## 第一:History历史:SOA架构思想,WebService标准及.net下的WS大一统解决方案WCF历史进程和快速搭建

一:SOA的概念

1. 进程

面向对象:解决代码维护性问题【设计模式】

面向组件:解决复用问题【react】将HTML分成了各个组件

面向服务:解决多系统互通问题【跨平台,跨语言】

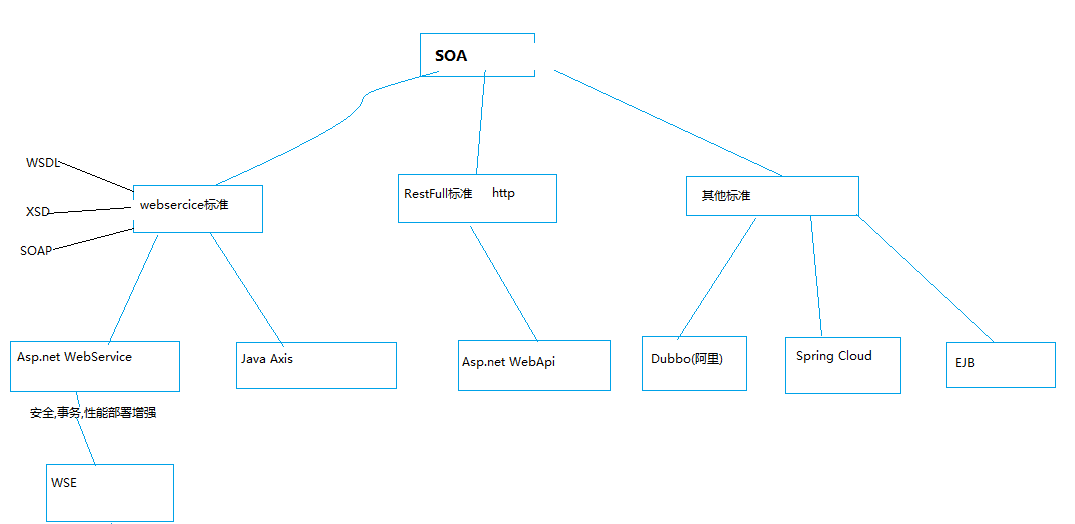
1. 概念:

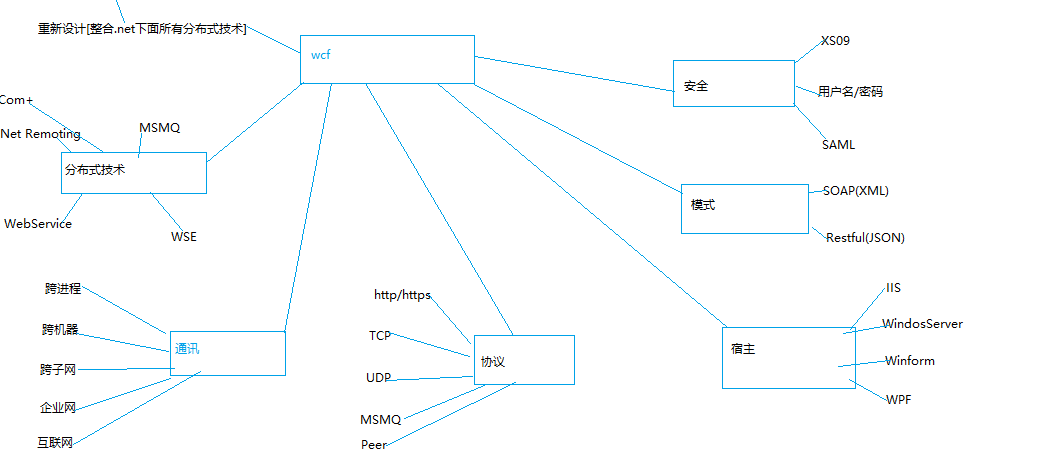
Service-Oriented Architecture 面向服务架构

1. 目的

解决IT企业信息化孤岛问题

1. WCF在SOA中的是什么地位?





WCF本质上是一个Asp.Net Web Service…

WCF出现的实收是WS标准盛行的时候,后来为了兼容restful模式,不得已才适配的…

1. 实现途径

WebService标准:

WSDL:公布服务内容

SOAP:简单对象访问协议【基于xml的文本格式】

XSD:将各自系统的强类型转成XSD规范进行互通 【XML序列化用的】

.net xsd java

System.String x:string java.lang.String

二.搭建项目

1.创建WCF Service Library

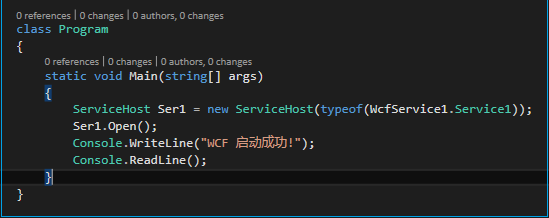


2.创建Console应用程序

(1).引用上面的lib dll.

(2).修改config文件

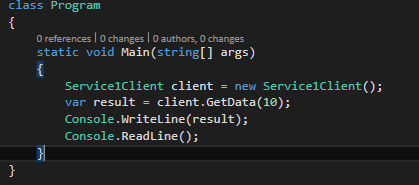
(3).配置数据



3.配置客户端

(1).添加服务引用

(2).调用服务



## 第二:Basic基础:WCF中的ABC三要素介绍及对Service和Client端的App.config中的System.serviceModel详讲

1. ABC
2. A?

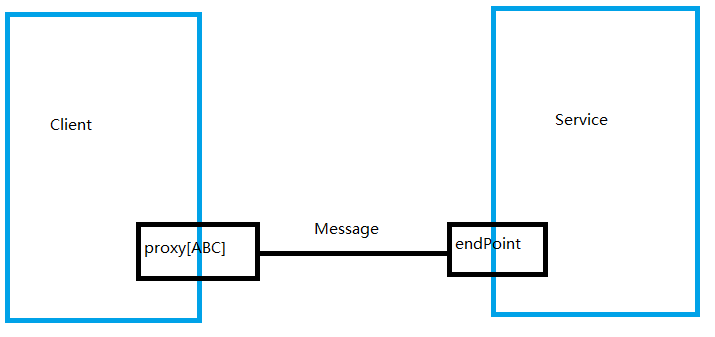
Address【你的家庭住址在哪,告诉我吧？】=>服务地址

1. B?

Binding【我该乘坐什么样的交通工具?】=>服务通道

1. C?

Contract【去你家做什么？打游戏，洗衣服？】=>服务边界



1. 配置文件

1.system.serviceModel 所有的wcf配置都在此节点上

《1》service:对外的endpoint进行配置

《2》bindings:对所有的B进行参数配置.(打开时间,关闭时间,传输时间等。。)

《3》behaviors:对影响wcf运行时的一些方法进行配置【服务线程数,服务实例,事物等等】.

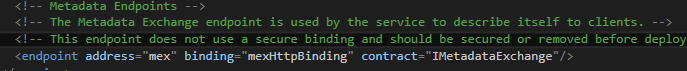


Add = baseAddresses + address;对外提供的服务地址;

binding = basicHttpBinding ; 通讯协议[http,tcp,msmq,udp]

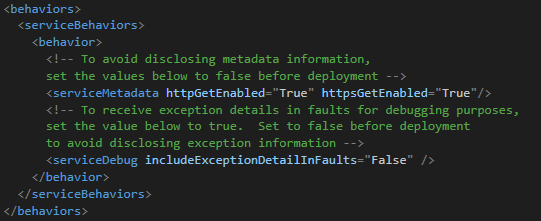
constract=代表我们的服务协议,提供哪些服务

以上第一个EndPoint包含的内容



第二个EndPoint是生成服务代理的地址

Add = baseAddress + address



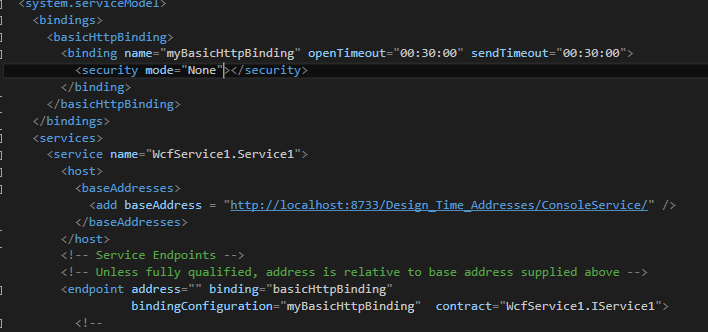
Behavior:

1. httpGetEnabled=”True” httpGetEnabled=”True”

开发项目的时候把他设置为False。将不会对外公开那些服务

1. serviceDebug includeExceptionDetailInFaults="False"

在调试的时候可以设置为true调试的时候将提供服务端的Exception信息

Binding的配置 

## 第三:Baisc基础:配置文件的缺点之如何通过在Service和Client端通过硬编码的方式抛弃配置文件提高代码的可读性

一:配置文件

1. 简简单单的一个程序,配置文件相比比较复杂。
2. 优缺点
3. 只需要更改配置文件,重启应用程序就可以及时更新,不需要修改程序。
4. 当wcf集群规模增加到5台+的时候配置文件的缺点就出来了。

解决方案：能进入代码的配置就进入代码…..

