以下代码：

```
## Script (Python) "sessionExample"
session=context.REQUEST.SESSION
# l is a list
l=session.get("myList", [])
l.append("spam")
# If you quit here, your changes to the list won't
```

```
# be saved. You need to save the session data by

# reassigning it to the session.

session["myList"]=l
```

如果你忘记了执行最后一步，即 session myList =l，对列表的更改不会保存下来，这样新的数据不会生效。

你当然可以在页面模板和 DTML 文档中调用 session。比如，下面的模板显示了用户最喜欢的颜色（存储在 session 中）。

```
<p tal:content="request/SESSION/favorite_color">Blue</p>
```

用 DTML，可以是这样：

```
<dtml-with SESSION mapping>

  <p><dtml-var favorite_color></p>

</dtml-with>
```

在 Zope 2.5 以后的版本中，提供了一个“购物车”例子程序，其中使用了 session。

4. 细节

Zope session 主要通过四种组件对象实现：

- Browser Id Manager（浏览器 Id 管理器） --这个对象用于检测远程客户端的浏览器 id，用于识别浏览器。浏览器 id 根据 form/querystring 变量、cookie 变量或部分 URL 进行编码。一般只需要设置一个 Browser Id Manager 就够用了，应用程序一般不需要直接访问它。使用的时候，是通过 SESSION 对象来进行的。

- Session Data Manager(Session 数据管理器) -- 这个对象用于存储 Session 数据。当需要 session 数据的时候，先根据 Browser Id Manager 取得当前浏览器的 id 并创建一个 session 数据或返回相应的 session 数据。一般不需要使用 session 数据管理器的方法来获得 session 数据对象，而是通过内置的 REQUEST.SESSION 对象来进行，这个对象代表了对应当前浏览器 id 的 session 数据对象。多个 session 数据管理器可以共享一个 Browser Id Manager。

- Transient Data Container -- 这个对象是真正的处理 session 信息的组件。它用来存储每个 session 的"transient data object"。

- Transient Data Object -- 这些对象存储在 session data container 中，通过 transient data manager 进行管理。它用来存储某个用户 session 的相关信息。

5. 常用术语

- Browser Id
    - 这是一个字符串或整数，表示了一次访问，它由 browser id manager 管理。比如：12083789728。

Browser Id Name

- 由 browser id manager 列出的 browser id 名称。比如：_ ZopeId _。

Browser Id Namespaces

- 搜索 Browser Id Name 的地方包括：表单元素，以及或查询字符串，cookie，或 URL。这几种情况组成了 Browser Id Namespaces。

Session Data Object

- 当前 browser id 所对应的 transient data 对象。

Session Id

- session 数据对象的 id。它表示了一个用户的访问。

6. 默认配置

Zope 已经默认配置好了 Session。 默认的 browser id manager 是位于根目录中的 browser_id_manager。 默认的 session data manager 是位于根目录中的 session_data_manager。 默认的 transient data container 是位于 temp_folder 中的 session_data。它存储在 RAM 内存中。不支持 transactions。

transient data container 存储 transient data objects。一般不需要更改默认的设置。如果要更改，只要通过添加列表添加新的对象就可以了。

7. 使用 Session

7.1. 概述

一般可以通过 REQUEST.SESSION 对象来读取 session 数据对象。当处理 REQUEST.SESSION 对象时，是在处理和当前某个用户相关的 session 数据。

Session 数据对象支持多种方法，从而可以读取和设置数据。同时也可以通过获取机制调用 session data manager 的方法。详细内容请参考 Zope API 部分。

7.2. 获得一个 Session Data Object

要获得与当前浏览器 id 对应的 session 数据对象，可以使用 REQUEST.SESSION。如果在 session data container 中不存在 session data object，则会自动创建。

```
<dtml-let data="REQUEST.SESSION">
  The 'data' name now refers to a new or existing session data object.
</dtml-let>
```

还可以使用 session data manager 的 getSessionData()方法完成相同的事情：

```
<dtml-let data="session_data_manager.getSessionData()">
  The 'data' name now refers to a new or existing session data object.
</dtml-let>
```

当使用 REQUEST.SESSION 或 getSessionData()的时候，如果当前请求中没有相应的 browser id，则会自动创建一个新的 browser id，同时创建一个新的 session 数据对象。如果要不使用这个功能，可以在 getSessionData()中使用“create=0”：

```
<dtml-let
   data="session_data_manager.getSessionData(create=0)"> The
   'data' name now refers to an existing session data object
   or None if there was no existing browser id or session data
   object.  </dtml-let>
```

7.3. 修改 Session Data Object

使用 REQUEST.SESSION 或 session_data_manager.getSessionData()的时候，可以通过 key/value 方式修改 session 中的值，方法包括：set,get,has_key，比如：

```
<dtml-let data="REQUEST.SESSION">
  <dtml-call "data.set('foo', 'bar')">
  <dtml-comment>Set 'foo' key to 'bar' value.</dtml-comment>
  <dtml-var "data.get('foo')">
  <dtml-comment>Will print 'bar'</dtml-comment>
  <dtml-if "data.has_key('foo')">
```

```
    This will be printed.
  <dtml-else>
    This will not be printed.
  </dtml-if>
</dtml-let>
```

session 中可以是任何对象类型。session data container 中提供 session 的时间存在范围，在时间范围内可以使用 session 数据对象。

7.4. 使 Session Data Object 失效

开发者可以通过调用 invalidate()方法使 session data object 失效。比如：

```
<dtml-let data="REQUEST.SESSION">
  <dtml-call "data.invalidate()">
</dtml-let>
```

这个 session data object 将会失效，如果再次调用它就会生成新的 session data object。 如果为 session data object 定义了“onDelete”方法，那么在数据对象失效以前将调用 onDelete 方法。

7.5. 使 Browser Id Cookie 失效

使 session data object 失效并不会使用户浏览器中的 cookie 失效。此时可以调用 browser id manager 的 flushBrowserIdCookie()方法，比如：

```
<dtml-call "REQUEST.SESSION.getBrowserIdManager().flushBrowserIdCookie()">
```

8. 例子：通过 DTML 使用 Session 数据

以下的例子显示了如何获得 session data object，以及如何添加新值：

```
<dtml-let a="REQUEST.SESSION">
  Before change: <dtml-var a><br>
  <dtml-call "a.set('zopetime', ZopeTime())">
  <dtml-comment>
```

```
  'zopetime' will be set to a datetime object for the current
  session
  </dtml-comment>
  After change: <dtml-var a><br>
</dtml-let>
```

当第一次调用这段代码，在没有设置新值以前返回为空，设置以后将返回日期时间。以后继续调用时，都会显示时间。

8.1. 在 dtml-with 中使用 mapping 关键字

DTML 可以很方便的对 session data object 建立一个名称映射，即使用 mapping 关键字，比如：

```
<dtml-let a="REQUEST.SESSION">
    <dtml-call "a.set('zopetime', ZopeTime())">
    <dtml-comment>
      'zopetime' will be set to a datetime object for the current
      session... the "set" it calls is the set method of the
      session data object.
    </dtml-comment>
</dtml-let>
<dtml-with "REQUEST.SESSION" mapping>
    <dtml-var zopetime>
    <dtml-comment>
      'dtml-var zopetime' will print the DateTime object just set
      because we've used the mapping keyword to map name lookups
      into the current session data object.
```

```
    </dtml-comment>
</dtml-with>
```

9. 通过 Python 调用 Session 数据

以下这个例子显示了如何通过 Python 外部方法 session：

```
import time
def setCurrentTime(self):
    a = self.REQUEST.SESSION
    a.set('thetime', time.time())
def getLastTime(self):
    a = self.REQUEST.SESSION
    return a.get('thetime')
```

调用 setCurrentTime 会在 session 中把'thetime'键值设成整数形式的当前时间。调用 getLastTime 方法将返回这个整数形式的当前时间。

10. 调用 Browser Id 数据

取得当前请求的 browser id 值可以是这样：

```
<dtml-var "REQUEST.SESSION.getBrowserIdManager().getBrowserId()">
```

另外一种方式是：

```
<dtml-var "REQUEST.SESSION.getContainerKey()">
```

上边的例子将显示 browser id 值，如果不存在当前请求的 browser id，就创建新值，并返回新值。

如果不想创建新值，则可以使用 create=0 参数：

```
<dtml-var "browser_id_manager.getBrowserId(create=0)">
```

如果不含有当前请求的 browser id 值，则会返回空（None）。

其中，browser id 可以是字符串或整数，没有特别的含义。

10.1. 判断 Browser Id 的名称空间

在有些情况下，需要知道当前的browser id所使用的名称空间，即："cookies", "form", 或 "url"。实现这个功能，可以使用isBrowserIdFromCookie(), isBrowserIdFromForm(), 和 isBrowserIdFromUrl()，比如：

```
<dtml-if "REQUEST.SESSION.getBrowserIdManager().isBrowserIdFromCookie()">

  browser id 来自 cookie.

</dtml-if>

<dtml-if "REQUEST.SESSION.getBrowserIdManager().isBrowserIdFromForm()">

  browser id 来自表单.

</dtml-if>

<dtml-if "REQUEST.SESSION.getBrowserIdManager().isBrowserIdFromUrl()">

  browser id 来自 URL.

</dtml-if>
```

如果当前请求中不存在browser id，则提示错误。 一般来说，这个功能不是很常用。

10.2. 获得 Browser Id 的名称和数值，并嵌入表单中

通过browser id manager可以获得browser id 的名称。有些时候需要把相应的值嵌入到表单中，比如：

```
<html>

<body>

<form action="thenextmethod">

<input type=submit name="submit" value=" GO ">

<input type=hidden name="<dtml-var "REQUEST.SESSION.getBrowserIdManager().getBrowserIdName()">"

 value="<dtml-var "REQUEST.SESSION.getBrowserIdManager().getBrowserId()">">

</form>

</body>
```

```
</html>
```

另外一种方法是使用browser id manager 的getHiddenFormField方法：

```
<html>
<body>
<form action="thenextmethod">
<input type="submit" name="submit" value=" GO ">
<dtml-var "REQUEST.SESSION.getBrowserIdManager().getHiddenFormField()">
</form>
</body>
</html>
```

以上DTML的执行结果，可以类似于这样：

```
<html>
<body>
<form action="thenextmethod">
<input type="submit" name="submit" value=" GO ">
<input type="hidden" name="_ZopeId" value="9as09a7fs70y1j2hd7at8g">
</form>
</body>
</html>
```

10.3. 判断 Browser Id 是否为新的

如果当前请求第一次发送，那么browser id可认为是新的，而相当于后续的请求则为旧的。通过调用browser id manager的isBrowserIdNew()方法可以进行判断，比如：

```
<dtml-if "REQUEST.SESSION.getBrowserIdManager().isBrowserIdNew()">
```

```
  Browser id is new.
<dtml-else>
  Browser id is not new.
</dtml-if>
```

如果当前请求没有browser id，则会提示错误。

11. 判断当前请求所对应的Browser Id是否存在相应的Session Data Object

如果你想判断session data manager中是否有相应的session data object，可以使用session data manager的hasSessionData()方法。如果存在则返回真，比如：

```
<dtml-if "session_data_manager.hasSessionData()">
   The sessiondatamanager object has session data for the browser id
   associated with this request.
<dtml-else>
   The sessiondatamanager object does not have session data for
   the browser id associated with this request.
</dtml-if>
```

实际上，这个功能在很多情况下使用REQUEST.SESSION就够了。

12. 把一个Browser Id嵌入到HTML链接中

你可以把browser id的名称和数值嵌入到HTML链接中。点击链接时，会把这些信息通过REQUEST.form传递给Zope。此时需要使用browser id manager的encodeUrl()方法，比如：

```
<html>
<body>
<a href="<dtml-var "REQUEST.SESSION.getBrowserIdManager().encodeUrl('/amethod')">">Here</a>
 is a link.
```

</body>

</html>

上边的这个例子会对"/amethod"进行编码，生成带有名称和数值的字符串。比如，类似于这样的结果：

<html>

<body>

Here

 is a link.

</body>

</html>

通过使用 encodeUrl()，可以传递 URL 参数，这些参数附加在所指定的 URL 后边。

你还可以指定"inline"格式，通过 URL 来实现，比如：

<html>

<body>

<a href="<dtml-var "REQUEST.SESSION.getBrowserIdManager().encodeUrl('/amethod', style='inline')">">Here

 is a link.

</body>

</html>

上边的代码执行结果类似于这样：

<html>

<body>

Here

 is a link.

```
</body>

</html>
```

13. 使用 onAdd 和 onDelete 事件

当创建 session 数据时，可以先运行 onAdd 事件，当 session 数据消失时，调用 onDelete 事件。这些事件相互独立。通过这些事件可以更为灵活的定制 session 数据的使用方式。

在/temp_folder/session_data 的管理界面中，可以添加 onAdd 和 onDelete 方法。其中的“Script to call when objects are added”用于指定 onAdd 事件，“Script to call when objects are deleted”用于指定 onDelete 事件。

13.1. 编写 onAdd 和 onDelete 方法

当创建 session 时调用 onAdd 事件，当消失时调用 onDelete 事件。 可以通过脚本或外部方法来编写这些事件方法。 应该定义两个参数，即 sdo 和 toc。sdo 表示正在创建或终止的 session data object，toc 表示存储 session 对象的 transient object container。 例如，要创建一个 onAdd 方法，它可以在 session 中添加 DateTime 对象，那么可以添加脚本，名称为"onAdd"，参数为"sdo" 和 "toc"，代码为：

```
sdo['date'] = context.ZopeTime()
```

然后就可以指定这个方法为 onAdd 事件。 要创建 onDelete 事件方法，用它在 session 消失时写入日志信息，可以这样来编写一个外部方法对象：

```
from zLOG import LOG, WARNING

def onDelete(sdo, toc):

    logged_out = sdo.get('logged_out', None)

    if logged_out is None:

        LOG('session end', WARNING,

            'session ended without user logging out!')
```

指定这个方法为 onDelete 事件。那么如果在 session 中没有找到 logged_out 键，则会在日志中记录下来。 应该注意的是，onDelete 事件不能保证在调用的时候必定和创建这个 session 的用户相关联。如果其中调用了"getSecurityManager().getUser()"，不一定返回和 session 相应的用户。 而 onAdd 则可以确定创建 session 的用户。 对于 onAdd 和 onDelete 事件，通常可以把执行的许可分配给 proxy 用户角色。

14. 配置和操作

14.1. 设置初始 Transient Object Container 参数

由于 transient object container 位于/temp_folder/session_data，它存储在 RAM 内存中，当重新启动 Zope 时会随之消失和创建。也就是说，所做的设置会自动恢复到初始值。对于这个问题，可以通过指定 Zope 的环境变量来解决，以下几个变量会影响 session 的设置：

ZSESSION_ADD_NOTIFY

- 用来指定 temp_folder 的 session_data 对象创建 session 时调用的对象。即 onAdd 事件。

ZSESSION_DEL_NOTIFY

- 用来指定 temp_folder 的 session_data 对象删除 session 时调用的对象。即 onDelete 事件。

ZSESSION_TIMEOUT_MINS

- 用来指定/temp_folder/session_data 时间长度，单位是分钟。

ZSESSION_OBJECT_LIMIT

- 用来指定/temp_folder 中的存放的最大 session 对象数量。

15. 使用多个 Browser Id Managers

Zope 默认提供一个 Browser Id Manager。如果需要，可以创建多个。一种使用多个 Browser Id Manager 的情况是站点分成两部分，一部分对安全要求高，一部分对安全要求不高，每一部分使用不同的 Browser Id Manager，并使用不同的安全设置。 在添加时需要注意的是，如果在下级文件夹中添加 Browser Id Manager，则需要删除根文件夹中默认的 Browser Id Manager。 如果从添加列表中选择“Browser Id Manager”，显示的表单中的选项包括：

- Id
    - 不用指定 id，默认必须为 "browser_id_manager"。

Title

- Browser Id Manager 的标题

Browser Id Name

- 这是用来搜索 Browser Id 值的名称。

Look for Browser Id Name In

- Browser Id 名称的来源，可选项包括 "cookies", "Forms and Query Strings" 和 "URLs"。

Automatically Encode Zope-Generated URLs With A Browser Id

- 如果选择了这个选项，那么 Zope 生成的 URL 中将包含 Browser Id 的名称和数值。

Cookie Path

- 这是发送给 Browser Id cookie 的路径元素。

Cookie Domain

- 这是发送给 Browser Id cookie 的域元素。

Cookie Lifetime In Days

- 设定 cookie 的有效时间。如果为 0，则只在浏览器运行期间有效。

Only Send Cookie Over HTTPS

- 如果选择了这个选项，只有使用 https 协议才发送 cookie。

设置完成以后，点击 Add 按钮即可。

15.1. 使用 Session Data Manager

Zope 中默认使用/session_data_manager，也可以添加新的 Session Data Manager。 你可以把新的 session data manager 放在任何地方，但必须能够找到可用的 browser_id_manager。从添加列表中选择“Session Data Manager”。其中的选项包括：

- Id
  - 输入一个 id。

Title

- 输入一个标题

Transient Object Container Path

- 输入 Transient Object Container 的路径，用来存储 session 数据。比如：/temp_folder/session_data

设置好以后，点击只 Add 按钮即可。

15.2. 使用 Transient Object Container

Zope 中默认使用/temp_folder/session_data。要添加新的 Transient Object Container，可以从添加列表中选择“Transient Object Container”，其中的选项包括：

- Id
    - 输入 id

Title (optional)

- 输入标题

Data object timeout in minutes

- 输入有效时间，单位是分钟。如果为 0，表示没有限制。

Maximum number of subobjects

- 输入能够存放的最大数量。

Script to call upon object add (optional)

- 指定 session 创建时触发执行的对象。

Script to call upon object delete (optional)

- 指定 session 消失时触发执行的对象。

多个 session data manager 可以共享一个 transient objects container。

16. 配置 Session 许可

以下是与 session 相关的许可。

16.1. 与 browser id manager 相关的许可：

Add Browser Id Manager

- 允许指定角色能够添加 browser id manager。管理员角色默认支持。

Change Browser Id Manager

- 允许指定角色能够更改 browser id manager。管理员角色默认支持。

Access contents information

- 允许指定角色能够获得browser id的数据。管理员和匿名角色默认支持。

16.2. 与session data manager相关的许可：

Add Session Data Manager

- 允许指定角色能够添加session data manager。管理员角色默认支持。

Change Session Data Manager

- 允许指定角色更改session data manager。管理员角色默认支持。

Access session data

- 允许指定角色访问session data object。管理员和匿名角色默认支持。

Access arbitrary user session data

- 允许指定角色访问特定的session数据。管理员角色默认支持。

Access contents information

- 允许指定角色访问session数据。管理员和匿名角色默认支持。

16.3. 与transient object container相关的许可

Add Transient Object Container

- 允许指定角色添加transient objects containers。管理员角色默认支持。

Change Transient Object Container

- 允许指定角色更改transient object container。

Access Transient Objects

- 允许指定角色访问transient object。

第二十章 性能扩展与 ZEO

当一个 Web 应用程序收到超过它所能承受的请求数量时，会变得缓慢并且响应迟钝。最坏的情况是，请求太多以至于使得服务器完全超载，停止处理请求，甚至可能瘫痪。这对于任何类型的服务器应用程序来说都会是一个问题，而不仅仅是 Zope。这个问题的最直接的解决方法是使用多台计算机，因此一旦某台计算机失效了，另外一台得以继续为 Web 站点提供服务。并且按照这种方式可以根据需要进行扩展。

使用多台计算机有明显的好处，但是也有一些缺点。例如，如果你有五台计算机在运行 Zope，那么你就必须确保所有这五台安装有 Zope 的计算机上拥有相同的信息。如果你只有一些静态对象，那么这不是一个非常困难的任务，但是对于大型的拥有上千个迅速变化的对象的大型站点，要手工保持五个分离的 Zope 同步运行将是一场恶梦。

为了解决这个问题，Zope 公司创建了 Zope 企业对象（ZEO）（ http://www.zope.org/Products/ZEO）。ZEO 使用 client/server 架构，允许多个 Zope 使用同一数据库信息。本章主要讲述 ZEO。有关更为深入的信息，请见随同 ZEO 软件包发布的文档，以及查看 ZEO 讨论区（ http://www.zope.org/Wikis/ZODB/FrontPage）。

1. 什么是 ZEO

ZEO 是一种使你能够在多个 Zope 进程中共享 ZODB 的系统。通过使用 ZEO，可以在一台或多台计算机中运行多个 Zope 实例，从而可以均匀的分散请求。如果请求数量增加，可以添加更多的计算机。另外，如果某台计算机失败或瘫痪了，在你修复中断的计算机的同时，其它计算机仍然可以为请求提供服务。ZEO 可以确保每个 Zope 调用相同的数据库信息。

ZEO 使用客户机/服务器架构。每个 Zope 进程是 ZEO 客户机。所有的客户机连接到一个中央 ZEO 存储服务器，如下图所示。

-

ZEO 结构

这些术语可能有些混乱。通常的情况下，把 Zope 看作是服务器，不是一个客户机。当使用 ZEO 时， Zope 的作用既担当服务器（相对于 Web 请求）又担当客户机（相对于 ZEO 服务器）。

ZEO 客户机和服务器使用标准的 Internet 协议进行通信，因此它们可以位于同一房间里或位于不同的国家。事实上，通过 ZEO 甚至可以把 Zope 站点分布在全球范围内。在本章，我们探讨一些有趣的方式，通过这些方式可以分布 ZEO 客户机。

2. 何时使用 ZEO

经过 ZEO 配置的 Zope 可以为大量的请求提供服务。如果站点没有达到数百万次的点击，那么你可能不需要 ZEO。需要使用 ZEO 的情况，例如：

- 你安装了一个 Zope，但不能快速的处理请求。Zope 是一种高性能的系统，并且一个 Zope 可以处理数百万次的点击，但毕竟还是有上限的。ZEO 可以让你通过添加计算机来扩展站点的处理能力。
- 你的站点要求很高，需要 7 天/24 小时连续运转。在这种情况下，ZEO 使你能够扩展性能。
- 你需要在全球范围内分布站点，以便增强远程站点的响应时间。通过 ZEO 可以让远程的站点使用相同的 ZODB。
- 你需要调试一个应用程序，这个程序由另外一个 Zope 进程提供 Zope 服务。这个技术适用于 Python 开发者，本书不讲述。

安装、配置和维护一个支持 ZEO 的系统需要一些系统管理知识。大多数 Zope 用户不需要 ZEO，或者可能没有维护一个象 ZEO 这样的分布式服务系统所必需的知识。ZEO 很有趣，并且非常有用，但是在开始安装 ZEO 以前，应该仔细考虑一下是否真的需要使用 ZEO。

3. 安装和运行 ZEO

安装 ZEO 需要从 Zope.org 的产品部分下载。使用 ZEO 之前，需要注意的是：

- 必须运行相同版本的 Zope 和 ZEO。确信所有计算机使用最新的版本。这是必需的，否则 Zope 可能运转不正常。
- 所有 ZEO 客户机必须安装有相同的第三方产品，并且版本必须相同。这是必需的，否则，你的第三方对象可能运转不正常或根本就不工作。
- 如果你的 Zope 系统需要访问外部的资源，例如邮件服务器或关系数据库，确信所有 ZEO 客户机有权访问这些资源。
- 客户机和服务器之间不好的连接会降低 ZEO 客户机的性能。你的 ZEO 客户机与它们的服务器之间应该有良好的连接。

安装 ZEO 需要做一些准备。要安装 ZEO，从 Zope.org Web 站点下载 ZEO 安装包，并把它放置在 Zope 安装目录中。现在解压缩安装包。在 UNIX 中，这个过程可以用以下命令完成：

```
$ tar -zxf ZEO-X.X.tar.gz
```

如果没有 GNU tar 工具，使用以下命令：

```
$ gzip -cd ZEO-X.X.tar.gz | tar -xvf -
```

Windows 中，可以使用 Winzip。

现在有了 ZEO-X.X 目录。然后使用以下命令：

```
$ cd ZEO-X.X

$ python2.1 setup.py install --home=/path/to/your/Zope/top/level/dir
```

其中 home 为安装 Zope 的根目录。注意要使用和 Zope 相同的 python。上边的命令执行完以后，会在 lib/python 中生成一个 ZEO 目录。

接下来，需要在 Zope 根目录中创建一个名为 custom_zodb.py 的特殊文件。在这个文件中，输入以下 Python 代码：

```
import ZEO.ClientStorage

Storage=ZEO.ClientStorage.ClientStorage(('localhost',7700))
```

这个这个程序把 Zope 配置成一个 ZEO 客户机。如果你象这段代码这样给 ClientStorage 传递一个元组，元组必须有两个元素：一个是含有服务器名称或地址的字符串，以及服务器监听的端口。在这个例子中，将给你展示如何在相同的机器上运行客户机和服务器，因此机器名称被设置为 localhost，端口为 7700。

现在已经安装好了 ZEO。试试启动服务器。在终端窗口中或在 DOS 窗口中进入顶级 Zope 目录并键入：

```
python2.1 lib/python/ZEO/start.py -p 7700
```

这样就会启动 ZEO 服务器，它监听计算机上的 7700 端口。现在，在另外一个窗口中，象通常一样使用 z2.py 脚本启动 Zope：

```
$ python z2.py -D

------

2000-10-04T20:43:11 INFO(0) client Trying to connect to server

------

2000-10-04T20:43:11 INFO(0) ClientStorage Connected to storage

------

2000-10-04T20:43:12 PROBLEM(100) ZServer Computing default pinky

------

2000-10-04T20:43:12 INFO(0) ZServer Medusa (V1.19) started at Wed Oct  4 15:43:12 2000

        Hostname: pinky.zopezoo.org

        Port:8080
```

注意以上例子是如何运行的，Zope 告诉你客户机试图连接到服务器，然后 ClientStorage 连接到了 storage。这意味着你的 ZEO 客户机已经成功连接到了 ZEO 服务器。现在，你可以访问 http://localhost:8080/manage（或者你的 ZEO 客户机正在监听的任何 URL），象通常一样登录到 Zope。

