# Mule ESB 3使用要点

## Mule ESB 3使用要点

Mule是一个灵活的消息处理和集成框架。你使用Mule的方式取决于你要尝试解决的问题。Mule3提供了多种配置构建方法，这些方法可以根据需要被混合和装配，来实现你的方案。

* 理解Mule配置
* 在流、模式或服务之间进行选择
* 消息源和消息处理器
* 配置组件
* 使用传输器做连接
* 配置端点
* 使用过滤器
* 使用转换器
* 使用Mule云连接来连接SaaS，社交媒体和电子商务
* Mule查询语言
* 使用表达式
* 消息属性域
* 事务管理
* 配置安全
* 错误处理
* 使用Web Services

## 理解Mule配置

### 理解Mule配置

[XML配置相关] [Schema参考] [默认值] [枚举值] [强类型语言的优点] [Spring配置相关] [Spring Beans] [Spring属性] [属性占位符] [Mule配置相关] [全局元素] [流] [配置模式] [服务] [自定义元素] [系统级配置] [管理器] [代理]

#### XML配置相关

Mule使用XML配置来定义每个应用，它完整的描述了运行应用需要的组成部分。一个基本的Mule应用可以使用非常简单的配置，例如：



下面我们将仔细查看这一配置的所有片段的细节。目前，尽管它很简单，却是一个完整的应用，并且非常易读：甚至对Mule只有短时间了解的人也能够解释清楚，它是在从标准输入向标准输出拷贝消息。

##### Schema参考

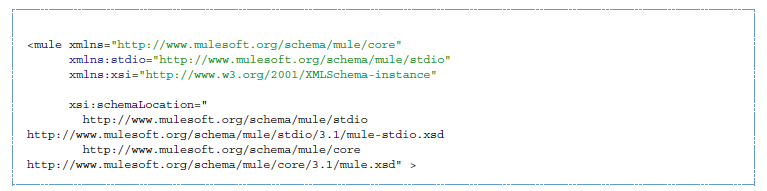
Mule配置的语法是由一组XML schema文件定义的。每一个配置列出了它用到的schema，并给出了它们的URL地址。它们中的大部分是你正在使用的Mule版本所对应的Mule schema， 但是另外也可能有一些第三方的schema，例如：

* Spring schema，定义了用到的所有Spring标签元素（比如Spring beans）的语法
* CXF schema，用于Mule CXF模块所处理的Web Services的配置

一个配置文件中引用到的每个schema都由两部分数据定义：

* 名字空间，这是一个URI
* 位置，这是一个URL

配置文件既定义了schema的名字空间URI作为XML名字空间，同时又关联了schema的名字空间和位置。像我们从上面的配置文件中看到的一样，这是由顶级元素mule标签来完成的：



这里将mule core的schema名字空间被定义为配置文件的默认名字空间。它是这里默认名字空间的最佳选择，因为有这么多配置的元素是属于核心名字空间的。

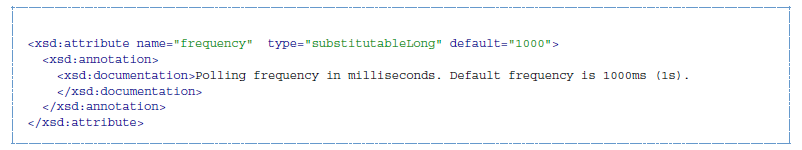
Mule stdio传输器允许使用标准I/O进行通信的，它的名字空间使用“stdio”作为前缀。Mule模块或者传输器里的约定是使用它们的名字作为名字空间的前缀。

Xsi:schema属性关联名字空间到它们的位置，这里它提供了stdio schema和mule core schema的位置。

Mule配置中包含这些部分是必须的，因为它们保证了schema能够被查找到，这样配置文件才能使用它们进行检验。

##### 默认值

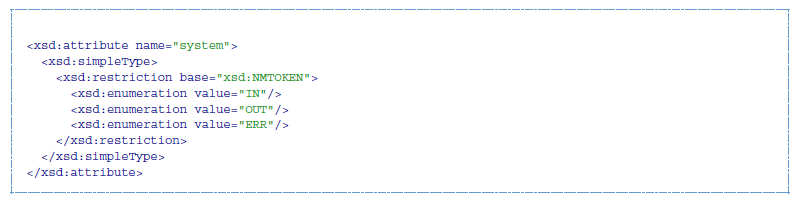
除定义了元素和属性的语法外，schema还可以给它们定义默认值。众所周知，这是对于让你的配置可读是极其有用的，因为它们不会因为不必要的信息变得杂乱。默认值可以通过schema本身来查找，或者在Mule的模块和传输器的该当中也定义了这些值。例如，用于重复地向端口轮询的<poll>元素，它的定义包含了以下的属性定义：



