WebService与web api是微软出的2个产品，webService用SOAP协议实现的，Web Api用的RESTful风格。它们是两种协议下的web服务实现

WCF既可以实现REST风格也可以实现SOAP风格，而且支持分布式。

# WebService

WebService将一个程序放在服务器上，当我们添加一个web引用之后，就可以像调用本地程序一样调用服务器中的程序。但是我们拿不到引用程序的dll。

WebService的创建与使用步骤：

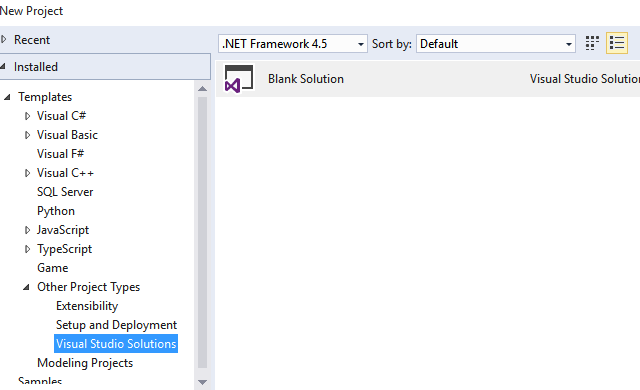
（1）定义Web服务

（2）添加代理类

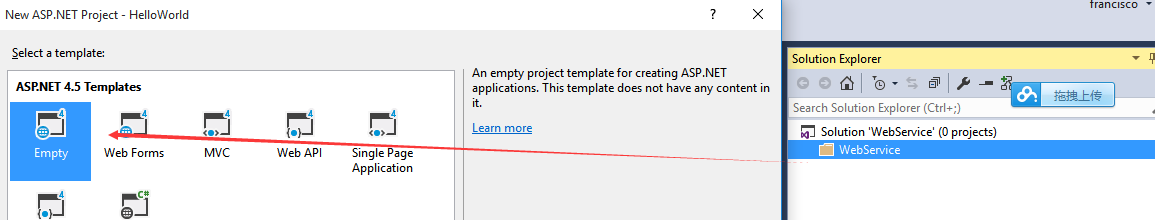
（3）使用代理类完成调用

## 定义Webservice服务

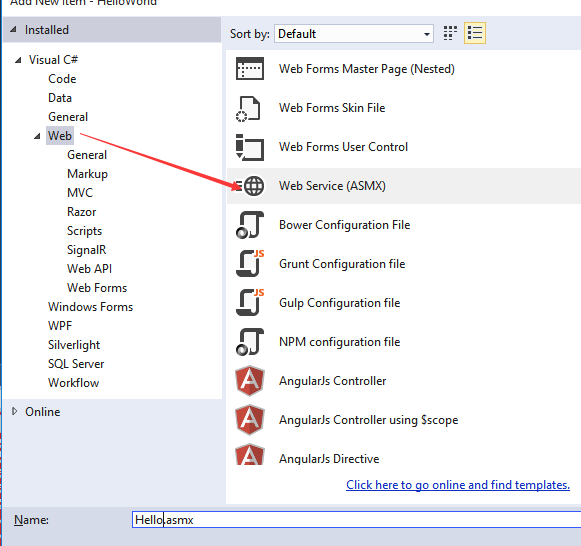
### 创建空解决方案



### 添加空的asp.net项目

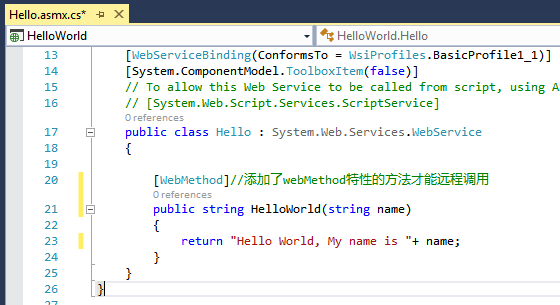


### 添加webservice项

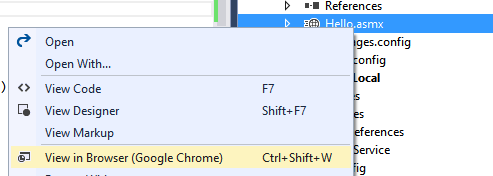


### 定义服务

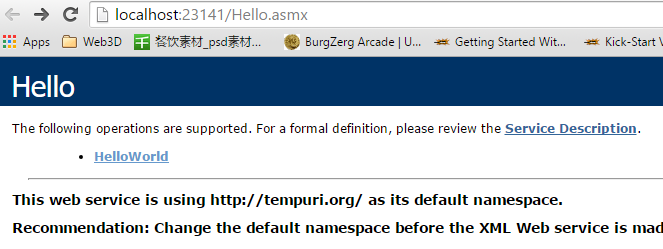
在asmx文件中定义HelloWorld服务。



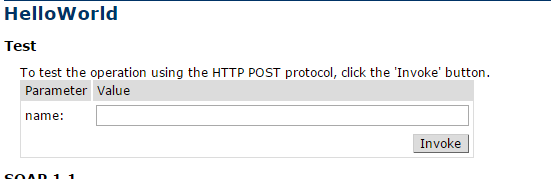
### 预览服务



结果：



点击helloworld可以对该服务进行测试：

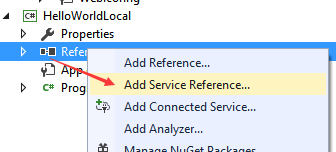


## 添加代理类

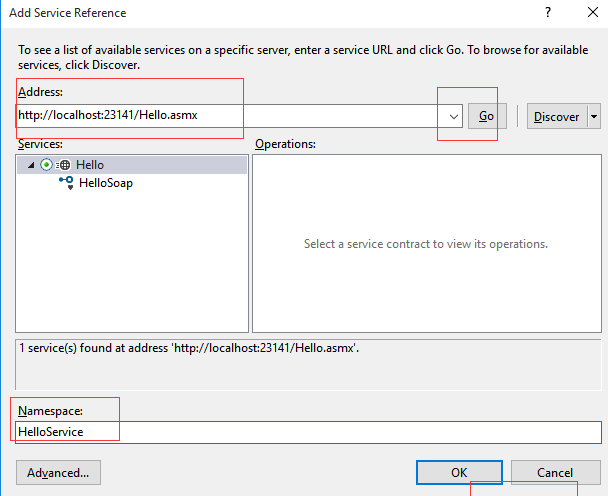
在本地程序中，要使用webService需要添加服务器引用。

