

序 言

ExtJS 是一个很不错的 Ajax 框架，可以用来开发带有华丽外观的富客户端应用，使得我们的 b/s 应用更加具有活力及生命力。ExtJS 是一个用 javascript 编写，与后台技术无关的前端 ajax 框架。因此，可以把 ExtJS 用在 .Net、Java、Php 等各种开发语言开发的应用中。最近我们在几个应用都使用到了 ExtJS，对公司以前开发的一个 OA 系统也正在使用 ExtJS2.0 进行改造，使得整个系统在用户体验上有了非常大的变化。本教程记录了前段时间本人学习 ExtJS 的一些心得及小结，希望能帮助正在学习或准备学习 ExtJS 的朋友们快速走进 ExtJS2.0 的精彩世界。

教程包括 ExtJS 的新手入门、组件体系结构及使用、ExtJS 中各控件的使用方法及示例应用等，是一个非常适合新手的 ExtJS 入门教程。本教程主要是针对 ExtJS2.0 进行介绍，全部代码、截图等都是基于 ExtJS2.0。

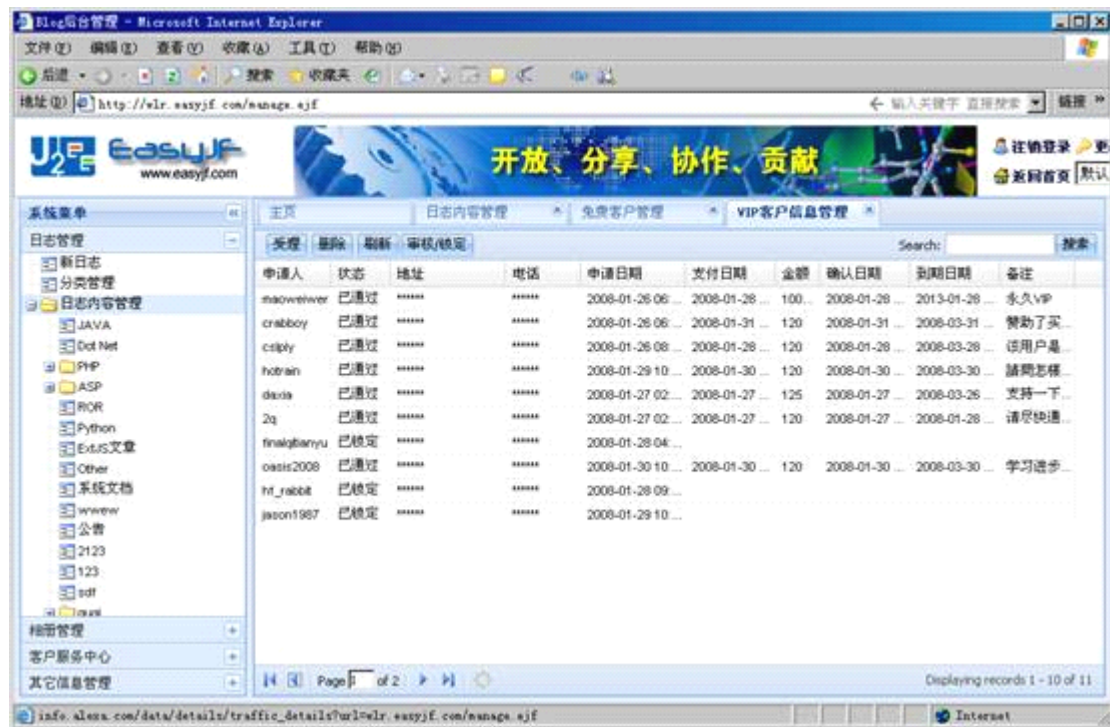
在学习了本教程后，可以下载 wlr.easyjf.com 这个基于 ExtJS2.0 开发的单用户 Blog 系统的源代码，这个系统是我们团队中的 williamraym 与大峡等人开发的一个演示系统，系统源码整体质量比较高，通过学习这套源代码相邻一定能提高您 ExtJS 的综合水平。

本教程比较适合 ExtJS 的新手作为入门教程及手册使用，并在我的博客 <http://www.easyjf.com/blog/lengyu/> 上进行发布；应一些朋友的要求，根据本教程的写作思路，我还编写了比本教程更为详细的《ExtJS 实用开发指南》，包含详细的 ExtJS 框架使用方法、各个控件详细配置参数、属性、方法及事件介绍，与服务器端集成及一个完整的示例应用系统介绍等内容，适合想深入学习 ExtJS 或正在使用 ExtJS 进行开发朋友们使用。该《指南》当前在 wlr.easyjf.com 作为 VIP 文档发布，供该站的 VIP 用户阅读及下载。凡是购买了《ExtJS 实用开发指南》文档的 VIP 用户，都可以在该指南印刷版出版后均会免费得到相应的印刷版本。

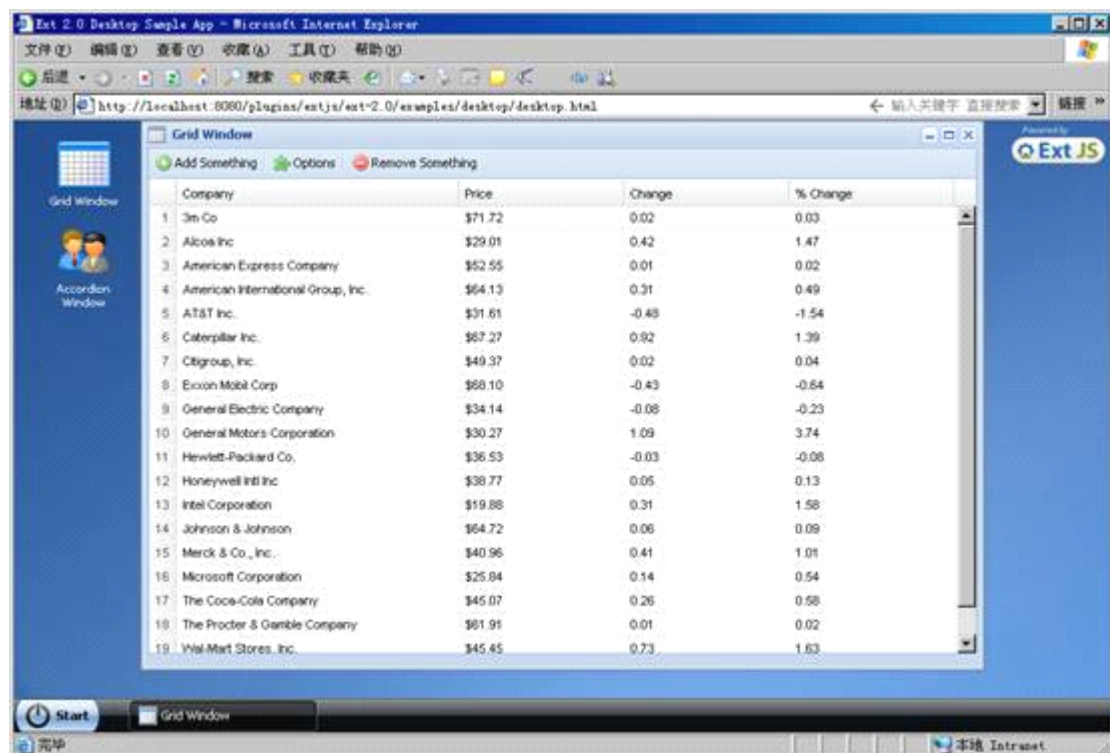
最后，希望广大网友把阅读本教程中的发现的错误、不足及建议等反馈给我们，让我们一起共同学习、共同进步，下面让我们一起进入精彩的 ExtJS 世界吧。

第一章、ExtJS简介

ExtJS是一个Ajax框架，是一个用javascript写的，用于在客户端创建丰富多彩的web应用程序界面。ExtJS可以用来开发RIA也即富客户端的AJAX应用，下面是一些使用ExtJS开发的应用程序截图：



(wlr的blog应用)



| Array Grid | | | | |
|-------------------------------------|---------|--------|----------|--------------|
| Company | Price | Change | % Change | Last Updated |
| 3m Co | \$71.72 | 0.02 | 0.03% | 09/01/2008 |
| Alcoa Inc | \$29.01 | 0.42 | 1.47% | 09/01/2008 |
| Altria Group Inc | \$83.81 | 0.28 | 0.34% | 09/01/2008 |
| American Express Company | \$52.55 | 0.01 | 0.02% | 09/01/2008 |
| American International Group, Inc. | \$64.13 | 0.31 | 0.49% | 09/01/2008 |
| AT&T Inc. | \$31.61 | -0.48 | -1.54% | 09/01/2008 |
| Boeing Co. | \$75.43 | 0.53 | 0.71% | 09/01/2008 |
| Caterpillar Inc. | \$67.27 | 0.92 | 1.39% | 09/01/2008 |
| Citigroup, Inc. | \$49.37 | 0.02 | 0.04% | 09/01/2008 |
| E.I. du Pont de Nemours and Company | \$40.48 | 0.51 | 1.28% | 09/01/2008 |
| Exxon Mobil Corp | \$68.10 | -0.43 | -0.64% | 09/01/2008 |
| General Electric Company | \$34.14 | -0.08 | -0.23% | 09/01/2008 |
| General Motors Corporation | \$30.27 | 1.09 | 3.74% | 09/01/2008 |
| Hewlett-Packard Co. | \$36.53 | -0.03 | -0.08% | 09/01/2008 |

(ExtJS的表格控件)



(不同主题的ExtJS弹出框效果)

ExtJS是一个用javascript写的，主要用于创建前端用户界面，是一个与后台技术无关的前端ajax框架。因此，可以把ExtJS用在.Net、Java、Php等各种开发语言开发的应用中。

ExtJs 最开始基于 YUI 技术，由开发人员 Jack Slocum 开发，通过参考 Java Swing 等机制来组织可视化组件，无论从 UI 界面上 CSS 样式的应用，到数据解析上的异常处理，都可算是一款不可多得的 JavaScript 客户端技术的精品。

第二章、开始 ExtJS

2.1 获得 ExtJS

要使用 ExtJS，那么首先要得到 ExtJS 库文件，该框架是一个开源的，可以直接从官方网站下载，网址 <http://extjs.com/download>，进入下载页面可以看到大致如图 xxx 所示的内容，可以选择选择 1.1 或 2.0 版本，本教程使用的 2.0 版本。

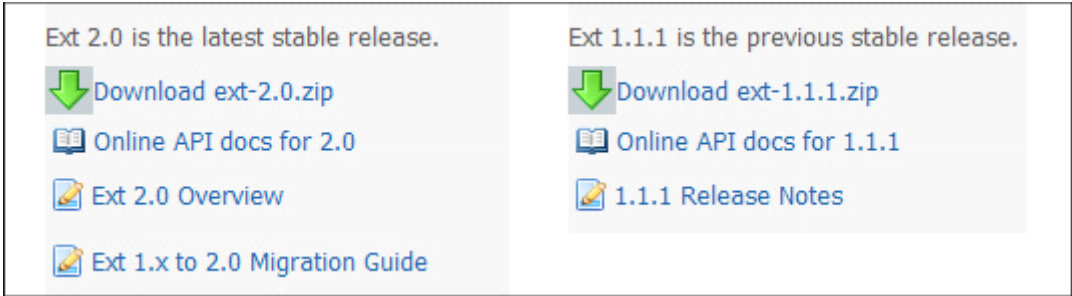


图 1-1 ExtJs 不同版本下载选择页面

单击上图中的【Download ext-2.0.zip】超链接进行下载，把下载得到的 ZIP 压缩文件解压缩到【D:\ExtCode】目录下，可以得到如如图 1-2 所示的内容。

| Name ^ | Size | Type |
|-------------------|--------|---------------------|
| adapter | | File Folder |
| build | | File Folder |
| docs | | File Folder |
| examples | | File Folder |
| resources | | File Folder |
| source | | File Folder |
| CHANGES.txt | 4 KB | Text Document |
| ext-all.js | 496 KB | JScript Script File |
| ext-all-debug.js | 901 KB | JScript Script File |
| ext-core.js | 85 KB | JScript Script File |
| ext-core-debug.js | 160 KB | JScript Script File |
| INCLUDE_ORDER.txt | 1 KB | Text Document |
| LICENSE.txt | 3 KB | Text Document |

图 1-2 ExtJS 发布包目录

- adapter: 负责将里面提供第三方底层库（包括 Ext 自带的底层库）映射为 Ext 所支持的底层库。
- build: 压缩后的 ext 全部源码(里面分类存放)。
- docs: API 帮助文档。
- exmaples: 提供使用 ExtJs 技术做出的小实例。
- resources: Ext UI 资源文件目录，如 CSS、图片文件都存放在这里面。
- source: 无压缩 Ext 全部的源码(里面分类存放) 遵从 Lesser GNU （LGPL） 开源的协议。

Ext-all.js: 压缩后的 Ext 全部源码。
 ext-all-debug.js: 无压缩的 Ext 全部的源码(用于调试)。
 ext-core.js: 压缩后的 Ext 的核心组件, 包括 sources/core 下的所有类。
 ext-core-debug.js: 无压缩 Ext 的核心组件, 包括 sources/core 下的所有类。

2.2、应用 ExtJS

应用 extjs 需要在页面中引入 extjs 的样式及 extjs 库文件, 样式文件为 resources/css/ext-all.css, extjs 的 js 库文件主要包含两个, adapter/ext/ext-base.js 及 ext-all.js, 其中 ext-base.js 表示框架基础库, ext-all.js 是 extjs 的核心库。adapter 表示适配器, 也就是说可以有多种适配器, 因此, 可以把 adapter/ext/ext-base.js 换成 adapter/jquery/ext-jquery-adapter.js, 或 adapter/prototype/ext-prototype-adapter.js 等。因此, 要使用 ExtJS 框架的页面中一般包括下面几句:

```
<script type="text/javascript" src="extjs/adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="extjs/ext-all.js"></script>
```

在 ExtJS 库文件及页面内容加载完后, ExtJS 会执行 Ext.onReady 中指定的函数, 因此可以用, 一般情况下每一个用户的 ExtJS 应用都是从 Ext.onReady 开始的, 使用 ExtJS 应用程序的代码大致如下:

```
<script>
function fn()
{
    alert('ExtJS库已加');
}
Ext.onReady(fn);
</script>
```

fn 也可以写成一个匿名函数的形式, 因此上面的代码可以改成下面的形式:

```
<script>
function fn()
{
    alert('ExtJS库已加载!');
}
Ext.onReady(function ()
{
    alert('ExtJS库已加载!');
}
);
```

```
</script>
```

2.3、ExtJS 版的 HelloWorld

下面我们写一个最简单的 ExtJS 应用，在 hello.html 文件中输入下面的代码：

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>ExtJS</title>
<link rel="stylesheet" type="text/css" href="extjs/resources/css/ext-all.css" />
<script type="text/javascript" src="extjs/adaptor/ext/ext-base.js"></script>
<script type="text/javascript" src="extjs/ext-all.js"></script>
<script>
Ext.onReady(function()
{
    Ext.MessageBox.alert("hello","Hello,easyjf open source");
});
</script>
</head>
<body>
</body>
</html>
```

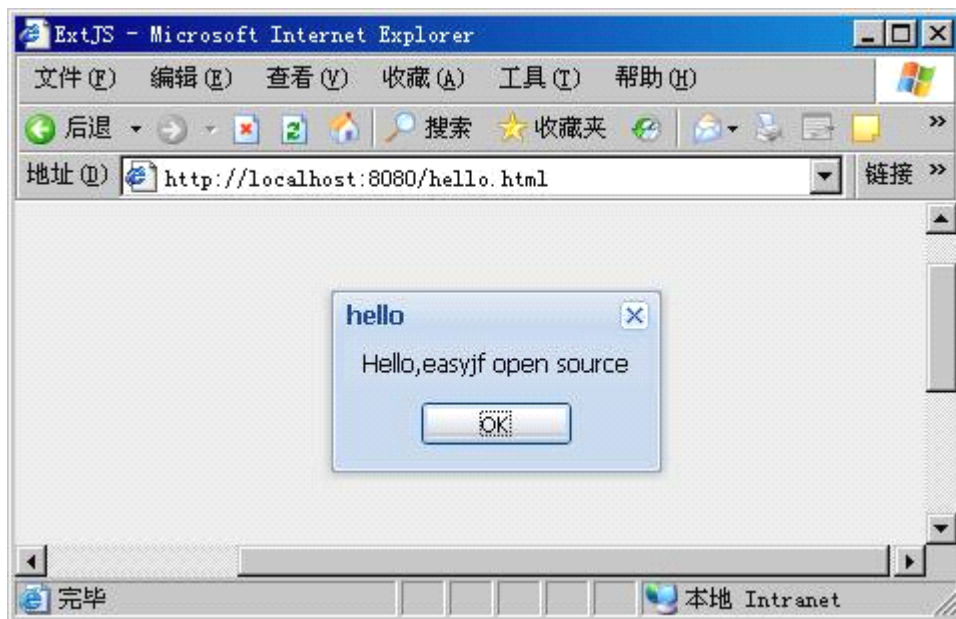
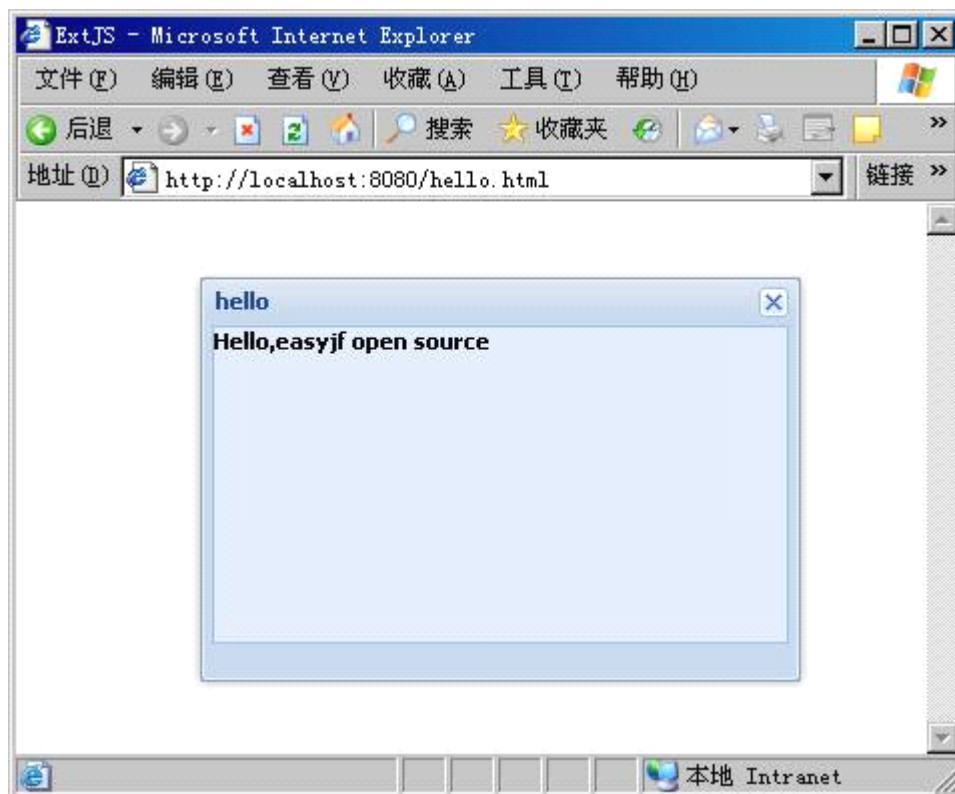


图1-11 hello.html页面效果

进一步，我们可以在页面上显示一个窗口，代码如下：

