第17章　使用PHP和MySQL实现身份验证

在本章中，我们将讨论如何在用户身份验证中应用各种PHP和MySQL技术。

在本章中，我们将主要介绍以下内容：

- 识别访问者
- 实现访问控制
- 使用基本身份验证
- 在PHP中使用基本身份验证
- 在Apache的.htaccess文件中使用基本身份验证
- 使用mod_auth_mysql身份验证
- 创建自定义身份验证

17.1　识别访问者

Web是一个匿名媒体，但是通常知道谁在访问网站是非常有意义的。如果没有访问者的帮助，很难了解他们的信息，这对于访问者的隐私来讲是有利的。但是通过一些操作，服务器就可以发现许多关于连接他们的计算机与网络的信息。Web浏览器通常都可以识别它自己，它可以告诉服务器自己是什么浏览器、浏览器版本、运行当前操作系统的用户等。通过使用JavaScript，还可以确定访问者屏幕的颜色深度与分辨率，以及他们的Web浏览窗口的大小。

每一台连接到互联网的计算机有一个唯一的IP地址。根据访问者的IP地址，可以推测出他的一些信息。可以发现谁拥有该IP，有时甚至可以猜测出访问者的地理位置。一些地址比其他地址更有用。通常，拥有永久互联网连接的用户将拥有永久地址。通过电话拨号连接到ISP的用户只能使用ISP分配的一个临时IP地址。所以，当再次看到这个IP地址时，它可以正由另一台计算机使用，下次看到该访问者时，他也可能正在使用另一个IP地址。

幸运的是，对于网络用户，他们的浏览器无法泄露可以识别他们身份的信息。如果要知道访问者的名字或者其他信息，必须亲自询问他。

许多网站强制要求用户提供他们的信息。纽约时报（http://www.nytimes.com）可以免费提供报刊内容，但是只有愿意提供详细信息（例如姓名、性别和家庭总收入）的人才能免费阅读。Nerd新闻组和讨论网站Slashdot（http://www.slashdot.org）允许注册用户使用昵称参与讨论，用户还可以自定义他们希望看到的界面。大多数电子商务网站在顾客订购第一份订单时记录顾客的详细信息，这就意味着网站不要求顾客每次都输入详细信息。

如果已经从访问者那里询问和获得了信息，只需要在他下次访问的时候将这些信息与访问者联系起来。如果愿意做出这样的假设：只有一个用户通过一台机器使用特定的账号访问网站，而且每个访问者都只使用一台机器，那么可以将一个cookie存储到用户机器中，这样也可以识别该用户。

当然，这对所有用户来说是不可能的。通常，许多人可能会共享一台计算机，而且许多人还可能使用多台计算机。至少在一段时间之后，要再次询问访问者的名字。除了询问访问者的名字，可能还会要求访问者提供某种证据来证明身份。

正如我们在第15章中介绍的：要求用户证明身份的操作称为身份验证。如今，在网站上经常使用的身份验证方法是要求访问者提供一个唯一的登录名和密码。通常，身份验证可以用来允许或禁止用户对特定页面或资源进行访问，但是这是可选的，或者也可以用于其他目的，如个性化设置。