### 添加服务器引用

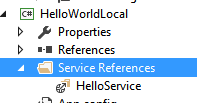
在引用上右击选择添加服务器引用。



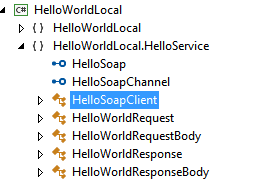
地址为运行webService服务的服务器的地址。



项目中多出下面文件。



自动生成代理类



## 使用代理类完成调用

|  |
| --- |
| //创建代理类。  HelloService.HelloSoapClient client = new HelloService.HelloSoapClient();  //调用HelloWorld  string helloworld = client.HelloWorld("hao");  Console.WriteLine(helloworld);  Console.Read(); |

调试输出结果：



# WebAPI

webService是面向活动的（请求的是方法），而Web API面向资源的，根据不同的请求类型，完成资源的调度，不需要明确调用某一个方法。

特点：

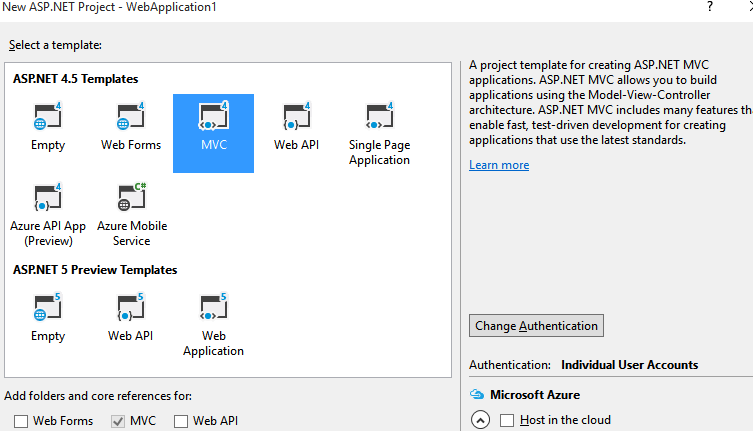
* 可以将数据转换成json、xml等格式，只提供一个请求，而不指定页面
* 可以在前台使用jquery完成调用
* 可以使用json.net完成序列化，在序列化时可以自定义节点特性，如空值不输出、序列化名称
* 使用REST风格的web服务
* 可以自定义方法，但是需要用HttPGet、HttpPost、HttpPut、HttpDelete约束方法

## 创建

### 创建项目

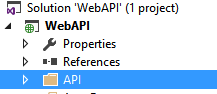
Web API是MVC支持的产品，创建项目时，创建MVC即可。Vs2015多了Web API

