# 1、kafka的安装

## 1.1 kafka的单机安装与简单使用

1、由于Kafka是用Scala语言开发的， 运行在JVM上， 因此在安装Kafka之前需要先安装JDK

2、kafka依赖zookeeper， 所以需要先安装zookeeper

3、下载安装包解压



4、启动服务

启动脚本语法：kafka­server­start.sh [-daemon] server.properties  
可以看到，server.properties的配置路径是一个强制的参数，­daemon表示以后台进程运行，否则ssh客户端退出后，就会停止服务。

(注意，在启动kafka时会使用linux主机名关联的ip地址，所以需要把主机名和linux的ip映射配置到本地host里，用vim /etc/hosts)



server.properties核心配置详解：

|  |  |  |
| --- | --- | --- |
| **Property** | **Default** | **Description** |
| broker.id | 0 | 每个broker都可以用一个唯一的非负整数id进行标识；这个id可以作为broker的“名字”，你可以选择任意你喜欢的数字作为id，只要id是唯一的即可。 |
| log.dirs | /tmp/kafka-logs | kafka存放数据的路径。这个路径并不是唯一的，可以是多个，路径之间只需要使用逗号分隔即可；每当创建新partition时，都会选择在包含最少partitions的路径下进行。 |
| listeners | 9092 | server接受客户端连接的端口 |
| zookeeper.connect | localhost:2181 | zooKeeper连接字符串的格式为：hostname:port，此处hostname和port分别是ZooKeeper集群中某个节点的host和port；zookeeper如果是集群，连接方式为 hostname1:port1, hostname2:port2, hostname3:port3 |
| log.retention.hours | 168 | 每个日志文件删除之前保存的时间。默认数据保存时间对所有topic都一样。 |
| min.insync.replicas | 1 | 当producer设置acks为-1时，min.insync.replicas指定replicas  的最小数目（必须确认每一个repica的写数据都是成功的），如果这个数目没有达到，producer发送消息会产生异常 |
| delete.topic.enable | false | 是否允许删除主题 |

