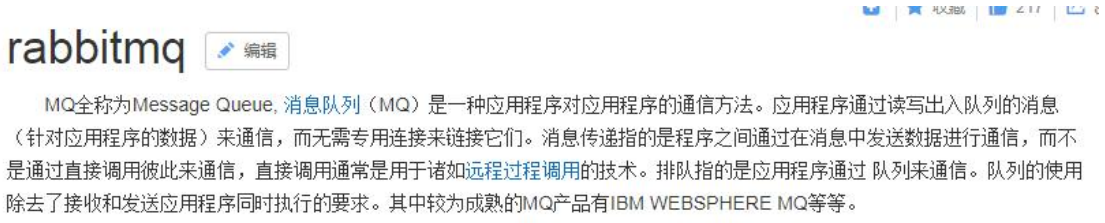
第四章 RabbitMQ 实战

（Spring Cloud 初级）

一、 什么是 RabbitMQ



二、 安装 RabbitMQ

1 安装 Erlang

1.1什么是 Erlang

Erlang(['ə:læŋ])是一种通用的面向并发的编程语言，它由瑞典电信设备制造商爱立信所辖的CS-Lab 开发，目的是创造一种可以应对大规模并发活动的编程语言和运行环境

系统版本：CentOS 6.5

RabbitMQ－Server：3.5.1

2 安装 erlang

2.1安装准备，下载安装文件

wget

https://packages.erlang-solutions.com/erlang-solutions-1.0-1.noarch.rpm

rpm -Uvh erlang-solutions-1.0-1.noarch.rpm

修改 primary.xml.gz 的 sha 的加密值

cd /var/cache/yum/x86_64/6/erlang-solutions

sha1sum primary.xml.gz

vim repomd.xml

修改

<data type="primary">

<checksum type="sha">结果为 sha1sum 命令结果</checksum>

3　安装 erlang

yum install erlang

4　安装完成后可以用 erl 命令查看是否安装成功

erl -version

5　安装 RabbitMQ Server

5.1安装准备，下载 RabbitMQ Server

wget

http://www.rabbitmq.com/releases/rabbitmq-server/v3.5.1/rabbitmq-server-3.5.1-1.noarch.rpm

5.2安装 RabbitMQ Server

rpm --import

http://www.rabbitmq.com/rabbitmq-signing-key-public.asc

yum install rabbitmq-server-3.5.1-1.noarch.rpm

6　启动 RabbitMQ

6.1配置为守护进程随系统自动启动，root 权限下执行:

chkconfig rabbitmq-server on

6.2启动 rabbitMQ 服务

/sbin/service rabbitmq-server start

7 安装 Web 管理界面插件

7.1 安装命令

rabbitmq-plugins enable rabbitmq_management

7.2 安装成功后会显示如下内容

The following plugins have been enabled:

mochiweb

webmachine

rabbitmq_web_dispatch

amqp_client

rabbitmq_management_agent

rabbitmq_management

Plugin configuration has changed. Restart RabbitMQ for changes to take effect.

8 设置 RabbitMQ 远程 ip 登录

这里我们以创建个 oldlu 帐号，密码 123456 为例，创建一个账号并支持远程 ip 访问。

8.1创建账号

rabbitmqctl add_user oldlu 123456

8.2设置用户角色

rabbitmqctl set_user_tags oldlu administrator

8.3设置用户权限

rabbitmqctl set_permissions -p "/" oldlu ".*" ".*" ".*"

