第4章 预约管理-套餐管理

1. 图片存储方案

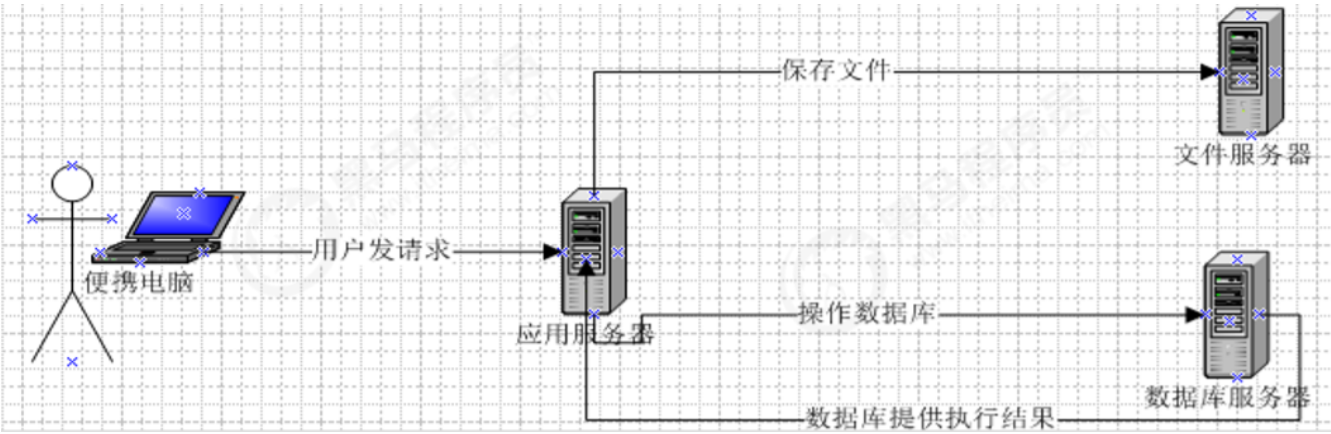
1.1 介绍

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、七牛云等

1.2 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/

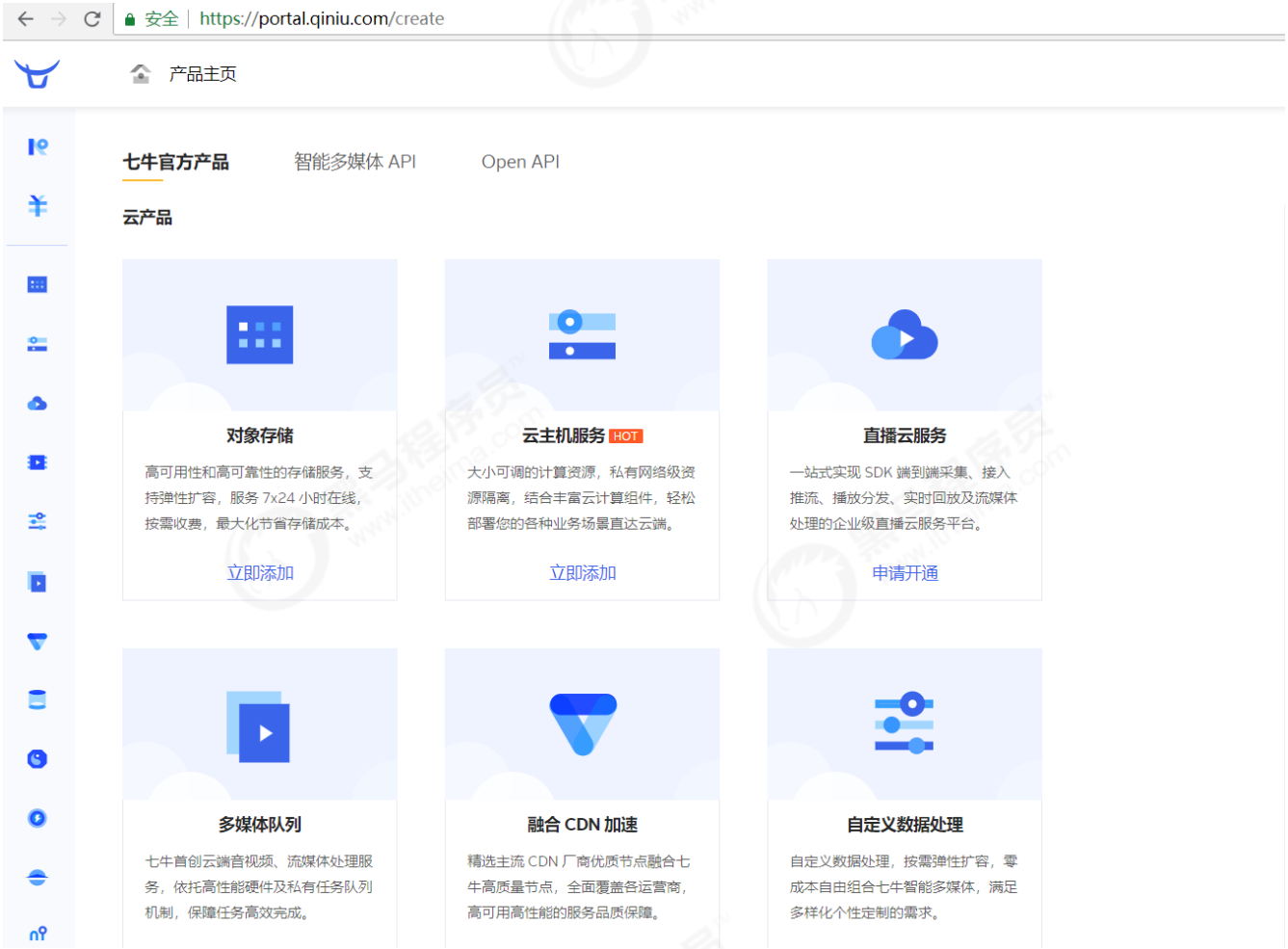
通过七牛云官网介绍我们可以知道其提供了多种服务，我们主要使用的是七牛云提供的对象存储服务来存储图片。

1.2.1 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/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：

登录成功后点击页面右上角管理控制台：



注意：登录成功后还需要进行实名认证才能进行相关操作。

1.2.2 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下的立即添加按钮，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。

1.2.3 查看存储空间信息

存储空间创建后，会在左侧的存储空间列表菜单中展示创建的存储空间名称，点击存储空间名称可以查看当前存储空间的相关信息

1.2.4 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，跳转到对象存储开发页面，地址：https://developer.qiniu.com/kodo

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

简介

此 SDK 适用于 Java 7 及以上版本。使用此 SDK 构建您的网络应用程序，能让您以非常便捷地方式将数据安全地存储到七牛云上。无论您的网络应用是一个网站程序，还是包括从云端（服务端程序）到终端（手持设备应用）的架构服务或应用，通过七牛云及其 SDK，都能让您应用程序的终端用户高速上传和下载，同时也让您的服务端更加轻盈。

Java SDK 属于七牛服务端SDK之一，主要有如下功能:

1. 提供生成客户端上传所需的上传凭证的功能
2. 提供文件从服务端直接上传七牛的功能
3. 提供对七牛空间中文件进行管理的功能
4. 提供对七牛空间中文件进行处理的功能
5. 提供七牛CDN相关的刷新，预取，日志功能

使用Java SDK操作七牛云需要导入如下maven坐标：

```
<dependency>
  <groupId>com.qiniu</groupId>
  <artifactId>qiniu-java-sdk</artifactId>
  <version>7.2.0</version>
</dependency>
```

