# 通用Mapper

## 通用Mapper介绍

### 持久层发展历程

1. JDBC.操作数据库最快
2. 数据库链接池.解决开关链接,导致数据库执行效率低(sql都需要手写)
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**.

概括:**以面向对象的方式,操作数据库.**

1. Hibernate

Hiberndate将JPA思想最终实现.以后操作数据库,程序员几乎不需要写sql.统统都是面向对象的形式操作.并且实现了全自动的ORM(对象关系映射)

对象-----🡪自动生成SQL,交给数据库. 入

数据库中的数据----------🡪对象 出

例子:hibernate框架实现单表映射时,效率很高.如果遇到多表关联时效率低.

Sql编辑:HQL语句解决多表关联问题.实用性弱.配置文件特别复杂.(一对一/一对多/多对多)

Sql特别的冗余.效率低

1. Mybatis

实现半自动的ORM.

程序员自己编辑Sql语句,实现入库操作.

查询时,返回的数据就是封装好的对象.

效率高:自己编辑的sql效率最高

弊端:再简单的sql也需要自己手动编辑!!!

## 通用Mapper插件

### 介绍

通用Mapper是基于Mybatis开发的一款简化单表操作的一款插件.可以实现单表自动CURD操作.以后程序员再也不需要写简单的sql.同时实现了JPA思想.

核心:用户操作是**对象**!!!

### 通用Mapper实现原理

1. 对象与数据库中的表一一映射.
2. 对象中的属性与表中的字段一一映射.
3. 定义一个公共的接口,在其中定义公共的CRUD操作.
4. 通过实现类实现接口中的方法.将用户传递的对象最终转化为SQL语句.

新增案例:

insert into **表名**(**字段名**……) values(**字段值……**)

删除案例:

delete from 表名 where id = 属性值 (值不为null)

### 添加jar包

<!-- 通用Mapper -->

<dependency>

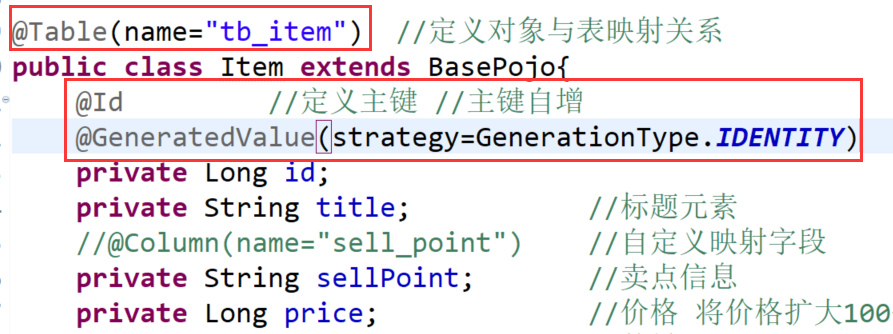
<groupId>com.github.abel533</groupId>

<artifactId>mapper</artifactId>

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

</dependency>

### 编辑POJO与表映射关系



### 添加插件

<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 -->

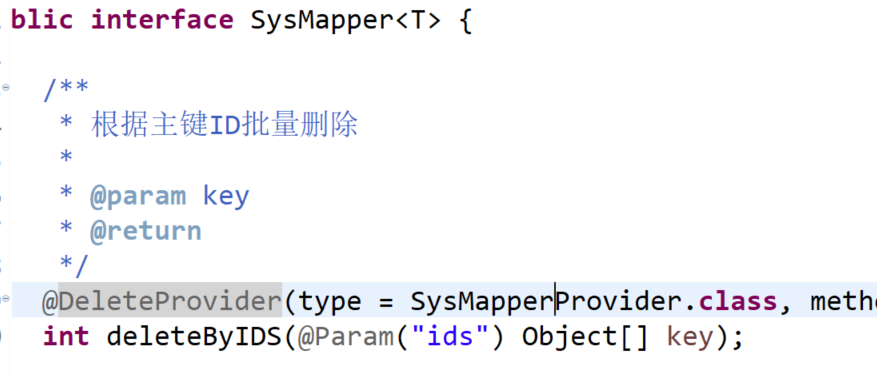
<property name=*"mappers"* value=*"com.jt.common.mapper.SysMapper"* />