8.4设置完成后可以查看当前用户和角色**(**需要开启服务**)**

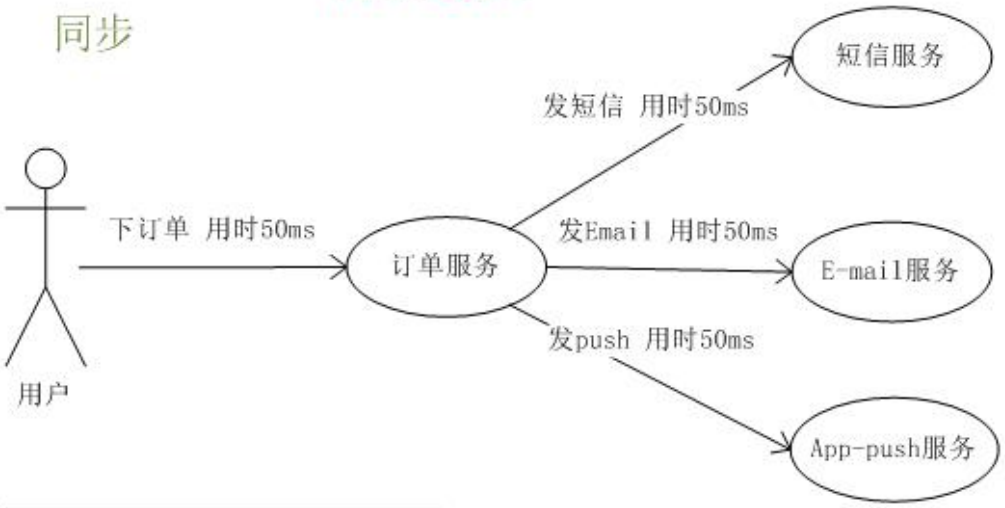
rabbitmqctl list_users

浏览器输入：serverip:15672。其中 serverip 是 RabbitMQ-Server 所在主机的 ip。

三、 为什么要使用 RabbitMQ？他解决了什么问题？

同步变异步

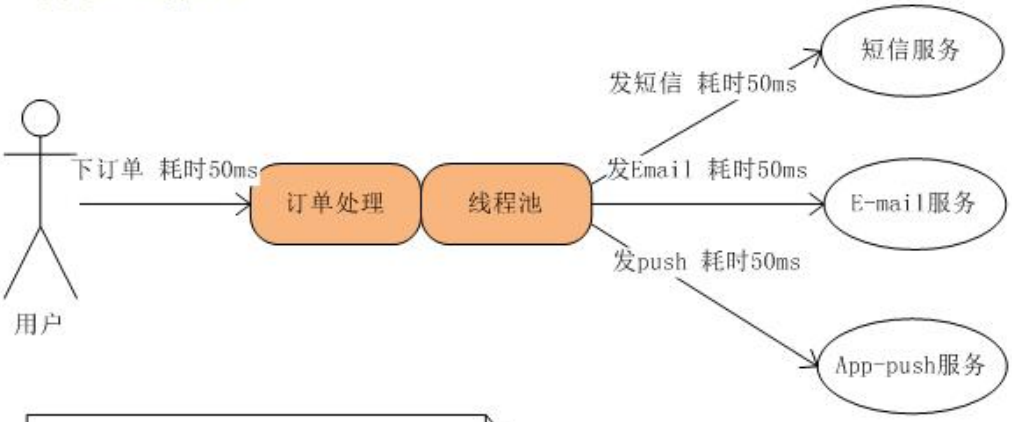
同步



缺点：耗时200ms，并且强耦合

同步变异步

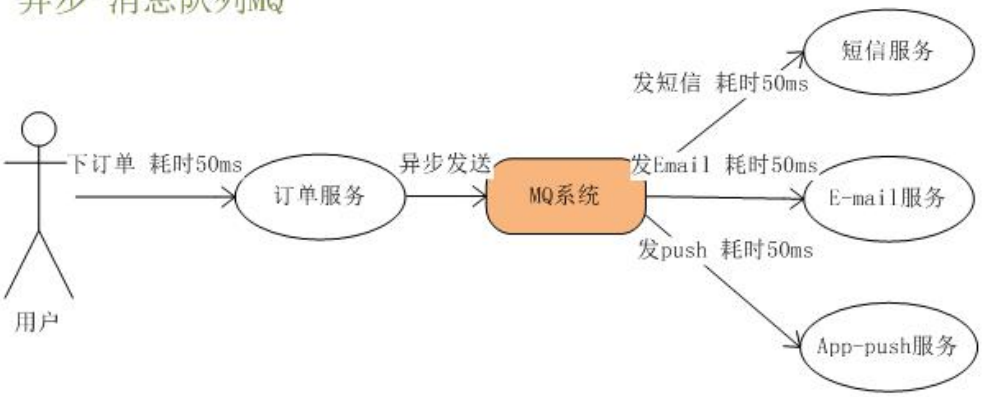
异步-线程池



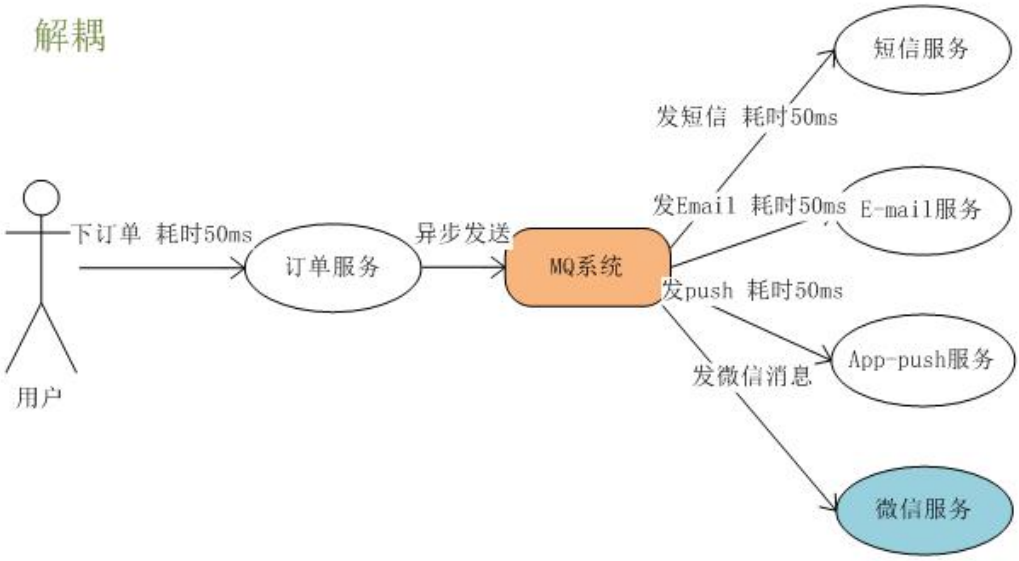
缺点：自己实现线程池，并且强耦合

同步变异步

异步-消息队列MQ

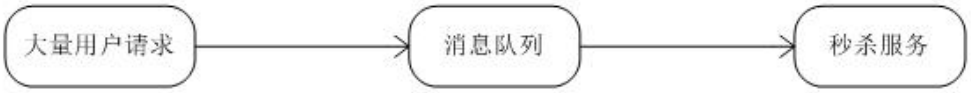


解耦



流量削锋

削锋



四、 消息队列基础知识。

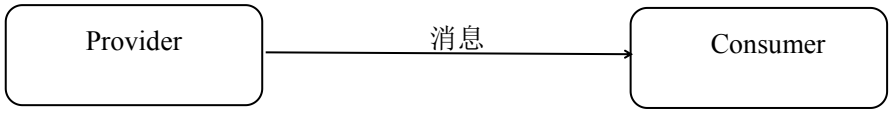
1 Provider

消息生产者，就是投递消息的程序。

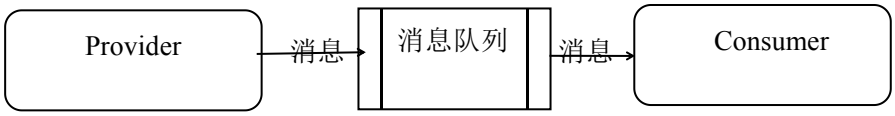
2 Consumer

消息消费者，就是接受消息的程序。

3 没有使用消息队列时消息传递方式



4 使用消息队列后消息传递方式



5 什么是队列？

队列就像存放了商品的仓库或者商店，是生产商品的工厂和购买商品的用户之间的中转站

6 队列里存储了什么？

在 rabbitMQ 中，信息流从你的应用程序出发，来到 Rabbitmq 的队列，所有信息可以只存储在一个队列中。队列可以存储很多信息，因为它基本上是一个无限制的缓冲区，前提是你的机器有足够的存储空间。

