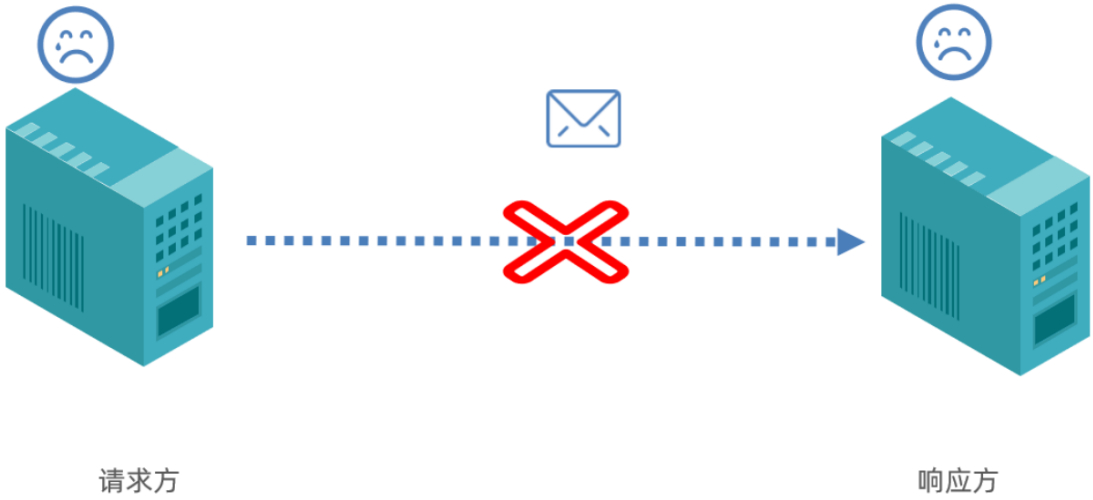
RocketMQ

1. MQ简介

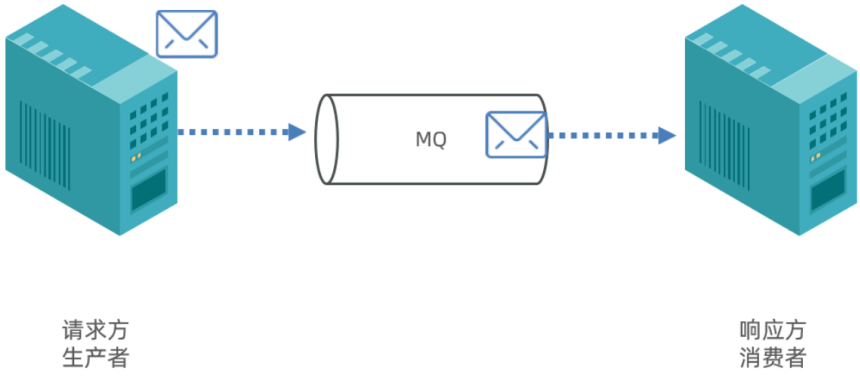
1.1 项目工程弊端



1.2 MQ简介

1. MQ（Message Queue）消息队列，是一种用来保存消息数据的队列.
    1. 队列：数据结构的一种，特征为 “先进先出”

        MQ全称 Message Queue（消息队列），是在消息的传输过程中保存消息的容器。多用于分布式系统之间进行通信。



    2. 何为消息
        1. 服务器间的业务请求
            1. 原始架构：
                1. 服务器中的A功能需要调用B、C模块才能完成
            2. 微服务架构：
                1. 服务器A向服务器B发送要执行的操作（视为消息）
                2. 服务器A向服务器C发送要执行的操作（视为消息）
2. 小节:MQ概念

