

# 第4章 预约管理-定时任务Quartz、预约设置

---

目标:

- 掌握Quartz使用方式 什么是quartz 有什么作用? 什么时候用? 怎么用?
- 了解什么是Apache POI
- 掌握Apache POI的使用方法 【重点】
- 掌握批量导入预约设置信息的实现过程 (预约设置 业务 使用poi) 【重点】
- 掌握日历方式展示预约设置信息的实现过程 (看前端需要什么数据, 后台提供什么数据) 【重点】
- 掌握基于日历实现预约设置信息的实现过程 【重点】
- 了解常见的移动端开发方式 混合方式原因
- 了解微信公众号开发过程

## 1. 定时任务组件Quartz

---

### 【目标】

定时任务组件Quartz

- 清除文件上传所产生的垃圾图片

### 【路径】

1: Quartz介绍

- 掌握场景 (定时任务)

2: Quartz入门案例

- spring整合Quartz (spring中配置)

3: cron表达式

4: cron表达式在线生成器

### 【讲解】

### 1.1. Quartz介绍

---

Quartz是Job scheduling (任务调度) 领域的一个开源项目, Quartz既可以单独使用也可以跟spring框架整合使用, 在实际开发中一般会使用后者。使用Quartz可以开发一个或者多个定时任务, 每个定时任务可以单独指定执行的时间, 例如每隔1小时执行一次、每个月第一天上午10点执行一次、每个月最后一天下午5点执行一次等。

什么: 是一个做后台任务调度的开源框架

作用: 定时后台任务开发

什么时候用: 需要后定时/重复执行任务时可以使用

怎么用:

1. 引入maven依赖
2. 创建一个任务类, 做任务的方法实现

### 3. 添加spring配置文件

- 自定义的任务类要注册到spring容器
- 配置jobdetail, 调用spring容器中的bean对象(任务类)中的方法(任务类中的方法), 是否并发(多线程)
- 配置trigger触发器, 编写触发时机表达式(7子表达式)
- 配置scheduler调度容器

4. 要启动它, 只启动spring容器, 加载这个spring的配置文件即可

官网: <http://www.quartz-scheduler.org/>

maven坐标:

```
1 <!--quartz的基础包-->
2 <dependency>
3     <groupId>org.quartz-scheduler</groupId>
4     <artifactId>quartz</artifactId>
5     <version>2.2.1</version>
6 </dependency>
7 <dependency>
8     <groupId>org.quartz-scheduler</groupId>
9     <artifactId>quartz-jobs</artifactId>
10    <version>2.2.1</version>
11 </dependency>
```

```
1 <!--spring整合Quartz-->
2 <dependency>
3     <groupId>org.springframework</groupId>
4     <artifactId>spring-context-support</artifactId>
5     <version>5.0.2.RELEASE</version>
6 </dependency>
7 <dependency>
8     <groupId>org.springframework</groupId>
9     <artifactId>spring-tx</artifactId>
10    <version>5.0.2.RELEASE</version>
11 </dependency>
```

## 1.2. Quartz入门案例

### 【路径】

- 1: 创建maven工程quartzdemo, 打包方式为jar, 导入jar包
- 2: 自定义一个Job
- 3: 提供Spring配置文件applicationContext-jobs.xml, 配置自定义Job、任务描述、触发器、调度工厂等
- 4: 创建启动类, 使用ClassPathXmlApplicationContext启动spring容器

### 【讲解】

本案例基于Quartz和spring整合的方式使用。具体步骤:

- (1) 创建maven工程quartzdemo, 导入Quartz和spring相关坐标, pom.xml文件如下



## 导入jar包

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.itheima</groupId>
8     <artifactId>quartzdemo</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <dependencies>
12         <dependency>
13             <groupId>org.springframework</groupId>
14             <artifactId>spring-context-support</artifactId>
15             <version>5.0.2.RELEASE</version>
16         </dependency>
17         <dependency>
18             <groupId>org.springframework</groupId>
19             <artifactId>spring-tx</artifactId>
20             <version>5.0.2.RELEASE</version>
21         </dependency>
22         <dependency>
23             <groupId>org.springframework</groupId>
24             <artifactId>spring-web</artifactId>
25             <version>5.0.2.RELEASE</version>
26         </dependency>
27         <dependency>
28             <groupId>org.quartz-scheduler</groupId>
29             <artifactId>quartz</artifactId>
30             <version>2.2.1</version>
31         </dependency>
32         <dependency>
33             <groupId>org.quartz-scheduler</groupId>
34             <artifactId>quartz-jobs</artifactId>
35             <version>2.2.1</version>
36         </dependency>
37     </dependencies>
38 </project>
```

## (2) 自定义一个Job

```
1 package com.itheima.job;
2
3 import org.springframework.stereotype.Component;
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6
7 /**
8  * Description: No Description
9  * User: Eric
10  */
11 @Component
12 public class MyJob {
13
```

```

14     private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
15
16     public void abc(){
17         System.out.println(Thread.currentThread()+ ":" + sdf.format(new
Date()));
18     }
19 }
20

```

(3) 提供Spring配置文件application-jobs.xml, 配置自定义Job、任务描述、触发器、调度工厂等  
【路径】

- 1: 创建JobDetail对象,作用是负责通过反射调用指定的Job, 注入目标对象, 注入目标方法
- 2: 注册一个触发器, 指定任务触发的时间
- 3: 注册一个统一的调度工厂, 通过这个调度工厂调度任务

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
6
7  <!-- 注册任务到spring容器 -->
8      <context:component-scan base-package="com.itheima.job"/>
9  <!-- 任务策略 -->
10     <bean id="jobDetail"
class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
11 <!-- 任务触发时调用的 目标对象 targetObject
12     targetMethod 目标对象中的方法
13 -->
14         <property name="targetObject" ref="myJob"/>
15         <property name="targetMethod" value="abc"/>
16     <!-- 并发 concurrent=false 单线程
17         true: 在规定的周期内任务没完成时, 又触发新的任务, 属于同一任务时。使用多线程来处理任务
18     -->
19         <property name="concurrent" value="true"/>
20     </bean>
21 <!-- 触发器 -->
22     <bean id="trigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
23 <!-- 七子表达式: 秒 分 时 日 月 周 年 (省略) -->
24         <property name="cronExpression" value="0/2 * * * * ?"/>
25         <property name="jobDetail" ref="jobDetail"/>
26     </bean>
27 <!-- 调度容器 -->
28     <bean
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
29         <property name="triggers">
30             <list>

```

```

31         <ref bean="trigger"/>
32     </list>
33 </property>
34 </bean>
35 </beans>

```

#### (4) 创建启动类

```

1 package com.itheima.job;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 import java.io.IOException;
6
7 /**
8  * Description: No Description
9  * User: Eric
10 */
11 public class JobApplication {
12     public static void main(String[] args) throws IOException {
13         new ClassPathXmlApplicationContext("classpath:applicationContext-
14 jobs.xml");
15         // 阻塞
16         System.in.read();
17     }
18 }

```

执行上面main方法观察控制台，可以发现每隔2秒会输出一次，说明每隔2秒自定义Job被调用一次。

image-20200626171019467

## 1.3 注解方式【扩展】

创建任务类MyJob2

```

1 package com.itheima.job;
2
3 import org.springframework.scheduling.annotation.Scheduled;
4 import org.springframework.stereotype.Component;
5
6 import java.text.SimpleDateFormat;
7 import java.util.Date;
8
9 /**
10  * Description: No Description
11  * User: Eric
12  */
13 @Component
14 public class MyJob2 {
15
16     private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
17 MM-dd HH:mm:ss");
18 }


```

```

18  /**
19   * initialDelay: 启动时延迟多少毫秒后才执行
20   * fixedDelay: 每间隔多长时间执行
21   */
22  @Scheduled(initialDelay = 1000,fixedDelay = 2000)
23  //@Scheduled(cron = "0/2 * * * * ?")
24  public void tt(){
25      System.out.println("job2:" + sdf.format(new Date()));
26  }
27  }
28