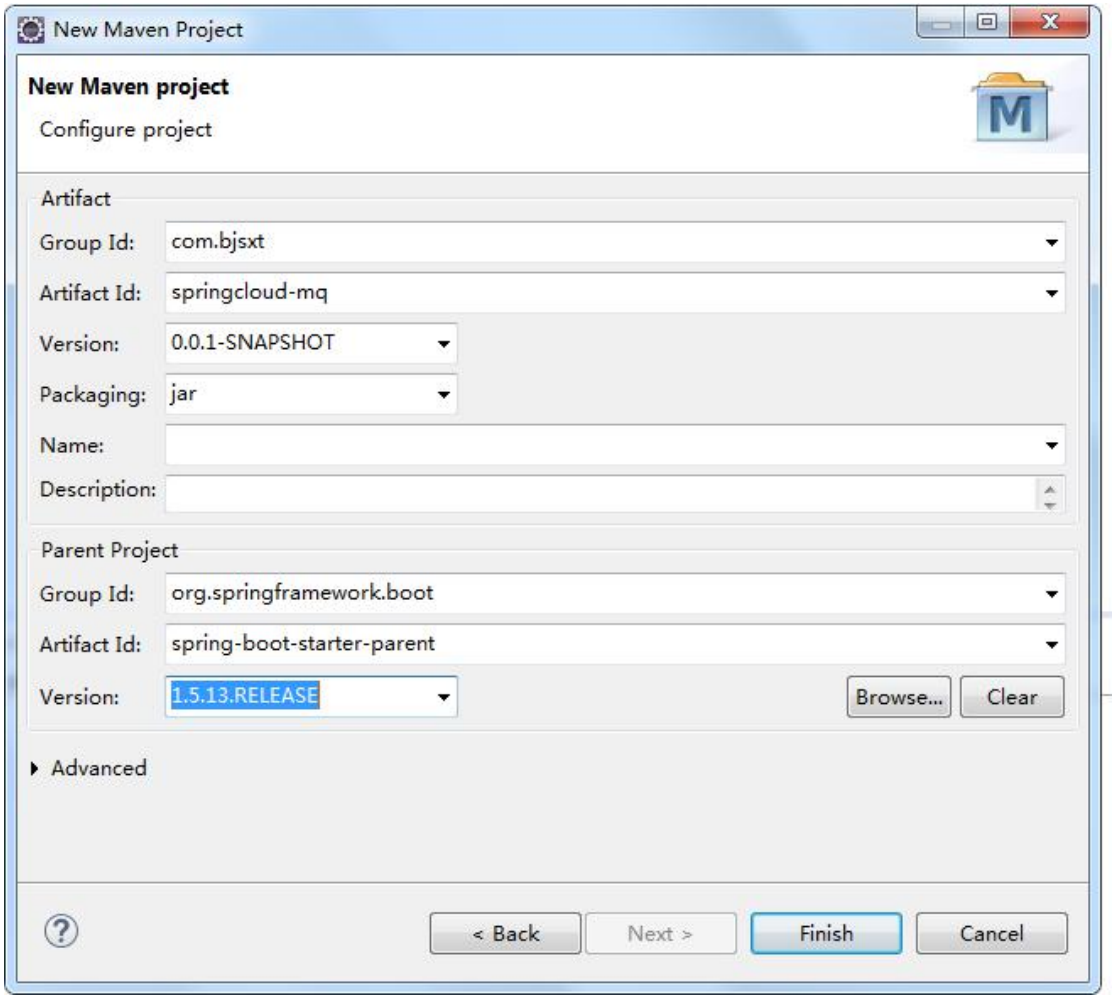
7 队列和应用程序的关系？

多个生产者可以将消息发送到同一个队列中，多个消息者也可以只从同一个队列接收数据。

五、 编写 RabbitMQ 的入门案例

1 搭建项目环境

1.1创建项目



1.2修改 pom 文件添加 RabbitMQ 坐标

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
```

1.3修改全局配置文件，添加 RabbitMQ 相关的配置

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
```

```
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456
```

2 编写代码

创建队列

```
/**
 * 创建消息队列
 * @author Administrator
 *
 */

@Configuration
public class QueueConfig {

    /**
     * 创建队列
     * @return
     */
    @Bean
    public Queue createQueue(){
        return new Queue("hello-queue");
    }
}
```

创建消息提供者

```
/**
 * 消息发送者
 * @author Administrator
 *
 */
@Component
public class Sender {

    @Autowired
    private AmqpTemplate rabbitAmqpTemplate;
    /*
     * 发送消息的方法
     */
    public void send(String msg){
        //向消息队列发送消息
```

```
            //参数一：队列的名称。
            //参数二：消息
            this.rabbitAmqpTemplate.convertAndSend("hello-queue",
msg);
        }
    }
```

消息接收者

```
    /**
     * 消息接收者
     * @author Administrator
     *
     */
    @Component
    public class Receiver {

        /**
         * 接收消息的方法。采用消息队列监听机制
         * @param msg
         */
        @RabbitListener(queues="hello-queue")
        public void process(String msg){
            System.out.println("receiver: "+msg);
        }
    }
```

启动类

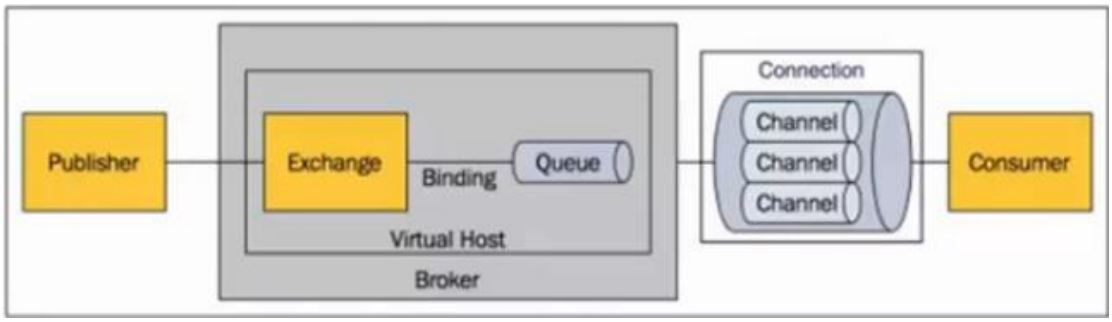
```
    @SpringBootApplication
    public class SpringbootServerApplication {

        public static void main(String[] args) {

    SpringApplication.run(SpringbootServerApplication.class,
args);
        }
    }
```

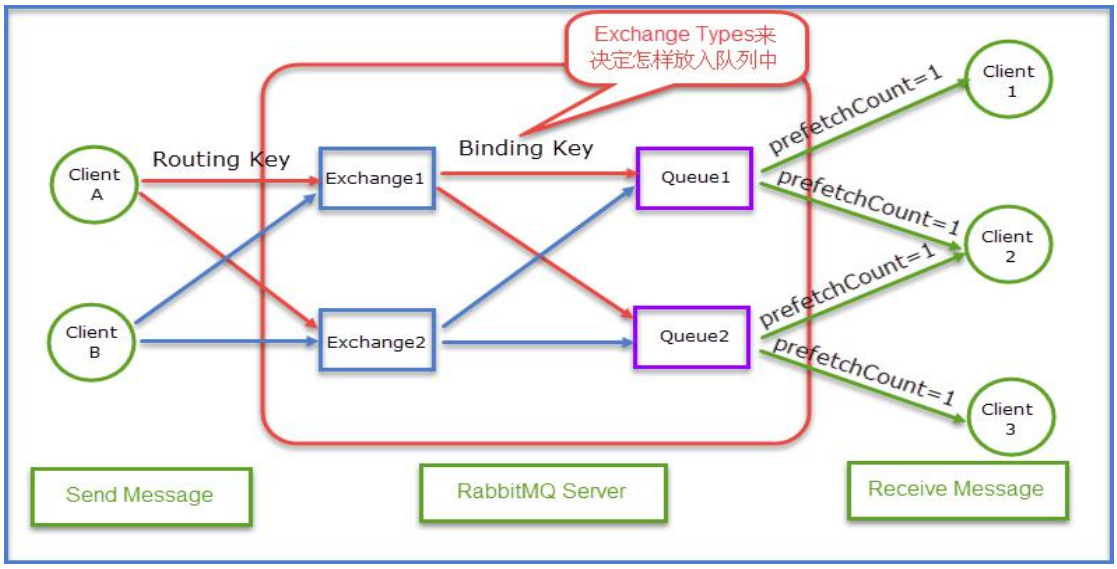
测试代码

```
    /**
     * 消息队列测试类
     * @author Administrator
```

```
 *
 */
@RunWith(SpringRunner.class)
@SpringBootTest(classes=SpringbootServerApplication.class)
public class QueueTest {

    @Autowired
    private Sender sender;

    /*
     * 测试消息队列
     */
    @Test
    public void test1(){
        this.sender.send("Hello RabbitMQ");
    }
}
```

