Microsoft

2012

ExtJs4 笔记

[键入公司地址]

目录

[第一章 ExtJs大比拼JQuery：Dom文档操作 3](#_Toc339984672)

[一、选择器 4](#_Toc339984673)

[二、属性 8](#_Toc339984674)

[四、筛选 10](#_Toc339984675)

[五、文档处理 11](#_Toc339984676)

[六、CSS 13](#_Toc339984677)

[七、事件 14](#_Toc339984678)

[第二章 ExtJs对js基本语法扩展支持 16](#_Toc339984679)

[一、动态引用加载 16](#_Toc339984680)

[二、对类的封装 17](#_Toc339984681)

[三、基本数据类型 19](#_Toc339984682)

[四、函数执行时间控制 20](#_Toc339984683)

[五、键盘事件侦听 21](#_Toc339984684)

[第三章 Ext.Ajax 对ajax的支持 23](#_Toc339984685)

[一、ExtJs中的Ajax：Ext.Ajax.request 23](#_Toc339984686)

[二、元素对象Ajax式更新：Ext.Updater,[已弃用] 29](#_Toc339984687)

[第四章 Ext.XTemplate 模板 31](#_Toc339984688)

[一、使用标签tpl和操作符for 33](#_Toc339984689)

[二、在子模板的范围内访问父元素对象 34](#_Toc339984690)

[三、数组元素索引和简单运算支持 34](#_Toc339984691)

[四、自动渲染单根数组 35](#_Toc339984692)

[五、条件逻辑判断 36](#_Toc339984693)

[六、即时执行任意的代码 36](#_Toc339984694)

[七、模板成员函数 37](#_Toc339984695)

[lipan 38](#_Toc339984696)

[第五章 Ext.Button 按钮 38](#_Toc339984697)

[预览 39](#_Toc339984698)

[一、基本按钮，三种方式实现按钮事件 40](#_Toc339984699)

[二、带图标菜单 41](#_Toc339984700)

[三、带分割线的按钮 43](#_Toc339984701)

[四、菜单式按钮 44](#_Toc339984702)

[四、按钮组合 45](#_Toc339984703)

[第六章 Ext.MessageBox 消息对话框 46](#_Toc339984704)

[一、警告对话框和确认对话框 46](#_Toc339984705)

[二、输入对话框 47](#_Toc339984706)

[三、自定义DIY对话框 48](#_Toc339984707)

[四、加载进度条对话框 49](#_Toc339984708)

[五、等待进度条对话框 50](#_Toc339984709)

[六、基本的等待对话框 51](#_Toc339984710)

[第七章 Ext.tip.ToolTip 提示 52](#_Toc339984711)

[一、基本提示 Ext.tip.ToolTip 52](#_Toc339984712)

[二、快速提示 Ext.tip.QuickTip 58](#_Toc339984713)

[三、在extjs控件上使用提示 59](#_Toc339984714)

[第八章 Ext.slider 滚轴控件、 Ext.ProgressBar 进度条控件、 Ext.Editor 编辑控件 61](#_Toc339984715)

[一、滚轴控件 Ext.slider 61](#_Toc339984716)

[二、进度条控件 Ext.ProgressBar 64](#_Toc339984717)

[三、编辑控件 Ext.Editor 68](#_Toc339984718)

[第九章 Ext.Panel 面板控件、 Ext.window.Window 窗口控件、 Ext.container.Viewport 布局控件 72](#_Toc339984719)

[一、面板控件 Ext.Panel 72](#_Toc339984720)

[二、窗口控件 Ext.window.Window 74](#_Toc339984721)

[三、布局控件 Ext.container.Viewport 77](#_Toc339984722)

[第十章 Ext.tab.Panel 选项卡 79](#_Toc339984723)

[一、基本选项卡 79](#_Toc339984724)

[二、操作选项卡 82](#_Toc339984725)

[三、选项卡按钮在下方 84](#_Toc339984726)

[四、可拖动的选项卡 85](#_Toc339984727)

[五、过多选项卡的菜单式展示 87](#_Toc339984728)

[六、选项卡的右键菜单 89](#_Toc339984729)

[七、分组式选项卡 91](#_Toc339984730)

[第十一章 Ext.ListView、Ext.view.View 数据视图 93](#_Toc339984731)

[一、Ext.ListView 93](#_Toc339984732)

[二、Ext.view.View 98](#_Toc339984733)

[第十二章 Ext.toolbar.Toolbar 工具栏、Ext.toolbar.Paging 分页栏、Ext.ux.statusbar.StatusBar 状态栏 103](#_Toc339984734)

[一、Ext.toolbar.Toolbar 103](#_Toc339984735)

[二、Ext.toolbar.Paging 115](#_Toc339984736)

[三、Ext.ux.statusbar.StatusBar 122](#_Toc339984737)

[第十三章 Ext.menu.Menu 菜单、Ext.draw.Component 绘图、Ext.resizer.Resizer 大小变更 128](#_Toc339984738)

[一、Ext.menu.Menu 菜单 128](#_Toc339984739)

[二、Ext.draw.Component 绘图 134](#_Toc339984740)

[三、Ext.resizer.Resizer 大小变更 137](#_Toc339984741)

[第十四章 layout 布局 138](#_Toc339984742)

[一、absolute 138](#_Toc339984743)

[二、accordion 140](#_Toc339984744)

[三、anchor 142](#_Toc339984745)

[四、border 144](#_Toc339984746)

[五、card 146](#_Toc339984747)

[六、column 148](#_Toc339984748)

[七、fit 150](#_Toc339984749)

[八、table 151](#_Toc339984750)

[九、vbox 152](#_Toc339984751)

[十、hbox 154](#_Toc339984752)

