

# Kafka

- [相關連結](#)
- [名詞解釋](#)
- [Docker建立Kafka cluster](#)
- [Kafka 架構](#)
- [Kafka 學習筆記](#)
- [producer\\_consumer範例](#)
- [kafka connect](#)

# 相關連結

## 【尚硅谷】2022版Kafka3.x教程（从入门到优秀，深入全面）

- <https://www.youtube.com/playlist?list=PLmOn9nNkQxJHTVxt3wxWXYheQPLlh-9T6>
- [https://s8jl-my.sharepoint.com/:f/g/personal/atguigu\\_s8jl\\_onmicrosoft\\_com/EoZ8qIFvkKpHr8Sq1YITz\\_ABBxlt2ZHjD1ldArUVOqf4e=P2UAe7](https://s8jl-my.sharepoint.com/:f/g/personal/atguigu_s8jl_onmicrosoft_com/EoZ8qIFvkKpHr8Sq1YITz_ABBxlt2ZHjD1ldArUVOqf4e=P2UAe7)

## 【尚硅谷】SpringBoot3零基础教程

笔记： <https://www.yuque.com/leifengyang/springboot3>

代码： <https://gitee.com/leifengyang/spring-boot-3>

- 076、SpringBoot3 消息服务 Kafka简介与使用  
<https://www.youtube.com/watch?v=hgHFZcSogow&list=PLmOn9nNkQxJEeIH75s5pdTUnCo9-xOc7c&index=76>
- 077、SpringBoot3 消息服务 使用KafkaTemplate发送消息  
<https://www.youtube.com/watch?v=YG-ezDlxFEI&list=PLmOn9nNkQxJEeIH75s5pdTUnCo9-xOc7c&index=77>
- 078、SpringBoot3 消息服务 使用KafkaListener监听消息  
<https://www.youtube.com/watch?v=Haxp9th26y0&list=PLmOn9nNkQxJEeIH75s5pdTUnCo9-xOc7c&index=78>
- 079、SpringBoot3 消息服务 小结  
<https://www.youtube.com/watch?v=njlPA3VBbQ4&list=PLmOn9nNkQxJEeIH75s5pdTUnCo9-xOc7c&index=79>

## Confluent platform

- <https://docs.confluent.io/platform/current/platform-quickstart.html>
- <https://github.com/confluentinc/cp-all-in-one/blob/7.5.1-post/cp-all-in-one-community/docker-compose.yml>

# 名詞解釋

## Kafka Producer & Kafka Consumer

Kafka 是一個分散式消息系統，它允許應用程序之間進行高效、可擴展的異步通信。在 Kafka 中，有兩個主要的元件，分別是生產者（Producer）和消費者（Consumer）。

### 1. Kafka 生產者（Producer）：

- 生產者是負責將數據消息發送到 Kafka 集群的應用程序或服務。
- 生產者將消息發布到 Kafka 主題（Topic），主題是消息的分類或主題。
- 生產者可以將消息發布到一個或多個主題，並可以指定消息的密鑰（Key）和分區（Partition）。
- 生產者將消息寫入 Kafka 集群後，它們不再關心消息的後續處理。生產者可以持續發送消息，而不必等待消費者處理。

### 2. Kafka 消費者（Consumer）：

- 消費者是負責從 Kafka 主題中讀取消息的應用程序或服務。
- 消費者可以訂閱一個或多個主題，以接收相關的消息。
- 消費者通常以消費者組（Consumer Group）的形式運行，每個消費者組可以有多个消費者實例，這些實例共同消費主題上的消息。
- Kafka 確保每條消息只被消費者組中的一個消費者實例處理，以實現消息的平衡分發。
- 消費者從指定的分區讀取消息，並可以在讀取消息後進行處理，例如數據處理、存儲或其他操作。

總結來說，Kafka 生產者用於將消息寫入 Kafka 主題，而 Kafka 消費者用於從主題中讀取消息，進行處理或儲存等操作。消費者通常以消費者組運行，以實現消息處理的分佈式和冗餘性，以確保高可用性和可擴展性。這使 Kafka 成為一個非常強大的消息系統，適用於大規模、高流量的數據流處理應用。

## Kafka Broker

Kafka Broker 是 Apache Kafka 集群中的核心元件之一，它是負責接收、存儲和分發消息的伺服器。以下是有關 Kafka Broker 的簡介：

