

## 基于 SoapHeader 的 API 访问授权和 API 调试方案

本方案适用于 Webservice 做 Api 的访问控制，不适用于 Webapi。

本方案内容如下

- 1、利用 SoapHeader 的支持进行 Webservice 的简单访问授权控制。
- 2、Client 访问 Api 时参数传递，简单类型和复杂类型参数。
- 3、如何对 api 进行调试，包括普通调试和跨项目的高级调试。

## Api 授权

华通项目中采用此种，如下

```
/// 根据车票号获取单张车票信息
/// add by ys on 2013-11-12
/// </summary>
/// <param name="ticketNo">车票号</param>
/// <returns></returns>
[ScriptMethod]
[WebMethod(Description = "根据车票号获取单张车票信息")]
[SoapHeader("htSoapHeader", Direction = SoapHeaderDirection.InOut | SoapHeaderDirection.Fault)]
public string GetTicketInfoByTicketNo(string ticketNo)
{
    if (!htSoapHeader.ValidateUser(htSoapHeader.UserName, htSoapHeader.Password)) return null;
    JavaScriptSerializer tool = new JavaScriptSerializer();
    APIReturnBO bo = new APIReturnBO();
    if (string.IsNullOrEmpty(ticketNo))
```

访问 api 时，如下

```
using (var service = new CarstartScheduleWebService.CarstartScheduleWebService())
{
    service.HtSoapHeaderValue = new HTSoapHeader();
    service.FinishCarstartScheduleAsync(carScheduleBo.Sid, carScheduleBo.CarStartNo);
    service.FinishCarstartScheduleCompleted += new CarstartScheduleWebService.FinishCarstartScheduleCompletedEventHandler
(service_FinishCarstartScheduleCompleted);
}
```

在发起 api 访问的时候，需要给 soap 赋值。否则会报一个 soapheader 的异常。

简单说下

服务端定义一个 header 类继承于 SoapHeader

```
public class HTSoapHeader : SoapHeader
{
    //系统配置的用户名密码
    private static readonly string _userName = ConfigurationManager.AppSettings["userName"];
    private static readonly string _password = ConfigurationManager.AppSettings["password"];

    public HTSoapHeader() { }
    public string UserName;
    public string Password;

    public bool ValidateUser(string userName, string password)
    {
        return (userName == _userName) && (MD5Manger.GetInstance().Get32Md5Str(password) == _password);
    }
}
```

两个属性，\_userName,\_password 是令牌的用户名和密码，写死在 config 中，密码明文。

```
<configuration>
  <appSettings>
    <!--api token-->
    <add key="userName" value="test"/>
    <add key="passWord" value="123456"/>
  </appSettings>
</configuration>
```

再写一个方法验证客户端传过来的令牌是否正确

客户端也需要定义一个 headerManager，如下，负责读取 token 的用户名和密码

```
public class HTSoapHeaderManage
{
    //系统配置的用户名密码
    private static readonly string _userName = ConfigurationManager.AppSettings["userName"];
    private static readonly string _passWord = ConfigurationManager.AppSettings["passWord"];

    static HTSoapHeaderManage()
    {
    }

    public static string UserName
    {
        get { return _userName; }
    }

    public static string PassWord { get { return _passWord; } }
}
```

属性只读，令牌密码采用同样的处理策略，写死在 config 中，密文。

```
<appSettings>
  <!--Api Token-->
  <add key="userName" value="test" />
  <add key="passWord" value="E1ADC3949BA59ABBE56E057F2F883E" />
</appSettings>
```

WebService 中处理如下

```
// [System.Web.Script.Services.ScriptService]
public class SoapApiWebService : System.Web.Services.WebService
{
    public HTSoapHeader htSoapHeader;
    [WebMethod(Description="访问控制测试")]
    [SoapHeader("htSoapHeader", Direction = SoapHeaderDirection.InOut | SoapHeaderDirection.Fault)]
    public string SoapAccessTest()
    {
        if (!htSoapHeader.ValidateUser(htSoapHeader.UserName, htSoapHeader.PassWord))
        {
            return "无权限访问";
        }
        else
        {
            return "有权限访问";
        }
    }
}
```

