mule教程

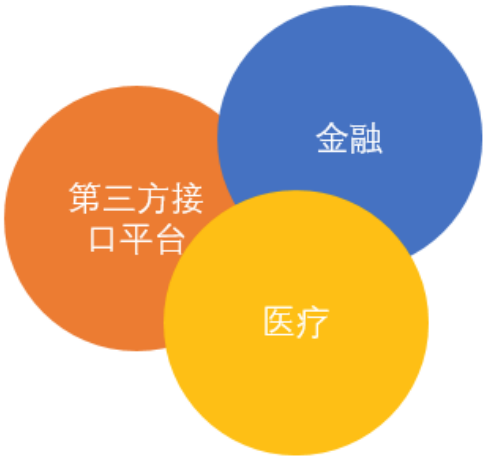
1. Mule的介绍

1.1 什么是Mule

它是一个以Java为核心的轻量级的消息框架和整合平台，基于EIP（Enterprise Integration Patterns,由Hohpe和Woolf编写的一本书）而实现的。

1.2 ESB提供的基本功能

1.3 Mule 的使用领域



1.4 Mule 掌握要点

必须熟悉MEL(Mule Expression Language)语言

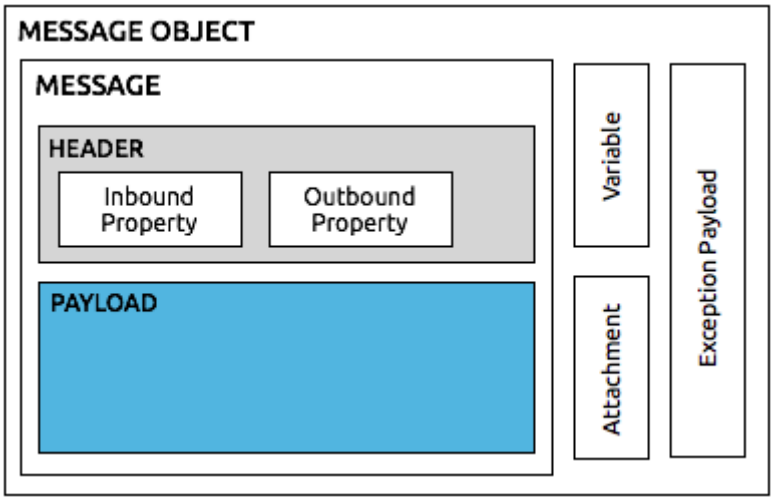
了解MuleMessage结构和Payload对象

对常用的connector、scoper、component、transformer、filter、flowControl、errorHanding要熟悉

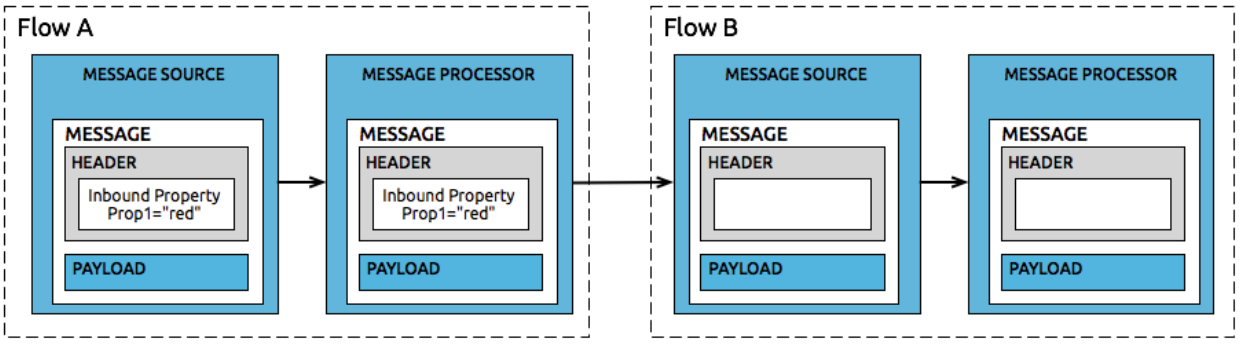
解APIKit Router和 APIKit Console，熟悉RAML

Message 消息结构

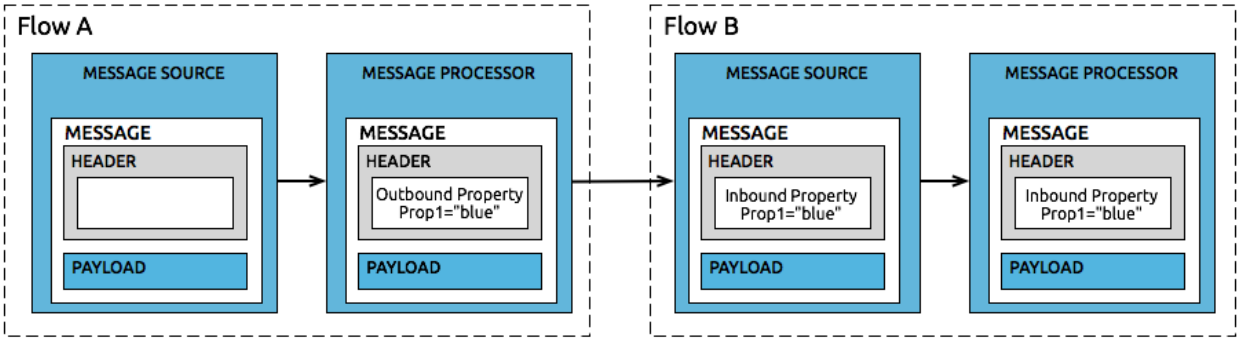
满足我们所有消息的传递



Inbound properties 入站参数



Outbound properties 出站参数



Payload 消息负载

用户请求数据

Variables 变量

变量类型	生命周期	调用方式
flowVars	同一个Flow	#[flowVars.定义变量]
sessionVars	同一个程序下的所有Flow	#[sessionVars.定义变量]

1.5 Mule 的四大内置对象

服务器内置对象 Server Context Object

Field	Read-only Access	Read-write Access	Field Description
`dateTime`		x	Date or time
`env`	x		Environment
`fileSeparator`	x		Character that separates components of a file path ("/" on UNIX and "" on Windows)
`host`	x		Fully-qualified domain name of a server
`ip`	x		IP address of a server
`locale`	x		Default locale (of type java.util.Locale) of the JRE (can access server.locale.language and server.locale.country)
`javaVersion`	x		JRE version
`javaVendor`	x		JRE vendor name
`nanoSeconds`	x		Measure of nanoseconds
`osName`	x		Operating system name
`osArch`	x		Operating system architecture
`osVersion`	x		Operating system version
`systemProperties`	x		Map of Java system properties
`timeZone`	x		Default TimeZone (java.util.TimeZone) of the JRE
`tmpDir`	x		Temporary directory for use by the JRE
`userName`	x		User name
`userHome`	x		User home directory
`userDir`	x		User working directory

Mule项目对象 Mule Context Object

Field	Read-only Access	Read-write Access	Field Description
`clusterId`	x		Cluster ID
`home`	x		File system path to the home directory of the Mule server installation
`nodeId`	x		Cluster node ID
`version`	x		Mule version

项目内置对象 App Context Object

Field	Read-only Access	Read-write Access	Field Description
`encoding`	x		Application default encoding
`name`	x		Application name
`registry`		x	Map representing the Mule registry
`standalone`	x		Evaluates to true if Mule is running standalone
`workdir`	x		Application work directory

消息内置对象 Message Context Object

Field	Read-only Access	Read-write Access	Field Description
`id`	x		Unique identifier of Mule message
`rootId`	x		Root ID of Mule message
`correlationId`	x		Correlation ID
`correlationSequence`	x		Correlation sequence
`correlationGroupSize`	x		Correlation group size
`replyTo`		x	Reply to
`dataType`	x		Data type of payload
`payload`		x	Mule message payload
`inboundProperties`	x		Map representing the message's immutable inbound properties
`inboundAttachments`	x		Map representing the message's inbound attachments
`outboundProperties`		x	Map representing the message's mutable outbound properties
`outboundAttachments`		x	Map representing the message's outbound attachments

1.6 MEL的语法实例

参考链接:

https://wenku.baidu.com/view/a1d1476180c758f5f61fb7360b4c2e3f572725de.html

重点入门内容

语法介绍：

表达式	描述
#[2 + 2]	该表达式取值等于 4.
#[2 + 2 == 4]	该表达式为逻辑表达式，取值为 true.
#[message]	该表达式引用了 MEL 中四个上下文对象中的 message 对象，该表达式取值为 message 对象的值。 四个上下文对象包括： (message, app, mule, and server).具体对象参数介绍请见下文。
#[message.payload]	该表达式链接了 message 对象的 payload 属性。
#[message.payload['name']]	该表达式链接了 message 对象的 payload 属性中 name 变量中的值。注意，name 是由单引号括起来的，这是由于 mule 的配置文件采用的是双引号，所以这里只能使用单引号。
#[message.payload[4]]	同上一表达式一样，只是这里是通过数字索引来引用该值。
#[message.header.get()]	该表达式链接了 message 对象的 header 对象，并调用 get 方法。

