掏钱学ext

Version: 0.1.2

说在前头的

有些话是要说在前头的，请详细阅读这些东西，免得后悔莫及。

为啥要掏钱学ext?

1. www.extjs.com是从yui-ext发展来的一套ajax控件，在下认为它是当今最好用的js控件库了，非常有学习的价值。

   如果没听说过extjs或者对extjs没有兴趣，请不要继续看下去。

2. extjs是一套完整的富客户端解决方案，也因为功能完整，造成ext-all.js有400多k，请注意，这还是压缩后的大小，而且因为是基于js和css的功能实现，对客户端机器性能也有一定的要求，比如不支持ie6以下的版本。

   如果您的项目建立在不畅通的网络条件下，对响应速度要求严格，或者客户端操作系统过于陈旧，请不要选择extjs，也没有必要继续看下去了。

3. 关键性问题：

   ### 注意

   为什么extjs有免费协议，而这个文档却是收费文档?

   如果您对靠自己的劳动赚取生活费有异议的话，请不要继续看下去了。

   如果您认为我们的劳动不值得您掏钱的话，请不要继续看下去了。

   如果您认为我们光靠喝西北风就可以活下去，请不要继续看下去了。

   如果您认为我们是好人，请不要继续看下去了。

很高兴您赞同我们上述的观点，并且愿意继续浏览下面的内容，如果您对上述任何一点有异议，那么请不要继续看下去了。

注意

本教程将每天更新一点点，全部完成后便开始封闭收费。预订可享受八折优惠。技术服务，收费咨询，找土豆(QQ)1184282。

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目录下找你需要的例子，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，所有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？

1. 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。

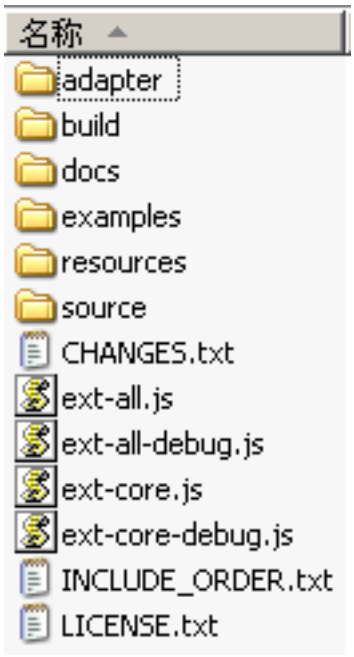
1.7. 入门之前，都看helloworld。

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

1. 先去http://www.extjs.com/download下载zip格式的发布包

2. 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。



3. 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.');
});
</script>
```

4. 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。

1. 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。

2. build/locale/ext-lang-zh_CN.js是中文翻译。

3. 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。），也没有默认的统计功能，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。

1. 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。

2. 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。

3. 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。

4. 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，对应在html里应该有一个 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。

5. 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。

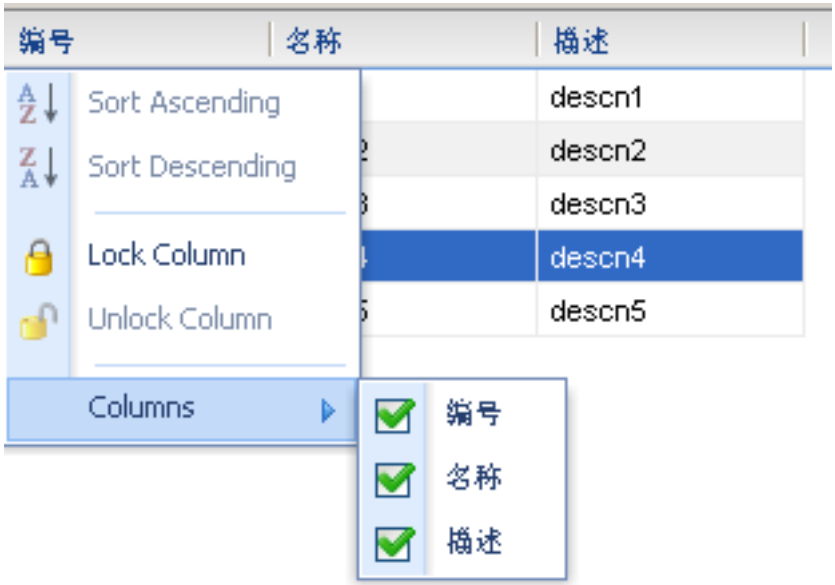
因为其他地方都一样，我就不多说了，html例子在是lingo-sample/2.0目录下的02-01.html。

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

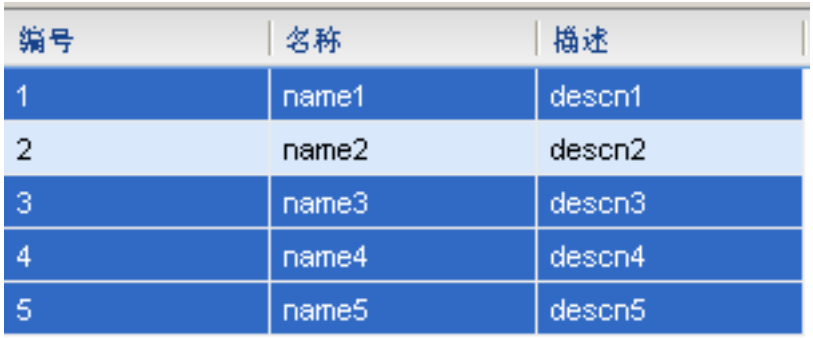
从截图上看，少了斑马条，下边多了一条线，应该只是css有所不同吧。

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。

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。



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。



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

2.4.1. 自主决定每列的宽度

到这里，估计同志们就想啦，怎么所有宽度都是一样的啊，这样不爽啊，万一哪列不够宽我还要自己动手拉，岂不是很麻烦？其实cm是支持给每列设置宽度啦，如果你不设置的话，它才会取一个默认值，默认宽度是100哟。

其实蛮简单的，只要给那列设置上一个width属性就好了：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',width:20},
    {header:'名称',dataIndex:'name',width:80},
    {header:'描述',dataIndex:'descn',width:200}
]);
```

原来等宽的表格就变成这样啦。

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

呼呼，例子就在lingo-sample/1.1.1/02-01a.html和lingo-sample/2.0/02-01a.html。

可有人又抱怨啦，好麻烦，还要让我一个一个算每列多宽，让它们自动填满grid好不好呀？嗯，可以满足，1.x里需要给grid配置一个autoSizeColumns，让它等于true就好了，它会自动计算每列宽度，把grid填充满。

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

```
    autoSizeColumns: true
});
```

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

2.0里就不能再用这个参数了，你需要的是viewConfig中的forceFit，从名字可以看出来，这个是给GridView用的配置，它告诉视图层重新计算所有列宽，填充grid。

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

1.x与2.0不同的是，虽然两者都是自动填充满grid，但1.x中自己设置的宽度值不会产生效果，最后每列都是等宽的，但是2.0的似乎是换算成比例显示出来了。嗯，还是2.0先进呀。

还有一点不同，1.x中的autoSizeColumns只在最初一次自动计算列宽，如果再进行拖拽改变列宽，就不再计算了。而2.0会在每次拖拽时重新计算其他列的大小，既不让超出，也不会出现过多空余撒。

1.x的例子在lingo-sample/1.1.1/02-01b.html。2.0的例子在lingo-sample/2.0/02-01b.html。

不过呢，除了autoSizeColumns/forceFit全部重新计算宽度以外，还可以考虑考虑autoExpandColumn，它可以让指定的列宽自动伸展，从而填充整个表格。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',width:20},
    {header:'名称',dataIndex:'name',width:80},
    {id:'descn',header:'描述',dataIndex:'descn',width:200}
]);

var grid = new Ext.grid.Grid('grid', {
    ds: ds,
    cm: cm,
    autoExpandColumn: 'descn'
});
```

autoExpandColumn只能指定一列的id，注意呀，是id，原来咱们设置的cm里都没有id，现在为了使用autoExpandColumn，我还去专门给cm的descn设置了id，这样渲染的时候descn就自动延伸，直至充满整个表格了。

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

嘿嘿~，只延展一列还是比较好控制的。例子在lingo-sample/1.1.1/02-01c.html和lingo-sample/2.0/02-01c.html。

2.4.2. 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'}
]);
```

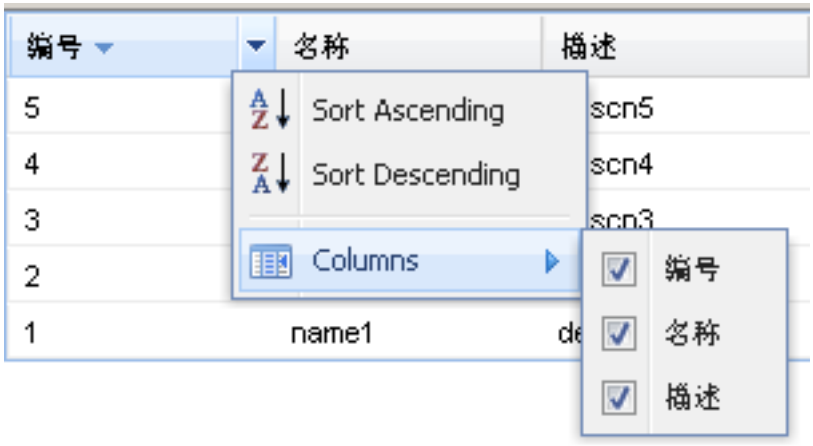
如果你英语还可以，或者懂得查字典的话（软件翻译也算），那么你就会知道，多出来的这个sortable属性应该是可以排序的意思。现在咱们试一下改动后的效果。

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，表格里的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？！-_-。

2.4.3. 中文排序是个大问题

你知道为啥外国人都都用ascii码排序的时候，咱们非要用拼音顺序排序呢？真是个大问题啊，ext自动排好的中文在咱们看起来却是一团糟，咋办呀？

测试数据是小学生常念的“啊啵呲嘚咯嗑嚯佛”：

```
var data = [
    ['1','啊','descn1'],
    ['2','啵','descn2'],
    ['3','呲','descn3'],
    ['4','嘚','descn4'],
    ['5','咯','descn5']
];
```

为了让咱们立刻可以看到排序效果，咱们为Ext.data.Store设置一个默认的排序方式。

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

```
    sortInfo: {field: "name", direction: "ASC"}
});
```

注意多出来的那行sortInfo，它对应一个JsonObject，field代表排序的列名，direction代表排序方式，ASC是正序，DESC是逆序。

好啦，现在你会看到默认的排序方式把咱们的数据搅成什么样子了。

编	名称	描述
3	呲	descn3
5	咯	descn5
1	啊	descn1
2	啵	descn2
4	嘚	descn4

加上这段代码吧，它会让你感觉好受些，一定要注意的是，这段代码我是从http://jstang.5d6d.com/thread-163-1-1.html看到的，发这帖子的兄弟说不知道是从哪里看到的，所以咱们在这里没法指名作者了：

```
Ext.data.Store.prototype.applySort = function() {
    if (this.sortInfo && !this.remoteSort) {
        var s = this.sortInfo, f = s.field;
        var st = this.fields.get(f).sortType;
        var fn = function(r1, r2) {
            var v1 = st(r1.data[f]), v2 = st(r2.data[f]);
            if (typeof(v1) == "string") {
                return v1.localeCompare(v2);
            }
            return v1 > v2 ? 1 : (v1 < v2 ? -1 : 0);
        };
        this.data.sort(s.direction, fn);
        if(this.snapshot && this.snapshot != this.data) {
            this.snapshot.sort(s.direction, fn);
        }
    }
};
```

这段代码是重写Ext.data.Store的applySort函数，你可以把这个加到ext-all.js文件的最后，或者放到html页面的最上边，反正是在ext初始化之后，实际代码调用之前就可以了。

说真的，没去仔细比较过这段代码与原来代码的区别，不过呢，感觉问题应该出在v1.localeCompare(v2)上，这个估计是支持国际化的字符串比较，让我们可以得到拼音顺序的排序结果，嘿嘿~，cool啊。

编	名称	描述
1	啊	descn1
2	啵	descn2
3	呲	descn3
4	嘚	descn4
5	咯	descn5

1.x的代码在lingo-sample/1.1.1/02-01d.html。2.0的代码在lingo-sample/2.0/02-01d.html。

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。

咱们先来个图片。

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 + "，随便用吧。" +
        "\")'>";
    return str;
}
```

来看看我们可以在render里用到多少参数：

1. value是当前单元格的值

2. 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。

3. record是这行的所有数据，你想要什么，record.data["id"]这样就获得了。

4. 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。

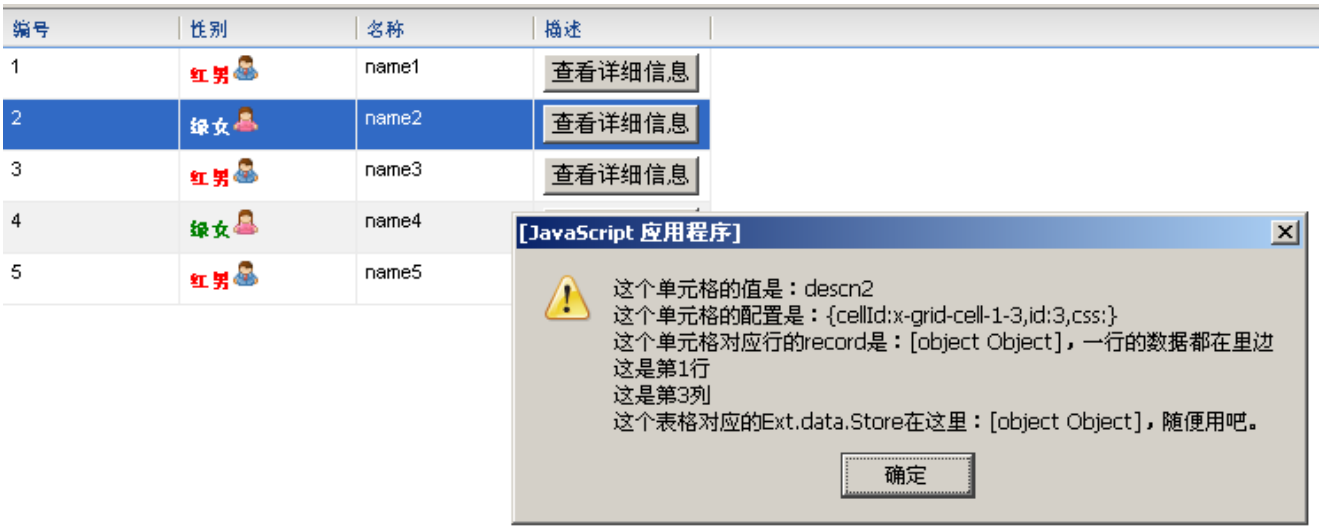
5. columnIndex列号太简单了。

6. 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。



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。

对了。

那个有人想知道怎么从后台取得date日期型数据，然后交给grid让它做格式化呢。不过你也知道咱们返回的json里都是数字和字符串，幸好ext已经为咱们准备好了一切，让咱们看看该如何做吧。

1. 第一步，我们要加一列数据了，代表日期的呢。

```
var data = [
    ['1','name1','descn1','1970-01-15T02:58:04'],
    ['2','name2','descn2','1970-01-15T02:58:04'],
    ['3','name3','descn3','1970-01-15T02:58:04'],
    ['4','name4','descn4','1970-01-15T02:58:04'],
```

```
    ['5','name5','descn5','1970-01-15T02:58:04']
];
```

看一下啦，最后一列是新加的，这一列的数据可都是日期哟。

2. 第一步是把json的字符串转换成日期。这一步我们在reader里配置。

```
var ds = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'},
        {name: 'date', type:'date',dateFormat:'Y-m-dTH:i:s'}
    ])
});
```

注意reader中对应字段的最后一行，我们不只有name，还多了type和dateFormat两个参数。这个type告诉reader要把这一列当作日期类型对待，dateFormat则告诉我们怎么把得到的字符串转换成日期，Y是年啦，m是月啦，d是日期啦，H是小时，i是分钟，s是秒。。。我是不是太啰唆啦。。。不说了。

3. 第二步是把ds里的date按咱们定的格式显示到grid里，这一步我们在cm里配置renderer，不过这次不用咱们自己写函数了，换上ext提供的格式化工具看看。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'},
    {header:'日期',dataIndex:'date',type:'date',renderer: Ext.util.Format.dateRenderer('Y年m月d日')}
]);
```

还是看日期那样行呗，这里我们只需要看renderer里是怎么写的就好啦。你看到我们对应的值是Ext.util.Format.dateRenderer('Y年m月d日')，哈哈~这可是ext提供的格式化方法，有了这个咱们就不用再去专门写格式化函数了，封装好的工具类多好呀。

1.x的例子在lingo-sample/1.1.1/02-02a.html。2.0的例子在lingo-sample/2.0/02-02a.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

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

自动计算行号呢，总是伴随着一个小小的问题。要是咱们删除了grid中间一行，就该看到行号不连续了，唉，虽然不是很明白为啥删除一行却不使用ds.load()重新加载，但是咱们还是有讨论的必要不是？

咱们在html里加上个按钮，然后在按下按钮的时候删除grid的第二行：

```
Ext.get('remove').on('click', function() {
    ds.remove(ds.getAt(1));
});
```

html多一个button，把它的id设置成remove就好了，我们使用Ext.get()获得这个按钮，然后监听它的click事件，执行ds.remove(ds.getAt(1))，就是获得第2行，然后删掉好了。最后结果变成这样：

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

删除第二行

1,3,4,5的组合让人心生烦恼呢，如果想要1,2,3,4那就要多一条语句了。

```
Ext.get('remove').on('click', function() {
    ds.remove(ds.getAt(1));
    grid.view.refresh();
});
```

一句refresh()就可以刷新咱们的视图，grid.view获得的是GridView对象，它可是专门用来显示的，刷新一下它就会调用那些renderer()去重新计算所有行号了。这样我们便得到我们想要的：

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

删除第二行

好的。1.x的代码在lingo-sample/1.1.1/02-03a.html，2.0的代码在lingo-sample/2.0/02-03a.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。

		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

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。

注意

对了，有人提到，既然有了checkbox那就不应该在单击行的时候选中这一行了，因为会自动变成选中这一行，而不是多选呢。这样就需要取消掉原来选择行的事件处理方法：

```
var sm = new Ext.grid.CheckboxSelectionModel({handleMouseDown: Ext.emptyFn});
```

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.		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

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。

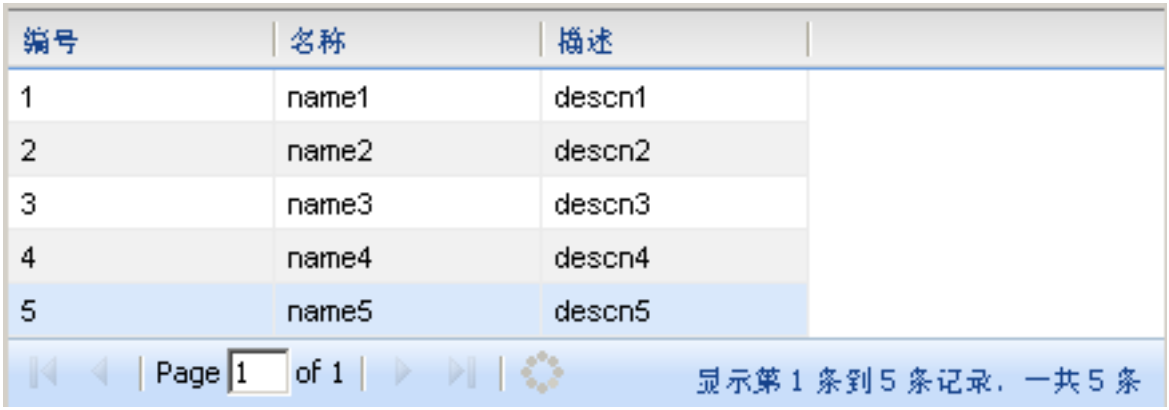
如果你有一千条信息，一次都输出到grid里，然后让客户用下拉条一点儿一点儿去找吧。我们这里可是要做一个分页效果了，不用滚动屏幕，或者滚动一下就可以看到本页显示的数据，如果想看其他的只需要翻页就可以了。同志们，加强客户体验呀。

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。



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么东西过来干嘛呀？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。

1. pageSize，是每页显示几条数据。

2. displayInfo，跟下面的配置有关，如果是false就不会显示提示信息。

3. 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。

4. 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。

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，再咕哝分页的配置。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。

不过还是要注意一点，与1.x中不同的是，如果配置了分页工具条，ds.load()就必须在构造grid以后才能执行，否则分页工具条会不起作用。看来分页工具条会把自己和ds做一些关联，来完成与grid共享数据模型的。

对了，还有一点不同，1.x中分页条不会随着浏览器的大小改变，自动放缩，这点在2.0中也解决了。

2.0的例子在lingo-sample/2.0/02-05.html。

注意

嗯，有个同志在使用中提到了一个挺奇怪的需求：让pagingToolbar右对齐。

这应该是非常难碰到的需求，ext中没有直接提供配置的参数，我想到用float:right放到css中，让div飘到右边，不过试验的结果是style="float:right"没有效果，只好使用cls指定一个定义好的class。

这个问题告诉我们，使用style会自动剔除一部分css样式，不知道它怎么判断的，关键时刻还是使用cls保险。

2.0的例子在lingo-sample/2.0/02-05a.html。1.x中因为提供自定义css的途径，所以暂时不知道如何实现，怕要自己写一个PagingToolbar了。

2.7.3. 迫不得已，要加上后台脚本了。

grid会每次都显示ds中所有的数据，咱们没法利用静态数据好好演示分页，于是，必须写后台脚本，让ext与后台进行数据交互才能看到真实的分页效果。

咱们尽量在原来的基础上修改啊，把注意力集中在关键部位，一次性突破。

我不会其他语言，后台就用jsp写啦。有个好消息是只要返回的数据格式一样，ext才不管后台是什么写的呢。也就是这样，不管你以后用什么实现后台，前台的ext代码都一样。

```
<%
String start = request.getParameter("start");
String limit = request.getParameter("limit");
try {
    int index = Integer.parseInt(start);
    int pageSize = Integer.parseInt(limit);

    String json = "{totalProperty:100,root:[";
    for (int i = index; i < pageSize + index; i++) {
        json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
        if (i != pageSize + index - 1) {
            json += ",";
        }
    }
    json += "]}";
    response.getWriter().write(json);
} catch(Exception ex) {
}
%>
```

下面我们来解读这段jsp代码：

1. 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。

2. 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。

3. 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。

1. 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。

2. 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。

3. 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。

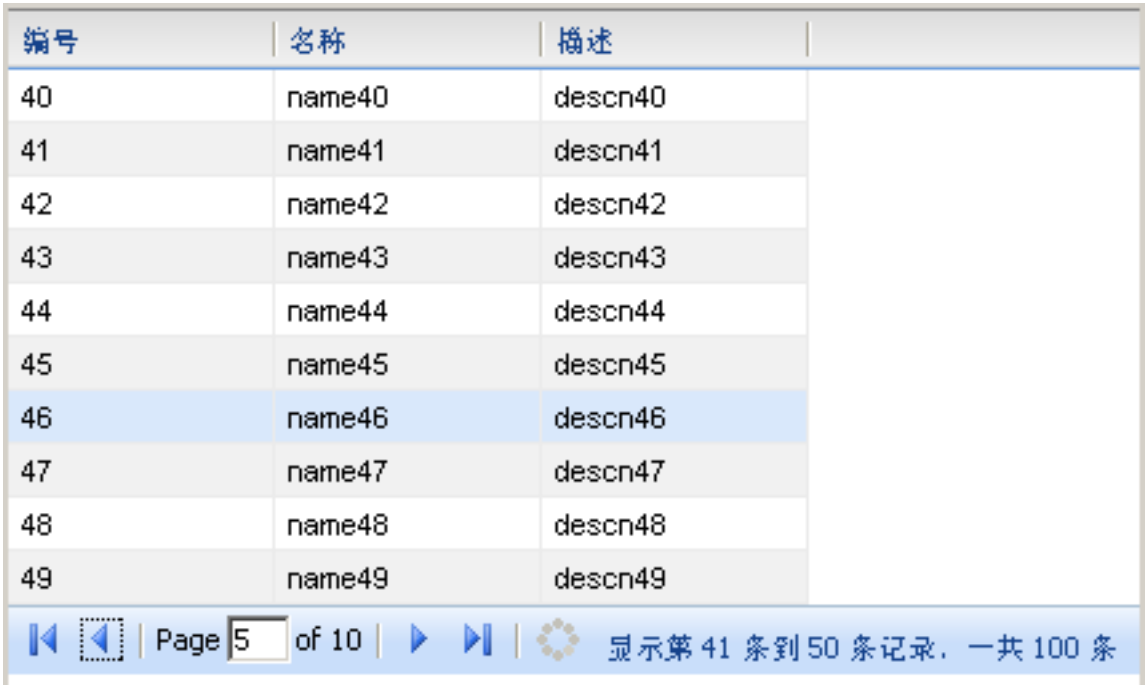
```
ds.load({params:{start:0,limit:10}});
```

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 。

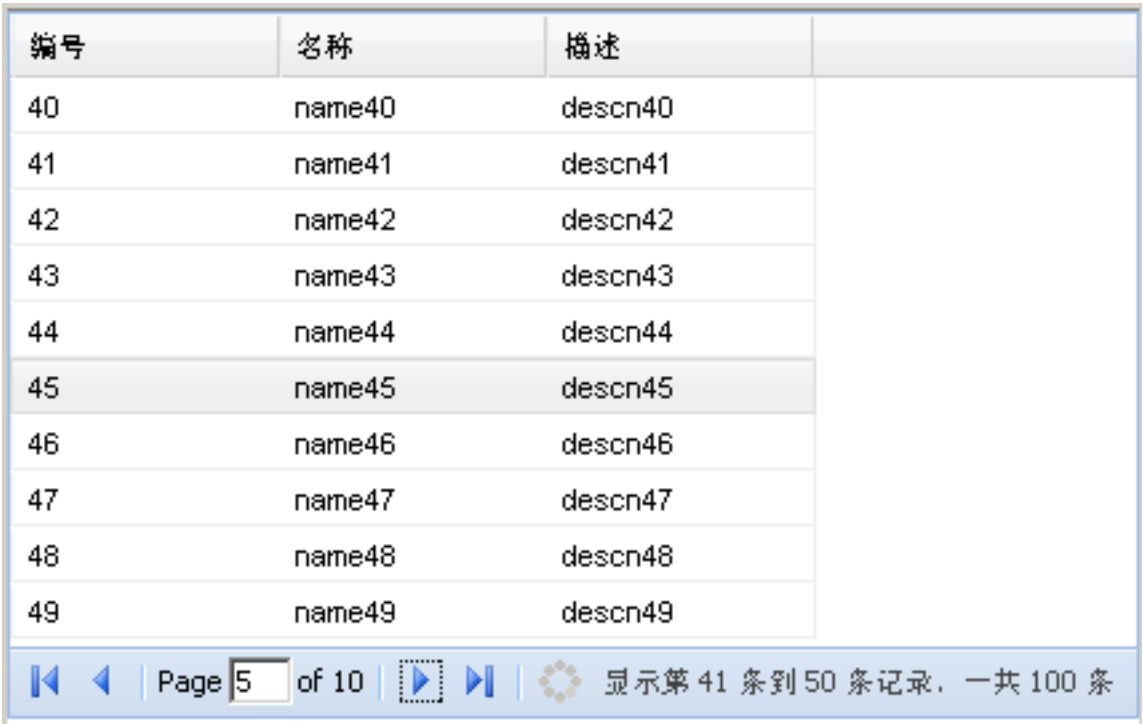
最后，我们就可以使用分页条上那些按钮试试分页了。呵呵~就这么简单。

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：



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。

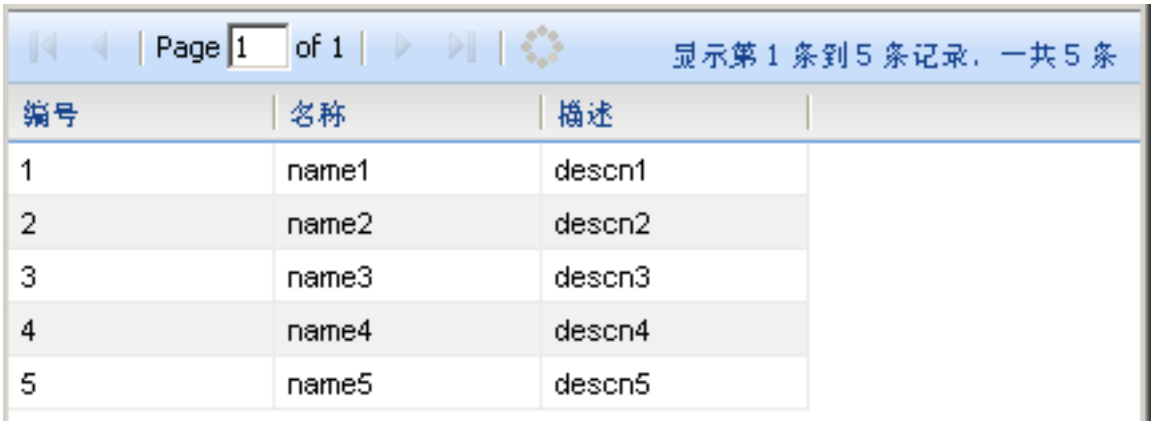
2.0的例子在lingo-sample/2.0/02-06.html，jsp文件在lingo-sample/2.0/grid.jsp，下面是它的截图：



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FootPanel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。

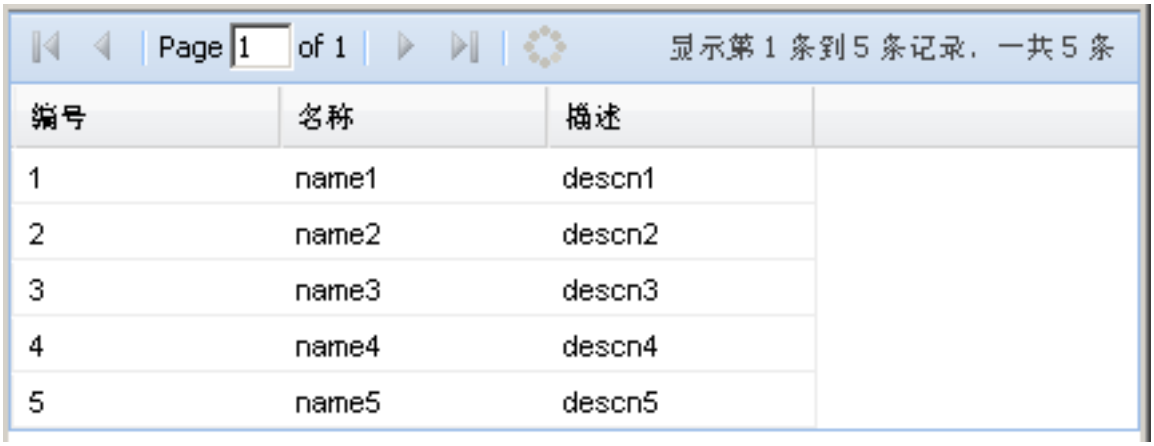
```
var gridHead = grid.getView().getHeaderPanel(true);

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'
});
```



1.x的例子可以在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"
    })
});
```



2.0的例子可以在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。

2.7.5. 谣言说ext不支持前台排序

前台排序是指，extremetable里那样，一次性从后台把所有数据都读取到客户端，然后客户端自动判断每次显示多少条数据，这样分页的时候就不用再去后台读取了，对于小数据量的分页是非常有利的。

可惜ext里没这种东西，它把所有得到的数据一股脑地显示到表格里，你的pageSize设置了多少都没用。

其实，确切的说，ext在默认情况下没有这项功能，不过2.0的examples/locale/里提供了一个PagingMemoryProxy.js扩展，它让我们可以从本地数组读取数据，分页显示。天哪，快看看他是怎么用的吧。

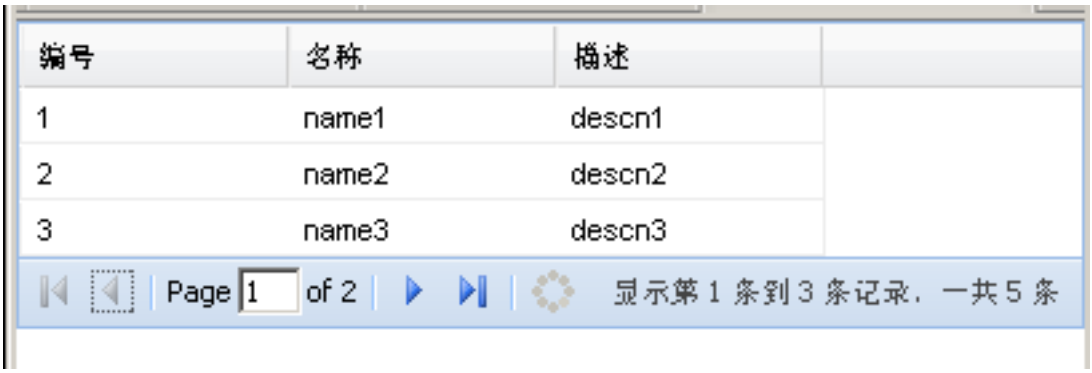
1. 把PagingMemoryProxy.js从examples/locale/目录下copy过来，导入html里：

```
<script type="text/javascript" src="PagingMemoryProxy.js"></script>
```

2. 把以前的MemoryProxy换成PagingMemoryProxy。

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

3. 好了，现在直接调用ds.load({params:{start:0,limit:3}})，让我们把头三条记录显示出来吧。



虽然它只能从本地js数组中分页读取数据，但也算是一大进步啦，你可以在前台生成js数组，或者ajax读取过来，再传递给PagingMemoryProxy，以此实现分页呢。

非常幸运的是PagingMemoryProxy.js不用任何修改就可以放到1.x里跑起来，感谢上帝。

1.x的例子在lingo-sample/1.1.1/02-07a.html，2.0的例子在lingo-sample/2.0/02-07a.html。

2.8. 爱生活，EditorGrid。

读几遍吧，广告词很酷的。

是不是觉得excel很酷，用户想要几行就添几行，不用就删除，最后一次保存就好了，要不然你还要一页一页的写，每次每次点提交，好麻烦哦。不过为啥咱们不直接用excel呢？因为我不会对excel做更精确的控制，比如每行限制多少列，对某一列如何进行完全控制，比如只能选择我们确定的某些东西，比如某些列不能为空，如果这些不满足的时候，最好弹出个对话框给个提示什么的。

当当当，现在EditGrid要登场啦。瞧瞧我们要解决什么问题，动态添加行，删除行，然后咱们还要可以动态修改某个单元格，这些单元格咱们暂定不能为空好了，当然提交的时候要检测这一切才行，否则不让你提交哟，验证信息的话必须提示一下呢。如何？这些该够用了吧，如果ok的话，让咱们继续哟。

2.8.1. 旋动舞步，看我们怎么把这个EditorGrid炫出来。

第一步都是从定义列开始的，你觉得这个多了个Edit的grid与之前会有什么不同呢？

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',editor:new Ext.grid.GridEditor(new Ext.form.TextField({
```

```
        allowBlank: false
    }))},
    {header:'名称',dataIndex:'name',editor:new Ext.grid.GridEditor(new Ext.form.TextField({
        allowBlank: false
    }))},
    {header:'描述',dataIndex:'descn',editor:new Ext.grid.GridEditor(new Ext.form.TextField({
        allowBlank: false
    }))}
]);
```

很不好意思，这一脚可以迈得有点儿大了，一下子多了这么多东西不知道你能不能受得了。实际上我希望你能尽量保持冷静，和我一起看完这部分再说。这里我们给每列增加了editor属性，里边的属性可是完全一样的TextField，嗯，嗯，让我们假设它就是用来编辑单元格的。到底是不是呢？让我们接着往下看。

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

结果我们第二步就跳到终点了，除了在Grid前头套上了Editor的帽子以外，其他的部分都没有变化。

好了，看结果吧，你现在可以随意修改单元格了，用TextField的方式，不过记得别让单元格变空，否则不会让你改的。

默认是需要咱们双击单元格，才能激活编辑器的，不过你也可以给grid配置上clicksToEdit:1，这样就变成单击修改了，孰优孰劣人均抉择啦。

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

为啥TextField这么聪明呢？说实在的，是我们告诉它不能为空，它只是履行自己的义务而已。那么，你觉得咱们是通过什么告诉它不能为空呢？

如果，allow是允许的意思，blank是空白的意思，那么allowBlank:false会不会是在说不能为空呢？嗯，我认为你是猜对了呢。

例子见lingo-sample/1.1.1/02-08.html。2.0版本的请改一改，而例子就在lingo-sample/2.0/02-08.html。

2.8.2. 添加一行，再把它踢掉

OK，OK。我知道只是可以编辑一下不会满足你的，所以现在我们要动态一些，想要几行就弄几行出来，这个主意怎么样？

首先做一个。

```
var Record = Ext.data.Record.create([
    {name: 'id', type: 'string'},
    {name: 'name', type: 'string'},
    {name: 'descn', type: 'string'}
]);
```

现在我们要两个按钮，一个可以添加行，另一个做反向工程，暂且就把他们放到grid的头部。

```
var gridHead = grid.getView().getHeaderPanel(true);
var tb = new Ext.Toolbar(gridHead, ['-', {
    text: '添加一行',
    handler: function(){
        var initValue = {id:'',name:'',descn:''};
        var p = new Record(initValue);

        grid.stopEditing();
        ds.insert(0, p);
        grid.startEditing(0, 0);

        p.dirty = true;
        p.modified = initValue;
        if(ds.modified.indexOf(p) == -1){
            ds.modified.push(p);
        }
    }
}, '-', {
    text: '删除一行',
    handler: function(){
        Ext.Msg.confirm('信息', '确定要删除？', function(btn){
            if (btn == 'yes') {
                var sm = grid.getSelectionModel();
                var cell = sm.getSelectedCell();

                var record = ds.getAt(cell[0]);
                ds.remove(record);
            }
        });
    }
}, '-']);
```

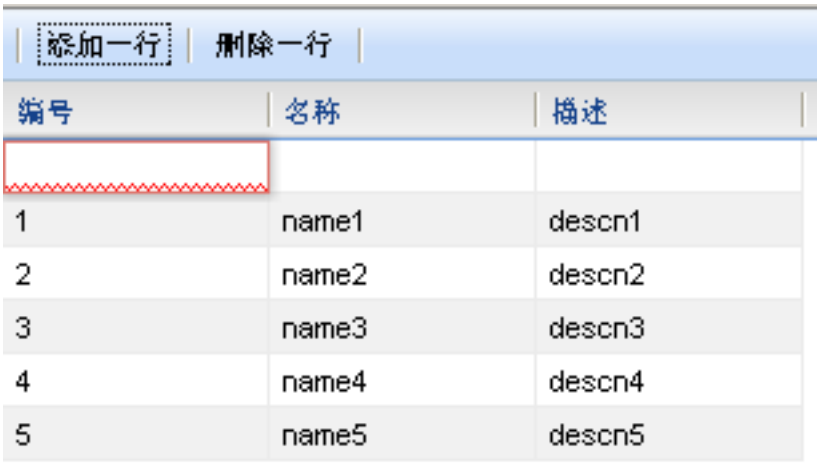
咱们就是通过gridHead创建一个Toolbar，然后在这个工具条里放两个按钮。一个叫“添加一行”，另一个“删除一行”，它们用text定义按钮显示的文字，handler定义按钮被按下的时候执行的函数。

好了，看看添加一行的按钮做了什么，首先new一个刚才定义的Record，记得给里边的属性赋值，要不EditorGrid最后显示的东西就乱套了。紧接着，关闭表格的编辑状态，然后把咱们刚才创建的Record插入ds的第一行，数据模型一改，表格也立即变化，酷啊。startingEditing()激活第一行第一列的编辑状态，提示你最好现在就开始写数据。dirty和modifed的操作是为了好看，grid会给予脏数据部分特殊显示。

同志们请注意，同志们请注意。p.dirty = true;到ds.modified.push(p);这段尤为重要，市场上常见的新增例子中都没有这部分，会导致提交的时候无法校验新增行的数据。

说实话，删除一行的按钮函数是我自己想出来的，不知道效率是否有问题，我呢，先获得grid的选择模型，从选择模型获得选中的单元格，这个单元格有两个属性，第一个是行号，第二个是列号。好的，我们通过行号得到ds这一行对应的Record，然后删掉它。于是你选中的这一行消失了。

就这样，我们完成了添加和删除的操作，稍微尝试了一下Toolbar的使用，另外指出一下，两个按钮之间的'-'会变成分隔符，让工具条感觉更酷一些，就像下面的一样：



例子lingo-sample/1.1.1/02-09.html。2.0版的例子放在lingo-sample/2.0/02-09.html，说实话改动可不小呢，不过都在可控范围内，比如gridHead换成tbar呀，EditorGrid换成EditorGridPanel呢。

2.8.3. 一切就绪，你可以按保存按钮了。

同志们注意啦，同志们注意啦。必须使用我们上节中的添加行方式，才能保证从ds.modified获得新增的行，否则可能出现校验错误。

好了，表面工作就是在toolbar上加个保存按钮：

```
{
    text: '保存',
    handler: function(){
        var m = ds.modified.slice(0);
        var jsonArray = [];
        Ext.each(m, function(item) {
            jsonArray.push(item.data);
        });

        Ext.lib.Ajax.request(
            'POST',
            'grid2.jsp',
            {success: function(response){
                Ext.Msg.alert('信息', response.responseText, function(){
                    ds.reload();
                });
            },failure: function(){
                Ext.Msg.alert("错误", "与后台联系的时候出现了问题");
            }},
            'data=' + encodeURIComponent(Ext.encode(jsonArray))
        );
    }
}
```

大概流程是这样，首先获得ds中修改过的数据，然后放到json数组里，用ajax发送给后台，最后根据后台的操作显示成功或者失败信息。

ds.modified.slice(0)，本来ds.modified就是经过修改的数组了，为啥还要slice(0)呢？这里主要是怕对原数据造成影响。slice()函数是对数组进行切割，0说明从第一个元素一直切到最后，这样就等于复制了一个数组，你再修改获得的数组，也不会对原来的数组产生影响。

下面把这些数据组装成简单数组，因为m里的都是Record，不是简单对象，咱们只需要里边的数据，所以只需要取出Record的data属性就可以了。其实使用普通循环也能实现，不过这里我们还是玩玩ext提供的each()函数。

Ext.each(array, fn)，这里用的两个参数，第一个是数组，第二个是函数，启动each()后，会循环数组里的每个元素，然后一个个的传给回调函数处理，这是简化循环的一种途径，尝试一下吧。

ajax部分比较单纯，设置好http方法，访问url，成功时的回调函数，失败时的回调函数，最后把参数转换成字符串，一气发送到后台。而在成功之后一般是执行ds.reload重新读取数据，保证grid不会再显示那些。

另外介绍一个ds的参数pruneModifiedRecords，把这个设置为true的话，每次进行remove或load操作时ds回去自动清除modified记录，避免出现下次提交还会把上次那些modified信息都带上的现象。

