# 概述

## 编制目的

通过本地发布 ESB 服务， 来管理平台所有webservice接口，及定制平台输入、输出XML 规范，达到webservice集中部署，集中管理，统一规范的目的

## 技术介绍

### 有关ESB

企业服务总线是一种软件架构的构造，生存在（业务）应用和通信之间。理想情况下，ESB总线上的应用程序能够代替所有直接接触使所有通信通过ESB。 ESB通过一个事件驱动和基于标准的消息引擎（总线）提供基本服务。多亏ESB，集成架构师可以利用消息的价值而无需编写代码。开发人员通常使用一类公认标准的中间件基础设施产品技术实现ESB。

ESB是一个比较新的概念。仅仅是在最近几年被普遍使用。人们通常将ESB和面向服务的体系结构联系在一起但他们有正交的关系。然而，如SOA、ESB本质上是许多企业软件产品直接实现的企业架构设计模式的集合。 WSO2 ESB是一种根据ApacheV2.0许可证发布的快速、轻量级和灵活的企业服务总线产品。使用ESB在HTTP、HTTPS、JMS、mail等协议基础上通过业务系统过滤、转换、路由和处理SOAP，二进制、纯XML和文本消息。

### wso2是什么？

WSO2企业服务总线（ESB）是一个快速，轻巧，100％开源的和用户友好的开源ESB，分布按照Apache软件许可证2.0版。WSO2 ESB允许系统管理员和开发消息路由，调解，改造，记录，任务调度，故障切换路由，负载均衡和更方便地配置。它还支持传输交换，事件，以规则为基础的调解和基于优先级的先进的集成要求调解。ESB运行时的设计是完全异步，非阻塞上Apache Synapse的调解的引擎和基于流的。 WSO2ESB是基于WSO2 carbon平台之上发展而来的，一个基于OSGi的SOA通过组件化的框架提供了无缝的模块化开发。此版本还包含了许多新的功能和一系列的可选组件（附件）可以安装自定义ESB的行为。此外，ESB任何现有的而环境不需要的功能都可以很容易的利用底层配置框架的carbon移除。

### ESB体系架构（ESB Architecture）

在企业中的基础应用设施可能是本身就很复杂，包括数百种语义完全不同的应用程序。其中的一些应用是自定义的，一些是从第三方获得还有一些是两者的结合，而且他们可以在不同的系统环境中运行。

这些异构应用之间的集成对企业来说是至关重要的，不同的服务可以使用不同的数据格式和不同的通信协议。服务的物理位置可以任意改变，所有的限制意味着应用程序任然是紧密耦合在一起的。

ESB可以用来解耦和连接在一起的服务和服务消费者。

WSO2ESB是一个为企业准备的完全成熟的ESB。WSO2ESB是建立在Apache Synapse项目基础上的。Apache Synapse是使用Apache Axis2创建的。

下图从信息角度描述了WSO2ESB架构，还有一些像Tasks、Events没有在图中描述出来。显示了如何在ESB中处理消息。

注意：管道的各种组成部分并不意味着这个图中的热河命令。

应用程序发送消息到ESB。

该消息由ESB Transport捡起。

Transport通过消息管道发送消息。像安全和可靠的消息传递的信息方面的质量受到这个pipe的照顾。在该pipe内部是axis2的流入和流出流。ESB可以有如下两种操作：

消息中介：使用单管道

代理服务：使用独立的管道运输到不同的代理服务。

消息转换和消息路由可以看做一个独立的单元。如图所示，消息转换组件和路由组件之间没有明显的分离。WSO2ESB调用这个中介框架。一些转换发生在路由决定之前，一些转换发生在路由决定之后。这一部分由Synapse执行。

然后根据目的地将消息注入到独立的管道。在这里再次确定消息服务方面的质量。

传输层负责通过ESB所需的传输协议的转换。

该图显示了如何通过ESB的体系架构将请求传到一个实际的endpoint。响应处理是这个操作的反向操作。

所有这些组件可以通过WSO2ESB管理控制台管理和检测。

# 开发指南

## 部署 WSO2 ESB

部署环境：Linux redhat6 ，JDK1.6 ，wso2esb-4.8.0

* 安装 linux 系统
* 在 linux 系统上配置 jdk1.6 环境变量。经测试：wso2esb jdk 版本为 1.6-1.7 ，其他版本不行

|  |
| --- |
| Cd ~  Vi ./.bash\_profile  PATH=$PATH:$HOME/bin  JAVA\_HOME=/home/esb/Desktop/jdk1.6.0\_45;export JAVA\_HOME  PATH=$JAVA\_HOME/bin:$PATH  export PATH |

配置 CARBON\_HOME

|  |
| --- |
| Cd ~  Vi ./.bash\_profile  PATH=$PATH:$HOME/bin  JAVA\_HOME=/home/esb/Desktop/jdk1.6.0\_45;export JAVA\_HOME  CARBON\_HOME=/home/esb/Desktop/wso2esb-4.8.0;export CARBON\_HOME  PATH=$JAVA\_HOME/bin:$CARBON\_HOME/bin:$PATH  export PATH |