六、 RabbitMQ 原理图





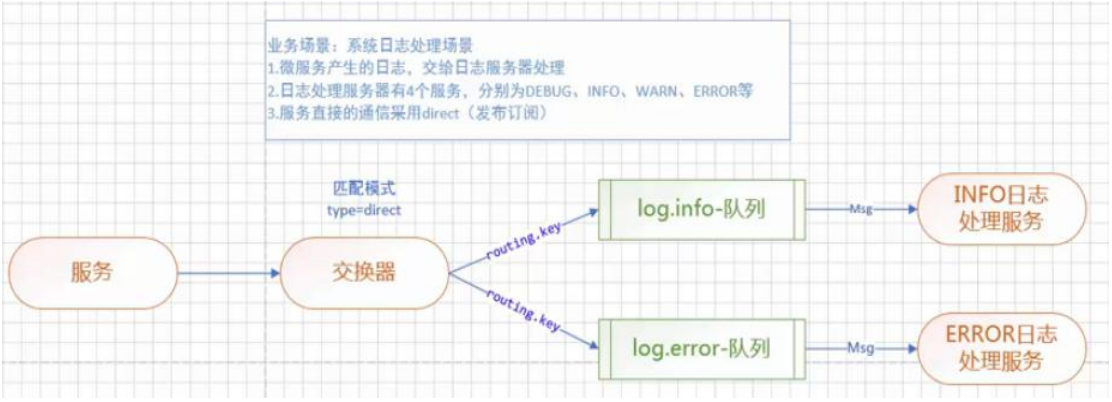
1.Message
消息。消息是不具名的，它由消息头消息体组成。消息体是不透明的，而消息头则由一系列可选属性组成，这些属性包括：routing-key(路由键)、priority(相对于其他消息的优先权)、delivery-mode(指出消息可能持久性存储)等。
2.Publisher
消息的生产者。也是一个向交换器发布消息的客户端应用程序。
3.Consumer
消息的消费者。表示一个从消息队列中取得消息的客户端应用程序。
4.Exchange
交换器。用来接收生产者发送的消息并将这些消息路由给服务器中的队列。 三种常用的交换器类型 1. direct(发布与订阅 完全匹配) 2. fanout(广播) 3. topic(主题，规则匹配)
5.Binding
绑定。用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。
6.Queue
消息队列。用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者链接到这个队列将其取走。
7.Routing-key
路由键。RabbitMQ 决定消息该投递到哪个队列的规则。 队列通过路由键绑定到交换器。 消息发送到 MQ 服务器时，消息将拥有一个路由键，即便是空的，RabbitMQ 也会将其和绑定使用的路由键进行匹配。 如果相匹配，消息将会投递到该队列。 如果不匹配，消息将会进入黑洞。
8.Connection
链接。指 rabbit 服务器和服务建立的 TCP 链接。
9.Channel
信道。 1，Channel 中文叫做信道，是 TCP 里面的虚拟链接。例如：电缆相当于 TCP，信道是一个独立光纤束，一条 TCP 连接上创建多条信道是没有问题的。 2，TCP 一旦打开，就会创建 AMQP 信道。 3，无论是发布消息、接收消息、订阅队列，这些动作都是通过信道完成的。
10.Virtual Host
虚拟主机。表示一批交换器，消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在链接时指定，RabbitMQ 默认的 vhost 是/
11.Borker
表示消息队列服务器实体。

交换器和队列的关系
交换器是通过路由键和队列绑定在一起的，如果消息拥有的路由键跟队列和交换器的路由键匹配，那么消息就会被路由到该绑定的队列中。 也就是说，消息到队列的过程中，消息首先会经过交换器，接下来交换器在通过路由键匹配分发消息到具体的队列中。 路由键可以理解为匹配的规则。
RabbitMQ 为什么需要信道？为什么不是 TCP 直接通信？
1. TCP 的创建和销毁开销特别大。创建需要 3 次握手，销毁需要 4 次分手。 2. 如果不用信道，那应用程序就会以 TCP 链接 Rabbit，高峰时每秒成千上万条链接会造成资源巨大的浪费，而且操作系统每秒处理 TCP 链接数也是有限制的，必定造成性能瓶颈。 3. 信道的原理是一条线程一条通道，多条线程多条通道同用一条 TCP 链接。一条 TCP 链接可以容纳无限的信道，即使每秒成千上万的请求也不会成为性能的瓶颈。

七、 Rabbit 交换器讲解

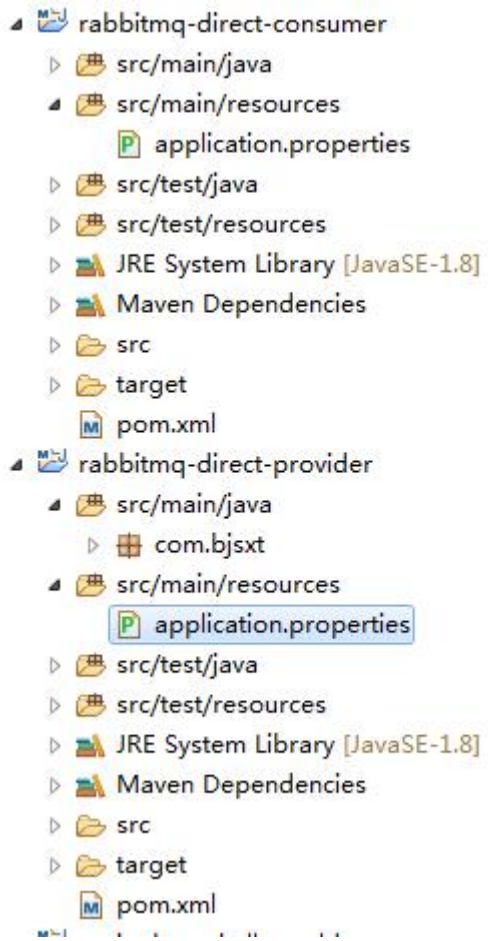
1 Direct 交换器(发布与订阅 完全匹配)

