消息中间件之RabbitMQ

一、概述

1.1 核心概念

1.1.1 JMS

JMS:Java Message Service,java消息服务,是一个消息服务的标准或者说是规范，允许应用程序组件基于JavaEE平台创建、发送、接收和读取消息。它使分布式通信耦合度更低，消息服务更加可靠以及异步性。

JMS是java的消息服务，JMS的客户端之间可以通过JMS服务进行异步的消息传输。

1.1.2 P2P

p2p:点对点发送，一个消息只能被消费一次

涉及：

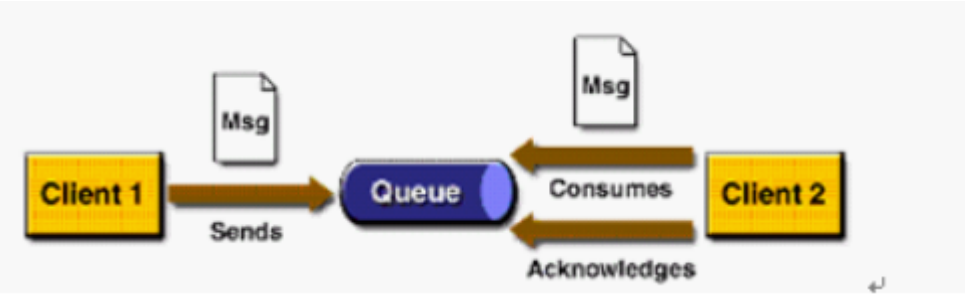
消息队列（Queue）

发送者(Sender)

接收者(Receiver)

每个消息都被发送到一个特定的队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时

示意图：



特点：

1、不用同时在线

2、一个消息只能被消费1次

1.1.3 Pub/Sub

Pub/Sub:发布订阅 一个消息可以被消费多次

涉及角色：

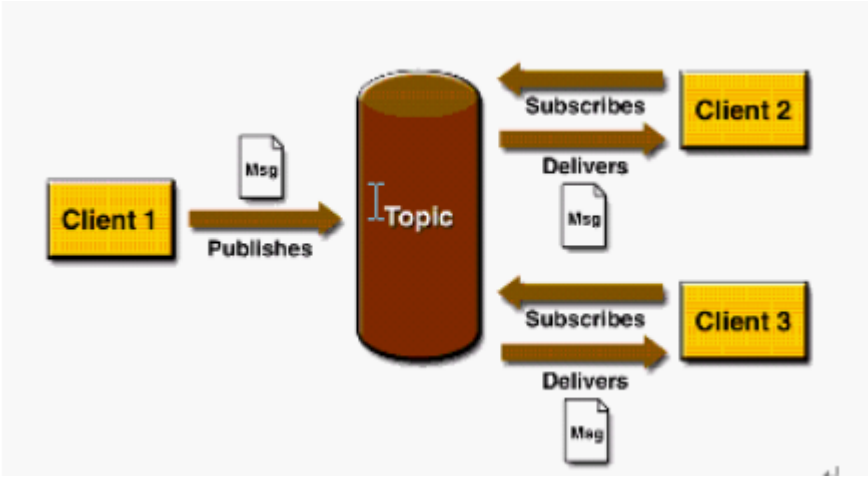
主题（Topic）

发布者（Publisher）

订阅者（Subscriber）

客户端将消息发送到主题。多个发布者将消息发送到Topic,系统将这些消息传递给多个订阅者

示意图：



特点：

1、发布者和订阅者同时在线

2、一个消息可以被多个订阅者消费

1.1.4 MQ

MQ：消息中间件（MOM：Message Orient middleware），消息队列

作为系统间通信的必备技术，低耦合、可靠传输、流量控制、最终一致性

实现异步消息通信

1.2 MQ的优缺

1、解耦

降低系统模块的耦合度

2、提高系统响应时间

3、异步消息

4、过载保护

基于MQ实现削峰填谷

1.3 主流MQ

1、ActiveMQ

Apache 下

完全支持Java的JMS协议

消息模式：1、点对点 2、发布订阅

2、RabbitMQ

Erlang语言 实现的开源的MQ中间件，支持多种协议

主要的通信协议是AMQP，即Advanced Message Queuing Protocol，高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。

