

Day21

预约管理-套餐管理

图片存储方案

介绍

七牛云存储

注册、登录

新建存储空间

查看存储空间信息

开发者中心

鉴权

Java SDK 上传

封装工具类

新增套餐

需求分析

完善页面

弹出新增窗口

动态展示检查组列表

图片上传并预览

添加后台代码

控制层

服务接口

服务实现类

Dao接口

Mapper映射文件

Redis

Redis 简介

NoSQL 概念

问题现象

问题现象

现象特征

造成原因

解决思路

NoSQL 介绍

概念

特征

常见的 Nosql 数据库

应用场景-电商为例

Redis 概念

概念

特征

Redis 应用场景

Redis 下载与安装

下载 Redis

解压安装包

编译（在解压的目录中执行）

Redis 命令工具

Redis 服务器启动

Redis 服务器启动

Redis 客户端启动

Redis 基础环境设置约定

配置文件启动与常用配置

服务器端设定

客户端配置

日志配置

Redis 基本操作

信息读写

帮助信息

重点总结

Redis 数据类型

预约管理-套餐管理

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、七牛云等

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。

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>

注意：注册并登录成功后最好进行实名认证再进行相关操作。

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。

在控制台首页，选择资源管理下的新建存储空间：

The screenshot shows the Qiniu Cloud Management Console homepage. At the top, there's a 'Financial Center' section with metrics like available credit, unpaid orders, and monthly resource packages. Below that is the 'Resource Management' section, which includes categories like Storage Space, Acceleration Domains, Custom Data Processing, QVS Space, and Multimedia Workflows. A red box highlights the '+ Create Storage Space' button under the Storage Space category. At the bottom, there are sections for the Qiniu Control Panel Mini Program and CPS Promotion Refund.

接着填写名称：

新建存储空间

X

存储空间名称:

存储空间名称不允许重复，遇到冲突请更换名称。

名称格式为 3 ~ 63 个字符，可以包含小写字母、数字、短划线，且必须以小写字母或者数字开头和结尾。

存储区域: 华东 华北 华南 北美 东南亚

华东-浙江2 NEW

此空间将会在 华南 创建。

访问控制: 公开 私有

公开和私有仅对 Bucket 的读文件生效，修改、删除、写入等对 Bucket 的操作均需要拥有者的授权才能进行操作。

确定

取消

查看存储空间信息

存储空间创建后，会显示在存储空间列表菜单中，点击存储空间名称可以查看当前存储空间的相关信息：

对象存储

概览

空间管理

跨区域同步

统计分析

rushuni-test

存储区域: 华南 存储量: 0 对象数: 0 访问控制: 公开 空间类型: 自有空间

空间概览 文件管理 域名管理 图片样式 转码样式 空间设置

CDN 加速域名

自定义域名

域名 协议 类型 状态 操作

暂无数据

CDN 测试域名

七牛融合 CDN 测试域名 (以 cloudn.com/qiniudn.com/qns1.com/qbox.me 结尾), 每个域名每日限总流量 10GB, 每个测试域名自创建起30个自然日后系统会自动回收, 仅供测试使用并且不支持 Https 访问, 详情查看[七牛测试域名使用规范](#), 点击下列域名可查看每个域名剩余回收时间。

qxe9mq9iw.bn-bkt.cloudn.com

基础配置

访问控制 生命周期 空间授权 事件通知 镜像回源 S3 域名

点击进入 点击进入 点击进入 点击进入 点击进入 点击查看

标准存储 低频存储 归档存储

0 今日存储量 0 今日文件数 0/0 本月 API 请求次数 (GET/PUT) 0 本月空间外网下载流量 0 本月空间 CDN 回源流量

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/>



产品帮助文档

我们精心挑选了一些文档, 特别推荐给您, 它们都是点击率非常高的文档。
当然, 还有一些您可能以前没有看到的新文档。

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

The screenshot shows the Qiniu developer documentation website at <https://developer.qiniu.com/kodo>. The left sidebar is titled '开发者中心' and includes a '对象存储' section with links to '产品动态', '产品简介', '购买指南', '快速入门', '使用指南', 'API 文档', 'SDK 下载', '实用工具', '最佳实践', '服务协议', '常见问题', and '术语表'. The main content area has sections for '产品动态' (with a '产品更新动态' link and a note about price reduction from August 1, 2021), '产品简介' (with links to '产品概述', '存储类型', and '编程模型'), '购买指南' (with a '计量项与计费项' link), and '快速入门'.