你可以看到，一切都没有变化。如果你进入到控制面板并单击 Database Management，你会看见 Zope 连接到一个 ZEO 存储（ZEO storage），它的状态是已连接。

在一台计算机上运行 ZEO 能够很方便的熟悉 ZEO 以及工作方式。但在一台计算机上运行 ZEO 不会提高站点速度。事实上，它还可能会略微降低速度。要真正得到 ZEO 在速度上提供好处，你需要运行多个 ZEO 客户机。

4. 如何运行多个 ZEO 客户机

添加新的 ZEO 客户机可以增强站点的。例如，假设你有四台计算机。名为 zooserver 的计算机是 ZEO 服务器，其它三台名为 zeoclient1、zeoclient2 和 zeoclient3 的计算机是 ZEO 客户机。另外，还假设这几台计算机都属于“.zopezoo.org”域。

第一步是在 zooserver 上运行 ZEO 服务器。要告诉 ZEO 服务器监听 zooserver 的 7700 端口，可以象以下这样用 start.py 脚本运行服务器：

```
$ python2.1 lib/python/ZEO/start.py -p 7700
```

这将启动 ZEO 服务器。接下来，启动客户机，方式是进入每个客户机，编辑以下 custom_zodb.py：

```
import ZEO.ClientStorage

Storage=ZEO.ClientStorage.ClientStorage(('zooserver.zopezoo.org',7700))
```

现在，你可以象前边所描述的那样启动每个客户机的 z2.py 脚本。注意主机和每个客户机的端口是相同的。这样它们就可以连接到相同的服务器。按照这个过程为配置好所有的客户机，你将拥有三个提供相同服务的 Zope。你可以通过访问这三台客户机器上的端口 8080 来验证这些。

你可能需要在多台计算机上运行 ZEO，这样你就可以提高访问速度。比起仅用一台计算机，运行多台计算意味着每秒钟可以给更多的点击提供服务。要分配 Web 站点的访问负载，还需要做一些工作。以下一节描述为什么和如何在多台计算机之间分配访问负载。

5. 如何分配负载

在前边的例子里，有了一台名为 zooServer 的 ZEO 服务器以及三台名为 zeoclient1、 zeoclient2 和 zeoclient3 的 ZEO 客户机。这三台 ZEO 客户机连接到 ZEO 服务器，并且每台客户机都能正常工作。

现在你有三台计算机给用户提供内容服务。下一个问题是如何真正的在三台 ZEO 客户机之间均匀分布 Web 请求。你的用户只知道 www.zopezoo.org，而不知道 zeoclient1、zeoclient2 或者 zeoclient3。如果让某些用户使用 zeoclient1，而其他用户使用 zeoclient3，这种方式将很麻烦，并且不能充分使用计算机资源。你需要自动或者至少非常轻松的完成在多台 ZEO 客户机之间均匀分配请求。

针对这个问题存在许多解决方法，一些容易，一些复杂，一些昂贵。以下一节讲述几种常用的分配 Web 请求的方法，其中一些基于免费的或商业的软件，其它一些基于特殊的硬件。

5.1. 让用户选择一个镜像站点

在许多 Web 服务器之间分配请求，最容易的方法是使用镜像站点，其中的每一个都是一台 ZEO 客户机。使用这个方法不需要其它的软件或硬件，它只需要维护镜像服务器。通过给用户提供镜像服务器的菜单，他们可以选择使用哪个服务器。

注意，这种分配请求的方法是被动的（你不能主动控制用户使用哪一个 ZEO 客户机），而它又是主动的（你的用户需要主动选择使用哪个 ZEO 客户机）。如果你的用户不使用一个镜像，那么请求被发送到你的那个为 www.zopezoo.org 提供服务的 ZEO 客户机。

如果你不想对镜像进行任何管理控制，那么这是一个相当简便的解决方法。如果镜像离线了，用户总是可以返回到主站点，你总是可以对主站点进行管理控制，从而可以让用户选择其它的镜像。

在全球的层次上，这个方法提高了性能。用户可以选择使用一台在地理上离他们更近的服务器，这样就能够加快访问速度。例如，如果你的主服务器在美国西海岸的俄勒冈州的波特兰，而你的用户在英国伦敦，他们可以选择伦敦镜像，这样他们的请求就不必来回穿越半个地球。

要使用这个方法，给根文件夹添加一个属性，类型为 lines，名称为 mirror。输入各个 ZEO 客户机的 URL，如下图所示。

-

镜像属性图

现在，给站点添加一些简单的 DTML 来显示镜像列表：

```
<h2>Please choose from the following mirrors:

<ul>
```

```
  <dtml-in mirrors>
  <li><a href="&dtml-sequence-item;"><dtml-var
  sequence-item></a></li>
  </dtml-in>
</ul>
```

或者使用脚本：

```
## Script (Python) "generate_mirror"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind subpath=traverse_subpath
##parameters=a, b
##title=
##
print "<h2>Please choose from the following mirrors: <ul>"
for mirror in container.mirrors:
    print "<li><a href="%s">%s</a>" % (mirror, mirror)
return printed
```

这个 DTML 或脚本显示镜像站点列表。使用这种方式，计算机所起的名称最好能够帮助用户选择镜像。例如，如果你按照地理位置进行分布负载，那么计算机名称可以选择国家的名称。

另外，如果你不想让用户主动选择一个镜像，你还可以让 www.zopezoo.org 站点的 index_html 方法生成 HTTP 重定向。例如，在 www.zopezoo.org 站点的 index_html 方法中使用以下代码：

<dtml-call expr="RESPONSE.redirect(_.whrandom.choice(mirror_servers))">

这段代码将使任何访问者从 www.zopezoo.org 随机跳转到一个镜像服务器。

5.2. 使用 Round-Robin DNS 分配负载

域名系统(DNS)是一种把计算机名称（比如 www.zope.org）转换成数字地址的 Internet 机制。这种机制可以把一个名称映射到多个地址。

进行负载均衡的最简单方法是使用 round-robin DNS，如下图所示。

用 round-robin DNS 进行负载均衡

当解析 www.zopezoo.org 时，得到的结果是 zeoclient1、zeoclient2 或 zeoclient3，每次按照轮流的顺序完成。例如，一个用户解析 www.zopezoo.org 可能得到 zeoclient1 的地址，而另外一个用户解析 www.zopezoo.org 可能得到 zeoclient2 地址。通过这种方式，用户被分散给各个 ZEO 客户机。

这不是一个完美的负载均衡方案，这是因为 DNS 解析的信息会被网络上其它名称服务器缓存。当 www.zopezoo.org 解析给某个 ZEO 客户机以后，所有后来的请求都发送到相同的 ZEO 客户机。最终的结果通常是可以接受的，这是因为总体上来说，所有的请求被分散给各个 ZEO 客户机。

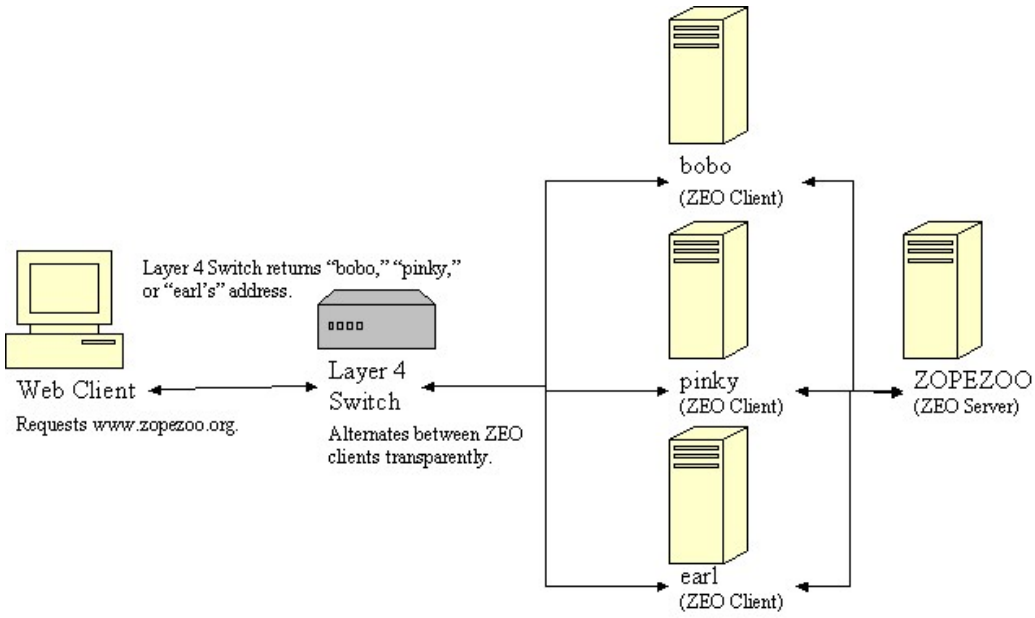
这种解决方法的潜在的一个问题是名称服务器刷新 www.zopezoo.org 的地址缓存会花费几个小时或几天时间。如果你不负责维护 ZEO 客户机，一旦某台机器失效，那么用户中的 1/N（N 是 ZEO 客户机的数量）将不能访问站点，直到名称服务器刷新缓存。

配置DNS服务器来处理round-robin名称解析是相当高级的技术，它不在本书中讲述。有关如何实现这种配置，可参考Apache文档。

用round-robin DNS分配负载不但实用而且价廉，但不是100%有效。DNS服务器有不同的缓存策略，并且你依赖DNS的方式来分配负载。以下一节描述一种更为复杂但是远为强大的分配负载的方法，它被称为Layer 4 交换。

5.3. 使用Layer 4交换分配负载

通过使用Layer 4交换（switching）可以使得一台计算机能够直接把请求传递给多台计算机。这是一种相当高级的技术，它超出了本书的范围。但是，值得指出几种支持Layer 4交换的产品。Layer 4 交换使用一台交换机，在请求进入的时候，根据设置的参数，它在一组ZEO客户机中进行选择，如下图所示。



Layer 4交换 图解

目前，可以使用硬件和软件Layer 4 交换机。现有许多软件解决方法，比较好的一种是Linux Virtual Server (LVS)。这是一个Linux操作系统的扩展，它使你能够把一个Linux计算机转换成一个Layer 4 交换机。在LVS的Web站点上可以找到更多的信息(http://www.linuxvirtualserver.org)。

还有许多硬件解决方法宣称可以提供比软件解决方法更高的性能。Cisco Systems有一种被称为 LocalDirector 的硬件路由器，它起到Layer 4 交换的作用，还有Alteon也制造了一种流行的Layer 4交换机。

5.4. 处理唯一失效点

离开了ZEO，整个Zope系统将是一种唯一失效点（single point of failure）系统。ZEO使你能够把失效点分散给多个不同的计算机。如果某个ZEO客户机失效了，其它客户机可以接替失效的客户机，继续响应请求。

然而，按照典型方式安装的ZEO，仍然有一个失效点，那就是ZEO服务器本身。如果不使用商业软件，这个失效点还不能去除。

一种流行的方法是接受唯一失效点的风险，降低这种风险的方式包括：通过 ZEO 服务器使用高端而可靠的设备，经常备份你的数据，以及 ZEO 客户机使用正规厂商的硬件等等。最主要的是增强 ZEO 服务器的性能（比如使用冗余动力供给、RAID 和其它避免失效的安全措施等等），这样一来可以充分确保 ZEO 服务器运转，即使多个不贵重的客户机失效也没关系。

可是，一些应用程序需要绝对的 100%连续运转。采用以上描述的解决方法， ZEO 服务器失效的机会仍然存在，你需要使用 ZEO 备份服务器，一旦失效，就能够立即接替失效的服务器。

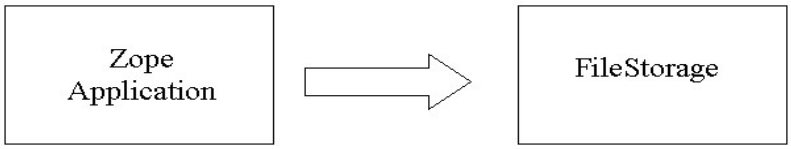
就像 Layer 4 交换一样，有许多软件和硬件产品可用来创建备份存储服务器。有一种适合 Linux 的软件解决方法，它被称为 fake （ http://vergenet.net/linux/fake/）。 fake 是一种基于 Linux 的工具，它通过“仿造出”网络地址可以让一台备份计算机接替失效的计算机。当 fake 和监视工具一起使用时，它可以保证 ZEO 服务器和 Layer 4 交换几乎 100%的时间连续运转。监视工具比如有 mon （ http://www.kernel.org/software/mon/）或者

- heartbeat （ http://www.linux-ha.org/）。关于 fake，请参考相关文档。

5.5. ZEO 服务器细节

最后一个让人模糊的问题是 ZEO 服务器在哪里存储信息。如果 ZEO 服务器失效了，如何确保备份服务器中是 ZEO 服务器中最新的信息？

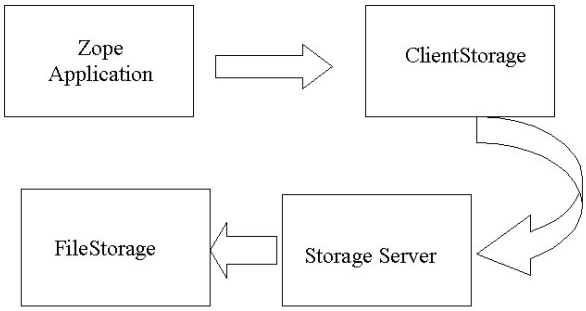
在讲述 ZEO 服务器如何工作的细节以前，需要了解 Zope 存储是如何工作。 Zope 不是直接把对象或信息存储到磁盘中，而是使用一种存储组件，它负责处理如何存储对象。 这是一种非常灵活的模型，这是因为 Zope 不需要关心打开文件，或者读取和写入数据库，以及通过网络发送数据（在 ZEO 情形中）。每种存储负责处理相应的任务。例如，普通的独立运行的 Zope 系统结构如下：

- 

Zope 连接到 FileStorage

你可以看到只有一个 Zope 应用程序，它指向一个 FileStorage 。这个存储就像它的名称所揭示的，把信息保存到计算机文件系统中的一个文件里。

当使用 ZEO 时，只是把 FileStorage 替换成 ClientStorage ，如下图所示。



使用 ClientStorage 和 Storage Server 的 Zope

不是把对象存储成文件， ClientStorage 通过网络连接把对象发送给 Storage Server。就像你在上图中所看到的，Storage Server 使用 FileStorage 把信息存储到 ZEO 服务器的文件系统中的文件里。这个文件位于 var 目录中，名称为 data.fs。

6. ZEO 注意事项

运行 ZEO 类似于运行 Zope，但是需要注意几个问题。

首先，它把信息写入 Zope 对象数据库的时间会相对长一些。这不会降低使用 Zope 的能力（因为 Zope 在这个写入操作期间不会妨碍你），但是它增加了遇到 ConflictError 错误的机会。 ConflictError 错误发生在两个 ZEO 客户机在同一时间试图写入同一对象的时候。其中一个 ZEO 客户机能够正常运转。其它 ZEO 客户机不得不再试一次。

ConflictError 错误应该尽可能少的发生，因为它们将会降低系统性能。尽管有几个 ConflictError 是正常的（由于当前 Zope 的特性），有许多 ConflictError 是不正常的。不正常的情况发生在多个 ZEO 客户机试图非常快速的一次又一次写入相同的对象。在这种情况下，会有许多 ConflictError ，并不断尝试。如果一台 ZEO 客户机试图三次写入数据库并且在一行中得到三个 ConflictError ，那么这个请求被放弃并且不写入数据。

因为 ZEO 花费相对长的时间来写入这个信息，得到一个 ConflictError 的机会高于你不运行 ZEO 的时候。因此，比起不使用 ZEO，使用 ZEO 是更为写入敏感的。当你在设计网络或应用程序时你需要记住这些。作为一个经验原则，频繁的写入数据库增加你得到 ConflictError 的机会。其次，使用快速的计算机和更为可靠的网络连接会降低获得 ConflictError 的机会。 ConflictError 通常可以避免。

最后，在撰写本文的时候，还没有内建的加密方法或者在 ZEO 服务器和客户机之间进行验证的方法。这意味着你必需非常小心的警惕使用 ZEO 服务器的人们。如果你把 ZEO 服务器开放给整个 Internet，那么任何人可以连接到 ZEO 服务器并把数据写入你的数据库。

然而，这是一个可以解决的问题，因为你可以使用其它的工具，例如防火墙，来保护 ZEO 服务器。如果你正在通过一个不安全的网络运行一个 ZEO 客户机/服务器连接，你需要保证信息保持私有状态，你可以使用工具，例如 OpenSSH (http://www.openssh.org) 和 stunnel (http://www.stunnel.org/) 来在 ZEO 客户机和服务器之间建立安全、加密通讯通道。这些工具如何工作以及如何设置它们超出了本书的范围。在它们的 Web 站点上为这两个工具提供了详细的文档。

第二十一章 用其它工具管理 Zope 对象

前边讲述了如何通过 Zope 提供的管理界面处理对象。接下来，本章讲述如何不通过浏览器方式访问和修改 Zope 中的内容。

在 Zope 管理界面中编辑内容和代码不是很方便。除了浏览器，Zope 支持使用其它的外部工具软件来编辑内容和代码。

1. 需要知道的事项

许多编辑软件可以处理文件，但 Zope 对象并不是严格意义上的文件，所以需要知道：

Zope 的数据不是按照文件进行存储的。因此编辑软件和 Zope 对象数据库之间应该通过一些方式进行连接。常见的连接方式是使用 FTP 和 WebDAV 协议。

当创建对象时，Zope 不要求必须使用文件扩展名。一些工具软件要求必须使用文件扩展名，比如 Macromedia Dreamweaver。因此你可以根据对象的类型，在对象名称中加入扩展名，比如给 ZPT 对象添加.html。

创建新对象时，有可能会产生错误。由于 Zope 不是根据扩展名来创建对象的，因此新创建的对象有可能不是所需的类型。比如，通过 FTP 上载 HTML 文件"foo.html"，将默认创建 DTML 文档对象，而你所需的可能是 Zope 页面模板对象。Zope 中提供了指定对象类型的方法，将在后面的章节中讲述。

外部工具不能处理 Zope 对象属性。如果你通过外部工具修改对象，将会忽略属性列表。

一些外部工具的特性可能会对 Zope 产生不好的影响。比如，一些工具软件会创建 Zope 不能识别的备份文件。另外，还有可能生成不必要的对象。

使用外部工具不能提供错误信息。因此，对象中存在的错误，比如页面模板中的语法错误，就不能显示给用户。

这些工具很多，各自都有各自的特点。

- 注意，使用外部工具，Zope 可能会对文件产生影响。相同的文件复制到 www.zope.org 或本地 Zope 中时，可能产生不同的结果。比如，在 CMF 中，Zope 将添加元数据。

通过使用外部的编辑工具可以管理大多数的 Zope 对象。

2. FTP 和 WebDAV

Zope 中大多数类似文件的对象，比如 DTML Methods, DTML Documents, Zope Page Templates, Script (Python)等等，可以通过 FTP 和 WebDAV 进行编辑。许多编辑器软件支持编辑远程服务器中的文档。它们的有点和缺点为：

FTP

FTP 就是文件传输协议。FTP 用于在计算机之间传送文件。许多编辑器软件都支持 FTP。

WebDAV

WebDAV 是一种新的 Internet 协议，它基于 HTTP 协议。DAV 的含义是 Distributed Authoring and Versioning（分布式编著和版本）。支持 WebDAV 的编辑器软件不如 FTP 那样多。

关于 WebDAV 可以参考 webdav.org。

3. 使用 FTP 管理 Zope 内容

许多编辑器软件都支持 FTP。可以通过 FTP 来管理 Zope 对象。

3.1. 找到 FTP 端口

在启动 Zope 的时候，可以看到 FTP 所使用的端口号：

```
------

2000-08-07T23:00:53 INFO(0) ZServer Medusa (V1.18) started at Mon Aug  7

53 2000

        Hostname: peanut

        Port:8080

------

2000-08-07T23:00:53 INFO(0) ZServer FTP server started at Mon Aug  7   16:00:53 2000

        Authorizer:None

        Hostname: peanut

        Port: 8021

------

2000-08-07T23:00:53 INFO(0) ZServer Monitor Server (V1.9) started on port 8099
```

上边的信息显示 FTP 服务器正在监听 8021 端口。

3.2. 使用 WS_FTP 传输文件

现有许多 FTP 客户端程序，以及许多 Web 浏览器，例如 Netscape 和 Microsoft Internet Explorer，自带有 FTP 客户端程序。本节也适用于其它的 FTP 客户端程序。

WS_FTP 是一个适合 Windows 平台的 FTP 客户端程序，用来把文档和文件上载到 Zope 里。WS_FTP 可以从 Ipswitch 主页（http://www.ipswitch.com/）下载。

当你启动 WS_FTP 以后，需要知道 FTP 服务器的名称和端口信息。输入服务器名称和端口，点击 connect 按钮，然后要求输入用户名和密码。在这里输入管理员用户名和密码。

如果输入正确，则会显示 Zope 中的文件夹和文档，以及其它对象。如下图所示：

通过 FTP 查看 Zope

使用 WS_FTP 上传和下载 Zope 文件很容易。在 WS_FTP 窗口的左边是一个文件选择框，其中显示了本地机器上的文件。在 WS_FTP 右边窗口中的文件选择框显示了 Zope 系统里的对象。选中你想传送的文件，然后单击左箭号（下载）或者右箭号（上载）就可以上传或下载对象。

4. 远程编辑

4.1. 使用 Emacs 的 FTP 模式编辑 Zope 对象

Emacs 是一个非常流行的文本编辑器。有两种风格的 Emacs，GNU Emacs 和 XEmacs。这两种都可以通过 FTP 直接操纵 Zope 文档和其它文本内容。 Emacs 使你能够像文件系统那样处理远程 FTP 系统，使得远程管理 Zope 内容变得很容易。 要登录进入 Zope，先运行 Emacs。用于打开 FTP 连接的文件取决于你正在运行哪一种文本编，XEmacs 或 Emacs：

Xemacs

要想用 Xemacs 访问远程目录，同时按 Ctrl-X D 键，然后按照以下格式输入目录名：/user@server#port:/ 。这样就会打开一个连接窗口，其连接到运行在 server 中的监听端口 port 的 FTP 服务器的 / 目录。

Emacs

要想用 Emacs 访问远程目录，同时按 Ctrl-X D 键，然后按照以下格式输入目录名：/user@server port:/ 。通过按下 Control 键和 Q 键，然后按下空格键，插入文字空格。

对于典型安装的Zope，使用XEmacs开启一个FTP对话的文件名为：/user@localhost#8021:/ Emacs要求你输入登录Zope FTP服务器的密码。如果你访问Zope 里的FTP服务器的 / 文件夹，Emacs列出根目录文件夹中的内容：

查看Zope根文件夹中的内容

然后你就可以使用Emacs编辑对象了。你也可以创建新的对象，如果没有指定类型，默认为DTML文档。

在Windows中使用Emacs需要指定FTP程序，需要下载并安装Cygwin，还要配置.emacs，比如：

```
(setq ange-ftp-ftp-program-name "/cygwin/bin/ftp.exe")

(setq ange-ftp-try-passive-mode t)

(setq ange-ftp-ftp-program-args '("-i" "-n" "-g" "-v" "--prompt" ""))
```

4.2. 用 WebDAV 编辑 Zope 对象

WebDAV是一种HTTP协议的扩展协议，它针对于众多用户同时在Web站点中编著和编辑内容，提供了丰富的特性。例如锁定、修订控制和用属性标记文档或对象等等。因为WebDAV要实现通过Web编辑内容，这个目标和Zope的一些目标相符，所以，Zope很早就开始支持WebDAV协议。

相对于HTTP或者FTP而言，WebDAV是一种比较新的Internet协议，因此支持它的客户端程序比较少。然而WebDAV发展很快并且一直都在开发更多的客户端程序。

WebDAV 协议发展迅速，新特性不断的被添加进来。你能用任何一种 WebDAV 客户端程序编辑你的 DTML 文档，只需给你的客户端程序指出文件的 URL，然后就可以编辑它。但是对于大多数客户端程序，这会让它们试图编辑文档生成的结果而不是文档源文件本身。对于 DTML 或 ZPT 来说，这可能有问题。

直到客户端程序赶上最新的 WebDAV 标准，才能能够理解文档源文件和其结果的不同。Zope 提供了一种特殊的 HTTP 服务器，你可以使用 -W 命令行选项使其生效。不同于 HTTP 服务器，这个服务器监听另外一个端口，并且为从那个端口进入的 WebDAV 请求返回不同的特定的源文件内容。

4.2.1. 注意

Zope 2.6 通过配置，可以实现通过 HTTP 端口返回源文件。方式是检查 HTTP 请求的用户代理起始部分。如果发现了和服务器中配置相匹配的字符串，则返回源文件。

有一个支持 WebDAV 的工具软件，名称为 cadaver，可在 WebDAV.org 找到。 比如，通过以下命令可以访问 WebDAV：

```
$ cadaver

dav:!> open http://saints.homeunix.com:9800/

Looking up hostname... Connecting to server... connected.

Connecting to server... connected.

dav:/> ls

Listing collection `/': (reconnecting...done) succeeded.

Coll:  Control_Panel                        0  Jun  14:03

Coll:  Examples                             0  Jun  14:01

Coll:  ZopeBook                             0  Jul  22:57

Coll:  temp                                 0  Jul   2002

Coll:  temp_folder                          0  Jul  19:47

Coll:  tutorial                             0  Jun  00:42

       acl_users                            0  Dec   1998

       browser_id_manager                   0  Jun  14:01

       index_html                          93  Jul  01:01
```

```
        session_data_manager                    0  Jun  14:01
        standard_error_message               1365  Jan   2001
        standard_html_footer                   53  Jan   2001
        standard_html_header                   80  Jan   2001
        standard_template.pt                  282  Jun  14:02
dav:/>
```

