火龙果•整理
uml.org.cn

Kafka架构原理

火龙果•整理
uml.org.cn

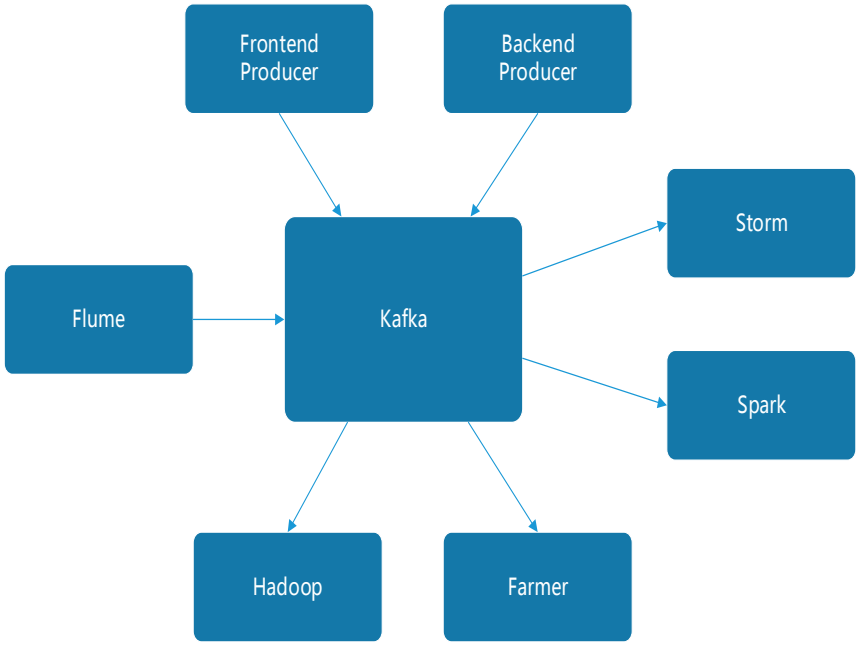
目录

1. Kafka应用场景
2. Kafka架构与功能
3. Kafka关键流程
4. Kafka常用命令和参数
5. Kafka高级专题

Kafka应用场景

Kafka – A high-throughput, distributed, publish-subscribe messaging system，是一个高吞吐、分布式、基于发布订阅的消息系统，利用Kafka技术可在廉价PC Server上搭建起大规模消息系统。

Kafka具有消息持久化、高吞吐、分布式、多客户端支持、实时等特性，适用于离线和在线的消息消费，如常规的消息收集、网站活性跟踪、聚合统计系统运营数据（监控数据）、日志收集等大量数据的互联网服务的数据收集场景。



使用消息系统的原因

- 解耦：消息系统在处理过程中插入一个隐含、基于数据的接口层。
- 冗余：消息队列持久化，防止数据丢失。
- 扩展性：消息队列解耦处理过程，容易扩展处理过程。
- 可恢复性：处理过程失效，恢复后可继续处理。
- 顺序保证：消息队列保证顺序。**Kafka**保证一个**Partition**内消息有序。
- 异步通信：消息队列允许消息加入队列，等需要时再处理。

火龙果•整理
uml.org.cn

目录

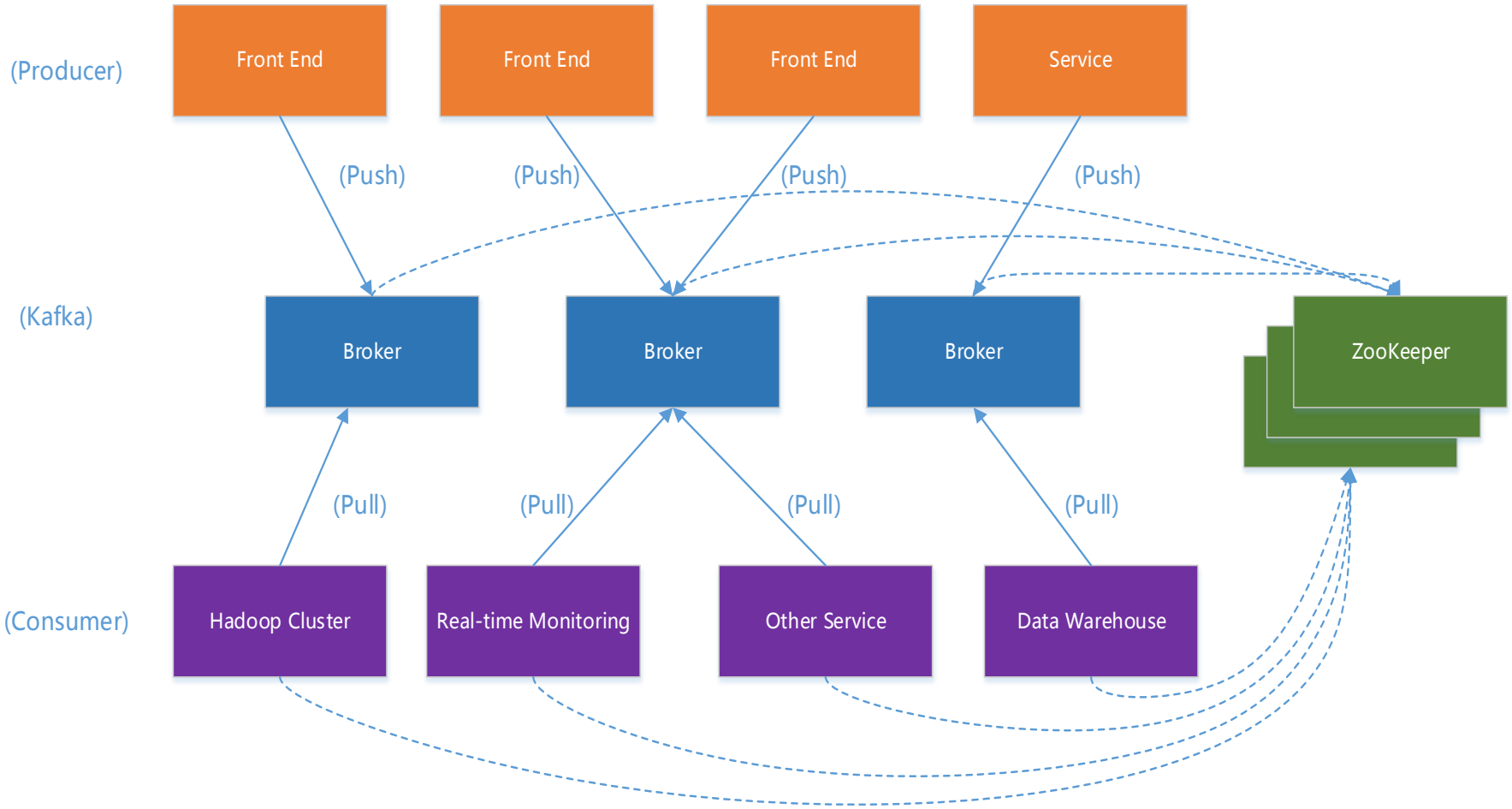
1. Kafka应用场景
2. **Kafka架构与功能**
3. Kafka关键流程
4. Kafka常用命令和参数
5. Kafka高级专题

Kafka术语

- **Broker**：Kafka集群包含一个或多个服务实例，这种服务实例被称为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。
- **Partition**：Partition是物理上的概念，每个Topic包含一个或多个Partition.
- **Producer**：负责发布消息到Kafka Broker。
- **Consumer**：消息消费者，向Kafka Broker读取消息的客户端。
- **Consumer Group**：每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name）。

