

## RabbitMQ

MQ消息队列

消息队列

概念

- 消息队列

优势

- 异步
- 离线
- 去中心化

组件

消息生产者和消费者

- 生产者
- 消费者
- 队列

组件

- 生产者
- 消费者
- 队列
- 交换机

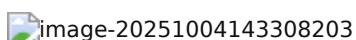
组件

- 生产者
- 消费者

组件

RabbitMQ组件

- virtual-host 虚拟主机
- publisher 发布者
- consumer 消费者
- queue 队列
- exchange 交换机



Java API

Spring AMQP API\*\* AMQP \*\*API

•

- 

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

- 

```
spring:
  rabbitmq:
    host: 192.168.150.101 # IP
    port: 5672 # 端口
    virtual-host: /hmall # 虚拟主机
    username: hmall # 用户名
    password: 123 # 密码
```

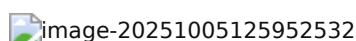
- SpringAMQP 通过 RabbitTemplate 操作消息队列
  - 使用 RabbitTemplate 的 convertAndSend 方法发送消息
- SpringAMQP 通过 @RabbitListener 注解处理消息队列
  - 定义 @Component 或者 Bean
  - 定义 @RabbitListener 处理方法

## Work Queues

通过 RabbitTemplate 操作消息队列

- 通过 RabbitTemplate 操作队列
- 通过 RabbitTemplate 消息队列的 prefetch 属性设置队列消费策略
  - prefetch 值为 1 表示每次只消费一个消息

```
spring:
  rabbitmq:
    listener:
      simple:
        prefetch: 1 # 每次只消费一个消息
```



•

通过 RabbitTemplate 操作消息队列

•

- Fanout ䷶
  - Direct ䷵
  - Topic ䷷

## Fanout

Fanout Exchange `fanoutExchange`\*`*` queue `**queue`

5 of 5

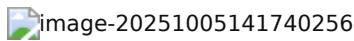
- `convertAndSend` `convertAndSend`

```
rabbitTemplate.convertAndSend(exchangeName, null, message);
```

**Direct** 

## Direct Exchange ┌─────────┐ Queue ┌─────────┐

- Queue → Exchange \*\* BindingKey \*\*
  - Exchange → Destination \*\* RoutingKey \*\*
  - Exchange → Queue BindingKey → RoutingKey → Destination



5

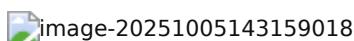
- `convertAndSend` `convertAndSend` `RoutingKey`

```
rabbitTemplate.convertAndSend(exchangeName, "red", message);
```

## Topic

Queue □ Exchange □ BingdingKey □□□□□□□□

- # 00000000
  - \* 11111111



1

-  `convertAndSend`  `RoutingKey`

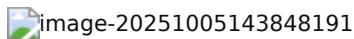
```
rabbitTemplate.convertAndSend(exchangeName, "usa.news", message);
```

10

Bean

SpringAMQP 

- Queue 声明语句 QueueBuilder 构建
- Exchange 声明语句 ExchangeBuilder 构建



- Binding 声明语句 BindingBuilder 构建

声明语句的实现类

- Binding bindingQueue1(Queue fanoutQueue1, FanoutExchange fanoutExchange) 实现 Bean<-->Spring

```

@Configuration
public class FanoutConfig {
    /**
     * 声明
     * @return Fanout
     */
    @Bean
    public FanoutExchange fanoutExchange(){
        return new FanoutExchange("hmall.fanout");
    }

    /**
     * ①
     */
    @Bean
    public Queue fanoutQueue1(){
        return new Queue("fanout.queue1");
    }

    /**
     * 声明
     */
    @Bean
    public Binding bindingQueue1(Queue fanoutQueue1, FanoutExchange fanoutExchange){
        return BindingBuilder.bind(fanoutQueue1).to(fanoutExchange);
    }

    /**
     * ②
     */
    @Bean
    public Queue fanoutQueue2(){
        return new Queue("fanout.queue2");
    }

    /**
     * 声明
     */
    @Bean
    public Binding bindingQueue2(Queue fanoutQueue2, FanoutExchange fanoutExchange){

```

```
        return BindingBuilder.bind(fanoutQueue2).to(fanoutExchange);
    }
}
```

MQ消息队列

SpringAMQP 消息队列 @RabbitListener 消息消费

- `bindings` 消息绑定
  - `values` 消息值
  - `exchange` 消息交换机
    - `name` 消息交换机名称 `type` 消息类型
  - `key` `bindingKeys` 消息键 `List`

```
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(name = "direct.queue1"),
    exchange = @Exchange(name = "hmall.direct", type = ExchangeTypes.DIRECT),
    key = {"red", "blue"})
)
public void listenDirectQueue1(String msg){
    System.out.println("直接模式direct.queue1" + msg + "!");
}
```

