**深圳天源迪科信息技术股份有限公司**

**Kafka环境搭建及配置**

**使用手册**

本文件属深圳天源迪科信息技术股份有限公司所有，

未经书面许可，不得以任何形式复印或传播。

**文件建立/修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 版本 | 建立或修改 | 建立/修改人  日期 | 审核人  日期 | 批准人  日期 |
| 1 | 0.1 | 建立 | 吴昌俊  2015年1月9日 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**目 录**

[1 简介 4](#_Toc440980428)

[1.1 目的 4](#_Toc440980429)

[1.2 适用范围 4](#_Toc440980430)

[1.3 术语表 4](#_Toc440980431)

[1.4 参考资料 4](#_Toc440980432)

[2 Kafka背景及应用场景 5](#_Toc440980433)

[2.1 Kafka介绍 5](#_Toc440980434)

[2.2 设计原理 5](#_Toc440980435)

[2.3 应用场景 10](#_Toc440980436)

[2.4 运行架构 11](#_Toc440980437)

[2.5 术语 13](#_Toc440980438)

[3 kafka集群配置 14](#_Toc440980439)

[3.1 安装软件列表 14](#_Toc440980440)

[3.2 JDK安装 14](#_Toc440980441)

[3.2.1 安装环境准备 14](#_Toc440980442)

[3.2.2 安装及配置 14](#_Toc440980443)

[3.3 Scala环境 14](#_Toc440980444)

[3.3.1 Scala下载 14](#_Toc440980445)

[3.3.2 Scala安装 15](#_Toc440980446)

[3.3.3 系统环境配置 16](#_Toc440980447)

[3.3.4 环境验证 16](#_Toc440980448)

[3.4 Zookeeper配置 16](#_Toc440980449)

[3.4.1 安装前提 16](#_Toc440980450)

[3.4.2 下载与安装 17](#_Toc440980451)

[3.4.3 参数配置 17](#_Toc440980452)

[3.4.4 验证服务 19](#_Toc440980453)

[3.5 Kafka安装配置 20](#_Toc440980454)

[3.5.1 下载与安装 20](#_Toc440980455)

[3.5.2 参数配置 21](#_Toc440980456)

[3.5.3 启动与验证 22](#_Toc440980457)

[3.5.4 验证与测试 22](#_Toc440980458)

[3.5.5 kafka 命令 23](#_Toc440980459)

[3.6 Kafka集群监控 24](#_Toc440980460)

[3.6.1 KafkaOffsetMonitor-assembly-0.2.0.jar下载 24](#_Toc440980461)

[3.6.2 安装步骤 24](#_Toc440980462)

[3.6.3 topic的所有partiton消费情况列表 25](#_Toc440980463)

[3.6.4 参考 26](#_Toc440980464)

[4 编程示例 26](#_Toc440980465)

[4.1 工程创建 26](#_Toc440980466)

[5 其他 28](#_Toc440980467)

[5.1 注意事项 28](#_Toc440980468)

[5.2 报错解决 29](#_Toc440980469)

[5.3 建议参数 29](#_Toc440980470)

[5.4 示例代码 30](#_Toc440980471)

[5.5 本次部署 30](#_Toc440980472)

[5.6 参考资料 31](#_Toc440980473)

# 简介

## 目的

为使用Storm作为流式实时计算集群应用场景的快速实施，提供使用规范及操作指导。

## 适用范围

运行环境：Linux平台上运行

使用对象：开发人员、实施人员

## 术语表

无

## 参考资料

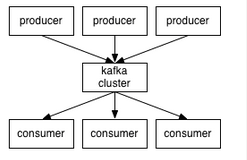
《Oreilly.Getting.Started.with.Storm.中文版.pdf》

<http://storm.apache.org/documentation/Configuration.html>

# Kafka背景及应用场景

## Kafka介绍

  Kafka is a distributed,partitioned,replicated commit logservice。它提供了类似于JMS的特性，但是在[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)实现上完全不同，此外它并不是JMS规范的实现。kafka对消息保存时根据Topic进行归类，发送消息者成为Producer,消息接受者成为Consumer,此外kafka集群有多个kafka实例组成，每个实例([server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128))成为broker。无论是kafka集群，还是producer和consumer都依赖于zookeeper来保证系统可用性集群保存一些meta信息。



## 设计原理

  kafka的[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)初衷是希望作为一个统一的信息收集平台,能够实时的收集反馈信息,并需要能够支撑较大的数据量,且具备良好的容错能力.

1、持久性

    kafka使用文件存储消息,这就直接决定kafka在性能上严重依赖文件系统的本身特性.且无论任何OS下,对文件系统本身的优化几乎没有可能.文件缓 存/直接内存映射等是常用的手段.因为kafka是对日志文件进行append操作,因此磁盘检索的开支是较小的;同时为了减少磁盘写入的次 数,broker会将消息暂时buffer起来,当消息的个数(或尺寸)达到一定阀值时,再flush到磁盘,这样减少了磁盘IO调用的次数.

2、性能

    需要考虑的影响性能点很多,除磁盘IO之外,我们还需要考虑网络IO,这直接关系到kafka的吞吐量问题.kafka并没有提供太多高超的技巧;对于 producer端,可以将消息buffer起来,当消息的条数达到一定阀值时,批量发送给broker;对于consumer端也是一样,批量 fetch多条消息.不过消息量的大小可以通过配置文件来指定.对于kafka broker端,似乎有个sendfile系统调用可以潜在的提升网络IO的性能:将文件的数据映射到系统内存中,socket直接读取相应的内存区域即 可,而无需进程再次copy和交换. 其实对于producer/consumer/broker三者而言,CPU的开支应该都不大,因此启用消息压缩机制是一个良好的策略;压缩需要消耗少量 的CPU资源,不过对于kafka而言,网络IO更应该需要考虑.可以将任何在网络上传输的消息都经过压缩.kafka支持gzip/snappy等多种 压缩方式.

    3、生产者

    负载均衡: producer将会和Topic下所有partition leader保持socket连接;消息由producer直接通过socket发送到broker,中间不会经过任何"路由层".事实上,消息被路由到 哪个partition上,有producer[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)决定.比如可以采用"random""key-hash""轮询"等,如果一个topic中有多个partitions,那么在producer端实现"消息均衡分发"是必要的.

    其中partition leader的位置(host:port)注册在zookeeper中,producer作为zookeeper client,已经注册了watch用来监听partition leader的变更事件.

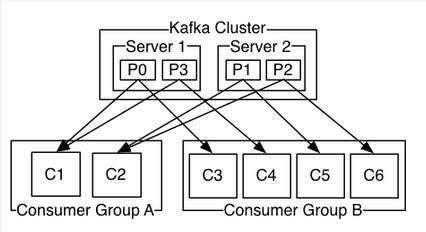
   异步发送：将多条消息暂且在客户端buffer起来，并将他们批量的发送到broker，小数据IO太多，会拖慢整体的网络延迟，批量延迟发送事实上提升了网络效率。不过这也有一定的隐患，比如说当producer失效时，那些尚未发送的消息将会丢失。

 4、消费者

consumer端向broker发送"fetch"请求,并告知其获取消息的offset;此后consumer将会获得一定条数的消息;consumer端也可以重置offset来重新消费消息.

