**Spring Cloud构建微服务架构（七）消息总线（续：Kafka）**

**原创**

[2016-10-03](http://blog.didispace.com/springcloud7-2/)

 翟永超

[Spring Cloud](http://blog.didispace.com/categories/Spring-Cloud/)

被围观 **17724** 次

Spring Cloud Bus除了支持RabbitMQ的自动化配置之外，还支持现在被广泛应用的Kafka。在本文中，我们将搭建一个Kafka的本地环境，并通过它来尝试使用Spring Cloud Bus对Kafka的支持，实现消息总线的功能。由于本文会以之前Rabbit的实现作为基础来修改，所以先阅读[《Spring Cloud构建微服务架构（七）消息总线》](http://blog.didispace.com/springcloud7/)有助于理解本文。

**Kafka简介**

Kafka是一个由LinkedIn开发的分布式消息系统，它于2011年初开源，现在由著名的Apache基金会维护与开发。Kafka使用Scala实现，被用作LinkedIn的活动流和运营数据处理的管道，现在也被诸多互联网企业广泛地用作为数据流管道和消息系统。

Kafka是基于消息发布/订阅模式实现的消息系统，其主要设计目标如下：

* 消息持久化：以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上数据也能保证常数时间复杂度的访问性能。
* 高吞吐：在廉价的商用机器上也能支持单机每秒100K条以上的吞吐量
* 分布式：支持消息分区以及分布式消费，并保证分区内的消息顺序
* 跨平台：支持不同技术平台的客户端（如：Java、PHP、Python等）
* 实时性：支持实时数据处理和离线数据处理
* 伸缩性：支持水平扩展

Kafka中涉及的一些基本概念：

* Broker：Kafka集群包含一个或多个服务器，这些服务器被称为Broker。
* Topic：逻辑上同Rabbit的Queue队列相似，每条发布到Kafka集群的消息都必须有一个Topic。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个Broker上，但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）
* Partition：Partition是物理概念上的分区，为了提供系统吞吐率，在物理上每个Topic会分成一个或多个Partition，每个Partition对应一个文件夹（存储对应分区的消息内容和索引文件）。
* Producer：消息生产者，负责生产消息并发送到Kafka Broker。
* Consumer：消息消费者，向Kafka Broker读取消息并处理的客户端。
* Consumer Group：每个Consumer属于一个特定的组（可为每个Consumer指定属于一个组，若不指定则属于默认组），组可以用来实现一条消息被组内多个成员消费等功能。

**快速入门**

在对Kafka有了一些基本了解之后，下面我们来尝试构建一个Kafka服务端，并体验一下基于Kafka的消息生产与消费。

**环境安装**

首先，我们需要从官网上下载安装介质。下载地址为：http://kafka.apache.org/downloads.html。本例中采用的版本为：Kafka-0.10.0.1

在解压Kafka的安装包之后，可以看到其目录结构如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | kafka  +-bin  +-windows  +-config  +-libs  +-logs  +-site-docs |

由于Kafka的设计中依赖了ZooKeeper，所以我们可以在bin和config目录中除了看到Kafka相关的内容之外，还有ZooKeeper相关的内容。其中bin目录存放了Kafka和ZooKeeper的命令行工具，bin根目录下是适用于Linux/Unix的shell，而bin/windows下的则是适用于windows下的bat。我们可以根据实际的系统来设置环境变量，以方便后续的使用和操作。而在config目录中，则是用来存放了关于Kafka与ZooKeeper的配置信息。

**启动测试**

下面我们来尝试启动ZooKeeper和Kafka来进行消息的生产和消费。示例中所有的命令均已配置了Kafka的环境变量为例。

* **启动ZooKeeper**，执行命令：zookeeper-server-start config/zookeeper.properties，该命令需要指定zookeeper的配置文件位置才能正确启动，kafka的压缩包中包含了其默认配置，开发与测试环境不需要修改。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | [2016-09-28 08:05:34,849] INFO Reading configuration from: config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)  [2016-09-28 08:05:34,850] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)  [2016-09-28 08:05:34,851] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)  [2016-09-28 08:05:34,851] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)  [2016-09-28 08:05:34,852] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)  [2016-09-28 08:05:34,868] INFO Reading configuration from: config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)  [2016-09-28 08:05:34,869] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)  ...  [2016-09-28 08:05:34,940] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory) |

从控制台信息中，我们可以看到ZooKeeper从指定的config/zookeeper.properties配置文件中读取信息并绑定2181端口启动服务。有时候启动失败，可查看一下端口是否被占用，可以杀掉占用进程或通过修改config/zookeeper.properties配置文件中的clientPort内容以绑定其他端口号来启动ZooKeeper。

