

# Day23

---

[使用 Redis 缓存完善文件上传](#)

[图片上传的问题](#)

[体检套餐分页查询列表](#)

[前端页面](#)

[定义分页相关模型数据](#)

[定义分页方法](#)

[完善分页方法执行时机](#)

[后台代码](#)

[控制层](#)

[服务接口](#)

[服务实现类](#)

[Dao接口](#)

[Mapper 映射文件](#)

[什么是 Quartz](#)

[Quartz Demo 搭建](#)

[Quartz核心详解](#)

[Job 和 JobDetail](#)

[JobExecutionContext](#)

[JobDataMap](#)

[Trigger、SimpleTrigger、CronTrigger](#)

[Trigger](#)

[SimpleTrigger](#)

[CronTrigger](#)

[在线生成Cron表达式工具](#)

[Quartz 和 Spring 整合](#)

[1. 创建 maven 工程 quartzdemo](#)

[2. 自定义一个Job](#)

[3. 新增 Spring 配置文件 spring-jobs.xml](#)

#### 4. 编写main方法进行测试

#### 定时清理垃圾图片

##### 概述

##### 操作步骤

1. 创建新的子工程 seehope-health-jobs
2. 配置web.xml
3. 配置log4j.properties
4. 配置applicationContext-redis.xml
5. 配置applicationContext-jobs.xml
6. 创建ClearImgJob定时任务类
  - 6.1 创建 Redis 常量类
  - 6.2 创建定时任务类

## 使用 Redis 缓存完善文件上传

### 图片上传的问题

前面我们已经完成了文件上传，将图片存储在了七牛云服务器中。

但是这个过程存在一个问题，就是如果用户只上传了图片而没有最终保存套餐信息到我们的数据库，这时我们上传的图片就变为了垃圾图片。

对于这些垃圾图片我们需要定时清理来释放磁盘空间。

这就需要我们能够区分出来哪些是垃圾图片，哪些不是垃圾图片。

一个方案就是利用 redis 来保存图片名称，具体做法为：

- 1、当用户上传图片后，将图片名称保存到 redis 的一个 Set 集合中，例如集合名称为 setmealPicResources
- 2、当用户添加套餐后，将图片名称保存到 redis 的另一个 Set 集合中，例如集合名称为 setmealPicDbResources

3、计算 setmealPicResources 集合与 setmealPicDbResources 集合的差值，结果就是垃圾图片的名称集合，清理这些图片即可

现在我们可以完成前面两步。

实现步骤：

1. 在health\_backend项目中提供Spring配置文件spring-redis.xml

```
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         xsi:schemaLocation="http://www.springframework.org/schema/beans
5                             http://www.springframework.org/schema/beans/spring-
6                             beans.xsd">
7
8      <!--Jedis连接池的相关配置-->
9      <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
10         <property name="maxTotal" value="200"/>
11         <property name="maxIdle" value="50"/>
12         <property name="testOnBorrow" value="true"/>
13         <property name="testOnReturn" value="true"/>
14     </bean>
15     <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
16         <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
17         <constructor-arg name="host" value="127.0.0.1" />
18         <constructor-arg name="port" value="6379" type="int" />
19         <constructor-arg name="timeout" value="30000" type="int" />
20     </bean>
21 </beans>
```

2. 在health\_common工程中提供Redis常量类

```
1 package com.rushuni.constant;
2
3 public class RedisConstant {
4     /**
5      * 套餐图片所有图片名称
6      */
7     public static final String SETMEAL_PIC_RESOURCES = "setmealPicResources";
8     /**
9      * 套餐图片保存在数据库中的图片名称
10    */
11    public static final String SETMEAL_PIC_DB_RESOURCES =
12        "setmealPicDbResources";
13 }
```

3. 完善SetmealController，在文件上传成功后将图片名称保存到redis集合中

```
1  @Autowired
2  private JedisPool jedisPool;
3
4  /**
5   * 图片上传
6   * @param imgFile
7   * @return
8   */
9  @RequestMapping("/upload")
10 public Result upload(@RequestParam("imgFile") MultipartFile imgFile){
11     try{
12         // 获取原始文件名
13         String originalFilename = imgFile.getOriginalFilename();
14         int lastIndexOf = originalFilename.lastIndexOf(".");
15         // 获取文件后缀
16         String suffix = originalFilename.substring(lastIndexOf - 1);
17         // 使用UUID随机产生文件名称, 防止同名文件覆盖
18         String fileName = UUID.randomUUID() + suffix;
19         QiniuUtils.upload2Qiniu(imgFile.getBytes(), fileName);
20         // 图片上传成功
21         Result result = new Result(true, MessageConstant.PIC_UPLOAD_SUCCESS);
22         // 将上传图片名称存入Redis, 基于Redis的Set集合存储
23
24         jedisPool.getResource().sadd(RedisConstant.SETMEAL_PIC_RESOURCES, fileName);
25         result.setData(fileName);
26         return result;
27     }catch (Exception e){
28         e.printStackTrace();
29         //图片上传失败
30         return new Result(false, MessageConstant.PIC_UPLOAD_FAIL);
31     }
32 }
```