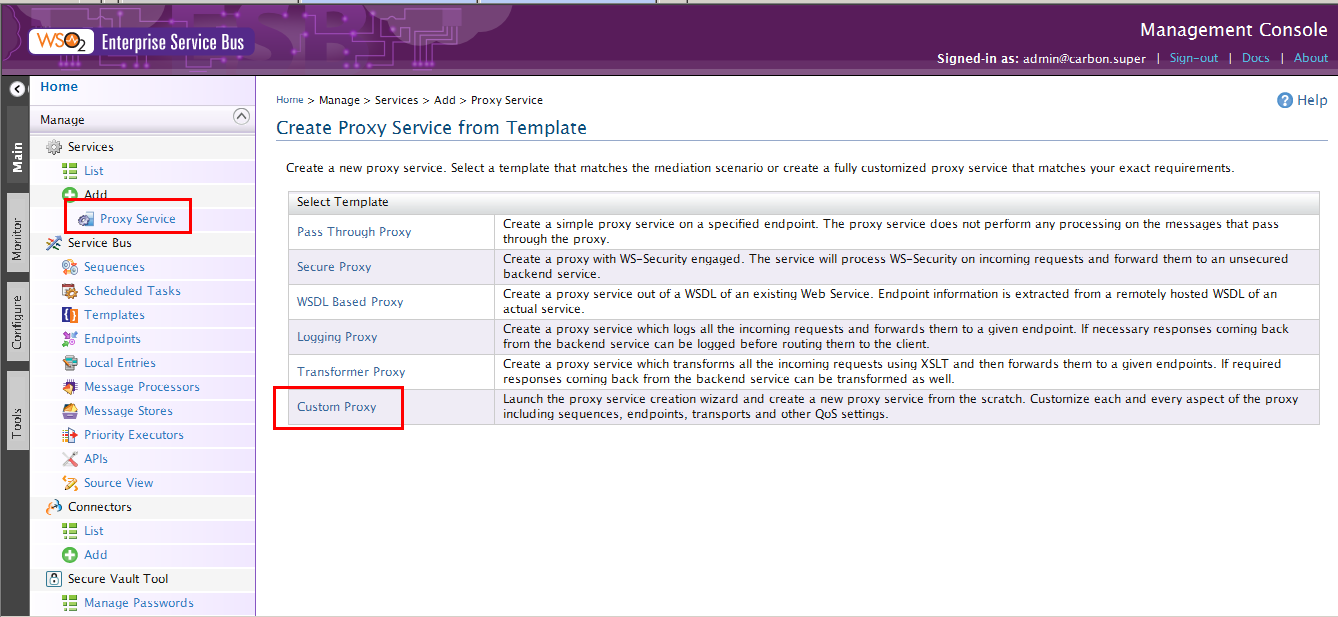
* 修改 repository\conf\axis2\axis2.xml 中bind\_address 和 WSDLEPRPrefix，改后在发布的webservice 中就会使用服务器 ip 来访问 esb，否则默认是服务器主机名，若没有添加 hosts 内容， 则不会访问到 webservice 。

|  |
| --- |
| Cd $CARBON\_HOME/repository/conf/axis2  Vi axis2.xml  <transportReceiver name="http" class="org.apache.synapse.transport.passthru.PassThroughHttpListener">  ...  <parameter name="bind-address" locked="false"> 192.170.100.15</parameter>  <parameter name="WSDLEPRPrefix" locked="false">http://192.170.100.15:8280</parameter>  ...  </transportReceiver>  <transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener">  ...  <parameter name="bind-address" locked="false"> 192.170.100.15</parameter>  <parameter name="WSDLEPRPrefix" locked="false"> https://192.170.100.15:8243</parameter>  ...  </transportReceiver> |

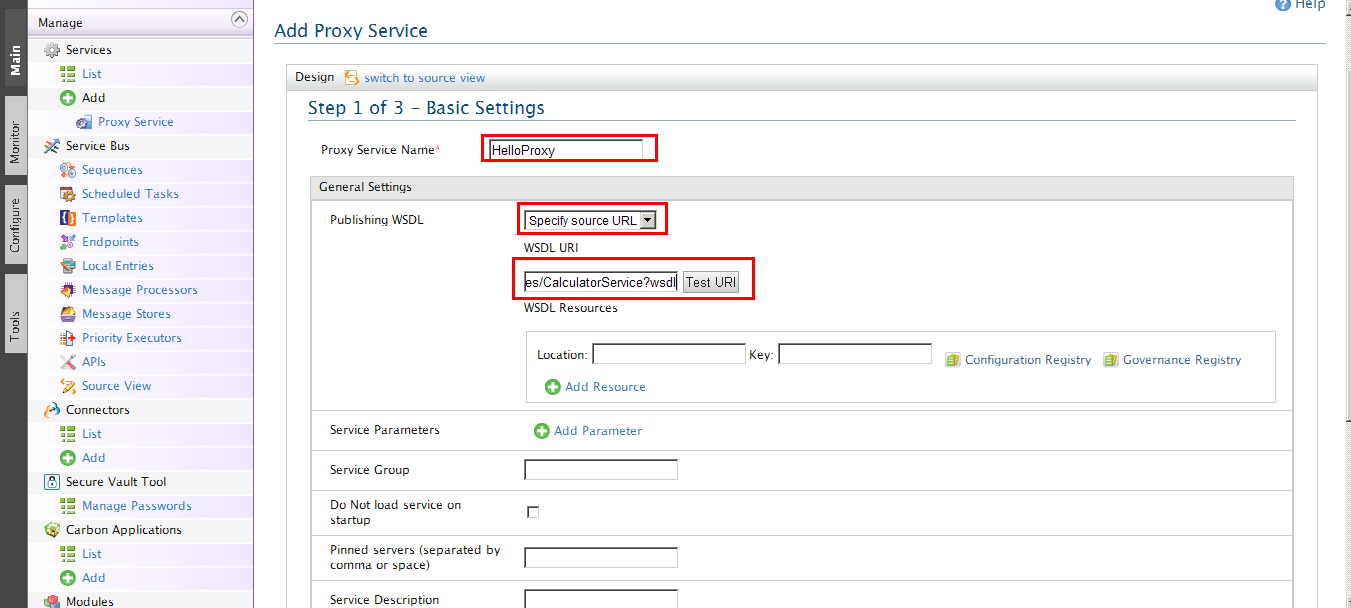
* 启动 wso2server.sh
* 访问地址<https://192.170.100.15:9443/carbon>
* 登录账号及密码：admin/admin

