**品优购电商系统开发**

**第4章**

**商品分类管理&FastDFS**

# 课程目标

目标1：完成商品分类功能

目标2：掌握FastDFS的基本理论

# 1.商品分类管理

## 1.1**需求及表结构分析**

### 1.1.1**需求分析**

实现三级商品分类列表查询功能

进入页面首先显示所有的一级分类，效果如下：



点击列表行的查询下级按钮，进入下级分类列表，同时更新面包屑导航



再次点击表行的查询下级按钮，进入三级分类列表，因为三级分类属于最后一级，所以在列表中不显示查询下级按钮，同时更新面包屑导航



点击面包屑导航，可以进行返回操作。

### 1.1.2**表结构分析**

**tb\_item\_cat 商品分类表**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 长度 | 含义 |
| id | bigint |  | 主键 |
| parent\_id | bigint |  | 上级ID |
| name | varchar |  | 分类名称 |
| type\_id | bigint |  | 类型模板ID |

## 1.2**列表实现**

### 1.2.1**前端代码**

pinyougou-manager-web/src/main/webapp/admin/item\_cat.html引入js

|  |
| --- |
| *<!-- 引入vue与axios -->* <**script src="/js/vue.min.js"**></**script**> <**script src="/js/axios.min.js"**></**script**> <**script src="/js/controller/itemCatController.js"**></**script**> |

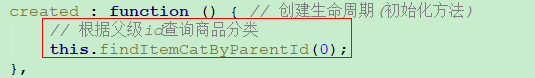
pinyougou-manager-web/src/main/webapp/js/controller/itemCatController.js

第一步：定义根据父级id查询商品分类、删除data中分页数据变量

|  |
| --- |
| findItemCatByParentId : **function**(parentId){ *// 根据父级id查询商品分类  // 发送异步请求* **axios**.get(**"/itemCat/findItemCatByParentId?parentId="** + parentId)  .then(**function**(response){  *// 获取响应数据* **vue.dataList = response.data;**  *// 设置ids数组为空* vue.**ids** = [];  }); }, |



第二步：在created方法中初始化调用



第三步：item\_cat.html页面用v-for循环

|  |
| --- |
| <**tr v-for="entity in dataList"**>  <**td**><**input type="checkbox"**></**td**>  <**td**>{{**entity**.**id**}}</**td**>  <**td**>{{**entity**.**name**}}</**td**>  <**td**>{{**entity**.**typeId**}}</**td**>  <**td class="text-center"**>  <**button type="button" class="btn bg-olive btn-xs"  @click="findItemCatByParentId(entity.id);"**>查询下级  </**button**>  <**button type="button" class="btn bg-olive btn-xs"  data-toggle="modal" data-target="#editModal"**>修改  </**button**>  </**td**> </**tr**> |

### 1.2.2**后端代码**

pinyougou-manager-web/src/main/java/com/pinyougou/manager/controller/包下创建ItemCatController.java

|  |
| --- |
| @RestController @RequestMapping(**"/itemCat"**) **public class** ItemCatController {  @Reference(timeout = 10000)  **private** ItemCatService **itemCatService**;  */\*\* 根据父级id查询商品分类 \*/* @GetMapping(**"/findItemCatByParentId"**)  **public** List<ItemCat> findItemCatByParentId(Long parentId){  **return itemCatService**.findItemCatByParentId(parentId);  } } |

pinyougou-interface/src/main/java/com/pinyougou/service/ItemCatService.java，增加方法

|  |
| --- |
| */\*\*  \* 根据父级id查询商品分类  \** ***@param parentId*** *父级id  \** ***@return*** *商品分类集合  \*/* List<ItemCat> findItemCatByParentId(Long parentId); |

pinyougou-sellergoods-service/src/main/java/com/pinyougou/sellergoods/service/impl/包下创建ItemCatServiceImpl.java

|  |
| --- |
| @Service(interfaceName = **"com.pinyougou.service.ItemCatService"**) @Transactional **public class** ItemCatServiceImpl **implements** ItemCatService {  @Autowired  **private** ItemCatMapper **itemCatMapper**;  @Override  **public void** save(ItemCat itemCat) {  }  @Override  **public void** update(ItemCat itemCat) {  }  @Override  **public void** delete(Serializable id) {  }  @Override  **public void** deleteAll(Serializable[] ids) {  }  @Override  **public** ItemCat findOne(Serializable id) {  **return null**;  }  @Override  **public** List<ItemCat> findAll() {  **return null**;  }  @Override  **public** List<ItemCat> findByPage(ItemCat itemCat, **int** page, **int** rows) {  **return null**;  }  */\*\* 根据父级id查询商品分类 \*/* @Override  **public** List<ItemCat> findItemCatByParentId(Long parentId){  **try**{  */\*\* 创建ItemCat封装查询条件 \*/* ItemCat itemCat = **new** ItemCat();  itemCat.setParentId(parentId);  **return itemCatMapper**.select(itemCat);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  }  } } |

