01_Kafka集群+SpringBoot

时间：2023年1月1日15:59:42

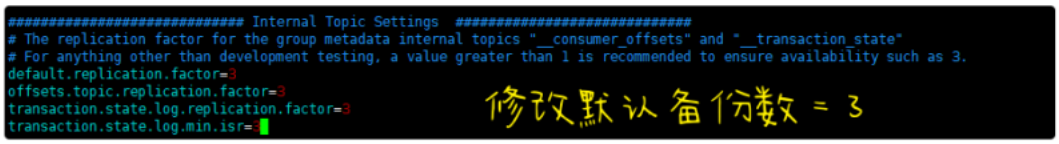
文档：

- 官方示例程序（多语言）
    - Apache Kafka and Java – Getting Started Tutorial
- 第三方（kafka工作机制与核心概念）
    - 关于kafka的分区和副本，这回终于讲清楚了！
    - 细说 Kafka Partition 分区_性能与架构的博客-CSDN博客_kafka partition
        - 细说 Kafka Partition 分区_性能与架构的博客-CSDN博客_kafka partition.pdf
    - Kafka理论之Partition & Replication_逸辰杳的博客-CSDN博客_kafka 默认partition数
        - Kafka理论之Partition & Replication_逸辰杳的博客-CSDN博客_kafka 默认partition数.pdf

---

一、概述

1. 基于Kafka3，使用内置的KRaft完成集群分布式一致性协议；
    a. 9092 服务暴露端口
    b. 9093 集群内节点通信端口



    c. vim /opt/package/kafka_2.13-3.3.1/config/kraft/server.properties

二、RabbitMQ和Kafka有什么区别?

- 消息是否有序

- 消息是否可以重复消费
- 吞吐量

RabbitMQ和kafka比较

1、Kafka可以保证顺序处理消息，RabbitMQ相对较弱。

2、在消息路由和过滤方面，RabbitMQ提供了更好的支持。

3、RabbitMQ有消息存活时间（TTL）和延迟/预定消息功能，Kafka没有。

4、在消息留存方面，RabbitMQ消息一旦消费成功就会删除，反之处理失败则放回，但Kafka会保留消息，根据超时时间来删除消息，所以Kafka可以反复消费消息。

5、在容错处理上，RabbitMQ提供了诸如交付重试和死信交换器（DLX）来处理消息处理故障，相反，Kafka没有提供这种开箱即用的机制，需要在应用层提供和实现消息的重试机制。

6、在伸缩方面，通常Kafka（使用顺序磁盘I/O来提供性能）被认为比RabbitMQ有更优越的性能，从Kafka使用分区的架构上看，它在横向扩展上会优于RabbitMQ，当然，RabbitMQ在纵向扩展上会有更多的优势，而且在吞吐量上，Kafka每秒可处理十几万消息，RabbitMQ每秒可处理几万消息，如果系统达不到百万级用户量，可以不关心伸缩性问题。

7、RabbitMQ（智能代理和傻瓜式消费者模式）比Kafka（傻瓜式代理和智能消费者模式）在消费者复杂度上更简单。

8、优先选择RabbitMQ的条件：

```
·高级灵活的路由规则
·消息时序控制（控制消息过期或消息延迟）
·高级的容错处理能力，在消费者更有可能处理消息不成功的情景中（瞬时或持久）
·更简单的消费者实现
```

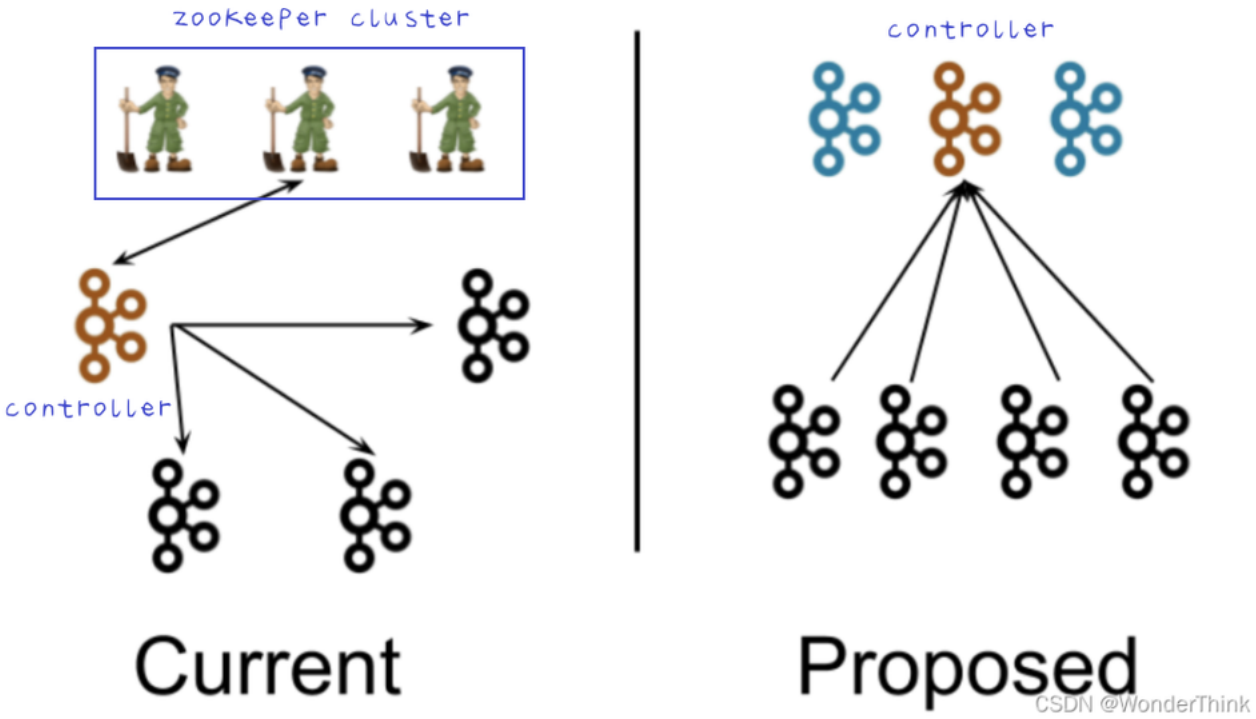
9、 优先选择Kafka的条件：

```
·严格的消息顺序
·延长消息留存时间，包括过去消息重放的可能
·传统解决方案无法满足的高伸缩能力
```