使用脚本操作kafka

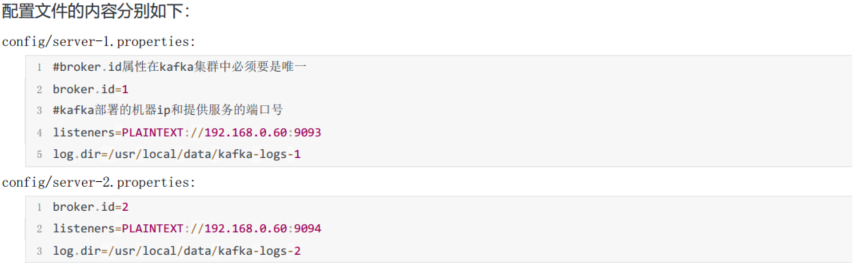






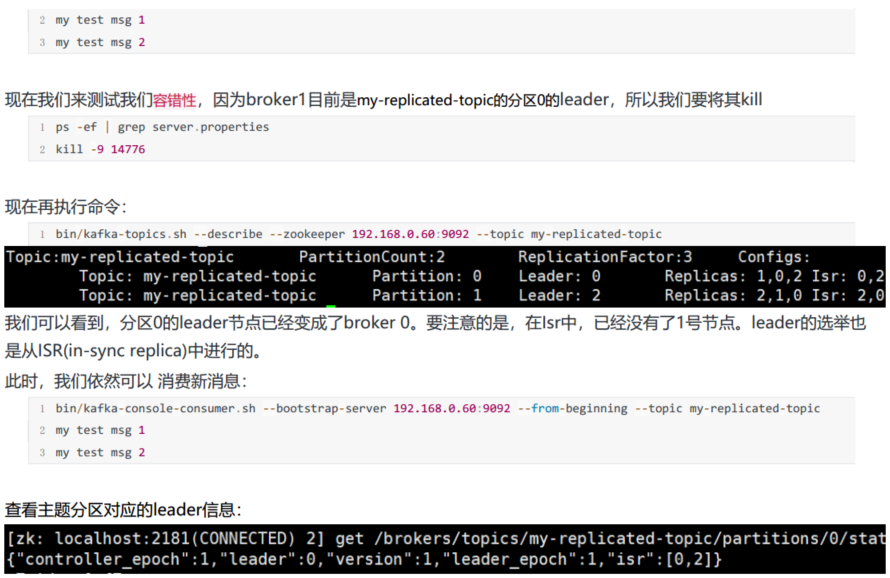
## 1.2 kafka集群配置

启用集群只需要再启动kafka节点即可，kafka集群通过zk来组织集群，使用同一个zk的kafka broker节点都归属于一个集群









# 2、kafka的使用

Kafka是最初由Linkedin公司开发，是一个分布式、支持分区的（partition）、多副本的（replica），基于zookeeper协调的分布式消息系统， 它的最大的特性就是可以实时的处理大量数据以满足各种需求场景：比如基于hadoop的批处理系统、低延迟的实时系统、 Storm/Spark流式处理引擎，web/nginx日志、访问日志，消息服务等等，用scala语言编写，Linkedin于2010年贡献给了Apache基金会并成为顶级开源项目