3、Kafka

Apache下开源项目

高性能分布式消息队列，一般海量数据传输 大数据部门比用

单机吞吐量：10W/S

4、RocketMQ

阿里 贡献给了Apache

参考了Kafka实现的基于Java 消息中间件

5、ZeroMQ

消息传输最快

1.4 对比

1.从社区活跃度

按照目前网络上的资料，RabbitMQ 、activeM 、ZeroMQ 三者中，综合来看，RabbitMQ 是首选。

2.持久化消息比较

ZeroMq 不支持，ActiveMq 和RabbitMq 都支持。持久化消息主要是指我们机器在不可抗力因素等情况下挂掉了，消息不会丢失的机制。

3.综合技术实现

可靠性、灵活的路由、集群、事务、高可用的队列、消息排序、问题追踪、可视化管理工具、插件系统等等。

RabbitMq / Kafka 最好，ActiveMq 次之，ZeroMq 最差。当然ZeroMq 也可以做到，不过自己必须手动写代码实现，代码量不小。尤其是可靠性中的：持久性、投递确认、发布者证实和高可用性。

4.高并发

毋庸置疑，RabbitMQ 最高，原因是它的实现语言是天生具备高并发高可用的erlang 语言。

5.比较关注的比较， RabbitMQ 和 Kafka

RabbitMq 比Kafka 成熟，在可用性上，稳定性上，可靠性上， RabbitMq 胜于 Kafka （理论上）。

另外，Kafka 的定位主要在日志等方面， 因为Kafka 设计的初衷就是处理日志的，可以看做是一个日志（消息）系统一个重要组件，针对性很强，所以 如果业务方面还是建议选择 RabbitMq 。

还有就是，Kafka 的性能（吞吐量、TPS ）比RabbitMq 要高出来很多。

A	B	C	D	E
特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
PRODUCER-COMSUMER	支持	支持	支持	支持
PUBLISH-SUBSCRIBE	支持	支持	支持	支持
REQUEST-REPLY	支持	支持		
API完备性	高	高	高	高
多语言支持	支持，JAVA优先	语言无关	只支持JAVA	支持，java优先
单机吞吐量	万级	万级	万级	十万级
消息延迟		微秒级	毫秒级	毫秒级
可用性	高（主从）	高（主从）	非常高（分布式）	非常高（分布式）
消息丢失	低	低	理论上不会丢失	理论上不会丢失
消息重复		可控制		理论上会有重复
文档的完备性	高	高	高	高
提供快速入门	有	有	有	有
首次部署难度		低		中
社区活跃度	高	高	中	高
商业支持	无	无	阿里云	无
成熟度	成熟	成熟	比较成熟	成熟日志领域
特点	功能齐全，被大 量开源项目使用	由于Erlang 语言的并发能 力，性能很好	各个环节分 布式扩展设计，主从 HA；支持上万个队列；多种消费模式；性能很好	
支持协议	OpenWire、 STOMP、 REST、XMPP、AMQP	AMQP	自己定义的一 套(社区提供 JMS--不成熟)	
持久化	内存、文件、数据库	内存、文件	磁盘文件	
事务	支持	支持	支持	
负载均衡	支持	支持	支持	
管理界面	一般	好	有web console实现	
部署方式	独立、嵌入	独立	独立	
评价	优点：成熟的产品，已经在很多公司得到应用（非大规模场景）。有较多的文档。各种协议支持较好，有多重语言的成熟的客户端； 缺点：根据其他 用户反馈，会出莫名其妙的问题，切会丢 失消息。其重心放到 activemq6.0 产品—apollo 上去了，目前社区不活跃，且对5.x 维护较少； Activemq 不适合用于上千个队列的应用场景。	优点：由于erlang语言的特性，mq性能较好；管理界面较丰富，在互联网公司也有较大规模的应用；支持amqp系诶，有多中语言且支持amqp的客户端可用； 缺点：erlang语言难度较大。集群不支持动态扩展。	优点：模型简单，接口易用（JMS的接口很多场合并不太实用）。在阿里大规模应用。目前支付宝中的余额宝等新兴产品均使用 rocketmq 。集群规模大概在50台左右，单日处理消息上百亿；性能非常好，可以大量堆积消息在broker中；支持多种消费，包括集群消费、广播消费等。开发度较活跃，版本更新很快。 缺点：产品较新文档比较缺乏。没有在mq核心中去实现JMS等接口，对已有系统而言不能兼容。阿里内部还有一套未开源的MQAPI，这一层API可以将上层应用和下层MQ的实现解耦（阿里内部有多个mq的实现，如notify、metaq1.x，metaq2.x，rocketmq等），使得下面mq可以很方便的进行切换和升级而对应用无任何影响，目前这一套东西未开源。	

