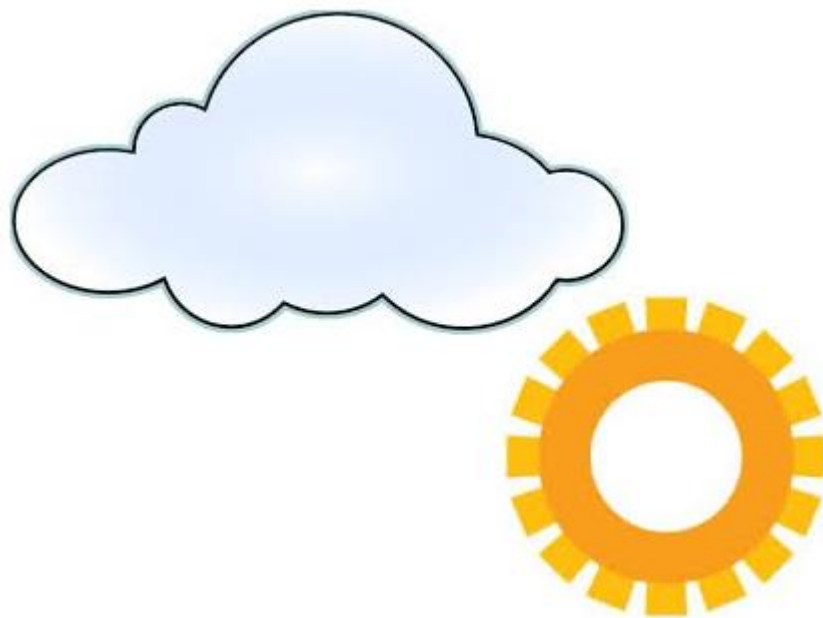


# kafka & mafka client 开发与实践



李志涛

[lizhitao@meituan.com](mailto:lizhitao@meituan.com)

平台业务部移动后台组



kafka介绍



kafka架构 & 稳定性



性能优化



性能测试



监控



mafka client开发

# 什么是kafka

kafka是最初由Linkedin公司开发，使用Scala语言编写,运行在jvm虚拟机上，Kafka是一个分布式、分区的、多副本的、多订阅者，基于zookeeper协调的分布式日志系统(分布式MQ系统)，常见可以用于web/nginx日志，搜索日志，监控日志，访问日志，消息服务等等，Linkedin于2010年贡献给了Apache基金会，成为旗下顶级开源项目。

## 整体设计的几个特点

- (1) 默认使用持久化
- (2) 优先考虑吞吐率
- (3) 信息的消费状态在 consumer 端记录而不是server 端.
- (4) kafka 完全是 分布式的, produces broker consumer 都认为是分布式的.

# 适用场景

- 适合场景
  - push发送
  - 高吞吐量
  - 可以作为大缓冲区使用
  - Hadoop或传统的数据仓库中存储消息用于离线分析
  - nginx日志收集
- 不适合场景：
  - 类似大象，对低延时，实时性要求比较严格，单条消息延时行



