Kafka 总结

目录

Kafka 总结 1
一、 Kafka 使用场景 3
二、 Kafka 相关概念 3
三、 Zookeeper 集群搭建 4
四、 Kafka 集群搭建 5
五、 kafka 基础 5
六、 Kafka 消息处理与集群维护 8
七、 Kafka 集群监控 12
八、 核心代码剖析 14
九、 kafka 设计模型 17
十、 Kafka Partition Leader 选主机制 19

一、 Kafka 使用场景

1. 对于一些常规的消息系统,kafka 是个不错的选择;partitons/replication 和容错,可以使 kafka 具有良好的扩展性和性能优势.不过到目前为止,我们应该很清楚认识到,kafka 并没有提供 JMS 中的"事务性""消息传输担保(消息确认机制)""消息分组"等企业级特性;kafka 只能使用作为"常规"的消息系统,在一定程度上,尚未确保消息的发送与接收绝对可靠(比如,消息重发,消息发送丢失等)
2. kafka 可以作为"网站活性跟踪"的最佳工具;可以将网页/用户操作等信息发送到 kafka 中.
3. kafka 的特性决定它非常适合作为"日志收集中心";application 可以将操作日志"批量""异步"的发送到 kafka 集群中,而不是保存在本地或者 DB 中;kafka 可以批量提交消息/压缩消息等,这对 producer 端而言,几乎感觉不到性能的开支.此时 consumer 端可以使 hadoop 等其他系统化的存储和分析系统.

目前使用较多的分布式消息系统对比:

	ActiveMQ	RabbitMQ	Kafka
所属社区/公司	Apache	Mozilla Public License	Apache/LinkedIn
开发语言	Java	Erlang	Scala
支持的协议	OpenWire、STOMP、REST、XMPP、AMQP	AMQP	仿AMQP
事务	支持	不支持	不支持
集群	支持	支持	支持
负载均衡	支持	支持	支持
动态扩容	不支持	不支持	支持(zk)

二、 Kafka 相关概念

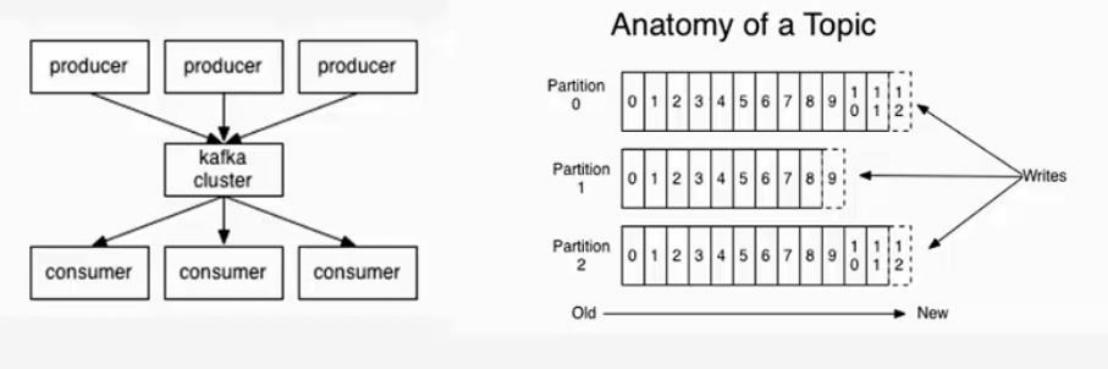
1. AMQP 协议

- 消费者(Consumer):从消息队列中请求消息的客户端程序
- 生产者(Producer):向 broker 发布消息的客户端程序
- AMQP 服务端(broker):用来接收生产者发送的消息,并将这些消息路由给服务器中的队列

2. Kafka 支持的客户端语言

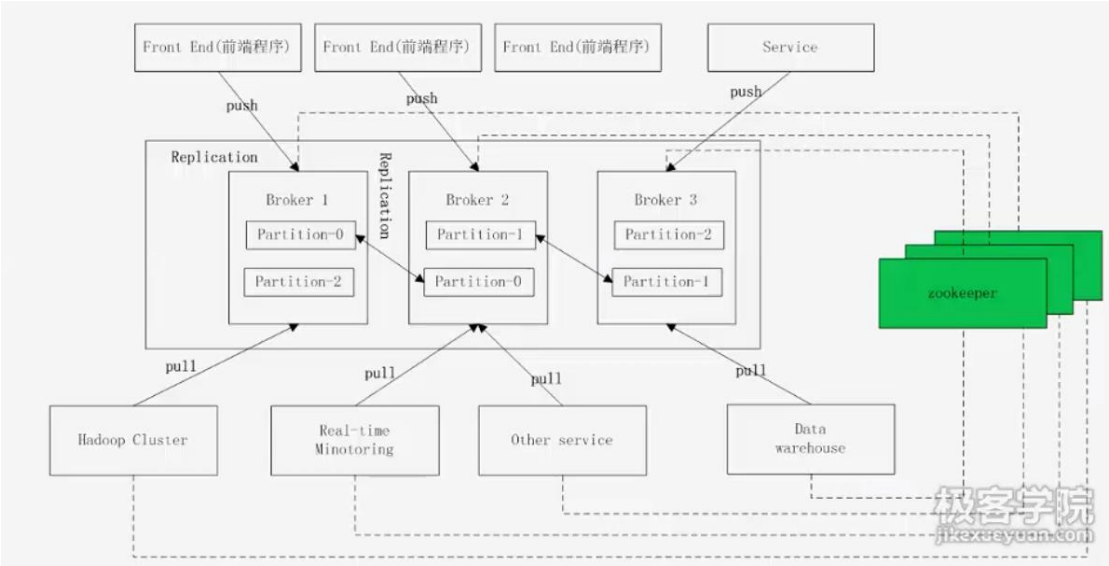
C,C++,Erlang,Java,.net,perl,Php,Python,Ruby,Go,JavaScript

3. Kafka 架构



一些基本概念:

- 主题(Topic):一个主题类似新闻中的体育,娱乐,教育等分类概念,在实际工程中通常一个业务一个主题;
- 分区(Partition):一个 Topic 中的消息数据按照多个分区组织,分区是 kafka 消息队列组织的最小单位,一个分区可以看做是一个 FIFO 的队列.



