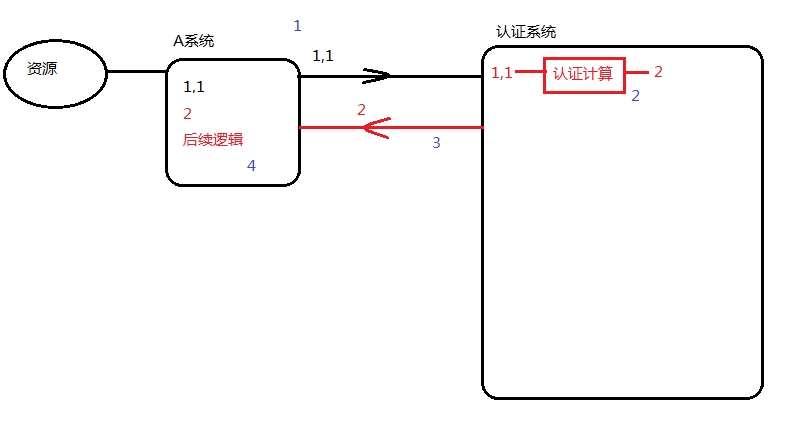
消息队列

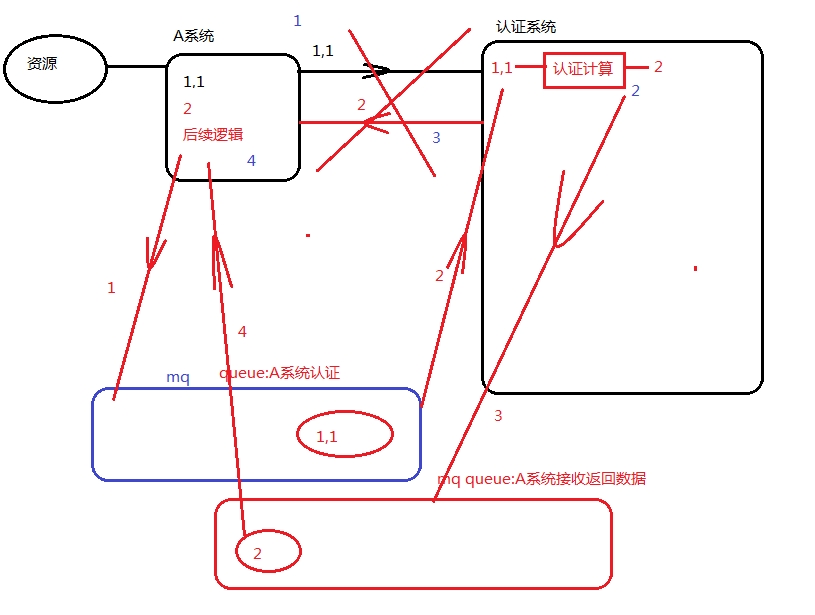
2018年9月15日 16:14

MQ:message queue

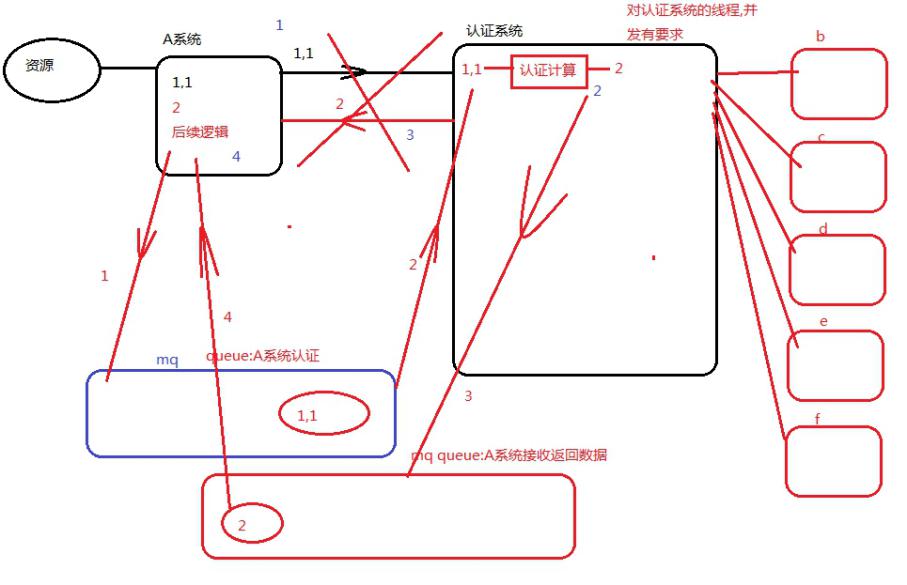
通信解耦

通信认证过程中,一旦通信步骤1,3出现问题,整个 认证需要重新开始,出现了通信过程的强耦合;

需要通过消息队列对2个系统间的通信进行解耦 的操作

消峰限流

分区 day05all 的第 23页

并发非常高的情况下,直接通信的过程会导致由于并发能力的限制,很多的请求无法直接响应,需

要利用消息队列完成消峰限流.

