第3章　基于AJAX的电子邮件处理

电子邮件（Email）无疑是当今互联网上最为流行的网络应用之一。本章节将介绍使用ASP.NET 2.0和ASP.NET AJAX技术，以及SQL Server 2005数据库共同实现基于AJAX的电子邮件处理的方法。本章介绍的基于AJAX的电子邮件处理应用程序是一个无刷新的电子邮件处理系统，包括发送电子邮件、接收电子邮件和邮箱管理等功能。该应用程序的主页面的效果图如图3.1所示。

图3.1　基于AJAX的电子邮件处理的效果图

3.1　基于AJAX的电子邮件处理应用程序构成

本小节介绍基于AJAX的电子邮件处理应用程序AjaxMail的组成、配置、数据库设计、系统参数设计等。

3.1.1　AjaxMail应用程序的组成

基于AJAX的电子邮件处理应用程序的名称为AjaxMail，它使用的数据库的名称为AjaxMailDB。在Visual Studio 2005的【解决方案资源管理器】面板中查看该应用程序，如图3.2所示。AjaxMail应用程序的组成元素说明如下：

- App_Code文件夹包含了2个类文件：ASPNETAJAXWeb.cs和Mail.cs。它们分别定义了AjaxMailSystem和Mail类。
- App_Themes文件夹包含了AjaxMail应用程序的主题和样式文件，如web.css、web.skin等。
- Bin文件夹包含了AjaxMail应用程序引入的程序集，如AjaxControlToolKit.dll、ASPNETAJAXWeb.ValidateCode.dll等。
- AddMailbox.aspx页面实现添加新邮箱文件夹的功能。

- Addresses.aspx页面实现配置邮件群发地址的功能。
- Default.aspx页面为邮箱主页面。
- Mailbox.aspx页面为邮箱管理页面。
- MailboxList.aspx页面为邮箱列表页面。
- MailTree.aspx页面以树型结构显示邮箱菜单操作树。
- ReadMail.aspx页面实现阅读邮件的功能。
- ReceiveMail.aspx页面实现接收邮件的功能。
- SendMail.aspx页面实现发送单个电子邮件的功能。
- SendMails.aspx页面实现群发邮件的功能。
- UpdateMailbox.aspx页面实现修改邮箱文件夹的功能。
- Web.config文件为AjaxMail应用程序的配置文件，它配置了数据库连接字符串、引用的程序集等属性。

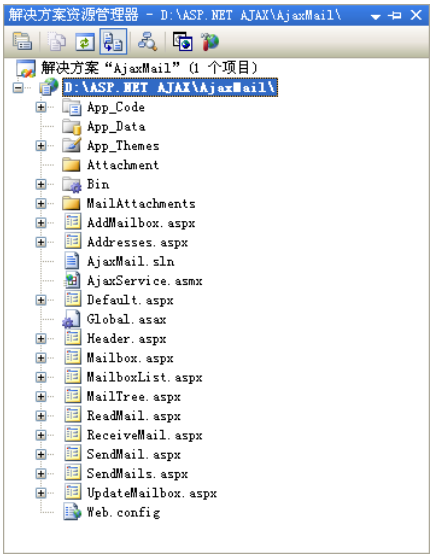

图3.2　在【解决方案资源管理器】面板中查看AjaxMail应用程序

3.1.2　AjaxMail应用程序的配置

AjaxMail应用程序的连接字符串放置在Web.config配置文件的<connectionString>元素中，具体代码如下：

```
<connectionStrings>
   <add name="SQLCONNECTIONSTRING"
      connectionString="data source=localhost;user id=sa;pwd=123456;
          database=AjaxEmailDB"
      providerName="System.Data.SqlClient"/>
</connectionStrings>
```

AjaxMail应用程序在Web.config配置文件的<controls>元素中还配置了与AJAX服务器控件相关的内容。该内容为AjaxControlToolkit.dll程序集中的控件提供了一个前缀字符串“ajaxToolkit”。因此，若该应用程序中的页面引用AjaxControlToolkit.dll程序集中的控件时，不再需要在每一个页面的HTML代码中添加“<Register>”代码，而是直接使用

“ajaxToolkit”前缀来引用AjaxControlToolkit.dll程序集中的控件。<controls>元素的具体代码如下：

```
<controls>
    <add namespace="AjaxControlToolkit" assembly="AjaxControlToolkit"
        tagPrefix="ajaxToolkit"/>
    ……
</controls>
```

3.1.3　数据库设计

基于AJAX的电子邮件处理的数据库的名称为AjaxMailDB，它包含4个表：Address、Attachment、Mail和Mailbox。其中，Address表保存用户地址的信息；Attachment表保存邮件附件的信息；Mail表保存邮件的信息；Mailbox保存邮箱的信息。

注意：邮件附件的文件保存在 Attachment 目录下，Attachment 表只保存邮件附件的链接地址。

1. Address表

Address表保存用户地址的信息，它包含的字段及其说明如表3-1所示。

表3-1　Address表

字段名	数据类型	字段说明	键引用	备注
ID	int	ID	PK	主键（自动增一）
Name	varchar(50)	地址名称		
Address	varchar(255)	地址		

2. Mailbox表

Mailbox表保存邮箱的信息，它包含的字段及其说明如表3-2所示。

表3-2　Mailbox表

字段名	数据类型	字段说明	键引用	备注
ID	int	ID	PK	主键（自动增一）
Name	varchar(50)	邮箱名称		

3. Mail表

Mail表保存邮件的信息，它包含的字段及其说明如表3-3所示。

表3-3　Mail表

字段名	数据类型	字段说明	键引用	备注
ID	int	ID	PK	主键（自动增一）
Title	varchar(200)	邮箱名称		
Body	text	邮件内容		
FromAddress	varchar(1000)	发送地址		
ToAddress	varchar(1000)	接收地址		
CCAddress	varchar(1000)	抄送地址		
ISHtmlFormat	bit	是否为HTML格式		
CreateDate	datetime	发送时间		
Size	int	大小		
Status	tinyint	状态		
MailboxID	int	所属邮箱	FK	引用Mailbox表的ID列。

4. Attachment表

Attachment表保存邮件附件的信息，它包含的字段及其说明如表3-4所示。

表3-4 Attachment表

字段名	数据类型	字段说明	键引用	备注
ID	int	ID	PK	主键（自动增一）
Name	varchar(50)	附件名称		
Url	varchar(255)	链接地址		
MailID	int	所属邮件	FK	引用Mail表的ID列。

综合上述，AjaxMailDB数据库中表的关系设计图如图3.3所示。

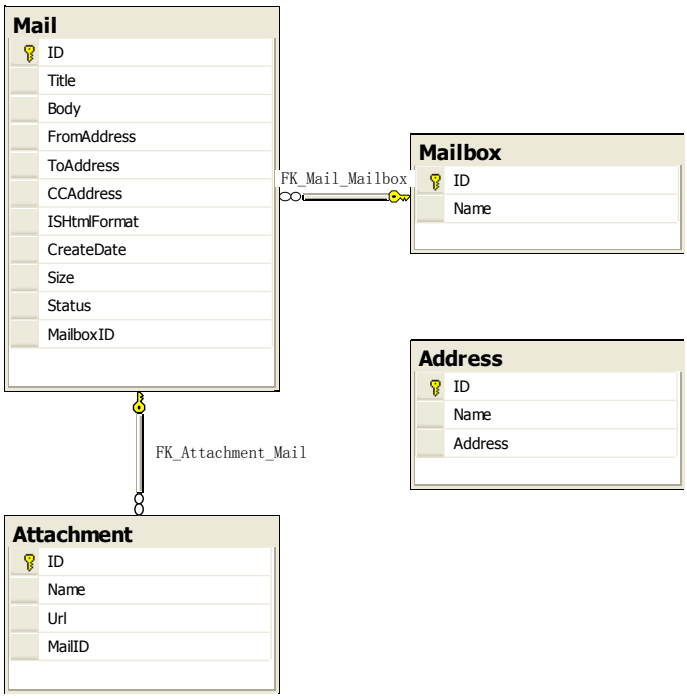

图3.3 AjaxMailDB数据库中表的关系设计图

3.1.4 系统参数设计

基于AJAX的电子邮件处理应用程序在Global.asax.cs文件中配置了两个重要参数：发送和接收邮件的服务器的IP地址和端口。这两个参数分别使用静态变量MAILSERVERHOST和MAILSERVERPORT表示。其中，邮件服务器端口的默认值为25。配置这个两个参数的程序代码如下：

```
<script runat="server">
/// <summary>
/// 邮件服务器的IP地址
/// </summary>
public static string MAILSERVERHOST = "服务器的IP地址或主机名";
/// <summary>
/// 邮件服务器的端口
```

```
    /// </summary>
    public static int MAILSERVERPORT = 25;
    </script>
```

3.2　发送电子邮件

本小节实现发送电子邮件的功能，如发送邮件界面设计、智能邮件地址提示、发送单个邮件、配置邮件群发地址、群发邮件等。

3.2.1　数据访问层设计

基于AJAX的电子邮件处理应用程序的数据访问层由Mail类实现，该类包含在ASPNETAJAXWeb.AjaxMail命名空间中。Mail类的代码定义在Mail.cs类文件中。该文件引入了多个新的命名空间，如System.Data.SqlClient、System.Net、System.Net.Mail等。下面的程序代码声明了Mail类。其中，该类中的方法代码已经省略。

```
using System;
using System.Data;
using System.Configuration;
using System.Data.SqlClient;
///引入新的命名空间
using System.Net;
using System.Net.Mail;
using System.IO;
using System.Text;
using System.Net.Sockets;
using System.Web;
using System.Collections;
namespace ASPNETAJAXWeb.AjaxMail
{
    public class Mail
    {
        public Mail(){}
        ……
    }
}
```

与邮件及其附件相关的数据访问层共实现8个功能。实现这些功能的方法（定义在Mail类中）具体描述如下：

- public DataSet GetMails()，获取所有邮件的信息。
- public DataSet GetMailByMailBox(int mailboxID)，获取指定邮箱中的邮件的信息。
- public SqlDataReader GetSingleMail(int mailID)，获取指定邮件的信息。
- public int AddMail(string title,string body,string fromAddress,string toAddress, string ccAddress, bool isHtmlFormat,int size)，添加邮件到数据库中。
- public int DeleteMail(int mailID)，删除指定的邮件。
- public int AddMailAttachment(string name,string url,int mailID)，添加邮件的附件到数据库中。
- public DataSet GetAttachmentByMail(int mailID)，获取指定邮件的附件。

- public DataSet GetAddresses()，获取所有地址信息。

其中，mailboxID参数指定邮箱的ID，mailID参数指定邮件的ID，title、body、fromAddress、toAddress、ccAddress、isHtmlFormat、size参数分别指定邮件的发送地址、接收地址、抄送地址、是否为HTML格式邮件的标记、邮件大小，name和url参数指定邮件附件的名称和链接地址。

在下述程序代码中，GetMails()方法获取所有邮件的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取所有邮件的信息的SQL语句“SELECT * FROM Mail”。

（4）创建获取数据的SqlDataAdapter对象da。

（5）打开数据库连接，并获取数据。获取的结果保存在DataSet对象ds中。

（6）如果上述操作成功，则返回ds。

```
public DataSet GetMails()
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "SELECT * FROM Mail";
    ///创建SqlDataAdapter
    SqlDataAdapter da = new SqlDataAdapter(cmdText,con);
    ///定义DataSet
    DataSet ds = new DataSet();
    try
    {   ///打开连接
        con.Open();
        ///填充数据
        da.Fill(ds,"DataTable");
    }
    catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
    finally{con.Close();}                                      ///关闭连接
    return ds;
}
```

在下述程序代码中，GetMailByMailBox(int mailboxID)方法获取指定邮箱中的邮件的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取指定邮箱中的邮件的信息的SQL语句“SELECT M.*,(SELECT COUNT(*) FROM Attachment AS A WHERE A.MailID=M.ID) AS AttachmentCount FROM Mail AS M WHERE MailboxID = @MailboxID ORDER BY CreateDate DESC”。其中，AttachmentCount字段表示邮件的附件数量。

（4）创建获取数据的SqlDataAdapter对象da。

（5）打开数据库连接，并获取数据。获取的结果保存在DataSet对象ds中。

（6）如果上述操作成功，则返回ds。

```
public DataSet GetMailByMailBox(int mailboxID)
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "SELECT M.*,(SELECT COUNT(*) FROM Attachment AS A
        WHERE A.MailID=M.ID) AS AttachmentCount FROM Mail AS M
        WHERE MailboxID = @MailboxID ORDER BY CreateDate DESC";
    ///创建SqlDataAdapter
    SqlDataAdapter da = new SqlDataAdapter(cmdText,con);
    ///创建参数并赋值
    da.SelectCommand.Parameters.Add("@MailboxID",SqlDbType.Int,4);
    da.SelectCommand.Parameters[0].Value = mailboxID;
    ///定义DataSet
    DataSet ds = new DataSet();
    try
    {   ///打开连接
        con.Open();
        ///填充数据
        da.Fill(ds,"DataTable");
    }
    catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
    finally{con.Close();}                                      ///关闭连接
    return ds;
}
```

在下述程序代码中，GetSingleMail(int mailID)方法获取指定邮件的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取指定邮件的SQL语句“SELECT * FROM Mail WHERE ID=@ID”。其中，@ID参数的值由mailID参数指定。

（4）创建获取数据的SqlCommand对cmd。

（5）打开数据库连接，并获取数据。获取的结果保存在SqlDataReader对象dr中。

（6）如果上述操作成功，则返回dr。

```
public SqlDataReader GetSingleMail(int mailID)
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "SELECT * FROM Mail WHERE ID=@ID";
    ///创建SqlCommand
```

```
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = mailID;
        ///定义SqlDataReader
        SqlDataReader dr;
        try
        {   ///打开连接
            con.Open();
            ///读取数据
            dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        return dr;
    }
```

在下述程序代码中，AddMail(string title,string body,string fromAddress,string toAddress, string ccAddress,bool isHtmlFormat,int size)方法将邮件添加到数据库中。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建插入照片的SQL语句“INSERT INTO Mail(Title,Body,FromAddress,ToAddress, CCAddress,ISHtmlFormat,CreateDate,Size,Status,MailboxID) VALUES(@Title,@Body, @FromAddress,@ToAddress, @CCAddress,@ISHtmlFormat,GETDATE(),@Size,0,2) SET @ID = @@Identity”。其中，@Title、@Body、@FromAddress、@ToAddress、@CCAddress、@ISHtmlFormat,、@Sizetitle参数分别指定邮件的标题、内容、发送地址、接收地址、抄送地址、是否为HTML格式邮件的标记、邮件大小。

（4）创建执行插入操作的SqlCommand对cmd。

（5）打开数据库连接，并执行插入操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
    public int AddMail(string title,string body,string fromAddress,
        string toAddress,string ccAddress,bool isHtmlFormat,int size)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
            "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "INSERT INTO Mail(Title,Body,FromAddress,ToAddress,
            CCAddress,ISHtmlFormat,CreateDate,Size,Status,MailboxID)"
            + "VALUES(@Title,@Body,@FromAddress,@ToAddress,
                @CCAddress,@ISHtmlFormat,GETDATE(),@Size,0,2)
                SET @ID = @@Identity";
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@Title",SqlDbType.VarChar,200);
```