1. 伺服器角色：Kafka Broker 是 Kafka 集群中運行 Kafka 伺服器軟體的實例。一個 Kafka 集群通常包含多個 Broker，它們共同協作以實現消息的分發和冗餘。
2. 消息存儲：Kafka Broker 負責存儲發送到 Kafka 集群的消息。這些消息存儲在 Broker 上的主題（Topics）中，每個主題都可以有多個分區（Partitions）。消息被持久性地保存，並根據配置的保留策略保留一段時間。
3. 分發消息：Kafka Broker 通過提供發送消息的接口來允許 Kafka 生產者將消息發佈到特定主題。同時，它也提供訂閱消息的接口，以供 Kafka 消費者讀取和處理消息。Broker 負責確保消息從生產者到消費者之間的有效分發。
4. 高可用性：Kafka 集群通常包含多個 Broker，這樣即使其中一個 Broker 發生故障，系統仍然可用。Kafka 通過分區（Partitions）的複製和領袖（Leader）和追隨者（Follower）的概念實現高可用性，確保即使在 Broker 失故障情況下，數據仍然可用。
5. 資料保留：Kafka Broker 配置了消息的保留策略，該策略確定消息存儲的時間長度。這允許根據需求保存數據，以滿足不同應用程序和法規的要求。

總之，Kafka Broker 是 Kafka 集群的核心組件，負責消息的接收、存儲和分發。它確保了高可用性、可擴展性和持久性，使 Kafka 成為一個流行的選擇，用於處理實時數據流、日誌記錄、事件驅動架構等各種應用。

## ZooKeeper

Apache ZooKeeper（通常簡稱ZooKeeper）是一個開源的分散式協調服務，用於協助分散式應用程序協調和管理數據。ZooKeeper 的主要目標是提供一個高可用性的環境，用於解決分散式系統中的一些共同問題，例如配置管理、分布式鎖、協調和分發信息等。以下是有關ZooKeeper的關鍵概念和功能的簡要說明：

1. **分布式系統協調**：ZooKeeper旨在為分散式應用程序提供一個共享的協調和控制基礎設施。它可以幫助不同部分的應用程序在分散式環境中協同工作。
2. **分布式配置管理**：ZooKeeper可以用來存儲和管理分散式系統的配置信息，例如主機名、端口、連接字符串等。這使得系統的配置變更變得容易，並且能夠在運行時進行動態更新。
3. **分布式鎖**：ZooKeeper提供了一種機制來實現分布式鎖，以協調多個進程之間的操作順序，防止競爭條件，確保操作的原子性。
4. **分布式協議**：ZooKeeper可以用於實現分布式協議，確保多個節點之間的相對順序，以協調事件的發生。
5. **數據一致性**：ZooKeeper保證了分散式環境中的數據一致性，這對於需要強一致性的應用程序非常重要。
6. **高可用性**：ZooKeeper自身的架構具有高可用性，並且能夠容忍部分節點的故障。它使用選主（Leader）和追隨者（Follower）的模型，確保即使在部分節點故障的情況下，系統仍能正常運行。
7. **輕量和快速**：ZooKeeper是一個輕量級的服務，並且可以快速執行。它設計用於高吞吐量和低延遲的操作。
8. **開發者友好**：ZooKeeper提供了多種編程語言的客戶端庫，使開發者可以方便地與ZooKeeper集成。

ZooKeeper通常用於協調分散式系統的操作，例如Apache Kafka、Apache HBase、分布式數據庫等。它提供了一個可靠的基礎設

施，以解決分散式系統中的一些關鍵問題，並確保各種應用程序能夠協同工作並達到高可用性和一致性。

---

## Kafka Connect

Kafka Connect是用來連接Kafka和外部資料系統的工具，它提供了資料匯入和匯出的能力。以下是Kafka Connect的主要功能：

1. **資料匯入：** Kafka Connect 可用於從外部資料來源（如資料庫、日誌檔案、訊息佇列等）將資料匯入到 Kafka 主題。它提供了各種連接器，方便輕鬆將不同資料來源的資料傳輸到 Kafka。
2. **資料匯出：** Kafka Connect 也可用於將 Kafka 主題中的資料匯出至外部資料系統。這使得您可以將 Kafka 作為資料整合的中間層，將資料傳遞到多個目的地。
3. **連接器管理：** Kafka Connect提供了可設定的連接器，用於定義資料來源和目標之間的資料流。這些連接器可以配置、管理和監視，以確保資料的可靠傳輸。
4. **多元化和可擴展性：** Kafka Connect是多元化的，可以在多個工作節點上運行，以實現高吞吐量和可擴展性。

在Kafka結構中，SchemaRegistry和KafkaConnect通常一起使用，以確保資料的有效傳輸系統和一致性。KafkaConnect用於連接不同的資料來源和目標，而SchemaRegistry用於管理資料模式，以確保資料的一致這兩個元件協同工作，有助於建立強大的資料流處理應用程式。