```

配置文件application-jobs.xml中，添加

image-20200911151954942

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xmlns:task="http://www.springframework.org/schema/task"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans
7          http://www.springframework.org/schema/beans/spring-beans.xsd
8          http://www.springframework.org/schema/context
9          http://www.springframework.org/schema/context/spring-context.xsd
10         http://www.springframework.org/schema/task
11         http://www.springframework.org/schema/task/spring-task.xsd">
12
13  <!-- 注册任务到spring容器 -->
14      <context:component-scan base-package="com.itheima.job"/>
15  <!-- 任务策略 -->
16      <bean id="jobDetail"
17          class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
18  <!-- 任务触发时调用的 目标对象 targetObject
19          targetMethod 目标对象中的方法
20      -->
21          <property name="targetObject" ref="myJob"/>
22          <property name="targetMethod" value="abc"/>
23      <!-- 并发 concurrent=false 单线程
24          true: 在规定的周期内任务没完成时，又触发新的任务，属于同一任务时。使用多线程来处理任务
25      -->
26          <property name="concurrent" value="true"/>
27      </bean>
28  <!-- 触发器 -->
29      <bean id="trigger"
30          class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
31  <!-- 七子表达式: 秒 分 时 日 月 周 年 (省略) -->
32          <property name="cronExpression" value="0/2 * * * * ?"/>
33          <property name="jobDetail" ref="jobDetail"/>
34      </bean>
35  <!-- 调度容器 -->
36      <bean
37          class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
38          <property name="triggers">
39              <list>

```

```

37         <ref bean="trigger"/>
38     </list>
39 </property>
40 </bean>
41 <!--注解支持 【注意】：使用的约束是spring-task，不要导错了-->
42 <task:annotation-driven/>
43 <!--任务调度线程池-->
44 <bean
45     class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"/>

```

## 1.3. cron表达式

上面的入门案例中我们指定了一个表达式：0/2 \* \* \* \* ?

这种表达式称为cron表达式，通过cron表达式可以灵活的定义出符合要求的程序执行的时间。本小节我们就来学习一下cron表达式的使用方法。如下图：



cron表达式分为七个域，之间使用空格分隔。其中最后一个域（年）可以为空。每个域都有自己允许的值和一些特殊字符构成。使用这些特殊字符可以使我们定义的表达式更加灵活。

下面是对这些特殊字符的介绍：

逗号 (,)：指定一个值列表，例如使用在月域上1,4,5,7表示1月、4月、5月和7月

横杠 (-)：指定一个范围，例如在时域上3-6表示3点到6点（即3点、4点、5点、6点）

星号 (\*)：表示这个域上包含所有合法的值。例如，在月份域上使用星号意味着每个月都会触发

斜线 (/)：表示递增，例如使用在秒域上0/15表示每15秒. 每间隔多长时间

问号 (?)：只能用在日和周域上，但是不能在这两个域上同时使用。表示不指定，表达式中必须存在一个? 要么在日上，要么在周上

井号 (#)：只能使用在周域上，用于指定月份中的第几周的哪一天，例如6#3，意思是某月的第三个周五 (6=星期五，3意味着月份中的第三周) 星期天做为每周的第一天，1 代表星期天，2：星期一

母亲节: 0 0 8 ? 5 1 #2

L：某域上允许的最后一个值。只能使用在日和周域上。当用在日域上，表示的是在月域上指定的月份的最后一天。用于周域上时，表示周的最后一天，就是星期六

W：W 字符代表着工作日 (星期一到星期五)，只能用在日域上，它用来指定离指定日的最近的一个工作日

## 1.4. cron表达式在线生成器

前面介绍了cron表达式，但是自己编写表达式还是有一些困难的，我们可以借助一些cron表达式在线生成器来根据我们的需求生成表达式即可。

<https://www.bejson.com/othertools/cron/>



## 1.6 【小结】

## 1: Quartz介绍

- 掌握场景（后台定时任务）

## 2: Quartz入门案例

- 引入Quartz依赖, spring-context-support, spring-tx
- 创建任务类与方法
- spring配置文件:
  - 任务调度容器 SchedulerFactoryBean triggers
  - 触发器 CronTriggerFactoryBean, jobdetail
  - 任务策略 JobDetail, MethodInvokingJobDetailFactoryBean targetObject.targetMethod
  - 自定义的任务, 要进入spring容器

## 3: cron表达式 七子表达式 秒 分 时 日 月 周 年

## 4: cron表达式在线生成器

<<https://qge2.com/cron>>有问题

<https://www.bejson.com/othertools/cron/> 可以用

# 2. 定时清理垃圾图片 【重在使用】

---

## 【目标】

Quartz整合项目, 完成定时7牛上的清理垃圾图片

## 【路径】

垃圾图片定义: 数据库上的图片就是要的, 数据中没有的是不需要。图片是存到7牛上。

垃圾图片: 7牛上的减掉数据上的剩下的就是垃圾

希望后台自动运行, 什么时候。触发时机凌晨4点, 没有人来操作时, 才能执行

实现步骤:

1. 创建health\_jobs工程
2. 引入依赖 接口health\_interface, quartz的2个依赖
3. 编写任务类与任务的方法
4. 配置spring-jobs.xml (非注解)
  - 任务策略
  - 触发器
  - 调度容器
  - dubbo
    - 扫包 (注册任务类)
    - 应用的名称
    - 注册中心在哪

注解:

dubbo

- 扫包 (注册任务类)



- 应用的名称
  - 注册中心在哪
  - 注解支持
  - ThreadPoolTaskScheduler
5. 启动spring容器，使用main方法启动

## 【讲解】

前面我们已经完成了体检套餐的管理，在新增套餐时套餐的基本信息和图片是分两次提交到后台进行操作的。也就是用户首先将图片上传到七牛云服务器，然后再提交新增窗口中录入的其他信息。如果用户只是上传了图片而没有提交录入的其他信息，此时的图片就变为了垃圾图片，因为在数据库中并没有记录它的存在。此时我们要如何处理这些垃圾图片呢？

解决方案就是通过定时任务组件定时清理这些垃圾图片。如何区分出来哪些图片是垃圾图片？

1. 七牛上的所有图片
2. 数据中的图片
3. 七牛上的减去数据库，剩下的就是垃圾图片了

本章节我们就会基于Quartz定时任务，通过计算redis两个集合的差值找出所有的垃圾图片，就可以将垃圾图片清理掉。

## 2.1. 操作步骤

- (1) 创建maven聚合工程health\_jobs，导入Quartz等相关坐标



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>health_parent</artifactId>
7         <groupId>com.itheima</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>health_jobs</artifactId>
13
14    <properties>
15        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16        <maven.compiler.source>1.8</maven.compiler.source>
17        <maven.compiler.target>1.8</maven.compiler.target>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>com.itheima</groupId>
22            <artifactId>health_interface</artifactId>
23            <version>1.0-SNAPSHOT</version>
24        </dependency>
25        <dependency>
26            <groupId>org.quartz-scheduler</groupId>
27            <artifactId>quartz</artifactId>
28        </dependency>
```

```

29         <dependency>
30             <groupId>org.quartz-scheduler</groupId>
31             <artifactId>quartz-jobs</artifactId>
32         </dependency>
33     </dependencies>
34 </project>

```

## (2) 配置log4j.properties

```

1  ### direct log messages to stdout ###
2  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3  log4j.appender.stdout.Target=System.err
4  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
5  log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
   %m%n
6
7  ### direct messages to file mylog.log ###
8  log4j.appender.file=org.apache.log4j.FileAppender
9  log4j.appender.file.File=c:\\mylog.log
10 log4j.appender.file.layout=org.apache.log4j.PatternLayout
11 log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
   %m%n
12
13 ### set log levels - for more verbose logging change 'info' to 'debug' ###
14
15 log4j.rootLogger=debug, stdout

```

## 2.1.1 遍历7牛上的图片

在QiNiuUitls, 已经有遍历7牛上图片的方法, 如下

 image-20200909161520420

## 2.1.2 接口与实现类

SetmealService接口

```

1  /**
2   * 查出数据库中的所有图片
3   * @return
4   */
5  List<String> findImgs();

```

SetmealServiceImpl实现类

```

1  /**
2   * 查出数据库中的所有图片
3   * @return
4   */
5  @Override
6  public List<String> findImgs() {
7      return setmealDao.findImgs();
8  }

