😀 纯洁的微笑 (/)

springboot(八)：RabbitMQ详解

📅 2016/11/30

RabbitMQ 即一个消息队列，主要是用来实现应用程序的异步和解耦，同时也能起到消息缓冲，消息分发的作用。

消息中间件在互联网公司的使用中越来越多，刚才还看到新闻阿里将RocketMQ捐献给了apache，当然了今天的主角还是讲RabbitMQ。消息中间件最主要的作用是解耦，中间件最标准的用法是生产者生产消息传送到队列，消费者从队列中拿取消息并处理，生产者不用关心是谁来消费，消费者不用关心谁在生产消息，从而达到解耦的目的。在分布式的系统中，消息队列也会被用在很多其它的方面，比如：分布式事务的支持，RPC的调用等等。

以前一直使用的是ActiveMQ，在实际的生产使用中也出现了一些小问题，在网络查阅了很多的资料后，决定尝试使用RabbitMQ来替换ActiveMQ，RabbitMQ的高可用性、高性能、灵活性等一些特点吸引了我们，查阅了一些资料整理出此文。

RabbitMQ介绍

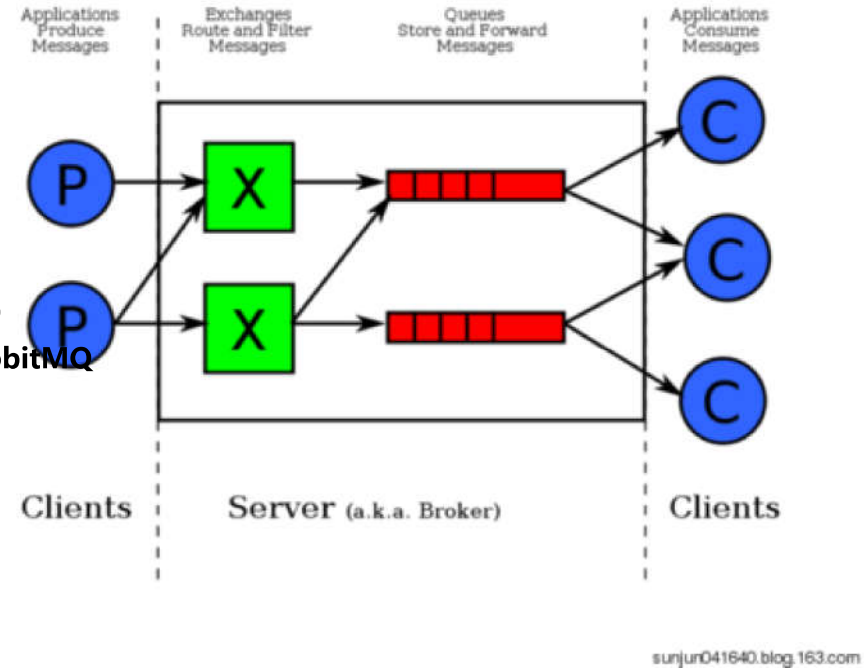
RabbitMQ是实现AMQP（高级消息队列协议）的消息中间件的一种，最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。RabbitMQ主要是为了实现系统之间的双向解耦而实现的。当生产者大量产生数据时，消费者无法快速消费，那么需要一个中间层。保存这个数据。

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。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。

相关概念

通常我们谈到队列服务, 会有三个概念： 发消息者、队列、收消息者，RabbitMQ 在这个基本概念之上, 多做了一层抽象, 在发消息者和 队列之间, 加入了交换器 (Exchange). 这样发消息者和队列就没有直接联系, 转而变成发消息者把消息给交换器, 交换器根据调度策略再把消息再给队列。



- 左侧 P 代表 生产者，也就是往 RabbitMQ 发消息的程序。
- 中间即是 RabbitMQ，*其中包括了 交换机 和 队列。*
- 右侧 C 代表 消费者，也就是往 RabbitMQ 拿消息的程序。