1.x的例子在lingo-sample/1.1.1/02-10.html。2.0的例子在lingo-sample/2.0/02-10.html。

2.8.4. 天马行空，保证提交的数据绝对有效

不知道你有没有发现一个问题，咱们这里设置的allowBlank:false只能限制修改已有数据的单元格，如果新建了一行，里边的空白是无法检测到的。客户没有修改这些空白，也可以直接提交数据，这样把无效数据提交到后台显然是不可取的。

很可惜的是，EditorGrid对这部分并没有默认的实现，我们需要自己在提交前进行判断。

```
for (var i = 0; i < m.length; i++) {
    var record = m[i];
    var fields = record.fields.keys;

    for (var j = 0; j < fields.length; j++) {
        var name = fields[j];
        var value = record.data[name];

        var colIndex = cm.findColumnIndex(name);
        var rowIndex = ds.indexOfId(record.id);
        var editor = cm.getCellEditor(colIndex).field;

        if (!editor.validateValue(value)) {
            Ext.Msg.alert('提示', '请确保输入的数据正确。', function(){
                grid.startEditing(rowIndex, colIndex);
            });
            return;
        }
    }
}
```

捡重点部位推敲一下。

1. 首先循环m，获得每个被修改的行，var record = m[i];代表某一行，var fields = record.fields.keys;是一共有多少列。
2. 第二步，按列循环，获得当前行里每一个单元格。具体获得的有name列名，value单元格的数值，colIndex列号，rowIndex行号，editor这一列用到的编辑器。
3. 准备充分之后，就剩下校验了，已经获得了editor，但直接调用editor.isValid()方法总会出现el不存在的错误，无奈之下使用editor.validateValue(value)的方式进行校验，这是isValid()内部的实现方法。

   如果校验成功，则进入下一个单元格，重复上述步骤。

   如果校验失败，弹出提示对话框，并激活那个单元格的编辑状态。嘿嘿~这样就实现啦。

总结下来，主要是人们不知道如何确定某个单元格的数据和对应的编辑器，其实ds，colModel之间都是相互关联的，只要好好查查api，一切问题都迎刃而解。

1.x例子在lingo-sample/1.1.1/02-11.html。2.0例子在lingo-sample/2.0/02-11.html。

2.8.5. 限制类型，让用户只能选择我们提供的数据

为什么不直接用TextField呢？用户可以想添什么就添什么。但这也太自由了吧？你愿意让用户随便乱跑四处惹祸吗？可能最后他搞垮了系统，还要埋怨你的软件不好用。而更多的时候，客户希望操作更简单，更直观，而不是麻烦的自由，所以用多种方式输入数据是必然趋势。

从组件堆了随便挑几个出来，比如NumberField控制只能输入数字，ComboBox只能输入备选的几项，DateField是选择日期，Checkbox则是从true和false中择一而为。

这样一来，咱们之前的数据模型就不够用了，首先要让数据类型更丰富，这样才好展示咱们成果。

```
// 放到grid里显示的原始数据
var data = [
    [1.1,1,new Date(),true],
    [2.2,2,new Date(),false],
    [3.3,0,new Date(),true],
    [4.4,1,new Date(),false],
    [5.5,2,new Date(),true]
];
```

第一列是数字，第二列是comboBox对应的id，第三列是日期，第四列是布尔型对错啦。

好了，现在对应这五列设置各自的编辑器，第一列咱们限制只能用数字，不能是负数，而且数字不能超过10。

```
var cm = new Ext.grid.ColumnModel([{
    header:'数字列',
    dataIndex:'number',
    editor:new Ext.grid.GridEditor(new Ext.form.NumberField({
        allowBlank: false,
```

```
        allowNegative: false,
        maxValue: 10
    }))
},{
```

使用NumberField就只能输入数字啦，allowBlank:false不能是空白，allowNegative:false就不能输入那个减号，maxValue:10现在可输入的最大值。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2007-12-14	是
2.2	ext扩展	2007-12-14	否
3.3	新版ext教程	2007-12-14	是
4.4	ext在线支持	2007-12-14	否
5.5	ext扩展	2007-12-14	是

第二列稍微复杂一点点，毕竟是comboBox嘛。先准备下拉列表里的数据：

```
var comboData = [
    ['0','新版ext教程'],
    ['1','ext在线支持'],
    ['2','ext扩展']
];
```

数据部分单独抽离出来，是为了在editor和renderer里保持数据同步，配置的参数都在上边的comboBox部分介绍过。

```
},{
    header:'选择列',
    dataIndex:'combo',
    editor:new Ext.grid.GridEditor(new Ext.form.ComboBox({
        store: new Ext.data.SimpleStore({
            fields:['value','text'],
            data: comboData
        }),
        emptyText: '请选择',
        mode: 'local',
        triggerAction: 'all',
        valueField: 'value',
        displayField: 'text',
        readOnly:true
    })),
    renderer: function(value){
        return comboData[value][1];
    }
},{
```

renderer这个渲染函数需要研究研究，一段时间，经常看到有兄弟询问EditorGrid里的ComboBox总是无法正常显示数据，其实呢，就是少了这个renderer函数，毕竟EditorGrid里的编辑器只在实际编辑

的时候起作用，grid与editor之间共享的是数据，显示层都要依靠各自的实现。其实这样做不是更灵活么？没有必要两者都使用一样的显现方式呢。

第三列选择日期，本来是司空见惯了的DatePicker，也就不多说了。

```
},{
    header:'日期列',
    dataIndex:'date',
    editor:new Ext.grid.GridEditor(new Ext.form.DateField({
        format: 'Y-m-d',
        minValue: '2007-12-14',
        disabledDays: [0, 6],
        disabledDaysText: '只能选择工作日'
    })),
    renderer: function(value) {
        return value.format("Y-m-d");
    }
},{
```

比较常用的有format:'Y-m-d'，这样显示的日期会变成'年-月-日'的格式，minValue设置日期最小值，disabledDays是很有趣的一个选项，你可以通过它禁止用户选择星期几，咱们这里禁用了周六和周日，同时也改一下提示信息：“只能选择工作日”。

第四列中的checkbox就单纯了，只能选择true或false，所以也没什么好配置的。

```
},{
    header:'判断列',
    dataIndex:'check',
    editor:new Ext.grid.GridEditor(new Ext.form.Checkbox({
        allowBlank: false
    })),
    renderer:function(value) {
        return value ? '是' : '否';
    }
}]);
```

尽管没有特别的必要，我们还是使用renderer对显示做了美化。嘿嘿~爱美之心，人皆有之。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2007-12-14	☑
2.2	ext扩展	2007-12-14	否
3.3	新版ext教程	2007-12-14	是
4.4	ext在线支持	2007-12-14	否
5.5	ext扩展	2007-12-14	是

好了，例子在lingo-sample/1.1.1/02-12.html。2.0例子在lingos-sample/2.0/02-12.html。

2.9. 连坐法，关于选择模型

古代中国的刑法，一人犯法，数人处罚，其中的诛连九族是非常恐怖的刑法了。

别奇怪我提这些，表格里就存在这个现象，你的鼠标点击一个单元格，选中的却是一行，如果按下shift或ctrl键，更可以选择好几行。这不是就跟连坐差不多了吗？

grid里提供的这种功能叫做选择模型，你点击的是一个单元格，选中的却是整个行，在使用Ext.grid.Grid的时候，默认是使用RowSelectionModel，行选择模型。行选择模型默认是支持多选的，鼠标单击的时候按住shift或ctrl就可以选择多行。如果希望只选择一行，需要设置singleSelect参数。

```
var grid = new Ext.grid.Grid('grid', {
    ds: ds,
    cm: cm,
    sm: new Ext.grid.RowSelectionModel({singleSelect:true})
});
```

这样，即使再按着ctrl或者shift也不会选中好几行了，只会高亮显示最后选中的一行。1.x的例子在lingo-sample/1.1.1/02-13.html，2.0里依然可以这样用，lingo-sample/2.0/02-13.html。

与RowSelectionModel相对的是CellSelectionModel，它就没有连坐的效果了，每次只能选择当前的单元格，这情况就是谁犯法谁坐牢，在EditorGrid里默认使用这种选择模型，它没有其他的配置参数，每次只能选中一个单元格。

注意

下面说一些题外话，树形里也是有两种选择模型，默认的DefaultSelectionModel每次只能选择一个节点，另外还有一个MultiSelectionModel可以使用ctrl（注意，shift不行）选择多个节点，不过在firefox上试验的时候，每次按住ctrl再点击节点，都会打开一个新标签，最后也没解决，所以这个多选的实用性有待考证。

说了上边的还不够，有同志就问了，你这样选择上有什么用？我还是不知道怎么读取选择到的行呀。比如现在我想点击一下某某行，就弹出个对话框，里头是这些行的信息。咋整？

嗯，这就要考虑我们说的选择模型了，看下代码：

```
grid.on('click', function() {
    var selections = grid.getSelectionModel().getSelections();
    for (var i = 0; i < selections.length; i++) {
        var record = selections[i];
        Ext.Msg.alert('提示', record.get("id") + "," + record.get("name") + "," + record.get("descn"));
    }
});
```

先从grid里获得selectionModel，再从选择模型中获得目前选中的东东，如何判断现在选中了多少记录呢？selections.length就是你要的东东，现在你就可以循环读取这个selections了，每一个元素都是一个Record，所有的数据都在它里边了。

有的同志就说了，那我要是想判断有没有行选中怎么办啊？如果连一行都没选就不能这样操作啦。其实超简单，可能说出来你都不信，只要先判断selections.length是否等于0，就知道有没有选中了。哈哈~简单吧？

例子在lingo-sample/1.1.1/02-13a.html。2.0版本在lingo-sample/2.0/02-13a.html。

2.10. 2.0有，1.x里没有的那些可怕的控件

2.0里共享了一堆你自己都想像不到的表格控件，让你的应用更加丰富多彩，除去RowNumberer和CheckboxSelectionModel这种插件式扩展，还有那些自成体系的扩展，嘿嘿~浏览一遍啵。

2.10.1. 谓之曰PropertyGrid属性表格

属性表格扩展自EditorGridPanel，所以可以直接编辑右边的内容，注意，只有右边的，即使你点左边的单元格，编辑器也只会出现在右边。

实际上我们可以用哈希表来形容这个PropertyGrid，左边呢可以看作key，右边的叫value，key是由我们指定好的，然后用户只需要修改对应的value就好了，对于某些专属配置是不是很方便呀？

代码方便，因为只有两列，所以我们不用担心columnModel了，ds的部分确定是key和value的组合，因此也不用再分几步准备了。

```
var grid = new Ext.grid.PropertyGrid({
    title: '属性表格',
    autoHeight: true,
    width: 300,
    renderTo: 'grid',
    source: {
        "名字": "不告诉你",
        "创建时间": new Date(Date.parse('12/15/2007')),
        "是否有效": false,
        "版本号": .01,
        "描述": "嗯，估计没啥可说的"
    }
});
```

啦啦啦，title就是整个表格的标题啦。autoHeight很有用，这样就不需要咱们再去为div指定style咯。width是需要的宽度。renderTo应该很眼熟了，指定了渲染的元素id。剩下的就是source啦，这个json里指定了一套key和value，最后他们都会显示到表格里。

PropertyGrid提供的还不只这些，试着点击一下单元格，你会发现string, date, bool, number对应着默认的编辑器，嘿嘿~分别是TextField，DateField，ComboBox和NumberField，记得bool对应的ComboBox里只有true和false两项，其他的就都ok啦。

很可惜，例子只有2.0有，见lingo-sample/2.0/02-14.html。

2.10.1.1. 小插曲：只能看不能动的PropertyGrid。

本来PropertyGrid应该是可编辑的，可偏偏还是有人认为做成只读的会比较好。于是有人提出个方法来啦，注册beforeedit事件，然后在里边设置一个e.cancel = true;就可以不让它修改什么了。然后偶又尝试了一下return false，这个可是标准的取消某某操作的方法哟，结果也成功了，呵呵~如果你想让PropertyGrid只读的话，就试试这个法子呗。就像这样子：

```
grid.on("beforeedit", function(e){
    e.cancel = true;
    return false;
});
```

例子在lingo-sample/2.0/02-14a.html。

2.10.1.2. 小舞曲：强制对name列排序

看到name列上的排序箭头了吗？PropertyGrid对source中的第一列自动使用正序排列，这就造成了最

终的显示顺序很可能跟咱们输入的顺序不同。

其实像ext这么强壮的控件库，你肯定跟我一样觉得这里肯定有一个选项可以控制排序的。可是非常不幸，事实是这个排序是写死在代码里的，而且没有参数可以控制。没天理啊，想修改排序方式只能改代码。

```
Ext.grid.PropertyGrid.prototype.initComponent = function(){
    this.customEditors = this.customEditors || {};
    this.lastEditRow = null;
    var store = new Ext.grid.PropertyStore(this);
    this.propStore = store;
    var cm = new Ext.grid.PropertyColumnModel(this, store);
    // store.store.sort('name', 'ASC');
    this.addEvents(
        'beforepropertychange',
        'propertychange'
    );
    this.cm = cm;
    this.ds = store.store;
    Ext.grid.PropertyGrid.superclass.initComponent.call(this);

    this.selModel.on('beforecellselect', function(sm, rowIndex, colIndex){
        if(colIndex === 0){
            this.startEditing.defer(200, this, [rowIndex, 1]);
            return false;
        }
    }, this);
};
```

搞了这么多代码，结果只是为了注释掉其中的store.store.sort('name', 'ASC');让它不再对name正序排序。上帝啊，孰得孰失，望君自重。

代码在lingo-sample/2.0/02-14b.html。

2.10.1.3. 小夜曲：根据name获得value

PropertyGrid既然已经是一个哈希表的形式，人们自然会想到直接通过name获得value。可惜PropertyGrid连这样的功能都没给咱们提供，没法子，咱们只好去store里找了。

```
grid.store.getById('名字').get('value');
```

先获得id为“名字”的那一行，然后获得value列的值。唉，这是的，随便提供一个快捷函数多好啊。

例子在lingo-sample/2.0/02-14c.html。

2.10.1.4. 奏鸣曲：自定义编辑器

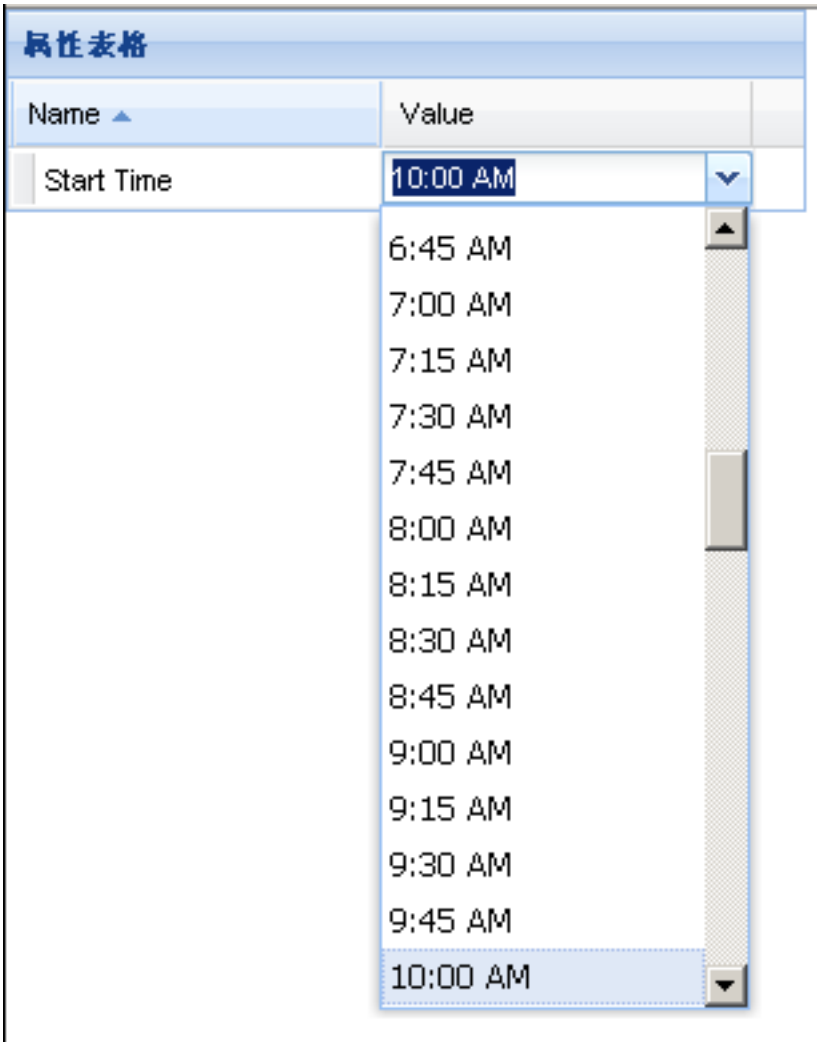
这可不是小打小闹，终于也有一个PropertyGrid提供给咱们的功能了，再去代码里乱掏咱们可撑不住了。

默认的TextField，DateField，ComboBox和NumberField也就只能对付一般情况，当咱们想对编辑器做

更详细的配置时，就需要这个了。PropertyGrid的customEditors让我们可以为指定id的那行数据设置对应的编辑器，咱们就按照api里的例子，做个TimeField看下吧。

```
var grid = new Ext.grid.PropertyGrid({
    title: '属性表格',
    autoHeight: true,
    width: 300,
    renderTo: 'grid',
    customEditors: {
        'Start Time': new Ext.grid.GridEditor(new Ext.form.TimeField({selectOnFocus:true}))
    },
    source: {
        'Start Time': '10:00 AM'
    }
});
```

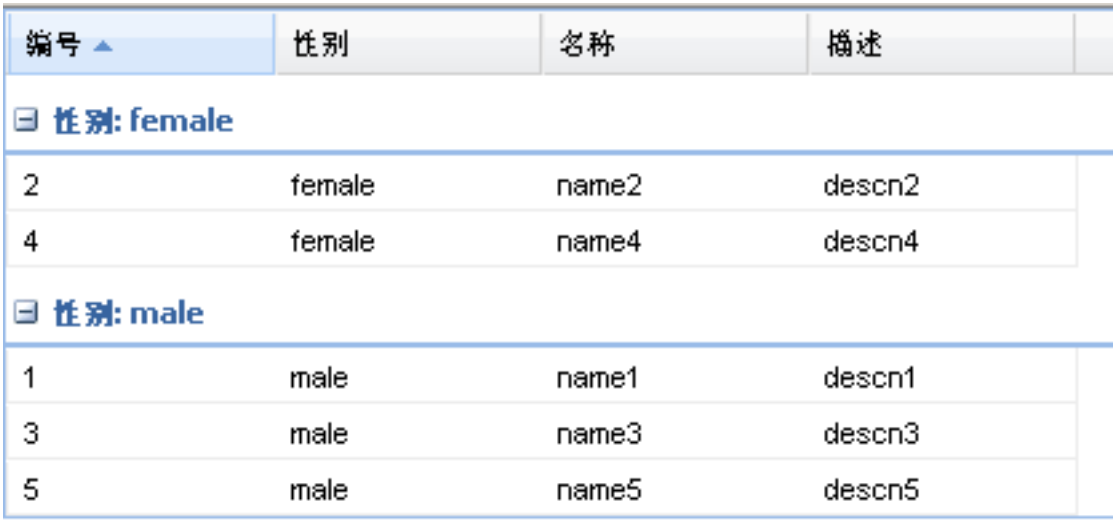
customEditors和source基本一样，前头的名字一样的两组数据会对应起来，后头的写法跟EditGrid里一样了，想要什么配置都写到里边好了。这样我们就获得了自制的编辑器，它会享受与默认编辑器一样的待遇了。



吼吼，如此一来我们可以获得对编辑器的完全控制权限咯，剩下的又只有自由发挥了。

例子在lingo-sample/2.0/02-14d.html。

2.10.2. 分组表格，嘻嘻，这是交叉报表吗？



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

是否回忆一下我们处理的数据：

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

这些数据将根据第二列，性别分组，然后还要按照id进行正序排列，通过Ext.data.GroupingStore对这些数据进行分检，然后提供给GridPanel。

```
var store = new Ext.data.GroupingStore({
    reader: reader,
    data: data,
    groupField: 'sex',
    sortInfo: {field: 'id', direction: "ASC"}
});
```

这里的reader和data还是原来那个，需要关注的是groupField，它用来告诉我们通过哪一项分组，令人困惑的是，GroupingStore要求必须设置sortInfo，这个参数对应的值里有两项，field是排序的属性，direction是排序的方式，ASC正序和DESC逆序。嘿嘿~咱们把数据输进去，流出来的就是分成一组一组的数据啦。不过要是想显示成我们期望的那种形式，我们还需要设置一个GroupingView。

```
var grid = new Ext.grid.GridPanel({
    store: store,
    columns: columns,
    view: new Ext.grid.GroupingView(),
    renderTo: 'grid'
});
```

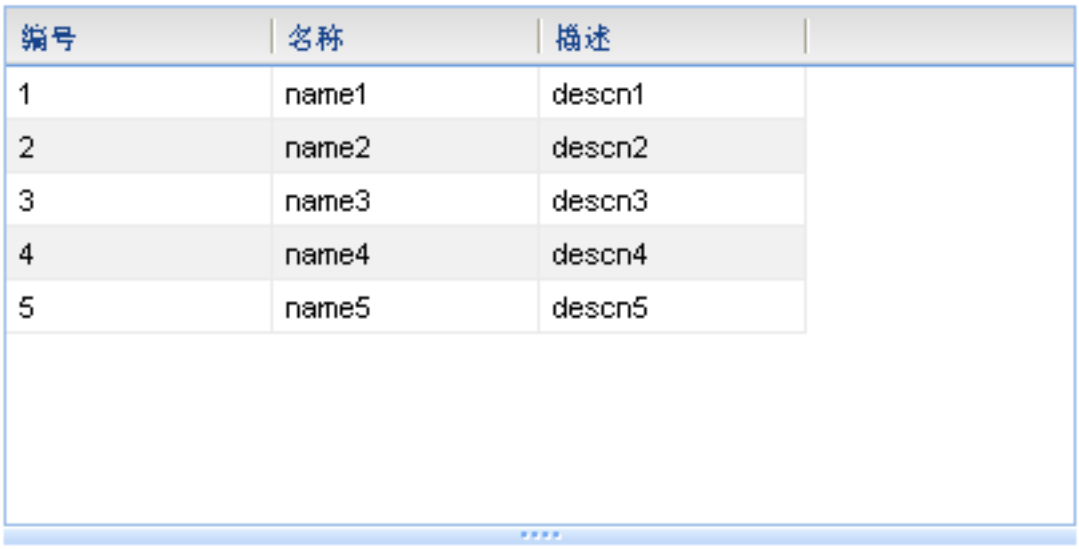
嘿嘿~这样就搞定啦，例子见lingo-sample/2.0/02-15.html。

另外还有GroupingSummary.js和RowExpand.js，不过这不包含在ext-all.js中。

2.11. 午夜怪谈，论可以改变大小，可以拖拽的表格

关于resizable，可能也要跟拖拽保持着一点点的关系，毕竟里边都有一个用鼠标点下，拖动的东西。

2.11.1. 先看看怎么拖拽改变表格的大小



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

可能到了grid下面那个蓝色的细条吗？把鼠标放到上边，就可以用拖拽改变表格的高度了。

想要这种效果不？唉，别担心，不用对你写好的grid动大手术，基本上是不会伤筋动骨的扩展一下，就赋予了表格这种神奇。

```
var rz = new Ext.Resizable('grid', {
    wrap:true,
    minHeight:100,
    pinned:true,
    handles: 's'
});
rz.on('resize', grid.autoSize, grid);
```

吼吼，根据ext例子中的注解，这一段要加在grid.render()之前才能获得更好的效果。其实呢，在1.x中resizable必须放在render之前，2.0中resizable必须放在render之后，否则就会出现问题。现在咱们来研究研究这些参数的构成。

1. 第一个参数是'grid'，就是说这个可改变大小的东东要在div id="grid"这个元素上起作用。怎么其作用呢？继续向下看。

2. wrap:true，这个参数会在构造Resizable的时候，自动在指定id的外边包裹一层div，这样就不用我们在html里定义其他附属的div了。方便方便。

3. minHeight:100，这个限制改变大小的最小高度。

4. pinned:true，spinned就是显示蓝色细线和上面的三个小点点，突出提示嘛。如果是true的话，这条线就一直显示，如果是false，只有鼠标放到它上面的时候才会出现，具体操作看你的喜好了。

5. handlers: 's'，大餐果然要放在最后一步，你要知道为什么只能拖动下边吗？不会认为只能修改下边吧。啊，非也非也呀。是这里的's'决定了用户只能修改下边。

   's'既'south'，南边的意思。为啥是南呢？地理课老师说过一句话："上北下南左西右东。"嘿嘿~明白了吧？ext用东南西北对应上下左右，用首字母来设置你可以拖拽哪几个方向，具体的配置像下边这样：

表 2.1. Resizable的handlers配置

值	英文
'n'	north
's'	south
'e'	east
'w'	west
'nw'	northwest
'sw'	southwest
'se'	southeast
'ne'	northeast
'all'	all

6. 最后别忘了注册事件，rz.on("resize", grid.autoSize, grid);，这样当拖拽完成之后，表格会调用autoGrid方法修改自己的大小，第三个参数是函数执行的scope，不用太深究，先记下来吧。

1.x的例子在lingo-sample/1.1.1/02-16.html。

2.0毕竟已经没有Ext.grid.Grid啦，对应的Ext.grid.GridPanel中又没有autoSize函数，这里就需要使用syncSize()函数啦，把最后一句改成这样就好。

```
rz.on('resize', grid.syncSize, grid);
```

2.0的例子在lingo-sample/2.0/02-16.html。

2.11.2. 再看怎么在表格里拖动行

这个，我还没试过CellSelectionModel，但拖动单元格除了用来炫耀之外，并没有太大作用。表格里的数据都是以行为单位的，所以这里就说说怎么把一行，或者几行拖来拖去的道理。

2.11.2.1. 无用功 之 同一个表格里拖拽

想来想去，在同一个表格里拖拽也就只能用在排序上，而且还必须是少量数据的情况下。不过说起来，因为ext的表格内置了对拖拽的支持，所以使用起来异常方便，我们只需要把enableDragDrop设置为true就行了。

```
var grid = new Ext.grid.Grid('grid', {
    ds: ds,
    cm: cm,
    enableDragDrop: true
});
```

如此一来，你就可以任意拖动了。嗯，就是有一个小问题，拖起来的行不能放下，总是有一个禁止drop的图标在哪里竖着。

唉，有些失落呢，grid虽然内置了拖拽开关，却没有默认实现的拖拽处理函数，我们必须添加自定义的处理函数，才能让拖起来的行确确实实放下去。

其实呢，放不下去的原因是grid里没有设置DropTarget，这个叫做扔东西目标如何？怎么好像是垃圾桶一样哦？拖拽的东东，都要从DragTarget拿起来，最后放到DropTarget上去。这个拖拽的基本应用，你看一下这里吧。第 7.3 节 “第二！代理proxy和目标target”。

那么我们的工作就是要设置一个DropTarget，找个地方把我们要放下的东西放下去。好了，这里都说是要做同一表格内的拖拽了，怎么能食言呢？当前grid就是我们要找的DropTarget。

对grid.container进行如下设置，让表格的容器成为DropTarget吧。

```
var ddrow = new Ext.dd.DropTarget(grid.container, {
    ddGroup : 'GridDD',
    copy    : false,
    notifyDrop : function(dd, e, data) {
        // 选中了多少行
        var rows = data.selections;
        // 拖动到第几行
        var index = dd.getDragData(e).rowIndex;
        if (typeof(index) == "undefined") {
```

```
            return;
        }
        // 修改ds
        for(i = 0; i < rows.length; i++) {
            var rowData = rows[i];
            if(!this.copy) ds.remove(rowData);
            ds.insert(index, rowData);
        }
    }
});
```

当当当，把grid.container做成DropTarget啦。你看到ddGroup了吗？这个分组名称必须跟DragZone里的分组一样才能有效果，grid里默认的ddGroup就叫做'GridDD'，把这两个弄成一样的就可以啦。

copy:false决定了我们拖拽以后是剪切效果还是复制效果。如果是copy:false，拖过去以后，原来的数据就应该不见了。如果是copy:true，原来的数据不会发生变化，这里我们当然选择false啦。

然后可要动真格的啦，notifyDrop对应的函数将处理拖拽事件，这个函数中主要分成三部分：

1. 第一，获得你到底选中的那几行。
2. 第二，根据拖拽事件获得拖拽目标的行索引，不过得到的index可能是undefined，这个时候我们就不处理啦。让它回归本位的好。
3. 第三，把你刚刚选中的那些东东，转移到拖拽的地方，如果copy:false的话，先删除，再添加，否则，不删除啊。

1.x的例子就在lingo-sample/1.1.1/02-17.html啦。2.0版本在lingo-sample/2.0/02-17.html。

2.11.2.2. 无间道 之 从这个表格拖到另一个表格

现在的问题是，如何超越表格自身的界限，我要把一个表格里的行搞到另一个表格里，咋办呢？

实际上我们先要有两个表格，当然都要设置上enableDragDrop:true啊，在这个基础上再配置上DropTarget，然后咱们就可以为所欲为啦。

```
var grid1 = new Ext.grid.Grid('grid1', {
    ds: ds1,
    cm: cm,
    enableDragDrop: true
});
var grid2 = new Ext.grid.Grid('grid2', {
    ds: ds2,
    cm: cm,
    enableDragDrop: true
});
```

这样我们有了两个表格，分别对应着各自的Ext.data.Store。好，下一步分别为它们两个创建各自的DropTarget。这里的修改仅仅限于最后对于转移数据的，下面是对应grid1的DropTarget，它会从ds2中删除数据，然后把这些数据添加到ds1里。

```

```

```
// 修改ds
for(i = 0; i < rows.length; i++) {
    var rowData = rows[i];
    if(!this.copy) ds2.remove(rowData);
    ds1.insert(index, rowData);
}
```

1.x的拖拽存在着bug，如果grid1里有6条记录，grid2里有4条记录的时候。把grid2的记录拖到grid1的最后一列时，handleMouseDown应该处理grid1的行，可它处理的却是grid2，当然就溢出了。这问题导致基本没法用。

1.x的例子在lingo-sample/1.1.1/02-18.html。

幸好2.0里是好的，对了，在2.0里不能用grid.container了，还是用grid.view.mainBody。

嘿嘿~1.x的例子是lingo-sample/2.0/02-18.html。

2.11.2.3. 无疆界 之 从表格里拖到树上

虽然很唐突，但树的情况请看下一章。

你觉得我们在做这步的时候应该特别注意哪些步骤呢？

1. 首先当然要有个grid，还要一个tree，先把这两个准备好我们才可以继续下一步呢。
2. 先整治grid，加上enableDragDrop:true让表格支持拖拽。

   使用grid.container构造DropTarget，让树节点可以扔到grid上。
3. 然后修理tree，使用enableDD:true启用拖拽功能。

   对应的事件是nodedragover和beforenodedrop。
4. 接下来要做的是设置ddGroup，如我们之前所说的，只有让grid和tree处于同一ddGroup才能成功拖拽。grid默认的ddGroup是GridDD，tree默认的ddGroup是TreeDD，我们只需要修改其中之一就可以。我在这里把tree的ddGroup改成了GridDD。
5. 最后一个要注意的是，在设置了这一切之后，要确保grid.render()在tree.render()之后才能让拖拽生效，否则树节点不能扔到表格上。

1.x例子在lingo-sample/1.1.1/02-19.html。2.0版本在lingo-sample/2.0/02-19.html。

嘿嘿~例子里没有实际拖过去放下的效果，因为树形的数据结构跟表格的很不相同，还要考虑拖拽父节点的情况呀，从表格拖拽多行到树形啊。这些就靠大家具体问题具体分析了。

2.12. 大步流星，后台排序

首先，ext不支持前台分页，一旦把后台的数据都传过来，不管pageSize是多少，都会一股脑的显示出来。一页上显示个几百条信息不但不好查找，ext的渲染速度也会让客户抓狂的。

其次，使用了分页，默认的排序只能管辖到当前读取数据，想对说有数据进行排序，还是需要把这些

信息搞到服务器那边，组装一个sql什么的。

基于上述两点，我们决定在这里对Ext.data.Store使用remoteSort啦。

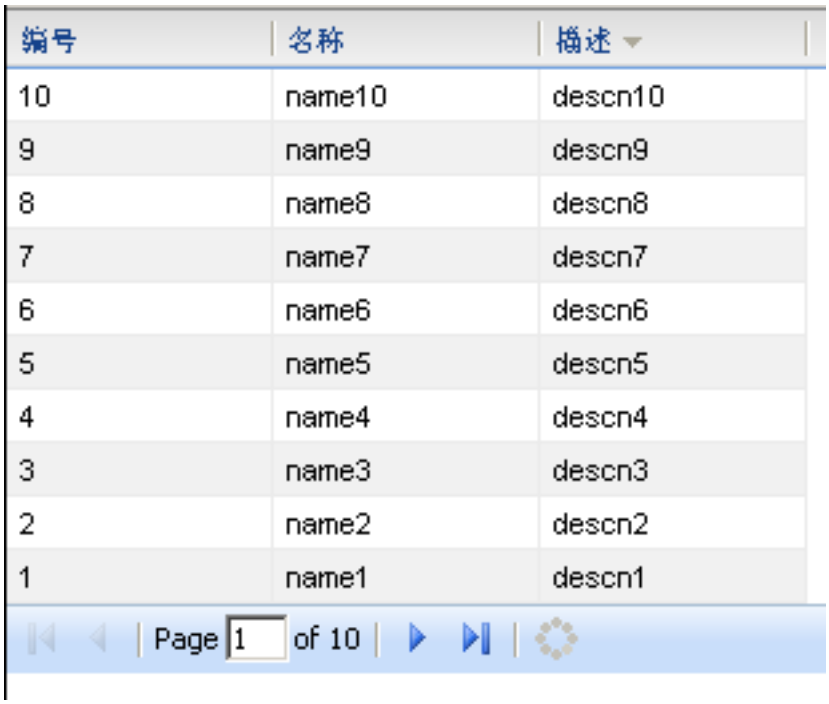
```
var ds = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'grid3.jsp'}),
    reader: new Ext.data.JsonReader({
        totalProperty: 'totalProperty',
        root: 'root'
    }, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ]),
    remoteSort: true
});
```

简简单单，只需要在ds里加一个remoteSort: true参数，下次排序的时候就有变化啦，不会自动计算已得的数据，而是多向后台发两个参数，sort表示需要排序的字段，dir表示ASC/DESC正序或者逆序。好了，后台可要接住这些数据，好做处理呢。看看咱们的grid3.jsp吧：

```
<%
String start = request.getParameter("start");
String limit = request.getParameter("limit");

String sort = request.getParameter("sort");
String dir = request.getParameter("dir");
System.out.println(dir);
if (dir == null) {
    dir = "ASC";
}

try {
    int index = Integer.parseInt(start);
    int pageSize = Integer.parseInt(limit);

    String json = "{totalProperty:100,root:[";

    if (dir.equals("ASC")) {
        for (int i = index; i < pageSize + index; i++) {
            json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
            if (i != pageSize + index - 1) {
                json += ",";
            }
        }
    } else {
        for (int i = pageSize + index; i > index; i--) {
            json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
            if (i != index - 1) {
                json += ",";
            }
        }
    }
    json += "]}";
    response.getWriter().write(json);
} catch(Exception ex) {
```

```
}
%>
```

使用request.getParameter()就可以的到那两个参数了，咱们这里没连数据，只对dir做了处理，如果是null或者是ASC就返回正序数据，如果是DESC就返回逆序。



编号	名称	描述
10	name10	descn10
9	name9	descn9
8	name8	descn8
7	name7	descn7
6	name6	descn6
5	name5	descn5
4	name4	descn4
3	name3	descn3
2	name2	descn2
1	name1	descn1

Page 1 of 10

jsp里根据排序的方式，让i++或者i--而已，要是你换做了sql可就更简单啦。直接用这两个字段拼sql啊。

```
String sql = "select * from t_user order by " + sort + " " + dir;
```

哈，其实这个根本就是为了sql而准备的呢。

1.x的代码在lingo-sample/1.1.1/02-20.html。2.0的代码在lingo-sample/2.0/02-20.html。

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 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哝出来的。

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

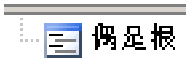
看到了吧，它自己都说它自己是根了，所以它就肯定是根没错。再看下面。

```
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。

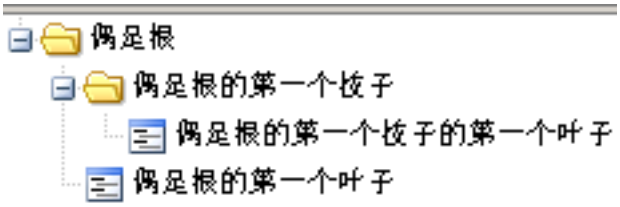
3.3. 超越一个根

上回书说道，我们要偷偷插上几个权子，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。

3.4. 你不会认为2.0里跟1.x是一样的吧？

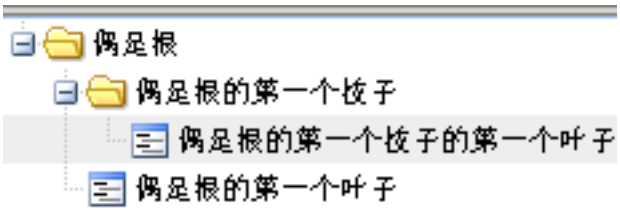
第一个区别就是TreePanel的定义，原来的id要放到{}中，对应的名字是el。像这样改：

```
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，却能表示枝权或者叶子，原理很简单，如果这个TreeNode下有其他节点，它就是一个枝权，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权了。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'})
});
```

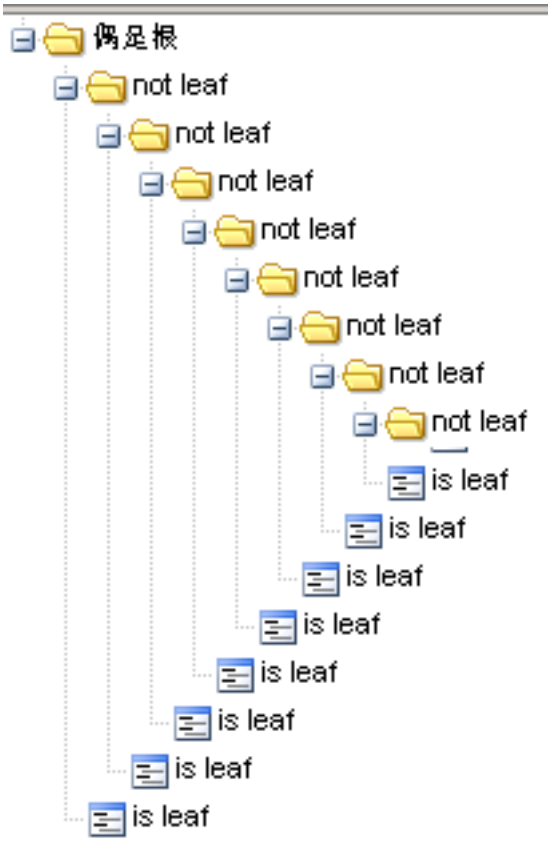
在此，TreeLoader仅包含一个参数dataUrl：’03-03.txt’，这个dataUrl表示，在渲染后要去哪里读取数据，为了方便模拟，我们写了一个txt文本文件提供数据，直接打开03-03.txt可以看到里边的内容。

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

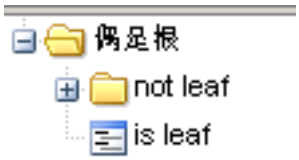
里边是一个包含了两个节点定义的数组，可能你会发觉那个多出来的属性leaf:true，它的效果很神奇，这一点我们马上就可以看到。

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。

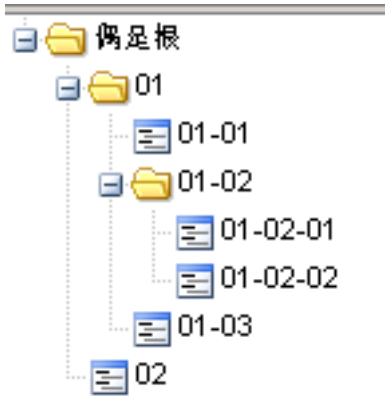
哈哈，现在就是关键部分了。因为咱们使用的03-03.txt提供的数据，不会判断当前节点的id，所以每次返回的数据都是一样的，这就是树会无限循环下去的原因了。

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}
]
```

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。

3.5.1. TreeLoader外传 之 JsonPlugin

嗯，接下来是TreeLoader外传，说些野史。

此事来源于struts2的JsonPlugin和springmvc的JsonView口述，如有失实，万望见谅。

事件的起因是这样的，TreeLoader只能处理json数组，就是中括号[]，JsonPlugin和JsonView只能返回json对象，就是大括号{}。于是矛盾就出现了。

一般来说就不会修改后台的，使用了JsonPlugin和JsonView就是为了方便，不用考虑pojo到json的转换了，这样一来我们就需要修改TreeLoader了。

下面给出一种修改的方法，依然使用TreeLoader，只是修改它其中的一个函数，使其可以支持对象，当然这个对象的值要是数组才行。

假设我们后台生成的json对象是这个样子：

```
{key:[
    {text:'01'},
    {text:'02',leaf:true}
]}
```

key对应的值就是树形需要的节点信息。

```
var loader = new Ext.tree.TreeLoader({dataUrl: '03-03a.txt'});
loader.processResponse = function(response, node, callback){
    var json = response.responseText;
    try {
        var json = eval("("+json+")");
        node.beginUpdate();
        // 从json中取得json数组
        var o = json["key"];

        for(var i = 0, len = o.length; i < len; i++){
            var n = this.createNode(o[i]);
            if(n){
                node.appendChild(n);
            }
        }
        node.endUpdate();
        if(typeof callback == "function"){
            callback(this, node);
        }
    }catch(e){
        this.handleFailure(response);
    }
};
```

实际上只添加了一行，var o = json["key"]，把key对应的值取出来，下面就是原封不动的老方法处理了，我可是一点儿都没动哦。

如果你只有一两处使用tree，那么这样改两下就够了，每次记得把key改成实际返回的值，如果还有更多地方需要这样转换，建议你还是扩展一个TreeLoader，以后就直接引用那个TreeLoader好了。

1.x的例子在lingo-sample/1.1.1/03-03a.html，2.0的例子在lingo-sample/2.0/03-03a.html。

3.5.2. TreeLoader外传 之 读取本地json

这算是一种使用TreeLoader的另类方法。

有的时候，咱们的树形数据不多，再去后台取实在不划算，可要是退回到一个节点一个节点生成的原始社会，又实在心有不甘，那么能不能让TreeLoader读取本地js中的json数据呢？

答案当然是yes啦。要不我在这边废这么多唾沫干啥？

首先为TreePanel设置一个参数为空的TreeLoader：

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

然后设置一个根节点，并为这个根节点设置上children属性：

```
var root = new Ext.tree.AsyncTreeNode({
    text:'偶是根',
    children: [
        {text: 'Leaf No. 1',leaf: true},
        {text: 'Leaf No. 2',leaf: true}
    ]
});
```

这里需要注意三点：

1. 第一。必须设置TreeLoader，否则根节点会一直处在读取状态，无法获得children中定义的子节点。
2. 第二。根节点必须是AsyncTreeNode，如果使用的是TreeNode，也无法生成子节点。
3. 第三。子节点中的叶子节点，必须都加上leaf: true，否则也会一直显示读取状态，让人看着很不爽。

