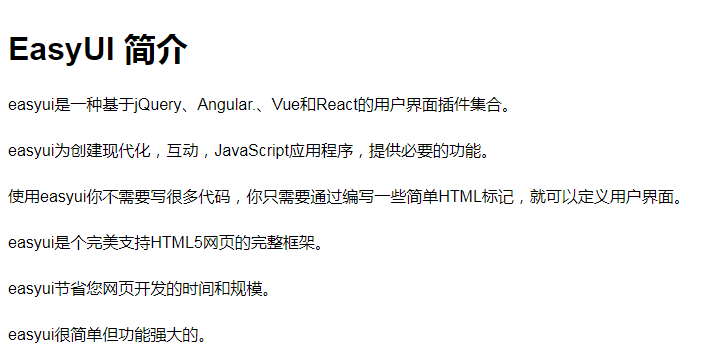
# 京淘项目商品管理

## EasyUI介绍

### EasyUI官网介绍

用户可以不用了解HTML代码.可以直接按照EasyUI指定的规则,灵活的创建页面.



## EasyUI入门案例

### 拖拽

1.引入js easyUI必须依赖于jQuery

<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" />

2.案例代码

<body>

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

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

</body>

## 通用页面跳转

### 业务需求

页面中每发起一个请求.则在Controller中必须编辑一个@RequestMapping方法进行拦截.

如果页面中有同样的请求100个.则必须写100个方法.特别繁琐.

如何优化???? 使用restFul风格

### 编辑通用跳转Controller

@Controller

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

/\*\*

\* 业务需求: 通过url实现页面跳转功能

\* /page/item-add

\* /page/item-list

\* /page/item-param-list

\*

\* 共性:

\* 1.请求路径的后半部分不同

\* 2.请求路径与跳转页面相同

\*

\*思考: 能否编辑一个方法实现页面的通用跳转??

\*

\*RestFul风格 必会内容

\* 1.url中的参数必须使用"/"分割

\* 2.服务端获取数据时.必须使用"{}"包裹参数

\* 3.使用@PathVariable注解实现数据转化

\*

\* 如果接收参数名称与url变量名称不一致.则需要使用value属性标识

\* **@PathVariable**(value = "moduleName") String abc

\*/

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

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

**return** moduleName;

}

}

### restFul实现应用

需求:没有页面条件下.要求测试用户入库操作代码是否正确???

说明:可以利用restFul风格.将用户参数拼接到url中使用/分割

案例应用:

/\*\*

//localhost:8091/saveItem/手机/永不关机/1000000

\* 使用时需要保证参数与属性名称一致

\* **@param** item

\* **@return**

\*/

@RequestMapping("/saveItem/{title}/{sellPoint}/{price}")

@ResponseBody

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

**return** item;

}

## 商品列表展现

### EasyUI表格展现

#### 表格编辑

<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>

#### JSON串封装

{

"total":200,

"rows":[

{"code":*"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"*}

]

}

#### 页面样式



### 封装VO对象

说明在jt-manage中添加vo对象

@Data

@Accessors(chain = **true**)

@NoArgsConstructor

@AllArgsConstructor

**public** **class** EasyUI\_Table {

**private** Integer total; //总记录数

**private** List<Item> rows;//分页后记录数

}

### 页面分析

1.url请求路径:/item/query **field**属性与对象的属性对应.可以自动的获取属性值.无需自己手动封装.

<table class="easyui-datagrid" id="itemList" title="商品列表"

data-options="singleSelect:false,fitColumns:true,collapsible:true,pagination:true,**url:'/item/query'**,method:'get',pageSize:20,toolbar:toolbar">

<thead>

<tr>

<th data-options="field:'ck',checkbox:true"></th>

<th **data-options="field:'id'**,width:60">商品ID</th>

<th **data-options="field:'title'**,width:200">商品标题</th>

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

<th data-options="field:'sellPoint',width:100">卖点</th>

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

<th data-options="field:'num',width:70,align:'right'">库存数量</th>

<th data-options="field:'barcode',width:100">条形码</th>

<th data-options="field:'status',width:60,align:'center',formatter:KindEditorUtil.formatItemStatus">状态</th>