2. Mule环境的搭建

2.1 安装JDK

jdk1.8

2.2 Mule的下载

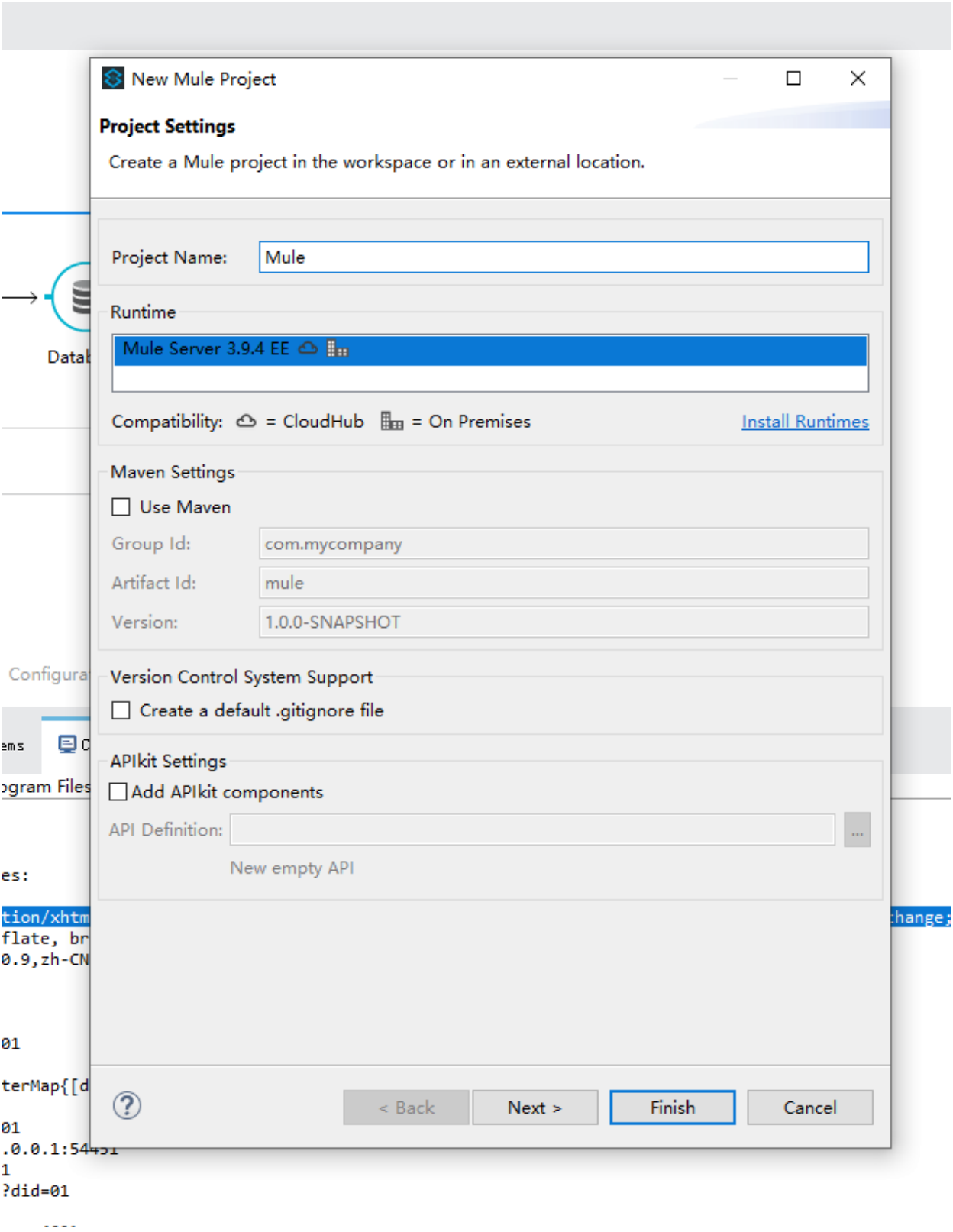
官网自己下载或者看conference

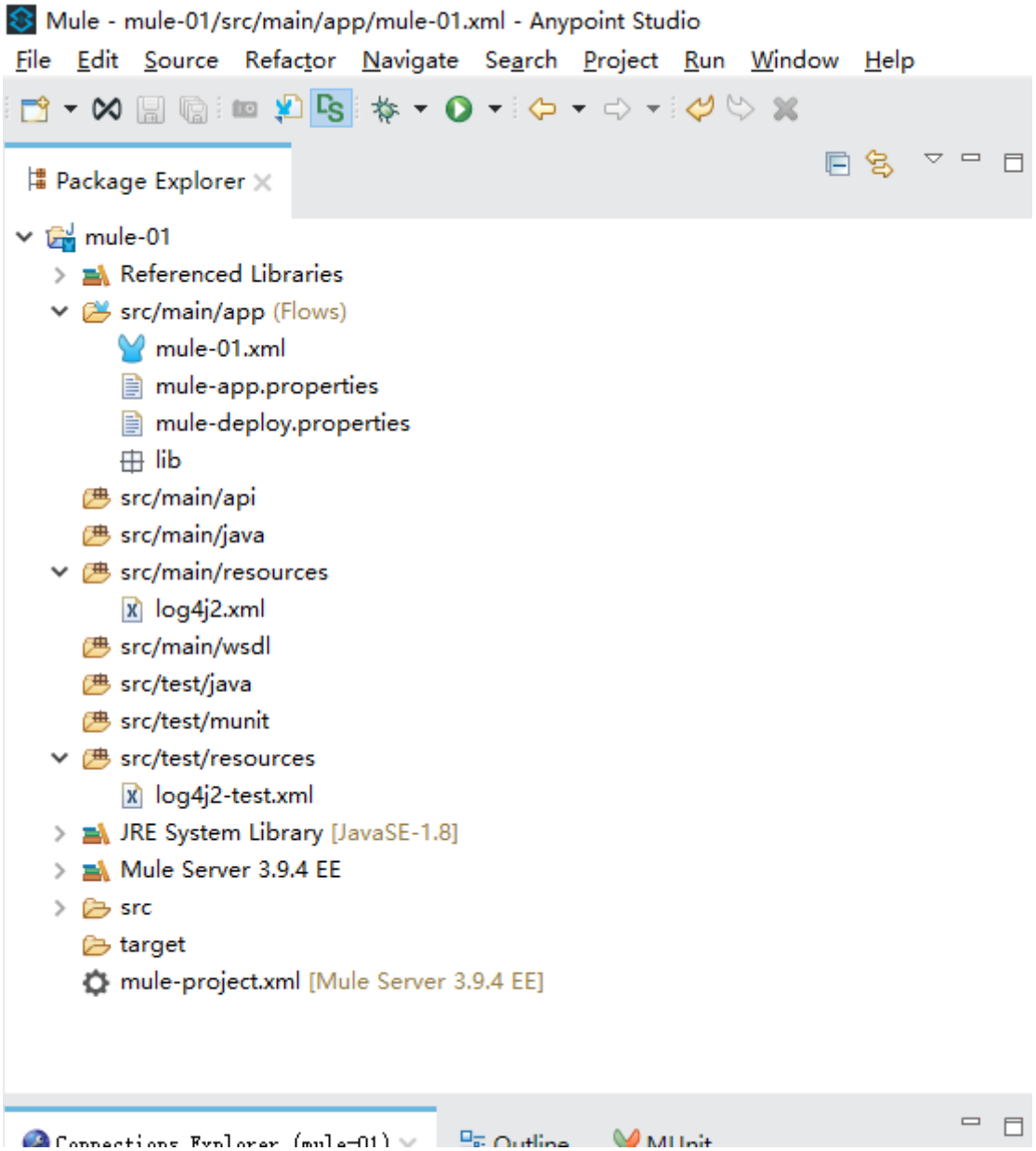
2.3 Mule 运行服务下载

anypoint studio自己下载

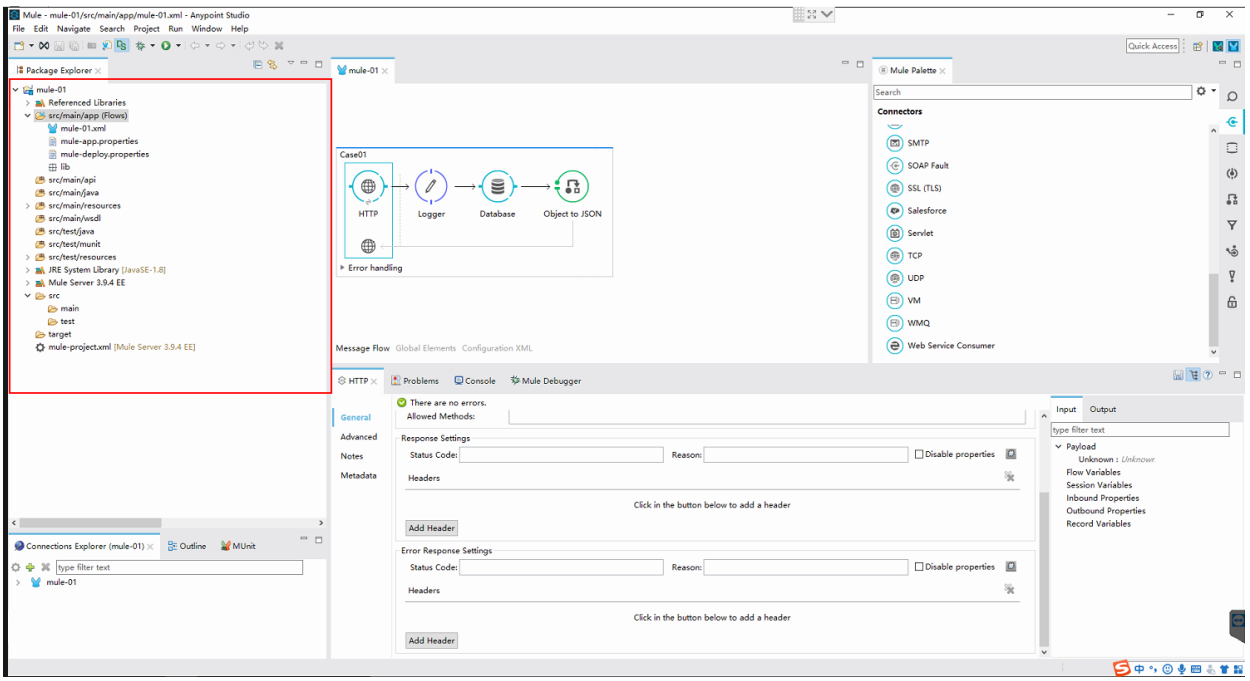
3. Mule 项目的创建及部署

3.1 创建一个简单的Mule项目


New Mule Project
Project Settings
Create a Mule project in the workspace or in an external location.
Project Name:
Mule
Runtime
Mule Server 3.9.4 EE
Compatibility: = CloudHub = On Premises
Install Runtimes
Maven Settings
Use Maven
Group Id:
com.mycompany
Artifact Id:
mule
Version:
1.0.0-SNAPSHOT
Version Control System Support
Create a default .gitignore file
APIkit Settings
Add APIkit components
API Definition:
New empty API
< Back
Next >
Finish
Cancel



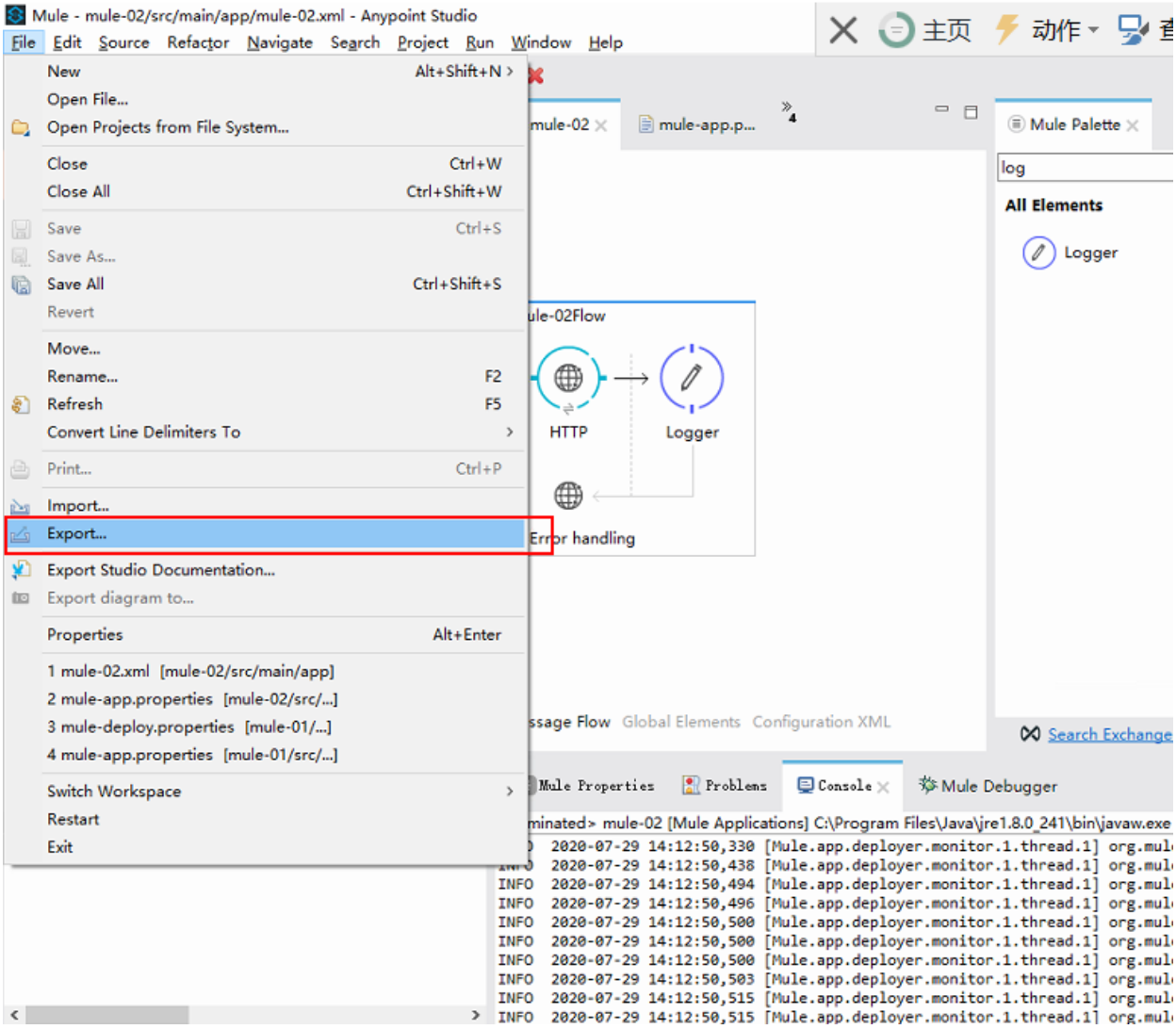
3.2 Mule 的开发结构介绍

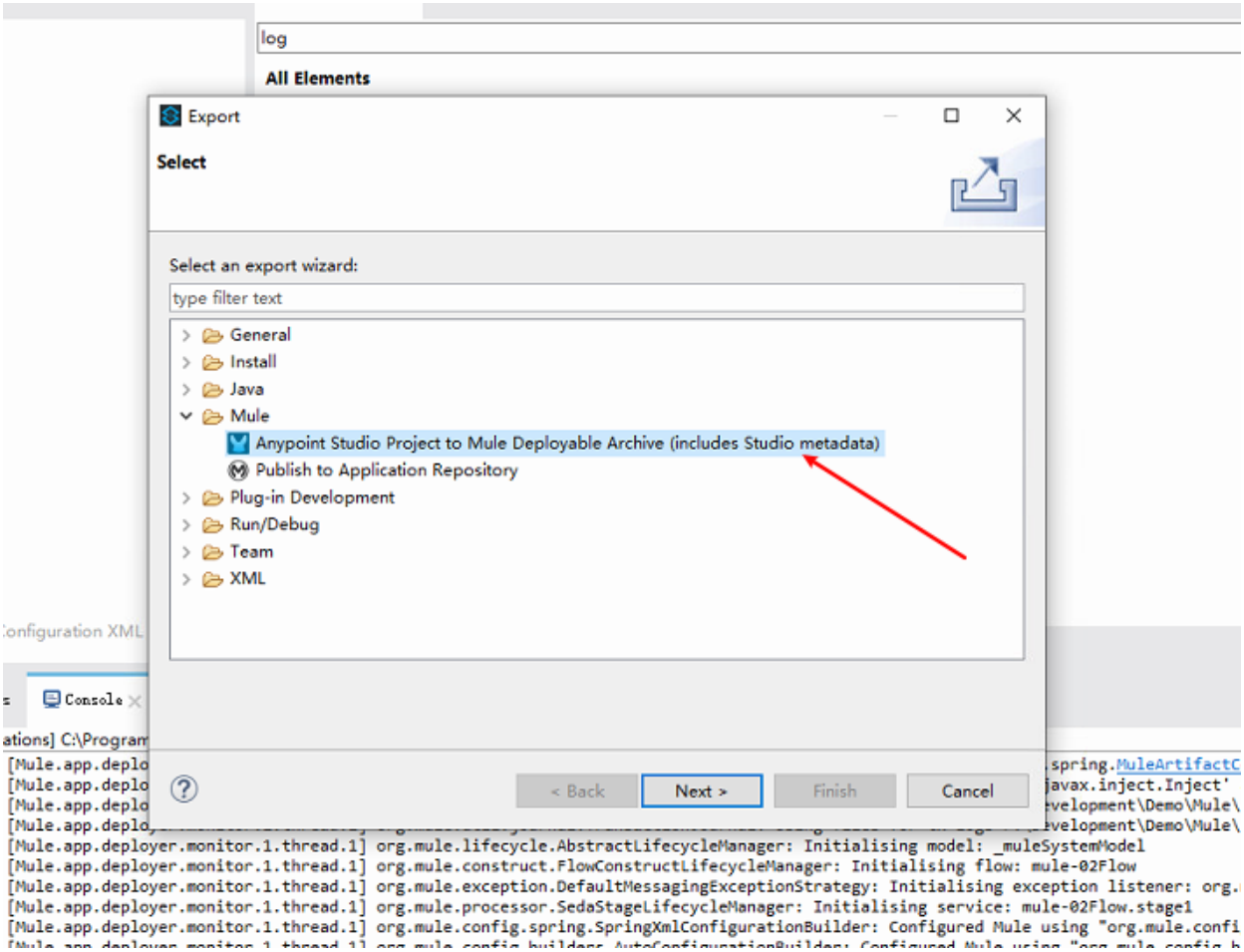


路径	说明
src/main/app	程序配置及Flow的开发
src/test/java	用户单元测试目录
src/main/java	用户自定义源码目录
src/main/resources	资源文件目录
src/test/resources	测试资源文件目录
lib	第三方Jar包存放目录