## 发布webservice至ESB

1. 创建 Proxy



1. 创建 proxy 名称， 并载入 wsdl 文件



Wsdl 路径可以是一个http路径，也可以是一个 repository 路径；如下

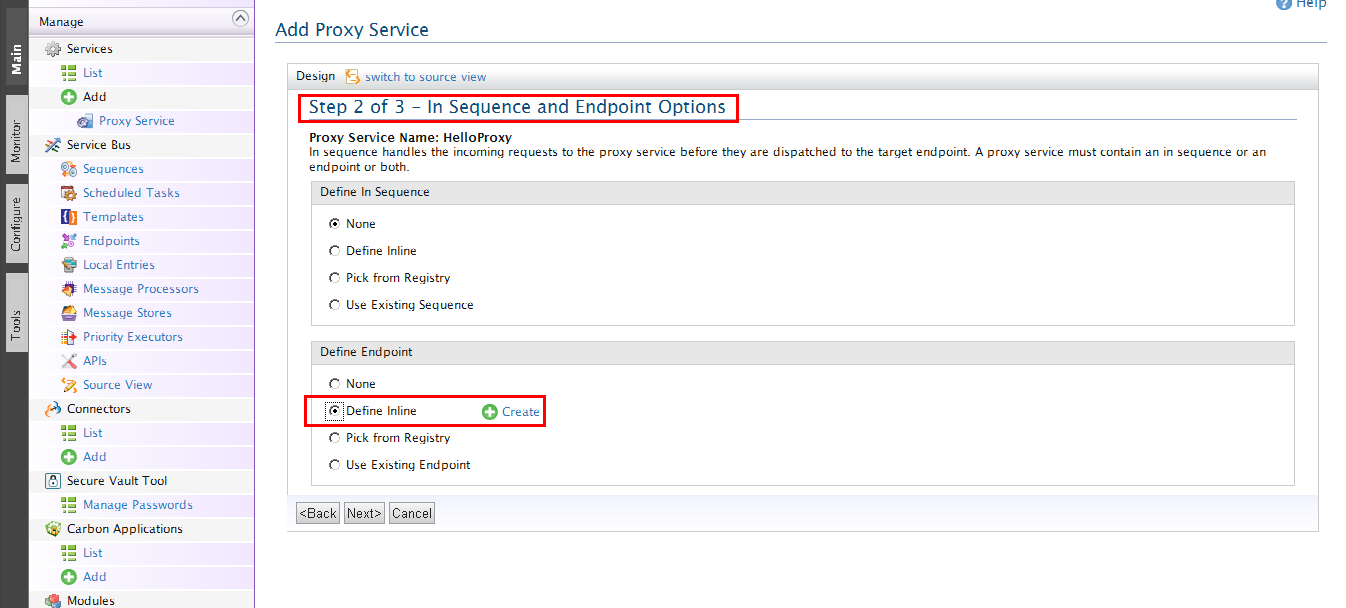
* <http://192.170.100.102:8088/myxfire/services/CalculatorService?wsdl>
* file: repository\samples\resources\proxy\sample\_proxy\_1.wsdl