1.3 MQ作用

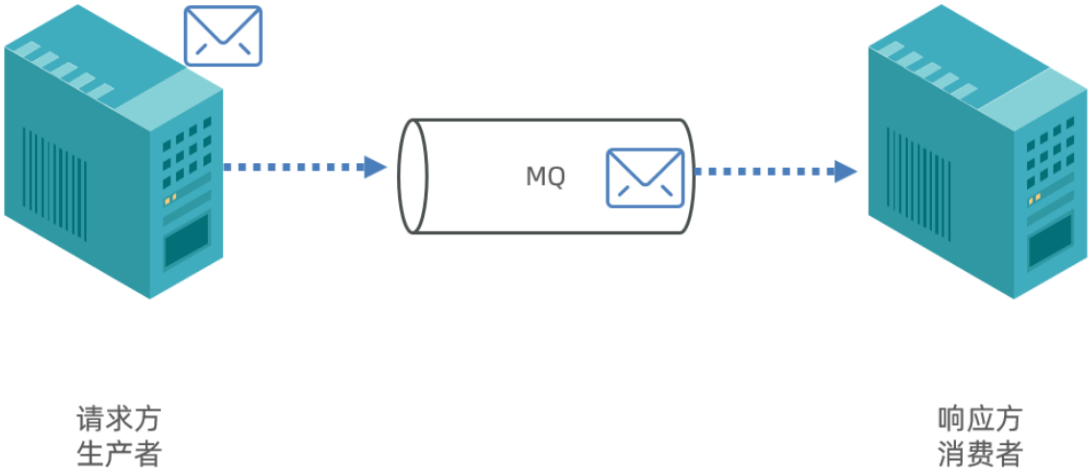
优势：

- 应用解耦
- 异步提速
- 削峰填谷

劣势：

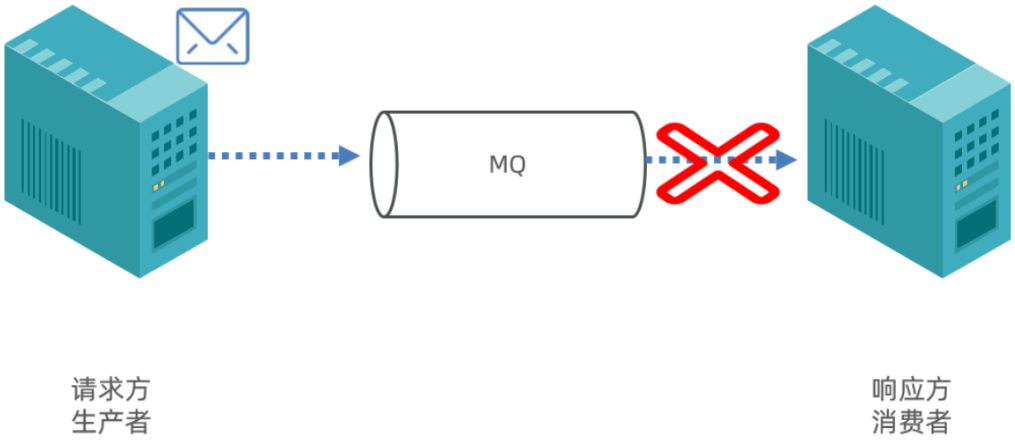
- 系统可用性降低
- 系统复杂度提高
- 一致性问题

1.4 MQ基本工作模式

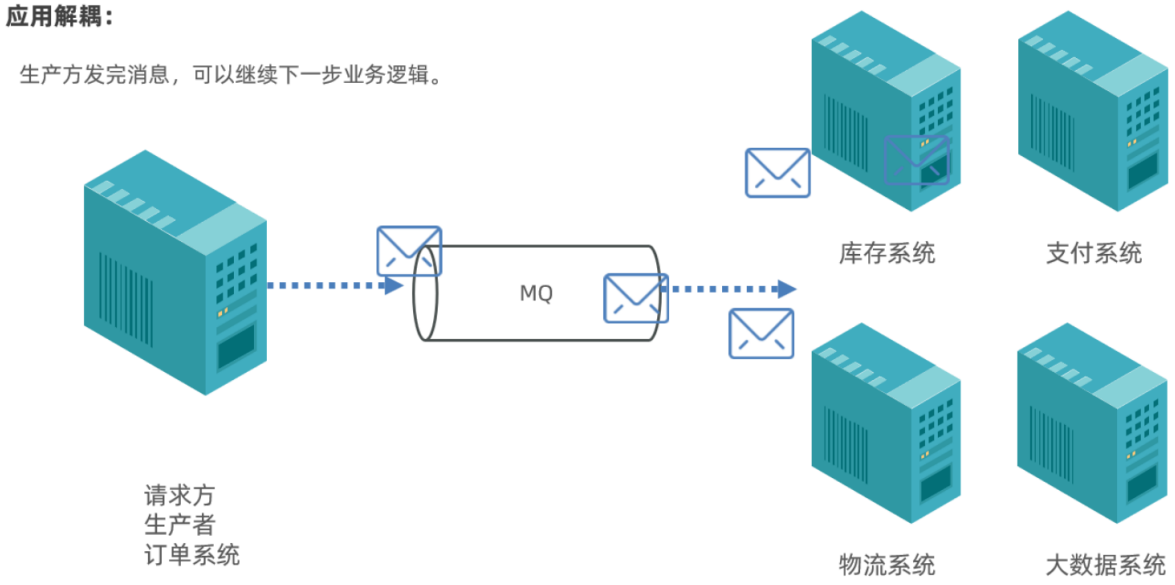


应用解耦：（异步消息发送）

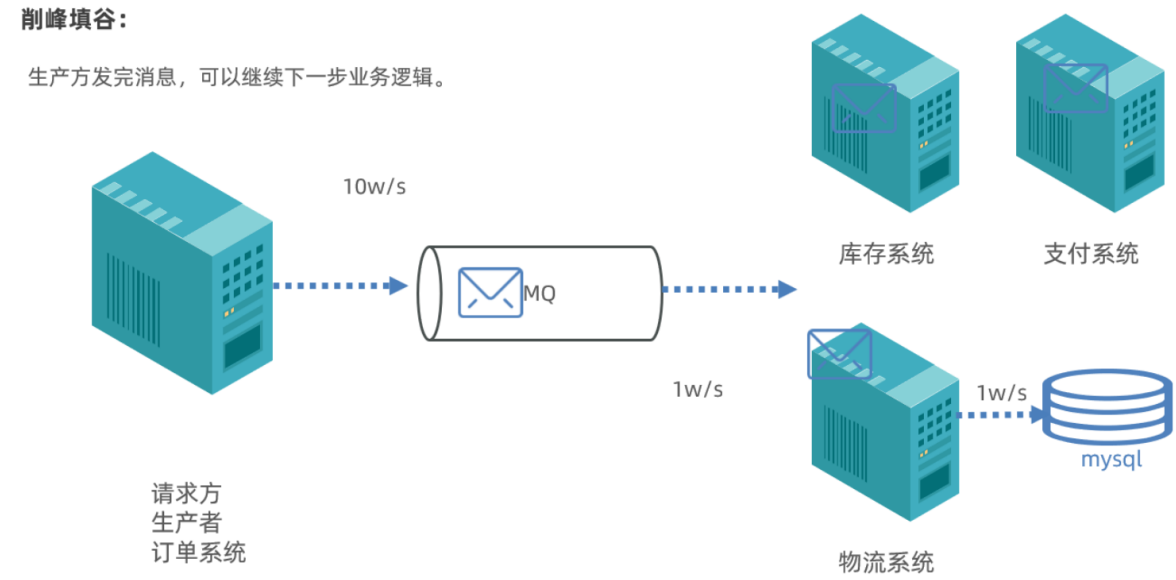
消费方存活与否不影响生产方



系统的耦合性越高，容错性就越低，可维护性就越低。



流量削锋:（异步消息发送）



1.5 MQ优缺点分析

优点（作用）：

1. 应用解耦
2. 快速应用变更维护
3. 流量削锋

缺点:

1. 系统可用性降低
2. 系统复杂度提高
3. 异步消息机制
    1. 消息顺序性

2. 消息丢失
3. 消息一致性
4. 消息重复使用

1.6 MQ产品介绍

1. ActiveMQ

   java语言实现，万级数据吞吐量，处理速度ms级，主从架构，**成熟度高**

2. RabbitMQ

   erlang语言实现，万级数据吞吐量，**处理速度us级**，主从架构，

3. RocketMQ

   java语言实现，**十万级**数据吞吐量，处理速度ms级，分布式架构，功能强大，扩展性强

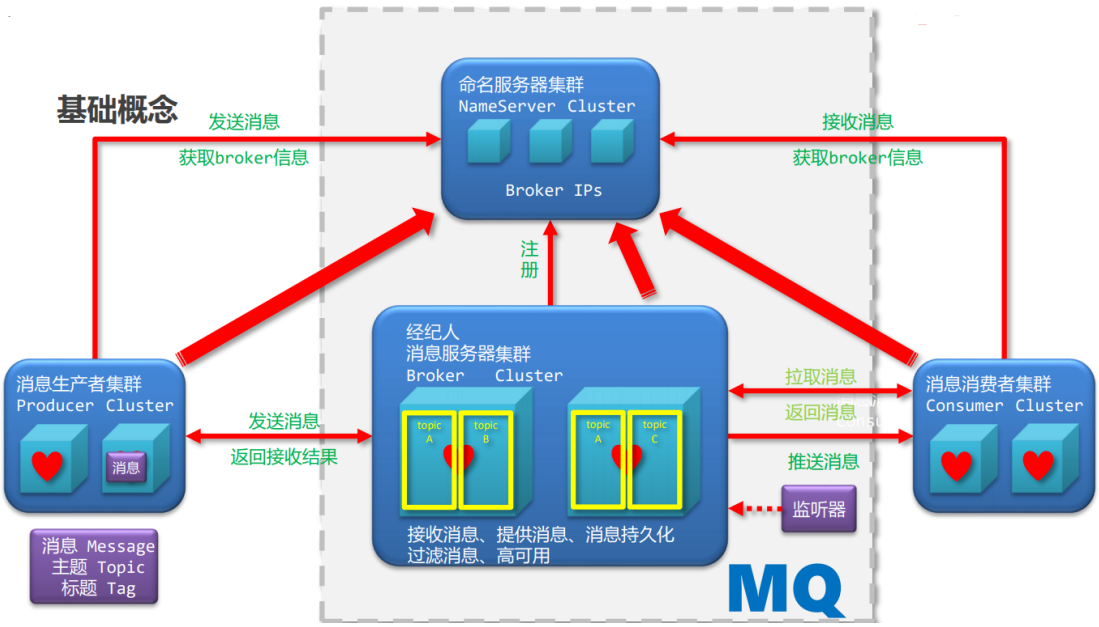
4. kafka

   scala语言实现，**十万级**数据吞吐量，处理速度ms级，分布式架构，功能较少，应用于大数据较多

RocketMQ

1. RocketMQ是阿里开源的一款非常优秀中间件产品，脱胎于阿里的另一款队列技术MetaQ，后捐赠给**Apache**基金会 作为一款孵化技术，仅仅经历了一年多的时间就成为Apache基金会的顶级项目。并且它现在已经在阿里内部被广泛 的应用，并且经受住了多次双十一的这种极致场景的压力（2017年的双十一，RocketMQ流转的消息量达到了万亿 级，峰值TPS达到5600万）
2. 解决所有缺点

2. 环境搭建



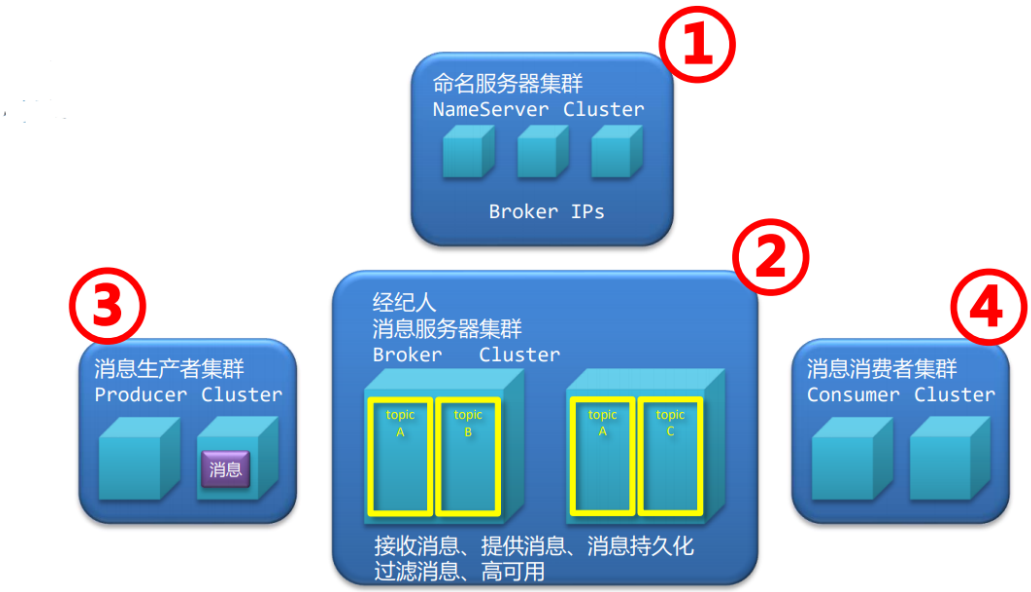
2.1 基础概念

1. 生产者
2. 消费者
3. 消息服务器
4. 命名服务器
5. 消息

    1. 主题
    2. 标签
6. 心跳
7. 监听器
8. 拉取消费、推动消费
9. 注册

2.2 安装

1. 命名服务器
2. 消息服务器



2.3 下载

● https://www.apache.org/

2.4 linux安装过程

1. 步骤1：安装JDK（1.8）
2. 步骤2：上传压缩包（zip）

```
yum -y install lrzsz
```

```
rz
```

3. 步骤3：解压缩

```
unzip rocketmq-all-4.5.2-bin-release.zip
```

4. 步骤4：修改目录名称

```
mv rocketmq-all-4.5.2-bin-release rocketmq
```

启动服务器

1. 步骤1：启动命名服务器（bin目录下）

```
sh mqnamesrv
```

2. 步骤2：启动消息服务器（bin目录下）

```
sh mqbroker -n localhost:9876
```

修改runbroker.sh文件中有关内存的配置（调整的与当前虚拟机内存匹配即可，推荐256m-128m）

测试服务器环境

1. 步骤1：配置命名服务器地址

```
export NAMESRV_ADDR=localhost:9876
```

2. 步骤2：启动生产者程序客户端（bin目录下）

```
sh tools.sh org.apache.rocketmq.example.quickstart.Producer
```

启动后产生大量日志信息（注意该信息是测试程序中自带的，不具有通用性，仅供学习查阅参考）

3. 步骤3：启动消费者程序客户端（bin目录下）

```
sh tools.sh org.apache.rocketmq.example.quickstart.Consumer
```