Config中心化

1. 缩减config配置文件,尽量能嵌入到code中
2. 实现config配置中心,基于zookeeper,Apollo

二:Service端代码段配置

配置文件:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="myBasicHttpBinding" openTimeout="00:30:00" sendTimeout="00:30:00">

<security mode="None"></security>

</binding>

</basicHttpBinding>

</bindings>

<services>

<service name="WcfService1.Service1">

<host>

<baseAddresses>

<add baseAddress = "http://localhost:8733/Design\_Time\_Addresses/ConsoleService/" />

</baseAddresses>

</host>

<!-- Service Endpoints -->

<!-- Unless fully qualified, address is relative to base address supplied above -->

<endpoint address="" binding="basicHttpBinding"

bindingConfiguration="myBasicHttpBinding" contract="WcfService1.IService1">

<!--

Upon deployment, the following identity element should be removed or replaced to reflect the

identity under which the deployed service runs. If removed, WCF will infer an appropriate identity

automatically.

-->

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<!-- Metadata Endpoints -->

<!-- The Metadata Exchange endpoint is used by the service to describe itself to clients. -->

<!-- This endpoint does not use a secure binding and should be secured or removed before deployment -->

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior>

<!-- To avoid disclosing metadata information,

set the values below to false before deployment -->

<serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>

<!-- To receive exception details in faults for debugging purposes,

set the value below to true. Set to false before deployment

to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="False" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

不需要配置文件代码如下:

static void Main(string[] args)

{

//1.配置baseAddress

ServiceHost Ser1 = new ServiceHost(typeof(WcfService1.Service1),new Uri("http://localhost:8733/Design\_Time\_Addresses/ConsoleService/"));

//2.添加endpoint

Ser1.AddServiceEndpoint(typeof(WcfService1.IService1), new BasicHttpBinding(), "");

//3.添加behaviors

var serviceMeta = new ServiceMetadataBehavior()

{

HttpGetEnabled = true,

HttpsGetEnabled = true

};

var serviceDebug = new ServiceDebugBehavior()

{

IncludeExceptionDetailInFaults = false

};

Ser1.Description.Behaviors.Add(serviceMeta);

//Ser1.Description.Behaviors.Add(serviceDebug);

//4.添加mex端点

Ser1.AddServiceEndpoint(typeof(IMetadataExchange), MetadataExchangeBindings.CreateMexHttpBinding(), "mex");

Ser1.Open();

Console.WriteLine("WCF 启动成功!");

Console.ReadLine();

}

三:Client代码配置

配置文件如下:

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpBinding\_IService1" />

</basicHttpBinding>

</bindings>

<client>

<endpoint address="http://localhost:8733/Design\_Time\_Addresses/ConsoleService/"

binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding\_IService1"

contract="ServiceReference1.IService1" name="BasicHttpBinding\_IService1" />

</client>

</system.serviceModel>

**不需要用配置文件代码1如下:**

static void Main(string[] args)

{

//1.添加binding和EndpointAddress

Service1Client client = new Service1Client(new System.ServiceModel.BasicHttpBinding(),new System.ServiceModel.EndpointAddress("http://localhost:8733/Design\_Time\_Addresses/ConsoleService/"));

var result = client.GetData(100);

Console.WriteLine(result);

Console.ReadLine();

}

**不需要用配置文件代码2如下(也不需要服务引用):**

static void Main(string[] args)

{

ChannelFactory<IService1> channelFactory = new ChannelFactory<IService1>(new BasicHttpBinding(), new EndpointAddress("http://localhost:8733/Design\_Time\_Addresses/ConsoleService/"));

var chanel = channelFactory.CreateChannel();

var result1 = chanel.GetData(100);

Console.ReadLine();

}

## 第四:Baisc基础:梳理wcf分层式通讯架构图【message,encoding,transport】及通过wireshark和fiddler详细分析

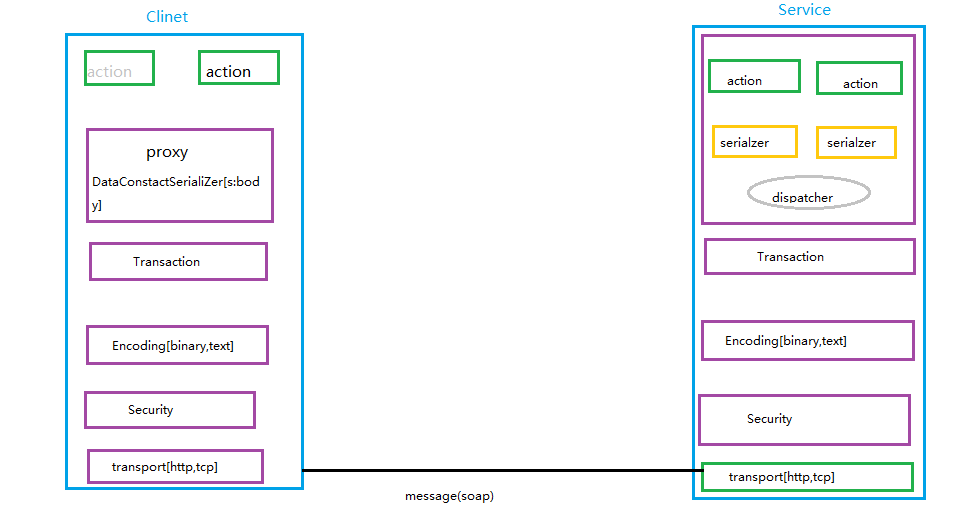
一:WCF体系结构

1. 协定
2. 服务运行时
3. 消息传递
4. 激活和承载

二:WCF通讯图

1. proxy

ServiceClient(cach)=>ChannelFactory=>IInputChannel/IOutputChannel….[http] HttpWebRequest



1. Transaction

Encoding

Security

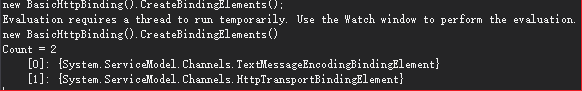
Transport

BasicHttpBinding [预定义协议栈]

Encoding(binary,text)

Transport(tcp,http)

《1》.**BasucHttpBinding:**



《2》. NetTcpBinding:

new NetTcpBinding().CreateBindingElements()

Count = 4

[0]: {System.ServiceModel.Channels.TransactionFlowBindingElement}

[1]: {System.ServiceModel.Channels.BinaryMessageEncodingBindingElement}

[2]: {System.ServiceModel.Channels.WindowsStreamSecurityBindingElement}

[3]: {System.ServiceModel.Channels.TcpTransportBindingElement}

《3》. NetMsmqBinding:

new NetMsmqBinding().CreateBindingElements()