- 备份(Replication):为了保证分布式可靠性,kafka0.8 开始对每个分区的数据进行备份(不同 Broker 上),防止其中一个 Broker 宕机造成分区数据不可用
- Zookeeper:一个提供分布式状态管理,分布式配置管理,分布式锁服务等的集群

三、 Zookeeper 集群搭建

软件环境:

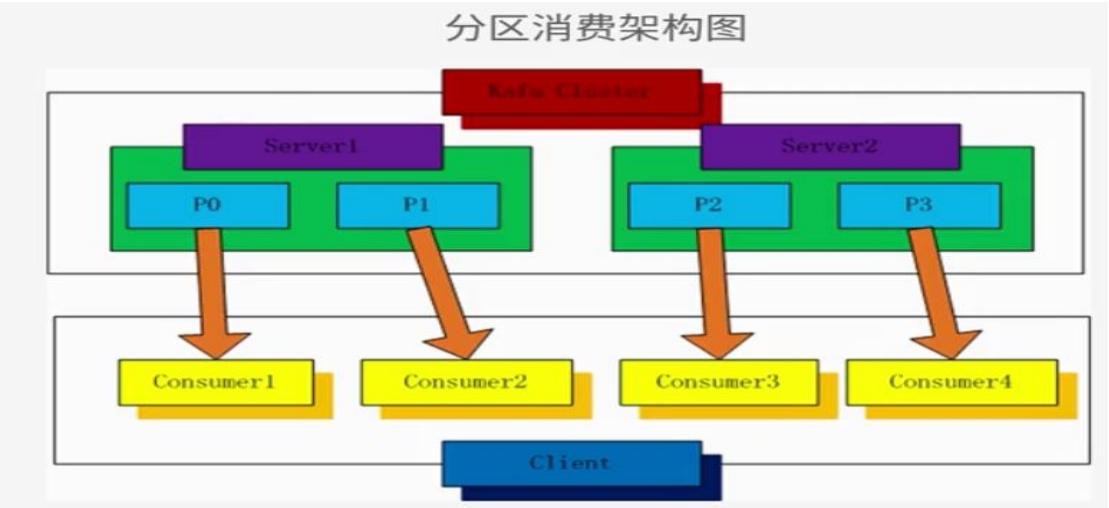
- Linux 服务器一台,三台,五台(2*n+1 台),一般是奇数台搭建,保证半数服务存活,集群的高可用
- Java jdk1.7
- Zookeeper3.4.6 版(稳定版,使用最多的版本)

四、 Kafka 集群搭建

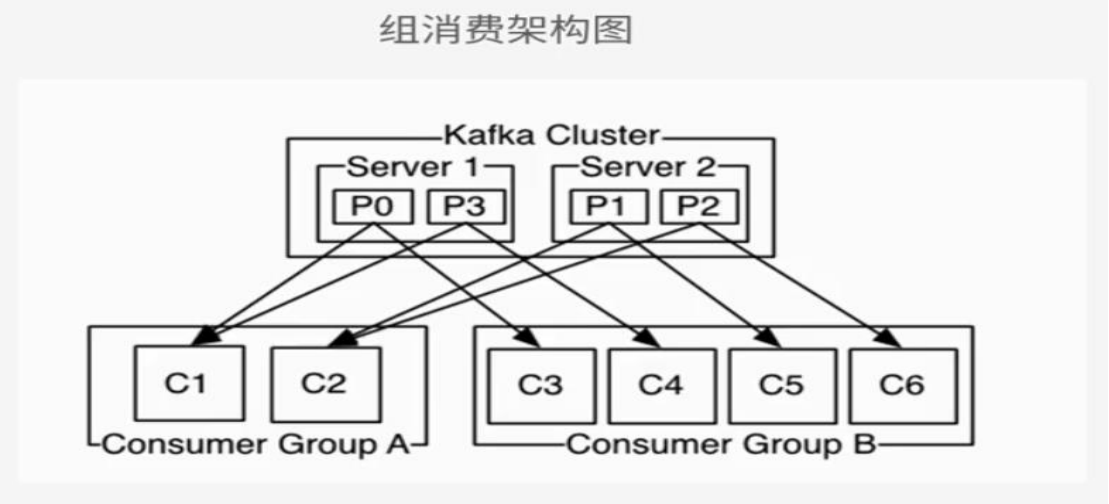
1. 软件环境:
- Linux 服务器一台或多台;
- 已经搭建好的 zookeeper 集群;
- Kafka_2.9.2-0.8.1.1;
2. 启动 kafka 集群命令:
   kafka-server-start.sh –daemon $KAFKA_HOME/config/server.properties
3. 重要的配置文件:
   server.properties 文件;
   consumer.properties 文件:
   producer.properties 文件:

五、 kafka 基础

1. kafka 消费者编程模型
   - 分区消费模型



- 组消费模型



按组(Group)消费伪代码描述

```
main()
  设置需要创建的流数N
  for index =0 to N
    create thread  consumer(Index)

第index个线程
consumer(index)
   创建到kafka broker的连接: KafkaClient(host,port)
   指定消费参数构建consumer: SimpleConsumer(topic, partitions)
   设置从头消费还是从最新的消费(smallest或largest)
   while True
       从指定topic的第index个流取数据
       处理
     (offset会自动提交到zookeeper,无需我们操作)
```

Consumer分配算法

```
1. For each topic T that Ci subscribes to
2. let PT be all partitions producing topic T
3. let CG be all consumers in the same group as Ci that consume topic T
4. sort PT (so partitions on the same broker are clustered together)
5. sort CG
6. let i be the index position of Ci in CG and let N = size(PT)/size(CG)
7. assign partitions from i*N to (i+1)*N - 1 to consumer Ci
8. remove current entries owned by Ci from the partition owner registry
9. add newly assigned partitions to the partition owner registry
   (we may need to re-try this until the original partition owner releases
   its ownership)
```

- 两种消费模型对比

  分区消费模型相对灵活,但是:

  1) 需要自己处理各种异常
  2) 需要自己管理 offset(以实现消息传递的其他语义)

组消费模型更加简单,但不灵活:

1) 不需要自己处理各种异常
2) 不需要自己管理 offset
3) 只能实现 kafka 默认的最少一次消息传递语义

2. kafka 消费者客户端实现

软件环境:

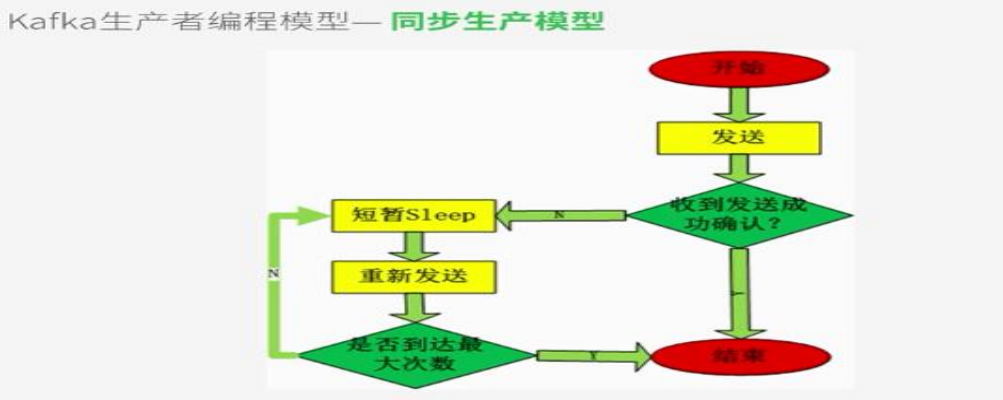
已经搭建好的 kafka 集群,Linux 服务器一台,Apache Maven3.2.3,kafka0.8.1

