一、安装MQ

1.下载镜像

docker pull rabbitmq:3-management

2.运行镜像

docker run -d -p 5672:5672 -p 15672:15672 --name myrabbitmq 500d74765467

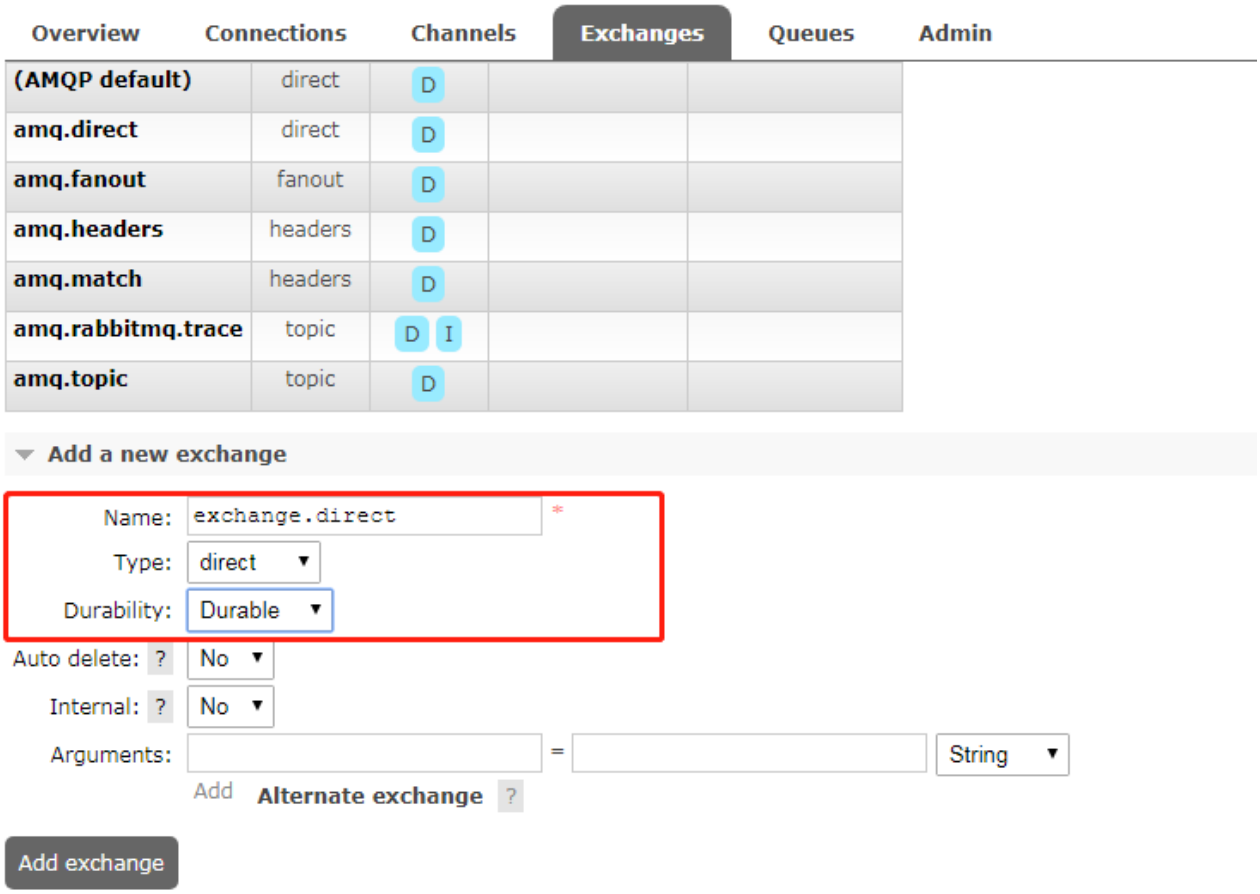
- 5672为rabbitmq的默认端口号
- 15672为rabbitmq控制管理台的默认端口号
- -d表示后台启动