1. 每一个 webservice 要添加一个 htSoapHeader，用来接收客户端传过来的 header
2. Webmethod 必须跟上描述信息，告诉使用者我这个方法是做什么用的。这是好的开发习惯
3. 针对方法，需要添加 SoapHeader 属性声明，在访问方法的时候，webservice module 会使用 soap 协议将数据包解析，取出包头，赋值给 htSoapHeader
4. Direction 属性标识，包头是作用于 client 还是 server，还是两端都作用。Inout 代表包头同时适用于 client 和 Server，in 标识 client 到服务器的访问需要有 header，如果没有，解析出的 header 是 null，Out，标识，webservice 在处理完数据以后，会按照 soap 的数据包格式将返回结果打包。
5. 最后验证令牌，做相应处理。

**Tips:** 注意 htSoapHeader 的属性必须为 public，否则浏览 webService 服务的时候报如下错误

## “/”应用程序中的服务器错误。

头属性/字段 SoapApiWebService.htSoapHeader 缺失或者不是公共的。

**说明:** 执行当前 Web 请求期间，出现未经处理的异常。请检查堆栈跟踪信息，以了解有关该错误以及代码中导致错误的出处的详细信息。

**异常详细信息:** System.Exception: 头属性/字段 SoapApiWebService.htSoapHeader 缺失或者不是公共的。

**源错误:**

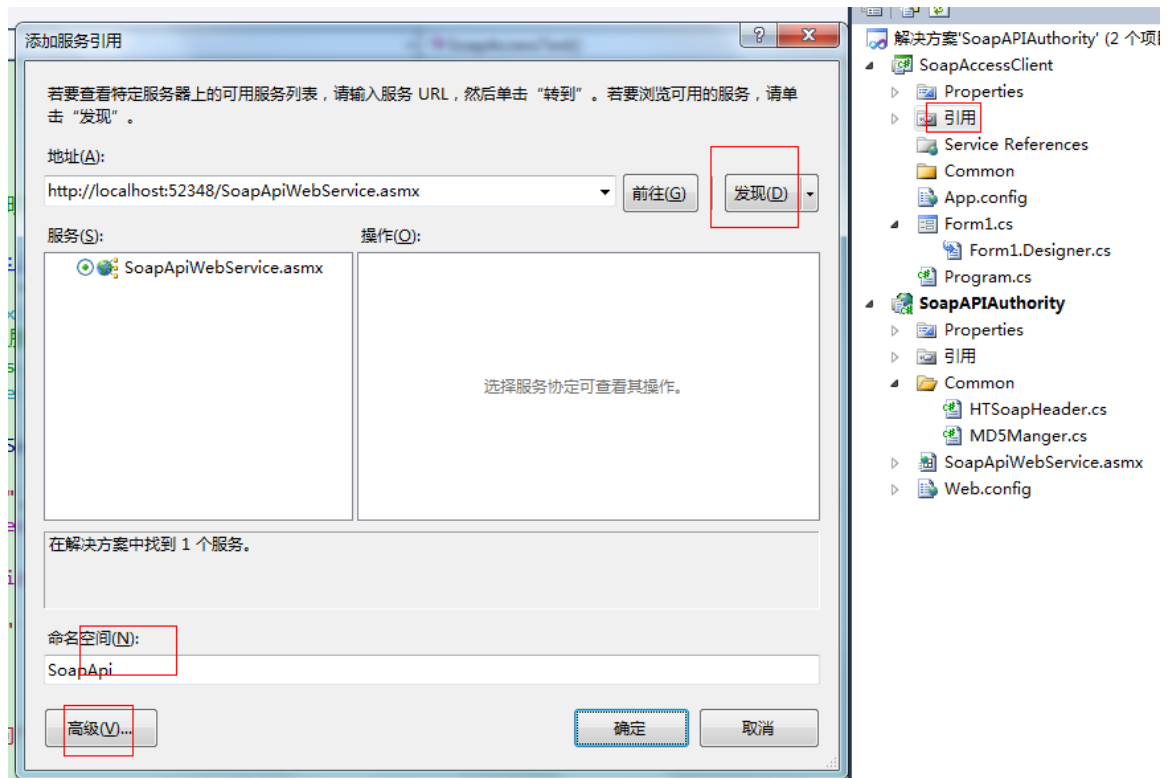
执行当前 Web 请求期间生成了未经处理的异常。可以使用下面的异常堆栈跟踪信息确定有关异常原因和发生位置的信息。