二、RabbitMQ

2.1简介

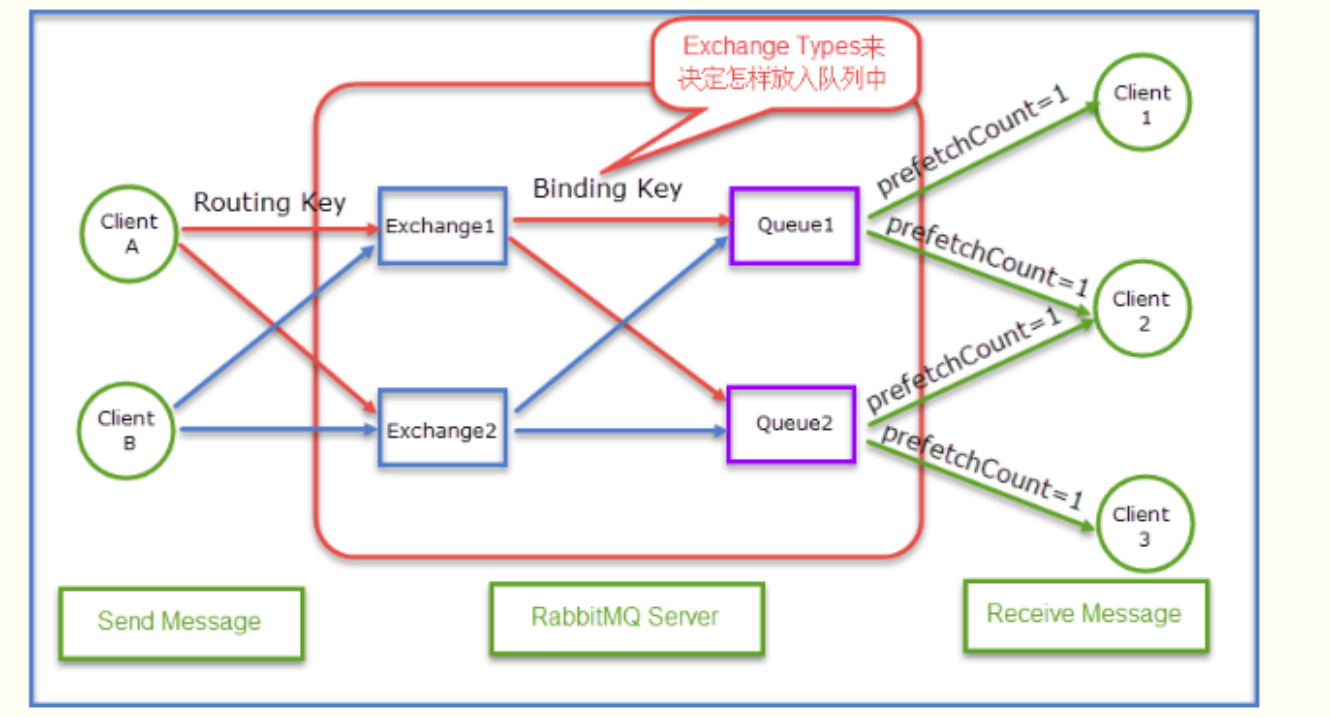
RabbitMQ是一个由erlang语言开发的AMQP（Advanced Message Queue ）的开源实现。AMQP 的出现其实也是应了广大人民群众的需求，虽然在同步消息通讯的世界里有很多公开标准（如 COBAR的 IIOP ，或者是 SOAP 等），但是在异步消息处理中却不是这样，只有大企业有一些商业实现（如微软的 MSMQ ， IBM 的 Websphere MQ 等），因此，在 2006 年的 6 月，Cisco 、Redhat、iMatix 等联合制定了 AMQP 的公开标准。

RabbitMQ是由RabbitMQ Technologies Ltd开发并且提供商业支持的。该公司在2010年4月被SpringSource（VMWare的一个部门）收购。在2013年5月被并入Pivotal。其实VMWare，Pivotal和EMC本质上是一家的。不同的是VMWare是独立上市子公司，而Pivotal是整合了EMC的某些资源，现在并没有上市。

RabbitMQ的官网是http://www.rabbitmq.com

消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然。 AMQP的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。 RabbitMQ是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP等，支持AJAX。用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。