kafka介绍



kafka架构 & 稳定性



性能优化



性能测试

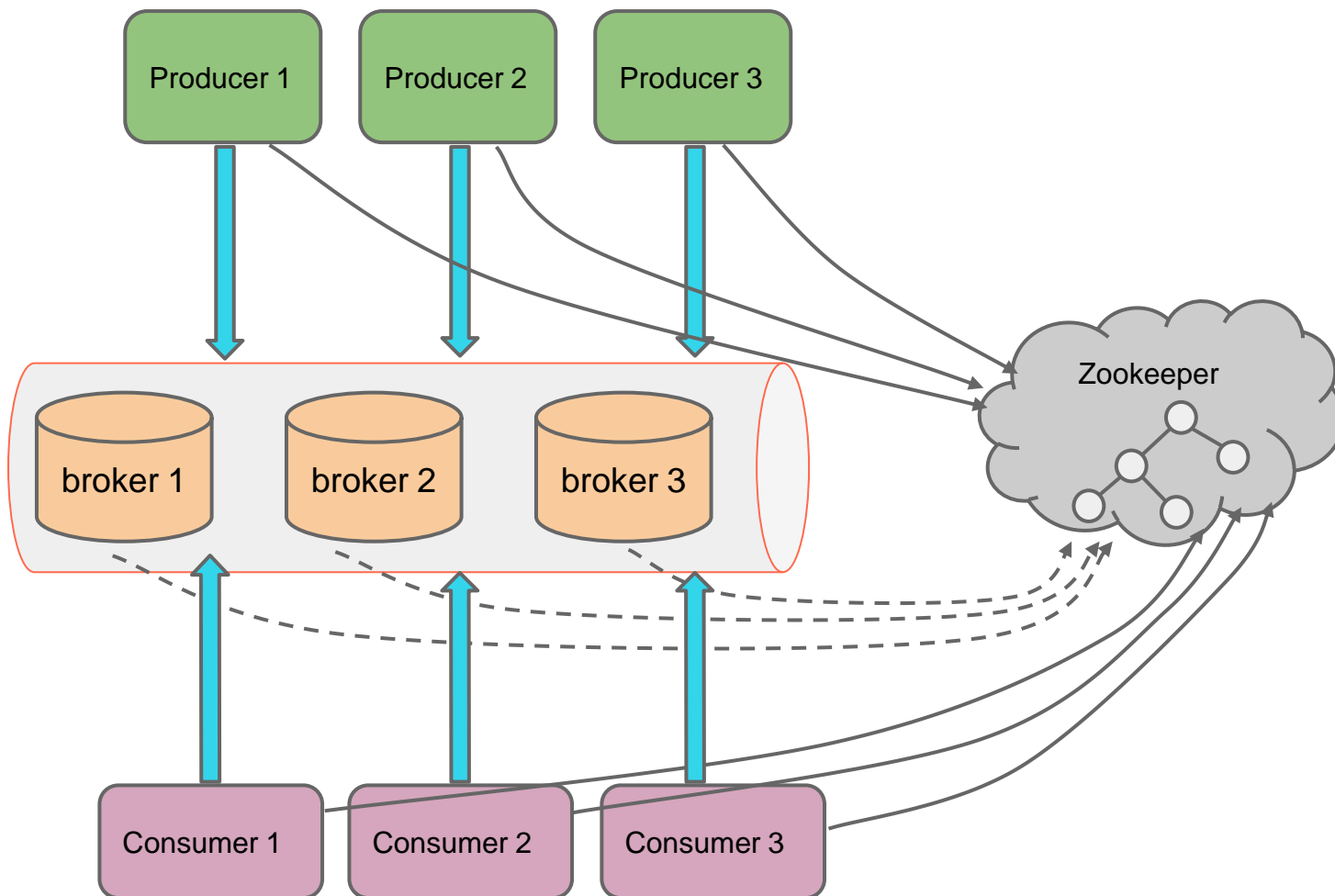


监控

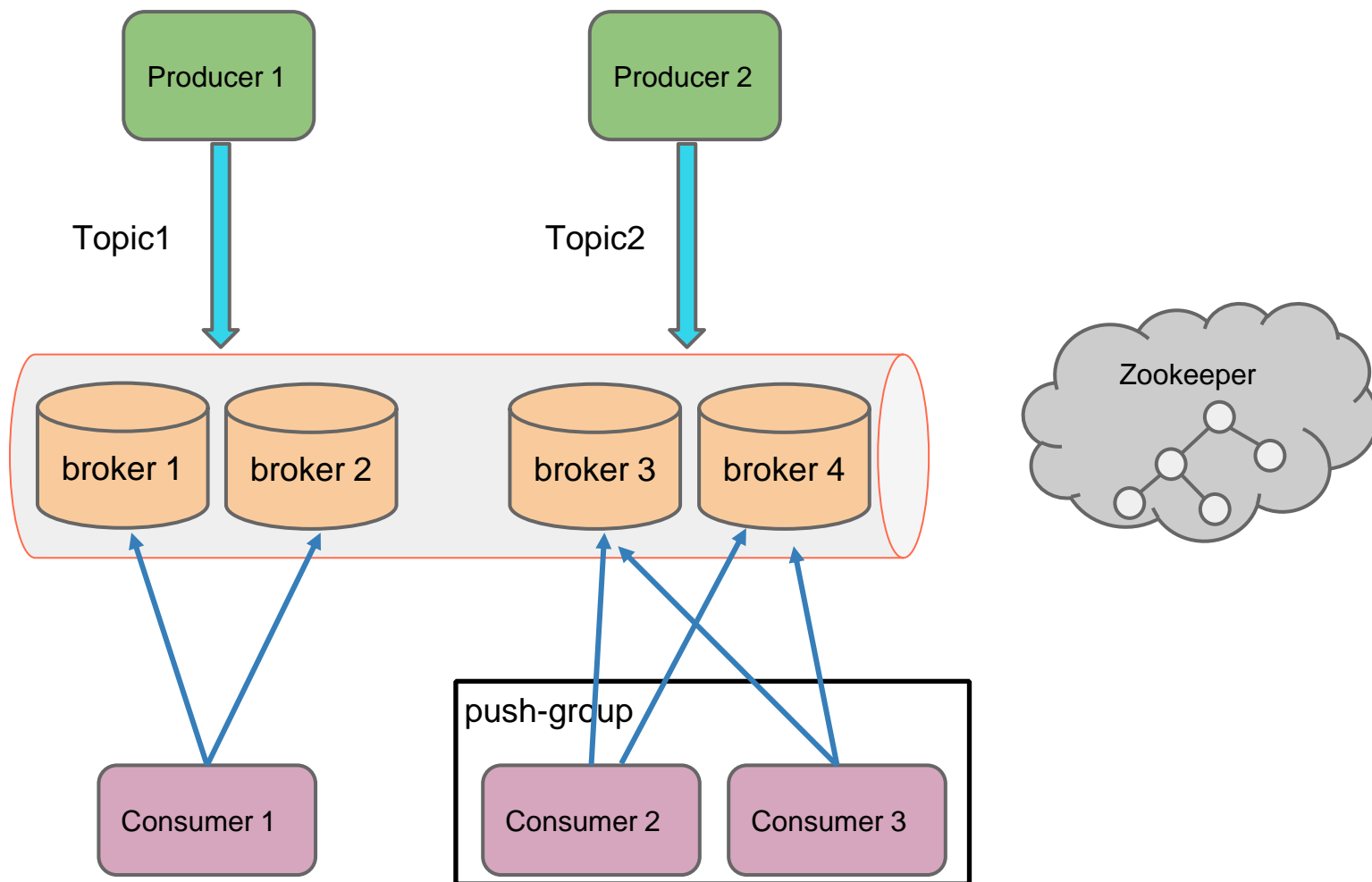


mafka client开发

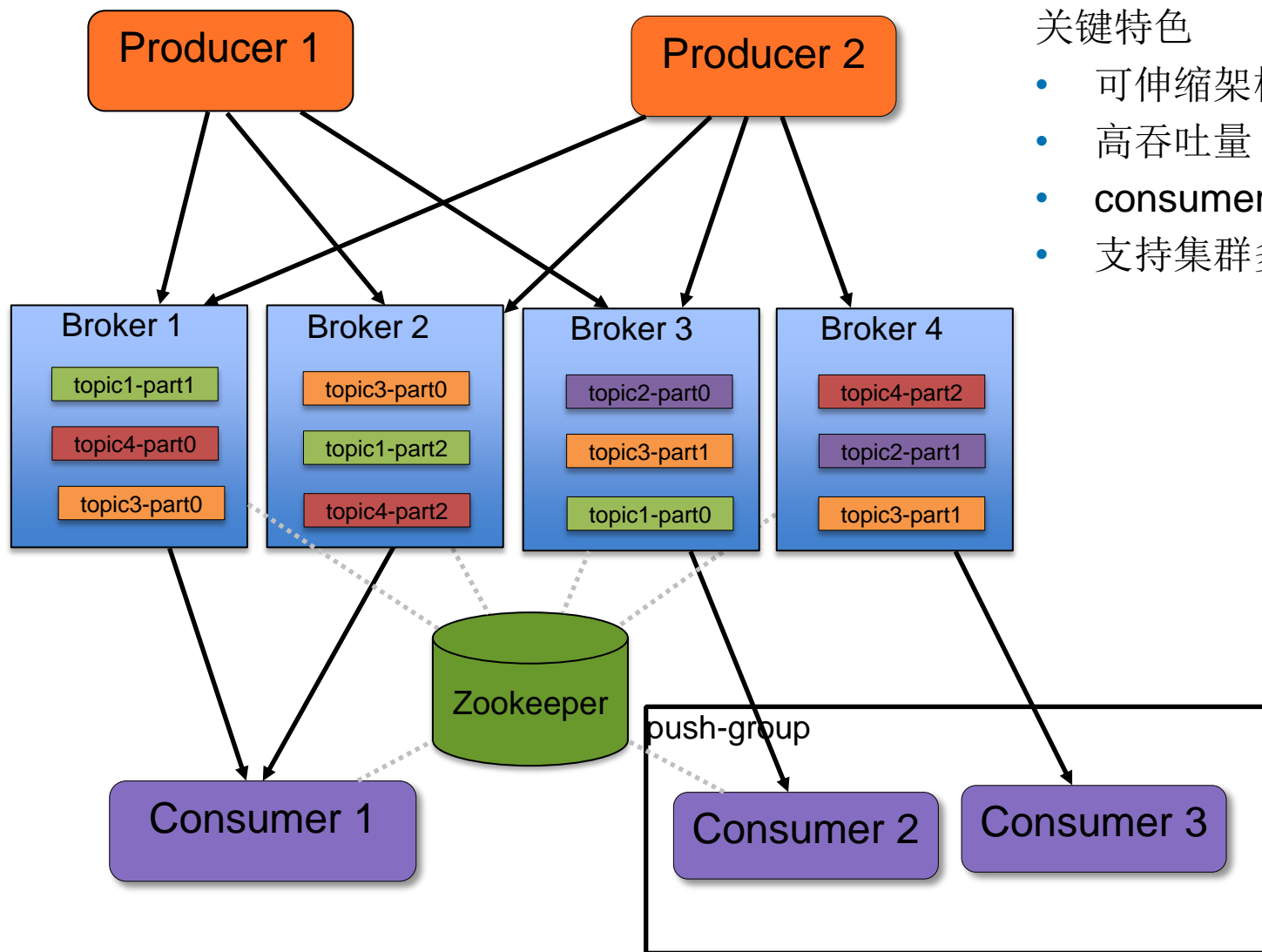
# Kafka 架构1



# Kafka 架构2



# Kafka 架构3



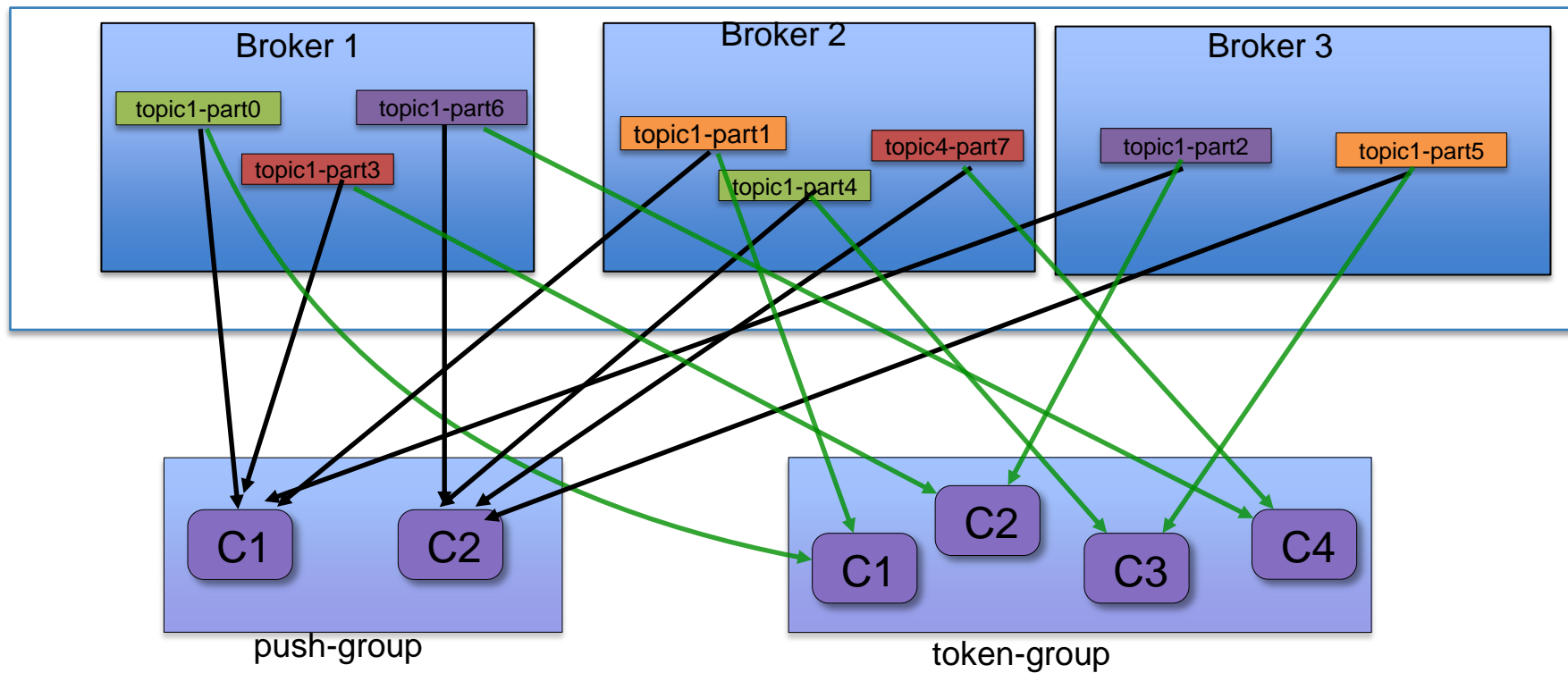
## 关键特色

- 可伸缩架构
- 高吞吐量
- **consumer**自动负载均衡
- 支持集群多副本



# consumer 负载均衡

## kafka cluster



# kafka中角色与术语

- **Producer:** 向Kafka发布消息的进程;
- **Consumer:** 从Kafka中订阅Topic的进程;
- **Consumer Group:** 同一个Consumer Group中的Consumers, Kafka将相应Topic中的每个消息只发送给其中一个Consumer;
- **Broker:** Kafka集群中的每一个Kafka服务;

