第30章　创建一个邮件列表管理器

站点建立了相当数量的订阅客户群之后，可以通过给他们发送新闻信件保持联系。在本章中，我们将实现一个邮件列表管理器（MLM）的前台程序。有些MLM允许每个订阅者给其他订阅者发送消息。我们在本章所创建的程序将是一个新闻信件系统，在这个系统中，只有列表管理员才能发送消息。我们把这个系统叫做金字塔式MLM。

这个系统与市场上其他已有系统相似。如果想了解系统的目标，可以访问如下站点：http://www.topica.com。

我们的程序允许管理员创建多个邮件列表，而且可以单独将新闻信件发送给每一个列表。该程序使用文件上载让管理员可以上载脱机创建的文本或HTML格式的新闻信件。这就意味着管理员可以使用任何自己喜欢的软件来创建新闻信件。

用户可以订阅站点上的任何一个列表，并且选择以文本形式还是HTML格式接收新闻信件。

在本章中，我们将主要介绍以下内容：

- 多个文件的文件上载
- Mime编码的电子邮件附件
- HTML格式的电子邮件
- 不需要人工交互的用户密码管理方法

30.1　解决方案的组成

我们希望建立一个在线新闻信件创作与发送系统。这个系统应该允许创作各种各样的新闻信件发送给用户，并且允许用户订阅这些新闻信件中的一种或多种。

特别地，该解决方案将满足如下所示的目标。

- 管理员应该能够建立和修改邮件列表。
- 管理员应该能够将文本或HTML格式的新闻信件发送给一个列表中的所有订阅者。
- 用户应该能够通过注册使用一个站点，并且可以进入并修改他们的个人资料。
- 用户应该能够订阅该站点的任意一个列表的新闻信件。
- 用户应该能够取消一个邮件列表的订阅。
- 用户应该能够据个人喜好以HTML格式或纯文本格式存储新闻信件。
- 出于安全的原因，用户应该不能将邮件发送到列表，或者不能看见其他用户的邮件地址。
- 用户和管理员应该能够查看有关邮件列表的信息。
- 用户和管理员应该能够查看过去已经发送给某个列表（存档文件）上的新闻信件。

到这里，我们已经对项目有了基本了解，就可以开始设计解决方案及其组成部分，例如，建立一个列表、订阅者和存档新闻信件的数据库，上载脱机创建的新闻信件；以及发送带附件的邮件。

30.1.1　建立列表和订阅者数据库

在这个项目中，我们除了记录系统用户订阅的列表外，还要记录用户的用户名和密码。当然，也要记录每个用户的个人选项：接收文本格式还是HTML格式的邮件，从而能够以适当的格式向用户发送新闻信件。

管理员是一个特殊的用户，他拥有创建新邮件列表和将新闻信件发送给该列表用户的权限。

对过去的新闻信件进行存档也是该系统一个很好的功能。订阅者可能没有保存以前的新闻信件，但是订阅者可能想要查找其中的某些东西。存档文件还可以作为新闻信件的一种市场推广工具，因为订阅者可以看到以前的新闻信件的主要内容。

在MySQL中建立该数据库，并且在PHP中创建访问MySQL的界面是非常简单的，没有什么新内容，也没有什么困难。

30.1.2　上载新闻信件

正如前面提到的，我们需要一个管理员界面来发送新闻信件。目前，我们还没有讨论过的内容是管理员如何创建新闻信件。系统可以为管理员提供一个表单，在这个表单中，管理员可以输入或者粘贴新闻信件内容。然而，让管理员在他喜爱的编辑器中创建新闻信件然后再上载到Web服务器，将增加系统的用户友好性。这也会使得管理员将图像加入到HTML新闻信件中变得更加简单。因此，我们可以使用在第19章“与文件系统和服务器的交互”中讨论的文件上载功能。

我们使用一个比前面项目中所用过的更加复杂的表单。对于这个项目，我们要求管理员上载文本格式和HTML格式的新闻信件，同时包括嵌入到HTML中的所有图像。

上载新闻信件后，需要建立一个界面，以便管理员在发送之前浏览该新闻信件。这样可以确保所有的文件都被正确地上载了。

请注意，系统也可以将所有这些文件保存在一个存档目录中，这样用户可以重复阅读新闻信件。这个目录必须具有Web服务器用户的写入权限。上载脚本尝试将新闻信件写入`./archive/`目录，这样我们必须确保创建了该目录并且正确设置了该目录的权限。

30.1.3　发送带附件的邮件

在这个项目中，我们想要根据用户喜好，既能够向用户发送纯文本格式的新闻信件，也能够发送“别致的”HTML格式的新闻信件。

要发送一个嵌入了图像的HTML文件，需要找到一种方法来发送附件。PHP简单的`mail()`函数并不能直接支持附件的发送。取而代之的是，我们将使用PEAR中的`Mail_Mime`包（该包最初由Richard Heyes编写）。它可以处理HTML附件，而且可以自动附加任何包含在HTML文件中的图像。

在附录A“安装PHP和MySQL”的“PEAR安装”一节中，给出了该包的安装说明。

30.2　解决方案概述

正如我们在第29章中所做的一样，对于这个项目，我们将再次使用事件驱动的方法来编写代码。

我们仍然将以系统流程图的绘制开始，绘制了用户使用系统的可能方式。在这个例子中，我们绘制了3幅代表用户与系统可能进行的3种不同交互方式的流程图。当用户没有登录时，用户拥有不同的权限。而当用户以普通用户或管理员身份登录时，他们又具有不同的权限。这些权限分别如图30-1、图30-2和图30-3所示。

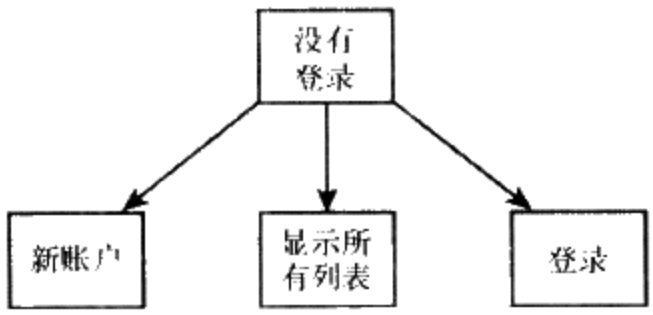

图30-1　当一个用户没有登录时，他只能进行有限的操作

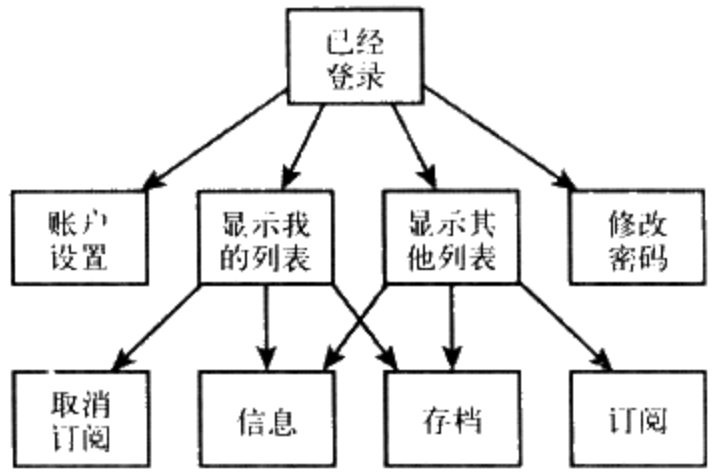

图30-2　用户登录后，可以通过不同的选项修改个人喜好

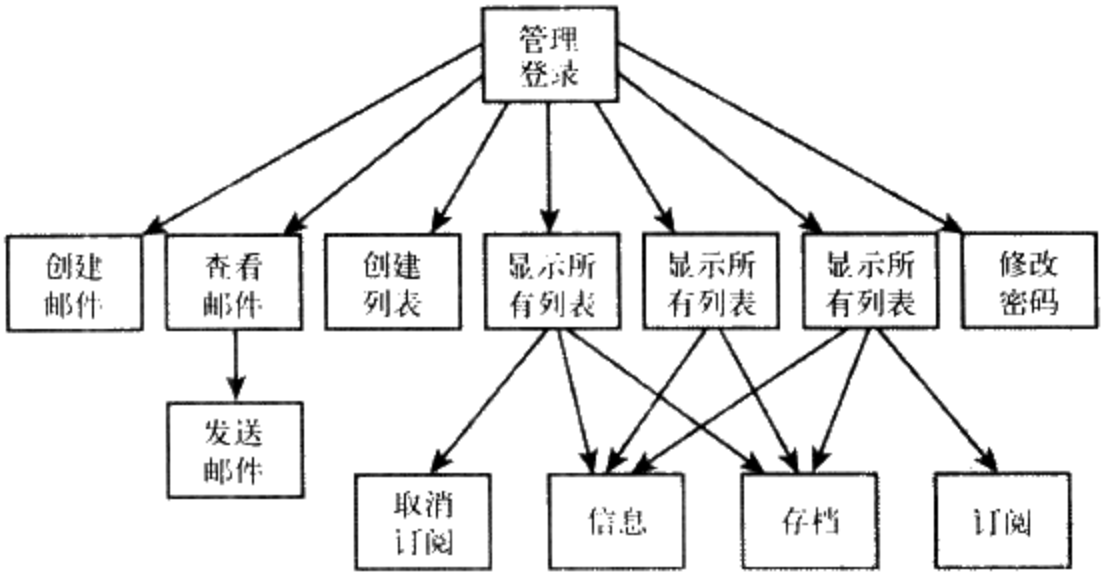

图30-3　管理员拥有专用的操作

在图30-1中，可以看到没有登录的用户所能进行的操作。可以看到，该用户可以登录（如果已经有一个账户）、创建一个账户（如果还没有账户），或者查看可供订阅的邮件列表（作为一种市场营销策略）。

图30-2显示了用户登录后可以进行的操作。用户可以修改账户设置（邮件地址和个人喜好），更改密码，修改所订阅的邮件列表。

图30-3显示了系统管理员登录后可以进行的操作。可以看到，系统管理员除了拥有大多数普通用户拥有的功能，还具有一些额外选项。管理员还可以创建新的邮件列表，通过上载文件

为某个邮件列表创建新邮件，并且在发送这些邮件前对它们进行预览。

由于我们再次使用了事件驱动的方法，因此应用程序的架构包含在文件index.php中，由它调用一系列函数库。该程序用到的文件总览如表30-1所示。

我们将以自己的方式来完成项目，以创建存储订阅者和列表信息的数据库开始。

表30-1　邮件列表管理器应用程序所用到的文件

文件名称	类　型	描　述
index.php	应用程序	运行整个程序的主要脚本
include_fns.php	函数	该程序包含文件的集合
data_valid_fns.php	函数	验证输入数据的函数集合
db_fns.php	函数	连接到mlm数据库的函数集合
mlm_fns.php	函数	该应用程序专用的函数集合
output_fns.php	函数	输出HTML的函数集合
upload.php	组件	该脚本对管理员使用的文件上载组件进行管理。单独列出来以简化安全机制
user_auth_fns.php	函数	用户验证的函数集合
create_database.sql	SQL	用来设置mlm数据库，设置Web用户和管理员用户的SQL

30.3　建立数据库

对于这个应用程序，需要保存如下所示的细节信息。

- **Lists**（列表）：可供订阅的邮件列表。
- **Subscribers**（订阅者）：系统用户和他们的个人喜好信息。
- **Sublists**（订阅列表）：记录哪些用户订阅了哪些邮件列表（多对多的关系）。
- **Mail**（邮件）：已经发送的邮件消息的记录。
- **Images**（图像）：由于我们想要发送由多个文件组成的邮件消息，必须跟踪每个邮件附带的文本、HTML和图像。

用来创建该数据库的SQL如程序清单30-1所示。

程序清单30-1　create_database.sql——创建mlm数据库的SQL

```
create database mlm;

use mlm;

create table lists
(
    listid int auto_increment not null primary key,
    listname char(20) not null,
    subinfo varchar(255)
);

create table subscribers
(
```

```
  email char(100) not null primary key,
  realname char(100) not null,
  mimetype char(1) not null,
  password char(40) not null,
  admin tinyint not null
);

# stores a relationship between a subscriber and a list
create table sub_lists
(
  email char(100) not null,
  listid int not null
);

create table mail
(
  mailid int auto_increment not null primary key,
  email char(100) not null,
  subject char(100) not null,
  listid int not null,
  status char(10) not null,
  sent datetime,
  modified timestamp
);

#stores the images that go with a particular mail
create table images
(
  mailid int not null,
  path char(100) not null,
  mimetype char(100) not null
);

grant select, insert, update, delete
on mlm.*
to mlm@localhost identified by 'password';

insert into subscribers values
('admin@localhost', 'Administrative User', 'H', sha1('admin'), 1);
```

注意可以通过输入如下命令来执行这个SQL脚本：

```
mysql -u root -p < create_database.sql
```

必须提供root用户密码（当然，也可以通过任何具有适当权限的MySQL用户来执行该脚本；这里，我们为了简便仅使用root用户）。在运行这个脚本之前，应该修改该脚本中的mlm用户和管理员的密码。

