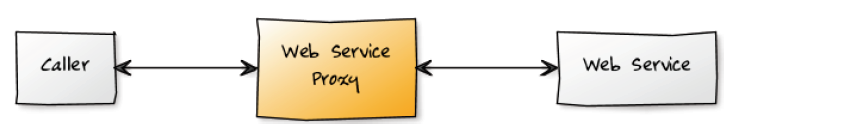
**Web Service Proxy Pattern**

**Web Service Proxy Pattern**

Proxying web services is a very common practice used for different reasons like security or auditing. This pattern allows a short and easy configuration of such a proxy.

Web Service代理模式是一种很普遍的应用，基于安全或审计的考虑。该模式可以通过简单配置完成。



***Core Features***

***核心特性***

A web service proxy acts as an intermediate between a caller application and the target web service. This gives the proxy a chance to transparently introduce new behaviors in the calling sequence. For example, it can:

Web service代理作为应用程序和目标web service之间的媒介。在此情况下，代理在被调用的过程中可以增加一些新的行为。例如，它可以：

add or remove HTTP headers,

添加或删除HTTP标头

transform the SOAP envelope (body or header) to add or remove specific entries,

转换SOAP信封（身体或头）来添加或删除特定条目

rewrite remote WSDLs so they appear to bind to services inside a corporate firewall,

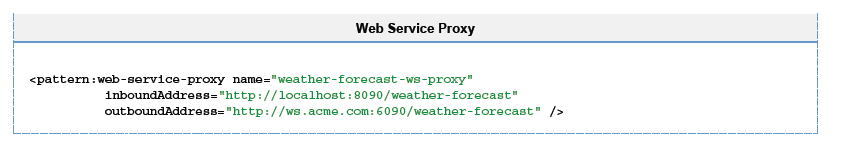
改写,使他们出现在企业防火墙内的服务绑定到远程的WSDL

introduce custom error handling.

引入自定义错误处理

Let's take a look at Web Service Proxy in action:

让我们看一个web service 代理的例子：



With this configuration in place, all calls to the local weather forecaster proxy will be redirected to the remote one.

The Web Service Proxy is provided by the ws module, which must be present on the classpath to be usable. Its namespace is

通过这个配置文件，所有调用本地天气预报代理将被重定向到一个远程服务。  
Web服务代理提供的WS模块，为了使用它必须将它存在类路径中。它的命名空间是

WS模块提供web services代理模式,所以此模快必须加入到相应的classpath。

http://www.mulesoft.org/schema/mule/ws and its schema location is http://www.mulesoft.org/schema/mule/ws/3.0/mule-ws.xsd.

The true value add comes from the automatic address rewriting that will be performed by the proxy: calling

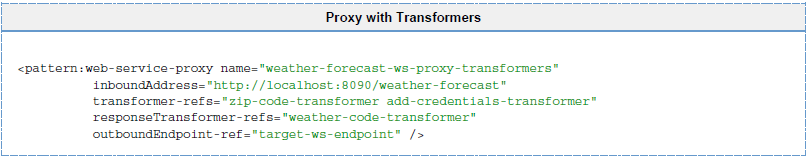
http://localhost:8090/weather-forecast?wsdl will return the remote WSDL where the port addresses will have been automatically rewritten base on the URL of the request hitting the proxy. That way, if your Mule instance is accessed behind a load balancer or any kind of network indirection, the generated WSDL will point the caller to port addresses that respect your particular network topology.

代理自动替换成实际值, 调用http://localhost:8090/weather-forecast?wsd将返回远程WSDL的端口地址,代理将基于请求的URL重写该地址端口。

这样的话，如果你的Mule实例访问背后负载平衡器或任何形式的网络重写向之后，生成的WSDL将指向调用者的端口地址，将会指导调用者访问实际的网络拓扑。

As said above, the proxy can perform changes on the SOAP invocation by the use of transformers. This is demonstrated hereafter:

正如上面说的，通过对transformers的使用，代理可以对soap进行转换。这是展示：

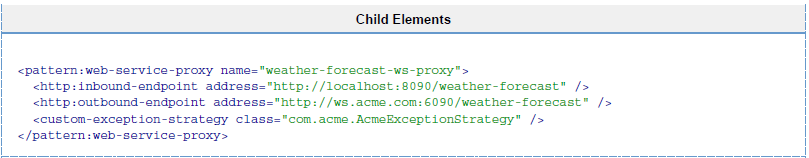


Notice how transformers are introduced by using references to globally declared ones. This technique is also applicable to global endpoints, as you can see with the above reference to target-ws-endpoint.

注意上图transformers如何在全局中被定义。此技术也可以应用到全局的endpoints，上图的target-ws-endpoint应用了此技术。

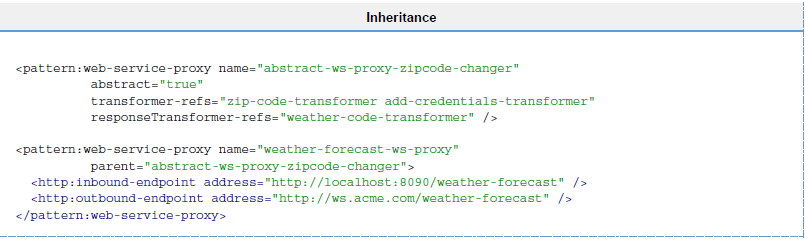
The Web Service Proxy element supports child elements. The following shows a configuration variant where endpoints are declared internally and an exception strategy has been added in:

Web service 代理节点支持子节点。下面显示了一个配置的变化，在内部中定义了endpoints字节点并增加了异常处理逻辑：



Finally, the Web Service Proxy also supports inheritance, which allows sharing common configuration attributes across several concrete instantiations of the proxy. Check out the following to see how inheritance works:

最后，Web service代理也支持继承，在不同的实例代理中共享常见的配置属性。看看下面是如何实现继承：



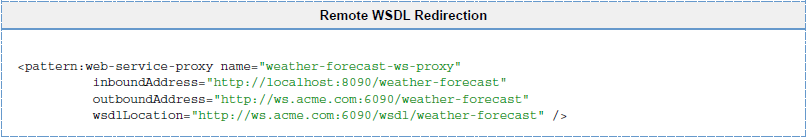
The proxy offers a few extra options as far as WSDL handling is concerned. Let's look at them.

代理提供WSDL处理需要的一些额外的选项。让我们来看看这些选项。

***WSDL Redirection***

In some cases, the remote web service doesn't follow the common practice of exposing its WSDL on the same address as the service with a "?wsdl" appended at the end. In that case, it is required to point the Web Service Proxy to the exact location of the remote WSDL, as illustrated there:

在某些情况下，远程Web服务不能提供以?WSDL结尾的文件。在这种情况下，它需要Web Service 代理准确指定远程WSDL地址，所下图所示：

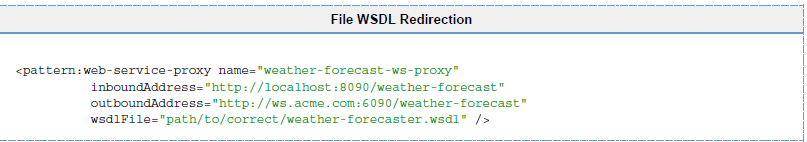


In this scenario, the remote WSDL will have its port addresses rewritten as explained above.