## 1.3**面包屑导航**

我们需要返回上级列表，需要通过点击面包屑来实现。

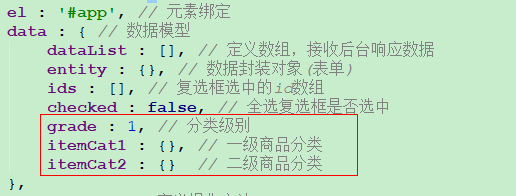
第一步：修改列表的查询下级按钮点击事件，设定级别值变量：grade

|  |
| --- |
| <**button type="button" class="btn bg-olive btn-xs"  v-if="grade!=3"  @click="selectList(entity, grade + 1);"**>查询下级</**button**> |

这里使用了v-if指令，用于条件判断，当级别不等于3的时候才显示“查询下级”按钮。

pinyougou-manager-web/src/main/webapp/js/controller/itemCatController.js

第二步：在数据模型中定义三个变量



第三步：在methods中定义方法

|  |
| --- |
| selectList : **function** (entity, grade) { *// 查询下级* **this**.**grade** = grade;  **if**(grade == 1){ *//如果为1级* **this**.**itemCat1** = {};  **this**.**itemCat2** = {};  } **else if**(grade == 2){ *//如果为2级* **this**.**itemCat1** = entity;  **this**.**itemCat2** = **{}**;  } **else if**(grade == 3){ *//如果为3级* **this**.**itemCat2** = entity;  }  */\*\* 查询下级商品分类列表 \*/* **this**.findItemCatByParentId(entity.**id**); } |

第四步：item\_cat.html页面绑定面包屑：

|  |
| --- |
| <**ol class="breadcrumb"**>  <**li**>  <**a href="#" @click="selectList({id:0}, 1)"**>顶级分类列表</**a**>  </**li**>  <**li v-if="grade > 1"**>  <**a href="#"  @click="selectList(itemCat1, 2)"**>{{itemCat1.name}}</**a**>  </**li**>  <**li v-if="grade > 2"**>  <**a href="#"  @click="selectList(itemCat2, 3)"**>{{itemCat2.name}}</**a**>  </**li**> </**ol**> |

## 1.4添加**商品分类**

实现商品分类，如下图:



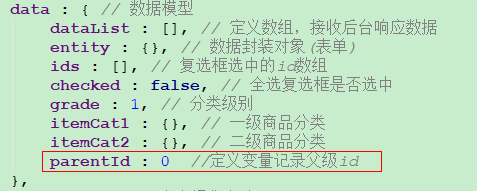
当前显示的是哪一分类的列表，我们就将这个商品分类新增到这个分类下。

### 1.4.1前端代码

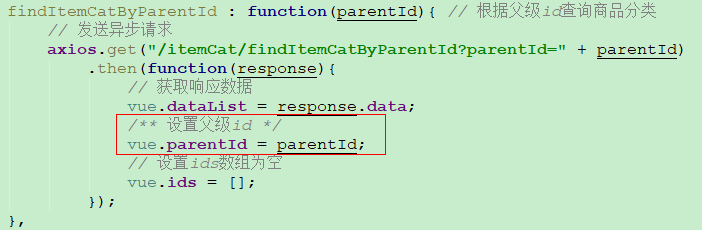
我们需要一个变量去记住上级ID，在保存的时候再根据这个ID来新增分类。

修改pinyougou-manager-web/src/main/webapp/js/controller的

itemCatController.js, 定义变量



查询下级时记录父级ID:



保存的时候，用到此变量：

|  |
| --- |
| saveOrUpdate : **function** () { *// 添加或修改* **var** url = **"save"**; *// 添加* **if** (**this**.**entity**.**id**){  url = **"update"**; *// 修改* }**else** {  */\*\* 添加时设置父级id \*/* **this**.**entity**.**parentId** = **this**.**parentId**;  }  *// 发送异步请求* **axios**.post(**"/itemCat/"** + url, **this**.**entity**)  .then(**function**(response){  *// 获取响应数据* **if** (response.**data**){ *// 操作成功  // 重新加载数据* vue.findItemCatByParentId(vue.**parentId**);  }**else** {  *alert*(**'操作失败！'**);  }  }); }, |