1.2.5 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：

1.2.6 Java 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。

```
//构造一个带指定Zone对象的配置类
Configuration cfg = new Configuration(Zone.zone0());
//...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 = "your access key";
String secretKey = "your secret key";
String bucket = "your bucket name";
//如果是Windows情况下，格式是 D:\\qiniu\\test.png
String localFilePath = "/home/qiniu/test.png";
//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
    }
}
```

```
//构造一个带指定Zone对象的配置类
Configuration cfg = new Configuration(Zone.zone0());
//...其他参数参考类注释

String accessKey = "your access key";
String secretKey = "your secret key";

String bucket = "your bucket name";
String key = "your file key";

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());
}
```

1.2.7 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：

```
package com.itheima.utils;

import com.google.gson.Gson;
import com.qiniu.common.QiniuException;
import com.qiniu.common.Zone;
import com.qiniu.http.Response;
import com.qiniu.storage.BucketManager;
import com.qiniu.storage.Configuration;
import com.qiniu.storage.UploadManager;
import com.qiniu.storage.model.DefaultPutRet;
import com.qiniu.util.Auth;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

/**
 * 七牛云工具类
 */
public class QiniuUtils {
    public  static String accessKey =
"dulF9Wze9bxujtuRvu3yyYb9JX1Sp23jzd3tO708";
    public  static String secretKey =
"vZkhW7iot3uWwcWz9vXfbaP4JepdWADFDHVLMZOe";
    public  static String bucket = "qiniutest";

    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.zone0());
        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.zone0());
        //...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.zone0());
```

```
        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());
        }
    }
}
```

将此工具类放在health_common工程中，后续会使用到。

2. 新增套餐

2.1 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。

2.2 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。

2.2.1 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

```
<el-button type="primary" class="butT" @click="handleCreate()">新建</el-button>
```

```
// 重置表单
resetForm() {
  this.formData = {};
  this.activeName='first';
  this.checkgroupIds = [];
  this.imageUrl = null;
}
// 弹出添加窗口
handleCreate() {
  this.dialogFormVisible = true;
  this.resetForm();
}
```

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）定义模型数据

```
tableData:[],//添加表单窗口中检查组列表数据
checkgroupIds:[],//添加表单窗口中检查组复选框对应id
```

（2）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

```
<table class="datatable">
  <thead>
    <tr>
      <th>选择</th>
      <th>项目编码</th>
      <th>项目名称</th>
      <th>项目说明</th>
    </tr>
  </thead>
  <tbody>
    <tr v-for="c in tableData">
      <td>
        <input :id="c.id" v-model="checkgroupIds" type="checkbox"
:value="c.id">
      </td>
      <td><label :for="c.id">{{c.code}}</label></td>
      <td><label :for="c.id">{{c.name}}</label></td>
      <td><label :for="c.id">{{c.remark}}</label></td>
    </tr>
  </tbody>
</table>
```

（3）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

```
// 弹出添加窗口
handleCreate() {
  this.dialogFormVisible = true;
  this.resetForm();
  axios.get("/checkgroup/findAll.do").then((res)=> {
    if(res.data.flag){
      this.tableData = res.data.data;
    }else{
      this.$message.error(res.data.message);
    }
  });
}
```

（4）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
@RequestMapping("/findAll")
public Result findAll(){
  List<CheckGroup> checkGroupList = checkGroupService.findAll();
  if(checkGroupList != null && checkGroupList.size() > 0){
    Result result = new Result(true,
MessageConstant.QUERY_CHECKGROUP_SUCCESS);
    result.setData(checkGroupList);
    return result;
  }
  return new Result(false,MessageConstant.QUERY_CHECKGROUP_FAIL);
}
```

CheckGroupService：

```
List<CheckGroup> findAll();
```

CheckGroupServiceImpl：

```
public List<CheckGroup> findAll() {
  return checkGroupDao.findAll();
}
```

CheckGroupDao：

```
List<CheckGroup> findAll();
```

CheckGroupDao.xml：

```
<select id="findAll" resultType="com.itheima.pojo.CheckGroup">
  select * from t_checkgroup
</select>
```

2.2.3 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：

