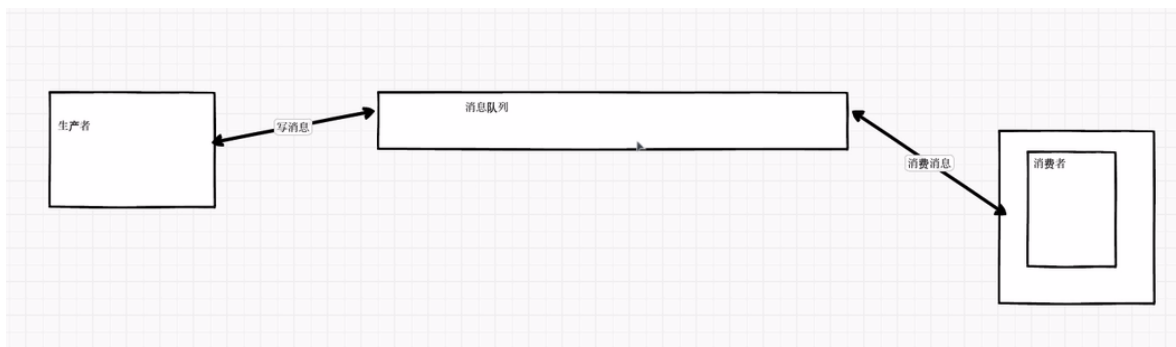


RabbitMQ实战教程

1.MQ引言

1.1 什么是MQ

MQ (Message Queue) : 消息队列，又别名 消息中间件，通过利用高效率的信息传递机制进行平台的数据交流，并基于数据通信来进行分布式系统的集成



1.2 MQ有哪些

- ActiveMQ,
- RabbitMQ,
- kafka
- 阿里自主研发的 RocketMQ
- 阿里自主研发的 RocketMQ

2.RabbitMQ的引言

2.1 RabbitMQ

- [官网](#)
- [下载地址](#)

2.2 RabbitMQ的安装

- Ubuntu 安装[CSDN帮助博客](#)

2.2.1 安装步骤

```
#1、安装erlang
sudo apt-get install erlang-nox

#2、安装rabbitmq
sudo apt-get install rabbitmq-server
```

3.RabbitMQ管理

3.1 rabbitmq管理命令行

```
#服务启动相关
systemctl start | restart | stop rabbitmq-server

#管理命令行
rabbitmqctl help

#插件管理
rabbitmq-plugins enable | list | disable
```

3.2 状态管理

```
#3、rabbitmq状态管理

# 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 rabbitmq-server status
```

3.3 web界面管理

```
#4、rabbitmq既可以命令行操作，也可以用rabbitmq自带的web管理界面，只需要启动插件便可以使用
sudo rabbitmqctl start_app
sudo rabbitmq-plugins enable rabbitmq_management

#访问： http://127.0.0.1:15672

#开启超级管理员登录
sudo rabbitmqctl add_user admin admin

#赋予权限
sudo rabbitmqctl set_user_tags admin administrator
```

4.RabbitMQ程序

4.1 消息模型

[参考地址](#)

4.2 引入依赖

```
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>5.7.2</version>
</dependency>
```

4.3 直连模型-点对点



- P：生产者
- C：消费者
- queue：消息队列，图中红色部分

1.开发生产者

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import org.junit.Test;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

//生产者
public class Provider {
    @Test
    public void testSendMesg() throws IOException, TimeoutException {
        /*
         * 1、创建连接mq的连接工厂对象
         * 2、设置连接rabbitmq的主机
         * 3、设置端口号
         * 4、设置连接虚拟机
         * 5、设置访问虚拟机的用户名和密码
         * 6、获取连接对象
         */
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("127.0.0.1");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("/demo");
        connectionFactory.setUsername("demo");
        connectionFactory.setPassword("123456");
        Connection connection = connectionFactory.newConnection();

        /*
         * 1、获取连接中通道对象
         * 2、通道对象绑定对应消息队列
         * 3、发布消息
         * 4、关闭连接
         */
        Channel channel = connection.createChannel();
        /* @参数1： 队列名称
```