|  |
| --- |
| [第一章 ExtJs大比拼JQuery：Dom文档操作](http://www.cnblogs.com/lipan/archive/2011/12/07/2269815.html) 现在主流的JS框架要数ExtJs和JQuery应用的比较广泛。JQuery属于轻量级的，一般做网站应用比较常见，可见块头小的优势。ExtJs比较庞大，它除了对基本的JS语法和HTML DOM操作方式的封装之外，还提供了一套强大的UI库。在企业级B/S解决方案应用上独占优势。就单说它在原生JS的封装，DOM操作方式封装方面也足以藐视JQuery。下面我就通过对比API，体现两种框架的异曲同工之处。我们已JQuery API为主线，来看看ExtJs是否有替代的方案。  注意一点：ExtJs4.0相对上一版本改动较大，本系列文章暂时只对ExtJs4.0及以上版本提供支持。  **一、选择器**  **1.上下文选择器对比**  JS对HTML节点的操作比较频繁。所以要经常定位和查询DOM元素。我们分别看看两种框架的实现  获取Id="div1"的DOM元素:  [JQuery]  $("#div1");  [ExtJs]  Ext.get("div1");  其实Ext.get("div1")与$("#div1")还是有区别的，前者只会获取匹配的第一个元素，后者是匹配的元素集合。ExtJs的等价用法如下：  [ExtJs]  Ext.select("#div1");  下面是一个完整的比较代码：  [ExtJs]  //Ext.Element类 :是ExtJs对dom对象的封装  //Ext.CompositeElement类：是ExtJs对dom对象集合的封装  //查询Id=div1的元素，返回第一个匹配元素的Ext.Element类型  //该方法只能查Id，不支持选择器  Ext.get("div1");  //功能同上  Ext.fly("div1");  //查询Id=div1的元素的dom对象  Ext.get("div1").dom;  //或者  Ext.getDom("div1");  //获取body元素的Ext.Element类型  Ext.getBody();  //返回当前HTML文档的Ext.Element类型  Ext.getDoc();  //查询Id=div1的元素,返回满足条件的集合Ext.CompositeElement类型  Ext.select("#div1");  //查询属性title=d1的div,返回满足条件的集合Ext.CompositeElement类型  Ext.select("div[title='d1']");  //查询属性title=d1的div,返回满足条件的dom节点集合  Ext.query("div[title='d1']");  **2.选择器语法简介**  JQuery选择器支持CSS3选择符，ExtJs同样也提供支持，除此之外还支持基本的XPath语法。下面分别讲解：  **1)CSS3选择符**  下面列举的命令是单个形式，都可以无限组合使用。  **元素选择符：**  任意元素   * **E** 一个标签为 E 的元素 * **E F** 所有 E 元素的分支元素中含有标签为 F 的元素 * **E > F** 或 **E/F** 所有 E 元素的直系子元素中含有标签为 F 的元素 * **E + F** 所有标签为 F 并紧随着标签为 E 的元素之后的元素 * **E ~ F** 所有标签为 F 并与标签为 E 的元素是侧边的元素   **属性选择符：**  @ 与引号的使用是可选的。例如：div[@foo='bar'] 也是一个有效的属性选择符。   * **E[foo]** 拥有一个名为 “foo” 的属性 * **E[foo=bar]** 拥有一个名为 “foo” 且值为 “bar” 的属性 * **E[foo^=bar]** 拥有一个名为 “foo” 且值以 “bar” 开头的属性 * **E[foo$=bar]** 拥有一个名为 “foo” 且值以 “bar” 结尾的属性 =bar] 拥有一个名为 “foo” 且值包含字串 “bar” 的属性 * **E[foo%=2]** 拥有一个名为 “foo” 且值能够被2整除的属性 * **E[foo!=bar]** 拥有一个名为 “foo” 且值不为 “bar” 的属性   **伪类：**   * **E:first-child** E 元素为其父元素的第一个子元素 * **E:last-child** E 元素为其父元素的最后一个子元素 * **E:nth-child(n)** E 元素为其父元素的第 n 个子元素（由1开始的个数） * **E:nth-child(odd)** E 元素为其父元素的奇数个数的子元素 * **E:nth-child(even)** E 元素为其父元素的偶数个数的子元素 * **E:only-child** E 元素为其父元素的唯一子元素 * **E:checked** E 元素为拥有一个名为“checked”且值为“true”的元素（例如：单选框或复选框） * **E:first** 结果集中第一个 E 元素 * **E:last** 结果集中最后一个 E 元素 * **E:nth(n)** 结果集中第 n 个 E 元素（由1开始的个数） * **E:odd** :nth-child(odd) 的简写 * **E:even** :nth-child(even) 的简写 * **E:contains(foo)** E 元素的 innerHTML 属性中包含“foo”字串 * **E:nodeValue(foo)** E 元素包含一个 textNode 节点且 nodeValue 等于“foo” * **E:not(S)** 一个与简单选择符 S 不匹配的 E 元素 * **E:has(S)** 一个包含与简单选择符 S 相匹配的分支元素的 E 元素 * **E:next(S)** 下一个侧边元素为与简单选择符 S 相匹配的 E 元素 * **E:prev(S)** 上一个侧边元素为与简单选择符 S 相匹配的 E 元素   **CSS 值选择符：**   * **E{display=none}** css 的“display”属性等于“none” * **E{display^=none}** css 的“display”属性以“none”开始 * **E{display$=none}** css 的“display”属性以“none”结尾 =none} css 的“display”属性包含字串“none” * **E{display%=2}** css 的“display”属性能够被2整除 * **E{display!=none}** css 的“display”属性不等于“none”   **2)XPath语法**  下面通过几个例子来说明：  /html/body/div/div :从根目录开始找，找到正文的第二层全部DIV。  div/div :在全文匹配DIV元素，并获取包含子DIV的全部子DIV集合。  **3.Ext.get和Ext.fly的区别：**  通俗点讲，他们的作用都是一样的，都是获取元素。但是前者每次调用都会生成一个Ext.Element对象，开辟新的内存空间，而后者共享了一个公用的内存空间，每次调用都会覆盖前一次的信息。由于Ext.Element 比较庞大，后者的好处是可以节省资源。如果你获取的Ext.Element 不需要长期保持重复调用，用后者较为合理。下面通过一个例子来体现他们的区别：  //这样更新的是div1  var div1 = Ext.get("div1");  Ext.get("div2");  div1.update("我想更新div1");  //将get替换成fly后....  //更新的却是div2  var div1 = Ext.fly("div1");  Ext.fly("div2");  div1.update("我想更新div1");  我们发现，再次调用Ext.fly后，更新的是DIV2，而再次调用Ext.get不会影响更新的元素。  **二、属性**  注意：CSS类操作的几个方法相对于早期版本有变化之前是：e.addClass("c2")  [JQuery]  var e = $("div[title=t1]");  //返回第一个匹配元素的title属性  alert(e.attr("title"));  //设置第一个匹配元素的title属性  e.attr("title", "newTitle");  //移除第一个匹配元素的title属性  e.removeAttr("title");  //CSS类  //给第一个匹配元素添加c2样式  e.addClass("c2");  //移除  e.removeClass("c1");  //轮回  e.toggleClass("c2");  //检查c2样式是否存在  e.hasClass("c2");  //Html  //获取Html  e.html();  //更新Html  e.html("<b>更新后的Html</b>");  //值  e.val();  e.val(150);  [ExtJs]  var e = Ext.select("div[title=t1]");  //属性  //返回第一个匹配元素的title属性  alert(e.first().getAttribute("title"));  //设置第一个匹配元素的title属性  e.first().set({ "title": "newTitle" });  //CSS类  //给第一个匹配元素添加c2样式  e.addCls("c2");  //移除  e.removeCls("c1");  //轮回  e.toggleCls("c2");  //检查c2样式是否存在  e.hasCls("c2");  //Html  //获取Html  e.first().dom.innerHTML;  //更新Html  e.first().update("<b>更新后的Html</b>");  //值  e = Ext.get("text1");  e.getValue();  e.set({ value: 150 });  **四、筛选**  [JQuery]  var e = $(".c1");  //过滤  //获取匹配的第二个元素  e.eq(1);  //再次筛选，属性title=t1的div  e.filter("div[title=t1]");  //父节点Id=div1时返回true  e.parent().is("#div1");  //查找  //获取集合中第一个，最后一个：  e.first();  e.last();  //前一个，后一个：  e.prev();  e.next();  //子节点第一个，最后一个：  var e2 = $("#div1");  e2.children().first();  e2.children().last();  [ExtJs]  var e = Ext.select(".c1");  //过滤  //获取匹配的第二个元素  e.item(1);  //再次筛选，属性title=t1的div,集合中不满足条件的元素自动移去  e.filter("div[title=t1]");  e = Ext.select(".c1");  //父节点Id=div1时返回true  e.first().parent().is("#div1");  //查找  //获取集合中第一个，最后一个：  e.first();  e.last();  //单个元素的前一个，匹配选择器的后一个:  e.item(1).next("div[title=t2]");  e.item(1).prev();  var e2 = Ext.get("div1");  //获取子节点的第一个，最后一个：  e2.first();  e2.last();  **五、文档处理**  [JQuery]  //插入  var e = $("div .c1");  //e元素结束之前插入  e.first().append("<b>新内容</b>");  //e元素开始之后插入  e.first().prepend("<b>新内容</b>");  //e元素开始之前插入  e.first().before("<b>新内容</b>");  //e元素结束之后插入  e.first().after("<b>新内容</b>");  //包裹  e.eq(2).wrap("<p></p>");  e.eq(2).unwrap();  //替换  e.eq(2).replaceWith("<b>被替换的</b>");  //删除  e.eq(3).remove();  //清空标签内的html  e.eq(0).empty();  [ExtJs]  var e = Ext.select("div .c1");  //插入  //分别在相对于文档e的四个位置插入html：  e.first().insertHtml("afterBegin", "<b>新内容</b>");  e.first().insertHtml("beforeEnd", "<b>新内容</b>");  e.first().insertHtml("beforeBegin", "<b>新内容</b>");  e.first().insertHtml("afterEnd", "<b>新内容</b>");  //包裹  e.wrap({  tag: 'p'  });  //用text1替换集合第五个元素  e.replaceElement(4, "text1", true);  //替换用当前元素去替换掉div2标签  e.first().replace("div2", true);  //删除  e.first().remove();  //移除集合中第三个元素  e.removeElement(2, true);  **六、CSS**  [JQuery]  var e = $("div1");  //设置样式  e.css("width", "550px");  e.css("position", "absolute");  //设置高度  e.height(100);  //获取宽度  e.width();  [ExtJs]  var e = Ext.get("div1");  //设置样式  e.setStyle("width", "550px");  e.applyStyles({  height: "500px",  color: "red",  position: "absolute"  });  //设置高度，带动画效果  e.setHeight(100, true);  //设置定位  e.setLeft("50px");  e.setTop("10px");  e.setLeftTop("100px", "50px");  //设置尺寸  e.setSize("100px", "200px");  //设置xy坐标  e.setXY([10, 10]);  //获取宽度  e.getWidth();  //获取坐标  e.getXY();  **七、事件**  [JQuery]  var e = $("#div1");  //事件绑定  //给元素绑定click事件：  var clickhandler = function () {  alert("click事件被触发！");  };  e.bind("click", clickhandler);  //模拟click事件：  e.trigger("click");  //解除绑定click事件  e.unbind("click", clickhandler);  //事件切换  e.hover(function () {  e.css("background-color", "Red");  }, function () {  e.css("background-color", "Aqua");  });  [ExtJs]  var e = Ext.get("div1");  //事件绑定  //给元素绑定click事件：  var clickhandler = function () {  Ext.Msg.alert("消息", "click事件被触发！");  };  e.on("click", clickhandler);  //解除绑定click事件  e.un("click", clickhandler);  //事件切换  e.hover(function () {  e.setStyle("background-color", "Red");  }, function () {  e.setStyle("background-color", "Aqua");  }); |
| |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [第二章 ExtJs对js基本语法扩展支持](http://www.cnblogs.com/lipan/archive/2011/12/08/2278507.html) 本篇主要介绍一下ExtJs对JS基本语法的扩展支持，包括动态加载、类的封装等。  **一、动态引用加载**  ExtJs有庞大的类型库，很多类可能在当前的页面根本不会用到，我们可以引入动态加载的概念来即用即取。这些代码都要写在Ext.onReady外面。  **1.动态引用外部Js**  [Js]  //加载配置可用  Ext.Loader.setConfig({ enabled: true });  //动态引用“../ux/”目录下所有Js文件类，映射到对应命名空间  Ext.Loader.setPath('Ext.ux', '../ux/');  **2.动态加载类**  [Js]  //加载单个类  Ext.require('Ext.window.Window');  //加载多个  Ext.require([  'Ext.grid.\*',  'Ext.data.\*',  'Ext.util.\*',  'Ext.grid.PagingScroller'  ]);  //加载所有类，除了“Ext.data.\*”之外  Ext.exclude('Ext.data.\*').require('\*');  **二、对类的封装**  Js本身是面向对象的语言，但是语法层面上对类的支持不够完善，ExtJs对此作了一系列的封装，下面看看类的定义、字段、构造函数、方法、静态字段，方法的实现方式，还用类的继承的用法。  [Js]  Ext.onReady(function () {  Ext.define("My.test.Animal", {  height: 0,  weight: 0  });  Ext.define("My.test.Person", {  //普通子段  name: "",  //属性  config: {  age: 0,  father: {  name: "",  age: 0  }  },  //构造方法  constructor: function (name, height) {  this.self.count++;  if (name) this.name = name;  if (height) this.height = height;  },  //继承  extend: "My.test.Animal",  //实例方法  Say: function () {  alert("你好，我是：" + this.name + ",我今年" + this.age + "岁,我的身高是：" + this.height  　　　　　　　　　　+ "。我的爸爸是：" + this.father.name + "，他" + this.father.age + "岁。");  },  //静态子段，方法  statics: {  type: "高等动物",  count: 0,  getCount: function () {  return "当前共有" + this.count + "人";  }  }  });  function test() {  var p = Ext.create("My.test.Person", "李四", 178);  p.setAge(21);  p.setFather({  age: 48,  name: "李五"  });  p.Say();  Ext.create("My.test.Person");  alert(My.test.Person.getCount());  }  test();  });  **三、基本数据类型**  ExtJs支持数值型、字符串型、日期型、布尔型等基本数据类型，内容比较简单，下面演示基本的声明用法，以及类型转换。  [Js]  //定义一个日期类型的数据  var date1 = new Date("2011-11-12");  var date = new Date(2011, 11, 12, 12, 1, 12);  //转化为字符串型  alert(date.toLocaleDateString());  //转化为数值型  alert(Number(date));  //布尔型，假  var myFalse = new Boolean(false);  //真  var myBool = new Boolean(true);  //定义数值  var num = new Number(45.6);  alert(num);  **四、函数执行时间控制**  主要用两个方面，1.让某个函数等待一段时间后自动执行。2.然某个函数按照一定频率反复执行。  **1.函数等待执行**  实现一个功能，页面加载完毕后，等待3秒后弹出提示。  [Js]  var func1 = function (name1, name2) {  Ext.Msg.alert("3秒钟后自动执行", "你好，" + name1 + "、" + name2 + "！");  };  Ext.defer(func1, 3000, this, ["张三", "李四"]);  **1.函数按照一定频率反复执行**  让div1每隔一秒更新一次显示当前时间，10秒又自动停止更新：  [Js]  //周期执行  var i = 0;  var task = {  run: function () {  Ext.fly('div1').update(new Date().toLocaleTimeString());  if (i > 10) Ext.TaskManager.stop(task);  i++;  },  interval: 1000  }  Ext.TaskManager.start(task);  **五、键盘事件侦听**  **1..Ext.KeyMap**  通过Ext.KeyMap可以建立键盘和用户动作（Actions）之间的映射。下面看看例子，页面html沿用Ext.Updater部分。  [Js]  var f = function () {  alert("B被按下");  }  var map = new Ext.KeyMap(Ext.getDoc(), [  {  key: Ext.EventObject.B,  fn: f  }, {  key: "bc",  fn: function () { alert('b,c其中一个被按下'); }  },  {  key: "x",  ctrl: true,  shift: true,  alt: true,  fn: function () { alert('Control + shift +alt+ x组合键被按下.'); },  stopEvent: true  }, {  key: "a",  ctrl: true,  fn: function () { alert('Control+A全选事件被阻止，自定义事件执行！'); },  stopEvent: true  }  ]);  我们看到，在IE中测试，当我们按下ctrl+A键时，全选功能被屏蔽，支持了我们自定义的方法。  **2.Ext.KeyNav**  Ext.KeyNav主要是用来绑定方向键的，已支持的键：enter, left, right, up, down, tab, esc, pageUp, pageDown, del, home, end,现在通过它来实现鼠标控制层移动的功能：  [Js]  var div1 = Ext.get("div1");  var nav = new Ext.KeyNav(Ext.getDoc(), {  "left": function (e) {  div1.setXY([div1.getX() - 1, div1.getY()]);  },  "right": function (e) {  div1.setXY([div1.getX() + 1, div1.getY()]);  },  "up": function (e) {  div1.move("up",1);  },  "down": function (e) {  div1.moveTo(div1.getX(), div1.getY() + 1);  },  "enter": function (e) {  }  }); | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第三章 Ext.Ajax 对ajax的支持**](http://www.cnblogs.com/lipan/archive/2011/12/09/2272793.html) 本篇主要介绍一下ExtJs常用的几个对JS语法的扩展支持，包括Ajax封装，函数事件操作封装，还有扩展的常用函数等。Ajax服务端交互式操作是提交到.NET MVC。后续服务端交互都采用这一方式实现。  **一、ExtJs中的Ajax：Ext.Ajax.request**  通过客户端向服务端发送Ajax请求，可以“直接”调用MVC的action方法，并传递参数，action返回值可以是普通字符串，也可以是json对象。请求可以添加自定义头信息。看下面例子：  **1.异步请求，发送自定义请求头：**  html页面如下：  [html]  <h1>Ajax服务端请求</h1>  <div class="content" id="div1">  <div>一共发起了<span id="span1" style="color:Red">0</span>次请求。</div>  </div>  我们先通过一个按钮单击事件触发ajax请求，服务端返回string到客户端，异步请求时，为请求添加自定义头“userHeader”，并在服务端获取。如下是js代码：  [Js]  Ext.onReady(function () {  new Ext.Button({  renderTo: "div1",  text: "后台Ajax提交",  handler: function () {  Ext.Ajax.request({  url: 'Ajax\_Func1',  headers: {  'userHeader': 'userMsg'  },  params: { a: 10, b: 20 },  method: 'GET',  success: function (response, options) {  Ext.MessageBox.alert('成功', '从服务端获取结果: ' + response.responseText);  },  failure: function (response, options) {  Ext.MessageBox.alert('失败', '请求超时或网络故障,错误编号：' + response.status);  }  });  },  id: "bt1"  });  });  服务端MVC接受请求的action代码：  [C#]  public ContentResult Ajax\_Func1(int a, int b)  {  string userHeaderMsg = Convert.ToString(Request.Headers["userHeader"]);  return Content((a + b).ToString() + "，userHeader：" + userHeaderMsg);  }  然后我们在火狐中调试：自定义请求头已被添加。  http://pic002.cnblogs.com/images/2011/99414/2011120223083041.jpg  查看执行结果：  http://pic002.cnblogs.com/images/2011/99414/2011120223095658.jpg  **2.异步请求，返回json：**  如果要返回json到客户端，MVC需要使用JsonResult的acton，自动将C#对象转换为json格式。