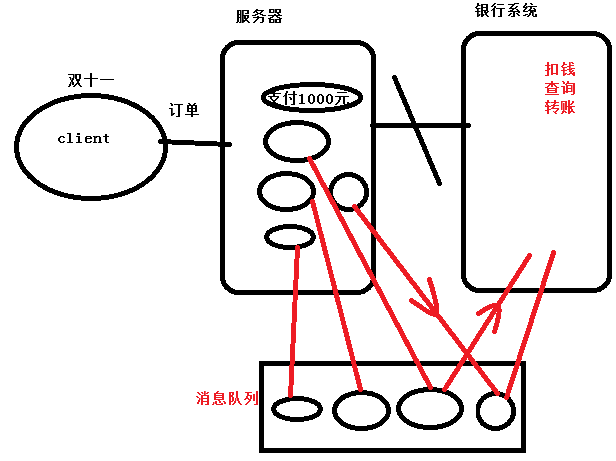
# 13.rabbitmq

## 13.1.消息队列概述

### 13.1.1.消息队列的作用



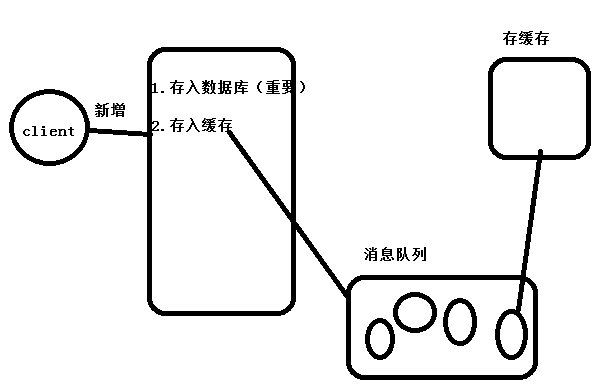
**消峰限流：**

客户端\服务器与服务器之间通信时，可以通过消息队列实现高峰无法处理的海量并发缓存

**解耦：**

可以通过消息队列实现异步通信

例如



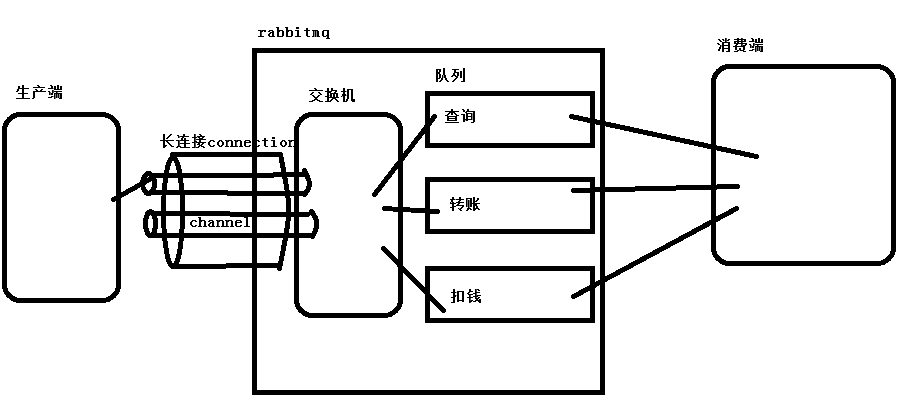
### 13.1.2.rabbitmq介绍

RabbitMQ是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ服务器是用[Erlang](https://baike.baidu.com/item/Erlang" \t "https://baike.baidu.com/item/rabbitmq/_blank)语言编写的，是企业级消息代理软件。

**AMQP：**sun对消息队列提出的规范（定义了一堆接口）

**Erlang语言：**一种并发语言（可以控制cpu核心线程资源），go语言也是并发语言。能将一个进程的并发性能处理到最强，Erlang语言是rabbitmq的核心组件交换机的开发语言

### 13.1.3.rabbitmq结构



客户端分为两种角色：

生产者：消息的发送者

消费者：消息的接收处理者

组件：

1.连接相关

connection：长连接，创建代价高

chennel：信道端连接，创建代价低

2.交换机-exchange

基于erlang语言开发，处理客户端生成的高并发的消息，将消息发送给队列

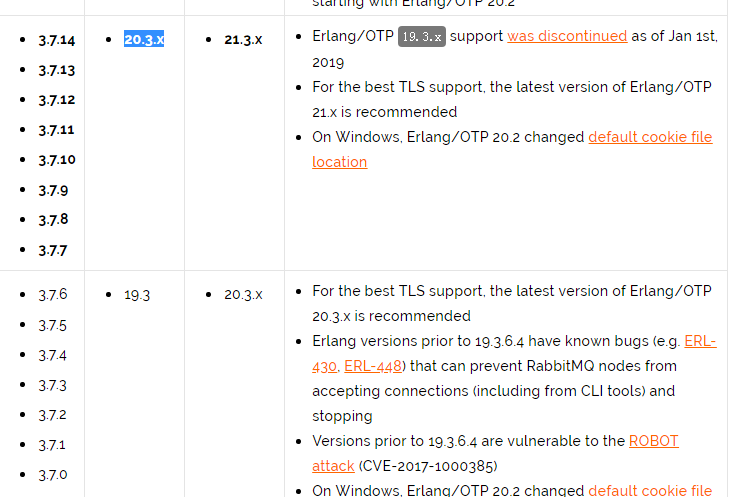
3.队列queue

真正存放\处理消息的组件，消息顺序排队，确认机制等都由queue维护

## 13.2.rabbitmq安装

rabbitmq核心组件exchange 基于erlang开发,相当于tomcat启动依赖java,linux需要安装erlang语言匹配rabbitmq的版本;

