JUI富客户端框架使用手册

目录

[概述 5](#_Toc122103407)

[设计思路 5](#_Toc122103408)

[学习JUI的建议 5](#_Toc122103409)

[JUI区别于其它JS框架，最大的优点 5](#_Toc122103410)

[版权声明 6](#_Toc122103411)

[JUI团队介绍 6](#_Toc122103412)

[PublicCMS作者对JUI的看法 7](#_Toc122103413)

[PublicCMS对JUI的升级 7](#_Toc122103414)

[HTML扩展 8](#_Toc122103415)

[Ajax链接扩展 8](#_Toc122103416)

[当前navTab中链接ajax post扩展 9](#_Toc122103417)

[dialog链接扩展 9](#_Toc122103418)

[navTab链接扩展 10](#_Toc122103419)

[Tab组件扩展 11](#_Toc122103420)

[Accordion组件 11](#_Toc122103421)

[容器高度自适应 11](#_Toc122103422)

[CSS Table 12](#_Toc122103423)

[在线编辑器 12](#_Toc122103424)

[分页组件 12](#_Toc122103425)

[ajaxTodo扩展 14](#_Toc122103426)

[dwzExport列表数据导出 14](#_Toc122103427)

[Input alt扩展 15](#_Toc122103428)

[Tree扩展 15](#_Toc122103429)

[Panel扩展 （该特性在dwz-for-Public-CMS中已删除） 16](#_Toc122103430)

[日历控件 16](#_Toc122103431)

[url变量替换 18](#_Toc122103432)

[checkbox全选、反选 19](#_Toc122103433)

[uploadify多文件上传 19](#_Toc122103434)

[combox组件 20](#_Toc122103435)

[suggest+lookup+主从结构 21](#_Toc122103436)

[查找带回 21](#_Toc122103437)

[主从结构 22](#_Toc122103438)

[Ajax表单 23](#_Toc122103439)

[表单查询 23](#_Toc122103440)

[普通Ajax表单提交 24](#_Toc122103441)

[服务器端响应 25](#_Toc122103442)

[文件上传表单提交 27](#_Toc122103443)

[服务器端响应 27](#_Toc122103444)

[Java服务器端表单处理示例 27](#_Toc122103445)

[JUI js库介绍 30](#_Toc122103446)

[JUI框架初始化 30](#_Toc122103447)

[jui.core.js 30](#_Toc122103448)

[jui.ui.js 30](#_Toc122103449)

[jui.ajax.js 30](#_Toc122103450)

[jui.alertMsg.js 30](#_Toc122103451)

[jui.jDialog.js 30](#_Toc122103452)

[jui.accordion.js 30](#_Toc122103453)

[jui.barDrag.js 31](#_Toc122103454)

[jui.navTab.js 31](#_Toc122103455)

[jui.scrollCenter.js 31](#_Toc122103456)

[jui.cssTable.js 31](#_Toc122103457)

[jui.tree.js 31](#_Toc122103458)

[jui.theme.js 31](#_Toc122103459)

[jui.validate.method.js 31](#_Toc122103460)

[jui.validate.zh.js 31](#_Toc122103461)

[jui.contextmenu.js 31](#_Toc122103462)

[jui.database.js 32](#_Toc122103463)

[jui.datepicker.js 32](#_Toc122103464)

[jui.combox.js 32](#_Toc122103465)

[jui.checkbox.js 32](#_Toc122103466)

[jui.uitl.date.js 32](#_Toc122103467)

[jui. regional.zh.js 32](#_Toc122103468)

[jui.validate.method.js 32](#_Toc122103469)

[Javascript混淆和压缩 33](#_Toc122103470)

[JUI如何中使用打包的js 33](#_Toc122103471)

[常见问题及解决 34](#_Toc122103472)

[JUI中如何整合第三方jQuery插件 34](#_Toc122103473)

[Error loading XML document: jui.frag.xml 34](#_Toc122103474)

[IIS不能用Ajax访问\*.html后缀的页面 34](#_Toc122103475)

[多个navTab页面或dialog页面ID冲突，解决方法 35](#_Toc122103476)

[如何自定义JUI分页参数参数 35](#_Toc122103477)

[如何关闭loading 35](#_Toc122103478)

[JUI局部刷新怎样做? 35](#_Toc122103479)

[补充说明和常见问题(xiaosuiba) 37](#_Toc122103480)

# 概述

JUI富客户端框架(jQuery RIA framework), 是中国人自己开发的基于jQuery实现的Ajax RIA开源框架.

JUI富客户端框架设计目标是**简单实用、扩展方便、快速开发、RIA思路、轻量级**

JUI框架支持用html扩展的方式来代替javascript代码, 只要懂html语法, 再参考JUI使用手册就可以做ajax开发.

开发人员不写javascript的情况下, 也能用ajax做项目和使用各种UI组件. 基本可以保证程序员不懂javascript, 也能使用各种页面组件和ajax技术. 如果有特定需求也可以扩展JUI做定制化开化.

做ajax项目时需要写大量的javascript才能达到满意的效果. 国内很多程序员javascript不熟, 大大影响了开发速度. 使用JUI框架自动邦定javascript效果. 不需要开发人员去关心javascript怎么写, 只要写标准html就可以了. JUI简单扩展了html标准, 给HTML定义了一些特别的class 和attribute. JUI框架会找到当前请求结果中的那些特别的class 和attribute, 并自动关联上相应的js处理事件和效果.

JUI基于jQuery可以非常方便的定制特定需求的UI组件, 并以jQuery插件的形式发布出来. 如有需要也可做定制化开发.

欢迎大家提出建议, 我们将在下一版本中进一步调整和完善功能．

JUI富客户端框架是开源项目, 可以免费获取源码. 希望有多的开发人员使用, 共同推进国内整体ajax开发水平.

在线演示地址 http://j-ui.com/

在线文档http://j-ui.com//doc/dwz-user-guide.pdf

Google Code下载: http://code.google.com/p/dwz/

## 设计思路

第一次打开页面时载入界面到客户端, 之后和服务器的交互只是数据交互, 不占用界面相关的网络流量.

支持HTML扩展方式来调用JUI组件.

标准化Ajax开发, 降低Ajax开发成本.

## 学习JUI的建议

刚接触JUI的人可能感觉JUI文档太少、入门困难，原因都是没有掌握正确的学方法。建议按下面的步骤来学习JUI框架：

* 通读JUI文档，很多新手提的问题文档中都写了。
* 看demo每个组件演示效果和代码（留意组件html结构）。
* 建议安装firebug，用firebug看html结构、CSS和调试JS都非常方便。见附录一 firebug介绍。
* 对于初学者不建议看JUI全部源码，但还是非常有必要看看**jui.ui.js**和**jui.ajax.js**
* 可以从google code下载dwz\_thinkphp版本，结合php后台去理解JUI和服务器端的交互方式

## JUI区别于其它JS框架，最大的优点

* 完全开源，源码没有做任何混淆处理，方便扩展
* CSS和js代码彻底分离，修改样式方便
* 简单实用，扩展方便，轻量级框架，快速开发
* 仍然保留了html的页面布局方式
* 支持HTML扩展方式调用UI组件，开发人员不需写js
* 只要懂html语法不需精通js，就可以使用ajax开发后台
* 基于jQuery，UI组件以jQuery插件的形式发布，扩展方便

## 版权声明

* JUI框架的源代码完全开放，在Apache License 2.0许可下, 可免费应用于个人或商业目的。
* 欢迎各大网站转载下载版本。
* 禁止把JUI框架包装成类外一个UI框架出售。

## JUI团队介绍

JUI研发组开发人员目前是3人（兼职）

杜权从事UI设计工作，有10年以上UI设计经验。做过至少1500个网站的UI设计。

吴平主要做Java web开发，兼ajax开发。一直从事电子商务、企业建站平台开发工作。目前就职于支付宝应用架构师职位。

张慧华主要做Java web开发，兼ajax开发。以前也是电子商务、企业建站平台开发工作。从2009年4月开始从事建筑能效评估IT解决方案。目前从Java开发转型做HTML5手机APP。

以前我们做的大部份java项目都用了Ajax，项目开发过程中经常自己做一些UI组件和界面效果。我们对RIA非常感兴趣，业余时间就做了JUI富客户端框架。JUI框架中的UI组件都是从我们做过的大量web项目中总结出来的，都是一些非常实用的UI组件。

**联系方式**

杜权(UI设计) d@j-ui.com

吴平(Ajax开发) w@j-ui.com

张慧华(Ajax开发) z@j-ui.com

**官方微博(欢迎加入)** http://weibo.com/dwzui

jQuery.JUI-jUI-1群(满员) 107983317

jQuery.JUI-jUI-2群(满员) 69611933

jQuery.JUI-jUI-3群(满员) 20866231

jQuery.JUI-jUI-4群(满员) 369203

jQuery.JUI-jUI-5群(满员) 85031937

jQuery.JUI-jUI-6群(欢迎加入) 172602882

jQuery.JUI-jUI-7群(满员) 210322217

jQuery.JUI-jUI-8群(欢迎加入) 139067378

**合作电话：010-52897073**

## PublicCMS作者对JUI的看法

随着技术革新jui已经推出广大技术人员视野，但是对于CMS这种特殊应用，服务器端渲染不可避免，我们对springmvc扩展出一套指令体系用于数据查询，而不是使用controller提供数据查询接口或查询页面。