只有当要覆盖默认的1s的值时，才需要指定这一属性。

##### 枚举值

许多Mule的属性只可以接受一个限定的集合内的值。它们在schema中被定义为枚举值，来保证只有这些值才能被指定。这里是一个stdio传输器的例子：



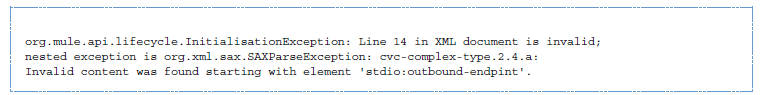
这强制指定了只接受有IN、OUT和ERR，它们分别代表标准输入、输出和错误。

##### 强类型语言的优点

要引用到所有schema的需求好像非常的笨重，但是它有两个远超出所付出的精力的优势。

第一，它帮助你一次就能创建出有效的配置。主流的集成开发环境都提供了schema感知的XML编辑器。因此，当你创建和编辑你的配置时，IDE会提示你在每个点上可以接受的元素和属性，在你只输入了几个字符后实例它们的名字，以及高亮指示出需要修改的输入错误。同样的，它提供了对枚举值的填充的相同的帮助功能。

第二，它允许Mule在应用启动时验证你的配置。不像一些其他的基于配置的系统那样，只是无声的忽略它们不认识的元素和属性，Mule会捕获这些错误，这样你就可以修正它们。例如，假设在上面的配置文件中，我们拼写错误了“outbound-endpoint”，一旦应用尝试启动，结果立即就会出现错误：



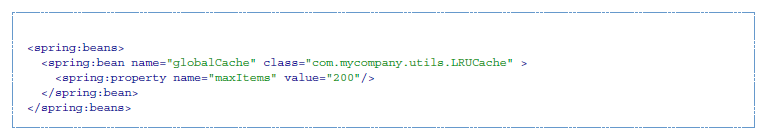
它直接指出了需要修正的那一行。这远比只是简单的忽略问题，让你困扰为什么没有任何输出要有用得多。

#### Spring配置相关

Mule解析配置的方便是嵌入于Spring中的，因此除了定义Mule相关的构造方法，Mule配置中可以做到Spring配置能做的所有事情：创建Sprign Bean，配置list和map，定义属性占位符等等。我们将在下面的章节中进行详细的研究。注意，一如往常，引用合适的schema是必不可少的。

##### Spring Beans

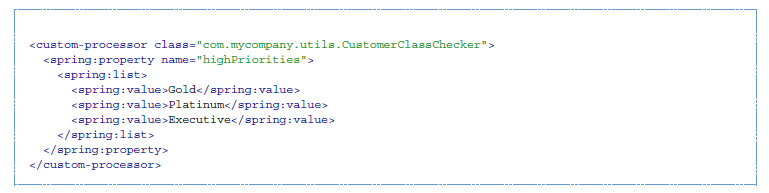
Mule配置中对Spring最简单的使用就是配置Spring bean。这些bean会和Mule相关的对象一起被 放到Mule注册中心里，你可以通过名字查找到任意自定义的Java对象，例如，自定义组件，你可以使用所有Spring的功能来创建它们，例如：



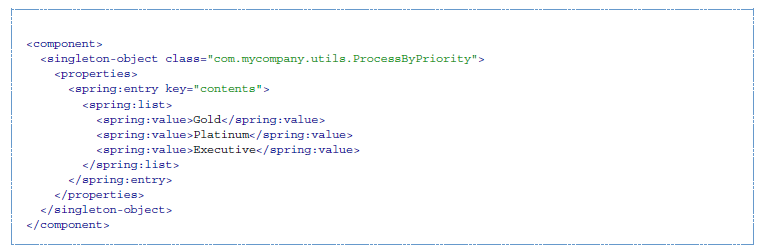
##### Spring配置

在Mule配置中，有很多地方都可以使用自定义的Java对象：自定义的转换器、过滤器、消息处理器等等。所有情况下，都可能要指定一个要实例化的类，和一组Spring属性用于配置这个实例。再说一次，你可以在属性中使用Spring语法的全集，包括list、map等等。

这里是一个例子：



为了在创建Java对象上允许更多的控制，创建自定义组件的语法稍有不同。例如，创建一个单例：



##### 属性占位符

Mule配置可以包含对属性占位符的引用，来保证能够引用到配置文件之外的值。这特性的一个重要的使用场景就是用户名和密码，它们应该使用一种更安全的方式来指定。属性占位符的语法就是简单的：${name}，这里的name就是一个Java属性文件里的一个属性。

这里是一个使用了属性占位符，以及它引用到的属性的例子：

配置文件：



属性文件：



注意这里给出的文件路径是基于classpath路径，另一种选择是使用URL，例如file:///etc/mule/conf/my-mule-app-override.properties。如上所示，也可以指定一组属性文件，它们中间使用逗号分隔。

#### Mule配置相关

##### 全局配置

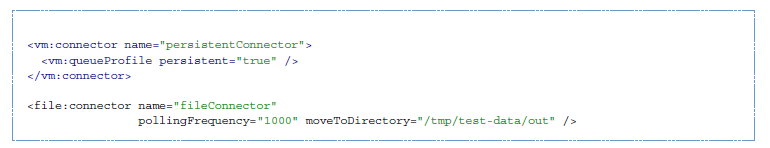
许多Mule元素可以在顶级进行指定，也就是说，作为最外层的mule元素的直接子元素。这些全局元素都有名称，这样才能被使用的地方被引用。注意对全局元素，Mule配置中使用了单一的，扁平的名字空间。任意两个全局元素不能共享同样的名称，即使它们是全然不同类型的东西，比如端点和过滤器。