```
imageUrl:null,//模型数据，用于上传图片完成后图片预览
```

（2）定义上传组件：

```
<!--
  el-upload：上传组件
  action：上传的提交地址
  auto-upload：选中文件后是否自动上传
  name：上传文件的名称，服务端可以根据名称获得上传的文件对象
  show-file-list：是否显示已上传文件列表
  on-success：文件上传成功时的钩子
  before-upload：上传文件之前的钩子
-->
<el-upload
            class="avatar-uploader"
            action="/setmeal/upload.do"
            :auto-upload="autoUpload"
            name="imgFile"
            :show-file-list="false"
            :on-success="handleAvatarSuccess"
            :before-upload="beforeAvatarUpload">
  <!--用于上传图片预览-->
  <img v-if="imageUrl" :src="imageUrl" class="avatar">
  <!--用于展示上传图标-->
  <i v-else class="el-icon-plus avatar-uploader-icon"></i>
</el-upload>
```

（3）定义对应的钩子函数：

```
//文件上传成功后的钩子，response为服务端返回的值，file为当前上传的文件封装成的js对象
handleAvatarSuccess(response, file) {
  this.imageUrl = "http://pqjroc654.bkt.clouddn.com/"+response.data;
  this.$message({
    message: response.message,
    type: response.flag ? 'success' : 'error'
  });
  //设置模型数据（图片名称），后续提交ajax请求时会提交到后台最终保存到数据库
  this.formData.img = response.data;
}

//上传文件之前的钩子
beforeAvatarUpload(file) {
  const isJPG = file.type === 'image/jpeg';
  const isLt2M = file.size / 1024 / 1024 < 2;
  if (!isJPG) {
    this.$message.error('上传套餐图片只能是 JPG 格式!');
  }
  if (!isLt2M) {
    this.$message.error('上传套餐图片大小不能超过 2MB!');
  }
  return isJPG && isLt2M;
}
```

（4）创建SetmealController，接收上传的文件

```java
package com.itheima.controller;

import com.alibaba.dubbo.config.annotation.Reference;
import com.itheima.constant.MessageConstant;
import com.itheima.entity.PageResult;
import com.itheima.entity.QueryPageBean;
import com.itheima.entity.Result;
import com.itheima.pojo.CheckGroup;
import com.itheima.pojo.Setmeal;
import com.itheima.service.CheckGroupService;
import com.itheima.service.SetmealService;
import com.itheima.utils.QiniuUtils;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.util.List;
import java.util.UUID;
/**
 * 套餐管理
 */
@RestController
@RequestMapping("/setmeal")
public class SetmealController {
    @Reference
    private SetmealService setmealService;

    //图片上传
    @RequestMapping("/upload")
    public Result upload(@RequestParam("imgFile")MultipartFile imgFile){
        try{
            //获取原始文件名
            String originalFilename = imgFile.getOriginalFilename();
            int lastIndexOf = originalFilename.lastIndexOf(".");
            //获取文件后缀

            String suffix = originalFilename.substring(lastIndexOf - 1);
```

```
            //使用UUID随机产生文件名称，防止同名文件覆盖
            String fileName = UUID.randomUUID().toString() + suffix;
            QiniuUtils.upload2Qiniu(imgFile.getBytes(),fileName);
            //图片上传成功
            Result result = new Result(true,
MessageConstant.PIC_UPLOAD_SUCCESS);
            result.setData(fileName);
            return result;
        }catch (Exception e){
            e.printStackTrace();
            //图片上传失败
            return new Result(false,MessageConstant.PIC_UPLOAD_FAIL);
        }
    }
}
```

注意：别忘了在spring配置文件中配置文件上传组件

```
<!--文件上传组件-->
<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver
">
  <property name="maxUploadSize" value="104857600" />
  <property name="maxInMemorySize" value="4096" />
  <property name="defaultEncoding" value="UTF-8"/>
</bean>
```

2.2.4 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

```
<el-button type="primary" @click="handleAdd()">确定</el-button>
```

完善handleAdd方法