消息队列

Rabbitmq(Haproxy),Kafka(天生分布式)

常见的系统解耦的消息队列.

学习其中的5种工作模式,很多的情况下,消息队列可以配合redis缓存完成非常复杂的业务逻辑;

消息队列安装

2018年8月17日 13:56

并发语言erlang(**云主机和虚拟机镜像无需安装**)

并发语言的特点,并发能力非常高,性能高;

可以控制cpu资源(核,线程);

rabbitmq中存在各种不同语言环境的客户端,所以需要一个稳定的并发能力的语言来编写一个并发的组件(交换机)

安装erlang的运行环境(虚拟机不需要做)

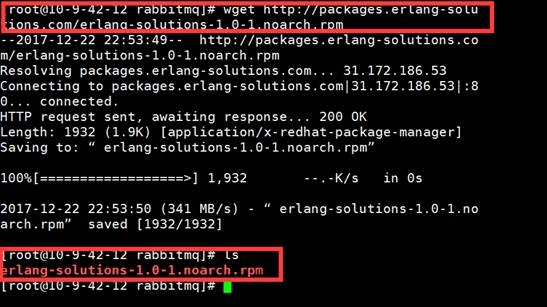
安装erlang语言，需要安装30-50分钟。centOs-empty镜像已经安装好的,erlang不需安装。OTP18,对应老版本