17.2　实现访问控制

实现简单的访问控制并不困难。程序清单17-1所示的代码可以将输出3个可能结果之一。

如果没有使用参数载入文件，该代码将显示一个要求输入用户名和密码的HTML表单。该表单类型如图17-1所示。

如果用户提交了这些参数，但是参数不正确，该代码将显示一个错误信息。错误信息如图17-2所示。

图17-1　HTML表单要求访问者提供访问站点所需的用户名和密码

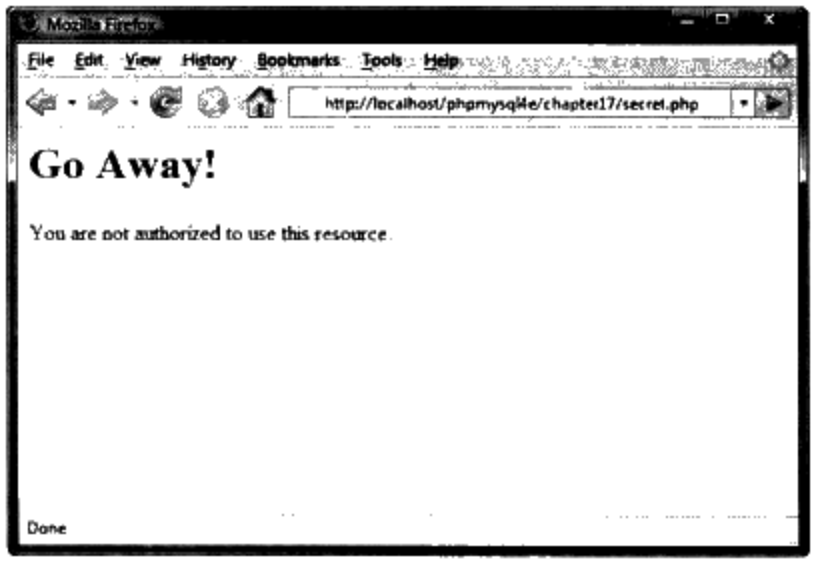

图17-2　当用户输入了不正确的信息时，将给出一个出错信息。在一个真正的网站上，可能要给出一个更友好的信息

如果用户提交了这些参数并且参数正确，它将显示该页的秘密内容。测试内容如图17-3所示。

图17-3　当用户提交正确的详细信息时，脚本将显示内容

创建如图17-1、图17-2和图17-3所示功能的代码如程序清单17-1所示。

程序清单17-1　secret.php——提供简单的身份验证机制的PHP和HTML

```
<?php
  //create short names for variables
  $name = $_POST['name'];
  $password = $_POST['password'];

  if ((!isset($name)) || (!isset($password))) {
  //Visitor needs to enter a name and password
?>
    <h1>Please Log In</h1>
    <p>This page is secret.</p>
    <form method="post" action="secret.php">
    <p>Username: <input type="text" name='name"></p>
    <p>Password: <input type="password" name='password'></p>
    <p><input type='submit' name="submit" value="Log In"></p>
    </form>
<?php
  } else if(($name=="user") && ($password=="pass")) {
    // visitor's name and password combination are correct
    echo '<h1>Here it is!</h1>
          <p>I bet you are glad you can see this secret page.</p>';
  } else {
    // visitor's name and password combination are not correct
    echo "<h1>Go Away!</h1>
          <p>You are not authorized to use this resource.</p>";
  }
?>
```

程序清单17-1所示的代码提供了一个简单的身份验证机制，它允许通过身份验证的用户浏览某个网页，但是它存在一些明显的问题。

该脚本：

- 在脚本中只对一个用户名和密码进行了硬编码。
- 将密码以普通文本形式保存。
- 只保护一个页面。
- 以普通文本形式传输密码。

通过不同程度的努力和实践，我们可以逐一解决这些问题。

17.2.1　保存密码

与将用户名和密码保存在脚本中相比，还有许多更好的用来保存用户名和密码的地方。在以上脚本中，要修改数据将会是非常困难的。编写一段脚本来修改脚本本身是可能的，但这却是一个非常糟糕的主意。这意味着服务器中有一脚本，它在服务器中执行，但是可以被其他脚

本写入或者修改。将数据保存在服务器的另一个文件可以使我们能够轻松地编写一个用来添加和删除用户以及修改密码的程序。

在不至于严重影响脚本的执行速度前提下，一个脚本或者其他数据文件内部能够保存的用户数量是有限的。正如前面所介绍的，如果考虑在一个文件中保存并查找大量的项目，应该考虑使用一个数据库来代替。如果要保存和搜索的内容多于100个项目，应该使用一个数据库而不是一个普通文件来实现，这是一条重要的规则。

使用数据库来保存用户名和密码将不会使脚本复杂很多，但是这将允许我们快速地验证不同用户的身份。它也允许我们轻松地编写一段脚本来添加新用户、删除用户并且允许用户修改自己的密码。

通过数据库验证访问者身份的脚本如程序清单17-2所示。

程序清单17-2　secretdb.php——使用MySQL来改进原有的简单身份验证机制

```
<?php
  $name = $_POST['name'];
  $password = $_POST['password'];

  if ((!isset($name)) || (!isset($password))) {
  //Visitor needs to enter a name and password
?>

    <h1>Please Log In</h1>
    <p>This page is secret.</p>
    <form method="post" action="secretdb.php">
    <p>Username: <input type="text" name="name"></p>
    <p>Password: <input type="password" name="password"></p>
    <p><input type="submit" name="submit" value="Log In"></p>
    </form>

<?php
  } else {
    // connect to mysql
    $mysql = mysqli_connect('localhost', "webauth", "webauth");
    if(!$mysql) {
      echo 'Cannot connect to database.";
      exit;
    }
    // select the appropriate database
    $selected = mysqli_select_db($mysql, "auth");
    if(!$selected) {
      echo "Cannot select database.";
      exit;
    }

    // query the database to see if there is a record which matches
    $query = "select count(*) from authorized_users where
```

```
                name = '".$name."' and
                password = '".$password."'";

    $result = mysqli_query($mysql, $query);
    if(!$result) {
      echo "Cannot run query.";
      exit;
    }
    $row = mysqli_fetch_row($result);
    $count = $row[0];

    if ($count > 0) {
    // visitor's name and password combination are correct
    echo "<h1>Here it is!</h1>
          <p>I bet you are glad you can see this secret page.</p>";
    } else {
      // visitor's name and password combination are not correct
      echo "<h1>Go Away!</h1>
            <p>You are not authorized to use this resource.</p>";
    }
  }
?>
```