在这种情况下，远程WSDL端口地址将被重写按照上面所描述。

For the case when no remote WSDL is available or if the remote WSDL needs manual adjustment before being exposed by the Web Service Proxy, the solution consists in storing the correct WSDL as a local file and have the proxy serve it. This is done as shown here:

在没有远程WSDL可用或者远程WSDL暴露 Web service 代理之前需要手动调整的情况下，这个解决方案由存储在本地的WSDL文件并且此文件能为代理所应用。如下图所示：



In this case, the WSDL will be served as is from the file: no rewriting will occur.

在这种情况下，WSDL文件从文件中获得：而不是远程改写地址。

**Message Sources and Message Processors**

**消息源和消息处理器**

**Elements of Mule Programming**

**Mule的编程元素**

[ Message Sources ] [ Message Processors ]

Mule 3 provides three different constructs that can be used to build applications:

Mule3提供了三种不同的结构，可以用来构建应用程序：

Using Mule Services are the classic Mule way of organizing message flow. Each service consists of three sections:

Mule的服务，是Mule组织信息流的典型方式。每个服务由三部分组成：

Input, where messages are received

输入，接收消息

An optional component, where any sort of application logic can be applied to a message

一个可选组件，可以应用于任何形式的应用程序逻辑消息

An optional output, where the messages are sent to other services or transports.

一个可选的输出，其中的消息被发送到其他服务或运输。

Using Flows for Service Orchestration, which are new in Mule 3. A flow is a combination of message sources and message processors that doesn't have as fixed a format as a service does. It can be as simple or as complex as required, and can include, for instance, processing by multiple components before any output is performed.

在Mule 3中，增加了流动服务编排功能。此消息流以不固定的格式作为服务的信息来源和信息处理器的组合。根据不同的需求，它可以是简单的或复杂的，并且可以包括，例如，由多个组件的处理之前进行任何输出。

Using Mule Configuration Patterns are also new in Mule 3. A configuration pattern is like a pre-built flow: the logic is already built into it, so that only some simple tailoring is needed to make it functional.

在Mule3中，增加了Mule的配置模式功能。该配置模式就像是一个实现了相应的逻辑预构建的流程，因此，只需要一些简单的剪裁，就可以完成其功能。

A Mule application can contain any or all of these.

Mule的应用程序可以包含任何功能

Within flows, the streamlined architecture of Mule 3 allows Transformers, Filters, Components, Routers and other message processing artifacts to be used, and nested, freely as required.

They all implement a common MessageProcessor interface and can be used interchangeably. Services also allow these building blocks to be used in some specific extension points.

在流中，通过Mule3精简架构可以使用转换器，过滤器，组件，路由和一上些其他的消息处理器根据需求可以灵活的被使用或嵌套。

它们都实现一个共同MessageProcessor接口，并可以互相使用。服务还允许在一些特定的扩展点使用这些构建块。

**Message Sources**

**消息源**



A message source receives or generates new messages to be processed by Mule. Inbound Endpoints are currently the only supported Message Source type.

一个消息源接收或产生的新的消息都会被Mule处理。

入站端点是目前唯一支持的消息源类型。

**Inbound Endpoints**

**入站端点**

Inbound Endpoints receive new messages from a channel or resource by using a server socket, polling a remote socket or resource, or by registering a listener.

入站站点通过使用一个服务socket来接收通道或资源中的消息，或者通过远程socket或资源的轮询，或注册一个监听器来完成。

For information about available transports see Connecting Using Transports

有关可用传输的信息，请参阅使用运输

For information about configuring endpoints see Configuring Endpoints

有关配置端点的信息，请参阅配置终结点

Poll (From Mule 3.1)

查询（从Mule3.1）

Rather than using an inbound endpoint you can poll any message processor and use the result as the source of your flow.

A frequency in milliseconds can be configured otherwise the default of 1s is used.

Examples of things you can poll are outbound-endpoints, other flows or processor chains or any message processor.

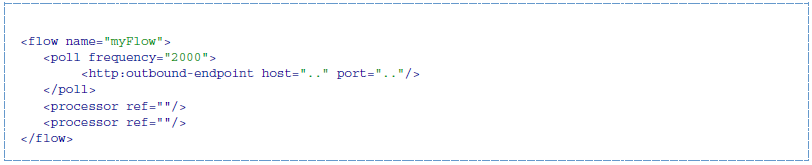
使用入站端点，而不是你可以查询任何消息处理器，并使用您的流量源。

频率是可配置的（以毫秒为单位），默认为1。

你可以轮询的对象包括出口端口或其他流或处理器链或一此消息处理器

To configure polling use the <poll> element instead of an inbound endpoint.

配置查询使用<poll>的元素，而不是入站端点



**Message Processors**

**消息处理器**



After a message has been received from a Message Source it is processed by Mule using one of more message processors. You can use message processors in certain extension points with Services, or with complete flexibility when using Flows.

当从消息源接收一个信息时此信息已经被Mule 许多消息处理器处理过。

消息处理器可提供充分的灵活性并可以在某些扩展点提供服务。

Message Processors can be categorized by function:

根据功能分类，消息处理器可以分为以下几类

消息处理器，可以按功能分类：

**Perform Some Logic**

**逻辑型消息处理器**



You'll often need to perform some business logic as part of your flow. Mule supports components implemented in Java and using scripting languages.

业务逻辑处理能力作为你的flow一部分。

Mule支持使用Java实现，使用脚本语言的组件。

Components can do whatever you need them to do including mutating the message if required.

Components support the concept of entry point resolution and lifecycle adaption designed to accommodate the use of component implementations with Mule without modification.

如果需要的话,组件可以实现任何你需要的功能包括可变的消息。

组件支持入口点分辨率和生命周期适应概念其目的是使用组件实现时并不需要修改相应组件。

Look here for more about configuring Mule Components.

在这里可以找到更多关于Mule配置组件。

**Transform the Message**

转换消息



When external systems use different message formats or you need to convert the payload type you'll need to one or more Message Transformers.

当外部系统使用不同的消息格式或你需要转换有效载荷的类型，你需要一个或多个消息转换器。

See Using Transformers for a list of available transformers as well as information on how to implement your own.

请参阅使用变压器的列表，可用的转换，以及如何实现自己的信息。

**Filter messages**

**过滤消息**

|  |  |
| --- | --- |
|  | To only allow certain messages to continue to be processed you can use a Filter. Filters may filter based on message type,message contents or some other criteria.  您可以使用一个过滤器，只允许某些消息继续进行处理。过滤器可以根据消息类型，消息的内容或一些其他的标准进行过滤。 |
|  | To prevent messages from being processed if security credentials are not provided or do not match you can use Security Filter  使用安全过滤器可以防止不提供安全证书或不匹配的消息被处理 |
|  | To filter out duplicate messages you should use an Idempotent Filter  使用等幂过滤器可以过滤掉重复的消息 |

**Control Message Flow**

**控制消息流**

There are a number of ways in which you can control message flow, which are described below. These are specified somewhat different in flows and services. In services, since the input and output sections are quite distinct, there are separate groups of Inbound Routers and Outbound Routers. In flows, routers are a subset of message processors.

