**淘淘商城**

**第三天**

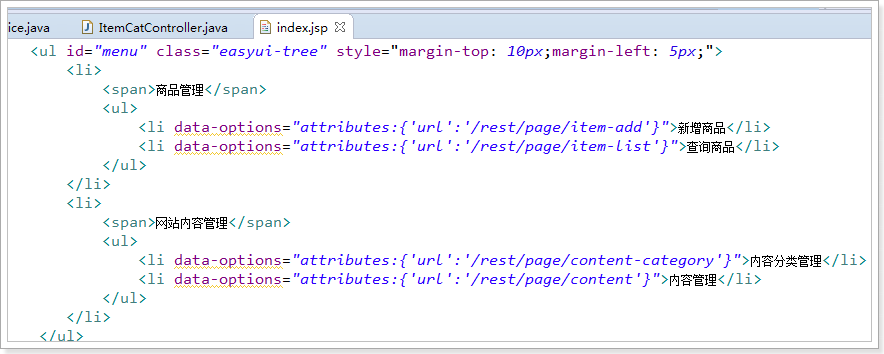
# 课程计划

1. 实现通用页面跳转方法
2. 使用域名访问后台系统
3. 使用Nginx反相代理服务器
4. 实现BaseService
5. 实现商品类目选择
6. 实现商品新增

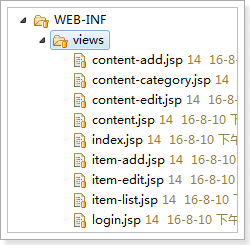
# 实现通用页面跳转方法

## 实现分析

查看taotao-manager-web系统的index.jsp



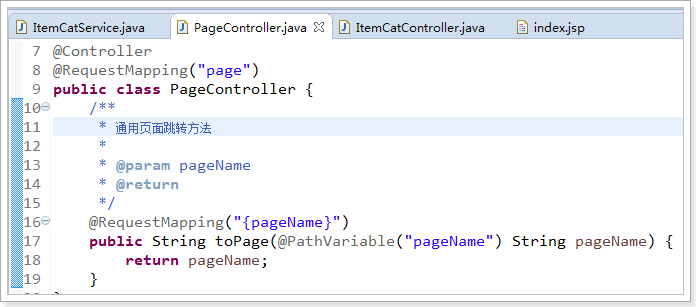
发现请求url都是差不多，/rest/page/xxx，其xxx的部分正好是views的视图名：



