Apache Camel DSL in Scala

willem.jiang@gmail.com

2013.4.25

关于我

- 06年参与Apache中间件项目开发
  - Apache CXF
  - Apache Camel
  - Apache ServiceMix
- 目前就职于红帽软件JBoss Team
- 微博id willemjiang

何为DSL?

何为DSL?

何为Camel DSL?

何为DSL?

何为Camel DSL?

为何要用Scala实现Camel DSL?

DSL

Digital Subscriber Line

DSL

DSL

- Domain-specific language

DSL

- Domain-specific language
- A type of programming language in software development

DSL

- Domain-specific language
- A type of programming language in software development
- Domain engineering dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.

DSL

- Domain-specific language
- A type of programming language in software development
- Domain engineering dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.

DSL 是提升开发效率的利器

有人写过系统互联程序吗?

有人写过系统互联程序吗?

大概要花多长时间实现?

有人写过系统互联程序吗?

大概要花多长时间实现?

利用框架来提高开发效率!

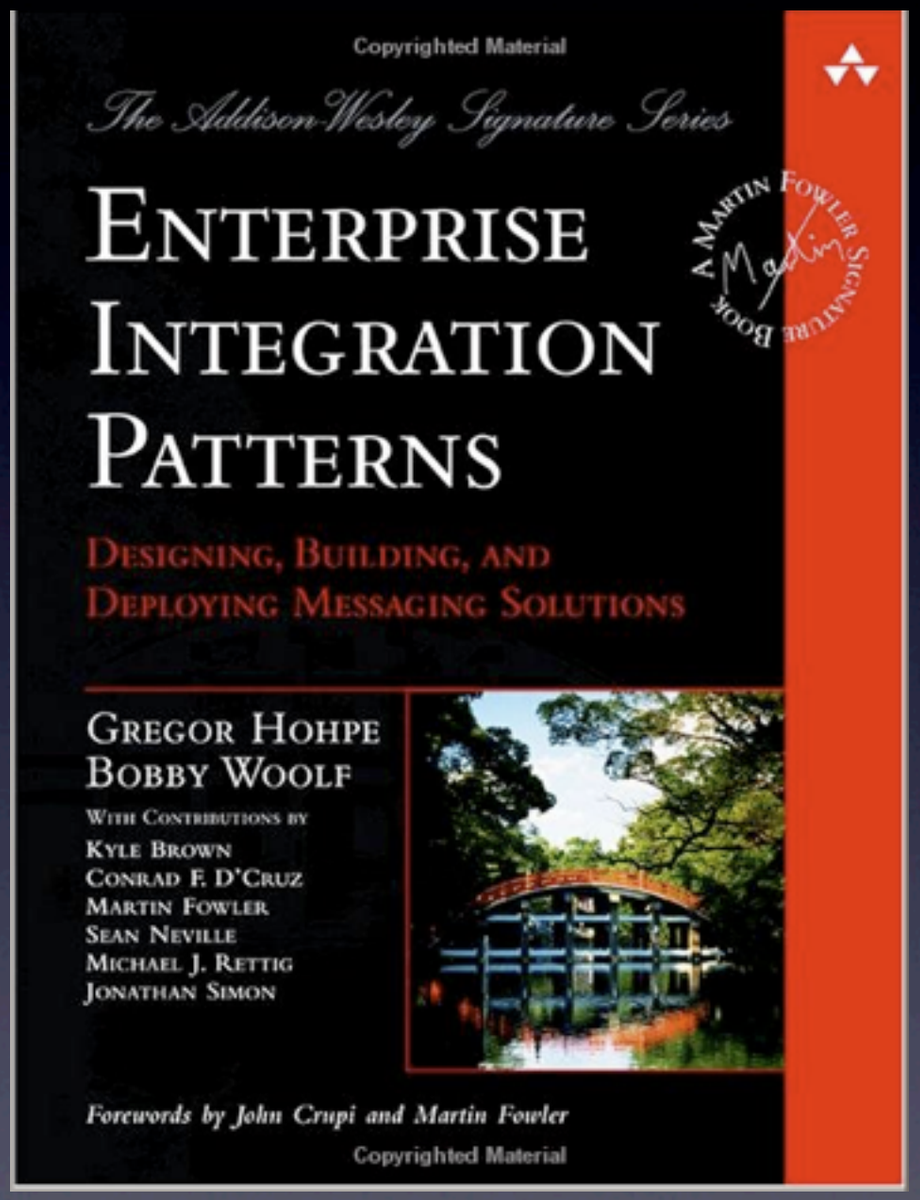
Camel又是什么呢?

Camel又是什么呢?

Camel 是一个建立在企业集成模式(EIP)基础之上的开源集成框架。

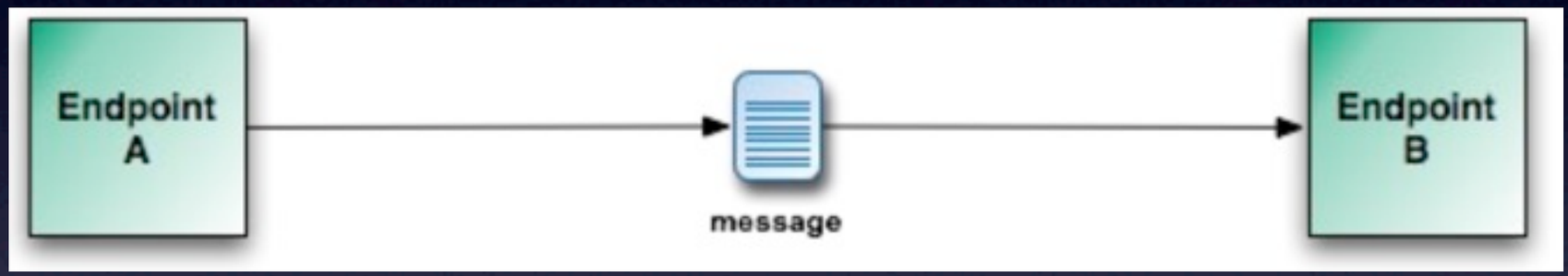
Camel是什么？

Book by Gregor & Bobby!

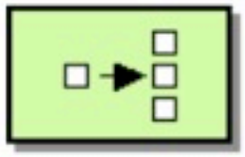
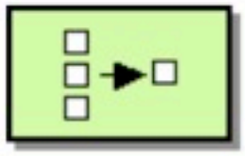
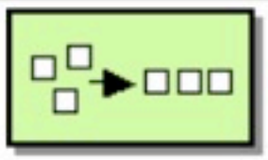
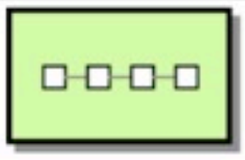