下面介绍几咱控制消息流的方法。

这些被指定的流及服务有所不同。

在服务方面，由于输入和输出部分是相当不同的，有不同的入站路由器组和出站路由器组。在流，路由器是消息处理器的一个字集。

**Resequence Messages**

**重排序消息**

|  |  |
| --- | --- |
|  | In order to re-sequencer incoming messages use a Message Re-sequencer  为了重新定序器传入的消息使用消息重新定序 |

**Split or Aggregate Messages**

**拆分或聚合消息**

|  |  |
| --- | --- |
|  | Message splitters allow a single incoming message to be split into n pieces each of the parts being passed onto the next message processor as a new message.  消息拆分允许一个单一的传入消息被分成n个作为一种新的信息被传递到下一个消息处理器的部分。 |
|  | Aggregators do the opposite and aggregate multiple inbound messages into a single message.  聚合与拆分正好相反，聚合多个入站消息为一个消息。 |

For information on provided splitter and aggregator implementations and details on how to implement your own see Message Splitting and Aggregation

关于分离器和聚合器的实现和细节的更多消息就参考Message Splitting and Aggregation

**Route Messages**

|  |  |
| --- | --- |
|  |  |

In order to determine message flow in runtime Message Routers are used. Message routing can be configured statically or is determined in runtime using message type, payload or properties or some other criteria. Some message routers route to a single route whereas other routers route to multiple routes.

为了确定在运行时消息路由器的信息流被使用。

消息路由可以配置成静态或者运行时消息类型，，负载或性能或一些其他的标准。

一些消息路由器路由到一个单一的路线，而其他路由器路由到多个路由

**Send Messages over a transport**

通过传输器发送消息



Once you have have completed message processing you may wish to send the resulting message to an external service or location. You may also need to invoke a remote service elsewhere in the flow.

一旦你已经完成的消息处理，你可能希望发送结果消息给外部服务或位置。您可能还需要在流的其他地方调用远程服务。

Outbound endpoints are used to send messages over a channel using a transport.

出站端点被用来发送消息在通道上使用传输

For information about available transports see Connecting Using Transports

有关可用传输的信息，请参阅使用运输

For information about configuring endpoints see Configuring Endpoints

有关配置端点的信息，请参阅配置终结点

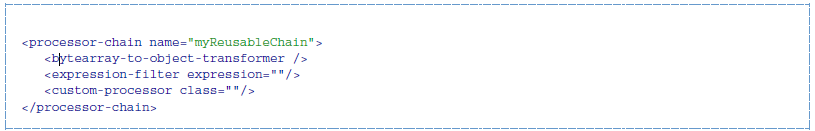
**Other**

***Message Processor Chain***

***消息处理器链***

A Message Processor Chain allows to define a reusable set of message processors that are chained together and invoked in sequence. When configuring Mule using XML a processor chain is defined using the *processor-chain* element.

消息处理器链可以定义一组可重用的消息处理器是串联在一起的，依次调用。当Mule中使用*processor-chain*节点配置一个处理器链。



***Response Adaptor***

***响应适配器***

A response adaptor is configured using the *response* element.

It is used when you want to use a Message Processor on a response message.

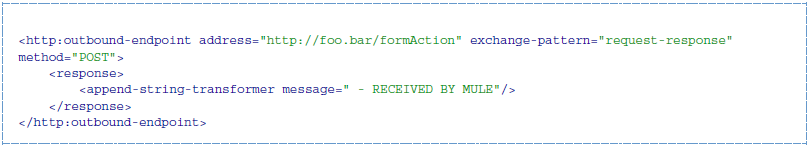
In the following case the append-string-transformer is invoked after response was received from the outbound endpoint invocation.

This can be useful when you have a message process that performs response processing (e.g. CXF) and you need to add a message processor after this.

通过使用response节点配置响应适配器。一般在响应消息使用消息处理器时被使用。

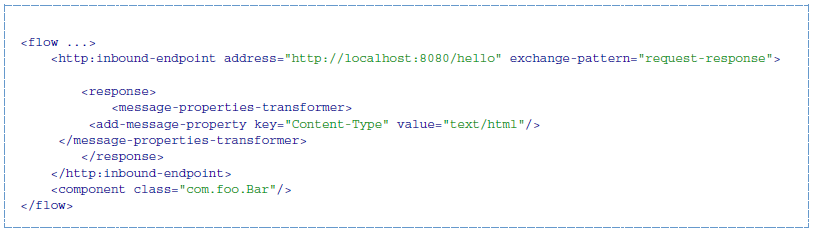
在下面的情况下，在出口端口被调用之后收到响应时消息处理器将调用append-string-transformer。

当你有一个消息处理过程来执行消息的响应处理那么你就需要加上一个消息处理器。



In the following example response block is invoked after the flow finished processing and before the response message is returned to the caller of the inbound endpoint.

在下面的示例响应块被调用的流程之后的完成处理和响应消息之前被返回到主叫方的入站端点。



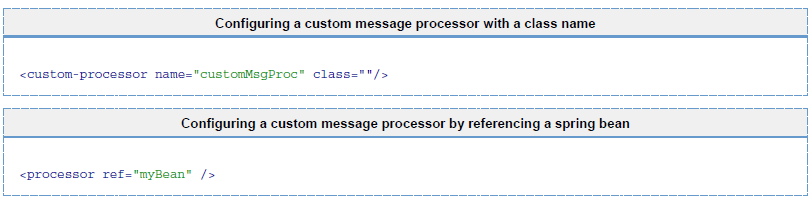
***Custom Message Processors***

***自定义消息处理器***

Custom Message Processors can be implemented by simply extending the MessageProcessor or InterceptingMessageProcessor interface and using the <custom-processor> element. If you prefer to used a referenced spring bean as a message processor then you can use the standard <processor ref=""/> element and reference it directly.

自定义消息处理器可以通过简单地MessageProcessor或者InterceptingMessageProcessor接口来实现或者通过<custom-processor>节点配置。

如果你喜欢使用一个被引用的Spring bean作为消息处理器，那么你可以使用的标准<processor ref=""/>节点直接引用它。



For information on implementing your own Filters or Transformers see the respective pages. There is also more detailed information on implementing your own Custom Message Processors.

如果你想查看更多的消息关于如何实现消息过滤器或转换器相关的页面

也有更详细的信息，实现自己的自定义消息处理器。

**Routing Message Processors**

**路由消息处理器**

**Using Message Processors to Control Message Flow**

**使用消息处理器来控制消息流**

[ Quick Reference ] [ All ] [ Async ] [ Choice ] [ Collection Aggregator ] [ Collection Splitter ] [ Custom Aggregator ] [ Custom Processor ] [ First Successful ] [ Idempotent Message Filter ] [ Idempotent Secure Hash Message Filter ] [ Message Chunk Aggregator ] [ Message Chunk Splitter ] [ Message Filter ] [ Processor Chain ] [ Recipient List ] [ Redelivery Policy ] [ Request Reply ] [ Resequencer ] [ Round Robin ] [ Splitter ] [ Until Successful ] [ WireTap ]

Message Processors are used within flows to control how messages are sent and received within that flow. This is further described in Using Flows for Service Orchestration. (Within services, the same job is performed by message routers – see Using Mule Services and Using Message Routers for further details.)