好了，例子在lingo-sample/1.1.1/03-03b.html和lingo-sample-2.0/03-03b.html。

3.6. jsp的例子是一定要有的

因为，我要是不写jsp例子，你就不知道后台怎么判断目前要展开哪个节点。这一次我们又要请上tomcat大人了。

前台的代码就用之前的，把TreeLoader里的dataUrl改成咱们现在用的tree.jsp就好咯。

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

哦，对了，另外还要给root节点设置一个id，这样后台就知道什么时候是在读取根节点了。

```
var root = new Ext.tree.AsyncTreeNode({
    id: '0',
    text: '偶是根'
});
```

我们设置root的id:'0'，注意id不要出现重复，我们马上就要根据id来判断究竟是哪个节点正在展开，从而返回对应的数据。

前台准备就绪，现在可以看后边了。它就是传说中的tree.jsp的脚本了。

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    // 获得node参数，对应的是正在展开的节点id
    String node = request.getParameter("node");
    System.out.println(node);

    String json = "";
    if ("0".equals(node)) {
        json += "[{id:1,text:'节点阿一'},{id:2,text:'节点阿二'}]";
    } else if ("1".equals(node)) {
        json += "[{id:11,text:'节点阿一一'},{id:12,text:'节点阿一二'}]";
    } else if ("2".equals(node)) {
        json += "[{id:21,text:'节点阿二一'},{id:22,text:'节点阿二二'}]";
    } else if ("11".equals(node)) {
        json += "[{id:111,text:'节点阿一一一'},{id:112,text:'节点阿一一二'}]";
    }

    response.getWriter().print(json);
%>
```

先要强调一点，因为ajax默认使用utf-8编码格式，所以我们的jsp也要使用utf-8编码发送数据。

其实，树形异步读取的关键点是这个node参数，当一个节点展开的时候，会让TreeLoader根据设置的dataUrl发送请求去后台读取数据，而发送请求的时候TreeLoader会把这个节点的id作为参数一并发送到后台去，对于后台来说，只要获得node参数，就知道是哪个节点想展开了。

剩下的就简单了，我们根据节点的id返回对应的json数据。返回的时候，记得返回id和text就可以了，id就是来后台取数据的id，text就是节点名称，其他的倒是还有一些属性，不过咱们还没碰到。

嘿嘿，实际操作一下就可以看到了，展开节点会提示正在读取，然后刷拉刷拉的显示出下面的节点来。

这个时候就不能用root.expand(true);进行递归展开了，它会不断向后台发送请求，一直到把所有节点都展开为止，起不到异步的效果撒，注意注意。

现在呢，不知道你没有注意到一个让人纳闷的现象，比如，到了像下面这种情况，

如果咱们继续展开节点的话，会变成这样子。

有几个节点，不但没有展开，它们的图标还发生了变化，一个个都从枝干变成叶子了。这是为了什么啊？

像我们上面谈到的，AsyncTreeNode会根据后台返回的数据生成不同的节点，如果返回的子节点数据包含leaf:true属性，就会生成TreeNode节点，并标记成叶子节点。而，TreeNode不能再自动展开了。

而我们现在的情况是，这几个节点本来是AsyncTreeNode，但它们去后台读取数据时，后台明显没有返回任何子节点数据，这时候AsyncTreeNode立刻意识到自己犯了错误，它自己很可能已经是叶子了，当然下面不会再有任何数据，所以它急中生智，摇身一变把自己变成了叶子模样，不但隐藏了展开的小加号，甚至连自己的图标都变成叶子模样，从此以后，它就确确实实是一个叶子了。

属下很聪明，咱们做老大的当然高兴，不过这里边隐藏着的一个问题是，这个变化过程很可能让使用的客户摸不着头脑，而且去后台访问也是要时间的，即使局域网速度很快也要消耗服务器的带宽。不管怎么说，让节点判断自己是不是叶子实在是个很烂的方法。

那该如何是好呢？我们的解决方案还是配置上leaf:true，这样就没有这种变来变去的情况了。嘿嘿~万变不离其宗撒。

修改其实很简单，后台返回数据的时候加上判断，如果是叶子，就设置leaf:true，这样前台没问题咯。修改后的tree2.jsp就是这样。

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    // 获得node参数，对应的是正在展开的节点id
    String node = request.getParameter("node");
    System.out.println(node);

    String json = "";
    if ("0".equals(node)) {
        json += "[{id:1,text:'节点阿一'},{id:2,text:'节点阿二'}]";
    } else if ("1".equals(node)) {
        json += "[{id:11,text:'节点阿一一',leaf:false},{id:12,text:'节点阿一二',leaf:true}]";
    } else if ("2".equals(node)) {
        json += "[{id:21,text:'节点阿二一',leaf:true},{id:22,text:'节点阿二二',leaf:true}]";
    } else if ("11".equals(node)) {
        json += "[{id:111,text:'节点阿一一一',leaf:true},{id:112,text:'节点阿一一二',leaf:true}]";
    }

    response.getWriter().print(json);
%>
```

1.x的例子见lingo-sample/1.1.1/03-05.html。2.0的例子在lingo-sample/2.0/03-05.html，注意2.0与1.x的小小不同。

3.7. 让你知道树都可以做些什么

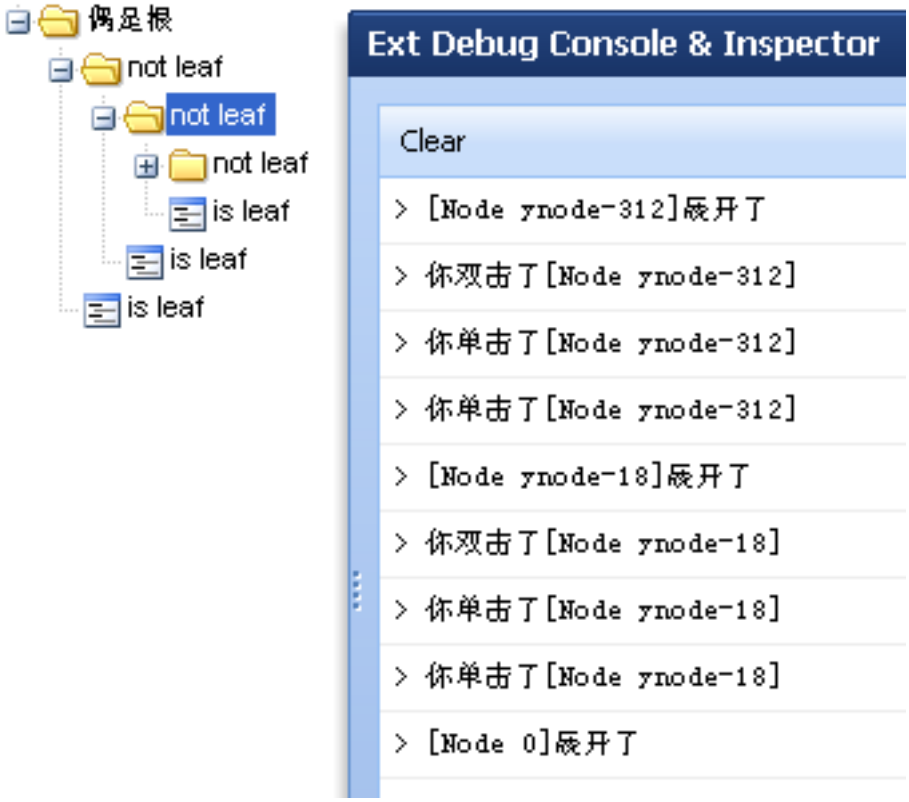
3.7.1. 检阅树形的事件

ext奇妙的一点就是，它的事件模型会告诉你很多你想知道的事情，比如哪个节点展开来，哪个节点折叠啦，谁点了一个树节点，谁双击了一个树节点什么的。而这一切借助ext的事件提醒系统都会变得很简单，只需要使用on注册时间监听器就可以了。比如像这样：

```
tree.on("expand", function(node){
    Ext.log(node + "展开了");
});
tree.on("collapse", function(node){
    Ext.log(node + "折叠了");
});
tree.on("click", function(node){
    Ext.log("你单击了" + node);
});
tree.on("dblclick", function(node){
    Ext.log("你双击了" + node);
});
```

```

```

而这样我们乱点一气的结果就是：



嘿嘿，这里像您隆重介绍一下图中的Ext debug console，这是只有引入了ext-all-debug.js才会出现的效果哟。因为使用alert()做测试的话，总是出不来双击的效果，所以只好借助它来给咱们显示效果了。我们为了使用Ext.log()方法，特意将以前的ext-all.js换成了ext-all-debug.js，这样就使得我们不在firefox+firebug的环境下也能清楚看到这些日志了。

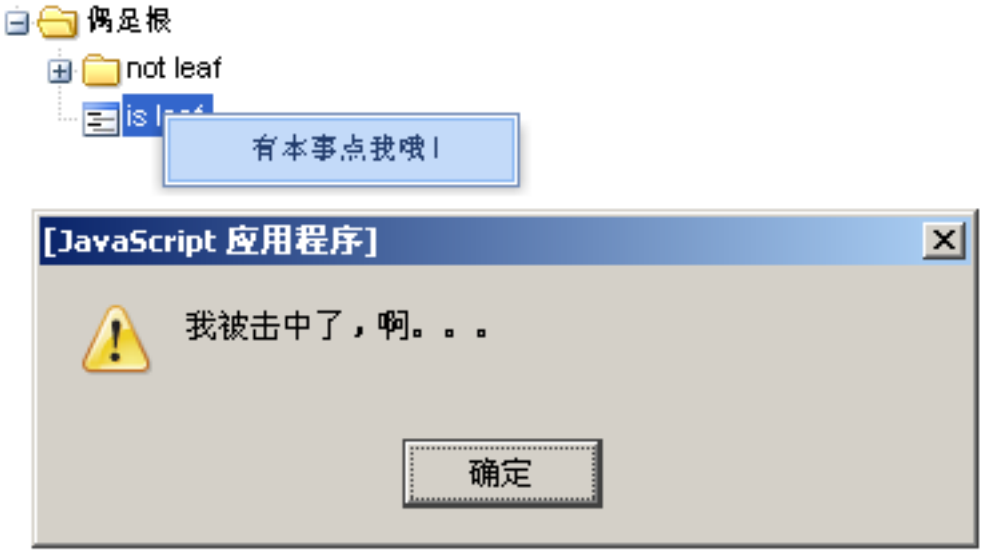
其实on("eventName")的用法非常简单哟，就是把一个函数绑定到一个事件上，你只要知道这个事件的名字就可以这么做了，到这个事件发生的时候，ext就回去找到所有绑定好的函数，然后一个一个执行。如此这边简单的原理，成就了我们绚丽的功能，你所需要做的就去api文档里查查你究竟需要处理哪些事件，然后使用on绑定一下就可以了呢。

2.0里的事件有了改变，原来的expand和collapse现在对应这个面板展开与折叠了，如果还想监听节点，需要换成对应的expandnode和collapsenode。

啊，大海啊，你是如此的伟大那么的纯粹，就好像我们的事件模型一样，让我们可以在lingo-sample/1.1.1/03-06.html和lingo-sample/2.0/03-06.html看到对应的例子吧。

3.7.2. 右键菜单并非单纯的事件

浏览器都有自己的右键功能菜单，咋才能弹出自定义的右键菜单呢？就像下面这样？



这个过程比较复杂，不过也是依托了ext的事件模型。我们要注册一个名叫contextmenu的事件，在这个事件发生的时候弹出咱们自己制造的菜单，然后显示出来，这样一切就都在咱们的掌控下了。

但是具体应该咋整呢？我们下来就一步一步的演示给你看，别眨眼哟。

1. 第一步，制作自定义的菜单

```
var contextmenu = new Ext.menu.Menu({
    id: 'theContextMenu',
    items: [{
        text: '有本事点我哦！',
        handler: function(){
            alert('我被击中了，啊。。。');
        }
    }]
});
```

像这样创建一个菜单，现在菜单里仅有一项，名字啊，单击时的处理函数都设置好了。这些都是为了即将的右键事件准备的。

2. 第二步，绑定contextmenu事件

```
tree.on("contextmenu", function(node, e){
    e.preventDefault();
    node.select();
    contextmenu.showAt(e.getXY());
});
```

这个事件会像绑定的函数传递两个参数，当前点中的节点和事件对象，实际上其他事件一般都是它们俩，不过我们都只用了第一个。

首先调用的是e.preventDefault()，这个函数的作用是防止浏览器弹出它默认的功能菜单，想想也知道，一下子跑出两个菜单来，用户一定晕了，这到底是该用哪个哟？

然后我们调用node.select()，选中当前节点，因为右键单击事件虽然发生了，当前节点可能没有被选中，咱们让它变成选中状态，可以避免以后可能出现的问题。

showAt(e.getXY())，这个方法比较爽，是通过e获得当前鼠标的坐标，contextmenu就显示到这个坐标，看起来就像是从节点上弹出来的。酷啊。

3. 到这里，本来菜单的监听已经完成了，你在菜单外点击一下鼠标，菜单应该会默认隐藏起来。但也存在着意外情况，菜单没有隐藏起来你又要进行其他操作，比如展开树。无法消失的菜单实在是太碍眼了，这个时候我们就需要调用contextmenu.hide()让它消失。呵呵~到时候把这句加在展开，折叠操作之前就可以了，以防后患嘛。

例子在lingo-sample/1.1.1/03-07.html和lingo-sample/2.0/03-07.html，嗯，没什么不同的地方。

3.7.3. 默认图标好单调，改一下撒

别告诉你从来没有想过要改改树里那些图标哦，我没改是因为我懒，不是我不想呢。

唉，咋说呢，主要还是找图标太麻烦了，好啦好啦，别闹，我告诉你方法，自己去试试吧。

其实呢，树节点就有两个属性，谓之曰：icon和iconCls，要改图标就靠这哥俩儿了。

嗯，咱们拿根节点开到，其他的节点不管是手工设置的，还是json取到的都是大同小异，你要融会贯通撒。

1. 如果你用icon的话：

```
var root = new Ext.tree.TreeNode({
    text: 'icon',
    icon: 'user_female.png'
});
```

2. 如果你用iconCls的话：

```
var node1 = new Ext.tree.TreeNode({
    text: 'iconCls',
    iconCls: 'icon-male'
});
```

另外还要加一个css的定义：

```
.x-tree-node-leaf .icon-male {
    background-image: url(user_male.png)
}
```

这里有一点点讲究，就是iconCls对应的是icon-male，但写在css里要用层叠的写法，定义.x-tree-node-leaf下的.icon-male，否则不会起作用。

3. 如果你两个都用的话：

```
var node1 = new Ext.tree.TreeNode({
    text: 'iconCls',
    iconCls: 'icon-male'
});
```

结果就是这样，两种方法都可以达到我们的要求，不过比较一下两个都用的情况，到底哪个优先级更高呢。

嘿嘿~结果就是icon获胜，其实超级好理解呢。iconCls只能定义背景图片，icon设置的是img的src部分，当然是icon会把背景部分挡住啦。不过实际用的时候咱们还是选一个用吧，两个都用太乱了。

这部分例子内容也差不多啦，见lingo-sample/1.1.1/03-08.html和lingo-sample/2.0/03-08.html。

3.7.4. 偷偷告诉你咋从节点弹出对话框

俺们这里追求滴是完美，如果你想知道怎么让一个对话框从刚刚点击的树节点飞出来，而不是生硬的凭空蹦出来，就往下看。

实际上，事件里获得的node只是一个对象，而不是html中一个实际的dom元素，所以你不能直接用animEl:node来实现飞出效果。

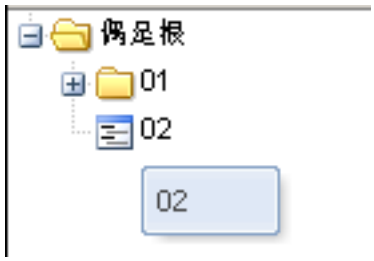
ext里的树节点都是遵从mvc设计，所以呢，要找对应的dom元素，就应该用它view的部分，在TreeNode里，这部分叫做ui。而ui里又包括好几部分，缩进用的空白呀，那些连接线呀，结点线的图标，还有就是显示的标题。

我们决定让对话框看起来像是从标题部分飞出来的，就像这样。

```
tree.on("click", function(node){
    Ext.Msg.show({
        title: '提示',
        msg: "你单击了" + node,
        animEl: node.ui.textNode
    });
});
```

动画不好截图，直接看例子吧。lingo-sample/1.1.1/03-09.html和lingo-sample/2.0/03-09.html。

3.7.5. 小小提示



嗯，就是鼠标在某某某上空，悬停几秒以后，就自动出来的小东东，虽然简单却惹人喜欢。如何？研究一下它是怎么出现的吗？

实际上我们要对提供的json数据改一改，加上提示内容，正如提示的英文所讲，quick tip，我们给每个节点数据都加上qtip:'xxx'，这样树就知道该显示什么提示信息了，03-10.txt样例数据如下：

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

光这样还不够，你还需要在js最前面加一条，对提示功能进行初始化。

```
// 开启提示功能
Ext.QuickTips.init();
```

嘿嘿~，这样你就可以看到咱们新添的提示了。要提示什么东西呢？你自己决定好了。

例子在lingo-sample/1.1.1/03-10.html和lingo-sample/2.0/03-10.html。

3.7.6. 给树节点设置超链接

虽然我们完全可以监听click事件，不过直接设置上超链接的地址也不失为一个好主意，很多人都想实现的功能哦。

现在让我们给节点添上href属性，就像这样：

```
{text:'01-01',qtip:'01-01',leaf:true,href:'03-10a.html'}
```

好了现在你点一下01-01节点，就会跳出03-10a.html页面了。呵呵~简单呢。不过问题又来了，这样设置的链接会改变当前页面，怎么才能另开一个窗口呢？你可能立刻就想到了用target，不过很可惜，tree里target这个名字似乎另有他用，所以我们要使用其他参数名称了，这里我们需要配置hrefTarget：

```
{text:'02',qtip:'02',leaf:true,href:'03-10a.html',hrefTarget:'_blank'}
```

哈哈，这样的href和hrefTarget的组合确保我们可以在正确的地方打开正确的网页，想要什么？在json里生成就好了。

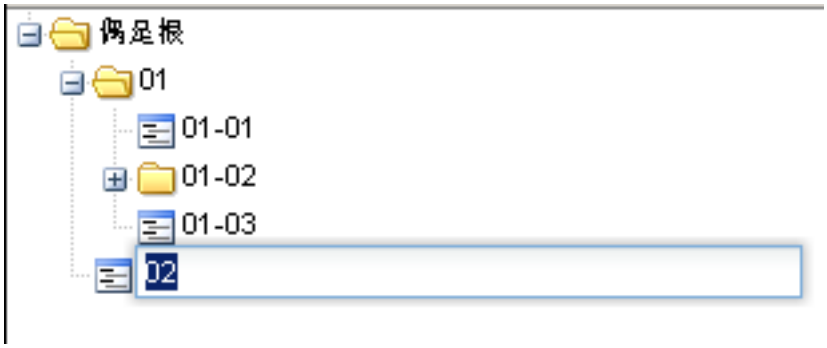
例子在lingo-sample/1.1.1/03-10a.html和lingo-sample/2.0/03-10a.html。

3.8. 灰壳显灵！让我直接修改树节点的名称吧！

在半年前，我的意思是2007年6月份以前，如果想修改一个节点的标题，我必须先选择这个节点，点一下距离很远的修改按钮，然后页面跳转到一个修改页面，在这里，我们修改对应的数据，然后进行提交，显示出一个操作成功的页面后，再次回到显示树形页面，这才能看到修改后的结果。

我累不累啊？！

其实我就为想改改那个标题呀，就像这样。



在标题上点一下鼠标，就可以修改，修改完了，一个回车就完成了修改过程。不知道你怎么想，自从第一次看到这功能，我就爱死它了。嘿嘿~

代码可是超简单，不用任何侵入性修改，只需要创建一个TreeEditor，再把TreePanel放进去，于是，万事休矣。

```
var treeEditor = new Ext.tree.TreeEditor(tree, {
    allowBlank    : false
});
```

稍微说一下参数，第一个参数tree就是我刚刚说的TreePanel。然后allowBlank属于数据校验部分，意思是必须输入点儿什么东西，你想想啊，一个没有名字的节点成什么样子哟。

好了，好了，这就是我们需要的所有东西呢，有了它们我们就可以自由修改节点了。

例子在lingo-sample/1.1.1/03-11.html和lingo-sample/2.0/03-11.html。

百尺竿头，更进一步。通过绑定事件可以让我们获得对TreeEditor实现更多控制，想知道这些事件是什么吗？

1. on("beforestartedit")，这个事件给你控制是否允许编辑当前节点的权利，我们用自己的方式判断

node的属性，如果满足某个条件，比如leaf:false，就不让你编辑。操作很简单，返回false就好了。

```
treeEditor.on("beforestartedit", function(treeEditor){
    return treeEditor.editNode.isLeaf();
});
```

在枝干节点和叶子节点上乱点几下，就看到效果了。

2. on("complete")，当完成修改，点一下回车，就会发生这个事件，告诉我们修改完成啦。然后我们就可以得到修改后的数据，保存啦，发送给谁啦，随意做任何事。

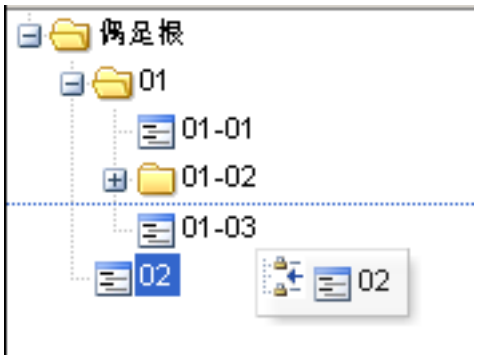
```
treeEditor.on("complete", function(treeEditor){
    alert(treeEditor.editNode.text);
});
```

记住，node.text就是我们修改后的结果啦。

现在我们知道如何处置TreeEditor了，例子在lingo-sample/1.1.1/03-12.html和lingo-sample/2.0/03-12.html里，试试呗。

3.9. 我拖，我拖，我拖拖拖。

想让node可以自由拖动的话，创建TreePanel的时候设置个enableDD:true就可以了。



例子就看看lingo-sample/1.1.1/03-13.html，lingo-sample/2.0/03-13.html。

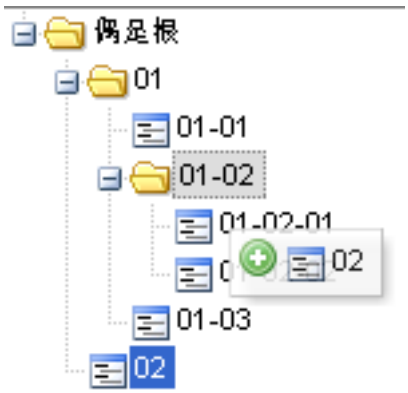
那个，朋友回来，别拖了，咱们还有东西要说呢，光设置成拖拽有啥用？我们还要拿其他东西配合着，才能实现需要的功能哦。

3.9.1. 树形节点的拖拽有三种形式

当然，拖（drag）就是把一个节点拽起来，你拽哪个节点都是一个样子。不过放（drop）就有学问了，我想说说ext里的三种形式，你听听有没有道理。

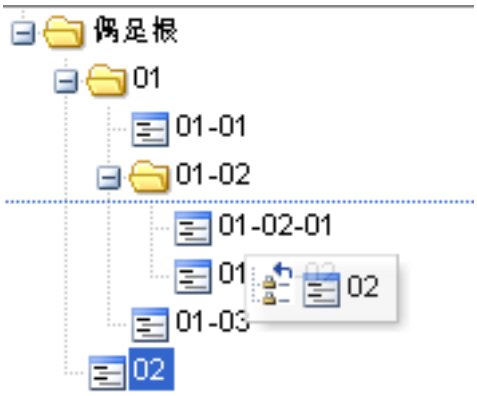
1. append，扔下去的节点就变成被砸中的节点的子节点了，最后就是父子关系啦。

append的标记是一个绿色的圈圈，里头带个加号。



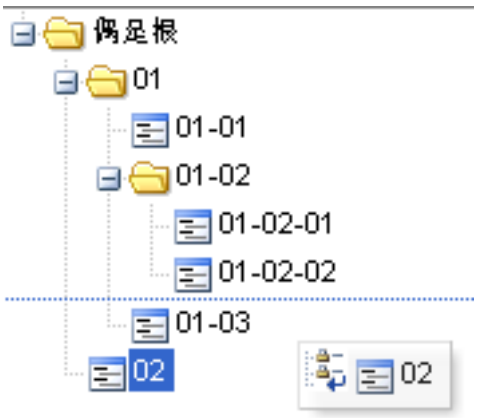
2. above，扔下去的节点跟被砸中的节点是兄弟关系，扔下去的节点排行在前。

above的标记是上头一个箭头，下头两个杠杠。



3. below，扔下去的节点跟被砸中的节点还是兄弟关系，不过这次扔下去的节点排行在后。

below的标记是上头一个箭头，下头两个杠杠。



3.9.2. 用事件控制拖拽

TreePanel里有N个事件跟拖拽有关，通过它们我们可以把拖拽从头顶控制到脚底下，这里就弄出几个偶常用的瞧瞧，先。

3.9.2.1. 叶子不能append

拖拖的时候有没有发现，叶子节点不能append啊？这是ext内部规定的，如果leaf:true，就不能用拖拽的方式添加子节点了。这个规定明显不合理，简直是限制人权哟，但修改源代码实在太可怕，我们还是用事件搞定吧。

我们需要的是nodedragover，当你拉着一个结点，划过某某某的头顶，就会发生这个事件。当这个某某某是叶子的时候，我们就进行一下操作，破坏掉ext的规矩。

```
// 拖拽判断
tree.on("nodedragover", function(e){
    var n = e.target;
    if (n.leaf) {
        n.leaf = false;
    }
    return true;
});
```

事件发生以后，会给绑定的函数发送一个参数e，e.target是目前鼠标划过的节点，我们先判断n.leaf，如果为真就说明这个节点是叶子节点，对它的操作就是让leaf属性变成false。这就可以给它添加子节点了。

例子在lingo-sample/1.1.1/03-14.html和lingo-sample/2.0/03-14.html。

3.9.2.2. 把节点扔到哪里啦

nodedrop事件告诉我们，手里拿着这个节点掉下去了。

```
tree.on("nodedrop", function(e){
    Ext.Msg.alert('提示', '咱们的节点' + e.dropNode + '掉到了' + e.target + '上，掉落方式是' + e.point);
});
```

这里充分利用了事件发生时传递过来的参数，e.dropNode是本来抓在手上的节点，e.target是掉下去砸到的节点，e.point是砸下去的方式：append,above,below。

通过这些数据，我们就可以知道当前节点的位置和状态，计算出数据来通过ajax发送给后台，让后台对节点的数据做更新。ok，让我们来实验一下，顺便看一下ext里怎么用ajax传输数据。

在原来的基础上添加ajax的部分。

```
tree.on("nodedrop", function(e){
    Ext.Msg.alert('提示', '咱们的节点' + e.dropNode + '掉到了' + e.target + '上，掉落方式是' + e.point);
    var item = {
        dropNode: e.dropNode.id,
        target: e.target.id,
        point: e.point
    };
    Ext.lib.Ajax.request(
        'POST',
        'tree3.jsp',
        {success: function(response){
            Ext.Msg.alert('信息', response.responseText);
        },failure: function(){
```

```
            Ext.Msg.alert("错误", "与后台联系的时候出现了问题");
        }},
        'data=' + encodeURIComponent(Ext.encode(item))
    );
});
```

这里的Ext.lib.Ajax实际上并非ext的一部分，怎么这样说呢？早前咱们也提过了，ext是基于其他js组件库建立起来的，很多底层功能都是利用其他js组件库的功能，ajax就是其中之一。不信你可以去找找，Ext.lib.Ajax的定义是写在ext-base.js里，而不是ext-all.js里。如果你使用了其他的adapter，Ext.lib.Ajax就定义到其他的适配器里了。对我们来说，只有这个Ext.lib.Ajax的名字不变，具体的内部实现都是你选择的适配器决定的。

如果你对底层没有太大兴趣的话，咱们只需要会用这个Ext.lib.Ajax就可以了，这里，咱们就只介绍它的用法好了，嘿嘿~其实是因为我也不知道它内部是怎么实现滴，我也只会用它而已。

咱们在这里调用了request()函数，函数包含四个参数，注意哟，其中success,failure的一大串是一个参数。

1. 第一个参数，'POST'是使用http的METHOD，如果你对http不了解的话，请闭上眼选择POST。可以避免一些中文参数乱码和缓存的问题。

2. 第二个参数，'tree3.jsp'请求的url。

3. 第三个参数，被称作回调函数的东东。之所以叫回调函数，是因为这里定义的函数会在ajax执行后被调用。{success:fn,failure:fn}，这个json里包含两个值，success对应的函数会在请求成功后调用，failure对应的函数会在请求失败后调用。

   呼，缓一口气。其实，这里的成功和失败跟你脑子里想的“数据库访问失败成功”根本是两码事，所以被指望用这个来分辨操作是否成功。ajax的成功失败在于http响应的状态码是否等于200。

   哦，又忘记你不懂http了，那就这么说吧，你直接通过浏览器访问页面的时候，如果没有返回一大堆404找不到页面，500服务器内部错误什么的，就是成功啦。ajax只能分辨这些错误，至于你的操作是否成功，自己回去判断去。

   咱们例子里的failure很简单，就是弹出错误提示。success稍微复杂一些，主要在于它有一个参数response。这个response就是响应对象，里边有响应状态，响应的内容等等一系列内容，ext都帮咱们封装好了。比如我们这里response.responseText获得的就是文本形式的返回内容，如果要xml格式的，就用response.responseXML，自己去试哟。

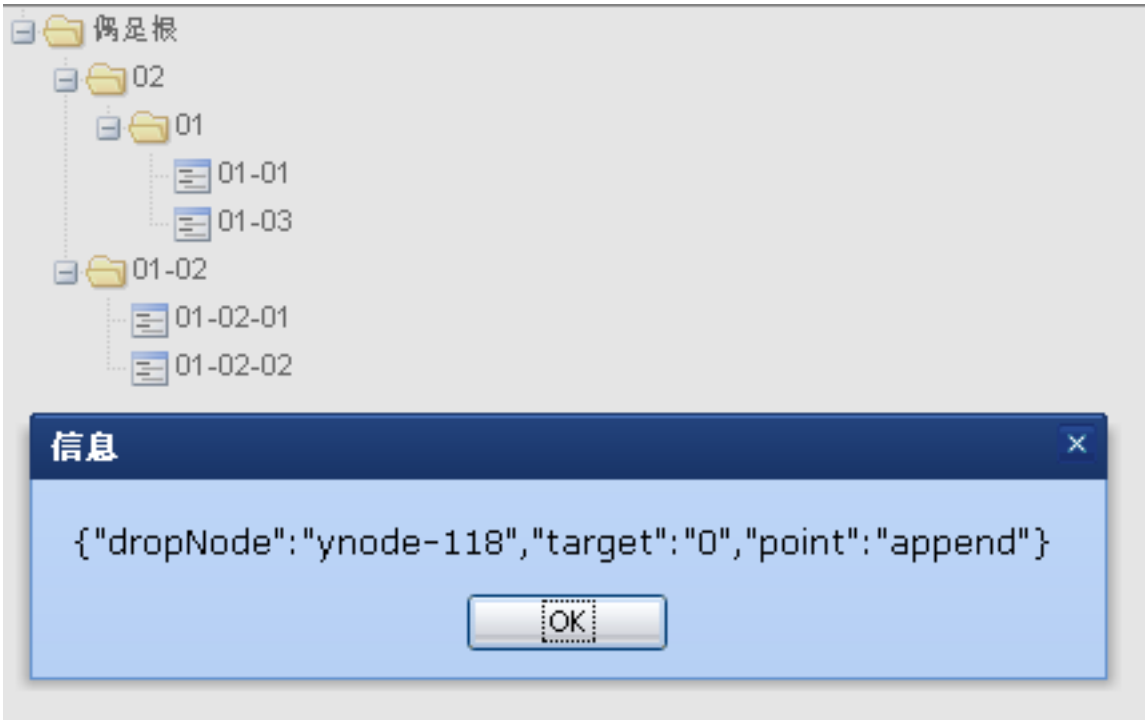
   第四个参数，是发送给后台的参数，参数格式name1=value1&name2=value2，至于你问为什么用encodeURIComponent(Ext.encode(item))这么长一串，那么我悄悄告诉你，Ext.encode()是把json转换成字符串，另外一个的作用是对参数编码，省得出现非法字符。通用方式啦，先记住。

名为tree3.jsp的后台脚本非常之简单，就是获得了参数，然后直接返回给前台。实际环境中你需要先对字符串进行解析，获得自己需要的数据进行处理，咱们只是演示，就不废话了。

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    String data = request.getParameter("data");
```

```
    System.out.println(data);

    response.getWriter().print(data);
%>
```

最后的结果就是这样，每次拖拽都会向后台发送数据，然后前台显示返回的响应。简单。简单。



例子就在lingo-sample/1.1.1/03-14.html和lingo-sample/2.0/03-14.html。

3.9.2.3. 裟椤双树，常与无常

一枯一荣。一棵树上可以兼具拖拽的特质，将这种特性扩展开，也可以在两棵树之间实现拖拽。但drag和并不一定同时存在，我们既可以使用enableDD:true，也可以单独使用enableDrag或enableDrop分别指定。如此一来可以限制用户只能从一棵树上拖到另一棵树上，如果你也想要这种效果，就往下看。

```
var tree1 = new Ext.tree.TreePanel('tree1', {
    enableDrag: true,
    loader: new Ext.tree.TreeLoader({dataUrl: '03-10.txt'})
});

var tree2 = new Ext.tree.TreePanel('tree2', {
    enableDrop: true,
    loader: new Ext.tree.TreeLoader({dataUrl: '03-10.txt'})
});
```

像这样设置，在tree1里使用enableDrag，tree2里使用enableDrop，用户就只能将tree1里的节点拖到tree2上，禁止了自身拖拽，方向相反也不能实现。

例子可以在lingo-sample/1.1.1/03-15.html和lingo-sample/2.0/03-15.html找到。

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combox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在任何地方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。

1. 像TextField，TextArea，NumberField这类当然是可以随便输入的了。
2. 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。
3. Checkbox和Radio，ext没有过多封装，基本上还是原来的方式。

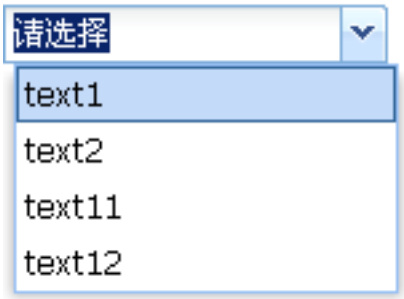
4. Button，这个东东其实就是一个好看的div，跟comboBox一样，不是对原有组件的美化，而是重新做的轮子。你可以选择用以前那种难看的type="button"，还是用咱们漂亮的div，看你的爱好了。type="submit"和type="reset"也一样没有对应的组件，都使用Button好了。

5. 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。

```
var data = [
    ['value1','text1'],
    ['value2','text2'],
    ['value11','text11'],
    ['value12','text12']
];

var store = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data : data
});


var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'query',
    valueField: 'value',
    displayField: 'text'
```

```
});
combo.applyTo('combo');
```

首先要定义comboBox将要显示的数据，我们这里使用的是二维数组data。

其次将这个二维数组放到Ext.data.SimpleStore，和Ext.data.Store差不多的意思，不过这里省得定义proxy和reader，更simple撒。

得到了数据就可以创建ComboBox了。

最后调用applyTo('combo')，对ComboBox进行渲染，需要注意的是id="combo"对应的必须是<input id="combo" type="text" />，简单定义一个div是会报错的。

回头看看它里边的参数：

1. store，用来提供数据滴，原始数据是一个二维数组，转换成SimpleStore以后呢，咱们把第一列命名为value，第二行命名为text，然后就这么一股脑得放进ComboBox里。

   再次重申，第一行是value，第二行是text，第一行是value，第二行是text。

2. 啦啦啦，你可以看到另外两个参数了，valueField和displayField。

   为啥它们的值跟store里定义那两个名字一样呢？难道是他们之间有所关联？

   哇，你真是举一反三，太聪明啦，简直是大葱，ComboBox正是根据他们之间的对应来显示数据的。

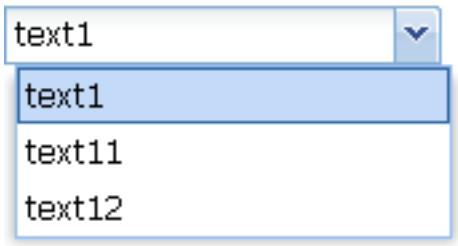
   这样你点一下ComboBox，弹出的列表里显示的就是text对应的数据，而当你选中某个数据时，comboBox的值就被自动设置成对应的value值。如此数据就都搞定喵。

3. emptyText超好理解，就是没选择任何数据的时候，comboBox里显示的提示信息，按你的心情，完全可以写成：“靠！你快选啊。”嘿嘿。

4. mode设置成local，comboBox就知道这些数据已经读取到本地了。

5. triggerAction设置成all，如果用默认的query，它会给你玩autocomplete。

   哦，你说不知道啥叫自动完成？就是像下面这样啦，每次帮你智能匹配结果咯。



   可这样一来就看不到所有结果啦，发没发现text2被吃掉了？快换成all吧。

例子在lingo-sample/1.1.1/04-02.html。

2.0里需要做小小的修改，看构造comboBox的部分：

```
var combo = new Ext.form.ComboBox({
```

```
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'query',
    valueField: 'value',
    displayField: 'text',
    applyTo: 'combo'
});
```

实际上只是把原来的applyTo()函数变成一个参数，让配置更紧凑哦，好哦。

例子见：lingo-sample/2.0/04-02.html。

4.4.2. 把select变成comboBox。

这要求html里必须先有一个select下拉框才行，像这样的：

```
<select id="combo">
    <option value="value1">text1</option>
    <option value="value2">text2</option>
    <option value="value11">text11</option>
    <option value="value12">text12</option>
</select>
```

然后我们在它的基础上构造ComboBox，数据就使用select里包含的数据。

```
var combo = new Ext.form.ComboBox({
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    transform: 'combo'
});
```

明显代码减少了很多，这都是因为transform:'combo'，它告诉ext去把那个select大卸八块，一条一条的把数据抽出来，添到ComboBox里边，最后还把整个select杀掉，换成comboBox，太残忍啦。

例子在lingo-sample/1.1.1/04-03.html。很高兴2.0这部分api没有任何修改，2.0的例子在lingo-sample/2.0/04-03.html。

4.4.3. 破例研究下comboBox的内在本质哟

首先请允许我用一下firebug看一下生成后的html样子，这样让我们更容易明白comboBox绚丽外表下隐藏的是些什么的东西。

```
<div id="ext-gen7" class="x-form-field-wrap" style="width: 144px;">
    <input id="combo" class="x-form-text x-form-field
    x-form-empty-field" type="text" autocomplete="off"/>
    <img id="ext-gen8" class="x-form-trigger
    x-form-arrow-trigger" src="../../resources/images/default/s.gif"/>
</div>
<div id="ext-gen13" class="x-layer x-combo-list" style="position: absolute; z-index:
11000; visibility: hidden; left: -10000px; top: -10000px; width: 142px;">
    <div id="ext-gen15" class="x-combo-list-inner" style="width: 142px;">
        <div class="x-combo-list-item">text1 </div>
        <div class="x-combo-list-item">text2 </div>
        <div class="x-combo-list-item">text11 </div>
        <div class="x-combo-list-item">text12 </div>
    </div>
</div>
```

哦，原来就是用div包裹了一个input和img啊。点击显示出来的列表也是用div做出来的，平常隐藏起来，按下选择再跑到input的下面显示一下，原来，智慧就是这么简单。这么简单。

实际上，comboBox还有第二般变化，我们只需要为它加上一个hiddenName属性，注意这hiddenName不能跟combo的id重复，这样生成的comboBox外表上看不出有什么变化，但firebug会将它的内在变化展露无疑。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'query',
    valueField: 'value',
    displayField: 'text',
    hiddenName: 'comboId'
});
```

看到了吧？最后一个参数，我让hiddenName的值是'comboId'，这样它跟div中的id="combo"不会重复。

```
<div id="ext-gen7" class="x-form-field-wrap" style="width: 144px;">
    <input id="comboId" type="hidden" name="comboId" value=""/>
    <input id="combo" class="x-form-text x-form-field
    x-form-empty-field" type="text" autocomplete="off"/>
    <img id="ext-gen8" class="x-form-trigger
    x-form-arrow-trigger" src="../../resources/images/default/s.gif"/>
</div>
```

从firebug展示的dom树你又看到了什么？我可是用蓝色标记出来啦。多出的这一行就是咱们使用hidden的结果。这让comboBox又多出一个hidden的input来，而这个隐藏的东西的id和name都为comboId（实际上，你也可以分别设置它的id和name，分别用hiddenId和hiddenName指定）。

这个隐藏东东的value会随着你的选择而改变，不过它的值一直等于选中项目的value就是了。

为啥多此一举呢，偶觉得应该是模拟普通的form吧，如果不单独做一个对应value值的输入框，咱们没有任何办法使用普通表单进行提交，这样完全绑定到ext的api上的作为可不是jack的初衷吧。呵呵~一

切灵活为准嘛。

4.4.4. 嘿嘿~本地的做完了，试试远程滴。

哈哈，其实超级简单啦。原来咱们用Ext.data.SimpleStore从本地数组里取数据，这次仿照grid的方式，用Ext.data.Store配合proxy和reader从后台读取数据好咯。

So ...