# kafka中角色与术语

- Topic: 每一类消息;
- Partition: 对Topic中的消息做水平切分, 每块称为一个Partition;
- Replication: 将Partition复制, 每一份叫做一个Replication;
- Replication Leader: 每一个Partition都有一个“Leader”负责该Partition上所有的读写操作;
- Controller: kafka cluster center controller, 它负责分配partitions, 副本, 选举partition leader, 以及topic所有partitions重新重新负载均衡。
- Replication Follower: 每一个Partition都

# Partitions

- 每一个Topic被切分为多个Partitions;
- Broker Group中的每个Broker保存Topic的一个或多个Partitions;
- Consumer Group中的每个Consumer读取Topic的一个或多个Partitions, 并且是唯一的Consumer;
- Broker Group和Consumer Group都可以动态调整;
- Partition是Topic并发的基本单元, 并

# Replication

- 目的：当集群中有Broker挂掉的情况，系统可以自动地使用Replicas提供服务；
- 设置：系统默认设置每个Topic的replication系数为1，可以在创建Topic时单独设置；
- 特点：
  - Replication的基本单位是Topic的Partition；
  - 每个Partition都有一个Leader Replica，0个或多个Follower Replica；
  - 所有的读和写都从Leader进行，Followers只是做为备份；

# Replication

- In-Sync:
  - Broker必须和Zookeeper保持session;
  - Follower需要不能落后Leader太多  
(`replica.lag.max.messages`);
- 对Producer来说, 在写消息时可以根据自己需要设置可靠性保证级别, `request.required.acks`设置为0/1/-1;
- 对Consumer来说, 读也是从Leader进行, 所以只要提交的消息, Consumer肯定都可以读到;
- Leader选举: ISR Approach

# 消息投递原则—>稳定性

producer, consumer, broker挂掉情况分析:

1. producer同步发送消息到broker, producer进程crash
2. producer异步送消息到broker, producer进程crash
3. consumer从broker拉取消息, consumer挂掉。
  - a. 异步消费, 定期修改consumer状态到zookeeper。
4. 当kafka broker挂掉, 分三种情况:
  - a. broker设置为单副本实时同步消息到disk情况.
  - b. broker设置为单副本 + pagecache情况
  - c. broker设置为多副本 + pagecache情况

# 消息投递原则—>稳定性

## 保证消息稳定和可靠—>我是如何做的

1. 对不同topic(主题)，分类别设置不同的安全(消息传输可靠性)级别，目前设置了三种不同安全级别，这些安全级别设置根据需要也可以相应调整

### 高安全(可靠性)级别：

- kafka broker设置消息实时持久化到磁盘。
- topic 的partitions设置多副本，producer发送消息到broker需要同步回执ack信息，consumer的消费状态实时持久化到本地，并定期更新到zookeeper上。

### 中等安全(可靠性)级别：



# 消息投递原则—>稳定性

- kafka broker设置消息批量或定期持久化到磁盘。
- topic的partitions单副本，producer发送消息到broker需要同步回执ack信息，consumer的消费状态定期更新到zookeeper上。

低等安全(可靠性)级别，高吞吐量优先，

极端情况可以有数据丢失：

- kafka broker设置消息批量或定期持久化到磁盘。
- topic中partitions设置为单副本，单线程或多线程producer异步发送消息到broker，consumer消费大批量消息定期更新消费状态到zk上



kafka介绍



kafka架构 & 稳定性



性能优化



性能测试



kafka监控



mafka client开发

# kafka优化前



# 性能优化策略

## ✿ 系统级优化

- ✿ **disk**读取优化
- ✿ **os**推测读写优化
- ✿ zero copy
- ✿ tcp参数优化

## ✿ 应用级优化

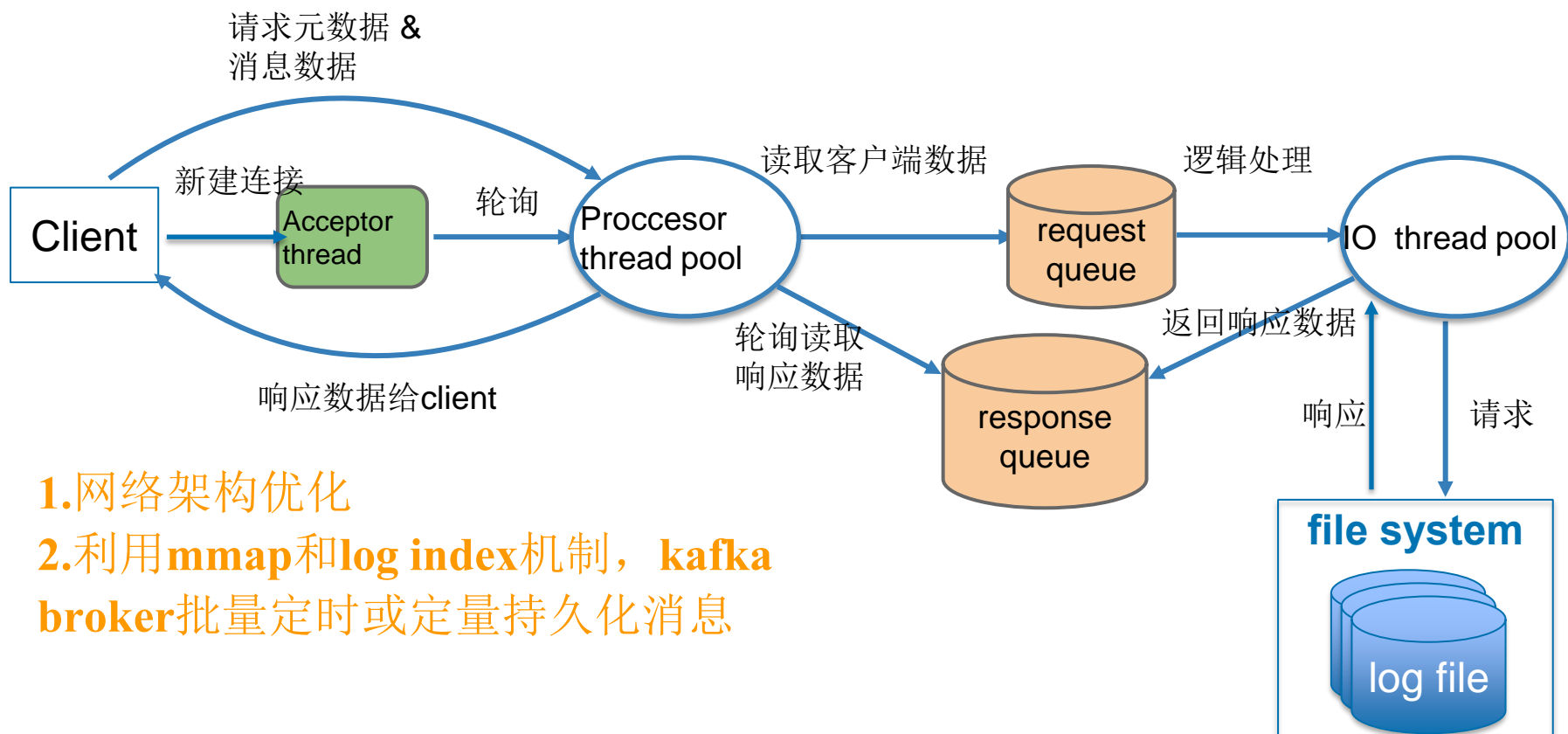
- ✿ kafka broker
- ✿ producer
- ✿ consumer