七牛云提供了多种方式操作对象存储服务，我们选择 Java SDK 方式，地址：

<https://developer.qiniu.com/kodo/1239/java>

Java SDK

最近更新时间：2021-06-23 11:50:52

简介

此 SDK 适用于 Java 8 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. 提供七牛融合 CDN 相关的刷新、预取、日志功能
6. 提供七牛视频监控 QVDS 设备管理、流管理等功能

根据文档，我们需要导入如下依赖：

[XML](#) | [复制代码](#)

鉴权

位置位于个人中心：<https://portal.qiniu.com/user/key>

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`，这对密钥可以通过如下步骤获得：

1. 点击[注册](#)开通七牛开发者帐号
2. 如果已有账号，直接登录七牛开发者后台，点击[这里](#)查看 Access Key 和 Secret Key

Java SDK 上传

参考官方的 demo：

```
1 //构造一个带指定 Region 对象的配置类
2 Configuration cfg = new Configuration(Region.region0());
3 //...其他参数参考类注释
4 UploadManager uploadManager = new UploadManager(cfg);
5 //...生成上传凭证，然后准备上传
6 String accessKey = "your access key";
7 String secretKey = "your secret key";
8 String bucket = "your bucket name";
9 //如果是Windows情况下，格式是 D:\\qiniu\\\\test.png
10 String localFilePath = "/home/qiniu/test.png";
11 //默认不指定key的情况下，以文件内容的hash值作为文件名
12 String key = null;
13 Auth auth = Auth.create(accessKey, secretKey);
14 String upToken = auth.uploadToken(bucket);
15 try {
16     Response response = uploadManager.put(localFilePath, key, upToken);
17     //解析上传成功的结果
18     DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
19         DefaultPutRet.class);
20     System.out.println(putRet.key);
21     System.out.println(putRet.hash);
22 } catch (QiniuException ex) {
23     Response r = ex.response;
24     System.err.println(r.toString());
25     try {
26         System.err.println(r.bodyString());
27     } catch (QiniuException ex2) {
28         //ignore
29     }
30 }
```