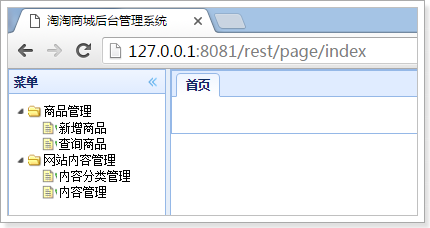
可以直接获取url的视图名称，然后返回，就可以实现通用跳转方法

## 编写Controller

在taotao-manager-web编写Controller



## 效果



# 使用域名访问后台系统

## 现在的方式存在的问题

现在访问的路径：http://127.0.0.1:8081/rest/page/index

存在的问题：

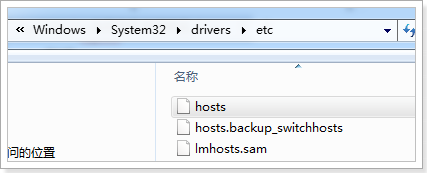
1. 开发环境和测试环境的ip不一样，每次环境变化时，都需要修改访问地址。

2. 页面加载资源文件，有可能使用url的全路径，一旦更换环境（ip变了），资源文件就无法加载了。

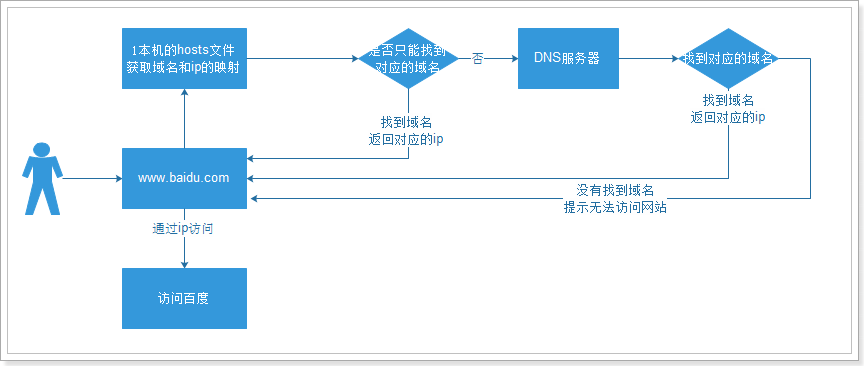
3. ip地址没有意义，不容易记忆，用户不会通过ip访问，一般通过域名访问

## 使用域名进行访问

可以通过修改hosts文件的方式，增加ip地址和域名的映射



使用域名访问的步骤

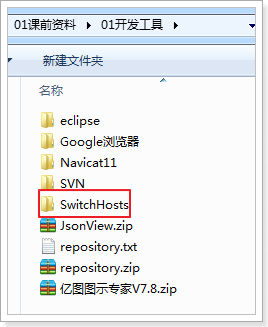


如果手动修改hosts文件，会非常麻烦，环境的切换非常复杂，企业中一般不这么干

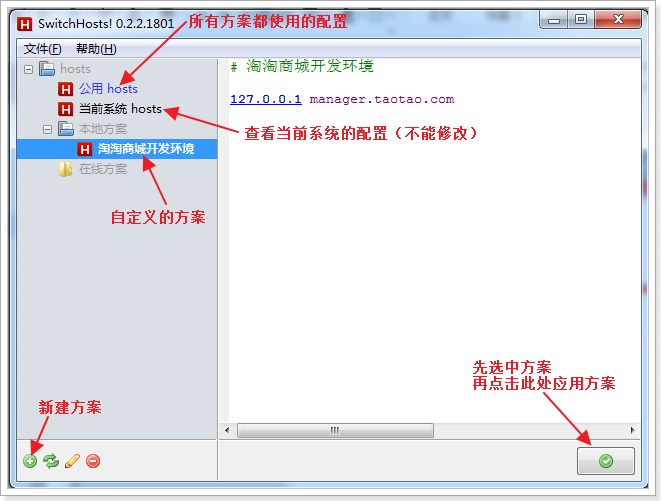
## 使用SwitchHosts管理

SwitchHosts帮助我们管理hosts文件，其实就是管理我们自定义ip和域名的映射

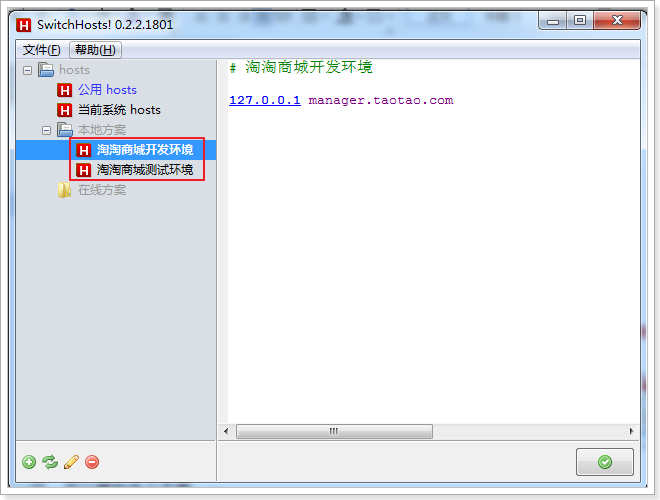
所在位置：



功能说明：



可以增加多个方案，淘淘商城开发环境和淘淘商城测试环境，可以方便的切换

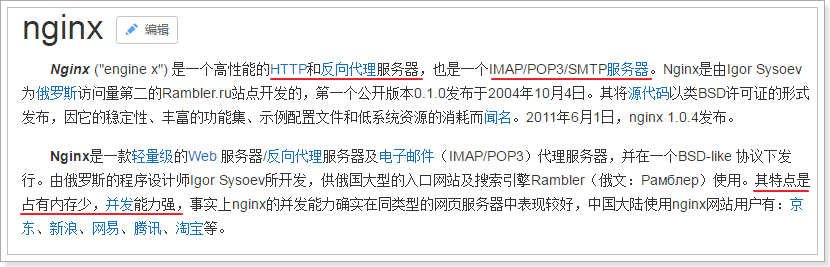


**存在的问题**

实现通过域名访问后，还存在端口号的问题

用户是直接输入域名，不会填写端口号。

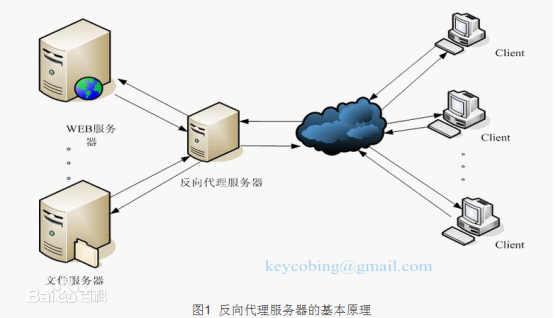
# Nginx介绍



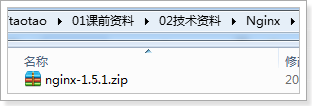
我们使用Nginx反相代理，虚拟主机，负载均衡

## 反相代理

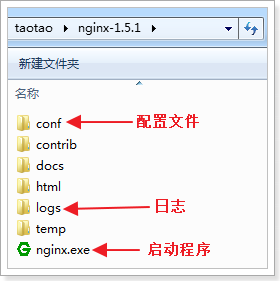
反向代理是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端。此时代理服务器对外就表现为一个反向代理服务器。