# kafka系统级性能优化

- Disk: 随机读写慢, 顺序读写快; 寻道时间;
- OS推测读写: read-ahead & write-ahead;
- Append messages : 顺序读写msgs数据;
- tcp参数优化: 调整缓冲区大小, 滑动窗口等。
- sendfile & zero copy: 减少字节copy;
- 文件 -> PageCache -> User buffer -> Socket Buffer -> NIC Buffer
- 文件 -> PageCache -> NIC Buffer

# kafka应用架构性能优化-broker

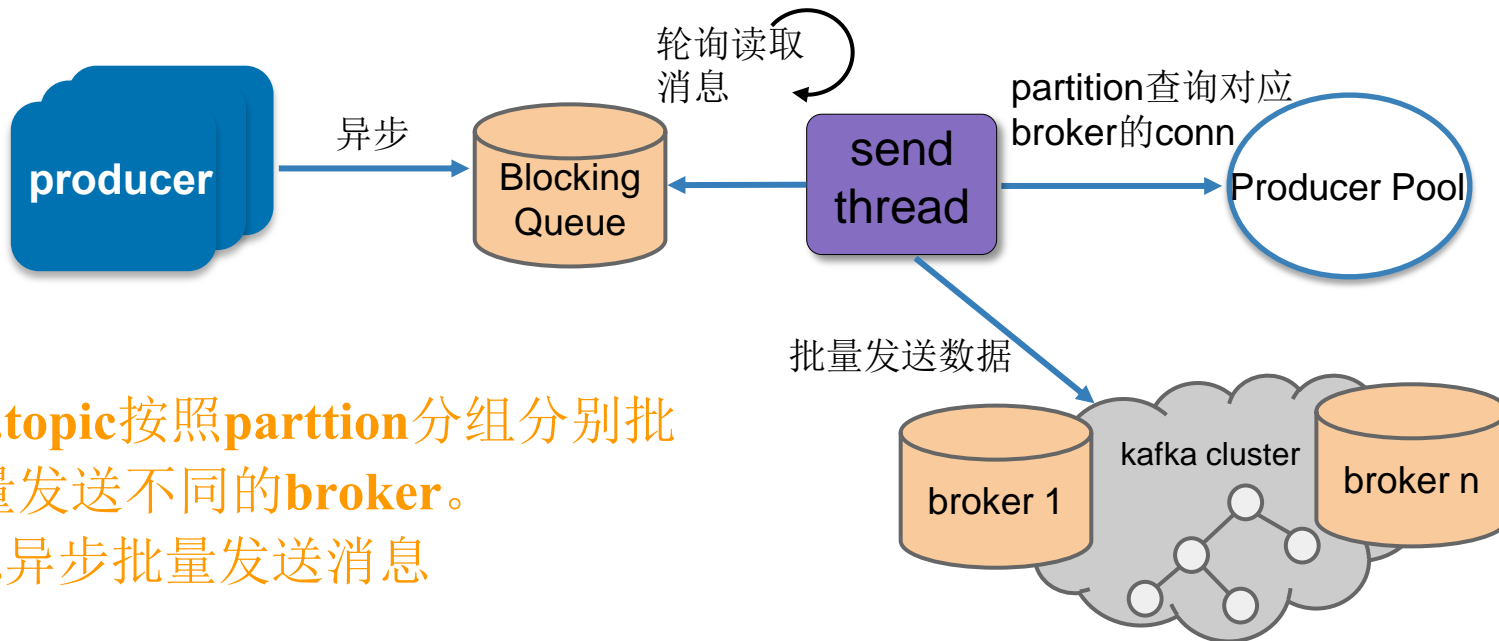
## kafka network 请求处理流程



1.网络架构优化

2.利用mmap和log index机制，kafka broker批量定时或定量持久化消息

# kafka应用架构性能优化-producer



1.topic按照partition分组分别批量发送不同的broker。

2.异步批量发送消息

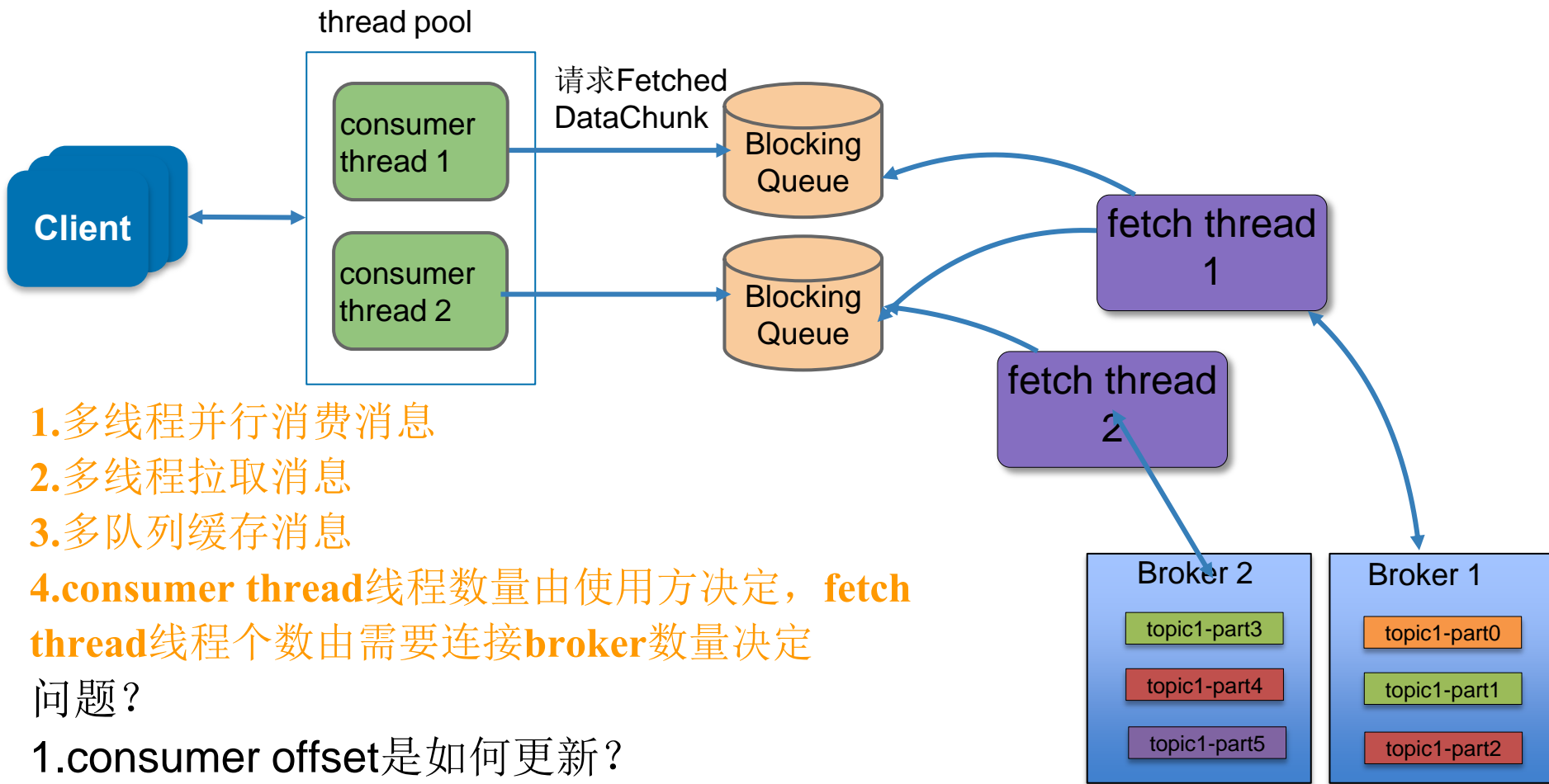
1.当producer同步发送时，client挂掉了，会否丢失数据

2.producer异步发送，当kafka broker挂掉，缓冲队列数据是否丢失

3.producer异步发送，当client挂掉，数据是否丢失

# kafka应用架构性能优化-consumer

kafka consumer高级API Zookeeper Consumer Connector处理逻辑





# kafka优化后



- 1 kafka介绍
- 2 kafka架构 & 稳定性
- 3 性能优化
- 4 性能测试
- 5 kafka监控
- 6 kafka client开发

# kafka性能测试

硬件环境：4 core 虚拟机 7200转stat硬盘 8GB内存 1Gb网卡，3个node组成一个kafka集群，分别有一个producer node和consumer node，以下是同时(运行)测试producer和consumer，而且consumer thread线程数量相同  
创建topic push-test 6 partitions

- **producer(sync) & consumer**

- **sync & 单线程测试**

producer速率:13MB/sec 左右 1.5w/sec consumer毫无压力

- **sync & 4线程测试 & 单副本**

producer速率:50MB/sec 左右 5w/sec consumer毫无压力

- **sync & 4线程测试**

单线程producer和consumer 2副本

# kafka性能测试

producer速率:34MB/sec 左右 3w/sec consumer毫无压力

- **producer(async) & consumer**

- **async & 单线程 & 单副本**

producer速率:60MB/sec 左右 6w/sec consumer毫无压力

- **async & 单线程 & 2副本**

producer速率:10MB/sec 左右 1.2w/sec consumer毫无压力

- **async & 2线程 单副本**

Pro速率:100MB/sec 10w/sec 左右 达到网卡极限 consumer即如此

- **async & 2线程 2副本**