在JMS实现中,Topic模型基于push方式,即broker将消息推送给consumer端.不过在 kafka中,采用了pull方式,即consumer在和broker建立连接之后,主动去pull(或者说fetch)消息;这中模式有些优点,首先 consumer端可以根据自己的消费能力适时的去fetch消息并处理,且可以控制消息消费的进度(offset);此外,消费者可以良好的控制消息消 费的数量,batch fetch.

其他JMS实现,消息消费的位置是有prodiver保留,以便避免重复发送消息或者将没有消费成功的消息重发等,同时还要控制消息的状态.这就要求 JMS broker需要太多额外的工作.在kafka中,partition中的消息只有一个consumer在消费,且不存在消息状态的控制,也没有复杂的消 息确认机制,可见kafka broker端是相当轻量级的.当消息被consumer接收之后,consumer可以在本地保存最后消息的offset,并间歇性的向 zookeeper注册offset.由此可见,consumer[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)也很轻量级.



5、消息传送机制

    对于JMS实现,消息传输担保非常直接:有且只有一次(exactly once).在kafka中稍有不同:

    1) at most once: 最多一次,这个和JMS中"非持久化"消息类似.发送一次,无论成败,将不会重发.

    2) at least once: 消息至少发送一次,如果消息未能接受成功,可能会重发,直到接收成功.

    3) exactly once: 消息只会发送一次.

    at most once: 消费者fetch消息,然后保存offset,然后处理消息;当client保存offset之后,但是在消息处理过程中出现了异常,导致部分消息未能继 续处理.那么此后"未处理"的消息将不能被fetch到,这就是"at most once".

    at least once: 消费者fetch消息,然后处理消息,然后保存offset.如果消息处理成功之后,但是在保存offset阶段zookeeper异常导致保存操作未能 执行成功,这就导致接下来再次fetch时可能获得上次已经处理过的消息,这就是"at least once"，原因offset没有及时的提交给zookeeper，zookeeper恢复正常还是之前offset状态.

    exactly once: kafka中并没有严格的去实现(基于2阶段提交,事务),我们认为这种策略在kafka中是没有必要的.

    通常情况下"at-least-once"是我们搜选.(相比at most once而言,重复接收数据总比丢失数据要好).

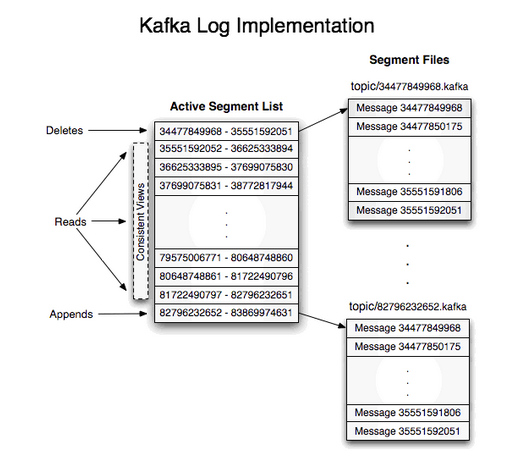
    6、复制备份

    kafka将每个partition数据复制到多个server上,任何一个partition有一个leader和多个follower(可以没有); 备份的个数可以通过broker配置文件来设定.leader处理所有的read-write请求,follower需要和leader保持同 步.Follower和consumer一样,消费消息并保存在本地日志中;leader负责跟踪所有的follower状态,如果follower"落 后"太多或者失效,leader将会把它从replicas同步列表中删除.当所有的follower都将一条消息保存成功,此消息才被认为 是"committed",那么此时consumer才能消费它.即使只有一个replicas实例存活,仍然可以保证消息的正常发送和接收,只要 zookeeper集群存活即可.(不同于其他分布式存储,比如hbase需要"多数派"存活才行)

    当leader失效时,需在followers中选取出新的leader,可能此时follower落后于leader,因此需要选择一个"up-to-date"的follower.选择follower时需要兼顾一个问题,就是新leader[server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128)上所已经承载的partition leader的个数,如果一个server上有过多的partition leader,意味着此server将承受着更多的IO压力.在选举新leader,需要考虑到"负载均衡".

    7.日志

    如果一个topic的名称为"my\_topic",它有2个partitions,那么日志将会保存在my\_topic\_0和my\_topic\_1两个目 录中;日志文件中保存了一序列"log entries"(日志条目),每个log entry格式为"4个字节的数字N表示消息的长度" + "N个字节的消息内容";每个日志都有一个offset来唯一的标记一条消息,offset的值为8个字节的数字,表示此消息在此partition中所 处的起始位置..每个partition在物理存储层面,有多个log file组成(称为segment).segmentfile的命名为"最小offset".kafka.例如"00000000000.kafka"; 其中"最小offset"表示此segment中起始消息的offset.



其中每个partiton中所持有的segments列表信息会存储在zookeeper中.

    当segment文件尺寸达到一定阀值时(可以通过配置文件设定,默认1G),将会创建一个新的文件;当buffer中消息的条数达到阀值时将会触发日志 信息flush到日志文件中,同时如果"距离最近一次flush的时间差"达到阀值时,也会触发flush到日志文件.如果broker失效,极有可能会 丢失那些尚未flush到文件的消息.因为[server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128)意外实现,仍然会导致log文件格式的破坏(文件尾部),那么就要求当server启东是需要检测最后一个segment的文件结构是否合法并进行必要的修复.

    获取消息时,需要指定offset和最大chunk尺寸,offset用来表示消息的起始位置,chunk size用来表示最大获取消息的总长度(间接的表示消息的条数).根据offset,可以找到此消息所在segment文件,然后根据segment的最 小offset取差值,得到它在file中的相对位置,直接读取输出即可.

    日志文件的删除策略非常简单:启动一个后台线程定期扫描log file列表,把保存时间超过阀值的文件直接删除(根据文件的创建时间).为了避免删除文件时仍然有read操作(consumer消费),采取copy-on-write方式.

    8、分配

    kafka使用zookeeper来存储一些meta信息,并使用了zookeeper watch机制来发现meta信息的变更并作出相应的动作(比如consumer失效,触发负载均衡等)

    1) Broker node registry: 当一个kafkabroker启动后,首先会向zookeeper注册自己的节点信息(临时znode),同时当broker和zookeeper断开连接时,此znode也会被删除.

    格式: /broker/ids/[0...N]   -->host:port;其中[0..N]表示broker id,每个broker的配置文件中都需要指定一个数字类型的id(全局不可重复),znode的值为此broker的host:port信息.