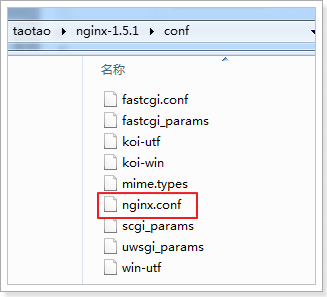
## Nginx目录结构



解压后的目录结构



## Nginx配置文件



修改配置文件：



## Nginx三个命令

启动cmd命令行输入命令

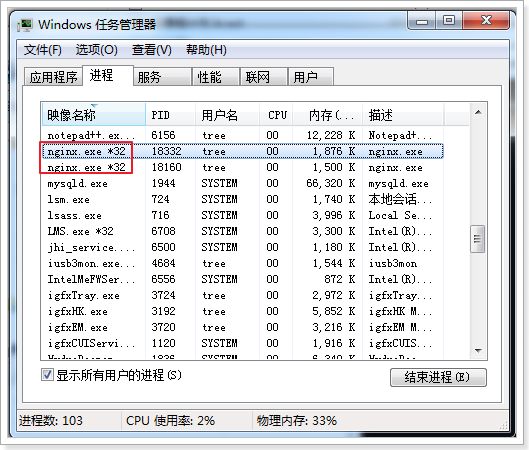
三个命令

启动: start nginx.exe

停止: nginx.exe –s stop

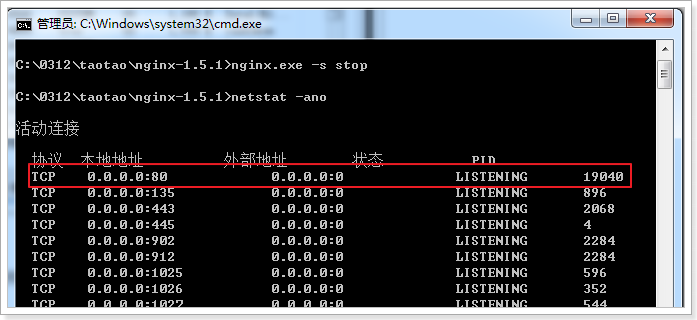
重载: nginx.exe –s reload

启动成功：有且仅有两个线程



启动报错,查看日志文件发现80端口被占用

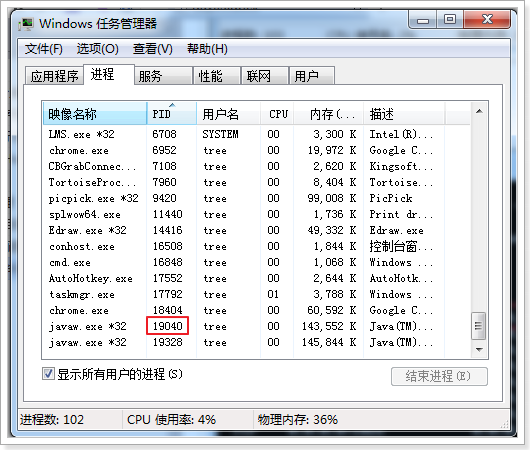
使用命令netstat -ano命令查看端口占用情况



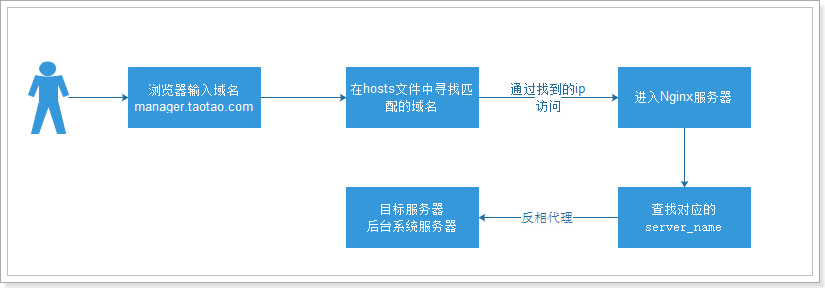
发现有程序占用80端口，根据pid查看是哪个进程

打开任务管理器，根据pid查询到是eclipse的Tomcat占用80端口

停止程序即可



## Nginx访问流程



# 实现BaseService

一般在项目开始时，就会编写BaseSerivce，否则后续编写BaseService时候，需要对所有service进行改造，增加开发成本

实现常用方法

1. queryById
2. queryAll
3. queryCountByWhere
4. queryListByWhere
5. queryByPage
6. queryOne
7. save
8. updateById
9. deleteById
10. deleteByIds

## 编写Service接口

在taotao-manager-interface子工程中编写接口

**public** **interface** BaseService<T> {

/\*\*

\* 根据id查询数据

\*

\* **@param** id

\* **@return**

\*/

**public** T queryById(Long id);

/\*\*

\* 查询所有数据

\*

\* **@return**

\*/

**public** List<T> queryAll();

/\*\*

\* 根据条件查询数据条数

\*

\* **@param** t

\* **@return**

\*/

**public** Integer queryCountByWhere(T t);

/\*\*

\* 根据条件查询数据

\*

\* **@param** t

\* **@return**

\*/

**public** List<T> queryListByWhere(T t);

/\*\*

\* 分页查询数据

\*

\* **@param** page

\* **@param** rows

\* **@return**

\*/

**public** List<T> queryByPage(Integer page, Integer rows);

/\*\*

\* 根据条件查询一条数据

\*

\* **@param** t

\* **@return**

\*/

**public** T queryOne(T t);

/\*\*

\* 新增

\*

\* **@param** t

\* **@return**

\*/

**public** **void** save(T t);

/\*\*

\* 新增，忽略空参数

\*

\* **@param** t

\* **@return**

\*/

**public** **void** saveSelective(T t);

/\*\*

\* 根据主键更新

\*

\* **@param** t

\* **@return**

\*/

**public** **void** updateById(T t);

/\*\*

\* 根据主键更新，忽略空参数

\*

\* **@param** t

\* **@return**

\*/

**public** **void** updateByIdSelective(T t);

/\*\*

\* 根据id删除数据

\*

\* **@param** id

\* **@return**

\*/

**public** **void** deleteById(Long id);

/\*\*

\* 根据ids批量删除数据

\*

\* **@param** ids

\* **@return**

\*/

**public** **void** deleteByIds(List<Object> ids);

}

## 编写Service实现类

在taotao-manager-service子工程中

**public** **class** BaseServiceImpl<T> **implements** BaseService<T> {

@Autowired

**private** Mapper<T> mapper;

**private** Class<T> clazz;

**public** BaseServiceImpl() {

// 获取父类的type

Type type = **this**.getClass().getGenericSuperclass();

// 强转为ParameterizedType，可以使用获取泛型类型的方法

ParameterizedType pType = (ParameterizedType) type;

// 获取泛型的class

**this**.clazz = (Class<T>) pType.getActualTypeArguments()[0];

}

@Override

**public** T queryById(Long id) {

T t = **this**.mapper.selectByPrimaryKey(id);

**return** t;

}

@Override

**public** List<T> queryAll() {

List<T> list = **this**.mapper.select(**null**);

**return** list;

}

@Override

**public** Integer queryCountByWhere(T t) {

**int** count = **this**.mapper.selectCount(t);

**return** count;

}

@Override

**public** List<T> queryListByWhere(T t) {

List<T> list = **this**.mapper.select(t);

**return** list;

}

@Override

**public** List<T> queryListByPage(Integer page, Integer rows) {

PageHelper.*startPage*(page, rows);

List<T> list = **this**.mapper.select(**null**);

**return** list;

}

@Override

**public** T queryOne(T t) {

T result = **this**.mapper.selectOne(t);

**return** result;

}

@Override

**public** **void** save(T t) {

**this**.mapper.insert(t);

}

@Override

**public** **void** saveSelective(T t) {

**this**.mapper.insertSelective(t);

}

@Override

**public** **void** updateById(T t) {

**this**.mapper.updateByPrimaryKey(t);

}

@Override

**public** **void** updateByIdSelective(T t) {

**this**.mapper.updateByPrimaryKeySelective(t);

}

@Override

**public** **void** deleteById(Long id) {

**this**.mapper.deleteByPrimaryKey(id);

}

@Override

**public** **void** deleteByIds(List<Object> ids) {

// 声明条件

Example example = **new** Example(**this**.clazz);

example.createCriteria().andIn("id", ids);

