SOA Assignment 3,

A E-commerce System Service Model Implementation with WSO2 ESB

13301026杨幸潮

13301025 邢金涛

13301052 延昊南

13301047王会

13301053杨洋

# Part 1: ESB集成

## 一 任务目标

本模块负责是把第二次作业的webservice集成到wso2提供的ESB服务上。使用的是api入口，连接部署在tomcat的webservice。

## 二 项目说明

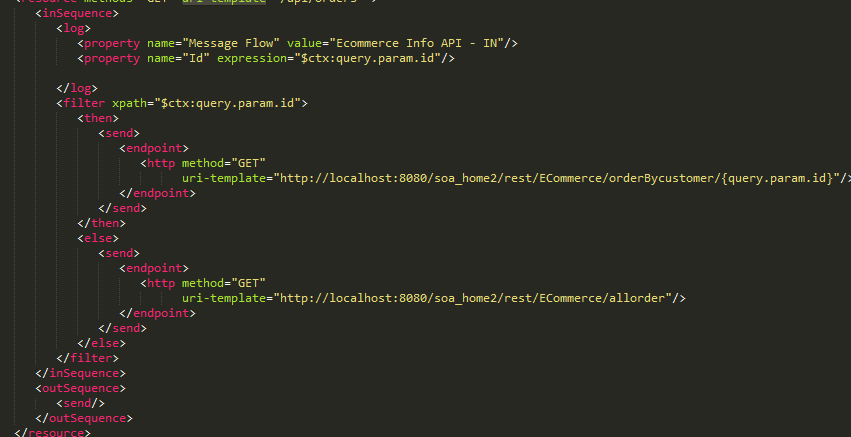
本项目运行时需要启动mysql和tomcat。同时启动wso2，之后用客户端访问api提供的url.,实现esb集成。

## 三 代码分析

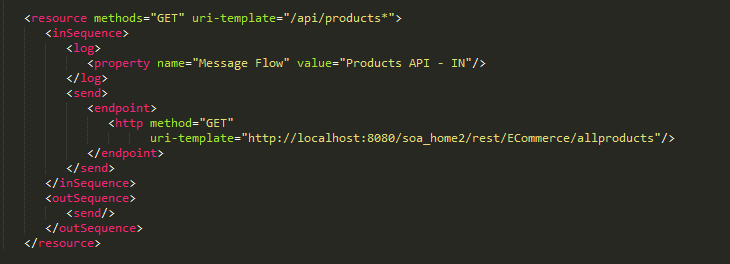
采用api形式集成，定义api的context等等有关信息。



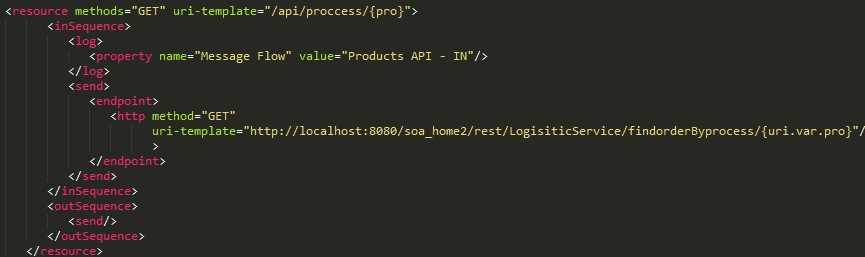
之后就是把每个resource配置进去，设置好insequence和outsequence，以下是显示订单的服务配置，设置filter，来匹配参数id，连接到tomcat里根据id查order的url。如果没有参数id，则连接到tomcat查询所有order的url。