**堆栈跟踪:**

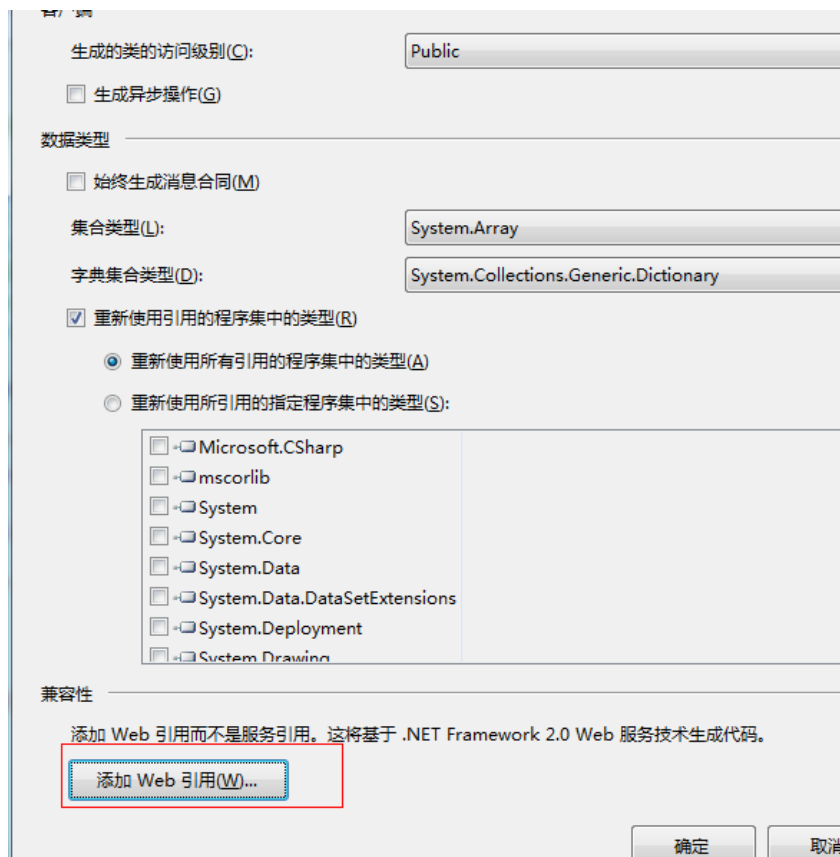
```
[Exception: 头属性/字段 SoapApiWebService.htSoapHeader 缺失或者不是公共的。]  
System.Web.Services.Protocols.SoapReflector.ReflectMethod(LogicalMethodInfo metho
```

使用 Api

添加对服务的引用



WebService api 的使用有两种形式，一种是 wcf 的添加服务引用，另一种是比较古老的添加 web 引用。从高级选项中进。



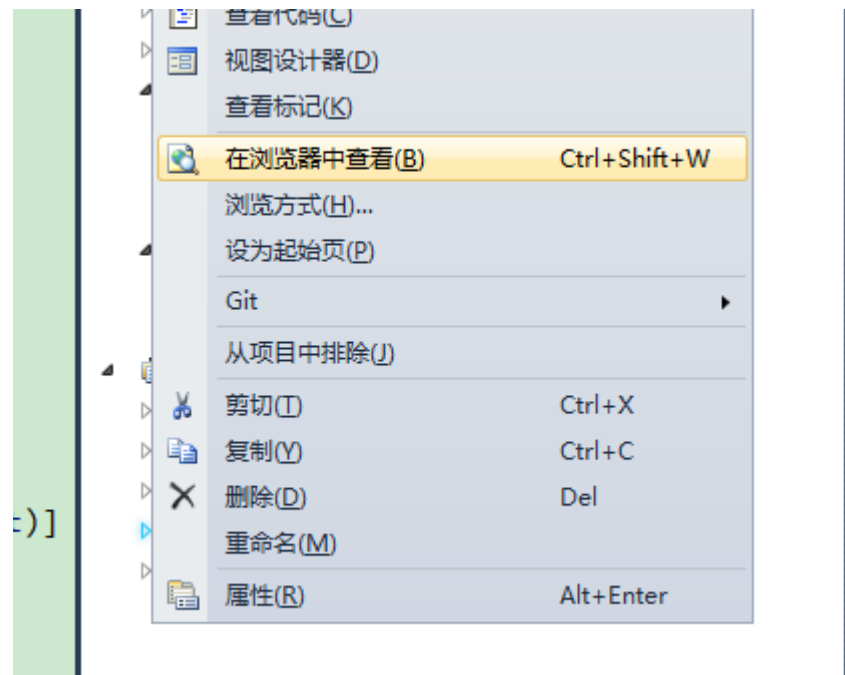
Web 引用是 Framwork 2.0 时代的产物。服务引用提供了更多的新特性。

