

## 第 1 章 闪烁吧！看看 extjs 那些美丽的例子。

### 1.1. 一切从 extjs 发布包开始

非常幸运的是，我们可以免费去 [extjs.com](http://www.extjs.com) 下载 ext 发布包，里边源代码，api 文档，例子一应俱全。不过要是想访问 svn 获得最新的代码，就要花钱了。不过我们现阶段只要这个免费的发布包就可以了，通过里边的范例，可以让我们体验一下 ext 的风范。

下载地址：<http://www.extjs.com/download>。到写文档的此时此刻，咱们可以选择 ext-1.1.1 或者是 ext-2.0 下载。明显可以看出来 ext-2.0 的版本高，12 月 4 日终于正式发布了，尚未经过详细测试，所以不敢说什么。下面我们把两个版本都介绍一点儿。

### 1.2. 看看 ext-1.1.1 的文档

docs 目录下是 api 文档，基本上所有的函数，配置，事件都可以在里边找到，直接打开 index.html 就可以查看，左侧菜单还包含了对 examples 目录下例子的引用，不过有些例子需要使用 json 与后台做数据交换，必须放到服务器上才能看到效果。还有一些后台代码是使用 php 编写的，如果想看这些例子的效果，还需要配置 php 运行环境。

如果你用 java，而且 jdk 在 1.5 以上，不如直接装个 resin-3 方便，它可以跑 php 呢。

### 1.3. 看看 ext-2.0 的文档

api 文档依然在 docs 目录下，虽然可以看到左边的菜单，但是点击之后，右侧的 api 页面都是靠 ajax 获得的，所以不能直接在本地上查看了，你必须把整个解压缩后的目录放到服务器上，通过浏览器访问服务器，才能看到。

问为什么 docs 打不开，只能看到一直 loading 的兄弟，都是因为没把这些东西放到服务器上的原因。

2.0 中的 api 文档中没有例子的链接了，你需要自己去 examples 目录下找你需要例子，然后打开看，当然还是有一些例子需要放在服务器上才能看到效果。

## 1.4. 为什么有的例子必须放在服务器上才能看到效果？

因为有些例子里，用到 Ajax 去后台读取数据，如果没在服务器上，Ajax 返回的状态一直是失败，也无法获得任何数据，所以就看不到正确的效果。不过以前在 extjs.com 论坛上看到过有人写了 localXHR，可以让 ajax 从本地文件系统获得数据，这样也许就可以摆脱服务器的束缚了。

## 1.5. 为什么自己按照例子写的代码，显示出来总找不到图片

ext 里经常用一个空白图片做占位符号，然后用 css 里配置的背景图片做显示，这样有利于更换主题。这个空白图片的名字就是 Ext.BLANK\_IMAGE\_URL，默认情况下它的值是 BLANK\_IMAGE\_URL : "http://"+"/extjs.com/s.gif"。虽然图片很小，也要去网上下载，一旦下载失败就显示找不到图片了。

看到这里可能有人奇怪了，为什么 examples 下的例子没有找不到图片的问题呢？看来你没有仔细看例子那些代码呢，每个例子都引用了 ../examples.js。在 examples.js 里设置了 Ext.BLANK\_IMAGE\_URL = '../resources/images/default/s.gif'；。所以要解决自己写的例子找不到图片的问题，只需要照 examples.js 里修改 s.gif 的本地路径就可以了。很简单吧？

## 1.6. 我们还需要什么？

- 介于本人对 firefox 的喜爱，以及 firebug 在调试 js 过程中的便利，隆重向您推荐 firefox+firebug 的开发组合。再说了 ext 开发者也都是倾向于 firefox 开发的，所以一般都是在 firefox 上跑的好好的，放到 ie 上就出问题。这也跟 ie 自身的问题有些关系，可是目前 ie 占据 90% 的浏览器市场，最后我们还是需要让自己的项目在 ie 上跑起来，所以要求我们能写出跨浏览器的 js 来。

firebug 的好处在于，可以显示动态生成的 dom，你甚至可以在 firebug 里直接对 dom 进行修改，而这些修改会直接反应到显示上。太厉害了

firebug 提供的 console，可以直接执行 js 脚本，配置 console.debug，console.info，console.error 等日志方法更便于跟踪。

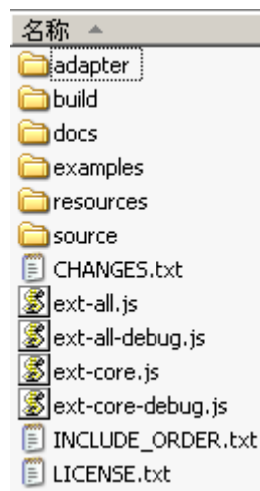
对于 ajax 发送接收的数据，firebug 都可以显示出来，并且可以查看发送的参数，以及返回的状态和信息。

## 1.7. 入门之前，都看 helloworld。

为了照顾连最基本应用都跑不起来的同志，我们给出两个入门版 helloworld 范例，并结合讲解，领你入门呢。

### 1.7.1. 直接使用下载的发布包

- 先去<http://www.extjs.com/download>下载zip格式的发布包
- 随便解压缩到什么目录下，不管目录名是什么，最后应该看到里边是这样的目录结构。



- 现在我们利用它的目录结构，写一个 helloworld 例子。

进入上图中的 examples 目录，新建一个 helloworld 目录，helloworld 目录下新建一个 helloworld.html 文件，将下列内容复制进 index.html 文件中。

```
<link rel="stylesheet" type="text/css"
href="../../../resources/css/ext-all.css" />
<script type="text/javascript"
src="../../../adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../../../ext-all.js"></script>
<script type="text/javascript" src="../../examples.js"></script>

<script>
Ext.onReady(function() {
    Ext.MessageBox.alert('helloworld', 'Hello World.');
```

- 双击 helloworld.html 打开页面，效果如下：



很高兴的告诉您，咱们的 helloworld 范例已经正确的执行了，下一步你最好把 examples 目录下的例子都看一看，再看看里边的代码怎么写的，好好感受一下 ext 的风范，再继续下去。

### 1.7.2. 只把必要的东西放进项目中

想把 ext 放入自己的项目，需要自己整理一下，因为发布包里的东西并非都是必要的，比如文档，比如例子，比如源代码。

必要的最小集合是这样：ext-all.js，adapter/ext/ext-base.js，build/locale/ext-lang-zh\_CN.js 和整个 resources 目录。

- ext-all.js，adapter/ext/ext-base.js 就包含了 ext 的所有功能，所有的 js 脚本都在这里了。
- build/locale/ext-lang-zh\_CN.js 是中文翻译。
- resources 目录下是 css 样式表和图片。

自己的项目里只需要包含这些东西，就可以使用 ext 了。使用时，在页面中导入：

```
<link rel="stylesheet" type="text/css" href="{放置 ext 的目录}/resources/css/ext-all.css" />
<script type="text/javascript" src="{放置 ext 的目录}/ext-base.js"></script>
<script type="text/javascript" src="{放置 ext 的目录}/ext-all.js"></script>
<script type="text/javascript" src="{放置 ext 的目录}/ext-lang-zh_CN.js"></script>
```

请注意 js 脚本的导入顺序，其他的就可以自由发挥了。

## 第 2 章 震撼吧！让你知道 ext 表格控件的厉害。

### 2.1. 功能丰富，无人能出其右

无论是界面之美，还是功能之强，ext 的表格控件都高居榜首。

单选行，多选行，高亮显示选中的行，推拽改变列宽度，按列排序，这些基本功能咱们就不提了。

自动生成行号，支持 checkbox 全选，动态选择显示哪些列，支持本地以及远程分页，可以对单元格按照自己的想法进行渲染，这些也算可以想到的功能。

再加上可编辑 grid，添加新行，删除一或多行，提示脏数据，推拽改变 grid 大小，grid 之间推拽一或多行，甚至可以在 tree 和 grid 之间进行拖拽，啊，这些功能实在太神奇了。更令人惊叹的是，这些功能竟然都在 ext 表格控件里实现了。

呵呵~不过 ext 也不是万能的，与 fins 的 ecside 比较，ext 不能锁定列（土豆说 1.x 里支持锁定列，但是 2.0 里没有了，因为影响效率。），也没有默认的统一功能，也不支持 excel, pdf 等导出数据。另外 fins 说，通过测试 ecside 的效率明显优于 ext 呢。:)

### 2.2. 让我们搞一个 grid 出来耍耍吧。

光说不练不是我们的传统，让我们基于 examples 里的例子，来自己搞一个 grid 看看效果，同时也可以知道一个 grid 到底需要配置些什么东西。

- 首先我们知道表格肯定是二维的，横着叫行，竖着叫列。跟设计数据库，新建表一样，我们要先设置这个表有几列，每列叫啥名字，啥类型，咋显示，这个表格的骨架也就出来了。

ext 里，这个列的定义，叫做 ColumnModel，简称 cm 的就是它，它作为整个表格的列模型，是要首先建立起来的。

这里我们建立一个三列的表格，第一列叫编号(code)，第二列叫名称(name)，第三列叫描述(descn)。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号', dataIndex:'id'},
    {header:'名称', dataIndex:'name'},
```

```
    {header:'描述', dataIndex:'descn' }  
  ]);
```

看到了吧？非常简单的定义了三列，每列的 header 表示这列的名称，dataIndex 是跟后面的东西对应的，咱们暂且不提。现在只要知道有了三列就可以了。

- 有了表格的骨架，现在我们要向里边添加数据了。这个数据当然也是二维了，为了简便，我们学习 examples 里的 array-grid.js 里的方式，把数据直接写到 js 里。

```
• var data = [  
•   ['1', 'name1', 'descn1'],  
•   ['2', 'name2', 'descn2'],  
•   ['3', 'name3', 'descn3'],  
•   ['4', 'name4', 'descn4'],  
•   ['5', 'name5', 'descn5']  
• ];
```

很显然，我们这里定义了一个二维数据，（什么？你不知道这是二维数组？快改行吧，这里不是你该待的地方。）

这个有五条记录的二维数组，显示到 grid 里就应该是五行，每行三列，正好对应这 id, name, descn，在我们的脑子里应该可以想像出 grid 显示的结果了，为了让想像变成显示，我们还需要对原始数据做一下转化。

- 因为咱们希望 grid 不只能支持 array，还可以支持 json，支持 xml，甚至支持咱们自己定义的数据格式，ext 为咱们提供了一个桥梁，Ext.data.Store，通过它我们可以把任何格式的数据转化成 grid 可以使用的形式，这样就不需要为每种数据格式写一个 grid 的实现了。现在咱们就来看看这个 Ext.data.Store 是如何转换 array 的。

```
• var ds = new Ext.data.Store({  
•   proxy: new Ext.data.MemoryProxy(data),  
•   reader: new Ext.data.ArrayReader({}, [  
•     {name: 'id'},  
•     {name: 'name'},  
•     {name: 'descn'}  
•   ])  
• });  
• ds.load();
```

ds 要对应两个部分：proxy 和 reader。proxy 告诉我们从哪里获得数据，reader 告诉我们如何解析这个数据。

现在我们用的是 Ext.data.MemoryProxy，它是专门用来解析 js 变量的。你可以看到，我们直接把 data 作为参数传递进去了。

Ext.data.ArrayReader 专门用来解析数组，并且告诉我们它会按照定义的规范进行解析，每行读取三个数据，第一个叫 id，第二个叫 name，第三个 descn。是不是有些眼熟，翻到前面 cm 定义的地方，哦，原来跟 dataIndex 是对应的。这样 cm 就知道哪列应该显示那条数据了。唉，你要是能看明白这一点，那你实在是太聪明了。

记得要执行一次 ds.load()，对数据进行初始化。

有兄弟可能要问了，要是我第一列数据不是 id 而是 name，第二列数据不是 name 而是 id 咋办？嗯，嗯，这个使用就用 mapping 来解决。改改变成这样：

```
var ds = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id', mapping: 1},
        {name: 'name', mapping: 0},
        {name: 'descn', mapping: 2}
    ])
});
```

编号	名称	描述
name1	1	descn1
name2	2	descn2
name3	3	descn3
name4	4	descn4
name5	5	descn5

这样如截图所见，id 和 name 两列的数据翻转了。如此这般，无论数据排列顺序如何，我们都可以使用 mapping 来控制对应关系，唯一需要注意的是，索引是从 0 开始的，所以对应第一列要写成 mapping:0，以此类推。

- 哈哈，万事俱备只欠东风，表格的列模型定义好了，原始数据和数据的转换都做好了，剩下的只需要装配在一起，我们的 grid 就出来了。

```
var grid = new Ext.grid.Grid('grid', {
    ds: ds,
```

- cm: cm
- });
- grid.render();

注意:上头是 ext-1.x 的写法,Ext.grid.Grid 的第一个参数是渲染的 id, 对应 html 里应该有一个 <div id="grid"></div> 的东西, 这样 grid 才知道要把自己画到哪里。

创建完 grid 以后, 还要用 grid.render() 方法, 让 grid 开始渲染, 这样才能显示出来。

- 好了, 把所有代码组合到一起, 看看效果吧。

```

•   var cm = new Ext.grid.ColumnModel([
•       {header:' 编号', dataIndex:' id' },
•       {header:' 名称', dataIndex:' name' },
•       {header:' 描述', dataIndex:' descn' }
•   ]);
•
•   var data = [
•       [' 1', ' name1', ' descn1' ],
•       [' 2', ' name2', ' descn2' ],
•       [' 3', ' name3', ' descn3' ],
•       [' 4', ' name4', ' descn4' ],
•       [' 5', ' name5', ' descn5' ]
•   ];
•
•   var ds = new Ext.data.Store({
•       proxy: new Ext.data.MemoryProxy(data),
•       reader: new Ext.data.ArrayReader({}, [
•           {name: ' id' },
•           {name: ' name' },
•           {name: ' descn' }
•       ])
•   });
•   ds.load();
•
•   var grid = new Ext.grid.Grid(' grid', {
•       ds: ds,
•       cm: cm
•   });
•   grid.render();
•

```



看看吧，这就是咱们搞出来的 grid 了。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

html 例子是 lingo-sample/1.1.1 目录下的 02-01.html，把这个目录 copy 到 ext-1.x 的 example 目录下，就可以直接打开观看效果。

## 2.3. 上边那个是 1.x 的，2.0 稍微有些不同哦

首先，Ext.grid.Grid 已经不见了，咱们需要用 Ext.grid.GridPanel。需要传递的参数也有少许区别。

```
var grid = new Ext.grid.GridPanel({  
    el: 'grid',  
    ds: ds,  
    cm: cm  
});
```

看到了吗？负责指定渲染位置的 id 放到了 {} 里边，对应的名字是 el。似乎 ext2 里对这些参数进行了统一，比以前更整齐了。

因为其他地方都一样，我就不多说了，html 例子在是 lingo-sample/2.0 目录下的 02-01.html。

名称	编号	描述
name1	1	descn1
name2	2	descn2
name3	3	descn3
name4	4	descn4
name5	5	descn5

从截图上，少了斑马条，下边多了一条线，应该只是 css 有所不同吧。

默认情况下，两个版本的 grid 都可以拖拽列，也可以改变列的宽度。不知道怎么禁用这两个功能呢。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

最大的不同应该是 1.x 里默认支持的右键效果，在 2.0 里不见了。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

按 shift 和 ctrl 多选行的功能倒是都有。区别是，全选后，1.x 必须按住 ctrl 才能取消，直接单击其中一个行，不会取消全选功能，而 2.0 里只需要任意点击一行，就取消全选，只选中刚才点击的那行。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

名称	描述	编号
name1	descn1	1
name2	descn2	2
name3	descn3	3
name4	descn4	4
name5	descn5	5

哦，哦，那颜色不要也算是区别吧。

## 2.4. 按顺序，咱们先要把常见功能讲到，让grid支持按列排序

其实很简单，需要小小改动一下列模型。

```
var cm = new Ext.grid.ColumnModel([
    {header:' 编号', dataIndex:' id', sortable:true},
    {header:' 名称', dataIndex:' name'},
    {header:' 描述', dataIndex:' descn' }
]);
```

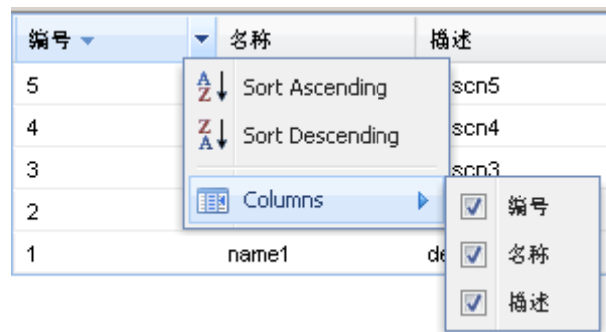
如果你英语还可以，或者懂得查字典的话（软件翻译也算），那么你就会知道，多出来的这个 `sortable` 属性应该是可以排序的意思。现在咱们试一下改动后的效果。

编号 ▼	名称	描述
5	name5	descn5
4	name4	descn4
3	name3	descn3
2	name2	descn2
1	name1	descn1

编号 ▼	名称	描述
5	name5	descn5
4	name4	descn4
3	name3	descn3
2	name2	descn2
1	name1	descn1

看到了没有？编号的标题上有个小小的箭头，表格里数据也是按照编号做的逆序排列，如此简单，我们就实现了按列排序。

很有趣的是，2.0 加上 sortable 以后，1.x 那种右键功能也跑回来了，不过它用的不是右键，而是下拉菜单似的实现方式。



什么？你问为什么其他两列无法排序？！嗯，好像是因为你还没有给另两列添加 sortable 属性。

怎么加？！按编号那样加就行了。

还是不会？！-\_-。