Count = 2

[0]: {System.ServiceModel.Channels.BinaryMessageEncodingBindingElement}

[1]: {System.ServiceModel.Channels.MsmqTransportBindingElement}

1. 对比

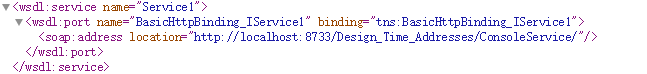
BinaryMessage 和TextMessage

## 第五:Constract契约:Service【Operaion】Contract在定义服务边界和[RPC,IsOneWay]操作和WSDL操控详解

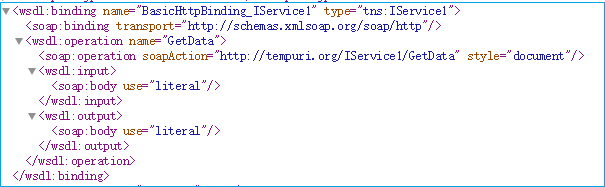
1. 契约【ABC的C】=》服务,操作契约【Service,Operation】
2. 作用：用于定义服务边界,也就是定制wsdl,这样就可以让客户端根据wsdl生成不同的代码。。。
3. wsdl可以找到的东西:
4. 服务地址:soap:address

Java wsimport由它来解析这个wsdl(生成硬编码)

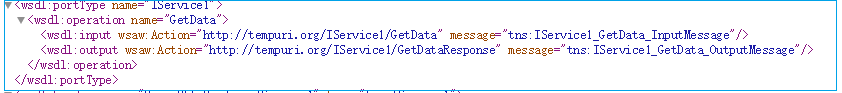
C# svcutil 由它来解析



1. 传输协议:soap:binding



1. 绑定类型:BasicHttpBinding
2. Operation 和 Message的对应关系(一对一,一对二？)



RPC通讯模式:因为是两条

最后都会通过硬编码到Dowork函数中…….

[System.ServiceModel.OperationContractAttribute(Action="http://tempuri.org/IService1/GetData", ReplyAction="http://tempuri.org/IService1/GetDataResponse")]

string GetData(int value);

二:三种模式

1. RPC模式

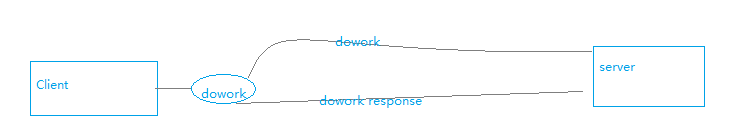
默认是请求响应模式,那么一个soap在网络往返传输中,如果区别是此方法的传出还是接受？？？

<1>对应关系

Operation=》Message

从WSDL中可以看到,一个operation对应着两条message的action

在Client的服务引用上也是可以看出来的。。



1. 单向模式(isOneway)分在线和离线

场景:不需要service及时给他消息,或者说不需要service返回消息…..

日志系统

Logstash[log4net] =>es mongodb

Wcf=>elasticSearch[UDP]

适合少量的丢失数据…..

发短信,发邮件【下单成功了….】

使用:

public interface IService1

{

[OperationContract(IsOneWay =true)]

string GetData(int value);

}

WSDL 只有一个input



1. fiddler监控:HTTP/1.1 202 Accepted,了解http 202 状态码
2. 应用场景:不需要service的逻辑响应,适合不需要等待结果的业务场景。
3. 双向通讯【发布订阅】

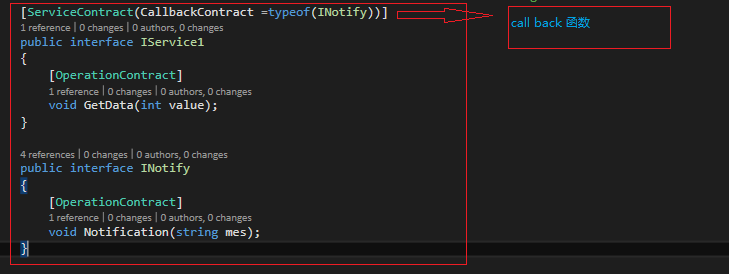
三.其他方式的定制

## 第六:Constract契约:理解WCF双工通讯中所体现的发布订阅模式思想及代码实践和xmind图分析

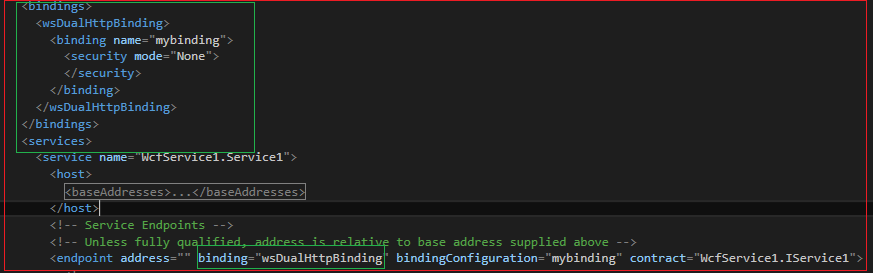
一:双工通讯

1. 作用:能够让client-service之间进行相互通讯
2. 模式:发布订阅模式
3. 代码实现

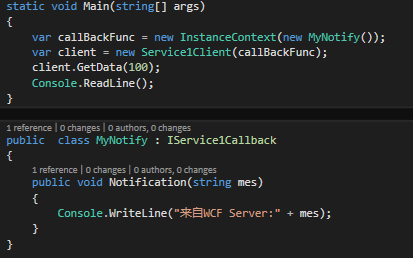
《1》.在ServiceConstract中指定当前Client的契约



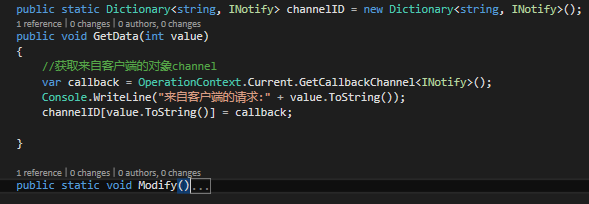
《2》.binding必须支持双工通讯



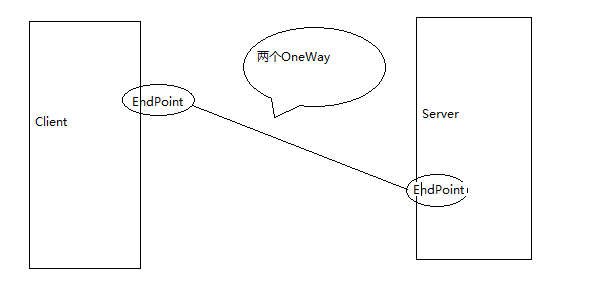
《3》.ICallBack的实现必须在客户端



《4》.在client调用的时候获取Clint的通道



1. 原理:通过两个IsOneWay进行实现.fiddler可查看,可画图



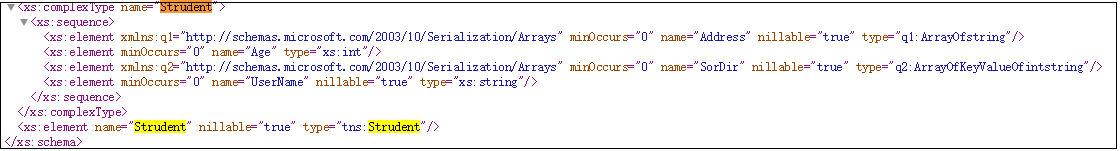
## 第七:Constract契约:DataContract在适配CLR类型和WSDL的XSD之间的灵活运通以及契约版本控制上的注意事项

一:WCF在消息交换中,可能会遇到的哪些复杂类型

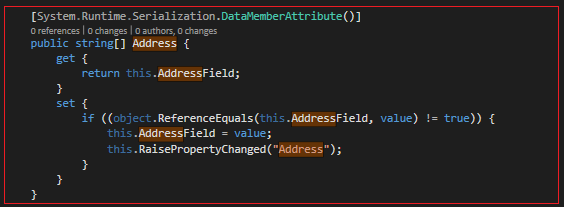
1. 普通的类【Class】
2. 集合对象
3. 字典类型
4. 枚举

二:对集合和枚举类型,需要解决的坑

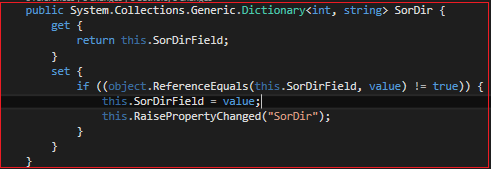
1. 对集合类型进行重新组装
2. 对字典进行重新组装



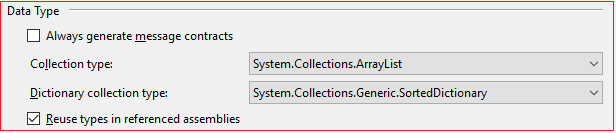
复杂类型List,SortedDictionary 都将在XSD中显示为Array

List 被svcutil解析结果为Array

SortedDictionary被svcutil解析结果为Dictionary



如何让其在客户端正确的显示:在服务引用上做手脚



**枚举为简单类型**

三:最后的方案

1. 如果是.net程序互通,尽量遵守同一份Entity,但是现实上,很难这样做到,尤其是跨语言互通.