在制作网站过程技术人员不需要修改程序，而是直接将设计稿制作普通html，然后用数据指令中的数据替换掉html中的演示数据。这套技术栈是cms中不可缺少的，我们也将它用在了后台功能中。JUI的设计思路使得整个后台的开发很简洁，我们在产品盈利后也曾经小小的赞助支持了几次JUI的原作者。

我们曾经反复寻找JUI的替代品，但是并没有找到更优秀的，JUI在内存管理、执行性能上都很优秀，被人诟病的就是后续更新乏力，UI逐渐老旧，不能很好的前端工程化。但是它真的很简单很轻量。

有一些问题我们也无法避免，但是我们还是觉得再它基础上进行一些升级改造、简化、美化，希望有同样需求的技术人员也能多一个选择。没有过时的技术，只有适合的和不适合的，最近又出现的htmx框架其实也还是JUI这种思想复现。

## PublicCMS对JUI的升级

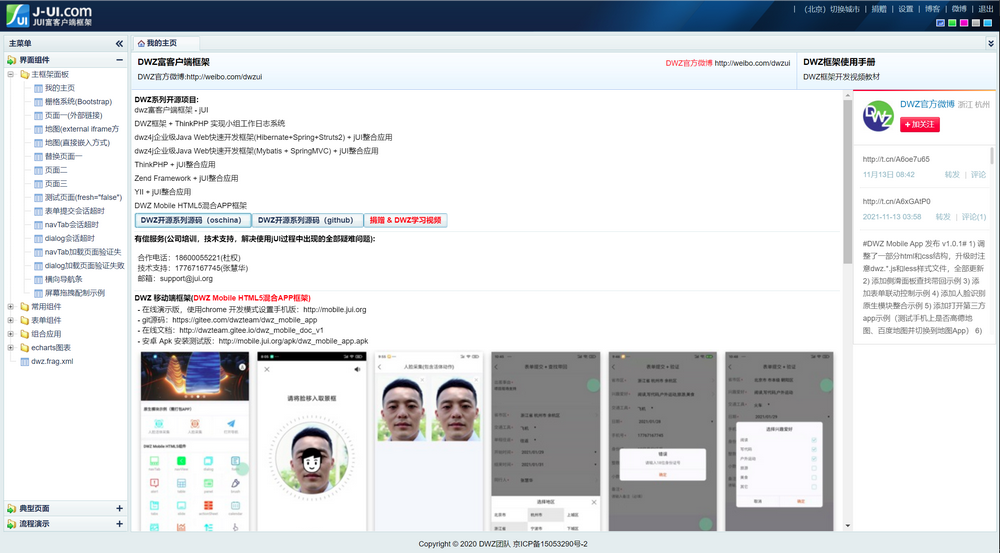
源码地址<https://gitee.com/sanluan/dwz-for-Public-CMS>

演示地址<https://www.publiccms.com/ui/>

dwz for Public CMS是publiccms开发团队自行维护的jui版本，其中界面在2019年由北京同创蓝宇科技有限公司公司帮忙进行了全新设计，对jui进行了大量修改维护，基于原理常用组件没有变化：

1. 样式调整
2. jquery升级到3.6.2
3. 修改为了HTML5
4. 增加了表格双击编辑
5. 编辑器支持百度编辑器、CK编辑器、TinyMCE编辑器、KindEditor、Codemirror编辑器
6. 编辑器js脚本按需加载，销毁后回收内存，提交表单时自动刷新到文本域
7. 减少了图片使用量
8. 改大了字体按钮等
9. 风格切换加载模式修改防止重复请求样式表
10. 修复百度编辑器图片居中bug
11. 增加繁体翻译功能
12. 百度编辑器升级
13. 增加异步树
14. 增加日文语言文件

原版JUI界面



dwz for Public CMS界面

电脑萤幕的截图

描述已自动生成

# HTML扩展

支持HTML扩展方式来调用JUI组件

## Ajax链接扩展

<a href=”xxx” target=“ajax” [rel=“boxId”]>

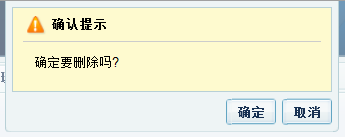
示例: <a href="w\_alert.html" target="ajax" rel="container">提示窗口</a>

## 当前navTab中链接ajax post扩展

<a href="user.do?method=remove" target="**ajaxTodo**">删除</a>

或

<a href="user.do?method=remove" target="**ajaxTodo**" title="确定要删除吗?">删除</a>



Title为可选项，如果设置，点击后将调用alertMsg.confirm与用户交互确认或取消，Title值为提示信息.Target值为ajaxTodo时会自动关联如下JS。

$("a[target=ajaxTodo]", $p).each(**function**(){

$(**this**).click(**function**(event){

**var** $this = $(**this**);

**var** title = $this.attr("title");

**if** (title) {

alertMsg.confirm(title, {

okCall: **function**(){

ajaxTodo($this.attr("href"));

}

});

} **else** {

ajaxTodo($this.attr("href"));

}

event.preventDefault();

});

});

## dialog链接扩展

<a href=”xxx” target=“**dialog**” [rel=“**dialogId**”]>

A所指向页面将会在dialog 弹出层中打开，rel标识此弹出层的ID，rel为可选项。

Html标签扩展方式示例:

<a href="w\_dialog.html" target="dialog" rel="page2">弹出窗口</a>

或

<a href="demo\_page1.html" target="dialog" [max=true, mask=true, maxable=true, minable=true, resizable=true,drawable=true] rel="dlg\_page1" title="[自定义标题]" width="800" height="480">打开窗口一</a>

Max 属性表示此dialog打开时默认最大化, mask表示打开层后将背景遮盖. maxable: dialog 是否可最大化，

minable: dialog 是否可最小化，

mixable: dialog 是否可最大化

resizable: dialog 是否可变大小

drawable: dialog 是否可拖动

width: dialog 打开时的默认宽度

height: dialog 打开时默认的高度

width,height分别打开dialog时的宽度与高度．

fresh: 重复打开dialog时是否重新载入数据，默认值true,

close: 关闭dialog时的监听函数，需要有boolean类型的返回值，

param: close监听函数的参数列表，以json格式表示，例{msg:’message’}

关闭窗口：

在弹出窗口页面内放置<button class="close" value="关闭"></button>即可。

JS调用方式示例:

$.pdialog.open(url, dlgId, title);

或

$.pdialog.open(url, dlgId, title, options);

options:{width:100px,height:100px,max:true,mask:true,mixable:true,minable:true,resizable:true,drawable:true,fresh:true,close:”function”, param:”{msg:’message’}”}, 所有参数都是可选项。

关闭dialog层：

$.pdialog.close(dialog); 参数dialog可以是弹出层jQuery对象或者是打开dialog层时的dlgId.

或者

$.pdialog.closeCurrent(); 关闭当前活动层。

$.pdialog.reload(url, {data:{}, dialogId:"", callback:null})

刷新dialogId指定的dialog，url：刷新时可重新指定加载数据的url, data：为加载数据时所需的参数。

## navTab链接扩展

<a href=”xxx” target=“navTab” [rel=“tabId”]>

**示例:**

<a href="url" target="navTab" >默认页面</a>

<a href="url" target="navTab" rel="page1" title="自定义标签名" fresh="false">自定义页面</a>

<a href="url" target="navTab" external="true">iframe方式打开</a>



target=navTab: 自动关联调用navTab组件

rel: 为navtab的ID值，后续可以用来重载该页面时使用，如当前页新增或删除数据可以通过该ID进行通知JS重载。注意rel的值区分大小写**.**

fresh: 表示重复打开navTab时是否重新加载数据

external: 为external="true"或者href是外网连接时，以iframe方式打开navTab页面

**Js调用**

navTab.openTab(tabid, url, { title:”New Tab”, fresh:false, data:{} });

其中data:{} json格式的数据

## Tab组件扩展

开发人员不需写任何javacsript, 只要使用下面的html结构就可以.

<div class="tabs">

<div class="tabsHeader">

<div class="tabsHeaderContent">

<ul>

<li class="selected"><a href="#"><span>标题1</span></a></li>

<li><a href="#"><span>标题2</span></a></li>

</ul>

</div>

</div>

<div class="tabsContent" style="height**:**150px**;**">

<div>内容1</div>

<div>内容2</div>

</div>

</div>

## Accordion组件

<div class="accordion" [fillSpace=”xxxKey”]>

<div class="accordionHeader">

<h2><span>icon</span>面板1</h2>

</div>

<div class="accordionContent" style="height**:**200px">

内容1

</div>

<div class="accordionHeader">

<h2><span>icon</span>面板2</h2>

</div>

<div class="accordionContent">

内容2

</div>

<div class="accordionHeader">

<h2><span>icon</span>面板3</h2>

</div>

<div class="accordionContent">

内容3

</div>

</div>

## 容器高度自适应

容器高度自适应, 只要增加扩展属性layoutH=”xx”, 单位是像素.

