# Flows、模式或者服务的选择

## Flows、模式或者服务的选择

因为Mule 3产生时就强调易用性、简洁性和灵活性，所以一开始就提供了多种选择。

### 选择配置模式和Flows

模式和Flows的配置参见Configuration Patterns和Flows。相对于单独使用服务，它们能让你走得更快更远。如果你是新手，第一步请阅读Configuration Patterns。

Mule 3包含很多配置模式，它们被设置来使用最少的XML解决常见的集成问题。这些模式类似在Mule中广泛支持的企业集成模式（Enterprise Integration Patterns），但是涉及更大的用户，例如创建一个webservice代理。我们看看一下模式的列表。

如果你的项目需要一些更复杂的操作，你可以考虑Mule 3中的Flows。

更多关于模式的信息，请参阅Configuration Patterns。

* Asynchronous Reply Router Configuration Reference
* Catch-all Strategy Configuration Reference
* Component Configuration Reference
* Endpoint Configuration Reference
* Exception Strategy Configuration Reference
* Filters Configuration Reference
* Global Settings Configuration Reference
* Inbound Router Configuration Reference
* Model Configuration Reference
* Notifications Configuration Reference
* Outbound Router Configuration Reference
* Properties Configuration Reference
* Security Manager Configuration Reference
* Service Configuration Reference
* Transactions Configuration Reference
* BPM Configuration Reference

### 选择服务

Mule基于服务的架构非常适合于用其作为服务容器、通过一个或多个接入点来暴露服务的很多场景。如果你熟练使用Mule之前的版本，那么只需要少许的改动，你可以继续基于相同的方式使用它。

## 使用Flows编排服务

### 使用Flows编排服务

#### 介绍

Flows是一种简单灵活的机制，它能够使用Mule ESB负责的消息流能力来编排服务。通过Flow，可以使用Mule提供的基础的Message Sources和Message Processors来建立自动集成处理和构建复杂的集成解决方案。正因为Mule的灵活性，它可以很容易地创建非常适合你需求的解决方案。Flow是Mule 3新增的功能。

#### 何时使用Flows

Mule中的Flow是一种多功能的、极其有力的集成机制。



相对于使用服务这种定义了一个具有明确的输入和输出阶段、有限灵活性的组件，Flow不定义任何东西，完全的形式自由。这意味着如果你的解决方案需要多个步骤，你不需要使用通过vm transport或chaining router黏合的多个服务。相反的，你只需要在同一个Flow完成所有配置。

Flow在很多情况下非常有用，包括：

* Simple integration tasks
* Scheduled data processing
* Connecting cloud and on-premise applications
* Event processing where multiple services need to be composed

在你开始创建Flows之前，请先阅读configuration patterns。对常见用例来说，它们可以减少大量的配置工作。

在某些场景下你仍然可以选择使用服务。例如你想要做的是通过服务组件暴露单个服务。如果你的需求比简单服务支持(Simple Service Support)要更复杂，Mule服务依然是一个很好的选择。Mule服务也可以集成到Flow中。但是Flow并不是Mule中的全部功能。特别地，服务非常适用于如下的场景：

* 任意地方（除endpoints、routers之外）的事务配置
* 请求-响应

Mule未来的版本将为Flow添加这些功能。

#### Flows分析

Flow本质上是一个Message Processors链。可以认为每个Message Processor是一个乐高积木块，Flow就是用它们组装起来的。Flow还有一个Message Source，它是Message Processor链处理的消息来源。



#### Flows配置

Flow在XML中使用flow元素来表示。每个flow有一个name属性，一个message source属性（除非它是一个内部的flow），一个或多个message processor，可选的exception strategy。

|  |
| --- |
| **基础结构** |
| <flow name="">  - 0..1 MessageSource  - 1..n MessageProcessor(s)  - 0..1 ExceptionStrategy  </flow> |