启动后产生大量日志信息

2.5 windows安装

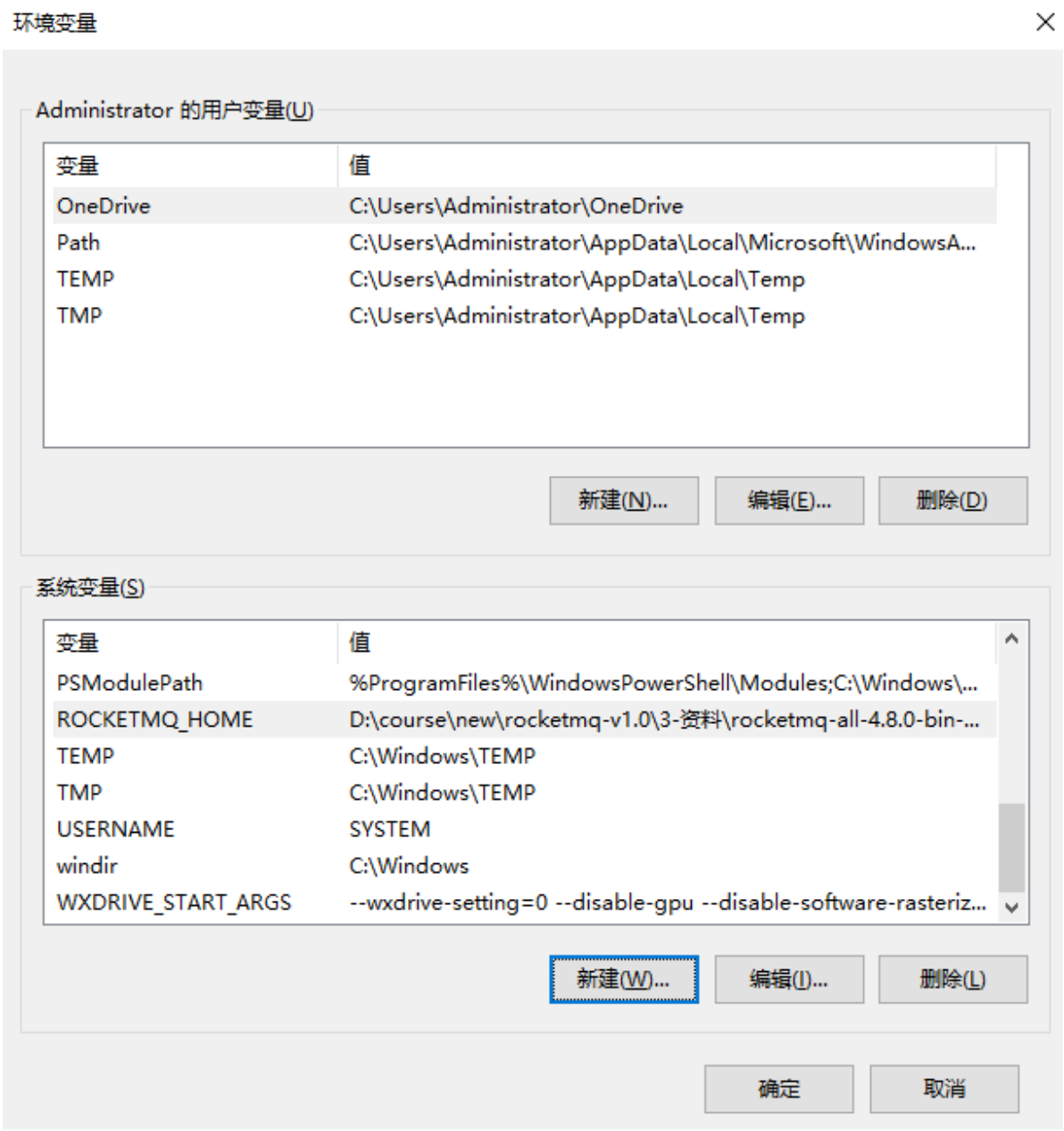
2.5.1系统环境变量配置

1、右键我的电脑-->属性

2、系统属性--环境变量

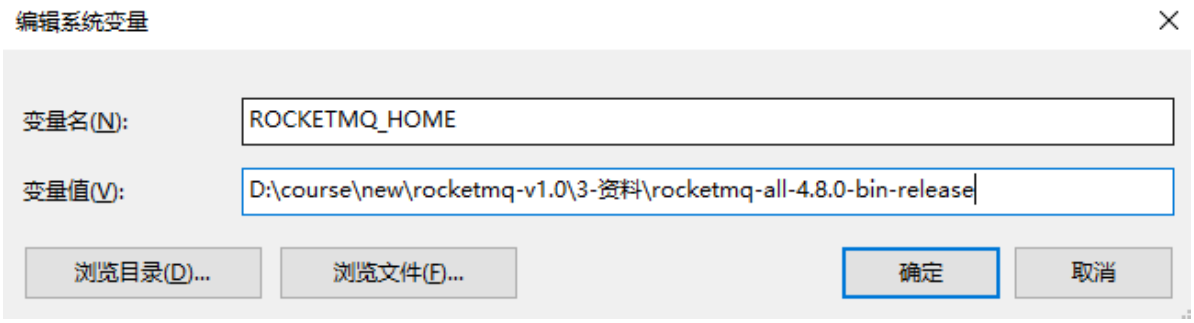

系统属性
计算机名
硬件
高级
系统保护
远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
环境变量(N)...
确定
取消
应用(A)

3、系统变量中-->新建



变量名：ROCKETMQ_HOME

变量值：MQ解压路径\MQ文件夹名



2.5.2启动

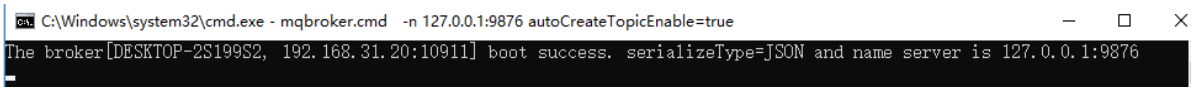
1、**启动NAMESERVER**

Cmd命令框执行进入至'MQ文件夹\bin'下 端口9876

```
start mqnamesrv.cmd
```

2、**启动BROKER**

```
start mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true
```