1.1 需求



1.2 搭建环境

1.2.1 创建项目



1.2.2 修改全局配置文件

修改 Consumer 的配置文件

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456

#设置交换器的名称
mq.config.exchange=log.direct

#info 队列名称
mq.config.queue.info=log.info
```

```
#info 路由键
mq.config.queue.info.routing.key=log.info.routing.key

#error 队列名称
mq.config.queue.error=log.error

#error 路由键
mq.config.queue.error.routing.key=log.error.routing.key
```

修改 Provider 的配置文件

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456


#设置交换器的名称
mq.config.exchange=log.direct

#info 路由键
mq.config.queue.info.routing.key=log.info.routing.key

#error 路由键
mq.config.queue.error.routing.key=log.error.routing.key

#error 队列名称
mq.config.queue.error=log.error
```

1.3 编写 Consumer

InfoReceiver

```
/**
 * 消息接收者
 * @author Administrator
 * @RabbitListener bindings:绑定队列
 * @QueueBinding  value:绑定队列的名称
 *                exchange:配置交换器
 *
 * @Queue value:配置队列名称
 *        autoDelete:是否是一个可删除的临时队列
 *
```

```
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
            bindings=@QueueBinding(

   value=@Queue(value="${mq.config.queue.info}",autoDelete="tr
ue"),

   exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.DIRECT),
                    key="${mq.config.queue.info.routing.key}"
            )
        )
    public class InfoReceiver {

       /**
        * 接收消息的方法。采用消息队列监听机制
        * @param msg
        */
       @RabbitHandler
       public void process(String msg){
          System.out.println("Info........receiver: "+msg);
       }
    }
```

ErrorReceiver

```
    /**
     * 消息接收者
     * @author Administrator
     * @RabbitListener bindings:绑定队列
     * @QueueBinding  value:绑定队列的名称
     *                exchange:配置交换器
     *
     * @Queue value:配置队列名称
     *        autoDelete:是否是一个可删除的临时队列
     *
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
```

```
            bindings=@QueueBinding(

   value=@Queue(value="${mq.config.queue.error}",autoDelete="true"),

   exchange=@Exchange(value="${mq.config.exchange}",type=ExchangeTypes.DIRECT),
                    key="${mq.config.queue.error.routing.key}"
            )
        )
    public class ErrorReceiver {

       /**
        * 接收消息的方法。采用消息队列监听机制
        * @param msg
        */
       @RabbitHandler
       public void process(String msg){
          System.out.println("Error.........receiver: "+msg);
       }
    }
```

1.4编写 Provider

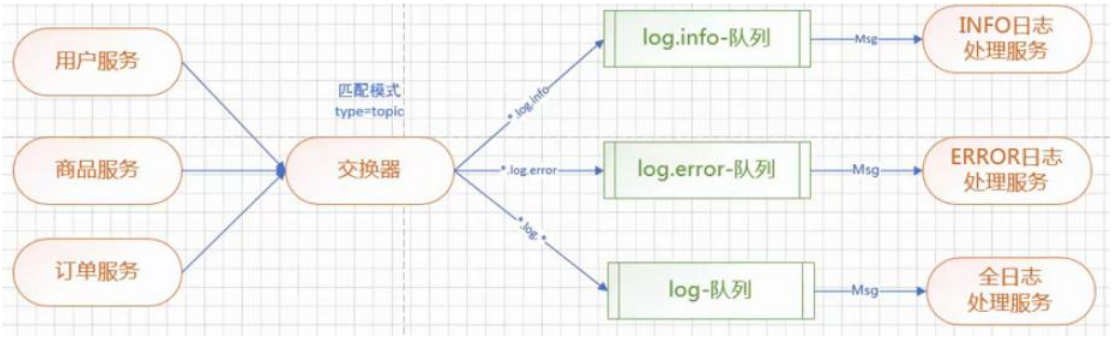
代码

```
    /**
     * 消息发送者
     * @author Administrator
     *
     */
    @Component
    public class Sender {

       @Autowired
       private AmqpTemplate rabbitAmqpTemplate;

       //exchange 交换器名称
       @Value("${mq.config.exchange}")
       private String exchange;

       //routingkey 路由键
       @Value("${mq.config.queue.error.routing.key}")
       private String routingkey;
```

```
        /*
         * 发送消息的方法
         */
        public void send(String msg){
            //向消息队列发送消息
            //参数一：交换器名称。
            //参数二：路由键
            //参数三：消息
            this.rabbitAmqpTemplate.convertAndSend(this.exchange,
this.routingkey, msg);
        }
    }
```

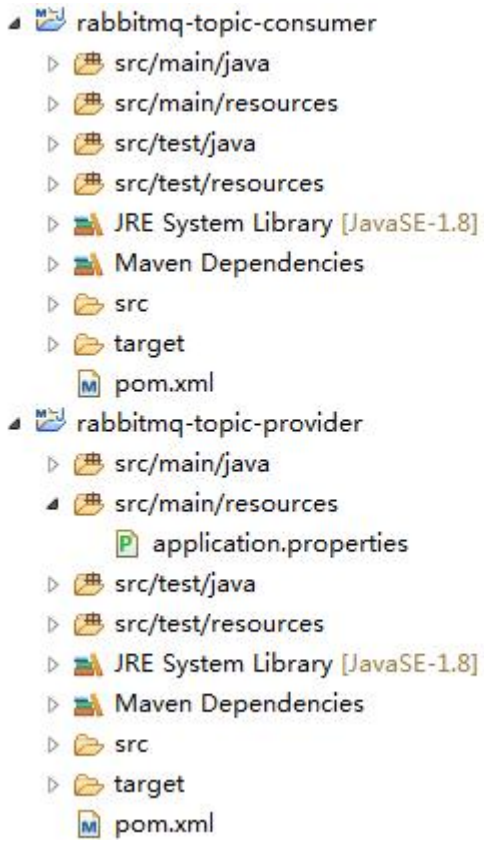
2 Topic 交换器(主题，规则匹配)

2.1 需求



2.2 搭建环境

2.2.1 创建项目



2.2.2 修改配置文件

Provider

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456


#设置交换器的名称
mq.config.exchange=log.topic
```

Consumer

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
```

```
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456

#设置交换器的名称
mq.config.exchange=log.topic

#info 队列名称
mq.config.queue.info=log.info

#error 队列名称
mq.config.queue.error=log.error

#log 队列名称
mq.config.queue.logs=log.all
```

