ASP.Net开发入门教程

# 前言

.NET是基于Windows操作系统运行的操作平台的软件应用，支持C#、VB、F#、C++多种开发语言，我们可以采用.Net平台开发桌面应用、Web应用（例如网站）、移动应用。当使用.Net进行Web应用开发时会用到基于网站的编程框架，此框架就是ASP.Net。

本课程以C#语言开发Web应用为案例，讲解Asp.Net开发技巧，实现基本的增、删、改、查功能。

# 开发工具介绍

1. Microsoft Visual Studio 2010

.net集成开发环境，它简化了有关创建、调试和部署应用程序的基本任务。借助于 Visual Studio 2010，我们可以轻松实现的目标，是.net开发的利器。

1. Microsoft SQL Server 2008

SQLServer2008 作为微软的关系型数据库管理系统，是我用过的RDBMS中操作最为方便的一款，它操作简便，与Visual Studio 2010集成更便捷、高效。

1. WeiSha.Data.Generete

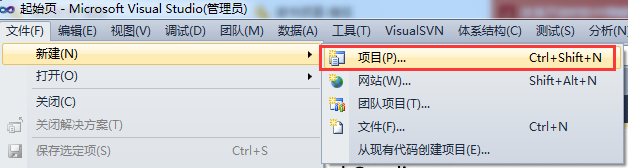
该工具为数据实体生成工具，我们课程示例将采用ORM实现对数据的增删改查。ORM中文释意为“对象关系映射（Object Relational Mapping）”，是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。通过它我们可以快速实现对数据库的操作。

**上述工具可以从云盘下载：**<http://pan.baidu.com/s/1qXNWZyk>

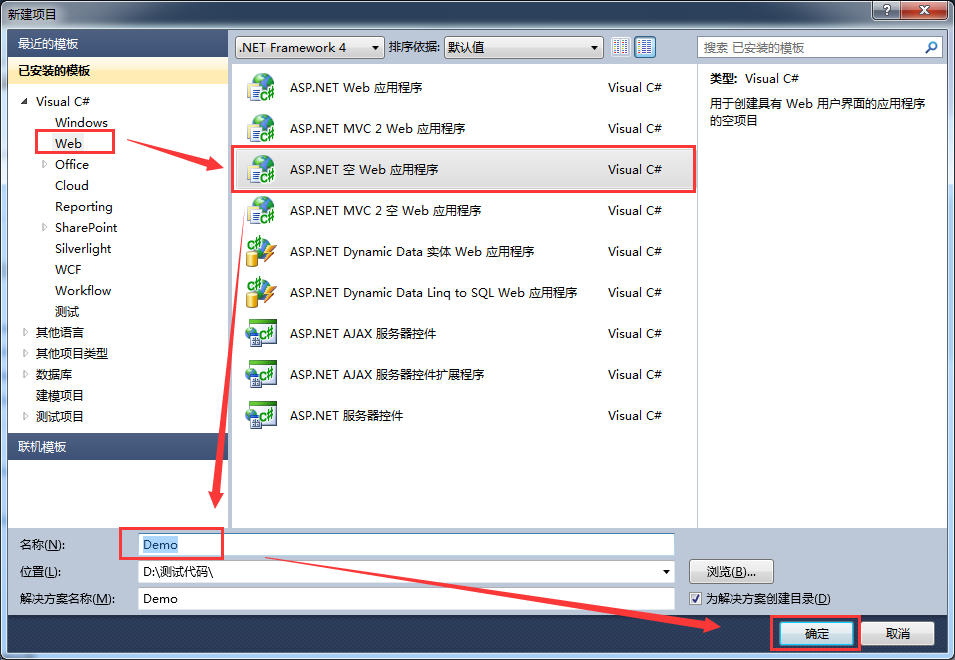
# 开发示例

## 创建项目

在下面示例中，我们创建一个基于C#语言的Web应用程序。首先，在开始菜单中打开Microsoft Visual Studio 2010，在窗体的左上角，点击“文件”->“新建”->“项目”，如下图所示：