那么，*其中比较重要的概念有 4 个，分别为：虚拟主机，交换机，队列，和绑定。*

- 虚拟主机：一个虚拟主机持有一组交换机、队列和绑定。为什么需要多个虚拟主机呢？很简单，RabbitMQ当中，*用户只能在虚拟主机的粒度进行权限控制。* 因此，如果需要禁止A组访问B组的交换机/队列/绑定，必须为A和B分别创建一个虚拟主机。每一个RabbitMQ服务器都有一个默认的虚拟主机“/”。
- 交换机：*Exchange 用于转发消息，但是它不会做存储* ，如果没有 Queue bind 到 Exchange 的话，它会直接丢弃掉 Producer 发送过来的消息。 这里有一个比较重要的概念：**路由键** 。消息到交换机的时候，交互机会转发到对应的队列中，那么究竟转发到哪个队列，就要根据该路由键。
- 绑定：也就是交换机需要和队列相绑定，这其中如上图所示，是多对多的关系。

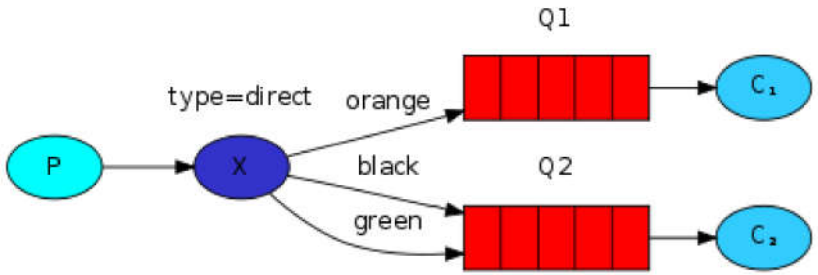
交换机(Exchange)

交换机的功能主要是接收消息并且转发到绑定的队列，交换机不存储消息，在启用ack模式后，交换机找不到队列会返回错误。交换机有四种类型：Direct, topic, Headers and Fanout

- Direct：direct 类型的行为是”先匹配, 再投送”. 即在绑定时设定一个 **routing_key**, 消息的 **routing_key** 匹配时, 才会被交换器投送到绑定的队列中去.
- Topic：按规则转发消息（最灵活）
- Headers：设置header attribute参数类型的交换机
- Fanout：转发消息到所有绑定队列

Direct Exchange

Direct Exchange是RabbitMQ默认的交换机模式，也是最简单的模式，根据key全文匹配去寻找队列。



第一个 X - Q1 就有一个 binding key，名字为 orange； X - Q2 就有 2 个 binding key，名字为 black 和 green。*当消息中的 路由键 和 这个 binding key 对应上的时候，那么就知道了该消息去到哪一个队列中。*

Ps：为什么 X 到 Q2 要有 black，green，2个 binding key呢，一个不就行了吗？ - 这个主要是因为可能又有 Q3，而Q3只接受 black 的信息，而Q2不仅接受black 的信息，还接受 green 的信息。

Topic Exchange

Topic Exchange 转发消息主要是根据通配符。 在这种交换机下，队列和交换机的绑定会定义一种路由模式，那么，通配符就要在这种路由模式和路由键之间匹配后交换机才能转发消息。

在这种交换机模式下：

- 路由键必须是一串字符，用句号（ . ） 隔开，比如说 agreements.us，或者 agreements.eu.stockholm 等。
- 路由模式必须包含一个 星号（ * ），主要用于匹配路由键指定位置的一个单词，比如说，一个路由模式是这样子：agreements..b.*，那么就只能匹配路由键是这样子的：第一个单词是 agreements，第四个单词是 b。 井号（#）就表示相当于一个或者多个单词，例如一个匹配模式是agreements.eu.berlin.#，那么，以agreements.eu.berlin开头的路由键都是可以的。

具体代码发送的时候还是一样，第一个参数表示交换机，第二个参数表示routing key，第三个参数即消息。如下：

```
rabbitTemplate.convertAndSend("testTopicExchange","key1.a.c.key2"," this is  RabbitMQ!");
```

topic 和 direct 类似, 只是匹配上支持了”模式”, 在”点分”的 routing_key 形式中, 可以使用两个通配符:

- * 表示一个词.
- # 表示零个或多个词.

Headers Exchange

headers 也是根据规则匹配, 相较于 direct 和 topic 固定地使用 routing_key , headers 则是一个自定义匹配规则的类型. 在队列与交换器绑定时, 会设定一组键值对规则, 消息中也包括一组键值对(headers 属性), 当这些键值对有一对, 或全部匹配时, 消息被投送到对应队列.