Java 客户端调优:

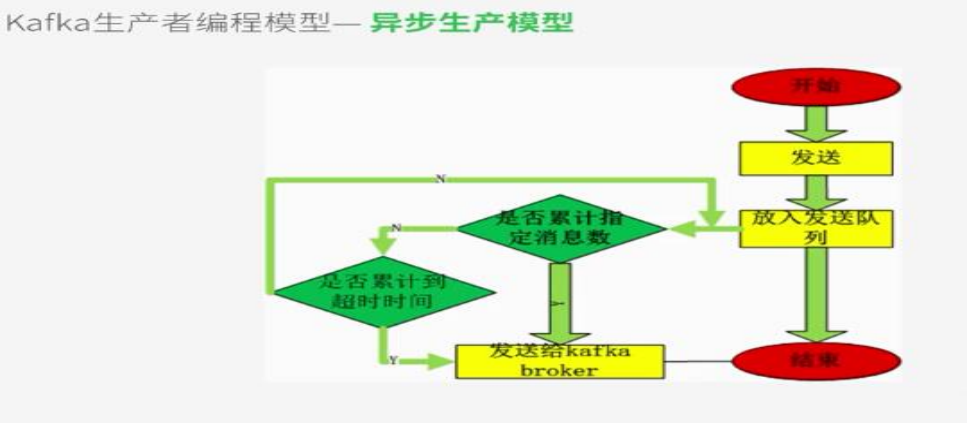
- fetchSize:从服务器获取单包大小
- bufferSize:kafka 客户端缓冲区大小
- group.id:分组消费时分组名

3. kafka 生产者编程模型

- 同步生产模型



- 异步生产模型



- 伪代码描述

- 两种生产模型对比

  同步生产模型:

  1) 低消息丢失率
  2) 高消息重复率(由于网络原因,回复确认未接收到)
  3) 高延迟

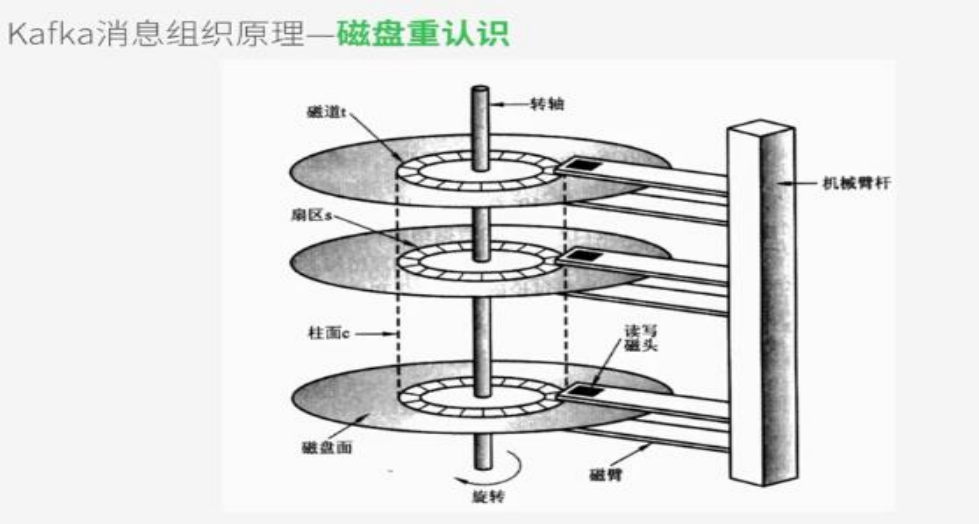
  异步生产模型:

  1) 低延迟
  2) 高发送性能
  3) 高消息丢失率(无确认机制,发送端队列满)

4. kafka 生产者客户端实现

六、 Kafka 消息处理与集群维护

1. Kafka 消息组织原理
   - 磁盘重认识



当需要从磁盘读取数据时,要确定读的数据在哪个磁道,哪个扇区:

1) 首先必须找到柱面,即磁头要移动对准相应磁道,这个过程叫寻道,所耗费的时间为寻道时间.
2) 然后目标扇区旋转到磁头下,这个过程耗费的时间为旋转时间

一次访问磁盘(读/写)完成过程由三个动作组成:

1) 寻道时间:磁头移动定位到指定磁道
2) 旋转延迟时间:等待指定扇区从磁头下旋转经过
3) 数据传输时间:数据在磁盘,内存与网络之间的实际传输

由于存储介质的特性,磁盘存取就比主存慢,再加上机械运动消耗,磁盘的存取速度网网是主存的主存百分之一甚至是千分之一

- Kafka 消息的写入原理

1) 一般的将数据从文件传到套接字的路径:
2) 操作系统将数据从磁盘读到内核空间的页缓存中;
3) 应用将数据从内核空间读到用户空间的缓存中;
4) 应用将数据写回内存空间的套接字缓存中;
5) 操作系统将数据从套接字缓存写到网卡缓存中,以便将数据经网络发出;

这样做明显是低效的,这里有四次拷贝,两次系统调用.如果使用 sendfile(Java 为:FileChannel.transferTo api),两次拷贝可以被避免;允许操作系统将数据直接从页缓存发送到网络上.优化后,只有最后一步将数据拷贝到网卡缓存中是需要的.