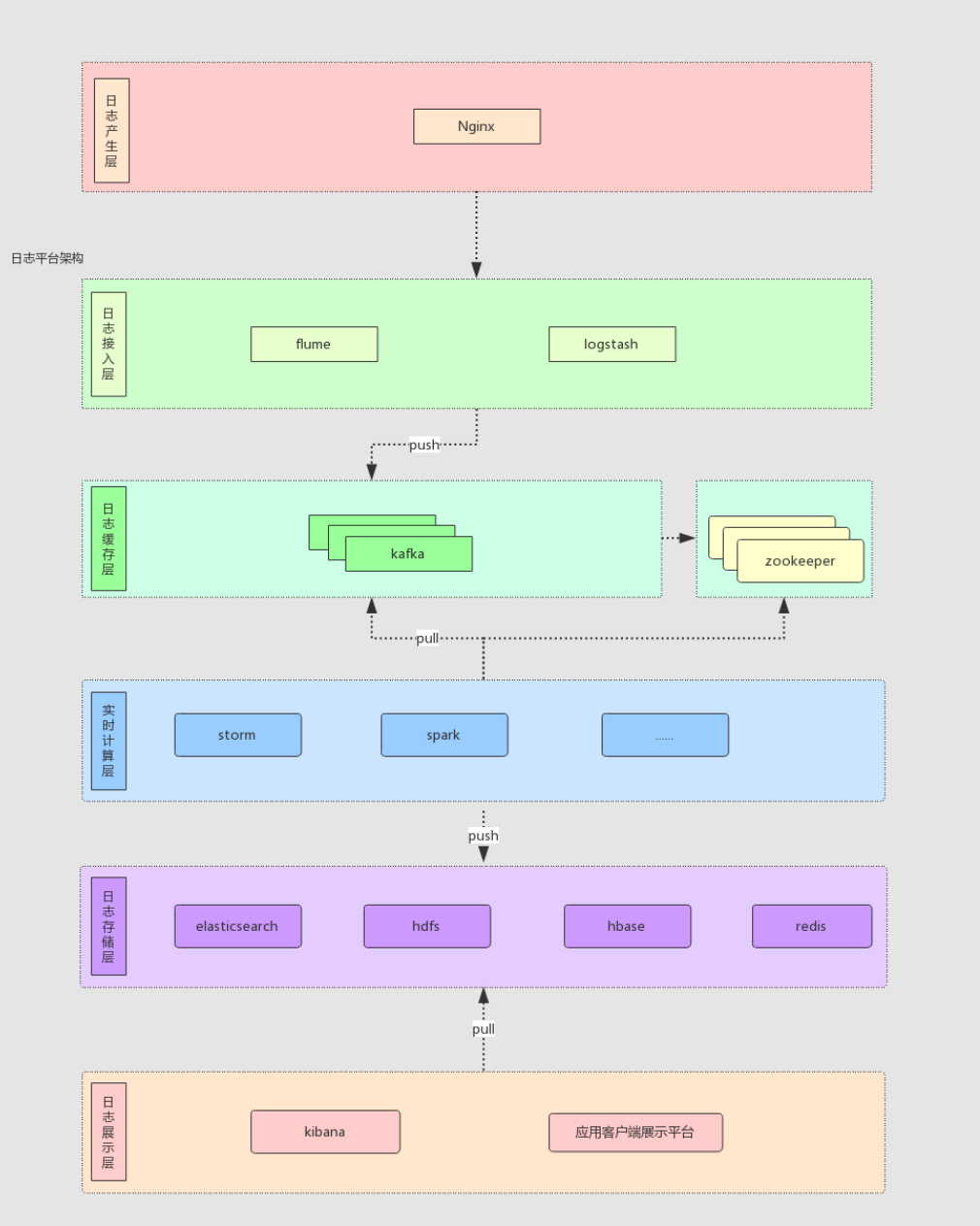
Kafka的使用场景

日志收集：一个公司可以用Kafka收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如hadoop、Hbase、Solr等。

消息系统：解耦和生产者和消费者、缓存消息等。

用户活动跟踪：Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者装载到hadoop、数据仓库中做离线分析和挖掘。

运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。



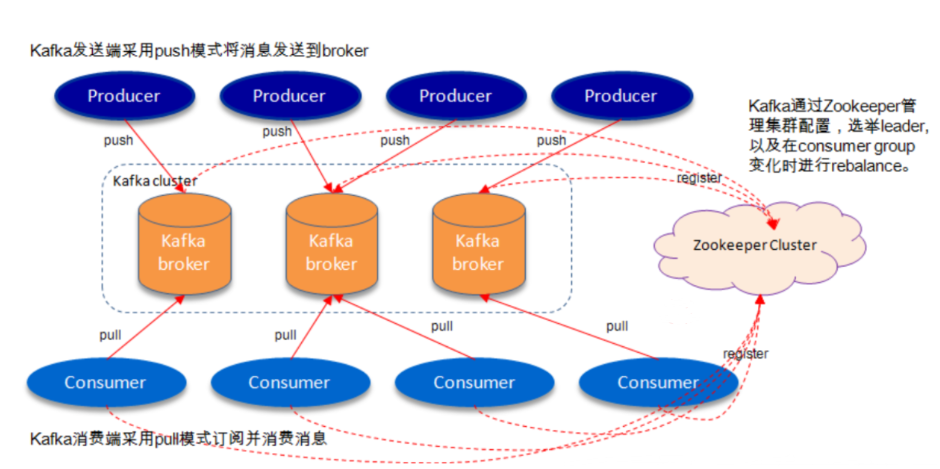
## 2.1 kafka的基本概念

kafka是一个分布式的，分区的消息(官方称之为commit log)服务。它提供一个消息系统应该具备的功能，但是确有着独特的设计。可以这样来说，Kafka借鉴了JMS规范的思想，但是确并没有完全遵循JMS规范。

首先，让我们来看一下基础的消息(Message)相关术语：

|  |  |
| --- | --- |
| **名称** | **解释** |
| Broker | 消息中间件处理节点，一个Kafka节点就是一个broker，一个或者多个Broker可以组成一个Kafka集群 |
| Topic | Kafka根据topic对消息进行归类，发布到Kafka集群的每条消息都需要指定一topic |
| Producer | 消息生产者，向Broker发送消息的客户端 |
| Consumer | 消息消费者，从Broker读取消息的客户端 |
| ConsumerGroup | 每个Consumer属于一个特定的Consumer Group，一条消息可以被多个不同Consumer Group消费，但是一个Consumer Group中只能有一个Consumer能够消费该消息，每个consumer持有各自的partition路由，当consumer挂掉后其持有的路由被分发给其他consumer |
| Partition | 物理上的概念，一个topic可以分为多个partition，每个partition内部消息是有序的，partition类似于rocketmq的topic下的队列 |