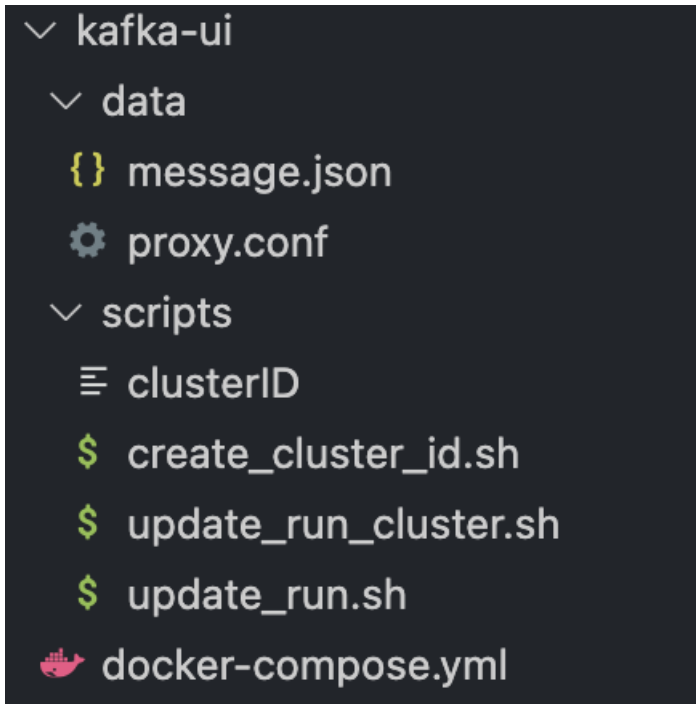
---

## Schema Registry

SchemaRegistry是Kafka生態系中的關鍵元件，用於管理Avro、JSONSchema等訊息序列化反序列化的模式。以下是SchemaRegistry的主要功能：

1. **模式管理：** SchemaRegistry用於儲存和管理不同訊息主題的資料模式。它允許Kafka生產者和消費者使用統一的資料模式，確保資料的一致性和可互通性。
  2. **模式註冊：** 生產者在將訊息傳送到Kafka時，將訊息的資料模式註冊到Schema註冊表。這使得消費者可以取得訊息的模式，以便正確解析和反序列化資料。
  3. **模式演進：** 當資料模式需要變更時，模式註冊表允許進行模式演進，以確保新的和舊的訊息都能夠被處理。這有助於系統的升級和維護。
  4. **資料驗證：** SchemaRegistry可以驗證傳送到Kafka主題的訊息是否與註冊的資料模式相容，從而保證資料的一致性和一致性。
-

# Docker建立Kafka cluster



docker-compose.yml

10.20.30.40 -> 改成自己的ip

```
---
version: '2'
services:
  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - 8081:8080
    depends_on:
      - kafka0
      - kafka1
      - kafka2
      - schema-registry0
      - kafka-connect0
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka0:29092,kafka1:29092,kafka2:29092
      KAFKA_CLUSTERS_0_METRICS_PORT: 9997
      KAFKA_CLUSTERS_0_SCHEMAREGISTRY: http://schema-registry0:8085
      KAFKA_CLUSTERS_0_KAFKACONNECT_0_NAME: first
      KAFKA_CLUSTERS_0_KAFKACONNECT_0_ADDRESS: http://kafka-connect0:8083

  kafka0:
    image: confluentinc/cp-kafka:7.2.1
    hostname: kafka0
    container_name: kafka0
    ports:
      - 9092:9092
      - 9997:9997
    environment:
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka0:29092,PLAINTEXT_HOST://10.20.30.40:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
```

```

KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
KAFKA_PROCESS_ROLES: 'broker,controller'
KAFKA_CLUSTER_ID:
KAFKA_NODE_ID: 1
KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka0:29093,2@kafka1:29093,3@kafka2:29093'
KAFKA_LISTENERS: 'PLAINTEXT://kafka0:29092,CONTROLLER://kafka0:29093,PLAINTEXT_HOST://0.0.0.0:9092'
KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
KAFKA_JMX_PORT: 9997
KAFKA_JMX_OPTS: -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -Djava.rmi.server.hostname=kafka0 -Dcom.sun.management.jmxremote.rmi.port=9997
volumes:
- ./scripts/update_run_cluster.sh:/tmp/update_run.sh
- ./scripts/clusterID:/tmp/clusterID
- ./data/kafka0/kraft-combined-logs:/tmp/kraft-combined-logs
command: "bash -c 'if [ ! -f /tmp/update_run.sh ]; then echo \"ERROR: Did you forget the update_run.sh file that came with this
docker-compose.yml file?\" && exit 1 ; else /tmp/update_run.sh && /etc/confluent/docker/run ; fi'"

```

```

kafka1:
image: confluentinc/cp-kafka:7.2.1
hostname: kafka1
container_name: kafka1
ports:
- 9093:9092
environment:
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka1:29092,PLAINTEXT_HOST://10.20.30.40:9093
KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
KAFKA_PROCESS_ROLES: 'broker,controller'
KAFKA_NODE_ID: 2
KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka0:29093,2@kafka1:29093,3@kafka2:29093'
KAFKA_LISTENERS: 'PLAINTEXT://kafka1:29092,CONTROLLER://kafka1:29093,PLAINTEXT_HOST://0.0.0.0:9092'
KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
KAFKA_JMX_PORT: 9997
KAFKA_JMX_OPTS: -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -Djava.rmi.server.hostname=kafka1 -Dcom.sun.management.jmxremote.rmi.port=9997
volumes:
- ./scripts/update_run_cluster.sh:/tmp/update_run.sh
- ./scripts/clusterID:/tmp/clusterID
- ./data/kafka1/kraft-combined-logs:/tmp/kraft-combined-logs
command: "bash -c 'if [ ! -f /tmp/update_run.sh ]; then echo \"ERROR: Did you forget the update_run.sh file that came with this
docker-compose.yml file?\" && exit 1 ; else /tmp/update_run.sh && /etc/confluent/docker/run ; fi'"

