链滴

RocketMQ 入门基础 - 环境 & 整合

作者：jianzh5

原文链接：https://ld246.com/article/1585530045129

来源网站：链滴

概述&选型

消息队列作为高并发系统的核心组件之一，能够帮助业务系统解构提升开发效率和系统稳定性。主要于三种典型场景：**应用解耦、流量消峰、消息分发。**

目前主流的MQ主要是Rocketmq、kafka、Rabbitmq，Rocketmq相比于Rabbitmq、kafka具有主优势特性有：

- 支持事务型消息（消息发送和DB操作保持两方的最终一致性，rabbitmq和kafka不支持）
- 支持结合rocketmq的多个系统之间数据最终一致性（多方事务，二方事务是前提）
- 支持18个级别的延迟消息（rabbitmq和kafka不支持）
- 支持指定次数和时间间隔的失败消息重发（kafka不支持，rabbitmq需要手动确认）
- 支持consumer端tag过滤，减少不必要的网络传输（rabbitmq和kafka不支持）
- 支持重复消费（rabbitmq不支持，kafka支持）

本文主要介绍RocketMQ的单机安装、双机主从高可用安装配置、运维管理平台搭建、与SpringBoo整合几个知识点，具备相关知识技能的同学请直接拉到最后点个“在看”即可。

文章开始之前需要先准备好JDK1.8或以上的服务器环境以及从rocketmq官网下载好二进制安装包，载地址http://rocketmq.apache.org/dowloading/releases/

单机安装配置

工欲善其事必先利其器，要想深入了解RocketMQ得先把环境安装好，咱们先开始单机版RocketMQ安装！

- 解压安装

```
unzip rocketmq-all-4.7.0-bin-release.zip
```

- 启动 Name Server

```
> nohup sh bin/mqnamesrv &
```

- 查看 Name Server启动日志

```
> tail -f ~/logs/rocketmqlogs/namesrv.log
```

```
[root@oadev rocketmq-all-4.7.0-bin-release]# tail -f ~/logs/rocketmqlogs/namesrv.log
2020-03-29 09:20:10 INFO main - tls.client.authServer = false
2020-03-29 09:20:10 INFO main - tls.client.trustCertPath = null
2020-03-29 09:20:10 INFO main - Using OpenSSL provider
2020-03-29 09:20:11 INFO main - SSLContext created for server
2020-03-29 09:20:11 INFO main - Try to start service thread:FileWatchService started:false lastThread:null
2020-03-29 09:20:11 INFO NettyEventExecutor - NettyEventExecutor service started
2020-03-29 09:20:11 INFO FileWatchService - FileWatchService service started
2020-03-29 09:20:11 INFO main - The Name Server boot success. serializeType=JSON
2020-03-29 09:21:11 INFO NSScheduledThread1 ----------------------------------------------------------
2020-03-29 09:21:11 INFO NSScheduledThread1 - configTable SIZE: 0
```

- 启动 Broker Server

```
> nohup sh bin/mqbroker -n localhost:9876 &
```

- 查看 Broker Server 启动日志

```
> tail -f ~/logs/rocketmqlogs/broker.log
```

```
[root@oadev rocketmq-all-4.7.0-bin-release]# tail -f ~/logs/rocketmqlogs/broker.log
2020-03-29 09:23:54 INFO main - Try to start service thread:FileWatchService started:false lastThread:null
2020-03-29 09:23:54 INFO FileWatchService - FileWatchService service started
2020-03-29 09:23:54 INFO main - Try to start service thread:PullRequestHoldService started:false lastThread:null
2020-03-29 09:23:54 INFO PullRequestHoldService - PullRequestHoldService service started
2020-03-29 09:23:54 INFO main - Try to start service thread:TransactionalMessageCheckService started:false lastThread:null
2020-03-29 09:23:54 INFO brokerOutApi_thread_1 - register broker[0]to name server localhost:9876 OK
2020-03-29 09:23:54 INFO main - The broker[oadev, [redacted]:10911] boot success. serializeType=JSON and name server is localhost:9876
2020-03-29 09:24:04 INFO BrokerControllerScheduledThread1 - dispatch behind commit log 0 bytes
2020-03-29 09:24:04 INFO BrokerControllerScheduledThread1 - Slave fall behind master: 0 bytes
2020-03-29 09:24:04 INFO brokerOutApi_thread_2 - register broker[0]to name server localhost:9876 OK
2020-03-29 09:24:34 INFO brokerOutApi_thread_3 - register broker[0]to name server localhost:9876 OK
```