三、部署kafka3集群（前置概念）

1. kraft集群

首先来看一下 KRaft 在系统架构层面和之前的版本有什么区别。KRaft 模式提出了去 Zookeeper后的 Kafka 整体架构如下，下图是前后的架构图对比：



CSDN @WonderThink

在当前架构中，Kafka集群包含多个broker节点和一个ZooKeeper 集群。我们在这张图中描绘了一个典型的集群结构：4个broker节点和3个ZooKeeper节点。Kafka 集群的controller (橙色)在被选中后，会从 ZooKeeper 中加载它的状态。controller 指向其他 broker节点的箭头表示 controller 在通知其他 broker 发生了变更。

在新的架构中，三个 controller 节点替代三个ZooKeeper节点。控制器节点和 broker 节点运行在不同的进程中。controller 节点中会选择一个成为Leader(橙色)。新的架构中，控制器不会向 broker 推送更新，而是 broker 从这个 controller Leader 拉取元数据的更新信息。

需要特别注意的是，尽管 controller 进程在逻辑上与 broker 进程是分离的，但它们不需要在物理上分离。即在某些情况下，部分或所有 controller 进程和 broker 进程是可以是同一个进程，即一个broker节点即是broker也是controller。另外在同一个节点上可以运行两个进程，一个是controller进程，一个是broker进程，这相当于在较小的集群中，ZooKeeper进程可以像Kafka broker一样部署在相同的节点上。

2. Process Roles

每个Kafka服务器现在都有一个新的配置项，叫做`process.roles`，这个参数可以有以下值:

- 如果process.roles = broker, 服务器在KRaft模式中充当 broker。
- 如果process.roles = controller, 服务器在KRaft模式下充当 controller。
- 如果process.roles = broker,controller，服务器在KRaft模式中同时充当 broker 和controller。
- 如果process.roles 没有设置。那么集群就假定是运行在ZooKeeper模式下。

3. Quorum Voters

系统中的所有节点都必须设置 `controller.quorum.voters` 配置。这个配置标识有哪些节点是 Quorum 的投票者节点。所有想成为控制器的节点都需要包含在这个配置里面。这类似于在使用ZooKeeper时，使用ZooKeeper.connect配置时必须包含所有的ZooKeeper服务器。

Quorum voters（仲裁投票者：controller）

然而，与ZooKeeper配置不同的是，`controller.quorum.voters` 配置需要包含每个节点的id。格式为: id1@host1:port1,id2@host2:port2。

因此，如果你有10个broker和 3个控制器，分别命名为controller1、controller2、controller3，你可能在 controller1上有以下配置:

```
process.roles=controller
node.id=1
listeners=CONTROLLER://controller1.example.com:9093
controller.quorum.voters=1@controller1.example.com:9093,2@controller2.example.com:9093,3@controller3.examp
```

每个broker和每个controller 都必须设置 `controller.quorum.voters`。需要注意的是，controller.quorum.voters 配置中提供的节点ID必须与提供给服务器的节点ID匹配。

比如在controller1上，node.id必须设置为1，以此类推。注意，控制器id不强制要求你从0或1开始。客户端不需要配置 `controller.quorum.voters`，只有服务端需要配置。