在流中使用消息处理器来控制消息的发送和接收。

此功能将在流动服务编排中进一步说明。

（在服务中，此功能由消息路由来完成- 详细说明请参阅使用Mule的服务和使用消息路由器。）

Click a link in the Quick Reference table below for details on a specific message processor.

点击下面的快速参考表中的链接在一个特定的消息处理器的详细信息。

**Quick Reference**

**Message Processor Description**

**消息处理器 描述**

All Broadcast a message to multiple targets

所有 广播消息到多个目标

Async Run a chain of message processors in a separate thread

异步 在一个单独的线程中运行链的消息处理器

Choice Send a message to the first matching message processor

选择 将消息发送到第一个匹配的消息处理器

Collection Aggregator Aggregate messages into a message collection

收集聚合 聚合消息到消息集合

Collection Splitter Split a message that is a collection

收集分配器 拆分消息是一个集合

Custom Aggregator A custom-written class that aggregates messages

定制的聚合 自定义编写的类，它可以聚合消息

Custom Processor A custom-written message processor

定制处理器 自定义编写的消息处理器

First Successful Iterate through message processors until one succeeds (added in 3.0.1)

首次成功 遍历一个成功的消息处理器，直到成功（3.0.1中添加）

Idempotent Message Filter Filter out duplicate message by message ID

等幂消息过滤器 通过消息ID过滤掉重复的消息

Idempotent Secure Hash Message Filter Filter out duplicate message by message content

幂等安全Hash消息过滤器 通过消息内容过滤重复的消息

Message Chunk Aggregator Aggregate messages into a single message

消息块聚合 汇总消息的到单一消息

Message Chunk Splitter Split a message into fixed-size chunks

消息块分配器 将消息分割成固定大小的块

Message Filter Filter messages using a filter

消息过滤器 使用过滤器过滤消息

Processor Chain Create a message chain from multiple targets

处理器链 创建一个消息链的多个目标

Redelivery Policy Specify processing for a message which has been redelivered sufficiently often

再投递政策 指定的消息已经重新传送足够的经常处理

**[Mule 3.2]** Request Reply Receive a message for asynchronous processing and accept the asynchronous response on a different channel

**Mule3.2** 请求回复 接收消息异步处理，并承担在不同的异步响应 channel 渠道

Resequencer Reorder a list of messages

再顺 重新排序的邮件列表

Round Robin Round-robin among a list of message processors (added in 3.0.1)

循环 循环赛之间的信息处理器的列表（在3.0.1添加）

**[Mule 3.2]** Until Successful Repeatedly attempt to process a message until successful

**[Mule 3.2]，** 直到成功 反复尝试处理消息，直到成功

Splitter Split a message using an expression

分离器 拆分消息，使用表达式

WireTap Send a message to an extra message processor as well as to the next message processor in the chain

窃听 一个额外的消息处理器将消息发送给链中的下一个消息处理器，以及

**All**

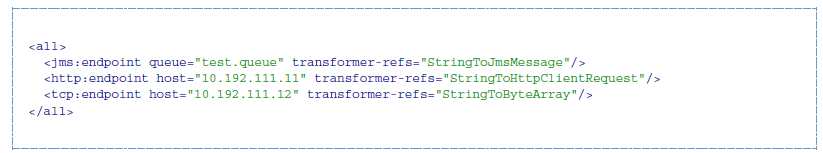
**所有**

The All message processor can be used to send the same message to multiple targets.

所有的消息处理器可以发送同样的消息到多个目标。

Configuration for this router is as follows:

该路由器的配置如下：



If any of the targets specified is an endpoint that has a filter configured on it, only messages accepted by that filter are sent to that endpoint.

如果任何指定的目标是一个端点，其上配置有一个过滤器，只有该过滤器所接受的消息被发送到该端点

All messages (if any) returned by the targets are aggregated together and form the response from this processor.

由目标返回的所有消息（如果有的话）被聚集在一起，并形成从该处理器的响应。

**Async**

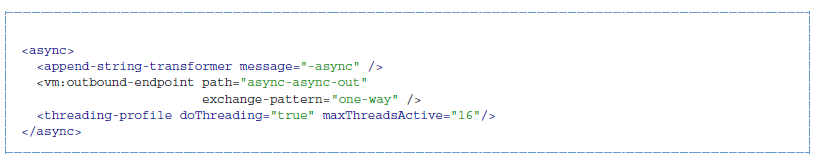
**异步**

The Async message processor runs a chain of message processors in another thread, optionally specifying a threading profile for the thread to be used.

The message processsor is configured as follows:

异步消息处理器运行链的消息在另一个线程处理器，也可以指定一个线程的线程配置文件 使用。

此消息处理器的配置文件如下图所示：



This transforms the current message and sends it to the specified endpoint, using a threadpool that contains up to 16 concurrent threads.

此转换器将当前的消息发送到指定的端口并使用一个线程池此线程池最鑫包含16个并发线程！

**Choice**

**选择**

The Choice message processor sends a message to the first message processor that matches. If none match and a message processor has been configured as "otherwise", the message is sent there. If none match and no otherwise message processor has been configured, an exception is thrown.

选择消息处理器发送一个消息到第一相匹配消息处理器。

如果没有匹配和信息处理器并且此消息处理器配置了否则节点，则此消息将走否则节点。

如果消息处理器没有相应的匹配而且没有配置otherwise节点，则抛出异常。

Choice is configured as follows:

选择配置如下：



If the message payload is "foo" or "bar", the corresponding transformer is run. If not, the transformer specified under "otherwise" is run.

如果消息中的负载为“foo”，或“bar”,，转换器将正常执行,否则转换器将在otherwise下执行。

**Collection Aggregator**

**收集聚合**

The Collection Aggregator groups incoming messages that have matching group IDs before forwarding them. The group ID can come from the correlation ID or another property that links messages together.

收集聚合组前转发，组ID匹配传入的消息。

本组ID可以来自. 相关ID或另一个属性，链接的消息。

You can specify the timeout attribute to determine how long the router waits in milliseconds for messages to complete the group. By default, if the expected messages are not received by the timeout time, an exception is thrown and the messages are not forwarded. You can also set the failOnTimeout attribute to false to prevent the exception from being thrown and simply forward whatever messages have been received so far.

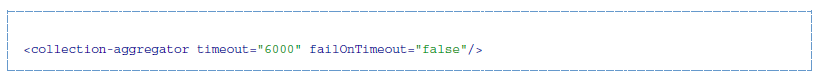
你可以指定timeout属性，以确定路由器完成消息组合需要多少毫秒。

默认情况下，如果在timeout设定的时间之内没有得到消息，将会跑出异常并且消息不会被转发。

您还可以设置 failOnTimeout 属性为false，以防止异常被抛出，简单的转发任何消息已收到远。

Configuration for the Collection Aggregator is as follows:

收集聚合的配置如下：



**Collection Splitter**

**收集分配器**

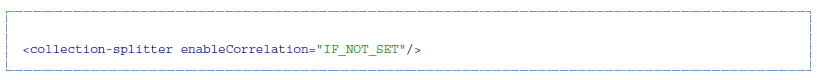
The Collection Splitter acts on messages whose payload is a Collection type. It sends each member of the collection to the next message processor as separate messages. You can specify the attribute enableCorrelation to determine whether a correlation ID is set on each individual message.

