第29章　创建一个基于Web的电子邮件服务系统

当今，越来越多的站点希望为用户提供一个基于Web的电子邮件系统。本章将介绍如何使用PHP IMAP库为一个已有的邮件服务器实现Web界面。可以用它通过Web页面来检查自己已有的邮箱，或者将它扩展，使其像基于Web的大规模电子邮件系统一样支持多用户，例如，Gmail、Yahoo! Mail和Hotmail。

在这个项目中，我们将建立一个电子邮件客户端，Warm Mail，它允许用户：

- 连接到POP3或IMAP邮件服务器上的账户
- 阅读邮件
- 发送邮件
- 回复邮件消息
- 转发邮件消息
- 从用户账户中删除邮件

29.1　解决方案的组成

为了使用户可以阅读邮件，需要找到一种方法连接到邮件服务器。通常，这种机制与连接Web服务器的机制不同。我们需要一种方法来与用户的邮箱进行交互，检查收到的邮件消息，并单独处理每一条消息。

29.1.1　电子邮件协议：POP3和IMAP

邮件服务器需要支持两个主要的协议：POP3和IMAP。通过这两个协议，用户能够查看邮箱。如果可能，我们应该同时支持二者。POP3表示邮局协议（Post Office Protocol，POP）第三版，IMAP表示互联网消息访问协议（Internet Message Access Protocol，IMAP）。

二者主要的不同是：POP3（也是较为常见的情况）是为了那些仅从服务器下载或删除邮件而短时间连接网络的人所设计的。而IMAP出于在线使用的目的，它可以使用户与邮件进行交互，而且邮件也可以永久保存在远程服务器上。在这里，我们将不会使用IMAP的一些更高级的特性。

如果对这两个协议之间的差别感兴趣，可以参考RFC标准（RFC 1939版本3和RFC 3501 IMAP版本4修改稿1）。可以在如下站点找到关于二者相比较的优秀文章：

http://www.imap.org/papers/imap.vs.pop.brief.html。

这两个协议都不是为发送邮件设计的，因此，我们必须使用简单电子邮件传输协议（Simple Mail Transfer Protocol，SMTP）来发送邮件，在本书的前面内容中，当我们介绍如何在PHP中使用`mail()`函数时，我们介绍了这个协议。该协议在RFC 821中进行了描述。

29.1.2　PHP对POP3和IMAP的支持

虽然PHP对IMAP和POP3有着不错的支持，但它都是通过IMAP函数库所提供的功能来实现的。为了使用本章所提供的代码，需要安装IMAP库。可以通过查看phpinfo()函数的输出来判断是否已经安装该库。

如果使用的是Linux或UNIX并且没有安装它，则需要下载该库。可以通过FTP从如下站点得到最新的版本：ftp://ftp.cac.washington.edu/imap/。

在UNIX平台下，你可以下载源代码并根据操作系统对其进行编译。

你必须在系统包含目录中创建一个保存IMAP文件的目录，例如imap（不要只是将这些文件复制到这个基本的包含目录中，因为这可能会导致一些冲突）。在新目录中，创建imap/lib和imap/include两个子目录。将安装中涉及的所有*.h文件复制到imap/include/目录。当完成了编译，将创建一个c-client.a文件。将其重命名为libc-client.a并且复制到imap/lib目录。

然后必须运行PHP的配置脚本，将--with-imap=*dirname*指令添加到所使用的任何其他参数，（这里，*dirname*是所创建的目录名称）并重新编译PHP。

要在Windows下使用IMAP扩展，可以打开php.ini配置文件，注释掉如下代码行：

```
extension=php_imap.dll
```

然后再重新启动Web服务器。

通过运行phpinfo()函数，可以确认IMAP扩展是否已经安装。函数输出将显示关于IMAP的内容。

需要注意的一件有趣的事情是：虽然这些函数都叫做IMAP函数，但它们同样能够支持POP3和网络新闻传输协议（Network News Transfer Protocol，NNTP）。在这个例子中，我们准备将它们用于IMAP和POP3，但Warm Mail应用程序可以很容易扩展，因此除了邮件客户端外，它还可以使用NNTP并且成为一个新闻阅读程序。

该函数库中包含一些函数，但为了实现该程序的功能，我们仅使用其中少数几个。具体使用函数时，我们会详细介绍这些函数，但是要注意有很多这样的函数。针对具体需求的不同，或者想为程序增加特别的特性，请查看相关资料。

仅使用部分的内置函数，就可以建立一个相当有用的邮件应用程序。这意味着只需要研究部分文档。在本章中，我们将使用的IMAP函数包括：

- imap_open()
- imap_close()
- imap_headers()
- imap_header()
- imap_fetchheader()
- imap_body()
- imap_delete()
- imap_expunge()

为了让用户能够读取邮件，需要获得用户邮件服务器和用户账户的详细信息。为了不必每次都从用户那里获得这些详细信息，我们将建立一个有用户名和密码的数据库以便存储这些资料。

通常，人们会有多个电子邮件账户（比如一个用于家庭，另一个用于工作），我们应该允许他们连接其中任何一个。因此，应该允许他们在数据库中有多条记录。

除了让用户能够发送新邮件外，还应该让用户能够阅读、回复、转发和删除已存在的邮件。我们可以使用IMAP或POP3完成所有的阅读功能，使用SMTP的`mail()`函数完成所有发送功能。

下面，让我们看看如何实现这些功能。

29.2　解决方案概述

这个基于Web的系统的大体流程图与其他电子邮件客户端没有明显差别。该系统流程和模块如图29-1所示。

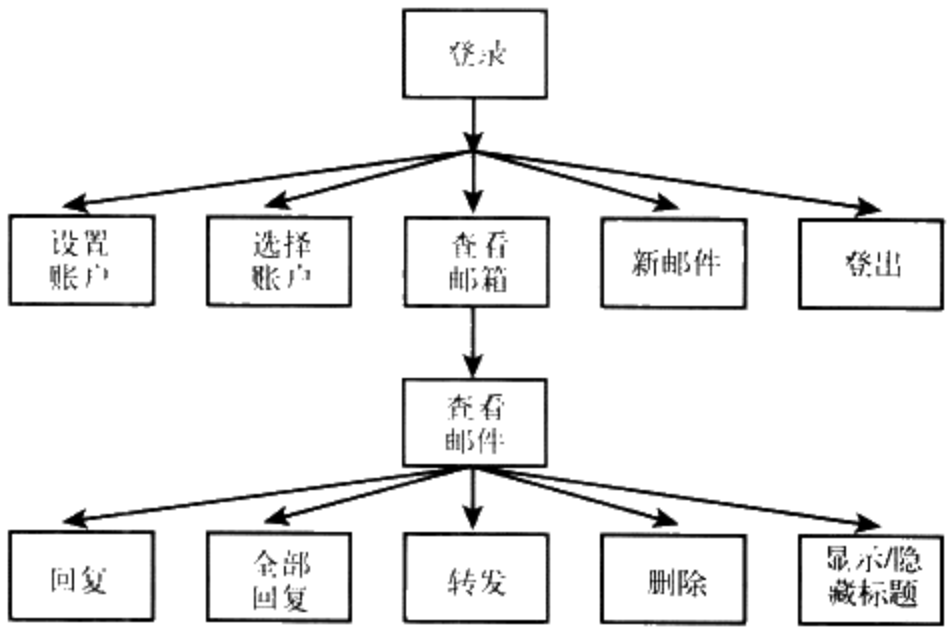

图29-1　Warm Mail界面为用户提供邮箱级的功能和消息级的功能

可以看到，我们首先要求用户登录，然后给他一个选项。用户可以选择创建一个新的邮件账户，或选择一个已有的账户来使用。用户也可以查看收到的来信——回复、转发或者删除它——还可以发送新邮件。

我们也给予用户查看特定消息详细标题的选项。查看完整标题可以显示更多关于消息的内容。我们可以看出该邮件来自哪一台机器——它是记录垃圾邮件的非常有用的工具。可以看出哪台机器转发了邮件，以及在什么时间到达每一台主机——这对于认清谁该对延时邮件负责是非常有用的。也可以看到发送者使用的是哪一种邮件客户端程序，如果应用程序将可选的信息加入到标题中的话。

在这个项目中，我们使用了一个稍有不同的应用程序架构。不是使用一组脚本，每一个脚本对应于一个模块，而是有一个稍长的脚本——`index.php`，该脚本就像由GUI驱动的事件循环一样工作。在站点上，我们通过点击按钮所触发的每一个操作都会将我们返回至`index.php`，但对应有一个不同的参数。根据参数的不同，脚本将调用不同的函数为用户提供适当的输出。这些函数照样都在函数库当中。

这种架构适合于像这个项目一样的小型应用程序。它适合那些基于事件驱动的应用程序，即用户动作触发某功能的应用程序。简单地使用事件处理不适合那些由整个团队人员操作的大型架构或项目。