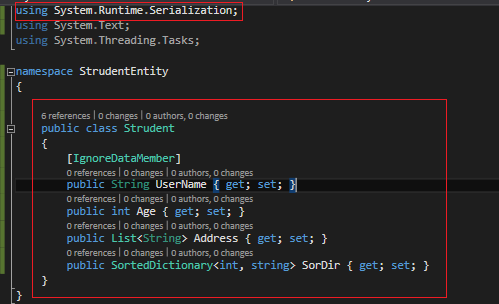
《1》.新建lib

《2》.服务端和客户端均引用这个Lib

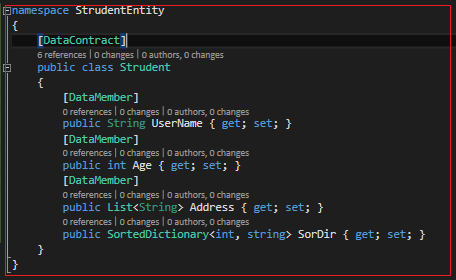
四:DataConstract,DataMember,EnumMember,对wsdl中的xsd进行可视化定制

1. 默认全部可以进行DataConstractSerializer序列化
2. 指定不需要再xsd中显示的字段

《1》.某些字段不显示,需要自己强制指定IgnoreDataMember



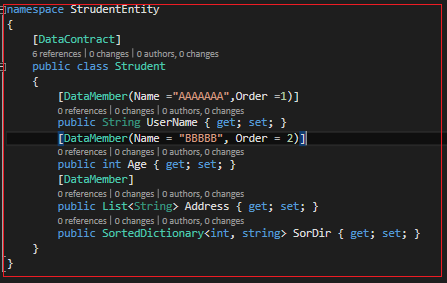
《2》.显shi进行DataConstact,DataMember声明



1. 其他方面的改进

《1》.DataMember改名对真是字段保护

《2》.调整Filed顺序【Order】



## 第八:Constract契约:MessageConstract采用基于Message编程下如何对SOAP高度定制化和巧用header下参数token验签

一:消息契约

<?xml version=”1.0” endcoding=”utf-8”?>

<s:Envelope xmlns:s=<http://schemas.xmlsoap.org/soap/envelope/>>

<s:Header>

</s:Header>

<s:Body>

<DoWork xmlns=<http://tempuri.org/>>

<msg>hellow world!</msg>

</Dowork>

</s:Body>

</s:Envelope>

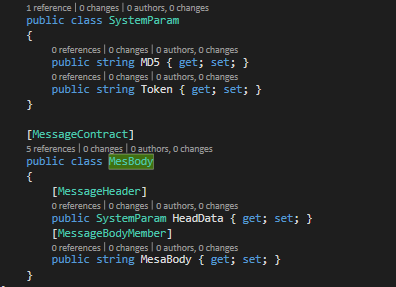
从上面一段Envelope中可以看到,数据契约的怎么变更,影响都是在s:Body中,而消息契约,可以影响到s:Header….

二.操作s:Header有什么好处?

1.可以单独在header中添加信息,方便第三方插件进行拦截的时候对header进行独立的处理。

1. 保证消息防篡改的token参数

《2》用户登录的用户名和密码的MD5验证



三.如何通过代码实现？

1.【MessageConstract】/[messageHeader]/[MessageBodyMember]实现soap和CLR的强类型映射

四.更大的灵活性之直接在参数中使用Message

1.更大的灵活性【编程麻烦,实际使用比较少,大家知晓】

2.代码实现

## 第九:Bindind绑定:高频使用之basicHttpBinding和java和netTcpBinding实现和.net程序进行通语言下跨机器互通代码实践

一:认识wcf支持的预定义binding

Binding就是提前定制好的协议栈【transport + encoding】

1. ws\*开头的binding……【用于跨语言互通的】

第一种:webservice模式【BasicHttpBinding,WSHttpBinding】

第二种:restful模式 【WebHttpBinding】

1. net开头的Binding…..【用于.net程序间的互通】

第一种:直连式的访问 【netTcpBinding,NetNamedPipedBinding】

第二种:断开时的访问 【netMSMQBinding】

1. 两种Binding的性能

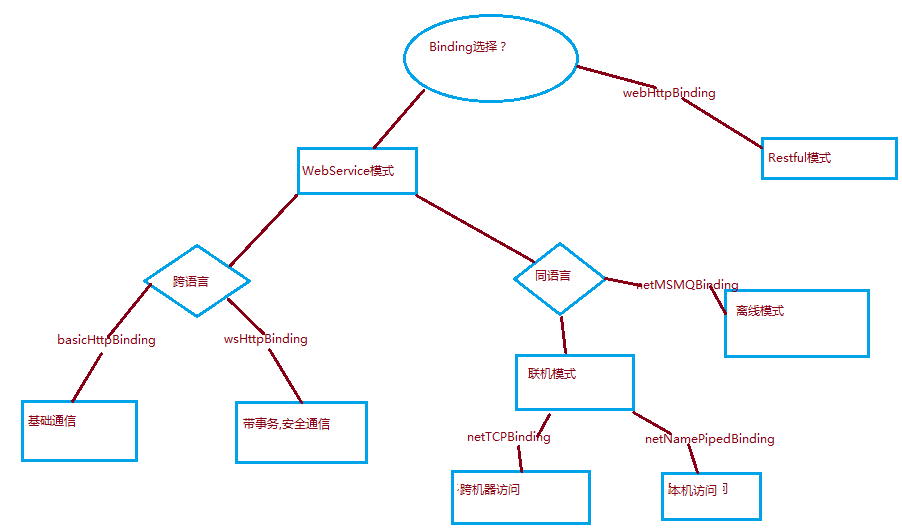
NetNamedPiped>netTcp>basic>WS

1. 遗憾

和现在的一些语言脱节了！！

WCF最大的遗憾就是tcp模式不支持跨语言互通,google开源的thrift就支持了这种模式,当然没人牵头做这种事情。

**Binding的选择**



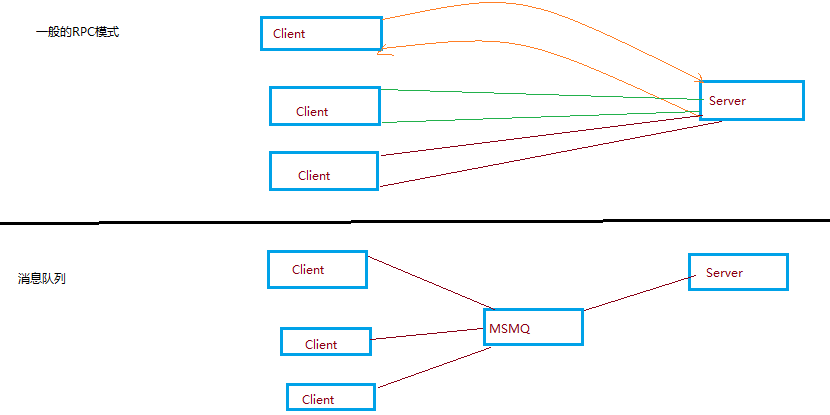
Thrift 代替TCP ….实现跨语言的互通

二:使用BasicHttp实现跨语言互通

三:使用netbinding实现通语言通讯

## 第十:Bindind绑定:基于msmq模式下netMSMQBinding实现和.net程序进行异步离线式互通代码实践

**消息队列的好处:**



1. 削峰

队列就是三峡,三峡可以拦截上游凶猛的洪水,否则下游年年洪灾

所以队列的好处就是可以有效避免底层系统被流量冲垮….

1. 同步架构异步化【解耦】

上游和下游的依赖关系呗拆解,上游不知道下游是谁,下游也不知道上游是谁,大家之和三峡放生关系,所以队列可以系统模块化,有益于横向扩容。

1. 并发请求串行化【无锁模式】

既然是队列,也就是FIFO,先行先出,也就不存在锁机制的问题了。

1. 提高系统稳定性

下游拓机挂掉,不影响上游业务,因为大家解耦了,比如淘宝下单,于物流公司对接平台挂了,大家就不相互影响.