```
var cm = new Ext.grid.ColumnModel([
    {header:' 编号', dataIndex:' id', sortable:true},
    {header:' 名称', dataIndex:' name', sortable:true},
    {header:' 描述', dataIndex:' descn', sortable:true}
]);
```

这样所有列都可以排序了。什么？怎么取消排序？！-\_-。

## 2.5. 让单元格里显示红色的字，图片，按钮，你还能想到什么？

嘿，希望你跟我一样，不愿意只能在 grid 里看到文字，至少不是单调的，毫无特色的文字。有些人就问了，如果我想改变一下单元格里显示内容，应该怎么办呢？

非常不幸的是，ext 的作者，伟大的 jack 早已经想到了，说真的，你没想到的，他都想到了，不只想到了，他还做出来了。

唉，这就是区别啊。为啥你就不能动手做些东西呢？就知道向别人要这要那，唉。

首先，我宣布，偶们的数据要扩充啦，每个人要加上一个性别字段。

```
var data = [
    ['1', 'male', 'name1', 'descn1'],
    ['2', 'female', 'name2', 'descn2'],
    ['3', 'male', 'name3', 'descn3'],
    ['4', 'female', 'name4', 'descn4'],
    ['5', 'male', 'name5', 'descn5']
];
```

男女搭配，干活不累撒。而且现在中国就是男多女少，我就还没对象呢。征婚中，单身女性加(QQ)771490531 详谈。

你可以试试不改其他的部分，显示的结果是不会改变的，因为原始数据要经过 ds 的处理才能被 grid 使用，那么下一步我们就开始修改 ds，把性别加进去。

```
var ds = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'sex'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
```

添加了一行 {name: 'sex'}，把数组的第二列映射为性别。现在 grid 可以感觉到 sex 了，嘿嘿。

不过 grid 还显示不了性别这列，因为咱们还没改 cm。

```
var cm = new Ext.grid.ColumnModel([
    {header: '编号', dataIndex: 'id'},
    {header: '性别', dataIndex: 'sex'},
    {header: '名称', dataIndex: 'name'},
    {header: '描述', dataIndex: 'descn'}
]);
```

到现在其实都没什么新东西，但是你不觉得光看平板字，很难分出哪个是 GG 哪个是 MM 吗？听说过红男绿女没？要是男的都加上红色，女的都变成绿色，那不是清楚多了。就像下面一样。

编号	性别	名称	描述
1	红男	name1	descn1
2	绿女	name2	descn2
3	红男	name3	descn3
4	绿女	name4	descn4
5	红男	name5	descn5

怎么样？是不是效果大不同了。你不会认为很难吧，嗯，确实，如果你对 html 和 css 完全搞不明白的话，劝你还是先去学学吧，对自己有信心的往下看。

```
var cm = new Ext.grid.ColumnModel([
    {header:' 编号', dataIndex:' id'},
    {header:' 性别', dataIndex:' sex', renderer:function(value) {
        if (value == 'male') {
            return "<span style=' color:red;font-weight:bold;'>红男
</span>";
        } else {
            return "<span style=' color:green;font-weight:bold;'>
绿女</span>";
        }
    }},
    {header:' 名称', dataIndex:' name'},
    {header:' 描述', dataIndex:' descn'}
]);
```

别被吓到，这么一大段其实就是判断是男是女，然后配上颜色。你要是觉得乱，也可以这么做。

```
function renderSex(value) {
    if (value == 'male') {
        return "<span style=' color:red;font-weight:bold;'>红男
</span>";
    } else {
        return "<span style=' color:green;font-weight:bold;'>绿女
</span>";
    }
}

var cm = new Ext.grid.ColumnModel([
```

```

    {header:' 编号', dataIndex:' id' },
    {header:' 性别', dataIndex:' sex', renderer:renderSex},
    {header:' 名称', dataIndex:' name' },
    {header:' 描述', dataIndex:' descn' }
  ]);





```

实际上这个 renderer 属性至关重要，它的值是一个 function，哦，你说不知道 js 里 function 可以这么用？那么恭喜你，现在你知道了。

renderer 会传递个参数进去，咱们 grid 里看到的，是这个函数的返回值，怎么样，神奇吧？

同志们，你们也应该看到了，返回 html 就可以，是 html 啊，html 里有的东西，你返回什么就显示什么，颜色，链接，图片，按钮，只要你愿意，整个网页都可以返回回去。还有什么做不到的？哦，你不会 html，那没辙，回去学吧。

咱们先来张图片。

编号	性别	名称	描述
1	红男 	name1	descn1
2	绿女 	name2	descn2
3	红男 	name3	descn3
4	绿女 	name4	descn4
5	红男 	name5	descn5

```

function renderSex(value) {
  if (value == 'male') {
    return "<span style=' color:red;font-weight:bold;' >红男</span><img src='user_male.png' />";
  } else {
    return "<span style=' color:green;font-weight:bold;' >绿女</span><img src='user_female.png' />";
  }
}

```

是不是太简单了，下面咱们来玩点儿高级的。

```

function renderDescn(value, cellmeta, record, rowIndex, columnIndex, store) {

```

```

    var str = "<input type='button' value=' 查看详细信息'
onclick='alert(\"" +
    "这个单元格的值是: " + value + "\\n" +
    "这个单元格的配置是: {cellId:" + cellmeta.cellId + ",id:" +
cellmeta.id + ",css:" + cellmeta.css + "}\n" +
    "这个单元格对应行的 record 是: " + record + ", 一行的数据都在里
边\n" +
    "这是第" + rowIndex + "行\n" +
    "这是第" + columnIndex + "列\n" +
    "这个表格对应的 Ext. data. Store 在这里: " + store + ", 随使用吧。
" +
    "\")'>";
    return str;
}

```

来看看我们可以在 render 里用到多少参数:

- value 是当前单元格的值
- cellmeta 里保存的是 cellId 单元格 id, id 不知道是干啥的, 似乎是列号, css 是这个单元格的 css 样式。
- record 是这行的所有数据, 你想要什么, record.data["id"]这样就获得了。
- rowIndex 是行号, 不是从头往下数的意思, 而是计算了分页以后的结果。
- columnIndex 列号太简单了。
- store, 这个厉害, 实际上这个是你构造表格时候传递的 ds, 也就是说表格里所有的数据, 你都可以随便调用, 唉, 太厉害了。

有个同学就问啦: EXT render 的参数, 是如何得到的呢。因为你讲的那些都是 EXT 自己内部的。它是如何把这些参数传递给 render 的呢?

这个问题其实比较简单, 只是你们想复杂了。既然是函数, 就肯定有调用它的地方, 你找到 GridView.js 在里边搜索一下 renderer, 就会看到调用 render 的地方, 这些参数都是在这里传进去的。

好, 看看效果吧。



编号	性别	名称	描述
1	红男	name1	查看详细信息
2	绿女	name2	查看详细信息
3	红男	name3	查看详细信息
4	绿女	name4	
5	红男	name5	

**[JavaScript 应用程序]**


 这个单元格的值是：descn2  
 这个单元格的配置是：{cellId:x-grid-cell-1-3,id:3,css:}  
 这个单元格对应的行的record是：[object Object]，一行的数据  
 这是第1行  
 这是第3列  
 这个表格对应的Ext.data.Store在这里：[object Object]，随

确定

剩下的，就是发挥各位聪明才智的时候了，舞台已经搭好，看你如何表演了。

html 例子，1.x 版本在 lingo-sample/1.1.1 目录下的 02-02.html，2.0 的版本在 lingo-sample/2.0 目录下的 02-02.html。

## 2.6. 更进一步，自动行号和多选checkbox

实际上行号和多选 checkbox 都是 renderer 的延伸，当然多选 checkbox 更酷一点儿，两者经常一起使用，所以让我们放在一起讨论好了。

### 2.6.1. 自动行号

只需要在 cm 中加上一行，这一行不会与 ds 中的任何数据对应，这也告诉我们可以凭空制作列，哈哈。

在之前的例子上改啦。

```
var cm = new Ext.grid.ColumnModel([
    {header:'NO.',renderer:function(value, cellmeta, record,
    rowIndex){
        return rowIndex + 1;
    }},
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

如吾等所愿，不指定 dataIndex，而是直接根据 renderer 返回 rowIndex + 1，因为它是从 0 开始的，所以加个一。截图如下。

NO.	编号	性别	名称	描述
1	1	male	name1	descn1
2	2	female	name2	descn2
3	3	male	name3	descn3
4	4	female	name4	descn4
5	5	male	name5	descn5

1.x 的例子在 [lingo-sample/1.1.1/02-03.html](http://lingo-sample/1.1.1/02-03.html)

很遗憾的是，2.0 里有自己的默认实现了，咱们不能展现自己的手工技艺了，还是乖乖使用 jack 提供的东东吧。

于是，在 2.0 里 cm 就变成了这幅模样。

```
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    {header:' 编号', dataIndex:' id'},
    {header:' 性别', dataIndex:' sex'},
    {header:' 名称', dataIndex:' name'},
    {header:' 描述', dataIndex:' descn'}
]);
```

你绝对会同意我的意见，Jack's work is amazing. 实在是太神奇了。看看截图就知道。

	编号	性别	名称	描述
1	1	male	name1	descn1
2	2	female	name2	descn2
3	3	male	name3	descn3
4	4	female	name4	descn4
5	5	male	name5	descn5

2.0 的例子在 [lingo-sample/2.0/02-03.html](http://lingo-sample/2.0/02-03.html)

2.6.2. 全选checkbox的时间了，请允许我让 2.0 先上场。

因为 2.0 里有 checkboxSelectionModel，这样完全可以证实用别人的轮子，比自己造轮子要方便。而且咱们造的轮子完全没有 jack 圆。不信的话，看下面 1.x 里的实现。

我们一直在修改 cm，这次我们也要对它动刀了，不过 SelectionModel 既 sm 也要处理一下，这是为了改变单选和多选行的方式，以前这些可以靠 shift 或 ctrl 实现，而现在这些都要与 checkbox 关联上了。

啦啦啦，先看图片，后看代码。

	<input checked="" type="checkbox"/>	编号	性别	名称	描述
1	<input checked="" type="checkbox"/>	1	male	name1	descn1
2	<input checked="" type="checkbox"/>	2	female	name2	descn2
3	<input checked="" type="checkbox"/>	3	male	name3	descn3
4	<input checked="" type="checkbox"/>	4	female	name4	descn4
5	<input checked="" type="checkbox"/>	5	male	name5	descn5

先看看具体多了什么

```
var sm = new Ext.grid.CheckboxSelectionModel();
```

神奇的是这个 sm 身兼两职，使用的时候既要放到 cm 里，也要放到 grid 中。代码如下。

```
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    sm,
    {header:' 编号', dataIndex:' id' },
    {header:' 性别', dataIndex:' sex' },
    {header:' 名称', dataIndex:' name' },
    {header:' 描述', dataIndex:' descn' }
]);

var grid = new Ext.grid.GridPanel({
    el: 'grid',
    ds: ds,
    cm: cm,
    sm: sm
});
```

然后你就可以得到效果啦，代码在 [lingo-sample/2.0/02-04.html](http://lingo-sample/2.0/02-04.html)。

### 2.6.3. 1.x时代的全选checkbox。

理论上只需要给 cm 再加一列，像自动编号那样，不对应数据，只显示 checkbox 就可以了。难点就是 checkbox 与某一行被选择的时候要对应上，用 checkbox 选择上，sm 里也要让这一行被选中，反之亦然。嗯，估计会比较复杂呢。

先放上显示 checkbox 的代码和截图：

```
var cm = new Ext.grid.ColumnModel([
    {header:'NO.',renderer:function(value, cellmeta, record,
    rowIndex) {
        return rowIndex + 1;
    }},
    {header:'<input type="checkbox"
onclick="selectAll(this)">',renderer:function(value, cellmeta, record,
    rowIndex) {
        return '<input type="checkbox" name="cb">';
    }},
    {header:' 编号',dataIndex:'id'},
    {header:' 性别',dataIndex:'sex'},
    {header:' 名称',dataIndex:'name'},
    {header:' 描述',dataIndex:'descn'}
]);
```

NO.	<input checked="" type="checkbox"/>	编号	性别	名称	描述
1	<input checked="" type="checkbox"/>	1	male	name1	descn1
2	<input checked="" type="checkbox"/>	2	female	name2	descn2
3	<input checked="" type="checkbox"/>	3	male	name3	descn3
4	<input checked="" type="checkbox"/>	4	female	name4	descn4
5	<input type="checkbox"/>	5	male	name5	descn5

与sm对接的方面比较麻烦，好在extjs.com上已经有扩展了，或者你可以看看我们弄下来的。[看看 1.x多选树的截图。](#)

## 2.7. 分页了吗？分页了吗？如果还没分就看这里吧。

如果你有一千条信息，一次都输出到 grid 里，然后让客户用下拉条一点儿一点儿去找吧。我们这里可是要做一个分页效果了，不用滚动屏幕，或者滚动一下就可以看到本页显示的数据，如果想看其他的只需要翻页就可以了。同志们，加强客户体验呀。

实际上，grid 控件挺耗性能的，据土豆讲一个页面上放 3 个 grid 就可以感觉到响应变慢，以前看过介绍，grid 里显示数据过多，听说是上千条，也会明显变慢。

所以说分页是必不可少滴，而且 jack 提供了方便集成分页工具条的方式，不用一下实在是太浪费了。

两步走，让 grid 集成分页。

### 2.7.1. 表面工作，先把分页工具条弄出来。

编号	名称	描述	
1	name1	descn1	
2	name2	descn2	
3	name3	descn3	
4	name4	descn4	
5	name5	descn5	
Page 1 of 1 显示第 1 条到 5 条记录，一共 5 条			

从图片可以清晰的看到，在 grid 下边多出来一行东东，包括了前一页，后一页，第一页，最后一页，刷新，以及提示信息。而我们不过写了如下几行代码而已，神奇呀。

```
var gridFoot = grid.getView().getFooterPanel(true);

var paging = new Ext.PagingToolbar(gridFoot, ds, {
    pageSize: 10,
    displayInfo: true,
    displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
    emptyMsg: '没有记录'
});
```

首先使用 `grid.getView().getFootPanel(true)`，获得 grid 下边那一条，嘿嘿，人家 grid 就设计的这么好，脚底下专门留了地方让你放东西。我们老实不客气，把 `Ext.PagingToolbar` 放到上边，就可以显示分页工具条了。

这分页工具条可不是个摆设，你按了前一页，后一页，整个 grid 都要有反应才对，要不咱们费劲弄这个东西过来干嘛呀？所以，我们在构造 `PagingToolbar` 的时候，不但告诉它，在 `gridFoot` 上显示，还告诉它，进行分页跳转的时候，要对 `ds` 也进行操作。这个 `ds` 就是 grid 用来获取和显示数据的，它一变整个 grid 都发生变化，嘿嘿~这就算共享数据模型了。厉害呀。

知道了分页的玄机，让我们揉揉眼睛，好好看看里边的参数。

- pageSize，是每页显示几条数据。
- displayInfo，跟下面的配置有关，如果是 false 就不会显示提示信息。
- displayMsg，只有在 displayInfo:true 的时候才有效，用来显示有数据的时候的提示信息，中国人应该看得懂汉语，到时候 {0}, {1}, {2} 会自动变成对应的数据，咱们只需要想办法把话说通就行了。
- emptyMsg，要是没数据就显示这个，jack 实在太贴心了，连这些小处都考虑得如此精细。

最好要注意的是，这段代码必须放在 grid.render() 之后，估计是因为动态生成 grid 的 dom 后，才能获得对应的 footPanel。

现在给出例子，1.x 的例子在 [lingo-sample/1.1.1/02-05.html](http://lingo-sample/1.1.1/02-05.html)。

### 2.7.2. 2.0 赐予我们更大的灵活性

其实，在下，一直，对于：必须先渲染 grid 才能获得 footPanel 这事非常愤恨，你想啊，本来分页也应该属于初始化的一部分，现在却要先初始化 grid，配置完毕，渲染，回过头来再从 grid 里把 footPanel 拿出来，再咕咚分页的配置。真，真，真郁闷呀。

所以 2.0 里的方式，简直大快民心。

```
var grid = new Ext.grid.GridPanel({
    el: 'grid',
    ds: ds,
    cm: cm,
    bbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: ds,
        displayInfo: true,
        displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
        emptyMsg: "没有记录"
    })
});
ds.load();
```

