·°Somnus
你若安好,便是晴天

个人资料

若水三千-LOVE

访问： 1241770次
积分： 14363
等级： BLOG 7
排名： 第578名

原创： 295篇
转载： 62篇
译文： 1篇
评论： 250条

我的家园

GitHub

文章分类

RocketMQ（二）集群配置

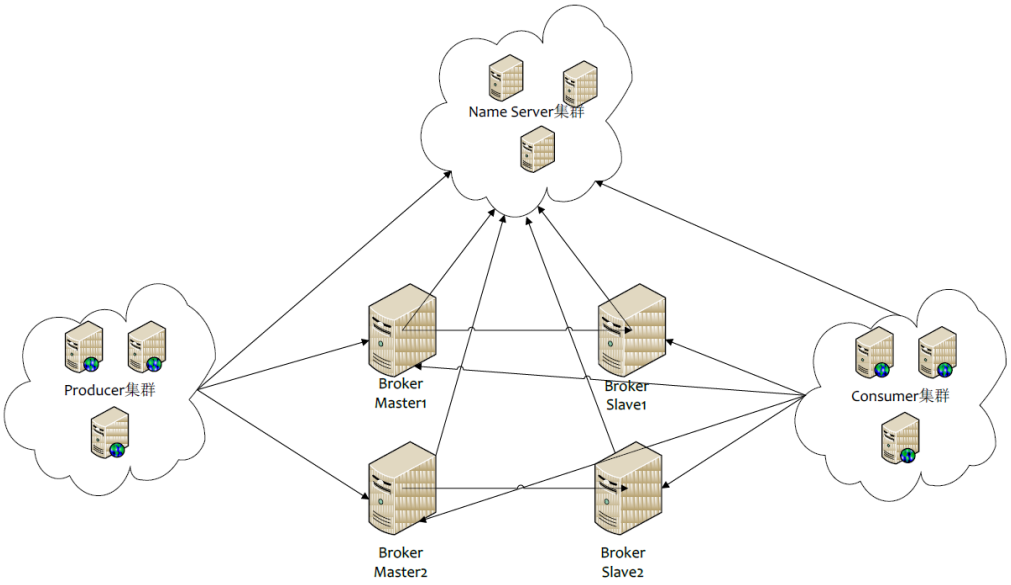
标签： rocketmq 集群 配置

2016-06-27 17:41 2463人阅读 评论(0) 收藏 举报

分类：

RocketMQ（6）

RocketMQ网络部署图



RocketMQ 网络部署特点

- Name Server 是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。
- Broker 部署相对复杂，Broker 分为Master 与Slave，一个Master 可以对应多个Slave，但是一个Slave 只能对应一个Master，Master 与Slave 的对应关系通过指定相同的BrokerName，不同的BrokerId来定 义，BrokerId为0 表示Master，非0 表示Slave。Master 也可以部署多个。每个Broker 与Name
- Producer 与Name Server 集群中的其中一个节点（随机选择）建立长连接，定期从Name Server 取Topic 路由信息，并向提供Topic 服务的Master 建立长连接，且定时向Master 发送心跳。Producer 完全无 状态，可集群部署。
- Consumer 与Name Server 集群中的其中一个节点（随机选择）建立长连接，定期从Name Server 取Topic 路由信息，并向提供Topic 服务的Master、Slave 建立长连接，且定时向Master、Slave 发送心跳。Consumer既可以从Master 订阅消息，也可以从Slave 订阅消息，订阅规则由Broker 配置决定。

Shiro (2)
restful (4)
dubbo (4)
RocketMQ (7)
消息中间件 (4)
redis (1)
支付 (3)

阅读排行

powerdesigner 15.0的 lic (87077)
Spring框架AOP源码剖析 (12841)
jQuery实现图片轮播效果 (10777)
Spring中@Autowired注解 (10120)
如何自定义注解Annotatic (9737)
java.lang.Excetion，java (9481)
事务四大特征：原子性， (8343)
Eclipse下编写C++程序— (8255)
Maven利用Profile构建不 (8247)
web.xml中load-on-startu (7616)

