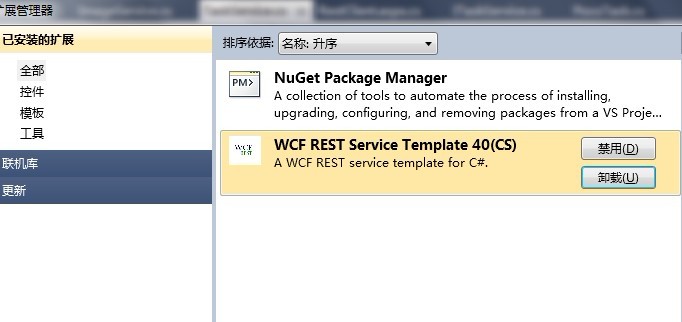
**[WCF REST 工作总结（一）](http://www.cnblogs.com/wlflovenet/archive/2011/10/29/WCFREST.html)**

最近工作中学习使用了WCF REST，REST 有很多好处 高效 简约  面向资源  而客户端调用 也变得非常简单。REST 入门的资料等 大家可以去网上找 这里主要分享下遇到的问题以及解决~

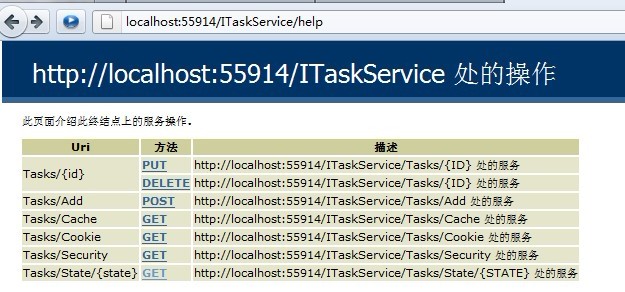
**一.环境准备**

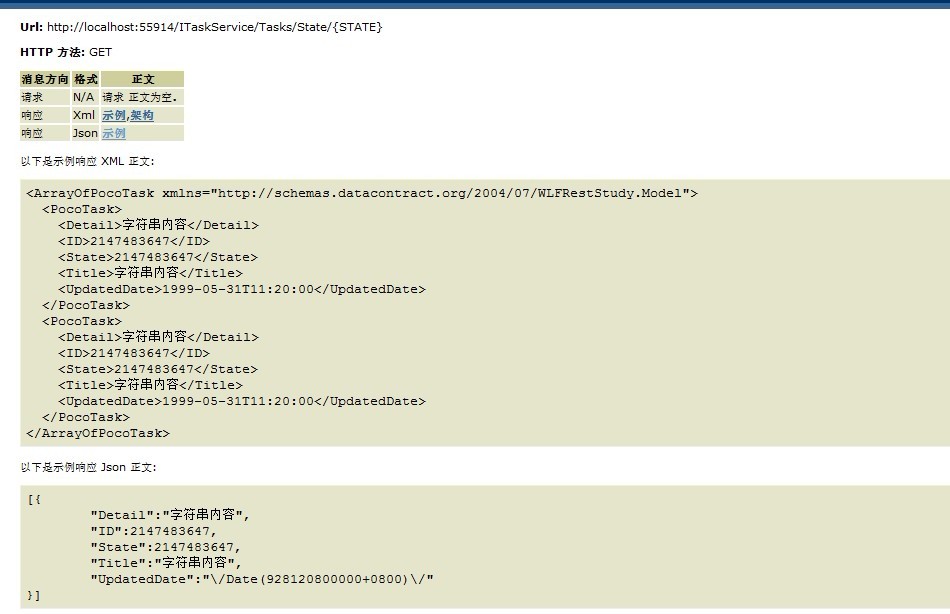
使用WCF REST 可以先下载一个模版 并大致预览



下载好这个模版后  新建项目 选择这个模版 会默认帮我们生成一个 CRUD 的代码  新手朋友可以用这个快速熟悉下REST风格的WCF.

