

# 订单状态定时处理、来单提醒和客户催单

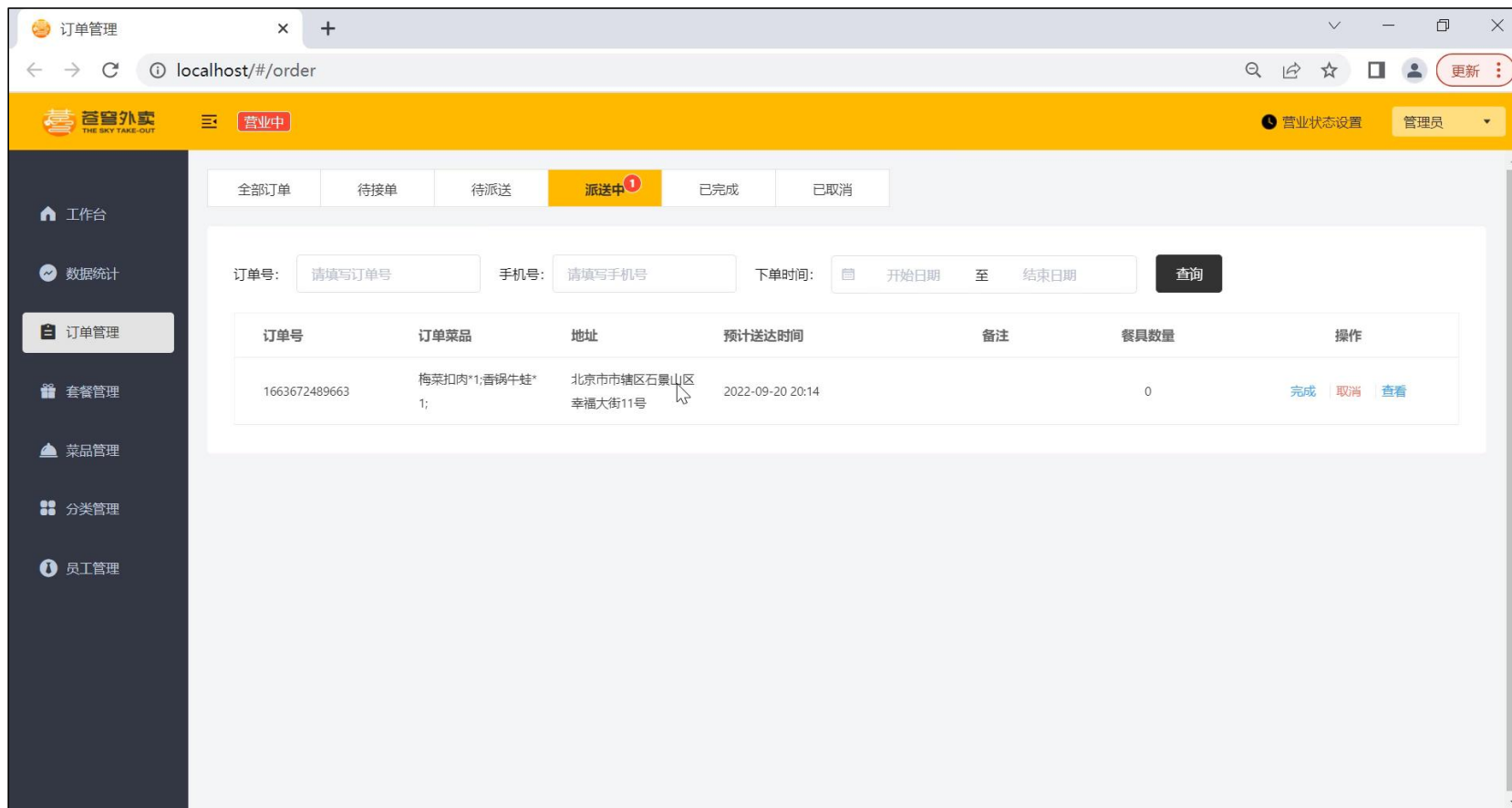


黑马程序员  
[www.itheima.com](http://www.itheima.com)