```
        cmd.Parameters.Add("@Body",SqlDbType.Text);
        cmd.Parameters.Add("@FromAddress",SqlDbType.VarChar,1000);
        cmd.Parameters.Add("@ToAddress",SqlDbType.VarChar,1000);
        cmd.Parameters.Add("@CCAddress",SqlDbType.VarChar,1000);
        cmd.Parameters.Add("@ISHtmlFormat",SqlDbType.Bit,1);
        cmd.Parameters.Add("@Size",SqlDbType.Int,4);
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = title;
        cmd.Parameters[1].Value = body;
        cmd.Parameters[2].Value = fromAddress;
        cmd.Parameters[3].Value = toAddress;
        cmd.Parameters[4].Value = ccAddress;
        cmd.Parameters[5].Value = isHtmlFormat;
        cmd.Parameters[6].Value = size;
        cmd.Parameters[7].Direction = ParameterDirection.Output;
        int result = -1;
        try
        {   ///打开连接
           con.Open();
           ///操作数据
           result = cmd.ExecuteNonQuery();
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return (int)cmd.Parameters[7].Value;
    }
```

在下述程序代码中，DeleteMail(int mailID)方法将删除指定邮件的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建删除指定邮件的SQL语句“DELETE Mail WHERE ID = @ID”。其中，@ID参数的值由mailID参数指定。

（4）创建执行删除操作的SqlCommand对cmd。

（5）打开数据库连接，并执行删除操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
    public int DeleteMail(int mailID)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
           "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "DELETE Mail WHERE ID = @ID";
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
```

```
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = mailID;
        int result = -1;
        try
        {   ///打开连接
            con.Open();
            ///操作数据
            result = cmd.ExecuteNonQuery();
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return result;
    }
```

在下述程序代码中，AddMailAttachment(string name,string url,int mailID)方法将照片添加到数据库中。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建插入照片的SQL语句“INSERT INTO Attachment(Name,Url,MailID)VALUES(@Name,@Url,@MailID)”。其中，@Name、@Url和@MailID参数的值分别由name、url和mailID参数指定。它们分别表示附件的名称、链接地址和所属邮件的ID值。

（4）创建执行插入操作的SqlCommand对cmd。

（5）打开数据库连接，并执行插入操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
    public int AddMailAttachment(string name,string url,int mailID)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
            "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "INSERT INTO Attachment(Name,Url,MailID)"
            + "VALUES(@Name,@Url,@MailID)";
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@Name",SqlDbType.VarChar,200);
        cmd.Parameters.Add("@Url",SqlDbType.VarChar,255);
        cmd.Parameters.Add("@MailID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = name;
        cmd.Parameters[1].Value = url;
        cmd.Parameters[2].Value = mailID;
        int result = -1;
        try
        {   ///打开连接
            con.Open();
            ///操作数据
```

```
            result = cmd.ExecuteNonQuery();
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return result;
    }
```

在下述程序代码中，GetAttachmentByMail(int mailID)方法获取指定邮件的附件。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取指定邮件的附件的SQL语句“SELECT Attachment.* FROM Attachment WHERE MailID = @MailID”。其中，@MailID参数的值由mailID参数指定，它表示邮件的ID值。

（4）创建获取数据的SqlDataAdapter对象da。

（5）打开数据库连接，并获取数据。获取的结果保存在DataSet对象ds中。

（6）如果上述操作成功，则返回ds。

```
    public DataSet GetAttachmentByMail(int mailID)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
            "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "SELECT Attachment.* FROM Attachment
            WHERE MailID = @MailID";
        ///创建SqlDataAdapter
        SqlDataAdapter da = new SqlDataAdapter(cmdText,con);
        ///创建参数并赋值
        da.SelectCommand.Parameters.Add("@MailID",SqlDbType.Int,4);
        da.SelectCommand.Parameters[0].Value = mailID;
        ///定义DataSet
        DataSet ds = new DataSet();
        try
        {   ///打开连接
            con.Open();
            ///填充数据
            da.Fill(ds,"DataTable");
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return ds;
    }
```

在下述程序代码中，GetAddresses()方法获取所有地址的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。
（3）创建获取所有地址的信息的SQL语句“SELECT * FROM Address”。
（4）创建获取数据的SqlDataAdapter对象da。
（5）打开数据库连接，并获取数据。获取的结果保存在DataSet对象ds中。
（6）如果上述操作成功，则返回ds。

```
public DataSet GetAddresses()
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "SELECT * FROM Address";
    ///创建SqlDataAdapter
    SqlDataAdapter da = new SqlDataAdapter(cmdText,con);
    ///定义DataSet
    DataSet ds = new DataSet();
    try
    {   ///打开连接
        con.Open();
        ///填充数据
        da.Fill(ds,"DataTable");
    }
    catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
    finally{con.Close();}                                      ///关闭连接
    return ds;
}
```

3.2.2 发送邮件界面设计

发送邮件由SendMail.aspx页面实现，SendMail.aspx.cs文件为它的代码隐藏文件。该页面提供了输入邮件的收件人、主题、抄送地址、内容、上载邮件附件，以及发送邮件等功能。SendMail.aspx页面的效果图如图3.4所示。

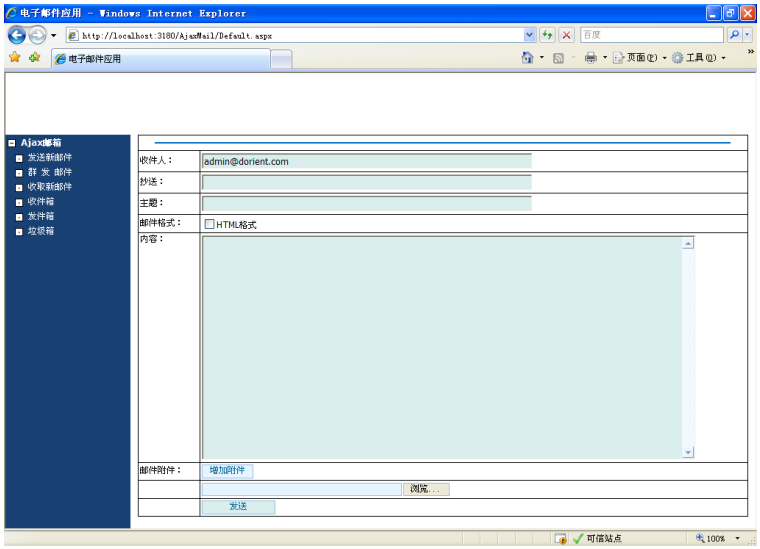

图3.4　SendMail.aspx页面的效果图

1. 界面设计

在下述程序代码中，SendMail.aspx页面添加了1个TextBox控件，用来输入邮件的收件人，其ID属性的值为tbTo。该控件使用了1个TextBoxWatermarkExtender控件和3个ValidatorCalloutExtender控件，它们的ID属性的值分别为wmeName、vceNameBlank、vceNameValue和vceNameRegex。其中，wmeName控件为tbTo控件显示水印值“请输入收件人地址”。vceNameBlank、vceNameValue和vceRevName控件分别以提示样式显示验证结果。

另外，tbTo控件使用2个RequiredFieldValidator控件和1个RegularExpressionValidator控件，它们的ID属性的值分别为rfNameBlank、rfNameValue和revName。其中，rfNameBlank控件验证tbTo控件的内容不能为空验证，即用户输入收件人地址不能为空。rfNameValue控件验证tbTo控件的内容不能等于“请输入收件人地址”（wmeName水印控件的水印值）。revName控件验证tbTo控件的内容（收件人地址）必须符合电子邮件的格式，它的正则表达式为“\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*”

```
    收件人：<asp:TextBox ID="tbTo" runat="server" SkinID="tbSkin" Width="60%"
        MaxLength="255"></asp:TextBox>
    <asp:RequiredFieldValidator ID="rfNameBlank" runat="server"
        ControlToValidate="tbTo" Display="none"
        ErrorMessage="收件人地址不能为空！"></asp:RequiredFieldValidator>
    <asp:RequiredFieldValidator ID="rfNameValue" runat="server"
        ControlToValidate="tbTo" Display="none" InitialValue="请输入收件人地址"
        ErrorMessage="收件人地址不能为空！"></asp:RequiredFieldValidator>
    <asp:RegularExpressionValidator ID="revName" runat="server"
        ControlToValidate="tbTo" Display="none"
        ErrorMessage="收件人地址的长度最大为255，请重新输入。"
        ValidationExpression=".{1,255}"></asp:RegularExpressionValidator>
    <ajaxToolkit:TextBoxWatermarkExtender ID="wmeName" runat="server"
        TargetControlID="tbTo" WatermarkText="请输入收件人地址"
```

```
        WatermarkCssClass="Watermark">
        </ajaxToolkit:TextBoxWatermarkExtender>
    <ajaxToolkit:ValidatorCalloutExtender ID="vceNameBlank" runat="server"
        TargetControlID="rfNameBlank" HighlightCssClass="Validator">
        </ajaxToolkit:ValidatorCalloutExtender>
    <ajaxToolkit:ValidatorCalloutExtender ID="vceNameValue" runat="server"
        TargetControlID="rfNameValue" HighlightCssClass="Validator">
        </ajaxToolkit:ValidatorCalloutExtender>
    <ajaxToolkit:ValidatorCalloutExtender ID="vceNameRegex" runat="server"
        TargetControlID="revName" HighlightCssClass="Validator">
        </ajaxToolkit:ValidatorCalloutExtender>
```

在下述程序代码中，SendMail.aspx页面添加了3个TextBox控件，它们的ID属性的值分别为tbCC、tbTitle和tbBody。其中，tbCC控件用来输入邮件的抄送地址；tbTitle控件用来输入邮件的主题；tbBody用来输入邮件的内容。

注意：HtmlCB 控件可以设置邮件的格式，如果它被选择，则表示邮件为 HTML 格式，否则为纯文本格式。

```
    抄送：<asp:TextBox ID="tbCC" runat="server" SkinID="tbSkin" Width="60%"
        MaxLength="50"></asp:TextBox>
    主题：<asp:TextBox ID="tbTitle" runat="server" SkinID="tbSkin" Width="60%"
        MaxLength="50"></asp:TextBox>
    邮件格式：<input id="HtmlCB" type="checkbox" runat="server" class="GbText"
        />HTML格式
    内容：<asp:TextBox ID="tbBody" runat="server" SkinID="tbSkin" Width="90%"
        Height="300px" TextMode="MultiLine"></asp:TextBox>
```

每单击SendMail.aspx页面中的【增加附件】按钮一次，就可以在该页面上添加一个上载文件的HTML按钮控件。该功能由javascript函数addFile()实现。其中，新增加的上载文件的HTML按钮控件追加在ID属性的值为FileList的子列表中。

注意：SendMail.aspx 页面中的 form1 表单 enctype 属性的值必须设置为“multipart/form-data”，否则不能实现同时发送多个附件的功能。

```
    邮件附件：<input type="button" value="增加附件" class="Button"
        onclick="addFile()" />
    <p id="FileList"><input id="File1" type="file" runat="server" size="50"
        name="File" class="Button" /></p>
    <script language="javascript" type="text/javascript">
    function addFile()
    {
        var filebutton = '<br><input type="file" size="50" name="File"
            class="Button" />';
        document.getElementById('FileList').insertAdjacentHTML(
            "beforeEnd",filebutton);
    }
    </script>
```

下述程序代码声明了1个Button控件，ID属性的值为btnCommit。该控件实现发送邮件的功能，并将发送的邮件保存到数据库中。

```
    <asp:Button ID="btnCommit" runat="server" Text="发送" SkinID="btnSkin"
        Width="100px" OnClick="btnCommit_Click" />
```

下述程序代码声明了SendMail.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="SendMail.aspx.cs"
        Inherits="SendMail" StylesheetTheme="ASPNETAjaxWeb" %>
    <head id="Head1" runat="server"><title>发送邮件</title></head>
    <asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

综合上述，SendMail.aspx页面的设计界面如图3.5所示。

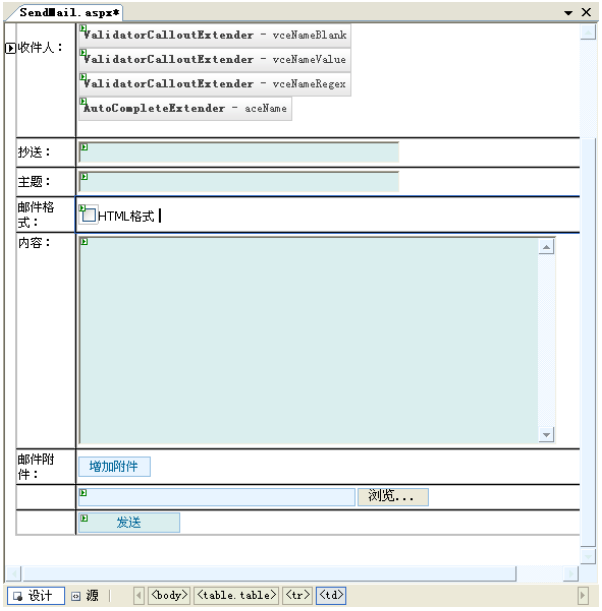

图3.5　SendMail.aspx页面的设计界面

3.2.3　智能邮件地址提示

当用户在SendMail.aspx页面中的【收件人】输入框中输入地址时，该输入框会显示一个下拉列表，用户可以在该下拉列表中选择收件人的地址。这种技术被称为“智能邮件地址提示”，效果如图3.6所示。

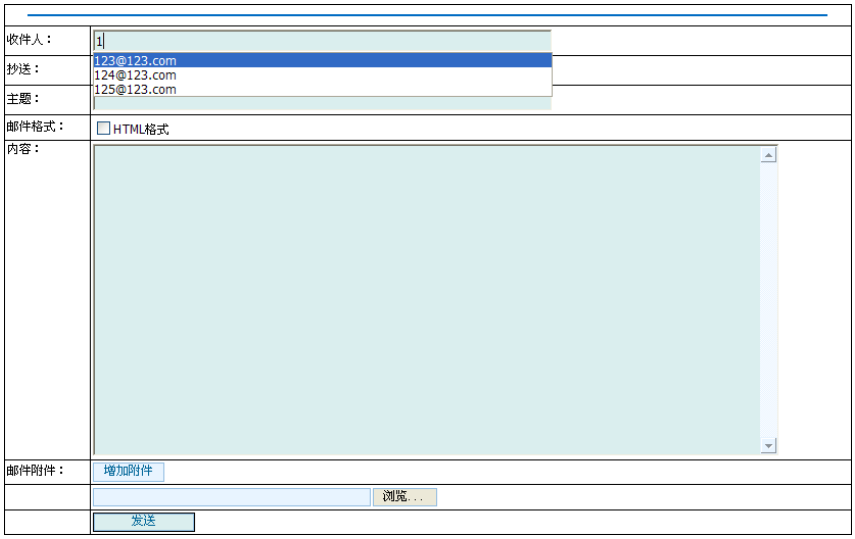

图3.6　SendMail.aspx页面的智能邮件地址提示效果图

在下述程序代码中，SendMail.aspx页面中的tbTo控件使用了1个AutoCompleteExtender控件（ID属性的值为aceName）实现智能提示功能。当用户在tbTo控件输入地址时，tbTo控件将显示一个下拉列表，用户可以在该下拉列表中选择收件人的地址。声明aceName控件的程序代码如下：

```
<ajaxToolkit:AutoCompleteExtender ID="aceName" runat="server"
   TargetControlID="tbTo" ServicePath="AjaxService.asmx"
   ServiceMethod="GetAddressList" MinimumPrefixLength="1"
   CompletionInterval="100" CompletionSetCount="20">
   </ajaxToolkit:AutoCompleteExtender>
```

tbTo控件使用了名称为AjaxService.asmx的Web服务，并使用GetAddressList(string prefixText, int count)方法获取智能提示信息（由收件人地址组成，并且这些地址保存在数据库的Address表中）。AjaxService.asmx Web服务引入了4个新的命名空间：System.Data、System.Web.Script.Services、ASPNETAJAXWeb.AjaxMail和AjaxControlToolkit。具体的程序代码如下：