```

    * @参数2： 定义队列特性分割是需要持久化
    * @参数3： 是否独占队列
    * @参数4： 是否在消费完后自动删除队列
    * @参数5： 额外附加参数
    */
    channel.queueDeclare("hello",false,false,false,null);
    /*
    * 参数1： 交换机名称
    * 参数2： 队列名称
    * 参数3： 传递消息的额外设置
    * 参数4： 消息的具体内容
    */
    channel.basicPublish("", "hello", null, "hello rabbitmq".getBytes());
    channel.close();
    connection.close();
}
}

```

2.开发消费者

```

import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class Consumer {

    public static void main(String[] args) throws IOException, TimeoutException {
        /*
        * 1、创建连接工厂
        * 2、设置连接rabbitmq
        * 3、设置连接虚拟主机和端口号
        * 4、设置连接用户和密码
        * 5、创建连接对象
        */
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("127.0.0.1");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("/demo");
        connectionFactory.setUsername("demo");
        connectionFactory.setPassword("123456");
        Connection connection = connectionFactory.newConnection();
        /*
        * 1、创建通道
        * 2、通道绑定对象
        * 3、消费信息
        * 4、关闭连接
        */
        Channel channel = connection.createChannel();
        channel.queueDeclare("hello", false, false, false, null);

        /*
        * 参数1： 队列名称
        * 参数2： 开始消息的自动确认机制
        */
    }
}

```

```

* 参数3：消费时的回调接口
**/
channel.basicConsume("hello", true, new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
        System.out.println("new String(Body) = "+ new String(body));
    }
});
channel.close();
connection.close();
}
}

```

3.工具类封装

```

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class RabbitMQUtils {

    private static ConnectionFactory connectionFactory;
    static {
        // 重量级资源，类加载只执行一次
        connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("127.0.0.1");
        connectionFactory.setPort(5672);
        connectionFactory.setVirtualHost("/demo");
        connectionFactory.setUsername("demo");
        connectionFactory.setPassword("123456");
    }
    //定义提供连接对象
    public static Connection getConnection() {
        try {

            return connectionFactory.newConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    //关闭通道和关闭连接工具方法
    public static void closeConnectionAndChannel(Channel channel, Connection connection){
        try {
            if (channel != null) channel.close();
            if (connection != null) connection.close();
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

4.4.Work Queues工作队列

work queues 又称 task queues,任务模型,多个消费者绑定到一个队列,共同消费队列中的消息,队列中的消息一旦消耗,就会消失,因此任务不会重复执行



1.开发生产者

```
import Utils.RabbitMQUtils;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

import java.io.IOException;

public class Provider {
    public static void main(String[] args) throws IOException {
        Connection connection = RabbitMQUtils.getConnection();
        Channel channel = connection.createChannel();
        channel.queueDeclare("work", true, false, false, null);
        for (int i = 0; i < 10; i++) {
            channel.basicPublish("", "work", null, (i + ": hello workqueues").getBytes());
        }
        RabbitMQUtils.closeConnectionAndChannel(channel, connection);
    }
}
```

2.开发消费者

- 消费者1 和消费者2 代码类似

```
import Utils.RabbitMQUtils;
import com.rabbitmq.client.*;

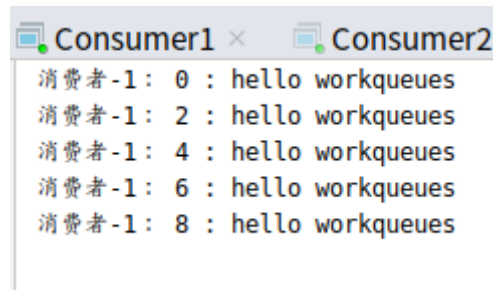
import java.io.IOException;

public class Consumer1 {
    public static void main(String[] args) throws IOException {
        Connection connection = RabbitMQUtils.getConnection();
        Channel channel = connection.createChannel();

        channel.queueDeclare("work", true, false, false, null);
        channel.basicConsume("work", true, new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
                System.out.println("消费者-1: " + new String(body));
            }
        });
    }
}
```