    2) Broker Topic Registry: 当一个broker启动时,会向zookeeper注册自己持有的topic和partitions信息,仍然是一个临时znode.

    格式: /broker/topics/[topic]/[0...N]  其中[0..N]表示partition索引号.

    3) Consumer and Consumer group: 每个consumer[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)被创建时,会向zookeeper注册自己的信息;此作用主要是为了"负载均衡".

    一个group中的多个consumer可以交错的消费一个topic的所有partitions;简而言之,保证此topic的所有 partitions都能被此group所消费,且消费时为了性能考虑,让partition相对均衡的分散到每个consumer上.

    4) Consumer id Registry: 每个consumer都有一个唯一的ID(host:uuid,可以通过配置文件指定,也可以由系统生成),此id用来标记消费者信息.

    格式:/consumers/[group\_id]/ids/[consumer\_id]

    仍然是一个临时的znode,此节点的值为{"topic\_name":#streams...},即表示此consumer目前所消费的topic + partitions列表.

5) Consumer offset Tracking: 用来跟踪每个consumer目前所消费的partition中最大的offset.

  格式:/consumers/[group\_id]/offsets/[topic]/[broker\_id-partition\_id]-->offset\_value

    此znode为持久节点,可以看出offset跟group\_id有关,以表明当group中一个消费者失效,其他consumer可以继续消费.

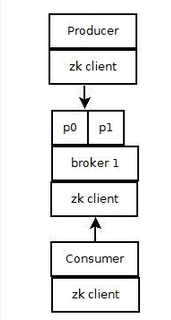
    6) Partition Owner registry: 用来标记partition被哪个consumer消费.临时znode

    格式:/consumers/[group\_id]/owners/[topic]/[broker\_id-partition\_id]-->consumer\_node\_id当consumer启动时,所触发的操作:

    A) 首先进行"Consumer id Registry";

    B) 然后在"Consumer id Registry"节点下注册一个watch用来监听当前group中其他consumer的"leave"和"join";只要此znode path下节点列表变更,都会触发此group下consumer的负载均衡.(比如一个consumer失效,那么其他consumer接管 partitions).

    C) 在"Broker id registry"节点下,注册一个watch用来监听broker的存活情况;如果broker列表变更,将会触发所有的groups下的consumer重新balance.



1) Producer端使用zookeeper用来"发现"broker列表,以及和Topic下每个partition leader建立socket连接并发送消息.

2) Broker端使用zookeeper用来注册broker信息,已经监测partitionleader存活性.

3) Consumer端使用zookeeper用来注册consumer信息,其中包括consumer消费的partition列表等,同时也用来发现broker列表,并和partition leader建立socket连接,并获取消息.

## 应用场景

* **消息队列**

比起大多数的消息系统来说，Kafka有更好的吞吐量，内置的分区，冗余及容错性，这让Kafka成为了一个很好的大规模消息处理应用的解决方案。消息系统 一般吞吐量相对较低，但是需要更小的端到端延时，并尝尝依赖于Kafka提供的强大的持久性保障。在这个领域，Kafka足以媲美传统消息系统，如[ActiveMR](http://activemq.apache.org/)或[RabbitMQ](https://www.rabbitmq.com/)。

* 行为跟踪

Kafka的另一个应用场景是跟踪用户浏览页面、搜索及其他行为，以发布-订阅的模式实时记录到对应的topic里。那么这些结果被订阅者拿到后，就可以做进一步的实时处理，或实时监控，或放到hadoop/离线数据仓库里处理。

* 元信息监控

作为操作记录的监控模块来使用，即汇集记录一些操作信息，可以理解为运维性质的数据监控吧。

* 日志收集

日志收集方面，其实开源产品有很多，包括Scribe、Apache Flume。很多人使用Kafka代替日志聚合（log aggregation）。日志聚合一般来说是从服务器上收集日志文件，然后放到一个集中的位置（文件服务器或HDFS）进行处理。然而Kafka忽略掉 文件的细节，将其更清晰地抽象成一个个日志或事件的消息流。这就让Kafka处理过程延迟更低，更容易支持多数据源和分布式数据处理。比起以日志为中心的 系统比如Scribe或者Flume来说，Kafka提供同样高效的性能和因为复制导致的更高的耐用性保证，以及更低的端到端延迟。

* 流处理

这个场景可能比较多，也很好理解。保存收集流数据，以提供之后对接的Storm或其他流式计算框架进行处理。很多用户会将那些从原始topic来的数据进行 阶段性处理，汇总，扩充或者以其他的方式转换到新的topic下再继续后面的处理。例如一个文章推荐的处理流程，可能是先从RSS数据源中抓取文章的内 容，然后将其丢入一个叫做“文章”的topic中；后续操作可能是需要对这个内容进行清理，比如回复正常数据或者删除重复数据，最后再将内容匹配的结果返 还给用户。这就在一个独立的topic之外，产生了一系列的实时数据处理的流程。[Strom](http://storm.incubator.apache.org/" \t "_blank)和[Samza](http://samza.incubator.apache.org/)是非常著名的实现这种类型数据转换的框架。

* 事件源

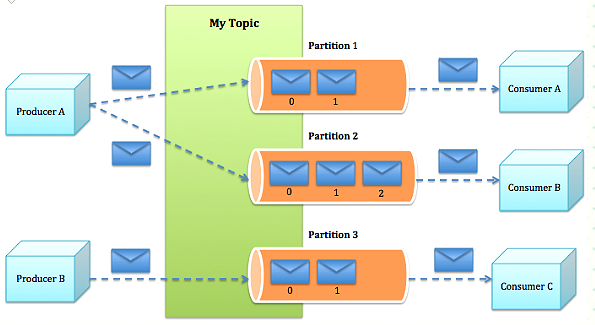
事件源是一种应用程序设计的方式，该方式的状态转移被记录为按时间顺序排序的记录序列。Kafka可以存储大量的日志数据，这使得它成为一个对这种方式的应用来说绝佳的后台。比如动态汇总（News feed）。

* 持久性日志（commit log）

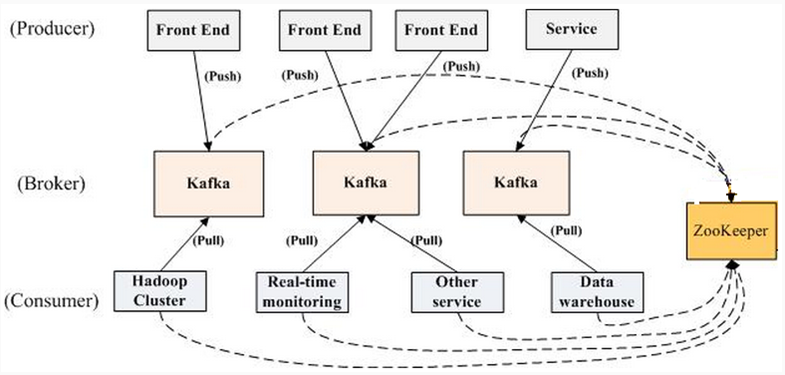
Kafka可以为一种外部的持久性日志的分布式系统提供服务。这种日志可以在节点间备份数据，并为故障节点数据回复提供一种重新同步的机制。Kafka中日志压缩功能为这种用法提供了条件。在这种用法中，Kafka类似于Apache BookKeeper项目。

