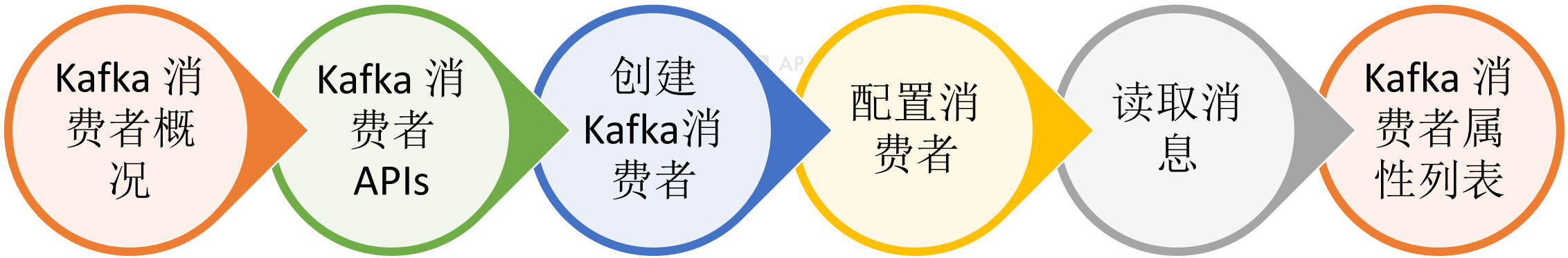
NIIT | training-china.com

利用Kafka实现分布式消息传递系统

Kafka 消费者

本章目标

本章你将学到：


Kafka 消费者概况
Kafka 消费者 APIs
创建 Kafka消费者
配置消费者
读取消息
Kafka 消费者属性列表

本章目标

上一章，你学了：

- Kafka生产者概况
- Java生产者API
- 创建Kafka生产者
- 配制生产者

本章，你将学到：

- Kafka 消费者概况
- Kafka 消费者APIs
- 创建Kafka消费者
- 配置消费者
- 读取消息
- Kafka 消费者属性列表

Kafka消费者简介

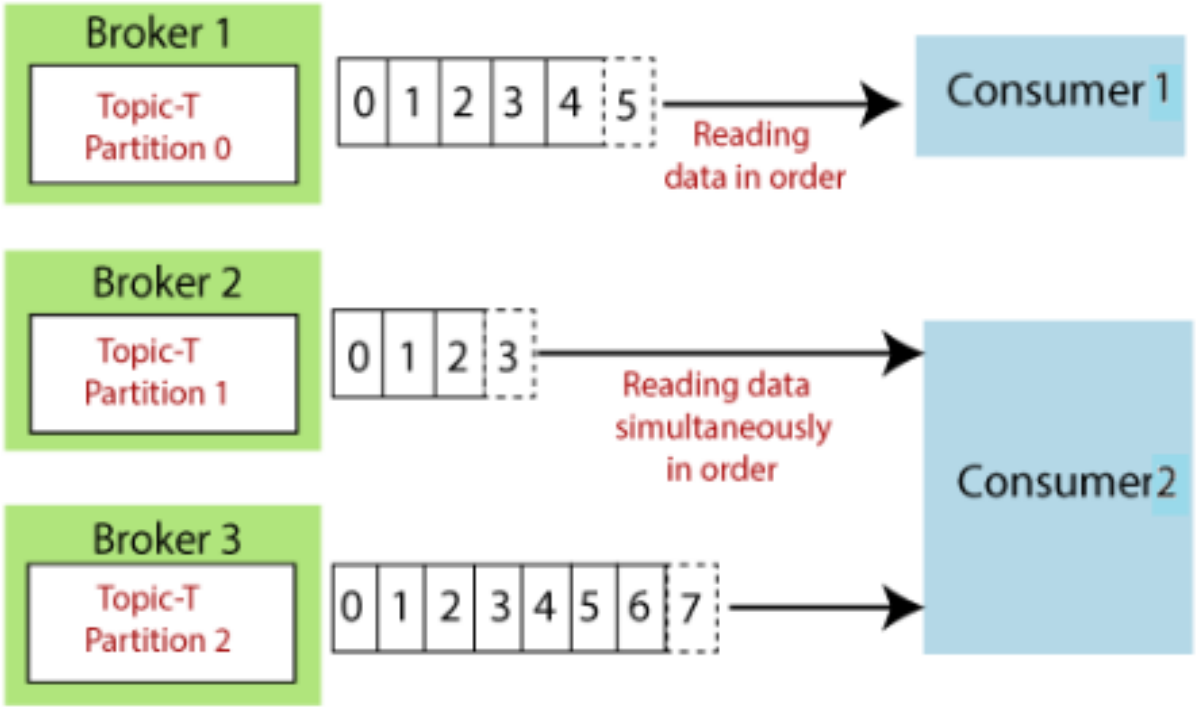
- Kafka Consumer，即消费者可以订阅生产者生产的消息主题，获取并处理从订阅主题中提取的数据（消费）。