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'04-04.txt'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'value'},
        {name: 'text'}
    ])
});
```

我们使用HttpProxy去04-04.txt读取数据，还是用ArrayReader把获得的数据分成value，text两列。04-04.txt大概是这样的。

```
[
    ['value1','text1'],
    ['value11','text11'],
    ['value111','text111'],
    ['value1111','text1111'],
    ['value11111','text11111'],
    ['value2','text2'],
    ['value22','text22'],
    ['value222','text222'],
    ['value2222','text2222'],
    ['value22222','text22222']
]
```

现在我们大咧咧去更新页面啦。咦？怎么点了选择没有数据？再点，还是没数据，再点，再点点。。。

兄弟，点累了吧？谁让你那么着急的，还没改完呢。

我们还需要把comboBox的mode参数从local改成remote，其实默认就是remote，不设置也可以。吼吼，上眼：

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text'
});
```

除了mode改成remote了以外，其他照旧。

其实，也有不改成remote就读取的方法哟，你手工运行一下store.load()就可以啦。像这样：

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'04-04.txt'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'value'},
        {name: 'text'}
    ])
});
store.load();
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text'
});
```

这样做与上边的不同之处是，你可以决定什么时候对comboBox的数据进行初始化，如果不自己运行store.load()，数据会在第一次点击选择按钮的时候进行读取。最后效果都是一样啦，如何使用，任君选择哈。

不过，请记住一点，如果你的mode设置为remote，又使用了store.load()，那么会造成store读取两次数据，store.load()时一次，点击选择的时候一次，请留意哟。

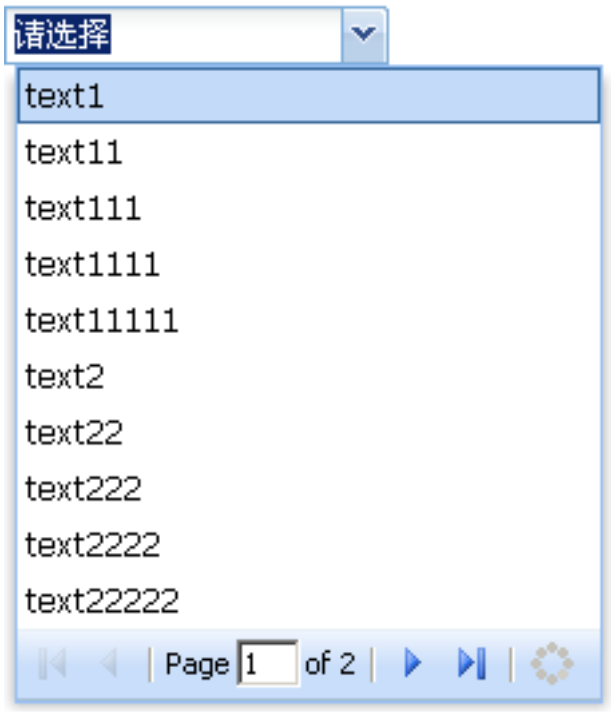
例子在lingo-sample/1.1.1/04-04.html。

2.0的部分，请记得将applyTo:'combo'的部分修改一下，例子见：lingo-sample/2.0/04-04.html。

4.4.5. 给咱们的comboBox安上零配件

教你几招另类配置，至于有用没用，你就自己试去吧。

1. 我们也来分页



代码如下哦，实际上只添加了两个参数就可以咯。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5
});
```

pageSize是主角，它决定每次显示多少个记录，ext内部自动计算是否进行分页。

minListWidth只是配角，但不设置它的话，你就只能看到不完整的分页条。

还有一点，mode必须是'remote'，如果你写成local，分页条就没法用了。因为你想啊，数据都已经在本地读取好了还分什么页啊。

最后就是，满足了上边的这些条件，咱们就获得了一个闪亮的分页条，让我们尽情的跳转吧。As your wish.

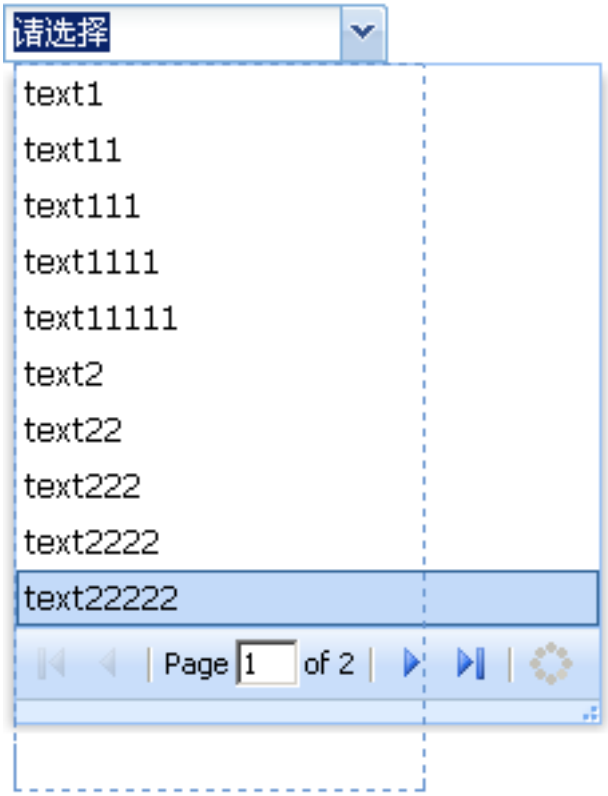
2. 手工改变弹出列表的大小

构造comboBox时加上resizable:true，就可以对弹出列表进行拖拽，现在你想要它变成多大都可以哦。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
```

```
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5,
    resizable: true
});
```

看一下图图吧，希望你注意到了右下角的拖拽标志，把鼠标放到上边，尽你所想哟。



3. 是否允许用户自己填写内容

吼吼，不知道你是否发现comboBox跟原生select之间的一个很大的不同。comboBox可以让用户自由填写数据哟。你也看到了咱们对comboBox内部的分析啦。显示框只是一个type="text"的input输入框，当然是允许用户输入的，但这可能不是我们想要的，为了达到与select一样的只能选择的效果，该如何呢？

答案现在揭晓：我们需要的只是一个readOnly。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5,
    resizable: true,
    readOnly: true
});
```

```

```

这使得用户只能从我们提供的数据中进行选择，不能随便填写自己的数据，保持了原生select的原貌。

偶把这三个配件组装到一起，放在了lingo-sample/1.1.1/04-05.html里，有兴趣的话就去看看的哟。

嗯，2.0还是applyTo:'combo'的区别，其他的一样哟，看看lingo-sample/2.0/04-05.html。

4.4.6. 每次你选择什么，我都知道

现在就要透露一些ext里一些悬疑的东西了，那就是event事件。所以要好好听哦，错过了没重播哟。

你猜我想要做什么？我现在可是要在你每次选择一项时，就弹出个窗口告诉全世界的人你选择了啥。就像下面这样。

嘿嘿，你问我怎么知道的？有人通知我撒。什么？你不知道？那怪谁撒？ext是会在你执行操作的时候，用秘密频率广播你到底做了些什么，你想知道如何收听这些广播吗？只需要如此这般，这般如此。

```
combo.on("select", function(comboBox){
    alert(comboBox.getValue() + '-' + comboBox.getRawValue());
});
```

这个on就是告诉combo，你要进行监听，第一个参数'select'告诉combo你要监听哪个频率，第二个参数是个function，传入的参数是combo本身，在它上面调用getValue()获得value，调用getRawValue()获得text，哈哈，真乃神仙也。

嘿嘿，你问我第一个参数为啥用'select'？这个，其实我是从api上查的，你打开docs里的Ext.form.ComboBox那一页，在Public Events一栏中可以看到好长一串的事件列表，其中就包括我们刚才用到的select，看看文档里是怎么说的：

select : (Ext.form.ComboBox combo, Ext.data.Record record, Number index)当列表中的一项被选中时，这个事件就会触发。

ext的事件机制就是这么神奇，当你选中了comboBox列表中的一项，ext就会找到所有绑定到select事件的监听器，比如咱们刚才就用on方法把咱们自定义的函数绑定到了select事件。注意哦，你可以绑定一个函数，就可以绑定N个函数，反正ext才不管多少，一视同仁都执行一遍，并把定义好的三个参

数传递过去，让咱们随便用。

你不相信吗？试试lingo-sample/1.1.1/04-06.html就知道咯。2.0在lingo-sample/2.0/04-06.html，唉唉唉，还是applyTo:'combo'的问题，不多说了。

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

思路是这样的，我们先准备好省市县的数据，都是二维数组，级联的数据就从它们之中选择。

```
var dataProvince = [
    ['河北','河北'],
    ['内蒙古','内蒙古']
];
var dataCityHebei = [
    ['唐山','唐山'],
    ['秦皇岛','秦皇岛'],
    ['承德','承德'],
    ['张家口','张家口']
];
var dataCityNeimenggu = [
    ['呼和浩特','呼和浩特'],
    ['包头','包头']
];
// 其中：遵化、迁安为县级市。
var dataCountyTangshan = [
    ['路南区','路南区'],
    ['路北区','路北区'],
    ['开平区','开平区'],
    ['古冶区','古冶区'],
    ['丰润区','丰润区'],
    ['丰南区','丰南区'],
    ['玉田','玉田'],
    ['遵化','遵化'],
    ['迁西','迁西'],
    ['迁安','迁安'],
    ['滦县','滦县'],
    ['滦南','滦南'],
    ['乐亭','乐亭'],
    ['唐海','唐海']
];
var dataCountyUnknow = [
    ['不知道','不知道']
];
```

然后我们要三个数据转换器，对应着省市县的三个Ext.data.SimpleData，原始数据经过它们的处理才可以交给comboBox，显示出我们期望的效果来。

```
var storeProvince = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: dataProvince
});
var storeCity = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: []
});
var storeCounty = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: []
});
```

注意到没有？只有storeProvince对应上了dataProvince，其他两个store里都用的空数组[]，这是因为没有选择省级地区之前，市级地区和县级地区是不应该有数据的。实际上你也不知道应该放什么，啥都没选呢。

现在可以制作三个comboBox了。这里我们提一句，每个combobox都配置成readOnly:true，这样省得用户输入无效信息，让triggerAction:all避免它去做autoComplete的判断。

```
var comboProvince = new Ext.form.ComboBox({
    store: storeProvince,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
var comboCity = new Ext.form.ComboBox({
    store: storeCity,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
var comboCounty = new Ext.form.ComboBox({
    store: storeCounty,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
comboProvince.applyTo('comboProvince');
comboCity.applyTo('comboCity');
comboCounty.applyTo('comboCounty');
```

在html里对应放上三个input，现在显示的部分就基本完成了。

```
<input id="comboProvince" type="text"/>
<input id="comboCity" type="text"/>
<input id="comboCounty" type="text"/>
```

现在我们来看关键部位，让三个comboBox关联起来，选择上级的时候，下级显示的信息要发生对应变化才可以。这就要用到我们上边说到的on('select')了。

```
comboProvince.on('select', function(comboBox){
    var province = comboBox.getValue();
    if (province == '河北') {
        storeCity.loadData(dataCityHebei);
    } else if (province == '内蒙古') {
        storeCity.loadData(dataCityNeimenggu);
    }
});
comboCity.on('select', function(comboBox){
    var city = comboBox.getValue();
    if (city == '唐山') {
        storeCounty.loadData(dataCountyTangshan);
    } else {
        storeCounty.loadData(dataCountyUnknow);
    }
});
comboCounty.on('select', function(comboBox){
    alert(comboProvince.getValue() + '-' + comboCity.getValue() + '-' + comboCounty.getValue());
});
```

像这样，给三个comboBox都设置上on('select')的事件监听，在选择省的时候，判断选择的是哪一项，如果是河北，就在市级comboBox里显示河北省的信息。

注意我们用到的storeCity.loadData(dataCityHebei);，这可以改变store内部的数据，store的数据改变了，对应的comboBox也就发生了变化。省市都是如此操作，最后在选择县的时候用alert弹出选择的信息。

如此这般，这般如此，我们的省市县三级级联就大功告成咯。庆贺下。

完整例子见lingo-sample/1.1.1/04-07.html。要在2.0里用的话，就把三个applyTo加上，其他的都一样啦，lingo-sample/2.0/04-07.html。

4.4.7.2. 再做一个有后台的，需要放在服务器上咯

后台用jsp编写，脚本的名字就叫做city.jsp，看看后台都包括些什么内容。

```
<%@ page contentType="text/html;charset=utf-8"%>
<%!
    String province = "[['河北','河北'],['内蒙古','内蒙古']]";
    String cityHebei = "[" +
        "['唐山','唐山']," +
        "['秦皇岛','秦皇岛']," +
```

```
        "['承德','承德'],"+
        "['张家口','张家口']"+
    "]";
    String cityNeimenggu = "["+
        "['呼和浩特','呼和浩特'],"+
        "['包头','包头']"+
    "]";
    String countyTangshan = "["+
        "['路南区','路南区'],"+
        "['路北区','路北区'],"+
        "['开平区','开平区'],"+
        "['古冶区','古冶区'],"+
        "['丰润区','丰润区'],"+
        "['丰南区','丰南区'],"+
        "['玉田','玉田'],"+
        "['遵化','遵化'],"+
        "['迁西','迁西'],"+
        "['迁安','迁安'],"+
        "['滦县','滦县'],"+
        "['滦南','滦南'],"+
        "['乐亭','乐亭'],"+
        "['唐海','唐海']"+
    "]";
    String countyUnknow = "[['不知道','不知道']]";
%>
<%
request.setCharacterEncoding("UTF-8");
response.setCharacterEncoding("UTF-8");

String type = request.getParameter("type");
if ("province".equals(type)) {
    response.getWriter().print(province);
} else if ("city".equals(type)) {
    String province = java.net.URLDecoder.decode(request.getParameter("id"));
    System.out.println(province);
    if ("河北".equals(province)) {
        response.getWriter().print(cityHebei);
    } else if ("内蒙古".equals(province)) {
        response.getWriter().print(cityNeimenggu);
    }
} else if ("county".equals(type)) {
    String city = request.getParameter("id");
    if ("唐山".equals(city)) {
        response.getWriter().print(countyTangshan);
    } else {
        response.getWriter().print(countyUnknow);
    }

}
%>
```

要提醒一句的是，因为ajax默认使用utf-8编码传输数据，我们这里例子里，我们需要把文件都转换成utf-8编码格式，否则就会出现乱码，对request和response也要进行处理，统一使用utf-8编码。实际上建议所有工程文件都统一使用utf-8编码，具有更好的扩展性。

开头的<%!%>部分，定义了我们可能返回的省市县三级数据，注意这里是json格式的字符串，回传到页面后，ext会把他们解析成需要的json数组。

主要程序体很简单，先判断type是什么，可能是省市县，根据这个再分别进入各自的流程。如果请求是需要省级信息该怎么办，如果是市级信息该如何做，乃至县级。

后台准备完毕，现在对前台进行修改，啊，其实就是把Ext.data.SimpleStore改成Ext.data.Store啦，让proxy都指向我们准备好的city.jsp，然后传递不同的参数就ok。

```
var storeProvince = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=province'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
var storeCity = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=city'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
var storeCounty = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=county'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
```

看到咯？除了url最后type的值不一样，前面都是一个模子里抠出来滴。

comboBox的定义完全没改呢，mode还是使用local方式，因为要控制读取数据的时机，所以手工操作store.load()更便于咱们这种情况。

只是在on('select')的时候改了几笔。

```
storeProvince.load();
comboProvince.on('select', function(comboBox){
    var value = comboBox.getValue();
    storeCity.load({params:{id:value}});
});
comboCity.on('select', function(comboBox){
    var value = comboBox.getValue();
    storeCounty.load({params:{id:value}});
});
comboCounty.on('select', function(comboBox){
    alert(comboProvince.getValue() + '-' + comboCity.getValue() + '-' + comboCounty.getValue());
});
```

看到咯，先执行storeProvince.load();将省级数据初始化，然后判断选择哪一项，取得数据以后才调用load()方法去后台读取数据，上一级comboBox的值被当作参数传递给后台，技巧就在于load的时候，使用{params:{id:value}}设置参数，后台就可以获得id的值来判断究竟要返回什么值了。

哦呵呵呵~为了统一编码，04-08.html和city.jsp都另存为utf-8编码，你可以在lingo-sample/1.1.1/

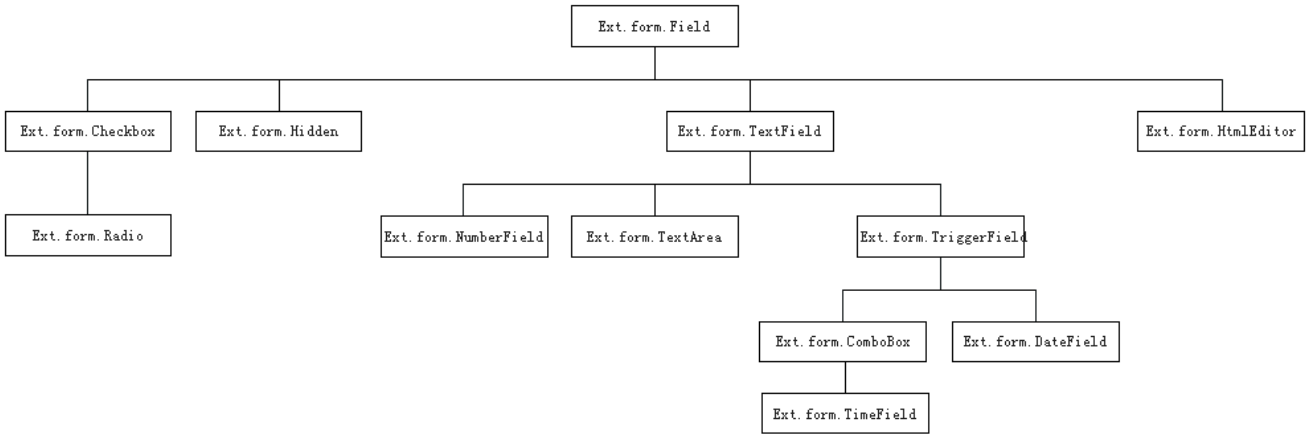
目录找到它们。

啊，受不了啦，2.0还是要改applyTo，你自己改去吧。如果实在搞不定就看lingo-sample/2.0/04-08.html。

4.5. 把form里的那些控件全部拿出来看看

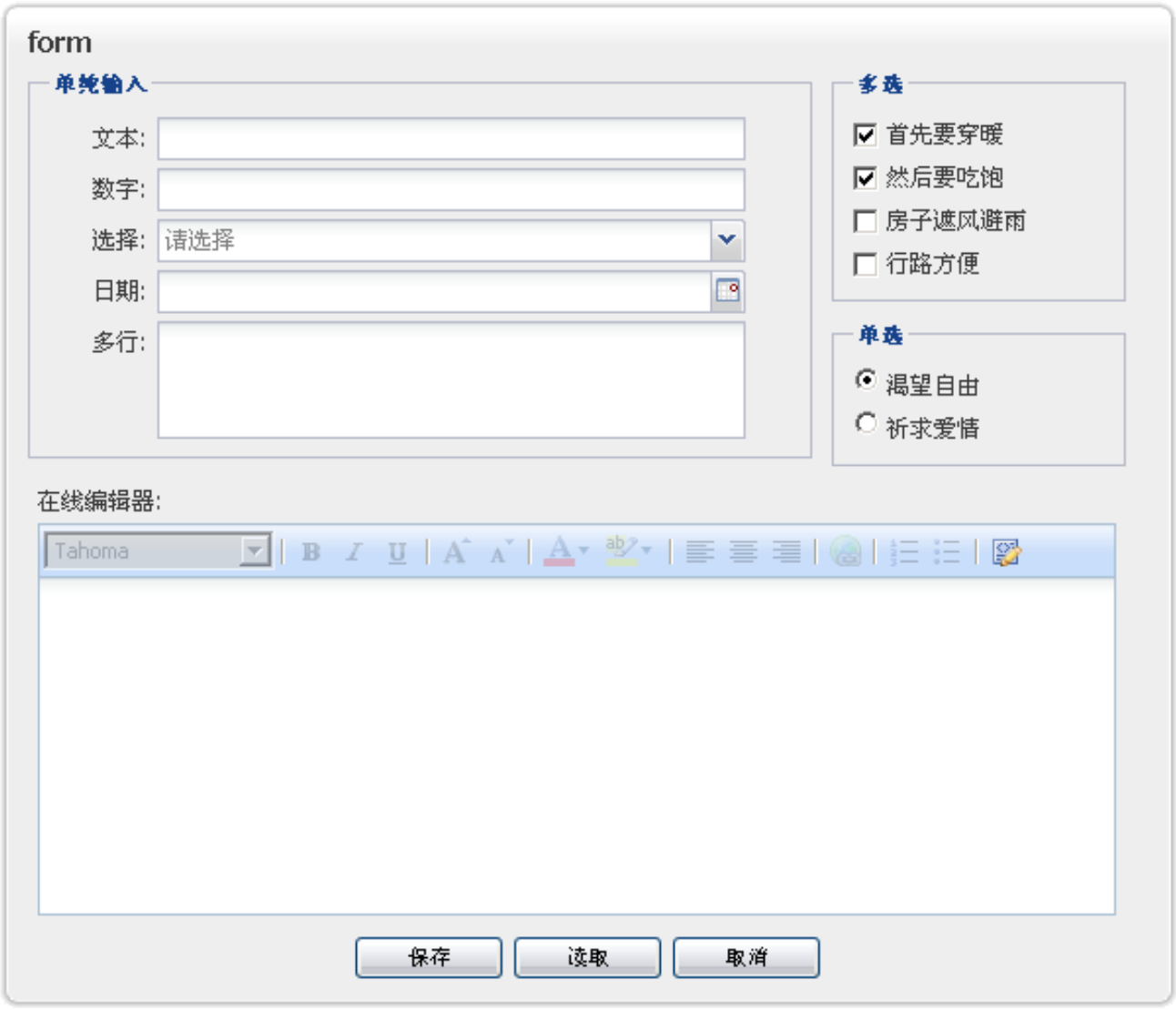
一个一个列出来就是，TextField, TextArea, CheckBox, Radio, ComboBox, DateField, HtmlEditor。然后2.0里多了Hidden和TimeField。全部放在一起的继承关系如下图。

Field继承体系

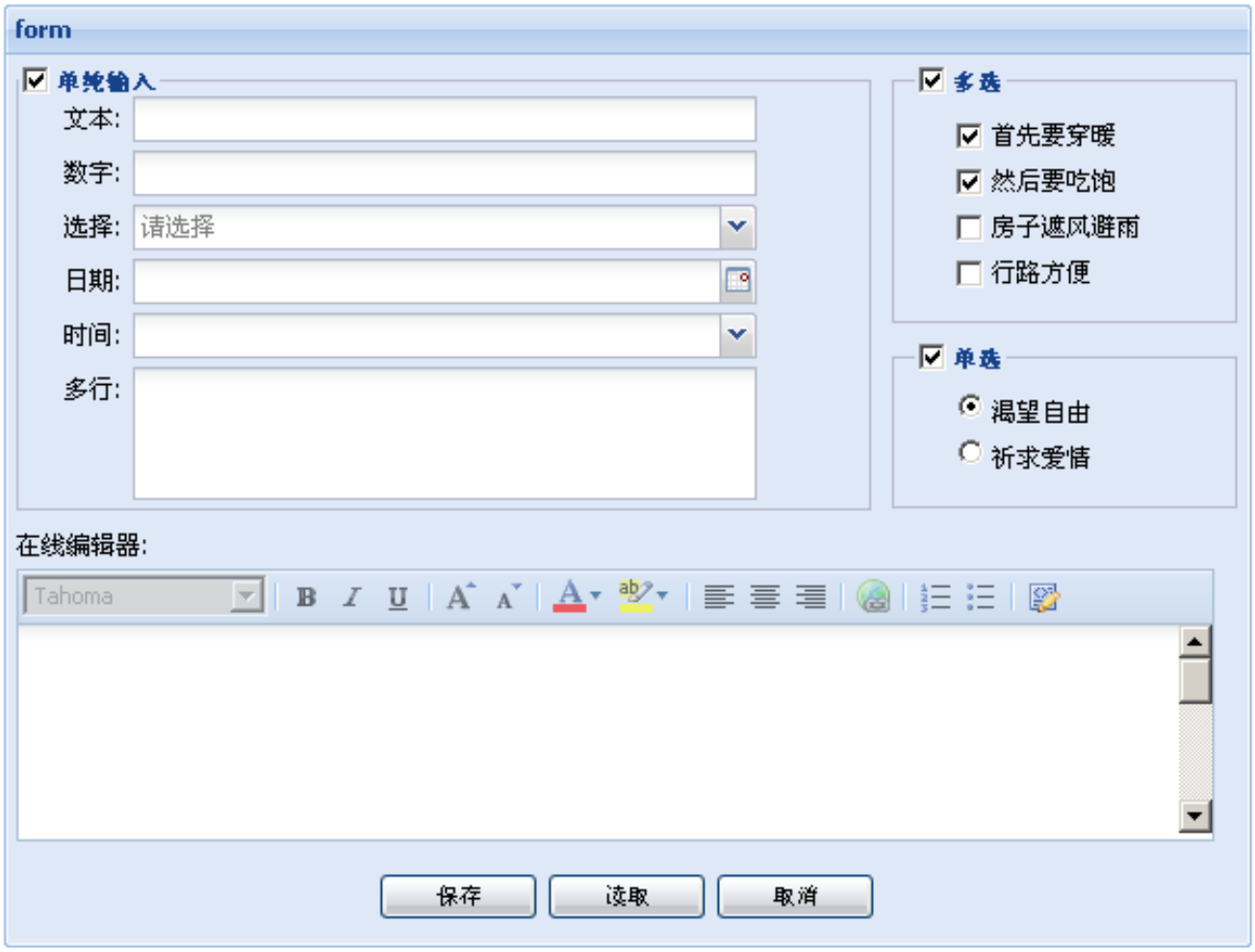


现在就拿出一个包含了所有组件的表单来从高空俯览一下这些控件，每每从宏观审视整体总让人有一种事事尽在掌握中的感觉。

1.x里的样子：



2.0里比1.x多了两个控件，但使用方法可是大不同，布局方式更加紧凑：



需要注意的一点是，2.0中HtmlEditor先需要Ext.QuickTips.init();初始化提示功能，否则会报错。

html内容太长，这里就不贴了，1.x在lingo-sample/1.1.1/04-09.html。2.0在lingo-sample/2.0/04-09.html。

4.6. form提交数据的三重门

form总是要提交的，否则要表单作甚哦？只不过，普通html中的表单只能做最单纯的submit提交，ext的form却支持三种形式的提交。一种是原始form形式的提交，两种ajax形式的。嗯，让我们慢慢讲来。

下面将会使用的例子是一个只包含一个TextField的form，让我们把这里的数据提交到后台。

4.6.1. ext中默认的提交形式

form对象有一个submit函数，用途就是提交数据，像这样：

```
Ext.onReady(function(){

    var form = new Ext.form.Form({
        labelAlign: 'right',
        labelWidth: 50,
        url: 'form.jsp'
    });
```

```
    form.add(new Ext.form.TextField({
        fieldLabel: '文本框',
        name: 'text'
    }));
    form.addButton({
        text: "按钮",
        handler: function() {
            form.submit();
        }
    });
    form.render("form");

});
```

这里是1.x的情况，要给form设置一个url参数，指向提交给后台的url地址，记得给TextField加上一个name属性，这样后台才知道接收到的数据是哪个控件的，最后是改造按钮。text表示按钮显示的文字，handler则是点击按钮时调用的函数，我们这里直接调用form的submit()函数，将整个form中的数据发送到后台。

后台接收数据的脚本form.jsp，跟以前的方式相同，唯一的区别就是不要跳转了，需要返回一个json字符串，提示操作是否成功了：

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    String text = request.getParameter("text");
    System.out.println(text);

    response.getWriter().print("{success:true,msg:'成功'}");
%>
```

2.0与1.x里虽然风格有了很大不同，但使用方式还是差不多，不过因为FormPanel是布局，没有提供submit()函数，你要先获得它里边包含的BasicForm才能提交。

```
buttons: [{
    text: '按钮',
    handler: function() {
        form.getForm().submit();
    }
}]
```

其他地方就都差不多啦。例子见lingo-sample/1.1.1/04-10.html和lingo-sample/2.0/04-10.html。

解决了一个问题，另一个问题又冒出水面。form默认使用ajax提交数据，咱们怎么知道提交是不是成功啦？这点一下按钮什么动静都没有可如何是好？别急，ajax都是可以回调的，更别说form还封装了自己的处理方式呢。

要想获得反馈，首先要修改submit方法，让它支持更多的功能：

```
```

```
form.submit({
    success:function(form, action){
        Ext.Msg.alert('信息', action.result.msg);
    },
    failure:function(){
        Ext.Msg.alert('错误', '操作失败！');
    }
});
```

点一下提交，成功之后的效果就变成这样啦。

回头看看代码，success和failure各自对应一个函数，这个跟直接使用ajax差不多，但是等等，success对应函数中传入的参数不一样，第一个参数是form，第二个参数是action。

这就是封装的结果了，第一个参数是form对象，想做什么都可以在它上边直接做了，比如，想提交一次之后清空所有数据，直接用form.reset()就行了。第二个参数是响应的信息，action.result给我们提供了一个简易通道，这样省去了获得responseText再转换成json的功夫，直接就可以从它里边调用返回的json数据了。比如我们后台返回的msg，就可以直接通过action.result.msg获得。

呵呵~比较简单吧？1.x和2.0的操作都是一样的，看看代码吧，在lingo-sample/1.1.1/04-11.html和lingo-sample/2.0/04-11.html。

4.6.2. 使用html原始的提交形式

ext默认的提交形式是不会进行页面跳转的，主要是考虑到one page one application的形式，不过还是用同志怀念以前提交一下就跳到其他地方的日子，这可不是什么难事，我还要再次强调一下，ext就是js，原来js能做的事情它都可以做，Ext.form.Form也只是对原始form的装饰而已，你找出那个咱们都熟悉的form，直接在它上边调用submit()就行了。

难点就是找出层层包裹在Ext.form.Form中的form，我们在这里给你提点一下，在ext中所有的控件都有el的，el都是有dom的，这个dom模型就是ext控件在页面上真实对应的部分了。于是我们只需要修改按钮的handler函数：

```
handler: function() {
    form.el.dom.action = "form.jsp";
    form.el.dom.submit();
}
```

要注意的是，ext动态生成了form，却没有把action部分添上，如果你不手工设置一下action，它就只能刷新本页。

2.0的稍有不同，不过主要思想是相同的。

```
handler: function() {
    form.getForm().getEl().dom.action = "form.jsp";
    form.getForm().getEl().dom.submit();
}
```

呵呵~，点一下提交，页面就刷新成form.jsp了，页面上显示着本来应该传给ajax的字符串，记得啊，它这样造成的页面跳转很可能把one page one application的系统搞乱，还是适合用在普通的环境下。

例子在lingo-sample/1.1.1/04-12.html和lingo-sample/2.0/04-12.html。

4.6.3. 单纯ajax

跳过form自己的ajax，另外使用ajax就比较单纯了，实际上效果跟使用form自身的submit都是利用了ajax，但如果你想自己对form里的数据做进一步的控制，那么自己调用ajax也是一个不错的选择。

既然是使用外部ajax，我们只要关系如何从form里把那些字段的数据取出来就行了，一共有下面几种方式：

1. form.getValues()函数，它有一个参数，如果是false，返回的是json组装的字符串，如果是false返回的是json对象，对应里边一个一个字段的名称和值。默认是false。
2. findField()函数，这个函数可以获得一个一个的控件，比如我们现在有一个TextField，名字是text，我想要这个控件就要这么做：

```
var text = form.findField('text');
```

我们使用最简单的getValues(true)函数，来配合ajax，用getValues(true)获得数据，使用ajax传递给后台，然后判断回调。

```
Ext.lib.Ajax.request(
    'POST',
    'form.jsp',
    {success:function(response){
        var result = Ext.decode(response.responseText);
        Ext.Msg.alert('信息', result.msg);
    },failure:function(){}},
    "text=" + encodeURIComponent(form.getValues(true))
);
```

呵呵~1.x和2.0都差不多啦，只要注意2.0要先使用getForm()获得FormPanel里的BasicForm再进行这些操作。例子在lingo-sample/1.1.1/04-13.html和lingo-sample/2.0/04-13.html。

4.7. 验证苦旅

绝对不能信任用户输入的数据，这是咱们做系统的基础。你要是不管他，他输入的东西100%都是不能用，怎么办？必须在规定好哪些数据要怎么添，如果填错了就不能提交到后台。这里ie和firefox有些许不同，如果校验失败，firefox下调用submit()是不会发生提交的，但ie下必须先使用form.isValid()自行判断，如果返回false，就不能再调用submit()，要不然还是会把非法数据都提交到后台。切记切记。

ext把验证封装到了form这些控件中，就让咱们看看怎么利用它。

4.7.1. 不能为空

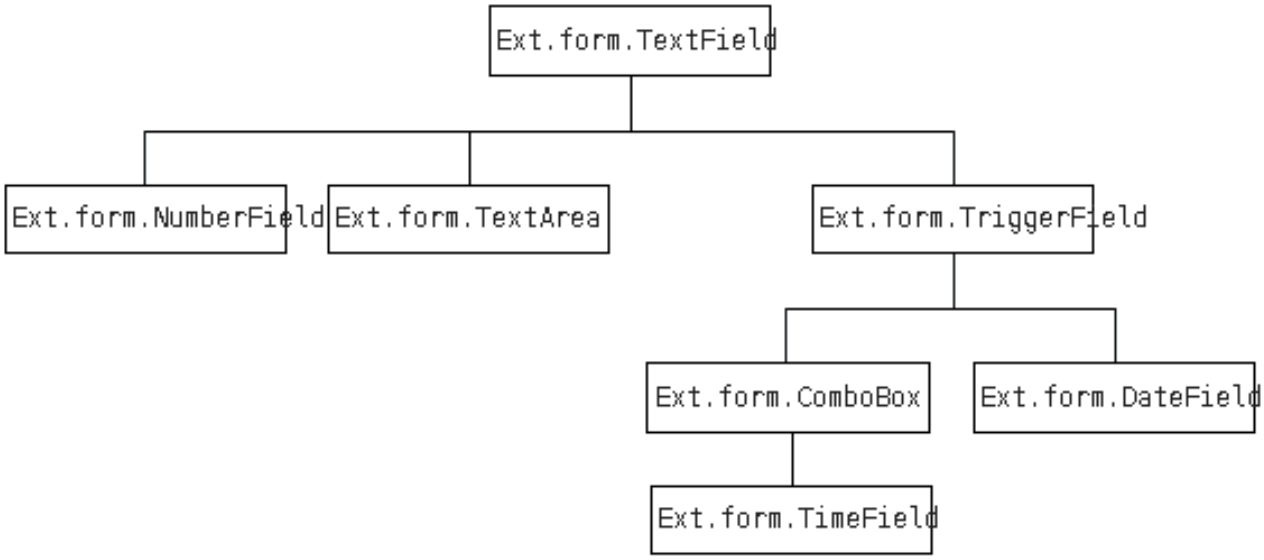
最最经典的验证手法，某个TextField或是什么东西必须添个值。

```
new Ext.form.TextField({
    name: 'text',
    fieldLabel: '文本框',
    allowBlank: false
});
```

allowBlank默认是true，就是说允许不输入数据，咱们把它改成false，然后看看会出现什么效果。

如果你没添任何东西，TextField下就会出现红线，不过鼠标放上去的提示需要Ext.QuickTips.init();的支持。

怎么样？很显眼吧？比邦邦地弹出alert好多了吧？那么究竟哪些控件可以使用这个allowBlank参数进行非空校验呢？因为allowBlank不是在Ext.form.Field中定义的，所以并不是所有控件都可以使用它，allowBlank第一次出现在Ext.form.TextField中，只有继承自它的控件才可以使用非空校验，具体可参考继承树：



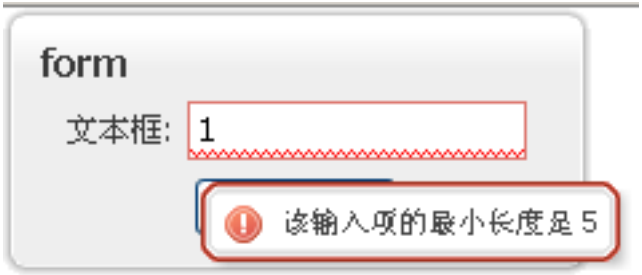
例子在lingo-sample/1.1.1/04-14.html和lingo-sample/2.0/04-14.html。

4.7.2. 最大长度，最小长度

最大长度是一种常见的校验方式，数据库只允许输入255个字符，客户输入超出限度的数据就会引起错误，最小长度跟最大长度是兄弟，所以一并介绍了。

```
new Ext.form.TextField({
    fieldLabel: '文本框',
    name: 'text',
    maxLength: 10,
    minLength: 5
});
```

咱们设置上最大不能超过10个字符，最小不能少于5个字符，如果不合标准就会提示错误：



例子在lingo-sample/1.1.1/04-15.html和lingo-sample/2.0/04-15.html。

4.7.3. 借助vtype

ext提供了一套默认的校验方案，其实就是一系列输入规则与出错提示，使用这些可以让我们不用去背诵那么一长串正则表达式，只要记忆一下vtype的名字，然后配置到控件里就行了。就像这样：

```
new Ext.form.TextField({
    fieldLabel: '文本框',
```

```
    name: 'text',
    vtype: 'email'
})
```

这里我们给vtype设置的是email，限制只能使用邮箱地址的格式。

呵呵~是不是很简单啊，只是设置了vtype:'email'就获得了这些。这些验证信息都定义在Ext.form.VTypes中，默认的VTypes一共有四种验证信息：

1. alpha，只能输入英文字母

2. alphanum，只能输入字母和数字

3. email，电子邮箱

4. url，网址

需要使用哪一种验证信息，只需要把vtype设置成对应的名称就可以了，你也可以自己做扩展，具体的方式可以参照VTypes.js。

邮箱验证的例子在lingo-sample/1.1.1/04-16.html和lingo-sample/2.0/04-16.html。

4.7.4. 自定义验证规则

其实就是正则表达式啦，写个regex对输入进行校验，这样无论什么样的数据形式，都逃不过咱们的法眼。那个，如果你没听过regex，或者自认是正则的苦手，那还是略过这节吧。咱们这里可不会花时间去将正则怎么用，那可太花时间了，反正上面的验证方式已经可以解决不少问题了。

这里我们验证一下只能输入汉字的情况，嗯，估计老外们也不会帮咱们预备这个。

```
new Ext.form.TextField({
    fieldLabel: '文本框',
    name: 'text',
    regex: /^[\u4E00-\u9FA5]+$/,
    regexText: '只能输入汉字'
})
```

嗯，嗯。其实原理很简单，就是为regex设置一个正则表达式，然后在进行校验的时候会去调用regex.test(value)，如果是true就表示校验通过，如果是false就提示校验失败，同时在提示框中放上regexText的内容。

例子在lingo-sample/1.1.1/04-17.html和lingo-sample/2.0/04-17.html。

4.7.5. 算不上校验的NumberField

Ext.form.NumberField是Ext.form.TextField的子类，本来应该继承了咱们上面所说的那些验证形式，不过这个NumberField可是另有奇妙之处，让人不得不把单独把它拿出来玩弄一番。

你只能在NumberField里输入数字，呼，这种方法简直让一般的校验方法靠边去了，其他控件都是在blur或者提交的时候对已输入的数据进行校验，NumberField根本就不让你输入它不想要的东西。厉害啊。

它这种排外的情绪到底到达一个什么程度呢？让我们来实地检测一下吧，测试项目如下：

1. 直接从键盘输入字母和非数字符号：无法输入NumberField。
2. 是不是它只限制了可以输入-, 0~9和小数点呢？那我们试着输入多个小数点，看看非法的数字形式可不可以输入进入：答案是可以的，NumberField只能限制输入的字符，但一旦输入结束，它就会自可进行校验，从左侧开始截取，一直保留到合法的数字格式为止。所以1.....就变成了1.00。
3. 那么，如果我直接把带有非数字的字符串粘贴到NumberField呢？粘贴是成功了，不过在编辑结束后，NumberField又对内容进行校验，把非法数据转化成最接近的数字了。

呵呵~由此可见，不管你是怎么输入的，反正它最后只留有效的数字，干得已经很漂亮了。

下面再摘几条数字方面的配置献给大家：

1. 不想让用户输入负数。将参数allowNegative设置为false，这个参数默认为true，这样就不能输入负号了。
2. 不想让用户输入小数。将参数allowDecimals设置为false，这个参数默认为true，这样就不能输入小数点了。
3. 这只小数精度，默认是保留到小数点后两位，百分位。修改参数decimalPrecision的值，就可以自己决定精确到小数点后多少位。
4. minValue和maxValue可以规定可输入数字的范围，这个可是数字专有的，让一个人输入的年龄在0到150之间不是很好么？

嗯，不过还是有点儿问题，即使设置了allowNegative:false和allowDecimals:false还是可以输入负号和小数点，虽然这两个符号会在验证的时候自动消失，但还是会让人觉得别扭，有没有办法想NumberField阻止其他字符一样，完全不允许输入呢？嗯，我看到了maskRe参数，希望它能来助咱们一臂之力。

让maskRe等于/\d/，它也是指向了一个正则表达式，把它镶到NumberField里就可以阻止符号和小数点了。呵呵~As your wish.

NumberField的例子在lingo-sample/1.1.1/04-18.html和lingo-sample/2.0/04-18.html。

4.8. 关于表单内部控件的布局问题

有言在先，1.x中的form布局实在不怎么样，所以提得不会很多。2.0里的布局更好用，咱们就多说一

些。

4.8.1. 什么都不做，默认的平铺布局

如果不做任何设置，直接把控件加入form，那么显示出来的是自上而下的默认布局。

看代码里就那么几行字。

```
form.add(
    new Ext.form.TextField({fieldLabel: '俩字'}),
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '四个汉字'})
);
```

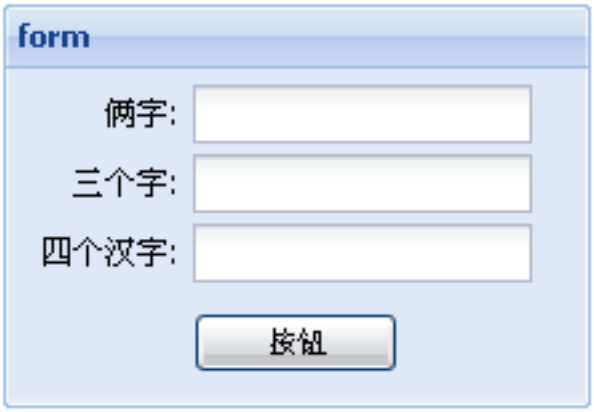
其实这个默认布局也能满足不少要求，简便不用额外配置就是它的好处，不过咱们也可以在它上面玩一些花样，修整它的边边角角。

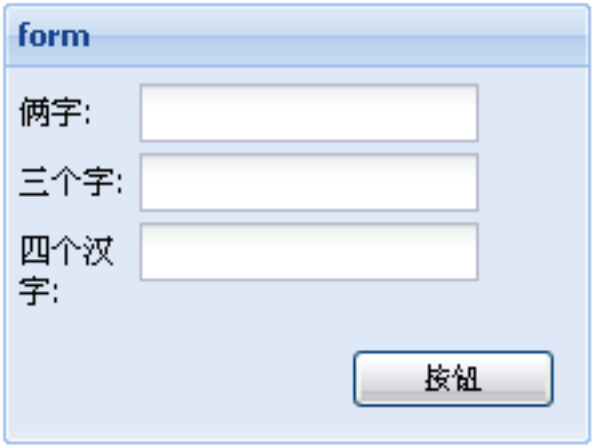
注意到这三个输入框的文字长度不同吗？我们可以通过配置labelAlign: 'right'使他们右对齐，一共有left, top, right三个值可以选，默认情况下是左对齐的。这些文字的宽度也可以设置，咱们这里使用的labelWidth: 60，宽度就是60像素，如果宽度不合适，文字会自动换行。按钮的位置也可以设置，有left, center, right三种选择，默认是right。

假如我们设置了labelAlign: 'left', labelWidth: 40和buttonAlign: 'right'就会变成这样。

1.x的例子在lingo-sample/1.1.1/04-19.html和04.20.html。

这个默认部分呢，2.0跟1.x没啥区别，看看图吧。





2.0的例子在lingo-sample/2.0/04-19.html和04-20.html。

4.8.2. 分裂，分列

别的别说，先分裂成1.x和2.0。这块不怪我，实在是1.x和2.0差太多了，而且有的功能1.x中不好实现，其实根本就是太难实现了，天哪，做个验证图片还要用DomHelper根据内部的id搞一大堆东西，还是别用1.x的啦。

4.8.2.1. 分裂，分列。1.x

简单来说，就是分成这个样子。

需要水平排布的时候，我们就需要借助form.column的力量了，它会从左到右把空间分成一列一列的，具体某一列的宽度在column中的第一个参数指定。