3.登录管理控制台

地址：http://47.105.103.45:15672/

账号/密码 guest/guest

4.创建交换器



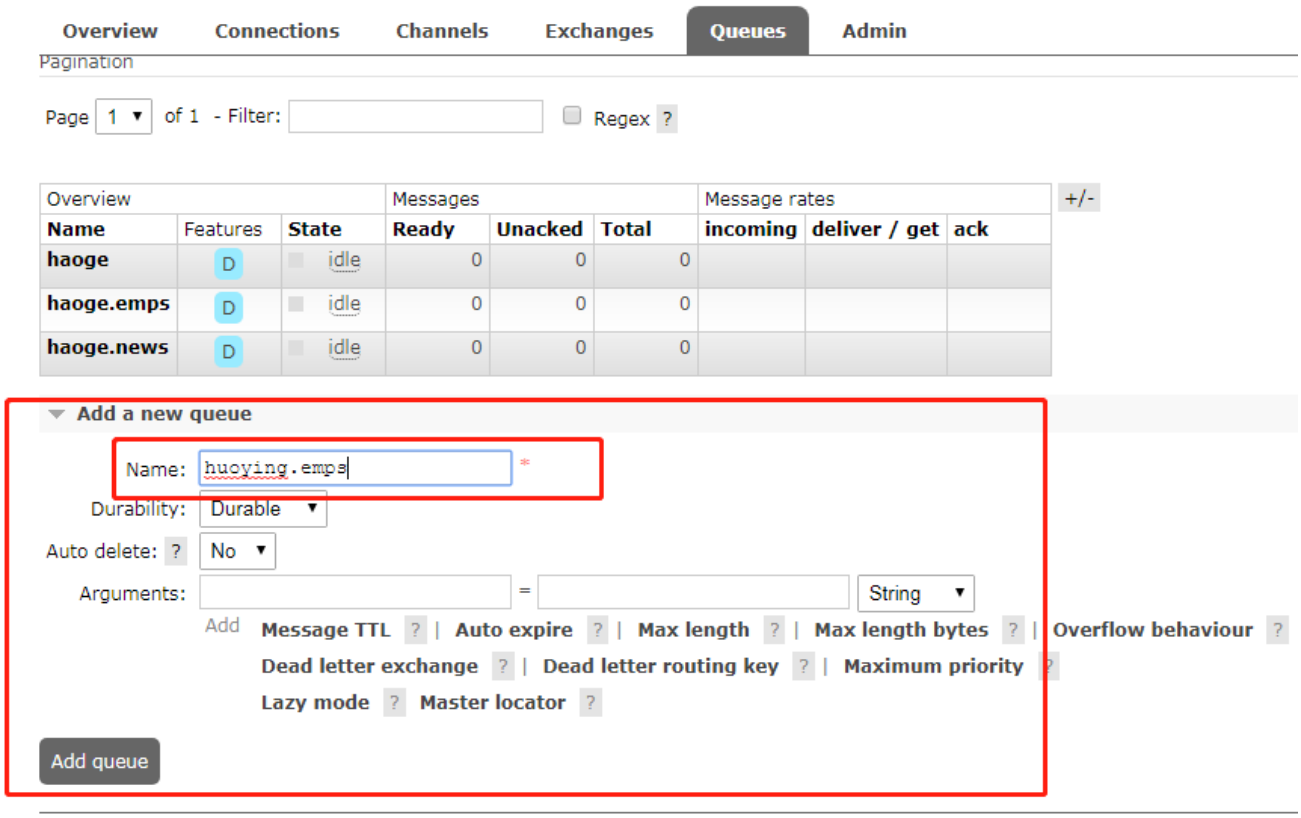
分别创建 exchange.direct exchange.fanout exchange.topic 三个交换器

