1. mule_esb 介绍

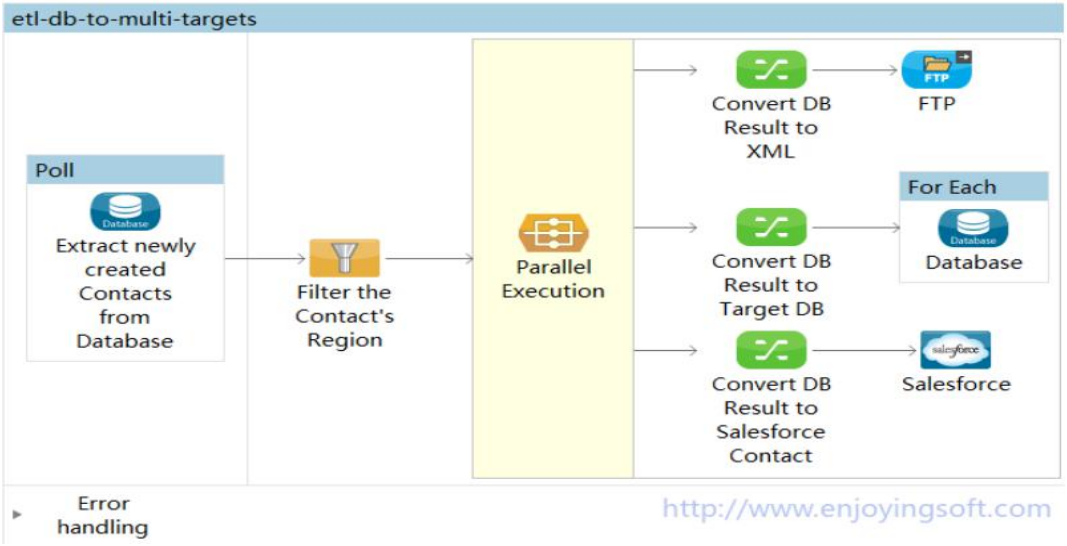
官网：https://www.mulesoft.com

1. 运用场景：

1.旧系统改造，开放系统的服务能力。举个例子，我们有一个电商系统，需要调用 SAP ERP 的订单接口来创建订单。这个时候就需要将 SAP 的订单服务暴露成我们流行的 Rest Service 接口，以方便电商系统调用。当然电商系统可以直接使用 SAP 的 JCO 包来调用 SAP 的 BAPI Function，但显然暴露一个通用的 Rest Service 更易于调用。使用 Mule ESB 实现如下：

2.系统集成。举个例子，很多系统之间数据交互可能还是用 FTP 目录。尤其是企业跟企业之间的数据交互，比如，A 企业丢一个 EDI 文件到 B 企业的 FTP 目录，然后 B 企业会从 FTP 目录下载解析并放置到数据库。这个场景用 Mule ESB 实现就很方便。

3.ETL。市面上有很多开源的 ETL 软件，其实 Mule ESB 也有 ETL 的功能，通过 Flow 设定 ETL 的数据转换和数据流向。下图就演示了 Mule 的 ETL 和数据分发功能。



2. Mule ESB 软件安装

用一个相对宽泛的标准来划分，Mule ESB 软件可以分成两部分。

一部分是客户端，也就是基于 Eclipse 的 Anypoint Studio，客户端就是用来开发集成应用的，通常面向的用户就是我们的开发人员。由于是基于 Eclipse 环境，所以 Java 开发人员会很熟悉，同时 Anypoint Studio 也可以集成 Git，SVN，Maven，Junit，Jenkins 等一系列生态工具。

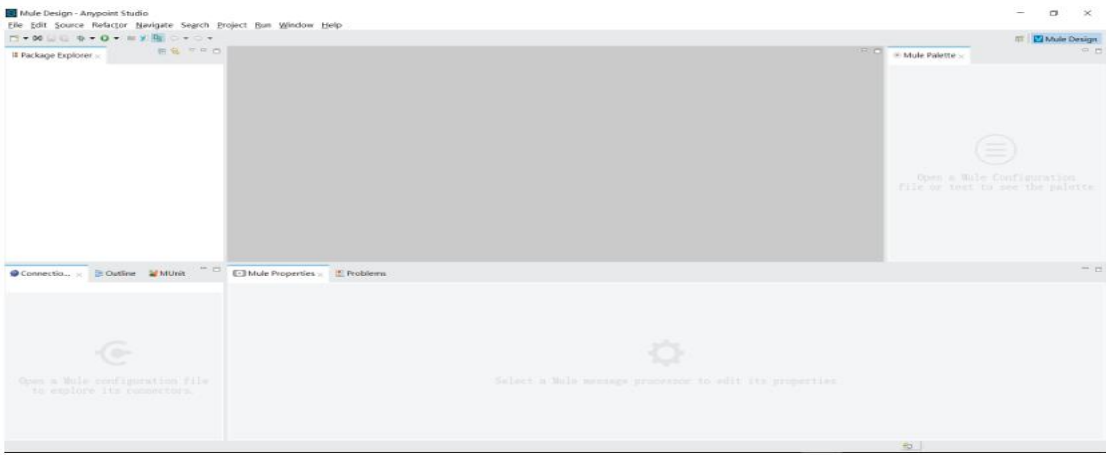
当然作为其他语言的开发人员也不用担心，Anypoint Studio 是一个可视化的拖拽的开发环境，大部分的工作只需要使用鼠标拖拽组件，配置属性即可。

还要一部分是服务端，也就是 Mule Runtime，也可以称作 Container，容器，注意不要跟 Docker 的概念混淆。这里的 Container 指的是 Java Container，你可以大概想象成 Tomcat 类似的东西。客户端开发的应用就部署运行在 Mule Runtime 中。

需要注意的是，Mule Runtime 是一个独立的 Container，只依赖 JDK，并不依赖其他第三方的容器。服务器中只需要安装 JDK 和 Mule Runtime，即可运行 Mule 应用。还有一种不太常见的做法，就是把 Mule App 跑在第三方的容器里(比如 Tomcat 等)，通过对容器的扩展来支持 Mule App。不过我们不推荐这种做法，还是推荐使用独立的 Mule Runtime，也就是官方说的 Standalone 版本。

客户端安装

Anypoint Studio 是基于 Eclipse 构建的，你可以在这里下载到(Windows 64Bit，Mac 64Bit)。请根据你的客户端的操作系统选择对应的版本。我们只要在 Windows 或者 Mac 电脑上安装好 JDK8 之后，解压缩 Zip 包，双击即可打开。打开的界面如下：

