苍穹外卖day10

1. Spring Task

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（定时任务）。

1. cron表达式

在线Cron表达式生成器：http://cron.qqe2.com，生成的有点问题。

标准Unix cron：

- `0` = 周日，`1` = 周一，`2` = 周二，... `6` = 周六。
- 因此 `5` 代表星期五，`6L` 可能被解析为 **每月最后一个周六**（若系统支持）。

Quartz cron：

- `1` = 周日，`2` = 周一，... `7` = 周六。
- `6L` 表示 **每月最后一个星期五**（此时 `6` 对应周五）。

系统兼容性

- 标准Linux cron **不支持** 在周字段使用 `L` 或者 `#` 字符。
- Quartz（Java任务调度框架）允许在周字段使用 `L`，但需注意数值映射规则

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）

日和周的位置往往只能定义一个

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

```
0 0/30 9-17 * * ?   朝九晚五工作时间内每半小时

0 15 10 ? * MON-FRI   表示周一到周五每天上午10:15执行作业

0 15 10 ? 6L 2002-2006   表示2002-2006年的每个月的最后一个星期五上午10:15执行作业

0 0 10,14,16 * * ?   每天上午10点，下午2点，4点

0 15 10 ? * 6#3   每月的第三个星期五上午10:15触发
```

2. 入门案例

导入maven坐标spring-context

启动类skyapplication添加注解

```
@EnableScheduling
```

翻看Scheduled.java，可以发现cron的语法又与上面不同

The fields read from left to right are interpreted as follows.

- second
- minute
- hour
- day of month
- month
- day of week

sky-server/src/main/java/com/sky/task/MyTask.java

```
package com.sky.task;

import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
@Slf4j
public class MyTask {

    @Scheduled(cron="0/5 * * * * ?") //每5秒执行一次，不指定周
    public void executeTask() {
        log.info("定时任务开始执行：{}",new Date());
```

```
    }
}
```

```
: 定时任务开始执行：Fri May 30 11:16:05 CST 2025
: 定时任务开始执行：Fri May 30 11:16:10 CST 2025
: 定时任务开始执行：Fri May 30 11:16:15 CST 2025
```

2. 订单状态定时处理

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

对于上面两种情况需要通过定时任务来修改订单状态，具体逻辑为：

- 通过定时任务每分钟检查一次是否存在支付超时订单（下单后超过15分钟仍未支付则判定为支付超时订单），如果
  存在则修改订单状态为“已取消”
- 通过定时任务每天凌晨1点检查一次是否存在“派送中”的订单，如果存在则修改订单状态为“已完成”

注释掉MyTask.java

sky-server/src/main/java/com/sky/task/OrderTask.java

以下代码为测试时候用，测完后把`@Scheduled`改回原来的

```java
package com.sky.task;


import com.sky.entity.Orders;
import com.sky.mapper.OrderMapper;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.weaver.ast.Or;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;
import java.util.List;

/**
 * 定时任务类，定时处理订单状态
 */
```

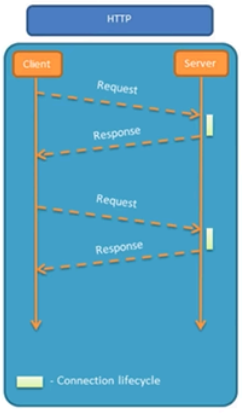
```java
@Component
@Slf4j
public class OrderTask {

    @Autowired
    private OrderMapper orderMapper;

    /**
     * 处理超时订单的方法
      */
    @Scheduled(cron="0 * * * * ?") //每分钟触发一次
    //@Scheduled(cron = "0/5 * * * * ?")
    public void processTimeoutOrder(){
        log.info("定时处理超时订单：{}", LocalDateTime.now());
        //select * from orders where status=? and order_time < (当前时间-15分钟)
        LocalDateTime time = LocalDateTime.now().plusMinutes(-15);
        List<Orders> orderList = orderMapper.getByStatusAndOrderTimeLT(Orders.PENDING_PAYMENT, time);

        if(orderList != null && orderList.size() > 0){
            for(Orders orders : orderList){
                orders.setStatus(Orders.CANCELLED);
                orders.setCancelReason("订单超时，自动取消");
                orders.setCancelTime(LocalDateTime.now());
                orderMapper.update(orders);
            }
        }

    }


    /**
     * 处理一直处于派送中状态的订单
     */
    @Scheduled(cron="0 0 1 * * ?")//每天凌晨1点触发一次
    //@Scheduled(cron = "1/5 * * * * ?")
    public void processDeliveryOrder(){
        log.info("定时处理处于派送中的订单：{}", LocalDateTime.now());
        LocalDateTime time = LocalDateTime.now().plusMinutes(-60);//查询上一天的订单
        List<Orders> ordersList=orderMapper.getByStatusAndOrderTimeLT(Orders.DELIVERY_IN_PROGRESS, time);

        if(ordersList != null && ordersList.size() > 0){
            for(Orders orders : ordersList){
                orders.setStatus(Orders.COMPLETED);
                orderMapper.update(orders);
            }
        }
    }

}