Java

复制代码

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：

```
1 package com.rushuni.util;
2
3 import com.google.gson.Gson;
4 import com.qiniu.common.QiniuException;
5 import com.qiniu.http.Response;
6 import com.qiniu.storage.BucketManager;
7 import com.qiniu.storage.Configuration;
8 import com.qiniu.storage.Region;
9 import com.qiniu.storage.UploadManager;
10 import com.qiniu.storage.model.DefaultPutRet;
11 import com.qiniu.util.Auth;
12
13 /**
14 * 七牛云工具类
15 * @author rushuni
16 * @date 2021/08/06
17 */
18 public class QiniuUtils {
19     public static String accessKey = "keT4kivGZHLqwsVFZwdBwsjjXK-
f6TJ0Rmzc24tR";
20     public static String secretKey =
"JlP97s7iVA00979neBdXWAvHxNZDgjGgEa9ZMCKZ";
21     public static String bucket = "seehope-health-space-1";
22
23 /**
24 * 文件上传
25 * 最简单的就是上传本地文件，直接指定文件的完整路径即可上传。
26 * @param filePath
27 * @param fileName
28 */
29 public static void upload2Qiniu(String filePath, String fileName){
30     //构造一个带指定Zone对象的配置类
31     Configuration cfg = new Configuration(Region.region2());
32     UploadManager uploadManager = new UploadManager(cfg);
33     Auth auth = Auth.create(accessKey, secretKey);
34     String upToken = auth.uploadToken(bucket);
35     try {
36         Response response = uploadManager.put(filePath, fileName,
upToken);
37         //解析上传成功的结果
38         DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
DefaultPutRet.class);
39     } catch (QiniuException ex) {
40         Response r = ex.response;
41         try {
42             System.err.println(r.bodyString());
43         } catch (QiniuException ex2) {
44             //ignore
45         }
46     }
47 }
```

```
45         }
46     }
47 }
48
49 /**
50 * 字节数组上传
51 * 可以支持将内存中的字节数组上传到空间中。
52 * @param bytes
53 * @param fileName
54 */
55 public static void upload2Qiniu(byte[] bytes, String fileName){
56     //构造一个带指定Zone对象的配置类
57     Configuration cfg = new Configuration(Region.region2());
58     //...其他参数参考类注释
59     UploadManager uploadManager = new UploadManager(cfg);
60
61     //默认不指定key的情况下，以文件内容的hash值作为文件名
62     String key = fileName;
63     Auth auth = Auth.create(accessKey, secretKey);
64     String upToken = auth.uploadToken(bucket);
65     try {
66         Response response = uploadManager.put(bytes, key, upToken);
67         //解析上传成功的结果
68         DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
69             DefaultPutRet.class);
70         System.out.println(putRet.key);
71         System.out.println(putRet.hash);
72     } catch (QiniuException ex) {
73         Response r = ex.response;
74         System.err.println(r.toString());
75         try {
76             System.err.println(r.bodyString());
77         } catch (QiniuException ex2) {
78             //ignore
79         }
80     }
81
82     //删除文件
83     public static void deleteFileFromQiniu(String fileName){
84         //构造一个带指定Zone对象的配置类
85         Configuration cfg = new Configuration(Region.region2());
86         String key = fileName;
87         Auth auth = Auth.create(accessKey, secretKey);
88         BucketManager bucketManager = new BucketManager(auth, cfg);
89         try {
90             bucketManager.delete(bucket, key);
91         } catch (QiniuException ex) {
92             //如果遇到异常，说明删除成功
93             System.err.println(ex.code());
```

```
94         System.err.println(ex.response.toString());
95     }
96 }
97 }
```

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。

新建按钮绑定单击事件，对应的处理函数为handleCreate

HTML | 复制代码

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

JavaScript | 复制代码

```
1 // 重置表单
2 resetForm() {
3     this.formData = {};
4     this.activeName='first';
5     this.checkgroupIds = [];
6     this.imageUrl = null;
7 }
8 // 弹出添加窗口
9 handleCreate() {
10     this.dialogFormVisible = true;
11     this.resetForm();
12 }
```

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

JavaScript | 复制代码

```
1 tableData: [], // 添加表单窗口中检查组列表数据
2 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

JavaScript | 复制代码

```
1 <table class="datatable">
2   <thead>
3     <tr>
4       <th>选择</th>
5       <th>项目编码</th>
6       <th>项目名称</th>
7       <th>项目说明</th>
8     </tr>
9   </thead>
10  <tbody>
11    <tr v-for="c in tableData">
12      <td>
13        <input :id="c.id" v-model="checkgroupIds" type="checkbox"
14        :value="c.id">
15      </td>
16      <td><label :for="c.id">{{c.code}}</label></td>
17      <td><label :for="c.id">{{c.name}}</label></td>
18      <td><label :for="c.id">{{c.remark}}</label></td>
19    </tr>
20  </tbody>
21 </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

JavaScript | 复制代码

```
1 // 弹出添加窗口
2 handleCreate() {
3   this.resetForm();
4   this.dialogFormVisible = true;
5   // 发送ajax请，查询所有的检查组数据，转为json展示到当前新增窗口中
6   axios.get("/checkgroup/findAll.do").then((res) => {
7     if(res.data.flag){
8       //查询成功
9       this.tableData = res.data.data;
10    }else{
11      //查询失败
12      this.$message.error(res.data.message);
13    }
14  });
15},
```

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：

Java | 复制代码

```
1 //查询所有
2 @RequestMapping("/findAll")
3 public Result findAll(){
4     List<CheckGroup> checkGroupList = checkGroupService.findAll();
5     if(checkGroupList != null && checkGroupList.size() > 0){
6         Result result = new Result(true,
7             MessageConstant.QUERY_CHECKGROUP_SUCCESS);
8         result.setData(checkGroupList);
9         return result;
10    }
11 }
```