数据库也可以通过以MySQL root用户的身份连接MySQL并运行程序清单17-3的脚本来创建。

程序清单17-3　createauthdb.sql——这些MySQL查询语句将创建auth数据库、auth表和两个示例用户

```
create database auth;
use auth;
create table authorized_users (name varchar(20),
                               password varchar(40),
                               primary key (name)
                              );
insert into authorized_users values ('username',
                                     'password');

insert into authorized_users values ('testuser',
                                     sha1('password'));
grant select on auth.*
             to 'webauth'
             identified by 'webauth';
flush privileges;
```

17.2.2　密码的加密

无论将数据保存在数据库中还是文件中，都没有必要冒险以普通文本格式存储密码。只要

稍微做一点工作，就可以通过一种使用单向哈希的算法来提高密码的安全性。

PHP提供了许多单向的哈希函数。最早的也是安全性最低的是UNIX Crypt算法，它由crypt()函数实现。md5()函数实现的消息摘要5（Message Digest 5）算法更强大一些。而功能更强大的加密的哈希算法1（SHA-1）。PHP的sha1()函数提供了一个功能强大的单向加密哈希函数。该函数原型如下所示：

```
string sha1 (string str [, bool raw_output])
```

给定字符串str，该函数将返回一个40个字符的伪随机字符串。如果将raw_output参数设置为true，将得到一个20个字符的二进制字符串数据。例如，给定字符串“password”，sha1()函数将返回如下所示的结果：

```
"5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8"
```

这个字符串无法解密，即使是字符串的创建者也无法还原成“password”字符串，因此初看起来，这个函数的作用并不大。但是真正使得该函数功能强大的原因在于它的输出是确定的。假设使用相同的字符串，sha1()函数每次运行都将返回相同的结果。

因此，我们就可以不使用如下所示的PHP代码：

```
if (($name == 'username') &&
    ($password == 'password')) {
  //OK passwords match
}
```

可以使用如下代码：

```
if (($name == 'username') &&
    (sha1($password) == '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8')) {
  //OK passwords match
}
```

在对字符串使用sha1()函数之前，并不需要知道加密结果表示什么。我们只需要知道输入的密码是否与最初通过sha1()运行的结果密码相同。

正如我们已经提到的，将用户名和密码硬编码到脚本中是一个非常糟糕的主意。可以使用一个独立的文件或者数据库存储它们。

如果要使用一个MySQL数据库来保存身份验证数据，可以使用PHP的sha1()函数，或者MySQL的SHA1()函数。MySQL提供了比PHP更广泛的哈希算法，但是二者的作用都是相同的。

要使用SHA1()函数，可以重新编写程序清单17-2中的SQL查询语句，如下所示：

```
select count(*) from authorized_users where
       name = '".$name."' and
       password = sha1('".$password."')
```

该查询将计算表authorized_users中具有用户名与$name的内容相同，并且密码值与已经过SHA1()函数处理后的密码相吻合的记录行数。假设我们强制用户使用唯一的用户名，该查询的结果就是0或1。

请记住，这个哈希函数通常将返回固定大小的数据。在SHA1这种情况下，它是用40个字符的字符串来表示。请确认数据库列具有这样的宽度。

通过程序清单17-3，可以看到我们创建了一个用户（'username'），其密码为未加密的密码，而另一个用户则具有加密的密码（'testuser'），从而对这两种可能的方法进行比较。

17.2.3 保护多个网页

要使类似于脚本程序清单17-1和程序清单17-2的脚本能够保护多个网页将更加困难。因为HTTP是无状态的，来自同一个人的连续请求之间并没有自动的连接或联系。这就使得将数据在页与页之间的传递变得更加困难，例如用户输入的身份验证信息。