四、部署kafka3集群（实际操作，非Docker部署）

> 推荐阅读：
>
> - https://blog.csdn.net/WonderThink/article/details/123582097

1. 下载二进制文件

- https://kafka.apache.org/downloads

2. 解压修改配置文件

- 配置Java环境
- 修改/etc/hosts

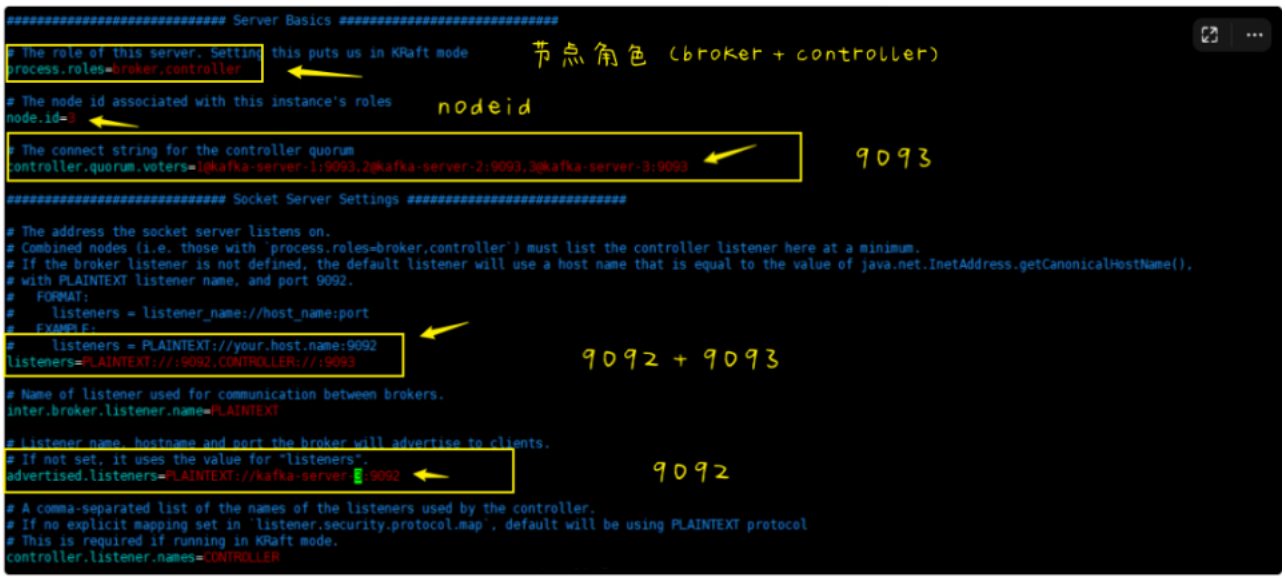
```
cat << EOF >> /etc/hosts
192.168.204.112 kafka-server-1
192.168.204.113 kafka-server-2
192.168.204.114 kafka-server-3
EOF
```

- tar -zvxf kafka_2.13-3.3.1.tgz

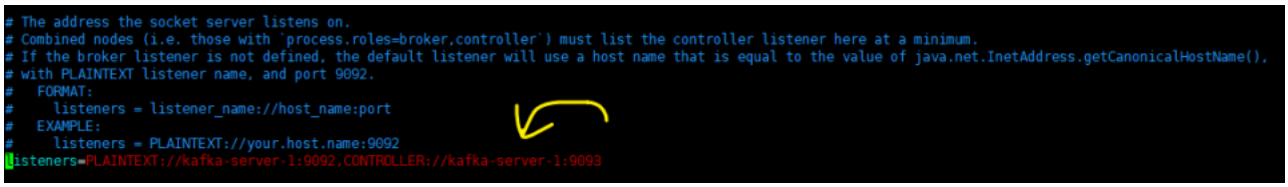
```
[root@nexus package]#
[root@nexus package]# cd kafka_2.13-3.3.1
[root@nexus kafka_2.13-3.3.1]# ll
总用量 64K
drwxr-xr-x. 3 root root 4.0K 9月  30 03:06 bin
drwxr-xr-x. 3 root root 4.0K 9月  30 03:06 config
drwxr-xr-x. 2 root root 8.0K 1月   1 16:59 libs
-rw-rw-r--. 1 root root  15K 9月  30 03:03 LICENSE
drwxr-xr-x. 2 root root  284 9月  30 03:06 licenses
-rw-rw-r--. 1 root root  28K 9月  30 03:03 NOTICE
drwxr-xr-x. 2 root root   44 9月  30 03:06 site-docs
[root@nexus kafka_2.13-3.3.1]#
```