客户端代码如下：  [Js]  Ext.onReady(function () {  new Ext.Button({  renderTo: "div1",  text: "后台Ajax提交方式2返回JSON",  handler: function () {  Ext.Ajax.request({  url: 'Ajax\_Func2',  params: { n: 10 },  method: 'POST',  callback: function (options, success, response) {  if (success) {  var responseJson = Ext.JSON.decode(response.responseText);  Ext.Msg.alert("成功", options.params.n + "的阶乘是：<font color='red'>" + responseJson.n1 + "</font><br />"  　　　　　　　　　　　　　　　　　　　　　　+ options.params.n + "的累加是：<font color='red'>" + responseJson.n2 + "</font>");  } else {  Ext.Msg.confirm('失败',  '请求超时或网络故障,错误编号：[' + response.status + ']是否要重新发送？', function (btn) {  if (btn == 'yes') {  Ext.Ajax.request(options);  }  });  }  }  });  }  });  });  服务端代码：  [C#]  public JsonResult Ajax\_Func2(int n)  {  int n1 = 1;  int n2 = 0;  for (int i = 1; i <= n; i++)  {  n1 \*= i;  n2 += i;  }  var data = new  {  n1 = n1,  n2 = n2  };  return Json(data, JsonRequestBehavior.AllowGet);  }  查看执行结果,我们先看看火狐调试响应的结果，可以查看返回的json内容：  http://pic002.cnblogs.com/images/2011/99414/2011120223244986.jpg  界面执行结果：  http://pic002.cnblogs.com/images/2011/99414/2011120223251821.jpg  **3.异步文件上传**  ExtJs通过强大的内部封装，使ajax请求看起来可以直接提交二进制流数据。每当提交文件数据时，ExtJs会自动创建iframe，在iframe中提交。提交完成后又自动移去，一切显得天衣无缝。下面演示一个通过异步请求无刷新文件上传的例子：  [html]  <h1>Ajax文件上传</h1>  <div class="content" id="div2">  <form id="form1">  请选择文本文件:<input type="file" name="file" />  <input type="button" id="button1" value="上传" />  </form>  </div>  [Js]  //文件上传  Ext.get("button1").on("click", function () {  Ext.Ajax.request({  url: "Ajax\_FileUp",  isUpload: true,  form: "form1",  success: function (response) {  Ext.MessageBox.alert("上传成功，文本文件内容：", response.responseText);  }  });  });  [C#]  public ContentResult Ajax\_FileUp(HttpPostedFileBase file)  {  System.IO.StreamReader r = new System.IO.StreamReader(file.InputStream, System.Text.UTF8Encoding.Default);  var str = r.ReadToEnd();  return Content(str);  }  提交后我们发现，上传的文本文件内容已经被正确读取。  **4.异步请求事件**  当发起ajax请求之前，可以监听beforerequest事件，本例每当发起ajax事件时，都会把计算器+1：  [Js]  var ajaxCount = 0;  //每当Ajax请求发起时触发：  Ext.Ajax.on('beforerequest', function () { Ext.get("span1").update(++ajaxCount) }, this);  这样，每当ajax发起就可以被记录,可用作日志等。  **二、元素对象Ajax式更新：Ext.Updater,[已弃用]**  ExtJs为我们提供了异步方式更新dom元素内容的支持。这可以用在一些异步交互性比较强的地方，例如webgame等。我们来看看具体实现方式。  我们先在视图中写这样的html：  [html]  <div id="div1" title="d1" style="width:50px; height:50px; position:absolute; background-color:Green; top:100px">1</div>  <div id="div2">2</div>  <form id="form1"><input id="a" name="a" value="20" /><input id="b" name="b" value="50" /></form>  **1.单个元素的及时更新**  现在，我要通过c#在服务端计算a+b的值，然后将结果填到div1元素里面:  [Js]  Ext.get("div1").getUpdater().update({  url: "Ext/UpdaterFunc",  params: { a: 10, b: 3 }  });  [c#]  public ContentResult UpdaterFunc(int a,int b)  {  return Content("a+b=" + (a + b));  }  页面加载后，将自动更新div1。再看看另外一种写法：  [Js]  Ext.get("div1").load({  url: "Ext/UpdaterFunc",  scripts: true,  params: { a: 10, b: 47 },  text: "请等待..."  });  **2.通过提交表单更新**  [Js]  Ext.get("div1").getUpdater().formUpdate("form1", "Ext/UpdaterFunc");  效果一样，只不过提交的参数来源于表单，参数名为对应的name值。  **3.周期性更新**  [Js]  //周期性更新  var up = new Ext.Updater("div1");  up.startAutoRefresh(5, "Ext/UpdaterFunc", { a: 10, b: 47 });  这段代码运行后将每隔5秒钟更新一次div1，用于需要适时核对服务端和客户端状态的情形。 | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第四章 Ext.XTemplate 模板**](http://www.cnblogs.com/lipan/archive/2011/12/12/2274740.html) 本篇将涉及到ExtJs中一个重要的概念,模板。话说Razor很神奇，但是我个人不是很喜欢那种html混编C#的方式，仿佛又回到了asp的时代。ExtJs的模板同样也可以组织生成灵活的html，而且代码和html有效的分离。本文将细细分析ExtJs模板的用法。  行文之前我先把公共的html和用到的js数据贴在前面，后面的代码都要引用到。  [html]  <h1>使用标签tpl和操作符for</h1>  <div class="content" id="div1"></div>  <h1>在子模板的范围内访问父元素对象</h1>  <div class="content" id="div2"></div>  <h1>数组元素索引和简单运算支持</h1>  <div class="content" id="div3"></div>  <h1>自动渲染单根数组</h1>  <div class="content" id="div4"></div>  <h1>条件逻辑判断</h1>  <div class="content" id="div5"></div>  <h1>即时执行任意的代码</h1>  <div class="content" id="div6"></div>  <h1>模板成员函数</h1>  <div class="content" id="div7"></div>  定义data数据源：  [Js]  var data = {  name: '张三',  job: 'C#程序员',  company: '惠普',  email: 'zhangsan@163.com',  address: '武汉市洪山区光谷软件园',  city: '武汉',  state: '正常',  zip: '430000',  drinks: ['绿茶', '红酒', '咖啡'],  friends: [{  name: '李四',  age: 6,  like: '鲜花'  }, {  name: '王五',  age: 26,  like: '足球'  }, {  name: '赵六',  age: 81,  like: '游戏'  }]  };  **一、使用标签tpl和操作符for**  现在我要把data数据源展示到页面上，并组织到table里面。但是这个html不是写死的，而是通过模板生成。配合使用标签tpl和操作符for,可以循环输出张三的朋友：  [Js]  //使用标签tpl和操作符for  var tpl = new Ext.XTemplate(  '<table cellpadding=0 cellspacing=0 border=1 width=400px>',  '<tr><td colspan=2 align=center><b>{name}的个人资料</b></td></tr>',  '<tr><td>姓名：</td><td>{name}</td></tr>',  '<tr><td>工作：</td><td>{job}</td></tr>',  '<tr><td>公司：</td><td>{company}</td></tr>',  '<tr><td>地址：</td><td>{address}</td></tr>',  '<tr><td>喜好饮品：</td><td>{drinks}</td></tr>',  '<tr><td>他的好友：</td><td>',  '<tpl for="friends">',  '<p>{name}：{age}岁</p>',  '</tpl></td></tr>',  '</table>'  );  tpl.overwrite(Ext.get("div1"), data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315452651.jpg  **二、在子模板的范围内访问父元素对象**  当我们在遍历输出张三的朋友时，需要再次访问张三的信息，怎么做呢，看如下js：  [Js]  //在子模板的范围内访问父元素对象  var tp2 = new Ext.XTemplate(  '<tpl for="friends">',  '<p>{name}是{parent.name}的好友。</p>',  '</tpl>'  );  tp2.overwrite(Ext.get("div2"), data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315461135.jpg  **三、数组元素索引和简单运算支持**  在循环访问数组时，可以通过{#}获取索引信息,基础数据还可以支持简单运算:  [Js]  //数组元素索引和简单运算支持  var tp3 = new Ext.XTemplate(  '<tpl for="friends">',  '<p>{#}、一年后,{name}的年龄是：{age+1}</p>',  '</tpl>'  );  tp3.overwrite(Ext.get("div3"), data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315463338.jpg  **四、自动渲染单根数组**  通过{.}可以自动渲染没有键值对的单根数组，示例如下:  [Js]  //自动渲染单根数组  var tp4 = new Ext.XTemplate(  '喜好饮品：<tpl for="drinks">',  ' {.}',  '</tpl>'  );  tp4.overwrite(Ext.get("div4"), data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315465679.jpg  **五、条件逻辑判断**  配合标签tpl和操作符if的使用，可以做一些简单的逻辑判断，注意没有else操作符，你可以写两个if来代替。另外大于，小于符号要经过html编码，不能直接写出。  [Js]  //条件逻辑判断  var tp5 = new Ext.XTemplate(  '<table cellpadding=0 cellspacing=0 border=1 width=400px>',  '<tr><td>他的好友：</td><td>',  '<tpl for="friends">',  '<tpl if="age < 18"><p>{name}：[未成年]</p></tpl>',  '<tpl if="age >= 18"><p>{name}：{age}岁</p></tpl>',  '</tpl></td></tr>',  '</table>'  );  tp5.overwrite("div5", data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315471969.jpg  **六、即时执行任意的代码**  在XTemplate中，{[ ... ]}范围内的内容会在模板作用域的范围下执行。这里有一些特殊的变量：  values：当前作用域下的值。若想改变其中的值，你可以切换子模板的作用域。  parent：父级模板的对象  xindex：若是循环模板，这是当前循环的索引index（从1开始）。  xcount：若是循环模板，这是循环的次数 。  [Js]  //即时执行任意的代码  var tp6 = new Ext.XTemplate(  '当前日期：{[new Date().toLocaleDateString()]}',  '<table cellpadding=0 cellspacing=0 border=1 width=400px>',  '<tpl for="friends"><tr>',  '<tpl if="xindex == 1"><td rowspan={[xcount]}>他的好友：</td></tpl>',  '<td>{["姓名：" + values.name + "，年龄：" + values.age + "，"+ (values.like=="鲜花"?"是个女孩":"是个男孩")]}</td>',  '</tr></tpl>',  '</table>'  );  tp6.overwrite("div6", data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315474579.jpg  **七、模板成员函数**  在模板中还可以调用自定义函数，这些函数通过配置传入。相关写法如下：  [Js]  //模板成员函数  var tp7 = new Ext.XTemplate(  '<b>他的好友：</b><tpl for="friends"><p>',  '<tpl if="this.isWoman(like)">{name}：是个女性。</tpl>',  '<tpl if="this.isMen(like)">{name}：是个男性。</tpl>',  '<tpl if="this.isChild(age)">{name}：是个小孩。</tpl>',  '</p></tpl>', {  isWoman: function (like) {  return like == '鲜花';  },  isMen: function (like) {  return like != "鲜花";  },  isChild: function (age) {  return age < 18;  }  }  );  tp7.overwrite(Ext.get("div7"), data);  查看输出效果：  http://pic002.cnblogs.com/images/2011/99414/2011120315480814.jpg | |  |  |  |  | | --- | --- | | |  | | --- | | [lipan](http://www.cnblogs.com/lipan/)  "好好的学习，珍惜每一寸光阴" |   posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第五章 Ext.Button 按钮**](http://www.cnblogs.com/lipan/archive/2011/12/13/2274797.html) 从本篇开始讲基础控件，ExtJs对所有的UI控件都有它自己的一套封装。本篇要讲到的是ExtJs的按钮。主要包括按钮事件、带分割线、带图标、带菜单的按钮，下拉选项式按钮和按钮组合等。我们先看看效果图:  **预览**  http://pic002.cnblogs.com/images/2011/99414/2011120316384433.jpg  如下是用到的html：  [html]  <h1>  三种方式实现事件：  </h1>  <div id="div1" class="content">  <ul>  <li id="li1"></li>  <li id="li2"></li>  <li id="li3"></li>  </ul>  </div>  <h1>  带图标菜单：  </h1>  <div id="div2" class="content">  </div>  <h1>  带分割线的按钮</h1>  <div id="div3" class="content">  </div>  <h1>  菜单式按钮</h1>  <div id="div4" class="content">  </div>  <h1>  按钮组合</h1>  <div id="div5" class="content">  </div>  **一、基本按钮，三种方式实现按钮事件**  这里介绍了最基本的按钮生成代码，第一个按钮具备弹起和按下两种状态，三个按钮分别别设置成三种大小。每个按钮的单击事件都是通过一种新的实现方式。下面看看Js代码：  [Js]  Ext.create("Ext.Button", {  renderTo: Ext.get("li1"),  text: "事件实现1",  allowDepress: true, //是否允许按钮被按下的状态  enableToggle: true, //是否允许按钮在弹起和按下两种状态中切换  handler: function () {  Ext.Msg.alert("提示", "第一个事件");  },  id: "bt1"  });  Ext.create("Ext.Button", {  renderTo: Ext.get("li2"),  text: "事件实现2",  listeners: { "click": function () {  Ext.Msg.alert("提示", "第二个事件");  }  },  id: "bt2",  scale: 'medium'  });  var bt3 = Ext.create("Ext.Button", {  renderTo: Ext.get("li3").dom,  text: "事件实现3",  id: "bt3",  scale: 'large'  });  bt3.on("click", function () {  Ext.Msg.alert("提示", "第三个事件");  });  **二、带图标菜单**  按钮可以带图标和菜单，我们可以在配置项里面配置：  [Js]  Ext.create("Ext.Button", {  renderTo: Ext.get("div2"),  id: "bt4",  text: "带菜单带图标",  iconCls: "add16",  menu:  {  items: [  {  text: '选项1'  }, {  text: '选项2'  }, {  text: '选项3',  handler: function () {  Ext.Msg.alert("提示", "来自菜单的消息");  }  }  ]  }  }).showMenu();  Ext.create("Ext.Button", {  renderTo: Ext.get("div2"),  id: "bt5",  text: "上图标下菜单",  iconCls: "add16",  iconAlign: 'top',  menu:  {  items: [  {  text: '选项1'  }, {  text: '选项2'  }, {  text: '选项3',  handler: function () {  Ext.Msg.alert("提示", "来自菜单的消息");  }  }  ]  },  arrowAlign: 'bottom'  });  **三、带分割线的按钮**  注意的地方：分割线的按钮来自于类Ext.SplitButton  [Js]  Ext.create("Ext.button.Split", {  renderTo: Ext.get("div3"),  text: "右图标下菜单",  iconCls: "add16",  iconAlign: 'right',  menu:  {  items: [  {  text: '选项1'  }, {  text: '选项2'  }, {  text: '选项3',  handler: function () {  Ext.Msg.alert("提示", "来自菜单的消息");  }  }  ]  },  arrowAlign: 'bottom'  });  **四、菜单式按钮**  按钮式菜单Ext.CycleButton与下拉不同的是，它具备选中状态，当选中下拉项时，选中文本会相应变化。  [Js]  Ext.create('Ext.button.Cycle', {  renderTo: Ext.get("div4"),  showText: true,  prependText: '请选择：',  menu:  {  items: [{  text: '选项1',  checked: true  }, {  text: '选项2'  }, {  text: '选项3'  }]  },  changeHandler: function (btn, item) {  Ext.Msg.alert('修改选择', item.text);  }  });  **四、按钮组合**  定义了一组按钮，并可以控制按钮排版。  [Js]  Ext.create("Ext.ButtonGroup",{  renderTo: Ext.get("div5"),  title: "按钮组合",  columns: 3,  //defaultType:'splitbutton',  items: [{  text: '按钮1',  iconCls: 'add16',  scale: 'large',  rowspan: 2  }, {  text: '按钮2', iconCls: 'add16', rowspan: 2, scale: 'large'  }, {  text: '按钮3', iconCls: 'add16'  }, {  xtype: 'splitbutton', text: '分割线按钮', iconCls: 'add16', menu: [{ text: '菜单1'}]  }]  }); | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第六章 Ext.MessageBox 消息对话框**](http://www.cnblogs.com/lipan/archive/2011/12/14/2275362.html) 本篇演示消息对话框的用法，ExtJs封装了可能用到的各类消息框，并支持自定义的配置。  如下是用到的html：  [html]  <h1>各种消息框</h1>  <div id="div1" class="content">  <button id="bt1" type="button" >Confirm</button>  <button id="bt2" type="button" >Prompt</button>  <button id="bt3" type="button" >DIY窗口</button>  <button id="bt4" type="button" >进度条</button>  <button id="bt5" type="button" >进度条2</button>  <button id="bt6" type="button" >wait</button>  </div>  **一、警告对话框和确认对话框**  展示一个带“是”，“否”按钮的确认对话框。当单击按钮时，执行回调函数，获取按钮类型，并弹出一个警告对话框。  [Js]  Ext.get("bt1").on("click", function () {  Ext.MessageBox.confirm("标题", "详细信息内容", function (btn) {  Ext.Msg.alert("提示", "你点击了" + btn + "按钮");  });  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120412591727.jpg  http://pic002.cnblogs.com/images/2011/99414/2011120412594967.jpg  **二、输入对话框**  展示一个带文本框的对话框，可以供用户录入。单击按钮后可以获取文本框的内容。  [Js]  Ext.get("bt2").on("click", function () {  Ext.MessageBox.prompt(  "标题",  "详细信息内容",  function (btn, text) {  Ext.Msg.alert("提示", "你点击了" + btn + "按钮,获取的文本：" + text);  },  this,  true, //表示文本框为多行文本框  "初始文本");  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120413004925.jpg  **三、自定义DIY对话框**  展示一个自定义的对话框，可以定义图标样式，按钮组的类型，是否带文本框，是否带进度条等信息。  [Js]  Ext.get("bt3").on("click", function () {  Ext.MessageBox.show({  title: "标题",  msg: "详细信息内容",  buttons: Ext.MessageBox.YESNOCANCEL, //对话框的按钮组合  multiline: false, //有文本框时，是否为多行文本框  closable: false, //是否可关闭  prompt: true,  icon: Ext.MessageBox.WARNING,  iconCls: "add16",  width: 400,  proxyDrag: true,  value: "初始文本",  progress: true,  progressText: "加载中..",  animateTarget: "bt3"  });  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120413013730.jpg  **四、加载进度条对话框**  展示一个带加载进度条的对话框，提示当前执行任务的进度信息。  [Js]  Ext.get('bt4').on('click', function () {  Ext.MessageBox.show({  title: '加载窗口',  msg: '详细信息内容',  progressText: '加载中...',  width: 300,  progress: true,  closable: false,  animateTarget: 'bt4'  });  //模拟加载环境  var f = function (v) {  return function () {  if (v == 12) {  Ext.MessageBox.hide();  Ext.Msg.alert("提示", "加载完毕！");  } else {  var i = v / 11;  Ext.MessageBox.updateProgress(i, Math.round(100 \* i) + '% 完成');  }  };  };  for (var i = 1; i < 13; i++) {  setTimeout(f(i), i \* 200);  }  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120413030466.jpg  **五、等待进度条对话框**  展示等待进度条的对话框，提示用户当前正在等待某一任务执行。  [Js]  Ext.get('bt5').on('click', function () {  Ext.MessageBox.show({  msg: '正在保存数据..',  progressText: '保存中...',  width: 300,  wait: true,  waitConfig: { interval: 200 },  icon: 'download',  animateTarget: 'bt5'  });  setTimeout(function () {  Ext.MessageBox.hide();  Ext.Msg.alert("提示", "保存完毕！");  }, 3000);  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120413041340.jpg  **六、基本的等待对话框**  这里演示基本的等待对话框的实现方式。  [Js]  Ext.get('bt6').on('click', function () {  Ext.MessageBox.wait("详细信息内容", "标题", {  interval: 100 //进度条加载速度  });  setTimeout(function () {  Ext.MessageBox.hide();  Ext.Msg.alert("提示", "完毕！");  }, 35000);  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011120413043318.jpg | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第七章 Ext.tip.ToolTip 提示**](http://www.cnblogs.com/lipan/archive/2011/12/16/2286837.html) 本篇介绍提示控件，ExtJs支持两种方式定义提示，可以支持普通html元素和一般的ExtJs UI控件。  **一、基本提示 Ext.tip.ToolTip**  **1.最简单的提示**  下面通过代码定义一个最简单的提示，首先在HTML加入一个div，我们要实现当鼠标移动到这个div上时，自动出现提示，如下是html内容：  [html]  <div id="tip1" class="TipDiv">普通提示</div>  接着在js中添加如下代码：  [Js]  Ext.create('Ext.tip.ToolTip', {  target: 'tip1',  html: '最简单的提示'  });  OK,第一个提示已经添加成功，我们来预览一下效果：  http://pic002.cnblogs.com/images/2011/99414/2011121321312535.jpg  **2.可关闭的提示**  [html]  <div id="tip2" class="TipDiv">不自动隐藏</div>  [Js]  Ext.create('Ext.tip.ToolTip', {  target: 'tip2',  html: '请点击关闭按钮',  title: '标题',  autoHide: false,  closable: true,  draggable: true //可以允许被拖动  });  效果如下，鼠标移移出后提示不消失，单击叉即可关闭：  http://pic002.cnblogs.com/images/2011/99414/2011121321320655.jpg  **3.Ajax提示，提示的内容来自服务端**  [html]  <div id="tip3" class="TipDiv"> Ajax提示</div>  [Js]  Ext.create('Ext.tip.ToolTip', {  target: 'tip3',  width: 200,  autoLoad: { url: 'AjaxTipData', params: { data: "测试参数"} },  dismissDelay: 15000 //15秒后自动隐藏  });  在服务端通过MVC控制层action来返回提示内容，代码如下：  [c#]  public ContentResult AjaxTipData(string data)  {  return Content("<font color='red'>这是Ajax提示信息：</font><br>客户端参数：" + data);  }  效果：  http://pic002.cnblogs.com/images/2011/99414/2011121321323290.jpg  **4.提示跟随鼠标移动**  [html]  <div id="tip4" class="TipDiv">跟随鼠标</div>  [Js]  Ext.create('Ext.tip.ToolTip', {  target: 'tip4',  html: '跟随鼠标的提示',  trackMouse: true // 跟随鼠标移动  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011121321334229.jpg  **5.带箭头的提示**  [html]  <div id="tip6" class="TipDiv">指定提示方向</div>  [Js]  Ext.create('Ext.tip.ToolTip', {  target: 'tip6',  anchor: 'buttom', //指定箭头的指向 top,left,right  width: 120,  anchorOffset: 50, //指定箭头的位置  html: '带箭头的提示，并指定方向'  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011121321343245.jpg  **5.图文并茂的提示内容**  在提示内容中可以加入图片，超链接等以及如何html元素，还可以自定义提示标题：  [html]  <div id="tip7" class="TipDiv"> 高级自定义</div>  <div style="display:none;">  <div id="tipContent">  <ul>  <li>提示项1</li>  <li>提示项2</li>  <li>提示项3</li>  <li>提示项4</li>  </ul>  <img src="http://www.cnblogs.com/Img/Ext/house.jpg" alt="图片" />  </div>  </div>  [Js]  Ext.create('Ext.tip.ToolTip', {  title: '<a href="#">链接式标题</a>',  id: 'toolTip7',  target: 'tip7',  anchor: 'left',  html: null,  width: 415,  autoHide: false,  closable: true,  contentEl: 'tipContent', //用id为tipContent的html标签内容作为提示信息  listeners: {  'render': function () {  this.header.on('click', function (e) {  e.stopEvent();  Ext.Msg.alert('提示', '标题被点击.');  Ext.getCmp('toolTip7').hide();  }, this, { delegate: 'a' });  }  }  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011121321351175.jpg  **二、快速提示 Ext.tip.QuickTip**  快速提示通过在html上添加特定的属性就可以体现出来，比较方便，只需要在代码里面通过如下方式初始化：  [Js]  Ext.QuickTips.init();  下面看看使用方式：  [html]  <div id="tip5" class="TipDiv" data-qtip="用HTML属性表示的提示" data-qtitle="标题"> 快速提示</div>  <div id="tip52" class="TipDiv" data-qtip="设置了宽度、位置的快速提示" data-qwidth="400" data-qalign="tl-br"> 快速提示2</div>  **data-qtip:**设置提示正文内容。 **data-qtitle:**设置提示的标题。 **data-qwidth:**设置提示的宽度。 **data-qalign:**表示用提示的一个基准点，对应到原件的哪个基准点。例如：tl-br表示用提示的左上角，对应到原件的右下角。  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011121321364531.jpg  http://pic002.cnblogs.com/images/2011/99414/2011121321372629.jpg  **三、在extjs控件上使用提示**  **1.按钮上的快速提示**  首先也要运行如下代码：  [Js]  Ext.QuickTips.init();  这样按钮配置项就可以使用“tooltip”了：  [Js]  Ext.create("Ext.Button", {  renderTo: Ext.get("tipdiv"),  text: "按钮上的快速提示",  tooltip: "提示信息"  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011121321384556.jpg  **2.按钮上的自定义提示**  [Js]  //按钮上的自定义提示  Ext.create("Ext.Button", {  renderTo: Ext.get("tipdiv"),  text: "按钮上的自定义提示",  id: "bt1"  });  Ext.create('Ext.tip.ToolTip', {  target: 'bt1',  anchor: 'buttom',  width: 120,  anchorOffset: 50,  html: '按钮上的自定义提示信息'  });  效果展示：  http://pic002.cnblogs.com/images/2011/99414/2011121321390689.jpg | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第八章 Ext.slider 滚轴控件、 Ext.ProgressBar 进度条控件、 Ext.Editor 编辑控件**](http://www.cnblogs.com/lipan/archive/2011/12/19/2289532.html) 本篇要登场的有三个控件，分别是滚轴控件、进度条控件和编辑控件。  **一、滚轴控件 Ext.slider**  **1.滚轴控件的定义**  下面我们定义三个具有代表意义滚轴控件，分别展示滚轴横向、纵向，以及单值、多值选择的特性：  [html]  <h1>滚轴控件</h1>  <div class="content">  <h2>横向，初始值50</h2>  <div id="slider1"></div>  <h2>纵向，带提示</h2>  <div id="slider2"></div>  <h2>多值，自定义提示</h2>  <div id="slider3"></div>  </div>  [Js]  //横向，初始值50  var slider1 = Ext.create('Ext.slider.Single', {  renderTo: 'slider1',  width: 214,  minValue: 0,  maxValue: 100,  value: 50  });  //纵向，带提示  new Ext.create('Ext.slider.Single', {  renderTo: 'slider2',  height: 150,  minValue: 0,  maxValue: 20,  vertical: true,  plugins: new Ext.slider.Tip()  });  //多值，自定义提示  var slider3 = Ext.create('Ext.slider.Multi', {  renderTo: 'slider3',  width: 214,  minValue: 0,  maxValue: 20,  values: [5, 12],  plugins: new Ext.slider.Tip({  getText: function (thumb) {  return Ext.String.format('当前：<b>{0}/20</b>', thumb.value);  }  })  });  **2.获取、设置滚轴控件的值**  [html]  <h1>操作滚轴控件</h1>  <div class="content">  <button id="button1">设置滚轴1的值为10</button>  <button id="button2">获取滚轴1的值</button>  <button id="button3">设置滚轴3的值为10,15</button>  <button id="button4">获取滚轴3的值集合</button>  </div>  [Js]  //设置滚轴1的值为10  Ext.fly("button1").on('click', function () {  slider1.setValue(10);  });  //获取滚轴1的值  Ext.fly("button2").on('click', function () {  Ext.MessageBox.alert("获取值", "滚轴1的值：" + slider1.getValue());  });  //设置滚轴3的值为10,15  Ext.fly("button3").on('click', function () {  slider3.setValue(0, 10);  slider3.setValue(1, 15);  });  //获取滚轴3的值集合  Ext.fly("button4").on('click', function () {  Ext.MessageBox.alert("获取值", "滚轴3的值集合：" + slider3.getValues());  });  **3.效果展示**  http://pic002.cnblogs.com/images/2011/99414/2011121522551427.jpg  **二、进度条控件 Ext.ProgressBar**  **1.加载进度条**  [html]  <div class="content">  <button id="button1">执行</button>  <div id="p1" style="width: 300px;"></div>  </div><br />  [Js]  //加载进度条  var progressBar1 = Ext.create("Ext.ProgressBar", {  id: "progressBar1",  text: '准备中...',  renderTo: 'p1'  });  Ext.fly("button1").on('click', function () {  //模拟加载环境  var f = function (v) {  return function () {  var i = v / 12;  progressBar1.updateProgress(i, '进度：' + v + '/12');  if (v == 12) {  Ext.Msg.alert("提示", "加载完毕！");  progressBar1.reset(); //复位进度条  progressBar1.updateText("完成。");  }  };  };  for (var i = 1; i < 13; i++) {  setTimeout(f(i), i \* 200);  }  });  **2.等候进度条**  [html]  <div class="content">  <button id="button2">执行</button>  <div id="p2"></div>  <span id="p2text"></span>  </div><br />  [Js]  //等候进度条  var pbar2 = Ext.create("Ext.ProgressBar", {  id: "progressBar2",  renderTo: 'p2',  width: '150px'  });  pbar2.on('update', function (val) {  //每次更新可以执行的动作  Ext.fly('p2text').dom.innerHTML += '>';  });  Ext.fly("button2").on('click', function () {  Ext.fly('p2text').update('正在启动windows2000：');  pbar2.wait({  interval: 200, //每次更新的间隔周期  duration: 5000, //进度条运作时间的长度，单位是毫秒  increment: 5, //进度条每次更新的幅度大小,表示走完一轮要几次（默认为10）。  fn: function () { //当进度条完成自动更新后执行的回调函数。该函数没有参数。  Ext.fly('p2text').update('完成。');  }  });  });  **3.等候进度条，等待第三方事件**  [html]  <div class="content">  <button id="button3">执行</button>  <div id="p3"></div>  <span id="p3text"></span>  </div>  [Js]  //等候进度条，当第三方事件结束时，停止。  var pbar3 = Ext.create("Ext.ProgressBar", {  renderTo: 'p3',  width: '250px'  });  Ext.fly("button3").on('click', function () {  pbar3.wait({  interval: 100,  increment: 5  });  Ext.fly('p3text').update('第三方事件正在执行，请稍候....');  setTimeout(function () {  pbar3.reset();  Ext.fly('p3text').update('执行完毕.');  }, 5000);  });  **4.效果展示**  http://pic002.cnblogs.com/images/2011/99414/2011121522572535.jpg  **三、编辑控件 Ext.Editor**  编辑控件可以作用在一般html元素或者其他ext基本控件上，从而然这些基本元素和控件具备了编辑某些值的能力。  **1.用文本框编辑普通文本**  下面通过一个编辑控件作用在span标签上，双击该标签即可编辑该标签的文本类容。  [html]  <h1>用文本框编辑普通文本</h1>  <div class="content">  <span id="span1">请双击我修改文字</span>  </div>  [Js]  //用文本框编辑普通文本  var editor1 = new Ext.Editor({  shadow: false,  completeOnEnter: true, //按回车时自动完成  cancelOnEsc: true, //按ESC自动退出编辑  updateEl: true, //有变化时更新  ignoreNoChange: true, //不理会没有变化的情况  listeners: {  complete: function (editor, value, oldValue) {  Ext.Msg.alert('文本被改变', "从“" + oldValue + "” 变为“" + value + "”");  }  },  field: {  allowBlank: false,  xtype: 'textfield',  width: 150,  selectOnFocus: true  }  });  Ext.get("span1").on('dblclick', function (event, span1\_dom) {  editor1.startEdit(span1\_dom);  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011121522581161.jpg  处于编辑状态时：  http://pic002.cnblogs.com/images/2011/99414/2011121522583186.jpg  **2.用下拉列表编辑**  这个例子要修改Ext.Panel控件的标题。  [html]  <h1>用下拉列表编辑</h1>  <div class="content" id="div2">    </div>  [Js]  //用下拉列表编辑  var editor2 = new Ext.Editor({  shadow: false,  completeOnEnter: true,  cancelOnEsc: true,  updateEl: true,  ignoreNoChange: true,  listeners: {  complete: function (editor, value, oldValue) {  Ext.Msg.alert('文本被改变', "从“" + oldValue + "” 变为“" + value + "”");  }  },  field: {  width: 110,  id: "combo1",  //renderTo: 'div2',  triggerAction: 'all',  xtype: 'combo',  editable: false,  forceSelection: true,  store: ['下拉项1', '下拉项2', '下拉项3']  }  });  var panel = new Ext.Panel({  renderTo: "div2",  width: 200,  height: 50,  collapsible: true,  layout: 'fit',  title: '请双击标题',  listeners: {  afterrender: function (panel) {  panel.header.titleCmp.textEl.on('dblclick', function (event, label1\_dom) {  editor2.startEdit(label1\_dom);  });  }  }  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011121522585529.jpg  处于编辑状态时：  http://pic002.cnblogs.com/images/2011/99414/2011121522591581.jpg | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第九章 Ext.Panel 面板控件、 Ext.window.Window 窗口控件、 Ext.container.Viewport 布局控件**](http://www.cnblogs.com/lipan/archive/2011/12/20/2294118.html) 本篇讲解三个容器类控件。  **一、面板控件 Ext.Panel**  一个面板控件包括几个部分，有标题栏、工具栏、正文、按钮区。标题栏位于最上面，工具栏可以在四个位置放置，围绕中间部分正文，按钮区位于最小方。下面介绍几个基本配置项：  **1.title:**设置面板标题文本。  **2.tbar,lbar,rbar,bbar:**分别设置上、左、右、下四个部位的工具栏。  **3.html，items:**前者用来设置正文部分的html，后者设置正文部分的ext控件。  **4.buttons:**设置按钮区的按钮。  