在打开的窗体中，选择Visual C#下面的Web，如下图所示：

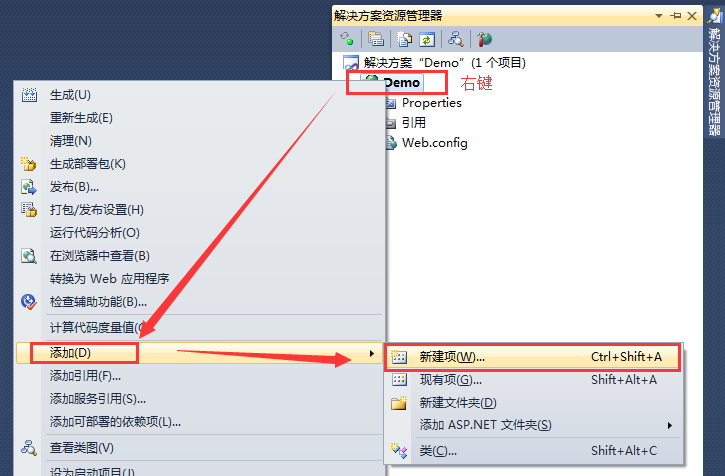


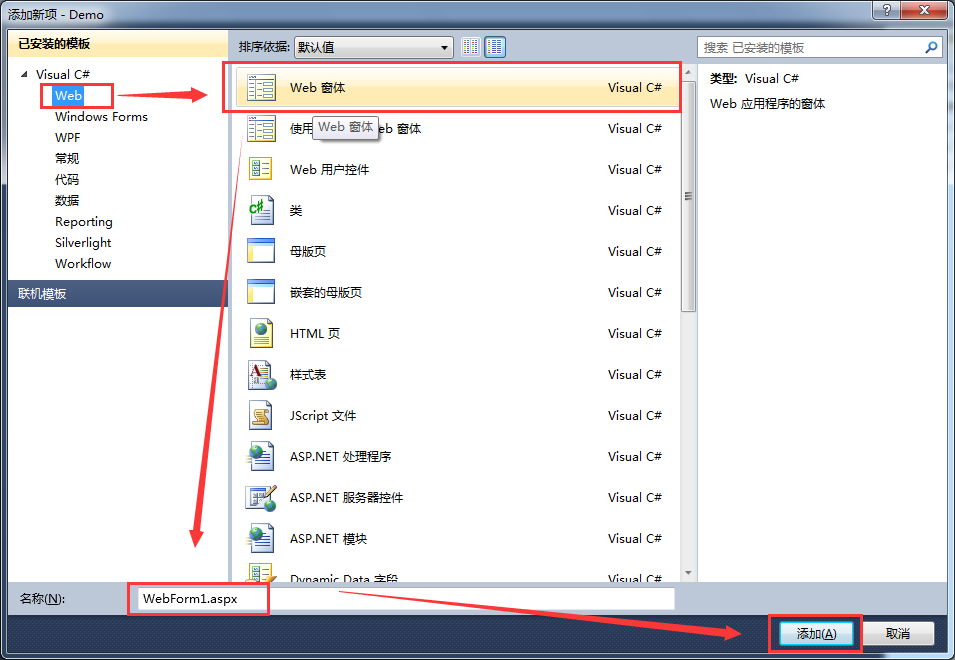
点击确定后，项目即创建完成，这个时候，只是一个空项目，什么内容都没有。

## HelloWorld

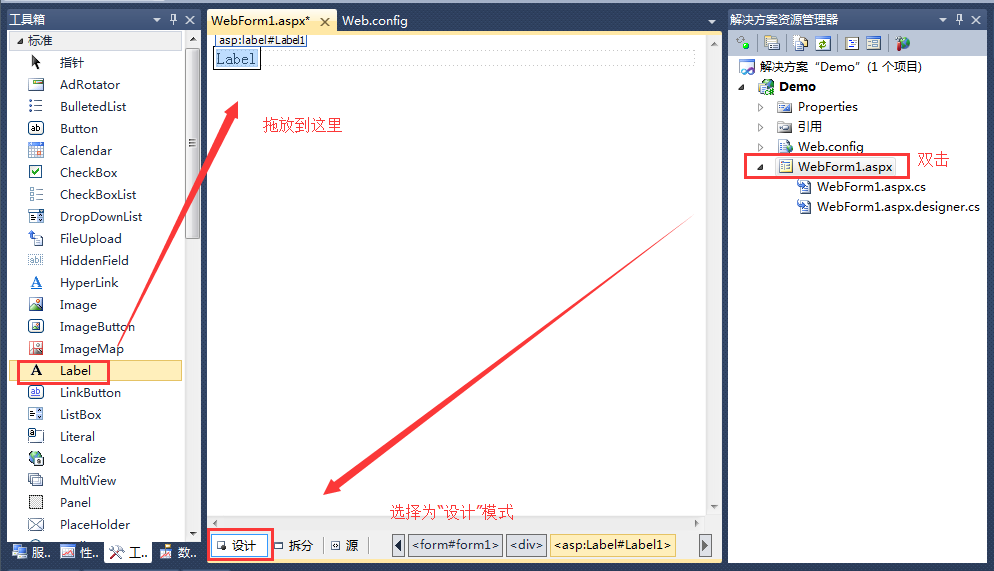
我们新建一个Asp.net的WebForm页面（即Web窗体），从而来理解asp.net是如何工作的。

