# 京淘项目第四天

## EasyUI介绍

### 入门案例

#### 引入js函数类库

<!--引入jquery的js，EasyUI的执行需要依赖于jQuery -->

<script type=*"text/javascript"*

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

<!--easyUI的js主文件 -->

<script type=*"text/javascript"*

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

<!--国际化的js文件 -->

<script type=*"text/javascript"*

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

<!--引入easyUI的样式 -->

<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"* />

#### 拖拽效果

<body>

<div class=*"easyui-draggable"*>拖动DIV</div>

<div class=*"easyui-draggable"*>测试div</div>

</body>

### 进度条加载

<script type="text/javascript">

/\*jQuery提供的 作用:让页面加载完成之后执行js\*/

$(**function**(){

//$("#Id")id选择 $(".class")类选择器 $("input")标签选择器

$("#b").click(**function**(){

onload();

})

/\*采用递归的方法实现进度条刷新 \*/

**var** i = 0;

**function** onload(){

$('#p').progressbar({ value:i++});

**if**(i<=100){

/\*延时加载,到了指定的时间,则执行函数,时间

单位是毫秒

setTimeout(执行的操作,延时时间 毫秒)

\*/

setTimeout(**function**(){

onload();

}, 1)

}

}

})

</script>

</head>

<body>

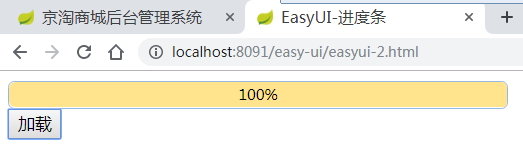
<div id="p" class="easyui-progressbar" style="width:400px;"></div>

<input id="b" type="button" value="加载"/>

</body>

</html>

效果:



## 京淘后台页面跳转

### 页面通用跳转

#### 业务描述

<li data-options="attributes:{'url':'**/page/item-add**'}">新增商品</li><li data-options="attributes:{'url':'**/page/item-list'**}">查询商品</li>

<li data-options="attributes:{'url':'**/page/item-param-list'**}">规格参数</li>

#### 问题说明

页面中的url地址对应后台服务器请求路径必须一一对应.如果页面中的url有多个则后台Controller必须添加多个RequestMapping方法.这样写比较繁琐.

#### RestFul实现

@Controller

**public** **class** IndexController {

/\*\*

\* 分析:如果能够获取url值,

\* 这个值就是页面跳转名称

\* 思路: 获取url中指定参数

\*

\* restFul:

\* 1.要求参数必须使用/分割

\* 2.参数位置必须固定

\* 3.接收参数时必须使用{}标识参数.

\* 使用特定的注解 **@PathVariable**

\* 并且名称最好一致

\*

\* **@return**

\*/

//根据用户请求,跳转页面

@RequestMapping("/page/{moduleName}")

**public** String itemAdd(@PathVariable String moduleName) {

**return** moduleName;

}

}

## 实现商品列表展现

### EasyUI表格展现

#### 表格定义

<div>

定义表格，并且通过url访问json数据， fitColumns:true表示自动适应，singleSelect:true 表示选中单个

<table class=*"easyui-datagrid"* style="width:*500px*;height:*300px*" data-options=*"url:'datagrid\_data.json',method:'get',fitColumns:true,singleSelect:true,pagination:true"*>

<thead>

<tr>

<th data-options=*"field:'code',width:100"*>Code</th>

<th data-options="field:'name',width:100">Name</th>

<th data-options="field:'price',width:100,align:'right'">Price</th>

</tr>

</thead>

</table>

</div>

#### 数据回显

{

"total":2000,

"rows":[

{"codeasdfasdf":*"A"*,"name":*"果汁"*,"price":*"20"*},

{"code":*"B"*,"name":*"汉堡"*,"price":*"30"*},

{"code":*"C"*,"name":*"鸡柳"*,"price":*"40"*},

{"code":*"D"*,"name":*"可乐"*,"price":*"50"*},

{"code":*"E"*,"name":*"薯条"*,"price":*"10"*},

{"code":*"F"*,"name":*"麦旋风"*,"price":*"20"*},

{"code":*"G"*,"name":*"套餐"*,"price":*"100"*}

]