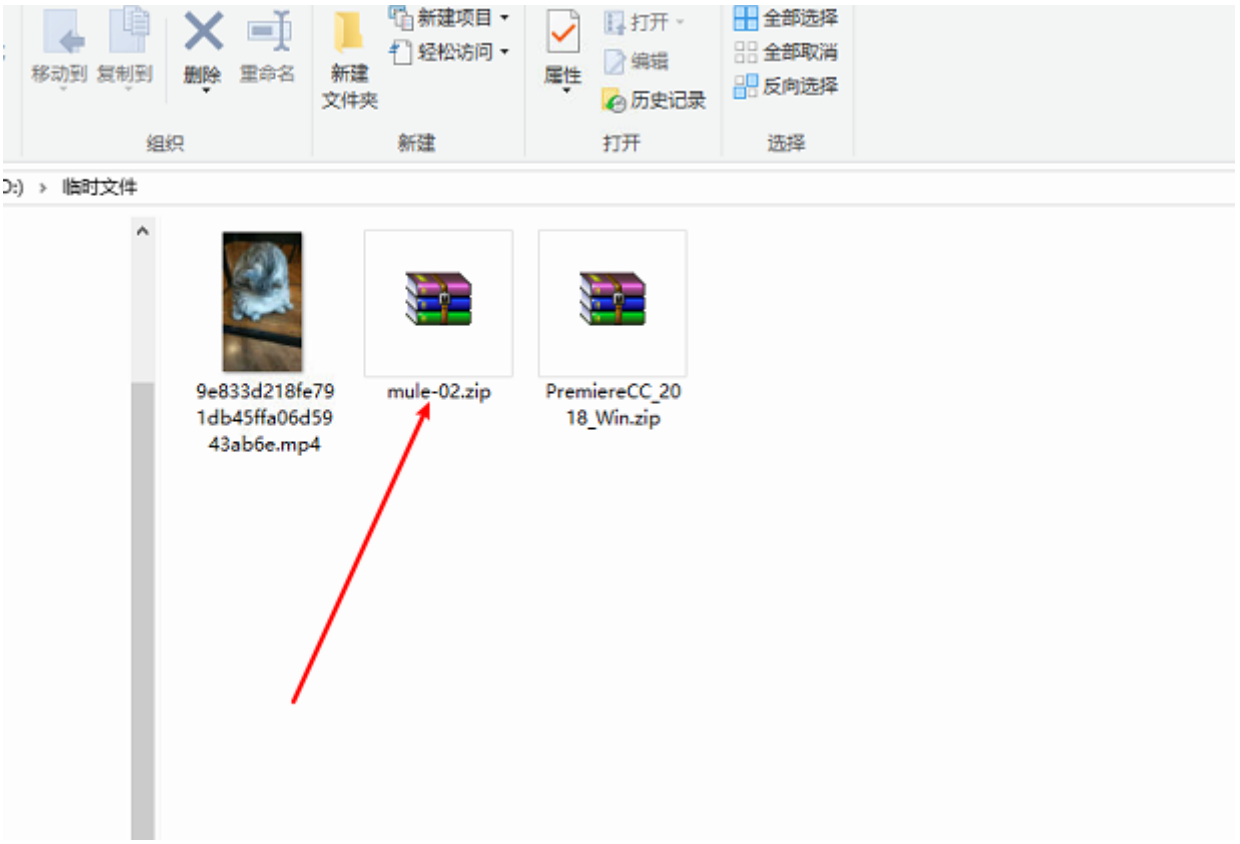
3.3 部署Mule项目到Linux 服务器

3.3.1 部署到自己搭建的Linux服务器





用工具上传下面的包到Linux服务器



解压，解压到当前目录，对部署目录给用户授权

```
# 如果使用其他用户启动mule，改变文件归属用户
```

```
chown -R 用户：所属组名
#对当前用户服务所有权限
chmod 777 -R 目录
```