还可以在 cadaver 中直接编辑文件：

```
dav:/> edit index_html
Connecting to server... connected.
Locking `index_html': Authentication required for Zope on server `saints.homeunix.com':
Username: admin
Password:
Retrying: succeeded.
Downloading `/index_html' to /tmp/cadaver-edit-001320
Progress: [============================>] 100.0% of 93 bytes succeeded.
Running editor: `vi /tmp/cadaver-edit-001320'...
```

这样可以直接调用 vi 编辑器。通过设置系统中的 EDITOR 环境变量可以指定编辑器。 通过 cadaver 还可以传输文件。 还有很多支持 WebDAV 的软件，比如 Macromedia Dreamweaver 和 Microsoft Office。详细请参见 WebDAV.org。

5. 使用 PUT_factory 指定对象类型

在使用 FTP 或 WebDAV 的时候，常常需要指定对象类型。这个过程被称为"PUT"。 创建对象时所采用的默认方式为：

内容类型	创建对象类型
'text/{anything}'	创建一个 DTML Document

'image/{anything}'　　创建一个 Image object

'{anything else}'　　创建一个 File object

Zope 允许你修改这个默认的设置，方法是在需要这个特性的文件夹中创建一个名为"PUT_factory"的脚本或外部方法。

以下这个例子中，在根文件夹中添加了一个外部方法对象，它的作用是当遇到 PUT 类型为"text/html" 或 "text/plain"时创建页面模板对象，而不是默认创建 DTML 文档对象。

```
from Products.PageTemplates.ZopePageTemplate import ZopePageTemplate

def PUT_factory(self, name, typ, body):

    if typ.startswith('text'):

        return ZopePageTemplate( name, text=body, content_type=typ )
```

在 Zope 安装目录中的 Extensions 文件夹中，添加一个具有上边代码的文件，名称 PUT_factory.py。然后在 Zope 根文件夹中创建一个外部方法对象，id 为 PUT_factory，title 为"PUT factory for Page Templates"，Module Name 和 Function Name 为 PUT_factory。添加完成以后，通过 FTP 或 WebDAV 传送的类型为 text/{anything}的文件就会生成页面模板对象。其它类型，比如图片，仍保持不变。

6. 使用 External Editor

Casey Duncan 编写了一个很实用的 Zope 产品，名为 External Editor。它可以让用户在浏览器中查看对象的时候，通过点击对象旁边的铅笔图标启动指定的编辑器，不同的对象可以使用不同的编辑器软件进行编辑。它需要在本机中安装客户端组件，以及在 Zope 中安装一个服务器端组件。 使用 External Editor 可以使用现有的编辑工具软件在管理界面中编辑对象。

External Editor 界面

第二十二章 扩展 Zope

Zope 可以进行扩展，方式是根据应用程序的特定需要创建新对象类型。新对象类型以产品（Products）的形式出现。产品是由 Zope 公司和许多其他第三方开发者为 Zope 创建的扩展程序。现在已经有几百个不同用途的产品。完整的产品库可以在 Zope.org 的下载部分找到(http://www.zope.org/Products/)。

产品可以通过两种方法开发：通过 Web 使用 ZClass 和采用 Python 程序语言。产品还可以是 Web 产品和 Python 代码的混合物本章讲述通过 Web 方式构建新产品，这个主题在“搜索和分类内容”中已经讲过了一些。完全采用 Python 产品编程的方式开发一个产品超出了本书的范围，你应该查看 Zope.org 中的产品开发者文档。

本章主要讲述：

在 Zope 中创建新产品

在产品中定义 ZClass

ZClasse 结合 Python

把产品分发给其它用户

1. 创建 Zope 产品

Web 产品存储在控制面板中的 Product Management（产品管理）文件夹里。单击根文件夹中的 Control_Panel，然后单机 Products。你会看到如下图所示的内容。

已经安装的产品

每个蓝色的盒子代表了一个已经安装的产品。从这个屏幕中，你可以管理这些产品。一些产品被默认内建到 Zope 中，而其它的产品由你或你的管理员安装。这些产品有一个关闭的盒子图标，如上图所示。图标显示为关闭的盒子的产品不能通过 Web 来管理你可以通过单击这些产品来得到它们的信息，但是你不能更改它们。

你可以创建自己的可以通过 Web 管理的产品。你的产品使你能够在 Zope 中创建新类型的对象。这些通过 Web 管理的产品有显示为打开的盒子的图标。如果你完成了“内容搜索和分类”中的例子，那么你就有了一个 News 产品，它的图标显示为一个打开的盒子。

你为什么需要创建产品呢？让我们举例说明，动物园中的每个看管者都需要一种构建简单在线展览的轻松方法。这些展览都必需有相同的格式和包含相似的信息结构，并且每一个展览都与动物园中的某种动物相关。

要实现这个功能，你可以为某种动物构建一个展览，然后为其它的每个展览复制和粘贴它，可是这将是一个繁琐的手工过程。所有的信息和属性将不得不为每个新展览而修改。甚至可能会有数千个展览。

这个问题还会更棘手，让我们比方说你需要给每个展览添加信息，告诉此种动物是否濒临灭绝。如果你使用复制和粘贴，你将不得不修改每个展览，一个接一个的添加这些信息。很明显，对于一个非常大的动物园来说复制和粘贴不会非常有效，并且将会非常费力。

你还需要确保每个展览是容易管理的。每个单独的展览的看管者应该是提供信息的人，但是动物园看管者对 Zope 或如何创建 Web 站点所知甚少，他们当然不希望浪费时间来学习。你只希望让他们在表单中输入一些简单的他们所感兴趣的信息，然后单击提交就可以了。

通过创建 Zope 产品，你可以快速而轻松的实现这些目标。你可以创建容易管理的对象，方便看管者使用。你可以定义展览模板，这样你可以更改一次就对所有的展览生效。你可以通过创建 Zope 产品做到这点。

2. 创建一个简单的产品

使用产品可以解决展览创建和管理问题。让我们用一个例子开始，它显示了如何创建一个简单的产品，这个产品使你能够收集展览信息和创建定制的展览。

Zope 产品的主要价值在于使你能够在一个统一的地方创建对象，并且让你通过产品添加列表来使用对象。这样就赋予你一种能力，它让你能够通过 Zope 的标准的管理界面构建通用型的服务。换句话说，通过产品使你能够定制 Zope。

让我们从控制面板中的产品文件夹开始。要创建一个新产品，单击产品管理文件夹中的 Add Product 按钮。这带你进入到产品添加表单。输入 id 为 ZooExhibit ，然后单击 Generate。你在产品管理文件夹中就会看到新产品。它的图标应该是一个带有敞开的盖子的蓝色盒子。打开的盖子意味着你可以单击这个产品并且可以通过 Web 管理它。

选中 ZooExhibit 产品，进入产品管理部分。一个产品的管理屏幕看起来就像一个文件夹，一些区别如下：

在最右边有一个名为 Distribution 的新视图。它用于产品打包和发布。这个视图在本章后边的“发布产品”一节中详细讲述。

如果你选择了 Add 列表，你会看到一些你可以添加的新对象类型，包括 ZClass、 Factory 和 Permission。

1. 带有一个问号的文件夹是 ZooExhibit 产品的帮助文件夹。这个文件夹可以用于存放产品的帮助文档。

还有一个名为Define Permissions（定义许可）的新视图，它定义与这个产品相关的许可。这是高级的特性，对于这个例子不是必需的。

在Contents视图里，创建一个名为hello的DTML方法，其内容如下：

```
<dtml-var standard_html_header>

<h2>Hello from the Zoo Exhibit Product</h2>

<dtml-var standard_html_footer>
```

这个方法用于测试你的产品。接下来，创建一个Factory，从产品添加列表中选择Zope Factory。出现Factory添加表单，如下图：

添加一个Factory

这种工厂对象可以在产品添加列表和你的产品之间创建一个桥梁。在id中输入myFactory。在Add list name（添加列表名称字段中输入Hello，在Method选择框中，选择hello。单击Generate。现在单击新工厂，修改Permission为Add Document, Images, and Files，然后单击Save Changes。这样就告诉Zope你必需拥有Add Documents, Images, and Files许可才能使用这个工厂。祝贺你，你刚才定制了Zope管理界面。进入根文件夹并单击产品添加列表。你会发现它包括了一个名为Hello的条目。从产品添加列表中选择Hello，它调用hello方法。

工厂的主要作用是把对象复制到当前的文件夹里。换句话说，你的方法可以访问它们被调用时所在的位置，并且它们可以对那个文件夹执行操作，包括把对象复制到其中。仅仅因为你可以用工厂和产品进行各种操作并不意味着你就应该这样。通常当人们从产品添加列表中选择了一个新对象时才期望使用能够指定id的添加表单。然后期望当单击Add时在他们的目录中创建一个具有指定id的新对象。因此，让我们看看如何满足这些期望。

首先在你的产品中创建一个名为 exhibitTemplate 的新文件夹。它将作为展览的模板。另外，在产品文件夹中，创建一个名为 addForm 的 DTML 方法和一个名为 add 的 Python 脚本。这些对象用于创建新展览实例。现在，返回工厂并修改它，为的是把添加列表名称变为 Zoo Exhibit 和把方法变为 addForm。

现在希望实现的是当某个人从产品添加列表中选择 Zoo Exhibit 时，就运行 addForm 方法。这个方法应该收集关于展览的 id 和 title 的信息。当用户单击 Add，addForm 方法应该调用 add 脚本，它把 exhibitTemplate 文件夹复制到当前文件夹中，并且把它重命名为指定的 id。下一步是编辑 addForm 方法，代码如下：

```
<dtml-var manage_page_header>

  <h2>Add a Zoo Exhibit</h2>

  <form action="add" method="post">

  id <input type="text" name="id"><br>

  title <input type="text" name="title"><br>

  <input type="submit" value=" Add ">

  </form>

<dtml-var manage_page_footer>
```

这段代码是一段相当简练的添加表单。当你创建自己的 Web 应用程序时，你可以对它进行扩充。 注意这个方法没有包括通常的 HTML 页眉和页脚。根据惯例，Zope 管理屏幕不使用普通的站点中那种页眉和页脚，而是使用 manage_page_header 和 manage_page_footer。这样就可以确保管理视图有统一的外观。另外，你还会注意到表单的行为是 add 脚本。在 add 脚本中输入以下内容，并输入参数：id ,title, REQUEST=None：

```
# Clone the template, giving it the new ID. This will be placed

# in the current context (the place the factory was called from).

exhibit=context.manage_clone(container.exhibitTemplate,id)

# Change the clone's title

exhibit.manage_changeProperties(title=title)

# If we were called through the web, redirect back to the context

if REQUEST is not None:

    try: u=context.DestinationURL()
```

```
except: u=REQUEST['URL1']

REQUEST.RESPONSE.redirect(u+'/manage_main?update_menu=1')
```

这段脚本复制exhibitTemplate，并且用指定的id把它复制到当前的文件夹中。然后它更改新展览的title属性。最终，它通过调用调用manage_main返回当前文件夹的主管理屏幕。

祝贺你，你已经通过创建一个新产品扩展了Zope。你已经创建了一个方法，这个方法通过产品添加列表把对象复制到Zope。然而，这个方法仍然存在本章前边论述过的一些问题。尽管你可以在一个集中的地方编辑展览模板，但是它只是一个模板。因此如果你给模板添加一个新属性，它不会影响任何现有的展览。要修改现有的展览，你必需手工修改每一个。

使用ZClass可以解决这个问题，它可用来提供核心模板。当你更改ZClass时，所有那种类型的对象随之更改。在下一节，我们给你展示如何创建定义ZClass。

3. 创建ZClass

ZClass是一种工具，它可以通过定义类来构建新对象类型。类就像是对象蓝图。当你定义一个类时，就是在定义将创建什么样的对象。一个类可以定义方法、属性和其它特性。

根据某个类创建的对象被称为那个类的实例。例如，只有一个Folder类，但是你可以在应用程序中有许多Folder实例。

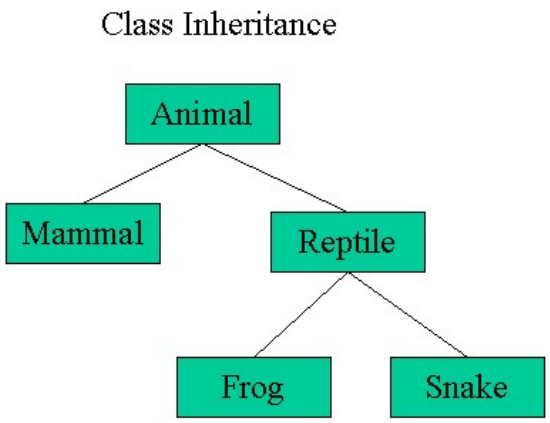
实例和它们的类一样有相同的方法和属性。如果你更改类，那么所有的实例都会表现出所做的更改。不同于上一节中所创建的模板，类将会继续控制实例。记住这只是单向工作。如果你更改一个实例，不会更改类或其它任何实例。

ZClass在现实世界中一个好的类推是文字处理模板。大多数文字处理软件带有一套预先设定好的模板，你可以使用这些模板来创建某种类型的文档，例如简历。世界上有成百上千的简历可能基于Microsoft Word Resume模板，但是只存在一个模板。一个ZClass 可看作是许多相似对象的模板。

ZClass是可以通过Web使用Zope的管理界面构建的类。另外，类还可以用Python编写，本书不对其进行详细阐述。

ZClass可以从其它类继承属性。继承使你能够基于其它类定义一个新类。例如，比方说你需要创建一种新类型的文档对象，它的一些属性可以通过继承来获得。此时，不用完全从空白开始构建文档功能，你可以从DTML Document 类继承那个功能，然后只添加你所感兴趣的新信息。

通过继承可以在类之间构建关系。例如，你可以创建一个名为Animal的类，它包含所有动物通常所具有的信息。然后，你可以创建Reptile 和 Mammal 类，它们从Animal类继承。甚至，你还可以创建另外两个类Lizard 和 Snake，它们从Reptile类继承，如图所示。



类继承的例子

ZClass 可以继承大多数 Zope 对象。另外，ZClass 还可以继承同一产品中定义的其它的类。

在进行下一个例子以前，你应该把 Zope 产品文件夹中的现有的 ZooExhibit 产品更名为别的名称，例如 ZooTemplate ，为的是不和这个例子冲突。现在在产品文件夹里创建一个名为 ZooExhibit 的新产品。

从 ZooExhibit 的 Contents 视图中的添加列表选择 ZClass ，然后进入 ZClass 添加表单。这个表单有点复杂，作用如下：

Id

- 这是要创建的类的名称。对于这个例子，选用 ZooExhibit 。

Meta Type

这是对象类型的名称。用于简要的描述对象的作用。对于这个例子，输入"Zoo Exhibit"。

Base Classes

- Base classes 定义类序列，其中的类用来继承属性。新类可以认为是扩展或源自于基础类。你可以从左边的列表中选择一个或多个类，然后单击 -> 按钮，把它们放在基础类列表中。<-按钮用于删除在右边选择的任何基础类。对于这个例子，不用选择任何基础类。在本章的后面，我们解释一些更为有趣的基础类，例如 ObjectManager 。

Create constructor objects?

通常需要选中这个选项，除非你需要亲自创建 form/action 和 Factory 对象。如果你想让 Zope 为你完成这个任务，保持选中状态。选中这个框意味着这个 Add 表单创建 5 个对象：一个 Class、一个 Constructor Form、 一个 Constructor Action、 一个 Permission 和一个 Factory。对于这个例子，保持这个框为选中状态。

Include standard Zope persistent object base classes?

这个选项应该选中，除非你不想把对象存储在数据库中。这是一个高级选项并且只适用于 Plugable Brains。对于这个例子，保持这个框为选中状态。

现在单击 Add。这会带你返回 ZooExhibit 产品，你会看见 5 个新对象，如图所示。

使用 ZClass 的产品

这 5 个对象都被自动的配置好了——你目前不需要更改它们。以下是每个被对象的简要说明：

- ZooExhibit

这是 ZClass 本身，它的图标是一个带有两个水平线的白色盒子。这是类的传统符号。

- ZooExhibit_addForm

这个 DTML 方法是 ZClass 的构造表单。它是只使用 id 和 title 参数。你可以定制这个表单，加入更多的参数。它类似于第一个例子中创建的 Add 表单。

- ZooExhibit_add

这个 DTML 方法由构造表单 ZooExhibit_addForm 调用。这个方法用于创建新对象并设置它的 id 和 title。你可以定制这个表单。它的功能和我们在前边的例子中所创建的 Python 脚本相同。

- ZooExhibit_add_permission

许可的图标显示为正在搬运蓝盒子的独特的细线小人。它定义了一个与添加新 ZooExhibit 对象相关联的许可。这使你能限制添加新动物园展览的能力。如果你单击这个许可，你可以看见这个新许可的名称为 Add ZooExhibits。

- ZooExhibit_factory

带有一个小烟囱的小工厂图标表示一个工厂对象。如果你单击这个对象，你可以在 Add list name 框中更改这个对象在添加列表中所显示的文本。Method 是指当用户从添加列表中选择对象时所调用的方法。通常是对象的构造表单，在这个例子中为 ZooExhibit _addForm。你还可以要求用户必须具有相应的许可才能添加这个对象，在这个例子中为 ZooExhibit _add_permission。你也可以指定一个普通的 Zope 许可。

这样，你已经创建了第一个 ZClass。单击这个 ZClass ，然后单击它的 Basic 标签。其中可以用来更改信息。你不能更改 ZClass 的基础类。就像你在本章前边所学到的，这些设置包括：

meta-type

ZClass 在产品添加列表中所显示的名称。

class id

类的唯一标识。如果你需要使用其它的 ZClass，修改这个就可以。

icon

类图标的路径。几乎没有必要来修改它。如果你需要更改类的图标，通过 Browse 按钮上载一个新文件即可。

至此，你可以开始创建新的 ZooExhibit ZClass 实例。首先需要创建一个公用位置，在此定义所有的展览。为此，进入你的根文件夹，然后从添加列表中选择 Folder 。创建一个新文件夹，id 为 Exhibits。现在单击你刚才创建的 Exhibits 文件夹，然后下拉添加列表。你可以看到，添加列表中有了 ZooExhibit 。

继续从添加列表中选择 ZooExhibit ，然后创建一个 id 为 FangedRabbits 的展览。然后通过单击来选择它。

可以看到，对象已经有了三个视图：Undo、Ownership 和 Security。不需要自己来定义这些视图，这是 Zope 自动完成的。在接下来的一节，添加其它几个用于编辑对象的视图。

3.1. 创建 ZClass 视图

根据作用不同，Zope 对象提供不同的管理屏幕，这些屏幕被称为视图。一般在管理界面中处理 Zope 对象时使用视图，即 Zope 对象中显示的带有标签的屏幕是视图。一些视图，比如 Undo，是内置的很多对象都有的视图。

视图是在 ZClass 的 Views 视图中定义。进入你的 ZooExhibit ZClass，然后单击 Views 标签。Views 视图看起来如下图所示。

Views 视图

在这个视图里，你可以看到三个随同新对象自动产生的视图：Undo、 Ownership 和 Security。因为几乎所有的对象都有这三个界面，所以为了方便，它们已经配置好了，如果需要的话，可以更改它们或者从这个视图中删除它们。但是通常不需要这样做。

视图表格被分割成三个列：Name、 Method 和 Help Topic 。Name 是视图的名称并且是管理界面中视图标签上的用于下拉的标注文字。Method 是类方法或属性单，用于显示视图。在 help topic 这个地方，可以把一个 Help Topic 对象与这个视图相结合 Help Topic 在“为 ZClass 提供上下文相关的帮助”一节里进行更为详细的讲解。

视图还和安全系统一起工作，这样就确保用户只看到他们有权看到的对象视图。视图不仅把对象管理界面划分成逻辑块，同时它们还控制谁能够看见哪一个视图。

在 Methods 视图上的 Method 列有中可以选择什么方法生成什么视图。与一个视图相关联的方法可以是 Methods 视图里的一个对象或 Property Sheets 视图中的一个属性单。

3.2. 创建 ZClass 的属性

属性是一些用来存储对象信息的变量。例如对于 ZooExhibit 对象，需要用到包含有展览相关信息的属性，比如展览什么动物，简介和看管者等等。

ZClass 属性略微区别于 Zope 对象属性。在 ZClass 里，属性是在属性单定义。属性单是一种组织属性的方法。进入 ZooExhibit ZClass，并且单击 Property Sheets 标签。要创建一个新属性单，单击 Add Common Instance Property Sheet。然后就会出现 Property Sheet 添加表单。给新属性单命名为 ExhibitProperties ，然后单击 Add 。

现在你可以看到新属性单 ExhibitProperties 已经在你的 ZClass 的 Property Sheets 视图里创建。单击新的属性单来管理它，如图所示。

属性单

你可以看到，这个属性单看起来非常类似于 Zope 对象的属性视图。你可以在这个属性单中创建新属性。属性单中的属性就像是 Zope 对象属性，它们都有名称、类型和值。

在这个属性单中创建以下三个新属性：

animal

这个属性的类型应该为字符型。用来保存要展出的动物的名称。

description

这个属性类型应该为文本型。用来保存展览简介。

caretakers

这个属性类型应该为多行型（lines）。用来提供展览看管者名称列表。

属性单有两个用途。如同你在这个例子中看到的，它们是组织对象属性的工具，第二个用途是，用来编辑那些属性，即同时生成 HTML 表单，以及相应的编辑属性所需的行为。HTML 编辑表单自动生成，因此要想看到属性单的编辑表单，只需要把一个视图关联给属性单。例如，返回 ZooExhibit ZClass，单击 Views 标签，创建一个名为 Edit 的新视图，然后关联 propertysheets/ExhibitProperties /manage_edit 方法。

你可以使用属性单来创建编辑屏幕，当然也可以创建多个属性单。通过使用多个属性单，你可以控制哪些属性可以一起来编辑。对于不同的属性单你还可以通过把属性和不同的许可相关联，从而把私有的属性从公共的属性中分离出来。

现在，返回你的Exhibits文件夹，或者查看现有的 ZooExhibit 实例，或者创建一个新的实例。你可以看到，一个新的名为Edit的视图已经添加进来，如图所示。

ZooExhibit编辑视图

这个编辑表单已经自动生成了。你只需要创建属性单并把它和一个视图相关联。如果你把其它的属性添加到ExhibitProperties 属性单，所有实例自动得到一个自动更新的编辑表单，这是因为当你更改一个ZClass时，所有类实例继承所做的更改。

对类所做的更改影响所有的实例，对于实例的更改不影响类或者任何其它实例。例如，在 ZooExhibit 实例（不是类）的Edit 视图上，为animal属性输入"Fanged Rabbit"，属性description中输入"Fanged, carnivorous rabbits plagued early medieval knights. They are known for their sharp, pointy teeth"，在caretaker中输入"Tim" 和 "Somebody Else"。现在单击Save Changes。

你可以看到，这些更改影响了这个实例，但是对于类发生了什么？返回 ZooExhibit 类，并且查看 ExhibitProperties 属性单。没有任何变化！对实例所做的更改不影响类。

你还可以为属性单上的属性提供默认值。例如，你可以在 ZooExhibit ZClass的description属性中输入文字"Describe your exhibit in this box"。现在，返回到你的 Exhibits 文件夹并且创建一个新的 ZooExhibit 对象，然后单击它的Edit 视图。这里，你看到在属性单里提供的值是实例的默认值。记住，如果你更改这个实例，属性单中的属性默认值不变。默认值使你能够在ZClass 中为属性设置有用的信息，这些信息以后就可以在实例中进行更改。

返回到你的ZClass，单击Views 标签，然后单击First 按钮，这样 Edit 视图成为了第一个显示的视图。现在，当你单击你的实例时，它们首先显示 Edit 视图。

3.3. 创建 ZClass 方法

ZClass 的 Methods 视图使你能够为 ZClass 实例定义方法。进入你的 ZooExhibit ZClass，然后单击 Methods 标签。如下图所示：

Methods 视图

你可以在 Methods 视图上创建任何类型的 Zope 对象，但是通常，只添加可调用的对象（比如 DTML 方法和脚本）。

方法的用途在于：

表现（Presentation）

当把一个视图关联一个方法，于是当用户选择实例中相应的视图时，就调用这个方法。例如，如果你有一个名为 showAnimalImages 的 DTML 方法和一个名为 Images 的视图，你可以把 showAnimalImages 方法关联给 Images 视图。某个人任何时候单击你的 ZClass 实例中的 Images 视图，就会调用 showAnimalImages 方法。

逻辑（Logic）

- 方法不是必需关联给视图。方法经常用来定义如何处理对象。例如，看看将在后边讲述的 ZooExhibit ZClass 的 isHungry 方法。它不为 ZooExhibit 定义一个视图，它只用来提供有关 ZooExhibit 的非常特定的信息。ZClass 中的方法可以相互调用，就像任何其它 Zope 方法一样，因此，表现方法中可以使用逻辑方法，尽管这些逻辑方法不定义视图。

共享对象（Shared Objects）

就像以前所指出的，在 ZClass 的 Methods 视图中可以创建任何类型的对象。ZClass 的所有实例共享 Methods 视图中的对象。例如如果你在 ZClass 的 Methods 视图里创建了一个 Z Gadfly Connection ，那么那个类的所有实例将共享相同的 Gadfly connection。被共享的对象可用在逻辑或表现处理中。

对于用作表现的方法，一个比较好的例子是用来给 Web 站点的访问者显示 Zoo Exhibit 的 DTML 方法。这经常被称为对象的公共界面。对于大多数 Zope 对象而言，常常与 View 视图相关联。

在 ZooExhibit ZClass 的 Methods 视图中创建一个名为 index_html 的 DTML 方法。这将成为对象实例的默认页面。把以下 DTML 输入到你刚才创建的 index_html 方法里：

```
<dtml-var standard_html_header>

  <h1><dtml-var animal></h1>

  <p><dtml-var description></p>

  <p>The <dtml-var animal> caretakers are:<br>

    <dtml-in caretakers>

      <dtml-var sequence-item><br>

    </dtml-in>

  </p>

<dtml-var standard_html_footer>
```

