6-1_rabbitmq案例_超时支付自动取消

一、题目要求

基于RabbitMQ的TTL以及死信队列，使用SpringBoot实现延迟付款，手动补偿操作。

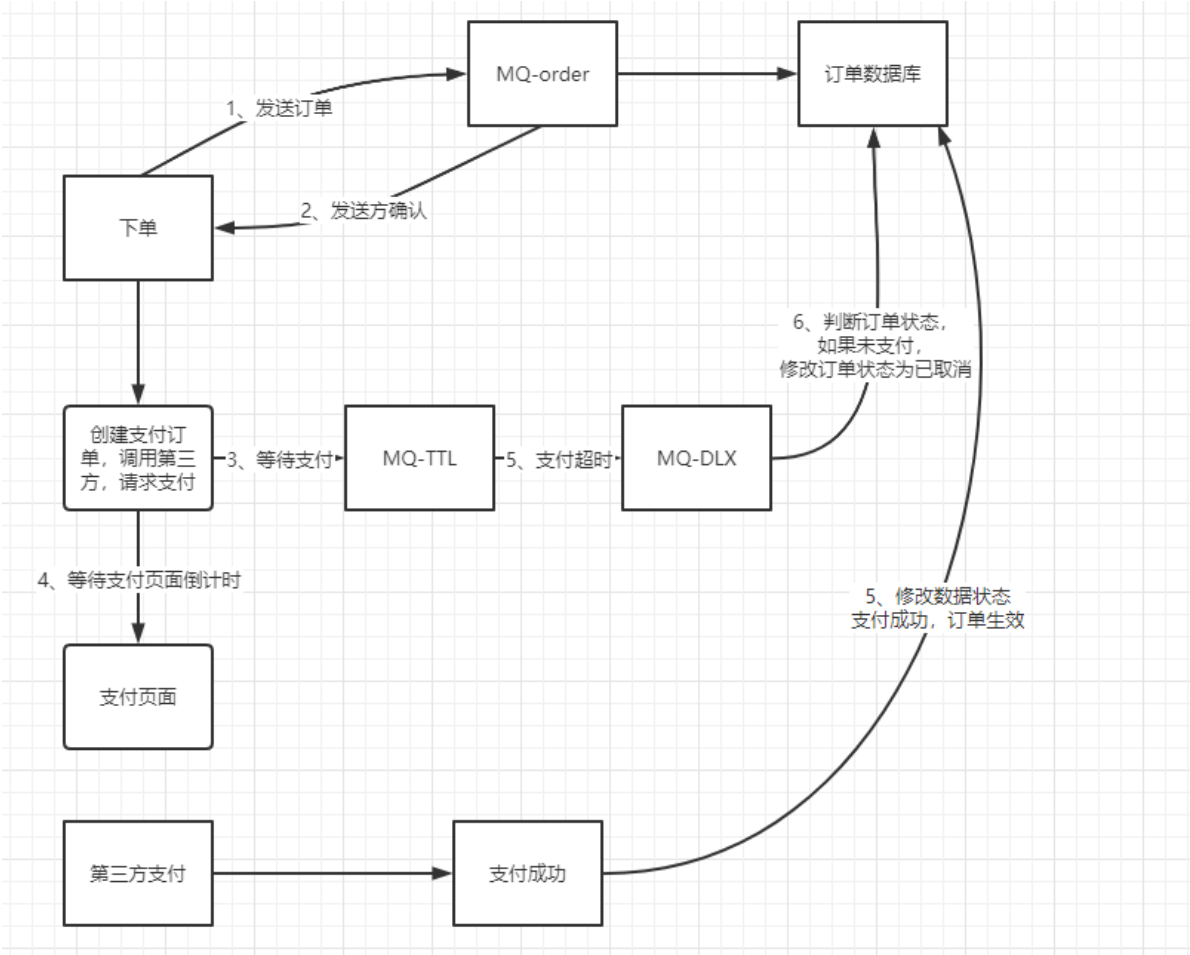
1、用户下单后展示等待付款页面

2、在页面上点击付款的按钮，如果不超时，则跳转到付款成功页面

3、如果超时，则跳转到用户历史账单中查看因付款超时而取消的订单。

二、思路分析

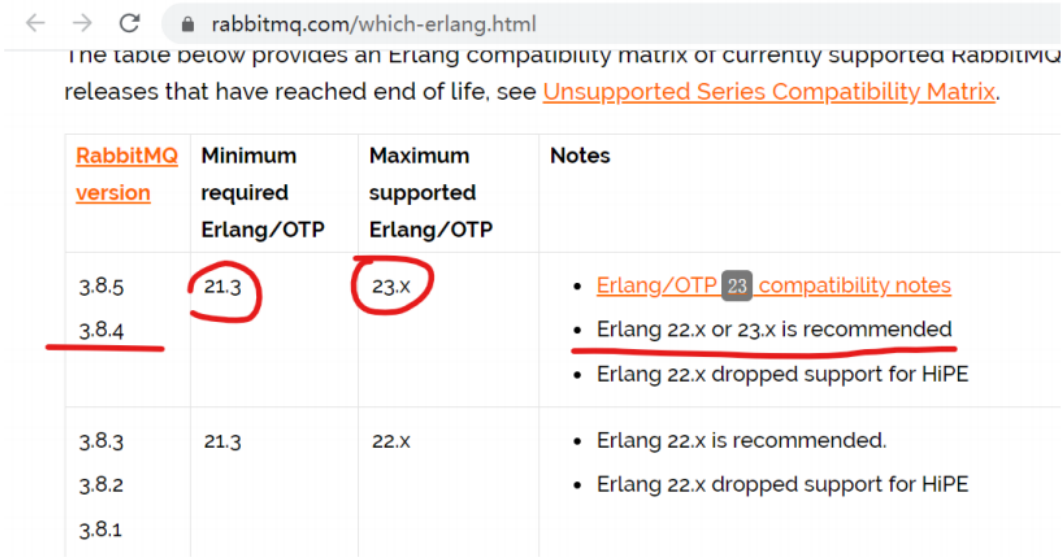
2.1 架构设计



2.2 前置准备

安装rabbitmq

- 安装环境：
    - 1. 虚拟机软件：VMWare 15.1.0