```
//添加
handleAdd () {
  axios.post("/setmeal/add.do?checkgroupIds=" +
this.checkgroupIds,this.formData).
  then((response)=> {
    this.dialogFormVisible = false;
    if(response.data.flag){
      this.$message({
        message: response.data.message,
        type: 'success'
      });
    }else{
      this.$message.error(response.data.message);
    }
  }).finally(()=> {
    this.findPage();
  });
}
```

2.3 后台代码

2.3.1 Controller

在SetmealController中增加方法

```
//新增
@RequestMapping("/add")
public Result add(@RequestBody Setmeal setmeal, Integer[] checkgroupIds){
  try {
    setmealService.add(setmeal,checkgroupIds);
  }catch (Exception e){
    //新增套餐失败
    return new Result(false,MessageConstant.ADD_SETMEAL_FAIL);
  }
  //新增套餐成功
  return new Result(true,MessageConstant.ADD_SETMEAL_SUCCESS);
}
```

2.3.2 服务接口

创建SetmealService接口并提供新增方法

```
package com.itheima.service;

import com.itheima.entity.PageResult;
import com.itheima.pojo.CheckGroup;
import com.itheima.pojo.Setmeal;
import java.util.List;
/**
 * 体检套餐服务接口
 */
public interface SetmealService {
    public void add(Setmeal setmeal, Integer[] checkgroupIds);
}
```

2.3.3 服务实现类

创建SetmealServiceImpl服务实现类并实现新增方法

```java
package com.itheima.service;

import com.alibaba.dubbo.config.annotation.Service;
import com.github.pagehelper.Page;
import com.github.pagehelper.PageHelper;
import com.itheima.dao.SetmealDao;
import com.itheima.entity.PageResult;
import com.itheima.pojo.Setmeal;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
/**
 * 体检套餐服务实现类
 */
@Service(interfaceClass = SetmealService.class)
@Transactional
public class SetmealServiceImpl implements SetmealService {
    @Autowired
    private SetmealDao setmealDao;

    //新增套餐
    public void add(Setmeal setmeal, Integer[] checkgroupIds) {
        setmealDao.add(setmeal);
        if(checkgroupIds != null && checkgroupIds.length > 0){
            //绑定套餐和检查组的多对多关系
            setSetmealAndCheckGroup(setmeal.getId(),checkgroupIds);
        }
    }
    //绑定套餐和检查组的多对多关系
    private void setSetmealAndCheckGroup(Integer id, Integer[]
checkgroupIds) {
        for (Integer checkgroupId : checkgroupIds) {
            Map<String,Integer> map = new HashMap<>();
            map.put("setmeal_id",id);
            map.put("checkgroup_id",checkgroupId);
            setmealDao.setSetmealAndCheckGroup(map);
        }

    }
```

```
}
```

2.3.4 Dao接口

创建SetmealDao接口并提供相关方法

```
package com.itheima.dao;

import com.itheima.pojo.Setmeal;
import java.util.Map;

public interface SetmealDao {
    public void add(Setmeal setmeal);
    public void setSetmealAndCheckGroup(Map<String, Integer> map);
}
```

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.itheima.dao.SetmealDao" >
    <!--新增-->
    <insert id="add" parameterType="com.itheima.pojo.Setmeal">
        <selectKey resultType="java.lang.Integer" order="AFTER"
keyProperty="id">
            SELECT LAST_INSERT_ID()
        </selectKey>
        insert into t_setmeal
            (code,name,sex,age,helpCode,price,remark,attention,img)
                values
            (#{code},#{name},#{sex},#{age},#{helpCode},#{price},#
{remark},#{attention},#{img})
    </insert>
    <!--绑定套餐和检查组多对多关系-->
    <insert id="setSetmealAndCheckGroup" parameterType="hashmap">
        insert into t_setmeal_checkgroup
            (setmeal_id,checkgroup_id)
                values
            (#{setmeal_id},#{checkgroup_id})
    </insert>
</mapper>
```