Pro速率:87MB/sec 8w/sec 左右 达到网卡极限 consumer即如此

- 1 kafka介绍
- 2 kafka架构 & 稳定性
- 3 性能优化
- 4 性能测试
- 5 kafka监控
- 6 kafka client开发

# Monitor

- Kafka服务节点数监控
  - zookeeper上xxx/mafka01/broker/ids目录下节点数量;
- Kafka Broker监控
  - Broker是否存活/Broker是否提供服务;
  - 数据流量(流入和流出);
  - Producer的请求数/请求响应时间;
  - Consumer的请求数/请求响应时间;
- Topic监控
  - 数据量大小;
  - offset;
  - 数据流量 (流入和流出);

# kafka monitor zabbix参数

kafka.status.ActiveControllerCount	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.status.AllTopicsMessagesInPerSec	06 Aug 2014 19:00:53	0	-0.01	<a href="#">Graph</a>
kafka.topic.open_platform_opt_push-FailedProduceRequestsPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.status.ProducerPurgatorySize	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.status.ProducerRequestTotalTime	06 Aug 2014 19:00:53	1	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_push-BytesOutPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_sms-BytesInPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_push-FailedFetchRequestsPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.status.AllTopicsFailedFetchRequestsPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_sms-BytesOutPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_push_plus1-MessagesInPerSec	06 Aug 2014 19:00:53	0	-0.01	<a href="#">Graph</a>
kafka.status.FetchPurgatorySize	06 Aug 2014 19:00:53	7819	-213	<a href="#">Graph</a>
kafka.status.UnderReplicatedPartitions	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>
kafka.topic.open_platform_opt_push-BytesInPerSec	06 Aug 2014 19:00:53	0	-	<a href="#">Graph</a>