文章存档

2016年10月 (3)
2016年09月 (3)
2016年08月 (10)
2016年07月 (12)
2016年06月 (6)
展开

评论排行

jQuery实现图片轮播效果 (116)
powerdesigner 15.0的 lic (21)
让我们一起来剖析初级Ja (10)
Java基础之反射（二）： (4)
JQuery上传插件Uploadif (4)
最常用的Eclipse快捷键 (4)
在Spring中配置jdbcTem (3)
GitHub使用终结版 (3)
EL表达式 (3)
利用 pinyin4j 把汉字转化 (3)

Broker集群部署方式主要有以下几种：（ Slave 不可写，但可读 ）

单个Master

> 这种方式风险较大，一旦Broker 重启或者宕机时，会导致整个服务不可用，不建议线上环境使用。

多Master模式

一个集群无 Slave，全是 Master，例如 2 个 Master 或者 3 个 Master。

> 优点：配置简单，单个Master 宕机或重启维护对应用无影响，在磁盘配置为 RAID10 时，即使机器宕机不可恢复情况下，由于RAID10 磁盘非常可靠，消息也不会丢（异步刷盘丢失少量消息，同步刷盘一条不丢）。性能最高。
>
> 缺点：单台机器宕机期间，这台机器上未被消费的消息在机器恢复之前不可订阅，消息实时性会受到受到影响。

先启动 NameServer，例如机器 IP 为：192.168.1.101:9876

```
nohup sh mqnamesrv &
```

- 在机器 A，启动第一个 Master

```
nohup sh mqbroker -n 192.168.1.101:9876 -c $ROCKETMQ_HOME/conf/2m-noslave/broker-a.proper
```

- 在机器 B，启动第二个 Master

```
nohup sh mqbroker -n 192.168.1.102:9876 -c $ROCKETMQ_HOME/conf/2m-noslave/broker-b.proper
```

多Master多Slave模式，异步复制

每个 Master 配置一个 Slave，有多对Master-Slave，HA 采用异步复制方式，主备有短暂消息延迟，毫秒级。

> 优点：即使磁盘损坏，消息丢失的非常少，且消息实时性不会受影响，因为 Master 宕机后，消费者仍然可以从 Slave 消费，此过程对应用透明。不需要人工干预。性能同多 Master 模式几乎一样。
>
> 缺点：Master宕机，磁盘损坏情况，会丢失少量消息。

先启动两台服务器的NameServer，例如机器 IP 为：192.168.1.101:9876 和192.168.1.102:9876

```
nohup sh mqnamesrv 1>$ROCKETMQ_HOME/log/ng.log 2>$ROCKETMQ_HOME/log/ng-error.log &
```

- 在机器 A，启动第一个 Master

```
nohup sh mqbroker -n 192.168.1.101:9876 -c $ROCKETMQ_HOME/conf/2m-2s-async/broker-a.prope
```

- 在机器 B，启动第二个 Master

```
nohup sh mqbroker -n 192.168.1.102:9876 -c $ROCKETMQ_HOME/conf/2m-2s-async/broker-b.prope
```

- 在机器 C，启动第一个 Slave

```
nohup sh mqbroker -n 192.168.1.101:9876 -c $ROCKETMQ_HOME/conf/2m-2s-async/broker-a-s.pro
```

- 在机器 D，启动第二个 Slave

```
nohup sh mqbroker -n 192.168.1.102:9876 -c $ROCKETMQ_HOME/conf/2m-2s-async/broker-b-s.pro
```

多Master多Slave模式，同步双写

每个 Master 配置一个 Slave，有多对Master-Slave，HA 采用同步双写方式，主备都写成功，向应用返回成功。

> 优点：数据与服务都无单点，Master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高
>
> 缺点：性能比异步复制模式略低，大约低10%左右，发送单个消息的 RT 会略高。目前主宕机后，备机不能自动切换为主机，后续会支持自动切换功能。

先启动两台服务器的NameServer，例如机器 IP 为：192.168.1.101:9876 和192.168.1.102:9876

```
nohup sh mqnamesrv 1>$ROCKETMQ_HOME/log/ng.log 2>$ROCKETMQ_HOME/log/ng-error.log &
```

- 在机器 A，启动第一个 Master

```
nohup sh mqbroker -n 192.168.1.101:9876 -c $ROCKETMQ_HOME/conf/2m-2s-sync/broker-a.proper
```

- 在机器 B，启动第二个 Master

```
nohup sh mqbroker -n 192.168.1.102:9876 -c $ROCKETMQ_HOME/conf/2m-2s-sync/broker-b.proper
```

