PERL 编程思想

应用篇

罗刚 编著

2004

目录

第1章 CGI 技术

本章将举例应用 Perl 的 CGI 方法做简单的页面。讲解使用 CGI.pm 模块等。

1.1 什么是 CGI?

CGI 是 Common Gateway Interface 的缩写。CGI 是 Web 服务器和 CGI 程序之间交流的标准方法。几乎所有的 WEB 服务器都支持 CGI 程序的运行。

CGI 在表现形式上是放置在服务器中的一段程序，WEB 服务器有选择性的将浏览器的请求转发给 CGI 程序，经过 CGI 程序的处理后将一些结果再通过 WEB 服务器反馈给浏览器。

1.2 CGI 技术原理

CGI 技术定义了一组标准的环境变量，这些环境变量存储着服务器与客户端的各种信息，提供给 CGI 程序作为参考。这其中就包括客户端发来的各种数据。Perl 支持这些环境变量，并且很方便的将他们放到一个名为 ENV 的 Hash 表中。

关于 CGI 的所有环境变量列表请见如下表格：

环境变量	意义
CONTENT_TYPE	如果表单是用 POST 递交，这个值将是 application/x-www-form-urlencoded. 在上载文件的表单中，content-type 是个 multipart/form-data。
CONTENT_LENGTH	对于用 POST 递交的表单, 标准输入口的字节数。
DOCUMENT_ROOT	你的服务器的根路径。
GATEWAY_INTERFACE	运行的 CGI 版本. 对于 UNIX 服务器, 这是 CGI/1.1。
HTTP_ACCEPT	浏览器能直接接收的 Content-types, 可以有 HTTP Accept header 定义。
HTTP_USER_AGENT	递交表单的浏览器的名称、版本 和其他平台性的附加信息。
HTTP_REFERER	递交表单的文本的 URL，不是所有的浏览器都发出这个信息，不要依赖它。

环境变量	意义
PATH_INFO	附加的路径信息, 由浏览器通过 GET 方法发出。
QUERY_STRING	脚本参数或者表单输入项(如果是用 GET 递交). QUERY_STRING 包含 URL 中问号后面的参数。
REQUEST_METHOD	POST 或 GET, 取决于你的表单是怎样递交的。
REMOTE_HOST	递交脚本的主机名，这个值不能被设置。
REMOTE_ADDR	递交脚本的主机 IP 地址。
REMOTE_USER	递交脚本的用户名. 如果服务器的 authentication 被激活，这个值可以设置。
SCRIPT_NAME	指向这个 CGI 脚本的路径，是在 URL 中显示的(如, /cgi-bin/thescript)。
SERVER_PROTOCOL	服务器运行的 HTTP 协议。一般是 HTTP/1.1。
SERVER_PORT	服务器运行的 TCP 端口，通常 Web 服务器是 80。
SERVER_NAME	CGI 脚本运行时的主机名和 IP 地址。
SERVER_SOFTWARE	你的服务器的类型如：Apache/1.3.28 (Unix)。

1.3 Apache 简介

Apache 最初作为美国国家超级计算应用中心(NCSA)的一个 httpd，即 web 服务器。现在它已经成为 internet 上最流行的 web 服务器。据统计，60%以上的网站都使用 Apache。

因为它是开放源码的，所以可以从它的网站（http://www.apache.org）上下载到最新的安装程序和源代码。

Apache 的工作原理是这样的：启动时首先读入配置文件 (httpd.conf) ，然后监听 80 端口(或其他的端口)，立刻改变到 user nobody，产生许多进程(可能是 10 个，50 个，150 个或更多)为请求服务，一个子进程的生命周期可能限制到只服务一定数量的请求，在每个请求后都记录日志。

Apache 从一开始设计的时候就是模块化的,你可以写一段代码很容易的集成进 Apache。设计者把服务器的运行定义成了各个阶段，每个阶段都有 API 接口。这些阶段包括服务器初始化（读配置文件），翻译请求的 URL 到服务器上的文件名，为整个过程记日志等。有很多这样的模块，当前有上百个，大部分都可以在 modules.apache.org 找到。

Apache 的模块性可能使配置过程变得很复杂。但是 Apache 缺省的带了一些有用的模块 ，而且大部分常见的使用都支持。

1.3.1 在 Unix 下的安装

首先下载源文件 apache_1.3.28.tar.gz。然后编译 apache。

1．解压

#gzip -dc apache_1.3. 28.tar.gz | tar xvf –

2．配置

#cd apache_1.3. 28

#./configure --prefix=/ur12/power/lg/ap --enable-module=so --enable-module=cgi

3．编译

#make

4．安装

#make install

5．运行 apache 的 httpd 服务：

#/ur12/power/lg/ap/bin/apachectl start

缺省时服务将监听端口 8080。

6．然后测试一下 cgi 程序：

http://130.59.1.15:8080/cgi-bin/printenv

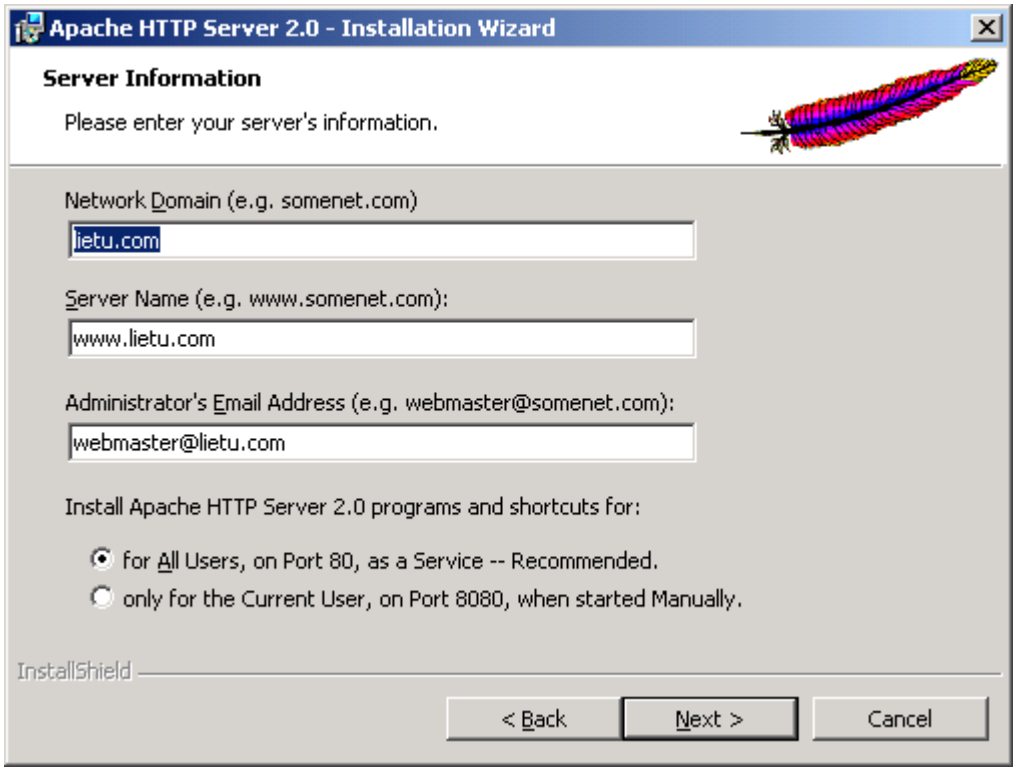
7．停止服务：

#/ur12/power/lg/ap/bin/apachectl stop

1.3.2 在 windows 下的安装

安装之前需要注意的是，最好检查一下，是否已经存在 Apache2 或 Apache 这样的目录。

如果想要安装一个全新的版本，需要删除或者备份已有的目录。其它一般采用缺省安装即可。



这些配置信息写在 C:\Program Files\Apache Group\Apache2\conf\httpd.default.conf 文件中。安装之后也可以修改相关信息。

服务运行后，从服务器本机的浏览器上运行：

http://localhost/cgi-bin/printenv.pl

如果能够看到正常的环境变量输出就说明 CGI 程序可以正常运行。

1.4 让你的程序以 CGI 方式运行

我们的第一个 CGI 脚本不应该是这样的：

```
#!c:/Perl/bin/Perl.exe


# 不要使用这个脚本!
# 应该使用 -w 选项, use strict,
```

```
# 最好加上 -T 选项


print "Content-type: text/html;charset=gb2312\n\n";

print "<html><head><title>第一个 CGI 脚本</title></head>";

print "<body bgcolor=\"#ffffcc\">";

print "<h1>不要学这个例子</h1>";

print "<p>";

print "Hello world";

print "</body></html>";
```

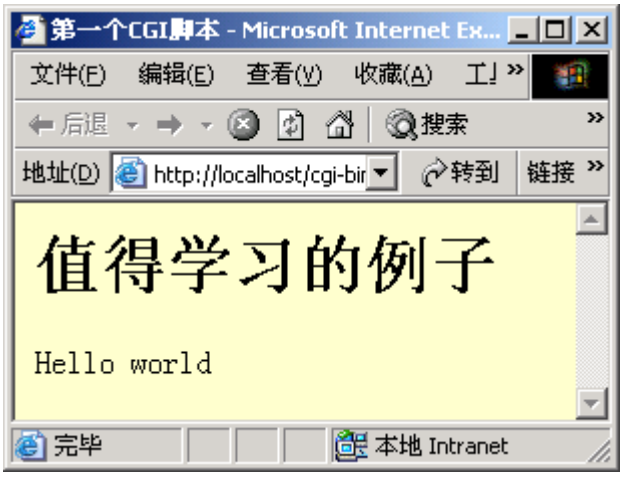
上面这个例子以最直接的方式打印出了要生成的 html 网页，而且通过“Content-type: text/html;charset=gb2312”告诉浏览器这是中文页面。

我们要学的是下面这个使用模块生成页面的例子。

```
#!c:/Perl/bin/Perl.exe -wT

use strict;

use CGI;


my $query = CGI->new();


print $query->header( "text/html;charset=gb2312" ),

      $query->start_html(-title      => "第一个 CGI 脚本",

                                 -bgcolor => "#ffffcc" ),

      $query->h1( "值得学习的例子" ),

      $query->p( "Hello world" ),
```

$query->end_html;

从浏览器看起来是这样的：



1.5 使用 CGI 模块

使用 CGI.pm 可以简化 HTML 代码生成。你可以通过函数调用的方式来保证生成正确的 HTML 语法。CGI.pm 同时支持用函数或者面向对象风格来使用这个包。

函数风格的用法如下：

```
#!/usr/bin/perl



use CGI qw/:standard/;



print header(),

start_html(-title=>"CGI.pm Example Script"),

h1('Hello World!'),

'Hey, this is cool!',

end_html();
```

面向对象的用法如下：

```
#!/usr/bin/perl

use CGI;


$query = new CGI;


print $query->header(),

$query->start_html(-title=>"CGI.pm Example Script"),

$query->h1('Hello World!'),

'Hey, this is cool!',

$query->end_html();
```

如果你想使用函数式的风格，可以在面向对象的代码上改动：

- 把 use CGI; 改成 use CGI qw(:standard); 。它将引起 CGI.pm 输出所有的标准方法，可以直接调用。
- 你不需要实例化一个新的 CGI 对象(即，去掉$query = new CGI;)。
- 删除掉$some_variable_name-> 部分(在这里是$query->)。

1.5.1 支持中文

CGI.pm 提供的缺省字符集都是 ISO-8859-1，为了使它支持中文，必须在头指定字符集。

```
print header(-charset=>'gb2312');


print start_html(-title=>'Untitled Document',

        -encoding=>'gb2312',
```

```
-lang=>'zh-cn',

-head =>

meta({

-http_equiv => 'Content-Type',

-content => 'text/html; charset=gb2312'}));
```

1.5.2 处理 GET 和 POST 请求

有两种基本的请求需要处理：GET 和 POST。当浏览器请求一个页面时，它发送下面这样的 GET 请求：

```
GET /index.html HTTP/1.1

Host: www.somedomain.com

...
```

Web 服务器接受 GET 请求，如果找到了就会返回对应的文档。如果没有找到该文档，就返回我们司空见惯的"404 Not Found error"错误。当你看见下面这样的 URL 时：

```
http://www.somedomain.com/cgi-bin/somescript.cgi?name=BillGates&position=monopolist
```

从前向后分析，我们看见了主机名: www.somedomain.com，路径：/cgi-bin/，和一个cgi脚本的名字，在这里是somescript.cgi。继续往前就是一个查询字符串(在Perl程序中，可以通过$ENV{'QUERY_STRING'}获取)。"name=BillGates"和"position=monopolist"指名字和值对，它们通过地址符分开(&)。这些名字/值对可供CGI脚本决定如何响应浏览器的响应。值得一提的是查询字符串的长度是有限制的，最大值依据Web服务器而定。

对于一个 POST 请求，浏览器将对名称/值对和在头部的一些信息分解开，并把它放在请求的主体内。通常，这些额外的信息也是上诉那种名称/值对，但它是从标准输入(STDIN)读入，而不是在$ENV(‘QUERY_STRING’)中。如果 CGI 仅提供这些，那自己写一个应用将会很困难，好在，我们还有方法。

假设要上传一个文件，你会在 HTML 文件中<form>标记处用”multipart/form-data”来标识类型（这会在后面内容深入讨论）。由于很难将一整个文件放在一个请求字符串中，数据被按不同方式分割。示意起来比解释更容易明白些，考虑下面在 Web 页面中的表单：

```
<form action="/cgi-bin/display.cgi" method="post" enctype="multipart/form-data">
    <input type="file" name="file"><br />
    <input type="hidden" name="hiddenData" value="nothing to see here, folks"><br />
    <input type="checkbox" name="someCheckbox" value="someValue">Checkbox<br />
    <input type="submit"><br>
</form>
```

注意有 3 个名称和值被设置，”file”的值是选中文件的内容。在这个例子中，我选择一个名为”temp.pl”文件，我们只是用这个文件来调试问题（里面包含在下面显示的一段 Perl 代码）。我还选中了复选框（否则，就不会为它发送值）。下面是 CGI 脚本在标准输入获得的内容：

```
-----------------------------7d03ca41c0
Content-Disposition: form-data; name="file"; filename="C:\WINDOWS\Desktop\temp.pl"
Content-Type: text/plain
#!C:/perl/bin/perl.exe -w
use strict;
$SIG{INT} = 'IGNORE';
my $count=0;
while (){
    print $count++;
}
-----------------------------7d03ca41c0
Content-Disposition: form-data; name="hiddenData"
nothing to see here, folks
-----------------------------7d03ca41c0
```

Content-Disposition: form-data; name="someCheckbox"

someValue

----------7d03ca41c0--

This data has been sent in MIME format (Multi-Purpose Internet Mail Extensions)。

我们使用 param 方法来取得请求，下面这个脚本打印出所有的参数和值：

```
#!C:\perl\bin\perl.exe -wT
use strict;
use CGI;

my $query = CGI->new();
my @names = $query->param; # Yup.    That's all it takes to get all names
                           # from name/value pairs.

print $query->header( "text/plain" ); # This is just to show that we can
                                      # specify our Content-type

foreach my $name ( @names ) {
    # If we know that a particular name only has one value, we can use
    # my $value = $query->param( $name );
    # However, if we assign the param to a scalar and there are multiple
    # values, we'll only get the first one.
```

```
    my @values = $query->param( $name );

    print $name . "=" . (join ", ", @values) . "\n";

}
```

是不是发现很简单？CGI.pm 模块允许从命令行进行简单的调试。如果要测试它，从命令行拿除-T 参数（如果不这么作会产生错误），不要把它当作 CGI 脚本来运行，并且输入：

```
perl -T myscriptname.cgi name=Ovid color=red color=blue
```

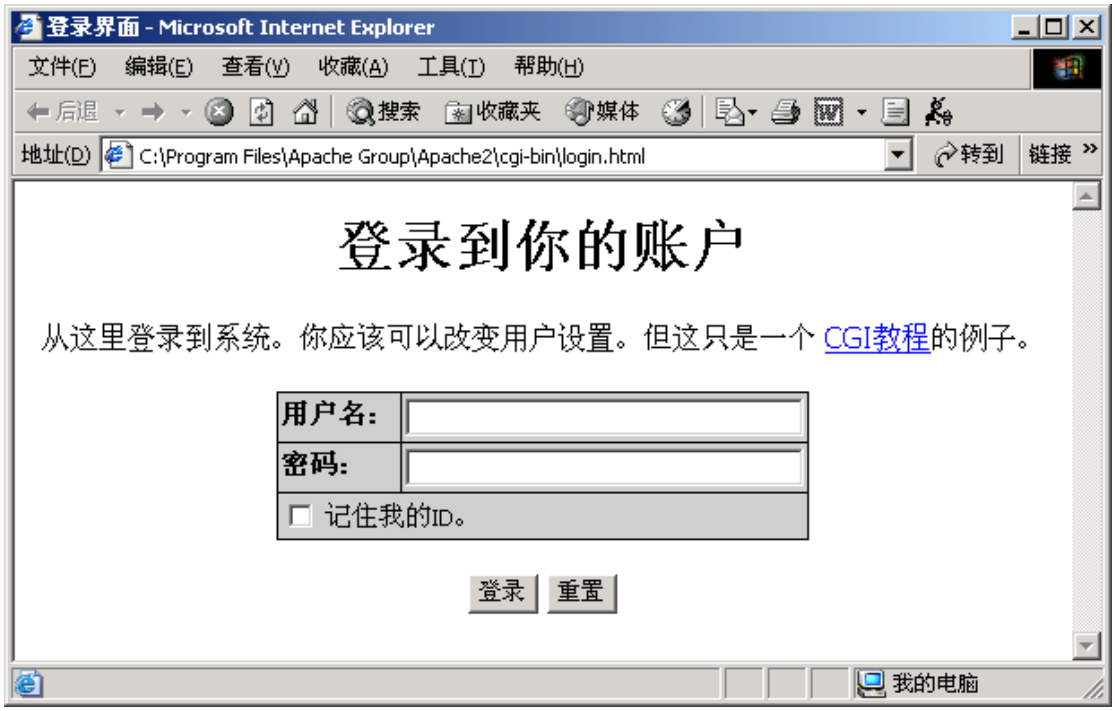
这会输出下面的结果：

```
Content-type: text/plain

name=Ovid

color=red, blue
```

1.6 创建登录页面

我们先从比较简单的地方开始。先来建立一个简单的登陆界面。用户输入登陆名、密码组合并有一个“记录我的 ID”的选项框，用户可以选中从而以后跳过登陆页。我们并不打算在这就实现实际登陆功能，所要作的是创建一段产生登陆页面的脚本，在以后我们会逐步来实现登陆功能。

你需要得到以下信息：登陆名、密码，以及一个复选框状态。这个页面看起来像下面所示：



html 文件的内容是这样的：

```
<html>
  <head>
    <title>登录界面</title>
  </head>
  <body style="background-color: #ffffff;">
    <div style="color:#000000; font-family: Tahoma, helvetica, arial;" align="center">
      <h1>登录到你的账户</h1>
      <p>从这里登录到系统。你应该可以改变用户设置。但这只是一个
      <a href="http://www.lietu.com/perl/">CGI 教程</a>的例子。</p>
      <form action="login.cgi" method="post" enctype="application/x-www-form-urlencoded">
        <table cellspacing="1" border="0" cellpadding="2" bgcolor="#000000" style="font: 10pt;">
          <tr style="background-color:#CCCCCC">
```

```
            <td><strong>用户名：</strong></td>
            <td><input type="text" size="30" maxlength="30" name="username"></td>
          </tr>
          <tr style="background-color:#CCCCCC">
            <td><strong>密码：</strong></td>
            <td><input type="password" size="30" maxlength="30" name="password"></td>
          </tr>
          <tr>
            <td colspan="2" style="background-color:#CCCCCC">
              <input type="checkbox" name="remember"> 记住我的 ID。
            </td>
          </tr>
        </table>
        <p><input type="submit" value="登录"> <input type="reset"></p>
      </form>
    </div>
  </body>
</html>
```

我们仔细来看一下 HTML 的内容，你或许对于这么多标签还不是很熟悉，让我们先来看一下<form>标签。

<form action="login.cgi" method="post" enctype="application/x-www-form-urlencoded">

<form>标签向浏览器标出这是用来向 Web 服务器端提交数据的表单。”action”属性告诉浏览器表单将提交给那个资源，可以是一个相对或者绝对链接，链接到 anchor 标签、img 标签或者向浏览器标识资源的任何一个标签。

“method”属性指定表单是用 POST 方法。对于基于网页的表单，通常调用 POST 或者 GET 方法。像在以前提到过的，GET 方法将在查询字符串（嵌入在 URL 中）发送表单的数据，而 POST 方法将在响应的报头之后的主体中发送。如果 POST 数据已经发送，表单接下来由”enctype”属性指定处理模块。

这个表单中“enctype”属性是” application/x-www-form-urlencoded”。对于这个”enctype”属性，它是 Web 表单的默认编码方式。如果 POST 方法中指定了编码类型，则在查询字符串中将采用指定方式编码（特别的名称/值对将被转为 16 进制，后面跟一个等号，每一对跟一个&符合）。这样，当头送出后，文件的主体拥有表单数据。还有一种常用的编码方式，是”multipart/form-data”形式。通常用在上传文件，因为文件数据编码比较麻烦些，也一般使得传送的数据膨胀。我们在稍后会讨论到文件的上传。

可以使用很多种方式使用 HTML 函数。首先看一个输入标签：

```
<input type="text" size="30" maxlength="30" name="username">
```

这个标签的名字是"input"，它有几个属性： "type"，"size"，"maxlength"和"name"。这些属性有一些对应的值相联系："text", "30", "30" and "username"。为了打印这个标签，只需调用标签的同名函数，使用属性和它们的值作为一个匿名的 hash 引用(一个由'{}'包含的列表)。属性名应该以一个横线开头：

```
print input( { -type => "text",   -size => "30", -maxlength => "30", -name => "username" } );
```

上面的代码使用了函数型的接口，面向对象的接口有一个对 CGI 对象的引用在$cgi 中，上面的代码变成了：

```
print $cgi->input( { -type => "text",  -size => "30", -maxlength => "30", -name => "username" } );
```

为了明了，我们仍然使用函数型的接口。现在让我们看一下段落标签：

```
<p>Are you still reading this?  Go out and play!</p>
```

生成这个标签相当简单：

```
print p( "Are you still reading this?  Go out and play!" );
```

好了，如果要打印一个有属性的段落标记怎么办呢？一个简单的规则是记住当你打开和关闭标签之间的文本，使用这些文本作为第二个参数，打开的标签的属性作为数组的第一个参数。例如，打印如下的标签：

```
<p class="notice">Are you still reading this?  Go out and play!</p>
```

使用下面的语法：

```
print p( { -class => "notice" }, "Are you still reading this?  Go out and play!" );
```

需要的时候也可以使用嵌套标签，如要创建这个：

```
<em><strong>Are you still reading this?  Go out and play!</strong></em>
```

可以使用这个：

```
print em( strong( "Are you still reading this?  Go out and play!" ) );
```

当然，超文本可能非常复杂。"td"标签嵌套在"tr"标签，而它们有嵌套在"table"标签，这些可能嵌套在"form"标签。这些关系可能难以追踪，关于这些函数的详情可以参考 CGI.pm 文档。需要仔细的阅读这些文档，如有些标签不是自关闭的(如"body"和"form"标签)。有些标签有大写的函数名称以避免与 Perl 的内部函数名冲突(这些标签是 Select，Tr，Link，Delete，Accept 和 Sub)。

这里是打印登录界面的程序：

```
#!C:/perl/bin/perl.exe -wT

use strict;

use CGI qw/:standard/;



print header( "text/html;charset=gb2312" ),

      start_html( "-title" => "登录界面"),

      div( { -align => "center",

              -style => "color:#000000; font-family: Tahoma, helvetica, arial;"},

        h1( "登录到你的账户" ),

        p( "从这里登录到系统。你应该可以改变用户设置。但这只是一个" .

          a( { -href => "http://www.lietu.com/perl/" },

            "CGI 教程" ), "的例子。" ),
```

```
start_form( { -action    => "login.cgi",
                    -enctype => "application/x-www-form-urlencoded",
                    -method    => "post" } ),
table( { -bgcolor          => "#000000",
                -border           => "0",
                -cellpadding => "2",
                -cellspacing => "1",
                -style             => "font: 10pt;" },
   Tr( { -style => "background-color:#CCCCCC" },
      td( strong( "用户名：" ) ),
      td( input( { -maxlength => "30",
                          -name           => "username",
                          -size           => "30",
                          -type           => "text"} )
      ) # end td
   ), # end Tr
   Tr( { -style => "background-color:#CCCCCC"},
      td( strong( "密码：" ) ),
      td( input( { -maxlength => "30",
                          -name           => "password",
                          -size           => "30",
                          -type           => "password"} )
```

```
            ) # end td
        ), # end Tr
        Tr(
            td( { -colspan => "2",
                  -style      => "background-color:#CCCCCC" },
                input ( { -name => "remember",
                          -type => "checkbox"} ),
                "记住我的 ID。",
            ) # end td
        ), # end Tr
    ), # End table
    p( input( { -type    => "submit",
                -value => "登录"} ),
        " ",
        input( { -type    => "reset"} ),
    ), # end p
    end_form,
  ), # End div
end_html;
```

我们可以采用数据库保存用户密码，下面是实现登录检验的代码：

```
use DBI;
```

```
my $tainted_username = param( 'username' ) || '';

my $password = param( 'password' ) || '';

my $remember = param( 'remember' ) || '';



my $username = '';

my $message = 'Your username and password information did not match.  Check to see that you do not have  Caps Lock on, hit the back button, and try again.';



# Note the regular expression that states very explicitly what we will allow

if ( $tainted_username =~ /^([a-zA-Z\d_]+)$/ )

{

    $username = $1;

}

else

{

    display_page( $message );

    exit;

}



$dbh = DBI->connect( "dbi:mysql:usertable", "usertable", "jutedi2")

unless (defined($dbh))

{

    $message = "Can't connect to db: ".$dbh->errstr;
```

```
        display_page( $message );
        exit;
}

$sth = $dbh->prepare("select * from users where username = ?")
unless(defined($sth))
{
        $message = "Can't select from table: ".$dbh->errmsg;
        display_page( $message );
        exit;
}

$sth->execute($username);
$hashref = $sth->fetchrow_hashref;
%uinfo = %{$hashref};

if (!(scalar %uinfo))
{
        $message = "Can't find your username!?";
        display_page( $message );
        exit;
}
```

```
# now encrypt the old password and see if it matches what's in the database
if ($uinfo{password} ne crypt($oldpass,substr($uinfo{password},0,2)) )
{
    $message = "Can't find your username!?";
    display_page( $message );
    exit;
}
else
{
    $message = "Hello, $username.    You gave me a good password";
}
exit;

sub display_page
{
    my $message = shift;
    print
        header,
        start_html,
        p( $message ),
        end_html;
```

```
}
```

1.7 上传文件

如下讨论一个例子让用户上传多个文件到服务器上的指定的目录。它的界面如下面所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<form action="" method="post" enctype="multipart/form-data" name="form1">
  <table width="75%" border="0">
    <tr>
      <td width="32%"><div align="right">First Name: </div></td>
      <td width="68%"><input type="text" name="firstname"></td>
    </tr>
    <tr>
      <td><div align="right">Last Name:</div></td>
      <td><input type="text" name="lastname"></td>
    </tr>
```

```
      <tr>
         <td><div align="right">Email Address: </div></td>
         <td><input type="text" name="email"></td>
      </tr>
      <tr>
         <td><div align="right">File One: </div></td>
         <td><input type="file" name="fileone"></td>
      </tr>
      <tr>
         <td><div align="right">File Two: </div></td>
         <td><input type="file" name="filetwo"></td>
      </tr>
      <tr>
         <td><div align="right">File Three: </div></td>
         <td><input type="file" name="filethree"></td>
      </tr>
   </table>

   <br>
   <br>

</form>
```

```
</body>

</html>
```

让我们使用 CGI.pm 来实现它，根据要求，必须有错误检查函数，例如检查空值和 email 有效性检查。实现代码如下：

```
#!/ur12/power/Perl58/bin/perl



# Use JavaScript to validate fill-out

# forms.

use CGI qw(:standard);



# Here's the javascript code that we include in the document.

#must set $SAVE_DIRECTORY

$SAVE_DIRECTORY = "./upload_dir";

$JSCRIPT=<<EOF;

    // validate that the user is the right age.    Return

    // false to prevent the form from being submitted.

    function validateForm() {



    var ret = validateEmail(document.form1.email);

    if (! ret) {

        return false;

    }

    ret = checkFirstName(document.form1.firstname);
```

```
  if (! ret) {
        return false;
  }
  ret = checkLastName(document.form1.lastname);
  if (! ret) {
        return false;
  }
  document.form1.state = 'valid';
  return true;
  }


// make sure that the contents of the supplied
// field contain a valid date.
function validateEmail(element) {

      //alert(element.value);
      if (element.value.indexOf("@")<0 ) {
      alert("Please valid your e-mail address");
      element.focus();
      element.select();
      return false;
      }
```

```
        return true;
    }

    function checkFirstName(element) {
    if (element.value.length    == 0 ) {
        alert(element.name+"can not be null.");
        element.focus();
    }
    }

    function checkLastName(element) {
    if (element.value.length    == 0 ) {
        alert(element.name+"can not be null.");
        element.focus();
    }
    }
EOF
    ;

# here's where the execution begins
print header;
```

```
print start_html(-title=>'Untitled Document',

        -head =>

        meta({

        -http_equiv => 'Content-Type',

        -content => 'text/html; charset=iso-8859-1'}),

        -script=>$JSCRIPT);



print_prompt();

print_response() if (param);



print end_html;



sub print_prompt {



    print start_multipart_form(-name=>'form1',

                -onSubmit=>"validateForm()",

                -method=>"post"),"\n";

    print table({-border=>0,-width=>"75%"},"\n",

        Tr(td({-width=>"32%"},div({-align=>"right"},'First                                    Name:
')) ,td({-width=>"68%"},input({-type=>'text',-name=>'firstname'})) ),

        Tr(

        [

            td([div({-align=>"right"},'Last Name:'),input({-type=>'text',-name=>'lastname'})]),
```

```
            td([div({-align=>"right"},'Email Address: '),input({-type=>'text',-name=>'email'})]),
            td([div({-align=>"right"},'File One: '),input({-type=>'file',-name=>'fileone'})]),
            td([div({-align=>"right"},'File Two: '),input({-type=>'file',-name=>'filetwo'})]),
            td([div({-align=>"right"},'File Three: '),input({-type=>'file',-name=>'filethree'})])
        ])
    );
    print hidden('state','init');
    print submit(),"\n";
    print end_form;
}

sub print_response {
    return unless(param('firstname'));
    return unless(param('lastname'));
    $value = param('email');
    return unless($value =~/@/);

    #create diretory
    unless(-e $SAVE_DIRECTORY)
    {
        unless (mkdir("upload_dir",0777))
        {
```

```
            print "cannot mkdir upload_dir: $!";
        }
}
my $i=0;

foreach $key (param()) {
        next unless      $key =~ /file/;
        my $file_name = param($key);

        next unless      length($file_name)>0;

        $file_name =~ /([^\\\/]+)$/;
              $SaveFilename = $1;
        unless (open(OUTFILE, ">$SAVE_DIRECTORY/$SaveFilename")){
               print "Cannot open file $SAVE_DIRECTORY/$SaveFilename for upload.\n",p;;
               next;
         }
              while ($Bytes = read($file_name,$Buffer,1024)) {
        print OUTFILE $Buffer;
              }
              close(OUTFILE);
              close($file_name);
```

```
        print "Upload file to $SAVE_DIRECTORY/$SaveFilename.\n",p;

        $i++;



    }

    print "Total file count $i .\n";



}
```

注意这段代码在检查 form 输入上面既有 Server 端的输入检查，又有 Client 端的输入检查，两者之间的区别是：

Client 端的输入检查更快，Server 端的输入检查导致页面需要刷新。但是用户的浏览器可能关闭了 Javascript 选项，因此在 Server 端的输入检查是必需的。

第2章 Mod_Perl 技术

2.1 什么是 Mod_Perl

如果你打算扩展 Apache Web 服务器的功能，你可以试试 mod_perl 模块的威力。有了 mod_perl，就可以完全使用 Perl 写 Apache 模块，这样让你很容易的实现传统的 CGI 程序做起来很困难或是不可能的事情，例如重定向 URL。

Apache 服务器中的模块是个集成到一起的程序集合，可以让网络开发者扩展其服务器的功能，而无需修改服务器软件自身。一旦有了模块，你就可以增强服务器的功能，例如对客户端的访问控制，或者也可以对你的服务器进行优化，以便让 Perl 脚本运行得更快。

稳定的 Perl 解释器嵌入到服务器节省了从外部加载解释器的时间。更重要的是，代码缓存特性，加载的模块和代码仅编译一次，下次访问时，就可以直接访问缓存了。你能完全访问 Web 服务器的内部工作，可以介入请求处理的任何阶段。这样就允许定制例如 URI->文件名翻译，认证，生成响应和日志的过程。这样就不必为 CGI 程序开一个单独的进程。mod_perl 包含两个通用的模块：Apache::Registry 能够透明的运行已有的 perl CGI 脚本，Apache::PerlRun 也完成类似的工作，而且允许你运行“更脏”的脚本。你可以在 Perl 中配置你的 httpd 服务器进程和 handlers (通过使用 PerlSetVar 和<Perl>节)。你甚至可以定义你自己的配置指示器，例如，如何使用 mod_perl。那么 mod_perl 具体能够提高多少性能呢？这依赖于你用 mod_perl 做什么，性能提升可能从 200%到 2000%。

2.2 安装 mod_perl 三部曲

安装 mod_perl 三个容易的步骤是：取得构建 mod_perl 的源代码，构建 mod_perl，然后安装它。

从互联网取得 Apache 和 mod_perl 的源代码，分别可以从 http://www.apache.org/dist/httpd/和 http://perl.apache.org/dist/得到。

2.2.1 在 Unix 下安装

从源代码构建 mod_perl 需要机器上有一些工具软件。你需要有一个 ANSI 兼容的 C 编译器 (例如 gcc)和 make 工具。所有的标准 Unix 类型的操作系统都包括了这样的工具。如果这样的工具没有安装，就使用系统提供的软件包管理工具安装它们 (rpm，apt， yast 等)。

需要一个较新的 Perl 版本，至少是 5.004。

可以使用下列命令检查：

```
% make -v

% gcc -v

% perl -v
```

注意， mod_perl 1.0 仅能够和 Apache 1.3 在一起，而 mod_perl 2.0 需要 Apache 2.0。在这里我们仅讨论 mod_perl 1.0/Apache 1.3，因此你需要的包叫做 apache_1.3.xx.tar.gz 和 mod_perl-1.xx.tar.gz，这里 xx 应该使用 mod_perl 和 Apache 的真实版本号。

把下载的包放到一个路径，/home/stas/src/，进行上面描述的步骤，你就把 mod_perl 装上了：

```
% cd /home/stas/src

% tar -zvxf apache_1.3.xx.tar.gz

% tar -zvxf mod_perl-1.xx.tar.gz

% cd mod_perl-1.xx

% perl Makefile.PL APACHE_SRC=../apache_1.3.xx/src \

APACHE_PREFIX=/home/httpd DO_HTTPD=1 USE_APACI=1 EVERYTHING=1

% make && make test

% su

# make install
```

就是这些了。

余下的工作是把一些配置行加到 Apache 配置文件(/usr/local/apache/conf/httpd.conf)，启动服务器，享受 mod_perl 带来的好处。

2.2.2 在 Windows 下安装

首先我们需要可以在命令行使用的 Visual C++。如果使用 Microsoft Visual Studio .NET 就运行 vsvars32.bat(在\Microsoft Visual Studio .NET\Common7\Tools 目录下)。如果使用 VC++6.0，运行 vcvars32.bat(在\Microsoft VisualStudio\VC98\bin 目录下)。

下载并安装如下 Perl 模块：

Devel-Symdump 2.03 [不是必须，但是推荐安装]

Compress-Zlib 1.33 [不是必须，但是推荐安装]

FCGI 0.67 [不是必须，但是推荐安装]

libwin32 0.191

CGI.pm 3.05 [不是必须，但是推荐安装]

HTML-Tagset 3.03

HTML-Parser 3.36

URI 1.31

libwww-perl 5.800

运行：

>perl Makefile.PL MP_AP_PREFIX=\Path\to\Apache2

MP_AP_PREFIX 指向 pache2 的安装路径。例如当你 Apache2 的安装在缺省路径时，就运行：

>perl Makefile.PL MP_AP_PREFIX="C:\Program Files\Apache Group\Apache2"

构建包的时候，可能会提示安装 apxs，如果不能自动下载安装 apxs，可以手工下载 http://perl.apache.org/dist/win32-bin/apxs_win32.tar.gz 然后安装 apxs。构建代码完成后，执行：

> nmake

> nmake test

> nmake install

2.3 Apache 的 mod_perl 配置

2.3.1 mod_perl 基本配置

修改配置文件 httpd.conf，增加：

```
LoadFile "c:/Perl/bin/perl58.dll"
```

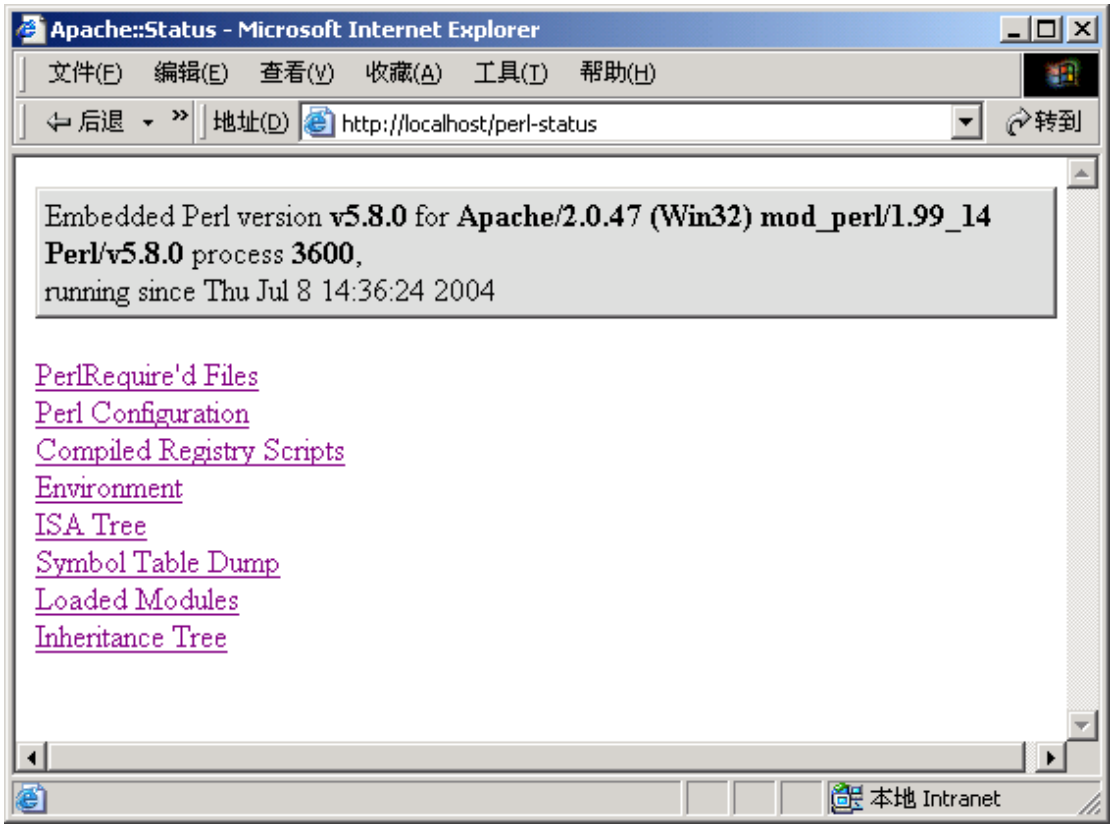
这里的"c:/Perl/bin/perl58.dll"依据你的 perl 安装路径和版本而定。然后加载模块：

