2

建立 Web 页面及 Http 处理程序

任务目标

- 新建数据库、数据表。
- 建立 Web 页面。
- 编写页面代码。
- 建立 Http 处理程序。
- 进行页面间传值的安全防范。

技能目标

- 建立相册、照片数据库并分析数据关系图。
- 建立 Web 页面查询照片名并编写页面代码。
- 建立 Http 处理程序。
- 掌握页面间传值的安全解决方法。

任务导航

在本项目中，显示相册与编辑相册是非常重要的功能，为了实现网站中相册和图片的显示，必须了解显示图片的方法，也就是掌握如何读取数据库中所保存的照片的存放路径信息，并在页面中显示该照片。

在本章中，首先讲述如何新建数据库，以便存储相册和照片的存放路径信息；然后介绍

如何通过自定义 Http 处理程序实现显示相册、显示指定照片和照片大小等功能；最后介绍页面间传值会出现的安全问题以及相应的安全解决方法。

技能基础

2.1 C#程序代码的基本书写规则

1. 程序代码区分字母大小写

例如 Console 和 console 在 C#中就是不同的标识符。

2. 语句书写规则

（1）每个语句都必须用一个分号（;）作为结尾。

（2）C#允许在同一个代码行上书写多个语句。但从可读性的角度来看，这种做法不宜提倡，良好的编程习惯是一个语句写成一行。

（3）C#是一种块结构的编程语言，所有的语句都是代码块的一部分。每个代码块用一对花括号（{、}）来界定，花括号本身不需要使用分号来结束。一个代码块中可以包含任意多行语句，也可以嵌套包含其他代码块。

（4）语句中作为语法成分的标点符号必须是西文标点符号，中文标点符号只能作为字符常量使用。

（5）作为目前通行的程序代码标准书写规则，代码块的书写广泛采用了缩进格式，越是嵌套在内层的代码块缩进越多，这样有助于进一步提高代码的可读性。

3. 注释信息

注释信息是程序中不可执行的部分，仅对程序代码加以解释说明，编译时会将其忽略。恰当地使用注释有助于提高程序的可读性，便于软件维护和协作开发。作为一个负责任的优秀程序员，必须养成及时为程序添加注释的习惯。

C#中的注释方法有以下 3 种：

（1）单行注释。

在一个语句行上，用双斜杠“//”作为引导符，其后的任何内容均为注释信息，编译时被忽略，通常用于注释字符串较短的场合。

单行注释可以书写在可执行代码语句的后面，也可以书写成单独的一行。

方式 1：string name = Console.ReadLine();　　//输入姓名字符串赋值给 name 变量

方式 2：//输入姓名字符串赋值给 name 变量

```
string name = Console.ReadLine();
```

（2）多行注释。

从“/*”开始到“*/”结束，其中的所有内容（可以是一行或多行）均为注释信息，但注

释文字中必须不包含“*/”。多行注释通常用于需要书写较大量注释的情况。

（3）XML 注释。

在一个代码行上，用“///”开始，其后的任何内容均为注释信息，编译时被提取出来，形成一个特殊格式的文本文件（XML），用于创建文档说明书。

2.2 常量与变量

1. 常量

常量就是在程序运行过程中值保持不变的量。声明常量的语法形式为：

```
常量修饰符 const 数据类型 常量名 = 常量值;
```

其中，常量修饰符可以是 public、private、protected、internal 或 protected internal，这些访问修饰符用于定义访问该常量的方式。常量修饰符可以省略。

数据类型可以是简单类型、枚举和字符串。例如：

```
const int speedmax=60;
const double PI=3.14159265;
const int hour = 7, minute = 25, second = 30;
```

2. 变量

在程序运行过程中，其值可以改变的量称为变量（Variable）。在 C#中，变量是表示数值、字符串值或类的对象。变量存储的值可能会发生更改，但变量名称保持不变，变量是字段的一种类型。下面的代码提供了一个简单示例，表示如何声明一个整数变量并初始化，然后为它赋一个新值。

```
int y= 1;      //将变量 y 初始化为 1
y= 2;          //将变量 y 赋值为 2，原来的值将被覆盖
string greet="Hello,World";
```

在 C#中，变量是用特定数据类型和标签声明的，必须指定变量类型，可以是 int、float、byte、short 等 20 多种不同数据类型中的任何一种。在程序中，变量用来保存临时数据。定义一个变量之后，C#编译器就会在内存中为其分配一个存储区域，以保存相应的信息。

每个变量都有一个名字和相应的数据类型。变量的类型决定了可存放数据的类型，同时也确定了该内存单元的结构。变量名实际上就是内存单元的名字。程序通过变量名引用变量，可以在程序运行的不同时刻通过赋值语句向变量赋予不同的值，或者从变量中读取已存储的内容。

C#是一种强类型语言，程序中用到的所有变量都必须遵循“先声明，后引用”的原则。

在 C#中，将变量从一种类型转换为其他类型时，会对内存进行重新分配。

注意：变量被定义之后，还必须对它进行初始化，才能在程序中被引用，否则编译时就会报告出错。考虑到程序安全性的需要，C#不允许使用未初始化的变量。

（1）静态变量和实例变量。

用 static 修饰符声明的变量称为静态变量，未用 static 修饰符声明的变量称为实例变量。

静态变量不属于某个特定的实例，不管创建了多少个类实例，在任何时候静态变量都只会有一个副本。实例变量属于某个实例，即类的每个实例都包含了该类的实例变量的一个副本。

例如：

```
public static long id = 1027;
public int number;
public decimal price;
int[] animals = new int[2];
```

其中，id 为静态变量，其他为实例变量。

（2）局部变量。

在块、for 语句、switch 语句、using 语句、函数中声明的变量称为局部变量，它只有局部作用域，只在该局部范围内有效。当程序运行到这一范围时它才起作用，程序离开该范围时它就失效。

（3）隐式类型的局部变量。

在定义变量时，可以不给出其所属的数据类型，而由编译器根据变量的初始值推断出变量的数据类型，这个变量就是隐式类型的局部变量。其语法形式为：

```
var 隐式变量名 = 初始值;
```