Warm Mail项目所包含的文件概要如表29-1所示。

下面，让我们继续了解这个应用程序。

表29-1　Warm Mail应用程序中的文件

文 件 名	类　型	描　述
index.php	应用程序	运行整个应用程序的主要脚本
include_fns.php	函数	该应用程序包含文件的集合
data_valid_fns.php	函数	使输入数据生效的函数的集合
db_fns.php	函数	连接到mail数据库的函数集合
mail_fns.php	函数	打开邮箱、阅读邮件与邮件相关的函数集合
output_fns.php	函数	输出HTML函数的集合
user_auth_fns.php	函数	验证用户的函数的集合
create_database.sql	SQL	用来建立book_sc数据库和建立用户的SQL

29.3　建立数据库

Warm Mail的数据库相当简单，因为实际上我们并不打算在其中保存任何电子邮件。

我们需要保存系统用户的信息，我们将为每个用户保存如下所示的字段信息：

- 用户名——用户期望出现在Warm Mail中的用户名。
- 密码——用户期望出现在Warm Mail中的用户密码。
- 地址——用户期望出现在From field中的电子邮件地址，该地址将出现在从该系统发出去的电子邮件的From域。
- 显示名称——他们想在发送给其他人的邮件中显示的名称。

我们也需要为那些检查该系统的用户存储账户信息。对于每一个账户，必须保存以下信息：

- 用户名——该账户所属的Warm Mail用户。
- 服务器——该账户所在的机器，例如，本地机或mail.tangledweb.com.au，或其他Domain（域）。
- 端口——当使用该账户时连接的端口号。通常，对于POP3服务器来说，这个端口是110，而对于IMAP服务器来说，则是143。
- 类型——用来连接到该服务器的协议，POP3或者IMAP。
- 远程用户——连接到邮件服务器的用户名。
- 远程口令——连接到邮件服务器的口令。
- 账户ID——区别账户的唯一标识符。

可以通过运行程序清单29-1中所示的SQL语句来创建这个应用程序的数据库。

程序清单29-1　create_database.sql——用来创建邮件数据库的SQL

```
create database mail;

use mail;

create table users
(
  username char(16) not null primary key,
  password char(40) not null,
  address char(100) not null,
  displayname char(100) not null
);

create table accounts
(
  username char(16) not null,
  server char(100) not null,
  port int not null,
  type char(4) not null,
  remoteuser char(50) not null,
  remotepassword char(50) not null,
  accountid int unsigned not null auto_increment primary key
);

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

请记住，我们可以通过输入如下所示的命令来执行这个SQL语句：

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

我们必须提供root用户密码。此外，还应该在运行以上命令之前修改create_database.sql和db_fns.php中的邮件用户密码。

在本书附带的文件中，我们提供了一个叫做populate.sql的SQL文件。在这个应用程序中，我们没有创建用户注册或管理进程。如果要在一个更大规模的系统上使用该软件，可以自己添加一个。但是如果要作为个人使用，只需要将自己的账户信息插入到数据库中。populate.sql脚本为做这些提供了一个模板，以此将详细信息插入到其中并运行它以便将自己设置为用户。

29.4　了解脚本架构

正如我们前面提到的，该应用程序使用一个脚本控制所有操作。该脚本叫做index.php，如程序清单29-2所示。该脚本比较长，我们逐段详细介绍。

程序清单29-2　index.php——Warm Mail系统的框架

```
<?php
// This file is the main body of the Warm Mail application.
// It works basically as a state machine and shows users the
// output for the action they have chosen.

//*****************************************************************************
// Stage 1: pre-processing
// Do any required processing before page header is sent
// and decide what details to show on page headers
//*****************************************************************************

  include ('include_fns.php');
  session_start();
  //create short variable names
  $username = $_POST['username'];
  $passwd = $_POST['passwd'];
  $action = $_REQUEST['action'];
  $account = $_REQUEST['account'];
  $messageid = $_GET['messageid'];

  $to = $_POST['to'];
  $cc = $_POST['cc'];
  $subject = $_POST['subject'];
  $message = $_POST['message'];

  $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 ($username || $password) {
    if(login($username, $passwd)) {
      $status .= "<p style=\"padding-bottom: 100px\">Logged in
                  successfully.</p>";
      $_SESSION['auth_user'] = $username;
      if(number_of_accounts($_SESSION['auth_user'])==1) {
        $accounts = get_account_list($_SESSION['auth_user']);
        $_SESSION['selected_account'] = $accounts[0];
      }
    } else {
      $status .= '<p style=\'padding-bottom: 100px\">Sorry, we could
                  not log you in with that username and password.</p>";
    }
  }
```

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

//need to process choose, delete or store account before drawing header
switch ($action) {
  case  delete-account':
    delete_account($_SESSION['auth_user'], $account);
  break;

  case 'store-settings':
    store_account_settings($_SESSION[ auth_user'], $_POST);
  break;

  case 'select-account':
    // if have chosen a valid account, store it as a session variable
    if(($account) && (account_exists($_SESSION['auth_user'], $account))) {
      $_SESSION['selected_account'] = $account;
    }
  break;
}
  // set the buttons that will be on the tool bar
  $buttons[0] =  view-mailbox';
  $buttons[1] = 'new-message ;
  $buttons[2] = 'account-setup ;

  //only offer a log out button if logged in
  if(check_auth_user()) {
    $buttons[4] =  log-out';
  }

//*****************************************************************************
// Stage 2: headers
// Send the HTML headers and menu bar appropriate to current action
//*****************************************************************************
  if($action) {
    // display header with application name and description of page or action
    do_html_header($_SESSION['auth_user'], "Warm Mail - ".
                   format_action($action),
                   $_SESSION['selected_account']);
  } else {
    // display header with just application name
    do_html_header($_SESSION['auth_user'], 'Warm Mail",
                   $_SESSION['selected_account']);
```

```
  }

  display_toolbar($buttons);

//*****************************************************************************
// Stage 3: body
// Depending on action, show appropriate main body content
//*****************************************************************************
  //display any text generated by functions called before header
  echo $status;

  if(!check_auth_user())
    echo "<p>You need to log in ";

    if(($action) && ($action!= 'log-out')) {
      echo "to go to".format_action($action);
    }
    echo ".</p>";
    display_login_form($action);
  } else {
    switch ($action) {
      // if we have chosen to setup a new account, or have just added or
      // deleted an account, show account setup page
      case 'store-settings':
      case 'account-setup':

      case 'delete-account' :
        display_account_setup($_SESSION['auth_user']);
      break;

      case 'send-message':
      if(send_message($to, $cc, $subject, $message)) {
        echo '<p style=\"padding-bottom: 100px\">Message sent.</p>";
      } else {
        echo "<p style=\'padding-bottom: 100px\'>Could not send message.</p>";
      }
    break;

    case 'delete':
      delete_message($_SESSION['auth_user'],
                     $_SESSION['selected_account'], $messageid);
      //note deliberately no 'break' - we will continue to the next case

    case 'select-account':

    case 'view-mailbox':
      // if mailbox just chosen, or view mailbox chosen, show mailbox
```

```
    display_list($_SESSION['auth_user'],
                 $_SESSION['selected_account']);
  break;

  case 'show-headers':
  case 'hide-headers':
  case 'view-message':
    // if we have just picked a message from the list, or were looking at
    // a message and chose to hide or view headers, load a message
    $fullheaders = ($action == 'show-headers');
    display_message($_SESSION['auth_user'],
                    $_SESSION['selected_account'],
                    $messageid, $fullheaders);
  break;

  case 'reply-all':
    //set cc as old cc line
    if(!$imap) {
      $imap = open_mailbox($_SESSION['auth_user'],
                           $_SESSION['selected_account']);
    }

    if($imap) {
      $header = imap_header($imap, $messageid);
      if($header->reply_toaddress) {
        $to = $header->reply_toaddress;
      } else {
        $to = $header->fromaddress;
      }

      $cc = $header->ccaddress;
      $subject = "Re: ".$header->subject;
      $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
      imap_close($imap);

      display_new_message_form($_SESSION['auth_user'],
      $to, $cc, $subject, $body);
    }

  break;

  case 'reply':
    //set to address as reply-to or from of the current message
    if(!$imap) {
      $imap = open_mailbox($_SESSION['auth_user'],
                           $_SESSION['selected_account']);
    }
```

```
if($imap) {
  $header = imap_header($imap, $messageid);
  if($header->reply_toaddress) {
    $to = $header->reply_toaddress;
  } else {
    $to = $header->fromaddress;
  }
  $subject = 'Re: '.$header->subject;
  $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
  imap_close($imap);

  display_new_message_form($_SESSION['auth_user'],
                           $to, $cc, $subject, $body);
}
break;

case 'forward':
  //set message as quoted body of current message
  if(!$imap) {
    $imap = open_mailbox($_SESSION['auth_user'],
                         $_SESSION['selected_account']);
  }

  if($imap) {
    $header = imap_header($imap, $messageid);
    $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
    $subject = "Fwd: ".$header->subject;
    imap_close($imap);

    display_new_message_form($_SESSION['auth_user'],
                             $to, $cc, $subject, $body);
  }
  break;

  case 'new-message':
    display_new_message_form($_SESSION['auth_user'],
                             $to, $cc, $subject, $body);
  break;
 }
}
//*****************************************************************************
// Stage 4: footer
//*****************************************************************************
  do_html_footer();
?>
```