Flows看起来简单，实际上功能非常强大。尤其是需要在Mule中组合表达式的时候，它们允许基于消息内容做一些非常复杂的处理。如下几种元素支持表达式：

* Transformers
* Filters
* Routers
* Message Enrichers

例子：

|  |
| --- |
| **一个简单的图书订单处理Flow** |
| <flow>  <file:inbound-endpoint path="/myDirectory">  <file:filename-filter name="\*.xml"/>  </file:inbound-endpoint>  <xml:xslt-transformer xsl-file="bookOrderTransformation.xsl"/>  <splitter expression="xpath://order"/>  <!-- The following message processors will be invoked for each order in the xml file -->  <expression-filter expression="xpath://order[@type='book']"/>  <component class="org.my.BookOrderProcessor"/>  <smtp:outbound-endpoint subject="Order Confirmation" address=""/>  <jdbc:outbound-endpoint />  <default-exception-strategy>  <jms:outbound-endpoint queue="failedOrders"/>  </default-exception-strategy>  </flow> |

#### Flows行为

当一个消息到达Message Source时，Flow就启动了，其Message Processor链中的Message Processor按照配置的顺序被依次调用。某些Message Processor可以包含Message Processor子元素，此时它们将在返回到下一步处理之前被调用。

上述描述的Flow被称为单向的(one-way)。如果Flow是请求-响应模式的(request-response)，则有一个接入点(inbound endpoint)被定义为请求-响应交换模式，Flow执行的结果返回给接入点，最后再返回给调用者。如果Flow中没有<response>块，并且也没有配置Message Processor来处理响应，则Flow简单的使用最后一个Message Processor的结果来作为响应。如果使用了<response>块，则此元素中配置的每一个Message Processor都可以用来处理响应消息。一些Message Processor（例如CXF）默认配置下就可以用来处理响应消息。

**注意**：当Flow中的最后一个元素是one-way <outbound-endpoint>，它不返回结果，因此返回的消息负载为NullPayload。如果一个one-way <outbound-endpoint>后面是另外一个processor，则它传递给此processor的不是NullPayload，而是它接收到的消息。



#### 内部Flows

内部Flow不能通过Mule Endpoint从JVM外部访问，因为它没有定义Message Source。

因此，内部Flows只能被运行在同一个Mule实例的其他Flow引用。可以在XML中配置<flow-ref>元素来包含其他Flow。

内部Flow不同于“Processor Chain”的使用，它包含自己的上下文和异常策略（Exception Strategy）。因此当一个Processor Chain被引用，它将在引用它的Flow或服务的上下文中执行。

|  |
| --- |
| **内部Flow例子** |
| <flow name="privateFlow">  <append-string-transformer message="b"/>  </flow>  <flow name="publicFlow">  <http:inbound-endpoint address="http://localhost:8080"/>  <append-string-transformer message="a"/>  <flow-ref name="privateFlow"/>  <append-string-transformer message="c"/>  </flow> |

#### 进一步阅读

可以通过阅读下列博客文章了解Mule 3中添加了Flow的原因：

Mule 3 架构，第一部分：回到基础( Mule 3 Architecture, Part 1: Back to Basics)

Mule 3架构，第二部分：Message Processor介绍( Mule 3 Architecture, Part 2: Introducing the Message Processor)

## 使用Mule 服务

### 使用Mule服务

#### 何时使用服务

除非你特别愿意使用服务，否则应该考虑使用配置模式或Flow。

服务比配置模式更灵活，但是不如Flow功能强，是Mule 2中配置集成的方式。

服务提供一个固定框架来建立集成机制，基于接入路由和接入点、单个服务组件、接出路由和接出点等概念。

为了完成复杂的路由或处理链，经常需要多个服务通过VM queues绑到一起。相比较而言，单个Flow就可以完成这样的复杂集成。

#### 服务组件

服务组件可以是类，web服务，或者一个包含了你想要接入Mule ESB框架的业务逻辑的应用程序。例如，一个服务组件可以从客户数据库添加信息到发货单，另一个服务组件可以是一个用来处理发货单的订单实现应用程序。可以使用任何已经存在的应用程序来实现一个服务组件，当然也可以完全新建一个。

服务组件可以不包含任意跟Mule规范相关的代码，相反地，可以用Mule规范相关的配置来把一个服务组件包装成为一个服务。服务配置指向服务组件，同时路由、过滤器和转换器被用来负载出入服务组件的消息。也可以为服务定义一个接入点，用于接收消息的；以及一个接出点，把消息传递出去。