该数据库的某些字段需要稍作解释，我们简单地看一下。lists表包括listid字段和

listname字段。它还包括一个blurb字段，它描述了该列表的主要内容。

subscribers表包括电子邮件地址（E-mail）字段和订阅者的名字（realname）字段。该表还保存了用户密码和一个用来表明该用户是否是管理员的标记（admin）字段。我们还要在mimetype字段中存储用户想要接收的邮件类型。这里，可以用H代表HTML或T代表文本。

sublists表包括关于来自subscribers表的邮件地址（E-mail）和来自lists表的listids。

mail表包括关于通过系统发送的每条邮件消息的信息。它保存了唯一标识（mailid）、邮件从何处发送而来的地址（E-mail）、邮件的主题行（subject）和已经发送的或将要被发送的列表listid。消息的实际文本或HTML版本可能是一个大文件，因此我们在数据库之外存储实际消息的存档文件。我们还保存一些常规状态信息：消息是否已经被发送了（status），什么时候发送的（sent），以及显示该记录最后一次修改的时间戳（modified）。

最后，我们用images表来记录任何与HTML消息有关的图像。同样，这些图像可能非常大，因此出于效率的考虑，我们将它们保存在数据库之外。我们需要记录与之相关联的mailid、图像实际存储位置的路径和图像的MIME类型（mimetype），例如image/gif。

前面所示的SQL脚本还将为PHP创建一个用户，通过这个用户，PHP可以连接数据库，同时，该用户还是系统的管理用户。

30.4　定义脚本架构

与前一个项目一样，对于这个项目，我们也采用了事件驱动的方法。该程序的架构保存在文件index.php中。该脚本有4个主要的部分，它们分别如下。

1. 执行预处理：标题发送前必须完成的所有处理工作。
2. 建立和发送标题：创建与发送HTML页的开始部分。
3. 执行一个动作：响应已经传入的事件。正如前一个例子一样，事件包含在$action变量中。
4. 发送页脚。

几乎所有的应用程序处理都是在这个文件中完成的。该程序还使用了表30-1中列出的函数库，正如前面所介绍的。

index.php脚本的所有代码如程序清单30-2所示。

程序清单30-2　index.php——金字塔式MLM的主要应用程序文件

```
<?php
/*************************************************************************
* Section 1 : pre-processing
*************************************************************************/

  include ( 'inc_sdc_fun.php');
  session_start();

  $action = ( GET['action'];
  $buttons = array();
```

```
//append to this string if anything processed before header has output
$status = '';

// need to process log in or out requests before anything else
if(!$_POST['email']) && !$_POST['password'])
  $login = login($_POST['email'], $_POST['password']);

  if($login == 'admin') {
    $status .= '<p style="padding-bottom: 10px">
                <strong>'.get_real_name($_POST['email']).'</strong>
                logged in successfully as
                <strong>Administrator</strong></p>';
    $_SESSION['admin_user'] = $_POST['email'];

  } else if($login == 'normal') {
    $status .= '<p style="padding-bottom: 10px">
                <strong>'.get_real_name($_POST['email']).'</strong>
                logged in successfully.</p>';
    $_SESSION['normal_user'] = $_POST['email'];

  } else {
    $status .= '<p style="padding-bottom: 10px">Sorry, we could
                not log you in with that email address
                and password.</p>';
  }

  if($action == 'log-out') {
    unset($action);
    $_SESSION = array();
    session_destroy();
  }

/*****************************************************************************
* Section 2: set up and display header
*****************************************************************************/
  // set the buttons that will be on the top bar
  if(check_normal_user()) {
    // if a normal user
    $buttons[0] = 'change-password';
    $buttons[1] = 'account-settings';
    $buttons[2] = 'show my lists';
    $buttons[3] = 'show-other-lists';
    $buttons[4] = 'log out';
  } else if(check_admin_user()) {
    // if an administrator
    $buttons[0] = 'change-password';
    $buttons[1] = 'create-list';
```

```
    $buttons[4] =  'create-mail' ;
    $buttons[3] =  'view-mail' ;
    $buttons[2] =  'log-out';
    $buttons[1] = 'show-all-lists' ;
    $buttons[0] =  'show-my-lists';
    $buttons[7] =  'show-other-lists' ;
  } else {
    // not logged in at all
    $buttons[5] = 'new-account';
    $buttons[1] = 'show-all-lists' ;
    $buttons[6] = 'log-in';
  }

  if($action) {
    // display header with application name and description of page or action
    do_html_header('Pyramid-MLM - '.format_action($action));
  } else {
    // display header with just application name
    do_html_header( 'Pyramid-MLM' );
  }

  display_toolbar($buttons);

  //display any text generated by functions called before header
  echo $data;

/*************************************************************************
* Section 3: perform action
*************************************************************************/

  // only these actions can be done if not logged in
  switch ($action) {
    case 'new-account':
    // get rid of session variables
    session_destroy();
    display_account_form();
  break;

  case 'store-account':
    if (store_account($_SESSION['normal_user'],
              $_SESSION['admin_user'], $_POST)) {
      $action = '';
    }

    if(!check_logged_in()) {
      display_login_form($action);
    }
```

```
break;

case 'log-in' :

case '' :
  if(!check_logged_in()) {
    display_login_form($action);
  }
break;

case 'show-all-lists':
  display_items( All Lists', get_all_lists(), 'information',
                 'show-archive ,' );
break;

case 'show-archive':
  display_items('Archive For ' .get_list_name($_GET['id']),
                get_archive($_GET['id'']), 'view-html',
                'view-text', '');
break;

case 'information':
  display_information($_GET['id']);
break;
}

//all other actions require user to be logged in
if(check_logged_in()) {
  switch ($action) {
    case 'account-settings':
      display_account_form(get_email(),
            get_real_name(get_email()), get_mimetype(get_email()));
    break;

    case 'show-other-lists':
      display_items( Unsubscribed Lists ,
                    get_unsubscribed_lists(get_email()), 'information',
                     show-archive , 'subscribe' );
    break;

    case 'subscribe':
      subscribe(get_email(), $_GET['id']);
      display_items('Subscribed Lists , get_subscribed_lists(get_email()),
                    'information', 'show-archive ,  unsubscribe' );
    break;

    case 'unsubscribe':
```

```
unsubscribe(get_email(), $_GET['id']);
  display_items('Subscribed Lists', get_subscribed_lists(get_email()),
                'information', 'show-archive', 'unsubscribe');
break;

case '':
case 'show-my-lists':
  display_items('Subscribed Lists', get_subscribed_lists(get_email()),
                'information', 'show-archive', 'unsubscribe');
break;

case 'change-password':
  display_password_form();
break;

case 'store-change-password':
if(change_password(get_email(), $_POST['old_passwd'],
  $_POST['new_passwd'], $_POST['new_passwd2'])) {
 echo '<p style=\"padding-bottom: 50px;\">OK: Password
       changed.</p>";
  } else {
    echo '<p style=\"padding-bottom: 50px\">Sorry, your
          password could not be changed.</p>";
    display_password_form();
  }
    break;
  }
}

// The following actions may only be performed by an admin user
if(check_admin_user()) {
  switch ($action) {
    case 'create-mail':
      display_mail_form(get_email());
    break;

    case 'create-list':
      display_list_form(get_email());
    break;

    case 'store-list':
      if(store_list($_SESSION['admin_user'], $_POST)) {
        echo '<p style=\"padding-bottom: 50px\">New list added.</p>';
        display_items('All Lists', get_all_lists(), 'information',
                      'show-archive');
      } else {
        echo "<p style=\"padding-bottom: 50px\">List could not be
```

```
                stored. Please try uga '.</p>';
        }
      break;

      case 'send' :
        send($_GET['id'], $_SESSION['admin_user']);
      break;

      case 'view mail' :
        display_items('Unsent Mail', get_unsent_mail(get_email()),
                      'preview.html', 'no-view-text', 'send');
      break;
    }
  }

/*************************************************************************
* Section 4: display footer
*************************************************************************/

  do_html_footer();
?>
```

在以上程序清单中，可以看出代码4个部分清楚地标记出来了。在预处理阶段，我们将建立会话，并且完成在标题发送前必要的处理，在这个例子中，包括登录和登出。

在标题阶段，我们设置用户将会见到的菜单按钮，并且使用output_fns.php中的函数do_html_header()来显示相应的标题。该函数仅显示标题栏和菜单，我们将不详细介绍。

在脚本的主要部分，我们根据用户选择的操作做了相应的响应。这些操作分为3个子集：用户没有登录时可以进行的操作、普通用户可以进行的操作，以及管理员用户可以进行的操作。使用check_logged_in()和check_admin_user()函数来检查用户是否允许进行后两组操作。这3个函数都位于user_auth_fns.php函数库中。这些函数以及check_normal_user()函数的代码如程序清单30-3所示。

程序清单30-3　user_auth_fns.php中的函数——这些函数检查用户是否登录以及以什么级别登录

```
function check_normal_user() {
// see if somebody is logged in and notify them if not

  if (isset($_SESSION['normal_user'])) {
    return true;
  } else {
    return false;
  }
}

function check_admin_user() {
// see if somebody is logged in and notify them if not
```

```
    if (isset($_SESSION['admin_user'])) {
      return true;
    } else {
      return false;
    }
}

function check_logged_in() {
  return ( check_normal_user() || check_admin_user() );
```

可以看到，这些函数使用会话变量normal_user和admin_user来检验用户是否登录。稍后，我们将讨论如何设置这些会话变量。

在该脚本的最后部分，我们使用output_fns.php中的函数do_html_footer()来发送HTML页脚。

我们简要地了解一下该系统中可能发生的动作，如表30-2所示。

表30-2　邮件列表管理器应用程序中可能的操作

动　作	执 行 者	描　述
log-in	任何人	为用户给出一个登录表单
log-out	任何人	结束一个会话
new-account	任何人	为用户创建一个新账户
store-account	任何人	保存账户详细信息
show-all-lists	任何人	显示可供使用的邮件列表的清单
show-archive	任何人	显示特定列表的存档新闻信件
information	任何人	显示特定列表的基本信息
account-settings	登录用户	显示用户账户设置
show-other-lists	登录用户	显示用户没有订阅的邮件列表
show-my-lists	登录用户	显示用户已经订阅的邮件列表
subscribe	登录用户	用户订阅特定列表
unsubscribe	登录用户	取消用户对特定列表的订阅
change-password	登录用户	显示修改密码表单
store-change-password	登录用户	在数据库中更新用户密码
create-mail	管理员	显示上载新闻信件表单
create-list	管理员	显示创建新邮件列表表单
store-list	管理员	在数据库中保存邮件列表的详细信息
view-mail	管理员	显示已经上载但仍未发送的新闻信件
send	管理员	向订阅者发送新闻信件

在这个表中，一个值得注意的省略是关于store-mail行的选项，即管理员通过create-mail动作上载输入新闻信件的操作。事实上，这个单独的功能是在另一个不同的文件upload.php中实现的。我们把它放在单独的一个文件中是为了减轻程序员对安全问题的注意。

在接下来的内容中，我们将讨论表30-2中给出的3组操作的实现，也就是，未登录用户的操作、登录用户的操作和管理员的操作。

30.5 实现登录

当一个新用户访问站点的时候，我们希望他完成3件事：首先，查看我们所提供的服务；其次，在网站进行注册；最后，登录。我们将依次讨论这3件事。

在图30-4中，可以看到用户每一次光临站点时我们所提供的页面。

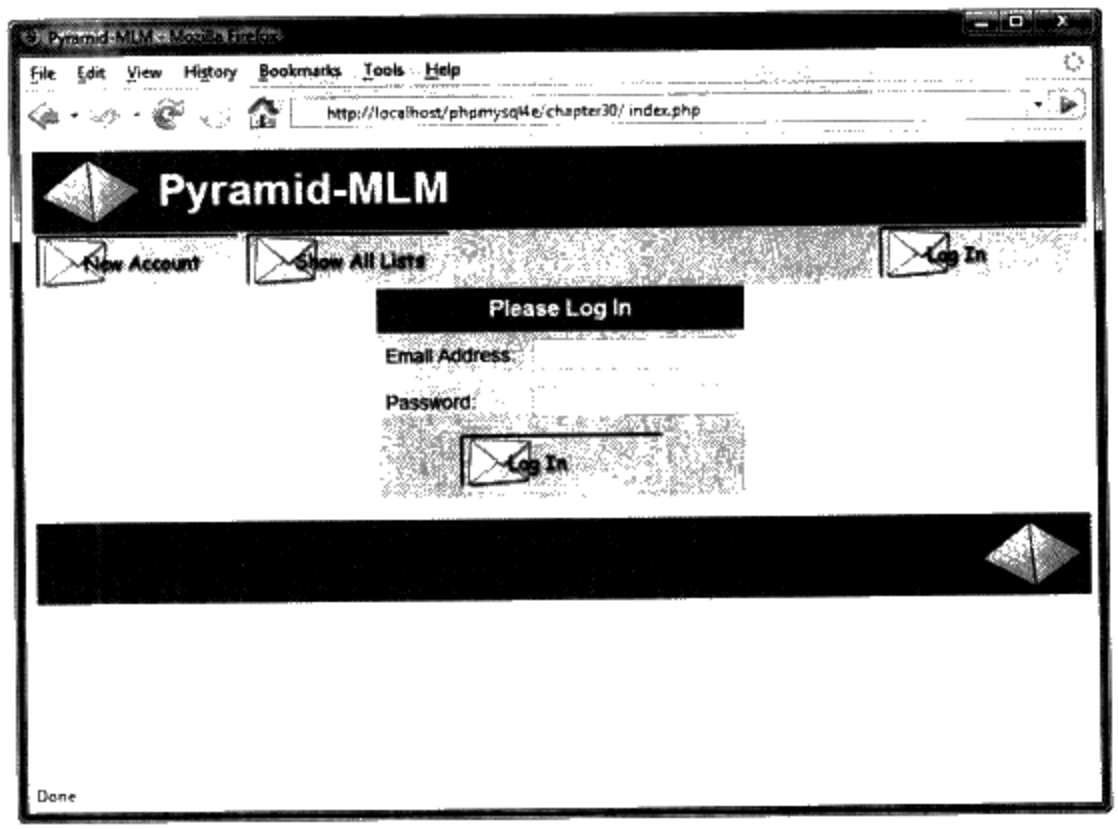

图30-4 访问该站点后，用户可以创建新账户、查看可用的邮件列表或进行登录

现在，我们先来了解新账户的创建和用户登录。在本章稍后的“用户函数的实现”和“管理函数的实现”部分中，我们将回顾列表细节。

30.5.1 新账户的创建

如果用户选择“New Account”菜单选项，将触发new-account动作。这样就会触发index.php中的如下代码段：

```
case 'new-account':
  // get rid of session variables
  session_destroy();
  display_account_form();
break;
```