4. 在health\_service\_provider项目中提供Spring配置文件spring-redis.xml

```

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       xsi:schemaLocation="http://www.springframework.org/schema/beans
5                           http://www.springframework.org/schema/beans/spring-
6                           beans.xsd">
7     <!--Jedis连接池的相关配置-->
8     <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
9         <property name="maxTotal" value="200"/>
10        <property name="maxIdle" value="50"/>
11        <property name="testOnBorrow" value="true"/>
12        <property name="testOnReturn" value="true"/>
13    </bean>
14    <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
15        <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
16        <constructor-arg name="host" value="127.0.0.1" />
17        <constructor-arg name="port" value="6379" type="int" />
18        <constructor-arg name="timeout" value="30000" type="int" />
19    </bean>
20 </beans>

```

5. 完善SetmealServiceImpl服务类，在保存完成套餐信息后将图片名称存储到redis集合中

```

1 @Autowired
2 private JedisPool jedisPool;
3
4 /**
5  * 新增套餐
6  * @param setmeal
7  * @param checkgroupIds
8  */
9 @Override
10 public void add(Setmeal setmeal, Integer[] checkgroupIds) {
11     setmealDao.add(setmeal);
12     if(checkgroupIds != null && checkgroupIds.length > 0){
13         //绑定套餐和检查组的多对多关系
14         setSetmealAndCheckGroup(setmeal.getId(),checkgroupIds);
15         //将图片名称保存到Redis集合中
16         String fileName = setmeal.getImg();
17
18         jedisPool.getResource().sadd(RedisConstant.SETMEAL_PIC_DB_RESOURCES,fileName)
19         ;
20     }
21 }

```

# 体检套餐分页查询列表

## 前端页面

### 定义分页相关模型数据

```
1 // 分页相关模型数据
2 pagination: {
3   // 当前页码
4   currentPage: 1,
5   // 每页显示的记录数
6   pageSize: 10,
7   // 总记录数
8   total: 0,
9   // 查询条件
10  queryString: null
11 },
```

JavaScript | 复制代码

### 定义分页方法

在页面中提供了findPage方法用于分页查询，为了能够在setmeal.html页面加载后直接可以展示分页数据，可以在VUE提供的钩子函数created中调用findPage方法

```
1 // 钩子函数，VUE对象初始化完成后自动执行
2 created() {
3   // VUE对象初始化完成后调用分页查询方法，完成分页查询
4   this.findPage();
5 },
6 // 分页查询
7 findPage() {
8   // 分页参数
9   var param = {
10    // 页码
11    currentPage: this.pagination.currentPage,
12    // 每页显示的记录数
13    pageSize: this.pagination.pageSize,
14    // 查询条件
15    queryString: this.pagination.queryString
16  };
17  // 请求后台
18  axios.post("/setmeal/findPage.do", param).then((response) => {
19    // 为模型数据赋值，基于VUE的双向绑定展示到页面
20    this.dataList = response.data.rows;
21    this.pagination.total = response.data.total;
22  });
23 }
```

## 完善分页方法执行时机

除了在 created 钩子函数中调用 findPage 方法查询分页数据之外，当用户点击查询按钮或者点击分页条中的页码时也需要调用 findPage 方法重新发起查询请求。

为查询按钮绑定单击事件，调用 findPage 方法

```
1 <el-button @click="findPage()" class="dalfBut">查询</el-button>
```

为分页条组件绑定current-change事件，此事件是分页条组件自己定义的事件，当页码改变时触发，对应的处理函数为handleCurrentChange



```

1 <el-pagination
2     class="pagiantion"
3     @current-change="handleCurrentChange"
4     :current-page="pagination.currentPage"
5     :page-size="pagination.pageSize"
6     layout="total, prev, pager, next, jumper"
7     :total="pagination.total">
8 </el-pagination>

```

定义handleCurrentChange方法

```

1 // 切换页码
2 handleCurrentChange(currentPage) {
3     // currentPage为切换后的页码
4     this.pagination.currentPage = currentPage;
5     this.findPage();
6 }

```

## 后台代码

### 控制层

在 SetmealController 中增加分页查询方法

```

1 /**
2  * 分页查询
3  * @param queryPageBean
4  * @return
5  */
6 @RequestMapping("/findPage")
7 public PageResult findPage(@RequestBody QueryPageBean queryPageBean){
8     return setmealService.pageQuery(queryPageBean);
9 }

```

### 服务接口

在 SetmealService 服务接口中扩展分页查询方法

```

1  /**
2   * 分页查询列表
3   * @param queryPageBean
4   * @return
5   */
6  PageResult pageQuery(QueryPageBean queryPageBean);

```

## 服务实现类

在 SetmealServiceImpl 服务实现类中实现分页查询方法，基于 Mybatis分页助手插件实现分页

```

1  /**
2   * 分页查询列表
3   * @param queryPageBean
4   * @return
5   */
6  @Override
7  public PageResult pageQuery(QueryPageBean queryPageBean) {
8      Integer currentPage = queryPageBean.getCurrentPage();
9      Integer pageSize = queryPageBean.getPageSize();
10     String queryString = queryPageBean.getQueryString();
11     PageHelper.startPage(currentPage, pageSize);
12     Page<Setmeal> page = setmealDao.findByCondition(queryString);
13     return new PageResult(page.getTotal(), page.getResult());
14 }