    - 2. 操作系统：CentOS Linux release 7.7.1908
    - 3. Erlang：erlang-23.0.2-1.el7.x86_64
    - 4. RabbitMQ：rabbitmq-server-3.8.4-1.el7.noarch

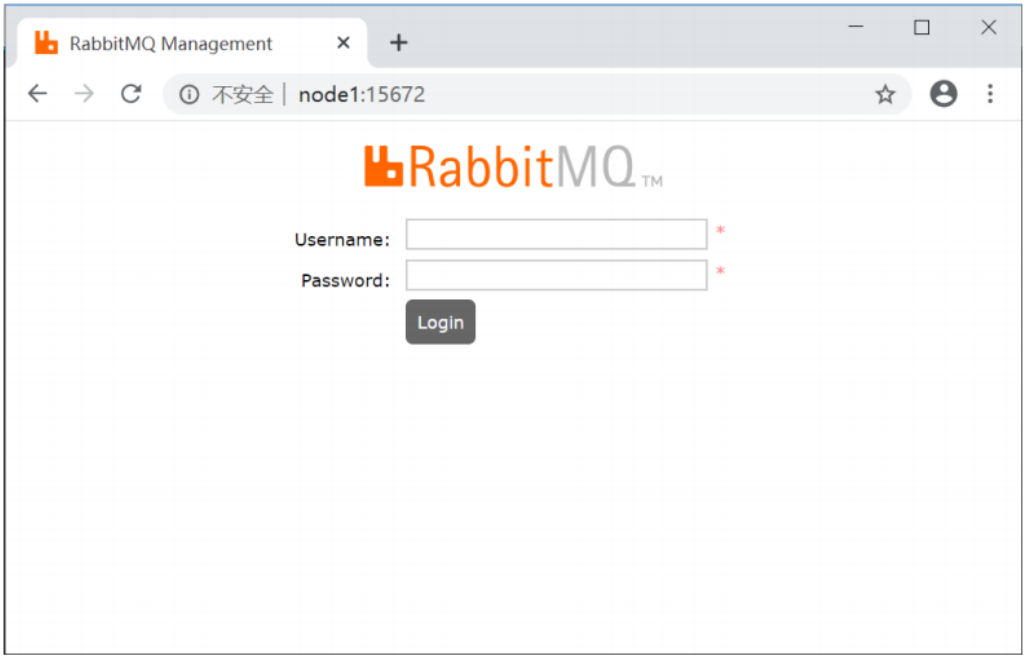
- RabbitMQ的安装需要首先安装Erlang，因为它是基于Erlang的VM运行的。
- RabbitMQ需要的依赖：socat和logrotate，logrotate操作系统中已经存在了，只需要安装socat就可以了。
- RabbitMQ与Erlang的兼容关系详见：https://www.rabbitmq.com/which-erlang.html
  - 