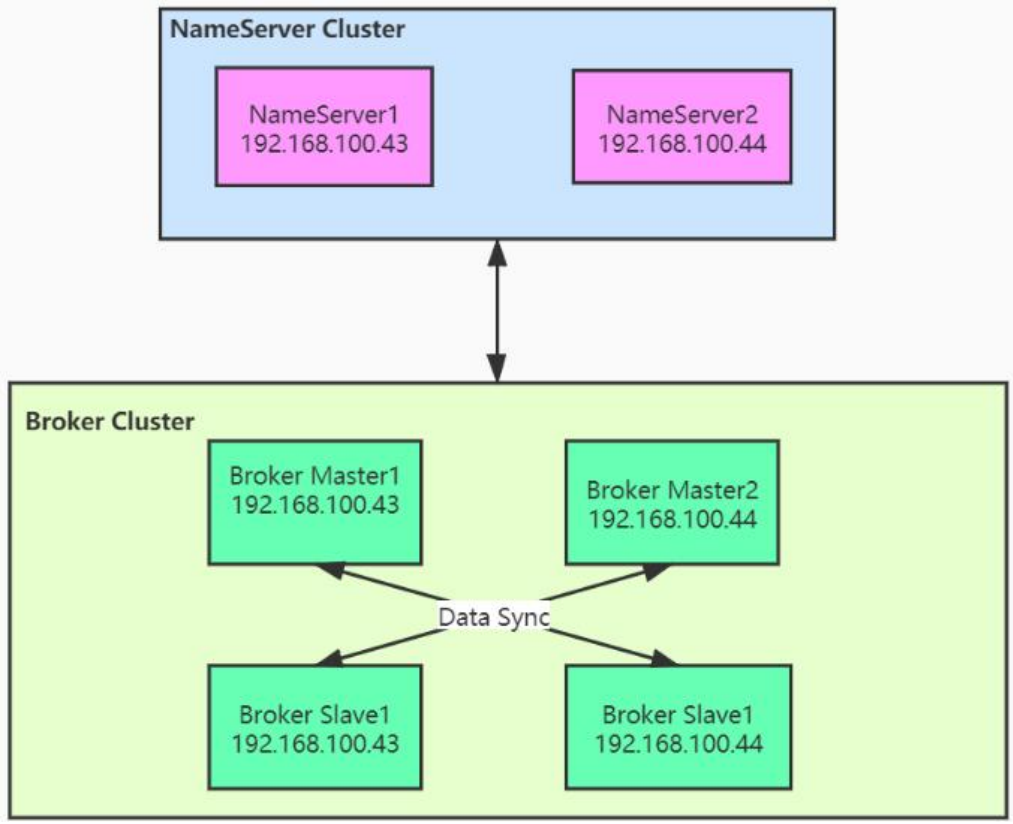
单机情况下安装使用RocketMQ很简单，只需要分别启动NameServer和Broker Server即可！

关闭RockerMQ需要使用下面的命令：

```
# 先关闭Broker Server
> sh bin/mqshutdown broker
# 再关闭NameServer
> sh bin/mqshutdown namesrv
```

双机主从高可用搭建

为了消除单机故障，增加可靠性或增大吞吐量，可以在多台服务器上部署多个NameServer和Broker并为每个Broker部署一个或多个Slave。本节将说明使用两台机器，搭建双主、双从、无单点故障的可用RocketMQ集群。假设现在有两台服务器，IP地址分别为：192.168.100.43和192.168.100.44，署架构如下：



启动多个NameServer 和 Broker

首先需要在两台服务器上分别启动NameServer（nohup sh bin/mqnamesrv &），这样我们就得了一个无单点的NameServer服务，服务地址为192.168.100.43:9876和192.168.100.44:9876。

然后在两台服务器中RocketMQ的conf目录分别建立两个文件 broker-master.properties，broker-slve.properties，下面是不同服务器的配置说明：

- 192.168.100.43 机器上的broker-master.properties文件：

```
namesrvAddr = 192.168.100.43:9876;192.168.100.44:9876
brokerClusterName = DefaultCluster
brokerName = broker-a
brokerId = 0
deleteWhen = 04
fileReservedTime = 48
brokerRole = SYNC_MASTER
flushDiskType = ASYNC_FLUSH
listenPort = 10911
storePathRootDir = /app/rocketmq/store-a
```

- 192.168.100.43 机器上的broker-slave.properties文件：

```
namesrvAddr = 192.168.100.43:9876;192.168.100.44:9876
brokerClusterName = DefaultCluster
brokerName = broker-b
brokerId = 1
deleteWhen = 04
fileReservedTime = 48
```

```
brokerRole = SLAVE
flushDiskType = ASYNC_FLUSH
listenPort = 11011
storePathRootDir = /app/rocketmq/store-b
```

- 192.168.100.44 机器上的broker-master.properties文件：

```
namesrvAddr = 192.168.100.43:9876;192.168.100.44:9876
brokerClusterName = DefaultCluster
brokerName = broker-b
brokerId = 0
deleteWhen = 04
fileReservedTime = 48
brokerRole = SYNC_MASTER
flushDiskType = ASYNC_FLUSH
listenPort = 10911
storePathRootDir = /app/rocketmq/store-b
```

- 192.168.100.44 机器上的broker-slave.properties文件：

```
namesrvAddr = 192.168.100.43:9876;192.168.100.44:9876
brokerClusterName = DefaultCluster
brokerName = broker-a
brokerId = 1
deleteWhen = 04
fileReservedTime = 48
brokerRole = SLAVE
flushDiskType = ASYNC_FLUSH
listenPort = 11011
storePathRootDir = /app/rocketmq/store-a
```