如果用户当前处于登录状态，这段代码将会登出用户，并将显示如图30-5所示的账户详细信息表单。

该表单是由output_fns.php文件的display_ account_form()函数生成的。这个函数除了在这里使用外，还在account-settings动作中使用，因为它可以用来显示一个创建新账户的表单。如果该函数由account-settings动作触发，表单会预先填好当前已有的用户数据。在这个例子中，表单是空的，准备接受新的账户信息。由于该函数只输出HTML，

在这里，我们不做详细介绍。

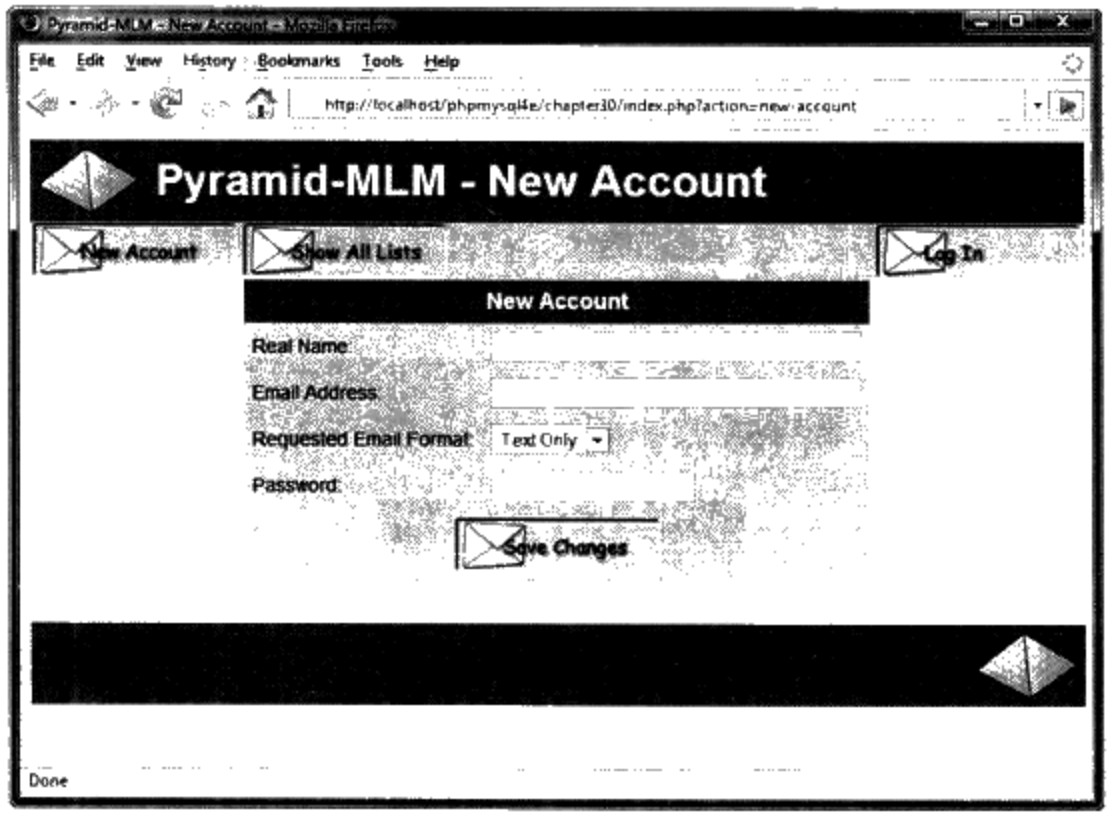

图30-5　新账户创建表单允许用户输入详细信息

以上表单中的提交按钮将触发store-account动作。该动作的代码如下所示：

```
case 'store-account':
  if (store_account($_SESSION['normal_user'],
          $_SESSION['admin_user'], $_POST)) {
    $action = '';
  }

  if(!check_logged_in()) {
    display_login_form($action);
  }
break;
```

store_account()函数将用户的账户信息写入数据库中。程序清单30-4给出了该函数的代码。

程序清单30-4　mlm_fns.php脚本中的store_account()函数——这些函数可以添加一个新用户或者修改数据库中一个已有的用户

```
// add a new subscriber to the database, or let a user modify their data
function store_account($normal_user, $admin_user, $details) {
  if(!filled_out($details)) {
    echo "<p>All fields must be filled in. Try again.</p>";
    return false;
  } else {
    if(subscriber_exists($details['email'])) {
      //check logged in as the user they are trying to change
```

```
  if(get_email()==$details['email']) {
    $query = 'update subscribers set
              realname = '.$details['realname'].',
              mimetype = '.$details['mimetype'].'
              where email = '".$details['email']."'";

    if($conn=db_connect()) {
      if ($conn->query($query)) {
        return true;
      } else {
        return false;
      }
    } else {
      echo "<p>Could not store changes.</p>";
      return false;
    }
  } else {
    echo '<p>Sorry, that email address is already registered here.</p>
          <p>You will need to log in with that address to
             change its settings.</p>";
    return false;
  }
} else {
  // new account
    $query = 'insert into subscribers
              values ('.$details['email'].",
                      '".$details['realname']."',
                       ".$details['mimetype'].",
                      sha1('.$details['new_password']."),
                      0)";

    if($conn=db_connect()) {
      if ($conn->query($query)) {
        return true;
      } else {
        return false;
      }
    } else {
      echo "<p>Could not store new account.</p>";
      return false;
    }
  }
}
```

该函数首先检查用户是否填好了所需的资料。如果这些信息已经填好，函数接着将创建新用户，或者如果该用户已经存在，函数将更新该账户资料。用户只能更新当前登录用户的账户资料。

这是通过调用get_email()函数来检查的，该函数将得到当前登录用户的邮件地址。在稍后的内容中，我们还将接触这个函数，因为它使用了用户登录时设置的会话变量。

30.5.2 登录

如果用户填写了前面图30-4中给出的登录表单，并且点击“Log In”（登录）按钮后，他将进入index.php脚本，同时email和password变量也将被传递到这个脚本。这将触发登录代码，登录代码是整个脚本的预处理阶段，如下所示：

```
// need to process log in or out requests before anything else
if((isset($_POST['email']) && (isset($_POST['password'])) {
  $login = login($_POST['email'], $_POST['password']);
  if($login == 'admin') {
    $status = "<p style=\"padding-bottom: 50px\">
              <strong>".get_real_name($_POST['email'])."</strong>
              logged in successfully as
              <strong>Administrator</strong>.</p>";
    $_SESSION['admin_user'] = $_POST['email'];

  } else if($login == 'normal') {
    $status = "<p style=\"padding-bottom: 50px\">
              <strong>".get_real_name($_POST['email'])."</strong>
              logged in successfully.</p>";
    $_SESSION['normal_user'] = $_POST['email'];
  } else {
    $status = "<p style=\"padding-bottom: 50px\">Sorry, we could
              not log you in with that email address
              and password.</p>";
  }
}
if($action == 'log-out') {
  unset($action);
  $_SESSION=array();
  session_destroy();
}
```

可以看到，我们首先使用user_auth_fns.php库中的login()函数让用户进行登录。这个函数与我们在其他地方用到的登录函数有所不同，因此我们对其进行分析。该函数的代码如程序清单30-5所示。

程序清单30-5　user_auth_fns.php库中的login()函数——该函数检查用户登录资料

```
function login($email, $password) {
// check username and password with db
// if yes, return login type
// else return false

  // connect to db
```

```
$conn = db_connect();
if (!$conn) {
  return 0;
}

$query = 'select admin from subscribers
                      where email='".$email."'
                      and password = sha1('".$password."')";

$result = $conn->query($query);
if (!$result) {
  return false;
}

if ($result->num_rows<1) {
  return false;
}

$row = $result->fetch_array();

if($row[0] == 1) {
  return 'admin';
} else {
  return 'normal';
}
}
```

在前面讨论过的登录函数中，如果登录成功，将返回true，否则将返回false。在这个例子中，登录失败仍然返回false，但是如果登录成功，将返回用户类型，用户类型可以是“管理员”或者“普通用户”。通过检索存储在subscribers表中的admin列，我们可以判断由邮件地址和密码组成一个特定组合的用户类型。如果没有返回结果，则返回false，如果用户是管理员，返回值为1（true），这样返回“admin”。否则，返回“normal”。

回到以上代码执行的主要部分，可以看到，我们注册了一个用于跟踪该用户身份的会话变量。如果该用户是管理员，该变量将是admin_user；如果是普通用户，该变量则是normal_user。无论我们设置的是哪一个变量，该变量都将包括用户的邮件地址。为了简化用户邮件地址的检查，我们使用前面介绍的get_email()函数。该函数代码如程序清单30-6所示。

程序清单30-6 user_auth_fns.php中的get_email()函数——返回登录用户的邮件地址

```
function get_email() {
  if (isset($_SESSION['normal_user'])) {
    return $_SESSION['normal_user'];
  }

  if (isset($_SESSION['admin_user'])) {
    return $_SESSION['admin_user'];
```

```
    }

    return false;
}
```

回到主程序中，我们将报告用户是否已经登录，以及以什么级别登录。图30-6所示的是一个登录操作的输出。

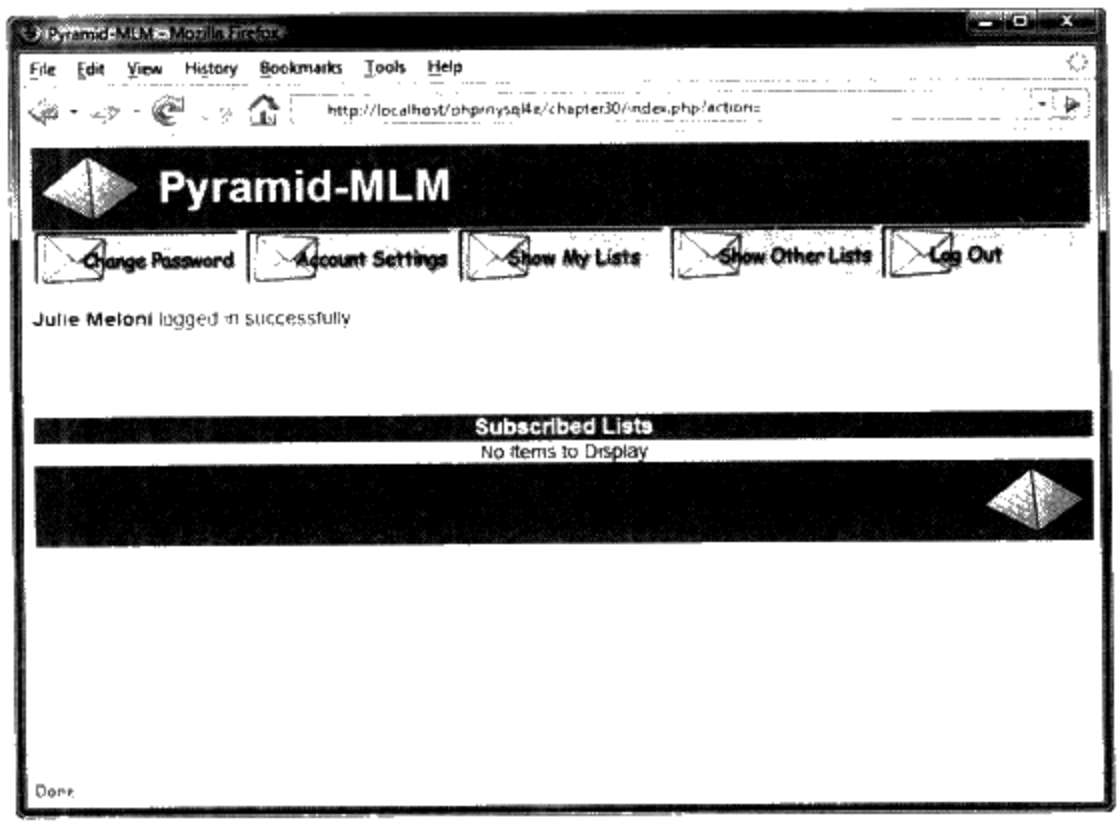

图30-6　系统报告用户登录成功

现在我们已经完成了一个用户的登录，下面讨论用户函数。

30.6　用户函数的实现

用户登录后，我们希望他们能够完成如下所示的5件事：

- 查看可供订阅的邮件列表
- 订阅或取消订阅邮件列表
- 修改建立账户的方法
- 更改密码
- 登出

在图30-6中，可以看到这些选项的大部分。接下来，我们介绍每个选项的具体实现。

30.6.1　查看列表

在这个项目中，我们将实现很多选项，这些选项可以用来查看可供使用的列表及列表细节。在图30-6中，可以看到其中两个选项："Show My Lists"，该选项用来获得该用户订阅的邮件列表；"Show Other Lists"，来获得该用户没有订阅的邮件列表。