```

## Dao接口

在 SetmealDao 接口中扩展分页查询方法

```

1  /**
2   * 根据条件进行查询
3   * @param queryString
4   * @return
5   */
6  Page<Setmeal> findByCondition(String queryString);

```

## Mapper 映射文件

在 SetmealDao.xml 文件中增加 SQL 定义

```
1 <!--根据条件进行查询-->
2 <select id="findByCondition" parameterType="string"
  resultType="com.rushuni.pojo.Setmeal">
3   select * from t_setmeal
4   <if test="value != null and value != '' and value.length > 0">
5     where code = #{value} or name = #{value} or helpCode = #{value}
6   </if>
7 </select>
```

## 什么是 Quartz

Quartz 是 OpenSymphony 开源组织在 Job scheduling 领域又一个开源项目，完全由 Java 开发，可以用来执行定时任务，类似于 `java.util.Timer`。但是相较于 `Timer`，Quartz 增加了很多功能：

- 持久性作业 – 就是保持调度定时的状态；
- 作业管理 – 对调度作业进行有效的管理；

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

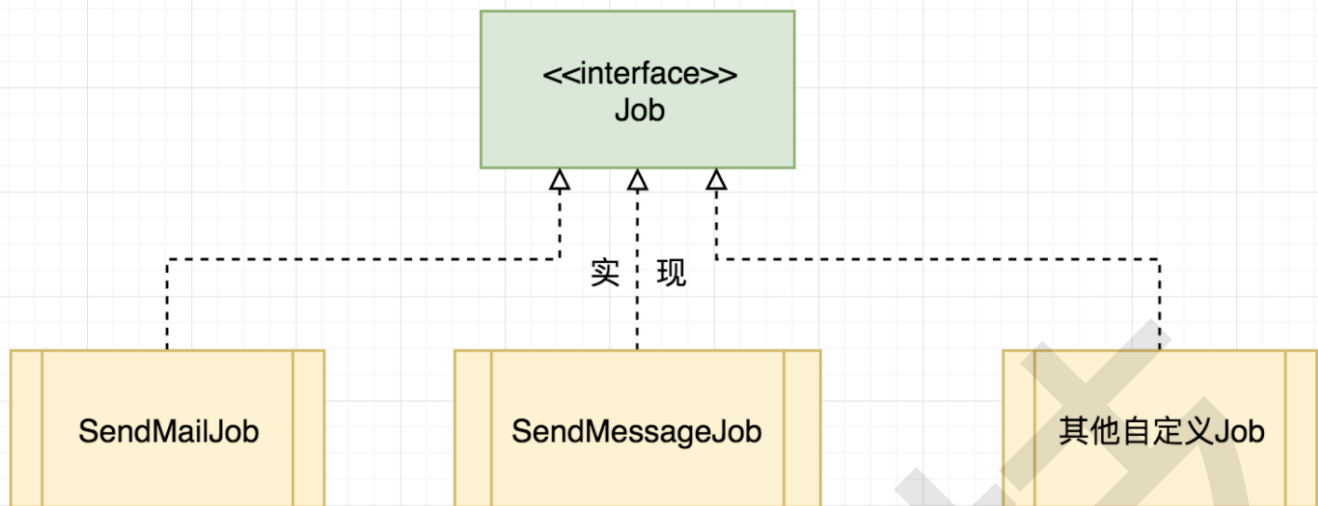
大部分公司都会用到定时任务这个功能。

拿火车票购票来说，当你下单后，后台就会插入一条待支付的 `task(job)`，一般是 30 分钟，超过 30min 后就会执行这个 `job`，去判断你是否支付，未支付就会取消此次订单；

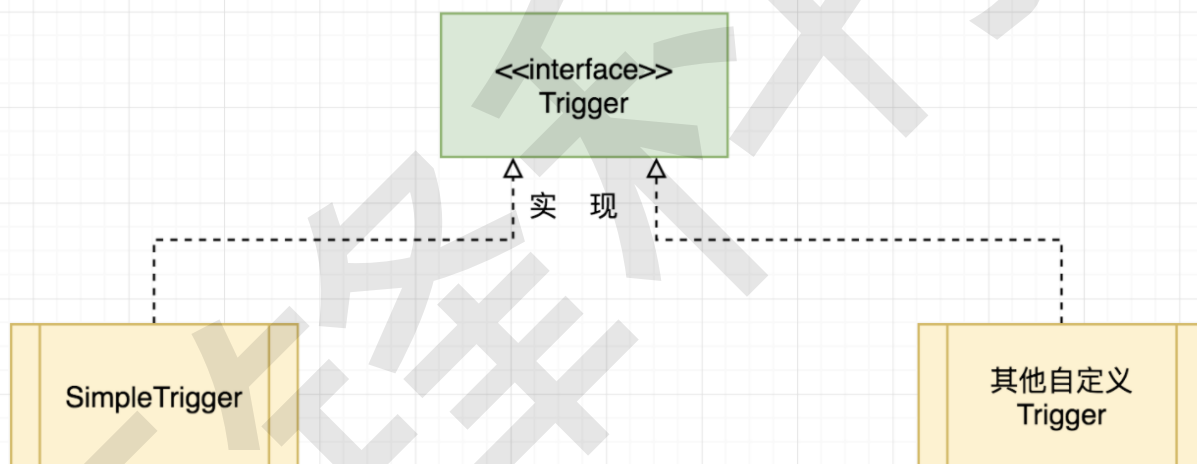
当你支付完成之后，后台拿到支付回调后就会再插入一条待消费的 `task (job)`，`Job` 触发日期为火车票上的出发日期，超过这个时间就会执行这个 `job`，判断是否使用等。