2.3编写 Provider

UserSender

```
/**
 * 消息发送者
 * @author Administrator
 *
 */
@Component
public class UserSender {

    @Autowired
    private AmqpTemplate rabbitAmqpTemplate;

    //exchange 交换器名称
    @Value("${mq.config.exchange}")
    private String exchange;

    /*
     * 发送消息的方法
     */
    public void send(String msg){
        //向消息队列发送消息
        //参数一：交换器名称。
        //参数二：路由键
        //参数三：消息

this.rabbitAmqpTemplate.convertAndSend(this.exchange,"user.
```

```
log.debug", "user.log.debug....."+msg);

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"user.
log.info", "user.log.info....."+msg);

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"user.
log.warn","user.log.warn....."+msg);

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"user.
log.error", "user.log.error....."+msg);
      }
   }
```

ProductSender

```
/**
 * 消息发送者
 * @author Administrator
 *
 */
@Component
public class ProductSender {

   @Autowired
   private AmqpTemplate rabbitAmqpTemplate;

   //exchange 交换器名称
   @Value("${mq.config.exchange}")
   private String exchange;

   /*
    * 发送消息的方法
    */
   public void send(String msg){
      //向消息队列发送消息
      //参数一：交换器名称。
      //参数二：路由键
      //参数三：消息

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"produ
ct.log.debug", "product.log.debug....."+msg);

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"produ
ct.log.info", "product.log.info....."+msg);
```

```
    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"produ
ct.log.warn","product.log.warn....."+msg);

    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"produ
ct.log.error", "product.log.error....."+msg);
        }
    }
```

OrderSender

```
    /**
     * 消息发送者
     * @author Administrator
     *
     */
    @Component
    public class OrderSender {

        @Autowired
        private AmqpTemplate rabbitAmqpTemplate;

        //exchange 交换器名称
        @Value("${mq.config.exchange}")
        private String exchange;

        /*
         * 发送消息的方法
         */
        public void send(String msg){
            //向消息队列发送消息
            //参数一：交换器名称。
            //参数二：路由键
            //参数三：消息

    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"order
.log.debug", "order.log.debug....."+msg);

    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"order
.log.info", "order.log.info....."+msg);

    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"order
.log.warn","order.log.warn....."+msg);

    this.rabbitAmqpTemplate.convertAndSend(this.exchange,"order
```

```
.log.error", "order.log.error....."+msg);
        }
    }
```

2.4编写 Consumer

InfoReceiver

```
    /**
     * 消息接收者
     * @author Administrator
     * @RabbitListener bindings:绑定队列
     * @QueueBinding  value:绑定队列的名称
     *                exchange:配置交换器
     *
     * @Queue value:配置队列名称
     *        autoDelete:是否是一个可删除的临时队列
     *
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
            bindings=@QueueBinding(

    value=@Queue(value="${mq.config.queue.info}",autoDelete="tr
ue"),

    exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.TOPIC),
                    key="*.log.info"
            )
        )
    public class InfoReceiver {

        /**
         * 接收消息的方法。采用消息队列监听机制
         * @param msg
         */
        @RabbitHandler
        public void process(String msg){
            System.out.println("......Info........receiver:
"+msg);
        }
```

```
    }
```

ErrorReceiver

```
    /**
     * 消息接收者
     * @author Administrator
     * @RabbitListener bindings:绑定队列
     * @QueueBinding  value:绑定队列的名称
     *                exchange:配置交换器
     *
     * @Queue value:配置队列名称
     *        autoDelete:是否是一个可删除的临时队列
     *
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
              bindings=@QueueBinding(

    value=@Queue(value="${mq.config.queue.error}",autoDelete="t
rue"),

    exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.TOPIC),
                      key="*.log.error"
              )
            )
    public class ErrorReceiver {

        /**
         * 接收消息的方法。采用消息队列监听机制
         * @param msg
         */
        @RabbitHandler
        public void process(String msg){
            System.out.println("......Error........receiver:
"+msg);
        }
    }
```

LogsReceiver

```
/**
 * 消息接收者
 * @author Administrator
 * @RabbitListener bindings:绑定队列
 * @QueueBinding  value:绑定队列的名称
 *                exchange:配置交换器
 *
 * @Queue value:配置队列名称
 *        autoDelete:是否是一个可删除的临时队列
 *
 * @Exchange value:为交换器起个名称
 *           type:指定具体的交换器类型
 */
@Component
@RabbitListener(
         bindings=@QueueBinding(

value=@Queue(value="${mq.config.queue.logs}",autoDelete="tr
ue"),

exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.TOPIC),
                  key="*.log.*"
            )
        )
public class LogsReceiver {

   /**
    * 接收消息的方法。采用消息队列监听机制
    * @param msg
    */
   @RabbitHandler
   public void process(String msg){
      System.out.println("......All........receiver:
"+msg);
   }
}
```