}

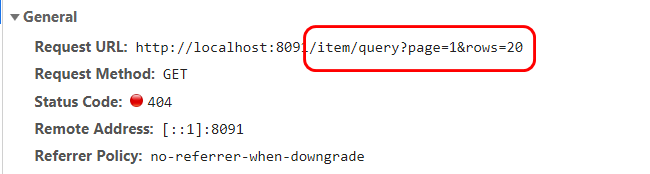
#### 数据显示的规则

1. 页面中定义表格之后定义url地址.定义数据的*field*
2. 数据必须按照指定的格式返回

{total:记录总数,rows:[{field:xxxx….},{},{}]}

### 商品列表展现

#### 页面URL分析



#### 编辑Controller

@RestController

@RequestMapping("/item")

**public** **class** ItemController {

@Autowired

**private** ItemService itemService;

//http://localhost:8091/item/query?page=1&rows=20

//查询商品列表信息 分页查询

@RequestMapping("/query")

**public** EasyUIData findItemByPage(Integer page,Integer rows) {

**return** itemService.findItemByPage(page,rows);

}

}

#### 编辑Service

@Service

**public** **class** ItemServiceImpl **implements** ItemService {

@Autowired

**private** ItemMapper itemMapper;

@Override

**public** EasyUIData findItemByPage(Integer page, Integer rows) {

//1.获取商品记录总数

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

/\*\*

\* 2.分页之后回传数据

\* sql: select \* from tb\_item limit 起始位置,查询行数

\* 第1页: 20

\* select \* from tb\_item limit 0,20

\* 第2页:

\* select \* from tb\_item limit 20,20

\* 第3页:

\* select \* from tb\_item limit 40,20

\* 第N页:

\* select \* from tb\_item

\* limit (page-1)rows,rows

\*/

//计算起始位置

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

List<Item> itemList = itemMapper.findItemByPage(start,rows);

**return** **new** EasyUIData(total,itemList);

}

}

#### 编辑Mapper

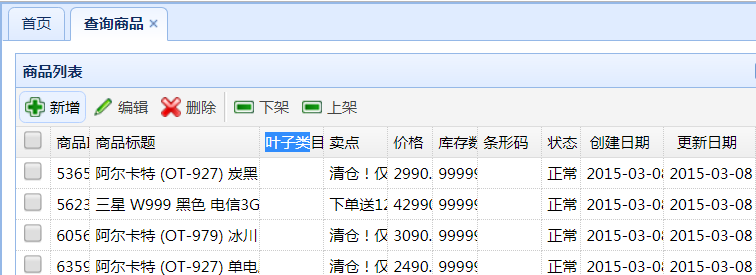
**public** **interface** ItemMapper **extends** BaseMapper<Item>{

@Select("select \* from tb\_item limit #{start},#{rows}")

List<Item> findItemByPage(@Param("start")Integer start,@Param("rows")Integer rows);