的选项。

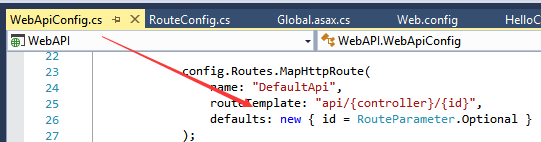


案例：修改

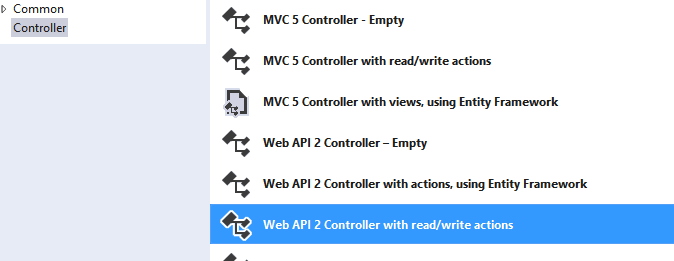
### 添加API文件夹



因为Web Api的路由规则是api开头的



### 创建Web API控制器



### 分析

|  |
| --- |
| public class HelloController : ApiController  {  // GET: api/Hello  // 获取对象列表  public IEnumerable<string> Get()  {  return new string[] { "value1", "value2" };  }  // GET: api/Hello/5  // 获取对象  public string Get(int id)  {  return "value";  }  // POST: api/Hello  // 添加对象  public void Post([FromBody]string value)  {  }  // PUT: api/Hello/5  // 修改对象  public void Put(int id, [FromBody]string value)  {  }  // DELETE: api/Hello/5  public void Delete(int id)  {  }  } |

根据请求的不同，会调用上面代码中不同的方法，实现资源调度（增删改查）。如果是自定义的方法需要用HttPGet、HttpPost、HttpPut、HttpDelete进行约束

|  |
| --- |
| [HttpPut]  public void getPut(int id)  {  } |

## 请求方式

WebAPI有两种请求方式：

* ajax调用jquery.ajax发送请求
* HttpClient对象，调用\*\*\*Async方法进行操作

### Ajax实现调用

#### 创建静态页面

新建一个静态页面。添加标签：

|  |
| --- |
| <body>  <input type="text" id="Id" />  <br />  <input type="text" id="Name" />  <br />  <input type="button" id="submit" value="修改" />  </body> |

#### Jquery异步请求

|  |
| --- |
| <script type="text/javascript">  $(function () {  $("#submit").click(function () {  var data = {  Id: $("#Id").val(),  Name: $("#Name").val()  };  data = JSON.stringify(data); //对象序列化为json格式  $.ajax({  // api/Hello/5  type: 'PUT',//请求类型。get,post,put,delete  url: 'api/Hello/' + $('#Id').val(),//请求地址  data: data,//参数  contentType: "application/json; charset=utf-8",//数据类型  dataType: 'text',//返回数据类型  success: function (data) {  //返回序列化格式："{\"Id\":1,\"Name\":\"hao123\"}"。需要2次JSON.parse  data = JSON.parse(data);  data = JSON.parse(data);  alert(data.Name);  //if (eval(msg) == '1') {  //InitData();  // alert("成功");  //}  }  });  });  });  </script> |

#### WebAPI接收信息

在2.2.1.2中，请求类型为Put，WebAPI中调用的是Put方法，因为传递过来的是Json序列化对象，可以用自动装配来接收数据。

这里我们创建一个实体对象（实际项目中可以使用EF来实现）。

|  |
| --- |
| public class CustomerInfo  {  public int Id { get; set; }  public string Name { get; set; }  } |

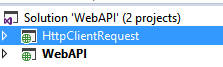
修改put方法

|  |
| --- |
| public string Put(int id, CustomerInfo customerInfo)  {  //修改数据  customerInfo.Name = customerInfo.Name + "123";  var d = new {Id=customerInfo.Id,Name=customerInfo.Name };  System.Web.Script.Serialization.JavaScriptSerializer jss = new System.Web.Script.Serialization.JavaScriptSerializer();  string data = jss.Serialize(d);  return data;  } |

### HttpClient

Ajax只能实现域内请求。跨域（域名或端口更换都算跨域）请求需要通过HttpClient实现。

#### 新建项目



不同项目端口不同，也就是跨域

#### 修改webAPI的get方法

|  |
| --- |
| // GET: api/Hello/5  public string Get(int id)  {  return "My Id is "+ id ;  } |