```

OrderMapper.java

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

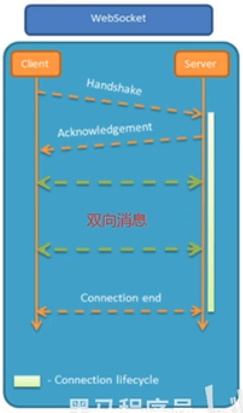
3. WebSocket

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



sky-server/src/main/java/com/sky/task/WebSocketTask.java

```
package com.sky.task;

import com.sky.websocket.WebSocketServer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@Component
public class WebSocketTask {
    @Autowired
```

```
    private WebSocketServer webSocketServer;

    /**
     * 通过WebSocket每隔5秒向客户端发送消息
     */
    @Scheduled(cron = "0/5 * * * * ?")
    public void sendMessageToClient() {
        webSocketServer.sendToAllClient("这是来自服务端的消息：" +
DateTimeFormatter.ofPattern("HH:mm:ss").format(LocalDateTime.now()));
    }
}
```

sky-server/src/main/java/com/sky/websocket/WebSocketServer.java

```
package com.sky.websocket;

import org.springframework.stereotype.Component;
import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.PathParam;
import javax.websocket.server.ServerEndpoint;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * WebSocket服务
 */
@Component
@ServerEndpoint("/ws/{sid}")
public class WebSocketServer {

    //存放会话对象
    private static Map<String, Session> sessionMap = new HashMap();

    /**
     * 连接建立成功调用的方法
     */
    @OnOpen
    public void onOpen(Session session, @PathParam("sid") String sid) {
        System.out.println("客户端：" + sid + "建立连接");
        sessionMap.put(sid, session);
    }

    /**
     * 收到客户端消息后调用的方法
     *
     * @param message 客户端发送过来的消息
     */
    @OnMessage
    public void onMessage(String message, @PathParam("sid") String sid) {
```

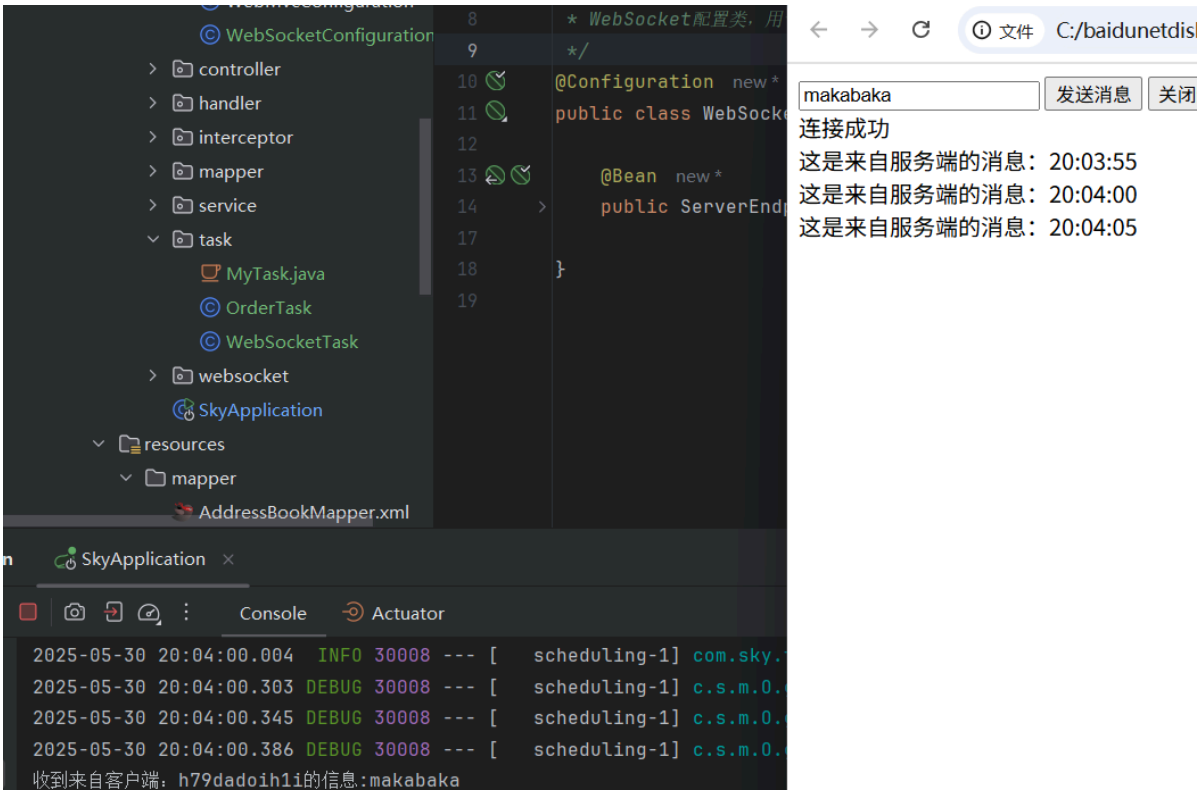
```
        System.out.println("收到来自客户端：" + sid + "的信息:" + message);
    }

    /**
     * 连接关闭调用的方法
     *
     * @param sid
     */
    @OnClose
    public void onClose(@PathParam("sid") String sid) {
        System.out.println("连接断开:" + sid);
        sessionMap.remove(sid);
    }

    /**
     * 群发
     *
     * @param message
     */
    public void sendToAllClient(String message) {
        Collection<Session> sessions = sessionMap.values();
        for (Session session : sessions) {
            try {
                //服务器向客户端发送消息
                session.getBasicRemote().sendText(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

}
```