```
///引入新的命名空间
using System.Data;
using System.Web.Script.Services;
using AjaxControlToolkit;
using ASPNETAJAXWeb.AjaxMail;
```

AjaxService.asmx Web服务定义了名称为AjaxService的类，该类将作为Web服务发布。为了使得该Web服务能够被AutoCompleteExtender控件aceName使用，还为其添加了一个脚本服务属性，即System.Web.Script.Services.ScriptService()。AjaxService类还定义了一个静态变量autoCompleteAddressList，它用来保存提示信息。定义AjaxService类的程序代码如下：

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService()]    ///添加脚本服务
public class AjaxService : System.Web.Services.WebService
{
```

```
    public static string[] autoCompleteAddressList = null;
    public AjaxService (){}
    ……
}
```

在下述程序代码中，GetAddressList(string prefixText,int count)方法返回一个字符串数组，它的内容就是tbTo控件的提示信息。其中，prefixText参数为用户输入的关键字，count参数指定每次返回提示信息的最大数量。GetAddressList(string prefixText,int count)方法实现的具体步骤如下：

（1）判断prefixText和count参数是否合法。如果不合法，则中止该方法。

（2）如果autoCompleteAddressList变量的值为空，则调用Mail类的GetAddresses()方法获取所有地址信息，并将地址添加到autoCompleteAddressList变量中，最后还对该变量进行排序。

（3）在autoCompleteAddressList变量中搜索prefixText参数出现的索引值。如果不存在prefixText参数，则把索引设置为0。

（4）从autoCompleteAddressList变量中搜索以prefixText参数开头的文件名称，并保存搜索结果的在autoCompleteFileList变量中的索引。

（5）根据（4）步骤中的索引值，将autoCompleteAddressList变量中的符合条件的内容复制到matchResultList变量中，并返回该变量。

```
    [System.Web.Services.WebMethod()]
    [System.Web.Script.Services.ScriptMethod()]
    public string[] GetAddressList(string prefixText,int count)
    {   ///检测参数是否为空
        if(string.IsNullOrEmpty(prefixText) == true
            || count <= 0) return null;
        if(autoCompleteAddressList == null)
        {   ///从数据库中获取所有地址
            Mail mail = new Mail();
            DataSet ds = mail.GetAddresses();
            if(ds == null || ds.Tables.Count <= 0
                || ds.Tables[0].Rows.Count <= 0) return null;
            ///将地址保存到临时数组中
            string[] tempAddressList = new string[ds.Tables[0].Rows.Count];
            for(int i = 0; i < ds.Tables[0].Rows.Count; i++)
            {
                tempAddressList[i]
                    = ds.Tables[0].Rows[i]["Address"].ToString();
            }
            ///对数组进行排序
            Array.Sort(tempAddressList,new CaseInsensitiveComparer());
            autoCompleteAddressList = tempAddressList;
        }
        ///定位二叉树搜索的起点
        int index = Array.BinarySearch(
            autoCompleteAddressList,prefixText,
            new CaseInsensitiveComparer());
        if(index < 0){index = ~index;}   ///修正起点
        ///搜索符合条件的地址
```

```
            int matchCount = 0;
            for(matchCount = 0; matchCount < count
                && matchCount + index < autoCompleteAddressList.Length;
                matchCount++)
            {   ///查看开头字符串相同的项
                if(autoCompleteAddressList[index + matchCount].StartsWith(
                    prefixText,StringComparison.CurrentCultureIgnoreCase)
                    == false){break;}
            }
            ///处理搜索结果
            string[] matchResultList = new string[matchCount];
            if(matchCount > 0)
            {   ///复制搜索结果
                Array.Copy(autoCompleteAddressList,
                    index,matchResultList,0,matchCount);
            }
            return matchResultList;
        }
```

3.2.4 发送单个邮件

发送单个邮件功能在SendMail.aspx页面的代码隐藏文件SendMail.aspx.cs中实现。用户单击SendMail.aspx页面中的【发送】按钮（btnCommit控件）时，将触发btnCommit控件的Click事件：btnCommit_Click(object sender,EventArgs e)。该事件实现发送电子邮件的功能。

为了能够实现发送单个邮件的功能，SendMail.aspx.cs文件引入了4个新的命名空间：System.IO、System.Text、System.Net.Mail和ASPNETAJAXWeb.AjaxMail。其中，发送邮件的类被包含在System.Net.Mail命名空间中。程序代码如下：

```
    ///引入新的命名空间
    using ASPNETAJAXWeb.AjaxMail;
    using System.Net.Mail;
    using System.Text;
    using System.IO;
```

发送单个邮件是一个非常复杂的过程，下面介绍这一个过程的具体实现步骤：

（1）创建保存邮件的MailMessage类的实例，并指定实例的名称为mail。其中，size变量将计算邮件的大小。

```
    protected void btnCommit_Click(object sender,EventArgs e)
    {   ///创建保存邮件的MailMessage类的示例
        MailMessage mail = new MailMessage();
        int size = 0;
```

（2）添加发件人地址到mail实例中，同时修正邮件的大小。在此，发件人地址固定设置为“admin@dorient.com”。

```
        ///添加发件人地址
        string from = "admin@dorient.com";
        mail.From = new MailAddress(from);
        size += mail.From.Address.Length;
```

（3）添加收件人地址到mail实例中，同时修正邮件的大小。由于用户在输入收件人地址时，可以同时输入多个地址，并且使用分号隔开。因此，需要将用户输入的收件人地址

进行分割，然后逐个将分割后的地址添加到mail实例中。

```
///添加收件人地址
string split = ";";
string[] toList = tbTo.Text.Trim().Split(split.ToCharArray());
for(int i = 0; i < toList.Length; i++)
{
   mail.To.Add(toList[i].Trim());
}
size += tbTo.Text.Length;
```

（4）添加抄送地址到mail实例中，同时修正邮件的大小。由于用户在输入抄送地址时，可以同时输入多个地址，并且使用分号隔开。因此，需要将用户输入的抄送地址进行分割，然后逐个将分割后的地址添加到mail实例中。

```
///添加抄送地址;
string[] ccList = tbCC.Text.Trim().Split(split.ToCharArray());
for(int i = 0; i < ccList.Length; i++)
{
   if(ccList[i].Trim().Length > 0){mail.CC.Add(ccList[i].Trim());}
}
size += tbCC.Text.Length;
```

（5）添加邮件主题到mail实例中，并设置邮件主题的编码为“UTF8”，同时修正邮件的大小。其中，邮件主题是用户在tbTitle控件中输入的内容。

```
///添加邮件主题
mail.Subject = tbTitle.Text.Trim();
mail.SubjectEncoding = Encoding.UTF8;
size += mail.Subject.Length;
```

（6）添加邮件内容到mail实例中，并设置邮件内容的编码为“UTF8”，同时修正邮件的大小。其中，邮件内容是用户在tbBody控件中输入的内容。

```
///添加邮件内容
mail.Body = tbBody.Text;
mail.BodyEncoding = Encoding.UTF8;
size += mail.Body.Length;
```

（7）设置mail实例的HTML格式。该格式由HtmlCB控件指定，如果HtmlCB控件被选中，则该邮件的格式为HTML格式。

注意：如果当邮件为 HTML 格式时，邮件会增加许多 HTML 元素从而把邮件构建为 HTML 格式，这一操作将增加邮件的大小。因此，如果当邮件为 HTML 格式时，需要修正邮件的大小。在此，把邮件的大小增加 100（该值不是一个精确值，它仅仅是一个估计值）。

```
mail.IsBodyHtml = HtmlCB.Checked;
if(mail.IsBodyHtml == true){size += 100;}
```

（8）添加邮件附件到mail实例中，同时修正邮件的大小。由于用户可以同时指定多个邮件附件，因此需要逐个处理每一个附件，并将这些附件添加到mail实例的Attachments属性（即邮件的附件列表）中。

```
///添加邮件附件
HttpFileCollection fileList = HttpContext.Current.Request.Files;
for(int i = 0; i < fileList.Count; i++)
{   ///添加单个附件
```

```
            HttpPostedFile file = fileList[i];
            if(file.FileName.Length <= 0 || file.ContentLength <= 0)
            {continue;}
            Attachment attachment = new Attachment(file.FileName);
            mail.Attachments.Add(attachment);
            size += file.ContentLength;
        }
```

（9）创建发送邮件的SmtpClient类的实例client，并设置发送邮件的服务器和端口。其中，发送邮件的服务器和端口分别保存在Global.asax文件的以下两个静态变量中。

- ASP.global_asax.MAILSERVERHOST，指定邮件服务器的IP地址或名称。
- ASP.global_asax.MAILSERVERPORT，指定邮件服务器的端口，默认值为25。

```
        try
        {   ///设置邮件服务器
            SmtpClient client = new SmtpClient();
            client.Host = ASP.global_asax.MAILSERVERHOST;
            client.Port = ASP.global_asax.MAILSERVERPORT;
```

（10）设置SmtpClient类的实例client的递送方式为SmtpDeliveryMethod.Network，并调用Send()方法发送邮件。

```
            ///发送邮件
            client.DeliveryMethod = SmtpDeliveryMethod.Network;
            client.UseDefaultCredentials = false;
            client.Send(mail);
```

（11）如果邮件发送成功，调用Mail类的AddMail(string title,string body,string fromAddress, string toAddress, string ccAddress, bool isHtmlFormat,int size)方法将刚才发送的邮件保存到发件箱中。其中，上述方法将返回邮件的ID值，并保存在mailID变量中。

```
            ///保存发送的邮件
            Mail mailbox = new Mail();
            int mailID = mailbox.AddMail(
                mail.Subject,mail.Body,from,tbTo.Text,tbCC.Text,
                mail.IsBodyHtml,size);
            if(mailID <= 0) return;
```

（12）如果保存邮件到发件箱中成功，则把该邮件的附件上载到服务器。同时，调用Mail类的AddMailAttachment(string name,string url,int mailID)方法将附件的信息保存到数据库中。

```
            ///保存发送邮件的附件
            for(int i = 0; i < fileList.Count; i++)
            {   ///添加单个附件
                HttpPostedFile file = fileList[i];
                if(file.FileName.Length <= 0 || file.ContentLength <= 0)
                {continue;}
                ///创建基于时间的文件名称
                string fileName = AjaxMailSystem.CreateDateTimeString();
                string extension = Path.GetExtension(file.FileName);
                ///构建保存文件位置的路径
                string url = "MailAttachments/" + fileName + extension;
                ///映射为物理路径
                string fullPath = Server.MapPath(url);
```

```
                ///保存附件到硬盘中
                file.SaveAs(fullPath);
                ///保存发送邮件的附件
                mailbox.AddMailAttachment(
                    Path.GetFileName(file.FileName),url,mailID);
            }
```

（13）如果上述操作步骤均成功，即意味着发送当前邮件成功，则重定向到显示发件箱信息的Mailbox.aspx页面。

```
    注意：发件箱页面 Mailbox.aspx 的 MailboxID 参数的值必须为 2。
            ///如果发送邮件成功，重定向到发件箱
            Response.Redirect("~/Mailbox.aspx?MailboxID=2");
        }
        catch{}
    }
```

3.2.5　配置邮件群发地址

Addresses.aspx和SendMails.aspx页面共同实现群发邮件的功能。其中，Addresses.aspx页面配置群发地址，并将地址保存到Session对象中。SendMails.aspx页面从Session对象中获取群发地址，然后对这些地址实行群发邮件操作。

配置邮件群发地址由Addresses.aspx页面实现，Addresses.aspx.cs文件为它的代码隐藏文件。该页面提供了配置邮件群发地址的功能，然后将配置好的邮件地址保存到Session对象中。Addresses.aspx页面的效果图如图3.7所示。

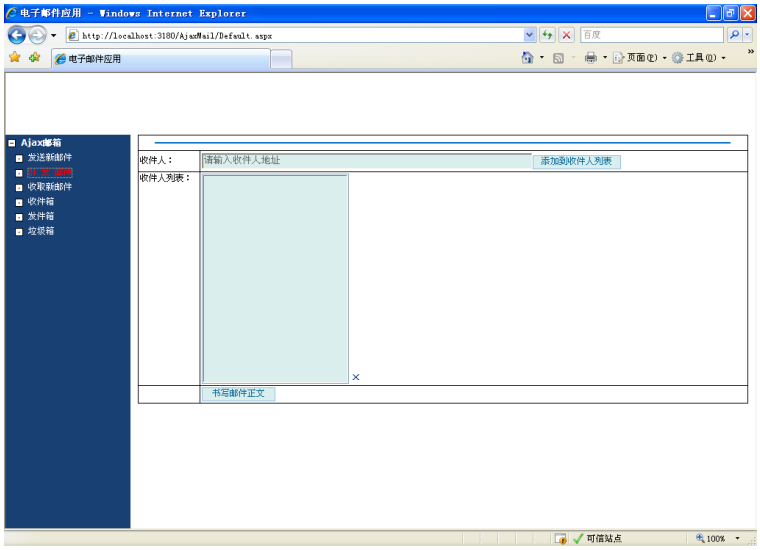

图3.7　Addresses.aspx页面的效果图

1. 界面设计

在下述程序代码中，Addresses.aspx页面添加了1个TextBox控件和1个Button控件，它们的ID属性的值分别为tbName和btnAdd。tbName控件供用户输入地址，单击【添加到收件人

列表】按钮（btnAdd控件）可以将该地址添加到收件人列表（即lbAddress控件，将在后续进行详细介绍）中。

tbTo控件使用了1个TextBoxWatermarkExtender控件和3个ValidatorCalloutExtender控件，它们的ID属性的值分别为wmeName、vceNameBlank、vceNameValue和vceNameRegex。其中，wmeName控件为tbTo控件显示水印值“请输入收件人地址”。vceNameBlank、vceNameValue和vceRevName控件分别以提示样式显示验证结果。

另外，tbTo控件还使用2个RequiredFieldValidator控件和1个RegularExpressionValidator控件，它们的ID属性的值分别为rfNameBlank、rfNameValue和revName。其中，rfNameBlank控件验证tbTo控件的内容不能为空验证，即用户输入收件人地址不能为空。rfNameValue控件验证tbTo控件的内容不能等于“请输入收件人地址”（wmeName水印控件的水印值）。revName控件验证tbTo控件的内容（收件人地址）必须符合电子邮件的格式，它的正则表达式为“\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*”。

```
<asp:UpdatePanel ID="up" runat="server"><ContentTemplate>
收件人：<asp:TextBox ID="tbTo" runat="server" SkinID="tbSkin" Width="60%"
   MaxLength="255"></asp:TextBox>
<asp:Button ID="btnAdd" runat="server" Text="添加到收件人列表"
   SkinID="btnSkin" OnClick="btnAdd_Click" Width="120px" />
<asp:RequiredFieldValidator ID="rfNameBlank" runat="server"
   ControlToValidate="tbTo" Display="none"
   ErrorMessage="收件人地址不能为空！"></asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="rfNameValue" runat="server"
   ControlToValidate="tbTo" Display="none" InitialValue="请输入收件人地址"
   ErrorMessage="收件人地址不能为空！"></asp:RequiredFieldValidator>
<asp:RegularExpressionValidator ID="revName" runat="server"
   ControlToValidate="tbTo" Display="none"
   ErrorMessage="请输入正确的电子邮件地址。"
   ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
   </asp:RegularExpressionValidator>
<ajaxToolkit:TextBoxWatermarkExtender ID="wmeName" runat="server"
   TargetControlID="tbTo" WatermarkText="请输入收件人地址"
   WatermarkCssClass="Watermark">
   </ajaxToolkit:TextBoxWatermarkExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameBlank" runat="server"
   TargetControlID="rfNameBlank" HighlightCssClass="Validator">
   </ajaxToolkit:ValidatorCalloutExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameValue" runat="server"
   TargetControlID="rfNameValue" HighlightCssClass="Validator">
   </ajaxToolkit:ValidatorCalloutExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameRegex" runat="server"
   TargetControlID="revName" HighlightCssClass="Validator">
   </ajaxToolkit:ValidatorCalloutExtender>
```