目录讲解：

apps 部署Mule程序目录

conf 服务配置bin 执行文件

lib 程序Jar 包目录

logs 所有mule程序日志

```
#启动mule
./mule start
#停止muel
./mule stop
```

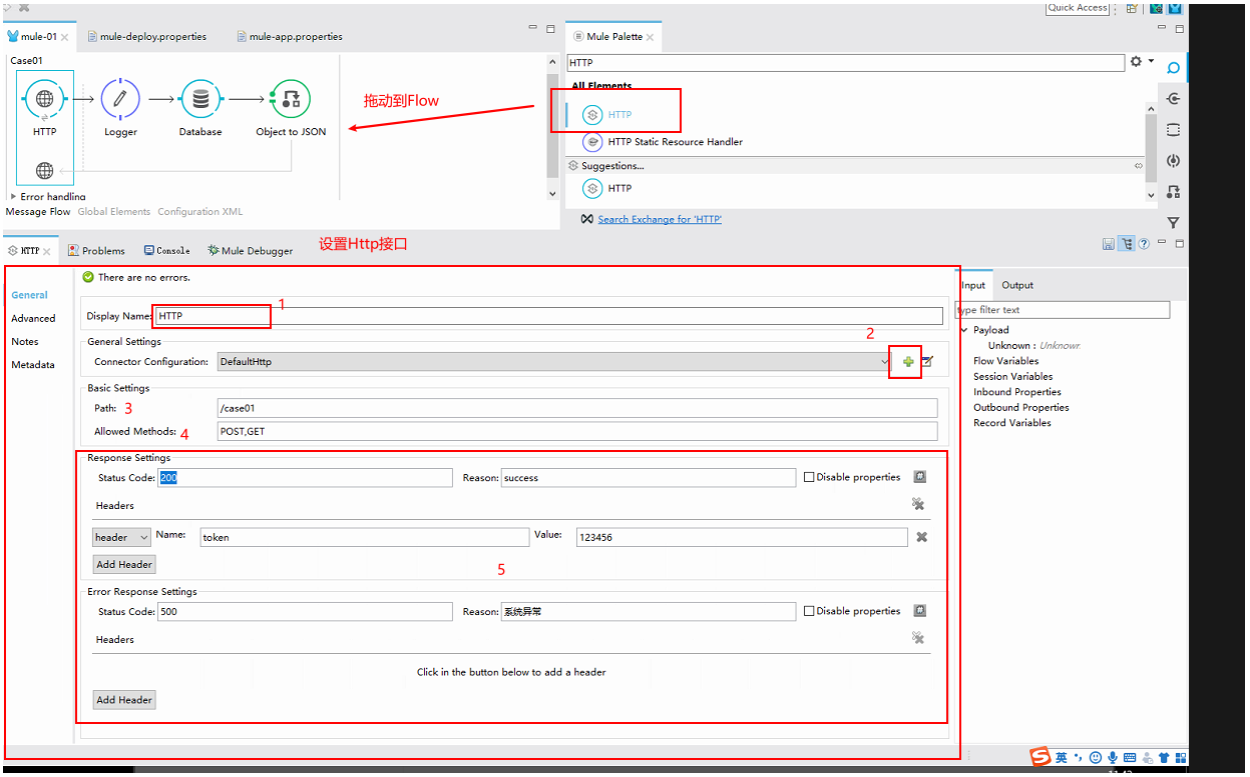
3.3.2 部署到自己公司项目服务器

George补充，好像是jenkins+docker部署

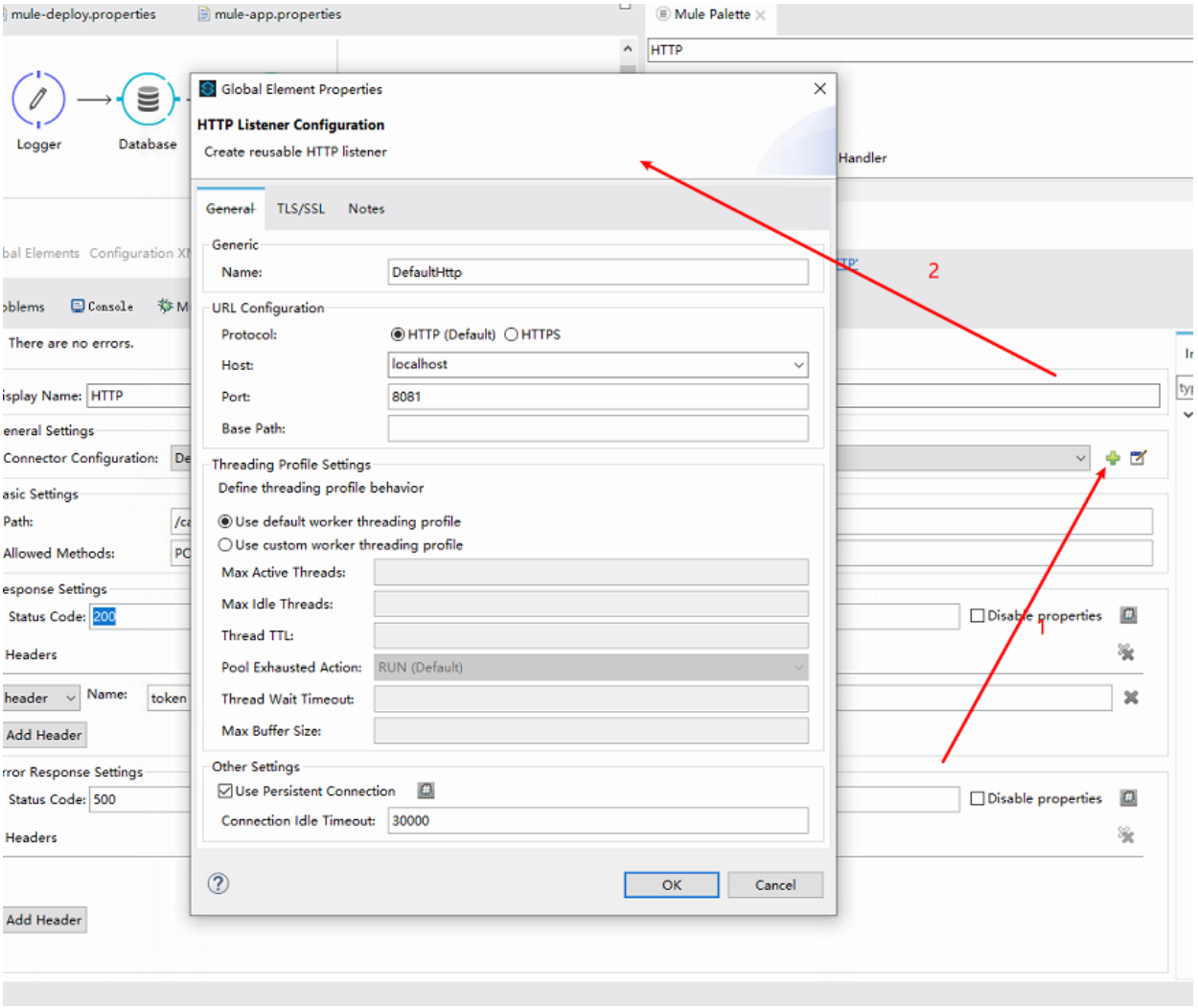
4. Mule组件

4.1 HTTP

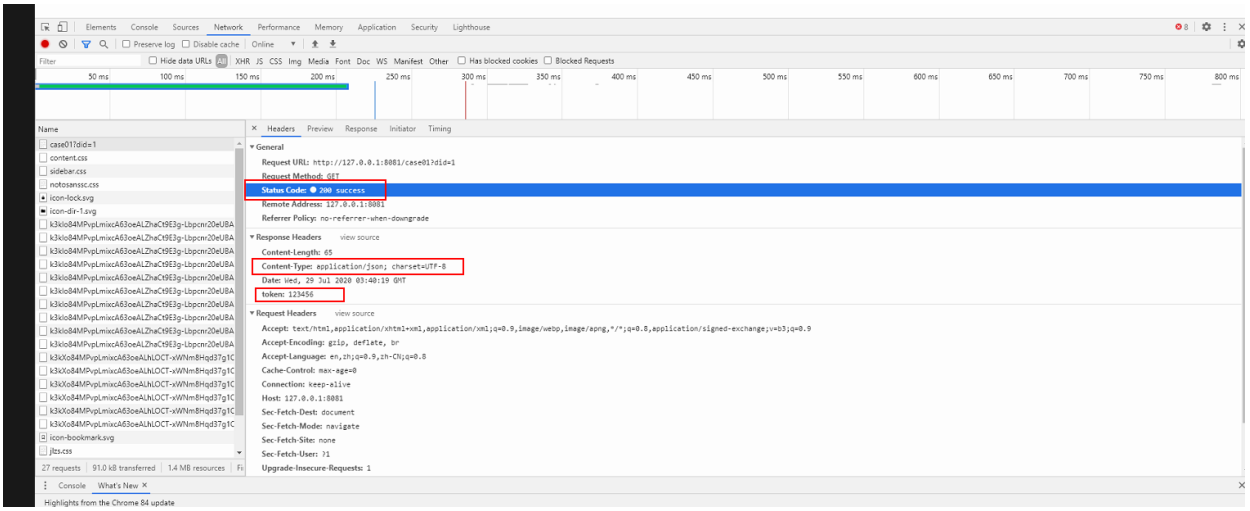
HTTP组件使用及基本参数设置



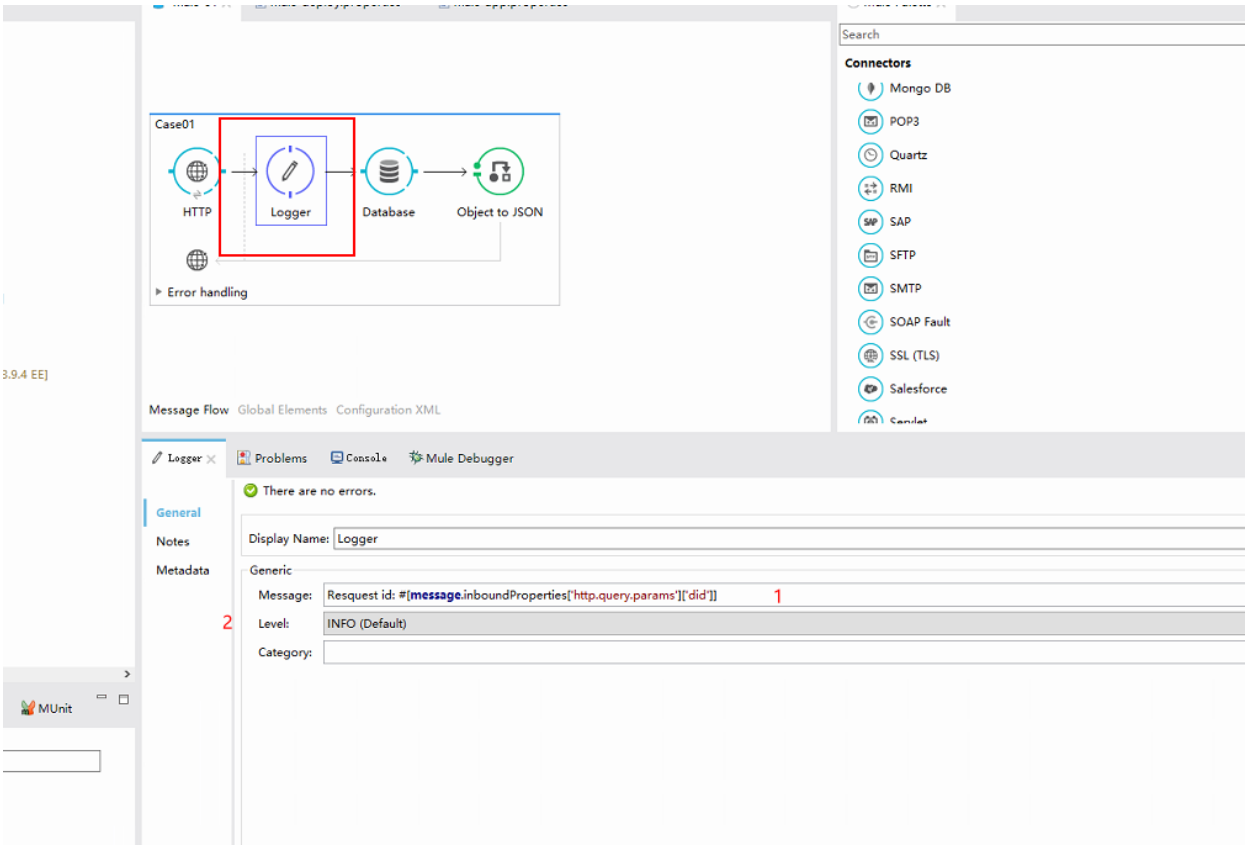
设置通用的Http设置


mule-deploy.properties
mule-app.properties
Mule Palette
HTTP
Logger
Database
Handler
Global Element Properties
HTTP Listener Configuration
Create reusable HTTP listener
General
TLS/SSL
Notes
Generic
Name:
DefaultHttp
URL Configuration
Protocol:
HTTP (Default)
HTTPS
Host:
localhost
Port:
8081
Base Path:
Threading Profile Settings
Define threading profile behavior
Use default worker threading profile
Use custom worker threading profile
Max Active Threads:
Max Idle Threads:
Thread TTL:
Pool Exhausted Action:
RUN (Default)
Thread Wait Timeout:
Max Buffer Size:
Other Settings
Use Persistent Connection
Connection Idle Timeout:
30000
OK
Cancel
There are no errors.
Status Code:
200
Headers
Add Header
Status Code:
500
Disable properties

调用结果


Request URL: http://127.0.0.1:8081/case01?did=1
Request Method: GET
Status Code: 200 success
Remote Address: 127.0.0.1:8081
Referrer Policy: no-referrer-when-downgrade
Content-Length: 65
Content-Type: application/json; charset=UTF-8
Date: Wed, 29 Jul 2020 03:40:19 GMT
token: 123456

4.2 日志组件



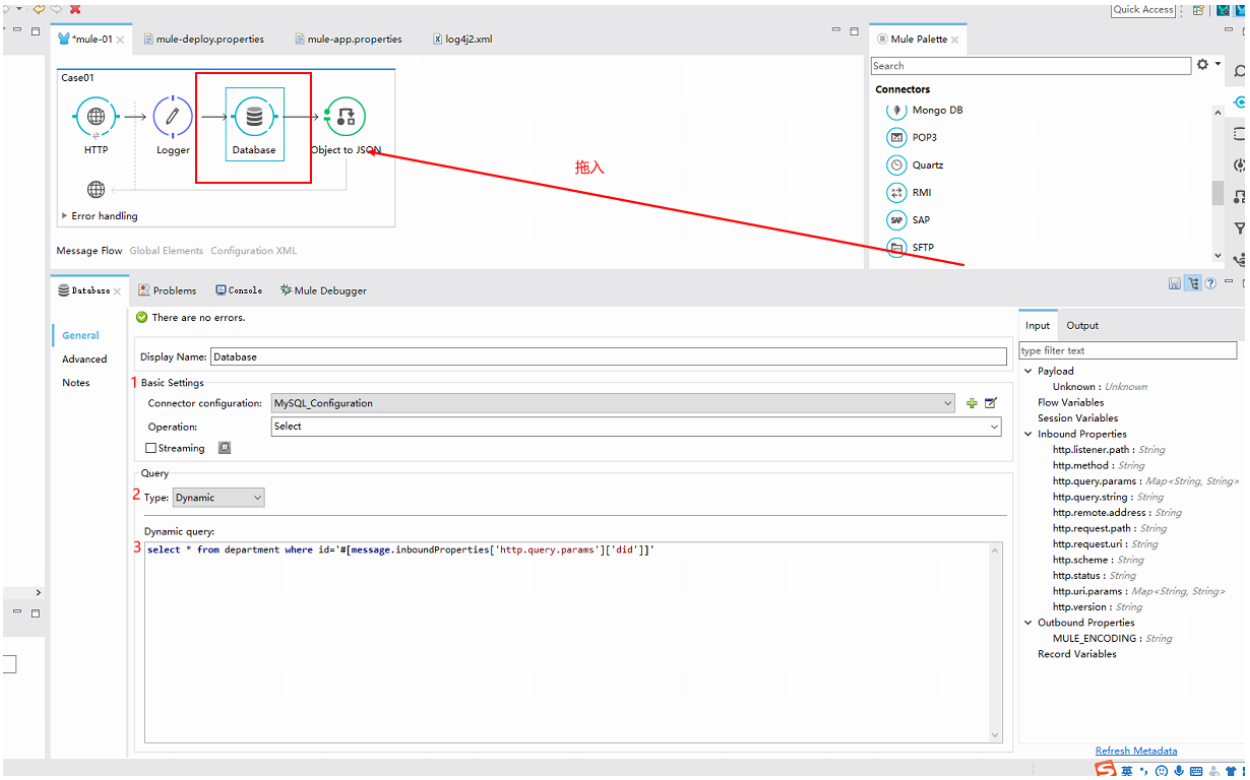
1.Message: 打印的内容，可以是从其他组件传入的值，也可以通过MEL表达式获取四大内置对象的参