**this**.mapper.deleteByExample(example);

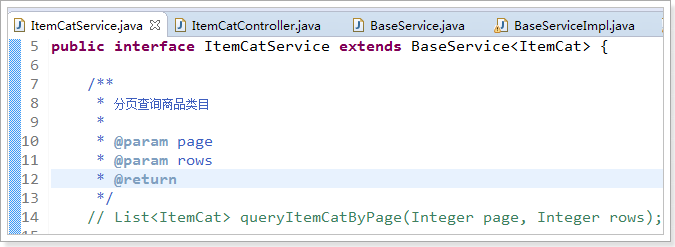
}

}

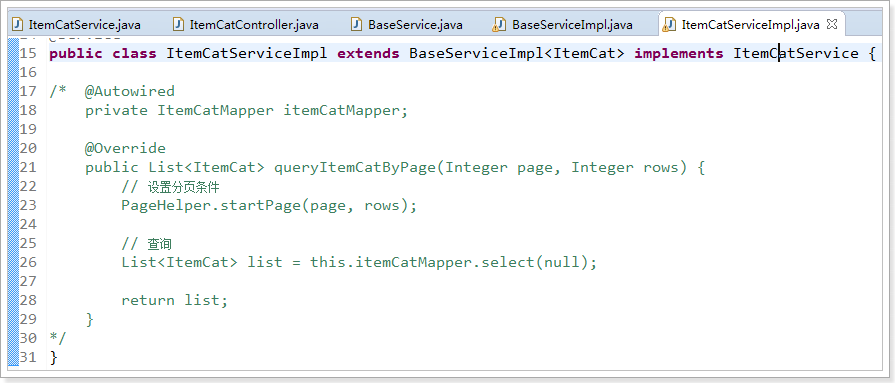
## 改造商品类目service和controller

使用BaseService，改造商品类目的service和controller

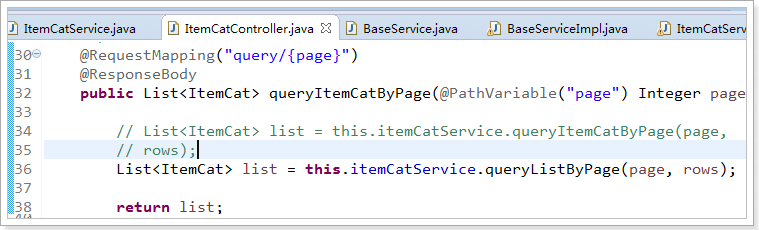
改造ItemCatService接口，继承BaseService，注释掉之前的方法



改造ItemCatServiceImpl实现类，把方法注释掉

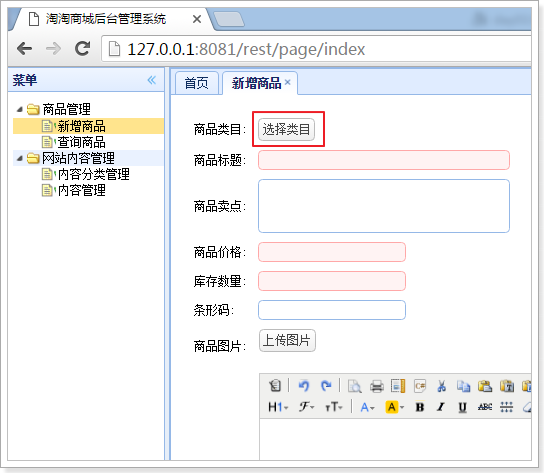


改造ItemCatController，使用BaseService的方法

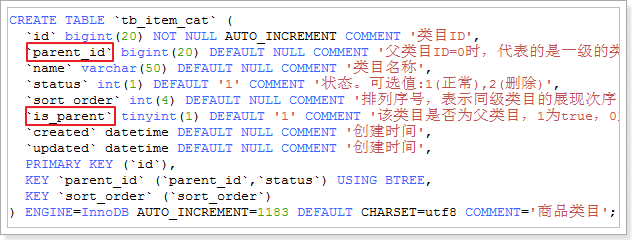


# 实现类目选择

点击新增商品，弹出类目选择窗口



## 数据库分析



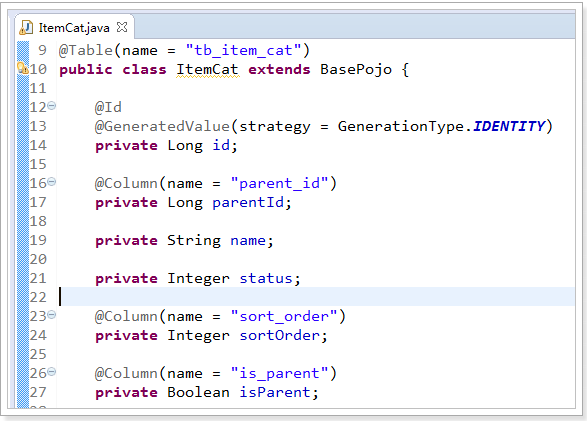
看到有parent\_id和is\_parent判断是一个树形结构

使用parent\_id作为条件进行查询，发现有三级类目

一共1182条数据

## 创建pojo

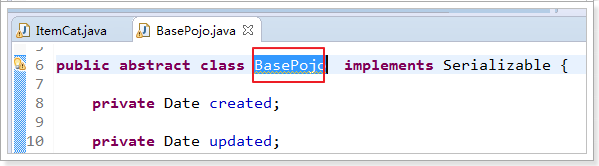
ItemCat里面没有新增时间和更新时间属性，这两个属性被抽取出来放在BasePojo中了