kafka目录

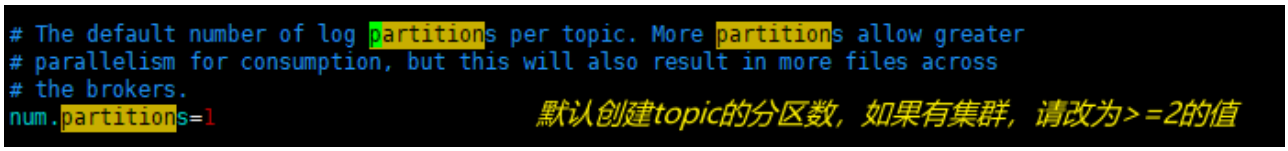
- ./config/kraft/server.properties



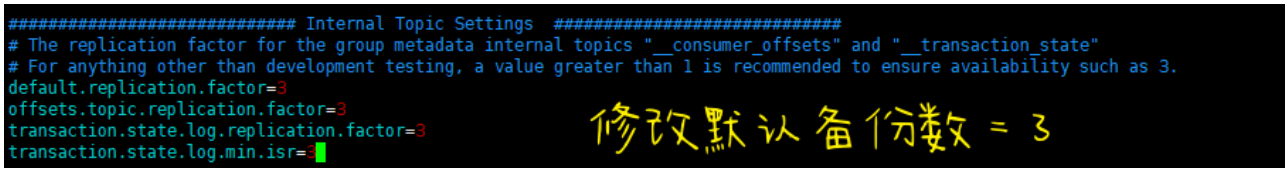
（注意listeners需要配置主机名/ip）



（修改默认分区数与备份数）

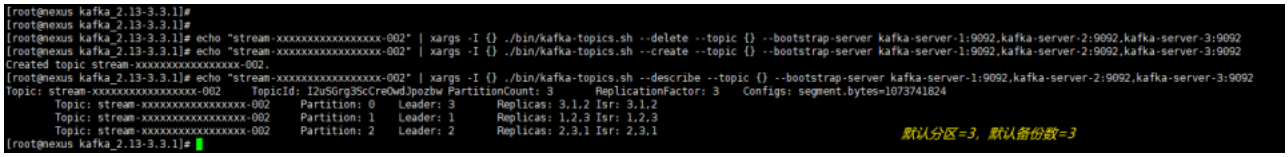


- 主题创建后，手动增加分区：https://cloud.tencent.com/developer/article/1378346




```
############################## Internal Topic Settings  ##############################
# The replication factor for the group metadata internal topics "__consumer_offsets" and "__transaction_state"
# For anything other than development testing, a value greater than 1 is recommended to ensure availability such as 3.
default.replication.factor=3
offsets.topic.replication.factor=3
transaction.state.log.replication.factor=3
transaction.state.log.min.isr=3
```

（修改默认分区数和备份数）



- 初始化集群ID，格式化存储目录

```
### KAFKA_SERVER_CLUSTER_ID=E014A984F2544921A13D5CD0637E6A6E
cd /opt/package/kafka_2.13-3.3.1
KAFKA_SERVER_CLUSTER_ID=$(./bin/kafka-storage.sh random-uuid)
echo $KAFKA_SERVER_CLUSTER_ID
### TN7EkWNLR3ew35EYVoPK1A

./bin/kafka-storage.sh format -t TN7EkWNLR3ew35EYVoPK1A -c ./config/kraft/server.properties
```

```
[root@nexus kafka_2.13-3.3.1]# ./bin/kafka-storage.sh format -t TN7EkWNLR3ew35EYVoPK1A -c ./config/kraft/server.properties
Formatting /tmp/kraft-combined-logs with metadata.version 3.3-IV3.
[root@nexus kafka_2.13-3.3.1]#
```

- 修改kafka初始化JVM大小

```
cd /opt/package/kafka_2.13-3.3.1 && vim ./bin/kafka-server-start.sh
```

```
#!/bin/bash
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

if [ $# -lt 1 ];
then
    echo "USAGE: $0 [-daemon] server.properties [--override property=value]*"
    exit 1
fi
base_dir=$(dirname $0)

if [ "x$KAFKA_LOG4J_OPTS" = "x" ]; then
    export KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:$base_dir/../config/log4j.properties"
fi

if [ "x$KAFKA_HEAP_OPTS" = "x" ]; then
    export KAFKA_HEAP_OPTS="-Xmx512M -Xms512M"
fi

EXTRA_ARGS=${EXTRA_ARGS-'-name kafkaServer -loggc'}

COMMAND=$1
case $COMMAND in
  -daemon)
    EXTRA_ARGS="-daemon "$EXTRA_ARGS
    shift
    ;;
  *)
    ;;
esac

exec $base_dir/kafka-run-class.sh $EXTRA_ARGS kafka.Kafka "$@"
~
~
~
~
```

修改kafka初始化JVM大小

3. 启动节点服务

- 启动

```
cd /opt/package/kafka_2.13-3.3.1 && nohup ./bin/kafka-server-start.sh ./config/kraft/server.properties &
```

- 启动日志

```
[root@nexus kraft-combined-logs]# cd /opt/package/kafka_2.13-3.3.1
[root@nexus kafka_2.13-3.3.1]# ll
总用量 140K
drwxr-xr-x. 3 root root 4.0K 1月   1 17:21 bin
drwxr-xr-x. 3 root root 4.0K 9月  30 03:06 config
-rw-r--r--. 1 root root  24K 1月   1 17:17 hs_err_pid46821.log
-rw-r--r--. 1 root root  24K 1月   1 17:18 hs_err_pid47977.log
-rw-r--r--. 1 root root  24K 1月   1 17:19 hs_err_pid48572.log
drwxr-xr-x. 2 root root 8.0K 1月   1 16:59 libs
-rw-rw-r--. 1 root root  15K 9月  30 03:03 LICENSE
drwxr-xr-x. 2 root root  284 9月  30 03:06 licenses
drwxr-xr-x. 2 root root 4.0K 1月   1 17:26 logs
-rw-rw-r--. 1 root root  28K 9月  30 03:03 NOTICE
drwxr-xr-x. 2 root root   44 9月  30 03:06 site-docs
[root@nexus kafka_2.13-3.3.1]# cd logs
[root@nexus logs]# ll
总用量 264K
-rw-r--r--. 1 root root    0 1月   1 17:22 controller.log
-rw-r--r--. 1 root root    0 1月   1 17:22 kafka-authorizer.log
-rw-r--r--. 1 root root    0 1月   1 17:22 kafka-request.log
-rw-r--r--. 1 root root  15K 1月   1 17:27 kafkaServer-gc.log
-rw-r--r--. 1 root root   61 1月   1 17:17 kafkaServer-gc.log.0
-rw-r--r--. 1 root root   61 1月   1 17:18 kafkaServer-gc.log.1
-rw-r--r--. 1 root root   61 1月   1 17:19 kafkaServer-gc.log.2
-rw-r--r--. 1 root root  14K 1月   1 17:23 kafkaServer-gc.log.3
-rw-r--r--. 1 root root  15K 1月   1 17:24 kafkaServer-gc.log.4
-rw-r--r--. 1 root root  722 1月   1 17:26 log-cleaner.log
-rw-r--r--. 1 root root 198K 1月   1 17:26 server.log
-rw-r--r--. 1 root root    0 1月   1 17:22 state-change.log
[root@nexus logs]#
```

kafka 启动日志

- 检查是否启动成功
  - 检查端口：

```
[root@nexus kraft-combined-logs]# n 909
tcp        0      0 0.0.0.0:9092            0.0.0.0:*               LISTEN      54086/java
tcp        0      0 0.0.0.0:9093            0.0.0.0:*               LISTEN      54086/java
[root@nexus kraft-combined-logs]#
```