注意：闪退回命令行

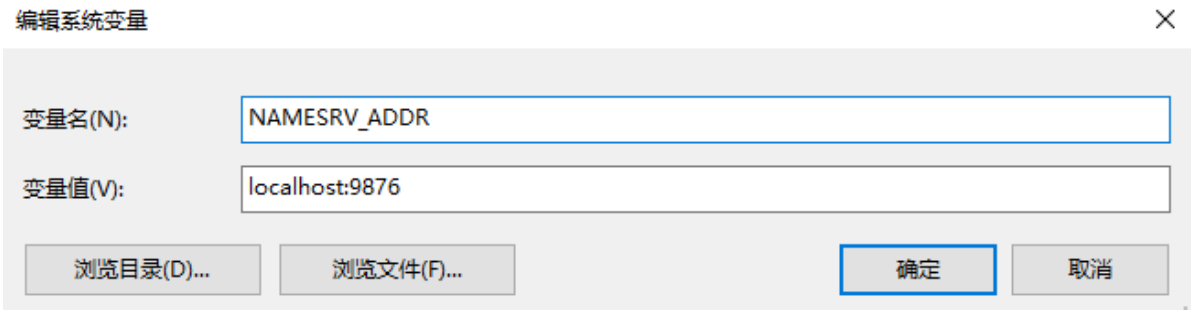
删除C:\Users\"当前系统用户名"\store下的所有文件。

2.5.3测试

1、**新建环境变量**

变量名：NAMESRV_ADDR

变量值：localhost:9876



2、测试生产者发送消息

bin目录下

```
tools.cmd  org.apache.rocketmq.example.quickstart.Producer
```

3、测试消费者接收消息

bin目录下

```
tools.cmd org.apache.rocketmq.example.quickstart.Consumer
```

2.5.4控制台安装

1下载源码

```
git clone https://github.com/apache/rocketmq-externals.git
```

2进入rocketmq-externals\rocketmq-console 工程，编译源码

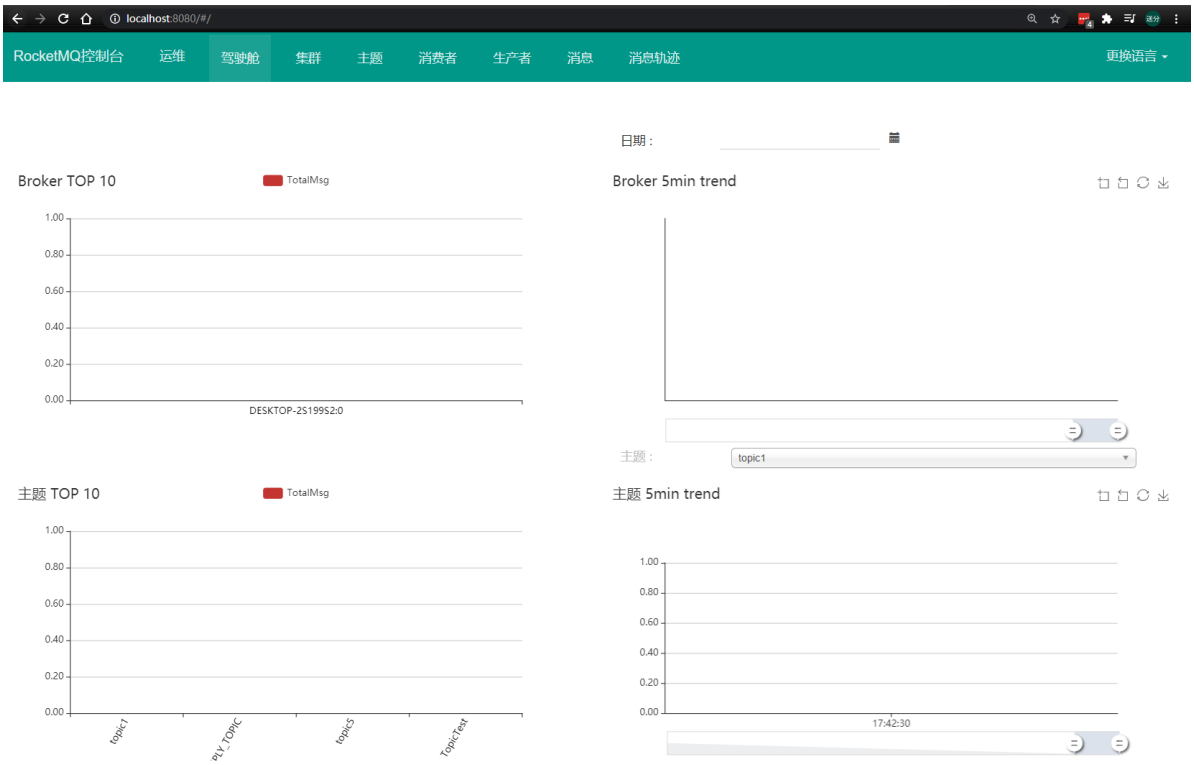
```
mvn clean package -Dmaven.test.skip=true
```

3target 目录生成 jar包

4运行

```
java -jar rocketmq-console-ng-2.0.0.jar
```

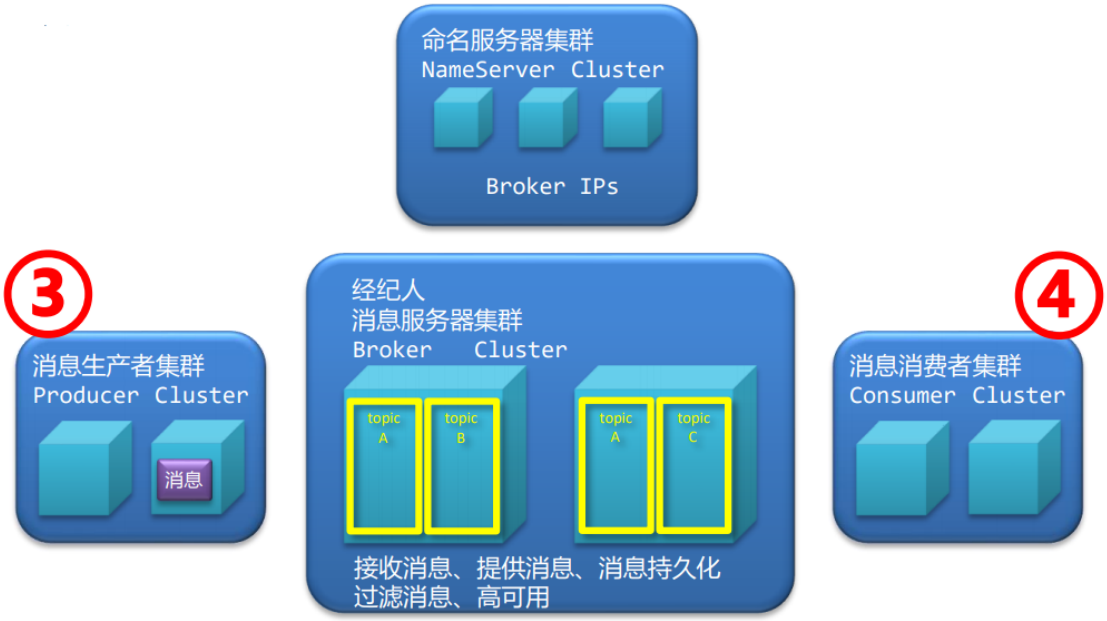
5访问 http://localhost:8080/#/



3. 消息发送（重点）

3.1 主要内容

1. 基于Java环境构建消息发送与消息接收基础程序
    1. 单生产者单消费者
    2. 单生产者多消费者
    3. 多生产者多消费者
2. 发送不同类型的消息
    1. 同步消息
    2. 异步消息
    3. 单向消息
3. 特殊的消息发送
    1. 延时消息
    2. 批量消息
4. 特殊的消息接收
    1. 消息过滤
5. 消息发送与接收顺序控制
6. 事务消息