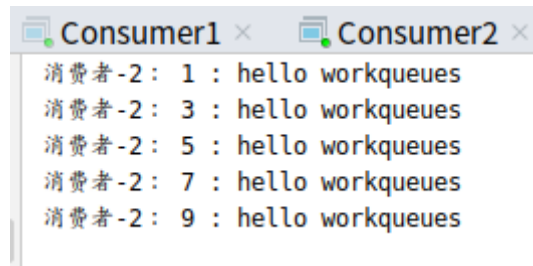
3.结果

- 消费者1：



```
Consumer1 x Consumer2
消费者-1: 0 : hello workqueues
消费者-1: 2 : hello workqueues
消费者-1: 4 : hello workqueues
消费者-1: 6 : hello workqueues
消费者-1: 8 : hello workqueues
```

- 消费者2：



```
Consumer1 x Consumer2 x
消费者-2: 1 : hello workqueues
消费者-2: 3 : hello workqueues
消费者-2: 5 : hello workqueues
消费者-2: 7 : hello workqueues
消费者-2: 9 : hello workqueues
```

- 总结：RabbitMQ将顺序将按顺序将每个消息发送到下一个试用者，每个消费者都会收到相同数量的消息，这种发布消息的方式称为轮询

4.消息确认机制

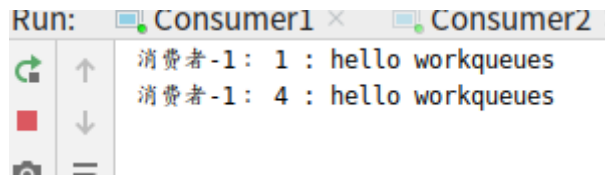
- 设置通道一次只能消费一个消息

```
channel.basicQos(1);//每次只能消费一个消息
```

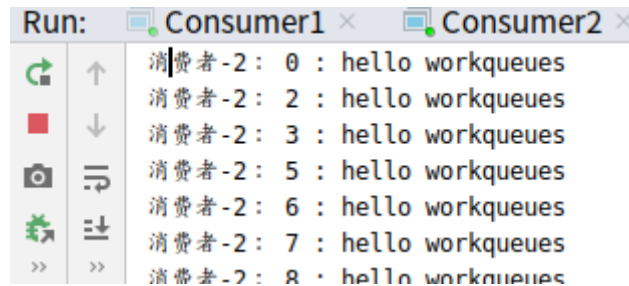
- 关闭消息的自动确认，开启手动确认

```
/*
 * 参数2： 消息自动确认 true,消费者自动向rabbitmq确认消息消费， false： 不会自动确认消息
 */
channel.basicConsume("work", false, new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
        System.out.println("消费者-2: " + new String(body));
        //手动确认
        /*
         * 参数1： 手动确认消息标识
         * 参数2： 是否开启多个消息确认， false 每次确认一个
         */
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
});
```

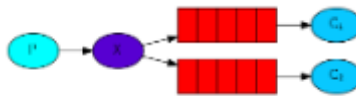
- 结果
 - 能者多劳
 - 消费者1（开启sleep线程，每1000ms执行一次）：



- 消费者 2 :



4.5 Fanout 广播



在广播模式下，

- 每个消费者有自己的队列
- 每个队列要绑定到交换机
- 生产者发送信息，只能发送到交换机，交换机决定发送给那哪个队列，生产者不能决定
- 交换机把消息发送给绑定过的所有的队列
- 队列的消费者都能拿到消息，实现一条消息被多个消费者消费

1.开发生产者

```
import Utils.RabbitMQUtils;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;

import java.io.IOException;

public class Provider {
    public static void main(String[] args) throws IOException {
        Connection connection = RabbitMQUtils.getConnection();
        Channel channel = connection.createChannel();
        /*
        * 将通道声明指定交换机
        * 参数1： 交换机名称
        * 参数2： 交换机类型*/
        channel.exchangeDeclare("fanoutdemo", "fanout");
        /*
        * 发送消息
        */
        for (int i = 0; i < 10; i++) {
            channel.basicPublish("fanoutdemo", "", null, (i + " : fanout type message").getBytes());
        }
        RabbitMQUtils.closeConnectionAndChannel(channel, connection);
    }
}
```