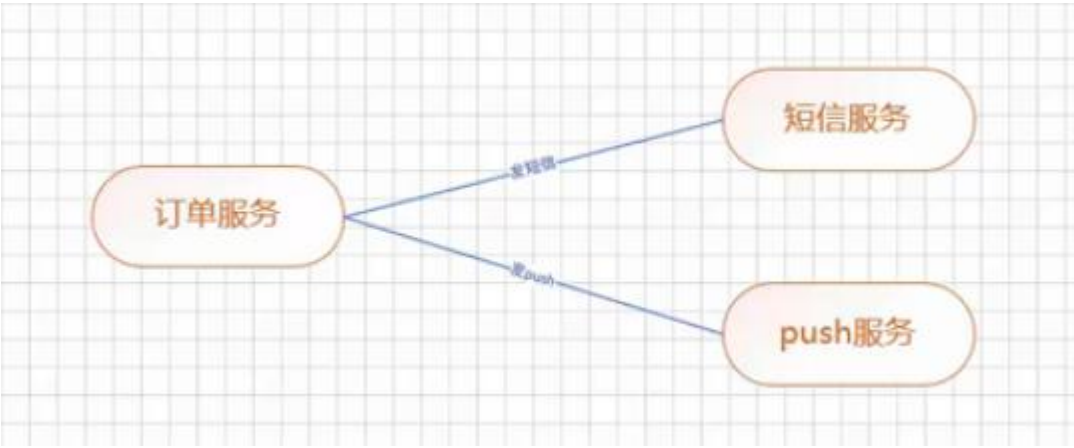
测试代码

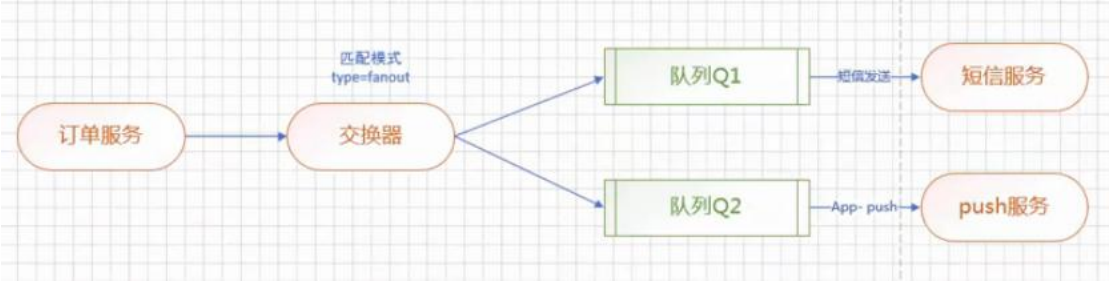
```
/**
 * 消息队列测试类
 * @author Administrator
```

```
 *
 */
@RunWith(SpringRunner.class)
@SpringBootTest(classes=SpringbootServerApplication.class)
public class QueueTest {

    @Autowired
    private UserSender usersender;

    @Autowired
    private ProductSender productsender;

    @Autowired
    private OrderSender ordersender;

    /*
     * 测试消息队列
     */
    @Test
    public void test1(){
        this.usersender.send("UserSender.....");
        this.productsender.send("ProductSender....");
        this.ordersender.send("OrderSender......");
    }
}
```

3 Fanout 交换器(广播)

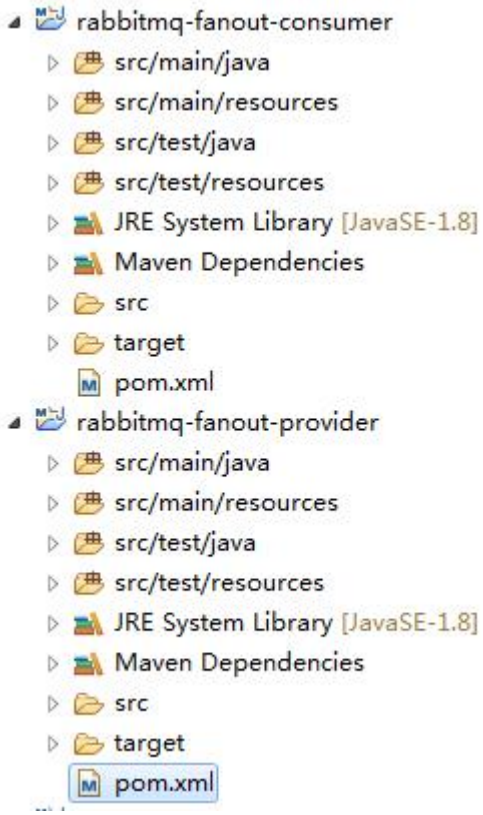
3.1 需求





3.2搭建环境

3.2.1创建项目



3.2.2修改配置文件

Consumer

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456

#设置交换器的名称
```

```
mq.config.exchange=order.fanout

#短信服务队列名称
mq.config.queue.sms=order.sms

#push 服务队列名称
mq.config.queue.push=order.push
```

Provider

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456


#设置交换器的名称
mq.config.exchange=order.fanout
```

3.3编写 Consumer

SmsReceiver

```
/**
 * 消息接收者
 * @author Administrator
 * @RabbitListener bindings:绑定队列
 * @QueueBinding  value:绑定队列的名称
 *                exchange:配置交换器
 *                key:路由键
 *
 * @Queue value:配置队列名称
 *        autoDelete:是否是一个可删除的临时队列
 *
 * @Exchange value:为交换器起个名称
 *           type:指定具体的交换器类型
 */
@Component
@RabbitListener(
         bindings=@QueueBinding(

value=@Queue(value="${mq.config.queue.sms}",autoDelete="tru
```

```
e"),

    exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.FANOUT)
                )
            )
    public class SmsReceiver {

        /**
         * 接收消息的方法。采用消息队列监听机制
         * @param msg
         */
        @RabbitHandler
        public void process(String msg){
            System.out.println("Sms........receiver: "+msg);
        }
    }
```

PushReceiver

```
    /**
     * 消息接收者
     * @author Administrator
     * @RabbitListener bindings:绑定队列
     * @QueueBinding  value:绑定队列的名称
     *                exchange:配置交换器
     *
     * @Queue value:配置队列名称
     *        autoDelete:是否是一个可删除的临时队列
     *
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
            bindings=@QueueBinding(

    value=@Queue(value="${mq.config.queue.push}",autoDelete="tr
ue"),

    exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.FANOUT)
                )
            )
    public class PushReceiver {
```

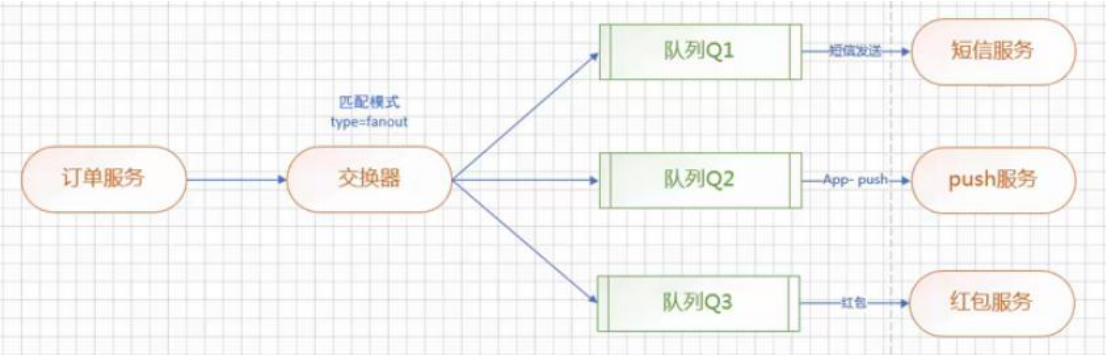
```
    /**
     * 接收消息的方法。采用消息队列监听机制
     * @param msg
     */
    @RabbitHandler
    public void process(String msg){
        System.out.println("Push.....receiver: "+msg);
    }
}
```

3.4编写 Provider

```
/**
 * 消息发送者
 * @author Administrator
 *
 */
@Component
public class Sender {

    @Autowired
    private AmqpTemplate rabbitAmqpTemplate;

    //exchange 交换器名称
    @Value("${mq.config.exchange}")
    private String exchange;

    /*
     * 发送消息的方法
     */
    public void send(String msg){
        //向消息队列发送消息
        //参数一：交换器名称。
        //参数二：路由键
        //参数三：消息

   this.rabbitAmqpTemplate.convertAndSend(this.exchange,"",
msg);
    }
}
```

八、 使用 RabbitMQ 实现松耦合设计

1 需求



2 搭建环境

2.1修改配置文件

```
spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456

#设置交换器的名称
mq.config.exchange=order.fanout

#短信服务队列名称
mq.config.queue.sms=order.sms

#push 服务队列名称
mq.config.queue.push=order.push

#红包服务队列名称
mq.config.queue.red=red
```

