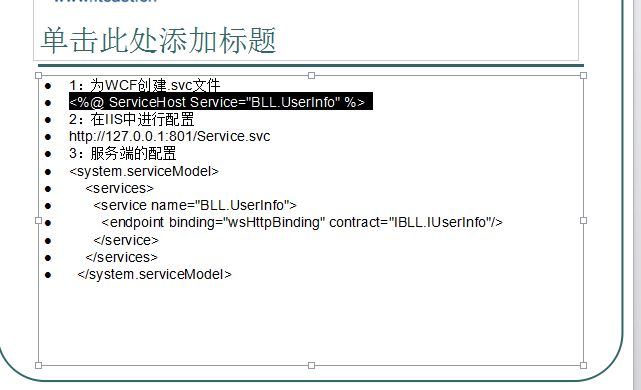
WCF的又一次扩充

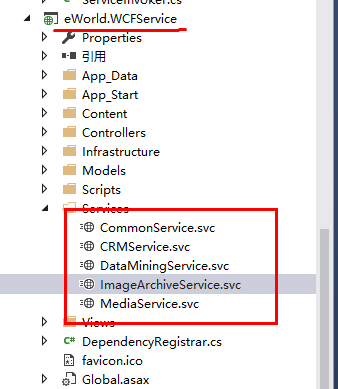
1.

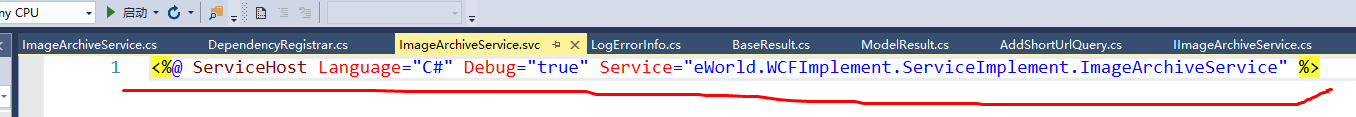
当创建好契约层，Model层，实现层，一种是发布成控制台，另一种就是发布在IIS上。

只需要在Web程序（WebForm或者WebMVC）上创建一个WCF服务，然后东西删光，再加上下面这句话就ok了。



查看公司的wcf服务，果然在WebService这一个web应用中，添加了这样的svc文件。

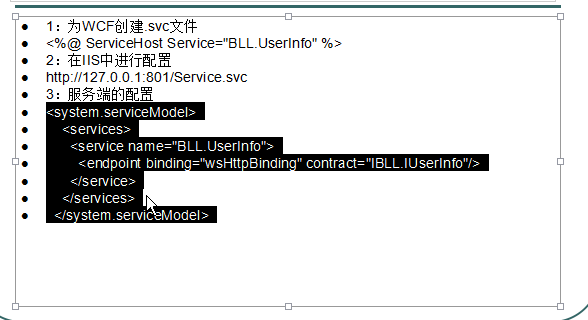




<%@ ServiceHost Language="C#" Debug="true" Service="eWorld.WCFImplement.ServiceImplement.ImageArchiveService" %>

老师这里的配置，项目里面没有，应该就是说的动态配置这种。

这里的配置是没有address的，因为地址是iis配置的时候才给定的。



项目中的类似代码：

namespace eWorld.WCFServerContract

{

public class eWorldWCFClient<TChannel>

{

private ServiceInvoker<TChannel> \_serverInvoker;

public eWorldWCFClient(bool IsReturnObj = false)

{

if (!IsReturnObj)

{

this.InitServerInvoker();

}

}

private void InitServerInvoker()

{

string wcfURL = string.Empty;

string IServiceName = typeof(TChannel).Name;

switch (IServiceName)

{

case "ICRMService":

wcfURL = WebServiceUrlManager.CrmWebServiceUrl;

break;

case "ICommonService":

wcfURL = WebServiceUrlManager.CommonWebServiceUrl;

break;

case "IImageArchiveService":

wcfURL = WebServiceUrlManager.ImageArchiveWebServiceUrl;

break;

case "IMediaService":

wcfURL = WebServiceUrlManager.MediaWebServiceUrl;

break;

case "IDataMiningService":

wcfURL = WebServiceUrlManager.DataMiningWebServiceUrl;

break;