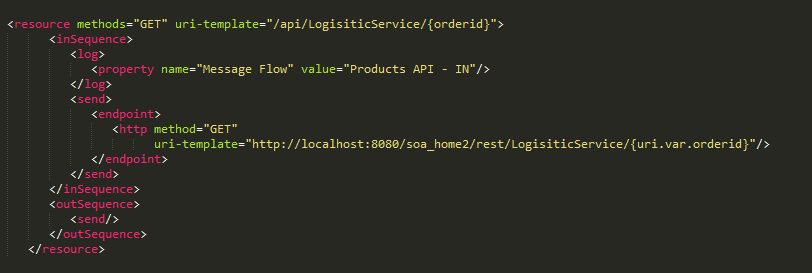
产品服务，列出所有产品



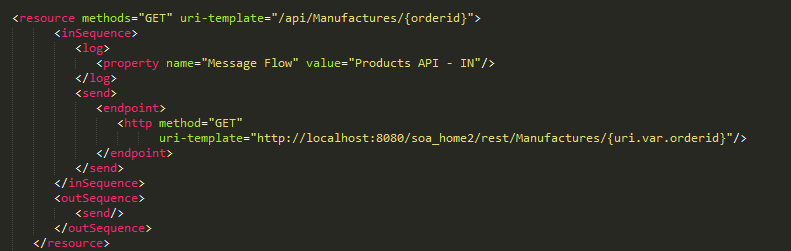
状态的查询服务：



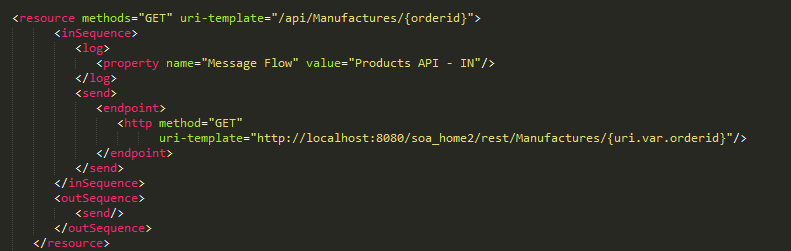
物流查询，根据id查询对应订单的物流状态。



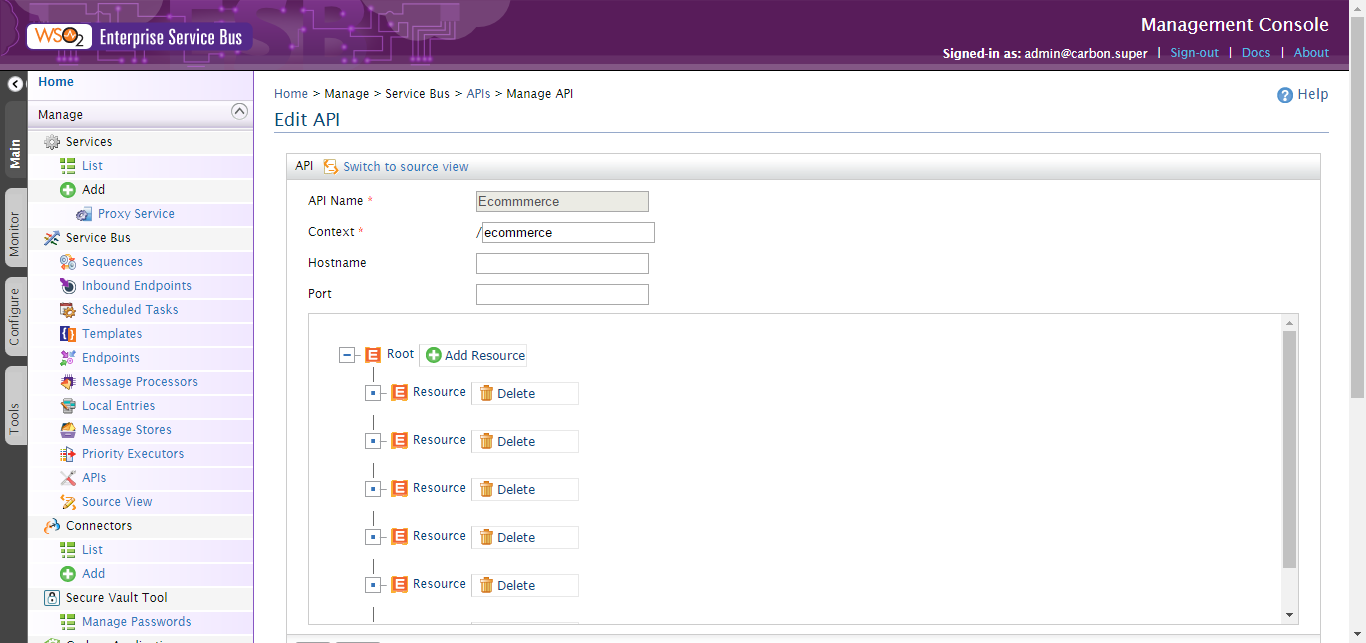
生产产品，减少产品的库存。



添加订单：



WSO2配置：

 esb的url地址：



# Part 2: OAuth2.0

## 一 任务目标

本模块为服务添加OAuth认证。

## 二 项目说明

### 1. 安装&启动

我已经把WSO2 Identity Server(IS)中的关键文件放到了项目中的‘OAuth2相关部分’文件夹中。替换本机的IS文件操作如下

1. 下载WSO2 Identity Server5.1

2. 将本项目的OAuth2相关部分/wso2is-5.1.0/repository/components/default文件夹替换掉下载好的IS中的对应文件夹

3. 将本项目的OAuth2相关部分/wso2is-5.1.0/repository 下的除了default的其他文件夹替换掉下载好的IS中的对应文件夹

4. 将本项目的OAuth2相关部分/wso2is-5.1.0 下的除了repository的其他文件夹替换掉下载好的IS中的对应文件夹

进入wso2is-5.1.0/bin目录后可直接运行脚本启动IS。

由于在esb的InSequence中Oauth配置项的端口是写死的，为9444，所以要保证IS运行所用的端口也为9444，否则要改EcommerceAPI.xml文件，将oauthService中remoteServiceUrl里的端口号改成自己的。

### 2. 使用

系统中已加入Oauth认证，所以在访问api的时候必须在header中带上Authorization=Bearer 92eec608d553120455be6139c97bfe64的形式。key为Authorization，value为bearer+空格+access\_token

如果没用携带access\_token或者携带了错误的access\_token则不会返回正确信息

### 3. 如何获取access\_token

启动IS

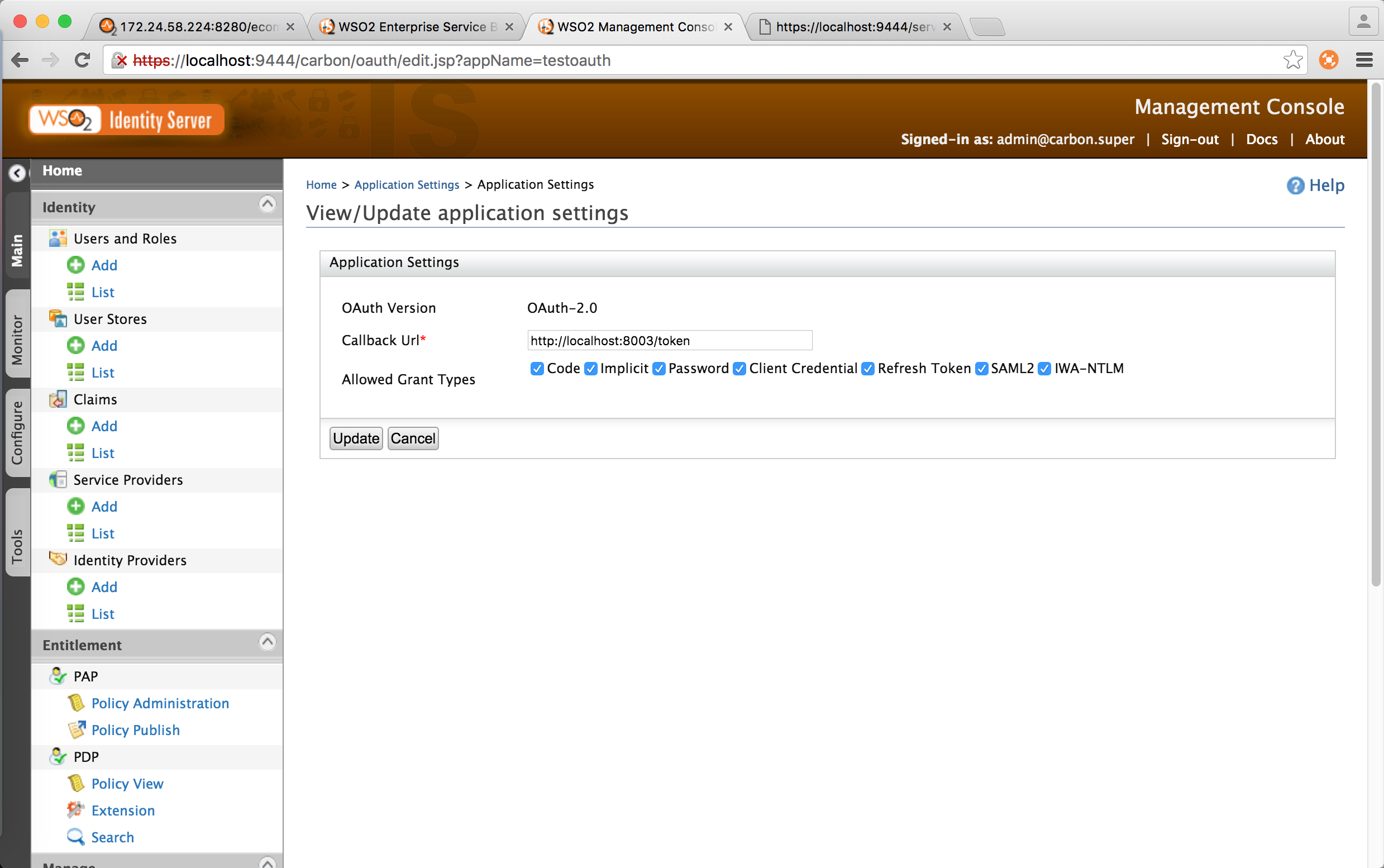
向IS发起请求以获取access\_token(可以用Curl命令，也可以用Postman等工具): curl -v -k -X POST --user CL7hb\_C7hAmNryv7dzVm9VDT6xMa:Ikb8fAFPKPSaapdznNhBGTExJnMa -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" -d 'grant\_type=password&username=admin&password=admin' https://localhost:9444/oauth2/token

记录返回的access\_token值

## 三 代码分析

1. 配置wso2 identity server





2. 配置esb api config

对每一个resource的inSequence添加oauthService

<oauthService remoteServiceUrl="https://localhost:9444/services" username="admin" password="admin"></oauthService>