## 运行架构

Kafka 的整体架构非常简单，是显式分布式架构，producer、broker（kafka）和consumer都可以有多个。 Producer，consumer实现Kafka注册的接口，数据从producer发送到broker，broker承担一个中间缓存和分发的作用。 broker分发注册到系统中的consumer。broker的作用类似于缓存，即活跃的数据和离线处理系统之间的缓存。客户端和服务器端的通信，是基 于简单，高性能，且与编程语言无关的TCP协议。



1. Producer根据指定的partition方法（round-robin、hash等），将消息发布到指定topic的partition里面
2. kafka集群接收到Producer发过来的消息后，将其持久化到硬盘，并保留消息指定时长（可配置），而不关注消息是否被消费。
3. Consumer从kafka集群pull数据，并控制获取消息的offset



如上图所示，一个典型的kafka集群中包含若干producer（可以是web前端产生的page view，或者是服务器日志，系统CPU、memory等），若干broker（Kafka支持水平扩展，一般broker数量越多，集群吞吐率越高），若干consumer group，以及一个[Zookeeper](http://zookeeper.apache.org/)集群。Kafka通过Zookeeper管理集群配置，选举leader，以及在consumer group发生变化时进行rebalance。producer使用push模式将消息发布到broker，consumer使用pull模式从broker订阅并消费消息。

## 术语

* Broker

Kafka集群包含一个或多个服务器，这种服务器被称为broker

* Topic

每条发布到Kafka集群的消息都有一个类别，这个类别被称为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

parition是物理上的概念，每个topic包含一个或多个partition，创建topic时可指定parition数量。每个partition对应于一个文件夹，该文件夹下存储该partition的数据和索引文件

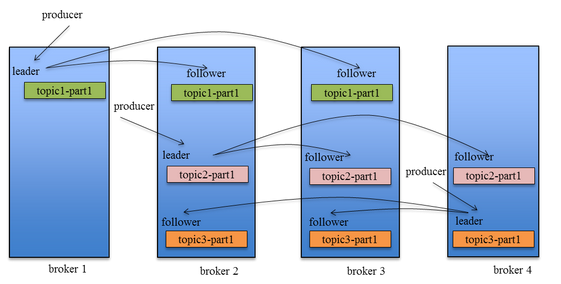
* Producer

负责发布消息到Kafka broker

* Consumer

消费消息。每个consumer属于一个特定的consumer group（可为每个consumer指定group name，若不指定group name则属于默认的group）。使用consumer high level API时，同一topic的一条消息只能被同一个consumer group内的一个consumer消费，但多个consumer group可同时消费这一消息。

* Replication



* Leader Election

Leader Election主要指Replica之间的Leader Election，引入Replication之后，同一个Partition可能会有多个Replica，而这时需要在这些Replication之间选出一个Leader，Producer和Consumer只与这个Leader交互，其它Replica作为Follower从Leader中复制数据。

因为需要保证同一个Partition的多个Replica之间的数据一致性（其中一个宕机后其它Replica必须要能继续服务并且即不能造成数据重复也不能造成数据丢失）。如果没有一个Leader，所有Replica都可同时读/写数据，那就需要保证多个Replica之间互相（N×N条通路）同步数据，数据的一致性和有序性非常难保证，大大增加了Replication实现的复杂性，同时也增加了出现异常的几率。而引入Leader后，只有Leader负责数据读写，Follower只向Leader顺序Fetch数据（N条通路），系统更加简单且高效。

# kafka集群配置

## 安装软件列表

|  |  |  |
| --- | --- | --- |
| **序号** | **软件名称** | **说明** |
| 1 | JDK 1.7 |  |
| 2 | Scala-2.11.4 | 环境配置需要依赖Scala环境 |

