RocketMQ开发环境搭建

1. 安装虚拟机

为了不影响主机的运行，将RocketMQ的开发环境放置在虚拟机中。

1.1 安装VirtualBox

首先需要在BIOS中开启虚拟化功能。安装文件[下载地址](http://cxyroad.com/ "https://www.virtualbox.org/wiki/Downloads")

1.2 安装Vagrant

这是一款用于创建和部署虚拟化环境的工具。**[下载地址](http://cxyroad.com/ "https://developer.hashicorp.com/vagrant/install?product_intent=vagrant")**

1.3 从Vagrant仓库中下载系统镜像并安装

Vagrant[系统镜像仓库地址](http://cxyroad.com/ "https://app.vagrantup.com/boxes/search")。
从仓库中选择所需的系统。本次环境搭建使用Ubuntu系统（ubuntu/focal64）。

1.3.1 修改系统镜像存储位置

Vagrant下载的镜像默认会存储在C盘，在开始创建虚拟机前，如果需要修改系统镜像的存储位置，可以通过设置环境变量VAGRANT_HOME来进行更改，例

如

VAGRANT_HOME=E:\vagrant_images.vagrant.d

1.3.2 创建虚拟机

打开windows命令行窗口，将路径切换到虚拟机安装路径，例如E:\virtual-machines\ubuntu。输入初始化虚拟机命令

```
vagrant init ubuntu/focal64
```

初始化完成后，输入命令启动虚拟机，创建出的系统的用户名和密码都是vagrant

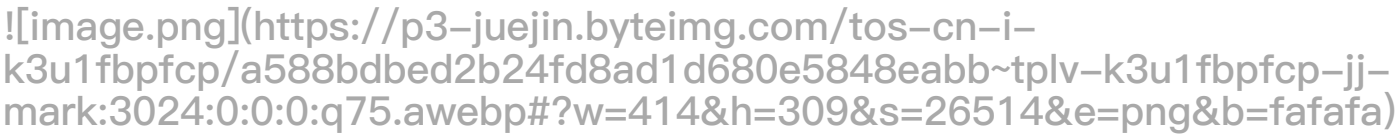
```
vagrant up
```

1.3.3 连接虚拟机

使用下面的命令可以直接进入虚拟机

```
vagrant ssh
```

为了方便开发以及通过其他ssh客户端连接虚拟机，需要给创建出的虚拟机配置固定的IP地址。查看virtual box仅主机模式的网络适配器，配置的IP地址与这个地址在一个网段即可，例如192.168.56.12



修改步骤[1.3.2 创建虚拟机](http://cxyroad.com/ "#title_132")切换到的系统路径下的Vagrantfile文件，去除 `config.vm.network "private_network"`所在行的#注释，将后面的IP地址改为192.168.56.12

config.vm.network "private_network", ip: "192.168.56.12"

使用`vagrant reload`命令重启虚拟机，启动完成后可通过固定IP访问虚拟机。

如果ssh客户端使用固定IP连接失败，提示秘钥问题，那么需要在连接时指定和Vagrantfile文件位于同目录下的私钥文件：`.vagrant\machines\default\virtualbox\private_key`，不同客户端指定方式不同。

2. 安装RocketMQ

以下步骤改编自[官网快速入门](http://cxyroad.com/ "https://rocketmq.apache.org/zh/docs/quickStart/01quickstart/")。

2.1 下载

首先下载jdk并解压，完成后配置环境变量:`vim ~/.bashrc`

```
export JAVA_HOME=/home/vagrant/programs/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

刷新使配置生效

```
source ~/.bashrc

```

接着下载rocketmq

```
curl
https://dist.apache.org/repos/dist/release/rocketmq/5.2.0/rocketmq-
all-5.2.0-bin-release.zip --output rocketmq.zip
unzip rocketmq.zip
cd rocketmq

```

2.2 启动NameServer

如果启动过程中提示内存不足，根据机器配置修改内存大小：`vim
bin/runserver.sh`

```
...
choose_gc_options()
{
...
JAVA_OPT="${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
...
}
...

```

```
nohup sh bin/mqnamesrv &

```

2.3 启动Broker+Proxy

Proxy在示例代码中会被使用，如果使用rocketmq-client库编写程序进行连接，可以不启用。此步骤中需要修改broker的监听IP地址为本机，即192.168.56.12，否则生产者无法通过虚拟机IP发送消息。

修改conf/broker.conf文件，添加如下配置

brokerIP1=192.168.56.12

如果启动过程中提示内存不足，根据机器配置修改内存大小：`vim bin/runbroker.sh`

```
...
JAVA_OPT="${JAVA_OPT} -server -Xms512m -Xmx512m"
...
```

```
nohup sh bin/mqbroker -n localhost:9876 -c conf/broker.conf --enable-proxy &
```

2.4 创建示例代码使用的topic