在我们实际的项目中，当 `Job` 过多的时候，肯定不能人工去操作，这时候就需要一个任务调度框架，帮我们自动去执行这些程序。那么该如何实现这个功能呢？

1. 首先我们需要定义实现一个定时功能的接口，我们可以称之为 `Task`（或`Job`），如定时发送邮件的 `task (Job)`，重启机器的 `task (Job)`，优惠券到期发送短信提醒的 `task (Job)`，实现接口如下：

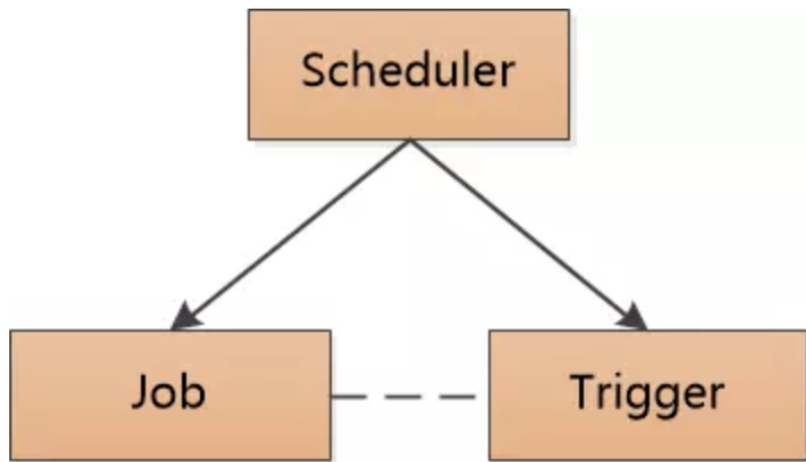


2. 有了任务之后，还需要一个能够实现触发任务去执行的触发器，触发器 Trigger 最基本的功能是指定 Job 的执行时间，执行间隔，运行次数等。



最基本的触发器，可以自定义Job执行时间，执行次数等

3. 有了 Job 和 Trigger 后，怎么样将两者结合起来呢？即怎样指定 Trigger 去执行指定的 Job 呢？这时需要一个 Schedule，来负责这个功能的实现。



上面三个部分就是 Quartz 的基本组成部分：

- 调度器：Scheduler
- 任务：JobDetail
- 触发器：Trigger，包括 SimpleTrigger 和 CronTrigger

## Quartz Demo 搭建

下面来利用 Quartz 搭建一个最基本的 Demo。

### 1. 导入依赖的jar包

```
1 <dependency>
2   <groupId>org.quartz-scheduler</groupId>
3   <artifactId>quartz</artifactId>
4   <version>2.3.2</version>
5 </dependency>
```

XML

复制代码

### 2. 新建一个能够打印任意内容的Job

```
1 package com.rushuni.quartz_demo;
2
3 import org.quartz.Job;
4 import org.quartz.JobExecutionContext;
5 import org.quartz.JobExecutionException;
6
7 import java.time.LocalDate;
8 import java.time.LocalTime;
9 import java.util.Random;
10
11 /**
12  * 打印任务
13  * @author rushuni
14  * @date 2021/08/09
15  */
16 public class PrintWordsJob implements Job {
17
18     @Override
19     public void execute(JobExecutionContext jobExecutionContext) throws
20     JobExecutionException {
21         System.out.println("PrintWordsJob start at: " + LocalDate.now() + " "
22         + LocalTime.now() +
23         ", prints: Hello Job-" + new Random().nextInt(100));
24     }
25 }
```