- 消费者的使用场景
  - 进行实时或接近实时分析的应用程序，
  - 使用NoSQL数据存储的应用程序
  - 大数据数据仓库解决方案
  - 后端服务的业务处理
  - Hadoop系统中的消费者
  - 其他基于订阅者（ subscriber-based ）的解决方案。

- 这些消费者也可以用Java，C和Python等语言实现（客户端支持多语言）。

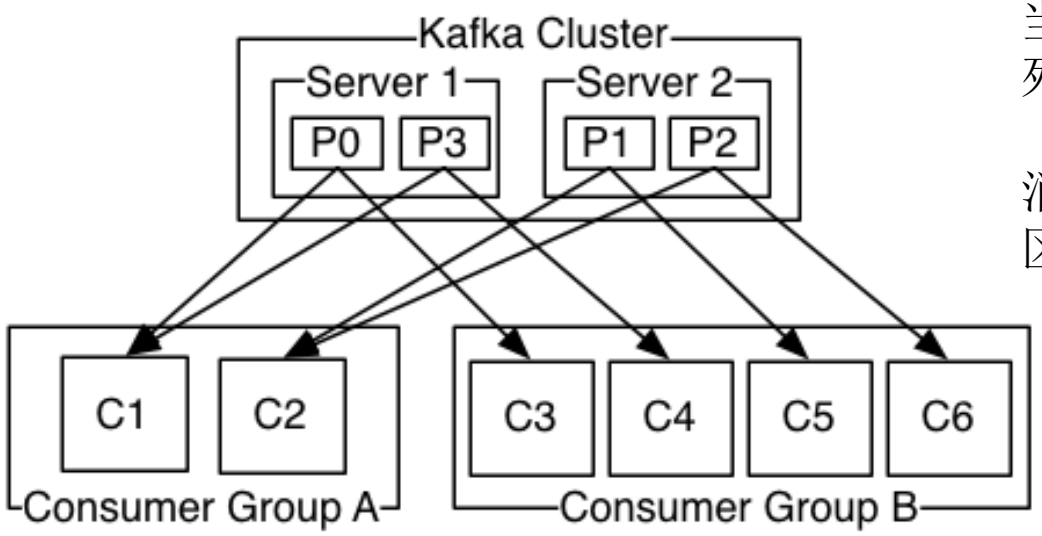
消费者

- 消费者在主题的帮助下从Kafka集群中消费或读取数据。
- 消费者知道应该由哪个代理服务器（broker）读取数据。
- 消费者有序读取每个分区中的数据。
- 消费者可以同时（simultaneous）读取来自多个Kafka代理服务器的数据。
- 消费者记得他们停止读取的地方的偏移量。



