## 图片存储方案

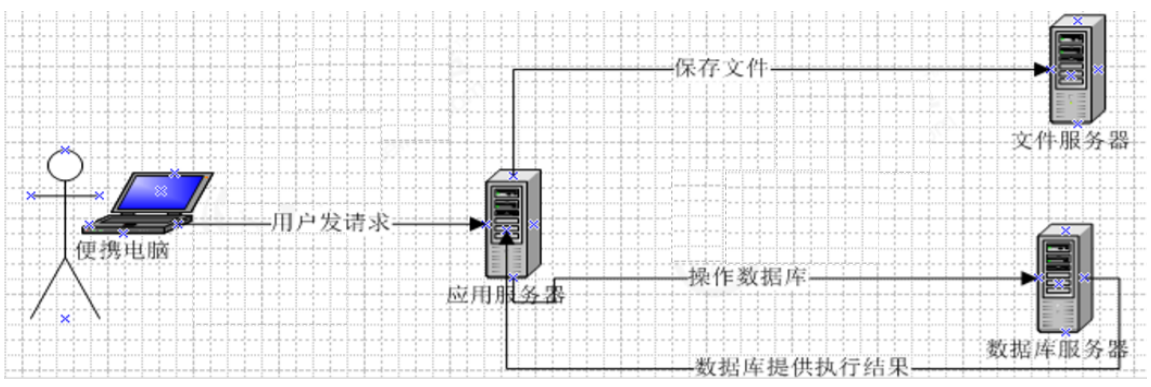
### 介绍

在实际开发中，我们会有很多处理不同功能的服务器。例如：

应用服务器：负责部署我们的应用

数据库服务器：运行我们的数据库

文件服务器：负责存储用户上传文件的服务器



分服务器处理的目的是让服务器各司其职，从而提高我们项目的运行效率。

常见的图片存储方案：

方案一：使用nginx搭建图片服务器

方案二：使用开源的分布式文件存储系统，例如Fastdfs、HDFS等

方案三：使用云存储，例如阿里云、七牛云

### 七牛云存储

七牛云（隶属于上海七牛信息技术有限公司）是国内领先的以视觉智能和数据智能为核 心的企业级云计算服务商，同时也是国内知名智能视频云服务商，累计为 70 多万家企业 提供服务，覆盖了国内80%网民。围绕富媒体场景推出了对象存储、融合 CDN 加速、容器云、大数据平台、深度学习平台等产品、并提供一站式智能视频云解决方案。为各行 业及应用提供可持续发展的智能视频云生态，帮助企业快速上云，创造更广阔的商业价值。

官网：https://www.qiniu.com/

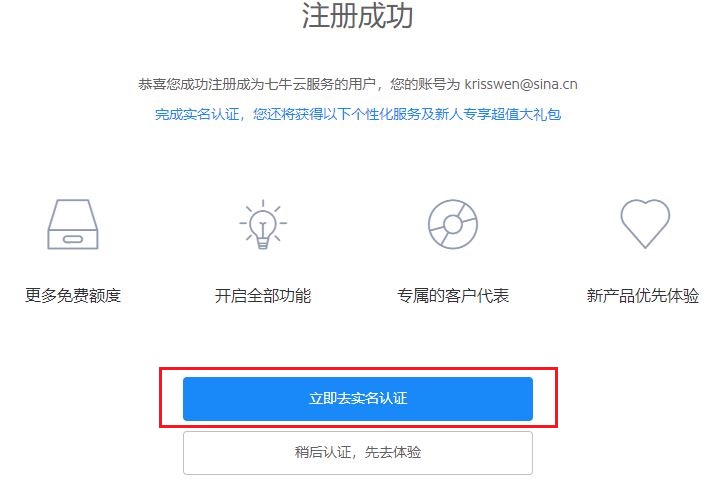
通过七牛云官网介绍我们可以知道其提供了多种服务，我们主要使用的是七牛云提供的 对象存储服务来存储图片。

### 1.1.3注册、登录

要使用七牛云的服务，首先需要注册成为会员。地址：[https://portal.qiniu.com](https://portal.qiniu.com/signup)



注册完成后就可以使用刚刚注册的邮箱和密码登录到七牛云。登录成功之后，还需要进行实名认证才能操作(创建存储空间)。



由于我们是个人使用，所以只需要申请个人认证。



### 1.1.4新建存储空间

要进行图片存储，我们需要在七牛云管理控制台新建存储空间。点击管理控制台首页**对象存储kodo**，页面跳转到新建存储空间页面：



可以创建多个存储空间，各个存储空间是相互独立的。

创建成功之后，我们可以看见:



### 1.1.5 查看存储空间信息

点击存储 空间名称可以查看当前存储空间的相关信息。



### 1.1.6 开发者中心

可以通过七牛云提供的开发者中心学习如何操作七牛云服务，地 址：<https://developer.qiniu.com/>

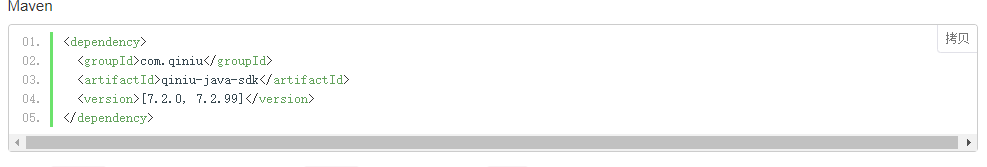


点击对象存储。

七牛云提供了多种方式操作对象存储服务，本项目采用Java SDK方式，地 址：<https://developer.qiniu.com/kodo/sdk/1239/java>



使用Java SDK操作七牛云需要导入如下maven坐标：(我们搭建项目的时候已经导入。)



### 1.1.7鉴权

Java SDK的所有的功能，都需要合法的授权。授权凭证的签算需要七牛账号下的一对有 效的Access Key和Secret Key，这对密钥可以在七牛云管理控制台的个人中心 （https://portal.qiniu.com/user/key）获得，如下图:



### 1.1.8Java SDK操作七牛云

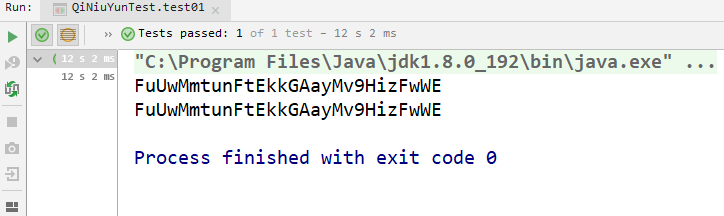
本章节我们就需要使用七牛云提供的Java SDK完成图片上传和删除，我们可以参考官方 提供的例子。



#### 1.1.8.1 上传文件

|  |
| --- |
| **public class** QiNiuYunTest {  @Test  **public void** test01() {  *//构造一个带指定 Region 对象的配置类* Configuration cfg = **new** Configuration(Zone.*zone2*());  *//...其他参数参考类注释* UploadManager uploadManager = **new** UploadManager(cfg);  *//...生成上传凭证，然后准备上传* String accessKey = **"zDx9DjpPq37DnvVo6Je\_eBvhAnCh2hQFYg4PVxO7"**;  String secretKey = **"Kg8YaOlgxwhrw6iMOvFeNTxLLzilpVAm3bc-uMBQ"**;  String bucket = **"health-01"**;*//创建的存储空间的名称  //如果是Windows情况下，格式是 D:\\qiniu\\test.png* String localFilePath = **"C:\\Users\\Hero\\Pictures\\Camera Roll\\images\\pic1.jpg"**;  *//上传文件的名称，默认不指定key的情况下，以文件内容的hash值作为文件名* String key = **null**;  Auth auth = Auth.*create*(accessKey, secretKey);  String upToken = auth.uploadToken(bucket);  **try** {  Response response = uploadManager.put(localFilePath, key, upToken);  *//解析上传成功的结果* DefaultPutRet putRet = **new** Gson().fromJson(response.bodyString(), DefaultPutRet.**class**);  System.***out***.println(putRet.**key**);  System.***out***.println(putRet.**hash**);  } **catch** (QiniuException ex) {  Response r = ex.**response**;  System.***err***.println(r.toString());  **try** {  System.***err***.println(r.bodyString());  } **catch** (QiniuException ex2) {  *//ignore* }  }   }  } |

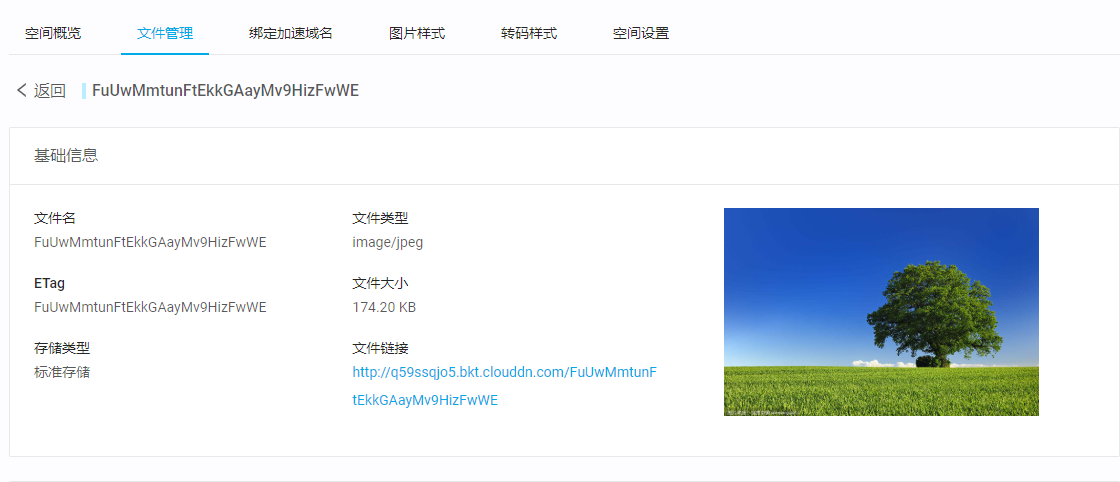
控制台输出效果:



进去七牛云文件管理中心查询文件上传的效果:



点击详情，查看上传文件的详细信息。



将上图中文件链接复制就可以直接访问上传的图片。

#### 1.1.8.2 删除文件

|  |
| --- |
| @Test **public void** test02() {  *//构造一个带指定 Region 对象的配置类* Configuration cfg = **new** Configuration(Zone.*zone1*());  *//...其他参数参考类注释  //...生成上传凭证，然后准备上传* String accessKey = **"zDx9DjpPq37DnvVo6Je\_eBvhAnCh2hQFYg4PVxO7"**;  String secretKey = **"Kg8YaOlgxwhrw6iMOvFeNTxLLzilpVAm3bc-uMBQ"**;  *//创建的存储空间的名称* String bucket = **"health-01"**;  String key = **"Fjtc4-v1fpIJyPjjyOGmyWGl0AEf"**;  Auth auth = Auth.*create*(accessKey, secretKey);  BucketManager bucketManager = **new** BucketManager(auth, cfg);  **try** {  bucketManager.delete(bucket, key);  } **catch** (QiniuException ex) {  *//如果遇到异常，说明删除失败* System.***err***.println(ex.code());  System.***err***.println(ex.**response**.toString());  } } |