- Durability表示该交换器是永久存在，下次启动MQ交换器还存在
- direct类型的交换器：消息中的路由键（routing key）如果和 Binding 中的 binding key 一致， 交换器就将消息发到对应的队列中。路由键与队 列名完全匹配
- fanout 每个发到 fanout 类型交换器的消息都会分到所 有绑定的队列上去。
- topic 交换器通过模式匹配分配消息的路由键属 性，将路由键和某个模式进行匹配，此时队列 需要绑定到一个模式

上。支持两个通配符：#和*， # 匹配0个或者多个单词 ，* 匹配一个单词

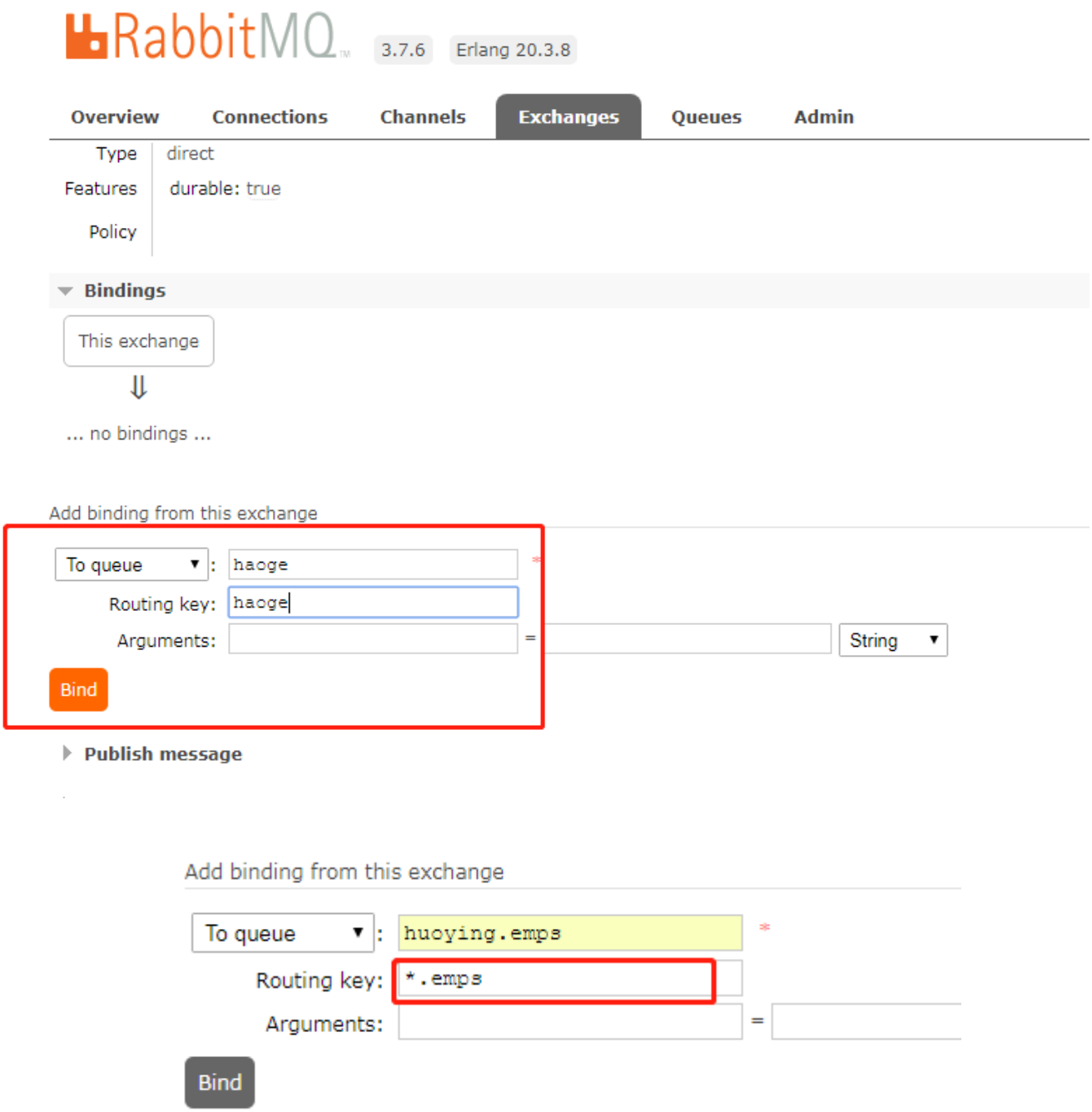
5.创建队列queues

分别创建四个队列：haoge haoge.news haoge.emps huoying.emps



6.将交换器和队列进行绑定

分别将交换器和队列进行绑定



7.进行测试

点进去每个exchange,在publish message模块分别设置路由键和消息内容，然后发送即可。

二、springboot整合rabbitMQ

1.POM依赖

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

2.配置文件

```yaml
spring:
  rabbitmq:
    addresses: 47.105.103.45
    username: guest
    password: guest
    port: 5672
```

3.自动配置原理

- RabbitMQ自动配置类RabbitAutoConfiguration
- RabbitAutoConfiguration帮我们自动配置了ConnectionFactory
- RabbitProperties封装了所有关于RabbitMQ的配置
- 自动配置了RabbitTemplate，可以给RabbitMQ发送和接收消息
- amqpAdmin：RabbitMQ系统管理功能组件

4.RabbitTemplate测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringbootRabbitMqApplicationTests
{

    @Autowired
    RabbitTemplate rabbitTemplate;

    @Test
    public void contextLoads() {
//      这个方法需要我们自己定义message，定义消息体内容和消息头
//      rabbitTemplate.send(exchange, routingKey, message);
//      object默认作为消息体，只需要传入发送的对象，自动序列化发送给rabbitMQ
        HashMap<String, Object> map = new HashMap<String,Object>();
        map.put("msg", "这是第一个消息");

map.put("data",Arrays.asList("123",true,"hello World"));

rabbitTemplate.convertAndSend("exchange.direct", "haoge.news", map);
    }
```

```
18      @Test
19      public void testTeceive() {
20          //接受消息