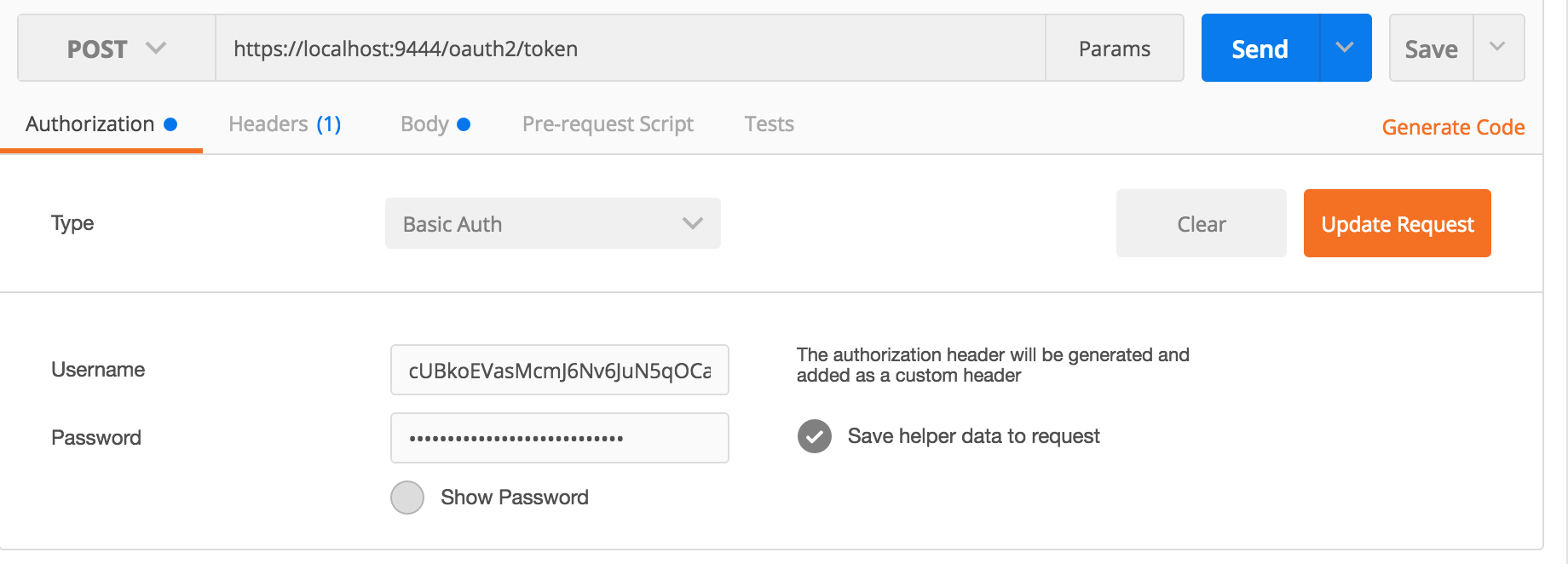


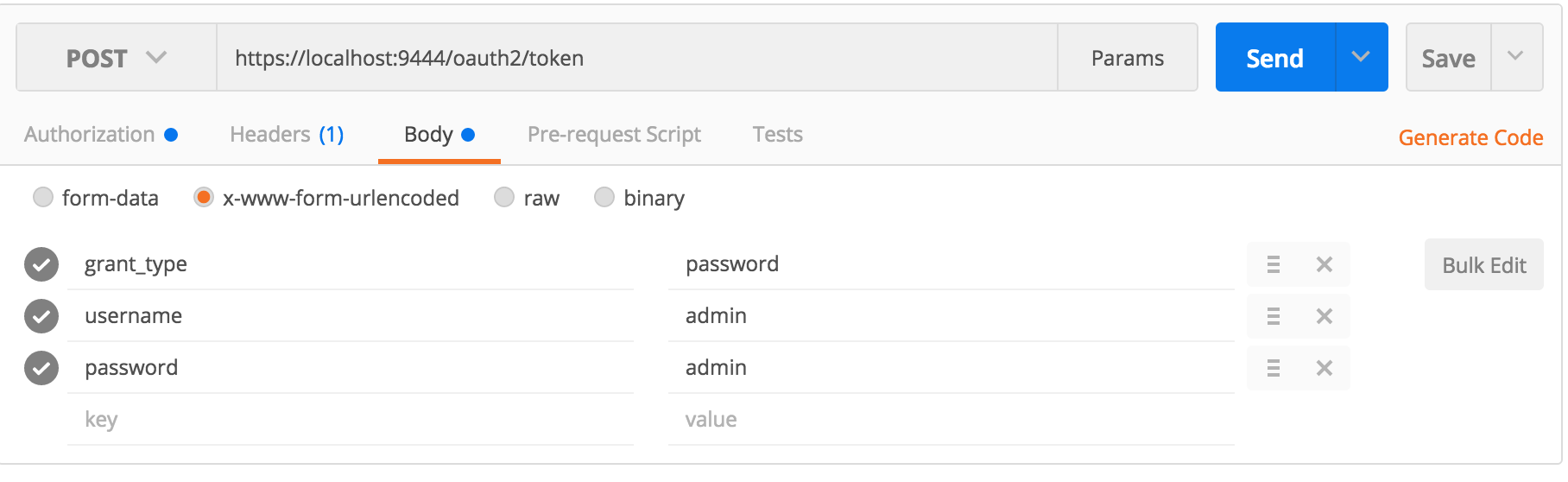
## 四 验证

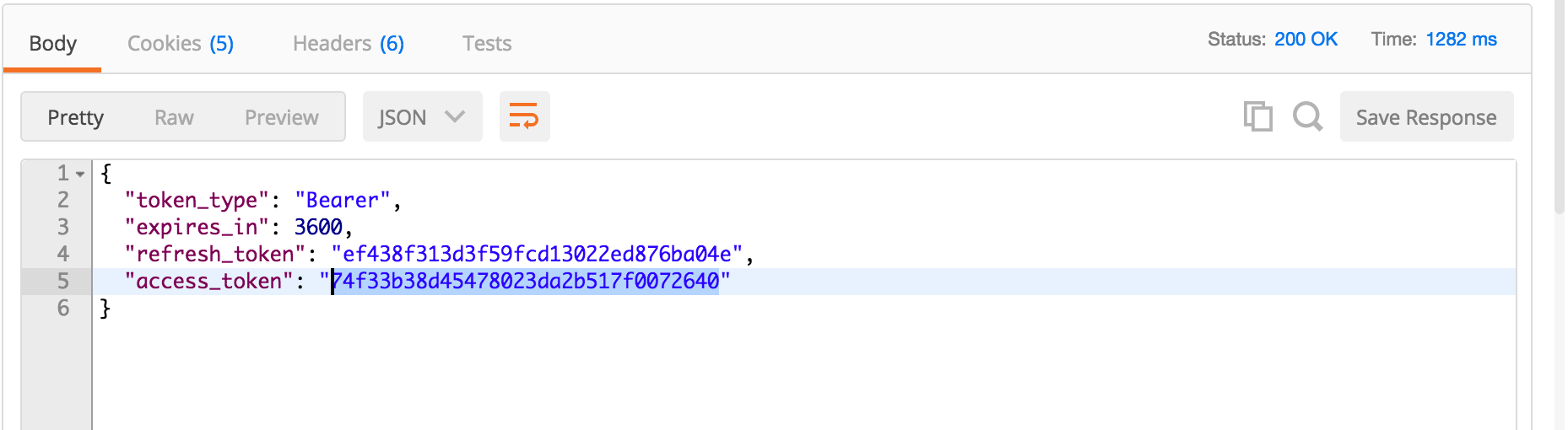
1. 启动wso2 esb server

2. 启动wso2 identity server

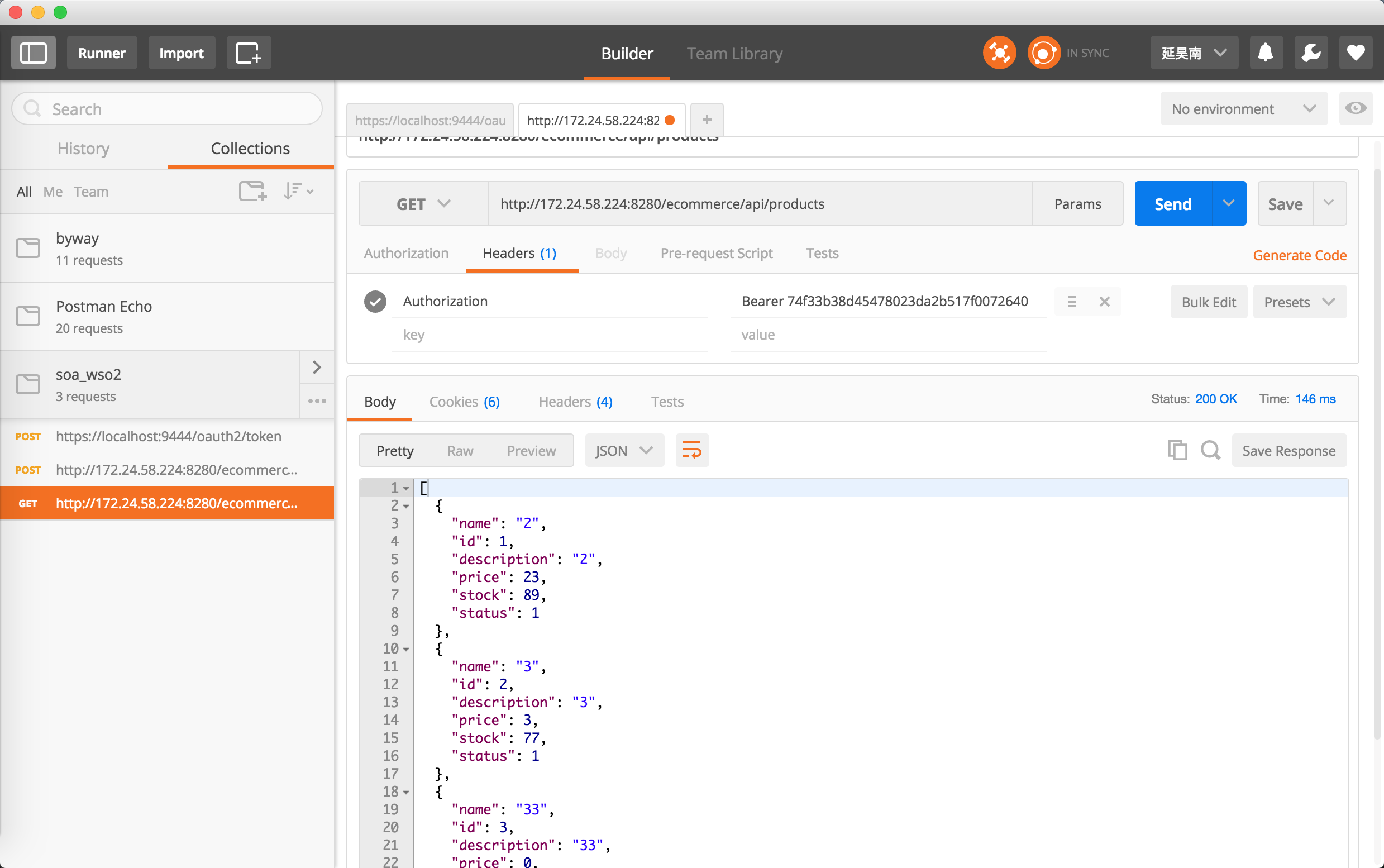
3. 获取access\_token







