第8章 PHP API

PHP 是一种脚本语言，可以编写包含嵌入式代码的 Web 页面，只要访问页面就能执行这些代码，并且这些代码还能生成动态的内容作为输出的一部分发送到客户机的 Web 浏览器。本章描述了如何用 PHP编写使用MySQL的基于Web的应用程序。有关C 与 PHP 和 Perl DBI API 的比较，请参阅第5章“ MySQL 程序设计介绍 ”。

本章的实例利用了样例数据库 samp_db，这个实例使用了为学分保存方案而创建的表和第1章“ MySQL 和 SQL 介绍 ”中为历史同盟而创建的表。虽然在写本书时 PHP 4还处于ß测试阶段，而当您读到本书时它可能已经发布了，但这里还是要涉及 PHP 3的内容。与PHP3兼容是 PHP 4 的一个明确设计目标, 因此这里所说的有关 PHP 3的各个方面都可应用到 PHP4中。从 PHP 3 更改到 PHP 4 有一组移植注意事项。如果使用 PHP 4，就应该阅读那些注释。

本章假设的环境是：PHP 与 Apache Server结合使用。必须安装 MySQL C 客户机库和头文件，因为建立 PHP 时需要这些文件，否则 PHP 不知道如何访问 MySQL 数据库。如果需要获得一些这样的软件，请参阅附录 A“ 获得和安装软件 ”。附录中还给出了获得本章开发的实例脚本的说明，您可以直接下载这些脚本。

在 UNIX 环境下，PHP 或者作为内建模块用于 Apache，该模块连接到 Apache 可执行的二进制程序上；或者作为独立的用于传统 CGI 程序的解释程序。在 Windows 环境下，虽然开发在Windows NT 环境下运行的 Apache PHP 4 模块的工作正在进行之中，但是此刻的 PHP只能作为一个独立程序来运行。

本章的大部分篇幅，都用来说明了在这里讨论所需要的 PHP 函数。要想较全面地了解所有关于 MySQL 的函数，请参阅附录 H“ PHP API 参考 ”，也可以查阅 PHP 手册，它描述了 PHP 提供的全部函数，包括使用除 MySQL 以外的数据库的函数（使用 MySQL时，PHP 不受限制的部分仅仅只有 DBI）。这个手册可以从 PHP Web 站点下载：http://www.php.net/。该 Web 站点也有从 PHP 3移植到 PHP 4的注意事项。

8.1 PHP 脚本的特点

PHP 脚本的文件名带有扩展名，该扩展名允许 Web 服务器识别文件名并执行 PHP 解释程序去处理它们。如果使用了不可识别的扩展名，则 PHP 脚本被当作纯文本。本章使用的扩展名为 .php，其他通用的扩展名为 .php3 和 .phtml。有关配置 Apache 来识别扩展名的说明，请参阅附录A。在机器上，如果没有掌握 Apache 的安装，那么需要与系统管理员一起检查，找出可使用的适当扩展名。

8.2 PHP基础

PHP 的基本功能就是解释一个脚本，来生成发送到客户机的 Web 页面。具有代表性的是，脚本包括逐字发送到客户机的 HTML 和作为程序执行的 PHP 代码的混合编码。无论代码生成什么样的输出，都会发送到客户机，因此客户机永远不会看到代码，它只能看到

结果的输出。

当 PHP 开始读取文件时，假设文件内容表示文字的 HTML，则它仅仅拷贝在那里找到的输出内容。当 PHP 解释程序遇到一个特殊的打开标记时，就从 HTML 模式切换到 PHP 代码模式，而作为要执行的 PHP 代码也开始解释文件。代码的结尾由另一个特殊的标记指出，解释程序在这个位置从代码模式切换回 HTML 模式。这就允许将静态的文本（HTML 部分）与动态产生的结果（ PHP 代码部分的输出）相混合，产生依赖于调用环境变化的页面。例如，可以使用 PHP 脚本来处理表格的结果，在这个格式中，用户已经输入了数据库搜索的参数。由于格式填入内容的不同，所以每次搜索的参数可能也不同，因此当脚本执行搜索的时候，每个作为结果的页面将反映不同的搜索。

让我们通过一个非常简单的 PHP 脚本看一看它是如何工作的：

```
<HTML>
<BODY>
hello, world
</BODY>
</HTML>
```

这个脚本并不很有趣，因为它不包括 PHP 代码！因此您会问：那它有什么好处？这个问题的回答是：它有时有助于建立包括想要生成页面的 HTML 框架的脚本，然后再加入 PHP 代码。这是非常有效的， PHP 解释程序用于它是没有问题的。

为了在脚本中包括 PHP 代码，您可从用两个特殊标记（脚本开始处的‘ <?php ’和脚本结束处的‘ ?> ’）把它与周围的文本区分开来。当 PHP 解释程序遇到开始的‘ <?php ’标记时，就从 HTML 模式切换到 PHP 模式，并解释它找到的任何 PHP 代码，直到看见结束的‘ ?> ’标记为止。它产生的所有输出解释并替换了两个标记之间的脚本。将前面的实例再重新编写一下，它包括了少量的 PHP 代码，如下所示：

```
<HTML>
<BODY>
<?php print ("hello, world\n"); ?>
</BODY>
</HTML>
```

此时，代码部分是很小的，由单行组成。当解释代码时，产生了输出“ hello, world ”，它作为输出部分发送到客户机浏览器。这样，这个脚本产生的 Web 页面与前面实例产生的 Web 页面一样，前面实例的脚本完全由 HTML 组成。

可以使用 PHP 代码产生 Web 页面的任何部分。我们已经看到了一个特别的实例，在那里整个脚本都由文字的 HTML 组成，而不包括 PHP 代码。另一个特别的实例是整个脚本都是 PHP 代码而不包括文字的 HTML：

```
<?php
print ("<HTML>\n");
print ("<BODY>\n");
print ("hello, world\n");
print ("</BODY>\n");
print ("</HTML>\n");
?>
```

这说明 PHP 在如何产生输出方面有很大的灵活性。但 PHP 也留下一个问题，那就是确定如何组合 HTML 和 PHP 代码才是合适的。不必把所有代码都放在一个地方， PHP 在这方面也很灵活。只要您高兴，就可以通过脚本在 HTML 和 PHP 代码模式之间进行转换。

PHP 脚本标记

除了本章实例中使用的标记之外，PHP还支持其他的脚本标记。您可以在其他人编写的 PHP 代码中看到它们，或者可以自己使用这些标记。PHP 识别四种标记风格：

- 缺省标记风格。这是PHP 配置为缺省时使用的风格：

```
<?php print ("hello, world\n"); ?>
```

- 简洁开标记风格。这个风格除了开标记较简洁外，其他与缺省风格相类似：

```
<? print ("hello, world\n"); ?>
```

- 兼容 ASP 的风格。这个风格在 Active Server Page 环境内部是通用的：

```
<% print ("hello, world\n"); %>
```

- <SCRIPT> 标记风格。如果使用与其他标记风格不同的 HTML 编辑器，这个风格是有用的。当然，它比较冗长，但是当您使用其他标记风格时，编辑器可能不处理 PHP 代码，这时您将发现它是必需的：

```
<SCRIPT LANGUAGE="php"> print ("hello, world\n"); </SCRIPT>
```

缺省时，简洁开标记风格和兼容 ASP 风格无效。请参阅附录 H 有关开启它们的说明。

独立的 PHP 脚本

当处理命令解释程序脚本或 Perl 脚本时，可以编写从命令行调用的独立的 PHP 脚本。这里有一个实例：

```
#! /usr/local/bin/php -q
<?php print ("hello, world\n"); ?>
```

前面的脚本可命名为 hello.php，利用 chmod +x 使之成为可执行的，并从命令解释程序中调用：

```
% hello.php
hello, world
```

本章中我们不编写任何独立的脚本。这里编写的所有实例都期望 Web 服务器调用它们来生成 Web 页面。

下一个脚本有一些实质性的内容，但仍然相当简短。它说明了如何较容易地从 PHP 访问 MySQL 数据库，并在 Web 页面中使用查询结果。此脚本在第 5 章表达得很简短，它形成了历史同盟 Web 站点主页的基础。在我们继续往下做的时候，应该使脚本更精巧一些，但直到现在为止，它所做的只是显示简短的欢迎消息和当前同盟会员资格的计数：

```
<HTML>
<HEAD>
<TITLE>US Historical League</TITLE>
</HEAD>
<BODY>
<P>Welcome to the US Historical League Website.
<?php
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
mysql_select_db ("samp_db")
    or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM member")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>The League currently has " . $row[0] . " members";
```

```
mysql_free_result ($result);
?>
</BODY></HTML>
```

欢迎消息就是一个静态文本，因此它作为文字的 HTML 是最容易编写的。另一方面，会员资格的计算是动态的，而且随时会更改，因此必须在它不工作时通过查询 samp_db 数据库中的 member 表来确定。

在开和闭脚本标记之间的代码文本执行一个简单的任务。首先，它打开了与 MySQL 服务器的连接，并使 samp_db 数据库成为缺省数据库。然后，它发送一个到服务器的查询来确定此刻历史同盟有多少成员（我们将它作为 member 表中的行数）。查询结果将作为包括会员资格计数消息的一部分显示出来，然后再做处理。