BasePojo

抽取共用的属性

不希望直接创建实例，加上abstract



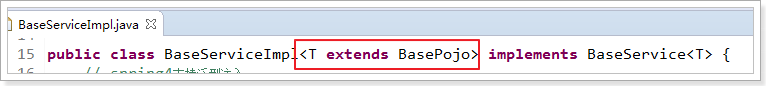
## 根据BasePojo改造BaseService

BasePojo抽取出共用的created属性和updated属性

可以在BaseServiceImpl的新增和更新方法中直接设置这两个属性，

其他人调用方法的时候就不用设置新增和更新属性了

### 限定泛型必须继承BasePojo



### 改造BaseService新增和更新方法

在BaseServiceImpl修改更新和新增方法

@Override

**public** **void** save(T t) {

// 需要判断，如果调用者没有设置时间，则这里设置，如果设置了时间，则这里不设置了

**if** (t.getCreated() == **null**) {

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

t.setUpdated(t.getCreated());

} **else** **if** (t.getUpdated() == **null**) {

t.setUpdated(t.getCreated());

}

**this**.mapper.insert(t);

}

@Override

**public** **void** saveSelective(T t) {

// 需要判断，如果调用者没有设置时间，则这里设置，如果设置了时间，则这里不设置了

**if** (t.getCreated() == **null**) {

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

t.setUpdated(t.getCreated());

} **else** **if** (t.getUpdated() == **null**) {

t.setUpdated(t.getCreated());

}

**this**.mapper.insertSelective(t);

}

@Override

**public** **void** updateById(T t) {

// 更新方法直接设置时间

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

**this**.mapper.updateByPrimaryKey(t);

}

@Override

**public** **void** updateByIdSelective(T t) {

// 更新方法直接设置时间

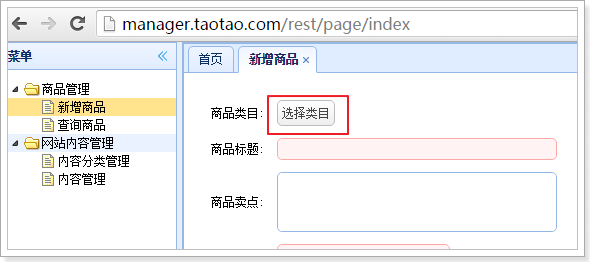
t.setUpdated(**new** Date());

**this**.mapper.updateByPrimaryKeySelective(t);

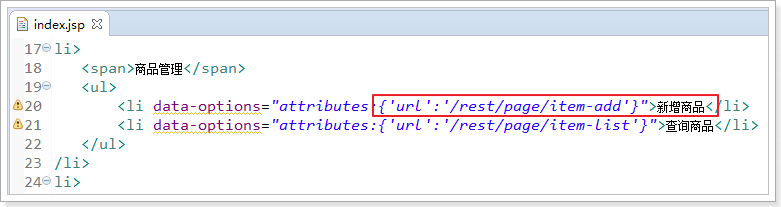
}

## 类目显示前端分析

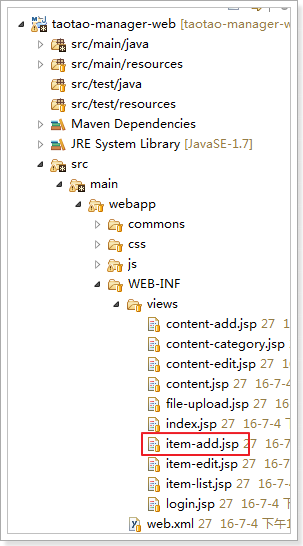
点击选择类目弹出窗口，分析按钮的点击事件



在index.jsp查看新增页面的按钮，确认新增页面的是item-add.jsp



查看新增页面

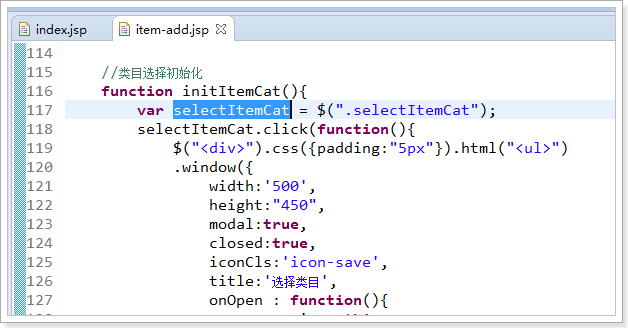


页面逻辑

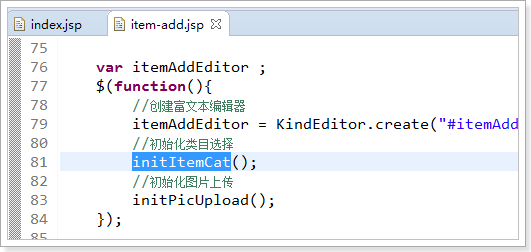


一般使用id或者class获取页面元素，这里只有class。

选中selectItemCat，按ctrl+k，查找当前页面同样的元素



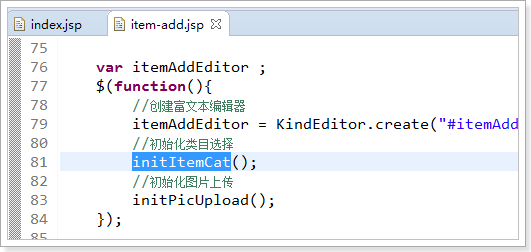
发现initItemCat是js方法名，搜索initItemCat



这里可以确认，当页面加载完成后，就会执行类目初始化方法

## js逻辑

当页面加载完成后，执行initItemCat()



调用的是initItemCat()方法

//类目选择初始化

**function** initItemCat(){

//获取class为selectItemCat的元素，其实就是类目选择按钮

**var** selectItemCat = $(".selectItemCat");

//给类目选择按钮增加点击事件

selectItemCat.click(**function**(){

//添加div标签，并设置css属性

//在div标签里面添加ul标签，并打开窗口

$("<div>").css({padding:"5px"}).html("<ul>")

.window({

//窗口创建属性

width:'500',

height:"450",

modal:**true**,

closed:**true**,

iconCls:'icon-save',

title:'选择类目',

//当窗口打开后执行的逻辑

onOpen : **function**(){

//这里的this是打开的窗口

**var** \_win = **this**;

//在窗口范围内，搜索ul标签

//找到ul标签，并创建EasyUI树

$("ul",\_win).tree({

//发起请求，创建树

url:'/rest/item/cat',

method:'GET',

animate:**true**,

//给树上的所有节点添加点击事件

onClick : **function**(node){

**if**($(**this**).tree("isLeaf",node.target)){

// 填写到cid中

selectItemCat.parent().find("[name=cid]").val(node.id);

selectItemCat.next().text(node.text);