2.开发消费者

```
import Utils.RabbitMQUtils;
import com.rabbitmq.client.*;

import java.io.IOException;

public class Consumer1 {
    public static void main(String[] args) throws IOException {
        Connection connection = RabbitMQUtils.getConnection();
        Channel channel = connection.createChannel();

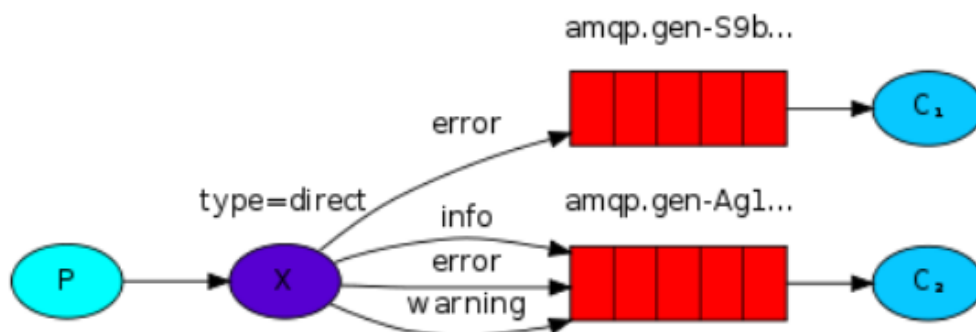
        //绑定交换机
        channel.exchangeDeclare("fanoutdemo", "fanout");
        //临时队列
        String queueName = channel.queueDeclare().getQueue();
        //绑定交换机进和队列
        channel.queueBind(queueName, "fanoutdemo", "");
        //消费信息
        channel.basicConsume(queueName, true, new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
                System.out.println("consumer1 : "+new String(body));
            }
        });
    }
}
```

4.6 Routing 路由

4.6.1 Routing之订阅模式-Direct（直连）

在 Direct 模型下：

- 队列于交换机的绑定不能是任意绑定，而是要指定一个 Routing key（路由key）
- 消息的发送放在向交换机发送消息时，也必须指定消息的 Routing key
- 交换机不能把消息交给每一个绑定的队列，而是根据消息的 Routing key 进行判断进行分发
- 流程



1.开发生产者

```
//通过通道声明交换机
channel.exchangeDeclare("directdemo", "direct");

//定义路由关键字
String routingkey = "error";

channel.basicPublish("directdemo", routingkey, null, "routing-direct msg".getBytes());
```

2.开发消费者

```
//基于routingkey绑定交换进和队列
//路由关键字： info
channel.queueBind(queueName, "directdemo", "info");
```

- 多个绑定

```
channel.queueBind(queueName, "directdemo", "error");
channel.queueBind(queueName, "directdemo", "info");
channel.queueBind(queueName, "directdemo", "warning");
```

4.6.2 Routing之订阅模式-Topic

Topic 模型的交换机与 Direct 相比，都可以根据路由关键字发送到不同的队列中，但是topic可以在路由关键字中使用通配符



1.通配符

#通配符

`*` : 匹配一个单词
`#` : 匹配零个或多个单词

例子

`audit.#` : 匹配`audit.irs.corporate`或者`audit.irs`等
`audit.*` : 只能匹配`audit.irs`

2.开发生产者

```
//通过通道声明交换机
channel.exchangeDeclare("topicdemo", "topic");

String routingkey = "user.save";

channel.basicPublish("topicdemo", routingkey, null, "routing-topic msg".getBytes());
```

3.开发消费者

```
//绑定交换机
channel.exchangeDeclare("topicdemo", "topic");
//临时队列
String queueName = channel.queueDeclare().getQueue();
//基于routingkey绑定交换机和队列
channel.queueBind(queueName, "topicdemo", "user.*");
```