```
<script>
Ext.onReady(function()
{
    var win=new Ext.Window({title:"hello",width:300,height:200,html:'<h1>Hello,easyjf open source</h1>'});
    win.show();
});
</script>
```

在浏览 hello.html，即可得在屏幕上显示一个窗口，如图 xxx 所示。

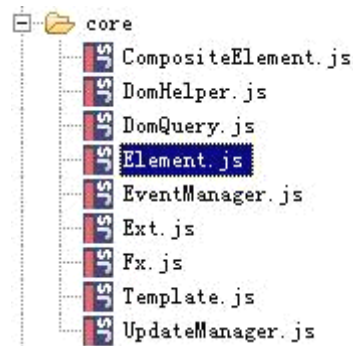


第三章 Ext 框架基础及核心简介

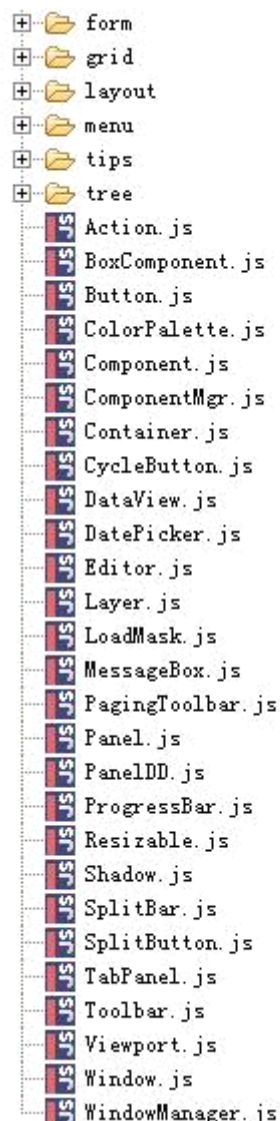
3.1、Ext 类库简介

ExtJS 由一系列类库组成，一旦页面成功加载了 ExtJS 库后，我们就可以在页面中通过 javascript 调用 ExtJS 的类及控件来实现需要的功能。ExtJS 的类库由以下几部分组成：

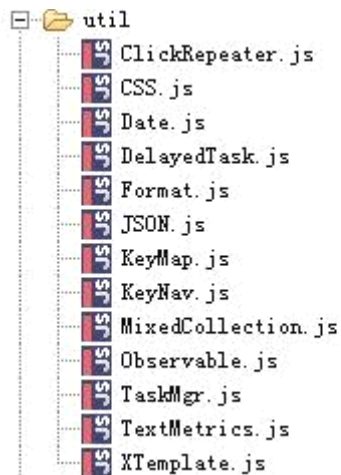
底层 API(core): 底层 API 中提供了对 DOM 操作、查询的封装、事件处理、DOM 查询器等基础的功能。其它控件都是建立在这些底层 api 的基础上，底层 api 位于源代码目录的 core 子目录中，包括 DomHelper.js、Element.js 等文件，如图 xx 所示。



控件(widgets): 控件是指可以直接在页面中创建的可视化组件，比如面板、选项板、表格、树、窗口、菜单、工具栏、按钮等等，在我们的应用程序中可以直接通过应用这些控件来实现友好、交互性强的应用程序的 UI。控件位于源代码目录的 widgets 子目录中，包含如图 xx 所示。



实用工具 Utils: Ext 提供了很多的实用工具,可以方便我们实现如数据内容格式化、JSON 数据解码或反解码、对 Date、Array、发送 Ajax 请求、Cookie 管理、CSS 管理等扩展等功能,如图所示:

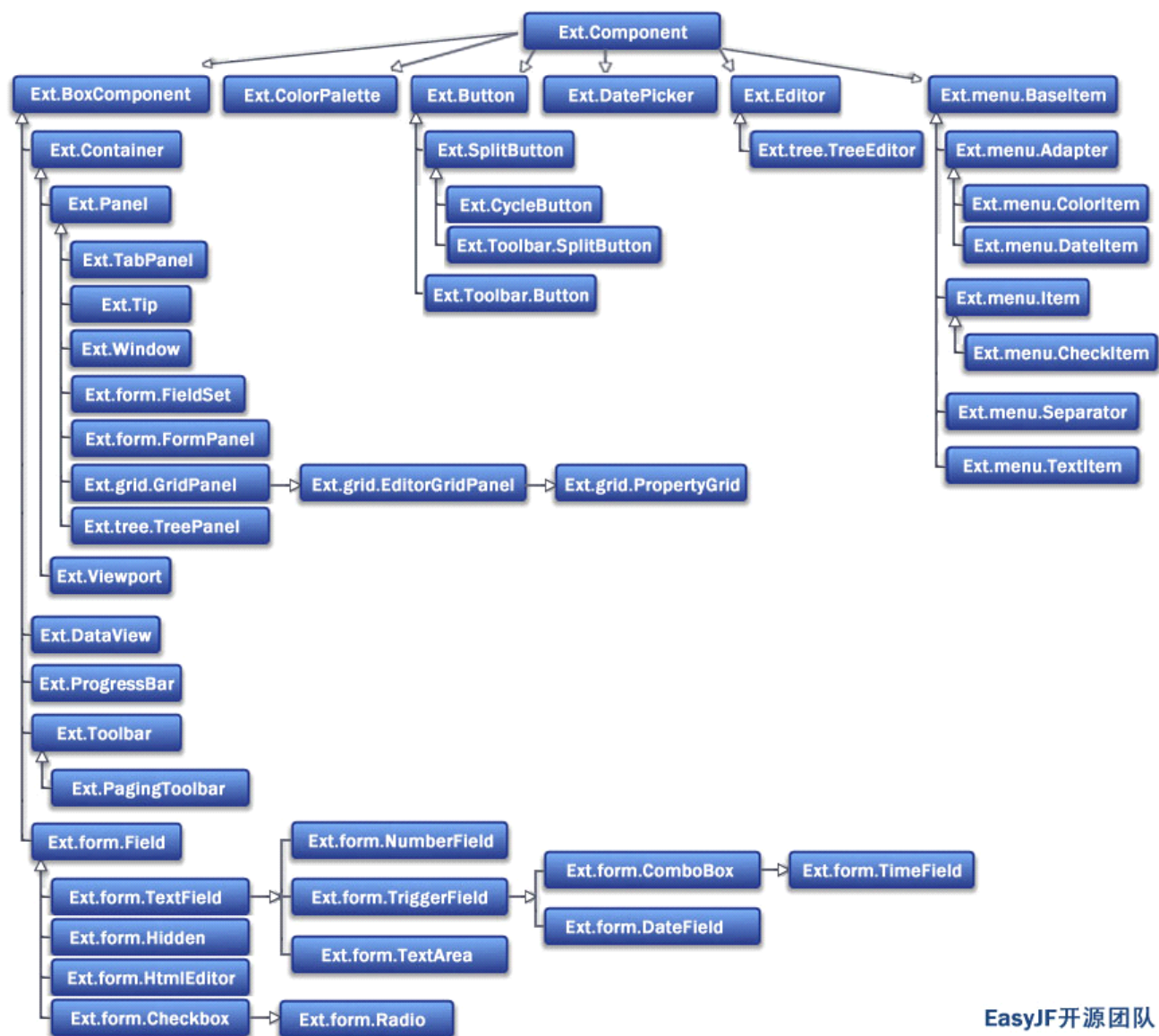


3.2、Ext 的组件

Ext2.0 对框架进行了非常大的重构,其中最重要的就是形成了一个结构及层次分明的组件体系,由这些组件形成了 Ext 的控件,Ext 组件是由 **Component** 类定义,每一种组件都有一个指定的 **xtype** 属性值,通过该值可以得到一个组件的类型或者是定义一个指定类型的组件。

ExtJS 中的组件体系由下图所示:

ExtJs 2.0组件继承结构图



EasyJF开源团队

组件大致可以分成三大类，即基本组件、工具栏组件、表单及元素组件。
基本组件有：

| xtype | Class |
|--------------|---------------------------------|
| box | Ext.BoxComponent 具有边框属性的组件 |
| Button | Ext.Button 按钮 |
| colorpalette | Ext.ColorPalette 调色板 |
| component | Ext.Component 组件 |
| container | Ext.Container 容器 |
| cycle | Ext.CycleButton |
| dataview | Ext.DataView 数据显示视图 |
| datepicker | Ext.DatePicker 日期选择面板 |
| editor | Ext.Editor 编辑器 |
| editorgrid | Ext.grid.EditorGridPanel 可编辑的表格 |

| | |
|-------------|---------------------------------|
| grid | Ext.grid.GridPanel 表格 |
| paging | Ext.PagingToolbar 工具栏中的间隔 |
| panel | Ext.Panel 面板 |
| progress | Ext.ProgressBar 进度条 |
| splitbutton | Ext.SplitButton 可分裂的按钮 |
| tabpanel | Ext.TabPanel 选项面板 |
| treepanel | Ext.tree.TreePanel 树 |
| viewport | Ext.ViewPort 视图 |
| window | Ext.Window 窗口 |
| 工具栏组件有 | |
| toolbar | Ext.Toolbar 工具栏 |
| tbbutton | Ext.Toolbar.Button 按钮 |
| tbfill | Ext.Toolbar.Fill 文件 |
| tbitem | Ext.Toolbar.Item 工具条项目 |
| tbseparator | Ext.Toolbar.Separator 工具栏分隔符 |
| tbspacer | Ext.Toolbar.Spacer 工具栏空白 |
| tbsplit | Ext.Toolbar.SplitButton 工具栏分隔按钮 |
| tbtext | Ext.Toolbar.TextItem 工具栏文本项 |
| 表单及字段组件包含 | |
| form | Ext.FormPanel Form 面板 |
| checkboxbox | Ext.form.Checkbox checkbox 录入框 |
| combo | Ext.form.ComboBox combo 选择项 |
| datefield | Ext.form.DateField 日期选择项 |
| field | Ext.form.Field 表单字段 |
| fieldset | Ext.form.FieldSet 表单字段组 |
| hidden | Ext.form.Hidden 表单隐藏域 |
| htmleditor | Ext.form.HtmlEditor html 编辑器 |
| numberfield | Ext.form.NumberField 数字编辑器 |
| radio | Ext.form.Radio 单选按钮 |
| textarea | Ext.form.TextArea 区域文本框 |
| textfield | Ext.form.TextField 表单文本框 |
| timefield | Ext.form.TimeField 时间录入项 |
| trigger | Ext.form.TriggerField 触发录入项 |

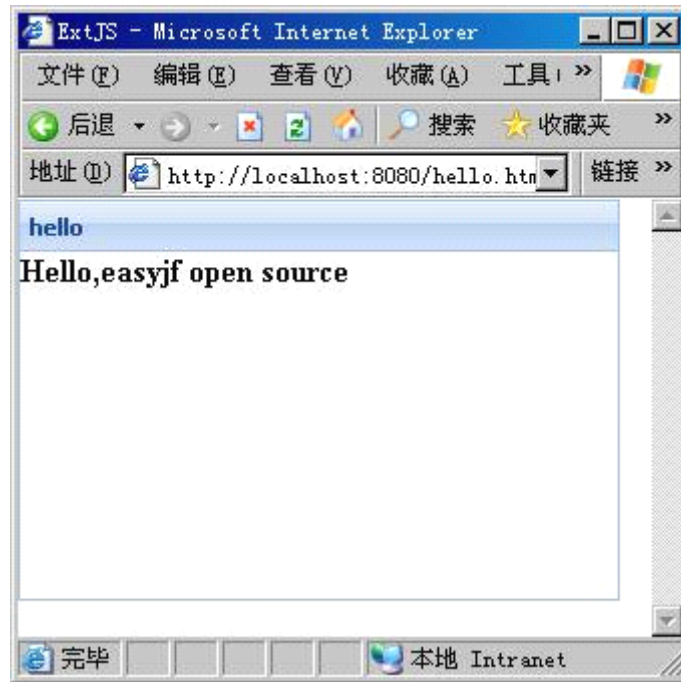
3.3、组件的使用

组件可以直接通过 new 关键字来创建，比如控件一个窗口，使用 new Ext.Window()，创建一个表格则使用 new Ext.GridPanel()。当然，除了一些普通的组件以外，一般都会在构造函数中通过传递构造参数来创建组件。

组件的构造函数中一般都可以包含一个对象，这个对象包含创建组件所需要的配置属性及值，组件根据构造函数中的参数属性值来初始化组件。如下面的例子：

```
var obj={title:"hello",width:300,height:200,html:'Hello,easyjf open source'};  
var panel=new Ext.Panel(obj); panel.render("hello");  
<div id="hello">&nbsp;  </div>
```

运行上面的代码可以实现如下图所示的结果:



可以省掉变量 obj, 直接写成如下的形式:

```
var panel=new Ext.Panel({title:"hello",width:300,height:200,html:'<h1>Hello,easyjf open source</h1>'});  
panel.render("hello");
```

render 方法后面的参数表示页面上的 div 元素 id, 也可以直接在参数中通过 renderTo 参数来省略手动调用 render 方法, 只需要在构造函数的参数中添加一个 renderTo 属性即可, 如下:

```
New Ext.Panel({renderTo:"hello",title:"hello",width:300,height:200,html:'<h1>Hello,easyjf open source</h1>'});
```

对于容器中的子元素组件, 都支持延迟加载的方式创建控件, 此时可以直接通过在需要父组件的构造函数中, 通过给属性 items 传递数组方式实现构造。如下面的代码:

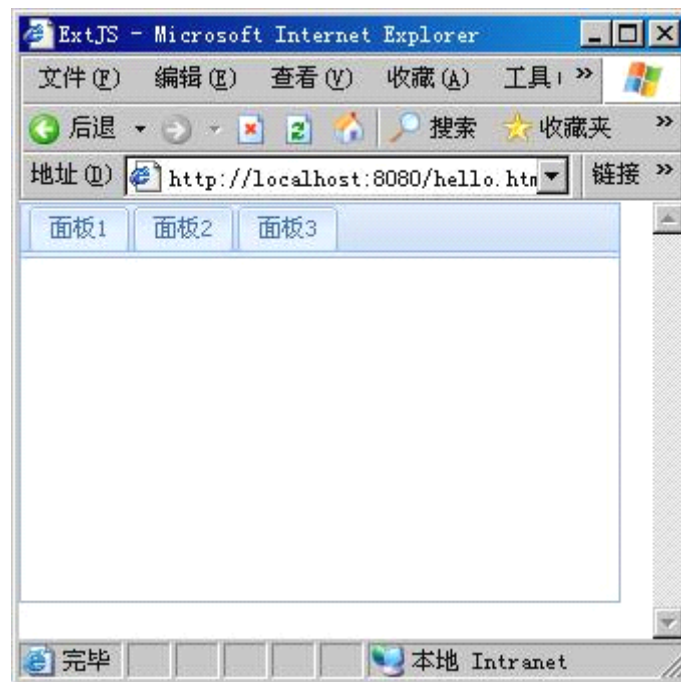
```
var panel=new Ext.TabPanel({width:300,height:200,items:[  
    {title:"面板1",height:30},{title:"面板2",height:30},{title:"面板3",height:30}]});panel.render("hello");
```

注意中括号中加粗部份的代码, 这些代码定义了 TabPanel 这个容器控件中的子元素,

这里包括三个面板。上面的代码与下面的代码等价：