保护多个网页最简单的方法就是使用Web服务器提供的访问控制机制。我们将简要地讨论这些内容。

要自己创建该功能，可以在每个要保护的网页中包含程序清单17-1中的部分脚本。使用auto_prepend_file和auto_append_file，可以自动地在特定目录下的每个文件中预先计划并添加所需的代码。要了解这些指令的使用，请参阅第5章。如果使用这种方法，当访问者进入网站后，访问多个网页时又会出现什么情况呢？网站会要求他们每浏览一页时都重新输入名字和密码，显然，这对访问者来说难以接受的。

我们可以将他们输入的细节信息附加网页的每个超级链接中。由于用户可能输入在URL中不允许出现的空格或者其他字符，应该使用urlencode()函数对这些字符进行安全的编码。

尽管如此，该方法仍然会有一些问题。因为这些数据将包含在发送给用户的网页和他们访问的URL中，他们访问的受保护页可以被任何使用同一台计算机的人看见，这些人可以通过回退步骤看到以前的缓存页，或者通过查看浏览器的历史清单而浏览这些受保护页。因为我们在每页被请求或者发送时重复地发送密码到浏览器，这些敏感信息的传输频率更高。

现在，有两个很好的方法可以解决这些问题：HTTP基本身份验证和会话。基本身份验证克服了缓存问题，但是在每次请求时，浏览器仍然会将密码发送给服务器。会话控制技术克服了这两方面问题。下面，我们将首先介绍HTTP基本身份验证，在第23章中，我们将介绍会话控制，在第27章中将更详细地讨论它。

17.3 使用基本身份验证

幸运的是，验证用户只是一个非常常见的任务，因此HTTP内置有身份验证功能。脚本或Web服务器可以通过Web浏览器请求身份验证。Web浏览器负责显示一个对话框或类似设备，从而从用户那里获得所需的信息。

尽管Web服务器对每个用户请求都要求新的身份验证详细信息，但Web浏览器不必在每页中都要求用户输入详细信息。通常，浏览器可以保存这些详细信息，只要用户打开一个浏览器窗口，它就会自动地将这些所需的详细信息重新发送到Web服务器而无须用户介入。

HTTP这个特性叫做*基本身份验证*。使用PHP或者内置于Web服务器中的机制，可以触发这些基本身份验证。接下来，我们将讨论PHP方法、Apache方法。

基本身份验证以普通文本方式传输用户名和密码，因此不太安全。HTT P1.1中包含一种更安全的方法，称为*摘要身份验证*。摘要身份验证使用哈希算法（通常是MD5）掩饰事务处理的细节。摘要身份验证被许多Web服务器和大多数Web浏览器的最新版本所支持。然而，目前还

有许多正在使用的浏览器早期版本不支持摘要身份验证，而且微软的某些IE和IIS版本所包含摘要验证标准的版本与非Microsoft产品并不兼容。

除了对Web浏览器的支持较差之外，摘要身份验证也不很安全。基本身份验证和摘要身份验证都只能提供低级别的安全。它们都不能保证用户正在处理的机器就是他要访问的机器。它们都可能允许入侵者向服务器重复发送同一请求。因为基本身份验证以普通文本的方式传输用户密码，这就使入侵者能够捕获这些信息包，并冒充该用户发送任何其他的请求。

基本身份验证提供的安全级别（低）与人们常用的通过Telnet或FTP连接机器一样，都是以普通文本方式传输密码的。摘要身份验证更安全一些，在传输密码之前将对密码进行加密。

当将SSL和数字证书结合起来实现基本的身份验证时，所有Web事务处理都可以得到全面的保护。如果希望获得全面的安全保护，请参阅第18章。但是对于许多情况来说，一个速度快但是相对不安全的方法却是合适的方法，例如，基本身份验证。

基本身份验证只能保护特定的领域，并且要求用户提供有效用户名和密码。指定领域是因为同一服务器上可以有多个这样的领域。同一台服务器上的不同文件和目录可以成为不同领域的一部分，每个领域由一组不同的用户名和密码来保护。指定领域也允许将一台主机或一台虚拟主机上的多个目录指定为一个领域，并用一个密码来保护它们。

17.4　在PHP中使用基本身份验证

一般地说，PHP脚本是跨平台的，但是基本身份验证的使用却依赖于服务器设置的环境变量。要在使用PHP作为Apache模块的Apache服务器或使用PHP作为ISAPI模块的IIS上运行一个HTTP身份验证脚本，需要检查服务器的类型并相应采取稍微不同的操作。