消费者组 Consumer Group

- 消费者组可以看做逻辑上的单一消费者，实际是多个拥有相同group.id的消费者，他们都订阅了相同的主题。
- 消费者将所有被订阅主题上的分区分配给同一消费者组中的物理消费者，每个分区只能被同一个group中的一个消费者所消费（反过来同一个group中的单个消费者可以被分配到订阅主题下的多个分区）。
- 属于同一组的单个消费者可以同时读取多个分区的数据，而这些分区可以分布在集群中的不同主机上（如图所示）。



当消费者组中只有一个消费者的时候，就是消息队列模型，不然就是发布-订阅模型，并且易于伸缩。

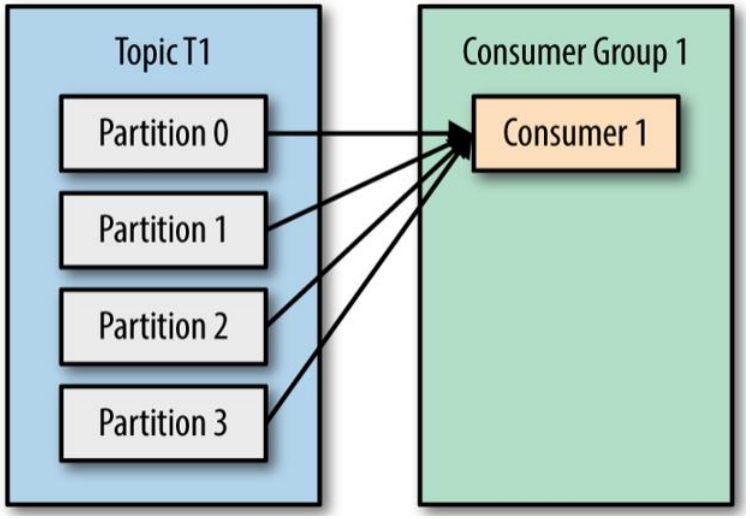
消费者组内消费者数应小于或等于分区数，topic分区数最好刚好是消费者组内成员数的倍数。

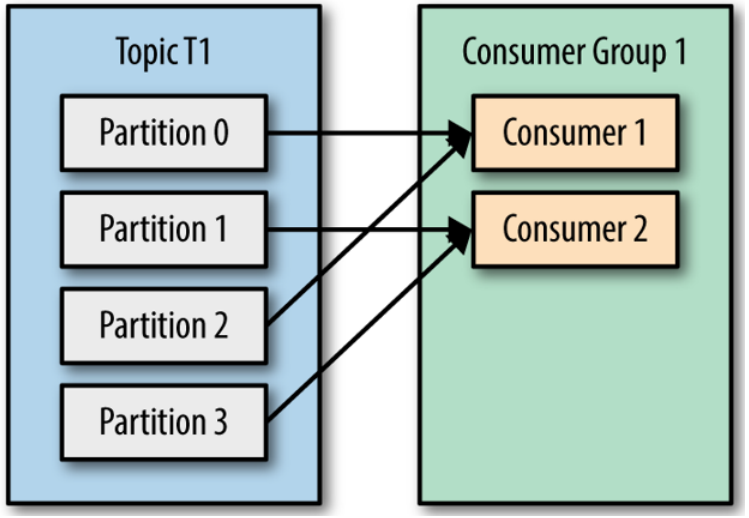
消费者组

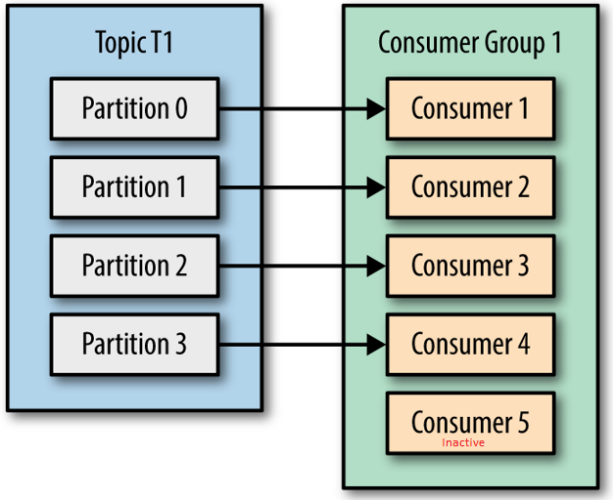
- 消费者组通过其group.id属性进行标识
- 连接到同一Kafka集群和使用相同group.id的所有消费者组成一个消费者组。
- 消费者可以随时添加组或离开组。 这两种情况都会触发重新平衡（Rebalance），重平衡就是对分区重新分配给消费者组，以确保每个分区只有组内的一个消费者消费。
- 一个单一的Kafka消费者可以看作一个只有一个成员的消费者组。
- 同样的记录只能被消费者组中的一个消费者消费。
- 消费者组中的消费者以负载均衡的方式消费主题中的数据。