$(\_win).window('close');

}

}

});

},

onClose : **function**(){

$(**this**).window("destroy");

}

}).window('open');

});

}

## 后台逻辑实现

前端创建树需要从后台获取数据。

url:'/rest/item/cat',

method:'GET',

根据前端页面的要求，实现后台逻辑

### 编写controller

在taotao-manager-web增加ItemCatController的方法

// url:'/rest/item/cat?id={nodeId}',

// method:'GET',

// 第一次请求是没有id参数，需要设置默认的parentId为0，查询一级类目

/\*\*

\* 根据parentId查询类目

\*

\* **@param** parentId

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***GET***)

@ResponseBody

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

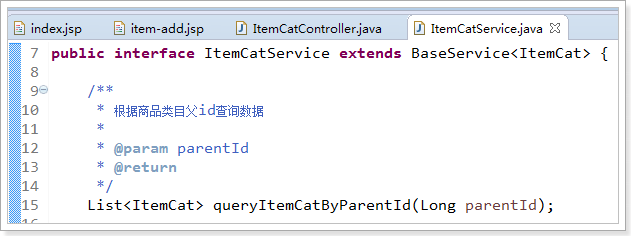
List<ItemCat> list = **this**.itemCatService.queryItemCatByParentId(parentId);

**return** list;

}

### 编写service

在taotao-manager-interface编写接口



在taotao-manager-service实现方法

@Override

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

// 设置查询条件

ItemCat param = **new** ItemCat();

param.setParentId(parentId);

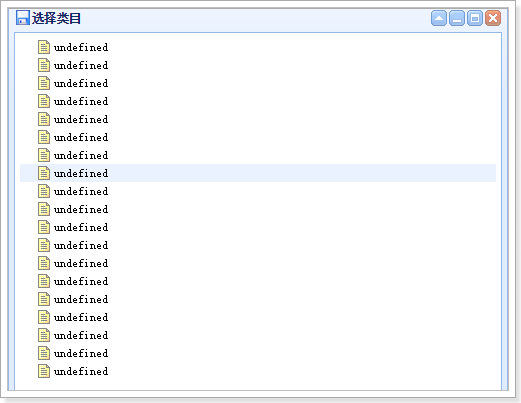
List<ItemCat> list = **super**.queryListByWhere(param);

**return** list;

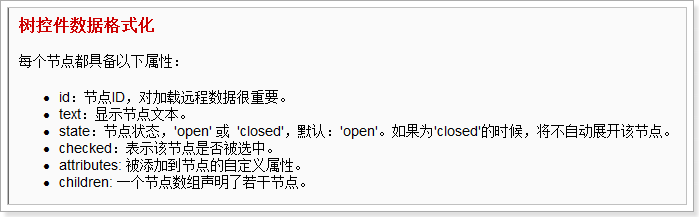
}

### 修改pojo

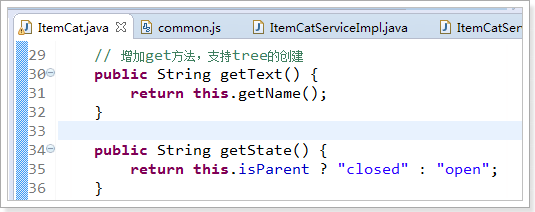
发现节点能查出来，但是都是undefined



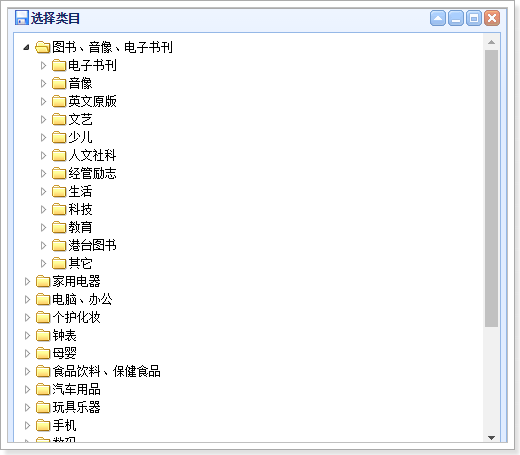
查看文档，返回的节点需要有text和state属性



改造ItemCat，增加get方法



## 效果



# 商品新增

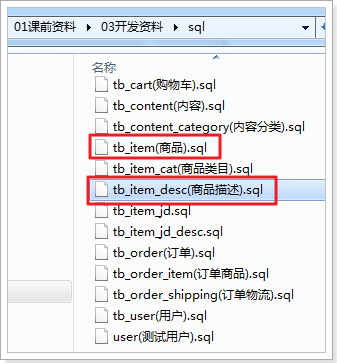
## 商品数据库表

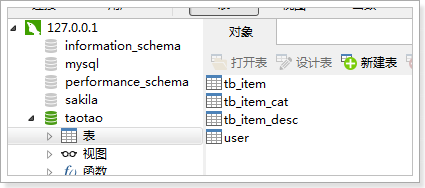
产品经理提供需求



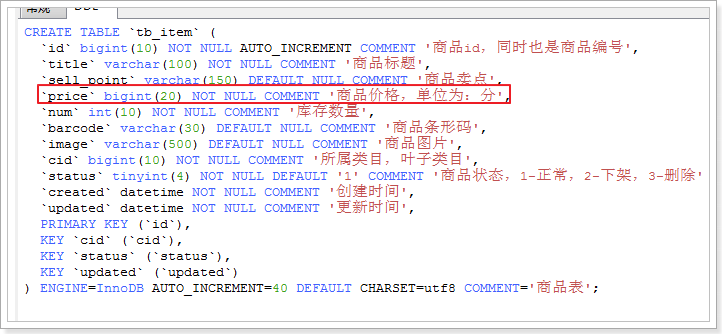
### 导入数据库表

导入两张表，商品表和商品描述表





### 商品表



数据库表设计的价格数据类型是bigint（优化点）

设置价格类型是整型，避免了小数问题

**public** **static** **void** main(String[] args) {

**float** a = 1.3f;

**double** b = 1.3;

**float** aa = a \* 3;

**double** bb = b \* 3;

System.***out***.println(aa);