# 线上实践情况-push系统

部署环境介绍:

6台虚拟机, 每台4核8GB内存, 一台producer node, 5台consumer node

高峰时每天发送消息9000w条消息, 每条消息大小大约1kb/message左右。

producer:最高可以达到跑满1000MB网卡, 10w/sec

consumer:5台consumer总的消费速率1.1w/sec 每台流量100MB/sec

运行情况: 经过几番调整和优化, 目前已经稳定运行了一段时间。





kafka介绍



kafka架构 & 稳定性



性能优化



性能测试



kafka监控



mafka client开发

# mafka-client变迁

## 1 `mafka-client_v1.0.x`版

自定义序列化支持

`producer`支持多种负载均衡方式

消息跟踪

`consumer`串 | 并行消费

## 2 `mafka-client_v1.1.x`版

暴露序列化接口

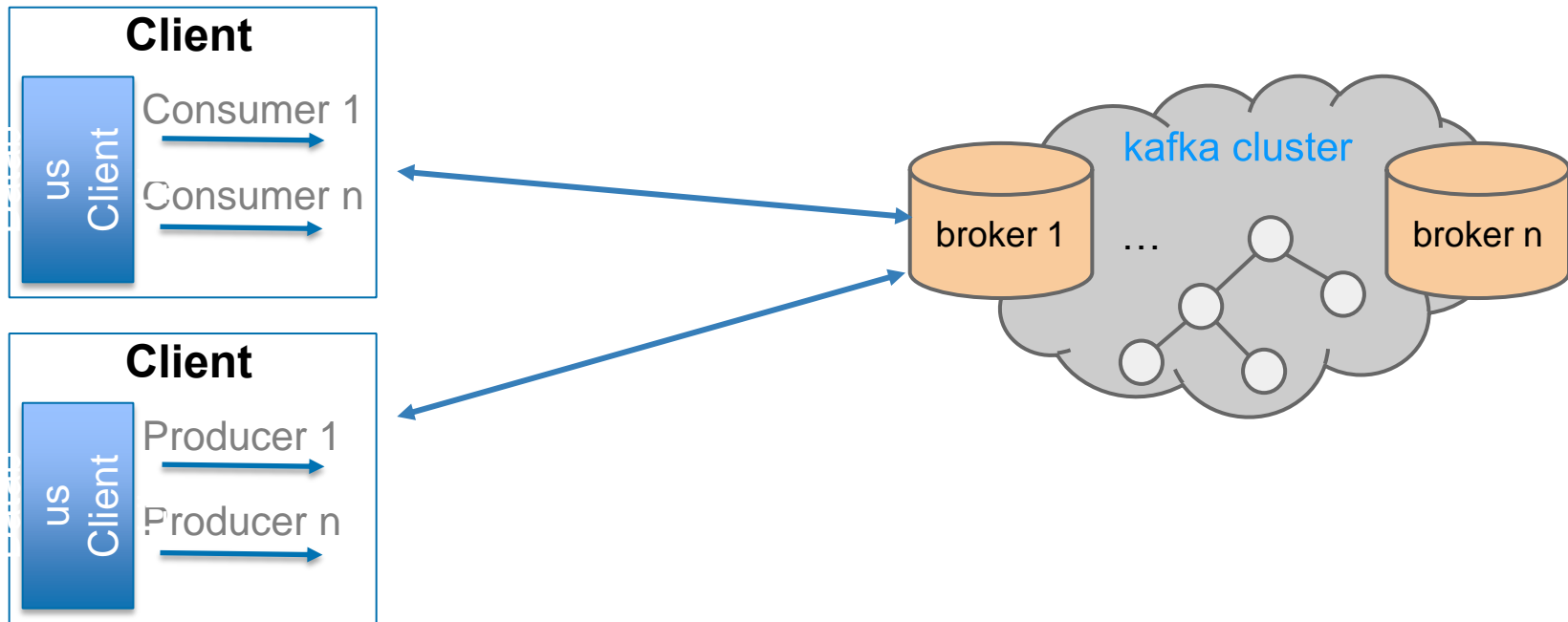
`producer`和`consumer`平滑扩容

`producer`和`consumer`集群平滑切换

`producer`和`consumer`在线配置参数优化

提供上报信息功能

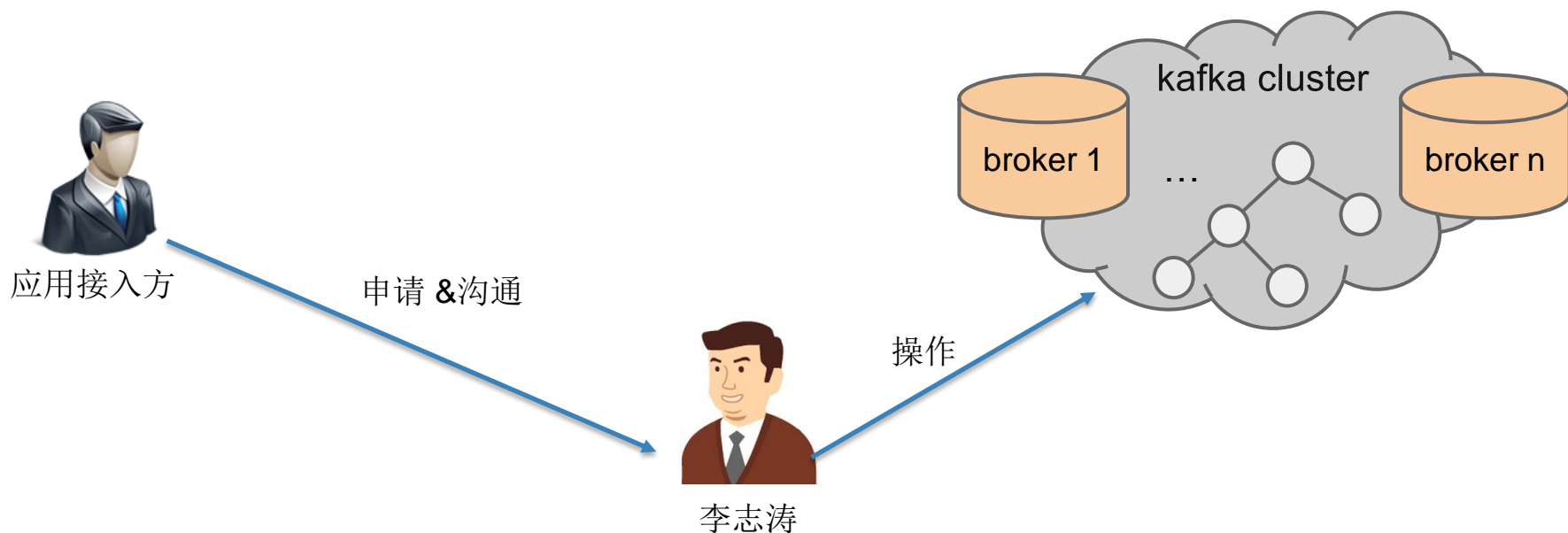
# kafka client功能 & 特性



- 1.客户端访问时**kafka**服务时，**kafka**集群间实现平滑切换
- 2.**kafka**集群内**broker**迁移或扩容。
- 3.**kafka**服务版本在线平滑升级
- 4.**producer**和**consumer**可以在线调参调优。
- 5.客户端上报运行版本和状态
- 6.实现**kafa broker**节点移除或下线

# kafka-mq应用&接入—准备工作

1. 申请appkey
2. 编写配置文件
3. 说明业务特点，部署环境，预估流量大小，吞吐量大小。对消息可靠性，延迟性等要求。



# mafka client

## 优点

- 1.接入方便，代码很少，容易上手，即看即用。
- 2.业务方不用关心实现服务端，以及架构，这些对于业务方是透明的
- 3.**mafka**客户端和**kafka cluster**可以进行平滑迁移或升级。

# mafka-client入口类—》MafkaClient

```
public class MafkaClient {
    private static final Logger logger = Logger.getLogger(MafkaClient.class);
    public static final String version = "1.0.13";
    static {
        logger.info("mafka current version:" + version);
    }
    /**
     *
     * @param topic    topic名称
     * @param strategyType 发送分区策略类型,分2种类型
     *    RoundRobinStrategy 分区配置: 指定分区索引表示为:0,1,2,3 设置分区索引区间为:[0-3]
     *    WeightStrategy    /* 按照轮询调研权重配置, 格式如下:partition1:weight,partition2:weight
     * @param partConfig 格式如上所示: [0-3]或partition1:weight,partition2:weight
     * @return IProducerProcessor 生产者处理类
     */
    public static IProducerProcessor buildProduceFactory(String topic,
                                                         ProduceStrategyType strategyType,
                                                         String partConfig) {

        return new DefaultProducerProcessor(topic,strategyType, partConfig);
    }
    /**
     * consumer入口类
     * @param topic
     * @return
     */
    public static IConsumerProcessor buildConsumerFactory(String topic) {

        return new DefaultConsumerProcessor(topic);
    }
}
```

# mafka-client 接口—》 prouducer













```
public interface IProducerProcessor<K,V> {  
    /**  
     * 任何可以序列化的类型  
     * @param message  
     * @throws Exception  
     */  