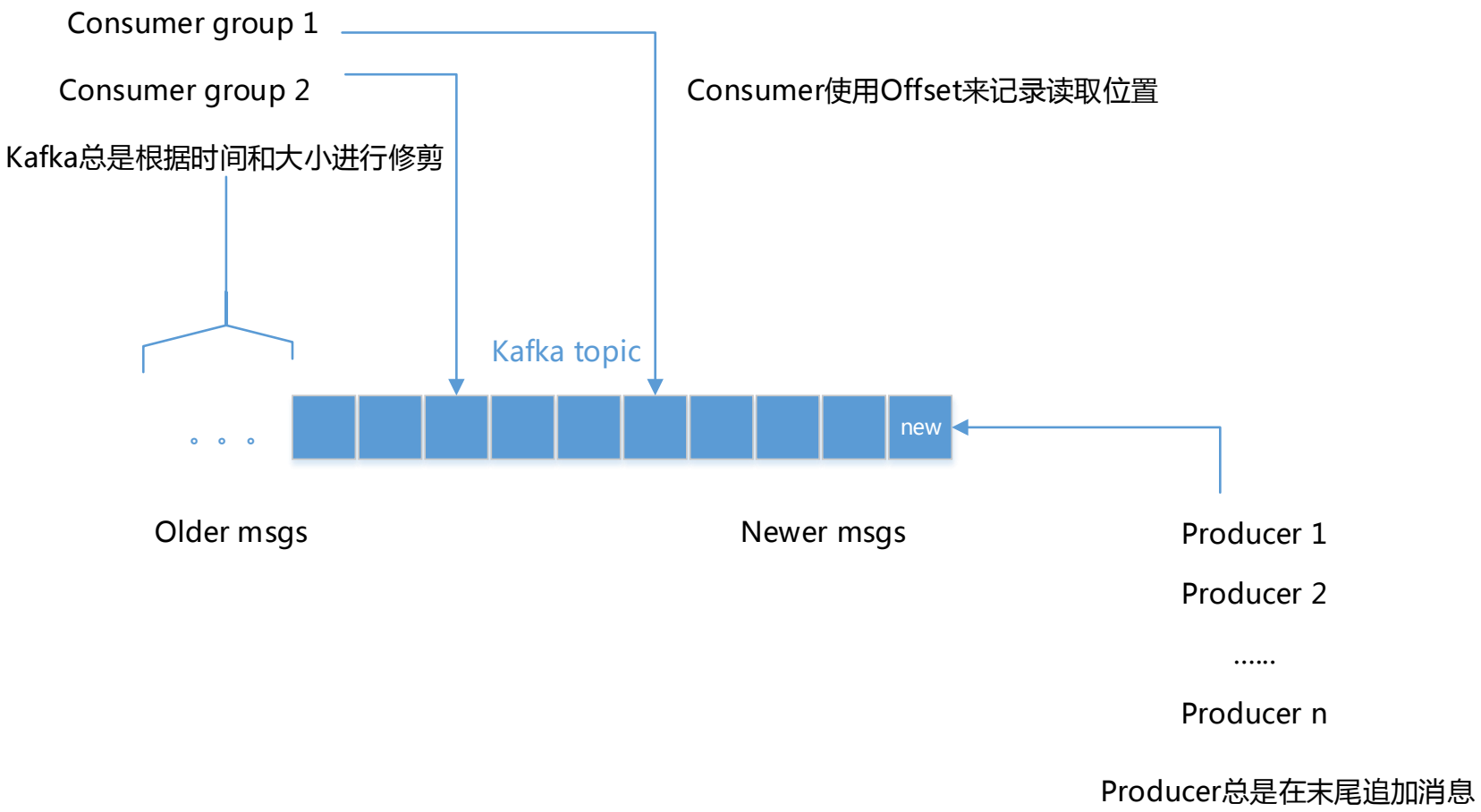
火龙果•整理
uml.org.cn

Kafka拓扑结构图



火龙果•整理
uml.org.cn

Kafka Topics

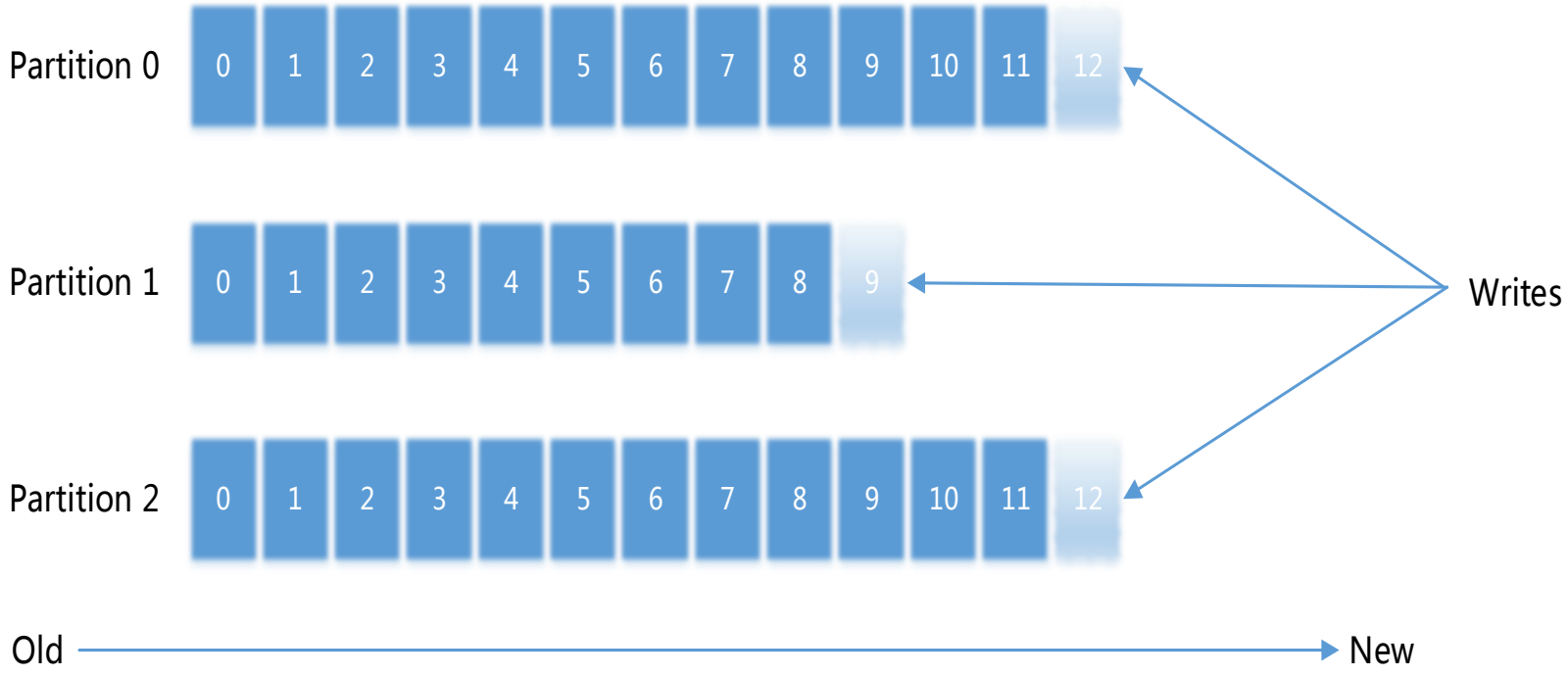


火龙果•整理
uml.org.cn

Kafka Partitions

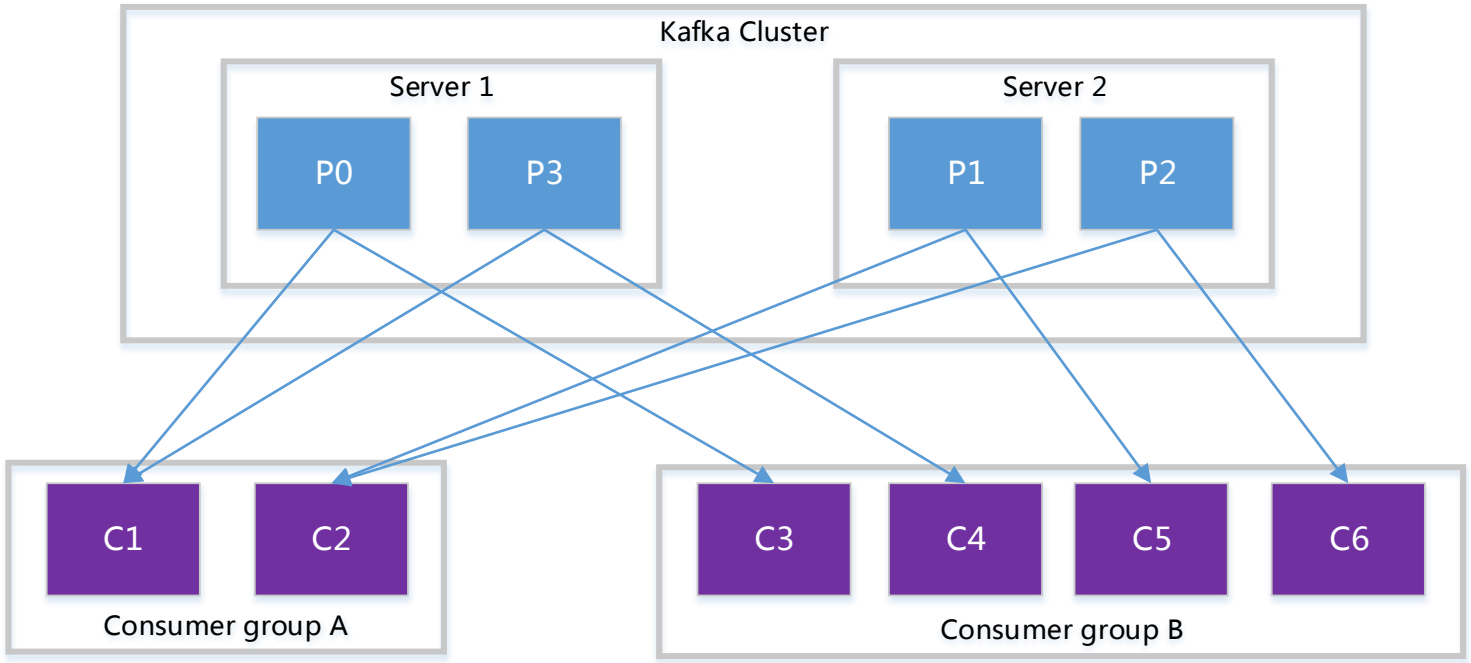
每个Topic 都有一个或者多个Partitions构成。

每个Partition都是有序且不可变的消息队列。



火龙果•整理
uml.org.cn

Kafka Partitions



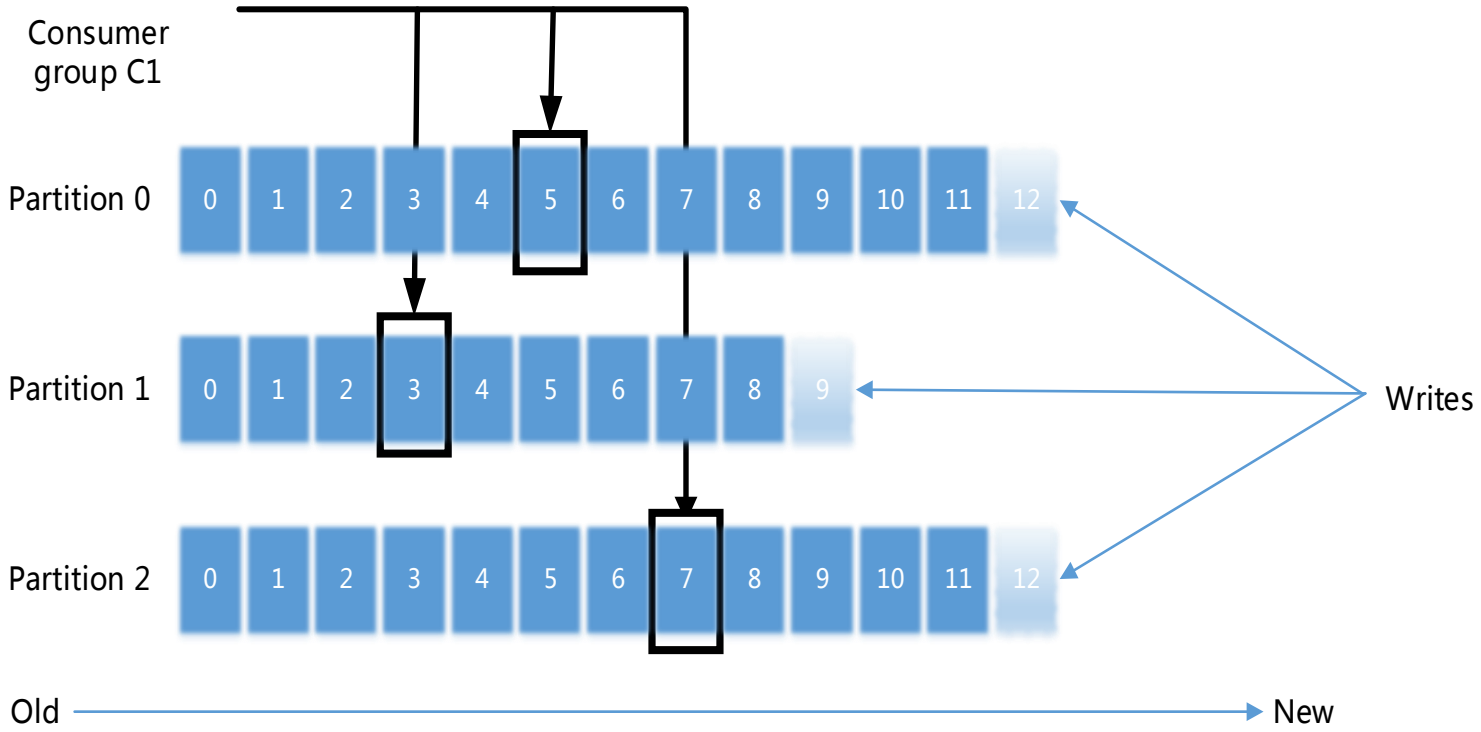