System.***out***.println(bb);

}

打印结果是

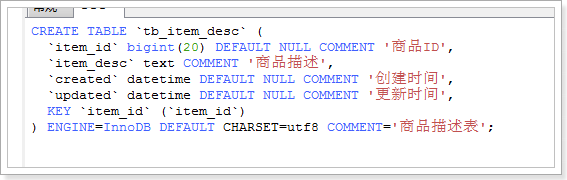
3.8999999

3.9000000000000004

存在的问题

前端显示是元，两位小数，保存在数据库是分，所以保存数据需要把显示的数据乘以100

### 商品描述表(优化点)



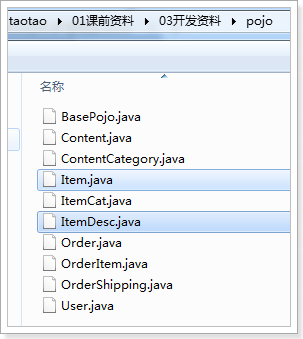
商品描述特点  
1数据量大

2修改频率低

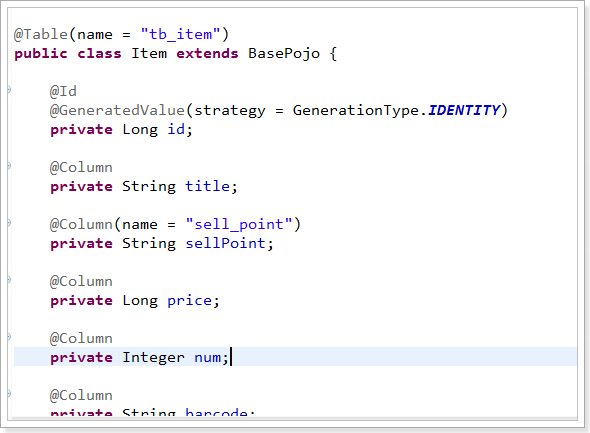
需要进行拆分

## 完成pojo

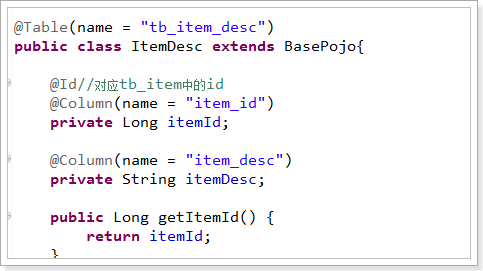
拷贝pojo到工程



### 商品pojo



### 商品描述pojo



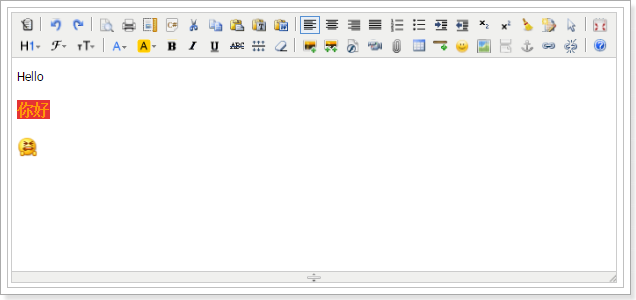
## 前端实现

### 商品价格



### 富文本编辑器kindeditor

#### 效果



原理：实际上就是对html代码的拼接

#### 前端实现

前端td只是一个多行文本





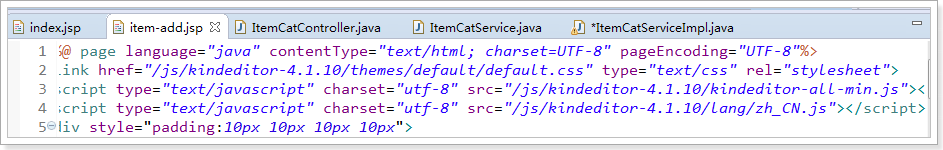
在页面加载成功后，创建富文本编辑器

参数一就是创建富文本编辑在哪个元素上面。

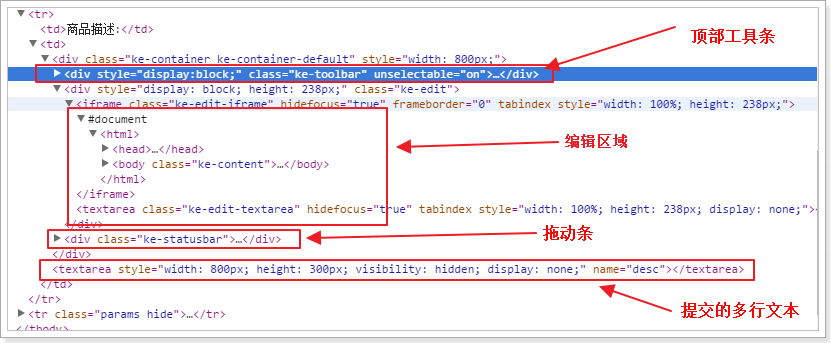
参数二就是创建富文本编辑器的参数

#### 需要加载资源文件js

上面使用了KindEditor创建富文本编辑器，要使用KindEditor这个对象，必须加载以下资源。类似使用$或者jQuery，必须加载jQuery的js。



#### 结构



## 表单提交前端逻辑

在item-add.jsp中的js提交逻辑：

//提交商品信息到后台

**function** submitForm(){

//校验表单

**if**(!$('#itemAddForm').form('validate')){

$.messager.alert('提示','表单还未填写完成!');

