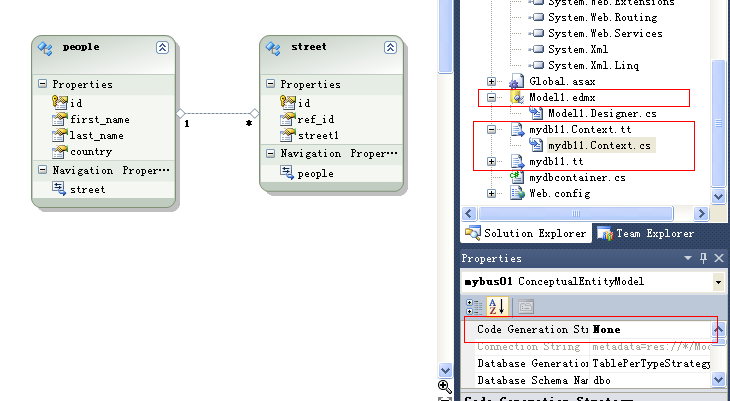
**WCF Restful Service:**

**本笔记的例子使用如下的 ADO.Net Entity Framework**

****

WCF 注意的地方：

1）如果对于服务类没有服务契约[ServiceContract], 那么我们可以将 [ServiceContract] 直接应用到服务类上。

如下：

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

[ServiceContract]

[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]

public class mysvc

{ … }

那么服务类与服务契约合二为一， 类既是契约， 契约也是类。

2）如果服务类继承自服务契约， 则服务类不能有[ServiceContract]，必须从服务类拿掉

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

[ServiceContract]

[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]

public class mysvc : idb

{ …. }

这样会提示出错：

Server Error in '/' Application.

*The service class of type webWCFRest1.mysvc both defines a ServiceContract and inherits a ServiceContract from type webWCFRest1.idb. Contract inheritance can only be used among interface types.  If a class is marked with ServiceContractAttribute, it must be the only type in the hierarchy with ServiceContractAttribute.  Consider moving the ServiceContractAttribute on type webWCFRest1.idb to a separate interface that type webWCFRest1.idb implements.*

3）如果服务类只继承自一个服务契约：[ServiceContract]

那么我们可以不在 web.config 里配置 ServiceEndpoint. ASP.Net 将自动生成

但是为了能够找到服务的描述： 必须添加 <ServiceBehaviors> - WSDL

<serviceBehaviors>

<behavior name="svc\_bh"> // **不能添加 Name 除非你指定了ServiceEndpoint -> behavorConfigName**

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

不加 Name属性， 对于单个服务契约的服务来说， 一切都取默认值， 自动匹配。

WCF Restful 服务需要用到的命名空间：

using System.Web;

using System.ServiceModel;

using System.ServiceModel.Description;

using System.ServiceModel.Web;

using System.ServiceModel.Activation;

using System.ComponentModel;

Global.asax.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.SessionState;

using System.Web.Routing;

using System.ServiceModel.Web;

using System.ServiceModel.Activation;

1. **需要 AspNetCompatibility = true;**

在 App.config 或者 web.config 必须有这句话, 否则出错。

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />

<configuration>

<system.serviceModel>

<behaviors>

<endpointBehaviors>

<behavior>

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true"/>

</behavior>

</endpointBehaviors>

</behaviors>

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />

</system.serviceModel>

</configuration>

另外服务类也需要：

[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]

[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]

public class mysvc : idb ｛

……

｝

1. **关于帮助信息：默认 Restful 是没有帮助页面的**

<system.serviceModel>

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint automaticFormatSelectionEnabled="true" helpEnabled="true" />

</webHttpEndpoint>

</standardEndpoints>

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />

</system.serviceModel>



另外也可以如此设置：

<system.serviceModel>

<behaviors>

―― serviceBehaviors 是为传统的 SOAP 服务提供 WSDL ――――――――――

<serviceBehaviors>

<behavior>

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

―――――――――――――――――――――――――――――――――――――

―――― WebServiceHost Help Page ---------―――――――――――――――――

<endpointBehaviors>

<behavior>

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true"/>

</behavior>

</endpointBehaviors>

―――――――――――――――――――――――――――――――――――――

</behaviors>

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />

</system.serviceModel>

如果两者都设置了，

<system.serviceModel>

<behaviors>

<serviceBehaviors>

<behavior>

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

――――WebServiceHost Help Page ――――――――――――――――――――――

<endpointBehaviors>

<behavior>

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true"/>

</behavior>

</endpointBehaviors>

―――――――――――――――――――――――――――――――――――――

</behaviors>

―――― WebServiceHost Help Page――――――――――――――――――――――

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint helpEnabled="false" automaticFormatSelectionEnabled="true" />

</webHttpEndpoint>

</standardEndpoints>

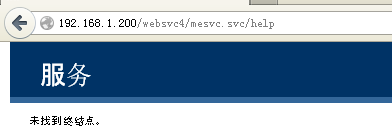
―――――――――――――――――――――――――――――――――――――――

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />

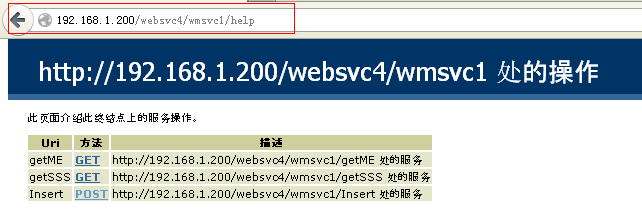
</system.serviceModel>

<standardEndpoints> 的优先级别比 <endpointBehaviors> 的高

所以以上设置是看不到帮助页面的：



如果 <standardEndpoints> 为 true , <endpointBehaviors> 为false , 则可以看到帮助页面



1. **Restful WCF 的托管， Host**

**方法一： 使用 Global.asax 在 Application\_Start() 事件里注册服务路由。无需 service1.svc svc文件**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.SessionState;

using System.ServiceModel.Web;

using System.ServiceModel.Activation;

using System.ServiceModel.Routing;

using System.Web.Routing;

namespace webWCF2

{

public class Global : System.Web.HttpApplication

{

protected void Application\_Start(object sender, EventArgs e)

{

this.regSvc();

}

private void regSvc()

{

ServiceRoute sr = new ServiceRoute("rest", new WebServiceHostFactory(), typeof(mydb));

RouteTable.Routes.Add(sr);

}

}

}

要使Resful WCF 能工作：需要做以下改动：

1. Web.config 添加以下项目：

<configuration>

<system.serviceModel>

――――― 这是添加帮助页面 ―――――――――

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint helpEnabled="true" automaticFormatSelectionEnabled="true" />

</webHttpEndpoint>

</standardEndpoints>

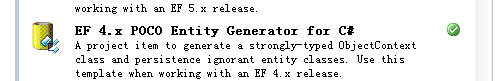
――――― 这是必须的， 否则出错 ――――――

<serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />

</system.serviceModel>

</configuration>

1. 如果是使用 ADO.Net Entity Framework ¸ xxx.edmx 则不能使用， 必须使用模板转化，通常使用



转化后的 ObjectContext: 不支持：ProxyCreationEnabled 和 LazyLoadingEnabled ， 所以必须设置为 false

public partial class mybus : ObjectContext

{

public const string ConnectionString = "name=mybus";

public const string ContainerName = "mybus";

#region Constructors

public mybus() : base(ConnectionString, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = false;

}

public mybus(string connectionString) : base(connectionString, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = false;

}

public mybus(EntityConnection connection) : base(connection, ContainerName)

{

this.ContextOptions.ProxyCreationEnabled = false;

this.ContextOptions.LazyLoadingEnabled = false;

}

#endregion

因为REST 是基于HTTP的， 所以对于 REST 的客户端的开发者，无法像传统的 WebService或者其他的WCF服务通过引用wsdl，享受“奢侈”的代码生成，而使用强类型的本地代理调用服务。 开发者只能通过 Http Request 的组装， 但正因为这种直接的HttpRequest组装，而使得客户端真正是语言无关的。这里不得不提一下 Microsoft.Http.dll 和 Microsoft.Http.Extensions.dll，它们是微软提供的REST客户端包。可以更加方便地操作 HttpRequest/Response，你可以在这里下到： <http://aspnet.codeplex.com/releases/view/24644> 

1. 定义服务契约，和服务类：

服务契约

[ServiceContract]

public interface iSQL

{

[OperationContract]

List<people> getP();

[OperationContract]

List<street> getS();

}

服务类

[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Single, InstanceContextMode=InstanceContextMode.Single)]

public class mydb : iSQL

{

public mybus db;

public mydb()

{

db = new mybus();

}

[WebGet(UriTemplate="getPP", RequestFormat=WebMessageFormat.Json, ResponseFormat=WebMessageFormat.Xml)]

public List<people> getP()

{

using (db = new mybus())

{

return db.people.ToList<people>();

}

}

[WebGet(UriTemplate = "getSS", ResponseFormat = WebMessageFormat.Json)]

public List<street> getS()