例如：

```
var age = 30;
var name = "张三";
```

注意：其实，使用隐式类型的局部变量是有限制的。

- 定义隐式类型的局部变量时，必须初始化。
- 不能把隐式类型的局部变量初始化为 null，如 var name = null;。
- 不能在同一语句中初始化多个隐式类型的变量，如 var age=20, count=30;。
- 隐式类型的局部变量不能作为函数的返回值和参数。
- 隐式类型的局部变量不能作为类的成员。

任务实施

2.3 任务一：建立数据库

2.3.1 新建数据库

新建本项目需要的数据库，以便存储相册和照片的存放路径信息。单击“开始”/“所有程序”/Microsoft SQL Server 2005/SQL Server Management Studio Express 命令（如图 2-1 所示），即可启动 SQL Server Management Studio Express 可视化管理工具，从而可以在其中新建数据库或创建数据表等。

图 2-1　启动数据库

右击“对象资源管理器”窗格中的“数据库”目录，如图 2-2 所示，在弹出的快捷菜单中选择“新建数据库”命令，打开如图 2-3 所示的“新建数据库”窗口。

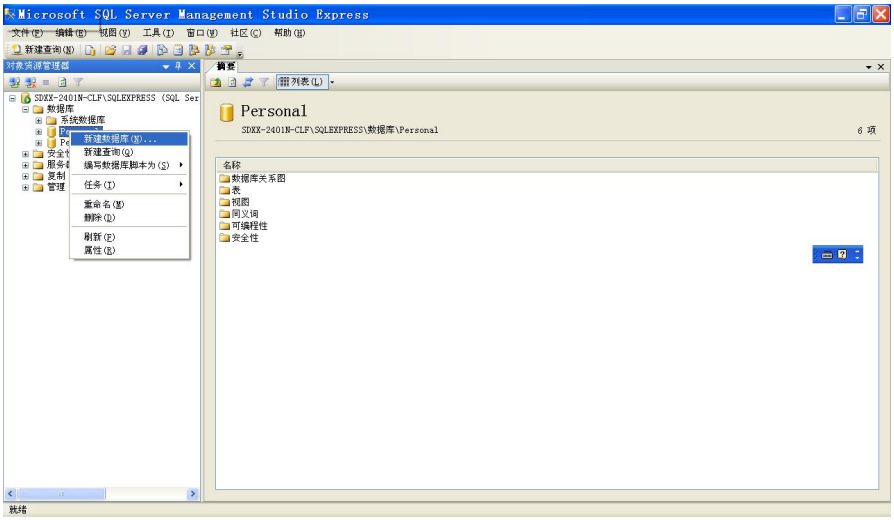

图 2-2　新建数据库操作

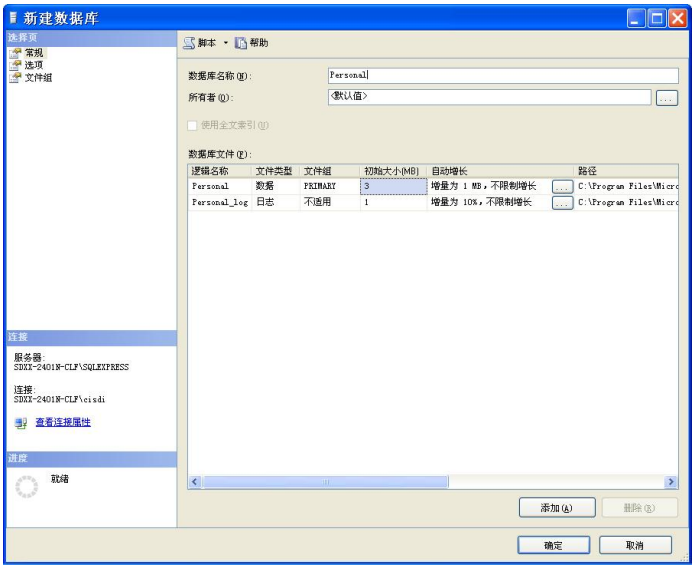

图 2-3　“新建数据库”窗口

在“数据库名称”文本框中输入数据库的名称 Personal，然后单击“确定”按钮，即可创建数据库 Personal。新建数据库 Personal 后，接下来在该数据库中新建数据表。