我们来看一下最常见的全局元素：

###### 连接器

连接器是Mule传输器的具体实例，它的属性描述了传输器是怎样使用的。所有Mule端点使用的连接器都一个继承了连接器属性的相同的传输器。

这里是连接器的例子：



Vm连接器指定了它所有的端点都使用持久队列。文件连接器指定了它的每个端点一秒轮询一次，同时还指定了一旦执行处理，这些文件会被移动到的目录。

注意属性可能被属性指定，也可能被子元素指定。你可以通过查看连接器对应的传输器的参考文档来确定怎么来配置连接器的属性。

端点和它的连接器的关系其实是相当灵活的：

* 如果端点使用名字指定了连接器，它就会使用这个连接器。当然，如果端点和连接器使用了不同传输器，就会出错。
* 如果端点上没有指定连接器，而它的传输器有一个连接器，那端点就使用这个连接器。
* 如果端点上没有指定连接器，并且它的传输器也没有连接器，Mule会为这个传输器的所有端点创建一个默认的连接器供它们使用。
* 如果端点上没有指定连接器，同时它的传输器上有多于一个的连接器，就会出错。

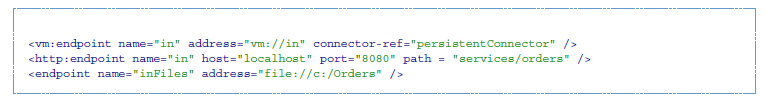
关于连接器和端点的传输器相关的信息，可以参考MULE3USER:Available Transports.

###### 端点

Mule端点是一个消息读出（入站）和写入（出站）的对象，它上面指定了一些属性，定义了这个过程是怎么完成的。端点可以使用两种方式指定：

* 作为全局元素配置的端点称为全局端点。流或者服务中配置的入站或者出站端点，可以使用ref属性引用一个全局端点。
* 流或者服务中配置的入站或者出站端点，可以不引用全局端点，直接进行配置。

全局端点指定了一组属性，包括它的位置。引用了这个全局端点的入站和出站端点都会继承它的属性。这里是几个全局端点的例子：

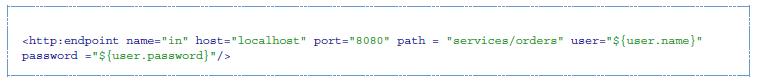


Vm端点指定了它的位置，并引用了上面的连接器。它使用了一个通用的address属性指定它的位置。Http端点使用了默认的http连接器。因为它被明确的配置为http端点，因此它可以使用http相关的属性host、port和path来指定它的位置。File端点指定了它读出（或写入）的目录，它会使用默认的文件连接器。因为它被配置为通用的端点，因此它必须使用address来指定位置。

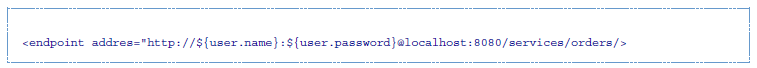
注意每个端点都会使用一种特定的传输器，但它可以通过两种不同的方式指定：

* 如果元素有前缀，它可以使用前缀对应的传输器，
* 如果没有前缀，前缀就会通过address属性来确定。

前缀风格的更值得推荐，尤其是位置配置会比较复杂时，比较一下



和

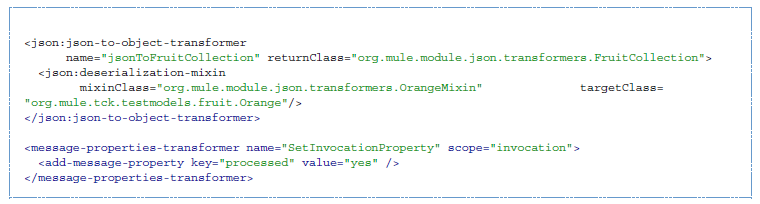


端点上面一个最重要的属性是它的消息交换方式（简称MEP），也就是说，消息是不是单向的或者请求返回响应的。这个可以在多个层次上指定：

* 一些传输器只支持一种MEP。例如，imap是单向的，因为当它读了一封email时，没有任何响应会被发送出去。另一种，servlet，总是请求-响应类型的。
* 每种传输器有一种默认的MEP。JMS默认是单向的，因为JMS消息通常不会和响应关联起来。http默认是请求-响应的，因为HTTP协议天生的对每个请求都有一个响应。
* 端点上可以定义MEP，尽管，只有MRP对它们的传输器合法，才能被接受。

###### 转换器

转换器是一个传输当前Mule消息的对象。Mule核心定义了转换器的基本集合，许多模块和传输器中定义了更多的转换器，例如JSON模块定义了的能够在对象和JSON间相互转换的转换器，而Email传输器定义了在二进行数组和MIME消息间转换的转换器。每种类型的转换器都通过定义XML配置定义了它们的属性。这里是一个转换器的例子：



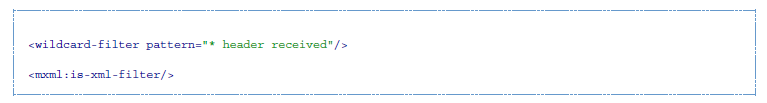
上面的转换器转换消息到JSON，指定了特定的处理过程来处理对org.mule.tck.testmodels. fruit.Orange类的转换。下面的转换器负责添加一个invocation-scoped属性到当前的消息上。