嘿嘿，加一个 bbar 的参数就可以了，bbar 就是 bottom bar 啦，底端工具条。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

Page 1 of 1
显示第 1 条到 5 条记录，一共 5 条

```
    json += "]}";  
    response.getWriter().write(json);  
} catch(Exception ex) {  
}  
%>
```

下面我们来解读这段 jsp 代码：

- 在进行操作之前，我们先要获得 ext 传递过来的两个参数：start 和 limit，start 指的从第几个数据开始显示，limit 是说从 start 开始，一共要用多少个数据，当然返回的数据可能会小于这个值。
- 咱们在后台模拟对 100 条数据进行分页，在获得了 start 和 limit 之后再生成 json 格式的数据。

何谓 json？在理论讲解之前先看看实例，让我们能有个感性认识，至于以后能不能升华到理性认识，就看各位的悟性了。

模拟 ext 访问后台，并传递两个参数 start=0&limit=10，把获得的数据稍微整理一下，是这个样子。

```
{totalProperty:100,root:[  
  {id:0,name:'name0',descn:'descn0'},  
  {id:1,name:'name1',descn:'descn1'},  
  {id:2,name:'name2',descn:'descn2'},  
  {id:3,name:'name3',descn:'descn3'},  
  {id:4,name:'name4',descn:'descn4'},  
  {id:5,name:'name5',descn:'descn5'},  
  {id:6,name:'name6',descn:'descn6'},  
  {id:7,name:'name7',descn:'descn7'},  
  {id:8,name:'name8',descn:'descn8'},  
  {id:9,name:'name9',descn:'descn9'}  
]}
```

请记住这个数据格式，不管后台是什么，只要满足了这样的格式要求，ext 就可以接收处理，显示到 grid 中。

我这里就不好好介绍 json，现在只需要知道 json 里头除了 name（名称）就是 value（值），值有好几种格式，如果是数字就不用加引号，如果加了引号就是字符串，如果用 [] 包裹就是数组，如果出现 {} 就说明是嵌套的 json。诸如此类。



简单瞄了 json 一眼，开头就是 `totalProperty:100`，这告诉 ext：“俺这里有 100 个数据呢。”，然后就是 `root:[]`，root 对应着一个数组，数组里有 10 个对象，每个对象都有 id 呀，name 呀，descn 呀。这 10 个数据最后就应该显示到表格里。

- jsp 里用 for 循环生成 root 数组里的数据，这样我们翻页的时候可以看到数据的变化，要不每次都是一样的数据，你怎么知道翻页是不是起作用了呢？

最后我们把得到的 json 字符串输出到 response 里，ext 也就可以获得这些数据了。

结果经过了一番折腾，我们的 jsp 已经确定可以返回我们所需要的数据了。现在我们可以忘掉后台是用什么语言写的了，直接切入 ext 代码，看看它是怎么样才能跑去后台获得这些数据呢？

因为引入了 json 作为数据传输格式，这次我们要对 ext 代码进行一次大换血了，请做好思想准备。

- 换掉 proxy，别在内存里找了，让我们通过 http 获得我们想要的。
- `proxy: new Ext.data.HttpProxy({url:'grid.jsp'}),`

创建 `HttpProxy` 的同时，用 `url` 这个参数指定咱们去哪里取数据，我们这里设置成 `grid.jsp`，就是我们刚才讨论的 jsp 脚本啦。

- 已经不是数组了，现在要用 json 咯。
- `reader: new Ext.data.JsonReader({`
- `totalProperty: 'totalProperty',`
- `root: 'root'`
- `}, [`
- `{name: 'id'},`
- `{name: 'name'},`
- `{name: 'descn'}`
- `])`

看比 `ArrayReader` 多了什么？`totalProperty` 对应咱们 jsp 返回的 `totalProperty`，也就是数据的总数。`root` 对应咱们 jsp 返回的 `root`，就是一个包含返回数据的数组。

- 好了，最后在初始化的时候，告诉我们希望获得哪部分的数据就可以了。

- `ds.load({params:{start:0,limit:10}});`

就在 ds 读取的时候添加两个参数，start 和 limit，告诉后台，我们从第一个数可以取起，最多要 10 个。

在这里有一个小插曲，如果你按照我们以前的设置，grid 是无法正常显示的，因为 ds.load() 无法在 grid.render() 前准备好所有数组，所以它不知道自己应该实现多高。没法子，我们只好为它指定一个固定的高度了。像这样`<div id="grid" style="height:265px;"></div>`。

最后，我们就可以使用分页条上那些按钮试试分页了。呵呵~就这么简单。

1.x 的例子在 `lingo-sample/1.1.1/02-06.html`，jsp 文件在 `lingo-sample/1.1.1/grid.jsp`，下面是它的截图：

编号	名称	描述	
40	name40	descn40	
41	name41	descn41	
42	name42	descn42	
43	name43	descn43	
44	name44	descn44	
45	name45	descn45	
46	name46	descn46	
47	name47	descn47	
48	name48	descn48	
49	name49	descn49	

Page 5 of 10 | 显示第 41 条到 50 条记录，一共 100 条

有趣的是，1.x 中，不需要为 div 指定高度，它自己就知道如何显示，不晓得为什么 2.0 里不这样做呢。

2.0 的例子在 `lingo-sample/2.0/02-06.html`，jsp 文件在 `lingo-sample/2.0/grid.jsp`，下面是它的截图：

编号	名称	描述	
40	name40	descn40	
41	name41	descn41	
42	name42	descn42	
43	name43	descn43	
44	name44	descn44	
45	name45	descn45	
46	name46	descn46	
47	name47	descn47	
48	name48	descn48	
49	name49	descn49	
			Page 5 of 10  显示第 41 条到 50 条记录，一共 100 条

#### 2.7.4. 其实分页不一定要踩在脚下，也可以顶在头上。

我的意思是，grid 除了 FooterPanel 以外，还有 HeaderPanel，意思就是头顶上的面板。我们把分页条放在上面也不会有任何问题。1.x 中的代码仅仅是将 getFooterPanel 改成 getHeaderPanel，这样分页条就跑到上头去了。

```
var gridHead = grid.getView().getHeaderPanel(true);

var paging = new Ext.PagingToolbar(gridHead, ds, {
    pageSize: 10,
    displayInfo: true,
    displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
    emptyMsg: '没有记录'
});
```

			Page 1 of 1  显示第 1 条到 5 条记录，一共 5 条
编号	名称	描述	
1	name1	descn1	
2	name2	descn2	
3	name3	descn3	
4	name4	descn4	
5	name5	descn5	

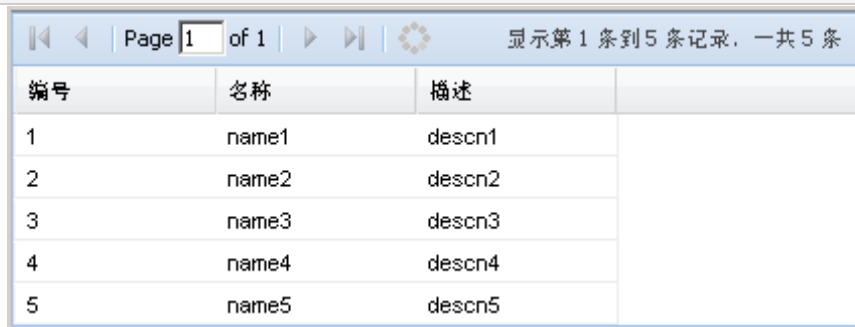
1.x 的例子可以在 [lingo-sample/1.1.1/02-07.html](http://lingo-sample/1.1.1/02-07.html) 找到。

2.0 就更简单了，只需要改一个字母，b -> t，呵呵~，让原来的 bbar(bottom bar) 变成 tbar(top bar) 就可以让我们的工具条登天了。

```

var grid = new Ext.grid.GridPanel({
    el: 'grid',
    ds: ds,
    cm: cm,
    tbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: ds,
        displayInfo: true,
        displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
        emptyMsg: "没有记录"
    })
});

```



编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

2.0 的例子可以在 [lingo-sample/2.0/02-07.html](http://lingo-sample/2.0/02-07.html) 找到。

呃，当然你可以让上下都加上分页条，反正它们最后都是共享一个 ds，功能不会有任何问题哈。吼吼。

## 2.8. 可编辑表格，改变大小，表格间拖拽，树与表格间拖拽。

未完待续

---

## 第3章 歌颂吧！只为了树也要学 ext。

### 3.1. 真的，我是为了树，才开始学 ext 的。

之前使用过 xtree 和 dojo 中的 tree，感觉都是怪怪的，界面简陋，功能也不好上手，待看到 ext 里的树形真是眼前一亮，在此之前，动态增添，修改删除节点，拖拽和右键菜单，我一直认为是不可能实现的任务，而在 ext 上却轻松实现了，而且界面和动画效果相当完美。真是让人爱不释手啊。

树形是非常典型的一种数据结构，多级菜单，部门组织结构，省市县三级这种金字塔结构都可以用树形表示，要表示一个老爸有一帮孩子的情况真是非树形莫属啊，做好了这部分，绝对是个亮点。

### 3.2. 传统是先做出一棵树来。

树形世界的万物之初是一个 TreePanel。

```
var tree = new Ext.tree.TreePanel('tree');
```

这里的参数 'tree'，表示渲染的 dom 的 id。html 写着个 <div id="tree"></div> 做呼应呢，最后这棵树就出现在这个 div 的位置上。

现在我们获得了树形面板，既然是树就必须有一个根，有了根才能在上边加枝子，放叶子，最后装饰的像一棵树似的。嗯，所以根是必要的，我们就研究研究这个根是怎么咕咚出来的。

```
var root = new Ext.tree.TreeNode({text:'偶是根'});
```

看到了吧，它自己都说它自己是根了，所以它就肯定是根没错。再看下面。

```
tree.setRootNode(root);  
tree.render();
```

首先，我们把这个根 root，放到 tree 里，用了 setRootNode() 方法，就是告诉 tree，这是一个根，你可得把它放好啊。

立刻对 tree 进行渲染，让它出现在 id="tree"的地方，这个 id 可是在上面指定的，如果有疑问，请翻回去继续研究，我们就不等你，继续了。

当当，我非常荣幸的向您宣布，咱们的第一棵树出来了。这是它的照片。



靠，这是树吗？

嗯，现在的确不太像树，因为它只有一个根。

靠，我们要的是树，你就搞出一个根来，有鸟用啊。

嗯，理论上等它自己发芽长成树，可能性是比较小，不如咱们偷偷插上几个枝子，说不定看起来就更像树了。

## 注意

虽然只有一个根，不过也算是棵树，1.x 的例子在 [lingo-sample/1.1.1/03-01.html](http://lingo-sample/1.1.1/03-01.html)。

## 3.3. 超越一个根

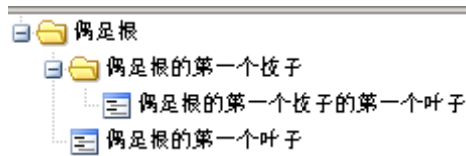
上回书说道，我们要偷偷插上几个杈子，让这个本来就是树的树，更像一棵树。

```
var root = new Ext.tree.TreeNode({text:'偶是根'});
var node1 = new Ext.tree.TreeNode({text:'偶是根的第一个枝子'});
var node2 = new Ext.tree.TreeNode({text:'偶是根的第一个枝子的第一个叶子'});
var node3 = new Ext.tree.TreeNode({text:'偶是根的第二个枝子的第一个叶子'});
node1.appendChild(node2);
root.appendChild(node1);
root.appendChild(node3);
```

我们从外边拿来三个 TreeNode，不管不顾就把 node2 插到 node1 上，node1, node2 一起插到 root 上，这下好了，咱们的树立刻就像是一棵树了。



咦？它怎么还是这么点儿东西？客官莫急，待小二儿给你整治一番，即看庐山真面目。



嗯，现在的确有点儿意思了，不过它开始的时候就那么缩在一团，看着很不爽，每次都要点这么几下才能看到底下的东西，咱们有没有办法让它每次渲染好就自己展开呢？

方法当然有咯，请上眼。

```
root.expand(true, true);
```

这一下就能解您燃眉之急，第一个参数是说，是否递归展开所有子节点，如果是 false，就只展开第一级子节点，子节点下面还是折叠着的，。第二个参数是说是否有动画效果，true 的话，你可以明显看出来那些节点是一点儿点儿展开的，否则就是刷拉一下子就出来了。

为了方便，咱们的例子里直接就可以展开了，省的再去点啊。1.x 的例子在 [lingo-sample/1.1.1/03-02.html](http://lingo-sample/1.1.1/03-02.html)。

### 3.4. 你不会认为 2.0 里跟 1.x 是一样的吧？

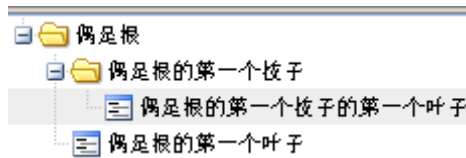
第一个区别就是 TreePanel 的定义，原来的 id 要放到 {} 中，对应的名字是 el。像这样改：

```
var tree = new Ext.tree.TreePanel({
    el: 'tree'
});
```

即使这样改完了，还是什么都看不见，我们错过了什么？用 findbug 看了一下 dom，height 竟然是 0，当然啥也看不见了，2.0 里的树为啥不会自动伸缩呢，只好咱们给它设置一个初始高度，在 html 里设置个 300px 的高度，就可以显示出来了。

```
<div id="tree" style="height:300px;"></div>
```

另一个也如法炮制。我们就可以看到 2.0 比 1.x 多了鼠标移到树节点上时的高亮显示。



好了，看了这些例子，应该对树型有些认识了，虽然这里只有 `TreeNode`，却能表示枝杈或者叶子，原理很简单，如果这个 `TreeNode` 下有其他节点，它就是一个枝杈，如果没有，它就是一个叶子，从它前头的图标就很容易看出来。嘿嘿，根其实就是一个没有上级节点的枝杈了。实际上，他们都是 `TreeNode` 而已，属性不同而已。

2.0 的例子在 `lingo-sample/2.0/` 目录下，分别是 `03-01.html` 和 `03-02.html`。

### 3.5. 这种装配树节点的形式，真是让人头大。

如此刀耕火种不但麻烦，而且容易犯错，有没有更简便一些的方法吗？答案是利用 `Ext.tree.TreeLoader` 和后台进行数据交换，我们在只提供数据，让 `TreeLoader` 帮咱们做数据转换和装配节点的操作。

啦啦啦，`json` 和 `ajax` 要登场了，不过你是否还记得我说过，一旦涉及到 `ajax` 就需要配合服务器了，`ajax` 是无法从本地文件系统直接取得数据的。

首先，让我们为 `TreePanel` 加上 `TreeLoader`

```
var tree = new Ext.tree.TreePanel('tree', {
    loader: new Ext.tree.TreeLoader({dataUrl: '03-03.txt'})
});
```

在此，`TreeLoader` 仅包含一个参数 `dataUrl: '03-03.txt'`，这个 `dataUrl` 表示，在渲染后要去哪里读取数据，为了方便模拟，我们写了一个 `txt` 文本文件提供数据，直接打开 `03-03.txt` 可以看到里边的内容。

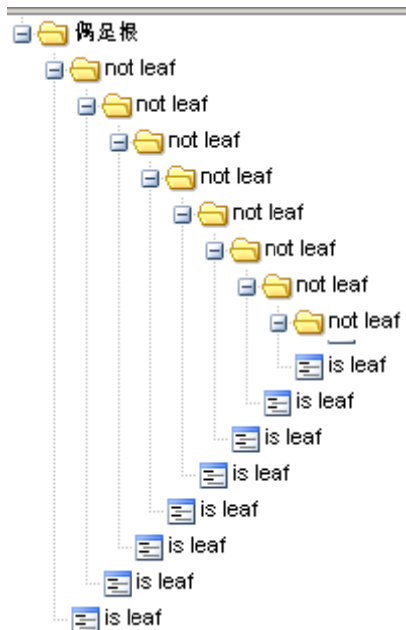
```
[
    {text:'not leaf'},
    {text:'is leaf',leaf:true}
]
```



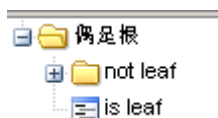
里边是一个包含了两个节点定义的数组，可能你会发觉那个多出来的属性 `leaf:true`，它的效果很神奇，这一点我们马上就可以看到。

如果你现在就去匆匆忙忙的刷新页面，想看一下咱们的成果，那一定会失望而归，页面上没有像你期待的那样，从 03-03.txt 读取数据，显示到页面上，你依然只能看到那个孤零零的根。这是因为 `TreeNode` 是不支持 ajax 的，我们需要把根节点换成 `AsyncTreeNode`，它可以实现咱们的愿望。

```
var root = new Ext.tree.AsyncTreeNode({text:'偶是根'});
```



估计谁第一次看到这场面都一定吓傻了。我们不是只定义了两个节点吗？怎么一下子跑出这么多东西来？先别着急，让我们先把 `root.expand(true, true)` 改成 `root.expand()`，避免节点无限展开下去，然后慢慢研究这个情况。



现在场面被控制住了，取消了递归展开，只展开根节点的第一层节点，我们得到的确实是与 03-03.txt 文件里相对应的两个节点，不过这两个节点有些不同，`not leaf` 节点的图标赫然是枝杈的图标，如果点击它前面的加号，便又成了上面的场景。Why?

