# Asp.net概述

## 什么是web应用

Web应用就是网站，网站就是程序，这个程序是B/S结构的

## 什么是B/S结构

B/S就是C/S。

客户端特指浏览器，服务器特指web服务器。无论什么客户端、服务器都是软件。

## 什么是asp.net

Asp.net 是动态网页开发技术，在服务器端运行的是.Net代码，动态生成HTML，然后响应给浏览器。

在浏览器中，可以使用JavaScript、Dom完成很多工作，但是有许多工作无法在浏览器端完成，如读取数据库，验证登陆。这就要求能实现在服务器进行数据读取、数据验证工作的技术，asp.net就是能完成这项工作的技术。

## 请求处理响应模型

当输入一个网址，按回车发送时就会发出http请求，服务器在收到请求后，读取数据，进行处理，把响应返回给客户端。这就是请求处理响应模型。

## Asp.net处理程序分类

Asp.net 处理程序有三种：

1. 一般处理程序（ashx）。刀耕火种阶段，但是最体现原理
2. Webform（aspx）：机械化时代，有快速开发的form
3. MVC（model，view，controller）：工业化时代，分工明确。

一般处理程序

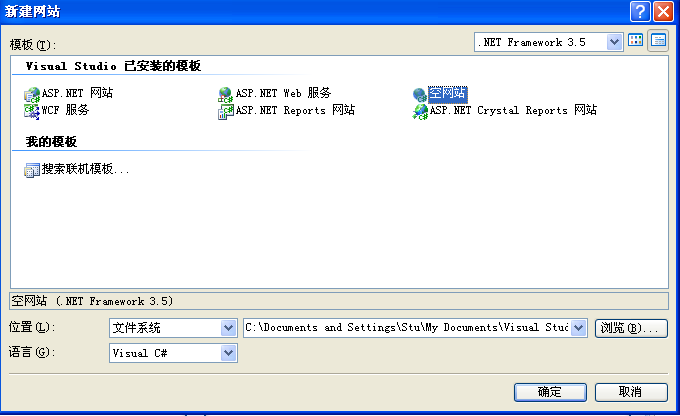
# 一般处理程序

一般处理程序就是实现了IHttpHandler接口的类，一般处理程序负责处理它所对应的一个或一组URL地址的访问请求。它接收客户端发出的访问请求信息（请求报文）并产生响应内容（响应报文）。

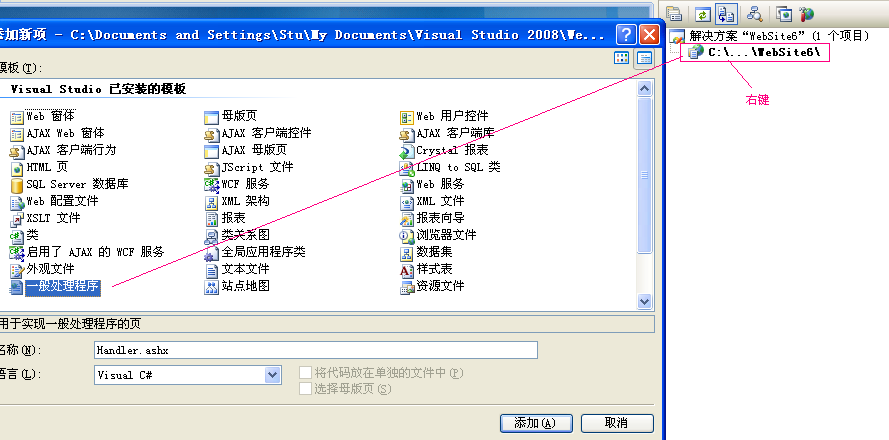
## 例1：向浏览器输出服务器当前的时间

知识点：向浏览器写入数据

1. 首先创建一个空的asp.net应用程序



1. 在项目上右击，选择添加新项，添加一般处理程序



1. 编写代码，输出服务器时间

|  |
| --- |
| <%@ WebHandler Language="C#" Class="Handler" %>  using System;  using System.Web;  public class Handler : IHttpHandler {    public void ProcessRequest (HttpContext context) {  context.Response.ContentType = "text/plain";  //向浏览器输出服务器当前时间  context.Response.Write(DateTime.Now);  }    public bool IsReusable {  get {  return false;  }  }  } |