查看一个建立服务的简单例子，请点击此处。

一个服务组件可以是POJO，spring bean或者脚本。典型地，你可以配置你自己的服务组件，也可以使用Mule内置的标准组件。更多信息，请参考配置组件。

#### 服务配置

大部分的配置信息都发生在服务级别。服务可以使用全局定义的接入点、转换器或过滤器，也可以配置成内联方式。更多信息，请参考配置服务。

#### 服务行为

当服务从接入点接收到一个消息，服务模型（默认是SEDA）决定了服务的线程和队列行为。同时消息模式定义了使用的接入接出消息交互模式。

#### 高级配置

你可以进一步的配置服务的安全（在接入点上）、事务和出错处理。

### 服务消息样式

#### Mule消息样式

##### 概述

Mule ESB可以异步的发送消息（不同处理步骤在不同的线程中完成），也可以同步的处理消息（组件接收到消息以后，消息的整个生命周期和请求响应都在同一个线程中完成）。你可以在连接器、接入点或者transport上设置request-response属性。

默认情况下，Mule ESB使用SEDA这种异步、单向、分阶段的队列。一个线程负责消息接入、一个线程用于在服务组件中处理消息、一个线程用于消息接出。可以配置接入消息使用单独的线程、其他步骤使用一个线程，或者所有的步骤公用同一个线程。

Mule ESB也提供请求-响应的消息样式。此时没有接出路由，消息被发送回到接入点，最后返回响应给请求方。

你也可以通过Mule使用混合单向和请求-响应的消息样式，也可以在单个的服务组件上使用混合样式。例如，一个服务组件可以有多个接出路由，根据不同的过滤条件路由到不同的接入点，但是你想在某些情况下需要响应，某些情况下不需要响应。

本节的剩余部分详细的描述几种不同的消息样式，如何配置它们，以及相关的消息交换模式(MEPs)。更多MEP和Mule的消息，请参考MEPs。

##### one-way



|  |  |
| --- | --- |
| 描述 | 接收一个消息放入SEDA队列。调用方线程立即返回，消息将被SEDA线程池处理。这个调用不会返回任何结果。 |
| 出错处理 | Mule Server将处理过程中出现的错误。它使用一个错误接入点来路由错误，发起调用的客户端可以通过单独的线程监听出错队列，或者可以直接创建一个出错处理客户端。 |
| Mule配置 | Mule服务必须包含一个接入点。 |
| 对应的MEPs | In-only |
| 讨论点 | 我们没有办法在web service外部支持Robust In-only MEP（在Mule中可以使用请求/响应）和在web service契约中定义此MEP。 |

配置示例：

<mule xmlns="http://www.mulesoft.org/schema/mule/core"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:mule="http://www.mulesoft.org/schema/mule/core"

xmlns:jms="http://www.mulesoft.org/schema/mule/jms"

xmlns:test="http://www.mulesoft.org/schema/mule/test"

xmlns:spring="http://www.springframework.org/schema/beans"

xsi:schemaLocation="

http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd

http://www.mulesoft.org/schema/mule/test

http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd

http://www.mulesoft.org/schema/mule/jms

http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<model name="Asynchronous\_Message\_Pattern">

<service name="AsynchronousService">

<inbound>

<jms:inbound-endpoint queue="test.in" exchange-pattern="one-way"/>

</inbound>

<test:component/>

</service>

</model>

</mule>

##### request-response



|  |  |
| --- | --- |
| 描述 | 接收一个消息，并由组件(component)返回一个消息。如果组件调用返回为null，则返回一个NullPayload的Mule消息。如果调用的方法为void，则返回请求消息本身。 |
| Mule配置 | Mule服务必需配置一个同步的接入点，没有接出点。可以使用synchronous=”true”来设置一个接入点为同步的。HTTP/S、SSL、TCP和Servlet接入点默认即为同步，不需要设置。 |
| 出错处理 | 总是会返回一个响应消息。客户端可以通过检查MuleMessage. getExceptionPayload()来获取服务器端出错的所有信息。起源于客户端调用的异常将被直接抛出。 |
| 对应的MEPs | In-Out，In-Optional-Out |
| 讨论点 | 如果调用没有返回结果，In-Optional-Out返回请求消息本身。 |