LayoutH表示容器内工具栏高度. 浏览器窗口大小改变时容器高度自适应, 但容器内工具栏高度是固定的, 需要告诉js工具栏高度来计算出内容的高度.

示例:

<div class=”layoutBox”>

<div layoutH=“150”>内容</div>

</div>

注意： layoutH=“150”的高度是根据含有class=”layoutBox”的父容器div动态更新的

## CSS Table

原生html + CSS实现，无js处理效果、最简单、最基本、性能最高的table。

在table标签上增加class="list", table外面包一个<div layoutH="xx">实现table固定高度

<div layoutH="120">

<table class="list" width="98%">

<thead>

<tr>

<th colspan="2">客户信息</th>

<th colspan="2">基本信息</th>

<th colspan="3">资料信息</th>

</tr>

<tr>

<th width="80">客户号</th>

<th width="100">客户名称</th>

<th width="100">客户划分</th>

<th>证件号码</th>

<th align="right" width="100">信用等级</th>

<th width="100">企业性质</th>

<th width="100">建档日期</th>

</tr>

</thead>

<tbody>

<tr>

<td>iso127309</td>

<td>北京市政府咿呀哟</td>

<td>政府单位</td>

<td>0-0001027766351528</td>

<td>四等级</td>

<td>政府单位</td>

<td>2009-05-21</td>

</tr>

</tbody>

</table>

</div>

## 在线编辑器

在textarea标签上增加class="editor"

示例:

<textarea class="editor" name="description" rows="15" cols="80">内容</textarea>

## 分页组件

分页思路服务器返回当前页的数据，总条数，再由js来生成分页标签。分页是配合服务器端来处理的, 不是存js做的分页。

因为如果数据量很大，比如有好几百页，存js分页就是悲剧了，存js分页是必须一次载入所有数据，性能很慢。

分页组件参数要由服务器传过来targetType，totalCount，numPerPage，pageNumShown，currentPage

框架会自动把下面的form中pageNum修改后，ajax重新发请求。下面这个form是用来存查询条件的

<form id="pagerForm" action="xxx" method="post">

<input type="hidden" name="pageNum" value="1" />/><!--【必须】value=1可以写死-->

<input type="hidden" name="numPerPage" value="20" /><!--【可选】每页显示多少条-->

<input type="hidden" name="orderField" value="xxx" /><!--【可选】查询排序-->

<input type="hidden" name="orderDirection" value="asc" /><!--【可选】升序降序-->

<!--【可选】其它查询条件，业务有关，有什么查询条件就加什么参数。

也可以在searchForm上设置属性rel=”pagerForm”，js框架会自动把searchForm搜索条件复制到pagerForm中 -->

<input type="hidden" name="name" value="xxx" />

<input type="hidden" name="status" value="active" />

……

</form>

**分页组件处理分页流程：**

1. pagerForm中缓存了当前的查询条件，加上一个pageNum字段
2. 点击分页时动态修改pageNum,重新提交pagerForm

**分页组件使用方法：**

<div class="panelBar pageBar">

<ul class="pagination">

<li>

<span>共54条</span>

<select class="combox" name="numPerPage" onchange="navTabPageBreak({pageNum:1,numPerPage:this.value});">

<option value="30" selected="true">每页30条</option>

<option value="50">每页50条</option>

<option value="100">每页100条</option>

<option value="500">每页500条</option>

</select>

</li>

<li class="disabled">

<a class="previous" href="javascript:;" onclick="navTabPageBreak({pageNum:1});return false;">上一页</a>

</li>

<li class="selected"><a href="javascript:;" onclick="navTabPageBreak({pageNum:1});return false;">1</a></li>

<li><a href="javascript:;" onclick="navTabPageBreak({pageNum:2});return false;">2</a></li>

<li>

<a class="next" href="javascript:;" onclick="navTabPageBreak({pageNum:2});return false;">下一页</a>

</li>

<li class="jumpto">

<input type="text" size="4" value="1"/>

<a href="javascript:;" class="goto" onclick="navTabPageBreak({pageNum:$(this).prev().val()});return false;"><i class="icon-large icon-chevron-right"></i></a>

</li>

</ul>

</div>

测试方法，navTabPageBreak函数参数pageNum从1改为2，就是第2页了

**分页函数说明:**

navTabPageBreak(args, rel) 处理navTab中的分页和排序

dialogPageBreak(args, rel) 处理dialog中的分页和排序

参数 args : {pageNum:"n", numPerPage:"n", orderField:"xxx", orderDirection:""}

参数 rel： 可选 用于局部刷新div id号

**注意：**

服务器端返回一个页面碎片，其中包括pagerForm, table, 和分页的div。只要把这个页面碎片组装好就行。

## ajaxTodo扩展

navTab页面上a链接添加target="ajaxTodo" 后框架会自动绑定相应的ajax处理。【参考jui.ajax.js】

可选a链接扩展属性[title="xxx"] 提示确认信息

**示例：**

<a href="/news.do?method=remove&id=${item.id}" target="ajaxTodo" title="确定要删除吗?">>删除</a>

<a href="/news.do?method=publish&id=${item.id}" target="ajaxTodo">发表</a>

框架自动绑定js

$("a[target=ajaxTodo]", $p).each(**function**(){

$(**this**).click(**function**(event){

ajaxTodo($(**this**).attr("href"));

event.preventDefault();

});

});

## dwzExport列表数据导出

链接添加target="dwzExport" 后框架会自动绑定相应的ajax处理。

targetType="navTab" 根据当期navTab页面中的pagerForm参数导出, 默认

targetType="dialog" 根据当期dialog页面中的pagerForm参数导出

title="实要导出这些记录吗?" 确认提示信息，可选项  
**示例：**

<a href="doc/dwz-team.xls" target="dwzExport" targetType="dialog" title="实要导出这些记录吗?">导出EXCEL</a>

## Input alt扩展

示例：

<input name="xxx" alt="请输入客户名称"/>

## Tree扩展

<ul class="tree [treeFolder treeCheck [expand|collapse]]" oncheck="kkk">

<li><a href="#" target="navTab" rel="main" tname="name" tvalue="value" checked="true">第一级菜单项 A</a>

<ul>

<li><a href="#" target="dialog" rel="dialog1" tname="name" tvalue="value" checked="true">第二级菜单项 A </a></li>

<li><a href="#">第二级菜单项 B </a></li>

<li><a href="#">第二级菜单项 C </a>

<ul>

<li><a href="#">第三级菜单项 A </a></li>

<li><a href="#">第三级菜单项 B </a></li>

</ul>

</li>

<li \_src="subtree.html?parentId=d">

<a href="#">第二级菜单项 d </a>

</li>

</ul>

</li>

<li><a href="#">第一级菜单项 B</a></li>

</ul>

树结构是按<ul>,<li>的嵌套格式构成，将最顶级的<ul>以class=”tree”标识即可。treeFolder, treeCheck, expand|collapse则为可选的。

treeFolder:在所有树节点前加上Icon图标

treeCheck:在所有树节点前加上checkbox，此时需要在<a> 加上三个扩展属性tname=””, tvalue=””, checked, 其中tname与tvalue对应该checkbox的name 与value属性

checked表示checkbox的默认状态是否checked.

expand与collapse：expand表示树的所有第一级节点默认是展开状态，collapse则表示所有第一级节点默认为折叠状态，当expand与collapse都没有时默认则会展开第一个节点。

扩展属性oncheck是自定义函数, 用来接收点击checkbox时返回值, 当点击非子树节点checkbox时返回数据格式为:{checked:true|false,items:{name:name, value:value}}, 当点击了树节点checkbox时, 此子树节点下所有的checkbox都将选中, 同时返回此子树节点下所有的checkbox的值, 格式为{checked:true|false, items:{{name:name, value:value}, {name:name, value:value}……}}

\_src:子树异步加载url，返回结果:（该特性为dwz-for-Public-CMS新开发特性）

<ul>

<li><a href="#">第三级菜单项 A </a></li>

<li><a href="#">第三级菜单项 B </a></li>

</ul>

## Panel扩展 （该特性在dwz-for-Public-CMS中已删除）

## 日历控件

<input type="text" name="xxx" class="date" [dateFmt="yyyy-MM-dd"] [minDate="{%y-80}"] [maxDate="{%y+5}"]/>

**日期格式：**

\* Field | Full Form | Short Form

\* -------------+--------------------+-----------------------

\* Year | yyyy (4 digits) | yy (2 digits), y (2 or 4 digits)

\* Month | MMM (name or abbr.)| MM (2 digits), M (1 or 2 digits)

\* | NNN (abbr.) |

\* Day of Month | dd (2 digits) | d (1 or 2 digits)