}

#### 页面效果

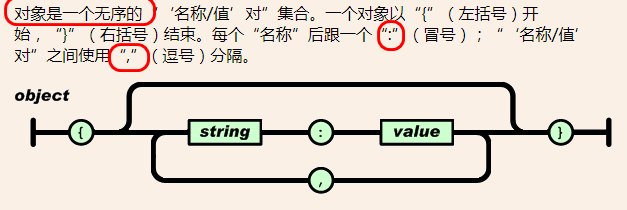


## JSON格式

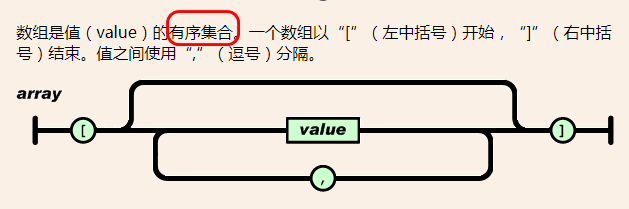
### JSON介绍

**JSON**(JavaScript Object Notation) 是一种轻量级的数据交换格式。 易于人阅读和编写。同时也易于机器解析和生成。

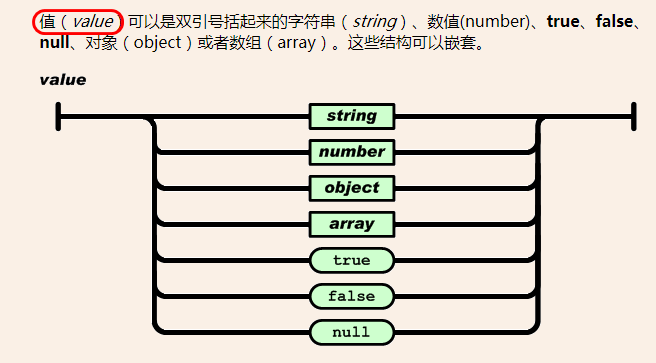
### Object格式



### Array格式



### 复杂格式



例子1:

{key:value,key2:value2,key3:true,key4:{key:value}

,key5:[1,2,3,4,5,{keyx:valuex…}]}

## PowerDesigner

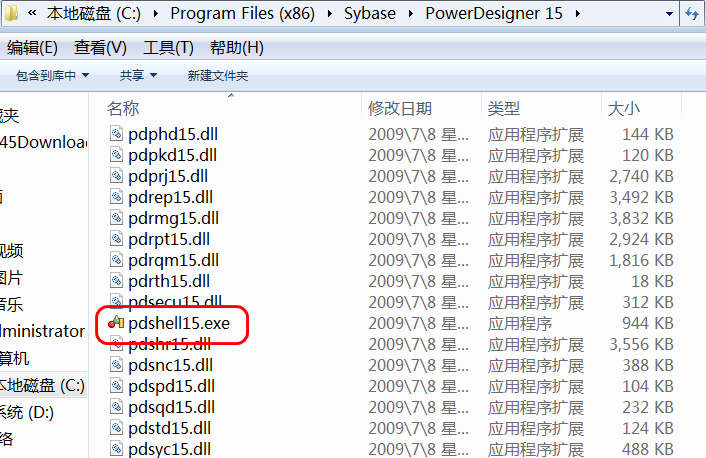
### 介绍

PowerDesigner最初由Xiao-Yun Wang（[王晓昀](https://baike.baidu.com/item/%E7%8E%8B%E6%99%93%E6%98%80/9236799)）在SDP Technologies公司开发完成。PowerDesigner是[Sybase](https://baike.baidu.com/item/Sybase" \t "_blank)的企业建模和设计解决方案，采用模型驱动方法，将业务与IT结合起来，可帮助部署有效的企业体系架构，并为研发生命周期管理提供强大的分析与设计技术。PowerDesigner独具匠心地将多种标准数据建模技术（UML、业务流程建模以及市场领先的[数据建模](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BB%BA%E6%A8%A1/720111" \t "_blank)）集成一体，并与 .NET、WorkSpace、PowerBuilder、Java™、Eclipse 等主流开发平台集成起来，从而为传统的软件开发周期管理提供业务分析和规范的数据库设计解决方案。此外，它支持60多种关系数据库管理系统（RDBMS）/版本。PowerDesigner运行在Microsoft Windows平台上，并提供了Eclipse插件.