Fanout Exchange

Fanout Exchange 消息广播的模式，不管路由键或者是路由模式，*会把消息发给绑定给它的全部队列*，如果配置了routing_key会被忽略。

springboot集成RabbitMQ

springboot集成RabbitMQ非常简单，如果只是简单的使用配置非常少，springboot提供了spring-boot-starter-amqp项目对消息各种支持。

简单使用

1、配置pom包，主要是添加spring-boot-starter-amqp的支持

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

2、配置文件

配置rabbitmq的安装地址、端口以及账户信息

```
spring.application.name=Spring-boot-rabbitmq

spring.rabbitmq.host=192.168.0.86
spring.rabbitmq.port=5672
spring.rabbitmq.username=admin
spring.rabbitmq.password=123456
```

3、队列配置

```
@Configuration
public class RabbitConfig {

    @Bean
    public Queue Queue() {
        return new Queue("hello");
    }

}
```

RabbitMQ介绍
- 相关概念
- 交换机(Exchange)

springboot集成RabbitMQ
- 简单使用
- 发送者
- 多使用
- 高级使用

参考

3、发送者

RabbitTemplate是springboot 提供的默认实现

```
public class HelloSender {

    @Autowired
    private AmqpTemplate rabbitTemplate;

    public void send() {
        String context = "hello " + new Date();
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("hello", context);
    }

}
```

4、接收者

```
@Component
@RabbitListener(queues = "hello")
public class HelloReceiver {

    @RabbitHandler
    public void process(String hello) {
        System.out.println("Receiver : " + hello);
    }

}
```



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

    @Autowired
    private HelloSender helloSender;

    @Test
    public void hello() throws Exception {
        helloSender.send();
    }

}
```

注意，发送者和接收者的queue name必须一致，不然不能接收

多对多使用

一个发送者，N个接收者或者N个发送者和N个接收者会出现什么情况呢?

一对多发送

对上面的代码进行了小改造,接收端注册了两个Receiver,Receiver1和Receiver2，发送端加入参数计数，接收端打印接收到的参数，下面是测试代码，发送一百条消息，来观察两个接收端的执行效果

```
@Test
public void oneToMany() throws Exception {
    for (int i=0;i<100;i++){
        neoSender.send(i);
    }
}
```

结果如下：

```
Receiver 1: Spring boot neo queue ****** 11
Receiver 2: Spring boot neo queue ****** 12
Receiver 2: Spring boot neo queue ****** 14
Receiver 1: Spring boot neo queue ****** 13
Receiver 2: Spring boot neo queue ****** 15
Receiver 1: Spring boot neo queue ****** 16
Receiver 1: Spring boot neo queue ****** 18
Receiver 2: Spring boot neo queue ****** 17
Receiver 2: Spring boot neo queue ****** 19
Receiver 1: Spring boot neo queue ****** 20
```

根据返回结果得到以下结论

> 一个发送者，N个接受者,经过测试会均匀的将消息发送到N个接收者中

多对多发送

复制了一份发送者，加入标记，在一百个循环中相互交替发送

```
@Test
    public void manyToMany() throws Exception {
        for (int i=0;i<100;i++){
            neoSender.send(i);
            neoSender2.send(i);
        }
}
```


结果如下：

```
Receiver 1: Spring boot neo queue ****** 20
Receiver 2: Spring boot neo queue ****** 20
Receiver 1: Spring boot neo queue ****** 21
Receiver 2: Spring boot neo queue ****** 21
Receiver 1: Spring boot neo queue ****** 22
Receiver 2: Spring boot neo queue ****** 22
Receiver 1: Spring boot neo queue ****** 23
Receiver 2: Spring boot neo queue ****** 23
Receiver 1: Spring boot neo queue ****** 24
Receiver 2: Spring boot neo queue ****** 24
Receiver 1: Spring boot neo queue ****** 25
Receiver 2: Spring boot neo queue ****** 25
```

结论：和一对多一样，接收端仍然会均匀接收到消息

高级使用

对象的支持

springboot以及完美的支持对象的发送和接收，不需要格外的配置。

```
//发送者
public void send(User user) {
        System.out.println("Sender object: " + user.toString());
        this.rabbitTemplate.convertAndSend("object", user);
}

//接收者
@RabbitHandler
public void process(User user) {
    System.out.println("Receiver object : " + user);
}
```


结果如下：