  - 查看启动日志：t /opt/package/kafka_2.13-3.3.1/logs/server.log

四、测试

```
cd /opt/package/kafka_2.13-3.3.1
```

```
./bin/kafka-topics.sh --create --topic foo --partitions 1 --replication-factor 3 --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092

echo "foo" | xargs -I {} ./bin/kafka-topics.sh --create --topic {} --partitions 1 --replication-factor 3 --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```

```
./bin/kafka-topics.sh --list --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```

```
./bin/kafka-topics.sh --describe --topic foo --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092

echo "foo" | xargs -I {} ./bin/kafka-topics.sh --describe --topic {}  --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```


```
./bin/kafka-topics.sh --delete --topic foo -bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092

echo "foo" | xargs -I {} ./bin/kafka-topics.sh --delete --topic {} --partitions 1 --replication-factor 3 --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```

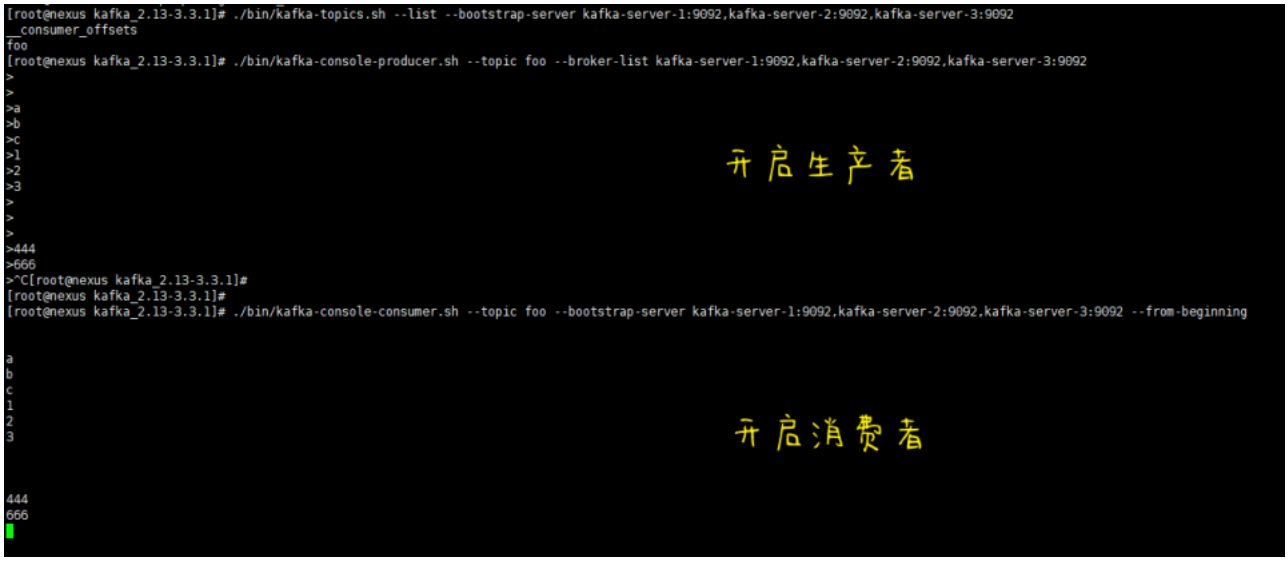
1. 开启消费者

```
echo "foo" | xargs -I {} ./bin/kafka-console-consumer.sh --topic {} --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092 --from-beginning
```

2. 开启生产者

```
./bin/kafka-console-producer.sh --topic foo --broker-list kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```

3. 生产消费结果测试



五、SpringBoot集成Kafka集群（Kraft）

1. 引入依赖

```
<!-- kafka -->
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
```

```
    <version>2.9.4</version>
</dependency>

<!-- spring-boot-test -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <version>2.6.8</version>
</dependency>
```

2. 配置文件

- 消费者端配置
    - Kafka Consumer Configurations for Confluent Platform | Confluent Documentation
    - Apache Kafka and Java – Getting Started Tutorial

```
#kafka-server
192.168.204.112 kafka-server-1
192.168.204.113 kafka-server-2
192.168.204.114 kafka-server-3
```

修改hosts文件