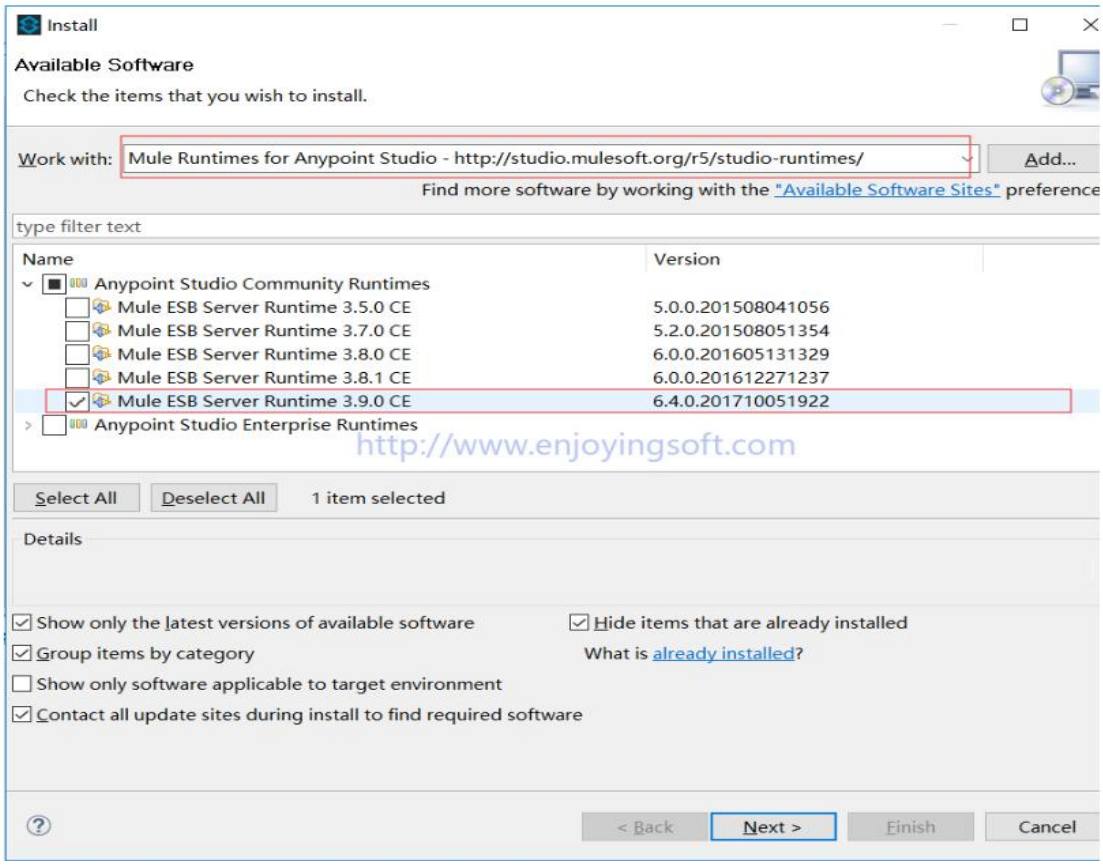


由于 Anypoint Studio 默认只内嵌了 Mule ESB 企业版运行时，如果你想使用社区版的话，我们还需要安装 Mule ESB 社区版运行时。为什么客户端也需要 Runtime？这很容易理解，我们开发调试都需要使用 Runtime，难道你编写代码后，不运行和调试吗-:)

安装社区版运行时，可以使用下列步骤。

点击 Help/Install New Software...菜单

在 Work with 下拉框中选择 Mule Runtimes for Anypoint Studio



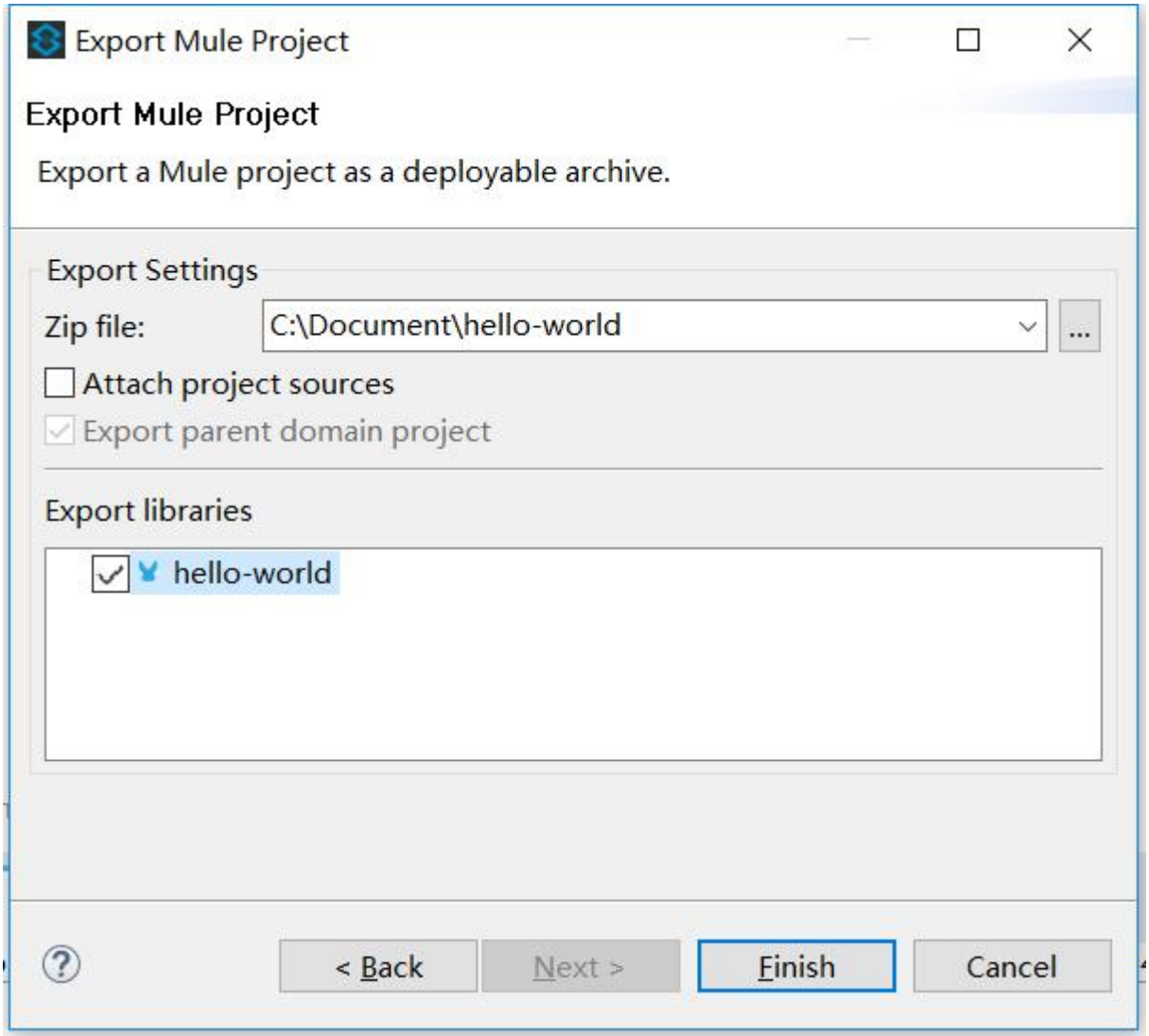
选中 Mule ESB Server Runtime 3.9.0 CE，CE 就是 Community Edition，社区版的意思。然后点击 Next 一直到完成。