- 在机器 C，启动第一个 Slave

```
nohup sh mqbroker -n 192.168.1.101:9876 -c $ROCKETMQ_HOME/conf/2m-2s-sync/broker-a-s.prop
```

- 在机器 D，启动第二个 Slave

```
nohup sh mqbroker -n 192.168.1.102:9876 -c $ROCKETMQ_HOME/conf/2m-2s-sync/broker-b-s.prop
```

> 以上 Broker 与 Slave 配对是通过指定相同的brokerName 参数来配对，Master 的 BrokerId 必须是 0，Slave 的BrokerId 必须是大与 0 的数。另外一个 Master 下面可以挂载多个 Slave，同一 Master 下的多个
> Slave 通过指定不同的 BrokerId 来区分。

除此之外，nameserver也需要集群。

下面以配置一主一备(同步)，2个nameserver为例测试。

1、环境两台机器：

- 192.168.36.101 为主
- 192.168.36.102 为备

同时在2台机器个启动一个nameserver。安装RocketMq请参考：
http://blog.csdn.net/zhu_tianwei/article/details/40948447

2、修改配置

（1）创建目录

```
mkdir /usr/local/alibaba-rocketmq/log   #创建日志目录
mkdir -p /usr/local/alibaba-rocketmq/data/store/commitlog  #创建数据存储目录
```

更改日志目录

```
cd /usr/local/alibaba-rocketmq/conf
sed -i 's#${user.home}#${user.home}/alibaba-rocketmq#g' *.xml
```

（2）修改主配置

```
vi  ./conf/2m-2s-sync/broker-a.properties
```

```
#Broker所属哪个集群，默认【DefaultCluster】
brokerClusterName=DefaultCluster
#本机主机名
brokerName=broker-a
#BrokerId，必须是大等于0的整数，0表示Master，>0表示Slave，一个Master可以挂多个Slave，Maste
brokerId=0
#删除文件时间点，默认凌晨4点
deleteWhen=04
#文件保留时间，默认48小时
fileReservedTime=48
#Broker的角色 - ASYNC_MASTER 异步复制Master - SYNC_MASTER 同步双写Master - SLAVE
brokerRole=SYNC_MASTER
#刷盘方式 - ASYNC_FLUSH 异步刷盘 - SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH

#Name Server地址
namesrvAddr=192.168.1.101:9876;192.168.1.102:9876
#Broker对外服务的监听端口,默认【10911】
listenPort=10911

defaultTopicQueueNums=4
#是否允许Broker自动创建Topic，建议线下开启，线上关闭，默认【true】
autoCreateTopicEnable=true
#是否允许Broker自动创建订阅组，建议线下开启，线上关闭，默认【true】
autoCreateSubscriptionGroup=true
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=50000000
destroyMapedFileIntervalForcibly=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88

storePathRootDir=/usr/local/alibaba-rocketmq/data/store
storePathCommitLog=/usr/local/alibaba-rocketmq/data/store/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000

checkTransactionMessageEnable=false
sendMessage        s=128
pullMessage readBufferNums=128
```


（3）修改备配置

```
vi  ./conf/2m-2s-sync/broker-a-s.properties
```

```
#Broker所属哪个集群，默认【DefaultCluster】
brokerClusterName=DefaultCluster
#本机主机名
brokerName=broker-a
#BrokerId，必须是大等于0的整数，0表示Master，>0表示Slave，一个Master可以挂多个Slave，Maste
brokerId=1
#删除文件时间点，默认凌晨4点
deleteWhen=04
#文件保留时间，默认48小时
fileReservedTime=48
#Broker的角色 - ASYNC_MASTER 异步复制Master - SYNC_MASTER 同步双写Master - SLAVE
brokerRole=SLAVE
#刷盘方式 - ASYNC_FLUSH 异步刷盘 - SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH

#Name Server地址
namesrvAddr=192.168.1.101:9876;192.168.1.102:9876
#Broker对外服务的监听端口,默认【10911】
```

```
，建议线下开启，线上关闭，默认【true】

组，建议线下开启，线上关闭，默认【true】
true
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=50000000
destroyMapedFileIntervalForcibly=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88

storePathRootDir=/usr/local/alibaba-rocketmq/data/store
storePathCommitLog=/usr/local/alibaba-rocketmq/data/store/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000

checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
```