```
sh bin/mqadmin updatetopic -n localhost:9876 -t TestTopic -c DefaultCluster
```

3. 使用程序收发消息

3.1 创建maven项目，pom文件内容如下

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>rocketmqDemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>


    <dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-slf4j-impl</artifactId>
            <version>2.19.0</version>
        </dependency>
    </dependencies>
</project>
```

日志配置文件log4j2.xml的内容如下

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration xmlns="http://logging.apache.org/log4j/2.0/config"
status="INFO">
    <Appenders>
```

```
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t]
%-5level %c{3}:%M:%L - %msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="INFO">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

3.2 发送端代码

```
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientConfigurationBuilder;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.message.Message;
import org.apache.rocketmq.client.apis.producer.Producer;
import org.apache.rocketmq.client.apis.producer.SendReceipt;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;

public class ProducerExample {
    private static final Logger logger =
LoggerFactory.getLogger(ProducerExample.class);

    public static void main(String[] args) throws ClientException,
IOException {
        // 接入点地址，需要设置成Proxy的地址和端口列表，一般是
xxx:8081;xxx:8081。
        String endpoint = "192.168.56.12:8081";
        // 消息发送的目标Topic名称，需要提前创建。
        String topic = "TestTopic";
        ClientServiceProvider provider =
ClientServiceProvider.loadService();
        ClientConfigurationBuilder builder =
ClientConfiguration.newBuilder().setEndpoints(endpoint);
        ClientConfiguration configuration = builder.build();
        // 初始化Producer时需要设置通信配置以及预绑定的Topic。
```

```
        Producer producer = provider.newProducerBuilder()
                .setTopics(topic)
                .setClientConfiguration(configuration)
                .build();

        // 普通消息发送。
        Message message = provider.newMessageBuilder()
                .setTopic(topic)
                // 设置消息索引键，可根据关键字精确查找某条消息。
                .setKeys("messageKey")
                // 设置消息Tag，用于消费端根据指定Tag过滤消息。
                .setTag("messageTag")
                // 消息体。
                .setBody("messageBody".getBytes())
                .build();

        try {
            // 发送消息，需要发送结果，并捕获失败等异常。
            SendReceipt sendReceipt = producer.send(message);
            logger.info("Send message successfully, messageId={}",
sendReceipt.getMessageId());
        } catch (ClientException e) {
            logger.error("Failed to send message", e);
        }
        producer.close();
    }
}

```

3.3 消费端代码

```
import org.apache.rocketmq.client.apis.ClientConfiguration;
import org.apache.rocketmq.client.apis.ClientException;
import org.apache.rocketmq.client.apis.ClientServiceProvider;
import org.apache.rocketmq.client.apis.consumer.ConsumeResult;
import org.apache.rocketmq.client.apis.consumer.FilterExpression;
import org.apache.rocketmq.client.apis.consumer.FilterExpressionType;
import org.apache.rocketmq.client.apis.consumer.PushConsumer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.util.Collections;
```

```
public class PushConsumerExample {
    private static final Logger logger =
LoggerFactory.getLogger(PushConsumerExample.class);

    public static void main(String[] args) throws ClientException,
IOException, InterruptedException {
        final ClientServiceProvider provider =
ClientServiceProvider.loadService();
        // 接入点地址，需要设置成Proxy的地址和端口列表，一般是
xxx:8081;xxx:8081。
        String endpoints = "192.168.56.12:8081";
        ClientConfiguration clientConfiguration =
ClientConfiguration.newBuilder()
                .setEndpoints(endpoints)
                .build();
        // 订阅消息的过滤规则，表示订阅所有Tag的消息。
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag,
FilterExpressionType.TAG);
        // 为消费者指定所属的消费者分组，Group需要提前创建。
        String consumerGroup = "YourConsumerGroup";
        // 指定需要订阅哪个目标Topic，Topic需要提前创建。
        String topic = "TestTopic";
        // 初始化PushConsumer，需要绑定消费者分组ConsumerGroup、通信
参数以及订阅关系。
        PushConsumer pushConsumer =
provider.newPushConsumerBuilder()
                .setClientConfiguration(clientConfiguration)
                // 设置消费者分组。
                .setConsumerGroup(consumerGroup)
                // 设置预绑定的订阅关系。
                .setSubscriptionExpressions(Collections.singletonMap(topic,
filterExpression))
                // 设置消费监听器。
                .setMessageListener(messageView -> {
                    // 处理消息并返回消费结果。
                    logger.info("Consume message successfully,
messageId={}", messageView.getMessageId());
                    return ConsumeResult.SUCCESS;
                })
                .build();
        Thread.sleep(Long.MAX_VALUE);
        // 如果不需要再使用 PushConsumer，可关闭该实例。
         pushConsumer.close();
    }
}

```