3.2 消息发送与接收开发流程

1. 谁来发?
2. 发给谁?
3. 怎么发?
4. 发什么?
5. 发的结果是什么?
6. 打扫战场

3.3 单生产者单消费者消息发送（OneToOne）

1新建maven项目rocketmq

2导入RocketMQ客户端坐标

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.8.0</version>
</dependency>
```

3生产者 com.itheima.base.Producer

```
package com.itheima.base;

import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;

public class Producer {
    public static void main(String[] args) throws Exception {
        /**
         1. 谁来发?
         2. 发给谁?
         3. 怎么发?
         4. 发什么?
         5. 发的结果是什么?
```

```
         6. 打扫战场
         **/

        //1.创建一个发送消息的对象Producer
        DefaultMQProducer producer = new DefaultMQProducer("group1");
        //2.设定发送的命名服务器地址
        producer.setNamesrvAddr("localhost:9876");
        //3.1启动发送的服务
        producer.start();
        //4.创建要发送的消息对象,指定topic，指定内容body
        Message msg = new Message("topic1", "hello rocketmq".getBytes("UTF-8"));
        //3.2发送消息
        SendResult result = producer.send(msg);
        System.out.println("返回结果：" + result);
        //5.关闭连接
        producer.shutdown();
    }
}
```

4消费者

```
package com.itheima.base;

import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;

import java.util.List;

public class Consumer {
    public static void main(String[] args) throws Exception {
        /**
         1. 谁来发？
         2. 发给谁？
         3. 怎么发？
         4. 发什么？
         5. 发的结果是什么？
         6. 打扫战场
         **/

        //1.创建一个接收消息的对象Consumer
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("group1");
        //2.设定接收的命名服务器地址
        consumer.setNamesrvAddr("localhost:9876");
        //3.设置接收消息对应的topic,对应的sub标签为任意
        consumer.subscribe("topic1","*");
        //3.开启监听，用于接收消息
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt>
list, ConsumeConcurrentlyContext consumeConcurrentlyContext) {
                //遍历消息
                for (MessageExt msg : list) {
```

```
                    System.out.println("收到消息："+msg);
                }
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        //4.启动接收消息的服务
        consumer.start();
        System.out.println("接受消息服务已经开启！");
        //5 不要关闭消费者！
    }
}
```

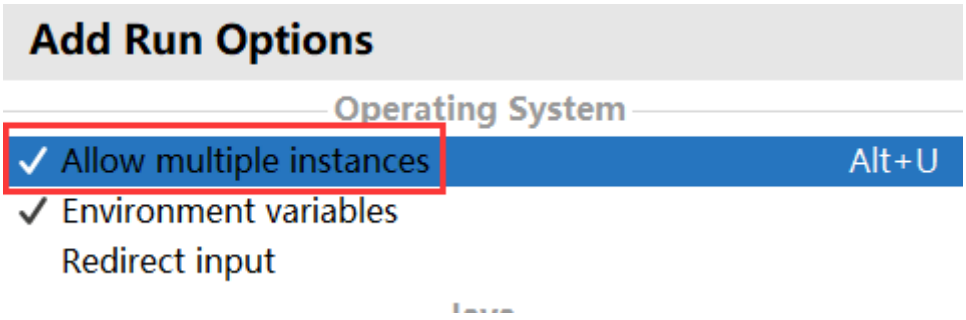
3.4 单生产者多消费者消息发送（OneToMany）

1生产者 com.itheima.one2many.Producer

```
        //1.创建一个发送消息的对象Producer
        DefaultMQProducer producer = new DefaultMQProducer("group1");
        //2.设定发送的命名服务器地址
        producer.setNamesrvAddr("localhost:9876");
        //3.1启动发送的服务
        producer.start();
        for (int i = 0; i < 10; i++) {
            //4.创建要发送的消息对象,指定topic，指定内容body
            Message msg = new Message("topic1", ("hello
rocketmq"+i).getBytes("UTF-8"));
            //3.2发送消息
            SendResult result = producer.send(msg);
            System.out.println("返回结果：" + result);
        }
        //5.关闭连接
        producer.shutdown();
```

2消费者（负载均衡模式：默认模式）

开启多实例运行




```
        //1.创建一个接收消息的对象Consumer
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("group1");
        //2.设定接收的命名服务器地址
        consumer.setNamesrvAddr("localhost:9876");
        //3.设置接收消息对应的topic,对应的sub标签为任意
        consumer.subscribe("topic1","*");
        //设置当前消费者的消费模式（默认模式：负载均衡）
        consumer.setMessageModel(MessageModel.CLUSTERING);
        //3.开启监听，用于接收消息
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
```

```
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt>
list, ConsumeConcurrentlyContext consumeConcurrentlyContext) {
                //遍历消息
                for (MessageExt msg : list) {
                    System.out.println("收到消息："+msg);
                    System.out.println("消息是："+new String(msg.getBody()));
                }
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        //4.启动接收消息的服务
        consumer.start();
        System.out.println("接受消息服务已经开启！");

        //5 不要关闭消费者！
```

注意：同一个消费者 多份。争抢topic数据。

3.5 单生产者多消费者消息发送（OneToMany）

消费者（广播模式）

```
        //1.创建一个接收消息的对象Consumer
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("group1");
        //2.设定接收的命名服务器地址
        consumer.setNamesrvAddr("localhost:9876");
        //3.设置接收消息对应的topic,对应的sub标签为任意
        consumer.subscribe("topic1","*");
        //设置当前消费者的消费模式（默认模式：负载均衡）
        //consumer.setMessageModel(MessageModel.CLUSTERING);
        //设置当前消费者的消费模式（广播模式）
        consumer.setMessageModel(MessageModel.BROADCASTING);
        //3.开启监听，用于接收消息
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt>
list, ConsumeConcurrentlyContext consumeConcurrentlyContext) {
                //遍历消息
                for (MessageExt msg : list) {
                    System.out.println("收到消息："+msg);
                    System.out.println("消息是："+new String(msg.getBody()));
                }
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        //4.启动接收消息的服务
        consumer.start();
        System.out.println("接受消息服务已经开启！");