- 1、安装依赖：
  - yum install socat -y
- 2、安装Erlang
  - erlang-23.0.2-1.el7.x86_64.rpm下载地址：
    - https://github.com/rabbitmq/erlang-rpm/releases/download/v23.0.2/erlang-23.0.2-1.el7.x86_64.rpm
  - 首先将erlang-23.0.2-1.el7.x86_64.rpm上传至服务器，然后执行下述命令：
    - rpm -ivh erlang-23.0.2-1.el7.x86_64.rpm
- 3、安装RabbitMQ
  - rabbitmq-server-3.8.4-1.el7.noarch.rpm下载地址：
  - https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.8.4/rabbitmq-server-3.8.4-1.el7.noarch.rpm
  - 首先将rabbitmq-server-3.8.4-1.el7.noarch.rpm上传至服务器，然后执行下述命令：
  - rpm -ivh rabbitmq-server-3.8.4-1.el7.noarch.rpm
- 4、启用RabbitMQ的管理插件
  - rabbitmq-plugins enable rabbitmq_management
- 5、开启RabbitMQ
  - systemctl start rabbitmq-server
  - 或者
  - rabbitmq-server
  - 或者后台启动
  - rabbitmq-server -detached
- 6、添加用户
  - rabbitmqctl add_user root 123456
- 7、给用户添加权限
  - 给root用户在虚拟主机"/"上的配置、写、读的权限
  - rabbitmqctl set_permissions root -p / "." "." ".*"
- 8、给用户设置标签
  - rabbitmqctl set_user_tags root administrator

