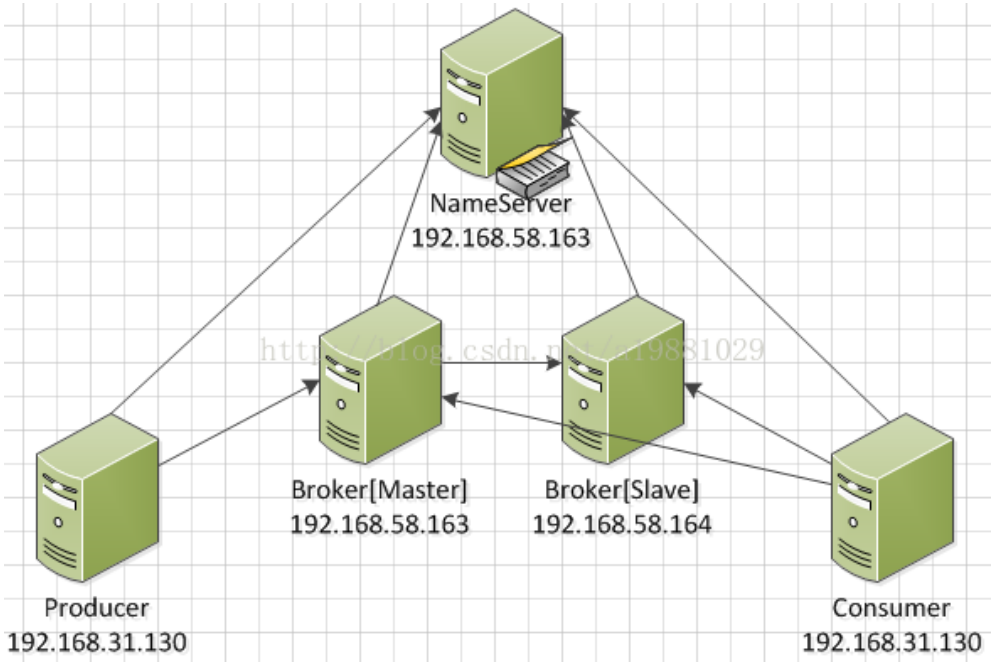
RocketMQ单机支持1万以上的持久化队列，前提是足够的内存、硬盘空间，过期数据数据删除（RocketMQ中的消息队列长度不是无限的，只是足够大的内存+数据定时删除）

RocketMQ版本：3.1.4



一，部署**NameServer**：

1，安装JDK并设置JAVA_HOME环境变量（启动脚本依赖JAVA_HOME环境变量）

2，cd /alibaba-rocketmq/bin进入RocketMQ的bin目录

2，调用nohup sh mqnamesrv &启动NameServer

报错如下：

```
[plain]
: command not found
: command not found
mqnamesrv: line 35: syntax error: unexpected end of file
```

在bin目录下调用dos2unix *将所有文件转化为unix格式，再次调用nohup sh mqnamesrv &

报错如下：

```
[plain]
/home/hadoop/alibaba-rocketmq
Invalid initial heap size: -Xms4g
The specified size exceeds the maximum representable size.
Could not create the Java virtual machine.
```

由于安装的JDK版本为32位，4g超过了JDK所支持的最大内存，不过32位JDK也无法发挥出RocketMQ的优势，换成64位JDK

这次启动成功

```
[plain]
[hadoop@hadoop bin]$ nohup sh mqnamesrv &
[1] 17676
[hadoop@hadoop bin]$ nohup: appending output to “nohup.out”
```

```
[hadoop@hadoop bin]$ cat nohup.out
The Name Server boot success.
[hadoop@hadoop bin]$ jps
17682 NamesrvStartup
17800 Jps
```

NameServer监听端口：9876

```
[java]
nettyServerConfig.setListenPort(9876);
```

如果服务器内存不够，可以修改runserver.sh脚本（mqnamesrv文件中通过runserver.sh脚本调用Name Server的主函数com.alibaba.rocketmq.namesrv.NamesrvStartup启动Name Server）中的JAVA_OPT_1参数

```
[plain]
JAVA_OPT_1="-server -Xms4g -Xmx4g -Xmn2g -XX:PermSize=128m -XX:MaxPermSize=320m"
```

二，部署**Broker**：消息中转角色，负责存储消息，转发消息

Broker集群有多种配置方式：

1，单Master

优点：除了配置简单没什么优点

缺点：不可靠，该机器重启或宕机，将导致整个服务不可用

2，多Master

优点：配置简单，性能最高

缺点：可能会有少量消息丢失（配置相关），单台机器重启或宕机期间，该机器下未被消费的消息在机器恢复前不可订阅，影响消息实时性

3，多Master多Slave，每个Master配一个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：性能同多Master几乎一样，实时性高，主备间切换对应用透明，不需人工干预

缺点：Master宕机或磁盘损坏时会有少量消息丢失

4，多Master多Slave，每个Master配一个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：服务可用性与数据可用性非常高