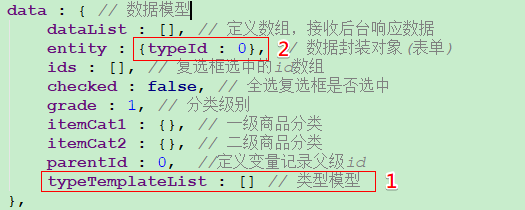
修改页面item\_cat.html

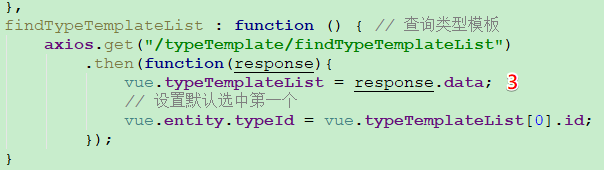
|  |
| --- |
| <**div class="modal-body"**>  <**table class="table table-bordered table-striped"  width="800px"**>  <**tr**>  <**td**>上级商品分类</**td**>  <**td**>  <**span v-if="grade >= 1"**>顶级分类</**span**>  <**span v-if="grade > 1"**>>> {{**itemCat1**.**name**}}</**span**>  <**span v-if="grade > 2"**>>> {{**itemCat2**.**name**}}</**span**>  </**td**>  </**tr**>  <**tr**>  <**td**>商品分类名称</**td**>  <**td**><**input class="form-control" v-model="entity.name"  placeholder="商品分类名称"**> </**td**>  </**tr**>  <**tr**>  <**td**>类型模板</**td**>  <**td**>  <**input select2 config="options['type\_template']"   placeholder="商品类型模板"   class="form-control" type="text"**/>  </**td**>  </**tr**>  </**table**> </**div**> <**div class="modal-footer"**>  <**button class="btn btn-success" data-dismiss="modal"  @click="saveOrUpdate()" aria-hidden="true"**>保存</**button**>  <**button class="btn btn-default" data-dismiss="modal"  aria-hidden="true"**>关闭</**button**> </**div**> |

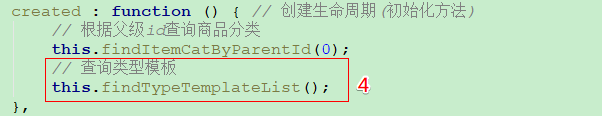
### 1.4.2**类型模板选择**

第一步：实现类型模板下拉列表前端代码

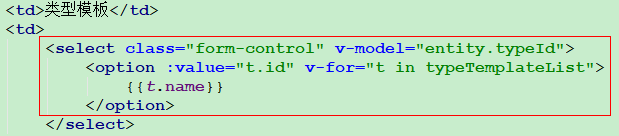
**itemCatController.js文件:**







**item\_cat.html**页面数据迭代，产生类型模板下拉列表



第二步：实现类型模板下拉列表后端代码

|  |
| --- |
| **TypeTemplateController.java：**  */\*\* 查询类型模版 \*/* @GetMapping(**"/findTypeTemplateList"**) **public** List<Map<String, Object>> findTypeTemplateList(){  **return typeTemplateService**.findTypeTemplateList(); }  **TypeTemplateService.java：**  */\*\* 查询类型模版 \*/* List<Map<String, Object>> findTypeTemplateList();  **TypeTemplateServiceImpl.java：**  */\*\* 查询类型模版 \*/* **public** List<Map<String, Object>> findTypeTemplateList(){  **try**{  **return typeTemplateMapper**.findTypeTemplateList();  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } }  **TypeTemplateMapper.java：**  */\*\* 查询类型模版 \*/* @Select(**"select id,name from tb\_type\_template order by id asc"**) List<Map<String, Object>> findTypeTemplateList(); |

### 1.4.3后端代码

|  |
| --- |
| **ItemCatController.java:**  */\*\* 添加商品分类 \*/* @PostMapping(**"/save"**) **public boolean** save(@RequestBody ItemCat itemCat){  **try**{  **itemCatService**.save(itemCat);  **return true**;  }**catch**(Exception ex){  ex.printStackTrace();  }  **return false**; }  **ItemCatService.java:**  */\*\* 添加商品分类 \*/* **void** save(ItemCat itemCat);  **ItemCatServiceImpl.java:**  */\*\* 添加商品分类 \*/* **public void** save(ItemCat itemCat){  **try**{  **itemCatMapper**.insertSelective(itemCat);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } } |

## 1.5**修改商品分类**

### 1.2.1**前端代码**

item\_cat.html页面上的“修改”按钮绑定点击事件

|  |
| --- |
| <**button type="button" class="btn bg-olive btn-xs"  @click="show(entity);"  data-toggle="modal" data-target="#editModal"**>修改 </**button**> |

### 1.2.1后端**代码**

|  |
| --- |
| **ItemCatController.java：**  */\*\* 修改商品分类 \*/* @PostMapping(**"/update"**) **public boolean** update(@RequestBody ItemCat itemCat){  **try**{  **itemCatService**.update(itemCat);  **return true**;  }**catch**(Exception ex){  ex.printStackTrace();  }  **return false**; }  **ItemCatService.java：**  */\*\* 修改商品分类 \*/* **void** update(ItemCat itemCat);  **ItemCatServiceImpl.java：**  */\*\* 修改商品分类 \*/* **public void** update(ItemCat itemCat){  **try**{  **itemCatMapper**.updateByPrimaryKeySelective(itemCat);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } } |

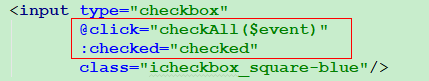
## 1.6**删除商品分类**

### 1.6.1**前端代码**

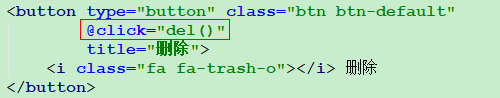
第一步：item\_cat.html页面为复选框绑定数据

|  |
| --- |
| <**td**><**input type="checkbox"  :value="entity.id" v-model="ids"**></**td**> |

第二步：item\_cat.html页面全选复选框



第三步：item\_cat.html页面删除按钮绑定事件



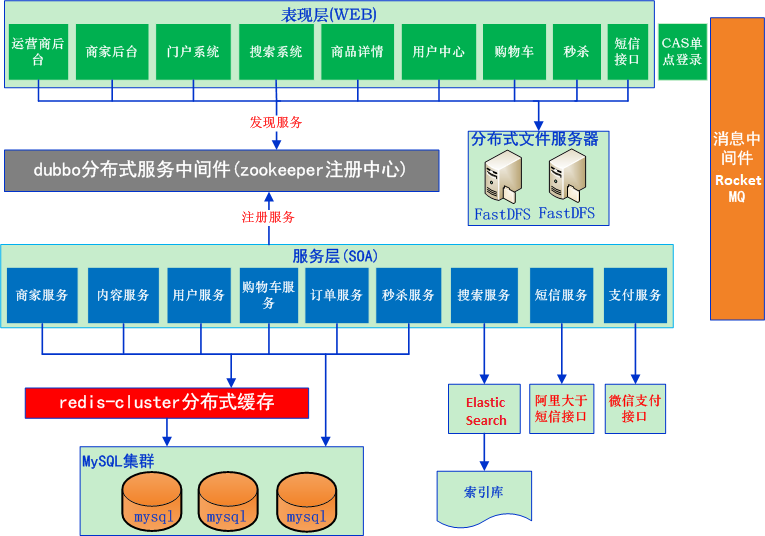
第四步：itemCatController.js文件

|  |
| --- |
| del : **function** () { *// 删除* **if** (**this**.**ids**.length > 0){  **axios**.get(**"/itemCat/delete?ids="** + **this**.**ids**).then(**function**(response){  **if** (response.**data**){  *// 重新加载数据* **vue.findItemCatByParentId(vue.parentId);**  }**else**{  *alert*(**"删除失败！"**);  }  });  }**else** {  *alert*(**"请选择要删除的记录！"**);  } }, |