在下述程序代码中，Addresses.aspx页面添加了1个ListBox控件、1个ImageButton控件和1个Button控件，它们的ID属性的值分别为lbAddress、ibtDelete和btnCommit。lbAddress控件以列表形式显示所有收件人地址。ibtDelete控件可以从lbAddress控件中删除选中的收件人地址。btnCommit控件将lbAddress控件中收件人地址保存到Session中，并重定向到群发邮件的页面。

```
    收件人列表：<asp:ListBox ID="lbAddress" runat="server" Rows="20"
        SkinID="lbSkin" Width="200px"></asp:ListBox>
    <asp:ImageButton ID="ibtDelete" runat="server"
        ImageUrl="~/App_Themes/ASPNETAjaxWeb/Images/delete.gif"
        CausesValidation="False" OnClick="ibtDelete_Click" /></td>
    <asp:Button ID="btnCommit" runat="server" Text="书写邮件正文"
        SkinID="btnSkin" Width="100px" OnClick="btnCommit_Click"
        CausesValidation="False" />
    </ContentTemplate></asp:UpdatePanel>
```

下述程序代码声明了Addresses.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Addresses.aspx.cs"
        Inherits="Addresses" StylesheetTheme="ASPNETAjaxWeb" %>
    <head runat="server"><title>创建群发邮件地址列表</title></head>
    <asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

综合上述，Addresses.aspx页面的设计界面如图3.8所示。

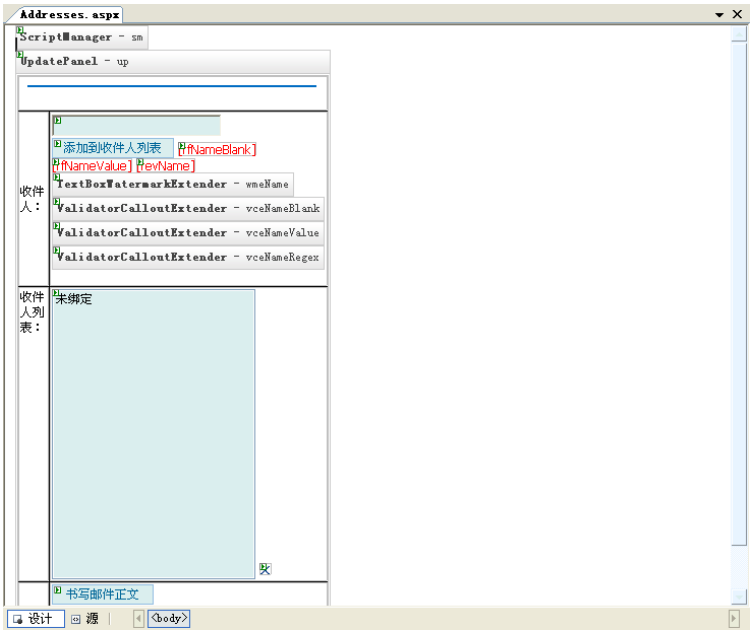

图3.8　Addresses.aspx页面的设计界面

2. 事件设计

Addresses.aspx.cs文件引入了ASPNETAJAXWeb.AjaxMail和System.Text的命名空间。实现上述功能的程序代码如下：

```
    ///引入新的命名空间
    using ASPNETAJAXWeb.AjaxMail;
    using System.Text;
```

用户单击Addresses.aspx页面中的【添加到收件人列表】按钮（btnAdd控件）时，将触发btnAdd控件的Click事件：btnAdd_Click(object sender,EventArgs e)。

在下述程序代码中，btnAdd_Click(object sender,EventArgs e)事件将用户输入的收件人

地址添加lbAddress控件中。其中，该事件执行添加操作之前，检查了被添加的收件人地址在lbAddress控件中是否存在。如果存在，则不添加到lbAddress控件中。检查操作由AjaxMailSystem类的静态方法IsExistItem(ListItem item,ListBox list)实现。

```
protected void btnAdd_Click(object sender,EventArgs e)
{
    if(AjaxMailSystem.IsExistItem(
        new ListItem(tbTo.Text.Trim()),lbAddress) == true)
    {
        AjaxMailSystem.ShowAjaxDialog((Button)sender,
            "输入的电子邮件已经存在，请重新输入。");return;
    }
    lbAddress.Items.Add(new ListItem(tbTo.Text));
}
```

用户单击Addresses.aspx页面中的【×】按钮（ibtDelete控件）时，将触发ibtDelete控件的Click事件：ibtDelete_Click(object sender,EventArgs e)。

在下述程序代码中，ibtDelete_Click(object sender,EventArgs e)事件将lbAddress控件中被选择的收件人地址删除。在执行删除操作之前，事件检查用户是否选择了收件人地址。如果没有选择，则弹出一个对话框提示用户首先选择被删除的收件人地址。如果已经选择，则从lbAddress控件中删除被选择的收件人地址。

```
protected void ibtDelete_Click(object sender,ImageClickEventArgs e)
{
    if(lbAddress.SelectedItem == null)
    {
        AjaxMailSystem.ShowAjaxDialog((Button)sender,
            "请选择被删除的电子邮件");return;
    }
    ///删除选择的电子邮件
    lbAddress.Items.Remove(lbAddress.SelectedItem);
}
```

用户单击Addresses.aspx页面中的【提交】按钮（btnCommit控件）时，将触发btnCommit控件的Click事件：btnCommit_Click(object sender,EventArgs e)。

在下述程序代码中，btnCommit_Click(object sender,EventArgs e)事件首先将lbAddress控件中收件人地址拼接为一个由分号分割的字符串，然后将该字符串保存到Session对象中。其中，标识收件人地址的关键字为AjaxMailSystem.ADDRESSESKEY变量的值。最后，该事件重定向到群发邮件的页面SendMails.aspx。

```
protected void btnCommit_Click(object sender,EventArgs e)
{   ///读取所有电子邮件
    StringBuilder sb = new StringBuilder();
    foreach(ListItem item in lbAddress.Items)
    {   ///添加电子邮件并使用分号分割
        sb.Append(item.Text);sb.Append(";");
    }
    ///保存电子邮件到Session中
    Session[AjaxMailSystem.ADDRESSESKEY] = sb.ToString();
    ///重定向到群发邮件页面
    Response.Redirect("~/SendMails.aspx");
}
```

3.2.6　群发邮件

群发邮件由SendMails.aspx页面实现，SendMails.aspx.cs文件为它的代码隐藏文件。该页面实现邮件群发功能。其中，群发的邮件地址由Addresses.aspx页面配置，并保存在Session对象中。SendMails.aspx页面的效果图如图3.9所示。

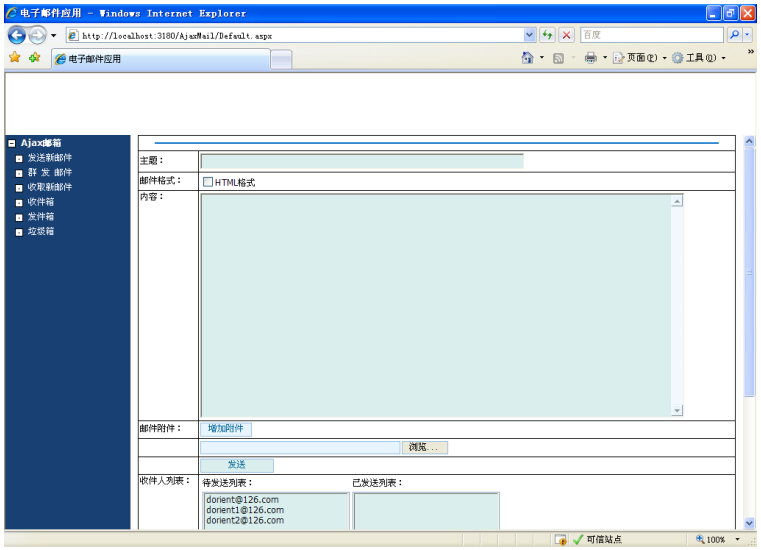

图3.9　SendMails.aspx页面的效果图

1. 界面设计

由于SendMails.aspx和SendMail.aspx页面的设计界面非常相似，因此在此不再详细介绍，SendMails.aspx页面的设计界面如图3.10所示。读者可以参考SendMail.aspx页面的设计界面。

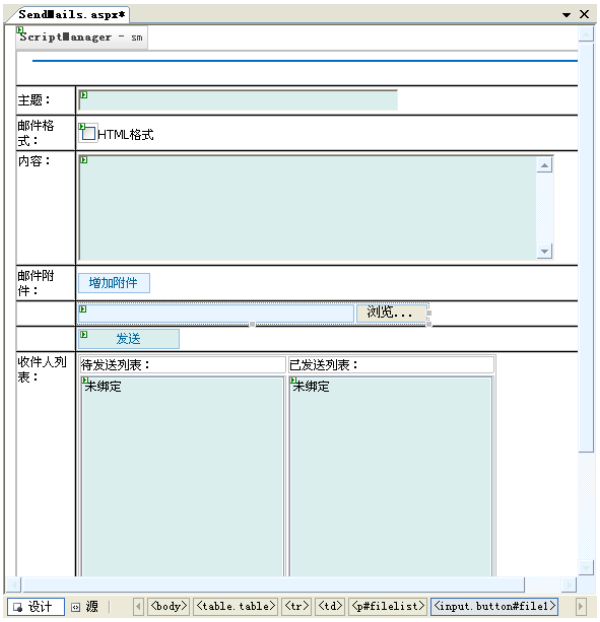

图3.10　SendMails.aspx页面的设计界面

下述程序代码声明了SendMails.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="SendMails.aspx.cs"
    Inherits="SendMails" StylesheetTheme="ASPNETAjaxWeb" %>
<head id="Head1" runat="server"><title>群发邮件</title></head>
<asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

在下述程序代码中，SendMails.aspx页面添加了2个ListBox控件，它们的ID属性的值分别为lbNoSendAddress和lbSendAddress。其中，lbNoSendAddress控件显示待发送的邮件地址；lbSendAddress控件显示已发送的邮件地址。

```
收件人列表：<asp:ListBox ID="lbNoSendAddress" runat="server" Rows="20"
    SkinID="lbSkin" Width="200px"></asp:ListBox>
<asp:ListBox ID="lbSendAddress" runat="server" Rows="20" SkinID="lbSkin"
    Width="200px"></asp:ListBox>
```

2. *初始化*

SendMails.aspx.cs文件引入了4个命名空间：System.IO、System.Text、System.Net.Mail和ASPNETAJAXWeb.AjaxMail。其中，发送邮件的类被包含在System.Net.Mail命名空间中。程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
using System.Net.Mail;
using System.Text;
using System.IO;
```

SendMails.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件首先从Session对象中读取群发的邮件地址（由Addresses.aspx配置），然后将这些地址添加到待发送的邮件地址列表（lbNoSendAddress控件）中。

```
protected void Page_Load(object sender, EventArgs e)
{   ///获取需要发送的邮件地址
    if(Session[AjaxMailSystem.ADDRESSESKEY] != null)
    {
        string addressString
            = Session[AjaxMailSystem.ADDRESSESKEY].ToString();
        string[] addresses = addressString.Split(
            new char[] { ';' },StringSplitOptions.RemoveEmptyEntries);
        ///清空邮件列表
        lbNoSendAddress.Items.Clear();
        foreach(string a in addresses)
        {   ///添加到未发送地址列表中
            lbNoSendAddress.Items.Add(new ListItem(a));
        }
    }
}
```

3. 发送多个邮件

群发邮件是指一次性发送多个电子邮件，它实际性上是重复发送单个邮件的步骤，从而达到一次性发送多个电子邮件的功能。用户单击SendMails.aspx页面中的【发送】按钮（btnCommit控件）时，将触发btnCommit控件的Click事件：btnCommit_Click(object sender, EventArgs e)。该事件实现群发邮件的功能。

btnCommit_Click(object sender,EventArgs e)事件群发邮件的实现步骤和发送单个电子邮件（3.2.4小节）的步骤基本相似。下面仅仅介绍和发送单个电子邮件不相同的实现步骤。

在下述程序代码中，btnCommit_Click(object sender,EventArgs e)事件在发送邮件之前，逐个将待发送的邮件地址列表（lbNoSendAddress控件）中的每一个邮件地址设置为收件人地址，然后发送该邮件，从而实现邮件群发功能。

注意：每一次向 mail 实例中添加收件人地址时，必须事先清空 mail 实例中现有收件人地址，然后再添加本次接收邮件的收件人地址。

```
        ……   ///具体步骤请查看3.2.4小节。
        foreach(ListItem item in lbNoSendAddress.Items)
        {   ///添加收件人地址
            mail.To.Clear();
            mail.To.Add(item.Text.Trim());
            ///发送邮件
            client.Send(mail);
            ……   ///具体步骤请查看3.2.4小节。
            lbSendAddress.Items.Add(item);
        }
        ……   ///具体步骤请查看3.2.4小节。
```

3.3　接收电子邮件

在基于AJAX的电子邮件处理应用程序AjaxMail中，接收电子邮件功能由Pop3Mail类实现。该类定义了多个方法，如连接邮件服务器的Connect()、接收邮件的ReceiveMails(string

user, string password)、断开邮件服务器连接的DisConnect()等方法。本小节将介绍Pop3Mail类的具体实现方法。

3.3.1 Pop3Mail类

Pop3Mail类定义了6个私有字段、2个属性和多个方法（包括私有方法和公开方法），它的结构图如图3.11所示。

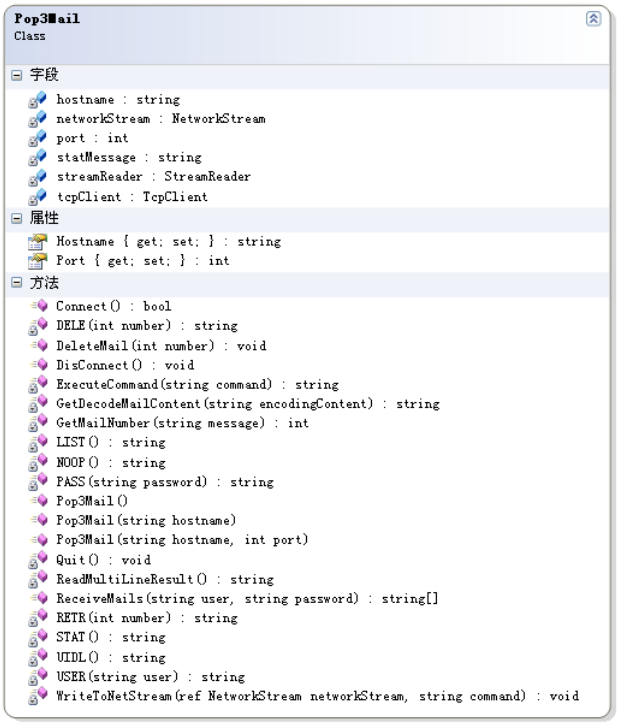

图3.11 Pop3Mail类的结构图

Pop3Mail类中的2个属性为Hostname和Port，它们分别保存邮件服务器的名称和端口。Pop3Mail类包含6个私有字段，如保存邮件服务器名称的hostname、保存邮件服务器端口的port、保存网络流的networkStream等。定义Pop3Mail类中的6个私有字段和2个属性的程序代码如下。

```
public class Pop3Mail
{
    private string hostname = string.Empty;
    private int port = 110;                    ///主机端口，默认值为110
    private TcpClient tcpClient = null;
    private NetworkStream networkStream = null;
    private StreamReader streamReader = null;
    private string statMessage = null;
    #region 属性
    /// <summary>
    /// 主机名称或IP地址
```