原因就来自 `AsyncTreeNode`，这个东西会继承根节点 `TreeLoader` 中 `dataUrl`，你点展开的时候，会执行这个节点的 `expand()` 方法，ajax 会跑到 `dataUrl` 指定的地址去取数据，用 `firebug` 可以看到当前节点 `id` 会作为参数传递给 `dataUrl` 指

定的地址，这样我们的后台就可以通过这个节点的 id 计算出该返回什么数据，得到了数据 TreeLoader 去解析数据并装配成子节点，然后显示出来。

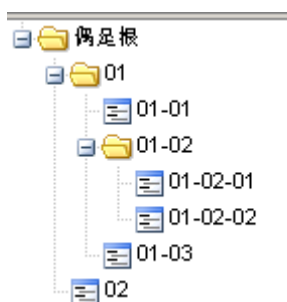
哈哈，现在就是关键部分了。因为咱们使用的 03-03.txt 提供的数据，不会判断当前节点的 id，所以每次返回的数据都是一样的，这就是树会无限循环下去的原因了。

那么为啥只有第一个节点会无限循环下去呢？第二个节点就没有那个小加号，呵呵~因为第二个节点不是 AsyncTreeNode，TreeLoader 在生成节点的时候会判断数据里的 leaf 属性，如果是 leaf:true，那么就会生成 TreeNode 而不是 AsyncTreeNode，TreeNode 可不会自动去用 ajax 取值，自然就不会无限循环展开了。

现实中，异步读取属性的节点是很爽的一件事情，因为你可能要保存成千上万条节点记录。一次性全部装载的话，无论读取和渲染的速度都会很慢。使用异步读取的方式，只有点击某一节点的时候，才去获得子节点属性，并进行渲染，极大的提高了用户体验。而且 ext 的树形本身有缓存机制，打开一次，再点击也不会去重复读取了，提升了响应速度。

html 例子，1.x 版本在 lingo-sample/1.1.1/03-03.html，2.0 版本在 lingo-sample/2.0/03-03.html。

为了巩固学习效果，咱们再写一个 json 获得数据的例子，这次的 json 稍微写复杂一点儿。



这次对应的 json 数据文件是 03-04.txt。

```
[
  {text:'01',children:[
    {text:'01-01',leaf:true},
    {text:'01-02',children:[
      {text:'01-02-01',leaf:true},
      {text:'01-02-02',leaf:true}
    ]},
    {text:'01-03',leaf:true}
  ]},
  {text:'02',leaf:true}
]
```

```
{text:'02',leaf:true}  
]
```

这也可以看作是在数据不多的情况下，一次加载所有数据的途径，只要确保所有叶子节点上都加上 `leaf:true` 的属性，就不会出现循环展开的问题了。

html 例子：1.x 在 `lingo-sample/1.1.1/03-04.html`，2.0 在 `lingo-sample/2.0/03-04.html`。

### 3.6. jsp的例子是一定要有的

### 3.7. 让你知道树都可以做些什么

#### 3.7.1. 检阅树形的事件

#### 3.7.2. 右键菜单并非单纯的事件

#### 3.7.3. 默认图标好单调，改一下撒

#### 3.7.4. 小小提示

### 3.8. 灰壳显灵！让我直接修改树节点的名称吧！

### 3.9. 我拖，我拖，我拖拖拖。

### 3.10. 更深一步，整合起来的是完整树形操作。

未完待续

---

## 第 4 章 祝福吧！把表单和输入控件都改成ext的样式。

### 4.1. 不用ext的form啊，不怕错过有趣的东西吗？

初看那些输入控件，其实就是修改了 css 样式表而已。你打开 firebug 看看 dom，确实也是如此，从这点看来，似乎没有刻意去使用 ext 的必要，诚然，如果单单要一个输入框，不管添入什么数据，就点击发送到后台，的确是不需要 ext 呢。

你不想用一些默认的数据校验吗？你不想在数据校验失败的时候，有一些突出的提示效果吗？你不想要超炫的下拉列表 combobox 吗？你不想要一些你做梦才能朦胧看到的选择控件吗？唉，要是你也像我一样禁不起诱惑，劝你还是随着欲望的节拍，试一下 ext 的 form 和输入控件。

### 4.2. 慢慢来，先建一个form再说

```
var form = new Ext.form.Form({
    labelAlign: 'right',
    labelWidth: 50
});
form.add(new Ext.form.TextField({
    fieldLabel: '文本框'
}));
form.addButton("按钮");
form.render("form");
```



简单来说，就是构造了一个 form，然后在里边放一个 TextField，再放一个按钮，最后执行渲染命令，在 id="form" 的地方画出 form 和里边包含的所有输入框和按钮来。刷拉一下就都出来了。

不过即使这样，圆角边框可不是 form 自带的，稍稍做一下处理，参见 html 里的写法。

```
<div style="width:220px;margin-left:0px;">
```

```

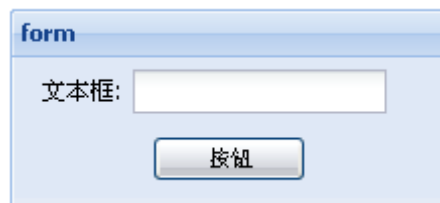
<div class="x-box-tl"><div class="x-box-tr"><div
class="x-box-tc"></div></div></div>
  <div class="x-box-ml"><div class="x-box-mr"><div class="x-box-mc">
    <h3 style="margin-bottom:5px;">form</h3>
    <div id="form"></div>
  </div></div></div>
  <div class="x-box-bl"><div class="x-box-br"><div
class="x-box-bc"></div></div></div>
</div>
<div class="x-form-clear"></div>

```

开头结尾那些 div 就是建立圆角的, 有了这些我们都可以 anywhere 使用这种圆角形式了, 不限于 form 哟。

html 例子, 1.x 在 [lingo-sample/1.1.1/04-01.html](#)

2.0 里的 FormPanel 跟 1.x 里已经基本完全不一样了, 咱们先看个简单例子:



代码如下:

```

var form = new Ext.form.FormPanel({
    defaultType: 'textfield',
    labelAlign: 'right',
    title: 'form',
    labelWidth: 50,
    frame: true,
    width: 220,

    items: [{
        fieldLabel: '文本框'
    }],
    buttons: [{
        text: '按钮'
    }]
});
form.render("form");

```

html 里不需要那么多东西了,只需要定义一个 `div id="form"` 就可以实现这一切。明显可以感觉到初始配置更紧凑,利用 `items` 和 `buttons` 指定包含的控件和按钮。

现在先感觉一下,我们后面再仔细研究,例子见 `lingo-sample/2.0/04-01.html`。

### 4.3. 胡乱扫一下输入控件

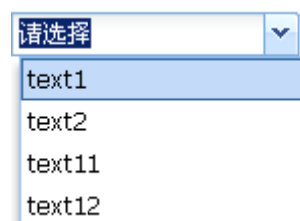
兄弟们应该都有 html 开发的经验了,像什么 `input` 用的不在少数了,所以咱们在这里也不必浪费唾沫,大概扫两眼也知道 `ext` 的输入控件是做什么的。

- 像 `TextField`, `TextArea`, `NumberField` 这类当然是可以随便输入的了。
- `ComboBox`, `DateField` 继承自 `TriggerField`。他们长相差不多,都是一个输入框右边带一个图片,点击以后就跳出一大堆东西来让你选择,输入框里头显示的是你选中的东西。
- `Checkbox` 和 `Radio`, `ext` 没有过多封装,基本上还是原来的方式。
- `Button`, 这个东东其实就是一个好看的 `div`, 跟 `comboBox` 一样,不是对原有组件的美化,而是重新做的轮子。你可以选择用以前那种难看的 `type="button"`, 还是用咱们漂亮的 `div`, 看你的爱好了。`type="submit"` 和 `type="reset"` 也一样没有对应的组件,都使用 `Button` 好了。
- 文件上传框, `type="file"`, 因为浏览器的安全策略,想上传文件,必须使用 `type="file"`, 而且我们不能使用 `js` 修改上传框的值,所以非常郁闷,目前的方式是把它隐藏起来,然后在点击咱们漂亮的 `Button` 时,触发上传框的点击事件,从而达到上传的目的。在这方面 `extjs.com` 论坛上有不少实现上传的扩展控件,咱们可以参考一下。

### 4.4. 起点高撒, 从 `comboBox` 往上蹦。

我觉得像 `TextField` 啊, `TextArea` 啊, 都是在原来的东西上随便加了几笔 `css` 弄出来的, 大家都会用, 所以没什么大搞头, 最后综合起来一说就 `ok` 了。而这个 `comboBox` 跟原有的 `select` 一点儿关系都没有, 完全是用 `div` 重写画的。所以, 嘿嘿~

耳听为虚, 眼见为实, 先看看所谓的 `comboBox` 究竟是个什么模样。



漂亮不？漂亮不？怎么看都比原生 select 强哟。啦啦啦，咱们看看代码撒。不过呢，comboBox 支持两种构造方式，一一看来。

4.4.1. 凭空变出个comboBox来。

>

4.4.2. 把select变成comboBox。

4.4.3. 破例研究下comboBox的内在本质哟

4.4.4. 嘿嘿~本地的做完了，试试远程滴。

4.4.5. 给咱们的comboBox安上零配件

4.4.6. 每次你选择什么，我都知道

4.4.7. 露一小手，组合上面所知，省市县三级级联。哈哈~

这是一个相当典型的案例，以前经常用 select 做，现在用 comboBox 实现的效果又会如何呢？

**4.4.7.1.** 先做一个模拟的，所有数据都在本地

**4.4.7.2.** 再做一个有后台的，需要放在服务器上咯

4.5. 还要做，字段验证呀，表单提交啊，表单布局咯，文件上传哟

未完待续

---

## 第 5 章 雀跃吧！超脱了一切的弹出窗口。

### 5.1. 呵呵~跳出来和缩回去总给人惊艳的感觉。

浏览器原声的 `alert()`，`confirm()`，`prompt()` 显得如此寒酸，而且还不能灵活配置，比如啥时候想加个按钮，删个按钮，或者改改按下按钮触发的事件了，都是难上加难的事情。

既然如此，为何不同 `ext` 提供的对话框呢？那么漂亮，那么好配置，可以拖啊，可以随便放什么东西，在里边用啥控件都可以，甚至放几个 `tab` 乱切换呀，连最小化窗口的功能都提供了。哈哈，神奇啊，完全可以让 `alert` 退役了。

### 5.2. 先看看最基本的三个例子

嘿嘿，为了加深认识，还是先去看看 `examples` 下的例子吧。1.x 在 `dialog` 目录下。2.0 在 `message-box` 目录下。

#### 5.2.1. `Ext.MessageBox.alert()`

```
Ext.MessageBox.alert('标题', '内容', function(btn) {  
    alert('你刚刚点击了 ' + btn);  
});
```

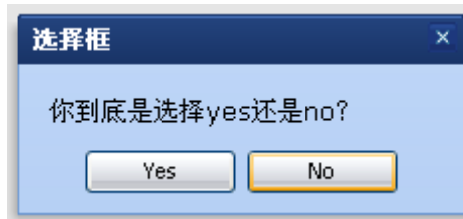


现在可以通过第一个参数修改窗口的标题，第二个参数决定窗口的内容，第三个参数是你关闭按钮之后（无论是点 `ok` 按钮还是右上角那个负责关闭的小叉叉），就会执行的函数，嘿嘿，传说中的回调函数。

#### 5.2.2. `Ext.MessageBox.confirm()`



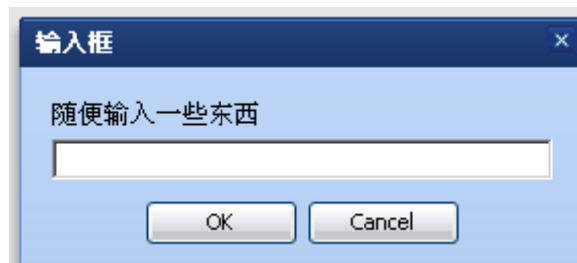
```
Ext.MessageBox.confirm('选择框', '你到底是选择 yes 还是 no?',  
function(btn) {  
    alert('你刚刚点击了 ' + btn);  
});
```



选择 yes 或者是 no，然后回调函数里可以知道你到底是选择了哪个东东。

### 5.2.3. Ext.MessageBox.prompt()

```
Ext.MessageBox.prompt('输入框', '随便输入一些东西', function(btn,  
text) {  
    alert('你刚刚点击了 ' + btn + ', 刚刚输入了 ' + text);  
});
```



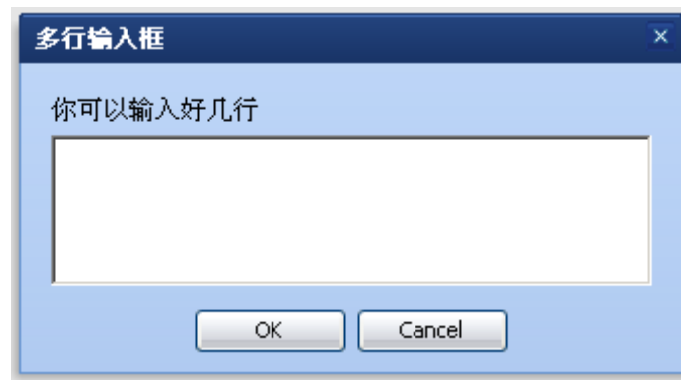
随便输入几个字，然后点按钮，它会告诉你输入了些什么东西

## 5.3. 如果你想的话，可以控制得更多

### 5.3.1. 可以输入多行的输入框

```
Ext.MessageBox.show({  
    title: '多行输入框',  
    msg: '你可以输入好几行',  
    width:300,  
    buttons: Ext.MessageBox.OKCANCEL,
```

```
multiline: true,  
fn: function(btn, text) {  
    alert('你刚刚点击了 ' + btn + ', 刚刚输入了 ' + text);  
}  
});
```

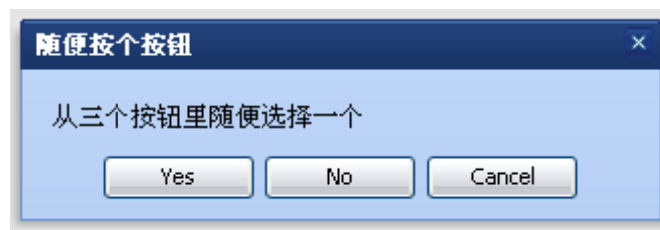


其实只需要 show，我们就可以构造各种各样的窗口了，title 代表标题，msg 代表输出的内容，buttons 是显示按钮，multiline 告诉我们可以输入好几行，最后用 fn 这个回调函数接受我们想要得到的结果。

### 5.3.2. 再看一个例子呗

可能让我们对 show 这个方法的理解更深

```
Ext.MessageBox.show({  
    title: '随便按个按钮',  
    msg: '从三个按钮里随便选择一个',  
    buttons: Ext.MessageBox.YESNOCANCEL,  
    fn: function(btn) {  
        alert('你刚刚点击了 ' + btn);  
    }  
});
```

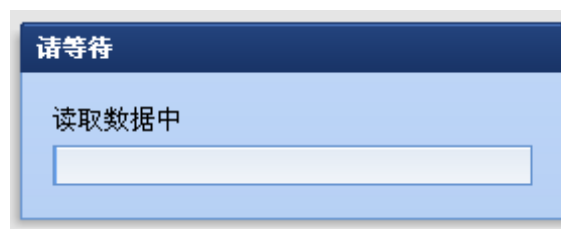


我相信 buttons 这个参数是一个数组，里边的这个参数绝对了应该显示哪些按钮，Ext.MessageBox 给我们提供了一些预先定义好的组合，比如 YESNOCANCEL, OKCANCEL，可以直接使用。

### 5.3.3. 下一个例子是进度条

实际上只需要将 progress 这个属性设置为 true，对话框里就会显示个条条。

```
Ext.MessageBox.show({
    title: '请等待',
    msg: '读取数据中',
    width:240,
    progress:true,
    closable:false
});
```



看到进度条了吧，不过它可不会自动滚啊滚的，你需要调用 Ext.MessageBox.updateProgress 让进度条发生变化。

另外多说一句，closable: false 会隐藏对话框右上角的小叉叉，这样咱们就不能随便关掉它了。

现在让咱们加上更新进度条的函数，使用 timeout 定时更新，这样咱们就可以看到效果了。呵呵~效果真不错，这样咱们以后就可以使用进度条了。

```
var f = function(v) {
    return function() {
        if(v == 11) {
            Ext.MessageBox.hide();
        } else {
            Ext.MessageBox.updateProgress(v/10, '正在读取第 ' + v + '
个，一共 10 个。');
        }
    };
};
for(var i = 1; i < 12; i++) {
```

```
        setTimeout(f(i), i*1000);  
    }  
}
```