右击“对象资源管理器”窗格中的 Personal 数据库，在弹出的快捷菜单中选择“新建查询”命令，打开执行 SQL 语句的界面，如图 2-4 所示。

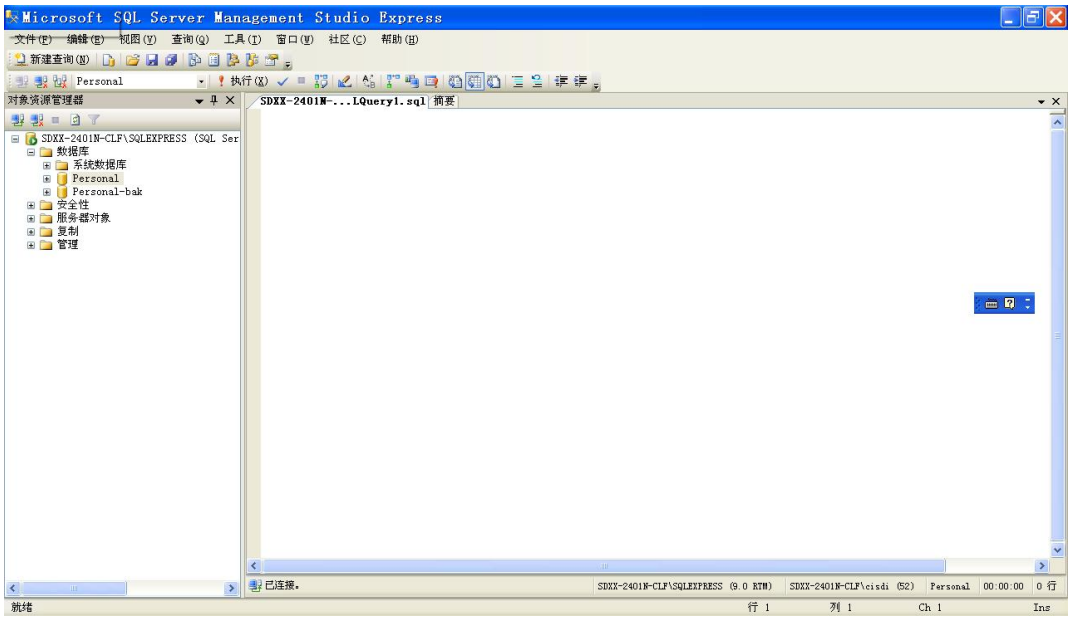

图 2-4　执行 SQL 语句界面

在执行 SQL 语句的界面中，输入代码 2-1 中的 SQL，确保在工具栏中“执行”按钮左边的下拉列表框中选择的是 Personal 数据库，然后单击“执行”按钮，即可新建相应的数据表 Albums 和 Photos，下一节具体分析这两个数据表的关系。

代码 2-1　建表 SQL

```
CREATE TABLE [dbo].[Albums](
        [AlbumID] [int] IDENTITY(1,1) NOT NULL,
        [Caption] [nvarchar](50) NOT NULL,
        [IsPublic] [bit] NOT NULL
)
ALTER TABLE [dbo].[Albums] ADD CONSTRAINT [PK_Albums]
PRIMARY KEY (AlbumID)

CREATE TABLE [dbo].[Photos](
        [PhotoID] [int] IDENTITY(1,1) NOT NULL,
        [AlbumID] [int] NOT NULL,
        [Caption] [nvarchar](50) NOT NULL,
        [OriginalFileName] [nvarchar](50) NOT NULL,
        [LargeFileName] [nvarchar](50) NOT NULL,
```

```
        [MediumFileName] [nvarchar](50) NOT NULL,
        [SmallFileName] [nvarchar](50) NOT NULL
)
ALTER TABLE [dbo].[ Photos] ADD CONSTRAINT [PK_Photos]
PRIMARY KEY (PhotoID)

ALTER TABLE [dbo].[Photos]    ADD    CONSTRAINT [FK_Photos_Albums]
FOREIGN KEY([AlbumID])    REFERENCES [dbo].[Albums] ([AlbumID])
```

2.3.2 分析数据库

右击“对象资源管理器”窗格中 Personal 下的“数据库关系图”，在弹出的快捷菜单中选择“新建数据库关系图”命令，如图 2-5 所示。

图 2-5 新建数据库关系图

选择表 Albums 和 Photos，单击“添加”按钮，如图 2-6 所示。

图 2-6 选择 Albums 和 Photos

最后生成如图 2-7 所示的数据库关系图。

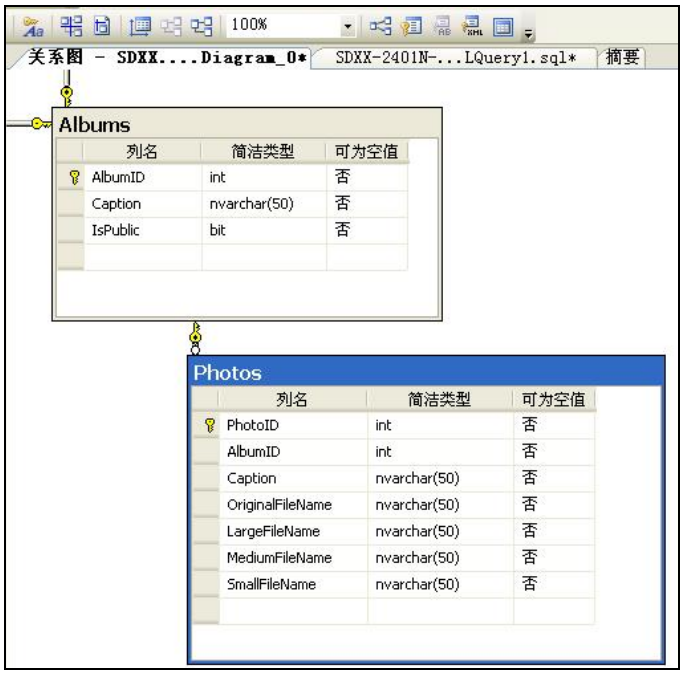

图 2-7　数据库关系图

Personal 数据库由两个数据表组成，一个表名称为 Albums，另一个表名称为 Photos。

数据表 Albums 中定义了 3 个字段：

- AlbumID：定义为主键。
- Caption：定义为存储相册的标题，用来说明该相册的内容。
- IsPublic：用来定义该相册是否可以公开，若可以公开，则任何浏览者均可以查看该相册的内容；若不允许公开，则只有能够登录进入该网站的成员才能浏览相册中的内容。

数据表 Photos 中定义了 7 个字段：

- PhotoID：定义为主键。
- AlbumID：是相册的唯一编号。
- Caption：定义为存储照片的标题，说明该照片的内容。
- OriginalFileName：定义为一个字符串类型的字段，用来存储原有照片的文件名称。
- LargeFileName：定义为一个字符串类型的字段，用来存储大像素的照片的存放路径。
- MediumFileName：定义为一个字符串类型的字段，用来存储中等像素的照片的存放路径。
- SmallFileName：定义为一个字符串类型的字段，用来存储小像素的照片的存放路径。

数据表 Albums 用来存储相册的内容，数据表 Photos 用来存储照片的相关内容，而这两个表通过 AlbumID 实现主键与外键的互相关联。

2.4 任务二：建立 Web 页面查询照片名

如图 2-8 所示为照片的文件存放位置，所有照片存放在项目目录的 Images 目录下，该目录下按照照片的大、中、小尺寸分为 3 个子目录：Large、Medium、Small。