如下图所示，在项目名称上点击鼠标右键，选择“添加”->“新建项”，创建Web窗体。

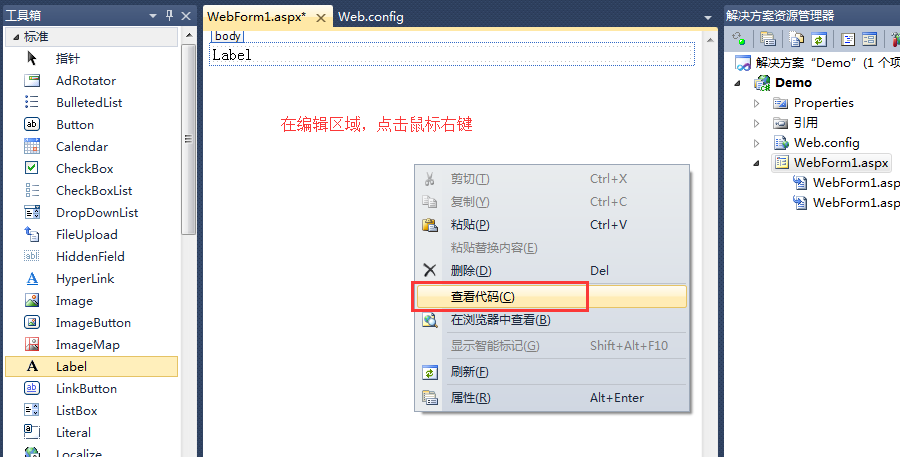




然后在右侧的解决方案区域中双击新创建的WebForm1.asp，在编辑区打开，在左侧工具箱中拖一个Label控件到界面中。如下图所示：

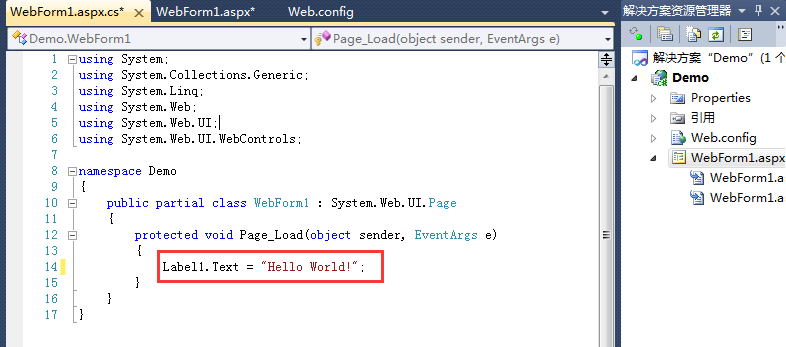


接下来在编辑区域点击鼠标右键，选择“查看代码”，如下图所示：

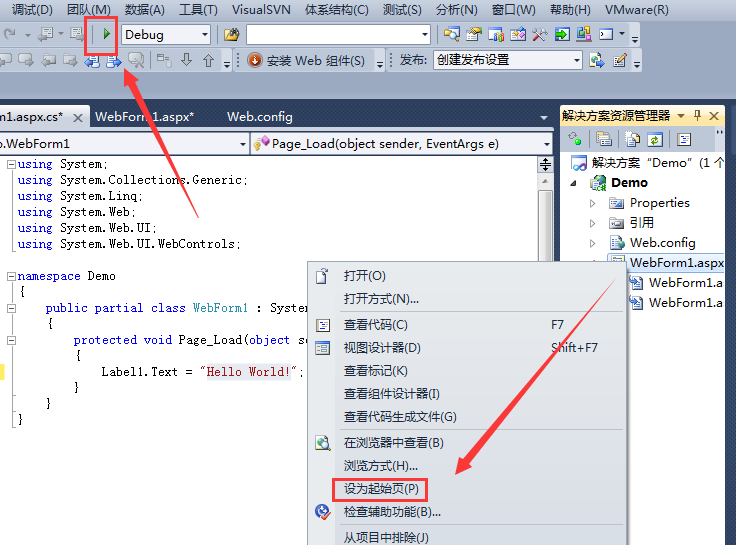


进入代码编辑界面，在Page\_Load方法中键入 Label1.Text=” Hello World!”;

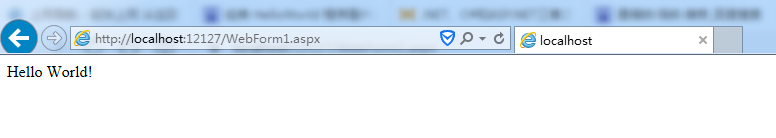
其中，Page\_Load方法是页面加载时会运行里面的代码；Label1是我们刚才拖到编辑区的那个label控件，系统自动命名为Label1。详情如下图：



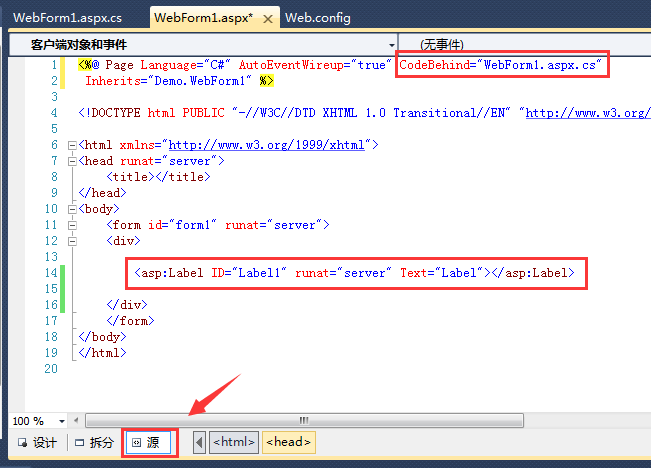
我们可以运行一下看看效果，首先要设置webform1.aspx为启始页，然后再点击执行，如下图所示：



运行效果如下：



可能有朋友不太理解为什么这样，我们看一下webform1.aspx网页代码，回到webform1.aspx编辑界面，进行“源”，如下图所示：



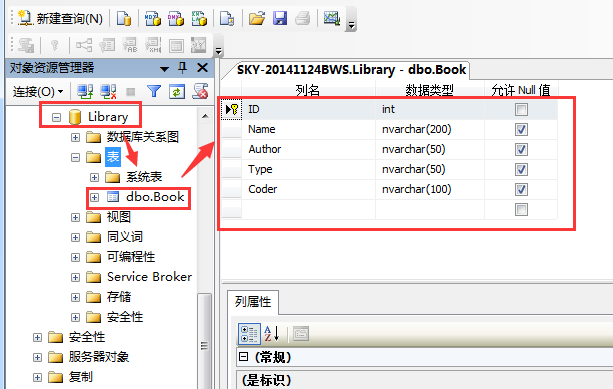
Asp.net采用CodeBehind（即代码后置）方式，在传统asp中，WEB页面的UI与逻辑都是写在一个页面里而无法分开。这种编写代码方式的最大弊端就是一旦UI或逻辑有所变动，都几乎不可避免的要对另一层进行相应改动，同时这种混合的代码编写方式暴露了程序的实现逻辑。Asp.net解决了这些问题。

上图中间的红框是label控件的标记，Label是一个ASP.net的服务器控件，服务器控件是一系列封装了用户界面及其相关功能的控件，它能够帮助我们更方便的编写Web应用，实现交互功能。

## 创建数据库

本课程主要讲解.Net开发，数据库并不是重点。但是软件开发总是会需要用到数据库（前面提到过大多数管理软件都是增删改查）进行数据的存储记录、查询分析。

我们做一个简单的数据库Library（图书馆），并创建一个记录图书的表Book，下图中右侧为Book的字段设置。

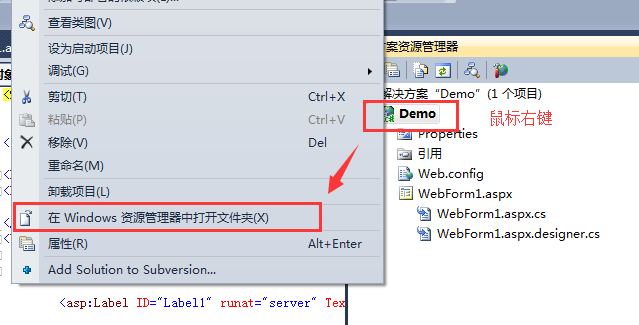


其中ID为表的自增主键，一定要主键，否则会报错。

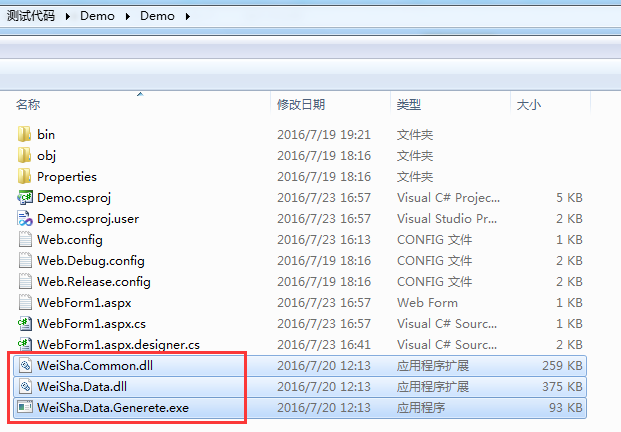
## 生成ORM实体

进行数据库的操作自然是采用ADO.Net，需要先链接数据库、创建链接对象、编写Sql执行、返回查询结果等一系列操作，我们此处采用ORM方式进行数据库操作，大大简化了编写流程。以下是操作实例。