数，例如上图中便是获取请求参数中的did参数值

2.Level： 设置日志打印级别，需要主要的是，当log4j2.xml 设置只显示ERROR级别时，INFO级别日志

不会打印

4.3 数据库组件



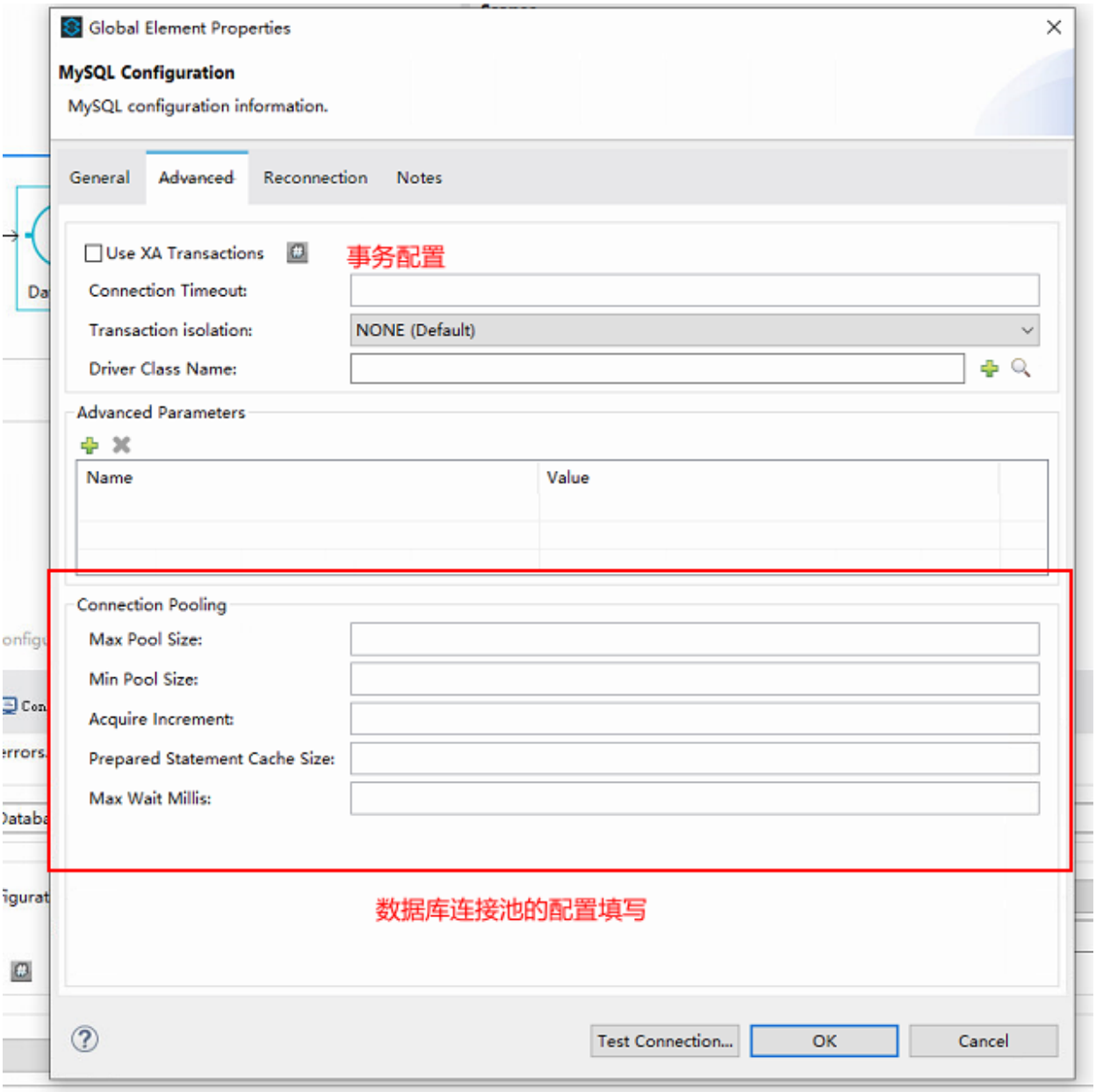
设置数据库连接存在三种方式

默认配置填写

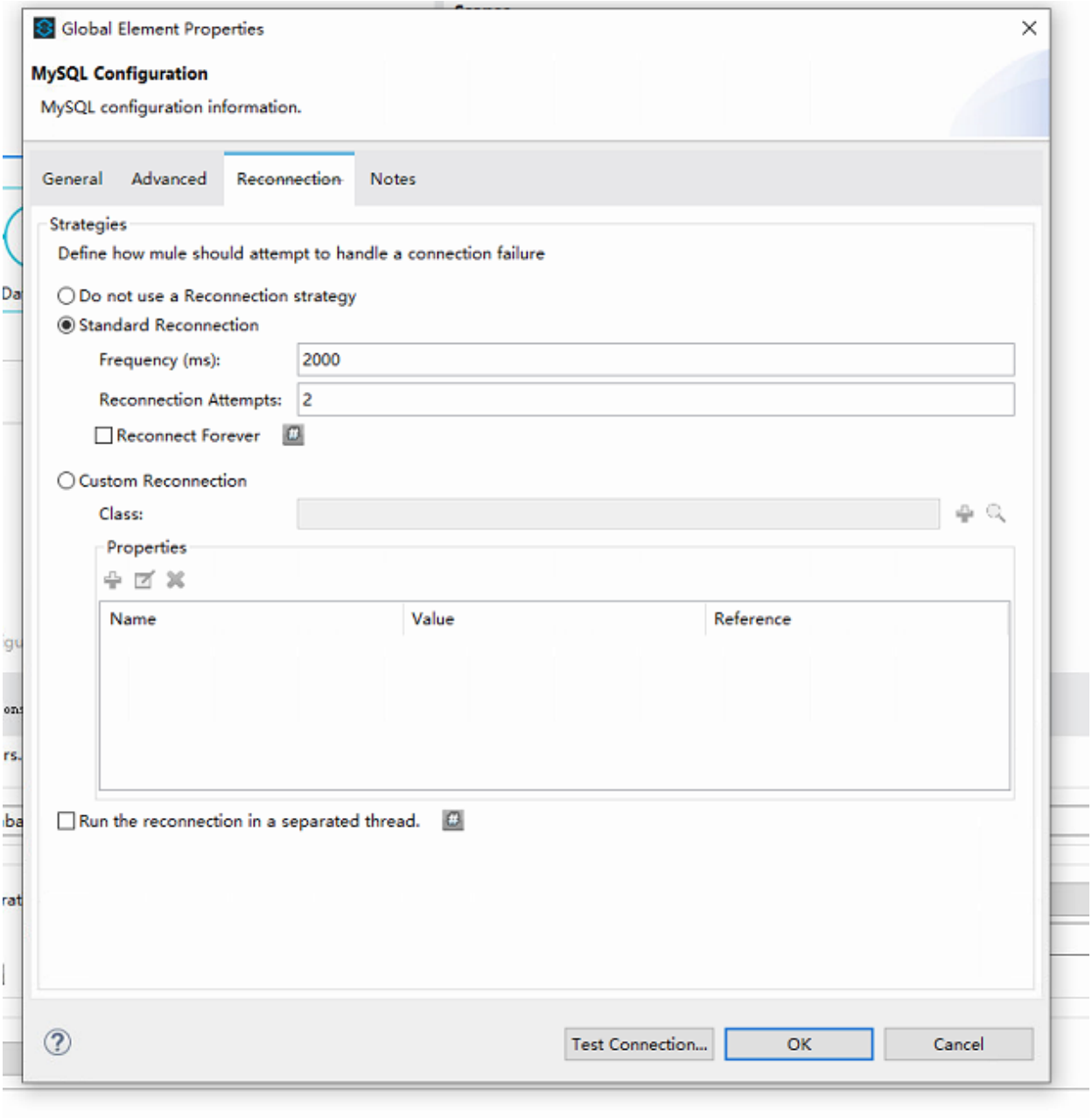
使用spring 中定义的 Datasource Bean

URL 方式

如有需要，可在Advance 高级配置中进行事务及连接池的配置

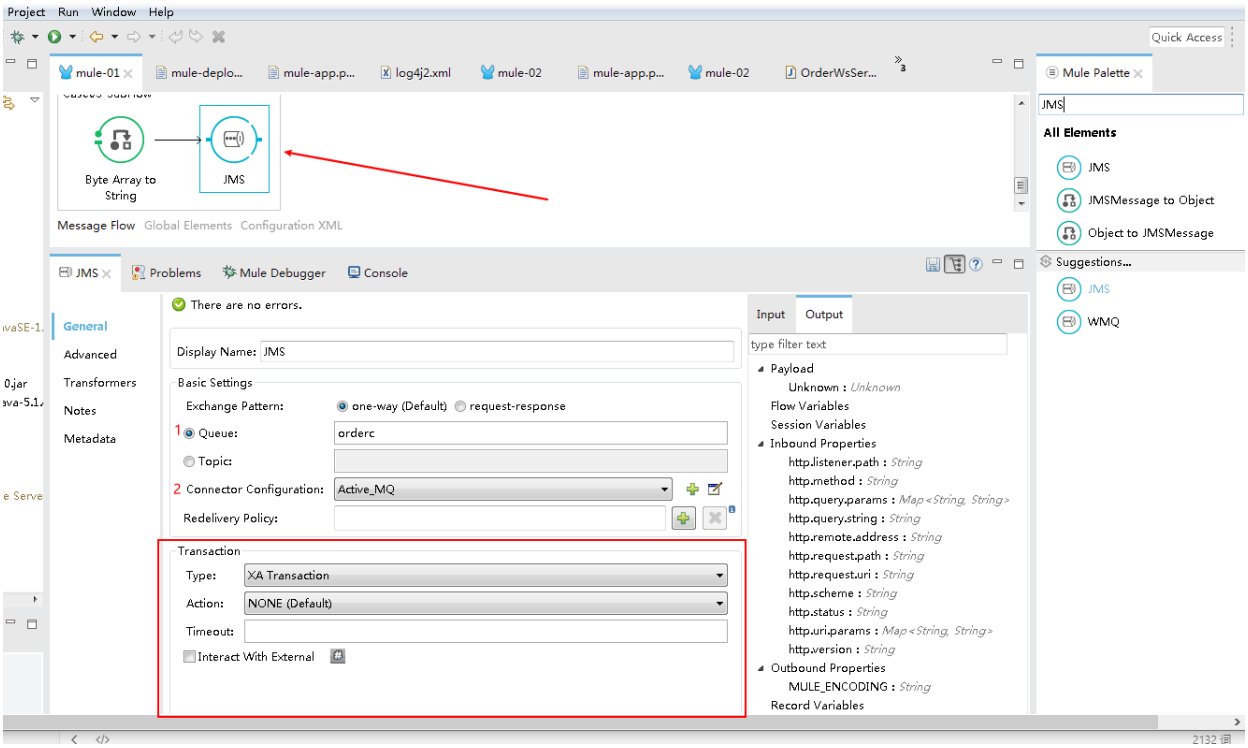

Global Element Properties
MySQL Configuration
MySQL configuration information.
General
Advanced
Reconnection
Notes
Use XA Transactions
事务配置
Connection Timeout:
Transaction isolation:
NONE (Default)
Driver Class Name:
Advanced Parameters
Name
Value
Connection Pooling
Max Pool Size:
Min Pool Size:
Acquire Increment:
Prepared Statement Cache Size:
Max Wait Millis:
数据库连接池的配置填写
Test Connection...
OK
Cancel