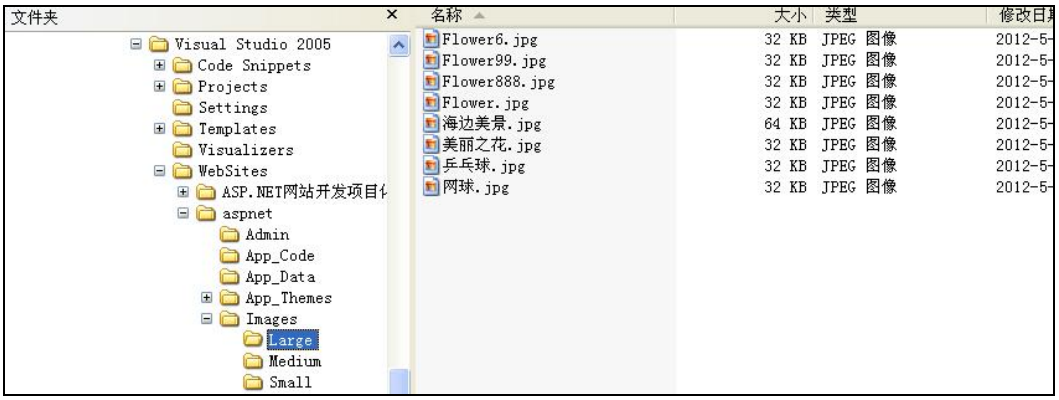

图 2-8 照片的文件存放位置

接下来建立一个网页，实现通过该网页查看照片名的功能。

2.4.1 新建 Web 页面

右击 Visual Studio 2005 左侧“解决方案资源管理器”窗格中的项目，在弹出的快捷菜单中选择“添加新项”命令，如图 2-9 所示。

图 2-9 添加新项

弹出“添加新项”对话框，在其中选择“Web 窗体”模板，在“名称”文本框中输入需要创建的页面名称为 DefaultLook.aspx，单击“添加”按钮，如图 2-10 所示。

在 Visual Studio 2005 中查看 DefaultLook.aspx 的视图，如图 2-11 所示。

图 2-10　创建页面

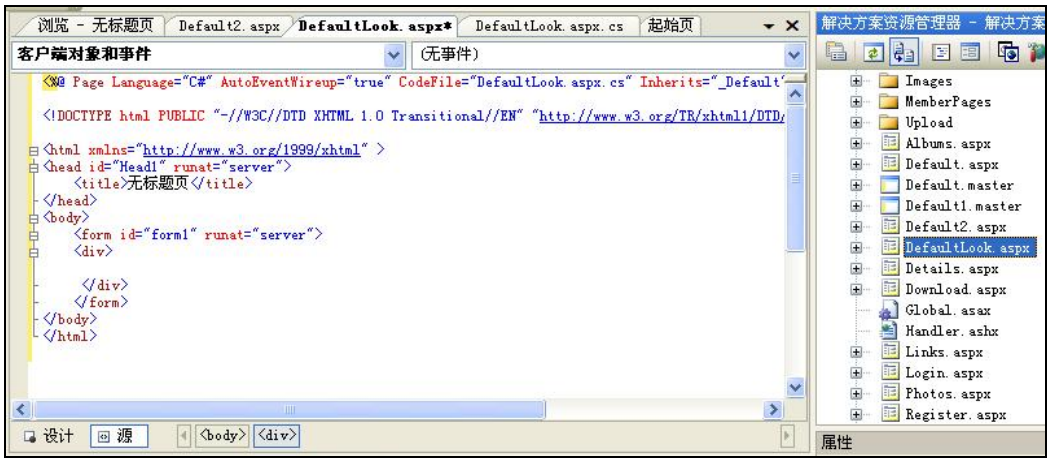

图 2-11　DefaultLook.aspx 的设计视图

从工具箱中拖放 1 个 Panel、1 个 TextBox、1 个 Button 控件到 DefaultLook.aspx 的设计视图中，如图 2-12 所示。

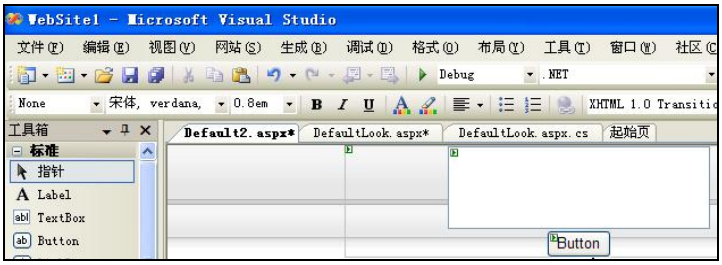

图 2-12　拖放 Button 控件

再从工具箱中拖放 1 个 Panel、1 个 Literal 控件到 DefaultLook.aspx 的设计视图中，并将控件的属性 Text 改为“得到文件名”，如图 2-13 所示。

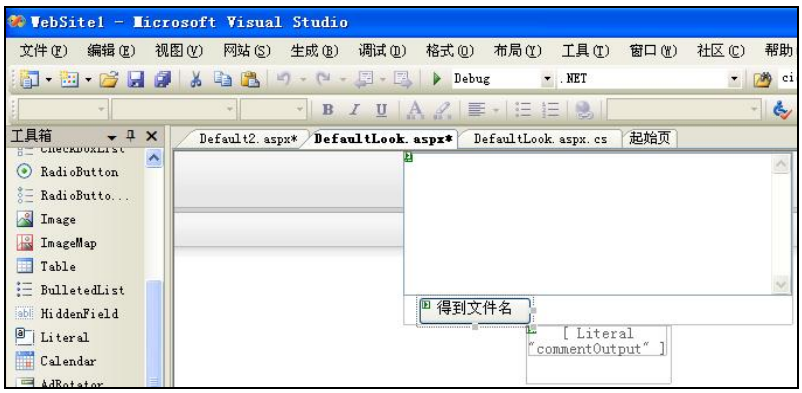

图 2-13　拖放 Literal 控件

最后得到代码 2-2 中的源代码。

代码 2-2　DefaultLook.aspx 的设计源代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="DefaultLook.aspx.cs" Inherits="_Default" ValidateRequest
="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Panel ID="commentPrompt" runat="server">
            <asp:TextBox ID="commentInput" runat="server" TextMode="MultiLine" Height="115px"
                Width="327px" /><br />
            <asp:Button ID="submit" runat="server"    OnClick="Button1_Click" Text="得到文件名" />
        </asp:Panel>