#### 1.1.8.3 封装工具类

为了方便操作七牛云存储服务，我们可以将官方提供的案例简单改造成一个工具类，在 我们的项目中直接使用此工具类来操作就可以了。

将工具类复制到health-common子模块中。

|  |
| --- |
| **public class** QiniuUtils {  **public static** String *accessKey* = **"zDx9DjpPq37DnvVo6Je\_eBvhAnCh2hQFYg4PVxO7"**;  **public static** String *secretKey* = **"Kg8YaOlgxwhrw6iMOvFeNTxLLzilpVAm3bc-uMBQ"**;  **public static** String *bucket* = **"health-01"**;   **public static void** upload2Qiniu(String filePath,String fileName){  *//构造一个带指定Zone对象的配置类* Configuration cfg = **new** Configuration(Zone.*zone2*());  UploadManager uploadManager = **new** UploadManager(cfg);  Auth auth = Auth.*create*(*accessKey*, *secretKey*);  String upToken = auth.uploadToken(*bucket*);  **try** {  Response response = uploadManager.put(filePath, fileName, upToken);  *//解析上传成功的结果* DefaultPutRet putRet = **new** Gson().fromJson(response.bodyString(), DefaultPutRet.**class**);  } **catch** (QiniuException ex) {  Response r = ex.**response**;  **try** {  System.***err***.println(r.bodyString());  } **catch** (QiniuException ex2) {  *//ignore* }  }  }   *//上传文件* **public static void** upload2Qiniu(**byte**[] bytes, String fileName){  *//构造一个带指定Zone对象的配置类* Configuration cfg = **new** Configuration(Zone.*zone2*());  *//...其他参数参考类注释* UploadManager uploadManager = **new** UploadManager(cfg);   *//默认不指定key的情况下，以文件内容的hash值作为文件名* String key = fileName;  Auth auth = Auth.*create*(*accessKey*, *secretKey*);  String upToken = auth.uploadToken(*bucket*);  **try** {  Response response = uploadManager.put(bytes, key, upToken);  *//解析上传成功的结果* DefaultPutRet putRet = **new** Gson().fromJson(response.bodyString(), DefaultPutRet.**class**);  System.***out***.println(putRet.**key**);  System.***out***.println(putRet.**hash**);  } **catch** (QiniuException ex) {  Response r = ex.**response**;  System.***err***.println(r.toString());  **try** {  System.***err***.println(r.bodyString());  } **catch** (QiniuException ex2) {  *//ignore* }  }  }   *//删除文件* **public static void** deleteFileFromQiniu(String fileName){  *//构造一个带指定Zone对象的配置类* Configuration cfg = **new** Configuration(Zone.*zone2*());  String key = fileName;  Auth auth = Auth.*create*(*accessKey*, *secretKey*);  BucketManager bucketManager = **new** BucketManager(auth, cfg);  **try** {  bucketManager.delete(*bucket*, key);  } **catch** (QiniuException ex) {  *//如果遇到异常，说明删除失败* System.***err***.println(ex.code());  System.***err***.println(ex.**response**.toString());  }  } } |

## 新增套餐

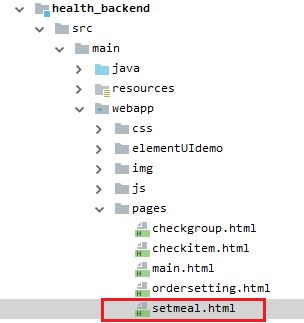
### 需求分析

**套餐其实就是检查组的集合**，例如有一个套餐为“入职体检套餐”，这个体检套餐可以包括 多个检查组：一般检查、血常规、尿常规、肝功三项等。所以在添加套餐时需要选择这个套餐包括的检查组。

套餐对应的实体类为Setmeal，对应的数据表为t\_setmeal。套餐和检查组为多对多关系，所以需要中间表t\_setmeal\_checkgroup进行关联。

### 完善页面

套餐管理页面对应的是setmeal.html页面，根据产品设计的原型已经完成了页面基本结构的编写，现在需要完善页面动态效果。



#### 弹出新增窗口

页面中已经提供了新增窗口，只是出于隐藏状态。只需要将控制展示状态的属性 dialogFormVisible改为true接口显示出新增窗口。点击新建按钮时绑定的方法为 handleCreate，所以在handleCreate方法中修改dialogFormVisible属性的值为true即 可。同时为了增加用户体验度，需要每次点击新建按钮时清空表单输入项。

由于新增套餐时还需要选择此套餐包含的检查组，所以新增套餐窗口分为两部分信息： 基本信息和检查组信息，如下图：

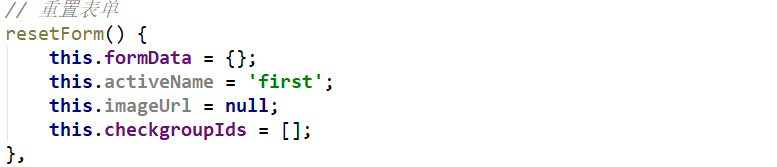




新建按钮绑定单击事件，对应的处理函数为handleCreate。







#### 1.2.2.2动态展示检查组列表

现在虽然已经完成了新增窗口的弹出，但是在检查组信息标签页中需要动态展示所有的 检查组信息列表数据，并且可以进行勾选。具体操作步骤如下：

1. 定义数据模型



1. 动态展示检查组列表数据，数据来源于上面定义的tableData模型数据



1. 完善handleCreate方法，发送ajax请求查询所有检查组数据并将结果赋值给 tableData模型数据用于页面表格展示



1. 分别在CheckGroupController、CheckGroupService、 CheckGroupServiceImpl、CheckGroupDao、CheckGroupDao.xml中扩展方法查询所 有检查组数据。

CheckGroupController:

|  |
| --- |
| *//查询所有检查组* @RequestMapping(**"/findAll"**) **public** Result findAll(){  **try**{  List<CheckGroup> list = **checkGroupService**.findAll();  **return new** Result(**true**,MessageConstant.***QUERY\_CHECKGROUP\_SUCCESS***,list);*//查询成功* }**catch** (Exception e){  e.printStackTrace();  **return new** Result(**false**, MessageConstant.***QUERY\_CHECKGROUP\_FAIL***);*//查询失败* } } |

CheckGroupService:

|  |
| --- |
| **public interface** CheckGroupService {  List<CheckGroup> findAll(); } |

CheckGroupServiceImpl:

|  |
| --- |
| @Override **public** List<CheckGroup> findAll() {  **return checkGroupDao**.findAll(); } |

CheckGroupDao:

|  |
| --- |
| **public interface** CheckGroupDao {  List<CheckGroup> findAll(); } |

CheckGroupDao.xml

|  |
| --- |
| *<!--查询所有检查组信息-->* <**select id="findAll" resultType="com.qf.pojo.CheckGroup"**>  select *\** from t\_checkgroup </**select**> |