```

## 2.1.3 Dao与映射文件

SetmealDao添加查询所有套餐图片方法

```
1  /**
2   * 查数据中套餐的所有图片
3   * @return
4   */
5  List<String> findImgs();
```

SetmealDao.xml映射文件

```
1  <select id="findImgs" resultType="String">
2      select img from t_setmeal
3  </select>
```

## 2.1.4 清理垃圾图片任务类

```
1  package com.itheima.health.job;
2
3  import com.alibaba.dubbo.config.annotation.Reference;
4  import com.itheima.health.service.SetmealService;
5  import com.itheima.health.utils.QiNiuUtils;
6  import org.springframework.stereotype.Component;
7
8  import java.util.List;
9
10 /**
11  * Description: 定时清理七牛上的垃圾图片
12  * User: Eric
13  */
14 @Component("cleanImgJob")
15 public class CleanImgJob {
16
17     /**
18     * 订阅服务
19     */
20     @Reference
21     private SetmealService setmealService;
22
23     public void cleanImg(){
24         // 查出7牛上的s所有图片
25         List<String> imgIn7Niu = QiNiuUtils.listFiles();
26         // 查出数据库中的所有图片
27         List<String> imgInDb = setmealService.findImgs();
28         // 7牛的-数据库的 imgIn7Niu剩下的就是要删除的
29         imgIn7Niu.removeAll(imgInDb);
30         // 把剩下的图片名转成数组
31         String[] strings = imgIn7Niu.toArray(new String[]{});
32         // 删除7牛上的垃圾图片
33         QiNiuUtils.removeFiles(strings);
34     }
35 }
36
```

## 2.1.5 spring-jobs.xml配置

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans.xsd
7       http://code.alibabatech.com/schema/dubbo
8       http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
9
10    <!-- 指定应用名称 -->
11    <dubbo:application name="health_job" />
12
13    <!--指定服务注册中心地址-->
14    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
15
16    <!-- 任务类要注册 使用dubbo 扫包 -->
17    <dubbo:annotation package="com.itheima.health.job"/>
18
19    <!-- 策略 -->
20    <bean id="jobDetail"
21          class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
22      <property name="targetObject" ref="cleanImgJob"/>
23      <property name="targetMethod" value="cleanImg"/>
24
25      <!--concurrent为true时，多线程来跑任务 异步
26      false: 则为单线程来跑任务 同步
27      -->
28      <property name="concurrent" value="false"/>
29    </bean>
30
31    <!-- 触发器 -->
32    <bean id="trigger"
33          class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
34      <property name="cronExpression" value="0/30 * * * * ?"/>
35      <property name="jobDetail" ref="jobDetail"/>
36    </bean>
37
38    <!-- 调度容器 -->
39    <!-- 调度容器 -->
40    <bean
41          class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
42      <property name="triggers">
43          <list>
44              <ref bean="trigger"/>
45          </list>
46      </property>
47    </bean>
48 </beans>
```

## 2.2 【小结】

1. Quartz的应用场景：后台任务（半夜执行任务）
2. 组件及执行过程：任务调度容器->触发器->任务策略（并发=false）->任务类->任务方法
3. 7牛所有的图片-保存到数据库的图片=要清理的集合

4. 启动的方式：只要能启动spring容器即可

## 3. 预约设置需求分析

---

### 【目标】

- 掌握预约设置的需求

### 【路径】

1. 预约设置需求分析
2. 对应的表结构和实体类

### 【讲解】

### 3.1. 预约设置需求分析

---

前面我们已经完成了检查项管理、检查组管理、套餐管理等。接下来我们需要进行预约设置，其实就是设置每一天的体检预约最大数量。客户可以通过微信端在线预约，在线预约时需要选择体检的时间，使得选择体检时间的已预约人数加1，如果客户选择的时间已经预约满则无法进行预约。

### 3.2. t\_ordersetting表结构

---



orderDate: 预约日期

number: 最多可预约人数

reservations: 已预约人数

### 3.3 【小结】

---

1. 预约设置 就是设置一天最大预约体检人数（number字段）
2. 客户预约的时候，判断是否可预约(reservations<number)，需要更新当前预约人数+1（reservations字段）  
防止超卖
3. 已预约人数不能大于可预约人数(reservations<=number)
4. 为什么要做预约设置：用户体验（人多，等时间长，体验差），成本管控（节假日，工作人员空缺）
5. mysql8 Date日期，{orderDate,jdbcType=DATE}

## 4. Apache POI

---

### 【目标】

了解什么是Apache POI

掌握Apache POI的使用方法

## 【路径】

1. POI介绍
2. POI入门案例
  - (1) 从Excel文件读取数据
  - (2) 向Excel文件写入数据
3. POI工具类的介绍

## 【讲解】

### 4.1. POI介绍

Apache POI是用Java编写的免费开源的跨平台的Java API，Apache POI提供API给Java程序对Microsoft Office格式档案读和写的功能，其中使用最多的就是使用POI操作Excel文件。

jxl: 专门操作Excel

maven坐标:

```
1      <dependency>
2          <groupId>org.apache.poi</groupId>
3          <artifactId>poi</artifactId>
4          <version>3.14</version>
5      </dependency>
6      <dependency>
7          <groupId>org.apache.poi</groupId>
8          <artifactId>poi-ooxml</artifactId>
9          <version>3.14</version>
10     </dependency>
```

POI结构的组件:

- |   |  |
|---|--|
| 1 | HSSF — 提供读写Microsoft Excel XLS格式档案的功能 97-2003的excel，后缀名.xls 工作表大行号65535        |
| 2 | XSSF — 提供读写Microsoft Excel OOXML XLSX格式档案的功能（我们使用） 2007以后.xlsx 工作表大行号1,048,576 |
| 3 | HWPf — 提供读写Microsoft Word DOC格式档案的功能   |
| 4 | HSLF — 提供读写Microsoft PowerPoint格式档案的功能   |
| 5 | HDGF — 提供读Microsoft Visio格式档案的功能   |
| 6 | HPBF — 提供读Microsoft Publisher格式档案的功能   |
| 7 | HSMF — 提供读Microsoft Outlook格式档案的功能   |

我们使用: XSSF - 提供读写Microsoft Excel OOXML XLSX格式档案的功能

### 4.2. 入门案例 【重点】

#### 4.2.1. 从Excel文件读取数据

【需求】

使用POI可以从一个已经存在的Excel文件中读取数据

【路径】

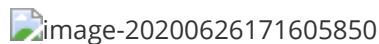
- 1: 创建工作簿对象
- 2: 获得工作表对象 传文件所在
- 3: 遍历工作表对象 获得行对象
- 4: 遍历行对象 获得单元格（列）对象
- 5: 获得数据
- 6: 关闭

### 【讲解】

#### 1. 实现

使用POI可以从一个已经存在的Excel文件中读取数据

创建poidemo工程



pom.xml中添加依赖

```
1 <dependencies>
2   <dependency>
3     <groupId>org.apache.poi</groupId>
4     <artifactId>poi</artifactId>
5     <version>3.14</version>
6   </dependency>
7   <dependency>
8     <groupId>org.apache.poi</groupId>
9     <artifactId>poi-ooxml</artifactId>
10    <version>3.14</version>
11  </dependency>
12  <dependency>
13    <groupId>junit</groupId>
14    <artifactId>junit</artifactId>
15    <version>4.12</version>
16    <scope>test</scope>
17  </dependency>
18 </dependencies>
```

把资料中的read.xlsx复制到D盘根目录下

创建读取excel测试类

```
1 package com.itheima.test;
2
3 import org.apache.poi.hssf.usermodel.HSSFWorkbook;
4 import org.apache.poi.ss.usermodel.Cell;
5 import org.apache.poi.ss.usermodel.Row;
6 import org.apache.poi.ss.usermodel.Sheet;
7 import org.apache.poi.ss.usermodel.Workbook;
8 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
9 import org.junit.Test;
10
11 import java.io.IOException;
12
13 /**
```