#### 5.3.4. 动画效果，跳出来，缩回去

超炫效果，让对话框好像是从一个按钮跳出来的，关闭的时候还会自己缩回去。你可以看到它从小变大，又从大变小，最后不见了。实际上的配置非常简单，加一个 `animEl` 吧。让我们看看上边那个三个按钮的例子会变成什么样子。

```
Ext.MessageBox.show({  
    title: '随便按个按钮',  
    msg: '从三个按钮里随便选择一个',  
    buttons: Ext.MessageBox.YESNOCANCEL,  
    fn: function(btn) {  
        alert('你刚刚点击了 ' + btn);  
    },  
    animEl: 'dialog'  
});
```

`animEl` 的值是一个字符串，它对应着 `html` 里一个元素的 `id`，比如 `<div id="dialog"></div>`。指定好了这个，咱们的对话框才知道根据哪个元素播放展开和关闭的动画呀。

只需要这样，咱们就得到动画效果，嘿嘿，截不到动画效果的图，大家自己去看吧。

以上的例子在 `examples` 里都可以找到，不过咱们也提供了一份自己的例子，1.x 在 `lingo-sample/1.1.1/05-01.html`。2.0 在 `lingo-sample/2.0/05-01.html`。

好消息是，这部分的 `api` 没有什么改动。不过表现形式上有些差别，如果像我在例子里写的那样，一次生成 `N` 个 `MessageBox`，只能显示最后一个对话框。

不过在 1.x 里明显有一些数据同步的问题，1.x 里的 `updateProgress` 甚至可以影响其他对话框的 `msg`，以及可以关闭最后那个对话框。2.0 里至少是好的。

### 5.4. 让弹出窗口，显示我们想要的东东，比如表格

2.0 需要 `window` 来完成这个任务，1.x 版的 `BasicDialog` 稍后加上。

#### 5.4.1. 2.0 的弹出表格哦

稍微说一下 window 咋用呢？其实看起来跟 MessageBox 差不多啦，只是可以在里边随便放东西，现在先看个单纯的例子。

```
var win = new Ext.Window({
    el: 'window-win',
    layout: 'fit',
    width: 500,
    height: 300,
    closeAction: 'hide',

    items: [{}],

    buttons: [{
        text: '按钮'
    }]
});
win.show();
```

首先要讲明的是，这个 window 需要一个对应的 div 呀，就像 el 对应的 'window-win' 一样，这个 div 的 id 就应该等于 'window-win'，然后设置宽和高，这些都很明朗。

其次，需要设置的是布局类型，layout: 'fit' 说明布局会适应整个 window 的大小，估计改变大小的时候也要做响应的改变。

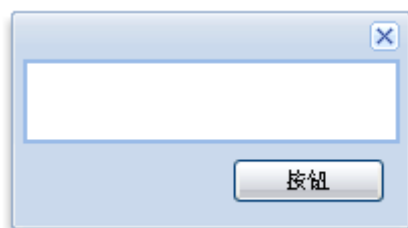
closeAction: 'hide' 是一个预设值，简单来说就是你用鼠标点了右上角的叉叉，会执行什么操作，这里就是隐藏啦。问为啥是隐藏？因为，因为预设啦，乖，背下来撒。

items 部分，嘿嘿~就是告诉咱们的 window 里要有什么内容啦。这里放表格，放树形，吼吼。

buttons 里设置在底端显示的按钮。我们就为了试一下，弄了一个按钮，但是按了没反应，嘿嘿。

最后调用一下 show()，让窗口显示出来。

看一下截图啦，更直观。



中间的空白就是 `items:[]` 的杰作，默认 `{}` 会成为一个 `Ext.Panel`，咱们什么都没定义，里边自然什么都没有。当然 `500*300` 不会只有这么大，但是为了让图片小一点儿，我把它拖下了，嘿嘿~自动支持的修改大小效果，帅吧？

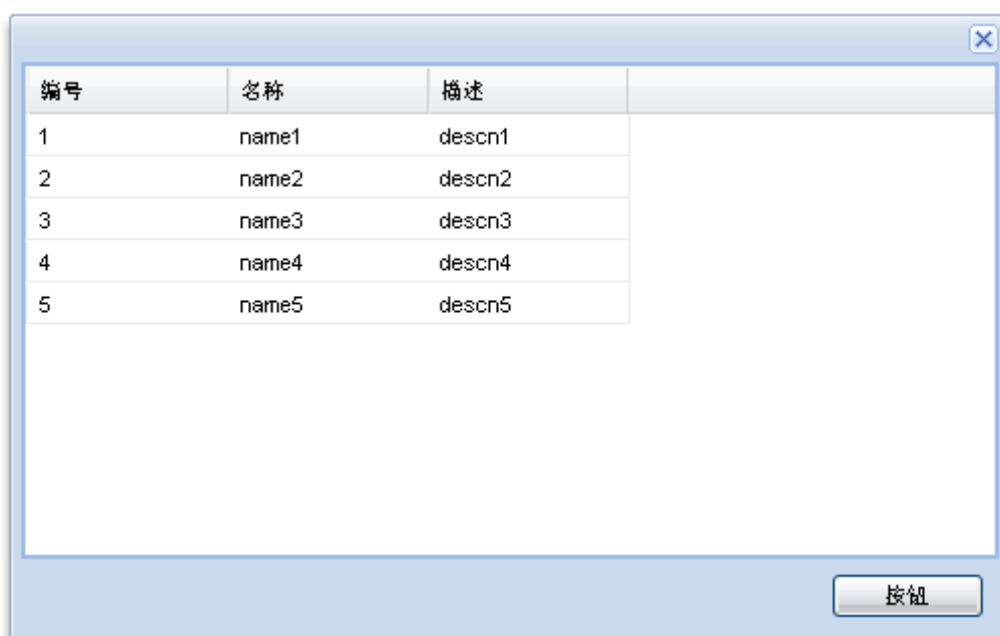
例子超简单，见 `lingo-sample/2.0/05-02.html`。

#### 5.4.2. 向 2.0 的window里加表格

唉，地方都划出来了，弄个表格放进去就好了呗。

首先弄一个 `grid`，超简单那种。我是直接把第二章的例子给 `copy` 了过来，嘿嘿，表格还是那个表格哟。

有了表格，直接扔到 `window` 里，然后 `ok`，哈哈~效果如下：



看到没？表格出来了，如果想加分页条什么的，只管动手，别怕伤到窗口。

现在回头让我们看看，需要注意些什么。

- 第一，grid 不用调用 render() 了，只要加入了 window，在 win.show() 的时候，会自动渲染里边的组件。
- 第二，html 里，要把 grid 对应的 div 写到 window 的 div 里面，嵌套关系。

```
• <div id="window-win">
•   <div id="grid"></div>
• </div>
```

- 第三，如果还不知道怎么把 grid 放进 window 里，我给你看下代码。

```
• var win = new Ext.Window({
•   el:'window-win',
•   layout:'fit',
•   width:500,
•   height:300,
•   closeAction:'hide',
•
•   items: [grid],
•
•   buttons: [{
•     text:'按钮'
•   }]
• });
```

看到 items:[grid] 了吗？就这么简单哟。

好了，具体例子在 [lingo-sample/2.0/05-03.html](http://lingo-sample/2.0/05-03.html)。敬请大家继续关注。

## 5.5. 更进一步撒。

希望上边的那些足够了，如果不够，咱们还有更厉害的，如果你看了 examples 里的例子，就知道里边还有更复杂的对话框，你甚至可以在里边用各种布局。

未完待续

---

## 第 6 章 奔腾吧！让不同的浏览器里显示一样的布局。

### 6.1. 有了它，我们就可以摆脱那些自称ui设计师的人了。

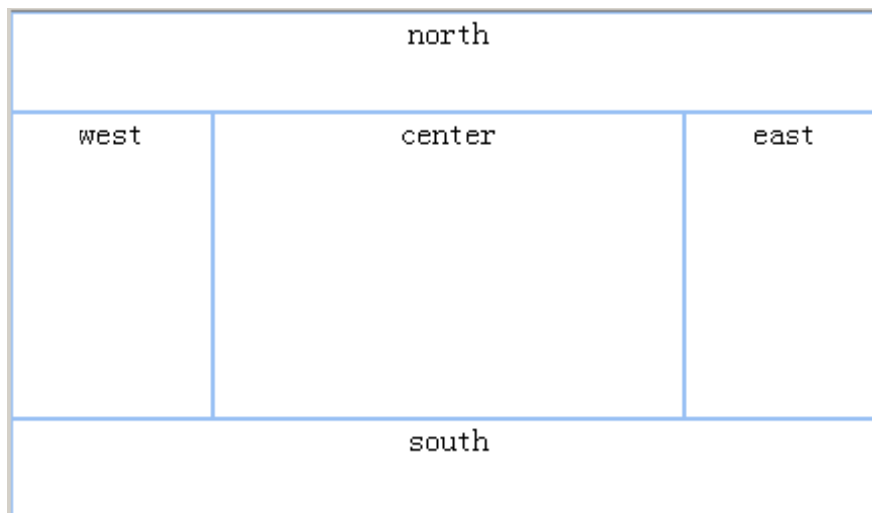
对布局很是不熟，至今为止，也是一直在抄土豆 demo 里的 BorderLayout, frank 的 deepcms ProjectTracker 里的 ViewPort 布局而已，不过有了布局，咱们不用再摆弄 frameset 了，只需要 div 就可以做成端端正正的布局，嗯，只这一点儿就吸引了多少眼球啊。

唉，咱们一起学学关于布局的用法吧。

### 6.2. 关于BorderLayout

理论上说，把整个窗口切成五块就够了吧？东南西北中，east, south, west, north, center 其中只有 center 中间这个部分是必须的，你完全可以把围绕在它四周的东西当作配角。

这样说还是太抽象，这个时候效果图绝对比其他途径来的直观。



实际上代码还是比较干净的。

```
var mainLayout = new Ext.BorderLayout(document.body, {  
    north: {  
        initialSize: 50  
    }, south: {
```



```

        initialSize: 50
    }, east: {
        initialSize: 100
    }, west: {
        initialSize: 100
    }, center: {
    }
});

mainLayout.beginUpdate();
mainLayout.add(' north', new Ext.ContentPanel(' north-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add(' south', new Ext.ContentPanel(' south-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add(' east', new Ext.ContentPanel(' east-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add(' west', new Ext.ContentPanel(' west-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add(' center', new Ext.ContentPanel(' center-div', {
    fitToFrame: true
}));
mainLayout.endUpdate();

```

html 需要五个 div 与其对应, div 与 ContentPanel 是一一对应的, 请看他们的 id。

```

<div id="north-div">north</div>
<div id="south-div">south</div>
<div id="east-div">east</div>
<div id="west-div">west</div>
<div id="center-div">center</div>

```

这个其实挺有意思的, 你必须先构造一个 BorderLayout, 指定需要渲染的部分, 这里是 document.body, 并指定 5 个部分的初始化大小, 然后调用 beginUpdate() 让整个布局先不要刷新, 当然我们最后会调用 endUpdate() 刷新布局, 这样用户就获得了更好的体验。

beginUpdate() 之后，我们立刻使用 add 方法，向 5 个部分分别加入 Ext.ContentPanel，这些面板的第一个参数是对应 dom 的 id，后边是附加的参数，比如 fitToFrame:true，它告诉面板在布局区域改变大小的时候调整自己的大小，然后是 closable:false，这样用户就不能点击关闭按钮，关闭这个面板。

好了，你也看到了，这五个部分明显已经分隔开了，使用的时候我们只需要在合适的地方放上合适的东西就行了。

例子在 [lingo-sample/1.1.1/06-01.html](http://lingo-sample/1.1.1/06-01.html)。

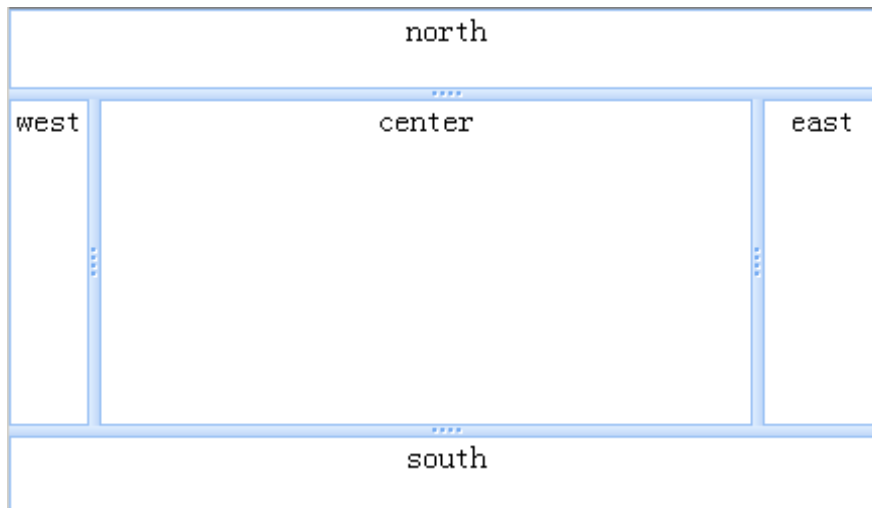
## 6.3. 嗯，不如再看看附加效果

其实，即使只能在不同浏览器，把一个窗口切成相同的部分，也是足够了，不过 ext 带给我们的不仅仅是如此，让我们再看看其他部分吧。

### 6.3.1. 先看看split

```
var mainLayout = new Ext.BorderLayout(document.body, {
    north: {
        initialSize: 50,
        split: true
    }, south: {
        initialSize: 50,
        split: true
    }, east: {
        initialSize: 100,
        split: true
    }, west: {
        initialSize: 100,
        split: true
    }, center: {
    }
});
```

让我们给所有区域都加上 split:true，看看会有什么效果？



请注意一点，这并不仅仅是那些边框变粗了，split 让我们可以自由拖动边框，让用户可以改变各个区域的大小。

当然，我们不会让用户为所欲为的，让我们加上一点点限制，这点儿限制绝对不会让用户感到难堪。

```
var mainLayout = new Ext.BorderLayout(document.body, {
    north: {
        initialSize: 50,
        minSize: 40,
        maxSize: 60,
        split: true
    }, south: {
        initialSize: 50,
        minSize: 40,
        maxSize: 60,
        split: true
    }, east: {
        initialSize: 100,
        minSize: 80,
        maxSize: 120,
        split: true
    }, west: {
        initialSize: 100,
        minSize: 80,
        maxSize: 120,
        split: true
    }, center: {
    }
});
```

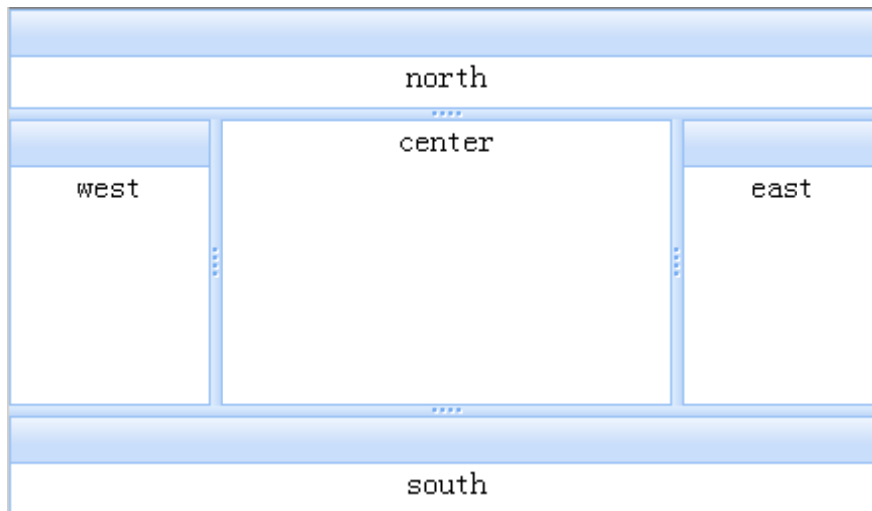
minSize 和 maxSize 让用户只能在我们决定的范围内修改区域的大小，既不会太大，也不会太小。用户的行为受限，也减少了他们抱怨的机会。嘿嘿，一切尽在掌握中。

例子见 [lingo-sample/1.1.1/06-02.html](http://lingo-sample/1.1.1/06-02.html)

### 6.3.2. 再试试titlebar

```
var mainLayout = new Ext.BorderLayout(document.body, {
    north: {
        initialSize: 50,
        minSize: 40,
        maxSize: 60,
        titlebar: true,
        split: true
    }, south: {
        initialSize: 50,
        minSize: 40,
        maxSize: 60,
        titlebar: true,
        split: true
    }, east: {
        initialSize: 100,
        minSize: 40,
        maxSize: 60,
        titlebar: true,
        split: true
    }, west: {
        initialSize: 100,
        minSize: 40,
        maxSize: 60,
        titlebar: true,
        split: true
    }, center: {
    }
});
```