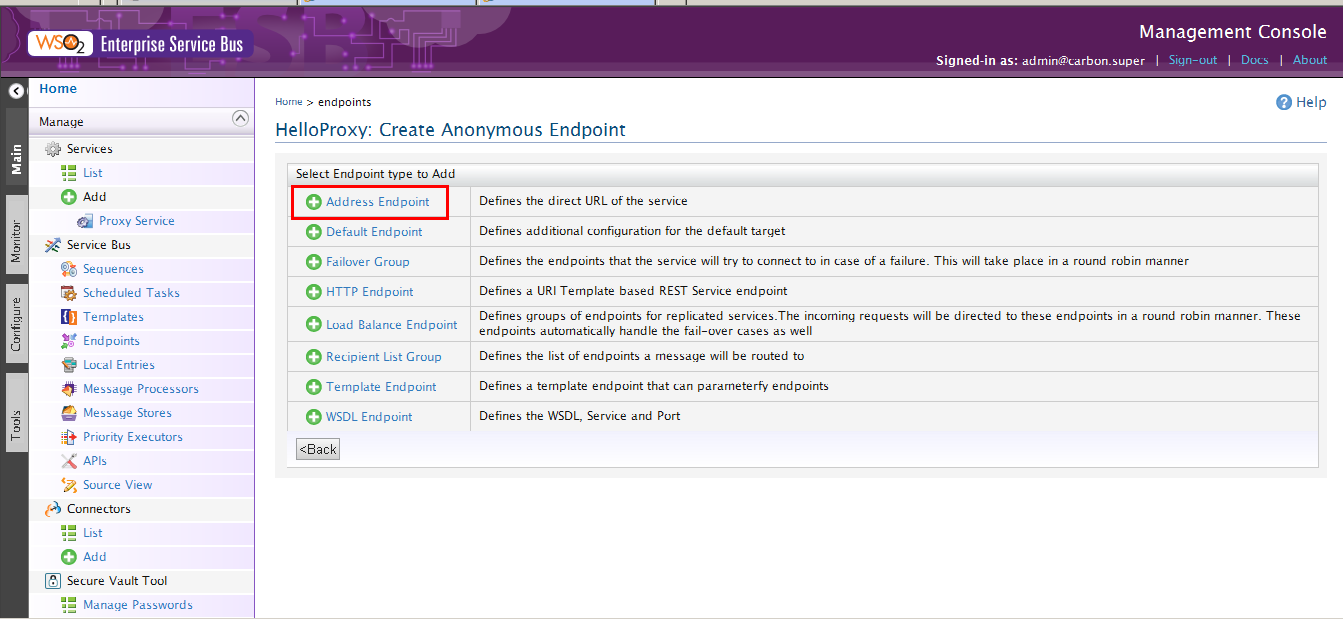
勾选 http 协议



**完成后点击 “NEXT”**

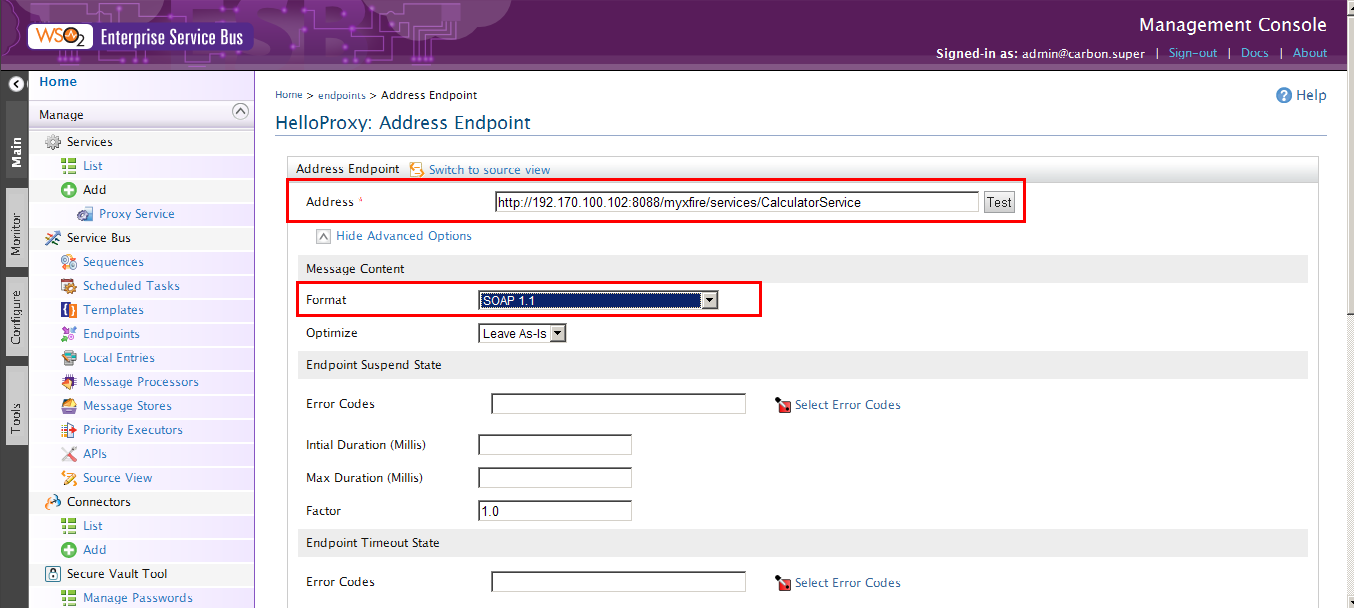
1. 创建 EndPoint





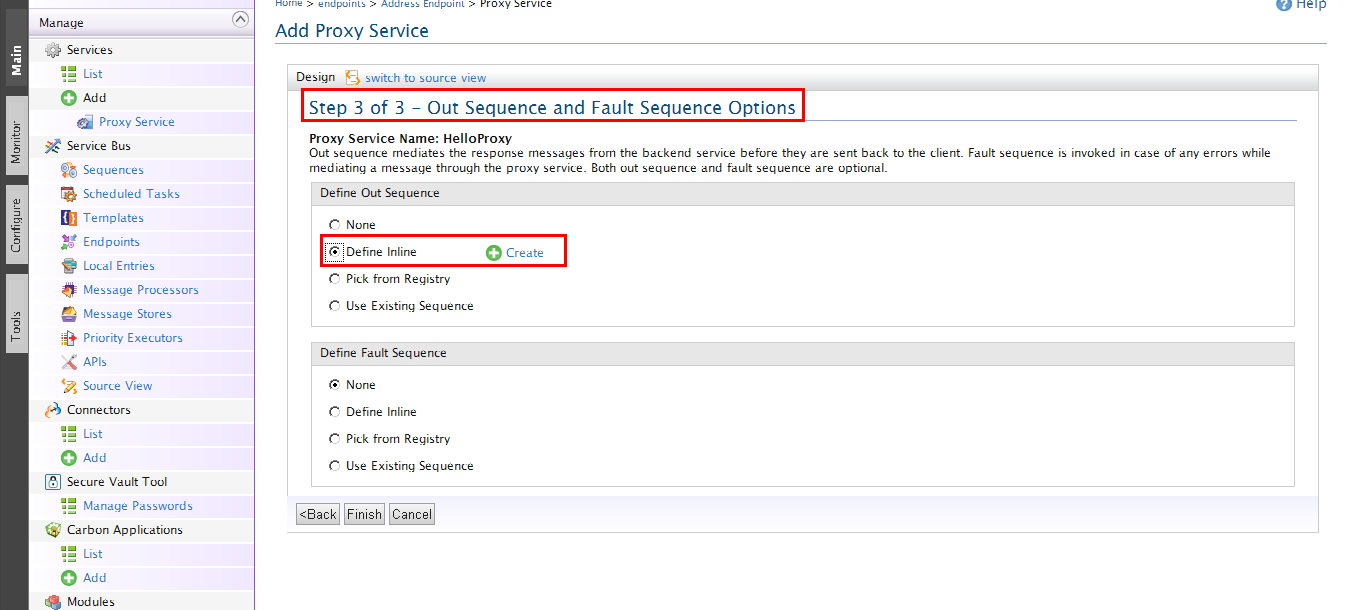
Address：webservice 处理地址

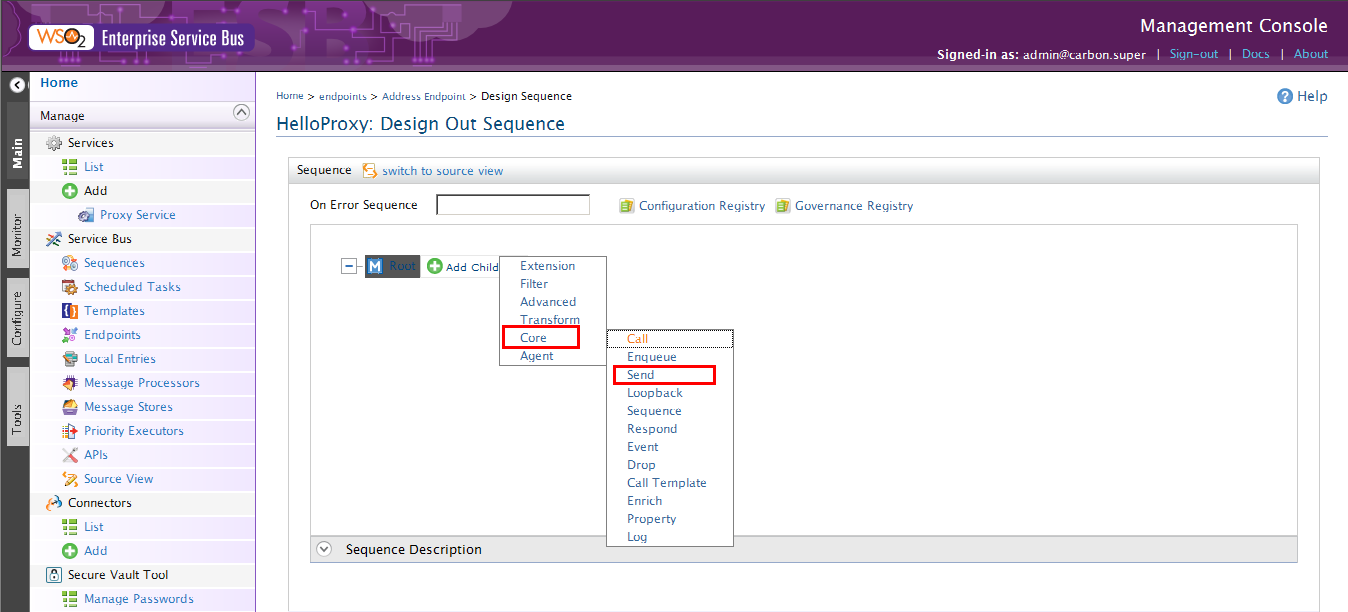
Format：选择 SOAP1.1 协议



**完成后点击 “NEXT”**

1. 创建 Out Sequence