这这里还给我们提供了一个很方便的功能 help  只需在我们的服务后面 加上help 就能显示服务的详情与介绍了





**二.搭建REST WCF服务**

1.接口的实现

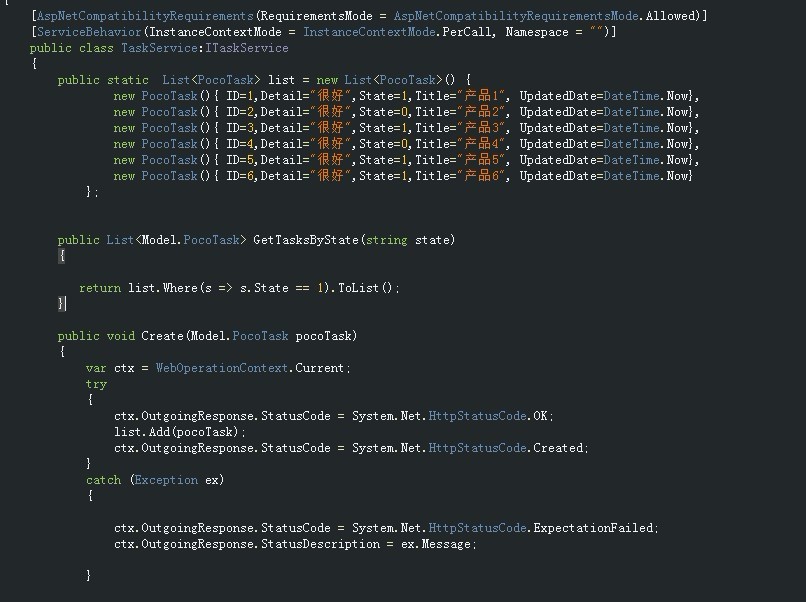


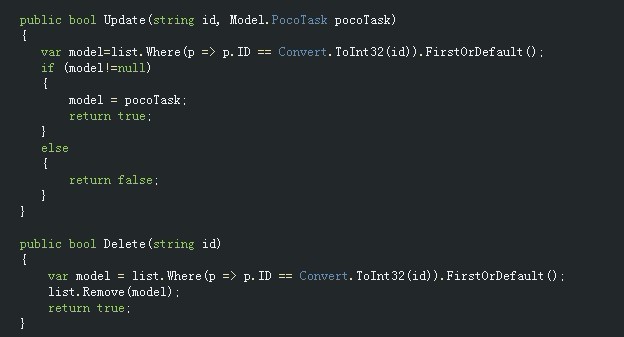
当我们的请求为 Get时  一般用来获取数据时  使用[WebGet]  而 增加 修改 删除 则使用[WebInvoke]  里面的Method 来决定使用POST 或PUT等 这里要注意下

Method 里的一定要为大写  POST PUT等  否则会出现 *HTTP* *错误* *405* *405* 不允许此方法 对于请求所标识的资源，不允许使用请求行中所指定的方法。

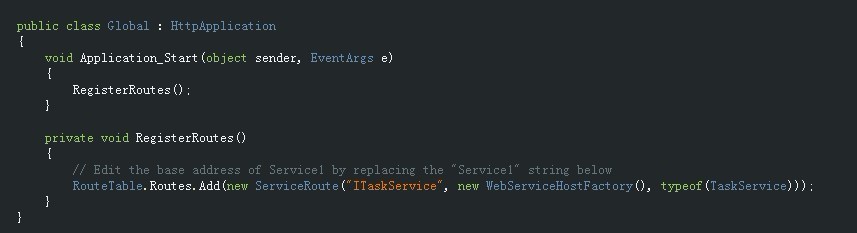
我们在这里 还可以通过 **ResponseFormat** 与 **RequestFormat** 来指定接受的参数与返回的参数格式

2.类继承接口 这里先贴一部分 最基础的CRUD





**3. Global 里注册路由**



用过mvc的朋友  是不是发现这样写完后 非常的像mvc呢~

**三.实现调用服务**

  1.实现get 服务

   很简单  我们可以直接通过浏览器 输入 http://localhost:55914/ITaskService/Tasks/State/1    即可获得state为1的所有数据  以json格式传输过来

**rest 风格的wcf  使得我们不用再像以前一样 客户端 添加 服务引用了**  **没有了wsdl  简洁了许多 效率也提高了很多  使得我们访问更容易**

  REST 由于没有了代理   而是发送请求   我们的客户端可以是 js 访问    如果是 jquery  只需$.get  即可发送请求   这里我主要说下C#的访问方式

  通过[WebClient](http://msdn.microsoft.com/zh-cn/library/system.net.webclient%28VS.80%29.aspx) 方式  这个类封装的比较好  调用起来简单  但由于封装的过多 实现一些功能时则不太容易

  通过[WebRequest](http://msdn.microsoft.com/zh-cn/library/system.net.webrequest%28VS.80%29.aspx)方式 这个则比较灵活

  上面两个是经常用到的两个类 这里再推荐给大家一个 是wcf工具里的其中一个 专门用来访问 rest 服务 也非常的简单

  通过[HttpClient](http://msdn.microsoft.com/en-us/library/system.net.http.httpclient%28v=VS.110%29.aspx) 这个也是微软开发的工具类  而且在最新的 .net 4.5里 集成了进去

  通过上面三个任意其中一个就可以实现提交了

  但是 这里有一个问题 我们以前有代理类  通过调用方法 得到代理类 可是现在呢  我们没有类  我们得到的是一串json 格式的数据 我们怎么把他变成像类一样的 方便操作呢？

  比如 A公司发布了rest wcf服务 数据格式 json    公司B 要调用A公司的服务   以前ws 或wcf都有代理  可以生成代理类  现在却得到的是json数据 而我们还没有要反序列化的实体类

  比如 我现在想把得到的数据 像表格一样的展示出来

  有两种方法

**1.对照json数据  客户端新建个类 然后通过反序列化  把json 反序列化为类对象**

   这样比较麻烦 类少还可以  要是多了 则需要新建很多实体类   更关键的是 服务端的实体类 更新了  以前WCF或ws 更新下服务即可 而现在 还要更新客户端实体类岂不是很麻烦？

   所以推荐用下面的方法

**2.将json 反序列话为 dynamic 类型**

    json 转换为dynamic 类型   起初 我用的是 第三方的 json.net  经过尝试 发现再把json 转换为集合类型时 总是出错  如果有朋友知道怎么弄 还望告知下

    后来经过尝试 我采用了 微软自带的 **JavaScriptSerializer  下面展示代码**



 使用方法很简单 经过  不论是序列化 集合 还是单个的  都没有问题 这个时间也会正常显示 没有问题~  结果上面的代码  就可以把接受的json 类型 展示为一个 table了