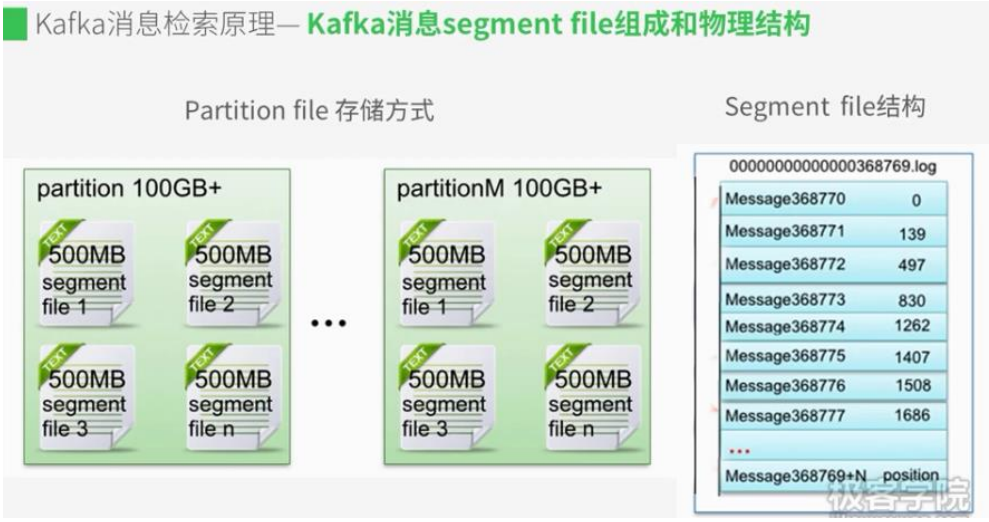
- Kafka 消息的删除原理

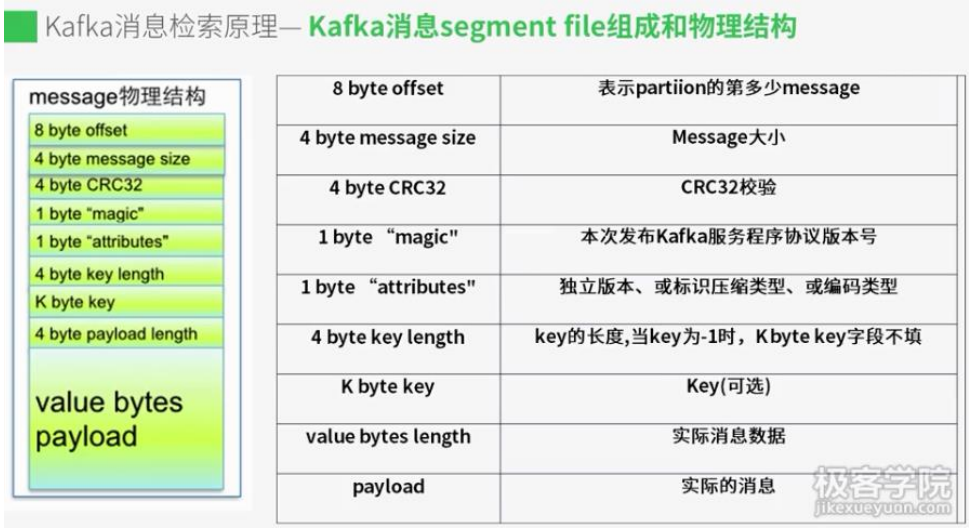
从最久的日志段开始删除(按日志段为单位进行删除),然后逐步向前推进,直到某个日志段不满足条件为止,删除条件:

1) 满足给定条件 predicate(配置项 log.retention.{ms,minutes,hours}) 和 log.retention.bytes 指定);
2) 不能是当前激活日志段;
3) 大小不能小于日志段的最小大小(配置项 log.segment.bytes 配置)
4) 要删除的是否是所有日志段,如果是的话直接调用 roll 方法进行切分因为 kafka 至少要保留一个日志段;

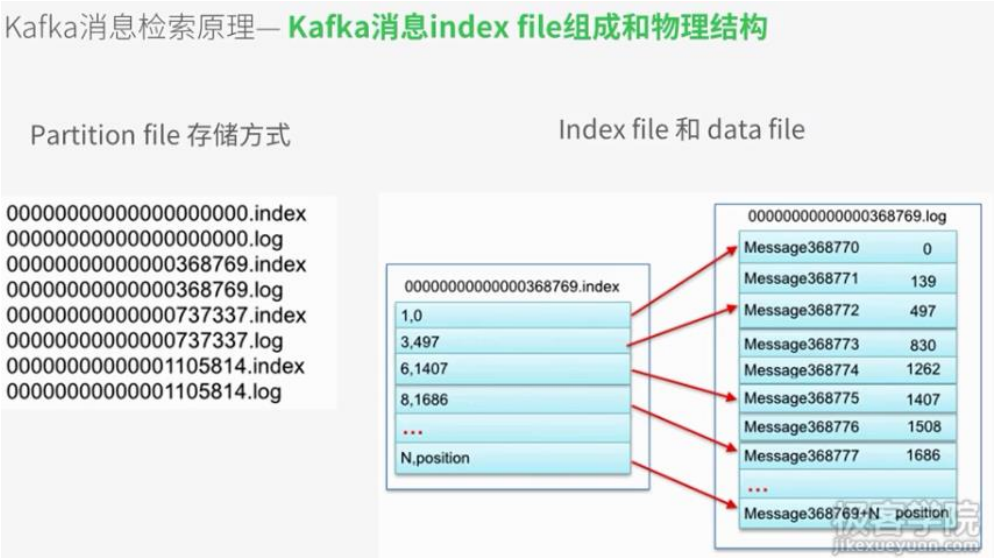
2. Kafka 消息检索原理

- Kafka 消息的 segment file 的组成和物理结构





- Kafka 消息的 index file 的组成和物理结构



3. Kafka 集群维护

- 集群信息实时查看(topic 工具):

1) 列出集群当前所有可用的 topic:

bin/kafka-topics.sh --list --zookeeper zookeeper_address

```
[test@ kafka_2.9.2-0.8.1.1]$ bin/kafka-topics.sh --list --zookeeper 10.206.212.14:12181
jiketest
test
```

2) 查看集群特定 topic 信息:

bin/kafka-topics.sh --describe --zookeeper zookeeper_address --topic topic_name

```
[test@ kafka_2.9.2-0.8.1.1]$ bin/kafka-topics.sh --describe --zookeeper 10.206.212.14:12181 --topic jiketest
Topic:jiketest  PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: jiketest Partition: 0    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: jiketest Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: jiketest Partition: 2    Leader: 1       Replicas: 1,2   Isr: 1,2
```

- 集群信息实时修改(topic 工具):

1) 创建 topic:

bin/kafka-tppics.sh --create --zookeeper zookeeper_address --replication-factor 1 --partitions 1 --topic topic_name

2) 增加(不能减少)partition(最后的 4 是增加后的值) :

bin/kafka-tppics.sh --zookeeper zookeeper_address --alter --topic topic_name --partitions 4

Topic –level configuration 配置都能修改

- 集群分区日志迁移

迁移 topic 数据到其他 broker,请遵循下面四步:

1) 写 json 文件,文件格式如下:

cat topics-to-move.json

{“topics”:[{“topic”:”fool”},

{“topic”:”foo2”}

],

“version”:1

}

2) 使用-generate 生成迁移计划(下面的操作是将 topic:foo1 和 foo2 移动到 broker5,6);

bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-

json-file topics-to-move.json --broker-list “5,6” –generate
这一步只是生成计划,并没有执行数据迁移;