{

using (db = new mybus())

{

return db.street.ToList<street>();

}

}

}

<system.serviceModel>

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint helpEnabled="true" automaticFormatSelectionEnabled="false" />

</webHttpEndpoint>

</standardEndpoints>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />

</system.serviceModel>

如果 automaticFormatSelectionEnabled="false" 不能自动选择格式。则严格按照 WebGet , WebInvoke 的设置发送和返回数据格式

查看帮助页面：



1. 命名空间：

using System.ServiceModel;

[ServiceContract]

[OperationContract]

[ServiceBehavior]

using System.ServiceModel.Web;

[WebGet(UriTemplate="getpp" ,RequestFormat=WebMessageFormat.Json, ResponseFormat=WebMessageFormat.Json)]

[WebInvoke(UriTemplate="ins",RequestFormat=WebMessageFormat.Json)]

**方法二：使用 xxx.svc 文件托管 Rest Service:**

<%@ ServiceHost Language="C#" Debug="true"

factory="System.ServiceModel.Activation.WebServiceHostFactory"

Service="wcfRest2.cls1"

CodeBehind="cls1.cs" %>

主要是 factory="System.ServiceModel.Activation.WebServiceHostFactory"

namespace wcfRest2

{

[ServiceBehavior(Name="dbsvc")]

以下是必须的，否则启动服务出错: AspNetCompatibilityRequirementsMode.Allowed

[AspNetCompatibilityRequirements(RequirementsMode=AspNetCompatibilityRequirementsMode.Allowed)]

public class cls1:idb01

{

public cls1() { }

[WebGet(UriTemplate="pps", ResponseFormat=WebMessageFormat.Json)]

public List<people> getPPS()

{

using (mydb1 db = new mydb1())

{

return db.people.ToList();

}

}

[WebGet( UriTemplate="get(id=lwh{id})", ResponseFormat = WebMessageFormat.Json)]

public people getPPID(string id)

{

using (mydb1 db = new mydb1())

{

int sid = Convert.ToInt32(id);

return db.people.Include("street").Where(pp => pp.id == sid ).SingleOrDefault();

}

}

[WebInvoke(UriTemplate = "ins",

// BodyStyle=WebMessageBodyStyle. WrappedRequest, // **不能指定, 否则出错.**

RequestFormat=WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

public string insPP(people p)

{

using (mydb1 db = new mydb1())

{

if (p != null)

{

db.people.AddObject(p);

db.SaveChanges();

}

}

if (p != null)

return p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "p is null";

}

[WebInvoke(UriTemplate = "updp",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

public string updPP(people p)

{

using (mydb1 db = new mydb1())

{

if (p != null)

return "update: " + p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "update: p is null";

}

}

[WebInvoke(UriTemplate="delp",

RequestFormat=WebMessageFormat.Json,

ResponseFormat=WebMessageFormat.Json, Method="POST")]

public string delPP(people p)

{

using (mydb1 db = new mydb1())

{

if (p != null)

return "Delete: " + p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "delete: p is null";

}

}

}

[ServiceContract]

public interface idb01 {

[OperationContract]

List<people> getPPS();

[OperationContract]

people getPPID(string iD);

[OperationContract]

string insPP(people p);

[OperationContract]

string updPP(people p);

[OperationContract]

string delPP(people p);

}

}

**客户端使用：JQuery Ajax:**

GET

$.ajax({

data: { },

dataType: "json",

error: function(xhr, tStatus, errorTh ) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function(req, tStatus) {

if( req.errorCode > 0 ) {

alert(tStatus);

return false;

} else {

toHTML(req);

}

},

type: "GET",

url: "http://192.168.1.200/wcfrest/rest2.svc/pps"

});

function add\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

// 重要：使用 JSON 内建的方法生成字符串。不能直接传递JSON object.

// 可以传递JSON字符: 必须单引号 ‘{“first\_name”:”William”}’ , 键值必须双引号。

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "POST",

url: "http://192.168.1.200/wcfrest/rest2.svc/ins"

});

}

### // PUT 和 DELETE 出现 ：[401 - 未授权: 由于凭据无效，访问被拒绝。](http://blog.csdn.net/xuejianwu/article/details/7787350)原因不明

function upd\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "PUT",

url: "http://192.168.1.200/wcfrest/rest2.svc/updp"

});

}

function del\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "DELETE",

url: "http://192.168.1.200/wcfrest/rest2.svc/delp"

});

}

**方法二：使用编程代码托管 Rest WCF Service:**

public WebServiceHost wsh;

private void Form1\_Load(object sender, EventArgs e) {

string str = null;

wsh = new WebServiceHost(

typeof(mybus),

new Uri[]{

new Uri("http://192.168.1.200:32000/rest")

}

);

ServiceEndpoint sep = wsh.AddServiceEndpoint(typeof(idb), new WebHttpBinding(), "good");

WebHttpBehavior whb = sep.Behaviors.Find<WebHttpBehavior>(); // **技巧：设置帮助页面**

if (whb != null) {

whb.AutomaticFormatSelectionEnabled = true;

whb.HelpEnabled = true;

} else {

whb = new WebHttpBehavior();

whb.AutomaticFormatSelectionEnabled = true;

whb.HelpEnabled = true;

sep.Behaviors.Add(whb); //帮助页面是附加在ServiceEndpoint上的。

}

wsh.Open();

str += string.Format("Service is :{0}\n\n", wsh.State);

foreach (ServiceEndpoint ep in wsh.Description.Endpoints)

{

str += string.Format("Ep: {0} - {1}\n", ep.Contract, ep.Address.Uri.AbsoluteUri);

}

this.label1.Text = str;

}

WCF SOAP & Rest 共存：

wsh = new ServiceHost(

typeof(mybus),

new Uri[]{

new Uri("http://192.168.1.200/rest")

}

);

/// 以下是 WCF SOAP 服务

ServiceMetadataBehavior smb = new ServiceMetadataBehavior();

smb.HttpGetEnabled = true;

smb.HttpGetUrl = new Uri("wss",UriKind.Relative);

wsh.Description.Behaviors.Add(smb);

ServiceEndpoint sep0 = wsh.AddServiceEndpoint(typeof(idb), new BasicHttpBinding(), "web");

/// 以下是 WCF Rest 服务

ServiceEndpoint sep = wsh.AddServiceEndpoint(typeof(idb), new WebHttpBinding(), "good");

WebHttpBehavior whb = sep.Behaviors.Find<WebHttpBehavior>();

if (whb != null) {

whb.AutomaticFormatSelectionEnabled = true;

whb.HelpEnabled = true;

} else {

whb = new WebHttpBehavior();

whb.AutomaticFormatSelectionEnabled = true;

whb.HelpEnabled = true;

sep.Behaviors.Add(whb);

}

wsh.Open();

SOAP WSDL 服务描述



WCF Rest Help 页面



**WCF SOAP 使用 App.config**

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="bdhttp" />

</basicHttpBinding>

</bindings>

<behaviors>

<serviceBehaviors>

<behavior name="svc\_bh">

<serviceMetadata httpGetEnabled="true" httpGetUrl="wss" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service behaviorConfiguration="svc\_bh" name="mysvc"> **// 注意 service . name 属性**

<endpoint address="http://192.168.1.200:32000/rest" binding="basicHttpBinding"

bindingConfiguration="bdhttp" name="ep" contract="WinRestServer.idb" />

<host>

<baseAddresses>

<add baseAddress="http://192.168.1.200/rest" />

</baseAddresses>

</host>

</service>

</services>

</system.serviceModel>

服务类：

[ServiceBehavior( InstanceContextMode=InstanceContextMode.PerCall,

ConcurrencyMode=ConcurrencyMode.Multiple,

**ConfigurationName**="mysvc")] // **对应config 的 service.name 属性**

class mybus : idb {

…………….