```
var panel=new Ext.TabPanel({width:300,height:200,items:[new Ext.Panel({title:"面板1",height:30}),new Ext.Panel({title:"面板2",height:30}),new Ext.Panel({title:"面板3",height:30})]);panel.render("hello");
```

前者不但省略掉了 new Ext.Panel 这个构造函数，最重要前者只有在初始化 TabPanel 的时候，才会创建子面板，而第二种方式则在程序一开始就会创建子面板。也就是说，前者实现的延迟加载。



3.4、组件的配置属性

在 ExtJS 中，除了一些特殊的组件或类以外，所有的组件在初始化的时候都可以在构造函数使用一个包含属性名称及值的对象，该对象中的信息也就是指组件的配置属性。

比如配置一个面板：

```
new Ext.Panel({
    title:"面板",
    html"面板内容",
    height:100}
);
```

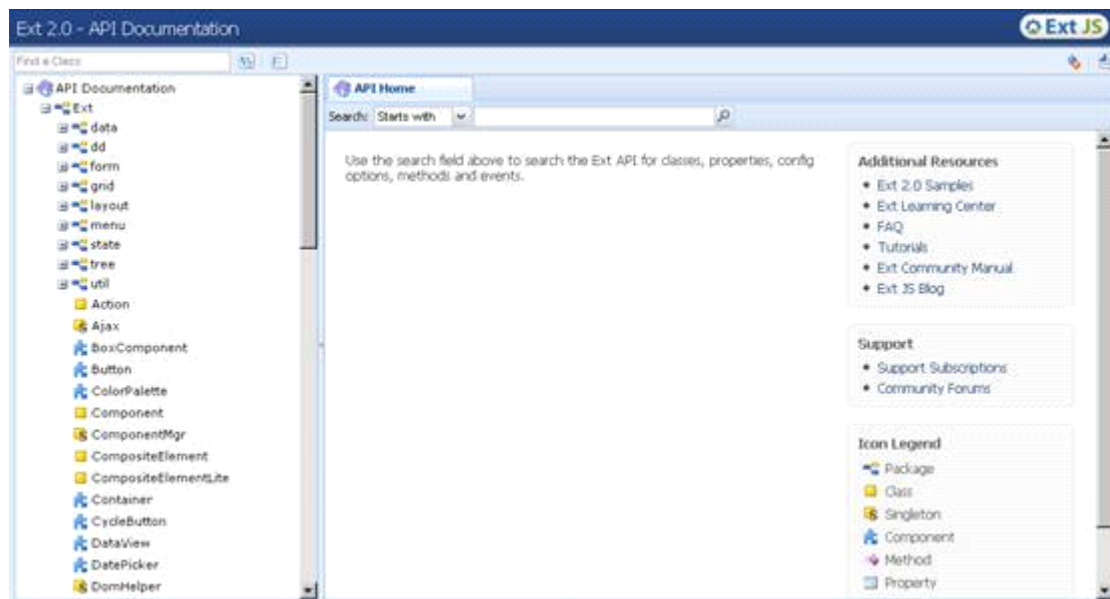
再比如创建一个按钮：


```
var b=new Ext.Button({
    text:"添加",
    pressed:true,
    heigh:30,
    handler:Ext.emptyFn
});
```

再比如创建一个 Viewport 及其中的内容:

```
new Ext.Viewport({
    layout:"border",
    items:[{region:"north",
    title:"面板",
    html:"面板内容",
    height:100},
    {region:"center",
    xtype:"grid",
    title:"学生信息管理",
    store:troe,
    cm:colM,
    store:store,
    autoExpandColumn:3
    }
    ]
});
```

每一个组件除了继承基类中的配置属性以外，还会根据需要增加自己的配置属性，另外子类中有的时候还会把父类的一些配置属性的含义及用途重新定义。学习及使用 ExtJS，其中最关键的是掌握 ExtJS 中的各个组件的配置属性及具体的含义，这些配置属性在下载下来的 ExtJS 源码文档中都有详细的说明，可以通过这个文档详细了解每一个组件的特性，如下图所示：



由于所有的组件都继承自 Ext.Component，因此在这里我们列出组件基类 Component 中的配置属性简单介绍。

| 配置属性名称 | 类型 | 简介 |
|----------------------|--------------|--|
| allowDomMove | Boolean | 当渲染这个组件时是否允许移动Dom节点（默认值为true）。 |
| applyTo | Mixed | 混合参数，表示把该组件应用指定的对象。参数可以是一节点的id，一个DOM节点或一个存在的元素或与之相对应的在document中已出现的id。当使用 applyTo ，也可以提供一个id或CSS的class名称，如果子组件允许它将尝试创建一个。如果指写 applyTo 选项，所有传递到 renderTo 方法的值将被忽略，并且目标元素的父节点将自动指定为这个组件的容器。使用 applyTo 选项后，则不需要再调用 render() 方法来渲染组件。 |
| autoShow | Boolean | 自动显示，如为true，则组件将检查所有隐藏类型的class（如：'x-hidden'或'x-hide-display'并在渲染时移除(默认为false)。 |
| cls | String | 给组件添加额外的样式信息，（默认值为""），如果想自定义组件或它的子组件的样式，这个选项是非常有用的。 |
| ctCls | String | 给组件的容器添加额外的样式信息，默认值为""。 |
| disabledClass | String | 给被禁用的组件添加额外的CSS样式信息，（默认为"x-item-disabled"）。 |
| hideMode | String | 组件的隐藏方式，支持的值有'visibility'，也就是css里的visibility，'offsets'负数偏移位置的值和'display'也就是css里的display，默认值为'display'。 |
| hideParent | Boolean | 是否隐藏父容器，该值为true时将会显示或隐藏组件的容器，false时则只隐藏和显示组件本身（默认值为false）。 |
| id | String | 组件的id，默认为一个自动分配的id。 |
| listeners | Object | 给对象配置多个事件监听器，在对象初始化会初始化这些监听器。 |
| plugins | Object/Array | 一个对象或数组，将用于增加组件的自定义功能。一个有效的组件插 |

| | | |
|--------------------|--------|--|
| | Array | 件必须包含一个init方法，该方法可以带一个Ext.Component类型参数。当组件 建立后，如果该组件包含有效的插件，将调用每一个插件的init方法，把组件传递给插件，插件就能够实现对组件的方法调用及事件应用等，从而实现对组件功 能的扩充。 |
| renderTo | Mixed | 混合数据参数，指定要渲染到节点的id，一个DOM的节点或一个已存在的容器。如果使用了这个配置选项，则组件的render()就不是必需的了。 |
| stateEvents | Array | 定义需要保存组件状态信息的事件。当指定的事件发生时，组件会保存它的状态（默认为none），其值为这个组件支持的任意event类型，包含组件自身的或自定义事件。（例如：['click','customerchange']）。 |
| stateId | String | 组件的状态ID，状态管理器使用该id来管理组件的状态信息，默认值为组件的id。 |
| style | String | 给 该 组 件 的 元 素 指 定 特 定 的 样 式 信 息 ， 有 效 的 参 数 为 Ext.Element.applyStyles 中的值。 |
| xtype | String | 指定所要创建组件的xtype，用于构造函数中没有意义。该参数用于在容器组件中创建创建子组件并延迟实例化和渲染时使用。如果是自定义的组件，则需要用Ext.ComponentMgr.registerType来进行注册，才会支持延迟实例化和渲染。 |
| el | Mixed | 相当于applyTo |

关于 ExtJS 中组件的详细使用说明，包括 Component 的属性 Properties、方法及事件详细，请参考 wlr.easyjy.com 中的 VIP 文档《ExtJS 组件 Component 详解(1)、(2)》。

3.5、Extjs 组件的事件处理

ExtJS 提供了一套强大的事件处理机制，通过这些事件处理机制来响应用户的动作、监控控件状态变化、更新控件视图信息、与服务器进行交互等等。事件统一由 Ext.EventManager 对象管理，与浏览器 W3C 标准事件对象 Event 相对应，Ext 封装了一个 Ext.EventObject 事件对象。支持事件处理的类(或接口)为 Ext.util.Observable，凡是继承该类的组件或类都支持往对象中添加事件处理及响应功能。

首先我们来看标准 html 中的事件处理，看下面的 html 代码：

```
<script>
    function a()    {
        alert('some thing');
    }
</script>
<input id="btnAlert" type="button" onclick="a();" value="alert框" />
```

点击这个按钮则会触发 `onclick` 事件，并执行 `onclick` 事件处理函数中指定的代码，这里直接执行函数 `a` 中的代码，也即弹出一个简单的信息提示框。再简单修改一下上面的代码，内容如下：

```
<script>
  function a()
  {
    alert('some thing');
  }
  window.onload=function(){
    document.getElementById("btnAlert").onclick=a;
  }
</script>
<input id="btnAlert" type="button" value="alert框" />
```

上面的代码在文档加载的时候，就直接对 `btnAlert` 的 `onclick` 赋值，非常清晰的指明了按钮 `btnAlert` 的 `onclick` 事件响应函数为 `a`，注意这里 `a` 后面不能使用括号 `()`。
ExtJS 中组件的事件处理跟上面相似，看下面的代码：

```
<script>
  function a(){
    alert('some thing');
  }
  Ext.onReady(function(){
    Ext.get("btnAlert").addListener("click",a);
  });
</script>
<input id="btnAlert" type="button" value="alert框" />
```

`Ext.get("btnAlert")` 得到一个与页面中按钮 `btnAlert` 关联的 `Ext.Element` 对象，可以直接调用该对象上的 `addListener` 方法来给对象添加事件，同样实现前面的效果。在调用 `addListener` 方法的代码中，第一个参数表示事件名称，第二个参数表示事件处理器或整个响应函数。
ExtJS 支持事件队列，可以往对象的某一个事件中添加多个事件响应函数，看下面的代码：

```
Ext.onReady(function(){
  Ext.get("btnAlert").on("click",a);
  Ext.get("btnAlert").on("click",a);
});
```

`addListener` 方法的另外一个简写形式是 `on`，由于调用了两次 `addListener` 方法，因此当点击按钮的时候会弹出两次信息。

当然，ExtJS 还支持事件延迟处理或事件处理缓存等功能，比如下面的代码：

```
Ext.onReady(function(){
    Ext.get("btnAlert").on("click",a,this,{delay:2000});
});
```

由于在调用 `addListener` 的时候传递指定的 `delay` 为 2000，因此当用户点击按钮的时候，不会马上执行事件响应函数，而是在 2000 毫秒，也就是两秒后才会弹出提示信息框。

当然，在使用 Ext 的事件时，我们一般是直接在控件上事件，每一个控件包含哪些事件，在什么时候触发，触发时传递的参数等，在 ExtJS 项目的文档中 都有较为详细的说明。比如对于所有的组件 Component，都包含一个 `beforedestroy` 事件，该事件会在 Ext 销毁这一个组件时触发，如果事件响应函数返回 `false`，则会取消组件的销毁操作。

```
Ext.onReady(function(){
    var win=new Ext.Window({
        title:"不能关闭的窗口",    height:200,width:300
    });
    win.on("beforedestroy",function(obj){
        alert("想关闭我，这是不可能的!");
        obj.show();
        return false;
    });
    win.show();});
```

由于在窗口对象的 `beforedestroy` 事件响应函数返回值为 `false`，因此执行这段程序，你会发现这个窗口将无法关闭。组件的事件监听器可以直接在组件的配置属性中直接声明，如下面的代码与前面实现的功能一样：

```
Ext.onReady(function(){
    var win=new Ext.Window({
        title:"不能关闭的窗口",
        height:200,width:300,
        listeners:{"beforedestroy":function(obj){
            alert("想关闭我，这是不可能的!");
            obj.show();    return false;
        }}
    });
    win.show();});
```

了解了 ExtJS 中的基本事件处理及使用方法，就可以在你的应用中随心所欲的进行事件相关处理操作了。

关于 ExtJS 中事件处理中作用域、事件处理原理、给自定义的组件添加事件、处理相关的 `Ext.util.Observable` 及 `Ext.EventManager` 类详细介绍，请参考 wlr.easyjif.com 中的 VIP 文档《ExtJS 中的事件处理详解》。

第四章、使用面板

4.1、Panel

面板 Panel 是 ExtJS 控件的基础，很高级控件都是在面板的基础上扩展的，还有其它大多数控件也都直接或间接有关系。应用程序的界面一般情况下是由一个一个的面板通过不同组织方式形成。

面板由以下几个部分组成，一个顶部工具栏、一个底部工具栏、面板头部、面板尾部、面板主区域几个部分组件。面板类中还内置了面板展开、关闭等功能，并提供一系列可重用的工具按钮使得我们可以轻松实现自定义的行为，面板可以放入其它任何容器中，面板本身是一个容器，他里面又可以包含各种其它组件。

面板的类名为 Ext.Panel，其 xtype 为 panel，下面的代码可以显示出面板的各个组成部分：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"面板头部header",
        width:300,
        height:200,
        html:'<h1>面板主区域</h1>',
        tbar:[{text:'顶部工具栏topToolbar'}],
        bbar:[{text:'底部工具栏bottomToolbar'}],
        buttons:[{text:"按钮位于footer"}]
    });
});
```

运行该代码，可以得到如图 xx 所示的输出结果，清楚的表示出了面板的各个组成部分。



一般情况下，顶部工具栏或底部工具栏只需要一个，而面板中一般也很少直接包含按钮，一般会把面板上的按钮直接放到工具栏上面。比如下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
```

```
renderTo:"hello",
title:"hello",
width:300,
height:200,
html:'<h1>Hello,easyjf open source!</h1>',
tbar:[{pressed:true,text:'刷新'}]
});
});
```

可以得到如图xx所示的效果，该面板包含面板Header，一个顶部工具栏及面板区域三个部分。



4.2、工具栏

面板中可以有工具栏，工具栏可以位于面板顶部或底部，Ext 中工具栏是由 **Ext.Toolbar** 类表示。工具栏上可以存放按钮、文本、分隔符等内容。面板对象中内置了很多实用的工具栏，可以直接通过面板的 `tools` 配置选项往面板头部加入预定义的工具栏选项。比如下面的代码：

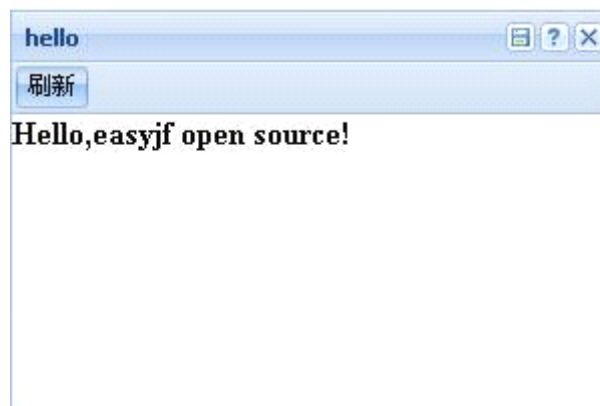
```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"hello",
        width:300,
        height:200,
        html:'<h1>Hello,easyjf open source!</h1>',
        tools:[{
            id:"save"},
            {id:"help"}
        ]
    });
});
```


```

        handler:function(){Ext.Msg.alert('help','please help me!');}
    },
    {id:"close"}],
    tbar:[{pressed:true,text:'刷新'}]
});
});

```

注意我们在Panel的构造函数中设置了tools属性的值，表示在面板头部显示三个工具栏选项按钮，分别是保存"save"、"help"、"close"三种。代码运行的效果图如下：



点击  help按钮会执行handler中的函数，显示一个弹出对话框，而点击其它的按钮不会有任何行为产生，因为没有定义他们的handler。

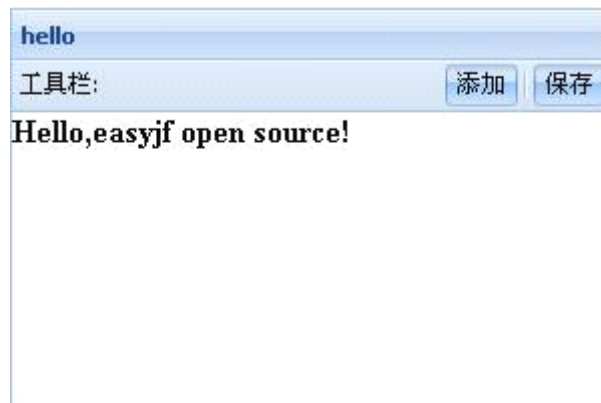
除了在面板头部加入这些已经定义好的工具栏选择按钮以外，还可以在顶部或底工具栏中加入各种工具栏选项。这些工具栏选项主要包括按钮、文本、空白、填充条、分隔符等。代码：