实例：

1.生产者Producer.java ,TransactionMQProducer使用

```
package com.somnus.rocketmq;

import java.util.concurrent.TimeUnit;

import com.alibaba.rocketmq.client.exception.MQClientException;
import com.alibaba.rocketmq.client.producer.LocalTransactionExecuter;
import com.alibaba.rocketmq.client.producer.LocalTransactionState;
import com.alibaba.rocketmq.client.producer.SendResult;
import com.alibaba.rocketmq.client.producer.TransactionCheckListener;
import com.alibaba.rocketmq.client.producer.TransactionMQProducer;
import com.alibaba.rocketmq.common.message.Message;
import com.alibaba.rocketmq.common.message.MessageExt;

public class Producer {

    public static void main(String[] args) throws MQClientException, InterruptedException
        /**
         * 一个应用创建一个Producer，由应用来维护此对象，可以设置为全局对象或者单例<br>
         * 注意：ProducerGroupName需要由应用来保证唯一,一类Producer集合的名称，这类Produc
         * 且发送逻辑一致<br>
         * ProducerGroup这个概念发送普通的消息时，作用不大，但是发送分布式事务消息时，比较
         * 因为服务器会回查这个Group下的任意一个Producer
         */
        final TransactionMQProducer producer = new TransactionMQProducer("ProducerGroupNa
        // nameserver服务
        producer.setNamesrvAddr("172.16.235.77:9876;172.16.235.78:9876");
        producer.setInstanceName("Producer");

        /**
         * Producer对象在使用之前必须要调用start初始化，初始化一次即可<br>
         * 注意：切记不可以在每次发送消息时，都调用start方法
         */
        producer.start();
        // 服务器回调Producer，检查本地事务分支成功还是失败
        producer.setTransactionCheckListener(new TransactionCheckListener() {

            public LocalTransactionState checkLocalTransactionState(
                    MessageExt msg) {
                System.out.println("checkLocalTransactionState --" + new String(msg.getBo
                return LocalTransactionState.COMMIT_MESSAGE;
```

```
            }
        });

        /**
         * 下面这段代码表明一个Producer对象可以发送多个topic，多个tag的消息。
         * 注意：send方法是同步调用，只要不抛异常就标识成功。但是发送成功也可会有多种状态
         * 例如消息写入Master成功，但是Slave不成功，这种情况消息属于成功，但是对于个别应用
         * 需要对这种情况做处理。另外，消息可能会存在发送失败的情况，失败重试由应用来处理
         */

        for (int i = 0; i < 10; i++) {
            try {
                {
                    Message msg = new Message("TopicTest1", // topic
                            "TagA",                         // tag
                            "OrderID001",                   // key消息关键词，多个Key用KE
                            ("Hello MetaQA").getBytes());   // body
                    SendResult sendResult = producer.sendMessageInTransaction(
                            msg, new LocalTransactionExecuter(){
                                public LocalTransactionState executeLocalTransactionBranch
                                    System.out.println("executeLocalTransactionBranch--ms
                                    System.out.println("executeLocalTransactionBranch--ar
                                    return LocalTransactionState.COMMIT_MESSAGE;
                                }
                            },
                            "$$$");
                    System.out.println(sendResult);
                }

                {
                    Message msg = new Message("TopicTest2", // topic
                            "TagB",                         // tag
                            "OrderID0034",                  // key 消息关键词，多个Key用K
                            ("Hello MetaQB").getBytes());   // body
                    SendResult sendResult = producer.sendMessageInTransaction(
                            msg, new LocalTransactionExecuter(){
                                public LocalTransactionState executeLocalTransactionBranch
                                    System.out.println("executeLocalTransactionBranch--ms
                                    System.out.println("executeLocalTransactionBranch--ar
                                    return LocalTransactionState.COMMIT_MESSAGE;
                                }
                            },
                            "$$$");
                    System.out.println(sendResult);
                }

                {
                    Message msg = new Message("TopicTest3", // topic
                            "TagC",                         // tag
                            "OrderID061",                   // key
                            ("Hello MetaQC").getBytes());   // body
                    SendResult sendResult = producer.sendMessageInTransaction(
                            msg, new LocalTransactionExecuter(){
                                public LocalTransactionState executeLocalTransactionBranch
                                    System.out.println("executeLocalTransactionBranch--ms
                                    System.out.println("executeLocalTransactionBranch--ar
                                    return LocalTransactionState.COMMIT_MESSAGE;
                                }
                            },
                            "$$$");
                    System.out.println(sendResult);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            TimeUnit.MILLISECONDS.sleep(1000);
        }

        /**
         * 应用退出时，要调用shutdown来清理资源，关闭网络连接，从MetaQ服务器上注销自己
         * 注意：我们建议应用在JBOSS、Tomcat等容器的退出钩子里调用shutdown方法
```

```
         */
        // producer.shutdown();
        Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
            public void run() {
                producer.shutdown();
            }
        }));
        System.exit(0);
    } // 执行本地事务，由客户端回调

}
```