**Kafka的生产消费模型**



服务端(brokers)和客户端(producer、consumer)之间通信通过**TCP协议**来完成

kafka集群，在配置的时间范围内，维护所有的由producer生成的消息，而不管这些消息有没有被消费。例如日志保留( log retention )时间被设置为2天。kafka会维护最近2天生产的所有消息，而2天前的消息会被丢弃。kafka的性能与保留的数据量的大小没有关系，因此保存大量的数据(日志信息)不会有什么影响。

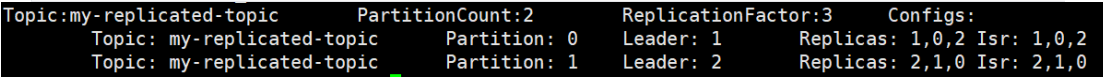
### 1.1 broker、topic、partition

#### Broker：

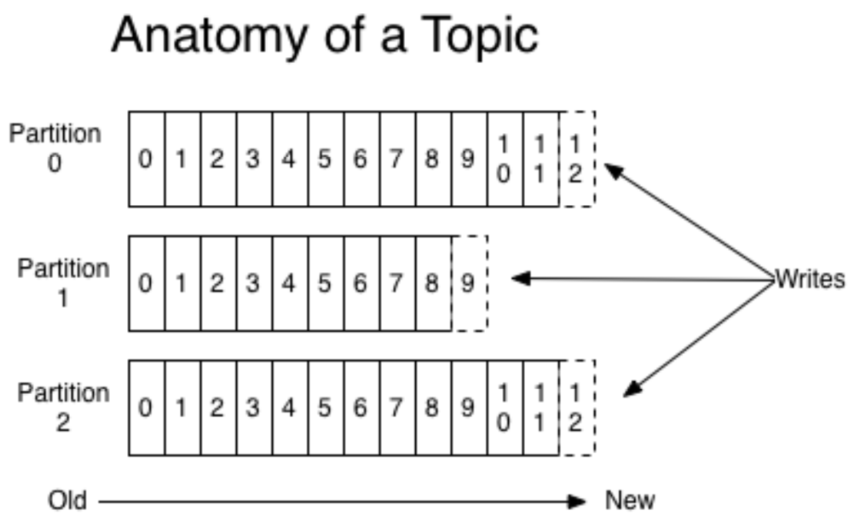
Broker即同rocketmq的broker，为kafka的一个节点进程，每个broker上可以备份其他broker上的partition副本，当主副本挂掉后从副本可以升级为主副本（这一功能比rocketmq强大）

#### Topic：

一个topic，代表逻辑上的一个业务数据集，比如按数据库里不同表的数据操作消息区分放入不同topic，订单相关操作消息放入订单topic，用户相关操作消息放入用户topic，对于大型网站来说，后端数据都是海量的，订单消息很可能是非常巨量的，比如有几百个G甚至达到TB级别，如果把这么多数据都放在一台机器上可定会有容量限制问题，那么就可以在topic内部划分多个partition来分片存储数据，不同的partition可以位于不同的机器上，每台机器上都运行一个Kafka的进程Broker。创建topic时指定partition数量，指定每个副本数量，副本会均匀的分布在所有的broker上，副本数量中包含主副本，从副本在主副本挂掉后可以变为主副本