配置示例：

<mule xmlns="http://www.mulesoft.org/schema/mule/core"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:mule="http://www.mulesoft.org/schema/mule/core"

xmlns:http="http://www.mulesoft.org/schema/mule/http"

xmlns:test="http://www.mulesoft.org/schema/mule/test"

xmlns:spring="http://www.springframework.org/schema/beans"

xsi:schemaLocation="

http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd

http://www.mulesoft.org/schema/mule/test

http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd

http://www.mulesoft.org/schema/mule/http

http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<model name="Request-Response\_Message\_Pattern">

<service name="SynchronousService">

<inbound>

<http:inbound-endpoint host="localhost" port="8080" path="/mule/services"

exchange-pattern="request-response"/>

</inbound>

<test:component/>

</service>

</model>

</mule>

##### 同步

注意：在Mule 3.0版本，接入端的“synchronous”标识被“exchange-pattern”属性代替，此属性有如下可选的值：

* one-way
* request-response

当如下条件之一为true时，消息将被同步地处理：

* 接入点是request-response模式的
* 事务处理激活状态
* 消息的“MULE\_FORCE\_SYNC”属性被设置为TRUE



|  |  |
| --- | --- |
| 描述 | 接收一个消息，并由组件处理，然后发送到另外的接入点。请求和处理发生在同一个线程。Mule阻塞住接出点，等待远程应用程序（如果可用）响应，直到达到响应超时的上限值（responseTimeout threshold）。如果没有接收到响应，它将返回null。当使用事务时，必须在接入端使用同步请求。同步请求永远返回结果，尽管它有一个接出端。 |
| Mule配置 | Mule服务必需配置一个请求-响应接入点，和一个接出点。可以使用message-exchange=”request-response”来设置一个接入点为请求-响应模式的。HTTP/S、SSL、TCP和Servlet接入点默认即为请求-响应模式的，不需要设置。 |
| 出错处理 | 总是会返回一个响应消息。客户端可以通过检查MuleMessage. getExceptionPayload()来获取服务器端出错的所有信息。起源于客户端调用的异常将被直接抛出。 |
| 对应的MEPs | In-Only，In-Out，In-Optional-Out |
| 讨论点 | Mule永远从组件返回结果给调用者，也会通过接出端把它发送出去。 |

配置示例：

<mule xmlns="http://www.mulesoft.org/schema/mule/core"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:mule="http://www.mulesoft.org/schema/mule/core"

xmlns:jms="http://www.mulesoft.org/schema/mule/jms"

xmlns:test="http://www.mulesoft.org/schema/mule/test"

xmlns:spring="http://www.springframework.org/schema/beans"

xsi:schemaLocation="

http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd

http://www.mulesoft.org/schema/mule/test

http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd

http://www.mulesoft.org/schema/mule/jms

http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<model name="Synchronous\_Message\_Pattern">

<service name="SynchronousService">

<inbound>

<jms:inbound-endpoint queue="test.in" exchange-pattern="request-response"/>

</inbound>

<test:component/>

<outbound>

<pass-through-router>

<jms:outbound-endpoint queue="test.out" exchange-pattern="one-way"/>

</pass-through-router>

</outbound>

</service>

</model>

</mule>

##### 异步request-response



|  |  |
| --- | --- |
| 描述 | 此模式允许后端处理，分别去调用其他服务，基于多个服务调用的结果来返回一个结果。通常使用Async Reply Router监听一个Reply To接入点来获取多个结果。 |
| Mule配置 | 在接出路由上设置reply-to地址，设置<async-reply>元素来监听响应端点。如果调用者需要一个响应，可以使用message-exchange=”request-response”设置一个同步接入端。 |
| 出错处理 | 总是会返回一个响应消息。客户端可以通过检查MuleMessage. getExceptionPayload()来获取服务器端出错的所有信息。起源于客户端调用的异常将被直接抛出。 |
| 对应的MEPs | In-Out，In-Optional-Out |
| 讨论点 | 无 |