2、消费者Consumer.java ，采用主动拉取方式消费。

```
package com.somnus.rocketmq;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.alibaba.rocketmq.client.consumer.DefaultMQPullConsumer;
import com.alibaba.rocketmq.client.consumer.PullResult;
import com.alibaba.rocketmq.client.exception.MQClientException;
import com.alibaba.rocketmq.common.message.MessageExt;
import com.alibaba.rocketmq.common.message.MessageQueue;

public class Consumer {

    // Java缓存
    private static final Map<MessageQueue, Long> offseTable = new HashMap<MessageQueue, L

    /**
     * 主动拉取方式消费
     *
     * @throws MQClientException
     */
    public static void main(String[] args) throws MQClientException {
        /**
         * 一个应用创建一个Consumer，由应用来维护此对象，可以设置为全局对象或者单例<br>
         * 注意：ConsumerGroupName需要由应用来保证唯一 ,最好使用服务的包名区分同一服务,一
         * 这类Consumer通常消费一类消息，且消费逻辑一致
         * PullConsumer：Consumer的一种，应用通常主动调用Consumer的拉取消息方法从Broker拉
         */
        DefaultMQPullConsumer consumer = new DefaultMQPullConsumer("ConsumerGroupName");
        // //nameserver服务
        consumer.setNamesrvAddr("172.16.235.77:9876;172.16.235.78:9876");
        consumer.setInstanceName("Consumber");
        consumer.start();

        // 拉取订阅主题的队列，默认队列大小是4
        Set<MessageQueue> mqs = consumer.fetchSubscribeMessageQueues("TopicTest1");
        for (MessageQueue mq : mqs) {
            System.out.println("Consume from the queue: " + mq);
            SINGLE_MQ: while (true) {
                try {
                    PullResult pullResult = consumer.pullBlockIfNotFound(mq, null, getMes
                    List<MessageExt> list = pullResult.getMsgFoundList();
                    if (list != null && list.size() < 100) {
                        for (MessageExt msg : list) {
                            System.out.println(new String(msg.getBody()));
                        }
                    }
                    System.out.println(pullResult.getNextBeginOffset());
                    putMessageQueueOffset(mq, pullResult.getNextBeginOffset());
                    switch (pullResult.getPullStatus()) {
                        case FOUND:
                            break;
                        case NO_MATCHED_MSG:
                            break;
```

```
                        case NO_NEW_MSG:
                            break SINGLE_MQ;
                        case OFFSET_ILLEGAL:
                            break;
                        default:
                            break;
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        consumer.shutdown();
    }

    private static void putMessageQueueOffset(MessageQueue mq, long offset) {
        offseTable.put(mq, offset);
    }

    private static long getMessageQueueOffset(MessageQueue mq) {
        Long offset = offseTable.get(mq);
        if (offset != null) {
            System.out.println(offset);
            return offset;
        }
        return 0;
    }
}
```


上一篇 RocketMQ（一）介绍
下一篇 RocketMQ（三）原理与实践

我的同类文章

RocketMQ（6）

- RocketMQ延时消息的使用... 2016-08-10 阅读 176
- RocketMQ与Kafka对比（1... 2016-08-09 阅读 85
- RocketMQ（五）性能测试... 2016-07-08 阅读 3152
- RocketMQ（四）特性 2016-07-01 阅读 3078
- RocketMQ（三）原理与实践 2016-06-28 阅读 3862
- RocketMQ（一）介绍 2016-06-24 阅读 3360

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场