服务端安装

服务端不区分操作系统，你可以从这里下载到 3.9.0 版本的社区版 Runtime。Mule ESB Runtime 可以运行在 Windows,Linux,Mac 等操作系统上，我们推荐在生产环境使用 Linux 部署 Mule ESB Runtime。服务端的安装也很简单，只要在服务器上安装好 JDK8 之后，解压缩 Runtime 即可。

3. 部署 Mule ESB 应用

我们已经在客户端开发完成了 Hello world，也顺利跑起来了。但是如何部署到服务端呢。和常见的 Java 开发一样，我们也需要打包应用。这里我们用的是普通工程，我们通过一个 Export 操作，即可导出 Mule App（当然 Mule App 也支持 Maven 工程，可以通过 Maven 打包）。在工程项目上右击弹出菜单，Export...，然后在对话框中选择 Anypoint Studio Project to Mule Deployable Archive。注意这里不要选中 Attach project sources。



将导出的 Zip 包放入到 Mule Runtime 的 apps 目录下，然后启动 Mule Runtime。如果是 Linux 或者 mac，那么在 Mule Runtime 的 bin 目录下执行`./mule`启动命令。./mule restart 重启命令．如果是 Windows，双击执行 bin 目录 mule.bat 即可启动。

4. 使用 Anypoint Studio 开发 ESB 项目

Anypoint Studio：是 MuleSoft 提供的基于 Eclipse 架构的集成开发环境（IDE）。使用 Anypoint Studio，程序员可以几分种内轻松地创建出集成化的流程图，并使用图形化转换器来完成数据的映射。这里的映射就是指消息转换，消息转换是 ESB 很重要的功能。Anypoint Studio 的图形化转换器功能非常强大，可以全方面的转换数据，包括数据结构，数据类型，数据内容的转换等。注意，图形化转换器是企业版才有的功能，社区版则需要自行开发，我们在社区版做过类似的图形化转换工具。

Anypoint Studio 可以让开发人员用拖拽的方式把连接器(Connectors)、转化器(Transformers)以及其他 Processor 放到开发环境的画布上，Anypoint Studio 会自动把图形化的流程界面转化为后台的 XML 配置文件。

下图是 Anypoint Studio 的各个区域说明

1. 包资源

   和 Java 工程类似，管理工程项目的资源。

2. 画布

主要的可视化开发环境，通过拖拽的方式组织逻辑，定义属性。

3. 工具箱

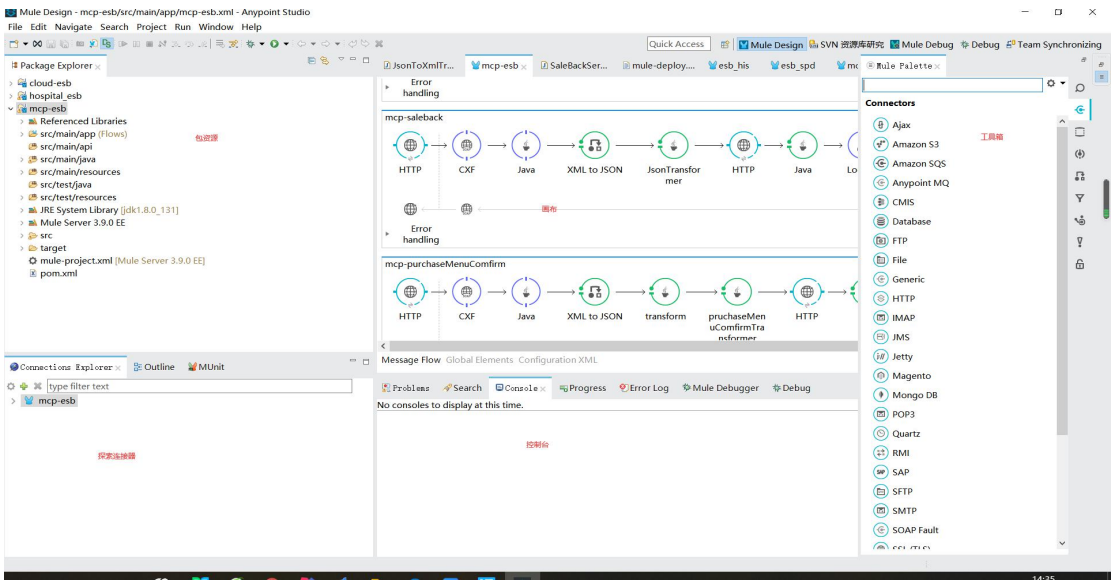
包括 Connector,Processor 等开箱即用的组件。

4. 连接探索器

这里可以展示全局的连接器配置属性。

5. 控制台

和 Java 工程一样，Console 输出控制台



5.Mule ESB 应用程序结构

我们再来看看 Mule ESB Application 的结构是什么。从图示可以发现，Mule 的应用程序和 Java 应用程序几乎一致，其中有几个新目录着重介绍一下。

1. src/main/app

这个目录就是放置 Mule 的配置文件，也就是 Mule Configuration File。打开 Mule Configuration File 就会开启设计器界面。