MQ消息

Java API for JNDI 消息队列

Java API for JSON 消息队列 JNDI 消息

- `jackson` 框架

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.10</version>
</dependency>
```

- `MessageConverter` 消息转换器

```
@Bean
public MessageConverter messageConverter(){
    // 1. 框架
    Jackson2JsonMessageConverter jackson2JsonMessageConverter = new
Jackson2JsonMessageConverter();
    // 2. 消息ID
    jackson2JsonMessageConverter.setCreateMessageIds(true);
    return jackson2JsonMessageConverter;
}
```

MQ消息队列

10

MQ

- 

```
spring:
  rabbitmq:
    connection-timeout: 1s # 指定MQ连接超时时间
    template:
      retry:
        enabled: true # 启用重试功能
        initial-interval: 1000ms # 初始重试间隔
        multiplier: 1 # 重试间隔倍数 = initial-interval * multiplier
        max-attempts: 3 # 最大重试次数
```

5 of 5

- ConfirmCallback မှု ဗိုလ်ချုပ် **Exchange**
    - ဗိုလ်ချုပ်ACK မှု ဗိုလ်ချုပ်NACK
  - ReturnsCallback မှု ဗိုလ်ချုပ် **Exchange** မှု ဗိုလ်ချုပ် **Queue**
    - ဗိုလ်ချုပ် မှု ဗိုလ်ချုပ် **Exchange** မှု ဗိုလ်ချုပ် မှု ဗိုလ်ချုပ်

## SpringAMQP 亂子\*\* Publisher Confirm □ Publisher Return \*\*乱子MQ乱子MQ乱子

- MQ PublisherReturn ACK
  - Exchange PublisherConfirm ACK
  - Exchange PublisherConfirm ACK
  - NACK

1

- **yaml** ဗိုလ်ချုပ်
    - **publisher-confirm-type** ဗိုလ်ချုပ်
      - **none** မူမှု confirm မှု
      - **simple** ဗိုလ်ချုပ်MQ ဗိုလ်ချုပ်
      - **correlated** MQ ဗိုလ်ချုပ်

```
spring:  
  rabbitmq:  
    publisher-confirm-type: correlated # ☐publisher confirm☐☐☐☐☐confirm☐  
    publisher-returns: true # ☐publisher return☐
```

- `RabbitTemplate` `ReturnCallback`

```
@Slf4j  
@AllArgsConstructor  
@Configuration  
public class MqConfig {  
    private final RabbitTemplate rabbitTemplate;
```

```

@PostConstruct
public void init(){
    rabbitTemplate.setReturnsCallback(new RabbitTemplate.ReturnsCallback() {
        @Override
        public void returnedMessage(ReturnedMessage returned) {
            log.error("return callback,");
            log.debug("exchange: {}", returned.getExchange());
            log.debug("routingKey: {}", returned.getRoutingKey());
            log.debug("message: {}", returned.getMessage());
            log.debug("replyCode: {}", returned.getReplyCode());
            log.debug("replyText: {}", returned.getReplyText());
        }
    });
}

```

- **ConfirmCallback**
  - Future CorrelationData
  - CorrelationData

```

@Test
void testPublisherConfirm() {
    // 1. CorrelationData
    CorrelationData cd = new CorrelationData();
    // 2. Future ConfirmCallback
    cd.getFuture().addCallback(new ListenableFutureCallback<CorrelationData.Confirm>()
    () {
        @Override
        public void onFailure(Throwable ex) {
            // 2.1. Future result
            log.error("send message fail", ex);
        }
        @Override
        public void onSuccess(CorrelationData.Confirm result) {
            // 2.2. Future result
            if(result.isAck()) { // result.isAck() boolean true ack false
                nack();
                log.debug("ack!");
            } else { // result.getReason() String nack reason
                log.error("nack, reason : {}", result.getReason());
            }
        }
    });
    // 3. convertAndSend
    rabbitTemplate.convertAndSend("hmall.direct", "q", "hello", cd);
}

```

参考

RabbitMQ 官方文档 3.6.0

- 
  - 
  - 
    - 

- MQ
  - IBM

## Lazy Queue

5

- 2048x2048
  - 2048x2048

### 3.12 Lazy Queue

□□□□□□□□□□ x-queue-mode □□□lazy□□

- Bean

```
@Bean  
public Queue queue(){  
    return QueueBuilder  
        .durable("Lazy-queue")  
        .lazy()  
        .build();  
}
```

- □□□□□□□