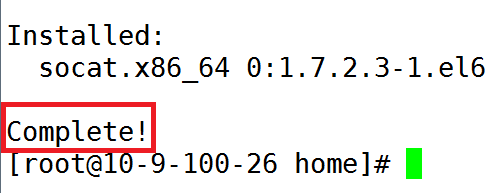
#erl查看erlang版本;（ctrl+z退出）



3.1准备一下使用的插件socat

rabbitmq3.7.7,有些命令,底层运行需要依赖插件socat(提供了相对应的脚本命令)

[root@10-9-100-26 home]# yum -y install socat



如果是虚拟机，需要安装centos的epel的扩展源：

[root@localhost home]# yum -y install epel-release

然后把/etc/yum.repos.d/epel.repo，文件第3行注释去掉，把第四行注释掉

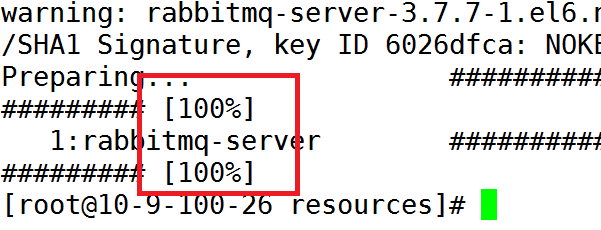
再安装socat

3.2利用rpm安装包安装rabbitmq

资源/home/resources/rabbitmq-server-3.7.7-1.el6.noarch.rpm

在资源所在文件夹执行安装rpm -ivh

[root@10-9-100-26 resources]# rpm -ivh rabbitmq-server-3.7.7-1.el6.noarch.rpm



3.3环境配置

* 远程访问开启除了locahost以外的用户登录权限

将模板文件rabbitmq.config.example文件拷贝到/ect/rabbitmq/文件夹下重命名为rabbitmq.config

[root@10-9-100-26 ~]# cp /usr/share/doc/rabbitmq-server-3.7.7/rabbitmq.config.example /etc/rabbitmq/rabbitmq.config

* 打开编辑拷贝后的文件rabbitmq.config修改61行左右

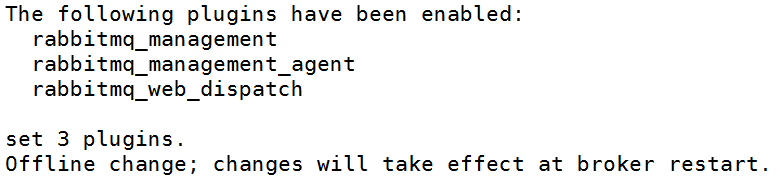


一定确定文件名称是rabbitmq.config

* 直接开启web控制台监听插件(启动web应用)

rabbitmq支持以视图页面web应用格式观察使用操作rabbitmq

[root@10-9-100-26 ~]# rabbitmq-plugins enable rabbitmq\_management



3.4启动rabbitmq

rabbitmq以rpm安装包方式安装完毕后,启动命令放到了

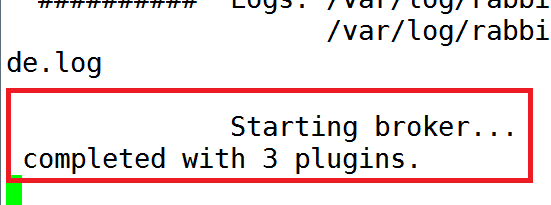
[root@10-9-100-26 bin]# pwd

/usr/lib/rabbitmq/bin

强制调用当前目录的rabbitmq-server启动服务

必须添加./

[root@10-9-100-26 bin]# ./rabbitmq-server start

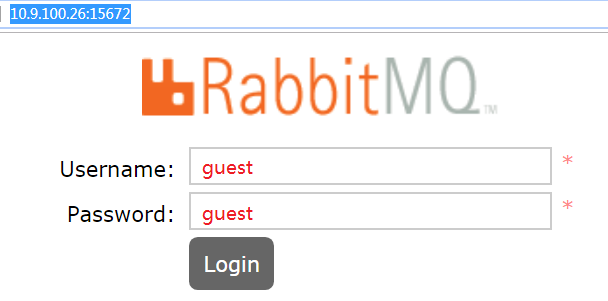


3.5验证访问web页面

10.9.100.26:15672(web默认访问端口)

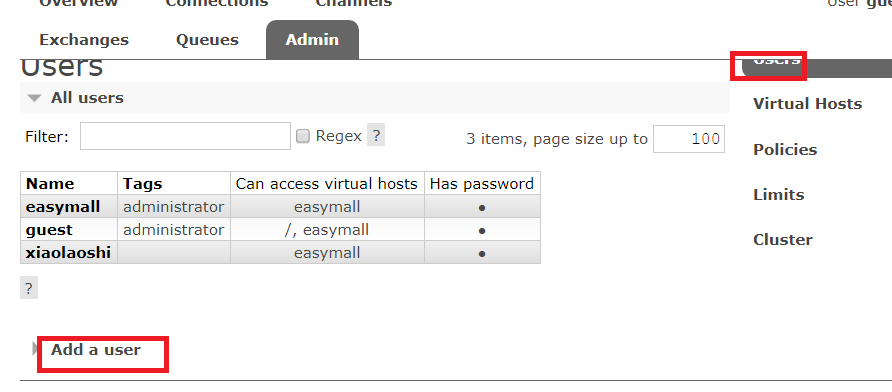
客户端代码访问要使用5672访问;