## 例2：输出一段带图片的html页面代码

知识点：ContentType。

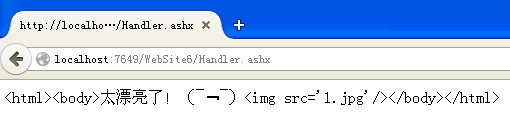
1. 项目根目录添加一个图片。（复制图片在根目录粘贴，或者直接拖拽）



1. 编写代码

|  |
| --- |
| <%@ WebHandler Language="C#" Class="Handler" %>  using System;  using System.Web;  public class Handler : IHttpHandler {    public void ProcessRequest (HttpContext context) {  context.Response.ContentType = "text/plain";  //向浏览器输出一段html代码  context.Response.Write("<html><body>太漂亮了！（¯﹃¯）<img src='1.jpg'/></body></html>");  }    public bool IsReusable {  get {  return false;  }  }  } |

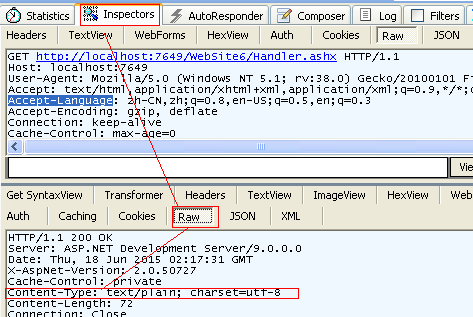
1. 用Firefox等标准浏览器打开，发现输出的是字符串：



1. 这是因为在代码中我们定义的ContentType为“text/plain”

|  |
| --- |
| context.Response.ContentType = "text/plain"; |

在fiddler中查看下响应文件，会发现context.Response.ContentType中定义的类型会作为响应的结果返还给客户端。



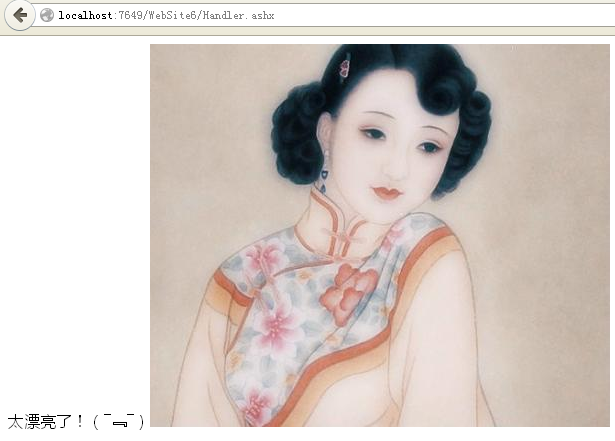
客户端接受到文本类型的内容时，会把它当做一般文本处理（无格式文本），也就是显示为字符串样式。

|  |
| --- |
| 知识加油站 |
| Content-Type：内容类型，通常被格式化为类型/子类型。一般是指网页中存在的Content-Type，用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件。 |

1. 修改Content-Type为“text/html”：

|  |
| --- |
| context.Response.ContentType = "text/html"; |

1. 显示效果如下：



## 一般处理程序的功能

1. 获取客户端通过HTML的Form表格提交的数据和URL参数
2. 创建对客户端的响应信息内容
3. 访问服务器端的文件系统
4. 连接数据库并开发基于数据库的应用
5. 调用其它类

# Http发送请求数据的方式

## get

GET方法是默认的HTTP请求方法，提交的数据将作为URL的一部分向Web服务器发送。

因此，如果使用GET方法来提交表单数据就存在着安全隐患上。例如

Http://127.0.0.1/login.jsp?Name=zhangshi&Age=30&Submit=%cc%E+%BD%BB

从上面的URL请求中，很容易就可以辩认出表单提交的内容。

另外由于GET方法提交的数据是作为URL请求的一部分所以提交的数据量不能太大。

### 案例1：手动提交请求数据

1. 新建一个html网页，用一个a标签来进行get请求。

|  |
| --- |
| <!DOCTYPE html>  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <title></title>  </head>  <body>  <!--?后面既是以键值对形式传递的数据-->  <a href="Handler.ashx?pwd=hyf123&name=hyf">请求</a>  </body>  </html> |