2.4 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spring-redis.xml

```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>
        <property name="testOnBorrow" value="true"/>
        <property name="testOnReturn" value="true"/>
    </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>
```

（2）在health_common工程中提供Redis常量类

```
package com.itheima.constant;

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

```
@Autowired
private JedisPool jedisPool;
//图片上传
@RequestMapping("/upload")
public Result upload(@RequestParam("imgFile")MultipartFile imgFile){
  try{
    //获取原始文件名
    String originalFilename = imgFile.getOriginalFilename();
    int lastIndexOf = originalFilename.lastIndexOf(".");
    //获取文件后缀
    String suffix = originalFilename.substring(lastIndexOf - 1);
    //使用UUID随机产生文件名称，防止同名文件覆盖
    String fileName = UUID.randomUUID().toString() + suffix;
    QiniuUtils.upload2Qiniu(imgFile.getBytes(),fileName);
    //图片上传成功
    Result result = new Result(true, MessageConstant.PIC_UPLOAD_SUCCESS);
    result.setData(fileName);
    //将上传图片名称存入Redis，基于Redis的Set集合存储

jedisPool.getResource().sadd(RedisConstant.SETMEAL_PIC_RESOURCES,fileName
);
    return result;
  }catch (Exception e){
    e.printStackTrace();
    //图片上传失败
    return new Result(false,MessageConstant.PIC_UPLOAD_FAIL);
  }
}
```

（4）在health_service_provider项目中提供Spring配置文件applicationContext-redis.xml

```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>
        <property name="testOnBorrow" value="true"/>
        <property name="testOnReturn" value="true"/>
    </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>
```

（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

```
@Autowired
private JedisPool jedisPool;
//新增套餐
public void add(Setmeal setmeal, Integer[] checkgroupIds) {
  setmealDao.add(setmeal);
  if(checkgroupIds != null && checkgroupIds.length > 0){
    setSetmealAndCheckGroup(setmeal.getId(),checkgroupIds);
  }
  //将图片名称保存到Redis
  savePic2Redis(setmeal.getImg());
}
//将图片名称保存到Redis
private void savePic2Redis(String pic){

jedisPool.getResource().sadd(RedisConstant.SETMEAL_PIC_DB_RESOURCES,pic);
}
```

3. 体检套餐分页

3.1 完善页面

3.1.1 定义分页相关模型数据

```
pagination: {//分页相关模型数据
  currentPage: 1,//当前页码
  pageSize:10,//每页显示的记录数
  total:0,//总记录数
  queryString:null//查询条件
},
dataList: [],//当前页要展示的分页列表数据
```

3.1.2 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

```
//钩子函数，VUE对象初始化完成后自动执行
created() {
  this.findPage();
}
```

```
//分页查询
findPage() {
  //分页参数
  var param = {
    currentPage:this.pagination.currentPage,//页码
    pageSize:this.pagination.pageSize,//每页显示的记录数
    queryString:this.pagination.queryString//查询条件
  };
  //请求后台
  axios.post("/setmeal/findPage.do",param).then((response)=> {
    //为模型数据赋值，基于VUE的双向绑定展示到页面
    this.dataList = response.data.rows;
    this.pagination.total = response.data.total;
  });
}
```

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

```
<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

```
<el-pagination
                class="pagiantion"
                @current-change="handleCurrentChange"
                :current-page="pagination.currentPage"
                :page-size="pagination.pageSize"
                layout="total, prev, pager, next, jumper"
                :total="pagination.total">
</el-pagination>
```

定义handleCurrentChange方法

```
//切换页码
handleCurrentChange(currentPage) {
  //currentPage为切换后的页码
  this.pagination.currentPage = currentPage;
  this.findPage();
}
```

3.2 后台代码

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){
  PageResult pageResult = setmealService.pageQuery(
    queryPageBean.getCurrentPage(),
    queryPageBean.getPageSize(),
    queryPageBean.getQueryString()
  );
  return pageResult;
}
```

3.2.2 服务接口

在SetmealService服务接口中扩展分页查询方法