```

Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"hello",
        width:300,
        height:200,
        html:'<h1>Hello,easyjf open source!</h1>',
        tbar:[new Ext.Toolbar.TextItem('工具栏:'),
            {xtype:"tbfill"},
            {pressed:true,text:'添加'},
            {xtype:"tbseparator"},
            {pressed:true,text:'保存'}
        ]
    });
});

```

将会得到如图xx所示的结果：



Ext中的工具栏项目主要包含下面的类：

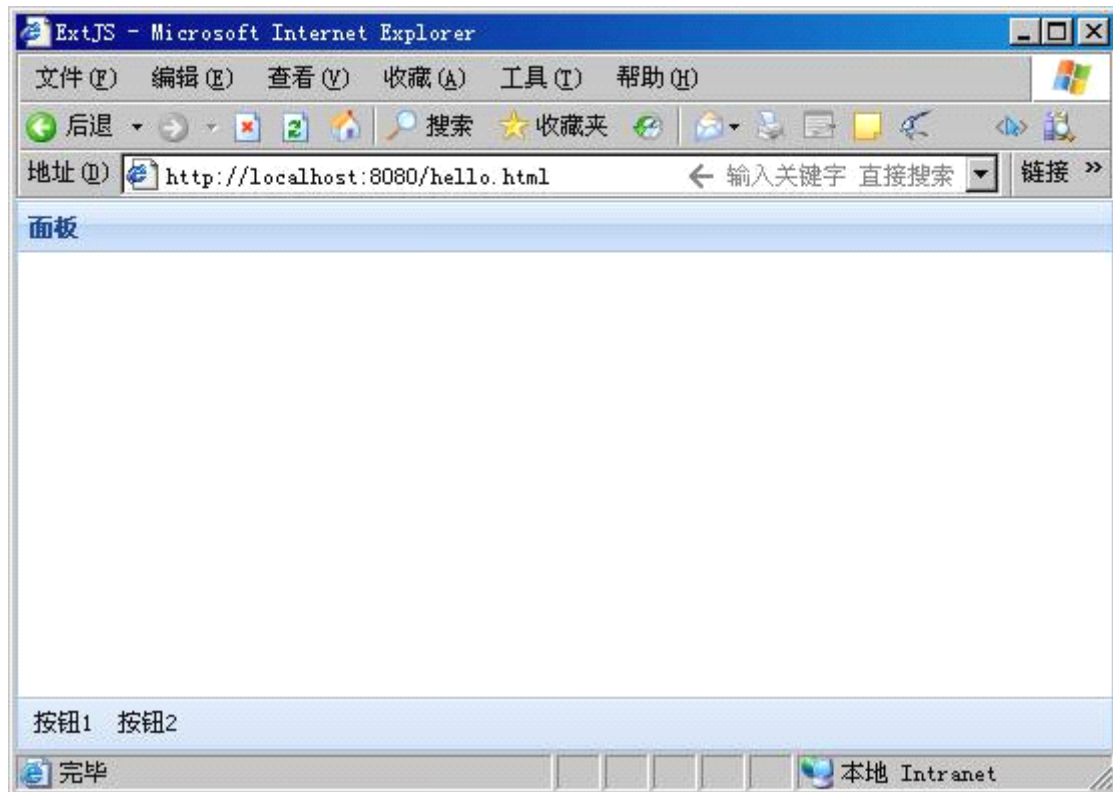
- Ext.Toolbar.Button—按钮，xtype为tbbutton
- TextItem—
- Ext.Toolbar.Fill—
- Separator—
- Spacer—
- SplitButton—

4.3、选项面板的 TabPanel

在前面的示例中，为了显示一个面板，我们需要在页面上添加一个，然后把 Ext 控件渲染到这个 div 上。**ViewPort** 代表整个浏览器显示区域，该对象渲染到页面的 **body** 区域，并会随着浏览器显示区域的大小自动改变，一个 页面中只能有一个 **ViewPort** 实例。看下面的代码：

```
Ext.onReady(function(){
    new Ext.Viewport({
        enableTabScroll:true,
        layout:"fit",
        items:[{title:"面板",
            html:"",
            bbar:[{text:"按钮1"},
                {text:"按钮2"}]
        }]
    });
});
```

运行上面的代码会得到如图 xxx 所示的输出结果。



Viewport 不需要再指定 `renderTo`, 而我们也看到 Viewport 确实填充了整个浏览器显示区域, 并会随着浏览器显示区域大小的改变而改变。

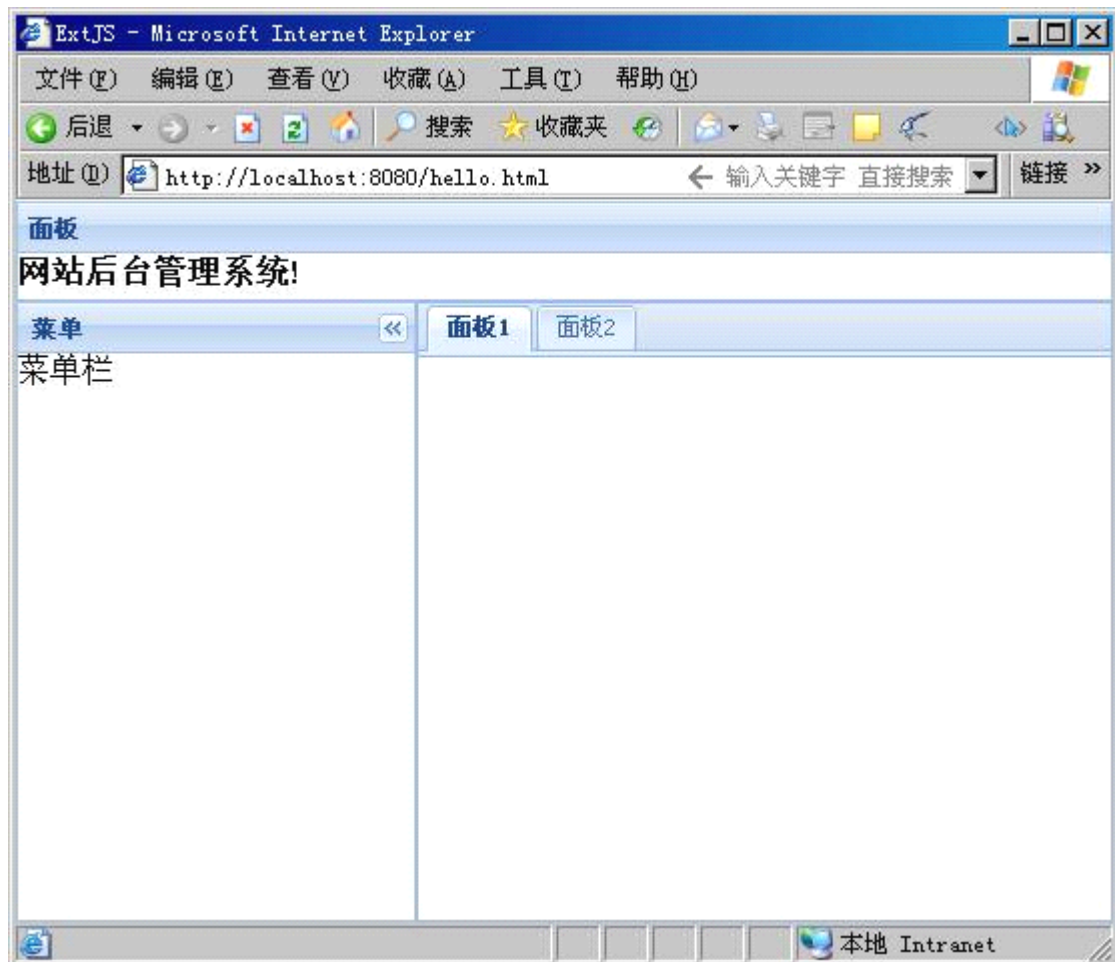
Viewport 主要用于应用程序的主界面, 可以通过使用不同的布局来搭建出不同风格的应用程序主界面。在 Viewport 上常用的布局有 `fit`、`border` 等, 当然在需要的时候其它布局也会常用。看下面的代码:

```
Ext.onReady(function(){
    new Ext.Viewport({
        enableTabScroll:true,
        layout:"border",
        items:[{title:"面板",
            region:"north",
            height:50,
            html:"<h1>网站后台管理系统!</h1>"
        },
        {title:"菜单",
            region:"west",
            width:200,
            collapsible:true,
            html:"菜单栏"
        },
        {
            xtype:"tabpanel",
            region:"center",
```



```
        items:[{title:"面板1"},  
               {title:"面板2"}]  
    }  
]  
});  
});
```

运行上面的程序会得如图 xx 所示的效果。



第五章、窗口及对话框

5.1、窗口基本应用

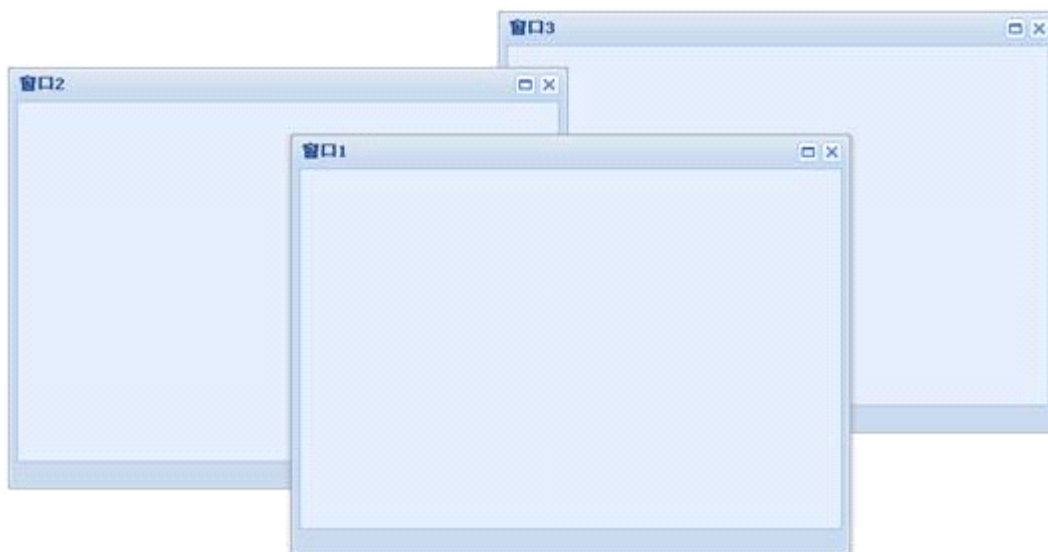
ExtJS 中窗口是由 **Ext.Window** 类定义，该类继承自 Panel，因此窗口其实是一种特殊的面板 Panel。窗口包含了浮动、可拖动、可关闭、最大化、最小化等特性。看下面的代码：

```
var i=0;
function newWin()
{
    var win=new Ext.Window({title:"窗口"+i++,
        width:400,
        height:300,
        maximizable:true});
    win.show();
}
Ext.onReady(function(){
    Ext.get("btn").on("click",newWin);
});
```

页面中的 html 内容:

```
<input id="btn" type="button" name="add" value="新窗口" />
```

执行上面的代码，当点击按钮“新窗口”的时候，会在页面中显示一个窗口，窗口标题为“窗口 x”，窗口可以关闭，可以最大化，点击最大化按钮会最大化窗口，最大化的窗口可以还原，如图 xxx 所示。



5.2、窗口分组

窗口是分组进行管理的，可以对一组窗口进行操作，默认情况下的窗口都在默认的组 Ext.WindowMgr 中。窗口分组由类 Ext.WindowGroup 定义，该类包括 bringToFront、getActive、hideAll、sendToBack 等方法用来对分组中的窗口进行操作。

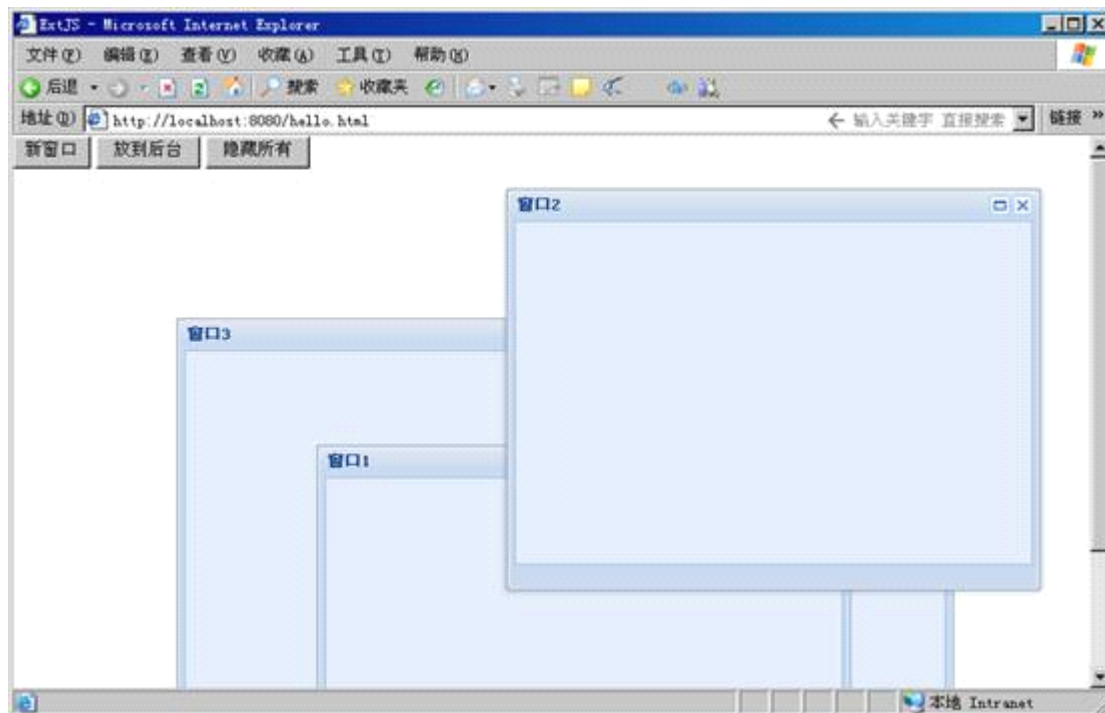
看下面的代码：

```
var i=0,mygroup;
function newWin()
{
    var win=new Ext.Window({title:"窗口"+i++,
        width:400,
        height:300,
        maximizable:true,
        manager:mygroup});
    win.show();
}
function toBack()
{
    mygroup.sendToBack(mygroup.getActive());
}
function hideAll()
{
    mygroup.hideAll();
}
Ext.onReady(function(){
    mygroup=new Ext.WindowGroup();
    Ext.get("btn").on("click",newWin);
    Ext.get("btnToBack").on("click",toBack);
    Ext.get("btnHide").on("click",hideAll);
});
```

页面中的 html 代码

```
<input id="btn" type="button" name="add" value="新窗口" />
<input id="btnToBack" type="button" name="add" value="放到后台" />
<input id="btnHide" type="button" name="add" value="隐藏所有" />
```

执行上面的代码，先点击几次“新窗口”按钮，可以在页面中显示几个容器，然后拖动这些窗口，让他们在屏幕中不同的位置。然后点“放到后台”按钮，可以实现把最前面的窗口移动该组窗口的最后面去，点击“隐藏所有”按钮，可以隐藏当前打开的所有窗口。如下图所示：



5.3、对话框

由于传统使用 `alert`、`confirm` 等方法产生的对话框非常古板，不好看。因此，ExtJS 提供了一套非常漂亮的对话框，可以使用这些对话框代替传统的 `alert`、`confirm` 等，实现华丽的应用程序界面。

Ext 的对话框都封装在 `Ext.MessageBox` 类，该类还有一个简写形式即 `Ext.Msg`，可以直接通过 `Ext.MessageBox` 或 `Ext.Msg` 来直接调用相应的对话框方法来显示 Ext 对话框。看下面的代码：

```
Ext.onReady(function(){
    Ext.get("btnAlert").on("click",function(){
        Ext.MessageBox.alert("请注意","这是ExtJS的提示框");
    });
});
```

Html 页面中的内容：

```
<input id="btnAlert" type="button" value="alert框" />
```

执行程序，点击上面的“alert 框”按钮，将会在页面上显示如下图所示的对话框。



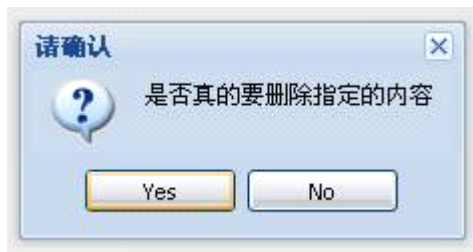
除了 alert 以外，Ext 还包含 **confirm**、**prompt**、**progress**、**wait** 等对话框，另外我们可以根据需要显示自定义的对话框。普通对话框一般包括四个参数，比如 confirm 的方法签名为 `confirm (String title, String msg, [Function fn], [Object scope])`，参数 title 表示对话框的标题，参数 msg 表示对话框中的提示信息，这两个参数是必须的；可选的参数 fn 表示当关闭对话框后执行的回调函数，参数 scope 表示回调函数的执行作用域。**回调函数可以包含两个参数，即 button 与 text**，button 表示点击的按钮，text 表示对话框中有活动输入选项时输入的文本内容。我们可以在回调函数中通过 button 参数来判断用户作了什么什么选择，可以通过 text 来读取在对话框中输入的内容。看下面的 例子：]