以下是输出内容的解释，第一行是所有分区的概要信息， 之后的每一行表示每一个partition的信息。  
leader节点负责给定partition的所有读写请求。Partition 0的leader为1，代表该分片的主副本在broker 1 这个节点上  
replicas 表示某个partition在哪几个broker上存在备份。 不管这个几点是不是” leader“， 甚至这个节点挂了，也会列出。  
isr 是replicas的一个子集，它只列出当前还存活着的，并且已同步备份了该partition的节点



#### Parttition：

Partition是一个有序的message序列，这些message按顺序添加到一个叫做commit log的文件中。每个partition中的消息都有一个唯一的编号，称之为offset，用来唯一标示某个分区中的message。

提示：每个partition，都对应一个commit log文件。一个partition中的message的offset都是唯一的，但是不同的partition中的message的offset可能是相同的。

为什么要对Topic下数据进行分区存储？

1、commit log文件会受到所在机器的文件系统大小的限制，分区之后，理论上一个topic可以处理任意数量的数据。

2、为了提高并行度。

**分布式distribution**

log的partitions分布在kafka集群中不同的broker上，每个broker可以请求备份其他broker上partition上的数据。kafka集群支持配置一个partition备份的数量。

针对每个partition，都有一个broker起到“leader”的作用，0个或多个其他的broker作为“follwers”的作用。leader处理所有的针对这个partition的读写请求，而followers被动复制leader的结果。如果这个leader失效了，其中的一个follower将会自动的变成新的leader。

### 1.2 producer、consumer

#### Producer：

生产者将消息发送到topic中去，同时负责选择将message发送到topic的哪一个partition中。通过round-robin做简单的负载均衡。也可以根据消息中的某一个关键字来进行区分。通常第二种方式使用的更多。

#### Consumer：

每个consumer是基于自己在commit log中的消费进度(offset)来进行工作的。在kafka中，消费offset由consumer自己来维护；一般情况下我们按照顺序逐条消费commit log中的消息，当然我可以通过指定offset来重复消费某些消息，或者跳过某些消息。

这意味kafka中的consumer对集群的影响是非常小的，添加一个或者减少一个consumer，对于集群或者其他consumer来说，都是没有影响的，因为每个consumer维护各自的offset。所以说kafka集群是无状态的，性能不会因为consumer数量受太多影响。kafka还将很多关键信息记录在zookeeper里，保证自己的无状态，从而在水平扩容时非常方便。

传统的消息传递模式有2种：队列( queue) 和（publish-subscribe）

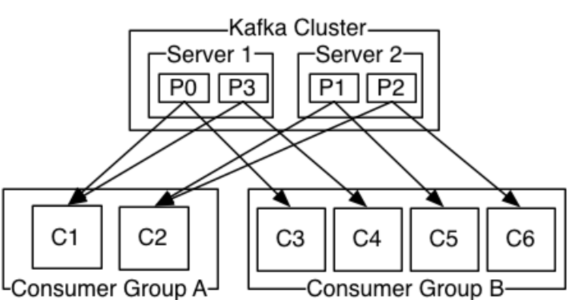
queue模式：多个consumer从服务器中读取数据，消息只会到达一个consumer。

publish-subscribe模式：消息会被广播给所有的consumer。

Kafka基于这2种模式提供了一种consumer的抽象概念：consumer group。

queue模式：所有的consumer都位于同一个consumer group 下。

publish-subscribe模式：所有的consumer都有着自己唯一的consumer group。Kafka集群内的每个consumer group都会接收到该topic下的所有消息，然后再分发给分组内的各个消费者，正常情况下每个消费者存在一个/组partition的路由，如果分组内只有一个消费者则其会接受到topic下的所有消息



上图说明：由2个broker组成的kafka集群，总共有4个partition(P0-P3)。这个集群由2个Consumer Group，A有2个consumer instances，B有四个。  
通常一个topic会有几个consumer group，每个consumer group都是一个逻辑上的订阅者（logical subscriber）。每个consumer group由多个consumer instance组成，从而达到可扩展和容灾的功能。