下面看看面板生成代码：  [html]  <h1>Panel</h1>  <div id="div1" class="content">  </div>  [Js]  Ext.onReady(function () {  var p = Ext.create('Ext.Panel', {  title: '面板标题',  collapsible: true,  renderTo: 'div1',  width: 400,  height: 300,  autoScroll: false,  bodyBorder: true,  buttonAlign: 'right',  buttons: [{  text: "按钮1",  handler: function () {  Ext.Msg.alert("提示", "第一个事件");  },  id: "bt1"  }, {  text: "按钮2",  id: "bt2"  }  ],  floating: true,  footerCfg: { tag: 'span', id: 'span1', html: '面板底部' },  items: [{  xtype: "button",  text: "按钮"  }],  tbar: Ext.create('Ext.toolbar.Toolbar', { items: ["工具栏"] }),  html: "<b>正文</b>"  });  p.setPosition(40, 50);  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122008155045.jpg  **二、窗口控件 Ext.window.Window**  窗口控件与面板控件基本类似，只不过他看起来像一个窗口，具备最大化，最小化，打开关闭、拖动等窗口操作，下面看看窗口生成代码：  下面看看面板生成代码：  [html]  <h1>窗口</h1>  <div class="content" style="height:300px">  <button id="button1">打开窗口</button>  <div id="win1"></div>  </div>  [Js]  Ext.onReady(function () {  var window1 = Ext.create('Ext.window.Window', {  applyTo: 'win1',  layout: 'table', //内部元素布局方式{absolute accordion anchor border card column fit form table}  width: 500,  height: 200,  closeAction: 'hide', //窗口关闭的方式：hide/close  plain: true,  title: "窗口标题",  maximizable: true, //是否可以最大化  minimizable: true, //是否可以最小化  closable: false, //是否可以关闭  modal: true, //是否为模态窗口  resizable: false, //是否可以改变窗口大小  items: [{  text: '按钮',  xtype: "button"  }, {  width: 214,  minValue: 0,  maxValue: 100,  value: 50,  xtype: "slider"  }, {  xtype: "button",  text: '一个菜单',  width: "60px",  height: "15px",  menu: {  items: [  new Ext.ColorPalette({  listeners: {  select: function (cp, color) {  Ext.Msg.alert('颜色选择', '你选择了' + color + '。');  }  }  }), '-',  { text: '菜单项1' }, { text: '菜单项2' }, { text: '菜单项3' }  ]  }  }],  buttons: [{  text: '确定',  disabled: true  }, {  text: '取消',  handler: function () {  window1.hide();  }  }]  });  Ext.fly("button1").on("click", function () {  window1.show(Ext.get("button1"));  });  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122008160729.jpg  **三、布局控件 Ext.container.Viewport**  布局控件一般用于整个页面的排版布局，它按四个方向分为四块区域，和中间正文部分，四个区域都可以自动隐藏，其实这个控件的核心功能就是用到了“border”方式的布局，下面看看生成代码：  [Js]  Ext.onReady(function () {  Ext.create('Ext.container.Viewport', {  layout: 'border',  items: [{  region: 'north',  html: '<h1>这里放置logo</h1>',  xtype: "panel",  title: "标题",  autoHeight: true,  border: false,  margins: '0 0 5 0'  }, {  region: 'west',  collapsible: true,  title: '左侧导航',  xtype: 'panel',  width: 200,  autoScroll: true  }, {  region: 'center',  xtype: 'tabpanel',  activeItem: 0,  items: {  title: '首页',  html: '这里是首页正文内容'  }  }, {  region: 'south',  title: '底部',  collapsible: true, //允许折叠  html: '这里放置版权信息',  split: true,  height: 100,  minHeight: 100  }]  });  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122008162435.jpg | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第十章 Ext.tab.Panel 选项卡**](http://www.cnblogs.com/lipan/archive/2011/12/21/2295355.html) 本篇讲解选项卡控件。  **一、基本选项卡**  首先我们来定义一个基本的选项卡控件，其中每个Tab各有不同，Tab的正文内容可以有三种方式获取：  **1.基本方式:**通过定义html和items的方式。  **2.读取其他html的信息:**通过设置contentEl就可以获取其他html的信息为当前tab正文。  **3.读取服务端数据:**通过定义autoLoad异步方式获取服务端数据。  另外，每个tab都可以设置是否可关闭，进入tab时的事件，以及tab是否可用，具体情况请看代码：  [html]  <h1>基本选项卡</h1>  <div class="content" style="height: 150px">  <div id="tabPanel">  <div style="display: none">  <div id="oneTab">  <p>这个tab所展示的内容是读取至其他HTML标签</p>  </div>  </div>  </div>  </div>  [Js]  //1.基本的选项卡  var tabs1 = Ext.createWidget('tabpanel', {  renderTo: "tabPanel",  activeTab: 1, //指定默认的活动tab  width: 600,  height: 120,  plain: true, //True表示tab候选栏上没有背景图片（默认为false）  enableTabScroll: true, //选项卡过多时，允许滚动  defaults: { autoScroll: true },  items: [{  id: "tab1",  title: '普通Tab',  html: "这只是一个非常普通的Tab。",  items:[{xtype:'button',text:'按钮'}],  closable: true //这个tab可以被关闭  }, {  id: "tab2",  title: '内容来至div',  contentEl: 'oneTab' //指定了当前tab正文部分从哪个html元素读取  }, {  id: "tab3",  title: 'Ajax Tab',  autoLoad: { url: 'AjaxTabContent', params: { data: "从客户端传入的参数" }, method: 'GET' }  }, {  id: "tab4",  title: '事件Tab',  listeners: { activate: handleActivate },  html: "带事件的Tab。"  }, {  id: "tab5",  title: '不可用Tab',  disabled: true,  html: "不可用的Tab，你是看不到我的。"  }]  });  //单击tab4后触发的事件  function handleActivate(tab) {  alert(tab.title + ': activated事件触发。');  }  我们查看一下生成的选项卡效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101075480.jpg  **二、操作选项卡**  选项卡生成后，我们可以通过js去操作它，比如动态新增、删除、插入选项卡，设置活动选项卡等，我们看看具体实现方法：  [html]  <h1>操作选项卡</h1>  <div class="content" id="content2"></div>  [Js]  var index = 0;  //新增一个Tab  Ext.createWidget("button", {  text: "新增一个Tab",  renderTo: 'content2',  handler: function () {  tabs1.add({  title: '新Tab ' + (++index),  id: "newTab" + index,  html: '选项卡文本部分 ' + (index) + '<br/><br/>',  closable: true  });  }  });  //插入一个Tab  Ext.createWidget("button", {  text: "在2号位置插入新Tab",  renderTo: 'content2',  handler: function () {  tabs1.insert(2, {  title: '新Tab ' + (++index),  id: "newTab" + index,  html: '选项卡文本部分 ' + (index) + '<br/><br/>',  closable: true  });  }  });  //删除一个Tab  Ext.createWidget("button", {  text: "删除2号位置的Tab",  renderTo: 'content2',  handler: function () {  tabs1.remove(2);  }  });  //删除id为“tab1”的Tab  Ext.createWidget("button", {  text: "删除id为“tab1”的Tab",  renderTo: 'content2',  handler: function () {  tabs1.remove("tab1");  }  });  //删除id为“tab1”的Tab  Ext.createWidget("button", {  text: "设置第三个Tab为活动tab",  renderTo: 'content2',  handler: function () {  tabs1.setActiveTab(2);  }  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101090571.jpg  **三、选项卡按钮在下方**  默认的选项卡按钮在上方，我们可以随意定义选项卡按钮的位置，下面代码演示了具体的用法：  [html]  <h1>选项卡按钮在下方</h1>  <div class="content" id="content3"></div>  [Js]  //选项卡按钮在下方  var tabs3 = Ext.createWidget('tabpanel', {  renderTo: "content3",  activeTab: 0,  width: 600,  height: 150,  tabPosition: 'bottom' //指定了选项卡的位置，left,right  });  for (var i = 0; i < 3; i++)  tabs3.add({  title: 'Tab ' + i,  id: "Tabs3\_" + i,  html: '选项卡文本部分 ' + (index) + '<br/><br/>',  closable: true  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101092944.jpg  **四、可拖动的选项卡**  通过官方扩展包我们可以增强选项卡控件的易用性，比如现在我们可以实现一个可以拖动选项卡按钮的功能：  [html]  <h1>可拖动的选项卡</h1>  <div class="content" id="content4"></div>  [Js]  //首先要动态加载ux扩展的js  Ext.Loader.setConfig({enabled: true});  Ext.Loader.setPath('Ext.ux', '/ExtJs/ux');  Ext.require([  'Ext.tip.QuickTipManager',  'Ext.tab.Panel',  'Ext.ux.TabScrollerMenu',  'Ext.ux.TabReorderer',  'Ext.ux.TabCloseMenu',  'Ext.ux.GroupTabPanel'  ]);  //以下是功能代码  //可拖动的选项卡  var tabs4 = Ext.createWidget('tabpanel', {  renderTo: "content4",  activeTab: 0,  width: 600,  height: 150,  plugins: Ext.create('Ext.ux.TabReorderer'),  items: [{  xtype: 'panel',  title: 'tab不可拖',  html: "这个选项卡不可被拖动",  reorderable: false,  closable: true  }]  });  for (var i = 0; i < 3; i++)  tabs4.add({  title: 'Tab ' + i,  id: "Tabs4\_" + i,  html: '选项卡文本部分 ' + (index) + '<br/><br/>'  });  效果如下，可见一个tab已经被移动：  http://pic002.cnblogs.com/images/2011/99414/2011122101110310.jpg  **五、过多选项卡的菜单式展示**  如果面板上的选项卡打开的过多而显示不下，那么需要对溢出的选项卡用菜单的方式展示出来，实现方式如下，注意要引入扩展的css样式：  [html]  <h1>过多选项卡的菜单式展示</h1>  <div class="content" id="content5"></div>  [Js]  //选项卡过多溢出时菜单显示  var tabs5 = Ext.createWidget('tabpanel', {  renderTo: "content5",  activeTab: 0,  width: 600,  height: 150,  plugins: Ext.create('Ext.ux.TabScrollerMenu', {  maxText: 15,  pageSize: 5  }),  items: [{  title: 'tab0',  html: '第一个tab'  }]  });  Ext.defer(function () {  var myTabs = [];  for (var i = 0; i < 15; i++) {  myTabs.push({  title: 'Tab ' + i,  id: "Tabs5\_" + i,  html: '选项卡文本部分 ' + (index) + '<br/><br/>'  });  }  tabs5.add(myTabs);  }, 1000);  效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101120021.jpg  **六、选项卡的右键菜单**  一般的应用程序都支持在选项卡按钮上面通过右键的方式去关闭多余的选项卡，在ext中也可以做到，实现方法如下：  [html]  <h1>选项卡的右键菜单</h1>  <div class="content" id="content6"></div>  [Js]  //选项卡的右键菜单  var currentItem;  var tabs6 = Ext.createWidget('tabpanel', {  renderTo: "content6",  activeTab: 0,  width: 600,  height: 150,  plugins: Ext.create('Ext.ux.TabCloseMenu', {  closeTabText: '关闭当前',  closeOthersTabsText: '关闭其他',  closeAllTabsText: '关闭所有',  extraItemsTail: [  '-',  {  text: '可关闭',  checked: true,  hideOnClick: true,  handler: function (item) {  currentItem.tab.setClosable(item.checked);  }  }  ],  listeners: {  aftermenu: function () {  currentItem = null;  },  beforemenu: function (menu, item) {  var menuitem = menu.child('\*[text="可关闭"]');  currentItem = item;  menuitem.setChecked(item.closable);  }  }  }),  items: [{  title: 'tab1',  html: '第一个tab'  }, {  title: 'tab2',  closable: true,  html: '第二个tab'  }, {  title: 'tab3',  closable: true,  html: '第三个tab'  }]  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101125028.jpg  **七、分组式选项卡**  我们还可以对选项卡进行分组，具体实现如下：  [html]  <h1>分组式选项卡</h1>  <div class="content" id="content7"></div>  [Js]  //分组式选项卡  var tabs7 = Ext.create('Ext.ux.GroupTabPanel', {  activeGroup: 0, //设置当前活动的分组  items: [{  expanded: false,  mainItem: 1, //设置主要的item，这个tab会在最上面，以文件夹方式展示出来。  items: [{  title: '项目1',  html: "<b>第一组第一项正文。</b>"  }, {  title: '项目2',  border: false,  html: "<b>第一组第二项正文。</b>"  }, {  title: '项目3',  border: false,  html: "<b>第一组第三项正文。</b>"  }]  }, {  expanded: true,  items: [{  title: '项目1',  html: "<b>第二组第一项正文。</b>"  }, {  title: '项目2',  html: "<b>第二组第二项正文。</b>"  }]  }]  });  Ext.create('Ext.Panel', {  renderTo: "content7",  width: 600,  height: 250,  collapsible: true,  layout: 'fit',  title: '分组tab演示',  items: tabs7  });  效果：  http://pic002.cnblogs.com/images/2011/99414/2011122101132262.jpg | |  |  |  | | --- | |  | |  | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第十一章 Ext.ListView、Ext.view.View 数据视图**](http://www.cnblogs.com/lipan/archive/2011/12/22/2296508.html) 本篇介绍两个用来展示数据的容器控件，分别是Ext.ListView和Ext.view.View。Ext.ListView就是大名鼎鼎的Ext GridPanel的前身，不过现在的Ext4已经将它整合到GridPanel一起了，做一些简单的表格数据展示还是可以用到的。而Ext.view.View则提供了通过模板自定义展示数据的方式，数据的展示形式不限于表格，可表现为灵活的方式，同时Ext为其提供了基本的排序分页、项选择、事件等支持。  如果不做特殊说明，本系列文章都以mvc的服务端返回json格式的数据作为数据源的方式。  **一、Ext.ListView**  下面我们看看一个基本的表格数据展示实例：  [html]  <h1>ListView</h1>  <div id="div1" class="content"></div>  <span id="span1"></span>  [Js]  Ext.onReady(function () {  var store = new Ext.data.JsonStore({  fields: [  { name: 'IntData', type: 'int' },  { name: 'StringData', type: 'string' },  { name: 'TimeData', type: 'date' }  ],  proxy: {  type: 'ajax',  url: 'ListView1Json',  reader: {  type: 'json',  root: 'rows'  }  },  sortInfo: { field: 'IntData', direction: 'DESC' }  });  store.load();  var listView = Ext.create('Ext.ListView', {  renderTo: "div1",  store: store,  multiSelect: true,  emptyText: '无数据',  reserveScrollOffset: true,  hideHeaders: false, //是否隐藏标题  columns: [{  header: "IntData",  dataIndex: 'IntData'  }, {  header: "StringData",  dataIndex: 'StringData'  }, {  header: "TimeData",  dataIndex: 'TimeData',  align: 'right',  xtype: 'datecolumn',  format: 'm-d h:i a'  }]  });  //当选择行改变时，输出被选行  listView.on('selectionchange', function (view, selectNodes) {  var msg = "";  for (var i = 0; i < selectNodes.length; i++) {  var index = 1 + selectNodes[i].index;  if (msg == "") {  msg = index;  }  else {  msg += "," + index;  }  }  if (msg == "") Ext.get("span1").update('当前没有选择任何数据。');  else Ext.get("span1").update('当前选择了第' + msg + '行数据。');  });  });  服务端模拟的数据源，以及通过MVC的jsonresult返回数据的代码：  [C# Mvc]  //控制层  public JsonResult ListView1Json()  {  var json = new  {  rows = BasicData.Table.Take(8).Select(x => new  {  IntData = x.IntData,  StringData = x.StringData,  TimeData = x.TimeData.ToShortDateString()  })  };  return Json(json, JsonRequestBehavior.AllowGet);  }  //模拟数据  public class BasicData  {  static List<BasicData> table;  static public List<BasicData> Table  {  get  {  if (table == null)  {  table = new List<BasicData>();  for (int i = 0; i < 1000; i++)  {  var obj = new BasicData()  {  IntData = i + 1,  StringData = "测试数据" + (i + 1),  TimeData = DateTime.Today.AddDays(i)  };  table.Add(obj);  }  }  return table;  }  }  public int IntData { get; set; }  public string StringData { get; set; }  public DateTime TimeData { get; set; }  }  来看看效果，数据已经正确加载，当我们选择行时，可以看到提示当前选择了哪些行:  http://pic002.cnblogs.com/images/2011/99414/2011122123274183.jpg  再看看通过火狐调试捕获到的服务端json数据：  http://pic002.cnblogs.com/images/2011/99414/2011122123281253.jpg  **二、Ext.view.View**  先看看实现代码：  [html]  <h1>Ext.view.View</h1>  <div class="content" id="div1"></div>  <span id="span1"></span>  [Css]  #view1 .data  {  background-color:#fff;  }  #view1 tr.hover {  background-color: #ddd;  }  #view1 .x-item-selected {  background-color:Yellow !important;  }  [Js]  Ext.Loader.setConfig({ enabled: true });  Ext.Loader.setPath('Ext.ux.DataView', '/ExtJs/ux/DataView');  Ext.onReady(function () {  //创建store  var store = Ext.create('Ext.data.Store', {  fields: ['IntData', 'StringData', 'TimeData'],  proxy: {  type: 'ajax',  url: 'DataView1Json',  reader: {  type: 'json',  root: 'rows'  }  }  });  store.load();  //定义模板  var tpl = new Ext.XTemplate(  '<table cellpadding=0 cellspacing=0 border=1 width=450px>',  '<tr><td colspan=3 align=center><b><font color=red>Ext.view.View取自服务端的数据表</font></b></td></tr>',  '<tr><td style="width:20%"><b>编号</b></td><td style="width:50%"><b>消息</b></td><td style="width:30%"><b>日期</b></td>',  '<tpl for=".">',  '<tr class="data"><td class="x-editable">{IntData}</td><td>{StringData}</td><td>{TimeData}</td></tr>',  '</tpl>',  '</table>'  );  //定义Ext.view.View控件  var view = Ext.create('Ext.view.View', {  renderTo: "div1",  store: store,  tpl: tpl,  autoHeight: true,  multiSelect: true,  //height: 310,  trackOver: true,  id: 'view1',  overItemCls: 'hover',  itemSelector: 'tr.data',  emptyText: '没有数据',  plugins: [  Ext.create('Ext.ux.DataView.DragSelector', {}),  Ext.create('Ext.ux.DataView.LabelEditor', { dataIndex: 'IntData' })  ],  listeners: {  selectionchange: function (dataView, selectNodes) {  var msg = "";  for (var i = 0; i < selectNodes.length; i++) {  var index = 1 + selectNodes[i].index;  if (msg == "") {  msg = index;  }  else {  msg += "," + index;  }  }  if (msg == "") Ext.get("span1").update('当前没有选择任何数据。');  else Ext.get("span1").update('当前选择了第' + msg + '行数据。');  }  }  });  });  [C# Mvc]  public JsonResult DataView1Json()  {  var json = new  {  rows = BasicData.Table.Take(8).Select(x => new  {  IntData = x.IntData,  StringData = x.StringData,  TimeData = x.TimeData.ToShortDateString()  })  };  return Json(json,JsonRequestBehavior.AllowGet);  }  **1.