```

```

kafka2:
image: confluentinc/cp-kafka:7.2.1
hostname: kafka2
container_name: kafka2
ports:
- 9094:9092
environment:
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka2:29092,PLAINTEXT_HOST://10.20.30.40:9094
KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
KAFKA_PROCESS_ROLES: 'broker,controller'
KAFKA_NODE_ID: 3
KAFKA_CONTROLLER_QUORUM_VOTERS: '1@kafka0:29093,2@kafka1:29093,3@kafka2:29093'
KAFKA_LISTENERS: 'PLAINTEXT://kafka2:29092,CONTROLLER://kafka2:29093,PLAINTEXT_HOST://0.0.0.0:9092'
KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
KAFKA_JMX_PORT: 9997

```

```
KAFKA_JMX_OPTS: -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -Djava.rmi.server.hostname=kafka1 -Dcom.sun.management.jmxremote.rmi.port=9997
volumes:
- ./scripts/update_run_cluster.sh:/tmp/update_run.sh
- ./scripts/clusterID:/tmp/clusterID
- ./data/kafka2/kraft-combined-logs:/tmp/kraft-combined-logs
command: "bash -c 'if [ ! -f /tmp/update_run.sh ]; then echo \"ERROR: Did you forget the update_run.sh file that came with this
docker-compose.yml file?\" && exit 1 ; else /tmp/update_run.sh && /etc/confluent/docker/run ; fi'"
```

#### schema-registry0:

```
image: confluentinc/cp-schema-registry:7.2.1
ports:
- 8085:8085
depends_on:
- kafka0
environment:
SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: PLAINTEXT://kafka0:29092
SCHEMA_REGISTRY_KAFKASTORE_SECURITY_PROTOCOL: PLAINTEXT
SCHEMA_REGISTRY_HOST_NAME: schema-registry0
SCHEMA_REGISTRY_LISTENERS: http://schema-registry0:8085

SCHEMA_REGISTRY_SCHEMA_REGISTRY_INTER_INSTANCE_PROTOCOL: "http"
SCHEMA_REGISTRY_LOG4J_ROOT_LOGLEVEL: INFO
SCHEMA_REGISTRY_KAFKASTORE_TOPIC: _schemas
```

#### kafka-connect0:

```
image: confluentinc/cp-kafka-connect:7.2.1
ports:
- 8083:8083
depends_on:
- kafka0
- schema-registry0
environment:
CONNECT_BOOTSTRAP_SERVERS: kafka0:29092
CONNECT_GROUP_ID: compose-connect-group
CONNECT_CONFIG_STORAGE_TOPIC: _connect_configs
CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR: 1
CONNECT_OFFSET_STORAGE_TOPIC: _connect_offset
CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR: 1
CONNECT_STATUS_STORAGE_TOPIC: _connect_status
CONNECT_STATUS_STORAGE_REPLICATION_FACTOR: 1
CONNECT_KEY_CONVERTER: org.apache.kafka.connect.storage.StringConverter
CONNECT_KEY_CONVERTER_SCHEMA_REGISTRY_URL: http://schema-registry0:8085
CONNECT_VALUE_CONVERTER: org.apache.kafka.connect.storage.StringConverter
CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: http://schema-registry0:8085
CONNECT_INTERNAL_KEY_CONVERTER: org.apache.kafka.connect.json.JsonConverter
CONNECT_INTERNAL_VALUE_CONVERTER: org.apache.kafka.connect.json.JsonConverter
CONNECT_REST_ADVERTISED_HOST_NAME: kafka-connect0
CONNECT_PLUGIN_PATH: "/usr/share/java,/usr/share/confluent-hub-components,/usr/share/filestream-connectors,/tmp/kfk"
volumes:
- ./data/kfk:/tmp/kfk:ro
- ./data/kfk/test.txt:/tmp/kfk/test.txt
```