像端点一样，转换器可以被配置成全局元素，被使用到的地方引用，或者也可以在使用点上进行直接配置。

更多Mule转换器的内容，参考Using Transformers。

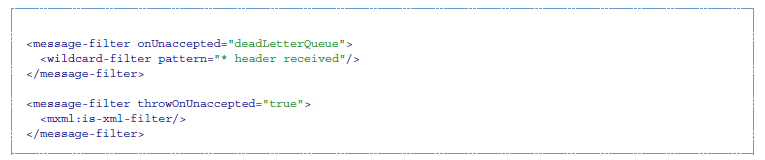
###### 过滤器

过滤器是确定消息是不是应该被处理的对象。和转换器一样，Mule核心定义了过滤器的基本集合，许多模块和传输器定义了更丰富的过滤器。这里是过滤器的一些例子：



上面的过滤器保证了只有当前消息匹配指定的模式时，才能被继续处理。下面的的过滤器只有当消息是XML文档时，才会继续处理。

还有几个搞定了其他的过滤器功能的特别的过滤器，第一个是message-filter：



如上所示，只有匹配指定的模式时，当前消息才会被继续处理。但是现在所有没有匹配的消息，不是被丢弃，而是被发送到dead letter队列来继续处理。在这里，只有当消息是XML文档，才会被处理，否则抛出异常。

其他特殊的过滤器是and-filter、or-filter和not-filter，它们可以让你将过滤器组合成逻辑表达式：



当消息是优先级1，或者消息是优先级2，并且来自加拿大以外的国家时，才会被处理。

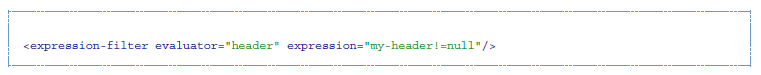
过滤器可以被配置成全局元素，被使用到的地方引用，或者也可以在使用点上进行直接配置。更多Mule过滤器的内容，参考Using Filters。

###### 表达式

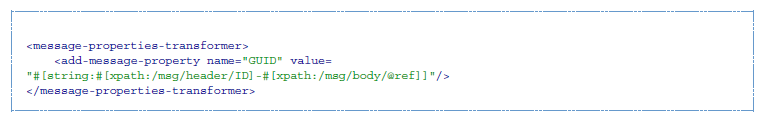
Mule有强大的表达式功能，允许基于系统不同部分的信息来影响消息的处理。因为Mule的许多不同部分可以判定表达式，指定一个表达式需要两个部分：

* 计算器，用来计算表达式。Mule提供了很多的计算器，同时你也可以添加你自己的。
* 表达式，用于被计算。表达式的语法是和计算器相关的。

根据表达式使用的位置不同，有两种方式指定表达式。常用的，基于表达式的元素，例如表达式转换器、表达式过滤器，基于表达式的路由器，例如表达式消息切分器，它们都有特定的表达式属性，计算器和自定义计算器。例如：

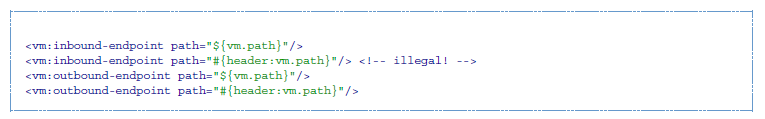


当使用string值替代表达式时，你要使用语法#[evaluator.expression]，例如：



这一表达式调用：首先xpath表达式被使用了两次从当前消息中抽取数据，然后调用string计算器，通过这些数据和连字符来构建出一个string。

表达式和属性占位符可能看起来很像，但是它们是完全不同的。属性占位符会在配置期会替换，用来构建静态的信息。表达式在运行时被替换，因此所有使用到它们的都是动态的。看一下下面的例子：



第一个是好的——它在配置期确定端点的位置。（当然，vm.path属性必须要设置）。

第二个是不合法的。入站端点的地址在配置期必须要设置，在配置被构建前，表达式是没有办法计算的。

第三个和第一个是很像的。

第四个是一个新的配置——动态端点。端点要发消息到的地点是在消息的处理过程中才确定的，可能每次都会不同。当然，没有配置vm.path属性的消息会导致出错。更多Mule表达式的内容，参考Using Expressions。

###### 名字和引用

正如我们看到的，许多Mule对象可以被全局定义。这样的好处是通过在都要好好的 地方引用它们，可以做到在整个应用中复用。对于这种使用，有一个通用的模式：

* 全局对象使用name属性来标识它的名字
* 它可能使用ref属性被引用

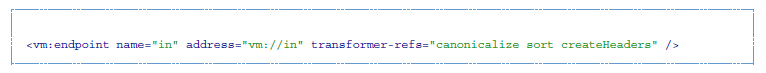
对每种类型的对象，都有一种泛型的元素用来引用它。

* 所有全局转换器都用transformer元素引用
* 所有全局的消息处理器都用processor元素进行引用
* 所有全局端点都用inbound-endpoint和outbound-endpoint进行引用
* 所有全局过滤器都用filter元素进行引用

例如：



另外，也有一些情况下，全局对象的名字是作为属性值来使用的，例如：



##### 流

流是Mule处理的基本单元。一个流从一个入站端点开始，从其中读取消息后，继续经常一系列的消息处理器，然后可选的终止于出站端点，从这里整个被处理后的消息被发送出去。我们已经见过一些消息处理器了：转换器和过滤器。其他类型的包括使用像Java或Groovy语言处理消息的组件，调用云服务的云连接器，和按需要选择消息流的路由器。下面是简单的流，我们将参考它来了解所有部分。