要注意的地方：**  对于数据item项，我们在其html 设置 class="data",那么我们在配置项中设置：itemSelector: 'tr.data'表明了这个CSS选择器选中的元素是一个item数据项。  当item被选择时，我们需要定义一个.x-item-selected 的样式，这样才可以在展示界面表现出item项被选中时的样式。  下面我们看看效果：  http://pic002.cnblogs.com/images/2011/99414/2011122123292340.jpg  **2.对Ext.view.View的扩展**  注意到配置项的如下代码：  [Js]  plugins: [  Ext.create('Ext.ux.DataView.DragSelector', {}),  Ext.create('Ext.ux.DataView.LabelEditor', { dataIndex: 'IntData' })  ],  这里分别引入了两个扩展支持，前者描述了数据项可以通过拖动鼠标选择多个item项，查看效果：  http://pic002.cnblogs.com/images/2011/99414/2011122123300847.jpg  第二个扩展支持了对item数据项记录集的单元格数据的编辑功能，这里表明了dataIndex为'IntData'的单元格可以被编辑。注意：在模板对应的位置要引入 class="x-editable"的样式来支持。下面看看效果：  http://pic002.cnblogs.com/images/2011/99414/2011122123305364.jpg | |  |  |  | | --- | | posts - 29, comments - 268, trackbacks - 0, articles - 0 | | [**第十二章 Ext.toolbar.Toolbar 工具栏、Ext.toolbar.Paging 分页栏、Ext.ux.statusbar.StatusBar 状态栏**](http://www.cnblogs.com/lipan/archive/2011/12/23/2298746.html) 本篇讲解三个工具栏控件。其中Ext.toolbar.Toolbar可以用来放置一些工具类操控按钮和菜单，Ext.toolbar.Paging专门用来控制数据集的分页展示，Ext.ux.statusbar.StatusBar用来展示当前的状态信息。  **一、Ext.toolbar.Toolbar**  工具栏控件可以被附加在面板、窗口等容器类控件中，可以在四个方位添加多个工具栏控件。我们演示多个Ext.toolbar.Toolbar控件，然后附加到面板的不同位置。  **1.在工具栏上添加菜单、按钮、搜索功能**  我们这里借用上一篇所讲到的listview控件作为数据展示，把listview放入一个面板控件中，然后把工具栏添加到面板顶部，并且在工具栏中实现数据集的服务端搜索的功能。  首先我们定义一个数据模型和Store：  [Js]  Ext.define('Datas', {  extend: 'Ext.data.Model',  fields: [  { name: 'IntData', type: 'int' },  { name: 'StringData', type: 'string' },  { name: 'TimeData', type: 'date' }  ],  proxy: {  type: 'ajax',  url: 'Toolbar1Json',  reader: {  type: 'json',  root: 'rows'  }  }  });  var store = new Ext.data.Store({  model: 'Datas',  sortInfo: { field: 'IntData', direction: 'DESC' },  autoLoad: true  });  store.load();  服务端的json输出代码：  [C# Mvc]  public JsonResult Toolbar1Json(string keyword)  {  var rows = BasicData.Table.Take(10).Select(x => new  {  IntData = x.IntData,  StringData = x.StringData,  TimeData = x.TimeData.ToShortDateString()  });  if (!string.IsNullOrEmpty(keyword))  {  rows = rows.Where(x => x.IntData.ToString() == keyword || x.StringData.Contains(keyword) || x.TimeData.Contains(keyword));  }  var json = new  {  results = BasicData.Table.Count,  rows = rows  };  return Json(json, JsonRequestBehavior.AllowGet);  }  接着定义一个listView，来自上篇  现在我们要定义一个toolbar，在工具栏里面添加了工具按钮、普通文字、分割线、和菜单，还实现了搜索的功能：  [Js]  var filed1 = new Ext.form.Field();  var tbar = Ext.create("Ext.Toolbar", {  items: ['文字', "-", {  xtype: "splitbutton",  text: "按钮"  }, {  text: "菜单",  menu:  {  items: [  {  text: '选项1'  }, {  text: '选项2'  }, {  text: '选项3',  handler: function () {  Ext.Msg.alert("提示", "来自菜单的消息");  }  }  ]  }  }, "->", "关键字：", filed1, {  text: "搜索",  handler: function () {  store.load({ params: { keyword: filed1.getValue()} });  }  }  ]  });  注意这里，我们通过load store，把keyword关键字传给了c#的action参数：  [Js]  {  text: "搜索",  handler: function () {  store.load({ params: { keyword: filed1.getValue()} });  }  }  最后我们定义一个Panel，把listView和toolbar都添加到Panel上，注意，tbar表示了这个工具栏在上方。  [Js]  var panel = new Ext.Panel({  renderTo: "div1",  width: 600,  height: 250,  collapsible: true,  layout: 'fit',  title: '演示工具栏',  items: listView,  tbar: tbar  });  大功告成，我们来看看效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300134049.jpg  我们输入关键字“6”后查看过滤效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300141439.jpg  **2.溢出测试**  如果工具栏上的item项目过多，而面板的长度不够那会怎么样？我们需要设置 overflowHandler: 'Menu'，代码如下：  [Js]  var bbar = Ext.create('Ext.toolbar.Toolbar', {  layout: {  overflowHandler: 'Menu'  },  items: ["溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试",  "溢出测试", "溢出测试",  {  xtype: "button",  text: "溢出按钮",  handler: function () {  Ext.Msg.alert("提示", "工具栏按钮被点击");  }  }, { text: "溢出按钮", xtype: "splitbutton"}]  });  预览下效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300145784.jpg  **3.在右侧的工具栏**  现在我们要实现放置在右侧的工具栏，这次我们直接在面板的代码里面写,代码如下：  [Js]  var panel = new Ext.Panel({  renderTo: "div1",  width: 600,  height: 250,  collapsible: true,  layout: 'fit',  title: '演示工具栏',  items: listView,  tbar: tbar,  bbar: bbar,  rbar: [{  iconCls: 'add16',  tooltip: '按钮1'  },  '-',  {  iconCls: 'add16',  tooltip: {  text: '按钮2',  anchor: 'left'  }  }, {  iconCls: 'add16'  }, {  iconCls: 'add16'  },  '-',  {  iconCls: 'add16'  }  ]  });  预览下效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300154876.jpg  最后奉上完整的代码：  [Js]  Ext.onReady(function () {  Ext.QuickTips.init();  Ext.define('Datas', {  extend: 'Ext.data.Model',  fields: [  { name: 'IntData', type: 'int' },  { name: 'StringData', type: 'string' },  { name: 'TimeData', type: 'date' }  ],  proxy: {  type: 'ajax',  url: 'Toolbar1Json',  reader: {  type: 'json',  root: 'rows'  }  }  });  var store = new Ext.data.Store({  model: 'Datas',  sortInfo: { field: 'IntData', direction: 'DESC' },  autoLoad: true  });  store.load();  var listView = Ext.create('Ext.ListView', {  store: store,  multiSelect: true,  emptyText: '当前没有数据展示',  reserveScrollOffset: true,  columns: [{  header: "IntData",  dataIndex: 'IntData'  }, {  header: "StringData",  dataIndex: 'StringData'  }, {  header: "TimeData",  dataIndex: 'TimeData',  align: 'right',  xtype: 'datecolumn',  format: 'm-d h:i a'  }]  });  var filed1 = new Ext.form.Field();  var tbar = Ext.create("Ext.Toolbar", {  items: ['文字', "-", {  xtype: "splitbutton",  text: "按钮"  }, {  text: "菜单",  menu:  {  items: [  {  text: '选项1'  }, {  text: '选项2'  }, {  text: '选项3',  handler: function () {  Ext.Msg.alert("提示", "来自菜单的消息");  }  }  ]  }  }, "->", "关键字：", filed1, {  text: "搜索",  handler: function () {  store.load({ params: { keyword: filed1.getValue()} });  }  }  ]  });  var bbar = Ext.create('Ext.toolbar.Toolbar', {  layout: {  overflowHandler: 'Menu'  },  items: ["溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试", "溢出测试",  "溢出测试", "溢出测试",  {  xtype: "button",  text: "溢出按钮",  handler: function () {  Ext.Msg.alert("提示", "工具栏按钮被点击");  }  }, { text: "溢出按钮", xtype: "splitbutton"}]  });  var panel = new Ext.Panel({  renderTo: "div1",  width: 600,  height: 250,  collapsible: true,  layout: 'fit',  title: '演示工具栏',  items: listView,  tbar: tbar,  bbar: bbar,  rbar: [{  iconCls: 'add16',  tooltip: '按钮1'  },  '-',  {  iconCls: 'add16',  tooltip: {  text: '按钮2',  anchor: 'left'  }  }, {  iconCls: 'add16'  }, {  iconCls: 'add16'  },  '-',  {  iconCls: 'add16'  }  ]  });  });  **二、Ext.toolbar.Paging**  **1.基本的分页工具栏控件**  Ext.toolbar.Paging就是一个特殊的工具栏，它提供了数据集翻页的功能，下面我们看看store的实现：  [Js]  var store = Ext.create('Ext.data.Store', {  fields: ['IntData', 'StringData', 'TimeData'],  pageSize: 15,  proxy: {  type: 'ajax',  url: 'PagingToolbar1Json',  reader: {  type: 'json',  root: 'rows',  totalProperty: 'results'  }  },  autoLoad: true  });  对应的服务端mvc的代码如下：  [C# Mvc]  public JsonResult PagingToolbar1Json(int start, int limit)  {  var json = new  {  results = BasicData.Table.Count,  rows = BasicData.Table.Skip(start).Take(limit).Select(x => new  {  IntData = x.IntData,  StringData = x.StringData,  TimeData = x.TimeData.ToShortDateString()  })  };  return Json(json, JsonRequestBehavior.AllowGet);  }  现在我们借用上篇的Ext.view.View控件，把它放置到一个面板中，面板的代码如下：  [Js]  var panel = Ext.create('Ext.Panel', {  renderTo: "div1",  frame: true,  width: 535,  autoHeight: true,  collapsible: true,  layout: 'fit',  title: '分页控件用在View',  items: Ext.create('Ext.view.View', {  store: store,  tpl: tpl,  autoHeight: true,  multiSelect: true,  id: 'view1',  overItemCls: 'hover',  itemSelector: 'tr.data',  emptyText: '没有数据',  plugins: [  Ext.create('Ext.ux.DataView.DragSelector', {}),  Ext.create('Ext.ux.DataView.LabelEditor', { dataIndex: 'IntData' })  ]  }),  bbar: Ext.create('Ext.toolbar.Paging', {  store: store,  displayInfo: true,  items: [  '-', {  text: '第10页',  handler: function () {  store.loadPage(10);  }  }]  })  });  注意上述代码，我们在分页工具栏控件中加入了一个按钮，当单击这个按钮时，数据集自动翻到第十页。  最后我们看看展示效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300175175.jpg  **2.扩展后的翻页控件**  我们可以通过ux扩展支持定义不同风格的分页控件，这效果就像三国杀扩展包一样，我们可以通过滚轴控件和进度条控件去展示当前处于分页项的哪个位置。我们在分页控件的配置部分添加如下代码:  [Js]  plugins: Ext.create('Ext.ux.SlidingPager', {})  展示效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300183752.jpg  plugins: Ext.create('Ext.ux.ProgressBarPager', {})  展示效果：  http://pic002.cnblogs.com/images/2011/99414/2011122300223358.jpg  完整的代码：  [Js]  Ext.Loader.setConfig({ enabled: true });  Ext.Loader.setPath('Ext.ux', '/ExtJs/ux');  //Ext.Loader.setPath('Ext.ux.DataView', '/ExtJs/ux/DataView');  Ext.onReady(function () {  var store = Ext.create('Ext.data.Store', {  fields: ['IntData', 'StringData', 'TimeData'],  pageSize: 15,  proxy: {  type: 'ajax',  url: 'PagingToolbar1Json',  reader: {  type: 'json',  root: 'rows',  totalProperty: 'results'  }  },  autoLoad: true  });  var tpl = new Ext.XTemplate(  '<table cellpadding=0 cellspacing=0 border=1 width=450px>',  '<tr><td colspan=3 align=center><b><font color=red>Ext.view.View取自服务端的数据表</font></b></td></tr>',  '<tr><td style="width:20%"><b>编号</b></td><td style="width:50%"><b>消息</b></td><td style="width:30%"><b>日期</b></td>',  '<tpl for=".">',  '<tr class="data"><td class="x-editable">{IntData}</td><td>{StringData}</td><td>{TimeData}</td></tr>',  '</tpl>',  '</table>'  );  var panel = Ext.create('Ext.Panel', {  renderTo: "div1",  frame: true,  width: 535,  autoHeight: true,  collapsible: true,  layout: 'fit',  title: '分页控件用在View',  items: Ext.create('Ext.view.View', {  store: store,  tpl: tpl,  autoHeight: true,  multiSelect: true,  id: 'view1',  overItemCls: 'hover',  itemSelector: 'tr.data',  emptyText: '没有数据',  plugins: [  Ext.create('Ext.ux.DataView.DragSelector', {}),  Ext.create('Ext.ux.DataView.LabelEditor', { dataIndex: 'IntData' })  ]  }),  bbar: Ext.create('Ext.toolbar.Paging', {  store: store,  displayInfo: true,  items: [  '-', {  text: '第10页',  handler: function () {  store.loadPage(10);  }  }],  plugins: Ext.create('Ext.ux.SlidingPager', {})  //plugins: Ext.create('Ext.ux.ProgressBarPager', {})  })  });  });  **三、Ext.ux.statusbar.StatusBar**  这个状态栏控件也是ext的一个扩展支持，不过它就好像军争包一样，这次不是小小改进，而是一个全新的控件。  首先定义一个函数，它在前2秒将状态栏设置为繁忙状态，2秒后恢复：  [Js]  var loadFn = function (btn, statusBar) {  btn = Ext.getCmp(btn);  statusBar = Ext.getCmp(statusBar);  btn.disable();  statusBar.showBusy();  Ext.defer(function () {  statusBar.clearStatus({ useDefaults: true });  btn.enable();  }, 2000);  };  接着我们将要几个按钮到状态栏，第一个设置状态为错误：  [Js]  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.setStatus({  text: '错误!',  iconCls: 'x-status-error',  clear: true // 自动清除状态  });  }  第二个设置状态为加载中：  [Js]  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.showBusy();  }  第三个为清除状态：  [Js]  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.clearStatus();  }  展示效果，分别是加载、错误、和清除状态：  http://pic002.cnblogs.com/images/2011/99414/2011122300231483.jpg  http://pic002.cnblogs.com/images/2011/99414/2011122300234479.jpg  http://pic002.cnblogs.com/images/2011/99414/2011122300240729.jpg  完整的代码：  Ext.Loader.setConfig({ enabled: true });  Ext.Loader.setPath('Ext.ux', '/ExtJs/ux');  Ext.onReady(function () {  var loadFn = function (btn, statusBar) {  btn = Ext.getCmp(btn);  statusBar = Ext.getCmp(statusBar);  btn.disable();  statusBar.showBusy();  Ext.defer(function () {  statusBar.clearStatus({ useDefaults: true });  btn.enable();  }, 2000);  };  var panel = new Ext.Panel({  renderTo: "div1",  width: 600,  height: 250,  collapsible: true,  //layout: 'fit',  title: '演示状态栏',  items: [{ xtype: "button", text: "测试", id:"button1", handler: function (btn, statusBar) {  loadFn("button1", "statusbar1");  }  }],  bbar: Ext.create('Ext.ux.statusbar.StatusBar', {  id: 'statusbar1',  defaultText: '就绪',  text: '没有任务',  iconCls: 'x-status-valid',  items: [  {  xtype: 'button',  text: '设置状态',  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.setStatus({  text: '错误!',  iconCls: 'x-status-error',  clear: true // 自动清除状态  });  }  },  {  xtype: 'button',  text: '设置为加载状态',  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.showBusy();  }  },  {  xtype: 'button',  text: '清除状态',  handler: function () {  var sb = Ext.getCmp('statusbar1');  sb.clearStatus();  }  }  ]  })  });  });  [Js] | |  | | |