使用guest/guest

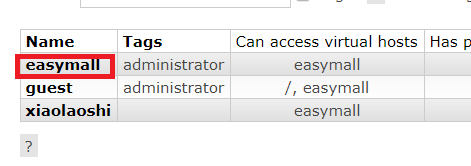


虚拟主机virtal host,每个虚拟主机交换机不互通的(队列是一样的),每生成一个虚拟主机,都会对应一批完全不相关的交换出现.形成最前端交换机的组隔离;

* 添加用户（可以不用添加，后面使用的是guest）

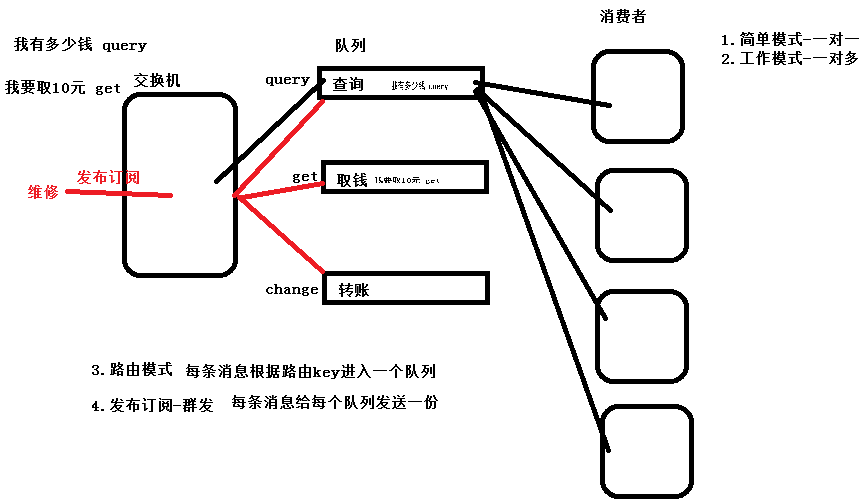


* 绑定虚拟注解virtual host





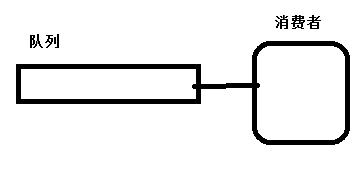
## 13.3.rabbitmq的五种工作模式



### 13.3.1.简单模式

针对队列和消费者逻辑关系

一个队列对应一个消费者，一发一接

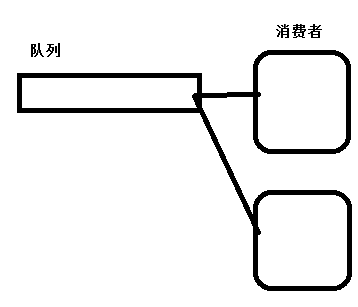


应用场景：短信、微信等的一对一发送

### 13.3.2.工作模式

针对队列和消费者逻辑关系

一个队列对应多个消费者，一发多接，但是只能有一个消费者抢到消息，先到先得（谁空闲谁得）

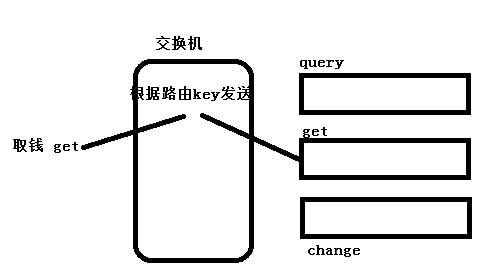


应用场景：抢红包、任务分配

### 13.3.3.路由模式

交换机与队列的逻辑

每条消息携带路由key，交换机根据路由key，发送到对应的队列中

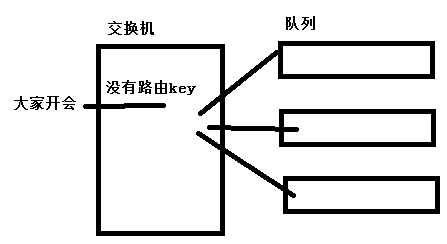


应用场景：请求按照类型消费，将程序开发时的error存入日志文件

### 13.3.4.发布订阅

交换机与队列的逻辑

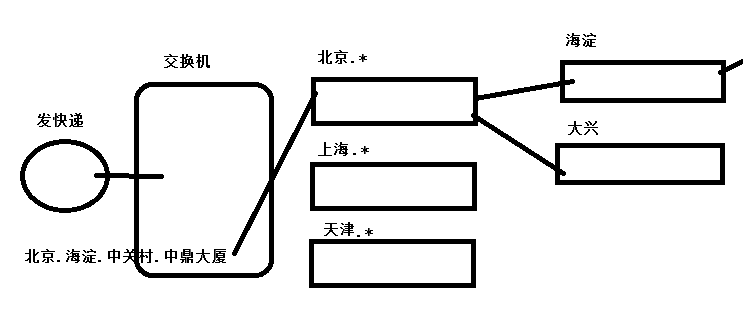
消息进入交换机时，不携带路由key，交换机会将这条消息发送到每个队列



应用场景：邮件群发、广告

### 13.3.5.主题模式

交换机与队列的逻辑



主题是一种特殊的路由，队列可以绑定topic交换机时，不使用具体的路由key，可以使用匹配符号。

\*：表示一个字符串

#：表示任意长度的字符串

## 13.4.五种模式代码实现

### 13.4.1.简单模式

1.pom导入依赖（easymall-search）

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

2.简单模式代码

|  |
| --- |
| **package** cn.tedu.rabbitmq.test;  **import** java.io.IOException;  **import** java.util.concurrent.TimeoutException;  **import** org.junit.Before;  **import** org.junit.Test;  **import** com.rabbitmq.client.Channel;  **import** com.rabbitmq.client.Connection;  **import** com.rabbitmq.client.ConnectionFactory;  **import** com.rabbitmq.client.~~QueueingConsumer~~;  **import** com.rabbitmq.client.~~QueueingConsumer~~.Delivery;  /\*\*  \* 实现一对一接发模式  \*/  **public** **class** SimpleMode {  //定义channel  **private** Channel channel;  //连接rabbitmq  @Before  **public** **void** initChannel() **throws** Exception{  //获取一个连接工厂  ConnectionFactory factory = **new** ConnectionFactory();  //设置连接参数  factory.setHost("192.168.171.142");  factory.setPort(5672);  factory.setUsername("guest");  factory.setPassword("guest");  factory.setVirtualHost("/");  //获取长连接  Connection conn = factory.newConnection();  //获取信道短连接  channel = conn.createChannel();  }  //实现生产端逻辑  @Test  **public** **void** send() **throws** Exception{  //准备消息  String msg = "特朗普禁华为";  //定义一个路由key  String routingKey = "news";  //声明一个队列，指定名称  //参一：queue，指定队列的名称  //参二：durable，表示队列（不含消息）是否持久化，  // 用于宕机重启恢复，true表示持久化，false表示不持久化  //参三：exclusive，表示队列是否专属，tru表示专属，  // 只有当前连接的客户端可以操作此队列，别人无法操作，false表示不专属，所有客户端都可以操作。  //参四：autoDelete，表示是否自动删除，true表示自动删除，  // 即队列使用完毕后自动删除，false表示不删除  //参五：arguments，表示queue的其他属性，例如队列的长度、超时时间  channel.queueDeclare("news", **false**, **false**, **false**,**null**);  //发送消息  //参一：交换机名称，空字符串代表默认的amqp交换机  //参二：路由key，交换机根据路由key，将消息交给指定的队列  //参三：消息是否持久化  //参四：需要传输的消息，byte[]格式  channel.basicPublish("", routingKey, **null**, msg.getBytes());  }  //消费者逻辑  @Test  **public** **void** consumer() **throws** Exception{  //创建消费者对象  ~~QueueingConsumer~~ consumer = **new** ~~QueueingConsumer~~(channel);  //为消费者绑定队列  channel.basicConsume("news", consumer);  //监听队列  **while**(**true**){  //获取一条消息对象  Delivery delivery = consumer.~~nextDelivery~~();  //打印消息  System.***out***.println(**new** String(delivery.getBody()));  }  }  } |