#### 1.2.2.3启动测试，查看效果



#### 1.2.2.4图片上传并预览

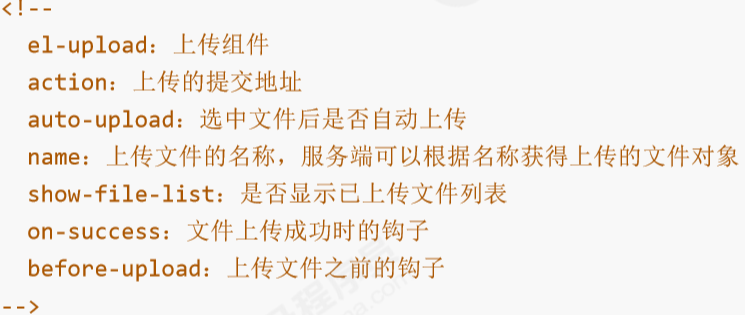
此处使用的是ElementUI提供的上传组件el-upload，提供了多种不同的上传效果，上传成功后可以进行预览。

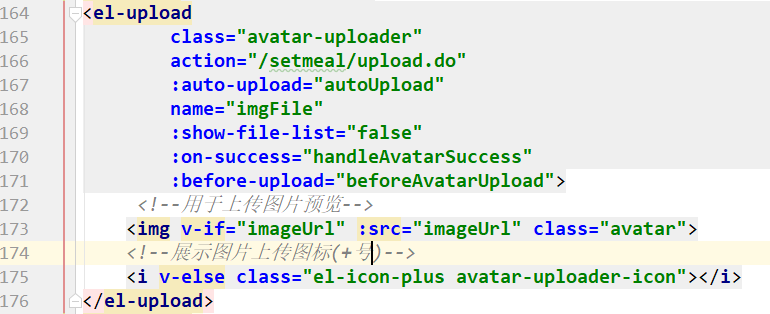
实现步骤:

1. 定义模型数据，用于后面上传文件的图片预览(回显)：



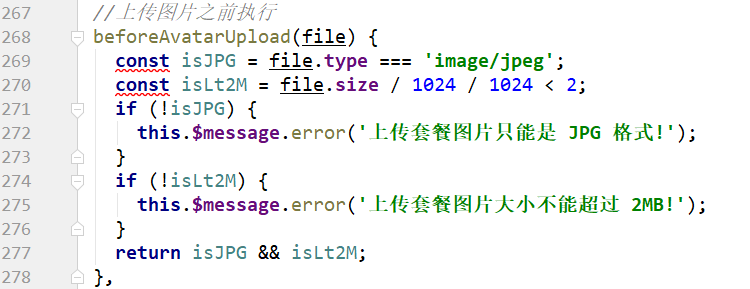
1. 定义文件上传组件





1. 定义对应的函数

文件上传之前的函数:



文件上传成功后的函数:



1. 创建SetmealController，接收上传的文件

|  |
| --- |
| @Controller @RequestMapping(**"/setmeal"**) **public class** SetmealController {  *//文件上传* @RequestMapping(**"/upload"**)  @ResponseBody  **public** Result upload(@RequestParam(**"imgFile"**) MultipartFile imgFile){  System.***out***.println(imgFile);  *//获取原始的文件上传的名字,主要用来截取文件后缀* String originalFilename = imgFile.getOriginalFilename();  **int** index = originalFilename.lastIndexOf(**"."**);// pic1.jpg  String extention = originalFilename.substring(index);*//.jpg* String fileName = UUID.*randomUUID*().toString()+extention;*//文件上传之后的名称* **try** {  QiniuUtils.*upload2Qiniu*(imgFile.getBytes(),fileName);  } **catch** (IOException e) {  e.printStackTrace();  **return new** Result(**false**,MessageConstant.***PIC\_UPLOAD\_FAIL***);  }  **return new** Result(**true**,MessageConstant.***PIC\_UPLOAD\_SUCCESS***,fileName);  }  } |

注意：别忘了在spring配置文件中配置文件上传组件:

在springmvc.xml中配置文件上传的配置(之前在搭建环境的时候，已经配好)



#### 1.2.2.5 启动测试



七牛云上查看:



#### 1.2.2.6提交请求

当用户点击新增窗口中的确定按钮时发送ajax请求将数据提交到后台进行数据库操作。提 交到后台的数据分为两部分：套餐基本信息（对应的模型数据为formData）和检查组id 数组（对应的模型数据为checkgroupIds）。

为确定按钮绑定单击事件，对应的处理函数为handleAdd。



完善handleAdd方法。



### 后台代码

#### controller

在SetmealController中增加方法。

|  |
| --- |
| @Reference **private** SetmealService **setmealService**; *//新增套餐* @RequestMapping(**"/add"**) **public** Result add(@RequestBody Setmeal setmeal, Integer[] checkgroupIds){  **try**{  **setmealService**.add(setmeal,checkgroupIds);  }**catch** (Exception e){  e.printStackTrace();  **return new** Result(**false**,MessageConstant.***ADD\_SETMEAL\_FAIL***);  }  **return new** Result(**true**,MessageConstant.***ADD\_SETMEAL\_SUCCESS***); } |

#### 1.2.3.2服务接口

创建SetmealService接口并提供新增方法。

|  |
| --- |
| **public interface** SetmealService {  **void** add(Setmeal setmeal, Integer[] checkgroupIds); } |

#### 1.2.3.3服务接口实现类

创建SetmealServiceImpl服务实现类并实现新增方法。

|  |
| --- |
| @Service(interfaceClass = SetmealService.**class**)*//明确当前服务实现的是哪个接口* @Transactional **public class** SetmealServiceImpl **implements** SetmealService {   @Autowired  SetmealDao **setmealDao**;  *//新增套餐信息，同时需要关联检查组* @Override  **public void** add(Setmeal setmeal, Integer[] checkgroupIds) {  **setmealDao**.add(setmeal);  Integer setmealId = setmeal.getId();  **this**.setSetmealAndCheckgroup(setmealId,checkgroupIds);  }  *//设置套餐和检查组多对多关系，操作t\_setmeal\_checkgroup* **private void** setSetmealAndCheckgroup(Integer setmealId, Integer[] checkgroupIds) {  **if**(checkgroupIds != **null** && checkgroupIds.**length** > 0){  **for** (Integer checkgroupId : checkgroupIds) {  Map<String,Integer> map = **new** HashMap<>();  map.put(**"setmealId"**,setmealId);  map.put(**"checkgroupId"**,checkgroupId);  **setmealDao**.setSetmealAndCheckGroup(map);  }  }  } } |