```
        /// </summary>
        public string Hostname
        {
            get { return hostname; }
            set { hostname = value; }
        }
        /// <summary>
        /// 主机的端口号
        /// </summary>
        public int Port
        {
            get { return port; }
            set { port = value; }
        }
        #endregion
```

Pop3Mail类提供了2个构造函数：Pop3Mail(string hostname)和Pop3Mail(string hostname, int port)。其中，Pop3Mail(string hostname)函数把邮件服务器的名称设置为hostname参数的值，把邮件服务器端口的值设置为默认值110。Pop3Mail(string hostname,int port)函数把邮件服务器的名称和端口分别设置为hostname和port参数的值。定义Pop3Mail类的构造函数的程序代码如下。

```
        #region  构造函数
        public Pop3Mail(){}
        /// <param name="host">主机名称或IP地址</param>
        public Pop3Mail(string hostname)
        {
            this.hostname = hostname;
        }
        /// <param name="host">主机名称或IP地址</param>
        /// <param name="port">主机的端口号</param>
        public Pop3Mail(string hostname,int port)
        {
            this.hostname = hostname;this.port = port;
        }
        #endregion
        ……
    }
```

3.3.2　Pop3Mail类的私有方法

Pop3Mail类的私有方法还定义了多个私有方法，如发送命令的WriteToNetStream()、获取邮件数量的GetMailNumber()、执行命令的ExecuteCommand()等方法。下面介绍这些方法的具体实现。

1. WriteToNetStream()*方法*

在下述程序代码中，WriteToNetStream(ref NetworkStream networkStream,String command)方法将命令（command参数的值）写入网络流（networkStream参数指定）。该方法首先将命令转换为byte数组，然后将该数组写入网络流networkStream中。

```
#region 私有方法
/// <summary>
/// 向网络访问的基础数据流中写数据（发送命令码）
/// </summary>
/// <param name="networkStream">可以用于网络访问的基础数据流</param>
/// <param name="command">命令行</param>
private void WriteToNetStream(ref NetworkStream networkStream,
    String command)
{
    string sendString = command + "\r\n";
    byte[] sendArray = System.Text.Encoding.ASCII.GetBytes(
        sendString.ToCharArray());
    networkStream.Write(sendArray,0,sendArray.Length);
}
```

2. GetMailNumber()方法

在下述程序代码中，GetMailNumber(string message)方法获取服务器邮件的数量。该方法将包含邮件信息的message参数使用空白字符分割，并返回邮件的数量。

```
/// <summary>
/// 邮箱中的未读邮件数
/// </summary>
/// <param name="message">执行完LIST命令后的结果</param>
/// <returns></returns>
private int GetMailNumber(string message)
{
    return Int32.Parse(message.Split(' ')[1]);
}
```

3. ReadMultiLineResult()方法

在下述程序代码中，ReadMultiLineResult()方法读取多行形式的命令。该方法从命令的第一行开始读取命令，直到命名结尾（结尾字符为“.”）。最后将这些行的命令组成为一个字符串，并返回该字符串。

```
/// <summary>
/// LIST、RETR和UIDL命令的结果要返回多行（以点号结尾）。
/// </summary>
/// <returns></returns>
private string ReadMultiLineResult()
{
    string message = streamReader.ReadLine();
    string stringTemp = string.Empty;
    while(message != ".")
    {
        stringTemp += message;message = streamReader.ReadLine();
    }
    return stringTemp;
}
```

4. ExecuteCommand()方法

在下述程序代码中，ExecuteCommand(string command)方法执行的具体步骤如下：

（1）调用WriteToNetStream(ref NetworkStream networkStream,String command)方法执行给定的命令（command参数的值）。

（2）根据命令的结果是否为多行，将执行不同的读取方式。

（3）如果命令的结果为多行，则调用ReadMultiLineResult()方法读取多行结果，并保存在message变量中。

（4）如果命令的结果为单行，则使用ReadLine()方法直接读取该行的命令，并保存在message变量中。

（5）返回命令的结果message。

```
/// <summary>
/// 执行Pop3命令，并检查执行的结果
/// </summary>
/// <param name="command">Pop3命令行</param>
/// <returns></returns>
private string ExecuteCommand(string command)
{   ///执行Pop3命令后返回的消息
    string message = null;
    try
    {   ///发送命令
        WriteToNetStream(ref networkStream,command);
        ///读取多行
        if(command.Substring(0,4).Equals("LIST") == true
            || command.Substring(0,4).Equals("RETR") == true
            || command.Substring(0,4).Equals("UIDL") == true)
        {
            message = ReadMultiLineResult();
            ///记录LIST后的消息（其中包含邮件数）
            if(command.Equals("LIST") == true) statMessage = message;
        }
        else{message = streamReader.ReadLine();}     ///读取单行
        ///判断执行结果是否正确
        if(message.IndexOf("+OK") > -1) return message;
        else return "Error";
    }
    catch(Exception ex){throw new Exception(ex.Message,ex);}
}
```

5. GetDecodeMailContent()*方法*

在下述程序代码中，GetDecodeMailContent(string encodingContent)方法对邮件内容进行BASE64解码。由于邮件的原始内容是为BASE64解码之后的字符串，因此，要获取邮件的时间内容，则必须对接收到邮件内容进行BASE64解码。

```
/// <summary>
/// 进行base64解码
/// </summary>
/// <param name="encodingContent">解码前的字符串</param>
/// <returns></returns>
private string GetDecodeMailContent(string encodingContent)
```

```
    {
        string content = encodingContent.Trim();
        string encodeString = null;
        int start = content.IndexOf("Base64");
        if(start <= -1) return string.Empty;
        ///获取被解码的字符串
        encodeString = content.Substring(start + 6,content.Length - start - 6);
        try
        {   ///返回base64解码后的字符串
            return Encoding.UTF8.GetString(
                Convert.FromBase64String(encodeString));
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}
    }
    #endregion
```

3.3.3 执行与邮件相关的命令

Pop3Mail类在接收邮件时，需要执行一系列的命令，如USER命令、PASS命令、LIST命令等。下面介绍执行这些命名的方法。

1. 执行USER命令

执行USER命令由USER(string user)方法实现（user参数指定用户名称）。USER(string user)方法调用ExecuteCommand(string command)方法执行USER命令，程序代码如下：

```
    #region Pop3命令
    /// <summary>
    /// USER命令
    /// </summary>
    private string USER(string user)
    {
        return ExecuteCommand("USER " + user) + "\r\n";
    }
```

2. 执行PASS命令

执行PASS命令由PASS(string password)方法实现（user参数指定用户密码）。PASS(string user)方法调用ExecuteCommand(string command)方法执行PASS命令，程序代码如下：

```
    /// <summary>
    /// PASS命令
    /// </summary>
    private string PASS(string password)
    {
        return ExecuteCommand("PASS " + password) + "\r\n";
    }
```

3. 执行LIST命令

执行LIST命令由LIST()方法实现。LIST()方法调用ExecuteCommand(string command)方法执行LIST命令，程序代码如下：

```
    /// <summary>
    /// LIST命令
    /// </summary>
    private string LIST()
    {
        return ExecuteCommand("LIST") + "\r\n";
    }
```

4. 执行UIDL命令

执行UIDL命令由UIDL()方法实现。UIDL()方法调用ExecuteCommand(string command)方法执行UIDL命令，程序代码如下：

```
    /// <summary>
    /// UIDL命令
    /// </summary>
    private string UIDL()
    {
        return ExecuteCommand("UIDL") + "\r\n";
    }
```

5. 执行NOOP命令

执行NOOP命令由NOOP()方法实现。NOOP()方法调用ExecuteCommand(string command)方法执行NOOP命令，程序代码如下：

```
    /// <summary>
    /// NOOP命令
    /// </summary>
    private string NOOP()
    {
        return ExecuteCommand("NOOP") + "\r\n";
    }
```

6. 执行STAT命令

执行STAT命令由STAT()方法实现。STAT()方法调用ExecuteCommand(string command)方法执行STAT命令，程序代码如下：

```
    /// <summary>
    /// STAT命令
    /// </summary>
    private string STAT()
    {
        return ExecuteCommand("STAT") + "\r\n";
    }
```

7. 执行RETR命令

执行RETR命令由RETR(int number)方法实现（number参数指定邮件编号）。RETR(int number)方法调用ExecuteCommand(string command)方法执行RETR命令，程序代码如下：

```
    /// <summary>
    /// RETR命令
    /// </summary>
```

```
private string RETR(int number)
{
    return ExecuteCommand("RETR " + number.ToString()) + "\r\n";
}
```

8. 执行DELE命令

执行DELE命令由DELE(int number)方法实现（number参数指定邮件编号）。DELE(int number)方法调用ExecuteCommand(string command)方法执行DELE命令，程序代码如下：

```
/// <summary>
/// DELE命令
/// </summary>
private string DELE(int number)
{
    return ExecuteCommand("DELE " + number.ToString()) + "\r\n";
}
```

9. 执行Quit命令

执行Quit命令由Quit()方法实现。Quit()方法调用WriteToNetStream(ref NetworkStream networkStream,String command)方法执行Quit命令，程序代码如下：

```
/// <summary>
/// QUIT命令
/// </summary>
private void Quit()
{
    WriteToNetStream(ref networkStream,"QUIT");
}
#endregion
```

3.3.4 连接邮件服务器

若应用程序需要从邮件服务器中接收邮件，则首先需要连接邮件服务器。其中，连接邮件服务器的功能由Connect()方法实现。在下述程序代码中，Connect()方法实现的具体步骤如下：

（1）判断邮件服务器的名称和端口是否正确。如果不正确，则中止方法。

（2）根据邮件服务器的名称和端口创建TcpClient对象tcpClient。

（3）获取tcpClient对象的网络流，并保存在networkStream变量中。

（4）根据tcpClient对象创建StreamReader对象streamReader。

（5）根据从服务器接收到的消息（即消息中是否包含“OK”字符串），判断是否连接成功。如果连接成功，则返回true，否则返回false。

```
/// <summary>
/// 连接给定的主机
/// </summary>
/// <returns></returns>
public bool Connect()
{   ///检测主机和端口是否合法
    if(string.IsNullOrEmpty(hostname) == true && port <= 0) return false;
    try
```

```
        {   ///创建连接
            tcpClient = new TcpClient(hostname,port);
            networkStream = tcpClient.GetStream();
            streamReader = new StreamReader(tcpClient.GetStream());
            ///判断是否连接，如果连接成功，则返回true，否则返回false
            string message = streamReader.ReadLine();
            if(message.IndexOf("+OK") > -1) return true;
            else return false;
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}
    }
```

3.3.5　接收邮件

当应用程序连接邮件服务器之后，就可以开始接收邮件。其中，接收邮件的功能由ReceiveMails(string user,string password)方法实现。该方法实现的具体步骤如下：

（1）验证用户名称是否正确。

（2）验证用户密码是否正确。

（3）检查接收邮件的状态是否正确。

（4）检查邮件列表是否正确。

（5）如果上述4个步骤都是正确的，则调用GetMailNumber(string message)方法获取邮件的数量，并保存在mailNumber变量中。

（6）根据mailNumber变量的值调用GetDecodeMailContent(string encodingContent)方法逐个接收邮件，并把邮件保存到mailContent数组中。

（7）返回mailContent数组，该数组保存接收到的邮件。

```
    /// <summary>
    /// 获取新邮件
    /// </summary>
    /// <param name="user">用户名称</param>
    /// <param name="password">用户密码</param>
    /// <returns></returns>
    public string[] ReceiveMails(string user,string password)
    {
        if(USER(user).Equals("Error")) throw new Exception("用户名称不正确！");
        if(PASS(password).Equals("Error")) throw new Exception(
            "用户密码不正确！");
        if(STAT().Equals("Error")) throw new Exception(
            "准备接收邮件时发生错误！");
        if(LIST().Equals("Error")) throw new Exception(
            "接收邮件列表时发生错误！");
        try
        {   ///获取新邮件数量
            int mailNumber = GetMailNumber(statMessage);
            //没有新邮件
            if(mailNumber == 0) return null;
            ///获取新邮件的内容
            string[] mailContent = new string[mailNumber];
```

```
            for(int i = 0; i < mailNumber; i++)
            {   ///读取邮件内容
                mailContent[i] = GetDecodeMailContent(RETR(i + 1));
            }
            return mailContent;
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}
    }
```

3.3.6 断开邮件服务器连接

当应用程序从邮件服务器接收邮件完成之后，将调用DisConnect()方法断开邮件服务器连接。在下述程序代码中，DisConnect()方法实现断开邮件服务器连接的功能。该方法首先执行“QUIT”命令，然后关闭网络流并释放占有的资源，最后关闭TCP连接。

```
    /// <summary>
    /// 断开所有与服务器的连接
    /// </summary>
    /// <returns></returns>
    public void DisConnect()
    {
        try
        {   ///退出连接
            Quit();
            ///关闭流，并释放所有资源
            if(streamReader != null) streamReader.Close();
            if(networkStream != null) networkStream.Close();
            if(tcpClient != null) tcpClient.Close();
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}
    }
```

3.3.7 删除邮件

应用程序调用Pop3Mail类的DeleteMail(int number)方法可以从服务器中删除指定的邮件。其中，number参数指定被删除邮件的编号。删除邮件功能需要执行“DELE”命令。DeleteMail(int number)方法的程序代码如下。

```
    /// <summary>
    /// 删除邮件
    /// </summary>
    /// <param name="number">邮件编号</param>
    public void DeleteMail(int number)
    {   ///删除邮件
        int mailNumber = number + 1;
        if(DELE(mailNumber).Equals("Error"))
          throw new Exception("删除第" + mailNumber.ToString() + "时出现错误！");
    }
```

3.4　邮箱管理

本小节介绍与邮箱管理相关的功能，如邮箱主页面、邮箱菜单操作树、邮箱列表页面、邮箱管理页面、阅读邮件、添加新邮箱文件夹、修改邮箱文件夹、删除邮件文件夹等。

3.4.1　数据库访问层设计

与邮箱相关的数据访问层共实现5个功能。实现这些功能的方法（定义在Mail类中）具体描述如下：

- public DataSet GetMailboxs()，获取所有邮箱的信息。
- public SqlDataReader GetSingleMailbox(int mailboxID)，获取指定邮箱的信息。
- public int AddMailbox(string name)，添加邮箱到数据库中。
- public int UpdateMailbox(int mailboxID,string name)，修改邮箱的名称。
- public int DeleteMailbox(int mailboxID)，删除指定的邮箱。

其中，mailboxID参数指定邮箱的ID值，name参数指定邮箱的名称。

在下述程序代码中，GetMailboxs()方法获取所有邮箱的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取所有邮箱的信息的SQL语句，并在该SQL语句中获取了每一个邮箱的新邮件数量（NewMailCount）、邮件总数量（TotalMailCount）和邮箱总使用空间（TotalSize）。

（4）创建获取数据的SqlDataAdapter对象da。

（5）打开数据库连接，并获取数据。获取的结果保存在DataSet对象ds中。

（6）如果上述操作成功，则返回ds。

```
public DataSet GetMailboxs()
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "SELECT M.*,"
        + "(SELECT COUNT(*) FROM Mail WHERE Mail.MailboxID=M.ID
           AND Status=0) AS NewMailCount,"
        + "(SELECT COUNT(*) FROM Mail WHERE Mail.MailboxID=M.ID)
           AS TotalMailCount,"
        + "(SELECT SUM(Size) FROM Mail WHERE Mail.MailboxID=M.ID)
           AS TotalSize"
        + " FROM Mailbox AS M";
    ///创建SqlDataAdapter
    SqlDataAdapter da = new SqlDataAdapter(cmdText,con);
    ///定义DataSet
    DataSet ds = new DataSet();
    try
    {   ///打开连接
```