3) 使用-execute 执行计划:
bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file expand-cluster-reassignment.json –execute
执行前最好保存当前的分配情况,以防出错回滚

4) 使用-verify 验证是否已经迁移完成
迁移摸个 topic 的某些特定的 partition 数据到其他 broker,步骤与上面一样,但是 json 文件如下面所示:
cat custom-reassignment.json
{"verson":1,"partitions":[{"topic":"foo1","partition":0,"replicas":[5,6]},
{"topic":"foo2","partition":1,"replicas":[2,3]},
]}
可以指定到 topic 的分区编号

5) kafka-reassign-partitions.sh 工具会复制磁盘上的日志文件,只有当完全复制完成,才会删除迁移前磁盘上的日志文件.执行分区日志迁移需要注意:
   - kafka-reassign-partitions.sh 工具的粒度只能到 broker,不能到 broker 的目录(如果不让 broker 上面配置了多个目录,是按照磁盘上面已驻留的分区数来均匀分配的),所以,如果 topic 之间的数据,或者 topic 的 partition 之间的数据本身就不均匀,很有可能造成磁盘数据的不均匀;
   - 对于分区数据较多的分区迁移数据会花大量的时间,所以建议在 topic 数据量较少或磁盘有效数据较少的情况下执行数据迁移操作;

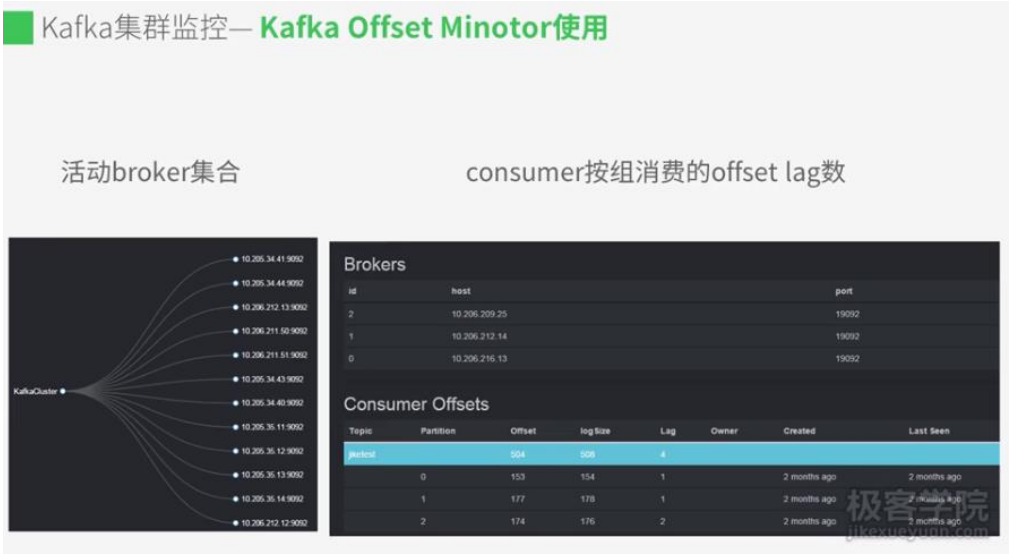
6) 进行分区迁移时最好先保留一个分区在原来的磁盘,这样不会影响正常的消费和生产,如果目的是将分区 5(broker1,5)迁移到 borker2,3.可以先将 5 迁移到 2,1,最后再迁移到 2,3.而不是一次将 1,5 迁移到 2,3.因为一次迁移所有的副本,无法正常消费和生产,部分迁移则可以正常消费和生产.

七、 Kafka 集群监控

1. kafka offset monitor 介绍
在生产环境需要集群高可用,所以需要对 kafka 集群进行监控.kafka offset monitor 可以监控 kafka 集群以下几项:
   1) kafka 集群当前存活的 broker 集合;
   2) kafka 集群当前活动 topic 集合;
   3) 消费者组列表;
   4) kafka 集群当前 consumer 按组消费的 offset lag 数(即当前 topic 当前分区目前有多少消费积压而没有及时消费);

2. kafka offset monitor 使用
部署 kafka offset monitor:
github 下载 jar 包 KafkaOffsetMonitor-assembly-0.2.0.jar:
https://github.com/quantifind/KafkaOffsetMonitor/releases
启动 kafka offset minotor:
java -cp KafkaOffsetMonitor-assembly-0.2.0.jar
com.quantifind.kafka.offsetapp.OffsetGetterWeb --zk zk-01,zk-02 --refresh

5.minutes --retain 1.day&



3. kafka manager 介绍

   kafka manager 由雅虎开源,提供一下功能:

   1) 管理几个不同的集群;
   2) 容易地检查集群的状态(topics,brokers,副本的分布,分区的分布);
   3) 选择副本
   4) 基于集群的当前状态产生分区分配
   5) 重新分配分区

4. kafka manager 使用

   kafka manager 的安装,方法一(不但要求能上网,还要求能翻墙):

   安装 sbt:

   http://www.scala-sbt.org/download.html

   下载后,解压并配置环境变量(将 SBT_HOME/bin 配置到 PATH 变量中)

   安装 kafka manager:

   1) git clone https://github.com/yahoo/kafka-manager
   2) cd kafka-manager
   3) sbt clean dist

   部署 kafka manager

   1) 修改 conf/application.conf,把 kafka-manager.zkhosts 改为自己的 zookeeper 服务器地址
   2) bin/kafka-manager -Dconfig.file=conf/application.conf -Dhttp.port=8007&

   kafka manager 的安装,方法二:

   1) 下载打包好的 kafka manager(也可以在课程源代码处下载):

   https://github.com/scootli/kafka-manager-1.0-SNAPAHOT/tree/master/kafka-manager-1.0-SNAPASHOT