1. 在Handler.ashx进行数据的接收与响应

http请求中的数据存在NameValueCollection集合中。通过QueryString可以进行数据读取。

|  |
| --- |
| //读取请求数据：http请求中的数据存在NameValueCollection集合中,QueryString对集合中的数据进行索引。  string pwd = context.Request.QueryString["pwd"];  string name = context.Request.QueryString["name"]; |

|  |
| --- |
| 知识加油站： |
| NameValueCollection集合比较特殊，它的键可以重复，进行索引时，值之间会以逗号隔开。 |
| <%@ WebHandler Language="C#" Class="NV" %>  using System;  using System.Web;  using System.Collections.Specialized;  public class NV : IHttpHandler {    public void ProcessRequest (HttpContext context) {  context.Response.ContentType = "text/plain";  NameValueCollection nmc = new NameValueCollection();  nmc.Add("qsh", "焚书坑儒");  nmc.Add("qsh", "统一度量衡");  nmc.Add("qsh", "书同文、车同轨");  context.Response.Write(nmc["qsh"]);  }    public bool IsReusable {  get {  return false;  }  }  } |
| 输出结果： |
|  |
| 问题 |
| <a href="Handler.ashx?pwd=hyf123&name=hyf&name=qsh">请求</a>  一般处理程序获取name的值后输出结果是什么？ |

1. 响应

|  |
| --- |
| context.Response.Write("用户名：" + name + "<br/>");  context.Response.Write("密码:" + pwd); |

1. 完整代码

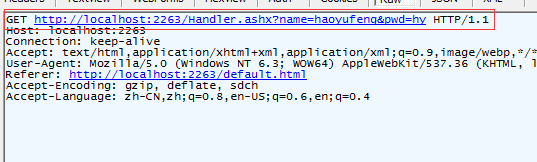
|  |
| --- |
| <%@ WebHandler Language="C#" Class="Handler" %>  using System;  using System.Web;  public class Handler : IHttpHandler  {  public void ProcessRequest(HttpContext context)  {  context.Response.ContentType = "text/html";  //读取请求数据：http请求中的数据存在NameValueCollection集合中,QueryString对集合中的数据进行索引。  string pwd = context.Request.QueryString["pwd"];  string name = context.Request.QueryString["name"];  context.Response.Write("用户名：" + name + "<br/>");  context.Response.Write("密码:" + pwd);  }  public bool IsReusable  {  get  {  return false;  }  }  } |

### 案例2：通过表单提交请求的数据

1. 在html中添加表单

|  |
| --- |
| <form action="Handler.ashx">  <!--在传递时name属性的值作为键，用户输入作为值。-->  用户名：<input type="text" name="name" />  密码： <input type="text" name="pwd" />  <input type="submit" /> <!--向action中定义的程序中提交数据-->  </form> |

1. 点击提交按钮时，自动把键值对附加到请求的URL上



### 案例3：通过javascript提交请求数据

1. 创建html代码

|  |
| --- |
| 用户名： <input type="text" id="uid" />  密码：<input type="text" id="password" />  <input type="button" id="btn" value="submit" /> |

1. 在head标签中添加javascript代码

|  |
| --- |
| <script type="text/javascript">  //窗口加载时，为按钮添加onclick事件，触  window.onload = function () {  document.getElementById("btn").onclick = function () {  var t = document.getElementById("uid").value;  var p = document.getElementById("password").value;  //javascript中定位到另一个地址的语句  window.location.href = "Handler.ashx?name=" + t + "&pwd=" + p;  }  }  </script> |

1. 点击按钮即可把值提交给处理程序。

|  |
| --- |
| 知识加油站： |
| 标签的name属性：提交数据时作为键使用。  标签的id属性： 是为了让javascript获取元素，从而调用元素中的内容时。 |