```
            con.Open();
            ///填充数据
            da.Fill(ds,"DataTable");
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return ds;
    }
```

在下述程序代码中，GetSingleMailbox(int mailboxID)方法获取指定邮箱的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建获取指定邮箱的SQL语句“SELECT * FROM Mailbox WHERE ID=@ID”。其中，@ID参数的值由mailboxID参数指定。

（4）创建获取数据的SqlCommand对cmd。

（5）打开数据库连接，并获取数据。获取的结果保存在SqlDataReader对象dr中。

（6）如果上述操作成功，则返回dr。

```
    public SqlDataReader GetSingleMailbox(int mailboxID)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
            "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "SELECT * FROM Mailbox WHERE ID=@ID";
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = mailboxID;
        ///定义SqlDataReader
        SqlDataReader dr;
        try
        {   ///打开连接
            con.Open();
            ///读取数据
            dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        return dr;
    }
```

在下述程序代码中，AddMailbox(string name)方法将邮箱添加到数据库中。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建插入邮箱的SQL语句“INSERT INTO Mailbox(Name)VALUES(@Name)”。其中，@Name参数的值由name参数指定。

（4）创建执行插入操作的SqlCommand对cmd。

（5）打开数据库连接，并执行插入操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
public int AddMailbox(string name)
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "INSERT INTO Mailbox(Name)VALUES(@Name)";
    ///创建SqlCommand
    SqlCommand cmd = new SqlCommand(cmdText,con);
    ///创建参数并赋值
    cmd.Parameters.Add("@Name",SqlDbType.VarChar,50);
    cmd.Parameters[0].Value = name;
    int result = -1;
    try
    {   ///打开连接
        con.Open();
        ///操作数据
        result = cmd.ExecuteNonQuery();
    }
    catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
    finally{con.Close();}                                      ///关闭连接
    return result;
}
```

在下述程序代码中，UpdateMailbox(int mailboxID,string name)方法修改指定邮箱的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建修改邮箱的SQL语句“UPDATE Mailbox SET Name=@Name WHERE ID=@ID”。其中，@Name和@ID参数的值分别由name和mailboxID参数指定。

（4）创建执行修改操作的SqlCommand对cmd。

（5）打开数据库连接，并执行修改操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
public int UpdateMailbox(int mailboxID,string name)
{   ///获取连接字符串
    string connectionString = ConfigurationManager.ConnectionStrings[
        "SQLCONNECTIONSTRING"].ConnectionString;
    ///创建连接
    SqlConnection con = new SqlConnection(connectionString);
    ///创建SQL语句
    string cmdText = "UPDATE Mailbox SET Name=@Name WHERE ID=@ID";
```

```
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@Name",SqlDbType.VarChar,50);
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = name;
        cmd.Parameters[1].Value = mailboxID;
        int result = -1;
        try
        {   ///打开连接
            con.Open();
            ///操作数据
            result = cmd.ExecuteNonQuery();
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return result;
    }
```

在下述程序代码中，DeleteMailbox(int mailboxID)方法将删除指定邮箱的信息。该方法的具体实现步骤如下：

（1）从Web.Config配置文件中获取数据库连接字符串，并保存在变量connectionString中。

（2）使用上述连接字符串创建SqlConnection对象con，该对象用来连接数据库。

（3）创建删除指定邮箱的SQL语句“DELETE Mailbox WHERE ID = @ID”。其中，@ID参数的值由mailboxID参数指定。

（4）创建执行删除操作的SqlCommand对cmd。

（5）打开数据库连接，并执行删除操作，并将该操作影响的行数保存在result变量中。

（6）如果上述操作成功，则返回result变量的值，否则返回-1。

```
    public int DeleteMailbox(int mailboxID)
    {   ///获取连接字符串
        string connectionString = ConfigurationManager.ConnectionStrings[
            "SQLCONNECTIONSTRING"].ConnectionString;
        ///创建连接
        SqlConnection con = new SqlConnection(connectionString);
        ///创建SQL语句
        string cmdText = "DELETE Mailbox WHERE ID = @ID";
        ///创建SqlCommand
        SqlCommand cmd = new SqlCommand(cmdText,con);
        ///创建参数并赋值
        cmd.Parameters.Add("@ID",SqlDbType.Int,4);
        cmd.Parameters[0].Value = mailboxID;
        int result = -1;
        try
        {   ///打开连接
            con.Open();
            ///操作数据
            result = cmd.ExecuteNonQuery();
```

```
        }
        catch(Exception ex){throw new Exception(ex.Message,ex);}   ///抛出异常
        finally{con.Close();}                                      ///关闭连接
        return result;
    }
```

3.4.2　邮箱主页面

邮箱主页面由Default.aspx页面实现，Default.aspx.cs文件为它的代码隐藏文件。该页面由多个框架（frame）构成，并分成3部分：页面头部分、页面左部分和页面主体部分。其中，页面头部分放置Header.aspx页面，页面左部分放置MailTree.aspx页面（显示邮箱菜单操作树），页面主体部分放置MailboxList.aspx页面（显示邮箱和邮箱列表管理界面）。邮箱主页面的组成如图3.12所示，效果图如图3.13所示。

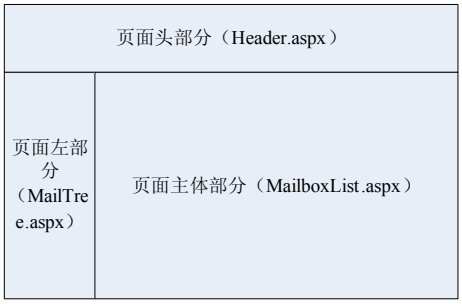

图3.12　邮箱主页面Default.aspx页面的组成

图3.13　邮箱主页面Default.aspx页面的效果图

在Default.aspx页面中，页面头部分的框架的名称为“Header”，页面左部分的框架的名称为“MenuTree”，页面主体部分的框架的名称为“Mailbox”。Default.aspx页面的HTML设计代码如下：

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
```

```
    Inherits="_Default" %>
<head id="Head1" runat="server"><title>电子邮件应用</title></head>
<frameset id="Default" rows="84,*" frameborder="0" border="0"
    framespacing="0">
    <frame id="Header" name="Header" src="Header.aspx" scrolling="auto"
        noresize></frame>
    <frameset id="Main" cols="170,*" rows="*" border="0" framespacing="0">
        <frame name="MenuTree" src="MailTree.aspx" scrolling="auto"
            frameborder="0" noresize></frame>
        <frame name="Mailbox" src="MailboxList.aspx" scrolling="auto"
            frameborder="0"></frame>
    </frameset>
</frameset>
```

3.4.3 邮箱菜单操作树

邮箱菜单操作树由MailTree.aspx页面实现，MailTree.aspx.cs文件为它的代码隐藏文件。该页面以树型结构显示邮箱的操作菜单，如“发送新邮件”、“群发邮件”、“收件箱”、“发件箱”等。

1. 界面设计

在下述程序代码中，MailTree.aspx页面使用TreeView控件（ID属性的值为tvMailTree）来显示邮箱的操作菜单。tvMailTree控件的根节点为“AJAX邮箱”，并添加了3个子节点：“发送新邮件”、“群发邮件”和“收取邮件”。这4个节点分别导航到MailboxList.aspx、SendMail.aspx、Addresses.aspx和ReceiveMail.aspx页面。

```
<asp:TreeView runat="server" ID="tvMailTree" SkinID="tvSkin">
    <Nodes>
        <asp:TreeNode Text="Ajax邮箱" Value="-1" Expanded="true"
            Target="Mailbox" NavigateUrl="~/MailboxList.aspx">
        <asp:TreeNode Text="发送新邮件" Value="9999" Expanded="true"
            Target="Mailbox" NavigateUrl="~/SendMail.aspx"></asp:TreeNode>
        <asp:TreeNode Text="群 发 邮件" Value="9998" Expanded="true"
            Target="Mailbox" NavigateUrl="~/Addresses.aspx">
            </asp:TreeNode>
        <asp:TreeNode Text="收取新邮件" Value="9997" Expanded="true"
            Target="Mailbox" NavigateUrl="~/ReceiveMail.aspx">
            </asp:TreeNode>
        </asp:TreeNode>
    </Nodes>
</asp:TreeView>
```

下述程序代码声明了MailTree.aspx页面所必须的“@Page”指令。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="MailTree.aspx.cs"
    Inherits="MailTree" StylesheetTheme="ASPNETAjaxWeb" %>
```

综合上述，MailTree.aspx页面的设计界面如图3.14所示。

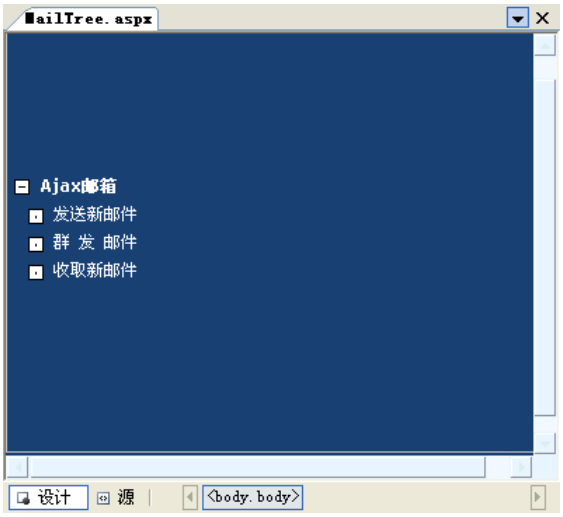

图3.14　MailTree.aspx页面的设计界面

2. 初始化

MailTree.aspx.cs文件引入了System.Data.SqlClient和ASPNETAJAXWeb.AjaxMail名空间，程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
using System.Data.SqlClient;
```

MailTree.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件调用BindPageData()函数在tvMailTree控件的根节点（“AJAX邮箱”）下添加邮箱的导航菜单节点。Mail类的GetMailboxs()方法（返回DataSet类型的对象）首先从数据库中读取邮箱的数据，然后将每一个数据项创建为一个TreeNode节点，并添加到tvMailTree控件中。另外，新创建的每一个TreeNode节点的导航页面都是Mailbox.aspx页面。

注意：MailTree.aspx 页面的 HTML 设计中的 tvMailTree 控件的节点是静态节点。BindPageData()函数创建的节点是动态节点，它紧跟在 tvMailTree 控件的静态节点之后。

```
protected void Page_Load(object sender,EventArgs e)
{
    if(!Page.IsPostBack){BindPageData();}
}
private void BindPageData()
{   ///获取数据
    Mail mail = new Mail();
    DataSet ds = mail.GetMailboxs();
    if(ds == null || ds.Tables.Count <= 0
        || ds.Tables[0].Rows.Count <= 0) return;
    ///显示邮箱
    foreach(DataRow row in ds.Tables[0].Rows)
    {   ///创建菜单节点
```

```
            TreeNode node = new TreeNode();
            node.Value = row["ID"].ToString();
            node.Text = row["Name"].ToString();
            node.Target = "Mailbox";
            node.NavigateUrl = "~/Mailbox.aspx?MailboxID=" + node.Value;
            ///添加到操作树中
            tvMailTree.Nodes[0].ChildNodes.Add(node);
        }
    }
```

3.4.4 邮箱列表页面

邮箱列表页面由MailboxList.aspx页面实现，MailboxList.aspx.cs文件为它的代码隐藏文件。该页面以列表形式显示所有邮箱的信息，如邮箱名称、新邮件数量、邮件总数、使用空间等，以及修改和删除邮箱的操作按钮。MailboxList.aspx页面的效果图如图3.15所示。

图3.15 MailboxList.aspx页面的效果图

1. 界面设计

MailboxList.aspx页面添加了1个GridView控件，ID属性的值为gvMailbox。该控件以列表形式显示邮箱的信息，它包含5个模板域：“邮箱名称”、“新邮件”、“邮件总数”、“使用空间”和“操作”。单击“邮箱名称”模板域中的链接可以查看邮箱包含的邮件。“操作”模板域提供了对邮箱的修改和删除操作。

```
    <asp:UpdatePanel runat="server" ID="up"><ContentTemplate>
    <asp:GridView ID="gvMailbox" runat="server" Width="100%"
        AutoGenerateColumns="False" SkinID="gvSkin" EmptyDataText="邮件为空。"
        OnRowCommand="gvMailbox_RowCommand"
        OnRowDataBound="gvMailbox_RowDataBound">
        <Columns>
            <asp:TemplateField HeaderText="邮箱名称"><ItemTemplate>
                <a href='Mailbox.aspx?MailboxID=<%# Eval("ID") %>'
```

```
                target="Mailbox"><%# Eval("Name")%></a>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="新邮件"><ItemTemplate>
                <%# Eval("NewMailCount") %>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="邮件总数"><ItemTemplate>
                <%# Eval("TotalMailCount") %>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="使用空间"><ItemTemplate>
                <%# ComputerSize(Eval("TotalSize") == DBNull.Value
                    ? string.Empty : Eval("TotalSize").ToString())%>
            </ItemTemplate></asp:TemplateField>
```

下述HTML代码为gvMailbox控件的“操作”的代码，它包含2个ImageButton控件。一个控件ID属性的值为“imgUpdate”，单击此控件可以重定向到修改邮箱UpdateMailbox.aspx页面。另外一个控件ID属性的值为“imgDelete”，单击此控件可以当前行的邮箱。

```
            <asp:TemplateField HeaderText="操作"><ItemTemplate>
               <asp:ImageButton ID="imgUpdate" runat="server" Visible='<%#
                   (int)Eval("ID") < 4 ? false : true %>'
                   CommandArgument='<%# Eval("ID") %>'
                   ImageUrl="~/App_Themes/ASPNETAjaxWeb/Images/edit.PNG"
                   CommandName="update" /> 
               <asp:ImageButton ID="imgDelete" runat="server"
                   CommandArgument='<%# Eval("ID") %>'
                   ImageUrl="~/App_Themes/ASPNETAjaxWeb/Images/delete.PNG"
                   CommandName="del" />
            </ItemTemplate></asp:TemplateField>
        </Columns>
    </asp:GridView>
    </ContentTemplate></asp:UpdatePanel>
```

下述程序代码声明了MailboxList.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
    <%@ Page Language="C#" AutoEventWireup="true"
        CodeFile="MailboxList.aspx.cs" Inherits="MailboxList"
        StylesheetTheme="ASPNETAjaxWeb" %>
    <head id="Head1" runat="server"><title>邮箱列表</title></head>
    <asp:ScriptManager ID="sm" runat="server" ></asp:ScriptManager>
```

综合上述，MailboxList.aspx页面的设计界面如图3.16所示。

图3.16　MailboxList.aspx页面的设计界面

2. *初始化*

MailboxList.aspx.cs文件引入了ASPNETAJAXWeb.AjaxMail名空间，程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
```

MailboxList.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件调用BindPageData()函数绑定gvMailbox控件的数据，并显示邮箱的信息。其中，获取邮箱数据的功能由Mail类的GetMailboxs()方法（返回DataSet类型的对象）实现。

```
protected void Page_Load(object sender,EventArgs e)
{
    if(!Page.IsPostBack){BindPageData();}
}
private void BindPageData()
{   ///获取数据
    Mail mail = new Mail();
    DataSet ds = mail.GetMailboxs();
    ///显示数据
    gvMailbox.DataSource = ds;
    gvMailbox.DataBind();
}
```

gvMailbox控件在数据绑定时还调用了ComputerSize(string totalSize)函数计算邮箱的使用空间。在下述程序代码中，ComputerSize(string totalSize)函数根据totalSize参数的值计算邮箱大小。如果totalSize参数的值小于1024，则直接返回该参数的值。如果totalSize参数的值小于1024×1024，则返回该参数的值除以1024的结果，然后添加单位“K”。如果totalSize