```
public PageResult pageQuery(Integer currentPage, Integer pageSize, String
queryString);
```

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

```
public PageResult pageQuery(Integer currentPage, Integer pageSize, String
queryString) {
  PageHelper.startPage(currentPage,pageSize);
  Page<CheckItem> page = checkGroupDao.selectByCondition(queryString);
  return new PageResult(page.getTotal(),page.getResult());
}
```

3.2.4 Dao接口

在SetmealDao接口中扩展分页查询方法

```
public Page<Setmeal> selectByCondition(String queryString);
```

3.2.5 Mapper映射文件

在SetmealDao.xml文件中增加SQL定义

```
<!--根据条件查询-->
<select id="selectByCondition" parameterType="string"
resultType="com.itheima.pojo.Setmeal">
  select * from t_setmeal
  <if test="value != null and value.length > 0">
    where code = #{value} or name = #{value} or helpCode = #{value}
  </if>
</select>
```

4. 定时任务组件Quartz

4.1 Quartz介绍

Quartz是Job scheduling（作业调度）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。

官网：http://www.quartz-scheduler.org/

maven坐标：

```
<dependency>
  <groupId>org.quartz-scheduler</groupId>
  <artifactId>quartz</artifactId>
  <version>2.2.1</version>
</dependency>
<dependency>
  <groupId>org.quartz-scheduler</groupId>
  <artifactId>quartz-jobs</artifactId>
  <version>2.2.1</version>
</dependency>
```

4.2 Quartz入门案例

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

```xml
<?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">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.itheima</groupId>
    <artifactId>quartdemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>2.2.1</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz-jobs</artifactId>
            <version>2.2.1</version>
        </dependency>
    </dependencies>
</project>
```

（2）自定义一个Job

```
package com.itheima.jobs;
/**
 * 自定义Job
 */
public class JobDemo {
    public void run(){
        System.out.println("job execute...");
    }
}
```

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

```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">
    <!-- 注册自定义Job -->
    <bean id="jobDemo" class="com.itheima.jobs.JobDemo"></bean>
    <!-- 注册JobDetail,作用是负责通过反射调用指定的Job -->
    <bean id="jobDetail"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFacto
ryBean">
        <!-- 注入目标对象 -->
        <property name="targetObject" ref="jobDemo"/>
        <!-- 注入目标方法 -->
        <property name="targetMethod" value="run"/>
    </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>0/10 * * * * ?</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>
```

（4）编写main方法进行测试

```
package com.itheima.jobs.com.itheima.app;

import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
    public static void main(String[] args) {
        new ClassPathXmlApplicationContext("spring-jobs.xml");
    }
}
```

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。

4.3 cron表达式

上面的入门案例中我们指定了一个表达式：0/10 * * * * ?

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

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

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)

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

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/

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

```xml
<?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.itheima</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>health_jobs</artifactId>
    <packaging>war</packaging>
    <name>health_jobs Maven Webapp</name>
    <url>http://www.example.com</url>
    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.itheima</groupId>
            <artifactId>health_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz-jobs</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>
```

（2）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>
```

（3）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=info, stdout
```

（4）配置applicationContext-redis.xml

```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>
        <property name="testOnBorrow" value="true"/>
        <property name="testOnReturn" value="true"/>
    </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>
```

（5）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">
    <context:annotation-config></context:annotation-config>
    <bean id="clearImgJob" class="com.itheima.jobs.ClearImgJob"></bean>
    <bean id="jobDetail"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFacto
ryBean">
        <!-- 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>
        </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>
```

（6）创建ClearImgJob定时任务类

```
package com.itheima.jobs;

import com.itheima.constant.RedisConstant;
import com.itheima.utils.QiniuUtils;
import org.springframework.beans.factory.annotation.Autowired;
import redis.clients.jedis.JedisPool;
import java.util.Set;

/**
 * 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);
            }
        }
    }
}
```