http://camel.apache.org/enterprise-integration-patterns.html

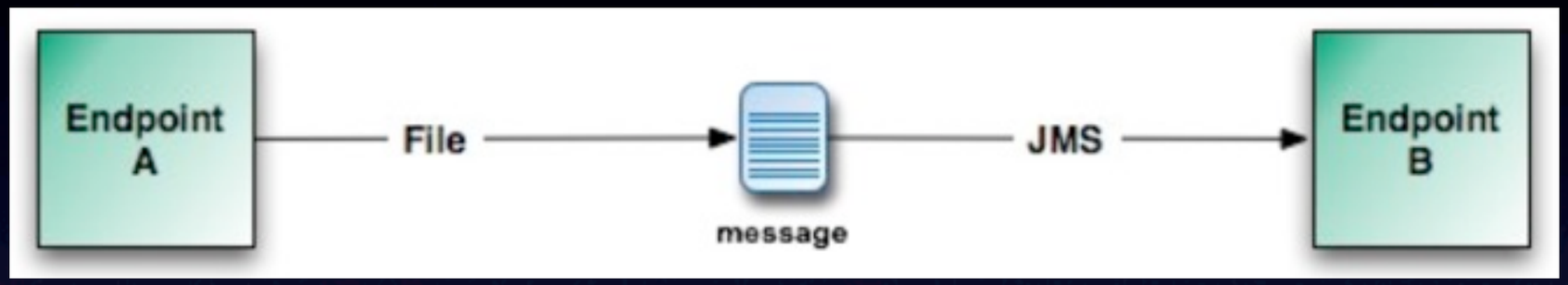
消息路由



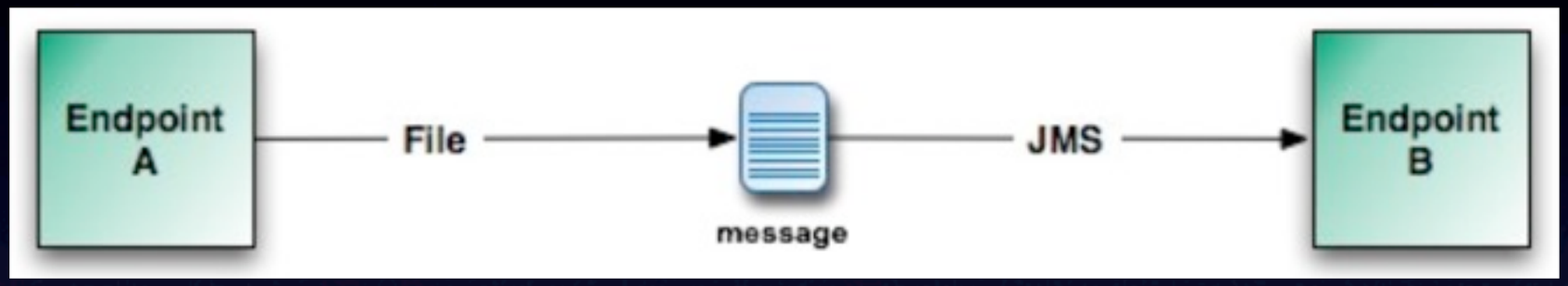
EIP中的消息路由

	Content Based Router	How do we handle a situation where the implementation of a single logical function (e.g., inventory check) is spread across multiple physical systems?
	Message Filter	How can a component avoid receiving uninteresting messages?
	Dynamic Router	How can you avoid the dependency of the router on all possible destinations while maintaining its efficiency?
	Recipient List	How do we route a message to a list of dynamically specified recipients?
	Splitter	How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
	Aggregator	How do we combine the results of individual, but related messages so that they can be processed as a whole?
	Resequencer	How can we get a stream of related but out-of-sequence messages back into the correct order?
	Routing Slip	How do we route a message consecutively through a series of processing steps when the sequence of steps is not known at design-time and may vary for each message?
	Throttler	How can I throttle messages to ensure that a specific endpoint does not get overloaded, or we don't exceed an agreed SLA with some external service?
	Delayer	How can I delay the sending of a message?
	Load Balancer	How can I balance load across a number of endpoints?
	Multicast	How can I route a message to a number of endpoints at the same time?
	Loop	How can I repeat processing a message in a loop?

一个简单的路由规则

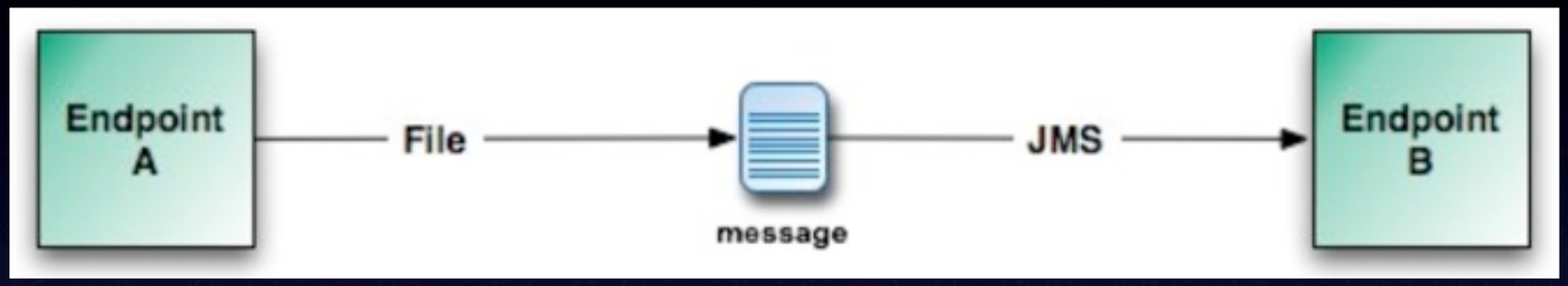


一个简单的路由规则



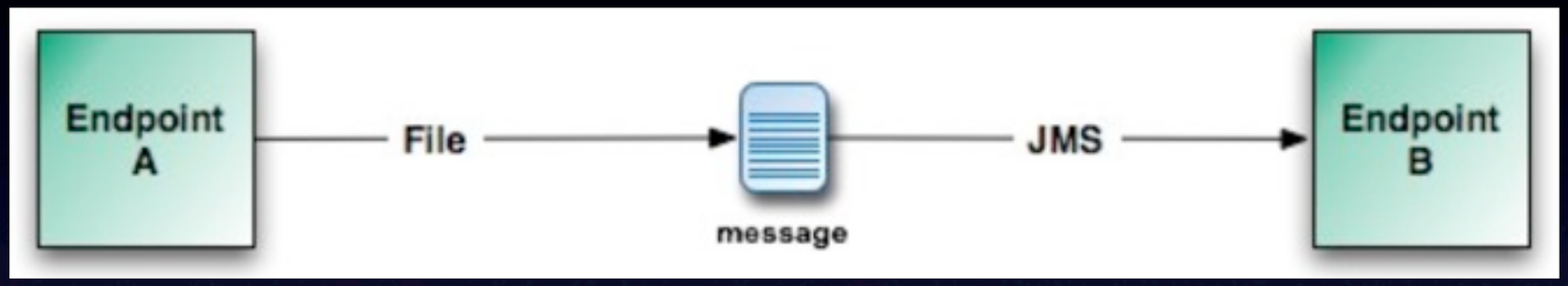

```
from("file:src/data?noop=true").
                to("jms:queue:myqueue");
```

一个简单的路由规则




```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("file:src/data?noop=true").
                to("jms:queue:myqueue");
    }
```