* **启动Kafka**，执行命令：kafka-server-start config/server.properties，该命令也需要指定Kafka配置文件的正确位置，如上命令中指向了解压目录包含的默认配置。若在测试时，使用外部集中环境的ZooKeeper的话，我们可以在该配置文件中通过zookeeper.connect参数来设置ZooKeeper的地址和端口，它默认会连接本地2181端口的ZooKeeper；如果需要设置多个ZooKeeper节点，可以为这个参数配置多个ZooKeeper地址，并用逗号分割。比如：zookeeper.connect=127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002。
* **创建Topic**，执行命令：kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test，通过该命令，创建一个名为“test”的Topic，该Topic包含一个分区一个Replica。在创建完成后，可以使用kafka-topics --list --zookeeper localhost:2181命令来查看当前的Topic。

另外，如果我们不使用kafka-topics命令来手工创建，直接进行下面的内容进行消息创建时也会自动创建Topics来使用。

* **创建消息生产者**，执行命令：kafka-console-producer --broker-list localhost:9092 --topic test。kafka-console-producer命令可以启动Kafka基于命令行的消息生产客户端，启动后可以直接在控制台中输入消息来发送，控制台中的每一行数据都会被视为一条消息来发送。我们可以尝试输入几行消息，由于此时并没有消费者，所以这些输入的消息都会被阻塞在名为test的Topics中，直到有消费者将其消费掉位置。
* **创建消息消费者**，执行命令：kafka-console-consumer --zookeeper localhost:2181 --topic test --from-beginning。kafka-console-consumer命令启动的是Kafka基于命令行的消息消费客户端，在启动之后，我们马上可以在控制台中看到输出了之前我们在消息生产客户端中发送的消息。我们可以再次打开之前的消息生产客户端来发送消息，并观察消费者这边对消息的输出来体验Kafka对消息的基础处理。

**整合Spring Cloud Bus**

在上一篇使用Rabbit实现消息总线的案例中，我们已经通过引入spring-cloud-starter-bus-amqp模块，完成了使用RabbitMQ来实现的消息总线。若我们要使用Kafka来实现消息总线时，只需要把spring-cloud-starter-bus-amqp替换成spring-cloud-starter-bus-kafka模块，在pom.xml的dependenies节点中进行修改，具体如下：

|  |  |
| --- | --- |
| 1  2  3  4 | <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-kafka</artifactId>  </dependency> |

如果我们在启动Kafka时均采用了默认配置，那么我们不需要再做任何其他配置就能在本地实现从RabbitMQ到Kafka的切换。我们可以尝试把刚刚搭建的ZooKeeper、Kafka启动起来，并将修改为spring-cloud-starter-bus-kafka模块的config-server和config-client启动起来。

在config-server启动时，我们可以在控制台中看到如下输出：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 2016-09-28 22:11:29.627 INFO 15144 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder : Using kafka topic for outbound: springCloudBus  2016-09-28 22:11:29.642 INFO 15144 --- [-localhost:2181] org.I0Itec.zkclient.ZkEventThread : Starting ZkClient event thread.  ...  016-09-28 22:11:30.290 INFO 15144 --- [ main] o.s.i.kafka.support.ProducerFactoryBean : Using producer properties => {bootstrap.servers=localhost:9092, linger.ms=0, acks=1, compression.type=none, batch.size=16384}  2016-09-28 22:11:30.298 INFO 15144 --- [ main] o.a.k.clients.producer.ProducerConfig : ProducerConfig values:  ...  2016-09-28 22:11:30.322 INFO 15144 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$1 : Adding {message-handler:outbound.springCloudBus} as a subscriber to the 'springCloudBusOutput' channel  2016-09-28 22:11:30.322 INFO 15144 --- [ main] o.s.integration.channel.DirectChannel : Channel 'config-server:7001.springCloudBusOutput' has 1 subscriber(s).  2016-09-28 22:11:30.322 INFO 15144 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$1 : started outbound.springCloudBus  ...  2016-09-28 22:11:31.465 INFO 15144 --- [ main] s.i.k.i.KafkaMessageDrivenChannelAdapter : started org.springframework.integration.kafka.inbound.KafkaMessageDrivenChannelAdapter@4178cb34  2016-09-28 22:11:31.467 INFO 15144 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$7 : Adding {message-handler:inbound.springCloudBus.anonymous.8b9e6c7b-6a50-48c5-b981-8282a0d5a30b} as a subscriber to the 'bridge.springCloudBus' channel  2016-09-28 22:11:31.467 INFO 15144 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$7 : started inbound.springCloudBus.anonymous.8b9e6c7b-6a50-48c5-b981-8282a0d5a30b |