在处理过程中，如果任何一点发生错误，该脚本都会简单地退出。它由于太简单而不产生错误输出，因此把访问该 Web 站点的人们搞糊涂了（如果依靠 PHP 代码生成整个 Web 页面，当因错误而退出，且不产生任何输出时，可能会使访问页面的人非常恼火，因为有些浏览器将显示“这个页面没有数据”的对话框）。

让我们把该脚本分成几段，看看它是如何工作的。第一步是用 mysql_pconnect() 连接到服务器：

```
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
```

mysql_pconnect() 把主机名、用户名和口令作为参数。连接建立成功，则返回连接标识符，如果发生错误，则返回 FALSE。如果连接失败，脚本立刻调用 exit() 来结束脚本，并且不再产生输出。

在 mysql_pconnect() 调用前的‘@’字符是什么意思呢？就是“请关闭”的意思。有些 PHP 函数在失败时除了返回状态代码之外还写出错误消息。在 mysql_pconnect() 情况下，失败的连接导致了下面的消息出现在发送到客户机浏览器的 Web 页面上：

```
Warning: MySQL Connection Failed: Access denied for user:
'paul@pit-viper.snake.net' (Using password: YES)
```

这样很难看，参观我们站点的人们可能不知道它是如何造成的和该怎样处理它。把‘@’字符放到 mysql_pconnect() 调用的前面，就可以取消这个错误消息，以使我们能在返回值的基础上自己选择如何处理错误。对于该脚本，如果发生错误，最好根本不产生属于会员资格计数的输出。这样，页面将只包含欢迎消息。

任何 PHP 函数都可以使用‘@’，但以笔者的经验来说，初始的 mysql_pconnect() 调用是最可能失败的一个。因此，本章的例子禁止来自该函数中的消息。

将名称和口令嵌入到所有人都能看到的脚本中，可能会使您紧张不安。是这样的，名称和口令出现在发送到客户机的 Web 页面上是对的，因为该脚本的内容由其输出的结果替换掉了。然而，如果 Web 服务器不知何故配置不当，没有识别出脚本需要由 PHP 代码来处理，它就会将脚本以纯文本发送出去，且会暴露连接参数。在 8.2.1节“使用函数和 include 文件”中，将简要地处理这类情况。

mysql_pconnect() 返回的连接标识符可以传递到 PHP API 中的几个与 MySQL 相关的调用中。然而，对于这样的调用，标识符总是可选择的。例如，可以使用下列格式之一来调用 mysql_select_db()：

```
mysql_select_db ($db_name, $link);
```

```
mysql_select_db ($db_name);
```

mysql_pconnect() 与 mysql_connect() 的对比

函数 mysql_pconnect() 与函数 mysql_connect() 相似，都具有主机名、用户名和口令参数，并返回连接标识符或 FALSE 来说明连接是否成功。两个调用之间的不同在于：mysql_pconnect() 建立了一个持久的连接，而 mysql_connect() 建立了一个非持久的连接。与非持久连接不同，持久连接在脚本终止时不关闭。如果另外一个 PHP 脚本随后由同一个 Apache 子处理执行，并用同样的参数调用 mysql_pconnect()，将重新使用这个连接。这比关闭后再建立连接的效率要高。

如果忽略了从一些相关 MySQL 的 PHP 调用中得到的连接参数，调用就会使用最近打开的连接。这样，如果脚本只打开单个连接，那么在任何 MySQL 调用中永远不必指定连接参数——即连接是缺省的。这就是 C 或 DBI API 与 MySQL 程序设计的极大不同之处，因为它们没有这样的缺省。

笔者使用 $link 变量在简单的主页上编写了下面的连接代码，使 mysql_pconnect() 返回哪种类型的值更加清晰：

```
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
```

然而，实际上，我们在脚本的其他地方都没有使用 $link，因此代码可更简单地写成：

```
@mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
```

假设连接建立成功，则下一步是选择一个数据库：

```
mysql_select_db ("samp_db")
    or exit ();
```

如果 mysql_select_db 失败，我们将会默默地退出。如果我们能够连接到服务器，并且数据库存在，那么似乎不可能发生错误，但是仍然要严格地检查问题并采取相应的行动。选择数据库之后，可将查询发送到服务器，提取结果，加以显示，然后释放结果集：

```
$result = mysql_query ("SELECT COUNT(*) FROM member")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>The League currently has " . $row[0] . " members";
mysql_free_result ($result);
```

mysql_query() 函数将查询发送到服务器中去执行。查询不用分号或者‘ \g ’终止。如果查询非法或因为某些原因不能执行，则 mysql_query() 返回 FALSE，否则返回一个结果集标识符。该标识符是我们能用来获得有关结果集信息的值。对于查询，该结果集由表示会员资格计数的单列值的单行组成。为得到这个值，我们可以把结果集标识符传给 mysql_fetch_row() 来获取行，将此行赋给变量 $row，并以 $row[0] 形式访问第一个元素（只有一个元素时也是这样）。

当处理完结果集时，将结果集传递给 mysql_free_result() 进行释放。实际上这种调用在我们的脚本中是不必要的，因为当脚本结束时，PHP 会自动地释放所有活动的结果集。

mysql_free_result() 有助于执行大型查询或大量查询的脚本。它防止大量内存的使用。

为了使用脚本，需要在某处安装它。本章将采用这样的约定：美国历史同盟在 Apache 文档树的最高一级中有自己的目录，称为 ushl，因此主页面脚本作为该树的 ushl/index.php 进行

安装。我们也将为学分保存方案开发脚本，因此给出目录 gp。如果 Web 站点主机是 pit-viper.snake.net，那么这两个目录中的页面将有如下开头的 URL：

```
http://pit-viper.snake.net/ushl/
http://pit-viper.snake.net/gp/
```

例如，每个目录的主页面都可称为 index.php，并以如下方式进行访问：

```
http://pit-viper.snake.net/ushl/index.php
http://pit-viper.snake.net/gp/index.php
```

8.2.1 使用函数和include 文件

PHP 脚本与 DBI 脚本的不同之处在于，PHP 脚本位于 Web 服务器文档树的内部，而 DBI 脚本位于 cgi-bin 目录中，这个目录在文档树的外部。这就提出了一个安全性问题：服务器配置不当的错误可能导致位于文档树内部的页面会以纯文本方式泄露给客户机。这意味着建立与 MySQL 服务器连接的用户名和口令如果在 PHP 脚本而非 DBI 脚本中使用，则将处于暴露给外界的很高的危险之中。

PHP 中的变量

在PHP中，可以通过简单地使用变量而使它们存在。主页脚本使用了三个变量：$link、$result 和 $row，没有一个变量会在所有地方都声明（声明变量的地方有上下文，如在函数内部引用全局变量时，我们随后会谈到这个问题）。

变量由美元符号（‘ $ ’）为开头的标识符表示。无论它表示什么类型的值都是正确的，尽管对于数组和对象要添加一些额外内容来访问值的单个元素。如果变量 $x 表示单个值，例如，数字或字符串，则可以写成 $x 来访问它。如果 $x 表示有数字索引的数组，则可以写成 $x[0]、$x[1] 等等来访问它的元素。如果 $x 表示有关联索引的数组，如“ yellow ”或者“ large ”，则可以写成 $x[“ yellow ”]、$x[“ large ”] 来访问它的元素。

PHP 数组可以同时拥有数字索引的元素和相关索引的元素。例如，$x[1] 和 $x[“ large ”] 都能作为同一数组的元素。如果 $x 代表一个对象，则可写成 $x->property_name 来访问它所具有的属性。例如，$x->yellow 和 $x->large 都是 $x 的属性。作为属性名，数字是不合法的，因此 $x->1 在 PHP 中是不合法的。

初始的历史同盟主页脚本也存在这个问题，因为它包括 MySQL 用户名和口令的直接值。让我们用两个 PHP 性能：函数和 include 文件，把这些连接参数移到脚本外面。我们将编写函数 samp_db_connect()来建立这个连接，并把函数放到一个 include 文件中——不是主脚本部分的，但可以从脚本中引用的文件。这种方法的优点如下：

- **编写连接建立代码比较容易。**不需要写出所有参数，就可以在连接后用 samp_db_connect() 选择数据库，使一个函数可以进行两个 PHP 函数的工作。由于你可以将精力集中于脚本的独特标记，而不必为连接建立代码分心，因此也使得脚本更加易于理解。
- **可从脚本中访问 include 文件，但可移到 Apache 文档树的外面。**这使它的内容对于客户机来说是不可访问的，即使 Web 服务器配置不当，连接参数也不会暴露给它们。使用 include 文件，对于隐藏不想由 Web 服务器发送到站点的任何类型的敏感信息都是个良策。

  虽然如此，但这并不意味着用户名和口令在任何意义上说都是安全的。如果没有采取预防措施，在 Web 服务器主机上注册的用户，能够直接读取 include 文件。请参阅

7.4.3节“从 Web 脚本连接到 MySQL 服务器”关于安装 DBI 配置文件所描述的预防措施，它们用于保护口令和用户名不受其他用户侵害。对 PHP 的 include 文件也要应用同样的防范措施。