        <asp:Panel ID="commentDisplay" runat="server" Visible="false">
            <asp:Literal ID="commentOutput" runat="server" />
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

2.4.2 编写 Web 页面的代码

双击控件“得到文件名”转到代码视图，如图 2-14 所示，我们将在 Button1_Click 事件中写入代码。

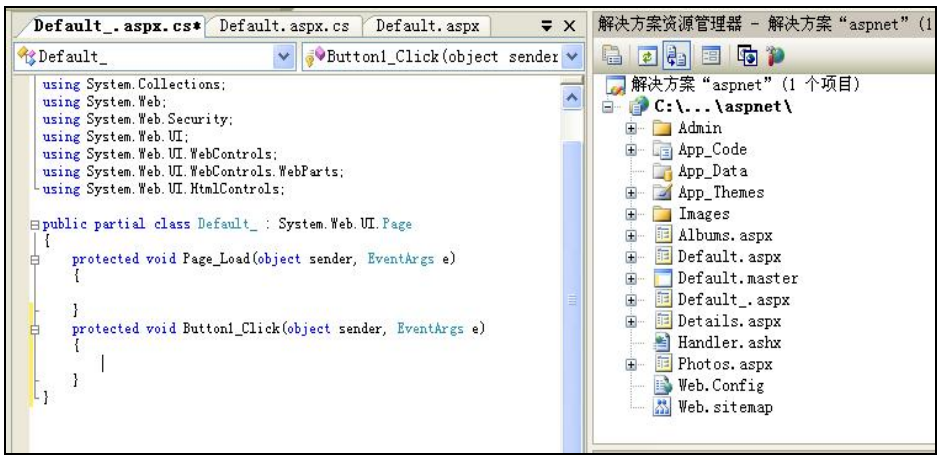

图 2-14 Button1_Click 事件代码

写入代码 2-3 中的代码，要养成良好的编程习惯，注意在代码中加入注释，便于以后自己理解，也便于后期的维护。

代码 2-3 Button1_Click 事件代码

```
protected void Button1_Click(object sender, EventArgs e)
{
        string sName;       //根据 photoId 得到的文件名
        this.commentDisplay.Visible = true;
        try
        {
            //将输入文本框中的数字字符串通过 Int32 类中的 Parse 方法转换成数字类型
            //然后调用 GetPhoto 方法，获得查询数据表 Photos 之后的结果
            sName= GetPhoto(Int32.Parse(Request["commentInput"]));
        }
        catch
        {
            sName="不存在";
        }
        this.commentOutput.Text = "PhotoID 为"+Request["commentInput"] +"的文件名是"+sName;
}
public static String GetPhoto(int photoId)
{
            //数据库连接字符串不是直接硬写数据库连接字符串
            //而是通过类 ConfigurationManager 读取 Web.config 文件中所设置数据库 Personal 的数据库连接字符串
            //这样即使修改数据库连接串，只需要在 Web.config 文件中修改，而不用修改代码，可维护性好
```

```
            //然后通过数据库连接字符串构造一个数据库连接 SqlConnection
            SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings
            ["Personal"].ConnectionString);
            //构造 SQL 查询字符串，查询结果是 OriginalFileName，该查询构造了条件语句，即只有被公开的照片才能
            //被查询
            string sql = " SELECT TOP 1 [OriginalFileName] FROM [Photos] LEFT JOIN [Albums] ON [Albums].[AlbumID]
            = [Photos].[AlbumID] " +"    WHERE [PhotoID] = @PhotoID AND ([Albums].[IsPublic] = @IsPublic OR
            [Albums].[IsPublic] = 1) ";
            //构造 SqlCommand 对象
            SqlCommand command = new SqlCommand(sql, connection);
            //设置了查询语句中的查询参数，当查询语句中含有参数时不能直接对查询参数赋值
            //需要通过这种设置方式来设置查询参数，否则会导致数据库不安全
            command.Parameters.Add(new SqlParameter("@PhotoID", photoId));
            command.Parameters.Add(new SqlParameter("@IsPublic", true));
            //打开数据库连接
            connection.Open();
            //实现数据库的查询，得到查询结果
            object result = command.ExecuteScalar();
            try
            {
                //处理获得的结果
                return ((string)result);
            }
            catch
            {
                return null;
            }
    }
```

在“解决方案资源管理器”窗格中右击 DefaultLook.aspx，在弹出的快捷菜单中选择“在浏览器中查看”命令，如图 2-15 所示。

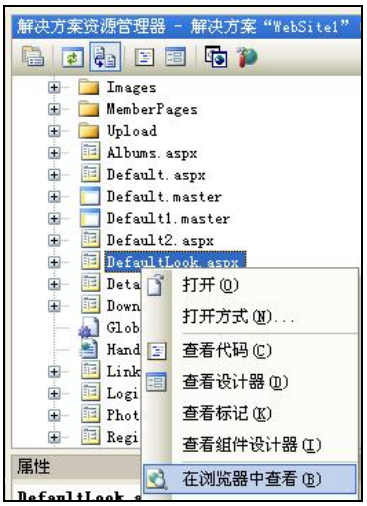

图 2-15　在浏览器中查看

运行后，在第一个文本框中输入 photoID 为 1，然后单击“得到文件名”按钮，如图 2-16 所示。

图 2-16　输入 photoID

结果如图 2-17 所示。

图 2-17　运行结果

2.5　任务三：建立 Http 处理程序

2.5.1　认识 Http 处理程序

为什么要使用 Http 处理程序呢？因为本项目中有一个功能是根据调用文件生成大、中、小的图像进行显示，此功能运行中不需要进行用户界面交互操作，要求快速执行，所以我们考虑使用 Http 处理程序。

ASP.NET 页面内部机制很复杂（例如有中间事件、视图管理等），这就降低了 ASP.NET 页面服务请求的速度。而 Http 处理程序会比 ASP.NET 页面快，Http 处理程序可以访问应用程序上下文，包括会话信息等。当请求 Http 处理程序时，ASP.NET 将调用相应处理程序上的