</plugin>

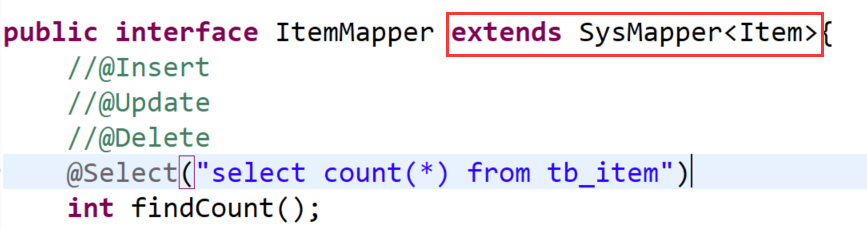
</plugins>

### 定义公共的接口

1. 定义接口



1. 扩展方法



实现:将jt-common打包处理

### 通用Mapper测试

@Override

**public** EasyUI\_Data findItemByPage(Integer page, Integer rows) {

//int total = itemMapper.findCount(); //商品记录总数

//如果没有where条件写null,如果有where条件,则为属性赋值即可.

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

/\*\*

\* SELECT \* FROM tb\_item LIMIT 起始位置,查询记录数

查询第一页

SELECT \* FROM tb\_item LIMIT 0,20 (0-19)

查询第二页

SELECT \* FROM tb\_item LIMIT 20,20 (20-39)

查询第N页

SELECT \* FROM tb\_item LIMIT (n-1)\*20,20\*/

**int** start = (page - 1) \* rows;

List<Item> itemList =

itemMapper.findItemByPage(start,rows);

//分页后 查询的结果

**return** **new** EasyUI\_Data(total,itemList);

}

# 后台商品的维护

## 树形结构展现

### 入门案例

1. 页面JS

$(function(){

$("#tree").tree({

url:"tree.json", //加载远程JSON数据

method:"POST", //请求方式 POST

animate:true, //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);

}

});

})

1. 数据结构分析

[{

"id":*"2"*,

"text":*"王者荣耀"*,

"children":[

{

"id":*"10"*,

"text":*"阿科"*

},{

"id":*"11"*,

"text":*"吕布"*

},{

"id":*"12"*,

"text":*"陈咬金"*

},{

"id":*"13"*,

"text":*"典韦"*

}

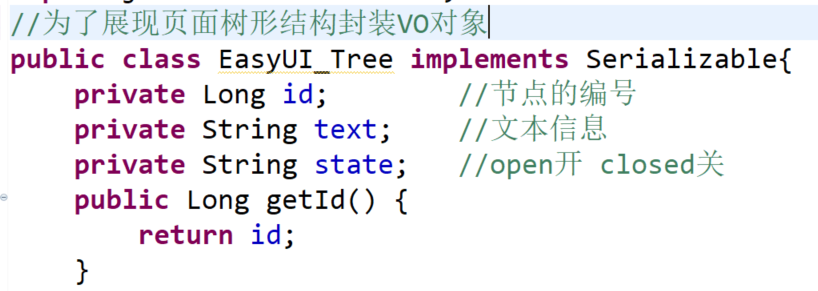
],

"state":*"closed"*

}]

经过分析:必须包含三个属性,id/text/state

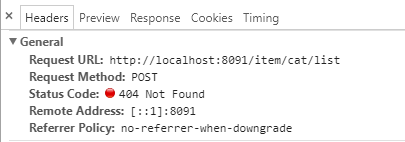
### 封装VO对象



## 实现树形结构展现

### 页面分析

1. 页面分析



1. 页面JS



### 编辑前台Controller

@Controller

@RequestMapping("/item")

**public** **class** ItemCatController {

@Autowired

**private** ItemCatService itemCatService;

//实现商品属性结构展现

@RequestMapping("/cat/list")

@ResponseBody

**public** List<EasyUI\_Tree> findTree(){

//展现一级列表信息

Long parentId = 0L;

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

}

}

### 编辑Service

@Service

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

@Autowired

**private** ItemCatMapper itemCatMapper;

**public** List<ItemCat> findItemCatByParentId(Long parentId){

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

itemCat.setParentId(parentId);