是否使用重连机制



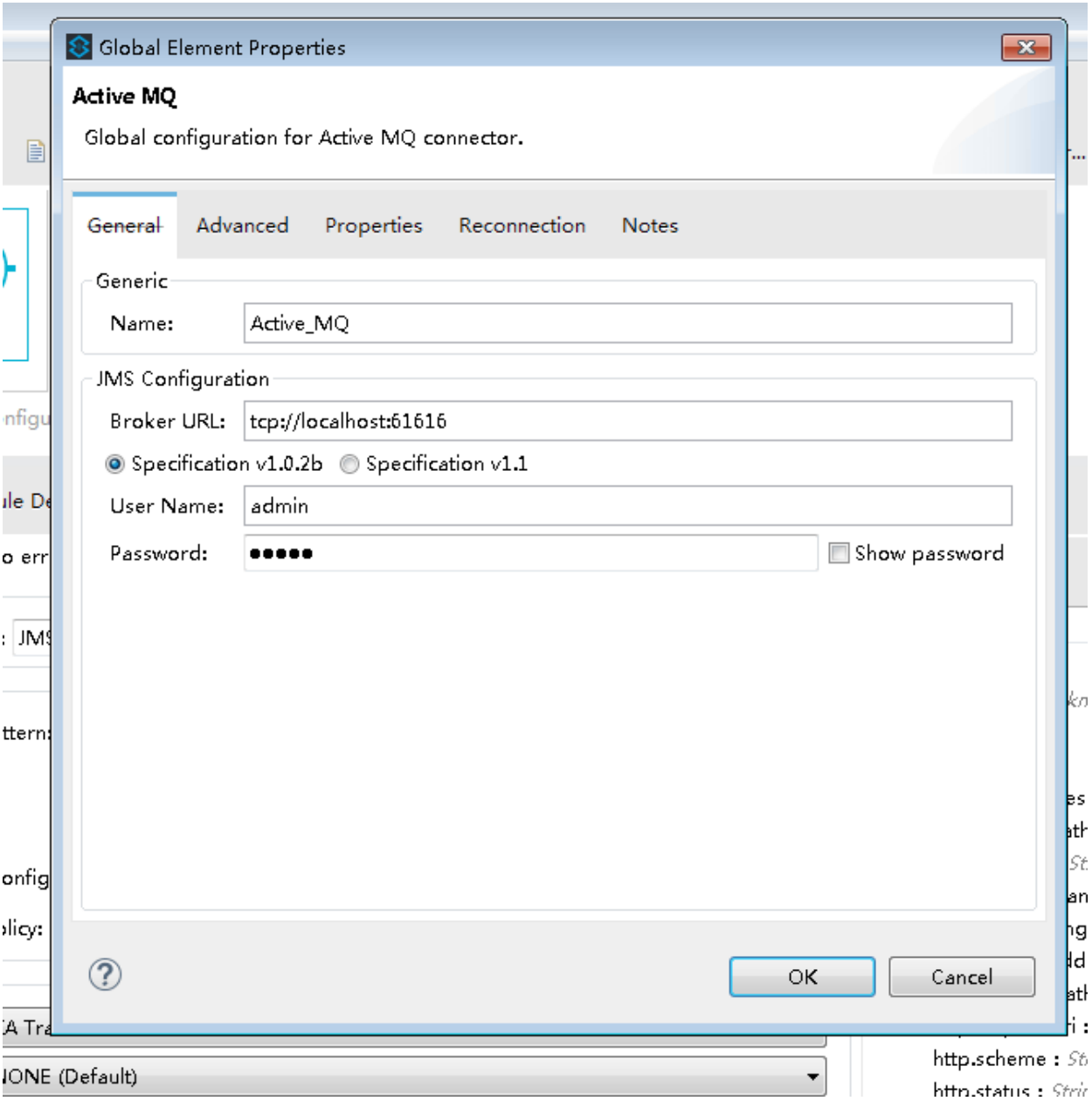
上图的配置为，当数据库丢失时，每隔2000ms 尝试连接，当连接2次时断开

4.4 JMS 组件



1.接受数据的队列

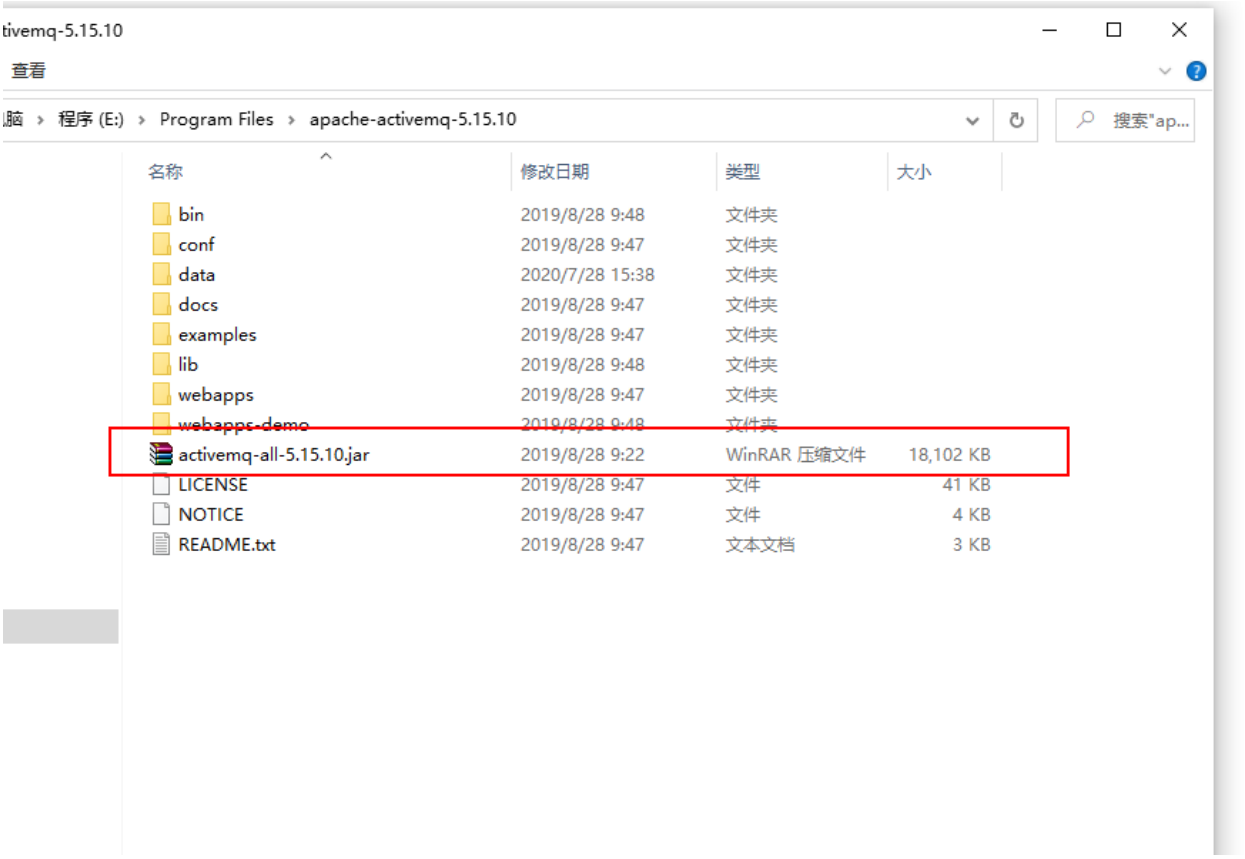
2.连接配置



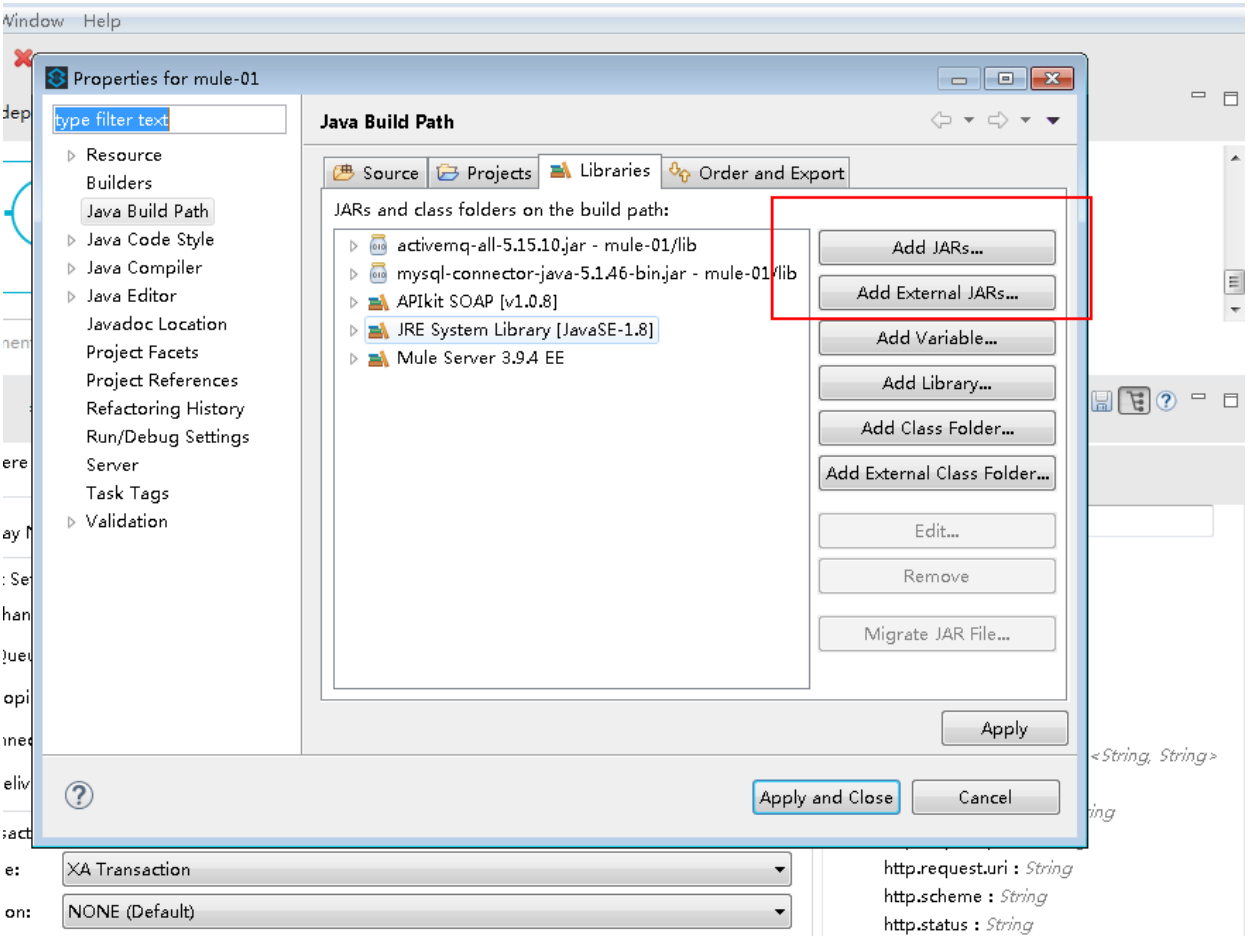
上图连接的时ActivitMQ示例，主要参数为URL，UserName，Password。此处需要注意的时，连接

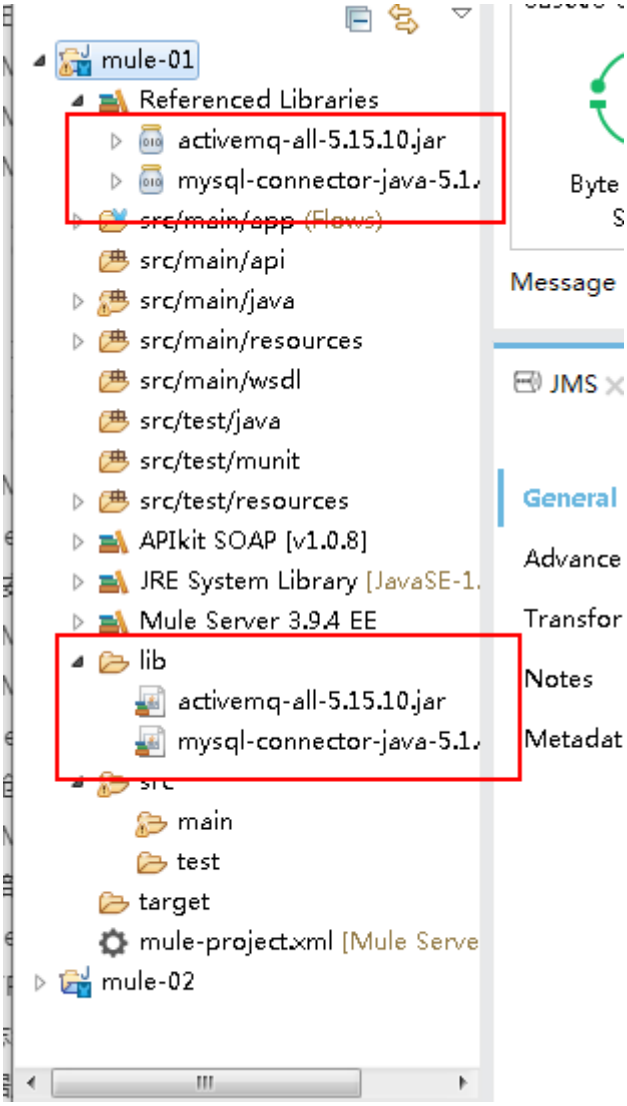
MQ及发送消息需要导入ActivityMQ的驱动包，一般可以从Maven 公共库中获取。

本例中ActiveMQ 依赖包可以从安装目录获取


tivemq-5.15.10
查看
脑 › 程序 (E:) › Program Files › apache-activemq-5.15.10
搜索"ap...
名称
修改日期
类型
大小
bin 2019/8/28 9:48 文件夹
conf 2019/8/28 9:47 文件夹
data 2020/7/28 15:38 文件夹
docs 2019/8/28 9:47 文件夹
examples 2019/8/28 9:47 文件夹
lib 2019/8/28 9:48 文件夹
webapps 2019/8/28 9:47 文件夹
webapps-demo 2019/8/28 9:48 文件夹
activemq-all-5.15.10.jar 2019/8/28 9:22 WinRAR 压缩文件 18,102 KB
LICENSE 2019/8/28 9:47 文件 41 KB
NOTICE 2019/8/28 9:47 文件 4 KB
README.txt 2019/8/28 9:47 文本文档 3 KB

导入至项目


Window Help
Properties for mule-01
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project Facets
Project References
Refactoring History
Run/Debug Settings
Server
Task Tags
Validation
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
activemq-all-5.15.10.jar - mule-01/lib
mysql-connector-java-5.1.46-bin.jar - mule-01/lib
APIkit SOAP [v1.0.8]
JRE System Library [JavaSE-1.8]
Mule Server 3.9.4 EE
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel
XA Transaction
NONE (Default)
http.request.uri : String
http.scheme : String
http.status : String