｝

启动服务：

ServiceHost wsh = new ServiceHost(typeof(mybus)); **// 不再需要指定任何配置，**

**//通过服务类ConfigurationName**="mysvc")] // **对应config 的 service.name 属性**

wsh.Open();

**WCF Rest 使用 App.config**

<system.serviceModel>

<bindings>

<webHttpBinding>

<binding name="bhrest"></binding>

</webHttpBinding>

<basicHttpBinding>

<binding name="bdhttp" />

</basicHttpBinding>

</bindings>

<behaviors>

<endpointBehaviors>

<behavior name="ep\_bh">

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />

</behavior>

</endpointBehaviors>

<serviceBehaviors>

<behavior name="svc\_bh">

<serviceMetadata httpGetEnabled="true" httpGetUrl="http://192.168.1.200:32000/rest/wss" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service behaviorConfiguration="svc\_bh" name="mysvc"> // **对应到服务类的 ConfigurationName**

**// SOAP 服务端点**

<endpoint address="http://192.168.1.200:32000/rest"

binding="basicHttpBinding"

bindingConfiguration="bdhttp"

name="ep"

contract="WinRestServer.idb" />

**// REST 服务端点**

<endpoint address="http://192.168.1.200:32000/rest/good"

behaviorConfiguration="ep\_bh" **对应到 endpointBehaviors.behavior,激活帮助页面等**

bindingConfiguration="bhrest" **对应到 binding**

binding="webHttpBinding"

name="wep"

contract="WinRestServer.idb" />

<host>

<baseAddresses>

<add baseAddress="http://192.168.1.200/rest" />

</baseAddresses>

</host>

</service>

</services>

</system.serviceModel>

**启动服务：**

WebServiceHost wsh = new WebServiceHost(typeof(mybus));

wsh.Open();

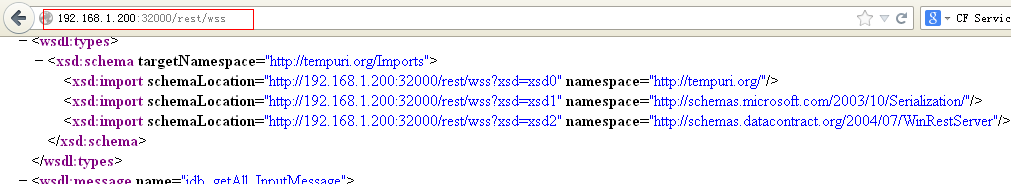
注意： 如果以 WebServiceHost 来启动服务， 则SOAP 服务无效， SOAP服务的WSDL，和服务端点查找与引用都不效。

ServiceHost wsh = new ServiceHost(typeof(mybus));

wsh.Open();

**如果**以 ServiceHost 启动服务， 则SOAP 和 REST 服务都有效。

<http://192.168.1.200:32000/rest/wss>



http://192.168.1.200:32000/rest/good/help



多个服务契约：

[ServiceContract]

public interface idb

{

[OperationContract]

[WebGet(UriTemplate="all", RequestFormat=WebMessageFormat.Json, ResponseFormat=WebMessageFormat.Json)]

List<people> getAll();

[OperationContract]

[WebGet( UriTemplate="byid(id={id})",

RequestFormat=WebMessageFormat.Json,

ResponseFormat=WebMessageFormat.Json)]

people getP(string id);

}

[ServiceContract]

public interface isql

{

[OperationContract]

[WebInvoke( UriTemplate = "insertp",

ResponseFormat = WebMessageFormat.Json,

RequestFormat = WebMessageFormat.Json, Method = "POST")]

string insP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "updatep",

ResponseFormat = WebMessageFormat.Json,

RequestFormat = WebMessageFormat.Json, Method = "PUT")]

string updP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "deletep",

ResponseFormat = WebMessageFormat.Json,

RequestFormat = WebMessageFormat.Json, Method = "DELETE")]

string delP(people p);

}

public interface idb

public interface isql

**服务类实现:**

[ServiceBehavior(

InstanceContextMode=InstanceContextMode.PerCall,

ConcurrencyMode=ConcurrencyMode.Multiple,

ConfigurationName="mysvc")]

class mybus: idb, isql {

public mybus() { }

public List<people> getAll() {

using (mydb01 db = new mydb01()) {

return db.people.ToList<people>();

}

}

public people getP(string id) {

using (mydb01 db = new mydb01())

{

int sid = Convert.ToInt32(id);

return db.people.Where(p => p.id == sid).SingleOrDefault();

}

}

public string insP(people p) {

using (mydb01 db = new mydb01()) {

db.people.AddObject(p);

db.SaveChanges();

}

if (p != null)

return string.Format("Insert: id:{0} fname:{1} lname:{2} cty:{3}", p.id, p.first\_name, p.last\_name, p.country);

else

return "Insert P is null";

}

public string updP(people p) {

using (mydb01 db = new mydb01()) {

db.people.AddObject(p);

db.SaveChanges();

}

if (p != null)

return string.Format("Update: id:{0} fname:{1} lname:{2} cty:{3}", p.id, p.first\_name, p.last\_name, p.country);

else

return "Update P is null";

}

public string delP(people p) {

using (mydb01 db = new mydb01()) {

db.people.AddObject(p);

db.SaveChanges();

}

if (p != null)

return string.Format("Delete: id:{0} fname:{1} lname:{2} cty:{3}", p.id, p.first\_name, p.last\_name, p.country);

else

return "Delete P is null";

}

}

配置文件：

<system.serviceModel>

<behaviors>

// REST Help Page

<endpointBehaviors>

<behavior name="ep\_bh">

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />

</behavior>

</endpointBehaviors>

//SOAP WSDL

<serviceBehaviors>

<behavior name="svc\_bh">

<serviceMetadata httpGetEnabled="true" httpGetUrl="http://192.168.1.200:32000/rest/wss" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

// Service Class ConfigurationName = **service.name**

<service behaviorConfiguration="svc\_bh" name="mysvc">

// SOAP ServiceEndpoint for Contract: idb, isql

<endpoint address="http://192.168.1.200:32000/rest/db"

binding="basicHttpBinding"

name="ep" contract="WinRestServer.idb" />

<endpoint address="http://192.168.1.200:32000/rest/sql"

binding="basicHttpBinding"

name="ep1" contract="WinRestServer.isql" />

// REST ServiceEndpoint for Contract: idb, isql

<endpoint address="http://192.168.1.200:32000/rest/good"

behaviorConfiguration="ep\_bh"

binding="webHttpBinding" name="wep" contract="WinRestServer.idb" />

<endpoint address="http://192.168.1.200:32000/rest/better"

behaviorConfiguration="ep\_bh"

binding="webHttpBinding" name="wep1" contract="WinRestServer.isql" />

<host>

<baseAddresses>

<add baseAddress="http://192.168.1.200/rest" />

</baseAddresses>

</host>

</service>

</services>

</system.serviceModel>

ServiceHost wsh = new ServiceHost(typeof(mybus));

wsh.Open();

**如果**以 ServiceHost 启动服务， 则SOAP 和 REST 服务都有效。

**对于使用 xxx.svc 的配置文件： Web.config**

namespace wcfRest2

{

[ServiceBehavior( Name = "dbsvc",

ConfigurationName = "mysvc111")]

**//重要的是 ConfigurationName 对应 Web.config**

[AspNetCompatibilityRequirements(RequirementsMode=AspNetCompatibilityRequirementsMode.Allowed)]

public class cls1:idb01,idb02

{

public cls1() { }

public List<people> getPPS() {

using (mydb1 db = new mydb1() ) {

return db.people.ToList();

}

}

public people getPPID(string id) {

using (mydb1 db = new mydb1()) {

int sid = Convert.ToInt32(id);

return db.people.Include("street").Where(pp => pp.id == sid ).SingleOrDefault();

}

}

public string insPP(people p) {

using (mydb1 db = new mydb1()) {

if (p != null) {

db.people.AddObject(p);

db.SaveChanges();

}

}

if (p != null)

return p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "p is null";

}

public string updPP(people p) {

using (mydb1 db = new mydb1()) {

if (p != null)

return "update: " + p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "update: p is null";

}

}

public string delPP(people p) {

using (mydb1 db = new mydb1()) {

if (p != null)

return "Delete: " + p.id + "||" + p.first\_name + "||" + p.last\_name + "||" + p.country;

else

return "delete: p is null";

}

}

}

[ServiceContract]

public interface idb01 {

[OperationContract]

[WebGet( UriTemplate = "pps",

BodyStyle=WebMessageBodyStyle.Wrapped,

RequestFormat=WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json)]

List<people> getPPS();

[OperationContract]

[WebGet( UriTemplate = "get(id=lwh{id})", ResponseFormat = WebMessageFormat.Json)]

people getPPID(string iD);

}

[ServiceContract]

public interface idb02

{

[OperationContract]

[WebInvoke( UriTemplate = "ins",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string insPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "updp",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "PUT")]

string updPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "delp",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "DELETE")]

string delPP(people p);

}

}

public interface idb01

public interface idb02

**Web.config 的解析：**

<system.serviceModel>

<behaviors>

**//只能通过下面的 endpointBehaviors. behavior. webHttp 来设置帮助页面**

<endpointBehaviors>

<behavior name="ep\_bh">

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />

</behavior>

</endpointBehaviors>

</behaviors>

**// 对于多服务契约的情况， standardEndpiont 的设置不起作用。 主要是帮助页面**

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint automaticFormatSelectionEnabled="true" helpEnabled="true" />

</webHttpEndpoint>

</standardEndpoints>

<services>

**// REST WCF 的服务端点配置多个服务契约,**

**//因为使用的是 xxx.svc 所以地址必须使用相对地址**

**//而绝对的 Uri 是 xxx.svc + 相对地址：** [**http://192.168.1.200/wcfrest/rest2.svc/**](http://192.168.1.200/wcfrest/rest2.svc/) **+ wsss 和 + wxxx**