这个脚本使用一个事件处理的方法。它包含对于每个事件应该调用哪个函数的知识或逻辑。

在这种情况下，事件是由用户点击站点内不同的按钮触发的，每一个按钮对应一种动作。大多数按钮都是由display_button()函数产生的，但是如果用户点击的按钮是提交按钮时，则是调用display_form_button()函数。这些函数都在output_fns.php中。这些都转至表单的URL：

```
index.php?action=log-out
```

当调用index.php时，action变量值决定哪一个事件处理程序被触发。

该脚本由如下所示的4个主要部分组成：

1）在将页面标题发送到浏览器之前要做一些必要的处理，例如，开始一个会话，执行用户选择的动作所必须的预处理操作，并确定页面标题的内容。

2）为用户选择的动作处理和发送适当的标题和菜单条。

3）选择要执行哪一块代码，取决于用户所选的动作。不同的动作将触发不同的函数调用。

4）发送页面的页脚。

如果粗略地看一下代码，会发现这四个部分都用注释作了标记。

为了完全理解脚本程序，让我们来真正使用这个站点。

29.5　登录与登出

当用户载入index.php页面时，将看到如图29-2所示的输出页面。

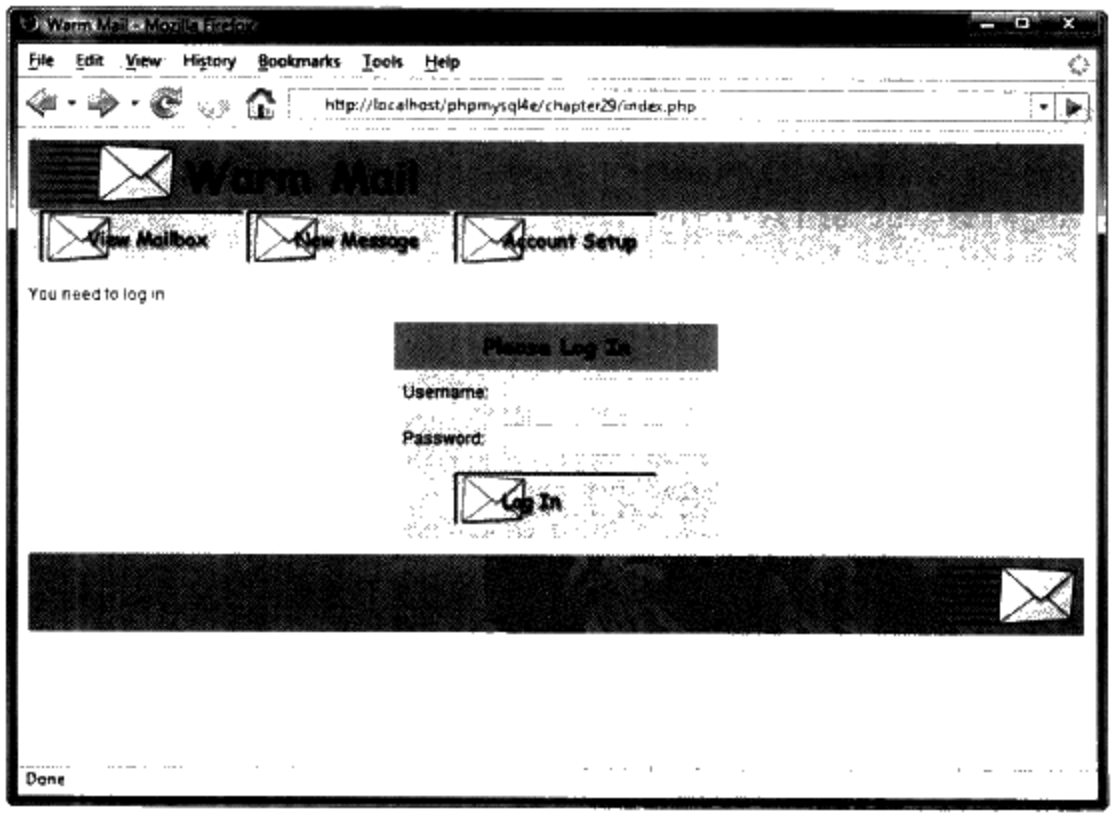

图29-2　Warm Mail系统的登录页面询问用户名与密码

这是该应用程序的默认动作。如果没有选择$action动作，并且也没有提供登录细节，将执行代码的下面部分。

在预处理阶段，PHP将首先执行如下所示的代码：

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

```
session_start();
```

以上代码将开始一个会话，该会话可以用来保存会话变量$auth_user和$selected_account，这两个变量后面会用到。

在其他应用程序中，我们创建了短的变量名称。从第1章“PHP快速入门教程”开始，我们就在与每一个表单相关的脚本中使用短的变量名称，因此除了action变量外，我们不需要再提这一点。根据应用程序功能模块的不同，它可能是一个GET或POST变量。因此，可以从$_REQUEST数组获得这个变量。对于account变量也是如此，因此通常它可以通过GET进行访问，但是当要删除一个账户时，需要通过POST进行访问。

为了在定制用户界面时减少工作量，可以使用一个数组来控制出现在工具条中的按钮。我们声明了一个如下所示的空数组：

```
$buttons = array();
```

并将按钮设置为我们希望在页面看到的那些按钮：

```
$buttons[0] = 'view-mailbox';
$buttons[1] = 'new-message';
$buttons[2] = 'account-setup';
```

如果用户是以管理员身份登录的，还需要在这个数组中添加更多按钮。

在标题阶段，我们打印了一个普通的vanilla标题：

```
do_html_header($_SESSION['auth_user'], 'Warm Mail',
               $_SESSION['selected_account']);
...
display_toolbar($buttons);
```

这些代码将打印标题和标题头，然后可以看到如图29-2所示的按钮工具条。这些函数可以在output_fns.php函数库中找到，但由于可以轻易地在图中看出他们的运行结果，因此在这里，我们不再详细介绍它们。

现在，我们来了解代码的主体部分：

```
if(!check_auth_user()) {
  echo "<p>You need to log in";

  if(($action) && ($action!='log-out')) {
    echo " to go to ".format_action($action);
  }
  echo ".</p>";
  display_login_form($action);
}
```

check_auth_user()函数来自于user_auth_fns.php库。我们已经在前面的某些项目中使用过与此非常相似的代码——它检查用户是否登录。在这里，作为一种情况，如果用户没有登录，我们将显示一个登录表单，如图29-2所示。我们使用output_fns.php中的display_login_form()函数绘制该表单。