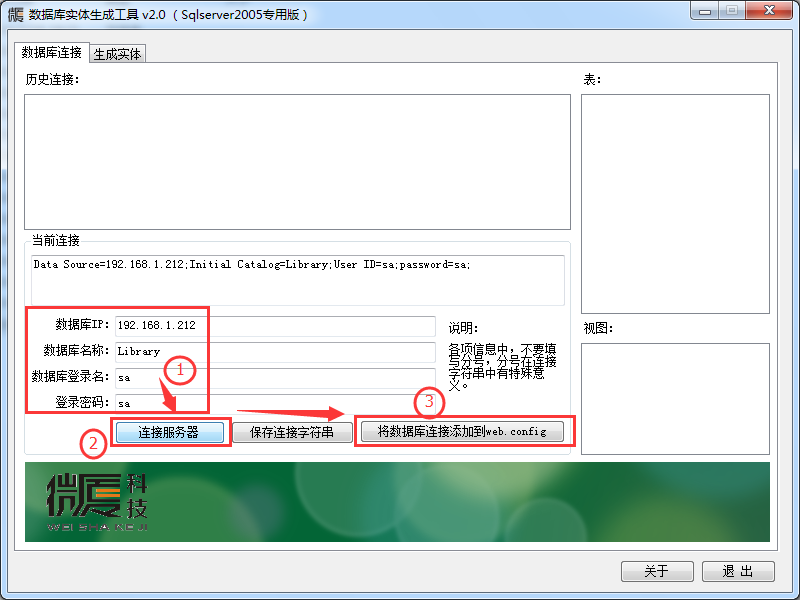
在进行数据库的操作之前，需生成ORM实体（每次数据库结构有变化，也需要重新生成），首先把WeiSha.Data.Generete.exe件复制到项目文件夹。如果找不到项目的文件夹，可以在项目名称上点击右键，选择“在windows资源管理器中打开 文件夹”，如下图：



复制三个文件，WeiSha.Data.Generete.exe与Weisha.Data.dll、Weisha.Common.dll，如下图：

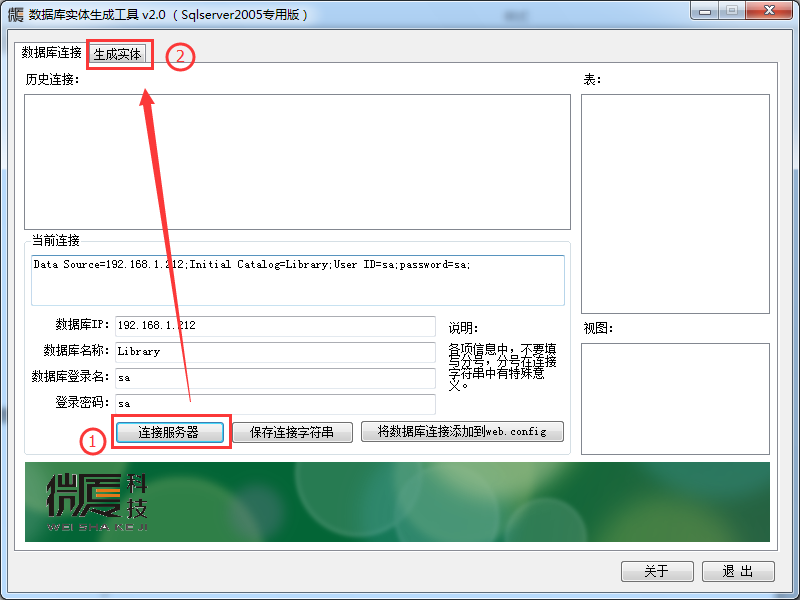


双击打开Weisha.Data.Generete.exe，填写数据库的IP地址、数据库名称、登录账号、密码，如下图：

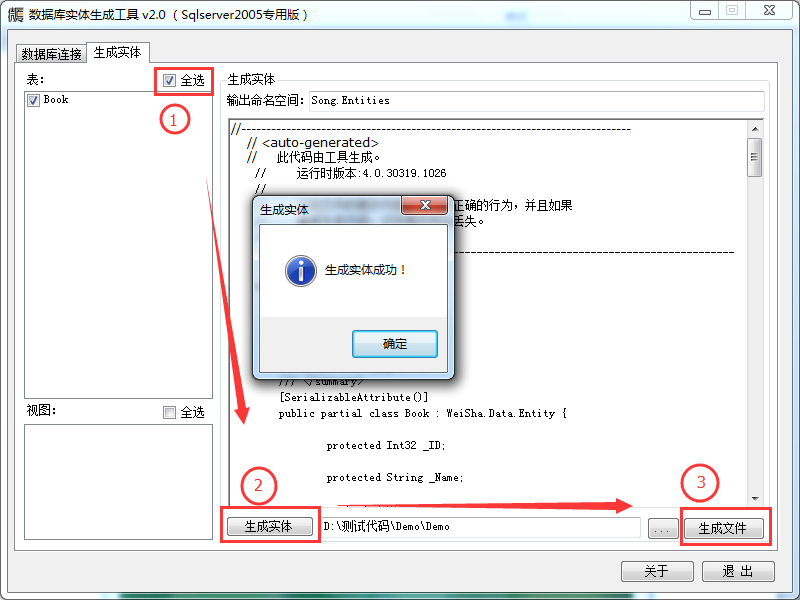


第三步时，点击“将数据库连接添加到web.config”时，会把数据库相关信息填写到web.config，它是Asp.net项目的配置文件。

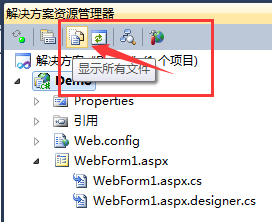
点击“连接服务器”，会在右侧显示当前数据库的所有表。切换到“生成实体”的操作界面，如下图所示：