2. src/main/api

这个目录是放置 Restful API 的定义文件。

3. src/main/java

这个目录是放置 java 源文件。

4. src/main/resources

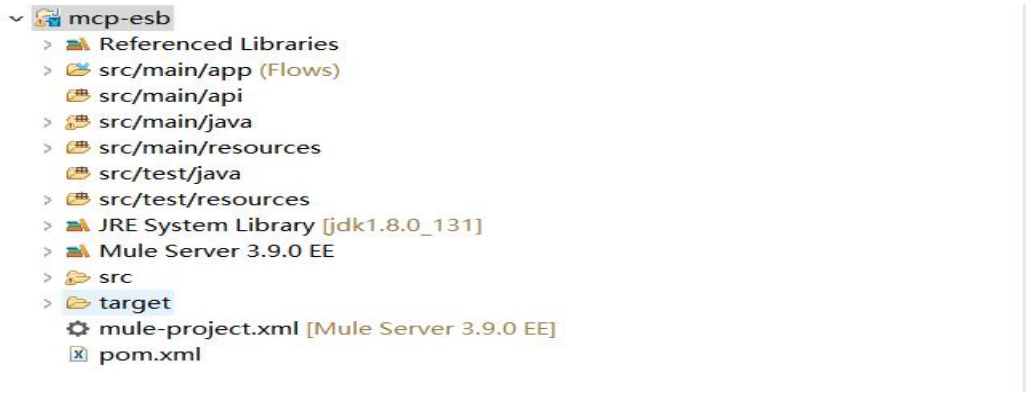
这个目录主要存放项目的资源文件

5. src/test/java

这个目录是放置 java 单元测试代码源文件

6. src/test/resources

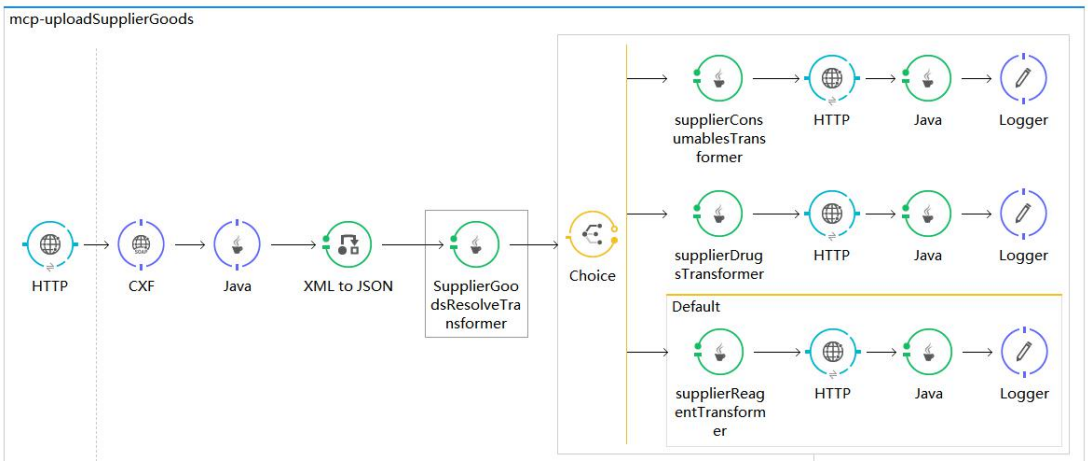
这个目录是放置 java 单元测试所需资源文件。



6.Mule ESB Application 整体构造

Mule 的应用程序就是由一个或者多个 Mule Configuration File 组成，每个 Mule Configuration File 里面可以放置一个或者多个 Flow。每一个 FLow 又是由 Connector 和 Processor 等组成。

Flow 是 Mule 的核心概念，下图展示了 Flow 的结构



1. Mule ESB 构造元素 - Flow

Mule ESB 的应用程序通常是设计用来接收和处理消息。接收消息我们通常使用 Connector Source 来做，而处理消息通常使用一个或者多个 Processor 来做。Flow 就是用来组织 Connector 和 Processor 的组。在一个 Flow 中，你可以将多个单独的 Mule 元素链接起来，用来实现接收，处理，过滤，路由消息等功能。

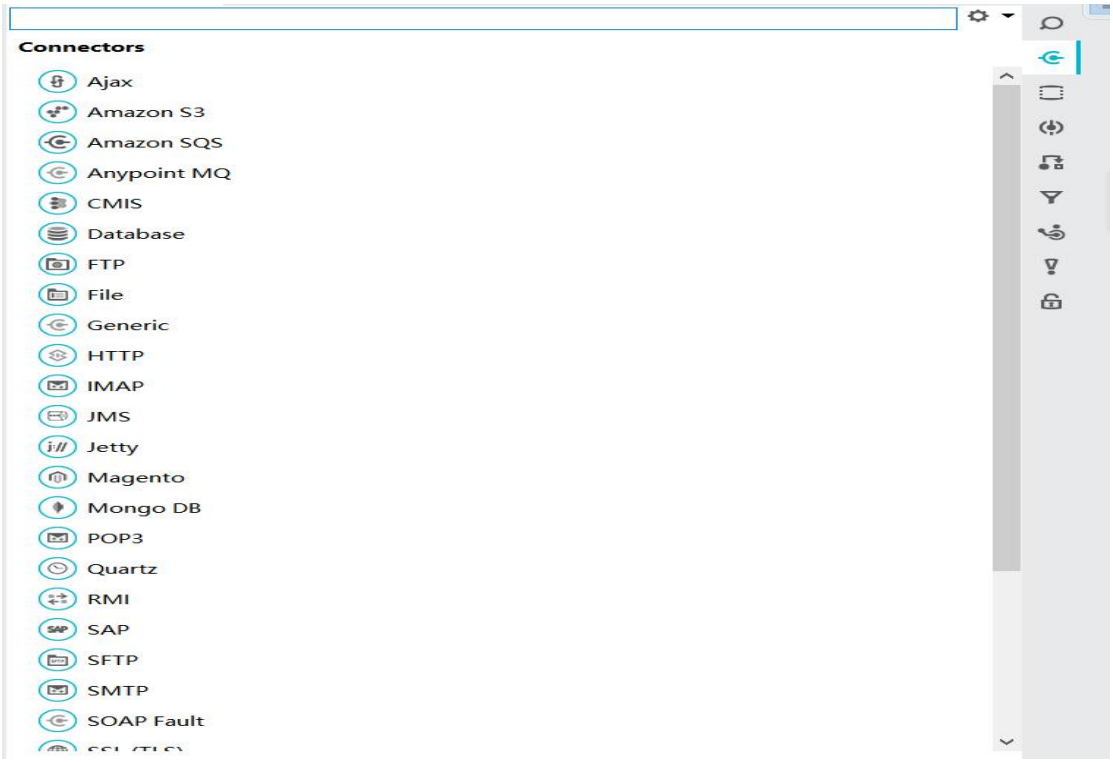
Flow 实际就是上图的边框。实际还有 Sub Flow 的概念，主要用于 Flow 的公用，这些不展开讲述。

2. Mule ESB 构造元素 - Connector

Mule 的 Connector 是其非常优秀的功能，数百个开箱即用 Connector 可以帮助开发者连接不同的应用。从常见的 HTTP，TCP，FTP，LDAP 等协议。

3. Mule ESB 构造元素 - Processor

Mule 的 Processor 包含的内容更广泛，从 Studio 右侧的工具箱可以看到很多的控件元素，除去上文讲述的 Connector，余下的基本都可以归纳到 Processor。