Topic的Partition数量可以在创建时配置。

Partition数量决定了每个Consumer group中并发消费者的最大数量。

Consumer group A有两个消费者来读取4个partition中数据；Consumer group B有四个消费者来读取4个partition中数据。

火龙果•整理
uml.org.cn

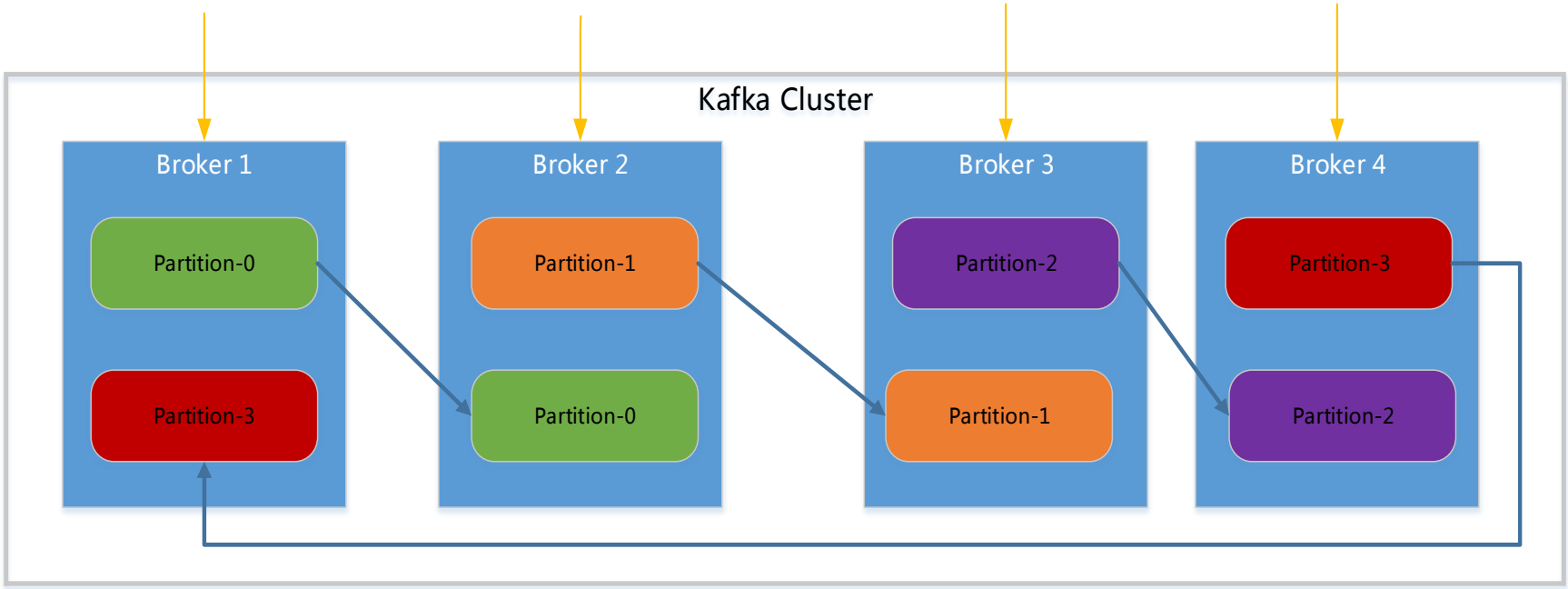
Kafka Partition offset



任何发布到此partition的消息都会被直接追加到log文件的尾部，

每条消息在文件中的位置称为offset（偏移量），offset是一个long型数字，它唯一标记一条消息。消费者通过（offset、partition、topic）跟踪记录。