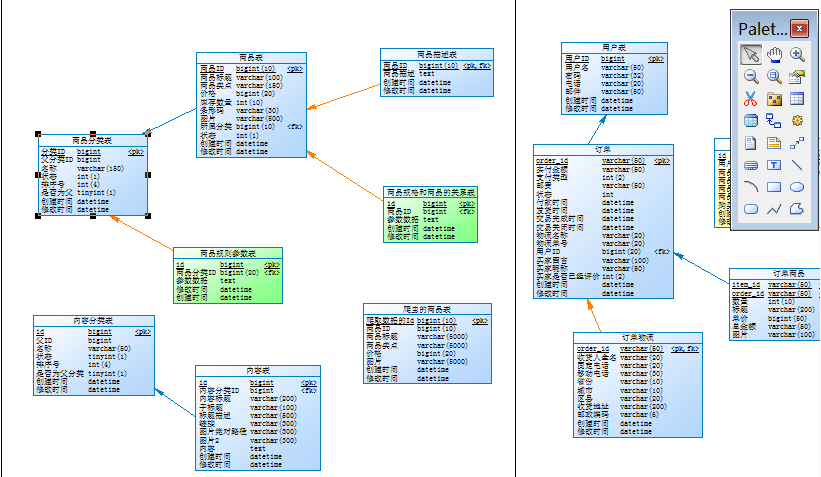
数据中的表需要通过可视化的形式直观的展现表与表之间的关联关系

### 汉化破解

说明:将汉化破解文件夹中的文件复制.替换PD安装根目录中的文件,完成破解.之后点击exe运行程序.



### 数据库模型



## 商品分类信息回显

### 页面数据回显格式化操作

#### 格式化价格

<th data-options="field:'price',width:70,align:'right',formatter:KindEditorUtil.formatPrice">价格</th>

格式化JS:

// 格式化价格 val:当前td值,row 当前行

formatPrice : **function**(val,row){

**return** (val/100).toFixed(2);

},

#### 格式化时间

<th data-options="field:'created',width:130,align:'center',formatter:KindEditorUtil.formatDateTime">创建日期</th>

格式化JS:

// 格式化时间

formatDateTime : **function**(val,row){

**var** now = **new** Date(val);

**return** now.format("yyyy-MM-dd hh:mm:ss");

},

#### 格式化商品目录JS

<th data-options="field:'cid',width:100,align:'center',formatter:KindEditorUtil.findItemCatName">叶子类目</th>

页面JS:

//格式化名称

/\*\*

\* 1.$.get(url,params,callback,dataType);

\* 2.$.post();

\* 3.$.ajax();

\* 4.$.load();

\* 5.$.getJSON();

\*/

findItemCatName : **function**(val,row){

**var** name;

$.ajax({

type:"post",

url:"/item/cat/queryItemName",

data:{itemCatId:val}, //代表参数

cache:**true**, //缓存

async:**false**, //表示同步 默认的是异步的true

dataType:"text",//表示返回值参数类型

success:**function**(data){

name = data;

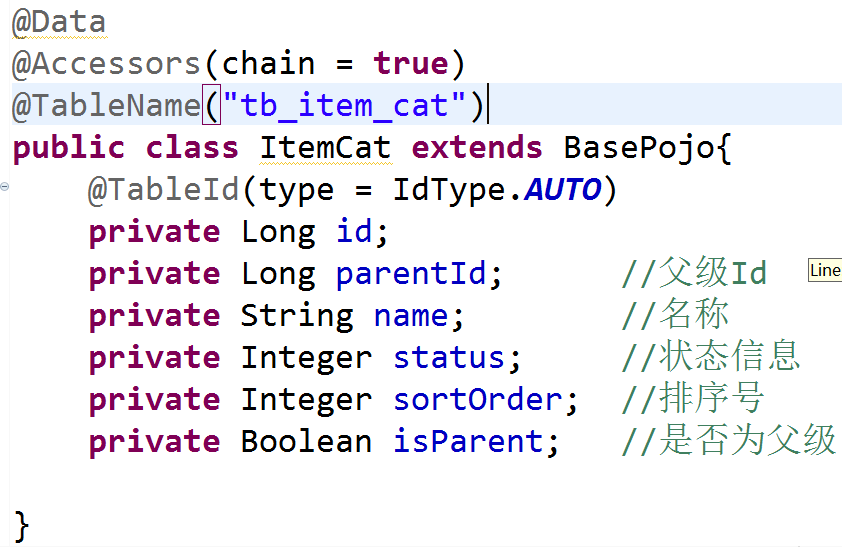
}

});