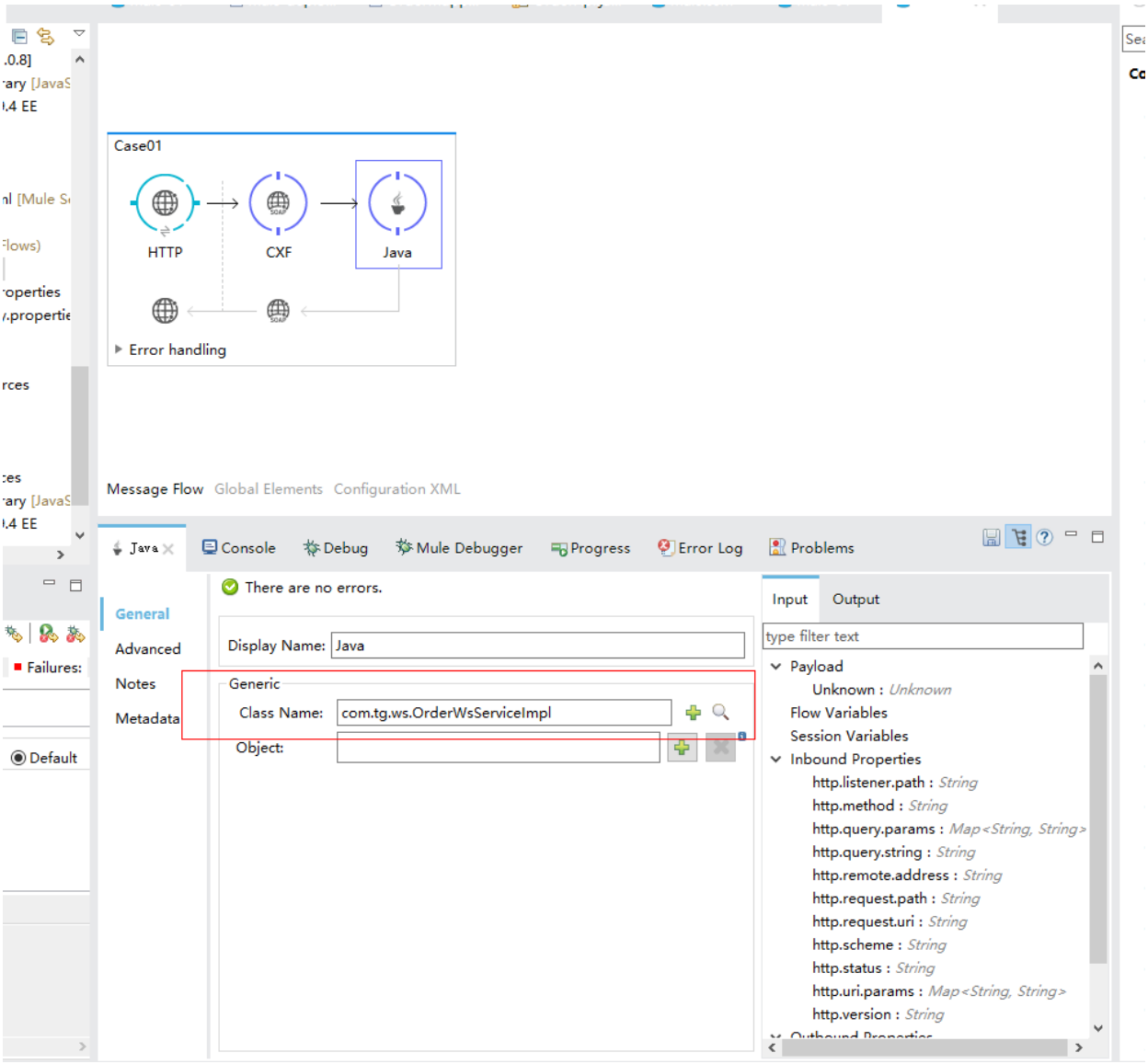


4.5 Java 组件

Java 组件又分为Component 组件及Transform组件,一个是调用方法,另一个则是自定义的数据转换类

4.6.1 Component

该组件使我们可以调用自定义的方法,用途广泛,如 WS接口的建立、数据库方法的调用等



4.6.2 Transform

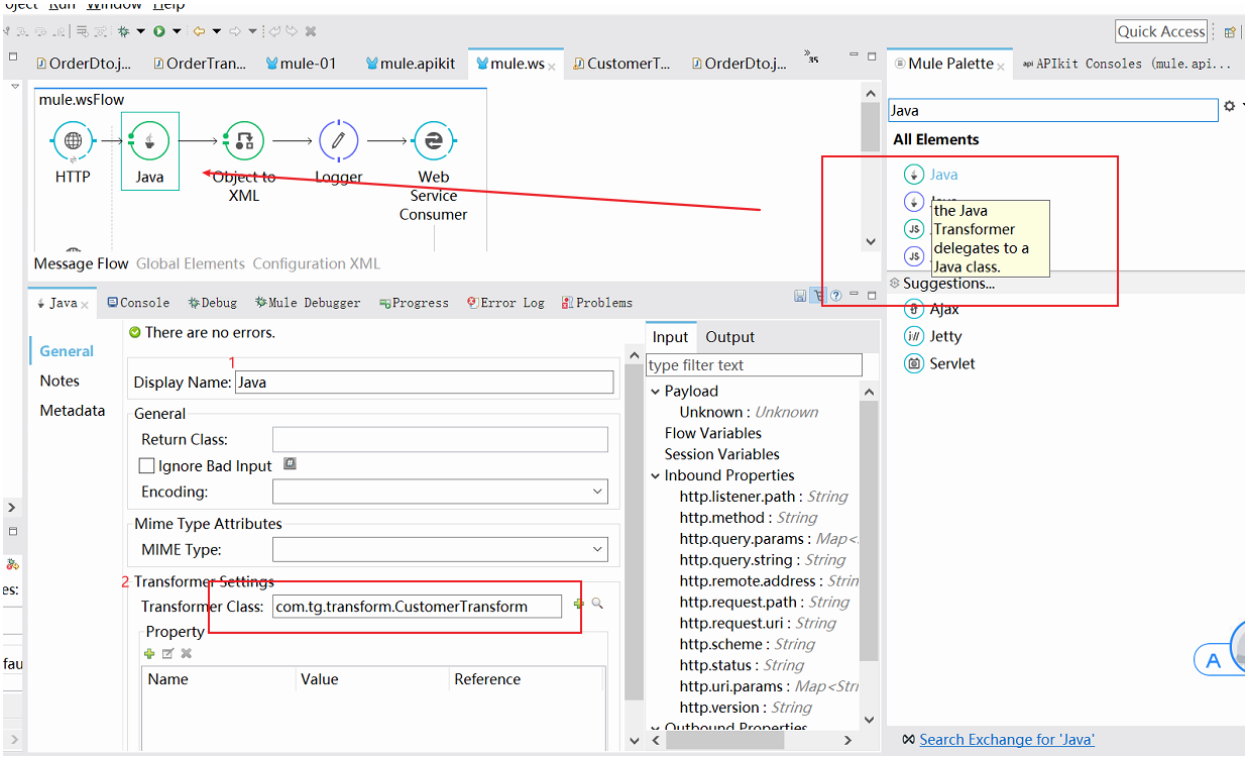
首先创建转换类

```
1  package com.tg.transform;
2  import java.util.Date;
3  import java.util.HashMap;
4  import java.util.Map;
5  import org.mule.api.MuleMessage;
6  import org.mule.api.transformer.TransformerException;
7  import org.mule.transformer.AbstractMessageTransformer;
8  import org.mule.util.UUID;
9  public class CustomerTransform extends AbstractMessageTransformer{
10   @Override
11   public Object transformMessage(MuleMessage message, String encode) thro
ws
12   TransformerException {
13   // TODO Auto-generated method stub
```

```
14    return null;
15    }
16 }
```

该类集成了Mule的抽象方法,当流程至Java组件时会调用默认的transformMessage方法,可以获取用户

请求的消息,经过你的处理,进行对象的返回,可以返回任意类型对象



1. transform 返回的对象类
2. 自定义数据转化类

4.6.3 invoke组件

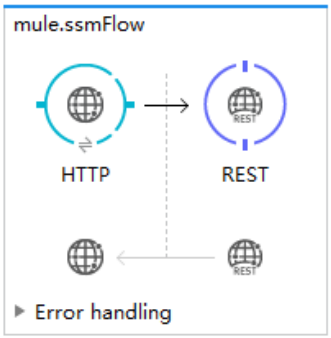
https://blog.csdn.net/Garensimida/article/details/78453420

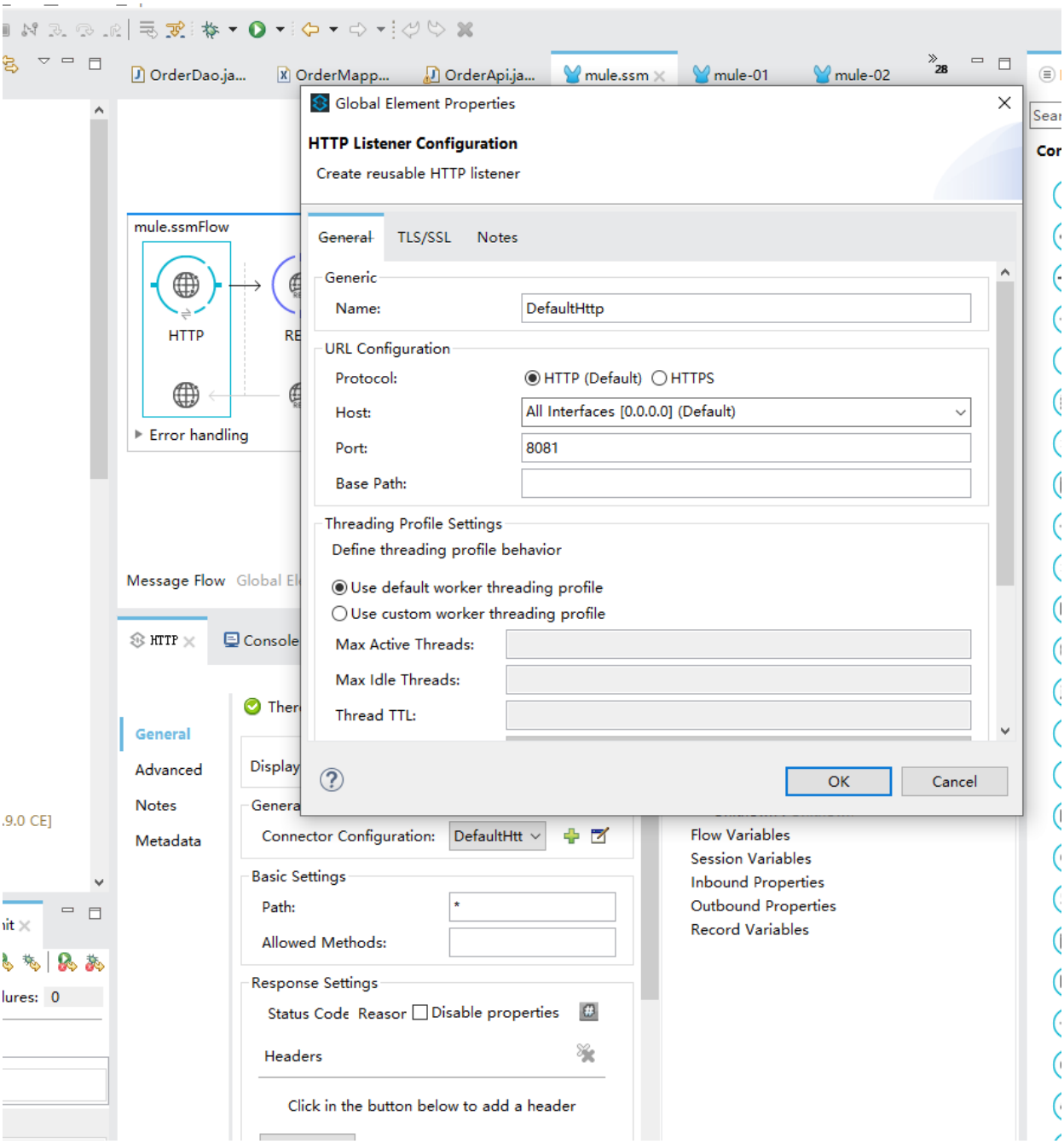
5. Mule 案例

5.1 Mule集成SSM

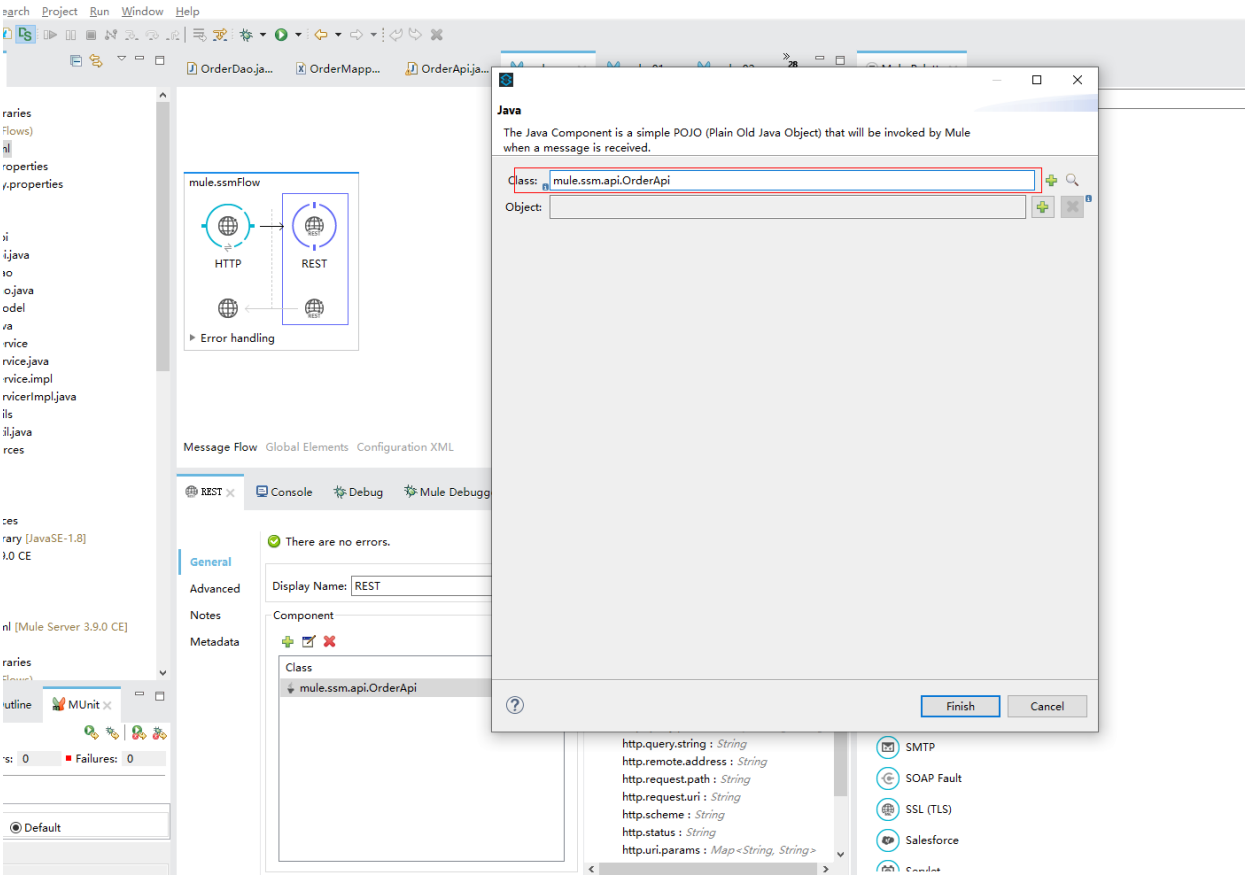
该案例是基于SSM框架，通过OrderId查询订单数据

Mule流程图提供Http接口，关联至相关处理类

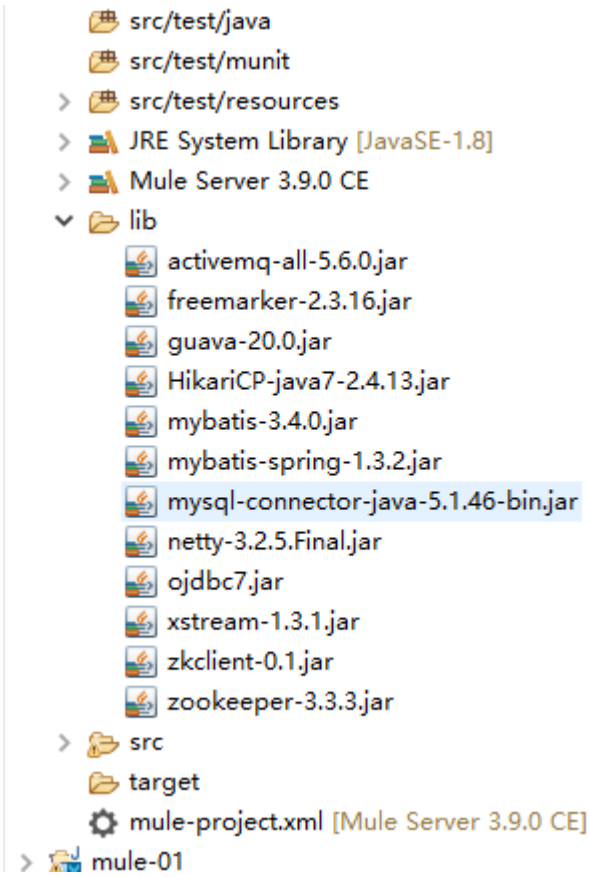

mule.ssmFlow
HTTP
REST
Error handling


Global Element Properties
HTTP Listener Configuration
Create reusable HTTP listener
General
TLS/SSL
Notes
Generic
Name:
DefaultHttp
URL Configuration
Protocol:
HTTP (Default)
HTTPS
Host:
All Interfaces [0.0.0.0] (Default)
Port:
8081
Base Path:
Threading Profile Settings
Define threading profile behavior
Use default worker threading profile
Use custom worker threading profile
Max Active Threads:
Max Idle Threads:
Thread TTL:
OK
Cancel
mule.ssmFlow
HTTP
Error handling
Message Flow
Connector Configuration:
Basic Settings
Path:
Allowed Methods:
Response Settings
Status Code
Reason
Disable properties
Headers
Click in the button below to add a header
Flow Variables
Session Variables
Inbound Properties
Outbound Properties
Record Variables