现在你可以直接通过 Web 访问 ZooExhibit 实例中的一个，例如， http://www.zopezoo.org/Exhibits/FangedRabbits/ 将给你显示 Fanged Rabbit 展览的公共界面。

你可以使用基于 Python 或基于 Perl 的脚本，甚至是 Z SQL 方法，来实现逻辑。逻辑对象可以相互调用，并且它们可以在表现方法中调用。要创建 isHungry 方法，首先在 ExhibitProperties 属性单里创建两个新属性，一个名为 last_meal_time，它是日期类型，另外一个名为 isDangerous，它是布尔类型。这样就可以给你的 Edit 视图添加两个新字段，在这里你可以输入喂动物的最新时间以及选择动物是否为危险的。

以下是一个用 Python 实现 isHungry 方法的例子：

```
## Script (Python) "isHungry"

##

Returns true if the animal hasn't eaten in over 8 hours
```

```
from DateTime import DateTime

if (DateTime().timeTime()

     * container.last_meal_time.timeTime() >  60 * 60 * 8):

    return 1

else:

    return 0
```

这个方法中的container表示ZClass实例。因此，你可以使用container表示一个ZClass实例，就如同在python方法中使用self一样。

你可以在index_html中使用以下这段DTML调用这个方法：

```
<dtml-if isHungry>

  <p><dtml-var animal> is hungry</p>

</dtml-if>
```

你甚至可以在显示方法中调用多个逻辑方法。例如，你可以这样改进：

```
<dtml-if isHungry>

  <p><dtml-var animal> is hungry.

  <dtml-if isDangerous>

    <a href="notify_hunger">Tell</a> an authorized

    caretaker.

  <dtml-else>

    <a href="feed">Feed</a> the <dtml-var animal>.

  </dtml-if>

  </p>

</dtml-if>
```

你的显示方法现在通过调用逻辑方法来判定合适的行为，并且创建相应的链接。

3.4. ObjectManager ZClasses

如果选择 ObjectManager 作为 ZClass 的基础类，那么你的类实例就能够像文件夹那样包含其它的 Zope 对象。容器类区别于其它 ZClass 的地方在于，它们有视图：Subobjects。

在 Subobjects 视图中，你可以控制实例中能够包含什么类型的对象。例如，如果你创建一个 FAQ 容器类，你希望把它限制为存储 Question 和 Answer 对象。从选择列表中选择一个或多个元类型（metatypes ），然后单击 Change 按钮。选项"Objects should appear in folder lists"控制容器类实例是否作为可扩展对象显示在导航面板中。

容器 ZClass 可以变得非常强大。一个非常普遍的 Web 应用程序模式是让两个类共同工作。一个类实现基础行为并存储数据。另外一个类包含基础类实例,并且组织和列出这些实例。以这种方式能够对许多问题建立模型。例如，标签管理器可以包含问题标签文档库可以包含文档，或者一个对象路由器可以包含路由规则，等等。典型的，容器类提供添加、删除、查询或定位所包含对象的方法。

3.5. ZClass 安全控制

当你构建新对象类型时，安全可以扮演重要的角色。例如，在动物园站点中需要以下三个角色：

Manager（管理员）

这个角色在 Zope 中默认存在。这种角色可以完全管理 Zope 系统。

Caretaker（看管者）

- 当你创建一个 ZooExhibit 实例以后，你希望让具有看管者角色的用户能够编辑展览。只有具有这个角色的用户才能够看到 ZooExhibit 实例的 Edit 视图。

Anonymous（匿名）

这个角色在 Zope 中默认存在。具有 Anonymous 角色的人们应该能够观看展览，但是不能以任何方式管理它或更改它。

就像你在“用户和安全”中学习到的，创建新角色是容易的，但是如何控制谁能够创建和编辑新的 ZooExhibit 实例？为此，你必需对 ZooExhibit ZClass 定义安全策略，用它控制访问 ZClass 的方法和属性单。

3.6. 控制访问方法和属性单

默认的情况下，Zope 可以对 ZClass 进行安全控制。可是，你可能需要以特殊的方式控制访问 ZClass 实例。

例如，动物园看管者只对看到 Edit 视图感兴趣（可能还有 Undo 视图，我们将在后边讲述），而不是 Security 或者 Ownership 视图。你不想让动物园看管者更改展览的安全设置，你甚至不想让他们看到展览的具体内容，你只想赋予他们编辑一个展览的能力，除此以外再没有其它的。

要实现这些，你需要在 ZooExhibit 产品（不是在 ZClass 里，许可只在产品中定义）中创建一个 Zope Permission 对象。为此，进入 ZooExhibit Product，然后从添加列表中选择 Zope Permission 。输入新许可的 id 为 edit_exhibit_permission，name 为 Edit Zoo Exhibits，然后单击 Generate。

现在选择你的 ZooExhibit ZClass，然后单击 Permissions 标签。如后出现 Permissions 视图，如图所示。

Permissions 视图

这个视图显示了你的 ZClass 中的已用许可以及其它可用许可。在右边是一个所有默认的 ZClass 自动继承的 Zope 许可的列表。在左边是一个多选框，在这里可以添加新许可。选中这个框中的 Edit Zoo Exhibits permission ，然后单击 Save Changes。这样就告诉 ZClass 使用这个许可以及右边的许可。

现在，单击 Property Sheets 标签，然后选择 ExhibitProperties 属性单，单击 Define Permissions 标签。

你需要告诉这个属性单只有那些拥有刚才创建的那个 Edit Zoo Exhibits 许可的用户才允许管理 ExhibitProperties 属性单上的属性。在这个视图上，下拉选择框，然后选择 Edit Zoo Exhibits。这样就把 Edit Zoo Exhibits 映射给属性单上的 Manage Properties 许可。这个你可以进行选择的许可列表来自于你刚才所在的 ZClass Permissions 视图，并且因为你在那个屏幕上选择 Edit Zoo Exhibits 许可，因此它给你显示出这个让你进行选择的列表。你会发现所有的选项默认为失效，它意味着属性单不能被其他任何人编辑。

现在，你可以返回 Exhibits 文件夹并选择 Security 视图。其中可以在左边提供的许可列表里看到新许可。现在你需要创建一个名为 Caretaker 的新角色并且把新角色映射给 Edit Zoo Exhibits 许可。

现在，用户要查看或使用任何 ZooExhibit 实例的 Edit 视图就必需拥有 Caretaker 角色。

通过相同方式，可以控制 ZClass 的 Methods 视图中的对象的访问。

3.7. 控制访问 ZClass 实例

前一节讲述了如何控制访问 ZClass 实例的方法和属性。访问控制就是控制谁可以创建 ZClass 的新实例。实例由工厂创建。工厂关联有许可。在 Zoo Exhibit 例子中，Add Zoo Exhibits 许可控制创建 Zoo Exhibit 实例的能力。

通常，只有管理员才拥有 Add Zoo Exhibits 许可，因此只有管理员才能创建新的 Zoo Exhibits。然而，就像所有的 Zope 许可一样，你可以在站点中的不同位置里调整让哪一个角色拥有这个许可。这个许可由 Edit Zoo Exhibits 许可单独控制，认识到这点很重要。这样就使得以下情况成为可能，那就是允许某些人们，比如看管者，进行修改而不能创建 Zoo Exhibits。

3.8. 为 ZClass 提供上下文相关的帮助

在 ZClass 的 View 屏幕中，你可以看到每个视图关联有一个帮助主题。这使你能够根据用户正在查看的视图，提供一个相应的帮助内容链接。例如，让我们为 ZooExhibit ZClass 的 Edit 视图创建一个帮助主题。

首先，你需要创建一个真实的帮助主题对象。首先进入 ZooExhibit 产品，其中 ZooExhibit ZClass，然后单击 Help 文件夹。图标看起来应该象带有一个蓝色问号的文件夹。

在这个特殊文件夹里，下拉添加列表并且选择 Help Topic。输入这个主题的 id 为 ExhibitEditHelp ，title 为"Help for Editing Exhibits"，然后单击 Add。

现在，Help 文件夹包含有一个新的名为 ExhibitEditHelp 帮助主题对象。你可以单击这个对象并编辑它，它就像一个 DTML 文档一样。在这个文档中，你应该输入一些帮助信息，比如：

```
<dtml-var standard_html_header>

  <h1>Help!</h1>

  <p>To edit an exhibit, click on either the <b>animal</b>,

  <b>description</b>, or <b>caretakers</b> boxes to edit

  them.</p>

<dtml-var standard_html_footer>
```

现在，你已经创建了帮助主题，你需要关联 ZClass 的 Edit 视图。为此，选中 ZooExhibit ZClass，单击 Views 标签。在右边在定义 Edit 视图的同一行里，下拉帮助选择框，选中 ExhibitEditHelp ，然后单击 Change。现在，进入你的某个 ZooExhibit 实例。Edit 视图现在就有了一个帮助！你可以单击这个链接来查看这个视图的帮助主题。

在接下来的一节中，将看到 ZClass 如何结合标准 Python 类来扩展功能。

4. 使用 Python 基础类

ZClass 给你提供了一种 Web 管理界面，用它可以在 Zope 中设计新对象类型。在本章的开始，我们给你展示了如何从一个基础类列表选择来扩展 ZClass。这些基础类中的大多数都是用 Python 编写的，在本节，你将看到如何编写定制的 Python 类并把它们包含进那个列表，这样你的 ZClass 就能扩展它们的方法。

编写 Python 基础类是很容易，但是涉及一些安装细节。要创建一个 Python 基础类，需要访问文件系统。在 lib/python/Products 目录中创建一个名为 AnimalBase 的目录，在这个目录里，创建一个名为 Animal.py 的文件，代码为：

```
class Animal:

    """

    A base class for Animals

    """

    _hungry=0

    def eat(self, food, servings=1):

        """

        Eat food

        """

        self._hungry=0

    def sleep(self):

        """

        Sleep

        """

        self._hungry=1

    def hungry(self):

        """

        Is the Animal hungry?

        """
```

```
        return self._hungry
```

这个类定义了几个方法和一个默认属性。你会发现，就像外部方法一样，这个类的方法可以访问私有属性。接下来，你需要用 Zope 注册基础类。在 AnimalBase 目录里创建一个具有以下内容的 init.py 文件：

```
from Animal import Animal

def initialize(context):

    """

    Register base class

    """

    context.registerBaseClass(Animal)
```

为了让 Zope 识别新的基础类，你需要重新启动 Zope。当 Zope 重新启动以后，你能够通过多种方式验证新注册的基础类。首先，进入控制面板中的产品文件夹，查看 AnimalBase 软件包。你应该看到一个具有关闭盒子图标的产品。如果你看到一个坏盒子，它意味着你的 AnimalBase 产品存在一些错误。

单击 Traceback 视图来查看 Python 回溯，它给你显示 Zope 试图注册基础类时发生了什么问题。当你解决了你的基础类中可能存在的问题以后，你需要再次重新启动 Zope。继续这个过程，直到 Zope 成功装载你的产品。现在你可以创建一个新的 ZClass，并且你将看到 Animal 作为一个选项出现在基础类选择字段中。

要测试这个新的基础类，创建一个 ZClass ，继承 Animal。创建一个名为 care 的 DTML 方法：

```
<dtml-var standard_html_header>

<dtml-if give_food>

  <dtml-call expr="eat('cookie')">

</dtml-if>

<dtml-if give_sleep>

  <dtml-call sleep>

</dtml-if>

<dtml-if hungry>

  <p>I am hungry</p>
```

```
<dtml-else>

  <p>I am not hungry</p>

</dtml-if>

<form>

<input type="submit" value="Feed" name="give_food">

<input type="submit" value="Sleep" name="give_sleep">

</form>

<dtml-var standard_html_footer>
```

现在，创建一个 animal 类的实例并测试它的 care 方法。care 方法让你喂动物，并且通过调用在 Python 基础类中定义的方法来让动物睡觉。另外，你会发现喂完动物以后，它不饿了，但是如果你让它睡上一会，它醒来又饿了。

你可以看到，创建产品和 ZClass 是一个相关联的过程，但是当你掌握基础以后理解起来就简单了。单独使用 ZClass ，你就可以创建一些相当复杂的适用于你的 Web 浏览器的 Web 应用程序。

在接下来的一节里，你将创建一个产品的发布版，这样一来，你就可以和其他人共享或者把它交付给顾客。

5. 分发产品

现在你已经创建了自己的产品，它使你能够在 Zope 中创建任何数量的展览。假设你在另外一个动物园有一个好友，他被你的新的在线展览系统深深打动，也想为他的动物园得到一个类似的系统。

也许你甚至属于美国动物园管理人协会，并且你想把产品发给任何对你的产品这样的展览系统感兴趣的人。Zope 使你能够让你的产品作为一个整体和容易传输的软件包来分发产品，其他的用户可以从你那里下载并安装到它们的 Zope 系统中。

要分发你的产品，单击 ZooExhibit 产品，然后选择 Distribution 标签。出现 Distribution 视图。

在这个视图上的表单使你能够控制你想创建的发布版本。Version 框用来为你的产品发布版本指定版本号。对于你制作的每个发布版本，Zope 自动增加这个数字，但是你也许想要自己指定它。保持它为默认的 1.0,除非你需要更改它。

接下来的两个圆形按钮用来选择是否想让其他人能够定制或重分发你的产品。如果你想给他们以不受限制的定制或重分发你的产品的能力，选择 Allow Redistribution 按钮。如果你想不允许他们的重分发你的产品，选择"Disallow redistribution and allow the user to configure only the selected objects:"按钮。如果你不允许重分发，你可以一个对象一个对象的选择用户在你的产品中所能够定制的内容。如果你不希望他们能够更改一些内容，那么就不要选择这个列表中的任何项。如果你想让他们能够更改 ZooExhibit ZClass，那么只选择那个 ZClass。如果你想让他们能够更改任何事情（但是仍然不能重分发你的产品），那么选择所有的列表中的对象。

现在，通过单击Create a distribution archive，就可以创建一个产品的发布版。Zope现在自动生成一个名为 ZooExhibit-1.0.tar.gz 的文件。就像其它任何产品一样，通过把它解压缩到你的Zope安装目录中的根目录中，这个产品可以安装在任何Zope里。

不要忘了，当你分发你的产品时，你还需要包含一些文件，例如你的类所依赖的外部方法文件和Python基础类。这个要求使得发布变得难一些，因此，当人们创建用于分发的Web产品时尽量避免依赖Python文件。

第二十三章 维护Zope

要想让Zope站点长期稳定的运行，需要涉及到一些维护工作。本章主要讲述：

启动时自动运行Zope

安装新的Zope产品

在控制面板中设置参数

监控

清除日志文件

打包和备份数据库

数据库恢复工具

1. 启动时自动运行Zope

一般在测试和开发时常用手工启动Zope的方式，但如果在实际运行的时候，需要在系统启动时自动运行Zope。

1.1. 调试模式和自动启动

如果你计划在Unix这样的系统中运行Zope，应该关闭调试模式。这意味着在启动脚本中去掉 -D 选项，以及不要设置Z_DEBUG_MODE变量。在调试状态，Zope不能离开终端，因此有可能会导致启动脚本生成错误。

在Windows中，以服务方式运行Zope会自动关闭调试模式。当然通过Z_DEBUG_MODE变量或直接手工加上 -D 选项还是会以调试模式运行。在实际运行时，如果使用调试模式，会影响性能。

1.2. Linux

1.2.1. 使用打包好的Zope

对于许多Linux发行版本，已经配置有打包好的Zope。比如Debian，SuSE，Mandrake和Gentoo等等。其中的Zope版本都会使用比较新的版本。都带有自动运行的启动脚本。此时不需要再配置。

1.2.2. 定制自动启动脚本

如果你不喜欢使用 默认的配置，也可以修改启动配置文件。 对于下面的例子，我们假设 Zope 安装在/usr/local/zope 目录中。另外，还假设系统中有一个用户，即 zope，一个组 nogroup。用户 zope 对$ZOPE_HOME 目录有只读权限，对 var 目录有写入权限。 如果以 root 身份启动 Zope，需要让 var 目录属于 root。即使用命令 chown root var。 作为例子，从 SuSE 启动脚本中截取部分代码如下：

```
#! /bin/sh

# Copyright (c) 1995-2000 SuSE GmbH Nuernberg, Germany.

#

# Authors: Kurt Garloff, Vladim?r Linek

#          <feedback@suse.de>

#

# init.d/zope

#

#   and symbolic its link

#

# /usr/sbin/rczope

#

# System startup script for the Zope server

#

### BEGIN INIT INFO

# Provides: zope

# Required-Start: $remote_fs

# Required-Stop:  $remote_fs

# Default-Start:  3 5
```

```
# Default-Stop:   0 1 2 6
# Description:    Start Zope server.
### END INIT INFO
# Source Zope relevant things
. /etc/sysconfig/zope
. /etc/sysconfig/apache
PYTHON_BIN="/usr/bin/python2.1"
test -x $PYTHON_BIN || exit 5
ZOPE_HOME="/opt/zope"
test -d $ZOPE_HOME || exit 5
ZOPE_SCRIPT="$ZOPE_HOME/z2.py"
test -f $ZOPE_SCRIPT || exit 5
# Shell functions sourced from /etc/rc.status:
#      rc_check         check and set local and overall rc status
#      rc_status        check and set local and overall rc status
#      rc_status -v     ditto but be verbose in local rc status
#      rc_status -v -r  ditto and clear the local rc status
#      rc_failed        set local and overall rc status to failed
#      rc_failed <num>  set local and overall rc status to <num><num>
#      rc_reset         clear local rc status (overall remains)
#      rc_exit          exit appropriate to overall rc status
. /etc/rc.status
```

```
# First reset status of this service

rc_reset

# Return values acc. to LSB for all commands but status:

# 0 - success

# 1 - generic or unspecified error

# 2 - invalid or excess argument(s)

# 3 - unimplemented feature (e.g. "reload")

# 4 - insufficient privilege

# 5 - program is not installed

# 6 - program is not configured

# 7 - program is not running

#

# Note that starting an already running service, stopping

# or restarting a not-running service as well as the restart

# with force-reload (in case signalling is not supported) are

# considered a success.

COMMON_PARAMS="-u zope -z $ZOPE_HOME -Z /var/run/zope.pid -l /var/log/zope.log"

PCGI_PARAMS="-p $ZOPE_HOME/Zope.cgi"

[ -z "$ZOPE_HTTP_PORT" ] && ZOPE_HTTP_PORT="8080"

ALONE_PARAMS="-w $ZOPE_HTTP_PORT"

# For debugging...

#SPECIAL_PARAMS="-D"
```

```
[ -z "$ZOPE_FTP_PORT" ] && ZOPE_FTP_PORT="8021"
if [ "$ZOPE_FTP" == "yes" ]; then
      SPECIAL_PARAMS="-f $ZOPE_FTP_PORT $SPECIAL_PARAMS"
fi
if [ "$ZOPE_PCGI" == "yes" ]; then
      PARAMS="$SPECIAL_PARAMS $PCGI_PARAMS $COMMON_PARAMS"
else
      PARAMS="$SPECIAL_PARAMS $ALONE_PARAMS $COMMON_PARAMS"
fi
case "$1" in
    start)
      echo -n "Starting zope"
      ## Start daemon with startproc(8). If this fails
      ## the echo return value is set appropriate.
      # NOTE: startproc return 0, even if service is
      # already running to match LSB spec.
      startproc $PYTHON_BIN $ZOPE_SCRIPT -X $PARAMS
      # Remember status and be verbose
      rc_status -v
      ;;
    stop)
      echo -n "Shutting down zope"
```

```
  ## Stop daemon with killproc(8) and if this fails
  ## set echo the echo return value.
  killproc -g -p /var/run/zope.pid -TERM $PYTHON_BIN
  # Remember status and be verbose
  rc_status -v
  ;;
try-restart)
  ## Stop the service and if this succeeds (i.e. the
  ## service was running before), start it again.
  ## Note: try-restart is not (yet) part of LSB (as of 0.7.5)
  $0 status >/dev/null &&  $0 restart
  # Remember status and be quiet
  rc_status
  ;;
restart)
  ## Stop the service and regardless of whether it was
  ## running or not, start it again.
  $0 stop
  $0 start
  # Remember status and be quiet
  rc_status
  ;;
```

```
force-reload)
  ## Signal the daemon to reload its config. Most daemons
  ## do this on signal 1 (SIGHUP).
  ## If it does not support it, restart.
  echo -n "Reload service zope"
  $0 stop  &&  $0 start
  rc_status
  ;;
reload)
  ## Like force-reload, but if daemon does not support
  ## signalling, do nothing (!)
  rc_failed 3
  rc_status -v
  ;;
status)
  echo -n "Checking for zope: "
  ## Check status with checkproc(8), if process is running
  ## checkproc will return with exit status 0.
  # Status has a slightly different for the status command:
  # 0 - service running
  # 1 - service dead, but /var/run/  pid  file exists
  # 2 - service dead, but /var/lock/ lock file exists
```

```
        # 3 - service not running
        # NOTE: checkproc returns LSB compliant status values.
        checkproc -p /var/run/zope.pid $PYTHON_BIN
        rc_status -v
        ;;
    probe)
        ## Optional: Probe for the necessity of a reload,
        ## give out the argument which is required for a reload.
        test $ZOPE_HOME/superuser -nt /var/run/zope.pid && echo reload
        ;;
    *)
        echo "Usage: $0 {start|stop|status|try-restart|restart|force-reload|reload|probe}"
        exit 1
        ;;
esac
rc_exit
```

你可以通过修改PYTHON_BIN,ZOPE_HOME和COMMON_PARAMS来定制自己的启动脚本。 如果要使用Zope中的python，以zope用户身份，并支持8081端口中运行WebDAV，可以这样设置：

```
ZOPE_HOME=/usr/local/zope
PYTHON_BIN=$ZOPE_HOME/bin/python
COMMON_PARAMS="-u zope -z $ZOPE_HOME -Z /var/run/zope.pid \
               -l /var/log/Z2.log -W 8081 "
```

另外，还可以在/etc/sysconfig/zope中添加变量 ZOPE_HTTP_PORT，ZOPE_FTP_PORT：

```
ZOPE_HTTP_PORT=80

ZOPE_FTP_PORT=21
```

如果不设置，默认为8080和8021。

各个Linux中的设置会有些不一样。我们只能试图编写比较通用的启动脚本。 Linux中的启动脚本通常位于/etc/init.d或/etc/rc.d/init.d。我们假设位于/etc/rc.d/init.d。 要让启动进程调用启动脚本，还需要在/etc/rc.d/rd?.d目录中放置一个启动脚本的符号链接，其中的?表示运行等级，通常是3或5（3表示完整多用户模式，5表示启动X的多用户模式），因此需要放两个。注意一些系统中认为等级2才是完整多用户模式，比如Debian。我们还假设主启动脚本位于/etc/rc.d/init.d/zope。

```
# cd /etc/rc.d/rc3.d

# ln -s /etc/rc.d/init.d/zope S99zope

# cd /etc/rc.d/rc5.d

# ln -s /etc/rc.d/init.d/zope S99zope
```

这段脚本会在启动和关机时自动调用。 一般，通用的启动脚本结构如下：

```
#!/bin/sh

# set paths and startup options

ZOPE_HOME=/usr/local/zope

PYTHON_BIN=$ZOPE_HOME/bin/python

ZOPE_OPTS=" -u zope -P 8000"

EVENT_LOG_FILE=$ZOPE_HOME/var/event.log

EVENT_LOG_SEVERITY=-300

# define more environment variables ...

export EVENT_LOG_FILE  EVENT_LOG_SEVERITY

# export more environment variables ...

umask 077

cd $ZOPE_HOME
```

```
case "$1" in
start)
    # start service
    exec $PYTHON_BIN $ZOPE_HOME/z2.py $ZOPE_OPTS
    # if you want to start in debug mode (not recommended for
    # production systems):
    # exec $PYTHON_BIN $ZOPE_HOME/z2.py $ZOPE_OPTS -D &
    ;;
stop)
    # stop service
    kill `cat $ZOPE_HOME/var/Z2.pid`
    ;;
restart)
    # stop service and restart
    $0 stop
    $0 start
    ;;
*)
  echo "Usage: $0 {start|stop|restart}"
  exit 1
  ;;
esac
```

这段脚本执行 start / stop / restart 操作：

start

启动 Zope

stop

终止 Zope

restart

先终止再启动 Zope

1.3. MS Windows

自动启动 Zope 最普遍的方式是以服务方式安装 Zope。如果是 Windows 95/98/Me，则需要在启动文件夹中放入一个 start.bat 的快捷方式。 如果在 Windows NT/2000/XP 中安装时不是以服务方式安装，而是后来想改成服务方式，可以执行以下命令，进行注册：

```
> cd c:\Program Files\zope

> bin\lib\win32\PythonService.exe /register

> bin\python.exe ZServer\ZService.py --startup auto install
```

其中，你可以根据自己的情况把 c:\Program Files\zope 改成自己的 Zope 安装路径。通过控制面板中的“服务”可以控制启动和关闭。

2. 安装新产品

要安装 Zope 产品，先从 Zope.org 中的产品部分下载文件。然后把文件解压缩。然后把文件复制或剪切到 lib/python/Products 目录。完成以后，重新启动 Zope，即可。

3. 服务器设置

Zope 服务器可以通过一些设置调整性能。

3.1. 数据库缓存

最重要的是数据库缓存设置。进入 Zope 控制面板，点击数据库管理连接。通常可以有 7 个数据库连接。每个连接都进行缓存。缓存数量在"Target number of objects in memory per cache"设置。如果为 400，那么内存中最多将缓存 2800 个对象。对于 Zope 外边的数据，比如文件系统或关系型数据库中的数据不进行缓存。

3.2. 解释器检测间隔

解释器检测间隔决定了解释器检测系统内部数据的频繁程序，比如信号处理和线程切换等等。数值越大，检测次数就越少。默认值 500 适用于大多数系统。系统性能越好，数值应该越大。