## JDK安装

### 安装环境准备

服务器系统：linux 64位操作系统

安装版本：jdk-7u67-linux-x64.tar.gz（jdk1.7及以上版本）

### 安装及配置

配置过程

在profile文件末尾加入：   
export JAVA\_HOME=/opt/jdk1.7.0\_67

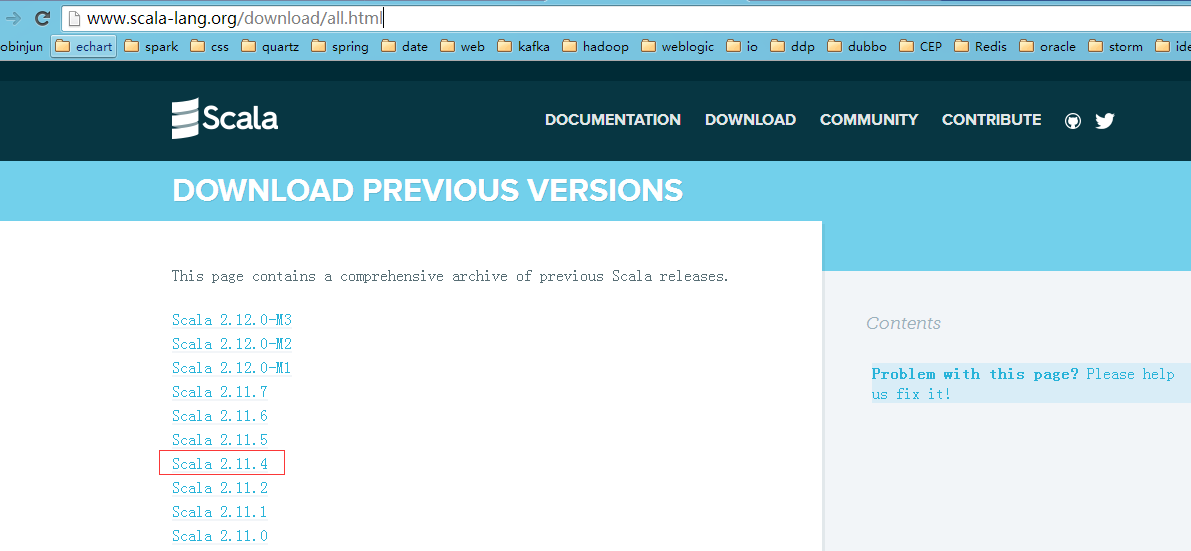
export PATH=$JAVA\_HOME/bin:$PATH

export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

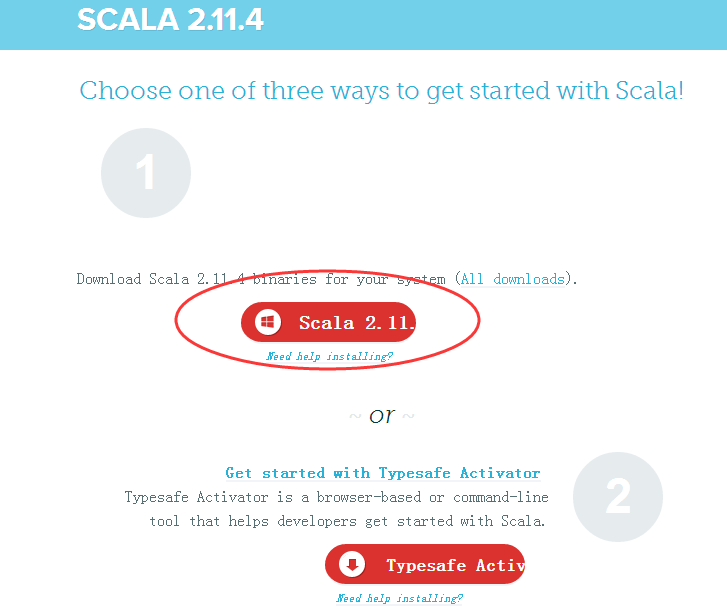
## Scala环境

### Scala下载

下载地址: http://www.scala-lang.org/download/all.html



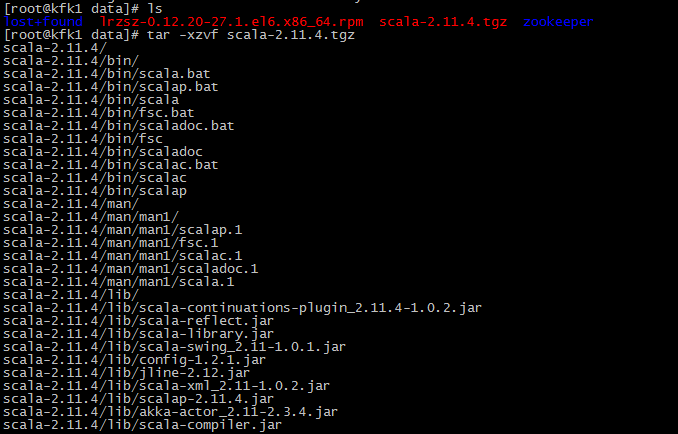
下载如下版本



### Scala安装

1. 解压文件

tar –xzvf scala-2.11.4.tgz



### 系统环境配置

vi /etc/profile

在文件的最后追加内容：

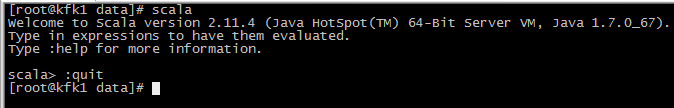
export PATH=/opt/scala-2.11.4/bin:$PATH

export SCALA\_HOME=/opt/scala-2.11.4

保存

source /etc/profile

### 环境验证



## Zookeeper配置

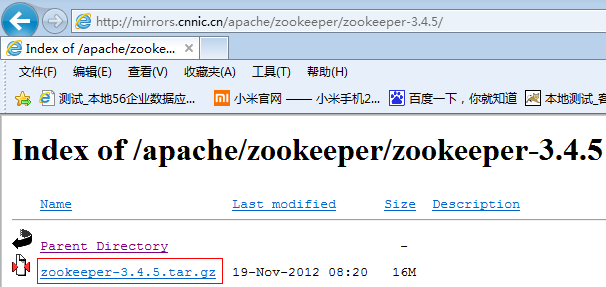
### 安装前提

* JDK环境：已经安装jdk（jdk1.6及以上版本），并已经正确配置jdk的环境变量（JAVA\_HOME,JRE\_HOME,CLASSPATH,PATH）
* 本例中为配置2台机器：192.168.128.76、192.168.128.77

### 下载与安装

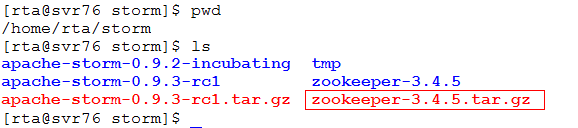
* **软件下载**

示例安装的版本是zookeeper-3.4.5.tar.gz，在Apache的官网下载：<http://zookeeper.apache.org/>



* **软件安装**

执行：tar -zxf zookeeper-3.4.5.tar.gz



### 参数配置

* **创建zoo.cfg文件**

进入到zookeeper的conf目录下将zoo\_sample.cfg文件拷贝一份，命名为“zoo.cfg”,修改zoo.cfg内容如下：

   tickTime=2000

 initLimit=10

 syncLimit=5

 clientPort=2182

 dataDir=/home/rta/storm/tmp/zookeeper/data

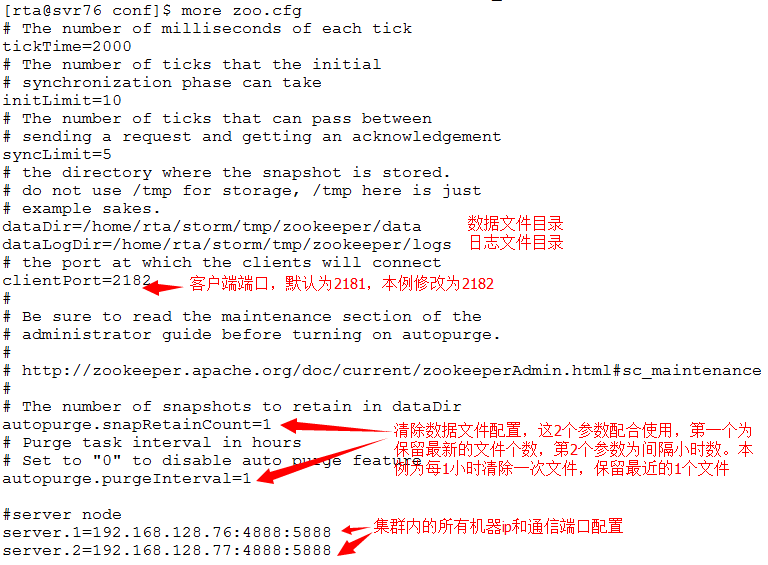
 dataLogDir=/home/rta/storm/tmp/zookeeper/logs

 #server node

 server.1=192.168.128.76:4888:5888

 server.2=192.168.128.77:4888:5888

截图说明如下：



参数说明：

  4888端口是zookeeper服务之间通信的接口（默认端口为2888）；

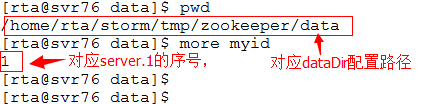
  5888端口是zookeeper与其他应用程序的通信接口（默认端口为3888）；