## post

通过post提交数据时，数据存在http报文体中，而get是在URL中。

### 案例：post请求提交数据

post只能通过form提交

1. 在html中进行设置，form的method设置为post

|  |
| --- |
| <!--method设置为post即可post提交-->  <form action="postHandler.ashx" method="post">  <input type="text" name="txt" />  <input type="submit" />  </form> |

1. 在处理程序中获取数据并处理。获取post请求需要用context.Request.Form进行索引

|  |
| --- |
| public void ProcessRequest(HttpContext context)  {  context.Response.ContentType = "text/html";  string txt = context.Request.Form["txt"];  context.Response.Write("提交的数据：" + txt);  } |

1. 在浏览器中打开html网页，输入数据后提交，查看fiddler:



## post与get区别

### 相对安全

POST相比与GET来说是相对安全，所谓的相对安全其实POST和GET都是明文传输的，通过抓包还是能看到里面的数据。相对安全是GET只要通过URL就可以看到传输的信息，而POST里面参数数据是存放在报文体中的。且GET的数据可能被浏览器缓存，他人在看浏览器记录的时候就可能可以获取你的查询的信息和条件。而POST数据不会被缓存，就不会有这种情况出现。

### URL限制

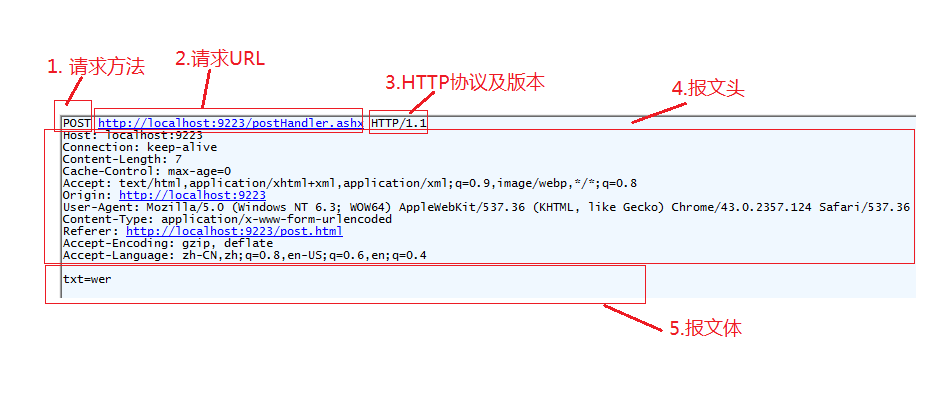
GET的URL长度是有做限制的，标准的URL不存在参数上的限制问题，因此HTTP协议规范也没有对URL长度进行限制，该限制取决于浏览器。

### 表单数据限制

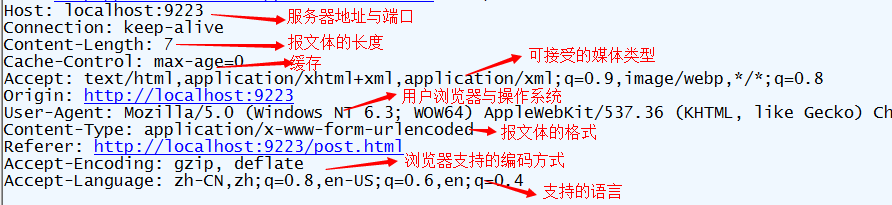
GET要求一定要为ASCII 类型数据。POST没有任何要求

## 请求报文

HTTP请求报文可以分成5部分：

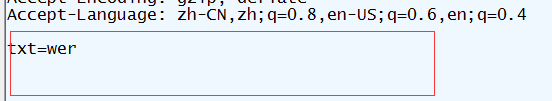


### 报文头



### 报文体

get请报文体没有数据，而post请求的数据在报文体中：



# HttpContext

### HttpContext

在处理请求的各个阶段中，会有一个对象在各个对象之间进行传递，这就需要保存请求的上下文信息，这个对象就是HttpContext对象。HttpContext封装了ASP.NET要处理的单次请求的所有信息。