ProcessRequest方法。处理程序的 ProcessRequest 方法创建一个响应，此响应随后发送回请求浏览器。

如何创建自定义 Http 处理程序呢？若要创建一个自定义 Http 处理程序，需要创建一个可实现IHttpHandler接口的类以创建同步处理程序，接口要求实现 ProcessRequest 方法。

2.5.2 建立 Http 处理程序

上节介绍了通过该网页可以查看想看的照片名，本节将介绍如何通过自定义建立 Http 处理程序实现让用户查看照片。

要创建一个自定义 Http 处理程序，需要创建一个可实现IHttpHandler接口的类以创建同步处理程序，接口要求实现 ProcessRequest 方法。

右击 Visual Studio 2005“解决方案资源管理器”窗格中的项目，在弹出的快捷菜单中选择“添加新项”命令，在弹出的“添加新项”对话框中选择“一般处理程序”模板，在“名称”文本框中输入需要创建的页面名称为 Handler.ashx，然后单击“添加”按钮，如图 2-18 所示。

图 2-18 创建页面

实现 ProcessRequest 方法，其中调用了 3 个方法：GetPhoto、GetFirstPhoto、GetPath，代码如代码 2-4 所示。

代码 2-4 ProcessRequest 方法

```
public void ProcessRequest (HttpContext context) {
    //此为 Http 处理程序的程序入口
    context.Response.ContentType = "image/jpeg";
    //为得到指定图片文件路径以及指定的图片大小参数，需要通过 Http 传输图片大小参数 Size
    //所以通过条件语句设置图片的枚举 PhotoSize 参数
    PhotoSize size;
```

```
        switch (context.Request.QueryString["Size"])
        {
            case "S":
                size = PhotoSize.Small;
                break;
            case "M":
                size = PhotoSize.Medium;
                break;
            case "L":
                size = PhotoSize.Large;
                break;
            default:
                size = PhotoSize.Original;
                break;
        }
        Int32 id = -1;
        Stream stream = null;
        String path = null;
        //获得指定图片编号 PhotoID
        if (context.Request.QueryString["PhotoID"] != null &&
            context.Request.QueryString["PhotoID"] != "")
        {
            id = Convert.ToInt32(context.Request.QueryString["PhotoID"]);
            //调用 GetPhoto 方法获得指定图片编号 PhotoID 的存放文件名
            path = GetPhoto(id);
        }
        else if (context.Request.QueryString["AlbumID"] != null && context.Request.QueryString["AlbumID"] != "")
        {
            id = Convert.ToInt32(context.Request.QueryString["AlbumID"]);
            path = GetFirstPhoto(id);
        }
        //调用 GetPath 方法获得指定图片大小的文件存放目录，和图片文件名组成文件路径
        path = GetPath(path, size);
        //判断是否得到了图片的存放路径，如果没有则说明没有指定图片编号 PhotoID 或者没有指定相册编号 AlbumID
        //或者指定了相册编号但该相册中没有任何图片，此时就会通过调用 GetPhoto 方法，根据不同的图片大小参
        //数读取默认的图片路径
        if (!path.Contains(".jpg"))
            path = GetPhoto(size);
        stream = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read);
        const int buffersize = 1024 * 16;
        byte[] buffer = new byte[buffersize];
        int count = stream.Read(buffer, 0, buffersize);
        while (count > 0)
        {
            context.Response.OutputStream.Write(buffer, 0, count);
            count = stream.Read(buffer, 0, buffersize);
        }
    }
```

```
public static String GetPhoto(PhotoSize size)
{
    //根据不同的图片大小参数读取默认的图片路径
    string path = HttpContext.Current.Server.MapPath("~/Images/");
    switch (size)
    {
        case PhotoSize.Small:
            path += "placeholder-100.jpg";
            break;
        case PhotoSize.Medium:
            path += "placeholder-200.jpg";
            break;
        case PhotoSize.Large:
            path += "placeholder-600.jpg";
            break;
        default:
            path += "placeholder-600.jpg";
            break;
    }
    return path;
}

public static String GetPhoto(int photoId)
{
    //本方法获得指定图片编号 PhotoID 的存放文件名
    //新建一个数据库连接对象
    SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["Personal"]
    .ConnectionString);
    //构造查询的 SQL 语句
    string sql = " SELECT TOP 1 [OriginalFileName] FROM [Photos] LEFT JOIN [Albums] ON [Albums].[AlbumID]
    = [Photos].[AlbumID] " +" WHERE [PhotoID] = @PhotoID AND ([Albums].[IsPublic] = @IsPublic OR [Albums]
    .[IsPublic] = 1) ";
    //构造 SqlCommand 对象
    SqlCommand command = new SqlCommand(sql, connection);
    //设置 SQL 语句中的参数
    command.Parameters.Add(new SqlParameter("@PhotoID", photoId));
    command.Parameters.Add(new SqlParameter("@IsPublic", false));
    connection.Open();
    //查询数据
    object result = command.ExecuteScalar();
    try
    {
        //处理查询结果
        return ((string)result);
    }
    catch
    {
        return null;
    }
}
```

```
        public static String GetFirstPhoto(int albumId)
        {
            //新建一个数据库连接对象
            SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["Personal"]
            .ConnectionString);
            //构造查询的 SQL 语句
            string sql = " SELECT TOP 1 [OriginalFileName] FROM [Photos] LEFT JOIN [Albums] ON [Albums].[AlbumID]
            = [Photos].[AlbumID] " +"    WHERE [Albums].[AlbumID] = @AlbumID AND ([Albums].[IsPublic] = @IsPublic
            OR [Albums].[IsPublic] = 1) ";
            SqlCommand command = new SqlCommand(sql, connection);
            command.Parameters.Add(new SqlParameter("@AlbumID", albumId));
            command.Parameters.Add(new SqlParameter("@IsPublic", false));
            connection.Open();
            object result = command.ExecuteScalar();
            try
            {
                return (string)result;
            }
            catch
            {
                return null;
            }
        }

        static private string GetPath(string path, PhotoSize size)
        {
            //获得指定图片大小的文件存放目录，和图片文件名组成文件路径
            switch (size)
            {
                case PhotoSize.Large:
                    //加上路径
                    path = "Large/" + path;
                    break;
                case PhotoSize.Original:
                    break;
                case PhotoSize.Small:
                    path = "Small/" + path;
                    break;
                default:
                    path = "Medium/" + path;
                    break;
            }
            if (path != null)
                path = HttpContext.Current.Server.MapPath("~/Images/") + path;
            return path;
    }
```

