1. MQ
2. MQ概述

MQ全称为Message Queue(消息队列)，是在消息的传输过程中保存消息的容器，多用于分布式系统之间进行通信。分布式系统通信有两种方式，一种是直接远程调用另一种是借助中间件来完成间接通信。通信的发送方为生产者，接收方为消费者。

1. 消息中间件

在项目中，可以将一些无需即时返回且耗时的操作提取出来，进行异步处理，而这种异步处理的方式大大地节省了服务器的请求响应时间，从而提高了系统的吞吐量。而消息中间件就是用于提供这种异步处理的。

1. MQ的优势
2. 任务异步处理

将不需要同步处理且耗时长的操作有消息队列通知消息接收方进行异步处理，提高了应用程序的响应时间。提升了用户的体验和系统的吞吐量。

1. 应用程序解耦合

MQ相当于一个中介，生产方通过MQ与消费方交互，它将应用程序进行解耦合，当某一个组件出现BUG时，不会影响到另一个组件，提高了系统的容错性，同时便于系统的扩展。

1. 削峰填谷

在数据库遇到高并发访问时会遭遇宕机的风险，而是用了MQ之后，请求的消息被MQ保存起来，然后系统可以按照自己的消费能力来消费，比如每秒1000个数据，这样慢慢写到数据库，不会有宕机风险，这样就把并发峰值给削掉了。同时，高峰期产生的数据依然在MQ中，在之后的一段时间内，消费消息的速度依然会维持在每秒1000个，知道消费完积压的消息，这叫做填谷。提高了系统的稳定性。

1. MQ的劣势
2. 系统可用性降低

由于使用MQ，不仅要保障通信的系统之间的稳定，同时要保障MQ的稳定，一旦MQ宕机，就会影响业务。

1. 系统复杂度提高

MQ的加入提高了系统的复杂度，异步调用与同步调用是大大不同的，如何保证消息不被重复消费，消息传递的顺序性都是需要考虑的问题。

1. 一致性问题

异步处理的数据一致性问题十分重要，只有数据保持一致，异步处理才有意义。

1. 使用MQ需要满足的条件
2. 生产者不需要从消费者获得反馈。因为异步处理不需要等待消费者的反馈，可以同步执行其他的操作，因此要使用MQ的条件是消费者的返回值为空。
3. MQ是通过MQ存储消息，之后让消费者处理的机制而实现的，因此一定会有处理时间的问题，使用MQ一定要容许有短暂的数据不一致性，不然就不能使用MQ。
4. 常见的MQ产品

业界上有很多的MQ产品，如RabbitMQ, RocketMQ, ActiveMQ, Kafka, ZeroMQ, MetaMQ等等，在技术选型时，要结合自身需求及MQ产品特征选择。

1. RabbitMQ

RabbitMQ是使用Erlang语言开发的，这是一款高并发的语言，因此，RabbitMQ的并发能力强，性能非常好。单级吞吐量为万级，消息延迟为微秒级。

1. ActiveMQ

ActiveMQ是由Apache基金会支持的，比较成熟，文档较多，单级吞吐量为万级，消息延迟为毫秒级。

1. RocketMQ

RocketMQ是阿里用Java研发的，功能比较完备，扩展性很好，单级吞吐量为十万级，消息延迟为毫秒级。

1. Kafka

Kafka是由Apache基金会支持的，开发语言为Scala&Java，多用于大数据领域，单级吞吐量为十万级，消息延迟为毫秒。

1. RabbitMQ
2. AMQP

AMQP，Advanced Message Queuing Protocol，是一个应用层协议的开放标准，为消息中间件设计的，类比与HTTP。基于此协议的客户端和消息中间件之间可以传递消息，并不受客户端/中间件不同产品，不同开发语言的限制。