创建Schedule, 执行任务

```
1 package com.rushuni.quartz_demo;
2
3 import org.quartz.*;
4 import org.quartz.impl.StdSchedulerFactory;
5
6 import java.util.concurrent.TimeUnit;
7
8 /**
9  * Quartz Demo
10  * @author rushuni
11  * @date 2021/08/09
12  */
13 public class MyScheduler {
14     public static void main(String[] args) throws SchedulerException,
15     InterruptedException {
16         // 1. 创建调度器 Scheduler
17         SchedulerFactory schedulerFactory = new StdSchedulerFactory();
18         Scheduler scheduler = schedulerFactory.getScheduler();
19         // 2. 创建 JobDetail 实例, 并与 PrintWordsJob 类绑定(Job执行内容)
20         JobDetail jobDetail =
21         JobBuilder.newJob(PrintWordsJob.class).withIdentity("job1",
22         "group1").build();
23         // 3. 构建 Trigger 实例, 每隔 1s 执行一次
24         Trigger trigger =
25         TriggerBuilder.newTrigger().withIdentity("trigger1", "triggerGroup1")
26         // 立即生效
27         .startNow()
28         .withSchedule(SimpleScheduleBuilder.simpleSchedule()
29         // 每隔1s执行一次
30         .withIntervalInSeconds(1)
31         // 一直执行
32         .repeatForever()).build();
33         // 4. 执行
34         scheduler.scheduleJob(jobDetail, trigger);
35         System.out.println("-----scheduler start ! -----");
36         scheduler.start();
37
38         // 5. 睡眠
39         TimeUnit.MINUTES.sleep(1);
40         scheduler.shutdown();
41         System.out.println("-----scheduler shutdown ! -----");
42     }
43 }
```

# Quartz核心详解

看一下Quartz框架中的几个重要参数：

- Job 和 JobDetail
- JobExecutionContext
- JobDataMap
- Trigger、SimpleTrigger、CronTrigger

## Job 和 JobDetail

Job 是 Quartz 中的一个接口，接口下只有 execute 方法，在这个方法中编写业务逻辑。

接口中的源码：

```
package org.quartz;  
  
public interface Job {  
    void execute(JobExecutionContext var1) throws JobExecutionException;  
}
```

JobDetail 用来绑定 Job，为 Job 实例提供许多属性：

- name
- group
- jobClass
- jobDataMap

JobDetail 绑定指定的 Job。

每次 Scheduler 调度执行一个 Job 的时候，首先会拿到对应的 Job。

然后创建该Job 实例，再去执行 Job 中的 execute() 的内容。

任务执行结束后，关联的 Job 对象实例会被释放，且会被JVM GC 清除。

为什么设计成 JobDetail + Job，不直接使用 Job？

JobDetail 定义的是任务数据，而真正的执行逻辑是在 Job 中。



这是因为任务是有可能并发执行，如果 Scheduler 直接使用 Job，就会存在对同一个 Job 实例并发访问的问题。而 JobDetail & Job 方式，Scheduler 每次执行，都会根据 JobDetail 创建一个新的 Job 实例，这样就可以规避并发访问的问题。

## JobExecutionContext

JobExecutionContext 中包含了 Quartz 运行时的环境以及 Job 本身的详细数据信息。

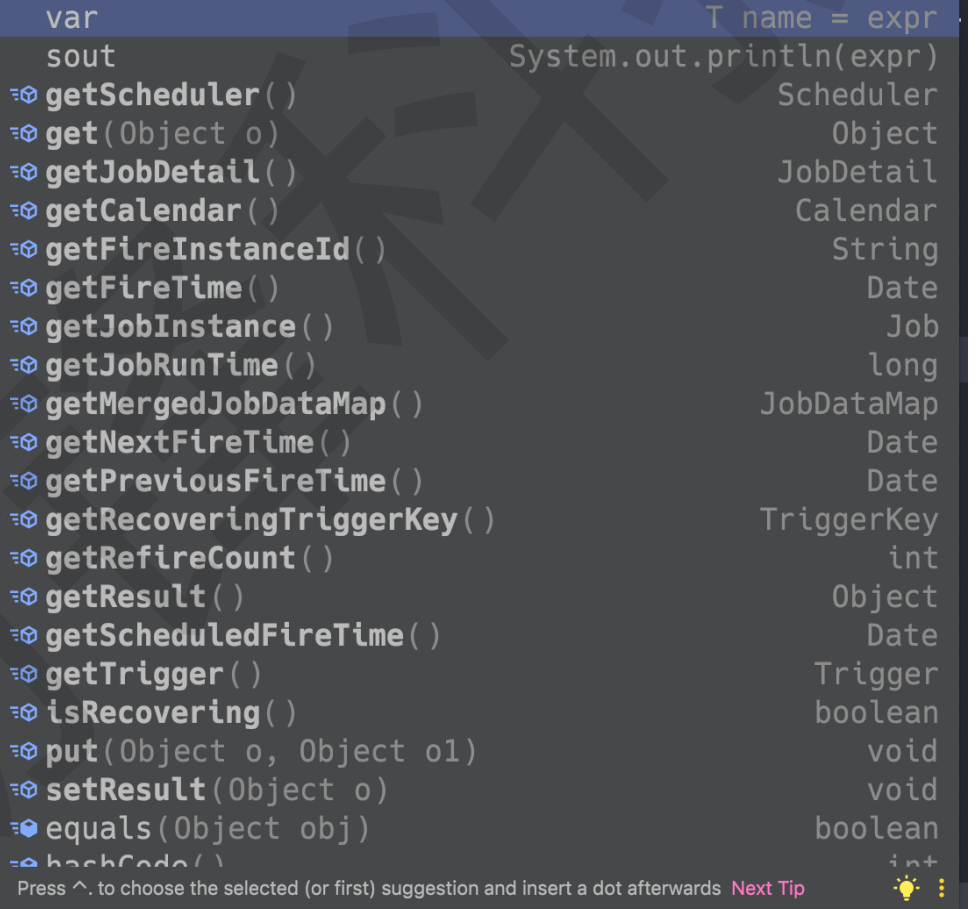
当 Schedule 调度执行一个 Job 的时候，就会将 JobExecutionContext 传递给该 Job 的 execute() 中，Job 就可以通过 JobExecutionContext 对象获取信息。