HttpContext的生命周期从服务器接收的HTTP请求开始到响应发送回客户端结束。

### Request

获取http请求提交的数据。有三个常用的索引QueryString、Form、Request。

QueryString用来获取获取包含在URL中的一些参数，Form用来获取用户输入的表单数据。而Request即可以获取get方式提交的数据也可以获取post方式提交的数据，还能获取cookie中的数据。

|  |
| --- |
| string name = context.Request["name"];//可以同时获取get和post请求。  string name1 = context.Request.Form["name"];  string name2 = context.Request.QueryString["name"]; |

### Response

获取当前http响应的HttpResponse对象，该对象中封装了http响应的所有信息。

|  |
| --- |
| context.Response.Write("Hello World"); |

上面的代码中调用HttpResponse对象中的write方法，向http响应的输出流中写入字符串。

### 一般处理程序代码注释

|  |
| --- |
| <%@ WebHandler Language="C#" Class="Handler" %>//头文件  using System;  using System.Web;  /// <summary>  /// 一般处理程序：实现了IHttpHandler接口，能够处理http请求，做出http响应。  /// </summary>  public class Handler : IHttpHandler  {  /// <summary>  /// 接受请求，处理请求，做出响应  /// </summary>  /// <param name="context">请求上下文对象，保存请求的上下文信息，在各个对象间进行传递</param>  public void ProcessRequest(HttpContext context)  { //ContentType属性：获取或设置响应输出流中的媒体类型（MIME type）。  context.Response.ContentType = "text/plain";  //Write方法向响应输出流中输出字符串  context.Response.Write("Hello World");  }  /// <summary>  /// 这次请求是否可以重用  /// </summary>  public bool IsReusable  {  get  {  return false;  }  }  } |

### Params

parames可以获得所有的请求

下面的代码会在网页中输出提交的所有请求：

|  |
| --- |
| public void ProcessRequest(HttpContext context)  {  context.Response.ContentType = "text/html";  for (int i = 0; i < context.Request.Params.Keys.Count; i++)  {  //获取键  string key = context.Request.Params.Keys[i];  //键对应的值  string value = context.Request.Params[context.Request.Params.Keys[i]];  //写入键以及值  context.Response.Write(key + ":" + value + "</br>");  }  } |

# 登录程序

## 实现思路

1. 创建一个html页面，为登录界面。form表单中数据用post方式发送。其中有一个隐藏元素用来判断是否是post请求。

|  |
| --- |
| <!DOCTYPE html>  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <title></title>  </head>  <body>  <form action="LoginHandler.ashx" method="post">  <!--隐藏元素，用来判断是否是post请求-->  <input type="hidden" name="\_viewState" value="a" />  <!--table中为登录界面-->  <table>  <tr>  <td>用户名：</td>  <td>  <!--@uid为占位符-->  <input type="text" name="uid" value="@uid" />  </td>  </tr>  <tr>  <td>密码：</td>  <td>  <input type="text" name="pwd" />  </td>  </tr>  <tr>  <td colspan="2" style="text-align:right"><input type="submit" /></td>  </tr>  </table>  </form>  <!--判断结果通过占位符写入，不会改变html文档的结构-->  <p>@loginResult</p>  </body>  </html> |

1. 创建一般处理程序，读取请求过来的数据，通过hidden元素中的值判断是否是post过来的。，如果post过来则判断是否登录成功，如果不是post过来，则表示是第一次登录，直接写入模板网页，并设置占位符所在位置为空。