从控制台的输出内容，我们可以看到config-server连接到了Kafka中，并使用了名为springCloudBus的Topic。

此时，我们可以使用kafka-topics --list --zookeeper localhost:2181命令来查看当前Kafka中的Topic，若已成功启动了config-server并配置正确，我们就可以在Kafka中看到已经多了一个名为springCloudBus的Topic。

我们再启动配置了spring-cloud-starter-bus-kafka模块的config-client，可以看到控制台中输出如下内容：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 2016-09-28 22:43:55.067 INFO 6136 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder : Using kafka topic for outbound: springCloudBus  2016-09-28 22:43:55.078 INFO 6136 --- [-localhost:2181] org.I0Itec.zkclient.ZkEventThread : Starting ZkClient event thread.  ...  2016-09-28 22:50:38.584 INFO 828 --- [ main] o.s.i.kafka.support.ProducerFactoryBean : Using producer properties => {bootstrap.servers=localhost:9092, linger.ms=0, acks=1, compression.type=none, batch.size=16384}  2016-09-28 22:50:38.592 INFO 828 --- [ main] o.a.k.clients.producer.ProducerConfig : ProducerConfig values:  ...  2016-09-28 22:50:38.615 INFO 828 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$1 : Adding {message-handler:outbound.springCloudBus} as a subscriber to the 'springCloudBusOutput' channel  2016-09-28 22:50:38.616 INFO 828 --- [ main] o.s.integration.channel.DirectChannel : Channel 'didispace:7002.springCloudBusOutput' has 1 subscriber(s).  2016-09-28 22:50:38.616 INFO 828 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$1 : started outbound.springCloudBus  ...  2016-09-28 22:50:39.162 INFO 828 --- [ main] s.i.k.i.KafkaMessageDrivenChannelAdapter : started org.springframework.integration.kafka.inbound.KafkaMessageDrivenChannelAdapter@60cf855e  2016-09-28 22:50:39.162 INFO 828 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$7 : Adding {message-handler:inbound.springCloudBus.anonymous.f8fc9c0c-ccd3-46dd-9537-07198f4ee216} as a subscriber to the 'bridge.springCloudBus' channel  2016-09-28 22:50:39.163 INFO 828 --- [ main] o.s.c.s.b.k.KafkaMessageChannelBinder$7 : started inbound.springCloudBus.anonymous.f8fc9c0c-ccd3-46dd-9537-07198f4ee216 |

可以看到，config-client启动时输出了类似的内容，他们都订阅了名为springCloudBus的Topic。

在启动了config-server和config-client之后，为了更明显地观察消息总线刷新配置的效果，我们可以在本地启动多个不同端口的config-client。此时，我们的config-server以及多个config-client都已经连接到了由Kafka实现的消息总线上。我们可以先访问各个config-client上的/from请求，查看他获取到的配置内容。然后，修改Git中对应的参数内容，再访问各个config-client上的/from请求，可以看到配置内容并没有改变。最后，我们向config-server发送POST请求：/bus/refresh，此时我们再去访问各个config-client上的/from请求，就能获得到最新的配置信息，各客户端上的配置都已经加载为最新的Git配置内容。

从config-client的控制台中，我们可以看到如下内容：

|  |  |
| --- | --- |
| 1 | 2016-09-29 08:20:34.361 INFO 21256 --- [ kafka-binder-1] o.s.cloud.bus.event.RefreshListener : Received remote refresh request. Keys refreshed [from] |

RefreshListener监听类记录了收到远程刷新请求，并刷新了from属性的日志。

**Kafka配置**

在上面的例子中，由于Kafka、ZooKeeper均运行于本地，所以我们没有在测试程序中通过配置信息来指定Kafka和ZooKeeper的配置信息，就完成了本地消息总线的试验。但是我们实际应用中，Kafka和ZooKeeper一般都会独立部署，所以在应用中都需要来为Kafka和ZooKeeper配置一些连接信息等。Kafka的整合与RabbitMQ不同，在Spring Boot 1.3.7中并没有直接提供的Starter模块，而是采用了Spring Cloud Stream的Kafka模块，所以对于Kafka的配置均采用了spring.cloud.stream.kafka的前缀，比如：

| **属性名** | **说明** | **默认值** |
| --- | --- | --- |
| spring.cloud.stream.kafka.binder.brokers | Kafka的服务端列表 | localhost |
| spring.cloud.stream.kafka.binder.defaultBrokerPort | Kafka服务端的默认端口，当brokers属性中没有配置端口信息时，就会使用这个默认端口 | 9092 |
| spring.cloud.stream.kafka.binder.zkNodes | Kafka服务端连接的ZooKeeper节点列表 | localhost |
| spring.cloud.stream.kafka.binder.defaultZkPort | ZooKeeper节点的默认端口，当zkNodes属性中没有配置端口信息时，就会使用这个默认端口 | 2181 |