    public void sendMessage(V message) throws Exception;  
  
    /**  
     *  
     * @param message 任何可以序列化的类型  
     * @param partKey 自定义partKey发送分区消息:  
     * (自定义规则)可以按照partKey规则, 发送消息到指定的分区  
     * @throws Exception  
     */  
    public void sendMessage(V message, Object partKey) throws Exception;  
  
    /**  
     * 关闭 producer和释放所有资源(包括broker,zookeeper等连接和数据资源)  
     */  
    public void close();  
}
```

# mafka-client 接口—》 consumer

```
public interface IConsumerProcessor {  
  
    /**  
     *  
     * @param type 序列化类型,包括(String,List,Map,Bean,java 8种类类型)  
     * @param messageListener  
     */  
    public void recvMessage(final Class type, final IMessageListener messageListener);  
  
    /**  
     * 多线程并行方式消费数据  
     * @param type 序列化类  
     * @param numThreads 线程数量  
     * @param messageListener  
     */  
    public void recvMessageWithParallel(final Class type,  
                                           final int numThreads,  
                                           final IMessageListener messageListener);  
  
    /**  
     * 手工更新consumer的offset到zookeeper  
     */  
    public void saveOffsetsToZookeeper();  
  
    /**  
     * 关闭consumer  
     */  
    public void close();  
}
```



# mafkafka client开发—》配置环境

- ▼  kafka-test [mafkafka-test] (~/mt\_wp/redis-cluster-  
  - ▶  dist
  - ▶  log
  - ▶  mafkafka-test
  - ▼  src
    - ▼  main
      - ▶  assemble
      - ▶  java
      - ▼  resources
        -  consumer.properties
        -  producer.properties
        -  start.sh

# mafka client开发—》 配置文件路径

客户端配置文件路径：

1. 默认情况下分为2种，
  - a. 非web工程，在项目工程跟目录下读取
  - b. web工程，则需要把配置放置到classes目录下.
2. 自定义路径(可选)
  - a. 设置系统变量可以自定义读取路径

例如

```
System.setProperty("mafka.conf.path", "/opt/xxx/");
```

# mafka client开发—>producer.proterties

zookeeper.connect=192.168.2.225:2181,192.168.2.225:2182,192.168.2.225:2183/config/mobile/mq

zookeeper.session.timeout.ms=5000

zookeeper.connection.timeout.ms=10000

mafka.client.appkey=appkey1

#producer.proterties与consumer.proterties配置完全相同

## mafka client开发demo:

[http://wiki.sankuai.com/pages/viewpage.action?pageId=85047298#mafkaclientA](http://wiki.sankuai.com/pages/viewpage.action?pageId=85047298#mafkaclientAPI调用说明-7.mafkaclientdemo)

[PI调用说明-7.mafkaclientdemo](#)

# 大家快来接入吧



一声呐喊，敲锣打鼓欢迎大家使用或接入消息服务

# 参考

linkined相关文档

kafka基础培训 鞠大升

<http://kafka.apache.org/>

<http://kafka.apache.org/performance.html>

<http://blog.csdn.net/lizhitao>

Thank you!

Any Quest?