  192.168.128.76、192.168.128.77 是服务主机ip(本例中是2个节点服务)。

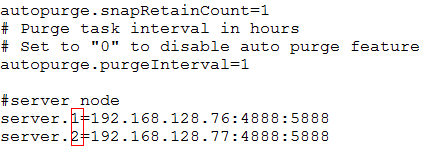
* **创建myid文件**

在/home/rta/storm/tmp/zookeeper/data创建文件myid文件，编辑myid文件（与zoo.cfg中的server编号一致）

1



注：与zoo.cfg中的server编号一致



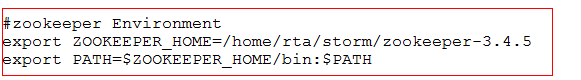
* **配置zookeeper环境变量**

vi /etc/profile,在文件内容的最后增加如下内容：

#zookeeper Environment

export ZOOKEEPER\_HOME=/home/rta/storm/zookeeper-3.4.5

export PATH=$ZOOKEEPER\_HOME/bin:$PATH



### 验证服务

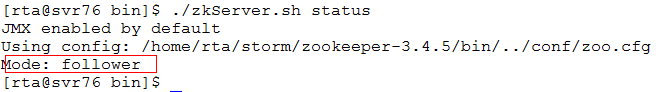
* 启动 192.168.128.76机器zookeeper服务

进入zookeeper的bin目录，执行命令zkServer.sh start，如：

nohup ./zkServer.sh start &

此时查看状态可能会报错，由于集群中其他节点没有启动，所以通信报错（不影响服务使用）

./zkServer.sh status



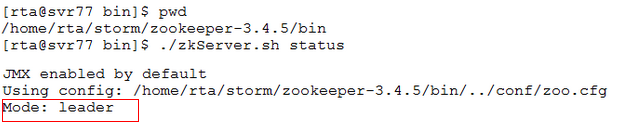
* 启动 192.168.128.77机器zookeeper服务

进入zookeeper的bin目录，执行命令zkServer.sh start，如：

nohup ./zkServer.sh start &

此时查看状态集群节点状态

./zkServer.sh status



可以看到上面的2台机器中一台模式为 leader，一台为 follower（主、从模式）

## Kafka安装配置

### 下载与安装

* **安装文件下载**

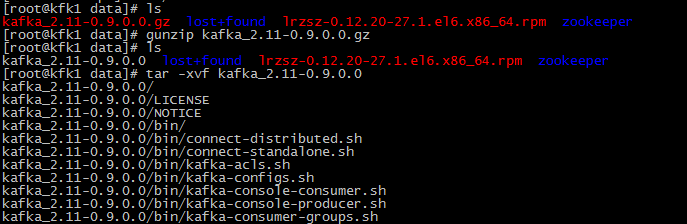
http://kafka.apache.org/downloads.html



* **解压安装**

gunzip kafka\_2.11-0.9.0.0.gz

tar -xvf kafka\_2.11-0.9.0.0



将解压后的文件夹复制到集群另外的2台机器上

scp -r ./ kafka\_2.11-0.9.0.0 root@ 10.31.31.13:/opt

scp -r ./ kafka\_2.11-0.9.0.0 root@ 10.31.31.14:/opt/

### 参数配置

Broker Configs配置项参考：



文件目录/opt/ kafka\_2.11-0.9.0.0/config

参数说明：

broker.id： 唯一，填数字，本文中分别为12/13/14

host.name：唯一，填服务器IP 10.31.31.12 / 10.31.31.13 / 10.31.31.14

log.dirs :日志目录

zookeeper.connect= 10.31.31.12:2181,10.31.31.13:2181,10.31.31.14:2181

本例中参数配置如下（以10.31.31.12 机器为例，另2台机器类同）

broker.id=12

host.name=10.31.31.12

log.dirs=/data/kafka/logs

zookeeper.connect=10.31.31.12:2181,10.31.31.13:2181,10.31.31.14:2181

vi server.properties

### 启动与验证

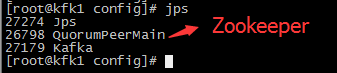
* **启动集群内的zookeeper服务**

集群内的3台机器zookeeper服务，本例为独立安装，分别到对应的目录下执行启动命令

目录：/opt/zookeeper-3.4.7

启动命令：cd /opt/zookeeper-3.4.7/bin/ && ./zkServer.sh start &

查看进程:jps



* **启动kafka进程**

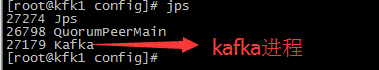
在集群内的3台机器上，分别启动kafka服务，分别进入对应的目录下执行命令

目录：/opt/kafka\_2.11-0.9.0.0

启动命令：

cd /opt/kafka\_2.11-0.9.0.0/bin && ./kafka-server-start.sh ../config/server.properties &

查看进程:jps



### 验证与测试

* 创建一个测试topic，名称为mytest、mytest2

bin/kafka-topics.sh --create --zookeeper kfk1:2181 --replication-factor 1 --partitions 1 --topic mytest

bin/kafka-topics.sh --create --zookeeper 10.31.31.12:2181 --replication-factor 1 --partitions 1 --topic mytest2

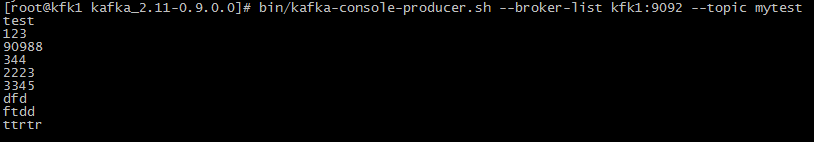
* 列出topic信息

bin/kafka-topics.sh --list --zookeeper kfk1:2181



* 生产者，输入数据

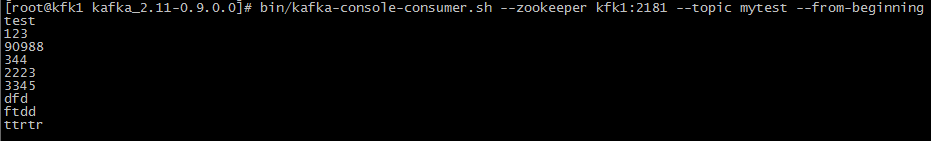
bin/kafka-console-producer.sh --broker-list kfk1:9092 --topic mytest



* 消费者，接收数据

另开一个终端窗口，观察消费者消费情况

bin/kafka-console-consumer.sh --zookeeper kfk1:2181 --topic mytest --from-beginning