**顺序消费**：

Kafka比传统的消息系统有着更强的顺序保证。  
一个partition同一个时刻在一个consumer group中只有一个consumer instance在消费，从而保证顺序。consumer group中的consumer instance的数量不能比一个Topic中的partition的数量多，否则，多出来的consumer消费不到消息。

Kafka只在partition的范围内保证消息消费的局部顺序性，不能在同一个topic中的多个partition中保证总的消费顺序性。

如果有在总体上保证消费顺序的需求，那么我们可以通过将topic的partition数量设置为1 ， 将consumer group中的consumer instance数量也设置为1 。

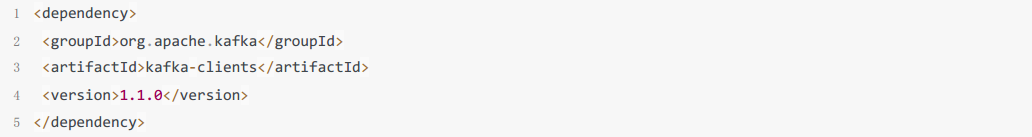
从较高的层面上来说的话，Kafka提供了以下的保证：  
发送到一个Topic中的message会按照发送的顺序添加到commit log中。意思是，如果消息 M1，M2由同一个producer发送，M1比M2发送的早的话，那么在commit log中，M1的offset就会比commit 2的offset小。一个consumer在commit log中可以按照发送顺序来消费message。