收集分配器应用在消息负载为收集类型的消息中。消息分配器分配集合中的每个元素作为一个分开的消息下一个消息处理器处理。

您可以指定属性enableCorrelation，以确定相关的ID是否设置了个人信息。

Configuration for the Collection Splitter is as follows:

收集分离器的配置如下：



**Custom Aggregator**

**自定义聚合**

A Custom Aggregator is an instance of a user-written class that aggregates messages.

This class must implement the interface MessageProcessor. Often, it will be useful for it to subclass AbstractAggregator, which provides the skeleton of a thread-safe aggregator implementation, requiring only specific correlation logic. As with most custom objects in Mule, it can be configured either with a fully specified class name or as a reference to a Spring bean. It can also be configured with the same timeout and failOnTimeout attributes described under Collection Aggregator.

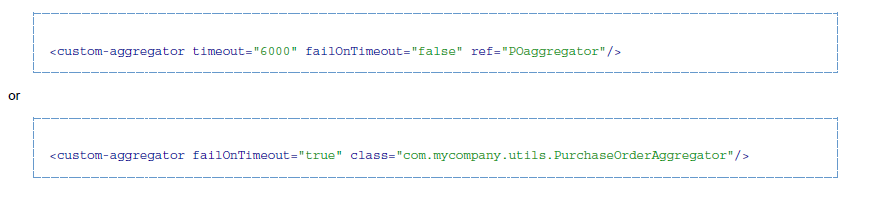
一个定制的聚合是用户编写的类，它可以聚合消息。这个类必须实现MessageProcessor .接口。

通过情况下，我们应用它的AbstractAggregator子类，该子类实现了线程安全性，只要实现相应的逻辑。

在Mule中的自定义对象，可以通过sping bean来进行配置。

Configuration for a Custom Aggregator is as follows:

一个定制的聚合配置如下：



**Custom Processor**

**自定义处理器**

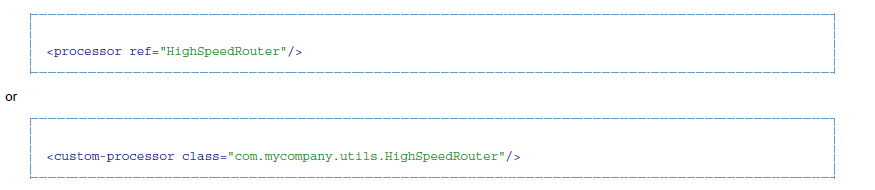
A Custom Processor is an instance of a user-written class that acts as a message processor. This class must implement the interface MessageProcessor. As with most custom objects in Mule, it can be configured either with a fully specified class name or as a reference to a Spring bean.

定制处理器的消息处理器，作为一个用户编写的类的一个实例。  
这个类必须实现MessageProcessor接口.。

自定义对象的骡子，它可以配置一个完全指定的类名或作为参考一 Spring bean.一样。

Configuration for a Custom Processor is as follows:

自定义处理器的配置如下：



**First Successful**

**首次成功**

The First Successful message processor iterates through its list of child message processors, routing a received message to each of them in order until one processes the message successfully. If none succeed, an exception is thrown.

第一次成功的消息处理器遍历列表子消息处理器，路由收到的消息，他们每个人在直到为了处理消息成功。 如果没有成功，抛出一个异常。

Success is defined as:

成功的定义是：

If the child message processor thows an exception, this is a failure.

如果子信息处理器泡出异常，收失败。

Otherwise:

否则：

If the child message processor returns a message that contains an exception payload, this is a failure.

如果子消息处理器返回一个包含异常负载的消息，则失败。

If the child message processor returns a message that does not contain an exception payload, this is a success.

如果子消息处理器退回一个不包含异常负载的消息,则成功。

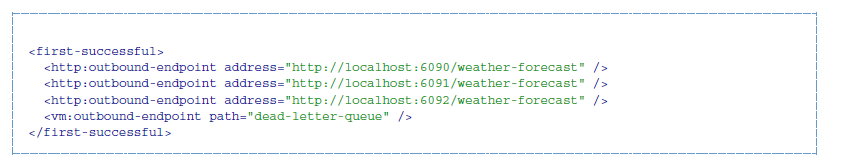
If the child message processor does not return a message (e.g. is a one-way endpoint), this is a success.

如果子消息处理器没有返回一个消息（例如，是一个单向的端点），则成功

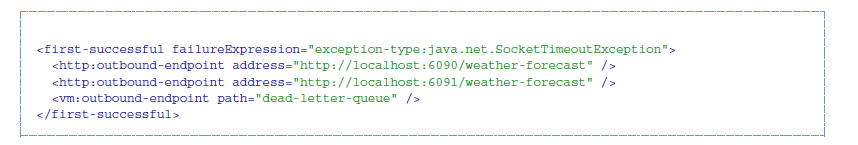
如果孩子消息处理器不会返回一个消息（例如，是一个单向的端点），这是成功的。

This message processor was added in Mule 3.0.1.

Mule3.0.1加入了消息处理器



从3.1.0开始,你可以通过指定faiureExpression属性来进一步自定义路由器行为，并且可以使用mule表达来定义一个故障。failureExpression 属性配置如下图所示：



In the above example a failure expression is being used to more exactly define the exception type that will be considered a failure, alternatively you can use any other Mule expression that can be used with expression filters, just remember that the expression denotes failure rather than success.

另外，在上述的例子中的故障是被用来表达的异常的类型，将被认为是失败的，或者，更准确地定义你可以使用任何其他的Mule表达式，可以用表达式过滤器，只记得表达式表示失败，而不是 success. 成功。

**Idempotent Message Filter**

**等幂消息过虑器**

An idempotent filter ensures that only unique messages are received by a service by checking the unique message ID of the incoming message. The ID can be generated from the message using an expression defined in the idExpression attribute. By default, the expression used is #[message:id], which means the underlying endpoint must support unique message IDs for this to work. Otherwise, a UniqueIdNotSupportedException is thrown.

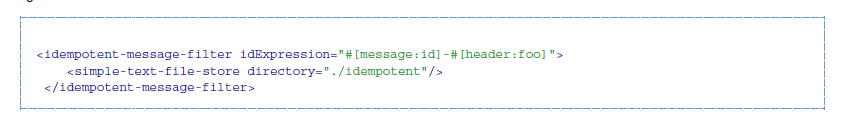
等幂过滤器必须确定通过服务接收的消息是唯 一的，通过message ID判断其唯一性。消息ID可以通过定义idExpression属性表达式自动产生。默认情况下，所使用的表达式是 #[message:id] ,， 这意味着底层的终端必须支持唯一的消息ID，否则一个 UniqueIdNotSupportedException 被抛出

There is a simple idempotent filter implementation provided at org.mule.routers.IdempotentMessageFilter . The default implementation uses a simple file-based mechanism for storing message IDs, but you can extend this class to store the IDs in a database instead by implementing the ObjectStore interface.

有一个简单的幂等滤波器的实现org.mule.routers.IdempotentMessageFilter接口。 此接口默认实现使用一个简单的基于文件的机制来存储信息的ID，但你可以扩展这个类通过实现ObjectStore接口，用数据库来存储消息ID。

Configuration for this router is as follows:

该路由器的配置如下：



The optional idExpression attribute determines what should be used as the unique message ID. If this attribute is not used, #[message:id] is used by default.