CheckGroupService:

Java | 复制代码

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

CheckGroupServiceImpl:

Java | 复制代码

```
1 public List<CheckGroup> findAll() {
2     return checkGroupDao.findAll();
3 }
```

CheckGroupDao:

Java | 复制代码

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

CheckGroupDao.xml:

XML | 复制代码

```
1 <select id="findAll" resultType="com.rushuni.pojo.CheckGroup">
2     select * from t_checkgroup
3 </select>
```

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：

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

JavaScript | 复制代码

2. 定义上传组件：

```
1 <!-- element-ui 的上传组件-->  
2 <!--  
3   el-upload: 上传组件  
4   action: 上传的提交地址  
5   auto-upload: 选中文件后是否自动上传  
6   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  
7   show-file-list: 是否显示已上传文件列表  
8   on-success: 文件上传成功时的钩子  
9   before-upload: 上传文件之前的钩子  
10 -->  
11 <el-upload  
12   class="avatar-uploader"  
13   action="/setmeal/upload.do"  
14   :auto-upload="autoUpload"  
15   name="imgFile"  
16   :show-file-list="false"  
17   :on-success="handleAvatarSuccess"  
18   :before-upload="beforeAvatarUpload">  
19   <!--用于上传图片预览-->  
20     
21   <!--用于展示上传图标-->  
22   <i v-else class="el-icon-plus avatar-uploader-icon"></i>  
23 </el-upload>
```

HTML | 复制代码

3. 定义对应的钩子函数：

```
1 // 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
2 handleAvatarSuccess(response, file) {
3     this.imageUrl = "http://qxebddxz2.hn-bkt.clouddn.com/" + response.data;
4     this.$message({
5         message: response.message,
6         type: response.flag ? 'success' : 'error'
7     });
8     // 设置模型数据 (图片名称), 后续提交ajax请求时会提交到后台最终保存到数据库
9     this.formData.img = response.data;
10 }
11 // 上传文件之前的钩子
12 beforeAvatarUpload(file) {
13     const isJPG = file.type === 'image/jpeg';
14     const isLt2M = file.size / 1024 / 1024 < 2;
15     if (!isJPG) {
16         this.$message.error('上传套餐图片只能是 JPG 格式!');
17     }
18     if (!isLt2M) {
19         this.$message.error('上传套餐图片大小不能超过 2MB!');
20     }
21     return isJPG && isLt2M;
22 }
```

4. 创建SetmealController，接收上传的文件：

Java | 复制代码

```
1  /**
2   * 套餐管理
3   */
4  @RestController
5  @RequestMapping("/setmeal")
6  public class SetmealController {
7      @Reference
8      private SetmealService setmealService;
9
10     //图片上传
11     @RequestMapping("/upload")
12     public Result upload(@RequestParam("imgFile") MultipartFile imgFile){
13         try{
14             //获取原始文件名
15             String originalFilename = imgFile.getOriginalFilename();
16             int lastIndexOf = originalFilename.lastIndexOf(".");
17             //获取文件后缀
18             String suffix = originalFilename.substring(lastIndexOf - 1);
19             //使用UUID随机产生文件名称, 防止同名文件覆盖
20             String fileName = UUID.randomUUID().toString() + suffix;
21             QiniuUtils.upload2Qiniu(imgFile.getBytes(),fileName);
22             //图片上传成功
23             Result result = new Result(true,
24             MessageConstant.PIC_UPLOAD_SUCCESS);
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     }
33 }
```

注意：另外在 spring 配置文件中配置文件上传组件

XML | 复制代码

```
1 <!--文件上传组件-->
2 <bean id="multipartResolver"
3
4     class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
5         <property name="maxUploadSize" value="104857600" />
6         <property name="maxInMemorySize" value="4096" />
7         <property name="defaultEncoding" value="UTF-8"/>
8     </bean>
```

