Apache Camel是一个**集成框架(Integration Framework)**。它的核心是一个路由引擎(Routing Engine)，更准确的说是一个路由引擎构造器。你可以使用它定义自己的路由规则，决定从哪里(Source)接收信息(Message)，如何处理(process)该信息，将处理过的信息发送到何处(Destination)。

Camel支持80多种通信协议和数据格式，它不是ESB(Enterprise Service Bus)。

**信息模型**：

org.apache.camel.Message：含有被搬运、路由数据的实体

org.apache.camel.Exchange：信息交换的抽象，含有输入信息和输出信息

信息组成：

Headers，Body，Attchments(可选)，每个信息都有一个字符串类型的唯一标识，格式不定，由信息提供者生成，如无，默认使用UUID。Headers和其他信息的头部信息类似，不过它的值类型是Object。attachments的典型用处在WebService和Email。Body部分的类型也是Object，这意味着你可以传送任意类型的内容，当信息发送者和接收者使用不同的信息格式时，需要进行转换。

Exchange可以看做信息在路由时的容器。它提供多种不同类型的信息交互(MEPs，Message Exchange Patterns)，交换模式有单路(one-way，比如JMS，只有输入)，请求-相应(request-response，比如HTTP，有InOut)。

Exchange组成：

Exchange ID，唯一标识符；MEP，属于inOnly还是inOut；Exception，路由过程中如发生异常，存储在此；Properties，相当于全局的Headers；In Message和out Message，如果MEP为inOnly，则没有out Message。

**生产者和消费者**

生产者创建并发送信息到端点(EndPoint)，它将信息包装在Exchange中。

消费者，接收生产者发送过来的信息，并将之送到Processer处理。有两类消费者，事件驱动(Event-Driven)和拉取(polling)。事件驱动的消费者多见于C/S系统和WebService，消费者监听某事件，等待客户端发送信息，当接收到相关事件时苏醒并处理。拉取类型的消费者，多见于文件、FTP、Email等。该种类型下，消费者多使用周期轮询的方式主动获取信息。

**路由和EIPs**：

基于内容的路由(CBR，Content-Based Router)，使用choice方法。如下：

from(“jms:order”).choice().when(条件).to(“jms:xmlOrders”).when(条件).to(“jms:csvOrders”)

过滤器(filter)，将不符合条件的信息过滤掉。如下：

from(“jms:order”).filter(xpath("/order[not(@test)]")).to(...)

广播(multicast)，设置多个接收者。使用multicast方法。如下：

from(“jms:order”).multicast().parallelProcessing().to(“jms:queue1”,”jms:queue2”)

接收者列表(recipient list)，动态设置接收者。比如在信息头部设置，如下：

from(“jms:order”).recipientList(header(“recipients”))

**数据(信息)转换**

分为两类：格式(format)转换和类型(type)转换，比如从CSV转为XML，从String转为JMS的Message。Camel提供了6种方法来进行格式转换。

使用EIPs和Java代码：①使用Processor，从Exchange中获取Message的Body，手动转换，再设置回去。②使用Bean，相当于一个自定义的类型转换器，有一些语法限制。③使用transform方法。如下：

from(“direct:order”).transform(new Expression()...)

使用Content Enricher EIPs：pollEnrich，使用消费者从另一个源处合并数据；enrich，使用生产者从另一个源处合并数据。简单区分生产者和消费者：写为生产，读为消费。Camel使用AggregationStrategy接口来合并数据。参考Camel In Action第104页。

XML转换：XLST和[un]marshaling。XLST是一个Component，使用方法和其他Component一样。如下：

from("file://rider/inbox").to("xslt://camelinaction/transform.xsl").to("activemq:transformed")

可使用XStream或JAXB来进行Object和XML之间的转换。如下：

from("direct:foo").marshal().xstream().to("activemq:transformed")

在Camel中，每种数据格式都实现org.apache.camel.spi.DataFormat接口，它包含两个方法：marshal，将一种格式转换为另一种格式，如将Java Object转换为XML；unmarshal，和前者相反。

Camel支持模板，比如Velocity和Freemarker。

类型转换器(Type Converter)

和Spring的类型转换器类似。

**使用Bean**

为了减少对Camel API的依赖，建议使用Bean。比如，用Bean替换Processor。

Service Activator pattern：

在请求方和Service方中间的一个协调者，将请求方的Request转换为适配Service的Request。在Camel中，担任这个角色的是BeanProcessor类。

Camel使用注册表(Registry)来定位Beans，它只是一个SPI(Service Provider Interface)，真正的查找动作委托给比如Spring的ApplicationContext等的可插拔的组件。一般使用如下API来获取Bean：

HelloBean hello = camelContext.getRegistry().lookup("helloBean", HelloBean.class);

当Camel与Spring集成时，自动使用ApplicationContextRegistry，即只要把Bean配置成为普通的SpringBean就行了。

BeanProcessor的处理步骤：①在注册表中查找Bean；②选择Bean中要调用的方法；③绑定方法的输入参数；④调用方法；⑤处理发生的异常；⑥将方法的返回值(如果有)设置到exchange的out message的body中。

关于第②点的解决方法中的重要事项：匹配方法时，方法名和参数都要匹配；exchange的in message的body会尝试转换为方法的第一个参数类型以进行匹配；尽可能在route中指定方法名。

关于第③点：方法第一个参数尽可能与message的body绑定。此后参数，可选Camel内部元素，比如exchange等。使用@Header、@Body之类的注解进行绑定。可与SpringMVC的方法参数绑定进行比较。

**错误处理：Error Handling**

错误分为可恢复的错误和不可恢复的错误。对于可恢复的错误，我们可以提供重试。对于前者，可以从org.apache.camel.Exchange中获取和设置，使用普通的getter/setter方法。对于后者，可在Exchange中设置Fault(故障)并提供故障信息，代码如下：

Message msg = exchange.getOut(); msg.setFault(true); msg.setBody(“Unknown Custom”);

Router中任何异常都会被Exchange捕获并setException(e)，这样Camel的错误处理机制Error Handler就能根据错误信息来进行相应的处理，重试、将错误信息返回给调用者、发送故障信息。

先了解下Errorhandler所处的位置。在Route中，每两个节点(node，比如consumer和processor)之间都会有一个Channel，从而保证对route过程的管理和监控。

Camel提供了几种Error Handler：①DefaultErrorHandler，不重试，将错误信息返回给调用者；②DeadLetterChannel(死信通道)，发生错误时，将Message(包括原始Message)和错误信息发送到一个名为死信的队列，不影响后续Message的发送；③TransactionaErrorHandler，与事务相关。④NoErrorHandler，禁用Error Handler；⑤LoggingErrorHandler，顾名思义，将error记录为error级别的日志。前三者都继承了RedeliveryErrorHandler，它们有一些共有的特性：1、重发策略，包括最大重试数，重试间隔等等；2、作用域，可设置为Camel上下文级别和当前route级别；3、异常策略，对不同的异常采取不同的动作；4、error handler是否处理当前error，如不处理，将错误返回给调用者。