21          Object object =
rabbitTemplate.receiveAndConvert("haoge.news");
22          System.out.println(object.getClass());
23          System.out.println(object);
24      }
25      //广播模式
26      @Test
27      public void sendMsg() {
28          //接受消息
29          HashMap<String, Object> map = new
HashMap<String,Object>();
30          map.put("msg", "这是第一个消息");
31
map.put("data",Arrays.asList("123",true,"hello
World"));
32
rabbitTemplate.convertAndSend("exchange.fanout",
 "", map);
33      }
34  }
```

编写自定义的MessageConverter

```java
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
//自定义MessageConverter,使之序列化的时候使用json进行序列化
//这个MessageConverter会自动生效
@Configuration
public class MyAMQPConfig {
    @Bean
    public MessageConverter messageConverter(){
        return new Jackson2JsonMessageConverter();
    }
}
```

三、使用监听器接收消息

1.开启基于注解的RabbitMQ模式

```java
@EnableRabbit//开启基于注解的RabbitMQ模式
@SpringBootApplication
public class SpringbootRabbitMqApplication {
    public static void main(String[] args) {

SpringApplication.run(SpringbootRabbitMqApplicati
on.class, args);
    }
}
```

2.使用@RabbitListener监听队列

```java
@Service
public class BookService {
    // RabbitMQ监听huoying.emps队列，如果有消息，
就会自动接收
    @RabbitListener(queues = "huoying.emps")
    public void receive(Message message) {
        System.out.println("123" +
message.getBody());
        System.out.println("123" +
message.getMessageProperties());
    }

    // RabbitMQ监听haoge.news队列，如果有消息，就会
自动接收
    // 然后将接收到的消息直接转换成Book对象
    @RabbitListener(queues = "haoge.news")
```