#### 1.2.3.4 Dao接口

创建SetmealDao接口并提供相关方法。

|  |
| --- |
| **public interface** SetmealDao {  **public void** add(Setmeal setmeal);  **public void** setSetmealAndCheckGroup(Map map); } |

#### 1.2.3.5 Mapper映射文件

创建SetmealDao.xml文件并定义相关SQL语句。

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>*** <**mapper namespace="com.qf.dao.SetmealDao"**>  *<!--插入套餐数据-->* <**insert id="add" parameterType="com.qf.pojo.Setmeal"**>  *<!--通过mybatis框架提供的selectKey标签获得自增产生的ID值-->* <**selectKey resultType="java.lang.Integer" order="AFTER" keyProperty="id"**>  select LAST\_INSERT\_ID()  </**selectKey**>  insert into t\_setmeal(code,name,sex,helpCode,remark,attention,age,price,img)  values  (#{code},#{name},#{sex},#{helpCode},#{remark},#{attention},#{age},#{price},#{img})  </**insert**>  *<!--设置套餐和检查组多对多关系-->* <**insert id="addSetmealCheckGroup" parameterType="map"**>  insert into t\_setmeal\_checkgroup(setmeal\_id,checkgroup\_id)  values  (#{setmealId},#{checkgroupId})  </**insert**> </**mapper**> |

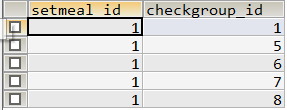
#### 1.2.3.6 启动测试

打开新增套餐对话框，进行数据插入并提交。然后查询对应的数据表(t\_setmeal、t\_setmeal\_checkgroup)

套餐表(t\_setmeal):



中间表(t\_setmeal\_checkgroup):



### 完善文件上传

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。如何实现呢？

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

本小节我们先来完成前面2个环节，第3个环节（清理图片环节）在后面会通过定时任务再实现。

实现步骤：

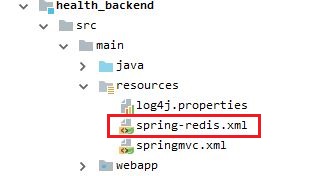
(1)在health\_backend项目中提供Spring配置文件applicationContext-redis.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:p="http://www.springframework.org/schema/p"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!--Jedis连接池的相关配置-->* <**bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig"**>  <**property name="maxTotal"**>  <**value**>200</**value**>  </**property**>  <**property name="maxIdle"**>  <**value**>50</**value**>  </**property**>  </**bean**>  <**bean id="jedisPool" class="redis.clients.jedis.JedisPool"**>  <**constructor-arg name="poolConfig" ref="jedisPoolConfig"** />  <**constructor-arg name="host" value="127.0.0.1"** />  <**constructor-arg name="port" value="6379" type="int"** />  <**constructor-arg name="timeout" value="30000" type="int"** />  </**bean**> </**beans**> |

在springmvc.xml中加载spring-redis.xml配置文件

|  |
| --- |
| *<!--加载redis配置文件-->* <**import resource="applicationContext-redis.xml"**></**import**> |

定义之后的目录结构如下:



(2) 在health\_common工程中提供Redis常量类.

|  |
| --- |
| **public class** RedisConstant {  *//套餐图片所有图片名称* **public static final** String ***SETMEAL\_PIC\_RESOURCES*** = **"setmealPicResources"**;  *//套餐图片保存在数据库中的图片名称* **public static final** String ***SETMEAL\_PIC\_DB\_RESOURCES*** = **"setmealPicDbResources"**; } |

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

|  |
| --- |
| @RequestMapping(**"/upload"**) @ResponseBody **public** Result upload(MultipartFile imgFile){  **try** {  //1、重命名  String originalFilename = imgFile.getOriginalFilename();  String extName = originalFilename.substring(originalFilename.lastIndexOf(**"."**));  String fileName = UUID.*randomUUID*().toString()+extName;   //2、上传  QiniuUtils.*upload2Qiniu*(imgFile.getBytes(),fileName);  **jedisPool**.getResource().sadd(RedisConstant.***SETMEAL\_PIC\_RESOURCES***,fileName);  **return new** Result(**true**, MessageConstant.***PIC\_UPLOAD\_SUCCESS***,fileName);  }**catch** (Exception e){  e.printStackTrace();  **return new** Result(**false**, MessageConstant.***PIC\_UPLOAD\_FAIL***);  } } |

4在保存完成套餐信息后将图片名称存储到redis 集合中

|  |
| --- |
| @RequestMapping(**"/add"**) @ResponseBody **public** Result add(@RequestBody Setmeal setmeal, Integer[] checkgroupIds){  **try** {  **setmealService**.add(setmeal,checkgroupIds);//新增套餐  **jedisPool**.getResource().sadd(RedisConstant.***SETMEAL\_PIC\_DB\_RESOURCES***,setmeal.getImg());  **return new** Result(**true**, MessageConstant.***ADD\_SETMEAL\_SUCCESS***);  }**catch** (Exception e){  e.printStackTrace();  **return new** Result(**false**, MessageConstant.***ADD\_SETMEAL\_FAIL***);  } } |