2.2 通信模块



2.3 核心类说明

ConnectionFactory：为Connection的制造工厂，可以设置服务器和端口号和账号密码等信息

Connection：是RabbitMQ的socket链接，它封装了socket协议相关部分逻辑

Channel：与RabbitMQ打交道的最重要的一个接口，我们大部分的业务操作是在Channel这个接口中完成的，包括定义Queue、定义Exchange、绑定Queue与Exchange、发布消息等

Queue：（队列）RabbitMQ的作用是存储消息，队列的特性是先进先出。上图可以清晰地看到Client A和Client B是生产者，生产者生产消息最终被送到RabbitMQ的内部对象Queue中去，而消费者则是从Queue队列中取出数据

Exchange(交换机、交换器)：根据绑定的匹配规则，对消息进行匹配处理

三、RabbitMQ初体验

涉及角色：

3.1、MQ服务器

可以基于Docker安装RabbitMQ

记住：

15672：网页版 可视化服务器数据

5672：客户端连接点的端口号

提供：

http://39.105.189.141:15672/#/

账号：guest

密码：guest

具体的安装步骤：

http://note.youdao.com/noteshare?id=a2c279156e4b6d0dddb84322a8bdbf97&sub=303EB5DF56984C2A9118F3638F44B19C

切记：开放端口，否则拦截

3.2、MQ消息发送者

1、依赖jar

2、代码编写

1、创建连接工厂

设置连接线信息

2、获取连接对象

3、创建通道

4、定义队列

5、发送消息 到指定队列

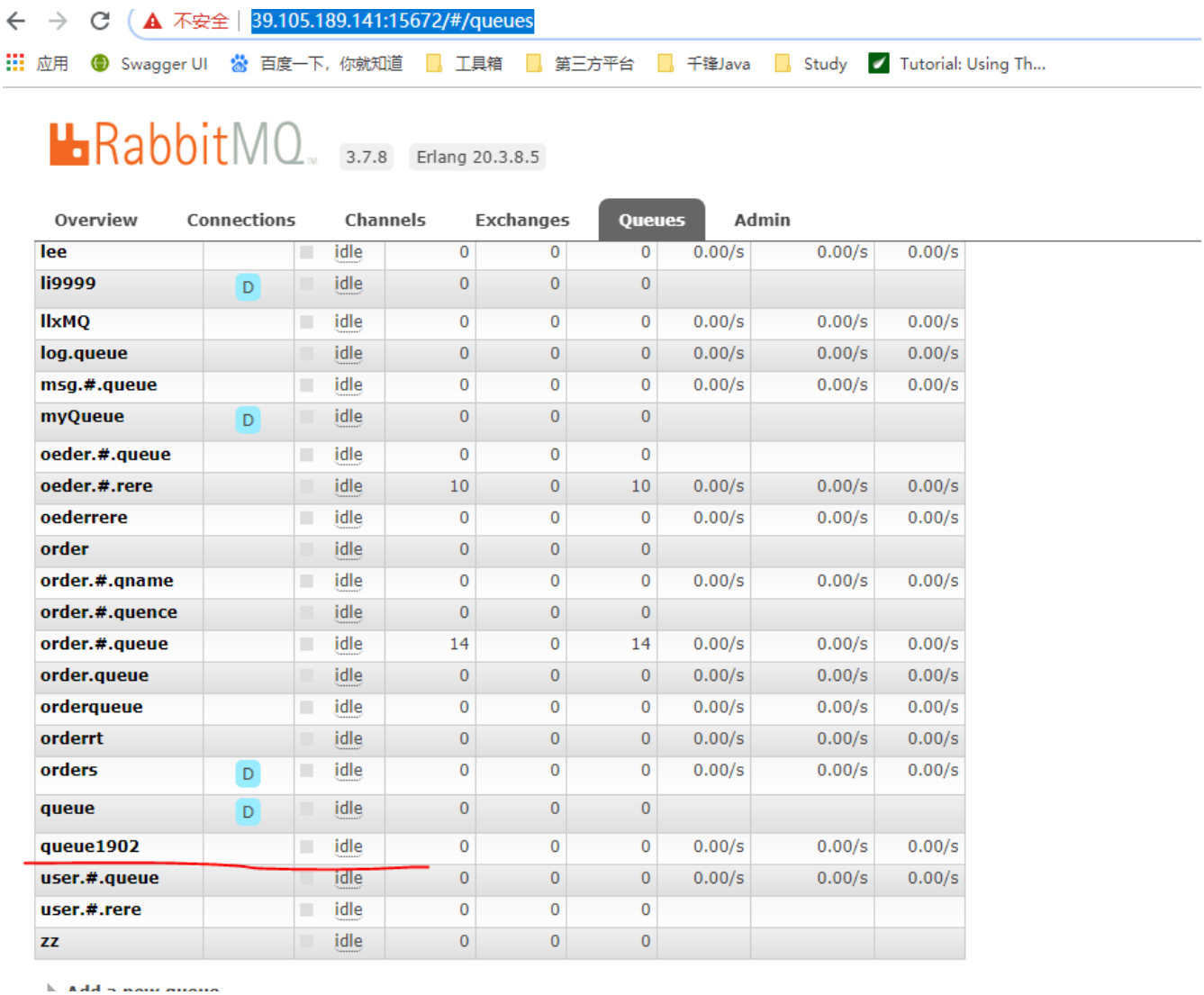
6、关闭连接

3、运行测试

打开浏览器，访问：

http://39.105.189.141:15672/#/queues

查看



消息发送者：

``

```
//http://39.105.189.141:15672
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
    Channel channel=connection.createChannel();
    //4、创建队列
    /**
     * 定义队列 参数说明
     * 1、队列名称
     * 2、是否持久化 队列消息是否存储到磁盘
```

```
         * 3、是否独占队列
         * 4、是否断开之后自动删除消息