**//** behaviorConfiguration **可以使用同一个端点行为来设置帮助页面**

<service name="mysvc111"> **// name 对应服务类的 ConfigurationName**

<endpoint address="wsss" binding="webHttpBinding" behaviorConfiguration="ep\_bh" contract="wcfRest2.idb01" />

<endpoint address="wxxx" binding="webHttpBinding" behaviorConfiguration="ep\_bh" contract="wcfRest2.idb02" />

</service>

</services>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true"

multipleSiteBindingsEnabled="true" />

</system.serviceModel>

[**http://192.168.1.200/wcfrest/rest2.svc**](http://192.168.1.200/wcfrest/rest2.svc)

**rest2.svc 文件内容：**

<%@ ServiceHost Language="C#" Debug="true"

factory="System.ServiceModel.Activation.WebServiceHostFactory"

Service="wcfRest2.cls1"

CodeBehind="cls1.cs" %>

**rest2.svc 同时有 SOAP 和 REST 服务：**

<system.serviceModel>

<behaviors>

<endpointBehaviors>

<behavior name="ep\_bh">

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />

</behavior>

</endpointBehaviors>

<serviceBehaviors>

**// 对应 service.behaviorConfiguration=”svc\_bh”，wsdl 好像不起作用**

<behavior name="svc\_bh">

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="mysvc111" behaviorConfiguration="svc\_bh">

**// SOAP 服务端点多服务契约**

<endpoint address="sap1" binding="basicHttpBinding" contract="wcfRest2.idb01" />

<endpoint address="sap2" binding="basicHttpBinding" contract="wcfRest2.idb02" />

**// mex 地址必须设置， 否则无法在客户端创建引用**

<endpoint address="mexx" binding="mexHttpBinding" name="mex\_name" contract="IMetadataExchange" />

**// REST 服务端点多服务契约**

<endpoint address="wsss" binding="webHttpBinding" behaviorConfiguration="ep\_bh" contract="wcfRest2.idb01" />

<endpoint address="wxxx" binding="webHttpBinding" behaviorConfiguration="ep\_bh" contract="wcfRest2.idb02" />

</service>

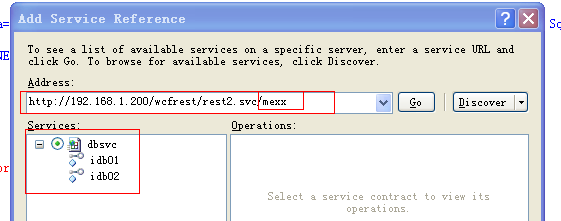
</services>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true"

multipleSiteBindingsEnabled="true" />

</system.serviceModel>

**地址都是使用相对地址， 基地址是 xxx.svc 的绝对地址 + 这些相对地址**

****

**特别的问题： PUT , DELETE 不工作出现：**HTTP 错误 405.0 - Method Not Allowed

问题：  
 使用WCF 4.0 开发REST服务，部署到windows 7 64位 IIS 7.5 （应用程序池选择的是ASP.NET V4.0） 上GET，POST方式都能成功；但是PUT,DELETE报错：  
HTTP 错误 405.0 - Method Not Allowed  
无法显示您正在查找的页面，因为使用了无效方法(HTTP 谓词)。

但是如果使用VS2010 调试模式，GET，POST,PUT,DELETE都成功，这说明代码本身没有任何问题，  
问题出在IIS 7.5 设置上。

在显示的错误页面上：可以看到如下信息，知道处理模块为WebDAVModule  
模块  
WebDAVModule  
通知  
MapRequestHandler  
处理程序  
StaticFile  
错误代码  
0x00000000  
然后查看Headers  发现允许的谓词如下： Allow: GET, HEAD, OPTIONS, TRACE

解决办法：  
设置IIS 7.5 -》 处理程序映射  
WebDAV-》请求限制-》谓词 勾选“全部谓词”

后来我由搜到一篇博客，你也可以尝试如下方法解决，我没有测试过是否可行：  
These days I run into this issue: my restful service hosting with IIS 7.5 and using Asp.net 4.0 not support PUT or DELETE verbs for http request.  
solved.  
the following configuration setting fixed our problem:  
     <modules runAllManagedModulesForAllRequests="true">  
        <remove name="WebDAVModule" />  
     </modules>  
As reported in [1], this is due to the WebDAV module now being an integral part of IIS 7.5 (and on 7.0 if installed). Removing the module allowed all 4 verbs to come through to our service tier. Cheers!

<system.webServer>   
  <modules … >   
    <remove name="WebDAVModule" />   
    ……   
  </modules>

**解决方法真的有效，神奇了！！**

<configuration>

<system.webServer>

<modules runAllManagedModulesForAllRequests="true">

<remove name="WebDAVModule" />

</modules>

</system.webServer>

…………..

</configuration>

**POST, PUT, DELETE 使用相同的 URI**

[ServiceContract]

public interface idb02

{

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Json, ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string insPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Json, ResponseFormat = WebMessageFormat.Json, Method = "PUT")]

string updPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Json, ResponseFormat = WebMessageFormat.Json, Method = "DELETE")]

string delPP(people p);

}

都使用相对地址 “op”

**我们在客户端调用：**

function add\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "POST",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

});

}

function upd\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "PUT",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

});

}

function del\_pp() {

var newp = {};

newp.id = "299";

newp.first\_name = "Jackson";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "json",

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "DELETE",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

});

}

**我们看到唯一不同就是 谓语动词：type: “POST”, “DELETE”, “PUT” 分别对应相应的操作。**

**以下讨论 WCF Rest 的数据格式问题：**

我们知道关于数据格式在 WCF Rest 有多处进行设置：

第一处：Web.config 或者 App.config

<endpointBehaviors>

<behavior name="ep\_bh">

<webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />

</behavior>

</endpointBehaviors>

<standardEndpoints>

<webHttpEndpoint>

<standardEndpoint automaticFormatSelectionEnabled="true" helpEnabled="true" />

</webHttpEndpoint>

</standardEndpoints>

默认自动格式选择是关闭的。

第二处：[WebGet] [WebInvoke]

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string insPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "PUT")]

string updPP(people p);

[OperationContract]

[WebInvoke( UriTemplate = "op",

RequestFormat = WebMessageFormat.Xml,

ResponseFormat = WebMessageFormat.Xml, Method = "DELETE")]

string delPP(people p);

第三处： 客户端调用时指定格式:

function upd\_pp() {

var newp = {};

newp.id = 145;

newp.first\_name = "Unjsoan";

newp.last\_name = "Json";

newp.country = "JSON";

$.ajax({

data: JSON.stringify(newp),

dataType: "html", //可以是 xml, html, json, script, text。对应的是返回的信息格式：Accept

contentType: "application/json; charset=utf-8", //可以是Application/XML, 对应发送的消息格式

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "PUT",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

});

}

dataType : 对应放回的数据格式. 即 Accept

contentType: 对应的是发送的消息格式：即 content-type

**总结:**

1. 优先级别来说： 客户端的设置 优先于 服务契约的设置
2. 对于客户端来说，

不设置放回的消息格式，则返回的消息格式等于发送的消息格式。 但是如果不设置发送的消息格式

$.ajax({

data: JSON.stringify(newp),

// dataType: "html", 不设置返回消息的格式。

contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

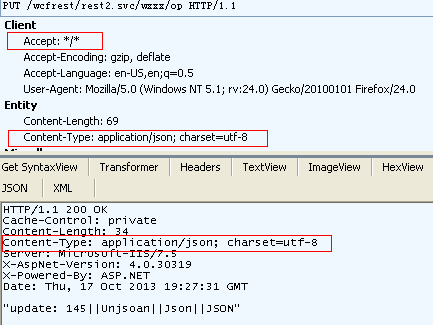
success: function (req, tStatus) { alert(req); },

type: "PUT",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

});

**可以看到 返回消息的格式 继承发送消息的格式。**

****

**如果我们设置返回消息的格式， 不设置发送消息的格式：**

$.ajax({

data: JSON.stringify(newp),

dataType: "xml",

//contentType: "application/json; charset=utf-8",

error: function (xhr, tStatus, errorTh) {

alert("Error (fform\_save.php): " + xhr.responseText + "\nStatus: " + tStatus);

},

success: function (req, tStatus) {

alert(req);

},

type: "DELETE",

url: "http://192.168.1.200/wcfrest/rest2.svc/wxxx/op"

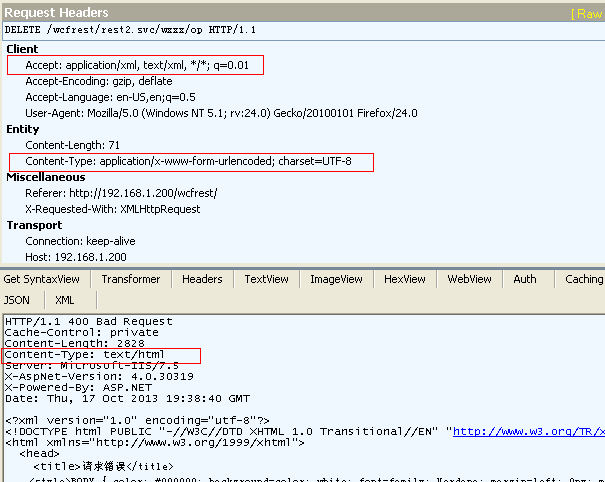
});