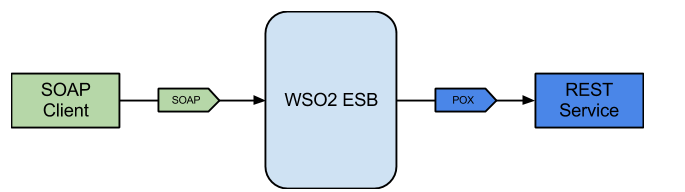
4. 调用接口 获取数据



# Part 3: soap信息和restful信息的转换

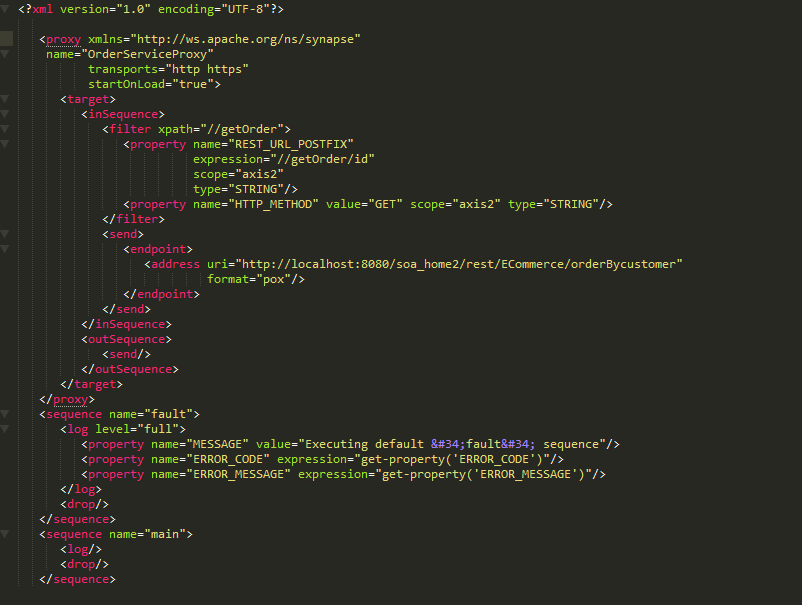
## 一 代码解析

该部分实现的功能是：将用户发送的soap message，通过proxy中的insequence的filter解析，将信息传输给我们部署在tomcat上的restful webservice，实现了soap信息和restful信息的转换，完整的实现了附加功能点。