**erlang安装**

**下载**

**wget http://packages.erlang-solutions.com/erlang- solutions-1.0-1.noarch.rpm**

创建一个rabbitmq的目录，将erlang语言rpm包下载。

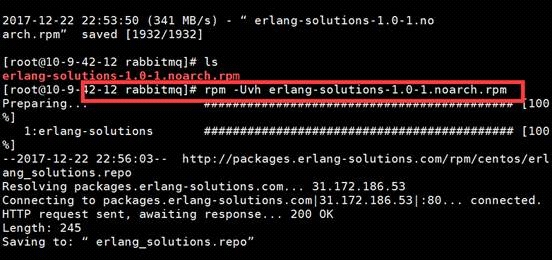
**安装**

使用如下命令进行erlang的安装：

**rpm -Uvh erlang-solutions-1.0.1.noarch.rpm**

-U：update packages，安装的时候所有的包自动更新。

分区 day05all 的第 25页

**yum -y install erlang**

这个过程非常缓慢，需要下载将近60个rpm安装包，一旦发现失败，再执行一次yum -y install erlang，失败的包会重新下载安

装，如再失败再重新执行，直到所有安装成功。

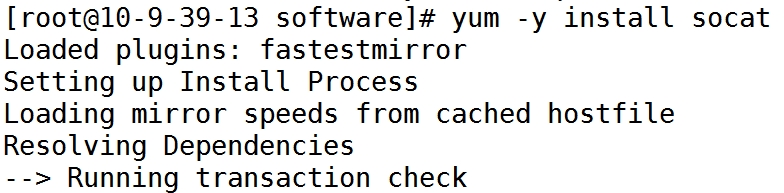
**nothing to do**

**Rabbitmq的安装和启动**

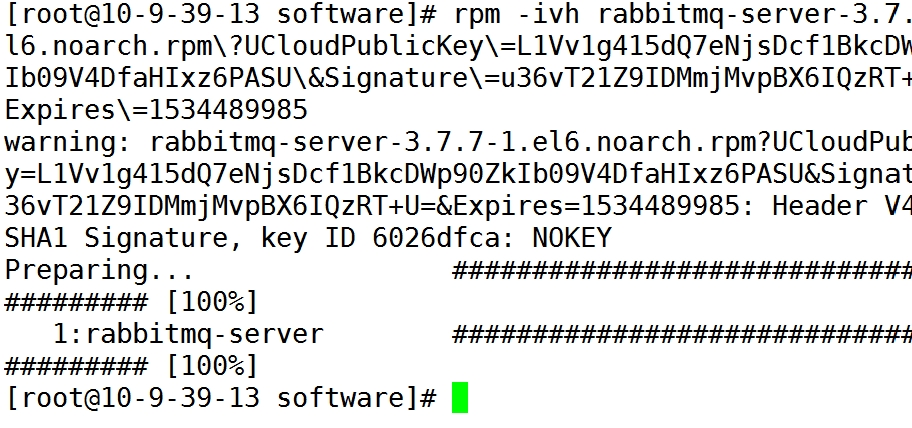
上传包安装，提示依赖socat

**安装socat**

yum -y install socat

**安装rabbit的rpm包**

分区 day05all 的第 26页

rabbitmq安装成功了，但是如果需要对外访问使用，开启远程登录用户的权限，默认只允许localhost

**web管理应用访问端口是 15672**

**程序（生产者，消费者客户端程序）访问端口5672**

配置远程连接权限，配置用户登录权限，需要一个

rabbitmq.config的配置文件，放到/etc/rabbitmq目录中

模板文件直接修改保存即可，/usr/share/doc/rabbitmq-

server-3.7.7/rabbitmq.config.example

将其拷贝到/etc/rabbitmq下，改名rabbitmq.config

**打开文件修改**

**61行（3.6.1是64行）**

%% {loopback\_users, []},

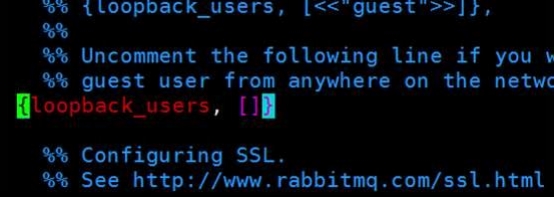
修改1：去掉前面的两个%%，

修改2：去掉最后面的逗号，保存。

修改结果如下：

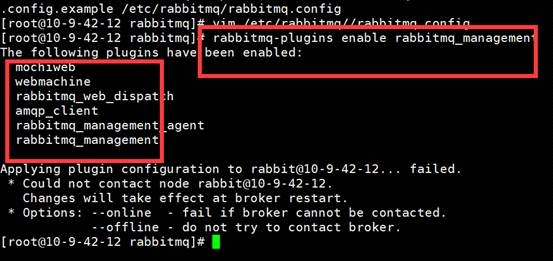
{loopback\_users, []}

分区 day05all 的第 27页

**开启后台管理插件（启动web应用，可以通过应用操作rabbitmq，查看监控）**

执行下面命令，开启web界面管理工具

#rabbitmq-plugins enable rabbitmq\_management

如上图界面，即为开启成功。

启动rabbitmq

service rabbitmq-server start不可以，环境暂时不支持服务的绑定启动,erlang.cookie。

启动rabbitmq出现erlang的cookie文件不能读取,缺少权限

chown rabbitmq:rabbitmq

/var/lib/rabbitmq/.erlang.cookie

rabbitmq3.7.7启动命令文件在

/usr/lib/rabbitmq/bin

**[root@10-9-39-13 bin]# ./rabbitmq-server start**

**./不能缺少，少了当前路径的指向./会到环境中寻找启动命令，相当用用service启动一样。**

分区 day05all 的第 28页

