# 商品信息维护

## 通用Mapper介绍

### Mybatis存在的问题

优点:实现了(半)自动化的ORM(对象关系映射),查询的结果可以自动的映射到对象.

缺点:即使再简单的sql都需要自己手动编辑.

### 持久层框架的发展

1. JDBC:每次链接时都需要手动的连接数据库.中间需要经过复杂的过程,实现数据的查询/更新操作.
2. 连接池:提前初始化一部分链接,供JDBC使用.可以在一定的程度上提高sql的执行的效率. C3p0/bandcp/druid
3. JPA思想:

JPA是Java Persistence API的简称，中文名**Java持久层API**，是JDK 5.0注解或XML描述**对象－关系表的映射关系**，并将运行期的实体[对象持久化](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1%E6%8C%81%E4%B9%85%E5%8C%96/7316192)到数据库中。[1]

概括:采用面向对象的方式操作数据库.

1. hibernate框架

优点:全部采用了面向对象的方式操作数据库.并且实现了全自动的ORM(对象关系映射),操作数据库时,sql几乎可以省略不写.操作的都是对象.

缺点:

1. 通过对象自动生成的sql,会有大量的冗余,尤其在多表关联时.

提供了hql语句.hql语句写法复杂.需要重新学习.

1. 操作的速度较慢.使用了大量的反射的机制.500万.
2. Mybatis

优点:sql语句自己手写.同时实现了对象关系映射.

优化:能否实现单表CURD操作,使用对象的方式进行操作????

### 实现思路

1. 将对象与数据库中的表一一映射(配置文件/注解)
2. 定义公共的接口/内部定义CRUD接口方法.
3. 通过实现类实现接口中的方法.
4. 将对象最终转化为Sql语句.