程序清单17-4所示的脚本可以在这两个服务器上运行。

程序清单17-4　http.php——PHP可以触发HTTP基本身份验证

```
<?php

// if we are using IIS, we need to set
// $_SERVER['PHP_AUTH_USER'] and                                    .
// $_SERVER['PHP_AUTH_PW']

if ((substr($_SERVER['SERVER_SOFTWARE'], 0, 9) == 'Microsoft') &&
    (!isset($_SERVER['PHP_AUTH_USER'])) &&
    (!isset($_SERVER['PHP_AUTH_PW'])) &&
    (substr($_SERVER['HTTP_AUTHORIZATION'], 0, 6) ==  Basic ')
   ) {

  list($_SERVER[ PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW']) =
    explode( : , base64_decode(substr($_SERVER['HTTP_AUTHORIZATION'], 6)));
}

// Replace this if statement with a database query or similar
if (($_SERVER['PHP_AUTH_USER'] != 'user') ||
```

```
    ($_SERVER['PHP_AUTH_PW'] != 'pass')) {

  // visitor has not yet given details, or their
  // name and password combination are not correct

  header( 'WWW-Authenticate: Basic realm="Realm-Name"' );

  if (substr($_SERVER['SERVER_SOFTWARE'], 0, 9) == 'Microsoft') {
    header('Status: 401 Unauthorized');
  } else {
    header('HTTP/1.0 401 Unauthorized');
  }

  echo '<h1>Go Away!</h1>
        <p>You are not authorized to view this resource.</p>';

} else {
  // visitor has provided correct details
  echo "<h1>Here it is!</h1>
        <p>I bet you are glad you can see this secret page.</p>";
}
?>
```

程序清单17-4中给出的代码可以实现与本章前面的程序清单给出的代码实现的类似方法。

如果用户尚未提供身份验证信息，代码就会要求用户输入这些信息。如果用户提供的信息不正确，代码就给用户发送一个拒绝消息。如果提供的用户名、密码都正确，该代码就会显示网页的内容。

在这个例子中，用户将看到与前一个程序清单所不同的界面。我们不再提供要求输入注册信息的HTML表单。用户的浏览器将显示一个对话框。一些人认为这是一个改进；也有些人更喜欢对界面可视部分的完全控制。在这个例子中，由Firefox提供的登录对话框如图17-4所示。

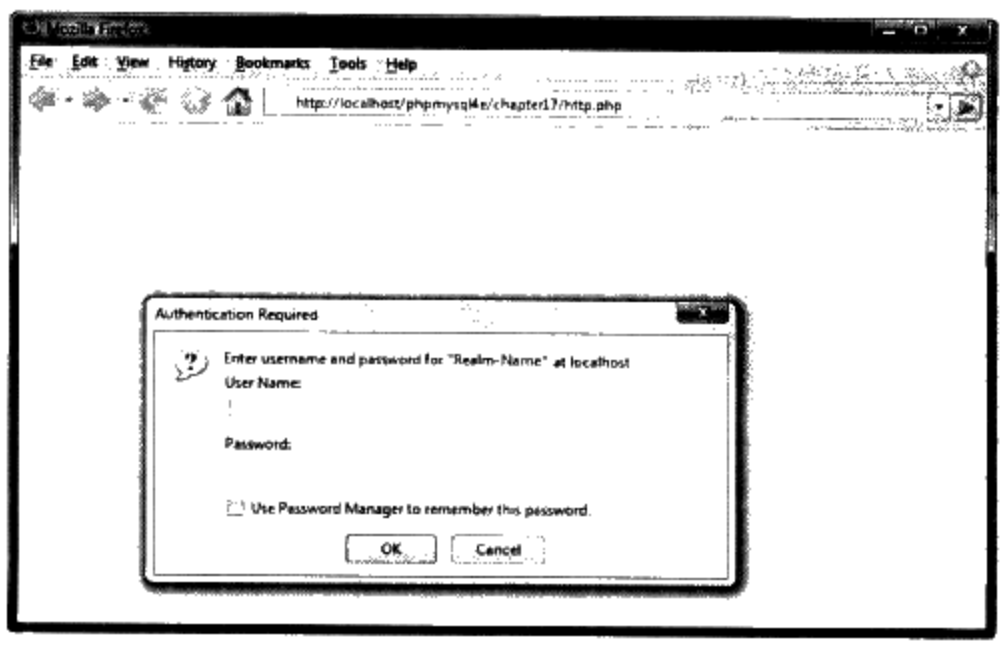

图17-4　使用HTTP 基本身份验证时，用户浏览器负责显示该对话框

因为该身份验证是通过内置于浏览器中的特性来实现的，因此浏览器将判断如何处理失败的身份验证。Internet Explorer在显示拒绝页面之前允许用户尝试验证3次。而Firefox则允许用户尝试无数次，每次尝试失败后都将出现一个对话框，询问“Authorization failed.Retry?（验证失败，是否需要重试？）”如果用户点击“取消”按钮，Firefox才会显示拒绝页面。