### 13.4.2.工作模式

|  |
| --- |
| **package** cn.tedu.rabbitmq.test;  **import** org.junit.Before;  **import** org.junit.Test;  **import** com.rabbitmq.client.Channel;  **import** com.rabbitmq.client.Connection;  **import** com.rabbitmq.client.ConnectionFactory;  **import** com.rabbitmq.client.~~QueueingConsumer~~;  **import** com.rabbitmq.client.~~QueueingConsumer~~.Delivery;  /\*  \* 工作模式  \*/  **public** **class** WordMode {  //定义channel  **private** Channel channel;  //连接rabbitmq  @Before  **public** **void** initChannel() **throws** Exception{  //获取一个连接工厂  ConnectionFactory factory = **new** ConnectionFactory();  //设置连接参数  factory.setHost("192.168.171.144");  factory.setPort(5672);  factory.setUsername("guest");  factory.setPassword("guest");  factory.setVirtualHost("/");  //获取长连接  Connection conn = factory.newConnection();  //获取信道短连接  channel = conn.createChannel();  }  //实现生产端逻辑  @Test  **public** **void** send() **throws** Exception{  //定义一个路由key  String routingKey = "news";  //声明一个队列  //参一：queue，指定队列的名称  //参二：durable，表示队列（不含消息）是否持久化，  // 用于宕机重启恢复，true表示持久化，false表示不持久化  //参三：exclusive，表示队列是否专属，tru表示专属，  // 只有当前连接的客户端可以操作此队列，别人无法操作，false表示不专属，所有客户端都可以操作。  //参四：autoDelete，表示是否自动删除，true表示自动删除，  // 即队列使用完毕后自动删除，false表示不删除  //参五：arguments，表示queue的其他属性，例如队列的长度、超时时间  channel.queueDeclare("news", **false**, **false**, **false**,**null**);  //发送消息  //参一：交换机名称，空字符串代表默认的amqp交换机  //参二：路由key，交换机根据路由key，将消息交给指定的队列  //参三：消息是否持久化  //参四：需要传输的消息，byte[]格式  **for**(**int** i = 0 ;i<100;i++){  String msg = "hello baby"+i;  channel.basicPublish("", routingKey, **null**, msg.getBytes());  }  }  //消费者逻辑  @Test  **public** **void** consumer1() **throws** Exception{  //创建消费者对象  ~~QueueingConsumer~~ consumer = **new** ~~QueueingConsumer~~(channel);  //为消费者绑定队列  channel.basicConsume("news",**false**,consumer);//默认是不回执  //监听队列  **while**(**true**){  //获取一条消息对象  Delivery delivery = consumer.~~nextDelivery~~();  //打印消息  System.***out***.println("消费者1："+**new** String(delivery.getBody()));  //手动回执  //参一：信息的标识，用来告诉队列删除哪条信息  //参二：false表示只删除此条消息，true表示连同上一条一起删除  channel.basicAck(delivery.getEnvelope().getDeliveryTag(), **false**);  }  }  //消费者逻辑  @Test  **public** **void** consumer2() **throws** Exception{  //创建消费者对象  ~~QueueingConsumer~~ consumer = **new** ~~QueueingConsumer~~(channel);  //为消费者绑定队列  channel.basicConsume("news",**false**,consumer);//默认是不回执  //监听队列  **while**(**true**){  //获取一条消息对象  Delivery delivery = consumer.~~nextDelivery~~();  //打印消息  System.***out***.println("消费者2："+**new** String(delivery.getBody()));  //手动回执  //参一：信息的标识，用来告诉队列删除哪条信息  //参二：false表示只删除此条消息，true表示连同上一条一起删除  channel.basicAck(delivery.getEnvelope().getDeliveryTag(), **false**);  }  }  } |