如果回头看看图30-4，会发现还有另一个选项——“Show All Lists”，该选项可以获得系统中所有可供使用的邮件列表。为了使系统真正可以升级，我们应增加分页功能（例如，每页显示10个查询结果）。出于简单的考虑，我们没有使用这项功能。

这3个选项将分别触发show-all-lists、show-other-lists和show-my-lists动作。我们可能已经意识到，所有这些动作的工作方式十分相似。以上3种动作的代码如下所示：

```
case 'show-all-lists':
  display_items( All Lists', get_all_lists(), 'information',
                 show-archive ,'');
break;
case 'show-other-lists':
    display_items('Unsubscribed Lists',
                  get_unsubscribed_lists(get_email()), 'information',
                   show-archive', 'subscribe');
break;
case '':
case 'show-my-lists :
   display_items('Subscribed Lists', get_subscribed_lists(get_email()),
                 'information', 'show-archive , 'unsubscribe');
break;
```

可以看到，所有这些操作都将调用output_fns.php库中的display_items()函数，但是每一个操作对该函数的调用都有不同的参数。它们也将使用前面提到的get_email()函数来获得用户的电子邮件地址。

图30-7所示的是这个函数的功能。

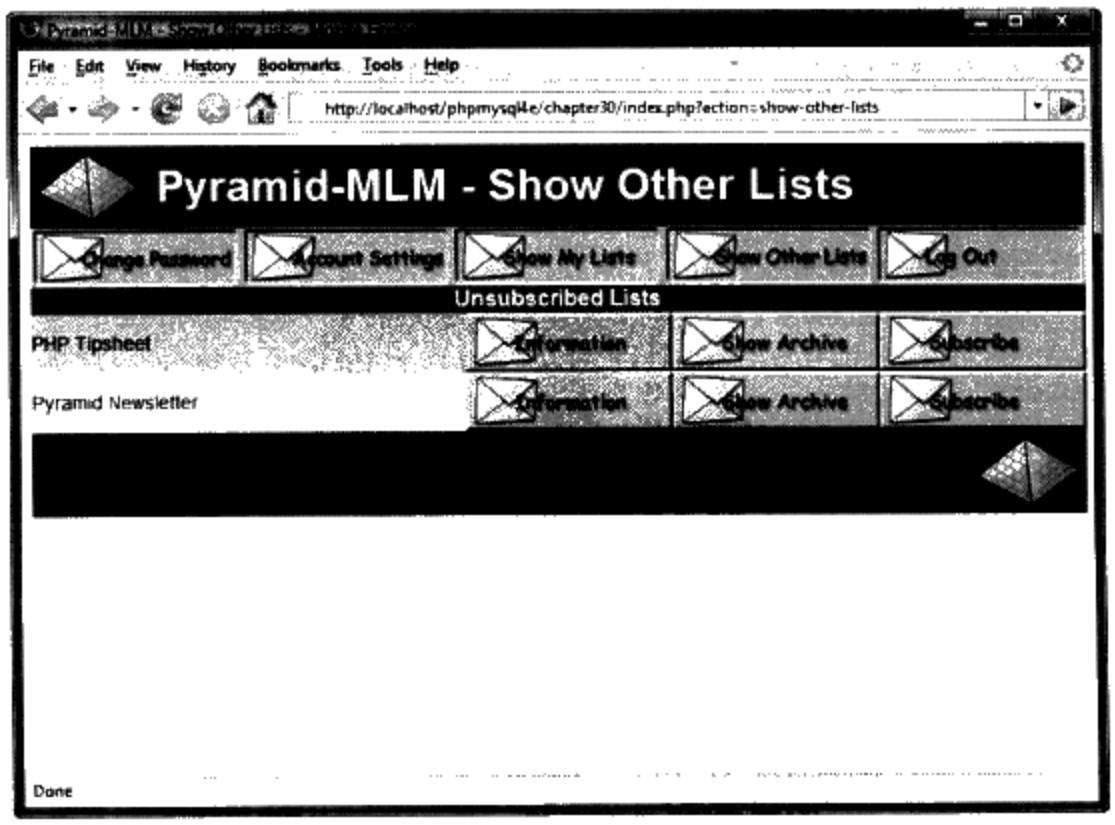

图30-7　display_items()函数用来列出用户未订阅的列表

以上就是“Show Other Lists”页面。

下面，让我们来看看display_items()函数的代码，如程序清单30-7所示。

程序清单30-7　output_fns.php库中的display_items()函数——该函数用来显示相关操作的条目列表

```
function display_items($title, $list, $action1='', $action2='',
        $action3='' ) {
  global $table_width;
  echo '<table width=\"$table_width\" cellspacing=\'0\'
         cellpadding=\'0\" border=\"0\">";

  // count number of actions
  $actions=(($action1!='' ) + ($action2!='') + ($action3!=''));

  echo "<tr>
        <th colspan=\"".(1+$actions)."\" bgcolor=\"#5B69A6\'>"
        .$title."</th>
        </tr>";

  // count number of items
  $items=sizeof($list);

  if($items == 0) {
    echo '<tr>
        <td colspan=\"".(1+$actions)." \' align=\'center\'>No
        Items to Display</td>
        </tr>";
  } else {
    // print each row
    for($i=0; $i<$items; $i++) {
      if($i%2) {
        // background colors alternate
        $bgcolor="#ffffff";
      } else {
        $bgcolor="#ccccff";
      }

      echo "<tr>
            <td bgcolor=\"".$bgcolor." \'
            width=\"".($table_width - ($actions * 149))."\">";

      echo $list[$i][1];

      if ($list[$i][2]) {
        echo " - ".$list[$i][2];
      }
        echo '</td>";
```

```
    // create buttons for up to three actions per line
    for($j=1; $j<=3; $j++) {
      $var="action".$j;

      if($$var) {
        echo "<td bgcolor=\"".$bgcolor."\" width=\"149\">";
        // view/preview buttons are a special case as they link to a file
        if(($$var == 'preview-html') || ($$var == 'view-html') ||
           ($$var == 'preview-text') || ($$var == 'view-text')) {
          display_preview_button($list[$i][3], $list[$i][0], $$var);
        } else {
          display_button($$var, "&id=" . $list[$i][0] );
        }
        echo "</td>";
      }
    }
    echo "</tr>\n";
  }
  echo "</table>";
 }
}
```

该函数将输出一个条目表，每个条目都具有3个相关的操作按钮。该函数具有5个参数，它们依次如下所示。

- $title是显示在表顶部的标题——在图30-7中，我们提交的标题为“Unsubscribed Lists”，正如我们前面讨论过的“Show Other Lists”的那段代码中所显示的。
- $list是表中每行显示的条目组成的数组。在这个例子中，它是该用户没有订阅的邮件列表组成的数组。我们将在get_unsubscribed_lists()函数中构建该数组（在这个例子中），我们将在稍后的内容中介绍这些函数。这是一个多维数组，该数组中的每一行包括4块数据。依次如下。
- $list[n][0]包含条目标识符，通常是行号。该序号为按钮给出了将要操作的行号。在我们的例子中，使用数据库中的ID——稍后将详细介绍。
- $list[n][1]包含该条目名称。为特定条目显示的文本。例如，在图30-7中，表中第一行的条目名是PHP Tipsheet。
- $list[n][2]和$list[n][3]是可选的。使用它们来表示还有其他信息。它们分别对应更多的信息文本和更多的信息ID。我们在讨论“管理函数的实现”部分的“View Mail”动作时了解使用这两个参数的例子。
- 该函数的第3、第4、第5个参数用来传递3个动作，这些动作将会显示在相应条目的按钮上。在图30-7中，3个动作按钮分别是“Information”、“Show Archive”和“Subscribe”。

通过传递动作名称，即information、show-archive和subscribe，可以获得

“Show All Lists”页面上的这3个按钮。通过display_button()函数，这些动作将被转换成带有文字的按钮，并且赋予它们相应的动作。

可以看到，在这些动作中，每一个Show动作都将以不同的方法调用display_items()函数。除了具有不同的标题和动作按钮外，这3个动作还会使用不同的函数建立要显示的条目数组。“Show All Lists”使用get_all_lists()函数。“Show Other Lists”使用get_unsubscribed_lists()函数。而“Show My Lists”将使用get_subscribed_lists()。所有这些函数的工作方式类似。这些函数都来自mlm_fns.php函数库。

下面，我们将了解get_unsubscribed_lists()函数，因为它是我们已经使用的例子。该函数的代码如程序清单30-8所示。

程序清单30-8　mlm_fns.php库中的get_unsubscribed_lists()函数——该函数用来建立用户没有订阅的邮件列表数组

```
function get_unsubscribed_lists($email) {
  $list = array();

  $query = "select lists.listid, listname, email from lists
           left join sub_lists on lists.listid = sub_lists.listid
           and email='".$email."' where email is NULL
           order by listname";

  if($conn=db_connect()) {
    $result = $conn->query($query);
    if(!$result) {
      echo '<p>Unable to get list from database.</p>';
      return false;
    }

    $num = $result->num_rows;
      for($i = 0; $i<$num; $i++) {
        $row = $result->fetch_array();
        array_push($list, array($row[0] , $row[1]));
      }
  }
  return $list;
}
```

可以看到，该函数要求传递一个邮件地址作为参数。这应该是我们正与之交互的订阅者的邮件地址。get_subscribed_lists()函数也要一个邮件地址作为参数，显然，get_all_lists()不需要任何参数。

给出订阅者邮件地址后，我们可以连接数据库并从中取出该订阅者没有订阅的所有邮件列表。可以使用LEFT JOIN来查找不匹配条目。通过遍历这个结果集，我们可以使用array_push()内置函数来逐行建立数组。

现在我们已经了解了列表的产生过程，下面，我们将介绍与这些显示相关的动作按钮。

30.6.2 查看邮件列表信息

在图30-7中，“Information”按钮将触发information动作，如下所示：

```
case 'information':
   display_information($_GET['id']);
break;
```

图30-8所示的是display_information()函数的输出结果。

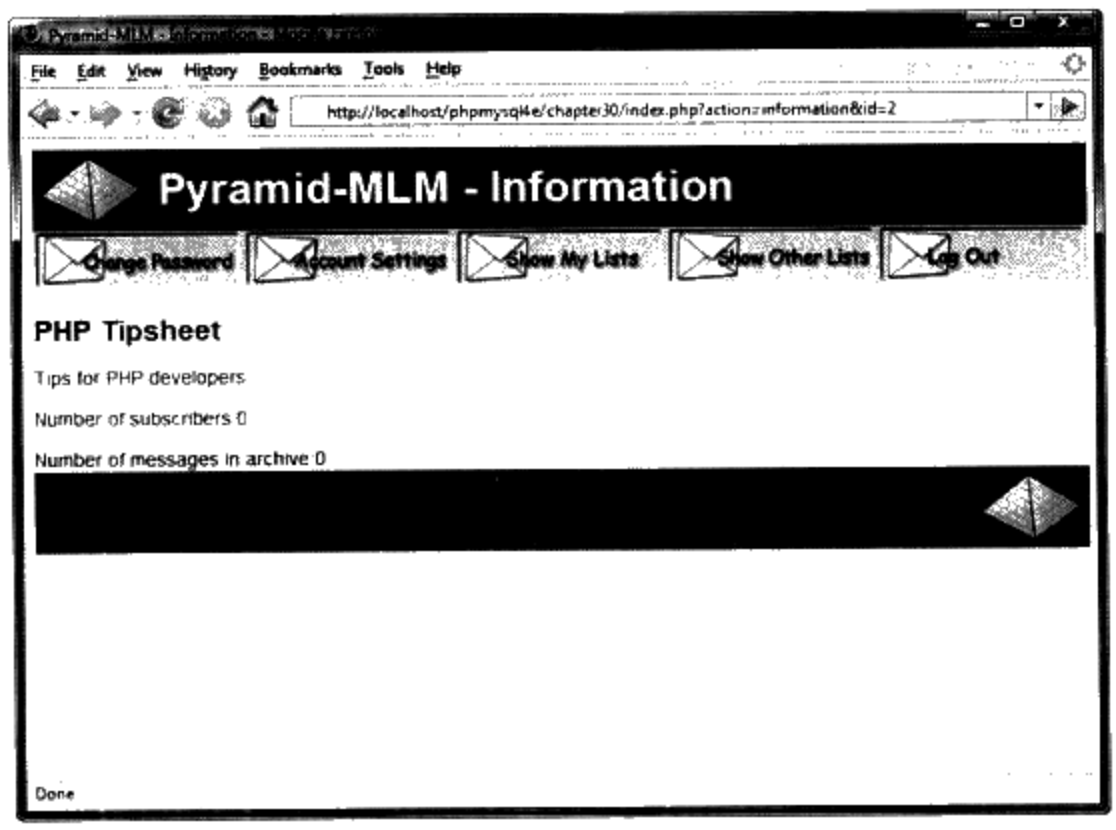

图30-8 display_information()函数显示邮件列表的介绍

该函数将显示特定邮件列表的常规信息，此外，还列出了该列表的订阅人数和将要发送给该列表上用户，以及在存档文件（稍后将进一步讨论）中可用的新闻信件数目。该函数的代码如程序清单30-9所示。