**Insert into** 表名**(**字段名称….**.)** **values** **(**value1,value2,value3….**)**

**Select** 字段名称…. **From** 表名 **where** xxxx = vvvvv

### 通用Mapper介绍

1. 导入jar包

<!-- 通用Mapper -->

<dependency>

<groupId>com.github.abel533</groupId>

<artifactId>mapper</artifactId>

<version>${mapper.version}</version>

</dependency>

2.导入插件

<plugins>

<!-- 通用Mapper插件 -->

<plugin interceptor=*"com.github.abel533.mapperhelper.MapperInterceptor"*>

<!--主键自增回写方法,默认值MYSQL,详细说明请看文档 -->

<property name=*"IDENTITY"* value=*"MYSQL"* />

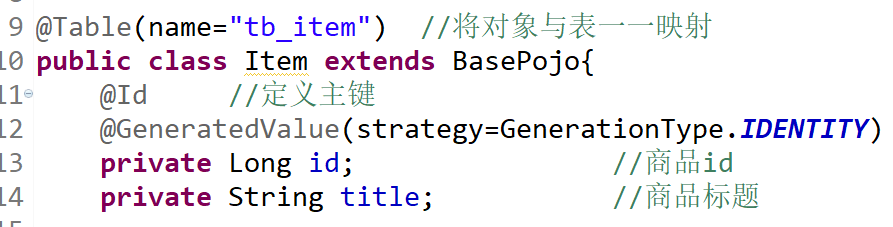
<!--通用Mapper接口，多个通用接口用逗号隔开 -->

<property name=*"mappers"* value=*"com.jt.common.mapper.SysMapper"* />

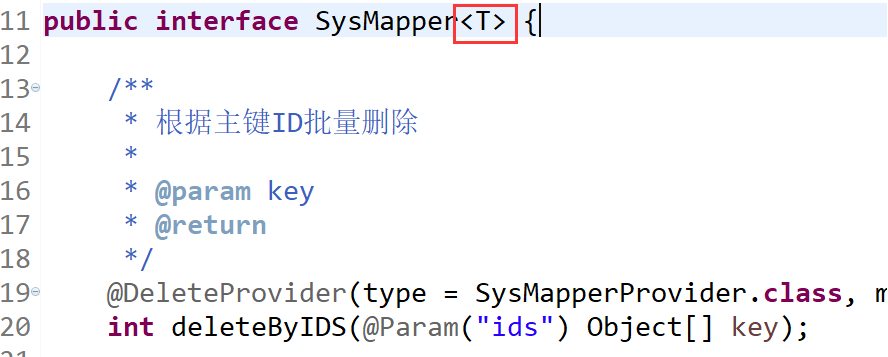
</plugin>

</plugins>

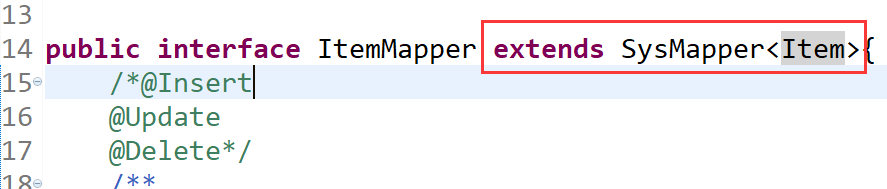
### 配置映射关系



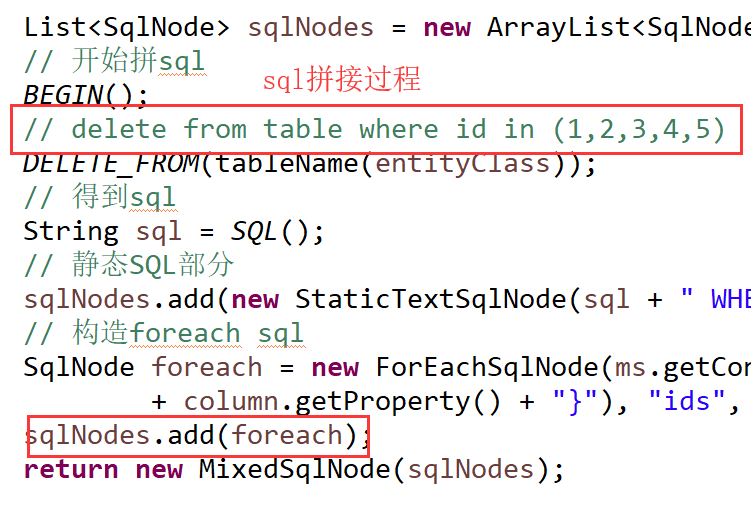
### 定义公共的接口



### 继承公共的接口



### 将接口方法转化为sql(了解)



### 通用Mapper测试

1.根据通用Mapper查询记录总数

1. int total = itemMapper.selectCount(null);

2.Sql语句:

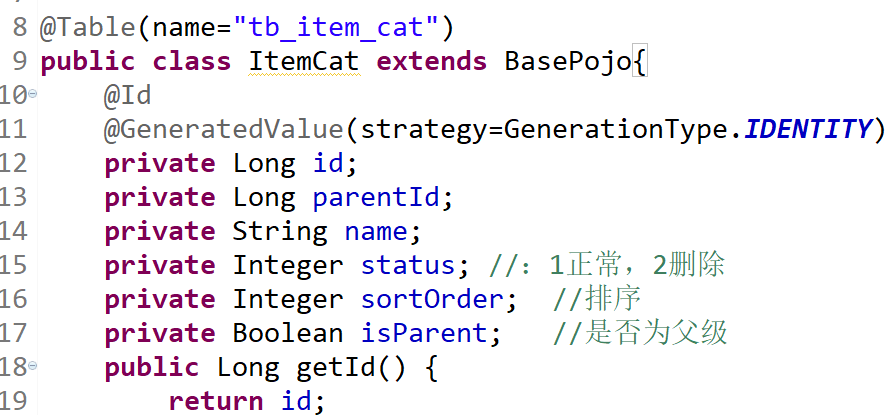
[DEBUG] ==> Preparing: **SELECT COUNT(\*) FROM tb\_item**

2018-11-01 10:45:02,623 [http-bio-8091-exec-1] [com.jt.manage.mapper.ItemMapper.selectCount]-[DEBUG] ==> Parameters:

2018-11-01 10:45:02,761 [http-bio-8091-exec-1] [com.jt.manage.mapper.ItemMapper.selectCount]-[DEBUG] <== Total: 1