```
form.column(
    {width: 200},
```

```
    new Ext.form.TextField({fieldLabel: '俩字'})
);
form.column(
    {width: 200},
    new Ext.form.TextField({fieldLabel: '三个字'})
);
form.column(
    {width: 200},
    new Ext.form.TextField({fieldLabel: '四个汉字'})
);
```

从这里我们可以看到，我们把三个输入框放到三列中，每一列的宽度是200，最后呈现的效果就是咱们刚才看到的三列平铺。

稍微注意一下，使用layout:'column'要记得去掉上层的defaultType，而且column下的那些对象里都要先用layout:'form'整理一下，怎么你会看不到fieldLabel哦，留意下吧。

例子：lingo-sample/1.1.1/04-21.html。

请别误解，别以为每列只能放一个控件，每列放几个都没问题呢：

```
form.column(
    {width: 200},
    new Ext.form.TextField({fieldLabel: '俩字'}),
    new Ext.form.TextField({fieldLabel: '俩字'})
);
form.column(
    {width: 200},
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'})
);
form.column(
    {width: 200},
    new Ext.form.TextField({fieldLabel: '四个汉字'}),
    new Ext.form.TextField({fieldLabel: '四个汉字'}),
    new Ext.form.TextField({fieldLabel: '四个汉字'}),
    new Ext.form.TextField({fieldLabel: '四个汉字'})
);
```

如你所见了，想加几个，就加几个。

例子在lingo-sample/1.1.1/04-22.html。

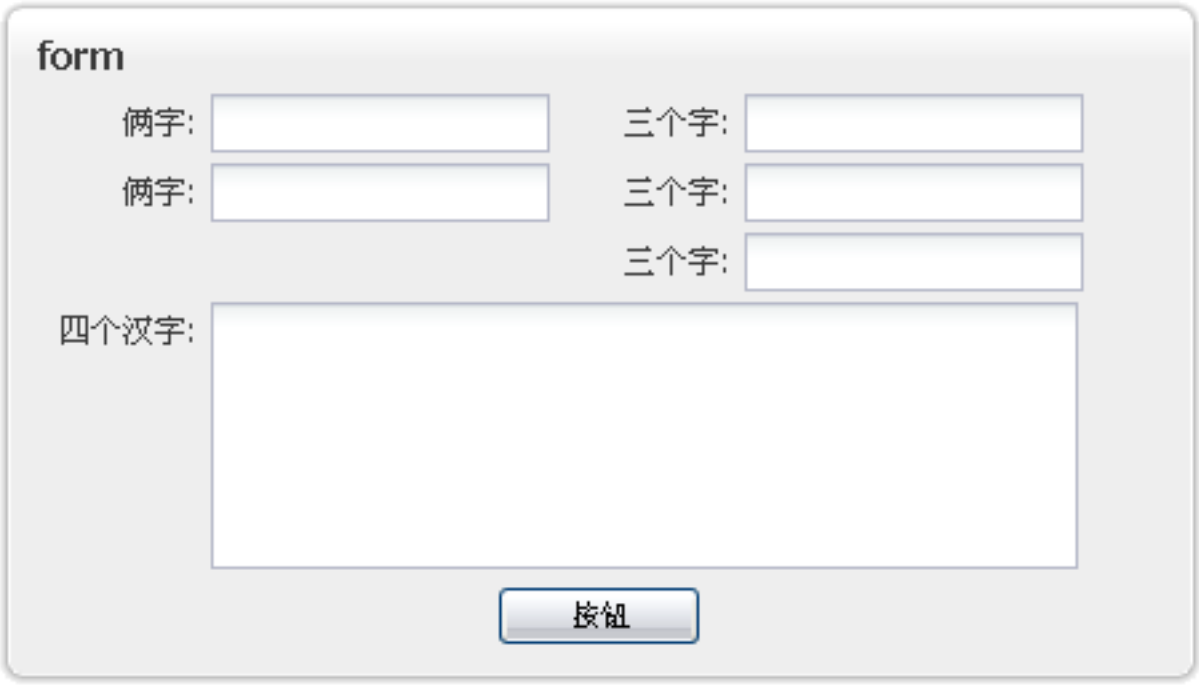
不过一直这样种树似的也没啥意思，列数太多了，看着也心烦，有没有法子换行啊？

答案是：可以。

我们只需要在需要换行的那一列上加上个clear:true就可以了，现在咱们打第二列开始换行，就在第二列上添这个参数。

```
form.column(
    {width: 200, clear: true},
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'})
);
```

为了让界面丰满一些，我们在换行以后放上个TextArea，呵呵~



例子在lingo-sample/1.1.1/04-23.html。

4.8.2.2. 分裂，分列。2.0

啊拉，让我们看看2.0的吧，口水都忍不住要流出来啦。

首先是分三列。

与1.x中不同的是，column不再是一个函数，而是一种布局，咱们需要先用layout: 'column'说明下面要使用的是列布局，然后在items中指定的每列中使用columnWidth指定每列所占的比例，唉，比例呀，可比用手掰着算具体的宽度好受多啦。

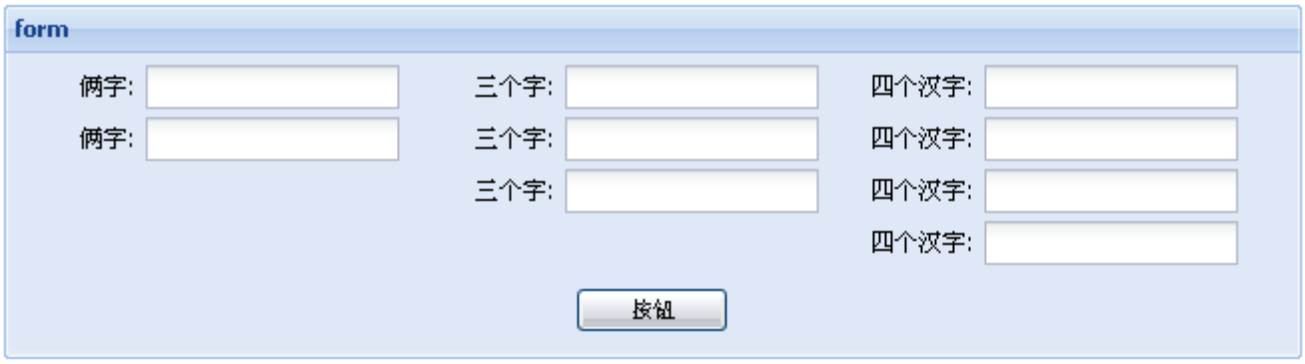
要注意一点儿，使用了column布局，就不能在form中直接使用defaultType指定默认的xtype了，否则会影响column的布局结果。每一列中也必须手工指定使用layout: 'form'，才能在每列中正常显示输入框和文字，顺便说一下，这个form其实就是FormPanel默认使用的布局方式，从上到下哟。

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 60,
    title: 'form',
    frame:true,
    width: 650,
    url: 'form.jsp',

    items: [{
        layout:'column',
        items: [{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '俩字'}]
        },{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '三个字'}]
        },{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '四个汉字'}]
        }]
    }],
    buttons: [{
        text: '按钮',
        handler: function() {
            form.getForm().submit();
        }
    }]
});
```

这里从最顶级分成三列，然后每列占0.33，就是33%啦。然后每列都使用form布局，每列都只包含一个TextField，这就完成了咱们2.0中的列布局了。

例子在lingo-sample/2.0/04-21.html。

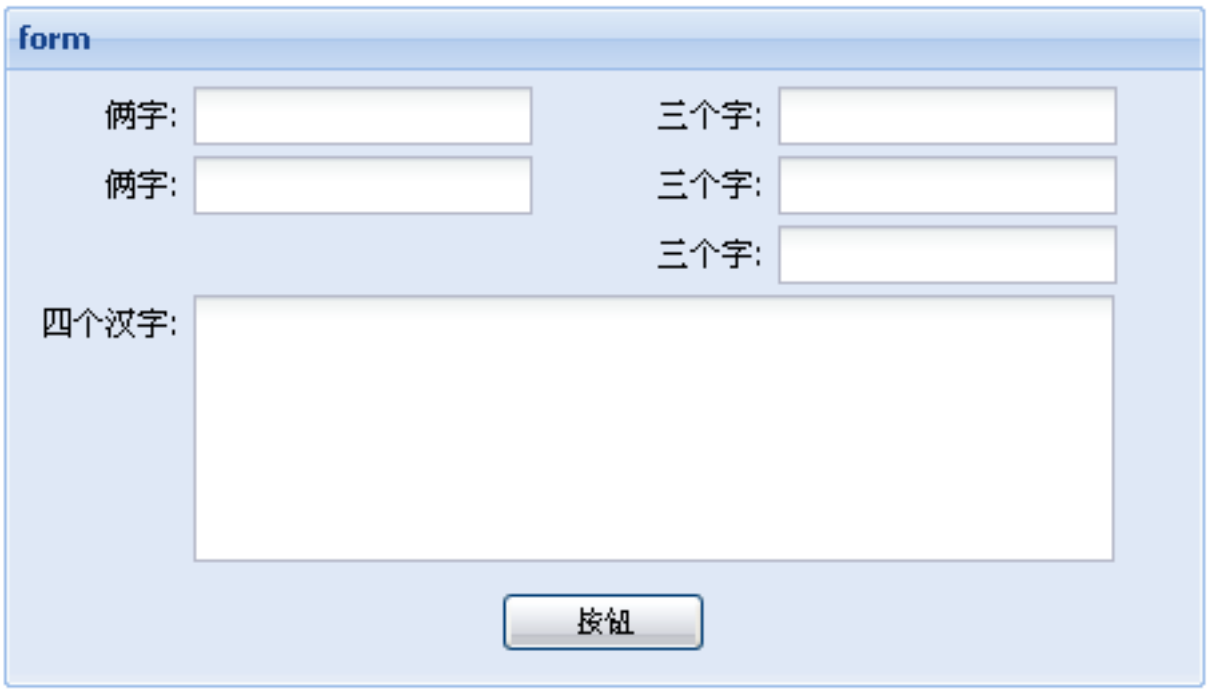
再来谈每列放多个输入框其实就挺无聊的啦，不过还是演示一下的好。



例子在lingo-sample/2.0/04-22.html。

现在是换行啦，说真的，马上你就能感到2.0的魔力啦。

啦啦啦，咱们要的就是这样一个效果。



不用搞什么clear:true啦，现在现在咱们让在layout:'column'下直接放一个textarea就好了，form和column两种布局的组合，可以搞出任何你可以想出来的效果啦。

```
items: [{
    layout:'column',
    items: [{
        columnWidth:.5,
        layout: 'form',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '俩字'},
            {fieldLabel: '俩字'}
        ]
    },{
        columnWidth:.5,
        layout: 'form',
        defaultType: 'textfield',
        items:[
```

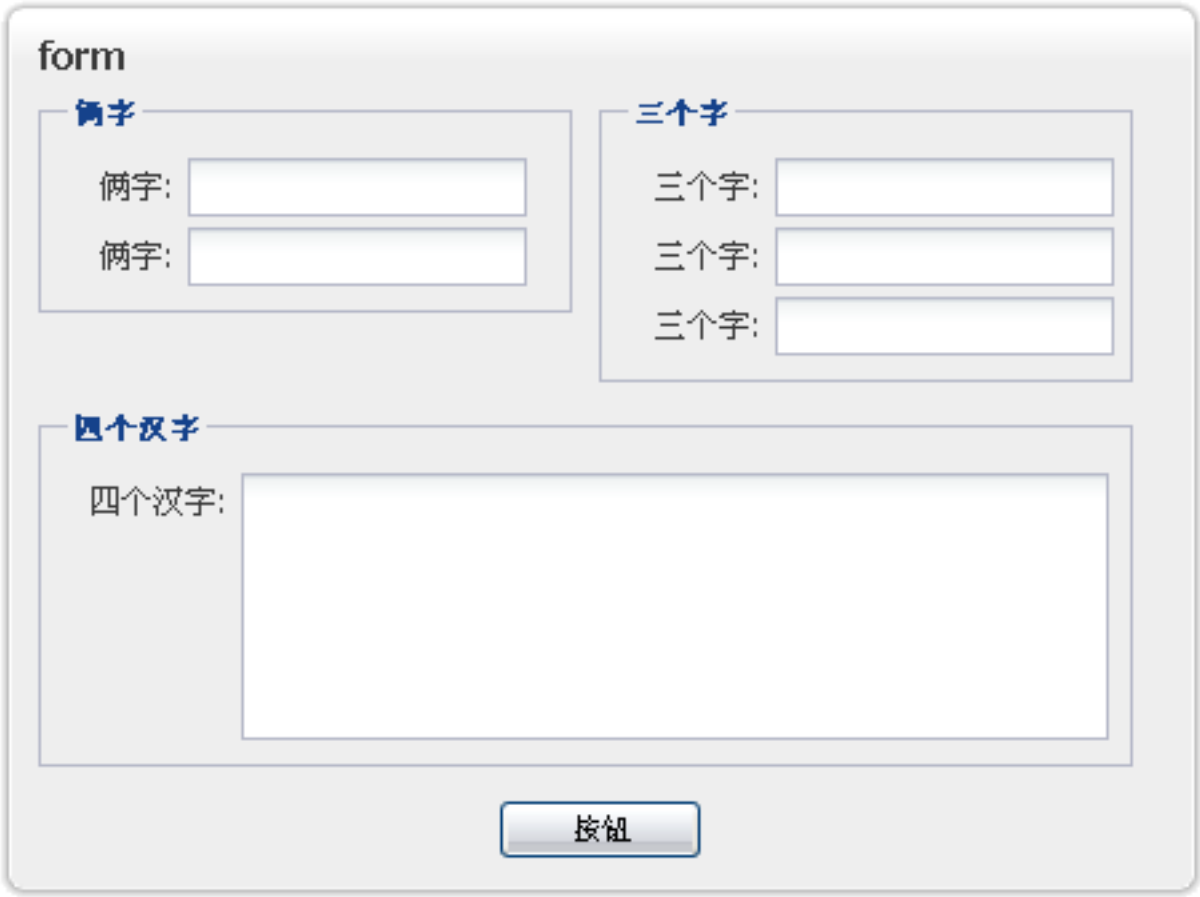
```
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'}
        ]
    }]
},{
    width: 345,
    height: 100,
    xtype: 'textarea',
    fieldLabel: '四个汉字'
}],
```

例子就在lingo-sample/2.0/04-23.html。

4.8.3. fieldset是个神奇的东西

标准html中，是要把input都放到fieldset中，以此来显示表示分组结构。虽然ext里的form已经这么漂亮了，实在是没有用fieldset做外框的必要，但我们依然可以用fieldset来进行内部分组。

为了好玩一些，我们把column和fieldset一起使用，嘿嘿~复杂一些更凸显效果滴。



是不是觉得更有趣了呢？你可以给你的那些开爱的小控件们分组了，还可以在fieldset上头写几个说明，像基础数据呀，可选数据啊，某某某某滴。看下代码，然后咱们好继续研究：

```
form.column({width: 200, labelWidth: 40});
form.fieldset(
    {legend:'俩字'},
```

```
    new Ext.form.TextField({fieldLabel: '俩字'}),
    new Ext.form.TextField({fieldLabel: '俩字'})
);
form.end();

form.column({width: 200, labelWidth: 50, style:'margin-left:10px', clear: true});
form.fieldset(
    {legend:'三个字'},
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'})
);
form.end();

form.column({width: 410, labelWidth: 60});
form.fieldset(
    {legend:'四个汉字'},
    new Ext.form.TextArea({
        width: 325,
        height: 100,
        fieldLabel: '四个汉字'
    })
);
form.end();
```

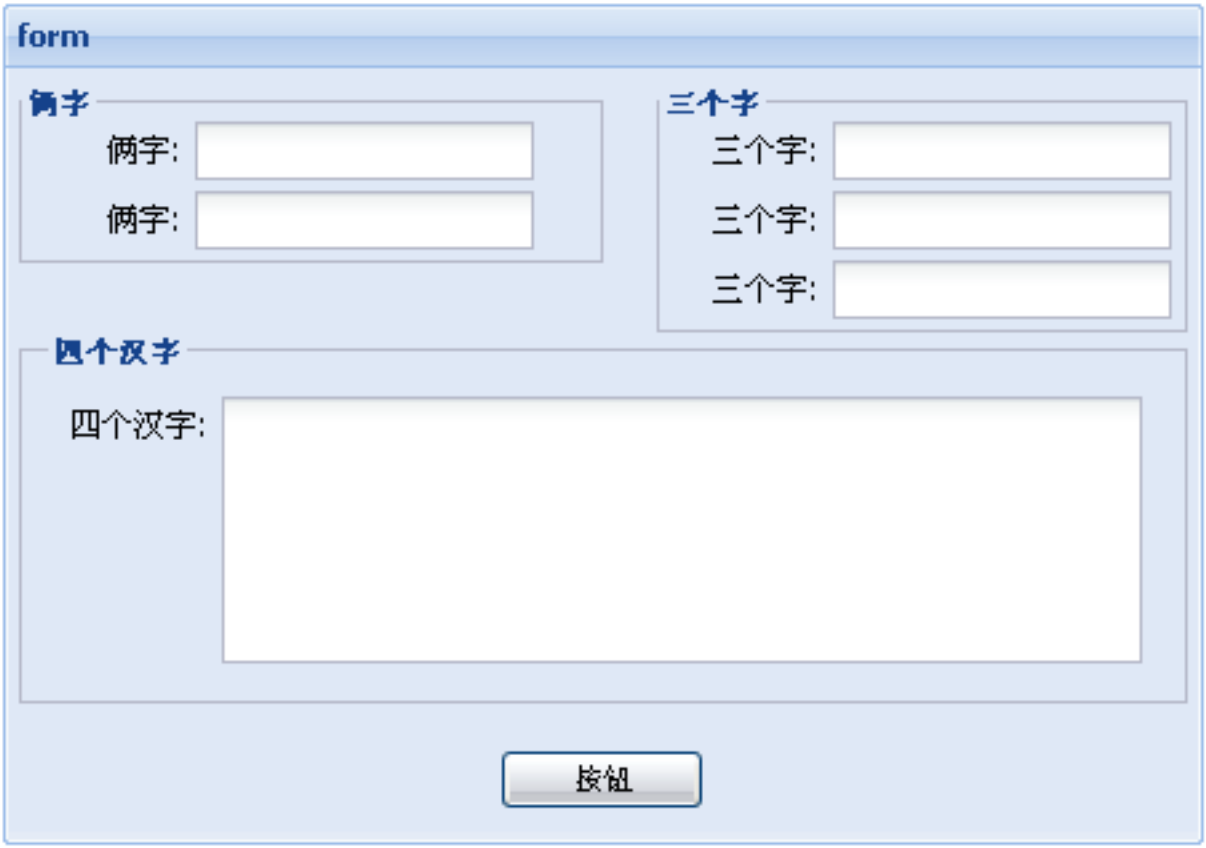
其实，我们就是把column拆开，然后在里边安装上fieldset，legend代表这个fieldset的标头，不过还要记住一点，使用过的column要在最后调用一下form.end()才能结束，否则不会开始下一列的。而且，你要想给换行后的textarea的也加上fieldset，你就要再次使用column，真真真，麻烦死啦，你自己去看例子吧。我不管你啦。lingo-sample/1.1.1/04-24.html。

这放到2.0里就很简单啦，fieldset只是一个普通的xtype。

```
items: [{
    layout:'column',
    items: [{
        columnWidth:.5,
        layout: 'form',
        xtype: 'fieldset',
        title: '俩字',
        autoHeight: true,
        defaultType: 'textfield',
        items:[
            {fieldLabel: '俩字'},
            {fieldLabel: '俩字'}
        ]
    },{
        columnWidth:.5,
        layout: 'form',
        xtype: 'fieldset',
        title: '三个字',
        autoHeight: true,
        style: 'margin-left: 20px;',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'}
```

```
        ]
    }]
},{
    xtype: 'fieldset',
    title: '四个汉字',
    autoHeight: true,
    items: [{
        width: 345,
        height: 100,
        xtype: 'textarea',
        fieldLabel: '四个汉字'
    }]
}],
```

注意加上title作为标题和autoHeight:true，让fieldset看起来更漂亮一些，至于那个style:'margin-left: 20px;'则是为了让第二列跟第一列别靠得太近了，css啦。最后看起来就像是这样。



这个这个，统一的写法，无论如何也比记一大堆复杂的函数用法要强，更别提1.x中不同的组合可能出现不同的结果啦。看看这个例子lingo-sample/2.0/04-24.html。

4.8.4. 当某一天，需要往form加个图片什么的，该咋办？

我们说的是怎么向form里添加不是Ext.form.Field的东西，在1.x中，这真是难比登天啊。基本的步骤是搞个fieldset，设置上id，在form.render()之后，我们就可以用找到这个id所在的dom，然后用DomHelper往里插任何东西了。

我们先给fieldset加上id，让它的id等于photo。

```
```

```
form.fieldset(
    {id: 'photo',legend:'三个字'},
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'}),
    new Ext.form.TextField({fieldLabel: '三个字'})
);
```

在渲染好表单之后，就可以搞到这个id的元素，然后往里放东西了。

```
var photo = Ext.get('photo');
var c = photo.createChild({
    tag:'center',
    cn: {
        tag:'img',
        src: 'user_female.png',
        style:'margin-bottom:5px;'
    }
});
```

最后就变成了这样，右边三条输入框下多了个小图片。

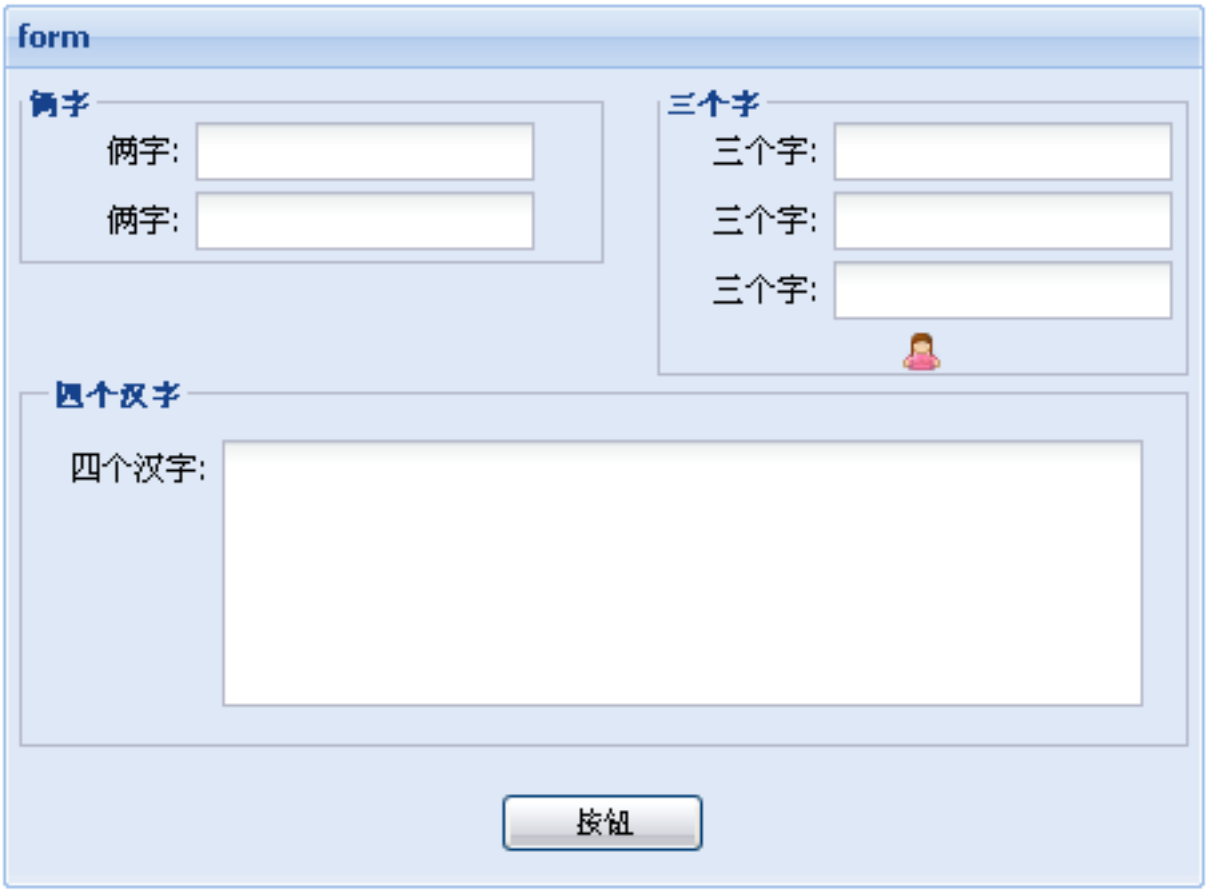
1.x中这种渲染后在dom里胡搞的方法，让我几乎要疯掉了，上帝啊，你自己去看例子吧。lingo-sample/1.1.1/04-25.html。

这个棘手的问题放到2.0里就太简单了，变成FormPanel以后，里边可以用你想的布局，你可以用各种

Panel来妆点表单了，这样我们就搞出个xtype:'panel'好了，里边放上个img，嘿嘿。

```
{
    columnWidth:.5,
    layout: 'form',
    xtype: 'fieldset',
    title: '三个字',
    autoHeight: true,
    style: 'margin-left: 20px;',
    defaultType: 'textfield',
    items:[
        {fieldLabel: '三个字'},
        {fieldLabel: '三个字'},
        {fieldLabel: '三个字'},
        {xtype: 'panel',html: '<center><img src="user_female.png" /></center>'}
    ]
}
```

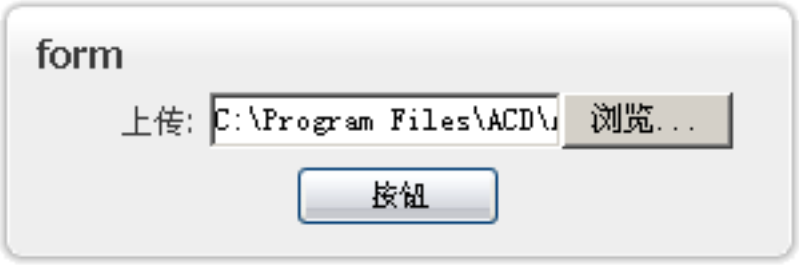
啦啦啦，上眼截图：



简单啊，例子在lingo-sample/2.0/04-25.html呢。

4.9. 还要做文件上传哟

有人问过ext有没有对应文件上传的组件，其实很简单呢，只要把设置inputType: 'file'就好了呢，不过可惜的是，咱们没法修改它的显示样式。



好了，现在需要了解一下，我们如何去做才能让form支持文件上传？

1. 第一，给form添加fileUpload: true。

```
var form = new Ext.form.Form({
    labelAlign: 'right',
    labelWidth: 60,
    fileUpload: true,
    url: 'form2.jsp'
});
```

2. 第二，给form添加一个field，然后把inputType改成file的。

```
form.add(new Ext.form.Field({
    fieldLabel: '上传',
    name: 'file',
    inputType: 'file'
}));
```

现在点击按钮进行提交，就可以上传啦，而且还是不需要刷新页面的ajax提交哟。form2.jsp里我用的是commons-fileupload的提交方式，要是你也想试试的话，把commons-io和commons-fileupload放到lib下，然后就可以用啦。

1.x的例子在lingo-sample/1.1.1/04-26.html里。

2.0的方法差不多啦，fileUpload和inputType就足够啦。例子在lingo-sample/2.0/04-26.html。

4.10. 非想非想，单选框多选框

看看继承结构图就知道，checkbox和radio是跟TextField完全不同的分支，以往的验证啊，布局啊，这里都不怎么能用到了，所以我们要对它们单独讨论了。

4.10.1. 多选呢checkbox

这个，要在form里加上多选框就要用上Ext.form.Checkbox了，可惜的是咱们没法子控制checkbox的样式，所以它还是那么难看。

你也看出来啦，咱们在这里用的是一个fieldset，把三个checkbox框到一起，但是你有没有注意到这三个标签跑到右边去了呢？这可不是咱们可以控制label的位置，而是我们使用了boxLabel，其实他们就只有位置不同啦，你考虑用哪个好了。

```
form.fieldset(
    {legend: '多选', hideLabels: true},
    new Ext.form.Checkbox({
        boxLabel: '多选一',
        name: 'checkbox'
    }),
    new Ext.form.Checkbox({
        boxLabel: '多选二',
        name: 'checkbox'
    }),
    new Ext.form.Checkbox({
        boxLabel: '多选三',
        name: 'checkbox'
    })
);
```

呵呵~为了避免出现前后两个标签，还是把hideLabels设置为true的好。

现在我们可以按提交按钮啦，可惜后台无法区分他们三个，因为默认的value都是on，提交的数据就变成这样：

```
checkbox=on&checkbox=on&checkbox=on
```

想区分这三个checkbox吗？废话，谁不想啊，要不整出三个来干啥？这里我们就需要用inputValue来指定这三个checkbox的值呢。

```
form.fieldset(
    {legend: '多选', hideLabels: true},
    new Ext.form.Checkbox({
        boxLabel: '多选一',
        name: 'checkbox',
        inputValue: '1',
```

```
        checked: true
    }),
    new Ext.form.Checkbox({
        boxLabel: '多选二',
        name: 'checkbox',
        inputValue: '2'
    }),
    new Ext.form.Checkbox({
        boxLabel: '多选三',
        name: 'checkbox',
        inputValue: '3'
    })
);
```

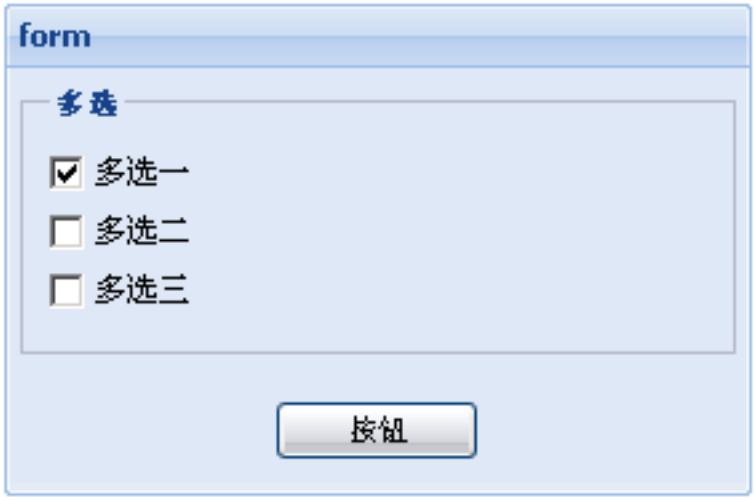
现在我们再提交就会得到这样的结果啦：

```
checkbox=1&checkbox=2&checkbox=3
```

嘿嘿~这不就知道了咱们选中了哪些框框吗？对了，你或许还想知道，咋才能让这几个checkbox一上来就是选中状态呢？嗯，就像在第一个里用的一样，加上checked: true参数，这个我就不单独做例子拉，你试试便知。

1.x的例子在lingo-sample/1.1.1/04-27.html。

2.0里就更简单啦，设置上xtype:'checkbox'，其他一切照旧啦。



2.0的例子在lingo-sample/2.0/04-27.html哟。

4.10.2. 单选呢radio

你要知道，单选可是继承自多选，所以checkbox的一切，radio都有啦，唯一区别是你要是想制作一组radio，让人们每次只选一个，那么就要动些手脚了。

```
form.fieldset(
    {legend: '单选', hideLabels: true},
    new Ext.form.Radio({
```

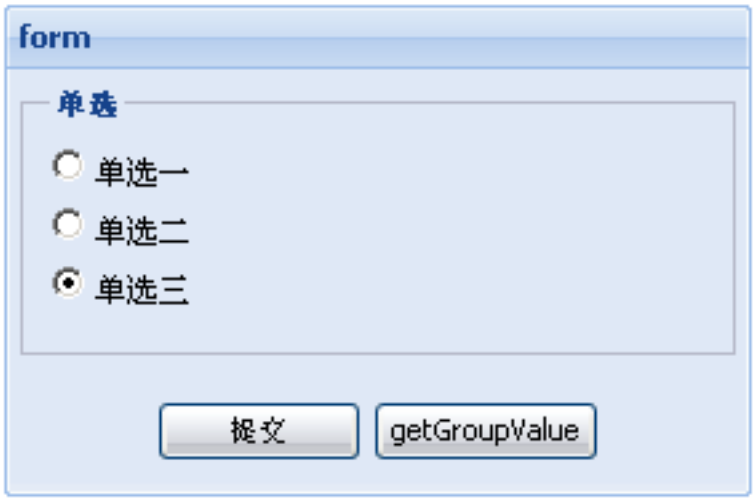
```
        boxLabel: '单选一',
        name: 'radio',
        inputValue: '1',
        checked: true
    }),
    new Ext.form.Radio({
        boxLabel: '单选二',
        name: 'radio',
        inputValue: '2'
    }),
    new Ext.form.Radio({
        boxLabel: '单选三',
        name: 'radio',
        inputValue: '3'
    })
);
```

注意一下那个name，相同name的Radio会放在一组里，这样才会实现只选中一个啦。就像这样呢：

因为name都是一样的，inputValue更显得尤为重要了，要不你根本不知道最后选择的是哪个了。

剔除Checkbox所拥有的呢，Radio还拥有一个自己的函数，getGroupValue()。这个函数直接获得一个分组中选中的那个值，省的咱们一个一个去判断啦，例子中我们专门加了个按钮做演示，你可以去试试看。例子在lingo-sample/1.1.1/04-28.html。

2.0里也是完全一样，改改就好啦，连getGroupValue()这个函数的用法都一样，具体看例子吧：



例子在lingo-sample/2.0/04-28.html。

4.11. 自动把数据填充到form里

添加数据与修改数据是一对兄弟，甚少有只能添加不让修改的，修改的话还是咱们原来的form，不同的就是要把原来的数据显示出来。咱们知道Ext.form.Field都有setValue()函数，可以设置这些控件的数据，但是把这些控件一个一个取出来，再一个一个赋值，还有不少数据要做类型转换的，实在是。。。

如果咱们有个表单：

只需要点一下读取按钮，就可以把里边的数据自动填充上，附带数据类型转换呢？

全自动化是不可能的，但我们至少可以使用一条统一的途径，让繁杂的工作显得中规中矩。当当，请上Ext.data.JsonReader来负责数据读取转换好了。

后台传过来的数据是只有一个元素的json数组：

```
[{
    text: 'textField',
    number: 12.34,
    date: '2008-01-01T00:00:00',
    combo: 1
}]
```

```

```

看看，这里有字符串，数字，日期，对应的reader就要写成：

```
var reader = new Ext.data.JsonReader({},[
    {name: 'text', type: 'string'},
    {name: 'number', type: 'float'},
    {name: 'date', type: 'date', dateFormat: 'Y-m-dTH:i:s'},
    {name: 'combo', type: 'int'}
]);
```

后台发过来的json会在这里转换成对应的数据类型，供form使用的。搞定了reader直接放到form里头，以后什么时候咱们调用form.load()都会借由它做数据转换。

```
var form = new Ext.form.Form({
    labelAlign: 'right',
    labelWidth: 60,
    url: 'form2.jsp',
    reader: reader
});
```

form.load()调用的时候又是用ajax去后台取数了，这里还有一个讲究，如果不给load()传递参数就会使用form的url参数，这个url不是提交数据的网址吗？两个东西都混在一起总觉得那么难看，还是自定一个专门读取数据的地方吧。

```
form.addButton({
    text: '读取',
    handler: function() {
        form.load({url:'04-29.txt'});
    }
});
```

好咯，给load()传递一个url参数，指定读取数据的网址咯，这个网址返回的数据就是咱们上面说的那些了，嘿嘿~实现了哟。

话说form除了load()函数，还专门有个loadRecord()，想想吧，从grid的ds里弄出一行一行的数据可都是record呢，可以直接放到form里的。

喔吼吼吼，1.x的例子放到lingo-sample/1.1.1/04-29.html了，2.0用法相仿，在lingo-sample/2.0/04-29.html。

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。

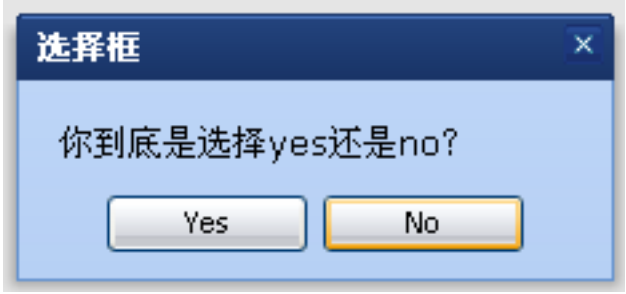
5.2.1. Ext.MessageBox.alert()

```
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()

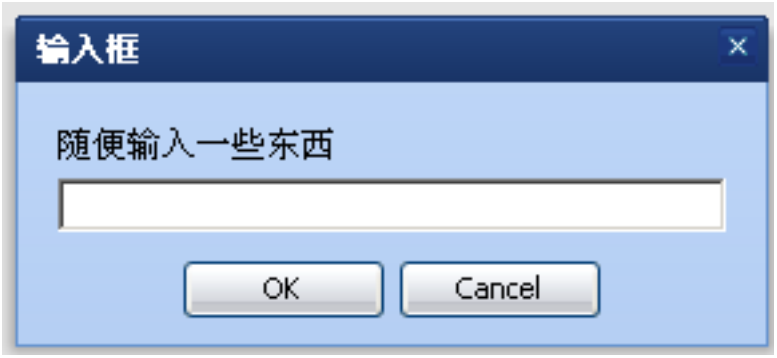
```
Ext.MessageBox.confirm('选择框', '你到底是选择yes还是no？', function(btn) {
    alert('你刚刚点击了 ' + btn);
});
```



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);
});
```

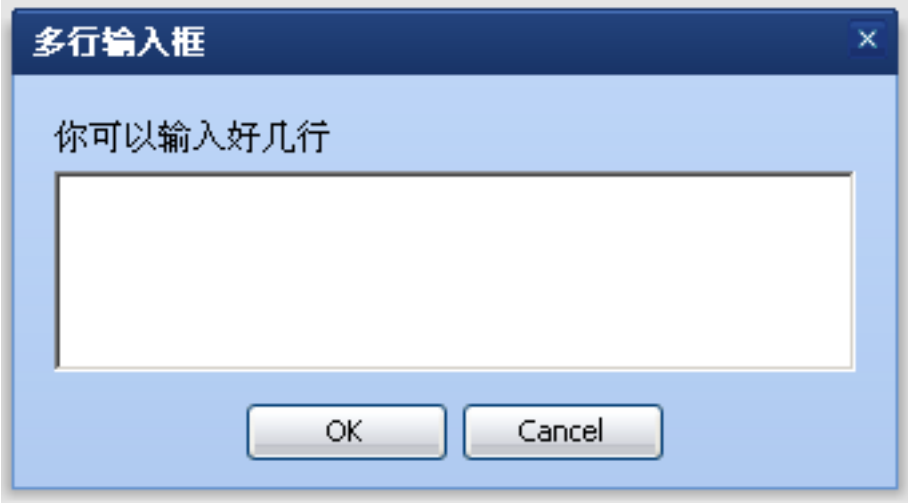


随便输入几个字，然后点按钮，它会告诉你输入了些什么东西

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);
    }
});
```

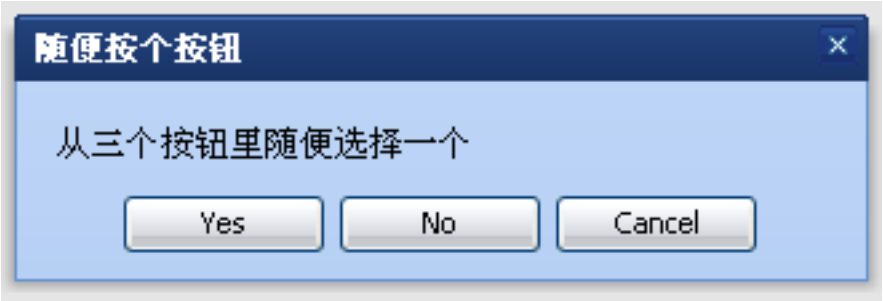


其实只需要show，我们就可以构造各种各样的窗口了，title代表标题，msg代表输出的内容，buttons是显示按钮，multiline告诉我们可以输入好几行，最后用fn这个回调函数接受我们想要得到的结果。

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，里边的这个参数绝定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。实际上的配置缺非常简单，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 。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相应的改变。

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。

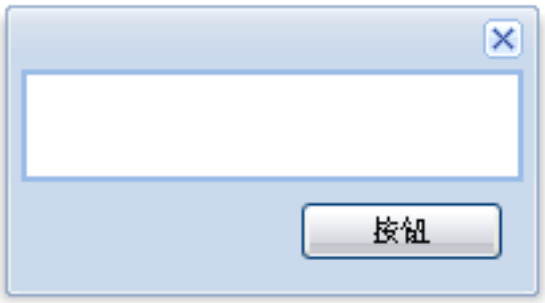
如果不想让用户使用右上角的小叉叉关闭窗口，就设置上closable: false，如果不想让用户把窗口拖来拖去，就设置draggable: true。

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？

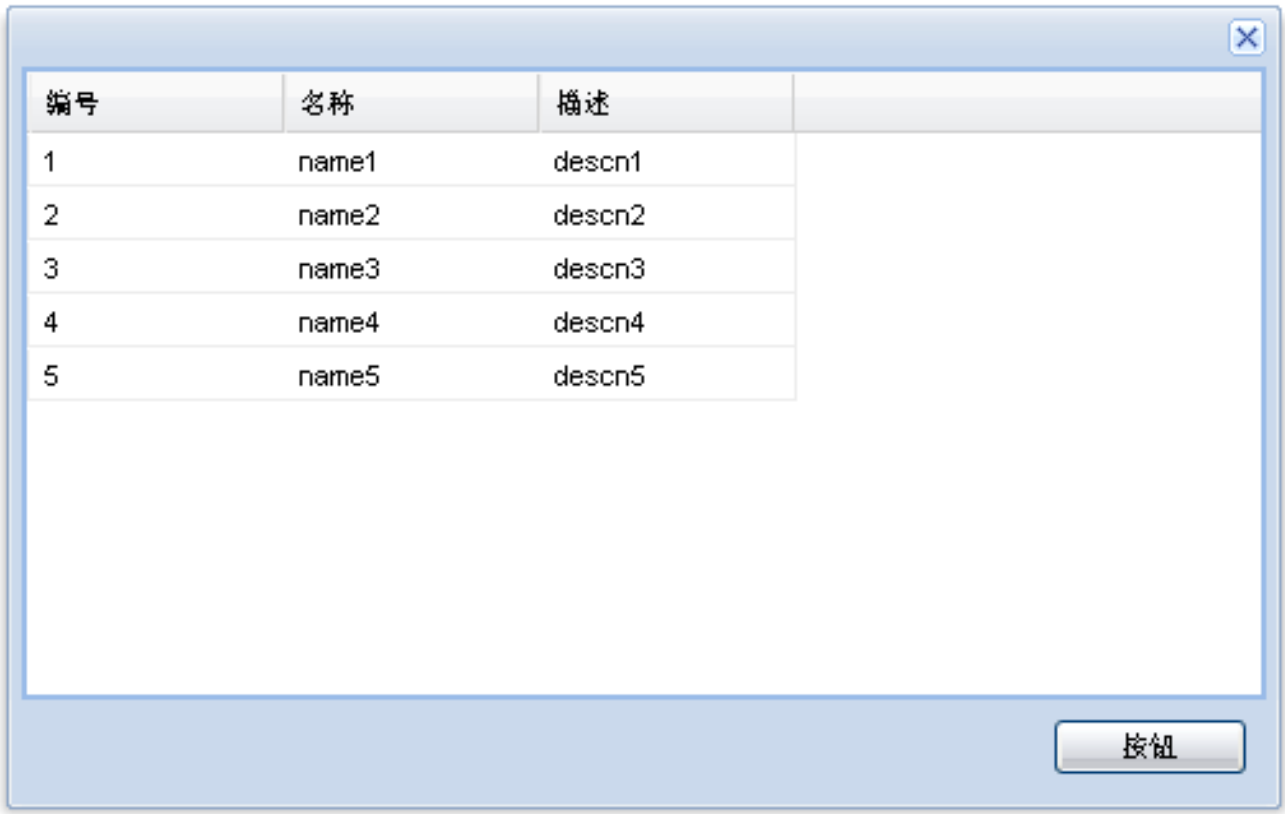
例子超简单，见lingo-sample/2.0/05-02.html。

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。

1. 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。

2. 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>
```