Kafka Partition Replicas



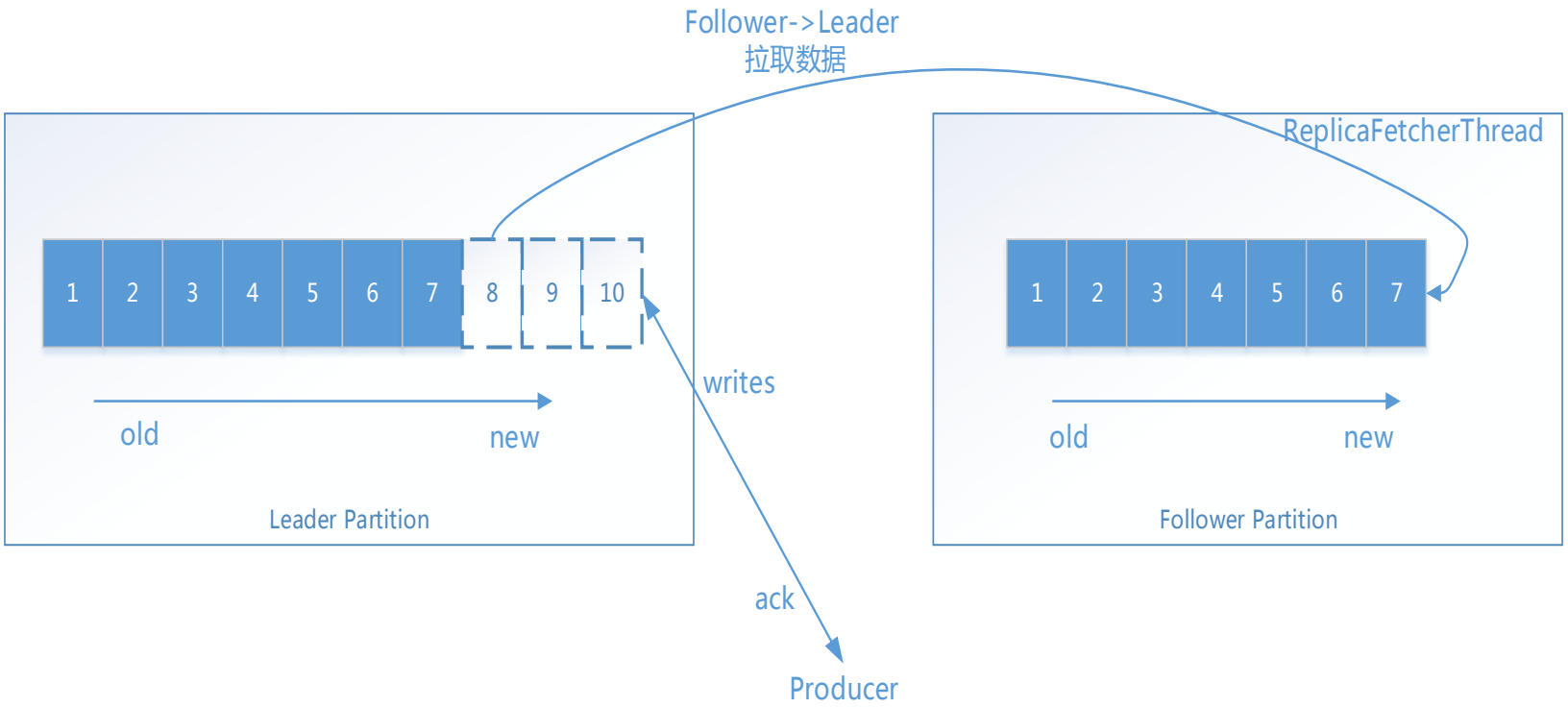
副本以分区为单位。每个分区都有各自的主副本和从副本。

主副本叫做leader，从副本叫做follower，处于同步状态的副本叫做in-sync replicas（ISR）。

Follower通过拉的方式从leader同步数据。消费者和生产者都是从leader读写数据，不与follower交互。

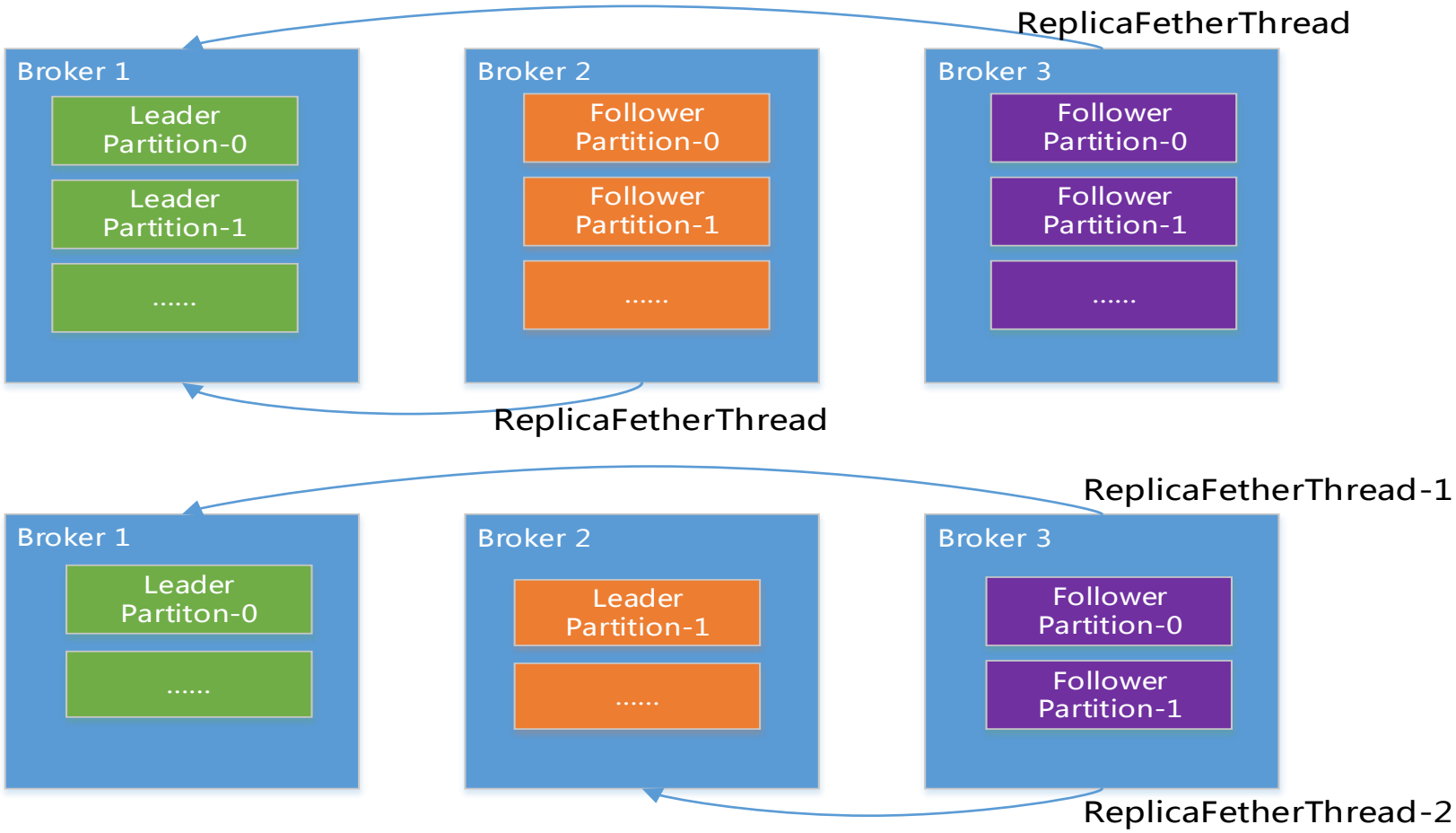
火龙果•整理
uml.org.cn

Kafka Partition Replicas



火龙果•整理
uml.org.cn

Kafka Partition Replicas

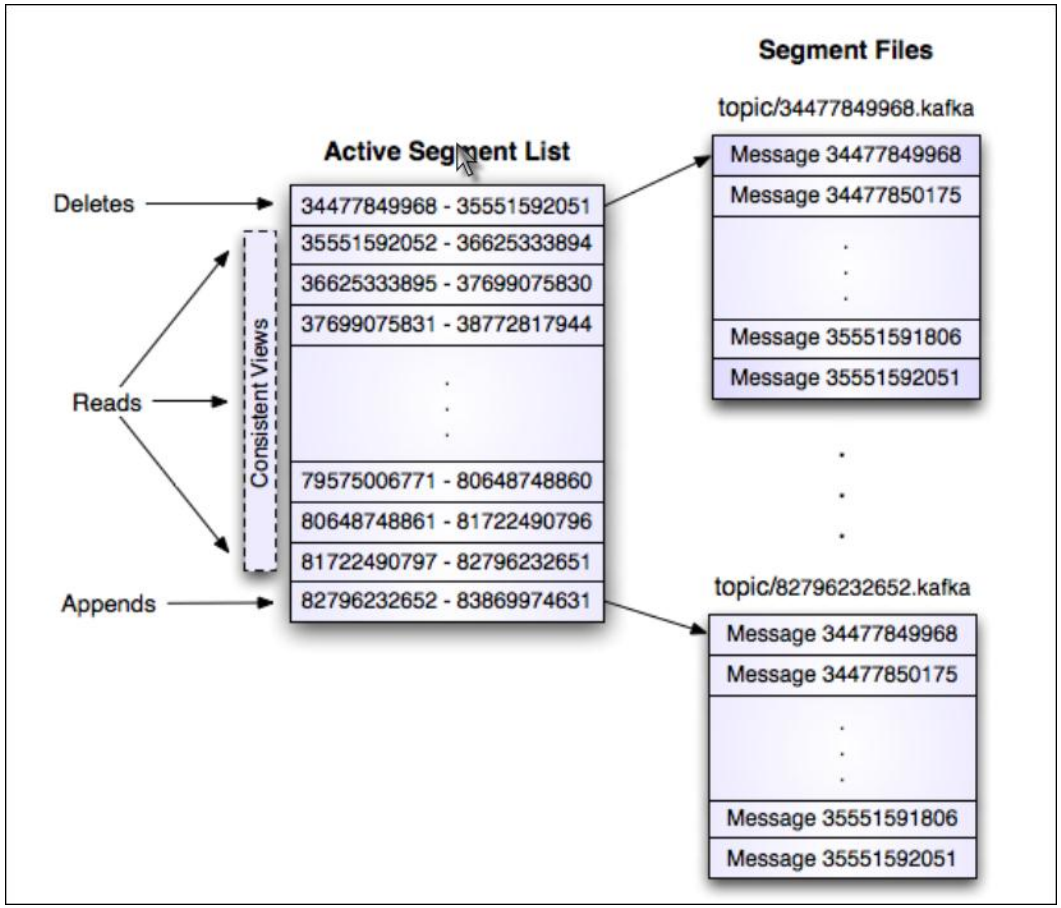


Kafka Logs

```
-rw------- 1 omm wheel    0 Jun 10 11:58 .lock
drwx------ 2 omm wheel 4096 Jun 12 14:59 example-metric1-0
-rw------- 1 omm wheel  102 Jun 13 17:57 recovery-point-offset-checkpoint
-rw------- 1 omm wheel  107 Jun 13 17:58 replication-offset-checkpoint
drwx------ 2 omm wheel 4096 Jun 12 10:52 test-0
drwx------ 2 omm wheel 4096 Jun 12 10:52 test-1
drwx------ 2 omm wheel 4096 Jun 12 11:08 test2-0
drwx------ 2 omm wheel 4096 Jun 12 11:08 test2-2
drwx------ 2 omm wheel 4096 Jun 12 14:59 test3-0
drwx------ 2 omm wheel 4096 Jun 12 14:59 test3-1
drwx------ 2 omm wheel 4096 Jun 12 14:59 test4-0
drwx------ 2 omm wheel 4096 Jun 12 14:59 test4-1
```

```
-rw------- 1 omm wheel 10485760 Jun 13 13:44 00000000000000000000.index
-rw------- 1 omm wheel  1081187 Jun 13 13:45 00000000000000000000.log
```

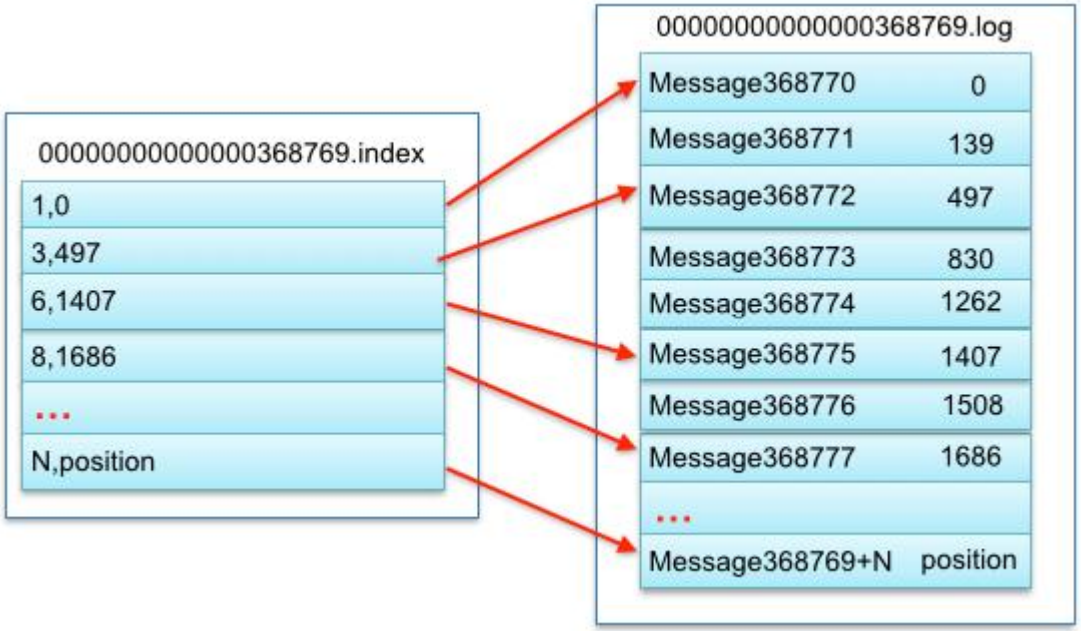
Kafka Logs



segment file组成：由2大部分组成，分别为index file和data file，此2个文件一一对应，成对出现，后缀".index"和“.log”分别表示为segment索引文件、数据文件.