\* Day of Week | EE (name) | E (abbr)

\* Hour (1-12) | hh (2 digits) | h (1 or 2 digits)

\* Hour (0-23) | HH (2 digits) | H (1 or 2 digits)

\* Hour (0-11) | KK (2 digits) | K (1 or 2 digits)

\* Hour (1-24) | kk (2 digits) | k (1 or 2 digits)

\* Minute | mm (2 digits) | m (1 or 2 digits)

\* Second | ss (2 digits) | s (1 or 2 digits)

\* AM/PM | a |

定义日期范围属性minDate,maxDate静态格式y-M-d或y-M或y，支持以下几种写法:

minDate="2000-01-15" maxDate="2012-12-15"

minDate="2000-01" maxDate="2012-12"

minDate="2000" maxDate="2012"

定义日期范围属性minDate,maxDate动态态格式%y-%M-%d或%y-%M或%y，支持以下几种写法:

minDate="{%y-10}-%M-%d" maxDate="{%y}-%M-{%d+1}"

minDate="{%y-10}-%M" maxDate="{%y+10}-%M"

minDate="{%y-10}" maxDate="{%y+10}"

**示例：**

<p>

<label>默认格式：</label>

<input type**=**"text" name**=**"date1" class**=**"date" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy-MM-dd</span>

</p>

<p>

<label>定义日期范围：</label>

<input type**=**"text" name**=**"date2" value**=**"2000-01-15" class**=**"date" minDate**=**"2000-01-15" maxDate**=**"2012-12-15" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

</p>

<p>

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date3" class**=**"date" dateFmt**=**"yyyy/MM/dd" minDate**=**"2000-01" maxDate**=**"2012-06" readonly**=**"true" />

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy/MM/dd</span>

</p>

<p>

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date4" class**=**"date" dateFmt**=**"dd/MM/yyyy" minDate**=**"2000" maxDate**=**"2012" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">dd/MM/yyyy</span>

</p>

<p>

<label>动态参数minDate：</label>

<input type**=**"text" name**=**"date5" class**=**"date" dateFmt**=**"dd/MM/yy" minDate**=**"{%y-10}-%M-%d" maxDate**=**"{%y}-%M-{%d+1}"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">dd/MM/yy</span>

</p>

<p>

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date6" class**=**"date" dateFmt**=**"yyyyMMdd" minDate**=**"2000-01-01" maxDate**=**"2020-12-31"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyyMMdd</span>

</p>

<p>

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date7" class**=**"date" dateFmt**=**"yyyy年MM月dd日" minDate**=**"2000-01-01" maxDate**=**"2020-12-31"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy年MM月dd日</span>

</p>

<p>

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date8" class**=**"date" dateFmt**=**"y年M月d日" minDate**=**"2000-01-01" maxDate**=**"2020-12-31"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">y年M月d日</span>

</p>

<div class**=**"divider"></div>

<h3>日期 + 时间</h3>

<div class**=**"unit">

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date10" class**=**"date" dateFmt**=**"yyyy-MM-dd HH:mm:ss" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy-MM-dd HH:mm:ss</span>

</div>

<div class**=**"unit">

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date11" class**=**"date" dateFmt**=**"yyyy-MM-dd HH:mm" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy-MM-dd HH:mm</span>

</div>

<div class**=**"unit">

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date12" class**=**"date" dateFmt**=**"yyyy-MM-dd HH:ss" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">yyyy-MM-dd HH:ss</span>

</div>

<div class**=**"unit">

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date13" class**=**"date" dateFmt**=**"y年M月d日 H点" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">y年M月d日 H点</span>

</div>

<div class**=**"unit">

<label>自定义日期格式：</label>

<input type**=**"text" name**=**"date14" class**=**"date" dateFmt**=**"EEE HH:mm:ss" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">EEE HH:mm:ss</span>

</div>

<div class**=**"unit">

<label>自定义只有时间：</label>

<input type**=**"text" name**=**"date15" class**=**"date" dateFmt**=**"HH:mm:ss" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">HH:mm:ss</span>

</div>

<div class**=**"unit">

<label>自定义时间：</label>

<input type**=**"text" name**=**"date16" class**=**"date" dateFmt**=**"HH:mm" mmStep**=**"15" readonly**=**"true"/>

<a class**=**"inputDateButton" href**=**"javascript:;">选择</a>

<span class**=**"info">HH:mm</span>

</div>

## url变量替换

HTML扩展方式navTab, dialog, ajaxTodo 的url支持变量替换。例如：*\_\_URL\_\_/edit/id/{xxx}*

大括号内的xxx就是变量名，主要功能是结合table组件一起使用，下面是dwz\_thinkphp中用户列表的示例：

下图中的删除、编辑、修改密码都是用了url变量替换：



删除、编辑、修改密码使用了变量*{sid\_user}*

<tbody>中<tr target=*"sid\_user"* rel=*"{$vo['id']}"*>

当选中一行时,tr上的rel值会自动替换到url变量中.

注意url变量名*{sid\_user}*和tr的target=*"sid\_user"*保持一致.

**代码示例：**

<a class=*"delete"* href=*"\_\_URL\_\_/foreverdelete/id/{sid\_user}/navTabId/\_\_MODULE\_\_"* target=*"ajaxTodo"* title=*"你确定要删除吗？"* warn=*"请选择用户"*><span>删除</span></a>

<a class=*"edit"* href=*"\_\_URL\_\_/edit/id/{sid\_user}"* target=*"dialog"* mask=*"true"* warn=*"请选择用户"*><span>编辑</span></a>

<a class=*"icon"* href=*"\_\_URL\_\_/password/id/{sid\_user}"* target=*"dialog"* mask=*"true"* warn=*"请选择用户"*><span>修改密码</span></a>

<table class=*"list"* width=*"100%"* layoutH=*"116"*>

<thead>

<tr>

<th width=*"60"*>编号</th>

<th width=*"100"*>用户名</th>

<th>昵称</th>

<th>Email</th>

<th width=*"100"*>添加时间</th>

<th width=*"120"*>上次登录</th>

<th width=*"80"*>登录次数</th>

<th width=*"80"*>状态</th>

</tr>

</thead>

<tbody>

<volist id=*"vo"* name=*"list"*>

<tr target=*"sid\_user"* rel=*"{$vo['id']}"*>

<td>{$vo['id']}</td>

<td>{$vo['account']}</td>

<td>{$vo['nickname']}</td>

<td>{$vo['email']}</td>

<td>{$vo['create\_time']|date="Y-m-d",###}</td>

<td>{$vo['last\_login\_time']|date="Y-m-d H:i:s",###}</td>

<td>{$vo['login\_count']}</td>

<td>{$vo['status']|showStatus=$vo['id']}</td>

</tr>

</volist>

</tbody>

</table>

## checkbox全选、反选

checkbox全选、反选。（demo  表单组件 多选框/单选框）

<label><input type="checkbox" name="c1" value="1" />选择1</label>

<label><input type="checkbox" name="c1" value="2" />选择2</label>

<label><input type="checkbox" name="c1" value="3" />选择3</label>

<input type="checkbox" class="checkboxCtrl" group="c1" />全选

<button type="button" class="checkboxCtrl" group="c1" selectType="invert">反选</button>

## uploadify多文件上传

<div id="fileQueue" class="fileQueue"></div>

<input id="testFileInput" type="file" name="image"

   uploader="uploadify/scripts/uploadify.swf"

   cancelImg="uploadify/cancel.png"

   script="ajaxDone.html"

   scriptData="{PHPSESSID:'xxx', ajax:1}"

   folder="/folder"

   fileQueue="fileQueue"

   [onComplete="uploadifyComplete"]

   [onAllComplete="uploadifyAllComplete"] />

参数说明：

uploader： flash组件uploadify.swf的访问路径

cancelImg: 取消按钮使用的图片路径

script:    服务器端处理上传文件的路径

scriptData:上传文件时需要传递给服务器的其他参数，是json格式

folder:     是服务器存储文件的目录

fileQueue：是上传进度显示区域

onAllComplete：可选参数，单个文件上传完时触发，参数有：

      event： event 事件对象

      Id：     上传进度队列的id

      fileObj: 是一个包含上传文件信息的对象，包括的信息有：

               name: 文件名

               filePath: 上传文件在服务器端的路径

               size:      文件的大小

               creationDate:文件创建的时间

               modificationDate：文件最后更改的时间

               type：是以"."开始的文件扩展名

      response:服务器端处理完上传文件后返回的文本

      data:    包含有两个参数的对象，

               fileCount:上传队列中还剩下的文件数

               speed:以KB/s为单位的文件上传平均速度

uploadifyAllComplete：可选参数，全部文件上传完成时调用的函数，参数有：

      event:event 事件对象

      data: 是一个包含以下信息的对象,

            filesUploaded: 已经上传的文件总数

            errors:        上传出错的文件总数

            allBytesLoaded：已经上传文件的总大小

            speed:          以KB/s为单位的上传文件的平均速度

以下3个方法是jui.ajax.js中定义的用于文件上传的会调函数：

**function** uploadifyAllComplete(event, data){

**if** (data.errors) {

**var** msg = "The total number of files uploaded: "+data.filesUploaded+"\n"

+ "The total number of errors while uploading: "+data.errors+"\n"