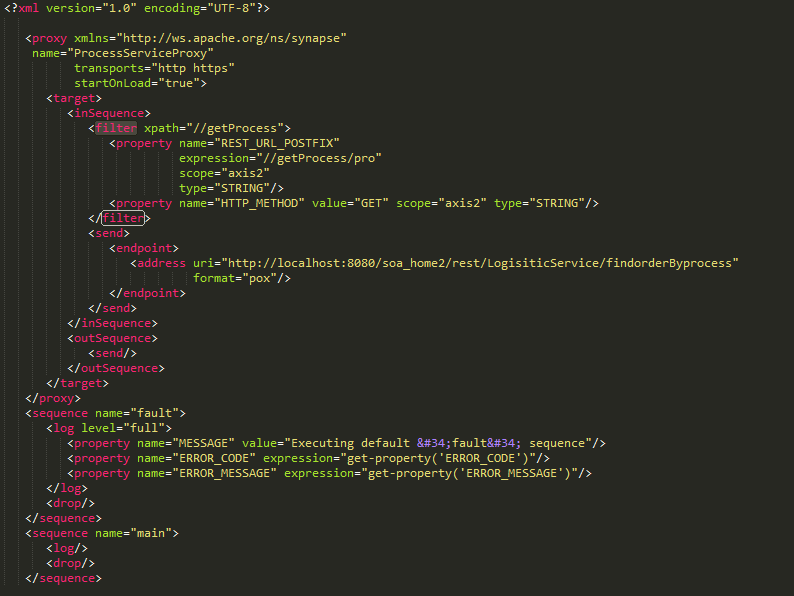


写了两个实例proxy，其他service的soap与restful转换基本相同。

以下是根据用户soap message中提供的用户id，返回用户所有订单的服务代码



以下是根据用户soap message中的状态，返回对应状态所有订单的服务。



其中，我们在insequence中的filter里设置我们要获取的信息域，



表示我们要获取soap信息中getOrder标签下面的内容。

我们之后的soap信息格式如下：

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

<soapenv:Header/>

<soapenv:Body>

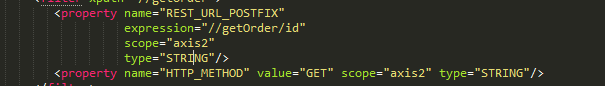
<getOrder>

<id>1</id>

</getOrder>

</soapenv:Body>

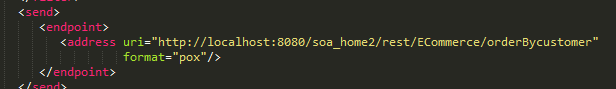
</soapenv:Envelope>



在之后的property提取中，我们没有选择使用arg标签对soap信息中的属性挨个提取，而是采取官方提供的REST\_URL\_POSTFIX方法，通过expression的解析，将id值提取，实现了soap格式的解析。之后设置url访问方式为GET