         * 5、额外设置的数据信息 */
        channel.queueDeclare("queue1902",false,false,false,null);
        //5、发送消息
        /*参数说明:
         * 1、交换机名称
         * 2、队列名称
         * 3、属性参数
         * 4、发送的消息内容 要求字节*/
        channel.basicPublish("","queue1902",null,"你睡着了吗".getBytes());
        //6、关闭
        channel.close();
        connection.close();

    }
```

3.3、MQ消息接收者

1、依赖jar

2、代码编写

1、创建连接工厂

设置连接线信息

2、获取连接对象

3、创建通道

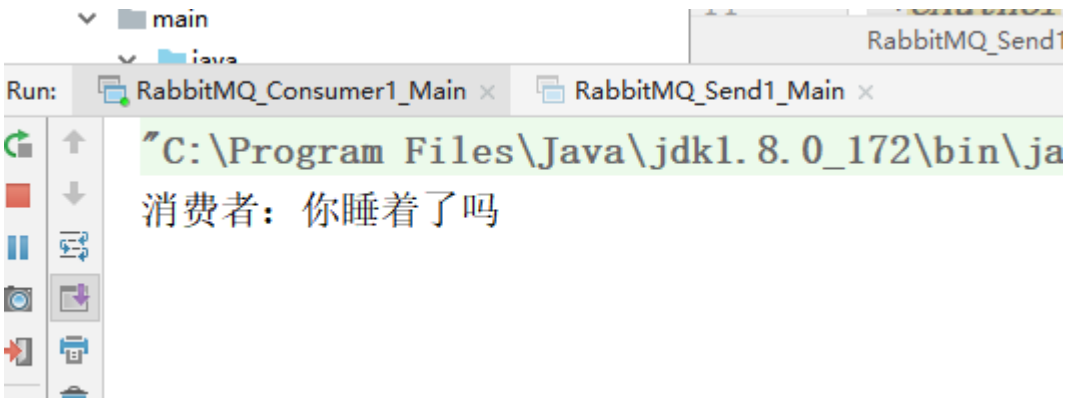
4、定义队列

5、创建消费者对象

6、设置消费者监听

3、运行测试

查看控制台能否获取消息



接收者代码:

```
```

```
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
    Channel channel=connection.createChannel();
    //4、创建队列
    /**
     * 定义队列 参数说明
     * 1、队列名称
     * 2、是否持久化 队列消息是否存储到磁盘
     * 3、是否独占队列
     * 4、是否断开之后自动删除消息
     * 5、额外设置的数据信息 */
    channel.queueDeclare("queue1902",false,false,false,null);
    //5、定义消费者
    Consumer consumer=new DefaultConsumer(channel){
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
            System.out.println("消费者: "+new String(body));
        }
    };
    //6、绑定消费者
    /**
     * 参数说明:
     * 1、队列名称
     * 2、是否自动应答
     * 3、消费者对象*/
    channel.basicConsume("queue1902",true,consumer);

}
```

