Please visit website: http://cxyroad.com

利用SpringBoot+rabbitmq 实现邮件异步发送，保证100%投递成功

==========

在之前的文章中，我们详细介绍了 SpringBoot 整合 mail 实现各类邮件的自动推送服务。

但是这类服务通常不稳定，当出现网络异常的时候，会导致邮件推送失败。

本篇文章将介绍另一种高可靠的服务架构，实现邮件 100% 被投递成功。类似的短信自动发送等服务也大体相同。

一、先来一张流程图

本文内容主要围绕这个流程图展开，利用 RabbitMQ 消息队列来实现邮件 100% 被投递，内容涵盖了 RabbitMQ 很多知识点，如：

* 生产者和消费者模型
* 消息发送确认机制
* 消费确认机制
* 消息的重新投递

二、实现思路

* 1.准备一台 Linux 服务器，并安装 RabbitMQ
* 2.开放 QQ 邮箱或者其它邮箱授权码，用于发送邮件
* 3.创建邮件发送项目并编写代码
* 4.发送邮件测试
* 5.消息发送失败处理

三、环境准备

获取邮箱授权码的目的，主要是为了通过代码进行发送邮件，例如 QQ 邮箱授权码获取方式，如下图：

点击【开启】按钮，然后发送短信，即可获取授权码，该授权码就是配置文件`spring.mail.password`需要的密码！

四、项目介绍

* springboot版本：2.1.5.RELEASE
* RabbitMQ版本：3.6.5
* SendMailUtil：发送邮件工具类
* ProduceServiceImpl：生产者，发送消息
* ConsumerMailService：消费者，消费消息，发送邮件

五、代码实现

5.1、创建项目

在 IDEA 下创建一个名称为`smail`的 Springboot 项目，`pom`文件中加入`amqp`和`mail`。

```
<dependencies>
    <!--spring boot核心-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
```

```
    </dependency>
    <!--spring boot 测试-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <!--springmvc web-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!--开发环境调试-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
    </dependency>
    <!--mail 支持-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <!--amqp 支持-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
    <!-- commons-lang3 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.4</version>
    </dependency>
    <!--lombok-->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.10</version>
    </dependency>
</dependencies>

```

5.2、配置rabbitMQ、mail

在`application.properties`文件中，配置`amqp`和`mail`！

```
#rabbitmq
spring.rabbitmq.host=192.168.0.103
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
# 开启confirms回调 P -> Exchange
spring.rabbitmq.publisher-confirms=true
# 开启returnedMessage回调 Exchange -> Queue
spring.rabbitmq.publisher-returns=true
# 设置手动确认(ack) Queue -> C
spring.rabbitmq.listener.simple.acknowledge-mode=manual
spring.rabbitmq.listener.simple.prefetch=100

# mail
spring.mail.default-encoding=UTF-8
spring.mail.host=smtp.qq.com
spring.mail.username=xxxx@qq.com
spring.mail.password=获取的邮箱授权码
spring.mail.from=xxxx@qq.com
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
```

其中，`spring.mail.password`第四步中获取的授权码，同时`username`和`from`要一致！

5.3、RabbitConfig配置类

```
@Configuration
@Slf4j
public class RabbitConfig {

    // 发送邮件
    public static final String MAIL_QUEUE_NAME = "mail.queue";
    public static final String MAIL_EXCHANGE_NAME = "mail.exchange";
    public static final String MAIL_ROUTING_KEY_NAME =
"mail.routing.key";
```

```
    @Autowired
    private CachingConnectionFactory connectionFactory;

    @Bean
    public RabbitTemplate rabbitTemplate() {
        RabbitTemplate rabbitTemplate = new
RabbitTemplate(connectionFactory);
        rabbitTemplate.setMessageConverter(converter());

        // 消息是否成功发送到Exchange
        rabbitTemplate.setConfirmCallback((correlationData, ack, cause) -
> {
            if (ack) {
                log.info("消息成功发送到Exchange");
            } else {
                log.info("消息发送到Exchange失败, {}, cause: {}",
correlationData, cause);
            }
        });

        // 触发setReturnCallback回调必须设置mandatory=true, 否则
Exchange没有找到Queue就会丢弃掉消息, 而不会触发回调
        rabbitTemplate.setMandatory(true);
        // 消息是否从Exchange路由到Queue, 注意: 这是一个失败回调, 只有消
息从Exchange路由到Queue失败才会回调这个方法
        rabbitTemplate.setReturnCallback((message, replyCode, replyText,
exchange, routingKey) -> {
            log.info("消息从Exchange路由到Queue失败: exchange: {}, route:
{}, replyCode: {}, replyText: {}, message: {}", exchange, routingKey,
replyCode, replyText, message);
        });

        return rabbitTemplate;
    }

    @Bean
    public Jackson2JsonMessageConverter converter() {
        return new Jackson2JsonMessageConverter();
    }

    @Bean
    public Queue mailQueue() {
        return new Queue(MAIL_QUEUE_NAME, true);
    }

    @Bean
    public DirectExchange mailExchange() {
        return new DirectExchange(MAIL_EXCHANGE_NAME, true, false);
```

```
    }

    @Bean
    public Binding mailBinding() {
        return
BindingBuilder.bind(mailQueue()).to(mailExchange()).with(MAIL_ROUTING
_KEY_NAME);
    }
}
```