**用用service启动一样。**

**在开启端口情况下，这时就可以利用web的浏览器登录rabbitmq服务查看内容了；**

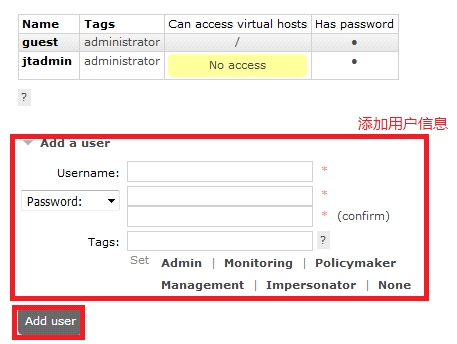
分区 da

消息队列的用户和虚拟机

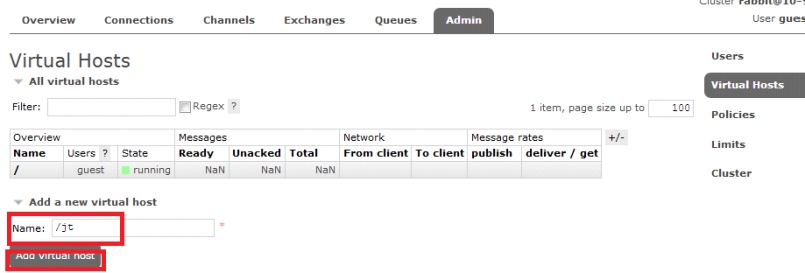
2018年9月17日 8:49

用户:赋予操作rabbitmq权限的登录名和密码

添加独立的登录用户

虚拟机(virtual host):每个用户需要绑定一个虚拟机使用(资源, 内存,cpu等划分的一个取余),不同的虚拟机虽然在同一个rabbitmq中存在,但是绑定的用户不同,管理的内容也不同;在不 绑定虚拟机的情况下无法操作rabbitmq

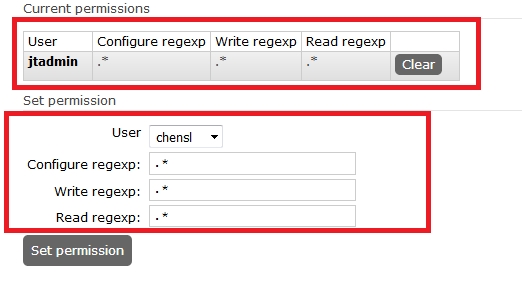
分区 day06all 的第 1页

虚拟机和用户的绑定:

3.7.7登录用户会自动和新建虚拟机绑定;

/jt和jtadmin绑定

点击虚拟机,绑定用户jtadmin

分区 day06all 的第 2页

rabbitmq的核心概念

2018年9月17日 9:16

生产者

连接rabbitmq操作队列,存放发送消息的客户端.这个客户端可 以是不同的语言编写的逻辑;需要编写代码逻辑的;

消费者

连接rabbitmq操作队列的代码客户端,监听队列,从中获取消息 消费,拿到消息后执行消费的逻辑代码

交换机

rabbitmq作为消息队列,解耦,消峰限流,需要并发能力极高(大量的客户端连接并发使用rabbitmq,消息也需要大量的并发完

成发送);并不会允许客户端来完成这种并发的需求.基于erlang

开发的rabbitmq核心组件;根据配置和要求,连接生产者,将大 量的消息并发发送到绑定的队列;

绑定

在使用rabbitmq时存在绑定的关系,队列与交换机绑定,没有与 这个交换机绑定的队列时不可能从交换机里获取消息的;

生产者与交换机也进行绑定(使用关系)

连接

客户端链接rabbitmq的消息队列技术,底层需要一个基于TCP 长链接的对象 connection

信道

基于底层tcp场链接的短连接,信道channel

队列

queue,所有的消息经过交换机发送存储在队列中,队列的另一 头绑定消费者;消费逻辑可以由代码确定,例如消费的一条消

息什么时候删除.

分区 day06all 的第 3页

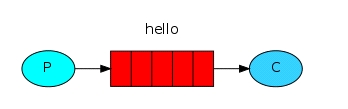
息什么时候删除.

分区 day06all 的第 4页

RabbitMQ的工作模式

2018年7月16日 星期一 下午 4:24

1. simple简单模式

P:消息的生产者;代码连接时（任何支持的代码python,java,c#）,将需要传递的消息,放到一个默认的交换机里,这个交换机在rabbitmq

中绑定所有的队列;

AMQP DEFAULT

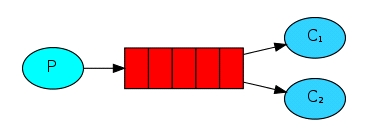
The default exchange is implicitly bound to every queue,

with a routing key equal to the queue name.

C:消息的消费者,异步监听队列,一旦发现任何新的消息,获取消息消费

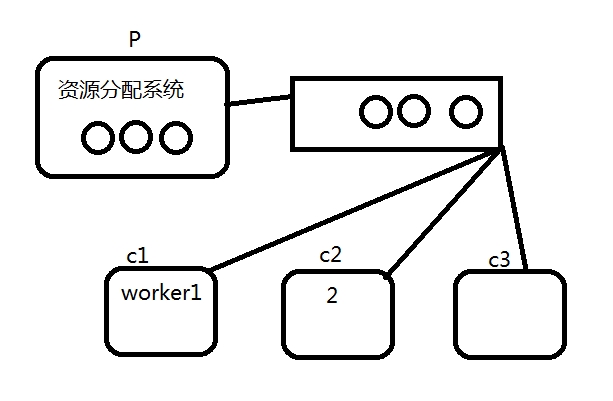
应用场景:短信消息,qq,

1. work工作模式(资源竞争)