消费者组

- 消费者组主要解决了单个消费者的消费速度跟不上生产者生产消息的速度的问题。
  - 一个消费者C1从主题T1的所有分区中读取
  - 两个消费者阅读主题T1的特定分区
  - 如果添加更多消费者，则它们将处于非活动状态

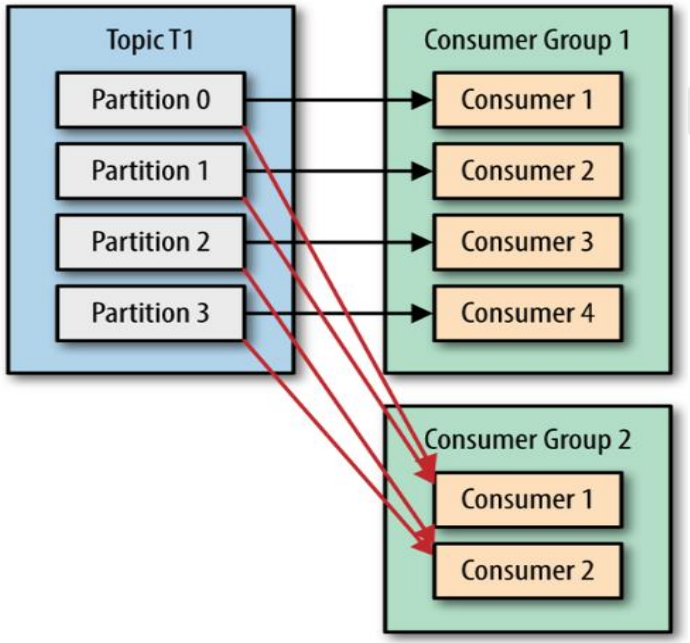






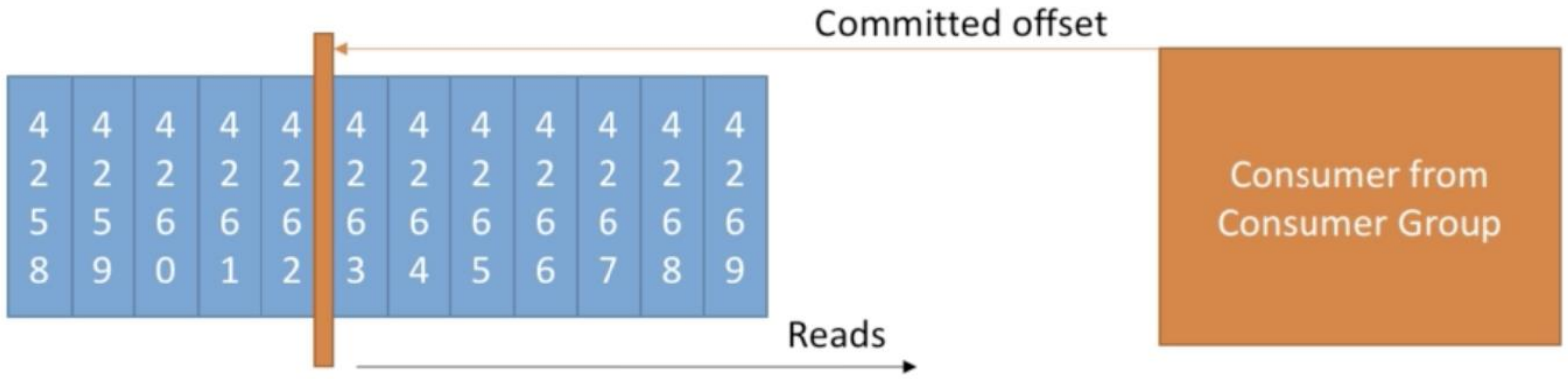
消费者组

- 很多时候可能有多个应用程序都需要消费同一个主题的数据。
- 这种情况下，我们希望每个应用程序都获取所有消息，而不仅仅是一个子集。
- 为确保每个应用程序都能及时处理主题中的所有消息，可以为应用程序分别创建自己的消费者组。
- Kafka可以添加大量的消费者和消费者组而不会降低性能。



消费者偏移量

- Kafka存储了消费者正在读取的偏移量。
- 在名为__consumer_offsets 的Kafka主题中可以找到实时提交的偏移量。
- 消费者组中的消费者处理了从Kafka接收的数据后，应该提交偏移量。
- 这样如果消费者挂掉，下次它将能从中断处而不是开始处进行读取。



交付语义

- 提交偏移量就像读者在阅读书籍或小说时使用的书签。 在Kafka中，使用了以下三种交付语义（delivery semantics）：
  - 最多一次（AMOS: At Most Once Semantic）：偏移量在消费者收到消息后立即提交。
  - 至少一次（ALOS: At Least Once Semantic）：在处理完消息后提交偏移量。
  - 恰好一次（EOS：Exactly Once Semantic）：使用Kafka Streams API可以实现从Kafka到Kafka工作流程的偏移量。

At most once	At least once	Exactly once
Message pulled once	Message pulled one or more times; processed each time	Message pulled one or more times; processed once
May or may not be received	Receipt guaranteed	Receipt guaranteed
No duplicates	Likely duplicates	No duplicates
Possible missing data	No missing data	No missing data

小测验