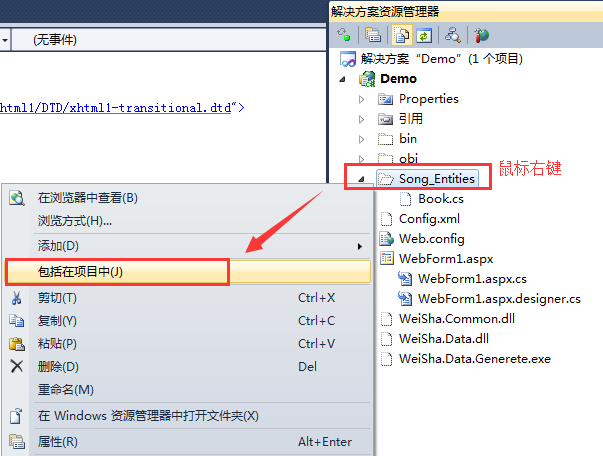
生成实体的代码，并生成类文件，如下图所示：



至此实体生成完成，但还不能使用。我们回到Visual Studio 2010操作界面，查找我们刚才生成的实体类的文件（文件后缀名为cs）。首先要点击“查看所有文件”，如下图：

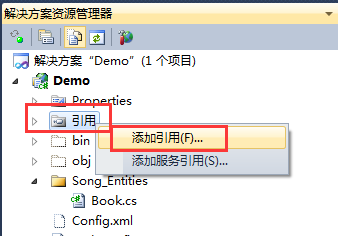


这个时候，我们就可以看到刚才生成的文件了，在上面点鼠标右键，把它们添加到项目，如下图所示：

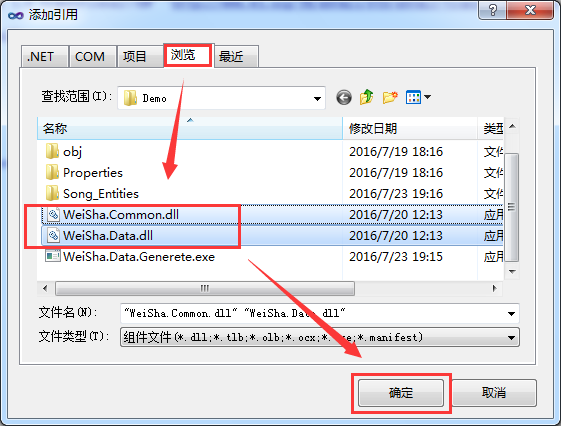


## 引用数据操作类

在进行数据库操作之前，需要引用ORM相关的dll文件，引用很简单，在项目中点击“引用”文件夹，选择“添加引用”，如下图：

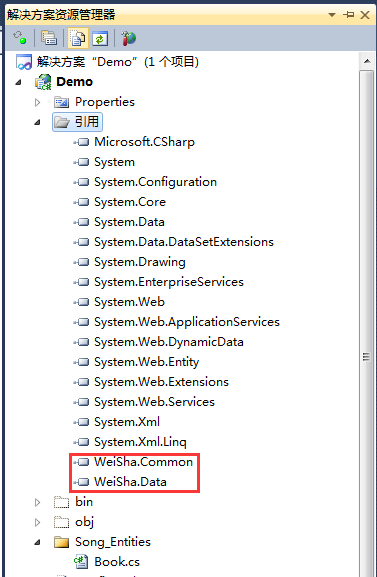


打开引用的操作界面，如下图：



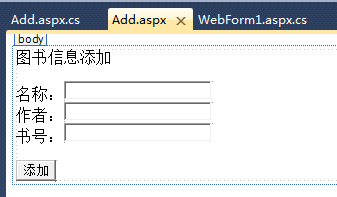
引用文件可以一次引用一个，也可以一次引用多个，如果要引用多个文件，可以按着ctrl键进行多文件的选择。

操作完成后我们可以在项目中看到它们，如下图：

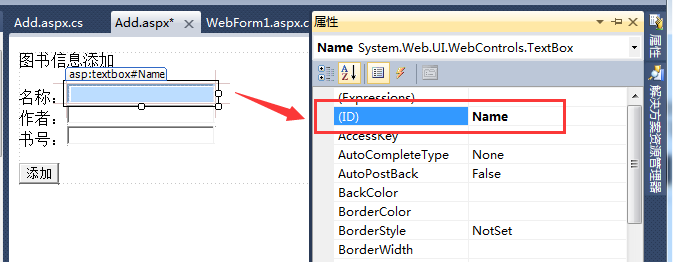


## 添加数据记录

在这里我们做一个添加图书信息的表单。首先我们创建一个Add.aspx的页面，双击打开，从左侧工具箱拖一些控件进来。如下图：



点中名称右面的输入框，给它的ID改名称，改成与数据库中Book表的字段一致（原因后会解释），如下图：

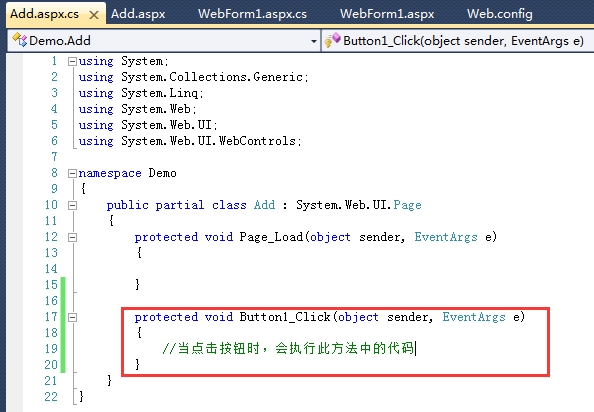


然后逐个输入框进行更改，如果嫌麻烦，可以进入html编辑界面，即点击下面的“源”，如图。

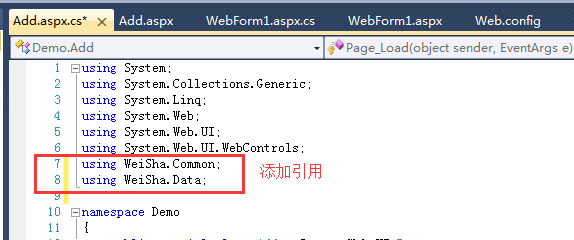
下图红框中即控件ID，全部改成与数据库名称对应的字段名，如下图：



然后回到“设计”界面，双击添加按钮，将会生成它的点击事件，如下图所示：



在写具体代码之前，请先引用WeiSha.Common与WeiSha.Data；如下图：



然后在按钮事件中写入如下代码：

protected void Button1\_Click(object sender, EventArgs e)