+ "The total number of bytes uploaded: "+data.allBytesLoaded+"\n"

+ "The average speed of all uploaded files: "+data.speed;

alert("event:" + event + "\n" + msg);

}

}

**function** uploadifyComplete(event, queueId, fileObj, response, data){

JUI.ajaxDone(JUI.jsonEval(response));

}

**function** uploadifyError(event, queueId, fileObj, errorObj){

alert("event:" + event + "\nqueueId:" + queueId + "\nfileObj.name:"

+ fileObj.name + "\nerrorObj.type:" + errorObj.type + "\nerrorObj.info:" + errorObj.info);

}

## combox组件

在传统的select 用class 定义：class=”combox”, html 扩展：保留原有属性name, 增加了属性：ref。

ref 属性则是为了做级联定义的，ref所指向的则是当前combox值改变成引起联动的下一级combox，ref用下一级combox的id属性来赋值。

注意：一般combox没必要设置id属性，只要级联时需要设置子级id等于父级ref，不同navTab和dialog中combox组件id必须唯一

**以下是级联示例：**

<select class="combox" name="province" ref="combox\_city" refUrl="city.do?code={value}">

<option value="all">所有省市</option>

<option value="bj">北京</option>

<option value="sh">上海</option>

</select>

<select class="combox" name="city" id="combox\_city" ref="combox\_area" refUrl=" area.do?code={value}">

<option value="all">所有城市</option>

</select>

<select class="combox" name="area" id="combox\_area">

<option value="all">所有区县</option>

</select>

**服务器端返回json格式：**

[

["all", "所有城市"],

["bj", "北京市"]

]

## suggest+lookup+主从结构

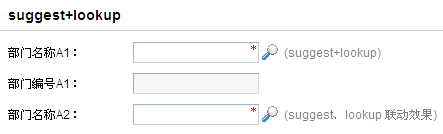
jui.database.js主要功能是数据库操作相关的界面组件。主要分为2部分，分别是查找带回和主从结构。

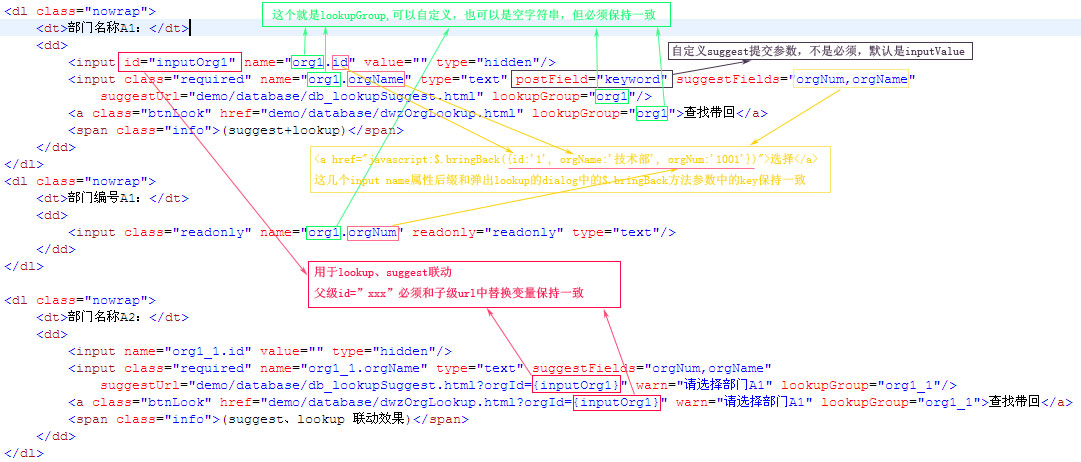
* 查找带回：lookup、suggest、lookup附件(文件上传带回)、多选查找带回multLookup几个jQuery插件，以及$.bringBack、$.bringBackSuggest两个配套查找带回工具方法
* 主从结构：itemDetail

suggest+lookup+主从结构 请参照demo：界面组件 🡪 常用组件 🡪 suggest+lookup+主从结构

### 查找带回

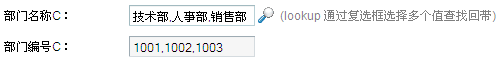
lookup、suggest都支持联动效果，比如类似选省份、城市联动效果。支持自定义查找带回主键lookupPk, 可选项默认为id。





**lookup 通过复选框选择多个值查找回带示例：**

请参照dwz-ria中 demo/database/ db\_widge.html和demo/database/dwzOrgLookup2.html页面

****

<button type="button" multLookup="orgId" warn="请选择部门">选择带回</button>

<input type="checkbox" name="orgId" value="{id:'1', orgName:'技术部', orgNum:'1001'}"/>

<input type="checkbox" name="orgId" value="{id:'2', orgName:'人事部', orgNum:'1002'}"/>

<input type="checkbox" name="orgId" value="{id:'3', orgName:'销售部', orgNum:'1003'}"/>

### 主从结构

针对主表和从表的数据库结构设计，实现主从结构复合表单，动态添加、删除从表字段。

请参照dwz-ria中 demo/database/ db\_widge.html

<table class="list nowrap itemDetail" addButton="新建从表1条目" width="100%">

<thead>

<tr>

<th type="text" name="items.itemString" size="12" fieldClass="required">从字符串</th>

<th type="text" name="items.itemInt" size="12" fieldClass="digits">从整数</th>

<th type="text" name="items.itemFloat" size="12" fieldClass="number">从浮点</th>

<th type="date" name="items.itemDate" size="12">从日期</th>

<th type="date" format="yyyy-MM-dd HH:mm:ss" name="items.itemDataTime" size="16">从日期时间</th>

<th type="lookup" name="jui.items.org.orgName" lookupGroup="items.org" lookupUrl="xxxUrl" suggestUrl="xxxUrl" suggestFields="orgName" size="12">部门名称</th>

<th type="enum" name="items.itemEnum" enumUrl="xxxUrl" size="12">从枚举</th>

<th type="attach" name="jui.items.attachment.fileName" lookupGroup="items.attachment" lookupUrl="xxxUrl" size="12">从附件</th>

<th type="del" width="60">操作</th>

</tr>

</thead>

<tbody></tbody>

</table>

<table>标签中class=”itemDetail” 必须用于关联主从结构js效果。addButton=”xxx”可选默认为”Add New”用来定义添加从表按钮的文字。

<th>标签中：type必填项，type类型有text、date、lookup、enum、attach、del

name必填项，定义子表字段名称

size可选项，默认size=”12”，定义从表input字段的长度

fieldClass可选项，用来定义表input字段的class

lookupGroup当type=”lookup” 或type=”attach”时必填

lookupUrl当type=”lookup”时lookupUrl和suggesUrl至少填一项，当type=”attach”时必填

suggestUrl当type=”lookup”时lookupUrl和suggesUrl至少填一项

suggestFields当type=”lookup”并且有suggestUrl时必填

enumUrl当type=”enum”时必填

# Ajax表单

Ajax表单相关的操作封装在jui.ajax.js中。表单查询、分页、表单提交js方法都已经封装在里面了。开发人员自己不需写js，按标准使用就可以了。

## 表单查询

JUI中定义表单查询和分页都是用这个pagerForm来临时存查询条件。所以需要在查询页面上放下面的form

<form id="pagerForm" action="xxx" method="post">

<input type="hidden" name="pageNum" value="1" /><!--【必须】value=1可以写死-->

<input type="hidden" name="numPerPage" value="20" /><!--【可选】每页显示多少条-->

<input type="hidden" name="orderField" value="xxx" /><!--【可选】查询排序-->

<!--【可选】其它查询条件，业务有关，有什么查询条件就加什么参数-->

<input type="hidden" name="status" value="active" />

</form>

**ajax表单查询**

<form action="xxx" method="post" onsubmit="**return** navTabSearch(**this**)">

或

<form action="xxx" method="post" onsubmit="**return** dialogSearch(**this**)">

**修改每页显示行数**

<select class="combox" name="numPerPage" onchange="navTabPageBreak({numPerPage:**this**.value})">

<option value="20">20</option>

<option value="50">50</option>

<option value="100">100</option>

<option value="200">200</option>

</select>

/\*\*

\* 处理navTab弹出层上的查询, 会重新载入当前navTab

\* **@param** {Object} form

\*/

**function** navTabSearch(form){

navTab.reload(form.action, $(form).serializeArray());

**return** false;

}

/\*\*

\* 处理dialog弹出层上的查询, 会重新载入当前dialog

\* **@param** {Object} form

\*/

**function** dialogSearch(form){

$.pdialog.reload(form.action, $(form).serializeArray());