**return** name;

},

### 编辑POJO对象



### 编辑Controller

@RestController

@RequestMapping("/item/cat")

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

@Autowired

**private** ItemCatService itemCatService;

/\*\*

\* 1.用户发起post请求携带了itemCatId=560

\* 2.servlet request response

\*

\* **@return**

\*/

//实现根据id查询商品分类信息

@RequestMapping("/queryItemName")

**public** String findItemCatNameById(Long itemCatId) {

**return** itemCatService.findItemCatNameById(itemCatId);

}

}

### 编辑Service

@Service

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

@Autowired

**private** ItemCatMapper itemCatMapper;

@Override

**public** String findItemCatNameById(Long itemCatId) {

ItemCat itemCat = itemCatMapper.selectById(itemCatId);

**return** itemCat.getName();

}

}

### 页面效果



### EasyUI调用小结

1. 通用页面跳转机制

RestFul结构/风格

//根据用户请求,跳转页面

@RequestMapping("/page/{moduleName}")

**public** String itemAdd(@PathVariable String moduleName) {

**return** moduleName;

}

1. EasyUI表格数据展现

Item-list页面发起Ajax请求

data-options="singleSelect:false,fitColumns:true,collapsible:true,

pagination:true,**url:'/item/query'**,method:'get',pageSize:20,

toolbar:toolbar">

1. 实现商品分类数据展现

//格式化名称

/\*\*

\* 1.$.get(url,params,callback,dataType);

\* 2.$.post();

\* 3.$.ajax();

\* 4.$.load();

\* 5.$.getJSON();

\*/

findItemCatName : **function**(val,row){

**var** name;

$.ajax({

type:"post",

url:"/item/cat/queryItemName",

data:{itemCatId:val}, //代表参数

cache:**true**, //缓存

async:**false**, //表示同步 默认的是异步的true

dataType:"text",//表示返回值参数类型

success:**function**(data){

name = data;

}

});

**return** name;

},

## 商品分类树形结构

### 弹出框展现

$("#btn1").bind("click",**function**(){

$("#win1").window({

title:"弹出框",

width:400,

height:400,

modal:**true** //这是一个模式窗口，只能点击弹出框，不允许点击别处

})

})

### 商品分类菜单描述

说明:一般电商网站商品分类一般3级.



数据库设计:一张表完成商品分类目录设计

业务逻辑: 一级菜单

二级菜单

三级菜单

/\*查询全部一级商品分类信息\*/

SELECT \* FROM tb\_item\_cat WHERE parent\_id=0

/\*查询全部二级商品分类信息\*/

SELECT \* FROM tb\_item\_cat WHERE parent\_id=495

/\*查询全部三级商品分类信息\*/

SELECT \* FROM tb\_item\_cat WHERE parent\_id=513

### EasyUI树形结构

#### EasyUI树形结构入门

$(function(){

$("#tree").tree({

url:"tree.json", //加载远程JSON数据

method:"get", //请求方式 POST

animate:false, //表示显示折叠端口动画效果

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);

}

});

})