如果用户正确填写了表单并点击“Log In”按钮，将看到如图29-3所示的输出页面。

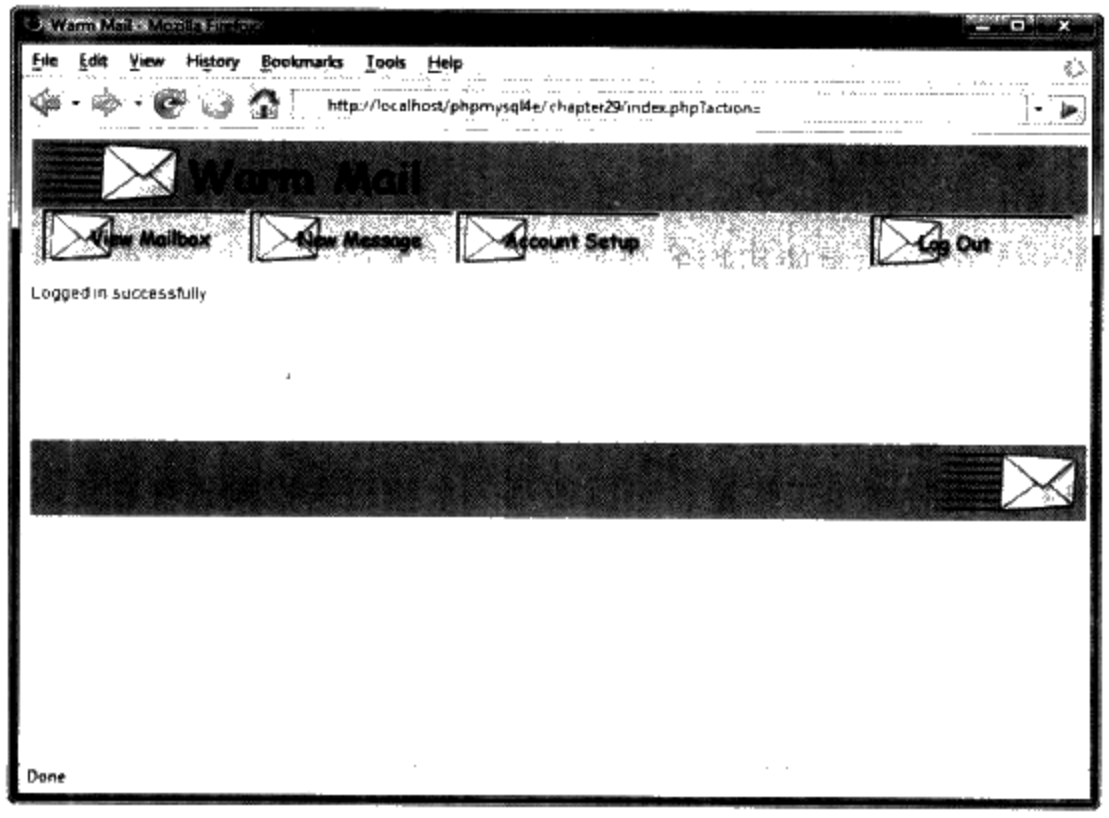

图29-3　在成功登录后，用户可以开始使用该应用程序

在这个脚本的执行过程中，我们将触发代码的不同部分。登录表单有两个字段，$username和$password。如果填好了这两个字段，如下所示的预处理代码将会被触发：

```
if ($username || $password) {
  if(login($username, $passwd)) {
    $status .= "<p style=\"padding-bottom: 100px\">Logged in successfully.</p>";
    $_SESSION['auth_user'] = $username;
    if(number_of_accounts($_SESSION['auth_user'])>1) {
      $accounts = get_account_list($_SESSION['auth_user']);
      $_SESSION['selected_account'] = $accounts[0];
    }
  } else {
    $status .= '<p style=\'padding-bottom: 100px\'>Sorry, we could not log you
                in with that username and password.</p>';
  }
}
```

可以看到，以上代码将调用login()函数，该函数与第27章“建立用户身份验证机制和个性化设置”和第27章“创建一个购物车”中的相似。如果一切正常，我们将在会话变量auth_user中注册该用户名。

除了设置当用户没有登录时所看到的按钮外，也可以添加另外的按钮让用户登出，如下所示：

```
if(check_auth_user()) {
   $buttons[4] = 'log-out';
}
```

可以在图29-3中看到这个“Log Out”按钮。

在“标题”阶段，我们再次显示了标题和按钮。在正文阶段，显示了以前设置的状态信息：

```
echo $status;
```

接下来，我们将打印页脚并等待用户的下一步操作。

29.6　建立账户

当用户第一次使用Warm Mail系统时，需要建立一些电子邮件账户。如果用户点击“Account Setup”按钮，将会把action变量的值设为account-setup并再次调用index.php脚本。用户将看到如图29-4所示的输出页面。

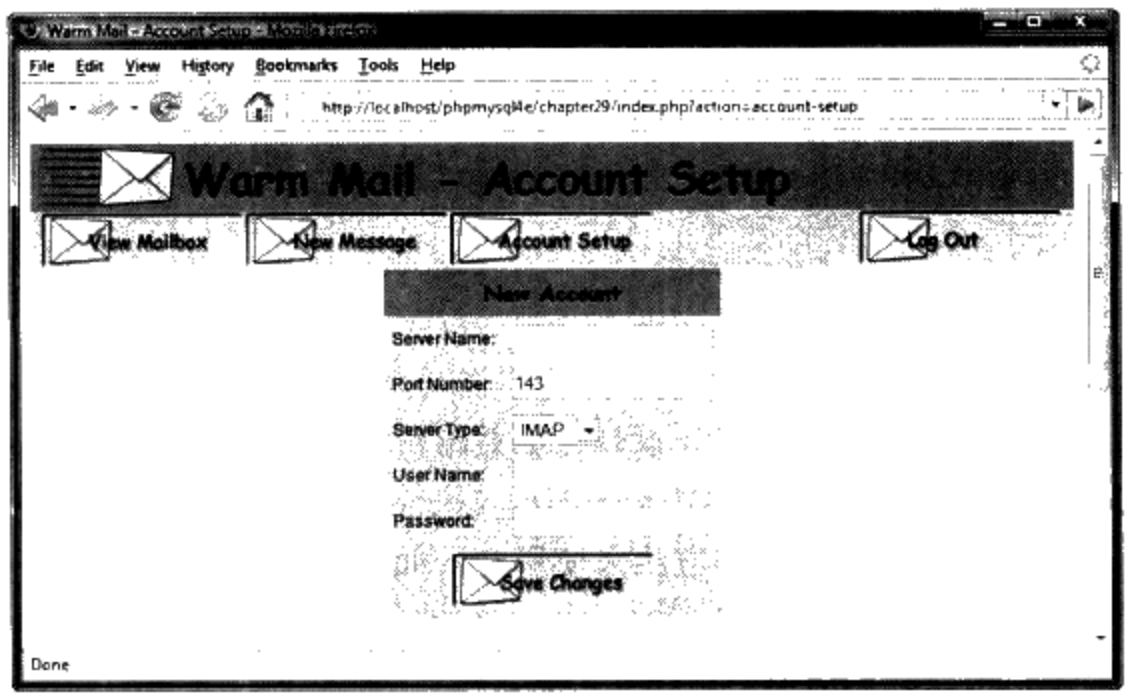

图29-4　用户在可以阅读电子邮件前必须建立电子邮件账户

现在，让我们回过头来看看程序清单29-2。这一次，由于$action值的不同，我们看到不同的行为。我们将得到稍有不同的标题，如下所示：

```
do_html_header($_SESSION['auth_user'], 'Warm Mail - '
               format_action($action),
               $_SESSION['selected_account']);
```

更重要的是，我们将得到一个不同的正文，如下所示：

```
case 'store-settings':
case  account-setup :
case  delete-account':
      display_account_setup($_SESSION['auth_user']);
break;
```

这是一个典型的模式：每个命令都将调用一个函数。在这个例子中，我们调用了display_account_setup()函数。该函数的代码如程序清单29-3所示。

程序清单29-3　来自output_fns.php文件的display_account_setup()函数——获得并显示账户详细信息的函数

```
function display_account_setup($auth_user) {
  //display empty 'new account' form

  display_account_form($auth_user);
  $list = get_accounts($auth_user);
  $accounts = sizeof($list);

  // display each stored account
  foreach($list as $key => $account) {
    // display form for each accounts details.
    // note that we are going to send the password for all accounts in the HTML
    // this is not really a very good idea
    display_account_form($auth_user, $account['accountid'], $account['server'],
                         $account['remoteuser'], $account['remotepassword'],
                         $account['type'], $account['port']);
  }
}
```

当我们调用该函数时，它将显示一个空白表单来添加新账户，接着是包括该用户所有当前邮件账户的可编辑文本框。display_account_form()函数将显示如图29-4所示的表单。可以发现，在这里我们通过两种不同的方法来使用它：如果不带参数，将显示一个空表单，如果带有一组完整的参数，则显示一条已有的记录。这个函数在output_fns.php库中；它只是简单地输出HTML，因此我们这里不详细介绍它。

检索任何已存在账户的函数是get_accounts()，它出自mail_fns.php库。函数代码如清单29-4所示。

程序清单29-4　来自mail_fns.php文件的get_accounts()函数——为某个特定用户检索所有的账户细节

```
function get_accounts($auth_user) {
  $list = array();
  if($conn->db_connect()) {
    $query = 'select * from accounts where username = '.$auth_user.'';
    $result = $conn->query($query);
    if($result) {
      while($settings = $result->fetch_assoc()) {
        array_push($list, $settings);
      }
    } else {
      return false;
    }
  }
  return $list;
}
```

可以看到，该函数将连接数据库，检索特定用户的所有账户并以数组的形式返回。

29.6.1　创建一个新账户

如果用户填写了账户表单并点击“Save Changes”按钮，store-settings动作将会被触发。下面，让我们来看看在index.php脚本中该动作的事件处理代码。在预处理阶段，执行如下所示的代码：

```
case 'store-settings':
   store_account_settings($_SESSION['auth_user'], $_POST);
break;
```

store_account_settings()函数将一个新账户的详细信息写入数据库。该函数的代码如程序清单29-5所示。

程序清单29-5　mail_fns.php文件的store_account_settings()函数——为用户保存一个新账户详细信息的函数

```
function store_account_settings($auth_user, $settings) {
  if(!filled_out($settings)) {
    echo '<p>All fields must be filled in. Try again.</p>";
    return false;
  } else {
    if($settings['account']>0) {
       $query = "update accounts set server = '".$settings[server]."',
                 port = ".$settings[port].", type = '".$settings[type]."',
                 remoteuser = '".$settings[remoteuser]."',
                 remotepassword = '".$settings[remotepassword]."'
               where accountid = '".$settings[account]."'
                 and username = '".$auth_user."'";
  } else {
    $query = "insert into accounts values ('".$auth_user."',
                  '".$settings[server]."', '".$settings[port]."',
                  '".$settings[type]."', '".$settings[remoteuser]."',
                  '".$settings[remotepassword]."', NULL)";
  }

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