```
13      public void receiveBook(Book book) {
14          System.out.println("123" + book);
15      }
16  }
17
```

四、AmqpAdmin的使用

可以使用AmqpAdmin创建queue,exchange,bingding

使用例子：

```
1       @Autowired
2       AmqpAdmin amqpAdmin;
3       @Test
4       public void useAmqpAdmin() {
5           //使用AmqpAdmin创建一个Queue
6           amqpAdmin.declareQueue(new
    Queue("amqp.queue", true));
7           //Exchange是一个接口，我们创建他的一个实现
    类
8           DirectExchange directExchange = new
    DirectExchange("amqp.exchange");
9           //创建Exchange
10
    amqpAdmin.declareExchange(directExchange);
11          //创建绑定规则
```

```
12          Binding binding = new
Binding("amqp.queue",
Binding.DestinationType.QUEUE, "amqp.exchange",
"admin.haoge", null);
13          amqpAdmin.declareBinding(binding);
14      }
```

五、PPT内容

1.消息队列的应用场景

```
1  - 异步处理
```

对于并不需要马上返回结果的处理过程，我们将其写入到消息队列中，然后立即对用户返回结果，增强用户体验。

如：用户注册之后，我们还需要给用户发一份邮件和一条短信。这个时候只要用户注册数据保存在数据库之后，我们可以立即告诉用户注册成功并且将用户注册信息写入消息队列中，之后再慢慢根据消息队列中用户的注册信息去调用发送邮件和短信的接口。而不是调用发送邮件和短信的接口之后才返回用户注册成功的消息。增强系统响应速度，提升用户体验度。

- 流量削峰

比如我们现在做一个秒杀系统，一共只有1000库存。我们可以在消息队列中存放1000条数据，前1000个请求进来的时候直接秒杀成功，并且将用户信息记录下来。之后再来的请求则直接返回秒杀失败。

- 应用之间解耦

如订单系统需要调用库存系统，这个时候我们可以在订单系统和库存系统之间增加一个消息队列，采用发布订阅模式，当订单系统中有一个下单成功之后，向消息队列中发送一条消息，库存系统自动订阅这个目的地，订单系统发送一条消息的时候，库存系统则自动减少一条库存。

2.目的地类型

- 队列（点对点消息通信）

  消息发送者发送消息，消息代理将其放入一个队列中，消息接收者从队列中获取消息内容， 消息读取后被移出队列 – 消息只有唯一的发送者和接受者，但并不是说只能有一个接收者

- 主题 topic 发布/订阅消息通信

  发送者（发布者）发送消息到主题，多个接收者（订阅者）监听（订阅）这个主题，那么 就会在消息到达时同时收到消息

3.消息规范

- JMS (Java Message Service) JAVA消息服务：

  基于JVM消息代理的规范。 ActiveMQ、 HornetMQ是JMS实现

- AMQP

  高级消息队列协议，也是一个消息代理的规范，兼容JMS，RabbitMQ是AMQP的实现

4.RabbitMQ核心概念

- Message：消息，消息是不具名的，它由消息头和消息体组成。
- Publisher：消息生产者
- 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 。Exchange有4种类型：direct(默认)， fanout, topic, 和headers 。其中headers不常用。
- 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。Exchange 和Queue的绑定可以是多对多的关系。
- Channel 信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内的虚 拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这 些动作都是通过信道完成。因为

对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所 以引入了信道的概念，以复用一条 TCP 连接。

- 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，必须在连接时指定， RabbitMQ 默认的 vhost 是 / 。

5.RabbitMQ运行机制

AMQP 中消息的路由过程和 Java 开发者熟悉的 JMS 存在一些差别， AMQP 中增加了 Exchange 和 Binding 的角色。生产者把消息发布到 Exchange 上，消息最终到达队列并被 消费者接收，而 Binding 决定交换器的消息应该发送到那个队列。