 好了 这个问题解决了 可以上三种方式发送get请求了 这个毕竟简单 大家直接看代码



可以看到 第一个 和第三个 都是非常简单的  这里要注意的是  要注意编码问题 因为传递的有中文 会有乱码问题

所以要注意编码问题  再第二个里 可以通过得到response.ContentType 先得到编码  这样就可以自动识别编码格式了

再说下HttpClient  这个真正发送请求  是在 message.EnsureStatusIsSuccessful();  这句话

通过上面任意一种方法 我们都可以得到json  这样再通过**将json 反序列话为 dynamic 类型** 就可以像有实体类一样简单的操作了~~

[**WCF REST 工作总结（二）**](http://www.cnblogs.com/wlflovenet/archive/2011/10/30/WCFREST2.html)

上一篇 我们搭建了WCF REST 的服务 平且通过 三种方式发送get请求  实现了获取json  没看过的朋友可以先去看下----[REST 工作总结一](http://www.cnblogs.com/wlflovenet/archive/2011/10/29/WCFREST.html)

今天主要写下  POST等其他方式 发送请求 以及 流方式 文件的上传与下载

**一.Post 提交数据**

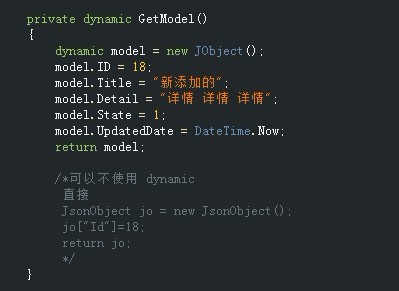
先来想下 POST和Get 的不同   Get 方式 我们直接通过 url  来传递参数   先来看下我们的  创建方法吧

[WebInvoke(UriTemplate = "Tasks/Add", Method = "POST", RequestFormat = WebMessageFormat.Json)]  
 void Create(PocoTask pocoTask);

我们需要接受一个实体类   格式为json格式  这就再次出现上文的问题  我们的客户端没有实体类  没法把实体类序列化为json  难道要自己拼接json么？ 当然不用。那简直太痛苦了。

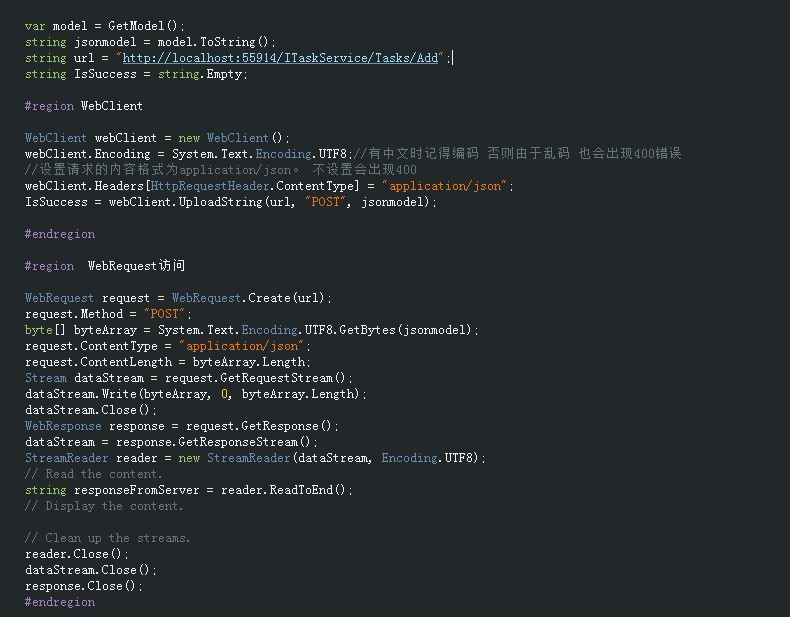
这里给大家推荐 [Json.Net](http://json.codeplex.com/)  这里有两个类 可以帮我们完成  单个实体的转换 以及集合实体的转换 分别是 JObject 和 JArray

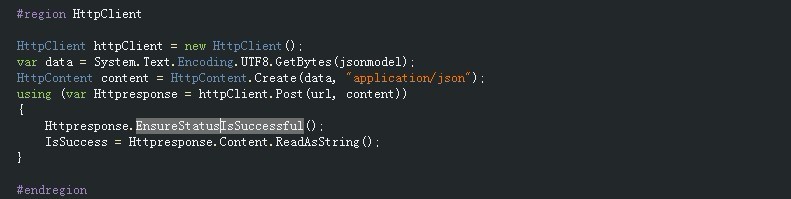
我们只需声明这两个类后  .toString 就可得到他的json格式了  很方便吧~ 看代码



这里完全可以不用dynamic的  像注释那样的写法就行 这样即使服务器不支持.net4.0 也可以使用的

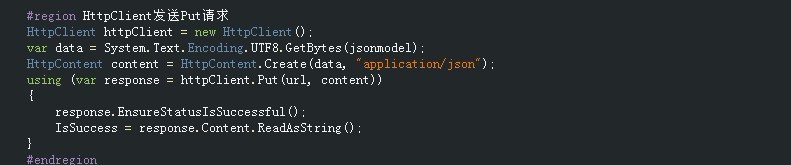
知道了如何构建POST 提交的数据 就开始实现POST提交吧~ 上代码 依然是三种方式 放一起 对比





这三个方式 个人很推荐 第三种 也是目前最新的 .net4.5里集成的 访问REST 非常的方便 因为他支持 直接点出来 PUT 以及 DELETE请求

简直是为rest 量身定制的~~ 看下 PUT请求



这里需要注意下  因为我们发送的是json 不论是哪种发送请求的方式 都要记得 设置ContentType=application/json

否则会抛出 HTTP 400的错误  400-错误的请求 很多都是客户端请求不正确造成的  所以一般出来400错误 可以重点检查下 客户端请求的问题

因为 PUT 与 DELETE 和 POST 基本类似  我就不上代码了

**二.REST 的文件上传 与下载**

流文件的上传和下载

先来看服务端的接口与实现



我们要实现三个功能

1. 展示一张图片

2.下载一个文件

3.上传一个文件

一个一个说 先说展示图片 实现代码

这个是在浏览器中直接打开  比较简单  这里记得设置输出的 ContentType  有时我们想打开的是别的格式 这里有个小方法  根据文件扩展名获得输出的ContentType