}

this.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

}

public static eWorldWCFClient<TChannel> CreateClient(string wcfURL)

{

var client = new eWorldWCFClient<TChannel>(true);

client.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

return client;

}

public TResult Invoke<TResult>(Func<TChannel, TResult> function)

{

return this.\_serverInvoker.Invoke(function);

}

public void Invoke(Action<TChannel> action)

{

this.\_serverInvoker.Invoke(action);

}

private static BasicHttpBinding \_cmdServiceBinding;

public static BasicHttpBinding CmdServiceBinding

{

get

{

if (\_cmdServiceBinding == null)

{

\_cmdServiceBinding = new BasicHttpBinding();

\_cmdServiceBinding.OpenTimeout = new TimeSpan(0, 1, 30);

\_cmdServiceBinding.CloseTimeout = new TimeSpan(0, 1, 30);

\_cmdServiceBinding.SendTimeout = new TimeSpan(1, 30, 0);

\_cmdServiceBinding.ReceiveTimeout = new TimeSpan(0, 3, 0);

\_cmdServiceBinding.MaxBufferSize = int.MaxValue;

\_cmdServiceBinding.MaxBufferPoolSize = int.MaxValue;

\_cmdServiceBinding.MaxReceivedMessageSize = int.MaxValue;

\_cmdServiceBinding.MaxBufferPoolSize = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxBytesPerRead = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxStringContentLength = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxNameTableCharCount = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxArrayLength = int.MaxValue;

}

return \_cmdServiceBinding;

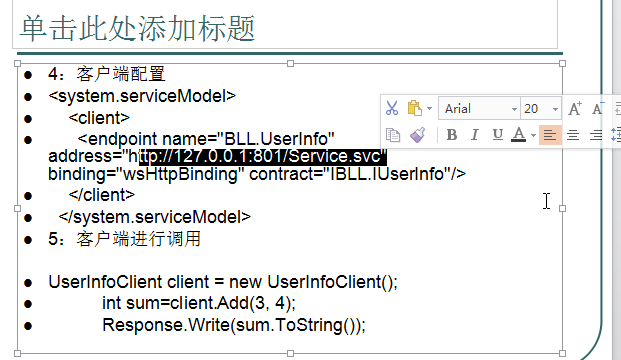
}

}

}

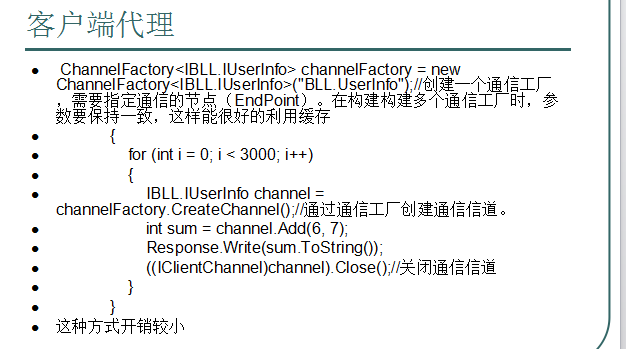
}

因为把服务发布在iis上面的时候，配置地址是没有给定，而是iis配置的时候用户给的。所以，wcf客户端在调用的时候，需要去配置这个已定的wcf服务地址。



2.

客户端代理：



客户端程序，控制台，winform，或者web程序，都是没有问题的。

调用的方式，除了添加服务应用，微软的工具，第三种用法，建立一个通信的管道，通信工厂。然后通过这个工厂，CreateChannel()；

在项目里面果然找到了！

private TChannel GetChannel()

{

return \_chanelFactory.CreateChannel();

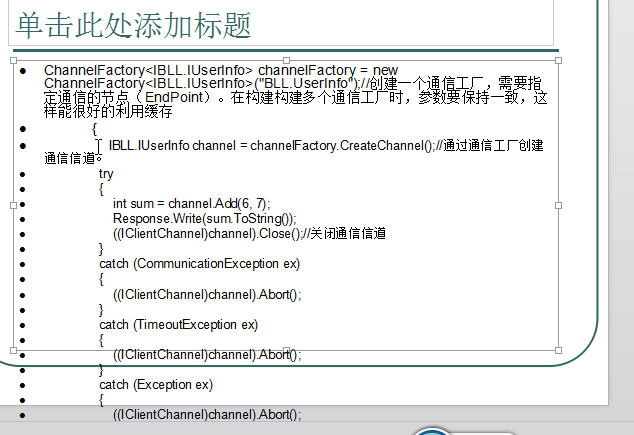
}

3.