### 13.4.3.路由模式

|  |
| --- |
| @Test  **public** **void** send() **throws** Exception{  //声明交换机  //参一：交换机名称  //参二：交换机类型 ，路由  channel.exchangeDeclare("direct01", "direct");  //声明两个队列  channel.queueDeclare("query", **false**, **false**, **false**, **null**);  channel.queueDeclare("get", **false**, **false**, **false**, **null**);  //绑定交换机和队列  channel.queueBind("query", "direct01", "queryM");  channel.queueBind("get", "direct01", "getM");  //发送消息  channel.basicPublish("direct01", "queryM", **null**, "查查我的钱".getBytes());  } |

### 13.4.4.发布订阅模式

|  |
| --- |
| @Test  **public** **void** sent() **throws** Exception{  //声明交换机  channel.exchangeDeclare("fanout01", "fanout");  //声明队列  channel.queueDeclare("piaolaoshi", **false**, **false**, **false**, **null**);  channel.queueDeclare("canglaoshi", **false**, **false**, **false**, **null**);  channel.queueDeclare("xiaozelaoshi", **false**, **false**, **false**, **null**);  //交换机绑定三个队列  channel.queueBind("piaolaoshi", "fanout01", "");  channel.queueBind("canglaoshi", "fanout01", "");  channel.queueBind("xiaozelaoshi", "fanout01", "");  //发送消息  channel.basicPublish("fanout01", "", **null**, "大家来开会".getBytes());  } |

### 13.4.5.主题模式

|  |
| --- |
| @Test  **public** **void** send() **throws** IOException{  //声明交换机  channel.exchangeDeclare("topic01", "topic");  //声明队列  channel.queueDeclare("beijing", **false**, **false**, **false**, **null**);  channel.queueDeclare("haidian", **false**, **false**, **false**, **null**);  channel.queueDeclare("zhongguancun", **false**, **false**, **false**, **null**);  //交换机绑定队列  channel.queueBind("beijing", "topic01", "beijing.#");  channel.queueBind("haidian", "topic01", "beijing.haidian.\*");  channel.queueBind("zhongguancun", "topic01", "beijing.haidian.zhongguancun.\*");  //发送消息  channel.basicPublish("topic01","beijing.haidian.zhongguancun", **null**, "快递".getBytes());  } |

## 13.5.springboot整合rabbitmq

### 13.5.1.整合channel过程(不用)

1.pom依赖

2.配置文件

3.配置类，初始化channel

4.哪用哪注入

### 13.5.2.springboot自带的整合

springboot整合redis过程

问题一：注入哪个类

RedisAutoConfiguration类中找到@Bean

发现有RedisTemplate、StringRedisTemplate

问题二：配哪些参数、怎么配

RedisProperties类中可以找到所有可以配置的内容

springboot整合es过程

ElasticsearchAutoConfiguration

ElasticsearchProperties

springboot整合rabbitmq

RabbitAutoConfiguration类的RabbitTemplate

RabbitProperties类参考配置文件内容

## 13.6.秒杀系统

### 13.6.1.系统搭建

1.quickstart（easymall-microservice-seckill）

2.pom继承easymall-parent，repository，resources，依赖amqp

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

3.配置文件

spring.rabbitmq.host=192.168.171.139

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

spring.rabbitmq.virtualHost=/

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** StarterSeckillCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterSeckillCenter.**class**, args);  }  } |

