教程：

<http://blog.csdn.net/lmj623565791/article/category/2386657>

<http://blog.csdn.net/column/details/rabbitmq.html>

基本配置：<https://www.cnblogs.com/wade-luffy/p/6003668.html>

官网： <https://www.rabbitmq.com/>

常用命令：<http://www.linuxidc.com/Linux/2016-10/136493.htm>

# 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。

# 安装

rabbitMQ是一个在AMQP协议标准基础上完整的，可服用的企业消息系统。它遵循Mozilla Public License开源协议，采用 Erlang 实现的工业级的消息队列(MQ)服务器，Rabbit MQ 是建立在Erlang OTP平台上。

## Windows下RabbitMQ的安装

### 安装Erlang环境

RabbitMQ依赖于Erlang环境

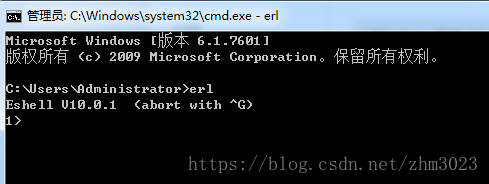
下载地址：https://www.erlang.org/downloads，本文选择OTP 21.0.1 Windows 64-bit Binary File (91707927)

设置环境变量，新建ERLANG\_HOME



修改环境变量path，增加Erlang变量至path，%ERLANG\_HOME%\bin;

打开cmd命令框，输入erl



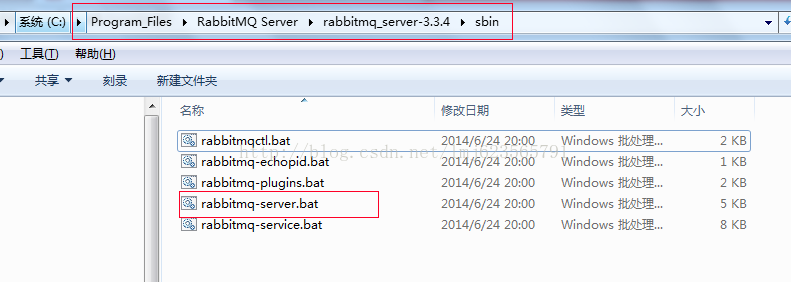
### 安装RabbitMQserver

下载RabbitMQ，地址：http://www.rabbitmq.com/releases/rabbitmq-server/v3.3.4/rabbitmq-server-3.3.4.exe ，双击安装即可

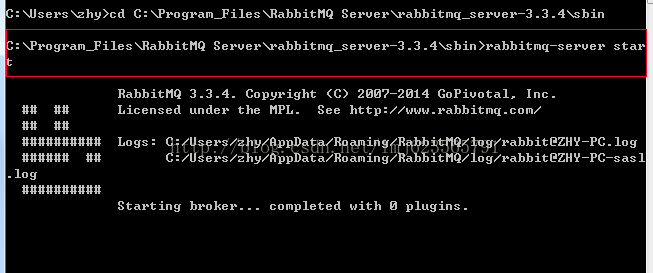
下载rabbit-client.jar ，Java代码时需要导入。地址：http://www.rabbitmq.com/releases/rabbitmq-java-client/v3.3.4/rabbitmq-java-client-bin-3.3.4.zip

安装完成后，在RabbitMQ的安装目录的sbin先会有：rabbitmq-server.bat

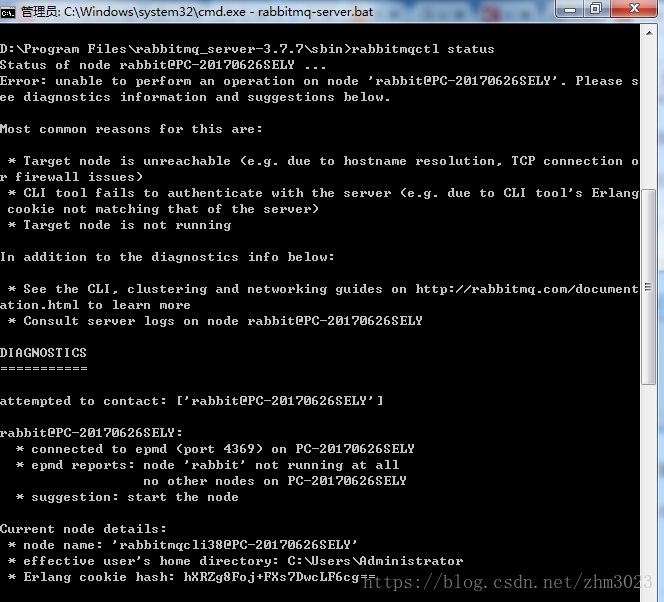
例如：



在cmd下：进入sbin目录，运行rabbitmq-server start，启动mq



打开cmd命令框，切换至D:\Program Files\rabbitmq\_server-3.7.7\sbin目录下，输入rabbitmqctl status



说明rabbmitmq未启动，继续下面操作。

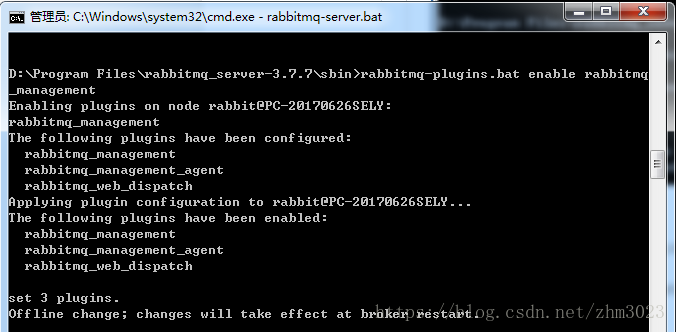
安装插件，命令：rabbitmq-plugins.bat enable rabbitmq\_management,出现