二:什么是MSMQ?

1. 它是一个Windows自带的消息队列
2. 它是基于硬盘的…
3. 基于单机的….无集群【高可能,高并发】

以上三点,就决定了一个很low的东西。。。【90年代】

给人的感觉就是WCF还是保持在2000的感觉

三:WCF模式下的消息队列

1. WCF目前只支持自家的MSMQ…..这也是我认为WCF团队不给力的一个地方,居然没有开发对接第三方消息队列组件,比如支持ActiveMQ,ZeroMQ,Kafka,Redis,唯一一个由RabbitMQ自家的WCF Binding,也放弃更新了.
2. 这种90年代的单机版硬盘化的MSMQ,现实场景比较小,小系统还可以接受,大家主要理解这种MQ思想

NetMSMQ一定要求操作契约是IsOneWay的方式。

1. 在线模式,离线模式

一个Message是4M

## 第十一:Bindind绑定:基于rest模式下webHttpBinding实现和互联网下的浏览器JavaScript进行互通

一:Restful【webHttpbinding】

1. 本质

《1》用URL定位唯一资源

《2》用Http动态描述对该’资源’的操作(CURD)。。。

1. 背景

在移动互联网盛行的时代,出现了各种客户端,比如’安卓’,’IOS’,’小爬虫’,从http模式下的restful就可以对这些客户端实现完整的统一【快语言互通】

1. Restful相比soap和传统模式解决了什么问题？

Eg:比如Student表的修改:

原来的模式是:

<http://localhost:80/Student/Get/1>

<http://localhost:80/Student/Add>

<http://localhost:80/Student/Modify>

<http://localhost:80/Student/Delete>

【看到设么现象?（我们发现url中有动词出现？）】

而使用Restful模式后:

<http://localhost:80/Student/1> [get]

<http://localhost:80/Student> [post]

<http://localhost:80/Student> [put]

<http://localhost:80/Student> [delete]

可以看到,url中不出现任何动词,

除了get,其他类型的url保持一致

这个就是restful实现的动词转移……

Restful的标准就是在url中不可以出现动词….【标准】

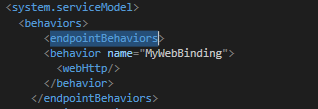
1. Soap更加专注于企业内部网(erp,oa),更加关注安全,事务细节,由于内部网,网络带宽不用关心。Restful更加关注于internet上的网络传输,所以比soap更加的节省带宽.

具体代码实现:

1. 更换相应的binding



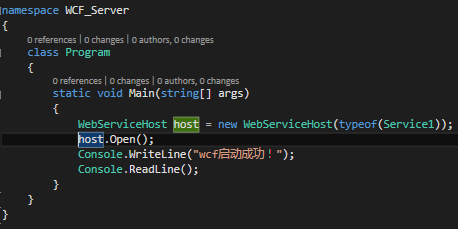
1. 增加Behaviors配置



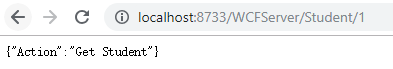
1. 引用using System.ServiceModel.Web;的dll文件
2. 服务契约端的配置如下:

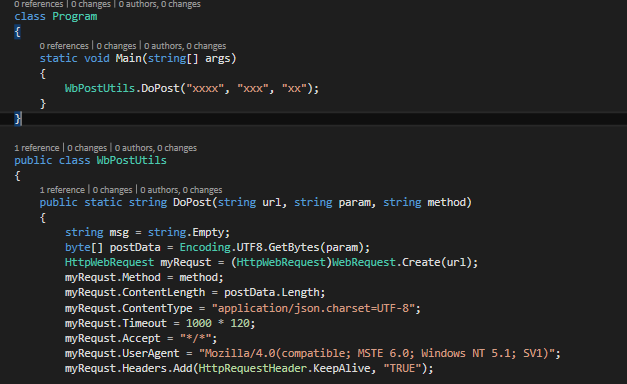


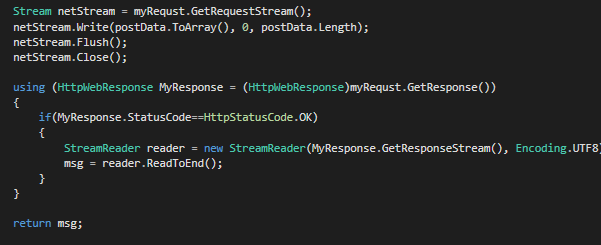
1. 启动WCF(使用WebServiceHost)



代码测试结果如下:



**客户端代码**: 



## 第十二:Host托管:实践WCF的四种托管环境IIS,Winform,Console,Window Server(topself)部署和优缺点分析

一:wcf服务托管

1. windowservice【windows服务】做宿主环境

topself方便部署【生产环境第一选择】

1. console【禁用关闭】

容易调试,禁用关闭按钮,防止误操作

如果在生产环境下WCF的个数在5个以下,使用Console也是很好的.

1. iis【svc】

w3wp进程

1. 创建WCF Application程序适用于IIS
2. 发布
3. 在IIS上创建网站(注意应用程序域是4.0才行)
4. winform/ wpf

可视化

window服务>console>iis>winform/wpf

**如何部署在Windowservice:**

**Topself**

1. **在Topself上面拷贝代码**

**Static void Main(string[] args )**

**{**

**HostFactory.Run(x=>**

**{**

**x.Service<ServiceHost>(s=>**

**{**

**s.ConstructUsing(name => new ServiceHost(typeof(HomeService)));**

**s.WhenStarted(tc => tc.Open());**

**s.WhenStopped(tc = > tc.Close());**

**});**

**x.RunAsLocalSystem();**

**x.SetDescription(“这是我的第一个wcf服务”);**

**x.SetDisplayName(“HomeService”);**

**x.SetServiceName(“HomeService”);**

**});**

**}**

1. **安装部署,用powshell在程序当前目录install**
2. **在Window服务中找到相应的服务,并启动(或者在安装时加入start参数直接自启动)**

我们可以用C#代码和powshell进行代码交互。这样我们就可以做到自动化部署

## 第十三:Behavior行为:教你如何控制服务实例的”并发(单线程,多线程)” 和”实例化(单例,多例)”及WCF自带熔断机制

一:WCF的Behavior【运行时改变wcf的行为】

常用的behavior

服务行为:【ServiceBehavior】 实例行为 + 事务行为 +并发行为+限制行为

端点行为:【EndPointBehavior】 消息检查 + 调度行为

操作行为:【OperationBehavior】 参数筛选

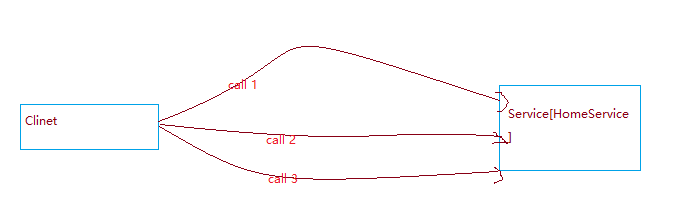
1. 服务行为【ServiceBehavior】

控制实例化 + 并发 +事务

默认情况: Single + PerCall

1. 场景:【实例行为 + 并发行为】

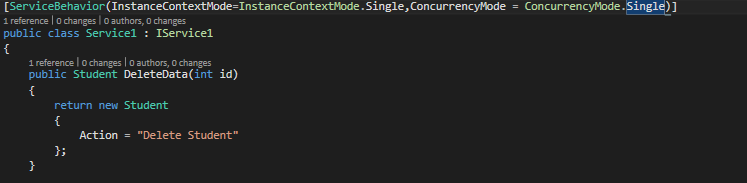
Call 一次 new HomeService 吗?还是只new一次 ???