2.2添加 RedReceiver

```
/**
 * 消息接收者
 * @author Administrator
 * @RabbitListener bindings:绑定队列
 * @QueueBinding  value:绑定队列的名称
 *                exchange:配置交换器
```

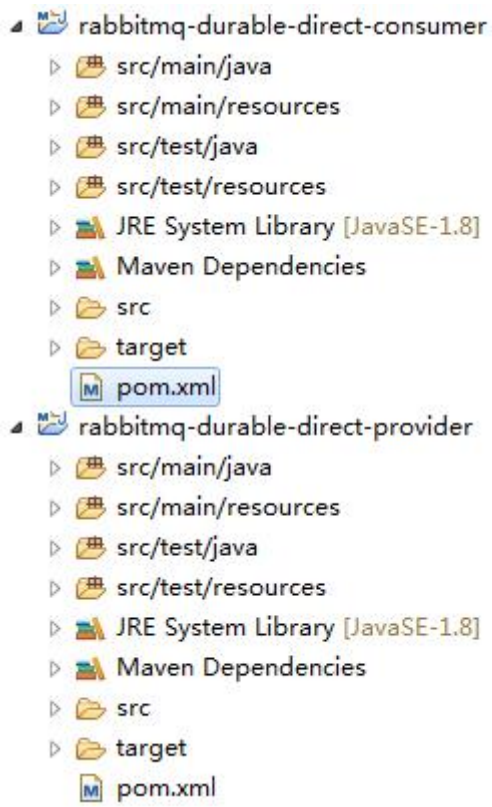
```
     *                key:路由键
     *
     * @Queue value:配置队列名称
     *        autoDelete:是否是一个可删除的临时队列
     *
     * @Exchange value:为交换器起个名称
     *           type:指定具体的交换器类型
     */
    @Component
    @RabbitListener(
            bindings=@QueueBinding(

    value=@Queue(value="${mq.config.queue.red}",autoDelete="tru
e"),

    exchange=@Exchange(value="${mq.config.exchange}",type=Excha
ngeTypes.FANOUT)
            )
        )
    public class RedReceiver {

        /**
         * 接收消息的方法。采用消息队列监听机制
         * @param msg
         */
        @RabbitHandler
        public void process(String msg){
            System.out.println("给用户发送 10 元红
包........receiver: "+msg);
        }
    }
```

九、 RabbitMQ 消息处理

1 RabbitMQ 的消息持久化处理

消息的可靠性是 RabbitMQ 的一大特色，那么 RabbitMQ 是如何保证消息可靠性的呢——消息持久化。

1.1创建项目



1.2autoDelete 属性

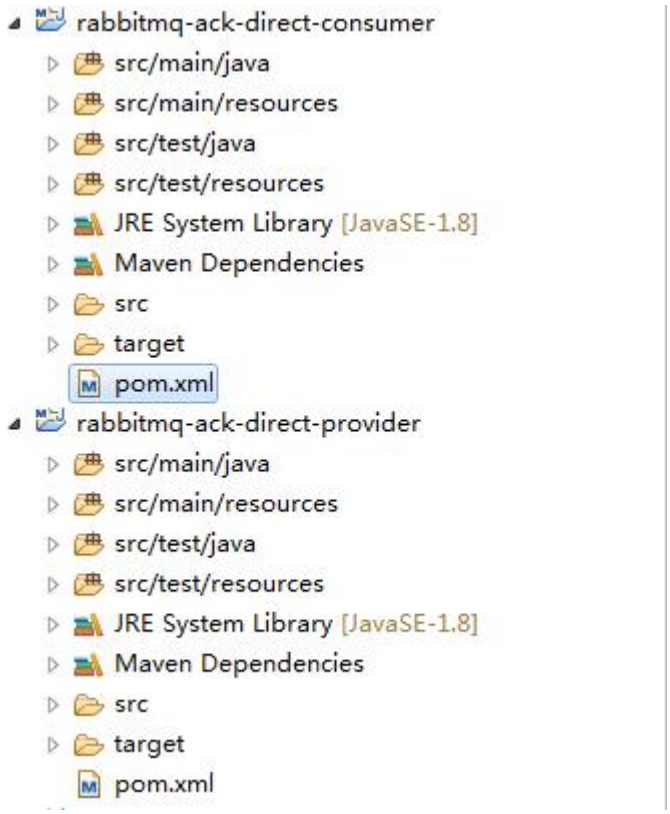
@Queue：当所有消费客户端连接断开后，是否自动删除队列 true：删除　false：不删除

@Exchange：当所有绑定队列都不在使用时，是否自动删除交换器 true：删除　false：不删除

2 RabbitMQ 中的消息确认 ACK 机制

1. 什么是消息确认ACK?
如果在处理消息的过程中，消费者的服务器在处理消息时出现异常，那可能这条正在处理的消息就没有完成消息消费，数据就会丢失。为了确保数据不会丢失，RabbitMQ支持消息确认-ACK.
2. ACK的消息确认机制
ACK机制是消费者从RabbitMQ收到消息并处理完成后，反馈给RabbitMQ，RabbitMQ收到反馈后才将此消息从队列中删除。 1. 如果一个消费者在处理消息出现了网络不稳定、服务器异常等现象，那么就不会有ACK反馈，RabbitMQ会认为这个消息没有正常消费，会将消息重新放入队列中。 2. 如果在集群的情况下：RabbitMQ会立即将这个消息推送给这个在线的其他消费者。这种机制保证了在消费者服务端故障的时候，不丢失任何消息和任务。 3. 消息永远不会从RabbitMQ中删除：只有当消费者正确发送ACK反馈，RabbitMQ确认收到后，消息才会从RabbitMQ服务器的数据中删除。 4. 消息的ACK确认机制默认是打开的。
3. ACK机制的开发注意事项
如果忘记了ACK,那么后果很严重。当Consumer退出时，Message会一直重新分发。然后RabbitMQ会占用越来越多的内存，由于RabbitMQ会长时间运行，因此这个“内存泄漏”是致命的。

2.1创建项目



修改 Consusmer 配置文件解决 ACK 反馈问题

```
spring.application.name=springcloud-mq

spring.rabbitmq.host=192.168.70.131
spring.rabbitmq.port=5672
spring.rabbitmq.username=oldlu
spring.rabbitmq.password=123456

#设置交换器的名称
mq.config.exchange=log.direct

#info 队列名称
mq.config.queue.info=log.info

#info 路由键
mq.config.queue.info.routing.key=log.info.routing.key

#error 队列名称
mq.config.queue.error=log.error

#error 路由键
mq.config.queue.error.routing.key=log.error.routing.key

#开启重试
spring.rabbitmq.listener.retry.enabled=true
#重试次数，默认为 3 次
spring.rabbitmq.listener.retry.max-attempts=5
```