5.SpringBoot整合RabbitMQ

5.1环境搭建

5.1.1 引入依赖

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

5.1.2 配置文件

```
spring:
  application:
    name: springboot_rabbitmq

rabbitmq:
  host: 127.0.0.1
  port: 5672
  username: demo
  password: 123456
  virtual-host: /demo
```

5.1.3 构建测试环境

```

@SpringBootTest(classes = SpringbootRabbitmqApplication.class)
@RunWith(SpringRunner.class)
public class TestRabbitMQ {
    //注入rabbitmqTemplate
    @Autowired
    private RabbitTemplate rabbitTemplate;
    //...
}

```

5.2 点对点模型

1.开发生产者

```

//...
@Test
public void testHello(){
    rabbitTemplate.convertAndSend("hello", "hello wrld");
}
//...

```

2.开发消费者

```

import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

/*
 * 定义队列特性
 * 默认：持久化 非独占 不自动删除
 * @Queue(...)*/
@Component
@RabbitListener(queuesToDeclare = @Queue("hello"))
public class HelloConsumer {
    @RabbitHandler
    public void recemsg(String msg){
        System.err.println("====="+msg);
    }
}

```

5.3 WorkQueue模型

1.开发生产者

```
//...
@Test
public void textWork(){
    for (int i = 0; i < 10; i++) {
        rabbitTemplate.convertAndSend("work", i + " : hello work");
    }
}
//...
```

2.开发消费者

```
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class WorkConsumer {

    /*
     * 第一个消费者*/
    @RabbitListener(queuesToDeclare = @Queue("work"))
    public void recemsg1(String msg){
        System.err.println("consumer1 ===== "+msg);
    }

    /*
     * 第二个消费者*/
    @RabbitListener(queuesToDeclare = @Queue("work"))
    public void recemsg2(String msg){
        System.err.println("consumer2 ===== "+msg);
    }
}
```

5.4 Fanout模型

1.开发生产者

```
@Test
public void testFanout(){
    rabbitTemplate.convertAndSend("fanout", "", "fanout模型");
}
```

2.开发消费者

```
import org.springframework.amqp.rabbit.annotation.Exchange;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.QueueBinding;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;
```

```

@Component
public class FanoutConsumer {
    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "fanout",type = "fanout")
        )
    })
    public void recemsg1(String msg){
        System.err.println("comsumer1 ===== "+msg);
    }

    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "fanout",type = "fanout")
        )
    })
    public void recemsg2(String msg){
        System.err.println("comsumer1 ===== "+msg);
    }
}

```

5.5 Routing-Direct模型

1.开发生产者

```

@Test
public void testRouteDirect(){
    rabbitTemplate.convertAndSend("routedirect", "error", "route-direct模型的error信息");
}

```

2.开发消费者

```

package com.hdl.route_dirrect;

import org.springframework.amqp.rabbit.annotation.Exchange;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.QueueBinding;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class Route_directConsumer {
    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "routedirect",type = "direct"),
            key = {"info","error","warn"}
        )
    })
    public void recemsg1(String msg){

```

```

        System.err.println("comsumer1 ===== "+msg);
    }

    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "routedirect",type = "direct"),
            key = {"error"},
        )
    })
    public void recemsg2(String msg){
        System.err.println("comsumer2 ===== "+msg);
    }
}

```

5.6 Routing-Topic模型

1.开发生产者

```

@Test
public void testRouteTopic(){
    rabbitTemplate.convertAndSend("routetopic", "user.save.info","route-topic模型的user.save.info信息");
}

```

2.开发消费者

```

import org.springframework.amqp.rabbit.annotation.Exchange;
import org.springframework.amqp.rabbit.annotation.Queue;
import org.springframework.amqp.rabbit.annotation.QueueBinding;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class RouteDirectConsumer {
    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "routedirect",type = "direct"),
            key = {"info","error","warn"}
        )
    })
    public void recemsg1(String msg){
        System.err.println("comsumer1 ===== "+msg);
    }

    @RabbitListener(bindings = {
        @QueueBinding(
            value = @Queue,//创建临时队列
            exchange = @Exchange(value = "routedirect",type = "direct"),