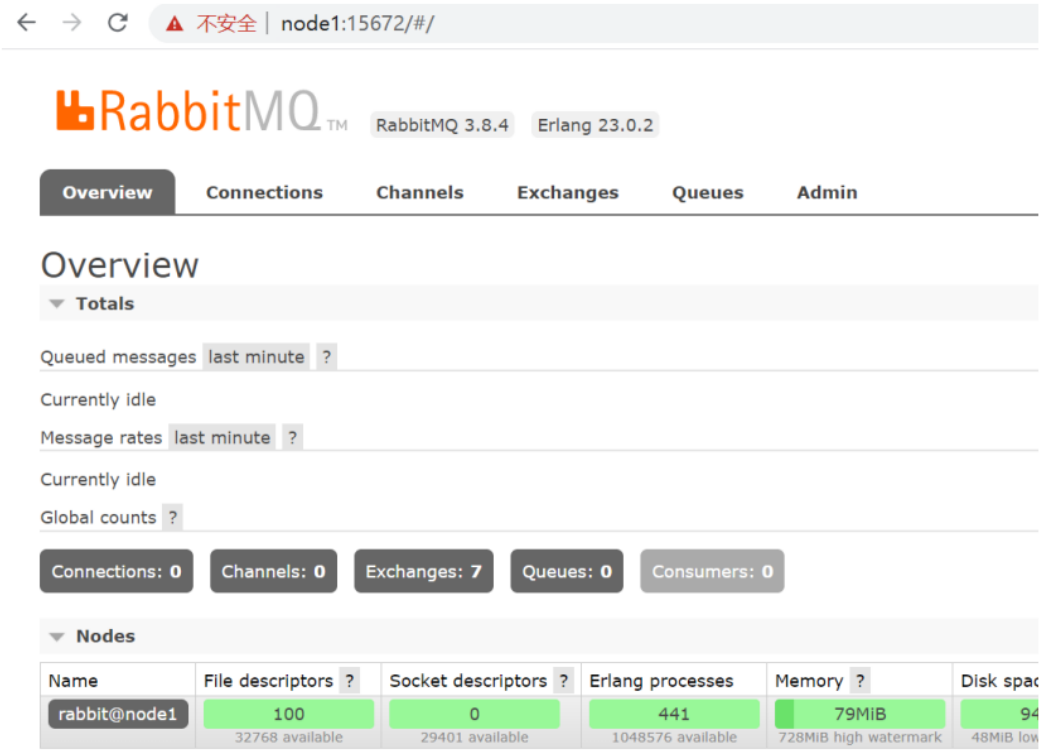
- 用户的标签和权限：

Tag	Capabilities
(None)	没有访问management插件的权限
management	可以使用消息协议做任何操作的权限，加上： 1. 可以使用AMQP协议登录的虚拟主机的权限 2. 查看它们能登录的所有虚拟主机中所有队列、交换器和绑定的权限 3. 查看和关闭它们自己的通道和连接的权限 4. 查看它们能访问的虚拟主机中的全局统计信息，包括其他用户的活动
policymaker	所有management标签可以做的，加上： 1. 在它们能通过AMQP协议登录的虚拟主机上，查看、创建和删除策略以及虚拟主机参数的权限
monitoring	所有management能做的，加上： 1. 列出所有的虚拟主机，包括列出不能使用消息协议访问的虚拟主机的权限 2. 查看其他用户连接和通道的权限 3. 查看节点级别的数据如内存使用和集群的权限 4. 查看真正的全局所有虚拟主机统计数据的权限
administrator	所有policymaker和monitoring能做的，加上： 1. 创建删除虚拟主机的权限 2. 查看、创建和删除用户的权限 3. 查看、创建和删除权限的权限 4. 关闭其他用户连接的权限

- 9、打开浏览器，访问http://<安装了CentOS的VMWare虚拟机IP地址>:15672



- 10、使用刚才创建的用户登录：



2.3 使用延迟队列

在AMQP协议和RabbitMQ中都没有相关的规定和实现。不过，我们似乎可以借助rabbitmq中的“死信队列”来变相的实现。

可以使用rabbitmq_delayed_message_exchange插件实现。

需要在虚拟机运行如下命令

```
rabbitmq-plugins enable rabbitmq_management
```

2.4 源码分析

pom.xml

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
```

```
            <groupId>org.springframework.amqp</groupId>
            <artifactId>spring-rabbit-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--lombok工具-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.4</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

配置文件application.properties

```
spring.application.name=payDemo
spring.rabbitmq.host=192.168.31.204
spring.rabbitmq.virtual-host=test
spring.rabbitmq.username=root
spring.rabbitmq.password=123456
spring.rabbitmq.port=5672

# 设置收到确认消息
spring.rabbitmq.publisher-confirm-type=correlated
spring.rabbitmq.publisher-returns=true
spring.rabbitmq.listener.direct.acknowledge-mode=manual

spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.check-template-location=true
spring.thymeleaf.suffix=.html
```