### 1.6.2**后端代码**

|  |
| --- |
| **ItemCatController.java：**  */\*\* 删除商品分类 \*/* @GetMapping(**"/delete"**) **public boolean** delete(Long[] ids){  **try**{  **itemCatService**.deleteAll(ids);  **return true**;  }**catch**(Exception ex){  ex.printStackTrace();  }  **return false**; }  **ItemCatServiceImpl.java：**  @Override **public void** deleteAll(Serializable[] ids){  **try**{  */\*\* 定义List集合 \*/* List<Long> idLists = **new** ArrayList<>();  **for** (Serializable id : ids){  */\*\* 添加id \*/* idLists.add((Long)id);  */\*\* 递归查询 \*/* findLeafNode(id, idLists);  }  */\*\* 批量删除商品分类 \*/* Example example = **new** Example(ItemCat.**class**);  *// 创建条件对象* Example.Criteria criteria = example.createCriteria();  *// 封装in条件* criteria.andIn(**"id"**, idLists);  *// 条件删除* **itemCatMapper**.deleteByExample(example);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } }  */\*\* 递归查询所有的子节点 \*/* **private void** findLeafNode(Serializable id, List<Long> idLists){  */\*\* 根据父节点查询所有的子节点 \*/* List<ItemCat> itemCatLists = findItemCatByParentId((Long)id);  */\*\* 判断集合是否为空，也是递归退出条件 \*/* **if** (itemCatLists != **null** && itemCatLists.size() > 0){  **for** (ItemCat itemCat : itemCatLists){  idLists.add(itemCat.getId());  findLeafNode(itemCat.getId(), idLists);  }  } } |

# 2.FastDFS【分布式文件系统】



## **2.1图片服务器介绍**

要实现图片上传功能，需要有一个图片服务器。图片服务器的特点：

1. 存储空间可扩展
2. 提供统一的访问方式
3. 访问效率高

企业中的图片服务器有两个方案：

1. 使用FastDFS分布式文件系统。
2. 使用Nginx服务器。

## **2.2什么是FastDFS**

FastDFS(Fast Distributed file system)是用c语言编写的一款开源的分布式文件系统。FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、横向扩展等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建**一套高性能的文件服务器集群提供文件上传、下载等服务**。

## **2.3 FastDFS架构**

FastDFS架构包括 Tracker server和Storage server。客户端请求Tracker server进行文件上传、下载，通过Tracker server调度最终由Storage server完成文件上传和下载。

Tracker server作用是负载均衡和调度，通过Tracker server在文件上传时可以根据一些策略找到Storage server提供文件上传服务。可以将tracker称为追踪服务器或调度服务器。

Storage server作用是文件存储，客户端上传的文件最终存储在Storage服务器上，Storage server**没有实现自己的文件系统而是利用操作系统的文件系统来管理文件**。可以将storage称为存储服务器。

如下图：



Tracker 管理集群，收集信息，处理信息，为了保证高可用，可以搭建集群。

Storage 保存文件，分为很多组，组和组之间的数据不一样，组内成员数据是一样的，保证数据的高可用，可以增加组，达到扩容的效果。

## **2.4文件上传流程**

文件上传流程如下图（时序图）：



客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。



* 组名：文件上传后所在的storage组名称，在文件上传成功后有storage服务器返回，需要客户端自行保存。
* 虚拟磁盘路径：storage配置的虚拟路径，与磁盘选项store\_path\*对应。如果配置了store\_path0则是M00，如果配置了store\_path1则是M01，以此类推。
* 数据两级目录：storage服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。两级目录的范围都是 00~FF
* 文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器IP地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

