## RabbitMQ

AMQP 是 Advanced Message Queuing Protocol 的简称，它是一个面向消息中间件的开放式标准应用层协议。AMQP定义了这些特性：

* 消息方向
* 消息队列
* 消息路由（包括：点到点和发布-订阅模式）
* 可靠性
* 安全性

RabbitMQ 是以 AMQP 协议实现的一种中间件产品，也称为面向消息的中间件，它可以支持多种操作系统，多种编程语言，几乎可以覆盖所有主流的企业级技术平台。

 RabbitMQ 的基本概念：

* Broker：可以理解为消息队列服务器的实体，是一个中间件应用，负责接收消息生产者的消息，然后将消息发送至消息接收者或者其他的 Broker
* Exchange：消息交换机，是消息第一个到达的地方，消息通过他指定的路由规则分发到不同的消息队列中去，有如下几种类型：

1. Direct：完全按照 key 进行投递，比如，绑定时设置了 Routing Key 为 abc，那么客户端提交消息，只有设置了 Key 为 abc 的才会被投递到队列
2. Topic：对于 Key 进行模式匹配后进行投递，可以使用符号 # 匹配一个或多个词，符号 \* 匹配正好一个词。比如， abc.# 可以匹配 abc.def.ghi ，而 abc.\* 只能匹配 abc.def
3. Fanout：不需要任何 Key，采取广播的模式，一个消息进来时，投递到与该交换机绑定的所以队列

* Queue：消息队列，消息通过发送和路由之后最终到达的地方，到达 Queue 的消息即进入逻辑上等待消费的状态。每个消息都会被发送到一个或多个队列。
* Binding：绑定，将 Exchange 和 Queue 按照路由规则绑定起来
* Routing Key：路由关键字，Exchange 根据该关键字进行消息投递
* Virtual Host：虚拟主机，他是对 Broker 的虚拟划分，将消费者、生产者和依赖的 AMQP 相关结构进行隔离，一般情况都是为了安全考虑。
* Connection：连接，代表生产者、消费者、Broker 之间进行通信的物理网络
* Channel：消息通道，用于连接生产者和消费者的逻辑结构，在客户端的每个连接里，可以建立多个 Channel，每个 Channel 代表一个会话任务，通过 Channel 可以隔离同一个连接的不同交互内容。
* Producer：消息生产者，制造消息并发送消息的程序
* Consumer：消息消费者，接收消息并处理消息的程序

## 资源引入

<dependency>

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

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

</dependency>

## 自动配置

在Spring Boot中实现了RabbitMQ的自动配置，在配置文件中添加如下配置信息

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=test

spring.rabbitmq.password=test

spring.rabbitmq.virtualHost=test

## Spring Boot与RabbitMQ的整合

import org.springframework.amqp.core.AcknowledgeMode;

import org.springframework.amqp.core.Binding;

import org.springframework.amqp.core.BindingBuilder;

import org.springframework.amqp.core.DirectExchange;

import org.springframework.amqp.core.Message;

import org.springframework.amqp.core.Queue;

import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;

import org.springframework.amqp.rabbit.connection.ConnectionFactory;

import org.springframework.amqp.rabbit.core.ChannelAwareMessageListener;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer;

import org.springframework.beans.factory.config.ConfigurableBeanFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

import com.rabbitmq.client.Channel;

/\*\*

\* Qmqp Rabbitmq

\*

\* http://docs.spring.io/spring-amqp/docs/1.4.5.RELEASE/reference/html/

\*

\* @author lkl

\* @version $Id: AmqpConfig.java, v 0.1 2015年11月01日 下午2:05:37 lkl Exp $

\*/

@Configuration

public class AmqpConfig {

public static final String EXCHANGE = "spring-boot-exchange";

public static final String ROUTINGKEY = "spring-boot-routingKey";

@Bean

public ConnectionFactory connectionFactory() {

CachingConnectionFactory connectionFactory = new CachingConnectionFactory();

connectionFactory.setAddresses("127.0.0.1:5672");

connectionFactory.setUsername("guest");

connectionFactory.setPassword("guest");

connectionFactory.setVirtualHost("/");

connectionFactory.setPublisherConfirms(true); //必须要设置

return connectionFactory;

}

@Bean

@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

//必须是prototype类型

public RabbitTemplate rabbitTemplate() {

RabbitTemplate template = new RabbitTemplate(connectionFactory());

return template;

}

/\*\*

\* 针对消费者配置

\* 1. 设置交换机类型

\* 2. 将队列绑定到交换机

\*

\*

FanoutExchange: 将消息分发到所有的绑定队列，无routingkey的概念

HeadersExchange ：通过添加属性key-value匹配

DirectExchange:按照routingkey分发到指定队列

TopicExchange:多关键字匹配

\*/

@Bean

public DirectExchange defaultExchange() {

return new DirectExchange(EXCHANGE);

}

@Bean

public Queue queue() {

return new Queue("spring-boot-queue", true); //队列持久

}

@Bean

public Binding binding() {

return BindingBuilder.bind(queue()).to(defaultExchange()).with(AmqpConfig.ROUTINGKEY);

}

@Bean

public SimpleMessageListenerContainer messageContainer() {

SimpleMessageListenerContainer container = new SimpleMessageListenerContainer(connectionFactory());

container.setQueues(queue());

container.setExposeListenerChannel(true);

container.setMaxConcurrentConsumers(1);

container.setConcurrentConsumers(1);

container.setAcknowledgeMode(AcknowledgeMode.MANUAL); //设置确认模式手工确认

container.setMessageListener(new ChannelAwareMessageListener() {

@Override

public void onMessage(Message message, Channel channel) throws Exception {

byte[] body = message.getBody();

System.out.println("receive msg : " + new String(body));

channel.basicAck(message.getMessageProperties().getDeliveryTag(), false); //确认消息成功消费

}

});

return container;

}

}