客户端调用服务端的两个常见异常：



对此，客户端的捕获异常：



果然也在项目中找到了，看得出来，用的是同一款啊，如果是老师首创，那公司这里的WCF架构用的就是黑马的架构：

/// <summary>

/// 直接调用，无返回值

/// </summary>

public void Invoke(Action<TChannel> action)

{

var proxy = GetChannel();

ICommunicationObject channel = proxy as ICommunicationObject;

if (null == channel)

{

throw new ArgumentException("The proxy is not a valid channel implementing the ICommunicationObject interface", "proxy");

}

try

{

action(proxy);

channel.Close();

信道用完要关闭，但如果出异常了，就无法关闭了，所以下面的Catch中给强行终止Abort掉了。

}

catch (TimeoutException)

{

channel.Abort();

throw;

}

catch (CommunicationException)

{

channel.Abort();

throw;

}

catch (Exception)

{

channel.Abort();

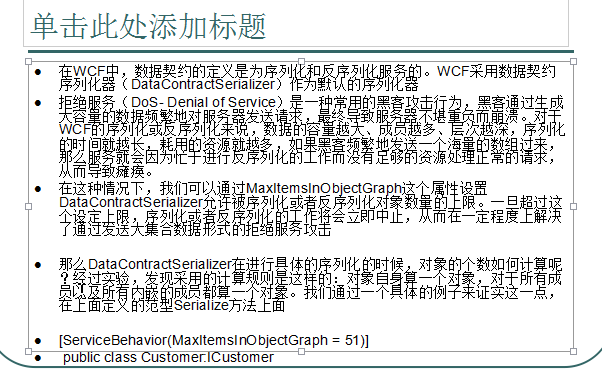
throw;

}

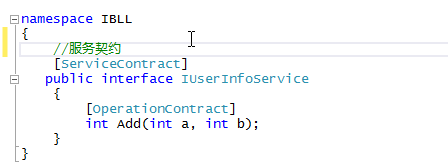
}

4.

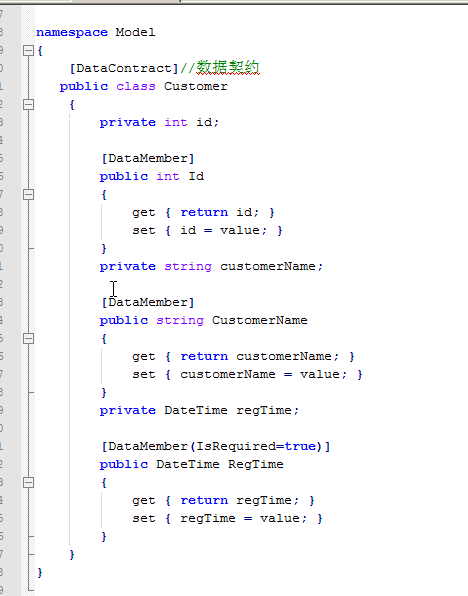




下面这个叫做服务契约



Model层就规定了数据契约：



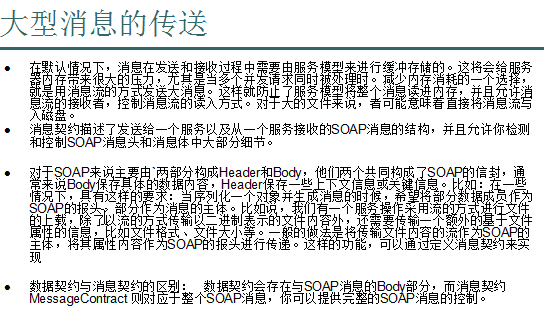
传输的过程中会对这些拥有数据契约的数据进行序列化。

但是如果我客户端能直接引用Model层，就不需要这个契约了，所以现在，Common，CRM，云归档，用的都是后者这种方式，把原来的eWorld.Core里面的数据类，现在都挪到了契约层当中。

主要目的也是为了Web层，也就是表现层和业务逻辑层的分离，这样以后还可以做服务层的集群，CRM不够，就多配几个CRM服务，云归档不够了，就多加几个云归档的服务。

5.