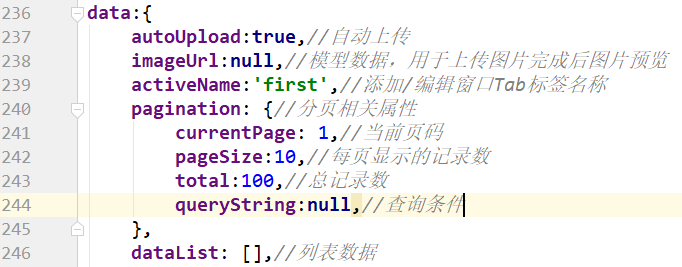
1. 测试



## 1.3体检套餐分页

### 1.3.1完善页面

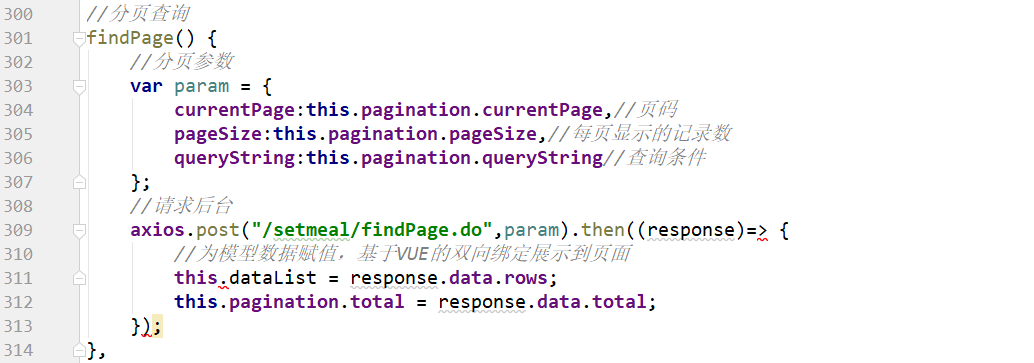
#### 1.3.1.1定义分页相关模型数据



#### 1.3.1.2定义分页的方法

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。





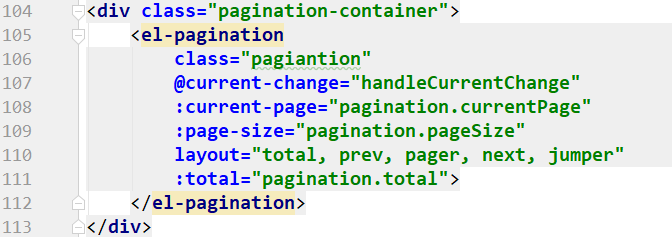
#### 1.3.1.3完善分页方法执行时机

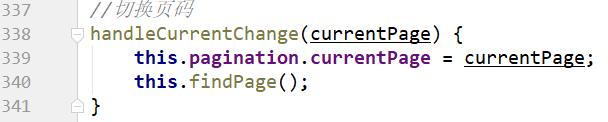
除了在created钩子函数中调用findPage方法查询分页数据之外，当用户点击查询按钮或 者点击分页条中的页码时也需要调用findPage方法重新发起查询请求。

为查询按钮绑定单击事件，调用findPage方法。



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。





### 1.3.2后台代码

#### 1.3.2.1 controller

在SetmealController中增加分页查询方法。

|  |
| --- |
| *//分页查询* @RequestMapping(**"/findPage"**) **public** PageResult findPage(@RequestBody QueryPageBean queryPageBean){  **return setmealService**.pageQuery(queryPageBean); } |

#### 1.3.2.2 服务接口

在SetmealService服务接口中扩展分页查询方法。

|  |
| --- |
| **public interface** SetmealService {  **void** add(Setmeal setmeal, Integer[] checkgroupIds);   **PageResult pageQuery(QueryPageBean queryPageBean);** } |

#### 1.3.2.3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。

|  |
| --- |
| @Override **public** PageResult pageQuery(QueryPageBean queryPageBean) {  Integer currentPage = queryPageBean.getCurrentPage();  Integer pageSize = queryPageBean.getPageSize();  String queryString = queryPageBean.getQueryString();  PageHelper.*startPage*(currentPage,pageSize);  Page<Setmeal> page = **setmealDao**.findByCondition(queryString);  **return new** PageResult(page.getTotal(),page.getResult()); } |

#### 1.3.2.4 Dao接口

在SetmealDao接口中扩展分页查询方法。

|  |
| --- |
| **public interface** SetmealDao {  **public void** add(Setmeal setmeal);  **public void** setSetmealAndCheckGroup(Map map);  **public Page<Setmeal> findByCondition(String queryString);** } |

#### 1.3.2.5 Mapper映射文件

在SetmealDao.xml文件中增加SQL定义。

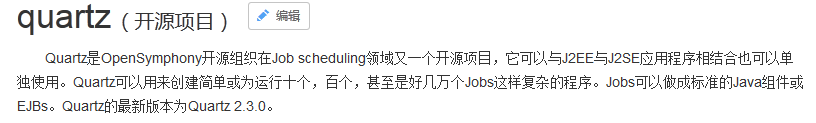
|  |
| --- |
| *<!--根据条件进行查询-->* <**select id="findByCondition" parameterType="string" resultType="com.qf.pojo.Setmeal"**>  select \* from t\_setmeal  <**if test="value != null and value != '' and value.length > 0"**>  where code = #{value} or name = #{value} or helpCode = #{value}  </**if**> </**select**> |

#### 1.2.2.6 测试



## 1.4 定时任务组件Quartz

### 1.4.1 Quartz介绍



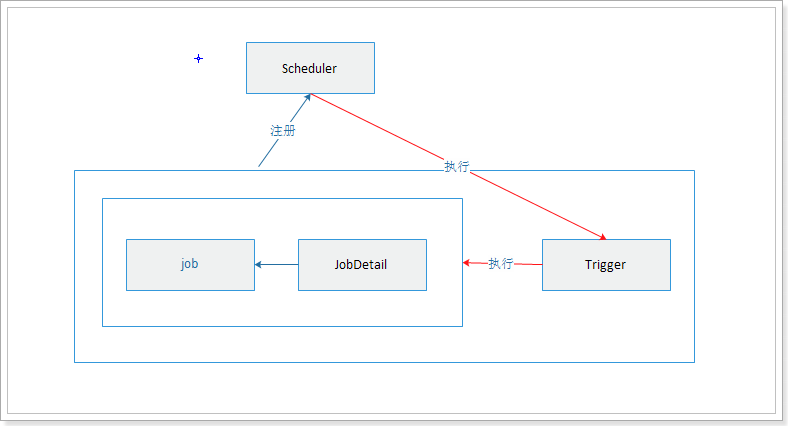
官网：<http://www.quartz-scheduler.org/>

### 1.4.2 Quartz框架的使用思路

1）job - 任务 - 你要做什么事？

2）Trigger - 触发器 - 你什么时候去做？

3）Scheduler - 任务调度 - 你什么时候需要去做什么事？



### 1.4.3.Quartz框架的基本使用方式

添加Quartz依赖:

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>org.quartz-scheduler</**groupId**>  <**artifactId**>quartz</**artifactId**>  </**dependency**> </**dependencies**> |