```

14  * Description: No Description
15  * User: Eric
16  */
17  public class ReadFormExcel {
18      @Test
19      public void readFromExcel() throws Exception {
20          // 创建工作簿，构造方法文件路径
21          Workbook workbook = new XSSFWorkbook("d:\\read.xlsx");
22          // 获取工作表
23          Sheet sht = workbook.getSheetAt(0);
24          // sht.getPhysicalNumberOfRows(); // 物理行数
25          // sht.getLastRowNum(); // 最后一行的下标
26          // fori遍历时使用getLastRowNum()
27          // 遍历工作表获得行对象
28          for (Row row : sht) {
29              // 遍历行对象获取单元格
30              for (Cell cell : row) {
31                  // 单元格的类型
32                  int cellType = cell.getCellType();
33                  if (Cell.CELL_TYPE_NUMERIC == cellType) {
34                      // 数值类型的单元格
35                      System.out.print(cell.getNumericCellValue() + ",");
36                  } else {
37                      // 从单元格取值
38                      System.out.print(cell.getStringCellValue() + ",");
39                  }
40              }
41              System.out.println();
42          }
43          // 关闭工作簿
44          workbook.close();
45      }
46  }

```

通过上面的入门案例可以看到，POI操作Excel表格封装了几个核心对象：

- 1 XSSFWorkbook：工作簿
- 2 XSSFSheet：工作表
- 3 XSSFRow：行
- 4 XSSFCell：单元格

上面案例是通过遍历工作表获得行，遍历行获得单元格，最终获取单元格中的值。

## 4.2.2. 向Excel文件写入数据

### 【需求】

使用POI可以在内存中创建一个Excel文件并将数据写入到这个文件，最后通过输出流将内存中的Excel文件保存到磁盘

### 【路径】



- 1 1.创建工作簿对象
- 2 2.创建工作表对象
- 3 3.创建行对象
- 4 4.创建列(单元格)对象, 设置内容
- 5 5.通过输出流将workbook对象下载到磁盘

### 【实现】

使用POI可以在内存中创建一个Excel文件并将数据写入到这个文件, 最后通过输出流将内存中的Excel文件保存到磁盘

```
1 package com.itheima.test;
2
3 import org.apache.poi.ss.usermodel.Cell;
4 import org.apache.poi.ss.usermodel.Row;
5 import org.apache.poi.ss.usermodel.Sheet;
6 import org.apache.poi.ss.usermodel.Workbook;
7 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
8 import org.junit.Test;
9
10 import java.io.File;
11 import java.io.FileNotFoundException;
12 import java.io.FileOutputStream;
13
14 /**
15  * Description: No Description
16  * User: Eric
17  */
18 public class WriteExcel {
19
20     @Test
21     public void createExcel() throws Exception {
22         // 创建工作簿, 内存中
23         Workbook workbook = new XSSFWorkbook();
24         // 创建工作表
25         Sheet sht = workbook.createSheet("测试写excel");
26         // 在工作表下创建行
27         Row row = sht.createRow(0); // 行的下标是从0开始
28         // 使用行创建单元格
29         Cell cell = row.createCell(0); // 单元格的下标也是从0开始, 多个单元格合并后
        成为1个单元格
30         // 给单元格赋值
31         // 表头
32         cell.setCellValue("姓名");
33         row.createCell(1).setCellValue("年龄");
34         row.createCell(2).setCellValue("所在地");
35
36         row = sht.createRow(1);
37         row.createCell(0).setCellValue("小明");
38         row.createCell(1).setCellValue(20);
39         row.createCell(2).setCellValue("北京");
40
41         row = sht.createRow(2);
42         row.createCell(0).setCellValue("小李");
43         row.createCell(1).setCellValue(30);
44         row.createCell(2).setCellValue("南京");
45         // 保存工作簿, 持久化本地硬盘里
```

```

46     workbook.write(new FileOutputStream(new
File("d:\\createExcel.xlsx")));
47     // 关闭工作簿
48     workbook.close();
49 }
50 }
51

```

 image-20200626172008115

小结：POI技术（向Excel文件写入数据）

- 创建工作簿的时候, 不需要传入参数(excel不存在的)
- 使用输出流, 输出excel

应用场景：1：从excel中读取数据，写入到数据库（导入）；2：从数据库查询数据，写入到excel文件（报表技术，导出）

### 4.2.3.POI工具类的介绍

将资料中的POIUtils工具类复制到health\_common工程

 image-20200626172843768

修改日期格式的常量修饰符，改为public

 image-20200626173012230

### 【小结】

1. POI apache开源的来操作offices 文档， excel (xls2003 HSSFWorkbook xlsx2007 XSSFWorkbook)
2. 入门
  - 读取excel
    - 创建工作簿，构造方法传文件的全路径
    - 获取工作表
    - 获取行对象，工作表遍历获取行对象
    - 通过行对象的遍历获取单元格
    - 通过单元格获取单元格的值，跟单元格的类型，如果是数值，就得用 getNumericCellValue
    - 关闭工作簿
  - 写excel
    - 创建工作簿，无参构造
    - 创建工作表, 通过下标也可通过名称
    - 创建行 下标从0开始
    - 行对象下创建单元格，从0开始
    - 对单元格赋值
    - 保存工作簿 write到输出流
    - 关闭工作簿
3. POIUtils, 读取上传的excel文件，读取后的值 为List<String[]> String[]代表一行记录

## 5. 批量导入预约设置信息 【重点】

### 【目标】

## 【路径】

### 1. 提供模板文件下载

- health\_web的webapp下创建template目录, 把资料中的""预约设置模板文件 \ordersetting\_template.xlsx" 复制到template目录下

### 2. 上传文件 导入

- ordersetting.html, 上传组件el-upload, action=上传的url处理, name=上传的文件的参数名
- OrderSettingController, POIUtils解析上传的文件, 得List<String[]> 转成List 调用业务导入, 返回结果给页面。
- OrderSettingServiceImpl
  - 遍历数据
  - 通过日期查询预约设置信息
  - 存在
    - 判断更新后的最大预约数是否大于已预约数
      - 小于, 则要报错, 最大可预约数必须大于已预约数
    - 更新最大可预约数
  - 不存在
    - 添加预约设置
  - 事务控制
- OrderSettingDao与映射文件
  - 通过日期查询预约设置信息 select \* from t\_ordersetting where orderDate=#{orderDate}
  - 更新最大可预约数 set number=#{number} where orderDate=#{orderDate}
  - 添加预约设置 insert into
- 上传成功后的方法里, 要提示结果

## 【讲解】

### 【需求】

从Excel读取预约设置的数据, 批量导入到数据库



### 【设计】

预约设置信息对应的数据表为t\_ordersetting, 预约设置操作对应的页面为ordersetting.html

1: t\_ordersetting表结构:



orderDate: 预约日期

number: 最多可预约人数

reservations: 已预约人数

### 【路径】

批量导入预约设置信息操作过程:

第一步、点击模板下载按钮下载Excel模板文件【文件下载】

第二步、将预约设置信息录入到模板文件中

第三步、点击上传文件按钮将录入完信息的模板文件上传到服务器【文件上传】

第四步、通过POI读取上传文件的数据并保存到数据库【poi导入】

2: 将ordersetting.html放置到health\_web中



## 5.1. 前台代码

### 5.1.1. 提供模板文件

资料中已经提供了Excel模板文件ordersetting\_template.xlsx，将文件放在health\_web工程的template目录下



### 5.1.2. 实现模板文件下载-已实现

(1) 为模板下载按钮绑定事件实现模板文件下载

```
1 <el-button style="margin-bottom: 20px;margin-right: 20px" type="primary"
  @click="downloadTemplate()">模板下载</el-button>
```

(2) downloadTemplate()

```
1 //下载模板文件
2 downloadTemplate(){
3     window.location.href="../../template/ordersetting_template.xlsx";
4 },
```

### 5.1.3. 文件上传 - 前端已实现

(1) 使用ElementUI的上传组件实现文件上传并绑定相关事件

```
1 <el-upload action="/ordersetting/upload.do"
2     name="excelFile"
3     :show-file-list="false"
4     :on-success="handleSuccess"
5     :before-upload="beforeUpload">
6     <el-button type="primary">上传文件</el-button>
7 </el-upload>
```

(2) handleSuccess方法：用于显示上传成功或者失败信息。