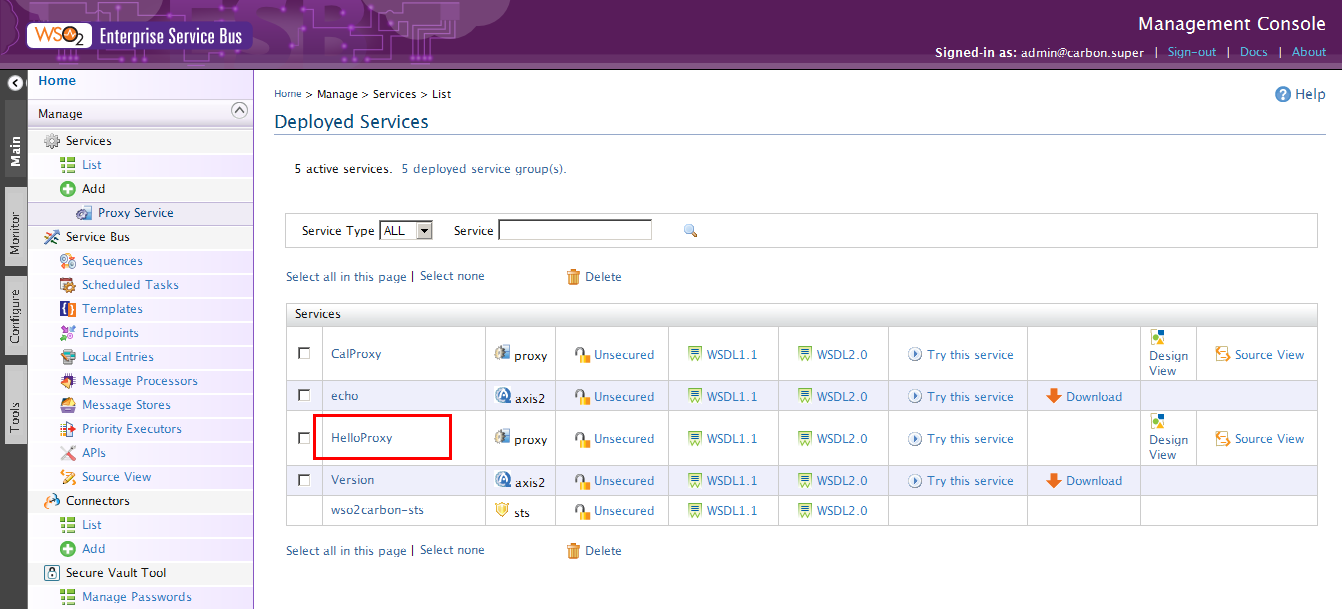


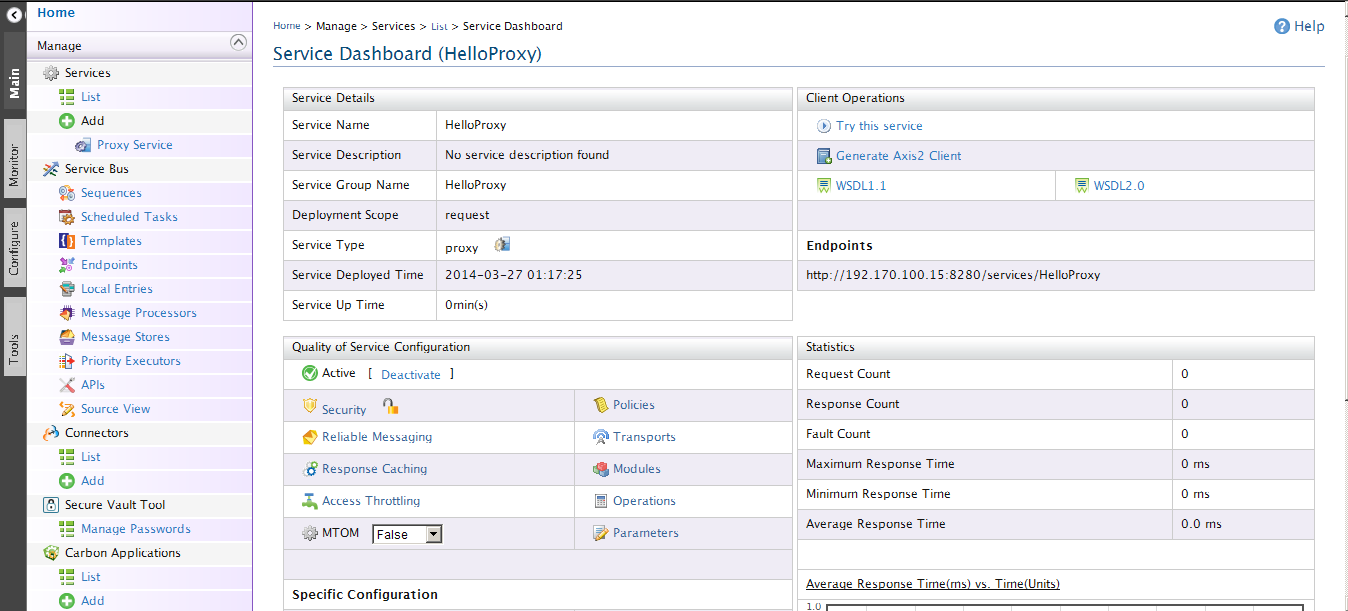


**完成后点击 “FINISH”**

1. 查看 Proxy 信息

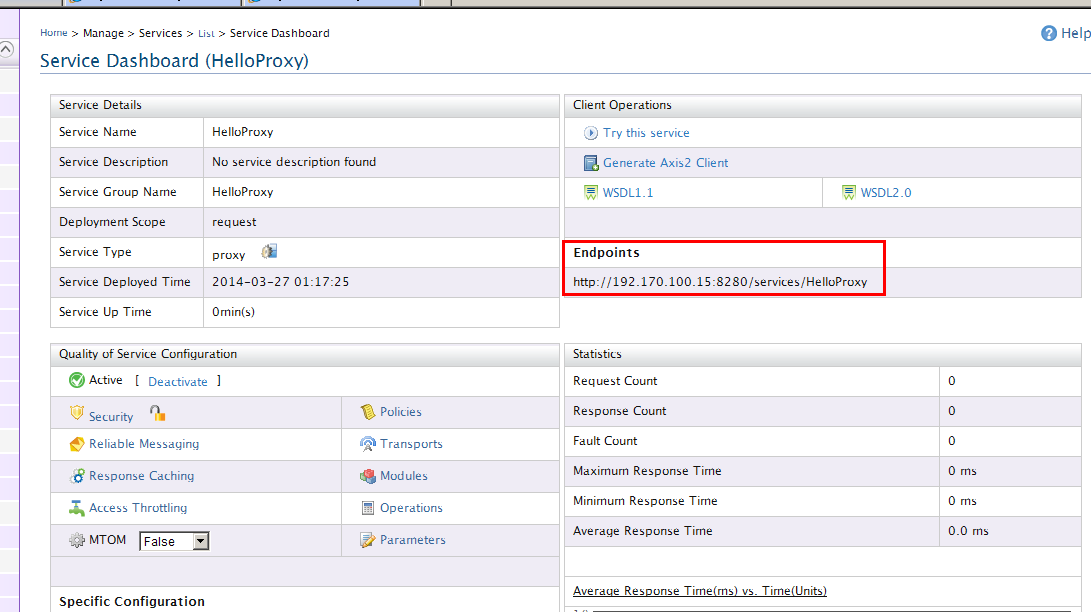
点击 HelloProxy





可以再此处对 webservice 进行监控

1. EndPoint 地址



<http://192.170.100.15:8280/services/HelloProxy>

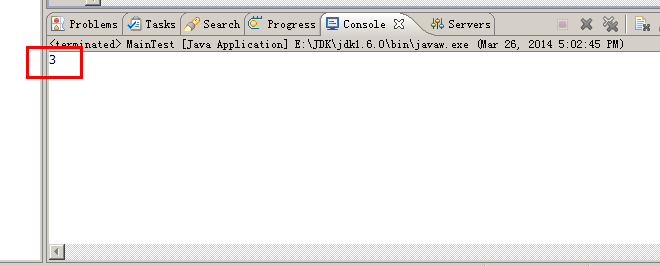