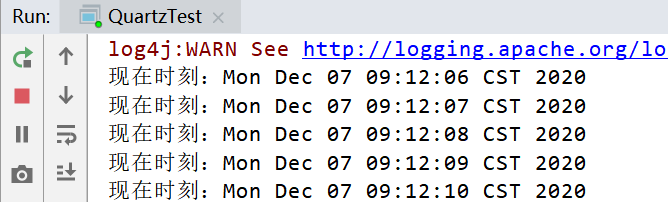
1. 自定义一个Job

|  |
| --- |
| /\*\*  \* 任务类  \*/ **public class** QuartzDemo **implements** Job {   //任务  @Override  **public void** execute(JobExecutionContext jobExecutionContext) **throws** JobExecutionException {  System.***out***.println(**"现在时刻："**+**new** Date());  } } |

1. 编写main方法进行测试

|  |
| --- |
| **public class** QuartzTest {  **public static void** main(String[] args) **throws** SchedulerException {  //1、创建job：做什么事  JobDetail job = JobBuilder.*newJob*(QuartzDemo.**class**).build();   //2、创建trigger：什么时候  CronTrigger trigger = TriggerBuilder.*newTrigger*().withSchedule(  CronScheduleBuilder.*cronSchedule*(**"0/1 \* \* \* \* ?"**)  ).build();   //3、创建scheduler：什么时候做什么事  Scheduler scheduler = StdSchedulerFactory.*getDefaultScheduler*();  scheduler.scheduleJob(job,trigger);  scheduler.start();  } } |

查看控制台:



执行上面main方法观察控制台，可以发现每隔1秒会输出一次，说明每隔1秒自定义 Job被调用一次。

### 1.4.3 Corn表达式

Cron 表达式是一个字符串，分为 6 或 7 个域，每一个域代表一个含义

Cron 有如下两种语法格式：

（1）Seconds Minutes Hours Day Month Week Year

（2）Seconds Minutes Hours Day Month Week

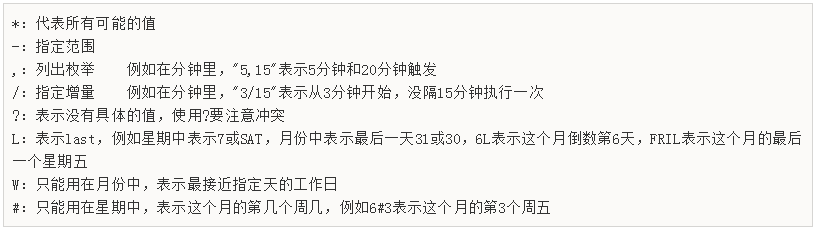
**一、结构**

corn 从左到右（用空格隔开）：秒 分 小时 月份中的日期 月份 星期中的日期 年份

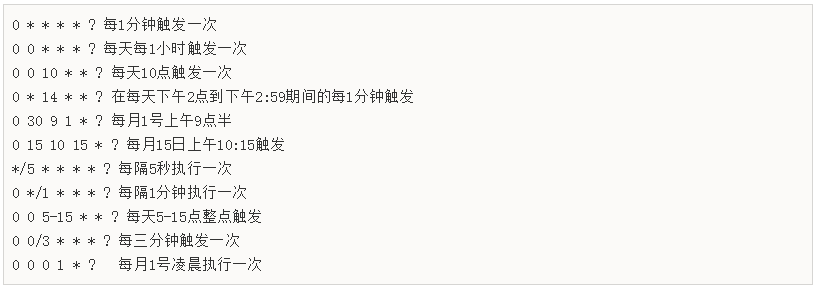
二、**cron表达式**



**字段含义：**



**示例：**



**?：**

“?”字符只在日期域和星期域中使用。它被用来指定“非明确的值”。

例如：

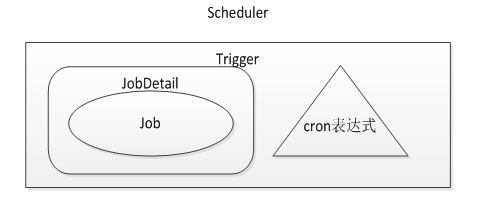
想在每月的20日触发调度0 0 0 20 \* ?，而且只能使用如下写法：0/1 0 0 20 \* ?。其中最后一位只能用？，而不能使 用\*，因为\*表示不管星期几都会触发恰好与20冲突。

### 1.4.4 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。

<http://cron.qqe2.com/>

### 1.4.5.spring整合 Quartz定时框架



#### 1.4.5.1.创建 Job类

|  |
| --- |
| **public class** QuartzDemo2 {  **public void** execute(){  System.***out***.println(**"现在时刻："**+**new** Date());  } } |

#### 1.4.5.2.Spring整合Quartz

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns:task="http://www.springframework.org/schema/task"  xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  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 http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task.xsd"**>  <**bean id="quartzDemo2" class="com.qf.job.QuartzDemo2"**></**bean**>  <!--job：干什么事-->  <**bean id="job" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>  <**property name="targetObject" ref="quartzDemo2"**></**property**>  <**property name="targetMethod" value="execute"**></**property**>   </**bean**>  <!--trigger：什么时候-->  <**bean id="trigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>  <**property name="cronExpression" value="\*/1 \* \* \* \* ?"**></**property**>  <**property name="jobDetail" ref="job"**></**property**>  </**bean**>   <!--scheduler：什么时候干什么事-->  <**bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean"**>  <**property name="triggers"**>  <**list**>  <**ref bean="trigger"**></**ref**>  </**list**>  </**property**>  </**bean**> </**beans**> |

#### 1.4.5.3.编写测试代码

|  |
| --- |
| **public class** QuartzTest {  **public static void** main(String[] args) **throws** SchedulerException, IOException {  **new** ClassPathXmlApplicationContext(**"applicationContext-quartz.xml"**);  } } |