对于程序清单17-1和程序清单17-2给出的代码，我们可以将它们包含到要保护的网页里，或者将它们自动预先写入到某个目录的每个文件中。

17.5　在Apache的.htaccess文件中使用基本身份验证

不编写PHP脚本，也可以得到与程序清单17-4非常类似的结果。

Apache Web服务器包含一些不同的身份验证模式，这些模式可以用于判断用户输入数据的有效性。最简单的是使用mod_auth，它将用户名-密码对与服务器中一个文本文件中的行进行比较。

要获得与前面脚本相同的输出结果，必须创建两个不同的HTML文件，一个用来显示成功登录后的内容，另一个用来显示拒绝页面。我们将跳过前例中一些HTML元素，但是真正生成HTML的时候还需要包括<html>和<body>标记。

程序清单17-5包含了经过身份验证后的用户可以看到的内容。该文件名称为content.html。程序清单17-6包含了用户身份验证失败后的拒绝页面内容。文件名称为rejection.html。出错的时候是否显示一个网页是可选的，但是如果在显示的网页中放一些有用的东西，未尝不是一个很好的专业化做法。对于一个用户试图进入保护区域但是被拒绝而显示的网页，有用的内容可能包括如何注册一个密码，或在遗忘密码的情况下，如何通过电子邮件重设密码的说明。

程序清单17-5　content.html——示例内容

```
<html><body>
<h1>Here it is!</h1>
<p>I bet you are glad you can see this secret page.</p>
</body></html>
```

程序清单17-6　rejection.html——401错误的示例页面

```
<html><body>
<h1>Go Away!</h1>
<p>You are not authorized to view this resource.</p>
</body></html>
```

在这些文件中，并没有什么新的内容。在这个例子中，唯一一个有趣的文件是程序清单17-7给出的页面。该文件需要调用.htaccess，它将控制对目录中任何文件和子目录的访问。

程序清单17-7　.htaccess——.htaccess文件可以设置许多Apache配置，包括是否激活身份验证

```
ErrorDocument 401 /chapter17/rejection.html
AuthUserFile /home/book/.htpass
AuthGroupFile /dev/null
AuthName 'Realm-Name'
```

```
AuthType Basic
require valid-user
```

程序清单17-7是一个.htaccess文件，用于在一个目录中开启基本身份验证功能。可以在.htaccess文件中修改许多设置，但是在这个例子中，我们所做6行修改都与身份验证有关。第一行：

```
ErrorDocument 401 /chapter17/rejection.html
```

将告诉Apache对验证失败的访问者显示什么样的文档（HTTP错误号401）。可以使用其他的ErrorDocument指令来提供不同的HTTP错误（例如，404）所需的页面。其语法如下所示：

```
ErrorDocument error_number URL
```

对于一个处理错误401的页面，给定的URL可以被公共访问是非常重要的。如果页面锁定在一个目录里，在该目录中访问者需要成功通过验证才能浏览页面，那么提供一个自定义的错误页面告诉人们身份验证失败并不是非常有意义的。

这一行：

```
AuthUserFile /home/book/.htpass
```

将告诉Apache在什么地方可以找到包含已经通过身份验证的用户的密码文件。通常，这个文件是.htpass，但是我们可以给它取任何自己更喜欢的名字。文件的名称不重要，重要的是保存该文件的位置。该文件不应保存在Web树目录中——因为保存在Web树目录中的话，人们可以通过Web服务器下载它。在这个例子中，我们给出.htpass示例文件，如程序清单17-8所示。

与指定通过身份验证的单个用户一样，指定只有在特定组中通过身份验证的用户才能访问资源也是可能的。我们选择不这样做，因此这行：

```
AuthGroupFile /dev/null
```