AMQP的架构是生产者将消息发送到交换机中，交换机通过路由将消息存储到消息队列中，消费者再从消息队列中取出消息进行消费。

1. RabbitMQ简介

Rabbit技术公司在2007年基于AMQP标准开发了RabbitMQ，采用Erlang语言开发，这是一种专用于开发高并发和分布式系统的一种语言。

1. RabbitMQ架构

RabbitMQ的架构与AMQP的架构差不多，RabbitMQ中间件包括交换机和消息队列，只不过RabbitMQ中是由多个隔离的虚拟机组成的，每个虚拟机中有多个交换机和消息队列，每个交换机绑定不同的消息队列。

为了连接的高效与节能，生产者和消费者与RabbitMQ建立连接是通过连接中的多个管道进行的，每次连接只需要单条管道。

1. RabbitMQ的相关概念
2. Broker

接受和分发消息的应用，就是RabbitMQ Server消息中间件。

1. Virtual host

处于多用户和安全因素设计的，把AMQP的基本组件划分到一个虚拟的分组中，当多个不同的用户使用同一个RabbitMQ Server提供的服务时，可以划分出多个Virtual host,每个用户在自己的Virtual host中创建 echange 和 queue。

1. Connection

生产者，消费者和消息中间件之间的TCP连接。

1. Channel

为了节省资源，提高链接的效率，在connection内部建立逻辑链接channel，如果程序支持多线程，那么每个thread可以创建单独的channel进行通讯。AMQP method包含了channel id 帮助客户端和message broker 识别 channel ，所有channel之间是完全隔离的。

1. Exchange

message到达broker之后，根据分发规则，查询表中的routing key, 分发消息到queue中，常见的有direct, topic, fanout。

1. Queue

消息从Exchange 被发送到Queue中存储，等待被消费者取走。

1. Binding

交换机和队列之间的虚拟链接，binding中可以包含routing key。binding信息被保存到exchange的查询表中，用于消息的分发依据。

1. RabbitMQ的工作模式

简单模式，work queues，Publish/Subscribe发布与订阅模式，Routing路由模式，Topic主题模式，RPC远程调用模式。

1. JMS

JMS是Java消息服务（JavaMessage Service）应用程序接口，是一个Java平台中关于面向消息中间件的API，类比与JDBC，很多消息中间件都实现了JMS规范，如ActiveMQ，但是RabbitMQ官方并没有提供JMS的实现包，只有开源社区中有。

1. 安装rabbitmq与erlang

由于安装rabbitmq需要首先安装erlang，因此我们首先在ubuntu上安装erlang。注意，安装环境为ubuntu16.04,。

1. 安装erlang

sudo apt-get install erlang-nox

如果在安装过程中报错：ubuntu16.04 E: dpkg 被中断，请输入‘sudo dpkg –configure -a’ 解决此问题。  
如果这样依旧不行，删除掉然后重新即可。

sudo rm /var/lib/dpkg/updates/\*

sudo apt-get update

1. 安装rabbitmq

sudo apt-get upgrade

sudo qpt-get install rabbitmq-server

1. rabbitmq管理

# 启动rabbitmq服务

sudo service rabbitmq-server start

# 关闭rabbitmq服务

sudo service rabbitmq-server stop

# 重启服务

sudo service rabbitmq-server restart

# 查看服务运行状态

sudo service rabbitmqctl status

1. 安装web管理插件

sudo rabbitmqctl start\_app

sudo rabbitmq-plugins enable rabbitmq\_management

安装完插件后，可以通过在ubuntu中通过网址<http://127.0.0.1:15672>进行访问，初始的用户名和密码都是guest，如果登录失败，说明没有开启超级管理员权限。需要新建用户并授予超级管理员权限：

新建用户

sudo rabbitmqctl add\_user admin admin

* 1

赋予权限

sudo rabbitmqctl set\_user\_tags admin administrator

* 1

赋予virtual host中所有资源的配置、写、读权限以便管理其中的资源，也是添加远程访问权限

sudo rabbitmqctl set\_permissions -p / admin '.\*' '.\*' '.\*'