```
@RabbitListener(queuesToDeclare = @Queue(  
    name = "lazy.queue",  
    durable = "true",  
    arguments = @Argument(name = "x-queue-mode", value = "lazy"))  
public void listenLazyQueue(String msg){  
    log.info("【{}】 lazy.queue【{}】{}", msg);  
}
```



10

1

RabbitMQ RabbitMQ

- #### • ack 消息确认 RabbitMQ 消息确认

- **nack** RabbitMQ ကြောင်းအပါး
- **reject** RabbitMQ ကြောင်းအပါး
  - မရရှိနေရန်

SpringAMQP ကြောင်းအပါး**ACK** များ

- **none** မရရှိနေရန်
- **\*\* manual \*\*** API အပါး**ack** သို့**reject**
- **auto** SpringAMQP ကြောင်းအပါး**AOP** အပါး**ack** သို့**reject**
  - မရရှိနေရန်
  - မရရှိနေရန်

```
spring:
  rabbitmq:
    listener:
      simple:
        acknowledge-mode: none # မရရှိ
```

မြတ်နည်း

SpringAMQP ကြောင်းအပါး**MessageRecoverer** များ

yaml မြတ်နည်း

```
spring:
  rabbitmq:
    listener:
      simple:
        retry:
          enabled: true # မရရှိနေရန်
          initial-interval: 1000ms # စူးစွမ်းလမ်း
          multiplier: 1 # စူးစွမ်းလမ်း = multiplier * last-interval
          max-attempts: 3 # မြတ်နည်း
          stateless: true # trueမှား falseမှား
```

SpringAMQP ကြောင်းအပါး**MessageRecoverer** များ

- **RejectAndDontRequeueRecoverer** မရရှိနေရန်
- **ImmediateRequeueMessageRecoverer** မရရှိနေရန်
- **RepublishMessageRecoverer** မရရှိနေရန်

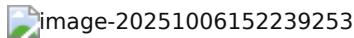
Java မြတ်နည်း

- **@Bean** မြတ်နည်း
- **RepublishMessageRecoverer** မြတ်နည်း

```
@Bean
public MessageRecoverer errorMessageRecover(RabbitTemplate rabbitTemplate) {
    return new RepublishMessageRecoverer(rabbitTemplate,"error.direct","error");
}
```

消息ID

消息的唯一性  $f(x)=f(f(x))$  消息的可追踪性



### 消息id

消息的唯一性id消息id消息的唯一性

- 消息的唯一性id消息的唯一性
- 消息的唯一性id消息的唯一性ID消息的唯一性
- 消息的唯一性id消息的唯一性

消息的唯一性

```
@Bean
public MessageConverter messageConverter(){
    // 1.消息转换器
    Jackson2JsonMessageConverter jjmc = new Jackson2JsonMessageConverter();
    // 2.消息的唯一性id消息的唯一性ID消息的唯一性
    jjmc.setCreateMessageIds(true);
    return jjmc;
}
```

消息对象 Message 消息对象

- 方法 `getBody()` 消息对象
- 方法 `getMessageProperties().getMessageId()` 消息ID

消息对象

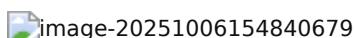
消息的唯一性id消息的唯一性ID消息的唯一性

消息对象

消息的唯一性id消息的唯一性ID消息的唯一性

- 方法 `basic.reject()` `basic.nack()` 消息对象 requeue `false`
- 方法 `basic.reject()`
- 方法 `basic.reject()`

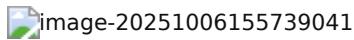
消息的唯一性id消息的唯一性ID消息的唯一性\*\* dead-letter-exchange 消息的唯一性id消息的唯一性ID消息的唯一性\*\*



- 方法 `basic.reject()`
- 方法 `basic.reject()`
- 方法 `basic.reject()`

消息对象

消息的唯一性id消息的唯一性ID消息的唯一性



RabbitMQ

@Exchange delayed true

```
@RabbitListener(bindings = @QueueBinding(
    value = @Queue(name = "delay.queue", durable = "true"),
    exchange = @Exchange(name = "delay.direct", delayed = "true"),
    key = "delay"
))
public void listenDelayMessage(String msg){
    log.info("delay.queue{}", msg);
}
```

Bean .delayed() .

```
@Bean
public DirectExchange delayExchange(){
    return ExchangeBuilder
        .directExchange("delay.direct") // 延迟交换机
        .delayed() // 延迟为true
        .durable(true) // 持久化
        .build();
}
```

x-delay 用途

- getMessageProperties() .setDelay()

```
// 2.发送消息
rabbitTemplate.convertAndSend("delay.direct", "delay", message, new
MessagePostProcessor() {
    @Override
    public Message postProcessMessage(Message message) throws AmqpException {
        // 延迟
        message.getMessageProperties().setDelay(5000);
        return message;
    }
});
```