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

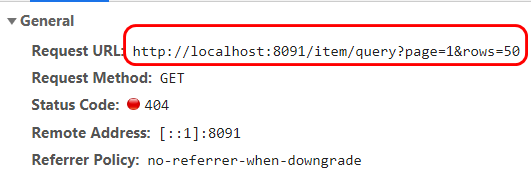
<th data-options="field:'updated',width:130,align:'center',formatter:KindEditorUtil.formatDateTime">更新日期</th>

</tr>

</thead>

</table>

2.用户请求的全路径



### 编辑Controller

@RestController

@RequestMapping("/item")

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

@Autowired

**private** ItemService itemService;

/\*\*

\* 1.根据分页信息实现查询

\*

\* 复习:

\* SpringMVC取赋值过程

\* http://localhost:8091/item/query?page=1&rows=50

\*

\* String page = request.getParameter("page");

\* String rows = request.getParameter("rows");

\* int intPage = Integer.parseInt(page);

\* int intRows = Integer.parseInt(rows);

注意:SpringMVC中方法的参数名称必须与提交的参数一致!!!!

\*/

@RequestMapping("/query")

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

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

}

}

### 编辑Service

@Service

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

@Autowired

**private** ItemMapper itemMapper;

/\*查询第一页 起始位置,显示条数\*/

//SELECT \* FROM tb\_item LIMIT 0,20

/\*查询第二页\*/

//SELECT \* FROM tb\_item LIMIT 20,20

/\*查询第三页\*/

//SELECT \* FROM tb\_item LIMIT 40,20

/\*查询第N页\*/

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

@Override

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

//1.获取总记录数

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

//2.分页查询List集合

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

List<Item> itemList =

itemMapper.findItemByPage(start,rows);

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

}

}

### 编辑Mapper

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

/\*\*

\* 查询数据库时,要求最近的记录显示在最前方. 倒叙排列

\*

\* **@param** start

\* **@param** rows

\* **@return**

\*/

@Select("SELECT \* FROM tb\_item ORDER BY updated DESC LIMIT #{start},#{rows}")

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

}

### 页面效果展现



## 实现叶子目录回显

### 格式化价格

// 格式化价格 val:数据库数据 99999 row

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

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

},

### 格式化日期

// 格式化时间

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

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

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

},

### 格式化叶子目录

//格式化名称

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

**var** name;

$.ajax({

type:"post",

url:"/item/cat/queryItemName",

data:{itemCatId:val},

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Controller

@RestController

@RequestMapping("/item/cat")

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

@Autowired

**private** ItemCatService itemCatService;

/\*\*

\* 业务说明:

\* 根据页面商品分类id号查询商品分类名称

\*利用ajax进行异步提交

\* url:"/item/cat/queryItemName"

\* 参数: itemCatId:val

\* 返回值类型: 返回具体分类名称

\*/

@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 =

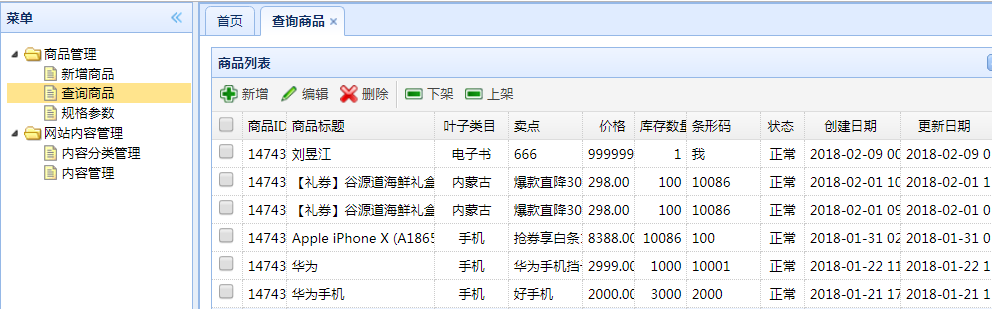
itemCatMapper.selectById(itemCatId);

**return** itemCat.getName();

}

}

### 页面效果



### 关于Ajax嵌套问题说明

说明:

1.EasyUI中展现商品信息时.其实在内部已经发起一个ajax

url:'/item/query

2.当发起ajax请求时,内部又嵌套一个ajax.