消息契约：



消息契约规定的是SOAP协议传输的格式。

老师的例子：

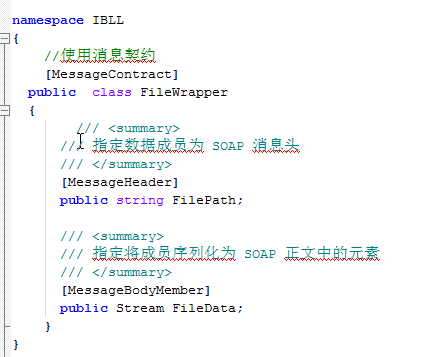
一个可以上传文件的方法，但传参是FileWrapper类型的。



FileWrapper类的具体定义，控制了SOAP契约传输的格式。控制了契约头和契约体。

头里面传文件的名字啊，路径啊，格式啊。SOAP体里面传文件的数据。这样就规定了SOAP协议传递数据的格式。

如果不自定义去规定，那所有的数据都放在SOAP协议的体里面去了。



6.

查看了项目里面的契约调用：

namespace eWorld.WCFServerContract

{

public class eWorldWCFClient<TChannel>

{

这里获取的是服务端的该契约TChannel对象

private ServiceInvoker<TChannel> \_serverInvoker;

public eWorldWCFClient(bool IsReturnObj = false)

{

if (!IsReturnObj)

{

this.InitServerInvoker();

}

}

private void InitServerInvoker()

{

string wcfURL = string.Empty;

string IServiceName = typeof(TChannel).Name;

switch (IServiceName)

{

case "ICRMService":

wcfURL = WebServiceUrlManager.CrmWebServiceUrl;

break;

case "ICommonService":

wcfURL = WebServiceUrlManager.CommonWebServiceUrl;

break;

case "IImageArchiveService":

wcfURL = WebServiceUrlManager.ImageArchiveWebServiceUrl;

break;

case "IMediaService":

wcfURL = WebServiceUrlManager.MediaWebServiceUrl;

break;

case "IDataMiningService":

wcfURL = WebServiceUrlManager.DataMiningWebServiceUrl;

break;

}

this.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

}

因为svc地址是通过iis配置的，所以这里要去获取。

老的项目是放在配置文件里面：

<!--公共 WCF服务-->

<add key="CommonWebServiceKey" value="http://localhost:9201/Services/CommonService.svc" />

<!--CRM WCF服务-->

<add key="CrmWebServiceKey" value="http://localhost:9201/Services/CRMService.svc" />

<!--影像存储与共享 WCF服务-->

<add key="ImageArchiveWebServiceKey" value="http://localhost:9201/services/ImageArchiveService.svc" />

<!--KeepAlive地址（如果IIS有虚拟目录，则包含虚拟目录名） -->

<add key="KeepAliveUrl" value="http://localhost:9203/" />

<!--电信WebServiceURL-->

<add key="ICACloudWebServiceURL" value="http://115.239.135.16/zjlhzx/services/prodTradeService?wsdl" />

获取了svc的url地址之后：

创建了wcf的客户端：

public static eWorldWCFClient<TChannel> CreateClient(string wcfURL)

{

var client = new eWorldWCFClient<TChannel>(true);

client.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

return client;

}

这里的CmdServiceBinding应该就是Binding：

public static BasicHttpBinding CmdServiceBinding

{

get

{

if (\_cmdServiceBinding == null)

{

\_cmdServiceBinding = new BasicHttpBinding();

\_cmdServiceBinding.OpenTimeout = new TimeSpan(0, 1, 30);

\_cmdServiceBinding.CloseTimeout = new TimeSpan(0, 1, 30);

\_cmdServiceBinding.SendTimeout = new TimeSpan(1, 30, 0);

\_cmdServiceBinding.ReceiveTimeout = new TimeSpan(0, 3, 0);

\_cmdServiceBinding.MaxBufferSize = int.MaxValue;

\_cmdServiceBinding.MaxBufferPoolSize = int.MaxValue;

\_cmdServiceBinding.MaxReceivedMessageSize = int.MaxValue;

\_cmdServiceBinding.MaxBufferPoolSize = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxBytesPerRead = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxStringContentLength = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxNameTableCharCount = int.MaxValue;