可以将AuthGroupFile设置为指向/dev/null，这是UNIX系统中一个特殊的文件，可以保证该文件为空。

与PHP示例一样，要使用HTTP身份验证，需要命名保护区域，如下所示：

```
AuthName 'Realm-Name'
```

可以根据自己的喜好选择任意的区域名称，但是必须记住将该名称向访问者显示。为了明显起见，我们将示例中的名称改为“Realm-Name”。

因为Apache支持许多不同的身份验证方法，必须指定使用哪一种身份验证方法。这里，我们使用的Basic身份验证方法是通过如下指令指定的：

```
AuthType Basic
```

需要指定允许访问的访问者。我们可以指定特定用户、特定组，或者就像这个例子一样，只简单地允许通过身份验证的用户进行访问。代码行：

```
require valid-user
```

指定了任何有效用户都可以访问。

程序清单17-8　.htpass——密码文件存储用户名和已加密的密码

```
user1:8nRp9M80GS7zM
user2:nC13sOTOhp.ow
user3:yjQMCPWjXFTzU
user4:LOmlMEi/hAme2
```

.htpass文件中的每一行都包含一个用户名、冒号和该用户的加密密码。

你的.htpass文件的确切内容会有所不同。要创建它，可以使用一个名为htpasswd的小程序，该程序包含在Apache软件包中。

htpasswd程序用于下列两种方法之一：

```
htpasswd [-cmdps] passwordfile username
```

或者

```
htpasswd -b[cmdps] passwordfile username password
```

唯一使用的开关是-c。使用-c可以告诉htpasswd创建文件。必须在第一个添加的用户中使用这个开关。将它用于其他用户的时候要小心，因为如果该文件存在，htpasswd将删除这个文件并创建一个新文件。

该程序的可选项m、d、p或s开关可以用来指定使用哪种加密算法（包括不加密）。开关b告诉程序要期望密码作为它的参数，而不提示输入密码。如果要作为批处理的一部分交互地调用htpasswd，这个开关就是有意义的，但是如果从命令行调用htpasswd，就不应该使用它。

下列命令创建如程序清单17-8所示的文件：

```
htpasswd -bc /home/book/.htpass user1 pass1
htpasswd -b /home/book/.htpass user2 pass2
htpasswd -b /home/book/.htpass user4 pass3
htpasswd -b /home/book/.htpass user4 pass4
```

请注意，htpasswd可能没有包含在路径中：如果没有，可能需要提供其完整路径。在许多系统中，可以在/usr/local/apache/bin目录下找到它。

这种类型的身份验证容易建立，但是按照这样的方法使用.htaccess文件还存在一些问题。

用户和密码保存在同一个文本文件中。在浏览器每次请求一个被.htaccess文件保护的文件时，服务器都必须解析.htaccess文件，然后再解析密码文件，以试图匹配用户名和密码。不使用.htaccess文件，我们可以在httpd.conf文件中指定同样的事情——httpd.conf文件是该Web服务器的主配置文件。在每次请求一个文件的时候，系统都要解析.htaccess文件。而httpd.conf文件只在服务器启动的时候解析。这样速度将更快，但是也意味着，如果要做修改，需要停止并重新启动服务器。

无论将服务器指令保存在什么地方，对于每次请求，都要搜索密码文件。这就意味着，它与其他使用普通文件的技术一样，对于成千上万的用户来说，这种方法也是不合适的。

17.6　使用mod_auth_mysql身份验证

正如我们已经提到的，在Apache上使用mod_auth易于安装并且效率很高。因为它将用户

信息保存到一个文本文件中，因此对于拥有广泛用户群的繁忙网站，mod_auth的使用是不实际的。

幸运的是，使用mod_auth_mysql，将拥有使用mod_auth的易用性，同时又获得访问数据库的更快速度。该模块的工作方式与mod_auth非常类似，但是因为它使用MySQL数据库而不是文本文件，所以可以在大量用户中进行搜索。
要使用它，需要在系统中编译和安装该模块，或者请系统管理员安装它。

17.6.1 安装mod_auth_mysql

要使用mod_auth_mysql，需要首先安装Apache和MySQL，安装说明请参阅附录A，然后再完成几步额外的步骤。在Apache和MySQL软件包中附带的README和USAGE文件都给出了完善的安装说明，但是某些地方是针对以前版本的。

在这里，我们只给出总结。

1）获得模块的存档文件。它包含在随书附带的文件中，但是也可以从下列网址得到最新版本：http: //sourceforge.net/projects/modauthmysql/