正如其名，这一个流是用来接收和处理订单的。当我们浏览时，记一下流的配置是怎么映射到它描述的逻辑上的：

消息从HTTP端口中被读取出来。

消息被转换成String。

将String用作Key来从数据库中查找订单列表。

订单被转换为XML格式。

如果订单没有准备好要被处理，跳过它。

列表可以选择被记录日志，以用于调试。

列表中的每一个订单被切分成单独的消息。

消息放大器在消息上添加一些信息

调用Authorize.net来授权订单。

订单上的邮箱地址保存备用。

调用Java组件处理订单。

调用另一个叫做processOrder的流处理订单。

processOrder的确认邮件发送订单的地址。

如果处理订单中导致了异常，异常处理策略就会被调用：

所有链上的消息处理器都被调用处理异常。

首先，消息被转换成字符器

最后，字符串被放入到错误队列中，等待人工处理。

这一个流中的任一步都在下面会被详细描述，通过构造进行组织。

###### 端点

之前，我们研究了全局端点的声明。现在我们看一下流里面的端点，它们是用于接收（入站）和发送（出站）消息的。入站端点只在流的开始出现，在这个位置它们供给了要被处理的消息。出站端点在后面的任意位置都可以出现。穿过流的消息的路径取决于它的端点的MEP：

* 如果入站端点是请求-响应方式，流就会在它中止时，返回当前消息给它的调用者。
* 如果入站端点是单向的，流就会在它中止时，直接退出。
* 当流到达了一个请求-响应类型的出站端点时，它会将当前消息发送给这一端点，然后等待响应，并将响应作为当前消息。
* 当流到达一个单向的出站端点时，它会将当前消息发送给这一端点，然后继续处理当前消息。

这里的例子中通过HTTP连接接收消息。消息负载被设置到接收到的字符数组中，HTTP头变成了入站消息属性。因为这个端点是请求-响应（HTTP默认）式的，在流的最后，当前消息会被返回给调用者。

接下来用消息作为参数调用了一个JDBC查询，使用查询结果替换掉当前消息。因为这一端点是请求-响应式的，查询结果会变成当前消息。

从分支流中返回的订单完成确认被邮寄出去。注意我们使用了之前被保存到了消息属性中的邮件地址。因为这一端点是单向的（邮件传输器的惟一MEP），当前消息不会被修改。

所有没有被正确处理的订单都会被放入JMS队列中等待人工检查。因为这一端点是单向的（JMS默认），当前消息不会被修改。

结果，在成功的情况下，最后发送回调用者的消息将会是确认消息。否则失败时，相同的字符串会被发送到JMS错误队列。

###### 转换器

如前面所述，转换器会修改当前消息。这里有几个例子。注意它们是在被使用的位置定义的。然后同样可以全局定义，然后在使用的位置进行引用。

上例中，字节数组被转换为字符串，来保证可以作为key进行数据库查询。

从数据库中读取的订单会转换成XML文档。

邮件地址被存到消息属性中。注意，不同于大部分转换器，消息属性转换器不会影响到消息负载，只是修改了它的属性。

导致了异常的消息被转换成字符串。注意由于使用相同的策略处理所有异常，所以我们不知道消息在这个位置会是哪种对象。它可能是字节数组，字符串，或者XML文档。转换所有这些消息成字符串，保证接收者知道需要什么。

###### 消息放大器

消息放大器使用enricher元素进行配置。不同于能够修改消息负载的消息转换器，放大器向消息添加额外的属性。这允许流逐步积聚起一批用于后续处理的信息。更多放大器的信息参见Message Enricher。

例子中，放大器调用云连接器来获得了信息，并存到消息属性内。因为云连接器在放大器内被调用，因此，它的返回被放大器处理，而不是变成消息。

###### 日志记录器

Logger元素允许从流中输出调用信息。更多logger的信息参见Logger Element for Flows。

例子中，每个从数据库中取出的订被输出出来，但仅在DEBUG模式启用时。这意味着这个流通常是不输出信息的，但在需要时可以很容易地启用调用。

###### 过滤器

过滤器确定了消息是否被处理。

例子中，如果取出的文档的状态不是“ready”，处理就会跳过。

###### 路由器

路由器修改了消息的流。在其他可能的情况下，它可能在多处消息处理器中选择，切分一条消息成多条，合并多条消息到一条。更多路由器的信息参考Routing Message Processors。

例子中，将从数据库中读取的文档在order元素处分切成多个订单。结果是零或多个订单，其中每个都会经常流的其余部分处理。也就是说，对接收到的每个HTTP消息，经过splitter的处理的单向的。而余下的部分可能被处理零次，一次或者多次，这取决于文档中包含了多少订单。

###### 组件

组件是使用Java、Groovy或者其他语言编写的消息处理器。Mule通过查找与消息类型最佳的匹配，来决定调用组件的哪个方法。例如，如果消息是一个InputStream，它会寻找接收一个InputStream类型（或者Stream，或者Object）的公开方法。为了订制这种搜索，Mule使用一种叫做入口解决器的对象，它们是配置在组件上的。这里是一些例子：