{

//创建书本的对象

Song.Entities.Book book = new Song.Entities.Book();

//将界面上填写的信息，填充到书本对象

book = WeiSha.Common.Data.EntityFill(this, book) as Song.Entities.Book;

//将填写的信息，保存到数据库

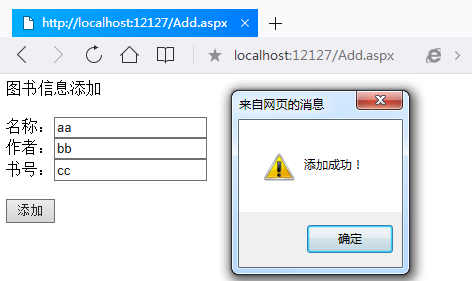
Gateway.Default.Save<Song.Entities.Book>(book);

//弹出提示信息

this.Alert("添加成功！");

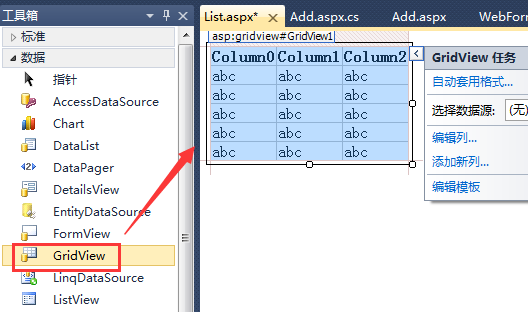
}

然后设置add.aspx为启始页，执行看一下效果。



## 查询数据记录

在项目中添加一个List.aspx的页面，并拖一个列表控件GridView上去，如下图：



在编辑区点击鼠标右键，选择“查看代码”项，进行代码编辑。

同样的，需要引用WeiSha.Common与WeiSha.Data；

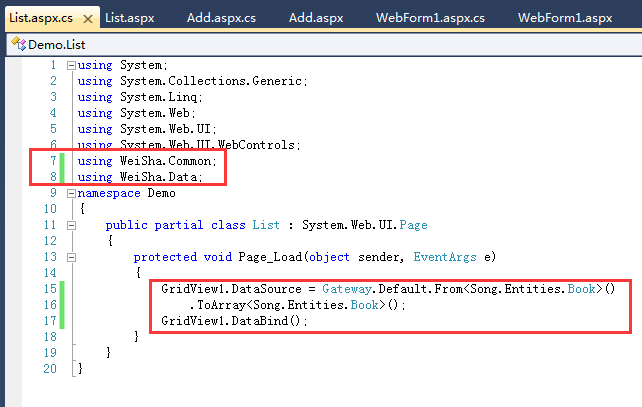
然后在Page\_load方法中录入以下代码：

GridView1.DataSource = Gateway.Default.From<Song.Entities.Book>()

.ToArray<Song.Entities.Book>();

GridView1.DataBind();

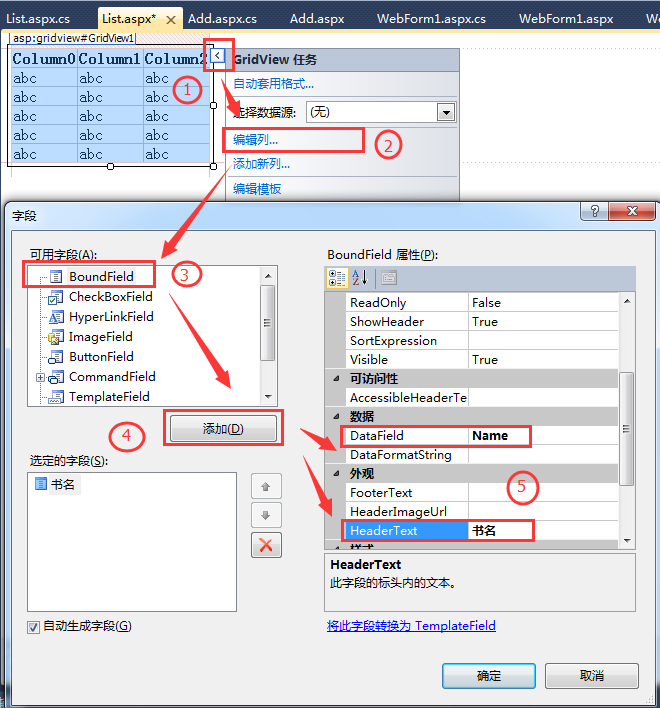
整体代码如下图所示：



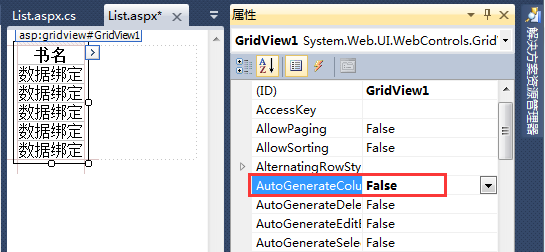
将List.aspx设为启始页，按F5键，执行一下看效果，如下图：



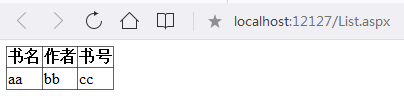
从上图可以看到，将才添加的信息已经列出来了，但是表头仍然是数据库中表的字段，如果要换成中文的名称，需要设置GridView的绑定项。