```
spring:
  # datasource:
  #   type: com.alibaba.druid.pool.DruidDataSource
  #   driver-class-name: com.mysql.cj.jdbc.Driver
  #   url: jdbc:mysql://192.168.204.112:23306/dev_test?
autoReconnect=true&useServerPreparedStmts=true&cachePrepStmts=true&rewriteBatchedStatements=true&allowMultiQu
```

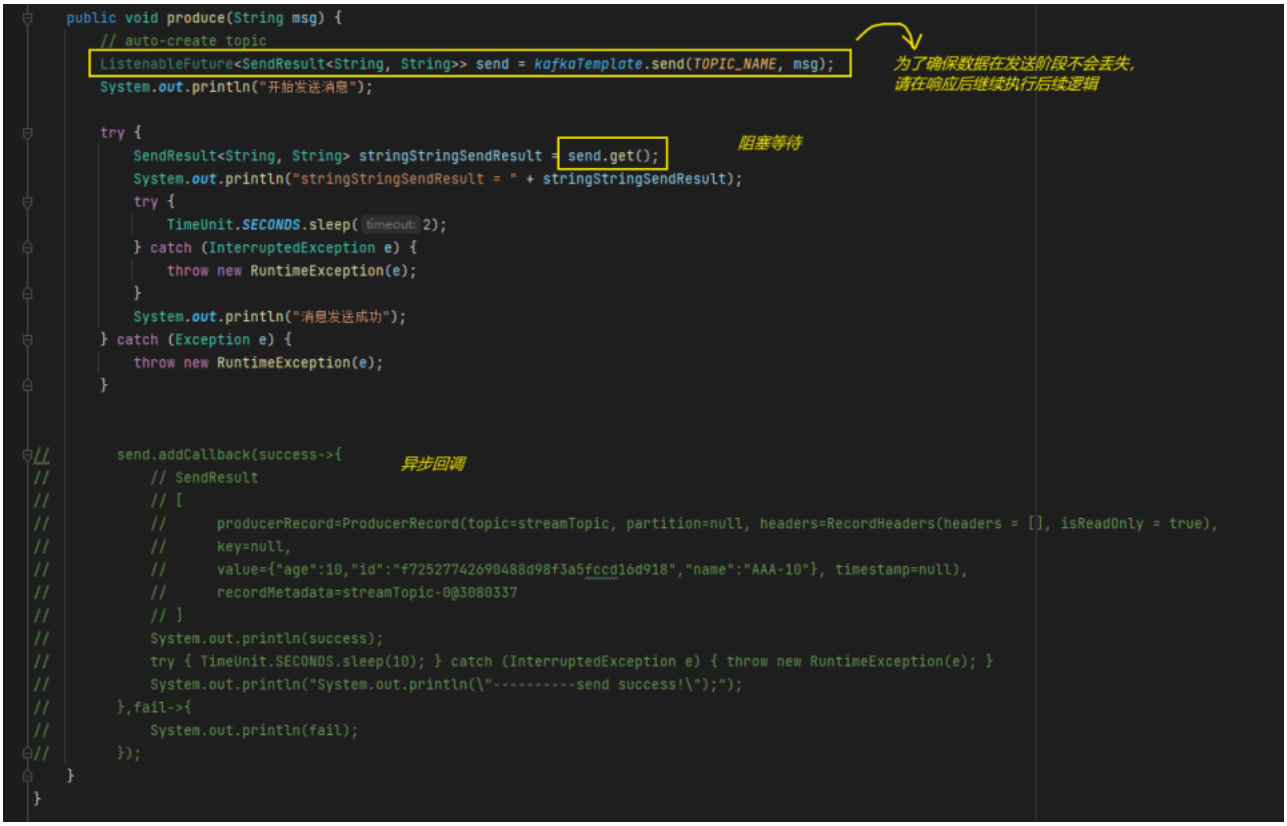
3. 生产者与消费者

```
import lombok.RequiredArgsConstructor;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class KafkaProducer {

    private final KafkaTemplate<String, String> kafkaTemplate;

    public void produce(String msg) {
        // auto-create topic
        kafkaTemplate.send("topic-foo", msg);
    }
}
```




```
public void produce(String msg) {
    // auto-create topic
    ListenableFuture<SendResult<String, String>> send = kafkaTemplate.send(TOPIC_NAME, msg);
    System.out.println("开始发送消息");

    try {
        SendResult<String, String> stringStringSendResult = send.get();
        System.out.println("stringStringSendResult = " + stringStringSendResult);
        try {
            TimeUnit.SECONDS.sleep( timeout: 2);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        System.out.println("消息发送成功");
    } catch (Exception e) {
        throw new RuntimeException(e);
    }


//    send.addCallback(success->{
//        // SendResult
//        // [
//        //      producerRecord=ProducerRecord(topic=streamTopic, partition=null, headers=RecordHeaders(headers = [], isReadOnly = true),
//        //      key=null,
//        //      value={"age":10,"id":"f72527742690488d98f3a5fccd16d918","name":"AAA-10"}, timestamp=null),
//        //      recordMetadata=streamTopic-0@3080337
//        // ]
//        System.out.println(success);
//        try { TimeUnit.SECONDS.sleep(10); } catch (InterruptedException e) { throw new RuntimeException(e); }
//        System.out.println("System.out.println(\"----------send success!\");");
//    },fail->{
//        System.out.println(fail);
//    });
}
}
```

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

import java.util.Optional;

@Component
public class KafkaConsumer {

    // 默认单条数据消费
    // 1. 可以使用ConsumerRecord<String, String>接收
    // 2. 也可以使用String直接接收传入的值
    @KafkaListener(topics = "topic-foo", groupId = "group-a")
    public void ProductInsertEvent1(ConsumerRecord<String, String> record) {
        Optional<String> kafkaMessage = Optional.ofNullable(record.value());
        kafkaMessage.ifPresent(msg -> {
            System.out.printf("KafkaConsumer1 kafka value:%s%n", msg);
        });
    }

    @KafkaListener(topics = "topic-foo", groupId = "group-b")
    public void ProductInsertEvent2(ConsumerRecord<String, String> record) {
        Optional<String> kafkaMessage = Optional.ofNullable(record.value());
        kafkaMessage.ifPresent(msg -> {
            System.out.printf("KafkaConsumer2 kafka value:%s%n", msg);
        });
    }

}
```