```
    }
}
```

可以看到，这个函数中有两个选项分别对应于插入一个新账户和更新一个已有的账户。函数执行相应的查询语句来保存账户详细信息。

在保存账户详细信息后，我们将返回到index.php的主体阶段：

```
case  store-settings :
case 'account-setup':
case 'delete-account':
      display_account_setup($_SESSION['auth_user']);
break;
```

可以看到，我们像以前一样执行display_account_setup()函数来列出用户的账户详细信息。新添加的账户就被包括进来了。

29.6.2　修改已有账户

修改已有账户的过程也是非常相似的。用户可以修改账户的详细信息并点击“Save Changes”按钮。这将再次触发store-settings动作，但是，这一次，它将更新账户详细信息而不是插入账户信息。

29.6.3　删除账户

要删除一个账户，可以点击显示在每个账户列表下面的“Delete Account”按钮。这会触发delete-account动作。

在index.php脚本的预处理阶段，该操作将执行如下所示的代码：

```
case 'delete-account':
   delete_account($_SESSION['auth_user'], $account);
break;
```

这段代码将调用delete_account()函数。该函数的代码如程序清单29-6所示。由于可能用到的账户包含在标题中，因此删除账户操作必须在“标题”阶段以前处理。在账户列表可以正确地列出之前需要进行更新。

程序清单29-6　mail_fns.php文件的delete_account()函数——删除一个账户详细信息的函数

```
function delete_account($auth_user, $accountid) {
  //delete one of this user's accounts from the DB

  $query = "delete from accounts where accountid = '".$accountid."'
           and username = '".$auth_user."'";
  if($conn=db_connect()) {
    $result = $conn->query($query);
  }
  return $result;
}
```

当程序执行返回到index.php时，主体阶段将运行如下所示的代码：

```
case 'store-settings' :
case 'account-setup':
case 'delete-account':
      display_account_setup($_SESSION['auth_user']);
break;
```

请注意，我们会发现这与前面运行的代码一样，它仅仅显示了用户账户的列表。

29.7 阅读邮件

在用户建立一些账户后，我们转移到重点问题上来：连接这些账户并读取邮件。

29.7.1 选择账户

我们需要选取一个用户账户以便从中读取邮件。当前选择的账户存储在会话变量$selected_account中。

如果该用户在系统中仅注册了一个账户，那么登录时会自动选取该账户，如下所示：

```
if(number_of_accounts($_SESSION['auth_user'])==1) {
  $accounts = get_account_list($_SESSION['auth_user']);
  $_SESSION['selected_account'] = $accounts[0];
}
```

来自mail_fns.php文件中的number_of_accounts()函数用来判断用户是否有多个账户。该函数源代码如程序清单29-7所示。get_account_list()函数将检索用户账户ID的数组。在这种情况下，用户仅有一个账户，因此也就是数组的第0个值。

程序清单29-7 mail_fns.php文件中的number_of_accounts()函数——用来计算某个用户注册了多少个账户的函数

```
function number_of_accounts($auth_user) {
  // get the number of accounts that belong to this user
  $query = 'select count(*) from accounts where
            username = '".$auth_user."'';

  if($conn=db_connect()) {
    $result = $conn->query($query);
      if($result) {
        $row = $result->fetch_array();
        return $row[0];
      }
  }
  return 0;
}
```

get_account_list()函数与我们以前看到的get_accounts()函数类似，不同之处在于它仅仅检索账户名称。

如果用户注册了多个账户，需要选择使用其中的一个。在这种情况下，标题中将包括一个列表框，列出所有可供使用的邮箱。选择一个适当的邮箱后系统会自动为用户显示该邮箱内容，如图29-5所示。

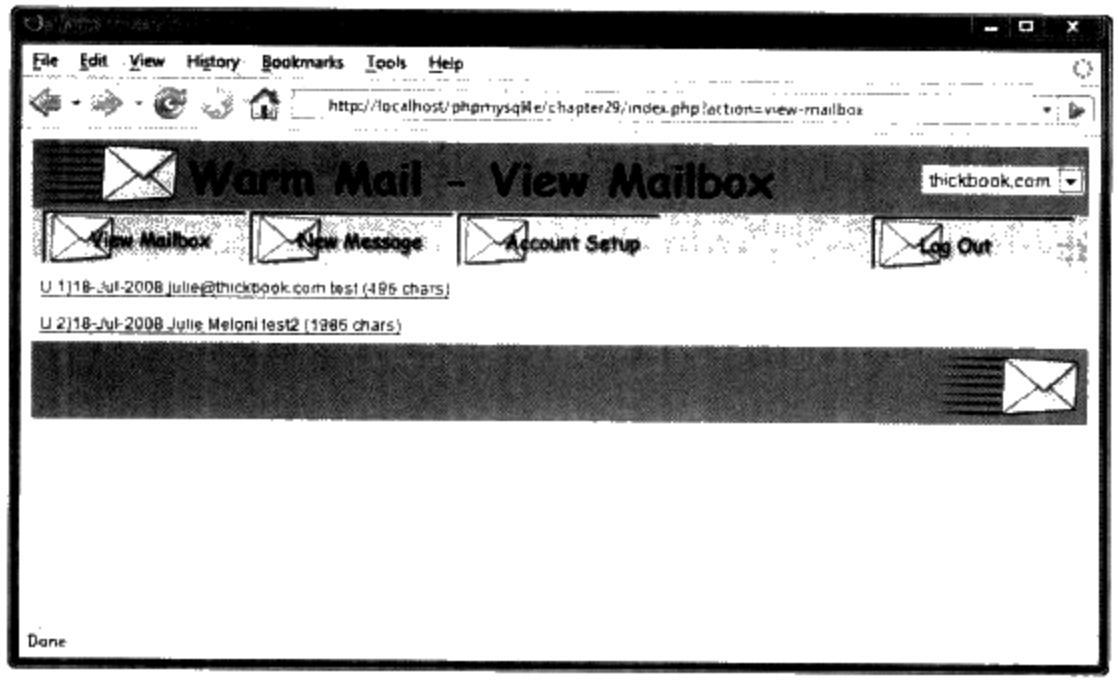

图29-5　从列表框中选定一个账户后，该账户的邮件将被下载并显示出来

列表框的选项由output_fns.php文件中的do_html_header()函数生成，如下代码段所示：

```
// include the account select box only if the user has more than one account
if(number_of_accounts($auth_user)>1) {
  echo "<form action=\'index.php?action=open-mailbox\' method=\"post\">
        <td bgcolor=\"#ff6600\" align=\"right\" valign=\'middle\">";
        display_account_select($auth_user, $selected_account);
  echo "</td>
        </form>";
}
```

在本书的例子中，我们一直尽量避免讨论HTML，但是由函数display_account_select()生成的HTML值得我们仔细查看一下。

根据当前用户账户的不同，display_account_select()函数将生成如下所示的HTML代码：

```
<select
  onchange="window.location=this.option[selectedIndex].value
            name=account'>
  <option
     value="index.php?action=select-account&account=4" selected >
     thickbook.com
  </option>
  <option
     value="index.php?action=select-account&account=3">
     localhost
```

```
    </option>
  </select>
```

其中，大部分代码仅仅是一个HTML的select元素，当然，它也包括一些JavaScript。与PHP生成HTML的方式相同，JavaScript也可以用来生成客户端脚本。

对于该元素来说，无论什么时候发生了变化事件，JavaScript就将window.location选项赋值为当前的选项值。如果用户选择了列表框中的第一项，window.location将被设置为“index.php?action = select-account & account =10”。这样就会使浏览器载入这个URL。很明显，如果用户使用的浏览器不支持JavaScript或禁用了JavaScript，这段代码将不会产生任何结果。

output_fns.php文件中的display_account_select()函数可以用来获得特定用户的所有可用账户列表并显示列表框。它也使用前面讨论过的get_account_list()函数。

在列表框中选择一个选项将触发select_account事件。观察图29-5中所示的URL，可以看到该事件和选中的账户ID都被添加到URL的末尾了。

在URL末尾添加这些参数产生两个作用。首先，在index.php的预处理阶段，选中的账户ID将被存储在会话变量$selected_account中，如下所示：

```
case 'select-account':
   // if have chosen a valid account, store it as a session variable
   if((!$account) && (account_exists($_SESSION['auth_user'],
         $account))) {
     $_SESSION['selected_account'] = $account;
   }
break;
```

其次，当执行到该脚本的主体部分时，将运行如下所示代码：

```
case 'select-account':
case 'view-mailbox':
   // if mailbox just chosen, or view mailbox chosen, show mailbox
   display_list($_SESSION['auth_user'],
                $_SESSION['selected_account']);
break;
```

可以看到，在这里，我们执行了相同的动作，如同用户选择了查看邮箱选项一样。下面我们接着讨论。

29.7.2 查看邮箱内容

使用display_list()函数，可以查看邮箱的内容。该函数将显示邮箱中所有消息的列表。该函数的代码如程序清单29-8所示。

程序清单29-8 output_fns.php文件中的display_list()函数——显示所有邮箱消息