```
Ext.onReady(function(){
    Ext.get("btn").on("click",function(){
        Ext.MessageBox.confirm("请确认","是否真的要删除指定的内容",function(button,text){
            alert(button);
            alert(text);
        });
    });
});
```

Html 内容:

```
<input id="对话框" type="button" value="btn" />
```

点击对话框按钮将会出现下面的对话框，然后选择 yes 或 no 则会用传统的提示框输出回调函数中 button 及 text 参数的内容。



因此，在实际的应用中，上面的代码可以改成如下的内容：

```
Ext.onReady(function(){
    Ext.get("btnAlert").on("click",function(){
        Ext.MessageBox.confirm("请确认","是否真的要删除指定的内容",function(button,text){
            if(button=="yes"){
                //执行删除操作
                alert("成功删除");
            }
        });
    });
});
```

这样当用户点击对话框中的 yes 按钮时，就会执行相应的操作，而选择 no 则忽略操作。下面再看看 prompt 框，我们看下面的代码：

```
Ext.onReady(function(){
```

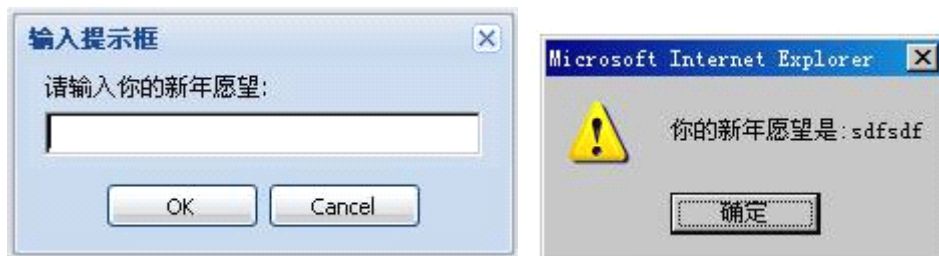


```
Ext.get("btn").on("click",function(){
    Ext.MessageBox.prompt("输入提示框","请输入你的新年愿望:",function(button,text){
        if(button=="ok"){
            alert("你的新年愿望是:"+text);
        }
        else alert("你放弃了录入!");
    });
});
});
```

Html 页面:

```
<input id="btn" type="button" value="对话框" />
```

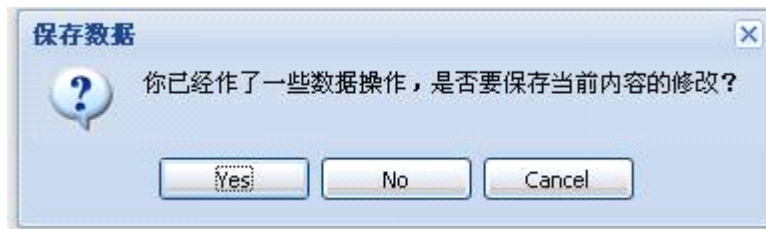
点击上面的“对话框”按钮可以显示如下图所示的内容，如果点击 OK 按钮则会输入你输入的文本内容，选择 cancel 按钮则会提示放弃了录入，如下图所示：



在实际应用中，可以直接使用 MessageBox 的 show 方法来显示自定义的对话框，如下面的代码：

```
function save(button)
{
    if(button=="yes")
    {
        //执行数据保存操作
    }
    else if(button=="no")
    {
        //不保存数据
    }
    else
    {
        //取消当前操作
    }
}
Ext.onReady(function(){
    Ext.get("btn").on("click",function(){
        Ext.Msg.show({
            title:'保存数据',
            msg: '你已经作了一些数据操作，是否要保存当前内容的修改？',
            buttons: Ext.Msg.YESNOCANCEL,
            fn: save,
            icon: Ext.MessageBox.QUESTION});
    });
});
```

点击“对话框”按钮可显示一个自定义的保存数据对话框，对话框中包含 yes、no、cancel 三个按钮，可以在回调函数 save 中根据点击的按钮执行相应的操作，如图 xx 所示。



第六章、布局 Layout

6.1、布局概述

所谓布局就是指容器组件中子元素的分布、排列组合方式。Ext 的所有容器组件都支持而局操作，每一个容器都会有一个对应的布局，布局负责管理容器组件中子元素的排列、组合及渲染方式等。

ExtJS 的布局基类为 **Ext.layout.ContainerLayout**，其它布局都是继承该类。ExtJS 的容器组件包含一个 **layout** 及 **layoutConfig** 配置属性，这两个属性用来指定容器使用的布局及布局的详细配置信息，如果没有指定容器组件的 **layout** 则默认会使用 **ContainerLayout** 作为布局，该布局只是简单的把元素放到容器中，有的布局需要 **layoutConfig** 配置，有的则不需要 **layoutConfig** 配置。看代码：

```
Ext.onReady(function() {  
    new Ext.Panel({  
        renderTo:"hello",  
        width:400,  
        height:200,  
        layout:"column",  
        items:[{columnWidth:.5,  
            title:"面板1"},  
            {columnWidth:.5,  
            title:"面板2"}]  
    });  
});
```

上面的代码我们创建了一个面板 **Panel**，**Panel** 是一个容器组件，我们使用 **layout** 指定该面板使用 **Column** 布局。该面板的子元素是两个面板，这两个面板都包含了一个与列布局相关的配置参数属性 **columnWidth**，他们的值都是 0.5，也就是每一个面板占一半的宽度。执行上面的程序生成如下图所示的结果：

| 面板1 | 面板2 |
|-----|-----|
| | |

Ext中的一些容器组件都已经指定所使用的布局，比如TabPanel使用card布局、FormPanel使用form布局，GridPanel中的表格使用column布局等，我们在使用这些组件的时候，不能给这些容器组件再指定另外的布局。

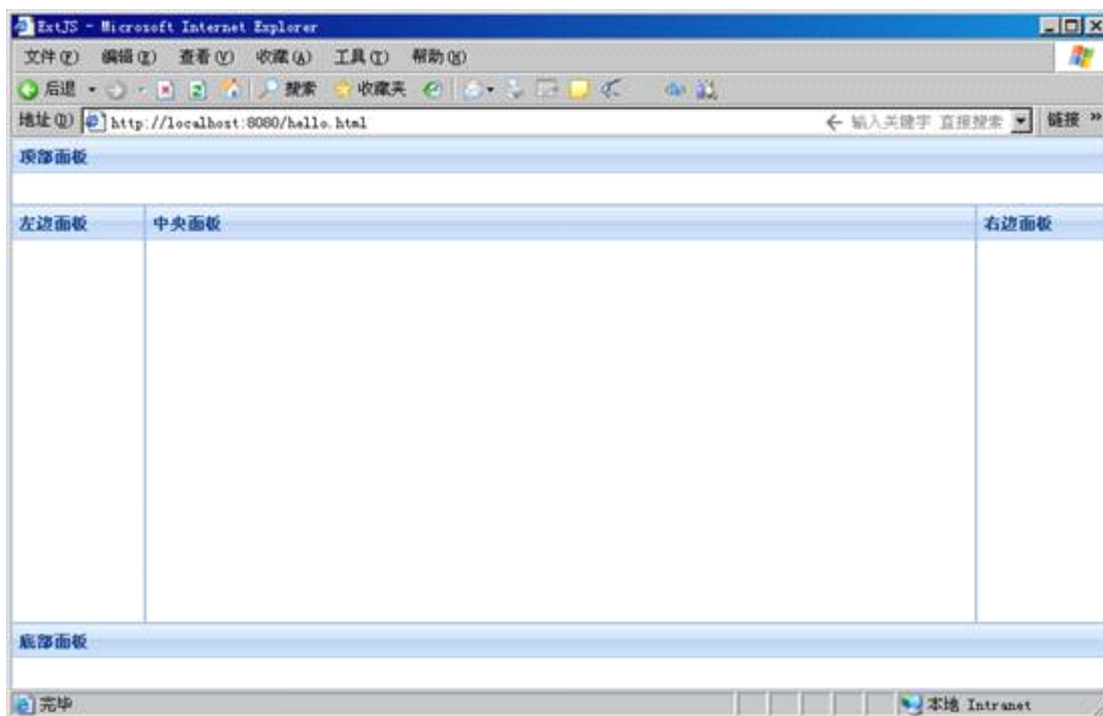
ExtJS2.0 一共包含十种布局，常用的布局有border、column、fit、form、card、table等布局，下面我们分别对这几种布局作简单的介绍。

6.2、Border 区域布局

Border 布局由类Ext.layout.BorderLayout定义，布局名称为border。该布局把容器分成东南西北中五个区域，分别由east, south, west, north, center来表示，在往容器中添加子元素的时候，我们只需要指定这些子元素所在的位置，Border布局会自动把子元素放到布局指定的位置。看下面的代码：

```
Ext.onReady(function(){
    new Ext.Viewport({
        layout:"border",
        items:[ {region:"north",
            height:50,
            title:"顶部面板"},
            {region:"south",
            height:50,
            title:"底部面板"},
            {region:"center",
            title:"中央面板"},
            {region:"west",
            width:100,
            title:"左边面板"},
            {region:"east",
            width:100,
            title:"右边面板"}
        ],
    });
});
```

执行上面的代码将会在页面中输出包含上下左右中五个区域的面板，如下图所示：



6.2、Column 列布局

Column 列布局由 Ext.layout.ColumnLayout 类定义，名称为 column。列布局把整个容器组件看成一行，然后往里面放入子元素的时候，可以通过在子元素中指定使用 `columnWidth` 或 `width` 来指定子元素所占的列宽度。`columnWidth` 表示使用百分比的形式指定列宽度，而 `width` 则是使用绝对像素的方式指定列宽度，在实际应用中可以混合使用两种方式。看下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"column",
        width:500,
        height:100,
        items:[ {title:"列1",width:100},
                {title:"列2",width:200},
                {title:"列3",width:100},
                {title:"列4"}
            ],
    });
});
```

上面的代码在容器组件中放入了四个元素，在容器组件中形成 4 列，列的宽度分别为 100,200,100 及剩余宽度，执行结果如下图所示。

| 容器组件 | | | |
|------|----|----|----|
| 列1 | 列2 | 列3 | 列4 |
| | | | |

也可使用 `columnWidth` 来定义子元素所占的列宽度，看下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"column",
        width:500,
        height:100,
        items:[{title:"列1",columnWidth:.2},
            {title:"列2",columnWidth:.3},
            {title:"列3",columnWidth:.3},
            {title:"列4",columnWidth:.2}
        ]
    });
});
```

注意 `columnWidth` 的总和应该为 1，执行代码将生成如下图所示的内容：

| 容器组件 | | | |
|------|----|----|----|
| 列1 | 列2 | 列3 | 列4 |
| | | | |

在实际应用中还可以混合使用，看下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"column",
        width:500,
        height:100,
        items:[{title:"列1",width:200},
```

```
{title:"列2",columnWidth:.3},
{title:"列3",columnWidth:.3},
{title:"列4",columnWidth:.4}
]}
);
});
```

执行上面的代码将会生成如下图所示的结果：

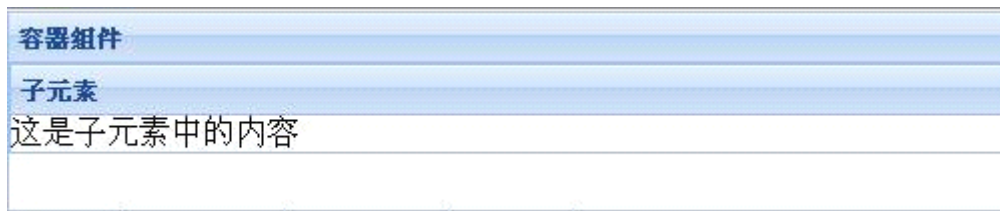
| 容器组件 | | | |
|------|----|----|----|
| 列1 | 列2 | 列3 | 列4 |
| | | | |
| | | | |

6.3、Fit 布局

Column 列布局由 Ext.layout.ColumnLayout 类定义，名称为 column。列布局把整个容器组件看成一行，然后往里面放入子元素的时候，可以通过在子元素中指定使用 `columnWidth` 或 `width` 来指定子元素所占的列宽度。`columnWidth` 表示使用百分比的形式指定列宽度，而 `width` 则是使用绝对像素的方式指定列宽度，在实际应用中可以混合使用两种方式。看下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"column",
        width:500,
        height:100,
        items:[{title:"列1",width:100},
                {title:"列2",width:200},
                {title:"列3",width:100},
                {title:"列4"}
            ]
    });
});
```

上面的代码在容器组件中放入了四个元素，在容器组件中形成 4 列，列的宽度分别为 100,200,100 及剩余宽度，执行结果如下图所示。



再看使用 Fit 布局后的代码，如下：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"fit",
        width:500,
        height:100,
        items:[{title:"子元素",html:"这是子元素中的内容"}]
    });
});
```

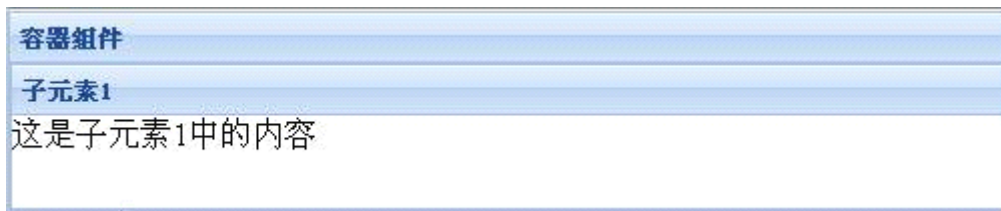
上面的代码指定父容器使用 Fit 布局，因此子将自动填满整个父容器。输出的图形如下：



如果容器组件中有多个子元素，则只会显示一个元素，如下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        layout:"fit",
        width:500,
        height:100,
        items:[{title:"子元素1",html:"这是子元素1中的内容"},
            {title:"子元素2",html:"这是子元素2中的内容"}]
    });
});
```

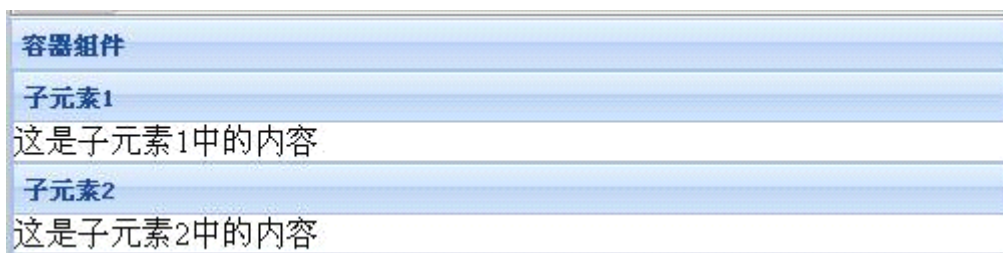
输出的结果如下：



如果不使用布局 Fit，代码如下：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        width:500,
        height:120,
        items:[{title:"子元素1",html:"这是子元素1中的内容"},
            {title:"子元素2",html:"这是子元素2中的内容"}
        ]
    });
});
```

输出的结果如下图所示：



6.4、Form 布局

Form 布局由类 `Ext.layout.FormLayout` 定义，名称为 form，是一种专门用于管理表单中输入字段的布局，这种布局主要用于在程序中创建表单字段或表单元素等使用。看下面的代码：

```
Ext.onReady(function(){
    new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        width:300,
        layout:"form",
        hideLabels:false,
        labelAlign:"right",
    });
});
```

```
height:120,
defaultType: 'textfield',
items:[
  {fieldLabel:"请输入姓名",name:"name"},
  {fieldLabel:"请输入地址",name:"address"},
  {fieldLabel:"请输入电话",name:"tel"}
]
);
});
```

上面的代码创建了一个面板，面板使用 Form 布局，面板中包含三个子元素，这些子元素都是文本框字段，在父容器中还通过 hideLabels、labelAlign 等配置属性来定义了是否隐藏标签、标签对齐方式等。上面代码的输出结果如下图所示：

可以在容器组件中把 hideLabels 设置为 true，这样将不会显示容器中字段的标签了，如下图所示：

在实际应用中，Ext.form.FormPanel 这个类默认布局使用的是 Form 布局，而且 FormPanel 还会创建与 <form> 标签相关的组件，因此一般情况下我们直接使用 FormPanel 即可。上面的例子可改写成如下的形式：

```
Ext.onReady(function(){
  new Ext.form.FormPanel({
    renderTo:"hello",
    title:"容器组件",
    width:300,
    labelAlign:"right",
    height:120,
    defaultType: 'textfield',
    items:[
      {fieldLabel:"请输入姓名",name:"name"},
      {fieldLabel:"请输入地址",name:"address"},
      {fieldLabel:"请输入电话",name:"tel"}
    ]
  });
});
```

```
    }}  
  );  
});
```