可选的 idExpression 属性决定了什么应该被用来作为唯一的消息ID。如果idExpression属性没有被使用，默认情况下使用#[message:id]。

The nested element shown above configures the location where the received message IDs are stored. In this example, they are stored to disk so that the router can remember state between restarts. If the directory attribute is not specified, the default value used is ${mule.working.dir}/objectstore where mule.working.dir is the working directory configured for the Mule instance.

如上所示嵌套元素来存储消息ID。在这个例子中，消息ID被存储在磁盘上，以便该路由器可以记得重新启动之间的状态。如果directory没有被指定,默认值为${mule.working.dir}/objectstore，${mule.working.dir}为mule实例的配置目录。

If no store is configured, the InMemoryObjectStore is used by default.

如果没有存储配置，默认情况下使用的InMemoryObjectStore。

**Idempotent Secure Hash Message Filter**

**等幂安全散列消息过滤器**

This filter ensures that only unique messages are received by a service by calculating the hash of the message itself using a message digest algorithm.

此消息过滤器必须确定通过服务接收的消息是唯一的，用消息整合算法来计算消息本身的HASH值来确定其唯一性。

This approach provides a value with an infinitesimally small chance of a collision and can be used to filter message duplicates.

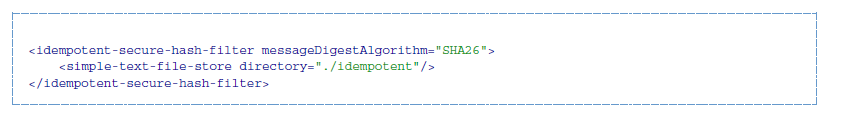
这种方法提供了价格与无穷小碰撞的机会来过滤重复的消息

Note that the hash is calculated over the entire byte array representing the message, so any leading or trailing spaces or extraneous bytes (like padding) can produce different hash values for the same semantic message content. Therefore, you should ensure that messages do not contain extraneous bytes. This router is useful when the message does not support unique identifiers.

该过滤器可确保计算散列的消息使用消息摘要唯一的消息被接收由服务算法。这种方法提供了价值与无穷小的碰撞的机会，可以用来过滤消息的重复。注意(like 在整个字节数组，表示该消息的哈希计算，所以任何前导或尾随空格或多余的字节（如填充物）可以产生不同的散列值相同的语义信息内容。因此，您应确保消息不包含多余的字节。有用的信息时，该路由器是不支持的唯一标识符。

Configuration for this filter is as follows:

此过滤器的配置是如下：



Idempotent Secure Hash Message Filter also uses object stores, which are configured the same way as the Idempotent Message Filter. The optional messageDigestAlgorithm attribute determines the hashing algorithm that will be used. If this attribute is not specified, the default algorithm SHA-256 is used.

等幂安全散列邮件筛选器也使用对象存储，存储方式与幂等邮件筛选器配置相同。 用可选的 messageDigestAlgorithm 属性确定将使用的哈希算法。如果这个属性没有指定，默认 为SHA-256算法。

**Message Chunk Aggregator**

**消息块聚合**

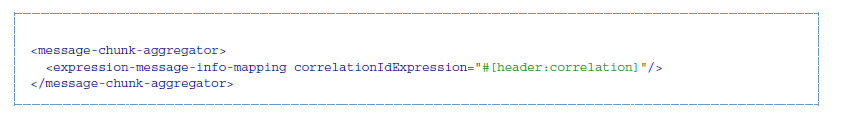
After a splitter such as the Message Chunk Splitter splits a message into parts, the message chunk aggregator router reassembles those parts back into a single message. The aggregator uses the message's correlation ID to identify which parts belong to the same message.

经过分离器如消息快分离器，将消息分割成几部分，消息块聚合路由器重新将这些分割的消息组合成一个新的消息

聚合器使用消息的相关ID，以确定哪些部分属于相同的消息。

Configuration for the Message Chunk Aggregator is as follows:

消息块在聚合器的配置是如下：



The optional expression-message-info-mapping element allows you to identify the correlation ID in the message using an expression. If this element is not specified, MuleMessage.getCorrelationId() is used.

The Message Chunk Aggregator also accepts the timeout and failOnTimeout attributes as described under Collection Aggregator.

可选的 expression-message-info-mapping 元素可以通过使用表达方式让你识别消息中的相关ID。 如果 此元素没有被指定，MuleMessage.getCorrelationId（）被使用。 消息块聚合也可以接受的 timeout 和 failOnTimeout 属性所描述的 收集聚合 。

**Message Chunk Splitter**

**消息块分配器**

The Message Chunk Splitter allows you to split a single message into a number of fixed-length messages that will all be sent to the same message processor. It will split the message up into a number of smaller chunks according to the messageSize attribute that you configure for the router. The message is split by first converting it to a byte array and then splitting this array into chunks. If the message cannot be converted into a byte array, a RoutingException is raised.

消息块分配器，可以让你把一个消息分离为一些固定长度的消息，并将这些消息发送到同一个消息处理器。，根据为路由器messageSize属性的配置，可以将讯息分割成较小的块数。该消息被分割，将其转换为一个字节数组，然后分裂成块的数组。如果消息不能被转换成一个字节数组，将抛出RoutingException异常。

A message chunk splitter is useful if you have bandwidth problems (or size limitations) when using a particular transport.

一个消息块分离器是非常有用的，如果你有使用特定的传输带宽问题（或​​大小限制）。

To put the chunked items back together again, you can use the Message Chunk Aggregator.

再次把分块的集中在一起，你可以使用消息块聚合。

Configuration for the Message Chunk Splitter is as follows:

消息块在分路器的配置是如下：



**Message Filter**

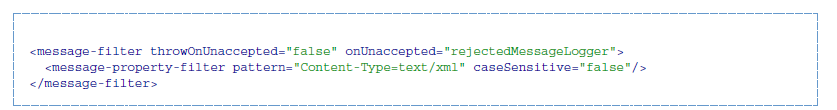
**消息过滤器**

The Message Filter is used to control whether a message is processed by using a filter. In addition to the filter, you can configure whether to throwan exception if the filter does not accept the message and an optional message processor to send unaccepted messages to.

消息过滤器是用来控制消息是否使用过滤器处理。除了过滤器，您可以配置是否抛出异常，如果过滤器不接受消息或者一个可选的消息处理器发送未被接受的消息。

Configuration for the Message Filter is as follows:

消息过滤器的配置是如下：



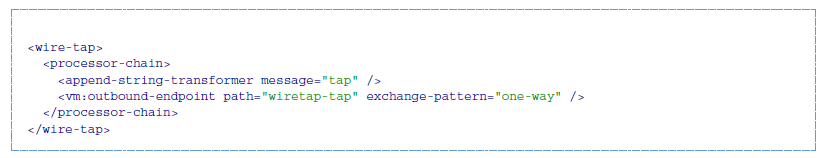
**Processor Chain**

**处理器链**

A Processor Chain is a linear chain of message processors which process a message in order. A Processor Chain can be configured wherever a message processor appears in a Mule Schema. For example, to allow a Wire Tap to transform the current message before sending it off, you can configure the following:

处理器链是许多消息处理器的直链。Mule框架中任何消息处理器都可以配置一个处理器链。

例如，允许将目前的信息，然后将它发送一个窃听，你可以进行以下配置：

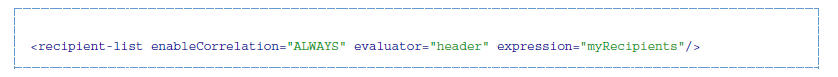


**Recipient List**

**收件人列表**

The Recipient List message processor allows you to send a message to multiple endpoints by specifying an expression that, when evaluated, provides the list of endpoints. These messages can optionally be given a correlation ID, as in the Collection Splitter. An example is

收件人列表消息处理器，通过指定的表达式允许你发送一个消息到多个端点，计算时，提供的端点清单。这些消息可以任选地给定的相关性ID，如Collection中的分配器。一个例子是



which finds the list of endpoints in the message header named myRecipients.