生产者P:将消息发送到队列

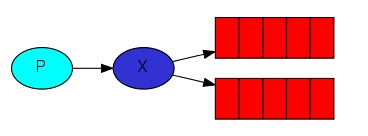
队列被多个消息消费者监听,一旦发现新的消息生成,将会根据自己的能力(性能,资源空闲率);

应用场景:秒杀,抢红包,大型项目的资源调度

1. 发布订阅模式(publish/fanout共享资源)

分区 day06all 的第 5页

3发布订阅模式(publish/fanout共享资源)

x:代表rabbitmq一种核心组件交换机;

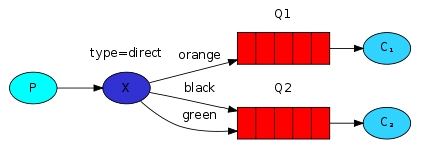
P:生产者不在直接将消息发送到队列中

交换机分发消息性能更高,是基于erlang语言开发的

一条消息,在交换机中,被复制同时发送到多个队列;

应用场景:群发,广告.

4 routing路由模式

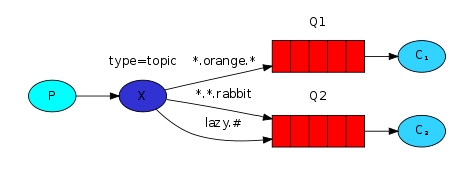
生产者发送消息,消息携带一个路由key

每个队列在绑定交换机时也有一个路由key

X交换机,根据消息携带的路由key,匹配具有相同路由key的queue队列

使用场景:错误消息的监听;

5 topic主题模式

生产者发送携带具体的路由key的消息

假设:路由key haha.orange.haha发送q1

haha. haha.haha.rabbit.haha 在交换机保留(ready) lazy.haah.haha.lazy 发送q2

lazy.orange.lazy 1,2全发

\*号:匹配一个单词(没有特殊符号的字符串)

分区 day06all 的第 6页

\*号:匹配一个单词(没有特殊符号的字符串)

#号:匹配所有（所有字符串）

应用场景:转发路由过程中,携带的信息非常具体,可以将信息分类处理

jt.selete.item.unback（具体的路由key，消息携带的）

jt.#:这个队列的路由key表示当前队列接收与jt项目有关的所有消息

jt.selete.#:表示与京淘项目的所有查询有关的消息

如何进行分类处理消息，需要根据系统需求，业务需求处

理。

分区 day06all 的第 7页

Rabbitmq的api简单模式

2018年9月17日 9:55

pom的依赖,需要依赖amqp

<dependency>

<groupId>org.springframework.amqp</groupId> <artifactId>spring-rabbit</artifactId>

<version>1.4.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot<groupId>

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

1 简单模式

生产者

package com.jt.amqp.rabbitmq;

import java.io.IOException;

import java.util.concurrent.TimeoutException; import org.junit.Test;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.Connection;

import com.rabbitmq.client.ConnectionFactory; public class SimpleTest {

//利用一个方法测试生产者代码如何编写逻辑

//无论是生产端还是消费端都需要连接rabbitmq @Test

分区 day06all 的第 8页

@Test

public void sender() throws Exception{

//获取连接获取信道

/\*1 创建连接工厂

\*2 从连接工厂获取connection

\*3 获取信道

\*4 绑定queue

\*5 发送消息

\*6 释放资源

\*/

//创建工厂

ConnectionFactory factory=new

ConnectionFactory();

//连接工厂配置参数

factory.setHost("106.75.120.140");

factory.setPort(5672);

factory.setUsername("jtadmin");

factory.setPassword("123456");

factory.setVirtualHost("/jt");

//获取连接

Connection conn = factory.newConnection();

//获取信道对象短连接

Channel channel = conn.createChannel();

//利用短连接信息对象来操作rabbitmq

String queue\_name="simple";

/\*

\* queue:String,即将生命创建的队列名称,可操作 范围内已有同名队列

分区 day06all 的第 9页

范围内已有同名队列

\* 就会直接绑定使用,没有就创建一个queue

\* durable: Boolean 表示当前队列是否具有持久 化的特性;

\* exclusive:Boolean 表示queue是否专属.当前连 接如果需要单独操作queue

\* 需要对queue进行专属的设定,true表示只有当

前创建的连接可以操作queue,false表示

\* 所有连接都可以操作队列

\* autoDelete:Boolean,是否自动删除,当前连接使 用完队列后是否删除

\* arguments:Map,表示队列的其他属性;