引用服务的时候，可以使用远程的 api，也可以使用本地的 api，无需要发布、这个通常在调试 api 的时候特别有用。后面会说。

**Tips: 添加本地服务的话，要先让服务运行起来，否则，添加服务会失败。**

操作如下

选择 webservice 文件右键，浏览器查看



如果 webservice 正确无误。将看到 wsdl 的提供信息，这是服务描述

## SoapApiWebService

支持下列操作。有关正式定义，请查看[服务说明](#)。

- [SoapAccessTest](#)

**此 Web 服务使用 <http://tempuri.org/> 作为默认命名空间。**

**建议: 公开 XML Web services 之前，请更改默认命名空间。**

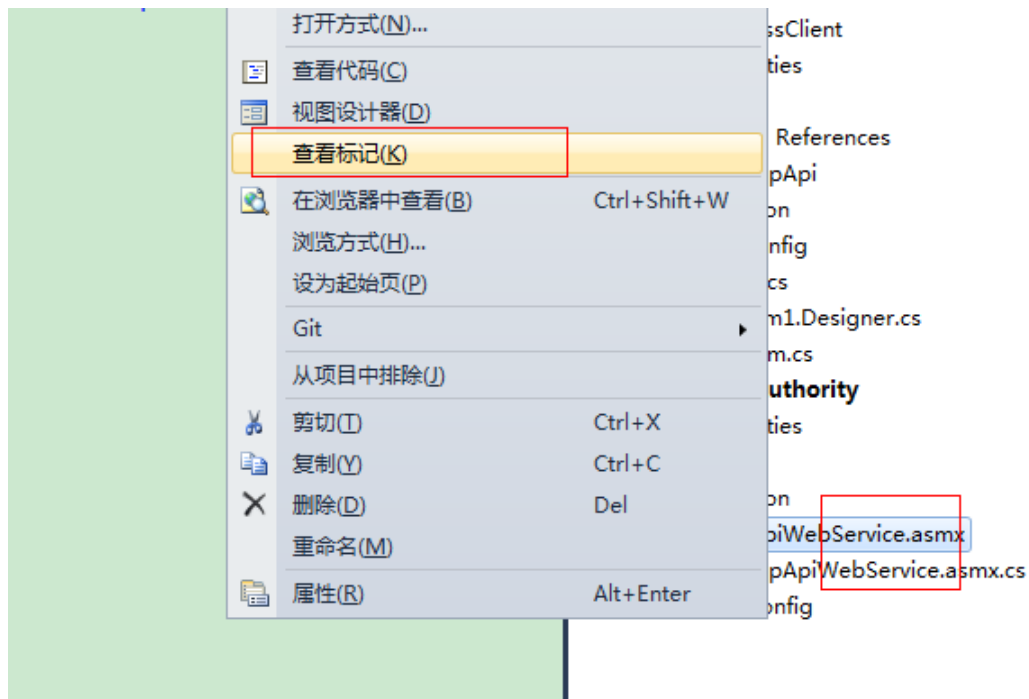
每个 XML Web services 都需要一个唯一的命名空间，以便客户端应用程序能够将它与 Web 上的其他应使用您控制的命名空间来标识 XML Web services。例如，可以使用公司的 Internet 域名作为命名空间。使用 ASP.NET 创建 XML Web services 时，可以使用 WebService 特性的 Namespace 属性更改默认命名空间。