解决方法：

将 C:\Users\Administrator\.erlang.cookie 同步至C:\Windows\System32\config\systemprofile\.erlang.cookie

同时删除：C:\Users\Administrator\AppData\Roaming\RabbitMQ目录

输入命令：rabbitmq-plugins.bat enable rabbitmq\_management ，出现下面信息表示插件安装成功：



<https://blog.csdn.net/zhm3023/article/details/82217222>

### 配置可视化界面

输入命令rabbitmq-plugins enable rabbitmq\_management，这样就可以添加可视化插件了。

查看可视化插件是否成功：

在web浏览器中输入地址：http://127.0.0.1:15672/

输入默认账号: guest   密码: guest

就可以登录查看rabbitmq里的资源信息。

备注：可以运行 rabbitmqctl stop 命令来停止服务

可以运行 rabbitmq-server  -detached命令来重启服务并后台运行。

RabbitMQ的日志信息，可以在 C:/Users/Administrator/AppData/Roaming/RabbitMQ/log/文件夹下进行查看，如：C:/Users/Administrator/AppData/Roaming/RabbitMQ/log/rabbit@mywin-PC.log

## Linux下安装rabbitMQ

### 安装RabbitMQ server

下载RabbitMQ安装源：http://www.rabbitmq.com/install-rpm.html

http://www.rabbitmq.com/releases/rabbitmq-server/v3.5.2/rabbitmq-server-3.5.2-1.noarch.rpm

在CentOS6 64的linux下，如果没有安装epel,可以运行以下命令安装epel。

rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86\_64/epel-release-6-8.noarch.rpm

利用wget下载rabbitmq-server的rpm源。

wget http://www.rabbitmq.com/releases/rabbitmq-server/v3.5.2/rabbitmq-server-3.5.2-1.noarch.rpm

RabbitMQ server 的安装步骤:

运行yum install rabbitmq-server-3.5.2-1.noarch.rpm  进行安装

ps:这里不用 rpm -ivh rabbitmq-server-3.5.2-1.noarch.rpm进行安装，而是用yum install rabbitmq-server-3.5.2-1.noarch.rpm 进行安装

rabbitmq server,因为它会自动下载安装rabbitmq-server所需要的依赖安装包,如Erlang安装包等。

安装rabbitmq-server为启动服务:

[root@mail rabbitmq]# whereis rabbitmqctl

[root@mail rabbitmq]# chkconfig rabbitmq-server on

[root@mail rabbitmq]# /sbin/service rabbitmq-server start

运行 rabbitmqctl  status 查看RabbitMQ运行的状态

[root@iZ23x0cj5yiZ rabbitmq]# /usr/sbin/rabbitmqctl  status

### 配置可视化界面

输入命令rabbitmq-plugins enable rabbitmq\_management，这样就可以添加可视化插件了。

输入账号: mytest  密码: mytest

这里可以先简单的输入以下的命令,添加mytest用户，及对mytest用户进行授权,使其可以用web的方式进行访问。

[root@iZ23x0cj5yiZ ~]# rabbitmqctl add\_user mytest mytest

[root@iZ23x0cj5yiZ ~]# rabbitmqctl set\_user\_tags mytest administrator

[root@iZ23x0cj5yiZ ~]# rabbitmqctl set\_permissions -p / mytest '.\*' '.\*' '.\*'

[root@iZ23x0cj5yiZ ~]# rabbitmqctl list\_permissions

添加mytest用户及授权之后,就可以用ip的形式进行访问了。

# 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。

## 配置pom

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>

## 配置文件

配置rabbitmq的安装地址、端口以及账户信息

spring.application.name=spirng-boot-rabbitmq

spring.rabbitmq.host=192.168.0.86

spring.rabbitmq.port=5672

spring.rabbitmq.username=admin

spring.rabbitmq.password=123456

## 队列配置

@Configuration

public class RabbitConfig {

@Bean

public Queue Queue() {

return new Queue("hello");

}

}

## 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);

}

}

## 接收者

@Component

@RabbitListener(queues = "hello")

public class HelloReceiver {

@RabbitHandler

public void process(String hello) {

System.out.println("Receiver : " + hello);

}

}

## 测试

@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: spirng boot neo queue \*\*\*\*\*\* 11

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 12

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 14

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 13

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 15

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 16

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 18

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 17

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 19

Receiver 1: spirng 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: spirng boot neo queue \*\*\*\*\*\* 20

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 20

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 21

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 21

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 22

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 22

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 23

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 23

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 24

Receiver 2: spirng boot neo queue \*\*\*\*\*\* 24

Receiver 1: spirng boot neo queue \*\*\*\*\*\* 25

Receiver 2: spirng 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

# 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);

}

//接受者

@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);

}

}

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