若遇到Config file配置文件找不到的情况，可以到rabbitmq的默认安装路径/usr/share/doc/rabbitmq-server中将rabbitmq.config复制到/etc/rabbitmq/rabbitmq.config中

cd /usr/share/doc/rabbitmq-server

cp rabbitmq.config /etc/rabbitmq/rabbitmq.config

这样重启rabbitmq-server服务后，配置文件配置成功了

service rabbitmq-server restart

1. rabbitmq命令行操作

sudo rabbitmqctl # 查看所有命令和帮助文档  
sudo rabbitmqctl stop # 停止服务  
sudo rabbitmqctl status # 查看服务状态  
sudo rabbitmqctl list\_users # 查看当前所有用户  
sudo rabbitmqctl list\_user\_permissions guest # 查看默认guest用户的权限  
sudo rabbitmqctl delete\_user guest # 删掉默认用户(由于RabbitMQ默认的账号用户名和密码都是guest。为了安全起见, 可以删掉默认用户）  
sudo rabbitmqctl add\_user username password # 添加新用户  
sudo rabbitmqctl set\_user\_tags username administrator# 设置用户tag  
sudo rabbitmqctl set\_permissions -p / username ".\*" ".\*" ".\*" # 赋予用户默认vhost的全部操作权限  
sudo rabbitmqctl list\_user\_permissions username # 查看用户的权限  
rabbitmqctl set\_user\_tags User Tag  
User为用户名， Tag为角色名(对应于上面的administrator，monitoring，policymaker，management，或其他自定义名称)。

1. rabbitmq的权限角色
2. 超级管理员（administrator）

可以登录管理控制台查看所有的信息，对用户，策略进行操作。

1. 监控着（monitoring）

可以登录管理控制台查看rabbitmq节点的相关信息（进程数，内存使用情况，磁盘使用情况）

1. 策略制定者（policymaker）

可以登录管理控制台，对策略进行管理，但无法查看节点的相关信息。

1. 普通管理者（management）

可以登录管理控制台，无法看淡节点信息，也无法对策略进行管理。

1. 其他

无法登录管理控制台，一般就是生产者和消费者。

1. Rabbitmq项目实践
2. 创建模块，配置pom.xml

Rabbitmq连接着生产者和消费者，因此创建两个maven模块，一个是生产者producer，一个是消费者consumer。在配置文件pom.xml中添加rabbitmq的客户端依赖。

<dependency>  
 <groupId>com.rabbitmq</groupId>  
 <artifactId>amqp-client</artifactId>  
 <version>5.6.0</version>  
</dependency>

1. 创建生产者类
2. 流程概述

在Rabbitmq的流程中，生产者通过连接将消息传输到 Rabbitmq服务器中，其中包括创建连接工厂，设置通信的参数如ip，端口等，之后创建连接，创建连接中的通道，在复杂的情况下还需要创建 交换机，最后创建队列，一切准备就绪后就可以发送消息了。

1. 创建连接工厂

ConnectionFactory factory = new ConnectionFactory();

1. 设置连接的参数

factory.setHost(“192.168.211.128”); //设置host

factory.setPort(5672); //设置端口，默认5672

factory.setVirtualHost(“/hello”); //设置虚拟机，默认值为/

factory.setUsername(“wxx”); //设置用户名，默认guest

factory.setPassword(“wxx”); // 设置密码，默认guest

1. 创建连接Connection

Connection connection = factory.newConnection();

1. 创建Channel

Channel channel = connection.createChannel();

1. 创建队列Queue

channel.queueDeclare(“hello\_world”, true, false, false, null);

queueDeclare函数的声明如下：

queueDeclare(String queue, Boolean durable, Boolean exclusive, Boolean autoDelte, Map<String,Object> argument);

该函数参数介绍如下：

queue: 队列名称

durable:是否持久化，即是否保存到硬盘中

exclusive:

是否独占，只能有一个消费者监听这个队列

当Connection关闭时，是否删除队列

autodelete: 当没有Consumer时，是否自动删除队列。

arguments: 参数

1. 发送消息

生产者发送消息使用的是通道的basicPublish函数，该函数的声明如下：

basicPublish(String exchange, String routingKey, BasicProperties pros, byte[] body)