$.ajax({

type:"post",

url:"/item/cat/queryItemName",

data:{itemCatId:val},

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;

}

});

规则:如果遇到ajax嵌套调用.则切记将内部ajax改为**同步调用**.否则没有数据.

## 商品分类实现

### 商品分类分析

1. 根据规则发现商品分类信息一般三级.并且有子父集关闭.

所以在表设计时,使用parentId的形式进行关联.

/\*查询一级商品分类信息 parent\_id=0\*/

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

/\*查询二级商品分类信息 parent\_id=0\*/

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

/\*查询三级商品分类信息 parent\_id=0\*/

SELECT \* FROM tb\_item\_cat WHERE parent\_id = 539;

说明:子父集关系实际中应用广泛. 组织机构/省市县乡/角色信息等.

### 树形结构(了解 下节课说)

常用树结构: zTree树形,TreeGread,各大js框架自带树

VO: EasyUITree

class{ id/text/state*open/closed*}

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

url:'/item/cat/list',

animate:**true**,

onClick : **function**(node){

# 补充知识

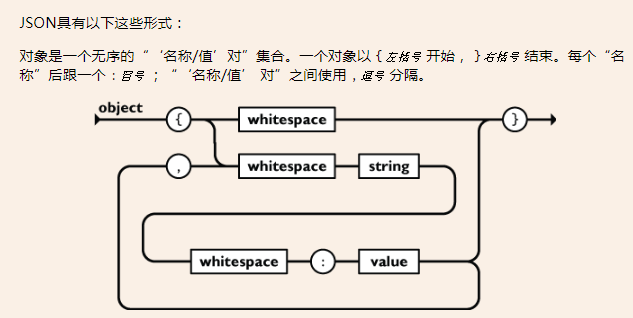
## JSON

### 概念

**JSON**(JavaScript Object Notation) 是一种轻量级的数据交换格式。

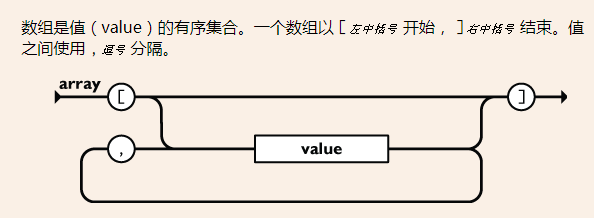
概括:json是一种特殊格式字符串

### Object格式



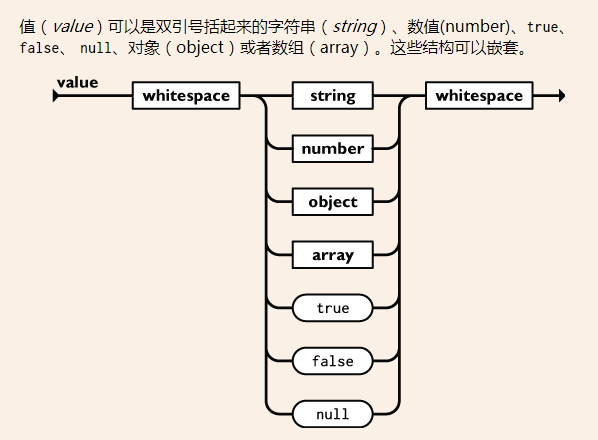
lg: {key1:"value1","key2":"value2"…..}

### Array格式



lg: ["value1","value2","value3"]

### 复杂格式



复杂格式可以是上述2中操作的无限层级嵌套. **value可以嵌套**

**lg:**

["value1","value2",123,true,{key1:["valu1",{key3:"value3",key4:"value4"}]}]

### 关于POST调整为get问题!!!!!!!!!

中午吃完饭,自己维护下

将课前资料中文件下的webapp文件夹导入jt-manage中覆盖原来的 即可完成post与get切换.自己完成!!!!!!