segment文件命名规则：partion全局的第一个segment从0开始，后续每个segment文件名为上一个全局partion的最大offset(偏移message数)。数值最大为64位long大小，19位数字字符长度，没有数字用0填充。

Kafka Logs

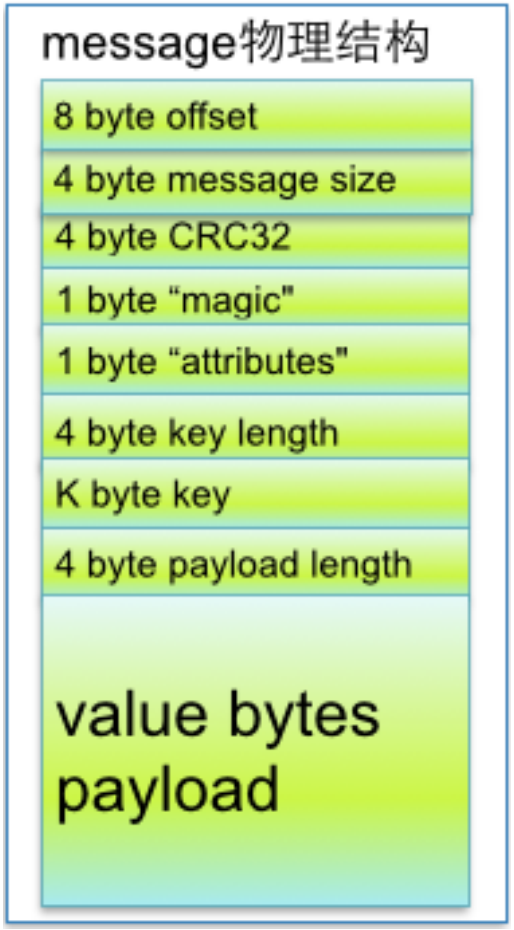


通过索引信息可以快速定位message。

通过index元数据全部映射到memory，可以避免segment file的IO磁盘操作。

通过索引文件稀疏存储，可以大幅降低index文件元数据占用空间大小。

Kafka Message



关键字	解释说明
8 byte offset	在partition（分区）内的每条消息都有一个有序的id号，这个id号被称为偏移（offset），它可以唯一确定每条消息在partition（分区）内的位置，即offset表示partition的第几个message
4 byte message size	Message大小
4 byte CRC32	用CRC32校验message
1 byte “magic”	表示本次发布Kafka服务程序协议版本号
1 byte “attributes”	表示为独立版本、或标识压缩类型、或编码类型
4 byte key length	表示key的长度，当key为-1时，K byte key字段不填
K byte key	可选
Value bytes payload	表示实际消息数据

Kafka Log Cleanup

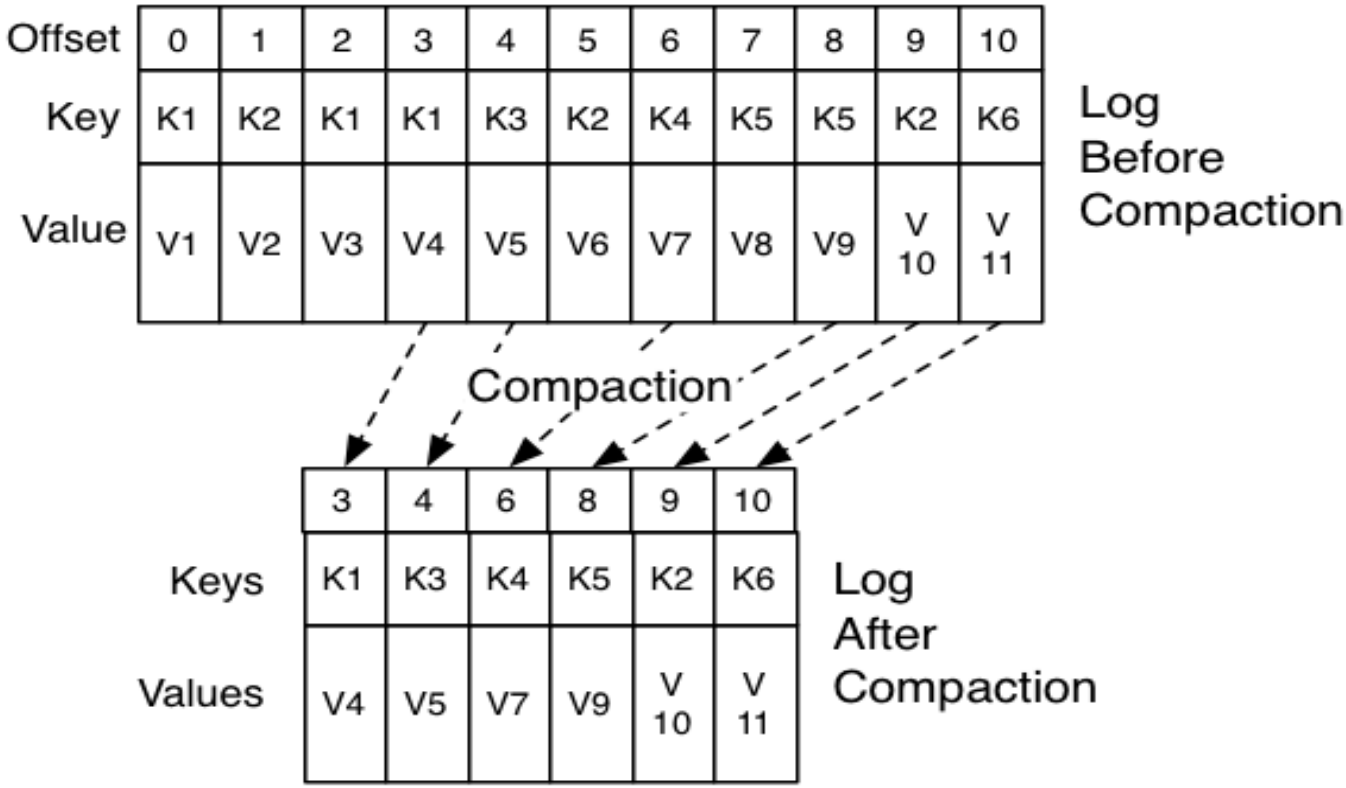
日志的清理方式有两种：delete 和 compact。

删除的阈值有两种：过期的时间和分区内总日志大小。

log.cleanup.policy	delete	This can take either the value *delete* or *compact*. If *delete* is set, log segments will be deleted when they reach the size or time limits set. If *compact* is set **log compaction** will be used to clean out obsolete records. This setting can be overridden on a per-topic basis (see **the per-topic configuration section**).
log.retention.{ms,minutes,hours}	7 days	The amount of time to keep a log segment before it is deleted, i.e. the default data retention window for all topics. Note that if both log.retention.minutes and log.retention.bytes are both set we delete a segment when either limit is exceeded. This setting can be overridden on a per-topic basis (see **the per-topic configuration section**).
log.retention.bytes	-1	The amount of data to retain in the log for each topic-partitions. Note that this is the limit per-partition so multiply by the number of partitions to get the total data retained for the topic. Also note that if both log.retention.hours and log.retention.bytes are both set we delete a segment when either limit is exceeded. This setting can be overridden on a per-topic basis (see **the per-topic configuration section**).

火龙果•整理
uml.org.cn

Kafka Log Cleanup



火龙果•整理
uml.org.cn

Message Delivery Semantics

- 最多一次（At Most Once）
  - 消息可能丢失。
  - 消息不会重复发送和处理。
- 最少一次（At Lease Once）
  - 消息不会丢失。
  - 消息可能会重复发送和处理。
- 仅有一次（Exactly Once）
  - 消息不会丢失。
  - 消息仅被处理一次。

火龙果•整理
uml.org.cn

Kafka Message Delivery

	同步发送不带确认	同步发送带确认	异步发送不带确认	异步发送带确认不重试	异步发送带确认有重试
无副本	最多一次	至少一次	最多一次	至少一次	至少一次
同步复制（**leader fellow**）	最多一次	至少一次	最多一次	至少一次	至少一次
异步复制 **(leader)**	最多一次	消息可能丢失或重复	最多一次	消息可能丢失或重复	消息可能丢失或重复

火龙果•整理
uml.org.cn

Kafka Message Delivery

高级消费者API默认保证at least once。
高级消费者API将消息的偏移量周期性地提交到Zookeeper中。

如果需要实现消息消费的exactly once，需要基于低级消费者API进行进一步的包装和开发。
低级消费者API允许消费者任意指定偏移量的保存方式。通过将消息的偏移量和消息本身保存到同一个位置从而实现原子操作，或者通过引入两阶段提交保证exactly once。