**return** false;

}

/\*\*

\* 处理navTab中的分页和排序

\* **@param** args {pageNum:"n", numPerPage:"n", orderField:"xxx"}

\*/

**function** navTabPageBreak(args){

**var** form = \_getPagerForm(navTab.getCurrentPanel(), args);

**if** (form) navTab.reload(form.action, $(form).serializeArray());

}

/\*\*

\* 处理dialog中的分页和排序

\* **@param** args {pageNum:"n", numPerPage:"n", orderField:"xxx"}

\*/

**function** dialogPageBreak(args){

**var** form = \_getPagerForm($.pdialog.getCurrent(), args);

**if** (form) $.pdialog.reload(form.action, $(form).serializeArray());

}

**ajax表单查询完整示例：**

<div class="pageHeader">

<form action="/render.do?method=search" method="post" onsubmit="**return** navTabSearch(**this**)">

<input type="hidden" name="resourceStatus" value="${param.resourceStatus}"/>

<input type="hidden" name="pageNum" value="1" />

<input type="hidden" name="orderField" value="${param.orderField}" />

<div class="searchBar">

<div class="searchContent">

<select name="resourceType">

<option value="">全部栏目</option>

<c:forEach var="item" items="${model.resourceTypes}">

<option value="${item.id}" ${param.resourceType eq item.id?"selected":"" }>${item.name}</option>

</c:forEach>

</select>

<input name="keywords" type="text" size="25" value="${param.keywords}"/>

</div>

<div class="subBar">

<ul>

<li><div class="buttonActive"><div class="buttonContent"><button type="submit">检索</button></div></div></li>

</ul>

</div>

</div>

</form>

</div>

## 普通Ajax表单提交

JUI框架Ajax无刷新表单提交处理流程是：

1. ajax表单提交给服务器
2. 服务器返回一个固定格式json结构
3. js会调函数根据这个json数据做相应的处理

**注意：**

JUI框架默认的ajax表单提交都是返回json数据，告诉客户端操作是否成功，成功或失败提示信息，以及成功后的处理方式（刷新某个navTab或关闭某个navTab或navTab页面跳转）。

表单提交后服务器操作失败了，客户端接收statusCode和message后给出错误提示，表单页面是不动的。这样可以方便用户看到出错原因后直接修改表单数据再次提交，而不用重填整个表单数据。当然如果你还是喜欢表单提交后直接载入html页面也是没有问题的，参照jui.ajax.js自己扩展一下也是没问题的。

**JUI** **表单提交jui.ajax.js**

* Ajax 表单提交后自动调用默认回调函数, 操作成功或失败提示.

Form标签上增加onsubmit="**return** validateCallback(**this**);

* Ajax 表单提交后如果需要重新加载某个navTab或关闭dialog，可以使用jui.ajax.js中事先定义的方法**navTabAjaxDone/dialogAjaxDone**

注意：如果表单在navTab页面上使用navTabAjaxDone，表单在dialog页面上使用dialogAjaxDone

Form标签上增加onsubmit="**return** validateCallback(**this, navTabAjaxDone**)"

或onsubmit="**return** validateCallback(**this, dialogAjaxDone**)"

* Ajax 表单提交后如果需要做一些其它处理也可以自定义一个回调函数**xxxAjaxDone**。例如下面表单提交成功后关闭当前navTab, 或者重新载入某个tab.

Form标签上增加onsubmit="**return** validateCallback(**this,** **xxxAjaxDone**)"

### 服务器端响应

Ajax表单提交后服务器端需要返回以下json代码：

{

"statusCode":"200",

"message":"操作成功",

"navTabId":"",

"rel":"",

"callbackType":"closeCurrent",

"forwardUrl":""

}

**以下是jui.ajax.js中定义的navTabAjaxDone和dialogAjaxDone代码片段：**

/\*\*

\* navTabAjaxDone是JUI框架中预定义的表单提交回调函数．

\* 服务器转回navTabId可以把那个navTab标记为reloadFlag=1, 下次切换到那个navTab时会重新载入内容.

\* callbackType如果是closeCurrent就会关闭当前tab

\* 只有callbackType="forward"时需要forwardUrl值

\* navTabAjaxDone这个回调函数基本可以通用了，如果还有特殊需要也可以自定义回调函数.

\* 如果表单提交只提示操作是否成功, 就可以不指定回调函数. 框架会默认调用JUI.ajaxDone()

\* <form action="/user.do?method=save" onsubmit="return validateCallback(this, navTabAjaxDone)">

\*

\* form提交后返回json数据结构statusCode=JUI.statusCode.ok表示操作成功, 做页面跳转等操作. statusCode=JUI.statusCode.error表示操作失败, 提示错误原因.

\* statusCode=JUI.statusCode.timeout表示session超时，下次点击时跳转到JUI.loginUrl

\* {"statusCode":"200", "message":"操作成功", "navTabId":"navNewsLi", "forwardUrl":"", "callbackType":"closeCurrent"}

\* {"statusCode":"300", "message":"操作失败"}

\* {"statusCode":"301", "message":"会话超时"}

\*

\*/

**function** navTabAjaxDone(json){

JUI.ajaxDone(json);

**if** (json.statusCode == JUI.statusCode.ok){

**if** (json.navTabId){ //把指定navTab页面标记为需要“重新载入”。注意navTabId不能是当前navTab页面的

navTab.reloadFlag(json.navTabId);

} **else** { //重新载入当前navTab页面

navTabPageBreak();

}

**if** ("closeCurrent" == json.callbackType) {

setTimeout(**function**(){navTab.closeCurrentTab();}, 100);

} **else** **if** ("forward" == json.callbackType) {

navTab.reload(json.forwardUrl);

}

}

}

/\*\*

\* dialog上的表单提交回调函数

\* 服务器转回navTabId，可以重新载入指定的navTab. statusCode=JUI.statusCode.ok表示操作成功, 自动关闭当前dialog

\*

\* form提交后返回json数据结构,json格式和navTabAjaxDone一致

\*/

**function** dialogAjaxDone(json){

JUI.ajaxDone(json);

**if** (json.statusCode == JUI.statusCode.ok){

**if** (json.navTabId){

navTab.reload(json.forwardUrl, {}, json.navTabId);

}

$.pdialog.closeCurrent();

}

}

**示例:**

<form method="post" action="url" class="pageForm required-validate" onsubmit="**return** validateCallback(**this**);">

<div class="pageFormContent" layoutH="56">

<p>

<label>E-Mail：</label>

<input class="required email" name="email" type="text" size="30" />

</p>

<p>

<label>客户名称：</label>

<input class="required" name="name" type="text" size="30" />

</p>

</div>

<div class="formBar">

<ul>

<li>

<div class="buttonActive"><div class="buttonContent"><button type="submit">保存</button></div></div>

</li>

<li>

<div class="button"><div class="buttonContent"><button type="Button" class="close">取消</button></div></div>

</li>

</ul>

</div>

</form>

## 文件上传表单提交

因为Ajax不支持enctype="multipart/form-data" 所以用隐藏iframe来处理无刷新表单提交.

<form method="post" action="url" class="pageForm required-validate" enctype="multipart/form-data" onsubmit="**return** iframeCallback(**this**);">

或

<form method="post" action="url" class="pageForm required-validate" enctype="multipart/form-data" onsubmit="**return** iframeCallback(**this, [navTabAjaxDone/dialogAjaxDone]**);">

### 服务器端响应

**JUI-v1.2版本开始服务器返回和validateCallback格式保持一致：**

{

"statusCode":"200",

"message":"操作成功",

"navTabId":"",

"rel":"",

"callbackType":"closeCurrent",

"forwardUrl":""

}

**JUI-v1.2以前版本**使用隐藏iframe来处理无刷新表单提交时，服务器端需要返回以下js代码：

<script type="text/javascript">

**var** statusCode="200";

**var** message="操作成功";

**var** navTabId="";

**var** forwardUrl="";

**var** callbackType="closeCurrent"

**var** response = {statusCode:statusCode,

message:message,

navTabId:navTabId,

forwardUrl:forwardUrl,

callbackType:callbackType

};

**if**(window.parent.donecallback) window.parent.donecallback(response);

</script>

### Java服务器端表单处理示例

**public** **class** NewsAction **extends** BaseAction {

**private** NewsManager manager = *bf*.getManager(BeanManagerKey.*newsManager*);

**private** News news = manager.newNews();

**private** Collection<News> newsList;

**public** String add() {

**return** *INPUT*;

}

**public** String insert() {

manager.createNews(news);

**return** ajaxForwardSuccess(getText("msg.operation.success"));

}

**public** String edit() {

news = manager.getNews(**this**.getNewsId());

**return** *INPUT*;

}

**public** String update() {

News m = manager.getNews(newsId);

m.copyProperties(news);

manager.updateNews(m);

**return** ajaxForwardSuccess(getText("msg.operation.success"));

}

**public** String publish() {

manager.publishNews(newsId);

**return** ajaxForwardSuccess(getText("msg.operation.success"));

}

**public** String disable() {

manager.disableNews(newsId);

**return** ajaxForwardSuccess(getText("msg.operation.success"));

}

**public** String delete() {

manager.removeNews(newsId);