Processor 分类

1. **Transformers**

可以称作转换器，用来转换消息的类型，结构和内容，比如将 XML 换成 JSON。

2. **Components**

组件，可以使用 Java 或者脚本语言组件，比如 JavaScript 等。这些组件使用程序语言来描述商业逻辑。

3. **Flow Control**

控制消息的流向，比如消息的路由，消息的分割聚合等。

4. **Scopes**

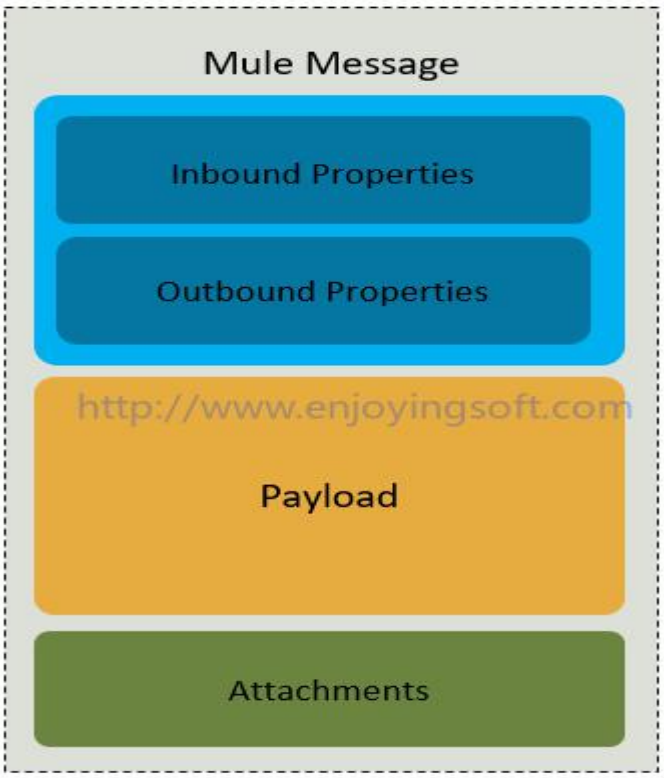
通过 Scope，我们可以改变内部 Processor 的行为特征。

5. **Filters**

过滤消息，我们可以定义规则过滤非法的消息。

4. Mule Message 结构

Flow 中在各个节点之间流动的就是 Mule Message，Mule Message 是一个数据结构，也有相对应的 Java Class。它包括几部分 Payload，Property，Attachment.



如何理解这幅图，大致可以和 HTTP 协议类比。

Property

Mule Message 的 Property 又分成 Inbound Properties 和 Outbound Properties。这一点类似于 HTTP 协议的请求头和响应头。

Payload

Mule 的 Payload 是一个对象，类型是不固定的。可能是 Stream，也可能是 Hashmap，也可能是 XML 字符串。这一点类似于 HTTP 协议的请求正文，或者说是请求体。

Attachment

Mule 的 Attachment 就是消息的附件，这一点类似于 HTTP 协议中的 multipartform-data 请求。

如果你想看到整个 MuleMessage 的结构，使用 Mule 的 Logger 组件可以很方便的看到 Message 完整的组成。使用 Logger 打印出 message，logger 组件会重载 message 的 toString 方法，打印出 Pretty 格式的 message。

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-current.xsd
http://www.mulesoft.org/schema/mule/core
http://www.mulesoft.org/schema/mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/http
http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd">
    <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081"
doc:name="HTTP Listener Configuration"/>
    <flow name="loggertestFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-payload value="#[&quot;Mule Message&quot;]" doc:name="Set Payload"/>
        <logger message="#[message]" level="INFO" doc:name="Logger"/>
    </flow>
</mule>
```

我们可以从下图的记录中找到和上图 Message Structure 相对应的节点

```
org.mule.DefaultMuleMessage
{
  id=f88d0090-074c-11e9-89b7-0c5415358ba9
  payload=java.lang.String
  correlationId=<not set>
  correlationGroup=-1
  correlationSeq=-1
  encoding=UTF-8
  exceptionPayload=<not set>

Message properties:
  INVOCATION scoped properties:
  INBOUND scoped properties:
    accept=*/*
    accept-encoding=gzip, deflate, br
    accept-language=zh-CN,zh;q=0.9,en;q=0.8
    cache-control=no-cache
    connection=keep-alive
    content-length=2
    content-type=text/plain;charset=UTF-8
    host=localhost:8081
    http.listener.path=/
    http.method=POST
    http.query.params=ParameterMap{[]}
    http.query.string=
    http.relative.path=/
    http.remote.address=/127.0.0.1:57630
    http.request.path=/
    http.request.uri=/
    http.scheme=http
    http.uri.params=ParameterMap{[]}
    http.version=HTTP/1.1
  SESSION scoped properties:
```

1. Mule Message 的 Payload

Payload 它是 Mule Message 的主要部分，也是 Mule 处理的主要对象。我们后续说的数据转换就是对 Payload 的转换。注意 Mule Message 的 Payload 是有可能为空的，比如接收到一个 Http Get 请求，Http Get 请求的请求体是空的，所以这个时候 Mule Message 的 Payload 是空的。

在 Flow 中，最常用的动作就是给 payload 赋值，给 Payload 赋值会使用 set-payload 组件。如果我们在 Flow 中想获取 payload，可以使用 MEL 表达式。

下面的代码表示 payload 的赋值和取值。

```
<flow name="payloadFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-payload value="#[&amp;quot;Mule Message&amp;quot;]" doc:name="Set Payload"/>
        <logger message="#[payload]" level="INFO" doc:name="Logger"/>
</flow>
```

2. Mule Message 的 Property

Mule Message 的 Property 是一个键值对，有 name 和对应的 value。Mule Message 有两种类型的 Property，Inbound Properties 和 Outbound Properties。Inbound Properties 或者 Outbound Properties 可以有多个 Property，也就是多个键值对。

Inbound Properties 是不可变的，是由 Message Source 产生的。就类似于 Http 的请求参数，是由用户的数据请求，经过 Java 的 Servlet，或者 Asp.Net 等框架封装成 Http Request 对象。

Outbound Properties 是可变的，我们在 Mule 的 Flow 中新增或者改变这些属性。注意，比如转换器，有些 Mule Processor 会自动增加有些属性。

在 Mule 中设定 Property 使用 set-property 组件，如果需要获取，同样使用 MEL 表达式。详细的 MEL 表达式，我们下篇会展开讲解。

```
<flow name="propertyFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-property propertyName="#[&quot;userName&quot;]" value="#[&quot;Tom&quot;]"
doc:name="Property"/>
</flow>
```

3. Mule Message 的 Attachment

Attachment，可以理解成消息的附件。想象一封邮件，有邮件发送人等头信息，也有邮件正文，同样还有邮件附件。和 Property 一样，Attachment 也有两种类型，Inbound Attachment 和 Outbound Attachment。我们通常将一些大的对象作为附件传输。

使用 set-attachment 设置附件，这里将 payload 作为 pdf 文档附件供消费者下载。