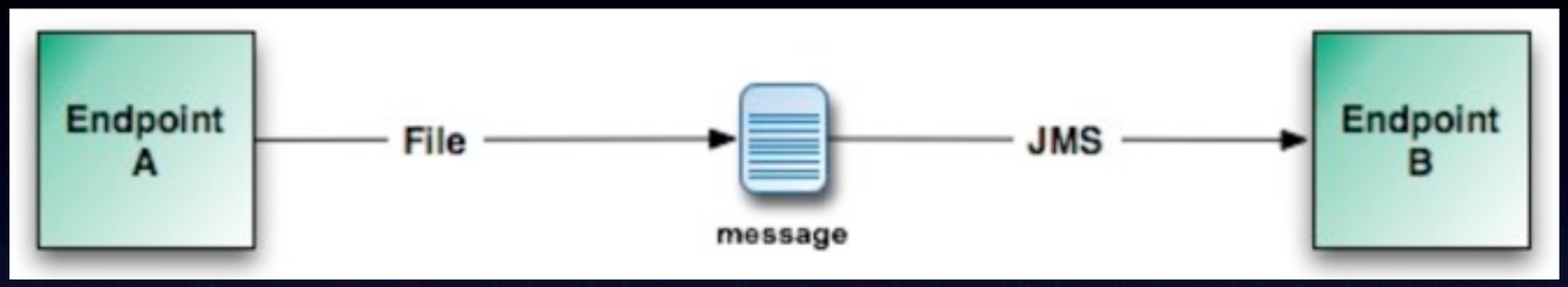
一个简单的路由规则




```
RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("file:src/data?noop=true").
                to("jms:queue:myqueue");
    }
```

URIs

一个简单的路由规则

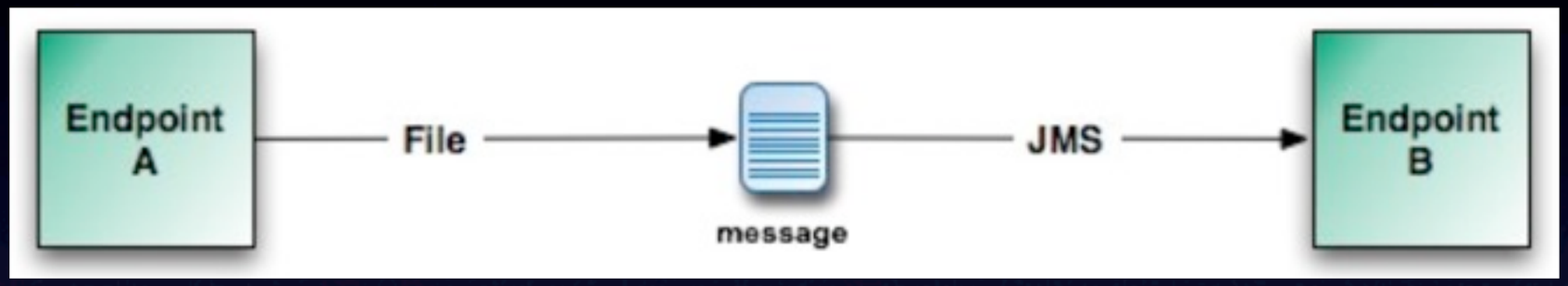



```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("file:src/data?noop=true").
      to("jms:queue:myqueue");
  }
```

protocol

URIs

一个简单的路由规则



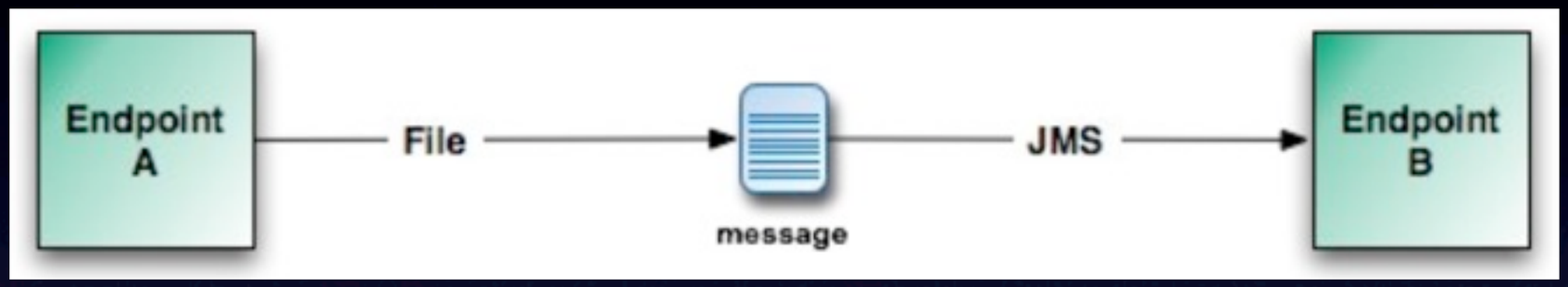

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("file:src/data?noop=true").
      to("jms:queue:myqueue");
  }
```

protocol

path

URIs

一个简单的路由规则




```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("file:src/data?noop=true").
      to("jms:queue:myqueue");
  }
```

protocol

path

options

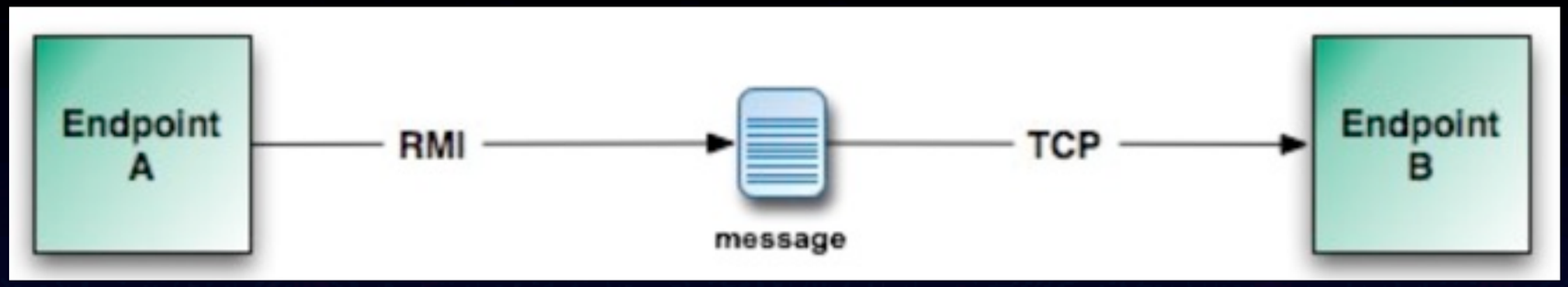
URIs

Camel 组件

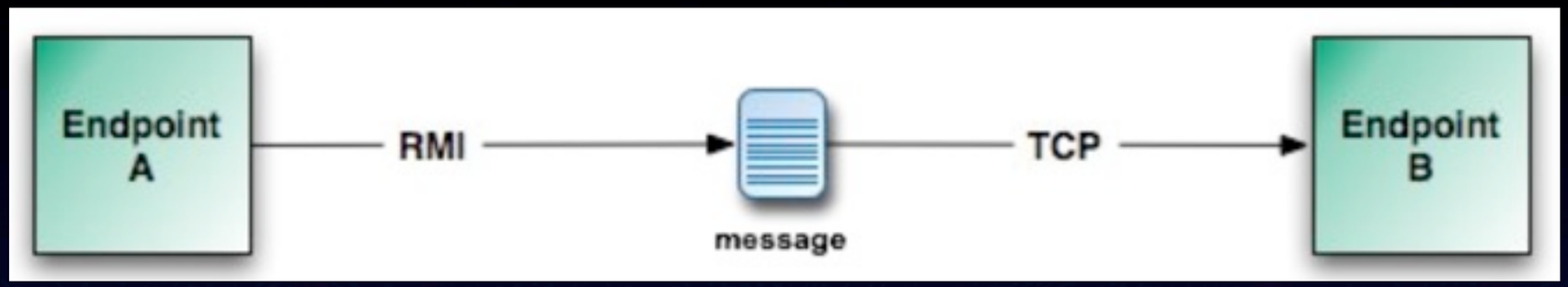
ActiveMQ	File	JBI	MINA	RMI	TCP
ActiveMQ Journal	FIX	JCR	Mock	RNC	Test
AMQP	Flatpack	JDBC	MSMQ	RNG	Timer
Atom	FTP	Jetty	MSV	SEDA	UDP
Bean	Hibernate	JMS	Multicast	SFTP	Validation
CXF	HTTP	JPA	POJO	SMTP	Velocity
DataSet	iBATIS	JT/400	POP	Spring Integration	VM
Direct	IMAP	List	Quartz	SQL	XMPP
Esper	IRC	Log	Queue	Stream	XQuery
Event	JavaSpace	Mail	Ref	String Template	XSLT

http://camel.apache.org/components.html

简单的消息路由续

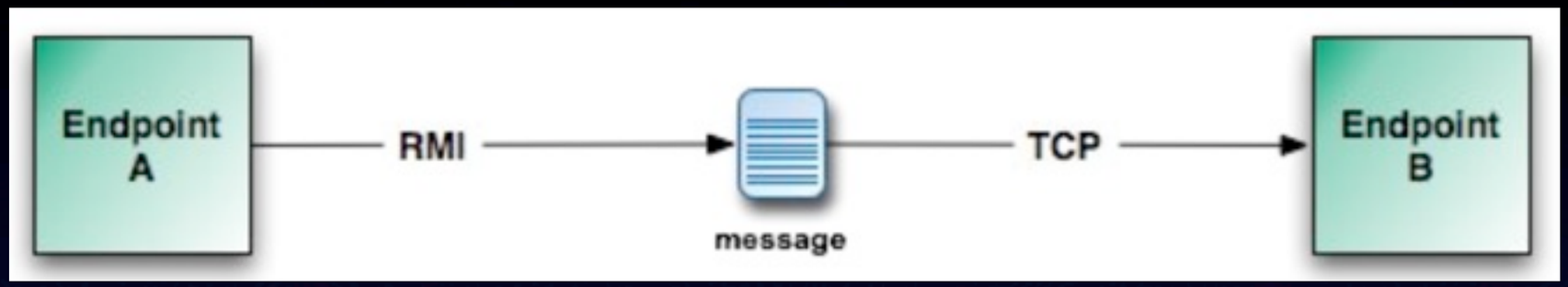


简单的消息路由续