```

```

        key = {"error",}
    )
}
}

public void recemsg2(String msg){
    System.err.println("comsumer2 ===== "+msg);
}
}

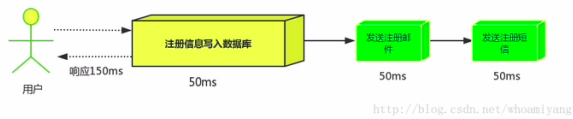
```

6.MQ应用场景

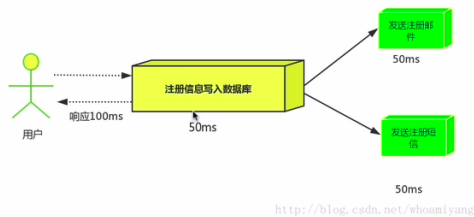
6.1异步处理

场景说明：用户注册后，需要发送注册邮件和注册短信，传统的做法有两种 1.串行的方式 2.并行的方式

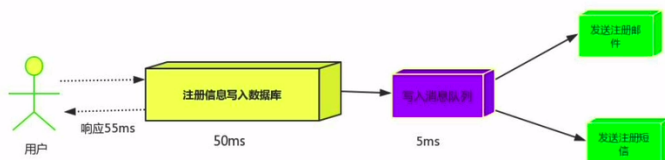
- **串行方式**：将注册信息写入数据库后,发送注册邮件,再发送注册短信,以上三个任务全部完成后才返回给客户端。这有一个问题是,邮件,短信并不是必须的,它只是一个通知,而这种做法让客户端等待没有必要等待的东西。



- **并行方式**：将注册信息写入数据库后,发送邮件的同时,发送短信,以上三个任务完成后,返回给客户端,并行的方式能提高处理的时间。



- **消息队列**：假设三个业务节点分别使用50ms,串行方式使用时间150ms,并行使用时间100ms。虽然并行已经提高的处理时间,但是,前面说过,邮件和短信对我正常的使用网站没有任何影响，客户端没有必要等着其发送完成才显示注册成功,应该是写入数据库后就返回。消息队列：引入消息队列后，把发送邮件,短信不是必须的业务逻辑异步处理



由此可以看出,引入消息队列后，用户的响应时间就等于写入数据库的时间+写入消息队列的时间(可以忽略不计),引入消息队列后处理后,响应时间是串行的3倍,是并行的2倍。

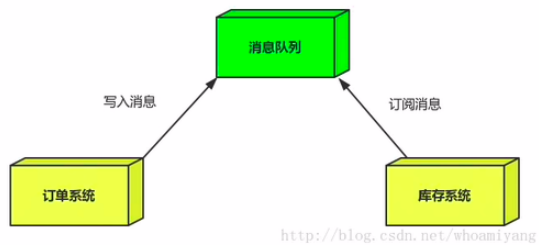
6.2应用解耦

场景：双11是购物狂节,用户下单后,订单系统需要通知库存系统,传统的做法就是订单系统调用库存系统的接口。



这种做法有一个缺点:

当库存系统出现故障时,订单就会失败。订单系统和库存系统高耦合. 引入消息队列



- **订单系统**: 用户下单后,订单系统完成持久化处理,将消息写入消息队列,返回用户订单下单成功。
- **库存系统**: 订阅下单的消息,获取下单消息,进行库操作。 就算库存系统出现故障,消息队列也能保证消息的可靠投递,不会导致消息丢失。

6.3流量削峰

- 场景：秒杀活动，一般因为流量过大，导致应用挂掉，为了解决这个问题，一般在应用端加入消息队列
- 作用
 1. 可以控制活动人口数量，超过一定阈值的订单直接丢弃
 2. 可以缓解短时间的高流量压垮应用（应用程序按自己的最大处理能力获取订单）