5. 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

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

HTML

复制代码

完善handleAdd方法

```
1 // 添加
2 handleAdd () {
3     axios.post("/setmeal/add.do?checkgroupIds=" +
4         this.checkgroupIds, this.formData).
5     then((response)=> {
6         this.dialogFormVisible = false;
7         if(response.data.flag){
8             this.$message({
9                 message: response.data.message,
10                type: 'success'
11            });
12        }else{
13            this.$message.error(response.data.message);
14        }
15    }).finally(()=> {
16        this.findPage();
17    });
18 }
```

JavaScript

复制代码

添加后台代码

控制层

在 SetmealController 中增加方法

Java | 复制代码

```
1 //新增
2 @RequestMapping("/add")
3 public Result add(@RequestBody Setmeal setmeal, Integer[] checkgroupIds){
4     try {
5         setmealService.add(setmeal,checkgroupIds);
6     }catch (Exception e){
7         //新增套餐失败
8         return new Result(false,MessageConstant.ADD_SETMEAL_FAIL);
9     }
10    //新增套餐成功
11    return new Result(true,MessageConstant.ADD_SETMEAL_SUCCESS);
12 }
```

服务接口

创建SetmealService接口并提供新增方法

Java | 复制代码

```
1 /**
2  * 体检套餐服务接口
3  */
4 public interface SetmealService {
5     public void add(Setmeal setmeal, Integer[] checkgroupIds);
6 }
```

服务实现类

创建SetmealServiceImpl服务实现类并实现新增方法

Java | 复制代码

```
1  /**
2   * 体检套餐服务实现类
3   */
4  @Service(interfaceClass = SetmealService.class)
5  @Transactional
6  public class SetmealServiceImpl implements SetmealService {
7      @Autowired
8      private SetmealDao setmealDao;
9
10     //新增套餐
11     public void add(Setmeal setmeal, Integer[] checkgroupIds) {
12         setmealDao.add(setmeal);
13         if(checkgroupIds != null && checkgroupIds.length > 0){
14             //绑定套餐和检查组的多对多关系
15             setSetmealAndCheckGroup(setmeal.getId(),checkgroupIds);
16         }
17     }
18     //绑定套餐和检查组的多对多关系
19     private void setSetmealAndCheckGroup(Integer id, Integer[] checkgroupIds)
20     {
21         for (Integer checkgroupId : checkgroupIds) {
22             Map<String, Integer> map = new HashMap<>();
23             map.put("setmeal_id",id);
24             map.put("checkgroup_id",checkgroupId);
25             setmealDao.setSetmealAndCheckGroup(map);
26         }
27     }
28 }
```

Dao接口

创建SetmealDao接口并提供相关方法

Java | 复制代码

```
1  public interface SetmealDao {
2      public void add(Setmeal setmeal);
3      public void setSetmealAndCheckGroup(Map<String, Integer> map);
4  }
```