```
LoadModule perl_module modules/mod_perl.so
```

添加察看 Perl 状态的功能。

```
<Location /perl-status>

SetHandler    perl-script

PerlHandler Apache::Status

</Location>
```

有了这个配置后就可以通过浏览器检查 mod_perl 的状态。



把所有 mod_perl 相关的配置放在配置文件的结束处是个不错的主意。为了减轻维护和简化多个服务器安装，Apache/mod_perl 配置系统允许你以几种不同的方式保存你的配置指令到不同的地方。在 httpd.conf 的 Include 指令允许包含其它文件的内容，就像信息包含在 httpd.conf 中一样。这是 Apache 自己的特性。例如，如果你想要所有的 mod_perl 配置放在一个独立的文件 mod_perl.conf ，你可以把这个指令增加到 httpd.conf：

Include conf/mod_perl.conf

如果想根据你的 apache 是否和 mod_perl 一起编译，条件性的包含这个配置，可以使用 IfModule 指令：

<IfModule mod_perl.c>

Include conf/mod_perl.conf

</IfModule>

mod_perl 更进一步增加了两个指令：<Perl>节允许你在服务器启动时从任何配置文件执行 Per 代码，简单的把文件的名字放在 PerlRequire 或 PerlModule 指令就可以执行任何包含 Perl 程序的文件。

2.3.2 Alias 配置

ScriptAlias 和 Alias 指令提供了一个映射 URI 到文件系统路径的方法。指令：

Alias /foo /home/httpd/foo

将映射所有的以/foo 开始的请求到以/home/httpd/foo/开始的文件。因此，当 Apache 取得一个请求 http://www.example.com/foo/test.pl，服务器将映射这个请求到在如下路径的 test.pl 文件。

/home/httpd/foo/

ScriptAlias 分配所有匹配这个 URI 的请求在 mod_cgi 下执行。

ScriptAlias /cgi-bin /home/httpd/cgi-bin

实际上等价于：

Alias /cgi-bin /home/httpd/cgi-bin

<Location /cgi-bin>

SetHandler cgi-script

Options +ExecCGI

</Location>

而后面的指令调用 mod_cgi。不要使用 ScriptAlias 指令除非想要请求在 mod_cgi 下处理。因此当配置 mod_perl 节时，只能使用 Alias。

在 mod_perl 下， Alias 指令随后跟着两个指令。第一个是 SetHandler perl-script 指令，它告诉 Apache 调用 mod_perl 运行脚本。第二个指令(例如 PerlHandler)告诉 mod_perl 脚本应该运行在哪一个 handler (Perl module)，属于请求的哪一步。要了解更多各种请求阶段的 handler，请参考后面的 Perl*Handlers。

2.3.3 运行位于同一路径下的 CGI, PerlRun 和 Registry 脚本

典型的普通 cgi 脚本：

ScriptAlias /cgi-bin/ /home/httpd/perl/

典型的 Apache::Registry 脚本：

Alias /perl/ /home/httpd/perl/

典型的 Apache::PerlRun 脚本：

Alias /cgi-perl/ /home/httpd/perl/

在上面的例子中，我们映射三个不同的 URI (http://www.example.com/perl/test.pl，

http://www.example.com/cgi-bin/test.pl 和 http://www.example.com/cgi-perl/test.pl)都对应同一个文件/home/httpd/perl/test.pl。这意味着我们可以让我们所有的 CGI 脚本位于文件系统中的同一个位置，以三种方式调用脚本，只需要并通过简单的改变 URI 的一个部件(cgi-bin|perl|cgi-perl)。

这个技术让移植脚本到 mod_perl 变得容易。如果脚本看起来不能很好的在 mod_perl 下运行， 在大多数情况下，你可以容易的在直接的 mod_cgi 模式或者 Apache::PerlRun 下调用脚本而不需要脚本作任何改变。只需要改变调用的 URL。

值得注意的是，从一个具有 mod_perl 功能的服务器在普通的 mod_cgi 模式运行脚本是不好的——这样资源消耗太高了。这样最好运行在普通的 Apache 服务器上。

2.3.4 <Location>配置

<Location>节指定了一些当请求的 URI 匹配 Location 时服务器应该遵循的规则。仅仅是一种广泛接受的约定，就像为你的 mod_cgi 脚本使用/cgi-bin，通常使用/perl 作为运行在 mod_perl 下的 perl 脚本的基本的 URI 。让我们看一下下面这个广泛使用的<Location>节：

Alias /perl/ /home/httpd/perl/

PerlModule Apache::Registry

<Location /perl>

SetHandler perl-script

PerlHandler Apache::Registry

Options ExecCGI

allow from all

PerlSendHeader On

</Location>

这个配置让所有的以/perl 开始的请求由 Apache 的 mod_perl 模块处理，并使用来源于 Apache::Registry 这个 Perl 模块的 handler。先看一下本例中在<Location>节的指令：

<Location /perl>

上面的 Alias 和它是一样的，如果你使用一个<Location>而没有同样的 Alias，则服务器将无法定位在文件系统中的脚本。Alias 只是提供 URI 到文件路径的转换规则。

有时候不执行脚本，而是执行模块。例如/perl-status，代码存储在一个 Apache 模块。载这时，我们不需要为这样的<Location>设置 Alias。

SetHandler perl-script

它指定 mod_perl Apache 模块处理请求的内容生成阶段。

PerlHandler Apache::Registry

告诉 Apache 在实际的内容生成阶段使用 Apache::Registry Perl 模块。

Options ExecCGI

这个指令接受各种参数(options)，其中一个是 ExecCGI。它告诉服务器该文件是一个程序应该执行它，而不是象静态 HTML 文件一样显示。

allow from all

这个指令用来设置基于域的访问控制。上面的设置允许从任何域的客户端运行脚本。

PerlSendHeader On

PerlSendHeader On 告诉服务器启动脚本时发送一个 HTTP header 给浏览器。这个设置在解析脚本生成的 header 之后调用 Apache 的 ap_send_http_header()方法。这意味着在 header 方面模拟 mod_cgi 行为。要发送 HTTP header ，使用 Apache Perl API 中的 $r->send_http_header 方法或者 CGI.pm 模块里的$q->header 方法更好些。

注意有时候在<Location>中使用模块之前，需要预加载模块。使用 Apache::Registry 的配置看起来像这样：

PerlModule Apache::Registry

<Location /perl>

SetHandler perl-script

PerlHandler Apache::Registry

Options ExecCGI

allow from all

PerlSendHeader On

</Location>

PerlModule 等价于 Perl 的内部函数 use() 调用。下面是使用 Apache::PerlRun 的一个非常相似的例子是：

<Location /cgi-perl>

SetHandler perl-script

PerlHandler Apache::PerlRun

Options ExecCGI

allow from all

PerlSendHeader On

</Location>

唯一于 Apache::Registry 不同的是 PerlHandler 指令，把 Apache::Registry 用 Apache::PerlRun 替换了。

2.3.5 PerlModule 和 PerlRequire 指令

在使用一个模块时，应该先加载它。PerlModule 和 PerlRequire 是完成这类功能的指令。PerlModule 用来加载模块，而 PerlRequire 用来加载文件。PerlModule 相当于 Perl 的 use()函数，而 PerlRequire 相当于 Perl 的 require()函数。可以传递一个或多个模块名称作为 PerlModule 的参数：

PerlModule Apache::DBI CGI DBD::Mysql

通常，模块从一个叫做 startup.pl 的启动脚本预加载。这个文件通过 PerlRequire 指令执行。例如：

PerlRequire /home/httpd/perl/lib/startup.pl

一个 PerlRequire 文件名可以是绝对路径的文件名，或者是带有 ServerRoot 或@INC 的相对路径的文件名。就像 use()或者 require()使用的任何 Perl 代码文件一样，startup.pl 必须返回真值。所以不要忘了在 startup.pl 的结尾处加 1; 。

2.3.6 Perl*Handlers

Apache 对于每个接收到的 HTTP 请求遍历一个循环。这个循环对应一系列请求处理器。

在编译和安装 mod_perl 之后，Apache 的 mod_perl 配置指令告诉 Apache 调用模块 mod_perl 作为它接收到的一些请求的处理器。尽管它能处理请求循环的所有阶段，但通常不会这样。可以通过配置文件中的 Perl*Handler 指令告诉 mod_perl ，它要处理哪些阶段(其余阶段留给其它模块或者缺省的 Apache 例程处理)。

因为 Perl 脚本做任何处理都需要 Perl 解释器 ，配置 Perl 和 C 处理器去处理请求的一部分就有些不同。通常，当编写一个 C 模块时，通过编译和配置挂钩到请求循环的一个指定的阶段。对一个 Perl 处理器 ，你编译 mod_perl 本身挂钩到合适的阶段，好像它要自己处理这些阶段。然后，你把 Perl*Handler 指示器放到你的配置文件，告诉 mod_perl，它把处理这些请求阶段的责任传递给你的 Perl 模块。

mod_perl 是一个用 C 编写的 Apache 模块。 因为多数程序员仅需要处理响应阶段，缺省状态下，大多数 Perl*Handlers 是关闭的。当你编译 mod_perl 配置 Makefile.PL 文件时，需要声明是否想处理部分请求循环而不仅仅是通常的内容生成阶段。

Apache 声明大约 11 个请求循环的阶段，依次是：Post-Read-Request，URI Translation，Header Parsing，Access Control，Authentication，Authorization，MIME type checking，FixUp，

Response (也叫做内容处理阶段)， Logging ， finally Cleanup。Apache API 允许模块进入这些请求的阶段。

Perl*Handler 是一个指令的集合，它涉及到请求的每一个阶段。我们把它叫做 Perl*Handler 指令集因为这些处理器都遵循同样的格式。例如 PerlLogHandler 是一个 Perl Handler，顾名思义，它处理日志阶段。一个例外是 PerlHandler，你可以把它看成是 PerlResponseHandler。它是内容生成处理器，因此也是最常使用的处理器。注意，是 mod_perl 认识这些指令，而不是 Apache，所以，它们是 mod_perl 指令。如果你得到错误信息说这些指令是"perhaps mis-spelled"，这就是一个信号，表示 mod_perl 模块的一部分或整个不存在。

Perl*Handler 指令集的全部的列表如下。按照 Apache 和 mod_perl 处理的顺序依次排列。

PerlChildInitHandler

PerlPostReadRequestHandler

PerlInitHandler

PerlTransHandler

PerlHeaderParserHandler

PerlAccessHandler

PerlAuthenHandler

PerlAuthzHandler

PerlTypeHandler

PerlFixupHandler

PerlHandler

PerlLogHandler

PerlCleanupHandler

PerlChildExitHandler

PerlDispatchHandler

PerlRestartHandler

PerlChildInitHandler 和 PerlChildExitHandler 不对应请求循环的一部分，它们允许你的模块在子进程启动和关闭时初始化数据结构和清理数据结构。例如，分配和释放内存。

所有的<Location>，<Directory>和<Files>节都包含物理路径声明。像 PerlChildInitHandler 和 PerlChildExitHandler，PerlPostReadRequestHandler 和 PerlTransHandler 就不能用在这些节，也不能用在.htaccess 文件中，因为直到 Translation Handler (PerlTransHandler) 阶段结束后，才完成路径转换，知道物理路径。

PerlInitHandler 随着使用的地点不同，它的行为也不同。服务一个请求时，在任何时候，PerlInitHandler 都是第一个被调用的处理器。如果在任何<Location>，<Directory>或<Files>节外部，它就是 PerlPostReadRequestHandler 的同义词。当在这些节的内部时，它就是 PerlHeaderParserHandler 的同义词。

从 PerlHeaderParserHandler 开始，请求 URI 映射到服务器物理路径，于是就能用于匹配<Location>，<Directory>或<Files>配置节，或访问.htaccess 文件，如果这个文件在翻译后的路径的确存在的话。

PerlDispatchHandler 和 PerlRestartHandler 不对应 Apache API 的一部分，但是允许你微调 mod_perl API。

2.3.7 启动文件配置

在服务器的启动时，在产生子进程接受请求之前，除了预加载文件以外还有很多事情可以做。例如，可以注册代码初始化数据库连接，对每个生成的子进程，绑定只读的 dbm 文件等。

startup.pl 文件是一个放置服务器启动时执行的代码理想的地方。准备好代码后，就把它加载到 httpd.conf 中，放在其余的 mod_perl 配置指示器之前，如下所示：

PerlRequire /home/httpd/perl/lib/startup.pl

必须强调的安全问题是，运行在服务器启动阶段的所有的代码是以 root 权限执行的 (因为在缺省的 80 端口上必须如此)。这意味着，任何人对一个 PerlModule 或 PerlRequire 加载的脚本或者模块有写的权限，就有了对系统的 root 权限。

因为启动文件也只是一个普通的 Perl 脚本，可以这样校验语法：

% perl -c /home/httpd/perl/lib/startup.pl

启动文件一个真实的例子如下：

startup.pl

```
# 这个编译指令是应该有的，虽然它使代码变得更长，但是它节省了以后调试时间。
use strict;
# 需要的话，扩展@INC
# 在服务起来以前修改@INC 仅有的机会。
# 以后运行的代码当它需要一些文件时也可以修改@INC，
# 但是@INC 的值就重置到原来了。
use lib qw(/dir/foo /dir/bar);
# 保证我们在一个健全的环境中。
# 如果 Apache/mod_perl 没有正常的搭建，就结束服务器启动。
$ENV{MOD_PERL} or die "not running under mod_perl!";
# 为了在"/perl" URL 下面的。
use Apache::Registry;
# 预加载 Perl 模块，这些模块用在服务请求的 Perl 代码中。
# 不要 import 这些变量或这函数，除非在 startup 文件里就需要它们，
# 因为额外的加载是浪费时间。
# 使用空的列表()告诉 import()函数不要 import 任何东西。
use LWP::UserAgent ();
use Apache::DBI ();
use DBI ();
# 允许更多的警告输出到错误日志文件。
use Carp ();
```

```
$SIG{__WARN__} = \&Carp::cluck;

# 加载 CGI.pm 并调用它的 compile()方法，

# 以便预编译它的 autoloaded 方法，但并不 import。

# 一些模块提供 autoloaded 方法以提高加载速度，但我们这里是为了提高代码执行速度。

use CGI ();

CGI->compile(':all');

# 为每个子进程初始化数据库连接。

Apache::DBI->connect_on_init

("DBI:mysql:database=test;host=localhost",

"user","password",

{

PrintError => 1, # 出错时 warn()。

RaiseError => 0, # 错误时并不结束执行。

AutoCommit => 1, # 立即提交。

}

);

# 这个是返回值，不要忘了。

1;
```

2.4 编写 Apache 模块

Apache 模块是符合 Apache API 声明的一段代码，它可以加载进 Apache Web 服务器。模块的加载方式可以有两种：

- 在编译 httpd 守护进程时静态加载。
- 在配置文件 httpd.conf 中动态加载。

动态加载模块的功能叫做 DSO 支持。 有了它，就不需要为了增加模块而重新编译 web server。一些著名的动态 Apache 模块包括：

- mod_frontpage：集成了 FrontPage 支持。
- mod_jk：集成了 Java Servlet 和 JSP 支持。
- mod_jserv：集成了 Java Servlet 支持。
- mod_perl：集成了 Perl 支持。
- mod_php4：集成了 PHP4 支持。
- mod_php3：集成了 PHP3 支持。

使用 LoadModule 指示器来加载动态模块到 Apache Web Server。LoadModule 应该放在配置文件 httpd.conf 的顶端。语法如下：

LoadModule MODULE-NAME modules/FILE-NAME

例如，增加 Status 模块，增加这行：

LoadModule status_module modules/mod_status.so

使用工具 APXS (APache eXtenSion)编译自己的 DSO 模块。使用 APXS 的命令行语法是：

% /usr/local/apache/1.3/bin/apxs OPTIONS MODULE_CODE

下面编写一个简单的例子来阐明：当服务器接收到一个给定页面的请求时，服务器返回 "Hello World"。

首先使用 apxs 命令来生成模块的框架代码，可以执行类似如下的命令：

>apxs -g -n hello

这个命令创建一个 hello 目录，并生成三个文件 mod_hello.c，Makefile 和.deps。然后编译并安装它：

>apxs -c -i -a /src/modules/mod_hello.c

我们现在只要修改 conf 目录中的 httpd.conf 文件：打开 httpd.conf 文件，找到 LoadModule 行，在该部分的最后一行后面，添加如下行：

<Location /hello>

SetHandler hello-handler

</Location>

该行告诉 Apache，任何对名为"/hello"的页面的请求，将会被转到 hello_world 模块，现在，重新启动 web 服务器，发出请求： http://localhost/hello。你将在你的浏览器中看到"The sample page from mod_hello.c" 。

2.5 编写 mod_perl 模块

我们使用 Perl 编写 Apache 模块实际上就是编写一个定义了 handler 函数的 Perl 模块(这只是一个约定——其实名字是可以改的)。这个函数接受一个参数 $r，类型是 Apache::RequestRec，它在 Perl API 中等价于 C API 的 request_rec *r。$r 是整个 Perl Apache API 的入口点，通过它你可以用面向对象的方式访问其中的方法。

让我们看一个简单的例子，Hello 模块返回服务器当前的时间。Hello.pm 的内容如下：

```
package Apache::Hello;



use Apache::Const qw(OK);

use Apache::RequestRec;



sub handler {

    my $r = shift;

    my $now = scalar localtime;

    $r->content_type('text/plain');



    print( "The local time is $now.");
```

```
    return OK;



}



1;
```

现在，我们需要设置 Apache 服务器，以便它可以找到我们希望使用的模块。一开始，要使用 mod_perl，我们需要创建一个名为 startup.pl 的文件，该文件将为 mod_perl 所使用，将会告诉 mod_perl 所有已安装程序的位置。我们将创建一个最小的 startup.pl 文件，可以用它来访问要运行的 Hello World 模块，要使得其他模块来说，也能够正常工作，就需要修改 startup.pl 文件。在你的 startup.pl 文件中，应包含如下内容：

```
use strict;



# Tell mod_perl where to find Hello.pm module

use lib qw(path/to/module)



# The name of the module to load using mod_perl

use Apache::Hello;



# This script must return TRUE, and this line accomplishes that, and conveniently

# also checks to ensure that we are actually running mod_perl with our version

# of Apache.



$ENV{MOD_PERL} or die "not running under mod_perl!";
```

下一步，要使得 Hello World 模块可以运行，就要对 Apache 服务器进行配置，以便使用 mod_perl。在前面的例子中，我们使用了 http://localhost/hello 作为访问用 C 语言编写的模块的请求。 对于 Perl 模块，我们使用 hello_perl。要设置得正确，就需要编辑 httpd.conf 文件。在配置文件的最后添加如下内容：

PerlRequire /path/to/startup.pl

<Location /hello_perl>

SetHandler perl-script

PerlHandler Apache::Hello

</Location>

PerlRequire 行告诉 mod_perl 到何处去寻找我们先前已经创建的 startup.pl 文件。标记为 Location 的一节将告诉 Apache 服务器,对/hello_perl 的请求应该用一个特殊的方式进行处理。在该节里，存在两个指示器：SetHandler 和 PerlHandler。

SetHandler 指示语将告诉 Apache，把所有的访问请求发送到 mod_perl，之后 PerlHandler 就指示 mod_perl 应该把这些请求转发到 Hello World 的 Perl 模块。

仅剩下最后一个步骤了：把 Hello.pm 文件复制到适当的位置。一旦创建了 startup.pl 文件，我们就规定好了 Hello.pm 模块的位置。我们需要在那个位置创建一个目录，把 Hello.pm 文件复制到该 Apache 目录中，所以，如果要在 startup.pl 文件中把 path/to/module 设置为 /usr/local/apache/perl/modules，就可以执行如下命令：

mkdir /usr/local/apache/perl/modules/Apache

cp Hello.pm

/usr/local/apache/perl/modules/Apache

这就把 Hello World 模块放到 Apache 名称空间中，确保没有其他的 Perl 脚本同它冲突。此时， Apache 和 mod_perl 都已经设置好，可以运行 Hello 模块了。我们可以重启动服务器。一旦服务器重启动之后，在你的浏览器位置栏打入 http://localhost/hello_perl，你可以看到如下信息：

The local time is Sun Jul 11 19:06:58 2004.

2.6 CGI 到 mod_perl 移植编码指南

这一章节主要说明从头写新的 CGI 脚本或者 perl 处理器以及将应用程序从传统的 CGI 移植到 mod_perl。

本节说明了现存的 CGI 脚本移植到新模式下可行的方法。通过在 Apache::PerlRun 模式下运行，把现存的 CGI 脚本移植到 mol_perl 的方法很容易，但是太“脏”了。

如果你的项目计划很紧，我建议你按以下步骤来移植到 mod_perl：

开始的时候，在 Apache::PerlRun 模式下运行所有脚本。如果时间允许的话，将它们放在 Apache::Registry 模式下，当你需要 Apache Perl API 函数的时候你可以增加对它的调用。

如果你打算写一段新 CGI 脚本，学习 mod_perl 相关易犯的错误并在一开始就避免它是一个很好的主意。

如果你不需要 mod_cgi 兼容性，最好在最开始就用 mod_perl API 来写代码。这样会使你的程序更有效些，并且很容易可以使用 mod_perl 提供的完整特性集，mod_perl 用 Apache 特定函数对 Perl 核心作了扩展，并且重载了一些 Perl 核心函数，让它们在 mod_perl 环境下更有效率。

2.6.1 探索 Apache::Registry 的秘密

让我们从一些简单的代码开始，来看看能出现什么错误，查找错误并且调试它们，讨论可能的“陷井”以及如何避免它们。

来看一个简单的 CGI 脚本，初始化$counter 为 0，然后增大其值并在浏览器打印出它的值。

counter.pl:

```
#!/usr/bin/perl -w

use strict;

print "Content-type: text/plain\r\n\r\n";

my $counter = 0;

for (1..5) {
```

```
increment_counter();

}

sub increment_counter{

$counter++;

print "Counter is equal to $counter !\r\n";

}
```

You would expect to see the output:

你应当会看到以下输出：

Counter is equal to 1 !

Counter is equal to 2 !

Counter is equal to 3 !

Counter is equal to 4 !

Counter is equal to 5 !

这是你第一次运行这个脚本时看见的输出，但是，我们多运行几次看看，

在加载几次之后，突然计数器不再是从 1 开始，继续加载看看接着会发生什么，计数器的开始值不是有规律的，几乎是随意的分布，10，10，10，15，20…不可思议！

Counter is equal to 6 !

Counter is equal to 7 !

Counter is equal to 8 !

Counter is equal to 9 !

Counter is equal to 10 !

在这段非常简单的脚本中我们发现两种不规则现象，计数器以 5 为单位的非预期的增长以及重复加载之后不一致的增长。让我们来分析一下这段脚本。

2.6.2 第一个迷团

首先来看错误日志文件，因为打开了警告，所以可以看到：

Variable "$counter" will not stay shared

at /home/httpd/perl/conference/counter.pl line 13.

变量"$counter"不保持共享的警告是当脚本中包含一个命名嵌套子程序引用了一个在这个嵌套子程序的外部定义的变量。

在脚本中发现命名嵌套子程序？你发现了吗？我没有！发生了什么？或许是一个 bug，但，等会，或许是 perl 解释器用另一种方式看这段代码，或许这段代码在实际执行前发生了一些变化？检查究竟发生了什么最好的办法是用调试器来运行这段脚本。

但是，因为我们必须在 Web 服务器运行它的时候调试，普通的调试器没法用，因为，调试器必须在 Web 服务器内被调用。幸运的是，Dong MacEachern 写了 Apache::DB 模块，我们可以用它来调试这个脚本。虽然 Apache::DB 允许交互式调试，但我们选择非交互式调试。

用下面的方式修改 httpd.conf 文件，

PerlSetEnv PERLDB_OPTS "NonStop=1 LineInfo=/tmp/db.out AutoTrace=1 frame=2"

PerlModule Apache::DB

PerlSetEnv PERLDB_OPTS "NonStop=1 LineInfo=/tmp/db.out AutoTrace=1 frame=2"

PerlModule Apache::DB

<Location /perl>

PerlFixupHandler Apache::DB

SetHandler perl-script

PerlHandler Apache::Registry

Options ExecCGI

PerlSendHeader On

</Location>

重新启动服务器，像以前一样请求执行 counter.pl。在表面上我们没有发现什么，跟以前得到的正确结果一样。但是，在背后有两个变化，第一，代码的执行踪迹被完整的写入/temp/db.out 文件。第二，如果你已经加载 Carp 模块，错误日志现在包含实际执行的代码。要加载 Carp 模块，只需要在 startup.pl 文件或者执行代码中加入：

use Carp;

下面是我们实际执行的代码：

```
package Apache::ROOT::perl::conference::counter_2epl;

use Apache qw(exit);

sub handler {

   BEGIN {

   $^W = 1;

   };

   $^W = 1;

   use strict;

   print "Content-type: text/plain\r\n\r\n";

   my $counter = 0;

   for (1..5) {

      increment_counter();

   }

   sub increment_counter{

      $counter++;

      print "Counter is equal to $counter !\r\n";

   }

}
```

错误日志中的代码不是像这样有层次的格式，我把它改成这样分层次的写法，是为了更清晰看出在 handler（）子程序中的代码。

我们从中发现了什么？首先，每一个 CGI 脚本都放在一个包内，包的名字是有 Apache::ROOT 前缀和相应脚本部分的 URL（perl::conference:::counter_2epl）通过用：：替代/用_2e 替代.构成。因此 mod_perl 知道应该从缓存中取得该脚本，每一个脚本都只是一个包含命名为 handler 的子程序的包。

如果在脚本中添加了诊断，在错误日志中可以找到提及嵌套子程序的问题，increment_counter 实际上是一个嵌套子程序。使用 mod_perl 时，在每一个 Apache::Registry 中的子程序都嵌套在 handler 子程序内。

需要注意的一点是：嵌套子程序只在代码在 Apache::Registry 范围内并声明了 handler 子程序的时候才起作用。如果你把代码放在一个 library 或者模块（module）中，被主代码 require 或者 use 调用，则不会起嵌套子程序的作用。比如，如果我们将脚本代码放到子程序 run 中，并把 run 放在 mylib.pl 文件中，把它保存在脚本所在的目录，在脚本中 require 调用它，这就不会有任何问题（在 library 的最后一行不要忘记加上 1;，不然 require 调用会失败）。

mylib.pl:

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

my $counter;

sub run{

print "Content-type: text/plain\r\n\r\n";

$counter = 0;

for (1..5) {

increment_counter();

}

}

sub increment_counter{

$counter++;

print "Counter is equal to $counter !\r\n";
```

```
}

1;
```

counter.pl:

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

use strict;

require "./mylib.pl";

run();
```

这种做法提供了最简单最便捷的解决嵌套子程序问题的方法，你所需要作的就是把代码放在一个单独的文件中，开始把初始代码封在一些函数里，然后可以在脚本中调用并且保证了局部变量不会在函数外发生问题。

但是从经验来说，除非脚本很短，我倾向于将所有代码写在外部 library 中，这样在主脚本中只需要几行代码。通常主脚本仅仅是调用 library 中的主函数，我一般把它命名为 init() 或者 run()。这样就不必担心嵌套子程序问题（除非是我自己造成的）。

不应该被这种问题阻吓，因为Perl应当是你的朋友。只需要打开警告模式，并且Perl会在发生这种问题时告诉你：变量"$counter"不能保持共享在…..，在发行产品之前，不要忘记去检查一些你的错误日志。

上面这个例子有些令人讨厌。在我最开始使用 mod_perl 的时候，我写了一个简单的用户注册程序，下面给出这个程序最简单的示例：

```
use CGI;

$q = CGI->new;

my $name = $q->param('name');

print_response();

sub print_response{

print "Content-type: text/plain\r\n\r\n";

print "Thank you, $name!";

}
```

我的老板和我在开发服务器上对这个程序进行了检查发现它运行良好，于是我们决定把它放在产品中，所有的事都准备好了，但是我的老板决定再提交不同的配置来测试一下，你可以想象一下他当时吃惊的样子，当提交他的名字（假设为“The Boss”）之后他看到的输出是：“Thank you, Stas Beknam!”。

接下来是我也来试这个产品，当时我对于 mod_perl 还了解不多，而且对于所带来的速度的提升太兴奋了根本没有注意到嵌套子程序问题，而让我遇到了这个问题。开始我以为是 Apache 开始把连接弄混乱了，返回了其他人请求的响应，当时是我错了。

为什么在我们的开发服务器上测试软件时没有注意到呢？接下来你会知道原因。

2.6.3 第二个迷团

让我们回到最初的那个例子，接着讲我们注意到的第二个谜团。为什么在多次加载之后我们看到不连续的结果？

其实很简单，每一次服务器得到一个请求来处理，它交给其中的一个 children 去处理，通常是一种循环方式。当你有 10 个 httpd children 存在，最初的 10 次加载看起来是正确的，因为我们谈到的问题在第二次再次时出现。

接下来再加载返回不是期望的结果。而且，请求看起来是随机的而且 children 并不总执行同样的脚本。在指定时间，一个子进程可能比另外一个进程服务同样的脚本更多次，而有的进程可能根本没有运行这个脚本。这就是我们看见奇怪的行为的原因。

现在你了解了，我们为什么没有注意到用户注册系统的问题。首先，我们没有察看 error_log(事实上，我们看了，但是日志中有许多警告，我们无法区分哪一个是重要的，哪一个不重要)。其次，我们有很多子进程，以致于没有注意这个问题。

一个测试方法是把服务器运行在单一进程下。做法是调用服务器使用–X 参数。因为没有其它的子进程，你能在第二次重新加载时就发现问题。

在此之前，让 error_log 帮助我们检查可能的错误——大多数警告都可能变成错误，因此你应该检查 perl 发现的每一个警告，最好把代码写成 error_log 不出现警告。如果每一个脚本调用时，你的 error_log 文件都新增加数百行，就很难发现和定位真正的问题。

2.6.4 Apache::Registry 总结

Apache::Registry 的工作方式是，代码进入一个新的包，叫做 Apache::ROOT::something。使用 eval()编译进一个子例程处理器：

```
$c = eval "
```

```
use Apache qw(exit);

package $NEW_PACKAGE;

sub handler {

$YOUR_PERL_CODE

}";
```

每次发送给 Apache::Request 请求时，就检查改变的程序。

它的副作用是任何时候只要这个变量不是一个块私有变量不会在请求结束时自动释放。所以我们应该：

- 关闭文件句柄。
- 如果有全局变量，在使用结束后 undef 它们。
- 应该使用私有变量(strict 可以帮助实现这一点)。
- Apache 的多个子进程意味着不能使用全局变量维护对这个程序两次调用之间的状态，在下次请求，一个不同的子进程可能处理它。

2.7 mod_perl 和数据库

使用 mod_perl 创建一个动态的网站通常要使用到关系型数据库。Apache::DBI 提供了一个数据库连接持久化的方法，它能加速 mod_perl 的性能。

当使用 CGI 和数据库打交道时，它的数据库连接不是持久的，每次请求 CGI 脚本就必须重新连接数据库，当请求完成时，连接就关闭了。Apache::DBI 提供了一个数据库的连接池，当你的 mod_perl 脚本需要数据库时，Apache::DBI 立即提供一个有效的联接，你的脚本可以立即开始工作而无需初始化数据库连接。这只是在有 mod_perl 功能的服务器上才有可能，因为在这种模式下当请求完成时，CGI 子进程并没有退出。

2.7.1 Apache::DBI 的原理

可以通过 DBI 模块来利用 Apache::DBI 模块的功能。当它加载的时候，DBI 模块测试是否已经设置了环境变量$ENV{MOD_PERL}，是否 Apache::DBI 模块已经加载。如果是这样，DBI 模块将传递每一个 connect()请求给 Apache::DBI 模块。

Apache::DBI 使用 ping()方法从前一个 connect()请求寻找一个数据库句柄，测试这个句

柄是否仍然有效。如果这两个条件满足，就返回这个数据库句柄。

如果没有合适的数据库句柄或者 ping()方法失败，Apache::DBI 将建立一个新的联接 ，并存储该句柄供以后重用。当脚本再次运行时，Apache::DBI 检查打开连接的缓存，并匹配它的主机名，用户名和密码。如果可能就返回一个匹配的连接，否则初始化一个新的返回。

在你的代码里，不需要删除 disconnect()语句。它们不会起作用，因为 Apache::DBI 模块覆盖了 disconnect()方法，用一个空的代替。

2.7.2 Apache::DBI 的适用条件

什么时候你应该使用它，什么时候不要用它呢？

如果打开几个数据库连接的时候，你会用到它。Apache::DBI 会让他们对每一个子进程持久化，因此如果你有 10 个子进程而每个打开两个不同的连接(使用不同的 connect() 参数) ，你就有了总共 20 个打开的而且时持久化的连接。在初始的 connect()后，对从 DBI 模块的每一个 connect()请求你都会节省连接时间。有大量数据库访问的服务器能够从此极大的受益。

如果为每一个用户打开一个特殊的连接，你就不应该使用这个模块。这意味着对每一个用户 login 参数是不同的。每一个连接会保持持久化，在一段时间以后，打开连接的数量将会达到允许的上限(这个值是通过数据库服务器配置的)，而新的数据库连接打开请求将会被拒绝，导致你的服务对一些用户是不可用的。

如果你想使用 Apache::DBI，但是在一台服务器上上面两种情况都存在，当前唯一的解决办法是运行两个 Apache/mod_perl 服务，一个使用 Apache::DBI，另外一个不用。

2.7.3 配置

在安装了这个模块以后，配置是很简单的——增加下面这个指示到 httpd.conf 文件。

```
PerlModule Apache::DBI
```

注意，加载的顺序很重要，应该把这个模块加载在 DBI 之前，也要在其它的 Apache*DBI 之前。你可以跳过预加载 DBI，因为 Apache::DBI 已经做了。但是把它放在 Apache::DBI 之后加载也是无妨的。

2.7.4 预打开 DBI 连接

如果你希望服务器起来以后你的脚本第一次执行以前一个连接已经打开，你可以在启动文件中使用 connect_on_init()方法，预加载你要使用的每一个连接。例如：

```
Apache::DBI->connect_on_init

("DBI:mysql:myDB:myserver",

   "username",

   "passwd",

   {

      PrintError => 1, # warn() on errors

      RaiseError => 0, # don't die on error

      AutoCommit => 1, # commit executes immediately

   }

);
```

这里有一个问题是，如果你调用 connect_on_init()而你的数据库没起来， Apache 的子进程在服务器起来时将会因为一直试图连接数据库而延迟。他们不会开始应答请求直到连接上数据库或者连接失败。这个过程可能需要等几分钟，我们可以设置连接的 timeout 值来避免长时间等待。

2.7.5 调试 Apache::DBI

如果你不确定这个模块正常工作，你应该在启动脚本打开调试模式：

```
$Apache::DBI::DEBUG = 1;
```

使用 ApacheDBI-0.84 启动，设置$Apache::DBI::DEBUG = 1 将生成最小的输出。为了全面追踪，你可以设置$Apache::DBI::DEBUG = 2。

在设置 DEBUG 级别后，当 Apache::DBI 初始化一个连接和当它从缓存返回一个连接时你都会在 error_log 看到一些条目。在 unix 下，使用如下的命令察看实事日志 (你的 error_log 可能在不同的目录下，在 Apache 配置文件中设置它)：

```
>tail -f /usr/local/apache/logs/error_log
```

在 windows 下察看文件 C:\Program Files\Apache Group\Apache2\logs\error.log。

2.7.6 数据库锁危险

当你使用 Apache::DBI 或者相似的持久连接时，要当心锁住数据库(LOCK TABLE ...)或单行操作。MySQL 线程保持表锁住直到线程结束(连接释放)或者表解锁。如果当表锁住时，你的会话死了，这些锁仍然在那里，因为你的连接没有关闭。

2.8 Apache::ASP

Apache::ASP，是用来在 mod_perl 下写 ASP 的。最早是在 1998 年发行，现在已经很成熟。并且在 perl 社区成员的几年的开发和使用中，创造了一个特色集合包括一些对微软标准的有用扩展。

利用 Apache::ASP，实现动态分页不是用 VBScript 或者 Jscript，而是 Perl。ASP 特色集已经很好的移植到 Perl 来，因此可以访问 ASP 内建的特性比如会话管理（采用定制的会话管理器，而不是 Apache::Session），邮件集成以及 ASP 对象模型。

Perl 的嵌入语法很简单：<% %>标签包围着 Perl 控制结构或者各种 Perl 指令，<%=%>标签内是想插入周围 HTML 的变量的值。例如：

```
<h2>Font sizes:</h2>
  <% foreach my $i (1..5) { %>
    <font size="<%= $i %>">Size = <%= $i %></font><br>
  <% } %>
```

这个例子的输出是：

```
<h2>Font sizes:</h2>
    <font size="1">Size = 1</font><br>
    <font size="2">Size = 2</font><br>
    <font size="3">Size = 3</font><br>
    <font size="4">Size = 4</font><br>
    <font size="5">Size = 5</font><br>
```

因为 Apache::ASP 是基于 ASP 的，在将 ASP/IIS 程序移植到 Apache 平台上时很自然的

会选择 Apache::ASP。移植过程必须将脚步由 VBScript 或者 Jscript 转换为 Perl，但站点的总体结构不需要变动。

除了标准的 ASP 特性外，Apache::ASP 还提供了另外一些特性，包括附加的事件处理，XML/XSLT 处理以及组件输出缓存。还提供不依赖 Cookie 来保存会话数据，这对于终端用户不愿意或者没法保存会话 cookie 来说是很便利的。对于 XSLT 的支持非常有意义，和输出缓存特性结合一起意味着可以在页面显示中使用动态 XSLT 转换，但是这种技术如果没有高速缓存从计算上说是不太现实的。

如果想对 Apache::ASP 有更多了解，请访问 http://www.apache-asp.org/。

第3章 使用 Mason

HTML 模版已经存在很久了。他们是在静止的页和把 HTML 放在一个程序内（这使得界面设计者不易接近）之间的一个好的折衷。 如果你想要一个动态改变的 HTML 页面，使用模版实现是一种好的方式。

Mason 就是一个模版系统，把 Perl 代码嵌入到超文本中。这样类似的系统还有许多，例如 Text::Template 或者 HTML::Template。但是许多人发现 Mason 的方式相当直接而且有益于完成工作。

Mason 最常用的应用在于构建大型动态网站，而且本章主要集中在 web 网站建设上。Mason 也广泛适用于精细的控制文档内容的任何场合，例如生成邮件，创建客户化的配置文件，甚至基于输入参数创建动态 GIF 图形。

Mason 鼓励你把你的网站当成结构化的项目，而不是过程脚本或者模块的集合。

3.1 安装 Mason

Mason 有两种安装方式：独立安装和 mod_perl 安装。

3.1.1 独立安装

Mason 1.10 或以上版本要求安装在 Perl 5.005 或以上版本。

要把 Mason 安装到任何 UNIX 平台，从 http://www.cpan.org/modules/by-module/HTML/ 或 http://www.masonhq.com/code/download/下载它，然后发出安装 Perl 模块的标准命令：perlMakefile.PL，然后 make，make test。如果没有错误就执行命令 make install。最后一步需要在管理者用户下执行，这样 Mason 就能为该计算机上所有用户使用了。

在 Win32 系统上，应该使用 ActiveState 的 PPM 工具安装 Mason。 PPMs for Mason 和它依赖的模块可以从 Randy Kobes 的 PPM 库 http://theory.uwinnipeg.ca/取得。

在安装的过程中，你可以注意到 Mason 查找 Apache 和 mod_perl。这些步骤也可以跳过。

Mason 依赖于一些模块，例如 Exception::Class，Class::Container 和 Params::Validate。因为手工安装这些模块很乏味，你可以使用 CPAN.pm 模块来自动化这些过程。你可以使用如下命令行开始 CPAN shell：

```
>perl -MCPAN -e shell
```

然后输入：

```
>install HTML::Mason
```

3.1.2 mod_perl 安装

这里的多数例子都假设 Mason 用在 web 上下文中。大多数人使用 Mason 时同时也使用 Apache 和 mod_perl。主要的成分是四部分：

- Perl 5.005 或以上版本。
- Apache：Mason 并不直接依赖于任何特定的 Apache 版本，你可以让 mod_perl 为你构建和安装 Apache。
- mod_perl：你可以从 http://perl.apache.org/dist/取得 mod_perl。mod_perl 遵循一个 perl 模块标准的安装过程，因此你可以向前面提到的那样发送同样的 perl Makefile.PL，make，maketest 和 makeinstall 命令。第一步有几个配置选项 ，为了全面的 Mason 兼容，使用命令 perl Makefile.PL EVERYTHING=1。我们不推荐构建 mod_perl 作为动态共享对象 (DSO)，因为这个配置在几个平台上不稳定。
- Mason：在安装完 mod_perl 之后，你就可以安装 Mason 了。其过程和前面独立安装说的完全一样，只不过如果安装 mod_perl 以后，make test 步骤将会和有 mod_perl 功能的 Apache 服务器一起测试 Mason。在发出 make install 命令之前确认所有的测试成功。

软件安装完成以后就是修改 Apache 配置文件。一个简单的配置是把下面放在站点的 httpd.conf 文件中：

```
PerlModule HTML::Mason::ApacheHandler

<LocationMatch "\.html$">

 SetHandler perl-script

 PerlHandler HTML::Mason::ApacheHandler

</LocationMatch>
```

这个配置告诉 Apache 所有的以.html 结尾的文件都是 Mason 组件，而且应该由 HTML::Mason 模块通过 mod_perl 处理。当然，你可以使用标准 Apache 配置指示器去声明传递请求给 Mason 的不同的条件。例如，如果你想限制 Mason 的影响到某一个路径，你可以使用像这样的配置：

```
PerlModule HTML::Mason::ApacheHandler
```

```
<Directory /path/to/subdirectory>

 <LocationMatch "\.html$">

  SetHandler perl-script

  PerlHandler HTML::Mason::ApacheHandler

 </LocationMatch>

</Directory>
```

很多配置都是可能的。如果你对你的站点有更复杂的需要，你需要熟悉 Apache 的配置指示器。关于 Apache 配置的相关文档可以在 http://httpd.apache.org/docs/找到。

3.2 第一个例子

如下是一个完整的 Mason 代码，叫做一个组件：

```
% my $planet = "World";

Hello, <% $planet %>!
```

当 Mason 运行这段代码，输出是：

```
Hello, World!
```

我们要知道的第一件事是以%字符开始的任何行告诉 Mason 这行包含 Perl 代码。这个例子中，它仅仅简单的设置在组件中要使用到的值，即$planet。

另外一个值得注意的是替代标签，它用顺序的<% %>表示。Mason 会求替代标签的值 ，并把结果插入到周围的文本中去。在这里，变量$planet 的值是 World，而整个组件的输出是 Hello, World! 注意，不是特殊的 Mason 记号的任何文本都简单的变成了输出组件的部分。

这种分离的机制能让你以一种服务器端包含(SSI)的机制使用 Mason。执行 mainpage.mas 将产生一个包含 header 和 footer 的全页面的 HTML。

header.mas 文件的内容如下：

```
<html>

 <head><title>Welcome to Wally World!</title></head>
```

```
<body bgcolor="#CCFFCC">
```

footer.mas 文件的内容如下：

```
<center><a href="/">Home</a></center>

</body></html>
```

下面这个例子引入组件调用标记语法<& &>，它用来调用另外一个组件并把它的输出插入到周围的文本。组件标记也能够接受参数，在这里它能够帮助统一站点设计，把页面 header 文本移到了 header.mas 组件。

mainpage.mas 文件的内容如下：

```
<& header.mas &>

<center><h1>Wally World Home</h1></center>

Here at Wally World you'll find all the finest accoutrements.

<& footer.mas &>
```

下面这个例子中的 header.mas 组件现在接受一个参数叫做$head，包含应该插入<h1>标记的文本。一个组件的参数使用<%args>块声明，随后你可以看到更多的细节。$head 参数变成了一个 Perl 变量可以在组件的余下部分使用。

header.mas 文件的内容如下：

```
<%args>

 $head

</%args>

<html>

<head><title>Welcome to Wally World!</title></head>

<body bgcolor="#CCFFCC">
```

```
<center><h1><% $head %></h1></center>
```

footer.mas 组件相当直接。它仅提供了一个指向根文档的连接。footer.mas 文件的内容如下：

```
<center><a href="/">Home</a></center>

</body></html>
```

在 mainpage.mas 组件中，使用标准的 Perl 语法把参数传递给 header.mas 组件(即， 逗号，引号和=> 操作符)。事实上，可以使用任何传递一个列表的 Perl 语法，因为底层实现使用的是就是参数列表。

mainpage.mas 文件的内容如下：

```
<& header.mas, head => "Wally World Home" &>

Here at Wally World you'll find all the latest accoutrements.

<& footer.mas &>
```

这样，header.mas 中的$head 的值就成了"Wally World Home"。

3.3 基本语法

Mason 解析组件的方法是，把组件的文本翻译成实际的 Perl 代码。这段 Perl 代码在执行时创建一个新的 HTML::Mason::Component 对象。对象反过来生成原来在组件中的文本。换句话说，这样把组件由嵌入 Perl 的文本转变成嵌入文本的 Perl 代码。

Mason 组件中使用的标识语言包含一个简单的标记做 Perl 表达式的原地替换。大多数这样的标识由于标记的不同而被赋予了不同的含义。

3.3.1 替换标记与逃避替换

<% %>这个最简单的 Mason 标记用于插入一个 Perl 表达式的结果到文本中去。这个标记相当类似于其它系统，如 JSP。一个简单的例子如下：

```
% $cd_count = 207; # 这是嵌入式 Perl

You have <% $cd_count %> CDs.
```

这个例子的输出是：

You have 207 CDs.

这个标记的内容是在列表上下文中计算出来然后合并到一起，就好像他们作为一个参数传递给 Perl 的 print()函数。

可以把更复杂的 Perl 表达式放在你的替换标记中。例如，处理复数属性，第二行可以写成：

```
You have <% $cd_count %> CD<% $cd_count != 1 ? 's': '' %>
```

根据$cd_count 变量的值不同，输出可能是下面任何一种：

You have 207 CDs.

You have 1 CD.

You have 0 CDs.

替换标记的内容最后成了 Perl 代码，因此忽略了空白符号。

Mason 提供的一个非常有用的特性是在标记输出以前逃避一个标记的内容的能力。逃避是把不安全的字符变得安全的处理过程。在 web 上下文，安全意味着我们不能生成可能被 HTML 误解的输出。而且，我们也可能需要做 URL 风格的逃避。

替换逃避用一个 | 跟着一个或者多个逃避标识，放在关闭标记之前。当前有三种有效的逃避标识，h 用于 HTML 实体逃避(例如，> 变成 >)，u 用于 URI 逃避(例如，> 变成 %3E)，而 n 用于不逃避。HTML 和 URI 逃避标识可以一起使用(例如，hu)或者单独使用。一个例子如下：

```
Name: <% $name | h %>

Homepage: <a href="redirect?url=<% $homepage | u %>">
```

HTML 逃避模式使用 HTML::Entities 模块逃避字符串，它意味着所有的控制和高位字符都逃避了，当然还包括大于和小于符号(< 和 >)，以及连接符 (&)和双引号(")。缺省情况下，HTML::Entities 假设使用 ISO-8859-1 字符集。如果要生成 gb2312 字符集的文本，就需要重写逃避的工作方式。

如果你正在显示一个从外部数据源数据的页面，HTM 逃避就特别有用。例如，如下的代码：

```
<textarea name="foo"><% $foo_data %></textarea>
```

如果$foo_data 包含字符串</textarea>，HTML 就是失效了。逃避输出可以用来防止这种情况发生：

```
<textarea name="foo"><% $foo_data | h %></textarea>
```

3.3.2 嵌入 Perl 代码

这里介绍“% 行”和“<%perl>块”语法。有两种方式嵌入 Perl 代码到 Mason 文本。第一种方式是 Perl 行的方式。它是以一个百分符号%开始的行。这行余下的部分就解释成 Perl 代码。这个百分符号前面不能有任何空白符号。一个典型的应用是实现 Perl 控制结构。例如：

```
% foreach my $person (@people) {
    Name: <% $person->{name} %>
    Age: <% $person->{age} %>
    Height: <% $person->{height} %>
    Weight: <% $person->{weight} %>
% }
```

也可以把大段代码放在 Perl 行，例如前面的代码等价于如下的形式：

```
% foreach my $person (@people) {
%     print "Name: ", $person->{name}, "\n";
%     print "Age: ", $person->{age}, "\n";
%     print "Height: ", $person->{height}, "\n";
%     print "Weight: ", $person->{weight}, "\n";
% }
```

如果有多行 Perl 代码在一起则最好使用 Perl 块。一个 Perl 块等价于一个连续的 Perl 行。Perl 块以<%perl>标记开始，以</%perl>标记结束。

例如，可以使用这个标记在文本中做一些数据处理工作：

```
<%perl>

  my @words = $sentence =~ /\b(\S+)\b/g;

  my @small_words = grep { length $_ <= 3 } @words;

  my @big_words = grep { length $_ > 3 } @words;

</%perl>

There were <% scalar @words %> in the sentence.