#### # kafka-init-topics:

```
# image: confluentinc/cp-kafka:7.2.1
# volumes:
# - ./data/message.json:/data/message.json
# depends_on:
# - kafka0
# command: "bash -c 'echo Waiting for Kafka to be ready... && \
# cub kafka-ready -b kafka0:29092 1 30 && \
# kafka-topics --create --topic second.users --partitions 3 --replication-factor 1 --if-not-exists --bootstrap-server kafka0:29092
&& \
# kafka-topics --create --topic second.messages --partitions 2 --replication-factor 1 --if-not-exists --bootstrap-server
kafka0:29092 && \
# kafka-topics --create --topic first.messages --partitions 2 --replication-factor 1 --if-not-exists --bootstrap-server kafka0:29092
&& \
```

```
# kafka-console-producer --bootstrap-server kafka0:29092 --topic second.users < /data/message.json"
```

```
# https://docs.kafka-ui.provectus.io/configuration/complex-configuration-examples/kraft-mode-+-multiple-brokers
```

data/message.json

```
{}
```

data/proxy.conf

```
server {
    listen    80;
    server_name localhost;

    location /kafka-ui {
        # rewrite /kafka-ui/(.*) /$1 break;
        proxy_pass http://kafka-ui:8080;
    }
}
```

scripts/clusterID

```
zIFiTJeITouhnlFwLWixw
```

scripts/create\_cluster\_id.sh

```
kafka-storage random-uuid > /workspace/kafka-ui/documentation/compose/clusterID
```

scripts/create\_update\_run\_cluster.sh

```
# This script is required to run kafka cluster (without zookeeper)
#!/bin/sh

# Docker workaround: Remove check for KAFKA_ZOOKEEPER_CONNECT parameter
sed -i '/KAFKA_ZOOKEEPER_CONNECT/d' /etc/confluent/docker/configure

# Docker workaround: Ignore cub zk-ready
sed -i 's/cub zk-ready/echo ignore zk-ready/' /etc/confluent/docker/ensure

# KRaft required step: Format the storage directory with a new cluster ID
echo "kafka-storage format --ignore-formatted -t $(cat /tmp/clusterID) -c /etc/kafka/kafka.properties" >> /etc/confluent/docker/ensure
```

scripts/update\_run.sh

```
# This script is required to run kafka cluster (without zookeeper)
#!/bin/sh

# Docker workaround: Remove check for KAFKA_ZOOKEEPER_CONNECT parameter
sed -i '/KAFKA_ZOOKEEPER_CONNECT/d' /etc/confluent/docker/configure

# Docker workaround: Ignore cub zk-ready
sed -i 's/cub zk-ready/echo ignore zk-ready/' /etc/confluent/docker/ensure

# KRaft required step: Format the storage directory with a new cluster ID
echo "kafka-storage format --ignore-formatted -t $(kafka-storage random-uuid) -c /etc/kafka/kafka.properties" >> /etc/confluent/docker/ensure
```