```

1 //上传成功提示
2 handleSuccess(response, file) {
3     if(response.flag){
4         this.$message({
5             message: response.message,
6             type: 'success'
7         });
8     }else{
9         this.$message.error(response.message);
10    }
11 },

```

(3) methods中的beforeUpload方法：用于校验上传的文件是否是excel文件

```

1 //上传之前进行文件格式校验
2 beforeUpload(file) {
3     // file.type指的是文件的类型，
4     // 部分同学可能得不到application/vnd.ms-excel
5     application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
6     // null ""
7     // 这是跟系统有关的，可以使用判断文件的后缀名方式来替换下面的文件类型校验
8
9     // var fileExt = file.name.substring(file.name.lastIndexOf("."));
10    // if (fileExt==".xls" || fileExt==".xlsx"){
11        //return true;
12    //}
13    const isXLS = file.type === 'application/vnd.ms-excel';
14    if(isXLS){
15        return true;
16    }
17    const isXLSX = file.type === 'application/vnd.openxmlformats-
18    officedocument.spreadsheetml.sheet';
19    if (isXLSX) {
20        return true;
21    }
22    this.$message.error('上传文件只能是xls或者xlsx格式!');
23    return false;
24 },

```

## 5.2. 后台代码

### 5.2.1. Controller

在health\_web工程创建OrderSettingController并提供upload方法

```

1 package com.itheima.health.controller;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.constant.MessageConstant;
5 import com.itheima.health.entity.Result;
6 import com.itheima.health.pojo.OrderSetting;
7 import com.itheima.health.service.OrderSettingService;
8 import com.itheima.health.utils.POIUtils;

```

```

9  import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RestController;
12 import org.springframework.web.multipart.MultipartFile;
13
14 import java.io.IOException;
15 import java.text.SimpleDateFormat;
16 import java.util.ArrayList;
17 import java.util.Date;
18 import java.util.List;
19
20 @RestController
21 @RequestMapping("/orderSetting")
22 public class OrderSettingController {
23
24     @Reference
25     private OrderSettingService orderSettingService;
26
27     // 上传excel文件
28     @PostMapping("/upload")
29     public Result upload(MultipartFile excelFile){
30         try {
31             // 读取excel内容
32             List<String[]> strings = POIUtils.readExcel(excelFile);
33             // 转成List<OrderSetting>
34             List<OrderSetting> orderSettingList = new
ArrayList<OrderSetting>();
35             // 日期格式解析
36             SimpleDateFormat sdf = new
SimpleDateFormat(POIUtils.DATE_FORMAT);
37             Date orderDate = null;
38             OrderSetting os = null;
39             for (String[] dataArr : strings) {
40                 orderDate = sdf.parse(dataArr[0]);
41                 int number = Integer.valueOf(dataArr[1]);
42                 os = new OrderSetting(orderDate, number);
43                 orderSettingList.add(os);
44             }
45             // 调用业务服务
46             orderSettingService.add(orderSettingList);
47             return new Result(true,
MessageConstant.IMPORT_ORDERSETTING_SUCCESS);
48         } catch (Exception e) {
49             e.printStackTrace();
50             return new Result(false,
MessageConstant.IMPORT_ORDERSETTING_FAIL);
51         }
52     }
53 }
54

```

## 5.2.2. 服务接口

创建OrderSettingService服务接口并提供新增方法

```

1  package com.itheima.health.service;
2

```

```

3  import com.itheima.health.exception.HealthException;
4  import com.itheima.health.pojo.OrderSetting;
5
6  import java.util.List;
7
8  /**
9   * Description: No Description
10  * User: Eric
11  */
12  public interface OrderSettingService {
13      /**
14       * 批量导入
15       * @param orderSettingList
16       */
17      void add(List<OrderSetting> orderSettingList) throws HealthException;
18  }
19

```

### 5.2.3. 服务实现类

创建服务实现类OrderSettingServiceImpl并实现新增方法

```

1  package com.itheima.health.service.impl;
2
3  import com.alibaba.dubbo.config.annotation.Service;
4  import com.itheima.health.dao.OrderSettingDao;
5  import com.itheima.health.exception.HealthException;
6  import com.itheima.health.pojo.OrderSetting;
7  import com.itheima.health.service.OrderSettingService;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.transaction.annotation.Transactional;
10
11  import java.util.List;
12
13  /**
14   * Description: No Description
15   * User: Eric
16   */
17  @Service(interfaceClass = OrderSettingService.class)
18  public class OrderSettingServiceImpl implements OrderSettingService {
19
20      @Autowired
21      private OrderSettingDao orderSettingDao;
22
23      @Override
24      @Transactional
25      public void add(List<OrderSetting> orderSettingList) throws
HealthException {
26          // 遍历
27          for (OrderSetting orderSetting : orderSettingList) {
28              // 判断是否存在，通过日期来查询，注意：日期里是否有时分秒，数据库里的日期是
没有时分秒的
29              OrderSetting osInDB =
orderSettingDao.findByOrderDate(orderSetting.getOrderDate());
30              if(null != osInDB){
31                  // 数据库存在这个预约设置，已预约数量不能大于可预约数量
32                  if(osInDB.getReservations() > orderSetting.getNumber()){

```

```

33         throw new HealthException(orderSetting.getOrderDate() +
    " 中已预约数量不能大于可预约数量");
34     }
35     orderSettingDao.updateNumber(orderSetting);
36 }else{
37     // 不存在
38     orderSettingDao.add(orderSetting);
39 }
40 }
41 }
42 }
43

```

## 5.2.4. Dao接口

创建Dao接口OrderSettingDao并提供更新和新增方法

```

1  package com.itheima.health.dao;
2
3  import com.itheima.health.pojo.OrderSetting;
4
5  import java.util.Date;
6
7  /**
8   * Description: No Description
9   * User: Eric
10  */
11 public interface OrderSettingDao {
12     /**
13      * 通过日期来查询预约设置
14      * @param orderDate
15      * @return
16      */
17     OrderSetting findByOrderDate(Date orderDate);
18
19     /**
20      * 更新可预约数量
21      * @param orderSetting
22      */
23     void updateNumber(OrderSetting orderSetting);
24
25     /**
26      * 添加预约设置
27      * @param orderSetting
28      */
29     void add(OrderSetting orderSetting);
30 }
31

```

## 5.2.5. Mapper映射文件

创建Mapper映射文件OrderSettingDao.xml并提供相关SQL

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

```



```

4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6      <mapper namespace="com.itheima.health.dao.OrderSettingDao">
7
8          <select id="findByOrderDate" parameterType="date"
resultType="OrderSetting">
9              select * From t_orderSetting where orderDate = #{orderDate}
10         </select>
11
12         <update id="updateNumber" parameterType="OrderSetting">
13             update t_ordersetting set number=#{number} where orderDate = #
{orderDate}
14         </update>
15
16         <insert id="add" parameterType="orderSetting">
17             insert into t_ordersetting (orderDate,number,reservations)
18             values(#{orderDate},#{number}, #{reservations})
19         </insert>
20     </mapper>

```

查看t\_ordersetting表



## 【小结】

### 1: 前台代码

(1) 提供模板文件 严格要求用户的行为, 因此导入的数据会进入数据库, 如果数据有问题, 将功能受影响

日期 yyyy/MM/dd

(2) 实现模板文件下载, 要下载的文件放到相应webapp/template的目录, 添加下载的连接

(3) 文件上传

### 2: 后台代码

业务:

(1) OrderSettingController.java (Controller)

从excel中读取预约设置信息转成List, 批量导入到数据库

(2) OrderSettingService.java (服务接口)

(3) OrderSettingServiceImpl.java (服务实现类)

1. 如果这个日期的预约信息不存就直接插入即可以
2. 如果存在预约设置信息, 通过日期查询预约设置
  - o 存在 判断已预约人数是否大于要更改的最大预约数
  - o 大于则抛出异常
  - o 小于或等于, 则更新最大预约数

(4) OrderSettingDao.java (Dao接口)

添加的方法, 通过日期查询, 通过日期更新最大预约数量

(5) OrderSettingDao.xml (Mapper映射文件)