### 13.6.2.初始化测试

1.通过配置类初始化数据（交换机、队列等）

|  |
| --- |
| **package** cn.tedu.seckill.config;  **import** org.springframework.amqp.core.Binding;  **import** org.springframework.amqp.core.BindingBuilder;  **import** org.springframework.amqp.core.DirectExchange;  **import** org.springframework.amqp.core.Queue;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  **public** **class** RabbitmqConfig {  //声明交换机名、队列名、路由key  **public** **static** **final** String ***exName***="seckillEx";  **public** **static** **final** String ***qName***="seckillQueue";  **public** **static** **final** String ***routingKey***="seckill";  //声明交换机  @Bean  **public** DirectExchange initDirectExchange(){  **return** **new** DirectExchange(***exName***,**false**,**false**);  }  //声明队列  @Bean  **public** Queue initQueue(){  **return** **new** Queue(***qName***,**false**,**false**,**false**);  }  //交换机和队列绑定  @Bean  **public** Binding initBinding(){  **return** BindingBuilder.*bind*(initQueue())  .to(initDirectExchange()).with(***routingKey***);  }  } |

2.生产端发送消息

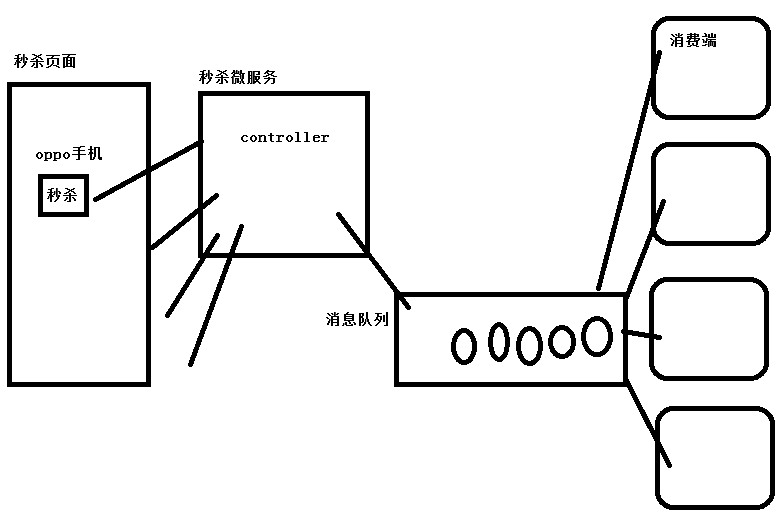
通过浏览器访问controller，然后将请求交给消息队列

|  |
| --- |
| **package** cn.tedu.seckill.controller;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** cn.tedu.seckill.config.RabbitmqConfig;  @RestController  **public** **class** SeckillController {  @Autowired  **private** RabbitTemplate client;  @RequestMapping("send")  **public** String sendMsg(String msg){  client.convertAndSend(RabbitmqConfig.***exName***,  RabbitmqConfig.***routingKey***,msg);  **return** "ok";  }  } |

启动报错，注释掉repository依赖

页面访问：http://localhost:8080/send?msg=goodbye

### 13.6.3.秒杀结构



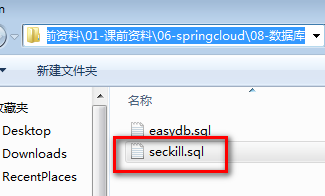
### 13.6.4.创建消费者（测试代码）

消费者本可以部署在不同的微服务中，这里为了方便，放在seckill中了

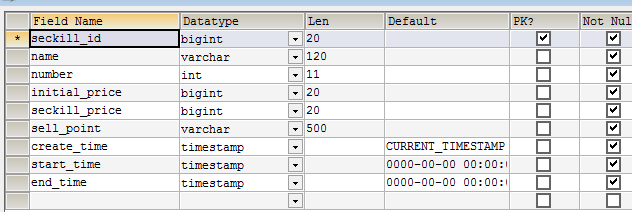
消费端需要一个监听者@RabbitListener，此注解必须作用在@Component类上

|  |
| --- |
| **package** cn.tedu.seckill.consumer;  **import** org.springframework.amqp.rabbit.annotation.RabbitListener;  **import** org.springframework.stereotype.Component;  **import** cn.tedu.seckill.config.RabbitmqConfig;  @Component  **public** **class** SeckillConsumer {  //定义一个消息处理的方法  //需要添加监听注解，并需要指定监听的队列  //@RabbitListener(queues={"queue01","queue01"})//监听多个队列  @RabbitListener(queues=RabbitmqConfig.***qName***)  **public** **void** processMsg(String msg){  System.***out***.println("消费者接收消息："+msg);  }  } |

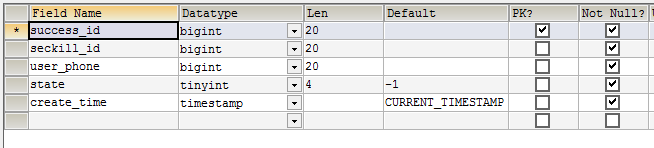
### 13.6.5.导入数据库



seckill结构：秒杀商品的信息



success表，秒杀订单表



### 13.6.6.秒杀商品查询

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall/seckills/list |
| 后台接收 | seckill/manage/list |
| 请求方式 | get |
| 请求参数 | 空的 |
| 返回数据 | List<Seckill>的查询数据 |