```
Sender object: User{name='neo', pass='123456'}
Receiver object : User{name='neo', pass='123456'}
```

Topic Exchange

topic 是RabbitMQ中最灵活的一种方式，可以根据routing_key自由的绑定不同的队列

首先对topic规则配置，这里使用两个队列来测试

```
@Configuration
public class TopicRabbitConfig {

    final static String message = "topic.message";
    final static String messages = "topic.messages";

    @Bean
    public Queue queueMessage() {
        return new Queue(TopicRabbitConfig.message);
    }

    @Bean
    public Queue queueMessages() {
        return new Queue(TopicRabbitConfig.messages);
    }

    @Bean
    TopicExchange exchange() {
        return new TopicExchange("exchange");
    }

    @Bean
    Binding bindingExchangeMessage(Queue queueMessage, TopicExchange exchange) {
        return BindingBuilder.bind(queueMessage).to(exchange).with("topic.message");
    }

    @Bean
    Binding bindingExchangeMessages(Queue queueMessages, TopicExchange exchange) {
        return BindingBuilder.bind(queueMessages).to(exchange).with("topic.#");
    }
}
```

使用queueMessages同时匹配两个队列，queueMessage只匹配”topic.message”队列

```
public void send1() {
        String context = "hi, i am message 1";
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("exchange", "topic.message", context);
}
public void send2() {
        String context = "hi, i am messages 2";
        System.out.println("Sender : " + context);
        this.rabbitTemplate.convertAndSend("exchange", "topic.messages", context);
}
```


发送send1会匹配到topic.#和topic.message 两个Receiver都可以收到消息，发送send2只有topic.#可以匹配所有只有Receiver2监听到消息

Fanout Exchange

Fanout 就是我们熟悉的广播模式或者订阅模式，给Fanout交换机发送消息，绑定了这个交换机的所有队列都收到这个消息。

Fanout 相关配置

```
@Configuration
public class FanoutRabbitConfig {

    @Bean
    public Queue AMessage() {
        return new Queue("fanout.A");
    }

    @Bean
    public Queue BMessage() {
        return new Queue("fanout.B");
    }

    @Bean
    public Queue CMessage() {
        return new Queue("fanout.C");
    }

    @Bean
    FanoutExchange fanoutExchange() {
        return new FanoutExchange("fanoutExchange");
    }

    @Bean
    Binding bindingExchangeA(Queue AMessage,FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(AMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeB(Queue BMessage, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(BMessage).to(fanoutExchange);
    }

    @Bean
    Binding bindingExchangeC(Queue CMessage, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(CMessage).to(fanoutExchange);
```

RabbitMQ介绍
- 相关概念
- 交换机(Exchange)

springboot集成RabbitMQ
- 简单使用
- 多对多使用
- 高级使用

参考

```
    }

}
```

这里使用了A、B、C三个队列绑定到Fanout交换机上面，发送端的routing_key写任何字符都会被忽略：


```
public void send() {
    String context = "hi, fanout msg ";
    System.out.println("Sender : " + context);
    this.rabbitTemplate.convertAndSend("fanoutExchange","", context);
}
```

结果如下：

```
Sender : hi, fanout msg
...
fanout Receiver B: hi, fanout msg
fanout Receiver A  : hi, fanout msg
fanout Receiver C: hi, fanout msg
```

结果说明，绑定到fanout交换机上面的队列都收到了消息

示例代码-github (https://github.com/ityouknow/spring-boot-examples)

示例代码-码云 (https://gitee.com/ityouknow/spring-boot-examples)

参考

RabbitMQ 使用参考 (https://www.zouyesheng.com/rabbitmq.html)

RabbitMQ：Spring 集成 RabbitMQ 与其概念，消息持久化，ACK机制 (https://github.com/401Studio/WeekLearn/issues/2)

> **作者:纯洁的微笑**
> **出处:www.ityouknow.com (http://www.ityouknow.com)**
> **版权所有，欢迎保留原文链接进行转载：)**
>
> **springboot集成RabbitMQ**
>
> - 简单使用
> - 多对多使用
> - 高级使用
>
> **参考**

扫码关注有惊喜

(转载本站文章请注明作者和出处 纯洁的微笑-ityouknow (http://www.ityouknow.com))

Show Disqus Comments

5 (https://github.com/ityouknow/blog-comments/issues/157) 条评论 hyb-bobo

说点什么

ⓘ 支持 Markdown 语法 (https://guides.github.com/features/mastering-markdown/) 评论

Jasonwang1117 (https://github.com/Jasonwang1117) 发表于 13 天前

接收者少了注解么?小白不懂,感谢大神的分享

WuXianquan (https://github.com/WuXianquan) 发表于 21 天前

HelloSender类少了给@component (https://github.com/component)注解

liuminda (https://github.com/liuminda) 发表于 大约 1 个月前

多谢大神的分享

zpcandzhj (https://github.com/zpcandzhj) 发表于 大约 2 个月前

一对多发送出现了消息丢失现象，老铁能否剖析一下~