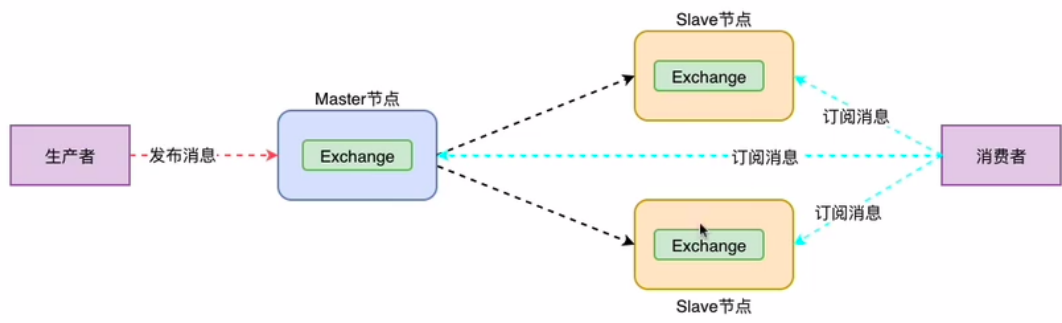
1. 用户请求，服务器受到之后，首先写入消息队列，加入消息队列长度超过最大值，则直接抛弃或跳转到错误页面
2. 秒杀业务根据消息队列中的请求信息，再做后续处理

7.RabbitMQ的集群

7.1集群架构

7.1.1普通集群（副本集群）

1. 架构图（主从复制）



核心解决问题：当集群中某一时刻master节点宕机，可以对Queue中信息进行备份

2. 集群搭建

1.克隆三台机器主机名和ip映射

```
vim /etc/hosts加入:
10.15.0.3 mq1
10.15.0.4 mq2
10.15.0.5 mq3

node1: vim /etc/hostname 加入: mq1
node2: vim /etc/hostname 加入: mq2
node3: vim /etc/hostname 加入: mq3
```

2.三个机器安装rabbitmq,并同步cookie文件,在node1上执行:

```
scp /var/lib/rabbitmq/.erlang.cookie root@mq2:/var/lib/rabbitmq/
scp /var/lib/rabbitmq/.erlang.cookie root@mq3:/var/lib/rabbitmq/
```

3.查看cookie是否一致:

```
node1: cat /var/lib/rabbitmq/.erlang.cookie
node2: cat /var/lib/rabbitmq/.erlang.cookie
node3: cat /var/lib/rabbitmq/.erlang.cookie
```

4.后台启动rabbitmq所有节点执行如下命令,启动成功访问管理界面:

```
rabbitmq-server -detached
```

5.在node2和node3执行加入集群命令:

```
1.关闭      rabbitmqctl stop_app
2.加入集群  rabbitmqctl join_cluster rabbit@mq1
3.启动服务  rabbitmqctl start_app
```

6.查看集群状态,任意节点执行:

```
rabbitmqctl cluster_status
```

7.如果出现如下显示,集群搭建成功:

```
Cluster status of node rabbit@mq3 ...
[{nodes,[{disc,[rabbit@mq1,rabbit@mq2,rabbit@mq3]}]},
{running_nodes,[rabbit@mq1,rabbit@mq2,rabbit@mq3]},
{cluster_name,<<"rabbit@mq1">>},
{partitions,[]},
{alarms,[{rabbit@mq1,[]},{rabbit@mq2,[]},{rabbit@mq3,[]}]}]
```

8.登录管理界面,展示如下状态:

Overview

Connections

Channels

Exchanges

Queues

Admin

Global counts

Connections: 1

Channels: 1

Exchanges: 7

Queues: 1

Consumers: 1

▼ Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit@mq1	38 1024 available	0 829 available	396 1048576 available	79MiB 390MiB high watermark	16GiB 48MiB low watermark	32m 14s	basic disc 1 rss	This node All nodes	
rabbit@mq2	36 1024 available	1 829 available	405 1048576 available	77MiB 390MiB high watermark	16GiB 48MiB low watermark	31m 35s	basic disc 1 rss	This node All nodes	
rabbit@mq3	37 1024 available	0 829 available	394 1048576 available	77MiB 390MiB high watermark	16GiB 48MiB low watermark	30m 56s	basic disc 1 rss	This node All nodes	

7.1.2镜像集群

1. 架构图