PHP 语言上的影响

如果有 C 程序设计经验，则可能注意到：脚本中许多语法的结构与 C 程序设计中的非常类似。实际上，PHP 语法很大程度上来自于 C，因此这种相似处并不是巧合。如果有些 C 的背景知识，就可以将它的许多内容转换到 PHP。事实上，如果不能确信如何用 PHP 编写表达式或控制结构，则可以试用 C 中编写它们的方法，这很可能也是正确的。虽然 PHP 的基本部分主要在 C 中，但也包含了使用 Java 和 Perl 的成分。可以在注释语法中查看它，在那里允许以下形式：

```
# Perl-style comment from '#' to end of line
// C++- or Java-style comment from '//' to end of line
/* C-style comment between slash-star to star-slash */
```

Perl 的其他相似性包括‘ . ’字符串连接操作符（包括‘ .= ’作为额外的连接），变量引用和转义序列的方法是在双引号内而非单引号内解释的。

- include 文件可以由多个脚本使用。这提高了代码的可重用性，使代码更加可维护。同时也允许对访问这个文件的每个脚本不费力地做出全局性的更改。例如，如果我们将数据库 samp_db 从 pit-viper 移动到 boa，则不必更改一簇单个脚本，而只要更改包含 samp_db_connect() 函数的include 文件中 mysql_pconnect() 调用的主机名参数即可。

为了使用include 文件，必须有存放它们的地方，而且必须使 PHP 找到它们。如果系统已经有了这样一个位置，则可以使用。如果没有，则使用下面的过程建立一个 include 文件的位置：

1) 创建一个目录来存放PHP 的 include 文件。该目录不能位于 Web 服务器文档树内部！笔者使用 /usr/local/apache/php 的 PHP include 目录，它与我的文档树在同一层次上（/usr/local/apache/htdocs），而不是在其内部。

2) 通过完整的路径名或者告诉 PHP 在搜索时寻找哪个目录来引用 include 文件。后者的方法更方便些，因为如果我们使用了文件的基名，PHP 就会找到它（PHP include 文件与 C 头文件有些类似，其中包括的PHP 将在多个目录中搜寻include 文件，就像 C 预处理程序在多个目录中搜寻 C 头文件一样）。为了告诉 PHP 去哪里查看，修改 PHP 初始化文件（系统上的 /usr/local/lib/php3.ini）来改变 include_path 的值。如果它没有值，可以将它设置为新的包含路径的完整路径名：

```
include_path = "/usr/local/apache/php"
```

如果 include_path 已经有值了，则把新的目录加到那个值中：

```
include_path = "current_value:/usr/local/apache/php"
```

3) 创建想使用的include 文件并将它放到include目录中。文件应该有一些有特点的名称，为了这个目的，这里我们使用 samp_db.inc。它的内容将在下面列出。对于我们这里开发的脚本，当连接到 MySQL 服务器上时，会一直使用 samp_db 数据库，因此连接函数 samp_db_connect() 也可以为我们选择 samp_db 数据库。如果连接成功并选择了这个数据库，这个函数就返回一个连接标识符；如果发生错误，则返回 FALSE。发生错误时将不打印消息，

并且允许调用者静静地退出，或者在环境允许时再打印消息。

```
<?php
# samp_db.inc
# samp_db sample database common functions
# Connect to the MySQL server using our top-secret name and password

function samp_db_connect ()
{
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret");
    if ($link && mysql_select_db ("samp_db"))
        return ($link);
    return (FALSE);
}
?>
```

观察一下，samp_db.inc 文件的内容由‘ <?php ’和‘ ?> ’括在一起。这是因为 PHP 是在 HTML 模式中开始读取文件的。如果没有这些标记 PHP 会把文件以纯文本发送出去，而不是作为 PHP 代码解释。如果想在文件中包含文字的 HTML 是很好的选择。但是，如果文件包含 PHP 代码，就必须在脚本标记内部封闭代码。

4) 使用下面的行从脚本中引用文件：

```
include ("samp_db.inc");
```

当 PHP 看到这一行时，就搜寻文件并读取内容。对于脚本的下列部分，文件中的任何事物都变成可访问的。

在建立了我们的include 文件 samp_db.inc 之后，就可以修改历史同盟主页来引用 include 文件，并通过调用 samp_db_connect() 函数连接到 MySQL 服务器上：

```
<HTML>
<HEAD>
<TITLE>US Historical League</TITLE>
</HEAD>
<BODY>
<P>Welcome to the US Historical League Website.
<?php
include ("samp_db.inc");'

samp_db_connect ()
    or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM member")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>The League currently has " . $row[0] . " members";
mysql_free_result ($result);
?>
</BODY></HTML>
```

include( ) 与 require( ) 的对比

PHP 的 require() 性能与 include() 相类似。不同之处在于，对 include() 来说，在 include() 执行时文件每次都要进行读取和评估；而对于 require() 来说，文件只处理一次（实际上，文件内容替换了 require() 语句）。这意味着如果有包含这些指令之一的代码和可能执行多次的代码，则使用 require() 的效率比较高。另一方面，如果每次执行代码时想读取不同的文件，或者有通过一组文件迭代的循环，就使用 include()，因为可以给想要包括的文件名设置一个变量，当参数为 include() 时使用这个变量。

samp_db.inc 文件对其他函数也是有用的，我们可以将它作为各种其他事物的储藏库。实际上，还可以再创建两个函数放入到文件中。我们编写的每个脚本在页面的开头都会产生一组相当醒目的 HTML 标记，而另一组在结尾。不必在每个脚本中将它们逐字地写出，我们可以通过编写函数 html_begin() 和 html_end() 来做这些事。函数 html_begin() 能够提取几个指定了页面标题和头的参数。两个函数的代码如下：

```
# Put out initial HTML tags for page.  $title and $header, if
# present, are assumed to have any special characters properly
# encoded.

function html_begin ($title, $header)
{
    print ("<HTML>\n");
    print ("<HEAD>\n");
    if ($title)
        print ("<TITLE>$title</TITLE>\n");
    print ("</HEAD>\n");
    print ("<BODY>\n");
    if ($header)
        print ("<H2>$header</H2>\n");
}


# put out final HTML tags for page.

function html_end ()
{
    print ("</BODY></HTML>\n");
}
```

然后我们可以修改历史同盟主页来使用这两个新函数，如下所示：

```
<?php
include ("samp_db.inc");

$title = "US Historical League";
html_begin ($title, $title);
?>
<P>Welcome to the US Historical League Website.

<?php
samp_db_connect ()
    or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM member")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>The League currently has " . $row[0] . " members";
mysql_free_result ($result);

html_end ();
?>
```

请注意代码被分成了两块，两块代码之间出现了欢迎消息的文字 HTML 文本。

产生页面开始和最后部分的函数用法给了我们一个重要的能力。如果想改变使用这些函数的页面头和尾的外观，可以在函数中包含一些代码。使用它们的每个脚本也都将自动地受到影响。例如，您可以把消息“ Copyright USHL ”放在每个历史同盟页面的底部。页面尾部函数 html_end() 会很容易地做到这一点。

8.2.2 一个简单的查询页面

已经嵌入到历史同盟主页中的脚本运行了一个只返回单个行的查询。下一个脚本介绍了如何处理多行的结果集。它获取并显示了 member 表中的内容。这就是第 7 章开发的 dump_members DBI 脚本的 PHP 等价物，因此称它为 dump_members.php。它与 DBI 版本的不同之处在于，它希望在 Web 环境中使用而不是在命令行中使用。由于这个原因，它需要产生 HTML 输出而不是简单地写出制表符分隔的文本。为了使行和列漂亮地排列，我们将以 HTML 表形式编写会员资格记录。脚本如下：

```
<?php
# dump_members.php - dump Historical League's membership list

include ("samp_db.inc");

$title = "US Historical League Member List";
html_begin ($title, $title);

samp_db_connect ()
    or die ("Cannot connect to server");

# issue query
$query = "SELECT last_name, first_name, suffix, email,"
    . "street, city, state, zip, phone FROM member ORDER BY last_name";
$result = mysql_query ($query)
    or die ("Cannot execute query");

print ("<TABLE>\n");
# read results of query, then clean up
while ($row = mysql_fetch_row ($result))
{
    print ("<TR>\n");
    for ($i = 0; $i < mysql_num_fields ($result); $i++)
    {
        # escape any special characters and print
        printf ("<TD>%s</TD>\n", htmlspecialchars ($row[$i]));
    }
    print ("</TR>\n");
}
mysql_free_result ($result);
print ("</TABLE>\n");

html_end ();
?>
```

这个脚本使用了 die() 函数来打印消息，如果发生错误则退出（die() 函数与 exit() 函数相类似，但是它在退出之前打印消息）。与我们在历史同盟主页中使用的静静地退出方法相比，这是一种不同的错误处理方法。在 dump_membes.php 中，我们希望看到一个特殊的结果，因此打印错误消息来说明发生的问题是有道理的。

脚本可以安装在 ushl 目录并用 http://pit-viper.snake.net/ushl/dump_members.php 访问。可以在历史同盟主页的新脚本中增加一个连接，以便人们了解它：

```
<?php
include ("samp_db.inc");

$title = "US Historical League";
html_begin ($title, $title);
```

```
?>

<P>Welcome to the US Historical League Website.

<?php
samp_db_connect ()
    or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM member")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>The League currently has " . $row[0] . " members.";
mysql_free_result ($result);
?>

<P>
You can view the directory of members
<A HREF="dump_members.php">here</A>.

<?php
html_end ();
?>
```