命名myRecipients在邮件标题中发现的端点清单

**Redelivery Policy**

**再投递政策**

The Redelivery Policy determines what happens when the processing of a message causes an exception, resulting in the message being redeliver over and over. The policy consists of

重新传送策略确定的消息处理时会发生什么导致了异常，从而导致消息重新传送一遍又一遍。该政策由以下几个部分组成

A message processor that will be called if the message is redelivered too many times without being processed successfully

如果一个消息发送了很多次都没有被处理成功，消息处理将被会调用。

How many redeliveries will be allowed before that occurs. (Default: 5)

多少次重复传递将被允许之前发生的。 （默认值：5）

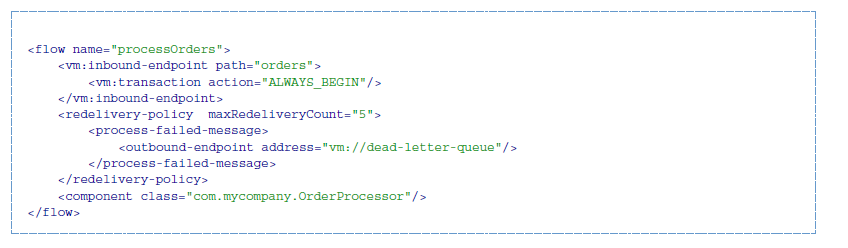
How to identify messages as being "the same"; By default, the message contents are used, as in the

Idempotent Secure Hash Message Filter but, optionally, an expression can be specified, as in the Idempotent Message Filter.

如何确定的消息是“相同的”默认情况下通过消息内容来确定，也可以通过等幂安全散列消息过滤器，并且可以通过等幂安全散列消息过滤器可以指定一个表达式来过滤。

Here is an example of its use:

关于它的使用如下图所示：



This flow reads messages transactionally from a vm endpoint, and passes them to the order-processing component. Supposes that this component intermittently throws exceptions when resources are unavailable. That will cause the transaction to be aborted and the message reprocessed, which is desirable. But if the the same message is reprocessed and rejected five times is a row, on its next redelivery it will be sent to the dead letter queue instead.

这个流中从VM商战读取消息事务，并把它们传递到订单处理组件。

假设一下一部分资源不可用,则抛出异常。理想情况下这将导致事务回滚消息将继续被发送。

如查同一个消息被处理并被拒绝了5次，则此消息将被发送到死信队列中。

**Request Reply**

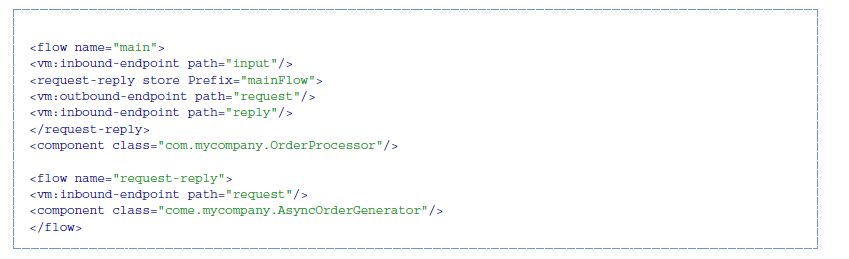
**请求答复**

**[Mule 3.2]**

[Mule3.2]

The Request Reply message processor receives a message on one channel, allows the back-end process to be forked to invoke other services asynchronously, and accepts the asynchronous result on another channel.

Here is an example that uses the Request Reply message processor:  
请求回复消息处理器在一个通道上接收到一条消息，让后端进程分叉到异步调用其他服务，并接受另一个通道上的异步结果。  
下面是一个使用请求的回复消息处理器例子：



The request is received in the main flow. It then configures an asynchronous request-reply that sends the message to the request-reply flow and implicitly sets the MULE\_REPLYTO message property to vm://reply. This tells the request-reply flow where to send its response. The main flow then waits for the response. The request-reply flow receives the message asynchronously. The request is then processed by the AsynchOrderGenerator component. When the process is complete, the reply is sent to vm://reply. The asynchronous response is received and

give to the OrderProcessor component to complete the order processing.

在主流程中一个请求被接收。然后配置一个异步请求 - 应答，将消息发送到请求 - 应答流和并为vm:/reply隐式设置MULE\_REPLYTO的消息属性。这会告诉请求-应答流向那里发送一个响应。然后主流程等待此响应。异步接收消息的请求 - 应答流。然后，该请求被处理由AsynchOrderGenerator组分。当处理完成时，发送答复是vm:/ /答复。异步响应，并给了OrderProcessor组件，以完成订单处理。

**Resequencer**

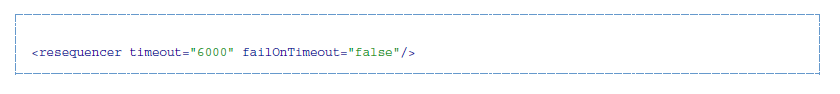
**重新排序**

The Resequencer sorts a set of received messages by their correlation sequence property and issues them in the correct order. It uses the timeout and fileOnTimeout attributes described in Collection Aggregator to determine when all the messages in the set have been received.

重新组合一组接收到的消息通过他们的序列号进行重新反序，以正确的顺序接到消息。重新排序支持timeout和fileOnTimeout属性，以确定人集合中所有的消息被接收。

The Resequencer is configured as follows:

重新排序配置如下：



**Round Robin**

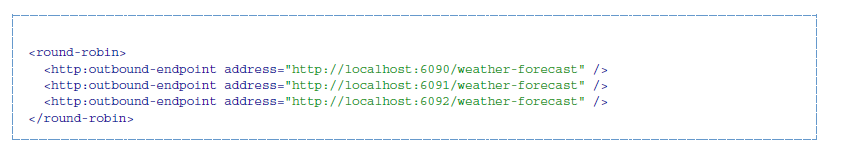
**循环**

The Round Robin message processor iterates through a list of child message processors in round-robin fashion: the first message received isrouted to the first child, the second message to the second child, and so on. After a message has been routed to each child, the next is routed to the first child again, restarting the iteration.

循环消息处理器通过round-robin方式遍历列表的子消息处理器第一个消息对应着第一个子元素第二个消息对应第二个子元素以此类推。已经排到每个孩子的消息后，下一个被路由的第一个孩子，重新启动迭代

This message processor was added in Mule 3.0.1.