主要信息有：

```
@Override
public void execute(JobExecutionContext jobExecutionContext) throws JobExecutionException {
    jobExecutionContext.
    System.out.println("JobExecutionContext: ");
    // ...
}

198911, prints: Hello
199192, prints: Hello
198449, prints: Hello
198568, prints: Hello
198832, prints: Hello
199219, prints: Hello
198800, prints: Hello
198833, prints: Hello
198785, prints: Hello

Killed by signal 2: SIGINT
```



## JobDataMap

JobDataMap 实现了 JDK 的 Map 接口，可以以 Key-Value 的形式存储数据。

JobDetail、Trigger 都可以使用 JobDataMap 来设置一些参数或信息，

Job 执行 execute() 方法的时候, JobExecutionContext 可以获取到 JobExecutionContext 中的信息:  
如:

```
Java 复制代码
1 JobDetail jobDetail2 = JobBuilder.newJob(PrintWordsJob.class)
2   .usingJobData("jobDetail2", "这个Job用来测试的")
3   .withIdentity("job2", "group2").build();
4
5 Trigger trigger2 = TriggerBuilder.newTrigger().withIdentity("trigger2",
6   "triggerGroup2")
7   .usingJobData("trigger2", "这是jobDetail2的trigger")
8   .startNow()
9   .withSchedule(SimpleScheduleBuilder.simpleSchedule()
10    .withIntervalInSeconds(1)
11    .repeatForever()).build();
```

Job执行的时候, 可以获取到这些参数信息:

```
Java 复制代码
1 @Override
2 public void execute(JobExecutionContext jobExecutionContext) {
3
4   System.out.println(jobExecutionContext.getJobDetail().getJobDataMap().get("jobDetail1"));
5
6   System.out.println(jobExecutionContext.getTrigger().getJobDataMap().get("trigger1"));
7
8   System.out.println("PrintWordsJob start at: " + LocalDate.now() + " " +
9     LocalTime.now() +
10     ", prints: Hello Job-" + new Random().nextInt(100));
11 }
```

## Trigger、SimpleTrigger、CronTrigger

### Trigger

Trigger 是 Quartz 的触发器, 会去通知 Scheduler 何时去执行对应 Job。

```
Java 复制代码
1 new Trigger().startAt():表示触发器首次被触发的时间;
2 new Trigger().endAt():表示触发器结束触发的时间;
```

### SimpleTrigger

SimpleTrigger可以实现在一个指定时间段内执行一次作业任务或一个时间段内多次执行作业任务。下面的程序就实现了程序运行5s后开始执行Job，执行Job 5s后结束执行：

Java [复制代码](#)

```
1 Date startDate = new Date();
2 startDate.setTime(startDate.getTime() + 5000);
3
4 Date endDate = new Date();
5 endDate.setTime(startDate.getTime() + 5000);
6
7 Trigger trigger = TriggerBuilder.newTrigger()
8     .withIdentity("trigger1", "triggerGroup1")
9     .usingJobData("trigger1", "这是jobDetail1的
trigger")
10     .startNow()
11     .startAt(startDate)
12     .endAt(endDate)
13
14     .withSchedule(SimpleScheduleBuilder.simpleSchedule()
15         .withIntervalInSeconds(5)
16         .repeatForever()).build();
```

## CronTrigger

CronTrigger 功能非常强大，是基于日历的作业调度，而 SimpleTrigger 是精准指定间隔，所以相比 SimpleTrigger，CronTrigger 更加常用。

CronTrigger 是基于 Cron 表达式的，先了解下 Cron 表达式：

由7个子表达式组成字符串的，格式如下：

[秒][分][小时][日][月][周][年]

Cron表达式的语法比较复杂，如：

Bash [复制代码](#)

```
1 * * 30 10 ? * 1/5 *
```

表示（从后往前看）[指定年份] 的 [周一到周五][指定月][不指定日][上午10时][30分][指定秒]

又如：

```
1 00 00 00 ? * 10,11,12 1#5 2021
```

表示 2021 年 10、11、12 月的第 1 周的星期五这一天的 0 时 0 分 0 秒去执行任务。

下面是一些例子：

表示式	说明
0 0 12 * * ?	每天12点运行
0 15 10 ? * *	每天10:15运行
0 15 10 * * ?	每天10:15运行
0 15 10 * * ? *	每天10:15运行
0 15 10 * * ? 2008	在2008年的每天10: 15运行
0 * 14 * * ?	每天14点到15点之间每分钟运行一次，开始于14:00，结束于14:59。
0 0/5 14 * * ?	每天14点到15点每5分钟运行一次，开始于14:00，结束于14:55。
0 0/5 14,18 * * ?	每天14点到15点每5分钟运行一次，此外每天18点到19点每5分钟也运行一次。
0 0-5 14 * * ?	每天14:00点到14:05，每分钟运行一次。
0 10,44 14 ? 3 WED	3月每周三的14:10分到14:44，每分钟运行一次。
0 15 10 ? * MON-FRI	每周一，二，三，四，五的10:15分运行。
0 15 10 15 * ?	每月15日10:15分运行。
0 15 10 L * ?	每月最后一天10:15分运行。
0 15 10 ? * 6L	每月最后一个星期五10:15分运行。
0 15 10 ? * 6L 2007-2009	在2007,2008,2009年每个月的最后一个星期五的10:15分运行。
0 15 10 ? * 6#3	每月第三个星期五的10:15分运行。