也就是上面用到的GetContentType方法  在园子的知识库里发现的[GetContentType](http://archive.cnblogs.com/a/982341/)

客户端 非常简单 直接引用连接即可



2.下载

上面的是直接把文件显示在浏览器中 如果我们希望下载资源呢 方法也很简单和以前的方法是一样的  直接上代码

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif文件下载

3.**上传**  
这个文件上传 才是重点要说的  折腾了好久~

首先要说的是  一定要记得配置文件 如果用REST 模版 配置方法如下

[复制代码](javascript:void(0);)

<system.serviceModel>  
 <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>  
 <standardEndpoints >  
 <webHttpEndpoint >  
 <!--   
 Configure the WCF REST service base address via the global.asax.cs file and the default endpoint   
 via the attributes on the <standardEndpoint> element below  
 -->  
 <standardEndpoint name="" maxReceivedMessageSize="3000000" defaultOutgoingResponseFormat="Json" helpEnabled="true" automaticFormatSelectionEnabled="true">  
 <readerQuotas maxArrayLength="300000"/>  
 </standardEndpoint>  
   
 </webHttpEndpoint>  
 </standardEndpoints>  
 </system.serviceModel>

[复制代码](javascript:void(0);)

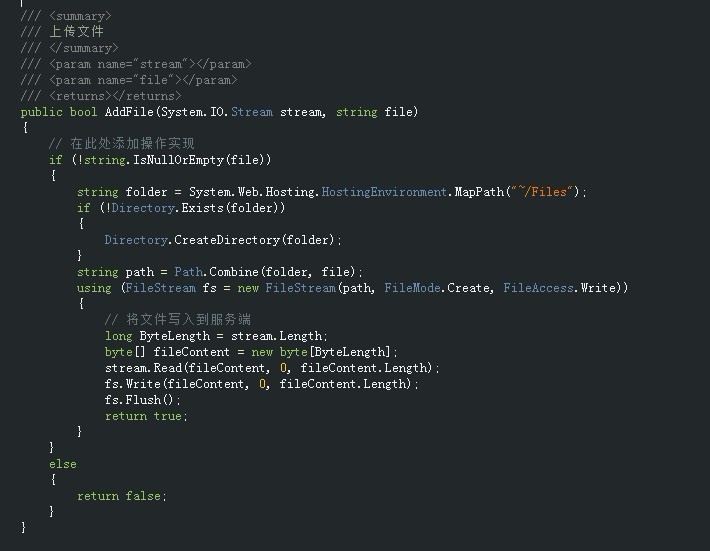
这样就可以上传大文件了

客户端实现上传很简单 借助 HttpClient



本来写到这  以为上传很简单的就能完成了 结果却弄了好久

先说下以前的写法

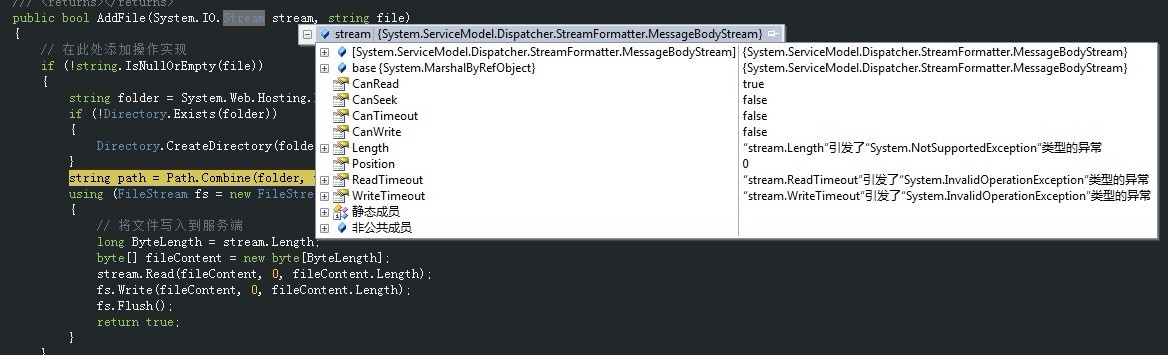


以前这样 都能成功  谁知道这次移到 REST WCF 中  竟然报错  发现不能读取Stream.Length   在这里问下各位高手 ~为什么读取Stream.Length 会报错？

CSDN 的  [fangxinggood](http://blog.csdn.net/fangxinggood)  给了解释 非常感谢

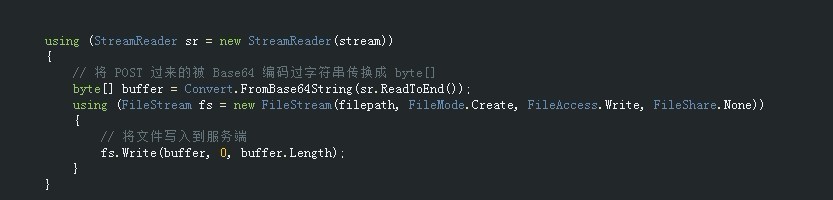
因为它被封装成了MessageBodyStream，就像WCF Streamed的数据一样，Length是不支持的。WCF里是客户端一边传服务端一边收。到达服务端开始写IO时，流并没有全部传输完，所以 MessageBodyStream不支持Length。但REST时是Http协议，所以其实Stream已经全部到达服务端了。如果你想取长度，可以 用： var len = WebOperationContext.Current.IncomingRequest.ContentLength;来获得。