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // 实现
}
```

如果点浏览器看看 webservice 报错，但是编译又通过，那就应该检查 webservice 的命名空间 Asmx 和 cs 文件绑定，命名空间不匹配，这通常是由于直接复制 webservice 再改名导致的。

解决办法，选中 webservice 文件，右键查看代码



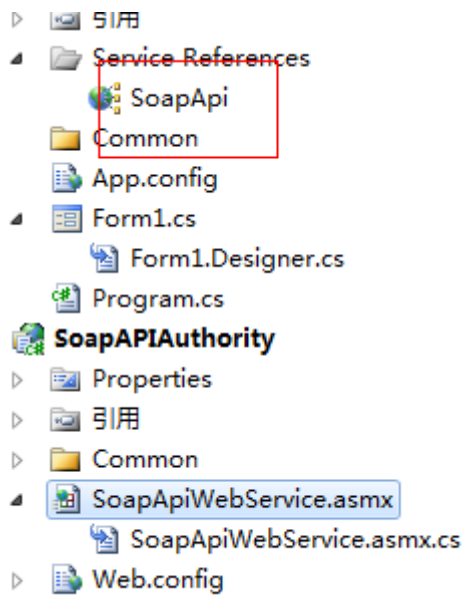
可以看到如下，检查这里的命名空间

```
WebService Language="C#" CodeBehind="SoapApiWebService.asmx.cs" Class="SoapAPIAuthority.S SoapApiWebService" %>
```

更改完，再次编辑浏览器查看服务，如果无误，添加就 ok，这样添加本地服务就完了。

如果添加远程服务，直接填入 url 地址或者 ip，点添加。

添加完毕后是这样



这里看到的服务名字并非服务的真实名称。显示的是我们在添加服务时填入的命名空间。

我采用添加添加服务引用的形式来演示服务的使用。

下面是两种使用形式，大同小异

```
using (var service = new SoapApi.SoapApiWebServiceSoapClient())
{
    try
    {
        var header = new SoapApi.HTSoapHeader();
        header.UserName = HTSoapHeaderManage.UserName;
        header.PassWord = HTSoapHeaderManage.PassWord;
        var result = service.SoapAccessTest(ref header);
        MessageBox.Show(result);
    }
    catch (System.Exception ex)
    {
        //write error log
    }
}
```

```

using (var client = new SoapApi.SoapApiWebServiceSoapClient())
{
    try
    {
        var header = new SoapApi.HTSoapHeader();
        header.UserName = HTSoapHeaderManage.UserName;
        //header.PassWord = HTSoapHeaderManage.PassWord;
        header.PassWord = string.Empty;

        var request = new SoapAccessTestRequest();
        request.HTSoapHeader = header;
        var response = ((SoapApiWebServiceSoap)(client)).SoapAccessTest(request);
        var result = response.SoapAccessTestResult;
        MessageBox.Show(result);
    }
    catch (System.Exception ex)
    {
        //write error log
    }
}

```

**Tips:**华通项目的策略是采用传统的 web 引用, 并且 Client 和 Server 共享同一类型 soapheader 然而 refencerce.cs 文件里已经重新帮我们生成好了 SoapHeader 类型, 个人不建议去修改这个文件。

修改 referecce.cs 中的任何代码。引用一旦 web 服务更新, 这些代码将重新生成, 之前所做的任何修改都将被覆盖。

何况 soap 本为跨平台而生, api 中的类型对 client 是不透明的, 又怎可共享一个类型。

## 参数传递

参数传递有两种, 简单类型和 Bo 类型。

简单类型的参数, 好理解, 也好调试, 但是粒度小, 服务新增删除参数, 都要重新在 client 引用服务。维护非常麻烦。就像这样。