## 商品分类目录展现

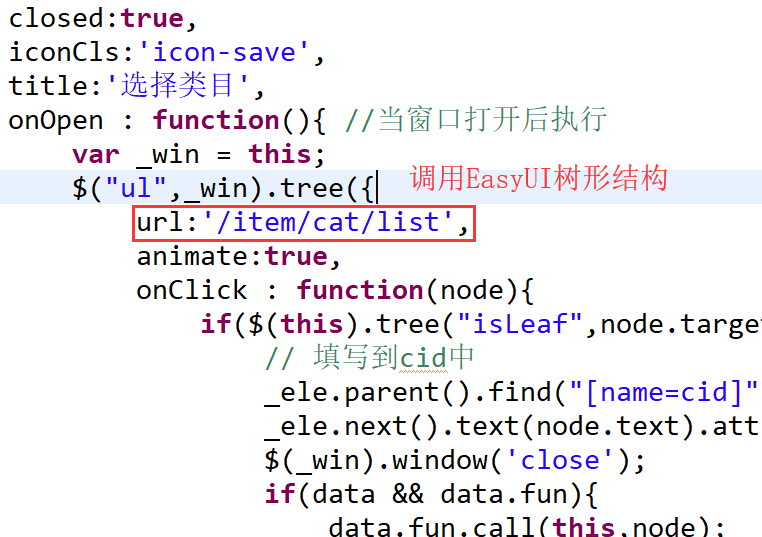
### 编辑pojo



### 需求分析

说明:商品分类目录展现,通过弹出框和树形结构共同展现.所以需要学习树形结构如何展现.

1. 页面分析



1. EasyUI中树形结构

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>EasyUI-3-菜单按钮</title>

<script type=*"text/javascript"*

src=*"/js/jquery-easyui-1.4.1/jquery.min.js"*></script>

<script type=*"text/javascript"*

src=*"/js/jquery-easyui-1.4.1/jquery.easyui.min.js"*></script>

<script type=*"text/javascript"*

src=*"/js/jquery-easyui-1.4.1/locale/easyui-lang-zh\_CN.js"*></script>

<link rel=*"stylesheet"* type=*"text/css"*

href=*"/js/jquery-easyui-1.4.1/themes/icon.css"* />

<link rel=*"stylesheet"* type=*"text/css"*

href=*"/js/jquery-easyui-1.4.1/themes/default/easyui.css"* />

<script type=*"text/javascript"*>

/\*通过js创建树形结构 \*/

$(**function**(){

$("#tree").tree({

url:"tree.json", //加载远程JSON数据

method:"POST", //请求方式 POST

animate:**false**, //表示显示折叠端口

checkbox:**true**, //表述复选框

lines:**false**, //表示显示连接线

dnd:**true**, //是否拖拽

onClick:**function**(node){ //添加点击事件

//控制台

console.info(node);

}

});

})

</script>

</head>

<body>

<h1>EasyUI-树形结构</h1>

<ul id=*"tree"*></ul>

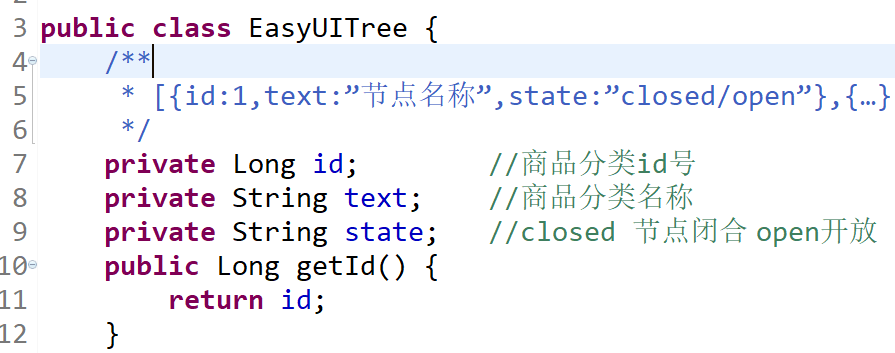
</body>

</html>

### 准备VO对象

说明:根据树形结构展现数据.需要准备id/text/state.数据结构

[{id:1,text:”节点名称”,state:”closed/open”},{…},{….}]



### EasyUI中异步树加载

树控件读取URL。子节点的加载依赖于父节点的状态。当展开一个封闭的节点，如果节点没有加载子节点，它将会把节点**id的值**作为http请求参数并命名为**'id'**，通过URL发送到服务器上面检索子节点。

### 编辑Controller

/\*\*

\* 实现商品分类目录展现

\* **@RequestParam**(value="id",defaultValue="0",required=true)

\* value="实现数据的传递"

\* defaultValue:如果参数为null时生效

\* required=true ,改参数必须传值.