SeckillController

|  |
| --- |
| @Autowired  **private** SeckillService seckillService;  @RequestMapping("seckill/manage/list")  **public** List<Seckill> queryAll(){  **return** seckillService.queryAll();  } |

SeckillService

|  |
| --- |
| **package** cn.tedu.seckill.service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.jt.common.pojo.Seckill;  **import** cn.tedu.seckill.mapper.SeckillMapper;  @Service  **public** **class** SeckillService {  @Autowired  **private** SeckillMapper seckillMapper;  **public** List<Seckill> queryAll() {  **return** seckillMapper.queryAll();  }  } |

SeckillMapper

|  |
| --- |
| **package** cn.tedu.seckill.mapper;  **import** java.util.List;  **import** com.jt.common.pojo.Seckill;  **public** **interface** SeckillMapper {  List<Seckill> queryAll();  } |

SeckillMapper.xml

|  |
| --- |
| <select id=*"queryAll"* resultType=*"Seckill"*>  select \* from seckill;  </select> |

修改配置文件

|  |
| --- |
| spring.rabbitmq.host=10.10.59.171  spring.rabbitmq.username=guest  spring.rabbitmq.password=guest  spring.rabbitmq.virtualHost=/  server.port=10007  #RedisCluster  spring.redis.cluster.nodes=192.168.171.138:8000,192.168.171.138:8001  spring.redis.cluster.maxTotal=100  spring.redis.cluster.maxIdle=10  spring.redis.cluster.minIdle=5  spring.datasource.driverClassName=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql:///seckill  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.initialSize=5  spring.datasource.maxActive=50  spring.datasource.maxIdle=10  spring.datasource.minIdle=5  mybatis.typeAliasesPackage=com.jt.common.pojo  mybatis.mapperLocations=classpath:mapper/\*.xml  mybatis.configuration.mapUnderscoreToCamelCase=true  mybatis.configuration.cacheEnables=false  spring.application.name=seckillservice  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka |

pom文件依赖repository和rediscluster

启动类：添加注解

@EnableEurekaClient

@MapperScan("cn.tedu.seckill.mapper")

**nginx**

location /seckills {

proxy\_pass http://127.0.0.1:9005/zuul-seckill/seckill/manage;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

**zuul**

zuul.routes.zuul-seckill.path=/zuul-seckill/\*\*

zuul.routes.zuul-seckill.serviceId=seckillservice

### 13.6.7.单个商品查询

|  |  |
| --- | --- |
| js请求地址 | <http://www.easymall/seckills/detail?seckillId=1> |
| 后台接收 | /seckill/manage/detail?seckillId=1 |
| 请求方式 | Get |
| 请求参数 | Long seckillId |
| 返回数据 | Seckill根据id的查询对象 |

SeckillController

|  |
| --- |
| @RequestMapping("/seckill/manage/detail")  **public** Seckill queryOne(Long seckillId){  **return** seckillService.queryOne(seckillId);  } |

SeckillService

|  |
| --- |
| **public** Seckill queryOne(Long seckillId) {  **return** seckillMapper.queryOne(seckillId);  } |

SeckillMapper

|  |
| --- |
| Seckill queryOne(Long seckillId); |

SeckillMapper.xml

|  |
| --- |
| <select id=*"queryOne"* parameterType=*"long"* resultType=*"Seckill"*>  select \* from seckill where seckill\_id=#{seckillId};  </select> |

问题：发现页面显示开始时间和结束时间一样



在seckill-list.html中57行数据拼接错误

$("#seckillList").append("<tr class=\"success\"><td>"+secName+"</td><td>"+number+"</td><td>"+initialPrice+"</td><td>"+seckillPrice+"</td><td>"+sellPoint+"</td><td>"+dateStart+"</td><td>"+dateEnd+"</td><td><a class=\"btn btn-primary btn-md\" href='javascript:void(0)' onclick='goDetail(\""+seckillId+"\")'target=\"\_blank\">详情页面</a></td>")

### 13.6.8.秒杀功能

在开始秒杀的一刻，并发量特别高，将userPhone和seckillId放入消息队列

|  |  |
| --- | --- |
| js请求地址 | [http://www.easymall/seckills/{seckillId}](http://www.easymall/seckills/%7bseckillId%7d) |
| 后台接收 | /seckill/manage/{seckillId} |
| 请求方式 | Get |
| 请求参数 | Long seckillId 路径传参 |
| 返回数据 | SysResult的返回对象  Integer status 200表示秒杀成功  String msg:ok表示成功  Object data:其他数据 |
| 备注 | 将userPhone和seckillId放入消息队列 |

注意：此案例中用户可以不用登陆进行秒杀，没有手机号数据，后台生成一个随机的手机号

SeckillController

|  |
| --- |
| @Autowired  **private** RabbitTemplate client;  @RequestMapping("/seckill/manage/{seckillId}")  **public** SysResult startSeckill(@PathVariable Long seckillId){  **try** {  //随机生成用户phone  String userPhone = "1386666"+RandomUtils.*nextInt*(1000,9999);  String msg = userPhone+"/"+seckillId;  //将消息发送到消息队列  client.convertAndSend(RabbitmqConfig.***exName***,  RabbitmqConfig.***routingKey***,msg);  **return** SysResult.*ok*();  } **catch** (Exception e) {  **return** SysResult.*build*(202, **null**, **null**);  }  } |