        //5 不要关闭消费者！
```

3.6 多生产者多消费者消息发送（ManyToMany）

1. 多生产者产生的消息可以被同一个消费者消费，也可以被多个消费者消费

3.7 小节

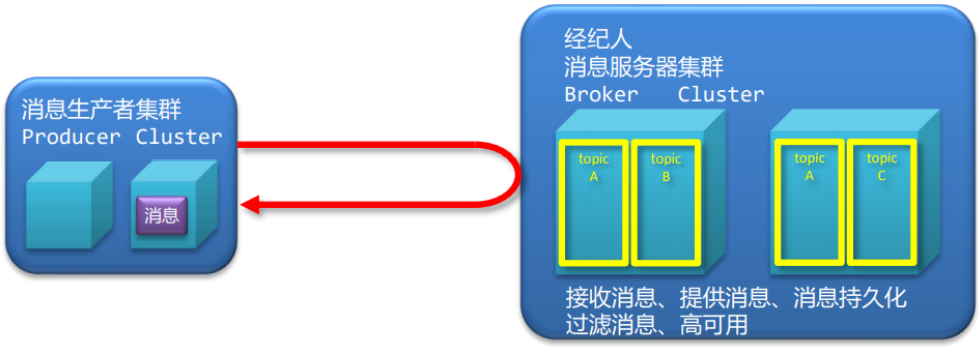
1. 消息发送
    1. One-To-One（基础发送与基础接收）
    2. One-To-Many（负载均衡模式与广播模式）
    3. Many-To-Many

4 消息类别

1. 同步消息
2. 异步消息
3. 单向消息

4.1 同步消息

特征：即时性较强，重要的消息，且必须有回执的消息，例如短信，通知（转账成功）



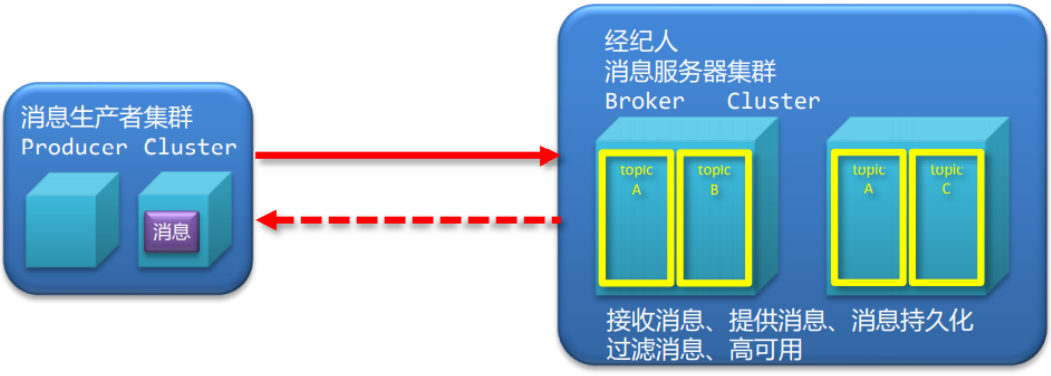
代码实现

com.itheima.messageType拷贝producer

```
SendResult result = producer.send(msg);
```

4.2 异步消息

特征：即时性较弱，但需要有回执的消息，例如订单中的某些信息



代码实现

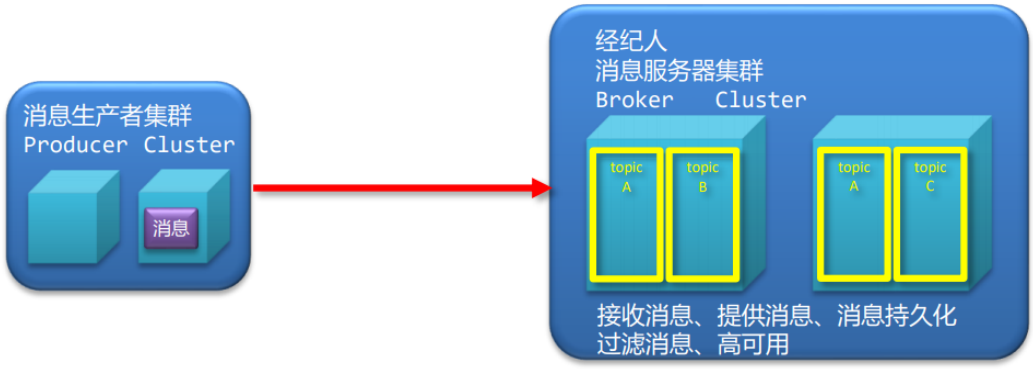
```
        //1.创建一个发送消息的对象Producer
        DefaultMQProducer producer = new DefaultMQProducer("group1");
        //2.设定发送的命名服务器地址
        producer.setNamesrvAddr("localhost:9876");
        //3.1启动发送的服务
        producer.start();
        for (int i = 0; i < 10; i++) {
            //4.创建要发送的消息对象,指定topic，指定内容body
            Message msg = new Message("topic1", ("hello
rocketmq"+i).getBytes("UTF-8"));
            //3.2 同步消息
            //SendResult result = producer.send(msg);
            //System.out.println("返回结果：" + result);

            //异步消息
            producer.send(msg, new SendCallback() {
                //表示成功返回结果
                @Override
                public void onSuccess(SendResult sendResult) {
                    System.out.println(sendResult);
                }
                //表示发送消息失败
                @Override
                public void onException(Throwable throwable) {
                    System.out.println(throwable);
                }
            });

            System.out.println("消息"+i+"发完了，做业务逻辑去了！");
        }
        //休眠10秒
        TimeUnit.SECONDS.sleep(10);
        //5.关闭连接
        producer.shutdown();
```

4.3 单向消息

特征：不需要有回执的消息，例如日志类消息



代码实现

```
producer.sendOneway(msg);
```

4.4 延时消息

消息发送时并不直接发送到消息服务器，而是根据设定的等待时间到达，起到延时到达的缓冲作用

```
Message msg = new Message("topic3",("延时消息：hello rocketmq "+i).getBytes("UTF-8"));
//设置延时等级3,这个消息将在10s之后发送(现在只支持固定的几个时间,详看delayTimeLevel)
msg.setDelayTimeLevel(3);
SendResult result = producer.send(msg);
System.out.println("返回结果："+result);
```

目前支持的消息时间

```
private String messageDelayLevel = "1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h";
```

4.5 批量消息

批量发送消息能显著提高传递小消息的性能.

发送批量消息:

```
        List<Message> msgList = new ArrayList<Message>();
        Message msg1 = new Message("topic1", ("hello rocketmq1").getBytes("UTF-8"));
        Message msg2 = new Message("topic1", ("hello rocketmq2").getBytes("UTF-8"));
        Message msg3 = new Message("topic1", ("hello rocketmq3").getBytes("UTF-8"));

        msgList.add(msg1);
        msgList.add(msg2);
        msgList.add(msg3);


        SendResult result = producer.send(msgList);
```

注意限制:

1这些批量消息应该有相同的topic

2相同的waitStoreMsgOK

3不能是延时消息

4消息内容总长度不超过4M

消息内容总长度包含如下:

- topic（字符串字节数）
- body （字节数组长度）
- 消息追加的属性（key与value对应字符串字节数）
- 日志（固定20字节）

4.6 消息过滤

4.6.1分类过滤

按照tag过滤信息。

生产者