   2) 下载后解压

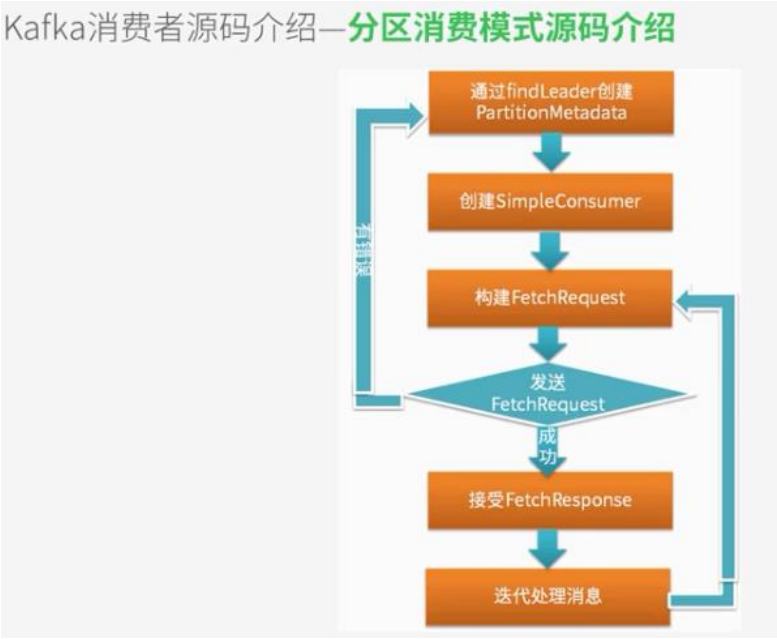
   修改 conf/application.conf,把 kafka-manager.zkhosts 改为自己的 zookeeper 服务器地址

   bin/kafka-manager -Dconfig.file=conf/application.conf -Dhttp.port=8007&

八、 核心代码剖析

1. kafka 消费者源码介绍

- 分区消费模式源码介绍:

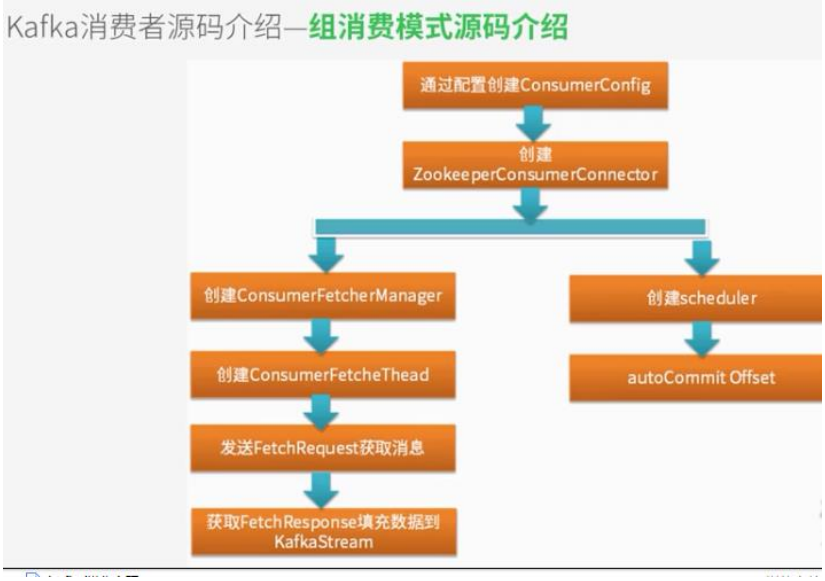


分区消费者模式直接由客户端(任何高级语言编写)使用 kafka 踢狗的协议向服务器发送 PRC 请求获取数据,服务器接收到客户端的 PRC 请求后将数据构造成 PRC 响应,返回给客户端,客户端解析相应的 PRC 相应获取数据.

kafka 支持的协议众多,使用比较重要的有:

1) 获取消息的 FetchRequest 和 FetchResponse
2) 获取 offset 的 OffsetRequest 和 OffsetResponse
3) 提交 offset 的 OffsetCommitRuest 和 OffsetCommitResponse
4) 获取 Metadata 的 Metadata Request 和 Metadata Response
5) 生产消息的 ProducerRequest 和 ProducerResponse

- 组消费模式源码介绍



- 两种消费模糊服务器端源码对比

  分区消费模式具有以下特点:

  1) 指定消费 topic,partiton 和 offset 通过向服务器发送 RPC 请求进行消费;
  2) 需要自己提交 offset;
  3) 需要自己处理各种错误,如:leader 切换错误
  4) 需要自己处理消费者负载均衡策略

  组消费模式具有以下特点:

  1) 最终也是通过向服务器发送 RPC 请求完成的(和分区消费模式一样);
  2) 组消费模式由 kafka 服务器端处理各种错误,然后将消息放入队列再封装为迭代器(队列为 FetchedDateChunk 对象),客户端只需在迭代取出消息;
  3) 由 kafka 服务器端周期性的通过 scheduler 提交当前消费的 offset,无需客户端

负责

4) kafka 服务器端处理消费者负载均衡

5) 监控工具 kafka offset monitor 和 kafka manager 均是基于组消费模式

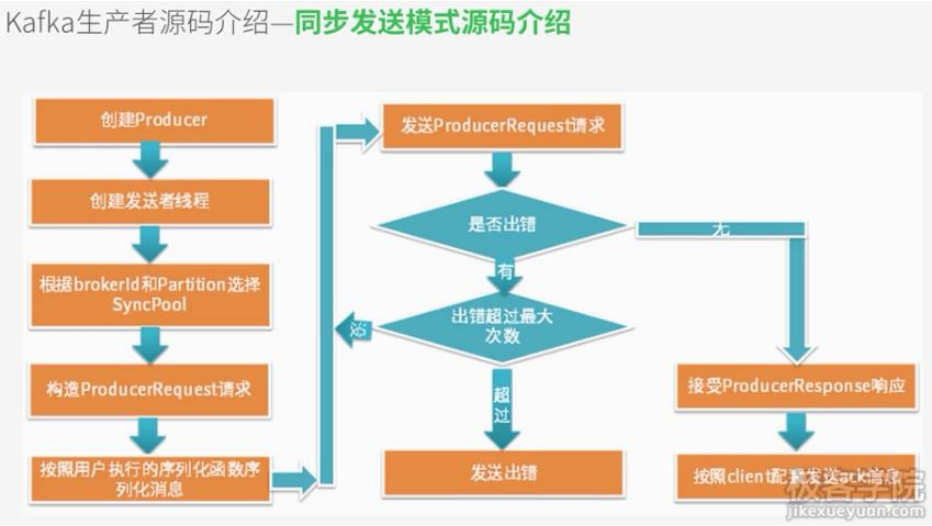
所以,尽可能使用组消费模式,除非你需要:

1) 自己管理 offset(比如为了实现消息投递的其他语义);