火龙果•整理
uml.org.cn

目录

1. Kafka应用场景
2. Kafka架构与功能
3. **Kafka关键流程**
4. Kafka常用命令和参数
5. Kafka高级专题

火龙果•整理
uml.org.cn

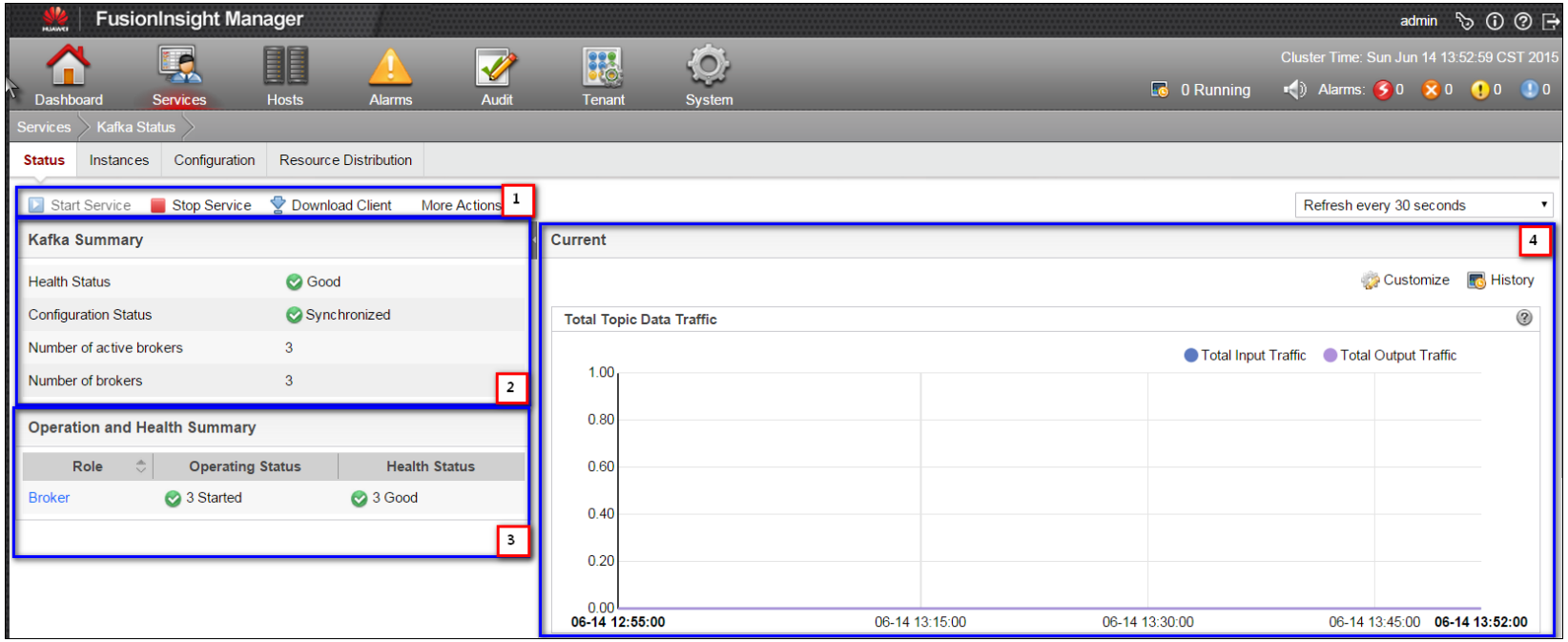
Kafka部署原则

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
Kafka	Broker	1GB	ZooKeeper	至少部署两个在数据节点上。若每天产生的数据量超过2T，建议单独部署在多个数据节点上。

注：一般采用以上方式部署高可靠的模式，即采用两个Broker，且副本因子为2。具体部署方法参见安装指南。

火龙果•整理
uml.org.cn

Kafka在Manager的Portal上呈现



1、管理操作，如启停、下载客户端等

2、服务总体概况

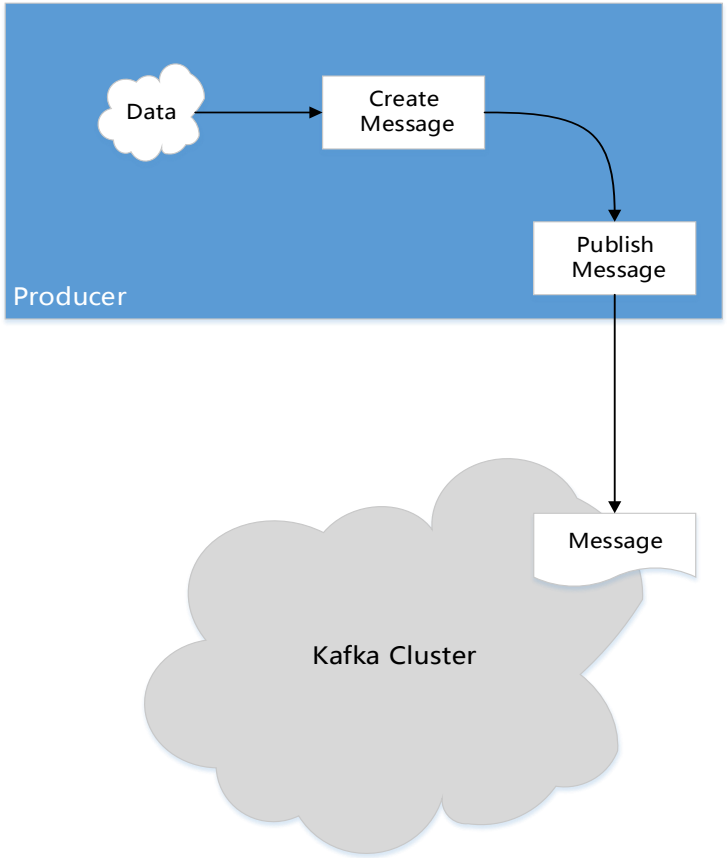
3、各组件操作和健康状态

4、指标动态图

Kafka关键流程

1. Kafka写流程
2. Kafka读流程

Producer写数据



●总体流程：

Producer连接任意存活的Broker，请求制定Topic、Partition的Leader元数据信息，然后直接与对应的Broker直接连接，发布数据。

●开放分区接口：

用户可以制定分区函数，使得消息可以根据Key，发送到特定Partition。

●构建Message对象：

一个Message对象代表一条数据记录

`KeyedMessage[K, V](val topic: String, val key: K, val message: V)`。

●Single or List?

Kafka提供了两个主要的写数据接口，一个可以写入单条数据，

`send(message: KeyedMessage[K,V])`。

另一个可以批量写入多条数据，

`send(messages: java.util.List[KeyedMessage[K,V]]`。

Producer写数据

```
public class CustomPartitioner<T> implements Partitioner {

    public CustomPartitioner(Properties properties) {}

    @Override
    public int partition(Object key, int numPartitions) {
        try {
            int partitionNum = Integer.parseInt((String) key);
            return Math.abs(Integer.parseInt((String) key) % numPartitions);
        } catch (Exception e) {
            return Math.abs(key.hashCode() % numPartitions);
        }
    }
}
```

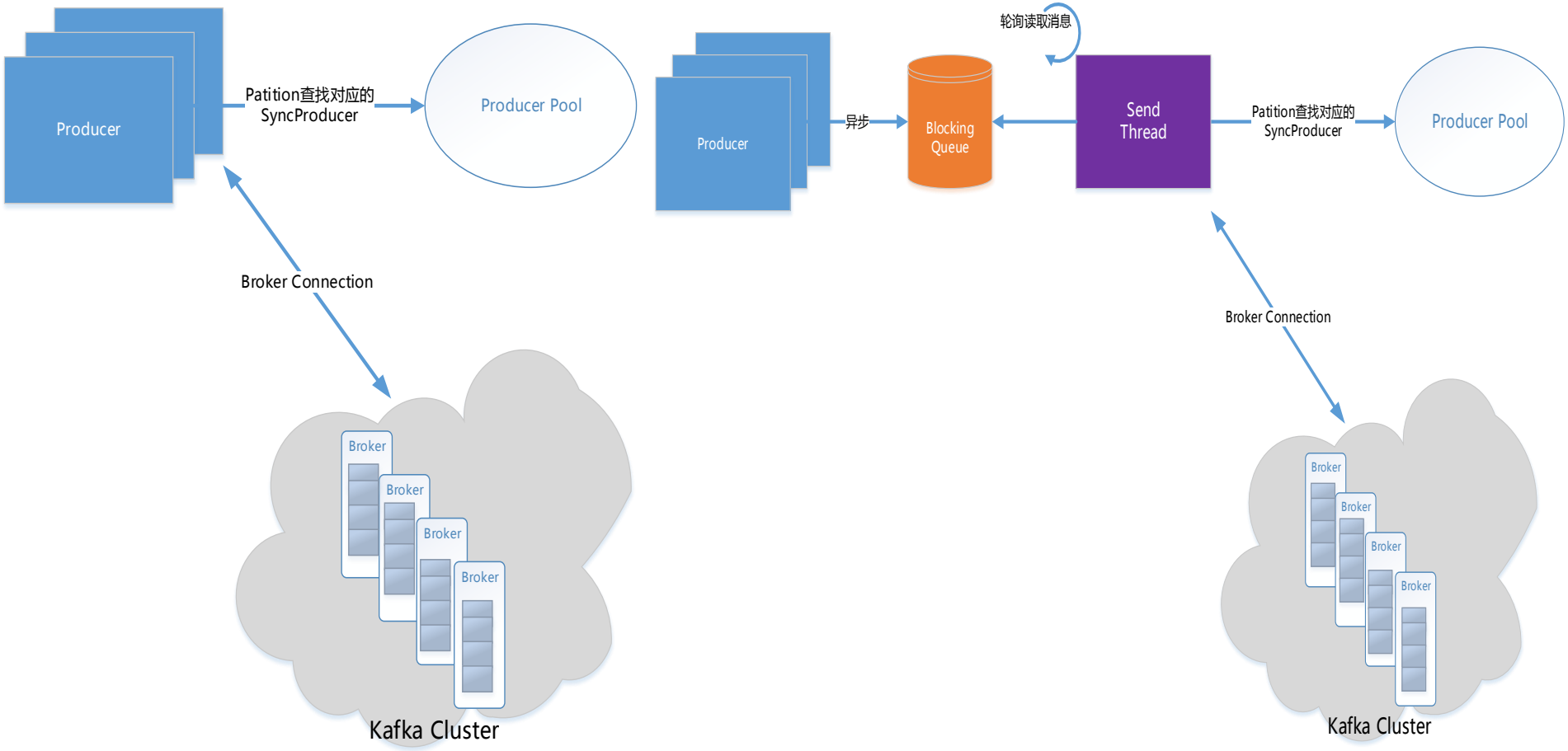
Producer写数据

```
Partition:3,    Message Payload: The 1 message for key 3
Partition:3,    Message Payload: The 2 message for key 3
Partition:3,    Message Payload: The 3 message for key 3
Partition:3,    Message Payload: The 4 message for key 3
Partition:3,    Message Payload: The 5 message for key 3
Partition:1,    Message Payload: The 1 message for key 1
Partition:1,    Message Payload: The 2 message for key 1
Partition:1,    Message Payload: The 3 message for key 1
Partition:1,    Message Payload: The 4 message for key 1
Partition:1,    Message Payload: The 5 message for key 1
Partition:2,    Message Payload: The 1 message for key 2
Partition:2,    Message Payload: The 2 message for key 2
Partition:2,    Message Payload: The 3 message for key 2
Partition:2,    Message Payload: The 4 message for key 2
Partition:2,    Message Payload: The 5 message for key 2
Partition:0,    Message Payload: The 1 message for key 0
Partition:0,    Message Payload: The 2 message for key 0
Partition:0,    Message Payload: The 3 message for key 0
Partition:0,    Message Payload: The 4 message for key 0
Partition:0,    Message Payload: The 5 message for key 0
```

```
public void sendMessage() throws InterruptedException{
    for(int i = 1; i <= 5; i++){
        List messageList = new
ArrayList<KeyedMessage<String, String>>();
        for(int j = 0; j < 4; j++ ) {
            messageList.add(new KeyedMessage<String,
String>("topic", j+"", "The " + i + " message for key " + j));
        }
        producer.send(messageList);
    }
    producer.close();
}
```