添加的方法，通过日期查询，通过日期更新最大预约数量

mysql8 的同学 where orderDate=#{orderDate,jdbcType=DATE}

## 6. 日历展示预约设置信息 【重点】

### 【目标】

日历展示预约设置信息

### 【路径】

1. ordersetting.html把leftobj注释放开，发现前端需要的数据格式

```
1  [  
2  { date: 1, number: 120, reservations: 1 },  
3  { date: 3, number: 120, reservations: 1 },  
4  { date: 6, number: 120, reservations: 1 },  
5  { date: 4, number: 120, reservations: 120 },  
6  { date: 8, number: 120, reservations: 1 }  
7  ]  
8  
9  -> List<Map<String,Integer>>
```

2. this.initData()它是画日历的，且画完日历后再填充数据，给this.leftobj赋值

3. 进入 页面后，在created()里获取当月的数据，提交年-月从后台查询数据，返回的数据绑定this.leftobj里，如果失败要提示

4. OrderSettingController 接收2020-09，调用服务查询，返回List<Map<String,Integer>>，包装到result，返回给页面。

5. OrderSettingService 拼接开始日期（-01号），结束的日期(-31)调用dao查询

6. OrderSettingDao提供日期范围查询

```
1  -- 函数可以放到select后使用，但是尽量不要放到where后使用，可能导致索引失效  
2  select CAST DATE_FORMAT(orderdate,'%d') as SIGNED date,  
3         number,  
4         reservations  
5  from t_ordersetting  
6  where orderDate between '2020-09-01' and '2020-09-31';
```

### 【讲解】

#### 【需求】

前面已经完成了预约设置功能，现在就需要通过日历的方式展示出来每天设置的预约人数。

在页面中已经完成了日历的动态展示，我们只需要查询当前月份的预约设置信息并展示到日历中即可，同时在日历中还需要展示已经预约的人数，效果如下：



## 6.1. 前台代码

### 6.1.1. 使用静态数据调试

为了能够快速看到效果，我们可以先使用静态数据模拟，然后再改为发送ajax请求查询数据库。

实现步骤：

(1) 预约设置数据对应的模型数据为leftobj，在initData方法最后为leftobj模型数据赋值：

```
1  this.leftobj = [  
2    { date: 1, number: 120, reservations: 1 },  
3    { date: 3, number: 120, reservations: 1 },  
4    { date: 4, number: 120, reservations: 120 },  
5    { date: 6, number: 120, reservations: 1 },  
6    { date: 8, number: 120, reservations: 1 }  
7  ];
```

其中date表示日期，number表示可预约人数，reservations表示已预约人数

(2) 使用VUE的v-for标签遍历上面的leftobj模型数据，展示到日历上：

```
1  <template>  
2    <template v-for="obj in leftobj">  
3      <template v-if="obj.date < dayobject.day.getDate()">  
4        <template v-if="obj.number > obj.reservations">  
5          <div class="usual">  
6            <p>可预约{{obj.number}}人</p>  
7            <p>已预约{{obj.reservations}}人</p>  
8          </div>  
9        </template>  
10       <template v-else>  
11         <div class="fulled">  
12           <p>可预约{{obj.number}}人</p>  
13           <p>已预约{{obj.reservations}}人</p>  
14           <p>已满</p>  
15         </div>  
16       </template>  
17     </template>  
18   </template>  
19   <button v-if="dayobject.day > today"  
20     @click="handleOrderSet(dayobject.day)" class="orderbtn">设置</button>  
21 </template>
```



## 6.1.2. 发送ajax获取动态数据

将上面的静态模拟数据去掉，改为发送ajax请求，根据当前页面对应的月份查询数据库获取预约设置信息，将查询结果赋值给leftobj模型数据

(1) 在钩子函数created中添加：

```
1  created: function () { //在vue初始化时调用  
2    this.initData(null);  
3    // 获取当前月份预约设置信息  
4    this.getDataByMonth();  
5  },
```

(2) 创建函数getDataByMonth()

组织this.leftobj的数据，返回List

```
1 //发送ajax请求，根据当前页面对应的月份查询预约设置信息
2 methods: {
3     // 通过月份来获取预约设置信息
4     getDataByMonth(){
5         // 获得月份 yyyy-MM
6         var month = this.currentYear + "-" + (this.currentMonth>9?
7 this.currentMonth:"0" + this.currentMonth);
8         axios.get('/ordersetting/getOrderSettingByMonth.do?month=' +
9 month).then(res => {
10             if(res.data.flag){
11                 this.leftobj = res.data.data;
12             }else{
13                 this.$message.error(res.data.message);
14             }
15         })
16     }
17     ...
18 }
```

(3) 上传文件成功后再调用一下，刷新日历

```
1 //上传成功提示
2 handleSuccess(response, file) {
3     if(response.flag){
4         this.$message({
5             message: response.message,
6             type: 'success'
7         });
8         //上传文件成功后再调用一下，刷新日历
9         this.getDataByMonth();
10    }else{
11        this.$message.error(response.message);
12    }
13    //console.log(response, file);
14 }
```

## 6.2. 后台代码

【路径】

1.OrderSettingController.java

2.OrderSettingServiceImpl.java

```
1 // 1.组织查询Map，dateBegin表示月份开始时间，dateEnd月份结束时间
2 // 2.查询当前月份的预约设置
3 // 3.将List<OrderSetting>，组织成List<Map>
```

3.OrderSettingDao.java

查询当前月份的预约设置

查询当前月份的预约设置（使用between and）

## 6.2.1. Controller

在OrderSettingController中提供getOrderSettingByMonth方法，根据月份查询预约设置信息

```

1  /**
2   * 通过月份查询预约设置信息
3   */
4  @GetMapping("/getOrderSettingByMonth")
5  public Result getOrderSettingByMonth(String month){
6      // 调用服务端查询
7      List<Map<String,Integer>> data =
8      orderSettingService.getOrderSettingByMonth(month);
9      return new Result(true, MessageConstant.GET_ORDERSETTING_SUCCESS,data);
10 }

```

## 6.2.2. 服务接口

在OrderSettingService服务接口中扩展方法getOrderSettingByMonth

```

1  /**
2   * 通过月份查询预约设置信息
3   * @param month
4   * @return
5   */
6  List<Map<String, Integer>> getOrderSettingByMonth(String month);

```

## 6.2.3. 服务实现类

在OrderSettingServiceImpl服务实现类中实现方法getOrderSettingByMonth

```

1  //根据日期查询预约设置数据
2  public List<Map> getOrderSettingByMonth(String month) {
3      // 1.组织查询Map, dateBegin表示月份开始时间, dateEnd月份结束时间
4      String dateBegin = month + "-1";//2019-03-1
5      String dateEnd = month + "-31";//2019-03-31
6      Map map = new HashMap();
7      map.put("dateBegin",dateBegin);
8      map.put("dateEnd",dateEnd);
9      // 2.查询当前月份的预约设置
10     List<OrderSetting> list = orderSettingDao.getOrderSettingByMonth(map);
11     List<Map> data = new ArrayList<>();
12     // 3.将List<OrderSetting>, 组织成List<Map>
13     for (OrderSetting orderSetting : list) {
14         Map orderSettingMap = new HashMap();
15         orderSettingMap.put("date",orderSetting.getOrderDate().getDate());//
16         //获得日期（几号）
17         orderSettingMap.put("number",orderSetting.getNumber());//可预约人数
18
19         orderSettingMap.put("reservations",orderSetting.getReservations());//已预约
20         //人数
21     }
22     return data;
23 }

```

```

18     data.add(orderSettingMap);
19     }
20     return data;
21 }

```

## 6.2.4. Dao接口

在OrderSettingDao接口中扩展方法getOrderSettingByMonth

```

1 List<OrderSetting> getOrderSettingByMonth(Map map);

```

## 6.2.5. Mapper映射文件

在OrderSettingDao.xml文件中扩展SQL

```

1 <!--根据月份查询预约设置信息-->
2 <select id="getOrderSettingByMonth"
3     parameterType="map"
4     resultType="OrderSetting">
5     select * from t_ordersetting where orderDate between #{dateBegin} and #
6     {dateEnd}
7 </select>