如图



最后就会抛出Http 400的错误

百思不得其解  只好换了另一种方法这种方法不需要 长度length



可是却发现在 转化成byte 这次是报错  输入的不是有效的 Base-64 字符串，因为它包含非 Base-64 字符、两个以上的填充字符，或者填充字符间包含非空白字符。

再次请问原因？  希望遇到过的朋友告知

**解决办法**

后来 通过谷歌 发现另外一种 获得长度的方法 在WCF REST里   这样就能用第一种方法了

把最早方法里的

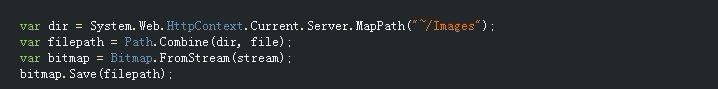
  long ByteLength=stream.Length;

  换成

  long ByteLength = WebOperationContext.Current.IncomingRequest.ContentLength;

  这样就可以得到长度了  上传问题也就解决了~~

 这里说下 如果上传的是图片 有个很简单的方法   核心代码 简单两行即可~



**三.Session与Cookie**

 在WCF 里  是可以使用session的  但是在REST 的WCF 里  貌似却不能使用session 查了些资料 没有发现使用session的办法

 可能是因为使用session会让服务器和客户端耦合。 后来找到了 REST 常见误区  <http://www.prescod.net/rest/mistakes/>

 里面提到了一条  session 是无关紧要的。

 cookie 还是可以正常使用的  这里说下 客户端读取cookie的方法  webclient 貌似不能读到 要使用更加灵活的 [HttpWebRequest](http://msdn.microsoft.com/zh-cn/library/system.net.httpwebrequest%28VS.80%29.aspx)



主要是这两句

     CookieContainer Cookie = new CookieContainer();  
     request.CookieContainer = Cookie;

**四.总结**

在做这方面的工作  把经验分享给大家 也希望会的人 能解答下我问中的问题~~

REST WCF  体现了ROA 面向资源编程  感觉更好了解耦了客户端与服务端  而且调用 REST 服务  也简单了很多 没有了代理等 效率也提高了 喜欢这种风格的方式

# [[WCF REST] 帮助页面与自动消息格式(JSON/XML)选择(转)](http://bbs.28tui.com/thread-687549-1-1.html)

转自Artech，更多WCF技术学习请见.net教程之《VS2010轻松学习C#-从零到深入-天轰穿.NET4趣味编程视频教程》更系统的教程讲解

可 以说WebHttpBinding和WebHttpBehavior是整个Web HTTP编程模型最为核心的两个类型，前者主要解决消息编码问题，而余下的工作基本上落在了终结点行为WebHttpBehavior上。 WebHttpBehavior属性HelpEnabled和AutomaticFormatSelectionEnabled是“帮助页面”与“自动消 息格式选择”这两个特性的总开关。[“自动消息格式(JSON/XML)选择”源代码从这里下载]

  1:publicclassWebHttpBehavior :IEndpointBehavior, ...

   2: {

  3://其他成员

   4:publicvirtualboolHelpEnabled{ get; set; }

  5:publicvirtualboolAutomaticFormatSelectionEnabled{ get; set; }

   6: }

**一、 帮助页面**

.NET教程 中 WCF 4.0为REST服务提供了帮助页面功能，我们可以通过浏览器访问服务帮助页面的地址得到所有操作的基本信息。但是这个功能在默认的情况下是关闭的，我们 需要通过应用在终结点上的WebHttpBehavior行为的HelpEnabled属性开启该功能。

  1:<configuration>

   2:<system.serviceModel>

  3:<behaviors>

   4:<endpointBehaviors>

  5:<behaviorname="webHttp">

   6:<webHttphelpEnabled="true"/>

  7:</behavior>

   8:</endpointBehaviors>

  9:</behaviors>

  10:<services>

11:<servicename="Artech.WcfServices.Service.EmployeesService">

  12:<endpointaddress="http://127.0.0.1:3721/employees"

13:binding="webHttpBinding"

  14:contract="Artech.WcfServices.Service.Inte\*\*ce.IEmployees"/>

15:</service>

  16:</services>

17:</system.serviceModel>

  18:</configuration>

同 样以之前演示的EmployeesService为例，我们通过如上的配置将终结点行为WebHttpHehavior应用在服务唯一的终结点上，并将 HelpEnabled属性设置为True。那么基于终结点的帮助页面将以地址{终结点地址}/Help发布出来，我们通过浏览器访问这个地址就会得到如 下图所示帮助页面。

如上图所示，帮助页面列出了包括相对地址、HTTP方法和基本描述在内的所有操作的基本信息。我们通过点击HTTP 方法对应的链接可以获得包括基于相应格式（Xml和Json）消息结构（Schema）和实例。在默认的情况下，帮助页面中表示操作描述信息的格式为 “Service at {操作地址}”，我们可以在定义服务契约的时候再操作方法上应用特性DescriptionAttribute来定义出现在帮助页面中的描述信息。

  1:[ServiceContract]