|  |
| --- |
| posts - 29, comments - 268, trackbacks - 0, articles - 0 |
| [**第十三章 Ext.menu.Menu 菜单、Ext.draw.Component 绘图、Ext.resizer.Resizer 大小变更**](http://www.cnblogs.com/lipan/archive/2011/12/26/2301426.html)   本篇讲解菜单、绘图、还有大小变更控件。菜单控件可以附加到各种其他控件中，比如按钮、工具栏等，甚至可以直接通过通过右键打开（模拟右键菜单）；ext对绘图的支持可以让我们通过js来绘图；大小变更控件可以让各种html元素获取可变更大小的能力。下面我们一一详细介绍。  **一、Ext.menu.Menu 菜单**  **1.菜单的定义**  下面我们来定义一个菜单，它包括一个下拉列表，还有颜色选择器，日期选择器等菜单项。  [Js]  //下定义一个下拉列表  var combo = Ext.create('Ext.form.ComboBox', {  store: new Ext.data.ArrayStore({  fields: ['id', 'name'],  data: [[1, "张三"], [2, "李四"], [3, "王五"], [4, "赵六"]]  }),  displayField: 'name',  typeAhead: true,  mode: 'local',  triggerAction: 'all',  emptyText: '请选择..',  selectOnFocus: true,  width: 135  });  //这里是菜单的定义  var menu = Ext.create('Ext.menu.Menu', {  id: 'mainMenu',  style: {  overflow: 'visible'  },  items: [  combo,  {  text: '复选框',  checked: true  }, '-', {  text: '单选子菜单',  menu: {  items: [  '<b>请选择一个人名</b>',  {  text: '张三',  checked: true,  group: 'theme'  }, {  text: '李四',  checked: false,  group: 'theme',  checkHandler: function () {  Ext.MessageBox.alert("消息", "李四被选择！")  }  }, {  text: '王五',  checked: false,  group: 'theme'  }, {  text: '赵六',  checked: false,  group: 'theme'  }  ]  }  }, {  text: '请选择一个日期',  menu: Ext.create('Ext.menu.DatePicker', {  handler: function (dp, date) {  Ext.MessageBox.alert('消息', '你选择了:' + date.format('Y-m-d'));  }  })  }, {  text: '选择一个颜色',  menu: Ext.create('Ext.menu.ColorPicker', {  handler: function (cm, color) {  Ext.MessageBox.alert('消息', '你选择了：' + color);  }  })  }  ]  });  **2.把菜单附加到面板工具栏**  我们定义一个面板，然后通过刚刚定义的菜单实现菜单栏。  [Js]  //把菜单附加到工具栏上  var panel = new Ext.Panel({  renderTo: 'div1',  width: 600,  height: 250,  collapsible: true,  layout: 'fit',  title: '演示工具栏',  tbar: [{ text: "菜单", menu: menu}]  });  下面看看展示效果：  http://pic002.cnblogs.com/images/2011/99414/2011122523531315.jpg  **2.实现右键菜单**  还是用第一步定义的菜单，现在我们通过右键事件的方式实现右键菜单：  [Js]  //定义右键菜单  Ext.getDoc().on('contextmenu', function (e, o) {  e.preventDefault();  menu.showAt(e.getXY());  //第二种写法：  //e.stopEvent();  //menu.showAt([e.getPageX(), e.getPageY()]);  });  单击右键，效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122523540216.jpg  **3.在ListView中定义右键菜单**  这里我们借用第11篇的ListView,我们要实现当数据行右键单击时，弹出菜单的效果。  [Js]  var store = new Ext.data.JsonStore({  fields: [  { name: 'IntData', type: 'int' },  { name: 'StringData', type: 'string' },  { name: 'TimeData', type: 'date' }  ],  proxy: {  type: 'ajax',  url: 'ListView1Json',  reader: {  type: 'json',  root: 'rows'  }  },  sortInfo: { field: 'IntData', direction: 'DESC' }  });  store.load();  var listView = Ext.create('Ext.ListView', {  renderTo: "div1",  store: store,  multiSelect: true,  emptyText: '无数据',  reserveScrollOffset: true,  hideHeaders: false,  columns: [{  header: "IntData",  dataIndex: 'IntData'  }, {  header: "StringData",  dataIndex: 'StringData'  }, {  header: "TimeData",  dataIndex: 'TimeData',  align: 'right',  xtype: 'datecolumn',  format: 'm-d h:i a'  }],  viewConfig: {  stripeRows: true,  listeners: {  itemcontextmenu: function (view, rec, node, index, e) {  e.stopEvent();  menu.showAt(e.getXY());  return false;  }  }  }  });  在ListView中单击右键，效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122523544379.jpg  **二、Ext.draw.Component 绘图**  Ext支持通过js的方式绘图，可以绘制的包括基本图形圆形、矩形等，还可以描绘路径，因为它支持SVG路径语法。  **1.实现文本绘图**  [Js]  Ext.create('Ext.draw.Component', {  renderTo: Ext.getBody(),  viewBox: false,  draggable: {  constrain: true, //允许拖动  constrainTo: Ext.getBody()  },  floating: true,  autoSize: true,  items: [{  type: 'text',  text: '图形化的文本',  fill: 'green',  font: '16px Arial',  rotate: {  degrees: 45  }  }]  });  通过上面的代码，我们可以展示出图片式文本，效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122523550835.jpg  **2.基本图形，路径绘图**  我们先通过基本图形绘制一个圆形，一个长方形，最后通过路径语法绘制一个等腰三角形：  [Js]  var drawComponent = Ext.create('Ext.draw.Component', {  viewBox: false,  items: [{  type: 'circle', //园  fill: '#79BB3F',  radius: 100,  x: 100,  y: 100  }, {  type: 'rect', //矩形  width: 50,  height: 30,  fill: '#f00',  x: 0,  y: 0  }, {  type: "path",  path: "M100 0 L150 50 L200 0 Z", //路径  "stroke-width": "1",  stroke: "#000",  fill: "blue"  }]  });  Ext.create('Ext.Window', {  width: 230,  height: 250,  layout: 'fit',  items: [drawComponent]  }).show();  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122523562080.jpg  **三、Ext.resizer.Resizer 大小变更**  官方示例上作者说，他奶奶都会写这段代码，所以我也不想多讲了，应该很简单。控件的实现方式如下：  [Js]  Ext.onReady(function () {  var basic = Ext.create('Ext.resizer.Resizer', {  target: 'basic',  pinned: true, //是否显示调节边框  width: 200,  height: 100,  minWidth: 100,  minHeight: 50,  dynamic: true, //动态设置  preserveRatio: true, //当变更大小时，长宽比例是否固定  heightIncrement: 20, //单次变更，高度变更值  widthIncrement: 20,  transparent: false //是否彻底隐藏调节边框  });  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122523564762.jpg |
|  |