#### 树形结构JSON说明

[

{

"id":*"1"*,

"text":*"英雄联盟"*,

"state":*"open"*

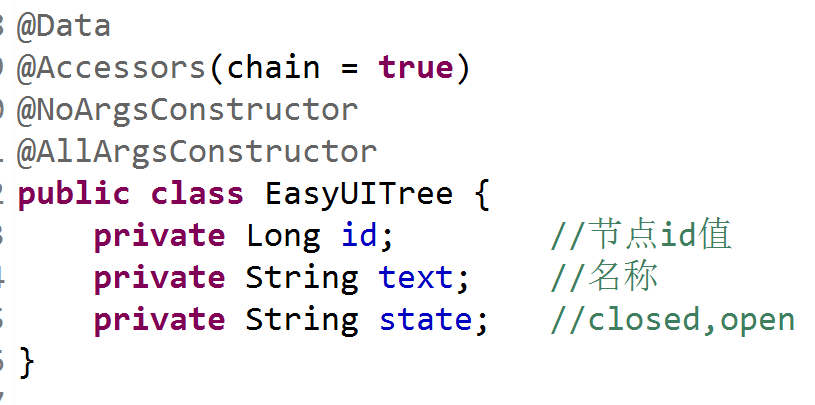
}

]

JSON:[{id:编号,text:节点名称,state:”open/closed”}]

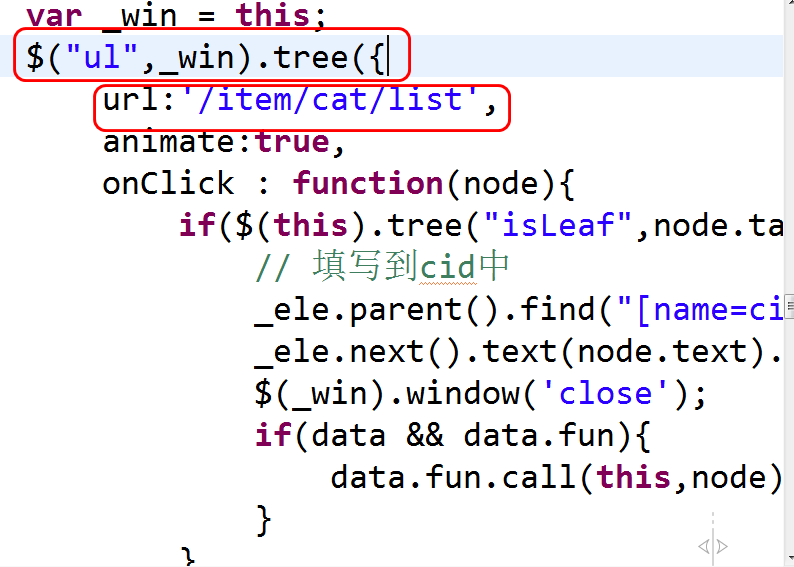
VO:EasyUITree{id,text,state}

### 定义VO对象



### 页面JS

说明:根据树形结构加载,查看页面JS.查找url请求地址,返回值是指定的VO对象的JSON数据.



### 编辑Controller

说明:根据页面url请求地址. 查询数据库数据.返回vo对象

//查询全部数据的商品分类信息

@RequestMapping("/list")

**public** List<EasyUITree> findItemCatByParentId(){

Long parentId = 0L; //查询一级商品分类信息

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

}

### 编辑Service

根据商品分类parentId查询数据.

/\*\*

\* 1.根据parentId查询数据库记录返回itemCatList集合

\* 2.将itemCatList集合中的数据按照指定的格式封装为

\* List<EasyUITree>

\*/

@Override

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

List<ItemCat> cartList = findItemCatList(parentId);

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

//遍历集合数据,实现数据的转化

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

EasyUITree uiTree = **new** EasyUITree();

uiTree.setId(itemCat.getId());

uiTree.setText(itemCat.getName());

String state = itemCat.getIsParent()?"closed":"open";

//如果是父级则closed 不是则open

uiTree.setState(state);

treeList.add(uiTree);

}

**return** treeList;

}

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

QueryWrapper<ItemCat> queryWrapper = **new** QueryWrapper<ItemCat>();

queryWrapper.eq("parent\_id", parentId);

**return** itemCatMapper.selectList(queryWrapper);

}

### 页面效果