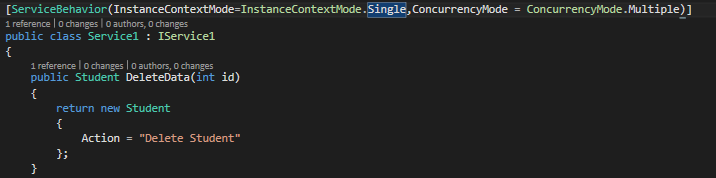
new HomeService()是如何接受Thread的???一个线程访问还是让多个线程访问？

默认情况: Single + PerCall 【来一次将new 一次HomeService,且只接收一个线程】

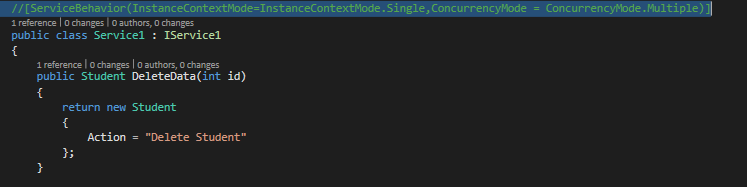
<1> Single +Single 看看请求是不是串行的??? [安全,性能差]



<2>Single +Muti 单实例多线程 =>风险自己控制同步[不安全,性能好]



<3>Percall + Single【默认】[安全,性能中]



备注:安全阀值

3.在构造函数中配置的’安全阀值’,提前评估机器负载和处理能力

当然这个是死值,其实你还可以根据分钟数,cpu当前利用率来动态拒绝客户端请求,这个档次才高。

<behaviors>

<endpointBehaviors>

<behavior name="MyWebBinding">

<webHttp/>

</behavior>

</endpointBehaviors>

<serviceBehaviors>

<behavior>

<serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>

<serviceDebug includeExceptionDetailInFaults="False" />

<serviceThrottling maxConcurrentCalls="1"/>

</behavior>

</serviceBehaviors>

</behaviors>

MaxConcurrentInstances:设置同一时刻服务上下文实例的最大数量。(疑问)【perSession 和perCall 起作用】【对Single不起作用】

MaxConcurrentSessions:设置同一时刻可以接受的最大会话数量

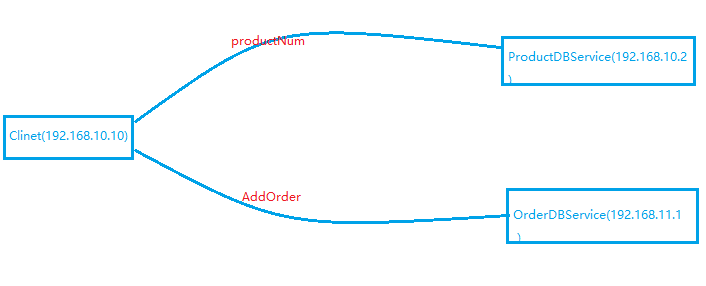
MaxConcurrentCalls:设置ServiceHost消息并发处理的最大值,(疑问)【Single和Multiple控制线程的数量】【对Single也起作用】每个实例来了,都用一个线程池线程,那最多控制在多少个线程忙碌中.

## 第十四:Behavior行为:分布式事务CAP,2PC理论讲解及如何使用WCF进行蛞机器分布式事务处理

一:分布式事务

ACID(单机版的)

将多个节点上的操作纳入一个执行单元里….



CAP理论:

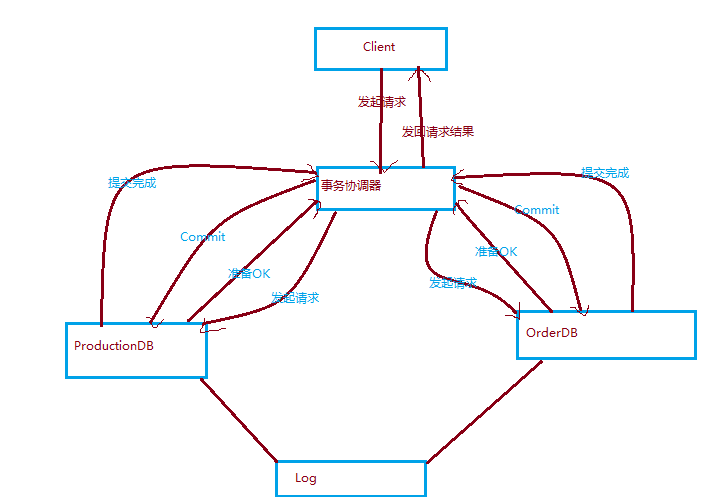
Consistency(一致性) 数据最终都能够落地

Availability (可用性) 理想的响应性能

Partition tolerance(分区容错性) 某些网络阻塞或机器挂掉不影响集群

二:分布式事务实现方式(CAP理论)

1.2PC提交



将一个分布式消息拆解成多个单节点上的CURD,通过事务协调器来进行协调【强一致性】

【准备阶段,执行阶段】,每个阶段都要想事务协调者汇报.（性能低）

三:使用队列进行’分布式事务处理’

1.‘交钱’和’拿饭’是两个动作

为了增加处理量,只需要给交钱的客户一个小票,然后在出货处等待叫号.

只要你有这个小票,就一定能拿到’饭‘【最终一致性】

四:WCF的分布式有DTC参与协调,它就是一个事务协调,相当于2PC中的事务协调器

分布式事务,CAP理论中,追求最终一致性。

五:使用WCF实现分布式事务

步骤:

**服务器端代码：**

1. 服务器契约上需要制定一定需要会话的binding,不是的话,会抛出异常,【ServiceConstract(SessionMode = SessionMode.Required)】
2. 操作契约上一定要指定该操作可以作为事务的一部分,【TransactionFlow(TransactionFlowOption.Allowed)】
3. 实现方法上需要制定该方法纳入事务Transactions事务范围[OperationBehavior(TransactionScopeRequired=true)]
4. Config中制定支持会话的binding,并开启事务流支持

**客户端代码:**

**Using(TransactionScope scope = new TransactionScope(TransactionScopeOption.RequiresNew)){**

**……..**

**scope.commit();**

**}**

**MSDTC window上的服务,必须开启才行。**

## 第十五:Behavior行为:真实案例之自定义IEndpointBehavior和DispatchMessageInspector实现对入站消息的实时统计

一:端点行为 EndpointBehavior

Client------------------------------>EndPointBehavior-------message------->Service

这个有点像MVC的管道………[AOP]动态篡改message…….

二:service方法的实时调用统计【监控大屏】

1.ModifyStudent TotalCount

2.GetStudent TotalCount

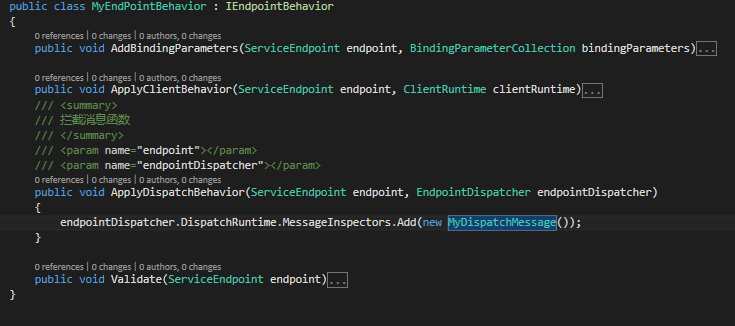
3.AddStudent TotalCount

服务降级 【一分钟之内,某些函数调用量太大,我们可以直接回绝此操作……】

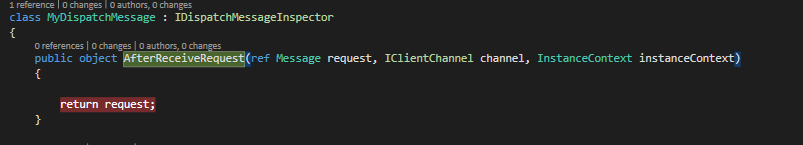
三:演示:

第一步新建Lib类库:

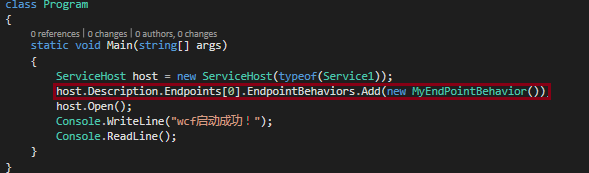
1. 实现IEndPointBehavior接口
2. 在ApplyDispatchBehavior函数中操作



1. 新建实现IDispatchMessageInspector接口的类
2. 在AfterReceiveRequest函数中将得到client端传递过来的request信息



1. 在主函数中添加EndPointBehavior端点拦截自定端点对象.