**return** ajaxForwardSuccess(getText("msg.operation.success"));

}

}

// BaseAction 代码片段

**public** **class** BaseAction **extends** ActionSupport {

**private** **int** statusCode = 200;

**private** String message = **null**;

**private** String forwardUrl = **null**;

**private** String ajaxForward(**int** statusCode) {

**this**.statusCode = statusCode;

**return** *OPERATION\_DONE*;

}

**protected** String ajaxForwardSuccess(String message) {

**this**.message = message;

**return** ajaxForward(200);

}

**protected** String ajaxForwardError(String message) {

**this**.message = message;

**return** ajaxForward(300);

}

**public** **int** getStatusCode() {

**return** statusCode;

}

**public** String getMessage() {

**return** message;

}

**public** String getForwardUrl() {

**return** forwardUrl;

}

**public** **void** setForwardUrl(String forwardUrl) {

**this**.forwardUrl = forwardUrl;

}

}

// 工具类判断是否ajax请求

**public** **class** ServerInfo {

**public** **static** **boolean** isAjax(HttpServletRequest request) {

**if** (request != **null**

&& "XMLHttpRequest".equalsIgnoreCase(request

.getHeader("X-Requested-With")))

**return** **true**;

**return** **false**;

}

}

# JUI js库介绍

## JUI框架初始化

在<head> 引入必要的js库

JUI框架初始化会自动读取jui.frag.xml中的页面组件碎片代码.

jui.frag.xml中定义了一些jui组件碎片和提示信息, 需要初始化到JUI环境中.

注意jui.frag.xml路径问题.

假设jui.frag.xml放在根目录下, 在页面js脚本中调用JUI.init("jui.frag.xml")

<script type="text/javascript">

$(**function**(){

JUI.init("jui.frag.xml", {

loginUrl:"login.html",

callback:**function**(){

initEnv();

$("#themeList").theme({themeBase:"resource/css/theme-",defaultTheme:"toptry"});

}

});

});

</script>

## jui.core.js

JUI核心库主要功能是JUI初始化, Javascript String增加了一些扩展方法.

定义jui ajax 加载扩展loadUrl(url, data, callback)和ajaxUrl(options)

## jui.ui.js

页面效果初始化，html扩展绑定js效果

## jui.ajax.js

ajax表单提交封装

## jui.alertMsg.js

* 确认提示框

alertMsg.confirm("您修改的资料未保存，请选择保存或取消！", {

okCall: function(){

$.post(url, {accountId: accountId}, JUI.ajaxDone, "json");

}

});

* 成功提示框 alertMsg.correct('您的数据提交成功！')
* 错误提示框 alertMsg.error('您提交的数据有误，请检查后重新提交！')
* 警告提示框 alertMsg.warn('您提交的数据有误，请检查后重新提交！')
* 信息提示框 alertMsg.info('您提交的数据有误，请检查后重新提交！')

## jui.jDialog.js

弹出层组件库

## jui.accordion.js

滑动面板组件库

## jui.barDrag.js

JUI左边的活动面板

## jui.navTab.js

导航tab组件库

navTab API

打开一个标签页 navTab.openTab(tabid, title, url, [data])

重新载入标签页，如果无tabid参数，就载入当前标签页navTab.reload(url, data, [tabid])

获取当前标签页容器 navTab. getCurrentPanel()

关闭一个标签页navTab.closeTab(tabid)

关闭当前标签页navTab.closeCurrentTab()

关闭全部标签页navTab.closeAllTab()

## jui.scrollCenter.js

页面容器自动居中组件

## jui.cssTable.js

简单table组件库

## jui.tree.js

tree组件库

## jui.theme.js

切换界面主题风格

## jui.validate.method.js

这是jquery.validate.js表单验证扩展方法

## jui.validate.zh.js

表单验证本地化

## jui.contextmenu.js

自定义鼠标右键菜单, 先在jui.frag.xml加入菜单项定义，下面是navTab和dialog两个组件的菜单项定义：

<\_PAGE\_ id="navTabCM"><![CDATA[

<ul id="navTabCM">

<li rel="closeCurrent">关闭标签页</li>

<li rel="closeOther">关闭其它标签页</li>

<li rel="closeAll">关闭全部标签页</li>

</ul>

]]></\_PAGE\_>

<\_PAGE\_ id="dialogCM"><![CDATA[

<ul id="dialogCM">

<li rel="closeCurrent">关闭弹出窗口</li>

<li rel="closeOther">关闭其它弹出窗口</li>

<li rel="closeAll">关闭全部弹出窗口</li>

</ul>

]]></\_PAGE\_>

示例：

$("body").contextMenu('navTabCM', {

bindings:{

closeCurrent:**function**(t){

// TODO

},

closeOther:**function**(t){

// TODO

},

closeAll:**function**(t){

// TODO

}

},

ctrSub:**function**(t,m){

**var** mCur = m.find("[rel='closeCurrent']");

**var** mOther = m.find("[rel='closeOther']");

**var** mAll = m.find("[rel='closeAll']");

// TODO

}

});

## jui.database.js

suggest自动完成的提示框组件

lookup查找带回组件

itemDetail 主从结构组件

selectedTodo批量选择操作组件(批量删除, 批量审核…)

## jui.datepicker.js

JUI日历控件库

## jui.combox.js

combox下拉菜单组件，支持多级联动

## jui.checkbox.js

checkbox全选、反选。（demo 🡪 表单组件 🡪多选框/单选框）

## jui.uitl.date.js

日期处理工具类

## jui. regional.zh.js

JUI本地化

## jui.validate.method.js

jquery.validate.js 扩展

class:

required <input type="text" name="name" class=”required”/>

email <input type="text" name="name" class=”email”/>

url <input type="text" name="name" class=”url”/>

date <input type="text" name="name" class=”date”/>

number <input type="text" name="name" class=”number”/>

digits <input type="text" name="name" class=”digits”/>

creditcard <input type="text" name="name" class=”creditcard”/>

attribute:

equalTo：selector <input type="text" name="name" equalTo="#name"/>

maxlength: <input type="text" name="name" maxlength="20"/>

minlength: <input type="text" name="name" minlength="2"/>

实现长度范围时是同时写上minlength 与 maxlength，此时的提示将是rangelength的提示。

max: <input type="text" name="name" max="2"/>

min: <input type="text" name="name" min="2"/>

实现值范围时是同时写上min与 max，此时提示将是range的提示。

提示内容更改在文件jui.regional.zh.js。

参考文档 http://docs.jquery.com/Plugins/Validation

# Javascript混淆和压缩

Javascript压缩后，可以把300K的js压缩到140K左右．不建议混淆。

**JUI压缩方法：**

1. Nodejs中安装uglifyjs
2. 在JUI源码目录执行uglifyjs core.js util.date.js datepicker.js drag.js tree.js accordion.js ui.js navTab.js tab.js dialog.js table.js taskBar.js ajax.js database.js input.js file.js -c -o ../jui.min.js

## JUI如何中使用打包的js

在index.html中移除全部jui.\*.js，引入下面2个js库

<script src="js/jui.min.js"></script>

<script src="js/lang/regional.zh.js"></script>

**使用时引入以下js：**

plugins/jquery/jquery.min.js 【必须】jQuery库

plugins/jquery/jquery.cookie.js 【可选】js操作cookie, 目前用于记住用户选择的theme风格

plugins/jquery/jquery.validate.js 【必须】表单验证

js/jui.min.js 【必须】 JUI框架js压缩包

js/lang/regional.zh.js 【可选】 用于国际化 ,其中zh还可以换成en,ja,zh\_tw以对应英文、日语、繁体中文

# 常见问题及解决

### JUI中如何整合第三方jQuery插件

jQuery插件一般是$(document).ready()中初始化

$(document).ready(function(){

// 文档就绪,初始化jQuery插件

});

// 或者或缩写形式

$(function(){

// 文档就绪,初始化jQuery插件

});

因为JUI RIA是富客户端思路，第一次打开时加载界面到浏览器端，之后和服务器的交互是存数据交互，不占用界面相关的网络流量。

也就是说，只需要在一个完整的页面（通常是起始页，如index.aspx/index.php/index.jsp等），只有这个页面包含完整的html结构(<head><body>)，<head>中引入全部css、js 。其它的页面只需要页面碎片，就是<body></body>中的部分。

因为ajax加载基本原理是：ajax加载一段html代码片段放到当前页面的某个容器中（通常是一个div）。当然也可以是xml结构、json结构，只是在插入到当前页面之前也要转化成html代码片段。如果你ajax加载一个完整的页面(就是包括<head><body>标签的)，插入的当前document就有问题了，因为一个document不可能有多个<head><body>标签。

理解了富客户端思路你也就明白了为什么JUI框架中整合第三方jQuery插件不能在<head>中通过$(document).ready()初始化。

JUI初始化ajax加载的页面中的第三方jQuery组件：

* 一般插件初始化jui.ui.js 中的initUI 里面处理，initUI()方法JUI框架封装的$.fn.loadUrl()自动调用。如果是jQuery原生load方法需要手动调用$.fn.initUI()插件。主意initUI()中初始化是要注意作用域，里面有一个$p代表当前ajax加载的容器，只要初始化当前容器新加载的内容就可以了。



* 如果一些特殊的ajax交互，自己写回调函数处理

## Error loading XML document: jui.frag.xml

直接用IE打开index.html弹出一个对话框：Error loading XML document: jui.frag.xml

原因：jui.frag.xml是一个核心文件，需要加载才可以正常使用。IE ajax laod本地文件有限制, 是ie安全级别的问题, 不是框架的问题.