```
function display_list($auth_user, $accountid) {
  // show the list of messages in this mailbox
```

```
global $table_width;

if(!$accountid) {
  echo "<p style=\'padding-bottom: 100px\'>No mailbox selected.</p>";
} else {

  $imap = open_mailbox($auth_user, $accountid);

  if($imap) {
    echo "<table width=\"".$table_width."\' cellspacing=\"0\"
                cellpadding=\'6\' border=\"0\">";

  $headers = imap_headers($imap);
  // we could reformat this data, or get other details using
  // imap_fetchheaders, but this is not a bad summary so we
  // just echo each

  $messages = sizeof($headers);
  for($i = 0; $i<$messages; $i++) {
    echo "<tr><td bgcolor=\"";
    if($i%2) {
      echo "#ffffff";
    } else {
      echo "#ffffcc";
    }
      echo "\"><a href=\'index.php?action=view-message&messageid="
            .($i+1)."\">";
      echo $headers[$i];
      echo "</a></td></tr>\n";
    }
    echo "</table>";
  } else {
    $account = get_account_settings($auth_user, $accountid);
      echo "<p style=\"padding-bottom: 100px\">Could not open mail
            box ".$account['server'].".</p>";
    }
  }
}
```

在这个函数中，我们实际上已开始使用了PHP的IMAP函数。该函数的两个关键部分是打开邮箱和阅读消息标题。

我们通过调用mail_fns.php文件中的open_mailbox()函数来打开用户的账户邮箱。函数代码如程序清单29-9所示。

程序清单29-9　mail_fns.php文件中的open_mailbox()函数——这个函数连接到用户邮箱

```
function open_mailbox($auth_user, $accountid) {

  // select mailbox if there is only one
```

```
  if(number_of_accounts($auth_user)==1) {
    $accounts = get_account_list($auth_user);
    $_SESSION['selected_account'] = $accounts[0];
    $accountid = $accounts[0];
  }

  // connect to the POP3 or IMAP server the user has selected
  $settings = get_account_settings($auth_user, $accountid);
  if(!sizeof($settings)) {
      return 0;
  }
  $mailbox = '{'.$settings[server];
  if($settings[type]=='POP3') {
    $mailbox .= '/pop3';
  }
  $mailbox .= ':'.$settings[port].'}INBOX';

  // suppress warning, remember to check return value
  @$imap = imap_open($mailbox, $settings['remoteuser'],
           $settings['remotepassword']);
  return $imap;
}
```

实际上，我们是使用imap_open()函数来打开邮箱的。函数原型如下：

```
int imap_open (string mailbox, string username, string password [, int options])
```

该函数所需的参数如下所示：

- mailbox——这个字符串应包括服务器名和邮箱名，以及可选的端口号和协议。该字符串的格式如下所示：

```
{hostname/protocol:port}boxname
```

如果没有指定协议，则默认为IMAP。在我们编写的代码中，可以看到，如果用户为某一特定账户指定了POP3协议，我们就会指定为POP3。

例如，要使用默认端口从本地机器读取邮件，对于IMAP协议，我们使用如下所示的邮箱：

```
{localhost:143}INBOX
```

对于POP3协议，我们使用如下所示的邮箱：

```
{localhost/pop3:110}INBOX
```

- username——该账户的用户名。
- password——该账户的密码。

也可通过提交可选的标记来指定某些选项，例如，以只读方式打开邮箱。

值得注意的是，我们在将邮箱名称字符串传递给imap_open()函数打开之前，是一段一段地将它组织起来的。因此，在构造该串时，必须仔细一些，因为在PHP中包含“{$”的字符串会引起问题。

如果邮箱可以成功打开，该函数调用将返回一个IMAP流，如果不能成功打开则返回false。

当完成对一个IMAP流的操作时，可以调用imap_close(imap_stream)函数关闭它。在这个函数中，IMAP流被传回至主程序。接着，我们使用imap_headers()函数获得要显示的电子邮件标题：

```
$headers = imap_headers($imap);
```

这个函数将返回我们已经连接的邮箱所有消息的标题信息。这些信息以数组的形式返回，每条消息占一行。我们并没有对这些消息进行格式化处理。每条消息在输出中将占用一行，可从图29-5看到其输出形式。

可以通过使用imap_headers()函数来得到关于电子邮件标题的更多信息，该函数与imap_header()名称类似，因此易发生混淆。在本示例中，imap_headers()函数给出的详细信息对我们的项目目标来说已经足够了。

29.7.3　阅读邮件消息

我们已经将前面的display_list()函数返回的消息数组中的每一条消息连接到了特定的邮件消息。每个连接的格式如下所示：

```
index.php?action=view-message&messageid=5
```

messageid是我们前面在标题中得到的索引号。请注意，IMAP消息索引是从1开始的，而不是0。

如果用户点击了其中一个链接，将看到如图29-6所示的输出。

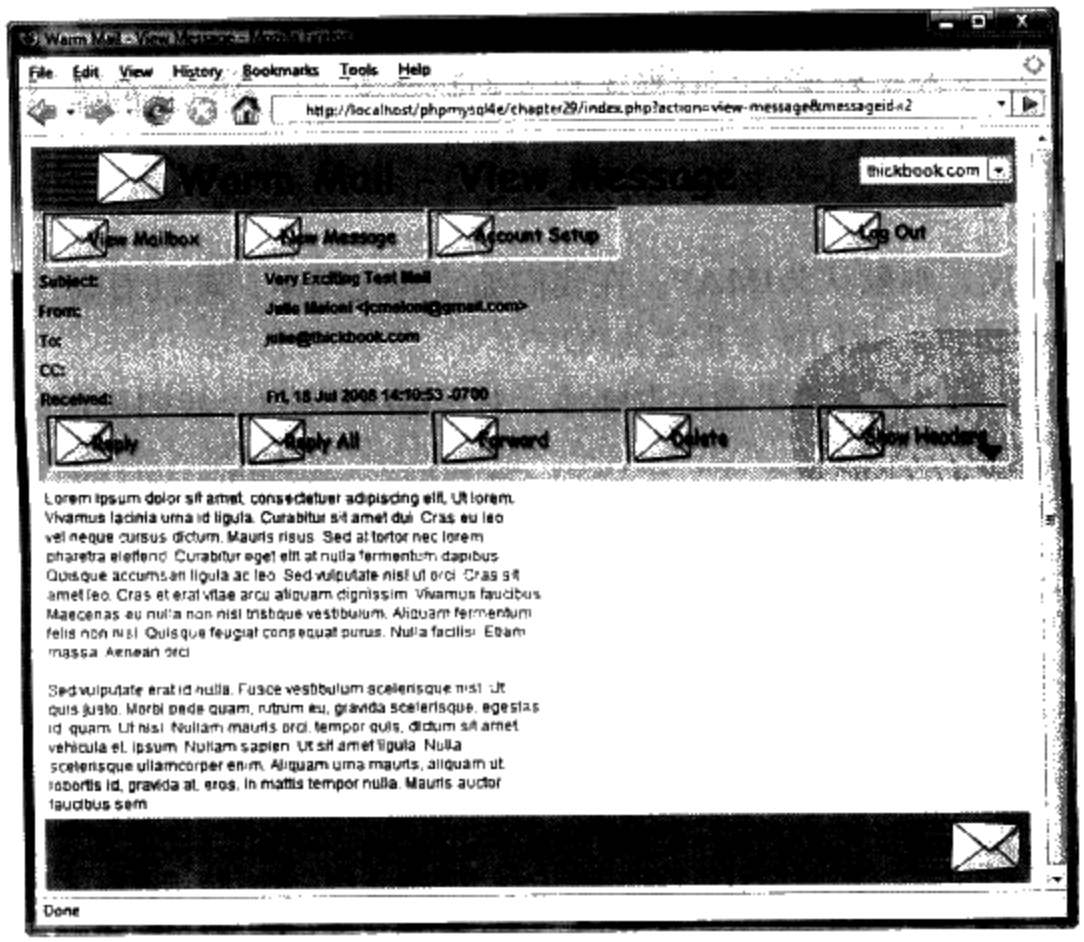

图29-6　用view-message动作来显示一个特定消息。在这个例子中，该消息是一封垃圾邮件

当我们将这些参数引入到index.php脚本后，将执行如下所示代码：

```
case 'show-headers':
case 'hide-headers' :
case 'view-message' :
  // if we have just picked a message from the list, or were looking at
  // a message and chose to hide or view headers, load a message
  $fullheaders = ($action == 'show-headers' );
    display_message($_SESSION['auth_user'],
                    $_SESSION['selected_account'],
                    $messageid, $fullheaders);
break;
```

这里，我们检查了$action的变量值是否等于"show-headers"。在这个例子中，该逻辑判断值为false，所以$fullheaders变量值被设置为false。稍后，我们再详细介绍"show-headers"动作。

如下代码行：

```
$fullheaders = ($action == 'show-headers' );
```

如果按如下方式编写，可能会更繁琐一些，但是，却可能更清楚：