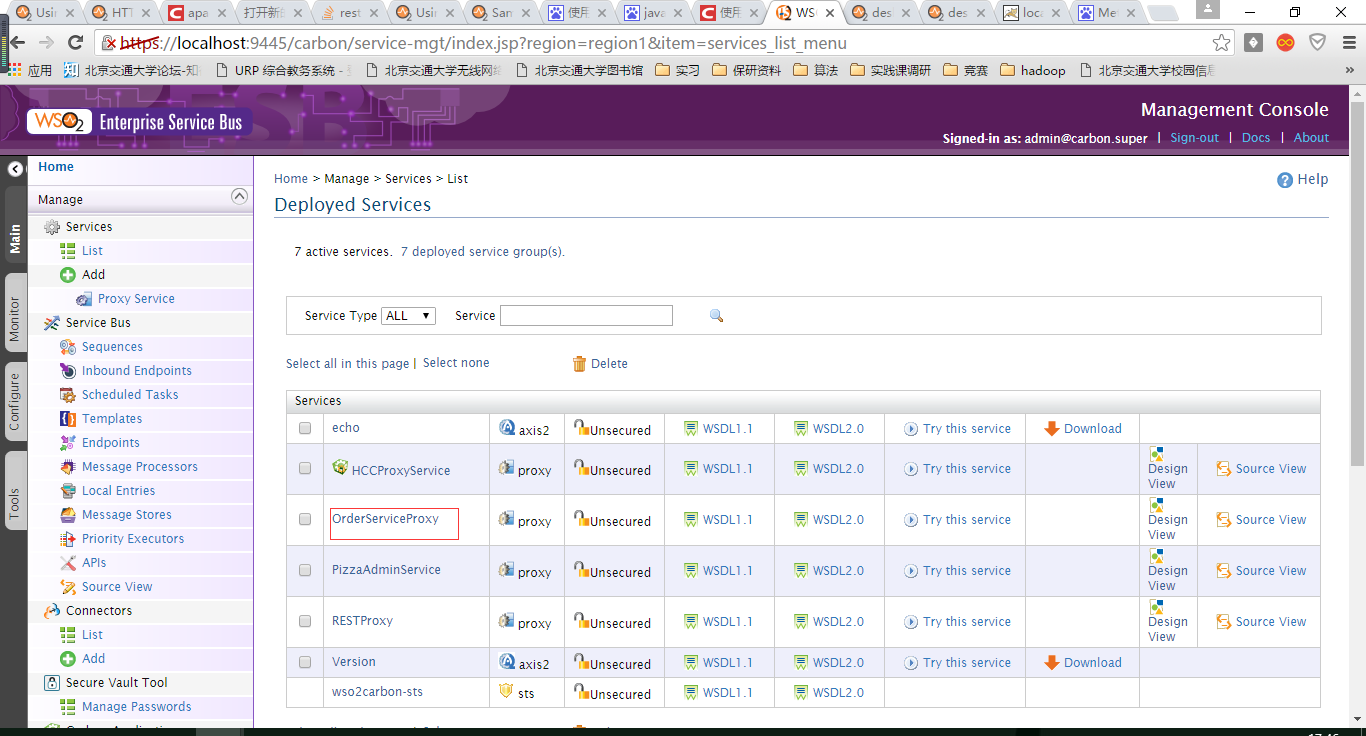


设置要访问的restful webservice的url地址：



## 二 操作流程

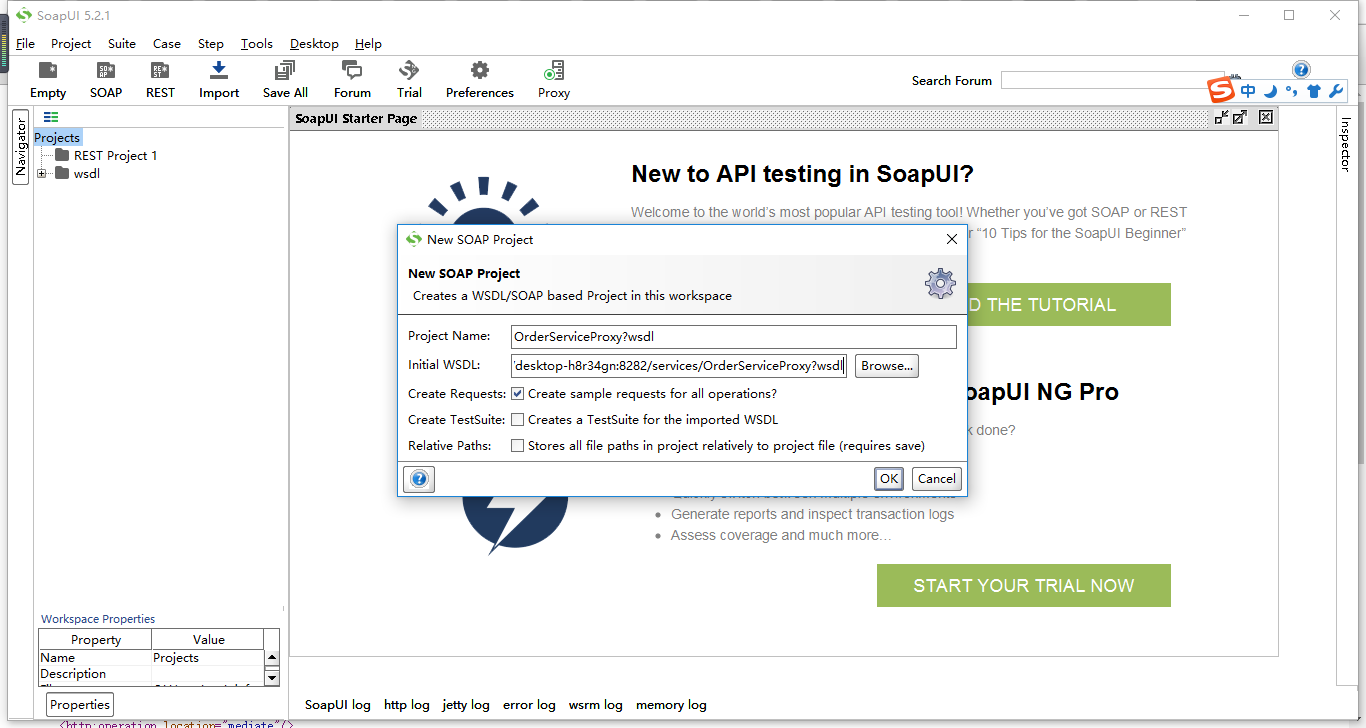
1 登录WSO2，选择我们的proxy service



2 点击其WSDL查看，将wsdl地址复制



3 打开soapUI，新建一个soap项目，将wsdl地址填入



4 新建一个soap请求，将之前的soap信息按照格式填入

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

<soapenv:Header/>

<soapenv:Body>

<getOrder>

<id>1</id>

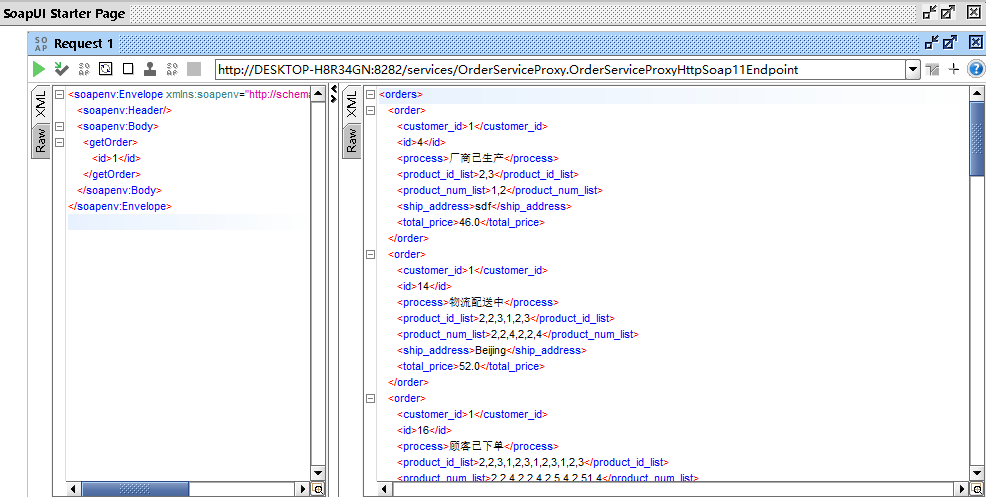
</getOrder>

</soapenv:Body>

</soapenv:Envelope>

构造一个soap信息，发送到esb

5 发送，得到返回结果:

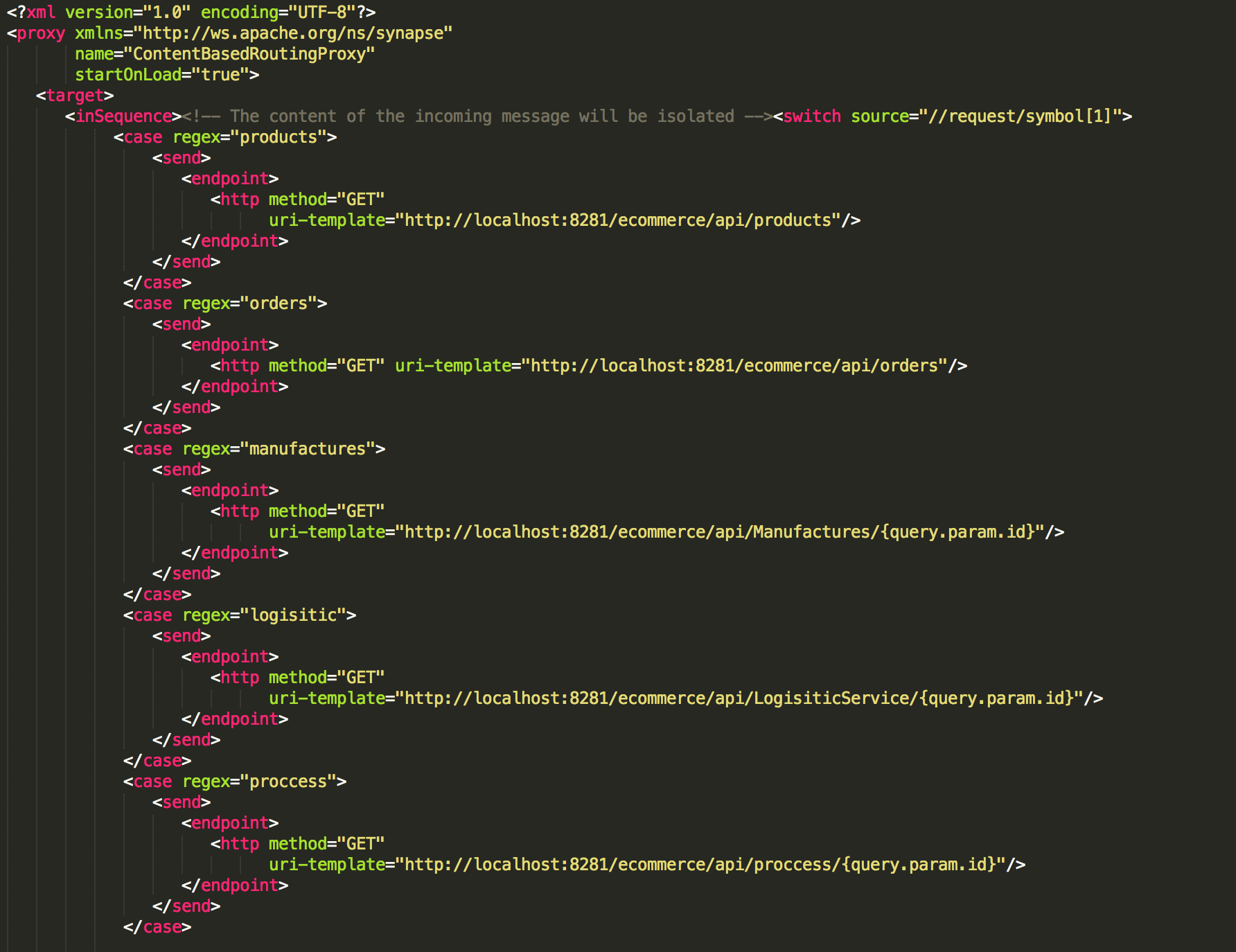


# Part 4:Content Based Routing(CBR)

## 一 代码解析

该部分实现的功能是：用户直接通过ContentBasedRoutingProxy发送想要进行的服务， ContentBasedRoutingProxy根据用户传递的信息通过switch case 进行解析，调用已部署在ESB上的相应服务，完整的实现了附加功能点。