//基于通用Mapper查询数据信息

**return** itemCatMapper.select(itemCat);

}

@Override

**public** List<EasyUI\_Tree> findTree(Long parentId) {

List<ItemCat> catList =

findItemCatByParentId(parentId);

List<EasyUI\_Tree> treeList = **new** ArrayList<>();

**for** (ItemCat itemCat : catList) {

EasyUI\_Tree tree = **new** EasyUI\_Tree();

tree.setId(itemCat.getId());

tree.setText(itemCat.getName());

String state =

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

tree.setState(state);

treeList.add(tree);

}

**return** treeList;

}

}

### 动态展现树形结构

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。

### 重新编辑Controller

/\*\*

\* 实现商品树形结构展现

\* 当页面中没有传递id时,默认值为0查询一级商品分类信息

\* 如果页面中传递了参数,则动态获取参数!!!

\* **@param** parentId

\* **@return**

\*/

@RequestMapping("/cat/list")

@ResponseBody

**public** List<EasyUI\_Tree> findTree

(@RequestParam(value="id",defaultValue="0")Long parentId){

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

}

## 商品新增

### 数据校验

1.必填项

data-options="required:true"

2.设定区间范围

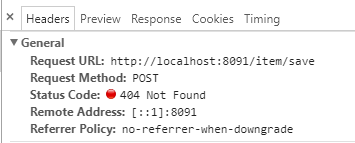
data-options="min:1,max:99999999,precision:2,required:true"

1. 校验长度

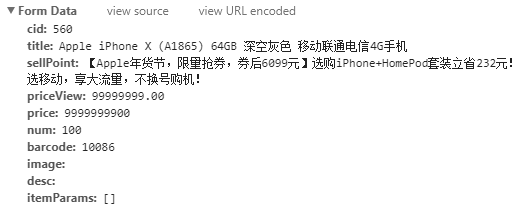
data-options="validType:'length[1,30]'"

### 页面分析

1.页面url



2.页面FORM表单提交



3.页面JS

$.post("/item/save",$("#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); //改商品正常

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

item.setUpdated(item.getCreated());

//利用通用Mapper实现入库操纵

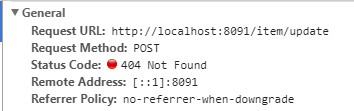
itemMapper.insert(item);

}

## 商品修改

### 页面分析

1.页面分析



2.修改JS

$.post("/item/update",$("#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

//商品修改

@RequestMapping("/update")

@ResponseBody

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

**try** {

itemService.updateItem(item);

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"商品修改失败");

}

### 商品更新

//商品更新

@Override

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

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

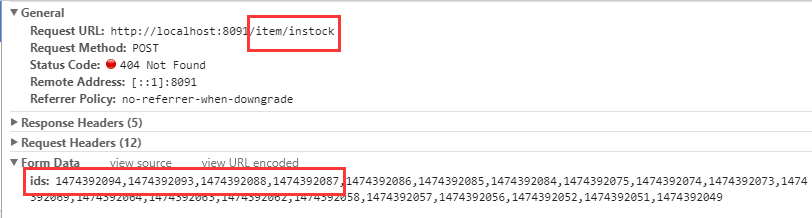
itemMapper.updateByPrimaryKeySelective(item);

}

## 商品下架/上架

### 页面分析

1.页面分析



2.页面JS

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

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

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

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

});

}

});

### 编辑Controller

//商品下架

@RequestMapping("/instock")

@ResponseBody

**public** SysResult instockItem(Long[] ids) {

**try** {

**int** status = 2; //下架

itemService.updateStatus(ids,status);

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

} **catch** (Exception e) {

e.printStackTrace();

}

**return** SysResult.*build*(201,"商品修改状态失败");

}

### 编辑页面Service

@Override

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

itemMapper.updateStatus(ids,status);

}

### 编辑Mapper接口/映射文件

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

编辑Mapper映射文件

<!--批量修改商品的状态信息

update tb\_item set status = #{status},

updated = now()

where id in (100,200,300)

-->

<update id=*"updateStatus"*>

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

where id in (

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

#{item}

</foreach>

)

</update>

## 作业 商品删除