配置示例：

<mule xmlns="http://www.mulesoft.org/schema/mule/core"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:mule="http://www.mulesoft.org/schema/mule/core"

xmlns:http="http://www.mulesoft.org/schema/mule/http"

xmlns:jms="http://www.mulesoft.org/schema/mule/jms"

xmlns:test="http://www.mulesoft.org/schema/mule/test"

xmlns:scripting="http://www.mulesoft.org/schema/mule/scripting"

xmlns:spring="http://www.springframework.org/schema/beans"

xsi:schemaLocation="

http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd

http://www.mulesoft.org/schema/mule/test

http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd

http://www.mulesoft.org/schema/mule/http

http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd

http://www.mulesoft.org/schema/mule/jms

http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd

http://www.mulesoft.org/schema/mule/scripting

http://www.mulesoft.org/schema/mule/scripting/3.0/mule-scripting.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<model name="Async\_Request-Response\_Message\_Pattern">

<service name="AsyncRequestResponseService">

<inbound>

<http:inbound-endpoint host="localhost" port="8080" path="/mule/services"

exchange-pattern="request-response"/>

</inbound>

<test:component/>

<outbound>

<multicasting-router>

<jms:outbound-endpoint queue="service1" exchange-pattern="one-way"/>

<jms:outbound-endpoint queue="service2" exchange-pattern="one-way"/>

<reply-to address="jms://reply.queue"/>

</multicasting-router>

</outbound>

<async-reply timeout="5000">

<jms:inbound-endpoint queue="reply.queue" exchange-pattern="one-way"/>

<collection-async-reply-router/>

</async-reply>

</service>

</model>

</mule>

### 配置服务

#### 配置服务

可以在<model>元素中使用<service>元素来配置服务。每个<service>元素代表一个Mule ESB 服务，提供一个唯一的名称标识服务，和一个可选的初始状态来决定此服务及其接入点是否在Mule启动的时候一起启动(支持三个值：started/stoped/paused)。

<mule>

<model>

<service name="GreeterUMO">

...

</service>

<service name="GreeterUMO2" initialState="stopped" >

...

</service>

</model>

</mule>

每个服务都可以配置下列可选的元素：

* <description> 描述服务
* <inbound> 配置接入点、接入路由和转换器
* <component> 配置服务组件
* <outbound> 配置接出点、接出路由和转换器
* <async-reply> 配置异步响应路由，用于异步请求-响应消息
* <exception-strategy> 配置服务的异常策略

如果你使用了多个元素，注意你必须按照上述的顺序配置它们。关于<service>元素和属性的更多细节信息，请参考 Service Configuration Reference。

下面是服务的各种元素配置示例：

<service name="GreeterUMO">

<description>Adds some text to the string before passing it on</description>

<inbound>

<stdio:inbound-endpoint system="IN">

<transformer ref="StdinToNameString"/>

</stdio:inbound-endpoint>

</inbound>

<component class="org.mule.example.hello.Greeter" />

<outbound>

<filtering-router>

<vm:outbound-endpoint path="chitchatter" />

<payload-type-filter expectedType="org.mule.example.hello.NameString" />

</filtering-router>

</outbound>

<default-service-exception-strategy>

<vm:outbound-endpoint path="systemErrorHandler" />

</default-service-exception-strategy>

</service>

下面几节介绍这个元素。

##### Inbound

这个元素用雷配置接入端和接入路由。接入端点用于接收传入的消息，接入路由用来决定这些消息讲被如何路由。接入点和路由独立配置于<inbound>元素内。

###### 接入点

接入点被用来接收输入的消息。接入点仅仅是一个指令集，用来表示接收消息来自于哪个传输端和地址，同时接收消息时也可以使用任何转换器、过滤器或者安全策略。你可以创建多个接入点，每个接收来自于不同传输端的消息。更多消息请参阅Configuring Endpoints和Transports Reference。