   2:publicinte\*\*ceIEmployees

  3: {

   4:    [WebGet(UriTemplate = "all")]

  5:     [Description("获取所有员工列表")]

   6:IEnumerable<Employee>GetAll();

  7:

   8:    [WebGet(UriTemplate = "{id}")]

  9:     [Description("获取指定ID的员工")]

  10:    Employee Get(string id);

11:

  12:    [WebInvoke(UriTemplate = "/", Method = "POST")]

13:     [Description("创建一个新的员工")]

  14:    Employee Create(Employee employee);

15:

  16:    [WebInvoke(UriTemplate = "/", Method = "PUT")]

17:     [Description("修改现有员工信息")]

  18:voidUpdate(Employee employee);

19:

  20:    [WebInvoke(UriTemplate = "{id}", Method = "DELETE")]

21:     [Description("删除指定ID的员工")]

  22:voidDelete(string id);

23: }

如上面的代码片断所示，我们在契约接口IEmployees中的所有操作方法上应用了DescriptionAttribute特性并指定了相应的描述信息。这些描述信息就是出现在如下图所示的帮助页面中。

**二、 自动消息格式选择**

REST 服务具有两种基本的消息格式（Xml和Json）。在定义服务契约的时候，我们可以通过应用在操作方法上的WebGetAttribute和 WebInvokeAttribute指定回复消息的格式。如果没有通过这种方式对消息格式进行显式设置，我们还可以通过终结点行为 WebHttpBehavior为回复消息设置一个默认的消息格式。除了这种显示设置方式之外，WCF还提供一种自动消息格式选择机制。

所 谓消息格式的自动选择，就是服务根据请求消息来选择一种适合的格式进行消息的序列化。在默认的情况下，这种自动选择机制是关闭的，我们需要通过 WebHttpBehavior的AutomaticFormatSelectionEnabled属性开启该机制。具体的消息格式选择机制策略（顺序） 如下：

* 如果作为请求的HTTP消息具有Accept报头，则根据该报头决定回复消息的格式；
* 如果作为请求的HTTP消息具有Content-Type报头，则根据该报头决定回复消息的格式；
* 如果在定义服务契约时通过WebGetAttribute或者WebInvokeAttribute对回复消息的格式进行了显式设置，则采用该消息格式；
* 如果通过终结点行为WebHttpBehavior设置了对回复消息的格式进行了显式设置，则采用该消息格式；
* 采用默认消息格式Xml（WebMessageFormat枚举的默认值）。

我们同样通过之前创建的EmployeesService的实例来演示消息格式的自动选择机 制。如下面的配置片断所示，我们将WebHttpBehavior行为应用到了寄宿服务的唯一终结点上，并且将 AutomaticFormatSelectionEnabled属性设置为True。

  1:<configuration>

   2:<system.serviceModel>

  3:<behaviors>

   4:<endpointBehaviors>

  5:<behaviorname="webHttp">

   6:<webHttpautomaticFormatSelectionEnabled="true" />

  7:</behavior>

   8:</endpointBehaviors>

  9:</behaviors>

  10:<services>

11:<servicename="Artech.WcfServices.Service.EmployeesService">

  12:<endpoint address="http://127.0.0.1:3721/employees"

13:behaviorConfiguration="webHttp"

  14:                      binding="webHttpBinding"

15:                       contract="Artech.WcfServices.Service.Inte\*\*ce.IEmployees"/>

  16:</service>

17:</services>

  18:</system.serviceModel>

19:</configuration>

对于契约接口IEmployees来说，我们通过WebGetAttribute特性用于返回所有员工列表的GetAll操作的回复消息格式设置为Xml。

  1:[ServiceContract]

   2:publicinte\*\*ceIEmployees

  3: {

   4://其他成员

  5:     [WebGet(UriTemplate = "all",ResponseFormat = WebMessageFormat.Xml)]

   6:IEnumerable<Employee>GetAll();

  7: }

对 于REST服务调用来说，其本质就是一种普通的HTTP请求，与针对某个网页的访问并没有什么本质的不同，所以我们完全可以手工生成HTTP请求来进行服 务的访问。为此我们创建了如下一个静态方法GetAllEmployees方法通过WebClient对服务的GetAll操作进行调用，并将整个回复消 息打印出来，该方法的两个参数分别是作为请求的HTTP消息的Content-Type和Accept报头值。

  1:staticvoidGetAllEmployees(stringcontentType, string accept)

   2: {

  3:WebClientwebClient= newWebClient();

   4:if (!string.IsNullOrEmpty(contentType))

  5:     {

   6:webClient.Headers.Add("Content-Type", contentType);

  7:     }

   8:

  9:if (!string.IsNullOrEmpty(accept))

  10:     {

11:webClient.Headers.Add("Accept", accept);

  12:     }

13:using (StreamReader reader = newStreamReader(webClient.OpenRead(

  14:"http://127.0.0.1:3721/employees/all")))

15:     {

  16:Console.WriteLine(reader.ReadToEnd());