8.2.3 处理查询结果

在这一节中，我们将更细致地检查如何执行 MySQL 查询并处理结果集。在 PHP 中，所有的查询都通过调用 mysql_query() 函数来发布，这个函数提取查询字符串和连接标识符作为参数。连接标识符是可选的，因此可以用下面任意一种形式调用 mysql_query()：

```
$result = mysql_query ($query, $link);    # use explicit connection
$result = mysql_query ($query);           # use default connection
```

对于不返回行的查询（非 SELECT 的查询，如 DELETE、INSERT、REPLACE 和 UPDATE），mysql_query() 返回 TRUE 或者 FALSE 说明查询成功或者失败。为了获得成功的查询，可以调用 mysql_affected_rows() 找出有多少行被改变（可能是删除、插入、替换或者更新）。

对于 SELECT 语句，mysql_query() 返回结果集标识符或者 FALSE 说明查询是成功或是失败。为了获得成功的查询，使用结果集标识符可以获得更多的有关结果集的信息。例如，可以找出结果集有多少行或列，或者提取这个结果集内部包括的行。

当 mysql_query() 返回 FALSE（也就是零）时，意味着查询失败 —— 换句话说就是发生一些错误而不能执行查询。查询失败可能有下面几个原因：

- 查询可能是畸形的并包含语法错误。
- 查询依照语法可能是正确的，但是在语义上却是无意义的，例如要从不包含列的表中选择列时。
- 没有充分的权利执行查询。
- 由于网络问题，可能已经连接不到 MySQL 服务器主机。

在以上每种情况中（还有其他情况），mysql_query() 都返回 FALSE。如果要想知道错误的详细原因，就调用 mysql_error() 或者 mysql_errno() 获得错误消息字符串或者数字错误代码（请参阅8.2.4节“处理错误”一节）。

考虑 mysql_query() 造成的两种最常见的错误，返回值是行计数，或者它包含查询返回的数据，两者都是错误的。

1. 处理不返回结果集的查询

下面的代码使用 DELETE 说明了如何处理不返回任何行的查询：

```
$result = mysql_query ("DELETE FROM member WHERE member_id = 149");
if (!$result)
    print ("query failed\n");
else
    printf ("number of rows deleted: %d\n", mysql_affected_rows ());
```

如果有一个 ID 号为 149 的成员，MySQL 就删除记录，且mysql_query() 返回 TRUE。如果没有这样的成员呢？这种情况下 mysql_query() 仍然返回 TRUE！这使将精力耗在误解 mysql_query() 的返回值是行计数的人们非常惊讶。两种情况都返回 TRUE，是因为不管实际上是否删除了一些行，查询都是合法的。由查询作用的行的数目则是完全不同的事。若要在查询成功之后提取值，可调用 mysql_affected_rows()。

2. 处理返回结果集的查询

下面的实例提供了 SELECT 查询处理的大致概况：

```
$result = mysql_query ("SELECT * FROM member");
if (!$result)
    print ("query failed\n");
else
{
    print ("number of rows returned: %d\n", mysql_num_rows ($result));
    while ($row = mysql_fetch_row ($result))
    {
        for ($i = 0; $i < mysql_num_fields ($result); $i++)
        {
            if ($i > 0)
                print (",");
            print ($row[$i]);
        }
        print ("\n");
    }
    mysql_free_result ($result);
}
```

不要假设 mysql_query() 会成功

在 PHP 邮件发送清单中，新的 PHP 用户会共同询问：执行脚本时为什么会发生下面的错误消息：

```
Warning: 0 is not a MySQL result index in file on line n
```

这个消息将为零的结果集标识符的值传递给了期望有效结果集的一些函数（如提取行的函数）。这意味着早时的 mysql_query() 调用返回了零值 —— 就是 FALSE。换句话说就是mysql_query() 失败，并且在由其他函数使用它之前，脚本编写程序对检查返回值并不烦恼。在使用 mysql_query() 时，要一直检测返回值。

如果查询失败，结果将为 FALSE，对这个结果我们只是打印一条消息（取决于环境，其他对错误的反应可能更合适）。如果查询成功，则 mysql_query() 返回结果集标识符。这个返回值在许多方法中是有用的（但不是作为行计数！）。结果集标识符可用于下列目标：

- 将它传递给 mysql_num_rows()，来确定结果集中的行数。
- 将它传递给 mysql_num_fields()，来确定结果集中的列数。
- 将它传递给提取行的例程，来提取结果集的连续行。这个实例使用了 mysql_fetch_

row()，但是还有其他的选择，我们马上就会看到它们。

- 将它传递给 mysql_free_result()，来释放结果集并允许 PHP 处理一些与之相关的源文件。

在 mysql_query() 成功地执行 SELECT查询之后（请参见表8-1），PHP 为检索结果集提供了几个提取行的函数。当不再有行时，每个函数都得到一个结果集标识符作为参数并返回 FALSE。

表8-1 PHP 行提取函数

函 数 名	返 回 值
mysql_fetch_row()	一个数组，由数字索引访问其元素
mysql_fetch_array()	一个数组，由数字索引或相关索引访问其元素
mysql_fetch_object()	一个对象，作为属性访问其元素

最基本的调用是 mysql_fetch_row()，它返回结果集的下一行作为一个数组。数组的元素通过从 0 到 mysql_num_fields()－1 范围内的数字索引访问。下面的实例说明了如何在每一行都提取和打印值的简单循环中使用 mysql_fetch_row()：

```
$query = "SELECT * FROM president";
$result = mysql_query ($query)
    or die ("Query failed");
while ($row = mysql_fetch_row ($result))
{
    for ($i = 0; $i < mysql_num_fields ($result); $i++)
    {
        if ($i > 0)
            print ("\t");
        print ($row[$i]);
    }
    print ("\n");
}
```

变量 $row 是一个数组。可用 $row[$i] 访问它，$i在这里是数字列索引。如果熟悉 PHP count() 函数，可以试着用它而不要用 mysql_num_fields() 来确定每一行的列数。count() 只计算这个数组中已设置值的元素的数量，PHP 不是与 NULL 列值相对应的元素设置值。count() 对返回列数的度量是不可靠的，因为那不是它想要的。它还用于另外两种提取行的函数。

第二个提取行的函数 mysql_fetch_array() 在表8-1中列出，它与 mysql_fetch_row() 相类似，但是由数字索引和相关索引返回的数组元素都是可靠的。换句话说，可以通过数字或名称访问元素：

```
$query = "SELECT last_name, first_name FROM president";
$result = mysql_query ($query)
    or die ("Query failed");
while ($row = mysql_fetch_array ($result))
{
    printf ("%s %s\n", $row[0], $row[1]);
    printf ("%s %s\n", $row["first_name"], $row["last_name"]);
}
```

由 mysql_fetch_array() 返回的信息是 mysql_fetch_row() 返回的信息的扩展集。除此之外，两个函数之间的不同性能是可以忽略不记的，调用 mysql_fetch_array()可以无特殊性能损耗。

第三个提取行的函数 mysql_fetch_object()，返回结果集的下一行作为对象，这意味着用

$row->col_name 语法访问行的元素。例如，如果从 President 表中检索 last_name 和 first_name 值，可以用 $row->last_name 和 $row->first_name 访问那些列：

```
$query = "SELECT last_name, first_name FROM president";
$result = mysql_query ($query)
    or die ("Query failed");
while ($row = mysql_fetch_object ($result))
    printf ("%s %s\n", $row->first_name, $row->last_name);
```

在查询结果中测试 NULL 值

可以使用 isset() 函数测试 SELECT 查询返回的列值是否为 NULL。例如，如果行包含在 $row 数组中，那么如果 $row[$i] 对应于 NULL 值，则 isset($row[$i]) 就为 FALSE，如果 $row[$i] 为非 NULL 值，则 isset($row[$i]) 就为 TRUE。相关的函数是 empty()，但是对于 NULL 和空字符串，empty() 返回的结果都是一样的，因此作为 NULL 值测试而言，这个函数是无用的。

如果查询包括计算的列怎么办？例如，发布一个作为表达式结果计算的返回值的查询：

```
SELECT CONCAT(first_name, " ", last_name) FROM president
```

这样编写的查询不适于使用 mysql_fetch_object()。选择的列名本身就是表达式，它不是合法的属性名。然而，可以通过给列赋予一个别名来提供合法的名称。下面的查询将列的别名命名为 full_name，如果用 mysql_fetch_object() 提取结果，就允许它以 $row->full_name 形式来访问：

```
SELECT CONCAT(first_name, " ", last_name) AS full_name FROM president
```

8.2.4 处理错误

PHP提供了三种处理错误的方法：

■ **用‘ @ ’取消错误消息。** 可以对一些显示消息的函数使用‘ @ ’。当我们调用 mysql_pconnect() 阻止来自于函数的错误消息不在发送到客户机的页面上出现时，就已经在做这一点了。

■ **使用 error_reporting() 函数。** 这个函数按下列级别将错误报告打开或者关闭：

错误级别	错误报告类型
1	正常函数错误
2	正常警告
4	分析程序错误
8	请注意

为了控制错误报告，可调用 error_reporting() 函数，且参数为想要激活的级别的总和。关闭级别1和级别2警告应该完全能够取消来源于 MySQL 的消息：

```
error_reporting (4 + 8);
```