The big words were:

% foreach my $word (@big_words) {

    <% $word %>

% }

The small words were:

% foreach my $word (@small_words) {

    <% $

word %>

% }
```

3.3.3 调用其它组件

<& &> Tags

Mason 中最强大的特性之一是一个组件执行另外一个组件的能力。这样被调用者组件的输出就出现在调用者的输出里。被调用者组件还可以调用其它组件。有几种方式可以调用组件，但是最简单的方式是通过连接符标记，就像下面这个例子：

```
<html>

<head>

<title>The Goober Guide</title>
```

```
</head>

<body>

<h1>Welcome to The Goober Guide!</h1>

<& menu &>

...

</body>

</html>
```

菜单组件可以包含一个用于站点的所有页面的导航条。另外一个调用的例子是：

```
<& /path/to/menu &>

<& $component &>

<& menu, width => 640, admin => 1 &>
```

这些例子解释了 Mason 的组件调用标记的几个方面。首先，组件可以通过直接使用名字的方式声明或者间接的作为 Perl 表达式的结果像例子中的$component 这样声明。而且组件调用可以像 Perl 函数那样接受参数(像第三个例子中的 width 和 admin 那样)。

3.3.4 其它命名块介绍

Mason 还有许多其它的命名块。他们的语法和<%perl>块类似的开始和结束标记。而且大部分都包含 Perl 语句。但是，这些块各有其特殊的含义。

<%init>块

<%init>块是最经常使用的 Mason 块之一。它包含的 Perl 代码在其它代码之前运行，但是不包括<%once> 或 <%shared> 块。在每次调用组件的时候就运行它。

使用这个模块和把一个<%perl>块放在组件的顶端的效果一样，但是这样允许你把它从组件的主体部分分开。

<%init>块典型的用作像参数检查，创建对象或者从数据库中查询数据这样的操作。下面的<%init>块初始化一个用于组件其它部分的变量。

```
It is currently <% $temp %> degrees.

<%init>

  my ($temp) = $dbh->selectrow_array("SELECT temperature FROM current_weather");

</%init>
```

<%args>块

如我们已经在前面提到的，组件能带有许多参数，参数可以来自于外部(HTTP 请求，例如)或者内部(一个组件调用另一个)。

可以声明一个组件期望的参数的名字，数据类型，以及这些参数的默认值。一个典型的<%args>块看起来象这样：

```
<%args>

  $color

  $size    => 20    # A default value

  @items => ( 1, 2, 'something else' )

  %pairs => ( key1 => 1, key2 => 'value' )

</%args>
```

这个例子演示了为这个块全部的语法可能性。 首先，我们有参数类型和名字。有效的类型是：标量，数组，以及哈西表。

可以给参数一个默认值。当调用该组件的时候没有提供参数值时就使用这个值。 没有默认值的任何参数被认为是要求的参数。调用一个组件而不指定全部它要求的参数将导致抛出一个致命的异常。

参数的默认值能引用前面的参数，因此这样是完全合法的：

```
<%args>

  $x

  $y => $x * 2 > 20 ? 50 : 100
```

```
</%args>
```

当这个块看起来好象它包含的是 Perl 代码，但是我们应该意识到它的语法实际上 Mason 独特的语法。 重要的是，行不应该以一个分号或者逗号结束，并且每个变量定义都一定在一条单个的行上。可以在一个参数声明之后或者在新行上包含注释。 注释以#开始并且直到本行的结束，就象在 Perl 中的一样。参数块同时也允许空白行。

<%filter>块

在已经完成运行组件之后调用<%filter>块。 在这个块中，可以通过$_变量访问组件的整个输出，同时对这个变量的任何变化都会反映在组件的输出上。例如，这个过滤器把组件所有的输出大写化：

```
<%filter>

 s/(\w+)/\U$1/g

</%filter>
```

<%once>块

每当组件装载进内存的时候，执行这个块。 它在任何其他块之前被执行(包括<%init>块)。在这里声明的任何变量都将保持存在，直到组件执行完毕。 <%once >块对创建数据库连接或者实例化大的资源密集的对象等类似的操作有用。例如：

```
The universe is this big: <% $size %>

<%once>

 my $size = calculate_size_of_universe( );

</%once>
```

<%cleanup>块

清除块刚好在那些组件退出之前被执行，并且对应于<%init>块。它能用于你已经创建好的资源——例如需要释放的循环引用。 实际上，它与放在一个组件结尾处的<%perl>块效果相同。这样同时意味着如果组件中途退出就不会执行清除块。

```
<%init>

  my $resource = get_a_resource( );

</%init>



... do something interesting with that resource



<%cleanup>

  $resource->dispose;

</%cleanup>
```

因为清理代码一般放在组件的末端，<%cleanup>块不常用。使用他们的主要优点是他们的名字清楚。

<%text>块

这个块的内容完全按照原样输出，没有任何解析。 这个块可用于写包括有关 Mason 的正文的组件。例如：

```
<%text>

  Substitution tags look like this: <% $var %>.

</%text>
```

<%doc>块

这个块是为组件作者编制文档目的准备的。将完全忽略它的内容。 将来 Mason 可能利用他们做更有用的事情。<%doc>块的例子如下：

```
<%doc>

=head1 My Story
```

This is the part where I tell you what the component does. But I'd

rather tell you a story about my childhood. When I was but a

child, my mother said to me ...

</%doc>

如你所见，没理由不在这些块里使用 POD。你甚至还能对一个组件文件运行 perldoc。

<%flags>和<%attr>块

这两个块具有相同的句法并且用来声明一个或更多个关键字/值对。 这个关键字只能包含字母，数目和下划线字符(_)。值可以是任何表示 Perl 标量可以有的值(例如号码，字符串，引用或者 undef)。

就像在<%args>块一样，在这些块内的语法看起来象 Perl，但是不是。首先，你不能以一个逗号或者分号结束一行。其次，整个关键字/值对必须在一个单行上。

在这两个之间的差别是<%flags>块可能只包含正式的 Mason 标识，这些标识可以用来影响组件的行为。目前，只定义了一个标识，它就是 inherit。它用来指定组件的父组件。<%attr>块可能包含你想要的任何关键字，因为在这个块里定义的变量不是被 Mason 使用，而是在你的代码里使用。它的内容的可通过调用对象的 attr ()方法得到，这个方法以想要的关键字作为参数。例子如下：

<%flags>

inherit => '/some/other/component'

</%flags>

<%attr>

color => "I'm so blue"

size => 'mucho grande'

</%attr>

My color: <% $m->base_comp->attr('color') %>

<%def>和<%method>块

这两个块使用有点不同于任何其他 Mason 块的语法，因为他们的内容是组件。 <%def>块包含子组件，可能被通过正常的 Mason 组件调用语法调用的一个嵌入的组件。<%method>块也包含一个嵌入的组件，但是它可以被一个组件的后代继承。 <%def>和 <%method>块要求在初始化标签内有一个名字。 在下面这个例子里，定义了一个叫做.make_a_link 的子组件：

```
<%def .make_a_link>
 <a href="<% $url %>"><% $text %></a>
 <%args>
  $path
  %query => ( )
  $text
 </%args>
 <%init>
  my $url = ...
 </%init>
</%def>
```

子组件或者方法的名字可以包含字母或数字，下划线(_)，短横(-)，或者圆点(.)。按照常规，为了他们和非嵌入式组件分辨开，圆点是子组件名字的第一个字符。方法一般不遵循这个命名方法，他们的名字没有前面的圆点。 在子组件和方法之间的主要差别仅仅是子组件只在他们被定义的组件内是可见的，而方法在组件外部是可见的。可以通过 Mason 的组件继承机制继承方法。

<%shared>块

这个块也包含 Perl 代码。 每次相应请求时执行在这个块内的代码，在<%init>块之前，但是与在<%init>块不同， 在这个块里声明的变量范围既包括组件的主体里又包括该组件可

能包含的任何子组件或者方法。这个块可用于在一个单个的组件的各个部分之间共享大量共同的代码。

逃避新行

当使用 Mason 时，你会发现有时候可能想抑制输出新行符，一个典型的例子如下：

```
<pre>
I am
% if ($height < 5) {
  not
% } elsif ( $height < 5.75 ) {
  not very
% } elsif ( $height > 6.25 ) {
  very
% }
tall
</pre>
```

如果$height 小于 5 将产生如下的输出：

```
<pre>
I am
  not
tall
</pre>
```

新行在输出内不合乎需要，但是因为需要 Perl 代码在独立的行存在，所以这是不可避免的。因此 Mason 提供除去新行能力。我们仅仅需要在它前加上一个反斜杠(\)。 如果我们

用逃避新行重写前面的例子，它看起来象这样：

```
<pre>
I am\
% if ($height < 5) {
  not\
% } elsif ( $height < 5.75 ) {
  not very\
% } elsif ( $height > 6.25 ) {
  very\
% }
tall
</pre>
```

现在这个例子中，如果$height 小于 5 产生的输出是：

```
<pre>
I am not tall
</pre>
```

3.3.5 组件参数

大多数组件期望得到命名的参数，并且这些参数可能以下列两种方式之一传递：组件能通过外部请求得到参数，例如那些通过 HTTP；他们也可以被另一组件调用而得到参数。在被调用的组件里可以通过几个机制得到这些参数。但是从一个组件的角度来看，怎样被调用基本上和它不相关。

因为我们谈论参数，值得再看一下以前讨论的<%args>块。这个块用来声明一个组件期望的参数。另外，它也能用来指定缺省值。 我们前面使用的块是：

```
<%args>
```

```
$color

$size => 20

@items => ( 1, 2, 'something else' )

%pairs => ( key1 => 1, key2 => 'value' )

</%args>
```

这个组件前面两个参数是两个标量，一个是颜色，这个是必要的参数，以及一个尺寸，这个不是必要的参数，缺省值是 20。组件的第三个参数是一个数组项目，缺省值是(1，2，'something else')，第四个参数是一个哈西表，缺省值是(key1 =>1，key2 =>'value')。后面两个参数都不是必需的。

这些参数在作为词法变量在组件里可以访问。 例如，你的组件将有一个$color 的词法范围的变量。除了在<%args>块你不需要在任何地方声明它。如果一个强制参数(即没有缺省值)没有在调用里提供，将抛出一个异常。

%ARGS

除了在<%args>块里通过声明创造的词法变量之外，每个组件也有一个词法范围的哈西表，叫做%ARGS。这个哈西表包含组件被调用的所有参数。有一点容易混淆的是在%ARGS 和<%args>块之间的差别。%ARGS 包含那些参数完全按照他们被传给组件的那样，无论他们是否声明在<%args>块。%ARGS 的关键字不包含 Perl 变量标识($，@或者%)。 声明的参数在<%args>块以$color 的形式出现，因此可以通过$ARGS{color}访问。任何通过 <%args>块的默认赋值，在%ARGS 里都不可见；这些值完全按照他们传递过来的那样赋予。另外，不管一个组件的<%args>块是否存在，% ARGS 总是存在。

如果你正期望许多类似名称项目的输入，例如 input1，input2，等等，一直到 input20，在<%args>块声明所有这些参数可能笨拙一点。 这样的话，%ARGS 哈西就很方便。

如果你要求的参数的名字不能被用于 Perl 变量的名字，这种情况下%ARGS 也有用。例如， 当提交通过点击一幅图像命名的提交时，浏览器将产生两个附加的 form 值，叫做 submit.x 和 submit.y。你不能有一个 Perl 变量的名称叫作$submit.x，，得到这个参数的唯一方法是检查$ARGS {' submit.x ' }。

%ARGS 和@_

Mason 传统总是使用命名的参数。不过，适合简单组件，你可能喜欢使用@_访问参数，正象在 Perl 子程序里。要注意的是，如果你的组件包含一个<%args>节，Mason 期望在@_中得到偶数个参数，以便能分配@_给%ARGS。 如果它得到奇数个参数，将发生致命错误。

但是不管怎样传递参数，在组件里总可以访问@_。 因此当组件得到偶数个参数时，下列代码几乎相同：

```
% foreach (sort %ARGS) {
    <% $_ %>
% }
```

```
% foreach (sort @_) {
    <% $_ %>
% }
```

3.4 对象风格的组件继承

Mason 优秀的特性之一是组件可以从其它组件继承行为，非常像面向对象层次结构中的类和对象。典型的，每一个组件继承自唯一的一个叫做 autohandler 的组件。autohandler 实现了所有的组件的通用行为，例如 headers 和 footers 的内容。独立的组件实现特定的行为 ，例如每一个页面的文本内容。

使用组件继承，我们可以用 Mason 更常用的用法重写上面的例子，像下面所做的那样：

autohandler 文件的内容如下：

```
<html>
<head><title>Welcome to Wally World!</title></head>
<body bgcolor="#CCFFCC">
<center><h1><% $m->base_comp->attr('head') %></h1></center>
% $m->call_next;
<center><a href="/">Home</a></center>
</body></html>
```

mainpage.mas 文件的内容如下：

```
<%attr>

 head => "Wally World Home"

</%attr>

Here at Wally World you'll find all the finest accoutrements.
```

注意，header 和 footer 都在 autohandler 这一个文件中。很明显，这有助于整合页面内容，因为像 <html> 和 <body>这样在 header 打开的标签都在同一个文件中关闭了。其他重要的差别是 mainpage.mas 不再需要显式的调用 header 和 footer 组件，转而由 Mason 自动的调用父组件并把 header 和 footer 放在主要内容的周围。页面 header 现在通过一个属性块声明，它是 Mason 的一个面向对象机制。

3.4.1 组件继承

使用 inherit 标记声明一个组件的父组件：

```
<%flags>

 inherit => 'mommy.mas'

</%flags>
```

如果一个组件不显式的声明父组件，Mason 分配一个缺省的父组件。任何常规组件(不是 autohandler)的缺省父组件是一个同一路径下叫做"autohandler"的组件。如果没有 autohandler 存在同一路径下，Mason 会查找父路径下的 autohandler，然后再上一个，依次递归直到抵达组件的根路径。如果没有发现 autohandler，就不指定父组件。

autohandler 的缺省父组件是上一个路径下的 autohandler。换句话说，autohandler 像其他组件一样继承，除了它不会继承它自己。注意，这些只是缺省值，任何组件，包括 autohandler，都可以通过设置 inherit 标志显式的声明一个父组件。

Autohandler 机制不仅仅能用于生成 header 和 footer。下面介绍两个非常常用的 autohandler 技巧。

3.4.2 把 Autohandler 用于初始化

首先，大多数站点都要与数据库交互。通常，想要在响应的开始处打开数据库连接并且使数据库句柄在请求的整个生命周期都可得到。autohandler 为实现它提供了一个便利的方法。下面是一个 autohandler 文件的内容：

```
<html>

<head><title>Example.com</title></head>

<body>

% $m->call_next;

<br><a href="/">Home</a>

</body>

</html>
```

```
<%init>

  $dbh = DBI->connect('DBI:mysql:mydb;mysql_read_default_file=/home/ken/my.cnf')

     or die "Can't connect to database: $DBI::errstr";

</%init>
```

view_user.mas 是一个使用数据库句柄的例子：

```
<%args>

  $user

</%args>


% if (defined $name) {

  <p>Info for user '<% $user %>':</p>
```

```
  <b>Name:</b> <% $name %><br>

  <b>Age:</b>  <% $age  %><br>

% } else {

  <p>Sorry, no such user '<% $user %>'.</p>

% }



<%init>

  my ($name, $age) = $dbh->selectrow_array

    ("SELECT name, age FROM users WHERE user=?", undef, $user);

</%init>
```

注意，变量$dbh 没有在任何组件内使用 my()声明，因此可以使用 Mason 参数声明 (或者，等效的，在 Apache 配置文件里的 MasonAllowGlobals 指令)。 allow_globals 参数告诉编译器当编译组件时增加 use vars 语句，允许使用指定的全局变量。这是在一个请求过程中在几个组件之间共享变量最容易的方法，但是应该谨慎使用这种方法，因为有太多全局变量就很难管理了。

我们将对这个例子做一个简短的追踪。 首先，Mason 得到一个请求，是 http://example.com/view_user.mas?user=ken。Mason 把它翻译成对 view_user.mas 组件的请求。象以前一样，autohandler 首先执行，产生头和脚标，而且这时候也连接到数据库。当 autohandler 传递控制给 view_user.mas，运行它的<%init>节并且使用在 autohandler 内创造的相同的$dbh 全局变量。 取得一些数据库值并且用于输出，当控制回传给 autohandler 后，请求就完成了。

3.4.3 把 Autohandler 用于 Filter

下面这个例子是 autohandlers 的另一种常见的用途。 经常每页的内容需要以一种系统的方式修改，例如，把<img src>标记中的相对 URL 转换成绝对 URL。

/autohandler

```
% $m->call_next;
```

```
<%filter>

 # Images are on images.mysite.com

 (my $host = $r->hostname) =~ s/^.*?(\w+\.\w+)$/images.$1/;

 # Remove final filename from path to get directory

 (my $path = $r->uri)          =~ s,/[^/]+$,,;



 # Matches site-relative paths

 s{(<img[^>]+src=\")/}              {$1http://$host/}ig;



 # Matches directory-relative paths

 s{(<img[^>]+src=\")(?!\w+:)} {$1http://$host$path/}ig;

</%filter>
```

这个特别的 autohandler 没有增加 header 和 footer 到页面，但是它完全可以这样做。 在 autohandler 里的任何附加内容象在我们的以前的例子里一样起作用，也一样得到过滤就像来自 call_next()的内容一样。

我们对整个页面做了两个替换。 第一个替换把象<img src="/img/picture.gif">这样的 URL 改变成<img src="http://images.mysite.com/img/picture.gif">。第二个替换把<img src="picture.gif">这样的 URL 改变成<img src="http://images.mysite.com/current_dir/picture.gif">。过滤器节非常方便做象这样的改变图像路径，改变与当前页面匹配的导航条或者做其他的简单变换。为非常复杂的处理使用过滤器不是一个好主意，因为解析 HTML 会变成很棘手的问题。但是通过合理使用继承可以得到对 HTML 精细的控制。

3.5 智能缓存机制

任何构建过动态站点的人都知道有时候站点的某些部分可能要花比你想要用户等的时间更长的时间去生成出来。进一步，站点的部分可能仅是"半动态的"，意味着它们的内容周期性的改变，但是在此改变期间之间长期保持静态。或者，像一个新闻站点，或者在线调查，内容可能持续改变，但是如果能够提高性能，更新内容延迟长达几分钟也是可以接受的。为了对付这些问题，Mason 提供了一个非常成熟的缓存机制 ，你可以用来控制一个组件重建

的频率。你可以基于失效时间或者某像用户名或者内容 ID 这样的一个关键参数，或者一个声明数据失效的代理。

缓存机制可以用于组件的输出，用于任意的文本块，或者任意你想缓存的 Perl 数据结构。对缓存的一流的支持是 Mason 最可爱的特性，它能节省你花数小时优化行动缓慢的代码。

为了帮助全局优化，Mason 也有一个智能的内部缓存机制。在执行期间，Mason 把每一个组件中的 Perl 源代码编译成字节代码，然后执行字节代码产生组件输出。每次一个组件需要执行的时候，重复这个过程可能是浪费计算资源，因此 Mason 在每一步都缓存。为了有助于快速开发，Mason 会检查你的组件的修改时间并在你改变组件时，把它在缓存中失效。这样保证你对站点所做的任何改变都会立即生效。当你的站点从开发到产品阶段时，你可能不想经常的改变站点，因此你可以关闭更新检查以提高站点的响应速度。

3.6 从组件生成静态站点

许多站点可能最好用一系列静态的文件而不是一系列动态创建的请求响应。例如，如果一个站点的内容仅仅一周变化一到两次，根据请求动态生成每个页面可能就太过了。而且，如果你不需要动态生成网页的机制，就可能经常发现一些更便宜的或免费的虚拟主机。

例如，我们分析一个电影评论的站点。它的原型是一个香港电影评论的站点。我们的例子主要是一些页面显示电影信息，包括电影标题，发布日期，导演，演员表，当然，还包括影评。我们从 Mason 组件生成这个站点，然后上传到主机。

首先，我们需要一个路径布局。假设我们从路径/home/dave/review-site 开始，如下是路径结构：

```
/home/dave/review-site (top level)

  /htdocs

    - index.html

      /reviews

        - autohandler

        - Anna_Magdalena.html

        - Lost_and_Found.html

        - ... (one file for each review)
```

/lib

- header.mas

- footer.mas

- film_header_table.mas

index 页面相当简单。

review-site/htdocs/index.html

```
<& /lib/header.mas, title => 'review list' &>

<h1>Pick a review</h1>

<ul>

% foreach my $title (sort keys %pages) {

  <li><a href="<% $pages{$title} | h %>"><% $title | h %></a>

% }

</li>

<%init>

  my %pages;



  local *DIR;

  my $dir = File::Spec->catfile( File::Spec->curdir, 'reviews' );

  opendir DIR, $dir

        or die "Cannot open $dir dir: $!";
```

```
    foreach my $file ( grep { /\.html$/ } readdir DIR ) {

        next if $file =~ /index\.html$/;



        my $comp = $m->fetch_comp("reviews/$file")

           or die "Cannot find reviews/$file component";



        my $title = $comp->attr('film_title');



        $pages{$title} = "reviews/$file";

    }



    closedir DIR

        or die "Cannot close $dir dir: $!";

  </%init>
```

这个组件做了一个可的到的评论的列表，它基于在/home/dave/review-site/reviews 子目录下以.html 作为后缀的文件。假设实际的电影标题作为组件的一个属性保存着(通过<%attr>节) ，因此我们加载组件并查询 film_title 属性。如果这个属性不存在，则抛出一个异常。 这样处理比一个空的连接更好。如果是一个动态的站点，我们可能希望跳过这个评论，继续下一个，但是我们假设是手工执行这个脚本，可以修复这个错误。

下一步，让我们为 reviews 子路径做 autohandler，它处理显示一个评论所有重复的工作。

review-site/htdocs/reviews/autohandler

```
  <& /lib/header.mas, title => $film_title &>

  <& /lib/film_header_table.mas, comp => $m->base_comp &>
```

```
% $m->call_next;

<& /lib/footer.mas &>



<%init>

  my $film_title = $m->base_comp->attr('film_title');

</%init>
```

这个页面很简单，我们获取电影标题这样就能把它传递给 header 组件。然后，我们调用 film_header_table.mas 组件。film_header_table.mas 组件使用传递给它属性生成一个表包含电影的标题，发布年，导演和演员表。

然后我们通过 call_next()调用 review 组件本身，并以 footer 结束。

Header 部分相当直接。

review-site/lib/header.mas

```
<html>

<head>

<title><% $real_title | h %></title>

</head>

<body>



<%args>

  $title

</%args>

<%init>
```

```
    my $real_title = "Dave's Reviews - $title";

</%init>
```

这个简单的 header 生成每一个页面需要的基本的 HTML 片断。唯一的特性是基于传入的$title 参数，合成一个唯一的标题。

footer 是最简单的。

review-site/lib/footer.mas

```
<p>

<em>Copyright &copy; David Rolsky, 1996-2002</em>.

</p>



<p>

<em>All rights reserved.  No part of the review may be reproduced or

transmitted in any form or by any means, electronic or mechanical,

including photocopying, recording, or by any information storage and

retrieval system, without permission in writing from the copyright

owner.</em>

</p>



</body>

</html>
```

最后一个构建块是/lib/film_header_table.mas 组件。

review-site/lib/film_header_table.mas

```
<table width="100%">
```

```
  <tr>
    <td colspan="2" align="center"><h1><% $film_title | h %></h1></td>
  </tr>
% foreach my $field ( grep { exists $data{$_} } @optional ) {
  <tr>
    <td><strong><% ucfirst $field %></strong>:</td>
    <td><% $data{$field} | h %></td>
  </tr>
% }
</table>

<%args>
  $comp
</%args>

<%init>
  my %data;
  my $film_title = $comp->attr('film_title');
  my @optional = qw( year director cast );
  foreach my $field (@optional)    {
        my $data = $comp->attr_if_exists($field);
        next unless defined $data;
```

```
        $data{$field} = ref $data ? join ', ', @$data : $data;

    }

</%init>
```

这个组件仅构建一个表格基于传递给这个组件的属性。必需的属性是电影标题，导演和演员表是可选的。

这里有两行稍微复杂一点。第一行是：

```
% foreach my $field ( grep { exists $data{$_} } @optional ) {
```

这里我们遍历@optional 中的成员匹配在%data 的键值。我们可以简单的调用 keys %data，但是我们希望以特定的顺序显示的同时仍然跳过不存在的键值。

另外一行值的解释的是：

```
$data{$field} = ref $data ? join ', ', @$data : $data;
```

我们检查$data 是否是一个引用，以便属性可以包含一个数组的引用，这样做是有用的。比如演员表可能不只包括一个人。如果演员表是一个数组，我们把其中所有的元素用逗号连接在一起 。否则就简单的使用它。

让我们看一眼一个评论组件看起来的样子：

```
<%attr>
  film_title => 'Lost and Found'
  year => 1996
  director => 'Lee Chi-Ngai'
  cast => [ 'Kelly Chan Wai-Lan', 'Takeshi Kaneshiro', 'Michael Wong Man-Tak' ]
</%attr>



<p>
  Takeshi Kaneshiro plays a man who runs a business called Lost and
```

```
    Found, which specializes in searching for lost things and people. In
    the subtitles, his name is shown as That Worm, though that seems
    like a fairly odd name, cultural barriers notwithstanding. Near the
    beginning of the film, he runs into Kelly Chan. During their first
    conversation, she says that she has lost something. What she says
    she has lost is hope. We soon find out that she has leukemia and
    that the hope she seeks seems to be Michael Wong, a sailor who works
    for her father's shipping company.
  </p>

  <p>
    blah blah blah...
  </p>
```

这样让写一个新的评论很容易了。我们所做的是输入评论和一些属性，其余的框架是自动构建的。

剩下来的工作是生成静态的 HTML 文件。

review-site/generate_html.pl

```
  #!/usr/bin/perl -w

  use strict;    # Always use strict!

  use Cwd;
  use File::Basename;
```

```
use File::Find;

use File::Path;

use File::Spec;

use HTML::Mason;



# These are directories.    The canonpath method removes any cruft

# like doubled slashes.

my ($source, $target) = map { File::Spec->canonpath($_) } @ARGV;



die "Need a source and target\n"

      unless defined $source && defined $target;



# Make target absolute because File::Find changes the current working

# directory as it runs.

$target = File::Spec->rel2abs($target);



my $interp =

      HTML::Mason::Interp->new( comp_root => File::Spec->rel2abs(cwd) );



find( \&convert, $source );



sub convert {
```

```
# We don't want to try to convert our autohandler or .mas

# components.    $_ contains the filename

return unless /\.html$/;



my $buffer;

# This will save the component's output in $buffer

$interp->out_method(\$buffer);



# We want to split the path to the file into its components and

# join them back together with a forward slash in order to make

# a component path for Mason

#

# $File::Find::name has the path to the file we are looking at,

# relative to the starting directory

my $comp_path = join '/', File::Spec->splitdir($File::Find::name);



$interp->exec("/$comp_path");

# Strip off leading part of path that matches source directory

my $name = $File::Find::name;

$name =~ s/^$source//;



# Generate absolute path to output file
```

```
        my $out_file = File::Spec->catfile( $target, $name );

        # In case the directory doesn't exist, we make it

        mkpath(dirname($out_file));



        local *RESULT;

        open RESULT, "> $out_file" or die "Cannot write to $out_file: $!";

        print RESULT $buffer or die "Cannot write to $out_file: $!";

        close RESULT or die "Cannot close $out_file: $!";

    }
```

我们利用了 Perl 包含的 File::Find 模块，它能够递归访问一个路径结构并对每一个发现的文件调用一个回调函数。这里的回调函数 convert() 对每一个后缀是 html 的文件调用了 HTML::Mason::Interp 对象的 exec()方法。然后把组件的结果写到目标路径。

这里也用到了一些器它的模块，包括 Cwd，File::Basename，File::Path 和 File::Spec。这些模块是 Perl 自带的，提供了跨平台处理文件系统的函数。

3.7 构建一个 Mason 站点

本章节详细介绍一个实际应用的 Web 应用程序。我们的程序就是 Perl 用户网站 http://apprentice.perl.org/。那是 2001 年 O'Reilly'开放源码组织大会，Adam 提出 Perl 社区需要一个站点让拥有项目计划但没有时间或者专门技术的人公布他们的想法，从而可以让其他程序员完成剩下的部分。

具有优秀想法的但没有足够时间的有经验的开发人员可以公布项目计划，并为经验稍欠缺的开发者的实现过程提供指导。相反，一个经验稍欠缺的开发人员拥有一个非常好的想法，但并不确切的知道该如何去实现的时候，可以找到人来指导，把这个想法实现。

这是一个相当简单的基于数据库的 Web 应用程序，Marson 正好可以用来实现。它并不需要很复杂地实现，但很好地体现了一些 Mason 地特性，包括如何使用组件来隔离个人站点元素、autohandler 和 dbhandler、以及简单使用<%method>块。

数据库访问选用的是Alzabo，那是Dave Rolsky创建并维护的一个项目。Alzabo是一个基于DBI的面向对象数据库mapper。它允许我们很容易的创建Perl对象代表在数据库中的内容，比如用户或者项目。我们并不打算在这详细讨论Alzabo及相关代码，因为那是一个很大的并

跟本章节偏离的内容。如果对某段Alzabo代码不理解，你可以把你当成一段伪代码。（http://www.alzabo.org/提供更多关于Alzabo的信息，Alzabo可以从CPAN下载。

3.7.1 功能

首先要决定站点所要实现的必须功能。我们的站点相当简单，需要实现下列特性：

- 索引页：索引页需要有欢迎信息，站点新闻，以及站点管理员选定的一些代表性的项目。

- 一致的且上下文相关的菜单：站点的左手端是一个上下文相关的导航菜单。注册用户跟 guest 用户看到的是不同的选项。站点管理员可以看到另外一系列选项。然而，这些选项在页面跳转的时候是相同的。在菜单的下面显示最近加入系统的 5 个项目。

- 用户信息：一些信息是公开的，包括用户的 username 以及 email 地址（以一种改变后的方式来显示，避免机器查询邮件地址）以及他们参与的项目列表。他们的实际姓名并不显示。

- 项目浏览：因为我们并不期望非常多的子任务，因此在最开始，我们决定不创建复杂的查询机制。查询项目的两个方式：列出系统下所有的项目，或者按分类浏览。用户可以点击任一显示的项目来查看更详细的信息。

- 用户帐号：用户应该可以创建新帐户，找回忘记的密码，登陆及登出。另外，我们还允许他们修改自己的帐号。用户拥有下列属性：用户名(Username)；密码(Password)；实际姓名（Real name）；邮件地址(Email address)；状态（在线、隐身以及忙碌）；管理员标识（当前用户是否为管理员？）。

- 编辑项目：登陆用户应该可以添加新项目并编辑他们拥有管理权限的项目，包括添加及删除项目成员的能力。项目具有以下属性：名字；描述；创建日期；难度（从 1 到 10）；项目状态：想法或者实现阶段；支持程度：很多、一些或者很少（如果项目是一个指导者建立的，这就是他能提供的支持程度。如果项目是有初级开发人员建立，这意味着他们认为需要的支持程度）；链接：每个链接有一个 URL，并可以有描述；分类：每个项目有 1 个以上的类型，比如数据库、图形用户接口 GUI 等；成员：项目成员或者是开发者，或者是指导者，任一成员都可以被赋予管理员权限。

- 站点管理：网站管理员应当能编辑任何成员和项目，另外，网站管理员还能够修改项目可用的分类列表。

- 安全：细心的读者可能已经注意到密码是以普通文本放入数据库的。这意味着如果有人侵入数据系统，就可以不费力气的得到所有密码。我们认为，这样作有几个原因，即使存储为变换后的密码，能有经验侵入操作系统的人一旦从数据库得到数据也可以用词典攻击来破解这些密码。还有，我们希望能在用户询问密码的时候将实际密码通过邮件发送，这是因为我们认为这是一个安全性要求比较低的站点。通常，你不得不在安全性和便捷性进行权衡。但请不要将密码设置同你的银行帐号密码相同。

3.7.2 实现

目录规划

由于 Marson 的 autohandler 特性，在设计一个站点时，目录层次时一个很重要的考虑方面。当然，你可以从组件继承并继承其他组件，但建立一个目录规划来使需求最小化更有意义。

因为这是一个学习站点，所以我们只有一种页面样式。这是在最上层的 autohandler 去处理。子目录用来实现访问权控制。下表是我们的目录层次示意。

路径	目的
/	*包含任何用户都可以看到的大多数组件。*
/users	*包含和用户账号相关的组件，例如新用户登录。*
/project	*包含显示一个项目的唯一的 dhandler。*
/logged_in	*包含仅对已登录用户可以访问的组件，例如创建新项目。*
/admin	*包含网站管理员可以访问的组件。*
/lib	*包含其它组件使用的组件。它们不作为顶层组件使用。*

文件扩展名

我们决定为组件提供不同的扩展名。以.html 结尾的文件是 Mason 处理的顶级组件，比如/index.html。文件名以.mas 结尾的是只能被其他组件访问的，并且是不能通过浏览器访问的。另外，我们还有是以.css 结尾的 Mason 处理的文件，这是我们的设计样式。

因为这个站点没有图片，所以我们不用担心它们是否工作正常。

Apache 配置

我们的 Apache 设置假定文件和组件的根路径是相同的，/home/apprentice/htdocs。这是最简单的做法，适用于单一用途的 Web 服务器。

我们的配置文件在 httpd.conf 中，以下面这个代码开始：

PerlModule Apprentice

Apprentice.pm 模块加载应用程序所用到的所有 Perl 模块，包括各种不同的 Apache::*modules，Digest::SHAI，Time::Piece 以及其它的模块。

PerlSetVar MasonCompRoot /home/apprentice/htdocs

PerlSetVar MasonDataDir /var/mason

PerlSetVar MasonAllowGlobals $Schema

PerlAddVar MasonAllowGlobals $User

这两个对象几乎在我们项目的始终都被用到，我们把它们放到顶级 autohandler 中，而不是将它们当作产生在不同模块下传递，因为那样会很繁琐，通过使用 local()来限制它们的生命期。

<LocationMatch "(\.mas|handler)$">

SetHandler perl-script

PerlModule Apache::Constants

PerlHandler "sub { return Apache::Constants::NOT_FOUND }"

</LocationMatch>

前面提到过，所有以.html 和.css 结尾的文件由 Mason 来处理。

PerlModule HTML::Mason::ApacheHandler

<Directory /home/apprentice/htdocs>

<LocationMatch "(\.html|\.css)$">

SetHandler perl-script

PerlHandler HTML::Mason::ApacheHandler

</LocationMatch>

</Directory>

没有任何理由让外部看到我们的.mas 组件以及顶级 autohandlers 和 dhandlers，所以从安全角度考虑，我们屏蔽了对它们的访问，我们返回一个文件未找到状态，这样一些脚本或许都不知道它的存在。

这就是运行我们的网站所需要的 Apache 配置。

3.7.3 组件

现在，准备工作都已经完成，我们开始来看如何实现构建站点的组件。我们不会一行行的看代码的细节，因为那样很无趣。另外，因为一些组件在概念上都相似，我们也不去看每一个组件的源码，不再重复的解说跟前面的代码很相似的代码。但如果你不相信我们，可以在http://www.masonbook.com/得到整个站点的源码。

这个站点并没有用到 Mason 的所有特性，但这并没有什么影响。如果非要去建立一个实现所有特性的站点将会是一个畸形（并且很庞大）。而我们建立了一个尽量优美的、清洁的工作站点。这个站点上我们追求简单和教学性，本来我们可以增加更多地特性。

我们尽了最大努力来使组件中使用地 HTML 符合过渡性地 HTML4.01 标准，但有一处例外，标准禁止窗体嵌入到表中，但是 CSS 布置不兼容 Netscape4.x 以及早期版本。通常，我们不说明我们实现地组件地 HTML 部分，因为我们要讲地是 Mason 编程，而不是如何作一个好看地 HTML。

我们遵循地一个原则是任何表以及表的一部分，比如一个<tr>或者<td>标识，必须在同一个组件内，因为当在一个组件表，而在另一个组件中结束很容易造成混乱。

另外，我们尽可能使单独的组件自包含，这样单独的组件就可以包含一个或更多的完整的表。由于表可以嵌入到其他表单元中，这样从一个单独的表单元中调用组件也是安全的。

自由部分

开始一个站点的好地方是索引页和其它任何人未登录就能看见的页面。以下是可以自由访问的组件，我们会逐一进行讨论它们：

/syshandler

/news.mas

/project/dhandler

/autohandler

/featured_project.mas

/users/new_user.html

/apprentice.css

/all_projects.html

/users/user_form.mas

/left_side_menu.mas

/search_results.mas

/users/new_user_submit.html

/lib/url.mas

/lib/paging_controls.mas

/users/login_submit.html

/latest_projects.mas

/lib/redirect.mas

/users/logout.html

/lib/format_date.mas

/lib/set_login_cookie.mas

/users/forgot_password.html

/index.html

/user.html

/users/forgot_password_submit.html

/welcome.mas

/login_form.html

/show_category.html

/browse.html

这些组件构成整个站点，剩下的是一些专为登陆用户和站点管理员的模块。

/syshandler

这是一个从顶级 autohandler 继承来的组件，它的作用是创建在几乎所有页面上要用到的一些对象。这样一些组件不从 autohandler 继承，他们继承该组件从而可以使用这些对象。这很有用，因为我们的一些组件并不需要顶级 autohandler 提供的响应包装。

这个组件本身很简单，在<%once>部分，我们创建了 schema 对象$Schema，这是访问数据库的接口，大部分组件都用到了它。它同一个 DBI 数据库句柄相似，但在更高的抽象层上。因为我们在很多地方都用到它，没有必要在每个请求再新建一个对象，所以把它放在全局。

$User 对象代表当前登陆用户或者 guest 用户。因为对这两种用户的应用程序接口是一样的，组件在使用$User 时不需要关注用户是否登陆。

处理 cookie 来检查是否是合法用户,采用 SHA1 算法生成的消息验证码(Message Authentication Code,MAC)。

这是一种很普通的验证技术。当一个用户登陆，根据用户的 ID 和特征（在我们这里是一个用户输入的短语），利用 Digest::SHA1 模块产生一个特定的字符串。然后传回给用户一个 cookie 包含用户 ID 和生成的 MAC。

当用户回到站点，我们仅根据 cookie 中的用户 ID 来重新生成 MAC。当 MAC 同我们要求的匹配时，确定它是一个合法的 cookie。如果不是，则要么是 cookie 被破坏了或者有人故意欺骗。组件仅检查 cookie 的值，并不生成它。Cookie 是另外一个组件生成的，我们待会讨论它。

我们把调用 row_by_pk()方法放在 eval()内，因为这个函数会在 row 不存在时抛出异常，而我们要忽略它，这种技术在整个站点都使用了。

一旦我们拥有一个代表实际用户或者 guest 的用户对象，就调用另一个组件。通常，这是一个 autohandler 放在/autohandler。

我们使用继承标志来显式关闭组件的继承，防止从这个组件到/autohandler 的循环继承。

虽然不打算花很多时间讲 Alzabo，但需要指出的是以_t 结尾的函数返回表的对象，以_c 结尾的函数返回列的对象，以免让人迷惑。

```
<%once>

 $Schema = Apprentice::Data->schema;

</%once>

<%init>
```

```
my %cookies = Apache::Cookie->fetch;

   # A "potential row" is an object that looks like something from the

# database but that does not really exist.    However, it has the

# same interface so it is handy for things like a generic "guest"

# user.

my $guest = $Schema->User_t->potential_row( values => { username => 'Guest' } );

my $user;

if ( exists $cookies{apprentice_user_login} )

{

      my %user_info = $cookies{apprentice_user_login}->value;

         if ( $user_info{user_id} && $user_info{MAC} )

      {

            # This method of using a MAC to make sure a cookie is valid

            # is discussed in the Eagle Book.

            my $MAC = Digest::SHA1::sha1_hex

                                 ( $user_info{user_id}, $Apprentice::Secret );

               # If the cookie's MAC matches the one we generate, we know

            # that the cookie has not been tampered with.

            if ( $user_info{MAC} eq $MAC )

            {

                  # This will be a _real_ row object, representing an

                  # actual entry in the User table
```

```
                $user = eval { $Schema->User_t->row_by_pk

                                    ( pk => $user_info{user_id} ) };

            }

        }

    }

      local $User = $user || $guest;

      $m->call_next;

  </%init>

  <%flags>

   inherit => undef

  </%flags>
```

/autohandler

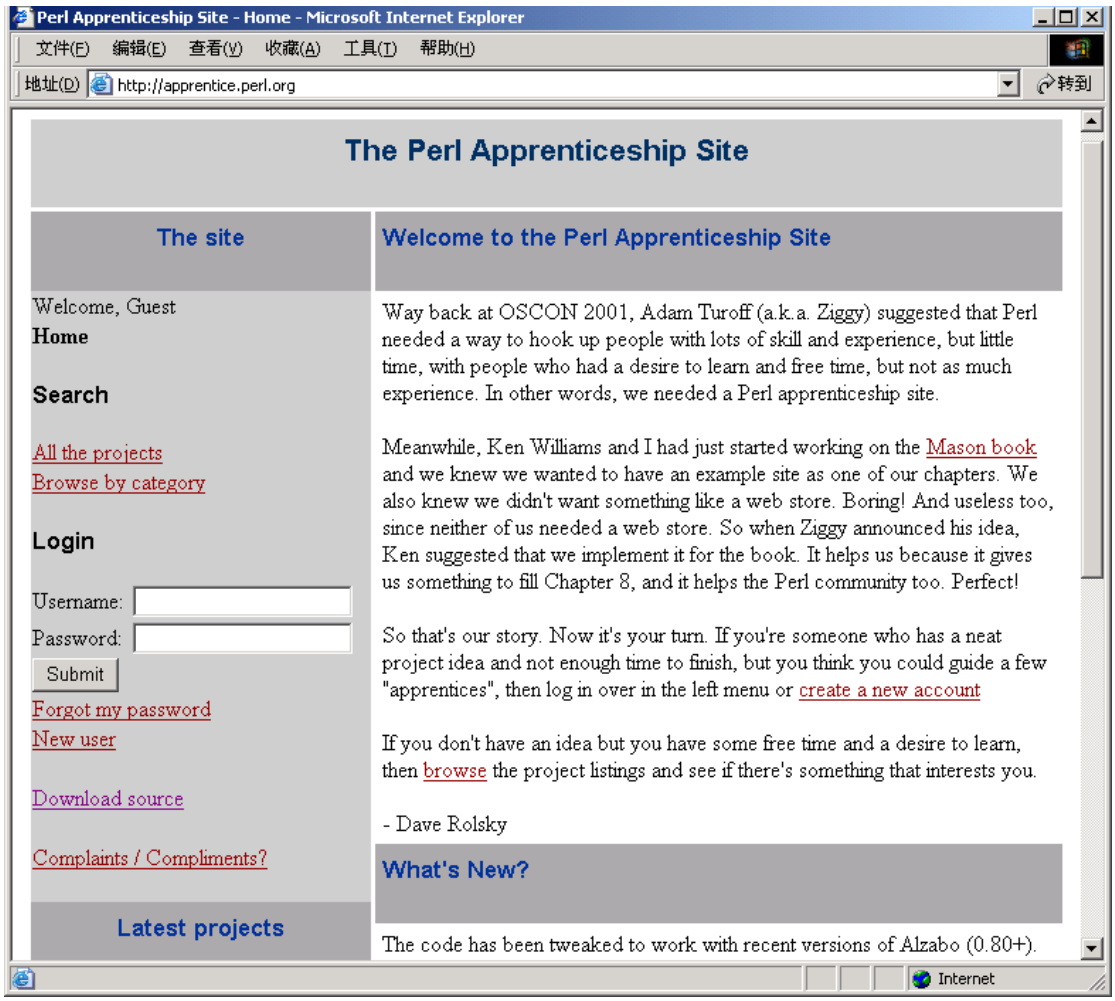
这个组件实现了查询站点的大量工作，为其他组件和函数提供代理。调用 SELF::title 允许单独的组件重写或者添加到默认标题"The Perl Apprenticeship Site"上。

我们建立了一个基本表，在页面的头部放置标题文字，并允许一些组件调用。第一个/left_side_menu.mas 组件，在页面的左面生成一个菜单，这个菜单在每个页面上都有。

接下来是/latest_projects.mas 组件，列出最近的 5 个项目，这是一种显示我们站点最新内容的好方法。

最后，我们调用请求对象的 call_next()函数把控制传给下一个组件。

autohandler 处理的部分是页面顶部的标题"The Perl Apprenticeship Site," 和左下部的所有内容。这部分内容在站点的每个页面中都基本一致。在右面 2/3 的页面是根据用户请求来生成。如下图所示。



前面提到过，/autohandler 组件是从/syshandler 组件继承来的。

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

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title><& SELF:title, %ARGS &></title>

<link rel="stylesheet" href="<& /lib/url.mas, path => '/apprentice.css' &>"

      type="text/css">

</head>

  <body bgcolor="#FFFFFF">
```

```
    <table width="100%" cellspacing="3" cellpadding="0">
  <tr valign="middle">
    <td colspan="3" bgcolor="#CCCCCC" align="center">
      <h1 class="headline">The Perl Apprenticeship Site</h1>
    </td>
  </tr>
  <tr valign="top">
    <td width="240">
<& left_side_menu.mas, %ARGS &>
<& latest_projects.mas &>
    </td>
    <td>
% $m->call_next;
    </td>
  </tr>
</table>
    </body>
</html>
<%flags>
  inherit => '/syshandler'
</%flags>
<%method title>
```

```
  Perl Apprenticeship Site

</%method>
```