```
Receiver1 : hello 0 2018-08-05 18:25:28
Receiver2 : hello 1 2018-08-05 18:25:28
Receiver1 : hello 4 2018-08-05 18:25:29
Receiver2 : hello 5 2018-08-05 18:25:29
Receiver1 : hello 8 2018-08-05 18:25:30
Receiver2 : hello 9 2018-08-05 18:25:30
Receiver1 : hello 12 2018-08-05 18:25:30
Receiver2 : hello 13 2018-08-05 18:25:31
Receiver1 : hello 16 2018-08-05 18:25:31
Receiver2 : hello 17 2018-08-05 18:25:31
Receiver1 : hello 20 2018-08-05 18:25:32
```

```
Receiver2 : hello 21 2018-08-05 18:25:32
Receiver1 : hello 24 2018-08-05 18:25:33
Receiver2 : hello 25 2018-08-05 18:25:33
Receiver1 : hello 28 2018-08-05 18:25:34
Receiver2 : hello 29 2018-08-05 18:25:34
Receiver1 : hello 32 2018-08-05 18:25:35
Receiver2 : hello 33 2018-08-05 18:25:35
Receiver1 : hello 36 2018-08-05 18:25:35
Receiver2 : hello 37 2018-08-05 18:25:36
Receiver1 : hello 40 2018-08-05 18:25:36
Receiver2 : hello 41 2018-08-05 18:25:36
Receiver1 : hello 44 2018-08-05 18:25:37
Receiver2 : hello 45 2018-08-05 18:25:37
Receiver1 : hello 48 2018-08-05 18:25:38
Receiver2 : hello 49 2018-08-05 18:25:38
Receiver1 : hello 52 2018-08-05 18:25:39
Receiver2 : hello 53 2018-08-05 18:25:39
Receiver1 : hello 56 2018-08-05 18:25:39
Receiver2 : hello 57 2018-08-05 18:25:40
Receiver1 : hello 60 2018-08-05 18:25:40
Receiver2 : hello 61 2018-08-05 18:25:40
Receiver1 : hello 64 2018-08-05 18:25:41
Receiver2 : hello 65 2018-08-05 18:25:41
Receiver1 : hello 68 2018-08-05 18:25:42
Receiver2 : hello 69 2018-08-05 18:25:42
Receiver1 : hello 72 2018-08-05 18:25:43
Receiver2 : hello 73 2018-08-05 18:25:43
Receiver1 : hello 76 2018-08-05 18:25:44
Receiver2 : hello 77 2018-08-05 18:25:44
```

RabbitMQ介绍

- 相关概念
- 交换机(Exchange)

springboot集成RabbitMQ

- 简单使用
- 多对多使用
- 高级使用

参考

```
Receiver1 : hello 80 2018-08-05 18:25:44
Receiver2 : hello 81 2018-08-05 18:25:45
Receiver1 : hello 84 2018-08-05 18:25:45
Receiver2 : hello 85 2018-08-05 18:25:45
Receiver1 : hello 88 2018-08-05 18:25:46
Receiver2 : hello 89 2018-08-05 18:25:46
Receiver1 : hello 92 2018-08-05 18:25:47
Receiver2 : hello 93 2018-08-05 18:25:47
Receiver1 : hello 96 2018-08-05 18:25:48
Receiver2 : hello 97 2018-08-05 18:25:48
```

RabbitMQ介绍

- 相关概念
- 交换机(Exchange)

springboot集成RabbitMQ

- 简单使用
- 多对多使用
- 高级使用

参考

yeasheng (https://github.com/yeasheng) 发表于 3 个月前

赞！

Post Directory