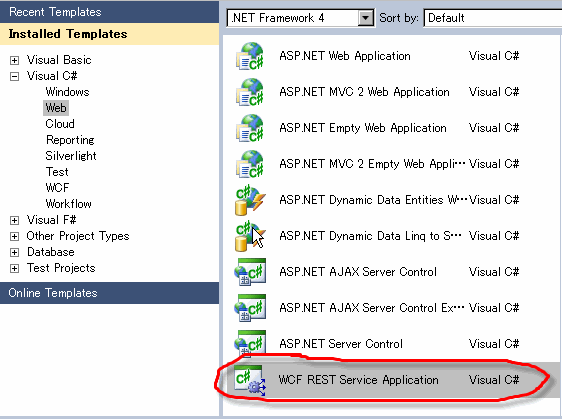
17:     }

  18: }

### [WCF4.0 –- RESTful WCF Services (3) (Raw Stream)](http://blog.csdn.net/fangxing80/article/details/6261431)

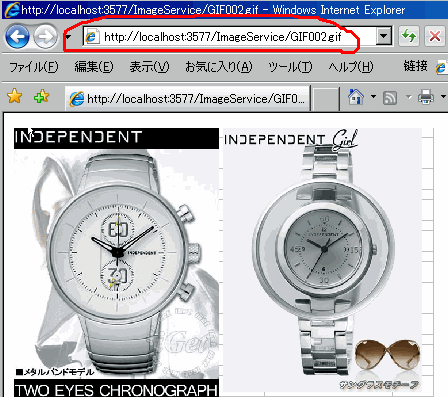
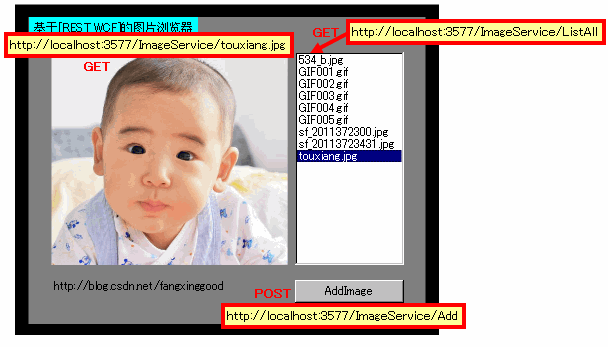
分类： [[05] WCF](http://blog.csdn.net/fangxinggood/article/category/767778) 2011-03-19 17:18 5452人阅读 [评论](http://blog.csdn.net/fangxing80/article/details/6261431#comments)(22) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/fangxing80/article/details/6261431#report)

[wcf](http://blog.csdn.net/tag/details.html?tag=wcf)[stream](http://blog.csdn.net/tag/details.html?tag=stream)[string](http://blog.csdn.net/tag/details.html?tag=string)[rest](http://blog.csdn.net/tag/details.html?tag=rest)[image](http://blog.csdn.net/tag/details.html?tag=image)[object](http://blog.csdn.net/tag/details.html?tag=object)

前面2篇blog，主要在介绍REST WCF如何通过json/xml 和客户端完成交互。 这篇文章将对REST WCF的原生(RAW)流传输进行一次实例分析。  
  
OK，还是利用 WCF REST Service Application 的工程模板，创建我们的REST WCF工程。  
  
  
因为WCF REST服务不需要公开metadata, 所以 OperationContract 的定义也省略了。直接定义[ServiceContract]  
如果修改模板中Service名，**Global.asax**中的**RegisterRoutes**方法也必须相应的修改。  
下面是服务的实现：

**[c-sharp]** [view plaincopy](http://blog.csdn.net/fangxing80/article/details/6261431)

1. [ServiceContract]
2. [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
3. [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall, Namespace="")]
4. public class ImageService
5. {
6. [WebGet(UriTemplate = "{image}")]
7. public Stream GetImage(string image)
8. {
9. var imageType = Path.GetExtension(image).TrimStart('.');
10. WebOperationContext.Current.OutgoingResponse.ContentType = "image/" + imageType;
11. var dir = System.Web.HttpContext.Current.Server.MapPath("~/Images");
12. var file = Path.Combine(dir, image);
13. return File.OpenRead(file);
14. }
16. [WebInvoke(UriTemplate = "Add/{image}", Method="POST")]
17. public void AddImage(Stream stream, string image)
18. {
19. var dir = System.Web.HttpContext.Current.Server.MapPath("~/Images");
20. var file = Path.Combine(dir, image);
21. var bitmap = Bitmap.FromStream(stream);
22. bitmap.Save(file);
23. }
25. [WebGet(UriTemplate = "ListAll", ResponseFormat=WebMessageFormat.Xml)]
26. public string[] GetAllImageNames()
27. {
28. var dir = System.Web.HttpContext.Current.Server.MapPath("~/Images");
29. var files = Directory.GetFiles(dir);
30. var images = new List<string>();
31. foreach (var file in files)
32. {
33. var ext = Path.GetExtension(file);
34. if (ext == ".jpg" || ext == ".gif")
35. images.Add(file);
36. }
37. for (int i = 0; i < images.Count; i++)
38. images[i] = Path.GetFileName(images[i]);
39. return images.ToArray();
40. }
41. }

- GetImage 用于返回图片 Stream，   
- AddImage 用于接受图片 Stream 并保存，   
- GetAllImageNames 返回所有图片名。  
  
在Http协议中，通过content-type控制客户端的解析行为。当content-type是"image/jpg"或者是"image/gif"的时候，  
浏览器就能够直接显示图片了。因此，在代码里有这样的设定：  
**WebOperationContext.Current.OutgoingResponse.ContentType = "image/" + imageType;**  
还可以通过 OutgoingResponse.Header 控制缓存。另外，这个REST WCF是建立在一个Web Application中，  
我们还可以利用  System.Web.HttpContext.Current.Server.MapPath 方法，获得物理路径。  
  
我 们知道一般的WCF服务中，可以实现Streamed传输。在那种场合下，服务契约必须的参数或者返回值必须是Stream，且不能有别的类型的参数（要 传递别的参数可以通过MessageHeader)。但是在上面的示例代码中，我定义的 AddImage 就有两个参数：   
一个是Stream,还有一个是string。这个string参数其实是通过url传递的： UriTemplate = "Add/**{image}**"，且这两个参数没有顺序要求。  
**如果没有写"{image}"会抛出下面的异常（因为Request的Body已经被定义为Stream，图片名只能通过Url传过来。）**  
Operation 'AddImage' of contract 'ImageService' specifies multiple request body parameters to be serialized without any wrapper elements. At most one body parameter can be serialized without wrapper elements. Either remove the extra body parameters or set the BodyStyle property on the WebGetAttribute/WebInvokeAttribute to Wrapped.   
  