\* null表示使用默认的属性,包括消息超时,消息长 度,队列的访问优先级

\*/

channel.queueDeclare

(queue\_name, false, false, false, null);

//发送消息,利用channel发送,发送给默认交换

机,""

String msg="hello simple model";

/\*

\* exchange: String 交换机名称,当前简单模式,没 有交换机使用默认的

\* routingKey:String 队列名称

\* props:BasicProps 消息的属性内

容,BasicProps.deliveryMode值可以是

\* 0 持久化,1不持久化,配合durable存在

\* body:byte[] 消息的二进制

\*/

分区 day06all 的第 10页

\*/

channel.basicPublish("", queue\_name, null, msg.getBytes());

//释放资源

channel.close();

conn.close();}}

连接工具类

public static Channel getResource(){

try{

ConnectionFactory factory=new ConnectionFactory();

//连接工厂配置参数

factory.setHost("106.75.120.140"); factory.setPort(5672);

factory.setUsername("jtadmin"); factory.setPassword("123456");

factory.setVirtualHost("/jt");

//获取连接

Connection conn =

factory.newConnection();

//获取信道对象短连接

Channel channel = conn.createChannel(); return channel;

}catch(Exception e){

e.printStackTrace();

return null;

}

}

分区 day06all 的第 11页

}

消费者逻辑

@Test

public void receiver() throws Exception{

//拿到信道

Channel channel = ChannelUtil.getResource(); String queue\_name="simple";

channel.queueDeclare(queue\_name, false, false, false, null);

//创建一个消费者对象

QueueingConsumer consumer=new QueueingConsumer(channel);

//绑定消费者和队列

/\*

\* queue:队列名称

\* autoAck:ack(acknowledge 确认) Boolean是 否自动确认,当前消费逻辑到底是

\* 那完了就回执,消费逻辑还没运行.还是等待消费

逻辑执行完毕后手动回执;代码编写回执的方法

\*

channel.ack,channel.nack.channel.basicReject

\* callback:绑定的消费者对象

\*/

channel.basicConsume(queue\_name, true, consumer);

//接收消息,编写一个监听逻辑,在测试当中使用 while死循环

分区 day06all 的第 12页

while死循环

while(true){

//获取一个下个消息的携带者,delivery Delivery delivery =

consumer.nextDelivery();

String msg=new

String(delivery.getBody());//byte数组

System.out.println("成功消费消息:"+msg);

}

}

分区 day06all 的第 13页

工作模式(争抢资源)

2018年9月17日 11:29

package com.jt.amqp.rabbitmq;

import java.io.IOException;

import org.junit.Test;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.QueueingConsumer;

import

com.rabbitmq.client.QueueingConsumer.Delivery;

public class WorkTest {

private static final String queue\_name="work";

//生产端逻辑

@Test

public void sender() throws Exception{

Channel channel = ChannelUtil.getResource(); channel.queueDeclare

(queue\_name, false, false, false, null); String msg="hello work model";

//利用for循环模拟海量消息的发送

for(int i=0;i<100;i++){

String msgDelive=msg+i;

channel.basicPublish("", queue\_name, null, msgDelive.getBytes());

System.out.println("生产者发送消息"+i+"条");

}

channel.close();

分区 day06all 的第 14页

channel.close();

}

//消费逻辑,模拟资源处理速度不一样,添加模拟消费时间的 代码Thread.sleep();

@Test

public void rec01() throws Exception{

Channel channel = ChannelUtil.getResource();

channel.queueDeclare(queue\_name, false, false, false, null);

//限制当前消费者的获取消息的数量,表示在回执之前 只能消费接收一条消息

channel.basicQos(1);

//创建一个消费者对象

QueueingConsumer consumer=new

QueueingConsumer(channel);

channel.basicConsume(queue\_name, false,

consumer);

//为了添加消费时间的逻辑有效,必须手动回执; while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者01接收消息:"+msg);

//没有手动添加回执的情况下,队列认为你没有消 费完,下一条消息不发送

Thread.sleep(50);

channel.basicAck

(delivery.getEnvelope().getDeliveryTag(), false);

分区 day06all 的第 15页

false);

}

}

@Test

public void rec02() throws Exception{

Channel channel = ChannelUtil.getResource();

channel.queueDeclare(queue\_name, false, false, false, null);