## **2.5文件下载流程**

文件下载流程如下图：



## **2.6架构图**



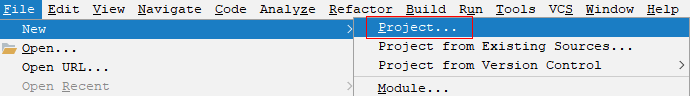
## **2.7安装FastDFS【运维】**

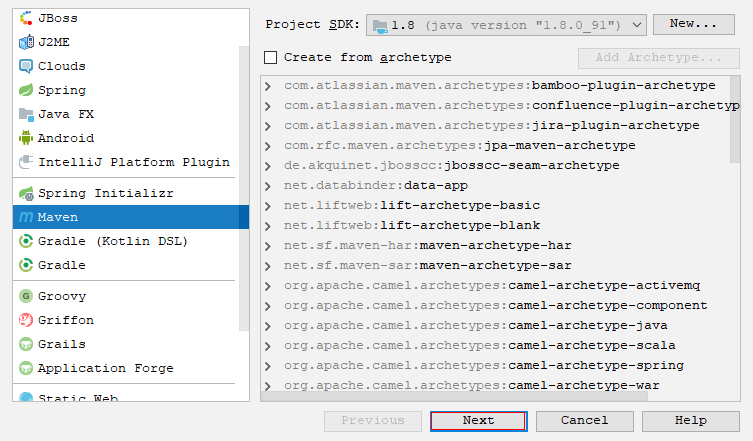
在导入的虚拟机中已经安装；

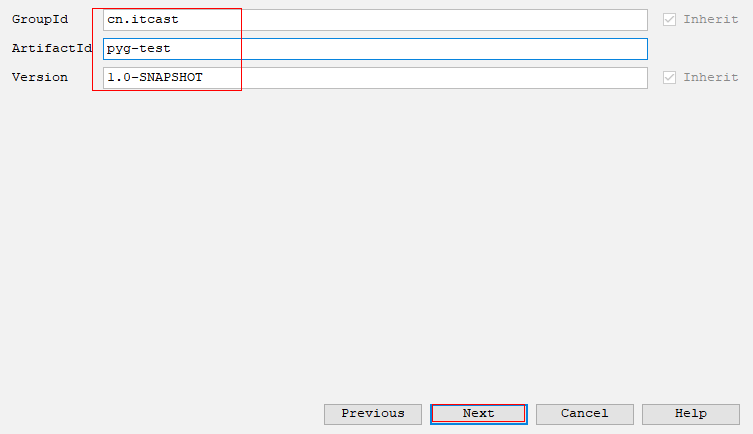
若需自行安装详见“【资料\FastDFS\文档\ **Linux安装FastDFS.docx**】”

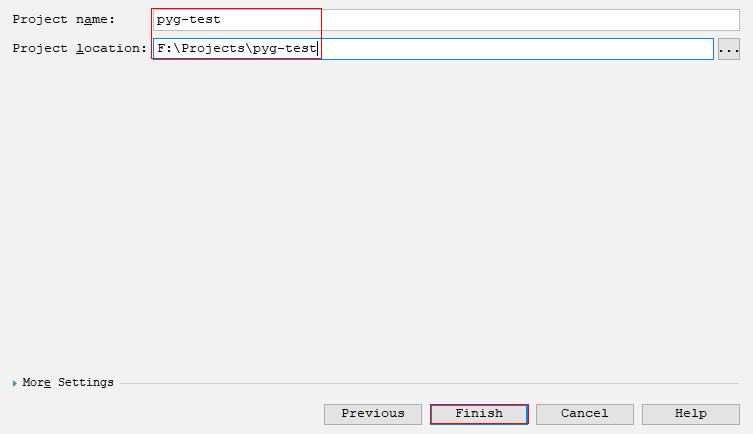
# 3.FastDFS客户端【上传&下载】

## **3.1创建pyg-test项目**











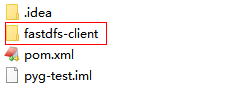
pyg-test\pom.xml，添加如下配置

|  |
| --- |
| <**packaging**>pom</**packaging**>  <**dependencies**>  <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  <**scope**>test</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>log4j</**groupId**>  <**artifactId**>log4j</**artifactId**>  <**version**>1.2.17</**version**>  </**dependency**> </**dependencies**>  *<!-- 构建部分 -->* <**build**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**version**>3.1</**version**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  <**encoding**>UTF-8</**encoding**>  </**configuration**>  </**plugin**>  </**plugins**> </**build**> |

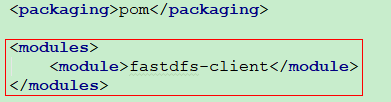
## **3.2 fastdfs-client客户端**

学习使用FastDFS需要有对应的Java客户端来支持；而遗憾的是FastDFS的作者没有把FastDFS的依赖jar放置到maven中央仓库提供下载；但可以自行导入项目并安装，然后再引用它。

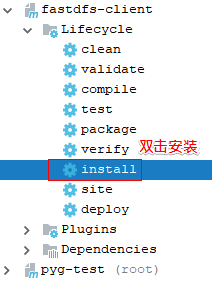
将“资料\FastDFS\java客户端\fastdfs-client”复制到pyg-test项目下。