```
from("rmi://localhost:1099/patch/to/service").
    to("netty:tcp://remotehost:1234");
```

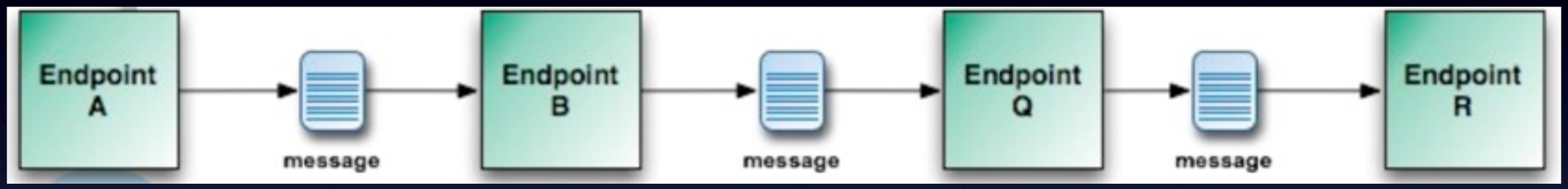
简单的消息路由续



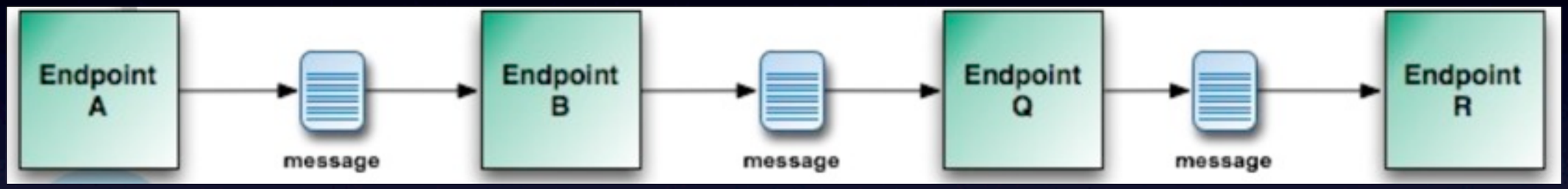

```
from("rmi://localhost:1099/patch/to/service").
        to("netty:tcp://remotehost:1234");
```