此地址即 http://192.170.100.102:8088/myxfire/services/CalculatorService发布至 ESB 后代理地址，可以通过访问HelloProxy地址，来达到访问 CalculatorService Webservice 的效果。

1. 开发客户端程序，调用ESB

使用 xfire 客户端访问 ESB ，载入所需 jar 包，开发 client 代码

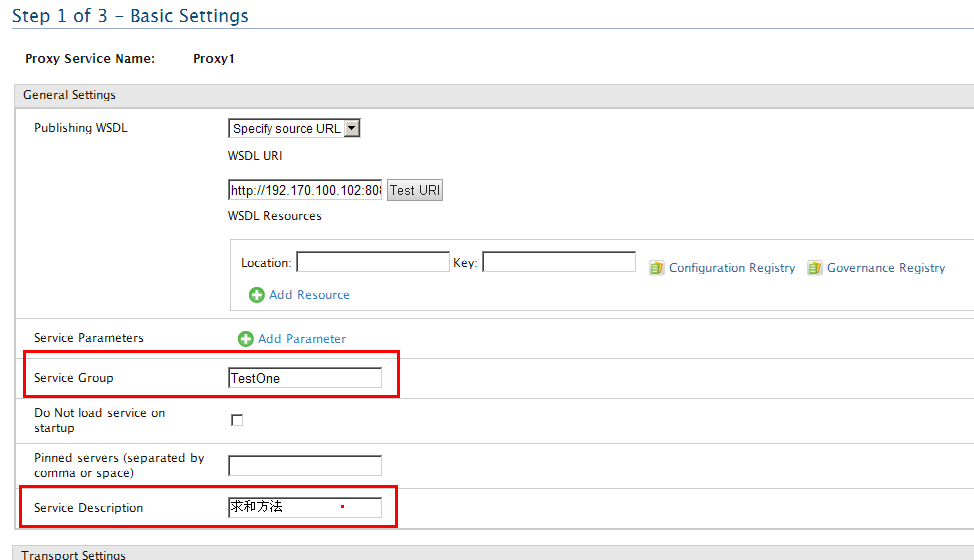
|  |
| --- |
| public static void main(String[] args) throws AxisFault {  try {  Client client = new Client(new URL("http://192.170.100.15:8280/services/HelloProxy?wsdl"));  Object[] obj = client.invoke("add", new Object[]{1,2});    System.out.println(obj[0]);  } catch (Exception e) {  // TODO: handle exception  e.printStackTrace();  }  } |