Mapper映射文件

创建SetmealDao.xml文件并定义相关SQL语句

XML | 复制代码

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3           "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
4 <mapper namespace="com.rushuni.dao.SetmealDao" >
5     <!--新增-->
6     <insert id="add" parameterType="com.rushuni.pojo.Setmeal">
7         <selectKey resultType="java.lang.Integer" order="AFTER"
keyProperty="id">
8             SELECT LAST_INSERT_ID()
9         </selectKey>
10    insert into t_setmeal
11        (code, name, sex, age, helpCode, price, remark, attention, img)
12        values
13        (#{{code}}, #{{name}}, #{{sex}}, #{{age}}, #{{helpCode}}, #{{price}}, #{{remark}}, #
attention}, #{{img}})
14    </insert>
15    <!--绑定套餐和检查组多对多关系-->
16    <insert id="setSetmealAndCheckGroup" parameterType="hashmap">
17        insert into t_setmeal_checkgroup
18            (setmeal_id, checkgroup_id)
19            values
20            (#{{setmeal_id}}, #{{checkgroup_id}})
21    </insert>
22 </mapper>
```

Redis

Redis 简介

NoSQL 概念

问题现象

在讲解NoSQL的概念之前呢，我们先来看一个现象：

问题现象

每年到了过年期间，大家都会自觉自发的组织一场活动，叫做春运！以前我们买票都是到火车站排队，后来有了12306，有了他以后就更方便了，我们可以在网上买票，但是带来的问题，大家也很清楚，春节期间买票进不去，进去了刷不着票。什么原因呢，人太多了！

除了这种做铁路的，它系统做的不专业以外，还有马爸爸做的淘宝，它面临一样的问题。淘宝也崩，也是用户量太大！作为我们整个电商界的东哥来说，他第一次做图书促销的时候，也遇到了服务器崩掉的这样一个现象，原因同样是因为用户量太大！

现象特征

再来看这几个现象，有两个非常相似的特征：

1. 用户比较多，海量用户
2. 高并发

这两个现象出现以后，对应的就会造成我们的服务器瘫痪。

核心本质是什么呢？其实并不是我们的应用服务器，而是我们的关系型数据库。

关系型数据库才是最终的罪魁祸首！

造成原因

什么样的原因导致的整个系统崩掉的呢：

1. 性能瓶颈：磁盘IO性能低下
关系型数据库在存取数据的时候和读取数据的时候都要走磁盘IO。磁盘这个性能本身是比较低的。
2. 扩展瓶颈：数据关系复杂，扩展性差，不便于大规模集群
我们说关系型数据库，它里面表与表之间的关系非常复杂，不知道大家能不能想象一点，就是一张表，通过它的外键关联了七八张表，这七八张表又通过她的外键，每张又关联了四五张表。这种情况下，查询一下，你要想拿到数据，你就要从 A 到 B、B 到 C、C 到 D 的一直这么关联下去，最终非常影响查询的效率。

同时，你想扩展下，也很难！

解决思路

面对这样的现象，我们要想解决怎么办呢。两方面：

1. 降低磁盘IO次数，越低越好。
2. 去除数据间关系，越简单越好。

第一种，降低磁盘IO次数，越低越好，怎么搞？

我不用你磁盘不就行了吗？

于是，内存存储的思想就提出来了。

我数据不放到你磁盘里边，放内存里，这样是不是效率就高了。

第二种，你的数据关系很复杂，那怎么办呢？

干脆简单点，我断开你的关系，我不存关系了，我只存数据，这样不就没这事了吗？