```
<camelContext>
  <route>
    <from uri="rmi://localhost:1099/patch/to/
service"/>
    <to uri="netty:tcp://remotehost:1234"/>
  <route>
</camelContext>
```

Pipeline

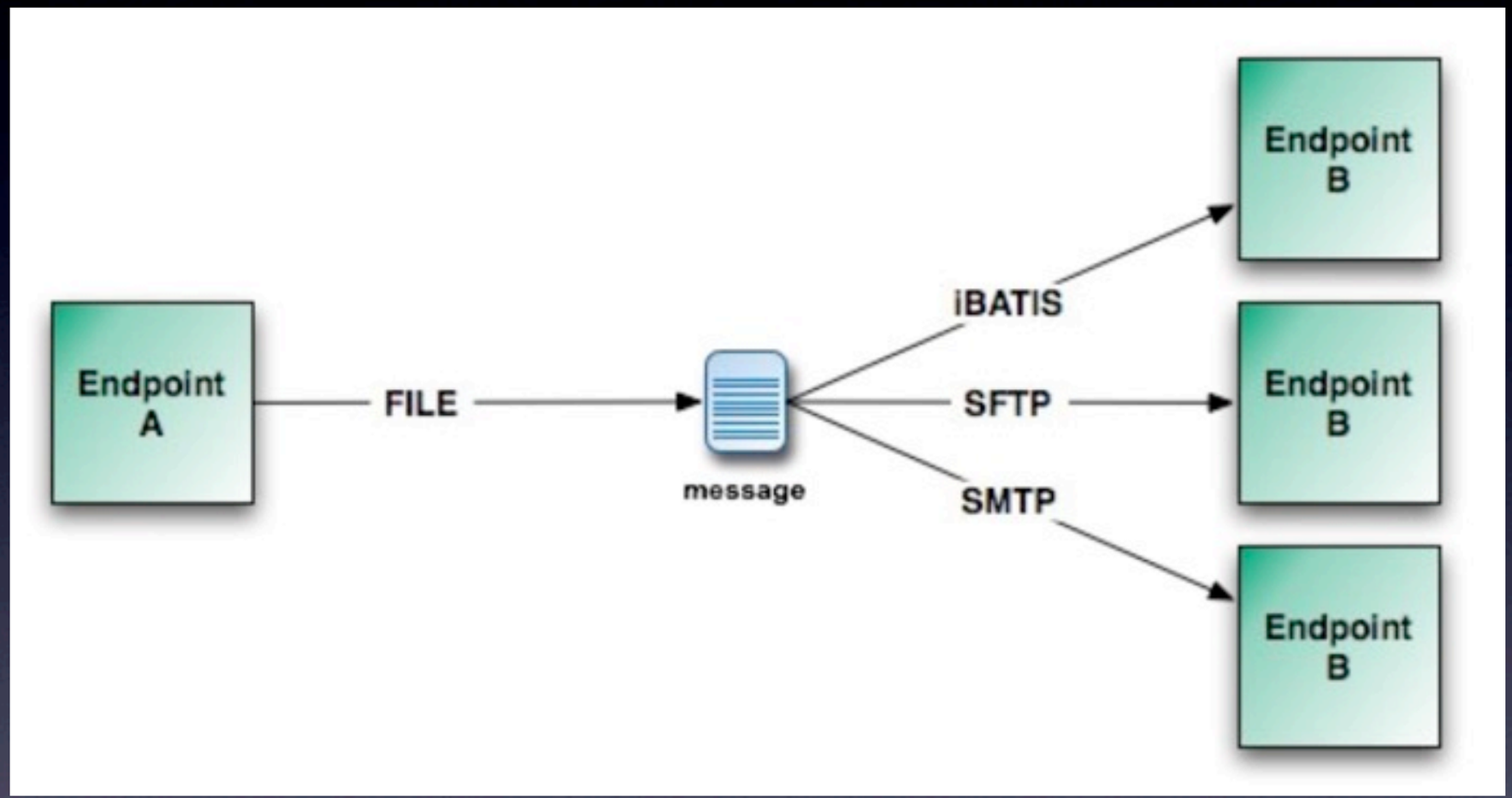

Endpoint A
message
Endpoint B
message
Endpoint Q
message
Endpoint R

Pipeline

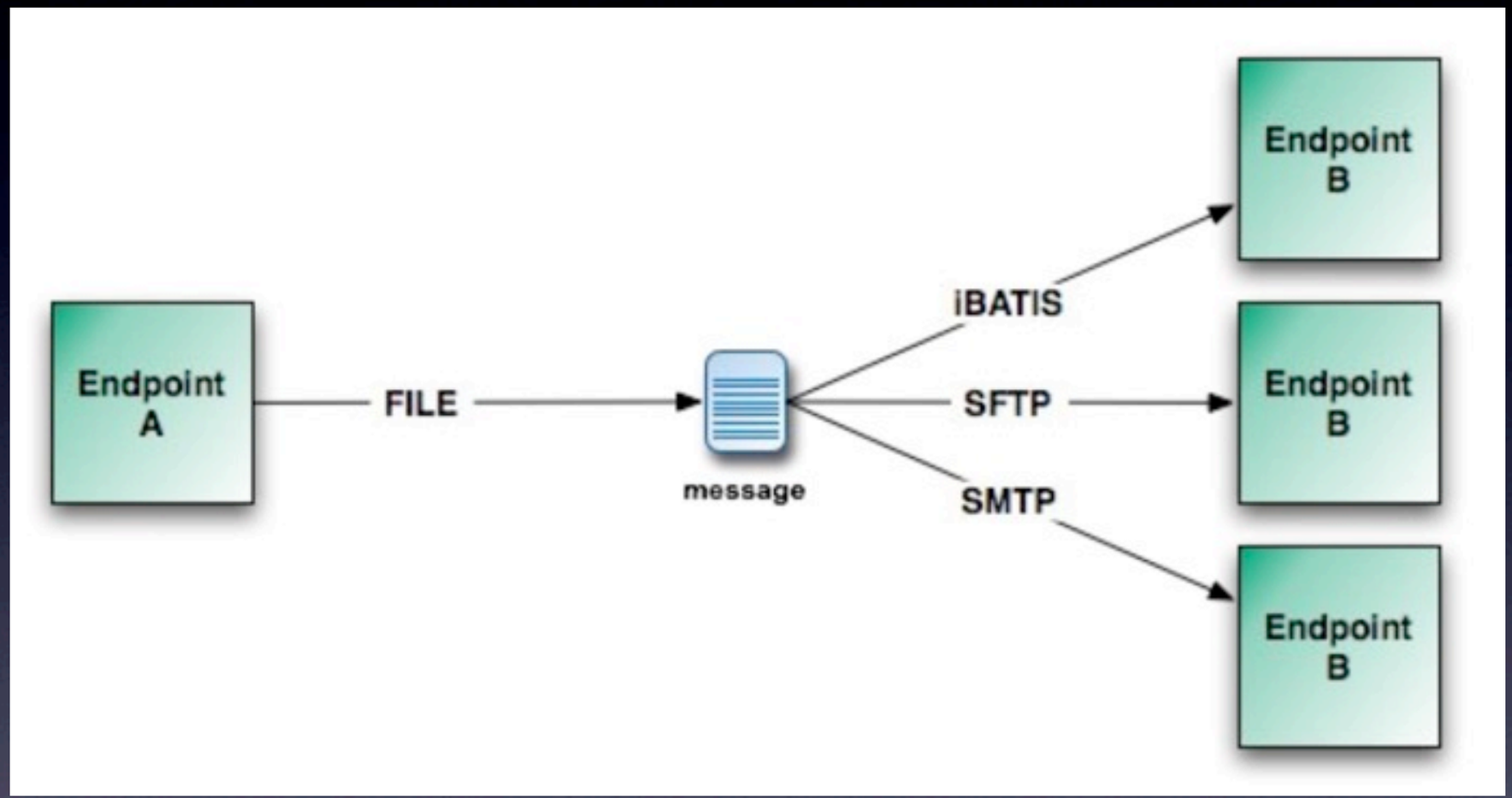



```
from("EndpointA").
pipleline("EndpointB", "EndpointQ",
          "EndpointR");
```

Multicast Routing

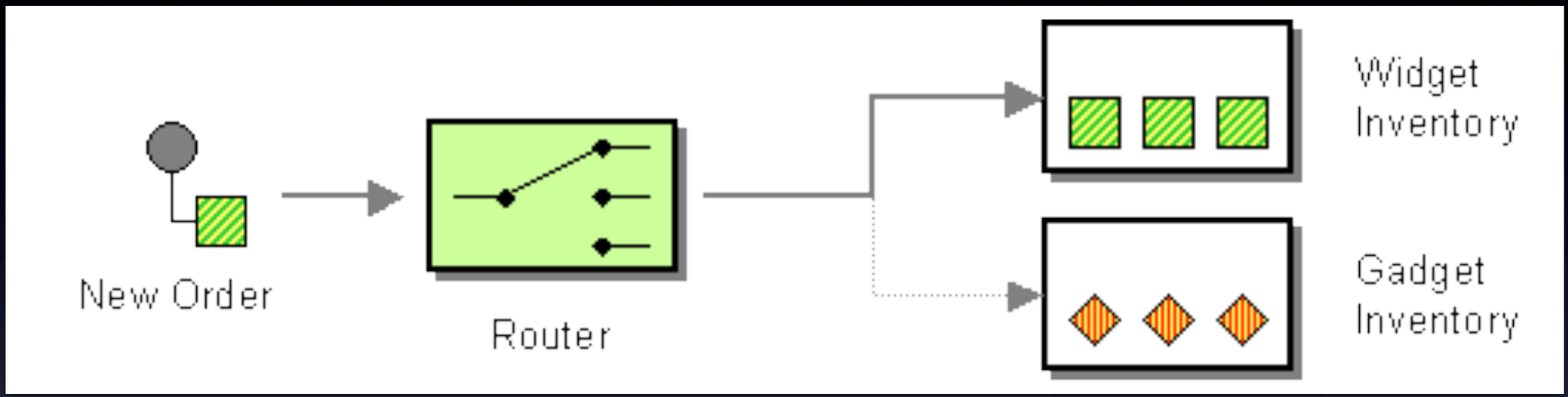


Multicast Routing

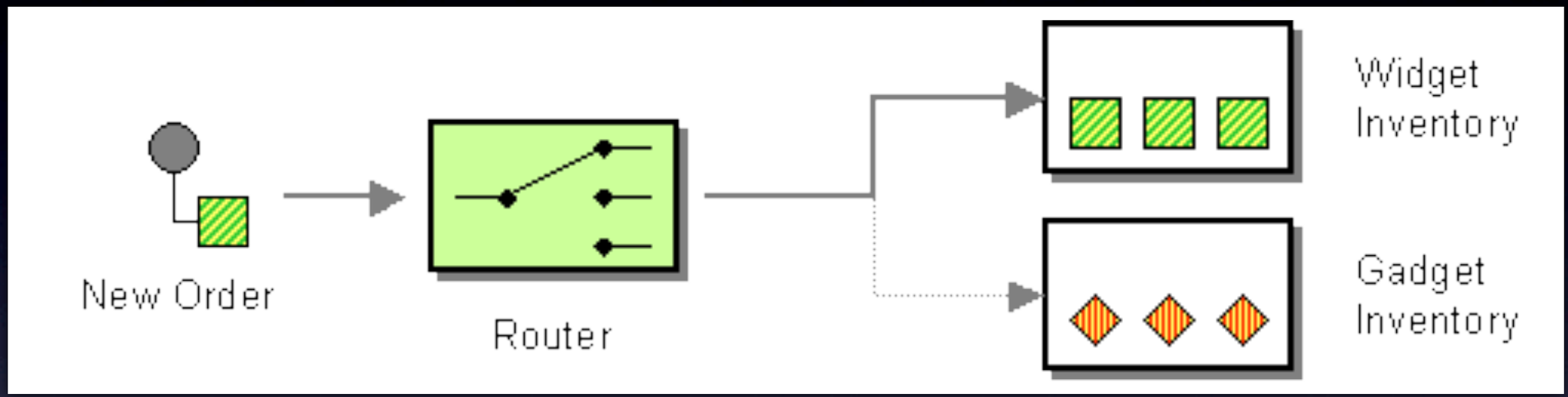