然后分别使用如下命令启动两台服务器的主节点和从节点

```
nohup sh bin/mqbroker -c conf/broker-master.properties &
nohup sh bin/mqbroker -c conf/broker-slave.properties &
```

这样一个高可用的RockerMQ集群就搭建好了，我们登陆可视化运维管理界面查看集群状态，集群正启动。

重要参数说明

本节主要是对Broker的配置文件中用到的参数进行说明

- namesrvAddr = 192.168.100.43:9876;192.168.100.44:9876

指定NameServer的地址，可以是多个。

- brokerClusterName = DefaultCluster

Cluster地址，如果集群数量比较多，可以分成多个Cluster，每个Cluster供一个业务群使用。

- brokerName = broker-a

Broker的名称，Master 和Slave 通过使用相同的 Broker 名称来表明相互关系，以说明某个Slave 是个Master 的 Slave。

- brokerId = 1

一个Master可以有多个Slave，0表示Master，大于0的表示不同Slave的ID。

- fileReservedTime = 48

在磁盘上保存消息的时长，单位是小时，自动删除超时的消息。

- deleteWhen = 04

与 fileReservedTime 参数对应，表明在几点做消息删除动作，默认是凌晨4点。

- brokerRole = SYNC_MASTER

brokerRole的可选参数有SYNC_MASTER，ASYNC_MASTER，SLAVE三种。SYNC 和ASYNC 表示ASTER 和SLAVE 之间同步消息的机制，SYNC的意思是当Slave 和 Master 的消息同步完成后再返回送成功的状态。

- flushDiskType = ASYNC_FLUSH

flushDiskType 表示刷盘策略，可选值有ASYNC_FLUSH 和 SYNC_FLUSH两种，分别代表同步刷盘异步刷盘。同步情况下，消息只有真正写入磁盘才返回成功状态；异步情况下，消息写入page_cach后就返回成功状态。

- listenPort = 11011

Broker监听的端口，一台服务器启动多个Broker，需要设置不同的监听端口避免端口冲突。

- storePathRootDir = /app/rocketmq/store-a

存储消息以及配置信息的根目录。

可视化管理平台

RocketMQ可以使用rocketmq-externals作为运维管理平台，Github地址https://github.com/apache/rocketmq-externals，我们需要将源码下载下来后再进行手动编译，过程如下：

- 下载

从github（https://github.com/apache/rocketmq-externals） 下载RocketMQ可视化管理工具 rocketmq-externals 的源码；

- 打包

下载完成后切换进rocketmq-console目录，使用maven命令对其打包 mvn clean package -Dmaven test.skip=true ，打包完成后生成可执行文件rocketmq-console-ng-1.0.1.jar

- 运行

使用 java -jar rocketmq-console-ng-1.0.1.jar --server.port=8080 --rocketmq.config.namesrvAdr=xxxx.xxx.xxx.xxx:9876 命令启动，这里注意需要设置两个参数：

--server.port 为运行的这个web应用的端口，如果不设置的话默认为8080；

--rocketmq.config.namesrvAddr 为RocketMQ命名服务地址，若NameServer为集群则使用英文；分割

- 访问

浏览器访问 xxx.xxx.xxx.xxx:8080 进入控制台界面，效果如下

SpringBoot整合RocketMQ

在SpringBoot中整合RocketMQ主要用到 rocketmq-spring-boot-starter 组件，下面是详细整合过。

- 引入组件 rocketmq-spring-boot-starter 依赖

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.1.0</version>
</dependency>
```

- 修改application.yml，添加RocketMQ相关配置

```
rocketmq:
  name-server: 192.168.100.43:9876;192.168.100.44:9876
  producer:
    group: test-group
    send-message-timeout: 3000
```

如果是集群，多个name-server使用英文；分割。

- 编写消息生产者 MessageProduce

```
/**
 * Description:
 * rocketMQ消息发送方法
 * @author javadaily
 */
@Component
public class MessageProduce {
```

```
    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    /**
     * 发送消息
     * @param topic 主题
     * @param message 消息体
     */
    public void sendMessage(String topic,String message){
        this.rocketMQTemplate.convertAndSend(topic,message);
    }
}
```

使用RocketMQTemplate发送消息

- 编写消息消费者 MessageConsumer

```
@Slf4j
@Component
@RocketMQMessageListener(
        topic = "test-topic",
        consumerGroup = "test-group",
        selectorExpression = "*"
)
public class MessageConsumer implements RocketMQListener<String> {
    @Override
    public void onMessage(String message) {
        log.info("received message is {}", message);
    }
}
```

消费者只需要继承RocketMQListener类即可，主要关注实现类上的 @RocketMQMessageListener 注解，配置的 topic 和 consumerGroup 需要跟消息生产者的配置保持一致。

- 编写单元测试发送消息

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MessageProduceTest {
    @Autowired
    private MessageProduce messageProduce;

    @Test
    public void testSendMessage() {
        messageProduce.sendMessage("test-topic","Hello,JAVA日知录");
    }
}
```

- 测试

先启动springboot应用，再执行测试用例。