设置方式是通过 z2.py 脚本使用 -i 参数。

3.3. ZServer 线程数

这个数字决定了 Zope 在响应服务时使用多少线程。默认为 4。如果站点访问数量巨大，可以增大这个数字。如果超过了 7，应当降低数据库连接数。

3.4. 数据库连接数

每个数据库连接有自己的缓存。因此增大数据库连接数将增大内存的需求。

调整这个数的方式是在 zope 安装目录中添加一个 custom_zodb.py 文件，然后输入以下代码：

```
import ZODB.FileStorage

import ZODB.DB

filename = os.path.join(INSTANCE_HOME, 'var', 'Data.fs')

Storage = ZODB.FileStorage.FileStorage(filename)

DB = ZODB.DB(Storage, pool_size=25, cache_size=2000)
```

其中"pool_size"参数是数据库连接数。注意，数据库连接数应该略大于 ZServer 线程数。

4. 监控

4.1. 事件日志和访问日志

如果你指定了 EVENT_LOG_FILE 环境变量，那么就可以查看事件日志。通过对 EVENT_LOG_SEVERITY 指定不同的值可以指定不同的内容：

```
TRACE=-300   -- Trace messages

DEBUG=-200   -- Debugging messages

BLATHER=-100 -- Somebody shut this app up.

INFO=0       -- For things like startup and shutdown.
```

```
PROBLEM=100  -- This isn't causing any immediate problems, but

deserves attention.

WARNING=100  -- A wishy-washy alias for PROBLEM.

ERROR=200    -- This is going to have adverse effects.

PANIC=300    -- We're dead!
```

另外 var 目录中 z2.log 是访问日志文件。

4.2. 监控 HTTP 服务

可以通过一些工具软件进行监控，比如 Nagios 或 VisualPulse 。也可以编写脚本程序。

比如，我们可以在服务器中创建一个 DTML 方法，只返回一个字符“1”。然后使用下面的代码进行监控，如果发现错误，则发送邮件告诉错误。

```
#!/bin/sh

# configure the values below

URL="http://localhost/ping"

EXPECTED_ANSWER="1"

MAILTO="your.mailaddress@domain.name"

SUBJECT="There seems to be a problem with your website"

MAIL_BIN="/bin/mail"

resp=`wget -O - -q -t 1 -T 1 $URL`

if [ "$resp" != "$EXPECTED_ANSWER" ]; then

    $MAIL_BIN -s "$SUBJECT" $MAILTO <<EOF

The URL

--------------------------------------------

$URL
```

```
-------------------------------------------

did not respond with the expected value of $EXPECTED_ANSWER.

EOF

fi;
```

通过 cron 每 10 分钟运行一次这个脚本就可以完成简单的监控工作。

5. 日志文件

5.1. 访问日志（Access Log）

默认的访问日志文件是$ZOPE_HOME/var/Z2.log。

5.2. 事件日志（Event Log）

通过 EVENT_LOG_FILE 进行指定，比如：EVENT_LOG_FILE=$ZOPE_HOME/var/event.log。

5.3. 日志轮换

日志文件会不断增长，因此需要定时轮换，即关闭旧文件，重命名，或是进行压缩处理，然后创建新文件。在 Unix 中可使用 logrotate 模块，比如可以这样配置：

```
compress

/usr/local/zope/var/Z2.log {

    rotate 25

    weekly

    postrotate

            /sbin/kill -USR2 `cat /usr/local/zope/var/Z2.pid`

    endscript

}
```

Windows 缺乏这样的工具软件。

6. 打包和备份数据库

Zope 中的数据会不断增多，需要及时的进行一些清理。方法是进入控制面板，然后点击"Database Management"链接，在此处点击 Pack 按钮即可。 如果你想自动处理，可以编写一个 Python 文件：

```
#!/usr/bin/python

import sys, urllib

host = sys.argv[1]

days = sys.argv[2]

url = "%s/Control_Panel/Database/manage_pack?days:float=%s" % \

      (host, days)

try:

    f = urllib.urlopen(url).read()

except IOError:

    print "Cannot open URL %s, aborting" % url

print "Successfully packed ZODB on host %s" % host
```

这个文件用到两个参数，一个是服务器的 URL，比如： http://mymachine.com，一个是要删除旧数据的天数。 假设文件位于/usr/local/sbin/zope_pack，在 Unix 中变成可执行权限，然后在 crontab 中加入以下一行：

```
5 4 * * sun     /usr/local/sbin/zope_pack http://localhost 7
```

在 Windows 中，可以使用任务规划工具定时运行这个程序。比如：

```
"C:\Program Files\zope\bin\python.exe" "C:\Program Files\zope_pack.py" "http://localhost" 7
```

备份数据库文件，只要处理$ZOPE_HOME/var/Data.fs 文件就可以了。在 Linux 中，不宜直接使用 tar 工具，这是因为如果在 tar 的时候，文件发生的变化，tar 会退出。也可以使用以下脚本：

```
#!/bin/sh

########################################

# File: /etc/cron.daily/zbackup.cron

#
```

```
# Backup Zope Database Daily

########################################

#

# rsync arguments:

# -q       ::= Quiet operation, for cron logs

# -u       ::= Update only, don't overwrite newer files

# -t       ::= Preserve file timestamps

# -p       ::= Preserve file permissions

# -o       ::= Preserve file owner

# -g       ::= Preserve file group

# -z       ::= Compress during transfer

# -e ssh   ::= Use the ssh utility to secure the link

#

ARCHTOP="/archive/zope/"

DOW=`date +%A`

ARCHDIR="${ARCHTOP}${DOW}"

#

# Insure Our Day-of-Week Directory Exists

[ -d ${ARCHDIR} ] || mkdir ${ARCHDIR} || {

    echo "Could Not Create Day-of-Week Directory: ${ARCHDIR}" ; exit 1

}

#
```

```
/usr/bin/rsync -q -u -t -p -o -g /var/zope/var/Data.fs ${ARCHDIR}

#

ln -sf ${ARCHDIR} ${ARCHTOP}Current

#

exit 0
```

这个脚本通过使用 cron，每天自动运行。

6.1. 数据库恢复工具

要恢复数据库，可以使用$ZOPE_HOME/lib/python/ZODB 中的程序 fsrecover，它的输出信息为：

```
python fsrecover.py [ <options> ] inputfile outputfile

Options:

-f -- force output even if output file exists

-v level -- Set the

verbosity level:

     0 -- Show progress indicator (default)

     1 -- Show transaction times and sizes

     2 -- Show transaction times and sizes, and

          show object (record) ids, versions, and sizes.

-p -- Copy partial transactions. If a data record in the middle of a

   transaction is bad, the data up to the bad data are packed. The

   output record is marked as packed. If this option is not used,

   transaction with any bad data are skipped.

-P t -- Pack data to t seconds in the past. Note that is the "-p"
```

option is used, then t should be 0.

第二十四章 DTML 参考

文档模板标记语言（DTML）是一种便捷的内置于 Zope 中的模板和表现语言。这个文档涉及所有的 DTML 标记符和它们的使用方法。

1.*call*:调用一个方法

call 标记符使你能够调用一个方法而不把结果插入到 DTML。

句法

call 标记符句法：

```
<dtml-call Variable|expr="Expression">
```

如果 call 标记符使用一个变量，DTML 自动传递方法的参数——就像 var 标记符那样。如果在一个表达式里指定方法，那么你必须亲自传递参数。

例子

用变量名称调用：

```
<dtml-call UpdateInfo>
```

它调用 UpdateInfo 对象，自动传递参数。

用表达式调用：

```
<dtml-call expr="RESPONSE.setHeader('content-type', 'text/plain')">
```

参见

var tag

2. *comment*: 注释 DTML

comment 标记符使你能够用注释为 DTML 做注解。你还可以用它把 DTML 注释掉，让 DTML 标记符暂时失效。

句法

comment 标记符句法：

```
<dtml-comment>
</dtml-comment>
```

comment 标记符是一种块标记符。块的内容不被执行，并且不被插入到 DTML 输出中。

例子

注解 DTML：

```
<dtml-comment>
  This content is not executed and does not appear in the
  output.
</dtml-comment>
```

注释掉 DTML：

```
<dtml-comment>
  This DTML is disabled and will not be executed.
  <dtml-call someMethod>
</dtml-comment>
```

3. *Functions*：DTML 函数

DTML 效用函数提供一些 Python 内建函数和一些 DTML 特效函数。

函数

abs(number)——返回数字 number 的绝对值。参数可以为一个普通整数或长整数或一个浮点数。如果参数是一个复数，则返回它的量值。

chr(integer)——返回一个字符，这个字符的 ASCII 代码是参数 integer。例如，chr(97)返回字符 a。相反的函数为 ord()。参数必需在 0 至 255 范围内，如果 integer 超出了那个范围就引发一个 ValueError 错误。

DateTime()——返回一个已知构造参数的 Zope DateTime 对象。有关构造参数的更多信息请参见附录 B“API 参考”中的“DateTime 类”部分。

divmod(number, number)——采用两个数字作为参数并在使用长除法时返回一对由它们的商和余数组成的数字。对于混合操作数类型，应用二进制算法操作符规则。对于普通和长整数，结果等同于(a / b, a % b)。对于浮点数，结果是 (q, a % b)，其中 q 是 math.floor(a / b)；然而，它可能会是 1，小于那个数。在任何情况中，q * b + a % b 非常接近 a。如果 a % b 是非 0，它有和 b 相同的符号并且 0 <= abs(a % b) < abs(b)。

float(number)——把一个字符串或一个数字转换成浮点数。如果参数是一个字符串，它必需包含一个可能带有符号的小数或者用空格嵌入的浮点数；它的作用等同于 string.atof(number)。另外，参数可以是一个普通整数或长整数或者一个浮点数，如果是浮点数，返回相同的值（在 Python 的浮点精度以内）。

getattr(object, string)——返回对象的被指定的属性的值。名称必需为一个字符串。如果 string 是对象中的某个属性的名称，结果就是那个属性的值。例如，getattr(x, "foobar") 等同于 x.foobar。如果被指定的属性不存在，提供了默认值就返回默认值，否则就引发一个 AttributeError 错误。

getitem(variable, render=0)——返回一个 DTML 变量的值。如果 render 为真，变量就被呈递。见 render 函数。

hasattr(object, string)——参数是一个对象和一个字符串。如果这个字符串是对象的某个属性的名称，结果为 1，否则结果为 0。（这是通过调用 getattr(object, name) 和查看它是否引发一个例外来实现的。）

hash(object)——返回对象的散列值（如果它有的话）。散列值是整数。它们用于在一个字典查找期间内快速比较字典键。相等的数字值有相同的散列值（即使它们属于不同的类型，例如 1 和 1.0）。

has_key(variable)——如果 DTML 名称空间包含 variable 则返回真。

hex(integer)——把一个整数（任意大小）转换成十六进制的字符。结果是一个有效的 Python 表达式。注意：这常常产生一个无符号的文字。例如在 32-bit 机器上，hex(-1) 结果为 0xffffffff。当在一个机器上使用相同的单词大小求值时，这个文字认为是-1；对于不同的单词大小，它可能会变成大的正数或引发一个 OverflowError 例外。

int(number)——把一个字符串或数字转换成一个普通整数。如果参数是一个字符串，它必须包含一个可能带有符号的十进制数字，这个数字可以作为一个 Python 整数提供，并且通过空格嵌入其中。这个行为等同于 'string.atoi(number[, radix])' 。radix 参数给出了转换的基数并且可以是 2 到 36 范围内的任何整数。如果指定了 radix 并且 number 不是一个字符串，引发一个 TypeError 错误。另外，参数可以是一个普通整数或长整数或浮点数。浮点数转换成整数按照 C 语言的方法来定义；通常，转换是把小数点后面数字趋向为零。

len(sequence)——返回对象的长度（数据项的数量）。参数可以为一个序列或一个映射（字典）。

max(s)——s 参数（例如，一个字符串，元组或列表）唯一时返回非空序列中的最大项。当有多个参数，它返回参数中的最大一个。

min(s)——s 参数（例如，一个字符串，元组或列表）唯一时返回非空序列中的最小项。当有多个参数时，它返回参数中的最大一个。

namespace([name=value]...)——返回一个新的 DTML 名称空间对象。关键字参数 name=value 对被加入到新的名称空间中。

oct(integer)——把一个整数（任意大小）转换成八进制字符。结果是一个有效的 Python 表达式。注意：这总是生成一个无正负符号的文字。例如，在一个 32-bit 机器上， oct(-1) 生成 037777777777。当在一个具有相同单词大小的机器上求值时，这个文字结果为-1；对于一个不同的单词大小，它可能生成一个大整数或引发一个 OverflowError 例外。

ord(character)——返回某个字符的 ASCII 值。例如，ord("a") 返回整数 97 。相反的函数为 chr() 。

pow(x, y [,z])——返回 x 的 y 次方。如果提供了 z，它首先求 x 的 y 方，再对 z 求余，返回结果（执行效率高于 'pow(x, y) % z'）。参数必须为数字类型。对于混合操作数类型，应用二进制算法操作符规则。有效的操作数类型也是结果的类型，如果结果不表现为这种类型，函数引发一个例外，例如 pow(2, -1) 或 pow(2, 35000)是不允许的。

range([start,] stop [,step])——这是一个用于创建含有算术级数的列表的通用函数。参数必需为普通整数。如果 step 参数被忽略，默认为 1。如果 start 参数被忽略，默认为 0。返回的完整形式是一个普通整数列表 '[sstart, start + step, start + 2 * step, ...]'。如果 step 是正数，最后一个元素是小于 stop 的最大'start + i step'；如果 step 是一个负数，最后一个元素是大于 stop 的最大'start + i step'。step 不得为 0（否则引发一个 ValueError 错误）。

round(x [,n])——返回 x 被四舍五入后的浮点值，它截至小数点后 n 位。如果 n 被忽略，它默认为零。结果是一个浮点数。值被四舍五入到最接近的 10 的负 n 次幂处的倍数；如果两个倍数一样近，四舍五入选择远离 0 的那个（例如，round (0.5)为 1.0 ，round (-0.5) 为-1.0） 。

render(object)——呈递对象。对于 DTML 对象，它使用当前的名称空间对 DTML 求值。对于其他对象，它等同于 str(object) 。

reorder(s [,with] [,without])——对 s 中的数据项根据在 with 中给定的顺序重新排序，不包括在 without 中提到的数据项。s 中没有在 with 中提到的数据项被删除。s、with 和 without 都是字符序列或键值元组序列，对键进行排序。这个函数对于构建有序的选择列表是有用的。

SecurityCalledByExecutable()——如果当前的对象（例如 DTML 文档或方法）由一个可执行对象（例如其它的 DTML 文档或方法，脚本或 SQL 方法）调用则返回真。

SecurityCheckPermission(permission, object)——检查安全关联是否允许给定对象上的给定许可。例如'ecurityCheckPermission('Add Documents, Images, and Files", this())'，如果当前的用户被授权能够在当前位置中创建文档、图像和文件，这个函数将返回真。

SecurityGetUser()——返回当前用户对象。通常，它等同于 REQUEST.AUTHENTICATED_USER 对象。然而，AUTHENTICATED_USER 对象是不安全的，这是因为它可以被替换。

SecurityValidate([object] [,parent] [,name] [,value])——如果当前用户可以访问 value 则返回真。object 是被访问的变量所在的对象，parent 是变量的容器，name 是用于访问变量的名称（例如，如果它通过'getattr'获得）。你可以忽略其中的一些参数。但是，最好提供所有的参数。

SecurityValidateValue(object)——如果对于当前用户是可访问的则返回真。这个函数等同于调用 SecurityValidate(None, None, None, object) 。

str(object)——返回一个包含一个适当的可打印出的表示对象的字符串。对于字符串，它返回字符串本身。

test(condition, result [,condition, result]... [,default])——对一对或多对 condition, result 进行测试并返回第一个条件为真的的结果。只返回一个结果，即使有多个条件为真。如果没有条件为真并且给定默认值，那么就返回默认值。如果没有条件为真并且没有默认值，返回 None 。

属性

None——None 对象等同于 Python 的内建对象 None。它通常用于表示一个空或假值。

参见

string module

random module

math module

Python 内建函数 (http://www.python.org/doc/current/lib/built-in-funcs.html)

4. *if*：测试条件

根据条件，if 标记符使你能够测试条件并且采取不同的行为。if 标记符反映 Python 的 if/elif/else 条件测试语句。

句法

if 标记符句法：

```
<dtml-if ConditionVariable|expr="ConditionExpression">

[<dtml-elif ConditionVariable|expr="ConditionExpression">]

 ...

[<dtml-else>]

</dtml-if>
```

if 标记符是一种块标记符。if 标记符和可选择的 elif 标记符使用一个条件变量名称或一个条件表达式，但只能是其中的一种。如果条件名称或表达式的值为真，那么 if 块被执行。真意味着不是 0、空字符串或一个空列表。如果条件变量没有被找到，那么这个条件被认为是假。

如果初始的条件为假，则按次序测试 elif 条件。如果 elif 条件为真，那么块中的内容就被执行。最终如果 if 和 elif 条件没有为真的，就执行可选的 else 块。则只有一个块被执行。

例子

测试一个变量：

```
<dtml-if snake>

  The snake variable is true
```

```
</dtml-if>
```

测试表达式条件：

```
<dtml-if expr="num > 5">
  num is greater than five
<dtml-elif expr="num < 5">
  num is less than five
<dtml-else>
  num must be five
</dtml-if>
```

参见

Python 指南: if 语句

(http://www.python.org/doc/current/tut/node6.html#SECTION006100000000000000000)

5. *in*：对序列进行循环

in 标记符可以让你有力的控制循环序列和执行批处理。

句法

in 标记符句法：

```
<dtml-in SequenceVariable|expr="SequenceExpression”>
[<dtml-else>]
</dtml-in>
```

in 块针对序列变量或序列表达式中的每一项重复一次。当前项在每次执行 in 块时被推入到 DTML 名称空间中。

如果在序列中没有数据项是变量或表达式，则执行可选的 else 块。

属性

mapping——叠代映射对象而不是实例。这使得映射对象的值可以作为 DTML 变量被访问。

reverse——翻转序列。

sort=string——按照给定的属性名称对序列排序。

start=int——要被显示的第一项的数字，其中数据项计数从 1 开始。

end=int——要被显示的最后一项的数字，其中数据项计数从 1 开始。

size=int——批处理的大小。

skip_unauthorized——如果遇到一个没有授权的数据项不引发一个例外。

orphan=int——预期最小批处理大小。

overlap=int——批处理块之间相互重叠的数据项的数字。默认为 3。

previous——如果有前一个批处理块则叠代一次。为前一个序列设置批处理变量。

next——如果有下一个批处理块则叠代一次。为下一个序列设置批处理变量。

标记符变量

当前数据项变量。

这些变量描述当前的数据项：

sequence-item——当前的数据项。

sequence-key——当前键字。当循环表单元组（key，value）时，in 标记符把它们转换成 (sequence-key, sequence-item) 。

sequence-index——当前数据项从 0 开始的索引。

sequence-number——当前数据项从 1 开始的索引。

sequence-roman——当前数据项以小写罗马数字表示的索引。

sequence-Roman——当前数据项以大写罗马数字表示的索引。

sequence-letter——当前数据项以小写字母表示的索引。

sequence-Letter——当前数据项以大写字母表示的索引。

sequence-start——如果当前数据项为第一个数据项则为真。

sequence-end——如果当前数据项为最后一个数据项则为真。

sequence-even——如果当前数据项索引为偶数则为真。

sequence-odd——如果当前数据项索引为奇数则为真。

sequence-length——序列的长度。

sequence-var-variable——在当前数据项中的一个变量。例如，sequence-var-title 是当前数据项的标题变量。通常，你可以直接访问这个变量，这是因为当前的数据项被推到 DTML 名称空间中。而且，当显示前一个和下一个批处理信息时这些变量就可以被用上。

sequence-index-variable——当前数据项的变量索引。

总结变量

这些变量总结了有关数据项变量数字方面的信息。要使用这些变量，你必须循环含有数字变量的对象（比如数据库查询结果）。

total-variable——一个数据项变量的所有实例的总数。

count-variable——一个数据项变量的实例数。

min-variable——一个数据项变量的最小值。

max-variable——一个数据项变量的最大值。

mean-variable——一个数据项变量的平均值。

variance-variable——一个数据项的 count-1 自由度方差。

variance-n-variable——一个数据项变量的 n 自由度方差。

standard-deviation-variable——一个数据项变量的 count-1 自由度标准偏差

standard-deviation-n-variable——一个数据项变量的 n 自由度标准偏差。

分组变量

这些变量使你能够跟踪当前数据项变量的变化：

first-variable——如果当前数据项是第一个具有特定变量值的数据项则为真。

last-variable——如果当前数据项是最后一个具有特定变量值的数据项则为真。

批处理变量

sequence-query——不包含 start 变量的查询字符。你可以使用这个变量来构建与下一个和前一个批处理块的链接。

sequence-step-size——批处理块的大小。

previous-sequence——如果当前批处理块不是第一个则为真。注意，这个变量只对第一次循环迭代为真。

previous-sequence-start-index——前一个批处理块的开始索引。

previous-sequence-start-number——前一个批处理块的开始数字。注意，它等同于 previous-sequence-start-index + 1。

previous-sequence-end-index——前一个批处理块的结束索引。

previous-sequence-end-number——前一个批处理块的结束数字。注意，它等同于 previous-sequence-end-index + 1。

previous-sequence-size——前一个批处理块的大小。

previous-batches——带有前一个批处理块所有相关信息的映射对象序列。每个映射对象有三个键：batch-start-index、 batch-end-index 和 batch-size 。

next-sequence——如果当前批处理块不是最后一个批处理块则为真。注意，这个变量只对最后的循环迭代才为真。

next-sequence-start-index——下一个序列的开始索引。

next-sequence-start-number——下一个序列的开始数字。注意，这等同于 next-sequence-start-index + 1。

next-sequence-end-index——下一个序列的结束索引。

next-sequence-end-number——下一个序列的结束数字。注意，这等同于 next-sequence-end-index + 1。

next-sequence-size——下一个索引的大小。

next-batches——带有下一个批处理块所有相关信息的映射对象序列。每个映射对象有三个键：batch-start-index, batch-end-index, and batch-size。

例子

循环下级对象：

```
<dtml-in objectValues>
  title: <dtml-var title><br>
</dtml-in>
```

循环元组列表（key，value）：

```
<dtml-in objectItems>
  id: <dtml-var sequence-key>, title: <dtml-var title><br>
</dtml-in>
```

创建具有交替颜色的表格单元：

```
<table>
<dtml-in objectValues>
<tr <dtml-if sequence-odd>bgcolor="#EEEEEE"
    <dtml-else>bgcolor="#FFFFFF"
```

```
    </dtml-if>
  <td><dtml-var title></td>
</tr>
</dtml-in>
</table>
```

基本批处理：

```
<p>
<dtml-in largeSequence size=10 start=start previous>
  <a href="<dtml-var absolute_url><dtml-var sequence-query>
start=<dtml-var previous-sequence-start-number>">Previous</a>
</dtml-in>

<dtml-in largeSequence size=10 start=start next>
  <a href="<dtml-var absolute_url><dtml-var sequence-query>
start=<dtml-var next-sequence-start-number>">Next</a>
</dtml-in>
</p>

<p>
<dtml-in largeSequence size=10 start=start>
```

```
  <dtml-var sequence-item>

</dtml-in>

</p>
```

这个例子创建前一个和下一个链接，从而在批处理块之间跳转。注意，通过使用 sequence-query，当你在批处理块之间跳转时，你不会丢失任何任何 GET 变量。

6. *let*：定义 DTML 变量

let 标记符定义 DTML 名称空间里的变量。

句法

let 标记符句法：

```
<dtml-let [Name=Variable][Name=”Expression”]...>

</dtml-let>
```

let 标记符是一种块标记符。变量通过标记符参数被定义。在 let 块被执行的同时，定义的变量被推进到 DTML 名称空间。变量通过属性定义。let 标记符可以有一个或多个任意命名的属性。如果属性用双引号定义，它们被认为是表达式。否则，它们按照名称查找相应变量。属性按顺序处理，因此后边的属性可以引用和覆盖前边的属性。

例子

基本用法：

```
<dtml-let name="'Bob'" ids=objectIds>
  name: <dtml-var name>
  ids: <dtml-var ids>
</dtml-let>
```

和 in 标记符一起使用 let 标记符：

```
<dtml-in expr="(1,2,3,4)">
  <dtml-let num=sequence-item
            index=sequence-index
            result="num*index">
    <dtml-var num> * <dtml-var index> = <dtml-var result>
  </dtml-let>
</dtml-in>
```

这会生成：

1 * 0 = 0

2 * 1 = 2

3 * 2 = 6

4 * 3 = 12

参见

with 标记符

7. *math*：DTML 数学函数

数学函数模块提供三角和其它数学函数。它是标准的 Python 模块。

函数

acos(x)——返回 x 的 arc cosine 值。

asin(x)——返回 x 的 arc sine 值。

atan(x)——返回 x 的 arc tangent 值。

atan2(x, y)——返回 atan(y / x) 值。

ceil(x)——以实数形式返回 x 的上限。

cos(x)——返回 x 的 cosine 值。

cosh(x)——返回 x 的双曲线 cosine 值。

exp(x)——返回 e**x 。

fabs(x)——返回实数 x 的绝对值。

floor(x)——以实数形式返回 x 的基数。

fmod(x, y)——返回 fmod(x, y) ，它由平台的 C 库定义。注意，Python 表达式 x % y 不返回相同的结果。

fexp(x)——以 (m, e)对的形式返回 x 的尾数和指数。m 是一个浮点数，e 是一个整数，比如'x == m

* 2*e’ 。如果 x 为零，它返回 (0.0, 0)，其他情况下返回 0.5 <= abs(m) < 1 。

hypot(x, y)——返回 欧几里得几何学的距离，sqrt(x*x + y*y) 。

ldexp(x, y)——返回 x * (2**i) 。

log(x)——返回 x 的自然对数值。

log10(x)——返回以 10 为底的 x 的对数。

modf(x)——返回 x 的分数和整数部分。两个结果都执行 x 的 sine 运算。整数部分以实数形式返回。

pow(x, y)——返回 x 的 y 次幂。

sin(x)——返回 x 的 sine 值。

sinh(x)——返回 x 的双曲线 sine 值。

sqrt(x)——返回 x 的平方根。

tan(x)——返回 x 的 tangent 值。

tanh(x)——返回 x 的双曲线 tangent 值 。

属性

e——数学常量 e

pi——数学常量 pi

参见

Python 数学模块 (http://www.python.org/doc/current/lib/module-math.html)