sky-server/src/main/java/com/sky/config/WebSocketConfiguration.java

```
package com.sky.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.socket.server.standard.ServerEndpointExporter;

/**
 * WebSocket配置类，用于注册WebSocket的Bean
 */
@Configuration
public class WebSocketConfiguration {

    @Bean
    public ServerEndpointExporter serverEndpointExporter() {
        return new ServerEndpointExporter();
    }

```

```
18 }
```

启动skyapplication，访问
file:///C:/baidunetdiskdownload/%E8%B5%84%E6%96%99/day10/WebSocket%E5%85%A5%E9%97%A8%E6%A1%88%E4%BE%8B/websocket.html



4. 来单提醒

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

前端发送数据给nginx 80端口，nginx转发给后端8080端口的服务

翻看nginx.conf，以下片段核心功能是将 `/ws/` 路径的请求转发到后端WebSocket服务器（`http://webservers/ws/`）

```
# WebSocket
        location /ws/ {
            proxy_pass   http://webservers/ws/;
            proxy_http_version 1.1;
            proxy_read_timeout 3600s;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "$connection_upgrade";
        }
```

原本微信支付成功时，微信会访问application-dev.yml的这个地址

```
notifyUrl: http://73531d2e.r3.cpolar.cn/notify/paySuccess #前半部分是cpolar返回的公网ip地址
```

但我们调用微信支付的关键代码给注释掉了，所以我们得让`/user/order/payment`路由主动调用`/notify/paySuccess`路由下的paySuccess()函数