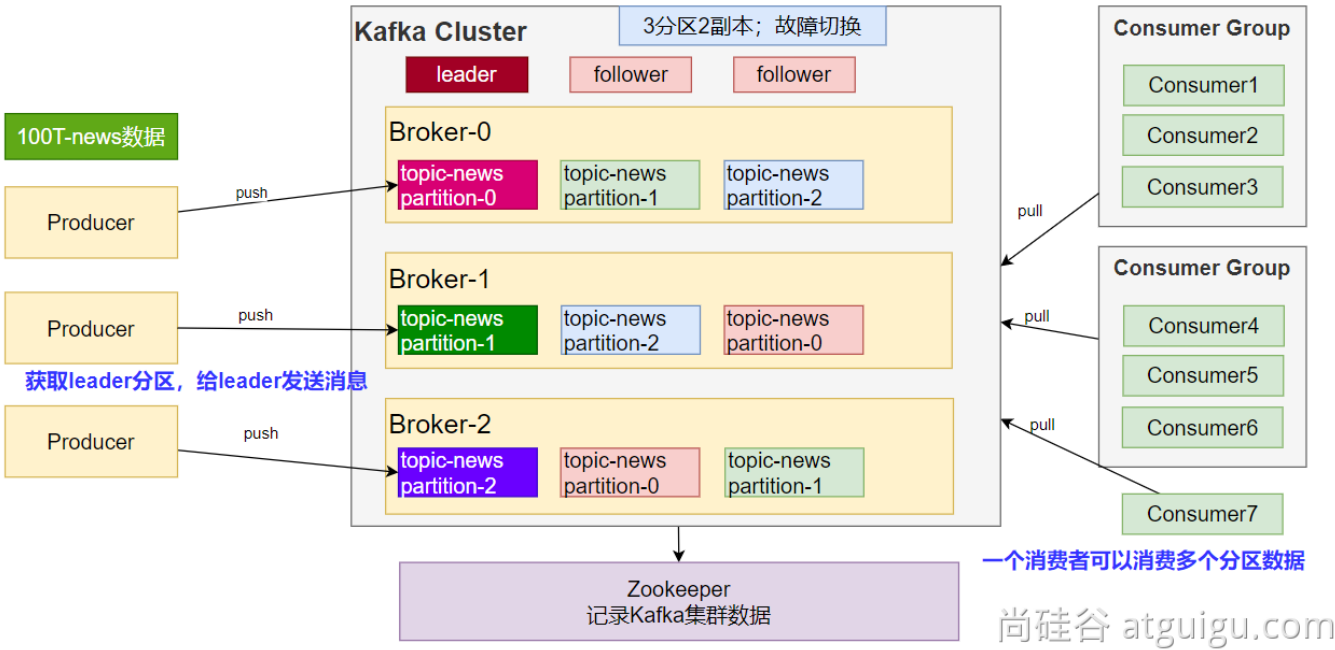


# Kafka 架構

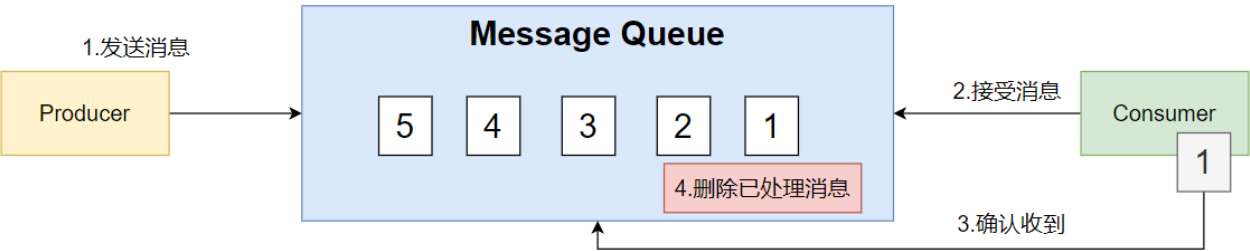
## Kafka原理

分区: 海量数据分散存储  
副本: 每个数据区都有备份

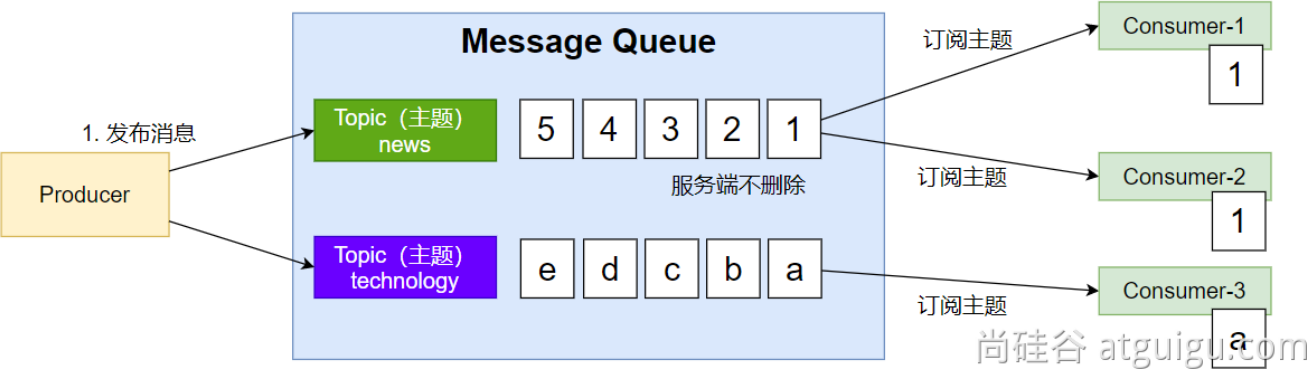
同一个消费者组里面的消费者是队列竞争模式  
不同消费者组里面的消费者是发布/订阅模式



## 消息点对点模式



## 消息发布订阅模式



# Kafka 學習筆記

## 一、概述

### 1、定义

#### 1) 统一定义

分布式 发布消息列

发布消息：分多种类型 生产者根据需求 选择性

#### 2) 最新定义

流平台（存、算）

### 2、消息列的应用场景

#### 1) 存消峰

#### 2) 解耦

#### 3) 异步通信

### 3、两种模式

#### 1) 点对点

(1) 一个生产者 一个消费者 一个topic 会删除数据 不多

#### 2) 发布消息

(1) 多个生产者 消费者多个 而且相互独立 多个topic 不会删除数据

### 4、架构

#### 1) 生产者

100T数据

#### 2) broker

(1) broker 服务器 hadoop102 103 104

(2) topic 主题 数据分类

(3) 分区

(4) 可靠性 副本

(5) leader follower

(6) 生产者和消费者 只读leader操作

#### 3) 消费者

(1) 消费者和消费者相互独立

(2) 消费者 (某个分区 只能由一个消费者消费)

#### 4) zookeeper

(1) broker.ids 0 1 2

(2) leader

## 二、入门

### 1、安装

1) broker.id 必须全局唯一

2) broker.id、log.dirs zk/kafka

3) 启动停止 先停止kafka 再停zk

#### 4) 脚本

```
#!/bin/bash
```

```
case $1 in
```

```
"start")
```

```
for i in hadoop102 hadoop103 hadoop104
```

```
do
```

```
ssh $i "mkdir"
```

```
done
```

```
;;
```

```
"stop")
```

```
;;
```

```
esac
```