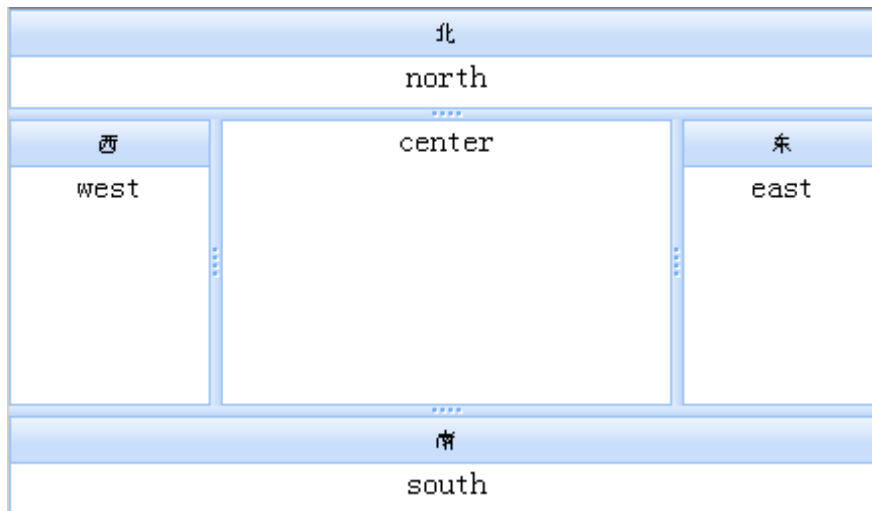
在上面的基础上，我们加入了 titlebar，然后看到的就是这幅情景。



标题栏是空的，要加标题我们另有地方，看看 ContentPanel 的部分。

```
mainLayout.beginUpdate();
mainLayout.add(' north', new Ext.ContentPanel(' north-div', {
    fitToFrame: true, closable: false, title: '北'
}));
mainLayout.add(' south', new Ext.ContentPanel(' south-div', {
    fitToFrame: true, closable: false, title: '南'
}));
mainLayout.add(' east', new Ext.ContentPanel(' east-div', {
    fitToFrame: true, closable: false, title: '东'
}));
mainLayout.add(' west', new Ext.ContentPanel(' west-div', {
    fitToFrame: true, closable: false, title: '西'
}));
mainLayout.add(' center', new Ext.ContentPanel(' center-div', {
    fitToFrame: true
}));
mainLayout.endUpdate();
```

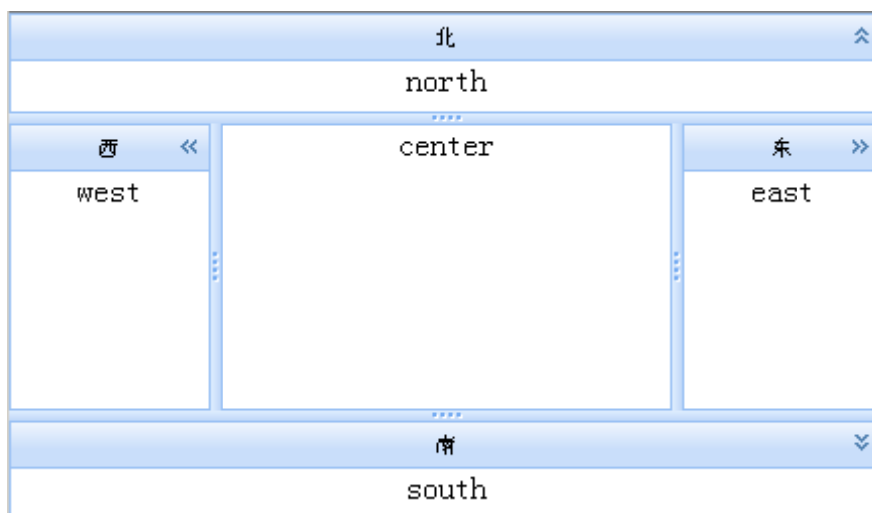
经过这些改变，整个布局就变成了这个样子。



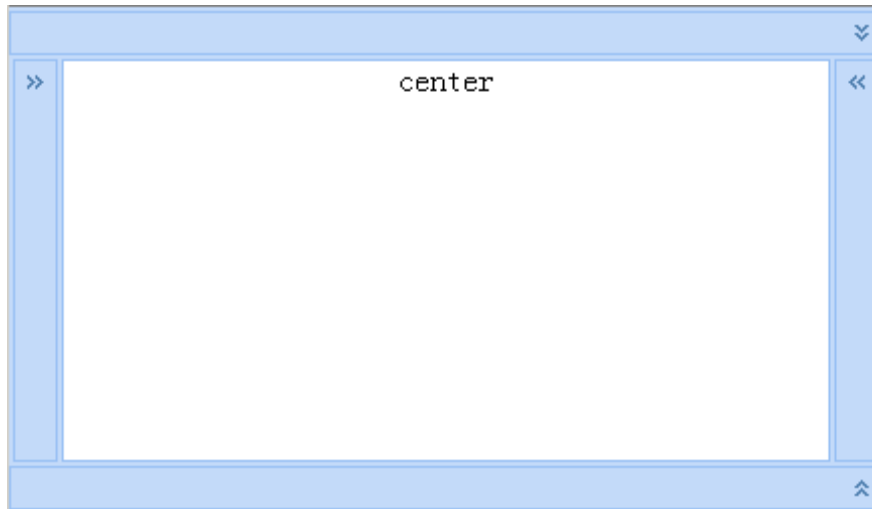
例子是 [lingo-sample/1.1.1/06-03.html](http://lingo-sample/1.1.1/06-03.html)。

### 6.3.3. 还不够，还不够，让四周的区域可以缩起来

很多软件都可以这样哦，看小面板不顺眼，就折叠起来，为中间的工作区留出更多空间哟。就像这样。



都折叠上以后就变成这样。



其实只要加一个属性就可以了，看看代码中的 `collapsible: true` 造就了现在的盛况。

```
var mainLayout = new Ext.BorderLayout(document.body, {
  north: {
    initialSize: 50,
    minSize: 40,
    maxSize: 60,
    titlebar: true,
    collapsible: true,
    split: true
  }, south: {
    initialSize: 50,
    minSize: 40,
    maxSize: 60,
    titlebar: true,
    collapsible: true,
    split: true
  }, east: {
    initialSize: 100,
    minSize: 40,
    maxSize: 60,
    titlebar: true,
    collapsible: true,
    split: true
  }, west: {
    initialSize: 100,
    minSize: 40,
    maxSize: 60,
    titlebar: true,
```

```
        collapsible: true,  
        split: true  
    }, center: {  
    }  
});
```

你还可以加上 `collapsedTitle` 属性，让北方和南方区域折叠之后显示，这个属性只在 `north` 和 `south` 部分有效，因为 `west` 和 `east` 是垂直的，似乎没有办法让文字旋转 90 度显示，所以我们需要其他方法。

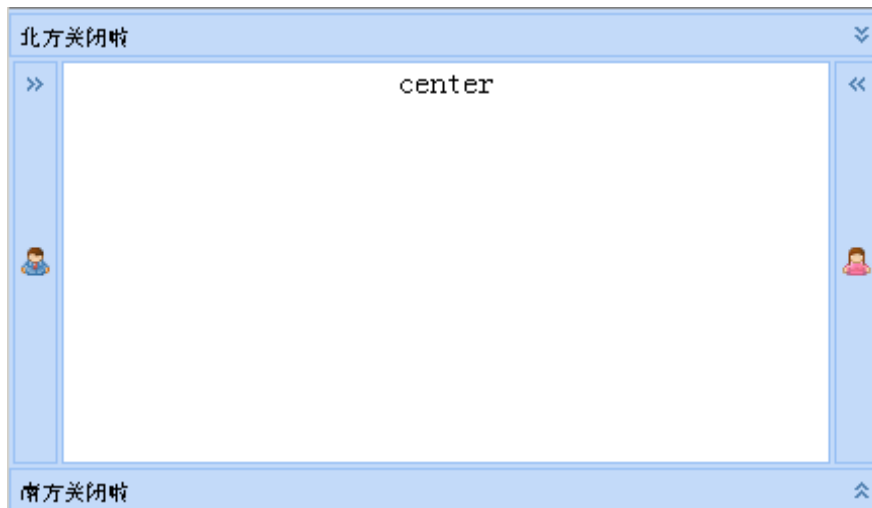
参考网上的方式，是用一个图片，窄窄高高的图片，然后把它作为对应 `css` 样式的背景图，这样在 `east` 和 `west` 折叠的时候就会显示它们了。让咱们试验一下好了。

我们需要设置的 `css` 有两个，`west` 对应左边，`east` 对应右边。

```
.x-layout-collapsed-west {  
    background-image: url(user_male.png);  
    background-repeat: no-repeat;  
    background-position: center;  
}  
  
.x-layout-collapsed-east {  
    background-image: url(user_female.png);  
    background-repeat: no-repeat;  
    background-position: center;  
}
```

最后就变成了这样。



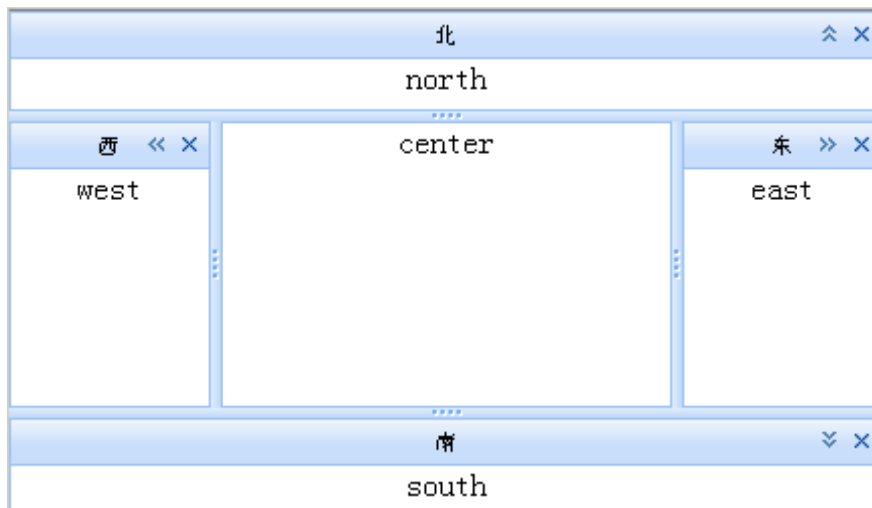


嘿嘿，有意思吧。代码都在 `lingo-sample/1.1.1/06-04.html` 里呢，你也试试吧。

#### 6.3.4. 给这些区域都加上个关闭按钮

```
mainLayout.beginUpdate();
mainLayout.add(' north', new Ext.ContentPanel(' north-div', {
    fitToFrame: true, closable: true, title: '北'
}));
mainLayout.add(' south', new Ext.ContentPanel(' south-div', {
    fitToFrame: true, closable: true, title: '南'
}));
mainLayout.add(' east', new Ext.ContentPanel(' east-div', {
    fitToFrame: true, closable: true, title: '东'
}));
mainLayout.add(' west', new Ext.ContentPanel(' west-div', {
    fitToFrame: true, closable: true, title: '西'
}));
mainLayout.add(' center', new Ext.ContentPanel(' center-div', {
    fitToFrame: true
}));
mainLayout.endUpdate();
```

这个部分跟 `ContentPanel` 有关，把参数 `closable` 改成 `true` 就会出现那个小叉叉，按一下这个区域就关上了。



可惜现在还不知道关闭以后再怎么打开，嘿嘿。

### 6.3.5. 用NestedLayoutPanel在五块中再进行分割，实现更复杂的布局

我不会用。

## 6.4. 2.0 的Viewport是完全不同的实现

简单来说，用了 ViewPort 摆脱先定义 BorderLayout，再 beginUpdate, endUpdate 的麻烦，我们就问了，为什么事情不能更简单明了呢，就让我们看看用 2.0 解决上头的五块是个什么样子？

首先 html 里的东东不变。

```
<div id="north-div">north</div>
<div id="south-div">south</div>
<div id="east-div">east</div>
<div id="west-div">west</div>
<div id="center-div">center</div>
```

剩下的就是代码了，还是那句话，我们想要在一个地方配置好所有东西，不想东奔西跑，说不定丢了什么也不知道，维护多个地方的配置简直是噩梦，2.0 啊，我们崇拜你。

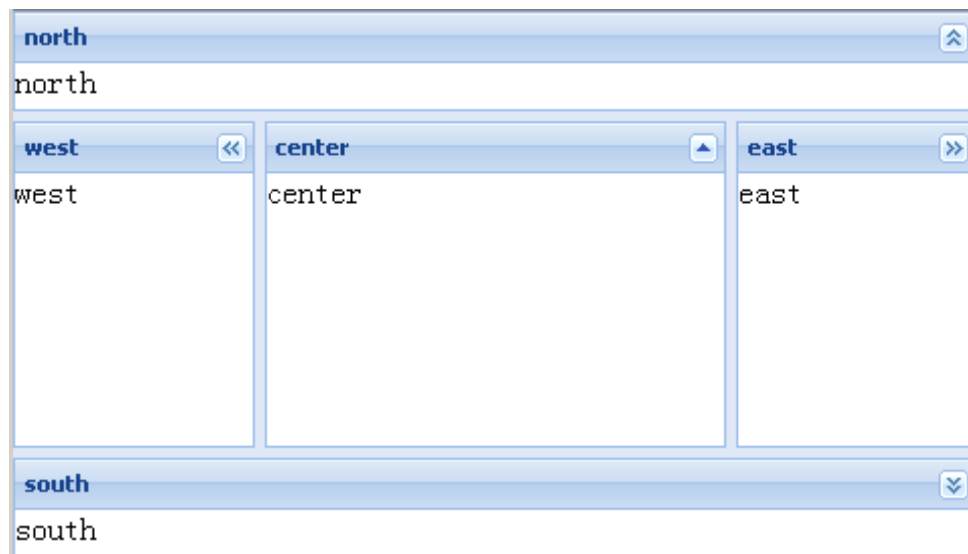
```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
```

```
    title: 'north',
    region: 'north',
    contentEl: 'north-div',
    split: true,
    border: true,
    collapsible: true,
    height: 50,
    minSize: 50,
    maxSize: 120
  }, {
    title: 'south',
    region: 'south',
    contentEl: 'south-div',
    split: true,
    border: true,
    collapsible: true,
    height: 50,
    minSize: 50,
    maxSize: 120
  }, {
    title: 'east',
    region: 'east',
    contentEl: 'east-div',
    split: true,
    border: true,
    collapsible: true,
    width: 120,
    minSize: 120,
    maxSize: 200
  }, {
    title: 'west',
    region: 'west',
    contentEl: 'west-div',
    split: true,
    border: true,
    collapsible: true,
    width: 120,
    minSize: 120,
    maxSize: 200
  }, {
    title: 'center',
    region: 'center',
    contentEl: 'center-div',
    split: true,
```

```
        border: true,  
        collapsible: true  
    }]  
});
```

如果非要挑刺的话，那就是没有 `closable` 的选项了，不过现实谁会去关闭一块面板啊？至少我不会滴。

现在所有配置都放在一起了，也不用先创建后布局两步走，方便呀。



6.5. 稍稍感叹一下 2.0 的简洁吧，让人吃惊的还在后头呢。

未完待续

---

## 第 7 章 低鸣吧！拖拽就像呼吸一样容易。

### 7.1. 如此拖拽，简直就像与生俱来的本能一样。

你可以拖拽 grid 里的行，让它们按你的方式去排列。

你可以拖拽 tree 里的节点，把节点从一个枝干拖向另一个枝干。

grid 和 tree 之间，也可以拖动。

layout 的 split 也是一种拖动，改变布局的大小。

resize 也算是拖动，改变大小。

### 7.2. 第一！乱拖。

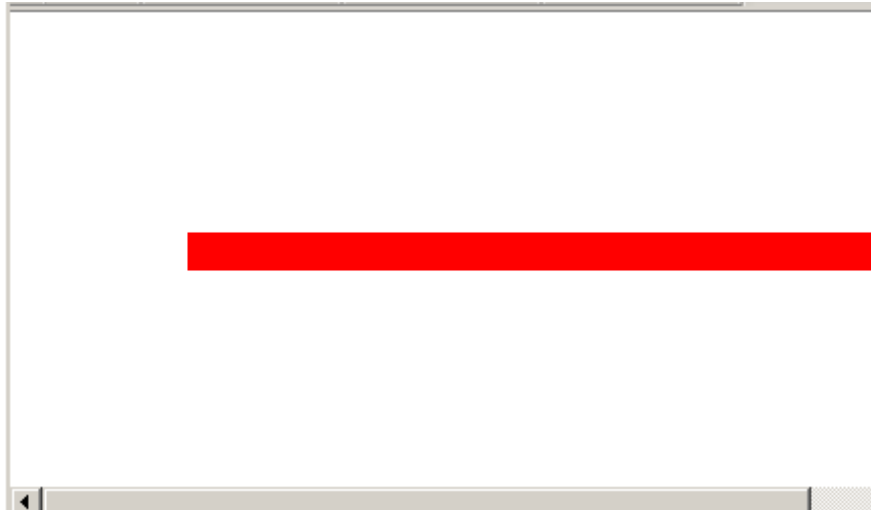
一直认为拖动很复杂，可实际上只需要一条语句就可以了。

```
new Ext.dd.DDProxy('block');
```