### 13.6.9.消费端逻辑

先将Success表的userId改为userPhone

需要一个successMapper操作订单表

从消息队列获取消息，判断是否有库存，根据userPhone和seckillId，增加订单

SeckillConsumer

|  |
| --- |
| **package** cn.tedu.seckill.consumer;  **import** java.util.Date;  **import** org.springframework.amqp.rabbit.annotation.RabbitListener;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Component;  **import** com.jt.common.pojo.Success;  **import** cn.tedu.seckill.config.RabbitmqConfig;  **import** cn.tedu.seckill.mapper.SeckillMapper;  **import** cn.tedu.seckill.success.SuccessMapper;  @Component  **public** **class** SeckillConsumer {  @Autowired  **private** SeckillMapper seckillMapper;  @Autowired  **private** SuccessMapper successMapper;  //定义一个消息处理的方法  //需要添加监听注解，并需要指定监听的队列  //@RabbitListener(queues={"queue01","queue01"})//监听多个队列  @RabbitListener(queues=RabbitmqConfig.***qName***)  **public** **void** processMsg(String msg){  //System.out.println("消费者接收消息："+msg);  //将消息按照/切分,获取userPhone和seckillId  String userPhone = msg.split("/")[0];  Long seckillId =Long.*parseLong*(msg.split("/")[1]);  //根据seckillId判断商品是否有库存，如果有库存，直接减掉库存  //存在一个小问题，判断时顺便判断时间是否符合要求  //nowTime>startTime,nowTime<endTime  //生成当前时间  Date now = **new** Date();  **int** result = seckillMapper.updateNum(seckillId,now);  **if**(result == 0){//库存修改失败  System.***out***.println("秒杀失败");  **return**;  }  //增加订单  Success success = **new** Success();  success.setCreateTime(now);  success.setSeckillId(seckillId);  success.setUserPhone(userPhone);  success.setState(0);  successMapper.saveSuccess(success);  }  } |

SeckillMapper

|  |
| --- |
| **int** updateNum(@Param("seckillId") Long seckillId,@Param("now") Date now); |

SuccessMapper

|  |
| --- |
| **void** saveSuccess(Success success); |

SeckillMapper.xml

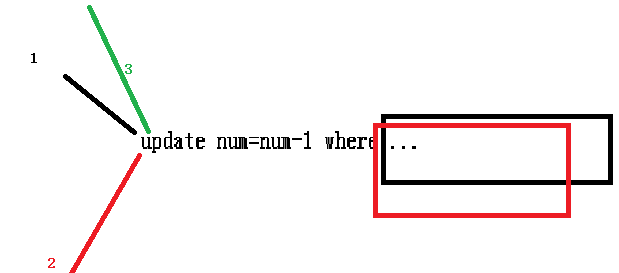
|  |
| --- |
| <update id=*"updateNum"*>  update seckill set number=numer-1  where seckill\_id=#{seckillId} and number>0  and #{now}>start\_time and #{now} &lt; end\_time;  </update> |

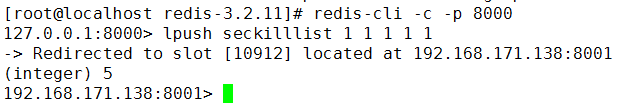
SuccessMapper.xml

|  |
| --- |
| <insert id=*"saveSuccess"* parameterType=*"Success"*>  insert into success(seckill\_id,user\_phone,state,create\_time)  values(#{seckillId},#{userPhone},#{state},#{createTime})  </insert> |

### 13.6.10.超卖逻辑

由于高并发，可能导致超卖现象，可以在redis中通过锁解决





在redis中声明list，并放入n（商品数量）个元素

在代码中先获取redis中list的元素，如果获取到，则继续秒杀，否则商品已卖完

|  |
| --- |
| @Component  **public** **class** SeckillConsumer {  @Autowired  **private** SeckillMapper seckillMapper;  @Autowired  **private** SuccessMapper successMapper;  //注入JedisCluster  @Autowired  **private** JedisCluster cluster;  //定义一个消息处理的方法  //需要添加监听注解，并需要指定监听的队列  //@RabbitListener(queues={"queue01","queue01"})//监听多个队列  @RabbitListener(queues=RabbitmqConfig.***qName***)  **public** **void** processMsg(String msg){  //System.out.println("消费者接收消息："+msg);  //判断redis中是否还有可秒杀的数据  **if**(!cluster.exists("seckilllist") ||  cluster.rpop("seckilllist") == **null**){  System.***out***.println("商品秒杀完毕");  **return**;  }  //将消息按照/切分,获取userPhone和seckillId  String userPhone = msg.split("/")[0];  Long seckillId =Long.*parseLong*(msg.split("/")[1]);  //根据seckillId判断商品是否有库存，如果有库存，直接减掉库存  //存在一个小问题，判断时顺便判断时间是否符合要求  //nowTime>startTime,nowTime<endTime  //生成当前时间  Date now = **new** Date();  **int** result = seckillMapper.updateNum(seckillId,now);  **if**(result == 0){//库存修改失败  System.***out***.println("秒杀失败");  **return**;  }  //增加订单  Success success = **new** Success();  success.setCreateTime(now);  success.setSeckillId(seckillId);  success.setUserPhone(userPhone);  success.setState(0);  successMapper.saveSuccess(success);  }  } |