2.5.3 运行 Http 处理程序

接下来测试上节新建的 Http 处理程序，运行网站，首先显示指定 photoID 为 31 的图片，在

IE 地址栏中输入 http://localhost:2524/aspnet/Handler.ashx?photoId=31，运行结果如图 2-19 所示。

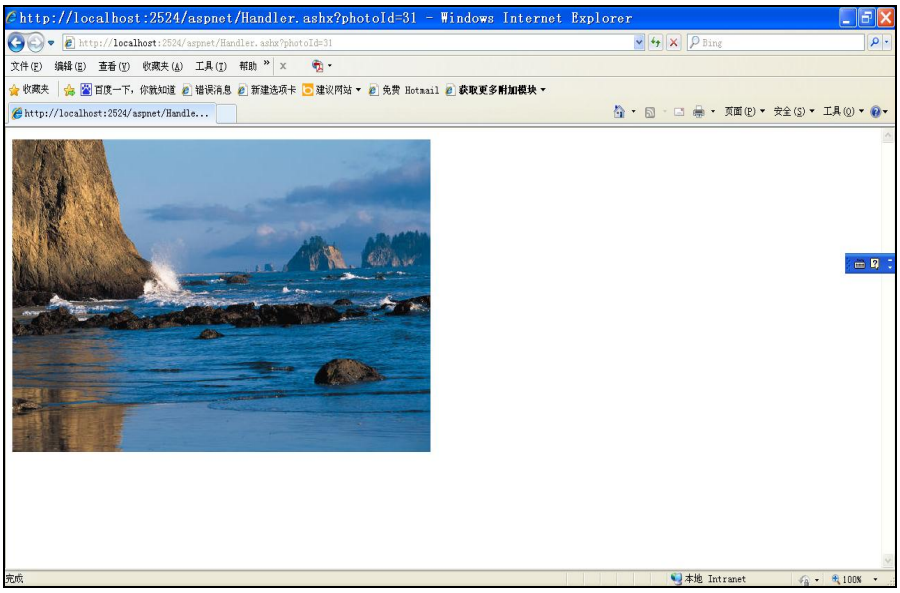

图 2-19　运行结果

接着显示指定 photoID 为 31 并且大小为小尺寸的图片，在 IE 地址栏中输入 http://localhost:2524/aspnet/Handler.ashx?photoId=31&size=S，运行结果如图 2-20 所示。

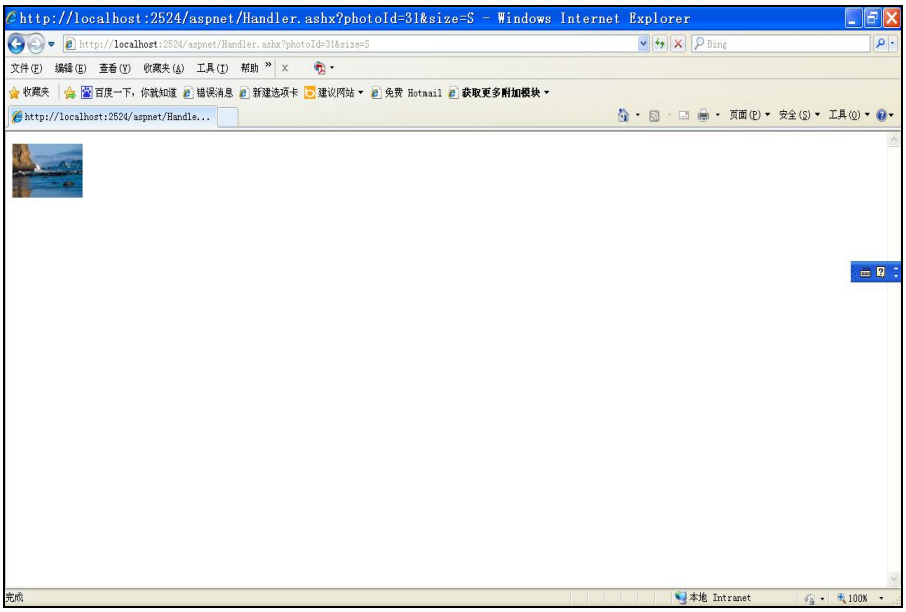

图 2-20　运行结果

再显示指定 photoID 为 33 并且大小为中尺寸的图片，在 IE 地址栏中输入 http://localhost:2524/aspnet/Handler.ashx?photoId=33&size=M，运行结果如图 2-21 所示。

图 2-21　运行结果

最后显示指定相册中的第一张照片并且大小为中尺寸的图片，于是在 IE 地址栏中输入 http://localhost:2524/aspnet/Handler.ashx?AlbumID=9&size=M，运行结果如图 2-22 所示。

图 2-22　运行结果

2.6　任务四：页面间传值的安全防范

2.6.1　页面间传值的安全问题

上节中页面间传值的 URLhttp://localhost:1738/WebSite1/Handler.ashx?AlbumID=1存在安全问题，用户可以修改查询字符串中传递的值，例如将 AlbumID=1 改成 AlbumID=8，这样修改

甚至可以访问到没有权限的页面，要解决查询字符串的安全问题，可以使用下面介绍的加密、解密方法对查询字符串中要传递的值进行加密。

使用 DESCryptoServiceProvider 类来实现加密与解密功能，DESCryptoServiceProvider 类的常用方法如下：

（1）DESCryptoServiceProvider • CreateEncryptor 方法。

该方法用指定的密钥（Key）和初始化向量（IV）创建对称数据加密标准解密对象，语法如下：

```
Public override IcryptoTransform CreateEncryptor(
        Byte []rgbKey
        Byte[]rgbIV
)
```