### kafka 命令

1. 启动Server

由于kafka集群依赖于zookeeper服务，首先启动zookeeper，然后启动kafka服务，命令如下：

nohup bin/kafka-server-start.sh config/server.properties &

2. 创建Topic

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic mytest

3. 查看命令

bin/kafka-topics.sh --list --zookeeper localhost:2181

4. 发送消息

bin/kafka-console-producer.sh --broker-list 192.168.1.5:9092 --topic mytest

5. 消费消息

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic mytest --from-beginning

6.停止服务

pkill -9 -f config/server.properties

7.删除无用的topic

bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic mytest

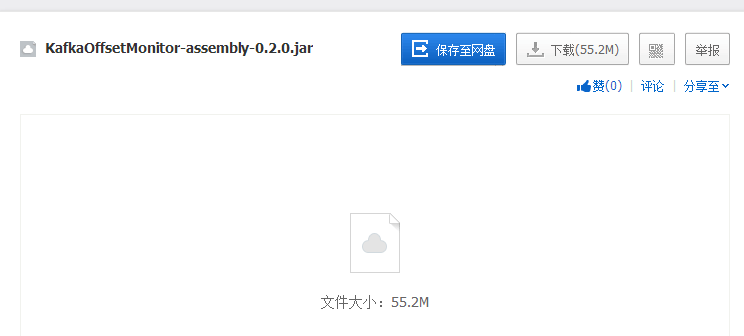
8．帮助命令

bin/kafka-topics.sh –help

## Kafka集群监控

### KafkaOffsetMonitor-assembly-0.2.0.jar下载

<http://pan.baidu.com/s/1sj0YERV>



### 安装步骤

1.新建在kafka的bin下面新建kafka-monitor文件夹，把jar包KafkaOffsetMonitor-assembly-0.2.0.jar丢进去，并且在文件夹里新建脚本文件kafka-monitor，文件内容如下：

#!/bin/bash

BASEDIR=`dirname $0`

echo $BASEDIR

java -cp $BASEDIR/KafkaOffsetMonitor-assembly-0.2.0.jar \

com.quantifind.kafka.offsetapp.OffsetGetterWeb \

--zk kfk1:2181,kfk2:2181,kfk3:2181 \

--port 8089 \

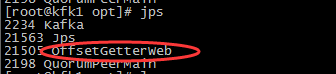
--refresh 10.seconds \

--retain 1.days

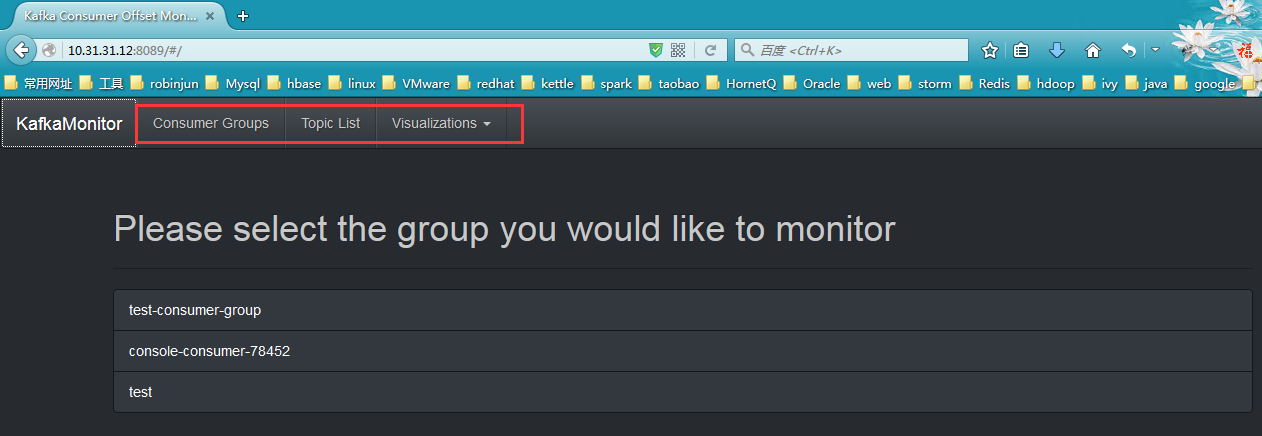
3.启动kafka之后，然后在运行kafka-monitor文件

nohup ./kafka-monitor &

4.jps查看进程

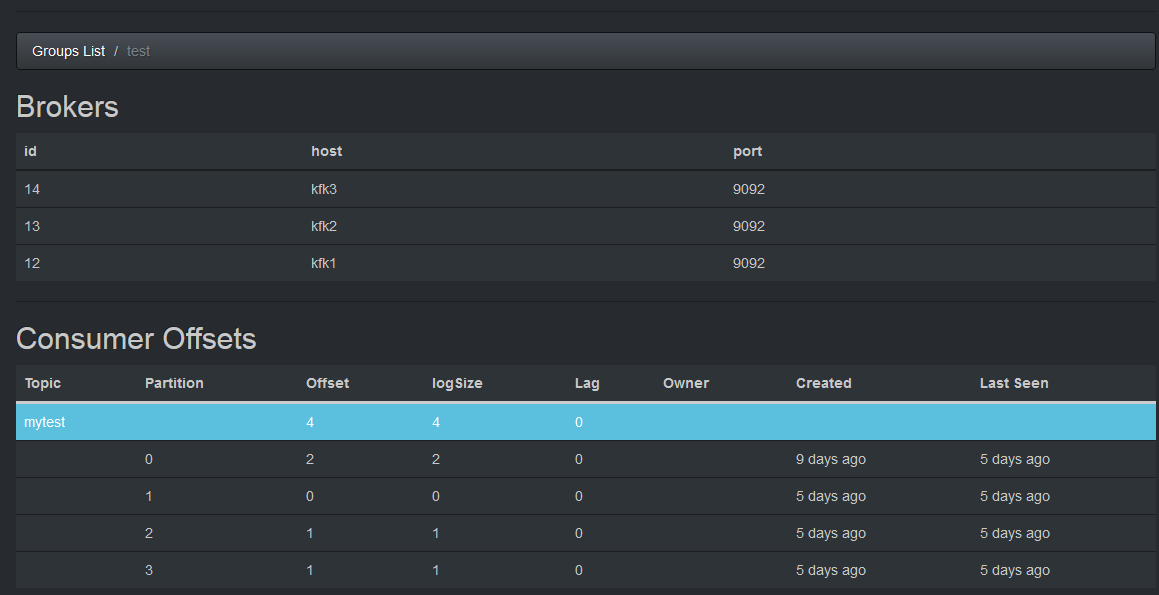


5.在浏览器中即可使用ip:8089访问kafka的监控页面。



6.本次部署机器：10.31.31.12 目录：/opt

### **topic的所有partiton消费情况列表**





### **参考**

<http://ju.outofmemory.cn/entry/136555>

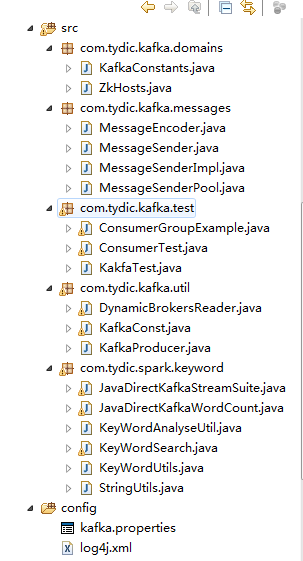
<http://www.cnblogs.com/smartloli/p/4562551.html>