**return** ;

}

//获取id为itemAddForm的元素，就是提交表单，

//查找属性有name=price的元素，就是需要提交到后台的input标签

//eval方法是做字符串计算"1"+"2" ="3"。这个方法执行成功后的效果是，把用户数据的价格乘以一百

$("#itemAddForm [name=price]").val(eval($("#itemAddForm [name=priceView]").val()) \* 100);

//把富文本编辑器编辑区域的html代码。同步到多行文本中，向后台提交的是多行文本

//因为编辑器的编辑区域是div标签，不能提交

itemAddEditor.sync();

//提交到后台的RESTful

$.ajax({

type: "POST",

url: "/rest/item",

data: $("#itemAddForm").serialize(),

success: **function**(msg){

$.messager.alert('提示','新增商品成功!');

},

error: **function**(){

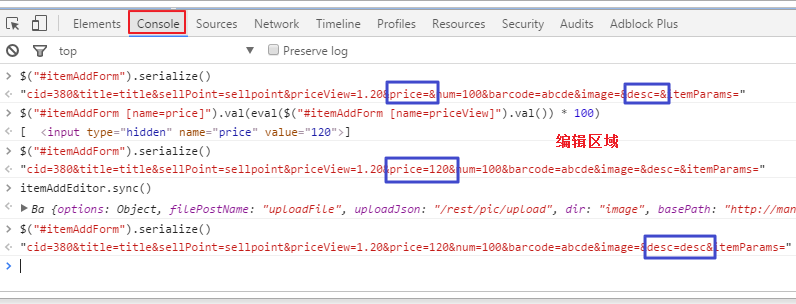
$.messager.alert('提示','新增商品失败!');

}

});

}

js代码演示效果



## 后端实现

分析前端逻辑

提交后台请求路径是

type: "POST",

url: "/rest/item",

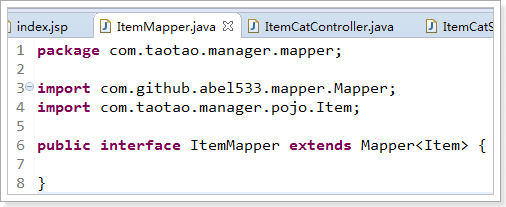
data: $("#itemAddForm").serialize(),

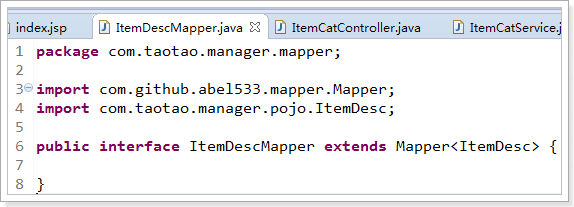
商品信息需要保存到两张表，一张商品表，一张商品描述表

不需要对返回值进行处理，返回值可以随意确定，我们返回OK

### 编写Mapper

在taotao-manager-mapper编写商品Mapper和商品描述Mapper

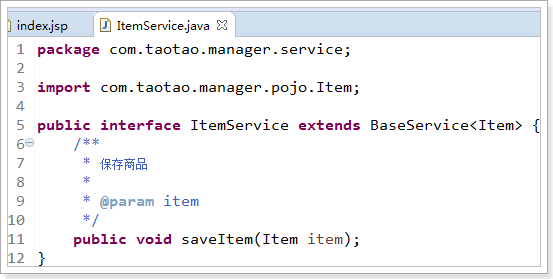


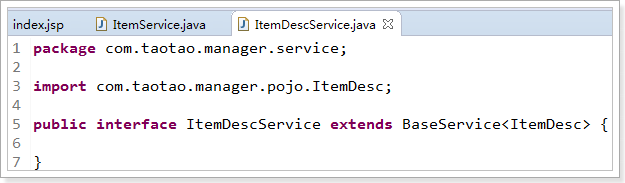


### 编写service

需要编写接口和实现类

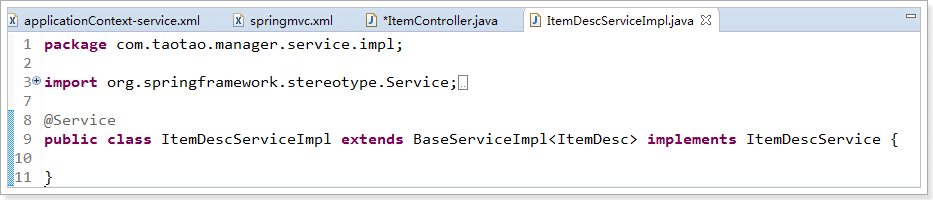
在taotao-manager-interface子工程编写接口





在taotao-manager-service实现方法

编写ItemDescServiceImpl实现类



编写ItemServiceImpl实现类

@Service

**public** **class** ItemServiceImpl **extends** BaseServiceImpl<Item> **implements** ItemService {

@Autowired

**private** ItemDescService itemDescService;

@Override

**public** **void** saveItem(Item item, String desc) {

// 保存商品

item.setStatus(1);

**super**.save(item);

// 保存商品描述

ItemDesc itemDesc = **new** ItemDesc();

itemDesc.setItemId(item.getId());

itemDesc.setItemDesc(desc);

**this**.itemDescService.save(itemDesc);

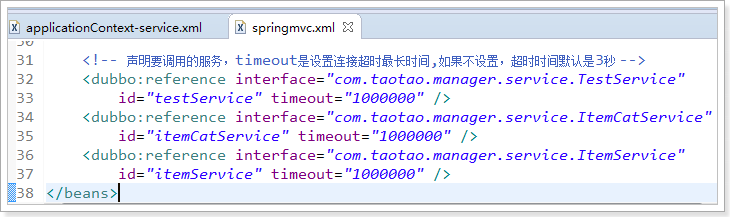
}

}

### 服务端发布服务



### 客户端调用服务



### 编写controller

在taotao-manager-web，编写controller

@Autowired

**private** ItemService itemService;

// type: "POST",

// url: "/rest/item",

/\*\*

\* 保存商品

\*

\* **@param** item

\* **@param** desc

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***POST***)

@ResponseBody

**public** void saveItem(Item item, String desc) {

**this**.itemService.saveItem(item, desc);

}

## 效果