3. 第三，如果还不知道怎么把grid放进window里，我给你看下代码。

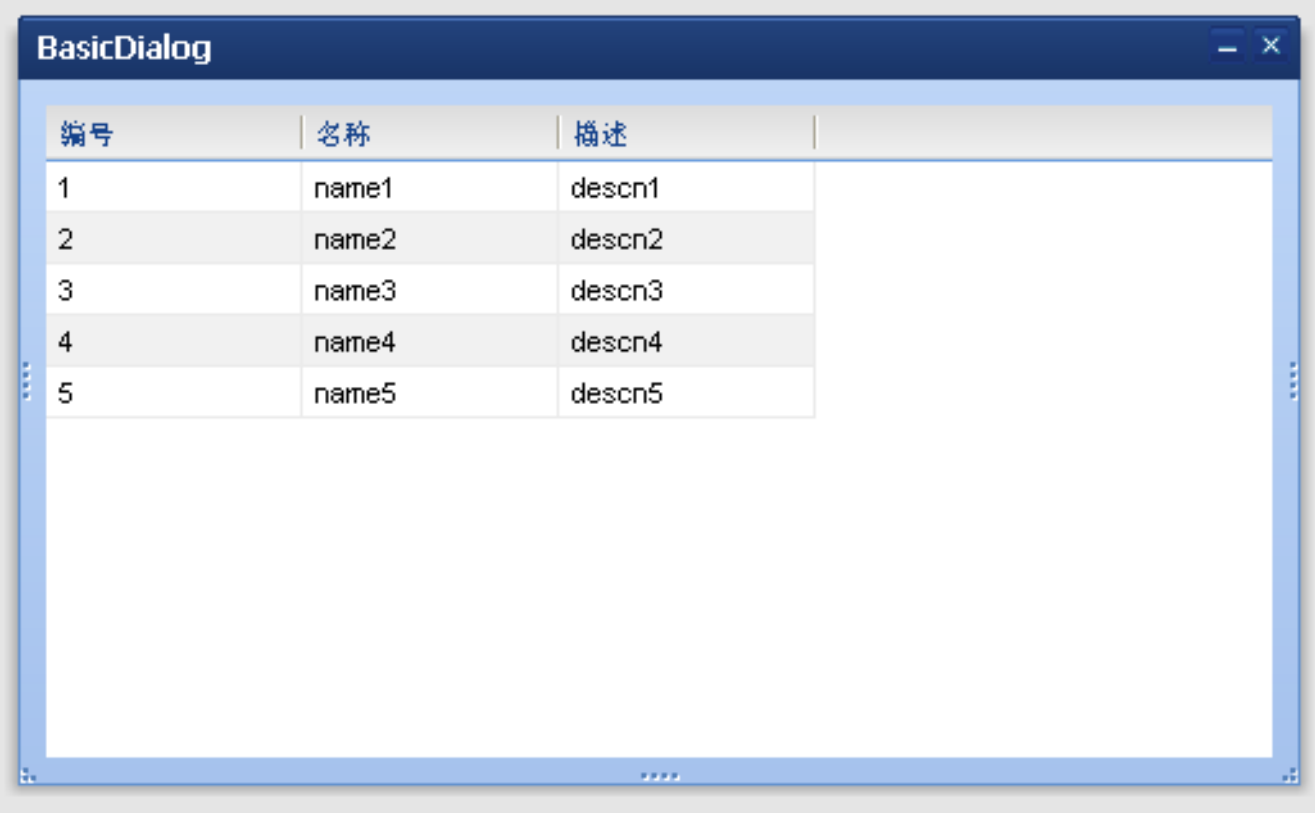
```
var win = new Ext.Window({
    el:'window-win',
    layout:'fit',
    width:500,
    height:300,
    closeAction:'hide',

    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。敬请大家继续关注。

5.4.3. 1.x里的叫做BasicDialog

2.0里的Ext.Window完全取代了Ext.BasicDialog和Ext.LayoutDialog，现在咱们来看看1.x中是怎么实现的。




```
var dialog = new Ext.BasicDialog("dialog", {
    width:500,
    height:300,
    shadow:true,
    minWidth:300,
    minHeight:250,
    title: 'BasicDialog',
    proxyDrag: true,
    modal: true
});
dialog.show();
```

dialog配置部分比较简单，设置标题，宽度和高度。proxyDrag控制拖拽效果，true的时候你在拖拽的时候会看到本体还在原地，你拖得是一个鬼影子。

modal:true则是著名的模式对话框，用上了这个家伙，让我们在弹出对话框以后就不能再操作其他部分，是个相当难得的效果呢。

创建好以后，用show()函数显示对话框，hide()隐藏它，隐藏以后还可以用show()再次显示，省去了重新构建的性能消耗。哈哈~就是我从草丛里跳出来啦，我又藏起来啦，我又跳出来啦，我又藏起来啦。快啊。

下面要向dialog里放表格了，我们不能再去指定div了，dialog的臭脾气让我们只能渲染它提供给我们的东西：

```
var grid = new Ext.grid.Grid(dialog.body, {
    ds: ds,
    cm: cm
});
```

我们使用dialog.body来指定grid将渲染的元素，这样表格就跑到dialog里头去了。比较来说，还是2.0中的window更灵活呀。

1.x的例子在lingo-sample/1.1.1/05-02.html。

5.4.4. 把form放进对话框里

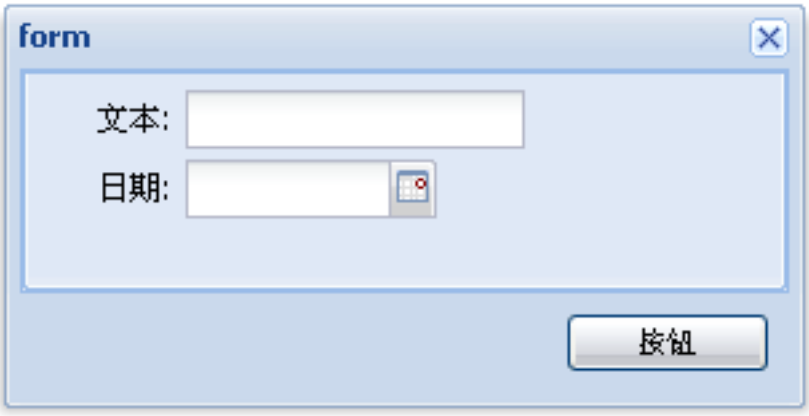
托布局的福，2.0里放Form也差不多

因为2.0里各种的panel都可以与布局对应，FormPanel直接可以放进window中，这里我们现在就准备一个简单的form做个试验。

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 50,
    frame: true,
    defaultType: 'textfield',
    items: [{
        fieldLabel: '文本',
        name: 'text'
    },{
        fieldLabel: '日期',
        name: 'data',
        xtype: 'datefield'
    }]
});
```

其他配置基本不变，去掉了title设置，这是因为window自己有了标题部分，width属性也去掉了，因为把form交给window布局管理，在window中设置了layout:'fit'，最终form会按照window的大小进行伸缩，不需要额外设置宽度了。

window的配置与上面相似，只是items:[grid]改成items:[form]，这样我们的工作就完成啦。看看截图先。



例子在lingo-sample/2.0/05-04.html。

1.x中的例子也是与上方相仿的。

```
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(dialog.body);
```

form的创建和组装都是老方法了，只是渲染的时候必须使用form.render(dialog.body)，把表单画在对话框的主体部位，但这样总给人拼凑的感觉，与2.0中灵活布局的方式逊色了很多。

1.x的例子在lingo-sample/1.1/05-03.html。

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。

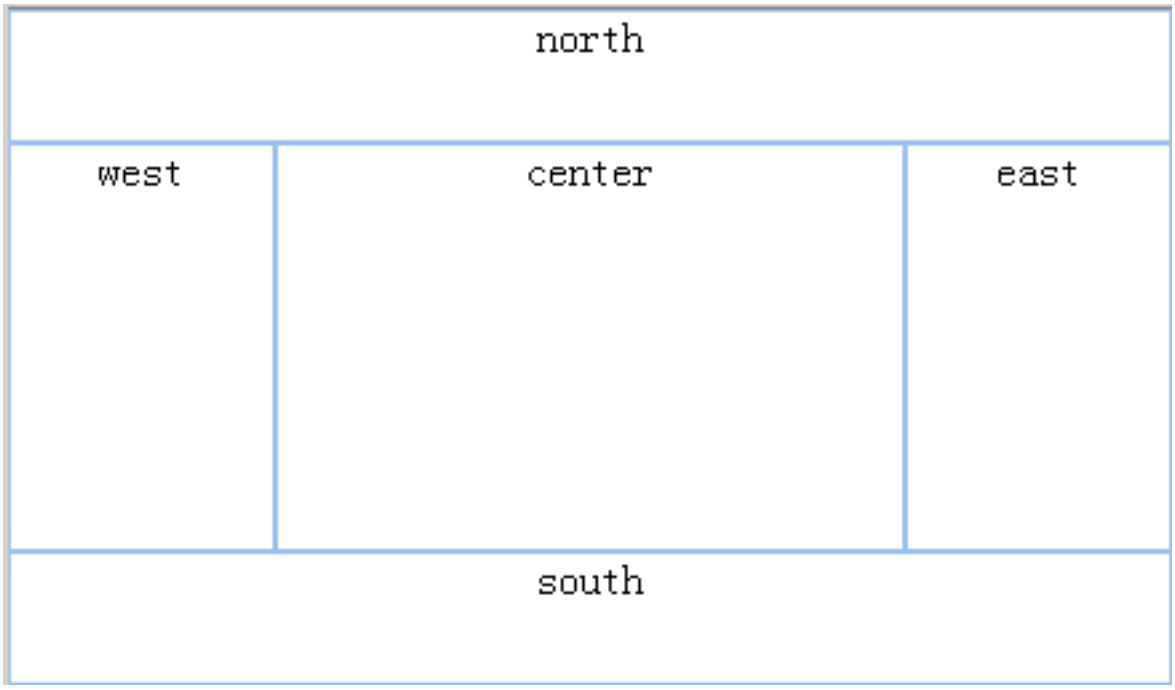
对布局很是不熟，至今为止，也是一直在抄土豆demo里的BorderLayout，frank的deepcms ProjectTracker里的ViewPort布局而已，不过有了布局，咱们不用再去摆弄frameset了，只需要div就可以做成端端正正的布局，嗯，只这一点儿就吸引了多少眼球啊。

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。

6.3. 嗯，不如再看看附加效果

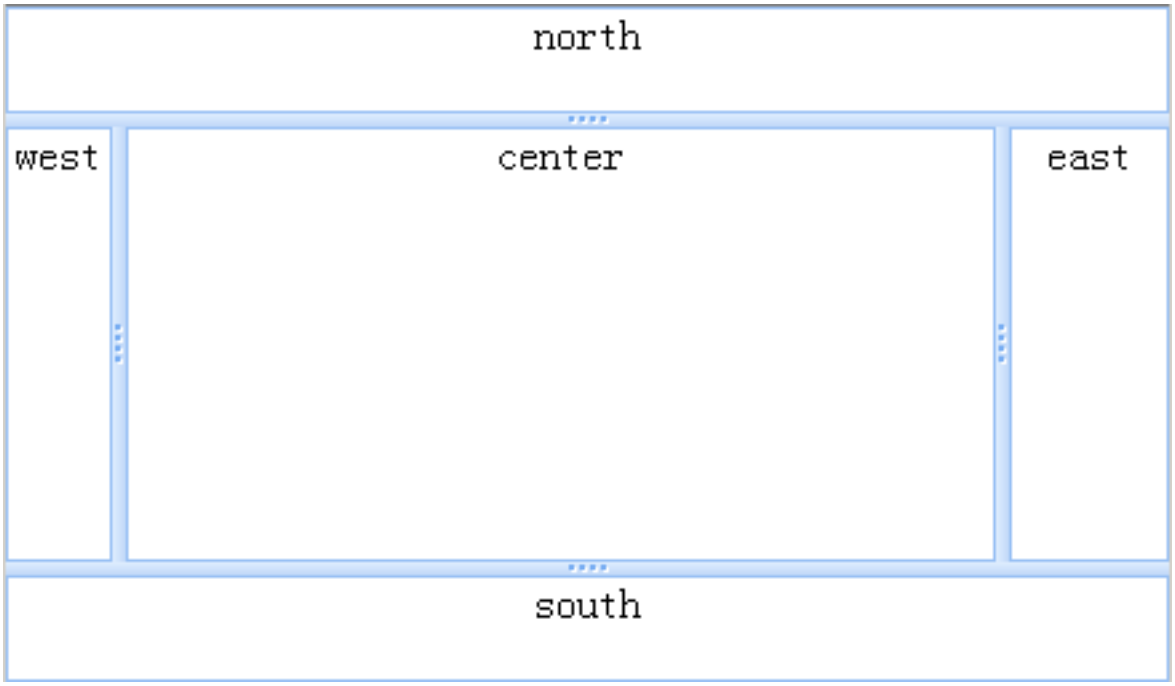
其实，即使只能在不同浏览器，把一个窗口切成相同的部分，也是足够了，不过ext带给我们的不仅仅是如此，让我们再看看其他部分吧。

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: {
    }
});
```

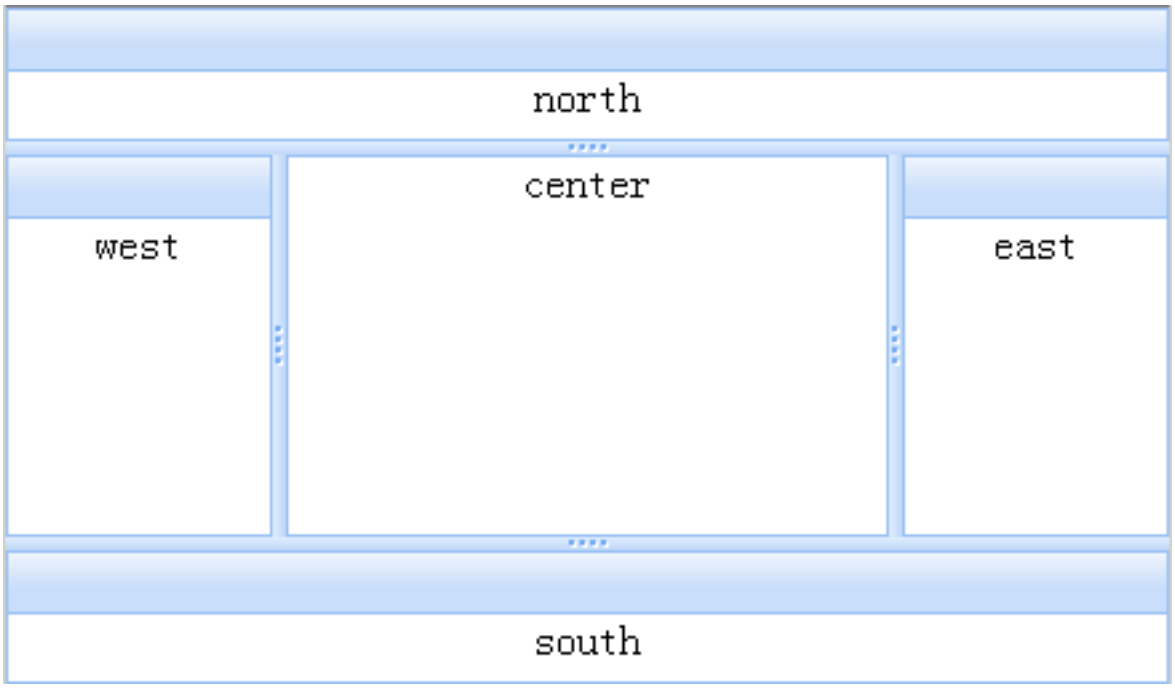
minSize和maxSize让用户只能在我们决定的范围内修改区域的大小，既不会太大，也不会太小。用户的行为受限，也减少了他们抱怨的机会。嘿嘿，一切尽在掌握中。

例子见lingo-sample/1.1.1/06-02.html

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: {
    }
});
```

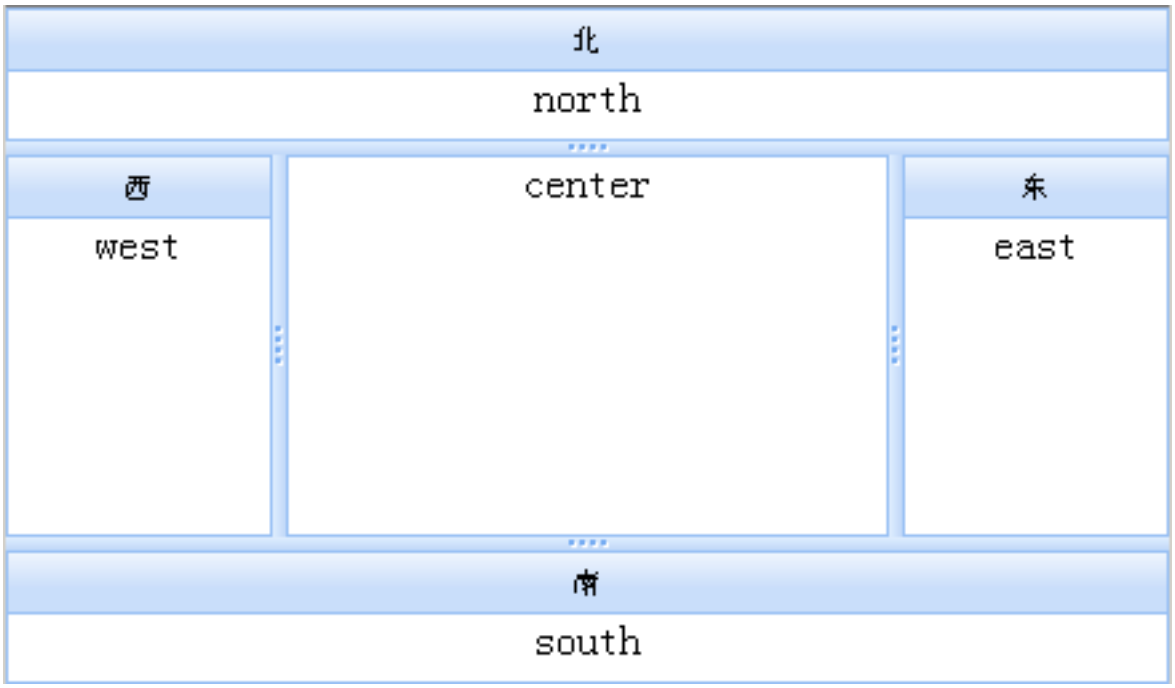
在上面的基础上，我们加入了titlebar，然后看到的就是这幅情景。



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();
```

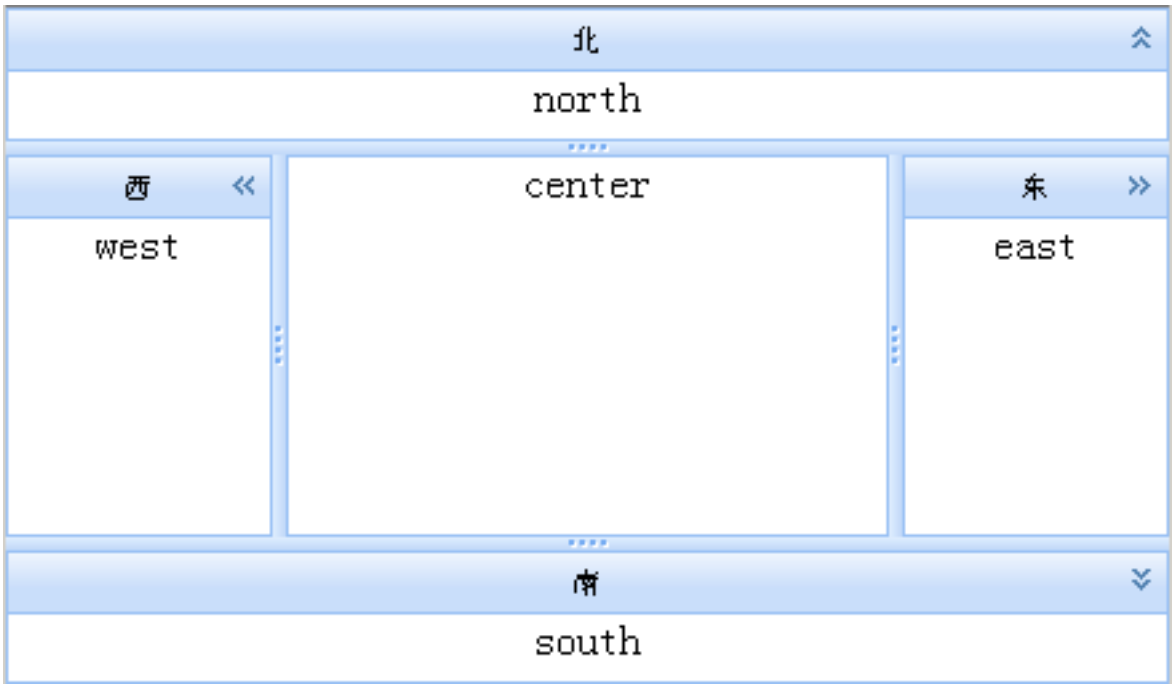
经过这些改变，整个布局就变成了这个样子。



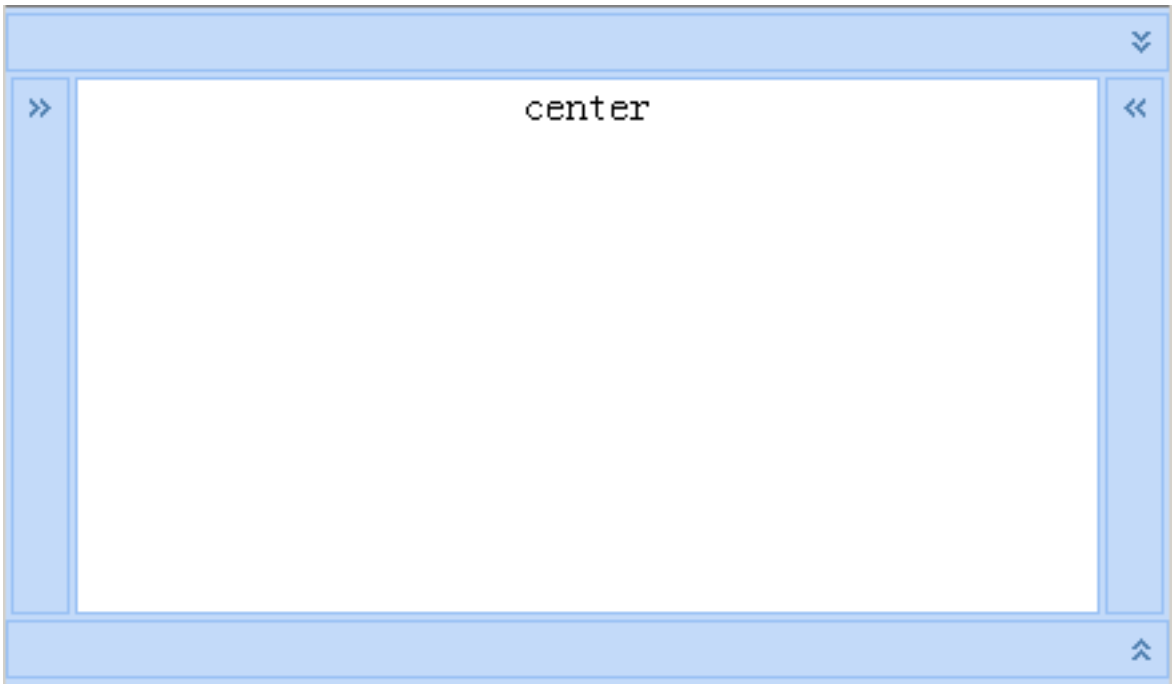
例子是lingo-sample/1.1.1/06-03.html。

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

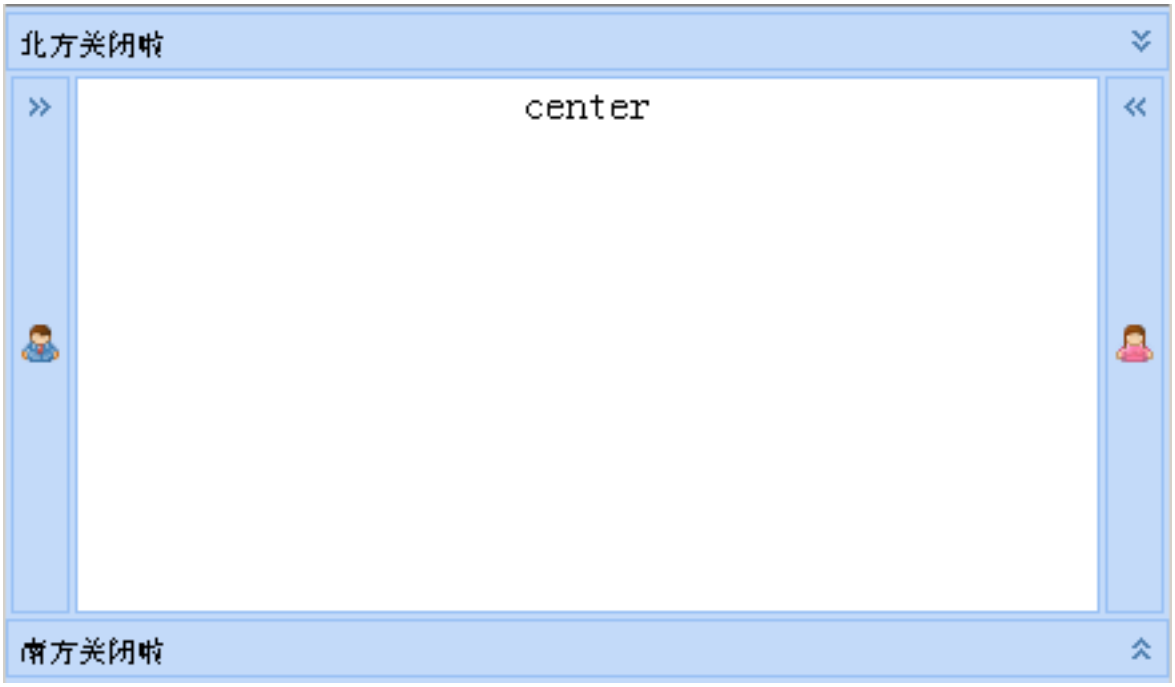
分有效，因为west和east是垂直的，似乎没有办法让文字旋转90度显示，所以我们需要其他方法。

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。



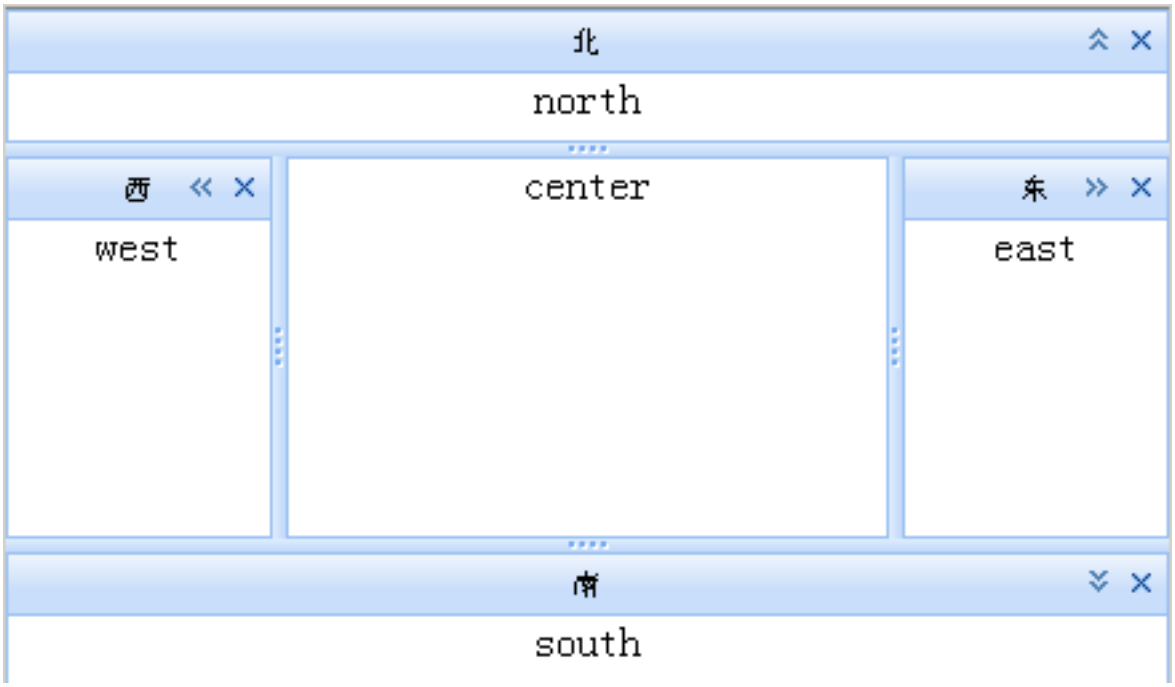
嘿嘿，有意思吧。代码都在lingo-sample/1.1.1/06-04.html里呢，你也试试吧。

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。

6.4. 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>
```

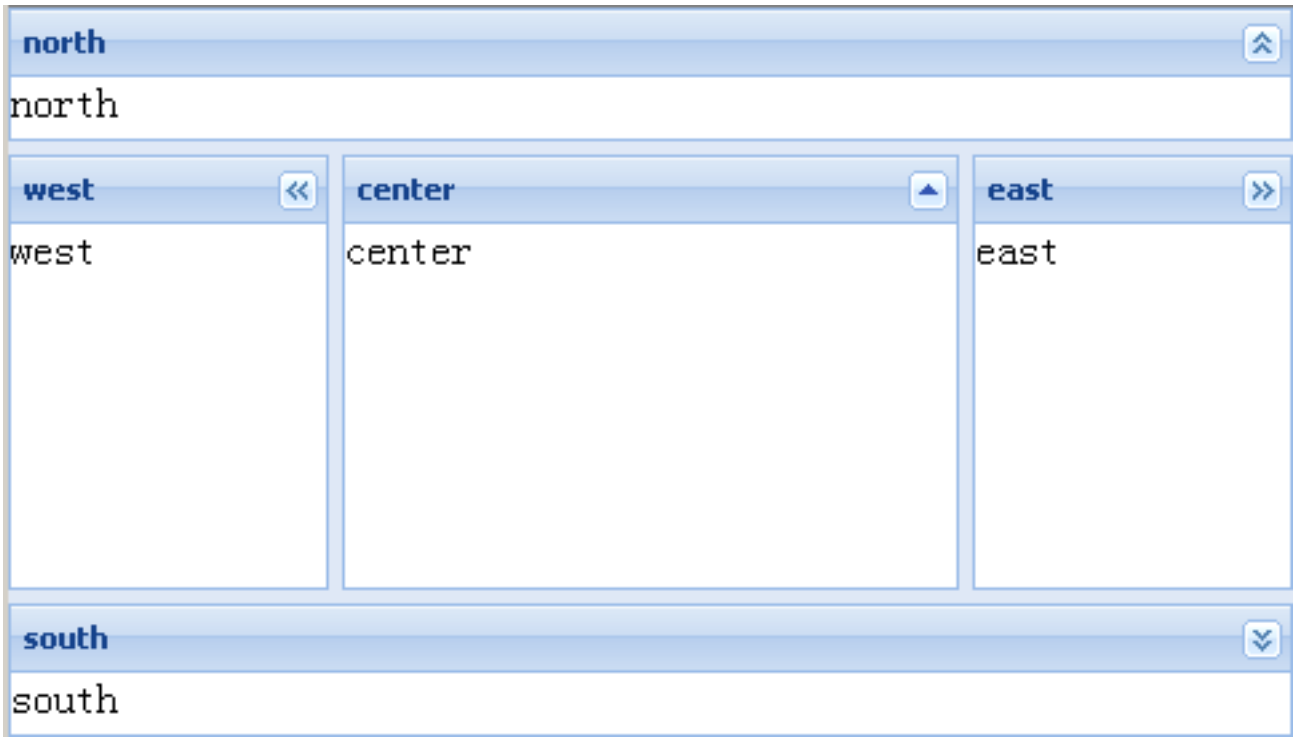
剩下的就是代码了，还是那句话，我们想要在一个地方配置好所有东西，不想东奔西跑，说不定丢了什么也不知道，维护多个地方的配置简直是噩梦，2.0啊，我们崇拜你。

```
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。



例子在lingo-sample/2.0/06-01.html下，看看呗。

6.5. 脑袋上有几个标签的tabPanel

就像这样，你点哪个标签就出来哪个内容，实际上他会作为其他内容的容器，把文字呀，控件呀，不管是什么东西都放进去就好了，虽然目前咱们只加了几个字，不过这也算是伟大的一步。

```
var tabs = new Ext.TabPanel(document.body);

tabs.addTab(Ext.id(), '标题1', '内容1');
tabs.addTab(Ext.id(), '标题2', '内容2', true);

tabs.activate(0);
```

就像这样，创建一个Ext.TabPanel，第一个参数就指定了它渲染的元素，我们直接把它放到html页面的body部分。

第二步使用addTab函数向TabPanel里添加两个标签，同时添加的还有对应的内容。看一下它里边的参数。

1. 第一个是这个tab对应的id，我们使用Ext.id()函数生成唯一的id值。

2. 第二个是标签上显示的标题。

3. 第三个参数是点击标签后显示的内容，实际上这里可以放置任何html，你可以把所有想放的东西放

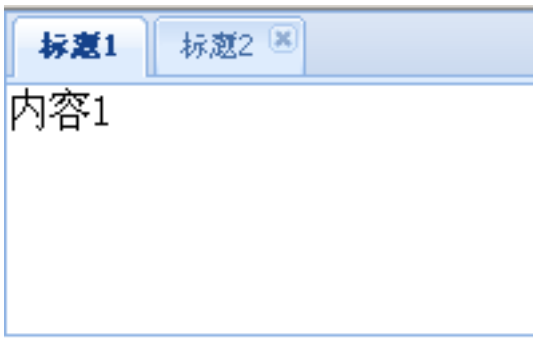
到里边了。

4. 最后一个参数是一个boolean值，true或false，这个参数决定了生成的标签是否可以手工关闭，默认是false，不会显示关闭按钮，当我们手工设置为true后，标签上显示一个关闭按钮，点击关闭按钮后，标签和对应的内容被关闭掉。

在添加完tab后，调用了activate()函数让我们指定的tab变成激活状态，参数是一个从0开始的索引值，activate(0)就是激活第一个标签。

1.x的例子在lingo-sample/1.1.1/06-06.html。

2.0里下的更帅气呢。



实际上布局相关的layout和panel都脱胎换骨了。

```
var tabs = new Ext.TabPanel({
    renderTo: document.body,
    height: 100
});

tabs.add({
    title: '标题1',
    html: '内容1'
});
tabs.add({
    title: '标题2',
    html: '内容2',
    closable: true
});

tabs.activate(0);
```

现在这些参数统一放到大括号里，这样我们更容易理解这些参数的含义，比如renderTo就是渲染的元素，height就是生成的高度。

然后看看title是对应标题，html是对应内容，用closable确定这些标签是否能关闭。

tab.activate(0)依然是激活第一个tab的意思。

2.0里的例子在06-02.html。

接下来呢？是不是要给添加的tab里边放一些东西呢？做两个按钮，一个按钮新建里边有grid的tab，另一个放layout如何？

其实只要知道一点就好了，2.0的布局里可以随意控制这些的。比如我们使用items:[grid]指定在tab里放上生成好的GridPanel，但记得要放一个layout:'fit'，这个布局形式可以让内部的GridPanel随着外部而自然展开，要不你得到的只是一个高度为零的东东，什么也见不到。

加入表格的代码大致如下：

```
var tab =  tabs.add({
    title: '表格' + id,
    closable: true,
    layout: 'fit',
    items: [grid]
});
tabs.activate(tab);
```

上边自动生成一个标题，设置closable:true，让我们可以手工关闭这个tab。layout:'fit'让里边的自动填充整个空间，最后items:[grid]告诉我们这里边方的是grid，这个grid当然是我们之前做好的。最后调用的tabs.activate(tab);为我们激活刚刚添加的这个tab，这样我们就可以直接看到结果了。

在tab里放置Panel的方法大致相同，说真的，默认添加的tab都是Panel型的，何苦再搞一个Panel呢？不过既然layout可能是'fit'，也就有可能是其他值，试验一下其他布局也不错呢。

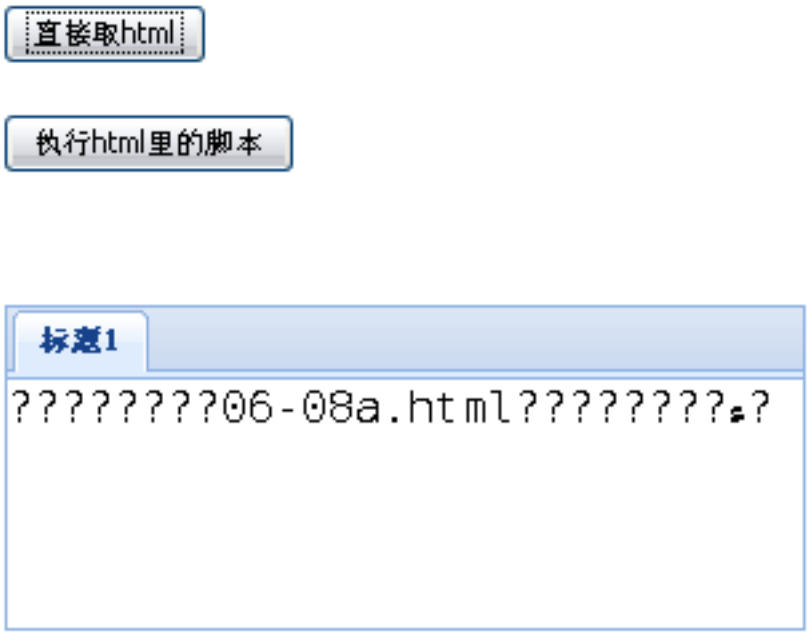
2.0的例子在lingo-sample/2.0/06-07.html。

单纯比较的话，1.x中的TabPanel的确很难看，标签多的时候也不能进行滚动，由此也可以看出2.0比1.x的优越了，1.x每次都要获得bodyEl，再使用render()渲染，2.0则都可以使用布局搞定，呵呵~。

1.x的代码在lingo-sample/1.1.1/06-07.html。

又提出一个问题，如果我们需要的内容是从后台得到的html该怎么做呢？如果这个html里有自己的js脚本，是不是可以在加载完成后执行这些脚本呢。

实际上在2.0中可以通过设置autoLoad来实现呢，这些tab默认是惰性加载，只在激活的时候采取用ajax去后台取数据：



看到乱码了吧？因为ajax需要utf-8，咱们的中文页面用了gbk编码，所以就有问题了，想正常显示中文你要把那些文件改成utf-8编码哟。

这部分代码倒是简单，把html去掉，换成autoLoad即可：

```
tabs.add({
    title: '标题1',
    autoLoad: {url: '06-08a.html'}
});
```

不过这样做的话，06-08a.html里如果有js脚本，也不会执行的，想执行包含在html里的脚本，还需要加一个参数，scripts：

```
tabs.add({
    title: '标题2',
    autoLoad: {url: '06-08b.html', scripts: true},
    closable: true
});
```

啦啦，现在06-28b.html里的脚本就可以执行啦。现在lingo-sample/2.0/06-08.html里有两个按钮，第一个按钮添加没脚本的06-08a.html，第二个按钮添加有脚本的06-08b.html。

用1.x的兄弟就没有这么幸运啦，没有autoLoad参数可用，只能在创建tab之后，手工调用setUrl()函数：

```
var tab1 = tabs.addTab(Ext.id(), '标题1', '内容1');
tab1.setUrl('06-08a.html', null, true);
```

setUrl()函数的第一个参数是读取的url地址，第二个实在不知道是做什么用的，第三个参数的名字叫loadOnce，如果是false会在每次激活tab的时候去后台读取数据，反之只有第一次更新。

要想让执行html里的js脚本就更麻烦啦，setUrl()函数也没提供直接的方法，必须从tab里提炼出UpdateManager，然后调用update()函数来触发ajax获得咱们想要的东西。

```
var tab2 = tabs.addTab(Ext.id(), '标题2', '内容2', true);
var um = tab2.getUpdateManager();
um.update({
    url: '06-08b.html',
    scripts: true
});
```

update函数的参数跟2.0就一样啦，设置url和scripts就可以支持html中的脚本啦。

1.x例子在lingo-sample/1.1.1/06-08.html中。

6.6. 让布局复杂一点儿

千言万语不如一张图片。

咳咳，其实这个布局比上边那个还要简单，去掉了east方向的Panel，north和south方向布局都不能收缩，也不能改变大小了。west方向变成了树形，center再次使用border布局分成两部分，其中的north部分放了一个grid，center部分放了一个form。嗯，这个form比较难看。

其实这些tree啊，grid啊，form啊，都是从前几章拣来的，使用布局组合起来之后就变成这样的效果。当然，也是需要稍稍做一些调整的啦。

反其道而行之，先看看最后布局组装时的代码：

```
// layout start
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'north',
        contentEl: 'north-div',
        height: 80,
        bodyStyle: 'background-color:#BBCCEE;'
    },{
        region: 'south',
        contentEl: 'south-div',
        height: 20,
        bodyStyle: 'background-color:#BBCCEE;'
    },tree,{
        region: 'center',
        split: true,
        border: true,
        layout: 'border',
```

```
        items: [grid,form]
    }]
});
// layout end
```

看到了吧？上下都没有变动，原来west的部分直接放上了tree，center部分先设置layout:'border'使用边界布局，然后在items里分别配置上grid和form。

分块讲解吧：

1. 先看这个north和south，上下部分就一起讲了。现在只有这两部分需要contentEl来指定html中的显示内容，所以

```
<div id="north-div">标题栏：6.6 让布局复杂一点儿</div>
<div id="south-div">状态栏：Copyright by www.family168.com</div>
```

   布局里只设置了高度和背景颜色，可以通过bodyStyle设置整体的css样式呢。

2. 接下来是第二简单的左侧树形了，ViewPort里看不出什么蹊跷来，让咱们找到TreePanel定义的地方看看有什么问题吧。