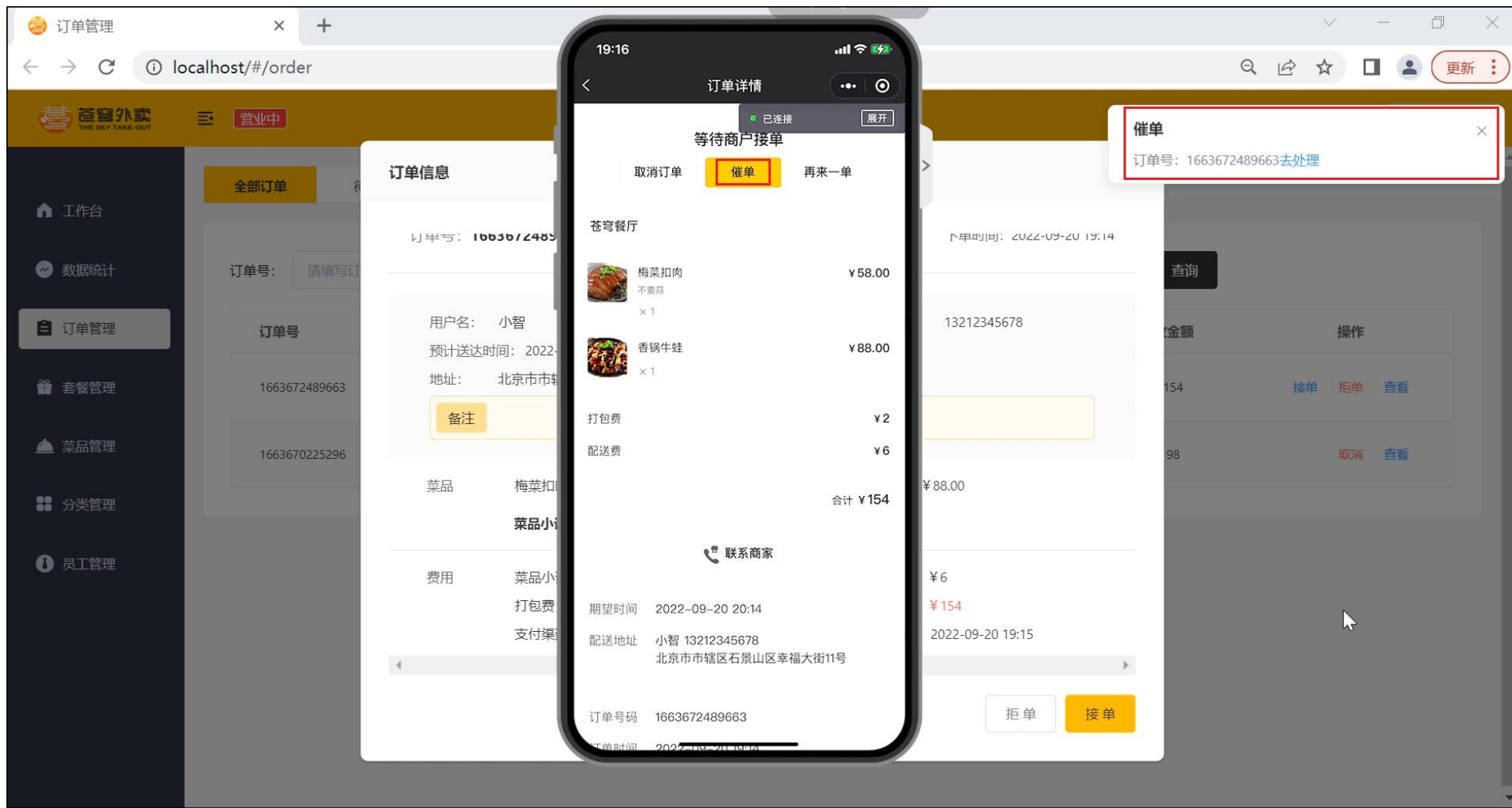
传智教育旗下  
高端IT教育品牌



支付超时的订单如何处理?



派送中的订单一直不点击完成如何处理?





# 目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



# Spring Task

- 介绍
- cron表达式
- 入门案例

## Spring Task 介绍

Spring Task 是Spring框架提供的任务调度工具，可以按照约定的时间自动执行某个代码逻辑。

# 定位：定时任务框架

作用：定时自动执行某段Java代码



为什么要在Java程序中使用Spring Task?



## Spring Task 介绍

应用场景：

- 信用卡每月还款提醒
- 银行贷款每月还款提醒
- 火车票售票系统处理未支付订单
- 入职纪念日为用户发送通知

只要是需要定时处理的场景都可以使用Spring Task





# Spring Task

- 介绍
- cron表达式
- 入门案例



## cron表达式

cron表达式其实就是一个字符串，通过cron表达式可以**定义任务触发的时间**

构成规则：分为6或7个域，由空格分隔开，每个域代表一个含义

每个域的含义分别为：秒、分钟、小时、日、月、周、年(可选)

秒	分钟	小时	日	月	周	年
0	0	9	12	10	?	2022

2022年10月12日上午9点整 对应的cron表达式为**0 0 9 12 10 ?**  
**2022**

## cron表达式

cron表达式在线生成器: <https://cron.qqe2.com/>

秒

分钟

小时

日

月

周

年

☒ 每秒 允许的通配符[ - \* / ]

☐ 周期从  -  秒

☐ 从  秒开始,每  秒执行一次

☐ 指定

☐ 00 ☐ 01 ☐ 02 ☐ 03 ☐ 04 ☐ 05 ☐ 06 ☐ 07 ☐ 08 ☐ 09

☐ 10 ☐ 11 ☐ 12 ☐ 13 ☐ 14 ☐ 15 ☐ 16 ☐ 17 ☐ 18 ☐ 19

☐ 20 ☐ 21 ☐ 22 ☐ 23 ☐ 24 ☐ 25 ☐ 26 ☐ 27 ☐ 28 ☐ 29

☐ 30 ☐ 31 ☐ 32 ☐ 33 ☐ 34 ☐ 35 ☐ 36 ☐ 37 ☐ 38 ☐ 39

☐ 40 ☐ 41 ☐ 42 ☐ 43 ☐ 44 ☐ 45 ☐ 46 ☐ 47 ☐ 48 ☐ 49

☐ 50 ☐ 51 ☐ 52 ☐ 53 ☐ 54 ☐ 55 ☐ 56 ☐ 57 ☐ 58 ☐ 59

表达式

秒

分钟

小时

日

月

星期

年

表达式字段:

Cron 表达式: 

反解析到UI



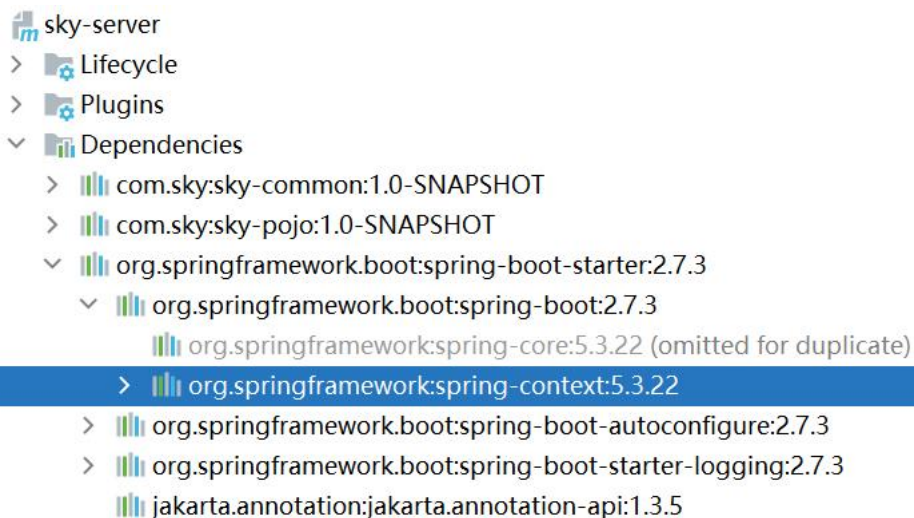
# Spring Task

- 介绍
- cron表达式
- 入门案例

## 入门案例

Spring Task使用步骤：

- ① 导入maven坐标 spring-context（已存在）
- ② 启动类添加注解 @EnableScheduling 开启任务调度
- ③ 自定义定时任务类