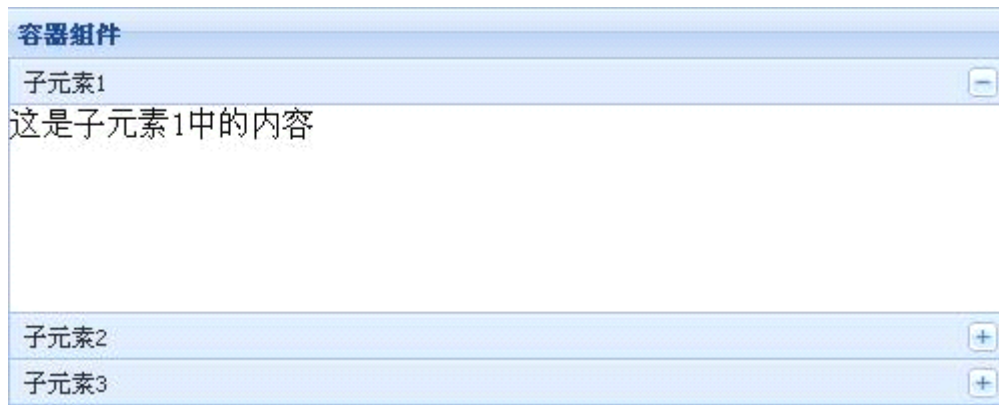
程序结果与前面使用 Ext.Panel 并指定 form 布局的一样，如下图所示：

6.5、Accordion 布局

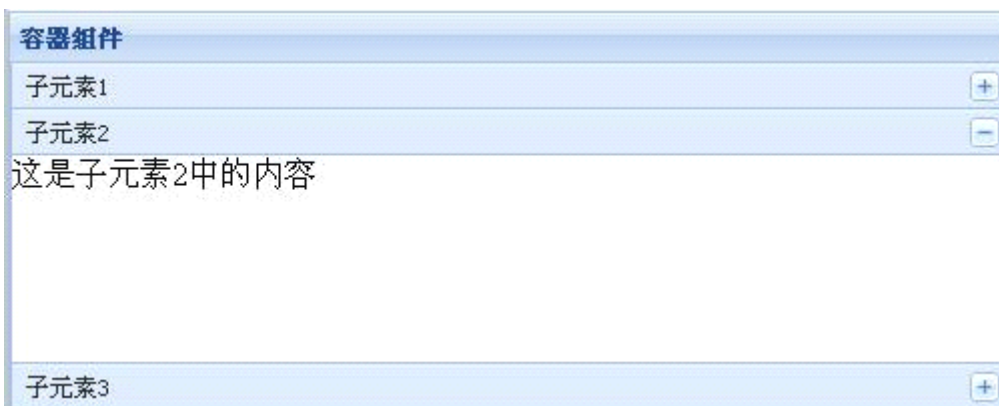
Accordion 布局由类 Ext.layout.Accordion 定义，名称为 accordion，表示可折叠的布局，也就是说使用该布局的容器组件中的子元素是可折叠的形式。来看下面的代码：

```
Ext.onReady(function(){  
    new Ext.Panel({  
        renderTo:"hello",  
        title:"容器组件",  
        width:500,  
        height:200,  
        layout:"accordion",  
        layoutConfig: {  
            animate: true  
        },  
        items:[ {title:"子元素1",html:"这是子元素1中的内容"},  
                {title:"子元素2",html:"这是子元素2中的内容"},  
                {title:"子元素3",html:"这是子元素3中的内容"}  
        ]  
    });  
});
```

上面的代码定义了一个容器组件，指定使用 Accordion 布局，该容器组件中包含三个子元素，在 layoutConfig 中指定布局配置参数 animate 为 true，表示在执行展开折叠时是否应用动画效果。执行结果将生成如下图所示的界面：



点击每一个子元素的头部名称或右边的按钮，则会展开该面板，并收缩其它已经展开的面板，如下图：



6.6、Table 布局及其它布局

Table 布局由类 Ext.layout.TableLayout 定义，名称为 table，该布局负责把容器中的子元素按照类似普通 html 标签

```
Ext.onReady(function(){
    var panel=new Ext.Panel({
        renderTo:"hello",
        title:"容器组件",
        width:500,
        height:200,
        layout:"table",
        layoutConfig: {
            columns: 3
        },

        items:[ {title:"子元素1",html:"这是子元素1中的内容",rowspan:2,height:100},
                {title:"子元素2",html:"这是子元素2中的内容",colspan:2},
                {title:"子元素3",html:"这是子元素3中的内容"},
                {title:"子元素4",html:"这是子元素4中的内容"}
    ]
    });
});
```

```
}  
);  
});
```

上面的代码创建了一个父容器组件，指定使用 Table 布局，layoutConfig 使用 columns 指定父容器分成 3 列，子元素中使用 rowspan 或 colspan 来指定子元素所横跨的单元格数。程序的运行效果如下图所示：

| 容器组件 | | |
|------------|------------|------------|
| 子元素1 | 子元素2 | |
| 这是子元素1中的内容 | 这是子元素2中的内容 | |
| | 子元素3 | 子元素4 |
| | 这是子元素3中的内容 | 这是子元素4中的内容 |

除了前面介绍的几种布局以外，Ext2.0 中还包含其它的 Ext.layout.AbsoluteLayout、Ext.layout.AnchorLayout 等布局类，这些布局主要作为其它布局的基类使用，一般情况下我们不会在应用中直接使用。另外，我们也可以继承 10 种布局类的一种，来实现自定义的布局。

关于ExtJS布局的详细说明，请参考wlr.easyjf.com中的VIP文档《ExtJS布局Layout详解（1）、（2）、（3）》。

第七章、使用表格控件

7.1、基本表格 GridPanel

ExtJS 中的表格功能非常强大，包括了排序、缓存、拖动、隐藏某一列、自动显示行号、列汇总、单元格编辑等实用功能。

表格由类 Ext.grid.GridPanel 定义，继承自 Panel，其 xtype 为 grid。ExtJS 中，表格 Grid 必须包含列定义信息，并指定表格的数据存储器 Store。表格的列信息由类 Ext.grid.ColumnModel 定义、而表格的数据存储器由 Ext.data.Store 定义，数据存储器根据解析的数据不同分为 JsonStore、SimpleStore、GroupingStore 等。

我们首先来看最简单的使用表格的代码：

```
Ext.onReady(function(){  
    var data=[ [1, 'EasyJWeb', 'EasyJF', 'www.easyjf.com'],  
                [2, 'jfox', 'huihoo', 'www.huihoo.org'],
```



```
[3, 'jdon', 'jdon', 'www.jdon.com'],
[4, 'springside', 'springside', 'www.springside.org.cn'] ];
var store=new Ext.data.SimpleStore({data:data,fields:["id","name","organization","homepage"]});
var grid = new Ext.grid.GridPanel({
    renderTo:"hello",
    title:"中国Java开源产品及团队",
    height:150,
    width:600,
    columns:[ {header:"项目名称",dataIndex:"name"},
    {header:"开发团队",dataIndex:"organization"},
    {header:"网址",dataIndex:"homepage"}],
    store:store,
    autoExpandColumn:2
});
});
```

执行上面的代码，可以得到一个简单的表格，如下图所示：

| 中国Java开源产品及团队 | | |
|---------------|------------|-----------------------|
| 项目名称 | 开发团队 | 网址 |
| EasyJWeb | EasyJF | www.easyjf.com |
| jfox | huihoo | www.huihoo.org |
| jdon | jdon | www.jdon.com |
| springside | springside | www.springside.org.cn |

上面的代码中，第一行“var data=...”用来定义表格中要显示的数据，这是一个[[[]]二维数组；第二行“var store=...”用来创建一个数据存储，这是 GridPanel 需要使用配置属性，数据存储 Store 负责把各种各样的数据（如二维数组、JSON 对象数组、xml 文本）等转换成 ExtJS 的数据记录集 Record，关于数据存储 Store 我们将在下一章中作专门介绍。第三行“var grid = new Ext.grid.GridPanel(...)”负责创建一个表格，表格包含的列由 columns 配置属性来描述，columns 是一数组，每一行数据元素描述表格的一列信息，表格的列信息包含列头显示文本(header)、列对应的记录集字段(dataIndex)、列是否可排序(sortable)、列的渲染函数(renderer)、宽度(width)、格式化信息(format)等，在上面的例子中只用到了 header 及 dataIndex。

下面我们简单看看表格的排序及隐藏列特性，简单修改一下上面的代码，内容如下：

```
Ext.onReady(function(){
    var data=[ [1, 'EasyJWeb', 'EasyJF', 'www.easyjf.com'],
    [2, 'jfox', 'huihoo', 'www.huihoo.org'],
    [3, 'jdon', 'jdon', 'www.jdon.com'],
    [4, 'springside', 'springside', 'www.springside.org.cn'] ];
    var store=new Ext.data.SimpleStore({data:data,fields:["id","name","organization","homepage"]});
    var colM=new Ext.grid.ColumnModel([ {header:"项目名称",dataIndex:"name",sortable:true},
    {header:"开发团队",dataIndex:"organization",sortable:true},
```

```
{header:"网址",dataIndex:"homepage"}}});
var grid = new Ext.grid.GridPanel({
    renderTo:"hello",
    title:"中国Java开源产品及团队",
    height:200,
    width:600,
    cm:colM,
    store:store,
    autoExpandColumn:2
});
});
```

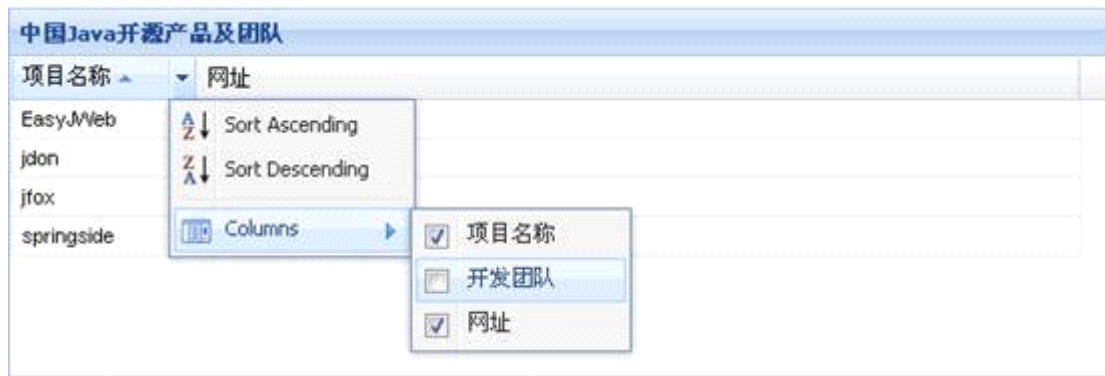
直接使用 new Ext.grid.ColumnModel 来创建表格的列信定义信息，在“项目名称”及“开发团队”列中我们添加了 sortable 为 true 的属性，表示该列可以排序，执行上面的代码，我们可以得到一个支持按“项目名称”或“开发团队”的表格，如图 xxx 所示。

| 中国Java开源产品及团队 | | |
|---------------|------------|-----------------------|
| 项目名称 ▲ | 开发团队 | 网址 |
| EasyJWeb | EasyJF | www.easyjf.com |
| jdon | jdon | www.jdon.com |
| jfox | huihoo | www.huihoo.org |
| springside | springside | www.springside.org.cn |

(按项目名称排序)

| 中国Java开源产品及团队 | | |
|---------------|------|-----------------------|
| 项目名称 ▲ ▼ | 开发团队 | 网址 |
| EasyJWeb | | www.easyjf.com |
| jdon | | www.jdon.com |
| jfox | | www.huihoo.org |
| springside | | www.springside.org.cn |

(可排序的列表头后面小按钮可以弹出操作菜单)



另外，每一列的数据渲染方式还可以自己定义，比如上面的表格中，我们希望用户在表格中点击网址则直接打开这些开源团队的网站，也就是需要给网址这一列添加上超级连接。下面的代码实现这个功能：

```
function showUrl(value)
{
    return "<a href='"+value+"'>";
}

Ext.onReady(function(){
    var data=[ [1, 'EasyJWeb', 'EasyJF','www.easyjf.com'],
    [2, 'jfox', 'huihoo','www.huihoo.org'],
    [3, 'jdon', 'jdon','www.jdon.com'],
    [4, 'springside', 'springside','www.springside.org.cn'] ];
    var store=new Ext.data.SimpleStore({data:data,fields:["id","name","organization","homepage"]});
    var colM=new Ext.grid.ColumnModel([ {header:"项目名称",dataIndex:"name",sortable:true},
    {header:"开发团队",dataIndex:"organization",sortable:true},
    {header:"网址",dataIndex:"homepage",renderer:showUrl} ]);
    var grid = new Ext.grid.GridPanel({
        renderTo:"hello",
        title:"中国Java开源产品及团队",
        height:200,
        width:600,
        cm:colM,
        store:store,
        autoExpandColumn:2
    });
});
```

上面的代码跟前面的示例差别不大，只是在定义“网址”列的时候多了一个 `renderer` 属性，即 `{header:"网址",dataIndex:"homepage",renderer:showUrl}`。 `showUrl` 是一个自定义的函数，内容就是根据传入的 `value` 参数返回一个包含 `<a>` 标签的 `html` 片段。运行上面的代码显示结果如下图所示：

| 中国Java开源产品及团队 | | |
|---------------|------------|--|
| 项目名称 | 开发团队 | 网址 |
| EasyJWeb | EasyJF | www.easyjf.com |
| jfox | huihoo | www.huihoo.org |
| jdon | jdon | www.jdon.com |
| springside | springside | www.springside.org.cn |

自定义的列渲染函数可以实现在单元格中显示自己所需要的各种信息，只是的浏览器能处理的 html 都可以。

除了二级数组以外，表格还能显示其它格式的数据吗？答案是肯定的，下面假如我们的表格数据 data 定义成了下面的形式：

```
var data=[{id:1,
  name:'EasyJWeb',
  organization:'EasyJF',
  homepage:'www.easyjf.com'},
  {id:2,
  name:'jfox',
  organization:'huihoo',
  homepage:'www.huihoo.org'},
  {id:3,
  name:'jdon',
  organization:'jdon',
  homepage:'www.jdon.com'},
  {id:4,
  name:'springside',
  organization:'springside',
  homepage:'www.springside.org.cn'}
];
```

也就是说数据变成了一维数组，数组中的每一个元素是一个对象，这些对象包含 name、organization、homepage、id 等属性。要让表格显示上面的数据，其实非常简单，只需要把 store 改成用 Ext.data.JsonStore 即可，代码如下：

```
var store=new Ext.data.JsonStore({data:data,fields:["id","name","organization","homepage"]});
var colM=new Ext.grid.ColumnModel([ {header:"项目名称",dataIndex:"name",sortable:true},
  {header:"开发团队",dataIndex:"organization",sortable:true},
  {header:"网址",dataIndex:"homepage",renderer:showUrl}]);
var grid = new Ext.grid.GridPanel({
  renderTo:"hello",
  title:"中国Java开源产品及团队",
```

```
height:200,
width:600,
cm:colM,
store:store,
autoExpandColumn:2
});
```

上面的代码得到的结果与前面的一样。当然，表格同样能显示 xml 格式的数据，假如上面的数据存放成 hello.xml 文件中，内容如下：

为了把这个 xml 数据用 ExtJS 的表格 Grid 进行显示，我们只需要把 store 部分的内容调整成如下的内容即可：

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <row>
    <id>1</id>
    <name>EasyJWeb</name>
    <organization>EasyJF</organization>
    <homepage>www.easyjf.com</homepage>
  </row>
  <row>
    <id>2</id>
    <name>jfox</name>
    <organization>huihoo</organization>
    <homepage>www.huihoo.org</homepage>
  </row>
  <row>
    <id>3</id>
    <name>jdon</name>
    <organization>jdon</organization>
    <homepage>www.jdon.com</homepage>
  </row>
  <row>
    <id>4</id>
    <name>springside</name>
    <organization>springside</organization>
    <homepage>www.springside.org.cn</homepage>
  </row>
</dataset>
```

```
var store=new Ext.data.Store({
    url:"hello.xml",
    reader:new Ext.data.XmlReader({
        record:"row"},
        ["id","name","organization","homepage"])
});
```

其它的部分不用改变，完整的代码如下：

```
function showUrl(value)
{
    return "<a href='http://'+value+' target='_blank'>"+value+"</a>";
}
Ext.onReady(function(){
    var store=new Ext.data.Store({
        url:"hello.xml",
        reader:new Ext.data.XmlReader({
            record:"row"},
            ["id","name","organization","homepage"])
        });
    var colM=new Ext.grid.ColumnModel([ {header:"项目名称",dataIndex:"name",sortable:true},
        {header:"开发团队",dataIndex:"organization",sortable:true},
        {header:"网址",dataIndex:"homepage",renderer:showUrl}]);
    var grid = new Ext.grid.GridPanel({
        renderTo:"hello",
        title:"中国Java开源产品及团队",
        height:200,
        width:600,
        cm:colM,
        store:store,
        autoExpandColumn:2
    });
    store.load();
});
```

store.load()是用来加载数据，执行上面的代码产生的表格与前面的完全一样。

关于表格控件 GridPanel 的详细说明，请参考 wlr.easyjif.com 中的 VIP 文档《ExtJS 表格 GridPanel 详解》。

7.2、可编辑表格 EditorGridPanel

可编辑表格是指可以直接在表格的单元格对表格的数据进行编辑，ExtJS 中的可编辑表格由类 `Ext.grid.EditorGridPanel` 表示，`xtype` 为 `editorgrid`。使用 `EditorGridPanel` 与使用普通的

GridPanel 方式一样，区别只是在定义列信息的时候，可以指定某一列使用的编辑即可，下面来看一个简单的示例。