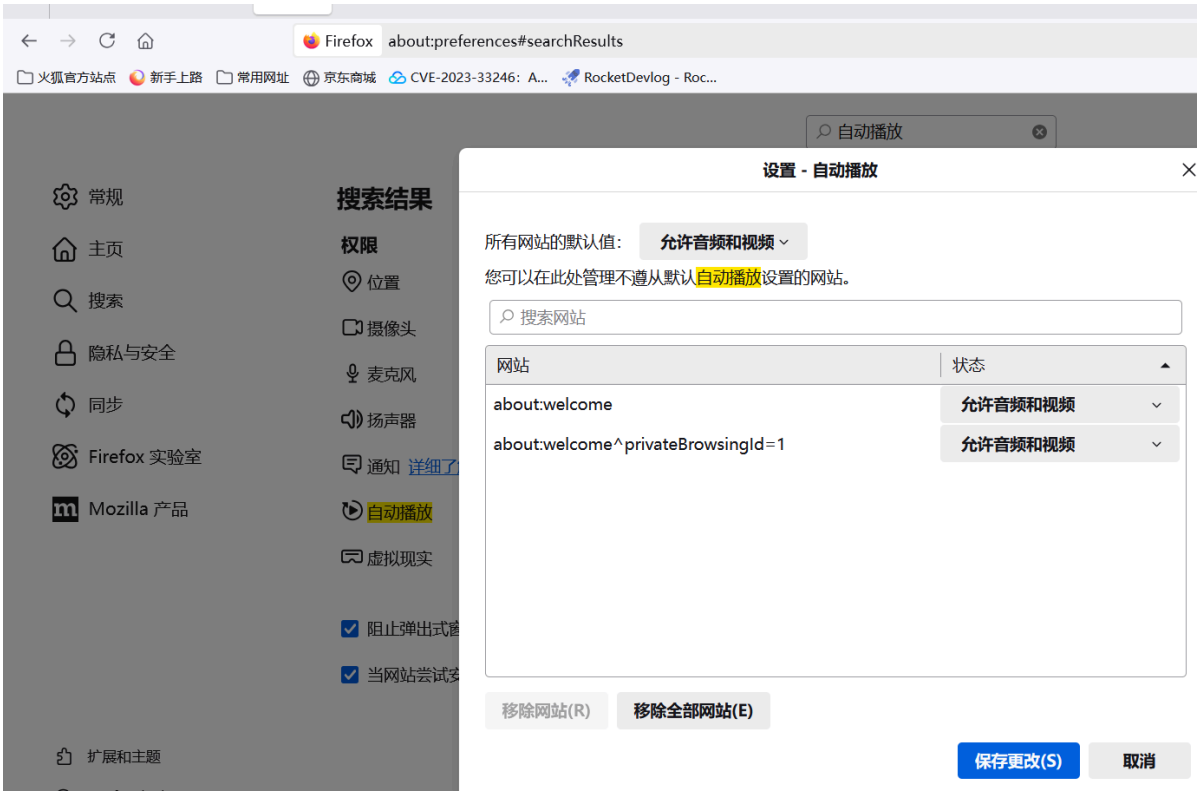
OrderServiceImpl.java

```
    @Autowired
    private WebSocketServer webSocketServer;

    /**
     * 订单支付
     *
     * @param ordersPaymentDTO
     * @return
     */
    public OrderPaymentVO payment(OrdersPaymentDTO ordersPaymentDTO) throws Exception {
        // 当前登录用户id
        Long userId = BaseContext.getCurrentId();
        User user = userMapper.getById(userId);

        //调用微信支付接口，生成预支付交易单
//        JSONObject jsonObject = weChatPayUtil.pay(
//                ordersPaymentDTO.getOrderNumber(), //商户订单号
//                new BigDecimal(0.01), //支付金额，单位 元
//                "苍穹外卖订单", //商品描述
//                user.getOpenid() //微信用户的openid
//        );
//
//        if (jsonObject.getString("code") != null && jsonObject.getString("code").equals("ORDERPAID")) {
//            throw new OrderBusinessException("该订单已支付");
//        }
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("code","ORDERPAID");
        OrderPaymentVO vo = jsonObject.toJavaObject(OrderPaymentVO.class);
        vo.setPackageStr(jsonObject.getString("package"));
        Integer OrderPaidStatus = Orders.PAID;//支付状态，已支付
```

```
        Integer OrderStatus = Orders.TO_BE_CONFIRMED;  //订单状态，待接单
        LocalDateTime check_out_time = LocalDateTime.now();//更新支付时间
        orderMapper.updateStatus(OrderStatus, OrderPaidStatus,
check_out_time, this.orders.getId());
        paySuccess(ordersPaymentDTO.getOrderNumber());//直接调用paySuccess方法
        return vo;
    }



    /**
     * 支付成功，修改订单状态
     *
     * @param outTradeNo
     */
    public void paySuccess(String outTradeNo) {

        // 根据订单号查询订单
        Orders ordersDB = orderMapper.getByNumber(outTradeNo);

        // 根据订单id更新订单的状态、支付方式、支付状态、结账时间
        Orders orders = Orders.builder()
                .id(ordersDB.getId())
                .status(Orders.TO_BE_CONFIRMED)
                .payStatus(Orders.PAID)
                .checkoutTime(LocalDateTime.now())
                .build();

        orderMapper.update(orders);

        //通过websocket向客户端浏览器推送消息 type orderId content
        Map map=new HashMap();
        map.put("type",1);//1表示来单提醒，2表示客户催单
        map.put("orderId",ordersDB.getId());
        map.put("content","订单号："+outTradeNo);

        String json= JSON.toJSONString(map);
        webSocketServer.sendToAllClient(json);//发送给所有客户端
    }
```

WebSocketTask.java注释掉 `@Scheduled(cron = "0/5 * * * * ?")` 防止音频一直响

firefox开启允许音频和视频



小程序开发工具支付成功后，我们在网页管理端查看一下

5. 客户催单

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			

com/sky/controller/user/OrderController.java

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

OrderService.java

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

OrderServiceImpl.java

```
    /**
```

```
     * 客户催单
     * @param id
     */
    @Override
    public void reminder(Long id) {
        // 根据id查询订单
        Orders ordersDB = orderMapper.getById(id);

        // 校验订单是否存在
        if (ordersDB == null) {
            throw new
OrderBusinessException(MessageConstant.ORDER_STATUS_ERROR);
        }

        //通过websocket向客户端浏览器推送消息 type orderId content
        Map map=new HashMap();
        map.put("type",2);//1表示来单提醒，2表示客户催单
        map.put("orderId",id);
        map.put("content","订单号："+ordersDB.getNumber());

        String json= JSON.toJSONString(map);
        webSocketServer.sendToAllClient(json);//发送给所有客户端
    }
```