```

也可以使用sql语句：SELECT \* FROM t\_ordersetting WHERE orderDate LIKE '2019-08-%'

## 6.3. 初始化下个月，上个月数据



(1) 点击事件

```

1 <div class="choose">
2     <span @click="goCurrentMonth(currentYear,currentMonth)" class="gotoday">
3     今天</span>
4     <span @click="pickPre(currentYear,currentMonth)"></span>
5     <span @click="pickNext(currentYear,currentMonth)"></span>
6 </div>

```

(2) 初始化日期数据（今天、上个月、下个月）：

```

1 //切换到当前月份
2 goCurrentMonth: function (year, month) {
3     var d = new Date();
4     this.initData(this.formatDate(d.getFullYear(), d.getMonth() + 1, 1));
5     this.getDataByMonth();
6 },
7 //向前一个月
8 pickPre: function (year, month) {
9     // setDate(0); 上月最后一天
10    // setDate(-1); 上月倒数第二天
11    // setDate(dx) 参数dx为 上月最后一天的前后dx天
12    var d = new Date(this.formatDate(year, month, 1));
13    d.setDate(0);
14    this.initData(this.formatDate(d.getFullYear(), d.getMonth() + 1, 1));

```

```

15     this.getDataByMonth();
16 },
17 //向后一个月
18 pickNext: function (year, month) {
19     var d = new Date(this.formatDate(year, month, 1));
20     d.setDate(35);////获取指定天之后的日期
21     this.initData(this.formatDate(d.getFullYear(), d.getMonth() + 1, 1));
22     this.getDataByMonth();
23 },

```

分别执行this.getDataByMonth()来获取所属月份数据。

## 【小结】

### 1: 前台代码

#### (1) 使用静态数据调试

预约设置数据对应的模型数据为leftobj, 在initData方法最后为leftobj模型数据赋值:

```

1  this.leftobj = [
2      { date: 1, number: 120, reservations: 1 },
3      { date: 3, number: 120, reservations: 1 },
4      { date: 4, number: 120, reservations: 120 },
5      { date: 6, number: 120, reservations: 1 },
6      { date: 8, number: 120, reservations: 1 }
7  ];

```

其中date表示日期, number表示可预约人数, reservations表示已预约人数

使用VUE的v-for标签遍历上面的leftobj模型数据, 展示到日历上:

```

1  <template>
2      <template v-for="obj in leftobj">
3          <template v-if="obj.date < dayobject.day.getDate()">
4              <template v-if="obj.number > obj.reservations">
5                  <div class="usual">
6                      <p>可预约{{obj.number}}人</p>
7                      <p>已预约{{obj.reservations}}人</p>
8                  </div>
9              </template>
10             <template v-else>
11                 <div class="fulled">
12                     <p>可预约{{obj.number}}人</p>
13                     <p>已预约{{obj.reservations}}人</p>
14                     <p>已满</p>
15                 </div>
16             </template>
17         </template>
18     </template>
19     <button v-if="dayobject.day > today"
20         @click="handleOrderSet(dayobject.day)" class="orderbtn">设置</button>
21 </template>

```

#### (2) 发送ajax获取动态数据

### 2: 后台代码

业务：

- 在页面上，使用日历展示预约设置信息

(1) OrderSettingController.java (Controller)

再来封装符合前端的数据格式

(2) OrderSettingService.java (服务接口)

(3) OrderSettingServiceImpl.java (服务实现类)

(4) OrderSettingDao.java (Dao接口)

(5) OrderSettingDao.xml (Mapper映射文件)

查询当前月份的预约设置(前端传递当前的年-月: 2019-06)

```
1 SELECT * FROM t_ordersetting WHERE orderDate LIKE '2019-06-%'
2 或者
3 SELECT * FROM t_ordersetting WHERE orderDate BETWEEN '2019-06-01' AND '2019-06-31'
```

页面需要的数据，使用List<Map<String,Integer>> [] -> list {}=>实体map [{key:[key2:{}}]} => list<Map<string,List<Map<string,map<String,String>>>>>

```
1 [
2     {date: 1, number: 120, reservations: 1},
3     {date: 3, number: 120, reservations: 1},
4     {date: 4, number: 120, reservations: 120},
5     {date: 6, number: 120, reservations: 1},
6     {date: 8, number: 120, reservations: 1}
7 ]
```

3: 初始化下个月，上个月数据

## 7.基于日历实现预约设置

### 【目标】

日历展示预约设置信息，点击【设置】按钮完成针对当前时间进行设置，设置最多可预约的人数

### 【路径】

1: 前台代码

(1) 为设置按钮绑定事件

(2) 弹出预约设置窗口，点击确定后发送ajax请求，传当前设置的日期，用户填写的数量，封装json{orderDate:, number:}对象中，再提交json对象

2: 后台代码

业务：

- 在页面上，基于日历实现预约设置

(1) OrderSettingController.java (Controller) 封装到OrderSetting对象中



(2) OrderSettingService.java (服务接口)

(3) OrderSettingServiceImpl.java (服务实现类)

通过日期判断预约设置是否存在？

- 存在：
  - 判断已经预约的人数是否大于要更新的最大可预约人数，`reverations > 传进来的number数量`，则不能更新，要报错
  - `reverations <= 传进来的number数量`，则要更新最大可预约数量
- 不存在：
  - 添加预约设置信息

## 【讲解】

### 【需求】

本章节要完成的功能为通过点击日历中的设置按钮来设置对应日期的可预约人数。效果如下：



## 7.1. 前台代码

### 7.1.1. 为设置按钮绑定事件

(1) 为日历中的设置按钮绑定单击事件，当前日期作为参数

```
1 <button v-if="dayobject.day > today" @click="handleOrderSet(dayobject.day)"
  class="orderbtn">设置</button>
```

(2) handleOrderSet()方法

```
1 //预约设置
2 handleOrderSet(day){
3     alert(day);
4 },
```

### 7.1.2. 弹出预约设置窗口并发送ajax请求

完善handleOrderSet方法，弹出预约设置窗口，用户点击确定按钮则发送ajax请求

参考：\$prompt



```
1 //预约设置
2 handleOrderSet(day){
3     //alert(day);
4     this.$prompt('请输入最大可预约数量', '提示', {
5         confirmButtonText: '确定',
6         cancelButtonText: '取消',
7         inputPattern: /^[1-9][0-9]{0,2}$/,
```

```

8      inputErrorMessage: '数量不正确，数量必须为正整数'
9    }).then(({ value }) => {
10      // value就是输入的值
11      var orderDate = this.formatDate2(day);
12      // 要提交的数据，必须包含设置日期与数值，submitData后用OrderSetting接收，
13      // 必须保证submitData中的属性名(key)与OrderSetting实体类中的属性名一致
14      var submitData = {orderDate: orderDate, number: value}
15
16      axios.post('/ordersetting/editNumberByDate.do', submitData).then(res=>{
17        this.$message({
18          message: res.data.message,
19          type: res.data.flag?"success":"error"
20        })
21        // 成功则要刷新日历，加载数据即可
22        if(res.data.flag){
23          this.getDataByMonth(day.getFullYear() + "-" +
24            (day.getMonth() + 1));
25        }
26      }).catch(() => {
27      });
28    }
29
30    在 methods中添加方法 formatDate2
31    // 返回 类似 2020-08-02 格式的字符串
32    formatDate: function (year, month, day) {
33      var y = year;
34      var m = month;
35      if (m < 10) m = "0" + m;
36      var d = day;
37      if (d < 10) d = "0" + d;
38      return y + "-" + m + "-" + d
39    },
40    // 返回 类似 2020-08-02 格式的字符串
41    formatDate2: function (date) {
42      var y = date.getFullYear();
43      var m = date.getMonth() + 1;
44      if (m < 10) m = "0" + m;
45      var d = date.getDate();
46      if (d < 10) d = "0" + d;
47      return y + "-" + m + "-" + d
48    }

```

## 7.2. 后台代码

### 7.2.1. Controller

在OrderSettingController中提供方法editNumberByDate

```

1  /**
2   * 基于日历的预约设置
3   */
4  @PostMapping("/editNumberByDate")
5  public Result editNumberByDate(@RequestBody OrderSetting orderSetting){
6      // 调用服务更新
7      orderSettingService.editNumberByDate(orderSetting);
8      return new Result(true, MessageConstant.ORDERSETTING_SUCCESS);
9  }