参数的值小于1024×1024×1024，则返回该参数的值除以（1024×1024）的结果，然后添加单位“M”。

```
protected string ComputerSize(string totalSize)
{   ///处理空值
    if(string.IsNullOrEmpty(totalSize) == true) return "0";
    ///转换为整数
    int size = Int32.Parse(totalSize);
    if(size == 0) return size.ToString();
    ///大小转换
    if(size < 1024) return size.ToString();
    if(size < 1024 * 1024) return (size / 1024).ToString() + "K";
    if(size < 1024 * 1024 * 1024)
        return (size / 1024 / 1024).ToString() + "M";
    return size.ToString();
}
```

3. 事件设计

gvMailbox控件定义了RowDataBound事件——gvMailbox_RowDataBound(object sender, GridViewRowEventArgs e)。在下述程序代码中，该事件在gvMailbox控件的每一行数据绑定之后触发，它为每一行中的删除按钮（imgDelete控件）添加了一个删除确认对话框。

```
protected void gvMailbox_RowDataBound(object sender,
    GridViewRowEventArgs e)
{   ///添加删除确认的对话框
    ImageButton imgDelete = (ImageButton)e.Row.FindControl("imgDelete");
    if(imgDelete != null)
    {
        imgDelete.Attributes.Add("onclick",
            "return confirm(\"您确认要删除当前行的邮件文件夹吗？\");");
    }
}
```

gvMailbox控件定义了RowCommand事件——gvMailbox_RowCommand(object sender, GridViewRowEventArgs e)。在下述程序代码中，该事件在用户单击gvMailbox控件的每一行中的按钮之后触发。如果被单击按钮的CommandName属性的值为“update”，则重定向到修改邮箱的页面UpdateMailbox.aspx，并将被修改邮箱的ID值传递到该页面。如果被单击按钮的CommandName属性的值为“del”，则调用Mail类的DeleteMailbox(int mailboxID)方法删除当前行的邮箱，并重新显示邮箱信息。

```
protected void gvMailbox_RowCommand(object sender,
    GridViewCommandEventArgs e)
{
    if(e.CommandName.ToLower() == "update")
    {   ///重定向到修改邮箱文件夹页面
        Response.Redirect("~/UpdateMailbox.aspx?MailboxID="
            + e.CommandArgument.ToString());
    }
    if(e.CommandName.ToLower() == "del")
    {   ///删除选择的邮件文件夹
        Mail mail = new Mail();
```

```
            if(mail.DeleteMailbox(
                Int32.Parse(e.CommandArgument.ToString())) > 0)
            {BindPageData();}
            return;
        }
    }
```

3.4.5 邮件管理页面

邮件管理页面由Mailbox.aspx页面实现，Mailbox.aspx.cs文件为它的代码隐藏文件。该页面以列表形式显示邮箱所包含邮件的具体内容，如邮件的发件人、主题、发送时间、大小等，以及提供了删除邮件的操作的功能。Mailbox.aspx页面的效果图如图3.17所示。

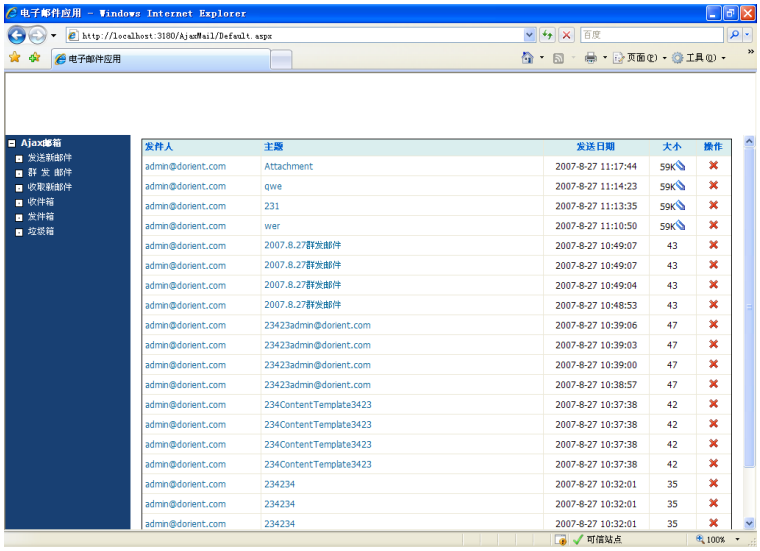

图3.17 Mailbox.aspx页面的效果图

1. 界面设计

Mailbox.aspx页面添加了1个TextBox控件、1个DropDownList控件、1个FileUpload控件、1个Label控件和1个Button控件，它们的ID属性的值分别为tbName、ddlCategory、fuPhoto、lbMessage和btnCommit。

Mailbox.aspx页面添加了1个GridView控件，ID属性的值为gvMailbox。该控件以列表形式显示邮箱所包含邮件的信息，它包含5个模板域：“发件人”、“主题”、“发送日期”、“大小”和“操作”。单击“主题”模板域中的链接可以查看该邮件的具体内容。“操作”模板域提供了删除邮件的操作。

```
    <asp:UpdatePanel runat="server" ID="up"><ContentTemplate>
    <asp:GridView ID="gvMailbox" runat="server" Width="100%"
        AutoGenerateColumns="False" SkinID="gvSkin" EmptyDataText="邮件为空。"
        OnRowCommand="gvMailbox_RowCommand"
        OnRowDataBound="gvMailbox_RowDataBound" AllowPaging="True"
        OnPageIndexChanging="gvMailbox_PageIndexChanging" PageSize="20">
        <Columns>
```

```
            <asp:TemplateField HeaderText="发件人"><ItemTemplate>
               <a href='SendMail.aspx?Address=<%# Eval("FromAddress") %>'>
                   <%# Eval("FromAddress")%></a>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="主题"><ItemTemplate>
               <a href='ReadMail.aspx?MailID=<%# Eval("ID") %>'>
                  <%# Eval("Title") %></a>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="发送日期"><ItemTemplate>
               <%# Eval("CreateDate") %>
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="大小"><ItemTemplate>
               <%# (int)Eval("Size") > 1024 ? ((int)Eval("Size") / 1024 + "K")
                   : Eval("Size") %>
               <asp:Image ID="imgAttachment" runat="server"
                  ImageUrl="~/App_Themes/ASPNETAjaxWeb/Images/attch.gif"
                  Visible='<%# (int)Eval("AttachmentCount") > 0
                     ? true : false %>' />
            </ItemTemplate></asp:TemplateField>
            <asp:TemplateField HeaderText="操作"><ItemTemplate>
                <asp:ImageButton ID="imgDelete" runat="server"
                   CommandArgument='<%# Eval("ID") %>'
                   ImageUrl="~/App_Themes/ASPNETAjaxWeb/Images/delete.PNG"
                   CommandName="del" />
            </ItemTemplate></asp:TemplateField>
         </Columns>
         <PagerSettings Mode="NextPreviousFirstLast" />
      </asp:GridView>
      </ContentTemplate></asp:UpdatePanel>
```

下述程序代码声明了Mailbox.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Mailbox.aspx.cs"
   Inherits="Mailbox" StylesheetTheme="ASPNETAjaxWeb" %>
<head id="Head1" runat="server"><title>邮箱</title></head>
<asp:ScriptManager ID="sm" runat="server" ></asp:ScriptManager>
```

综合上述，Mailbox.aspx页面的设计界面如图3.18所示。

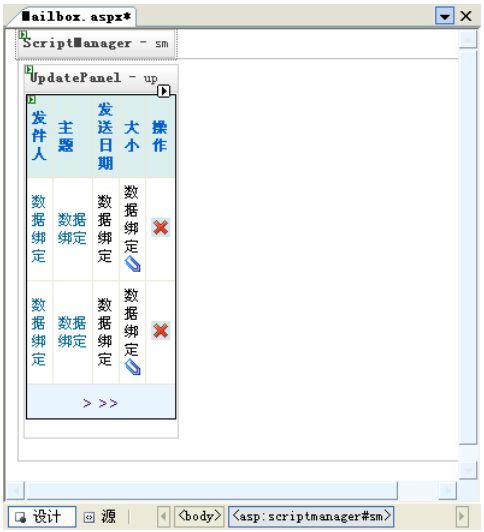

图3.18　Mailbox.aspx页面的设计界面

2. *初始化*

Mailbox.aspx.cs文件引入了ASPNETAJAXWeb.AjaxMail名空间，并定义了指定邮箱的ID值的mailboxID变量。程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
int mailboxID = -1;
```

Mailbox.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件首先从Mailbox.aspx页面的地址栏中获取邮箱的ID值，并保存在mailboxID变量中。然后调用BindPageData(int mailboxID)函数显示指定邮箱（由mailboxID参数指定）所包含邮件的具体内容。其中，获取数据的功能由Mail类的GetMailByMailBox(int mailboxID)方法（返回DataSet类型的对象）实现。

```
protected void Page_Load(object sender, EventArgs e)
{
    if(Request.Params["MailboxID"] != null)
    {
        mailboxID = Int32.Parse(Request.Params["MailboxID"].ToString());
    }
    if(!Page.IsPostBack && mailboxID > 0){BindPageData(mailboxID);}
}
private void BindPageData(int mailboxID)
{   ///获取数据
    Mail mail = new Mail();
    DataSet ds = mail.GetMailByMailBox(mailboxID);
    ///显示数据
    gvMailbox.DataSource = ds;
    gvMailbox.DataBind();
}
```

3. 事件设计

gvMailbox控件定义了RowDataBound事件——gvMailbox_RowDataBound(object sender, GridViewRowEventArgs e)。在下述程序代码中，该事件在gvMailbox控件的每一行数据绑定之后触发，它为每一行中的删除按钮（imgDelete控件）添加了一个删除确认对话框。

```
protected void gvMailbox_RowDataBound(object sender,
    GridViewRowEventArgs e)
{   ///添加删除确认的对话框
    ImageButton imgDelete = (ImageButton)e.Row.FindControl("imgDelete");
    if(imgDelete != null)
    {
        imgDelete.Attributes.Add("onclick",
            "return confirm(\"您确认要删除当前行的邮件吗？\");");
    }
}
```

gvMailbox控件定义了RowCommand事件——gvMailbox_RowCommand(object sender, GridViewRowEventArgs e)。在下述程序代码中，该事件在用户单击gvMailbox控件的每一行中的按钮之后触发。如果被单击按钮的CommandName属性的值为“del”，则调用Mail类的DeleteMailbox(int mailboxID)方法删除当前行的邮件，并重新显示当前邮箱中的所有邮件信息。

```
protected void gvMailbox_RowCommand(object sender,
    GridViewCommandEventArgs e)
{
    if(e.CommandName.ToLower() == "del")
    {   ///删除选择的邮件
        Mail mail = new Mail();
        if(mail.DeleteMail(Int32.Parse(e.CommandArgument.ToString())) > 0)
        {
            BindPageData(mailboxID);
        }
        return;
    }
}
```

4. 分页设计

gvMailbox控件启用了分页功能，并设置分页模式为NextPreviousFirstLast，同时还定义了分页事件：gvMailbox_PageIndexChanging(object sender,GridViewPageEventArgs e)。在下述程序代码中，该事件首先设置gvMailbox控件的新页码，然后重新绑定gvMailbox控件的数据，并显示邮件信息。

```
protected void gvMailbox_PageIndexChanging(object sender,
    GridViewPageEventArgs e)
{   ///设置新的页码，并重新显示数据
    gvMailbox.PageIndex = e.NewPageIndex;
    BindPageData(mailboxID);
}
```

3.4.6　阅读邮件

阅读邮件由ReadMail.aspx页面实现，ReadMail.aspx.cs文件为它的代码隐藏文件。该页

面显示邮件的详细内容，如邮件的发件人、主题、内容、邮件附件等。单击【回复该邮件】按钮可以迅速导航到发送邮件的SendMail.aspx页面。ReadMail.aspx页面的效果图如图3.19所示。

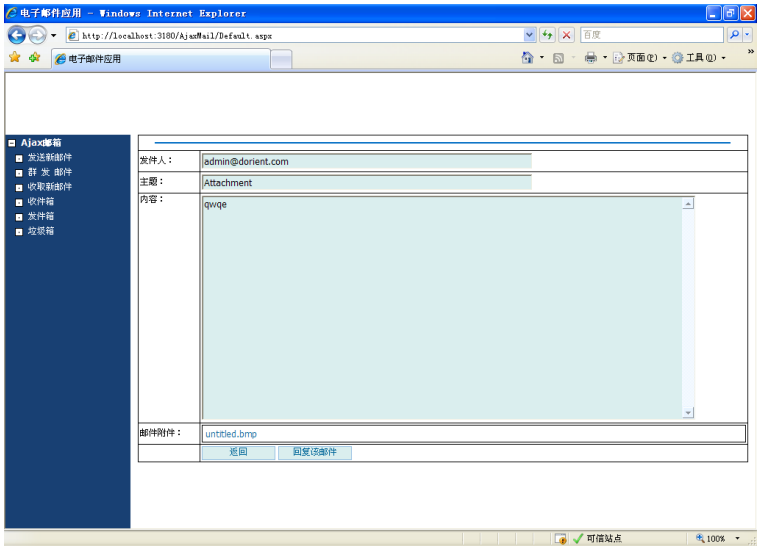

图3.19　ReadMail.aspx页面的效果图

1. 界面设计

在下述程序代码中，ReadMail.aspx页面添加了3个TextBox控件和1个GridView控件，它们的ID属性的值分别为tbTo、tbTitle、tbBody和gvAttachment。其中，tbTo控件显示邮件的发件人；tbTitle控件显示邮件的主题；tbBody显示邮件的内容；gvAttachment控件以列表形式显示邮件的附件，并且单击附件的名称可以查看或下载该附件。

```
发件人：<asp:TextBox ID="tbTo" runat="server" SkinID="tbSkin" Width="60%"
    MaxLength="255"></asp:TextBox>
主题：<asp:TextBox ID="tbTitle" runat="server" SkinID="tbSkin" Width="60%"
    MaxLength="50"></asp:TextBox>
内容：<asp:TextBox ID="tbBody" runat="server" SkinID="tbSkin" Width="90%"
    Height="300px" TextMode="MultiLine"></asp:TextBox>
邮件附件：<asp:GridView ID="gvAttachment" runat="server"
    AutoGenerateColumns="False" SkinID="gvSkin" EmptyDataText="附件为空。"
    ShowHeader="False" >
    <Columns>
        <asp:TemplateField><ItemTemplate>
            <a href='<%# Eval("Url") %>' target="_blank">
              <%# Eval("Name") %></a>
        </ItemTemplate></asp:TemplateField>
    </Columns>
</asp:GridView>
```

在下述程序代码中，ReadMail.aspx页面添加了2个Button控件，它们的ID属性的值分别为btnCommit和btnReply。单击【返回】按钮（btnCommit控件）重定向到邮箱管理页面Mailbox.aspx，并将当前邮件所属邮箱的ID值传递到该页面。单击【回复该邮件】按钮

（btnReply控件）重定向到回复该邮件的SendMail.aspx页面，并将当前邮件的发件人地址传递到该页面。

```
<asp:Button ID="btnCommit" runat="server" Text="返回" SkinID="btnSkin"
    Width="100px" OnClick="btnCommit_Click" /> 
<asp:Button ID="btnReply" runat="server" Text="回复该邮件" SkinID="btnSkin"
    Width="100px" OnClick="btnReply_Click" />
```

下述程序代码声明了ReadMail.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ReadMail.aspx.cs"
    Inherits="ReadMail" StylesheetTheme="ASPNETAjaxWeb" %>
<head id="Head1" runat="server"><title>阅读邮件</title></head>
<asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