以下是具体实现代码（以查询所有产品为例）：



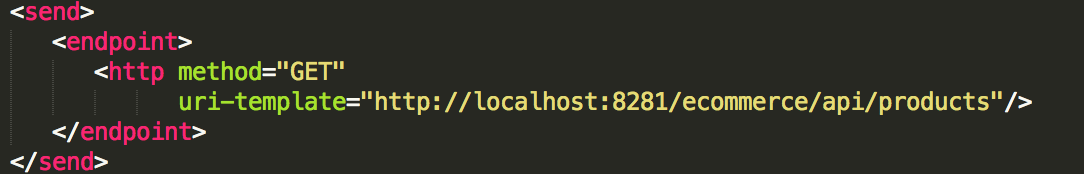
其中，我们在insequence中的switch里设置我们要获取的信息域，表示获取request标签下的第一个symbol标签



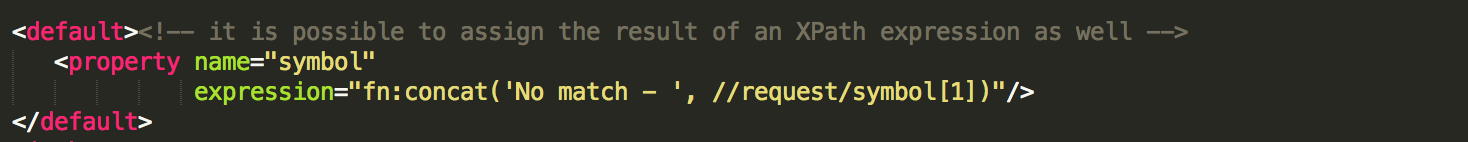
然后通过case标签进行判断，如下图是判断内容是否是products



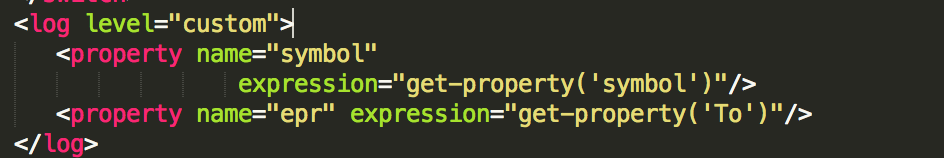
在case内输入符合条件后进行的操作，如下图是把消息发送到之前部署到esb的查询所有产品的服务上



default标签是switch中不满足case时进行的操作，下图是把不满足case的symbol内容储存在property中。



通过log 可以定制想要的日志输出，下图将symbol标签的内容以及消息转发的地址进行了输出。

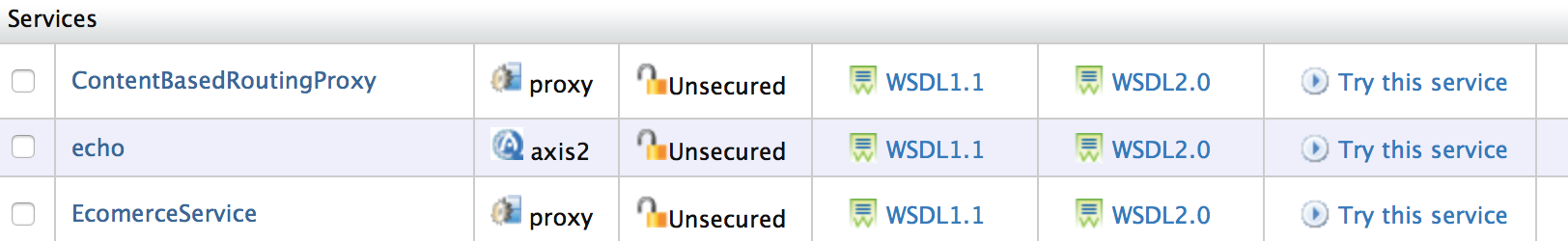


## 二 操作流程

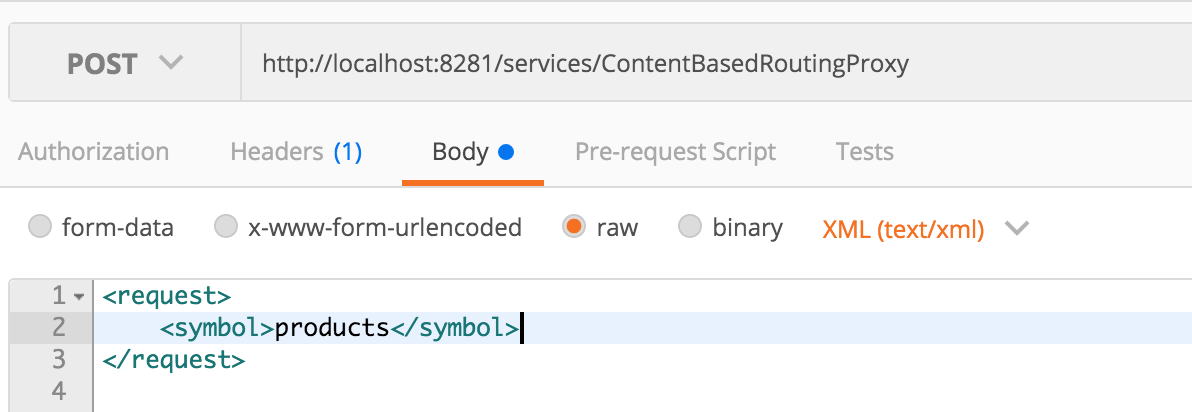
1 因为是用写XML文件的形式写的ContentBasedRoutingProxy，因此需要先将服务部署到esb中。点击Service Bus 下的Source View标签将文件内容复制进行即可实现注册（或者可直接将文件拷贝到synapse的代理服务器目录下）。