如下图：

则发送消息的格式是默认的：Content-Type: application/x-www-form-urlencoded; charset=UTF-8

返回消息的格式是设置的： Accept: application/xml, text/xml, \*/\*; q=0.01

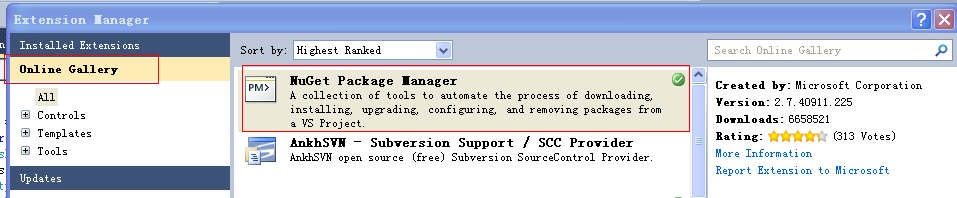
但是服务器端返回的格式是：Content-Type: text/html 也并非按照WebInvoke 设置的 JSON 格式放回。估计是请求出错的问题。

****

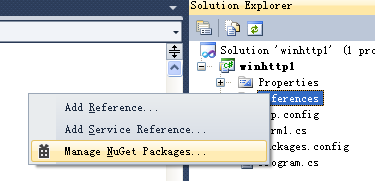
**对于C# 的客户端 通常使用 HttpClient 来调用WCF Rest . HttpClient 是.Net4.5 内置的对象。 而.Net4.0 需要安装。**

HttpClient的安装：

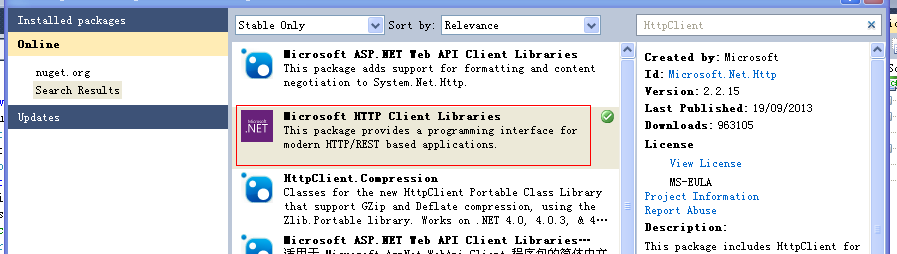
1. 在 VS2010 -> Tools -> Extension Manager 下找到 Online Gallery



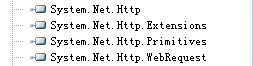
1. 安装完以后，对着 Solution 或者 Project 或者 Reference 鼠标右键，会出现



1. 查找到 HttpClient library



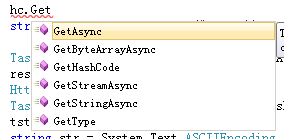
1. 安装以后，会自动引用以下组件：



MSDN HttpClient 的技术资料：

<http://msdn.microsoft.com/en-us/library/system.net.http.httpclient.aspx>

**HttpClient – Get 操作：只有Uri , 没有HttpContent, 因为get 传递的参数是属于Uri 的一部分。**



GetAsync - [Task](http://msdn.microsoft.com/en-us/library/dd321424.aspx)<[HttpResponseMessage](http://msdn.microsoft.com/en-us/library/system.net.http.httpresponsemessage.aspx)> [System.Threading.Tasks](http://msdn.microsoft.com/en-us/library/dd321424.aspx)

GetAsync( string, CancellationToken );

GetAsync( Uri, CancellationToken );

GetAsync( string, HttpCompletionOption);

GetAsync( Uri, HttpCompletionOption);

对于 WCF 服务在服务契约方法上可以操作进出的信息：WebOperationContext.Current

WebOperationContext woc = WebOperationContext.Current;

例如: 人工干预返回的状态码：

WebOperationContext woc = WebOperationContext.Current;

woc.OutgoingResponse.StatusCode = System.Net.HttpStatusCode.Conflict;

woc.OutgoingResponse.StatusDescription = "bad bad good good better";

状态对于 JQuery Ajax 有严重的影响。

例如： 人为设置 StatusCode = Conflict

public List<people> getPPS()

{

WebOperationContext woc = WebOperationContext.Current;

woc.OutgoingResponse.StatusCode = System.Net.HttpStatusCode.Conflict;

woc.OutgoingResponse.StatusDescription = "bad bad good good better";

using (mydb1 db = new mydb1()) {

return db.people.ToList();

}

}

则客户端的 JQuery Ajax:

$.ajax({

data: { },

dataType: "json",

error: function(xhr, tStatus, errorTh ) {

alert("Error From Rest: " + xhr.responseText + "\nStatus: " + tStatus + "\nMessage:" + errorTh);

},

success: function(req, tStatus) {

toHTML(req);

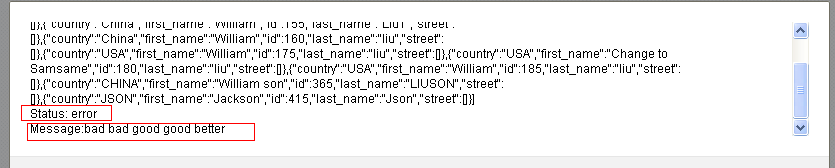
},

type: "GET",

url: "http://192.168.1.200/wcfrest/rest2.svc/wsss/pps"

});

则直接呼叫错误处理事件：Alert



**在客户端设置返回信息的格式， 是优先级别最高的。**

hc = new HttpClient();

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/xml"));

hc = new HttpClient();

// GetAsync

string url = string.Format("http://192.168.1.200/wcfrest/rest2.svc/wsss/pps");

Task<HttpResponseMessage> resTask = hc.GetAsync(url, HttpCompletionOption.ResponseHeadersRead);

this.listBox1.Items.Add("Status:" + resTask.Status);

resTask.Wait();

this.listBox1.Items.Add("Status:" + resTask.Status);

// 获取 HttpResponseMessage

HttpResponseMessage resMsg = resTask.Result;

this.listBox1.Items.Add( string.Format("RespMsg:{0}-{1}-{2}",

resMsg.StatusCode, // 获取错误代码: StatusCode

resMsg.ReasonPhrase, // 获取错误信息：StatusDescription

resMsg.IsSuccessStatusCode ));

//可以通过 **IsSuccessStatusCode** 来判断是否成功

// 这是服务端的设置错误的代码， 对应上面的属性

WebOperationContext woc = WebOperationContext.Current;

woc.OutgoingResponse.StatusCode = System.Net.HttpStatusCode. Conflict;

woc.OutgoingResponse.StatusDescription = "bad bad good good better";

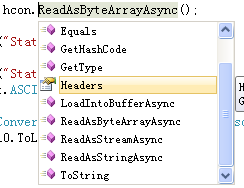
// 以上是服务端设置错误的代码.

// 获取相关的信息

HttpResponseHeaders hd = resMsg.Headers;

HttpRequestHeaders hreqh = resMsg.RequestMessage.Headers;

//获取HttpContent

HttpContent hcon = resMsg.Content;

Task<byte[]> readTask = hcon.ReadAsByteArrayAsync();

可以看到右图， 有多种读取的方法: 以下直接读成字符串

Task<string> readTask = hcon.ReadAsStringAsync();

readTask.Wait();

string str = readTask.Result;

this.listBox1.Items.Add("Status:" + readTask.Status);

readTask.Wait();

this.listBox1.Items.Add("Status:" + readTask.Status);

string str = System.Text.ASCIIEncoding.ASCII.GetString(readTask.Result);

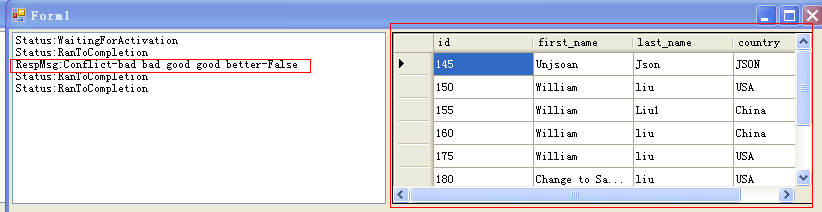
List<person> pl0 = JsonConvert.DeserializeObject<List<person>>(str);

this.dgv.DataSource = pl0.ToList();

输出结果如下图：

虽然我们在服务端人为设置了错误，但是不影响我们提取正确的返回信息。

我们可以看到输出的状态，以及自定义的错误代码。



Uri 带参数的Get:

[OperationContract]

[WebGet(UriTemplate = "get(id=lwh{id})", ResponseFormat = WebMessageFormat.Json)]

people getPPID(string id);