In the listed delivery semantic, which one is likely to lose data and never delivery again?

a. At most once

b. At least once

c. Exactly once

d. All are correct above items

Which programming language does Kafka consumer API support?

a. Java

b. C

c. Python

d. All are correct above items

Which option can be used to set the ID of Kafka consumer group?

a. consumergroup.id

b. groupid

c. group.id

d. consumergroupid

消费者组和分区重新平衡（Rebalance）

- 在消费者组中，Kafka会因为一些事件（分区数变动，消费者数变动，订阅主题变动）触发重平衡，重平衡会重新分配每个消费者具体消费哪个分区。
- 这个在消费者之间更改分区所有权的过程称为重新平衡。
- 重新平衡很重要，因为它们为消费者组提供了高可用性和可伸缩性，但是在正常情况下，应该尽量避免触发不必要的重平衡操作。

- 因为重平衡过程中，消费者无法从**kafka**消费消息，这对**kafka**的吞吐量（**TPS**）影响极大，而如果**kafka**集群内节点较多，比如数百个，那重平衡可能会耗时极多。数分钟到数小时都有可能，而这段时间**kafka**基本处于不可用状态。所以在实际环境中，应该尽量避免重平衡发生。

消费者群体和分区重新平衡

- 重新平衡发生在以下事件：：
  - 扩大消费者组规模（ Scaling Up ）：将新消费者添加到消费者组。 新消费者开始使用先前由另一个消费者使用的分区中的消息。
  - 缩减消费者组规模（ Scaling Down ）：将消费者从消费者组中删除。 另一个消费者开始使用已删除分区中的消息。
  - 主题增加分区
  - 应用程序崩溃
  - 消费者客户端关闭：存活的消费者开始使用先前由另一个消费者使用的分区中的消息。
  - 由于消费者客户端崩溃导致组协调器（ group coordinator）将某位消费者视为DEAD时，也可能发生在消费者忙于长时间运行处理时。意味着在此期间，消费者没有在配置的会话间隔内向组协调器发送任何心跳信号。
  - 有消费者订阅任何主题。
  - 如果您订阅尚未创建的主题，则创建主题后将触发重新平衡。 如果您订阅的主题被删除，也会触发重发重平衡。