然后看看 html 里的部分

```
<div id="block" style="background: red;">&nbsp;</div>
```

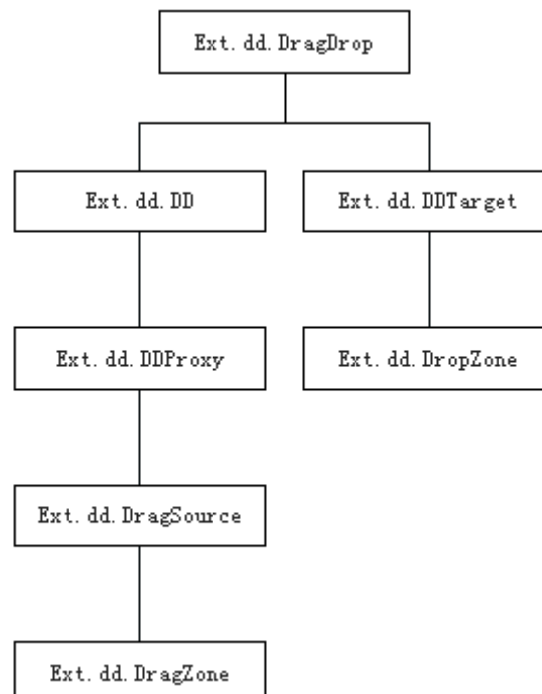
如果不做任何标记，我们根本什么都看不到，所以我给加上了红色背景颜色。现在你可以乱拖了。



截图里的红条条可以随便拖, 实际看例子吧。lingo-sample/1.1.1/07-01.html, 2.0 里用法一样, lingo-sample/2.0/07-01.html。

### 7.3. 第二！代理proxy和目标target

实际上我们刚才看到了DDProxy可以随便拖, 另外一个DDTarget是不能拖动的, 这东西是用来放DDProxy的一个区域。看一看继承体系图可能有助于我们的理解。



简单来说, 左边都是可以随你的鼠标拖动的, 拖动起来以后, 直接把他们扔到右边那些定义好的区域就好了。proxy 是可拖动对象, target 是拖动的目的地。

在知道了是与非之后，我们要验证一下自己的理念了。

proxy 是我们要拖来拖去的东西。

```
var proxy = new Ext.dd.DragSource(' proxy', {group:' dd'});
```

target 告诉用户，他应该把上边的 proxy 放到哪里

```
var target = new Ext.dd.DDTarget(' target', ' dd');
```

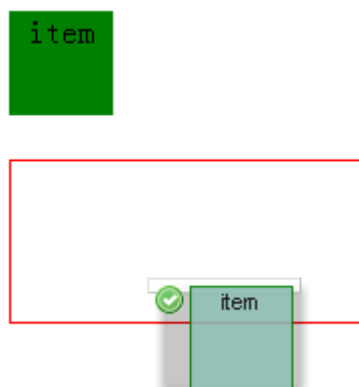
注意到两者之中相同的 dd 了吗？这是个分组标志，咱们通过这个限制用户不会像第一节那样，把 proxy 乱扔到任何地方。只有相同组名的目的地才能接收它，好比你能只能把超市货架上的商品放进篮子，而不能满地乱扔一样。

不过这也仅仅是拖拽而已，没有任何效果不是很不爽吗？让我们加入一些小技巧吧，可以让拖拽显得更神奇一些。

```
proxy.afterDragDrop = function(target, e, id) {  
    var destEl = Ext.get(id);  
    var srcEl = Ext.get(proxy.getEl());  
    srcEl.insertBefore(destEl);  
};
```

这个函数是说，在 dragdrop 之后，会执行这个函数，然后通过 id 获得 target，根据 proxy.getEl() 获得 proxy，然后把 proxy 添加到 target 的前头，看起来是放在里边了。

好，脚本都组织好了，打开页面看看效果吧。



当然，为了让画面花花绿绿的，咱们加了不少 css 样式，稍微给你们看一下吧，省得那些不愿意交钱的人说咱们的截图是用 photoshop 画的。

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=gbk">
    <title>dd</title>
    <link rel="stylesheet" type="text/css"
href="../../../resources/css/ext-all.css" />
    <script type="text/javascript"
src="../../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript"
src="../../../ext-all.js"></script>
    <style type="text/css">
HR {
  clear: both;
  visibility: hidden
}
.block {
  border: 1px red solid;
  margin: 10px;
  min-height: 80px;
}
.item {
  border: 1px green solid;
  background: green;
  float: left;
  margin: 10px;
  width: 50px;
  min-height: 50px;
  text-align: center;
}

    </style>
    <script type="text/javascript">
Ext.onReady(function() {

  var proxy = new Ext.dd.DragSource(' proxy', {group:' dd' });

  proxy.afterDragDrop = function(target, e, id) {
    var destEl = Ext.get(id);
    var srcEl = Ext.get(proxy.getEl());
    srcEl.insertBefore(destEl);
  };
};
```



```
var target = new Ext.dd.DDTarget('target', 'dd');
});
</script>
</head>
<body>
  <script type="text/javascript" src="../examples.js"></script>
  <div id="proxy" class="item">item</div>
  <hr />
  <div id="target" class="block">
    <hr />
  </div>
</body>
</html>
```

好了，其实你也可以在 `lingo-sample/1.1.1/07-02.html` 看到咱们的例子。很高兴的是，`lingo-sample/2.0/07-02.html` 跟它完全一样。

## 7.4. 再拖！再拖拖。

还有 `JsonView` 啦，`DataView` 什么的，都可以拖，但渲染的时候更见方便，这些等于是 `Template` 和 `Store` 的一种结合体制。

未完待续

---

## 第 8 章 哭泣吧！现在才开始讲基础问题。

### 8.1. Ext.get

ext 里用来获得 Element 的一个函数，用途还算比较广，可以通过不少途径获得咱们需要的 Element，而这个 Element 包括很多有趣的功能。

Element 跟 document.getElementById("myDiv") 得到的 dom 对象是不一样的，虽然你还可以使用老方式获得指定 id 的元素，但那样就失去了 ext 提供的各种常用操作，动画啦，定位啦，css 啦，事件啦，拖拽啦。其实也不用担心，即便使用了 Ext.get() 获得了 myDiv，还是可以直接访问 document.getElementById() 应该得到的部分，而且挺简单的，Ext.get().dom 就可以了。

下面让偶们来看看这些基本的功能会是咋样呢？

- 先获得一个 Element
- `var myDiv = Ext.get('myDiv');`

这里我们传入的是一个 id，你可以在 html 里看到<div id="myDiv"></div>，然后用 Ext.get('myDiv') 从 html 里取得这个 div，然后封装成 Element 对象，现在这个对象就已经放到缓存中了，以后再用的时候就更快撒。

- 最吸引眼球的是动画效果，所以我们先动两下。
- `myDiv.highlight();`

红色高亮，然后渐退。

```
myDiv.addClass('red');
```

添加自定义 CSS 类，css 里有 .red {background: red;} 的定义，这样 myDiv 的背景直接变成了红色。

```
myDiv.center();
```

myDiv 移动的窗口中间，包括垂直和竖直居中。

```
myDiv.setOpacity(.25);
```

使 myDiv 半透明

- 再看看怎么才好渐变动画
- myDiv.setWidth(100);

这样可以直接设置 myDiv 的宽度，是没有渐变动画的。

```
myDiv.setWidth(100, true);
```

这样就打开了动画开关，如此简单就可以看到 myDiv 在动咯。

咱们还可以控制动画的动作，如下

```
myDiv.setWidth(100, {  
    duration: 2,  
    callback: this.highlight,  
    scope: this  
});
```

duration 是间隔，数字越大移动越慢，callback 说是动画完成后执行，但我没饰演出来，scope 是 callback 执行的范围。

动画没法截图，还是看看 [lingo-sample/1.1.1/08-01.html](#)，  
[lingo-sample/2.0/08-01.html](#) 吧，四个按钮可以让 myDiv 在窗口里乱动，哈哈。

## 8.2. 要是我们想一下子获得一堆元素咋办？

现在像 css 那样的批量选择方式真的很流行，ext 里也没有落伍，一定会赶这个潮流。

- 选择所有<P>元素

现在我们要获得所有<P>元素，然后让他们都闪一下。

```
Ext.select("p").highlight();
```

- 按照 css 的 class 选择

首先我们有几个 div，都使用 class="red"，然后我们让他们都闪一下，嘿嘿嘿~因为 highlight() 调用比较简单嘛。

```
Ext.select("div.red").highlight();
```

这种方式在 prototype 和 jquery 里已经发扬光大，而且还光大得很呢，你只需要按照 css 的选择方式，就可以得到你需要的集合。这方面其实 jquery 颇为神奇，把 select 用的真是出神入化，可叹，它对 js 封装太狠，你用 jquery 的时候完全感觉不到自己是在用 javascript，这样接触原生方法的机会很少，等于把自己绑定到 jquery 上，最后权衡利弊，只好忍痛割爱了。

批量选择，见 [lingo-sample/1.1.1/08-02.html](http://lingo-sample/1.1.1/08-02.html) 和 [lingo-sample/2.0/08-02.html](http://lingo-sample/2.0/08-02.html)。

## 8.3. DomHelper和Template动态生成html

用 dom 生成 html 元素一直是头疼的事情，以前都是听 springside 的教导，使用 jsTemplate 和 Scriptaculous 的组合。现在到了 ext 里面，我们就来看看它自己的实现。

### 8.3.1. DomHelper用来生成小片段

使用 DomHelper 非常灵活，超简单就可以生成各种 html 片段，遇到复杂情况也要求助于它。

大概就是这么用

```
var list = Ext.DomHelper.append('parent', {tag: 'div', cls: 'red'});
```

它就是向 id=parent 这个元素里，添加一个 div 元素。

按照文档里讲的，第二个参数 {} 里，除了四个特殊属性以外都会复制给新生成元素的属性，这四个特殊属性是

- tag, 告诉我们要生成一个什么标签, div 啦, span 啦, 诸如此类。

千万别告诉我到现在你还不知道这些 html 标签, 中间告诉你多少次先去学学 html 和 css 啦? 你飞过来的不成?

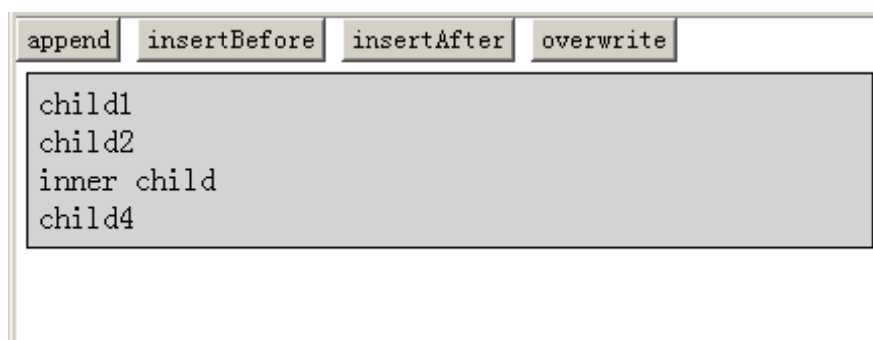
- cls, 指的是<div class="red"></div>这种标签中的 class 属性, 因为 class 是关键字, 正常情况下应该写成 className, 可 jack 说 className 太长了, 最后就变成 cls 了。-\_-。

他就喜欢玩这个, 把 datastore 写成 ds, DomHelper 写成 dh, Element 写成 el, ColumnModel 写成 cm, SelectionModel 是 sm。唉, 发明的专业名词缩写好多呀。

- children, 用来指定子节点, 它的值是一个数组, 里边包含了更多节点。
- html, 对应 innerHTML, 觉得用 children 描述太烦琐, 直接告诉节点里边的 html 内容也是一样。

DomHelper 除了 append 还有几个方法, 指定将新节点添加到什么位置。

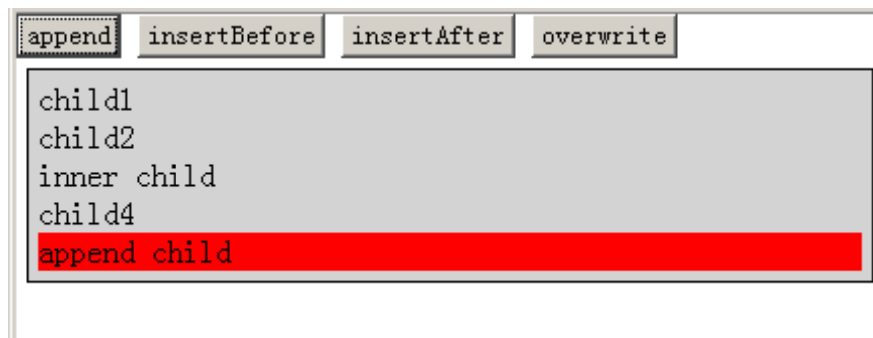
为了比对效果, 先放一个初始页面。



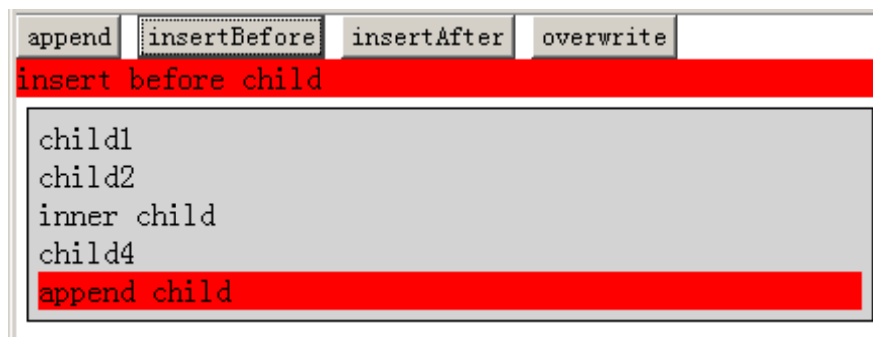
原始的 html 是这样的。一个 div 下有 4 个节点, 其中第三个子节点下还有自己的子节点。

```
<div id="parent" style="border: 1px solid black;padding: 5px;margin: 5px;background: lightgray;">
  <p id="child1">child1</p>
  <p id="child2">child2</p>
  <div id="child3">
    <p id="child5">inner child</p>
  </div>
  <p id="child3">child4</p>
</div>
```

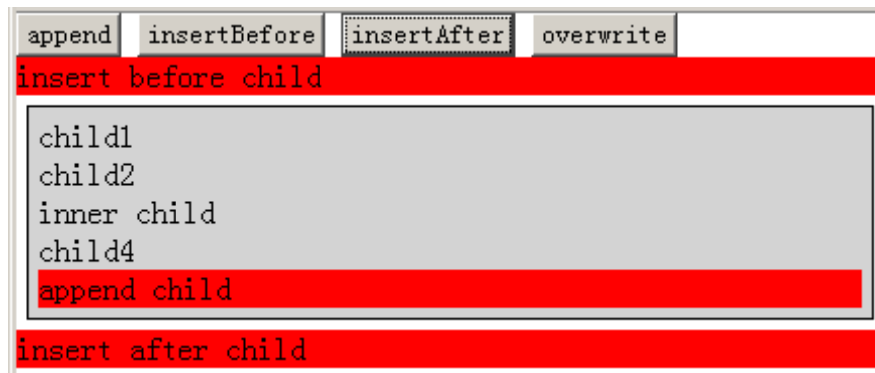
- append 是将新节点放到指定节点的最后。
- Ext.DomHelper.append('parent', {tag: 'p', cls: 'red', html: 'append child'});



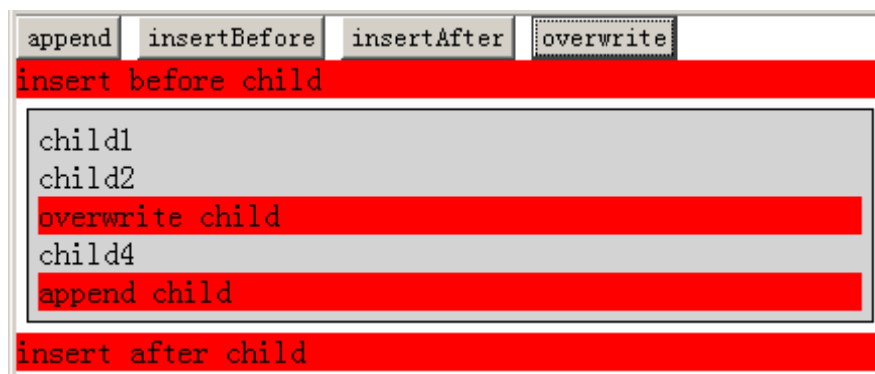
- insertBefore, 新节点插入到指定节点前面。
- Ext.DomHelper.insertBefore('parent', {tag: 'p', cls: 'red', html: 'insert before child'})



- insertAfter, 新节点插入到指定节点后面。
- Ext.DomHelper.insertAfter('parent', {tag: 'p', cls: 'red', html: 'insert before child'})



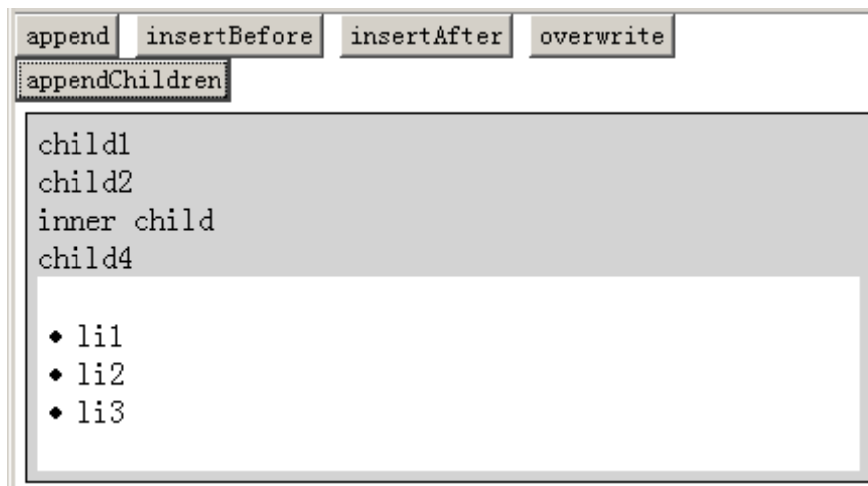
- overwrite, 会替换指定节点的 innerHTML 内容。
- Ext.DomHelper.overwrite('child3', {tag: 'p', cls: 'red', html: 'overwrite child'})



闲来无聊，也看一看 children 这个属性的用法。