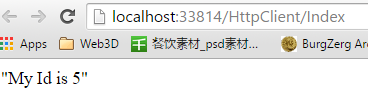
#### 通过HttpClient调用

|  |
| --- |
| public ActionResult Index()  {  //创建对象  HttpClient client = new HttpClient();  //获取结果，调用 get方法，传入参数为5  string result = client.GetAsync("http://localhost:31034/api/Hello/5").Result.Content.ReadAsStringAsync().Result;  ViewBag.result = result;  return View();  } |

#### 视图展示

|  |
| --- |
| <div>  @ViewBag.result  </div> |

结果:



# WCF

## 契约

在WCF中契约通过特性实现，有两种：

* 数据契约：传输数据的方式，添加DataContract、DataMember特性
* 服务契约：调用服务（接口）的方式，添加ServiceContract、OperationContract。

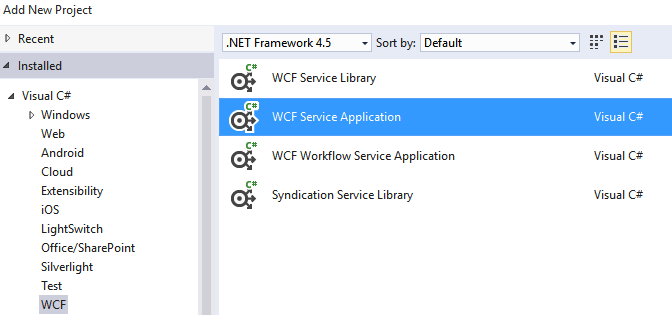
## 宿主

可以寄宿在控制台、winForm、等各种程序中

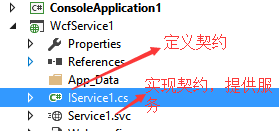
## WCF流程

|  |
| --- |
| （1）创建类库，编写数据契约（DataContract、DataMember）  服务契约（ServiceContract、OperationContract）  （2）创建WCF项目，让类实现服务契约，以数据契约作为参数或返回值  （3）修改配置文件，将协议改为http与wsHttpBinding  <protocolMapping>  <add binding="wsHttpBinding" scheme="http" />  </protocolMapping>  （4）创建控制台项目，引用契约项目  ICustomerService client = ChannelFactory<ICustomerService>.CreateChannel(  new WSHttpBinding(),  new EndpointAddress("url地址")  ); |