/apprentice.css

Mason 并不必是用来生成 HTML 页面。这个组件生成了站点的样式模版。它是动态的，因为我们希望如果浏览器是 IE 时使用小字体，除此外就是标准文字。样式单是基于 perl.org 系列网站的（包括 http://dev.perl.org/和 http://dev.perl.org/）。

设置继承标志来确保组件不被其他 autohandler 包含。

```
/* Netscape 4 doesn't inherit from the body class so we need to

    specify everything. */

body, table, td, p, span, ul

{

   color: black; font-size: <% $font_size %>; font-family: serif

}

 h1

{ font-size: 16pt;

   font-weight: bold;

   font-family: sans-serif

}

   h1.headline

{ color: #003366;

   line-height: 200%;

   font-size: 16pt;

   font-weight: bold;
```

```
    font-family: sans-serif
}
    h2
{ font-size: 13pt;
    font-weight: bold;
    font-family: sans-serif
}
    h2.headline
{
    color: #003399;
    line-height: 150%;
    font-size: 13pt;
    font-weight: bold;
    font-family: sans-serif
}
    h3
{
    font-size: 12pt;
    font-weight: bold;
    font-family: sans-serif
}
    td.heading
```

```
{
    background-color: #AAAAAA
}
    .error
{
    color: #CC3333;
    font-size: 10pt
}
    a:vlink
{ color: #690020 }
    a:active
{ color: #003600 }
    a:hover
{ color: #696040 }
    a:link
{ color: #900000 }
    <%init>
  $r->content_type('text/css');
     # For some reason IE seems to make fonts look bigger.
  my $font_size = "10pt";
  $font_size = "9pt" if $r->header_in("User-Agent") =~ m/MSIE/;
</%init>
```

<%flags>

inherit => undef

</%flags>

/left_side_menu.mas

这个组件比前面的都长些，但并不意味更复杂。需要注意其中一些特性。

第一个是基于$User->is_logged_in()的返回值是否为真来对菜单变换。其中$User 对象是在/syshandler 组件中生成的并代表一个真实登陆用户或者 guest 用户。

如果一个用户登陆，他可以看到选项允许创建一个新的项目，编辑他拥有编辑权限的任何项目，更改他的帐户信息，以及注销登陆。编辑项目的连接只在他确实拥有权限编辑项目的时候显示。

我们用/lib/url.mas 组件来构建了所有的 URL，后面会对这个组件作解释。这个组成部分处理正确躲开的任意的复杂性的 URL 的建设。 过后，为全部 URL 使用这个组成部分将使容易加进像基于 URL 的 Session。

对注销 URL 来说，我们重新生成 URL 当前页面所具有的查询字符串。这么作是因为这个组件负责处理注销，/user/ logout.html 将使客户向后回到点击退出链接的那些页面。

回到菜单组成部分,我们能看见那些未登录用户，我们产生一个表单向/user/login_ submit.html 发送 POSTS 请求。 再次，我们将当前 URL 和查询参数 login 组件，这样用户可以回到以前的位置， 由 Cookie 表明一个成功的注册号或者一条错误信息。 那条错误信息被恰好在表单开始的地方上面处理，在那我们检查变量$login_error。

我们利用 POST 查询也可以有查询字符串，来把%caller_args 哈西表放入到查询字符串中，这样我们可以确定服务器按照正确的顺序受到 key 和 value。如果我们把 key 和 value 放在表单中作为隐藏对象，就不能保证浏览器按照预定的格式提交。

更进一步，当$User->is_admin()返回为真的话，添加上为站点管理员的一些额外的链接。

组件中<%filter>示意了过滤的一般用法。我们首先判断是否本地 URL，接着如果菜单上的链接和页面匹配，则用<b>替换来替换<a>。

我们需要特别处理 URL /index.html，因为这个页面的链接仅是<a href="/">。我们用来一个规则表达式来实现，这样如果将来我们决定在增加其他目录的链接，它也能工作良好。

<table width="100%" bgcolor="#CCCCCC" cellspacing="0" cellpadding="5">

<tr>

```
    <td colspan="2" align="center" class="heading">
      <h2 class="headline">The site</h2>
    </td>
  </tr>
</table>
<table width="100%" bgcolor="#CCCCCC" cellspacing="0" cellpadding="1">
  <tr>
    <td colspan="2">Welcome, <% $User->username %></td>
  </tr>
  <tr>
    <td colspan="2"><a href="<& /lib/url.mas, path => '/' &>">Home</a></td>
  </tr>
  <tr>
    <td colspan="2"> </td>
  </tr>
  <tr>
    <td colspan="2"><h3>Search</h3></td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas,
                  path => '/all_projects.html' &>">All the projects</a>
```

```
  </td>
 </tr>
 <tr>
  <td colspan="2">
   <a href="<& /lib/url.mas, path => '/browse.html' &>">Browse by category</a>
  </td>
 </tr>
% if ( $User->is_logged_in ) {
 <tr>
  <td colspan="2"> </td>
 </tr>
 <tr>
  <td colspan="2">
   <a href="<& /lib/url.mas,
                path => '/logged_in/new_project.html' &>">Add a new project</a>
  </td>
 </tr>
%   if ( $User->has_projects ) {
 <tr>
  <td colspan="2">
   <a href="<& /lib/url.mas,
                path => '/logged_in/editable_project_list.html' &>">
```

```
      Edit one of your projects</a>
  </td>
  </tr>
%     }
% }
  <tr>
    <td colspan="2"> </td>
  </tr>
% if ( $User->is_logged_in ) {
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas,
                    path => '/users/logout.html',
                    query => { caller_url    => $r->uri,
                               caller_args => \%query_args },
                  &>">Logout</a></td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas,
                    path => '/logged_in/edit_self.html' &>">Edit your account</a>
    </td>
```

```
  </tr>
% } elsif ( $r->uri !~ m,/login_form, ) {
  <tr>
    <td colspan="2"><h3>Login</h3></td>
  </tr>
%     if ($login_error) {
  <tr>
    <td colspan="2"><span class="error"><% $login_error | h %></td>
  </tr>
%     }
  <form action="<& /lib/url.mas,
                          path => '/users/login_submit.html',
                          query => { caller_url    => $r->uri,
                                     caller_args => \%query_args }
                        &>" method="POST">
  <tr>
    <td>Username:</td>
    <td><input type="text" name="username"></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><input type="password" name="password"></td>
```

```
  </tr>
  <tr>
    <td colspan="2"><input type="submit" value="Submit"></td>
  </tr>
  </form>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas,
                  path => '/users/forgot_password.html' &>">Forgot my password</a>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas, path => '/users/new_user.html' &>">New user</a>
    </td>
  </tr>
% }
% if ($User->is_admin) {
  <tr>
    <td colspan="2"> </td>
  </tr>
  <tr>
```

```
    <td colspan="2"><h3>Admin</h3></td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas, path => '/admin/user_list.html' &>">Edit users</a>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="<& /lib/url.mas,
                    path => '/admin/edit_categories.html' &>">Edit categories</a>
    </td>
  </tr>
% }
  <tr>
    <td colspan="2"> </td>
  </tr>
  <tr>
    <td colspan="2">
      <a href="mailto:dave@perl.org">Complaints / Compliments?</a>
    </td>
  </tr>
```

```
  <tr>
    <td colspan="2"> </td>
  </tr>
</table>
<%args>
 $username => ''
 $login_error => ''
</%args>
<%init>
 my %query_args = $m->request_args;
    # These arguments are intended for use on this page and do not need
 # to be passed through to the login_submit.html component
 delete @query_args{ 'username', 'login_error' };
</%init>
<%filter>
 (my $url = $r->uri) =~ s/index\.html$//;
 $url = $m->scomp( '/lib/url.mas', path => $url );
    s{<a href="$url">([^<]+)</a>}
   {<b>$1</b>};
</%filter>
```

/lib/url.mas

该组件的目的是利用接受的参数来构建正确格式化的查询字符串。

它不能处理嵌套数据结构或者对象，比如%query hash 的值。 对这些，我们使用一个 session 机制而不是尝试在 URL 周围传递。因为 URL 对象的 query_ form() 方法不允许 hash 引用，在传递%query 到 query_form()函数之前，我们将%query 的值转换为数组引用。目前我们还没有利用该组件允许传递的大多数参数，但这很容易并将来可以很快实现。

最后一行的反斜杠是用来确保我们不是意外地添加一个新行到 URL。

```
<%args>
 $scheme     => 'http'
 $username => undef
 $password => ''
 $host        => undef
 $port        => undef
 $path
 %query       => ( )
 $fragment => undef
</%args>
<%init>
 my $uri = URI->new;
    if ($host) {
      $uri->scheme($scheme);
        if (defined $username) {
        $uri->authority( "$username:$password" );
      }
        $uri->host($host);
     $uri->port($port) if $port;
```

```
    }

        # Sometimes we may want to path in a query string

    # but the URI module will escape the question mark.

    my $q;

        if ( $path =~ s/\?(.*)$// ) {

          $q = $1;

    }

        $uri->path($path);

        # If there was a query string, we integrate it into the query

    # parameter.

    if ($q) {

          %query = ( %query, split /[&=]/, $q );

    }

        # $uri->query_form doesn't handle hash ref values properly

    while ( my ( $key, $value ) = each %query ) {

          $query{$key} = ref $value eq 'HASH' ? [ %$value ] : $value;

    }

        $uri->query_form(%query) if %query;

        $uri->fragment($fragment) if $fragment;

  </%init>

  <% $uri->canonical | n %>\
```

/latest_projects.mas

用这个组件我们实现了显示最近新加的 5 个项目。显示这些项目的名称和创建日前。日期是有 MySQL 以'YYYY-MM-DD'格式返回的，通过/lib/format_date.mas 组件来格式化。

这是我们第一次看到项目链接。所有项目链接都是/project/<project id number>.html 的形式。显然，我们的实际文件不是像/project/10012491.html 的，这些 URL 是中间获取的。在这些链接下面我们显示系统总共存在的项目数。

为使系统能在最开始就工作正常，必须处理系统没有项目的情况。这段代码并不长但很重要。

```
<table width="100%" bgcolor="#CCCCCC" cellspacing="0" cellpadding="5">
 <tr>
  <td colspan="2" align="center" class="heading">
   <h2 class="headline">Latest projects</h2>
  </td>
 </tr>
</table>
<table width="100%" bgcolor="#CCCCCC" cellspacing="0" cellpadding="3">
% if ($count) {
%     while (my $project = $projects->next) {
 <tr>
  <td>
   <a href="<& /lib/url.mas,
                    path => '/project/' . $project->project_id . '.html' &>">
    <% $project->name | h %></a>
  </td>
  <td>
```

```
    <& /lib/format_date.mas, date => $project->creation_date, short => 1 &>
   </td>
  </tr>
%     }
  <tr>
   <td colspan="2"> </td>
  </tr>
  <tr>
   <td colspan="2">
    <% $count %> project<% $count > 1 ? 's' : '' %> in the system.
   </td>
  </tr>
% } else {
  <tr>
   <td colspan="2">No projects in the system.</td>
  </tr>
% }
</table>
<%init>
  my $count = $Schema->Project_t->row_count;
    # This grabs a list of the five most recent projects, sorted first
  # by descending creation date, and then by name in ascending.
```

```
my $projects = $Schema->Project_t->all_rows

  ( order_by => [ $Schema->Project_t->creation_date_c, 'desc',

                  $Schema->Project_t->name_c,          'asc' ],

    limit => 5,

  );
```

</%init>/lib/format_date.mas

这个简单组件从 MySQL 得到日期，并把它转换为友好的格式，可以生成短日期格式（如"Feb 4, 1970"）或者长日期格式（如"February 04, 1970"）。

选用这种格式是为了使全球的用户都能理解。一个全部是数字的格式比如"02/10/2002"会造成混淆，取决于你希望是一个美国人或者欧洲人来读日期。

更智能的站点或许允许用户作为帐户的一部分指定风格。

```
<%args>

 $date

 $short => 0

</%args>

<%init>

 my $format;

   if ( $short ) {

     $format = '%b %d, %Y';

 } else {

     $format = '%B %e, %Y';

 }

   # remove time if it exists
```

```
  $date =~ s/ .*$//;

</%init>

<% Time::Piece->strptime( $date, '%Y-%m-%d' )->strftime($format)%>\
```

/index.html

这什么也没有。索引页只是调用了一些其它组件，几乎没有提供任何内容。它重载了在/autohandler 组件中定义的 title 函数。<& PARENT:title &>函数将调用/autohandler 组件中的 title 函数，我们前面提到过，这个函数只是生成"Perl Apprenticeship Site"字符串，在这之后，我们加上" - Home"来标识索引页。

下面我们来讲一下实际生成索引页的组件。

```
<& welcome.mas &>

<& news.mas &>

<& featured_project.mas &>

  <%method title>

 <& PARENT:title &> - Home

</%method>
```

/welcome.mas

这个组件仅包含一段代码，鼓励参与站点，我们想要提供适当上下文的连接。Guest 用户应该被鼓励登录或者创建一个新帐号，而已经登录了的一个用户应该看到一个建立新工程的链接，这使网站看起来更智能些并且用 Mason 来实现也很容易。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading">

   <h2 class="headline">Welcome to the Perl Apprenticeship Site</h2>

  </td>

 </tr>
```

```
<tr>
 <td>
  <p>
  Way back at OSCON 2001, Adam Turoff (a.k.a. Ziggy) suggested that
  Perl needed a way to hook up people with lots of skill and
  experience, but little time, with people who had a desire to
  learn and free time, but not as much experience.  In other words,
  we needed a Perl apprenticeship site.
  </p>
    <p>
  Meanwhile, Ken Williams and I had just started working on the <a
  href="http://www.masonbook.com/">Mason book</a> and we knew we
  wanted to have an example site as one of our chapters.  We also
  knew we didn't want something like a web store.  Boring!  And
  useless too, since neither of us needed a web store.  So when Ziggy
  announced his idea, Ken suggested that we implement it for the
  book.  It helps us because it gives us something to fill Chapter 8,
  and it helps the Perl community too.  Perfect!
  </p>
    <p>
  So that's our story.  Now it's your turn.  If you're someone who
  has a neat project idea and not enough time to finish, but you
```

```
    think you could guide a few 'apprentices', then

% if ($User->is_logged_in) {

    <a href="<& /lib/url.mas, path => '/logged_in/new_project.html' &>">

    post your project idea</a>.

    If you're someone with an idea but you need some guidance, then you

    too can <a href="<& /lib/url.mas, path => '/logged_in/new_project.html' &>">

    post a project</a>

    and look for a mentor.

% } else {

    log in over in the left menu or

    <a href="<& /lib/url.mas, path => '/users/new_user.html' &>">create

    a new account</a>

% }

    </p>

      <p>

    If you don't have an idea but you have some free time and a desire

    to learn, then <a href="<& /lib/url.mas, path => '/browse.html' &>">

    browse</a> the project

    listings and see if there's something that interests you.

    </p>

      <p>

    - Dave Rolsky
```

```
    </p>

   </td>

  </tr>

 </table>
```

/news.mas

网站的新特性可以用该组件显示。新特性通过编辑组件文本修改。

我们通过调用 stat()得到组件文件的最后修改的时间。 我们认为组件只在有新消息的时候被更改。现在，整个站点都是新的，因此除此外没有很多消息。

```
 <table width="100%" cellspacing="0" cellpadding="5">

  <tr>

   <td class="heading"><h2 class="headline">What's New?</h2></td>

  </tr>

  <tr>

   <td>

    <p>

    The whole site, at this point.

    </p>

      <p>

    <em>Last modified: <% $last_mod %></em>

    </p>

   </td>

  </tr>

 </table>
```

```
<%init>

 my $comp_time = (stat $m->current_comp->source_file)[9];

 my $last_mod =

     Time::Piece->strptime( $comp_time, '%s' )->strftime( '%B %e, %Y %H:%M' );

</%init>
```

/featured_project.mas

该组件能用来实现特定项目的页面特征，如果站点管理员认同的话。 管理员可以仅在<%init>内修改$project_id 的值。 如果该值被设为零或者 undef，组件会在生成文本之前返回。我们可以通过这种方式根本不指定特征工程。

可以将指定特征的项目信息存放在数据库中，而且以后我们也可能采用这种方案。但现在我们尽量让它简单并保证可以拥有访问 web 服务器上组件文件的用户能够实现该功能。

当然，如果站点使用多台 web 服务器，存储项目特征的方法就会有很好的扩展性。我们可以为更多的项目提供特征。设想在<%init>内用下面的语句开头：

```
my @ids = (1, 3, 129, 440);

my $project_id = $ids[ rand @ids ];
```

现在每次生成页面时，@ids 数组中的四个项目 ID 中的一个就会调出来作为特征项目。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading" colspan="2">

   <h2 class="headline">Featured Project</h2>

  </td>

 </tr>

 <tr>

  <td>
```

```
    <h2><a href="<& /lib/url.mas,
                        path => "/project/$project_id.html" &>">
      <% $project->name | h %></a></h2>
  </td>
  <td>Created: <& /lib/format_date.mas, date => $project->creation_date &></td>
</tr>
<tr>
  <td><b>Categor<% @categories > 1 ? 'ies' : 'y' %>:</b></td>
  <td><% join ', ', @categories %></td>
</tr>
<tr>
  <td colspan="2"><h3>Members</h3></td>
</tr>
% while ( my $user = $members->next ) {
<tr>
  <td>
    <a href="<& /lib/url.mas,
                    path    => '/user.html',
                    query => { user_id => $user->user_id } &>">
      <% $user->username | h %></a>
  </td>
  <td>
```

```
%     if ($project->user_is_admin($user)) {
<b>Admin</b>
%     } else {
 
%     }
   </td>
  </tr>
% }
  <tr>
   <td colspan="2">
    <% HTML::FromText::text2html ( $project->description, paras => 1 ) %>
   </td>
  </tr>
  <tr>
   <td colspan="2">
    <p>
    Sure, it might be a dummy project but we think it is pretty cool
    stuff.  Help out!
    </p>
   </td>
  </tr>
</table>
```

```
<%init>

 my $project_id = 1;

    return unless $project_id;

    my $project = eval { $Schema->Project_t->row_by_pk( pk => $project_id ) }

       || return;

    # This grabs all of the project's members, ordered by their admin

 # status and then their username.

 my $members =

       $Schema->join( select => $Schema->User_t,

                              join     =>

                              [ $Schema->tables( 'ProjectMember', 'User' ) ],

                              where    =>

                              [ $Schema->ProjectMember_t->project_id_c, '=', $project_id ],

                              order_by =>

                              [ $Schema->ProjectMember_t->is_project_admin_c, 'desc',

                                 $Schema->User_t->username_c, 'asc' ] );

    my @categories =

       map { $_->name }

       $project->Categories( order_by => $Schema->Category_t->name_c )->all_rows;

</%init>
```

我们使用 HTML:::FromText 模块(在 CPAN 上可提供)获取工程的文本描述并且把它变成 HTML。通过设 paras 为 1 来指定文本是以段落为单位，这样就可以解析为正确的 HTML 标记。

/all_projects.html

该组件实际上把大多数工作委派给/search_results.mas 组件。该组件所作的工作就是创建一个代表需查询的行的游标。在这，仅需将查询设为'all projects'。我们利用 MySQL 的限制偏移来只查询我们关心的行。我们马上可以看到/search_results 显示的结果，每页 20 行。

此外，该组件还需要获知未限制时查询到的记录总数。该组件还对执行的查询生成一个文本描述显示给用户。

参数$start 和$limit 是结果页面系统的一部分。为了使分页系统工作，任何实现查询的组件都应接受这两个参数。

```
<& search_results.mas,

    count => $count,

    projects => $projects,

    summary => $summary,

    start => $start,

    limit => $limit,

    %ARGS

 &>

<%args>

 $start => 0

 $limit => 20

</%args>

<%init>

 my $summary = 'all projects';

    my $count = $Schema->Project_t->row_count;

    my $projects =
```

```
        $Schema->Project_t->all_rows

            ( order_by =>

              [ $Schema->Project_t->creation_date_c, 'desc',

                $Schema->Project_t->name_c,          'asc' ],

              limit => [ $limit, $start ],

            );

</%init>

<%method title>

 <& PARENT:title &> - All projects

</%method>
```

/search_results.mas

这是实际实现结果显示的地方。目前只有两个组件用到它，但是它是被设计成如果我们增加更多的搜寻选择，例如一个关键字搜寻，也能处理。

该组件用$summary 和$count 参数来告知用户他选择执行了何种查询以及总共有多少结果记录。如果结果在一页上显示不完，就调用/lib/paging_controls.mas 组件来生成到其他结果页面的链接。

最后，如果有结果，就在游标和相应项目的显示信息之间循环。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading" colspan="4">

   <h2 class="headline">Search Results</h2>

  </td>

 </tr>

 <tr>
```

```
    <td colspan="4">
     You searched for <% $summary | h %>.
     There <% $count == 1 ? 'is' : 'are' %> <% $count %>
     result<% $count != 1 ? 's' : '' %>.
    </td>
  </tr>
% if ($count > $limit) {
  <tr>
    <td colspan="4">
<& /lib/paging_controls.mas, %ARGS &>
    </td>
  </tr>
% }
% if ($count) {
  <tr>
    <td width="40%"><b>Name</b></td>
    <td width="30%"><b>Created on</b></td>
    <td width="15%"><b>Difficulty</b></td>
    <td width="15%"><b>Project status</b></td>
  </tr>
%     while (my $project = $projects->next) {
  <tr>
```

```
    <td>
      <a href="<& /lib/url.mas,
                    path => '/project/' . $project->project_id . '.html' &>">
        <% $project->name | h %></a>
    </td>
    <td><& /lib/format_date.mas, date => $project->creation_date &></td>
    <td><% $project->difficulty %></td>
    <td><% $project->status %></td>
  </tr>
% }
% }
</table>
<%args>
 $count
 $projects
 $summary
 $start
 $limit
</%args>
```

/lib/paging_controls.mas

在 Web 应用程序中生成分页的查询结果是一个共同需求。如果你有成百上千的数据库，或者搜寻项目更多，你需要一种方法来处理大的结果集。 做这的这种通常的方法是把结果分为多页，每页显示其他页的连接。

该组件生成其他页的链接，看起来像这样，

<< 1 2 3 4 5 6 7 8 >>

" < <"向前移动一页，" >>"向后移动一页。 用户目前正观看的页有粗体的正文不作为连接。 如果用户在首页或者尾页上，向前或者下一页连接不被显示。

当你有 100 页的时候，需要另一种层次的导航，所以我们用下面这种方式：

... << 21 22 23 24 25 26 27 28 29 30 >> ...

第一个"..."链接将回到上一个 10 个页面的页面组，结尾的"..."链接将前进到下 10 个页面的组，在本例中，进入 31 页。

这种设计可以较好地处理大量页面，但如果你通常生成地结果包含上千个页面，你或许希望增加另外一种导航方式允许前进或者后退更多页。

我们关注于该组件如何生成链接。我们采用 Apache 的请求对象的 uri()函数来判断当前 URL 而不是要求将 URL 传入到该组件。我们使用$m->request_args()函数来找出传递给该页面的参数。这么作是因为我们要从新生成用户生成的参数，而不是组件上次被调用时的参数。我们删除了 limit 和 start 参数，因为在每个链接中都会重载。

```
<table width="100%">
  <tr>
    <td>Displaying results <% $start + 1 %> - <% $last_shown %>.</td>
  </tr>
</table>
<table width="100%">
  <tr>
    <td width="7%">
% if ( $previous_tenth >= 10 ) {
      <a href="<& /lib/url.mas,
                  path => $r->uri,
```

```
                    query => { start => ($previous_tenth - 1) * $limit,
                               limit => $limit,
                               %query }
                &>"><a></a>
% } else {
     
% }
  </td>
  <td width="7%">
% if ( $current_page > 1 ) {
    <a href="<& /lib/url.mas,
                    path => $r->uri,
                    query => { start => $start - $limit,
                               limit => $limit,
                               %query }
                &>"><a></a>
% }
  </td>
% foreach  my $page ( ($previous_tenth + 1)..($next_tenth - 1) ) {
%   if ( $page <= $total_pages ) {
  <td width="7%">
%     if ( $page != $current_page ) {
```

```
    <a href="<& /lib/url.mas,
                    path => $r->uri,
                    query => { start => ($page - 1) * $limit,
                               limit => $limit,
                               %query }
                  &>"><% $page %></a>
%       } else {
    <b><% $page %></b>
%       }
%    } else {
     
%    }
% }
   </td>
   <td width="7%">
% if ( $current_page < $total_pages ) {
    <a href="<& /lib/url.mas,
                    path => $r->uri,
                    query => { start => $start + $limit,
                               limit => $limit,
                               %query }
                  &>">&gt;&gt;</a>
```

```
% } else {

     

% }

  </td>

  <td width="7%">

% if ( $next_tenth <= $total_pages ) {

    <a href="<& /lib/url.mas,

                path => $r->uri,

                query => { start => ($next_tenth - 1) * $limit,

                           limit => $limit,

                           %query }

              &>">...</a>

% } else {

     

% }

  </td>

 </tr>

</table>

<%args>

 $start

 $limit

 $count
```

```
</%args>

<%init>

 my %query = $m->request_args;

 delete @query{ 'start', 'limit' };

 my $total_pages = int( $count / $limit );

 $total_pages++ if $count % $limit;

    my $current_page = ( $start / $limit ) + 1;

   my $previous_tenth =

       $current_page -

       ( $current_page % $limit ? $current_page % $limit : $limit );

    my $next_tenth = $previous_tenth + 11;

    my $last_shown = $start + $limit > $count ? $count : $start + $limit;

</%init>
```

/browse.html

该页面通过全部不同的工程种类重复。 如果一个种类有工程， 然后我们产生连接浏览那个种类。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading"><h2 class="headline">Browse by Category</h2></td>

 </tr>

% while (my $category = $categories->next) {

 <tr>

  <td>
```

```
%     if (my $count = $category->project_count) {
    <a href="<& /lib/url.mas,
                    path    => 'show_category.html',
                    query => { category_id => $category->category_id } &>">
      <% $category->name | h %></a>
    (<% $count %> project<% $count > 1 ? 's' : '' %>)
%     } else {
    <% $category->name | h %> (No projects)
%     }
   </td>
  </tr>
% }
</table>
<%init>
 my $categories =
      $Schema->Category_t->all_rows( order_by => $Schema->Category_t->name_c );
</%init>
<%method title>
 <& PARENT:title &> - Browse by category
</%method>
```

/show_category.html

这就是/browse.html 为每个分类提供的链接。代码跟在/all_projects.html 看到的很相似，都用到相同的组件/search_results.mas 来完成实际的工作。

该组件的一个新特性是 title 函数动态地在页面标题增加分类名称。 我们使用A<%shared>在这里是为了避免再次创造相同的分类对象。如果给的我们的种类 ID 是无效的，然后我们仅仅使用户使回到主页。它是懒惰的，但是这比仅仅显示一条错误信息好。

```
<& search_results.mas,

    count => $count,

    projects => $projects,

    summary => $summary,

    start => $start,

    limit => $limit,

    %ARGS

 &>

<%shared>

 my $category =

      eval { $Schema->Category_t->row_by_pk

                  ( pk => $m->request_args->{category_id} ) }

           || $m->comp( '/lib/redirect.mas', path => '/' );

</%shared>

<%args>

 $start => 0

 $limit => 20

 $category_id

</%args>

<%init>
```

```
    my $summary = 'projects in the "' . $category->name . '" category';

    my $count = $category->project_count;

    my $projects =

      $Schema->join( select => $Schema->Project_t,

                     join     =>

                     [ $Schema->tables( 'Project', 'ProjectCategory' ) ],

                     where    =>

                     [ $Schema->ProjectCategory_t->category_id_c, '=',

                       $category_id ],

                     order_by =>

                     [ $Schema->Project_t->creation_date_c, 'desc',

                       $Schema->Project_t->name_c,          'asc' ],

                     limit => [ $limit, $start ],

                   );

</%init>

  <%method title>

 <& PARENT:title &> - <% $category->name | h %> projects

</%method>
```

/user.html

这是我们的用户信息显示组件，没有什么新的内容，创建对象，显示对象的信息。但该处不是复制其他组件的代码，只是跟它们很相似。

/project/dhandler

该组件同/user.html 相当相似，但不是通过一个查询字符串调用而是用一个像

/project/77.html 的 URL 来调用，其中 77 是项目 ID。在这使用 dhandler 是一个随意的选择，但可以使我们拥有看起来好看的便于搜索的 URL。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading" colspan="2">

   <h2 class="headline"><% $project->name | h %></h2>

  </td>

 </tr>

 <tr>

  <td colspan="2">

   Created: <& /lib/format_date.mas, date => $project->creation_date &>

  </td>

 </tr>

 <tr>

  <td colspan="2">

   <% HTML::FromText::text2html ( $project->description, paras => 1 ) %>

  </td>

 </tr>

 <tr>

  <td><b>Categor<% @categories > 1 ? 'ies' : 'y' %>:</b></td>

  <td><% join ', ', @categories %></td>

 </tr>

 <tr>
```

```
    <td><b>Project status:</b></td>
    <td><% $project->status | h %></td>
  </tr>
  <tr>
    <td><b>Support level:</b></td>
    <td><% $project->support_level | h %></td>
  </tr>
  <tr>
    <td colspan="2"><h3>Members</h3></td>
  </tr>
% while (my $user = $members->next) {
  <tr>
    <td>
      <a href="<& /lib/url.mas,
                  path    => '/user.html',
                  query => { user_id => $user->user_id } &>">
        <% $user->username | h %></a>
    </td>
    <td>
%     if ($project->user_is_admin($user)) {
<b>Admin</b>
%     } else {
```

```
 
%     }
   </td>
  </tr>
% }
% if ( $Schema->ProjectLink_t->row_count
%                ( where => [ $Schema->ProjectLink_t->project_id_c, '=', $project_id ] ) ) {
  <tr>
   <td colspan="2"> </td>
  </tr>
  <tr>
   <td colspan="2"><h3>Links</h3></td>
  </tr>
%     while (my $link = $links->next) {
  <tr>
   <td colspan="2">
    <a href="<% $link->url %>"><% $link->description | h %></a>
   </td>
  </tr>
%     }
% }
% if ($User->is_admin || $User->is_project_admin($project)) {
```

```
  <tr>
   <td colspan="2"> </td>
  </tr>
  <tr>
   <td colspan="2">
    <a href="<& /lib/url.mas,
                path    => '/logged_in/edit_project.html',
                query => { project_id => $project->project_id } &>">
    Edit this project</a>
   </td>
  </tr>
% }
</table>
<%shared>
  my ($project_id) = $m->dhandler_arg =~ /(\d+).html/;
  my $project = eval { $Schema->Project_t->row_by_pk( pk => $project_id ) }
        || $m->comp( '/lib/redirect.mas', path => '/' );
</%shared>
<%init>
 my $links = $project->Links( order_by => $Schema->ProjectLink_t->url_c );
    my $members =
       $Schema->join( select => $Schema->User_t,
```

```
                join     =>
                [ $Schema->tables( 'ProjectMember', 'User' ) ],
                where    =>
                [ $Schema->ProjectMember_t->project_id_c, '=', $project_id ],
                order_by =>
                [ $Schema->ProjectMember_t->is_project_admin_c, 'desc',
                  $Schema->User_t->username_c,                  'asc' ] );
    my @categories =
      map { $_->name }
      $project->Categories( order_by => $Schema->Category_t->name_c )->all_rows;
</%init>
  <%method title>
 <& PARENT:title &> - <% $project->name | h %>
</%method>
```

/login_form.html

这是一个简单的登陆表单，把它所接受到的各种参数传递给/users/login_submit.html 组件，比如$success_url 和%success_args。

```
<table width="100%" cellspacing="0" cellpadding="5">
  <tr>
    <td class="heading" colspan="2"><h2 class="headline">Login</h2></td>
  </tr>
% if ($message) {
  <tr>
```

```
    <td colspan="2"><% $message | h %></td>
  </tr>
% }
% if ($login_error) {
  <tr>
    <td colspan="2"><% $login_error | h %></td>
  </tr>
% }
  <form action="<& /lib/url.mas,
                        path => '/users/login_submit.html',
                        query => { caller_url      => $r->uri,
                                   success_url     => $success_url,
                                   success_args => \%success_args }
                      &>" method="POST">
  <tr>
    <td>Username:</td>
    <td><input type="text" name="username" value="<% $username | h %>"></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><input type="password" name="password"></td>
  </tr>
```

```
    <tr>
   <td colspan="2"><input type="submit" value="Submit"></td>
  </tr>
  </form>
 </table>
 <%args>
  $message => undef
  $login_error => undef
  $success_url => '/'
  %success_args => ( )
  $username => ''
 </%args>


 <%method title>
  <& PARENT:title &> - Login
 </%method>
```

/users/new_user.html

该组件把大部分工作委派给/users/user_form.mas 组件来完成实际生成表单的工作。

$new_user 对象代表一个可能的数据库记录，同真实的 user 对象一样具有相同的 API。但是，一条准记录并不对应数据库中的实际数据。simplifies 创建/users/user_form.mas 组件，该组件使用记录对象 API，而不论是一个新用户还是在编辑一个已经存在的用户。

$available_status 对象代表 UserStatus 表中状态为'Available'的实际记录对象。

```
 <table width="100%" cellspacing="0" cellpadding="5">
```

```
 <tr>
  <td class="heading" colspan="2"><h2 class="headline">New User</h2></td>
 </tr>
<& user_form.mas, submit_to => 'new_user_submit.html', user => $new_user, %ARGS &>
</table>
<%init>
 my $available_status =
      $Schema->UserStatus_t->one_row
           ( where =>
             [ $Schema->UserStatus_t->column('status'), '=', 'Available' ] );
    my $new_user =
      $Schema->User_t->potential_row
           ( values =>
             { username           => '',
               password           => '',
               real_name          => '',
               email_address      => '',
               user_status_id => $available_status->user_status_id,
             } );
</%init>
<%method title>
 <& PARENT:title &> - New user
```

```
</%method>
```

/users/user_form.mas

这个表单适用于创建新用户和编辑已经存在的用户。它首先查看%ARGS 哈西表来准备更新表单区域。 如果这里这些领域有值，它以为这些有优先权， 因为唯一%ARGS 有值的情况是那些提交然而被验证数据所拒绝，无论在哪种情况下浏览器都回到提交页面。当发生这个情况时，我们想要让用户看到被拒绝的值。如果没有，我们为这些值查找$user 对象。

除非生成页面的用户是一个管理员，不然我们不显示标识管理员的复选框，因为复选框只在管理员提交该表单的时候才起作用。

$submit_to 变量用来设定表单的行为属性。这使我们可以用这个表单来创建新用户或者编辑老用户。

$return_to 变量只是把表单传递给处理表单的组件，那用来确定如果提交成功将浏览器设在哪个页面。

```
% foreach my $err (@errors) {
  <tr>
    <td colspan="2"><span class="error"><% $err | h %></td>
  </tr>
% }
  <form action="<& /lib/url.mas, path => $submit_to &>" method="POST">
  <input type="hidden" name="return_to" value="<% $return_to %>">
% if ($user->user_id) {
  <input type="hidden" name="user_id" value="<% $user->user_id %>">
% }
  <tr>
    <td>Username:</td>
    <td>
```

```
    <input type="text" name="username"
           value="<% $form_vals{username} | h %>" size="20" maxlength="30">
  </td>
</tr>
<tr>
  <td>Password:</td>
  <td>
    <input type="password" name="password"
           value="<% $form_vals{password} | h %>" size="20" maxlength="100">
  </td>
</tr>
<tr>
  <td>Confirm password:</td>
  <td>
    <input type="password" name="password2"
           value="<% $form_vals{password2} | h %>" size="20" maxlength="100">
  </td>
</tr>
<tr>
  <td>Real name:</td>
  <td>
    <input type="text" name="real_name"
```

```
            value="<% $form_vals{real_name} %>" size="20" maxlength="75">
      </td>
    </tr>
    <tr>
      <td>Email address:</td>
      <td>
        <input type="text" name="email_address"
            value="<% $form_vals{email_address} %>" size="20" maxlength="150">
      </td>
    </tr>
    <tr>
      <td>How available are you?</td>
      <td>
        <select name="user_status_id">
  % while (my $status = $user_statuses->next) {
          <option value="<% $status->user_status_id %>"
            <% $form_vals{user_status_id} == $status->user_status_id ? 'selected="selected"' :
''%>>
            <% $status->status | h %>
          </option>
  % }
        </select>
      </td>
```

```
  </tr>
% if ($User->is_admin) {
  <tr>
    <td>Site admin:</td>
    <td>
      <input type="checkbox" name="is_admin"
                value="1" <% $form_vals{is_admin} ? 'checked="checked"': '' %>>
    </td>
  </tr>
% }
  <tr>
    <td colspan="2"><input type="submit" value="Submit"></td>
  </tr>
  <form>
<%args>
  $submit_to
  $return_to => '/'
  $user
  @errors => ( )
</%args>
<%init>
  my $user_statuses =
```

```
        $Schema->UserStatus_t->all_rows
            ( order_by => $Schema->UserStatus_t->status_c );


    my %form_vals;
    foreach my $field ( qw( username password real_name email_address
                            user_status_id is_admin ) ) {
        $form_vals{$field} =
            exists $ARGS{$field} ? $ARGS{$field} : $user->$field( );
    }


    $
  form_vals{password2} =
        exists $ARGS{password2} ? $ARGS{password2} :
        exists $ARGS{password} ? $ARGS{password} :
        $user->password;
    </%init>
```

/users/new_user_submit.html

因为数据验证在模块中已经处理了，该组件没有实现很多内容。如果插入成功，设置 cookie 来标识一个成功的登陆，并将客户端重定向到变量$return_to 中的路径。需要注意的是，我们只在提交的用户是一个站点管理员的时候才将 is_admin 标识设为真。

该组件调用了其他一些组件，但它使用$m->comp()而不是<& &>来实现。这部分是因为在<%init>内调用组件很方便，另外需强调的是那些组件不是由 HTML 输出生成的。

```
    <%args>
    $return_to
```

```
</%args>

<%init>

 # When inserting a new row, data validation checks are performed and an

 # exception is thrown if any of the checks fail.

 my $user =

       eval { $Schema->User_t->insert

                     ( values =>

                       { ( map { $_ => $ARGS{$_} }

                             qw( username password password2

                                 real_name email_address

                                 user_status_id ) ),

                             is_admin    => $User->is_admin ? $ARGS{is_admin} : 0,

                       }

                     );

              };


 # One or more data validation checks failed

 $m->comp( '/lib/redirect.mas',

              path => 'new_user.html', query => { %ARGS, errors => $@->errors } )

       if $@ && UNIVERSAL::isa( $@, 'Apprentice::Exception::DataValidation' );


 # Some other unforeseen error happened
```

```
    die $@ if $@;



    $m->comp( '/lib/set_login_cookie.mas', user => $user );



    $m->comp( '/lib/redirect.mas', path => $return_to );

  </%init>

  <%flags>

    inherit => '/syshandler'

  </%flags>
```

/lib/redirect.mas

利用 Mason 内建的 redirect()函数，这个组件很简单。我们用 scomp()函数来获取从/lib/url.mas 组件表单字符串中的 URL，然后传递给 redirect()函数，它会生成合适的头信息并发送回客户。

```
  <%init>

    my $url = $m->scomp( '/lib/url.mas', %ARGS );

      $m->redirect($url);

  </%init>
```

/users/login_submit.html

该组件是我们在/left_side_menu.mas 和/login_form.html 中看到的登陆表单提交的目标对象。

我们检查指定的用户名称来确认它存在并检查密码是否于数据库中的匹配。如果不匹配，我们就将用户重定向到调用页并给出一个错误。如果匹配的话，我们设置 cookie 的值标识一个成功的链接并且重定向到$success_url 指定的 URL。这是 Web 应用程序的常用模式。利用一个 URL 来处理提交需将浏览器重新定向到另一个页面，这样你可以使提交接收组件可以接受对客户请求重定向的参数。

```
  <%args>
```

```
$username

$password

$caller_url

%caller_args => ( )

$success_url => undef

%success_args => ( )

</%args>

<%init>

my $user =

    $Schema->User_t->one_row

        ( where => [ $Schema->User_t->username_c, '=', $username ] );

  unless ( $user && $password eq $user->password ) {

    $m->comp( '/lib/redirect.mas',

                path    => $caller_url,

                query => { caller_args => \%caller_args,

                           username => $username,

                           login_error => 'Invalid login.' },

              );

}

  $m->comp( '/lib/set_login_cookie.mas', user => $user );

 # By default, we just send them back to the calling page.

$success_url = $caller_url unless defined $success_url && length $success_url;
```

```
    %success_args = %caller_args unless %success_args;

      $m->comp( '/lib/redirect.mas', path => $success_url, query => \%success_args );

  </%init>

  <%flags>

    inherit => '/syshandler'

  </%flags>
```

/lib/set_login_cookie.mas

在解释/syshandler 组件时，我们讨论了使用 MAC 用于验证。在这里我们简单的设置包含用户的 user Id 的 Cookie 和基于这个用户 ID 的 MAC。

影响发送到客户端的头的组件，比如像这个，必须在发送头之前被调用。因为该站点关闭了 autoflush，所以这不是一个问题，因为我们在内容产生之后才发送头。

```
  <%args>

    $user

  </%args>

  <%init>

    Apache::Cookie->new

        ( $r,

          -name    => 'apprentice_user_login',

          -value => { user_id => $user->user_id,

                      MAC =>

                      Digest::SHA1::sha1_hex

                          ( $user->user_id, $Apprentice::Secret ) },

          -path    => '/',
```

```
            -domain    => 'apprentice.perl.org',
            -expires => '+1M',
          )->bake;
    </%init>
```

/users/logout.html

这里我们删除了/lib/set_login_cookie.mas 设置的 cookie，通过设置 cookie 为过去的日期，浏览器会将该 cookie 删除。