2）解压缩源代码。

3）切换到mod_auth_mysql目录，分别运行make和make install命令。必须在make文件（MakeFile）中告诉它MySQL的安装位置。

4）在httpd.conf文件中添加如下所示行，以便动态地将模块加载到Apache

```
LoadModule Mysql_auth_module libexec/mod_auth_mysql.so
```

5）创建数据库和包含身份验证信息的MySQL表。它并不需要是一个单独的数据库或表；可使用一个已有表，例如在本章前面的示例创建的auth数据库。

6）在httpd.conf文件添加mod_auth_mysql用来连接MySQL所需的参数，该指令如下所示：

```
Auth_MySQL_Info hostname user password
```

检查编译是否正常工作的最简单方法是查看Apache是否可以启动。要启动具有SSL支持类型的Apache，可以使用如下命令：

```
/usr/local/apache/bin/apachectl startssl
```

如果没有启用SSL支持类型，可以输入如下命令：

```
mod_auth_mysql was successfully added.
```

如果启动后在httpd.conf文件中有Auth_MySQL_Info指令，mod_auth_mysql模块就已经成功添加。

17.6.2 使用mod_auth_mysql

在成功地安装该模块之后，使用它与使用mod_auth一样简单。程序清单17-9显示了一个示例的.htaccess文件，该文件用加密密码验证用户身份，加密密码存储在本章前面所创建的数据库中。

程序清单17-9　.htaccess——该.htaccess文件使用MySQL数据库来验证用户身份

```
ErrorDocument 401 /chapter17/rejection.html

AuthName "Realm Name"
AuthType Basic

Auth_MySQL_DB auth
Auth_MySQL_Encryption_Types MySQL
Auth_MySQL_Password_Table authorized_users
Auth_MySQL_Username_Field name
Auth_MySQL_Password_Field password

require valid-user
```

可以看到，程序清单17-9与程序清单17-7的大部分代码都是相同的。我们仍然需要指定一个出错文档来显示发生401错误（此时身份验证失败）时的警告内容。我们还指定了基本身份验证和给出区域名。与程序清单17-7一样，我们将允许任何有效的、通过身份验证的用户进行访问。

因为我们使用的是mod_auth_mysql而且并不希望使用所有默认设置，所以还可以指定一些其他指令，这些指令可以指定mod_auth_mysql如何工作。Auth_MySQL_DB、Auth_MySQL_Password_Table、Auth_MySQL_Username_Field、Auth_MySQL_Password_Field分别用来指定数据库名称、表名称、用户名字段和密码字段。

我们包含了Auth_MySQL_Encryption_Types指令来指定要使用MySQL密码加密。该指令可接受的值包括Plaintext、Crypt_DES或MySQL。而Crypt_DES是其默认值，它使用标准的UNIX DES加密密码。

从用户观点来看，mod_auth_mysql示例的工作方式与mod_auth示例完全一样。当用户访问该网站时，都看到一个由用户浏览器给出的对话框。如果用户成功通过身份验证，浏览器将显示网页内容。如果验证失败，浏览器将显示一个出错页面。

对于许多网站来说，mod_auth_mysql是理想的。它速度快，使用相对简单，而且允许使用任何方便的机制添加新用户数据库条目。如果要求更大的灵活性以及更精确地控制网页各个部分，可能要使用PHP和MySQL来实现身份验证。

17.7　创建自定义身份验证

在本章中，我们已经讨论了创建我们自己的身份验证的方法（这些方法包含一些缺陷和折中），以及使用内置的验证方法（这与编写自己的代码相比，缺乏灵活性）。在本书的稍后内容介绍会话控制时，我们就可以编写自定义的身份验证，从而减少本章中遇到的折中情况。

在第23章中，我们将开发一个简单的用户验证系统。在该系统中，通过使用会话在网页之间记录变量，我们可以避免本章所遇到的一些问题。

在第27章中，我们会将此方法应用到实际项目中，并且讨论如何应用它实现一个精确的身份验证系统。

17.8 进一步学习

RFC 2617指定了HTTP身份验证的详细信息，可以通过下列网址访问：http://www.rfc-editor.org/rfc/rfc2617.txt

而Apache中的用来控制基本验证的mod_auth文档，则可以在如下网址找到：http://www.apache.org/docs/mod/mod_auth.html

mod_auth_mysql文档包含在下载文件中。它非常小，因此如果希望找到更多关于它的信息，可以下载该存档文件并查看readme文件。

17.9 下一章

在下一章中，我们将解释如何在各个处理阶段保护数据，这些阶段包括输入、传输和存储。此外，我们还将介绍SSL、数字证书和加密技术的使用。