把这两个特征一合并一起，就出来了一个新的概念：NoSQL

NoSQL 介绍

概念

NoSQL：即 Not-Only SQL（泛指非关系型的数据库），作为关系型数据库的补充。

作用：应对基于海量用户和海量数据前提下的数据处理问题。

他说这句话说的非常客气，什么意思呢？就是我们数据存储要用SQL，但是呢可以不仅仅用SQL，还可以用别的东西，那别的东西叫什么呢？于是他定义了一句话叫做NoSQL。

这个意思就是说我们存储数据，可以不光使用SQL，我们还可以使用非SQL的这种存储方案，这就是所谓的NoSQL。

特征

可扩容，可伸缩。SQL数据关系过于复杂，你扩容一下难度很高，那我们Nosql 这种的，不存关系，所以它的扩容就简单一些。

大数据量下高性能。当数据非常多的时候，它的性能高，因为你不走磁盘IO，你走的是内存，性能肯定要比磁盘IO的性能快一些。

灵活的数据模型、高可用。他设计了自己的一些数据存储格式，这样能保证效率上来说是比较高的，最后一个高可用，我们等到集群内部分再去它！

常见的 Nosql 数据库

目前市面上常见的Nosql产品：

- Redis
- memcache
- MongoDB
- HBase
-

应用场景-电商为例

我们以电商为例，来看一看他在这里边起到的作用。

第一类，在电商中我们的基础数据一定要存储起来，比如说商品名称，价格，生产厂商，这些都属于基础数据，这些数据放在MySQL数据库。

第二类，我们商品的附加信息，比如说，你买了一个商品评价了一下，这个评价它不属于商品本身。就像你买一个苹果，“这个苹果很好吃”就是评论，但是你能说很好吃是这个商品的属性嘛？不能这么说，那只是一个人对他的评论而已。这一类数据呢，我们放在另外一个地方，我们放到MongoDB。它也可以用来加快我们的访问，他属于 NoSQL 的一种。

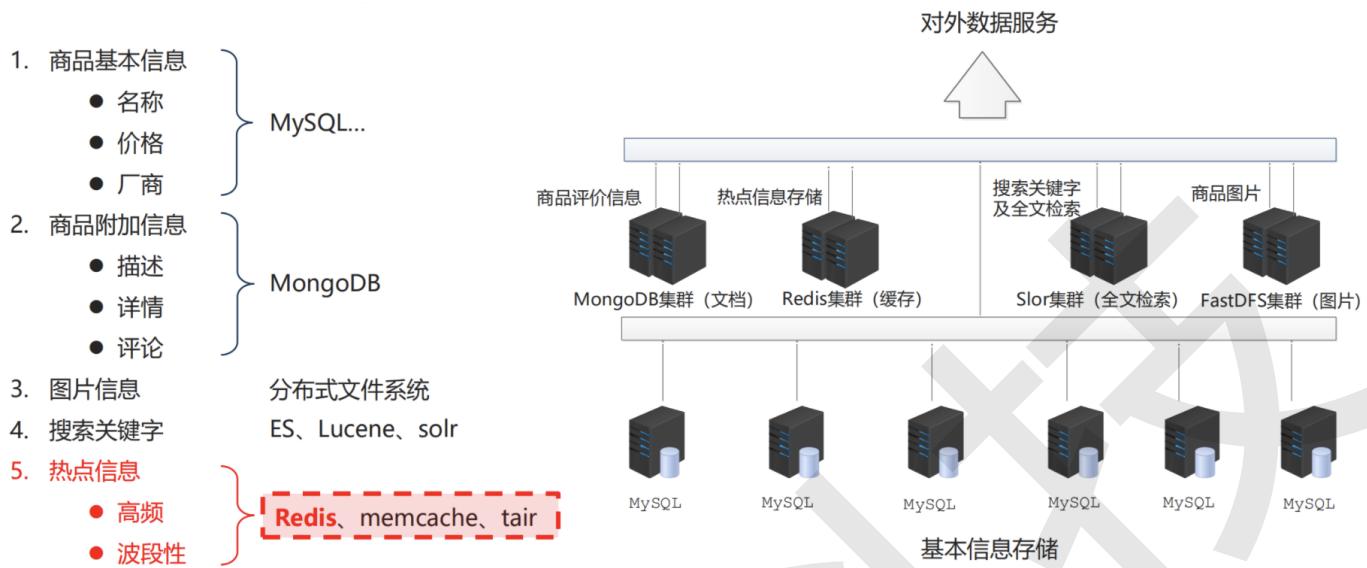
第三，图片内的信息。注意这种信息相对来说比较固定，他有专用的存储区，我们可以用文件系统的方式来存储。至于是不是分布式，要看你的系统的一个整个瓶颈了。如果说你发现你需要做分布式，那就做，不需要的话，一台主机就搞定了。

第四，搜索关键字。为了加快搜索，我们会用到一些技术，有些人可能了解过，像 ES、Lucene、solr 都属于搜索技术。那说的这么热闹，我们的电商解决方案中还没出现我们的redis 啊！注意第五类信息。

第五，热点信息。访问频度比较高的信息，这种东西的第二特征就是它具有波段性。

换句话说他不是稳定的，它具有一个时效性的。那么这类信息放哪儿了，放到我们的redis这个解决方案中来进行存储。

从整个电商的数据存储结构设计如下图：



我们的基础数据都存MySQL,在它的基础之上，我们把它连在一块儿，同时对外提供服务。

向上走，有一些信息加载完以后，要放到我们的MongoDB中。

还有一类信息，我们放到我们专用的文件系统中（比如图片），就放到我们的这个搜索专用的，如Lucene、solr及集群里边，或者用ES的这种技术里边。

那么剩下来的热点信息，放到我们的redis 里面。

Redis 概念

概念

概念：Redis (REmote DIctionary Server) 是用 C 语言开发的一个开源的高性能键值对 (key-value) 数据库。

特征

1. 数据间没有必然的关联关系；
2. 内部采用单线程机制进行工作；
3. 高性能。

- a. 官方提供测试数据，50个并发执行100000个请求，读的速度是110000次/s，写的速度是81000次/s。
- 4. 多数据类型支持
 - a. 字符串类型，string list
 - b. 列表类型，hash set
 - c. 散列类型，zset/sorted_set
 - d. 集合类型
 - e. 有序集合类型
- 5. 支持持久化，可以进行数据灾难恢复

Redis 应用场景

- 为热点数据加速查询（主要场景）。
 - 如热点商品、热点新闻、热点资讯、推广类等高访问量信息等。
- 即时信息查询。
 - 如各位排行榜、各类网站访问统计、公交到站信息、在线人数信息（聊天室、网站）、设备信号等。
- 时效性信息控制。
 - 如验证码控制、投票控制等。
- 分布式数据共享。
 - 如分布式集群架构中的 session 分离，消息队列。

Redis 下载与安装

下载 Redis