\_cmdServiceBinding.ReaderQuotas.MaxArrayLength = int.MaxValue;

}

return \_cmdServiceBinding;

}

}

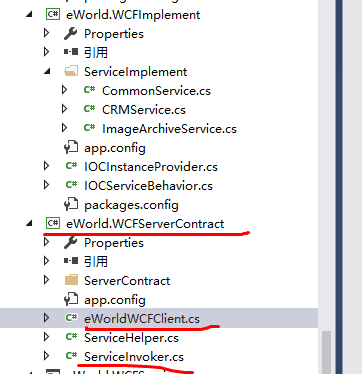
这里的C，也就是契约Contract没找到，还需要去查找这种客户端代理方式。

在想，这里的泛型，TChannel，是不是就是契约？它的名字确实是类似ICRMService这样的。

this.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

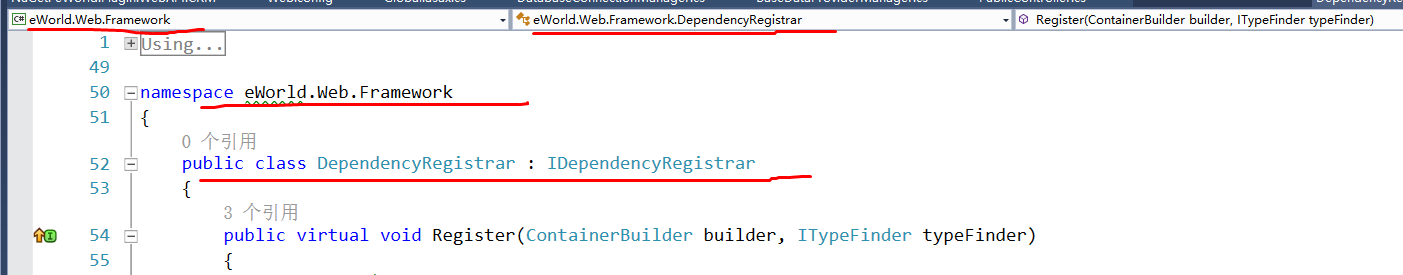
以上都在存在于契约层当中：

原来在契约层是包含了WCF客户端怎么去调用服务端的方式的：



新的配置是通过初始化程序去获取：

找到了WCF的注入：



namespace eWorld.Web.Framework

{

public class DependencyRegistrar : IDependencyRegistrar

{

public virtual void Register(ContainerBuilder builder, ITypeFinder typeFinder)

{

//初始化配置工具、WCF客户端调用

builder.RegisterType<InitializeConfigManage>().As<IInitializeConfigManage>().WithParameters(new List<NamedParameter>()

{

new NamedParameter("systemCode",1100),

new NamedParameter("initServiceSystemKey","bac11f3bd919f9fb")

}).SingleInstance();

//Service

builder.RegisterGeneric(typeof(eWorldWCFClient<>)).As(typeof(eWorldWCFClient<>)).UsingConstructor(typeof(IInitializeConfigManage)).InstancePerHttpRequest();

//open

builder.RegisterType<AppInfoService>().As<IAppInfoService>().InstancePerHttpRequest();

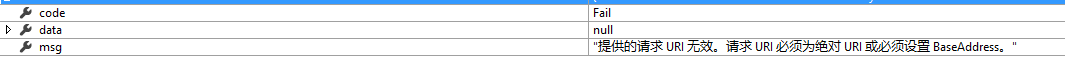
builder.RegisterType<AbilityInfoService>().As<IAbilityInfoService>().InstancePerHttpRequest();

//Yiwangqian

builder.RegisterType<YiWangQianSign.YiWangQianSign>().As<YiWangQianSign.IYiWangQianSign>().InstancePerHttpRequest();

}

我用控制台程序测试，都是fail的



上面已经配置了用初始化工具，所以跳到这里：

/// <summary>

/// 使用“初始化配置管理”创建客户端调用实例

/// </summary>

public eWorldWCFClient(IInitializeConfigManage initConfigManage)

{

string ServiceName = GetTypeName(typeof(TChannel));

string wcfURL = initConfigManage.GetWCFServiceUrl(ServiceName);

this.\_serverInvoker = new ServiceInvoker<TChannel>(CmdServiceBinding, new EndpointAddress(wcfURL));