--------------------------------------------------------------------------------------------------------------------------------

public people getPPID(string id) // 对于以上的非传统Uri， 参数必须是string

{

using (mydb1 db = new mydb1())

{

int sid = Convert.ToInt32(id);

return db.people.Where(pp => pp.id == sid ).SingleOrDefault();

}

}

----------------------------------------------------------------------------------------------------------------------------------

特别注意：

1. 我们使用的 URI 的形式可以多样化：非传统模式的参数必须是 string 类型，否则出错。
2. 如果是使用传统模式， 则参数可以是其他简单类型如：int , DateTime 等

传统方式就是： get? id={id}&name={name}&cc={country}

[OperationContract]

[WebGet(UriTemplate = "get?id={id}", ResponseFormat = WebMessageFormat.Json)]

people getPPID(**int** id); //传统方式可以使用 int 类型。

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wsss/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

string str1 = hc.GetAsync("get(id=lwh145)").Result.Content.ReadAsStringAsync().Result;

person pp = JsonConvert.DeserializeObject<person>(str1); //字符串JSON 反序列化回对象。

this.listBox1.Items.Add(string.Format("{0}:{1}:{2}", pp.id, pp.first\_name, pp.country));

**HttpClient – Put 操作：有 Uri, 也有 HttpContent**

使用 XML 格式传递：前提我们必须获得 XML Schema 的定义。

我们必须借助帮助页面，获取服务传输信息的Schema:

****



建议在调用服务前，先查看帮助页面，查看传递信息的Schema，包括JSON, XML 格式。

**PUT XML:**

hc = new HttpClient();

//这个属性重要，是客户端来决定返回信息的格式。

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/xml"));

hc.DefaultRequestHeaders.AcceptLanguage.Add(new StringWithQualityHeaderValue("cn-zh"));

hc.DefaultRequestHeaders.AcceptEncoding.Add(new StringWithQualityHeaderValue("gzip"));

string str2 = @"

<people xmlns='http://schemas.datacontract.org/2004/07/wcfRest2'> // XML 格式必须和 Schema 严格相同

<country>China</country> // 包括字段的顺序和个数必须严格相同

<first\_name>Hello world</first\_name>

<id>145</id>

<last\_name>Good Last</last\_name>

</people>";

决定发送消息格式的 Content-Type 由以下决定：

StringContent str111 = new StringContent( str2 , Encoding.UTF8, "text/xml");

Task<HttpResponseMessage> res1 = hc.PutAsync("http://192.168.1.200/wcfrest/rest2.svc/wxxx/op",str111);

Task<string> str3 = res1.Result.Content.ReadAsStringAsync();

this.label1.Text = str3.Result;

如果我将顺序改变：

string str2 = @"

<people xmlns='http://schemas.datacontract.org/2004/07/wcfRest2'>

<id>145</id>

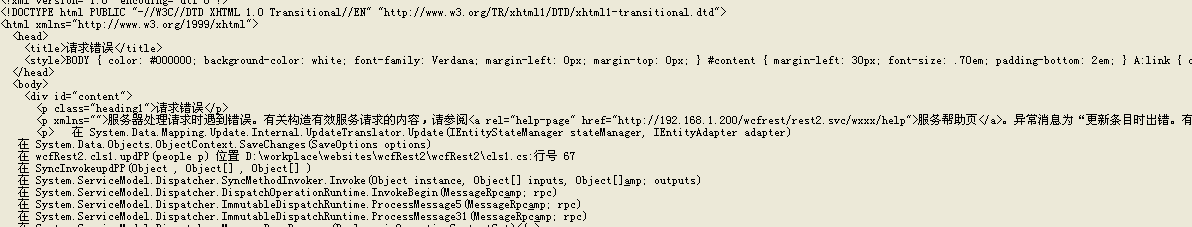
<first\_name>Hello world</first\_name>

<last\_name>Good Last</last\_name>

<country>China</country>

</people>";

则收到错误信息：



使用精简语句：

hc = new HttpClient();

// 设置基地址， 以后操作就可以使用相对地址了

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wxxx/");

// 设置返回格式： JSON

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

string str2 = @"

<people xmlns='http://schemas.datacontract.org/2004/07/wcfRest2'>

<country>China</country>

<first\_name>Hello world</first\_name>

<id>145</id>

<last\_name>Good Last</last\_name>

</people>";

StringContent strXML = new StringContent( str2 , Encoding.UTF8, "text/xml");

// 使用相对地址, 使用多点操作多个方法。让结果一步到位

string str3 = hc.PutAsync("op", strXML).Result.Content.ReadAsStringAsync().Result;

this.label1.Text = str3;

注意相对地址的运算：

<http://192.168.1.200/wcfrest/rest2.svc/wxxx> + op

等于：http://192.168.1.200/wcfrest/rest2.svc/op

<http://192.168.1.200/wcfrest/rest2.svc/wxxx>/ + /op

等于：/op

<http://192.168.1.200/wcfrest/rest2.svc/wxxx> + ./op

等于：http://192.168.1.200/wcfrest/rest2.svc/op

<http://192.168.1.200/wcfrest/rest2.svc/wxxx>/ + ./op

等于：http://192.168.1.200/wcfrest/rest2.svc/wxxx/op

<http://192.168.1.200/wcfrest/rest2.svc/wxxx>/ + ../op

等于：http://192.168.1.200/wcfrest/rest2.svc/op

**HttpContent：有三种**

1. **StringContent**
2. **ByteArrayContent**
3. **StreamContent**

**传递JSON格式：**

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wxxx/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

person uppp = new person { id = 145, first\_name = "William", last\_name = "Liu",country="USA" };

JObject jppp = JObject.FromObject(uppp);

StringContent strJSON = new StringContent(jppp.ToString(), Encoding.UTF8, "application/json");

string str3 = hc.PutAsync("op", strJSON).Result.Content.ReadAsStringAsync().Result;

this.label1.Text = str3;

注意：

对于JSON 格式，字段顺序无所谓， 但是字段名称和个数必须一致。个数可以多出schema 定义的。

**HttpClient – Post 操作：有 Uri, 也有 HttpContent**

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wxxx/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

person uppp = new person { id = 145,

first\_name = "William" + DateTime.Now.ToLongTimeString(),

last\_name = "Liu",

country="USA" };

JObject jppp = JObject.FromObject(uppp);

StringContent strJSON = new StringContent(jppp.ToString(), Encoding.UTF8, "application/json");

string str3 = hc.PostAsync("op", strJSON).Result.Content.ReadAsStringAsync().Result;

this.label1.Text = str3;

**HttpClient – Delete 操作：有 Uri, 没有 HttpContent**

[OperationContract]

[WebInvoke( UriTemplate = "op(id={pid})",

BodyStyle = WebMessageBodyStyle.Bare,

RequestFormat = WebMessageFormat.Xml,

ResponseFormat = WebMessageFormat.Xml, Method = "DELETE")]

string delPP(string pid);

注意: 非传统的URI , ｛参数｝ 必须对应方法里的参数， 而且参数类型必须是string, 否则整个服务出错。

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wxxx/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

string str3 = hc.DeleteAsync("op(id=420)").Result.Content.ReadAsStringAsync().Result;

this.label1.Text = str3;

**HttpClient – Send 操作：有** HttpRequestMessage**, 没有 Uri , HttpContent**

SendAsync(HttpRequestMessage);

SendAsync(HttpRequestMessage, HttpCompletionOption)

SendAsync(HttpRequestMessage, HttpCompletionOption, cancellationToken)

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wxxx/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/html"));

string str2 = @"

<people xmlns='http://schemas.datacontract.org/2004/07/wcfRest2'>

<country>China</country>

<first\_name>Hello" + DateTime.Now.ToLongTimeString() + @"</first\_name>

<id>145</id>

<last\_name>Good Last</last\_name>

</people>";

HttpRequestMessage hrm = new HttpRequestMessage(HttpMethod.Put, new Uri("op(id=415)", UriKind.Relative));

// 可以事后更改

HttpRequestMessage hrm = new HttpRequestMessage()；

hrm.Method = HttpMethod.Post;

hrm.RequestUri = new Uri("op",UriKind.Relative);

// 设置返回的格式

hrm.Headers.Clear();

hrm.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("text/html"));

hrm.Headers.AcceptCharset.Add(new StringWithQualityHeaderValue("ASCII"));

hrm.Headers.AcceptEncoding.Add(new StringWithQualityHeaderValue("WinZip"));

//设置其他头信息

hrm.Headers.Add("Cache-Control", "No");

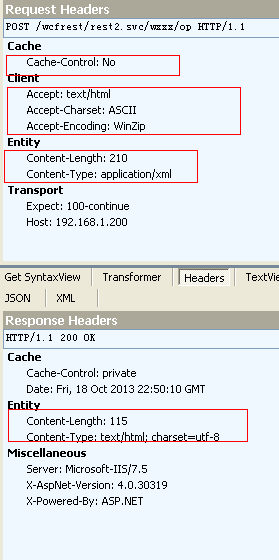
//设置发送信息的格式

hrm.Content = new StringContent(str2, Encoding.UTF8, "application/json");