其它的表头以此类推。还要将GridView设置为不自动生成列，如下图：

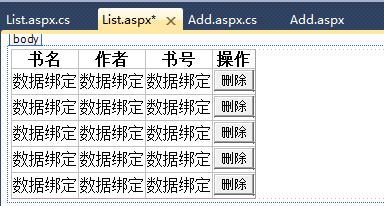


再次执行看一下效果。

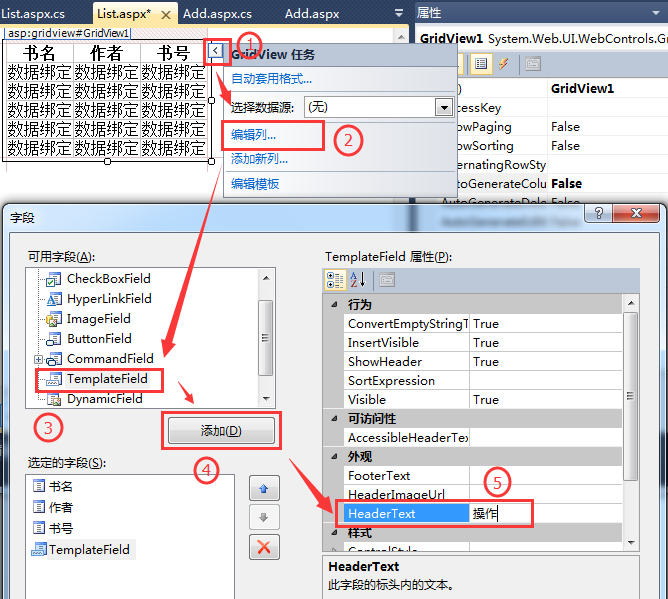


## 删除记录

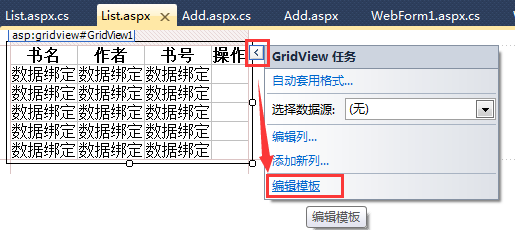
我们将在上面的Gridview增加一个删除按钮，实现如下的效果：



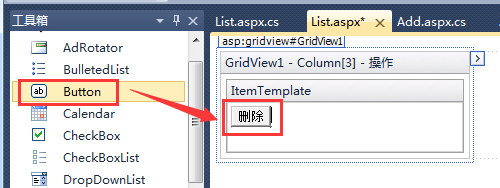
当点击删除按钮时，会删除当前行的记录。添加删除按钮的操作如下所示：



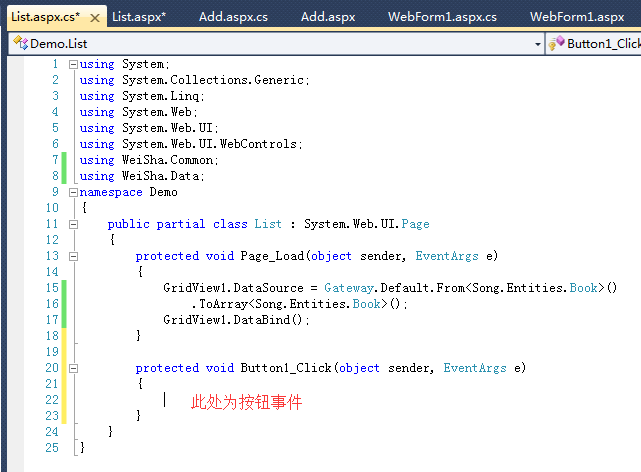
添加完模板列后，点击确定，回到编辑界面，选择编辑模板，如下所示：



从左侧的工具箱拖一个Button按钮到ItemTemplate区域，并设置名称。



然后双击此按钮，给它添加操作事件。双击后直接跳转到代码编辑中，如下图所示：



此时应该注意到，按钮列的每个按钮都用这一个事件，那么系统怎么知道它是要删除哪一行记录呢？所以我们要知系统，这个按钮所关联的记录是哪一条。

回到List.aspx的编辑中，找到这个按钮，给它增加一个CommandArgument属性，并把数据记录的ID传给它。CommandArgument='<%# Eval("ID") %>'

注意单引号与双引号的区别。



回到代码编辑界面，在按钮事件中写入如下代码：

protected void Button1\_Click(object sender, EventArgs e)

{

//sender即按钮，但是需要转换一下类型

Button btn = (Button)sender;

//获取数据记录的id

string id = btn.CommandArgument;

//删除记录，通过数据记录的id

//Song.Entities.Book.\_.ID == id 是删除条件

Gateway.Default.Delete<Song.Entities.Book>(Song.Entities.Book.\_.ID == id);

this.Alert("成功删除!");

//重新绑定数据列

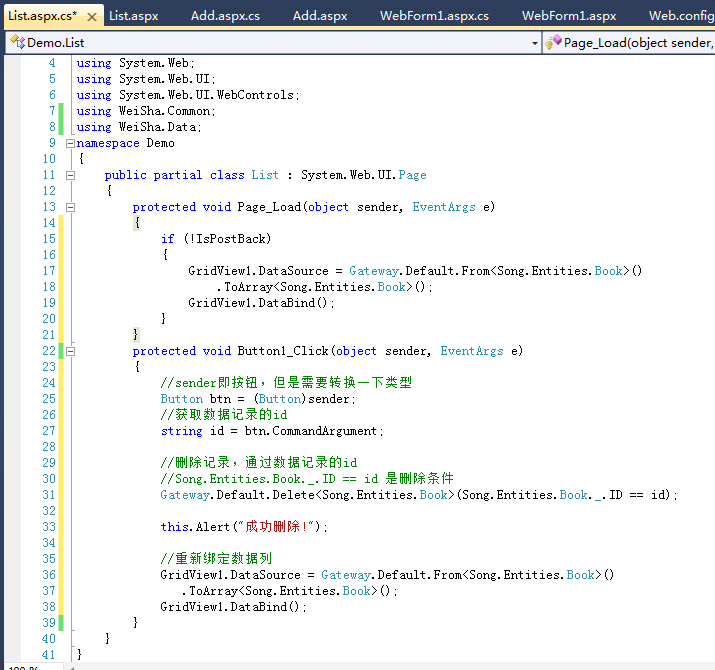
GridView1.DataSource = Gateway.Default.From<Song.Entities.Book>()

.ToArray<Song.Entities.Book>();

GridView1.DataBind();