这个例子让preProcessXMLOrder和preProcessTextOrder变成了候选者。Mule使用消息的类型，执行反射进行选择。

调用消息属性中名字为methodToCall的方法。

即使它没有参数，也会调用generate方法。

入口解决器是高级使用方式。几乎任何情况下，Mule都可以在没有任何特别引导下找到正确的方法来调用。

这是一个Java组件，它通过类名进行指定，在处理当前消息时进行调用。在这个例子中，它会预处理消息。更多入口解决器的信息，参考Entry Point Resolver Configuration Reference。

###### 云连接器

一个云连接器会调用一个云服务。

上例中，它调用authorize.net来授权信用卡购买，这是通过从消息中发送给它信息实现的。更多云连接器的信息，参考Integrating with Cloud Connect。

###### 处理器链

处理器链是一组消息处理器，它们将会按顺序被执行。它允许你在一个配置中使用多于一个的处理器，不然就只允许一个。就像是把多条Java语句放到大括号里面。

例子中，它用于执行异常策略的两步。首先转换消息，然后邮件发送。

###### 分支流

分支流是一种能够被另一个流调用的流。它表示了一个可重用的处理步骤。调用它非常像调用Java方法，将当前消息传入分支流，当它返回时，调用的流恢复处理这个由分支流返回的消息。

例子中，调用它用来处理已经被处理过的订单，产生一条确认消息。

###### 异常策略

当异常产生在异常策略的作用域内时，会被调用，就非常像Java的异常处理器。它可以定义对任何待定的事务做什么处理，以及异常对流是不是致命，而且还包括处理异常的逻辑。

例子中，它将导致异常的消息写入JMS队列，队列会被检查。更多关于异常策略的信息，参考MULE3USER:Exception Handling。

##### 配置模式

流有强大和灵活的优势。Mule可以做的任何事情都可以被放入流。Mule还可以使用配置模式，它们就是设计用于简化Mule的常用配置的。熟悉这些模式，并在可能的情况下使用它们，是非常有价值的。这和你应该使用一个类库，而不是从头写相同的功能是一样的道理。现在Mule支持四种配置模式：

* Pattern:bridge 建立了入站端点和出站端点之间的桥梁。
* Pattern:simple-service 是从一个入站端点到组件之间的简单流。
* Pattern:validator 它像单身的bridge，不过它会在发送消息到出站端点前进行验证消息。
* Pattern:web-service-proxy 是一个Web Service代理。

从Mule 3.1.1开始，如上所示它们都在pattern名字空间里。在Mule 3早一些的版本中，它们是在core名字空间中的，除了web-service-proxy，那时是ws:proxy。这些老的名字在Mule 3.1.x的版本中将继续可用，不过之后会删除掉。

###### 通用特性

为了灵活性，所有这些模式都允许端口通过多种方式进行指定：

* 本地端点可以作为子元素声明，就像在流里面一样。
* 对全局端点的引用可以作为子元素声明，就像在流里面一样。
* 对全局端点的引用可以作为属性inboundEndpoint-ref和outboundEndpoint-ref的值声明。
* 端点的地址可以使用属性inboundAddress和outboundAddress的值指定。

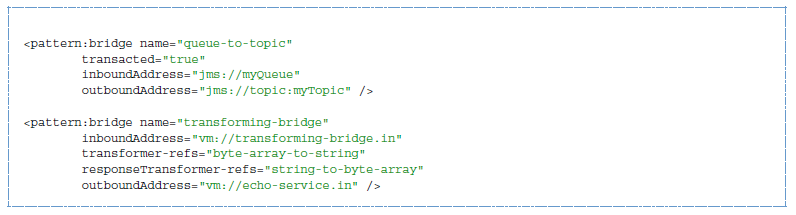
所有配置模式都可以指定异常策略，就像流上一样。

###### 桥

该模式允许你除入站和出站端点之外，还可以配置，

* 一组应用于请求上的转换器
* 一组应用于响应上的转换器
* 是不是在一个事务里处理消息

这里是一些例子：



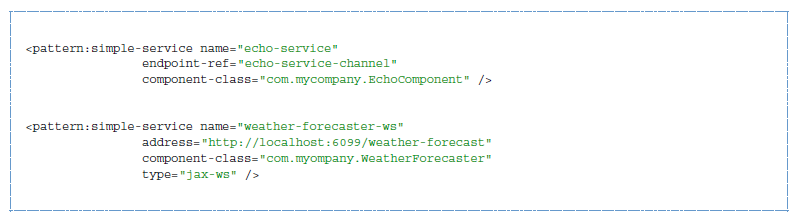
上例中，Mule使用事务从JMS队列拷贝消息到JMS主题。从一个vm入站端点读取字节数组，将它们转换为字符串，然后写到vm出站端点中。响应是字符串的，它们会被转换成字节数组，然后被写到出站端点中。

###### 简单服务

该模式允许你除入站端点外，还可以配置：

* 一组应用于请求上的转换器
* 一组应用于响应上的转换器
* 一个组件
* 一个组件类型，让你可以使用Jersey和CXF组件

这里是一些例子：



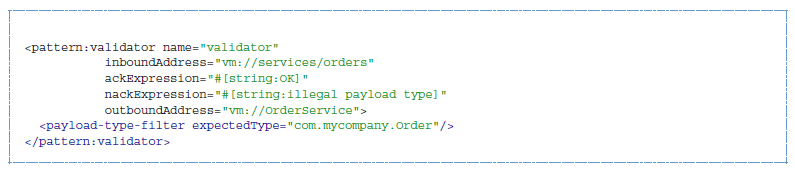
这里是一个显示请求的简单服务。它使用了一个CXF组件。注意创建这个组件只需要如此少的配置。