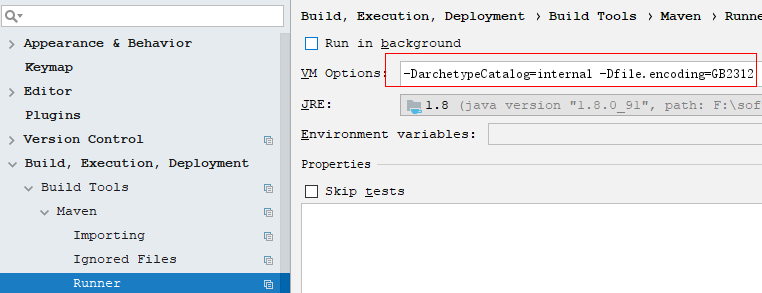
配置pyg-test的pom.xml：



安装jar包到本地仓库：



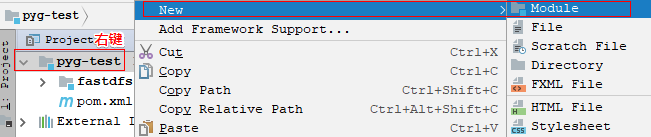
解决Maven控制台输出乱码问题：

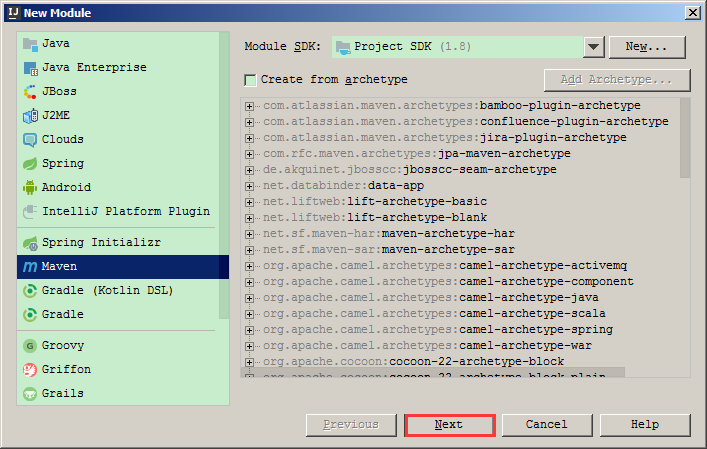


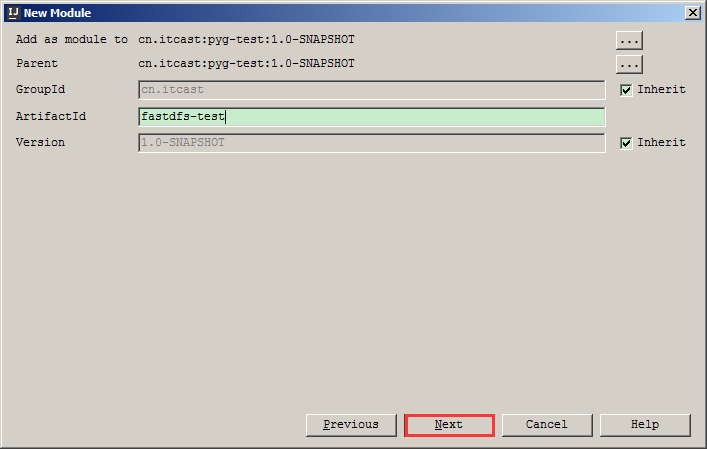
**参数：-DarchetypeCatalog=internal -Dfile.encoding=GB2312**

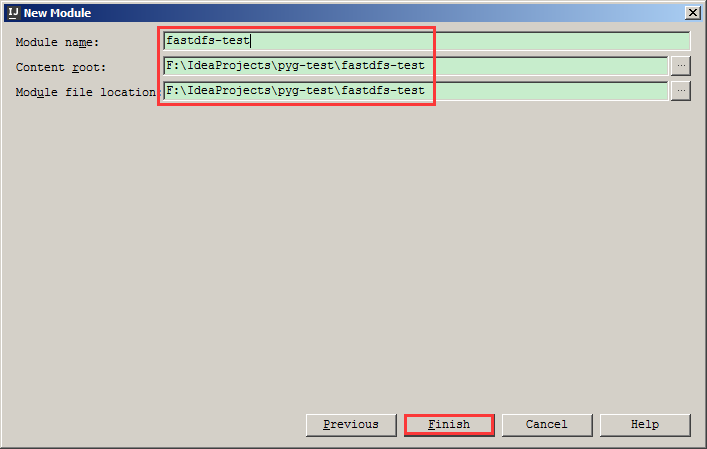
## **3.3 fastdfs-test测试模块**

创建fastdfs-test模块(jar)









fastdfs-test/pom.xml依赖信息如下：

|  |
| --- |
| <**dependencies**>  *<!-- fastdfs-client -->* <**dependency**>  <**groupId**>org.csource</**groupId**>  <**artifactId**>fastdfs-client</**artifactId**>  <**version**>1.25-RELEASE</**version**>  </**dependency**>  </**dependencies**> |