8. *mime*:用 MIME 格式数据

mime 标记符使你能够创建 MIME 编码数据。它主要用于在 sendmail 标记符内格式电子邮件。

句法

mime 标记符句法：

```
<dtml-mime>
[<dtml-boundry>]
...
</dtml-mime>
```

mime 标记符是一种块标记符。块可以用一个或多个 boundry 标记符分隔来创建多部分 MIME 消息。mime 标记符可以签套。mime 标记符主要用在 sendmail 标记符里。

属性

所有的 mime 和 boundry 标记符都有相同的属性：

encode=string——MIME 内容传输编码报头，默认为 base64。有效的编码选项包括 base64、quoted-printable、 uuencode、 x-uuencode、 uue、 x-uue 和 7bit。在块上不编码，数据被假定为一种有效的 MIME 格式。

type=string——MIME 内容-类型报头。

type_expr=string——作为变量表达式的 MIME 内容-类型报头。你不能同时使用 type 和 type_expr 。

name=string——MIME 内容-类型报头名称。

name_expr=string——作为变量表达式的 MIME 内容-类型报头名称。你不能同时使用 name 和 name_expr。

disposition=string——MIME 内容-部署报头。

disposition_expr=string——作为变量表达式的 MIME 内容-部署报头。你不能同时使用 disposition 和 disposition_expr。

filename=string——MIME 内容-部署报头文件名。

filename_expr=string——作为变量表达式的 MIME 内容-部署报头文件名。你不能同时使用 filename 和 filename_expr 。

skip_expr=string——一个变量表达式，如果为真就跳过块。你可以使用这个属性来有选择性的包含 MIME 块。

例子

发送一个文件附件：

```
<dtml-sendmail>

To: <dtml-recipient>

Subject: Resume

<dtml-mime type="text/plain" encode="7bit">
```

```
Hi, please take a look at my resume.


<dtml-boundary type="application/octet-stream" disposition="attachment"
encode="base64" filename_expr="resume_file.getId()">
<dtml-var expr="resume_file.read()"></dtml-mime>
</dtml-sendmail>
```

参见

Python Library: mimetools (http://www.python.org/doc/current/lib/module-mimetools.html)

9. *raise*：引发一个例外

raise 标记符引发一个例外，反映 Python 的 raise 语句。

句法

raise 标记符语句：

```
<dtml-raise ExceptionName|ExceptionExpression>
</dtml-raise>
```

raise 标记符是一种块标记符。它引发一个例外。例外可以是一个 exception 类或一个字符串。标记符的内容作为错误的值。

例子

引发一个 KeyError：

```
<dtml-raise KeyError></dtml-raise>
```

引发 HTTP 404 错误:

```
<dtml-raise NotFound>Web Page Not Found</dtml-raise>
```

参见

try 标记符

Python 指南: Errors and Exceptions

(http://www.python.org/doc/current/tut/node10.html)

Python 内建例外 (http://www.python.org/doc/current/lib/module-exceptions.html)

10. *random*: DTML 伪随机数函数

random 模块提供了伪随机数函数。用它，你可以生成随机数字以及从序列中选择随机元素。这个模块是一个标准 Python 模块。

函数

choice(seq)——从非空序列 seg 中选择一个随机元素，并返回它。

randint(a, b)——返回一个随机整数 N，并且 a<=N<=b 。

random()——返回下一个[0.0…1.0]范围内的随机浮点数。

seed(x, y, z)——从整数 x，y 和 z 初始随机数字生成器。当这个模块首次被导入，随机数字使用从当前时间获得的值初始化。

uniform(a, b)——返回一个随机实数 N，并且 a<=N<b 。

参见

Python 随机模块 (http://www.python.org/doc/current/lib/module-whrandom.html)

11. *return*: 返回数据

return 标记符停止执行 DTML 并返回数据。它反映 Python 的 return 语句。

句法

return 标记符句法：

```
<dtml-return ReturnVariable|expr="ReturnExpression">
```

它停止执行 DTML 并返回一个变量或表达式。DTML 输出不被返回。通常，一个返回表达式比起返回一个变量更为有用。脚本

使得这个标记符在很大程度上荒废了。

例子

返回一个变量：

```
<dtml-return result>
```

返回一个 Python 字典：

```
<dtml-return expr="{'hi':200, 'lo':5}">
```

12. *sendmail*: 通过 SMTP 发送邮件

sendmail 标记符使用 SMTP 发送一个电子邮件信息。

句法

sendmail 标记符句法：

```
<dtml-sendmail>
</dtml-sendmail>
```

sendmail 标记符是一种块标记符。它需要一个邮件主机或一个 smtphost 参数，但是不是两者都需要。标记符块作为一条邮件消息被发送。块的开始描述电子邮件报头。报头和正文之间用一个空行分开。To、 From 和 Subject 报头可以通过标记符参数设置。

属性

mailhost——用于发送电子邮件的 Zope 邮件主机对象的名称。你不能同时指定 mailhost 和 smtphost 。

smtphost——用于发送电子邮件的 SMTP 服务器的名称。你不同同时指定 mailhost 和 smtphost 。

port——如果使用 smtphost 属性，那么 port 属性用于指定连接的端口号。如果没有指定，那么就使用端口 25 。

mailto——接收地址或一个用逗号分开的接收地址列表。这也可以用 To 报头指定。

mailfrom——发送者地址。还可以用 From 报头指定。

subject——电子邮件主题。还可以用 Subject 报头指定。

例子

使用一个 mailhost 发送一个电子邮件：

```
<dtml-sendmail mailhost="mailhost">

To: <dtml-var recipient>

From: <dtml-var sender>

Subject: <dtml-var subject>
```

```
Dear <dtml-var recipient>,


You order number <dtml-var order_number> is ready.

Please pick it up at your soonest convenience.

</dtml-sendmail>
```

参见

RFC 821 (SMTP Protocol) (http://www.ietf.org/rfc/rfc0821.txt)

mime 标记符

13. *sqlgroup*：格式复杂的 SQL 表达式

sqlgroup 标记符格式复杂的布尔 SQL 表达式。你可以用它配合 sqltest 标记符来构建动态 SQL 查询，使之满足需要。这个标记符被用在 SQL 方法中。

句法

sqlgroup 标记符句法：

```
<dtml-sqlgroup>

[<dtml-or>]

[<dtml-and>]
```

```
...
</dtml-sqlgroup>
```

sqlgroup 标记符是一种块标记符。它用一个或多个可选的 or 和 and 标记符划分成块。sqlgroup 标记符可以签套，从而产生复杂的逻辑。

属性

required=boolean——指出这组 SQL 语句是否为必需的。如果它不是必需的并且为空，它就不在 DTML 输出中显示。

where=boolean——如果为真，则包含字符串 “Where”。这用于一个 SQL select 查询中的最外面的 sqlgroup 标记符。

例子

例子用法：

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqltest salary op=gt type=float optional>
<dtml-and>
  <dtml-sqltest first op=eq type=string multiple optional>
<dtml-and>
  <dtml-sqltest last  op=eq type=string multiple optional>
```

```
</dtml-sqlgroup>
```

如果 first 为 Bob ，last 为 Smith，McDonald，它处理结果为：

```
select * from employees
where
(first='bob'
 and
 last in ('Smith', 'McDonald')
)
```

如果 salary 为 50000 ，last 为 Smith，它处理结果为：

```
select * from employees
where
(salary > 50000.0
 and
 last='Smith'
)
```

签套的 sqlgroup 标记符：

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqlgroup>
     <dtml-sqltest first op=like type=string>
  <dtml-and>
     <dtml-sqltest last op=like type=string>
  <dtml-sqlgroup>
<dtml-or>
  <dtml-sqltest salary op=gt type=float>
</dtml-sqlgroup>
```

给定例子参数，这个模板处理的结果是这样的 SQL：

```
select * form employees
where
(
  (
   name like ‘A*’
   and
   last like ‘Smith’
   )
 or
```

```
 salary > 20000.0
)
```

参见

sqltest 标记符

14. *sqltest*: 格式 SQL 条件测试

sqltest 标记符把一个条件测试插入到 SQL 代码中。它对一个列和一个变量进行测试。这个标记符用在 SQL 方法里。

句法

sqltest 标记符句法：

```
<dtml-sqltest Variable|expr="VariableExpression">
```

sqltest 标记符是一种独立标记符。它插入一段 SQL 条件测试语句。它用于构建 SQL 查询。sqltest 标记符恰当的避开了被插入的变量。被指定的变量或变量表达式和 SQL 列之间使用指定的比较操作符进行比较。

属性

type=string——变量的类型。有效的类型包括 string、int、float 和 nb。nb 含义是非空字符串。类型属性是必需的并且被用于把一个被插入的变量转换成恰当的类型。

column=string——用来进行测试的 SQL 列的名称。这个属性默认为变量名称。

multiple=boolean——如果为真，那么变量可以是一个用来测试列的值序列。

optional=boolean——如果为真，那么测试是可选择项，并且如果变量为空或不存在，它将不被处理。

op=string——比较操作。有效的比较包括以下：

eq—— 等于

gt—— 大于

lt—— 小于

ne—— 不等于

ge—— 大于或等于

le—— 小于或等于

默认的比较是等于。如果比较不被识别，就用它。这样，你总是可以这样使用比较。

例子

基本用法：

```
select * from employees

  where <dtml-sqltest name type="string">
```

如果 name 变量是 Bob，那么这会呈递为：

```
select * from employees

  where name = 'bob'
```

多个值：

```
select * from employees
  where <dtml-sqltest empid type=int multiple>
```

如果 empid 变量是(12,14,17)，那么这会呈递为：

```
select * from employees
  where empid in (12, 14, 17)
```

参见

sqlgroup 标记符

sqlvar 标记符

15. *sqlvar*：插入 SQL 变量

sqlvar 标记符把变量插入到 SQL 代码中。这个标记符用在 SQL 方法中。

句法

sqlvar 标记符句法:

```
<dtml-sqlvar Variable|expr="VariableExpression">
```

sqlvar 标记符是一种独立标记符。就像 var 标记符，sqlvar 标记符查找变量并插入它。不像 var 标记符，格式选项是为了用于调整 SQL 代码。

属性

type=string——变量的类型。有效的类型包括 string、 int,、 float 和 nb。nb 含义为非空字符串。类型属性是必需的并且被用来把一个被插入的变量转换成恰当的类型。

optional=boolean——如果为真，并且变量为空或者不存在，那么不插入任何内容。

例子

基本用法：

```
select * from employees

  where name=<dtml-sqlvar name type="string">
```

这段 SQL 引用 name 字符串变量

参见

sqltest 标记符

16. *string*：DTML 字符串函数

字符串模块提供了字符处理、转换和搜索函数。它是一个标准的 Python 模块。

函数

atof(s)——把一个字符串转换成一个浮点数。字符串必需在字面上有 Python 中的浮点标准句法，可选择在前面添加有符号（“+” 或 “-”）。注意，这个行为在传递一个字符串时等同于内建的函数 float()。

atoi(s [,base])——把字符串转换成一个给定基数中的整数。字符串必需由一个或多个数字组成，可选择在前面添加有符号（“+” 或 “-”）。基数默认为 10。如果它为 0，一个默认基数被选择，取决于字符串的头几个字符（删去符号以后）： “0x” 或者 “0X”意味着 16， “0” 意味着 8，其它的意味着 10。如果基数是 16，首字为 “0x” 或 “0X”总是可以接受的，尽管不是必须的。

atol(s, [,base])——把字符串转换成一个给定基数中的长整数。字符串必需由一个或多个数字组成，可选择在前面添加有符号（“+” 或 “-”）。基数参数的含义和 atoi()中的一样。结尾是"l” 或 "L”是不允许的，除了基数是 0 以外。

capitalize(word)——大写参数的第一个字符。

capwords(s)——使用 split()把参数分割成单词，大写每个单词使用 capitalize()，连接大写化的单词使用 join()。注意，它把多个连续空格用一个空格替换并且删除头部和尾部的空格。

find(s, sub [,start [,end]])——返回在 s 里找到字符串子集 sub 处的最低索引，这样一来，sub 完全包含在 s[start:end]中。失败时它返回一个-1 。start、end 和负数值的解释和分片（slices）中的一样。

rfind(s, sub [,start [,end]])——类似于 find()，不同点在于它查找最高索引。

index(s, sub [,start [,end]])——类似于 find()，不同点在于当字符串子集没有被找到时，引发一个 ValueError。

rindex(s, sub [,start [,end]])——类似于 rfind()，不同点在于当字符串子集没有被找到时，引发一个 ValueError。

count(s, sub [,start [,end]])——返回字符串子集 sub 在字符串 s[start:end]中出现的次数（不重叠）。默认的 start、 end 和 负数值的解释和分片（slices）中的一样。

lower(s)——返回一个 s 的副本，但是把大写字母转换成小写字母。

makestrans(from, to)——返回适合传递给 translate()的转换表，其把每个 from 中的字符映射成在 to 中相同位置的字符。from 和 to 必须长度相同。

split(s, [,sep [,maxsplit]])——返回字符串 s 中的单词列表。如果可选择的第二个参数 sep 缺少或为 None，这些单词就用空格字符分隔（比如空格，tab，换行符，回车，进纸）。如果提供了第二个参数 sep 并且不为 None。它指定一个用分割单词的字符。比起字符串中的非重叠分隔符出现的次数，返回的列表将多一个项。可选择的第三个参数 maxsplit 默认为 0。如果它为非零，至多发生 maxsplit 次的分割，并且余下的字符串作为最终的列表元素被返回（这样列表将有至多 maxsplit+1 个元素）。

join(words [,sep])——用插入分隔符 sep 的方法把单词列表或元组连接起来。sep 的默认值为一个空格符。string.join(string.split(s, sep), sep)等于 s 总为真。

lstrip(string)——返回一个最前边没有空格符的字符串 s 的副本。

rstrip(string)——返回一个结尾没有空格符的字符串 s 的副本。

strip(string)——返回一个最前边或结尾没有空格符的字符串 s 的副本。

swapcase(s)——返回一个把小写字母转换成大写字母的字符串 s 的副本，反之亦然。

translate(s, table [,deletechars])——从 s 中删除所有的 deletechars （如果提供的话）里的字符，然后使用 table 转换这些字符，其中必须是为每个字符值和次序索引赋予转换的 256 字符集字符串。

upper(s)——返回一个把小写字母转换成大写字母的字符串副本。

ljust(string, width)——在一个给定宽度的区域中左对齐一个字符串。返回一个至少达到字符宽度要求的字符串，方式是通过把空格填充到字符串中，直到给定的宽度为止。字符串从不被删节。

rjust(string, width)——在一个给定宽度的区域中右对齐一个字符串。返回一个至少达到字符宽度要求的字符串，方式是通过把空格填充到字符串中，直到给定的宽度为止。字符串从不被删节。

center(string, width)——在一个给定宽度的区域中居中一个字符串。返回一个至少达到字符宽度要求的字符串，方式是通过把空格填充到字符串中，直到给定的宽度为止。字符串从不被删节。

zfill(s, width)——用 0 数字填充数字字符串的左侧，直到达到指定的宽度。前边带有符号的字符串可以被正确处理。

replace(s, old, new [,maxsplit])——把字符串 s 中的原有的字符串子集 old 替换成新的字符串子集 new，返回最终的字符串副本。如果给定可选的参数 maxsplit ，那么出现的第一个 maxsplit 被替换。

属性

digits——字符串 0123456789 。

hexdigits——字符串 0123456789abcdefABCDEF 。

letters——所有下边所述的小写字母和大写字母。

lowercase——包含所有的小写字母的字符串。在大多数系统上，是 abcdefghijklmnopqrstuvwxyz 。

octdigits——字符串 01234567 。

uppercase——包含所有的大写字母的字符串 ABCDEFGHIJKLMNOPQRSTUVWXYZ。

whitespace——包含所有被认为是空格符的字符串。在大多数系统中，包括空格符、tab、换行符、回车符、进纸和垂直 tab。

参见

Python 字符串模块 (http://www.python.org/doc/current/lib/module-string.html)

17. t*ree*：插入一个树部件

tree 标记符通过查询 Zope 对象显示一个动态的树部件。

句法

tree 标记符句法：

```
<dtml-tree [VariableName|expr="VariableExpression"]>

</dtml-tree>
```

tree 标记符是一种块标记符。它呈递一个 HTML 形式的动态树部件。树的根由变量或表达式给定。另外，它默认为当前的对象。通过把当前的节点映射成 DTML 名称空间，对每个节点进行树块呈递。

树的状态被设置在 HTTP cookies 中。这样，要使用树，就必须激活 cookies 。另外，每页只显示一个树。

属性

branches=string——通过调用指定的方法查找树的分支。默认的方法是 tpValues，大多数 Zope 对象支持它。

branches_expr=string——通过对表达式求值来查找树的分支。

id=string——用于测定树的状态的方法或 id 的名称。它默认为 tpId，大多数 Zope 对象都支持它。这个属性只适合高级用法。

url=string——用于测定树的数据项 URL 的方法或属性的名称。它默认为 tpURL，大多数 Zope 对象都支持它。这个属性适合高级用法。

leaves=string——用来呈递没有子项的节点的 DTML 文档或方法的名称。注意：这个文档应该以<dtml-var standard_html_header>，以<dtml-var standard_html_footer>结束，从而确保正确的在树中显示

header=string——节点展开以前被显示的 DTML 文档或方法的名称。如果页眉没有被找到，它就被忽略掉。

footer=string——节点展开以后被显示的 DTML 文档或方法的名称。如果页脚没有被找到，它就被忽略掉。

nowrap=boolean——如果为真，那么节点不是被隐藏起来，节点而是按照现有大小被缩短。

sort=string——按照指定的属性对分支进行排序。

reverse——颠倒分支的次序。

assume_children=boolean——假设节点有子项。如果取得和查询子节点是一个费时的过程，它是有用的。在加号框内可以被下拉的的结果紧挨所有的节点。

single=boolean——只允许一次展开一个分支。当你展开一个新的分支时，任何其他的已经展开的分支关闭。

skip_unauthorized——忽略用户不允许看到的节点，而不是引发一个错误。

urlparam=string——被包含在展开和缩进部件连接里的查询字符串。这个属性只适用于高级用法。

标记符标量

tree-item-expanded——如果当前的节点已经展开则为真。

tree-item-url——当前节点的 URL。

tree-root-url——根节点的 URL。

tree-level——当前节点的深度。顶级节点的深度为 0。

tree-colspan——正在被呈递的树的最大层级。当把行插入到树表格中时，这个变量可以随同树级变量被用来计算表格行和列跨度设置。

tree-state——用 id 列表和 id 列表子集表示的树的状态。这个变量只适用于高级用法。

标记符控制变量

通过设置以下变量，你可以控制树标记符。

expand_all——如果这个变量为真，那么整个树是展开的。

collapse_all——如果这个变量为真，那么整个树是合拢的。

例子

以下显示一个在当前对象中生成的树：

```
<dtml-tree>
  <dtml-var title_or_id>
</dtml-tree>
```

以下显示一个用特定的分支方法在另外一个对象中生成的树：

```
<dtml-tree expr="folder.object" branches="objectValues">

  Node id : <dtml-var getId>

</dtml-tree>
```

18. *try*：处理例外

try 标记符使你能够用 DTML 处理例外，反映 Python 中的 try/except 和 try/finally 构造。

句法

try 标记符有两个不同的句法，try/except/else 和 try/finally

try/except/else 句法:

```
<dtml-try>

<dtml-except [ExceptionName] [ExceptionName]...>

...

[<dtml-else>]

</dtml-try>
```

try 标记符在一个块中捕捉和处理例外。可以有一个或多个 except 标记符用来处理无例外或多个例外。如果一个 except 标记符没有指定一个例外，那么它处理所有的例外。

当一个例外被引发，立即让第一个标记符处理这个例外。如果没有 except 标记符来处理例外，那么例外以通常的方式被引发。

如果没有例外被引发，并且有一个 else 标记符，那么 else 标记符在 try 标记符正文以后被执行。

except 和 else 标记符是可选择的。

try/finally 句法

<dtml-try>

<dtml-finally>

</dtml-try>

finally 标记符不能象 except 和 else 标记符那样在相同的 try 块中使用。如果有一个 finally 标记符，它的块不管例外在 try 块内是否被引发都将被执行。

属性

except——0 个或多个例外名称。如果没有列出例外，那么 except 标记符处理所有的例外。

标记符变量

在 except 块内部定义了这些变量：

error_type——例外类型。

error_value——例外的值。

error_tb——回溯

例子

捕捉一个数学错误：

```
<dtml-try>

<dtml-var expr="1/0">

<dtml-except ZeroDivisionError>

You tried to divide by zero.

</dtml-try>
```

返回关于被处理的例外的信息：

```
<dtml-try>

<dtml-call dangerousMethod>

<dtml-except>

An error occurred.

Error type: <dtml-var error_type>

Error value: <dtml-var error_value>

</dtml-try>
```

使用 finally 来确保正常执行，不管是否引发一个错误：

```
<dtml-call acquireLock>

<dtml-try>

<dtml-call someMethod>

<dtml-finally>

<dtml-call releaseLock>
```

```
</dtml-try>
```

参见

raise 标记符

Python 指南: 错误和例外 (http://www.python.org/doc/current/tut/node10.html)

Python 内建例外 (http://www.python.org/doc/current/lib/module-exceptions.html)

19. *unless*：测试一个条件

unless 标记符提供了一个测试相反条件的快捷方式。对于更为完整的条件测试，请使用 if 标记符。

句法：

unless 标记符句法:

```
<dtml-unless ConditionVariable|expr="ConditionExpression">
</dtml-unless>
```

unless 标记符是一种块标记符。如果条件变量或表达式求值为假，那么就执行块中所包含内容。就像 if 标记符，没有提供变量被认为假。

例子：

测试一个变量：

```
<dtml-unless testMode>
  <dtml-call dangerousOperation>
</dtml-unless>
```

如果 testMode 不存在就执行这个块，或者只要为假就退出。

参见

if 标记符

20. *var*：插入一个变量

var 标记符使你能够把一个变量插入到 DTML 输出中。

句法

var 标记符句法：

```
<dtml-var Variable|expr="Expression">
```

var 标记符是一种独立标记符。var 标记符通过搜索 DTML 名称空间查找一个变量，DTML 名称空间通常包括当前对象、当前对象的容器和 Web 请求。如果变量被找到，就把它插入到 DTML 输出中。如果它没有被找到，Zope 引发一个错误。

var 标记符实体句法：

&dtml-variableName;

实体句法是一种插入的快捷方式并且 HTML 引用变量。当把变量插入到 HTML 标记符中时会用到它。

带有属性的 var 标记符实体句法：

&dtml.attribute1[.attribute2]...-variableName;

在某种程度上，你可以用实体句法指定属性。你可以包含 0 个或多个用句点分开的属性。你不能使用实体句法为属性提供参数如果你提供 0 个或多个属性，那么变量不是自动的被 HTML 引用。这样一来，你可以避免使用这种句法的 HTML 引用，&dtml.-variableName;。

属性

html_quote——把在 HTML 中具有特定含义的字符转换成 HTML 字符实体。

missing=string——在 Zope 不能找到变量的情况下，指定一个默认值。

fmt=string——格式一个变量。Zope 提供一些内建的格式，包括 C 风格的格式字符串。关于 C 风格的格式字符串方面的信息，请见 Python 库参考(http://www.python.org/doc/current/lib/typesseq-strings.html)。如果格式字符串不是一个内建格式，那么它被假定为一个对象方法，然后它被调用。

whole-dollars——按美元格式化变量。

dollars-and-cents——按美元和美分格式化变量。

collection-length——变量的长度，假设它是一个序列。

structured-text——按照结构文本格式化变量。关于结构文本方面的更多信息，请见 Zope.org Web 站点中的 Structured Text How-To (http://www.zope.org/Members/millejoh/structuredText)。

null=string——如果变量为 None 时使用的默认值。

lower——把大写字母转换成小写字母。

upper——把小写字母转换成大写字母。

capitalize——把被插入的单词的首个字符变成大写。

spacify——把被插入的值中的下划线更改成空格。

thousands_commas——在包含数字的值中，从小数点左边开始每隔三个数字插入逗号，例如 12000 变为 12,000 。

url——通过调用对象的 absolute_url 方法插入对象的 URL 。

url_quote——把 URL 中具有特殊含义的字符转换成 HTML 字符实体。

url_quote_plus——URL 引用字符，例如 url_quote ，但是它还把空格转换成加号。

sql_quote——把单引号转换为成对的单引号。安全的在 SQL 字符串中包含值时需要用到它。

newline_to_br——把换行符（包含回车）转换成 HTML 换行符。

size=arg——按照给定长度截取变量（注意：如果在被截取的字符串的下半部有空格，那么字符串被截取到最右边的空格。

etc=arg——指定一个用来添加到被截取的字符串（通过设置前边所述的 size 属性）结尾处的字符串。默认为… 。

例子

在一个文档中插入一个简单变量：

```
<dtml-var standard_html_header>
```

补加：

```
<dtml-var colors size=10 etc="， etc.”>
```

如果 colors 是字符串 red yellow green，生成以下输出：

red yellow, etc.