```

## 5.2.2. 服务接口

在OrderSettingService服务接口中提供方法editNumberByDate

```

1  // 通过日期修改可预约人数，这里要抛出自定义的异常
2  void editNumberByDate(OrderSetting orderSetting) throws HealthException;

```

## 7.2.3. 服务实现类

在OrderSettingServiceImpl服务实现类中实现editNumberByDate

```

1  /**
2   * 通过日期设置预约信息
3   * @param orderSetting
4   */
5  @Override
6  public void editNumberByDate(OrderSetting orderSetting) throws
    HealthException {
7      //通过日期判断预约设置是否存在?
8      OrderSetting os =
        orderSettingDao.findByOrderDate(orderSetting.getOrderDate());
9      //- 存在:
10     if(null != os) {
11         // 判断已经预约的人数是否大于要更新的最大可预约人数, reverations > 传进来的
            number数量, 则不能更新, 要报错
12         if(orderSetting.getNumber() < os.getReservations()){
13             // 已经预约的人数高于最大预约人数, 不允许
14             throw new HealthException("最大预约人数不能小已预约人数!");
15         }
16         // reverations <= 传进来的number数量, 则要更新最大可预约数量
17         orderSettingDao.editNumberByOrderDate(orderSetting);
18     }else {
19         //- 不存在:
20         // - 添加预约设置信息
21         orderSettingDao.add(orderSetting);
22     }
23 }

```

## 【小结】

1: 前台代码

- (1) 为设置按钮绑定事件
- (2) 弹出预约设置窗口, 封装到json对象再发送ajax请求

## 2: 后台代码

业务:

- 在页面上, 基于日历实现预约设置

(1) OrderSettingController.java (Controller) 用Ordersetting接收json

(2) OrderSettingService.java (服务接口)

(3) OrderSettingServiceImpl.java (服务实现类)

# 8. 移动端开发

## 8.1. 移动端开发方式

### 【目标】

了解常见的移动端开发方式

### 【路径】

- 基于手机API开发
- 基于手机浏览器开发
- 混合开发

随着移动互联网的兴起和手机的普及, 目前移动端应用变得愈发重要, 成为了各个商家的必争之地。例如, 我们可以使用手机购物、支付、打车、玩游戏、订酒店、购票等, 以前只能通过PC端完成的事情, 现在通过手机都能够实现, 而且更加方便, 而这些都需要移动端开发进行支持, 那如何进行移动端开发呢?

移动端开发主要有三种方式:

- 1、基于手机API开发 (原生APP apk, ios) 用手机来安装 C/S
- 2、基于手机浏览器开发 (移动web) B/S
- 3、混合开发 (混合APP) 相当于浏览器 安装app+web

### 【讲解】

#### 8.1.1. 基于手机API开发

手机端使用手机API, 例如使用Android、ios 等进行开发, 服务端只是一个数据提供者。手机端请求服务端获取数据 (json、xml格式) 并在界面进行展示。这种方式相当于传统开发中的C/S模式, 即需要在手机上安装一个客户端软件。

这种方式需要针对不同的手机系统分别进行开发, 目前主要有以下几个平台:

- 1、苹果ios系统版本, 开发语言是Objective-C
- 2、安卓Android系统版本, 开发语言是Java
- 3、微软Windows phone系统版本, 开发语言是C#
- 4、塞班symbian系统版本, 开发语言是C++

此种开发方式举例: 手机淘宝、抖音、今日头条、大众点评

### 8.1.2. 基于手机浏览器开发

生存在浏览器中的应用，基本上可以说是触屏版的网页应用。这种开发方式相当于传统开发中的B/S模式，也就是手机上不需要额外安装软件，直接基于手机上的浏览器进行访问。这就需要我们编写的html页面需要根据不同手机的尺寸进行自适应调节，目前比较流行的是html5开发。除了直接通过手机浏览器访问，还可以将页面内嵌到一些应用程序中，例如通过微信公众号访问html5页面。

这种开发方式不需要针对不同的手机系统分别进行开发，只需要开发一个版本，就可以在不同的手机上正常访问。

本项目会通过将我们开发的html5页面内嵌到微信公众号这种方式进行开发。

### 8.1.3. 混合开发

是半原生半Web的混合类App。需要下载安装，看上去类似原生App，访问的内容是Web网页。其实就是把HTML5页面嵌入到一个原生容器里面。

#### 【小结】

##### 1. 基于手机API开发: 原生App

- 优点: 用户体验特别好, 调用手机硬件方便。获取系统权限, 获取个人隐私 开发商
- 缺点: 不跨平台, 每一个系统都需要独立开发, 浪费成本, 每次重新, 都要下载新软件, 浪费流量. CS

##### 2. 基于手机浏览器开发 : webApp B/S

- 优点: 跨平台, 只需要开发一套, 节约成本, 服务器更新时客户端不需要更新, 都有内置浏览器。
- 缺点: 用户体验不是特别好, 调用手机硬件不方便。无法获取更新的系统权限及个人信息

##### 3. 混合开发: 原生+ webApp

框架底层: APP 获取更多的系统信息与个人信息....

页面展示 : WebApp, B/S

## 8.2. 微信公众号开发

#### 【目标】

了解微信公众号开发过程

#### 【路径】

- 帐号分类
- 注册帐号
- 自定义菜单
- 上线要求

#### 【讲解】

要进行微信公众号开发，首先需要访问微信公众平台，官网：<https://mp.weixin.qq.com/>。

### 8.2.1. 帐号分类

在微信公众平台可以看到，有四种帐号类型：服务号、订阅号、小程序、企业微信（原企业号）。





本项目会选择订阅号这种方式进行公众号开发。

### 8.2.2. 注册帐号

要开发微信公众号，首先需要注册成为会员，然后就可以登录微信公众平台进行自定义菜单的设置。

注册页面：[https://mp.weixin.qq.com/cgi-bin/registermidpage?action=index&lang=zh\\_CN&token=](https://mp.weixin.qq.com/cgi-bin/registermidpage?action=index&lang=zh_CN&token=)



选择订阅号进行注册：

使用邮箱注册。



输入邮箱、邮箱验证码、密码、确认密码等按照页面流程进行注册

### 8.2.3. 自定义菜单

注册成功后就可以使用注册的邮箱和设置的密码进行登录，登录成功后点击左侧“自定义菜单”进入自定义菜单页面



在自定义菜单页面可以根据需求创建一级菜单和二级菜单，其中一级菜单最多可以创建3个，每个一级菜单下面最多可以创建5个二级菜单。每个菜单由菜单名称和菜单内容组成，其中菜单内容有3种形式：发送消息（视频、语言、文字、图片）、跳转网页、跳转小程序。

跳转页面：<http://www.helloitcast.xin/>



关注微信公众号

1：搜索黑马健康



### 8.2.4. 上线要求（了解）

如果是个人用户身份注册的订阅号，则自定义菜单的菜单内容不能进行跳转网页，因为个人用户目前不支持微信认证，而跳转网页需要微信认证之后才有限权。

如果是企业用户，首先需要进行微信认证，通过后就可以进行跳转网页了，跳转网页的地址要求必须有域名并且域名需要备案通过。

## 【小结】

1. 注册的是订阅号 个人的。

- 公众号，不要选到小程序（邮件地址只能用一种，要么小程序，要么公众号）
- 自定义菜单（企业：跳转网页(url:<http://xxx.com>, xxx域名要备案（企业）-1500。把项目部署上去)
  - 项目公网部署

- 要ECS服务器（华为、腾讯、阿里、、、）要部署项目的, Centos7, mysql,tomcat, jdk
- 申请域名, 域名解析（域名映射服务器的ip地址）
- ECS服务开放安全设置（mysql端口（改）, 80端口, 进出站）
- 域名备案（企业资料, 15个工作日, 找平台人做）最长30天的时间可以访问, 过后会封网站。快的一个星期内就给封
- 也有一海外服务器是不需要备案.....

## 2. 上线要求

- 企业认证
- 设置菜单的网址需要域名备案