```
from("file:src/data?noop=true").
multicast("ibatis://xxx", "sftp://xxx",
          "smtp://xxx");
```

基于内容的路由



基于内容的路由




```
from("direct:a")
    .choice()
      .when(header("foo").isEqualTo("bar"))
        .to("direct:b")
      .when(header("foo").isEqualTo("cheese"))
        .to("direct:c")
      .otherwise()
        .to("direct:d");
```

Camel DSL

- Camel路由的编程接口
- 提供了EIP的实现
- 提供了多种语言的实现
  - 利用各种语言特性
  - Camel是以类库的方式进行工作
  - 通过Java模型进行交互

Camel DSL

- 内部 DSL
  - Java (annotation)
  - Scala
  - Groovy
- 外部 DSL
  - Spring XML
  - Blueprint

用Scala实现Camel DSL

- Apache Camel-Scala 是在2008加入的
- 为什么要使用Scala来实现?
  - Scala语言很强大
  - Camel DSL会更加简洁

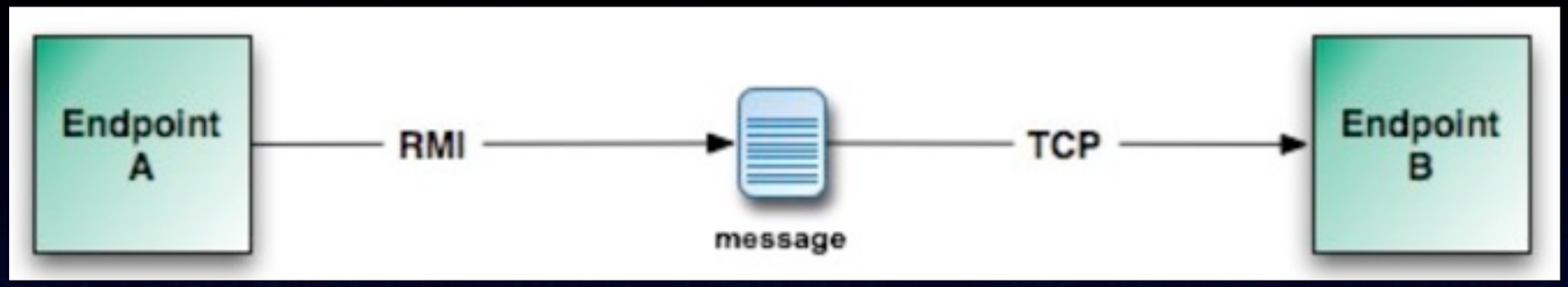
Scala介绍

- Sca(lable) la(nguage)
- 面向对象与函数式编程
- 静态类型，编译过程中使用类型推断
- 运行在JVM中
  - Scala 代码被编译成Java二进制代码
  - Java 代码和Scala代码可以相互调用

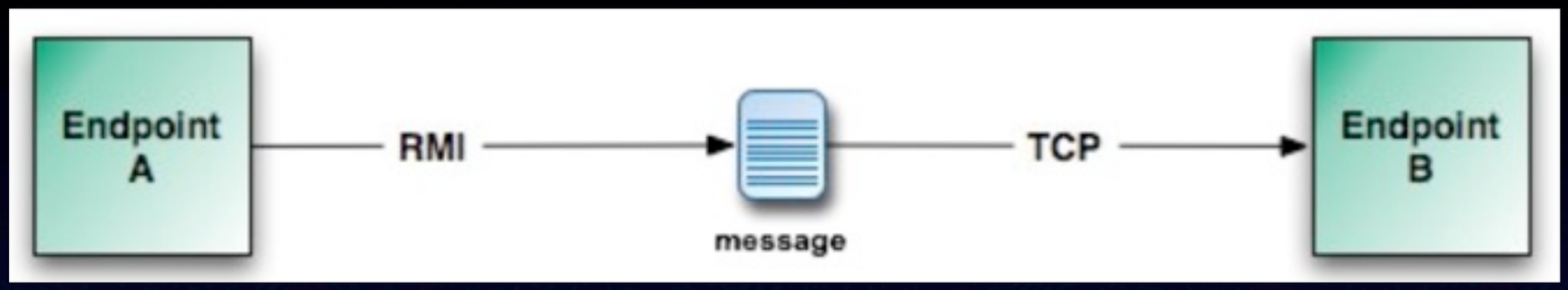
Scala 语言

- Scala对DSL支持友好
- 隐式转换
- 能够将函数或者块作为参数传入
- By-name parameters

Scala DSL支持

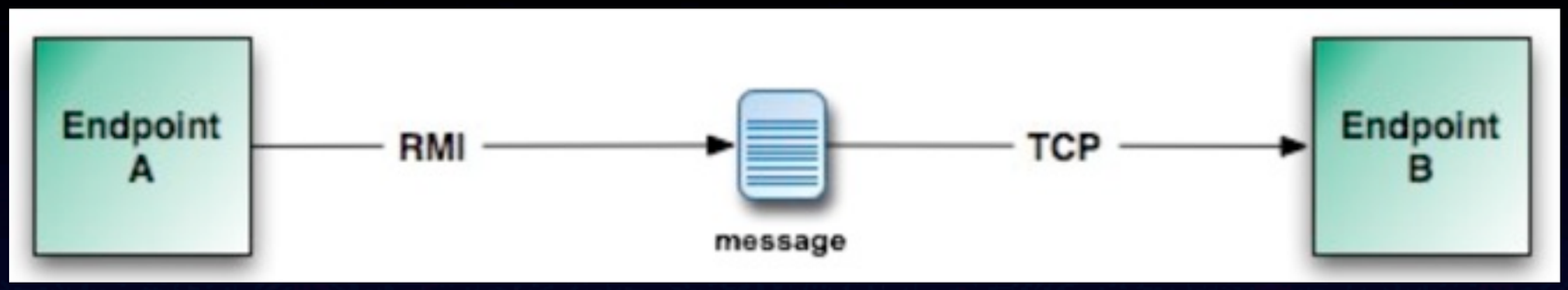


Scala DSL支持



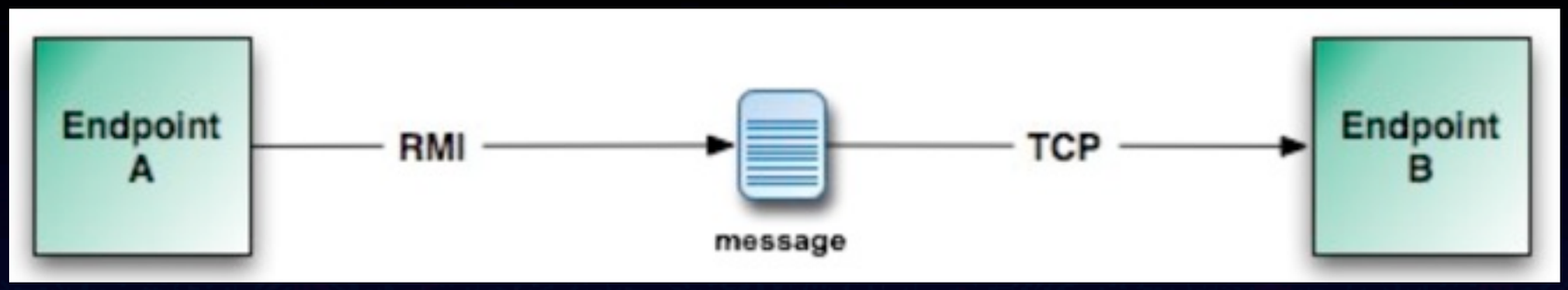