运行服务，直接在浏览器里输入服务URL：<http://localhost:3577/ImageService/GIF002.gif> 我们就可以看到结果了。  
另外，利用REST WCF返回原生流的特性，我们完全可以取代以前在 ASP.NET 中专门生成验证码或者图片读取步骤的页面类，  
是不是很方便？  
  
  
  
为了体现 REST WCF 的便利性， 我设计了一个 WPF 的客户端：  
  
客户端代码：

**[c-sharp]** [view plaincopy](http://blog.csdn.net/fangxing80/article/details/6261431)

1. public partial class MainWindow : Window
2. {
3. public MainWindow()
4. {
5. InitializeComponent();
6. }
8. private void Window\_Loaded(object sender, RoutedEventArgs e)
9. {
10. op.RestoreDirectory = true;
11. op.Filter = "Jpeg Files(\*.jpg)|\*.jpg|Gif Files(\*.gif)|\*.gif";
13. BindData();
14. }
16. private void BindData()
17. {
18. var url = "http://localhost:3577/ImageService/ListAll";
19. var client = new HttpClient();
20. var stream = client.Get(url).Content.ReadAsStream();
21. var dataSer = new DataContractSerializer(typeof(string[]));
22. var obj = (string[])dataSer.ReadObject(stream);
23. this.listBox1.DataContext = obj;
24. }
26. private Microsoft.Win32.OpenFileDialog op = new Microsoft.Win32.OpenFileDialog();
28. private void button1\_Click(object sender, RoutedEventArgs e)
29. {
30. var ret = op.ShowDialog();
31. if (!ret.HasValue || !ret.Value || !op.CheckFileExists) return;
32. var file = op.FileName;
33. var name = System.IO.Path.GetFileName(file);
34. var url = "http://localhost:3577/ImageService/Add/" + name;
35. var client = new HttpClient();
36. var content = HttpContent.Create(File.OpenRead(file));
37. var resp = client.Post(url, content);
38. resp.EnsureStatusIsSuccessful();
40. BindData();
41. }
43. private void button2\_Click(object sender, RoutedEventArgs e)
44. {
45. this.Close();
46. }
47. }
49. [ValueConversion(typeof(string), typeof(string))]
50. public class UriConverter : IValueConverter
51. {
52. public object Convert(object value, Type targetType,
53. object parameter, System.Globalization.CultureInfo culture)
54. {
55. if (value == null) return null;
56. string sourceValue = value.ToString();
57. return "http://localhost:3577/ImageService/" + sourceValue;
58. }
60. public object ConvertBack(object value, Type targetType,
61. object parameter, System.Globalization.CultureInfo culture)
62. {
63. return value;
64. }
65. }

实现的注意点：  
1) 获得所有图片名，服务端返回的是string[]的序列化的xml，用DataContractSerializer反序列化。  
    var dataSer = new DataContractSerializer(typeof(string[]));  
2) WPF中的OpenFileDialog在Microsoft.Win32命名空间下。（必须引用WindowsBase.dll）  
3) 利用 Microsoft.Http.HttpClient 上传图片, 实例化 HttpContent, 通过client.Post上传  
    var url = "<http://localhost:3577/ImageService/Add/>" + name;  
    var client = new HttpClient();  
    var content = HttpContent.Create(File.OpenRead(file));  
    var resp = client.Post(url, content);  
    resp.EnsureStatusIsSuccessful();  
4) 定义了一个Converter, 将 ListBox 绑定到 Image 控件的 Source  
    Source="{Binding **ElementName**=listBox1, **Path**=SelectedItem, **Converter**={StaticResource uriConv}}"

源代码下载：[**REST WCF Raw Stream 示例代码**](http://download.csdn.net/detail/fangxinggood/3627896)

【REST WCF系列】  
[RESTful WCF Services (1) (入门)](http://blog.csdn.net/fangxinggood/archive/2011/03/10/6235662.aspx)  
[RESTful WCF Services (2) (实现增，删，改，查)](http://blog.csdn.net/fangxinggood/archive/2011/03/14/6247297.aspx)  
[RESTful WCF Services (3) (Raw Stream)](http://blog.csdn.net/fangxinggood/archive/2011/03/19/6261431.aspx)  
[RESTful WCF Services (4) (Basic Security)](http://blog.csdn.net/fangxinggood/archive/2011/03/21/6263780.aspx)  
[RESTful WCF Services (实例) (并发同步服务 SyncService)](http://blog.csdn.net/fangxinggood/archive/2011/03/24/6272886.aspx)

分享到：

 上一篇：[WCF Data Services 与 WCF RIA Services 的比较](http://blog.csdn.net/fangxing80/article/details/6255140)

 下一篇：[WCF4.0 –- RESTful WCF Services (4) (Basic Security)](http://blog.csdn.net/fangxing80/article/details/6263780)