2 点击Services 下的List 即可看到代理已注册。

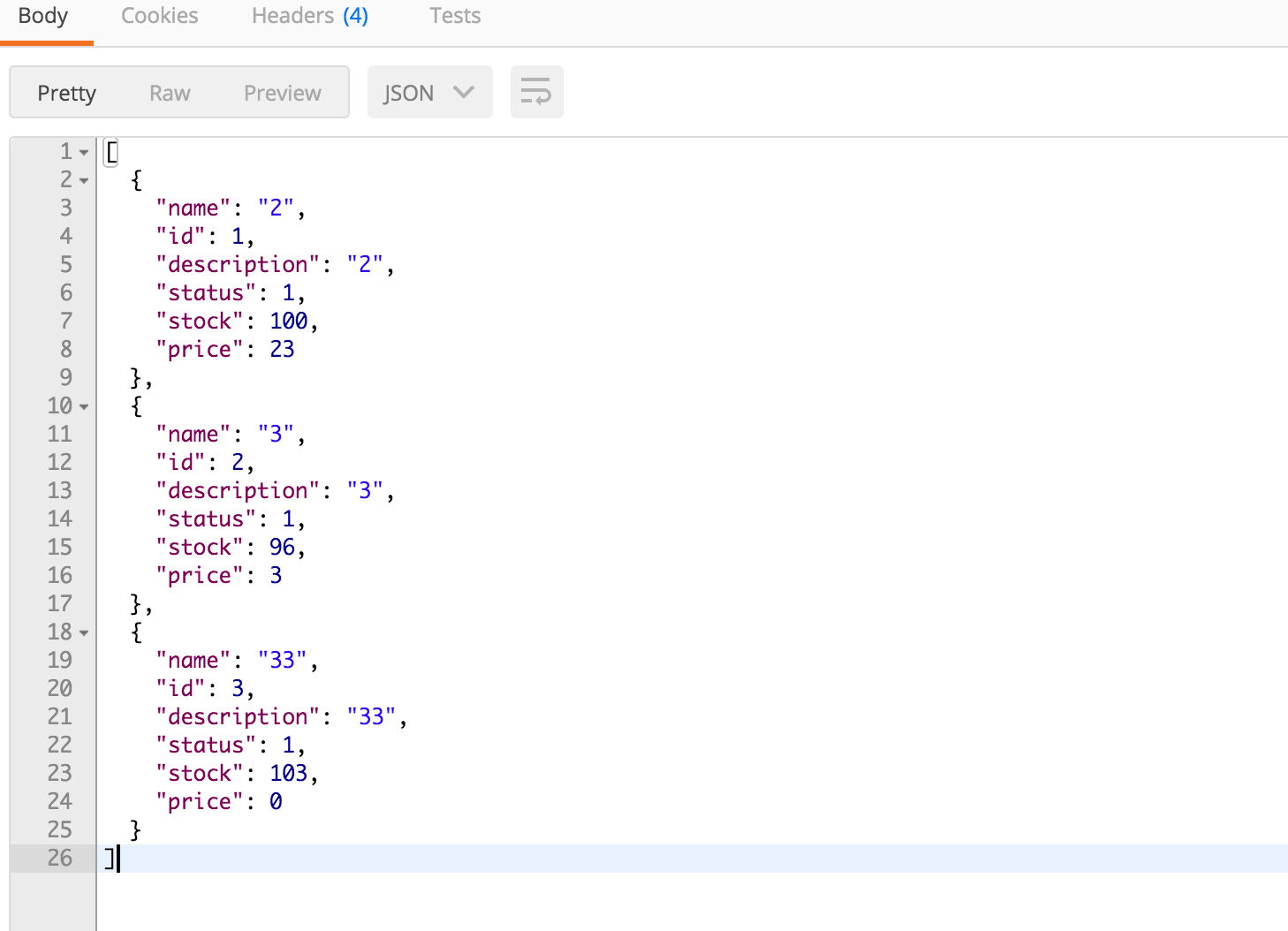


3 打开postman，新建一个Post请求，网址为localhost+端口＋services/ContentBasedRoutingProxy。

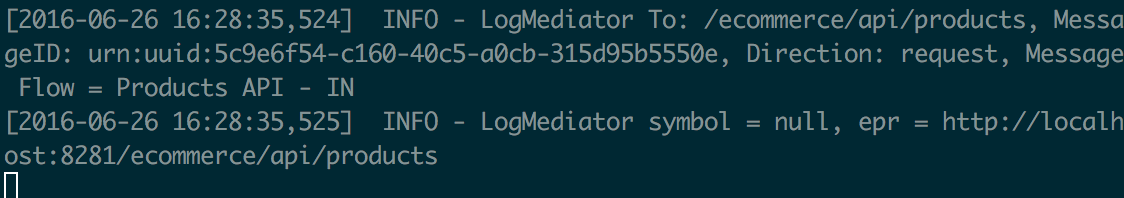


同时在body内写入信息。如需要查询所有产品，在symbol标签中打入products即可。

4 点击发送，得到返回信息。



5 可以在控制台中看到，CBR 代理将消息转发回了之前部署的服务



# Part 5: 界面截图

## 商品列表



****

## 订单状态



## 订单列表



## 物流状态



# Part 6: 作业总结

本次soa第三次作业实现了用ESB连接整合四个不同子系统的服务，并且为服务添加了安全验证、转换、路由等功能。

## 本次soa作业实现的缺点

1. 依赖于Weblogic

2. 对大数据包进行XML操作比较耗CPU

3. 将大的Object转换为XML是个繁重的操作

## ESB优点

### 1. 不需要写大量的代码

Esb产品采用了可配置的方式，用户无需写大量的代码，通过esb产品就可以很轻松地实现数据的流转整合。传统写代码的方式，在控制数据整合时，需要写大量的jdbc或者其他相关的代码，工作量巨大，而且容易出错；而采用esb产品的话，所有工作都是通过配置方式实现的，不需写代码或者只需少量的代码即可实现相关功能，大大提高了工作效率。

### 2. 屏蔽了底层实现

数据传输和整合不仅仅是写jdbc代码这么简单，对于不是直接操作数据库的情况，如采用webservice接口或者jms、socket等方式的情况，用户还需掌握这些底层技术，增加了技术复杂度，带来了额外的学习成本；而通过esb产品，用户只需不同的组件跟流程通过配置就可实现相关功能，无需关心底层的技术实现。

### 3. 准确性高

写代码的方式需要在代码中指定操作哪些数据源，哪些用户，哪些表，取哪些字段，只要发生一个小的错误就可能导致数据传输的失败或者数据的不准确；而esb产品采用可配置的方式，所有跟数据操作相关的元素都由esb产品来获取，用户只需简单选择即可，避免了错误的发生，可以达到百分之百的准确。

### 4. 丰富的协议

Esb产品中包括各种常用或者不常用的传输协议，用户可以根据实际情况选择使用，可以应付各种不同的项目情况。

### 5. 丰富的行业积累

Esb产品作为一个产品化的产物，不仅仅是解决一些基础的问题，产品自身还会根据在项目中的应用不断完善和发展。Esb产品会把各种行业解决方案中用到的内容逐渐丰富到产品中，随着项目的增多，行业的增多，esb产品的功能也越来越丰富，可以应对不同行业的不同需求。

### 6. 可维护性高

写代码的方式，如果需求等方面出现变动，就需要修改相应代码，少则可能几处，多则可能十几处几十处，给维护工作带来了很大的不便，中间的任何疏漏都可能导致错误的发生；esb产品采用组件实现，即使需求发生变化，只需修改几个组件即可，既减少了维护的工作量又增加了可靠性。

### 7. 安全性高

写代码的方式很少考虑安全性方面的要求，代码只满足了基本的需求，但是安全性不高；esb产品是成熟的全面的产品，不仅实现功能，还包括安全性方面的内容，比如在数据传输中采用安全性的协议，对传输的数据进行备份，对传输过程进行日志记录，即使在复杂的网络情况下也能保证数据安全传输，即使出现问题也能通过日志和备份进行分析查找并恢复，将损失降到最低。

### 8. 性能高

Esb产品充分考虑性能问题，在性能方面做了大量优化，可以极大提高传输效率，这是写代码所无法比拟的。