你可能不想关闭有关分析错误的级别 4警告，如果关掉了，可能要有一段艰难的时间用来调试它对脚本造成的改动！级别 8警告常常被忽略，但有时它指出脚本中应该注意的问题，因此您可能也想把它激活。还有 16和32错误级别，它们都来自于 PHP 核心发动机，而非函数，因此通常情况下不必考虑它们。

■ **使用 mysql_error() 和 mysql_errno()。** 这些函数报告了 MySQL 服务器返回的错误信息。它们与相应的 C API 调用相类似。mysql_error() 以字符串形式返回错误信息（如

果不发生错误就返回空字符串)。mysql_errno() 返回一个错误数字（如果不发生错误就返回0)。两个函数都有指定与 MySQL 服务器连接的连接标识符参数，在返回状态的连接上都返回最近调用的 MySQL 函数的错误信息。连接标识符是可选的，如果缺失，就使用最近打开的连接。例如，可以这样报告 mysql_query() 的错误：

```
vay:
  if (!($result = mysql_query ( ... )))
  {
      print ("errno: " . mysql_errno());
      print ("error: " . mysql_error());
  }
```

mysql_error() 和 mysql_errno() 的 PHP 版本在一个重要方面与 C API 中对应的部分不同。在 C 中，即使试图连接服务器失败，也会得到错误信息。相反， PHP 调用直到连接建立成功，才返回有用的连接信息。换句话说，如果连接失败，就不能用 mysql_error() 和 mysql_errno() 报告失败原因。如果要报告连接失败的特殊原因而不是普通原因，则必须做特殊的考虑。请参阅附录 H “ PHP API 参考 ”，其中详细介绍了如何去做。

当检测到错误时，本章的脚本打印了相当普通的错误信息，如 “ 查询失败 ”。然而在开发脚本时，您会发现：加入一个 mysql_error() 调用对帮您发现错误的特殊原因是很有用的。

8.2.5 引用问题

在 PHP 中构造查询字符串时，知道引用问题是必要的，就像在 C 和 Perl 中一样。虽然函数名在各种语言中是不同的，但处理引用问题的方法却是类似的。假设正在构造一个，将新的记录插入到表中的查询，可以在值的前后加上引号插入到字符串列中：

```
$last = "O'Malley";
$first = "Brian";
$expiration = "2002-9-1";
$query = "INSERT member (last_name,first_name,expiration)"
       . " VALUES('$last','$first','$expiration')"
```

这里的问题是引用值的本身还包含着引号（“ O ’ Malley ”），如果将查询发送到 MySQL 服务器会导致语法错误。在 C 中我们调用 mysql_escape_string() 解决这个问题。在 Perl DBI 脚本中则使用 quote( )。PHP 有一个 addslashes( ) 函数可以完成同样的事情。例如，调用 addslashes（“ O’ Malley ”）返回 “ O’ Malley ” 值。将前面的实例做如下编写来解决引用问题：

```
$last = addslashes ("O'Malley");
$first = addslashes ("Brian");
$expiration = addslashes ("2002-9-1");
$query = "INSERT member (last_name,first_name,expiration)"
       . " VALUES('$last','$first','$expiration')"
```

DBI quote( ) 方法是把前后的引号加到字符串中。 addslashes( ) 则不是，因此我们仍需在查询字符串中要插入值的周围将那些引号显式地指定出来。

当编写信息出现在 Web 页面上时也将发生引用问题。如果正在编写一个将作为 HTML 或 URL的部分出现的字符串，而且这个字符串包含 HTML 或 URL 内部的特殊字符，最好将它编码。 PHP 函数 htmlspecialchars( ) 和 urlencode( ) 可以做到这点，它们与 CGI.pm escapeHTML( ) 和 escape( ) 方法相类似。

8.3 运行 PHP

在本章的剩余部分中，将解决我们在第 1章中为自己定下的目标：

- 对于学分保存方案，需要编写一个允许我们输入、编辑测试和测验分数的脚本。
- 对于历史同盟，需要开发一个有关美国总统的联机测验，使之成为交互式的，以便不做任何事情就可以为这个 Web 站点的访问者产生试题。
- 我们也想允许历史同盟成员联机编辑它们的目录项，使信息维持最新并减少我们自己编辑项的数量。

每个脚本都产生多个 Web 页面，并依靠在创建页面中嵌入的信息来在脚本的调用之间进行通信。

8.3.1 输入学生分数

在本节中，我们将把注意力转向学分保存方案。我们 Web 站点上的这个区域的 URL 是 http://pit-viper.snake.net/gp/，应该为它编写一个简短的主页 index.php，下面的页面就正在做这件事。它包括了与第 7 章编写的 score_browser 脚本的连接，因为这个脚本适合于学分保存方案。

```
<?php
include ("samp_db.inc");

$title = "Grade-Keeping Project";
html_begin ($title, $title);
?>

<P>
<A HREF="/cgi-bin/score_browse">View</A> scores for quizzes and tests

<?php
html_end ();
?>
```

现在让我们考虑如何设计和实现脚本 score_entry.php，它将让我们输入一组新的测试或测验分数，或者修改一组已经存在的分数。后者的性能对于处理由于生病或者其他原因缺席（或者，放弃这个想法以免输入分数失败）造成考试或测验比其他学生晚的学生的分数是必要的。分数项脚本的概要是这样的：

1) 最初的页面代表一系列已知的登记事件，并允许选择一个事件或者指定应该创建的新事件。

2) 如果选择创建一个新事件，脚本就给出允许指定日期和事件类型的页面。创建这个事件记录之后，脚本重新显示事件列表页面来显示这个新事件。

3) 当选择了事件后，脚本给出在顶部（事件 ID、日期、类型）显示事件信息的分数项页面，后接每个学生一项的列表。对于新事件，项将是空白的。对于已存在的事件，项将显示每个学生已存在的分数。选择提交按钮时，分数输入到 score 表中。

脚本需要执行几个不同的操作，这意味着我们需要从一个页面到另一个页面周而复始地传递状态变量，以便脚本在每次调用时能够知道假设要做什么。在 PHP 中很容易做到这一点，因为 PHP 处理作为 URL 参数传递的变量，并把它们转换为与参数具有相同名称的变量。例如，可以在脚本 URL 的末尾对参数 action 进行如下编码：

```
http://pit-viper.snake.net/gp/score_entry.php?action=value
```

当调用 score_entry.php 时，参数 action 作为变量 $action 来编码，这样就可以直接访问它了。这也适用于格式中的域。设想一个包括域 name 和 address 的表格，当客户机传递表格时，Web 服务器就调用脚本访问表格的内容。脚本能够找出通过检查变量 $name 和 $address 的值而输入到表格中的值。对于包括许多域的表格，全部给出唯一的命名是有困难的。PHP 很容易地把数组在表格中传入和传出。如果使用了如 x[0]、x[1] 等等的域名，则 PHP 把它们作为 $x 数组的元素进行编码。可以将这些元素作为 $x[0]、$x[1] 等等来访问。

我们通过使用页面中的 action 参数，可以将信息从 score_entry.php 脚本的一个调用传送到另一个调用，并在脚本中用变量 $action 检查它的值。脚本的框架是这样的：

```
<?php
# score_entry.php - Score Entry script for grade-keeping project

include ("samp_db.inc");

# define action constants
define (INITIAL_PAGE, 0);
define (SOLICIT_EVENT, 1);
define (ADD_EVENT, 2);
define (DISPLAY_SCORES, 3);
define (ENTER_SCORES, 4);

/* ... put functions here ... */

$title = "Grade-Keeping Project — Score Entry";
html_begin ($title, $title);

samp_db_connect()
    or die ("Cannot connect to database server");

if (empty ($action))
    $action = INITIAL_PAGE;

switch ($action)        # what are we supposed to do?
{
case INITIAL_PAGE:      # present initial page
    display_events ();
    break;
case SOLICIT_EVENT:     # ask for new event information
    solicit_event_info ();
    break;
case ADD_EVENT:         # add new event to database
    add_new_event ();
    display_events ();
    break;
case DISPLAY_SCORES:    # display scores for selected event
    display_scores ();
    break;
case ENTER_SCORES:      # enter new or edited scores
    enter_scores ();
    display_events ();
    break;
default:
    die ("Unknown action code ($action)");
}
```

```
html_end ();
?>
```

变量 $action 可以取若干值，我们已在 switch() 语句中测试过了（为避免在脚本中使用文字的数字，可以用 PHP 的 define() 构造来定义常量）。PHP switch() 语句与它在 C 中相应的部分相类似。在 score_entry.php 中，它用来确定采用什么操作，并且调用实现这个操作的函数。