```
    <%args>
     $caller_url
     %caller_args => ( )
    </%args>
    <%init>
     Apache::Cookie->new
          ( $r,
            -name    => 'apprentice_user_login',
            -value => '',
            -path    => '/',
            -domain    => 'apprentice.perl.org',
            -expires => '-1d',
          )->bake;
        $m->comp( '/lib/redirect.mas', path => $caller_url, query => \%caller_args );
    </%init>
    <%flags>
```

```
    inherit => '/syshandler'

  </%flags>
```

/users/forgot_password.html

这是一个简单的用户用来取回密码的表单，用户输入用户名称，系统发送邮件告诉他密码。

/users/forgot_password_submit.html

该组件完成实际的发送邮件给忘记密码的用户。假设用户输入的用户名同数据库中相匹配，我们就生成一个简单的邮件告诉他密码。

我们用$r->register_cleanup()函数来延迟发送邮件知道输出发送到客户端之后。这种技术对于耗时很长的操作很有用，但缺点是如果操作失败，不能很方便地通知客户。要解决这个问题，你必须在用户等待输出的同时发送。$r->register_cleanup()在 Apache 模块文档中有介绍。

```
  <%args>

    $username

  </%args>

  <%init>

    my $user =

        $Schema->User_t->one_row

            ( where => [ $Schema->User_t->username_c, '=', $username ] );



    unless ( $user ) {

        $m->comp( '/lib/redirect.mas',

                  path => 'forgot_password.html',

                  query => { error => 'Invalid username.' } );

    }
```

```
    my $body = "Your password is:\n\n" . $user->password .
                "\n\nwebmaster\@apprentice.perl.org";
    $r->register_cleanup
      ( sub { Apprentice::send_email
                ( to      => $user->email_address,
                  from => 'webmaster@apprentice.perl.org',
                  subject => 'Your password for apprentice.perl.org',
                  body => $body ) } );

  $m->comp( '/lib/redirect.mas',
            path => '/index.html',
            query => { login_error => 'Your password has been mailed to you.' } );
</%init>
<%flags>
 inherit => '/syshandler'
</%flags>
```

带访问控制的组件

前面那些组件是任何访问者都可以看到的组件，没有登陆要求。剩下的组件分为两个目录：一个是为登录用户；另一个是为站点管理员。只有在用户登陆之后才可用，它们是：

/logged_in/autohandler

/lib/check_access_level.mas

/logged_in/edit_self.html

/logged_in/edit_user_submit.html

/logged_in/new_project.html

/logged_in/project_form.mas

/logged_in/new_project_submit.html

/logged_in/editable_project_list.html

/logged_in/edit_project.html

/logged_in/check_access_to_project.mas

/logged_in/edit_project_submit.html

/logged_in/edit_members.html

/logged_in/add_project_member.html

/logged_in/remove_project_member.html

/logged_in/delete_project.html

这些组件都是用来编辑站点上的内容。

/logged_in/autohandler

该组件的作用就是来检验用户信息，如果不是一个已登陆用户，就不能访问该目录下的任何组件。

```
<%init>

 $m->comp( '/lib/check_access_level.mas', level => 'is_logged_in' );

   $m->call_next;

</%init>/lib/check_access_level.mas
```

如果不满足访问权限要求，该组件将用户重定向到登陆表单。如果用户登陆信息正确，则会重定向到他开始被阻止访问的页面。

```
<%args>
```

```
 $level

</%args>

<%init>

 my $requested_url = $r->uri;

 my %query_args = $m->request_args;

 my $level_description = $level eq 'is_logged_in' ? 'a logged-in' : 'an admin';

   $m->comp( '/lib/redirect.mas',

             path => '/login_form.html',

             query => { message => "This area requires $level_description user.",

                        success_url    => $requested_url,

                        success_args => \%query_args,

                      } )

    unless $User->$level( );

</%init>
```

/logged_in/edit_self.html

前面对用户信息的编辑使用/users/user_form.mas 组件，这次我们用表单的另一种行为特性，通过 submit_to 参数来设置。并不比以前的简单。

```
<table width="100%" cellspacing="0" cellpadding="5">

 <tr>

  <td class="heading" colspan="2">

   <h2 class="headline">Edit Your Account</h2>

  </td>

 </tr>
```

```
<& /users/user_form.mas,
    submit_to => 'edit_user_submit.html',
    return_to => $r->uri,
    user => $User,
    %ARGS
 &>
</table>

<%method title>
 <& PARENT:title &> - Edit your account
</%method>
```

/logged_in/edit_user_submit.html

该组件执行额外的访问权检查。我们要确保提交这个表单的要么是网站管理员或者帐户所有者。并且当创建新用户时，我们只在提交者是一个网站管理员时才不把 is_admin 标识设置为 false。

```
<%args>
 $user_id
 $return_to
</%args>
 <%init>
 $m->comp( '/lib/redirect.mas', path => '/' )
     unless $User->is_admin or $User->user_id == $user_id;
    my $user =
```

```
        eval { $Schema->User_t->row_by_pk( pk => $user_id ) }

            || $m->comp( '/lib/redirect.mas', path => '/' );

    eval {

        $user->update( ( map { $_ => $ARGS{$_} }

                                qw( username password password2

                                    real_name email_address

                                    user_status_id ) ),

                               is_admin    => $User->is_admin ? $ARGS{is_admin} : 0,

                            );

    };

      $m->comp( '/lib/redirect.mas',

                path => $return_to, query => { %ARGS, errors => $@->errors } )

        if $@ && UNIVERSAL::isa( $@, 'Apprentice::Exception::DataValidation' );

      die $@ if $@;

      $m->comp( '/lib/redirect.mas', path => $return_to );

  </%init>

    <%flags>

   inherit => '/syshandler'

  </%flags>
```

/logged_in/new_project.html

项目的创建和编辑页面同创建和编辑用户的很相似。我们可以利用 Mason 的组件系统来获得较高层次的复用。

/logged_in/project_form.mas

该页面同/users/user_form.mas 极为相似。我们再次需要处理编辑项目或者创建一个默认设置的新项目时，表单初始化已存在的值。还要考虑到当数据验证的时候发生错误也会回到这来，这样我们希望保留用户提交的值。

我们还是用$submit_to 参数来设置表单，跟设置用户信息表单一样。

该组件代码更多些，因为项目比用户信息要多。项目可以属于多个分类，零个或者多个不同的描述链接。

链接的处理方式比较有趣，我们需要一种方式来区分编辑或者删除一个存在的链接还是增添。我们通过给表单项目不同的名称来实现。对于存在的链接，包含我们单独存储的链接的 ID，这样我们可以在/logged_in/edit_project_submit.html 组件中用到。稍后我们会讨论到。

```
% foreach my $err (@errors) {
  <tr valign="top">
    <td colspan="2"><span class="error"><% $err | h %></td>
  </tr>
% }
  <form action="<& /lib/url.mas, path => $submit_to &>" method="POST">
% if ($project->project_id) {
  <input type="hidden" name="project_id" value="<% $project->project_id %>">
% }
  <tr valign="top">
    <td>Name:</td>
    <td>
      <input type="text" name="name"
             value="<% $form_vals{name} | h %>" size="20" maxlength="30">
    </td>
  </tr>
```

```
  <tr valign="top">
    <td>Description:</td>
    <td>
      <textarea name="description" rows="5" cols="40">\
<% $form_vals{description} | h %>\
</textarea>
    </td>
  </tr>
  <tr valign="top">
    <td>Categories<br>(1 or more):</td>
    <td>
      <select name="category_ids" multiple="1" size="4">
% while (my $category = $categories->next) {
        <option value="<% $category->category_id %>"
          <% $current_categories{ $category->category_id } ?
              'selected="selected"' : '' %>>
         <% $category->name | h %>
        </option>
% }
      </select>
    </td>
  </tr>
```

```
  <tr valign="top">
   <td>Difficulty:</td>
   <td>
    <select name="difficulty">
% foreach (1..10) {
      <option value="<% $_ %>"
       <% $form_vals{difficulty} == $_ ? 'selected="selected"' : '' %>>
       <% $_ %>
      </option>
% }
    </select>
   </td>
  </tr>
  <tr valign="top">
   <td>Status:</td>
   <td>
    <select name="project_status_id">
% while (my $status = $statuses->next) {
      <option value="<% $status->project_status_id %>"
       <% $status->project_status_id == $form_vals{project_status_id} ?
           'selected="selected"' : '' %>>
       <% $status->status %>
```

```
      </option>
% }
    </select>
   </td>
  </tr>
% unless ($member_count) {
  <tr valign="top">
   <td>
   My role will be:</td>
   <td>
    <select name="role_id">
%     while (my $role = $roles->next) {
      <option value="<% $role->role_id %>"
       <% $form_vals{role_id} == $role->role_id ? 'selected="selected"': '' %>>
       <% $role->role | h %>
      </option>
%     }
   </td>
  </tr>
% }
  <tr valign="top">
   <td colspan="2">
```

```
    <p>
    If you chose the 'Mentor' role, then this is the
    support level you will provide.  If you chose the
    'Apprentice' role, then this is the support level you
    think you require.
    </p>
   </td>
  </tr>

  <tr valign="top">
   <td>Support level:</td>
   <td>
    <select name="project_support_level_id">
% while (my $level = $support_levels->next) {
      <option value="<% $level->project_support_level_id %>"
       <% $level->project_support_level_id ==
          $form_vals{project_support_level_id} ?
          'selected="selected"' : '' %>>
       <% $level->support_level %>
      </option>
% }
    </select>
```

```
  </td>
 </tr>
 <tr valign="top">
  <td colspan="2">
   <table width="100%" cellpadding="0">
    <tr valign="top">
     <td colspan="2"><h3>Links</h3></td>
    </tr>
    <tr valign="top">
     <td>URL</td>
     <td>Description</td>
    </tr>
% foreach my $link (@links) {
    <input type="hidden" name="project_link_ids" value="<% $link->{id} %>">
%     next unless defined $link->{url};
    <tr valign="top">
     <td>
      <input type="text" name="url<% $link->{id} %>"
                value="<% $link->{url} | h %>" size="30" maxlength="200">
     </td>
     <td>
      <input type="text" name="description<% $link->{id} %>"
```

```
                value="<% $link->{description} | h %>" size="50" maxlength="200">
        </td>
      </tr>
% }
% foreach (1..2) {
      <tr valign="top">
        <td>
          <input type="text" name="new_url<% $_ %>"
                value="<% $ARGS{"new_url$_"} || '' | h %>"
                size="30" maxlength="200">
        </td>
        <td>
          <input type="text" name="new_description<% $_ %>"
                value="<% $ARGS{"new_description$_"} || '' | h %>"
                size="50" maxlength="200">
        </td>
      </tr>
% }
    </table>
  </td>
 </tr>
 <tr valign="top">
```

```
  <td colspan="2"><input type="submit" value="Submit"></td>
 </tr>
 <form>
  <%args>
 $submit_to
 $project
 @category_ids => ( )
 @errors => ( )
</%args>

<%init>
 my $statuses =
     $Schema->ProjectStatus_t->all_rows
         ( order_by => $Schema->ProjectStatus_t->status_c );
   my $support_levels =
     $Schema->ProjectSupportLevel_t->all_rows
         ( order_by =>
           $Schema->ProjectSupportLevel_t->project_support_level_id_c );
   my $categories =
     $Schema->Category_t->all_rows
         ( order_by => $Schema->Category_t->name_c );
   my $links = $project->Links;
```

```
    my @links;
while (my $link = $links->next) {
        my $id = $link->project_link_id;
        # the link was deleted but we've returned to this page because
        # of some error.
        if (exists $ARGS{"url$id"} && ! length $ARGS{"url$id"}) {
            push @links, { id => $id, url => undef };
        } elsif (exists $ARGS{"url$id"} && length $ARGS{"url$id"}) {
            push @links, { id => $id,
                           url => $ARGS{"url$id"},
                           description => $ARGS{"description$id"} };
        } else {
            push @links, { id => $id,
                           url => $link->url,
                           description => $link->description };
        }
}
    my %current_categories;
if (@category_ids) {
        %current_categories = map { $_ => 1 } @category_ids;
} else {
        %current_categories =
```

```
            map { $_->category_id => 1 } $project->Categories->all_rows;
    }
        my $member_count =
            $Schema->ProjectMember_t->row_count
                ( where =>
                    [ $Schema->ProjectMember_t->project_id_c,
                        '=', $project->project_id ] );
        my %form_vals;
    foreach my $field ( qw( name description difficulty
                                project_status_id project_support_level_id ) ) {
            $form_vals{$field} =
                exists $ARGS{$field} ? $ARGS{$field} : $project->$field( );
    }
        $form_vals{role_id} = $ARGS{role_id} || 0;
        # Only used if a project has no members (i.e. a new project)
    my $roles;
    $roles =
            $Schema->Role_t->all_rows( order_by => $Schema->Role_t->role_id_c )
                unless $member_count;
  </%init>
```

/logged_in/new_project_submit.html

我们在这处理创建新项目，以及相关的成员、分类、链接等。它同/users/new_user_submit.html.看起来很相似。

因为是一个新项目，我们给它一个成员就是创建的用户，该用户对该项目具有管理权限，也即可以编辑该项目。

/logged_in/editable_project_list.html

该组件用来列出当前用户具有管理权限的项目。提供编辑每个项目内容和成员以及删除项目的链接。

/logged_in/edit_project.html

这里用来编辑项目。

/logged_in/check_access_to_project.mas

这是一个在其他一些地方调用用来确保用户允许编辑给定项目的帮助组件。用户必须是网站管理员或者对该项目具有管理权限。

```
<%args>

 $project

</%args>

  <%init>

 unless ($User->is_admin || $User->is_project_admin($project)) {

      $m->comp( '/lib/redirect.mas', path => '/' );

 }

</%init>
```

/logged_in/edit_project_submit.html

同用来编辑用户的组件相似，这个稍复杂些。发现一项工程不再应该在一个种类方面的事实， 我们需要对提交该组件的那些在数据库里检查工程的种类的当前目录。 与此类似，我们需要检查被提交的目录看看是否有任何种类已经不属于该工程。

对于链接，当文本编辑框中 URL 被删除时就删除相关链接。对于其他情况我们进行更新就可以。如果有新的链接，就把它加到数据库中。

```
<%args>
```

```
 $project_id

 @project_link_ids => ( )

 @category_ids => ( )

</%args>

<%init>

 my $project =

      eval { $Schema->Project_t->row_by_pk( pk => $project_id ) }

           || $m->comp( '/lib/redirect.mas', path => '/' );

$m->comp( 'check_access_to_project.mas', project => $project );

eval {

      $project->update

           ( name => $ARGS{name},

             description => $ARGS{description},

             difficulty => $ARGS{difficulty},

             project_status_id => $ARGS{project_status_id},

             project_support_level_id => $ARGS{project_support_level_id},

           );

 };

$m->comp( '/lib/redirect.mas',

             path => '/logged_in/edit_project.html',

             query => { %ARGS, errors => $@->errors } )

      if $@ && UNIVERSAL::isa( $@, 'Apprentice::Exception::DataValidation' );
```

```
my %current_categories =
        map { $_->category_id => 1 } $project->Categories->all_rows;
foreach my $id (@category_ids) {
        $Schema->ProjectCategory_t->insert( values => { project_id => $project_id,
                                                        category_id => $id } )
            unless exists $current_categories{$id};
  }
{
        # This is the categories selected on the project editing page.
        my %selected_categories = map { $_ => 1 } @category_ids;
   # These are categories the project currently has which were
        # _not_ selected on the editing page.
        my @to_delete;
        foreach my $id (keys %current_categories) {
            push @to_delete, $id unless $selected_categories{$id};
        }
if (@to_delete) {
            foreach ( $Schema->ProjectCategory_t->rows_where
                        ( where =>
                          [
                            [ $Schema->ProjectCategory_t->project_id_c,
                              '=',    $project_id ],
```

```
                                [ $Schema->ProjectCategory_t->category_id_c,

                                    'IN', @to_delete    ]

                                ]

                               )->all_rows ) {

                        $_->delete;

                    }

                }

            }

        {

            # This is basically the same logic as was used for categories

            # except that if a link wasn't deleted, we may need to update

            # it.

            my @to_delete;

            foreach my $id (@project_link_ids) {

                if ( defined $ARGS{"url$id"} && length $ARGS{"url$id"} ) {

                    my $link =

                        eval { $Schema->ProjectLink_t->row_by_pk( pk => $id ) }

                            || next;

                    $link->update( url => $ARGS{"url$id"},

                                   description => $ARGS{"description$id"} );

                } else {

                    push @to_delete, $id
```

```
            }

        }

    if (@to_delete) {

            foreach ( $Schema->ProjectLink_t->rows_where

                                ( where =>

                                  [ $Schema->ProjectLink_t->project_link_id_c,

                                    'IN', @to_delete ] )->all_rows ) {

                $_->delete;

            }

        }

    }

# Finally, insert any new links from the previous page.

    foreach (1..2) {

        if (exists $ARGS{"new_url$_"} && length $ARGS{"new_url$_"}) {

            $Schema->ProjectLink_t->insert

                ( values =>

                  { project_id => $project->project_id,

                    url => $ARGS{"new_url$_"},

                    description =>

                    defined $ARGS{"new_description$_"} ?

                    $ARGS{"new_description$_"} : $ARGS{"new_url$_"},

                  }
```

```
                );
        }
    }
  $m->comp( '/lib/redirect.mas',
                path => '/logged_in/edit_project.html',
                query => { project_id => $project_id } );
</%init>
<%flags>
  inherit => '/syshandler'
</%flags>
```

/logged_in/edit_members.html

因为项目编辑页面已经很满了，所以决定让用户来编辑他自己的页面，避免界面太多了。

我们故意不允许用户去授予和取消对一个已存在的项目成员。其实只需将该成员删除并再次以不同权限添加进来就可以了。

我们也不允许用户将自己从项目中删除，因为这样很可能是用户误操作的。并且如果用户将自己删除了，可以让项目除了网站管理员外没人可以编辑该项目。

```
<table width="100%" cellspacing="0" cellpadding="5">
  <tr>
    <td class="heading" colspan="5">
      <h2 class="headline">Edit Project Members</h2>
    </td>
  </tr>
  <tr>
```

```
  <td colspan="2">
   <a href="<& /lib/url.mas,
                  path    => '/logged_in/edit_project.html',
                  query => { project_id => $project->project_id } &>">
    Edit project</a>
  </td>
 </tr>
 <tr>
  <td colspan="5"><h3>Current members for <% $project->name | h %></h3></td>
 </tr>
% while (my $member = $members->next) {
 <tr>
  <td><% $member->username | h %></td>
  <td><% $member->role %></td>
  <td>
%    if ($member->is_project_admin) {
    <b>Project admin</b>
%    } else {
     
%    }
  </td>
%    if ( $member->username eq $User->username ) {
```

```
    <td colspan="2"> </td>
%     } else {
    <form action="<& /lib/url.mas,
                            path => 'remove_project_member.html' &>" method="POST">
      <input type="hidden" name="project_id" value="<% $project_id %>">
      <input type="hidden" name="user_id" value="<% $member->user_id %>">
    <td colspan="2"><input type="submit" value="Remove"></td>
    </form>
%     }
  </tr>
% }
  <tr>
    <td colspan="5"><h3>Add a new member</h3></td>
  </tr>
% if ($error) {
  <tr>
    <td colspan="5"><span class="error"><% $error | h %></span></td>
  </tr>
% }
  <form action="<& /lib/url.mas,
                            path => 'add_project_member.html' &>" method="POST">
    <input type="hidden" name="project_id" value="<% $project_id %>">
```

```
  <tr>
    <td><input type="text" name="username" value="<% $username | h %>"></td>
    <td>
      <select name="role_id">
%       while (my $role = $roles->next) {
        <option value="<% $role->role_id %>"
          <% $role_id == $role->role_id ? 'selected="selected"': '' %>>
          <% $role->role | h %>
        </option>
%       }
    </td>
    <td>
      As admin?
      <input type="checkbox" name="is_project_admin"
                value="1" <% $is_project_admin ? 'checked="checked"': '' %>>
    </td>
    <td><input type="submit" value="Add"></td>
  </tr>
  </form>
</table>
<%shared>
my $project =
```

```
    eval { $Schema->Project_t->row_by_pk

                ( pk => $m->request_args->{project_id} ) }

        || $m->comp( '/lib/redirect.mas', path => '/' );

</%shared>

<%args>

 $project_id

 $username => ''

 $role_id => 0

 $is_project_admin => 0

 $error => ''

</%args>

 <%init>

 $m->comp( 'check_access_to_project.mas', project => $project );

 my $members =

     $Schema->join( select => $Schema->ProjectMember_t,

                    join     =>

                    [ $Schema->tables( 'ProjectMember', 'User' ) ],

                    where    =>

                    [ $Schema->ProjectMember_t->project_id_c, '=', $project_id ],

                    order_by => $Schema->User_t->username_c );

 my $roles = $Schema->Role_t->all_rows( order_by => $Schema->Role_t->role_id_c );

</%init>
```

```
<%method title>

 <& PARENT:title &> - Members of <% $project->name | h %>

</%method>
```

/logged_in/add_project_member.html

该组件验证提交的用户名存在，当存在时在项目成员表里增加一行记录。

/logged_in/remove_project_member.html

该组件检查访问权限，从数据库删除一条记录，并且重定向。

/logged_in/delete_project.html

该组件同删除项目成员的组件相似。主要区别在我们判断在删除项目之后将用户重定向到哪个页面。如果他还有其他项目，我们重定向到他的项目列表，否则，就把他重定向到顶级页面。

```
<%args>

 $project_id

 $redirect_to => undef

</%args>

<%init>

 my $project = $Schema->Project_t->row_by_pk( pk => $project_id );

 $m->comp( 'check_access_to_project.mas', project => $project );

 $project->delete;

unless ($redirect_to) {

      $redirect_to =

           $User->has_projects ? '/logged_in/editable_project_list.html' : '/';

 }
```

```
$m->comp( '/lib/redirect.mas', path => $redirect_to );

</%init>

<%flags>

 inherit => '/syshandler'

</%flags>
```

最后是/admin 目录下的一些组件：

/admin/autohandler

该组件同/logged_in/autohandler 几乎一样，除了有不同的访问权限检查和标题方法。

/admin/user_list.html

该组件用来显示一个分页的用户列表供管理员浏览。每个用户的链接允许管理员编辑该用户信息。

/admin/edit_user.html

该组件同/logged_in/edit_self.html 很相似，只不过它带$user_id 参数，允许对任何用户进行编辑。该组件也使用了/users/user_form.mas 组件。

/admin/edit_categories.html

该组件提供一个表单允许编辑、删除或者增加分类。

/admin/alter_category.html

管理员可以修改分类的名称。

/admin/add_category.html

该组件用来增加一些新的分类。

/admin/delete_category.html

该组件删除一个已存在的分类。

3.7.4 小结

这就是我们的站点。从开始到结束搭建这个站点一般在 30－40 人*小时左右，不太坏。许多的时间被花在调整 HTML 上了，因为那不是我们的强项。如果不使用 Mason，做一个传统的 CGI 应用程序，或许要花更长的时间取得代码重用，或者会很草率的结束。

3.7.5 展望

我们在本章节开始时提到，我们想要使这个网站小得足以在一个章节内讲完（虽然是一个相当长的章节），为避免介绍 Alzabo 相关内容，我们没有使用 Alzabo 的一些特性来简化代码。 当你设计你自己的网站时，你将没有这些限制条件。 你可以试着为网站增加一些东西。

- 多继承：

注意到最后 3 个组件， /admin/add_category.html, /admin/alter_category.html 和 /admin/delete_category.html，都具有一些属性，比如它们重定向到相同位置并都从/syshandler 继承。可以将这些共同行为封为一个组件，然后从它继承。

- 更有效的使用 Alzabo：

因为 Alzabo 可以提供数据库的元数据（比如字段名称、类型、长度等），我们可以从 widget 组件生成合适大小和合适字段长度的表单。事实上，Alzabo 在发行中包含相应的 Mason 组件，但未避免介绍 Alzabo，我们没有使用它。

- 双层组件根：

如果你打算在不只一个地方复制该站点，可以有两种做法。第一是在每一处都安装组件和模块。第二种方法是在一个组件根下为所有站点创建一个共享组件集，只在另一个组件根下需要重载时重载相关内容。这样就创建了一个更灵活的网站框架。

创建一个这样的框架的一种方法是将所有组件放到共享组件根下，只重载你的站点特定根下的一些部分。这种方法需要一些再设计，但这是移植一次性站点的比较好的作法。

第4章 使用 Maypole 快速开发 web 应用

Maypole 是 Perl 中为数不多的 web 应用程序开发框架。在这一章里，我们将描述使用 Maypole 如何建立一个简单的 CRUD(create, replace, update, delete)web 应用程序。

同时，我们将介绍 Maypole 的技术基础，Class::DBI 和 Template toolkit(模版工具集)以及他们和 Maypole 之间的关系。

啤酒数据库的例子显示了如何从系统自动提供的模版和行为到使用用户声明的行为去定制应用程序。最后，我们通过例子演示了如何用 Maypole 构建大型 web 应用 。BuySpy 实现了一个 portal 应用，这个例子来源于 ASP.NET。

4.1 Maypole——Perl 快速开发框架

Maypole 和 Jakarta 的 Struts 类似，是一个 Perl 框架用于基于 MVC 的 web 应用。Maypole 设计成使用最少的代码创建简单的基于数据库的 web 界面 ，同时保持足够的灵活性以支持企业 web 应用。

许多 web 应用遵循这样的操作流程：响应用户的请求，混杂着一些数据库操作，然后通过模版系统，把结果返回给用户。

例如，查找一个公司的产品目录会从远端的 web 客户端接受一个请求，搜索数据库，返回匹配查询条件的数据行，格式化数据行以后显示在页面上。

这个设置有时候叫做"MVC"，在这里，这意味着和用户交互 (控制元素) ，请求数据库的动作(模型元素)和展现数据 (视图元素)是优雅的分开的。

Maypole 提供了一个通用的方式处理这个分离的实现。它提供了顶层代码检验用户的请求，触发模型中相关的方法，比较数据，要求视图类格式化数据并把它返回给用户。

一些通用的 Perl 模块，例如 Apache，Class::DBI 和 Template，已经执行了模型，视图或者控制操作需要的重要部分，因此 Maypole 提供了包裹类把它们集成到一个框架里。

这样有什么好处呢？因为这意味着你可以用少于 20 行代码写一个全功能的数据库接口。

什么是 web 应用程序的 MVC 呢?Maypole 以前叫做 Apache::MVC，代表它基于 Model-View-Controller 设计模式。后来因为 Maypole 并没有绑定在 Apache 上所以有了现在这个名字。它采用了和 Java 的 Struts 框架同样的设计模式。这个设计模式主要用于图形化应用。MVC 对于 web 应用来说是，把你的数据库代码放在一个地方，应用程序代码放在另外一个地方，而展现代码放在第三个地方。这样，你就可以根据需要改变不同的元素，而不

会影响到其它的部分。

Maypole 有一些数据库驱动，一些前台驱动，一些模版显示驱动。在一般情况下，Maypole 精确的提供了你所需的，你可以集中精力写业务逻辑。这也是 Maypole 让你的开发如此快速的原因：因为在大多数时候，你根本不需要任何开发。

4.2 安装 Maypole

你需要使 Maypole 运行起来的第一件事情就是安装它。Maypole 需要一些来自 CPAN 的 Perl 模块帮助它完成工作。 因为这样实现了代码重用。

使用 CPAN.pm 安装只需要输入以下命令中的一个：

```
>perl -MCPANPLUS -e 'install Maypole'

>perl -MCPAN -e 'install Maypole'
```

使用编译源代码的方式安装，可以从 http://maypole.simon-cozens.org/download.html 下载源代码，它需要的模块有：

Class::DBI，

CGI::Untaint，

UNIVERSAL::moniker，

UNIVERSAL::require，

Apache::Request（在 CPAN 包 libapreq-1.3 中），

Template，

Template::Plugin::Class。

你将也需要一台数据库服务器和一台 web 服务器。对数据库来说，推荐 SQLite(如果你安装 DBD::SQLite 模块，你就得到了 SQLite 库)； 大型应用的用户应该使用 Postgresql 或者 Oracle——Maypole 全部支持它们。Maypole 最适合的运行环境是在 Apache::Request 模块的 Apache mod_perl 下，但是，它同时时可以灵活定制的。已经支持的运行环境有 Apache mod_perl 版本 1 和版本 2。，同时也有一个 CGI::Maypole 前端支持作为独立 CGI 脚本运行方式。

4.3 啤酒数据库的例子

我们使用 Maypole 来讲述一个啤酒的故事。第一个 Maypole 应用就是用来纪录一个地区可得到的很多不同的啤酒——他们的风格，他们的口味，他们的生产厂，价格等等。这是一个演示不同种类的数据库关系的很好的数据模型。

我们有一张啤酒厂表，这里有几种啤酒。 我们将它称为一对多的关系。 每一种啤酒有一种风味； 风味储存在单独的表内，因此啤酒有一个风味。 啤酒在几个酒店销售，一个酒店有几种啤酒，因此啤酒和酒店在一种多对多的关系里。我们使用一张叫 handpump 的连接表关联酒店与啤酒。总的说来，我们如下这个模式：

```
create table brewery (
    id int not null auto_increment primary key,
    name varchar(30),
    url varchar(50),
    notes text
);

create table beer (
    id int not null auto_increment primary key,
    brewery integer,
    style integer,
    name varchar(30),
    url varchar(120),
    score integer(2),
    price varchar(12),
    abv varchar(10),
```

```
    notes text
);

create table handpump (
    id int not null auto_increment primary key,
    beer integer,
    pub integer
);

create table pub (
    id int not null auto_increment primary key,
    name varchar(60),
    url varchar(120),
    notes text
);

create table style (
    id int not null auto_increment primary key,
    name varchar(60),
    notes text
);
```

如果你已经安装了 DBD::SQLite 模块，安装 Maypole 时就会创建一个象这样的数据库。如果没有，可以使用上面的模式创建一个。现在让我们看怎样用它建立起一个 web 界面。

接下来我们需要的是完成全部工作的一个模块。令人欣慰的是，这个模块不需要自己完成全部的工作。它将是一个 Maypole 或者象 Apache::MVC 这样的 Maypole 前端的子类。如下是我们的啤酒数据库应用的驱动类。在这里我们不过多的深入细节。它非常简洁，这是你实现一个简单的数据库前端所需要写的全部代码：

```
package BeerDB;

use base 'Apache::MVC';

BeerDB->setup("dbi:SQLite:t/beerdb.db");

BeerDB->config->{uri_base} = "http://localhost/beerdb/";

BeerDB->config->{rows_per_page} = 10;

BeerDB->config->{display_tables} = [qw[beer brewery pub style]];

BeerDB::Brewery->untaint_columns( printable => [qw/name notes url/] );

BeerDB::Style->untaint_columns( printable => [qw/name notes/] );

BeerDB::Beer->untaint_columns(

    printable => [qw/abv name price notes/],

    integer => [qw/style brewery score/],

    date => [ qw/date/],

);



use Class::DBI::Loader::Relationship;

BeerDB->config->{loader}->relationship($_) for (

    "a brewery produces beers",

    "a style defines beers",

    "a pub has beers on handpumps");

1;
```

这个模块定义了 BeerDB 应用。它继承自 Apache::MVC，将是 mod_perl 处理器。同时，这意味着我们需要把它告诉给 Apache 配置函数：

<Location /beerdb>

SetHandler perl-script

PerlHandler BeerDB

</Location>

现在我们需要一些模版。有几种类型的模版。我们把它们从 Maypole 源代码的 templates/目录的全部复制到我们 web 根目录下的的/beerdb 目录。

这就是所有我们要做的工作。我们现在应该能去 http://localhost/beerdb/并且看见一个可以浏览的菜单； http: //localhost/beerdb/beer/list 将给出一个啤酒的目录。这里还没有任何东西，你还要往这里面放东西。

4.3.1 Class::DBI

在啤酒数据库的模型类使用了 Class::DBI，接下来我们介绍它。因为它极其简明，而且还有一组扩展类使得它更为简单。例如，如果我这样开始：

使用 CDBI::Loader 创建类：

use Class::DBI::Loader;

Class::DBI::Loader->new(namespace => "BeerDB", dsn => "dbi:mysql:beerdb");

然后，不可思议地，BeerDB::Beer, BeerDB::Brewery 、BeerDB::Pub 、和 BeerDB::Handpump 这几个类就出现了，然后我可以指明：

my $hooky = BeerDB::Beer->retrieve(12); # Retrieve by ID

my $generation = BeerDB::Beer->search(name => "Generation");

$generation->score(9); # Good beer!

如果接下来我要指定表之间的关系，我可以像这样指明：

BeerDB::Beer->has_a(brewery => "BeerDB::Brewery");

```
my $rch             = BeerDB::Brewery->find_or_create( name => "RCH" );
my $pitchfork    = BeerDB::Beer->create({
    name       => "Pitchfork",
    brewery => $rch
});

print $pitchfork->brewery->name;
```

4.3.2 Template Toolkit

模板工具箱(Template Toolkit)是一个非常强大和通用的模版系统。它使用它自己的格式化语言支持循环，条件语句，哈西数组解引用和方法调用，宏处理。它有一个 plug-in 系统把它和外部 Perl 模件连结起来。 有几篇关于这个模板工具箱的介绍：你应该有一个安装成 Template::Tutorial::Datafile 的文档；在 http://www.perl.com/pub/a/2003/07/15/nocode. html 有一篇，当然还有"獾书" ——Perl 模板工具箱，是模板工具箱的开发者安迪等的著作。

我们将在这里通过解析一个 Maypole 应用使用的模板简短的介绍模板工具箱。这里是啤酒数据库应用的首页的模板，custom/frontpage。

```
[% INCLUDE header %]

<h2> The beer database </h2>

<TABLE BORDER="0" ALIGN="center" WIDTH="70%">
[% FOR table = config.display_tables %]
<TR>
<TD>
```

```
<A HREF="[%table%]/list">List by [%table %]</A>

</TD>

</TR>

[% END %]

</TABLE>
```

首先要指出的是，模板工具箱标签外([% 和 %])的东西是逐字输出的。即，在输出的某个地方你将保证看见：

```
<h2> The beer database </h2>



<TABLE BORDER="0" ALIGN="center" WIDTH="70%">
```

神奇发生在标签里面。 第一个标签是[% INCLUDE header %]指令。它找到一个称为 header 的文件，并且处理文件的内容好象他们正在这个模板内一样。我们的 header 文件正好不包含任何[% %]标签，但是如果它也包含，将会以与在 frontpage 里同样的方式处理。 下一个神奇之处在这里：

```
[% FOR table = config.display_tables %]
```

在这里我们看见许多新的东西。config 是我们首先应该看的地方。这是一个模板变量。这是使模板有意义的地方——模版意味着从外面的某处得到数据并且以有用方式把它展现给用户。config 实际上是 Maypole 应用配置参数的哈西，其中一个关键字是 display_tables，我们允许操作的数据库表。在这个例子里，我们可以这样写：

```
BeerDB->config->{display_tables} = [qw[beer brewery pub style]];
```

这里在一个数组内储存 4 个值：啤酒，啤酒厂，酒店和风格。数组放在配置哈西的 display_tables 键值。现在我们返回原来的代码。

FOR 循环将重复代码 4 次，设置我们的新变量到合适的数组元素。下面的代码：

```
[% FOR table = config.display_tables %]

    Hello [% table %]!

[% END %]
```

将产生如下的输出：

Hello beer!

Hello brewery!

Hello pub!

Hello style!

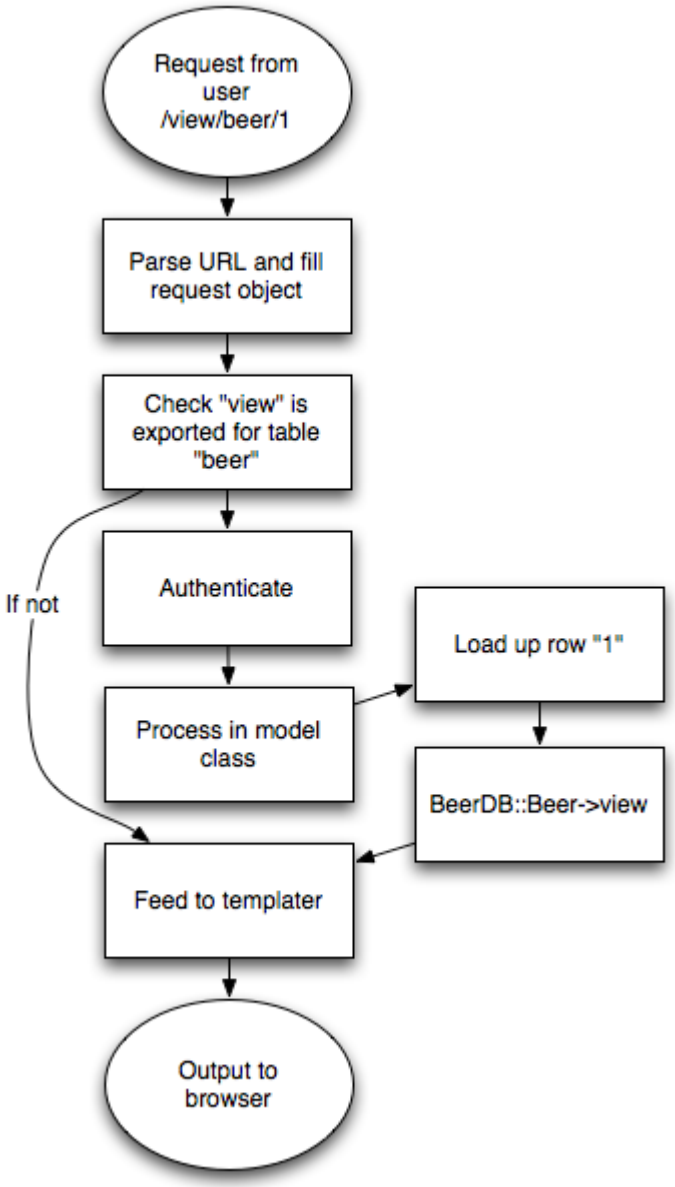
在我们这个例子里，将打印出一个连接每个数据库表的表元素。

4.4 Maypole 请求的阶段

现在你应该对 Maypole 能够做的事情有一个印象了。但这绝不是 Maypole 能做的全部。你已经在啤酒数据库例子里看见的是所有那些如果你根本不定制 Maypole，它能做的事情。

记住，我们甚至没有告诉 Maypole 我们的数据库中有什么表，或者每个表有哪些字段。我们没有告诉 Maypole 如何调用那些表或者怎样展示他们。 我们没有告诉 Maypole 做什么——我们想要列举，搜寻，编辑并且删除啤酒和啤酒厂。Maypole 自己把所有这些都做了。我们能定制它并且让 Maypole 用我们的数据库做各种各样有趣的事情，下面的大部分都是讲关于如何定制的事情。

为了实现定制，我们需要看 Maypole 实际上做了什么。就象已经提到的那样，Maypole 负责把 URL 变成一个对象，一个方法调用，和一些模版化的输出。这里有一张方便的图解释它怎样实现的：



Maypole 的处理围绕着 Maypole 请求对象的概念。这有点象 Apache 的请求对象，但是在高得多的层次上——实际上，在基于 mod_perl 的 Maypole 前端，Apache 请求对象内置在 Maypole 请求对象中。所有 Maypole 做的是逐渐充实这个对象，直到它包含输出成员的一些东西， 然后它被发送回用于输出的前端。 从开始，我们分析 Apache 的请求(或者 CGI 对象)并且深层剖析它。 例如，我们把 URL /beer/view/1 转换成：

```
{
    table => "beer",
    action => "view",
    args => [ 1 ]
```

```
}
```

然后 Maypole 将检查 beer 是一个真正的表，并且发现建模它的类。 它也检查我们是否允许通过网络调用 view 方法：

```
{
    table => "beer",
    action => "view",
    args => [ 1 ],
    model_class => "BeerDB::Beer"
}
```

然后有一种用户定义的验证方法，这个方法缺省让我们做任何事情。现在我们处理模型类。这个类装载对象，并且决定我们想要使用什么模版：

```
{
    table => "beer",
    action => "view",
    args => [ ],
    objects => [ BeerDB::Beer->retrieve(1) ],
    model_class => "BeerDB::Beer",
    template => "view"
}
```

然后它调用 BeerDB::Beer->view，把请求对象作为参数传递进来，并且传递用于建立模版的 view 类。

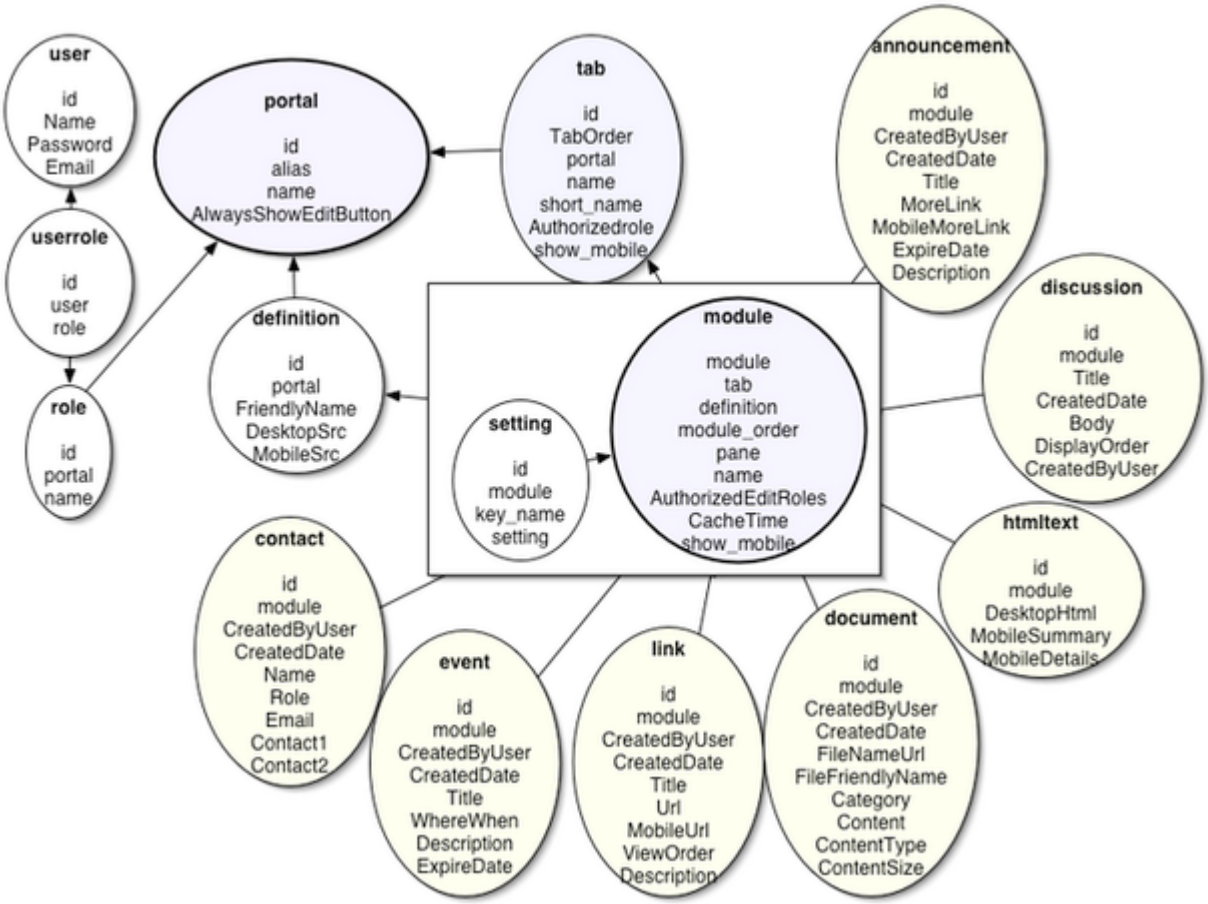
4.5 BuySpy 实现

比较 Maypole 与 ASP.NET 是一件很有趣的事情。IbuySpy 门户演示了如何使用 ASP.NET 构建一个 portal 站点。下面我们要实现一个 Maypole 版本的 IbuySpy。

我们从把数据库模式和初始化数据转变成 Mysql 数据库开始。把 MS SQL 转化成 Mysql 语句可以利用 SQL::Translator 模块，这个模块将在第五章解析 SQL 语句部分介绍。

ibsportal 数据库有一些表描述 portal 看起来应该是怎样的，也有许多表描述数据是否应该显示。Portal 根据一个模块的集合而确定； 每个模块在某处接受一些数据，并且指定一个模版用来格式化数据。 这十分不同于 Maypole 通常的操作，因此我们可以完全复制这种设计，或者使用让 Portal 展示方式由模版本身确定的更自然的实现方法。在后者，全部模版就在模版工具箱代码里指定而不是从数据库那里拿出来。因为你无需独立处理每个模块的模版，这将快得多。

我们需要做的第一个事情是确定数据库中表之间的关系，并得到一个图形化的显示，它看起来象这样：



很自然的产生如下的驱动代码：