}

接下来就是GetWCFServiceUrl(ServiceName)方法：

/// <summary>

/// 获取WCF地址，如CRMService

/// </summary>

/// <param name="ServiceName">服务名称</param>

/// <param name="ReturnCompleteWCFUrl">是否返回完整的WCF地址，false仅返回IP</param>

/// <returns></returns>

public string GetWCFServiceUrl(string ServiceName, bool ReturnCompleteWCFUrl = true)

{

if (\_cacheWCFConnectSetting == null || \_cacheWCFConnectSetting.Count <= 0)

\_cacheWCFConnectSetting = InvokeApiGetWCFConnectString();

上面获取所有的WCF连接，然后这底下筛选出来当前TChannel名字的那个wcf的url地址。

var currentWCFItem = \_cacheWCFConnectSetting.FirstOrDefault(s => s.Name == ServiceName);

if (currentWCFItem == null || string.IsNullOrEmpty(currentWCFItem.Url))

throw new Exception(string.Format("初始化服务程序[WCF地址配置]暂未配置{0}地址！", ServiceName));

Uri hostUri = new Uri(currentWCFItem.Url);

if (!ReturnCompleteWCFUrl)

return hostUri.ToString();

else

{

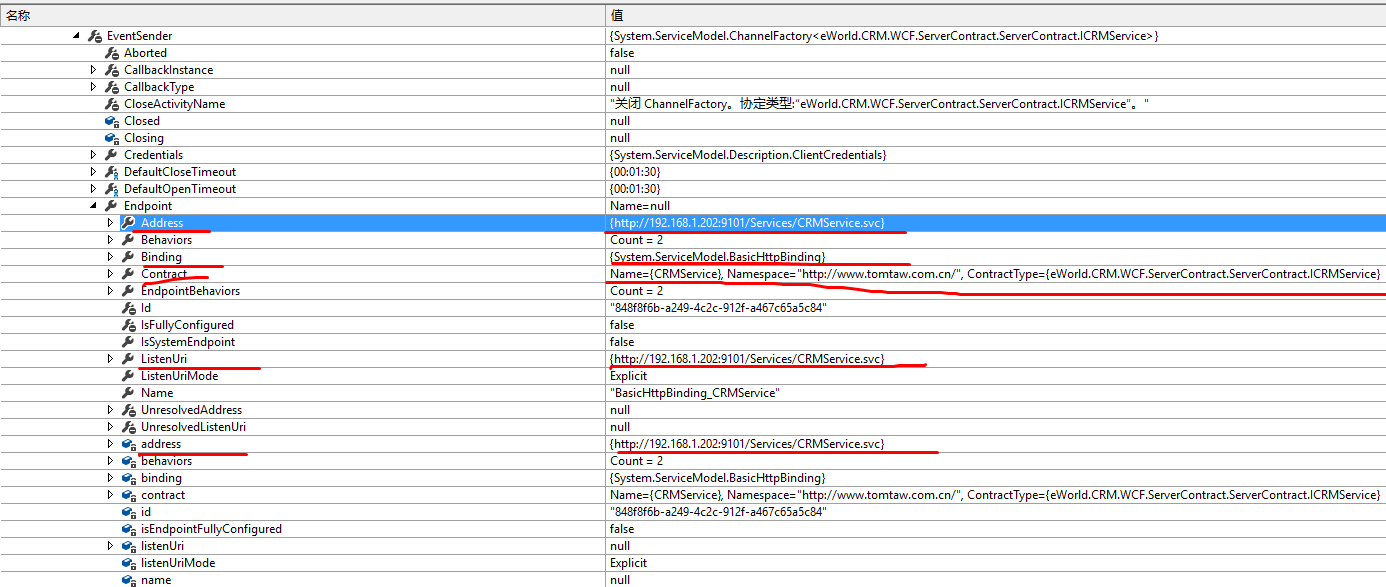
Uri retUri = new Uri(hostUri, string.Format("/Services/{0}.svc", ServiceName));

return retUri.ToString();

}

}

继续断点调试（CRM的api），找到了wcf的地址：



找错，发现它是跳到了这个无参的构造函数了，所以客户端都没有赋值。



明天来使用ImageCloud的wcf调用工具试试。因为Image归档，那里已经正常了。

7.

8.