## 在线生成Cron表达式工具

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

下面的代码就实现了每周一到周五上午10:30执行定时任务

```

1  public class MyScheduler2 {
2      public static void main(String[] args) throws SchedulerException,
        InterruptedException {
3          // 1、创建调度器Scheduler
4          SchedulerFactory schedulerFactory = new StdSchedulerFactory();
5          Scheduler scheduler = schedulerFactory.getScheduler();
6          // 2、创建JobDetail实例，并与PrintWordsJob类绑定(Job执行内容)
7          JobDetail jobDetail = JobBuilder.newJob(PrintWordsJob.class)
8              .usingJobData("jobDetail1", "这个Job用来测试的")
9              .withIdentity("job1", "group1").build();
10         // 3、构建Trigger实例，每隔1s执行一次
11         Date startDate = new Date();
12         startDate.setTime(startDate.getTime() + 5000);
13
14         Date endDate = new Date();
15         endDate.setTime(startDate.getTime() + 5000);
16
17         CronTrigger cronTrigger =
        TriggerBuilder.newTrigger().withIdentity("trigger1", "triggerGroup1")
18             .usingJobData("trigger1", "这是jobDetail1的trigger")
19             .startNow()//立即生效
20             .startAt(startDate)
21             .endAt(endDate)
22             .withSchedule(CronScheduleBuilder.cronSchedule("* 30 10 ? *
        1/5 2018"))
23             .build();
24
25         //4、执行
26         scheduler.scheduleJob(jobDetail, cronTrigger);
27         System.out.println("-----scheduler start ! -----");
28         scheduler.start();
29         System.out.println("-----scheduler shutdown ! -----");
30
31     }
32 }

```

## Quartz 和 Spring 整合

### 1. 创建 maven 工程 quartzdemo

注意需要添加：spring-context-support, spring-tx 依赖

### 2. 自定义一个Job

```
1 package com.rushuni.quartz_demo.jobs;
2
3 /**
4  * 自定义Job
5  * @author rushuni
6  * @date 2021/08/09
7  */
8 public class JobDemo {
9     public void run(){
10         System.out.println("job execute...");
11     }
12 }
```

### 3. 新增 Spring 配置文件 spring-jobs.xml

用于配置自定义Job、任务描述、触发器、调度工厂等

```

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         xsi:schemaLocation="http://www.springframework.org/schema/beans
5                             http://www.springframework.org/schema/beans/spring-
6 beans.xsd">
7     <!-- 注册自定义Job -->
8     <bean id="jobDemo" class="com.rushuni.quartz_demo.jobs.JobDemo"/>
9     <!-- 注册JobDetail,作用是负责通过反射调用指定的Job -->
10    <bean id="jobDetail"
11          class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBe
12 an">
13        <!-- 注入目标对象 -->
14        <property name="targetObject" ref="jobDemo"/>
15        <!-- 注入目标方法 -->
16        <property name="targetMethod" value="run"/>
17    </bean>
18    <!-- 注册一个触发器,指定任务触发的时间 -->
19    <bean id="myTrigger"
20          class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
21        <!-- 注入JobDetail -->
22        <property name="jobDetail" ref="jobDetail"/>
23        <!-- 指定触发的时间,基于Cron表达式 -->
24        <property name="cronExpression" value="0/10 * * * * ?"/>
25    </bean>
26    <!-- 注册一个统一的调度工厂,通过这个调度工厂调度任务 -->
27    <bean id="scheduler"
28          class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
29        <!-- 注入多个触发器 -->
30        <property name="triggers">
31            <list>
32                <ref bean="myTrigger"/>
33            </list>
34        </property>
35    </bean>
36 </beans>

```

## 4. 编写main方法进行测试

```
1 package com.rushuni;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 /**
6  * @author rushuni
7  * @date 2021/08/09
8  */
9 public class Application {
10     public static void main(String[] args) {
11         new ClassPathXmlApplicationContext("spring-jobs.xml");
12     }
13 }
```

执行上面 main 方法观察控制台，可以发现每隔 10 秒会输出一次，说明每隔 10 秒自定义 Job 被调用一次。

## 定时清理垃圾图片

### 概述

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。

为了能够区分出来哪些图片是垃圾图片，我们在文件上传成功后将图片保存到了一个redis集合中，当套餐数据插入到数据库后我们又将图片名称保存到了另一个redis集合中，通过计算这两个集合的差值就可以获得所有垃圾图片的名称。