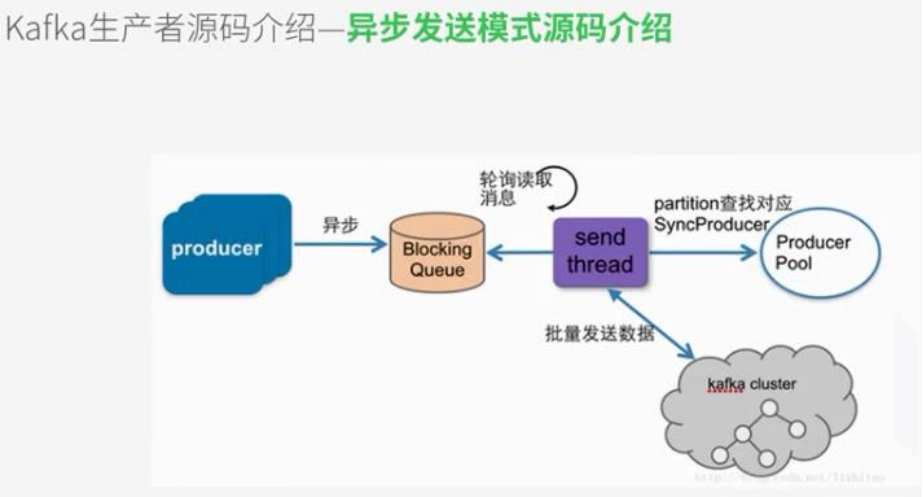
2) 自己处理各种错误(根据自己业务的需求)

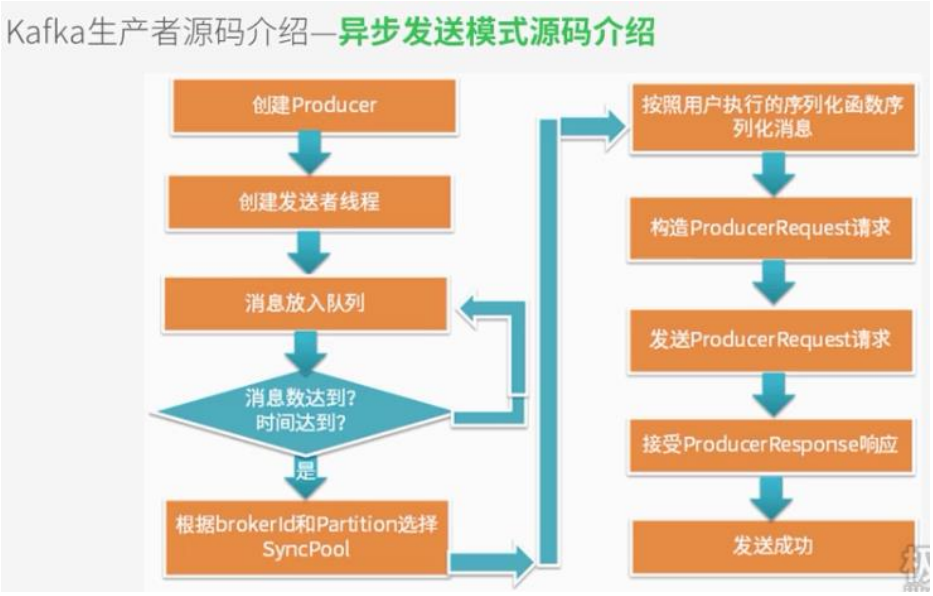
2. kafka 生产者源码介绍

- 同步发送模式源码介绍



- 异步发送模式源码介绍





- 两种生产模式服务器端源码对比

  同步发送模式具有以下特点:

  1) 同步的向服务器发送 RPC 请求进行生产;
  2) 发送错误可以重试;
  3) 可以向客户端发送 ack;

  异步发送模式具有以下特点:

  1) 最终也是通过向服务器发送 RPC 请求完成的(和同步发送模式一样)
  2) 异步发送模式先将一定量消息放入队列中,待达到一定数量后再一起发送
  3) 异步发送模式不支持发送 ack,但是 client 可以调用回调函数获取发送结果

  所以,性能比较高的场景使用异步发送,准确性要求高的场景使用同步发送.

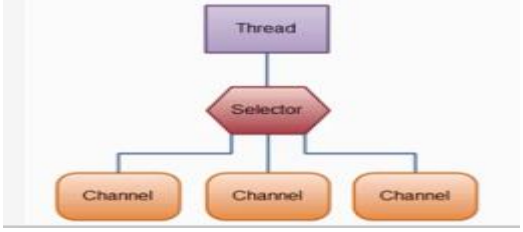
九、 kafka 设计模型

1. 认识 Java Nio

   Channels,Buffers,Selectors

   - Channel(通道)和 java 中的 streamyiyang ,用于传输数据的数据流,数据可以从 Channel 读到 Buffer 中,也可以从 Buffer 写到 Channel 中.
   - Selector 允许单线程处理多个 Channel.使用 Selector,首先得向 Selector 注册 hannel,然后调用它的 select()方法,此方法会一直阻塞到某个注册的 Channel 有事件就绪.一旦这个方法返回,线程就可以处理这些事件,事件的例子如新连接进了,数据接收等.

   下图为一个单线程中使用一个 Selector 处理 3 个 Channel:



2. 认识 Linux epoll 模型

epoll 是一种 IO 多路复用技术,在 linux 内核中广泛使用.常用的三种 IO 多路复用技术为 select 模型,poll 模型和 epoll 模型.

- select 模型需要轮询所有的套接字查看是否有事件发生.缺点:

(1) 套接字最大支持 1024 个;
(2) 主动轮询效率很低;
(3) 事件发生后需要将套接字从内核空间拷贝到用户空间,效率低

- poll 模型和 select 模型原理一样,但是修正了 select 模型最大套接字限制的缺点;
- epoll 模型修改主动轮询为被动通知,当有事件发生时,被动接收通知.所以 epoll 模型注册套接字后,主程序可以做其他事情,当事件发生时,接收到通知后再去处理,修改了 select 模型的三个缺点(第三点使用共享内存修正)