## **3.4上传文件**

### **3.4.1配置tracker server地址**

新增fastdfs-test/src/main/resources/fastdfs\_client.conf文件：

|  |
| --- |
| # 配置追踪服务器连接地址 tracker\_server=192.168.12.131:22122 |

### **3.4.2编写上传代码**

提供fastdfs-test/src/test/java/cn/itcast/fastdfs/FastdfsTest.java

|  |
| --- |
| **public class** FastdfsTest {  */\*\* 上传文件 \*/* @Test  **public void** upload\_file() **throws** Exception{  */\*\* 加载配置文件，产生该文件的绝对路径 \*/* String conf\_filename = **this**.getClass()  .getResource(**"/fastdfs\_client.conf"**).getPath();  */\*\* 初始化客户端全局的对象 \*/* ClientGlobal.*init*(conf\_filename);  */\*\* 创建存储客户端对象 \*/* StorageClient storageClient = **new** StorageClient();  */\*\* 上传文件 \*/* **String[] arr = storageClient.upload\_file("C:/Users/Public/Pictures/Sample Pictures/15.jpg", "jpg", null);**  */\*\*  \* 访问路径：http://192.168.12.131/group1/M00/00/00/wKgMg1o53fOAL1CRAABonuLw4M4127.jpg  \* [group1, M00/00/00/wKgMgFlIkk2AHfnLAABonuLw4M4075.jpg]  \* 数组中的第一个元素：组的名称  \* 数组中的第二个元素：远程文件名称  \*/* System.***out***.println(Arrays.*toString*(arr));  }  } |

**访问地址：**

**http://192.168.12.131/M00/00/00/KgMgFlHqumASKsWAABonuLw4M4076.jpg**

### **3.4.3上传步骤小结**

1. 加载tracker server的配置文件地址，配置文件为tracker server的地址；在这里设置了tracker\_server=192.168.12.131:22122
2. 初始化客户端全局信息ClientGlobal.init(conf)
3. 创建存储客户端StorageClient
4. 调用storageClient的upload\_file()方法上传图片
5. 解析并获取上传结果，返回组名和文件的相对路径

## **3.5下载文件**

fastdfs-test/src/test/java/cn/itcast/fastdfs/FastdfsTest.java，新增download\_file()方法

|  |
| --- |
| */\*\* 文件下载 \*/* @Test **public void** download\_file() **throws** Exception{  */\*\* 加载配置文件，产生绝对路径 \*/* String conf\_filename = **this**.getClass()  .getResource(**"/fastdfs\_client.conf"**).getPath();  */\*\* 初始化客户端全局的对象 \*/* ClientGlobal.*init*(conf\_filename);  */\*\* 创建存储客户端对象 \*/* StorageClient storageClient = **new** StorageClient();  */\*\* 下载文件 \*/* **byte**[] data = storageClient.download\_file(**"group1"**, **"M00/00/00/wKgMgFlIkk2AHfnLAABonuLw4M4075.jpg"**);  System.***out***.println(data.**length**);  FileOutputStream fos = **new** FileOutputStream(**new** File(**"F:/workspace/java/fastdfs-test/src/test/resources/1.jpg"**));  fos.write(data);  fos.flush();  fos.close(); } |

IDEA中查看项目磁盘路径：



## **3.6删除文件**

fastdfs-test/src/test/java/cn/itcast/fastdfs/FastdfsTest.java，新增delete\_file()方法

|  |
| --- |
| */\*\* 删除文件 \*/* @Test **public void** delete\_file() **throws** Exception{  */\*\* 加载配置文件，产生绝对路径 \*/* String conf\_filename = **this**.getClass()  .getResource(**"/fastdfs\_client.conf"**).getPath();  */\*\* 初始化客户端全局的对象 \*/* ClientGlobal.*init*(conf\_filename);  */\*\* 创建存储客户端对象 \*/* StorageClient storageClient = **new** StorageClient();  */\*\* 删除文件 \*/* **int** res = storageClient.delete\_file(**"group1"**,  **"M00/00/00/wKgMgFlIkk2AHfnLAABonuLw4M4075.jpg"**);  System.***out***.println(res); } |