```
Ext.DomHelper.append('parent', {
    tag: 'ul',
    style: 'background: white;list-style-type: disc;padding: 20px;',
    children: [
        {tag: 'li', html: 'li1'},
        {tag: 'li', html: 'li2'},
        {tag: 'li', html: 'li3'}
    ]
});
```

这样就在 parent 里添加了一个 ul 标签，ul 里包含三个 li。呵呵~炫啊。



代码见 [lingo-sample/1.1.1/08-03.html](#) 和 [lingo-sample/2.0/08-03.html](#)。

### 8.3.2. 批量生成还是需要Template模板

场景模拟：目前有三男两女的 json 数据，要输出成 html 显示出来。

```
var data = [  
  ['1', 'male', 'name1', 'descn1'],  
  ['2', 'female', 'name2', 'descn2'],  
  ['3', 'male', 'name3', 'descn3'],  
  ['4', 'female', 'name4', 'descn4'],  
  ['5', 'male', 'name5', 'descn5']  
];
```

照搬 grid 时的测试数据呢，嘿嘿。只不过这次我们用的不再是 ds, cm, grid 的方式解析输出，而是用模板自己定义输出的格式。

首先要定义一个模板

```
var t = new Ext.Template(  
  '<tr>'  
    '<td>{0}</td>'  
    '<td>{1}</td>'  
    '<td>{2}</td>'  
    '<td>{3}</td>'  
  '</tr>'  
);  
t.compile();
```



索引从 0 开始，一共 4 个元素。然后在用的时候，这样子。

```
for (var i = 0; i < data.length; i++) {  
    t.append('some-element', data[i]);  
}
```

这段代码对应 html 中的一个表格，id="some-element"是 tbody 的 id，我们使用模板为 table 增添了四行。

```
<table border="1">  
    <tbody id="some-element">  
        <tr>  
            <td>id</td>  
            <td>sex</td>  
            <td>name</td>  
            <td>descn</td>  
        </tr>  
    </tbody>  
</table>
```

最终的显示结果就是包含五行数据的表格：

id	sex	name	descn
1	male	name1	descn1
2	female	name2	descn2
3	male	name3	descn3
4	female	name4	descn4
5	male	name5	descn5

定义模板的时候，可以使用 Ext.util.Format 里的工具方法，对数据进行格式化。常用的就是 trim 去掉收尾空格和 ellipsis(10)，ellipsis 判断，当字符长度超过 10 时，自动截断字符串并在末尾添加省略号，很常用的功能哩。

在模板里使用这些函数的话也很简单，不过我不说，你还是不知道，嘿嘿

```
var t = new Ext.Template(  
    '<tr>'  
    '<td>{0}</td>'  
    '<td>{1:trim}</td>'  
    '<td>{2:trim}</td>'
```

```

        '<td>{3:ellipsis(10)}</td>',
    '</tr>'
);
t.compile();

```

如此这般，冒号加函数名称就可以实现我们的愿望了。






可惜人算终不如天算，jack 再神奇，也不可能考虑到所有的可能性，比如现在我们就想根据性别不同显示图片，jack 怕是想破了脑袋，也想不出这种可能来，所以呢，他干脆不想了，而是给咱们留了一个自定义函数的接口。

```

var t = new Ext.Template(
    '<tr>',
        '<td>{0}</td>',
        '<td>{1:this.renderSex}</td>',
        '<td>{2:trim}</td>',
        '<td>{3:ellipsis(10)}</td>',
    '</tr>'
);
t.renderSex = function(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span><img src='user_male.png' />";
    } else {
        return "<span style='color:green;font-weight:bold;'>绿女</span><img src='user_female.png' />";
    }
};
t.compile();

```

显示的红男绿女，就像我们预想的那样呈现在我们眼前了。

id	sex	name	descn
1	红男 	name1	descn1
2	绿女 	name2	descn2
3	红男 	name3	descn3
4	绿女 	name4	descn4
5	红男 	name5	descn5

你可以从 [lingo-sample/1.1.1/08-05.html](#) 和 [lingo-sample/2.0/08-05.html](#) 看到这些例子，实际上，这两个文件的内容是完全相同的。

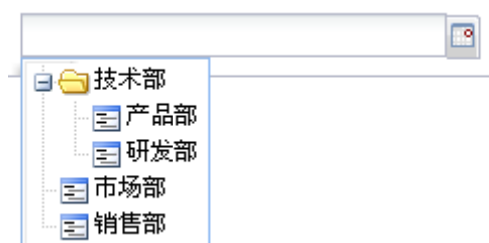
---

## 第 9 章 沉寂吧！我们要自己的控件。

[免费给的东西没人珍惜。鉴于此，决定将好玩的扩展控件都移到第九章，并且封闭代码，明码标价会让人们知道价值所在。](#)

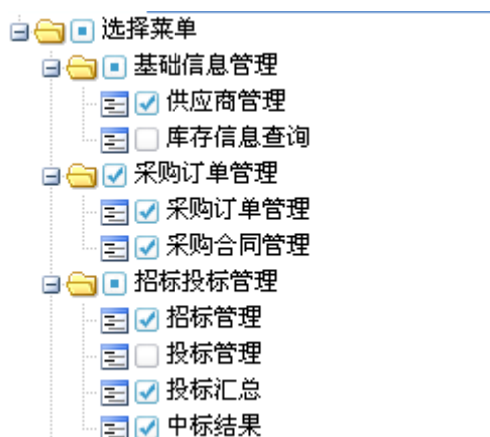
### 9.1. 下拉树形选择框TreeField

土豆拍拍，TriggerTreeField，50 元。目前仅有 1.x 实现。



### 9.2. 带全选的checkbox树形CheckBoxTree

土豆压箱底树形扩展。目前仅有 1.x 实现。



### 9.3. 带全选的checkbox的grid

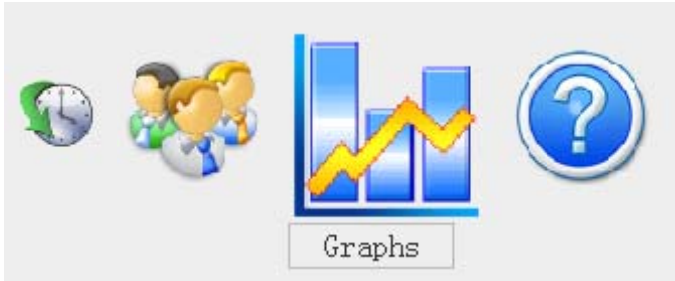
因为 2.0 中提供了 checkboxModel，所以这个 1.x 的扩展的意义并不大。

菜单设置		帮助					
新增		修改		删除			
ID	菜单名称	提示	访问路径	图片	更新时间	描述	
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
20							
21							

第 1 页，共 2 页
15 每页

9.4. fisheye

苹果机操作系统的菜单效果。



9.5. 可以设置时间的日期控件

2007-12-05 土豆新货上架

出生日期: 2007年12月05日 19时

民族: 2007年十二月

工作单位: 日 一 二 三 四 五 六

住址: 25 26 27 28 29 30 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 31 1 2 3 4 5

医疗信息

病人类型: 时间:19时 42分 今天

病历号:

过敏史:

## 9.6. JsonView实现用户卡片拖拽与右键菜单

2007-12-06 土豆新货上架

入科病人信息

序号	住院号	姓名	性别
1.	00000003	ssss	男
2.	00000006	dddd	女
3.	00000007	ttttt	男

004 基本信息 医疗信息

住院号: 00000005

姓名: wwwwww

年龄: 0岁

诊断: 流行性出...

护理等级:

005 基本信息 医疗信息

住院号: 00000008

姓名: 需人鬼

年龄: 31岁

诊断: 流行性出...

护理等级:

000 基本信息 医疗信息

住院号:

姓名:

年龄:

诊断:

护理等级:

011 基本信息 医疗信息

住院号: 00000001

姓名: 病人乙

年龄: 40岁

诊断: 流行性出...

护理等级:

012 基本信息 医疗信息

住院号:

姓名:

年龄:

诊断:

护理等级:

ddd

## 9.7. 下拉列表选择每页显示多少数据

Ext.ux.PageSizePlugin.js. 2.0 的扩展可在 [extjs.com](http://extjs.com) 上找到, 我们提供修改后的 1.x 版。

编号	名称	描述	
1	name1	descn1	
2	name2	descn2	
3	name3	descn3	
4	name4	descn4	
5	name5	descn5	

Page 1 of 1

10

per

5

15

30

50

100



## 附录 A. 常见问题乱弹

### A. 1. ext到底是收费还是免费

老多同志对这个问题感兴趣了，实际上答案很简单，jack都写到<http://www.extjs.com/license>里了，对各种情况都做了讲解。不过有些同志对英文头疼，所以在下把ext的授权形式简单讲解一下。

ext 授权大体分三种形式：

- 第一种是企业授权。

jack 说，如果你不愿意受到免费协议的限制，如果你们内部协议要求必须用企业授权，如果你愿意在经济上支持 ext 开发团队的持续发展。就可以掏钱了。

- 第二种是免费授权。

同志们别高兴，看你们乐的，免费协议是有限制的，不可能让你随使用。想用免费协议，必须满足以下的条件之一：

条件一。如果你在做一个开源项目，而这个项目里没有非开源软件。那么你可以免费用 ext。

条件二。如果你是自己玩玩，如果你是用在教学方面，或者是你没有用在盈利项目上。那么你可以免费用 ext。

条件三。如果你死也不愿意赞助 ext 开发团队，而且还非要把 ext 用在你的商业项目中，那么好，你可以用。但是不能用 ext 做软件开发库，不能用 ext 做开发工具。

是不是复杂的呀？简单来说，就是如果你不赚钱，就可以免费用。如果你非要赚钱不可，也可以用，但是不能跟 ext 抢生意，不能再把 ext 封装起来当工具库卖。

我们还要谈谈何谓LGPL，具体内容在这里，当然也是英文的<http://www.gnu.org/licenses/lgpl-3.0.txt>。我就简单说一下LGPL的含义，因为内容实在太多了。LGPL被认为能够较好保护开发者的利益的一个开源协议。简单来说，LGPL的软件，你可以用，但不能改，如果非要改也行，请把你改了的部分公开出来，否则告你哟。



- 第三种不太明白，名字是 OEM / Reseller License

你要是想把 ext 作为开发库（software development library）或者工具包（toolkit）来卖，就需要跟 ext 开发团队搞一个专门的协作授权了，因为免费授权是不允许做开发库和工具包的。

然后他说了搞合作的好处，比如可以不用受到 LGPL 的限制，你的产品就成了市场上惟一个基于 ext 开发团队官方支持的产品了，你也获得了更多的合作机会，也获得了 ext 团队直接授权的技术支持。

感觉，说的太玄了，要是你想做一套 ide，还是去跟 jack 好好谈谈吧。

大体说了一下，大家应该有认识了。ext 提供多种授权，选择一种最适合自己的就好了。

## A. 2. 怎么查看ext2 里的api文档

因为 ext2 里读取 api 文档的时候要使用 ajax，而在本地文件系统 ajax 返回的状态码一直是失败，所以无法正常显示页面，解决方法有两个：

- 一般的方法是，将整个 ext2 包复制到 iis 或者 tomcat 这类服务器上，然后通过服务器访问 api 文档，这样 ajax 就可以返回正常结果。
- 实际上 extjs.com 官网上有人发布过 localXHR.js，只要导入这个 js 就可以在本地文件系统使用 ajax。找到这个 localXHR.js（在 lingo-sample/1.1.1/下可以找到），复制到 docs 目录下，然后在 index.html 中加入 `<script src="localXHR.js"></script>`，注意，这一行要加在 ext-all.js 的后面，然后直接打开 index.html 就可以查看文档了。

## A. 3. 如何在页面中引用ext

ext 只是单纯的 js，引用方式和平常你使用外部 js 文件的方式一样。

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

放入工程中 ext 需要包括以下几部分，ext-all.js，adapter 目录，resources 目录。

- ext-all.js 是对所有源文件压缩合并后的结果，包含了所有 ext 的控件

开发时可以考虑使用 ext-all-debug.js，这个是未经压缩的版本，通过 firebug 可以找到具体哪行出现问题，如果使用 ie，也可以用其中附加的 Ext.log 进行调试，但功能没有 firebug 强劲。

- adapter 目录下是各种适配器，用的时候选择一种适配器即可。

目前提供的有 ext-base.js, prototype, yui 和 jquery。ext 在这些核心库的基础上构筑起强大的功能，开发者根据自己的实际需要选择对应的适配器，便可以在之前的经验基础上进行 ext 的开发。

- resources 目录是 css 和图片资源，不一定和 js 脚本放在一起，保持 css 和 image 的对应位置就可以。

使用 ext 的样式和图片，只需要在页面中引入 ext-all.css。

```
<link rel="stylesheet" type="text/css"
href="../../resources/css/ext-all.css" />
```

- 如果需要国际化支持，还需要从 build 目录下复制 locale 目录，导入到项目中，下面另行介绍。

## A. 4. 想把弹出对话框单独拿出来用的看这里

别用ext了，建议去看看lightbox，看是不是你要的效果，<http://www.huddletogogether.com/projects/lightbox/>。

## A. 5. 想把日期选择框单独拿出来用的看这里

别用ext了，js的日期选择控件数不胜数，这里推荐一个，<http://www.dynarch.com/projects/calendar/>。

## A. 6. 听说有人现在还不会汉化ext

ext 提供了国际化的脚本，这些东东都在 build/locale/目录下，你只需要把对应语言的脚本加入页面就可以了，比如我们要 chinese。

```
<script type="text/javascript"
src="../../adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../../ext-all.js"></script>
<script type="text/javascript"
src="../../build/locale/ext-lang-zh_CN.js"></script>
```

注意，把翻译的脚本放在 `ext-all.js` 之后，翻译脚本是 `utf-8` 编码，如果你需要 `gb2312` 或者其他编码格式，请自行转换编码。

嗯，那个问我为啥 `zh_CN` 是中文的人，请把这个背下来，下次就不会有这个疑问了。

## A. 7. 碰到使用ajax获得数据，或者提交数据出现乱码

英文情况下不会出现乱码，用了中文才可能乱掉，这是因为咱们的 `win` 操作系统，保存文件的默认编码是 `gb2312`，而 `ajax` 传输数据的默认编码是 `utf-8`，推荐大家将数据格式统一为 `utf-8`，不但可以解决眼前的乱码问题，对以后扩展为多语言也有好处。

## A. 8. TabPanel使用autoLoad加载的页面中的js脚本没有执行

使用 `scripts:true` 属性，可以执行 `TabPanel` 加载页面中的 `js` 脚本，如下所示：

```
var tabItem = tabPanel.add({
    id:title,
    title:title,
    closable:true,
    autoScroll:true,
    autoLoad: {url:url, scripts:true}
});
tabPanel.activate(tabItem);
```

---