|  |
| --- |
| posts - 29, comments - 268, trackbacks - 0, articles - 0 |
| [**第十四章 layout 布局**](http://www.cnblogs.com/lipan/archive/2011/12/27/2302719.html) 本篇讲解Ext另一个重要的概念：布局。一般的容器类控件都是通过配置项items添加子控件的，这些子控件相对于父控件怎么定位呢，这里就要用到布局。某些容器类控件，它本身默认就集成了一种布局方式，例如比较典型的是：Ext.container.Viewport 布局控件，它其实就是一个border布局的容器，还有Ext.form.Panel、Ext.tab.Panel等。本节我们系统的分析各种布局方式。  **一、absolute**  这种方式的布局可以对子元素相对于父级容器控件进行绝对定位，它包含了x、y两个配置项用于定位。  我们来看看一个例子：  [Js]  //absolute  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div1',  width: 400,  height: 300,  layout: 'absolute',  items: [{  title: '面板1',  xtype: "panel",  html: "子元素1",  width: 200,  height: 100,  x: 50,  y: 50  }, {  title: '面板2',  xtype: "panel",  html: "子元素2",  width: 200,  height: 100,  x: 100,  y: 80  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700401322.jpg  **二、accordion**  有的js插件里面accordion都是一个ui控件，但是Ext是通过布局的方式实现的，我们可以用面板控件作为它的折叠项，并且还可以用js来翻动活动项。  [Js]  //accordion  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div2',  width: 400,  height: 300,  layout: 'accordion',  items: [{  tools: [{ type: 'gear', handler: function () {  Ext.Msg.alert('提示', '配置按钮被点击。');  }  }, { type: 'refresh'}],  title: '面板1',  xtype: "panel",  html: "子元素1"  }, {  title: '面板2',  xtype: "panel",  html: "子元素2"  }, {  id: 'panel3',  title: '面板3',  xtype: "panel",  html: "子元素3"  }]  });  Ext.create("Ext.Button", {  renderTo: 'div2',  text: "打开第三页",  handler: function () {  Ext.getCmp('panel3').expand(true);  }  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700405623.jpg  **三、anchor**  这个布局就是表单面板默认支持的，每一项占据一行，支持用anchor配置项分配各个子项的高度和宽度。为百分比时表示当前大小占父容器的百分比，为数字的时一般为负数，表示父容器的值减去差值，剩下的为子项的大小。  [Js]  //anchor  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div3',  width: 400,  height: 300,  layout: 'anchor',  items: [{  tools: [{ type: 'gear', handler: function () {  Ext.Msg.alert('提示', '配置按钮被点击。');  }  }, { type: 'refresh'}],  title: '面板1',  xtype: "panel",  html: "子元素1",  anchor: '80% 20%'  }, {  title: '面板2',  xtype: "panel",  html: "子元素2",  anchor: '-50 -200'  }, {  title: '面板3',  xtype: "panel",  html: "子元素3",  anchor: '100% 30%'  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700414284.jpg  **四、border**  这个布局可以定义东南西北四个方向的子元素，还有一个居中的子元素，一般用它来做页面整页布局，所以Ext.container.Viewport默认就支持了这个布局方式。  [Js]  //border  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div4',  width: 400,  height: 300,  layout: 'border',  defaults: {  split: true, //是否有分割线  collapsible: true, //是否可以折叠  bodyStyle: 'padding:15px'  },  items: [{  region: 'north', //子元素的方位：north、west、east、center、south  title: '北',  xtype: "panel",  html: "子元素1",  height: 70  }, {  region: 'west',  title: '西',  xtype: "panel",  html: "子元素2",  width: 100  }, {  region: 'east',  title: '东',  xtype: "panel",  html: "子元素2",  width: 100  }, {  region: 'center',  title: '主体',  xtype: "panel",  html: "子元素3"  }, {  region: 'south',  title: '南',  xtype: "panel",  html: "子元素4",  height: 70  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700422215.jpg  **五、card**  这个布局可以像卡片一样的切换每个子元素，各个子元素都会独占父元素的容器空间。我们可以定义翻页按钮来控制当前处于活动状态的子元素。  [Js]  //card  var cardNav = function (incr) {  var l = Ext.getCmp('cardPanel').getLayout();  var i = l.activeItem.id.split('card')[1];  var next = parseInt(i, 10) + incr;  l.setActiveItem(next);  Ext.getCmp('cardPrev').setDisabled(next === 0);  Ext.getCmp('cardNext').setDisabled(next === 2);  };  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div5',  width: 400,  height: 300,  layout: 'card',  activeItem: 1, //默认活动项  id: 'cardPanel',  items: [{  id: 'card0',  title: '面板1',  xtype: "panel",  html: "子元素1"  }, {  id: 'card1',  title: '面板2',  xtype: "panel",  html: "子元素2"  }, {  id: 'card2',  title: '面板3',  xtype: "panel",  html: "子元素3"  }],  bbar: ['->', {  id: 'cardPrev',  text: '« 前一页',  handler: Ext.Function.bind(cardNav, this, [-1])  }, {  id: 'cardNext',  text: '后一页 »',  handler: Ext.Function.bind(cardNav, this, [1])  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700430166.jpg  **六、column**  这个布局把子元素按照列进行划分。  [Js]  //column  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div6',  width: 400,  height: 300,  layout: 'column',  defaults: { //设置没一列的子元素的默认配置  layout: 'anchor',  defaults: {  anchor: '100%'  }  },  items: [{  columnWidth: 4 / 10, //设置列的宽度  items: [{  title: '面板1',  border: false,  html: '子元素1'  }, {  title: '面板2',  border: false,  html: '子元素2'  }]  }, {  width: 120,  items: [{  title: '面板3',  border: false,  html: '子元素3'  }]  }, {  columnWidth: .40,  items: [{  title: '面板4',  border: false,  html: '子元素4'  }]  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700435234.jpg  **七、fit**  这个布局下子元素会独占全部的容器空间，一般用于只有一个子项的情况。  [Js]  //fit  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div7',  width: 400,  height: 300,  layout: 'fit',  items: [{  title: '面板',  html: '子元素',  border: false  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700442173.jpg  **八、table**  这个布局用表格定位的方式去组织子元素，我们可以像表格一样设置rowspan和colspan。  [Js]  //table  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div8',  width: 400,  height: 300,  layout: {  type: 'table',  columns: 4  },  defaults: { frame: true, width: 70, height: 50 },  items: [  { html: '元素1', rowspan: 3, height: 150 },  { html: '元素2', rowspan: 2, height: 100 },  { html: '元素3' },  { html: '元素4' },  { html: '元素5', colspan: 2, width: 140 },  { html: '元素6' },  { html: '元素7' },  { html: '元素8' }  ]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700444998.jpg  **九、vbox**  这个布局把所有的子元素按照纵向排成一列。  [Js]  //vbox  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div9',  width: 400,  height: 300,  layout: {  type: 'vbox',  pack: 'start', //纵向对齐方式 start：从顶部；center：从中部；end：从底部  align: 'stretchmax' //对齐方式 center、left、right：居中、左对齐、右对齐；stretch：延伸；stretchmax：以最大的元素为标准延伸  },  defaults: {  xtype: 'button'  },  items: [{  text: '小按钮',  flex: 1 //表示当前子元素尺寸所占的均分的份数。  }, {  xtype: 'tbspacer', //插入的空填充  flex: 3  },  {  text: '中按钮',  scale: 'medium'  }, {  text: '大按钮',  width: 120,  scale: 'large',  flex: 1  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700451718.jpg  **十、hbox**  跟vbox类似，只不过变成了横向的。  [Js]  //hbox  Ext.create('Ext.Panel', {  title: '容器面板',  renderTo: 'div10',  width: 400,  height: 300,  layout: {  type: 'hbox',  pack: 'end',  align: 'middle' //对齐方式 top、middle、bottom：顶对齐、居中、底对齐；stretch：延伸；stretchmax：以最大的元素为标准延伸  },  defaults: {  xtype: 'button'  },  items: [{  text: '小按钮'  },{  text: '中按钮',  scale: 'medium'  }, {  text: '大按钮',  width: 120,  scale: 'large'  }]  });  效果如下：  http://pic002.cnblogs.com/images/2011/99414/2011122700455239.jpg  作者：李盼（Lipan） 出处：[[Lipan]](http://www.cnblogs.com/lipan/) （<http://www.cnblogs.com/lipan/>） 版权声明：本文的版权归作者与博客园共有。转载时须注明本文的详细链接，否则作者将保留追究其法律责任。 |
|  |