# 目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



## 订单状态定时处理

- 需求分析
- 代码开发
- 功能测试

## 需求分析

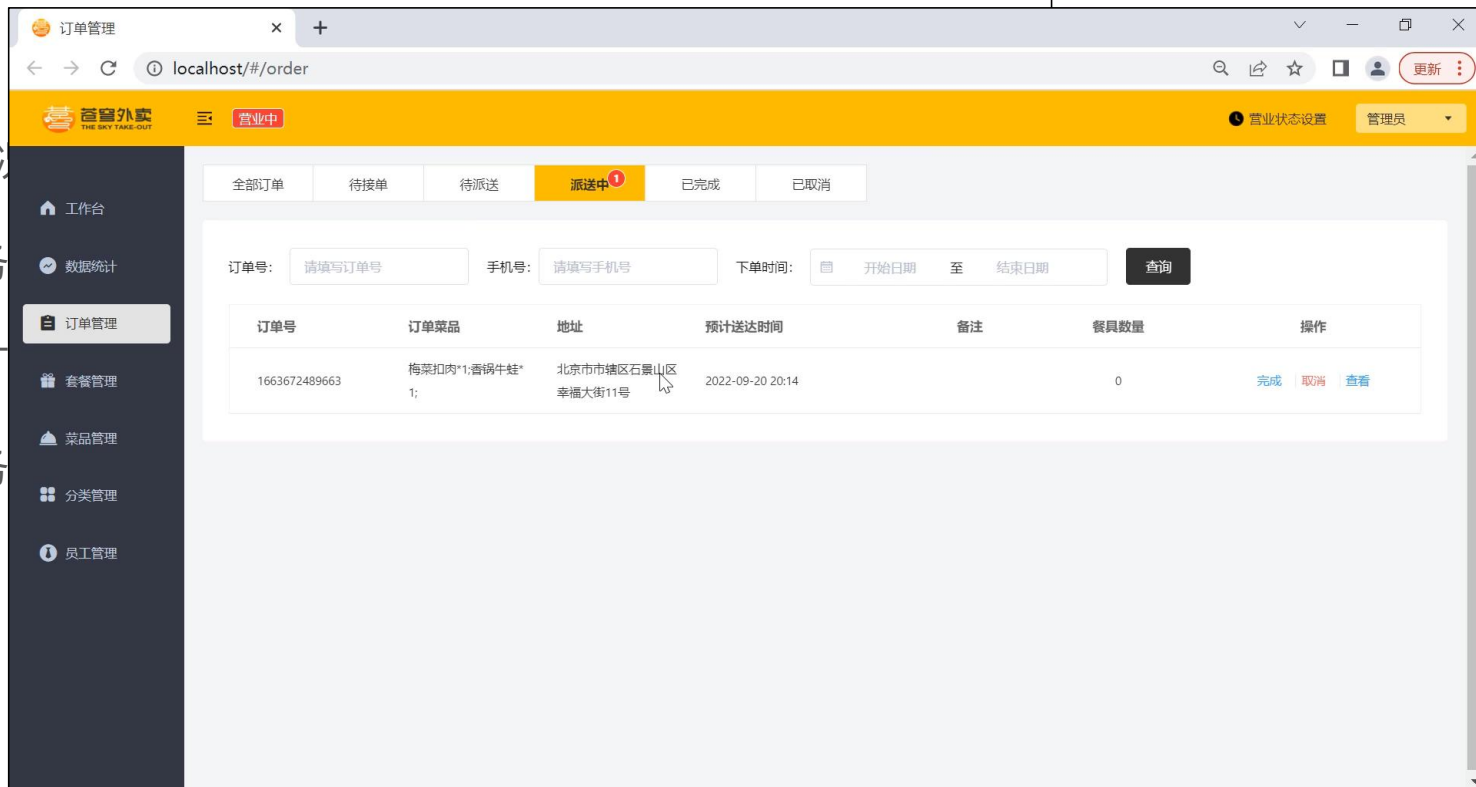
用户下单后可能存在的情况：

- 下单后未支付，订单一直处于“待支付”状态
- 用户收货后管理端未点击完成按钮，订单一直处于“派送中”状态



对于上面两种情况

- 通过定时任务
- 存在则修改订
- 通过定时任务



支付超时订单)，如果

已完成”



## 订单状态定时处理

- 需求分析
- 代码开发
- 功能测试



## 代码开发

自定义定时任务类OrderTask（待完善）：

```
@Component
@Slf4j
public class OrderTask {
    @Autowired
    private OrderMapper orderMapper;

    /**
     * 支付超时订单处理
     * 对于下单后超过15分钟仍未支付的订单自动修改状态为[已取消]
     */
    @Scheduled(cron = "0 * * * * ?") // 每分钟执行一次
    public void processTimeoutOrder() {
        log.info("开始进行支付超时订单处理:{}", LocalDateTime.now());
    }

    /**
     * 派送中状态的订单处理
     * 对于一直处于派送中状态的订单，自动修改状态为[已完成]
     */
    @Scheduled(cron = "0 0 1 * * ?") // 每天凌晨1点执行一次
    public void processDeliveryOrder(){
        log.info("开始进行未完成订单状态处理:{}", LocalDateTime.now());
    }
}
```

## 代码开发

在OrderMapper接口中扩展方法

```
/**  
 * 根据订单状态和下单时间查询订单  
 * @param status  
 * @param orderTime  
 */  
@Select("select * from orders where status = #{status} and order_time < #{orderTime}")  
List<Orders> getByStatusAndOrdertimeLT(Integer status, LocalDateTime orderTime);
```

## 代码开发

完善定时任务类的processTimeoutOrder方法：

```
@Scheduled(cron = "0 * * * * ?")
public void processTimeoutOrder(){
    log.info("处理支付超时订单: {}", new Date());

    LocalDateTime time = LocalDateTime.now().plusMinutes(-15);
    List<Orders> ordersList = orderMapper.getByStatusAndOrdertimeLT(Orders.PENDING_PAYMENT, time);

    if(ordersList != null && ordersList.size() > 0){
        ordersList.forEach(order -> {
            order.setStatus(Orders.CANCELLED);
            order.setCancelReason("支付超时, 自动取消");
            order.setCancelTime(LocalDateTime.now());
            orderMapper.update(order);
        });
    }
}
```

## 代码开发

完善定时任务类的processDeliveryOrder方法：

```
@Scheduled(cron = "0 0 1 * * ?")
public void processDeliveryOrder(){
    log.info("处理派送中订单: {}", new Date());

    LocalDateTime time = LocalDateTime.now().plusMinutes(-60);
    List<Orders> ordersList = orderMapper.getByStatusAndOrdertimeLT(Orders.DELIVERY_IN_PROGRESS, time);

    if(ordersList != null && ordersList.size() > 0){
        ordersList.forEach(order -> {
            order.setStatus(Orders.COMPLETED);
            orderMapper.update(order);
        });
    }
}
```