C 风格字符格式：

```
<dtml-var expr="23432.2323" fmt="%.2f">
```

呈递为

23432.23

用实体句法在一个 HTML A 标记符内插入一个变量链接：

```
<a href="&dtml-link;">Link</a>
```

使用带有属性的实体句法给一个文档 doc 加入一个链接：

```
<a href="&dtml.url-doc;"><dtml-var doc fmt="title_or_id"></a>
```

这样就给一个对象创建一个使用它的 URL 和标题的 HTML 链接。这个例子针对 URL（使用 url 属性）调用对象的 absolute_url 方法，针对标题调用它的 title_or_id 方法。

21. *with*：控制查找 DTML 变量

with 标记符把一个对象推进到 DTML 名称空间。变量首先在被加入的对象中查找。

句法

with 标记符句法：

```
<dtml-with Variable|expr="Expression">
</dtml-with>
```

with 标记符是一种块标记符。它在 with 块的持续期间内把指定的变量或变量表达式推进到 DTML 名称空间中。这样，首先就

在被添加的对象中查找名称。

属性

only——限制 DTML 名称空间，使之仅包含在 with 标记符中定义的名称空间。

mapping——表示变量或表达式是一个映射对象。这样就确保在映射对象里正确查找变量。

例子

在 REQUEST 里查找一个变量：

```
<dtml-with REQUEST only>
  <dtml-if id>
    <dtml-var id>
  <dtml-else>
     ‘id’ was not in the request.
  </dtml-if>
</dtml-with>
```

把 first child 添加到 DTML 名称空间：

```
<dtml-with expr=”objectValues()[0]”>
  First child's id: <dtml-var id>
</dtml-with>
```

参见

let 标记符

第二十五章 API 参考

这个参考主要描述了最常用的 Zope 对象接口。适用于编写 DTML、Perl 和 Python 脚本，从而创建和操纵 Zope 对象。

1. AccessControl 模块

1.1. AccessControl: 安全函数和类

这个模块中的函数和类适用于基于 Python 的脚本和页面模板。

1.1.1. SecurityManager 类

SecurityManager 提供检查权限的方法。

1.1.1.1. calledByExecutable(self)

返回一个布尔变量，表示是否通过可执行对象进行调用。

许可

始终存在

1.1.1.2. validate(accessed=None, container=None, name=None, value=None, roles=None)

验证权限

参数:

accessed

正在访问的对象

container

对象所在的容器对象

name

要访问的变量名称

value

通过访问要取得的数值

roles

已知对象的角色

这些参数可以通过关键字参数的方式提供。其中一些参数可以忽略，但在某些情况下有可能会拒绝访问。最好提供所有的参数。

许可

始终存在

1.1.1.3. checkPermission(self, permission, object)

检查指定的对象是否具有指定的许可。

许可

始终存在

1.1.1.4. getUser(self)

取得当前已授权用户。参见 AuthenticatedUser 类。

许可

始终存在

1.1.1.5. validateValue(self, value, roles=None)

常用的验证数值函数

许可

始终存在

2. AuthenticatedUser 模块

2.1. AuthenticatedUser 类

这个接口需要由用户验证后返回的对象来提供支持，这个接口用于访问控制。

2.1.1. getUserName()

返回用户名称

许可

始终存在

2.1.2. getId()

取得用户的 ID。ID 可以用于通过 Python 从用户的 UserDatabase 中得到用户。

许可

始终存在

2.1.3. has_role(roles, object=None)

如果用户具有 roles 列表中的值，至少要有一个，则返回真。object 是可选的环境对象。

许可

始终存在

2.1.4. getRoles()

返回用户角色的列表。

许可

始终存在

2.1.5. has_permission(permission, object)

如果用户在 object 对象上有 permission 许可，则返回真。

许可

始终存在

2.1.6. getRolesInContext(object)

返回分配用户的角色列表，包含在 object 对象环境中分配的本地角色。

许可

始终存在

2.1.7. getDomains()

返回用户的域限制列表。

许可

始终存在

3. DTMLDocument 模块

3.1. DTMLDocument(ObjectManagerItem, PropertyManager) 类

DTML 文档是包含和执行 DTML 代码的 Zope 对象。它用于显示 Web 页面。

3.1.1. manage_edit(data, title)

更改 DTML 文档，用 data 替换内容，用 title 更改标题。 data 参数可以是一个文件对象或一个字符串。

许可

Change DTML Documents

3.1.2. document_src()

返回 DTML 文档的源文本。

许可

View management screens

3.1.3. __call__(client=None, REQUEST={}, RESPONSE=None, **kw)

调用 DTML 文档，执行文档中所包含的 DTML 代码。这个方法返回对象执行的结果。

在这个过程中，DTML 文档经常需要把各种名称解析成对象。例如，当执行代码 <dtml-var spam> 时，DTML 引擎试图解析名称 spam。

要解析名称，必须给文档传递一个名称空间，在这个名称空间里查找这些名称。这个过程可以通过以下几种方式完成：

通过传递一个 client 对象 -- 如果传递参数 client，那么名称作为参数的属性来查找。

- 通过传递一个 REQUEST 映射 -- 如果传递了参数 REQUEST，那么名称作为参数的数据项来查找。如果对象不是一个映射，当视图查找名称时引发一个 TypeError 错误。

通过传递关键字参数 -- 名称和它们的值可以作为文档的关键字参数来传递。

DTML 文档所被赋予的名称空间是这三种方式的混合物。你可以传递任意数量的参数或者根本就不传递参数。名称首先在关键字参数中查找，然后在 client 里，最后在 REQUEST 映射里。

除了在指定的 client 里查找名称，DTML 文档本身可以作为一个 client 参数来传递它自己，因此还可以在 DTML 文档本身里查找名称。

通过名称空间给 DTML 文档传递参数也可以是认为是给文档提供关联环境。

DTML 文档可以通过三种方式调用。

通过 DTML

DTML 文档可以通过另外一个 DTML 方法或文档来调用：

```
<dtml-var standard_html_header>

  <dtml-var aDTMLDocument>

<dtml-var standard_html_footer>
```

在这个例子里，文档 aDTMLDocument 是通过另外一个 DTML 对象采用名称的方式来调用的。调用方法传递的 client 参数值为 this，REQUEST 参数为当前的 DTML 名称空间。前边的代码等同于以下 DTML 里的 Python 表达式中的用法：

```
<dtml-var standard_html_header>

  <dtml-var "aDTMLDocument(_.None, _)">

<dtml-var standard_html_footer>
```

通过 Python

产品、外部方法和脚本可以采用相同的方式调用 DTML 文档，就像它们在 DTML 里通过一个 Python 表达式调用 DTML 文档那样（如前边的例子所示）。

通过公布者（By the Publisher）

当通过 URL 调用一个 DTML 文档时，DTML 文档通过公布者调用。REQUEST 对象作为第二个参数传递给文档。

许可

View

3.1.4. get_size()

返回 DTML 文档的源文本的大小，单位是字节。

许可

View

3.2. ObjectManager 构造器

3.2.1. manage_addDocument(id, title)

给当前的 ObjectManager 添加一个 DTML 文本。

4. DTMLMethod 模块

4.1. DTMLMethod(ObjectManagerItem) 类

DTML 方法是包含和执行 DTML 代码对象。它可以作为一个模板来显示其它对象。它还可以存储一小段内容，这些内容可以被插入到其它的 DTML 文档或 DTML 方法里。

DTML 方法的 id 通过 document_id 变量提供，title 通过 document_title 变量提供。

4.1.1. manage_edit(data, title)

用于更改 DTML 方法，用 data 替换内容，更改 title。data 参数可以是一个文件对象或字符串。

许可

Change DTML Methods

4.1.2. document_src()

返回 DTML 方法的源文本。

许可

View management screens

4.1.3. __call__(client=None, REQUEST={}, **kw)

调用 DTML 方法，执行其中的 DTML 代码。这个方法返回对象执行的结果。

在这个过程中，DTML 方法经常需要把各种名称解析成对象。例如，当执行代码 <dtml-var spam> 时，DTML 引擎试图解析名称 spam。

要解析名称，必须给文档传递一个名称空间，在这个名称空间里查找这些名称。这个过程可以通过以下几种方式完成：

通过传递一个 client 对象 -- 如果传递参数 client，那么名称作为参数的属性来查找。

- 通过传递一个 REQUEST 映射 -- 如果传递了参数 REQUEST，那么名称作为参数的数据项来查找。如果对象不是一个映射，当视图查找名称时引发一个 TypeError 错误。

通过传递关键字参数 -- 名称和它们的值可以作为文档的关键字参数来传递。

DTML 方法所被赋予的名称空间是这三种方式的混合物。你可以传递任意数量的参数或者根本就不传递参数。名称首先在关键字参数中查找，然后在 client 里，最后在 REQUEST 映射里。

不同于 DTML 文档，DTML 方法不在它们自己的实例字典里查找名称。

通过名称空间给 DTML 文档传递参数也可以是认为是给文档提供关联环境。

DTML 文档可以通过三种方式调用。

通过 DTML

DTML 方法可以通过另外一个 DTML 方法或文档来调用：

```
<dtml-var standard_html_header>

  <dtml-var aDTMLMethod>

<dtml-var standard_html_footer>
```

在这个例子里，方法 aDTMLMethod 是通过另外一个 DTML 对象采用名称的方式来调用的。调用方法传递的 client 参数值为 this，REQUEST 参数为当前的 DTML 名称空间。前边的代码等同于以下 DTML 里的 Python 表达式中的用法：

```
<dtml-var standard_html_header>

  <dtml-var "aDTMLMethod(_.None, _)">

<dtml-var standard_html_footer>
```

通过 Python

产品、外部方法和脚本可以采用相同的方式调用 DTML 方法，就像它们在 DTML 里通过一个 Python 表达式调用 DTML 方法那样（如图前边的例子所示）。

通过公布者（By the Publisher）

当通过 URL 调用一个 DTML 方法时，DTML 文档通过公布者调用。REQUEST 对象作为第二个参数传递给文档。

许可

View

4.1.4. get_size()

返回 DTML 方法的源文本的大小，单位是字节。

许可

View

4.2. ObjectManager 构造器

4.2.1. manage_addDTMLMethod(id, title)

给当前的 ObjectManager 添加一个 DTML 文本。

5. DateTime 模块

5.1. DateTime 类

DateTime 对象提供处理日期和时间的接口。 DateTime 还提供了日历操作方法、日期和时间算法和格式化方法。

DateTime 对象可以表示当前事件，并且提供显示接口，不影响对象绝对值。

DateTime 对象可以通过形式多样的字符串或数值来创建，或者通过其它 DateTime 对象计算得到。 DateTime 可以把时间的格式转换成不同时区的格式，还可以在一个按照给定时区创建 DateTime 对象。

DateTime 对象支持数值计算：

- 两个 DateTime 对象可以相减，从而获得两者间的时间差。
- 一个 DateTime 对象和一个正的或负的数字可以相加，从而获得一个新 DateTime 对象。
- 一个正的或负的数字和一个 DateTime 对象可以相加，从而获得一个新 DateTime 对象。
- 一个正的或负的数字可以从一个 DateTime 对象中减去，从而获得一个新 DateTime 对象。

DateTime 对象通过使用标准的 int、long 和 float 函数可以把从 1901 年 1 月 1 日以来的时间转换成整数、长整数和浮点数形式。（兼容性注意：int、long 和 float 返回自从 1901 年以来的 GMT 天数，而不是按照本地时区计算的天数）。DateTime 对象还可以访问以浮点数格式表示的值，它可以和 Python 的 time 模块一起使用，前提是对象的值属于这个基于新纪元的 time 模块的时间值的范围之内。

DateTime 对象应该认为是不变的，所有转换和数值操作返回一个新的 DateTime 对象而不是修改当前的对象。

DateTime 对象总是按照绝对 UTC 时间维护数值，根据时区以及参数来提供数值。DateTime 对象方法基于时区返回相应的值

注意如果没有指定时区，默认使用本地时区来表示时间。

创建 DateTime 对象可使用 0 到 7 个参数。

如果采用不带参数的方式调用函数，那么返回当前的日期时间，并且按照本地时区表示。

- 如果采用带有一个字符串参数的方式调用函数，其中这个字符串代表时区名称，那么返回指定时区示当前时间的 DateTime 对象。

如果采用带有唯一的字符串参数的方式调用函数，其中字符串表示一个有效的日期或时间，那么就返回相应的日期或时间对象。

  - 一般来说，任何北美居民可以明确识别的日期或时间格式都是可接受的。（其中的原因是：在北美，像 2/1/1994 这样的日期被认为是 February 1, 1994，然而在世界上的一些地方，它被认为是 January 2, 1994。）一个日期时间对象包含两个部分：日期部分和可选的时间部分，由一个或多个空格分隔。如果时间部分忽略，则假定为 12:00am。时区名称可以在日期时间字符串最后一个元素中指定，任何可以识别的时区名称用于计算日期时间值。（如果你用字符串 Mar 9, 1997 1:45pm US/Pacific 创建一个 DateTime 对象，它的值在本质上等同于你在一台属于那个时区的机器上在指定的日期和时间捕捉的时间）：

```
e=DateTime("US/Eastern")

# returns current date/time, represented in US/Eastern.

x=DateTime("1997/3/9 1:45pm")

# returns specified time, represented in local machine zone.

y=DateTime("Mar 9, 1997 13:45:00")

# y is equal to x
```

日期部分由年、月和日的值组成。年的值必须为 1 位、2 位或 4 位数的整数。如果使用 1 位或 2 位数，年被假设属于 20 世纪。月可以是一个整数，从 1 到 12，也可以是月的名称或月的缩写，其中一个句点可以选择性的跟随在缩写后。日必须属于从 1 到该月

的天数之间的整数。年、月和日的值可以用句点、连字号、右箭号或空格分隔。在分隔符周围允许使用额外的空格。年、月和日的值可以按照任何顺序给定，只要能够区分出组件。如果所有这三个组件都是小于 13 的数字，那么假定的顺序为月-日-年。

time 部分由小时、分钟和秒的值组成，用冒号分隔。小时的值必须是一个 0 至 23（包含 0 和 23）之间的整数，分钟的值必须为 0 至 59（包含 0 和 59）之间的整数。秒的值可以为 0 至 59.999（包含 0 和 59.999）之间的整数。秒的值，或者分钟和秒的值，可以忽略。时间可以跟随大写或小写格式的 am 或 pm，其被假定为 12 小时制。

- 如果 DateTime 函数被调用时带有一个数字参数，这个数字被假定为浮点数值，例如由 time.time()返回的值。返回的 DateTime 对象表示了用本地时区表示的浮点数形式的 gmt 值。

如果函数调用时带有两个数字参数，那么第一个被认为是一个整数年，第二个参数被认为是在本地时区中本年开始以来的天数偏移量。返回的日期时间值是用本地时区表示的给定年份开始以来的天数的给定偏移量。偏移量可以是正数或负数。两位数的年被假定为 20 世纪当中的年份。

- 如果函数调用时带有两个参数——第一个以浮点数形式提供的参数表示了在 GMT 里新纪元以来的秒数，就像那些由 time.time()返回的数字，第二个以字符串形式提供的参数指定一个可识别的时区，返回具有 GMT 时间值的 DateTime 对象并按照给定时区形式表示。

```
import time

t=time.time()

now_east=DateTime(t,'US/Eastern')

# Time t represented as US/Eastern

now_west=DateTime(t,'US/Pacific')

# Time t represented as US/Pacific

# now_east == now_west

# only their representations are different
```

- 如果函数调用时带有三个或更多的数字参数，那么第一个被认为是整数年，第二个被认为是整数月，第三个被认为是整数天。如果结合在一起的值无效，那么引发一个 DateTimeError 。两位数的年被认为是 20 世纪中的年份。第 4、5、6 个参数分别指定小时、分钟和秒——小时和分钟应该为正整数，秒应该为一个正的浮点数——所有这些如果没有给定则默认为 0。可以给定一个可选择的字符串，从而作为最后一个参数来表示时区（这个效果就好像是你已经在一台位于指定时区中的机器上选定了 time.time()的值。）

如果传递给 DateTime 构造器的字符串参数不能解析，它引发一个 DateTime . SyntaxError 错误。无效的日期、时间或时区组件引发一个 DateTime . DateTimeError 错误。

模块函数 Timezones()返回一个 DateTime 模块可识别的时区列表。时区名称识别是不区分大小写的。

5.1.1. strftime(format)

返回按照 format 格式提供的日期时间字符串。

参见 Python 中的 time.strftime 函数。

5.1.2. dow()

返回用整数表示的星期中的天数，星期日是 0。

许可

始终存在

5.1.3. aCommon()

返回按照“Mar 1, 1997 1:45 pm”格式表示的日期时间字符串。

许可

始终存在

5.1.4. h_12()

返回 12 小时制的小时数。

许可

始终存在

5.1.5. Mon_()

兼容：见 pMonth。

许可

始终存在

5.1.6. HTML4()

按照符合 HTML 4.0 规范的格式返回对象，这个规范是 ISO8601 标准之一。

参见 HTML 4.0 规范

日期输出格式为：YYYY-MM- MM:SSZ T，其中 Z 是文本字符。时间为 UTC（通用协调时间）时间。

许可

始终存在

5.1.7. greaterThanEqualTo(t)

和其它 DateTime 对象或浮点数比较 DateTime 对象，比如由 Python 的 time 模块返回的数值。如果对象表示一个大于或等于指定的 DateTime 或 time 模块风格的时间的日期或时间对象，则返回真。通过比较长整数型的毫秒，它可以给出更为精确的结果。

许可

始终存在

5.1.8. dayOfYear()

返回按照对象所在时区表示的年的天数。

许可

始终存在

5.1.9. lessThan(t)

与其它的 DateTime 对象或一个浮点数比较 DateTime 对象，比如由 Pythontime 模块返回的数字。如果对象表示一个小于指定的 DateTime 或 time 模块风格的时间的日期或时间对象，则返回真。通过比较长整数型毫秒，它可以给出更为精确的结果。

许可

始终存在

5.1.10. AMPM()

返回一个对象的最接近秒的时间字符串。

许可

始终存在

5.1.11. isCurrentHour()

如果这个对象在所在时区中表示一个属于当前小时范围里的日期或时间对象，则返回真。

许可

始终存在

5.1.12. Month()

返回完整的月份的名称。

许可

始终存在

5.1.13. mm()

以两位数字符形式返回月份。

许可

始终存在

5.1.14. ampm()

返回适当的时间修饰语（am 或 pm）。

许可

始终存在

5.1.15. hour()

返回以 24 小时制表示的小时。

许可

始终存在

5.1.16. aCommonZ()

返回以"Mar 1, 1997 1:45 pm US/Eastern"格式表示对象值的字符串。

许可

始终存在

5.1.17. Day_()

兼容：见 pDay。

许可

始终存在

5.1.18. pCommon()

返回以"Mar. 1, 1997 1:45 pm"格式表示的对象值的字符串。

许可

始终存在

5.1.19. minute()

返回分钟。

许可

始终存在

5.1.20. day()

返回以整数表示的天。

许可

始终存在

5.1.21. earliestTime()

返回一个新的表示最早时间（全部按秒计算）的 DateTime 对象，它仍然属于对象所在时区中的当前天。

许可

始终存在

5.1.22. Date()

返回对象的日期字符串。

许可

始终存在

5.1.23. Time()

返回对象的最接近秒的时间字符串。

许可

始终存在

5.1.24. isFuture()

如果这个对象表示一个晚于调用时间的时间日期对象，则返回真。

许可

始终存在

5.1.25. greaterThan(t)

和其它的 DateTime 对象或一个浮点数比较 DateTime 对象，比如和由 Pythontime 模块返回的数字比较。如果对象表示一个大于指定的 DateTime 或符合 time 模块风格的时间的 date/time 对象，则返回真。通过比较长整数型毫秒，它可以给出更为精确的结果。

许可

始终存在

5.1.26. TimeMinutes()

返回对象的时间字符串，不显示秒。

许可

始终存在

5.1.27. yy()

返回以两位数字符表示的日历年。

许可

始终存在

5.1.28. isCurrentDay()

如果对象在所在时区中表示一个属于当前天范围内的日期时间对象，则返回真。

许可

始终存在

5.1.29. dd()

返回以两位数字符形式表示的天。

许可

始终存在

5.1.30. rfc822()

返回以 RFC 822 格式显示的日期。

许可

始终存在

5.1.31. isLeapYear()

如果当前年（在对象所属时区中）是闰年则返回真

许可

始终存在

5.1.32. fCommon()

返回一个以"March 1, 1997 1:45 pm"格式表示的对象值的字符串。

许可

始终存在

5.1.33. isPast()

如果对象表示一个早于调用时间的日期时间对象，则返回真。

许可

始终存在

5.1.34. fCommonZ()

返回一个以"March 1, 1997 1:45 pm"格式表示的对象值的字符串。

许可

始终存在

5.1.35. timeTime()

返回 UTC 中按照 Python time 模块所使用的格式以浮点数形式表示的日期时间。注意，采用那些拥有对于 time 模块来说没有含义的值的 DateTime 来创建日期或时间是可能的。

许可

始终存在

5.1.36. toZone(z)

返回当前对象在指定的 z 时区中的 DateTime 。

许可

始终存在

5.1.37. lessThanEqualTo(t)

和另外一个 DateTime 对象或一个浮点数比较 DateTime 对象，比如和由 Python time 模块返回的数字进行比较。如果对象表示一个小于或等于指定的 DateTime 或 time 模块风格的时间的日期时间，则返回真。通过比较长整数型毫秒，它可以给出更为精确的结果。

许可

始终存在

5.1.38. Mon()

兼容：参见 aMonth。

许可

始终存在

5.1.39. parts()

返回包含对象的日历年、月、日、小时、分钟、秒和时区值的元组。

许可

始终存在

5.1.40. isCurrentYear()

如果这个对象在所属时区中表示一个属于当前年范围以内的日期时间对象，则返回真。

许可

始终存在

5.1.41. PreciseAMPM()

返回对象的时间字符串。

许可

始终存在

5.1.42. AMPMMinutes()

返回对象的时间字符串，不显示秒。

许可

始终存在

5.1.43. equalTo(t)

和另外一个 DateTime 对象或一个浮点数比较 DateTime 对象，比如和由 Python time 模块返回的数字进行比较。如果对象表示一个等于指定的 DateTime 或 time 模块风格时间的日期时间，则返回真。通过比较长整数型毫秒，它可以给出更为精确的结果。

许可

始终存在

5.1.44. pDay()

返回星期的简短名称（带有句点）。

许可

始终存在

第二十六章 页面模板参考

Zope 页面模板用于生成 HTML/XML。这个参考主要讲述：Template Attribute Language (TAL)（模板属性语言），TAL Expression Syntax (TALES)（TAL 表达式句法），Macro Expansion TAL (METAL)（宏扩展 TAL）。另外，还讲述了其它一些相关信息。

1. TAL 概述

模板属性语言（TAL）标准是一种属性语言，用于创建动态模板。它可以实现文档元素的替换、重复或忽略。

TAL 中的语句是 XML 属性。这些属性可以在 XMl 或 HTML 中使用，从而实现页面模板的功能。

TAL 语句由属性和数值组成。例如，content 语句可以这样 tal:content="string:Hello"。大多数的语句需要表达式，但表达式句法并不属于 TAL，而是属于 TALES。

1.1. TAL 名称空间

TAL 名称空间 URI 定义为：

xmlns:tal="http://xml.zope.org/namespaces/tal"

当在 Zope 中创建模板时，不需要用内容类型 text/html 声明 XML 名称空间，对于其它内容类型时才需要声明。

1.2. TAL 语句

TAL 语句包括：

tal:attributes - 动态更改元素属性。

tal:define - 定义变量。

tal:condition - 测试条件。

tal:content - 替换元素中的内容。

tal:omit-tag - 忽略一个元素，保留元素内容。

tal:on-error - 处理错误。

tal:repeat - 重复一个元素。

tal:replace - 替换元素的内容，删除元素，保留内容。

语句中的表达式可以返回任何类型的数值，大多数语句只使用字符型数据。表达式语言必须定义一个名为 nothing 的数值，它不是字符串。这个数值适用于删除元素或属性。

1.3. 执行顺序

如果每个元素中只有一个 TAL 语句，执行的顺序很简单。从根元素开始，顺序执行每个元素语句。

在同一元素中可以结合多个语句。但 content 和 replace 不能一起使用。

由于 TAL 把语句看作是 XML 属性，即使在 HTML 文档中，不能按照编写的顺序来决定执行的顺序。因此需要对执行的顺序进行规定。

当一个元素中含有多个语句时，按照以下顺序执行：

define

condition

repeat

content or replace

attributes

omit-tag

由于 on-error 语句只有发生错误时才用到，因此没有在此列出来。

按照这样的顺序执行，原因是：由于其它语句会用到变量，因此 define 先执行。接下来需要判断是否需要使用当前的元素，因此接下来是 condition；另外 condition 依赖于刚才定义的变量，所以 define 后边是 condition。由于通过循环的方式用不同值替换某些元素更具有价值，因此接下来是 repeat。由于替换属性没有更多的意义，因此放在后边。

1.4. 参见

TALES 概述

METAL 概述

tal:attributes

tal:define

tal:condition

tal:content

tal:omit-tag

tal:on-error

tal:repeat

tal:replace

2. attributes: 替换元素属性

2.1. 句法

tal:attributes 句法:

```
argument             ::= attribute_statement [';' attribute_statement]*

attribute_statement  ::= attribute_name expression

attribute_name       ::= [namespace-prefix ':'] Name

namespace-prefix     ::= Name
```

*注意：如果你想在表达式中包含分号(;)，必须通过使用两个符号(;;)来使用。

2.2. 描述

tal:attributes 语句用一个动态数值替换（或创建一个属性）属性值。如果你通过多个名称空间生成 XML 文档，需要在属性中加上名称空间前缀，比如 html:table。如果需要 的话，每个表达式的值会转换成字符串。

如果表达式结果为 nothing，那么就从语句元素中删除。如果表达式结果为默认值，则属性保持不变。每个属性是独立的，因此可以在同一语句中处理多个属性。

如果在一个元素中使用了 tal:attributes 语句，还使用了 tal:replace，则 tal:attributes 将忽略。

如果同时使用了 tal:attributes 和 tal:repeat 语句，每次循环都进行属性替换。

2.3. 例子

替换一个链接：

```
<a href="/sample/link.html"
 tal:attributes="href here/sub/absolute_url">
```