5.4、Mail 邮件实体类

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Mail {

    // 目标邮箱
    private String to;

    // 标题
    private String title;

    // 正文
    private String content;

    // 消息ID
    private String msgId;
}
```

5.5、SendMailUtil邮件发送类

```
@Component
@Slf4j
public class SendMailUtil {

    @Value("${spring.mail.from}")
```

```
    private String from;

    @Autowired
    private JavaMailSender mailSender;

    /**
     * 发送简单邮件
     *
     * @param mail
     */
    public boolean send(Mail mail) {
        String to = mail.getTo();// 目标邮箱
        String title = mail.getTitle();// 邮件标题
        String content = mail.getContent();// 邮件正文

        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(from);
        message.setTo(to);
        message.setSubject(title);
        message.setText(content);

        try {
            mailSender.send(message);
            log.info("邮件发送成功");
            return true;
        } catch (MailException e) {
            log.error("邮件发送失败, to: {}, title: {}", to, title, e);
            return false;
        }
    }
}
```

5.6、ProduceServiceImpl 生产者类

```
@Service
public class ProduceServiceImpl implements ProduceService {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Override
    public boolean send(Mail mail) {
        //创建uuid
```

```
        String msgId = UUID.randomUUID().toString().replaceAll("-", "");
        mail.setMsgId(msgId);

        //发送消息到rabbitMQ
        CorrelationData correlationData = new CorrelationData(msgId);
rabbitTemplate.convertAndSend(RabbitConfig.MAIL_EXCHANGE_NAME,
RabbitConfig.MAIL_ROUTING_KEY_NAME,
MessageHelper.objToMsg(mail), correlationData);

        return true;
    }
}
```

5.7、ConsumerMailService 消费者类

```
@Component
@Slf4j
public class ConsumerMailService {

    @Autowired
    private SendMailUtil sendMailUtil;

    @RabbitListener(queues = RabbitConfig.MAIL_QUEUE_NAME)
    public void consume(Message message, Channel channel) throws
IOException {
        //将消息转化为对象
        String str = new String(message.getBody());
        Mail mail = JsonUtil.strToObj(str, Mail.class);
        log.info("收到消息: {}", mail.toString());

        MessageProperties properties = message.getMessageProperties();
        long tag = properties.getDeliveryTag();

        boolean success = sendMailUtil.send(mail);
        if (success) {
            channel.basicAck(tag, false);// 消费确认
        } else {
            channel.basicNack(tag, false, true);
        }
    }
}
```

5.8、TestController 控制层类

```
@RestController
@RequestMapping(”/test”)
@Slf4j
public class TestController {

    @Autowired
    private ProduceService testService;

    @PostMapping(”send”)
    public boolean sendMail(Mail mail) {
        return testService.send(mail);
    }
}

```

六、测试服务

启动 SpringBoot 服务之后，用 postman 模拟请求接口。

查看控制台信息。

查询接受者邮件信息。

![](https://p6-xtjj-sign.byteimg.com/tos-cn-i-

73owjymdk6/fded246e569f4a61b63e0e6b478b7f65~tplv-73owjymdk6-watermark.image?rk3s=f64ab15b&x-expires=1721958627&x-signature=1wiEU5MAOw%2F0yzhZkSiDyZsYVrU%3D)

邮件发送成功！

七、小结

本文主要是通过发送邮件这个业务案例，来讲解 Springboot 与 rabbitMQ 技术的整合和使用！

使用了 rabbitMQ 的手动确认模式，当开启了之后，必须手动调用 ack 或者 nack 方法，否则消息会一直存储在 rabbitMQ 服务器中。

项目源代码地址：[spring-boot-example-smail](http://cxyroad.com/ ”https://gitee.com/pzblogs/spring-boot-example-demo”)

原文链接: https://juejin.cn/post/7389650655428788234