```
Ext.onReady(function(){
    var data=[{id:1,
        name:'小王',
        email:'xiaowang@easyjf.com',
        sex:'男',
        bornDate:'1991-4-4'},
        {id:1,
        name:'小李',
        email:'xiaoli@easyjf.com',
        sex:'男',
        bornDate:'1992-5-6'},
        {id:1,
        name:'小兰',
        email:'xiaoxiao@easyjf.com',
        sex:'女',
        bornDate:'1993-3-7'}
    ];
    var store=new Ext.data.JsonStore({
        data:data,
        fields:["id","name","sex","email",{name:"bornDate",type:"date",dateFormat:"Y-n-j"}]
    });
    var colM=new Ext.grid.ColumnModel([
        {header:"姓名",
        dataIndex:"name",
        sortable:true,
        editor:new Ext.form.TextField()},
        {header:"性别",
        dataIndex:"sex"
        },
        {header:"出生日期",
        dataIndex:"bornDate",
        width:120,
        renderer:Ext.util.Format.dateRenderer('Y年m月d日')},
        {header:"电子邮件",
        dataIndex:"email",
        sortable:true,
        editor:new Ext.form.TextField()
        }
    ]);
    var grid = new Ext.grid.EditorGridPanel({
        renderTo:"hello",
```

```

        title:"学生基本信息管理",
        height:200,
        width:600,
        cm:colM,
        store:store,
        autoExpandColumn:3
    });
});

```

上面的程序首先定义了一个包含学生信息的对象数组，然后创建了一个 `JsonStore`，在创建这个 `store` 的时候，指定 `bornDate` 列的类型为日期 `date` 类型，并使用 `dateFormat` 来指定日期信息的格式为“Y-n-j”，Y 代表年，n 代表月，j 代表日期。定义表格列模型的时候，对于“姓名”及“电子邮件”列我们使用 `editor` 来定义该列使用的编辑器，这里是使用 `Ext.form.TextField`，最后使用 `new Ext.grid.EditorGridPanel(...)` 来创建一个可编辑的表格。执行上面的程序可以生成一个表格，双击表格中的“姓名”、或“电子邮件”单元格中的信息可以触发单元格的编辑，可以在单元格的文本框中直接编辑表格中的内容，修改过的单元格会有特殊的标记，如下图所示：

| 学生基本信息管理 | | | |
|----------|----|-------------|---------------------|
| 姓名 | 性别 | 出生日期 | 电子邮件 |
| 小王 | 男 | 1991年04月04日 | xiaowang@easyjf.com |
| 小李 | 男 | 1992年05月06日 | xiaoli@easyjf.com66 |
| 小兰 | 女 | 1993年03月07日 | xiaoxiao@easyjf.com |

为了能编辑“性别”及“出生日期”列，同样只需要在定义该列的时候指定 `editor` 即可。由于出生日期是日期类型，因此我们可以使用日期编辑器来编辑，“性别”列的数据不应该让用户直接输入，而应该是通过下拉框进行选择。日期编辑器可以直接使用 `Ext.form.DateField` 组件，下拉选择框编辑器可以使用 `Ext.form.ComboBox` 组件，下面是实现对性别及出生日期等列信息编辑的代码：

```

var colM=new Ext.grid.ColumnModel([
    {
        header:"姓名",
        dataIndex:"name",
        sortable:true,
        editor:new Ext.form.TextField(),
        {header:"性别",
        dataIndex:"sex",
        editor:new Ext.form.ComboBox({transform:"sexList",
        triggerAction: 'all',
        lazyRender:true})
    }
]);

```

```

    },
    {header:"出生日期",
    dataIndex:"bornDate",
    width:120,
    renderer:Ext.util.Format.dateRenderer('Y年m月d日'),
    editor:new Ext.form.DateField({format:'Y年m月d日'})},
    {header:"电子邮件",
    dataIndex:"email",
    sortable:true,
    editor:new Ext.form.TextField()
    });
var grid = new Ext.grid.EditorGridPanel({
    renderTo:"hello",
    title:"学生基本信息管理",
    height:200,
    width:600,
    cm:colM,
    store:store,
    autoExpandColumn:3,
    clicksToEdit:1
});

```

注意在定义 EditorGridPanel 的时候，我们增加了一个属性“clicksToEdit:1”，表示点击一次单元格即触发编辑，因为默认情况下该值为 2，需要双击单元格才能编辑。为了给 ComboBox 中填充数据，我们使用设置了该组件的 transform 配置属性值为 sexList，sexList 是一个传统的<select>框，我们需要在 html 页面中直接定义，代码如下：

```

<select>
<option>男</option>
<option>女</option>
</select>

```

执行上面的程序，我们可以得到一个能对表格中所有数据进行编辑的表格了。点击上面的“性别”一列的单元格时，会出现一个下拉选择框，点击“出生日期”一列的单元格时，会出现一个日期数据选择框，如图 xxxx 所示：

| 学生基本信息管理 | | | |
|----------|----|-------------|---------------------|
| 姓名 | 性别 | 出生日期 | 电子邮件 |
| 小王 | 男 | 1991年04月04日 | xiaowang@easyjf.com |
| 小李 | 男 | 1992年05月06日 | xiaoli@easyjf.com |
| 小兰 | 男 | 1993年03月07日 | xiaoxiao@easyjf.com |

(编辑性别列中的数据)

| 学生基本信息管理 | | | |
|----------|----|-------------|---------------------|
| 姓名 | 性别 | 出生日期 | 电子邮件 |
| 小王 | 男 | 1991年04月04日 | xiaowang@easyjf.com |
| 小李 | 男 | | lyjf.com |
| 小兰 | 女 | | easyjf.com |

<
April 1991
>

| | | | | | | |
|----|----|----|----|----|----|----|
| S | M | T | W | T | F | S |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Today

(编辑出生日期列中的数据)

那么如何保存编辑后的数据呢？答案是直接使用 **afteredit** 事件。当对一个单元格进行编辑完之后，就会触发 **afteredit** 事件，可以通过该事件处理函数来处理单元格的信息编辑。比如在 <http://wlr.easyjf.com> 这个单用户 blog 示例中，当我们编辑一个日志目录的时候，需要把编辑后的数据保存到服务器，代码如下：

```

this.grid.on("afteredit",this.afterEdit,this);

...
afterEdit:function(obj){
    var r=obj.record;
    var id=r.get("id");
    var name=r.get("name");
    var c=this.record2obj(r);
    var tree=this.tree;
    var node=tree.getSelectionModel().getSelectedNode();
    if(node && node.id!="root")c.parentId=node.id;
    if(id=="-1" && name!=""){
        topicCategoryService.addTopicCategory(c,function(id){
            if(id)r.set("id",id);
            if(!node)node=tree.root;
            node.appendChild(new Ext.tree.TreeNode({
                id:id,
                text:c.name,
                leaf:true
            }));
            node.getUI().removeClass('x-tree-node-leaf');
            node.getUI().addClass('x-tree-node-expanded');
            node.expand();
        });
    }
}

```

```

}
else if(name!="")
{
topicCategoryService.updateTopicCategory(r.get("id"),c,function(ret){
if(ret)tree.getNodeById(r.get("id")).setText(c.name);
});
}
}
}

```

关于可编辑表格控件的详细说明，请参考 wlr.easyjf.com 中的 VIP 文档《ExtJS 可编辑表格 EditorGridPanel 详解》。

7.3、与服务器交互

在实际的应用中，表格中的数据一般都是直接存放在数据库表或服务器的文件中。因此，在使用表格控件的时候经常需要与服务器进行交互。ExtJS 使用 **Ajax 方式提供了一套**与服务器交互的**机制**，也就是可以**不用刷新页面**，就可以访问服务器的程序进行数据读取或数据保存等操作。

比如前面在表格中显示 xml 文档中数据的例子中，就是一个非常简单的从服务器端读取数据的例子，再回顾一下代码：

```

var store=new Ext.data.Store({
    url:"hello.xml",
    reader:new Ext.data.XmlReader({
        record:"row"},
        ["id","name","organization","homepage"])
});

```

因为 Sote 组件接受一个参数 url，如果设置 url，则 ExtJS 会创建一个与服务器交互的 Ext.data.**HttpProxy** 对象，该对象通过指定的 Connection 或 Ext.Ajax.request 来向服务端发送请求，从而可以读取到服务器端的数据。

经验表明，服务器端产生 JSON 数据是一种非常不错的选择，也就是说假如服务器的 url“student.ejf?cmd=list”产生下面的 JSON 数据输出：

```

{results:[{id:1,
name:'小王',
email:'xiaowang@easyjf.com',
sex:'男',
bornDate:'1991-4-4'},
{id:1,
name:'小李',
email:'xiaoli@easyjf.com',

```

```
sex:'男',
bornDate:'1992-5-6'},
{id:1,
name:'小兰',
email:'xiaoxiao@easyjf.com',
sex:'女',
bornDate:'1993-3-7'}
]}
```

则前面显示学习信息编辑表格的 store 可以创建成下面的形式：

```
var store=new Ext.data.Store({
    url:"student.ejf?cmd=list",
    reader:new Ext.data.JsonReader({
        root:"result",
        ["id","name","organization","homepage"]
    });
});
```

或者：

```
var store=new Ext.data.JsonStore({
    url:"student.ejf?cmd=list",
    root:"result",
    fields:["id","name","organization","homepage"]});
```

其中 root 表示包含记录集数据的属性。

如果在运行程序中需要给服务器端发送数据的时候，此时可以直接使用 ExtJS 中提供的 Ext.Ajax 对象的 request 方法。比如下面的代码实现放服务器的 student.ejf?cmd=save 这个 url 发起一个请求，并在 params 中指定发送的 Student 对象：

```
var store=new Ext.data.JsonStore({
    url:"student.ejf?cmd=list",
    root:"result",
    fields:["id","name","organization","homepage"]});
function sFn()
{
    alert('保存成功');
}
function fFn()
{
    alert('保存失败');
}
Ext.Ajax.request({
    url: 'student.ejf?cmd=save'
    success: sFn
```



```
failure: fFn,
params: { name: '小李',email: 'xiaoli@easyjf.com',bornDate: '1992-5-6',sex: '男'}
});
```

关于ExtJS中各控件与服务器端如何交互、Ext.Ajax的详细使用说明等请参考wlr.easyjf.com中的VIP文档《ExtJS中客户端控件与服务端控件交互详解》。

第八章、数据存储 Stroe

8.1、Record

在前面的表格应用中，我们已经知道表格的数据是存放类型为 Store 的数据存储器中，通过指定表格 Grid 的 store 属性来设置表格中显示的数据，通过调用 store 的 load 或 reload 方法可以重新加载表格中的数据。ExtJS 中用来定义控件中使用数据的 API 位于 Ext.dd 命名空间中，本章我们重点对 ExtJS 中的数据存储 Store 进行介绍。

1、Record

首先需要明确是，ExtJS 中有一个名为 Record 的类，表格等控件中使用的数据是存放在 Record 对象中，一个 Record 可以理解成关系数据表中的一行，也可以称为记录。Record 对象中即包含了记录（行中各列）的定义信息（也就是该记录包含哪些字段，每一个字段的数据类型等），同时又包含了记录具体的数据信息（也就是各个字段的值）。

我们来看直接使用 Record 的代码：

```
Ext.onReady(function(){
    var MyRecord = Ext.data.Record.create([
        {name: 'title'},
        {name: 'username', mapping: 'author'},
        {name: 'loginTimes', type: 'int'},
        {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
    ]);
    var r=new MyRecord({
        title:"日志标题",
        username:"easyjf",
        loginTimes:100,
        loginTime:new Date()
    });
    alert(MyRecord.getField("username").mapping);
    alert(MyRecord.getField("lastLoginTime").type);
    alert(r.data.username);
});
```

```
alert(r.get("loginTimes"));
});
```

首先使用 Record 的 create 方法创建一个记录集 MyRecord，MyRecord 其实是一个类，该类包含了记录集的定义信息，可以通过 MyRecord 来创建包含字段值的 Record 对象。在上面的代码中，最后的几条语句用来输出记录集的相关信息，MyRecord.getField("username") 可以得到记录中 username 列的字段信息，r.get("loginTimes") 可以得到记录 loginTimes 字段的值，而 r.data.username 同样能得到记录集中 username 字段的值。

对 Record 有了一定的了解，那么要操作记录集中的数据就非常简单了，比如 r.set(name,value) 可以设置记录中某指定字段的值，r.dirty 可以得到当前记录是否有字段的值被更改过等等。

8.2、Store

Store 可以理解为数据存储器，可以理解为客户端的小型数据表，提供缓存等功能。在 ExtJS 中，GridPanel、ComboBox、DataView 等控件一般直接与 Store 打交道，直接通过 store 来获得控件中需要展现的数据等。一个 Store 包含多个 Record，同时 Store 又包含了数据来源，数据解析器等相关信息，Store 通过调用具体的数据解析器(DataReader)来解析指定类型或格式的数据(DataProxy)，并转换成记录集的形式保存在 Store 中，作为其它控件的数据输入。

数据存储器由 Ext.data.Store 类定义，一个完整的数据存储器要知道数据源(DataProxy)及数据解析方式(DataReader)才能工作，在 Ext.data.Store 类中数据源由 proxy 配置属性定义、数据解析（读取）器由 reader 配置属性定义，一个较为按部就班创建 Store 的代码如下：

```
var MyRecord = Ext.data.Record.create([
    {name: 'title'},
    {name: 'username', mapping: 'author'},
    {name: 'loginTimes', type: 'int'},
    {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
]);
var dataProxy=new Ext.data.HttpProxy({url:"link.ejf"});
var theReader=new Ext.data.JsonReader({
    totalProperty: "results",
    root: "rows",
    id: "id"
},MyRecord);
var store=new Ext.data.Store({
    proxy:dataProxy,
    reader:theReader
});
```

```
store.load();
```

当然，这样的难免代码较多，Store 中本身提供了一些快捷创建 Store 的方式，比如上面的示例代码中可以不用先创建一个 HttpProxy，只需要在创建 Store 的时候指定一个 url 配置参数，就会自动使用 HttpProxy 来加载参数。比如，上面的代码可以简化成：

```
var MyRecord = Ext.data.Record.create([
    {name: 'title'},
    {name: 'username', mapping: 'author'},
    {name: 'loginTimes', type: 'int'},
    {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
]);
var theReader=new Ext.data.JsonReader({
    totalProperty: "results",
    root: "rows",
    id: "id"
},MyRecord);
var store=new Ext.data.Store({
    url:"link.ejf",
    proxy:dataProxy,
    reader:theReader
});
store.load();
```

虽然不再需要手动创建 HttpProxy 了，但是仍然需要创建 DataReader 等，毕竟还是复杂，ExtJS 进一步把这种常用的数据存储容器进行了封装，在 Store 类的基础上提供了 SimpleStore、SimpleStore、GroupingStore 等，直接使用 SimpleStore，则上面的代码可以进一步简化成下面的内容：

```
var store=new Ext.data.JsonStore({
    url:"link.ejf?cmd=list",
    totalProperty: "results",
    root: "rows",
    fields:[{name: 'username', mapping: 'author'},
    {name: 'loginTimes', type: 'int'},
    {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
    ]
});
store.load();
```

8.3、DataReader

DataReader 表示数据读取器，也就是数据解析器，其负责把从服务器或者内存数组、xml 文档中获得的杂乱信息转换成 ExtJS 中的记录集 Record 数据对象，并存储到 Store 里面的记录集数组中。

数据解析器的基类由 Ext.data.DataReader 定义，其它具体的数据解析器都是该类的子类，ExtJS 中提供了读取二维数组、JSON 数据及 Xml 文档的三种数据解析器，分别用于把内存中的二维数组、JSON 格式的数据及 XML 文档信息解析成记录集。下面简单的介绍：

1) ArrayReader

Ext.data.ArrayReader 一数组解析器，用于读取二维数组中的信息，并转换成记录集 Record 对象。首先看下面的代码：

```
var MyRecord = Ext.data.Record.create([
    {name: 'title', mapping:1},
    {name: 'username', mapping:2},
    {name: 'loginTimes', type:3}
]);
var myReader = new Ext.data.ArrayReader({
    id: 0
}, MyRecord);
```

这里定义的 myReader 可以读取下面的二维数组：

```
[ [1, '测试', '小王',3], [2, '新年好', 'williamraym',13] ]
```

2) JsonReader

Ext.data.JsonReader 一 Json 数据解析器，用于读取 JSON 格式的数据信息，并转换成记录集 Record 对象。看下面使用 JsonReader 的代码：

```
var MyRecord = Ext.data.Record.create([
    {name: 'title'},
    {name: 'username', mapping: 'author'},
    {name: 'loginTimes', type: 'int'}
]);
var myReader = new Ext.data.JsonReader({
    totalProperty: "results",
    root: "rows",
    id: "id"
}, MyRecord);
```

这里的 JsonReader 可以解析下面的 JSON 数据:

```
{ 'results': 2, 'rows': [
  { id: 1, title: '测试', author: '小王', loginTimes: 3 },
  { id: 2, title: 'Ben', author: 'williamraym', loginTimes:13} ]
}
```

JsonReader 还有比较特殊的用法, 就是可以把 Store 中记录集的配置信息存放直接保存在从服务器端返回的 JSON 数据中, 比如下面的例子:

```
var myReader = new Ext.data.JsonReader();
```

这一个不带任何参数的 myReader, 可以处理从服务器端返回的下面 JSON 数据:

```
{
  'metaData': {
    totalProperty: 'results',
    root: 'rows',
    id: 'id',
    fields: [
      {name: 'title'},
      {name: 'username', mapping: 'author'},
      {name: 'loginTimes', type: 'int'} ]
    },
  'results': 2, 'rows': [
    { id: 1, title: '测试', author: '小王', loginTimes: 3 },
    { id: 2, title: '新年好', author: 'williamraym', loginTimes:13} ]
}
```

3) XmlReader

Ext.data.XmlReader—XML 文档数据解析器, 用于把 XML 文档数据转换成记录集 Record 对象。看下面的代码:

```
var MyRecord = Ext.data.Record.create([
  {name: 'title'},
  {name: 'username', mapping: 'author'},
  {name: 'loginTimes', type: 'int'}
]);
var myReader = new Ext.data.XmlReader({
  totalRecords: "results",
  record: "rows",
  id: "id"
}, MyRecord);
```

上面的 myReader 能够解析下面的 xml 文档信息:

```
<topics>
  <results>2</results>
  <row>
    <id>1</id>
    <title>测试</ title >
    <author>小王</ author >
    <loginTimes>3</ loginTimes >
  </row>
  <row>
    <id>2</id>
    <title>新年好</ title >
    <author> williamraym </ author >
    <loginTimes>13</ loginTimes >
  </row>
</topics>
```

8.4、DataProxy 与自定义 Store

DataProxy 字面解释就是数据代理，也可以理解为数据源，也即从哪儿或如何得到需要交给 DataReader 解析的数据。数据代理（源）基类由 Ext.data.DataProxy 定义，在 DataProxy 的基础，ExtJS 提供了 Ext.data.MemoryProxy、Ext.data.HttpProxy、Ext.data.ScriptTagProxy 等三个分别用于从客户端内存数据、Ajax 读取服务器端的数据及从跨域服务器中读取数据等三种实现。

比如像 SimpleStore 等存储器是直接从从客户端的内存数组中读取数据，此时就可以直接使用 Ext.data.MemoryProxy，而大多数需要从服务器端加载的数据直接使用 Ext.data.HttpProxy，HttpProxy 直接使用 Ext.Ajax 加载服务器的数据，由于这种请求是不能跨域的，所以要读取跨域服务器中的数据时就需要使用到 Ext.data.ScriptTagProxy。

关于 DataProxy 的更多内容，请参考 <http://wlr.easyjf.com> 的 VIP 文档中的《ExtJS 数据存储 Store 详解》中的相关内容。

在实际应用中，除了基本的从内存中读取 javascript 数组对象，从服务器读取 JSON 数组，从服务器取 xml 文档等形式的数据外，有时候还需要使用其它的数据读取方式。比如熟悉 EasyJWeb 中远程 Web 脚本调用引擎或 DWR 等框架的都知道，通过这些框架我们可以直接在客户端使用 javascript 调用服务器端业务组件的方法，并把服务器端的结果返回到客户端，客户端得到的是一个 javascript 对象或数组。由于这种方式的调用是异步的，因此，相对来说有点特殊，即不能直接使用 Ext.data.MemoryProxy，也不能直接使用 Ext.data.HttpProxy，当然更不需要 Ext.data.ScriptTagProxy，这时候就需要创建自定义的 DataProxy 及 Store，然后使用这个自定义的 Store 来实现这种基于远程脚本调用引擎的框架

得到数据。

第九章、TreePanel

9.1、TreePanel 之基本使用

在应用程序中，我们经常会涉及到要显示或处理树状结构的对象信息，比如部门信息、地区信息，或者是树状的菜单信息，操作系统中的文件夹信息等。

对于传统的 html 页面来说，要自己实现显示树比较困难，需要写很多的 javascript，特别是对于基于 Ajax 异步加载的树来说，不但涉及到 Ajax 数据加载及处理技术，还需要考虑跨浏览器支持等，处理起来非常麻烦。ExtJS 中提供了现存的树控件，通过这些控件可以在 B/S 应用中快速开发出包含树结构信息的应用。

TreePanel基本使用

树控件由 **Ext.tree.TreePanel** 类定义，控件的名称为 **treepanel**，TreePanel 类继承自 Panel 面板。在 ExtJS 中使用树控件其实非常简单，我们先来看下面的代码

```
Ext.onReady(function(){
    var root=new Ext.tree.TreeNode({
        id:"root",
        text:"树的根"});
    root.appendChild(new Ext.tree.TreeNode({
        id:"c1",
        text:"子节点"
    }));
    var tree=new Ext.tree.TreePanel({
        renderTo:"hello",
        root:root,
        width:100
    });
});
```

代码的第一句使用 **new Ext.tree.TreeNode** 类来创建一个树节点，第二句使用树节点的 **root** 的 **appendChild** 方法来往该节点中加入一个子节点，最后直接使用 **new Ext.tree.TreePanel** 来创建一个树面板，要树面板的初始化参数中指定树的 **root** 属性值为前面创建的 **root** 节点，也就是树根节点。上面的程序执行效果如下图所示：

树的节点信息。ExtJS 的树控件提供了对这种功能的支持，你只需要在创建树控件的时候，通过给树指定一个节点加载器，可以用来从服务器端动态加载树的节点信息。我们来看下面的代码：

```
var root=new Ext.tree.AsyncTreeNode({
    id:"root",
    text:"树的根"});
var tree=new Ext.tree.TreePanel({
    renderTo:"hello",
    root:root,
    loader: new Ext.tree.TreeLoader({url:"treedata.js"}),
    width:100
});
```

treedata.js 这个 url 返回的内容如下:

```
[{
    id: 1,
    text: '子节点1',
    leaf: true
},{
    id: 2,
    text: '儿子节点2',
    children: [{
        id: 3,
        text: '孙子节点',
        leaf: true
    }]
}]
```

执行上面的程序，可以得到一棵异步加载子节点的树，点击“根节点”会到服务器端加载子节点，如下图所示：

当然上面的程序是一次性加载完了树的所有节点信息，我们也可以实现让每一个节点都支持动态加载的树，只需要在通过服务器请求数据的时候，每次服务器端返回的数据只只包含子节点，而不用把孙子节点也返回即可。比如把上面 treedata.js 中的内容改为下面的内容：

```
[{
    id: 1,
    text: '子节点',
    leaf: false
}]
```

也就是节点树中只包含一个子节点，而该子节点通过指定 leaf 值为 false (默认情况该值为 false)，表示该节点不是一个叶子节点，其下面还有子节点。再执行前面的程序，不断点击“子节点”可以得到如下图所示的效果：

当然这是一个无限循环的树，在实际应用中我们服务器端返回的数据是程序动态产生

的，因此不可能每一次都产生 leaf 为 false 的节点，如果是叶子节点的时候，则需要把返回的 JOSN 对象中的 leaf 设置为 true。如下所示：

```
{
  id: 1,
  text: '子节点',
  leaf:true
}
```

事件处理

当然，仅仅能显示一棵树还不够，我们一般还需要在用户点击树节点的时候执行相应的东西，比如打开某一个连接，执行某一个函数等，这就需要使用到事件处理。比如下面的代码：

```
Ext.onReady(function(){
  var root=new Ext.tree.TreeNode({
    id:"root",
    text:"树的根"});
  var c1=new Ext.tree.TreeNode({
    id:"c1",
    text:"子节点"
  });
  root.appendChild(c1);
  var tree=new Ext.tree.TreePanel({
    renderTo:"hello",
    root:root,
    width:100
  });
  tree.on("click",function(node,event){
    alert("您点击了"+node.text);
  }
  );
  c1.on("click",function(node,event){
    alert("您点击了"+node.text);
  }
  );
});
```

执行上面的程序，当用户点击树控件中的任意节点时，都会弹出一个提示信息框，当用户点击 c1 这个子节点时，会弹出两次提示信息框。因为我们除了指定 tree 的 click 事件响应函数以外，另外又给 node 节点指定单独的事件响应函数。

当然，如果只是要实现当点击树节点时跳到某一个指定 url 的功能则非常简单。看下面

的代码:

```
Ext.onReady(function(){
    var root=new Ext.tree.TreeNode({
        id:"root",
        href:"http://www.easyjf.com",
        hrefTarget:"_blank",
        text:"树的根"});
    var c1=new Ext.tree.TreeNode({
        id:"c1",
        href:"http://wlr.easyjf.com",
        hrefTarget:"_blank",
        text:"子节点"
    });
    root.appendChild(c1);
    var tree=new Ext.tree.TreePanel({
        renderTo:"hello",
        root:root,
        width:100
    });

});
```

执行程序, 点击树节点, 将会在浏览新窗口中打开节点中 href 指定的链接。

9.2、TreeNode

在 ExtJS 中, 不管是叶子节点还是非叶子节点, 都统一用 **TreeNode** 表表示树的节点。在 ExtJS 中, 有两种类型的树节点。一种节点是普通的简单树节点, 由 **Ext.tree.TreeNode** 定义, 另外一种是需要**异步加载子节点信息的树节点**, 该类由 **Ext.tree.AsyncTreeNode** 定义。看下面的代码:

```
Ext.onReady(function(){
    var tree=new Ext.tree.TreePanel({
        renderTo:"hello",
        root:new Ext.tree.AsyncTreeNode({
            text:"根节点"
        }),
        width:100
    });

});
```

执行程序, 点击树中的“根节点”则会一直发现树会尝试加载这个节点的子节点, 由这

里没有指定树的加载器，所以“根节点”会变成一直处于加载的状态。如下图所示：

对于普通的 `TreeNode` 来说，可以通过调用节点的 `appendChild`、`removeChild` 等方法来往该节点中加入子节点或删除子节点等操作。

`TreeNode` 与 `AsyncTreeNode` 可以同时使用，比如下面的代码：

```
Ext.onReady(function(){

    var root=new Ext.tree.TreeNode({
        id:"root",
        text:"树的根"
    });
    var c1=new Ext.tree.TreeNode({
        text:"子节点1"
    })
    var c2=new Ext.tree.AsyncTreeNode({
        text:"子节点2"
    });
    root.appendChild(c1);
    root.appendChild(c2);
    var tree=new Ext.tree.TreePanel({
        renderTo:"hello",
        root:root,
        width:300,
        loader:new Ext.tree.TreeLoader({
            applyLoader:false,
            url:"treedata.js"
        })
    });

});
```

`treedata.js` 中的内容仍然是：

```
{
    id: 1,
    text: '子节点'
}
```

执行上面的程序可以得到一棵如下图所示的树：

另外要在树以外的程序中得到当前选择的节点，可以通过 TreePanel 的 getSelectionModel 方法来获得，该方法默认返回的是 Ext.tree.DefaultSelectionModel 对象，DefaultSelectionModel 的 getSelectedNode 方法返回当前选择的树节点。比如要得到树 tree 中当前选择节点，代码如下：

```
tree.getSelectionModel().getSelectedNode()
```

9.3、TreeLoader

对于 ExtJS 中的树来说，树加载器 TreeLoader 是一个比较关键的部件，树加载器由 Ext.tree.TreeLoader 类定义，只有 AsyncTreeNode 才会使用 TreeLoader。看下面的代码：

```
Ext.onReady(function(){
    var loader=new Ext.tree.TreeLoader({
        url:"treedata.js"
    });
    var root=new Ext.tree.AsyncTreeNode({
        id:"root",
        text:"根节点",
        loader:loader});
    var tree=new Ext.tree.TreePanel({
        renderTo:"hello",
        root:root,
        width:100
    });
});
```

首先我们使用 Ext.tree.TreeLoader 来初始化了一个 TreeLoader 对象，构造函数中的配置参数 url 表示获得树节点信息的 url。然后在初始化根节点的时候我们使用的是 AsyncTreeNode，在该节点中指定该节点的 loader 为前面定义的 loader。执行这段程序，在点击“根节点”时，会从服务器端指定 root 节点的子节点信息。

TreeLoader 严格来说是针对树的节点来定义的，可以给树中的每一个节点定义不同的 TreeLoader，默认情况下，如果一个 AsyncTreeNode 节点在准备加载子节点的时候，如果该节点上没有定义 loader，则会使用 TreePanel 中定义的 loader 作为加载器。因此，我们可以直接在 TreePanel 上面指定 loader 属性，这样就不需要给每一个节点指定具体的 TreeLoader 了。因此，上面的代码可以改成如下所示的内容：

9.4 自定义 TreeLoader

在 ExtJS 自己的 TreeLoader 中，当要实现从远程服务器端异步加载树节点信息的时候，都是通过请求服务器上的某一个 URL 来进行的，这个 URL 返回下面的信息：

```
[{
  id: 1,
  text: 'A leaf Node',
  leaf: true
},{
  id: 2,
  text: 'A folder Node',
  children: [{
    id: 3,
    text: 'A child Node',
    leaf: true
  }]
}]
```

假如我们是直接通过类似 DWR 或 EasyJWeb 的远程脚本引擎在客户端直接调用服务器的业务方法，直接跳过了 WEB（不需要 Struts、JSP 或其它 Web 层的代码）这一层，这时我们没有 URL，这时该怎么办呢？这就需要用到自定义的 TreeLoader，下面我们通过一个实例来做简单的讲解。

看服务器端的 ITopicCategoryService

```
public interface ITopicCategoryService {
    List loadCategory(Long id);
}
```

loadCategory 方法返回一个类型为 Node 的列表，也就是返回指定 id 的下级分类节点信息，Node 对应树节点的信息，代码如下：

```
public class Node {
    private TopicCategory category;
    Node(TopicCategory category) {
        this.category = category;
    }
    public String getId() {
        return category.getId().toString();
    }
    public boolean getLeaf() {
```

```
return category.getChildren().size() < 1;
}
public String getText() {
    return category.getName();
}
public String getQtip() {
    return category.getName();
}
}
```

Node 在这里相当于一个简单适配器，其实就是把数据库中的日志分类实体适配成包树节点对象。

把 **ITopicCategoryService** 发布成可供客户端远程调用，使用 EasyJWeb 的话引如下面三个 js:

```
<script type="text/javascript" src="/ejf/easyajax/prototype.js"></script>
<script type="text/javascript" src="/ejf/easyajax/engine.js"></script>
<script type="text/javascript" src="/ejf/easyajax/topicCategoryService.js"></script>
```

使用 DWR 的话引入下面的两个 js:

```
<script type="text/javascript" src="/dwr/dwr/engine.js "></script>
<script type="text/javascript" src="/dwr/dwr/util.js "></script>
<script type="text/javascript" src="/dwr/dwr/interface/ topicCategoryService.js "></script>
```

这样我们可以在页使用下面的 javascript 来从服务器端获得某一个节点的子节点信息，代码如下:

```
function test()
{
    topicCategoryService.loadCategory(1,function(ret)
    {
        alert("一共有"+ret.length+"个子节点");
    }
}
```

如何让 ExtJS 的树面板能通过这个远程 web 脚本方法 topicCategoryService.loadCategory 来加载异步加载树节点信息呢？其实很简单，跟一般的使用没什么两样，树面板 TreePanel 的代码如下:

```
var tree = new Ext.tree.TreePanel({
    autoScroll:true,
    animate:true,
    width:'100px',
    height:'300px',
    enableDD:true,
    containerScroll: true,
    loader: loader
    root: new Ext.tree.AsyncTreeNode({
        text: '日志分类',
        id:'root'
    });
});
```

然后区别是在 loader 部分，使用远程 Web 调用来加载树节点的 loader，代码如下：

```
var loader=new WebInvokeTreeLoader({
    fn:topicCategoryService.loadCategory
});
loader.on("beforeload",function(l,node){
    l.args[0]=(node.id!='root'?node.id:"-1");
});
```

再回顾一下传统的直接通过 url 加载树节点的 TreeLoader 代码，如下所示：

```
var loader=new Ext.tree.TreeLoader({
    url:'/topicCategory.ejff?cmd=getCategory&pageSize=-1&treeData=true'
});
loader.on("beforeloader",function(loader,node){
    loader.baseParams.id=(node.id!='root'?node.id:"");
});
```

区别在于，远程脚本调用方式加载树节点信息使用的是 **WebInvokeTreeLoader**，需要通过 fn 属性来指定用于加载数据的远程方法，并在 beforeload 事件处理器设置参数远程方法调用的参数值。而传统的树节点加载器是 Ext.tree.TreeLoader，需要指定一个 url 来获得 json 数据。

WebInvokeTreeLoader 是自定义的树加载器，代码其实比较简单，你可以自己写一个。本方案仅供参考，关于 WebInvokeTreeLoader 的源代码我已经传到了我用 ExtJS 开发的 Blog 示例网站上了，仅供 VIP 会员浏览，有兴趣的朋友可跟我联系。