```
package Portal;

use base 'Apache::MVC';

Portal->setup("dbi:mysql:ibsportal");
```

```
use Class::DBI::Loader::Relationship;

Portal->config->{loader}->relationship($_) for (

    "A module has a definition",    "A module has settings",

    "A tab has modules",              "A portal has tabs",

    "A role has a portal",            "A definition has a portal",

    "A module has announcements", "A module has contacts",

    "A module has discussions",     "A module has events",

    "A module has htmltexts",        "A module has links",

    "A module has documents",

    "A user has roles via userrole"

);

1;
```

如你所见，一个 portal 由许多不同的 tab 页面组成，而每个 tab 页面包含一些模块，而且这些模块被分在不同的面板里，因此一个模件知道它属于左边的面板，右边的面板或者中心。 一个模块也知道它出现在面板的哪里。我们将首先制作 portal 的视图，象下面一样：

```
use Portal;

my $portal = Portal::Portal->retrieve(2);

for my $tab ($portal->tabs) {

    print $tab,"\n";

    for my $pane (qw(LeftPane ContentPane RightPane)) {

        print "\t$pane:\n";

        for (sort { $a->module_order <=> $b->module_order }

            $tab->modules(pane => $pane)) {
```

```
                print "\t\t$_:\t", $_->definition,"\n";
            }
        }
        print "\n";
    }
```

这样与在每个 tab 页面中的模块和它们的类型内一起产生出我们的 portal 的 tab 页面。同时，这让我们检查我们是否已经正确地建立起数据库。如果是肯定的，它应该产生象这样的布局：

```
Home
        LeftPane:
                Quick link:     Quicklink
        ContentPane:
                Welcome to the IBuySpy Portal:  Html Document
                News and Features:      announcement
                Upcoming event: event
        RightPane:
                This Week's Special:    Html Document
                Top Movers:     XML/XSL

...
```

现在我们想要首页面建立起来。此时，我们将让它象我们的文本模型一样展示模块名和他们的定义，稍后我们将充实现有的模块。

但是在我们实现模块以前，我们将写一个前端 URL 处理器方法，给把我们允许符合 ASP 文件名字。 我们为什么想要使 Maypole 站点看起来象在运行一个. aspx 文件呢？因为我们可以做到!而且我想要显示我们不一定必须遵从 Maypole 传统的方式，让我们的 URL 看

起来象/table/action/id/arguments。

```
our %pages = (
    "DesktopDefault.aspx" => { action => "view", table => "tab" },
    "MobileDefault.aspx"    => { action => "view_mobile", table => "tab" },
);

sub parse_path {
    my $self = shift;
    $self->{path} ||= "DesktopDefault.aspx";
    return $self->SUPER::parse_path if not exists $pages{$self->{path}};
    my $page = $pages{$self->{path}} ;
    $self->{action} = $page->{action};
    $self->{table} = $page->{table};
    my %query = $self->{ar}->args;
    $self->{args} = [ $query{tabid} || $query{ItemID} || 1];
}

1;
```

这里我们覆盖了 parse_path 方法。它接受请求的路径槽值并且展开表，行动和参数槽值。如果用户询问我们不知道的一个页面，我们要求通常的 Maypole 路径处理方法试一试；在后面，这将变得很重要。我们把默认页，DesktopDefault.aspx，转换成相当的/tab/view/1，除非另一个 tabid 或者 ItemID 在查询参数已经给出了；这样让我们使用 ASP.NET 风格的 DesktopDefault.aspx?tabid=3 选择一个 tab 页面。

现在我们必须建立我们的 tab/view 模版；主要的部分是模仿 DesktopDefault.aspx 源代码，但是我们的面板看起来象这样：

```
<td id="LeftPane" Width="170">
    [% pane("LeftPane") %]
</td>
<td width="1">
</td>
<td id="ContentPane" Width="*">
    [% pane("ContentPane") %]
</td>
<td id="RightPane" Width="230">
    [% pane("RightPane") %]
</td>
<td width="10">
     
</td>
```

面板宏必须是 Perl 代码的模版工具箱的模拟：

```
[% MACRO pane(panename) BLOCK;
    FOR module = tab.modules("pane", panename);
        "<P>"; module; " - "; module.definition; "</P>";
    END;
END;
```

现在，iBuySpy portal 的工作方式是每个模块有一个定义，并且每个定义包含指向一个模版的路径：$module->definition->DesktopSrc 为一个 ascx 组件文件返回一个路径名。 我们所需要做的全部事情把那些文件从 ASP 转变成模版工具箱，并且让 Maypole 为每个模块处理每个组件。

4.5.1 组件和模版

既然页面由许多不同的模块组成，而这些模块都需要它们自己的对象集合，对每一个模块，应该可以使用独立的 Maypole 子请求。我在 Portal::Module 里创建了一个方法，它将把模版设置到合适的 ascx 文件：

```
sub view_desktop :Exported {

    my ($self, $r) = @_;

    $r->{template} = $r->objects->[0]->definition->DesktopSrc;

}
```

并且改变面板宏，为每个模块触发一个子请求：

```
[% MACRO pane(panename) BLOCK;

    FOR module = tab.modules("pane", panename);

        SET path = "/module/view_desktop/" _ module.id;

        request.component(path);

    END;

END; %]
```

这样能正确的工作，到/module/view_desktop/12 的调用将查找 Html 文档模块定义，发现 DesktopSrc 是 DesktopModules/HtmlModule.ascx，并且用那个模版处理模块 12。 我把 HtmlModule. ascx 转化成一个模版工具箱文件(我们将立即看模版的转换)，。它将很好地展示的在我们的 portal 上。唯一的缺点是它非常慢，因为我们匆忙地发出一系列 Maypole 请求，以便每个模版都能得到需要的对象。但是请求占用了 5 秒钟时间。

其实这些模版根本不需要不同的对象。传给/module/view_desktop 所关心的唯一的对象是一个模块对象，并且每个模版描述每一件事情都是通过附加的调用。但是我们已经有一个模块对象，在我们的 FOR 循环中——我们正在使用它达成组件调用。为什么不只直接在循环里面处理每个模版？

```
[% MACRO pane(panename) BLOCK;

    FOR module = tab.modules("pane", panename);
```

```
            SET src = module.definition.DesktopSrc;
            TRY;
                PROCESS $src;
            CATCH DEFAULT;
                "Bah, template $src broke on me!";
            END;
        END;
    END; %]
```

这样工作的更好些，并且把请求花费的时间从 5 秒减少到可接受的秒级以下的水平。我可以再次把 view_desktop 方法拿出来；保持的更少的代码行总是好的。现在视图这边剩下的工作的是将我们的 ASP 模版转变成一些有意义的事情。

4.5.2 ASP 到模版工具箱

这些模版语言全部都差不多。他们中的一些，只不过比其它的稍微罗嗦些。例如，在 header 里出现的这个状态栏模版由 104 行 ASP 代码组成。这些代码中的大多数用来创建我们能看到的导航条。 到现在为止，我们还没有登录用户的概念， 因此我们没能区分开任何人都能看见的那些 tab 和只有管理员能看见的那些 tab，但是我们稍后将解决这个问题。 仍然是 104 行代码。实际的 tab 目录经过条理清楚的整理后出现在这里：

```
    <tr>
        <td>
            <asp:datalist id="tabs" cssclass="OtherTabsBg"
 repeatdirection="horizontal" ItemStyle-Height="25"
 SelectedItemStyle-CssClass="TabBg" ItemStyle-BorderWidth="1"
 EnableViewState="false" runat="server">
                <ItemTemplate>
                     <a href='<%= Request.ApplicationPath %>/
```

```
DesktopDefault.aspx?tabindex=<%# Container.ItemIndex %>&tabid=

<%# ((TabStripDetails) Container.DataItem).TabId %>' class="OtherTabs">

<%# ((TabStripDetails) Container.DataItem).TabName %></a> 

                    </ItemTemplate>

                    <SelectedItemTemplate>

                         <span class="SelectedTab">

<%# ((TabStripDetails) Container.DataItem).TabName %></span> 

                    </SelectedItemTemplate>

                </asp:datalist>

            </td>

        </tr>
```

但是必须使用大约 22 行的 C#代码把它建立起来。这些代码创建和展开一个数组，然后把它和一个模版参数绑定起来。 看见那些<%#和<%= tags 了吗?，他们相当于我们这里的模版工具箱 [% %]标签。 Request.ApplicationPath 呢？那是我们的基础模版参数。在我们的版本中，我们查询 portal 有什么 tab 页面，并且直接展示当前选中的 tab 页：

```
    <table id="Banner_tabs" class="OtherTabsBg" cellspacing="0" border="0">

        <tr>

    [% FOR a_tab = portal.tabs %]

        [% IF a_tab.id == tab.id %]

            <td class="TabBg" height="25">

                 <span class="SelectedTab">[%tab.name%]</span> 

        [% ELSE %]

            <td height="25">

                 <a href='[%base%]DesktopDefault.aspx?tabid=[%a_tab.id%]'
```

```
                class="OtherTabs">[%a_tab.name%]</a> 

        [% END %]

            </td>

    [% END %]

        </tr>

    </table>
```

这就是这种应用应该有的样子。但是等一下，我们从哪里取出这些 portal 变量呢？我们需要告诉 Portal 类把缺省的 portal 放到模版参数里去：

```
    sub additional_data {

        shift->{template_args}{portal} = Portal::Portal->retrieve(2);

    }
```

翻译全部其它的 ASP.NET 组件是重复的劳动。我们仅仅把一个特别冗余的模版语言转变成了一个更复杂的(并且我不再看到 asp:BoundColumn BoundColumn，失去它并不是损失) 。

最简单的组件，HtmlModule.ascx，为了它相关的 htmltexts 请求一个模块， 然后为在表格里他们中的每个展示 DesktopHtml。 这是 40 行 ASP.NET 代码，包括 C# 调用 SQL 语句并且查询 htmltext。 但是我们能魔术般的实现所有这些检索，因此我们的 HtmlModule.ascx 看起来象这样：

```
    [% PROCESS module_title %]

    <portal:title EditText="Edit" EditUrl="~/DesktopModules/EditHtml.aspx" />

    <table id="t1" cellspacing="0" cellpadding="0">

        <tr valign="top">

            <td id="HtmlHolder">

            [% FOR html = module.htmltexts; html.DesktopHtml; END %]

            </td>
```

```
        </tr>

    </table>
```

到现在为止，我们还没有实现 portal:title 标签——显然我们应该把它变成一个编辑这个模块的 HTML 连接。下面最简单的一个实际上确实是一个挑战。ImageModule.ascx 从模块的设置表取得高度，宽度和图像来源属性，并且使用合适的值显示一个 IMG 标签。这只有点困难，因为我们必须将这个 module.settings 的数组安排进一个哈西表的“关键字=>设置”对。坦白地说，在这个模版内实现不了这个，因此我们将再次把它增加进这个模版的_args。这次 addition_data 看起来象下面这样：

```
    sub additional_data {

        my $r = shift;

        shift->{template_args}{portal} = Portal::Portal->retrieve(2);

        if ($r->{objects}->[0]->isa("Portal::Module")) {

            $r->{template_args}{module_settings} =

                { map { $_->key_name, $_->setting }

                  $r->{objects}->[0]->settings };

        }

    }
```

ImageModule.ascx 从 30 多行 ASP 代码变成了：

```
    [% PROCESS module_title; %]

    <img id="Image1" border="0" src="[% module_settings.src %]"

      width="[% module_settings.width %]"

      height="[% module_settings.height %]" />

    <br>
```

我们的 portal 正在成形；在一些更多的模版已经被翻译之后，我们现在已经有了一个首页和它的全部 tab 页的完整复制品。我们很快就把它开发出来了，并且迄今，只写了不到 50 行 Perl 代码。

第5章 使用 Perl 解析文本

这里介绍使用 Parse::RecDescent 来解析文本文件，同时结合一个有用的工具软件介绍一下数据库程序开发人员熟悉的 SQL 语句的解析。

5.1 解析文本文件

举一个例子，在某公司内部需要本市提供的气象数据用作参考。我们要做的工作是每小时把远程取得的文本形式存放的数据加载到 Oracle 数据库中去。

数据库表——未来 48 到 72 小时天气趋势预报(QXXX_SS)，的结构如下：

英文列名	中文列名
rq_id	日期列
tq	天气
fx	风向
fs	风速
zgqw	最高气温
zdqw	最低气温

其中日期列是主键。

A 市气象台提供的文本文件的样例如下：

未来 48 到 72 小时天气趋势预报

A 市专业气象台　　发布日期:2003.07.31

****

预报时段:2003 年 8 月 1 日 20 时---2003 年 8 月 3 日 20 时

8 月 1 日 20 时-8 月 2 日 20 时:

天 气:多云 转阴有小到中雨

风 向：南风

风 力: 2-3 级

最高气温：31/33 摄氏度

最低气温：24/25 摄氏度

8 月 2 日 20 时-8 月 3 日 20 时:

天 气:阴有中雨到大雨转多云

风 向：南风

风 力: 2-3 级

最高气温：30/31 氏度

最低气温: 24/25 摄氏度

我们可以使用 Parse::RecDescent 来实现这个功能。

```
use Parse::RecDescent;

use DBI;

use Tie::DBI;

use Getopt::Std;
```

```
my $grammar = <<'EOG';

 addition_expr: title dept publish_date_title ':' publish_date comments time_range data_row(1..4)

                        { return substr ($item[5],0,4);}

                      | <error: illegal expression $text>

   title: /A 地区未来三到四天天气趋势预报/ | /未来 48 到 72 小时天气趋势预报/

   dept: /A 市专业气象台/

   publish_date_title: /发布日期/

   publish_date: /\d\d\d\d.\d+.\d+/

   comments:/\*+/

   time_range:/预报时段:\d\d\d\d 年\d+月\d+日---\d\d\d\d 年\d+月\d+日/ | /预报时段:\d\d\d\d 年\d+月\d+日\d+时---\d\d\d\d 年\d+月\d+日\d+时/

   data_row: report_date ':' tq_column fx_column fl_column zgqw_column zdqw_column

   report_date:report_date1 | report_date2

   report_date1:/\d+月\d+日\(/ '周' /../ ')'

                        {    $item[1]=~s/ 月 /-/;    $item[1]=~s/ 日 //;    $item[1]=~s/\(//; $main::col_rq[$main::row_count] = $item[1];}

   report_date2:/\d 月\d+日\d+时/ '-' /\d 月\d+日\d+时/

                        {   $item[3]=~s/ 月 /-/;   $item[3]=~s/ 日 /  /;   $item[3]=~s/ 时 /:00:00/; $main::col_rq[$main::row_count] = $item[3];}

   tq_column:'天  气:' tq_data

   tq_data: /.*\n/

                        {    $item[1]=~s/\s+//g;   $main::col_tq[$main::row_count]=$item[1]; }

   fx_column:'风  向:' fx_data
```

```
    fx_data: /.*\n/

                    { $item[1]=~s/\s//g;    $main::col_fx[$main::row_count] = $item[1];}

    fl_column:'风  力:' fl_data

    fl_data: /.*\n/

                    { $item[1]=~s/\s+$//;    $main::col_fl[$main::row_count] = $item[1]; }

    zgqw_column:'最高气温:'    zgqw_data | '最高气温  '    zgqw_data

    zgqw_data: /.\d+/    token_a(0..1)    '摄氏度'

                    { $item[1]=~s/\s+$//;    $main::col_zgqw[$main::row_count] = $item[1]; }

    token_a: end_a /.\d*/

    end_a:'-' | '/'

    zdqw_column:'最低气温:' zdqw_data

    zdqw_data:    /.\d+/ token_a(0..1)      '摄氏度'

                    {   $item[1]=~s/\s+$//;     $main::col_zdqw[$main::row_count]   =   $item[1];
$main::row_count++; }

EOG


## main


#command arguments ,please see usage

use vars qw($opt_h $opt_v $opt_f $opt_S $opt_U $opt_P);


getopts('hvf:S:U:P:') or usage() and exit;
```

```
usage() and exit if $opt_h;



open(INPUT,"< $opt_f")|| die "不能打开文件。\n";



my @lines = <INPUT>;

my $text = join('', @lines);

$text =~ s/：/:/g;

$text =~ s/）/)/g;

my $parserRef = new Parse::RecDescent($grammar);



#USE TIE

my $dbh = DBI->connect("dbi:Oracle:$opt_S",$opt_U,$opt_P,{AutoCommit => 0}) || die "Unable to connect to xxx";



#~ my $rq_id;

my %h;

my $debug_level;



if($opt_v)

{

    $debug_level=2;

}

else
```

```
{
    $debug_level=0;
}

tie %h,Tie::DBI,{db             => $dbh,
                 table      => 'QXXX_SS',
                 key          => 'RQ_ID',
                 CLOBBER    => 2,
                 DEBUG      => $debug_level};

#CLOUMN DATA
our @col_tq;
our @col_fx;
our @col_fl;
our @col_zgqw;
our @col_zdqw;
our @col_rq;

our $row_count=0;

my $ret = $parserRef->addition_expr($text);
```

```
die $@ if $@;



for my $i ( 0 .. $#col_tq)

{

	$col_rq[$i]=$ret.'-'.$col_rq[$i];

	print "rq$i =".$col_rq[$i]."\n" if ($opt_v);

	print "tq$i =".$col_tq[$i]."\n" if ($opt_v);



	$h{$col_rq[$i]} = { tq => $col_tq[$i] ,fx => $col_fx[$i], fs=>$col_fl[$i], zgqw=>$col_zgqw[$i], zdqw=>$col_zdqw[$i]};

}



tied(%h)->commit;



$dbh->disconnect;



sub usage

{

	print STDERR << "EOF";



	用法：$0 [-hvd] [-f file] -s path

	 -h		：显示帮助信息

	 -f file		：天气预报数据文件名
```

-S serverName ：数据库服务器名

-U UserName ：数据库用户名

-P Password ：数据库密码

-v ：是否详细输出（缺省不详细输出）

例子： $0 -v -f ss02.txt -S dwdb -U dwdb -P dwdb

EOF

}

数据库表——A 市各区县天气实况(QXXX_SK)，的结构如下：

英文列名	中文列名
xzq	行政区
tq	天气
fx	风向
fs	风速
qy	气压
js	降水
xdsd	相对湿度
njd	能见度
tqxx	天气现象
wd	温度

要解析的文本文件样例如下：

2003 年 06 月 25 日 08 时 A 市各区县天气实况

站 名　　天 气　风 向　风 速 气　压　降　水 相对湿度 能见度 天气现象 温度

(M/S)　(hPa)　(mm)　(%)　(Km)　(℃)

C 区　　多云　SW　2　1002　0　53　15　////　25.3

B 区　　晴　W　2　1001.6　0　41　20　////　26.4

A 区　　晴　NW　3　1002.5　0　82　15　////　23.8

D 区　　晴　E　2　1001.4　0　57　20　////　24.7

```
use Parse::RecDescent;

use DBI;

use Tie::DBI;

use Getopt::Std;


my $grammar = <<'EOG';

 addition_expr: publish_date <resync> <resync> <resync> <resync>    data_row(0..100)

                        { return substr ($item[5],0,4);}

                      | <error: illegal expression>

   publish_date: /\d\d\d\d 年\d\d 月\d\d 日\d\d/ '时天津市各区县天气实况'

                        {    $item[1]=~s/ 年 /-/;    $item[1]=~s/ 月 /-/;    $item[1]=~s/ 日 /  /;
$main::col_rq[$main::row_count] = "$item[1]:00:00";}

   header1:/站  名.+/

   header2:/\(M\/S\).+\(℃\)/
```

data_row:column_xzq column_tq column_fx column_fl column_qy column_js column_xdsd column_njd column_tqxx column_wd

column_xzq:/......./

{ $item[1]=~s/\s+$//; $main::col_xzq[$main::row_count] = $item[1];}

column_tq:/...../

{ $item[1]=~s/\s+$//; $main::col_tq[$main::row_count] = $item[1];}

column_fx:/...../

{ $item[1]=~s/\s+$//; $main::col_fx[$main::row_count] = $item[1]; }

column_fl:/...../

{ $item[1]=~s/\s+$//; $main::col_fl[$main::row_count] = $item[1]; }

column_qy:/...../

{ $item[1]=~s/\s+$//; $main::col_qy[$main::row_count] = $item[1]; }

column_js:/...../

{ $item[1]=~s/\///g; $item[1]=~s/\s+$//; $main::col_js[$main::row_count] = $item[1]; }

column_xdsd:/...../

{ $item[1]=~s/\s+$//; $main::col_xdsd[$main::row_count] = $item[1]; }

column_njd:/...../

{ $item[1]=~s/\s+$//; $main::col_njd[$main::row_count] = $item[1]; }

column_tqxx:/...../

{ $item[1]=~s/\///g; $item[1]=~s/\s+$//; $main::col_tqxx[$main::row_count] = $item[1]; }

column_wd:/...../

{ $item[1]=~s/\s+$//; $main::col_wd[$main::row_count] = $item[1];

```
$main::row_count++;$main::col_rq[$main::row_count] =$main::col_rq[$main::row_count - 1]}

EOG



## main



our $row_count=0;



#CLOUMN DATA

use vars qw(@col_xzq @col_tq @col_fx @col_fl @col_qy @col_js @col_xdsd @col_njd @col_tqxx @col_wd);



#command arguments ,please see usage

use vars qw($opt_h $opt_v $opt_f $opt_S $opt_U $opt_P);



getopts('hvf:S:U:P:') or usage() and exit;



usage() and exit if $opt_h;

usage() and exit unless $opt_f;



open(INPUT,"< $opt_f")|| die "不能打开文件$opt_f。\n";



my @lines = <INPUT>;

my $text = join('', @lines);
```

```
my $parserRef = new Parse::RecDescent($grammar);



#USE TIE

my $dbh = DBI->connect("dbi:Oracle:$opt_S",$opt_U,$opt_P,{AutoCommit => 0}) || die "Unable to connect to $opt_S";



my $rq_id;

my %h;

my $debug_level;



if($opt_v)

{

	$debug_level=2;

}

else

{

	$debug_level=0;

}



tie %h,Tie::DBI,{db		=> $dbh,

			table	=> 'BI_F_QXXX_SK',

			key		=> 'RQ_ID',
```

```
                        CLOBBER    => 2,

                        DEBUG      => $debug_level};



my $ret = $parserRef->addition_expr($text);



die $@ if $@;



for my $i ( 0 .. $#col_tq)

{

    #~ print "$col_tq[$i]\n" if($opt_v) ;

    print "rq$i =".$col_rq[$i]."\n" if($opt_v) ;

    $h{$col_rq[$i]} = {xzq => $col_xzq[$i],  tq => $col_tq[$i] ,fx => $col_fx[$i], fs=>$col_fl[$i], qy=>$col_qy[$i], js=>$col_js[$i], xdsd=>$col_xdsd[$i], njd=>$col_njd[$i], tqxx=>$col_tqxx[$i], wd=>$col_wd[$i]};

}



##update

my $sql = 'UPDATE BI_F_QXXX_SK SET N_TQID =(SELECT N_TQID FROM BI_D_GG_TQ WHERE BI_D_GG_TQ.VA_TQMS=BI_F_QXXX_SK.TQ) WHERE N_TQID IS NULL';

$dbh->do($sql);



tied(%h)->commit;
```

```
$dbh->disconnect;


sub usage
{
    print STDERR << "EOF";

    用法：$0 [-hvd] [-f file] -s path
     -h                 ：显示帮助信息
     -f file            ：天气预报数据文件名
     -S serverName   ：数据库服务器名
     -U UserName      ：数据库用户名
     -P Password      ：数据库密码
     -v                 ：是否详细输出（缺省不详细输出）

    例子：  $0 -v -f ss02.txt -S dwdb -U dwdb -P dwdb
EOF
}
```

5.2 解析 SQL 语句

我们在工作中可能会碰到这样的问题。

比如有一个在 MySQL 上开发的应用，但是需要部署到 Oracle 数据库。当数据库的模式较小时，可以采用手工做移植。如果有很多表或者数据需要处理时，可以考虑采用工具。以前，可以采用 Power Designer 帮助实现，但是现在 Perl 中有一个叫做 SQL::Translator 的模块可以帮你做这件事情。

5.2.1 使用 SQL::Translator

SQL::Translator 的安装方法是这样的：

```
$ perl -MCPAN -e 'install SQL::Translator'
```

采用手工安装时，需要一些模块的最新版本。如 Parse::RecDescent，Template 和 Text::RecordParser 等。

可以通过批处理文件 sqlt 调用 SQL::Translator 的功能，例如把 MySQL 模式转换到 Oracle 使用命令：

```
$ sqlt -f MySQL -t Oracle mysql.sql > oracle.sql
```

当 mysql.sql 的内容是：

```
CREATE TABLE aaaa (
    id int auto_increment PRIMARY KEY,
    foo varchar(255) not null default '',
    updated timestamp
);
```

时，将产生如下的输出：

```
-- We assume that default NLS_DATE_FORMAT has been changed
-- but we set it here anyway to be self-consistent.
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';



--
-- Table: aaaa
--
```

```
CREATE TABLE aaaa (
    id number(11) NOT NULL,
    foo varchar2(255) DEFAULT '' NOT NULL,
    updated date,
    CONSTRAINT pk_aaaa PRIMARY KEY (id)
);

CREATE SEQUENCE sq_aaaa_id;
CREATE OR REPLACE TRIGGER ai_aaaa_id
BEFORE INSERT ON aaaa
FOR EACH ROW WHEN (
  new.id IS NULL OR new.id = 0
)
BEGIN
  SELECT sq_aaaa_id.nextval
  INTO :new.id
  FROM dual;
END;
/

CREATE OR REPLACE TRIGGER ts_aaaa_updated
BEFORE INSERT OR UPDATE ON aaaa
```

```
FOR EACH ROW WHEN (new.updated IS NULL)

BEGIN

  SELECT sysdate INTO :new.updated FROM dual;

END;

/
```

由上例可以看出，这个模块自动修改了日期格式，并且利用 Oracle 的序列和触发器实现了自增列。

5.2.2 SQL::Translator 的实现方法

SQL::Translator 采用了 Parse::RecDescent 解析 SQL 语句。这个任务可以分成两部分：首先执行解析，然后执行翻译。Tim Bunce 写过一个叫做"mysql2ora"的脚本，他使用了一套很好的的 MySQL 语法并且把做标识符和数据类型翻译成 Oracle 认识的形式。

SQL::Translator 在这个脚本的基础上，创建了两个模块，一个只负责解析 MySQL，另外一个只产生 Oracle 输出，这两个模块相互独立。

SQL::Translator 有两种解析方式。一种是直接解析文本形式的 DDL，还有一种通过数据库的系统表提取 DDL。后者借助了 DBI(Perl 的数据库接口)。DBI 解析更快，因为它直接读取系统表，使用查询返回的结果。

除了 SQL 外，SQL::Translator 还能把 Excel 转换成 SQL 表。它规范化列名，扫描数剧列以决定数据类型。除了 Excel，它也可以转换任意分割符的文本文件(例如，逗号分隔，tab 分割的等等)。

在软件装上以后，可以通过"sqlt"脚本附加参数"--list"列举出所有的解析者和生产者，如下所示：

```
$ ./sqlt --list

    Parsers:

    DBI

    DBI-MySQL

    DBI-PostgreSQL
```

DBI-SQLite

DBI-Sybase

Excel

MySQL

Oracle

PostgreSQL

SQLite

Storable

Sybase

XML

XML-SQLFairy

YAML

xSV

Producers:

ClassDBI

Diagram

Dumper

GraphViz

HTML

MySQL

Oracle

POD

PostgreSQL

SQLite

Storable

Sybase

TTSchema

XML

XML-SQLFairy

YAML

这些生产者和解析器之间通过 Schema 对象(SQL::Translator::Schema)连接起来。Schema 对象描述数据库对象和他们之间的关系。

下面分析一下 MySQL 的解析器。它要处理的与创建表相关的语法如下：

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]

[table_options] [select_statement]

or

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name LIKE old_table_name;

create_definition:

col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]

[PRIMARY KEY] [reference_definition]

or PRIMARY KEY (index_col_name,...)

or KEY [index_name] (index_col_name,...)

or INDEX [index_name] (index_col_name,...)

or UNIQUE [INDEX] [index_name] (index_col_name,...)

or FULLTEXT [INDEX] [index_name] (index_col_name,...)

or [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)

[reference_definition]

or CHECK (expr)

type:

TINYINT[(length)] [UNSIGNED] [ZEROFILL]

or SMALLINT[(length)] [UNSIGNED] [ZEROFILL]

or MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]

or INT[(length)] [UNSIGNED] [ZEROFILL]

or INTEGER[(length)] [UNSIGNED] [ZEROFILL]

or BIGINT[(length)] [UNSIGNED] [ZEROFILL]

or REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]

or DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]

or FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]

or DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]

or NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]

or CHAR(length) [BINARY]

or VARCHAR(length) [BINARY]

or DATE

or TIME

or TIMESTAMP

or DATETIME

or TINYBLOB

or BLOB

or MEDIUMBLOB

or LONGBLOB

or TINYTEXT

or TEXT

or MEDIUMTEXT

or LONGTEXT

or ENUM(value1,value2,value3,...)

or SET(value1,value2,value3,...)

index_col_name:

col_name [(length)]

reference_definition:

REFERENCES tbl_name [(index_col_name,...)]

[MATCH FULL | MATCH PARTIAL]

[ON DELETE reference_option]

[ON UPDATE reference_option]

reference_option:

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:

TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }

or AUTO_INCREMENT = #

or AVG_ROW_LENGTH = #

or CHECKSUM = {0 | 1}

or COMMENT = "string"

or MAX_ROWS = #

or MIN_ROWS = #

or PACK_KEYS = {0 | 1 | DEFAULT}

or PASSWORD = "string"

or DELAY_KEY_WRITE = {0 | 1}

or ROW_FORMAT= { default | dynamic | fixed | compressed }

or RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#

or UNION = (table_name,[table_name...])

or INSERT_METHOD= {NO | FIRST | LAST }

or DATA DIRECTORY="absolute path to directory"

or INDEX DIRECTORY="absolute path to directory"

相关的解析文法如下：

```
$GRAMMAR = q!



{

    my ( %tables, $table_order, @table_comments );

}



#

# The "eofile" rule makes the parser fail if any "statement" rule

# fails.    Otherwise, the first successful match by a "statement"

# won't cause the failure needed to know that the parse, as a whole,

# failed. -ky

#

startrule : statement(s) eofile { \%tables }



eofile : /^\Z/



statement : comment

    | use

    | set

    | drop

    | create
```

```
        | <error>


use : /use/i WORD ';'

        { @table_comments = () }


set : /set/i /[^;]+/ ';'

        { @table_comments = () }


drop : /drop/i TABLE /[^;]+/ ';'


drop : /drop/i WORD(s) ';'

        { @table_comments = () }


create : CREATE /database/i WORD ';'

        { @table_comments = () }


create : CREATE TEMPORARY(?) TABLE opt_if_not_exists(?) table_name '(' create_definition(s /,/) ')' table_option(s?) ';'

        {

            my $table_name                              = $item{'table_name'};

            $tables{ $table_name }{'order'}         = ++$table_order;

            $tables{ $table_name }{'table_name'} = $table_name;
```

```
if ( @table_comments ) {
    $tables{ $table_name }{'comments'} = [ @table_comments ];
    @table_comments = ();
}

my $i = 1;
for my $definition ( @{ $item[7] } ) {
    if ( $definition->{'supertype'} eq 'field' ) {
        my $field_name = $definition->{'name'};
        $tables{ $table_name }{'fields'}{ $field_name } =
            { %$definition, order => $i };
        $i++;

        if ( $definition->{'is_primary_key'} ) {
            push @{ $tables{ $table_name }{'constraints'} },
                {
                    type      => 'primary_key',
                    fields => [ $field_name ],
                }
            ;
        }
    }
```

```
            elsif ( $definition->{'supertype'} eq 'constraint' ) {
                push @{ $tables{ $table_name }{'constraints'} }, $definition;
            }
            elsif ( $definition->{'supertype'} eq 'index' ) {
                push @{ $tables{ $table_name }{'indices'} }, $definition;
            }
        }

        if ( my @options = @{ $item{'table_option(s?)'} } ) {
            $tables{ $table_name }{'table_options'} = \@options;
        }

        1;
    }

opt_if_not_exists : /if not exists/i

create : CREATE UNIQUE(?) /(index|key)/i index_name /on/i table_name '(' field_name(s /,/) ')' ';'
    {
        @table_comments = ();
        push @{ $tables{ $item{'table_name'} }{'indices'} },
            {
```

```
                    name      => $item[4],
                    type      => $item[2] ? 'unique' : 'normal',
                    fields => $item[8],
                }
            ;
    }

create_definition : constraint
    | index
    | field
    | comment
    | <error>

comment : /^\s*(?:#|-{2}).*\n/
    {
        my $comment =    $item[1];
        $comment       =~ s/^\s*(#|-{2})\s*//;
        $comment       =~ s/\s*$//;
        $return        = $comment;
        push @table_comments, $comment;
    }
```

```
field_comment : /^\s*(?:#|-{2}).*\n/

    {

        my $comment =    $item[1];

        $comment        =~ s/^\s*(#|-{2})\s*//;

        $comment        =~ s/\s*$//;

        $return         = $comment;

    }


blank : /\s*/


field : field_comment(s?) field_name data_type field_qualifier(s?) reference_definition(?) field_comment(s?)

    {

        my %qualifiers    = map { %$_ } @{ $item{'field_qualifier(s?)'} || [] };

        if ( my @type_quals = @{ $item{'data_type'}{'qualifiers'} || [] } ) {

            $qualifiers{ $_ } = 1 for @type_quals;

        }


        my $null = defined $qualifiers{'not_null'}

                    ? $qualifiers{'not_null'} : 1;

        delete $qualifiers{'not_null'};


        my @comments = ( @{ $item[1] }, @{ $item[6] } );
```

```
        $return = {
            supertype      => 'field',
            name             => $item{'field_name'},
            data_type      => $item{'data_type'}{'type'},
            size             => $item{'data_type'}{'size'},
            list             => $item{'data_type'}{'list'},
            null             => $null,
            constraints => $item{'reference_definition(?)'},
            comments       => [ @comments ],
            %qualifiers,
        }
    }
    | <error>

field_qualifier : not_null
    {
        $return = {
            null => $item{'not_null'},
        }
    }
```

```
field_qualifier : default_val
    {
        $return = {
            default => $item{'default_val'},
        }
    }

field_qualifier : auto_inc
    {
        $return = {
            is_auto_inc => $item{'auto_inc'},
        }
    }

field_qualifier : primary_key
    {
        $return = {
            is_primary_key => $item{'primary_key'},
        }
    }

field_qualifier : unsigned
```

```
    {
        $return = {
            is_unsigned => $item{'unsigned'},
        }
    }
```

```
field_qualifier : /character set/i WORD
    {
        $return = {
            character_set => $item[2],
        }
    }
```

```
reference_definition : /references/i table_name parens_field_list(?) match_type(?) on_delete_do(?)
on_update_do(?)
    {
        $return = {
            type                => 'foreign_key',
            reference_table     => $item[2],
            reference_fields => $item[3][0],
            match_type          => $item[4][0],
            on_delete_do        => $item[5][0],
            on_update_do        => $item[6][0],
```

```
            }
        }

match_type : /match full/i { 'full' }
        |
        /match partial/i { 'partial' }

on_delete_do : /on delete/i reference_option
        { $item[2] }

on_update_do : /on update/i reference_option
        { $item[2] }

reference_option: /restrict/i |
        /cascade/i      |
        /set null/i     |
        /no action/i |
        /set default/i
        { $item[1] }

index : normal_index
        | fulltext_index
```

```
    | <error>

table_name      : NAME

field_name      : NAME

index_name      : NAME

data_type       : WORD parens_value_list(s?) type_qualifier(s?)
    {
        my $type = $item[1];
        my $size; # field size, applicable only to non-set fields
        my $list; # set list, applicable only to sets (duh)

        if ( uc($type) =~ /^(SET|ENUM)$/ ) {
            $size = undef;
            $list = $item[2][0];
        }
        else {
            $size = $item[2][0];
            $list = [];
        }
```

```
unless ( @{ $size || [] } ) {
    if ( lc $type eq 'tinyint' ) {
        $size = 4;
    }
    elsif ( lc $type eq 'smallint' ) {
        $size = 6;
    }
    elsif ( lc $type eq 'mediumint' ) {
        $size = 9;
    }
    elsif ( $type =~ /^int(eger)?$/ ) {
        $type = 'int';
        $size = 11;
    }
    elsif ( lc $type eq 'bigint' ) {
        $size = 20;
    }
    elsif (
        lc $type =~ /(float|double|decimal|numeric|real|fixed|dec)/
    ) {
        $size = [8,2];
```

```
        }
    }

    if ( $type =~ /^tiny(text|blob)$/i ) {
        $size = 255;
    }
    elsif ( $type =~ /^(blob|text)$/i ) {
        $size = 65_535;
    }
    elsif ( $type =~ /^medium(blob|text)$/i ) {
        $size = 16_777_215;
    }
    elsif ( $type =~ /^long(blob|text)$/i ) {
        $size = 4_294_967_295;
    }

    $return             = {
        type            => $type,
        size            => $size,
        list            => $list,
        qualifiers => $item[3],
    }
```

```
    }

parens_field_list : '(' field_name(s /,/) ')'
    { $item[2] }

parens_value_list : '(' VALUE(s /,/) ')'
    { $item[2] }

type_qualifier : /(BINARY|UNSIGNED|ZEROFILL)/i
    { lc $item[1] }

field_type      : WORD

create_index : /create/i /index/i

not_null        : /not/i /null/i { $return = 0 }

unsigned        : /unsigned/i { $return = 0 }

default_val : /default/i /'(?:.*?\\')*.*?'|(?:')?[\w\d:.-]*(?:')?/
    {
        $item[2] =~ s/^\s*'|'\s*$//g;
```

```
        $return    =    $item[2];
    }

auto_inc : /auto_increment/i { 1 }

primary_key : /primary/i /key/i { 1 }

constraint : primary_key_def
    | unique_key_def
    | foreign_key_def
    | <error>

foreign_key_def : foreign_key_def_begin parens_field_list reference_definition
    {
        $return                    =    {
            supertype              => 'constraint',
            type                    => 'foreign_key',
            name                    => $item[1],
            fields                  => $item[2],
            %{ $item{'reference_definition'} },
        }
    }
```

```
foreign_key_def_begin : /constraint/i /foreign key/i
    { $return = '' }
    |
    /constraint/i WORD /foreign key/i
    { $return = $item[2] }
    |
    /foreign key/i
    { $return = '' }

primary_key_def : primary_key index_name(?) '(' name_with_opt_paren(s /,/) ')'
    {
        $return             = {
            supertype => 'constraint',
            name           => $item{'index_name(?)'}[0],
            type           => 'primary_key',
            fields         => $item[4],
        };
    }

unique_key_def : UNIQUE KEY(?) index_name(?) '(' name_with_opt_paren(s /,/) ')'
    {
```

```
        $return             = {
            supertype => 'constraint',
            name            => $item{'index_name(?)'}[0],
            type            => 'unique',
            fields          => $item[5],
        }
    }

normal_index : KEY index_name(?) '(' name_with_opt_paren(s /,/) ')'
    {
        $return             = {
            supertype => 'index',
            type            => 'normal',
            name            => $item{'index_name(?)'}[0],
            fields          => $item[4],
        }
    }

fulltext_index : /fulltext/i KEY(?) index_name(?) '(' name_with_opt_paren(s /,/) ')'
    {
        $return             = {
            supertype => 'index',
```

```
                type            => 'fulltext',
                name            => $item{'index_name(?)'}[0],
                fields          => $item[5],
            }
        }

name_with_opt_paren : NAME parens_value_list(s?)
        { $item[2][0] ? "$item[1]($item[2][0][0])" : $item[1] }

UNIQUE : /unique/i { 1 }

KEY : /key/i | /index/i

table_option : WORD /\s*=\s*/ WORD
        {
            $return = { $item[1] => $item[3] };
        }

CREATE : /create/i

TEMPORARY : /temporary/i
```

```
TABLE : /table/i


WORD : /\w+/


DIGITS : /\d+/


COMMA : ','


NAME       : "`" /\w+/ "`"
      { $item[2] }
      | /\w+/
      { $item[1] }


VALUE      : /[-+]?\.?\d+(?:[eE]\d+)?/
      { $item[1] }
      | /'.*?'/
      {
            # remove leading/trailing quotes
            my $val = $item[1];
            $val       =~ s/^["']|["']$//g;
            $return = $val;
      }
```

```
    | /NULL/

    { 'NULL' }



!;
```

这个模块另外一个有用的功能是：一个叫做"sqlt-diff"的脚本，能够检查两个模式的差异，并写出要把第一个模式变成第二个模式需要执行的 SQL 命令。这个工具的一个用处是能帮助跟踪在现有的数据库里和当前最新的版本不有什么不一样。

第6章 Perl 与生物信息学

Bioperl 是一个超过 500 个 Perl 模块的集合。其中有些模块是独立的，有些模块与其他模块有联系。

首先介绍 Bioperl 中数据持久保存技术——BioSQL 与 Bioperl-DB。然后介绍使用 Bioperl 处理 microarray 数据。

6.1 快速上手

我们准备了一个简单的例子让新来者了解 Bio::Perl 模块的功能。例如，下面这个脚本检索一个 swissprot 序列把它输出成 fasta 格式。

```
use Bio::Perl;

# 这个脚本要求你所运行的计算机有一个 internet 连接。

# 可以从中取得序列的数据库包括：

# 'swiss'，'genbank'，'genpept'，'embl'和'refseq'

$seq_object = get_sequence('swiss',"ROA1_HUMAN");

write_sequence(">roa1.fasta",'fasta',$seq_object);
```

第二个参数，'fasta'，是序列格式。你可以选择 SeqIO 支持的所有序列格式。

另外一个例子是像 NCBI 那样以 blast 方法比对一个序列。如果你想做大量的 BLAST 查找，请下载 blast 包并在本地安装。

```
use Bio::Perl;

$seq = get_sequence('swiss',"ROA1_HUMAN");

# 使用缺省数据库——在这里是 nr

$blast_result = blast_sequence($seq);

write_blast(">roa1.blast",$blast_result);
```

Bio::Perl 还有一些其它易用的功能，包括：

功能	说明
get_sequence	gets a sequence from standard, internet accessible databases
read_sequence	reads a sequence from a file
read_all_sequences	reads all sequences from a file
new_sequence	makes a bioperl sequence just from a string
write_sequence	writes a single or an array of sequence to a file
translate	provides a translation of a sequence
translate_as_string	provides a translation of a sequence, returning back just the sequence as a string
blast_sequence	BLASTs a sequence against standard databases at NCBI
write_blast	writes a blast report out to a file

使用 Bio::Perl.pm 模块可以操作序列数据而不用显式的创建 Seq 或 SeqIO 对象。但是，在这种模式下操作数据是受限的。

6.2 安装 Bioperl 简介

使用如下的命令安装 Bioperl：

```
perl -MCPAN -e shell;
```

操作系统会给出一个 CPAN shell 提示：

```
cpan>
```

可以输入这个命令发现 Bioperl 安装：

```
cpan> i /bioperl/
```

Bundle::BioPerl 有一些特别有用的模块。我们从安装 Bundle 开始，开始安装 Bioperl：

```
cpan> install Bundle::BioPerl
```

当然也可以从 http://bio.perl.org/Core/Latest/index.shtml 下载源代码后，编译安装。解压

bioperl-1.4.zip 后，到解压缩文件目录下。输入如下命令：