参数的介绍如下：

exchange : 交换机名称，简单模式下交换机会使用默认的””，即不使用交换机。

routingKey: 路由名称，在简单模式下不使用交换机，则路由名称直接用队列的名称。

props: 配置信息

body: 发送的消息数据，注意是byte[]格式的。

String body = “hello world”;

channel.basicPublish(“”, “hello\_world”, null, body.getBytes());

1. 测试

执行生产者类代码后，我们发现在Rabbitmq的管理界面中多了一条队列，说明执行成功。

1. 创建消费者类
2. 概述

消费者类中的内容是消费者监听Rabbitmq服务器，从队列中通过连接的通道监听消息，并且消费消息的过程。因此，创建连接，创建通道的操作与生产者都是一样的，后面的从通道接受消息操作是不同的。

1. 接受消息

使用通道的basicConsume函数，该函数的声明如下：

basicConsume(String queue, Boolean autoAck, Consumer callback)

参数介绍：

queue: 队列名称

autoAck: 是否自动确认

callback: 回调对象

这个回调对象是一个Consumer接口，这个接口有个实现类DefaultConsumer，该实现类有唯一一个构造方法，通过传入通道进行构造。我们需要创建一个DefaultConsumer实现类，并且重写它的handleDelivery方法，这是一个回调方法，当收到信息后，会自动执行该方法。

handDelivery函数的声明如下：

handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body);

参数介绍：

consumerTag: 消息的标识

envelope: 获取一些信息，我们可以从中取出信息，如交换机，路由key

properties: 配置信息

body: 数据

Consumer consumer = new DefaultConsumer(channel){

@Override

public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body){

System.out.println(“consumerTag:” + consumerTag);

System.out.println(“Exchange”+ envelope.getExchange());

System.out.println(“RoutingKey” + envelope.getRoutingKey());

System.out.println(“properties” +properties);

System.out.println(“body” +new String(body));

}

};

channel.basicConsume(“hello\_world”, true, consumer);

注意，消费者是一个监听程序，不需要关闭资源！！！

1. 测试

执行消费者代码后，在Rabbitmq管理界面中看到之前生成的队列已经被取走了，说明消费者执行成功。

1. Rabbitmq的工作模式
2. Work queues工作队列模式

与简单模式不同，工作队列模式多了一个或者一些消费端，多个消费端共同消费同一个队列中的消息。这种模式常常应用于任务过多的情况，可以提高任务处理的速度。消费者之间对于同一个消息的关系是竞争的关系。

1. Pub/Sub 订阅模式
2. 流程介绍

订阅模式中，多了交换机的转发过程。生产者将消息发送给交换机，交换机决定如何处理这些消息，如交给所有的队列，或交给特定的队列，或是丢弃这些消息，这些操作取决于 交换机的类型。队列得到这些消息后，消费者监听队列并得到消息。

1. 交换机类型
2. Fanout

广播，将消息交给所有绑定到交换机的队列。

1. Direct

定向，将消息交给符合指定Routing key的队列。

1. Topic

通配符，将消息交给符合Routing pattern（路由模式）的队列。

注意，交换机没有存储能力，只负责转发消息，因此若没有队列和交换机绑定或没有符合路由规则的队列时，消息会丢失。

1. Fanout模式实例
2. 创建生产者类

生产者类的流程与之前差不多，多了创建交换机，绑定队列和交换机的流程。

1. 创建交换机

创建交换机是使用通道的exchangeDeclare方法进行创建，exchangeDeclare方法的声明如下：

exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String,Object> arguments);

参数介绍；

exchange: 交换机名称

type: 交换机类型

DIRECT(“direct”): 定向

FANOUT(“fanout”): 扇形（广播），发送消息到每个与之绑定的队列

TOPIC(“topic”): 通配符

HEADERS(“headers”):参数匹配

durable: 是否持久化

autoDelete: 是否自动删除

internal: 是否内部使用，一般为false

arguments: 参数

调用exchangeDeclare函数如下：

String exchangeName = “test\_fanout”;

channel.exchangeDeclare(exchangeName, BuiltinExchangeType.FANOUT, true, false, false, null);

1. 创建多个队列

String queue1Name = “test\_fanout\_queue1”;