```
<flow name="attachmentFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-attachment attachmentName="#[&quot;doc&quot;]" value="#[payload]"
contentType="application/pdf" doc:name="Attachment"/>
</flow>
```

4. Mule 的 Variable

Variable 也就是变量，有几种类型的变量，或者说几种不同范围的变量，如下：Flow Variable，Session Variable，Record Variable。Flow Variable 在一个 Flow 是有效的，Session Variable 是可以跨 Flow 的，Record Variable 则是处理数据列表时会用到。

这里不详细讲述。从使用上说，有些类似于 Java 里面的局部变量，Session 变量，但不完全一致。后续实战文章会分析这一点。

在 Mule 里，使用 set-variable 和 MEL 表达式对变量做赋值和取值操作。

```
<flow name="variableFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-variable variableName="orderNo" value="#[&quot;1238&quot;]"
doc:name="Variable"/>
</flow>
```

5. 使用 Java 操作 Mule Message

如何使用 Java 操作 Mule Message 呢？通过 Java 代码我们可以清楚的看到 Mule Message 的结构，成员变量和方法等。

```
public void explorMessage(MuleMessage message) {
    // 获取InboundProperty
    String requestPath = message.getInboundProperty("http.request.path");
    // 设定OutboundProperty
    message.setOutboundProperty("content-type", "application/json");
    // 获取Payload
    Object payload = message.getPayload();
    // 获取InboundAttachment
    DataHandler fileAttachment = message.getInboundAttachment("fileName");
    // 获取flow变量
    message.getProperty("flowVarTest", PropertyScope.INVOCATION);
}
```

7.Mule Expression Language - MEL 表达式

1. MEL 的优势

在 Mule ESB 上有很多方法可以操作 Mule Message，比如 Java 语言或者其他脚本语言(比如 JavaScript 等)。但是 MEL 表达式是 Mule 推荐使用，在 Mule 应用中的一个统一和标准的方法。

MEL 表达式为开发人员提供了一个一致的标准化语言，用来访问和计算 Mule Message 的 Payload（负载），Property（属性）和 Variable（变量）。
MEL 基于 Mule 特定的对象，Studio 中提供 auto-complete（自动完成，语法提示）的功能，帮助开发者快速编码。
更重要的是，Mule 的绝大多数组件都支持 MEL，比如路由组件，过滤组件等。
MEL 的示例，这个示例在在 Mule 的 Logger 组件中使用 MEL 表达式获取 FlowVars。
从下图可以看到，我们在 Logger 组件中使用 MEL 表达式，能够提供语法提示，该提示带出了上一步设定的 customerNo 变量。
Xml 配置如下：

```
<flow name="mel-flow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <set-variable variableName="customerNo" value="#[1008]" doc:name="customerNo"/>
        <logger message="The customerNo is #[flowVars.customerNo]" level="INFO"
doc:name="Logger"/>
</flow>
```

注意：MEL 是一种表达式，和脚本语言类似，但并不相同。表达式通常用于动态获取值或者设定值，或对数据进行简单的操作。表达式语言和脚本语言之间在功能上存在重叠，但如果您编写的内容非常复杂，需要的不仅仅是几行代码，或者您需要包含条件逻辑，那么脚本语言通常会更有用。如果简单的获取或设定值，调用方法或执行函数，则使用表达式则更方便。

2. MEL 的使用场景

MEL表达式常用的使用场景大概可以分成三种。

- 获取值
  - `#[payload]`
    - 表示获取message的负载
  - `#[message.inboundProperties.'http.query.params'.customerNo]`
    - 表示获取查询参数customerNo
  - `#[payload.callMethod(parameters)`
    - 表示调用payload对象的callMethod方法，并获取方法返回值
  - `#[xpath('//root/element')]`
    - 表示使用xpath语法解析并获取相应节点内容。
- 条件比较，返回的结果就是布尔变量
  - `#[payload.amount > 2000]`
  - `#[message.inboundProperties.'http.method' == 'GET']`
    - 表示判断HTTP请求是不是GET方法
- 设定值，通常用于Message Enricher组件。
  - `#[flowVars.dbResult]`
    - 这里表示相应的值设定到dbResult变量中。

3. MEL 的示例

使用表达式提取值，根据消息的内容，属性决定执行流程。在下面的示例中，payload 是一

个 Java 对象，我们根据购买类型，将订单分发路由到不同的 JMS 消息队列中。

```
<choice>
    <when expression="#[payload.getOrderType() == 'book']">
          <jms:outbound-endpoint queue="bookQueue" />
     </when>
    <when expression="#[payload.getOrderType() == 'music']">
          <jms:outbound-endpoint queue="musicQueue" />
     </when>
</choice>
```

使用表达式提取值，并将值传递给 Connector，如下示例就是使用 MEL 计算的值设定 SMTP Connector 的邮件标题，邮件接收人等。

```
<smtp:outbound-endpoint from="#[flowVars.mailFrom]" to="#[flowVars.mailTo]" subject="#[payload.mailSubject]"  doc:name="SMTP"/>
```

如果 payload 是 Java 对象，可以调用 payload 方法，获取方法的返回值。示例就说调用 calAmount 方法，并打印计算出来的金额。

```
<logger message="#[payload.calAmount()]" />
```

4. MEL 的上下文对象

我们在上述的 MEL 表达式示例中可以看到 MEL 有多个部分组成，第一部分就是上下文对象。MEL 常见的上下文对象如下：

上下文对象	说明
#[server]	当前服务器，可以获取服务器的时间，JDK版本等，如#[server.dateTime]，#[server.javaVersion]
#[mule]	当前Mule实例，可以获取Mule的版本，目录等。如#[mule.version]
#[app]	当前Mule应用的实例，可以获取应用的名称等。如#[app.name]
#[message]	这个是我们最经常使用的对象，就说Mule message。如#[message.payload]，#[message.inboundProperties.'http.query.params'.customerNo]等

server 上下文对象的常用属性：

Field	Field描述
dateTime	系统当前时间
host	主机名
ip	主机IP
osName	操作系统名称
userName	当前用户
userDir	当前用户工作目录

mule 上下文对象的常用属性：

Field	Field描述
home	Mule Runtime的安装目录
version	Mule Runtime的版本
nodeId	集群下的本机ID
clusterId	集群ID

message 上下文对象的常用属性：

Field	Field描述
id	message的唯一ID
rootId	message的根ID
payload	message的负载
inboundProperties	message的inbound头信息
inboundAttachments	message的inbound附件信息
outboundProperties	message的outbound头信息
outboundAttachments	message的outbound附件信息

5. MEL 的 Variable

不同于第 4 点提到的上下文对象，MEL 中还可以使用变量，使用变量并不要求在表达式中使用上下文对象。变量是顶层的标识符。MEL 中常见的变量如下：

- flowVars - flowVars 的有效范围是在一个 Flow 中，定义 flowVars 之后，后续的 Message Processor 都可以使用。
- sessionVars - 在跨 Flow 通信时，可以使用 sessionVars 来传递变量。需要注意的是，sessionVars 并不总是有效的，其实取决于 Inboud Endpoint 的类型。后续再出专题介绍 flowVars 和 sessionVars 等之间的区别