解决方法：放到apache或iis下就可以了. 如果不想安装apache或iis用firefox打开就正常了。

## IIS不能用Ajax访问\*.html后缀的页面

Ajax访问\*.html后缀的页面在Apache很好的工作,但在IIS不行,IIS下firebug调试报错ajax 405 Method Not Allowed。

Http 405原因是IIS不允许ajax post方式访问\*.html后缀的页。

IIS在使用Ajax post方式请求页面时,一定要动态网页后缀的或者用改用get方式！这是IIS的问题，不是框架bug。

也可以试试修改IIS配置，添加扩展名（.html）的脚本映射。

## 多个navTab页面或dialog页面ID冲突，解决方法

如果多个navTab页面或dialog页面有相同的ID，假设这个ID为：xxxId

还需要尽量减少id使用，优先使用class等

$("#xxxId", navTab.getCurrentPanel()); // 获取当前navTab中的xxxId

$("#xxxId", $.pdialog.getCurrent()); // 获取当前dialog中的xxxId

## 如何自定义JUI分页参数参数

pagerForm默认使用的当前页参数是pageNum, 每页显示条数numPerPage, 查询排序字段名orderField, 升序降序orderDirection, 更改其它参数需要设置JUI.init(pageFrag, options) 的options[“pageInfo”]:

<form id="pagerForm" action="xxx" method="post">

      <input type="hidden" name="pageNum" value="1" />/><!--【必须】value=1可以写死-->

      <input type="hidden" name="numPerPage" value="20" /><!--【可选】每页显示多少条-->

      <input type="hidden" name="orderField" value="xxx" /><!--【可选】查询排序字段-->

      <input type="hidden" name="orderDirection" value="asc|desc" />/><!--【可选】升序|降序-->

</form>

<script type="text/javascript">

$(**function**(){

JUI.init("jui.frag.xml", {

loginUrl:"login.html", // 跳到登录页面

statusCode:{ok:200, error:300, timeout:301}, //【可选】

pageInfo:{pageNum:"pageNum", numPerPage:"numPerPage", orderField:"orderField", orderDirection:"orderDirection"}, //【可选,这里自定义分页参数】

debug:false, // 调试模式 【true|false】

callback:**function**(){

initEnv();

$("#themeList").theme({themeBase:"resource/css/theme-",defaultTheme:"toptry"});

}

});

});

</script>

## 如何关闭loading

jui的ajax方法每次调用都会出现读取数据的loading，怎么修改可选的？我自己写了一个局部更新的ajax函数，结果loading太烦人 怎么关掉好？

jui.ui.js中注册了ajax全局事件：

**var** ajaxbg = $("#background,#progressBar");

ajaxbg.hide();

$(document).ajaxStart(**function**(){

ajaxbg.show();

}).ajaxStop(**function**(){

ajaxbg.hide();

});

$.ajax() 有个参数global (Boolean) : (默认: true) 是否触发全局 AJAX 事件.设置为 false 将不会触发全局 AJAX 事件，如 ajaxStart 或 ajaxStop 可用于控制不同的 Ajax 事件。

## JUI局部刷新怎样做?

API调用方式：

$("#xxxId").loadUrl(url,data, callback);

html扩展链接方式：

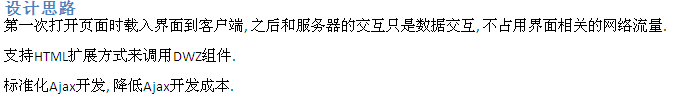
<a href="url" target="ajax" rel="xxxId"></a>

## 补充说明和常见问题(xiaosuiba)

作者：xiaosuiba ([xiaosuiba@qq.com](mailto:xiaosuiba@qq.com)) 2011年6月

**1、我如何在项目中使用jui?**

手册中有如下说明：



也就是说，只需要在一个页面（通常是起始页，如index.aspx/index.php）包含框架，这里的框架是指demo中index.html页面的所有元素（<div class=”page”可自定义），完整的html结构。其它的页面只需要页面碎片，就是<body></body>中的部分。

**2、我如何解析json数据来重绘表格/页面？**

很多人不明白jui的工作方式，认为jui的navTab的页面看起来是ajax方式解析的，那就要从服务器传回json再手动解析。jui事实上是这样工作得，只是他传回的不是一部分数据，而是整个页面，然后通过JSON.eval()加载到navTab上，这个过程对使用者是透明的，也即你不需要关心页面的数据处理，以前怎么写的页面，现在还是怎么写页面。jui会将普通请求转换为ajax方式（前提是正确使用jui提供的接口）。

**3、如果不是传输数据，jui的json是用于哪里？**

jui的服务器端响应上提到一个服务器端响应json，很多初学者问这个json如何传递数据，用于自己拼接页面等。如第3点所说，jui的页面是不需要手动处理ajax的，这个json结构是对ajax表单提交、ajax post链接(ajaxTodo)状态的响应，而不涉及具体的页面数据。

**4、提交表单或者ajax post链接后，如何刷新本navTab？**

一定要记住在返回的json中加上要刷新的navTabId。

**5、如何在ajax连接扩展中使用回调函数？**

手册上没有写，其实从1.2RC1开始，ajax link就有了callback属性，用于指定回调函数，如<a target=”ajaxTodo” callback=”MyOwnFunction”。

**6、如何使用css table的排序功能？**

Table的排序功能是手册中没有提到的，其实jui的排序功能相当强大，这里我简单介绍一下流程：

1. 给要排序的表格<table中加上asc=”asc” desc=”desc”，指定排序别名。
2. 给要排序的表格表头th加上orderField=”fieldName”属性，这样点击该表头才能出发提交事件。Th的class=”asc”/class=”desc”会分别显示向上和向下的箭头，这个不是只显示这么简单，往下看。
3. 在pageForm加上orderField和orderDirection，点击排序后提交的依然是pagerForm，orderField会绑定点击的th的orderField，而orderDirection则会反向绑定th的class，这是jui智能的地方，也就是你不用手动记住状态来反向（感谢细心的作者），class=”asc”就会提交orderDirection = desc。注意每次要将orderDirection绑定回th的class。

**7、如何使用css table的分页功能？**

分页功能是大家用得比较多，也是不容易理解的一点。这里我凭着自己的理解给初学者讲讲。

手册上讲得很清楚，jui不是客户端分页，而是服务器端分页，结合本文第3点可以知道jui的分页就是每次将分页数据提交回后台，后台生成分页数据显示到页面上。需要注意的是以下几点：

1. 分页只需要pagerForm与pagination两个jui组件，点击分页提交的是pagerForm。
2. pagerForm用于带查询的分页数据的缓存，说缓存是因为这里的参数都需要自己手动从后台读取绑定（pageNum除外）。
3. Pagination可以理解为一个页码生成器，他需要totalCount="200" numPerPage="20" pageNumShown="10" currentPage="1"几个参数来显示，每次后台需要绑定这几个参数，jui不会帮你做什么事情，也就是你想他显示第几页就是第几页。
4. 点击分页，jui将pagination的currentPage绑定到pagerForm的pagerNum，然后提交pagerForm到后台。
5. 初学者可以做这件事情来帮助理解：页面只放pagerForm和一个div和pagination，div每次显示当前的pageNum和pagination，点击分页来好好体验以下这个过程，这对第三点的理解也有帮助。

**8、最后谈谈我的看法（xiaosuiba）**

jui作为一个开源的ajax前端框架，为广大的web开发者提供了极大的方便，这点对初学者可能还没有这么深刻，不过一些老程序员可是感动得一塌糊涂。可以看出，这个框架倾注了作者极大的心血，而大家的热情就是对这种奉献精神的最好回报。本人接触jui不过区区1个月，但是一直坚持和大家一起讨论问题，就是希望有越来越多的人能够使用jui，这样才能使其具有长久的生命力。

对于web开发老手来说，jui很容易上手，对于新手，我想提几条建议：

1. 先要有基础的web知识，手上常备js手册和jui手册。
2. 遇到问题先试着从手册和demo里寻找解决办法，尽量不要问手册中已经存在的问题，相信没有人会喜欢一遍一遍回答诸如navTab是什么，navTabId是什么之类的问题。
3. 相信jui能够用于项目，现在已经有人成功了。所以放手去做吧。
4. 尽量使用最新版本，作为一个开源项目，jui更新是频繁的，通过这种更新来消除bug，同时引入新的特性。所以请升级你的版本到最新稳定版本或RC版本。