综合上述，ReadMail.aspx页面的设计界面如图3.20所示。

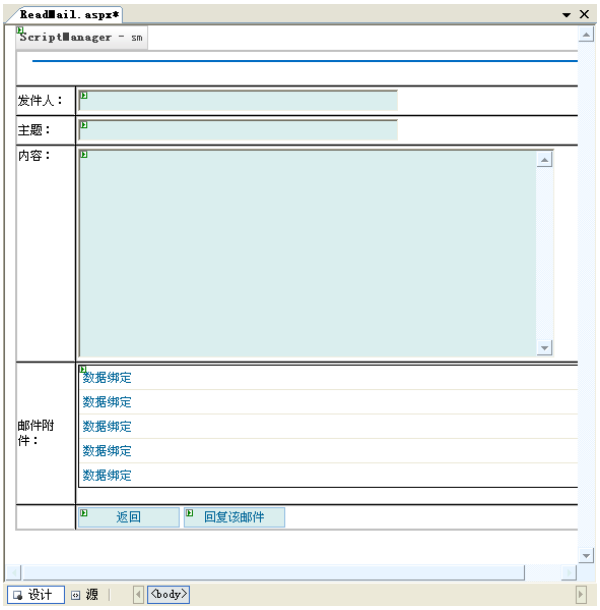

图3.20　ReadMail.aspx页面的设计界面

2. *初始化*

ReadMail.aspx.cs文件引入了System.Data.SqlClient和ASPNETAJAXWeb.AjaxMail名空间，并定义了保存被阅读的邮件ID值的mailID变量。程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
using System.Data.SqlClient;
int mailID = -1;
```

ReadMail.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件首先从ReadMail.aspx页面的地址栏中获取邮件的ID值，并保存在mailID变量中。然后调用BindPageData(int mailID)函数显示被阅读的邮件（由mailID参数指定）的具体内容。其中，获取数据的功能由Mail类的GetSingleMail(int mailID)方法（返回SqlDataReader类型的对象）实现。

```
protected void Page_Load(object sender, EventArgs e)
{   ///获取邮件ID
    if(Request.Params["MailID"] != null)
    {
        mailID = Int32.Parse(Request.Params["MailID"].ToString());
    }
    ///显示数据
    if(!Page.IsPostBack && mailID > 0){BindPageData(mailID);}
}
private void BindPageData(int mailID)
{   ///读取数据
    Mail mail = new Mail();
    SqlDataReader dr = mail.GetSingleMail(mailID);
    if(dr == null) return;
    if(dr.Read())
    {   ///显示数据
        tbTitle.Text = dr["Title"].ToString();
        tbBody.Text = dr["Body"].ToString();
        tbTo.Text = dr["FromAddress"].ToString();
        ///获取邮件的附件
        DataSet ds = mail.GetAttachmentByMail(mailID);
        gvAttachment.DataSource = ds;
        gvAttachment.DataBind();
    }
    dr.Close();
}
```

3.4.7 添加新邮箱文件夹

添加新邮箱文件夹由AddMailbox.aspx页面实现，AddMailbox.aspx.cs文件为它的代码隐藏文件。该页面提供了输入邮箱文件夹的名称，以及将邮箱文件夹保存到数据库中的功能。AddMailbox.aspx页面的效果图如图3.21所示。

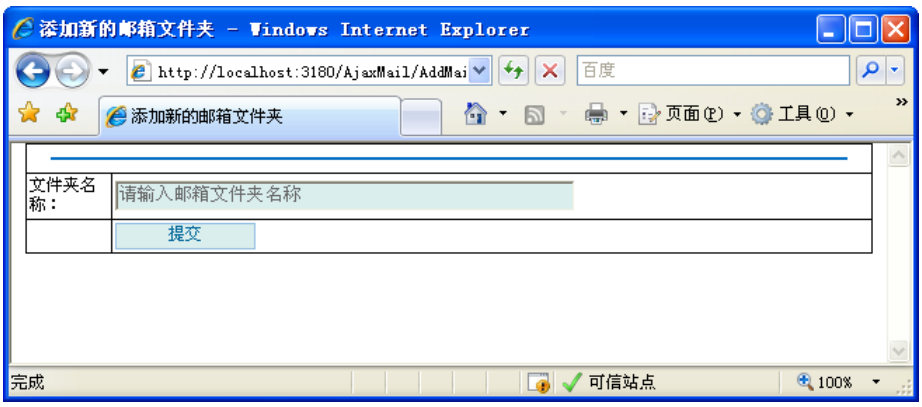

图3.21 AddMailbox.aspx页面的效果图

1. 界面设计

AddMailbox.aspx页面添加了1个TextBox控件和1个Button控件，它们的ID属性的值分别为tbName和btnCommit。

在下述程序代码中，AddMailbox.aspx页面声明了1个TextBox控件，用来输入邮箱文件夹的名称，其ID属性的值为tbName。该控件使用了1个TextBoxWatermarkExtender控件和3个ValidatorCalloutExtender控件，它们的ID属性的值分别为wmeName、vceNameBlank、vceNameValue和vceNameRegex。其中，wmeName控件为tbName控件显示水印值“请输入邮箱文件夹名称”。vceNameBlank、vceNameValue和vceRevName控件分别以提示样式显示验证结果。

另外，tbName控件使用2个RequiredFieldValidator控件和1个RegularExpressionValidator控件，它们的ID属性的值分别为rfNameBlank、rfNameValue和revName。其中，rfNameBlank控件验证tbName控件的内容不能为空验证，即用户输入邮箱文件夹的名称不能为空。rfNameValue控件验证tbName控件的内容不能等于“请输入邮箱文件夹名称”（wmeName水印控件的水印值），这样可以防止用户不输入任何内容就提交到数据库。revName控件验证tbName控件的内容的最小长度为1、最大长度为20。

```
文件夹名称：<asp:TextBox ID="tbName" runat="server" SkinID="tbSkin"
    Width="60%" MaxLength="50"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfNameBlank" runat="server"
    ControlToValidate="tbName" Display="none"
    ErrorMessage="名称不能为空！"></asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="rfNameValue" runat="server"
    ControlToValidate="tbName" Display="none"
    InitialValue="请输入邮箱文件夹名称" ErrorMessage="名称不能为空！">
    </asp:RequiredFieldValidator>
<asp:RegularExpressionValidator ID="revName" runat="server"
    ControlToValidate="tbName" Display="none"
    ErrorMessage="邮箱文件夹名称的长度最大为20，请重新输入。"
    ValidationExpression=".{1,20}"></asp:RegularExpressionValidator>
<ajaxToolkit:TextBoxWatermarkExtender ID="wmeName" runat="server"
  TargetControlID="tbName" WatermarkText="请输入邮箱文件夹名称"
  WatermarkCssClass="Watermark"></ajaxToolkit:TextBoxWatermarkExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameBlank" runat="server"
  TargetControlID="rfNameBlank" HighlightCssClass="Validator">
  </ajaxToolkit:ValidatorCalloutExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameValue" runat="server"
  TargetControlID="rfNameValue" HighlightCssClass="Validator">
  </ajaxToolkit:ValidatorCalloutExtender>
<ajaxToolkit:ValidatorCalloutExtender ID="vceNameRegex" runat="server"
    TargetControlID="revName" HighlightCssClass="Validator">
    </ajaxToolkit:ValidatorCalloutExtender>
```

下述程序代码声明了1个Button控件，ID属性的值为btnCommit。该控件可以将用户输入的邮箱文件夹提交到数据库中。

```
<asp:UpdatePanel ID="upbutton" runat="server"><ContentTemplate>
<asp:Button ID="btnCommit" runat="server" Text="提交" SkinID="btnSkin"
    Width="100px" OnClick="btnCommit_Click" />
</ContentTemplate></asp:UpdatePanel>
```

下述程序代码声明了AddMailbox.aspx页面所必须的“@Page”指令和ScriptManager控

件等内容。

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="AddMailbox.aspx.cs" Inherits="AddMailbox"
    StylesheetTheme="ASPNETAjaxWeb" %>
<head id="Head1" runat="server"><title>添加新的邮箱文件夹</title></head>
<asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

综合上述，AddMailbox.aspx页面的设计界面如图3.22所示。

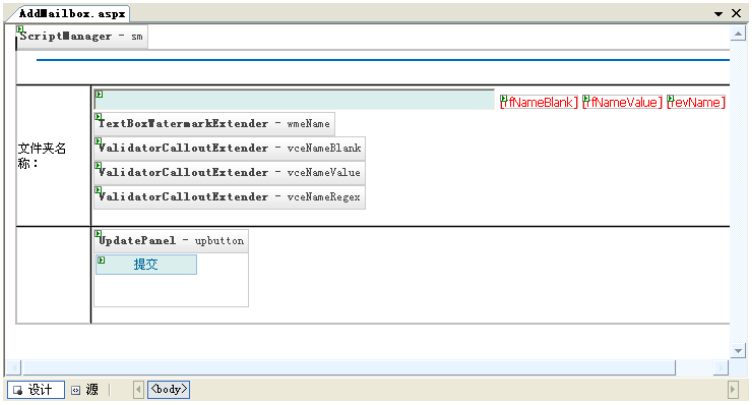

图3.22　AddMailbox.aspx页面的设计界面

2. 事件设计

用户单击AddMailbox.aspx页面中的【提交】按钮（btnCommit控件）时，将触发btnCommit控件的Click事件：btnCommit_Click(object sender,EventArgs e)。

在下述程序代码中，btnCommit_Click(object sender,EventArgs e)事件实现了将用户输入的邮件文件夹的名称提交到数据库的功能。该事件调用Mail类中的AddMailbox(string name)方法将邮箱文件夹提交到数据库。若上述操作成功，重定向到邮箱主页面Mailbox.aspx。

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
protected void btnCommit_Click(object sender,EventArgs e)
{
    Mail mail = new Mail();
    ///创建新的邮箱文件夹
    if(mail.AddMailbox(tbName.Text) > 0)
    {   ///重定向到管理页面
        Response.Redirect("~/Mailbox.aspx");
    }
}
```

3.4.8　修改邮箱文件夹

修改邮箱文件夹由UpdateMailbox.aspx页面实现，UpdateMailbox.aspx.cs文件为它的代码隐藏文件。该页面提供了输入邮箱文件夹的名称，以及将邮箱文件夹的信息保存到数据库中的功能。UpdateMailbox.aspx页面的效果图如图3.23所示。

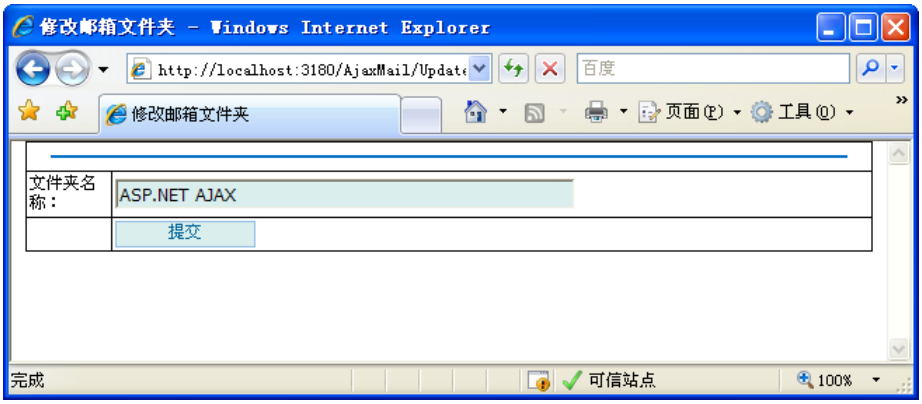

图3.23　UpdateMailbox.aspx页面的效果图

1. 界面设计

UpdateMailbox.aspx页面添加了1个TextBox控件和1个Button控件，它们的ID属性的值分别为tbName和btnCommit。由于该页面的设计界面和AddMailbox.aspx页面的设计界面非常相似，因此，在此不做详细介绍。UpdateMailbox.aspx页面的设计界面如图3.24所示。

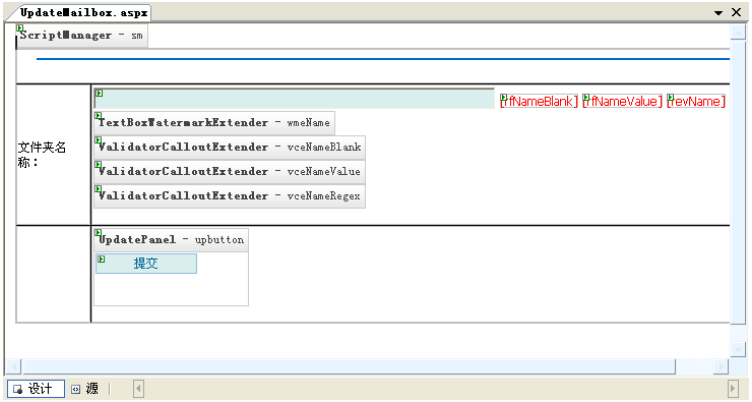

图3.24　UpdateMailbox.aspx页面的设计界面

下述程序代码声明了UpdateMailbox.aspx页面所必须的“@Page”指令和ScriptManager控件等内容。

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="UpdateMailbox.aspx.cs" Inherits="UpdateMailbox"
    StylesheetTheme="ASPNETAjaxWeb" %>
<head id="Head1" runat="server"><title>修改邮箱文件夹</title></head>
<asp:ScriptManager ID="sm" runat="server"></asp:ScriptManager>
```

2. 初始化

UpdateMailbox.aspx.cs文件首先引入了2个新的命名空间：System.Data.SqlClient和ASPNETAJAXWeb.AjaxMail，然后定义了1个变量：mailboxID。其中，mailboxID变量用来保存被修改邮箱文件夹的ID值，它的值从UpdateMailbox.aspx页面的地址栏中获取。实现上述功能的程序代码如下：

```
///引入新的命名空间
using ASPNETAJAXWeb.AjaxMail;
using System.Data.SqlClient;
int mailboxID = -1;
```

UpdateMailbox.aspx页面的初始化功能由其Page_Load(object sender, EventArgs e)事件实现。在下述程序代码中，该事件首先设置mailboxID变量的值，然后调用BindPageData(int mailboxID)函数读取并显示被修改邮箱文件夹的名称。其中，读取数据由Mail类的GetSingleMailbox(int mailboxID)方法（返回SqlDataReader类型的对象）实现。

另外，Page_Load(object sender, EventArgs e)事件还根据mailboxID变量的值设置了btnCommit控件的可用性。如果mailboxID变量的值大于0，则该控件可用，否则不可用。

```
protected void Page_Load(object sender,EventArgs e)
{   ///获取被修改数据的ID
    if(Request.Params["MailboxID"] != null)
    {
        mailboxID = Int32.Parse(Request.Params["MailboxID"].ToString());
    }
    ///显示被修改的数据
    if(!Page.IsPostBack && mailboxID > 0){BindPageData(mailboxID);}
    ///设置按钮是否可用
    btnCommit.Enabled = mailboxID > 0 ? true : false;
}
private void BindPageData(int mailboxID)
{   ///读取数据
    Mail mail = new Mail();
    SqlDataReader dr = mail.GetSingleMailbox(mailboxID);
    if(dr == null) return;
    if(dr.Read())
    {   ///显示数据
        tbName.Text = dr["Name"].ToString();
    }
    dr.Close();
}
```

3. 事件设计

用户单击UpdateMailbox.aspx页面中的【提交】按钮（btnCommit控件）时，将触发btnCommit控件的Click事件：btnCommit_Click(object sender,EventArgs e)。

在下述程序代码中，btnCommit_Click(object sender,EventArgs e)事件实现了将用户修改后的邮箱文件夹名称提交到数据库的功能。该事件调用Mail类中的UpdateMailbox(int mailboxID,string name)方法将用户修改后的邮箱文件夹名称提交到数据库。若上述操作成功，重定向到邮箱主页面Mailbox.aspx。

```
protected void btnCommit_Click(object sender,EventArgs e)
{
    Mail mail = new Mail();
    ///修改邮箱文件夹的名称
    if(mail.UpdateMailbox(mailboxID,tbName.Text) > 0)
    {   ///重定向到管理页面
        Response.Redirect("~/Mailbox.aspx");
```

```
        }
    }
```