```
from("rmi://localhost:1099/patch/to/service").
    to("netty:tcp://remotehost:1234");
```

Scala DSL支持



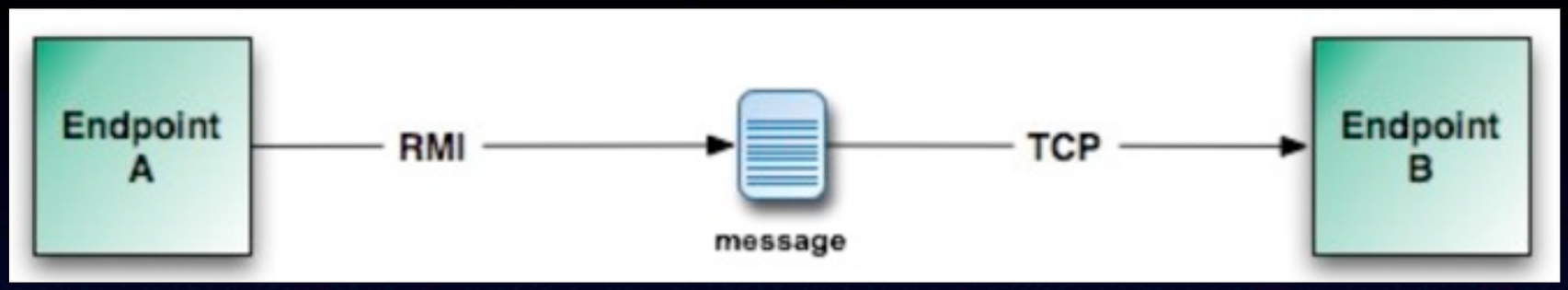

```
from("rmi://localhost:1099/patch/to/service").
        to("netty:tcp://remotehost:1234");

from "rmi://localhost:1099/patch/to/service" to
        "netty:tcp://remotehost:1234"
```

Scala DSL支持




```
from("rmi://localhost:1099/patch/to/service").
        to("netty:tcp://remotehost:1234");

from "rmi://localhost:1099/patch/to/service" to
        "netty:tcp://remotehost:1234"

"rmi://localhost:1099/patch/to/service" -->
        "netty:tcp://remotehost:1234"
```

Scala DSL支持




```
from("rmi://localhost:1099/patch/to/service").
        to("netty:tcp://remotehost:1234");

from "rmi://localhost:1099/patch/to/service" to
        "netty:tcp://remotehost:1234"

   "rmi://localhost:1099/patch/to/service" -->
        "netty:tcp://remotehost:1234"

"rmi://localhost:1099/patch/to/service" ! "Hello"
```

隐式转换

隐式转换

```
"rmi://localhost:1099/patch/to/service" ! "Hello"
```

隐式转换

```
"rmi://localhost:1099/patch/to/service" ! "Hello"
```

- 简单的字符串不具备收发消息的能力

隐式转换

```
"rmi://localhost:1099/patch/to/service" ! "Hello"
```

- 简单的字符串不具备收发消息的能力
- Java中的String 是一个 final类型，我们不能通过继承方式进行扩展

隐式转换

```
"rmi://localhost:1099/patch/to/service" ! "Hello"
```

- 简单的字符串不具备收发消息的能力
- Java中的String 是一个 final类型，我们不能通过继承方式进行扩展
- 通过隐式转换实现这样的功能

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  implicit def stringToUri(uri:String) = new RichTestUri(uri,
this)
...
```

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  implicit def stringToUri(uri:String) = new RichTestUri(uri,
this)
...
```

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  implicit def stringToUri(uri:String) = new RichTestUri(uri,
this)
...

class RichTestUri(uri: String, support: ScalaTestSupport) {

  def !(messages: Any*) {
    messages.foreach {
      _ match {
        case exchange: Exchange => support.getTemplate.send(uri,
exchange)
        case anything: Any => support.getTemplate.sendBody(uri,
anything)
      }
    }
  }
```

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  implicit def stringToUri(uri:String) = new RichTestUri(uri,
this)
...