###### 校验器

该模式允许你除入站和出站端点外，还可以配置：

* 一组应用于请求上的转换器
* 一组应用于响应上的转换器
* 一个执行校验的过滤器
* 创建响应的表达式，用于表明验证成功或失败

这里是一个例子：



上例中在调用其他服务前，验证消息负载是不是正确的类型。

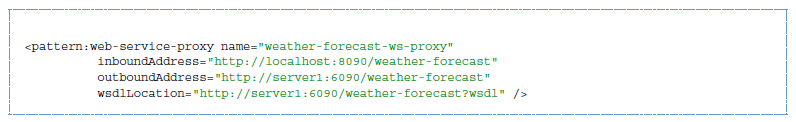
###### Web Service代理

该模式会创建一个Web Service代理。它通过修改已经发布的WSDL，来包含代理的URL。

可以让你除入站和出站端点外，还可以配置：

* 一组应用于请求上的转换器
* 一组应用于响应上的转换器
* 服务WSDL的位置，用URL或者文件名

这里是一个例子：



上例创建了位于server1上的天气预报服务的代理。

更多配置模式相关的信息，参考Using Mule Configuration Patterns。

##### 服务

服务是Mule很老的特性。它们既不像流一个灵活，也不能像配置模式一样友好。尽管服务仍然是完全支持的，但我们推荐新的开发使用流和模式来完成。前面已经提到，服务使用了许多和流相同的理念，并且不难使用和构建。

一个服务分为三个部分：

* 入站。这包含了一个入站端点，加上服务允许的单一组件的任意前置处理过程。它可以由入站路由器、转换器、过滤器和其他消息处理器组成。
* 组件。这个和流中出现的组件相同。它是可选的。
* 出站。这是所有组件之后的处理过程。它由一组出站路由器组成。这里最简单的是pass-through路由器，它只是简单的把消息传递给出站端点。

服务，像流和模式一样，可以定义异常策略。

服务是放置在一个叫做模型的构件内的。构件负责聚合服务，并让它们共享一些配置：

* 异常策略
* 组件的入口解决器

把这里所有的组合起来，我们可以比较一下，使用流、配置模式和服务来创建一个拷贝标准输出流的简单应用。



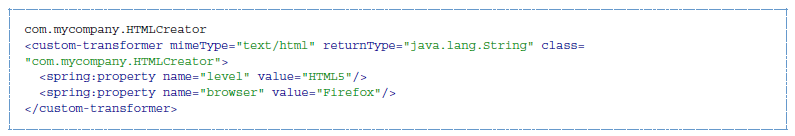
桥模式是最简单的，流也没有太落后。服务方式不难读懂，但是它明显的又长又复杂。更多服务的信息，参考Service Configuration Reference。

##### 自定义元素

Mule是可扩展的，这意味着你可以创建你自己的对象（通常是通过扩展Mule的类）。在你完成了这个以后，可以通过标准的方式将它们放置到配置文件中。假设，例如你创建了com.mycompany.HTMLCreater，它负责转换很多种文档到HTML。它是一个Spring bean：

* 有一个默认的构造器
* 它通过设置bean属性进行定制

现在你可以通过使用custom-transformer元素将它放到你的配置中：



注意标准的转换器的Mule属性是通过通常的方式指定的。惟一不同的是对象是通过类名和Spring属性来创建的，而不是通过基于schema的元和属性。每种类型的Mule对象都有一个用于自定义扩展的元素：

* 用于连接器的custom-connector
* 用于入口解决器的custom-entry-point-resolver
* 用于异常策略的custom-exception-strategy
* 用于过滤器的custom-filter
* 用于消息处理器的custom-processor
* 用于路由器的custom-router
* 用于转换器的custom-transformer

##### 系统级配置

配置包含了多个全局配置，这会影响到整个的mule应用。所有都是mule顶级子元素configuration元素的子元素。它们都属于两个组内：线程监测和超时。

###### 线程监测

线程监测确定了Mule怎样管理线程池。在大部分场景下，默认设置会运行良好，但如果你确信，例如，你的端点正在接收非常大的流量，以至于它们需要额外的线程来处理，你就可以通过修改默认设置来调整它。这可以是选择的端点，也可以是对所有端点。可以被调整的设置所对应的元素如下：

* 所有线程池default-threading-profile
* 对所有转发消息的线程池default-dispatcher-threading-profile
* 对所有接收消息的线程池 default-receiver-threading-profile
* 所有处理服务的线程池 default-service-threading-profile

###### 超时

同样，默认超时时间通常工作良好，但是如果你想调整它们，你可以针对每个或者全局调整。可以被调整的超时所对应的属性如下：

* 对同步响应等待多长时间，以毫秒计，默认是10s，defaultResponseTimeout
* 对事务完成等待多长时间，以毫秒计，默认是30s，defaultTransactionTimeout
* 对Mule优雅地关闭要等待多长时间，以毫秒计，默认是5s，shutdownTimeout