四、RabbitMQ的消息模式

4.1 普通消息

点对点消息

一个消息只能消费一次

只需要队列就可以，不需要交换器

消息发送者和消息接收者可以不同时在线

4.2 交换器消息

RabbitMQ特色就在于Exchange，主要以下类型：

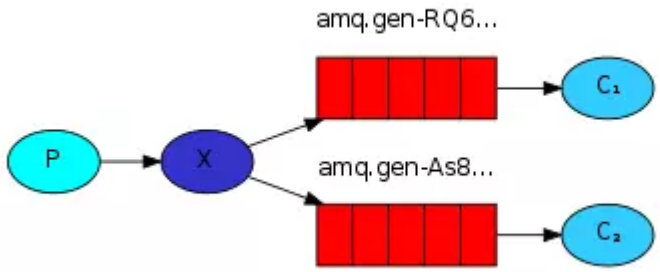
fanout： 只要有消息就转发给绑定的队列，不会进行消息的路由判断

direct: 会根据路由匹配规则，将消息发送到指定的队列中 注意路由规则不支持特殊字符

topic：会根据路由匹配规则，将消息发送到指定的队列中 注意路由规则支持特殊字符，比如：* #

4.2.1 Fanout消息

fanout类型的Exchange路由规则非常简单，它会把所有发送到该Exchange的消息路由到所有与它绑定的Queue中。



上图中，生产者（P）发送到Exchange（X）的所有消息都会路由到图中的两个Queue，并最终被两个消费者（C1与C2）消费。

代码示例：

消息提供者：

``

```
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
    Channel channel=connection.createChannel();
    //4、定义交换器
    channel.exchangeDeclare("exchange1902",BuiltinExchangeType.FANOUT);
    //5、创建队列
    channel.queueDeclare("laoxing",false,false,false,null);
    channel.queueDeclare("jingjie",false,false,false,null);
    //6、绑定队列
    /**
```

```
     * 参数说明:
     * 1、交换器名称
     * 2、路由规则
     * 3、要绑定的队列名称*/
    //channel.exchangeBind("exchange1902","",qn);
    /**
     * 队列绑定
     * 参数说明:
     * 1、队列名称
     * 2、交换器名称
     * 3、路由规则*/
    channel.queueBind("laoxing","exchange1902","");
    channel.queueBind("jingjie","exchange1902","");
    //7、发送消息
    channel.basicPublish("exchange1902","",null,"小雨：请假，约了心理医生".getBytes());
    channel.close();
    connection.close();
}
```

消息消费者：

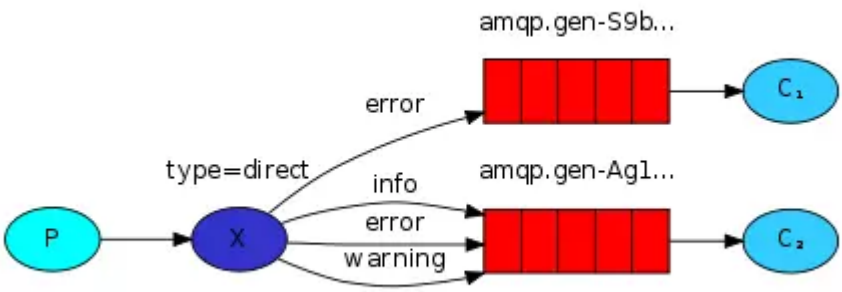
``

```
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
    Channel channel=connection.createChannel();
    //4、定义交换器
    channel.exchangeDeclare("exchange1902",BuiltinExchangeType.FANOUT);
    //5、创建队列
    channel.queueDeclare("laoxing",false,false,false,null);

    //6、绑定队列
    /**
     * 参数说明:
     * 1、交换器名称
     * 2、路由规则
     * 3、要绑定的队列名称*/
    //channel.exchangeBind("exchange1902","",qn);
    /**
     * 队列绑定
     * 参数说明:
     * 1、队列名称
     * 2、交换器名称
     * 3、路由规则*/
```

```
    channel.queueBind("jingjie","exchange1902","");
    //7、发送消息
    Consumer consumer=new DefaultConsumer(channel){
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
            System.out.println("消费者: "+new String(body));
        }
    };
    channel.basicConsume("jingjie",true,consumer);

}
```