String queue2Name = “test\_fanout\_queue2”;

channel.queueDeclare(queue1Name, true, false, false);

channel.queueDeclare(queue2Name, true, false, false);

1. 绑定队列和交换机

使用通道的queueBind函数进行绑定，queueBind函数的声明如下：

queueBind(String queue, String exchange, String routingkey)

参数介绍：

queue: 队列名称

exchange: 交换机名称

routingkey: 路由键，绑定规则，若交换机的类型为fanout,则routingkey设置为””，即全部发送。

queueBind函数的调用如下所示：

channel.queueBind(queue1Name, exchangeName, “”);

channel.queueBind(queue2Name, exchangeName, “”);

（二）测试生产者类

执行生产者类的代码后，我们可以在Rabbitmq的管理界面看到，队列多了两个，每个队列中有同等数量的消息，因为是广播发送的。

完整的生产类代码如下：

package com.rabbitmq.producer;  
  
  
import com.rabbitmq.client.BuiltinExchangeType;  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.ConnectionFactory;  
  
import java.io.IOException;  
import java.util.concurrent.TimeoutException;  
  
public class Producer\_PubSub {  
 public static void main(String[] args) throws TimeoutException, IOException {  
 //1. 创建连接工厂  
 ConnectionFactory factory = new ConnectionFactory();  
 //2. 设置参数  
 factory.setHost("192.168.211.128");  
 factory.setPort(5672);  
 factory.setVirtualHost("hello");  
 factory.setUsername("wxx");  
 factory.setPassword("wxx");  
 //3.创建连接  
 Connection connection = factory.newConnection();  
 //4.创建通道  
 Channel channel = connection.createChannel();  
 //5.创建交换机  
 String exchangeName = "test\_fanout";  
 channel.exchangeDeclare(exchangeName, BuiltinExchangeType.*FANOUT*, true, false, false, null);  
  
 //6.创建队列  
 String queue1Name = "test\_fanout\_queue1";  
 String queue2Name = "test\_fanout\_queue2";  
 channel.queueDeclare(queue1Name, true, false, false, null);  
 channel.queueDeclare(queue2Name, true, false, false, null);  
 //7.绑定交换机和队列  
 channel.queueBind(queue1Name, exchangeName, "");  
 channel.queueBind(queue2Name, exchangeName, "");  
 //6. 发送消息  
 for (int i = 1; i <=10 ; i++) {  
 String body = i + "Pub/Sub";  
 channel.basicPublish(exchangeName, "", null, body.getBytes());  
 }  
  
 //7. 释放资源  
 channel.close();  
 connection.close();  
 }  
}

1. Routing 路由模式
2. 流程概述

路由模式中交换机与队列进行绑定时，需要指定一个RoutingKey，同时消息在发送给交换机时也必须制定消息的RoutingKey。交换机在转发消息时，根据消息的RoutingKey进行判断，只有队列的RoutingKey与消息的RoutingKey一致，才会收到消息。

1. 创建交换机

路由模式需要修改交换机的类型，改为DIRECT类型。

String exchangeName = “test\_direct”;

channel.exchangeDeclare(exchangeName, BuiltinExchangeType.DIRECT, true, false, false, null);

1. 创建队列

String queue1Name = “test\_direct\_queue1”;

String queue2Name = “test\_direct\_queue2”;

channel.queueDeclare(queue1Name, true, false, false, null);

channel.queueDeclare(queue2Name, true, false, false, null);

1. 绑定交换机和队列

这里绑定交换机和队列时我们需要制定RoutingKey，本案例的RoutingKey是日志的级别，第一个队列绑定error级别，第二个队列绑定info, error, warning级别。

channel.queueBind(queue1Name, exchangeName, “error”);

channel.queueBind(queue2Name, exchangeName, “info”);

channel.queueBind(queue2Name, exchangeName , “error”);

channel.queueBind(queue2Name, exchangeName, “warning”);

1. 发送消息

路由模式下发送消息同样需要绑定RoutingKey。

String body = “log:… loglevel:info”;

channel.basicPublish(exchangeName, “info”, null. body.getBytes());