##### 管理器

Mule有多个全局对象，用于管理系统级设施。它们将在下面进行讨论。

###### 事务管理器

Mule使用JTA来管理 XA事务。因此，要使用XA 事务，必须要有一个JTA事务管理器，并在配置中指定。Mule对其中的许多有显式的配置，同样，也允许你使用自定义的管理器。用于配置事务管理器的元素是mule的直接子元素。

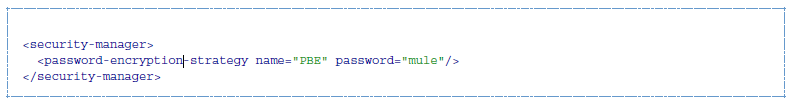
* WebSphere事务管理器：websphere-transaction-manager
* JBoss事务管理器：jboss-transaction-manager
* WebLogic事务管理器：\*weblogic-transaction-manager
* JRun事务管理器：jrun-transaction-manager
* Resin事务管理器：resin-transaction-manager
* 在JNDI中查找事务管理器：\*jndi-transaction-manager
* 自定义查找事务管理器：\*custom-transaction-manager

加\*的事务管理器，在执行查找前允许你配置JNDI环境。更多关于事务管理器的信息，参考Transaction Management。

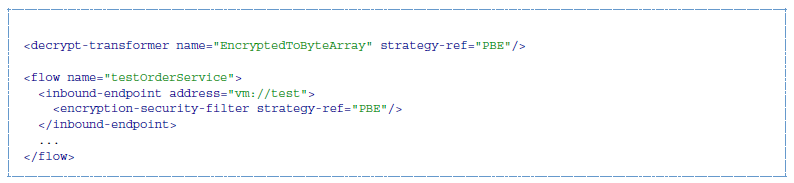
###### 安全管理器

Mule安全管理器可以使用一个或多个加密策略进行配置，然后可以用于加密转换器、安全过滤器或者像SSL或HTTPS这样的安全传输器中。这些加密策略可以大大减化安全管理器的配置，因为它们可以在组件间共享。安全管理器使用全局security-manager元素进行设置，它是mule的直接子元素。

例如，这里是一个基于密码的加密策略（PBE），它提供了使用JCE的密码加密。用户必须要指定密码，并且还可以选择一个salt和迭代次数。默认的算法是使用MD5和DES的PBE，但用户可以指定任意被JCE支持的算法。



这个加密策略可以在系统中被其他组件引用，比如过滤器或转换器。



更多Mule安全的信息，参考Configuring Security。

###### 通知管理器

Mule可以在消息发送，接收或者处理时发送通知。你必须要注册一些对象来接收它们，这样通知才能真正被创建和发送。这是通过全局的notifications元素来未完成的，这是一个mule的直接子元素。它允许你指定一个对象来接收通知，也可能指定哪一个通知器来发送它。注意一个对象只能接收它实现了对应的接口的通知（这些接口定义在org.mule.api.context. notification.package）。这里是一个例子：



假设ComponentMessageNotificationLogger实现了ComponentMessageNotificationListener接口， EndpointMessageNotificationLogger实现了EndpointMessageNotificationListener。

Mule创建了监听器bean，并且给两个组件和端口通知都注册了这两个bean。但因为ComponentMessageNotificationLogger只实现了组件通知的接口，只有这种才是它能接收到的（相同，另一个EndpointMessageNotificationLogger也一样）。

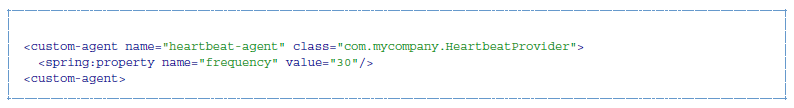
更多通知相关的信息，参考Notifications Configuration Reference。

##### 代理

Mule允许你定义代理来扩展Mule的功能。Mule会管理代理的生命周期（初始化它们，在启动时启动它们，在关闭时停止和销毁它们）。这些代理可以做任何事情，惟一要求是它们要实现org.mule.api.agent.Agent，这保证Mule能管理它们。更多Mule代理的信息，参考Mule Agents。

###### 自定义代理

简单的声明全局的custom-agent元素，就可以创建自定义代理，它是mule的直接子元素。代理是Sprign bean，所以一样它需要类名和一组Spring属性来完成配置。另外，它需要一个名字，Mule用它在日志输出中标识代理。这里是一个例子：



上例创建了一个代理，进行每30s一次的心跳。由于Mule会启动和停止它，心跳会在Mule服务器运行过程中准确的进行。

###### 管理代理

Mule在管理名字空间中实现了多种管理代理。

* Management:jmx-server创建JMX服务器，允许对Mule的JMX bean进行本地和远程访问
* Management:jmx-mx4j-adapter创建一个服务，允许对JMX bean使用HTTP访问
* Management:rmi-server 创建一个服务，允许对JMX bean使用RMI访问
* Management:jmx-notifications创建一个代理，发送Mule通知到JMX
* Management:jmx-log4j允许JMX管理Mule对Log4J的使用
* Management:jmx-default-config保证同时创建以上的部分
* Management:log4j-notifications创建一个代理，发送Mule通知到Log4J
* Management:chainsaw-notifications创建一个代理，发送Mule通知到Chainsaw
* Management:publish-notifications创建一个代理，发布Mule通知到Mule出站端点
* Management:yourkit-profiler创建一个代理，暴露YourKit监测信息到JMX