```
if ($action == 'show-headers') {
  $fullheaders = true;
} else {
  $fullheaders = false;
}
```

接下来，我们将调用display_message()函数。该函数的输出大部分都是纯HTML，因此在这里，我们将不做详细介绍。调用retrieve_message()函数从邮箱中检索适当的消息：

```
$message = retrieve_message($auth_user, $accountid, $messageid, $fullheaders);
```

retrieve_message()函数在mail_fns.php库中，其代码如程序清单29-10所示。

程序清单29-10　mail_fns.php文件的retrieve_message()函数——该函数将从邮箱中取得指定的消息

```
function retrieve_message($auth_user, $accountid, $messageid,
                          $fullheaders) {
  $message = array();

  if(!($auth_user && $messageid && $accountid)) {
    return false;
  }
  $imap = open_mailbox($auth_user, $accountid);
  if(!$imap)
    return false;
  }
  $header = imap_header($imap, $messageid);
```

```
if(!$header) {
  return false;
}
$message['body'] = imap_body($imap, $messageid);
if(!$message['body']) {
  $message['body'] = "[This message has no body]\n\n\n\n\n\n";
}
if($fullheaders) {
  $message['fullheaders'] = imap_fetchheader($imap, $messageid);
} else {
  $message['fullheaders'] = '';
}

$message['subject'] = $header->subject;
$message['fromaddress'] = $header->fromaddress;
$message['toaddress'] = $header->toaddress;
$message['ccaddress'] = $header->ccaddress;
$message['date'] = $header->date;

// note we can get more detailed information by using from and to
// rather than fromaddress and toaddress, but these are easier

imap_close($imap);
return $message;
}
```

我们将再次使用open_mailbox()函数打开用户的邮箱。但是，这次我们是查找一个指定的消息，使用该函数库，我们可以分别下载消息标题与消息正文。

在这里，我们使用的3个IMAP函数分别是imap_header()、imap_fetchheader()和imap_body()。请注意，这两个标题函数不同于我们前面使用的imap_headers()。它们的名字在某种程度上容易造成混淆。简而言之：

- imap_headers()——返回邮箱中所有消息的标题摘要。每条消息作为数组的一个元素，并且以数组形式返回。
- imap_header()——以对象的形式返回一条特定消息的标题。
- imap_fetchheader()——以字符串的形式返回一条特定消息的标题。

在这个例子中，我们使用imap_header()函数来填充指定的标题域，用imap_fetchheader()函数来显示用户请求的完整标题（稍后，我们还将详细介绍这一点）。

我们使用imap_header()函数和imap_body()函数建立一个包括我们感兴趣的消息的所有元素的数组。我们以如下方式调用imap_header()函数：

```
$header = imap_header($imap, $messageid);
```

可以从这个对象中获得所请求的每一个域：

```
$message['subject'] = $header->subject;
```

调用imap_body()函数可以将消息体添加到该数组，如下所示：

```
$message['body'] = imap_body($imap, $messageid);
```

最后，我们将调用imap_close()函数关闭邮箱并返回已经创建的数组。display_message()函数可以在表单中显示消息的域，如图29-6所示。

29.7.4　查看消息标题

如图29-6所示，可以看到一个“Show Headers”按钮。这个按钮可以触发“show-headers”选项，它会将该消息的完整标题加入到消息显示中。如果用户点击此按钮，可以看出如图29-7所示的页面。

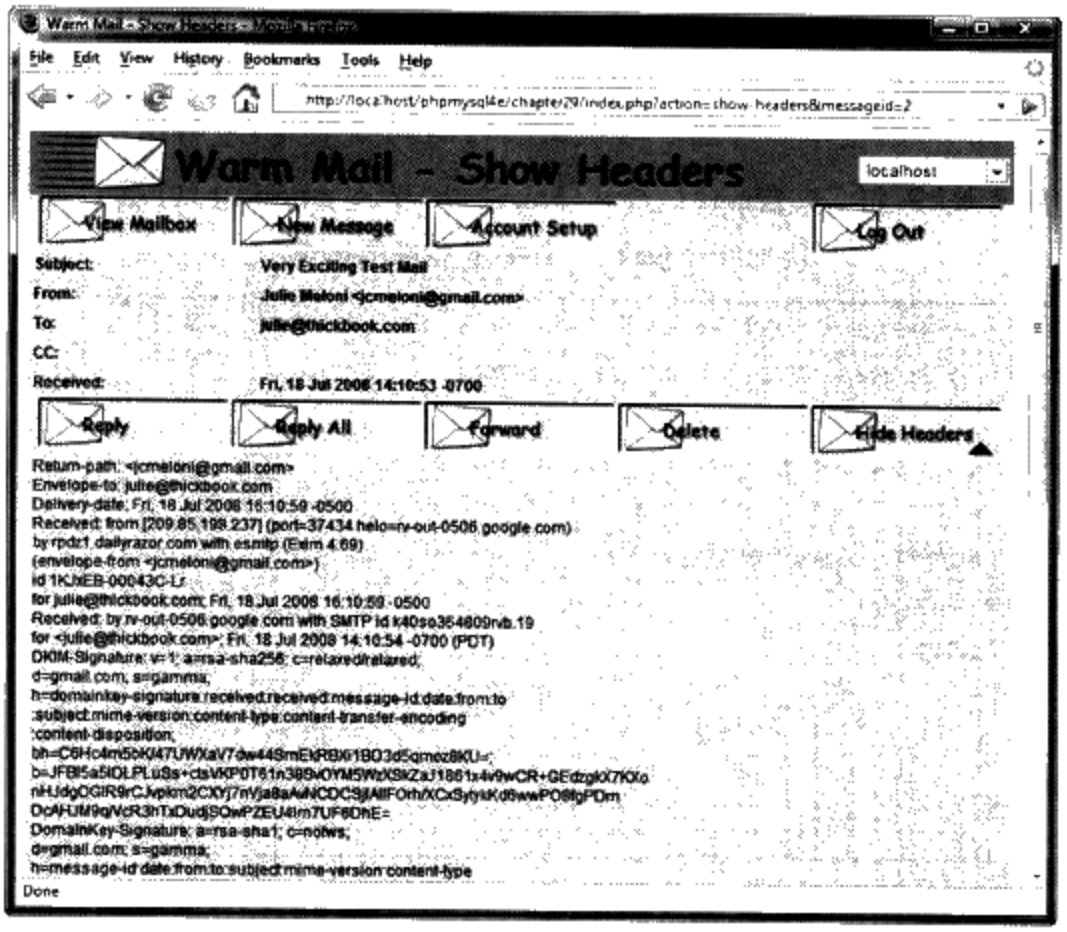

图29-7　使用“show-headers”选项查看消息的完整标题将帮助用户跟踪垃圾邮件的来源

注意view-message的事件处理也包括show-headers（和与之对应的hide-headers）。如果选择了该选项，会和前面一样执行相同的操作。但在retrieve_message()函数中，也可得到标题的完整文本，如下所示：

```
if($fullheaders) {
  $message['fullheaders'] = imap_fetchheader($imap, $messageid);
}
```

然后我们可以将这些标题显示给用户。

29.7.5　删除邮件

如果用户点击某一特定邮件上的“删除”按钮，将激活“delete”操作。该操作执行

index.php脚本中的如下代码：

```
case 'delete':
   delete_message($_SESSION['auth_user'],
                  $_SESSION['selected_account'], $messageid);
   //note deliberately no 'break' - we will continue to the next case

case 'select-account':
case 'view-mailbox':
   // if mailbox just chosen, or view mailbox chosen, show mailbox
   display_list($_SESSION['auth_user'],
                $_SESSION['selected_account']);
break;
```

可以看到，我们使用delete_message()函数删除消息，然后，像前面讨论的那样显示删除消息后的邮箱。delete_message()函数代码如程序清单29-11所示。

程序清单29-11　来自mail_fns.php的delete_message()函数——从邮箱中删除一个指定消息的函数

```
function delete_message($auth_user, $accountid, $message_id) {
  // delete a single message from the server

  $imap = open_mailbox($auth_user, $accountid);
  if($imap) {
    imap_delete($imap, $message_id);
    imap_expunge($imap);
    imap_close($imap);
    return true;
  }
  return false;
}
```

可以看到，该函数使用了几个IMAP函数。其中，我们还没有使用过的函数是imap_delete()和imap_expunge()。请注意，imap_delete()函数只是将邮件标记为删除。我们可以将任意数量的邮件标记为删除。imap_expunge()函数的调用才真正删除消息。

29.8　发送邮件

最后，我们来介绍如何发送邮件。该脚本程序中有多种方法可以完成此操作：用户可以发送一则新的邮件消息，回复或者转发邮件。我们来看看它们是如何工作的。

29.8.1　发送一则新消息

点击“New Message”按钮，用户可以选择发送一则新的邮件消息。这将触发“new-message”操作，该操作将执行index.php/case 'New-Message'脚本中的如下代码：