class RichTestUri(uri: String, support: ScalaTestSupport) {

  def !(messages: Any*) {
    messages.foreach {
      _ match {
        case exchange: Exchange => support.getTemplate.send(uri,
exchange)
        case anything: Any => support.getTemplate.sendBody(uri,
anything)
      }
    }
  }
```

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  implicit def stringToUri(uri:String) = new RichTestUri(uri,
this)
...

class RichTestUri(uri: String, support: ScalaTestSupport) {

  def !(messages: Any*) {
    messages.foreach {
      _ match {
        case exchange: Exchange => support.getTemplate.send(uri,
exchange)
        case anything: Any => support.getTemplate.sendBody(uri,
anything)
      }
    }
  }
```

```
abstract class ScalaTestSupport extends CamelTestSupport with
RouteBuilderSupport with Preamble {

  "rmi://localhost:1099/patch/to/service" ! "Hello"

  implicit def stringToRichTestUri(uri: String) = new RichTestUri(uri,
this)
...

class RichTestUri(uri: String, support: ScalaTestSupport) {

  def !(messages: Any*) {
    messages.foreach {
      _ match {
        case exchange: Exchange => support.getTemplate.send(uri,
exchange)
        case anything: Any => support.getTemplate.sendBody(uri,
anything)
      }
    }
  }
```

```
public class MyRouteBuilder extends RouteBuilder {
       public void configure() throws Exception {
          from("direct:a").processor(
             new Processor() {
                 public void process(Exchange exchange) throws Exception {
                     exchange.getIn().setBody("<hello/>");
                 }
             }
          ).to("mock:a");
       }
}
```

```
public class MyRouteBuilder extends RouteBuilder {
       public void configure() throws Exception {
          from("direct:a").processor(
             new Processor() {
                public void process(Exchange exchange) throws Exception {
                   exchange.getIn().setBody("<hello/>");
                }
             }
          ).to("mock:a");
       }
}
```

```
public class MyRouteBuilder extends RouteBuilder {
       public void configure() throws Exception {
          from("direct:a").processor(
             new Processor() {
                 public void process(Exchange exchange) throws Exception {
                     exchange.getIn().setBody("<hello/>");
                 }
             }
          ).to("mock:a");
       }
}
```

```
public class MyRouteBuilder extends RouteBuilder {
       public void configure() throws Exception {
          from("direct:a").processor(
             new Processor() {
                 public void process(Exchange exchange) throws Exception {
                     exchange.getIn().setBody("<hello/>");
                 }
             }
          ).to("mock:a");
       }
  }
    public interface Processor() {
        void process(Exchange exchange) throws Exception;
    }
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").processor(
            new Processor() {
                public void process(Exchange exchange) throws Exception {
                    exchange.getIn().setBody("<hello/>");
                }
            }
         ).to("mock:a");
      }
   }
   public interface Processor() {
       void process(Exchange exchange) throws Exception;
   }
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").processor(
            new Processor() {
                public void process(Exchange exchange) throws Exception {
                    exchange.getIn().setBody("<hello/>");
                }
            }
         ).to("mock:a");
      }
}
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").processor(
            new Processor() {
                public void process(Exchange exchange) throws Exception {
                    exchange.getIn().setBody("<hello/>");
                }
            }
         ).to("mock:a");
      }
}
```

```
val builder = new RouteBuilder {
      "direct:a" processor(_.in = "<hello/>") to("mock:a")
}
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").processor(
            new Processor() {
                public void process(Exchange exchange) throws Exception {
                    exchange.getIn().setBody("<hello/>");
                }
            }
         ).to("mock:a");
      }
}
```

```
val builder = new RouteBuilder {
      "direct:a" processor(_.in = "<hello/>") to("mock:a")
}
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").processor(
            new Processor() {
                public void process(Exchange exchange) throws Exception {
                    exchange.getIn().setBody("<hello/>");
                }
            }
         ).to("mock:a");
      }
}
```

```
val builder = new RouteBuilder {
      "direct:a" processor(_.in = "<hello/>") to("mock:a")
}
```

```
(exchange:Exchange) => exchange.in = "<hello/>"
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").filter(body().isEqualTo("<hello/>"))
.to("mock:a");
      }
   }
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").filter(body().isEqualTo("<hello/>"))
.to("mock:a");
      }
   }
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").filter(body().isEqualTo("<hello/>"))
.to("mock:a");
      }
   }
```

断言

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").filter(body().isEqualTo("<hello/>"))
.to("mock:a");
      }
   }
```

断言

```
val builder = new RouteBuilder {
      "direct:a" filter(_.in == "<hello/>") to("mock:a")
}
```

```
public class MyRouteBuilder extends RouteBuilder {
      public void configure() throws Exception {
         from("direct:a").filter(body().isEqualTo("<hello/>"))
.to("mock:a");
      }
   }
```

断言

```
val builder = new RouteBuilder {
      "direct:a" filter(_.in == "<hello/>") to("mock:a")
}
```

```
(exchange:Exchange) => exchange.in == "<hello/>"
```

```
def filter(predicate: Exchange => Any) =
SFilterDefinition(target.filter(predicateBuilder(predicate)))
```

```
def filter(predicate: Exchange => Any) =
SFilterDefinition(target.filter(predicateBuilder(predicate)))
```

```
def filter(predicate: Exchange => Any) =
SFilterDefinition(target.filter(predicateBuilder(predicate)))
```

```
implicit def predicateBuilder(predicate: Exchange => Any) =
new ScalaPredicate(predicate)
```

```
def filter(predicate: Exchange => Any) =
SFilterDefinition(target.filter(predicateBuilder(predicate)))
```

```
implicit def predicateBuilder(predicate: Exchange => Any) =
new ScalaPredicate(predicate)
```

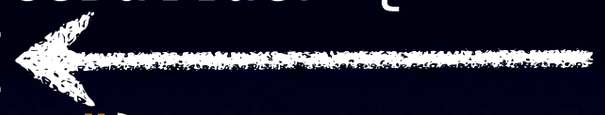
```
class ScalaPredicate(function: Exchange => Any) extends Predicate {

  override def matches(exchange: Exchange) = {
    val predicate = function(exchange)
    predicate match {
      case f : LanguageFunction => f.matches(exchange)
      case _ => evaluateValuePredicate(predicate)
    }
  }

}
```

```
public class MyRouteBuilder extends RouteBuilder {
    public void configure() throws Exception {
        from("direct:a").to("mock:polyglot")
          .choice()
                .when(body().isEqualTo("<hallo/>"))
                  .to("mock:dutch")
                .when(body().isEqualTo("<guten/>"))
                  .to("mock:german")
              .endChoice()
            .when(body().isEqualTo("<hello/>"))
                .to("mock:english")
            .otherwise()
                .to("mock:french");
    }
}
```

```
public class MyRouteBuilder extends RouteBuilder {
    public void configure() throws Exception {
        from("direct:a").to("mock:polyglot")
          .choice()
                .when(body().isEqualTo("<hallo/>"))
                  .to("mock:dutch")
                  .to("mock:german")
                .endChoice()
               .when(body().isEqualTo("<hello/>"))
                  .to("mock:english")
               .otherwise()
                  .to("mock:french");
    }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
            to ("mock:dutch")
            to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
            to ("mock:dutch")
            to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```



```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
            to ("mock:dutch")
            to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
            to ("mock:dutch")
            to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
           to ("mock:dutch")
           to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
           to ("mock:dutch")
           to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
           to ("mock:dutch")
           to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
def ==> (block: => Unit) : SRouteDefinition =
this.apply(block).asInstanceOf[SRouteDefinition]
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
           to ("mock:dutch")
           to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
def ==> (block: => Unit) : SRouteDefinition =
this.apply(block).asInstanceOf[SRouteDefinition]
```

```
val builder = new RouteBuilder {
      "direct:a" ==> {
      to ("mock:polyglot")
      choice {
         when (_.in == "<hello/>") to ("mock:english")
         when (_.in == "<hallo/>") {
           to ("mock:dutch")
           to ("mock:german")
         }
         otherwise to ("mock:french")
      }
   }
}
```

```
trait Block {
  def apply(block: => Unit) : DSL
}
```

```
def ==> (block: => Unit) : SRouteDefinition =
this.apply(block).asInstanceOf[SRouteDefinition]
```

Scala的其它语言特点

- traits and mixins
- pattern matching
- partially applied functions
- actors
- annotation support
- ...

使用Scala的心得

- 隐式转换很强大，但是要注意选择好正确的作用域
- 类型，静态类型，编译器能帮助你学习使用正确的类型
- 使用 REPL 或者 IDE来实验语言特性
- Maven plugin可以帮助你在Java项目中使用Scala

Camel DSL

为何要用Scala实现Camel DSL?