```
>perl Makefile.PL

>make

>make install
```

6.3 序列

首先介绍使用 Bio::SeqIO 类进行格式转换。下面这个例子实现了从 SwissProt -> Fasta 格式的转换。

```
#! /local/bin/perl -w

use strict;

use Bio::SeqIO;


# Construct a SwissProt formatted sequences input stream

#  这里使用到的数据文件是 seqs.sp。

my $in = Bio::SeqIO->newFh ( -file => '<seqs.sp',

-format => 'swiss' );


# Construct a fasta formatted sequences output stream

my $out = Bio::SeqIO->newFh ( -file => '>seqs.fasta',

-format => 'fasta' );

#? for all sequences:

# read the sequence from input stream
```

```
# write it to output stream

print $out $_ while <$in>;
```

接下来介绍序列类。

如下的例子从远程服务器加载一个序列。

```
use Bio::DB::SwissProt;

my $database = new Bio::DB::SwissProt;

# Create a sequence object for the MALK_ECOLI SwissProt entry

my $seq = $database->get_Seq_by_id('MALK_ECOLI');

# Print its Accession number and description

print "Seq: ", $seq->accession_number(), " -- ", $seq->desc(), "\n\n";
```

构建序列类方法总结如下：

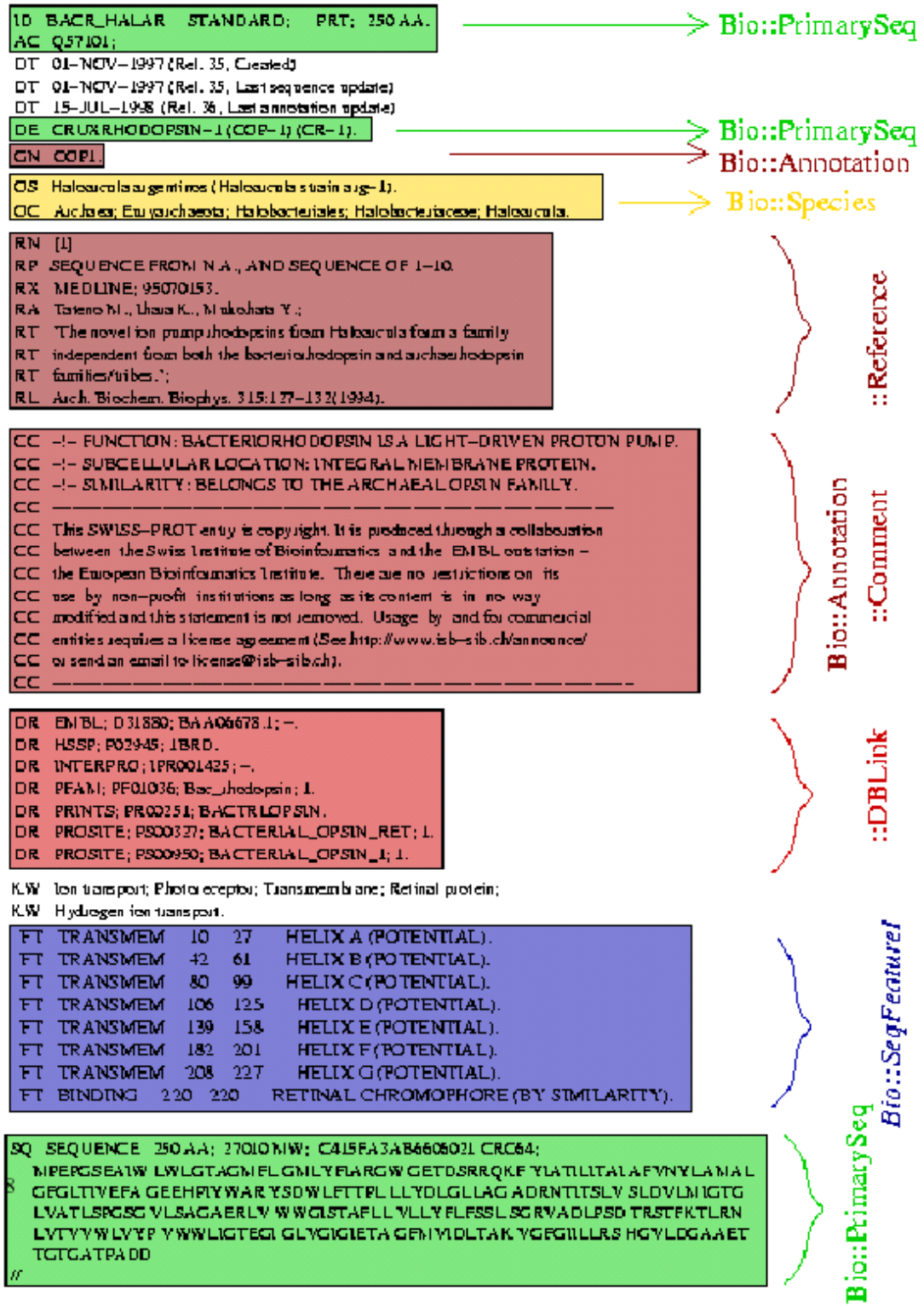
方法	示例
直接	use Bio::Seq; my $seq1 = Bio::Seq->new (-seq => 'ATGAGTAGTAGTAAAGGTA', -id => 'my seq', -desc => 'this is a new Seq');

方法	示例
来源于一个文件	use Bio::SeqIO; my $seqin = Bio::SeqIO->new (-file => 'seq.fasta', -format => 'fasta'); my $seq3 = $seqin->next_seq(); my $seqin2 = Bio::SeqIO->newFh (-file => 'seq.fasta', -format => 'fasta'); my $seq4 = <$seqin2>; my $seqin3 = Bio::SeqIO->newFh (-file => 'golden sp:TAUD_ECOLI |', -format => 'swiss'); my $seq5 = <$seqin3>;

从远程数据库取得一个条目	use Bio::DB::SwissProt;
	my $banque = new Bio::DB::SwissProt;
	my $seq6 = $banque->get_Seq_by_id('KPY1_ECOLI');

从一个 bioperl 索引的本地数据库取得条目	use Bio::Index::Swissprot;
	my $inx = Bio::Index::Swissprot->new(-filename => 'small_swiss.inx');
	my $seq7 = $inx->fetch('MALK_ECOLI');

下图显示了一个 SwissProt 条目和对应的 Bio::Seq 组件的关系。


ID BACR_HALAR STANDARD; PRT; 250 AA.
AC Q57101;
DT 01-NOV-1997 (Rel. 35, Created)
DT 01-NOV-1997 (Rel. 35, Last sequence update)
DT 15-JUL-1998 (Rel. 36, Last annotation update)
DE CRUXRHODOPSIN-1 (COP-1) (CR-1).
GN COP1.
OS Haloarcula argentinea (Halobacterium sp. arg-1).
OC Archaea; Euryarchaeota; Halobacteriales; Halobacteriaceae; Haloarcula.
RN [1]
RP SEQUENCE FROM N.A., AND SEQUENCE OF 1-10.
RX MEDLINE; 95070153.
RA Tateno M., Ihara K., Makahara Y.;
RT "The novel ion pump rhodopsins from Haloarcula form a family
RT independent from both the bacteriorhodopsin and archaerhodopsin
RT families/tribes.";
RL Arch. Biochem. Biophys. 315:127-132(1994).
CC -!- FUNCTION: BACTERIORHODOPSIN IS A LIGHT-DRIVEN PROTON PUMP.
CC -!- SUBCELLULAR LOCATION: INTEGRAL MEMBRANE PROTEIN.
CC -!- SIMILARITY: BELONGS TO THE ARCHAEAL OPSIN FAMILY.
CC ---
CC This SWISS-PROT entry is copyright. It is produced through a collaboration
CC between the Swiss Institute of Bioinformatics and the EMBL outstation -
CC the European Bioinformatics Institute. There are no restrictions on its
CC use by non-profit institutions as long as its content is in no way
CC modified and this statement is not removed. Usage by and for commercial
CC entities requires a license agreement (See http://www.isb-sib.ch/announce/
CC or send an email to license@isb-sib.ch).
CC ---
DR EMBL; D31880; BAA06678.1; -.
DR HSSP; P02945; 1BRD.
DR INTERPRO; IPR001425; -.
DR PFAM; PF01036; Bac_rhodopsin; 1.
DR PRINTS; PR00251; BACTRLOPSN.
DR PROSITE; PS00327; BACTERIAL_OPSIN_RET; 1.
DR PROSITE; PS00950; BACTERIAL_OPSIN_1; 1.
KW Ion transport; Photoreceptor; Transmembrane; Retinal protein;
KW Hydrogen ion transport.
FT TRANSMEM 10 27 HELIX A (POTENTIAL).
FT TRANSMEM 42 61 HELIX B (POTENTIAL).
FT TRANSMEM 80 99 HELIX C (POTENTIAL).
FT TRANSMEM 106 125 HELIX D (POTENTIAL).
FT TRANSMEM 139 158 HELIX E (POTENTIAL).
FT TRANSMEM 182 201 HELIX F (POTENTIAL).
FT TRANSMEM 208 227 HELIX G (POTENTIAL).
FT BINDING 220 220 RETINAL CHROMOPHORE (BY SIMILARITY).
SQ SEQUENCE 250 AA; 27010 MW; C415FA3AB6605021 CRC64;
MPEPGSEATW LWLGTAGMFL GMLYFLARGW GETDSRRQKF YLATLLTALA FVNYLAMALS
GFGLTIVEFA GEEHPIYWAR YSDWLFTTPL LLYDLGLLAG ADRNTITSLV SLDVLMIGTG
LVATLSPGSG VLSAGAERLV WWGLSTAFLL VLLYFLFSSL SGRVADLPSD TRSTEKTLRN
LVTYYWLVYP YWWLIGTEGI GLYGIGIETA GFMYVLDLTA KYGFGILLLR SRGYLDGAAE T
GTGATPADD
//

Bio::PrimarySeq
Bio::PrimarySeq
Bio::Annotation
Bio::Species
::Reference
Bio::Annotation
::Comment
::DBLink
Bio::SeqFeature
Bio::PrimarySeq

6.4 对位

6.4.1 AlignIO

Bio::AlignIO 类设计成和 SeqIO 类一样。因此，可以象下面这样做格式转换：

```
#! /local/bin/perl -w

use strict;

use Bio::AlignIO;

my $inform = shift @ARGV || ’clustalw’;

my $outform = shift @ARGV || ’fasta’;

my $in = Bio::AlignIO->newFh ( -fh => \*STDIN, -format => $inform );

my $out = Bio::AlignIO->newFh ( -fh => \*STDOUT, -format => $outform );

print $out $_ while <$in>;
```

6.4.2 简单对位

首先，让我们看一下 SimpleAlign 对象的一些基本方法：

```
#! /local/bin/perl -w

use strict;

use Bio::AlignIO;

my $in = new Bio::AlignIO ( -file =>, $ARGV[0], -format => ’clustalw’ );

my $aln = $in->next_aln();

print "same length of all sequences: ", ($aln->is_flush()) ? "yes" : "no", "\n";

print "alignment length: ", $aln->length(), "\n";

printf "identity: %.2f %%\n", $aln->percentage_identity();
```

```
printf "identity of conserved columns: %.2f %%\n", $aln->overall_percentage_identity();
```

SimpleAlign 类包含选择序列或者列的方法，但是它不能过滤对位。为了通过属性过滤列，你必须自己提取列，过滤它们并且重建新的序列。如下的例子过滤间隔列：

```
#! /local/bin/perl -w

use strict;

use Bio::AlignIO;

my $in = new Bio::AlignIO ( -file => $ARGV[0], -format => 'clustalw' );

my $out = newFh Bio::AlignIO ( -fh => \*STDOUT, -format => 'clustalw' );

my $aln = $in->next_aln();

# if you need to work on the columns, create a list containing all columns as

# strings

my @aln_cols = ();

foreach my $seq ( $aln->each_alphabetically() ) {

my $colnr = 0;

foreach my $chr ( split("", $seq->seq()) ) {

$aln_cols[$colnr] .= $chr;

$colnr++;

}

}

# then do the work:

# we want to eliminate all the columns containing gaps

# 1/ we create a list containing all the columns without any gap
```

```
my $gapchar = $aln->gap_char();

my @no_gap_cols = ();

foreach my $col ( @aln_cols ) {

next if $col =~ /\Q$gapchar\E/;

push @no_gap_cols, $col;

}

# 2/ we modify the sequences in the alignment

# 2a/ we reconstruct the sequence strings

my @seq_strs = ();

foreach my $col ( @no_gap_cols ) {

my $colnr = 0;

foreach my $chr ( split"", $col ) {

$seq_strs[$colnr] .= $chr;

$colnr++; }}

# 2b/ we replace the old sequences strings with the new ones

foreach my $seq ( $aln->each_alphabetically() ) {

$seq->seq(shift @seq_strs);}

print $out $aln;
```

6.5 分析序列

6.5.1 使用 Blast

Bioperl 的 Blast 模块提供了一个 BLAST 程序的 API。它允许解析，运行，HTML 格式化和操作 Blast 数据通过简单的调用 Perl 对象的方法。但是它本身没有提供 Blast 算法的实

现，而是依赖于外部的应用程序实现这个计算密集的操作。

BLAST 程序可以从 ftp://ftp.ncbi.nlm.nih.gov/blast/下载得到。在 windows 平台的安装方法是：自解压二进制软件包 blast-2.2.9-ia32-win32.exe，创建一个 ncbi.ini 文件，包含指向 BLAST 数据的路径，例如：

[NCBI]

Data="C:\Perl\bin\data\"

然后建立起本地的 BLAST 数据库。详细说明参见软件自带的 blast.txt 文档。

下面这个例子使用 Bio::Tools::Run::StandAloneBlast 独立的运行本地的 Blast。

```
use Bio::SeqIO;

use Bio::Tools::Run::StandAloneBlast;

my $Seq_in = Bio::SeqIO->new (-file => $ARGV[0], -format => ’fasta’);

my $query = $Seq_in->next_seq();

my $factory = Bio::Tools::Run::StandAloneBlast->new(’program’ => ’blastp’,

’ database’ => ’swissprot’,

_READMETHOD => "Blast"

);

my $blast_report = $factory->blastall($query);

my $result = $blast_report->next_result;

while( my $hit = $result->next_hit()) {

print "\thit name: ", $hit->name(), " significance: ", $hit->significance(), "\n";}
```

6.5.2 Genscan

Genscan 是一个用于基因预测的工具。下面这个例子显示了如何使用 Bio::Tools::Genscan 解析器。

```
use Bio::Tools::Genscan;


my $genscan_file = $ARGV[0];
$genscan = Bio::Tools::Genscan->new(-file => $genscan_file);


while(my $gene = $genscan->next_prediction()) {
   my $prot = $gene->predicted_protein;
   print "protein (", ref($prot), ") :\n", $prot->seq, "\n\n";}
```

带子序列的 Genscan 解析：

```
# Genscan: example with sub-sequences


use Bio::Tools::Genscan;


my $genscan_file = $ARGV[0];


$genscan = Bio::Tools::Genscan->new(-file => $genscan_file);
while(my $gene = $genscan->next_prediction()) {


      my $prot = $gene->predicted_protein;
      print "protein (", ref($prot), ") :\n", $prot->seq, "\n\n";


      # display genscan predicted cds (if -cds genscan option)
```

```
        my $predicted_cds = $gene->predicted_cds;

        print "predicted CDS: \n", $predicted_cds->seq, "\n\n";



        foreach my $exon ($gene->exons()) {

        my $loc = $exon->location;

       print "exon - primary_tag: ",$exon->primary_tag, " coding? ",$exon->is_coding(), " start-end:
", $loc->start, "-", $loc->end, "\n";

        }



        foreach my $intron ($gene->introns()) {

        my $loc = $intron->location;

        print "intron primary_tag: ",$intron->primary_tag, " start-end: ", $loc->start, "-", $loc->end,
"\n";

        }



        foreach my $utr ($gene->utrs()) {

        my $loc = $utr->location;

        print "utr primary_tag: ",$utr->primary_tag, " start-end: ", $loc->start, "-", $loc->end, "\n";

        }

        print "---------------------------------\n";

}
```

6.6 BioSQL 与 Bioperl-DB

bioperl-db 包提供了一个 perl 接口简化了生物关系数据库操作。

6.6.1 BioSQL 与 Bioperl-DB 的适用范围

通过这个模块，可以建立随机访问的本地基因库：

- 本地缓存或者公开数据库的复制；
- 建立索引后数据的随机访问，这样就能够容易的检索数据；
- 保留注释(特性, dbxrefs,…)，甚至可能保留格式。

可以建立关系型格式的基因库：

- 规格化的模式，可预测的组装；
- 允许随机的查询；
- 允许增加表来支持我的数据或问题等。

可以集成基因库，Swiss-Prot 数据库，位点连接：

- 统一关系型模式；
- 给不同来源标注的基因提供通用的视图。

可以建立集成了我的实验室序列和我的注释的数据库：

- 存储 FASTA 格式的序列；
- 增加，更新，修改或删除各种类型的注释。

可以通过它为自己的生物信息工具集提供持久化存储的功能：

- 关系型的模型容纳对象模型；
- 持久化 API 和透明的插入，更新和删除。

BioSQL 的特点是：

- 针对生物信息的互操作的关系型数据存储，它有 Bioperl, Biojava,Biopython, Bioruby 的语言绑定。

- 非常灵活规范的，基于本体论的模式。Focal 实体是 Bioentry，Seqfeature，Term 和 Dbxref。

- 有针对不同数据库(MySQL, PostgreSQL, Oracle)的模式实例化脚本。

BioSQL版本 1.0 即将发布。在过去 3 个月，模式已经稳定了。有相对完善的文档(安装手册，how-to和ER图)。邮件列表是(biosql-l@open-bio.org)，CVS (biosqlschema),连接在 http://obda.open-bio.org

BioSQL 的发展过程是这样的。

- Ewan Birney 从 11 月 2001 年开始 BioSQL 和 Bioperl-db，最开始的功能是提供 Bio::Seq 的序列化/反序列化。对象从本地序列存储的倒入倒出(作为 SRS 的一个替换)。

- 在 2002 年和 2003 年又重新设计了模式。修改的分类模型以遵循 NCBI 的完全的本体论模型， 类似于 GO 模型。特性可以有 dbxrefs。命名规范进行了统一。

6.6.2 基本功能

语言绑定（OR 映射）的原理是这样的：

- 对象——关系（OR）映射连接了两个世界，对象模型 (Bioperl) 和 关系模型(Biosql)。对象和关系模型基本上是正交的(虽然还是有关系的)。例如，实例与 n:n 连接，联接的导航与 join。

- OR 映射的一般的目标是：对象和实体间的双向映射；透明的持久化接口反映了所有的 INSERT, UPDATE, DELETE, SELECT。

- 同时也存在一般的方法，它们中的大多数都是商用的。例如：TopLink，CMP (比如 Jboss), JDO, Tangram。

Bioperl-db 是一个对象——关系映射器。举例如下：

```
# 取得数据库的持久化适配器的工厂类

my $db = Bio::DB::BioDB->new(-database => ’biosql’,

-dbcontext => $dbc);

# 打开从文本文件解析的对象流
```

```
my $stream = Bio::SeqIO->new(-fh => \*STDIN,

-format => 'genbank');

while(my $seq = $stream->next_seq()) {

  # 转换成持久化对象

  $pobj = $db->create_persistent($seq);

  # 插入数据存储

  $pobj->create();

}
```

那么从哪里能得到 Bioperl-db 呢？Bioperl-db 是一个 Bioperl 的子项目，在 http://www.bioperl.org/有相关的连接和新闻。它的最新的代码 CVS 存储库是在 bioperl 下的 bioperl-db。(/home/repository/bioperl/bioperl-db)

当前的代码库还没有发布出来，可用的版本是 0.1，但是版本 0.2 即将发布。

Bioperl-db 的关键特性有：

- 在对象 API 上的透明的持久化 API。持久化对象知道它们的主键 ，能够更新，插入和删除它们自己。完全的 API 在 Bio::DB::PersistentObjectI 中。持久对象支持持久化 API 和它们本地的方法。

- 在持久适配器 API 上的几个检索方法是： find_by_primary_key(), find_by_unique_key(),find_by_query(), find_by_association() 。完全的 API 在 Bio::DB::PersistenceAdaptorI 中。

- 可扩展的框架把对象适配器逻辑和模式逻辑分离开。中心化的工厂类在运行时加载和实例化依赖数据存储的适配器工厂类。适配器工厂类在运行时加载和实例化持久适配器，这样就不需要硬编码适配器名字。在对象空间构建查询，而且查询在运行时通过模式驱动被翻译成 SQL 语句。设计 Bioperl-db 时考虑了增加绑定到除了 BioSQL 的其他的模式 (例如，Chado，Ensembl，MyBioSQL 等)。

6.6.3 Bioperl-db 使用举例

从大的方面来说，使用 Bioperl-db 可以分为两步：首先取得适配器工厂类，然后根据需要可以进行各种操作。

步骤一：连接和取得适配器工厂类

```
use Bio::DB::BioDB;
# 创建数据库相关的适配器工厂类
# (继承了 Bio::DB::DBAdaptorI)
$db = Bio::DB::BioDB->new(-database =>”biosql”,
# user, pwd, driver, host …
-dbcontext => $dbc);
```

步骤二要依赖于使用情况。首先介绍加载序列的情况：

```
use Bio::SeqIO;
# 打开从文本文件解析的对象流
my $stream = Bio::SeqIO->new(-fh => \*STDIN,
-format => ’genbank’);
while(my $seq = $stream->next_seq()) {
# 转换到持久化对象
$pseq = $db->create_persistent($seq);
# $pseq 现在实现了 Bio::DB::PersistentObjectI
# 插入到数据存储
$pseq->create();
}
```

步骤二的另外一种使用情况是通过可选键检索序列：

```
use Bio::Seq; use Bio::Seq::SeqFactory;

# 建立 Seq 对象作为查询模版

$seq = Bio::Seq->new(-accession_number => “NM_000149”,

-namespace => “RefSeq”);

# 传递一个工厂类而保留模版类不动

$seqfact = Bio::Seq::SeqFactory->new(-type=>“Bio::Seq”);

# 取得用于查询的对象适配器类(类名也可以)

# 适配器实现了 Bio::DB::PersistenceAdaptorI

$adp = $db->get_object_adaptor($seq);

# 执行查询

$dbseq = $adp->find_by_unique_key(

$seq, -obj_factory => $seqfact);

warn $seq->accession_number(),

”  not found in namespace RefSeq\n“ unless $dbseq;
```

步骤二的第三种使用情况是通过查询检索序列：

```
use Bio::DB::Query::BioQuery;

# 建立起查询对象作为一个查询模版

$query = Bio::DB::Query::BioQuery->new(

    -datacollections => [“Bio::Seq s”,

      “Bio::Species=>Bio::Seq sp”],

    -where => [“s.description like ‘%kinase%’”,
```

```
        “sp.binomial = ?”]);
# 取得用于查询的对象适配器
$adp = $db->get_object_adaptor(“Bio::SeqI”);
# 执行查询
$qres = $adp->find_by_query($query, -name => “bosc03”,
        -values => [“Homo sapiens”]);
# 通过循环遍历结果集
while(my $pseq = $qres->next_object()) {
        print $pseq->accession_number,”\n”;
}
```

步骤二的最后一种使用情况是：检索序列，增加注释，更新数据库。

```
use Bio::Seq; use Bio::SeqFeature::Generic;
# 由于某种原因检索序列对象…
$adp = $db->get_object_adaptor(“Bio::SeqI”);
$dbseq = $adp->find_by_unique_key(
Bio::Seq->new(-accession_number => “NM_000149”,
        -namespace => “RefSeq”));
# 创建一个特性作为新的注释
$feat = Bio::SeqFeature::Generic->new(
        -primary_tag => “TFBS”,
        -source_tag => “My Lab”,
```

```
    -start=>23,-end=>27,-strand=>-1);

# 给序列增加新的注释

$dbseq->add_SeqFeature($feat);

# 更新数据库

$dbseq->store();
```

除了基本的使用外，Bioperl-db 还有很好的可扩展性，这样我们就可以通过增加自己的适配器处理自己的对象。

扩展的第一种方法是，定制序列类：

```
package MyLab::Y2HSeq;

@ISA = qw(Bio::Seq);

sub get_interactors{

    my $self = shift;

    return @{$self->{'_interactors'}};

}

sub add_interactor{

    my $self = shift;

    push(@{$self->{'_interactors'}}, @_);

}

sub remove_interactors{

    my $self = shift;

    my @arr = $self->get_interactors();

    $self->{'_interactors'} = [];

    return @arr;
```

```
}
```

扩展的另外一种方法是定制适配器类：

```
package Bio::DB::BioSQL::Y2HSeqAdaptor;

@ISA = qw(Bio::DB::BioSQL::SeqAdaptor);

sub store_children{

my ($self,$obj) = @_;

# call inherited method

$self->SUPER::store_children(@_);

# obtain persistent term object for the rel.ship type

my $term = Bio::Ontology::Term->new(

-name => “interacts-with”,

-ontology => “Relationship Types”);

my $termadp = $self->db->get_object_adaptor($term);

my $reltype = $termadp->find_by_unique_key($term) or

$self->db->create_persistent($term)->create();

# store the interacting sequences

foreach my $seq ($obj->get_interactors()) {

# each interactor needs to be persistent object

$seq = $self->db->create_persistent($seq)

unless $seq->isa("Bio::DB::PersistentObjectI");

# each interactor also needs to have a primary key
```

```
$seq = $seq->adaptor->find_by_unique_key() or

$seq->create();

# associate the interactor with this object

$seq->adaptor->add_association(

-objs => [$obj, $seq, $reltype],

-contexts => [“object”,”subject”,undef]);

}

return 1; # done

}
```

6.6.4 可以即时使用的脚本

这个模块中还有几个可以即时使用的脚本，第一个是在 bioperl-db/scripts/biosql 路径下的 load_seqdatabase.pl。它用于加载和更新条目和它们的注释。所有 Bio::SeqIO 支持的格式它都支持，如 genbank， embl，swiss，locuslink，fasta，gcg，ace 等。所有的 Bio::ClusterIO 支持的格式它也支持，如 Unigene。

它许多命令行选项。如用于灵活的处理更新有 –lookup， --noupdate， --remove，--mergeobjs。用于过滤和后处理序列的有 --seqfilter, --pipeline。

第二个可以即时使用的脚本是 bioperl-db/scripts/biosql 路径下的 load_ontology.pl。它的作用是加载和更新本体和条目。它支持所有 Bio::OntologyIO 支持的格式。包括 dagflat (incl. Soflat，goflat)，InterPro 和简单层次。通过了 GO 和 SOFA 测试。

它命令行选项包括两方面。用于处理更新和废弃的条目的选项有：--lookup, --noupdate, --remove， --noobsolete, --updobsolete, --delobsolete, --mergeobjs。用于计算传递闭包的有 –computetc。

第三个介绍的脚本是 biosql-schema/scripts 目录下的 load_ncbi_taxonomy.pl。它用于使用 NCBI 分类学数据库加载和更新分类表。如果需要的话，就从 NCBI 自动的下载数据库。有一些选项可用来配置和调优加载和更新。它能够自动的更新在分类表中的嵌套的集合数据。

6.6.5 总结

- BioSQL 是一个非常灵活的，本体论驱动的，稳定的关系型模式，用来捕捉有丰富标记的数据库实体。
- 可以在生物信息多个项目之间使用 BioSQL 作为持久存储层。
- Bioperl-db 是 Bioperl 对象到 BioSQL 的对象关系映射。
- Bioperl-db 增加了透明的持久化 API 到所有支持的 Bioperl 对象。
- 当前支持的对象模型领域是：序列、特征、注释、聚集、本体。

6.7 处理 microarray 数据

就是通常所说的生物芯片。它的学名叫做基因芯片（gene chip）也叫 DNA 芯片、DNA 微阵列（DNA microarray）、寡核苷酸阵列（oligonucleotide array），是指采用原位合成或显微打印手段，将数以万计的 DNA 探针固化于支持物表面上，产生二维 DNA 探针阵列，然后与标记的样品进行杂交，通过检测杂交信号来实现对生物样品快速、并行、高效地检测或医学诊断。

Bioperl 中有一个简单处理 microarray 数据的模块包，你可以从 http://www.bioperl.org/DIST/获得，其说明书在 http://doc.bioperl.org/bioperl-microarray/，另外作者还主持了个邮件列表：http://bioperl.org/mailman/listinfo/bioperl-microarray/。

目前该模块支持 Affymetrix 公司开发的 MicroArraySuite 软件生成的文件。Affymetrix 文件主要包括 DAT 文件(图像文件, ~10^7 pixels, ~50 MB)，CEL 文件 (细胞强度文件，探针级别的 PM 和 MM 值)和 CDF 文件 (芯片描述文件)。

目前支持以下文件类型的输出输入：

- Affymetrix GeneChip CEL 文件(读和写)；
- Affymetrix GeneChip CDF 文件(读)；
- Affymetrix GeneChip Microarray Suite 5.0 normalized 文件(读)；
- Affymetrix GeneChip dChip normalized 文件(读)。

计划中的 0.2 版本将支持：

- Bio::MAGE 对象和 MAGE-ML (读和写)；

- Affymetrix SNPChip 基因型文件(读)；
- GenePix GPR 文件(读)。

第7章 Kwiki

CGI::Kwiki 是一套简单但强大的 Wiki 环境；它是由布莱恩英格森用 Perl 撰写而成的，并以 CPAN 模块的形式加以散布。

Kwiki 的中心思想是合作（Collaboraion）。这里指的不是内容合作，而是程序上的合作，虽然 Kwiki 两种都可以做到。对 Kwiki 的探索目前还没完全达成，当初设计时的目的是想要分享程序，想要设计一个架构，可以让人们设计自己想要的程序。举例来说：当有人为 Kwiki 写了新的功能，像是 Email 通知这种外挂（Plugin），其中需要加入任何设定时，相关的设定就会自动加到 Kwiki 的“偏好设定”里面，不需要开发者的主动配合。开发者只是规划 Kwiki 的世界，尽量让它足够弹性，让每个人都可以做事。可做到快速合作是这个模块很好的地方。

7.1 功能简介

那么 Kwiki 有哪些功能呢？把 Kwiki 当成 Wiki 其实是对它的误解，Kwiki 并不是一种 Wiki，它是一个让人开发 Wiki 的架构，开发者希望人们可以使用 Kwiki 来开发他们自己的 Wiki。所以 Kwiki 的每个部份都充分抽象化，它的每个部份都是一个类（class），但是还需要一个配置文件，文件的格式采用了 YAML。因为 Kwiki 非常抽象，所以很容易让其它人误会。

Kwiki 当前有两种版本：一种是官方的主版本，还有一种是唐宗汉本地化的版本，我们把后者称为汉化版。主版本对中文也有一定程度的支持，但是有时候仍然有乱码，因此只有汉化版完全支持中文。

汉化版有的功能主要是：

- Wiki 功能；
- 制作在线幻灯片的功能；
- 博客 blog。

7.2 安装 Kwiki

7.2.1 主版本的安装

这里介绍安装的是 Kwiki 的官方版本，当前的最新版本是 0.31。先准备好应有的系统环境，包括 apache、perl，以及数个相关的 perl modules，这些模块列表如下：

AppConfig-1.56

Template-Toolkit-2.13

Tie-File-0.96

IO-All-0.22

Spoon-0.16

Spiffy-0.18

安装大意是：以 root 进入设定自己的机器，cpan 安装 perl 的一个模块： Kwiki，然后把 Kwiki 的网页安装到 Apache 路径下，修改 httpd.conf 就可以使用这个 kwiki。

如果是在 Unix 下就以 root 登录，然后进入 CPAN.pm 的 Shell：

>perl -MCPAN -e 'install Kwiki;'

或者下载：

http://www.cpan.org/authors/id/I/IN/INGY/Kwiki-0.31.tar.gz

然后以 tar zxvf kwiki 把文件解压缩。

执行：

>nmake

>nmake test

>nmake install

在 Apache2 的 cgi-bin 下新建一个目录，例如 wiki。如果是在 Unix 下把 nmake 换成 make 即可。然后安装：

D:\temp\Kwiki-0.31>perl kwiki -new "C:\Program Files\Apache Group\Apache2\cgi-bin\wiki"

切换到 cgi-bin 以外的网页目录，如： /var/www/html/

更改 kwiki 的预设解开目录名称： 用指令 mv long-directory-name kwiki，然后切换入 /var/www/html/kwiki/，改 kwiki-install 为可执行 755。执行./kwiki-install –reinstall。

httpd.conf 更改，加入下面几行：

Alias /kwiki/ /var/www/html/kwiki/

Order allow,deny

Allow from all

Options ExecCGI FollowSymLinks Indexes

AddHandler cgi-script .cgi

DirectoryIndex index.cgi

重新加载 httpd.conf 设定，重新启动 Aapche 服务。

进入： http://localhost/wiki/

首页图片设定、微调：打开 kwiki/config.yaml, 更改这行设定值 kwiki_image: logo.gif 并将 logo.gif 上传到 kwiki 档案匣里面。

7.2.2 汉化版的安装

从 http://p4.elixus.org/snap/CGI-Kwiki.tar.gz 取得汉化版本。安装这个模块之前无需安装附加的模块。执行：

>perl Makefile.PL

>nmake

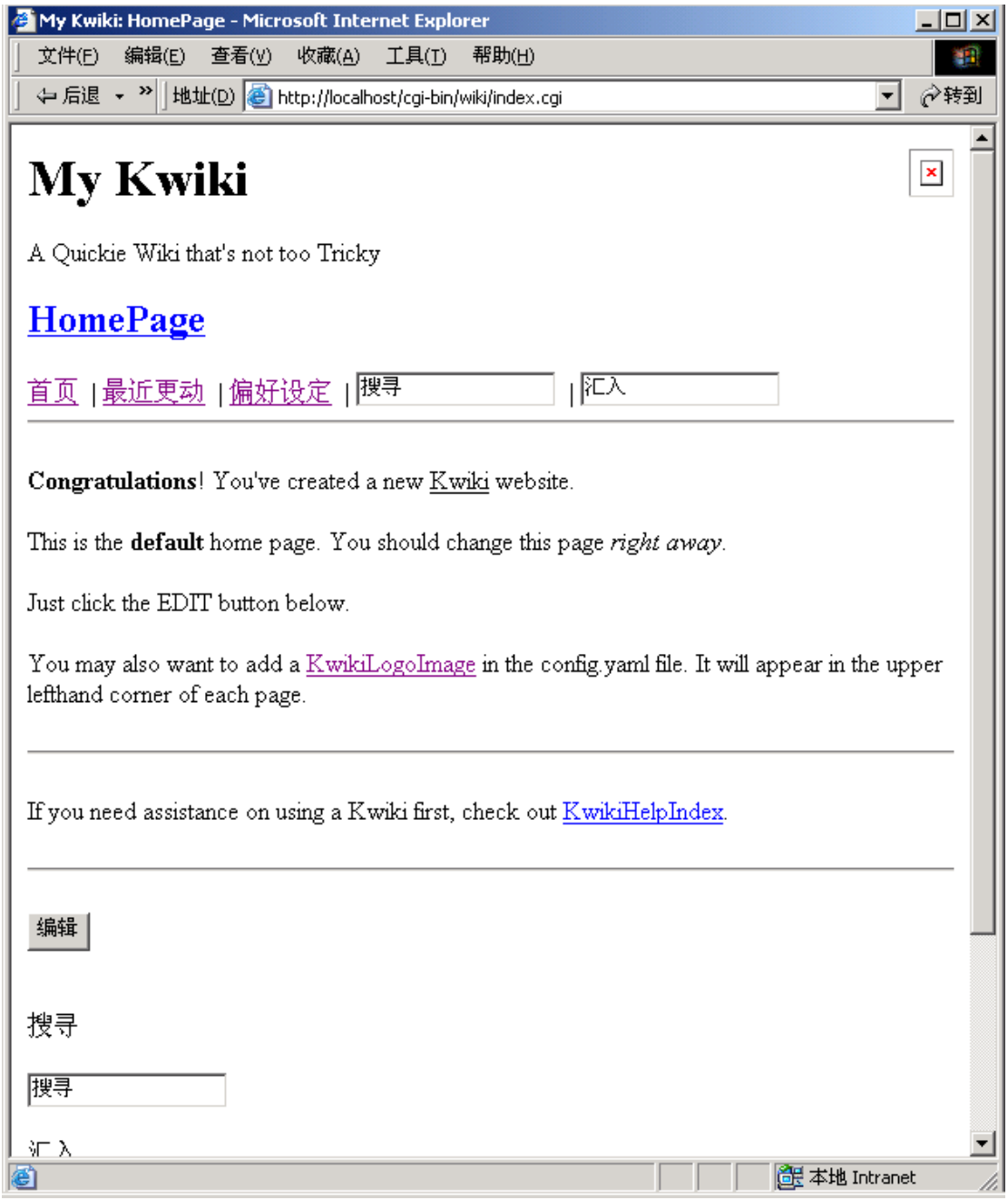
>nmake test

>nmake install

创建一个存放 wiki 网页的路径，例如 wiki。到该路径下，直接执行：

C:\Program Files\Apache Group\Apache2\cgi-bin\wiki>kwiki-install

即可。



编辑 /usr/local/apache/etc/httpd.conf，加入以下的设定。

Alias /kwiki/ /usr/local/www/data/kwiki/

<Directory /usr/local/www/data/kwiki/>

Order allow,deny

Allow from all

AllowOverride None

Options ExecCGI

AddHandler cgi-script .cgi

DirectoryIndex index.cgi

</Directory>

改好之后，告诉 apache 使设定生效。

%apachectl graceful

调整 kwiki 的设定，把透过 perl 使用 subversion 的功能开启。修改 config.yaml，把 CGI::Kwiki::Backup 改成 CGI::Kwiki::Backup::SVNPerl，并把 CGI::Kwiki::Plugin 改成 CGI::Kwiki::Plugin::Diff。

最后，告诉 kwiki 让上述设定都生效。

%kwiki-install --upgrade

如果 upgrade 失败，请检查 /usr/local/lib/perl5/site_perl/5.8.2/ 是否有 SVN 和 Text 这两个模块的目录和文件。

7.3 管理 Kwiki

7.3.1 YAML 文件格式介绍

YAML 是一个直接的机器可以解析的数据序列化格式，它设计成可以供人阅读的，并且可以使用 Perl 和 Python 这样的脚本语言交互。YAML 为数据序列化，配置设置，日志文件，网上消息和过滤优化。

例如下面这段 XML 表示的配置文件：

<bindings>

<binding>

<ircEvent>PRIVMSG</ircEvent>

<method>newUri</method>

```
      <regex>^http://.*</regex>
    </binding>
    <binding>
      <ircEvent>PRIVMSG</ircEvent>
      <method>deleteUri</method>
      <regex>^delete.*</regex>
    </binding>
    <binding>
      <ircEvent>PRIVMSG</ircEvent>
      <method>randomUri</method>
      <regex>^random.*</regex>
    </binding>
</bindings>
```

表示成 YAML 是这样的：

```
---
-
      - PRIVMSG
      - newUri
      - '^http://.*'
-
      - PRIVMSG
      - deleteUri
```

- ^delete.*

-

- PRIVMSG

- randomUri

- ^random.*

在这里，YAML 使用 --- 分割开一个文件中的文档，每个部分表示一个分隔的有向图。

7.3.2 定制 Kwiki

定制一个 Kwiki 站点有三个基本的级别。从易到难依次是：

- 改变配置文件。
- 改变 Template/CSS。

你的 kwiki 安装里有两个目录控制你的网页的布局和出现，它们分别是 template 和 css。

你可以按照适合你的方式改变 html 和 css 文件。 最好复制被修改的文件到 local/template 和 local/css。 这样你做的变化将不会因为 kwiki-install –upgrade 命令而被覆盖掉。

- 改变 Perl 代码。

应该首先把你修改的任何模板复制到"local/template/"目录。这将使他们不被 CGI::Kwiki 的升级覆盖。 CGI::Kwiki 在搜寻 template/目录之前，自动在 local/template/目录里寻找模板。

7.3.3 优化

Kwiki 装好之后，在某些速度比较慢的机器上，效能表现可能不是很好。针对 Apache Web Server，Kwiki 本身的说明文件里建议了两种方法：mod_perl 与 mod_fastcgi。搭配 mod_perl 使用 wiki 非常容易。

首先要有一份编译时就选择要支持 mod_perl 的 Apache 服务器。这方面的信息请见 http://perl.apache.org。然后按照一般的安装 wiki 步骤来安装。

最后在 Apache 配置文件里加上这些配置：