```

/// <returns></returns>
[ScriptMethod]
[WebMethod]
[SoapHeader("htSoapHeader", Direction = SoapHeaderDirection.InOut | SoapHeaderDirection.Fault)]
public string SubmitTicketOrder(string banciId, string backBanciId, int count, string currencyType,
                                float orderPrice, string loginUserName, string firstDate, string secondDate,
                                string printedTickets, string seats, string backSeats, string machineNo,
                                string oldTicketString)

```

一个方法几十个参数, 无论客户端还是服务端, 都容易出错, api 通信接口粒度太小, 维护困难, 华通 api 设计的一大败笔。

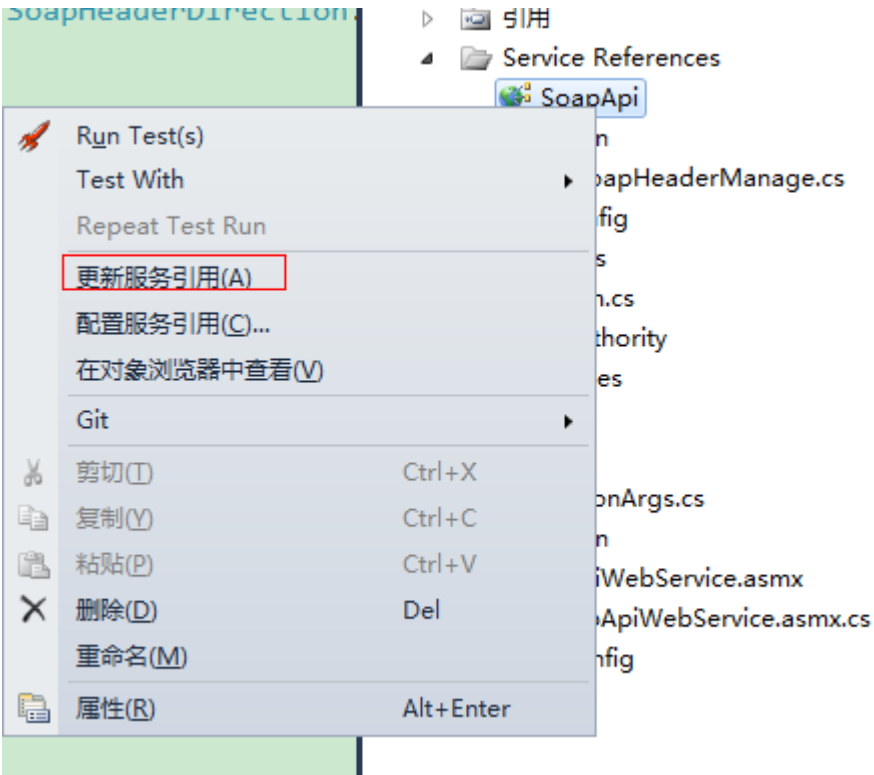


更优的做法，类似基于消息通信，将 api 通信接口粒度放大，定义 poco 参数类，采用 bo 进行传递。

bo 类型的参数，虽然造成了服务的调试不便(无法通过浏览器输入参数运行调试)。但是给更新和维护带来的好处，显然更甚。服务更新后，如果服务引用不更新，所有的 client 都会访问瘫痪。

而 bo 参数，则不存在这个问题，即使服务引用不更新，client 也能找出正常工作

对 webservice 服务进行更新后，比如本地 host 服务，需要编译 webservice 项目，然后到 client 去更新服务引用



简单参数传递

```

using (var service = new SoapApi.SoapApiWebServiceSoapClient())
{
    try
    {
        var header = new SoapApi.HTSoapHeader();
        header.UserName = HTSoapHeaderManage.UserName;
        header.PassWord = HTSoapHeaderManage.PassWord;
        var result = service.SimpleArgTest(ref header, 27);
        MessageBox.Show(result);
    }
    catch (System.Exception ex)
    {
        //write error log
    }
}

```

Bo 参数传递

```

using (var service = new SoapApi.SoapApiWebServiceSoapClient())
{
    try
    {
        var header = new SoapApi.HTSoapHeader();
        header.UserName = HTSoapHeaderManage.UserName;
        header.PassWord = HTSoapHeaderManage.PassWord;