## 订单状态定时处理

- 需求分析
- 代码开发
- 功能测试

## 功能测试

可以通过如下方式进行测试：

- 查看控制台sql
- 查看数据库中数据变化



# 目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



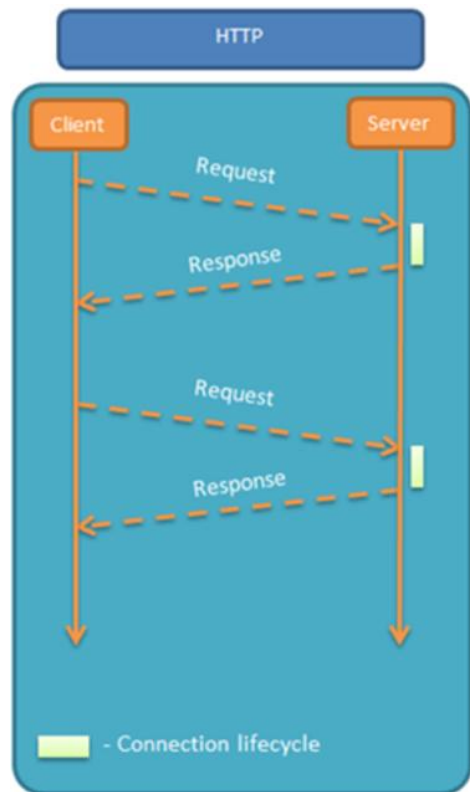
## WebSocket

- 介绍
- 入门案例



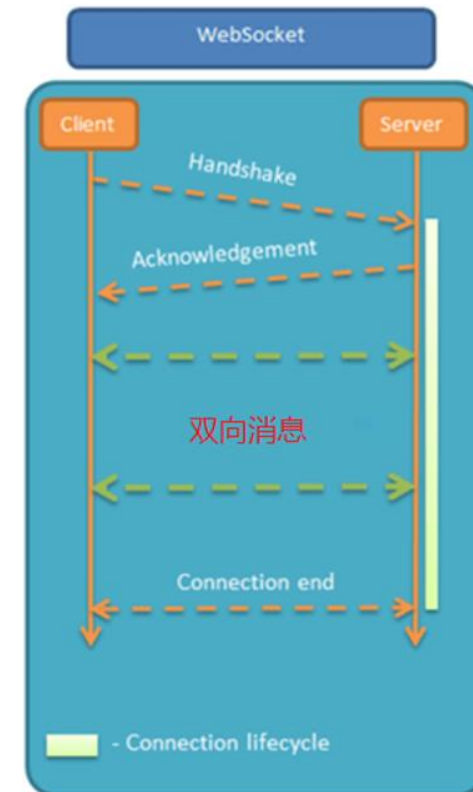
## 介绍

WebSocket 是基于 TCP 的一种新的网络协议。它实现了浏览器与服务器全双工通信——浏览器和服务器只需要完成一次握手，两者之间就可以创建持久性的连接，并进行双向数据传输。



HTTP协议和WebSocket协议对比：

- HTTP是短连接
- WebSocket是长连接
- HTTP通信是单向的，基于请求响应模式
- WebSocket支持双向通信
- HTTP和WebSocket底层都是TCP连接