参数说明：

- rgbKey：用于对称算法的密钥。
- rdbIV：用于对称算法的初始化向量。
- 返回值：对称 DES 解密器对象。

（2）DESCryptoServiceProvider • CreateDecryptor 方法。

该方法用指定的密钥（Key）和初始化向量（IV）创建对称数据解密标准加密对象，语法如下：

```
Public override IcryptoTransform CreateDecryptor(
        Byte []rgbKey
        Byte[]rgbIV
)
```

参数说明：

- rgbKey：用于对称算法的密钥。
- rdbIV：用于对称算法的初始化向量。
- 返回值：对称 DES 加密器对象。

（3）CryptoStream 构造函数。

用目标数据流、要使用的转换和流的模式初始化 CryptoStream 类的新实型。该方法的语法如下：

```
Public CryptoStream(
    Stream stream,
    IcryptoTransform transform,
    CryptoStreamMode mode
)
```

参数说明：

- stream：对其执行加密转换的流。
- transform：要对流执行的加密转换。
- mode：CryptoStream Mode 值之一。

对值的加密方法如代码 2-5 所示。

代码 2-5　对值加密方法

```
using System.Security.Cryptography;
using System.IO;
using System.Text;
private static string Encryptor (String s2)
{
    String ss;
    byte[] bytIn = System.Text.Encoding.Default.GetBytes(s2);
    byte[] iv={102,16,93,156,78,4,218,32};
    byte[] key={55,103,246,79,36,99,167,3};
    DESCryptoServiceProvider dsp = new DESCryptoServiceProvider();
    dsp.Key = key;
    dsp.IV = iv;
    ICryptoTransform ict = dsp.CreateEncryptor();
    MemoryStream ms = new MemoryStream();
    CryptoStream cs = new CryptoStream(ms, ict, CryptoStreamMode.Write);
    cs.Write(bytIn, 0, bytIn.Length);
    cs.FlushFinalBlock();
    ss=Convert.ToBase64String(ms.ToArray());
    return ss;
}
```

对值的解密方法如代码 2-6 所示。

代码 2-6　对值解密方法

```
private static string Decryptor(String ss)
{
    byte[] bytIn = Convert.FromBase64String(ss);
    byte[] iv ={ 102, 16, 93, 156, 78, 4, 218, 32 };
    byte[] key ={ 55, 103, 246, 79, 36, 99, 167, 3 };
    DESCryptoServiceProvider dsp = new DESCryptoServiceProvider();
    dsp.Key = key;
    dsp.IV = iv;
    ICryptoTransform ict = dsp.CreateDecryptor();
    MemoryStream ms = new MemoryStream(bytIn);
    CryptoStream cs = new CryptoStream(ms, ict, CryptoStreamMode.Read);
    StreamReader sr = new StreamReader(cs, Encoding.Default);
    ss = sr.ReadToEnd();
    return ss;
}
```


2.6.2　本项目页面间传值的安全解决方法

本项目中，将代码 2-6 的方法进行修改，如代码 2-7 所示。

代码 2-7　页面间传值的方法

```
public void ProcessRequest (HttpContext context) {
    //此为 Http 处理程序的程序入口
    …

    Int32 id = -1;
    Stream stream = null;
    String path = null;
    //获得指定图片编号 PhotoID
    if (context.Request.QueryString["PhotoID"] != null &&
        context.Request.QueryString["PhotoID"] != "")
    {
        //id = Convert.ToInt32(context.Request.QueryString["PhotoID"]);
        id= Convert.ToInt32(Decryptor(context.Request.QueryString["PhotoID"]) );
        path = GetPhoto(id);
    }
    else if (context.Request.QueryString["AlbumID"] != null && context.Request.QueryString["AlbumID"] != "")
    {
        //id = Convert.ToInt32(context.Request.QueryString["AlbumID"]);
        id = Convert.ToInt32(Decryptor(context.Request.QueryString["AlbumID"]));
        path = GetFirstPhoto(id);
    }
 …
}
```

加密 URL 参数 AlbumID 值后，输入 http://localhost:1738/WebSite1/Handler.ashx?AlbumID=v9JRx0NQ0KI=，结果如图 2-23 所示。

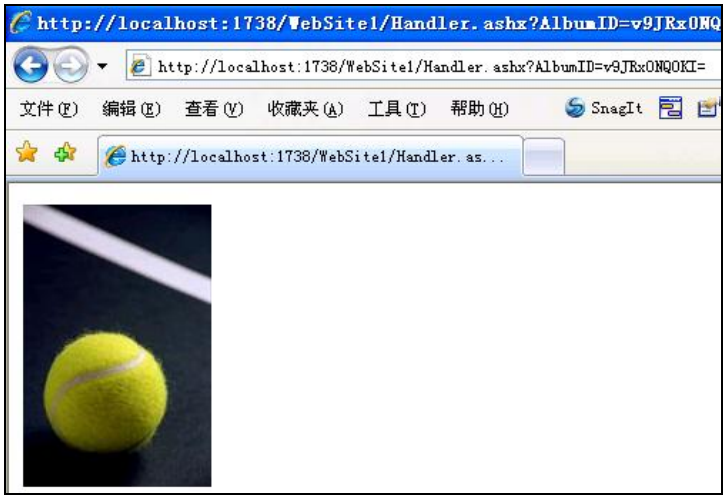

图 2-23　加密后的运行结果

如果输入http://localhost:1738/WebSite1/Handler.ashx?AlbumID=1，结果如图 2-24 所示，不能访问，有效地解决了安全问题。

图 2-24　加密后的运行结果

综合练习

1．在网页间如何传递参数以及如何获得传递参数的值？

2．修改 Http 处理程序，实现通过文件名和图片大小显示图片。

3．新建一个数据库，存放图片的路径、图片标题、拍摄日期信息。

4．将手机拍摄的照片存放在指定目录中，利用第 3 题建立的数据库，编写自定义 Http 处理程序，显示这些照片。