1. Topics通配符模式

通配符模式即使用通配符制定规则。如news.#，凡是以news.开头的RoutingKey都能被匹配到。#代表了零到多个单词，\*代表了一个单词。

1. Spring整合RabbitMQ
2. 概述

所谓的用Spring来整合Rabbitmq，就是Rabbitmq所需要的一些配置先在配置文件中设置好，在代码中需要使用时直接注入到Spring容器中，这样就可以大大地减少代码量。

1. Spring整合Rabbitmq生产者类
2. 导入依赖坐标

导入Spring的坐标，以及rabbitmq整合的相关坐标。

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.1.17.RELEASE</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.amqp</groupId>  
 <artifactId>spring-rabbit</artifactId>  
 <version>2.1.8.RELEASE</version>  
 </dependency>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>5.1.7.RELEASE</version>  
 </dependency>  
</dependencies>  
  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

1. rabbitmq的配置文件

将连接rabbitmq所需要参数全部都配置到一个properties文件中，如ip地址，端口号，用户名，密码，虚拟机名称。

rabbitmq.host=192.168.211.100  
rabbitmq.port=5672  
rabbitmq.username=  
rabbitmq.password=  
rabbitmq.virtual-host=/

1. Spring的核心配置文件

（1）概述

这个配置文件就将我们在使用rabbitmq时所用到的连接，交换机，队列等等都配置到了Spring容器中，这样，当我们需要使用这些组件时直接注入到容器中即可使用，非常方便。

1. 加载配置文件

我们之前抽去了rabbitmq的配置文件，而在Spring的配置文件中我们需要使用到rabbitmq的一些参数，因此我们需要加载rabbitmq.properties配置文件。

<context:property-placeholder location=”classpath:rabbitmq.properties”/>

1. 定义rabbitmq connectionFactory

连接由连接工厂产生，因此我们需要定义一个连接工厂，该连接工厂的参数都是由rabbitmq.properties配置文件的参数提供的，用于连接到rabbitmq服务器。

<rabbit:connection-factory id="connectionFactory" host="${rabbitmq.host}"  
 port="${rabbitmq.port}"  
 username="${rabbitmq.username}"  
 password="${rabbitmq.password}"  
 virtual-host="${rabbitmq.virtual-host}"/>

1. 定义管理交换机，队列的admin工厂

交换机，队列都需要进行管理，因此设置一个管理的工厂就是我们上面定义的连接工厂。

<rabbit:admin connection-factory="connectionFactory"/>

1. 定义持久化队列，不存在则自动创建

这些队列会自动创建，如果没有被绑定到响应的交换机中，它们会默认绑定到默认的交换机，默认的交换机类型为direct，名称为“”，路由键为队列的名称。

<rabbit:queue id="spring\_queue" name="spring\_queue" auto-declare="true"/>  
<rabbit:queue id="spring\_fanout\_queue\_1" name="spring\_fanout\_queue\_1" auto-declare="true"/>  
<rabbit:queue id="spring\_fanout\_queue\_2" name="spring\_fanout\_queue\_2" auto-declare="true"/>

1. 定义广播类型交换机，绑定队列

广播类型的交换机，绑定不需要路由键。

<rabbit:fanout-exchange id="spring\_fanout\_exchange" name="spring\_fanout\_exchange" auto-declare="true">  
 <rabbit:bindings>  
 <rabbit:binding queue="spring\_fanout\_queue\_1"/>  
 <rabbit:binding queue="spring\_fanout\_queue\_2"/>  
 </rabbit:bindings>  
</rabbit:fanout-exchange>

1. 定义用于Topic交换机的持久化队列与Topic交换机

持久化队列的创建都是一样的，关键是不同交换机的创建是不同的，topic类型的交换机在绑定队列时需要设置路由键的规则。

<rabbit:queue id="spring\_topic\_queue\_star" name="spring\_topic\_queue\_star" auto-declare="true"/>  
<rabbit:queue id="spring\_topic\_queue\_well" name="spring\_topic\_queue\_well" auto-declare="true"/>  
<rabbit:queue id="spring\_topic\_queue\_well2" name="spring\_topic\_queue\_well2" auto-declare="true"/>  
  