程序清单30-9 output_fns.php库中的display_information()函数——显示邮件列表信息的函数

```
// diplay stored information about each list
function display_information($listid) {
  if(!$listid) {
    return false;
  }

  $info=load_list_info($listid);
  if($info) {
    echo "<h2>".pretty($info['listname']).'</h2>
          <p>'.pretty($info['blurb']).'
          </p><p>Number of subscribers:".$info['subscribers'].'
          </p><p>Number of messages in archive:'
              .$info['archive'].'</p>";
```

```
    }
}
```

display_information()函数调用其他两个函数来协助完成存档Web任务：load_list_info()函数和pretty()函数。load_list_info()函数实际上是从数据库中获取数据。而pretty()函数只是通过去除数据中的斜杠，并且执行一些替换操作来格式化来自数据库中的数据，例如，将换行转变成HTML的行标记等。

下面，我们简单介绍load_list_info()函数。该函数保存在mlm_fns.php函数库中。代码如程序清单30-10所示。

程序清单30-10　mlm_fns.php库中的load_list_info()函数——该函数建立邮件列表信息的数组

```
function load_list_info($listid) {
  if(!$listid) {
    return false;
  }

  if(!($conn=db_connect())) {
    return false;
  }

  $query = "select listname, blurb from lists where listid =
            '".$listid."'";
  $result = $conn->query($query);

  if(!$result) {
    echo "<p>Cannot retrieve this list.</p>";
    return false;
  }

  $info = $result->fetch_assoc();

  $query = "select count(*) from sub_lists where listid =
            '".$listid."'";
  $result = $conn->query($query);

  if($result) {
    $row = $result->fetch_array();
    $info['subscribers'] = $row[0];
  }

  $query = "select count(*) from mail where listid = '".$listid."'
            and status = 'SENT'";

  $result = $conn->query($query);

  if($result) {
```

```
    $row = $result->fetch_array();
    $info['archive'] = $row[0];
  }

  return $info;
}
```

该函数将运行3个数据库查询语句，分别用来收集lists表中邮件列表的名称和简介，以及sub_lists表中订阅者的数目和由mail表发送的邮件数目。

30.6.3 查看邮件列表存档

除了可以查看邮件列表简介外，用户还可以通过点击“Show Archive”按钮查看发送到邮件列表上的所有邮件。这样将触发show-archive动作，其代码如下所示：

```
case 'show-archive':
  display_items('Archive For '.get_list_name($_GET['id']),
                get_archive($_GET['id']), 'view-html',
                'view-text', '');
break;
```

需要再次提到的是，该函数也用到了display_items()函数，它使用这个函数列出已经发送到列表上的各种条目。这些条目可以通过调用mlm_fns.php中的get_archive()函数来获得。该函数如程序清单30-11所示。

程序清单30-11 mlm_fns.php库中的get_archive()函数——该函数可以给出特定列表中的存档新闻信件数组

```
function get_archive($listid) {
  //returns an array of the archived mail for this list
  //array has rows like (mailid, subject)

  $list = array();
  $listname = get_list_name($listid);

  $query = "select mailid, subject, listid from mail
            where listid = '".$listid."' and status = 'SENT'
            order by sent";

  if($conn=db_connect()) {
    $result = $conn->query($query);
    if(!$result) {
      echo "<p>Unable to get list from database.</p>";
      return false;
    }

    $num = $result->num_rows;
```

```
    for($i = 0; $i<$num; $i++) {
      $row = $result->fetch_array();
      $arr_row = array($row[0], $row[1],
                  $listname, $listid);
      array_push($list, $arr_row);
    }
  }
  return $list;
}
```

该函数将从数据库中获得所需的信息——在这个例子中，所需的信息是已经发送出去的邮件详细信息——并且构建一个适合传递给display_items()函数的数组。

30.6.4 订阅与取消订阅

在图30-7所示的邮件列表中，每个邮件列表都有一个按钮可供用户订阅。与此类似，如果用户选择“Show My Lists”选项查看已经订阅的邮件列表，他们将看到在每个邮件列表后面都有一个“Unsubscribe”按钮。

这些按钮可以触发订阅或取消订阅的动作，这两个动作分别触发下面两段代码：

```
case 'subscribe':
   subscribe(get_email(), $_GET['id']);
   display_items('Subscribed Lists',
           get_subscribed_lists(get_email()),
           'information', 'show-archive', 'unsubscribe');
break;

case 'unsubscribe':
   unsubscribe(get_email(), $_GET['id']);
   display_items('Subscribed Lists',
           get_subscribed_lists(get_email()),
           'information', 'show-archive', 'unsubscribe');
break;
```

在每一种情况中，我们都将调用一个函数（subscribe()或unsubscribe()）并调用display_items()函数重新显示该用户订阅的邮件列表。

subscribe()和unsubscribe()函数如程序清单30-12所示。

程序清单30-12　mlm_fns.php函数库中的subscribe()和unsubscribe()函数——这些函数可以为一个用户增加或删除邮件列表的订阅

```
// subscribe this email address to this list
function subscribe($email, $listid) {
  if((!$email) || (!$listid) || (!list_exists($listid))
     || (!subscriber_exists($email))) {
    return false;
  }
```

```
  //if already subscribed exit
  if(subscribed($email, $listid)) {
    return false;
  }

  if(!($conn=db_connect())) {
    return false;
  }

  $query = 'insert into sub_lists values ('.$email.', $listid)';

  $result = $conn->query($query);
  return $result;
}

// unsubscribe this email address from this list
function unsubscribe($email, $listid) {

  if ((!$email) || (!$listid)) {
    return false;
  }

  if(!($conn=db_connect())) {
    return false;
  }

  $query = "delete from sub_lists where email = '".$email."' and
            listid =  ".$listid."'";

  $result = $conn->query($query);
  return $result;
}
```

subscribe()函数将在对应的sub_lists表中增加一行；而unsubscribe()函数则删除一行。

30.6.5 更改账户设置

点击“Account Settings”按钮时，将触发account-settings动作。该动作的代码如下所示：

```
case 'account-settings':
   display_account_form(get_email(),
   get_real_name(get_email()), get_mimetype(get_email()));
break;
```

可以看到，我们再次使用了最初用来创建账户的函数display_account_form()。但是这次，我们向这个函数传递了用户当前的资料，这些资料会显示在表单中以便用户编辑。用

户点击该表单中的提交按钮时，就会像前面讨论过的一样触发store-account动作。

30.6.6 更改密码

点击“Change Password”按钮将触发change-password动作，该动作触发如下所示的代码：

```
case 'change-password':
   display_password_form();
break;
```

display_password_form()函数（来自output_fns.php函数库）只是显示一个可供用户修改密码的表单。该表单如图30-9所示。

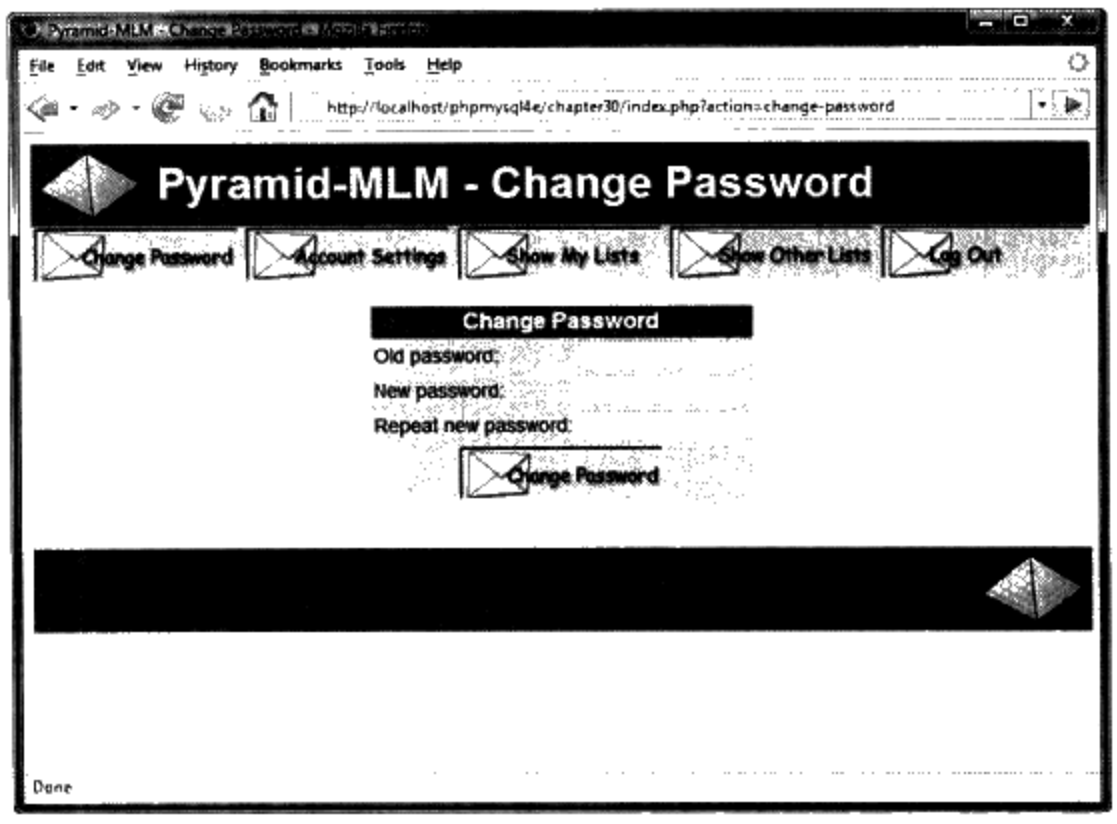

图30-9　display_password_form()函数允许用户更改密码

当用户点击表单底部的“Change Password”按钮时，将触发store-change-password动作。该动作代码如下所示：

```
case 'store-change-password':
  if(change_password(get_email(), $_POST['old_passwd'],
     $_POST['new_passwd'], $_POST['new_passwd2'])) {
    echo '<p style=\"padding-bottom: 50px\">OK: Password
          changed.</p>";
  } else {
    echo "<p style=\"padding-bottom: 50px\">Sorry, your
          password could not be changed.</p>";
    display_password_form();
  }
break;
```

可以看到，以上代码将使用change_password()函数来更改密码并向用户报告密码更改是否成功。change_password()函数可以在user_auth_fns.php函数库中找到。该函数的代码如程序清单30-13所示。

程序清单30-13 user_auth_fns.php函数库中的change_password()函数——该函数可以校验并更新用户密码

```
function change_password($email, $old_password, $new_password,
                         $new_password_conf) {
// change password for email/old_password to new_password
// return true or false

  // if the old password is right
  // change their password to new_password and return true
  // else return false
  if (login($email, $old_password)) {
    if($new_password==$new_password_conf) {
      if (!($conn = db_connect())) {
        return false;
      }

      $query = "update subscribers
                set password = sha1('".$new_password."')
                where email = '".$email."'";
      $result = $conn->query($query);
      return $result;
    } else {
      echo "<p>Your passwords do not match.</p>";
    }
  } else {
    echo '<p>Your old password is incorrect.</p>';
  }

  return false; // old password was wrong
}
```

该函数与我们前面所看到的密码设置和修改函数类似。它比较用户输入的两个密码，判断是否一致，如果一致，则在数据库中更改该用户的密码。

30.6.7 登出

当用户点击“Log Out”按钮时，将触发log-out动作。在主脚本中，该动作所执行的代码实际上主要在预处理阶段，如下所示：

```
if($action == 'log-out') {
  unset($action);
```

```
    $_SESSION=array();
    session_destroy();
}
```

以上代码段将清除会话变量并结束该会话。请注意，它还将清除action变量——这意味着进入主case语句而不发生任何动作，这将触发下面的代码：

```
default:
    if(!check_logged_in()) {
        display_login_form($action);
    }
break;
```

这将使另一个用户登录，或者当前用户以其他身份登录。

30.7　管理功能的实现

如果某人以管理员身份登录，将看到一些额外的菜单选项，如图30-10所示。

管理员身份独有的选项是“Create List”（创建新的邮件列表）、“Create Mail”（创建新的新闻信件）和“View Mail”（查看并发送那些已经创建但还没发出去的新闻信件）。下面，我们将依次讨论它们。