检查一下每次处理一个操作的函数。第一个函数 display_events()，检索来自 MySQL 的 event 表的行并加以显示。表的每一行都列出了事件 ID、日期和时间类型（测试或测验），还有编写事件 ID作为可以选择用来修改事件分数的连接：

```
function display_events ()
{
global $PHP_SELF;

    print ("Select an event by clicking on its number,\n");
    print ("or select New Event to create a new grade event:<BR><BR>\n");
    $query = "SELECT event_id, date, type"
            . " FROM event"
            . " ORDER BY event_id";
    $result = mysql_query ($query)
        or die ("Cannot execute query");
    print ("<TABLE BORDER>\n");
    print ("<TR>\n");
    display_cell ("TH", "Event ID", 1);
    display_cell ("TH", "Date", 1);
    display_cell ("TH", "Type", 1);
    print ("</TR>\n");

    # associate each event id with a link that will show the
    # scores for the event; use mysql_fetch_array() so we
    # can refer to columns by name.
    while ($row = mysql_fetch_array ($result))
    {
        print ("<TR>\n");
        $url = sprintf ("%s?action=%d&event_id=%d",
                        $PHP_SELF, DISPLAY_SCORES, $row["event_id"]);
        display_cell ("TD",
                    "<A HREF=\"$url\">" . $row["event_id"] . "</A>",
                    0);
        display_cell ("TD", $row["date"], 1);
        display_cell ("TD", $row["type"], 1);
        print ("</TR>\n");
    }
    # add one more link for "new event"
    print ("<TR ALIGN=CENTER>\n");
    $url = sprintf ("%s?action=%d", $PHP_SELF, SOLICIT_EVENT);
    display_cell ("TD COLSPAN=3",
                    "<A HREF=\"$url\">" . "New Event" . "</A>",
                    0);
    print ("</TR>\n");

    print ("</TABLE>\n");
}
```

表中的连接用 $PHP_SELF 来构造。这个变量包括了脚本自己的 URL，它为脚本再次调用自己提供了一个方便的方法。然而，请注意函数开始处的 global 行:

```
global $PHP_SELF;
```

在 PHP 函数中，全局变量是不可访问的，除非显式地声明要使用它们。没有 global 行，$PHP_SELF 将被看成局部变量（因为我们没有将值赋给它，因此是空的）。在函数内部，使用 global 来访问依靠 URL 参数或者作为表格域传递到脚本中的参数也是必需的。

用来生成表的函数 display_cell() 与第 7 章编写的同名 DBI 函数相类似。PHP 版本如下：

```
function display_cell ($tag, $value, $encode)
{
    if ($encode)
        $value = htmlspecialchars ($value);
    if ($value == "")
        $value = " ";
    print ("<$tag>$value</$tag>\n");
}
```

如果在 display_events() 给出的表中选择了“ New Event ”连接，则脚本通过操作 SOLICIT_ EVENT进行再次调用。它引发了对 solicit_event_info() 的调用，这个函数显示了允许输入新事件信息的表格：

```
function solicit_event_info ()
{
global $PHP_SELF;

   printf ("<FORM METHOD=\"post\" ACTION=\"%s?action=%d\">\n",
                                          $PHP_SELF, ADD_EVENT);
   print ("Enter information for new grade event:<BR><BR>\n");
   print ("Date: <INPUT TYPE=text NAME=\"date\"");
   print (" VALUE=\"\" SIZE=10> ");
   print ("Type: ");
   print ("<INPUT TYPE=\"radio\" NAME=\"type\" VALUE=\"T\" CHECKED>Test\n");
   print ("<INPUT TYPE=\"radio\" NAME=\"type\" VALUE=\"Q\">Quiz\n");
   print ("<BR><BR>\n");
   print ("<INPUT TYPE=\"submit\" NAME=\"button\" VALUE=\"Submit\">\n");
   print ("</FORM>\n");
}
```

由 solocit_event_info() 生成的表格包括输入数据的编辑域、指定新事件是测试还是测验的两个单选按钮、 Submit 按钮。当递交表格时， ADD_EVENT 操作将调用 score_entry.php。调用 add_new_event() 函数在 event 表中输入一个新的行：

```
function add_new_event ()
{
global $date, $type;

    if (empty ($date))  # make sure a date was entered
        die ("No date specified");
    $query = sprintf ("INSERT INTO event (date,type) VALUES(\"%s\",\"%s\")",
                    addslashes ($date), $addslashes ($type));
    if (!mysql_query ($query))
        die ("Could not add event");
}
```

在 add_new_event() 中，我们使用 global 访问在新事件项表格中使用的域值（ date 和 type，用变量 $date 和 $type 访问）。做出最低限度的安全检查，确定数据为非空白之后，在 event 表中输入一个新记录。输入这个事件记录之后，主程序将再次显示事件列表，这样就可以选择新事件并开始输入分数了。

函数 display_scores() 为给定的事件查找已存在的分数，并列出显示他们的表格，包括学生姓名：

```
function display_scores ()
{
global $PHP_SELF, $event_id;

    # select scores for the given event
    $query = "
        SELECT
            student.student_id, student.name, event.date,
            score.score AS score, event.type
        FROM student, event
            LEFT JOIN score ON student.student_id = score.student_id
                    AND event.event id = score.event id
       WHERE event.event_id = $event_id
       ORDER BY student.name
    ";
    $result = mysql_query ($query)
        or die ("Cannot execute query");

    printf ("<FORM METHOD=\"post\" ACTION=\"%s?action=%d&event_id=%d\">\n",
                                $PHP_SELF, ENTER_SCORES, $event_id);

    # print scores in a table, and print the event date and type
    # preceding the table. (however, we cannot print the date and
    # type until we've fetched the first row of the result set)

    $needheading = 1;
    while ($row = mysql_fetch_array ($result))
    {
        if ($needheading)
        {
            printf ("Event ID: %s, Event date: %s, Event type: %s\n",
                                $event_id, $row["date"], $row["type"]);
            print ("<BR><BR>\n");
            print ("<TABLE BORDER>\n");
            print ("<TR>\n");
            display_cell ("TH", "Name", 1);
            display_cell ("TH", "Score", 1);
            print "</TR>\n";
            $needheading = 0;
        }
        print ("<TR>\n");
        display_cell ("TD", $row["name"], 1);
        $col_val =
            sprintf ("<INPUT TYPE=text NAME=\"score[%s]\"",
                                    $row["student_id"]);
        $col_val .=
            sprintf (" VALUE=\"%s\" SIZE=5><BR>\n",
                                    $row["score"]);
        display_cell ("TD", $col_val, 0);
        print ("</TR>\n");
    }

    print ("</TABLE>\n");
    print ("<BR>\n");
    print ("<INPUT TYPE=\"submit\" NAME=\"button\" VALUE=\"Submit\">\n");
    print "</FORM>\n";
}
```

display_scores() 用于检索所选事件的分数信息的查询并不是表之间的简单连接，因为它不会为事件中没有分数的学生选择行。特别是，对于新的事件，连接会选择无记录，这就有了一个空项表格！我们使用 LEFT JOIN 强迫为每个学生检索行，无论学生是否在 score表中已经有了分数。与 display_scores() 用来检索来自于 MySQL 的分数记录相类似的查询背景，已在3.8.2节“检查表中未给出的值”中给出了介绍。那里的查询只选择缺失分数，这里的查询只选择特殊事件的分数。

分数在表格中使用了有名称的域进行编码，如 score[*n*]，这里的*n*是 student_id 的值。当表格送回 Web 服务器时，PHP 将这些域转换为 $score 数组的元素，我们可以访问数组元素以恢复表格的内容。

当完成输入或者编辑分数，并提交给表格后，ENTER_SCORES 操作调用 score_entry.php，并且调用函数 enter_scores() 处理表格信息：

```
function enter_scores ()
{
global $score, $event_id;

    $invalid = 0;
    $blank = 0;
    $nonblank = 0;
    while (list ($student_id, $newscore) = each ($score))
    {
        $newscore = trim ($newscore);
        if (empty ($newscore))  # no score, delete if present in table
        {
            ++$blank;
            $query = "DELETE FROM score"
                . " WHERE event_id = $event_id"
                . " AND student_id = $student_id";
        }
        else if (ereg ("^[0-9]+$", $newscore))  # must be integer
        {
            ++$nonblank;
            $query = "REPLACE INTO score (event_id,student_id,score)"
                . " VALUES($event_id,$student_id,$newscore)";
        }
        else
        {
            ++$invalid;
            continue;
        }
        if (!mysql_query ($query))
            die ("score entry failed, event_id $event_id, student_id
$student_id");
    }
    printf ("Scores entered: %d<BR>\n", $nonblank);
    printf ("Scores missing: %d<BR>\n", $blank);
    printf ("Invalid scores: %d<BR>\n", $invalid);
    print ("<BR>\n");
}
```

学生 ID 的值和相关的分数通过迭代 PHP 的 each 函数的 $score 数组来获得。每个分数处理如下：

- 如果分数是空白的，则表明什么也没有输入，但是我们还要试图删除这个分数，以免它以前曾经存在（也许以前我们为缺席的学生错误地输入了分数）。

- 如果分数不是空白的，就对值进行一些根本的确认。用函数 trim() 去掉前后的空格之后，如果剩余部分是空白或者整数，就接受这个结果。然而，表格值通常作为字符串来编码，因此不能用 is_long() 或者 is_int() 检查值是否为整数。即使值只包括数字，这些函数也会返回 FALSE。既然这样，最好用模型匹配操作。如果字符串从开始到结束每个字符都是数字，则下面的测试为 TRUE：

```
ereg ("^[0-9]+$", $str)
```

如果分数检查完毕，我们就将它加到 score 表中。查询使用 REPLACE 而不用 INSERT，因为我们可能替换了已存在的分数而不是输入一个新的分数（ REPLACE 在两种情况下都适用）。