<rabbit:topic-exchange id="spring\_topic\_exchange" name="spring\_topic\_exchange" auto-declare="true">  
 <rabbit:bindings>  
 <rabbit:binding pattern="xx.\*" queue="spring\_topic\_queue\_star"/>  
 <rabbit:binding pattern="xx.#" queue="spring\_topic\_queue\_well"/>  
 <rabbit:binding pattern="wxx.#" queue="spring\_topic\_queue\_well2"/>  
 </rabbit:bindings>  
</rabbit:topic-exchange>

1. 定义rabbitTemplate对象

最重要的一个对象是定义rabbitTemplate对象，该对象用于在代码中发送消息，只要我们将该对象注入到容器中就可以使用了。

<rabbit:template id=”rabbitTemplate” connection-factory=”connectionFactory”/>

1. 测试生产者类

在test包内进行测试。

1. 导入注解@RunWith

[测试需要使用注解 @RunWith(SpringJUnit4ClassRunner.class)](mailto:测试需要使用注解@RunWith(SpringJUnit4ClassRunner.class))

1. 导入注解@ContextConfiguration

我们需要加载Spring的配置文件。因此，[使用注解@ContextConfiguration(location=”classpath:spring-rabbitmq-producer.xml”)](mailto:使用注解@ContextConfiguration(location=”classpath:spring-rabbitmq-producer.xml”))，括号内为配置文件的地址。

1. 注入RabbitmqTemplate对象

使用该对象来发送消息。

@Autowired

private RabbitTemplate rabbitTemplate;

1. 发送消息

发送消息使用rabbitTemplate对象的convertAndSend方法，该方法有很多重载，任君选择。

简单模式：

rabbitTemplate.convertAndSend(“spring\_queue”, “hello\_world”);

广播模式：

rabbitTemplate.convertAndSend(“spring\_fanout\_exchange”, “”, “hello\_fanout”);

通配符模式：

rabbitTemplate.convertAndSend(“spring\_topic\_exchange”, “xx.xx”, “hello\_topic”);

1. Spring整合rabbitmq消费者类
2. 配置pom.xml文件

配置方法与生产者类相同。

1. 配置Spring核心配置文件
2. 定义rabbitmq的连接工厂

定义方法与生产者相同。

1. 定义监听类的Bean对象

Bean对象的id是自己定义的，class是监听类的全包名。

<bean id="springQueueListener" class="com.rabbitmq.listener.SpringQueueListener"/>

1. 定义监听器的容器

容器中存储着监听器，每个监听器监听着对应的队列。

<rabbit:listener-container connection-factory="connectionFactory" auto-declare="true">  
 <rabbit:listener ref="springQueueListener" queue-names="spring\_queue"/>  
 <rabbit:listener ref="fanoutListener1" queue-names="spring\_fanout\_queue\_1"/>  
</rabbit:listener-container>

1. 创建监听类

我们在Spring的配置文件中定义了监听类的Bean对象，监听类会一直持续地监听队列的消息，若有消息，就会调用回调函数，作出相应的操作。

监听类需要实现MessageListener接口，该接口有 重载方法onMessage，这个方法就是一个回调函数，负责响应监听者获取了队列中消息，作出相应的操作。

package com.rabbitmq.listener;  
  
import org.springframework.amqp.core.Message;  
import org.springframework.amqp.core.MessageListener;  
  
public class SpringQueueListener implements MessageListener {  
 @Override  
 public void onMessage(Message message) {  
 System.*out*.println(new String(message.getBody()));  
 }  
}