4.2.2 Direct消息

direct类型的Exchange路由规则也很简单，它会把消息路由到那些binding key与routing key完全匹配的Queue中。



以上图的配置为例，我们以routingKey="error"发送消息到Exchange，则消息会路由到Queue1（amqp.gen-S9b...，这是由RabbitMQ自动生成的Queue名称）和Queue2（amqp.gen-Agl...）；如果我们以routingKey="info"或routingKey="warning"来发送消息，则消息只会路由到Queue2。如果我们以其他routingKey发送消息，则消息不会路由到这两个Queue中。

消息发送者代码示例：

``

```
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
```

```
    Channel channel=connection.createChannel();
    //4、定义交换器
    channel.exchangeDeclare("exc_1902_direct",BuiltinExchangeType.DIRECT);
    //5、创建队列
    channel.queueDeclare("order1902",false,false,false,null);
    channel.queueDeclare("user1902",false,false,false,null);
    //6、绑定队列到交换器
    channel.queueBind("order1902","exc_1902_direct","order");
    channel.queueBind("user1902","exc_1902_direct","user");
    //7、发布消息
    channel.basicPublish("exc_1902_direct","user",null,"新用户注册".getBytes());
    //销毁
    channel.close();
    connection.close();
}
```

消息消费者代码示例：

``

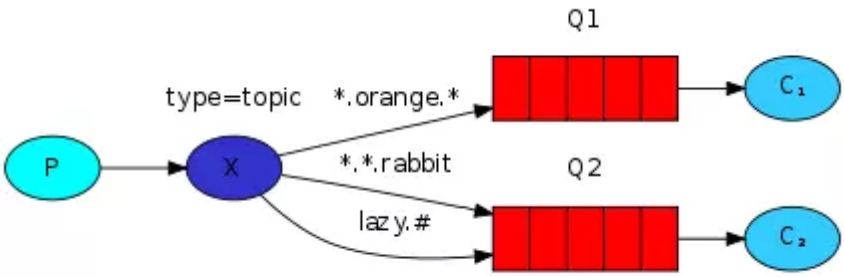
```
 public static void main(String[] args) throws IOException, TimeoutException {
        //1、创建连接工厂
        ConnectionFactory factory=new ConnectionFactory();
        //设置连接信息
        factory.setHost("39.105.189.141");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        //2、获取连接对象
        Connection connection=factory.newConnection();
        //3、获取通道对象
        Channel channel=connection.createChannel();
        //4、定义交换器
       // channel.exchangeDeclare("exchange1902",BuiltinExchangeType.FANOUT);
//         //5、创建队列
//         channel.queueDeclare("laoxing",false,false,false,null);

        //6、绑定队列
        /**
         * 参数说明:
         * 1、交换器名称
         * 2、路由规则
         * 3、要绑定的队列名称*/
        //channel.exchangeBind("exchange1902","",qn);
        /**
         * 队列绑定
         * 参数说明:
         * 1、队列名称
         * 2、交换器名称
         * 3、路由规则*/
        //channel.queueBind("order1902","exc_1902_direct","");
        //7、发送消息
        Consumer consumer=new DefaultConsumer(channel){
```

```
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者-支付: "+new String(body));
            }
        };
        channel.basicConsume("pay1902",true,consumer);

    }
```

4.2.3 Topic消息

direct类型的Exchange路由规则是完全匹配binding key与routing key，但这种严格的匹配方式在很多情况下不能满足实际业务需求。topic类型的Exchange在匹配规则上进行了扩展，它与direct类型的Exchage相似，也是将消息路由到binding key与routing key相匹配的Queue中，但这里的匹配规则有些不同，它约定：

- routing key为一个句点号“. ”分隔的字符串（我们将被句点号“. ”分隔开的每一段独立的字符串称为一个单词），如“stock.usd.nyse”、“nyse.vmw”、“quick.orange.rabbit”
- binding key与routing key一样也是句点号“. ”分隔的字符串
- binding key中可以存在两种特殊字符“*”*与“#”，用于做模糊匹配，其中“**”*用于匹配一个单词，“#”用于匹配多个单词（可以是零个）



以上图中的配置为例，routingKey=”quick.orange.rabbit”的消息会同时路由到Q1与Q2，routingKey=”lazy.orange.fox”的消息会路由到Q1与Q2，routingKey=”lazy.brown.fox”的消息会路由到Q2，routingKey=”lazy.pink.rabbit”的消息会路由到Q2（只会投递给Q2一次，虽然这个routingKey与Q2的两个bindingKey都匹配）；routingKey=”quick.brown.fox”、routingKey=”orange”、routingKey=”quick.orange.male.rabbit”的消息将会被丢弃，因为它们没有匹配任何bindingKey。