火龙果•整理
uml.org.cn

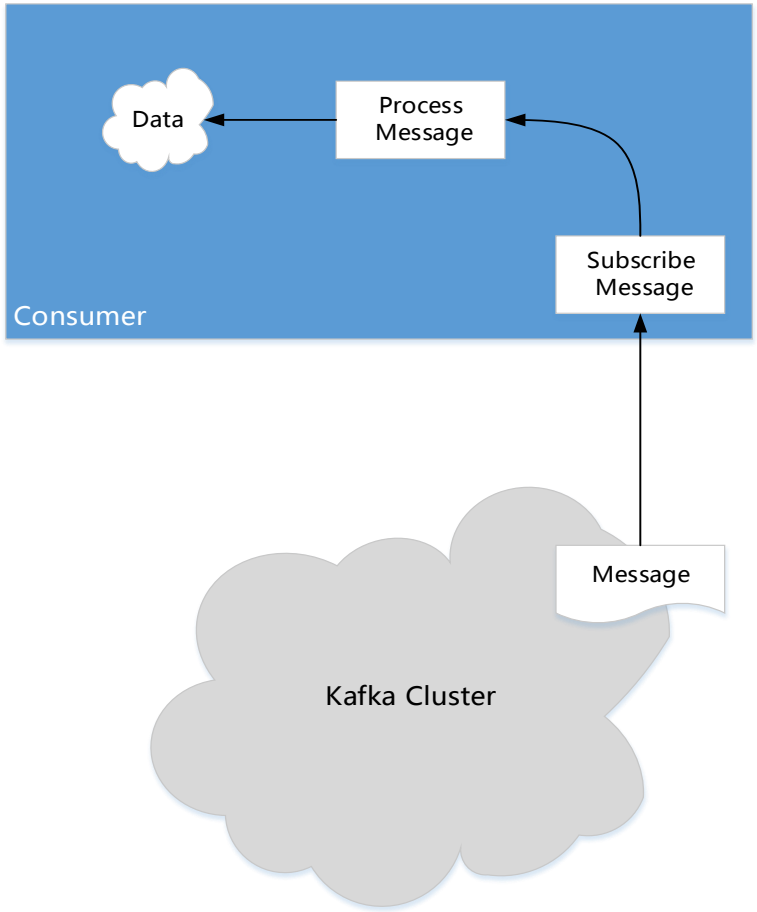
Producer (Sync vs Async)



Kafka关键流程

1. Kafka写流程
2. Kafka读流程

Consumer读数据



●总体流程：

Consumer连接指定Topic Partition所在的Leader Broker，采用pull方式从Kafka Logs中获取消息。

●High Level API：

简单

隐藏Consumer与Broker细节；相关信息保存在Zookeeper中。

●Low Level API：

非常灵活

需要自己负责维护链接Leader Broker， 保存Offset，Consumer Partition对应关系。

火龙果•整理
uml.org.cn

目录

1. Kafka应用场景
2. Kafka架构与功能
3. Kafka关键流程
4. **Kafka常用命令和参数**
5. Kafka高级专题

ZooKeeper Shell

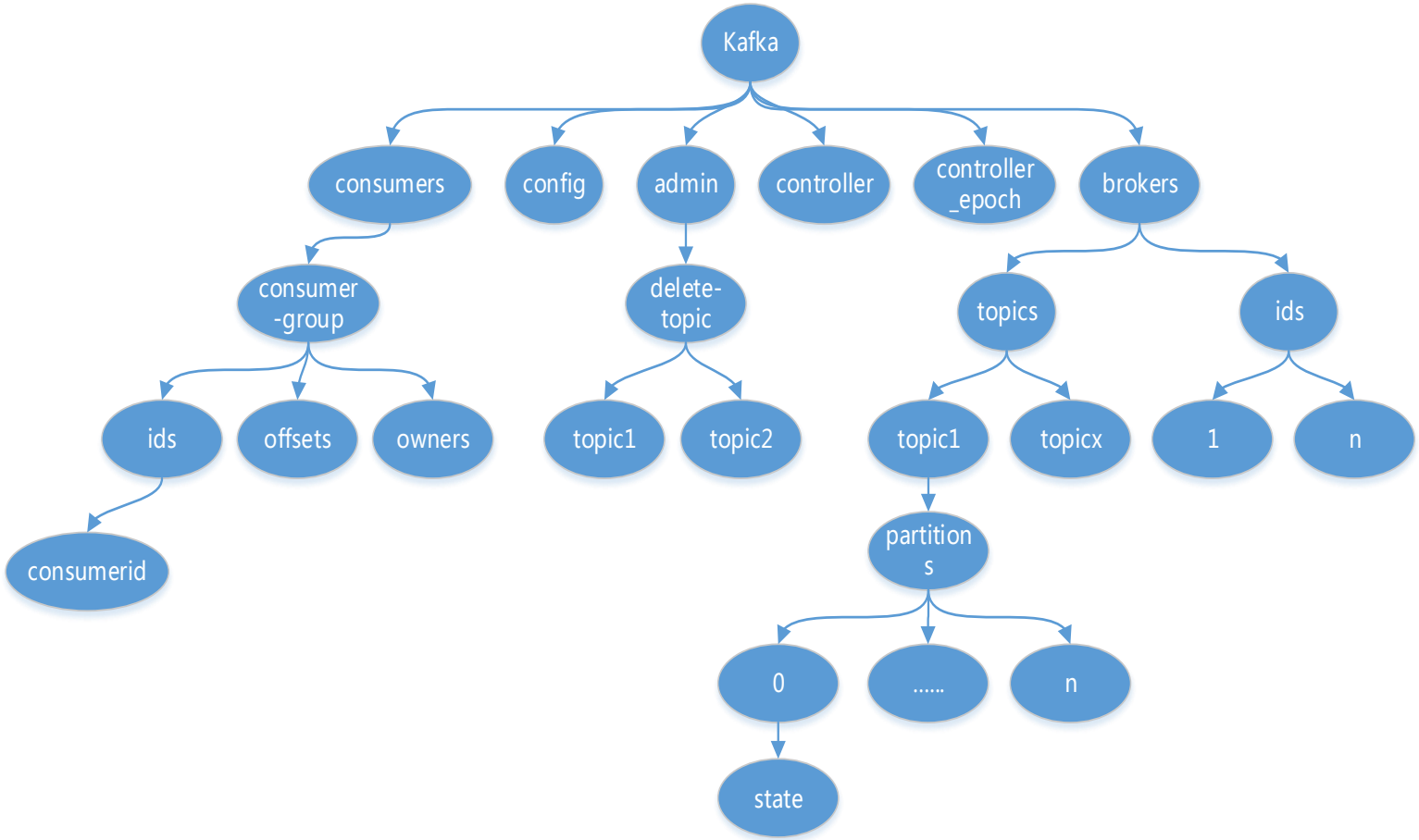
通过zkCli来连接正在运行的ZooKeeper Shell客户端，可以通过ls 和 get命令来获取kafka相关信息

用法:

```
# bin/zkCli.sh -server  zk_host:port/chroot

[zk: 192.168.0.90:24002/kafka(CONNECTED) 1]ls /
[admin, brokers, config, consumers, controller, controller_epoch]

[zk: 192.168.0.90:24002/kafka(CONNECTED) 2] ls /brokers/ids
[26, 27, 28]

[zk: 192.168.0.90:24002/kafka(CONNECTED) 4] get /brokers/ids/26
{"jmx_port":21006,"timestamp":"1434266063915","host":"streaming-90","version":1,"port":21005
}

[zk: 192.168.0.90:24002/kafka(CONNECTED) 5] ls /brokers/topics
[test, test1, test2]

[zk: 192.168.0.90:24002/kafka(CONNECTED) 6] get /brokers/topics/test
{"version":1,"partitions":{"1":[26,27],"0":[28,26]}}
```

火龙果•整理
uml.org.cn

Kafka in ZooKeeper



Kafka Shell

通过kafka-topics.sh来执行topic相关命令

用法：

```
# bin/kafka-topics.sh

Create, delete, describe, or change a topic.
Option                                    Description
------                                    -----------
--alter                                   Alter the configuration for the topic.
--create                                  Create a new topic.
--delete                                  Delete a topic
--describe                                List details for the given topics.
--list                                    List all available topics.
```

Create

创建Topic时，可以选择多个参数，但Topic名称、Zookeeper地址、Partition个数、Replica因子是必须的参数，其它参数还包括replica-assignment 、cleanup.policy等

用法：

```
bin/kafka-topics.sh  kafka-topics.sh --create --topic test1 --partitions 1 --replication-factor 2
                      --zookeeper 192.168.0.90:24002/kafka

Created topic "test1".


bin/kafka-topics.sh  kafka-topics.sh --create --topic test3 --partitions 1 --replication-factor 1
                      --zookeeper 192.168.0.90:24002/kafka --config cleanup.policy=delete

Created topic "test2".
```

Delete

删除Topic时，Topic名称、Zookeeper地址是必须的参数，且在服务端delete.topic.enable设置为true生效

用法：

```
bin/kafka-topics.sh  --delete --topic test1 --zookeeper 192.168.0.90:24002/kafka
Topic test1 is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Alter

更改Topic信息，Topic名称、Zookeeper地址是必须的参数， 其它参数还包括Partition个数等

用法：