```
Alias /kwiki/ /home/ingy/kwiki/

<Directory /home/ingy/kwiki/>
    Order allow,deny
    Allow from all
    AllowOverride None
    Options None
    SetHandler perl-script
    PerlHandler CGI::Kwiki
</Directory>
<Directory /home/ingy/kwiki/css/>
    Order allow,deny
    Allow from all
    AllowOverride None
    Options None
    SetHandler none
</Directory>
<Directory /home/ingy/kwiki/javascript/>
    Order allow,deny
    Allow from all
    AllowOverride None
    Options None
    SetHandler none
```

</Directory>

这样就行了！你马上就可以体会到效能暴增。

第8章 使用 Bricolage 做内容管理

Bricolage 是一个全功能的企业级的内容管理软件。Bricolage 被设计成由非技术人员使用 。虽然修改和维护 Bricolage 需要很多经验。但是使用 Apache 或者 Perl 的人通常是程序员或者系统管理员，而使用 Bricolage 的人通常是作家，编辑或者 Web 站点的开发人员。

使用 Bricolage 的成功案例有：在美国，Bricolage 在相当长一段时间已经是 Salon 杂志的内容管理软件，同时也是 eWeek 和 Register 的内容管理软件。在台湾 http://www.pots.com.tw/和 http://publish.pots.com.tw 也采用了 Bricolage 出版系统。

Bricolage 提供易于使用的基于浏览器的界面，全功能的模版系统和灵活的完全的编程语言支持，还有许多其它的特性支持等。它运行于 Apache/mod_perl 环境，而且使用 PostgreSQL 数据库最为它的存储库。同时移植到 Oracle 数据也是可能的，Bricolage 最初的版本是支持 Oracle 的。同时它支持多国语言，特别是对繁体中文有较好的支持。简体中文也有较好的支持。

完整的英文文档可以在 Bricolage 站点(http://bricolage.cc/documentation.html)找到。这里给出了一些参考：

如果要安装或者管理系统，参见：

Bric::Admin

Bric::DBA

Bric::Security

如果要开始使用 Bricolage，参见：

Bric::ElementAdmin

Bric::AssetEditing

Bric::Alert

如果要学习如何使用 Bricolage 模版系统，参见：

Bric::Templates

Bric::AdvTemplates

Bric::HTMLTemplate

如果想了解其内部结构，可参见 Bric::Hacker。

8.1 主要特点

可以通过它管理文件的“结构”，“内容”与“产出”。结构通过元素来体现。模版定义了版面内容的形式，产出表现为输出频道。

所有的内容都有版本控制。它采用了浏览器界面，而且极度依赖 Javascript。浏览器可以是 Mozilla 或 IE。

它包含完全可自订出版的工作流程与完善的权限管理。

用户可用多重发布方式发布内容，包括：档案系统，FTP ，或者自己写个 Perl 模块发布。另外提供了 SOAP 界面发布，用户可以用命令行操作。

8.2 技术基础

每一个 CMS 的核心都是数据库。商用的 CMS 软件通常使用 Oracle 或者微软的 SQL Server 作为后端服务器。许多开源的项目，包括许多基于 PHP 的系统，使用 MySQL。 但是 Bricolage 使用 PostgreSQL 作为它的后端存储。在开发中的 Bricolage2.0 也将支持 MySQL。

PostgreSQL 是一个全功能的开源数据库，它支持事务的概念。PostgreSQL 也支持其它大型数据库支持的功能，例如视图，用户自定义函数，子查询，unions，外键和完整性检查。PostgreSQL 也支持 Unicode，在多语言站点中这个特性变得越来越重要了。

Bricolage 使用 PostgreSQL 作为它的后端存储，但是应用程序使用 Perl。在 Web 上，通常有两种方式运行服务器端的 Perl 程序：CGI，通常较慢，安全，可移植性好； mod_perl，快速的，潜在不安全而且只能在 Apache 上运行。Bricolage 在 mod_perl 下运行，意味着，它的代码——作为一个 Perl 模块的集合——编译一次，作为 Perl 字节代码缓存在内存中，然后执行多次。因此，Bricolage 运行的很快。

Bricolage 在 Apache 和 PostgreSQL 之间保持建立的数据库连接是通过 Apache::DBI 模块。这样就不需要每次用户发出请求时重新建立数据库连接。

最后，Bricolage 使用一个 Perl/HTML 模版的集合展示数据给终端用户。Perl 中有许多这样的通用或者专用于 mod_perl 的模版。Bricolage 使用了前面介绍的 HTML::Mason。

这样，它就集成了一些最令人感兴趣的技术——PostgreSQL，mod_perl，Apache 和 HTML::Mason——到一个单一的应用，这样方便了最终用户。

8.3 基本定义

Bricolage 里面有些什么东西？首先看一下内容管理系统应该解决的问题：使用内容管理系统的角色有这样几类：作者，管理者，上稿者。内容管理系统管理的稿件可能有多重分类、多重结构、多重输出格式、多重输出频道。稿件出版之前，可能需要经过层层上级审核。一篇稿子可能要输出成 html 与 pdf，分别 ftp 并且 email 到某处。

Bricolage 利用以下这些对象解决：

- 多媒体文件(Media)：

内容管理系统管理的内容就是文件。文件的格式可以是.jpg .gif .wav .mpg .avi .txt .doc。同时支持各种 MIME Type，上传时自动判断。多媒体文件可有自定义字段，但没有对应的模版。出版就是文件放到分类目录下。其它对象产生关联以“相关的媒体”(Related Media) 元素与 Media 类型的元素产生关联。

一个多媒体文件的例子如下：

Title: Bricolage talk

Location: Geego

Category: Taipei.pm

Cover Date: 2004-05-29

/taipei_pm/2004/05/29/Bricolage.mpg

- 元素 (Element)：元素定义了被管理的事务资料(Business Data)的模样。可以有各种自定义字段，也可以包含其它元素（子元素）。元素不定义输出的版面格式，只定义输出的版面资料会有什么样的内容。

- 元素种类(Element Type)：

- 模版(Templat)：通过模版来定制输出的外形。

- 资产(Asset) ：资产是 Bricolage 对其管理物的称呼。任何文件，媒体，模版，都称为“资产”。有些资产可以被出版，有些不用被出版。

- 事务数据(Business Data) ：文件各种字段的实际内容叫做事务资料。假设有个元素，

里面有个字段叫做戏弄者“teaser”，那么它的里面包含的事务资料可能就是“这个笨小丑真可爱啊”。

- 烧录器(Burner)：
- 分类(Category)：
- 工作流(Workflow)：流程就是文件在一张张的桌面间传递来传递去。预设的流程有 Edit，Copy，Legal，Publish，在此桌上的稿件表示已经可以出版了。
- 输出频道(Output Channel)：元素需关联到某个输出频道，才可以被出版。

新增文件就是依照元素所定义的结构填表。

有两种类型的“根”元素：Story 与 Media（稿件对象与媒体对象）。其余种类的元素都被当作子元素。举例来说：

Column 为 Story 类型的元素，所以它是一种“根元素”。Column 里面有 Page 元素，而 Page 元素里面有 Paragraph 字段。就此，新增一份 Column 稿件，便相当于： 对于每一页，新增一个 Page 元素，对于每一页里面的每一段，依序填入 Paragraph 字段。

树状结构的例子如下：

Column:

- title: 人咬狗

- Page:

  - Paragraph: 今天稍早，有民众检举人咬狗事件

  - Paragraph: 但狗尚未出面发表声明

只有“根元素”是可以被出版的单位。而非根元则必须直接或是间接关联到某个根元素，才会被出版。每个根元素都必须要有“title”这个字段的内容。

进行“出版”的动作时，会根据元素的种类与其对应的模版，产出排版过后的内容。

Page 是一个特殊用途的元素，在模版中有特定的函数处理分页。多媒体档案的出版，只是把档案复制过去而已。

8.4 创建新站点

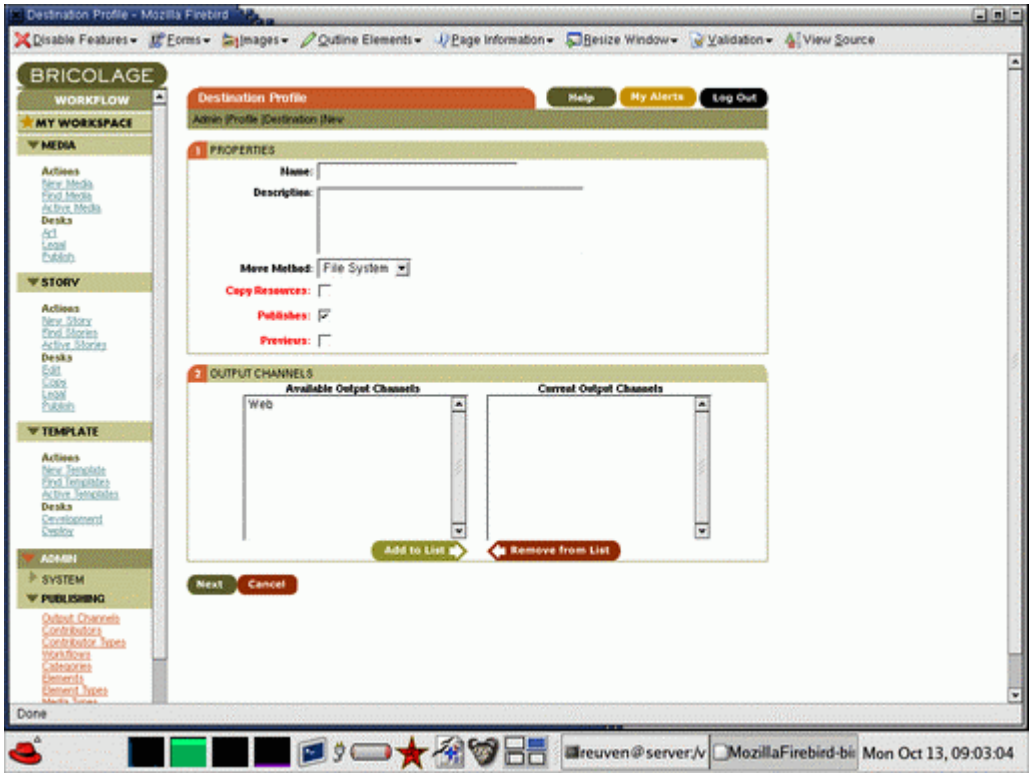
开始的第一步是创建一个可用来出版内容的站点。现在创建了一个虚拟主机叫做 output.lerner.co.il，以及它自己的错误和访问日志。我增加下面这个 VirtualHost 指示器到 Apache 配置文件。

```
<VirtualHost 69.55.225.93>

    ServerName output.lerner.co.il

    ServerAdmin reuven@lerner.co.il

    DocumentRoot /usr/local/apache/v-sites/output.lerner.co.il/www

    CustomLog /usr/local/apache/v-sites/output.lerner.co.il/logs/access-log combined

    CustomLog /usr/local/apache/v-sites/output.lerner.co.il/logs/referer-log referer

    ErrorLog /usr/local/apache/v-sites/output.lerner.co.il/logs/error-log

</VirtualHost>
```

现在，当服务器接受到到 output.lerner.co.il 的一个请求，就查找 output.lerner.co.il/www 路径而不是缺省的主文档路径。

在我们使用 Bricolage 出版任何稿件到 Web 之前，必须告诉 CMS，新文件应该存放在哪里。在 Bricolage，这个通过 Distributions 标题下的 Destinations 菜单选项设置。你可以创建 多个输出目标，允许使用一个 Bricolage 实例管理多个站点。对应这样的情况，一个出版商的编辑出版几个不同的报纸。每一个输出目的可以是在本地文件系统或者通过 FTP 访问的远程站点。

点击 New Destination 创建一个新的出版目标。大多数小站点仅需要一个频道，这样允许所有的文档输出到站点的 DocumentRoot 路径。这时候要明确：是否这个 Web 站点和 Bricolage 系统在同一台机器上，或者文件必须使用 FTP 拷贝到远程的计算机。我们假设 Bricolage 服务器和最终的 Web 服务器在同一台计算机上——但是在大规模环境，特别是为了提高性能，可能把它们分开。第一个新的目标屏幕显示如下：



尽管 Bricolage 允许定义新的输出频道（Output Channel），但是现在保持缺省的 Web 频道。输出频道和目标这两个概念容易混淆。可以把输出频道看成是逻辑目的而目标看成是物理目的，它们之间可以有各种组合方式。你可以由多个输出频道对应一个目标，或者一个输出频道对应多个目标。

元素需关联到某个输出频道，才可以被出版。输出频道将资产用某种方法移送到某主机上的某处，方法有：FTP、SFTP、WebDAV、Email、/bin/cp，或者自己撰写 Bric::Dist::Action 的子模块。

提供一些关于目标的基本的信息后，必须至少定义一个动作(“move”)，然后定义最重要的部分，服务器节。在服务器节，指明文件最终处理到那个服务器。我们使用拷贝文件而不是 FTP，因此只需填写目标路径，它匹配 Apache 虚拟主机的 DocumentRoot。

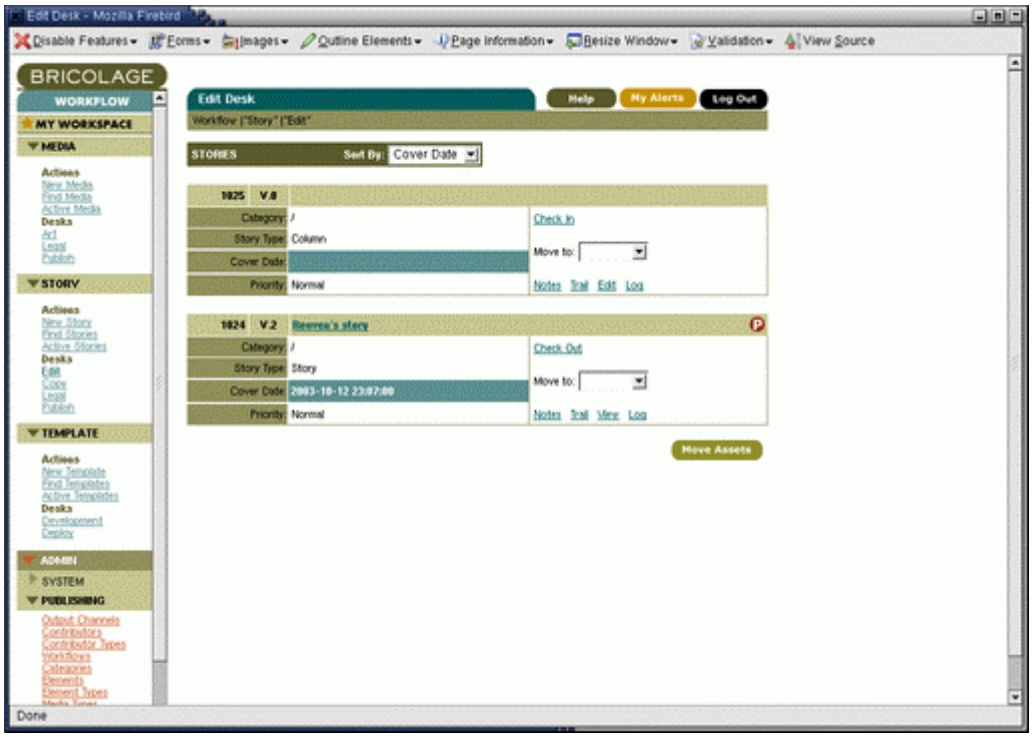
8.5 编写与出版

既然我们已经创建了一个输出频道，现在可以返回创建一个新的故事(story)。点击 Story 菜单的 New Story 项，至少给它一个标题，一个类别(/，除非你已经定义了其它的类别)和优先级(normal)。记住，每一个这样的菜单通常意味着被杂志社不同的人使用。也可以设置出版日期，它不仅表示故事什么时候应该出版到 Web，也表示了相关的 URL 什么时候可以访问。Salon.com 的长期读者可能注意到每一个故事的 URL 包含出版日期。 所以一点也不奇怪，从在 Salon 开发的内容管理系统进化而来的 Bricolage，继续了这个传统。

当已经填入了这些基本信息后，就点击 Create 按钮。现在有机会创建稿件的内容，增

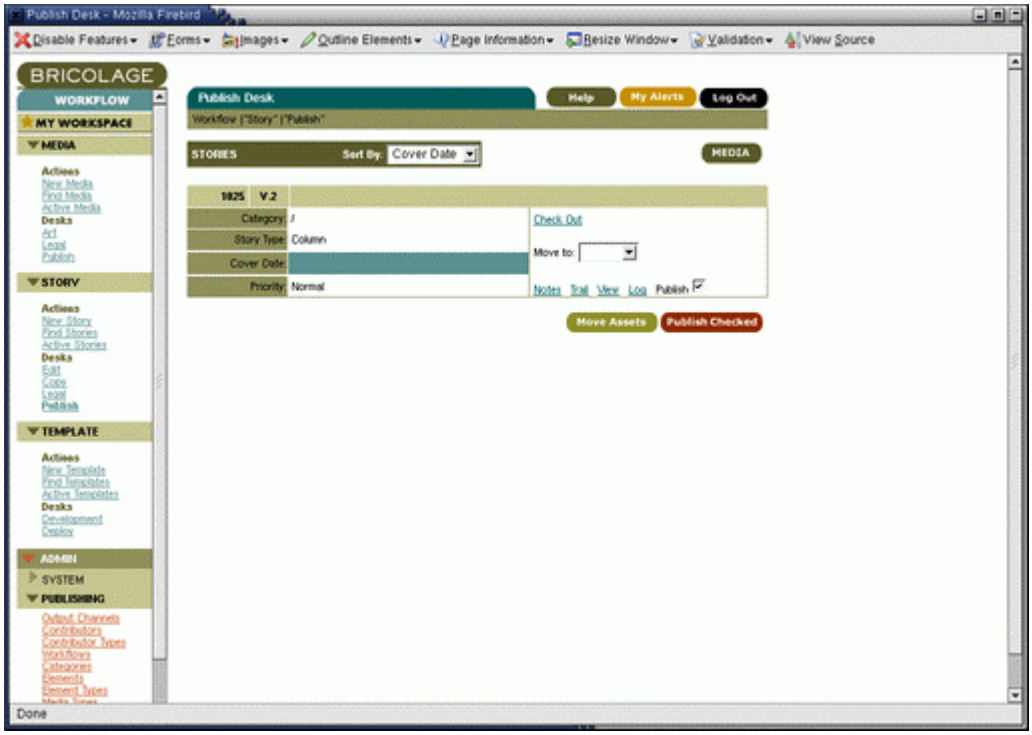
加段落(每次一个或者空行分开的使用 bulk Edit 按钮)。当你的故事完成后，可以使用页面底端的 Check In 按钮，提交它到编辑桌面，发送它到编辑桌面。

在提交稿件到编辑桌面以后，我们可以假装成编辑修改这个故事。点击 Stories 菜单下的 Desks 子主题的 Edit 菜单，你得到一个当前检入到编辑桌面的所有故事的列表。已经出版的故事在他们的标题栏有一个 P 标记。可以编辑检出的故事 (即，可以由一个编辑修改)。假设稿件不需要修改了，我们把它送到 Publish 桌面，在进行最后一次审核后，稿件将发送到 Web。我们在 Move To 菜单选择 Publish，然后点击 Move Assets。



然后，我们来到出版桌面，在这里，项目进行出版到 Web 前的最后一次审查。注意，Bricolage 对审稿和合法性评审提供了内在的支持。也可以创建新的桌面，如果你的组织结构比缺省的设置更复杂的话。

在许多方面，出版桌面都和其它的桌面一样，它允许你增加注释到故事，浏览一个故事的编辑踪迹或者甚至察看变更日志。你也可以通过把故事检出和点击编辑连接编辑出版桌面的故事。然而，出版桌面最重要的部分是附加的标志成出版的检查框和一个标志成出版检查的按钮。如下图所示：



你点击出版检查，Bricolage 给你表明什么时候故事被出版的选项。 这不同于故事的出版日期。例如，你可能想要 2 月 1 日的一个故事在 2 月 7 日可访问。

8.6 管理

当一份印刷的报纸或者杂志出版一个故事时，我们没有办法改变已经展示的内容。但是在 Web 上，你可以随心所欲的修改有问题的故事，并且每人立即看到变化。如果不考虑这个因技术上的提高导致新闻工作者的伦理学的问题，则 Bricolage 让改正一个故事然后在现场再次发布它变得容易。

去“Active Stories”菜单项修改一个已经发表的故事，它显示出一个目前可提供的故事的目录。你可以选择检出一个故事并且使它成为可供当前用户编辑(Bricolage 的版本控制系统保证同一时间只有一个人编辑某个文件)。然后你能编辑文件并且把它发送到编辑桌面，就象你最初写这个故事一样。 并且象以前一样，编辑桌面必须把它传送到出版桌面。从那里把它出版到 Web。

如果你以前只在小网站工作过，那么这项任务可以看起来太错综复杂。难道我们能改变资料并且立即看见那些变化的结果不是 Web 的优点吗？确实是这样，一个内容管理系统引入了一套折衷的方案到一个网站的运转。你不再能简单的在工作站上仅仅修改一个文件。你必须登录， 检出文件，修改，检入文件并且把它送到出版桌面，而出版桌面甚至可能拒绝你的变更。

虽然这个程序可能错综复杂而且官僚的。但是它比不同的人为一个文件的所有权而战或者不同的人在一个活跃站点同时编辑一个文件可能更好。CMS 非常象一条安全带用于限制

你的移动范围。而且，这个官僚的系统能在紧急的时候挽救你，因为资料的一切以前的版本总是存在的。 因此如果你偶然把一个重要的故事删除了一半，你能恢复到 Bricolage 已经储存在它的数据库里的以前的版本。

8.7 模版

我们通过模版来定制输出，让最终的 Web 站点吸引人。

Bricolage 以 Perl 的数种模版系统为基础，目前支持 Mason、HTML::Template、Template Toolkit，以后也会支持 XSLT。每个元素在每个分类下都有其对应的模版。就此可以根据分类不同，有丰富的输出。如果找不到，便往分类上层寻找。

Bricolage 模版内有三个全局变量：$story、$burner 和$element。最重要的是$story，它包含当前故事(story)的信息。故事包含元素，元素自己能包含另外的元素；当前元素可以通过$element 对象访问。最后，Bricolage 使用一个称为烧录器的机制把页面发送到一条输出通道(通常，但是不一定，是到一个网站)。

8.7.1 $story

是基本的「文件」对象，存取基本的信息。组件有：Title,Description,URI,Keywords,Contributors。一个故事的例子如下：

```
<html>
  <head>
    <title><% $story->get_title %></title>
    <meta name="keywords" content="<% join ', ',
        map { $_->get_name } $story->get_keywords %>" />
  </head>
  <body>
    <h1><% $story->get_title %></h1>
    <p><% $story->get_description %></p>
  </body>
```

```
</html>
```

8.7.2 $burner

烧录器($burner)处理模版架构与输出问题，这个变量得作用就像 Mason 的 $m。可用它存取相关的其它对象与资料，例如：输出频道、类别、分页的档名。

```
<%perl>

    my $next_txt = $element->get_data('next');

    my $prev_txt = $element->get_data('previous');

    my $next = $burner->next_page_file;

    my $prev = $burner->prev_page_file;

</%perl>



<!-- Show previous page link if it exists -->

% if ($prev and $prev_txt) {

        [Page <% $page %>]&lt;&lt;&lt;

        <a href="<% $prev %>"><% $prev_txt %></a>

% }



<!-- Show next page link if it exists -->

% if ($next and $next_txt) {

        <a href="<% $next %>"><% $next_txt %></a>

        &gt;&gt;&gt;[Page <% $page + 2 %>]

% }
```

8.7.3 $element

$element 表示此模版所处理的元素，它就像面向对象程序设计里面的 $self。是模版里面重要的变量。它包含以下信息：子元素，自订字段，元素的前后顺序，相关文件，相关图片。下面这个例子显示一个元素的内容。

```
<!-- The page element -->

% my $page = $burner->get_page;



<!-- Only display this title if we are on the first page -->

% unless ($page) {

        <h1><% $story->get_title %></h1>

% }



<!-- Show the content for this element -->

<%perl>

    foreach my $e ($element->get_elements) {

        if ($e->has_name('paragraph')) {

            $m->out('<p>');

            $burner->display_element($e);

            $m->out('</p>');

        } elsif ($e->has_name('pull_quote')) {

            $m->out('<p><i>');

            $burner->display_element($e);

            $m->out('</i></p>');
```

```
        } elsif ($e->has_name('inset')) {

            $burner->display_element($e);

        }

    }

</%perl>
```

8.7.4 多重模版

假设目前有「review」元素与「/reviews」「/reviews/book」两个分类目录，每个目录层下都有 review.mc 这个 Mason 模版。模版结构如下：

/review.mc

/reviews/review.mc

/reviews/book/review.mc

就此，隶属不同目录层下的 review 资产，会分别使用不同的模版输出。

8.7.5 分类的模版

就像是 Mason 的 autohandler，连名字都一样。由上而下处理：

/autohandler

/News/autohandler

/News/Politics/column.mc

模版内的 $burner->chan_next 即为下一层模版的“进入点”。

```
<!-- Code for /autohandler -->

<html>

    <head><title>The Site</title></head>

    <body bgcolor='white'>
```

```
% $burner->chain_next;

    </body>

</html>



<!-- Code for /News/autohandler -->

<h1>每周新闻</h1>

<table><tr><td width=570>

% $burner->chain_next;

</td></tr></table>



<!-- Code for /News/Politics/column.mc -->

<b><% $story->get_title %></b>

% my $n = 1;

% while (my $p = $element->get_data('paragraph', $n++)) {

<p><% $p %></p>

% }
```

8.7.6 完整的输出范例

```
<!-- Code for /autohandler -->

<html>

    <head><title>这站</title></head>

    <body bgcolor='white'>
```

```
<!-- Code for /News/autohandler -->

<h1>每周新闻</h1>

<table><tr><td width=570>



    <!-- Code for /News/Politics/column.mc -->

    <b>人咬狗</b>

    <p>今天稍早，有民众检举人咬狗事件</p>

    <p>但狗尚未出面发表声明</p>



</td></tr></table>



</body>

</html>
```

8.7.7 使用模版的例子

假设你已经创建和出版了一个故事。一旦你已经出版一个故事，将通过一条特别的输出通道发送它。发送方式可以是通过复制在文件系统上的一个文件或者通过使用 FTP 把它转移到一台远程服务器。

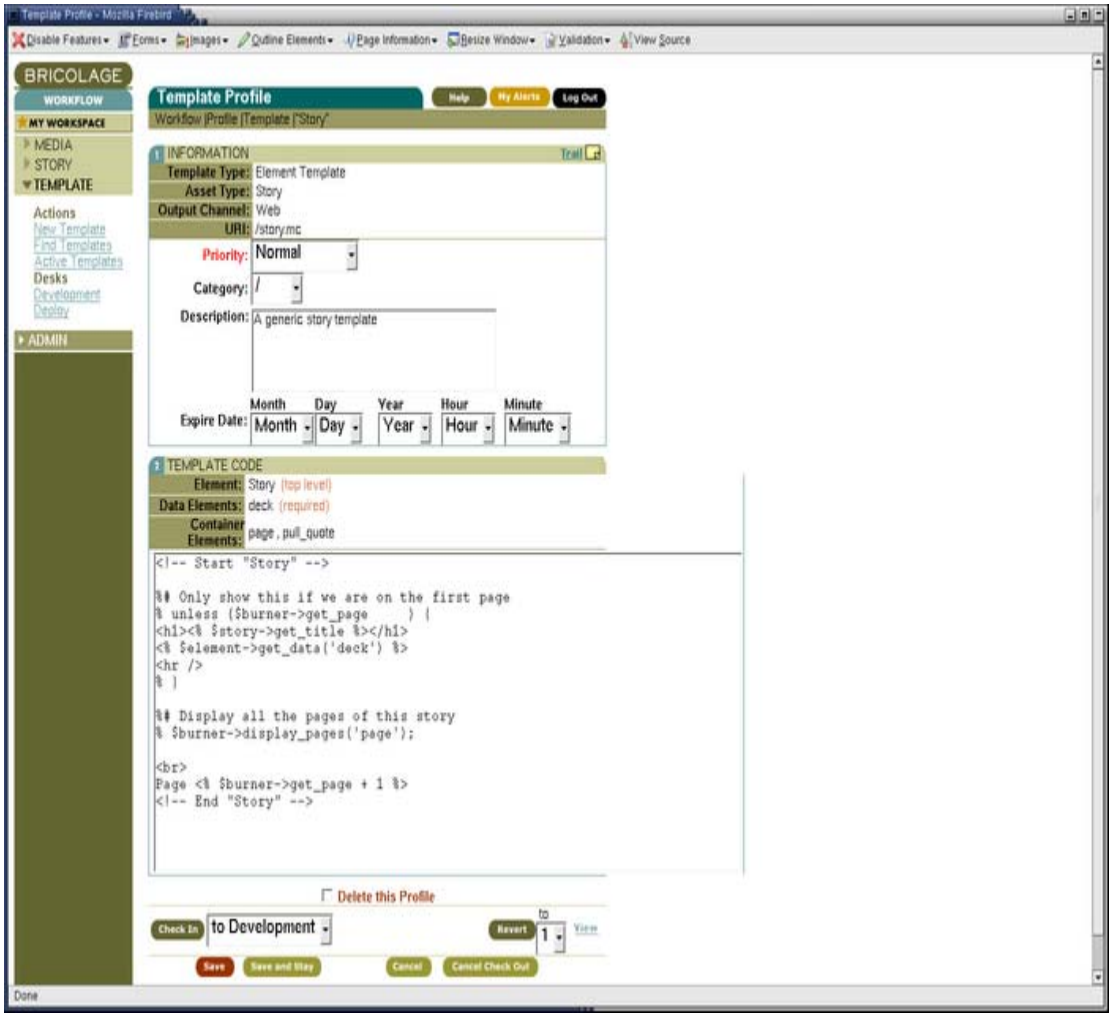
故事的外观由模版决定。在创建自己的模版之前，看一下系统自带的基本的例子。从左边的 Bricolage 管理界面(用户名是 administrator，密码是 change me now)到 Template 菜单下的 Find Template 连接，不在文本域输入任何东西，直接点击 Search 按钮 。你可以看见一个模版的列表，每一个对应系统中的一种元素类型。

Bricolage 模版在很多方面看起来都象故事：在不同的桌面已经建立，编辑并且部署他们；对他们的访问局限于某些用户和组；Bricolage 使用与一个简单的版本控制系统跟踪变化(避免碰撞)。每个模版有一个与它相关的一个版本号。通过在页的底部点击相关的检查框然后点击检出按钮，可以把每一个模版检出版本控制系统。可以从在模版菜单下的活跃的模版连接中获得检出的模版。

在 Bricolage 里，一个故事只是一类可出版的元素。更进一步，每个元素可以包含许多另外的元素。Bricolage 自带着几种预定义的顶层的元素，象故事，书评和列，以及用于被包含在其他元素的几种附加元素，比如“pull quote”。

如果你考察一份日报，你应该注意到每个栏目风格是不同的，即使对于相似的元素来说。例如， 在纽约时报的地铁部分的专栏看起来不同于商业区的专栏。Bricolage 通过允许你给一种元素分配一个种类解决这个问题。 如果你给体育专栏写一个栏目，就表明它是运动种类的一部分。Bricolage 出版那些栏目到 Web，它寻找/sports/column.mc。 如果这个模版存在，Bricolage 就应用那个特定的模版。否则，Bricolage 在最高(root)种类寻找一个 column.mc 模版。换句话说，如果你对一个元素有一个高层的模版，它就会作为在系统上的那种类型的所有元素的保底或者缺省模版。你可以通过定义种类相关的模版的方法给你的整个站点或者是站点的一部分一个不同的观感。

就象你从下图见到的那样，我检出模版/story.mc。它是故事的顶层模版。我有一个编辑连接允许我修改这个模版。 我也能通过活跃的模版页编辑这个模版。它为我提供了一个相似的编辑连接。打开这个模版用于编辑，将显示出类似下图的屏幕：



编辑一个模版类似于编辑一个故事或者任何其他的元素类型，除了你正在修改故事将被

插入其中的包容器。如果你只插入静态的 HTML，则每个元素看起来都是相同的。技巧就是使用预定义的$story，$element 和$burner 填写有动态内容的页面。 例如，这个是默认的/story.mc 模版：

```
<!-- Start "Story" -->


%# Only show this if we are on the first page

% unless ($burner->get_page          ) {

<h1><% $story->get_title %></h1>

<% $element->get_data('deck') %>

<hr />

% }


%# Display all the pages of this story

% $burner->display_pages('page');


<br>

Page <% $burner->get_page + 1 %>

<!-- End "Story" -->
```

如你所见，上述模版相当简单。一个真实的站点可能设置在 CSS 的一些样式或者包括一些额外的静态文本标识这个站点。基本版本的/story.mc 做如下事情：

- 它从$burner->get_page()得到当前的页号。页码编号从 0 开始；不过，如果我们在首页上，我们展示故事的标题和元素的装饰物。标题来自$story 对象，通过$story->get_title()方法，和来自元素自己的装饰物(一个摘要)。注意到，$element->get_data()是一个相当通用的方法； 我们能使用$element->it 检索一个元素里的任意域。

- 我们通过从烧录器请求来展示这个故事，使用$burner->display_pages('page')。

- 最后，我们使用$burner->get_page()在页面的底部再次显示页号。

如果我们删除页号并且在这个模版的顶部或者底部插入我们自己的一些静态的 HTML 将发生什么？我们的变化将从系统上的全部故事上反映出来。记住不是所有的元素都是故事，因此/story.mc 上的这些变化不会影响栏目，书评或者其他类型的元素。

你编辑/ story.mc 的时候，你能在页面的底部点击检入按钮。 当你检入一个模版时，你能把它送到开发模版桌面(默认)或者你能立刻部署它。如果你在影响整个站点的一个模版上发现错误，这种选择尤其有用。你能修改这个模版，部署它并且立即看见结果。

最后，注意到怎么/story.mc 不包含任何<html> 或<title>标记。那是因为这样的项目被实施在 autohandler 里了。您能从模版菜单察看、检出和编辑的/autohandler 模版定义如下：

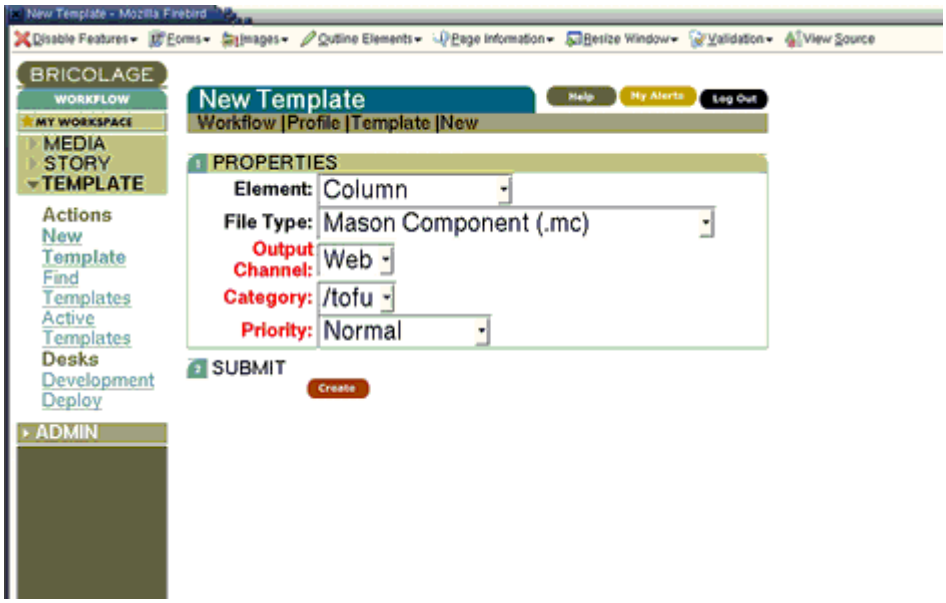
```
<!-- Start "autohandler" -->

<html>

    <head>

        <title><% $story->get_title %></title>

    </head>

    <body>



    % $burner->chain_next;

    </body>

</html>

<!-- End "autohandler" -->
```

autohandler, 对整个站点是全局性的，安置一个故事的标题在适当的 HTML <title> 标记内。 它还通过调用$burner->chain_next()整合适当的页面内容进入模版。

如果您想要包括一个全球性的 CSS stylesheet, 增加一份标准菜单到每个页面顶上或把您的公司的商标放在每个页面的顶部，就在 autohandler 实现它。并且因为 autohandlers 嵌套, 您可以为您的整体站点创建一个全局性的 autohandler, 而且在各个类别还有特定的 autohandler。

迄今为止，我们只察看了现有的模版。然而，建立一个新模版也是十分容易的。只需要

去模版菜单并且点击新模版。 你应该看见下面这样的屏幕，它让你指明你的模版应该使用的输出通道和种类。点击下一步，输入你的模版应该使用的元素类型。



“种类——频道——元素”组合必须是唯一的。因此你能有一条输出通道的多个模版。这些模版属于一个种类或者为一种特别的元素。一个例外是，在根(/)种类的 Web 输出通道内的故事元素只能有一个模版。如果你试图违反这个唯一性限制条件，Bricolage 将发布警告，告诉你那个组合已经有一个模版了。有几个解决这个问题的方法：一个是创建一个新的元素类型；另一个是建立一个新种类；还有一个是为那个组合修改现有的模版。最好的解决方法取决于你的具体目标。

我将要为在栏目内的 tofu 种类的创造新模版。在 Bricolage 里，这个模版叫做 /tofu/column.mc。一旦我点击创建按钮，就出现了允许我建立或者修改我的模版的编辑屏幕。我的模版极其简单：

```
<!-- Start "tofu/column" -->


%# Display this story

% $burner->display_pages('page');


<!-- End "tofu/column" -->
```

看看我们怎样把 HTML 注释放在定义周围。 当把它变成 HTML 并且送到用户的浏览器时，将会使调试这个模版更容易。一旦我从检入菜单选择并且发布，然后点击检入按钮，

我的模版就部署了。是 tofu 种类的任何栏目用现在这个模版而不是全局的模版格式化。

当然，如果我想要返回编辑我的模版，我能像先前提到的那样找到它，检出它，然后编辑它。

8.8 提示

我们来看一下我比较欣赏的 Bricolage 一个特征：提示。通过它可以使用户知道 Bricolage 系统的各种活动的记录。提示不但可以告诉我们发生了什么，而且还是一种了解系统工作原理的好方法，通过查看记录的对象列表，这些对象发生的动作，以及提示是如何被调用的。

8.8.1 提示

Bricolage，象大多数 CMS 软件一样，在一个传播途径里通过 station 来移动文章。 在 Bricolage 里，这些 station 被称为桌面，反映出该软件最开始用于新闻方面。 故事如此从编辑桌面开始，移到编辑部，然后到法律桌面，最后到出版桌面，其中他们实际上可能在 Web 上出版。

在只少数人们管理的一个小的网站上，跟踪哪篇文章被保持在哪里很容易。但当不再是少量用户和少量文章的时候，跟踪每篇文章发生的操作变得很困难。

处理这工作流程的一种方法是看各种各样的桌面，一次一个，看见每一个上都发生了什么操作，然后采取适当措施。 但是这样察看很乏味，并且你可能想要跟踪新闻种类的一些内容，或者跟踪某个作者，而不是所有的。而且，能通过电子邮件收到工作流事件的通知将是很让人高兴的。

Bricolage 存在这样的功能，并且是真正的开源，同时可以在很高的层次上定制。点击那些右边的系统菜单单下的提示类型，可以新建或者修改提示。这是管理员界面的菜单，只对有管理的特权的用户。结果画面，同 Bricolage 内大多数其他管理屏幕一样，允许你按名称搜索提示或者按首字母浏览提示。

8.8.2 创建提示

通过点击建立新提示的链接，建立一种新提示的类型。 将出现一个 HTML 表单要你鉴定在哪个对象上你想要得到Bricolage的提示。如果你想在story发生操作时得到通知，在story上选中提示。 如果想得到系统添加了新用户的提示，在 user 上选中提示。 简而言之，Bricolage 中的几乎任何对象都可以设置提示。

作为一个例子，让我们创造一个提示告诉我们以"inux"标题的 story 发生的从一去另一个的桌面的移动。Bricolage 使我们很容易监控任何对象。这无疑是很多提示的一种。

我们现在从管理菜单中选择创造新提示类型， 然后对 Story 对象建立了一个提示。 我们就可以获得 Bricolage 能记录的活动列表，从增加目录到出版到删除内容。从这个例子，我们选择故事移到桌面和把它命名为 Linux story 移动。

每种提示的类型必须有一位拥有者； 在该例里，拥有人是我，因为我作为自己登录。最开始 Bricolage 创建一个用户，其注册号是 admin，被称为 Bricolage 管理者。admin 用户应该与 Unix 的 root 用户一样对待。admin 用户当然能拥有提示，但是你可以建立另外的用户(使用管理/用户菜单在最左手边上的角落里)，给你自己管理的权限，然后作为自己而不是作为管理员登录。

无论如何， 点击在该页底部的按钮将带你到编辑提示类型的主页面， 用来建立新提示的类型并且修改现有的。每个提示有如下 4 个部分：

- 特性：提示类型的名字和拥有人，这些我们在前一页上输入。
- 规则：什么时候应该激发提示的描述。 每个规则由一变量(被从的下拉选项屏中选出来组成， 如此为避免可能的拼写错误)，一个比较对照，和一个用来输入比较内容的文本输入域。
- 内容：送给提示的接受者的电子邮件消息。这包括许多不同的变量，从故事的标题到它的出版日期。
- 接受者：接受提示的用户和组，你可以发送一个提示给全部编辑，全部作者或是 George 和 Frank 而不是 Deborah 和 Mary。

8.8.3 提示规则

规则或许是 Bricolage 提示系统最有趣的部分，因为它相当容易，非程序员也可以能创造并且编辑它们。不过，用=~和!~操作符还是有些潜在危险，Perl 程序员应该把它们看成是正规表达式的标识。 这是一把双刃剑，因为正规表达式对于了解的用户功能强大，但对不会使用者来说则是很危险的。

因此，我们从左边从属性目录中挑选故事标题来创造提示，从比较运算符目录选择=~操作符，在文本框里输入 Linux 这个值：

Story title =~ Linux

我们必须使用=~而并非=， 因为我们想要寻找在标题的任何部分是 Linux 而不是与整个标题相配。 如果对 Linux 或者 Perl 感兴趣如果，我们搜索：

Story title =~ Linux|Perl

有经验的 Perl 用户可能感到惊讶，=~和!~在这里是不区分大小写的。

提示的主题和正文能包含你希望的任何内容，包括插入 Bricolage 已经为我们确定的 Perl 变量。在 Java 脚本里，Bricolage 让你从一个选择表中挑选变量，避免输入错误和其他错误的可能性。 此外，这是方很便捷的方式，当用 story 工作时，你不需要记得所有可提供的变量的名字。 用这种方法，我们能确定提示的主题：

The story $title was just moved to $desk

提示消息的主体包含一条不同的消息，但是它也能包含变量。 例如，你能设定提示读取：

You asked to be notified when Linux-related articles

are moved to a new desk. Well, $trig_full_name

just moved "$title" to the $desk

desk. I hope you're happy now.

8.8.4 接受提示

当消息激发时，消息通过两种方式传递到所有的指定接受方， 一个电子邮件消息发送给每一位注册用户，通知我们在上面确定的消息已经发生。 但是 Bricolage 也在它的数据库内跟踪这些项目，使在 Web 上容易跟踪提示。 例如，如果有关 Linux 项目激发了提示，我们将得到电子邮件指示。 我们可以在提示页得到所有提示摘要， 通过点击屏幕顶上的我的提示。

提示会停留在屏幕上直到用户查看。提示屏幕不是一套长期存储系统；相反，它是一个简单的消息代理，允许编辑查看网站最近发生得所有相关事情。

你可以查看，然后从我的提示页面去除该提示，通过点击一个或多个复选框，然后点击列表的末端的标记选中为已读。还可以不用复选框而点击标记所有为已读。

如果我是一个中型网站的编辑，我将花很多时间定义提示，当重要事情发生让我知道，比如当作者将项目发给我的时候。

8.8.5 从提示获得信息

提示 Bricolage 中给用户通知相关信息的一种极好和实际的方式， 而不是迫使用户去寻找信息。另外还带来的好处是，帮助新 Bricolage 管理员、程序员能了解在系统里不同对象上发生的不同行为。 一个简单的例子是 user 对象： 我们能创造一个提示通知我们创造了用户或者删除了一个用户，这类提示对编辑来说显然不关心的，但是对系统管理员具有最高的重要性

一个更复杂的例子是模板对象， 模板确定 story 被展示的方式，因此跟踪任何修改对它们的操作是很重要的。模板相关的项目比用户更多，你可以跟踪模板什么时候被部署，被编辑以及被移到它的桌面。的确，如果你以前没理解模板有他们自己的桌面，正如 story 一样，提示的系统就会使你清楚。

我自己通过不同角度对系统探究对 Bricolage 了解了很多，包括定义提示。 如果你对 Bricolage 不熟悉并且没完全想出一切怎样工作， 甚至在读文档和展开所有菜单之后，浏览提示的定义将会使事情变得清楚。 而且， 如果你不确信，一个特别的对象怎样被在系统里处理，你总可以通过创建和注册该对象的提示，当你了解它如何工作之后再将它屏蔽。

8.9 中文化

界面中文化的方式。有三部分需要修改，分别是 zh_cn.pm，Javascript，以及相关图片的修改。可以采用 Locale::Maketext::Lexicon 进行 web 界面的汉化。

首先翻译中文资源文件(Bric/Util/Language/zh_cn.pm)

'Active' => '起用',

'Add New Field' => '增加一个新栏位',

'Add a New Alert Type' => '增加新的警告类型',

'Add a New Category' => '增加一个新的分类',

'Add a New Contributor Type' => '增加新的供稿者类型',

'Add a New Desk' => '增加一个新桌面',

'Add a New Destination' => '增加新的目标',

然后翻译界面提示 Javascript。

英文 (comp/media/js/en_us_messages.js)

var passwd_msg1 = "Passwords must be at least ";

var passwd_msg2 = " characters!";

var passwd_match_msg = "Passwords must match!";

var passwd_start_msg = "Passwords cannot have spaces at the beginning!";

var passwd_end_msg = "Passwords cannot have spaces at the end!"

中文 (comp/media/js/zh_cn_messages.js)

var slug_chars_msg =

"slug 的内容只能是字母或者数字！（A-Z,0-9,- 与 _）";

var role_msg = "你必须替这个角色取个独特的名字！ ";

var login_msg1 = "使用者名称至少要 ";

var login_msg2 = " 个字符！ ";

var passwd_msg1 = "密码至少要";

var passwd_msg2 = " 个字符！ ";

var passwd_match_msg = "密码一定要匹配！ ";

var passwd_start_msg = "密码开头不能有空白！ ";

var passwd_end_msg = "密码结尾不能有空白！ ";

var days_msg = "这个日期根本不存在！它已经被改为 ";

var data_msg = "你必须将所有栏目都填上数据！ ";

var empty_field_msg = "你必须给定这个的值 ";

var illegal_chars_msg = " 内含非法字符！ ";

var warn_delete_msg = "这样会把好些内容都删除了，您确定要继续吗？ "

最后是修改图形文件资源。

[~/dev/briczhtran/comp-media-images] ls zh_cn/

CVS/ login.gif

D_green.gif logout.gif

D_red.gif media_dgreen.gif

P_green.gif move_assets_lgreen.gif

P_red.gif my_alerts_orange.gif

acknowledge_all_red.gif my_workspace_off.gif

acknowledge_checked_red.gif my_workspace_on.gif