|  |
| --- |
| <%@ WebHandler Language="C#" Class="LoginHandler" %>  using System;  using System.Web;  public class LoginHandler : IHttpHandler  {  public void ProcessRequest(HttpContext context)  {  context.Response.ContentType = "text/html";  //读取form表单中的数据  string uid = context.Request.Form["uid"];  string pwd = context.Request.Form["pwd"];  string vs = context.Request.Form["\_viewState"];  //判断隐藏元素\_viewState是否为空，如果为空，则是get请求，也就是说页面是第一次加载，不是通过点击submit跳转过来的。  bool postback = !string.IsNullOrEmpty(vs);  //想对地址转换为绝地地址。  string tempPath = context.Request.MapPath("\_Login.html");  //读取\_Login.html作为模板。模板网页一般以下划线开头。  string tempHtml = System.IO.File.ReadAllText(tempPath);  //如果是post过来的，也就是通过submit按钮请求的。则判断是否登录成功。  if (postback)  {  if (uid == "admin" && pwd == "888")  {  context.Response.Write("登录成功");  }  else  {  //通过占位符替换字符串中的内容。  tempHtml = tempHtml.Replace("@uid", uid).Replace("@loginResult", "登录失败");  context.Response.Write(tempHtml);  }  }  //如果不是post过来的，则是第一次加载。不判断登录，直接写入模板。  else  {  tempHtml = tempHtml.Replace("@uid", "");  tempHtml = tempHtml.Replace("@loginResult", "");  context.Response.Write(tempHtml);  }  }  public bool IsReusable  {  get  {  return false;  }  }  } |

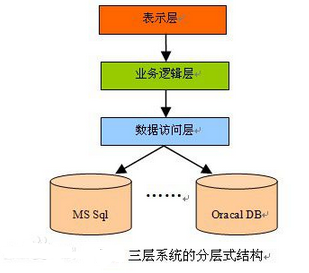
1. 这样就实现了登录效果。

# 搭建三层

三层架构(3-tier architecture) 通常意义上的三层架构就是将整个业务应用划分为：

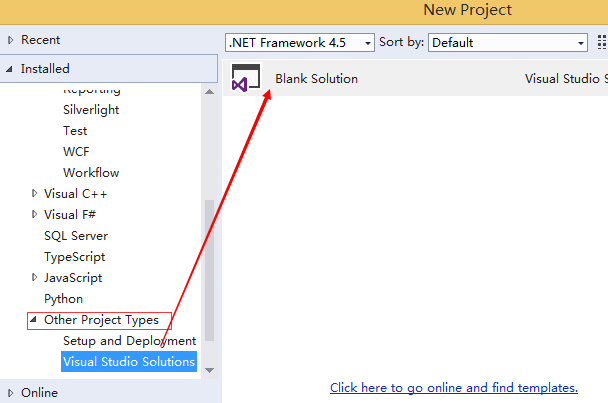
* 数据访问层（Data access layer）：主要是对原始数据（数据库或者文本文件等存放数据的形式）的操作层，而不是指原始数据，也就是说，是对数据的操作，而不是数据库，具体为业务逻辑层或表示层提供数据服务．
* 业务逻辑层（Business Logic Layer）：主要是针对具体的问题的操作，也可以理解成对数据层的操作，对数据业务逻辑处理，如果说数据层是积木，那逻辑层就是对这些积木的搭建。
* 表现层（Presentation layer）：主要表示WEB方式，也可以表示成WINFORM方式，WEB方式也可以表现成：aspx，如果逻辑层相当强大和完善，无论表现层如何定义和更改，逻辑层都能完善地提供服务。