```
bin/kafka-topics.sh --alter --topic test3 --zookeeper  192.168.0.90:24002/kafka --config
                     cleanup.policy=compact
Updated config for topic "test3".


bin/kafka-topics.sh --alter --topic test3 --partitions 2 --zookeeper  192.168.0.90:24002/kafka
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of
         the messages will be affected
Adding partitions succeeded!
```

List

显示集群中所有的Topic， Zookeeper地址是必须的参数

用法:

```
bin/kafka-topics.sh  --list --zookeeper 192.168.0.90:24002/kafka
test
test1
test2
test3
```

Describe

显示Topic分区和副本等详细信息，Zookeeper地址是必须的参数，其它参数还包括Topic名称、topics-with-overrides、unavailable-partitions、under-replicated-partitions等

用法：

```
bin/kafka-topics.sh --describe --zookeeper  192.168.0.90:24002/kafka
Topic:test   PartitionCount:2      ReplicationFactor:2   Configs:
      Topic: test    Partition: 0   Leader: 28     Replicas: 28,26       Isr: 28,26
      Topic: test    Partition: 1   Leader: 26     Replicas: 26,27       Isr: 26,27
Topic:test1  PartitionCount:2      ReplicationFactor:2   Configs:
      Topic: test1   Partition: 0   Leader: 27     Replicas: 27,28       Isr: 27,28
      Topic: test1   Partition: 1   Leader: 28     Replicas: 28,26       Isr: 28,26
Topic:test2  PartitionCount:3      ReplicationFactor:2   Configs:
      Topic: test2   Partition: 0   Leader: 26     Replicas: 26,27       Isr: 26,27
      Topic: test2   Partition: 1   Leader: 27     Replicas: 27,28       Isr: 27,28
      Topic: test2   Partition: 2   Leader: 28     Replicas: 28,26       Isr: 28,26
Topic:test3  PartitionCount:1      ReplicationFactor:1   Configs: cleanup.policy=delete
      Topic: test3   Partition: 0   Leader: 27     Replicas: 27   Isr: 27
```

Kafka Shell

通过kafka-reassign-partitions.sh来执行topic partition重分布命令

用法:

```
# bin/kafka-reassign-partitions.sh

This command moves topic partitions between replicas.
Option                                  Description
------                                  -----------
--generate                              Generate a candidate partition
                                        reassignment configuration. Note
                                        that this only generates a candidate
                                        assignment, it does not execute it.
--execute                               Kick off the reassignment as specified
                                        by the --reassignment-json-file
                                        option.
--verify                                Verify if the reassignment completed
                                        as specified by the --reassignment-
                                        json-file option.
```

Generate

通过kafka-reassign-partitions.sh generate命令对指定Topic Partition产生重分布信息

用法:

```
# bin/kafka-reassign-partitions.sh
      --zookeeper <zk_host:port/chroot>
      --topics-to-move-json-file <topics to reassign json file path>
      --broker-list <brokerlist>
      --generate

# bin/kafka-reassign-partitions.sh
      --zookeeper 192.168.0.90:24002/kafka
      --topics-to-move-json-file topics-to-move.json
      --broker-list "26,27,28"
      --generate
Current partition replica assignment

{"version":1,"partitions":[{"topic":"test1","partition":0,"replicas":[33]}]}
Proposed partition reassignment configuration

{"version":1,"partitions":[{"topic":"test1","partition":0,"replicas":[32]}]}
```

Execute

通过kafka-reassign-partitions.sh execute命令执行topic partition重分布

用法:

```
# bin/kafka-reassign-partitions.sh
      --zookeeper <zk_host:port/chroot>
      --reassignment-json-file <manual assignment json file path>
      --execute

# bin/kafka-reassign-partitions.sh
      --zookeeper 192.168.0.90:24002/kafka
      --reassignment-json-file expand-cluster-reassignment.json
      --execute

Current partition replica assignment

{"version":1,"partitions":[{"topic":"test1","partition":0,"replicas":[33]}]}

Save this to use as the --reassignment-json-file option during rollback
Successfully started reassignment of partitions
{"version":1,"partitions":[{"topic":"test1","partition":0,"replicas":[32]}]}
```

Verify

通过kafka-reassign-partitions.sh verify命令确认topic partition重分布是否完成

用法:

```
# bin/kafka-reassign-partitions.sh
      --zookeeper <zk_host:port/chroot>
      --reassignment-json-file <manual assignment json file path>
      --verify

# bin/kafka-reassign-partitions.sh
      --zookeeper 192.168.0.90:24002/kafka
      --reassignment-json-file expand-cluster-reassignment.json
      --verify


Status of partition reassignment:
Reassignment of partition [test1,0] completed successfully
```

火龙果•整理
uml.org.cn

Kafka Producer通用配置

配置	描述	设置方法
metadata.broker.list	获取元数据的broker列表	在Producer程序中设置
producer.type	同步或异步发送	在Producer程序中设置
request.required.acks	Procuder收到响应格式表示请求成功。	在Producer程序中设置
batch.num.mesage	在异步模式下，批量发送消息大小	在Producer程序中设置
queue.buffering.max.ms	在异步模式下，最大缓存时间	在Producer程序中设置

火龙果•整理
uml.org.cn

Kafka Consumer通用配置

配置	描述	设置方法
group.id	Consumer group标识	在Consumer程序中设置
zookeeper.connect	Zookeeper连接信息	在Consumer程序中设置

火龙果•整理
uml.org.cn

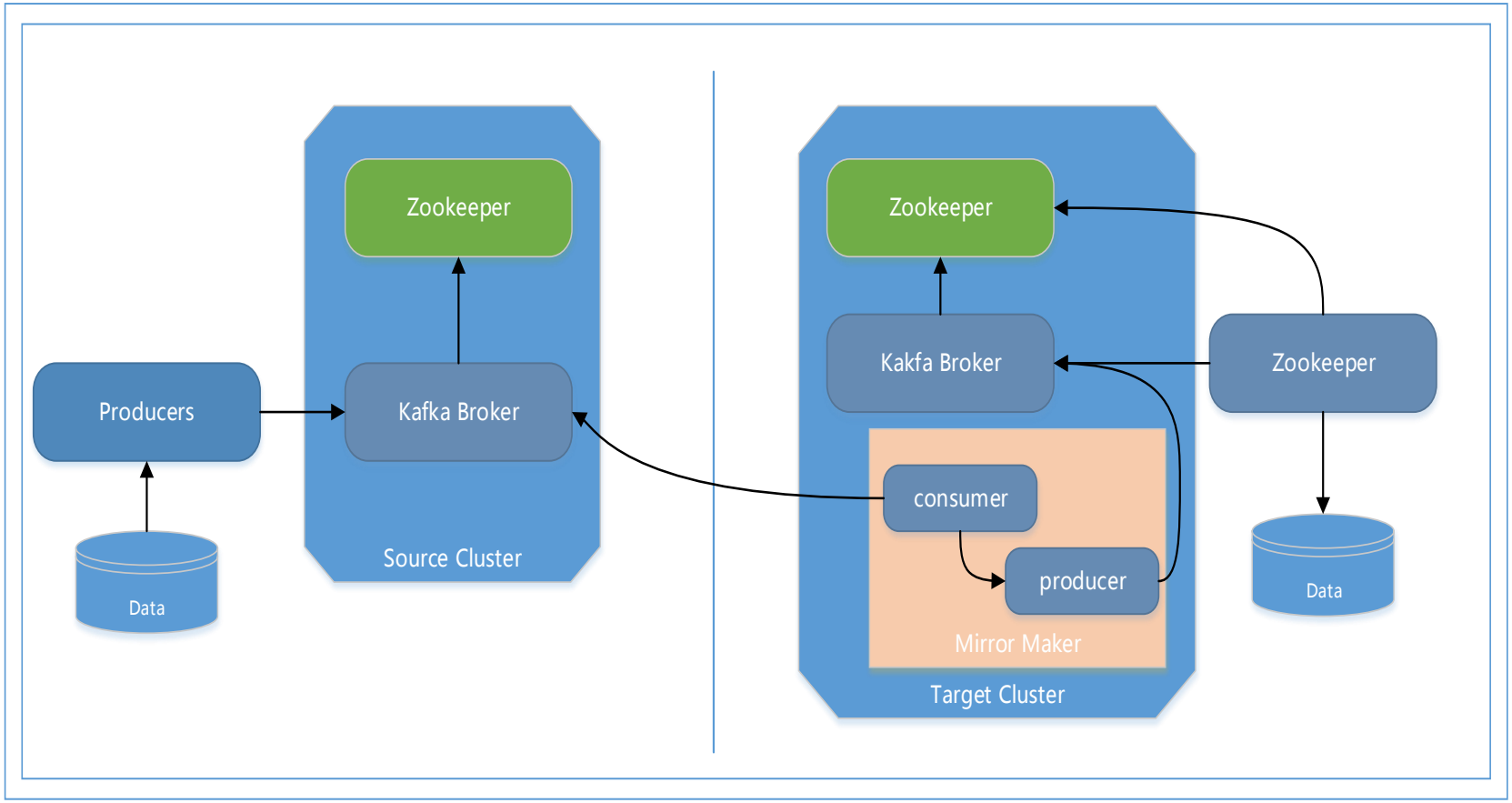
Kafka Server通用配置

配置	描述	设置方法
broker.id	Kafka broker唯一编号	FI自动生成
log.dirs	Kafka 数据存储目录	在FI客户端界面设置
port	端口号	在FI客户端界面设置
KAFKA_HEAP_OPTS	Kafka broker 内存参数。	在FI客户端界面设置

火龙果•整理
uml.org.cn

目录

1. Kafka应用场景
2. Kafka架构与功能
3. Kafka关键流程
4. Kafka常用命令和参数
5. Kafka高级专题

火龙果•整理
uml.org.cn

Kafka Cluster Mirroring