### 2、常用命令行

#### 1) 主题 kafka-topic.sh

(1) --bootstrap-server hadoop102:9092,hadoop103:9092

(2) --topic first

(3) --create

(4) --delete

(5) --alter

(6) --list

(7) --describe

(8) --partitions

(9) --replication-factor

#### 2) 生产者 kafka-console-producer.sh

(1) --bootstrap-server hadoop102:9092,hadoop103:9092

(2) --topic first

#### 3) 消费者 kafka-console-consumer.sh

(1) --bootstrap-server hadoop102:9092,hadoop103:9092

(2) --topic first

## 三、生产者

### 1、原理

### 2、异步发送API

#### 0) 配置

(1) 连接 bootstrap-server

(2) key value序列化

#### 1) 创建生产者

```
KafkaProducer<String,String>()
```

#### 2) 发送数据

```
send() send(,new Callback)
```

3) 源码

3、同步发送

。。。

send() send(,new Callback).get()

。。。

4、分片

1) 好处

存片

计算

2) 默认分片

(1) 指定分片 按分片走

(2) key key的hashCode值%分片

(3) 有指定key 有指定分片 粘性

第一随机

3) 自定义分片

定义类 实现partitioner接口

5、吞吐量提高

1) 批次大小 16k 32k

2) linger.ms 0 => 5-100ms

3) 内存

4) 内存大小 32m => 64m

6、可靠性

acks

0 丢失数据

1 也可能丢失 输出普通日志

-1 完全可靠 + 副本大于等于2 isr >=2 => 数据重复

7、数据重复

1) 幂等性

<pid, 分片号, 序列号>

默认打印

2) 事务

底层基于幂等性

(1) 初始化

(2) 启动

(3) 消费者offset

(4) 提交

(5) 中止

8、数据有序

分片有序 (有条件)

多分片有序怎么?

9、顺序

1) inflight =1

2) 有幂等性 inflight =1

3) 有幂等性

四、broker

1、zk存储了哪些信息

(1) broker.ids

(2) leader

(3) 辅助 controller

2、工作流程

3、服役

1) 准备一台干服务器 hadoop100

2) 哪个主操作

3) 形成计划

4) 执行计划

5) 计划

4、退役

1) 要退役的点不存数据

2) 退出点

5、副本

1) 副本的好处 提高可靠性

2) 生境中通常2个 默认1个

3) 有leader follower leader

4) isr

5) controller isr[0 2 3] 存活 ar[0 1 2 3]

6) Leader 挂了

7) follower 挂了

8) 副本分配 默认

9) 手动副本分配 制定计划 执行计划 计划

10) leader partition的均衡 10%

11) 手动增加副本因子

6、存机制

broker topic partition log segment 1g 稀疏索引 4kb

7、删除数据

默认7天 3天 7小时

两种 删除

删除:

两种:

8、高效

1) 集群 采用分片

2) 稀疏索引

3) 顺序

4) 零拷贝 和页存

五、消费者

1、总体流程

2、消费者

3、按照主消

0) 配置信息

接

反序列化

id

1) 建消者

2) 主

3) 消据

4、按照分

5、消者案例

id

6、分分配策略 再平衡

7个分 3个消者

range

0 1 2 x

3 4

5 6

roundrobin

轮询

0 3 6

1 4

2 5

粘性 2 2 3

0 3 4

2 6

1 5

7、offset

1) 默存在系统主

2) 自动提交 5s 默

3) 手动提交 同步 异步

4) 指定offset消 seek ()

5) 按照间消

6) 漏消 重复消

8、事务

生端 =》 集群

集群 =》 消者

消者 =》 框架

9、据

1、增加分 增加消者个

2、生 =》 集群 4个参

3、消端 两个参 50m 500条

六、生优 硬件

1、100万日志 \* 人每天生日志100条 = 1亿条 (中型公司)

处理日志速度 1亿条 / (24 \* 3600s) = 1150条/s

1条日志 (0.5k - 2k 1k)

1150条 \* 1k/s = 1m/s

高峰值 (中午小高峰 8-12) : 1m/s \* 20倍 = 20m/s -40m/s

2、多少台服务器

服务器台 = 2 \* (生者峰值生速率 \* 副本 / 100) + 1

= 2 \* (20m/s \* 2 / 100) + 1

= 3 台

3、磁

kafka 按照顺序 机械硬和固硬 顺序速度差不多

1亿条 \* 1k = 100g

100g \* 2个副本 \* 3天 / 0.7 = 1t

建三台服务器总的磁大小 大于1t

4、存

kafka 存 = 堆存 (kafka 部配置) + 页存 (服务器存)

1) 堆存 10-15g

2) 页存 segment (1g) (分Leader (10) \* 1g \* 25%) / 3 = 1g

一台服务器 10g + 1g

5、CPU

32cpu

6、网

:

1、batch.size=16384 linger.ms=0 9.76 MB/sec

2、batch.size=32768 linger.ms=0 9.76 MB/sec

3、batch.size=4096 linger.ms=0 3.81 MB/sec

4、batch.size=4096 linger.ms=50 3.83 MB/sec

5、batch.size=4096 linger.ms=50 compression.type=snappy 3.77 MB/sec

6、batch.size=4096 linger.ms=50 compression.type=zstd 5.68 MB/sec

7、batch.size=4096 linger.ms=50 compression.type=gzip 5.90 MB/sec

8、batch.size=4096 linger.ms=50 compression.type=lz4 3.72 MB/sec

9、batch.size=4096 linger.ms=50 buffer.memory=67108864 3.76 MB/sec

消者 一次处理500条 81.2066m/s

消者 一次处理2000条 138.0992m/s

消费者 一次处理2000条 fetch.max.bytes=104857600 145.2033m/s

# producer\_consumer 範例

## gradle

```
// kafka dependency
// https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
implementation("org.springframework.kafka:spring-kafka")
// https://mvnrepository.com/artifact/org.slf4j/slf4j-api
implementation("org.slf4j:slf4j-api:2.0.5")
// https://mvnrepository.com/artifact/org.slf4j/slf4j-simple
implementation("org.slf4j:slf4j-simple:2.0.5")
testImplementation("org.junit.jupiter:junit-jupiter-api:5.9.2")
testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine")
```

## properties

```
kafka.listen.start:true
spring.application.name=kafka_demo
spring.kafka.bootstrap-servers=10.20.30.40:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.properties.retries=10
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.group-id=group1
```

## producer

```
@Controller
@RequestMapping("/api")
public class ProducerController {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @GetMapping("/kafka")
    public ResponseEntity<String> send() throws InterruptedException {
        for (int i = 0; i < 100; i++) {
            kafkaTemplate.send("first", "data"+i, "data"+i);
            Thread.sleep(1000);
        }
        return ResponseEntity.ok("{\"success:true}");
    }
}
```

## consumer

```
package com.momo.appteam.liveapi.listener;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.annotation.PartitionOffset;
import org.springframework.kafka.annotation.TopicPartition;
import org.springframework.stereotype.Component;

@Component
public class LiveStreamKafkaListener {

    @KafkaListener(topics = "first", autoStartup = "${kafka.listen.start:false}")
    public void consumeLiveStream(ConsumerRecord record)
    { // 参: 收到的 value
        // System.out.println("收到的信息: " + record.toString());
        Object topic = record.topic();
```

```

Object key = record.key();
Object value = record.value();
System.out.println("收到的信息: 【topic】 " + topic + " 【key】 " + key + " 【value】 " + value );
//ConsumerRecord(
// topic = momoLiveStream, partition = 0,
// leaderEpoch = 0, offset = 760,
// CreateTime = 1697695508640,
// serialized key size = 6,
// serialized value size = 9,
// headers = RecordHeaders(headers = [], isReadOnly = false),
// key = liveld, value = afasffasf
// )
}
@KafkaListener(
    groupId = "default",
    autoStartup = "${kafka.listen.start:false}" ,
    topicPartitions = {
        @TopicPartition(
            topic = "first",

            partitionOffsets = {
                @PartitionOffset(partition = "0",initialOffset = "0")
            }
        )
    }
})
public void consumeAllLiveStream(ConsumerRecord record){
    Object topic = record.topic();
    Object key = record.key();
    Object value = record.value();
    System.out.println("收到的信息: 【topic】 " + topic + " 【key】 " + key + " 【value】 " + value );
}
}

```

# kafka connect

架設Kafka與Kafka Connect讓訊息自動同步至MongoDB

connect 說明

<https://docs.confluent.io/platform/current/connect/index.html#connectors>

下載 sink

[https://www.confluent.io/product/connectors/?\\_ga=2.24281594.1343452646.1705657587-1925276915.1697595506&\\_gl=1\\*1ibzdsy\\*\\_ga\\*MTkyNTI3NjkxNS4xNjk3NTk1NTA2\\*\\_ga\\_D2D3EGKSGD\\*MTcwNTY1NzU4N](https://www.confluent.io/product/connectors/?_ga=2.24281594.1343452646.1705657587-1925276915.1697595506&_gl=1*1ibzdsy*_ga*MTkyNTI3NjkxNS4xNjk3NTk1NTA2*_ga_D2D3EGKSGD*MTcwNTY1NzU4N)

<https://www.confluent.io/hub/mongodb/kafka-connect-mongodb>

```
{
  "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
  "key.converter.schemas.enable": "false",
  "database": "momoLiveDB",
  "topics": "momoLiveStream",
  "namespace.mapper.value.collection.field": "order",
  "value.converter.schemas.enable": "false",
  "connection.uri": "mongodb://10.20.43.200:27017/",
  "name": "appLiveStream",
  "value.converter": "org.apache.kafka.connect.json.JsonConverter",
  "namespace.mapper": "com.mongodb.kafka.connect.sink.namespace.mapping.FieldPathNamespaceMapper",
  "key.converter": "org.apache.kafka.connect.storage.StringConverter"
}
```