```
Message msg = new Message("topic6","tag2",("消息过滤按照tag：hello rocketmq
2").getBytes("UTF-8"));
```

消费者

```
//接收消息的时候，除了制定topic，还可以指定接收的tag,*代表任意tag
consumer.subscribe("topic6","tag1 || tag2");
```

4.6.2语法过滤（属性过滤/语法过滤/SQL过滤）

基本语法

- 数值比较，比如：**>，>=，<，<=，BETWEEN，=；**
- 字符比较，比如：**=，<>，IN；**
- **IS NULL** 或者 **IS NOT NULL；**
- 逻辑符号 **AND，OR，NOT；**

常量支持类型为：

- 数值，比如：**123，3.1415；**
- 字符，比如：**'abc'，必须用单引号包裹起来；**
- **NULL**，特殊的常量
- 布尔值，**TRUE** 或 **FALSE**

生产者

```
//为消息添加属性
msg.putUserProperty("vip","1");
msg.putUserProperty("age","20");
```

消费者

```
//使用消息选择器来过滤对应的属性，语法格式为类SQL语法
consumer.subscribe("topic7", MessageSelector.bySql("age >= 18"));
consumer.subscribe("topic6", MessageSelector.bySql("name = 'litiedan'"));
```

注意：SQL过滤需要依赖服务器的功能支持，在broker.conf配置文件中添加对应的功能项，并开启对应功能

```
enablePropertyFilter=true
```

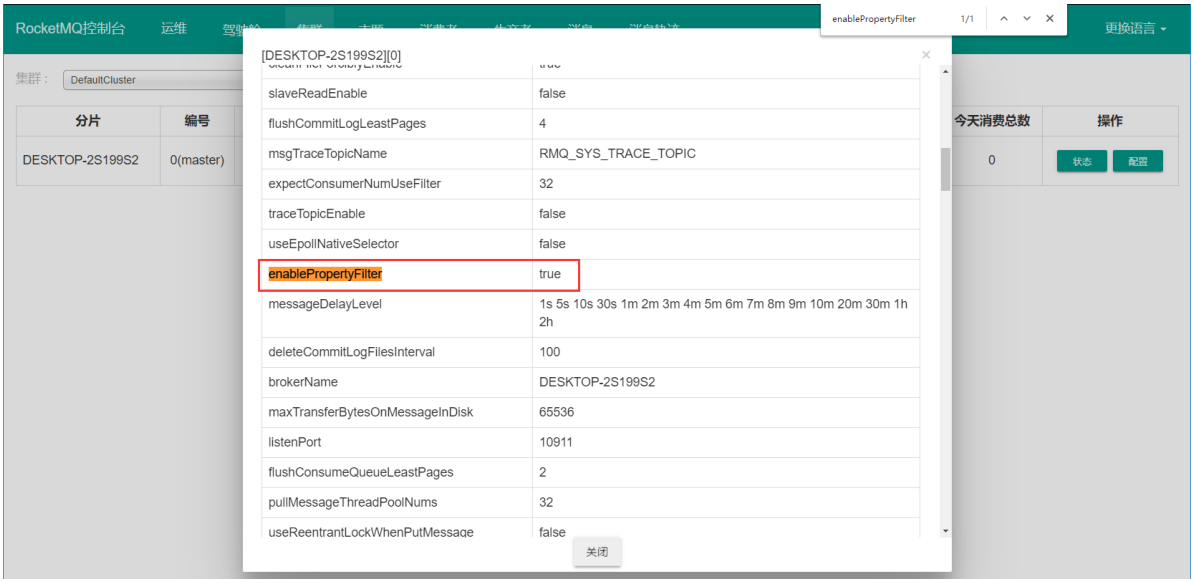
重启broker

```
start mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true
```

或者直接cmd中输入

```
mqadmin.cmd updateBrokerConfig -blocalhost:10911 -kenablePropertyFilter -vtrue
```

页面查看开启与否



5springboot整合

新建 springboot项目

5.1导包

```
    <dependency>
        <groupId>org.apache.rocketmq</groupId>
        <artifactId>rocketmq-spring-boot-starter</artifactId>
        <version>2.0.3</version>
    </dependency>
```

5.2配置文件

```
rocketmq.name-server=localhost:9876
rocketmq.producer.group=demo_producer
```

5.3实体类

```
public class user implements Serializable {
    String userName;
    String userId;

    public user(){

    }

    public user(String userName, String userId) {
        this.userName = userName;
        this.userId = userId;
    }

    @Override
```

```
    public String toString() {
        return "demoEntity{" +
                "userName='" + userName + '\'' +
                ", userId='" + userId + '\'' +
                '}';
    }
```

5.4生产者

```
@RestController
public class DemoProducers {
    @Autowired
    private RocketMQTemplate template;

    @RequestMapping("/producer")
    public String producersMessage() {
        User user = new User("sharfine", "123456789");
        template.convertAndSend("demo-topic", user);
        return JSON.toJSONString(user);
    }
}
```

5.5消费者

```
@Service
@RocketMQMessageListener(topic = "demo-topic", consumerGroup = "demo_consumer")
public class DemoConsumers1 implements RocketMQListener<user> {
    @Override
    public void onMessage(user user) {
        System.out.println("Consumers1接收消息:" + demoEntity.toString());
    }
}
```

5.6其他消息

异步发送

```
       rocketMQTemplate.asyncSend("topic9", user, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                System.out.println(sendResult);
            }

            @Override
            public void onException(Throwable throwable) {
                System.out.println(throwable);
            }
        });
```

单向发送

```
rocketMQTemplate.sendOneWay("topic9",user);
```

延时消息

```
rocketMQTemplate.syncSend("topic9", MessageBuilder.withPayload("test
delay").build(),2000,2);
```

批量

```
List<Message> msgList = new ArrayList<>();
msgList.add(new Message("topic6", "tag1", "msg1".getBytes()));
msgList.add(new Message("topic6", "tag1", "msg2".getBytes()));
msgList.add(new Message("topic6", "tag1", "msg3".getBytes()));
rocketMQTemplate.syncSend("topic8",msgList,1000);
```

Tag过滤

消费者

```
@RocketMQMessageListener(topic = "topic9",consumerGroup =
"group1",selectorExpression = "tag1")
```

Sql过滤

```
@RocketMQMessageListener(topic = "topic9",consumerGroup =
"group1",selectorExpression = "age>18"
        ,selectorType= SelectorType.SQL92)
```

改消息模式

```
@RocketMQMessageListener(topic = "topic9",consumerGroup = "group1",messageModel =
MessageModel.BROADCASTING)
```