## 第十六:Behavior行为:自定义OperationBehavior通过IParameterInspector实现对方法参数进行注入检查和代码实战

一:操作行为(OperationBehavior)【AOP编程】

1. 作用域

Public class HomeService

{

[OperationBehavior]

public void DoWork(string username)

{

}

}

mvc:Route(“home/index/{id:int}”)

webget,webInvoke

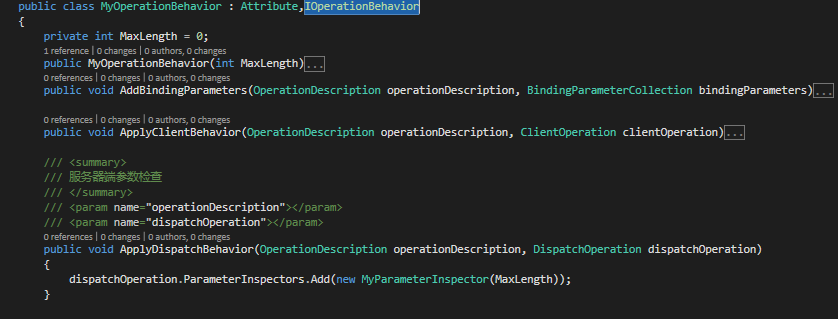
1. 参数检查器 ParameterInspectors

[OperationBehavior] > Dowork > [OperationBehavior]

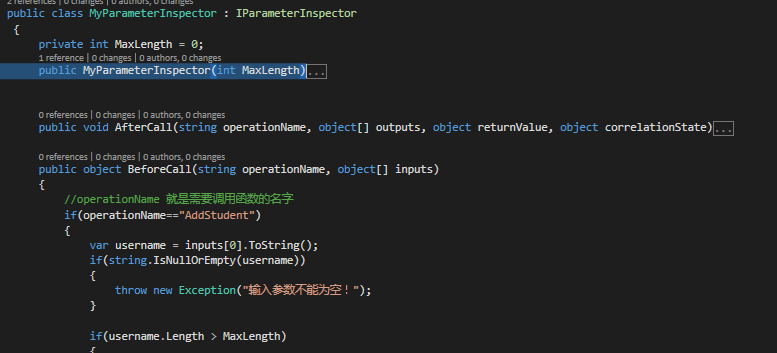
比如:可空检查,正则检查

二:**实现**

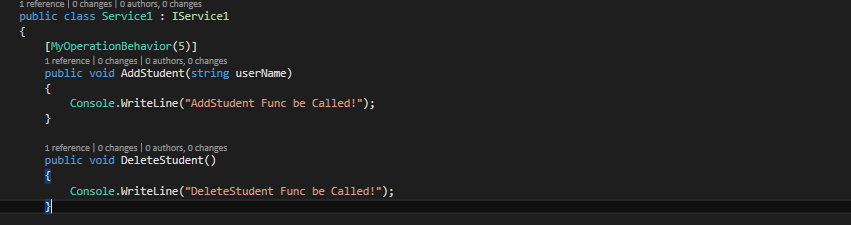
《1》新创建Lib继承IOperationBehavior接口



《2》创建MyparameterInspector 去继承IParameterInspector接口 :

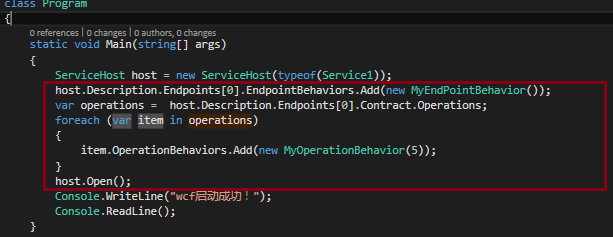


《3》在实现契约的类上加上相应的特性:



以上操作是在函数上加上相应的特性。如果是基础验证的话,就需要在每一个函数上面加对应的特性这样将会很麻。可以通过下面硬编码实现:

**硬编码:**

****

## 第十七:Security安全:巧用IEndpointBehavior在Client和Server对message进行零注入拦截修改实现身份验证

一:WCF安全验证

安全:任何人都可以调用….

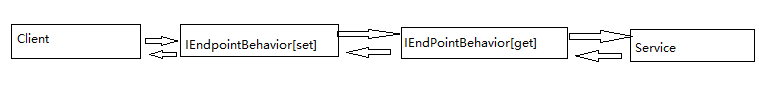
消息级和线路级…SSL证书

1. 所有的wcf服务,不要使用证书….一个证书很贵
2. 非证书使用方式

如果你的wcf需要对外,那么增加验证是必须的….

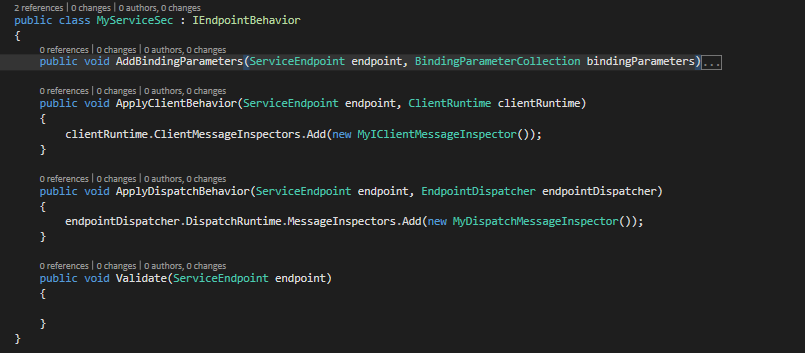
1. 如果对方是C#程序,可以在client和service端在header中增加参数再使用消息拦截,这样两边都不需要额外代码.

Head 头部加上用户名和密码

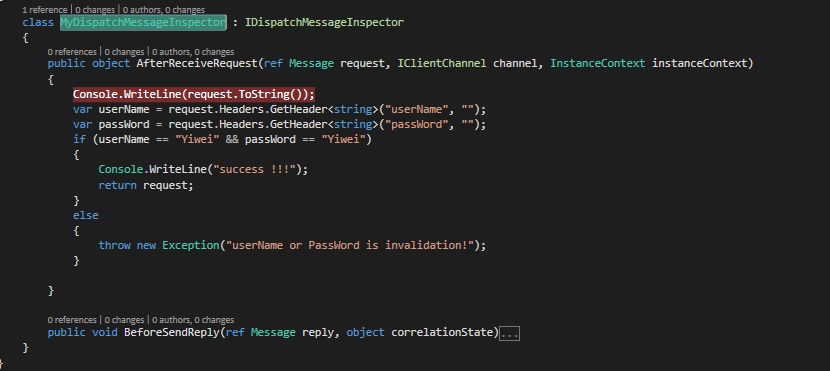


**代码案例如下:**

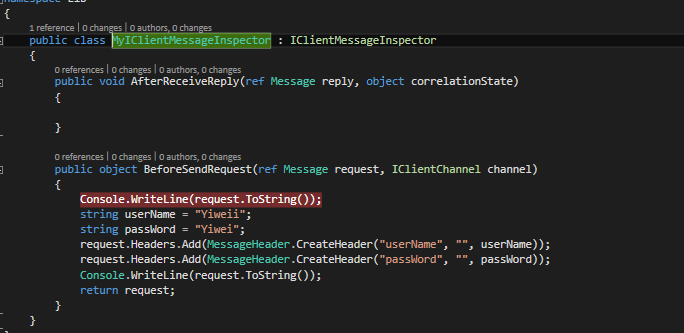
1. 新建Lib继承IEndpointBehavior接口



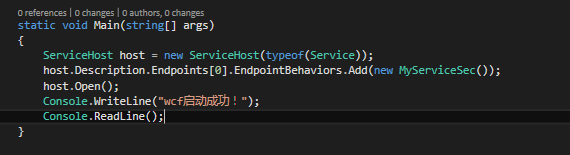
实现服务器端IDispatchMessageInspector接口:



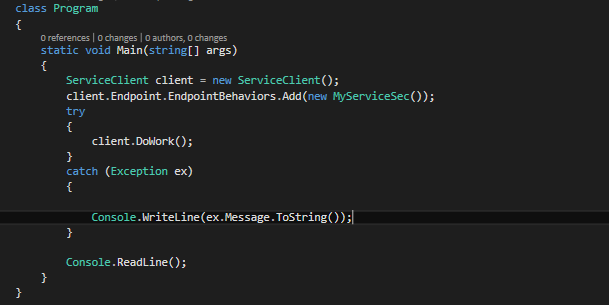
实现客户端MyIClientMessageInspector接口:

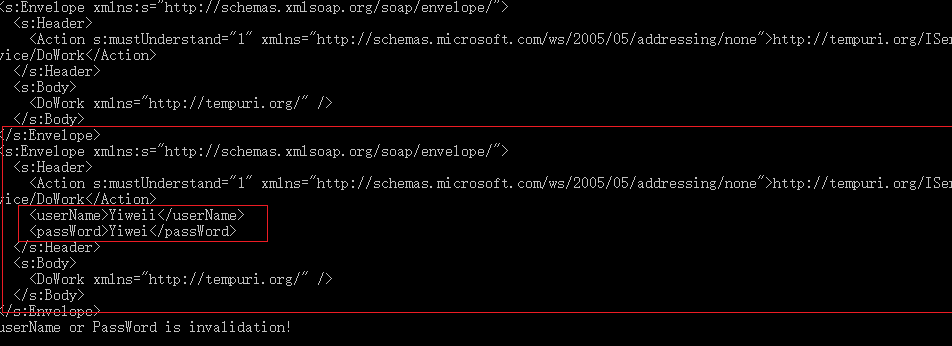


1. 服务器端加入创建的MyServiceSec类



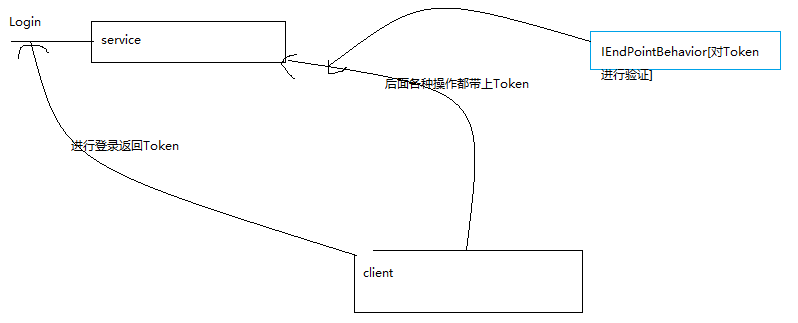
1. 客户端加入创建的MyServiceSec类



**客户端测试结果如下:** 

Java程序,使用用户名和密码,获取token,以后每个接口中都传输一个token参数进去

1. 如果对方不是C#程序,可以使用一个登陆接口,获取token参数之后,client在以后的每个调用中都需要传递token参数



## 第十八:LoadBalance负载均衡:使用反向代理Nginx实现基于basicHttpBinding下的wcf负载均衡和xmind图分析

一:WCF集群

.net系统下一个整合框架【治理框架】

Java【spring cloud dubbo】服务限流,降级,服务发现,服务注册,监控….

好玩的东西都在其云上。

1. 好处

《1》.分流请求流量

《2》.避免单机拓机

《3》.方便升级

1. 场景:

百草味5000w订单量,120G大内存,80G数据,单体全内存查询,优化算法,速度最快(20s-30s)…..

Hash算法0（1）,不敢用,直接用数据代替hash

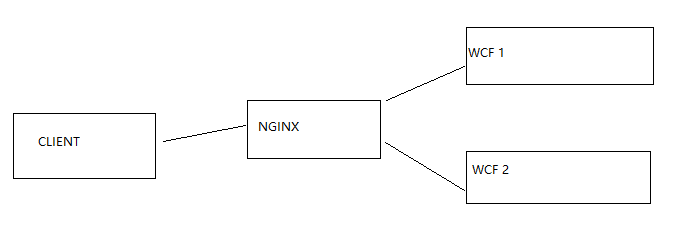
Spark [hadoop]

1. 如何搭建 nginx

在CentOS

1. 创建文件夹
2. 在文件夹下面wget <http://xxxxxxx>
3. 解压tar-xzvrf
4. 安装
5. 在相关路劲下找到.config文件

./configure –prefix=/user/apps/nginx && make && make install



在网上搜索,因为存在一些依赖项~

## 第十九:LoadBalance负载均衡:使用redis的HashSet集中化管理wcf集群下的所有服务连接实现前段分流代码实践

一:nginx缺陷

1. 使用wcf大多还是同语言内通信,所以更多采用效率更高的tcp通信。

Nginx开源版只适合http通讯….

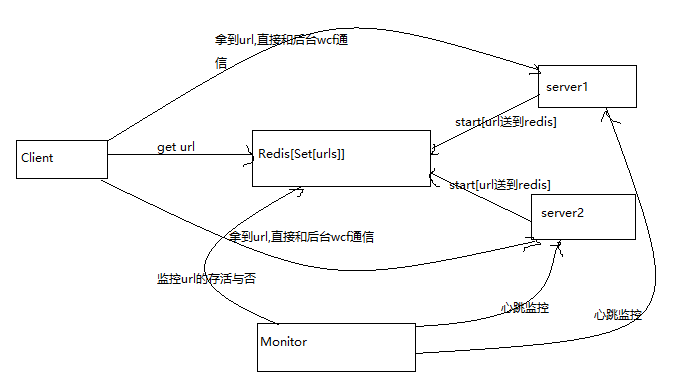
1. 我们把所有的url扔到redis中,然后clinet去读取

二:使用redis作为urls池

《1》强依赖redis,那么redis可以做一个master -slave….

《2》redis访问次数.[client端缓存几秒钟]

《3》还需要监控url的列表



三:搭建

1. 下载
2. 在centos上部署
3. Wget 下载
4. 解压 tar –xzvf
5. make
6. 下载C#的驱动
7. 修改配置文件

《1》#127.0.0.0

《2》protect mode –on

注意:防火墙需要关掉(firewall –cmd off ?)

C#驱动 StackExchange.Redis

四:zookeeper优化

## 第二十:Optimize优化:使用google编码器ProtoBuf替代DadaConstactSerizer优化WCF的XML infoSet实现最小化网络传输

一:wcf 序列化问题

ws标准流行:xml

二:protobuf【序列化编码器】

1. ProtoBuf,全称是Protocal Buffers,它是谷歌内部用的一种高效的,可扩展的对结构化数据进行编码的格式规范。

不是xml，也不是json

三:场景

报表数据太多下的优化

数据压缩:

【gzip】 > 【bitArray 代替 List】 > 【protobuf】

四:原始 vs Protobuf-net

代码在Entity中

[protoConstract]

Public class Person\_Entity

{

[protoMemerber]

Public string Name{get;set;}

}

## 第二十一:Optimize优化:使用携程开源的集中化配置框架Apollo实现WCF集群下的AppConfig管理和代码实践

一:Apollo集中化配置

1. 分布式环境下的痛点

10台wcf….配置文件是不是要修改10次？

<1>配置分散,修改起来麻烦

ConnectionString

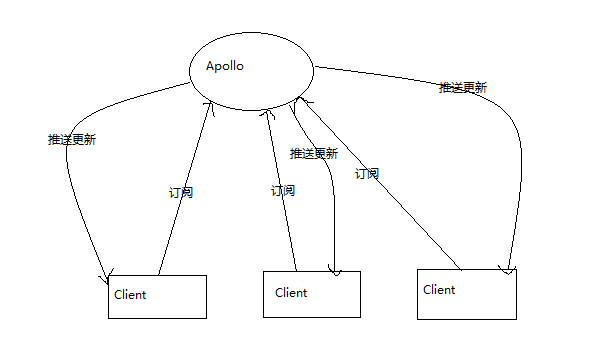
MongodbString

RedisString

将分散的wcf配置集中化管理

如何搭建:

前提:我们需要mysql centos java



详情查看网上开源代码

## 第二十二:WCF新功能:介绍基于webSocket协议的NetHttpBinding和基于UDP和UdpBinding代码实践

一:WCF新功能

1.websocket通信使用NetHttpBinding

Html 5

Websocket 和 http之间的关系 #他们不是一个东西

他们之间就是一个交集的关系

Websocket-------------长连接

双工【两个oneway的操作】

始终 – 强制使用 WebSocket.

从不-阻止使用WebSocket,尝试使用具有此设置的双工协定将导致异常

2.udp通信 UdpBinding