## 1.5 定时清理垃圾图片

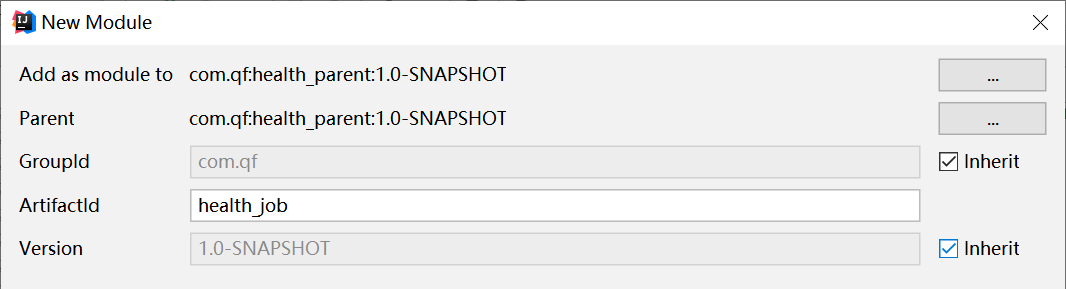
前面我们已经完成了体检套餐的管理，在新增套餐时套餐的基本信息和图片是分两次提 交到后台进行操作的。也就是用户首先将图片上传到七牛云服务器，然后再提交新增窗 口中录入的其他信息。如果用户只是上传了图片而没有提交录入的其他信息，此时的图 片就变为了垃圾图片，因为在数据库中并没有记录它的存在。此时我们要如何处理这些 垃圾图片呢？

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maven工程health\_jobs，打包方式为war导入Quartz等相关坐标。



导入相关的坐标:

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**parent**>  <**artifactId**>health\_parent</**artifactId**>  <**groupId**>com.qf</**groupId**>  <**version**>1.0-SNAPSHOT</**version**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>   <**artifactId**>health\_jobs</**artifactId**>  <**packaging**>war</**packaging**>   <**dependencies**>  <**dependency**>  <**groupId**>com.qf</**groupId**>  <**artifactId**>health\_interface</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.quartz-scheduler</**groupId**>  <**artifactId**>quartz</**artifactId**>  </**dependency**>  </**dependencies**>  <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.tomcat.maven</**groupId**>  <**artifactId**>tomcat7-maven-plugin</**artifactId**>  <**configuration**>  *<!-- 指定端口 -->* <**port**>83</**port**>  *<!-- 请求路径 -->* <**path**>/</**path**>  </**configuration**>  </**plugin**>  </**plugins**>  </**build**> </**project**> |

1. 配置web.xml

|  |
| --- |
| **<!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" *>*** <**web-app**>  <**display-name**>Archetype Created Web Application</**display-name**>  *<!-- 加载spring容器 -->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:applicationContext-\*.xml</**param-value**>  </**context-param**>  <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**> </**web-app**> |

1. 配置log4j.properties

|  |
| --- |
| *### direct log messages to stdout ###* **log4j.appender.stdout**=**org.apache.log4j.ConsoleAppender log4j.appender.stdout.Target**=**System.err log4j.appender.stdout.layout**=**org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern**=**%d{ABSOLUTE} %5p %c{1}:%L - %m%n** *### direct messages to file mylog.log ###* #log4j.appender.file=org.apache.log4j.FileAppender #log4j.appender.file.File=c:\\mylog.log #log4j.appender.file.layout=org.apache.log4j.PatternLayout #log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n*### set log levels - for more verbose logging change 'info' to 'debug' ###* **log4j.rootLogger**=**debug, stdout** |

1. 配置applicationContext-redis.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd "**>  <!--1、把JedisPoolConfig放到spring的ioc容器-->  <**bean id="config" class="redis.clients.jedis.JedisPoolConfig"**>  <**property name="maxTotal" value="30"**></**property**>  <**property name="minIdle" value="2"**></**property**>  </**bean**>  <!--2、把JedisPool放到spring的ioc容器-->  <**bean id="pool" class="redis.clients.jedis.JedisPool"**>  <**constructor-arg name="poolConfig" ref="config"**></**constructor-arg**>  <**constructor-arg name="host" value="192.168.204.153"**></**constructor-arg**>  <**constructor-arg name="port" value="6379"**></**constructor-arg**>  </**bean**> </**beans**> |

1. 配置applicationContext-jobs.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!--开启spring注解使用-->* <**context:annotation-config**></**context:annotation-config**>  *<!--注册自定义Job-->* <**bean id="clearImgJob" class="com.qf.jobs.ClearImgJob"**></**bean**>   <**bean id="jobDetail"  class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>  *<!-- 注入目标对象 -->* <**property name="targetObject" ref="clearImgJob"**/>  *<!-- 注入目标方法 -->* <**property name="targetMethod" value="clearImg"**/>  </**bean**>  *<!-- 注册一个触发器，指定任务触发的时间 -->* <**bean id="myTrigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>  *<!-- 注入JobDetail -->* <**property name="jobDetail" ref="jobDetail"**/>  *<!-- 指定触发的时间，基于Cron表达式 -->* <**property name="cronExpression"**>  *<!--  <value>0 0 2 \* \* ?</value>  -->* <**value**>0/10 \* \* \* \* ?</**value**>  </**property**>  </**bean**>  *<!-- 注册一个统一的调度工厂，通过这个调度工厂调度任务 -->* <**bean id="scheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean"**>  *<!-- 注入多个触发器 -->* <**property name="triggers"**>  <**list**>  <**ref bean="myTrigger"**/>  </**list**>  </**property**>  </**bean**> </**beans**> |

1. 创建ClearImgJob定时任务类

|  |
| --- |
| */\*\*  \* 自定义Job，实现定时清理垃圾图片  \*/* **public class** ClearImgJob {  @Autowired  **private** JedisPool **jedisPool**;  **public void** clearImg(){  *//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* Set<String> set = **jedisPool**.getResource().sdiff(RedisConstant.***SETMEAL\_PIC\_RESOURCES***, RedisConstant.***SETMEAL\_PIC\_DB\_RESOURCES***);  **if**(set != **null**){  **for** (String picName : set) {  *//删除七牛云服务器上的图片* QiniuUtils.*deleteFileFromQiniu*(picName);  *//从Redis集合中删除图片名称* **jedisPool**.getResource().srem(RedisConstant.***SETMEAL\_PIC\_RESOURCES***,picName);  System.***out***.println(**"自定义任务执行，清理垃圾图片:"** + picName);  }  }  } } |