<http://bigbo.github.io/pages/2015/01/17/kafka_web_console/>

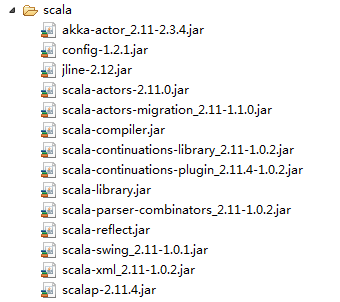
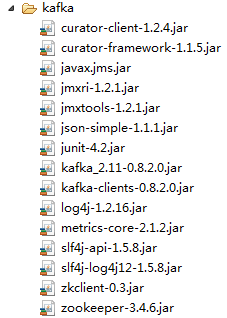
# 编程示例

## 工程创建

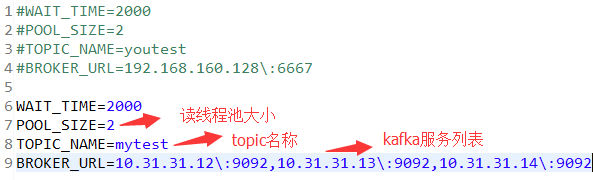
* **工程结构图**



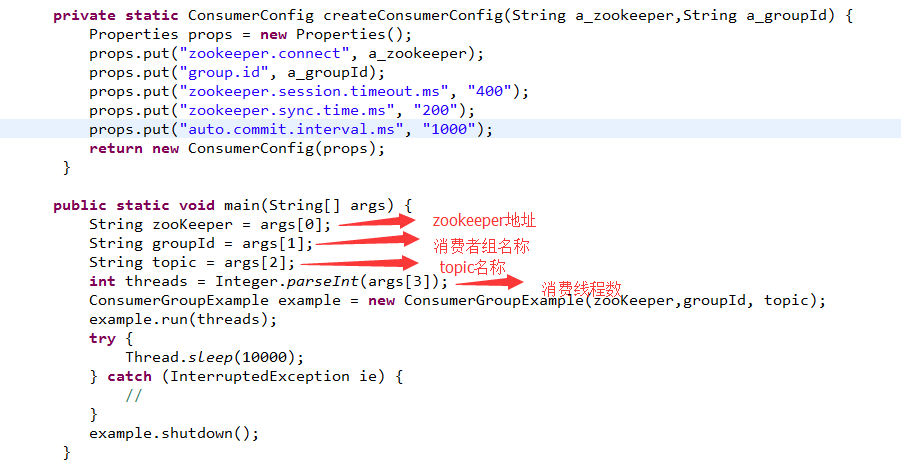
引入jar包如下，注：scala版本需要与集群scala版本一致，否则会抛异常



kafka.properties 配置参数



ConsumerGroupExample.java 代码片段



# 其他

## 注意事项

* **关闭防火墙**

1）重启后永久性生效：

　　开启： chkconfig iptables on

　　关闭： chkconfig iptables off

2）即时生效，重启后失效：

　　开启： service iptables start

　　关闭： service iptables stop

3）指定开放端口

/sbin/iptables -I INPUT -p tcp --dport 2182 -j ACCEPT

/etc/rc.d/init.d/iptables save

/etc/init.d/iptables restart

## 报错解决

* **遇到Producer connection to hostname:9092 unsuccessful**



解决办法：

    寻找linux下的/etc/hosts和window下的C:\Windows\System32\drivers\etc\hosts所有IP地址与主机名是否配置正确。

最后发现一个低级错误，window下的hostname配置IP地址错误了。正确配置如下：

10.31.31.12 kfk1

10.31.31.13 kfk2

10.31.31.14 kfk3

## 建议参数

**delete.topic.enable=true**

否则执行bin/kafka-topics.sh --delete --zookeeper kfk1:2181 --topic mytest会提示如下错误



解决思路：

前提条件： 在启动broker时候开启删除topic的开关，即在server.properties中添加：  delete.topic.enable=true

配置后仍然会提示，但是已经物理删除

**创建topic时partition >= broker**

为了更好的做负载均衡，Kafka尽量将所有的Partition均匀分配到整个集群上。一个典型的部署方式是一个Topic的Partition数量大于Broker的数量

**创建topic时replica : 2N+1**

为了保证Leader Election的正常进行，它所能容忍的fail的follower个数比较少。如果要容忍1个follower挂掉，必须要有3个以上的Replica，如果要容忍2个Follower挂掉，必须要有5个以上的Replica。也就是说，在生产环境下为了保证较高的容错程度，必须要有大量的Replica，而大量的Replica又会在大数据量下导致性能的急剧下降。

## 示例代码



## 本次部署

机器IP地址：10.31.31.12 10.31.31.13 10.31.31.14

zookeeper部署目录：/opt/zookeeper-3.4.7/

手工启动命令：

/opt/zookeeper-3.4.7/bin/zkServer.sh start

后台启动命令：

cd /opt/zookeeper-3.4.7/bin/ && ./zkServer.sh start &

kafka部署目录：/opt/kafka\_2.11-0.9.0.0/

手工启动命令：

/opt/kafka\_2.11-0.9.0.0/bin/kafka-server-start.sh ../config/server.properties

后台启动命令：

cd /opt/kafka\_2.11-0.9.0.0/bin && nohup ./kafka-server-start.sh ../config/server.properties &

注：不适用nohup窗口关闭后进程会消失

cd /opt/kafka\_2.11-0.9.0.0

bin/kafka-topics.sh --list --zookeeper kfk1:2181

bin/kafka-topics.sh --list --zookeeper kfk2:2181

bin/kafka-topics.sh --create --zookeeper kfk1:2181 --replication-factor 1 --partitions 1 --topic mytest

bin/kafka-console-producer.sh --broker-list kfk1:9092 --topic mytest

bin/kafka-console-consumer.sh --zookeeper kfk1:2181 --topic mytest --from-beginning

创建topic:

bin/kafka-topics.sh --create --zookeeper kfk1:2181 --replication-factor 1 --partitions 3 --topic mytest

删除topic：

bin/kafka-topics.sh --delete --zookeeper kfk1:2181 --topic mytest2

列出topic:

bin/kafka-topics.sh --list --zookeeper kfk1:2181

创建topic:

bin/kafka-topics.sh --create --zookeeper kfk1:2181 --replication-factor 1 --partitions 4 --topic mytest

## 参考资料

Kafka设计解析:Kafka High Availability

<http://www.infoq.com/cn/articles/kafka-analysis-part-2>

[Kafka深度解析](http://www.jasongj.com/2015/01/02/Kafka%E6%B7%B1%E5%BA%A6%E8%A7%A3%E6%9E%90/)

<http://www.jasongj.com/2015/01/02/Kafka%E6%B7%B1%E5%BA%A6%E8%A7%A3%E6%9E%90/>

kafka入门：简介、使用场景、设计原理、主要配置及集群搭建

<http://www.cnblogs.com/likehua/p/3999538.html>

分布式消息系统 Kafka 简介：特点、架构、设计、应用场景

<http://www.aboutyun.com/thread-11914-1-1.html>