```
#[flowVars.foo = sessionVars.bar]
```

上述的表达式的意思是，将 session 变量赋值给 flow 变量。

6. MEL 访问属性

1. 点语法。适用对象通常是Java Pojo。MEL中可以使用点语法来访问相关的对象属性，同样对象属性的属性也是可以用点号来访问的。

```
#[message.payload.item.name]
```

2. Null安全性访问。Java编程中经常遇到NullPointerException错误，也就是说对空对象进行访问操作会报错。而在MEL表达式，可以通过点语法.?来避免出错。如下示例，即使item为null，该表达式仍然不会报错，它会返回null值。

```
#[message.payload.?item.name]
```

3. 属性名称的转义。如果属性名称有特殊字符，那么使用点语法会遇到问题，这个时候可以单引号进行转义。如下示例，http.query.params是一个整体。我们访问这个属性名，必须使用单引号进行转义。

```
#[message.inboundProperties.'http.query.params'.customerNo]
```

4. 中括号语法。如果对象是数组，或者Map，那么可以使用中括号进行访问

```
#[payload[5]]
```

```
#[payload['userName']]
```

7. MEL 操作符

常用的操作符如下，和普通的开发语言类似。还有更多的操作符可以查阅官方手册。

算术运算符 + - / * %
比较运算符 == != > < >= <=
逻辑运算符 && ||

8. 控制消息的流向-数据路由

1. 使用场景

我们来看一个常见的应用场景。我们的集成应用可以接收客户端提交订单的请求，有多种不同类型的客户端。手机客户端是一个 App，我们在 App 上使用的消息格式通常是轻量级的 JSON 格式。而 PC 客户端是一个 Windows 时代的产物，它仍然使用 XML 格式。我们的集成应用在接收到订单请求后，会根据订单的类别分发到不同的仓库，订单类别存储在订单内容的 OrderType 字段中。

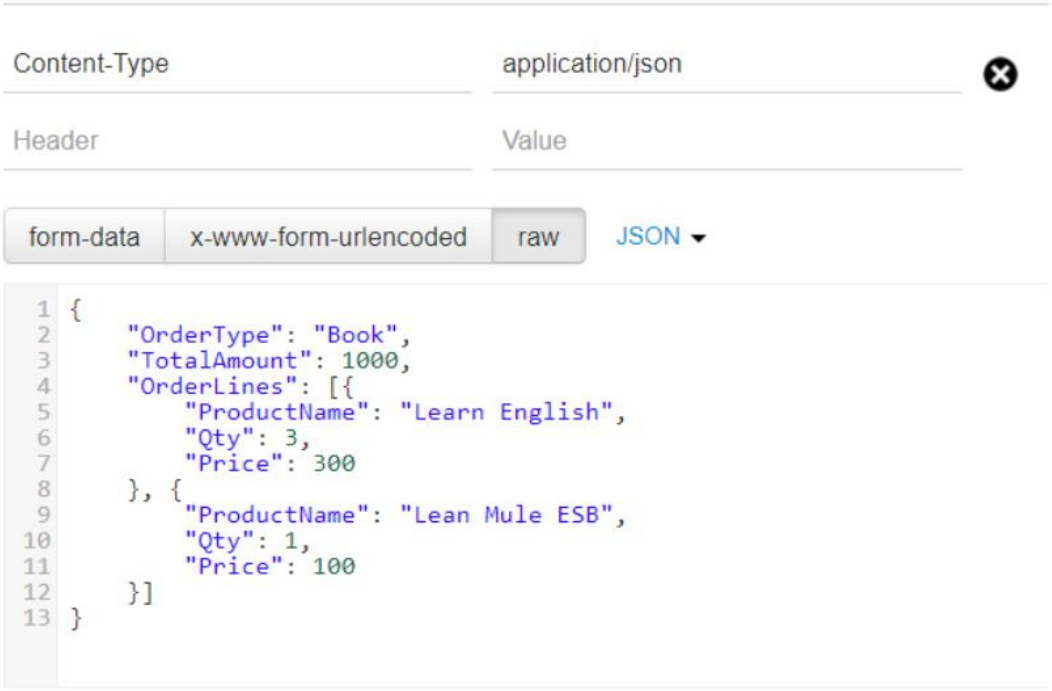
2. 基于消息头的路由

我们首先要区分订单的格式，因为 JSON 和 XML 的处理逻辑是完全不同的。在 Mule 中我们可以使用 MEL 的 JSON Path 来解决 JSON 文件的解析，XML Path 来解决 XML 文件的解析。MEL 的详细用法可以参考上一篇文章。订单的格式通常会放在消息头上传递，而 Mule Message 中对应消息头的部分就是 Message Inboud Properties。

使用 **JSON** 提交订单的消息