注意 score_entry.php 脚本。现在所有的分数项和编辑项都能从 Web 浏览器执行。一个明显的缺点是：脚本没有提供安全措施，连接到 Web 服务器的任何人都可以对分数进行编辑。以后，我们用编辑历史同盟成员项编写的脚本来说明这个脚本所采取的简单确认方案。也可以使用 PHPLIB 程序包来提供更完善的确认。

8.3.2 美国总统测验

历史同盟 Web 站点的目标之一就是用它给出测验的在线版本，这类似于同盟在时事通信“美国编年史”的儿童部分发表的一些测验。实际上我们创建了 president 表，因此对基于历史的测验可以用它作为问题的来源。为了给出这个测验，我们将编写称为 pres_quiz.php 的脚本。

基本的想法是随机挑选一个总统，问一个关于他的问题，然后请求用户回答并且察看答案是否正确。为了简单一点，可以把主题限制为询问总统出生在哪里。另外一种简单的衡量就是以多个选择的格式给出这个问题。这对用户来讲很容易，他只需从一组选择中挑选一个，而不用将之键入等待回应。这对我们来讲也是容易的，因为我们不需做任何棘手的匹配字符串来检查用户可能键入的内容，而只需对用户的选择和我们寻找的值做一个简单的比较。

显示这个测验的脚本必须执行两个函数。第一个，对于它最初的调用，将从 president表中查阅信息来生成并显示一个新的问题。第二个，如果脚本已经被调用是因为用户正提交一个回答，那么就需要检查这些答案并给出一些反馈信息来指出它是否正确。如果正确，脚本会生成并显示一个新的问题。如果回答不正确，将再次显示同一问题。

为了生成这些问题，我们将使用 MySQL 3.23 中出现的一个 ORDER BY RAND()特性。使用这个函数就能从 president表中随机地进行行选择。例如，为了随机地挑选总统的姓名和出生地，查询将执行这样的操作：

```
SELECT CONCAT(first_name, ' ', last_name) AS name,
    CONCAT(city, ', ', state) AS place
    FROM president ORDER BY RAND() LIMIT 1
```

name是选择的总统的名字，出生地是问题“总统出生在哪里？”的正确答案，我们还需要给出一些错误的选择，可用类似的查询：

```
SELECT DISTINCT CONCAT(city, ', ', state) AS place
   FROM president ORDER BY RAND()
```

从这个查询的结果中，我们选择了与正确答案不同的最前面的四个值。发布这个查询并检索结果的函数如下：

```
function setup_quiz ()
{
    # issue query to pick a president and get birthplace
    $query = "SELECT CONCAT(first_name, ' ', last_name) AS name,"
            . " CONCAT(city, ', ', state) AS place"
            . " FROM president ORDER BY RAND() LIMIT 1";
    $result = mysql_query ($query)
        or die ("Cannot execute query");
    $row = mysql_fetch_array ($result)
        or die ("Cannot fetch result");
    $name = $row["name"];
    $place = $row["place"];

    # Construct the set of birthplace choices to present.
    # Set up the $choice array containing five birthplaces, one
    # of which is the correct response.
    $query = "SELECT DISTINCT CONCAT(city, ', ', state) AS place"
            . " FROM president ORDER BY RAND()";
    $result = mysql_query ($query)
        or die ("Cannot execute query");
    $choice[] = $place; # initialize array with correct choice
    while (count ($choice) < 5 && $row = mysql_fetch_array ($result))
    {
        if ($row["place"] == $place)
            continue;
        $choice[] = $row["place"];  # add another choice
    }
    # randomize choices, then display form
    shuffle ($choice);
    display_form ($name, $place, $choice);
}
```

为了给出测验问题的信息，我们使用了显示总统姓名、一组列出可能选择的单选按钮和一个Submit按钮的表格。这个表格需要做两件事情：必须对客户机给出测验信息；当用户提交回答时必须将信息传送回 Web 服务器，以便检查回答是否正确。

为了安排表格执行这些操作，我们使用了隐藏域把测验信息包括在表格中。把域称为 name、place 和 choice，它们代表总统的姓名、出生地和一组可能的选择。使用 implode() 连接值和特殊字符，这样，这些选择可以很容易地作为单个字符串来编码（我们需要特殊字符，以便如果需要重新显示问题时可以用 explode() 分离字符串）。显示表格的函数如下：

```
function display_form ($name, $place, $choice)
{
global $PHP_SELF;

    printf ("<FORM METHOD=\"post\" ACTION=\"%s\">\n", $PHP_SELF);
    hidden_field ("name", $name);
    hidden_field ("place", $place);
    hidden_field ("choice", implode ("#", $choice));
    printf ("Where was %s born?<BR><BR>\n", $name);
    for ($i = 0; $i < 5; $i++)
    {
        print ("<INPUT TYPE=\"radio\" NAME=\"response\"\n");
        printf ("VALUE=\"%s\">%s<BR>\n", $choice[$i], $choice [$i]);
    }
    print ("<BR><INPUT TYPE=\"Submit\" VALUE=\"Submit\">\n");
    print ("</FORM>\n");
}
```

函数hidden_field( )为表格中的隐藏域编写了HTMT:

```
function hidden_field ($name, $value)
{
    printf ("<INPUT TYPE=\"HIDDEN\" NAME=\"%s\" VALUE=\"%s\">\n",
                                            $name, $value);
}
```

当用户做出选择并提交表格时，答案作为 response 域值在发送回 Web 服务器的表格内容中编码。我们可以通过检查变量 $name、$place 和 $choice 发现 name、place 和 choice 域的值。这也给了我们一个方法，指出是否是第一次调用脚本，或者用户是否给以前显示过的表格提交了回答，如果是第一次调用则不会设置那些变量。这样，通过检查其中一个变量，脚本的主体就决定了应该做的事情：

```
if (!$name)         # called for first time
    setup_quiz ();
else                # user submitted response to form
    check_response ();
```

我们仍然需要编写 check_response() 函数来将用户的回答与正确答案做比较。我们将正确答案在表格的 place 域进行编码，用户的回答则在表格的 response 域进行编码，因此我们所要做的就是比较 $place 和 $response。在比较结果的基础上，我们提供了一些反馈信息，之后每次都生成显示一个新的问题，或者再次显示相同的问题：

```
function check_response ()
{
global $name, $place, $choice;
global $response;

    if ($place == $response) # correct response; generate new question
    {
        print ("That is Correct!<BR>\n");
        printf ("%s was born in %s.<BR>\n", $name, $place);
        print ("Try the next question:<BR><BR>\n");
        setup_quiz();
    }
    else                    # incorrect response; redisplay question
    {
        print ("That is not correct.  Please try again.<BR><BR>\n");
        $choice = explode ("#", $choice);
        display_form ($name, $place, $choice);
    }
}
```

这样，我们就做完了。将 pres_quiz.php 的链接加到历史同盟主页上，参观者可以做一下这个测验来测试他们的知识。

8.3.3 历史同盟联机成员项的编辑

最终的脚本 edit_member.php 允许历史同盟成员编辑他们自己的联机项。无论何时，成员都可以校正或者更新他们的成员信息，而不必向同盟部提交这些更改。这个性能使成员目录总是保持最新的，而且减少了秘书的工作量。

我们需要采取的一个防范措施就是：除了该项目的成员之外，防止任何其他人修改项目。这意味着我们需要一些安全性的表单。作为一个简单的身份确认表单的示范，我们将使用 MySQL 存放每个成员的口令，并要求成员提供正确的口令以访问脚本给出的编辑表单。该脚

本操作如下：

- 当初次调用时，edit_script.php 给出包括成员 ID 和口令域的表单。
- 当提交初始表单时，脚本用成员 ID 作为关键字寻找相关的口令来搜索口令表。如果口令相符，脚本将从 member 表中查找成员项，并显示要编辑的内容。
- 当提交编辑过的表单后，我们就用表单的内容更新项。

edit_member.php 的框架如下所示：

```
<?php
# edit_member.php - Edit Historical League member entries via the Web

include ("samp_db.inc");

# define action constants
define (INITIAL_PAGE, 0);
define (DISPLAY_ENTRY, 1);
define (UPDATE_ENTRY, 2);

/* ... put functions here ... */

if (empty ($action))
    $action = INITIAL_PAGE;

$title = "US Historical League — Member Editing Form";
html_begin ($title, $title);

samp_db_connect ()
    or die ("Cannot connect to server");

switch ($action)
{
case INITIAL_PAGE:
    solicit_member_id ();
    break;
case DISPLAY_ENTRY:
    display_entry ();
    break;
case UPDATE_ENTRY:
    update_entry ();
    break;
default:
    die ("Unknown action code ($action)");
}

html_end ();
?>
```

初始页面显示了请求成员 ID 和口令的表单：

```
function solicit_member_id ()
{
global $PHP_SELF;

    printf ("<FORM METHOD=\"post\" ACTION=\"%s?action=%d\">\n",
                                        $PHP_SELF, DISPLAY_ENTRY);
    print ("Enter your membership ID number and password,\n");
    print ("then select Submit.\n<BR><BR>\n");
    print ("<TABLE>\n");
    print ("<TR>");
    print ("<TD>Member ID</TD><TD>");
    print ("<INPUT TYPE=text NAME=\"member_id\" SIZE=10><BR>\n");
```

```
    print ("</TD></TR>");
    print ("<TR>");
    print ("<TD>Password</TD><TD>");
    print ("<INPUT TYPE=password NAME=\"password\" SIZE=10><BR>\n");
    print ("</TD></TR>");
    print ("</TABLE>\n");
    print ("<INPUT TYPE=\"submit\" NAME=\"button\" VALUE=\"Submit\">\n");
    print "</FORM>\n";
}
```