Mule3.0.1加入了此消息处理器。



**Splitter**

**分离器**

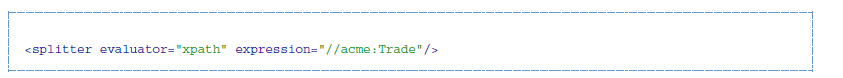
A Splitter uses an expression to split a message into pieces, all of which are then sent to the next message processor. Like other splitters, it can optionally specify non-0default locations within the message for the message ID and correlation ID.

分离器使用表达式将一个消息分割成片，然后将被分割成的消息送到下一个消息处理器。

像其他分路器，它可以选择性地指定非0default的位置内的消息的消息ID和相关性ID。

The Splitter is configured as shown below:

分配器被配置为如下所示：



This uses the specified XPath expression to find a list of nodes in the current message and sends each of them as a separate message.

使用指定的XPath表达式找到当前消息一个节点，并将他们作为单独的消息发送。

**Until Successful**

**直到成功为止**

**[Mule 3.2]**

The Until Successful message processor processes a message with its child message processor until the processing succeeds. This processingoccurs asynchronously, therefore execution is returned to the parent flow immediately.

The Until Successful message processor is able to retry:

直到成功消息处理器用他的子消息处理处理一个消息直到处理成功

这processingoccurs异步的，因此执行立即返回到父流。  
直到成功消息处理器支持重试：

Dispatching to outbound endpoints, for example, when you’re reaching out to a remote web service that may have availability issues.

调度出站端点，例如，当你调用一个远程Web服务，可能有可用性问题。

Execution of a component method, for example, to retry an action on a Spring Bean that may depend on unreliable resources.

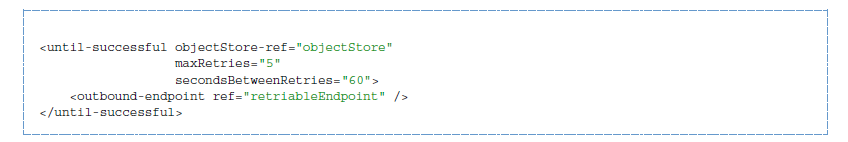
执行组件的方法，例如，重试一个Spring Bean，它可能取决于在不可靠的资源上的动作。

A sub-flow execution, to keep reexecuting several actions until they all succeed.

子流程执行，以保持重新执行几个动作，直到他们全部获得成功。

Any other message processor execution, to allow more complex scenarios.

任何其他的消息处理器执行，允许更复杂的情况。



This message processor needs an ListableObjectStore instance in order to persist messages pending (re)processing. There are severalimplementations available in Mule, including the following:

为了坚持待（再）处理的消息，此消息处理器需要一个ListableObjectStore的实例。有可在Mule severalimplementations，其中包括以下：

DefaultInMemoryObjectStore. The default in-memory store.

默认情况下，在内存中存储。

DefaultPersistentObjectStore. The default persistent store

默认持久性存储

FileObjectStore. A file-based store.

基于文件的存储

QueuePersistenceObjectStore. The global queue store.

全球队列

SimpleMemoryObjectStore. An in-memory store

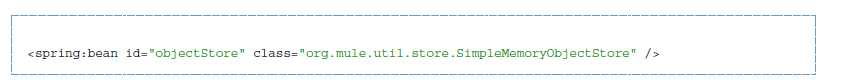
一个在内存中存储

See Mule Object Stores for further information about object stores in Mule.

关于对Mule对象存储的进一步了解请参考Mule的对象存储

Here is how you would create an in-memory store:

下图显示如何时创建一个内存存储



Success or failure are defined as:

成功或失败的定义如下：

If the child message processor throws an exception, this is a failure.

如果子消息处理器抛出一个异常，这是失败的。

If the child message processor does not return a message (e.g. is a one-way endpoint), this is a success.

如果子消息处理器不会返回一个消息（例如，是一个单向的端点），这是成功的。

If a 'failure expression' (see below) has been configured, the return message is evaluated against this expression to determine failure or not.

如果错误表达式（见下文）进行配置，返回的消息进行评估，以确定故障或对这种表达。

Otherwise:

否则

If the child message processor returns a message that contains an exception payload, this is a failure.

如果子消息处理器返回一个消息，其中包含一个异常的有效载荷，这是失败的。

If the child message processor returns a message that does not contain an exception payload, this is a success.

如果子消息处理器返回一个消息不包含一个异常的有效载荷，这是成功的。

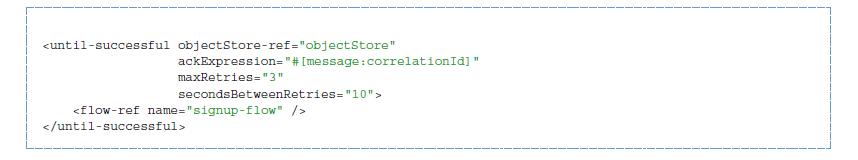
Here is an example showing how to configure the failure expression:

下面是一个例子，显示了如何配置失败的表达式：



The Until Successful message processor is also able to synchronously acknowledge that it has accepted a message and will try to process it repeatedly. The following is an example where the message correlation ID is used as an acknowledgement message:

直到成功消息处理器也能同步确认，已经接受了消息，并会尝试重复处理。以下是一个确认消息的消息相关的ID被用作一个例子，其中：



It is also possible to define a DLQ (dead letter queue) endpoint to which messages will be sent if they have failed processing too many times:

它也可以定义一个DLQ（死信队列）端点的信息将被发送，如果他们没有处理过很多次：

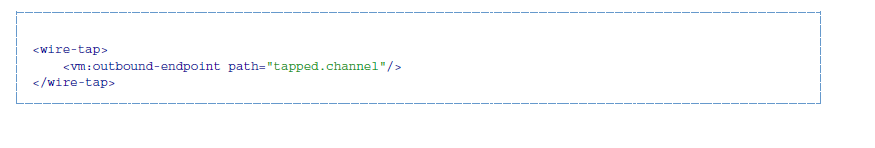


**WireTap**

**窃听**

The WireTap message processor allows you to route certain messages to a different message processor as well as to the next one in the chain. For instance, To copy all messages to a specific endpoint, you configure it as an outbound endpoint on the WireTap routing processor:

窃听消息处理器让您的某些消息路由到不同的消息处理器以及链中的下一个。例如，要复制所有消息到一个特定的端点，将其配置为出站端点上的窃听路由处理器：



***Using Filters with the WireTap***

***使用过滤器窃听***

The WireTap routing processor is useful both with and without filtering. If filtered, it can be used to record or take note of particular messages or to copy only messages that require additional processing. If filters aren't used, you can make a backup copy of all messages received. The behavior here is similar to that of an interceptor, but interceptors can alter the message flow by preventing the message from reaching the component. WireTap routers cannot alter message flow but just copy on demand. In this example, only messages that match the filter expression are copied

to the vm endpoint.

窃听路由处理器是有用的，未经过滤的。如果过滤的，它可以被用来记录或采取注意到特定的消息或仅复制消息，需要额外的处理。如果不使用过滤器，可以使接收的所有邮件的备份副本。这里的行为是类似的一个拦截器，但拦截器可以改变通过防止到达该组件的消息的消息流。窃听的路由器不能改变的消息流，但只是复制的需求。在这个例子中，只有相匹配的过滤器表达式的消息被复制到虚拟机的端点。