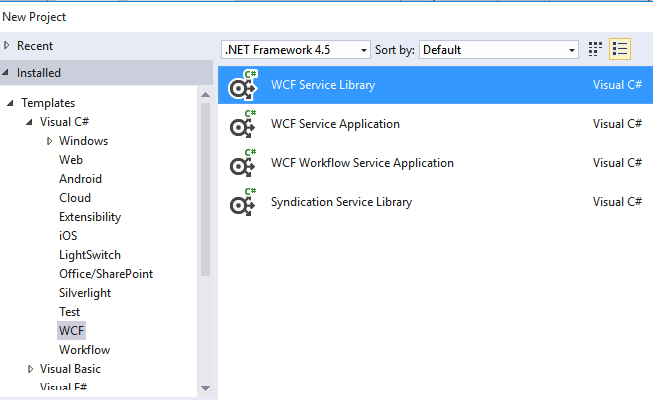
上述流程中需要创建2个项目，实际上只需要创建一个wcf应用程序即可。



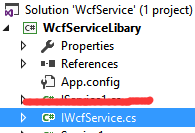
在这个程序中可以定义契约，以及实现契约。



### 创建类库



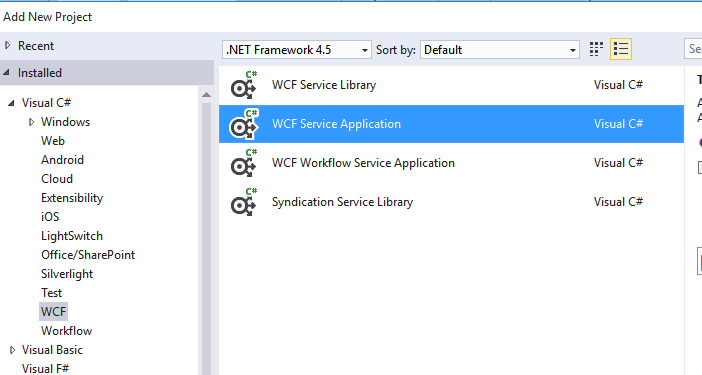
创建类：



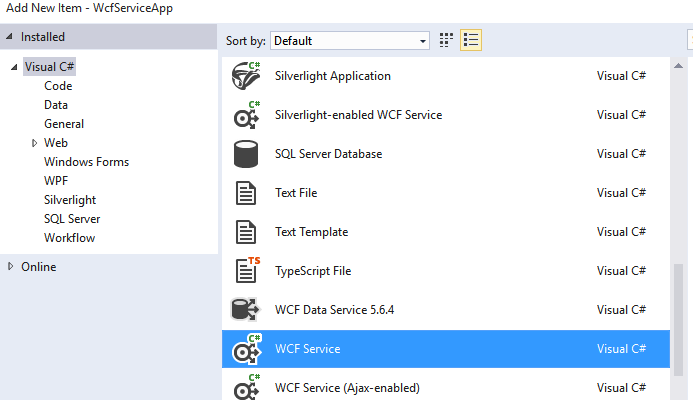
编写契约：

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Runtime.Serialization;//数据契约需要的命名空间  using System.ServiceModel;//服务契约需要的命名空间  using System.Text;  using System.Threading.Tasks;  namespace WcfServiceLibary  {  //定义数据契约,这个类型的对象可以在服务的参数、返回值处使用。  [DataContract]  public class Person  {  //参数用的契约  [DataMember]  public int Id { get; set; }  [DataMember]  public string Name { get; set; }  }  //服务契约  [ServiceContract]  public interface IPerson  {  //提供的服务  [OperationContract]  string HelloPerson(Person p);  }  } |

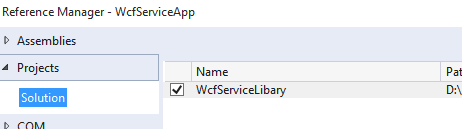
### 创建WCF项目



添加wcf服务



引用类库



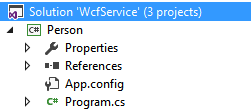
### 修改配置文件

将协议改为http与wsHttpBinding

|  |
| --- |
| <system.serviceModel>  .. . . .  <protocolMapping>  <add binding="wsHttpBinding" scheme="http" />  </protocolMapping>  ~~<serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />~~  </system.serviceModel> |

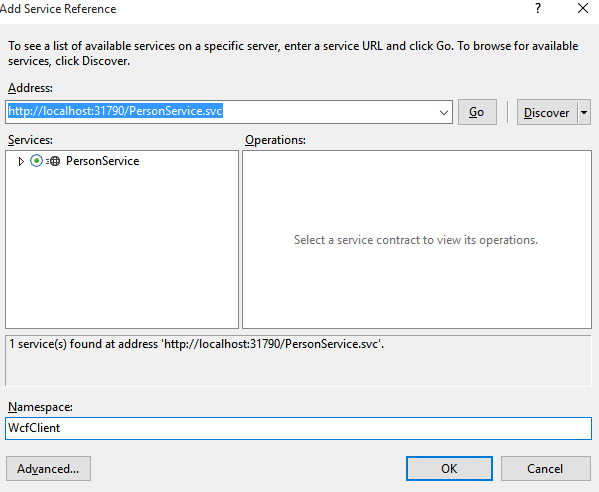
### 创建控制台项目，引用契约项目

创建项目



#### 第一种方式

添加引用：

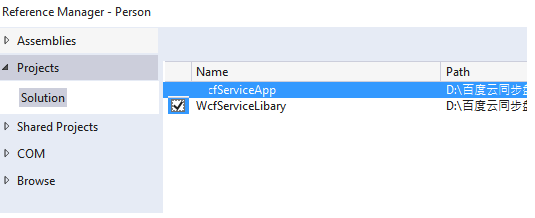


代理客户端调用

|  |
| --- |
| static void Main(string[] args)  {  PersonServiceClient client = new PersonServiceClient();  string hel = client.HelloPerson(new Person() { Id = 1, Name = "hao" });  Console.WriteLine(hel);  Console.Read();  } |

#### 第2种方式

引用



调用

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Runtime.Serialization;  using System.ServiceModel;  using Person.WcfServiceReference;  namespace Person  {  class Program  {  static void Main(string[] args)  {  WcfServiceLibary.IPersonService person = ChannelFactory<WcfServiceLibary.IPersonService>.CreateChannel(new WSHttpBinding(),  new EndpointAddress("http://localhost:31790/PersonService.svc"));  string hello = person.HelloPerson(new WcfServiceLibary.Person() { Id = 2, Name = "hao" });  Console.WriteLine(hello);  Console.Read();  }  }  } |