4. 测试

```
import com.xii.mp.kafka.KafkaProducer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;

@SpringBootTest
public class AppTest {

    @Autowired
    private KafkaProducer kafkaProducer;

    @Test
    public void test01() {
        for (int i = 0; i < 1000; i++) {
            kafkaProducer.produce("hello,kafka-" + i);
            System.out.println("hello,kafka-" + i+": 消息发送成功！");
        }

        try {
            TimeUnit.SECONDS.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

六、手动扩容Topic分区

> 这里只说明扩容Topic分区，如果节点扩容后负载均衡Topic请阅读：https://cloud.tencent.com/developer/article/1559892

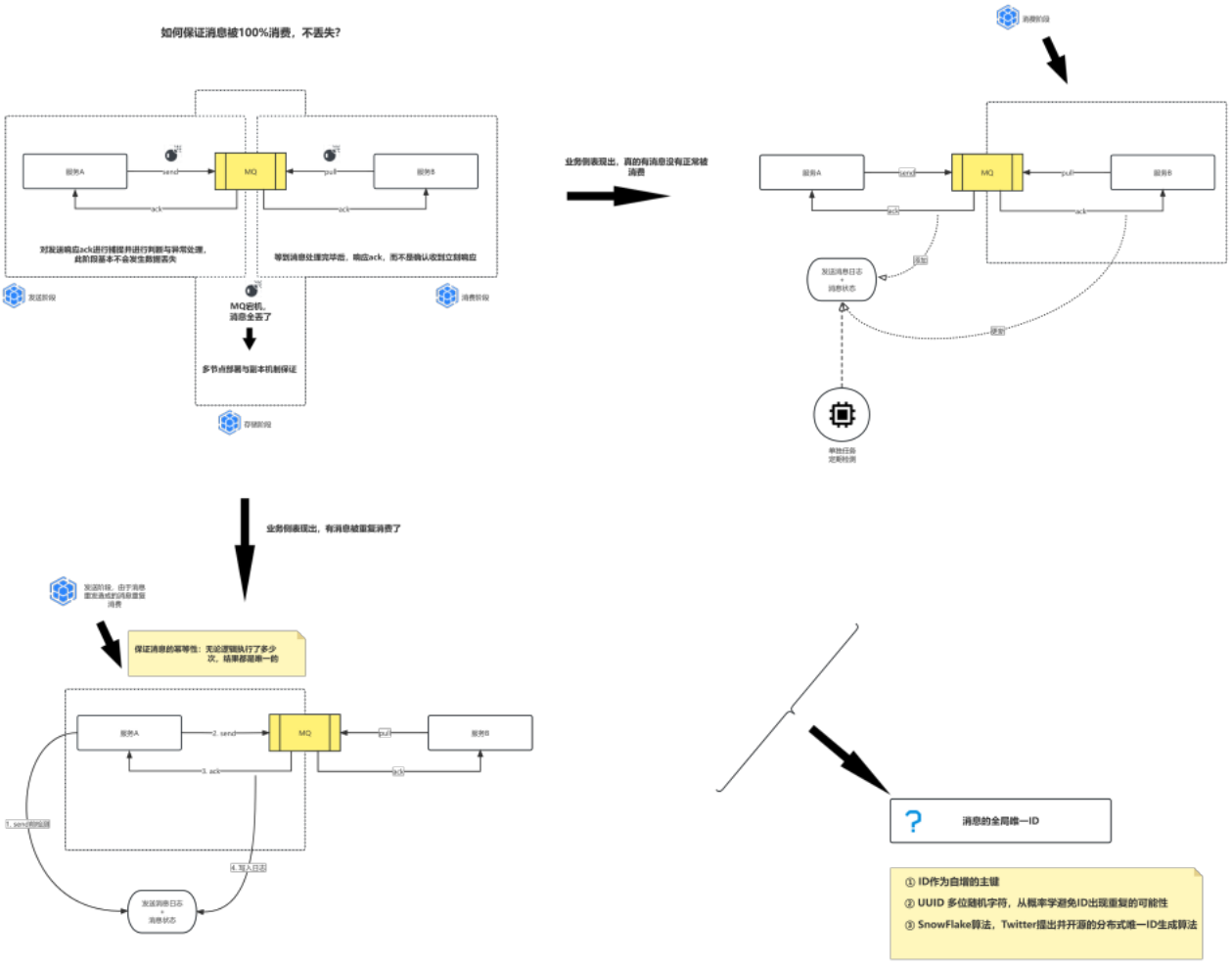
> 为什么要扩容某个Topic的分区？ 解决Kafka消息堆积问题
>
> - 分区只能增加，不能减少