```
index.php/case 'new-message':
```

```
    display_new_message_form($_SESSION['auth_user'],
                             $to, $cc, $subject, $body);
break;
```

新消息表单仅仅是一个供发送邮件用的表单，如图29-8所示。实际上，该图显示的是回复邮件而非新建邮件，但是用到的表单都是相同的。接下来，我们将了解如何转发和回复邮件。

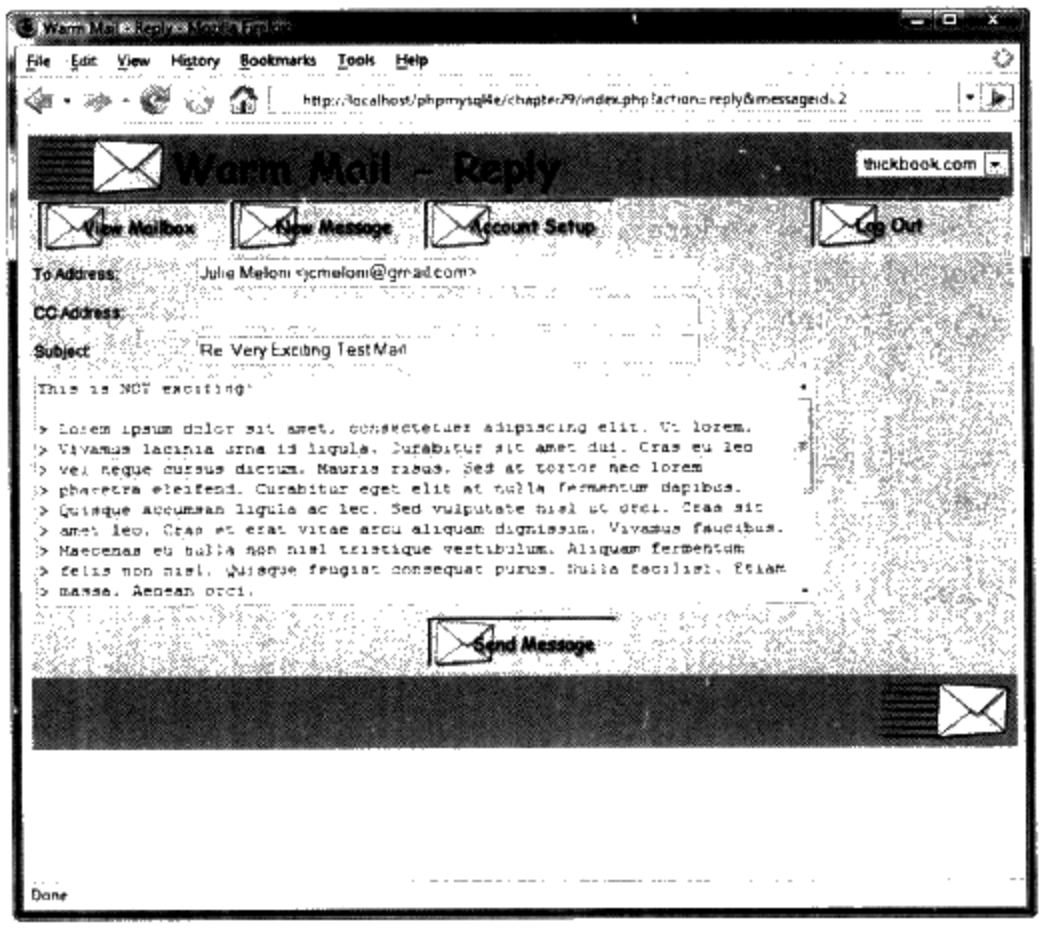

图29-8　你可以将邮件回复或转发给其他用户

点击“Send Message”按钮将调用“send-message”操作，该操作将执行index.php脚本中的如下代码：

```
case 'send-message':
   if(send_message($to, $cc, $subject, $message)) {
     echo "<p style=\"padding-bottom: 100px\">Message sent.</p>";
   } else {
     echo "<p style=\"padding-bottom: 100px\">Could not send message.</p>";
}
```

以上代码调用了send_message()函数，该函数真正执行邮件发送的操作。函数如程序清单29-12所示。

程序清单29-12　mail_fns.php的send_message()函数——发送用户键入的消息的函数

```
function send_message($to, $cc, $subject, $message) {
  // send one email via PHP

  if (!$conn=db_connect()) {
```

```
    return false;
  }
  $query = "select address from users where
username='".$_SESSION['auth_user']."'";

  $result = $conn->query($query);
  if (!$result) {
    return false;
  } else if ($result->num_rows==0) {
    return false;
  } else {
    $row = $result->fetch_object();
    $other = 'From: '.$row->address;
    if (!empty($cc)) {
       $other.="\r\nCc: $cc";
    }

    if (mail($to, $subject, $message, $other)) {
      return true;
    } else {
      return false;
    }
  }
}
```

可以看到，该函数使用了mail()函数来发送邮件，但是，首先该函数将从数据库中载入用户的电子邮件地址，并将其设置在待发邮件的From域。

29.8.2　回复或转发邮件

“Reply”、“Reply All”和“Forward”功能发送邮件的方法跟发送新邮件的方法相同。其不同之处在于，在将邮件发送表单显示给用户之前，系统已经自动填写好了某些域。回顾图29-8。我们要回复的消息正文已经用“>”符号缩进了，主题行以“Re:”为开始。同样，“Forward”和“Reply All”会预先填好接收者、主题行和缩进的消息正文。

完成这些操作的代码都是由index.php脚本的主体部分触发，如下所示：

```
case 'reply-all':
  //set cc as old cc line
  if(!$imap) {
    $imap = open_mailbox($_SESSION['auth_user'],
    $_SESSION['selected_account']);
  }

  if($imap) {
    $header = imap_header($imap, $messageid);

    if($header->reply_toaddress) {
```

```
            $to = $header->reply_toaddress;
        } else {
            $to = $header->fromaddress;
        }

        $cc = $header->ccaddress;
        $subject = "Re: ".$header->subject;
        $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
        imap_close($imap);

        display_new_message_form($_SESSION['auth_user'],
                                 $to, $cc, $subject, $body);
    }

break;

case 'reply':
    //set to address as reply-to or from of the current message
    if(!$imap) {
    $imap = open_mailbox($_SESSION['auth_user'],
    $_SESSION[ selected_account']);
    }

    if($imap) {
    $header = imap_header($imap, $messageid);
    if($header->reply_toaddress) {
        $to = $header->reply_toaddress;
    } else {
        $to = $header->fromaddress;
    }
    $subject = "Re: ".$header->subject;
    $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
    imap_close($imap);

    display_new_message_form($_SESSION['auth_user'],
                             $to, $cc, $subject, $body);
    }

break;

case 'forward':
    //set message as quoted body of current message
    if(!$imap) {
    $imap = open_mailbox($_SESSION['auth_user'],
    $_SESSION[ selected_account']);
    }

    if($imap) {
```

```
        $header = imap_header($imap, $messageid);
        $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
        $subject = "Fwd: ".$header->subject;
        imap_close($imap);

        display_new_message_form($_SESSION['auth_user'],
                                 $to, $cc, $subject, $body);
    }
  break;
```

可以看到，每一个选项都将建立相应的标题，必要时还会对标题进行格式化处理，然后再调用display_new_message_form()函数来建立表单。

这就是我们的Web邮件阅读系统的整个功能。

29.9 扩展这个项目

我们可以对该项目做出很多扩展和改进。我们可以观察平常使用的邮件阅读器并从中受到启发，下面是一些非常有用的补充。

- 为用户增加在该站点注册的功能。（关于用户注册功能，可重复利用第27章“建立用户身份验证机制和个性化设置”的一些代码。）
- 为用户提供具有多个邮件地址的功能。很多用户有多个电子邮件地址；例如可能一个用于个人，一个用于工作。通过把存储的电子邮件地址从用户表转移到账户表，用户能够使用多个地址。我们还需要在其他代码中进行有限的修改。发送邮件表单需要有一个下拉列表框来选择使用哪一个地址。
- 增加发送、接收、查看附件的功能。如果用户想要发送附件，就必须增加如第19章“与文件系统和服务器的交互”中讨论过的文件上传功能。而发送带有附件的邮件则在第30章“创建一个邮件列表管理器”中讨论。
- 增加地址簿功能。
- 增加网络新闻阅读功能。使用IMAP函数从一个NNTP服务器阅读新闻和从邮箱阅读邮件几乎是相同的。只需要在imap_open()函数调用中指定一个不同的端口号和协议。我们不是命名一个像INBOX一样的邮箱，而是命名一个想要获得新闻的新闻组。我们可以将此特性与第31章“创建一个Web论坛”项目中用到的创建主题功能结合起来，创建一个基于Web的线程化新闻阅读器。

29.10 下一章

在下一章中，我们将创建另外一个与电子邮件相联系的项目——在这个项目中，我们将创建一个支持向多个主题发送新闻信件的应用程序，它将新闻发送给站点的订阅用户。