消息发送者代码示例：

``

```
public static void main(String[] args) throws IOException, TimeoutException {
    //1、创建连接工厂
    ConnectionFactory factory=new ConnectionFactory();
    //设置连接信息
    factory.setHost("39.105.189.141");
```

```
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    //2、获取连接对象
    Connection connection=factory.newConnection();
    //3、获取通道对象
    Channel channel=connection.createChannel();
    //4、定义交换器
    channel.exchangeDeclare("exc_1902_topic",BuiltinExchangeType.TOPIC);
    //5、创建队列
    channel.queueDeclare("pay1902",false,false,false,null);
    channel.queueDeclare("msg1902",false,false,false,null);
    channel.queueDeclare("oss1902",false,false,false,null);
    channel.queueDeclare("msg1901",false,false,false,null);
    //6、绑定队列到交换器
    channel.queueBind("pay1902","exc_1902_topic","pay.#");
    channel.queueBind("msg1902","exc_1902_topic","msg.#");
    channel.queueBind("oss1902","exc_1902_topic","oss.#");
    channel.queueBind("msg1901","exc_1902_topic","msg.#");
    //7、发布消息
    channel.basicPublish("exc_1902_topic","msg.",null,"订单222预支付信息".getBytes());
    //销毁
    channel.close();
    connection.close();
}
```

消息消费者代码示例：

``

```
  public static void main(String[] args) throws IOException, TimeoutException {
        //1、创建连接工厂
        ConnectionFactory factory=new ConnectionFactory();
        //设置连接信息
        factory.setHost("39.105.189.141");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        //2、获取连接对象
        Connection connection=factory.newConnection();
        //3、获取通道对象
        Channel channel=connection.createChannel();
        //4、定义交换器
       // channel.exchangeDeclare("exchange1902",BuiltinExchangeType.FANOUT);
//         //5、创建队列
//         channel.queueDeclare("laoxing",false,false,false,null);

        //6、绑定队列
        /**
         * 参数说明:
         * 1、交换器名称
         * 2、路由规则
         * 3、要绑定的队列名称*/
```

```
        //channel.exchangeBind("exchange1902","",qn);
        /**
         * 队列绑定
         * 参数说明:
         * 1、队列名称
         * 2、交换器名称
         * 3、路由规则*/
        channel.queueBind("pay1902","exc_1902_topic","");
        //7、发送消息
        Consumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者: "+new String(body));
            }
        };
        channel.basicConsume("pay1902",true,consumer);

    }
```

五、RabbitMQ总结

5.1核心

异步消息通信 消息服务器共享的

消息发送者:

1、直接发送给队列

2、将消息发给交换器

消息消费者:

1、监听队列的数据变化

2、自动接收消息

RabbitMQ知识点:

1、MQ服务器 Docker

2、发送消息 basicPublish

3、消费消息 basicConsumer

1、队列必须定义、交换器必须定义 交换器和队列绑定 queueBind

5.2 应用场景

1.其实我们在双11的时候，当我们凌晨大量的秒杀和抢购商品，然后去结算的时候，就会发现，界面会提醒我们，让我们稍等，以及一些友好的图片文字提醒。而不是像前几年的时代，动不动就页面卡死，报错等来呈现给用户。在这业务场景中，我们就可以采用队列的机制来处理，因为同时结算就只能达到这么多。

流量降级

2.在我们平时的超市中购物也是一样，当我们在结算的时候，并不会一窝蜂一样涌入收银台，而是排队结算。这也是队列机制，就是排队。一个接着一个的处理，不能插队。

并发串行 削峰填谷

3.业务逻辑解耦

```
登录的逻辑---JWT
1、验证是否被冻结
2、验证手机号是否存在
3、校验密码
4、生成JWT令牌
5、存储令牌
6、登录失败  校验登录失败次数
7、冻结账号
8、记录登录日志

每一步0.1秒
0.8秒
```

```
基于RabbitMQ
1、验证是否被冻结
2、验证手机号是否存在
3、校验密码
4、生成JWT令牌
5、存储令牌
6、发送消息
0.6
```

消息监听者： 1、验证是否成功 6、登录失败 校验登录失败次数 7、冻结账号 8、记录登录日志 0.4秒

示例源码：https://github.com/xingpenghui/Point_Study