1. 测试消费者类

在测试类中进行测试。

1. Spring配置文件详解
2. rabbit约束头

Spring配置文件中导入了rabbit约束头以及rabbit约束文件，因此可以使用rabbit命名空间，如下所示：

<rabbit:connection-factory>

1. 定义队列的属性

定义队列时，我们会设置一些属性，与之前没有使用Spring时的代码参数差不多，如下所示：

id: bean的名称

name: queue的名称

auto-declare: 自动创建

auto-delete: 自动删除，最后一个消费者和该队列断开连接后，自动删除队列。

exclusive: 是否独占

durable: 是否持久化

1. SpringBoot整合RabbitMQ生产者
2. 创建SpringBoot工程

添加一个maven模块。

1. 配置pom.xml

SpringBoot项目需要在pom.xml中继承父工程依赖，以及rabbitmq的依赖包。

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.4.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

</dependency>

</dependencies>

1. 配置application.yml

application.yml中配置了连接rabbitmq的连接参数，直接输入rabbitmq就会有提示，不用担心找不到参数名称的问题。

spring:  
 rabbitmq:  
 host: 192.168.211.100  
 port: 5672  
 username:   
 password:   
 virtual-host: /

1. 创建启动类

创建SpringBoot项目的入口，即 启动类。

package com.rabbitmq;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class ProducerApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ProducerApplication.class);  
 }  
}

1. 创建配置类

Spring将交换机，队列等配置在xml文件中，SpringBoot将它们配置在一个配置类中，使用注解的方式。

1. 首先在该配置类上添加@Configuration表示这是一个配置类
2. 配置交换机

添加一个方法，返回交换机对象，并且在该方法上添加@Bean(“bootExchange”)注解，表示这是一个id为bootExchange的Bean对象。

那么，交换机对象如何生成呢？通过ExchangeBuilder可以创建多种类型的交换机，同时可以设置它的属性，最终通过build()函数生成交换机。

public static final String *EXCHANGE\_NAME* = "boot\_topic\_exchange";  
public Exchange bootExchange(){  
 return ExchangeBuilder.*topicExchange*(*EXCHANGE\_NAME*).durable(true).build();  
}

1. 配置队列

同理，使用同样的方法配置队列。

public static final String *QUEUE\_NAME* = "boot\_queue";  
@Bean("bootQueue")  
public Queue bootQueue(){  
 return QueueBuilder.*durable*(*QUEUE\_NAME*).build();  
}

1. 绑定交换机和队列

同样的，添加注解@Bean，不过这次不需要id，因为这个绑定关系是不需要注入进去的。要绑定交换机和队列，就需要知道具体的交换机和队列的名称，因此，我们需要在该函数的参数中传入交换机和队列，并且用@Qualifier注解通过id进行区分不同的交换机和队列。

函数体的内容是使用BindingBuiler的bind方法绑定队列和交换机，以及后面的属性设置。

@Bean  
public Binding bingQueueExchange(@Qualifier("bootQueue")Queue queue, @Qualifier("bootExchange")Exchange exchange){  
 return BindingBuilder.*bind*(queue).to(exchange).with("boot.#").noargs();  
}

（六）测试生产者类

定义完所有的配置后，我们可以编写代码进行测试生产者发送消息的情况了，在test包下创建一个测试类，通过注解的方式进行注入rabbitTemplate，使用该对象发送消息。

@SpringBootTest  
@RunWith(SpringRunner.class)  
public class ProducerTest {  
  
 @Autowired  
 private RabbitTemplate rabbitTemplate;  
  
 @Test  
 public void test(){  
 rabbitTemplate.convertAndSend(*EXCHANGE\_NAME*, "boot.haha", "hello\_world");  
 }  
  
}

1. SpringBoot整合RabbitMQ消费者

消费者中同样要配置pom.xml，application.yml文件，但是不需要设置配置类。同时消费者还要设置监听类，对相应的队列进行监听工作。

使用注解@Component表示这是一个Bean对象，创建一个监听方法，在其上添加注解@RabbitListener，在该注解的参数中需要传入监听的队列的名称（不是id）。在该监方法的参数中传入一个Message对象，该消息对象用于存储从队列中监听到的消息，用于方法体中的操作。

@Component

public class RabbitMQListener{

@RabbitListener(queue=”boot\_queue”)

public void ListenerQueue(Message message){

System.out.println(message);

}

}

注意，这个监听类要和入口类在同一个包下才能被扫描到。