//限制当前消费者的获取消息的数量,表示在回执之前 只能消费接收一条消息

channel.basicQos(1);

//创建一个消费者对象

QueueingConsumer consumer=new

QueueingConsumer(channel);

channel.basicConsume(queue\_name, false, consumer);

//为了添加消费时间的逻辑有效,必须手动回执;

while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者02接收消息:"+msg);

//没有手动添加回执的情况下,队列认为你没有消 费完,下一条消息不发送

Thread.sleep(10);

channel.basicAck

(delivery.getEnvelope().getDeliveryTag(), false);

分区 day06all 的第 16页

false);

}

}

}

分区 day06all 的第 17页

发布订阅模式

2018年9月17日 11:37

package com.jt.amqp.rabbitmq;

import java.io.IOException;

import org.junit.Test;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.QueueingConsumer;

import com.rabbitmq.client.QueueingConsumer.Delivery;

public class PublishTest {

private static final String exchange\_name="publish1805"; private static final String queue01="pqueue01";

private static final String queue02="pqueue02";

//生产端

@Test

public void sender() throws Exception{

//自定义交换机名称

Channel channel = ChannelUtil.getResource();

//声明交换机,生产者只需要与交换机互动,声明队列可以不 再生产端完成

//交换机类型4种:fanout发布订阅,direct路由,topic主题模 式,header rpc

channel.exchangeDeclare(exchange\_name, "fanout");

for(int i=0;i<100;i++){

String msg="hello publish mode"+i;

分区 day06all 的第 18页

String msg="hello publish mode"+i;

//发布订阅的路由key,""

channel.basicPublish(exchange\_name, "", null, msg.getBytes());

}

}

@Test

public void rec01() throws Exception{

Channel channel = ChannelUtil.getResource();

//声明队列

channel.queueDeclare(queue01,false,false,false,null);

//声明交换机

channel.exchangeDeclare(exchange\_name, "fanout");

//绑定关系,交换机和队列也需要绑定

channel.queueBind(queue01, exchange\_name, "");

//消费者

QueueingConsumer consumer=new

QueueingConsumer(channel);

//绑定消费者队列

channel.basicConsume(queue01, true,consumer);

//while 接收消费消息

while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者01接收消息:"+msg);

}

}

@Test

分区 day06all 的第 19页

@Test

public void rec02() throws Exception{

Channel channel = ChannelUtil.getResource();

//声明队列

channel.queueDeclare(queue02,false,false,false,null);

//声明交换机

channel.exchangeDeclare(exchange\_name, "fanout");

//绑定关系,交换机和队列也需要绑定

channel.queueBind(queue02, exchange\_name, "");

//消费者

QueueingConsumer consumer=new

QueueingConsumer(channel);

//绑定消费者队列

channel.basicConsume(queue02, true,consumer);

//while 接收消费消息

while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者02接收消息:"+msg);

}

}

}

分区 day06all 的第 20页

分区 day06all 的第 21页

路由模式和主题模式

2018年9月17日 11:49

package com.jt.amqp.rabbitmq;

import org.junit.Test;

import com.rabbitmq.client.Channel;

import com.rabbitmq.client.QueueingConsumer;

import com.rabbitmq.client.QueueingConsumer.Delivery; public class RoutingTest {

private static final String exchange\_name="direct1805"; private static final String queue01="dqueue01";

private static final String queue02="dqueue02";

//生产端

@Test

public void sender() throws Exception{

//自定义交换机名称

Channel channel = ChannelUtil.getResource();

//声明交换机,生产者只需要与交换机互动,声明队列可以不 再生产端完成

//交换机类型4种:fanout发布订阅,direct路由,topic主题模 式,header rpc

channel.exchangeDeclare(exchange\_name, "direct");

String msg="我要新增一条商品";

channel.basicPublish(exchange\_name, "item.add", null, msg.getBytes());

}

分区 day06all 的第 22页

@Test

public void rec01() throws Exception{

Channel channel = ChannelUtil.getResource();

//声明队列

channel.queueDeclare(queue01,false,false,false,null);

//声明交换机

channel.exchangeDeclare(exchange\_name, "direct");

//绑定关系,交换机和队列也需要绑定

channel.queueBind(queue01, exchange\_name, "item.update");

//消费者

QueueingConsumer consumer=new

QueueingConsumer(channel);

//绑定消费者队列

channel.basicConsume(queue01, true,consumer);

//while 接收消费消息

while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者01接收消息:"+msg);