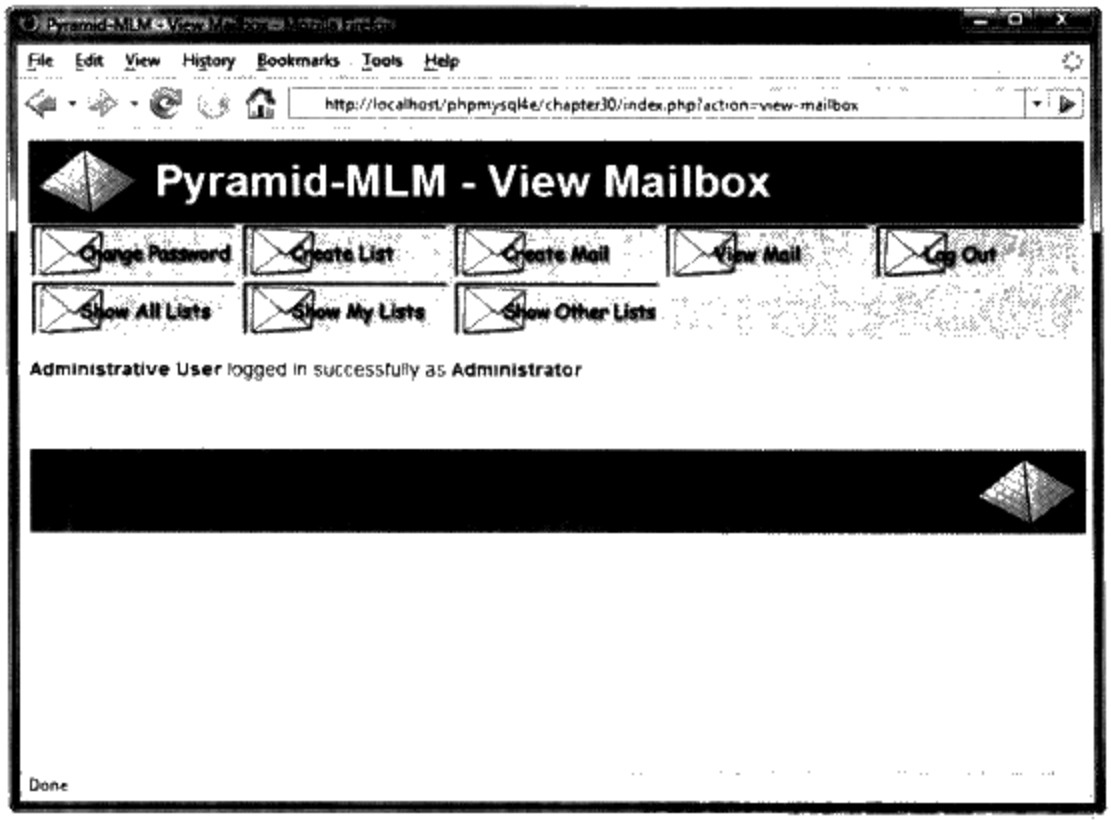

图30-10　管理员菜单选项允许创建和维护邮件列表

30.7.1　创建新的邮件列表

如果管理员点击“Create List”按钮来建立新的邮件列表，将触发create-list动作，该动作与如下所示代码相关：

```
case 'create-list':
```

```
    display_list_form(get_email());
break;
```

display_list_form()函数将显示一个可供管理员输入新邮件列表详细信息的表单。该函数保存在output_fns.php函数库中。它只是用来输出HTML，因此在这里，我们不深入讨论它。该函数的输出如图30-11所示。

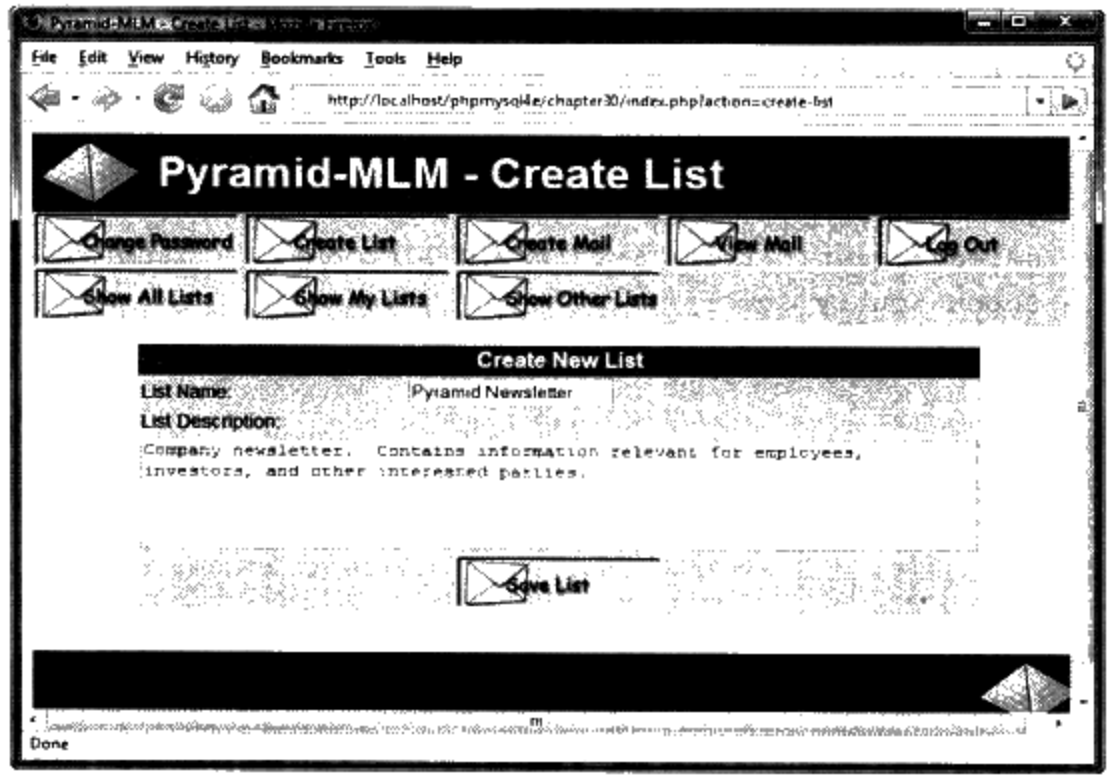

图30-11　“Create List”选项要求管理员为新的邮件列表输入名称和描述（或简介）

当管理员点击“Save List”按钮时，将触发store-list动作，而该动作将触发index.php中的如下所示代码：

```
case 'store-list':
   if(store_list($_SESSION['admin_user'], $_POST)) {
      echo '<p style=\"padding-bottom: 50px\">New list added.</p>";
      display_items('All Lists', get_all_lists(), 'information',
                    'show archive','');
   } else {
      echo '<p style=\"padding-bottom: 50px\">List could not be
            stored. Please try again.</p>";
   }
break
```

可以看到，以上代码将保存新的列表详细信息，并显示新的列表。列表详细信息是通过store_list()函数来保存的。该函数的代码如程序清单30-14所示。

程序清单30-14　mlm_fns.php函数库的store_list()函数——该函数在数据库中插入一个新的邮件列表

```
function store_list($admin_user, $details) {
  if (!filled_out($details)) {
```

```
    echo "<p>All fields must be filled in. Try again.</p>";
    return false;
  } else {
    if(!check_admin_user($admin_user)) {
      return false;
      // how did this function get called by somebody not logged in as admin?
    }

    if(!($conn=db_connect())) {
      return false;
    }

    $query = "select count(*)from lists where listname = '".$details['name']."'";
    $result = $conn->query($query);
    $row = $result->fetch_array();

    if($row[0] > 0) {
      echo '<p>Sorry, there is already a list with this name.</p>';
      return false;
    }

    $query = "insert into lists values (NULL,
                                        '".$details['name']."',
                                        '".$details['blurb']."')";
      $result = $conn->query($query);
      return $result;
    }
  }
```

该函数在写入数据库之前进行一些有效性检查：它将检查是否已经提供了所有需要的详细信息、当前的用户是否是管理员、以及该列表的名称是否是唯一的。如果所有检查都正确，该列表将被添加到数据库中的lists表。

30.7.2　上载新的新闻信件

最后，我们再来看看该应用程序的关键部分：上载并发送新闻信件到邮件列表。

当管理员点击“Create Mail”按钮时，将触发create-mail动作，如下所示：

```
case  create-mail :
   display_mail_form(get_email());
break;
```

管理员将看到如图30-12所示的表单。

请记住，对于该程序，我们假设管理员已经在脱机环境中创建了HTML格式和文本格式的新闻信件，而且在发送该新闻信件之前，需要上载这两个版本的文件。我们选择用这种实现方法是为了让管理员可以使用他们最喜欢的软件创建新闻信件。这使得该程序更容易使用。

该表单有几处地方需要由管理员填写。在表单的最上方是一个下拉列表框，可供管理员选择邮件列表。管理员还必须填写新闻信件的主题——这就是最终邮件的主题行。

所有其他的表单域都是文件上载域，可以通过它们右边的“Browse”按钮分辨出来。要发送新闻信件，管理员必须给出该邮件的文本和HTML 两种版本（可以根据需要进行修改）。此外，表单中还有很多可选的图像域，在这里，管理员可以上载任何嵌入到HTML的图像。这些文件必须明确指定并分别上载。

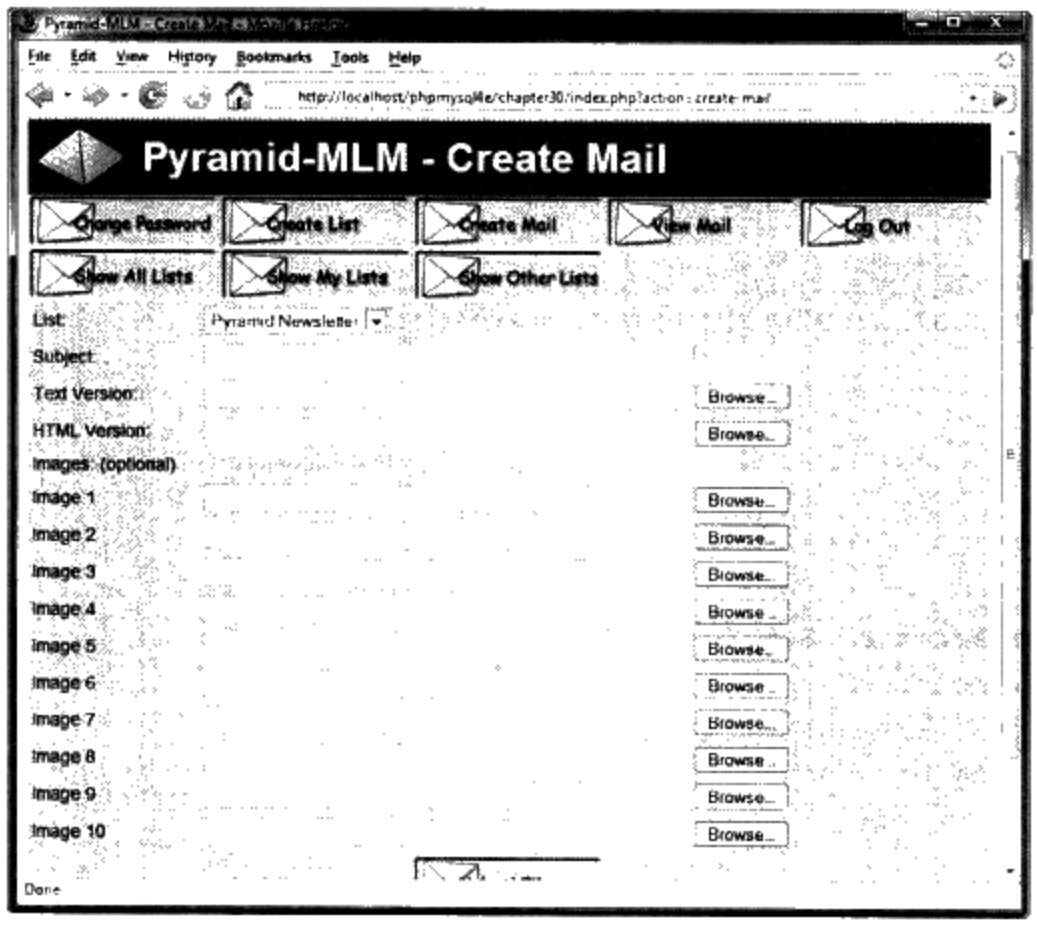

图30-12　“Create Mail”选项提供了可供管理员上载新闻信件文件的界面

该表单与普通的上载表单相似，只是在这个例子中，我们使用它来上载多个文件。这就必然使得在表单语法与我们在另一端处理上载文件的方法上会存在一些差异。

display_mail_form()函数的代码如程序清单30-15所示。

在这里，需要注意的是我们要在一系列文本输入框中输入上载文件的名称，以及相应的文件类型。这些文件名称在程序中的命名为userfile[0]到userfile[n]。事实上，我们使用与处理复选框一样的方法来处理这些表单域，并使用数组命名惯例对其命名。