```
var tree = new Ext.tree.TreePanel({
    loader: new Ext.tree.TreeLoader({dataUrl: '03-04.txt'}),
    title: 'west',
    region: 'west',
    split: true,
    border: true,
    collapsible: true,
    width: 120,
    minSize: 80,
    maxSize: 200
});
```

   嗯，一下子多了这么多参数，结果就是tree把以前写到ViewPort里定义大小啊，边界啊，是否可拖拽啊，标题啊，是否可折叠啊，这些参数都转移到TreePanel里来了。

   或许你在疑惑。Why，为什么可以把这些东西直接放到里边来。说来话长啊，这个ViewPort里的各个部分都是Panel，这个Panel包含的范围可广啦，像什么TreePanel啊，GridPanel啊，FormPanel啊，从名字看就知道他们都是一家子，所以这个Panel可以放进去的地方，他们也都可以放进去，Panel的那些配置参数，他们也都可以用，咱们以前没指定具体用什么，ViewPort就默认都用Panel了，现在咱们指定上TreePanel了，ViewPort也没有意见，但是原来的参数就要放到TreePanel中了。

   就是这样，咱们的TreePanel就代替了原来干枯瘦瘪的Panel，占据了西方重地。

3. 上古人类以青(紫)、黄、赤(红)、白、黑五色作为主要的颜色，后来传统国教道家将其配上了方位：青(紫)色指东，赤(红)色指南，白色指西，黑色指北，四向合一，皆围中央无极土，就是黄色。“黄”字与“皇”字谐音，因此黄色的方位，就在东南西北四向的中央，因此，具有神圣不可侵犯的尊严，成为无上正统的象征。

正所谓青龙白虎朱雀玄武，皆围中央无极土。这个center的地位真是举足轻重呀，现在可就要对它动刀了。这一刀拦腰斩玉带，直把它斩成两段。

说实在的，真是拖了2.0里布局的洪福了，任何一个Panel里设置上layout:'border'，就可以再次分成五块，五块里再用Panel，用region指定各自所处的位置，如果里边的Panel再设置上layout，嘿嘿~子子孙孙无穷尽矣。

嘿嘿~，让我们看看中间朝上的grid。

```
var grid = new Ext.grid.GridPanel({
    ds: ds,
    cm: cm,
    title: 'center-north',
    region: 'north'
});
```

呵呵~看里边多了title设置布局上的标题，region告诉是放在上边呀。不过是在center的上边也就变成中上啦。

然后是中下的表单哟，多个title标题呢，region设置中间的中间了。

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

    title: 'center-center',
    region: 'center',

    items: [{
        fieldLabel: '文本框'
    }],
    buttons: [{
        text: '按钮'
    }]
});
```

啦啦，感觉不错吧？2.0的复杂布局已经尽在掌握中啦。2.0的例子在06-09.html中哟。

在1.x中实现这么复杂嵌套布局一定要使用Ext.NestedLayoutPanel啦，看好那块要用嵌套的地方，搞进去就好啦。

```
// 让布局在我们安排了所以部分之后，再显示
var innerLayout = new Ext.BorderLayout('center-div', {
    north: {
        initialSize : '26px',
        autoScroll  : false,
        split       : false
```

```
    },
    center: {
        autoScroll  : false,
        split       : true,
        titlebar    : false,
        collapsible : true,
        showPin     : true,
        animate     : true
    }
});
innerLayout.add('north', new Ext.ContentPanel('center-north'));
innerLayout.add('center', new Ext.ContentPanel('center-center'));
```

看啊，就在原来的center-div上边制造个BorderLayout，然后呢，里边放上north和center的定义。然后呢，像以前一样在里边添上ContentPanel。然后呢，就可以用innerLayout创建个NestedLayoutPanel，放进咱们最大的这个Layout里。这样就获得了复杂布局。

```
mainLayout.add('center', new Ext.NestedLayoutPanel(innerLayout));
```

html里也要做一点儿相应的修改，咱们要添加innerLayout需要的两个部分，就像这样的：

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

唉，复杂吧？1.x的例子见lingo-sample/1.1.1/06-09.html吧。

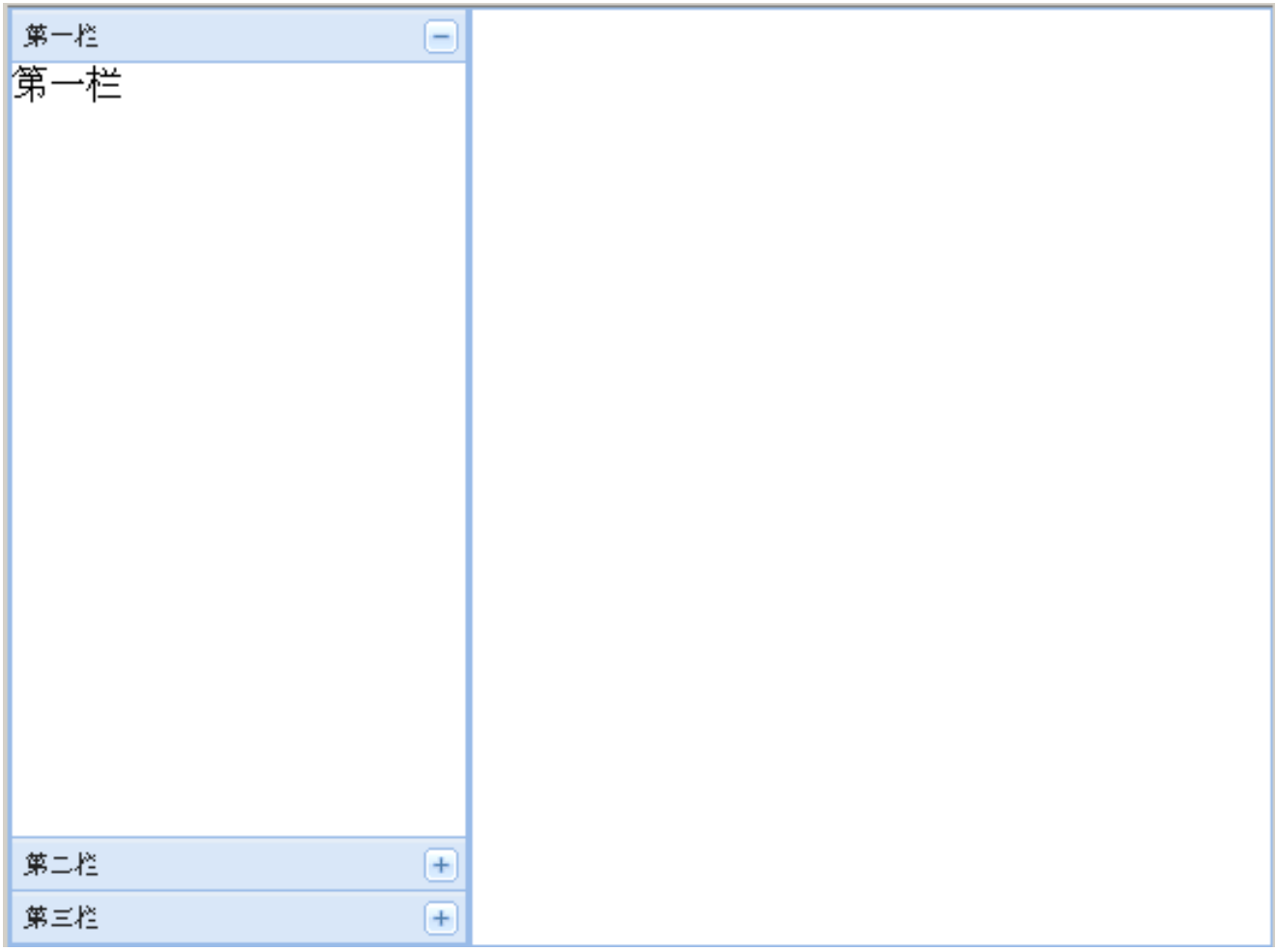
6.7. 向诸位介绍一下2.0里各具特色的布局

那个，BorderLayout已经用烂了，就不说了，FitLayout就是让里边的东东填满整个布局，也没啥讨论的价值，FormLayout是专门为表单准备的，用了这个才会自动以fieldLabel : input的形式显示那些控件，ColumnLayout就更没意思了，从左到右横排而已。至于AnchorLayout和AbsoluteLayout简直是去直接设置x,y坐标去布局了，唉。

想了半天，还是觉得只有accordion，card，table三个有讨论价值，下面咱们正好一一道来。

6.7.1. accordion就是QQ那样的伸缩菜单

传说中的accordion，在1.x时代只能通过扩展的方式加入这项功能，到了2.0时代，accordion作为默认布局的一部分，什么时候想用了，直接该上layout:'accordion'就好了，其他部分基本无需改动，额滴神啊，他怎么做到的啊？



我们利用ViewPort制作的只有west和center两个部分的BorderLayout，west部分放的就是AccordionLayout了，看看实现代码：

```
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'west',
        width: 200,
        layout: 'accordion',
        layoutConfig: {
            titleCollapse: true,
            animate: true,
            activeOnTop: false
        },
        items: [{
            title: '第一栏',
            html: '第一栏'
        },{
            title: '第二栏',
            html: '第二栏'
        },{
            title: '第三栏',
            html: '第三栏'
        }]
    },{
        region: 'center',
        split: true,
        border: true
    }]
```

```
});
```

设置了layout:'accordion'后，再使用items添加上三个元素，记得每个子元素里都要加上title参数，accordion可没提供默认的标题，不设置是一定会出错的。

跟布局有关的配置参数都写到layoutConfig里了，来，坐，看看里边都有些啥呢？

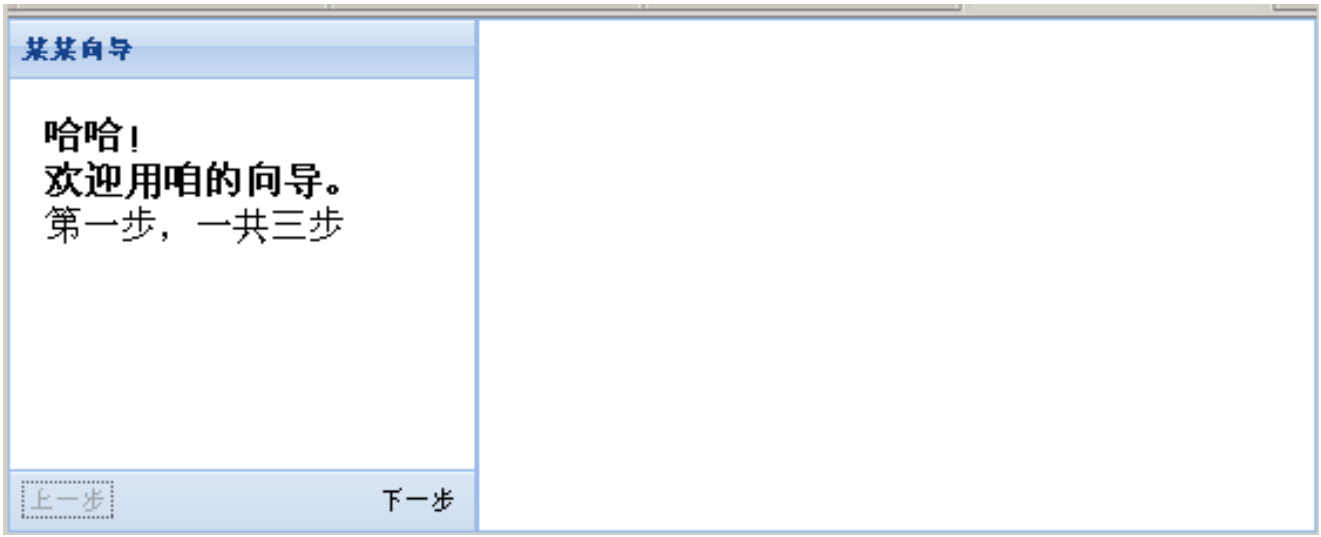
1. titleCollapse，默认是就是true，点击标题就可以折叠子面板，要是你非要设置成false，就只能点击title右边的图标折叠子面板咯。
2. animate，展开折叠的时候是否使用动画效果呢？我一般都设置上true的，为了好看呗。
3. activeOnTop，这个挺有意思的，默认值是false，进行展开折叠后，子面板的顺序不会改变，如果改成true，就会随着展开折叠改变顺序，咋改变呢，就是总把展开的子面板放在最上头啦。第一次见我还意思程序出问题了呢，呵呵~。

只有2.0的例子，lingo-sample/2.0/06-10.html。

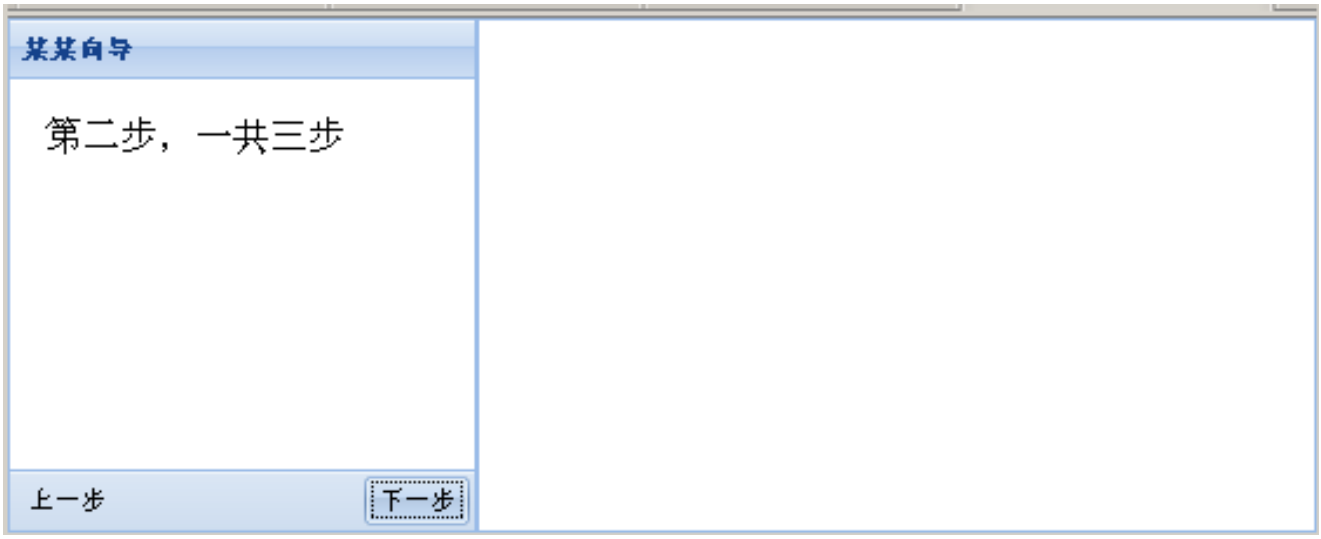
6.7.2. CardLayout？其实就是Wizard啦。

Sorry。不应该用鸟语撒，其实就是向导。啥是向导呢？嗯，真难给它下个定义，还是直接看实物吧。

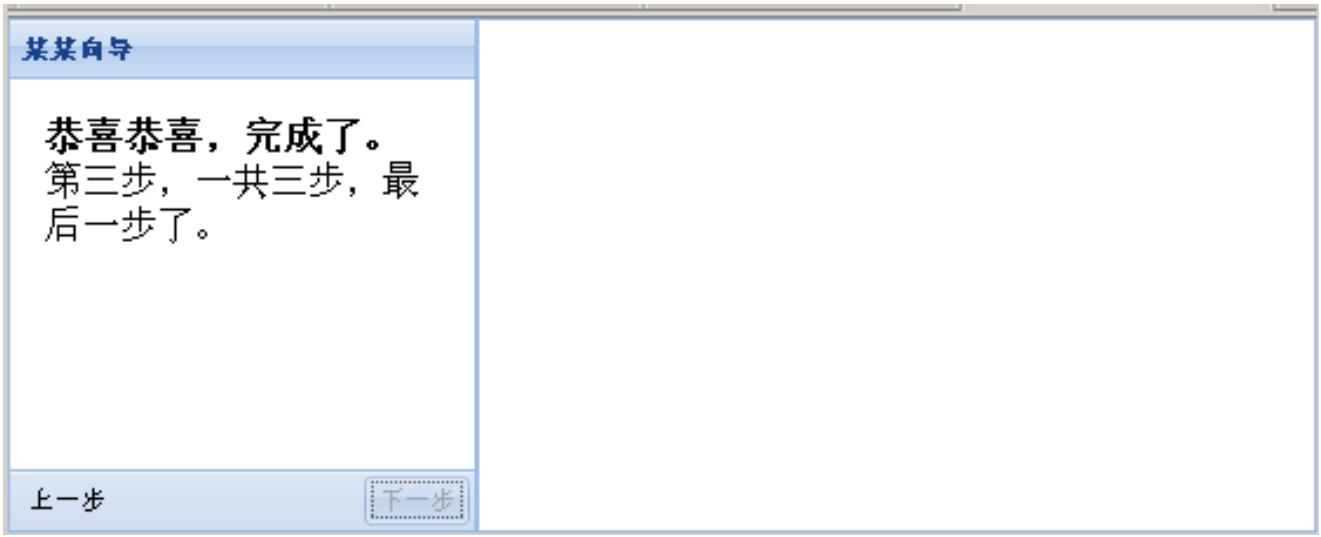
打开页面，映入眼帘的就是俺们的某某向导啦：



好，点下一步，看到没？看到没？画面变化啦。



再点下一步，就到头啦。



到了最后还可以按上一步回到刚才看过的面板，咋样，这就是向导啦。

CardLayout呢，就是你虽然给它配置了几个子面板，但它每次只显示其中一个，好像幻灯片呢，刷拉刷拉的。布局倒是很简单的，只要改一下layout:'card'而已。

```
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'west',
        id: 'wizard',
        width: 200,
        title: '某某向导',
        layout:'card',
        activeItem: 0,
        bodyStyle: 'padding:15px',
        defaults: {
            border:false
        },
        bbar: [{
            id: 'move-prev',
            text: '上一步',
            handler: navHandler.createDelegate(this, [-1]),
            disabled: true
        }, '->',{
```

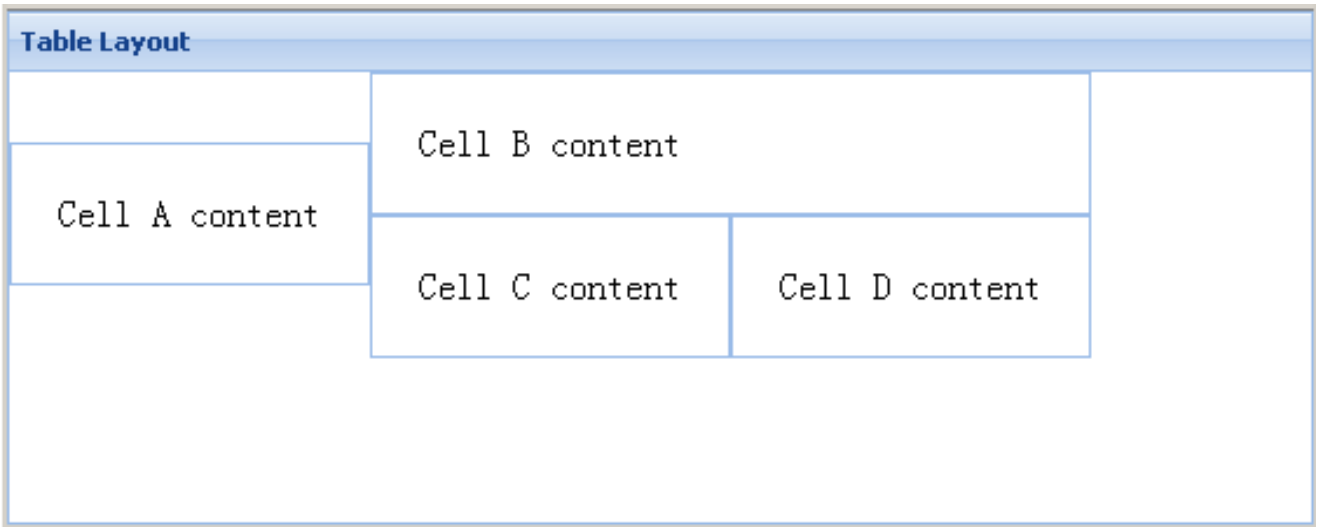
```
            id: 'move-next',
            text: '下一步',
            handler: navHandler.createDelegate(this, [1])
        }],
        items: [{
            id: 'card-0',
            html: '<h1>哈哈！<br />欢迎用咱的向导。</h1><p>第一步，一共三步</p>'
        },{
            id: 'card-1',
            html: '<p>第二步，一共三步</p>'
        },{
            id: 'card-2',
            html: '<h1>恭喜恭喜，完成了。</h1><p>第三步，一共三步，最后一步了。</p>'
        }]
    },{
        region: 'center',
        split: true,
        border: true
    }]
});
```

bbar里设置的是两个按钮，items里的布局还是老样子，倒是控制上下翻页的代码handler比较麻烦，不过这就不在咱们布局范围内了，有兴趣可以去翻翻代码。

lingo-sample/2.0/06-11.html。

6.7.3. 呼呼，TableLayout就是合并行，合并列

上来就是图：



代码啦：

```
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'center',
        title: 'Table Layout',
        layout:'table',
        defaults: {
            bodyStyle:'padding:20px'
```

```
        },
        layoutConfig: {
            columns: 3
        },
        items: [{
            html: '<p>Cell A content</p>',
            rowspan: 2
        },{
            html: '<p>Cell B content</p>',
            colspan: 2
        },{
            html: '<p>Cell C content</p>'
        },{
            html: '<p>Cell D content</p>'
        }]
    }]
});
```

其中layoutConfig中定义了一个要分几列，然后在items中定义各个子面板，使用rowspan和colspan来设置如何进行行列的合并。它可是遵循从左到右，从上到下的，就跟平常你用table的时候是一个样子。

唉，这个TableLayout实在没什么可稀奇的，跟上两个比可差远了呢。例子在lingo-sample/2.0/06-12.html。

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。

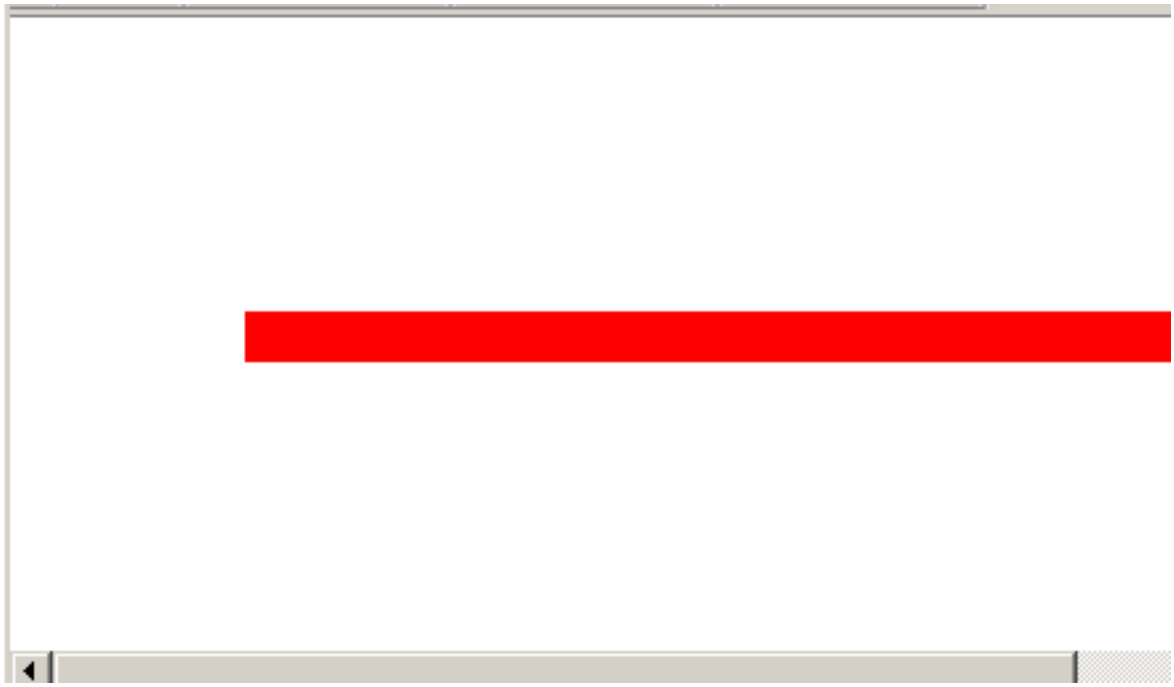
一直认为拖动很复杂，可实际上只需要一条语句就可以了。

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

然后看看html里的部分

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

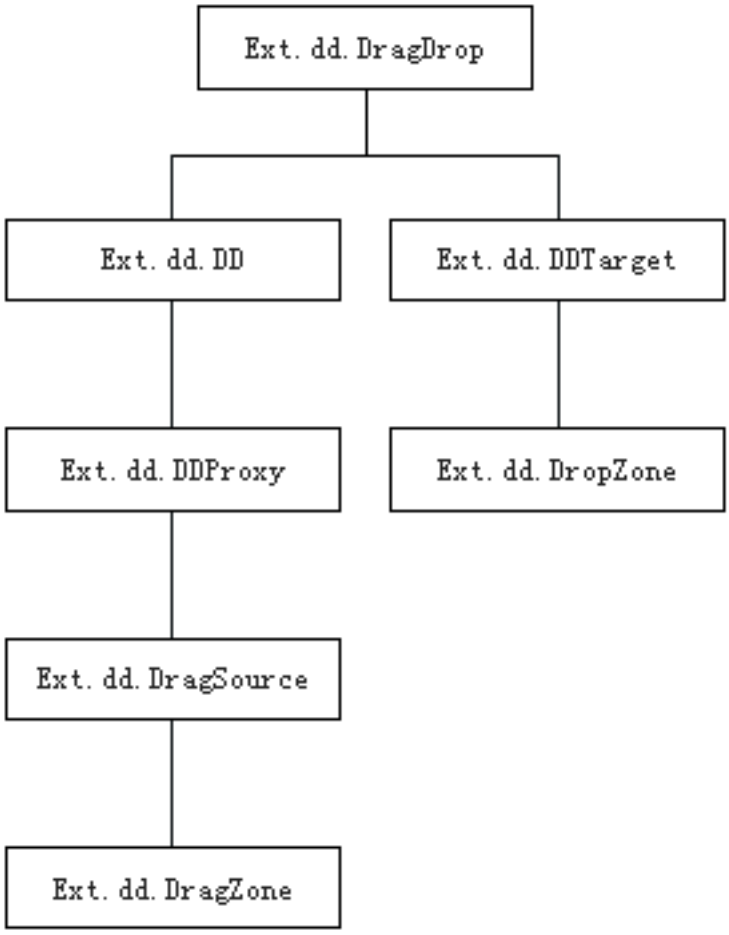


截图里的红条条可以随便拖，实际看例子吧。lingo-sample/1.1.1/07-01.html，2.0里用法一样，

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

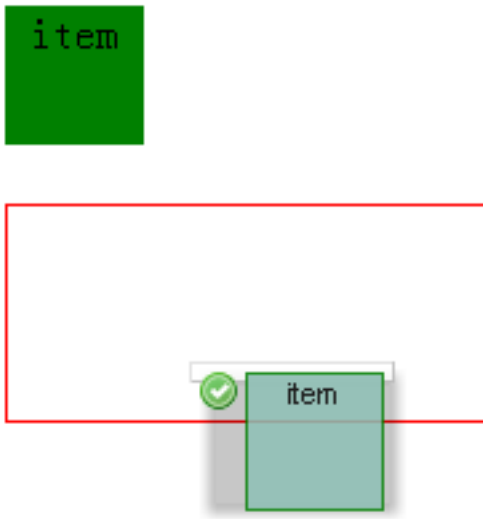
乱扔到任何地方。只有相同组名的目的地才能接收它，好比你只能把超市货架上的商品放进篮子，而不能满地乱扔一样。

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. yui自远方来，不亦乐乎

虽然ext里将拖拽单独列在一个命名空间里，但是examples里竟然半个例子都没有，这让我们在享受grid和tree内置拖拽的同时，又感觉超级无奈，难道就没有什么地方告诉我们怎么把Ext.dd单独拿出来用吗？难道我们就只能眼巴巴得看着这些好东东不成？

幸好yui给咱们提供了不少拖拽的例子，ext本来又是从yui里发展出来的，这下子可便宜咱们啦，借yui的东西搞搞Ext.dd。准备停当，让咱们一一看来。

7.4.1. Basic，基础

咱们看看yui里这个最基本的例子是怎么写的吧。

```
dd1 = new Ext.dd.DD("dd-demo-1");
dd2 = new Ext.dd.DD("dd-demo-2");
dd3 = new Ext.dd.DD("dd-demo-3");
```

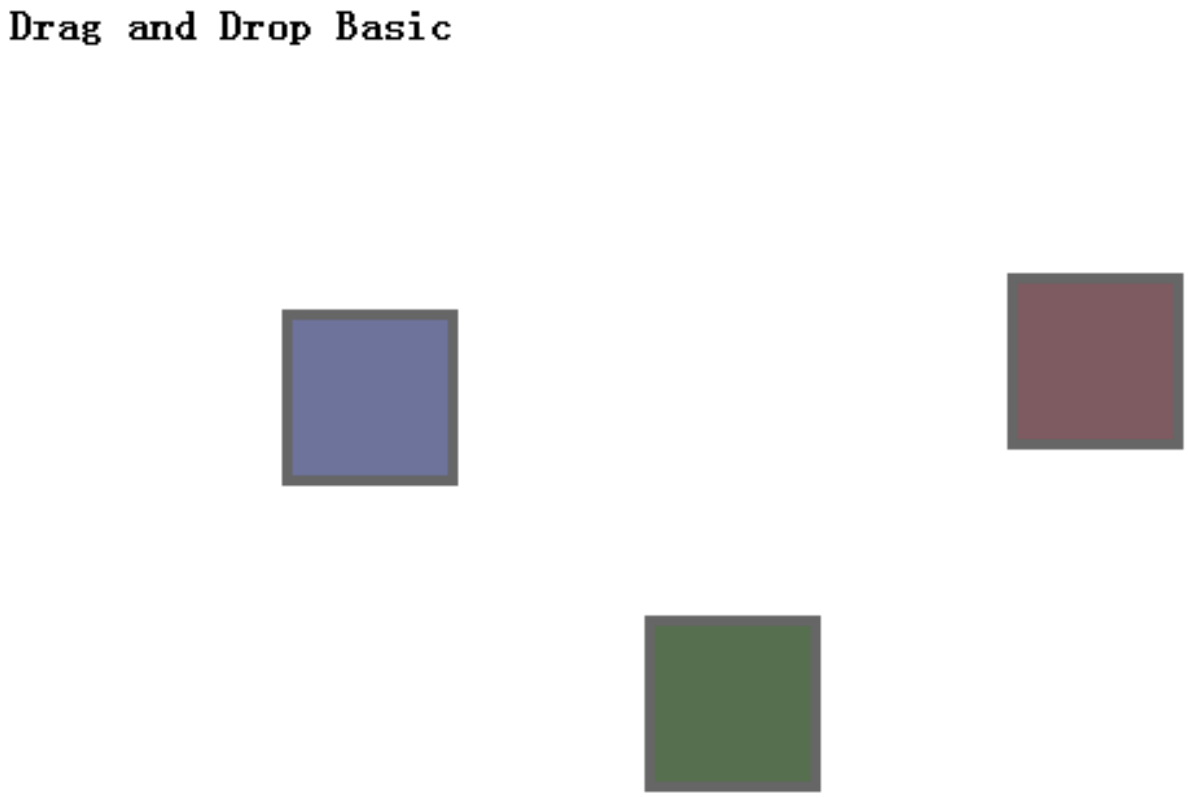
```

```

这块是js代码，创建三个拖动对象，参数对应三个html中的id，让这三个东东变成可拖动的。

```
<div id="dd-demo-1" class="dd-demo"></div>
<div id="dd-demo-2" class="dd-demo"></div>
<div id="dd-demo-3" class="dd-demo"></div>
```

class部分不用看了，那些css样式只是为了漂亮，唯一有关联的就是那些id。这三个div现在已经被ext改造成可拖拽的对象了，可以随便拖起来，也可以放到任何地方。

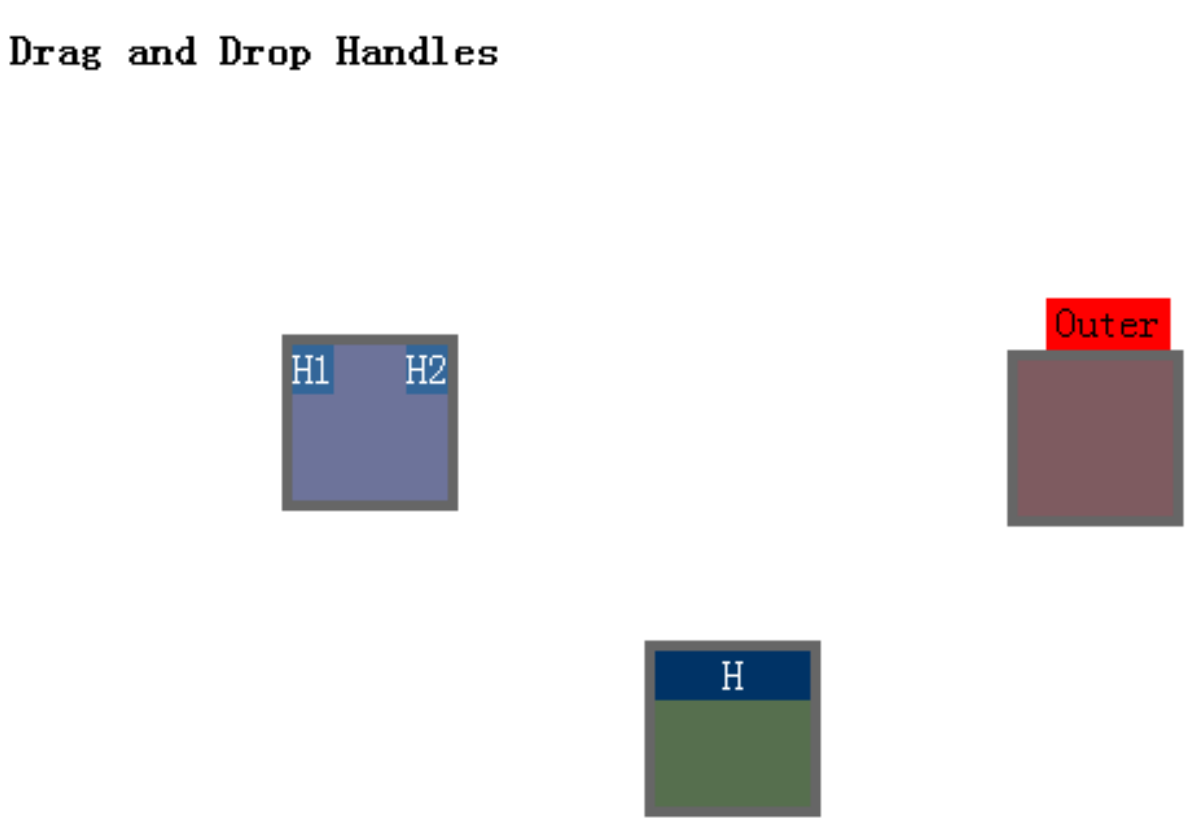
东西很简单，简单到没什么用处，效果如下：



例子在lingo-sample/1.1.1/07-03.html和lingo-sample/2.0/07-03.html。

7.4.2. Handles，把手

这个把手的意思就是，拿菜刀，没人去抓刀刃，有了木头把，咱们才能抄刀切肉。所以即便是可以拖拽的对象，也要拿对了地方才能让你拖滴。



还是刚才的三个框框，第一个div上放两个小把手，第二个div上放一个把手，第三个把手放到div外边，不会随着div一起拖动的。现在，只有鼠标放到handle的位置上才能拖动，其他部分是不起作用的。

先看一下div的结构呢，在原来的div基础上加上handle对应的div。

```
<div id="dd-demo-1" class="dd-demo">
    <div id="dd-handle-1a" class="dd-multi-handle-1">H1</div>
    <div id="dd-handle-1b" class="dd-multi-handle-2">H2</div>
</div>
<div id="dd-demo-2" class="dd-demo">
    <div id="dd-handle-2" class="dd-handle">H</div>
</div>
<div id="dd-handle-3" class="dd-outer-handle">Outer</div>
<div id="dd-demo-3" class="dd-demo"></div>
```

你看啊，前两个是放到里边的，第三个handle放在外边咯，然后我们就给这些handle添加功能。

```
dd1 = new Ext.dd.DD("dd-demo-1");
    dd1.setHandleElId("dd-handle-1a");
    dd1.setHandleElId("dd-handle-1b");
dd2 = new Ext.dd.DD("dd-demo-2");
    dd2.setHandleElId("dd-handle-2");
dd3 = new Ext.dd.DD("dd-demo-3");
    dd3.setOuterHandleElId("dd-handle-3");
```

用法很简单，只要调用Ext.dd.DD的setHandleElId()函数，指定handle的id就好的哦。还有一个setOuterHandleElId()是专门指定外部handle的。这样配置一下，以后就只能在handle上拖拽了呢。

例子是在lingo-sample/1.1.1/07-04.html和lingo-sample/2.0/07-04.html。

7.4.3. On Top，总在上边

为了拖拽方便，让咱们正在拖拽的div总是现在在最上边，这样它不会被其他什么东西遮挡住，让咱们能看清楚到底拖拽到了个什么地方。

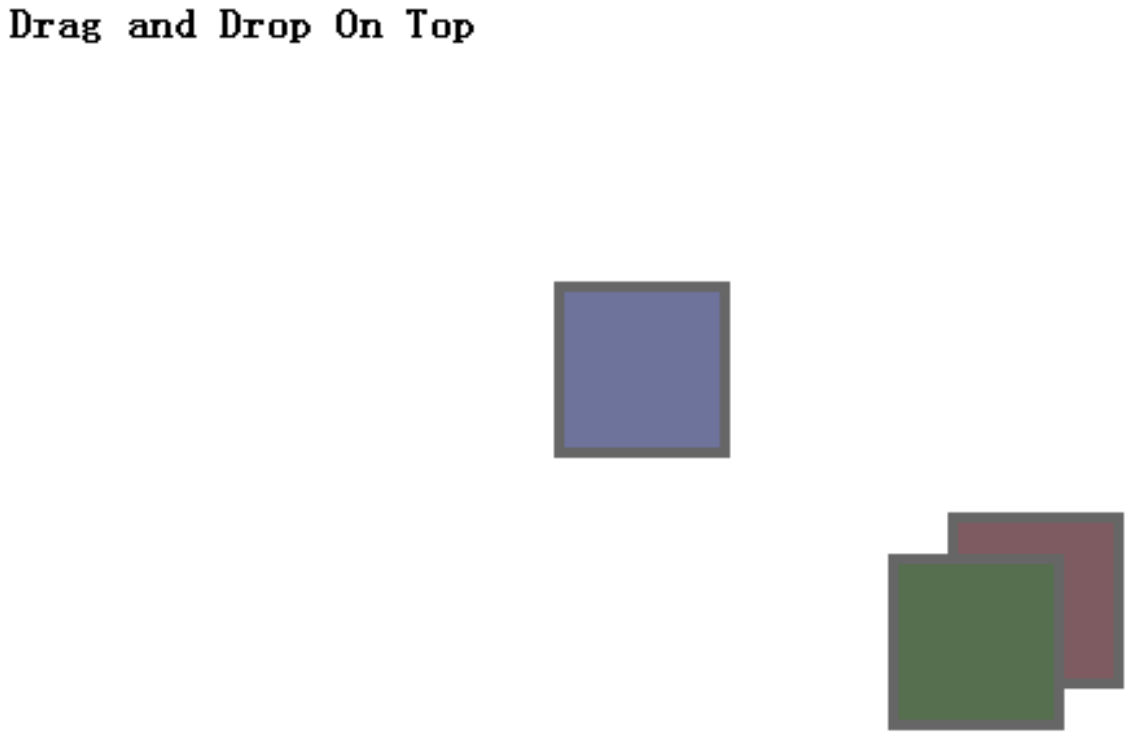
为了实现这个效果，咱们要添加代码了，重写监听拖拽事件的函数哦。

```
Ext.ux.DDOnTop = function(id, sGroup, config) {
    Ext.ux.DDOnTop.superclass.constructor.apply(this, arguments);
};

Ext.extend(Ext.ux.DDOnTop, Ext.dd.DD, {
    origZ: 0,

    startDrag: function(x, y) {
        var style = this.getEl().style;
        this.origZ = style.zIndex;
        style.zIndex = 999;
    },

    endDrag: function(e) {
        this.getEl().style.zIndex = this.origZ;
    }
});
```

咱们在这里定义了一个新对象Ext.ux.DDOnTop，让它继承Ext.dd.DD，还要重写startDrag()和endDrag()两个函数的。开始拖拽的时候会执行startDrag()把正在拖拽的这个el的zIndex设置成999，这个值已经很大了，基本可以保证一直现在到最上面了，等到停止拖拽的时候再执行endDrag()把咱们这个el的zIndex回归到原来的数据。

下面不需要再改什么啦，创建三个Ext.ux.DDOnTop对象就好咯，于是总在最上边的效果就出来啦。

```
dd1 = new Ext.ux.DDOnTop("dd-demo-1");
dd2 = new Ext.ux.DDOnTop("dd-demo-2");
dd3 = new Ext.ux.DDOnTop("dd-demo-3");
```