区分层次的目的即为了“高内聚低耦合”的思想。在软件体系架构设计中，分层式结构是最常见，也是最重要的一种结构。

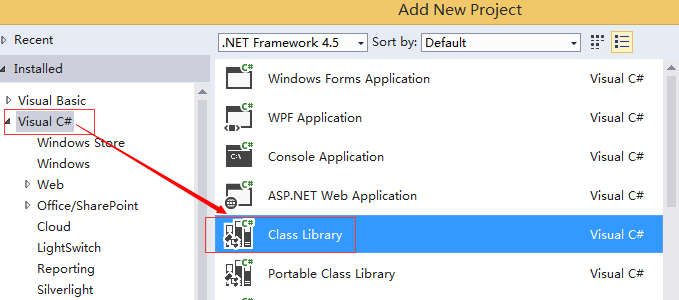


## 创建三层架构

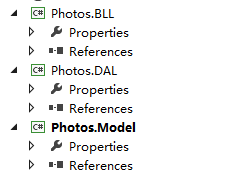
1. 新建空解决方案



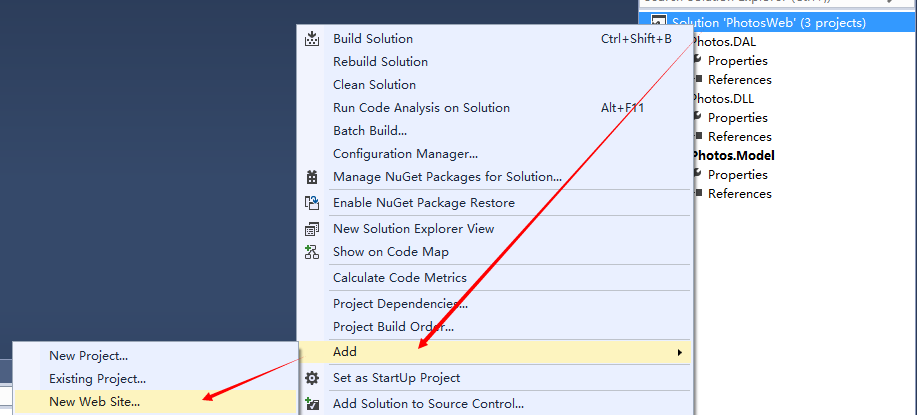
1. 新建三个类库项目



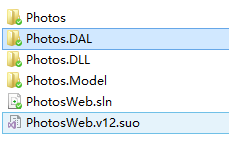
1. 这三个类库分别为：Model、DAL、BLL层。自带的class1.cs都删除掉。