//必须有内容以后才能设置

hrm.Content.Headers.ContentType = new MediaTypeHeaderValue("application/xml");

//hrm.Content.Headers.ContentLength = 300090; 不能乱输入数据.

string tMsg = hc.**SendAsync(hrm)**.Result.Content.ReadAsStringAsync().Result;

this.label1.Text = tMsg;

使用JSON格式:

person uppp = new person { id = 145,

first\_name = "William" + DateTime.Now.ToLongTimeString(),

last\_name = "Liu", country = "USA" };

JObject jppp = JObject.FromObject(uppp);

hrm.Content = new StringContent( jppp.ToString(),

Encoding.UTF8,

"application/json");

hrm.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

string tMsg = hc.SendAsync(hrm).Result.Content.ReadAsStringAsync().Result;

**WCF REST for raw Stream**

**下载文件: WebGet**

[OperationContract]

[WebGet(UriTemplate="image(name={imgname})")]

Stream getImage(string imgname);

public Stream getImage(string imgname)

{

string imageType = Path.GetExtension(imgname).TrimStart('.');

//指定放回信息的格式,是可选的. 可以不指定也行

WebOperationContext.Current.OutgoingResponse.ContentType = "image/" + imageType;

var dir = System.Web.HttpContext.Current.Server.MapPath("~/images");

var file = Path.Combine(dir, imgname); //打开文件必须获得文件在服务器的绝对路径.

FileStream fs = File.OpenRead(file);

return fs;

}

**在客户端: JQuery 赋予uri 即可**

var uri = "http://192.168.1.200/wcfrest/rest2.svc/wsss/image(name=winter.jpg)";

$("#myImg").attr("src", uri);

<img id="myImg" src="" width="300" height="200" />

如果想把文件直接下载:

public Stream getImage(string imgname)

{

string imageType = Path.GetExtension(imgname).TrimStart('.');

WebOperationContext.Current.OutgoingResponse.ContentType = "image/" + imageType;

// 下载时候的文件名称设置

WebOperationContext.Current.OutgoingResponse.Headers.Add("Content-disposition", "attachment; filename=winwin.jpg");

var dir = System.Web.HttpContext.Current.Server.MapPath("~/images");

var file = Path.Combine(dir, imgname);

FileStream fs = File.OpenRead(file);

return fs;

}

客户端:

var uri = "http://192.168.1.200/wcfrest/rest2.svc/wsss/image(name=winter.jpg)";

window.location.href = uri;

**在C# 的客户端:**

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wsss/");

// 这个不会影响到实际放回的类型

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/html"));

Stream fso = hc.GetAsync("image(name=winter.jpg)").Result.Content.ReadAsStreamAsync().Result;

this.pbx.Image = Image.FromStream(fso);

**上传文件: WebInvoke POST**

[OperationContract]

[WebInvoke(UriTemplate = "image1", Method = "POST")]

string addImage(Stream t);

注意: 1) 方法参数必须只有一个, 是 Stream 类型, 不能带有其他的类型

[OperationContract]

[WebInvoke(UriTemplate = "image1/{image\_name}", Method = "POST")]

string addImage(Stream t, string image\_name);

以上想将 image\_name 当作URI 的一部分传递是错误的. 虽然网上很多这样的例子

2) 可以有返回值.

3) 要想传递文件名称, 只能想其他办法, 例如可以通过 头信息 来传递, 下面例子会讲到.

配置文件的设置很重要:

<bindings>

<webHttpBinding>

<binding name ="rest\_binding"

transferMode="Streamed" //传输模式: 可以是 Buffered, Streamed

maxBufferPoolSize="30000000" // 这个不重要. 可以设置很小的值如: 1024.

maxReceivedMessageSize="30000000" />

// maxReceivedMessageSize这个非常重要, 如果太小, 容纳不下传过来信息的大小, 出现 413 错误:

</webHttpBinding>

</bindings>

详细错误信息如下:

HTTP/1.1 413 Request Entity Too Large

Cache-Control: private

Server: Microsoft-IIS/7.5

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Date: Sun, 20 Oct 2013 21:23:09 GMT

Content-Length: 0

<bindings>

<webHttpBinding>

<binding name ="rest\_binding"

transferMode="Streamed"

maxBufferPoolSize="1024"

maxBufferSize="100000" //**必须和下面的**maxReceivedMessageSize**一致**

maxReceivedMessageSize="30000" />

</webHttpBinding>

</bindings>

**注意:**  maxBufferSize 和 maxReceivedMessageSize 必须相同, 否则服务会出错, 不管传输模式是:Buffered 还是 Streamed

服务类的方法:

public string addImage(Stream t)

{

var dir = HttpContext.Current.Server.MapPath("~/images");

// HttpContext.Current.Request.Headers 获取请求的头信息 此处可以输出 All Keys

var sss = string.Join(",", HttpContext.Current.Request.Headers.AllKeys);

// 获取头信息的键值.

var imgName = HttpContext.Current.Request.Headers.Get("myfile");

// 保存文件,需要服务器的文件系统的物理地址.

var file = Path.Combine(dir, imgName);

var bitmap = Bitmap.FromStream(t);

bitmap.Save(file);

return "Save to:" + " : " +imgName;

}

客户端的上传代码:

OpenFileDialog fd = new OpenFileDialog();

DialogResult rs = fd.ShowDialog();

if (rs == DialogResult.OK) {

var file = fd.FileName;

var name = Path.GetFileName(file);

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wsss/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

// 生成 StreamContent, 并设置内容的头信息

StreamContent scon = new StreamContent( File.OpenRead(file) );

scon.Headers.Add("myfile", name);

// 提交上传的流

string restr = hc.PostAsync("image1", scon).Result.Content.ReadAsStringAsync().Result;

this.label1.Text = restr;

}

关于WebInvoke POST 的几点说明:

1. 以下是合法的: 把 所有参数写在 URI 里

[OperationContract]

[WebInvoke(UriTemplate = "sumer/{a}+{b}", Method = "POST")]

string sumup(string a, string b);

1. 以下是错误的做法: URI 没有参数, 方法有两个参数

[OperationContract]

[WebInvoke(UriTemplate = "sumer", Method = "POST")]

string sumup(string a, string b);

1. 以下做法是可以的,. URI , 参数都存在

[OperationContract]

[WebInvoke( UriTemplate = "sumer/{a}",

RequestFormat=WebMessageFormat.Json,

ResponseFormat = WebMessageFormat.Json, Method = "POST")]

string sumup(string a, string b);

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wsss/");

hc.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

string xmlStr = @"<string xmlns='http://schemas.microsoft.com/2003/10/Serialization/'>Hello world</string>";

StringContent scon = new StringContent(xmlStr, Encoding.UTF8, "text/xml");

string restr = hc.PostAsync("sumer/william", scon).Result.Content.ReadAsStringAsync().Result;

this.label1.Text = restr;

如果是配合 Stream 则不行, 服务无法启动: 以下是会导致服务启动出错.

[OperationContract]

[WebInvoke(UriTemplate = "image1/{a}", Method = "POST")]

string addImage(Stream t, string a);

1. 以下做法也是可以的:

[OperationContract]

[WebInvoke(UriTemplate = "sumer", Method = "POST")]

string sumup(string a);

// 以下是请求代码:

hc = new HttpClient();

hc.BaseAddress = new Uri("http://192.168.1.200/wcfrest/rest2.svc/wsss/");

string xmlStr = @"<string xmlns='http://schemas.microsoft.com/2003/10/Serialization/'>dkfasdk sdkfjdk</string>";

StringContent scon = new StringContent(xmlStr, Encoding.UTF8, "text/xml");

string restr = hc.PostAsync("sumer", scon).Result.Content.ReadAsStringAsync().Result;

如何使用JSON 来请求, 有待研究. 通常简单类型如 string , 最好作为 URI 的参数来传递.