缺点：性能比异步HA略低，当前版本主宕备不能自动切换为主

Master和Slave的配置文件参考conf目录下的配置文件

Master与Slave通过指定相同的brokerName参数来配对，Master的BrokerId必须是0，Slave的BrokerId必须是大于0的数

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

部署一Master一Slave，HA采用异步复制方式：

Master：

```
[plain]
[hadoop@hadoop bin]$ nohup sh mqbroker -n "192.168.58.163:9876" -c ../conf/2m-2s-
async/broker-a.properties &
[2] 25493
```

```
[hadoop@hadoop bin]$ nohup: appending output to “nohup.out”

[hadoop@hadoop bin]$ cat nohup.out
load config properties file OK, ../conf/2m-2s-async/broker-a.properties
The broker[broker-
a, 192.168.58.163:10911] boot success. and name server is 192.168.58.163:9876
[hadoop@hadoop bin]$ jps
25500 BrokerStartup
25545 Jps
17682 NamesrvStartup
```

Slave：

```
[plain]
[hadoop@hadoop bin]$ nohup sh mqbroker -n "192.168.58.163:9876" -c ../conf/2m-2s-
async/broker-a-s.properties &
[1] 1974
[hadoop@hadoop bin]$ nohup: appending output to “nohup.out”

[hadoop@hadoop bin]$ cat nohup.out
load config properties file OK, ../conf/2m-2s-async/broker-a-s.properties
The broker[broker-
a, 192.168.58.164:10911] boot success. and name server is 192.168.58.163:9876
[hadoop@hadoop bin]$ jps
2071 Jps
1981 BrokerStartup
```

Broker监听端口：10911

```
[java]
nettyServerConfig.setListenPort(10911);
```

如果服务器内存不够，可以修改runbroker.sh脚本（mqbroker文件中通过runbroker.sh脚本调用Broker的主函数com.alibaba.rocketmq.broker.BrokerStartup启动Broker）的JAVA_OPT_1参数

```
[plain]
JAVA_OPT_1="-server -Xms4g -Xmx4g -Xmn2g -XX:PermSize=128m -XX:MaxPermSize=320m"
```

三，Producer

必须要设置Name Server地址

```
[java]
package com.sean;

import com.alibaba.rocketmq.client.producer.DefaultMQProducer;
import com.alibaba.rocketmq.client.producer.SendResult;
import com.alibaba.rocketmq.common.message.Message;

public class Producer {
    public static void main(String[] args){
        DefaultMQProducer producer = new DefaultMQProducer("Producer");
        producer.setNamesrvAddr("192.168.58.163:9876");
        try {
            producer.start();

            Message msg = new Message("PushTopic",
                    "push",
                    "1",
                    "Just for test.".getBytes());

```

```
            SendResult result = producer.send(msg);
            System.out.println("id:" + result.getMsgId() +
                    " result:" + result.getSendStatus());

            msg = new Message("PushTopic",
                    "push",
                    "2",
                    "Just for test.".getBytes());

            result = producer.send(msg);
            System.out.println("id:" + result.getMsgId() +
                    " result:" + result.getSendStatus());

            msg = new Message("PullTopic",
                    "pull",
                    "1",
                    "Just for test.".getBytes());

            result = producer.send(msg);
            System.out.println("id:" + result.getMsgId() +
                    " result:" + result.getSendStatus());
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            producer.shutdown();
        }
    }
}
```

四，**Consumer**

必须要设置Name Server地址

```
[java]
package com.sean;

import java.util.List;

import com.alibaba.rocketmq.client.consumer.DefaultMQPushConsumer;
import com.alibaba.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import com.alibaba.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import com.alibaba.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import com.alibaba.rocketmq.common.consumer.ConsumeFromWhere;
import com.alibaba.rocketmq.common.message.Message;
import com.alibaba.rocketmq.common.message.MessageExt;

public class Consumer {
    public static void main(String[] args){
        DefaultMQPushConsumer consumer =
                new DefaultMQPushConsumer("PushConsumer");
        consumer.setNamesrvAddr("192.168.58.163:9876");
        try {
            //订阅PushTopic下Tag为push的消息
            consumer.subscribe("PushTopic", "push");
            //程序第一次启动从消息队列头取数据
            consumer.setConsumeFromWhere(
                    ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
            consumer.registerMessageListener(
                new MessageListenerConcurrently() {
                    public ConsumeConcurrentlyStatus consumeMessage(
                            List<MessageExt> list,
                            ConsumeConcurrentlyContext Context) {
                        Message msg = list.get(0);
                        System.out.println(msg.toString());
                        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
                    }
                }
```

```
            );
            consumer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

先运行Consumer，然后运行Producer

Producer运行结果：

```
[plain]
id:C0A83AA300002A9F00000000000009EA result:SEND_OK
id:C0A83AA300002A9F0000000000000A77 result:SEND_OK
id:C0A83AA300002A9F0000000000000B04 result:SEND_OK
```

Consumer运行结果：

```
[plain]
MessageExt [queueId=1, storeSize=141, queueOffset=6, sysFlag=0, bornTimestamp=1403765668792,
{TAGS=push, KEYS=2, WAIT=true, MAX_OFFSET=7, MIN_OFFSET=0}, body=14]]
MessageExt [queueId=0, storeSize=141, queueOffset=6, sysFlag=0, bornTimestamp=1403765668698,
{TAGS=push, KEYS=1, WAIT=true, MAX_OFFSET=7, MIN_OFFSET=0}, body=14]]
```