程序清单30-15　output_fns.php函数库中的display_mail_form()函数——该函数将显示文件上载表单

```
function display_mail_form($email, $listid=0) {
  // display html form for uploading a new message
  global $table_width;
  $list=get_all_lists();
  $lists=sizeof($list);
?>
```

```
<table cellpadding='4' cellspacing='0' border='0'
  width='<?php echo $table_width; ?>'>
<form enctype="multipart/form-data' action="upload.php" method='post">
<tr>
  <td bgcolor="#cccccc">List:</td>
  <td bgcolor="#cccccc">
    <select name="list">
    <?php
    for($i=0; $i<$lists; $i++) {
      echo "<option value=\"".$list[$i][0].'\"';

      if ($listid== $list[$i][0]) {
        echo " selected';
      }

      echo ">".$list[$i][1]."</option>\n";
    }
    ?>
    </select>
  </td>
</tr>
<tr>
  <td bgcolor='#cccccc">Subject:</td>
  <td bgcolor='#cccccc">
    <input type="text" name='subject"
     value='<?php echo $subject; ?>"
     size="60' /></td>
</tr>
<tr>
  <td bgcolor='#cccccc">Text Version:</td>
  <td bgcolor="#cccccc">
     <input type='file' name="userfile[0]" size="60"/></td>
  </tr>
  <tr><td bgcolor='#cccccc">HTML Version:</td>
  <td bgcolor='#cccccc">
    <input type='file' name='userfile[1]" size='60" /></td>
  </tr>
  <tr><td bgcolor="#cccccc' colspan='2'>Images: (optional)

<?php
  $max_images=10;
  for($i=0; $i<10; $i++) {
    echo "<tr><td bgcolor=\"#cccccc\'>Image ".($i+1)." </td>
          <td bgcolor=\"#cccccc\"><input type=\"file\"
              name=\"userfile[".($i+2).']\" size=\"60\"/></td>
          </tr>";
  }
?>
```

```
  <tr><td colspan="2" bgcolor="#cccccc" align='center'>
  <input type='hidden' name="max_images"
        value='<?php echo $max_images; ?>'>
  <input type="hidden' name="listid"
        value="<?php echo $listid; ?>'>
  <?php display_form_button('upload-files'); ?>
  </td>
  </form>
  </tr>
  </table>
<?php
}
```

如果想通过PHP脚本上载多个文件并且以数组方式轻松地处理它们，必须遵循这个命名惯例。

在处理该表单的脚本中，我们实际上以3个数组结束。下面，让我们来看一下这个脚本。

30.7.3 多文件上载的处理

文件上载代码是一个单独的文件upload.php，该文件的完整代码如程序清单30-16所示。

程序清单30-16 upload.php——上载新闻信件所需的所有文件脚本

```
<?php
  // this functionality is in a separate file to allow us to be
  // more paranoid with it
  // if anything goes wrong, we will exit

$max_size = 50000;

include ( include_fns.php');
session_start();

// only admin users can upload files
if(!check_admin_user()) {
  echo "<p>You do not seem to be authorized to use this page.</p>";
  exit;
}

// set up the admin toolbar buttons
$buttons = array();
$buttons[0]   'change-password';
$buttons[1]   'create-list';
$buttons[2]    create-mail';
$buttons[3]   'view-mail';
$buttons[4]   'log-out';
$buttons[5]   'show-all-lists';
$buttons[6]   'show-my-lists';
```

```
$buttons[7] = 'show-other-lists';

do_html_header('Pyramid-MLM - Upload Files');

display_toolbar($buttons);

// check that the page is being called with the required data
if((!$_FILES['userfile']['name'][0]) ||
   (!$_FILES['userfile']['name'][1]) ||
   (!$_POST['subject']||!$_POST['list'])) {
    echo "<p>Problem: You did not fill out the form fully.
          The images are the only optional fields.
          Each message needs a subject, text version
          and an HTML version.</p>";
    do_html_footer();
    exit;
}

$list = $_POST['list'];
$subject = $_POST['subject'];

if(!($conn=db_connect())) {
   echo "<p>Could not connect to db.</p>";
   do_html_footer();
   exit;
}

// add mail details to the DB
$query = "insert into mail values (NULL,
                    '".$_SESSION['admin_user']."',
                    '".$subject."',
                     ".$list.",
                    'STORED', NULL, NULL)";

$result = $conn->query($query);
if(!$result) {
  do_html_footer();
  exit;
}

//get the id MySQL assigned to this mail
$mailid = $conn->insert_id;

if(!$mailid) {
  do_html_footer();
  exit;
}
```

```
// creating directory will fail if this is not the first message archived
// that's ok
@mkdir( archive/'.$list, 0700);

// it is a problem if creating the specific directory for this mail fails
if(!mkdir('archive/'.$list. /'.$mailid, 0700)) {
  do_html_footer();
  exit;
}

// iterate through the array of uploaded files
$i = 0;
while (($_FILES['userfile']['name'][$i]) &&
       ($_FILES['userfile']['name'][$i] != 'none')) {
  echo "<p>Uploading ".$_FILES['userfile']['name'][$i].' - '.
       $_FILES['userfile']['size'][$i]." bytes.</p>";

  if ($_FILES['userfile']['size'][$i]==0) {
    echo "<p>Problem: ".$_FILES['userfile']['name'][$i].
         " is zero length";
    $i++;
    continue;
  }

  if ($_FILES['userfile']['size'][$i]>$max_size) {
    echo "<p>Problem: ".$_FILES['userfile']['name'][$i]." is over "
         .$max_size.' bytes';
    $i++;
    continue;
  }

  // we would like to check that the uploaded image is an image
  // if getimagesize() can work out its size, it probably is.
  if(($i>1) && (!getimagesize($_FILES['userfile']['tmp_name'][$i]))) {
    echo "<p>Problem: ".$_FILES['userfile']['name'][$i].
         " is corrupt, or not a gif, jpeg or png.</p>";
    $i++;
    continue;
  }

  // file 0 (the text message) and file 1 (the html message) are special cases
  if($i==0) {
    $destination = 'archive/'.$list.'/'.$mailid.'/text.txt';
  } else if($i == 1) {
    $destination = "archive/".$list."/".$mailid."/index.html";
  } else {
    $destination = "archive/".$list."/".$mailid."/"
```

```
                    .$_FILES['userfile']['name'][$i];
    $query = "insert into images values ('".$mailid."',
                    '".$_FILES['userfile']['name'][$i]."',
                    '".$_FILES['userfile']['type'][$i]."')";

    $result = $conn->query($query);
  }

  if (!is_uploaded_file($_FILES['userfile']['tmp_name'][$i])) {
    // possible file upload attack detected
    echo '<p>Something funny happening with '
          .$_FILES['userfile']['name'].", not uploading.";
    do_html_footer();
    exit;
  }

  move_uploaded_file($_FILES['userfile']['tmp_name'][$i],
                    $destination);
  $i++;
}

display_preview_button($list, $mailid, 'preview html');
display_preview_button($list, $mailid, 'preview-text');
display_button('send', "&id=$mailid");
  echo "<p style=\"padding-bottom: 50px\"> </p>";
  do_html_footer();
?>
```

下面，让我们来分析一下上载文件的步骤。首先，开始一个会话并检验用户是否以管理员身份登录——我们不希望其他的用户能够上载文件。

严格地讲，我们可能还需要对list和mailid变量进行检验，检查这些变量是否包含不希望的字符，在这里，我们只是从简化代码的角度考虑，忽略了该功能。

接下来，我们建立并发送页面的标题，并检验该表单已经正确填写。这非常重要，因为对用户来说，它是一个相当复杂的表单。

接着，需要在数据库中为该邮件创建一条记录，并且在存档目录中创建一个用来保存邮件的目录。

接下来就是脚本程序的主要部分，该部分将检查并转移上载的文件。在上载多个文件时，该部分还有一点不同。这里我们需要处理4个数组。这些数组分别是$_FILES['userfile']['name']、$_FILES['userfile']['tmp_name']、$_FILES['userfile']['size']和$_FILES['userfile']['type']。它们对应单文件上载时与之命名相似的变量，只是在这里它们都是数组。表单中第一个文件的详细内容在$_FILES['userfile']['tmp_name'][0]、$_FILES['userfile']['name'][0]、

$_FILES['userfile']['size'][0]和$_FILES['userfile'] ['type'][0]中。

针对这3个数组，我们将进行常规的安全检查并将文件转移到存档中。

最后，我们为管理员提供几个按钮，在发送新闻信件前，这些按钮可以用来预览已经上传的新闻信件文件。图30-13所示的是upload.php的输出。

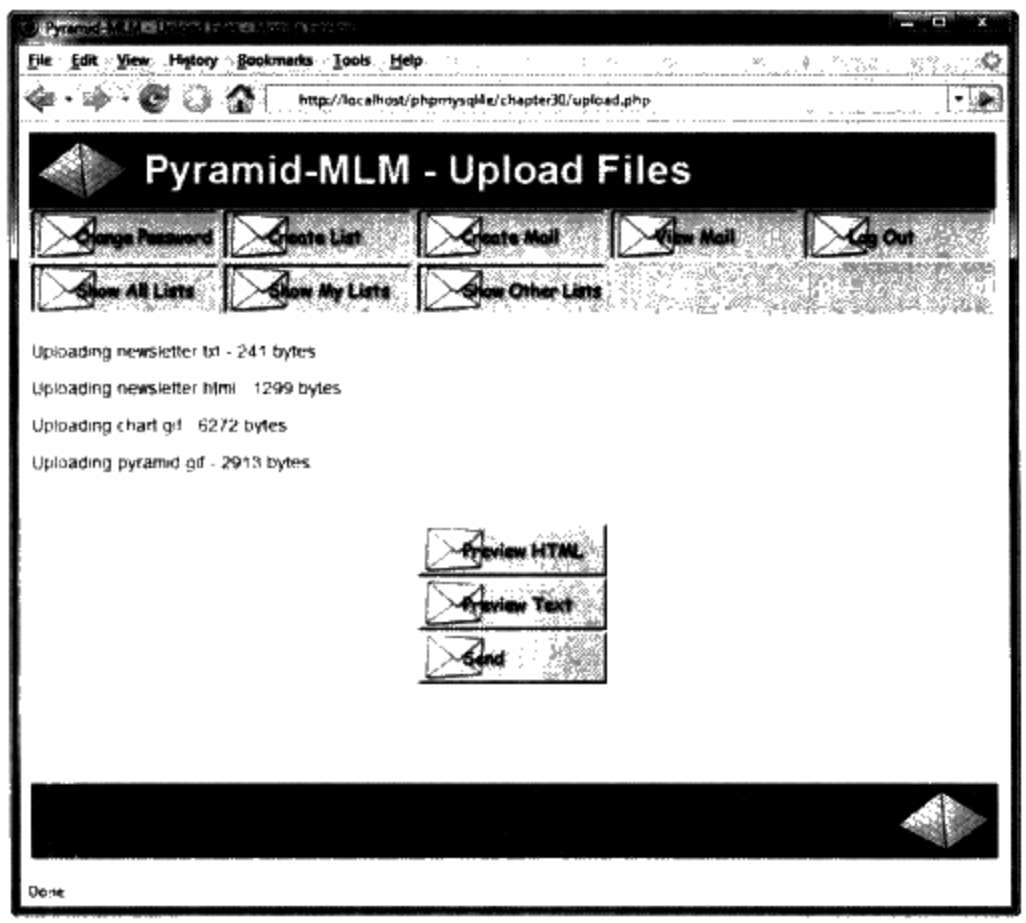

图30-13　上传脚本将报告所上载的文件及其大小

30.7.4　预览新闻信件

在发送一个新闻信件前，管理员可以通过两种方法来预览新闻信件。如果希望上载后立即预览，可以从上载页面上使用预览功能。如果管理员希望预览邮件并在以后再发送，可以通过点击“View Mail”按钮，该按钮将显示系统中所有未发送的新闻信件。“View Mail”按钮将触发view-mail动作，触发如下代码：

```
case 'view-mail':
    display_items('Unsent Mail', get_unsent_mail(get_email()),
                  'preview-html', 'preview text', 'send');
break;
```

可以看到，在preview-html、preview-text和send动作中，我们再次使用了display_items()函数。

请注意，“Preview”按钮实际上并不触发任何动作，而是直接连接到存档中的新闻信件。如果回顾程序清单30-7及程序清单30-16，可以看到，我们使用了display_preview_button()函数来创建这些按钮，而不是使用display_button()函数。

display_button()函数将创建一个指向特定脚本的图像链接，该函数必须给出一个GET参数；而display_preview_button()函数给出一个指向存档的普通链接。该链接将会显示在一个新的弹出窗口中，这可以通过在HTML的anchor标记中设置target = new属性来实现。我们将看到如图30-14所示的HTML版本的新闻信件。