```
1 wget https://download.redis.io/releases/redis-6.2.5.tar.gz
```

Shell | 复制代码

解压安装包

```
1 tar -xvf redis-6.2.5.tar.gz
```

Shell | 复制代码

编译（在解压的目录中执行）

```
1 make
```

Shell | 复制代码

安装（在解压的目录中执行）

Shell | 复制代码

```
1 make install
```

Redis 命令工具

Redis安装完成后生成的几个可执行文件：

- redis-server
 - 服务器启动命令
- redis-cli
 - 客户端启动命令
- redis.conf
 - redis核心配置文件
- redis-check-dump
 - RDB文件检查工具（快照持久化文件）
- redis-check-aof
 - AOF文件修复工具

Redis 服务器启动

Redis服务器启动

- 启动服务器——参数启动
 - redis-server [--port port]
 - 例子

```
1 redis-server --port 6379
```

Shell | 复制代码

- 启动服务器——配置文件启动
 - redis-server config_file_name
 - 例子

```
1 redis-server redis.conf
```

Shell | 复制代码

Redis客户端启动

- 启动客户端
 - redis-cli [-h host] [-p port]

- 例子

Shell | 复制代码

```
1 redis-cli -h 61.129.65.248 -p 6384
```

注意：服务器启动指定端口使用的是--port，客户端启动指定端口使用的是-p，它们 - 的数量不同。

Redis基础环境设置约定

- 创建配置文件存储目录
 - mkdir conf
- 创建服务器文件存储目录（包含日志、数据、临时配置文件等）
 - mkdir data
- 创建快速访问链接
 - ln -s redis-6.2.5 redis

配置文件启动与常用配置

服务器端设定

- 设置服务器以守护进程的方式运行，开启后服务器控制台中将打印服务器运行信息
 - 这里的信息和打印的日志一样
 - daemonize yes|no
- 绑定主机地址
 - bind ip
- 设置服务器端口
 - port port
- 设置服务器文件保存地址
 - dir path

客户端配置

- 服务器允许客户端连接最大数量，默认0，表示无限制。
 - 当客户端连接数到达上限后，Redis会拒绝新的连接。
 - maxclients count
- 客户端闲置等待最大时长，达到最大值后关闭对应连接。
 - timeout seconds
 - 如需关闭该功能，设置为 0

日志配置

- 设置服务器以指定日志记录级别
 - loglevel debug|verbose|notice|warning
- 日志记录文件名
 - logfile filename

注意：日志级别开发期设置为 verbose 即可，生产环境中配置为 notice，简化日志输出量，降低写日志IO的频度。

Redis基本操作

信息读写

设置 key, value 数据

`set key value`

范例

`set name rushuni`

根据 key 查询对应的 value，如果不存在，返回空（nil）

`get key`

范例

`get name`

帮助信息

获取命令帮助文档

`help [command]`

范例

`help set`

获取组中所有命令信息名称

`help [@group-name]`

范例

`help @string`

1.6.4 退出命令行客户端模式

退出客户端

`quit exit`

快捷键

Ctrl+C

重点总结

Redis 的下载与安装，包括：

- 下载与安装

- 服务器与客户端启动
- 相关配置文件（3类）

Redis 的基本操作，包括：

- 包括数据读写
- 退出与帮助信息获取

Redis 数据类型

常用的有五种：字符串，哈希，列表，集合，有序集合。

查看资料熟悉相关操作。