Http 配置

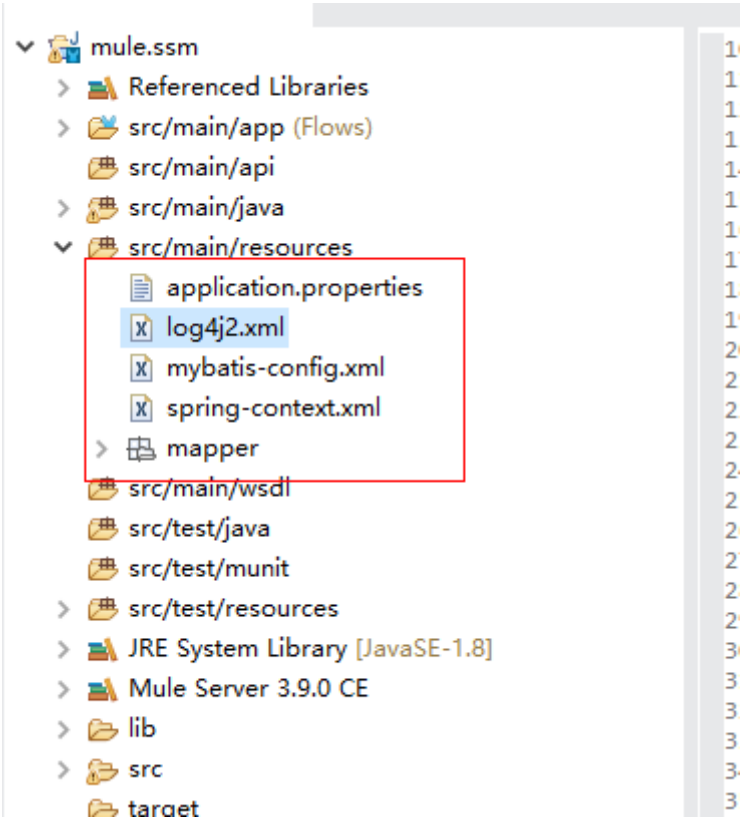


Rest组件关联至Rest 接口处理类



导入SSM 框架的相关Jar包，以上Jar包比较广泛，请自行根据自身情况引用Jar包

在src/main/resources 创建以下文件



spring-context.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-3.0.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx.xsd
 http://code.alibabatech.com/schema/dubbo
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
 <!-- 扫描Bean -->
 <context:component-scan base-package="mule.ssm" >
 <!-- <context:include-filter type="annotation"
 expression="com.alibaba.dubbo.config.annotation.Service" /> -->
 </context:component-scan>
 <!-- 数据库相关 -->
 <bean id="xthis" class="com.zaxxer.hikari.HikariDataSource" destroy-
 method="close">
```

```
<property name="driverClassName" value="${datasource.driver}"/>
<property name="jdbcUrl" value="${datasource.url}"/>
<property name="username" value="${datasource.username}"/>
<property name="password" value="${datasource.password}"/>
<property name="maximumPoolSize" value="50"/>
<property name="minimumIdle" value="2"/>
<property name="dataSourceProperties">
<props><prop key="cachePrepStmts">true</prop>
<prop key="prepStmtCacheSize">250</prop>
<prop key="prepStmtCacheSqlLimit">2048</prop>
<prop key="useServerPrepStmts">true</prop>
</props>
</property>
</bean>
<!-- 配置Mybatis -->
<bean id="jdbcTemplateOra"
class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="xthis"/>
</bean>
<!-- 事务管理 -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManage
r">
<property name="dataSource" ref="xthis"/>
</bean>
<tx:annotation-driven transaction-manager="txManager"/>
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="xthis" />
<property name="configLocation" value="classpath:mybatis-config.xml"/>
<!-- 自动扫描mapping.xml文件-->
<property name="mapperLocations" value="classpath:mapper/*.xml">
</property>
</bean>
<!-- 扫描持久化层 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
<property name="basePackage" value="mule.ssm.dao" />
<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory">
</property>
</bean>
```

```
</beans>
```

application.properties 文件

```
#数据库配置
datasource.url=jdbc:mysql://127.0.0.1:3306/db?
useUnicode=true&characterEncoding=utf8&useSSL=false&autoReconnect=true&serverTim
ezone=GMT%2B8
datasource.username=root
datasource.password=123456
datasource.driver=com.mysql.jdbc.Driver
```

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
<settings>
<setting name="mapUnderscoreToCamelCase" value="true"/> <!-- 驼峰命名 -->
<setting name="lazyLoadingEnabled" value="true"/> <!-- 是否开启懒加载 -->
<setting name="logImpl" value="STDOUT_LOGGING"/> <!-- 打印查询日志 -->
<setting name="callSettersOnNulls" value="true"/> <!-- 为null的字段忽略 --
>
</settings>
</configuration>
```

业务逻辑代码的编写

实体类 Order.java

```
package mule.ssm.model;
import java.time.LocalDate;
import java.io.Serializable;
/**
*
* 订单

*/
public class Order implements Serializable {
 private static final long serialVersionUID = 1L;
 private String rowId;
 private String orderId;
 private LocalDate orderDate;
```

```
14 private LocalDate shipDate;
15 private String shipMode;
16 private String customerId;
17 private String customerName;
18 private String Segment;private String City;
19 private String State;
20 private String Country;
21 private String postalCode;
22 private String Market;
23 private String Region;
24 private String productId;
25 private String Category;
26 private String subCategory;
27 private String productName;
28 private Double Sales;
29 private Integer Quantity;
30 private Double Discount;
31 private Double Profit;
32 private Double shippingCost;
33 private String orderPriority;
34 // 省略getter setter 方法
35 }
```

Controller 层

```
1 package mule.ssm.api;
2 import java.util.Map;
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.QueryParam;
7 import org.mule.api.MuleContext;
8 import org.mule.api.context.MuleContextAware;
9 import mule.ssm.model.Order;import mule.ssm.service.OrderService;
10 import mule.ssm.service.impl.OrderServicerImpl;
11 import mule.ssm.utils.SpringUtil;
12 @Path("/order")
13 public class OrderApi implements MuleContextAware{
14 @GET
15 @Path("/get")
16 public Order getOrder(@QueryParam("orderId") String orderId) {
17 Order order = null;
```

```
System.out.println("------ 请求的数据为 ----------");
System.out.println(orderId);
try {
OrderService orderService = (OrderService)
SpringUtil.getBean(OrderServicerImpl.class);
order = orderService.getOrderById(orderId);
}catch(Exception e) {
e.printStackTrace();
}
return order;
}
@Override
public void setMuleContext(MuleContext context) {
// TODO Auto-generated method stub
}
}

```

Service 层及实现

interface

```
package mule.ssm.service;
import org.springframework.stereotype.Service;
import mule.ssm.model.Order;
public interface OrderService {
 Order getOrderById(String orderId);
}
```

impl

```
package mule.ssm.service.impl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import mule.ssm.dao.OrderDao;
import mule.ssm.model.Order;
import mule.ssm.service.OrderService;
@Service
public class OrderServicerImpl implements OrderService {
@Autowired
 OrderDao orderDao;
 @Override
 public Order getOrderById(String orderId) {
 // TODO Auto-generated method stub
```

```
Order order = null;
try {
order = orderDao.selectByOrderId("ES-2011-1406120");
} catch (Exception e) {
e.printStackTrace();
}
order.setOrderId("jdfsdf45485");
return order;
}
}
```

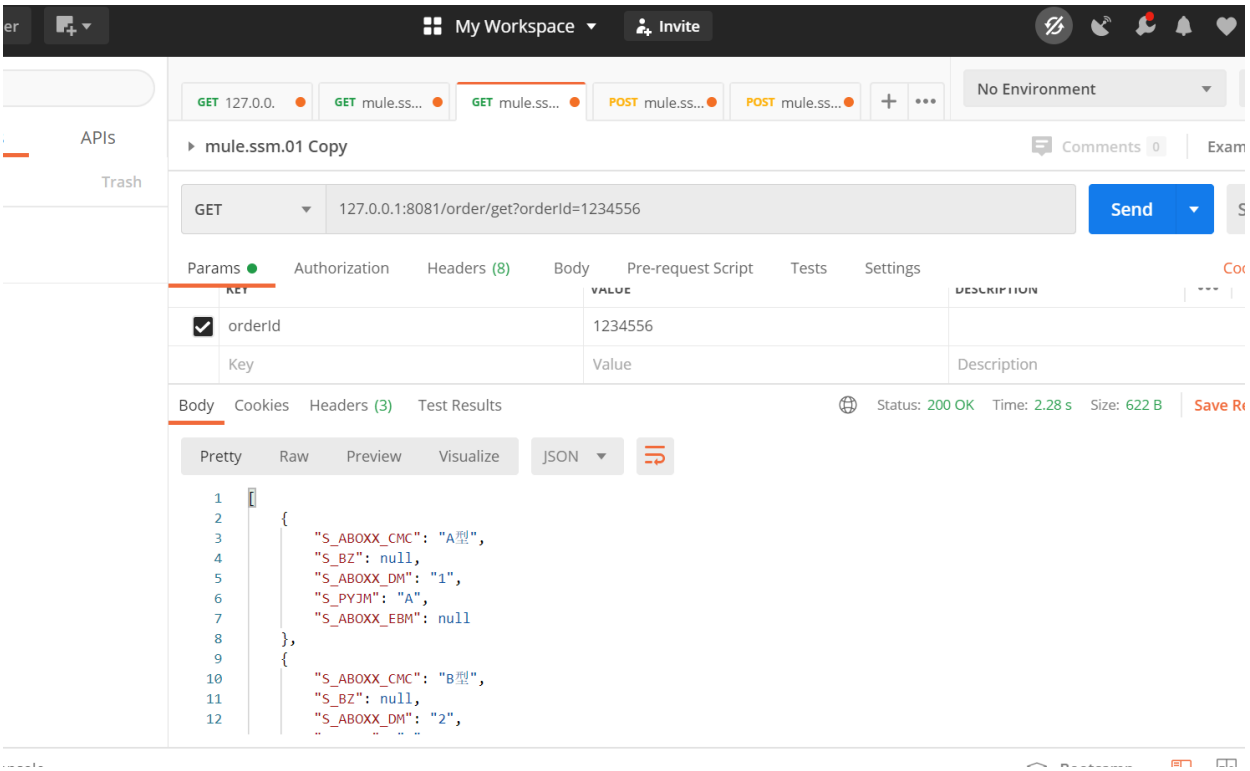
Dao 层的实现

```
package mule.ssm.dao;
import org.apache.ibatis.annotations.Param;
import org.springframework.stereotype.Repository;
import mule.ssm.model.Order;
@Repository
public interface OrderDao {
 Order selectByOrderId(@Param("orderId")String orderId);
}
```

Mapper 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">调用服务
6. 附录
6.1 一些好的建议
<mapper namespace="mule.ssm.dao.OrderDao">
<!-- 通用查询结果列 -->
<sql id="Base_Column_List">
Row_ID, Order_ID, Order_Date, Ship_Date, Ship_Mode, Customer_ID,
 Customer_Name, Segment, City, State, Country, Postal_Code, Market, Regio
n,
 Product_ID, Category, Sub_Category, Product_Name, Sales, Quantity, Disco
unt,
 Profit, Shipping_Cost, Order_Priority
 </sql>
 <select id="selectByOrderId" parameterType="java.lang.String"
 resultType="mule.ssm.model.Order">
 select * from tb_order where Order_ID=#{orderId} limit 1
 </select>
 </mapper>
```

调用服务



6. 附录

6.1 一些好的建议

这个微服务框架资料贼少，还有不要用springcloud等组件的经验去玩mule，容易翻车