```
[root@rabbitmq kafka_2.13-3.3.1]# cd bin
[root@rabbitmq bin]# ll
总用量 164K
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 connect-distributed.sh
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 connect-mirror-maker.sh
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 connect-standalone.sh
-rwxrwxr-x. 1 root root  861 9月  30 03:03 kafka-acls.sh
-rwxrwxr-x. 1 root root  873 9月  30 03:03 kafka-broker-api-versions.sh
-rwxrwxr-x. 1 root root  860 9月  30 03:03 kafka-cluster.sh
-rwxrwxr-x. 1 root root  864 9月  30 03:03 kafka-configs.sh
-rwxrwxr-x. 1 root root  945 9月  30 03:03 kafka-console-consumer.sh
-rwxrwxr-x. 1 root root  944 9月  30 03:03 kafka-console-producer.sh
-rwxrwxr-x. 1 root root  871 9月  30 03:03 kafka-consumer-groups.sh
-rwxrwxr-x. 1 root root  948 9月  30 03:03 kafka-consumer-perf-test.sh
-rwxrwxr-x. 1 root root  871 9月  30 03:03 kafka-delegation-tokens.sh
-rwxrwxr-x. 1 root root  869 9月  30 03:03 kafka-delete-records.sh
-rwxrwxr-x. 1 root root  866 9月  30 03:03 kafka-dump-log.sh
-rwxrwxr-x. 1 root root  863 9月  30 03:03 kafka-features.sh
-rwxrwxr-x. 1 root root  865 9月  30 03:03 kafka-get-offsets.sh
-rwxrwxr-x. 1 root root  870 9月  30 03:03 kafka-leader-election.sh
-rwxrwxr-x. 1 root root  863 9月  30 03:03 kafka-log-dirs.sh
-rwxrwxr-x. 1 root root  870 9月  30 03:03 kafka-metadata-quorum.sh
-rwxrwxr-x. 1 root root  873 9月  30 03:03 kafka-metadata-shell.sh
-rwxrwxr-x. 1 root root  862 9月  30 03:03 kafka-mirror-maker.sh
-rwxrwxr-x. 1 root root  959 9月  30 03:03 kafka-producer-perf-test.sh
-rwxrwxr-x. 1 root root  874 9月  30 03:03 kafka-reassign-partitions.sh
-rwxrwxr-x. 1 root root  874 9月  30 03:03 kafka-replica-verification.sh
-rwxrwxr-x. 1 root root  11K 9月  30 03:03 kafka-run-class.sh
-rwxrwxr-x. 1 root root 1.4K 1月   1 17:23 kafka-server-start.sh
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 kafka-server-stop.sh
-rwxrwxr-x. 1 root root  860 9月  30 03:03 kafka-storage.sh
-rwxrwxr-x. 1 root root  945 9月  30 03:03 kafka-streams-application-reset.sh
-rwxrwxr-x. 1 root root  863 9月  30 03:03 kafka-topics.sh
-rwxrwxr-x. 1 root root  879 9月  30 03:03 kafka-transactions.sh
-rwxrwxr-x. 1 root root  958 9月  30 03:03 kafka-verifiable-consumer.sh
-rwxrwxr-x. 1 root root  958 9月  30 03:03 kafka-verifiable-producer.sh
-rwxrwxr-x. 1 root root 1.7K 9月  30 03:03 trogdor.sh
drwxrwxr-x. 2 root root 4.0K 9月  30 03:03 windows
-rwxrwxr-x. 1 root root  867 9月  30 03:03 zookeeper-security-migration.sh
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 zookeeper-server-start.sh
-rwxrwxr-x. 1 root root 1.4K 9月  30 03:03 zookeeper-server-stop.sh
-rwxrwxr-x. 1 root root 1019 9月  30 03:03 zookeeper-shell.sh
[root@rabbitmq bin]#
```


```
echo "notAdultTopic" | xargs -I {} ./bin/kafka-topics.sh --alter --topic {} --partitions 4 --bootstrap-server kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
```

七、MQ的三个问题?

1. 消息丢失
2. 消息重复消费
3. 消息堆积


如何保证消息被100%消费，不丢失?
服务A
send
MQ
pull
服务B
ack
对发送响应ack进行捕捉并进行判断与异常处理，
此阶段基本不会发生数据丢失
等到消息处理完毕后，响应ack，而不是确认收到立刻响应
发送阶段
消费阶段
MQ宕机，
消息全丢了
多节点部署与副本机制保证
存储阶段
业务侧表现出，真的有消息没有正常被
消费
消费阶段
服务A
send
MQ
pull
服务B
ack
ack
持久
发送消息日志
+
消息状态
硬刷
单独任务
定期检测
业务侧表现出，有消息被重复消费了
发送阶段，由于消息
重发造成的消息重复
消费
保证消息的幂等性：无论逻辑执行了多少
次，结果都是唯一的
服务A
2. send
MQ
pull
服务B
3. ack
ack
1. send消息
4. 写入日志
发送消息日志
+
消息状态
消息的全局唯一ID
① ID作为自增的主键
② UUID 多位随机字符，从概率学避免ID出现重复的可能性
③ SnowFlake算法，Twitter提出并开源的分布式唯一ID生成算法
消息堆积
1. 为什么会出现消息堆积
生产端 X
中间件 X消费端性
能问题 √
2. 如何处理 + 解决
线上突发问题：① 临时扩容，增加消费端数量 ② 降级一些非必要业务
解决问题：通过监控、日志等措施，分析消费端业务逻辑的具体问题并优化解决
先处理，后解决
扩容注意事项
对于Kafka来说，消费者如果达到分区上限，请同步
扩容消息分区数
一个消息组内，一个消费者可以消费多个分区数据，
但是推荐一个组内的消费者个数和分区数保持一致。