**Part IV. Spring Cloud Stream**

**24. Introducing Spring Cloud Stream**

Spring Cloud Stream适用于构建消息驱动的微服务应用的框架。它构建在Spring Boot之上用以创建独立的工业级的应用程序，并且通过Spring Integration提供了和消息代理的连接。Spring Cloud Stream为一些供应商的消息中间件产品提供了个性化的自动化配置实现(目前仅支持RabbitMQ和Kafka)，同时引入了发布订阅、消费组和分区的语义概念。

你可以通过添加@EnableBinding 注解快速连接到一个message broker，同时你可以给方法添加@StreamListener注解，以使其接收流处理的事件。以下是接收外部消息的简单接收器应用程序。

*@SpringBootApplication*

*@EnableBinding(Sink.class)*

**public** **class** VoteRecordingSinkApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.run(VoteRecordingSinkApplication.**class**, args);

}

*@StreamListener(Sink.INPUT)*

**public** **void** processVote(Vote vote) {

votingService.recordVote(vote);

}

}

@EnableBinding注解需要一个或多个接口作为参数（在这种情况下，该参数是单个Sink接口）。接口声明输入和/或输出通道。Spring Cloud Stream提供了接口Source，Sink和Processor; 你也可以定义自己的接口。

以下是Sink接口的定义：

**public** **interface** Sink {

String INPUT = "input";

*@Input(Sink.INPUT)*

SubscribableChannel input();

}

注解@Input定义了一个输入通道，应用可以通过该通道接收到消息。@Output注解定义了一个输出通道，应用可以通过该通道法发布消息。这两个注解接收一个通道名作为参数，默认会采用被注解的方法名作为参数。

Spring Cloud Stream 会给你创建一个接口的实现类，你可以使用直接通过注入到应用中进行使用：

*@RunWith(SpringJUnit4ClassRunner.class)*

*@SpringApplicationConfiguration(classes = VoteRecordingSinkApplication.class)*

*@WebAppConfiguration*

*@DirtiesContext*

**public** **class** StreamApplicationTests {

*@Autowired*

**private** Sink sink;

*@Test*

**public** **void** contextLoads() {

assertNotNull(**this**.sink.input());

}

}

**25.主要概念**

Spring Cloud Stream 提供了一些简写消息驱动微服务框架的概念和基本单位。这部分的大体内容如下：

Spring Cloud Stream的应用模型

Binder的概念

持久化发布-订阅模式的支持

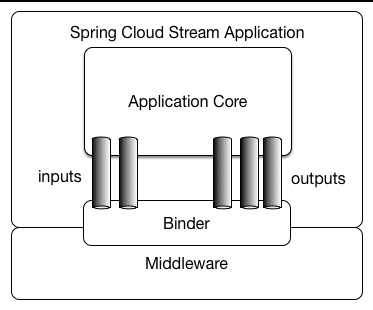
消费组模式的支持

分区模式的支持

可拔插的Binder API

**25.1应用模型**

Spring Cloud Stream 由第三方中间件组成。应用间的通信通过输入通道（input channel）和输出通道（output channel）完成。这些通道是有Spring Cloud Stream 注入的。而通道与外部的代理（可以理解为上文所说的数据中心）的连接又是通过Binder实现的。



**25.1.1Fat JAR**

Spring Cloud Stream应用可以直接在IDE运行。这样会很方便测试。但在生产环境下，Spring Boot为maven和Gradle提供了打包成可运行jar的工具，你可以使用这个工具将Spring Cloud Stream应用打包。

**25.2 The Binder Abstraction**

Spring Cloud Stream 提供了Kafka和Rabbit MQ的Binder实现（Binder可以理解为提供了Middleware操作方法的类）。Spring Cloud Stream还实现了一个专门用于测试的TestSupportBinder，开发者可以直接使用它来对通道的接收内容进行可靠的测试断言。如果要使用除了RabbitMQ和Kafka以外的消息中间件的话，我们也可以通过使用它所提供的扩展API来实现其他中间件的Binder。

Spring Cloud Stream使用spring boot的配置方式进行配置，Binder的概念使得应用可以很灵活的连接到中间件。举例来说，开发者可以再程序运行中动态的选择channels 连接到哪个destitation（kafka的topic或者RabbitMQ的exchanges）。这样的配置可以通过springboot支持的任何一种配置方式进行设置（包括程序参数,环境变量,和 application.yml 或 application.properties 文件）。在24章的例子中，将属性spring.cloud.stream.bindings.input.destination设为raw-sensor-data会使得读取位置变为raw-sensor-data 的Kafka topic或者一个绑定了raw-sensor-data rabbitMQ exchanges的队列。

Spring Cloud Stream可以自动检测并使用路径下的binder。你只需在编译时选择不同的binder，就可以在不改动代码的情况下使用不同的中间件。对于一些复杂的使用情况，你也可以package multiple binders with your application and have it choose the binder, and even whether to use different binders for different channels, at runtime.

**25.3 Persistent Publish-Subscribe Support**

在Spring Cloud Stream中的消息通信方式遵循了发布-订阅模式，当一条消息被投递到消息中间件之后，它会通过共享的Topic主题进行广播。