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。



# 操作步骤

## 1. 创建新的子工程 seehope-health-jobs

打包方式为 war，导入 Quartz 相关依赖。

## 2. 配置web.xml

```
XML | 复制代码
1 <!DOCTYPE web-app PUBLIC
2     "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3     "http://java.sun.com/dtd/web-app_2_3.dtd" >
4 <web-app>
5     <display-name>Archetype Created Web Application</display-name>
6     <!-- 加载spring容器 -->
7     <context-param>
8         <param-name>contextConfigLocation</param-name>
9         <param-value>classpath*:applicationContext*.xml</param-value>
10    </context-param>
11    <listener>
12        <listener-
13            class>org.springframework.web.context.ContextLoaderListener</listener-class>
14    </listener>
15 </web-app>
```

## 3. 配置log4j.properties

```
Bash | 复制代码
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 -
6 %m%n
7 ### direct messages to file mylog.log ###
8 log4j.appender.file=org.apache.log4j.FileAppender
9 log4j.appender.file.File=../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
```

## 4. 配置applicationContext-redis.xml

```

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
6                             http://www.springframework.org/schema/beans/spring-beans.xsd
7                             http://www.springframework.org/schema/context
8                             http://www.springframework.org/schema/context/spring-
context.xsd">
9      <!--开启spring注解使用-->
10     <context:annotation-config/>
11     <!--注册自定义Job-->
12     <bean id="clearImgJob" class="com.rushuni.jobs.ClearImgJob"/>
13
14     <bean id="jobDetail"
15
16         class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBe
17         an">
18         <!-- 注入目标对象 -->
19         <property name="targetObject" ref="clearImgJob"/>
20         <!-- 注入目标方法 -->
21         <property name="targetMethod" value="clearImg"/>
22     </bean>
23     <!-- 注册一个触发器，指定任务触发的时间 -->
24     <bean id="myTrigger"
25
26         class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
27         <!-- 注入JobDetail -->
28         <property name="jobDetail" ref="jobDetail"/>
29         <!-- 指定触发的时间，基于Cron表达式 -->
30         <property name="cronExpression" value="0/10 * * * * ?"/>
31     </bean>
32     <!-- 注册一个统一的调度工厂，通过这个调度工厂调度任务 -->
33     <bean id="scheduler"
34
35         class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
36         <!-- 注入多个触发器 -->
37         <property name="triggers">
38             <list>
39                 <ref bean="myTrigger"/>
40             </list>
41         </property>
42     </bean>
43 </beans>

```

## 5. 配置applicationContext-jobs.xml

```

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         xsi:schemaLocation="http://www.springframework.org/schema/beans
5                             http://www.springframework.org/schema/beans/spring-
6                             beans.xsd">
7      <!--Jedis连接池的相关配置-->
8      <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
9          <property name="maxTotal" value="200"/>
10         <property name="maxIdle" value="50"/>
11         <property name="testOnBorrow" value="true"/>
12         <property name="testOnReturn" value="true"/>
13     </bean>
14     <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
15         <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
16         <constructor-arg name="host" value="127.0.0.1" />
17         <constructor-arg name="port" value="6379" type="int" />
18         <constructor-arg name="timeout" value="30000" type="int" />
19     </bean>
20 </beans>

```

## 6. 创建ClearImgJob定时任务类

### 6.1 创建 Redis 常量类

```

1  /**
2   * Redis 常量
3   * @author rushuni
4   * @date 2021/8/8
5   */
6  public class RedisConstant {
7      /**
8       * 套餐图片所有图片名称
9       */
10     public static final String SETMEAL_PIC_RESOURCES = "setmealPicResources";
11     /**
12      * 套餐图片保存在数据库中的图片名称
13      */
14     public static final String SETMEAL_PIC_DB_RESOURCES =
15         "setmealPicDbResources";
16 }

```

### 6.2 创建定时任务类

```
1 package com.rushuni.jobs;
2
3 import com.rushuni.constant.RedisConstant;
4 import com.rushuni.util.QiniuUtils;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import redis.clients.jedis.JedisPool;
7
8 import java.util.Set;
9
10 /**
11  * 自定义Job, 实现定时清理垃圾图片
12  * @author rushuni
13  * @date 2021/08/09
14  */
15 public class ClearImgJob {
16     @Autowired
17     private JedisPool jedisPool;
18
19     /**
20      * 清理图片
21      */
22     public void clearImg(){
23         // 根据 Redis 中保存的两个 set 集合进行差值计算, 获得垃圾图片名称集合
24         Set<String> set =
25             jedisPool.getResource().sdiff(RedisConstant.SETMEAL_PIC_RESOURCES,
26             RedisConstant.SETMEAL_PIC_DB_RESOURCES);
27         if(set != null){
28             for (String picName : set) {
29                 //删除七牛云服务器上的图片
30                 QiniuUtils.deleteFileFromQiniu(picName);
31                 //从Redis集合中删除图片名称
32                 jedisPool.getResource().srem(RedisConstant.SETMEAL_PIC_RESOURCES, picName);
33                 System.out.println("自定义任务执行, 清理垃圾图片:" + picName);
34             }
35         }
36     }
37 }
```

研銳科技