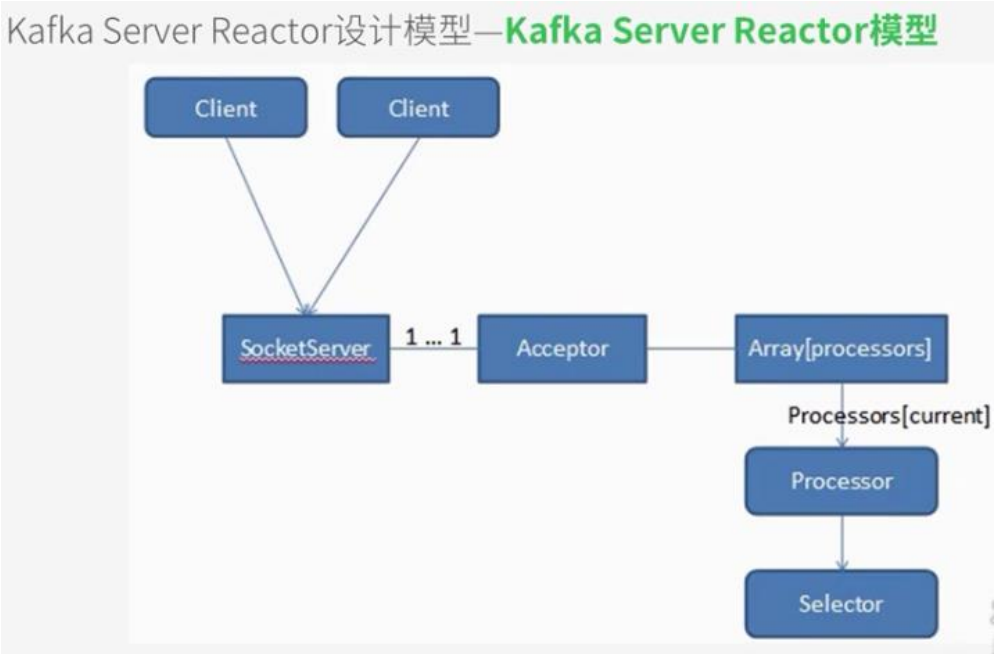
Java Nio 的 Selector 模型底层使用的就是 epoll IO 多路复用模型

3. kafka Server Reactor 模型

kafka SocketServer 是基于 Java Nio 开发的,采用了 Reactor 的模式(已被大量实践证明非常高效,在 Netty 和 Mina 中广泛使用).Kafka Reactor 的模型包含三种角色:

- Acceptor;
- Processor;
- Handler;

Kafka Reacator 包含了 1 个 Acceptor 负责接受客户端请求,N 个 Processor 线程负责读写数据(为每个 Connection 创建出一个 Processor 去单独处理,每个 Processor 中均引用独立的 Selector),M 个 Handler 来处理业务逻辑.在 Acceptor 和 Processor,Processor 和 Handler 之间都有队列来缓冲请求.



- Acceptor 的主要职责是监听客户端的连接请求,并建立和客户端的数据传输通道,然后为这个客户端指定一个 Processor,它的工作就到此结束,这样它就可以去响应下一个客户端的连接请求了;
- Processor 的主要职责是负责从客户端读取数据和将响应返回给客户端,它本身

不处理具体的业务逻辑,每个 Processor 都有一个 Selector,用来监听多个客户端,因此可以非阻塞地处理多个客户端的读写请求,Processor 将数据放入 RequestChannel 的 RequestQueue 中和从 ResponseQueue 读取响应;

- Handler(kafka.server.KafkaRequestHandler,kafka.server.KafkaApis)的职责是从 RequestChannel 中的 RequestQueue 取出 Request,处理以后再将 Response 添加到 RequestChannel 中的 ResponseQueue 中;

十、 Kafka Partition Leader 选主机制

1. 大数据常用的选主机制

   Leader 选举算法非常多,大数据领域常用的有以下两种:

   - Zab(zookeeper 使用);
   - Raft;
   - .....

   它们都是 Paxos 算法的变种.

   Zab 协议有四个阶段:

   - Leader election;
   - Discovery(或者 epoch establish);
   - Synchronization(或者 sysnc with followers)
   - Broadcast

   比如 3 个接点选举 leader,编号为 1,2,3.1 先启动,选择自己为 leader,然后 2 启动首先也选择自己为 leader,由于 1,2 都没过半,选择编号大的为 leader,所以 1,2 都选择 2 为 leader,然后 3 启动发现 1,2 已经协商好且数量过半,于是 3 也选择 2 为 leader,leader 选举结束.

   在 Raft 中,任何时候一个服务器可以扮演下面角色之一

   leader:处理所有客户端交互,日志复制等,一般只有一个 Leader;

   follower:类似选民,完全被动

   candidate 候选人:可以被选为一个新的领导人

   启动时在集群中指定一些机器为 Candidate,然后 Candidate 开始向其他机器(尤其是 follower)拉票,当某一个 Candidate 的票数超过半数,它就成为 leader.

2. 常用选主机制的缺点

   由于 kafka 集群依赖 zookeeper 集群,所以最简单最直观的方案是,所有 follower 都在 zookeeper 上设置一个 watch,一旦 leader 宕机,其对应的 ephemeral znode 会自动删除,此时所有 follower 都尝试创建该节点,而创建成功者(zookeeper 保证只有一个能创建成功)即是新的 leader,其他 Replica 即为 follower.

   前面的方案有以下缺点:

   - split-brain(脑裂):这是由 zookeeper 的特性引起的,虽然 zookeeper 能保证所有 watch 按顺序触发,但并不能保证同一时刻所有 Replica"看"到的状态是一样的,这就可能造成不同 Replica 的响应不一致;
   - herd effect(羊群效应):如果宕机那个 Broker 上的 Partition 比较多,会造成多个 watch 被触发,造成集群内大量的调整;
   - zookeeper 负载过重:每个 Replica 都要为此在 zookeeper 上注册一个 watch,当集群规模增加到几千个 Partition 时 zookeeper 负载会过重

3. Kafka Partition 选主机制

kafka 的 leader election 方案解决了上述问题,它在所有 broker 中选出一个 controller,所有 Partition 的 leader 选举都由 controller 决定.controller 会将 leader 的改变直接通过 RPC 的方式(比 zookeeper queue 的方式更高效)通知需为此作为响应的 broker.
kafka 集群 controller 的选举过程如下:
每个 broker 都会在 controller path 上注册一个 watch.当前 controller 失败时,对应的 controller path 会自动消失(因为它是 ephemeral node-临时节点),此时该 watch 被 fire,所有"活"着的 broker 都会去竞选成为新的 controller(创建新的 controller path),但是只会有一个竞选成功(这点由 zookeeper 保证).竞选成功者即为新的 leader,竞选失败者则中心在新的 controller path 上注册 watch.因为 zookeeper 的 watch 是一次性的,被 fire 一次之后即失效,所以需要重新注册.
kafka partition leader 的选举过程如下(由 controller 执行):

- 从 zookeeper 中读取当前分区的所有 ISR(in-sync replicas)集合
- 调用配置的分区选择算法选择分区的 leader

所以,对于下图 partition 0 先选择 broker2,之后选择 broker 0 作为 leader;对于 partition 1 先选择 broker 0,之后选择 broker 1 作为 leader;partition 2 先选择 broker 1,之后选择 broker 2 作为 leader.

```
[test@ kafka_2.9.2-0.8.1.1]$ bin/kafka-topics.sh --describe --zookeeper 10.206.212.14:12181 --topic jiketest
Topic:jiketest  PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: jiketest Partition: 0    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: jiketest Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: jiketest Partition: 2    Leader: 1       Replicas: 1,2   Isr: 1,2
```