}

}

@Test

public void rec02() throws Exception{

Channel channel = ChannelUtil.getResource();

//声明队列

channel.queueDeclare(queue02,false,false,false,null);

分区 day06all 的第 23页

channel.queueDeclare(queue02,false,false,false,null);

//声明交换机

channel.exchangeDeclare(exchange\_name, "direct");

//绑定关系,交换机和队列也需要绑定

channel.queueBind(queue02, exchange\_name, "item.add");

//消费者

QueueingConsumer consumer=new

QueueingConsumer(channel);

//绑定消费者队列

channel.basicConsume(queue02, true,consumer);

//while 接收消费消息

while(true){

Delivery delivery = consumer.nextDelivery(); String msg=new String(delivery.getBody());

System.out.println("消费者02接收消息:"+msg);

}

}

}

作业:主题模式课下测试

分区 day06all 的第 24页

springboot整合rabbitmq

2018年9月17日 13:59

springboot收集连接信息之后,帮你完成了对象的创建.只需要注入模板对象即可操作rabbitmq(整合时

依赖的jar包starter-amqp)

1 配置application.properties

spring.rabbitmq.host=106.75.120.140 spring.rabbitmq.port=5672

spring.rabbitmq.username=jt

spring.rabbitmq.password=123456 spring.rabbitmq.virtualHost=/jt

框架创建管理一个rabbitmqTemplate的模板对象,底 层都是获取连接,生成信道对象操作的.可以完成

amqp的一些手动对象的封装来实现交换机的创建,

队列的创建

1. 利用注解Bean和方法来创建我们需要的内容(交换机和队列)--路由模式为例

在一个Configuration的类上添加队列情况的配置; package com.jt.config;

import

org.springframework.amqp.core.Binding; import

org.springframework.amqp.core.BindingBuilder

;

import

org.springframework.amqp.core.DirectExchang

分区 day06all 的第 25页

org.springframework.amqp.core.DirectExchang e;

import org.springframework.amqp.core.Queue; import

org.springframework.context.annotation.Bean;

import

org.springframework.context.annotation.Config uration;

@Configuration

public class RabbitmqConfiguration {

//根据需要创建交换机,创建队列,一个交换机,2 个队列

@Bean

public Queue getQueue01(){

return new Queue("itemUpdateQueue");

}

@Bean

public Queue getQueue02(){

return new Queue("itemAddQueue");

}

//声明交换机,路由模式

@Bean

public DirectExchange getExchange(){

return new

DirectExchange("1805directSpringboot")

;

}

分区 day06all 的第 26页

}

//单独做绑定

@Bean

public Binding binding01(){

return

BindingBuilder.bind(getQueue01()).

to(getExchange()).with("item.upd ate");

}

@Bean

public Binding binding02(){

return

BindingBuilder.bind(getQueue02()).

to(getExchange()).with("item.add ");}}

1. 编写生产者代码

生产者发送消息,可以注入一个rabbitTemplate的 对象,利用这个对象将消息发送给交换机;

@Autowired

private RabbitTemplate rabbitmq; @RequestMapping("send/{msg}") @ResponseBody

public String sendMsg(@PathVariable String msg){

//接收的消息参数就会被绑定item.add

的路由key,发送到指定交换机

rabbitmq.convertAndSend("1805dire ctSpringboot", "item.add", msg);

分区 day06all 的第 27页

ctSpringboot", "item.add", msg); return "success";

}

1. 消费者代码

利用注解底层实现异步监听

@RabbitListener 绑定监听队列

@Component

public class RabbitMqReceiver {

//消费逻辑的方法,其中的参数就是我们传递 的消息

@RabbitListener(queues="itemAddQue ue")

public void process01(String msg){

System.out.println("增加的消费端收到 消息:"+msg);

}

@RabbitListener(queues="itemUpdateQ ueue")

public void process02(String msg){

System.out.println("更新的消费端收到 消息:"+msg);}}

整合的过程中,框架管理维护了大部分底层各种连接

绑定的关系内容,开发人员只需要根据业务逻辑配置rabbitmq中队列,交换机等的结构,类型,把大量精力放

到业务执行上;如果我们需要对template的对象进行 二次封装,也可以完成生产,消费的逻辑;

分区 day06all 的第 28页

二次封装,也可以完成生产,消费的逻辑;

消费端不止可以在与生产端同工程的代码编写,还可 以在任何一个连接相同rabbitmq的工程中完成监听;

分区 day06all 的第 29页

分区 day06all 的第 34页