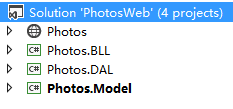
1. 新建一个website作为UI层。



1. 文件夹架构如下：

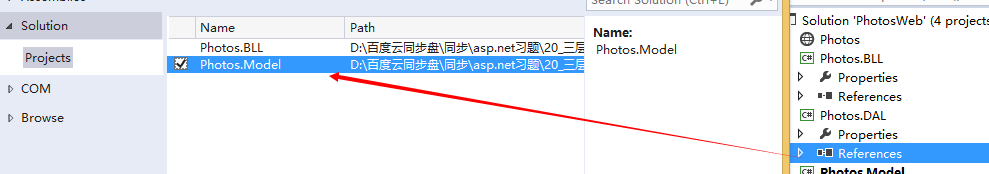


1. 项目架构如下：



## 添加引用

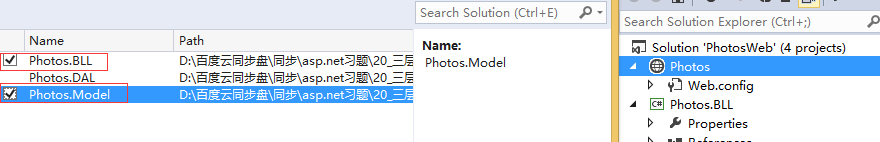
1. DAL引用Model



1. BLL引用DAL和Model

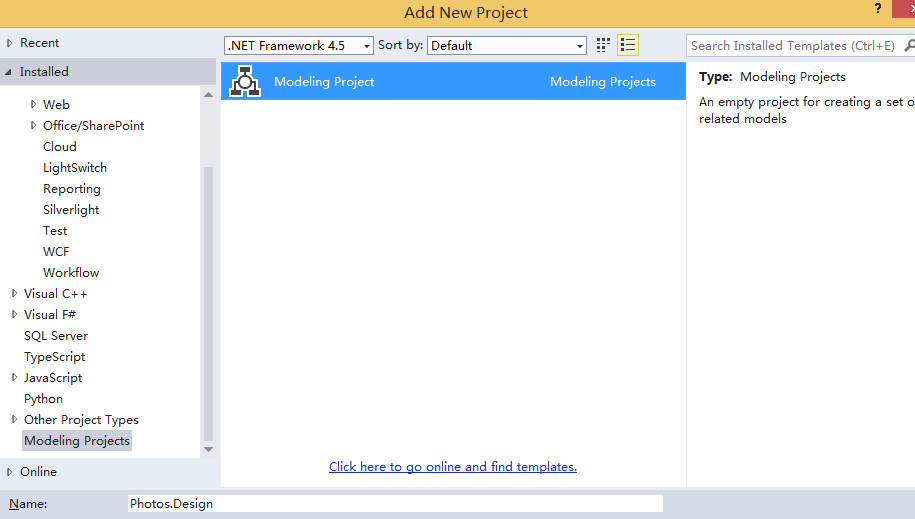


1. UI引用BLL和Model

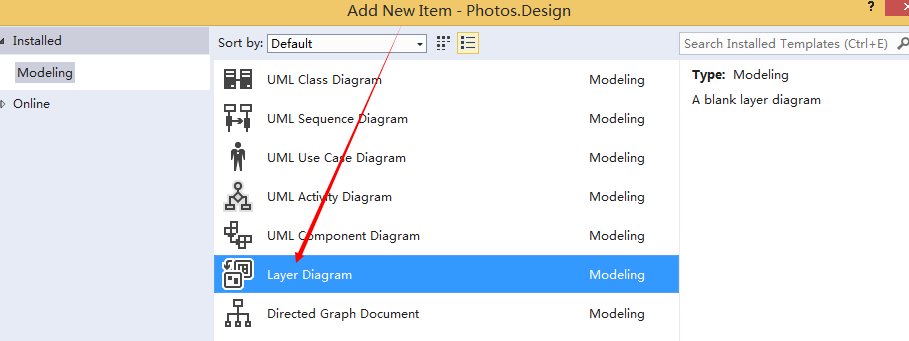


## 设计建模

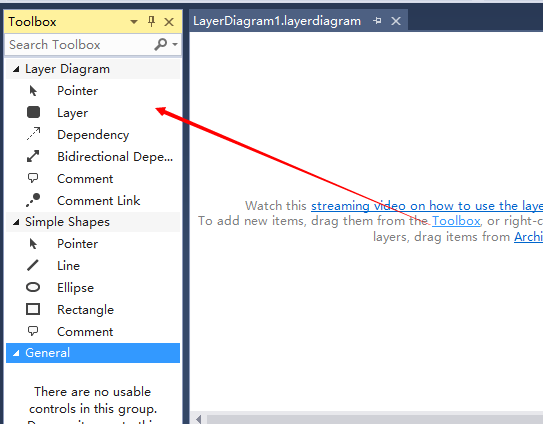
1. 添加建模项目



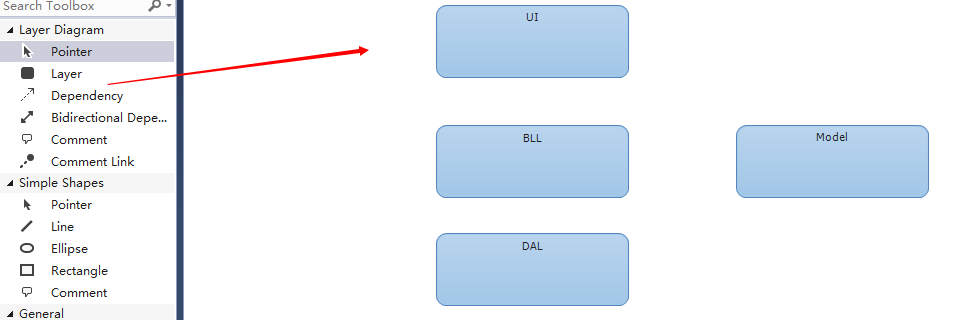
1. 添加一个层图



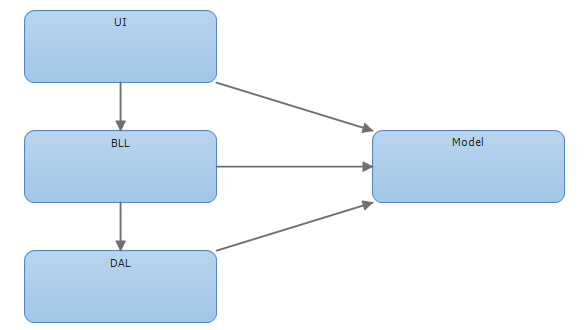
1. 显示工具箱



1. 拖到层到面板。总共添加4个层，双击命名



1. 添加依赖关系（Dependency）。



1. UI依赖于BLL和Model；BLL依赖于DAL和Model；DAL依赖于Model。