替换两个属性：

```
<textarea rows="80" cols="20"
 tal:attributes="rows request/rows;cols request/cols">
```

3. condition: 根据条件插入或删除元素

3.1. 句法

tal:condition 句法:

```
argument ::= expression
```

3.2. 描述

根据 tal:condition 语句，如果符合条件，则在模板中插入元素，否则忽略。如果它的表达式求值为真，则继续处理元素，否则立即从模板中删除。nothing 为假，default 等同于返回真。

注意：Zope 认为缺少变量，None，0，空字符串和空序列为假，其它的值为真。

3.3. 例子

在插入变量前先测试（第一个例子测试是否存在并求值，第二个例子只测试是否存在）：

```
<p tal:condition="request/message | nothing"
 tal:content="request/message">message goes here</p>
<p tal:condition="exists:request/message"
 tal:content="request/message">message goes here</p>
```

测试多个条件：

```
<div tal:repeat="item python:range(10)">
<p tal:condition="repeat/item/even">Even</p>
<p tal:condition="repeat/item/odd">Odd</p>
</div>
```

4. content: 替换元素内容

4.1. 句法

tal:content 句法:

```
argument ::= (['text'] | 'structure') expression
```

4.2. 描述

不是替换整个元素，而是在元素内容部分用 tal:content 替换，插入文本或结构(structure)。这个语句的参数和 tal:replace 相同。如果表达式结果为 nothing，内容部分为空。如果表达式结果为默认值，则内容部分保持不变。

默认的替换方式是文本替换。通过使用 structure 关键字，可以保持文本不变，即可以插入 HTML/XML 标记符。

4.3. 例子

插入用户名称：

```
<p tal:content="user/getUserName">Fred Farkas</p>
```

插入 HTML/XML：

```
<p tal:content="structure here/getStory">marked <b>up</b>

content goes here.</p>
```

4.4. 参见

tal:replace

5. define: 定义变量

5.1. 句法

tal:define 句法:

```
argument       ::= define_scope [';' define_scope]*

define_scope   ::= (['local'] | 'global') define_var

define_var     ::= variable_name expression

variable_name  ::= Name
```

*注意：如果你想在表达式中包含分号(;)，必须使用两个分号(;;)。

5.2. 描述

tal:define 用于定义变量。你可以定义两种类型的 TAL 变量：本地变量和全局变量。当你在一个元素中定义了一个本地变量则只能在那个元素和所包含的元素中使用。如果在所包含的元素中重新定义本地变量，那么新的定义替换原来的定义。当你定义了一个全局变量，就可以在定义以后在任何位置使用变量。如果你重新定义了全局变量，从重新定义的位置开始有效。

注意：默认为本地变量。

如果表达式求值结果为 nothing，那么变量的值为 nothing。如果求值结果为默认值，则变量的值为默认值。

5.3. 例子

定义一个全局变量：

```
tal:define="global company_name string:Zope Corp, Inc."
```

定义两个变量，第二个变量依赖于第一个:

```
tal:define="mytitle template/title; tlen python:len(mytitle)"
```

6. omit-tag: 删除元素，保留内容

6.1. 句法

tal:omit-tag 句法:

argument ::= [expression]

6.2. 描述

tal:omit-tag 语句保留元素中的内容，同时删除起始和结尾部分的标记符。

如果表达式求值结果为假，则继续处理，标记符不删除。如果表达式求值结果为真，或没有提供表达式，则替换元素中的内容。

Zope 认为空字符串、空序列、0、None 和 nothing 为假，其它值为真，包括默认值。

6.3. 例子

无条件忽略一个标记符：

```
<div tal:omit-tag="" comment="This tag will be removed">
  <i>...but this text will remain.</i>
</div>
```

有条件忽略一个标记符：

```
<b tal:omit-tag="not:bold">I may be bold.</b>
```

上边的例子中，如果变量 bold 为假，则忽略 b 标记符。

创建 10 个 p 标记符，没有合拢标记符：

```
<span tal:repeat="n python:range(10)"
      tal:omit-tag="">
  <p tal:content="n">1</p>
</span>
```

7. on-error: 处理错误

7.1. 句法

tal:on-error 句法:

argument ::= (['text'] | 'structure') expression

7.2. 描述

tal:on-error 语句用于处理错误。当一个 TAL 语句产生错误时，会搜索 tal:on-error 语句，调用第一个找到的 tal:on-error。使用方法如同 tal:content。

本地变量 error 有三个属性：

type

例外的类型

value

例外实例

traceback

回溯对象

常见的错误是文字错误或表达式为 nothing。更复杂一点的处理错误的方式是调用一个脚本。

7.3. 例子

显示简单的错误消息：

```
<b tal:on-error="string: Username is not defined!"
 tal:content="here/getUsername">Ishmael</b>
```

删除元素：

```
<b tal:on-error="nothing"
   tal:content="here/getUsername">Ishmael</b>
```

调用一个脚本：

```
<div tal:on-error="structure here/errorScript">
```

```
  ...
</div>
```

处理错误的脚本可以这样：

```
## Script (Python) "errHandler"
##bind namespace=_
##
error=_['error']
if error.type==ZeroDivisionError:
    return "<p>Can't divide by zero.</p>"
else
    return """<p>An error ocurred.</p>
              <p>Error type: %s</p>
              <p>Error value: %s</p>""" % (error.type,
                                           error.value)
```

7.4. 参见

Python Tutorial: Errors and Exceptions

Python Built-in Exceptions

8. repeat: 重复元素

8.1. 句法

tal:repeat 句法:

```
argument      ::= variable_name expression
variable_name ::= Name
```

8.2. 描述

tal:repeat 语句根据序列中的每个数据项重复处理元素。表达式求值结果应该为一个序列。如果序列为空，那么这个语句所在的元素被删除，否则重复处理序列中的每一项。如果表达式为默认值，那么不发生改变，并且没有新定义的变量。

8.2.1. 循环变量

使用循环变量可访问当前循环相关信息，比如循环的序号。循环变量和本地变量一样，但是只能通过内建的变量 repeat 来调用。

循环变量包括以下信息：

index - 循环的序号，从 0 开始。

number - 循环的序号，从 1 开始。

even - 对于偶数项为真(0, 2, 4, ...)。

odd - 对于奇数项为真(1, 3, 5, ...)。

start - 对于起始循环为真(index 0)。

end - 对于最后的循环为真。

first - 对于组中第一个数据项为真。参见下面的注意。

last - 对于组中的最后一个数据项为真。参见下面的注意。

length - 序列的长度，即循环总数。

letter - 用小写字母表示的循环序号。比如"a" - "z", "aa" - "az", "ba" - "bz", ..., "za" - "zz", "aaa" - "aaz",等等

Letter - 用大写字母表示的循环序号。

roman - 以小写罗马数字表示的循环序号，比如："i", "ii", "iii", "iv", "v", 等等。

Roman - 以大写罗马数字表示的循环序号。

你可以通过路径表达式或 Python 表达式访问循环变量。在路径表达式中，需要用三部分组成名称，即 repeat、语句变量名称和要访问的变量名称。比如，repeat/item/start。在 Python 表达式中，使用字典方法来访问信息，比如，"python:repeat['item'].start"。

除了 start、end 和 index，所有 repeat 变量的属性都是方法。因此，当你用 Python 表达式时，必须这样来调用："python:repeat['item'].length()"。

注意，first 和 last 适用于过滤序列。它们试图根据相同值把序列分成组。如果提供了一个路径，那么根据这个路径取得的数值进行分组，否则使用数据项的值。你可以通过传递参数提供路径，比如"python:repeat['item'].first(color)"，或通过 repeat 变量附加路径，比如，"repeat/item/first/color"。

8.3. 例子

对字符串序列进行循环：

```
<p tal:repeat="txt python:'one', 'two', 'three'">
   <span tal:replace="txt" />
</p>
```

插入表格行，使用变量 repeat 显示行号：

```
<table>
  <tr tal:repeat="item here/cart">
      <td tal:content="repeat/item/number">1</td>
      <td tal:content="item/description">Widget</td>
      <td tal:content="item/price">$1.50</td>
  </tr>
</table>
```

嵌套的循环：

```
<table border="1">
  <tr tal:repeat="row python:range(10)">
    <td tal:repeat="column python:range(10)">
      <span tal:define="x repeat/row/number;
                        y repeat/column/number;
```

```
                        z python:x*y"

            tal:replace="string:$x * $y = $z">1 * 1 = 1</span>

    </td>

  </tr>

</table>
```

插入对象。按照 meta-type 进行分组。

```
<div tal:repeat="object objects">

  <h2 tal:condition="repeat/object/first/meta_type"

    tal:content="object/meta_type">Meta Type</h2>

  <p tal:content="object/getId">Object ID</p>

  <hr tal:condition="repeat/object/last/meta_type" />

</div>
```

注意，上边例子中的对象应该已经按照 meta-type 进行了排序。

9. replace: 替换元素

9.1. 句法

tal:replace 句法:

```
argument ::= (['text'] | 'structure') expression
```

9.2. 描述

tal:replace 语句的作用是用动态内容替换一个元素。替换的内容可以是文本或结构。表达式中可加上类型前缀。表达式结果会自动转换成字符类型。如果使用了前缀 structure，则会在输出文本中保持特殊字符不变，即把 "&" 转换成 "&", 把"<" 转换成 "<", 把">" 转换成 ">"。

如果为 nothing，则只删除元素。如果为默认值，元素保持不变。

9.3. 例子

插入模板标题的两种方式：

```
<span tal:replace="template/title">Title</span>
```

```
<span tal:replace="text template/title">Title</span>
```

插入 HTML/XML：

```
<div tal:replace="structure table" />
```

插入 nothing（空）：

```
<div tal:replace="nothing">This element is a comment.</div>
```

9.4. 参见

tal:content

10. TALES 概述

模板属性语言表达式句法（TALES）描述了用于 TAL 和 METAL 的表达式。TALES 提供了多种表达类型。

以下是基本的 TALES 句法：

```
Expression  ::= [type_prefix ':'] String
```

```
type_prefix ::= Name
```

以下是一些例子：

```
a/b/c path:a/b/c nothing path:nothing python: 1 + 2 string:Hello, ${user/getUserName}
```

表达式前边可加上可选的类型前缀。如果不指定一个前缀，默认为路径表达式。

10.1. TALES 表达式类型：

path 表达式 - 根据路径确定数值。

exists 表达式 - 测试路径是否有效。

nocall 表达式 - 根据路径定位一个对象。

not 表达式 - 逻辑非表达式

string 表达式 - 字符串格式

python 表达式 - 运行 Python 表达式

10.2. 内建名称

TALES 表达式中可使用以下名称：

nothing - 用于表示空值的特殊值。

default - 用于指定不可替换的文本。

options - 传递给模板的关键字参数。通常出现在通过方法和脚本调用模板的时候。

repeat - tal:repeat 语句中的 repeat 变量。

attrs - 包含当前语句初始属性值的字典。

CONTEXTS - 标准名称的列表。用于访问隐藏的内建变量。

root - 系统中最顶级的对象：根文件夹。

here - 应用模板的对象。

container - 模板所属的文件夹。

template - 模板本身。

request - 请求对象。

user - 已验证用户对象。

modules - 可以访问的模块。

注意 root, here, container, template, request, user, 和 modules 是可选的名称，TALES 标准中不要求必须使用。

10.3. 参见

TAL Overview

METAL Overview

exists expressions

nocall expressions

not expressions

string expressions

path expressions

python expressions

11. TALES Exists 表达式

11.1. 句法

Exists 表达式句法:

exists_expressions ::= 'exists:' path_expression

11.2. 描述

Exists 表达式测试路径是否存在。如果路径存在返回真。当不能定位一个对象时，返回假。

11.3. 例子

测试一个表单变量：

```
<p tal:condition="not:exists:request/form/number">
  Please enter a number between 0 and 5
</p>
```

注意，此时不能使用 not:request/form/number，这是由于如果变量 number 存在并为 0，表达式值将为真。

12. TALES Nocall 表达式

12.1. 句法

Nocall 表达式句法:

nocall_expression ::= 'nocall:' path_expression

12.2. 描述

Nocall 表达式避免了调用路径表达式的结果。

通常路径表达式会调用对象。也就是说，如果对象是函数、脚本方法或其它可执行对象，则表达式会调用这些对象运行的结果。大多数情况下需要这样，但不总需要这样。例如，如果你想通过一个变量使用 DTML 文档，然后调用它的属性，此时就不能使用通常的路径表达式，否则会得到文档运行后的字符串。

12.3. 例子

使用 nocall 得到文档属性:

```
<span tal:define="doc nocall:here/aDoc"

      tal:content="string:${doc/getId}: ${doc/title}">

Id: Title</span>
```

对一个函数使用 nocall 表达式:

```
<p tal:define="join nocall:modules/string/join">
```

这个例子定义了一个变量 join，它绑定给 string.join 函数。

13. TALES Not 表达式

13.1. 句法

Not 表达式句法:

```
not_expression ::= 'not:' expression
```

13.2. 描述

Not 表达式先对 expression 字符串求值，然后返回它的布尔中的逻辑非值。如果提供的表达式求值后不是一个布尔值，则会根据以下规则转换成布尔类型：

数字 0 为假

正数和负数为真

空的字符串或其它序列为假

非空的字符串或其它序列为真

空值为假，比如：void, None, Nil, NULL 等等。

其它值为真。

如果没有提供 expression 字符串，会引发错误。

13.3. 例子

测试一个序列：

```
<p tal:condition="not:here/objectIds">

  There are no contained objects.

</p>
```

14. TALES Path 表达式

14.1. 句法

Path 表达式句法:

```
PathExpr    ::= Path [ '|' Expression ]

Path        ::= variable [ '/' PathSegment ]*

variable    ::= Name

PathSegment ::= ( '?' variable ) | PathChar+

PathChar    ::= AlphaNumeric | ' ' | '_' | '-' | '.' | ',' | '~'
```

14.2. 描述

路径表达式由一个路径组成，也可以在后边通过竖线符号(|)增加另外一个表达式，如果前一个表达式无效则使用后一个。路径由一个或多个非空字符串组成，用/分开。第一个字符串必须为一个变量名称（内建变量或用户定义的变量），其余部分可以包含字母、数字、空格和符号"_","-",".","~"。

表达式中可以使用符号?表示一个动态变量，这个变量必须是一个字符串，运行时会替换成对应的字符串。

例如：

```
request/cookies/oatmeal
```

nothing

here/some-file 2001_02.html.tar.gz/foo

root/to/branch | default

request/name | string:Anonymous Coward

here/?tname/macros/?mname

当对表达式求值时，从左到右依次处理路径。

如果出现错误，显示错误信息。

附加的表达式可以是任何 TALES 表达式。比如，request/name | string:Anonymous Coward，就是一个有效的路径表达式。通过这个表达式，可以提供默认值。也可以提供多个附加的路径表达式，比如，first | second | third | nothing。

如果没有路径，则为 nothing。

由于每个路径必须以一个变量名称开始，因此这个起始变量应该能够找到相应的对象才行。参见前边列出的内建变量。变量名称先在本地搜索，然后在内建变量列表中搜索，因此内建变量就像 Python 中内建的变量一样。也可以在变量名称前声明范围。你也可以直接通过 CONTEXTS 访问内建变量，比如，CONTEXTS/root, CONTEXTS/nothing。

14.3. 例子

插入一个 cookie 变量或一个属性：

```
<span tal:replace="request/cookies/pref | here/pref">

  preference

</span>
```

插入用户名称：

```
<p tal:content="user/getUserName">

  User name

</p>
```

15. TALES Python 表达式

15.1. 句法

Python 表达式句法:

任何有效的 Python 语言表达式

15.2. 描述

Python 表达式在安全限制环境内对 Python 代码求值。Python 表达式和脚本对象和 DTML 中的变量表达式一样。

15.2.1. 安全限制

Python 表达式的限制和脚本对象中的一样，这些限制包括：

访问限制

Python 表达式受到 Zope 许可和角色安全的限制。另外，表达式还不能访问那些名称用下划线开头的对象。

写入限制

Python 表达式不能更改 Zope 对象属性。

15.2.2. 内建函数

Python 表达式和脚本对象中的内建函数一样，但增加了一些。

标准的内建函数包括: None, abs, apply, callable, chr, cmp, complex, delattr, divmod, filter, float, getattr, hash, hex, int, isinstance, issubclass, list, len, long, map, max, min, oct, ord, repr, round, setattr, str, tuple.

range 和 pow 函数不能生成非常巨大的数值和序列。

另外，还可以使用： DateTime ,test,same_type。参考 DTML 函数部分。

最后，可以在 Python 表达式中使用这些函数：

path(string)

对一个 TALES path 表达式求值。

string(string)

对一个 TALES string 表达式求值。

exists(string)

对一个 TALES exists 表达式求值。

nocall(string)

对一个 TALES nocall 表达式求值。

15.2.3. Python 模块

可以使用默认提高的模块，也可以增加模块。访问模块的方式可以通过路径表达式，比如：modules/string/join，也可以通过 Python 语句映射对象，比如：modules string .join。以下是默认的模块：

string

string 模块。注意，模块中的大多数函数也可以通过字符串对象的方法进行访问。

random

随机模块。

math

数学模块。

sequence

具有强大排序功能的模块。

Products. PythonScripts .standard

提供各种 HTML 格式处理函数。

ZTUtils

提供批块处理功能。

AccessControl

提供安全和权限检查功能。

15.2.4. 例子

使用一个模块，从列表中随机选择一个数据项：

```
<span tal:replace="python:modules['random'].choice(['one',
                    'two', 'three', 'four', 'five'])">
  a random number between one and five
</span>
```

字符串处理，把用户名变成大写格式：

```
<p tal:content="python:user.getUserName().capitalize()">
  User Name
</p>
```

数学处理，把图像大小转换成兆字节：

```
<p tal:content="python:image.getSize() / 1048576.0">
  12.2323
</p>
```

处理字符串格式，把浮点型格式为两个小数位：

```
<p tal:content="python:'%0.2f' % size">
  13.56
</p>
```

16. TALES String 表达式

16.1. 句法

String 表达式句法:

```
string_expression ::= ( plain_string | [ varsub ] )*
varsub            ::= ( '$' Path ) | ( '${' Path '}' )
plain_string      ::= ( '$$' | non_dollar )*
```

```
non_dollar        ::= any character except '$'
```

16.2. 描述

字符串表达式按照文本处理字符串。如果没有提供字符串，则为空。字符串中可以通过$name 或${path}格式使用变量，其中 name 为变量名称，path 为路径表达式。如果要插入$符号，需要使用$$。

16.3. 例子

处理字符串：

```
<span tal:replace="string:$this and $that">

  Spam and Eggs

</span>
```

使用路径：

```
<p tal:content="total: ${request/form/total}">

  total: 12

</p>
```

包含一个$符号:

```
<p tal:content="cost: $$$cost">

  cost: $42.00

</p>
```

17. METAL 概述

宏扩展模板属性语言（METAL）用于处理 HTML/XML 宏（Macro）。它可以结合 TAL 和 TALES 一起使用。

宏提供了一种共享模板数据的方式。如果宏发生了变化，那么相应的共享数据也会发生变化。另外，宏可以扩展，从而具有最终结果的样子，方便编辑处理。

17.1. METAL 名称空间

METAL 名称空间 URI 定义为：

xmlns:metal="http://xml.zope.org/namespaces/metal"

当创建内容类型具有 text/html 的模板时，Zope 不要求声明 XML 名称空间。但对于其它类型则需要声明。

17.2. METAL 语句

METAL 定义的语句包括：

metal:define-macro - 定义一个宏。

metal:use-macro - 使用一个宏。

metal:define-slot - 定义一个宏定制点。

metal:fill-slot - 定制一个宏

参见

TAL Overview

TALES Overview

metal:define-macro

metal:use-macro

metal:define-slot

metal:fill-slot

18. define-macro: 定义一个宏

18.1. 句法

metal:define-macro 句法:

argument ::= Name

18.2. 描述

metal:define-macro 语句定义一个宏。通过语句表达式进行定义，有效返回是当前所在的元素和下级元素。

可以通过模板的 macros 对象调用宏定义。比如，在模板 master.html 中定义了宏 header，那么你可以使用路径表达式 master.html/macros/header 访问这个宏。

18.3. 例子

简单的宏定义：

```
<p metal:define-macro="copyright">
  Copyright 2001, <em>Foobar</em> Inc.
</p>
```

参见

metal:use-macro

metal:define-slot

19. define-slot: 定义一个宏定制点

19.1. 句法

metal:define-slot 句法:

```
argument ::= Name
```

19.2. 描述

metal:define-slot 语句定义了一个宏定制点，或内容块(slot)。当使用一个宏时，这个内容块可以替换成其它内容。内容块定义提供了默认的内容。如果不调用内容块，则显示默认的内容。

metal:define-slot 语句必须在 metal:define-macro 之中。

内容块的名称必须唯一。

19.3. 例子

简单的内容块定义：

```
<p metal:define-macro="hello">
  Hello <b metal:define-slot="name">World</b>
</p>
```

这个例子定义了一个宏和一个名为 name 的内容块。当你使用这个宏的时候，可以在 b 元素中添加进定制的内容。

19.4. 参见

metal:fill-slot

20. fill-slot: 定制一个宏

20.1. 句法

metal:fill-slot 句法:

argument ::= Name

20.2. 描述

metal:fill-slot 语句通过替换宏中的内容块来定制宏。

metal:fill-slot 语句必须在 metal:use-macro 中使用。

内容块名称必须唯一。

如果宏中不存在指定的内容块，则删除这个内容块。

20.3. 例子

先定义宏:

```
<p metal:define-macro="hello">
  Hello <b metal:define-slot="name">World</b>
</p>
```

你可以这样来填充内容块 name:

```
<p metal:use-macro="container/master.html/macros/hello">
  Hello <b metal:fill-slot="name">Kevin Bacon</b>
</p>
```

20.4. 参见

metal:define-slot

21. use-macro: 使用一个宏

21.1. 句法

metal:use-macro 句法:

argument ::= expression

21.2. 描述

metal:use-macro 语句用宏替换当前元素中的内容。expression 描述了一个宏定义。

expression 将生成一个路径表达式，用来指向另外一个模板中定义的宏。参见 metal:define-macro 部分。

扩展宏的效果就是从另外一个文档或当前文档中提取部分内容，然后放进当前元素之中，即替换当前的内容。原来的内容保持不变，如果使用了内容块，则内容块中使用新的内容。

当扩展宏时，使用 metal:use-macro 语句。

21.3. 例子

基本用法：

```
<p metal:use-macro="container/other.html/macros/header">

  header macro from defined in other.html template

</p>
```

这个例子引用了在 other.html 模板中定义的 header。当扩展这个宏时，p 元素和它的内容将替换成宏中的内容。

21.4. 参见

metal:define-macro

metal:fill-slot

22. ZPT 特定的行为

Zope 页面模板的行为几乎完全可以通过 TAL、TALES 和 METAL 进行描述。但还有一些特定的行为。

22.1. HTML 支持的特性

当一个页面模板的内容类型设置为 text/html，处理过程会有些不同于其它内容类型。在 TAL 名称空间中讲过，HTML 不需要声明名称空间，默认提供 TAL 和 METAL 名称空间。

HTML 文档采用非 XML 解析器，因此可以处理一些不符合 XML 要求的标记符。特别是那些没有结束标记符的元素，比如段落和列表，但还不能认为是错误，除非它们是语句元素。这样就容易导致错误，比如使用用 元素，但不嵌套在 中，此时由于 标记符，则会引发一个 NestingError 错误。因此解决的方法是使用。

没有合拢的语句元素通常认为是错误，因此应该尽量指明合拢的标记符。对于那些没有结束标记符的元素，比如 image 和 input 元素，不要求加上合拢标记符或使用 XHTML 中的<tag/>格式。

某些布尔属性，比如 checked 和 selected，在 tal:attributes 中处理方式不同。数值为真或假。如果为真，则会处理成 attr="attr"格式，如果为假，则会忽略。如果值为 default，则此时，如果属性已经存在认为是真，不存在为假。比如：

```
<input type="checkbox" checked tal:attributes="checked default">

<input type="checkbox" tal:attributes="checked string:yes">

<input type="checkbox" tal:attributes="checked python:42">
```

结果为:

```
<input type="checkbox" checked="checked">
```

对于:

```
<input type="checkbox" tal:attributes="checked default">

<input type="checkbox" tal:attributes="checked string:">

<input type="checkbox" tal:attributes="checked nothing">
```

将生成:

```
<input type="checkbox">
```

这种处理方式可以在所有的浏览器中正确显示。

第二十七章 DTML 名称搜索规则

在 DTML 中，这些规则用于解析 name= 和 expr= 中的名称。搜索路径中按照从开始到结束的顺序应用这些规则。

DTML 中调用的方式如下：

```
def __call__(client=None, mapping={}, **kw)
```

client 参数一般不会在 DTML 中引用，通常是指被调用的方法所在的环境（比如，最简单的情况是指 DTML 所在的文件夹）。

参数 mapping 通常在 DTML 中使用 _ 符号表示。

关键字参数，即**kw，相应于 DTML 中的名称。

搜索关键字参数。

搜索 mapping 对象。

搜索 client 属性，包括继承的和获取的属性。

如果在 DTML 方法或文档中使用 DTML，并且变量名称为 document_id 或 document_title，那么使用这个文档或方法的 id 或 title。

1. 搜索包含这个 DTML 对象的文件夹的属性。这些属性包括文件夹的内容、属性和其它定义的属性，比如 ZopeTime。文件夹属性包括上级文件夹的属性。

搜索用户定义的 web 请求变量，即 REQUEST.other 名称空间。

搜索表单定义的 web 请求变量，即 REQUEST.form 名称空间。

搜索 cookie 定义的 web 请求变量，即 REQUEST.cookie 名称空间。

搜索 CGI 定义的 web 请求变量，即 REQUEST.environ 名称空间。