Json.Net

常用的函数:

* SerializeObject , DeserializeObject

List<pson> plist = new List<pson>();

plist.Add(new pson { id = 100, name = "User1", school = "Mar1" });

plist.Add(new pson { id = 101, name = "User2", school = "Mar2" });

plist.Add(new pson { id = 102, name = "User3", school = "Mar3" });

plist.Add(new pson { id = 103, name = "刘¢?委¡¥会¨¢", school = "第Ì¨²一°?小?学¡ì" });

plist.Add(new pson { id = 104, name = "User4", school = "中D国¨²大ä¨®学¡ì" });

string jstr = JsonConvert.SerializeObject(plist);

this.label1.Text = jstr;

List<pson> ppp = JsonConvert.DeserializeObject<List<pson>>(jstr);

string str = null;

foreach (pson el in ppp)

{

str += String.Format("\nID:{0} Name:{1} School:{2}", el.id, el.name, el.school);

}

this.rtb.Text = str;

* JObject.FromObject(object) , JObject.ToString()

pson p1 = new pson{id = 200, name="William Liu", school="Zhengzhou"};

JObject jo1 = JObject.FromObject(p1);

this.rtb.Text = jo1.ToString();

* JObject.Parse( string );

string jstr1 = @"{id:105, name:'William Liu', city:'Burnaby', state:'BC'}";

JObject jo2 = JObject.Parse(jstr1);

this.rtb.Text = jo2.ToString();

* JObject.Add(JProperty);

string jstr1 = @"{id:105, name:'William Liu', city:'Burnaby', state:'BC'}";

JObject jo2 = JObject.Parse(jstr1);

jo2.Add(new JProperty("country", "China"));

jo2.Add(new JProperty("unit", "Someone"));

* JObject() Add(JProperty) JArray

JObject jo2 = new JObject();

jo2.Add(new JProperty("country", "China"));

jo2.Add(new JProperty("unit", "Someone"));

jo2.Add(new JProperty("score", new JArray(100, 200, 300)) );

JObject jo2 = new JObject();

jo2["fname"] = "William";

jo2["others"] = new JArray("One", "Two", "Three");

this.rtb.Text =jo2["others"].ToString();

输出: [“One”, “Two”, “Three”];

**三.使用Linq to JSON**

**1.查询**  
首先准备Json字符串，是一个包含员工基本信息的Json

string json = "{\"Name\" : \"Jack\", \"Age\" : 34, \"Colleagues\" : [{\"Name\" : \"Tom\" , \"Age\":44},{\"Name\" : \"Abel\",\"Age\":29}] }";

①获取该员工的姓名

//将json转换为JObject

JObject jObj = JObject.Parse(json);

//通过属性名或者索引来访问，仅仅是自己的属性名，而不是所有的

JToken ageToken = jObj["Age"];

Console.WriteLine(ageToken.ToString());

②获取该员工同事的所有姓名

//将json转换为JObject

JObject jObj = JObject.Parse(json);

var names=from staff in jObj["Colleagues"].Children()

select (string)staff["Name"];

foreach (var name in names)

Console.WriteLine(name);

"Children()"可以返回所有数组中的对象

**2.修改**

①现在我们发现获取的json字符串中Jack的年龄应该为35

//将json转换为JObject

JObject jObj = JObject.Parse(json);

jObj["Age"] = 35;

Console.WriteLine(jObj.ToString());

注意不要通过以下方式来修改：

JObject jObj = JObject.Parse(json);

JToken age = jObj["Age"];

age = 35;

②现在我们发现Jack的同事Tom的年龄错了，应该为45

//将json转换为JObject

JObject jObj = JObject.Parse(json);

JToken colleagues = jObj["Colleagues"];

colleagues[0]["Age"] = 45;

jObj["Colleagues"] = colleagues;//修改后，再赋给对象

Console.WriteLine(jObj.ToString());

**3.删除**  
①现在我们想删除Jack的同事

JObject jObj = JObject.Parse(json);

jObj.Remove("Colleagues");//跟的是属性名称

Console.WriteLine(jObj.ToString());

②现在我们发现Abel不是Jack的同事，要求从中删除

JObject jObj = JObject.Parse(json);

jObj["Colleagues"][1].Remove();

Console.WriteLine(jObj.ToString());

**4.添加**  
①我们发现Jack的信息中少了部门信息，要求我们必须添加在Age的后面

//将json转换为JObject

JObject jObj = JObject.Parse(json);

jObj["Age"].Parent.AddAfterSelf(new JProperty("Department", "Personnel Department"));

Console.WriteLine(jObj.ToString());

②现在我们又发现，Jack公司来了一个新同事Linda

//将json转换为JObject

JObject jObj = JObject.Parse(json);

JObject linda = new JObject(new JProperty("Name", "Linda"), new JProperty("Age", "23"));

jObj["Colleagues"].Last.AddAfterSelf(linda);

Console.WriteLine(jObj.ToString());

**四.简化查询语句**

使用函数SelectToken可以简化查询语句，具体：  
①利用SelectToken来查询名称

JObject jObj = JObject.Parse(json);

JToken name = jObj.SelectToken("Name");

Console.WriteLine(name.ToString());

②利用SelectToken来查询所有同事的名字

JObject jObj = JObject.Parse(json);

var names = jObj.SelectToken("Colleagues").Select(p => p["Name"]).ToList();

foreach (var name in names)

Console.WriteLine(name.ToString());

③查询最后一名同事的年龄

//将json转换为JObject

JObject jObj = JObject.Parse(json);

var age = jObj.SelectToken("Colleagues[1].Age");

Console.WriteLine(age.ToString());

***二.为什么要用Json.Net?***

*我们知道在.Net中内置了读写Json的对象(****DataContractJsonSerializer,JavaScriptSerializer)****,为什么我们还要用Json.Net呢?  
　　功能对比：*

|  |  |  |  |
| --- | --- | --- | --- |
| 特性 | Json.Net | DataContractJsonSerializer | JavaScriptSerializer |
| Json | 支持 | 支持 | 支持 |
| Bson | 支持 | 不支持 | 不支持 |
| Json Schema | 支持 | 不支持 | 不支持 |
| .Net 2.0 | 支持 | 不支持 | 不支持 |
| .Net 3.5 | 支持 | 支持 | 支持 |
| .Net 4.0 | 支持 | 支持 | 支持 |
| Silverlight | 支持 | 支持 | 不支持 |
| Windows Phone | 支持 | 支持 | 不支持 |
| Windows 8 Metro | 支持 | 支持 | 不支持 |
| Linq to Json | 支持 | 不支持 | 不支持 |
| Indented Json(有换行格式的Json) | 支持 | 不支持 | 不支持 |
| Json和XML转换 | 支持 | 不支持 | 不支持 |
| 序列化ＤataTable和DataSet | 支持 | 不支持 | 不支持 |
| 序列化Entity Framework | 支持 | 不支持 | 不支持 |

# 以下内容与Rest 无关

# 纠结了两天的DataContractSerializer 反序列化xml的问题

class Program {

    static void Main(string[] args) {

        //var c = new Contract {

        //    Title = "合同版权",

        //    Copyrights = new List<Copyright> {

        //        new Copyright{WorksName="worksname1"},

        //        new Copyright{WorksName="worksname2"},

        //        new Copyright{WorksName="worksname3"}

        //    }

        //};

        //using (MemoryStream ms = new MemoryStream()) {

        //    var xs = new XmlSerializer(typeof(Contract));

        //    xs.Serialize(ms, c);

        //    Console.WriteLine(Encoding.UTF8.GetString(ms.GetBuffer()));

        //}

        string data = "<Contract><Title> 合同版权</Title><Copyrights><Copyright ><WorksName>青 色平原</WorksName>< /Copyright><Copyright><WorksName>西游记< /WorksName></Copyright></Copyrights></Contract>";

        var xs = new XmlSerializer(typeof(Contract));

        var c = xs.Deserialize(new StringReader(data)) as Contract;

        Console.WriteLine(c.Copyrights.Count);

        foreach(var each in c.Copyrights){

            Console.WriteLine(each.WorksName);

        }

        Console.WriteLine("press any key to exit.");

        Console.ReadLine();

    }

}

[DataContract]

public class Contract {

    [DataMember]

    public string Title { get; set; }

    private List<Copyright> m\_copyrights = new List<Copyright>();

    [DataMember]

    public List<Copyright> Copyrights {

        get { return this.m\_copyrights; }

        set { this.m\_copyrights = value; }

    }

}

//版权类

[DataContract]

public class Copyright {

    [DataMember]

    public string WorksName { get; set; }

}

## **G 类型**

阶乘单步函数如下：

|  |  |
| --- | --- |
| 1 | **g** = λf. λn. (ISZERO n) 1 (MULT n (f (PRED n))) |

c# 中可表达为：g => f => n => n == 0 ? 1 : n \* f(n – 1)

先来推断 f 的类型：

* f 的参数：f 接收 n – 1 作为参数，因此，f 参数的类型和 n – 1 类型相同，即 n 的类型：int；
* f 的返回值：为 0 时返回 1，否则返回 n \* f(n-1)，f 的返回值类型也就是整个递归函数的返回值类型，即 long。

**可确定 f 类型为 Func<int, long>**。

**n => n == 0 ? 1 : n \* f(n – 1)** 是传入一个 int 返回一个 long，其类型 **Func<int, long>**。

先来变换下 g 的表示形式：

|  |  |
| --- | --- |
| 1  2  3  4  5 | var g = (Func<int, long> f) => {  Func<int, long> t =  n => n == 0 ? 1 : n \* f(n - 1);  return t;  }; // 示意代码 |

从上面这段代码可以清楚到看出 g 接收一个 **Func<int, long>** 类型的参数 f，返回一个类型为 **Func<int, long>** 的委托，可得出：

**g 的类型为** **Func<Func<int, long>,** **Func<int, long>>**