## 介绍

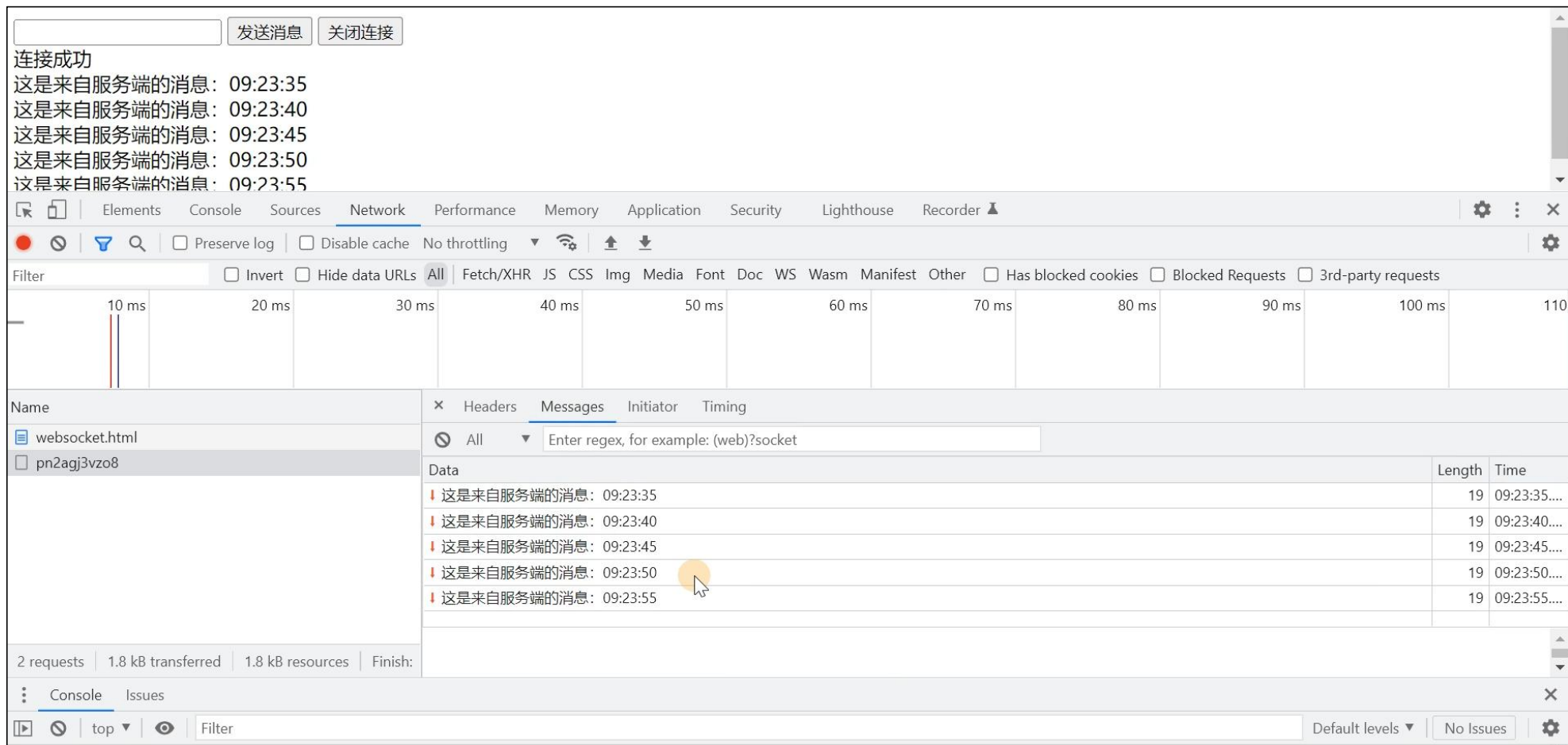
### 应用场景：

- 视频弹幕
- 网页聊天
- 体育实况更新
- 股票基金报价实时更新



## 介绍

效果展示：



The screenshot displays a web application interface at the top with a text input, a '发送消息' (Send Message) button, and a '关闭连接' (Close Connection) button. Below the input, a log shows five messages: '连接成功' (Connection successful) and four messages from the server at timestamps 09:23:35, 09:23:40, 09:23:45, 09:23:50, and 09:23:55.

The bottom half of the image shows the Chrome DevTools Network tab. The 'Messages' sub-tab is selected for the 'websocket.html' resource. A filter '(web)?socket' is applied. The message log shows five entries, each with a red icon and the text '这是来自服务端的消息: [timestamp]'. A mouse cursor is hovering over the entry for 09:23:50.

Name	Headers	Messages	Initiator	Timing
websocket.html				
pn2agj3vzo8				

Data	Length	Time
这是来自服务端的消息: 09:23:35	19	09:23:35....
这是来自服务端的消息: 09:23:40	19	09:23:40....
这是来自服务端的消息: 09:23:45	19	09:23:45....
这是来自服务端的消息: 09:23:50	19	09:23:50....
这是来自服务端的消息: 09:23:55	19	09:23:55....

Summary: 2 requests | 1.8 kB transferred | 1.8 kB resources | Finish:



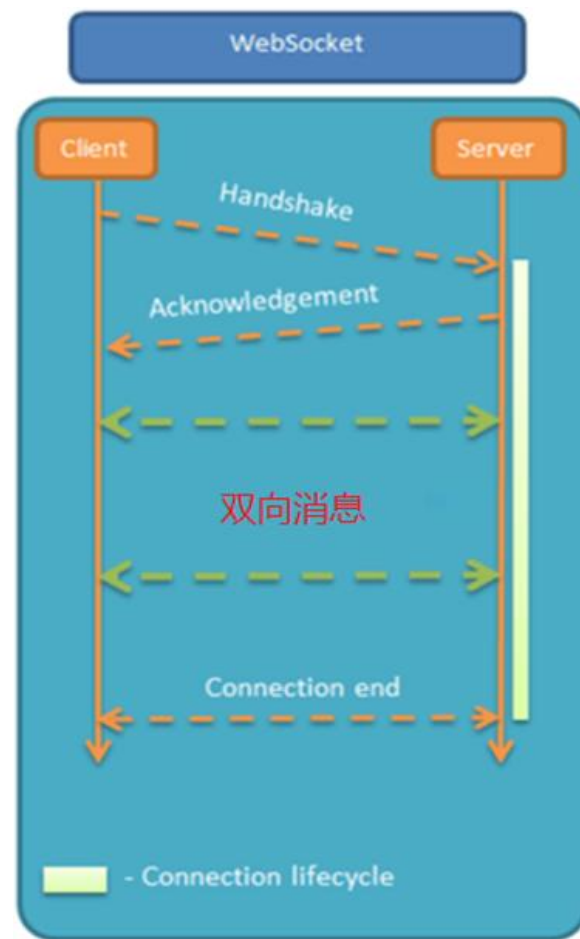
## WebSocket

- 介绍
- 入门案例

## 入门案例

实现步骤：

- ① 直接使用websocket.html页面作为WebSocket客户端
- ② 导入WebSocket的maven坐标
- ③ 导入WebSocket服务端组件WebSocketServer，用于和客户端通信
- ④ 导入配置类WebSocketConfiguration，注册WebSocket的服务端组件
- ⑤ 导入定时任务类WebSocketTask，定时向客户端推送数据