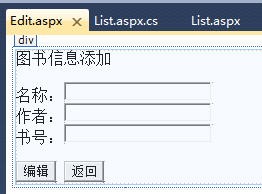
}

完整的代码截图如下：

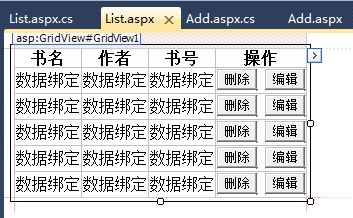


## 编辑信息

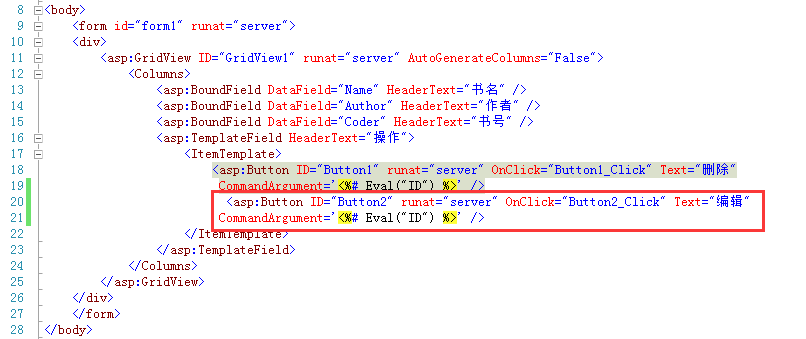
创建一个Edit.aspx页面，并添加控件等，有些类似于添加记录的界面，如下图：



我们用添加删除按钮的方法，再创建一个编辑按钮，如下图：



同样的，给它添加CommandArgument属性与按钮事件，如下图：



回到代码编辑，仍然要先引WeiSha.Common与WeiSha.Data，然后要获取传递过来的id值，如下代码：

//获取传递过来的Id，如果不存在，则为0

private int id = WeiSha.Common.Request.QueryString["id"].Int32 ?? 0;

??的意思是，如果获取不到，或为Null时，默认返回0。

在Page\_load事件中，加载当前对象到显示界面，为了实现第一次打开时加载，这里采用了IsPostBack属生的判断。如下图：

//获取传递过来的Id，如果不存在，则为0

private int id = WeiSha.Common.Request.QueryString["id"].Int32 ?? 0;

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

//获取当前书本的对象

Song.Entities.Book book = Gateway.Default.From<Song.Entities.Book>()

.Where(Song.Entities.Book.\_.ID == id).ToFirst<Song.Entities.Book>();

//填充到界面

WeiSha.Common.Data.EntityBind(this, book);

}

}

在编辑按钮的事件中，将用户编辑后的数据重新保存到数据库库，代码如下：

/// <summary>

/// 编辑按钮的事件

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

protected void btnEdit\_Click(object sender, EventArgs e)

{

//获取当前书本的对象

Song.Entities.Book book = Gateway.Default.From<Song.Entities.Book>()

.Where(Song.Entities.Book.\_.ID == id).ToFirst<Song.Entities.Book>();

//将界面上填写的信息，填充到书本对象

book = WeiSha.Common.Data.EntityFill(this, book) as Song.Entities.Book;

//将对象保存到数据库

Gateway.Default.Save<Song.Entities.Book>(book);

//返回列表页

btnBack\_Click(null, null);

}

protected void btnBack\_Click(object sender, EventArgs e)

{

Response.Redirect("list.aspx");

}

完整的代码截图如下（红框中为关键代码）：



## 小结

我们通过上述示例实现了一个简单的书籍管理的应用，能够完成基本的增删改查，在这里我们没有用到ADO.net相关代码，而是用了ORM实现数据库的操作，大大减化了代码量。下面我梳理一下流程。

1. 首先要生成ORM实体

把WeiSha.Data.Generete.exe与Weisha.Data.dll、Weisha.Common.dll，三个文件复制到项目文件夹下。生成实体，并把数据库链接保存到web.config中。

1. 引用Weisha.Data.dll、Weisha.Common.dll

把这个两个dll文件，引用到项目。

1. 控件名称与数据库字段一一对应

在添加、修改记录时，界面上的控件要与数据库字段一一保持对应，这样就可以很简便的实现添加与修改操作。

# ORM编写说明

## 获取单条记录

//获取条个记录,ToFirst

Song.Entities.Book book = Gateway.Default.From<Song.Entities.Book>()

.ToFirst<Song.Entities.Book>();

## 保存记录

保存用Save方法，不管理新增还是修改，都是用它

//将对象保存到数据库

Gateway.Default.Save<Song.Entities.Book>(book);

## 删除记录

//删除记录，通过数据记录的id

//Song.Entities.Book.\_.ID == id 是删除条件

Gateway.Default.Delete<Song.Entities.Book>(Song.Entities.Book.\_.ID == id);

## 获取多条记录

//获取数据记录的集合

Song.Entities.Book[] books = Gateway.Default.From<Song.Entities.Book>()

.ToArray<Song.Entities.Book>();

## 分页查询

//分页获取

int size = 10; //每页显示的条数

int index = 1; //页码，

Song.Entities.Book[] books = Gateway.Default.From<Song.Entities.Book>()

.ToArray<Song.Entities.Book>(size, (index - 1) \* size);

## 条件查询

//按条件获取

Song.Entities.Book[] books = Gateway.Default.From<Song.Entities.Book>()

.Where(Song.Entities.Book.\_.Author == "作者名称")

.ToArray<Song.Entities.Book>();

## 排序

//排序

Song.Entities.Book[] books = Gateway.Default.From<Song.Entities.Book>()

.Where(Song.Entities.Book.\_.Author == "作者名称")

.OrderBy(Song.Entities.Book.\_.ID.Desc)

.ToArray<Song.Entities.Book>();

## 平均值，求和，汇总

//求和

int sum = 0; int.TryParse(Gateway.Default.Sum<Song.Entities.Book>(Song.Entities.Book.\_.ID, null).ToString(),out sum);

//求平均值

int avg = 0; int.TryParse(Gateway.Default.Avg<Song.Entities.Book>(Song.Entities.Book.\_.ID, null).ToString(),out avg);

//求记录数

int count = Gateway.Default.Count<Song.Entities.Book>(null);

//取最大值，此为取最大的id

int max = 0; int.TryParse(Gateway.Default.Max<Song.Entities.Book>(Song.Entities.Book.\_.ID, null).ToString(),out max);

//取最小值

int min = 0;

int.TryParse(Gateway.Default.Min<Song.Entities.Book>(Song.Entities.Book.\_.ID, null).ToString(),out min);