Java 消费者API – 高级 API

Java Consumer API - High Level API

- 每个主题中每个分区的偏移量管理
- Kafka服务器故障转移（failover，broker list添加多个broker主机地址）。
- 分区和消费者改变导致的负载重平衡（load balancing）
- 整洁统一的API：高级消费者结合了旧版本（Low Level）消费者客户端的“Simple”和原本新版本（High Level）消费者客户端的功能
- 减少依赖：新消费者使用纯Java编写（旧消费者使用Scala语言编写）。
- 安全性更好

Java 消费者 API – 高级 API

KafkaConsumer

- Kafka消费者客户端，用来消费Kafka集群中的消息。

ConsumerRecord

- 从Kafka接收的消息的键/值对。 还包括主题名称，接收记录的分区号，所在Kafka分区中记录的偏移量，以及由相应生产者标记的消息的时间戳。

ConsumerRecords

- 它是消费者记录的容器。 为了保留特定主题的每个分区的消费者记录列表，我们使用此API。

Java 消费者 API – 高级 API

NIIT | training.com china

KafkaConfig

- 可帮助设置配置属性。

ConsumerConnector

- 负责消费者与ZooKeeper交互。

KafkaStreams

- 通过KafkaStreams API，Kafka客户端可以对来自一个或多个主题的输入执行连续计算，并将输出发送到零个，一个或多个主题。

TopicPartition

- 这有助于从特定主题和分区中检索消息。

Java 消费者 API – 低级 API

Java Consumer API - Low Level API，不常用或仅在调试时使用（在较新的版本中已取消）

实现了：

- 更好地控制Kafka 消息的使用，例如：
  - 多次读取相同的消息
  - 只消费主题的部分分区
  - 手动管理事务确保消息只处理一次且仅处理一次（(Exactly Once primitive)
- 更灵活的控制，但以复杂性为代价：
  - 偏移量不再透明
  - 缓存代理自动故障转移需要手动处理
  - 添加消费者，分区和Broker之后需要您自己进行负载平衡。

Java 消费者 API – 低级 API

- 简单消费者（kafka.javaapi.consumer.SimpleConsumer）有助于连接到lead broker，进而获取主题中的消息，并提供了一些获取主题元数据和偏移量列表的方法。
- 发起拉取请求 (kafka.api.FetchRequest),
- 偏移量请求 (kafka.javaapi.OffsetRequest),
- 偏移量拉取请求 (kafka.javaapi.OffsetFetchRequest),
- 偏移量提交请求 (kafka.javaapi.OffsetCommitRequest),
- 主题元数据请求 (kafka.javaapi.TopicMetadataRequest).

Kafka 消费者

1. Kafka 消费者的工作原理?
2. 解释交付语义?

使用高级API创建 Kafka 消费者

要创建Kafka消费者，您需要先设置属性。

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>( props );
```

订阅主题：

1. 单一主题, `consumer.subscribe(Collections.singletonList("topicName"));` //不可变单值列表
2. 多个主题 `consumer.subscribe(Arrays.asList("topicNames"));` // 固定长度的列表，可为多值
3. 使用正则表达式: `consumer.subscribe(Pattern.compile"regexExp");` // Pattern.compile("my-topic-*")

Poll轮询处理协调、分区重新平衡、heartbeats和数据提取的所有详细信息,

```
while (true) {
  ConsumerRecords<String, String> records = consumer.poll(100); // 超时时间
  for (ConsumerRecord<String, String> record : records) {
    System.out.printf("offset = %d, key = %s, value = %s\n",
    record.offset(), record.key(), record.value());
  }
}
```

配置 Kafka 消费者

步骤

- 创建消费者属性
- 创建消费者

代码

```
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
  KAFKA_SERVER_URL + ":" + KAFKA_SERVER_PORT);
props.put(ConsumerConfig.GROUP_ID_CONFIG, CLIENT_ID);
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
  "org.apache.kafka.common.serialization.IntegerDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
  "org.apache.kafka.common.serialization.StringDeserializer);

KafkaConsumer<Integer,String> consumer
  = new KafkaConsumer<>(props);
```

读消息

步骤

- 为消费者订阅特定主题.