此处用的virtual-host是test， 也可用默认的spring.rabbitmq.virtual-host=/

配置类RabbitMqConfig.java

```
@Configuration
@EnableRabbit
@ComponentScan("com.idstaa")
public class RabbitMqConfig {
    @Bean
    /**
     * 订单消息队列
     */
    public Queue orderQueue() {
        return QueueBuilder.durable("q.order").build();
    }

    @Bean
    /**
     * 订单消息队列
     */
    public Queue ttlQueue() {
        Map<String, Object> args = new HashMap<>();
        args.put("x-message-ttl", 10000);
        args.put("x-dead-letter-exchange", "ex.dlx");
        args.put("x-dead-letter-routing-key", "key.dlx");
```

```
        return new Queue("ttl.order", true, false, false, args);
    }

    /**
     * 死信队列，用于取消用户订单
     */
    @Bean
    public Queue dlxQueue() {
        Map<String, Object> args = new HashMap<>();
        return new Queue("q.dlx", true, false, false, args);
    }

    /**
     * 订单交换器
     */
    @Bean
    public Exchange orderExchange() {
        Map<String, Object> args = new HashMap<>();
        DirectExchange exchange = new DirectExchange("ex.order",
                true,
                false,
                args);
        return exchange;
    }

    /**
     * ttl交换器
     */
    @Bean
    public Exchange ttlExchange() {
        Map<String, Object> args = new HashMap<>();
        DirectExchange exchange = new DirectExchange("ex.ttl",
                true,
                false,
                args);
        return exchange;
    }

    /**
     * 订单交换器
     */
    @Bean
    public Exchange dlxExchange() {
        Map<String, Object> args = new HashMap<>();
        DirectExchange exchange = new DirectExchange("ex.dlx",
                true,
                false,
                args);
        return exchange;
    }

    /**
     * 用于发送下单，做分布式事务的MQ
     */
    @Bean
    public Binding orderBinding() {
```

```
        return
BindingBuilder.bind(orderQueue()).to(orderExchange()).with("key.order").noargs()
;
    }

    /**
     * 用于等待用户支付的延迟队列绑定
     */
    @Bean
    public Binding ttlBinding() {
        return
BindingBuilder.bind(ttlQueue()).to(ttlExchange()).with("key.ttl").noargs();
    }


    /**
     * 用于支付超时取消用户订单的死信队列绑定
     */
    @Bean
    public Binding dlxBinding() {
        return
BindingBuilder.bind(dlxQueue()).to(dlxExchange()).with("key.dlx").noargs();
    }

    @Bean
    public RabbitAdmin rabbitAdmin(ConnectionFactory connectionFactory){
        return new RabbitAdmin(connectionFactory);
    }

   @Bean(name="rabbitMessageListenerContainer")
    public DirectMessageListenerContainer listenerContainer(ConnectionFactory
connectionFactory){
        DirectMessageListenerContainer container = new
DirectMessageListenerContainer(connectionFactory);
        container.setAcknowledgeMode(AcknowledgeMode.MANUAL);
        container.setPrefetchCount(5);
        container.setConsumersPerQueue(5);
        container.setMessagesPerAck(1);

       ThreadPoolTaskExecutor taskExecutor = new ThreadPoolTaskExecutor();
       taskExecutor.setCorePoolSize(10);
       taskExecutor.setMaxPoolSize(20);
       // 设置改属性，灵活设置并发
       container.setTaskExecutor(taskExecutor);
       return container;
   }

   @Bean
    public MessageConverter messageConverter(){
        return new Jackson2JsonMessageConverter();
   }


}
```

订单控制器类OrderController.java

```
@Controller
public class OrderController {
    @Autowired
    private RabbitTemplate rabbitTemplate;

    @RequestMapping("/createOrder")
    public String createOrder(Model model) throws ExecutionException,
InterruptedException {
        Order order = new Order();
        order.setOrderId(UUID.randomUUID().toString().substring(0,10));
        order.setStatus("待支付");
        order.setUserId("jianyi");
        OrderDetail detail = new OrderDetail();
        detail.setItemId(UUID.randomUUID().toString().substring(0,5));
        detail.setItemName("");
        detail.setItemPrice(100d);
        detail.setNum(2);
        ArrayList detailList = new ArrayList();
        detailList.add(detail);
        order.setDetail(detailList);
        CorrelationData correlationData = new CorrelationData();
        rabbitTemplate.convertAndSend("ex.order",
                "key.order",
                order,
                correlationData);
        CorrelationData.Confirm confirm = correlationData.getFuture().get();
        boolean ack = confirm.isAck();
        if(!ack){
            return "failOrder";
        }
        System.out.println("发送延迟取消信息，10s不支付就取消"+",当前时间"+ new
SimpleDateFormat("yyyy-MM-dd hh:mm:ss").format(new Date()));
        rabbitTemplate.convertAndSend("ex.ttl","key.ttl",order.getOrderId());
        model.addAttribute("orderId",order.getOrderId());
        return "order";
    }

    @RequestMapping("/failOrder/{orderId}")
    public String failOrder(@PathVariable  String orderId,Model model) throws
ExecutionException, InterruptedException {
        // 修改订单状态
        System.out.println(orderId);
        model.addAttribute("orderId",orderId);
        return "fail";
    }

    @RequestMapping("/pay")
    public String pay(String orderId,Model model) throws ExecutionException,
InterruptedException {
        // 修改订单状态
        System.out.println(orderId+"订单状态为已支付");
        model.addAttribute("orderId",orderId);
        return "success";
    }
```

```
    @RequestMapping("/cancelOrderView")
    public String cancelOrderView(String orderId,Model model) throws
ExecutionException, InterruptedException {
        // 修改订单状态
        System.out.println(orderId+"订单状态为已取消");
        model.addAttribute("orderId",orderId);
        return "cancelOrderView";
    }
}
```