输出信息为：

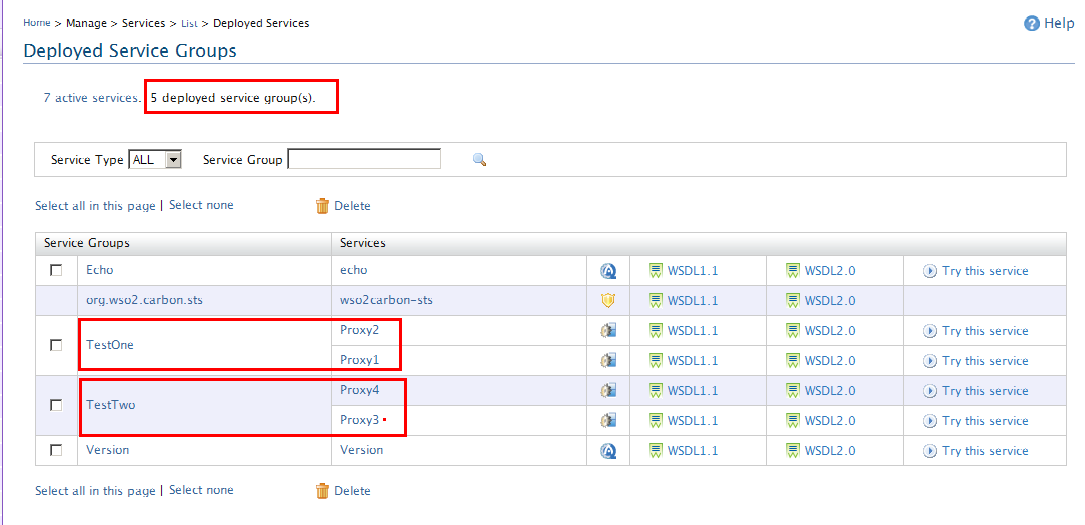


## 分组管理发布代理

当esb 上代理服务非常多时，可以采取分组管理services。

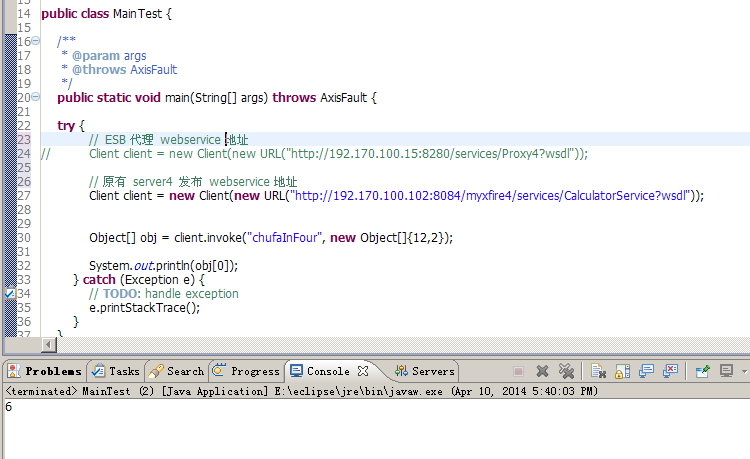


在 Step 1 时，键入 Service Group 的名称，其他有同组代理时，填写相同的 Group 名称即可。Service Description 属性是对此代理的备注描述。



点击 Deployed service group(s) 可查看各 Groups 及其下属代理。

## Client交叉调用代理



交叉调用代理是指按照原有旧模式，若想在一个工程内部调用另一个工程的webservice 时，wsdl 路径为 myxfire4 发布的路径，搭建 ESB 后，直接调用 ESB 发布的 Proxy4 地址亦可调用 myxfire4 的webservice，统一出入口进行 services的管理。

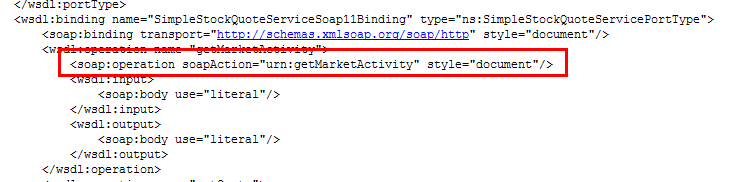
# 问题解析

|  |
| --- |
| org.codehaus.xfire.fault.XFireFault: The endpoint reference (EPR) for the Operation not found is /services/CalProxy.CalProxyHttpSoap11Endpoint and the WSA Action = . If this EPR was previously reachable, please contact the server administrator.  at org.codehaus.xfire.fault.Soap11FaultSerializer.readMessage(Soap11FaultSerializer.java:31)  at org.codehaus.xfire.fault.SoapFaultSerializer.readMessage(SoapFaultSerializer.java:28)  at org.codehaus.xfire.soap.handler.ReadHeadersHandler.checkForFault(ReadHeadersHandler.java:111)  at org.codehaus.xfire.soap.handler.ReadHeadersHandler.invoke(ReadHeadersHandler.java:67)  at org.codehaus.xfire.handler.HandlerPipeline.invoke(HandlerPipeline.java:131)  at org.codehaus.xfire.client.Client.onReceive(Client.java:406)  at org.codehaus.xfire.transport.http.HttpChannel.sendViaClient(HttpChannel.java:139)  at org.codehaus.xfire.transport.http.HttpChannel.send(HttpChannel.java:48)  at org.codehaus.xfire.handler.OutMessageSender.invoke(OutMessageSender.java:26)  at org.codehaus.xfire.handler.HandlerPipeline.invoke(HandlerPipeline.java:131)  at org.codehaus.xfire.client.Invocation.invoke(Invocation.java:79)  at org.codehaus.xfire.client.Invocation.invoke(Invocation.java:114)  at org.codehaus.xfire.client.Client.invoke(Client.java:336)  at org.codehaus.xfire.client.Client.invoke(Client.java:368)  at xfire.ant.client.MainTest.main(MainTest.java:33) |

1. 若在客户端调用遇到下叙问题，则表示在调用 webservice 的过程中找不到 operation，需要手动给 webservice 指定 action 。下列提供两种解决办法，仅供参考：

* 【ESB端改法，不需改服务端】将生成的 wsdl 下载下来，手动将 action 地址添加至 soapAction, 格式为

“urn:方法名”，例如：



* 【服务端生产webservice改法】 生产webservice的时候，需手动添加 soapAction地址，具体代码如下：
* 官方网站下载最新的xfire包。
* Java 代码写法

|  |  |
| --- | --- |
| |  | | --- | | **package com.ws.test;**  **import javax.jws.WebMethod;**  **import javax.jws.WebService;**  **@WebService(name = "testsoapaction", serviceName = "hahah", targetNamespace = "http://xxoo.com")**  **public class SoapServiceImpl {**  **@WebMethod(action = "urn:Test", operationName = "test")**  **public String example(String message) {**  **return message;**  **}**  **}** | |

* service.xml

|  |
| --- |
| <beans>         <service xmlns="http://xfire.codehaus.org/config/1.0">        <name>HelloService</name>       <serviceClass>com.ws.test.SoapServiceImpl</serviceClass>       <serviceFactory>org.codehaus.xfire.annotations.AnnotationServiceFactory</serviceFactory>    </service>  </beans> |

# 附录结构列表

* wso2esb-4.8.0.zip
* jdk-6u45-linux-x64.bin