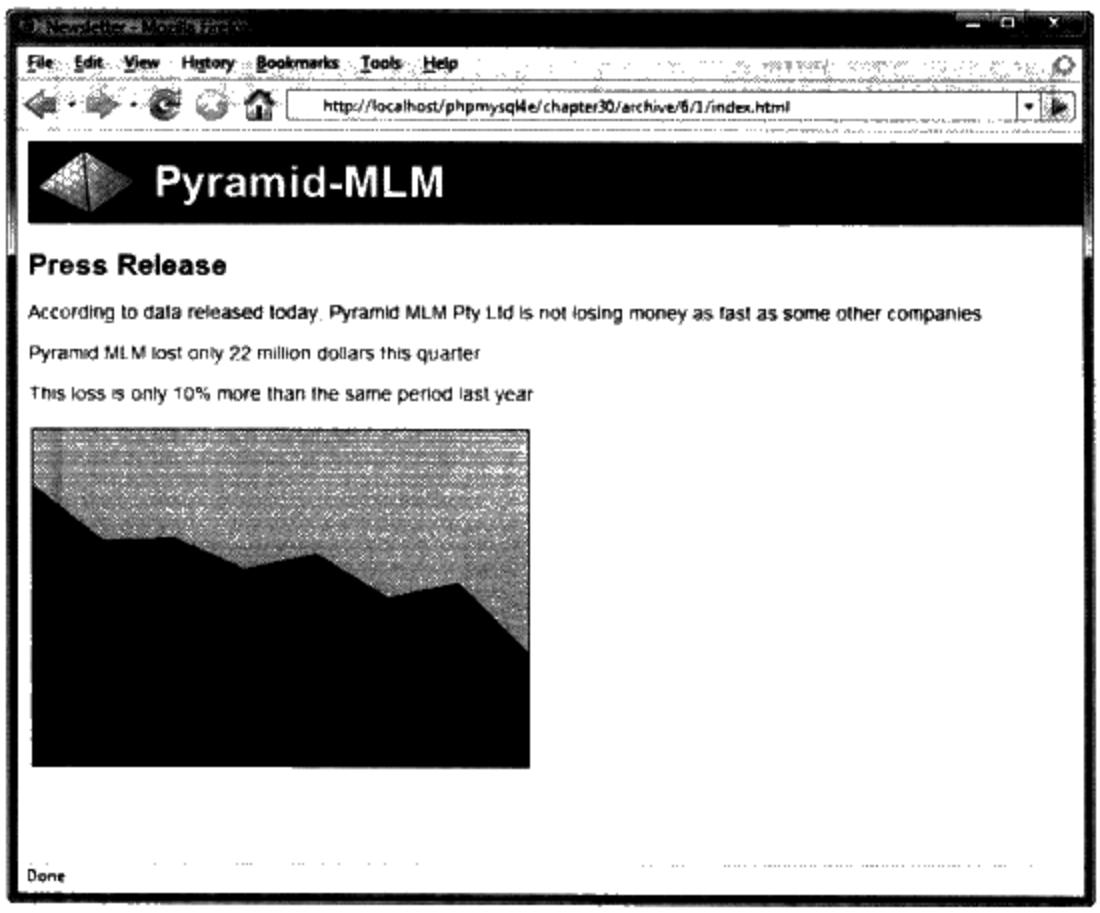

图30-14　对一个内嵌有图像的HTML格式新闻信件进行预览

30.7.5　发送邮件

点击新闻信件的发送按钮将触发发送动作，该动作将触发如下代码：

```
case 'send':
   send($_GET  id'], $_SESSION['admin_user']);
break;
```

该代码将调用send()函数，该函数可以在mlm_fns.php函数库中找到。这是一个相当长的函数。它也是我们使用Mail_mime类的地方。

该函数的代码如程序清单30-17所示。

程序清单30-17　mlm_fns.php函数库中的send()函数——该函数最终将发送一个新闻信件

```
// create the message from the stored DB entries and files
// send test messages to the administrator, or real messages to the whole list
function send($mailid, $admin_user) {
  if(!check_admin_user($admin_user)) {
    return false;
  }
```

```
if(!($info = load_mail_info($mailid))) {
  echo "<p>Cannot load list information for message ".$mailid."</p>";
  return false;
}

$subject = $info['subject'];
$listid = $info['listid'];
$status = $info['status'];
$sent = $info['sent'];

$from_name = 'Pyramid MLM';

$from_address = 'return@address';
$query = "select email from sub_lists where listid = '".$listid."'";

$conn = db_connect();
$result = $conn->query($query);
if (!$result) {
  echo $query;
  return false;
} else if ($result->num_rows==0) {
  echo "<p>There is nobody subscribed to list number ".$listid."</p>";
  return false;
}

// include PEAR mail classes
include('Mail.php');
include('Mail/mime.php');

// instantiate MIME class and pass it the carriage return/line feed
// character used on this system
$message = new Mail_mime("\r\n");

// read in the text version of the newsletter
$textfilename = "archive/".$listid."/".$mailid."/text.txt";
$tfp = fopen($textfilename, "r");
$text = fread($tfp, filesize($textfilename));
fclose($tfp);

// read in the HTML version of the newsletter
$htmlfilename = "archive/".$listid."/".$mailid."/index.html";
$hfp = fopen($htmlfilename, "r");
$html = fread($hfp, filesize($htmlfilename));
fclose($hfp);

// add HTML and text to the mimemail object
$message->setTXTBody($text);
```

```
$message->setHTMLBody($html);

// get the list of images that relate to this message
$query = "select path, mimetype from images where
          mailid = ".$mailid." ";
$result = $conn->query($query);
if(!$result) {
  echo "<p>Unable to get image list from database.</p>";
  return false;
}

$num = $result->num_rows;
for($i = 0; $i<$num; $i++) {
  //load each image from disk
  $row = $result->fetch_array();
  $imgfilename = 'archive/$listid/$mailid/'.$row[0];
  $imgtype = $row[1];
   // add each image to the object
  $message->addHTMLImage($imgfilename, $imgtype,
            $imgfilename, true);
}

// create message body
$body = $message->get();

// create message headers
$from = '"'.get_real_name($admin_user).'" <'.$admin_user.'>';
$hdrarray = array(
                'From'    => $from,
                'Subject' => $subject);

$hdrs = $message->headers($hdrarray);

// create the actual sending object
$sender =& Mail::factory('mail');

if($status == 'STORED') {

  // send the HTML message to the administrator
  $sender->send($admin_user, $hdrs, $body);

  // send the plain text version of the message to administrator
  mail($admin_user, $subject, $text,
      'From: '.get_real_name($admin_user).'" <'.$admin_user.'>');

  echo 'Mail sent to '.$admin_user."";
```

```
    // mark newsletter as tested
    $query = "update mail set status =  TESTED' where
              mailid = '".$mailid." ";
    $result = $conn->query($query);

    echo "<p>Press send again to send mail to whole list.
          <div align=\"center\">";
    display_button('send', '&id='.$mailid);
    echo "</div></p>";

  } else if($status == 'TESTED' ) {
    //send to whole list

    $query = "select subscribers.realname, sub_lists.email,
                     subscribers.mimetype
              from sub_lists, subscribers
              where listid = $listid and
                    sub_lists.email = subscribers.email";

    $result = $conn->query($query);
    if(!$result) {
      echo "<p>Error getting subscriber list</p>";
    }
    $count = 0;
    // for each subscriber
    while ($subscriber = $result->fetch_row()) {
      if($subscriber[2] == 'H') {
        //send HTML version to people who want it
        $sender->send($subscriber[1], $hdrs, $body);
      } else {
        //send text version to people who don't want HTML mail
        mail($subscriber[1], $subject, $text,
                        'From: "'.get_real_name($admin_user).'"
                        <'.$admin_user.'>');
      }
      $count++;
    }

    $query = "update mail set status = 'SENT', sent = now()
              where mailid = '".$mailid." ";
    $result = $conn->query($query);
    echo "<p>A total of $count messages were sent.</p>";
  } else if ($status == 'SENT') {
    echo "<p>This mail has already been sent.</p>";
  }
}
```

该函数将完成几项不同的操作。在将新闻信件发送出去之前，该函数测试性地将它发送给管理员。它通过记录数据库中邮件的状态来保存这些信息。当上载程序脚本程序上载一封邮件时，该函数将该邮件的初始状态设置为“STORED”。

如果send()函数发现某邮件状态为“STORED”，则将它更改为“TESTED”并将它发送给管理员。“TESTED”状态表示该新闻信件已经被发送并经过管理员测试了。如果send()函数遇到某邮件状态为“TESTED”，则把它变成“SENT”并将它发送给列表上所有用户。这意味着每封邮件必须经过两次发送：一次是测试模式；一次是实际模式。

该函数还可以发送两种不同的邮件：通过PHP的mail()函数发送的文本版本，和用Mail_mime类发送的HTML版本。在本书中，我们已经在多处用到了mail()函数，因此在这里我们了解一下如何使用Mail_mime类。但是，我们并不会全面地讨论该类，只解释如何在这个典型的程序中使用它。

首先，我们将类文件包含进来并创建Mail_mime类的一个实例：

```
// include PEAR mail classes
include( Mail.php );
include( Mail/mime.php );

// instantiate MIME class and pass it the carriage return/line feed
// character used on this system
$message = new Mail_mime("\r\n");
```

请注意，我们在这里包含了两个类文件。我们还将使用PEAR软件包中的常规Mail类，正是这个类真正完成了邮件发送功能。这个类随同PEAR软件包一起安装。

Mail_mime类可以用来创建将要发送的MIME格式消息。

接下来，我们将读入邮件的文本和HTML版本，并且将它们添加到Mail_mime类：

```
// read in the text version of the newsletter
$textfilename = "archive/".$listid."/".$mailid."/text.txt";
$tfp = fopen($textfilename, "r");
$text = fread($tfp, filesize($textfilename));
fclose($tfp);

// read in the HTML version of the newsletter
$htmlfilename = "archive/".$listid."/".$mailid."/index.html";
$hfp = fopen($htmlfilename, "r");
$html = fread($hfp, filesize($htmlfilename));
fclose($hfp);

// add HTML and text to the mimemail object
$message->setTXTBody($text);
$message->setHTMLBody($html);
```

完成邮件读入后，该脚本循环地从数据库中载入每一个图片细节，并将其添加到需要发送的邮件中：

```
$num = $result->num_rows;
```

```
for($i = 0; $i<$num; $i++) {
  //load each image from disk
  $row = $result->fetch_array();
  $imgfilename = 'archive/'.$listid.'/'.$mailid.'/'.$row[0];
  $imgtype = $row[1];
    // add each image to the object
  $message->addHTMLImage($imgfilename, $imgtype, $imgfilename, true);
}
```

在这里，传递给addHTMLImage()函数的参数分别是：图像文件的名称（或者我们可以传递图像数据）、图像的MIME类型、文件名称，以及用来标记第一个参数是图像文件名称而不是文件数据的true（如果想传递原始图像数据，可以传递数据、MIME类型、空参数和false）。这些参数有些繁琐。

在这一步，我们需要创建邮件正文。在设置邮件标题前，必须先创建邮件正文。我们按如下代码创建邮件正文：

```
// create message body
$body = $message->get();
```

接下来，可以调用Mail_mime类的headers()函数创建该邮件标题：

```
// create message headers
$from = '" .get_real_name($admin_user). " < .$admin_user. > ';
$hdrarray = array(
            'From'    => $from,
            'Subject' => $subject);
```

最后，在设置完邮件后，就可以发送它了。在发送该邮件前，还需要初始化PEAR Mail类，并且将其传递给我们所创建的邮件。初始化该类的代码如下所示：

```
// create the actual sending object
$sender =& Mail::factory('mail');
```

（在这里，'mail'参数只是告诉Mail类使用PHP的mail()函数来发送邮件。很明显，也可以使用'sendmail'或'smtp'作为该参数值。）

接下来，我们将该邮件发送给每一个订阅者，通过检索该列表的每一个订阅者，并且根据用户的MIME类型，使用Mail的send()函数或常规的mail()函数来发送邮件：

```
if($subscriber[2]=='H') {
  //send HTML version to people who want it
  $sender->send($subscriber[1], $hdrs, $body);
} else {
  //send text version to people who don't want HTML mail
  mail($subscriber[1], $subject, $text,
                  'From: "'.get_real_name($admin_user).'"
                  <'.$admin_user.'>');
}
```

$sender->send()函数的第一个参数应该是用户的电子邮件地址，第二个参数是邮件标

题，而第三个参数是邮件正文。

这样就完成了邮件列表应用程序的构建。

30.8 扩展这个项目

就像这些常规项目一样，通常有许多种扩展功能的方法。我们可能希望做到以下几点。

- 与订阅者确认成员关系，这样那些没有得到许可的人不能订阅。通常，可以通过向这些账户发送邮件并且删除那些没有回复的用户来实现。该方法也可以从数据库中清除任何不正确的邮件地址。
- 给予管理员批准或拒绝申请订阅列表的用户的权限。
- 增加开放列表功能以使任何成员都向列表发送电子邮件。
- 对于某特定的邮件列表仅注册成员可以看到存档。
- 允许用户搜索满足特定条件的列表。例如，用户可能对高尔夫球新闻信件感兴趣。一旦用户数量超过一定的大小，搜索指定的邮件的功能将会很有用。
- 如果希望创建一个大型的邮件列表，PHP应用程序可能不是发送这些邮件的理想选择。一个实用的邮件列表管理器（例如，ezmlm）的效率会比在PHP中调用`mail()`函数高得多。当然，我们还可以使用PHP来构建前台程序，而使用ezmlm来处理繁重的工作。

30.9 下一章

在下一章中，我们实现一个Web论坛程序，该论坛允许用户联机讨论，这些讨论可以是按照主题和对话内容进行组织的。