```
consumer.subscribe(Collections.singletonList(this.topic));
```

- 循环拉取数据

```
ConsumerRecords<Integer,String> records = consumer.poll(100);
```

- 遍历消费

```
for (ConsumerRecord<Integer, String>record : records)
{    System.out.println("Receivedmessage:
            (" + record.key() + ",    + record.value() + ")
     at offset   " + record.offset());
}
```

Creating Kafka Consumer

Kafka consumer properties list	
Field	**Description**
AUTO_COMMIT_INTERVAL_MS_CONFIG	auto.commit.interval.ms
AUTO _OFFSET_RESET_CONFIG	auto.offset.reset
BOOTSTRAP_SERVERS_CONFIG	bootstrap.servers
CONNECTIONS_MAX_IDLE_MS_CONFIG	connections.max.idle.ms
DEFAULT_API_TIMEOUT_MS_CONFIG	default.api.timeout.ms
ENABLE_AUTO_COMMIT_CONFIG	enable.auto.commit
GROUP_ID_CONFIG	group.id
HEARTBEAT_INTERVAL_MS_CONFIG	heartbeat.interval.ms
ISOLATION_LEVEL_CONFIG	isolation.level
KEY_DESERIALIZER_CLASS_CONFIG	key.deserializer
MAX_POLL_INTERVAL_MS_CONFIG	max.poll.interval.ms
MAX_POLL_RECORDS_CONFIG	max.poll.records
RECEIVE_BUFFER_CONFIG	receive.buffer.bytes

Kafka 消费者

提到kafka消费者的重要属性有?

小测验

消费者如何使用Subscribe (...)方法订阅主题？

a. 单个主题
b. 多个主题
c. 正则表达式
d. 以上全部

活动

活动 4.1

创建 Java 消费者

活动4.2

创建定制序列化器

请参考实验指南进行此项活动

关键词

第四章主要词汇		
English	Chinese	Pinyin
Offset	偏移量	Piān yí liàng
Consumer	消费者	Xiāo fèi zhě
Subscribe	订阅者	Dìng yuè zhě
MetaData	元数据	Yuán shù jù
Streams	流	Líu
Rebalance	重新平衡	Chóng xīn píng héng
Delivery	传递	Chuán dì
Semantics	语义	Yǔyì

本章总结

第四章		
交付物	状态	备注
Kafka消费者概况	已做	
Kafka 消费者APIs	已做	
创建Kafka消费者		
配置消费者		
读取消息		
Kafka消费者属性列表		

摘要

- 消费者是消费Kafka生产者发布的消息并处理从他们那里提取的数据的应用程序。
- 消费者组可以描述为订阅一组指向应用程序主题的单个逻辑消费者。 它可以在两种情况下使用。
- Kafka存储消费者群体一直在读取的偏移量。__consumer_offsets
- 交付语义：最多一次，至少一次，恰好一次
- 高级API：Kafka 消费者 <K，V>，消费者记录，消费者配置，消费者连接器，Kafka流，主题分区

摘要

- 低级API：更灵活的控制，但以复杂性为代价
- The subscribe () 方法将主题列表作为参数进行订阅：
- The subscribe () 方法将主题列表作为参数进行订阅：
- Poll轮询处理协调、分区重新平衡、heartbeats和数据提取的所有详细信息，

下一章，
你将学到

管理和监控Kafka

- 主题运营
- 消费者运营
- 动态配置更改
- 监控Kafka