如果一个topic的备份因子设置为N，那么Kafka可以容忍N-1 个服务器的失败， 而存储在commit log中的消息不会丢失。

## 2.2 java 整合kafka

### 2.1 spring整合kafka

引入maven依赖

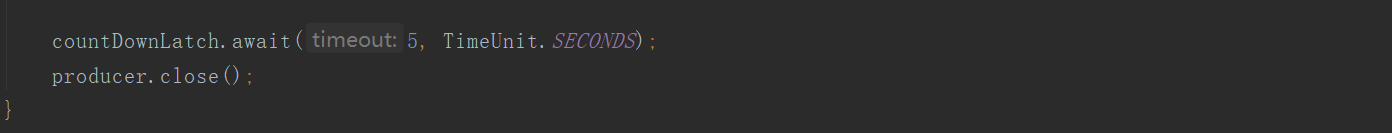


客户端代码





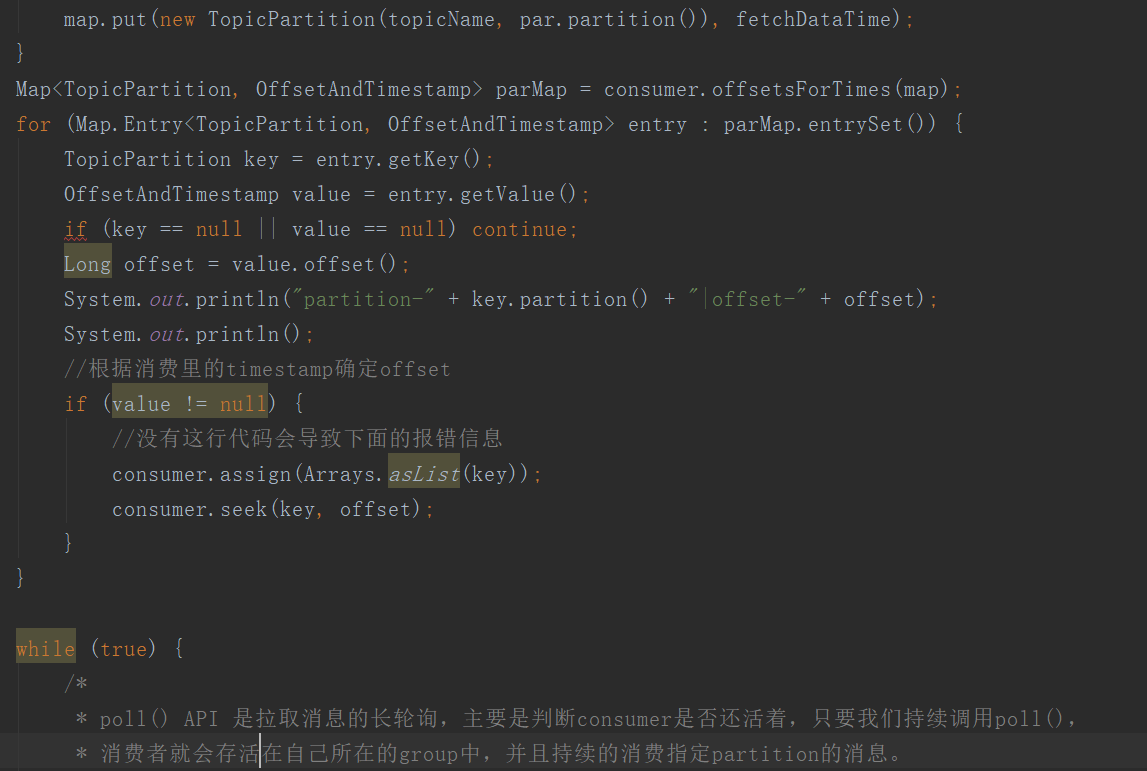


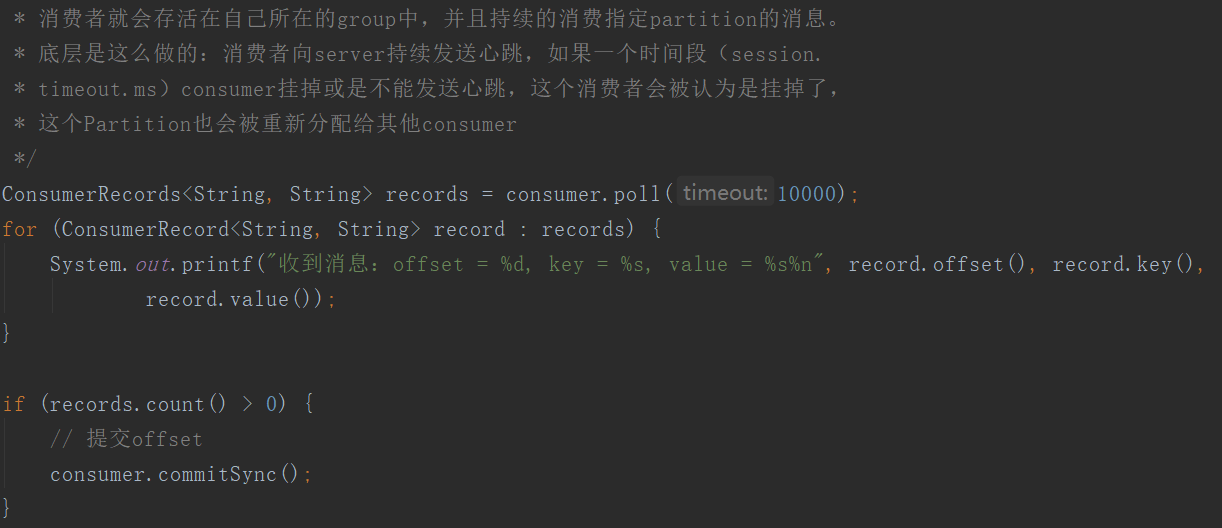


消费端代码



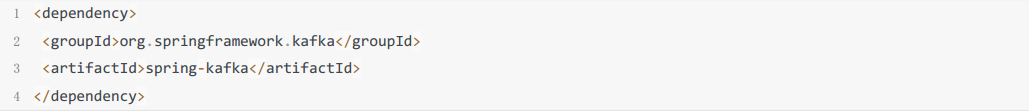






### 2.2 spring boot 整合kafka

1、引入依赖



2、application.yml配置如下：



发送者代码



消费者代码