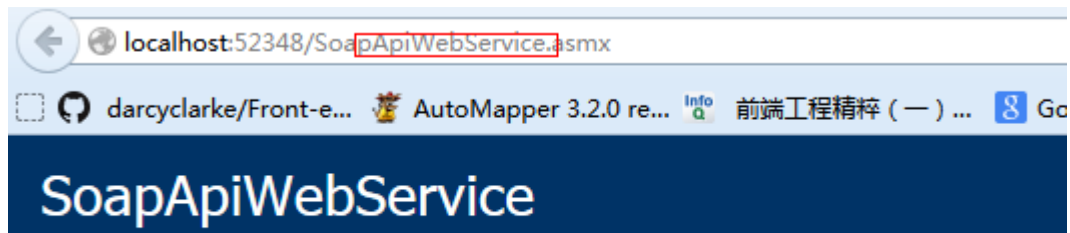
        var boArgs=new PersonArgS()
        {
            Name="yangshuang",
            Age=27
        }
        var result = service.BoArgTest(ref header, boArgs);
        MessageBox.Show(result);
    }
    catch (System.Exception ex)
    {
        //write error log
    }
}

```

## Api 调试

Api 调试，是让头疼，比较麻烦，尤其是 client 看似无问题，但是 server 端始终报错。需要调试，调试有两种，跨项目调试和普通调试

以下为普通调试



支持下列操作。有关正式定义，请查看[服务说明](#)。

- [BoArgTest](#)  
bo参数传递测试
- [SimpleArgTest](#)  
普通参数传递测试
- [SoapAccessTest](#)  
访问控制测试

此 Web 服务使用 <http://tempuri.org/> 作为默认命名空间。

设置 webservice 服务文件为起始页，设置该项目为启动项目，常规一样，选择要调试的方法

## SimpleArgTest

普通参数传递测试

### 测试

若要使用 HTTP POST 协议对操作进行测试，请单击“调用”按钮。

参数	值
arg:	<input type="text"/>

### SOAP 1.1

输入参数，debug 进去。

有时候我们把 api 发布后，client 总报错，就是不知道何原因。希望能从 client 直接 Debug 跟进到 Server 端，到底能不能行呢，肯定能行。

这是 VS 牛逼的地方。

这种调试方法尤其适用于 bo 参数传递的服务访问。可以看到 bo 参数的 web 服务，是不能通过和简单参数的服务一样进行调试的。

# SoapApiWebService

单击[此处](#)，获取完整的操作列表。

## BoArgTest

bo参数传递测试

### 测试

测试窗体只能用于使用基元类型作为参数的方法。

### SOAP 1.1

以下是 SOAP 1.2 请求和响应示例。所显示的占位符需替换为实际值。

```
POST /SoapApiWebService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/BoArgTest"

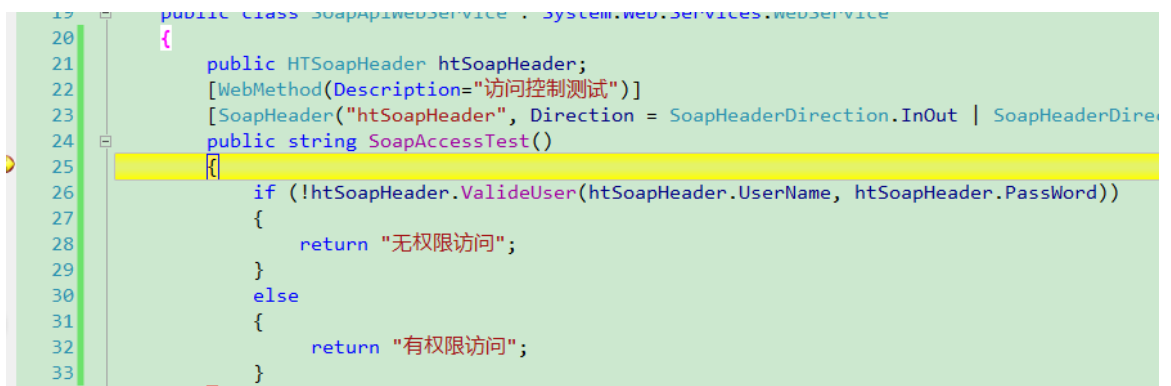
<?xml version="1.0" encoding="utf-8"?>
```

亦可用 log4net 日志来调试的，看状态，来回切换不甚麻烦。

显而易见，要调试 api，我们要用 self-host 的方式，添加服务为本地服务，方可单步 debug 进 api。远程 api 无法调试。

如下

1. 先让本地服务运行起来，webservice 右键，浏览器查看。
2. 添加本地服务到 client，如上，不赘述。
3. 编译 client，设为启动项目，然后 client 进入调试模式。注意到 api 访问的时候，F11 进入单步模式，即可顺序跟进到 Webservice。



```
19 public class SoapApiWebService : System.Web.Services.WebService
20 {
21     public HTSoapHeader htSoapHeader;
22     [WebMethod(Description="访问控制测试")]
23     [SoapHeader("htSoapHeader", Direction = SoapHeaderDirection.InOut | SoapHeaderDirection.Out)]
24     public string SoapAccessTest()
25     {
26         if (!htSoapHeader.ValidateUser(htSoapHeader.UserName, htSoapHeader.PassWord))
27         {
28             return "无权限访问";
29         }
30         else
31         {
32             return "有权限访问";
33         }
34     }
35 }
```

关于 webservice api 的访问授权，调试和高级调试，client 访问等介绍完了，有问题是私下找我。

2014-7-19