思考

既然WebSocket支持双向通信，功能看似比HTTP强大，那么我们是不是可以基于WebSocket开发所有的业务功能？

WebSocket缺点：

- 服务器长期维护长连接需要一定的成本
- 各个浏览器支持程度不一
- WebSocket 是长连接，受网络限制比较大，需要处理好重连

结论：WebSocket并不能完全取代HTTP，它只适合在特定的场景下使用



# 目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单




## 来单提醒

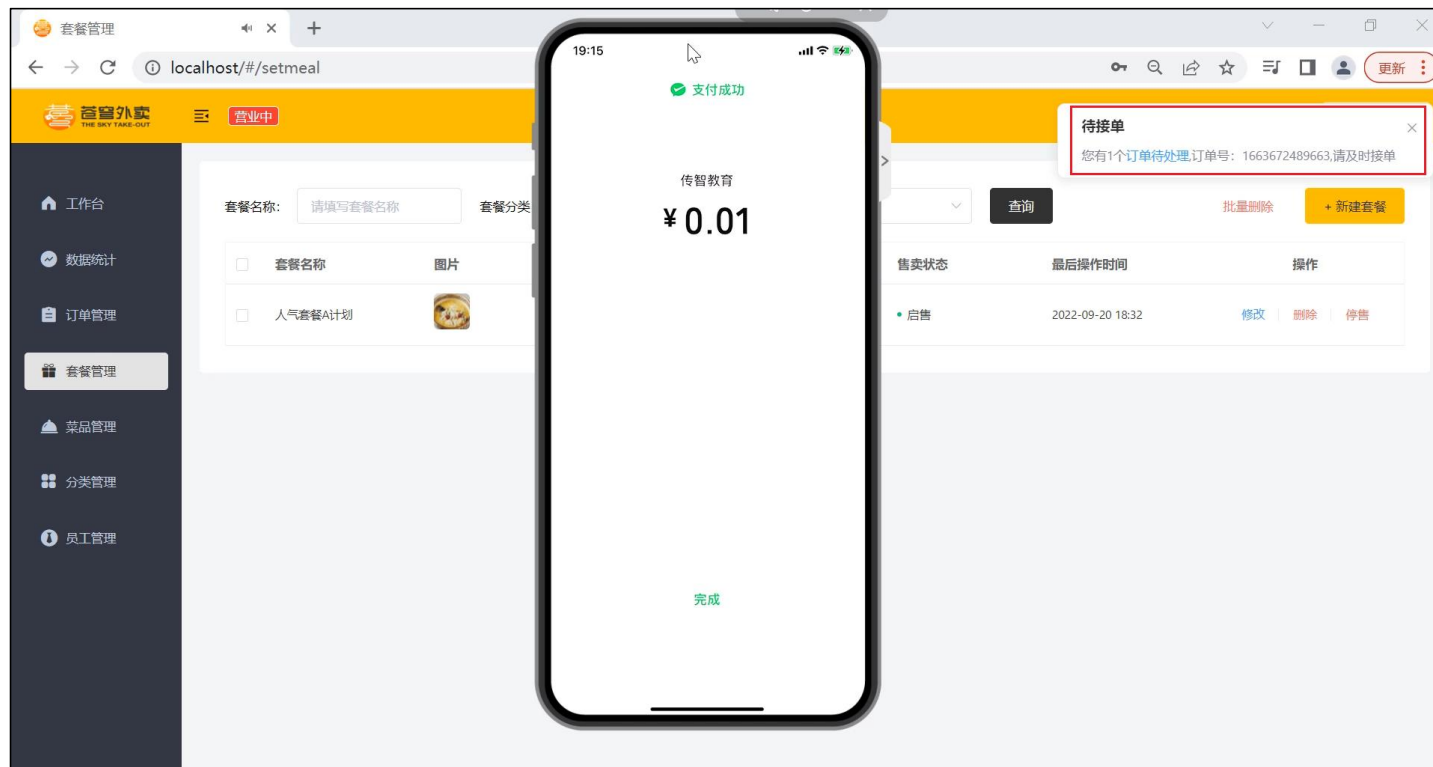
- 需求分析和设计
- 代码开发
- 功能测试



## 需求分析和设计

用户下单并且支付成功后，需要第一时间通知外卖商家。通知的形式有如下两种：

- 语音播报 
- 弹出提示框



## 需求分析和设计

设计:

- 通过WebSocket实现管理端页面和服务端保持长连接状态
- 当客户支付后，调用WebSocket的相关API实现服务端向客户端推送消息
- 客户端浏览器解析服务端推送的消息，判断是来单提醒还是客户催单，进行相应的消息提示和语音播报
- 约定服务端发送给客户端浏览器的数据格式为JSON，字段包括：type, orderId, content
  - type 为消息类型，1为来单提醒 2为客户催单
  - orderId 为订单id
  - content 为消息内容



## 来单提醒

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

在OrderServiceImpl中注入WebSocketServer对象，修改paySuccess方法，加入如下代码：

```
Map map = new HashMap();  
map.put("type", 1); //通知类型 1来单提醒 2客户催单  
map.put("orderId", orders.getId()); //订单id  
map.put("content", "订单号:" + outTradeNo);  
  
websocketServer.sendToAllClient(JSON.toJSONString(map));
```