订单实体类Order.java OrderDetail.java

```
@Data
public class Order {
    private String orderId;

    private String userId;

    private String status;

    private ArrayList<OrderDetail> detail;
}

@Data
public class OrderDetail {
    private String itemId;

    private String itemName;

    private double itemPrice;

    private int num;
}
```

静态页面order.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,shrink-
to-fit=no">
    <title>支付界面</title>
    <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{/css/signin.css}" rel="stylesheet">
    <script src="../static/js/jquery-1.10.2.js"></script>
</head>
<script language="javascript">
    var num = 9; //倒计时的秒数
    var URL = "/cancelOrderView?orderId=[[${orderId}]]";
    var id = window.setInterval('doUpdate()', 1000);
    function doUpdate() {
        document.getElementById('page_div').innerHTML = '支付时间还剩'+num+'秒' ;
        if(num == 0) {
            window.clearInterval(id);
            window.location = URL;
```

```
        }
        num --;
    }
</script>
<body>
<div  th:text="${orderId}"></div>
<div>订单创建成功</div>
<div class="time"> <p id="page_div">支付时间还剩10s，请尽快支付</p>
<a th:href="@{/pay/(orderId=${orderId})}">去支付</a>
</div>
</body>
</html>
```

其他静态页面

cancelOrderView.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,shrink-
to-fit=no">
    <title>取消订单</title>
    <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{/css/signin.css}" rel="stylesheet">
    <script src="../static/js/jquery-1.10.2.js"></script>
</head>
<body>
<div  th:text="${orderId}"></div>订单已取消
</body>
</html>
```

failOrder.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    failorder
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <a href="/createOrder">下单</a>
</body>
</html>
```

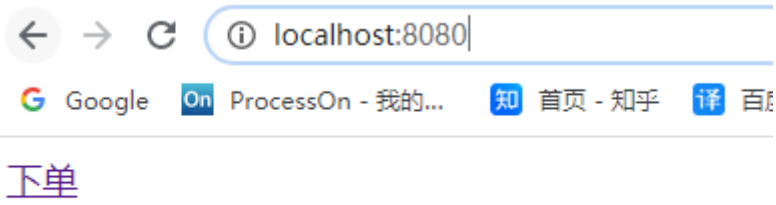
success.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,shrink-
to-fit=no">
    <title>取消订单</title>
    <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
    <link th:href="@{/css/signin.css}" rel="stylesheet">
    <script src="../static/js/jquery-1.10.2.js"></script>
</head>
<body>
<div  th:text="${orderId}"></div>支付成功
</body>
</html>
```

三、效果演示

前置准备，启动rabbitmq。设置Virtualhost = test 的虚机

1、启动spring项目。访问页面



2、点击下单按钮。不支付

支付时间还剩9秒

c6adf5f3-a订单创建成功

去支付

页面倒计时，查看后台日志

```
发送延迟取消信息，10s不支付就取消,当前时间2021-03-09 11:14:44
写数据库，订单为待支付
Order(orderId=c6adf5f3-a, userId=jianyi, status=待支付, detail=[OrderDetail(itemId=2bdc2, itemName=, itemPrice=100.0, num=2)])
取消订单"c6adf5f3-a"时间2021-03-09 11:14:54
c6adf5f3-a订单状态为已取消
c6adf5f3-a订单状态为已取消
```

10s后自动取消订单，并跳转订单取消页面

c6adf5f3-a订单已取消

3、点击下单按钮。支付

ff25805c-7支付成功

查看后台日志

```
写数据库，订单为待支付
Order(orderId=ff25805c-7, userId=jianyi, status=待支付, detail=[OrderDetail(itemId=cece6, itemName=, itemPrice=100.0, num=2)])
ff25805c-7订单状态为已支付
取消订单"ff25805c-7"时间2021-03-09 11:17:16
```

4、也可查看rabbitmq队列的变化，可自行查看