\* **@param** parentId

\* **@return**

\*/

@RequestMapping("/cat/list")

@ResponseBody

**public** List<EasyUITree> findItemCatListById(@RequestParam(value="id",defaultValue="0") Long parentId){

//查询一级商品分类标题

**return** itemCatService.findItemCatListById(parentId);

}

### 编辑Service

@Service

**public** **class** ItemCatServiceImpl **implements** ItemCatService {

@Autowired

**private** ItemCatMapper itemCatMapper;

@Override

**public** List<EasyUITree> findItemCatListById(Long parentId) {

//根据父级Id查询商品分类信息

ItemCat itemCat = **new** ItemCat();

itemCat.setParentId(parentId);

List<ItemCat> itemCatList =

itemCatMapper.select(itemCat);

List<EasyUITree> treeList = **new** ArrayList<>();

**for** (ItemCat itemCatTemp : itemCatList) {

EasyUITree easyUITree = **new** EasyUITree();

easyUITree.setId(itemCatTemp.getId());

easyUITree.setText(itemCatTemp.getName());

String state =

itemCatTemp.getIsParent() ? "closed" : "open";

easyUITree.setState(state);

treeList.add(easyUITree);

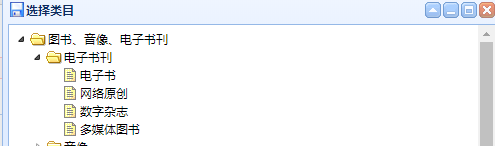
}

**return** treeList;

}

}

### 页面效果



## 商品的新增

### 页面描述

alert($("#itemAddForm").serialize());

$.post("/item/save",$("#itemAddForm").serialize(), **function**(data){

**if**(data.status == 200){

$.messager.alert('提示','新增商品成功!');

}**else**{

$.messager.alert("提示","新增商品失败!");

}

});

### 编辑Controller

//实现商品的新增

@RequestMapping("/save")

@ResponseBody

**public** SysResult saveItem(Item item){

**try** {

itemService.saveItem(item);

**return** SysResult.*oK*();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"商品新增失败");

}

### 编辑Service

@Override

**public** **void** saveItem(Item item) {

item.setStatus(1); //表示商品正常

item.setCreated(**new** Date());

item.setUpdated(item.getCreated());

itemMapper.insert(item);

//利用通用Mapper实现入库操作

}

## 商品修改

### 页面分析

$.post("/item/update",$("#itemeEditForm").serialize(), **function**(data){

**if**(data.status == 200){

$.messager.alert('提示','修改商品成功!','info',**function**(){

$("#itemEditWindow").window('close');

$("#itemList").datagrid("reload");

});

}**else**{

$.message.alert("提示",data.msg);

}

});

### 编辑Controller

//实现商品信息的修改

@RequestMapping("/update")

@ResponseBody

**public** SysResult updateItem(Item item){

**try** {

itemService.updateItem(item);

**return** SysResult.*oK*();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"商品更新失败");

}

### 编辑Service

@Override

**public** **void** updateItem(Item item) {

item.setUpdated(**new** Date());

//动态更新,更新其中不为空的数据

itemMapper.updateByPrimaryKeySelective(item);

}

## 修改商品的状态

### 页面分析

$.post("/item/reshelf",params, **function**(data){

**if**(data.status == 200){

$.messager.alert('提示','上架商品成功!',undefined,**function**(){

$("#itemList").datagrid("reload");

});

}

});

### 编辑Controller

/\*\*

\* 实现商品的上架

\* SpringMVC中参数接收

\* 1.接收简单类型 int... String

\* 2.使用对象对象 pojo

\* 3.使用集合数据接收参数

\* 页面: name="ids" value="1,2,3,4,5"

\* 可以采用String[]接收.

\*

\* List集合接收

\* 页面: name="lists[0]" value="tomcat"

\* (List lists)

\* 4.为对象的引用赋值 第一遍

\* 页面: name="id" value="100",

\* name="bigDog.type" value="比特犬"

\* name="bigDog.food.caixi" value="粑粑"

\* 对象: User{id,name,age,bigDog}/Dog{id,type,food}/Food(caixi,liang)

\* 接收: (User user)

\* **@param** ids

\* **@return**

\*/

@RequestMapping("/reshelf")

@ResponseBody

**public** SysResult reshelf(String[] ids){

**try** {

**int** status = 1; //表示正常

itemService.updateStatus(ids,status);

**return** SysResult.*oK*();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"上架失败");

}

### 编辑Service

@Override

**public** **void** updateStatus(String[] ids, **int** status) {

itemMapper.updateStatus(ids,status);

}

### 编辑mapper

1.编辑接口文件

**void** updateStatus(@Param("ids")String[] ids,@Param("status")**int** status);

2.编辑映射文件

<!--实现item数据状态更新操作

map集合 collection="map中的key"

array数组 collection="array"

List集合 collection="list"

-->

<update id=*"updateStatus"*>

update tb\_item set status = #{status},updated = now()

where id in (

<foreach collection=*"ids"* item=*"id"* separator=*","*>

#{id}

</foreach>

)

</update>