###### 接入路由

<inbound>元素中可以配置接入路由。接入路由控制和操作服务接收的消息，然后传递它们给服务组件。典型地，接入路由被用来过滤输入的消息，聚合一组消息，或者重排消息。接入路由也被用来为一个服务注册多个接入端。多个接入路由有了可以一起组成一个链，这时每个路由都必须被匹配，消息才能传递到服务组件。你也可以配置一个catch-all strategy，此时如果没有路由接受当前消息，它将被调用。

更多信息，请参阅Inbound Routers。

##### Component

<component>元素被用来配置服务组件，它将在接入路由处理接入消息之后被调用。如果没有配置组件，服务则扮演一个桥梁的角色，简单地把消息传递给接出路由。

有几个标准的组件，例如<log-component>，它记录组件调用日志、并将接收的消息作为字符串输出；<echo-component>扩展了<log-component>，记录和显示输入的消息。一般来说，你可以通过POJO对象来创建自己的组件，只需要把它配置成<component>元素。

更多关于组件类型及其配置的信息，请参阅Configuring Components。你也可以在你的Mule模块中实现新的组件类型，并在你的配置中使用它们。在Mule 2.0中，实现和使用一个新的非Java的组件类型是非常容易的。

##### Outbound

<outbound>元素用来配置接出路由和其他的接出端。因为接出路由被用来决定哪个端点被用来在组件处理完消息之后分发它们，接出端配置于接出路由上，而不是直接在<outbound>元素上。接出路由允许开发者为给定消息定义多个路由规则。你也可以配置一个catch-all strategy，此时如果没有路由接受当前消息，它将被调用。更多信息请参阅Outbound Routers。

##### Asynchronous Reply Router

<async-reply>元素用于在异步request-response场景下接收响应信息，你必须在当前服务通过其接入点返回之前，合并来自远程端点的多个响应。这个方式的典型例子是，一个请求过来以后，多个任务并发地执行。在返回响应给调用方之前，每个任务都需要执行完毕、拿到结果。这里#超链接#有一个异步request-response的说明。更多信息，请参阅Asynchronous Relay Router和Configuration Endpoints。

##### Exception Strategy

异常策略被用来捕获消息处理过程中发送的错误。可以在服务上配置异常策略。如果没有配置异常策略，讲使用DefaultServiceExceptionStrategy。

更多信息，请参阅Error Handling。

##### Service Bridge

在Mule 2.X服务组件配置是可选的。默认情况下隐式的使用PassThroughComponent。此组件自动地桥接接入消息到接出阶段，然后简单地传递消息到接出路由。如果你想从一个传输端传递消息到另外的传输端，这种方式非常有用。

从Mule 2.0开始，你不再需要显示的配置BridgeComponent。



下面的示例展示如何读取一个文件的内容并发送到一个JMS主题中。

<service name="FileToJmsBridge">

<inbound>

<file:inbound-endpoint path="/data/in">

<file:filename-wildcard-filter pattern="\*.txt"/>

</inbound-endpoint>

</inbound>

<!-- No need to configure a component here -->

<outbound>

<pass-through-router">

<jms:outbound-endpoint topic="receivedFiles"/>

</pass-through-router>

</outbound>

</service>

如果你想发送一个响应给接入端，在接出端使用chaining-router代替pass-through-router。同时接入端必须是同步的。

<service name="HttpProxyService">

<inbound>

<inbound-endpoint address="http://localhost:8888" synchronous="true"/>

</inbound>

<outbound>

<chaining-router>

<outbound-endpoint address="http://www.webservicex.net#[header:http.request]" synchronous="true"/>

</chaining-router>

</outbound>

</service>

##### Service Model

默认情况下，Mule使用分段式事件驱动架构SEDA。SEDA是一个架构模型，应用程序由显式队列连接的、分阶式的事件驱动网络构成。此架构允许服务被良好的加载，在请求超出服务能力时防止系统过载。因此，SEDA提供了一个非常高效的、基于事件的队列模型，来使性能和吞吐量最大化。

##### Service消息样式

消息样式决定了用于接入端和接出端的消息交换模式。消息交换模式允许端点配置为同步request-response、异步in-only或其他模式。

消息样式配置在端点上，同一个服务上允许配置多个消息样式。更多信息请参阅Service Messaging Style。