```
{
    "OrderType": "Book",
    "TotalAmount": 1000,
    "OrderLines": [{
        "ProductName": "Learn English",
        "Qty": 3,
        "Price": 300
    }, {
        "ProductName": "Lean Mule ESB",
        "Qty": 1,
        "Price": 100
    }]
}
```

使用 Postman 提交 JSON，注意 Content-Type 设置为 application/json。

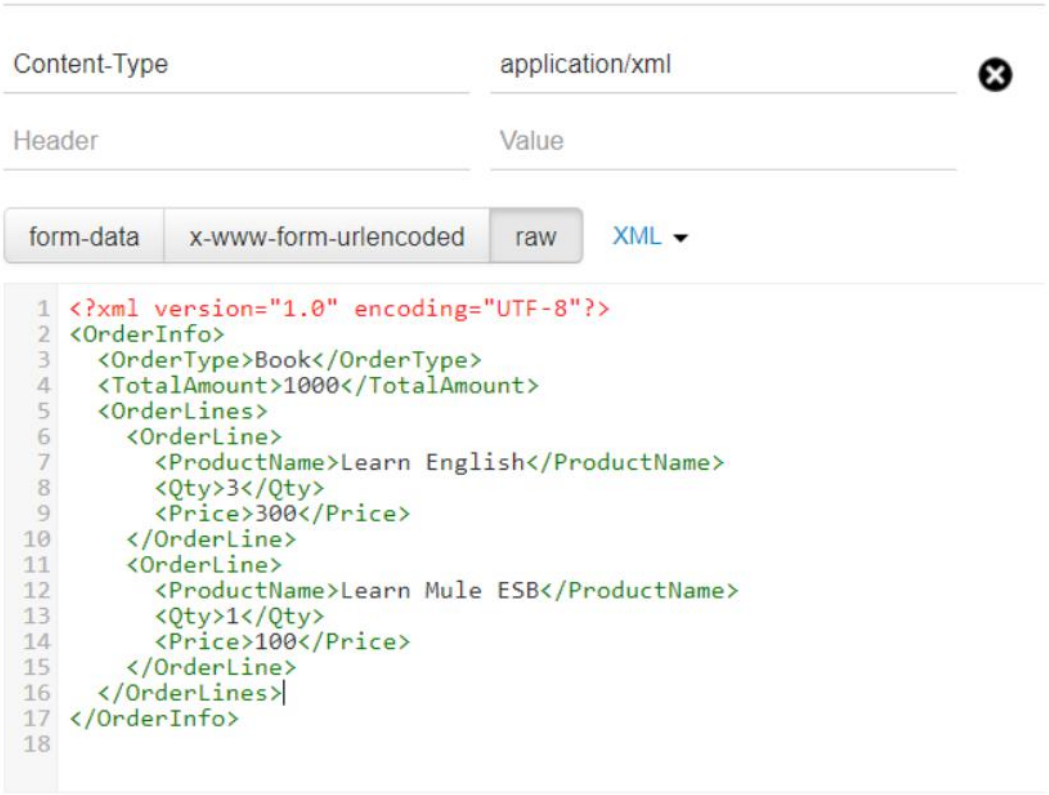


使用 **XML** 提交订单的消息

```
<?xml version="1.0" encoding="UTF-8"?>
<OrderInfo>
  <OrderType>Book</OrderType>
  <TotalAmount>1000</TotalAmount>
  <OrderLines>
    <OrderLine>
      <ProductName>Learn English</ProductName>
      <Qty>3</Qty>
      <Price>300</Price>
    </OrderLine>
    <OrderLine>
      <ProductName>Learn Mule ESB</ProductName>
      <Qty>1</Qty>
```

```
        <Price>100</Price>
      </OrderLine>
   </OrderLines>
</OrderInfo>
```

使用 Postman 提交 XML，注意 Content-Type 设置为 application/xml。



使用 **Choice** 组件判断

Mule ESB 最基本的路由组件就是 Choice 组件，组件类似于我们在编程语言中的 if/esle if/else。订单的格式是存放在 Message Inbound Properties 中的。这里就使用 Choice 组件来判断订单的格式。

Mule XML Config:

```
    <choice doc:name="Choice">
    <when expression="#[sessionVars.goodsType == '3']">
      <custom-transformer class="com.mule.mcp.transformer.SupplierConsumablesTransformer" doc:name="supplierConsumablesTransformer"/>
            <http:request config-ref="mcpBaseHttp" path="/consumables/supplierconsumables/save" method="POST" doc:name="HTTP">
                <http:request-builder>
                    <http:header headerName="supplierCode" value="#[message.inboundProperties.supplierCode]"/>
                     <http:header headerName="sign" value="#[message.inboundProperties.sign]"/>
                </http:request-builder>
            </http:request>
             <custom-transformer class="com.mule.mcp.transformer.JsonToXmlTransformer" doc:name="Java"/>
             <logger level="INFO" doc:name="Logger"/>
    </when>
    <when expression="#[sessionVars.goodsType == '1']" >
      <custom-transformer class="com.mule.mcp.transformer.SupplierDrugsTransformer" doc:name="supplierDrugsTransformer"/>
            <http:request config-ref="mcpBaseHttp" path="/drugs/supplierdrugs/save" method="POST" doc:name="HTTP">
                <http:request-builder>
                    <http:header headerName="supplierCode" value="#[message.inboundProperties.supplierCode]"/>
                    <http:header headerName="sign" value="#[message.inboundProperties.sign]"/>

                </http:request-builder>
            </http:request>
             <custom-transformer class="com.mule.mcp.transformer.JsonToXmlTransformer" doc:name="Java"/>
            <logger level="INFO" doc:name="Logger"/>
    </when>
      <otherwise>
          <custom-transformer class="com.mule.mcp.transformer.SupplierReagentTransformer" doc:name="supplierReagentTransformer"/>
          <http:request config-ref="mcpBaseHttp" path="/reagent/supplierreagents/save" method="POST" doc:name="HTTP">
             <http:request-builder>
                    <http:header headerName="supplierCode" value="#[message.inboundProperties.supplierCode]"/>
                    <http:header headerName="sign" value="#[message.inboundProperties.sign]"/>

             </http:request-builder>
          </http:request>
            <custom-transformer class="com.mule.mcp.transformer.JsonToXmlTransformer" doc:name="Java"/>
          <logger level="INFO" doc:name="Logger"/>
      </otherwise>
  </choice>
```

3.其他控制流向的组件

除了使用 Choice 组件改变 Flow 执行路径以外，还要很多控制流向的组件。比如 Round-Robin，Resequencer，Scatter-Gather，Splitter，Aggregator 等组件。这些组件的用法各不相同，这里主要讲一下 Round-Robin 的用法。Round-Robin 是一种轮询调度算法，在负载均衡的策略里面经常见到 Round-Robin，轮询调度算法的原理是每一次把来自用户的请求轮流分配给内部中的服务器，从 1 开始，直到 N(内部服务器个数)，然后重新开始循环。

9.开发 webservice

Apache CXF Web Service wsdl 代理

使用 Apache CXF 搭建的 Web Service 只有一个 `SupplierGoodsService`，主要的类文件和配置文件如下：

```
package com.mule.mcp.service;

import javax.jws.WebService;

@WebService
public interface SupplierGoodsService {

    String uploadSupplierGoods(String text);

}
```

```
package com.mule.mcp.service.impl;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import com.mule.mcp.service.SupplierGoodsService;
@WebService(endpointInterface = "com.mule.mcp.service.SupplierGoodsService",serviceName = "supplierGoodsService")

public class SupplierGoodsServiceImpl  implements SupplierGoodsService {

    @WebMethod(action="uploadSupplierGoods")
    public String uploadSupplierGoods(@WebParam(name="param") String param) {
        return param;

   }
}
```

```
<cxf:jaxws-service
serviceClass="com.mule.mcp.service.SupplierGoodsService"
doc:name="CXF">
        </cxf:jaxws-service>
        <component
```

```
class="com.mule.mcp.service.impl.SupplierGoodsServiceImpl"
doc:name="Java"/>
```

Web Service 配置在一个 Web Application 中，在 AnypointStudio 的 Tomcat Server 启动后的 Web Service wsdl 文件访问地址是

http://localhost:8888/mcp-server/uploadsuppliergoods?wsdl