当然，我们还需要一些口令，一个简单的方法就是随机地生成它们。下面的语句建立member_pass表，然后，通过从随机数中生成 MD5 校验来为每个成员创建口令。您可以让成员们来选择他们自己的口令，也可以调用 mysql 并发布这些语句作为一种既快速又容易的方法：

```
CREATE TABLE member_pass
(
    member_id INT UNSIGNED NOT NULL PRIMARY KEY,
    password CHAR(8)
)
INSERT INTO member_pass (member_id, password)
    SELECT
        member_id,
        LEFT(MD5(RAND()), 8) AS password
    FROM member
```

我们将一个特殊项加到这个表中作为编号 0，它有一个用于管理的（超级用户）口令。可以使用这个口令访问所有想要访问的项：

```
INSERT INTO member_pass (member_id, password) VALUES(0, "secret");
```

在创建口令表之后，您可以停止使用第 7章中编写的 samp_browse 脚本，该脚本允许任何人在 samp_db 数据库中浏览任何表的内容，其中包括 member_pass 表。

当成员输入 ID 和口令并提交该表单时，edit_member.php 显示该编辑的项：

```
function display_entry ()
{
global $PHP_SELF;
global $member_id, $password;

    $member_id = trim ($member_id);
    if (empty ($member_id))
        die ("No member ID specified");
    if (!ereg ("^[0-9]+$", $member_id)) # must be integer
        die ("Invalid member ID specified (must be number)");
    if (empty ($password))
        die ("No password specified");
    if (check_pass ($member_id, $password)) # regular member
        $admin = 0;
    else if (check_pass (0, $password))     # administrator
        $admin = 1;
    else
        die ("Invalid password");

    $query = "SELECT last_name, first_name, suffix, email,"
        . "street, city, state, zip, phone, interests,"
        . "member_id, expiration"
        . " FROM member"
        . " WHERE member_id = $member_id"
        . " ORDER BY last_name";
    $result = mysql_query ($query)
        or die ("Cannot execute query");
```

```
if (mysql_num_rows ($result) == 0)
    die ("No user with member_id = $member_id found");
if (mysql_num_rows ($result) > 1)
    die ("More than one user with member_id = $member_id found");

printf ("<FORM METHOD=\"post\" ACTION=\"%s?action=%d\">\n",
                                    $PHP_SELF, UPDATE_ENTRY);
hidden_field ("member_id", $member_id);
hidden_field ("password", $password);
print ("<TABLE>\n");
# read results of query and format for editing
$row = mysql_fetch_array ($result);
display_column ("Member ID", $row, "member_id", 0);
# allow user with admin password to edit expiration
display_column ("Expiration", $row, "expiration", $admin);
display_column ("Last name", $row, "last_name", 1);
display_column ("First name", $row, "first_name", 1);
display_column ("Suffix", $row, "suffix", 1);
display_column ("Email", $row, "email", 1);
display_column ("Street", $row, "street", 1);
display_column ("City", $row, "city", 1);
display_column ("State", $row, "state", 1);
display_column ("Zip", $row, "zip", 1);
display_column ("Phone", $row, "phone", 1);
display_column ("Interests", $row, "interests", 1);
print ("</TABLE>\n");
print ("<INPUT TYPE=\"submit\" NAME=\"button\" VALUE=\"Submit\">\n");
print "</FORM>\n";
```

display_entry() 需要做的第一件事就是校验口令。对于给定的成员 ID，如果表单中输入的口令与 member_pass 表中存放的口令相符，或者如果它与管理口令相符（即成员 0 的口令），edit_member.php 就显示编辑的项。口令检查函数 check_pass() 将执行一个简单的查询从 member_pass 表中移出一条记录：

```
function check_pass ($id, $pass)
{
    $query = "SELECT password FROM member_pass WHERE member_id = $id";
    if (!($result = mysql_query ($query)))
        die ("Error reading password table");
    if (!($row = mysql_fetch_array ($result)))
        return (FALSE);
    return ($row["password"] == $pass); # TRUE if password matches
}
```

因为不能修改它，所以编辑表单作为只读文本显示成员 ID 的值。对于正常的成员，截止日期也作为只读文本显示，因为不能让成员改动它。然而，如果给出管理口令，则截止日期就成为可编辑的，允许同盟秘书为成员更新日期来重新更新他们的会员资格。

member 表项的列由 display_column() 函数显示。它按照第三个参数值把列作为可编辑的文本或作为只读文本加到编辑表单中：

```
# Display a column of a member entry.  $label is the visible label
# displayed to the user.  $row is the array containing the entry.
# $col_name is the name of a column in the entry.  $value is the
# column value.  $editable is non-zero if the user is allowed to
# change the value.  The value is displayed as non-editable text
# otherwise.  Field names are constructed as row[col_name] so that
# when the form is submitted, values can be accessed using an array
# rather than a bunch of individual variables.
```

```
function display_column ($label, $row, $col_name, $editable)
{
    print ("<TR>\n");
    printf ("<TD>%s</TD>\n", htmlspecialchars ($label));
    $value = htmlspecialchars ($row[$col_name]);
    if ($editable)  # display as edit field
    {
        $str = sprintf ("<INPUT TYPE=text NAME=\"row[%s]\"", $col_name);
        $str .= sprintf (" VALUE=\"%s\" SIZE=\"80\">\n", $value);
    }
    else            # display as read-only text
        $str = $value;
    printf ("<TD>%s</TD>\n", $str);
    print ("</TR>\n");
}
```

display_entry() 函数在格式中作为隐藏字段嵌入了 member_id 和 password，因此当成员提交编辑的项时将继续 edit_script.php 的下一个调用。这允许自动校验 ID 的口令，而不用请求成员再次输入（请注意，我们的简单的确认身份的方法是以文本形式来回传递口令。通常这不是个好主意，但是历史同盟不是对安全性要求很高的运作机构，因此这种方法足够满足要求。如果在运行金融业务，可能需要更强的安全性操作）。

更新项的函数如下：

```
function update_entry ()
{
global $row, $member_id, $password;

    $member_id = trim ($member_id);
    if (empty ($member_id))
        die ("No member ID specified");
    if (!ereg ("^[0-9]+$", $member_id)) # must be integer
        die ("Invalid member ID specified (must be number)");
    if (!check_pass ($member_id, $password) && !check_pass (0, $password))
        die ("Invalid password");

    # We'll need a result set to use for assessing nullability of
    # member table columns.  This query gives us one without
    # selecting any rows.
    $result = mysql_query ("SELECT * FROM member WHERE 1 = 0");
    if (!$result)
        die ("Cannot query member table");

    # iterate through each field in the form, using the values to
    # construct the UPDATE statement.

    $query = "UPDATE member ";
    $delim = "SET "; # put "SET" before first column, "," before others
    while (list ($col_name, $val) = each ($row))
    {
        $query .= "$delim $col_name =";
        $delim = ",";
        # if a value is empty, update the value with NULL if the
        # column is nullable.  This prevents trying to put an
        # empty string in the expiration when it should be NULL,
        # for example.
        $val = trim ($val);
        if (empty ($val))
        {
```

```
            if (nullable ($result, $col_name))
                $query .= "NULL";   # enter NULL
            else
                $query .= "\"\"";   # enter empty string
        }
        else
            $query .= "\"" . addslashes ($val) . "\"";
    }
    $query .= " WHERE member_id = $member_id";
    if (mysql_query ($query) && mysql_affected_rows () > 0)
        print ("Entry updated successfully.\n");
    else
        print ("Entry not updated.\n");
}
```

首先，重新校验口令，确定没人发送假表单来愚弄我们，然后更新项。更新时需要注意，因为如果表单中的字段是空白的，则可能需要作为 NULL 而不是作为空字符串输入。expiration 列就是这样的例子。NULL的成员截止日期具有特殊的含义，即“终生会员”。如果将一个空字符串插入到此列中，值转换成“ 0000-00-00 ”，则成员不再具有终生会员资格。

为了处理这个问题，我们查找该列的元数据并检查它是作为 NULL 还是作为 NOT NULL 进行声明的。该信息由函数 mysql_fetch_field() 返回。不幸地是，此函数通过数值的索引查找列。在 member 表中按名称访问列会更方便，因此我们编写一个小函数 nullable()，它获取一个列名并查找相应的元数据对象：

```
# Determine whether or not the column with the given name in
# the result set is nullable.

function nullable ($result, $col_name)
{
    for ($i = 0; $i < mysql_num_fields ($result); $i++)
    {
        if (!($fld = mysql_fetch_field ($result, $i)))
            continue;
        if ($fld->name == $col_name)
            return (!$fld->not_null);
    }
    return (0);
}
```

mysql_fetch_field() 函数需要包含检查列所在表的结果集标识符。这可通过执行简单的不返回行的 SELECT 查询来获得。虽然该查询返回一个空结果集，但是，对于检索要评估 member 表中列的空性能（nullability）的元数据来说，这种方法足够了：

```
SELECT * FROM member WHERE 1 = 0
```

安装脚本，让成员们知道他们的口令，这样他们就能更新自己的成员信息了。