链滴

Rabbit 安装踩坑及 Springboot 和 Rabbit MQ 整合（Direct 模式、Topic 模式、Fanout 模式）

作者：HuixiaZhang
原文链接：https://ld246.com/article/1543392920760
来源网站：链滴

一： RabbitMQ

1.由于RabbitMQ是由Elang语言开发的，所以安装RabbitMQ 需要首先安装 Elang 环境(需要添加入境变量才可以正常启动RabbitMQ)。

2.在安装过程中需要注意ERlang和RabbitMQ的版本对应，如果版本不正确的话也会导致RabbitMQ装失败。

3.当使用springboot去远程连接rabbitMQ时，不能使用默认的guest用户，官方文档也写到，该用引用于本地连接使用。

4.RabbitMQ有web的UI界面，不过需要启动（rabbitmq-plugins.bat enable rabbitmq_management命令），windows\linux下如果启动不成功，可能是没有在管理员的权限下进行运行。开启和关闭RbbitMQ的命令分别为net start\stop RabbitMQ。

二：Springboot和RabbitMQ整合

创建maven项目并导入需要的jar包

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
    <dependency>
        <groupId>com.rabbitmq</groupId>
        <artifactId>amqp-client</artifactId>
        <version>3.6.5</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-test</artifactId>
    </dependency>
</dependencies>
```

并配置好application.properties:

```
spring.application.name=Spring-boot-rabbitmq

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=#username
spring.rabbitmq.password=#password
```

1.Direct模式：最简单的消息队列模式

RabbitConfig.java：RabbitMQ的配置类，在这里可以定义一个消息队列。

```
@Configuration
public class RabbitConfig {

    @Bean
    public Queue Queue(){
        // ("hello") 值得是创建这个队列的名字
        return new Queue("hello");
```

```
    }
}
```

HelloSender.java: 发送端的编写

```
@Service
public class HelloSender {

    @Autowired
    private AmqpTemplate rabbitTemplate;

    public void send(int i ){
        String context = "hello" + new Date();
        System.out.println("Sender:" + context + "******" +  i);
        // 这里需要指定发送的消息队列
        this.rabbitTemplate.convertAndSend("hello",context);
    }
}
```

HelloReceiver.java: 接收端的编写

```
@Component
@RabbitListener(queues = "hello")
public class HelloReceiver {

    @RabbitHandler
    public void process1(String hello){
        System.out.println("Receiver1:" + hello);
    }

}
```

TestRabbitMQ：对于队列的发送测试

```
@SpringBootTest
@RunWith(SpringJUnit4ClassRunner.class)
public class TestRabbitMQ {
    @Autowired
    private HelloSender helloSender;

    @Test
    public void testRabbit(){
        helloSender.send(1);
    }
}
```

最后编写springboot的项目启动类后启动，会看到发送端和接收端的数据。

2.Topic模式：可以自定义接收队列的模式，是最灵活的一种。

RabbitConfig中创建两个消息队列，把其都绑定在同一个交换机中，通过with来限定满足什么条件的候接收器触发。

```
@Configuration
public class RabbitConfig {

    // 创建 queue
    @Bean(name = "message")
    public Queue Queue(){
        return new Queue("topic.message");  // topic.message ，是 rounting-key 匹配规则
    }

    @Bean(name = "messages")
    public Queue queueMessages(){
        return new Queue("topic.messages");
    }

    /**
     * 创建交换机
     * @return
     */
    @Bean
    public TopicExchange exchange(){
        return new TopicExchange("exchange");
    }

    // 根据绑定规则将队列绑定到相应的交换机上
    @Bean
    public Binding bindingExchangeMessage(@Qualifier("message") Queue queueMessage, To
icExchange exchange){
        return BindingBuilder.bind(queueMessage).to(exchange).with("topic.message");
    }

    @Bean
    public Binding bindingExchangeMessages(@Qualifier("messages") Queue queueMessages,
TopicExchange exchange){
        // * 表示一个词， # 表示零个或多个词
        return BindingBuilder.bind(queueMessages).to(exchange).with("topic.#");
    }
}
```

当发送的消息是queueMessage的时候两个接收器都会接受，而当发送的消息是queueMessages的候只有第二个接收器会接受。

3.Fanout模式：意为广播模式，可以同时分发到绑定交换机的所有接收器。

RabbitMQ中编写了3个消息队列，把其都绑定在交换机上。

```
@Configuration
/**
 * 广播模式 Fanout
 */
public class FanoutRabbitConfig {

    @Bean
```

```
    public Queue AMessage(){
        return new Queue("fanout.A");
    }

    @Bean
    public Queue BMessage(){
        return new Queue("fanout.B");
    }

    @Bean
    public Queue CMessage(){
        return new Queue("fanout.C");
    }


    // 构建交换器
    @Bean
    FanoutExchange fanoutExchange(){
        return new FanoutExchange("fanoutExchange");
    }

    // 把 A,B,C都绑到 fanoutExchange 上
    @Bean
    Binding bindingExchangeA(Queue AMessage , FanoutExchange fanoutExchange){
        return BindingBuilder.bind(AMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeB(Queue BMessage , FanoutExchange fanoutExchange){
        return BindingBuilder.bind(BMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeC(Queue CMessage , FanoutExchange fanoutExchange){
        return BindingBuilder.bind(CMessage).to(fanoutExchange);
    }
}
```

运行的结果是当发送消息的时候交换器会将其分发到3个接收器上