## 来单提醒

- 需求分析和设计
- 代码开发
- 功能测试

## 功能测试

可以通过如下方式进行测试：

- 查看浏览器调试工具数据交互过程
- 前后端联调



# 目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单




## 客户催单

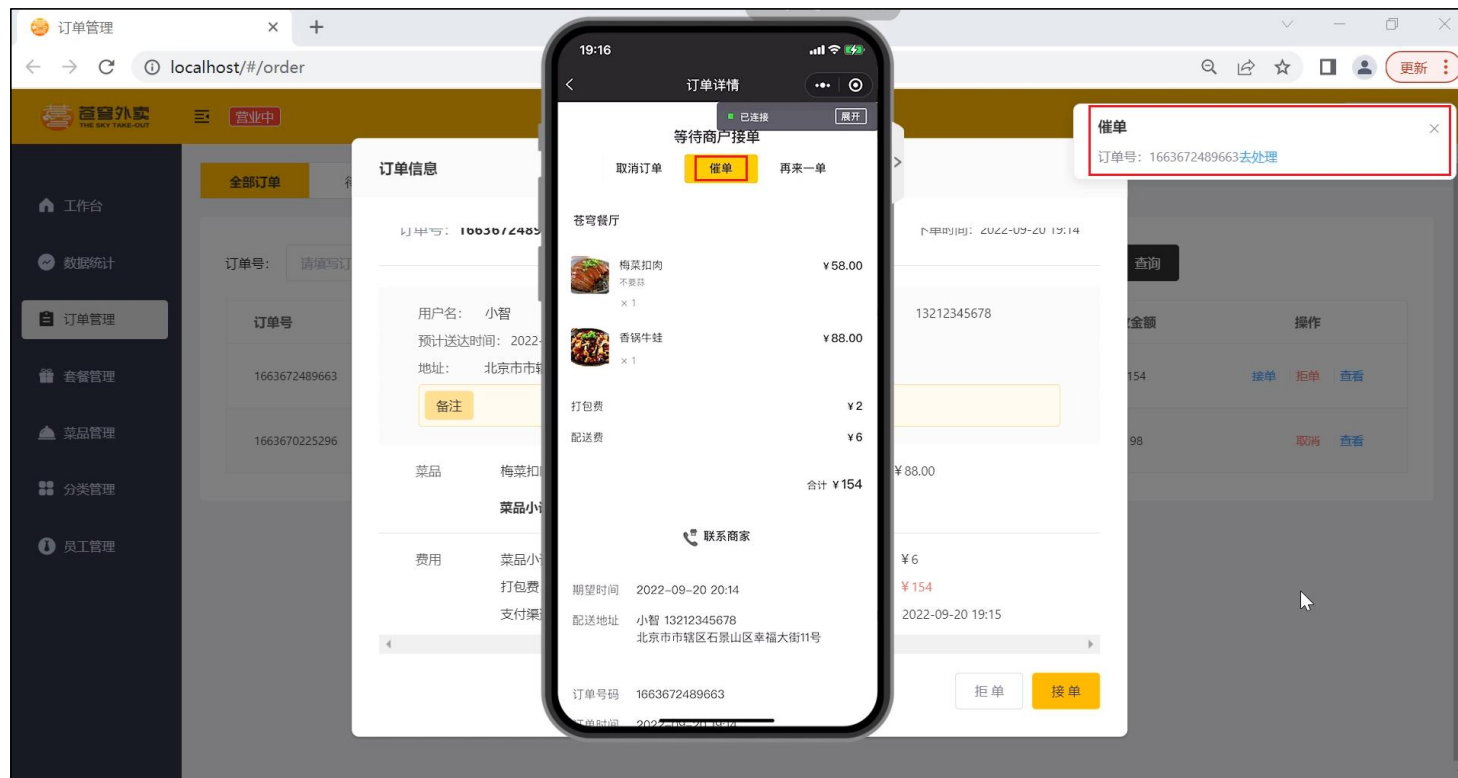
- 需求分析和设计
- 代码开发
- 功能测试



## 需求分析和设计

用户在小程序中点击催单按钮后，需要第一时间通知外卖商家。通知的形式有如下两种：

- 语音播报 
- 弹出提示框



## 需求分析和设计

设计:

- 通过WebSocket实现管理端页面和服务端保持长连接状态
- 当用户点击催单按钮后，调用WebSocket的相关API实现服务端向客户端推送消息
- 客户端浏览器解析服务端推送的消息，判断是来单提醒还是客户催单，进行相应的消息提示和语音播报
- 约定服务端发送给客户端浏览器的数据格式为JSON，字段包括：type, orderId, content
  - type 为消息类型，1为来单提醒 2为客户催单
  - orderId 为订单id
  - content 为消息内容

## 需求分析和设计

接口设计：

### 基本信息

Path: /user/order/reminder/{id}

Method: GET

接口描述：

### 请求参数

#### 路径参数

参数名称	示例	备注
id	101	订单id

### 返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



## 客户催单

- 需求分析和设计
- 代码开发
- 功能测试

## 代码开发

根据用户催单的接口定义，在user/OrderController中创建催单方法：

```
/**
 * 用户催单
 *
 * @param id
 * @return
 */
@GetMapping("/reminder/{id}")
@ApiOperation("用户催单")
public Result reminder(@PathVariable("id") Long id) {
    orderService.reminder(id);
    return Result.success();
}
```

## 代码开发

在OrderService接口中声明reminder方法:

```
/**  
 * 用户催单  
 * @param id  
 */  
void reminder(Long id);
```

## 代码开发

在OrderServiceImpl中实现reminder方法:

```
/**
 * 用户催单
 *
 * @param id
 */
public void reminder(Long id) {
    // 查询订单是否存在
    Orders orders = orderMapper.getById(id);
    if (orders == null) {
        throw new OrderBusinessException(MessageConstant.ORDER_NOT_FOUND);
    }

    //基于WebSocket实现催单
    Map map = new HashMap();
    map.put("type", 2); //2代表用户催单
    map.put("orderId", id);
    map.put("content", "订单号: " + orders.getNumber());
    websocketServer.sendToAllClient(JSON.toJSONString(map));
}
```



## 客户催单

- 需求分析和设计
- 代码开发
- 功能测试



## 功能测试

可以通过如下方式进行测试：

- 查看浏览器调试工具数据交互过程
- 前后端联调



传智教育旗下高端IT教育品牌