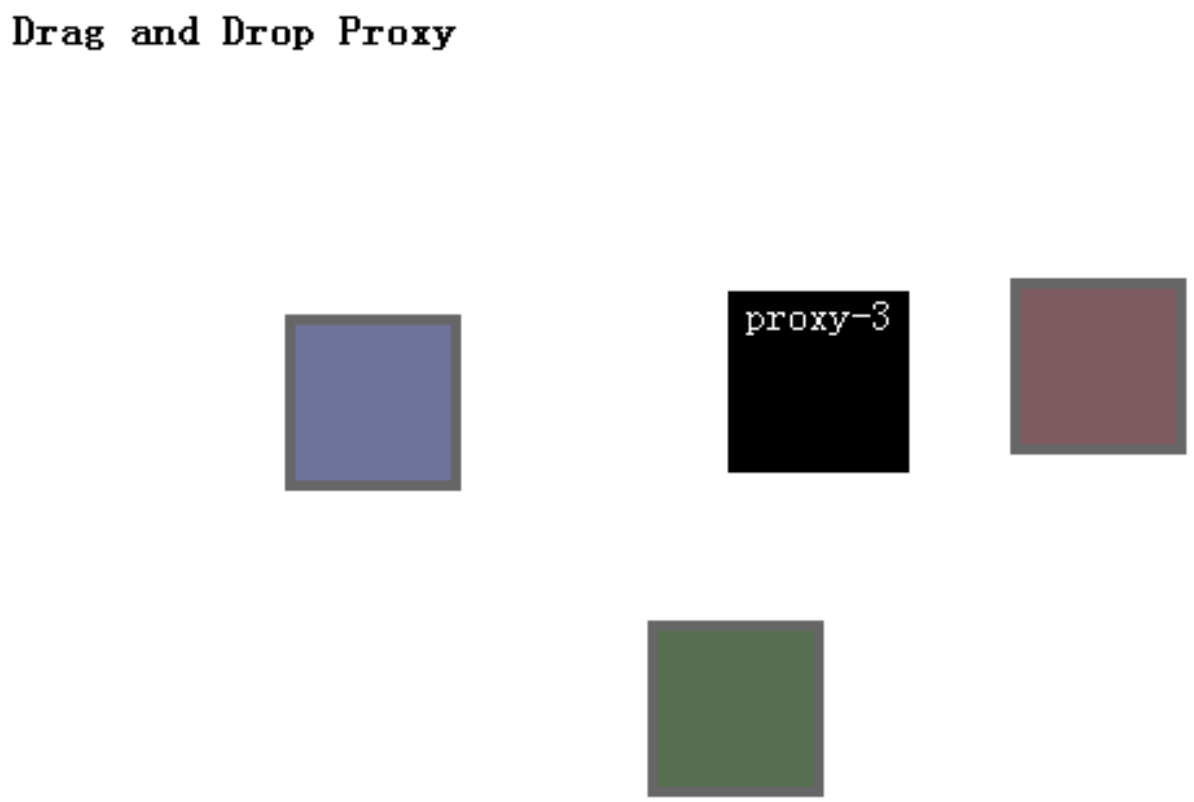


例子在lingo-sample/1.1.1/07-05.html和lingo-sample/2.0/07-05.html。

7.4.4. Proxy，代理

代理啦，就是拖拽的时候，原div不动，鼠标上挂着的是一个叫做Proxy的东西，嗯，值得研究一下呢。

到时候出来的效果是这样滴：



对于这个代理呢，最简单的办法就是把原来的Ext.dd.DD改成Ext.dd.DDProxy呢。

```
dd1 = new Ext.dd.DDProxy("dd-demo-1");
dd2 = new Ext.dd.DDProxy("dd-demo-2");
```

这样默认就会出现一个只有外框的空白proxy，不过咱们也可以自定义proxy的形式，比如咱们就可以搞成截图里黑色proxy。

```
dd3 = new Ext.dd.DDProxy("dd-demo-3", "default", {
    dragElId: "dd-demo-3-proxy",
    resizeFrame: false
});
```

为了实现自定义proxy，咱们在创建DDProxy的时候就要设置三个参数，第一个参数是对应的div的id，第二个参数是拖拽的组，只有同组的Drag才能放到同组的Drop上的，这个咱们现在没看到，不用考虑呢，第三个参数就是其他参数啦。

dragElId的值是咱们自定义的proxy，而resizeFrame:false告诉ext不用去把proxy的大小刻意变成原div一样的。

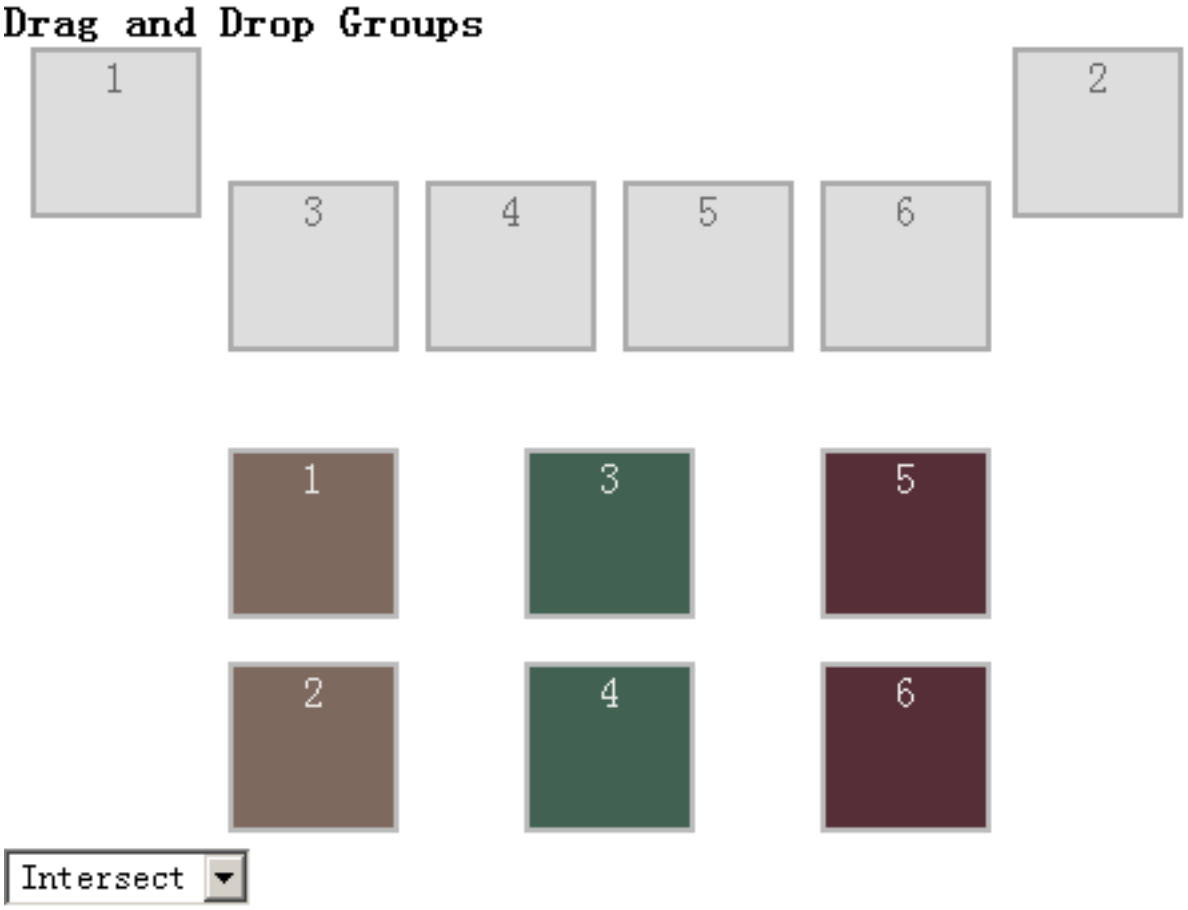
看到啦，第三个proxy对应着的dd-demo-3，咱们再给html里加上这个div就好了。

```
<div id="dd-demo-3-proxy">proxy-3</div>
```

呼呼，例子在lingo-sample/1.1.1/07-06.html和lingo-sample/2.0/07-06.html。

7.4.5. Groups，组

这个例子可是比较复杂了，不过多数的代码都是用来实现高亮效果的，分组的代码是Ext.dd中内置的，使用起来还是很方便的。



简要说明一下的，下边放着6个框框，上边也对应着6个框框，咱们要做的就是把下边的东东拖到上边的框里。不过现在可没有以前那么自由了，下边的框框只能拖到上边的框框里，而且这6个方框的拖拽形式也是有区别的哦，1号和2号只能放到上边的1，2号上，3，4号可以放到3，4，5，6号上，5，6号可以放到任何一个框上。这种方式就叫做分组了，只有在同一组里的才能拖拽放上去的。

在Ext.dd里可以拖拽的东东就叫做DDProxy，可以让这些DDProxy放下的区域叫DDTarget。我们可以把DragSource拖动到同组的DDTarget里，这样就限制了拖拽的范围咯。

在html中的内容是这样了：

```
<div id="workarea">
    <div class="slot" id="t1" >1</div>
    <div class="slot" id="t2" >2</div>
    <div class="slot" id="b1" >3</div>
    <div class="slot" id="b2" >4</div>
    <div class="slot" id="b3" >5</div>
    <div class="slot" id="b4" >6</div>

    <div class="player" id="pt1" >1</div>
    <div class="player" id="pt2" >2</div>
```

```
    <div class="player" id="pb1" >3</div>
    <div class="player" id="pb2" >4</div>
    <div class="player" id="pboth1" >5</div>
    <div class="player" id="pboth2" >6</div>
</div>
<select id="ddmode" >
    <option value="0" selected>Point</option>
    <option value="1">Intersect</option>
</select>
```

这里包括，上边6个框框，下边6个框框，还有一个下拉框用来选择拖拽碰撞的检测模式。

现在来看看如何在js中，将这些div转换成可拖拽的组件呢。

首先为上边6个div创建DDTarget。

```
var slots = [];
// slots
slots[0] = new Ext.dd.DDTarget("t1", "topslots");
slots[1] = new Ext.dd.DDTarget("t2", "topslots");
slots[2] = new Ext.dd.DDTarget("b1", "bottomslots");
slots[3] = new Ext.dd.DDTarget("b2", "bottomslots");
slots[4] = new Ext.dd.DDTarget("b3", "bottomslots");
slots[5] = new Ext.dd.DDTarget("b4", "bottomslots");
```

第一个参数是对应的id，第二个参数是组名，这里把6个DDTarget分成了两个组，topslots和buttomslots。以后的操作就都是基于这两个组来的了。

再下面就是操作Ext.dd.DDProxy了，咱们这里为了实现那些高亮的样式，继承Ext.dd.DDProxy实现了新的类型Ext.ux.DDPlayer，这些不影响咱们的分组操作。

```
var players = [];
// players
players[0] = new Ext.ux.DDPlayer("pt1", "topslots");
players[1] = new Ext.ux.DDPlayer("pt2", "topslots");
players[2] = new Ext.ux.DDPlayer("pb1", "bottomslots");
players[3] = new Ext.ux.DDPlayer("pb2", "bottomslots");
players[4] = new Ext.ux.DDPlayer("pboth1", "topslots");
players[4].addToGroup("bottomslots");
players[5] = new Ext.ux.DDPlayer("pboth2", "topslots");
players[5].addToGroup("bottomslots");
```

1，2号DDPlayer对应的分组是topslots，3，4号DDPlayer对应的分组是bottomslots，5，6号DDPlayer在创建的时候都是对应在topslots这个组的，随后立刻调用addToGroup()这个函数，为这两个DDPlayer添加上bottomslots组，这样让5，6号DDPlayer跟两个组都绑定了，它两个就可以拖放到所有的DDTarget上了。

最后的这块是为Ext.dd.DragDropMgr设置碰撞检测模式，分别是point和intersect，point对应的值是0，intersect对应的值是1：

```
Ext.dd.DragDropMgr.mode = Ext.get('ddmode').dom.selectedIndex;
```

```
Ext.get('ddmode').on('change', function() {
    Ext.dd.DragDropMgr.mode = Ext.get('ddmode').dom.selectedIndex;
});
```

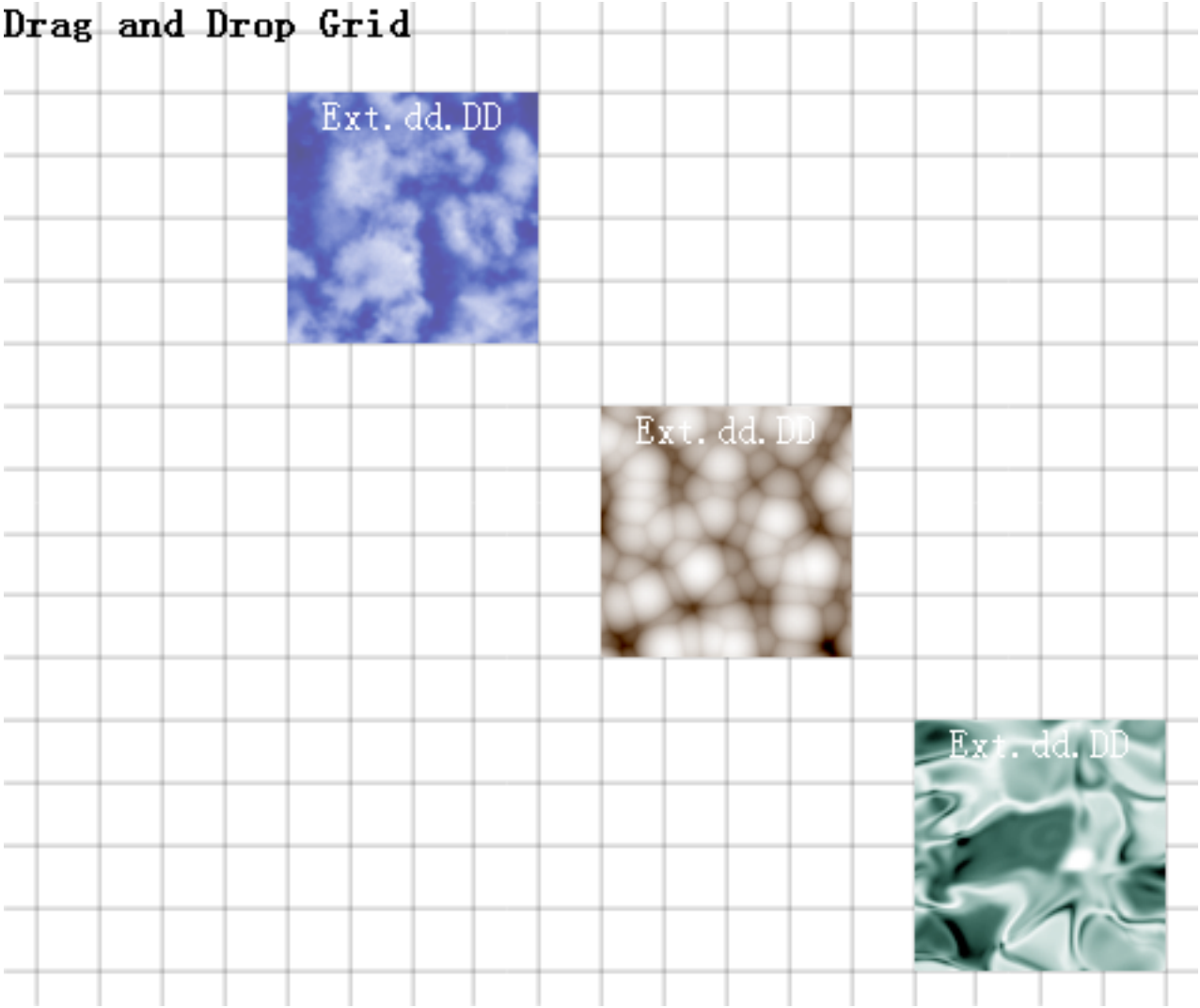
这两者的区别是，在point模式下，当拖拽的鼠标进入DDTarget的范围才能放下。在intersect模式下，当拖拽的DDProxy边缘与DDTarget有重叠的情况下就可以放下。

Ext.ux.DDPlayer里可是超级复杂的，继承自Ext.dd.DDProxy，然后在里边使用Ext.dd.DragDropMgr操作互相之间有关联的元素吧。

例子在lingo-sample/1.1.1/07-07.html和lingo-sample/2.0/07-07.html。

7.4.6. Grid，网格

为拖拽对象设置步长，每次拖拽就像是按照网格跳着走了，要是可以根据这个做页面模板布局，应该是很爽的哟。



这里是html里的定义：

```
<div id="dragDiv1" class="testSquare">Ext.dd.DD</div>
<div id="dragDiv2" class="testSquare">Ext.dd.DD</div>
<div id="dragDiv3" class="testSquare">Ext.dd.DD</div>
```

```
```

页面上显示的网格和拖拽的图片都是通过css定义的，背景是使用了这样的定义：

```
body { background: url("img/grid.png") }
```

有了这些准备，我们再利用Ext.dd来制作拖拽对象，不过这次还要加上对步长的控制：

```
dd1 = new Ext.dd.DD("dragDiv1");
dd1.setXConstraint(1000, 1000, 25);
dd1.setYConstraint(1000, 1000, 25);

dd2 = new Ext.dd.DD("dragDiv2");
dd2.setXConstraint(1000, 1000, 25);
dd2.setYConstraint(1000, 1000, 25);

dd3 = new Ext.dd.DD("dragDiv3");
dd3.setXConstraint(1000, 1000, 25);
dd3.setYConstraint(1000, 1000, 25);
```

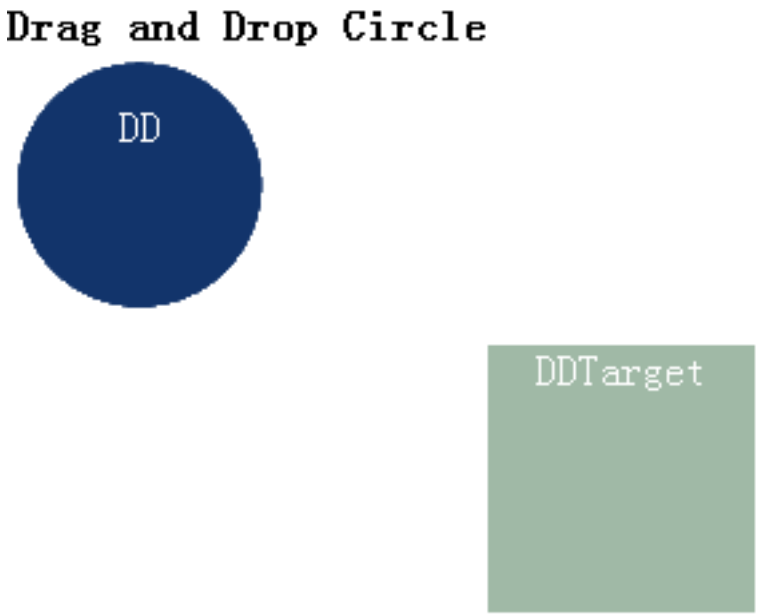
前面生成Ext.dd.DD对象的形式与之前完全一样，不同的是为这些dd设置步长。看一下setXConstraint()和setYConstraint()两个函数吧。

这两个函数一共有三个参数，对于setXConstraint()分别是左侧可以达到的最远距离，右侧可以达到的最远距离和每次移动的步长，setYConstraint()对应的就是竖直上的距离了。这样，左右上下可以从-1000到1000，每次移动25个像素正好跟背景格子的大小一样啦。嘿嘿~

例子在lingo-sample/1.1.1/07-08.html和lingo-sample/2.0/07-08.html。

7.4.7. Circle，圆形

呼呼，当然html里是不能画圈的，这里是用的一个圆形图片啦，不过在为了让效果逼真，在拖拽的时候检测碰撞的算法还是把这个拖拽体当作圆形来看待的，这样我们也可以看到如何自己设定碰撞边界了。



这个圆形就是可以拖拽的东西，方形是对应的DDTarget，只有把圆形拖到方形上才能真正放下，要不又会飞回原地，不信的话你可以试试呀。

为了实现这种效果，咱们要为拖拽对象设置拖拽校验，判断圆形与方形碰撞的时候才会放到方形里，而不是默认的两个方框之间的碰撞了。

1. 第一步，获得拖拽对象的初始位置。

```
var clickRadius = 46;

var el = Ext.get("dd-demo-1");
startPos = el.getXY();
```

clickRadius是圆形的半径，以后用来计算碰撞边界的。

2. 第二步，生成拖拽对象。

```
dd = new Ext.dd.DD(el);
```

3. 第三步，为推拽对象设置校验函数。

```
dd.clickValidator = function(e) {

    var el = this.getEl();
    var region = Ext.get(el).getRegion();

    var r = clickRadius;

    var x1 = e.getPageX(), y1 = e.getPageY();

    var x2 = Math.round((region.right + region.left) / 2);
```

```
    var y2 = Math.round((region.top + region.bottom) / 2);

    e.preventDefault();

    return ( ((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)) <= r*r );
};
```

我们先获得DragTarget的region，这个region里保存了上下左右四条边界，用这个可以计算出DragTarget中心点的位置x2,y2，然后再获得x1,y1，这个是拖拽的时候，鼠标的x,y坐标。

好，我们计算一下这两点之间的距离，就是鼠标与DragTarget之间的距离啦，如果这个距离小于半径，就是说圆形可以放到DragTarget上了，否则就是无效的啦。在这两种情况下会分别执行onDragDrop()和onInvalidDrop()两个函数。

onDragDrop()会把拖拽对象的位置设置成DragTarget的位置。

```
dd.onDragDrop = function(e, id) {
    Ext.get(this.getEl()).setXY(Ext.get(id).getXY());
}
```

onValidDrop()会让拖拽对象返回原位，moveTo()函数的前两个坐标是x和y，第三个参数是true的时候会开启动画效果。

```
dd.onInvalidDrop = function(e) {
    Ext.get(this.id).moveTo(startPos[0], startPos[1], true);
}
```

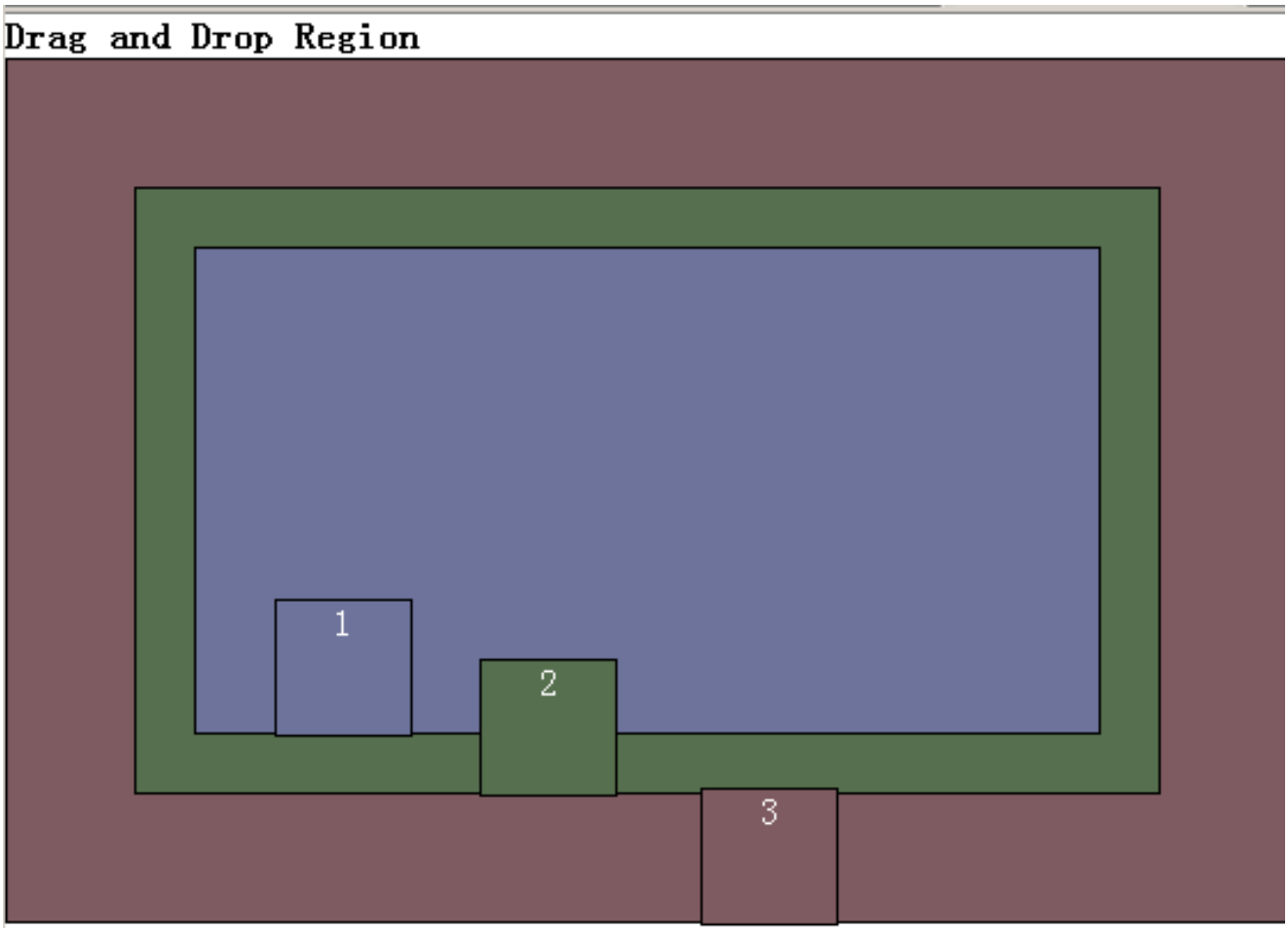
4. 最后一步，就是设置DDTarget。

```
dd2 = new Ext.dd.DDTarget("dd-demo-2");
```

一切都搞定了，你可以试验这个圆形的拖拽了。例子在lingo-sample/1.1.1/07-09.html和lingo-sample/2.0/07-09.html。

7.4.8. Region，范围

在页面上画个框，让拖拽对象只能在这个框里拖来拖去，限制可以推拽的边界呢，这样用户就不能把拖拽对象扔到天边去了，呼呼。



就像这样，蓝色方框只能在蓝色区域里移动，绿色方框只能在绿色方框里移动，棕色方框只能在棕色区域里移动。

看一下html里的内容，把三个可拖拽的div放到三个不同大小的div里：

```
<div id="dd-demo-canvas1">
    <div id="dd-demo-canvas2">
        <div id="dd-demo-canvas3">
            <div id="dd-demo-1" class="dd-demo"><div>1</div></div>
            <div id="dd-demo-2" class="dd-demo"><div>2</div></div>
            <div id="dd-demo-3" class="dd-demo"><div>3</div></div>
        </div>
    </div>
</div>
```

dd-demo-1,2,3是可拖拽的对象，dd-demo-canvas1,2,3对应的是三个拖拽的范围，为了实现这效果，还专门创建了Ext.ux.DDRegion对象：

```
dd1 = new Ext.ux.DDRegion('dd-demo-1', '', { cont: 'dd-demo-canvas3' });
dd2 = new Ext.ux.DDRegion('dd-demo-2', '', { cont: 'dd-demo-canvas2' });
dd3 = new Ext.ux.DDRegion('dd-demo-3', '', { cont: 'dd-demo-canvas1' });
```

DDRegion是继承自Ext.dd.DD的，所以创建时候的三个参数就是拖拽div的id，推拽组，和附加参数，这里的附加参数使用的cont对应了可拖拽区域的id，这个参数是在DDRegion里设置的，让咱们回头看

看DDRegion里是如何做的吧。

```
Ext.ux.DDRegion = function(id, sGroup, config) {
    this.cont = config.cont;
    Ext.ux.DDRegion.superclass.constructor.apply(this, arguments);
};
```

首先在构造函数里把config.cont赋给this.cont，然后调用DDRegion父类的构造函数进行初始化。

```
Ext.extend(Ext.ux.DDRegion, Ext.dd.DD, {
    cont: null,
    init: function() {
        Ext.ux.DDRegion.superclass.init.apply(this, arguments);
        this.initConstraints();
    },
    initConstraints: function() {
        var region = Ext.get(this.cont).getRegion();

        var el = this.getEl();
        var xy = Ext.get(el).getXY();

        var width = parseInt(Ext.get(el).getStyle('width'), 10);
        var height = parseInt(Ext.get(el).getStyle('height'), 10);

        var left = xy[0] - region.left;
        var right = region.right - xy[0] - width;

        var top = xy[1] - region.top;
        var bottom = region.bottom - xy[1] - height;

        this.setXConstraint(left, right);
        this.setYConstraint(top, bottom);
    }
});
```

剩下的这些都是在initConstraints()函数里进行配置的。this.cont在这里用计算它所在的region，获得了可拖拽区域的上下左右四个边界，然后把这四个值用setXConstraint()和setYConstraint()设置给Ext.dd.DD啦，这就限定了四个方向可移动的范围呢。

说到底，还是利用了setXConstaint()和setYConstraint()定义拖拽移动的范围了。例子就在lingo-sample/1.1.1/07-10.html和lingo-sample/2.0/07-10.html上的。

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？

1. 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 ，然后我们用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。

2. 最吸引眼球的是动画效果，所以我们先动两下。

```
myDiv.hightlight();
```

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

3. 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。

1. 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();
```

2. 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和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

1. 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？

2. cls，指的是 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。-_-。

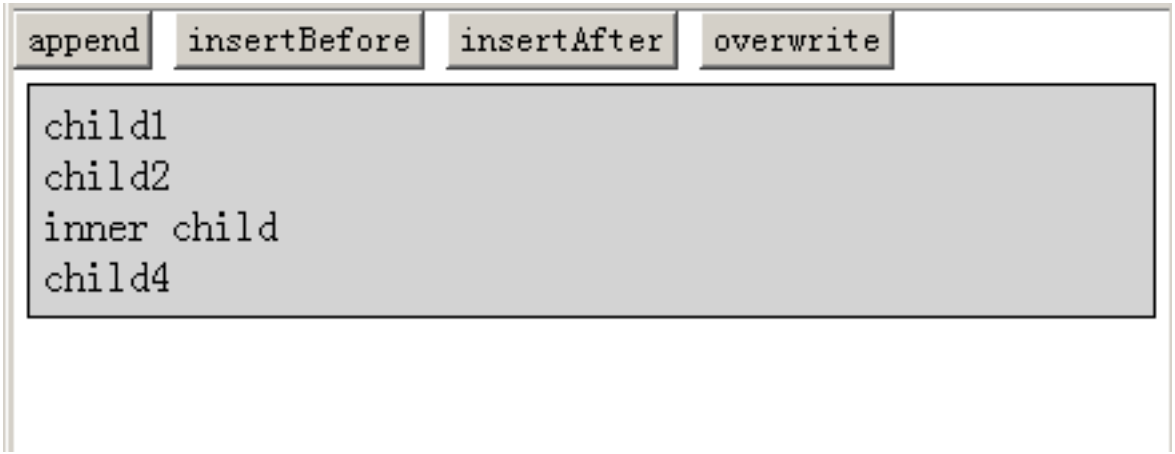
   他就喜欢玩这个，把dataStore写成ds，DomHelper写成dh，Element写成el，ColumnModel写成cm，SelectionModel是sm。唉，发明的专业名词缩写好多呀。

3. 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。

4. 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。

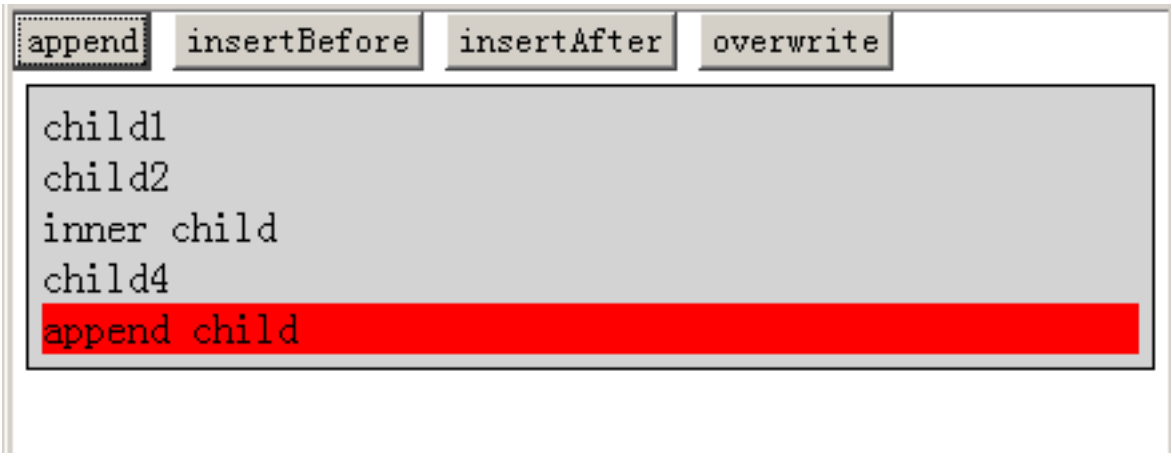


原始的html是这样的。一个div下有4个节点，其中第三个子节点下还有自己的子节点。

```
<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>
```

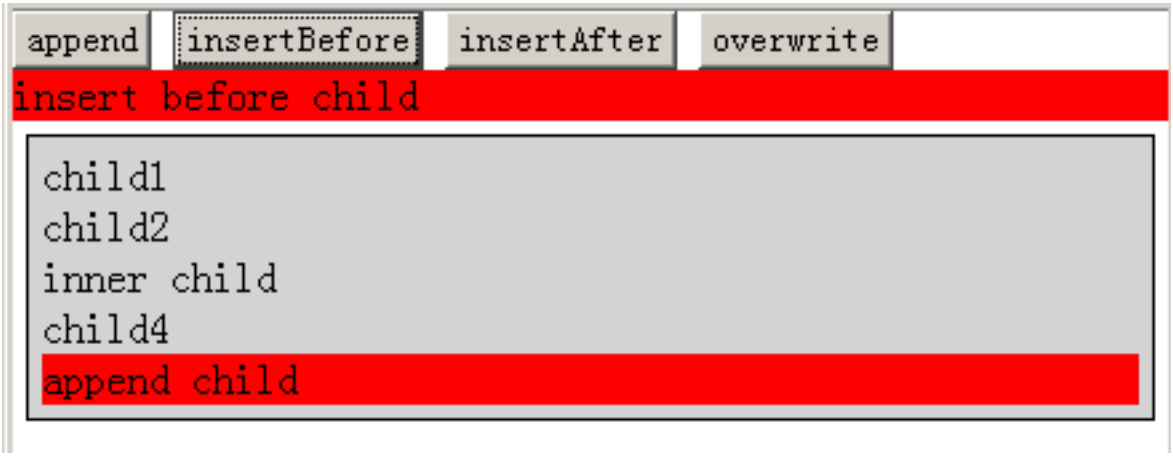
1. append是将新节点放到指定节点的最后。

```
Ext.DomHelper.append('parent', {tag: 'p', cls: 'red', html: 'append child'});
```



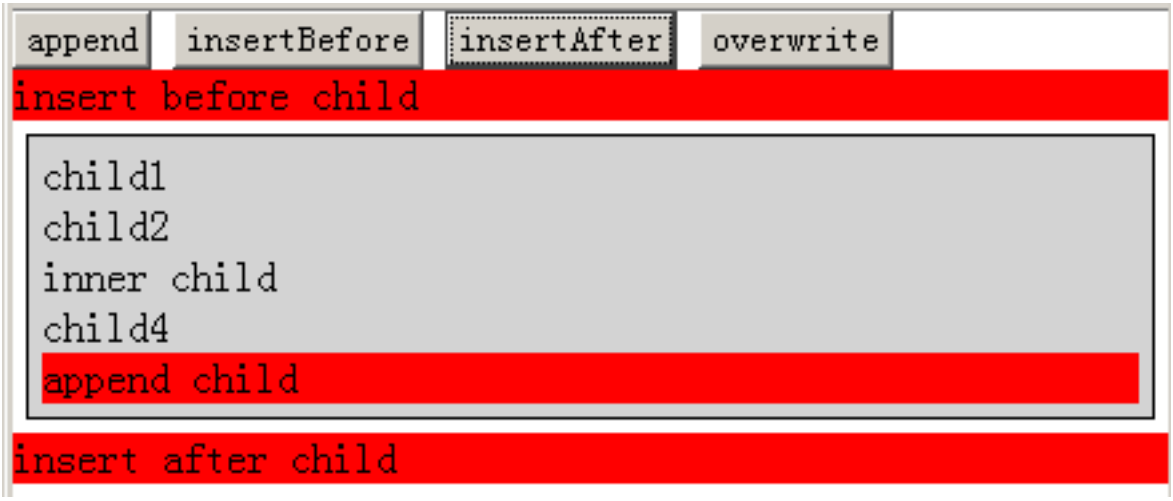
2. insertBefore，新节点插入到指定节点前面。

```
Ext.DomHelper.insertBefore('parent', {tag: 'p', cls: 'red', html: 'insert before child'})
```



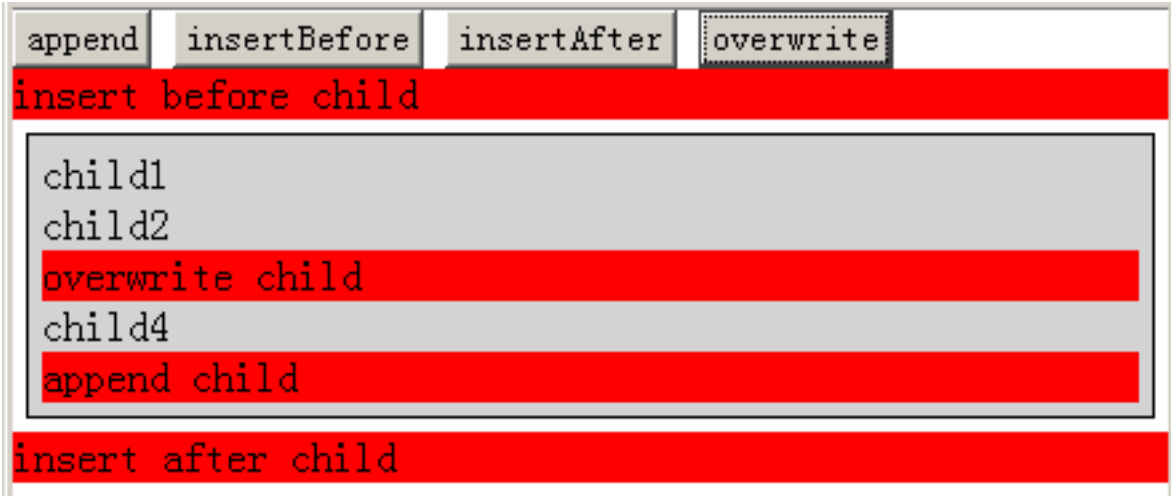
3. insertAfter，新节点插入到指定节点后面。

```
Ext.DomHelper.insertAfter('parent', {tag: 'p', cls: 'red', html: 'insert before child'})
```



4. overwrite，会替换指定节点的innerHTML内容。

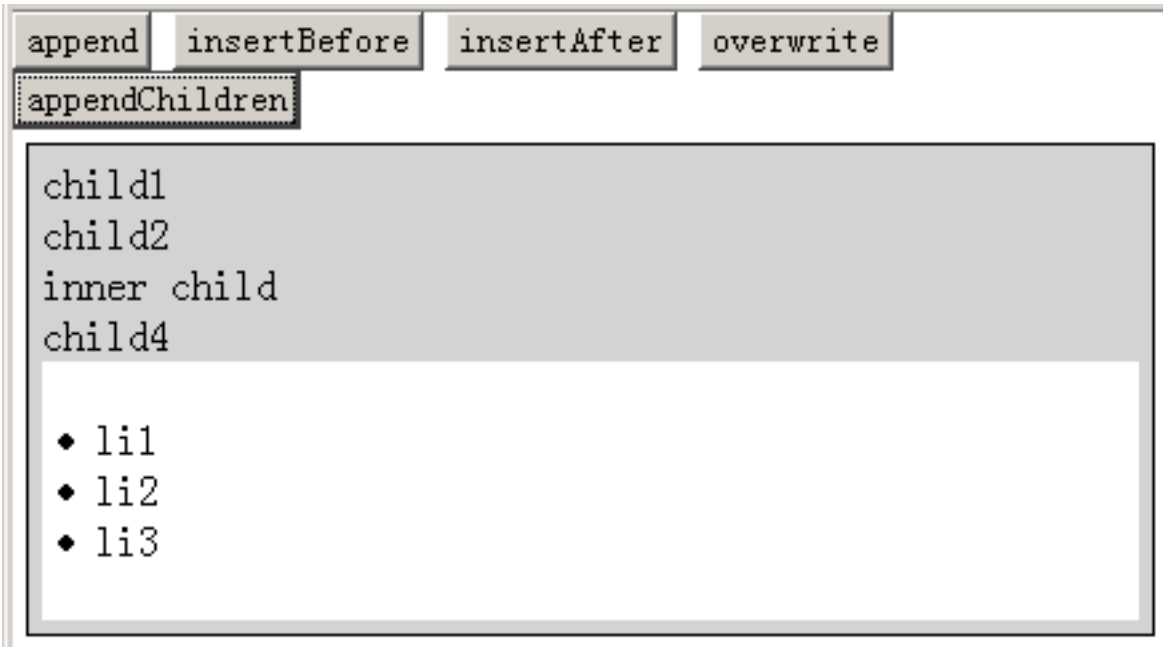
```
Ext.DomHelper.overwrite('child3', {tag: 'p', cls: 'red', html: 'overwrite child'})
```



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[]);
}
```

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。

8.3.3. 醍醐灌顶，MasterTemplate和XTemplate。

对Template的功能进行增强，1.x里的MasterTemplate和2.0中的XTemplate都支持在模板内部使用子模板，不同的是XTemplate中更具有for和if的功能。

8.3.3.1. 1.x中的MasterTemplate

借用api中的例子，咱们使用MasterTemplate定义一个select的模板，可以修改select的name，可选的option则由子模板提供。

1. 第一步定义模板：

```
var t = new Ext.MasterTemplate(
    '<select name="{name}" size="{size}">',
        '<tpl name="options"><option value="{value:trim}">{text:ellipsis(10)}</option></tpl>',
    '</select>'
);
```

里边这个tpl标签就表示子模板，用name给子模板也做一个标识，以后才好专门为它填充数据。

2. 第二步给子模板设置数据：

```
t.addAll('options', [
    {value: '河北', text: '河北省'},
    {value: '唐山', text: '唐山市'},
    {value: '路北', text: '路北区'}
]);
```

addAll()提供数组给子模板循环显示出来，还有一个add()函数是用jsonObject作为参数的，一次只能填充一条数据，感觉不是很有用。

3. 第三步输出模板结果：

```
t.append('f', {name: 's', size: 5});
```

这里是把数据填充到<form id='f'></form>里边，同时提供了父模板的两个参数name和size，最后生成的就是一个三个选项的select。

例子见lingo-sample/1.1.1/08-05a.html。

8.3.3.2. 2.0中的XTemplate

发展到了2.0时代，MasterTemplate已经完全被XTemplate取代了，XTemplate不仅仅支持子模板，甚至有内置的for循环和if选择，可惜的是没有else呢，希望下个版本能支持上。

先试试用XTemplate输出上边那个select的例子。

```
var data = {
    name: 's',
    size: 5,
    options: [
        {value: '河北', text: '河北省'},
        {value: '唐山', text: '唐山市'},
        {value: '路北', text: '路北区'}
    ]
};

var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}">',
        '<tpl for="options"><option value="{value:trim}">{text:ellipsis(10)}</option></tpl>',
    '</select>'
);

t.append('f', data);
```

大致的区别是子模板的name改成了for默认就会进行循环输出。父模板和子模板的数据也不必分几次设置，一个json数据就直接搞定啦，数据与显示分离的更清晰，体现着2.0中统一配置的优势。

例子见lingo-sample/2.0/08-05a.html。

不过，要是XTemplate就只有这点儿本事的话，咱们实在没有单独提出来讨论的必要，现在给你看看它隐藏的实力。

1. 先看看如何操作简单数组：

```
var data = {
    name: 's',
    size: 5,
    options: ['河北省', '唐山市', '路北区']
};

var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}">',
        '<tpl for="options">',
            '<option value="{.:trim}">{.:ellipsis(10)}</option>',
        '</tpl>',
    '</select>'
);
```

   options里改成字符串为内容的简单数组，子模板tpl里直接使用{.}引用每一个数据，默认的格式化函数依然有效呢。

2. 修改一下模板，选中奇数行：

```
var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}" multiple>',
        '<tpl for="options">',
```

```
            '<tpl if="xindex % 2 == 0">',
                '<option value="{.:trim}">{.:ellipsis(10)}</option>',
            '</tpl>',
            '<tpl if="xindex % 2 == 1">',
                '<option value="{.:trim}" selected>{.:ellipsis(10)}</option>',
            '</tpl>',
        '</tpl>',
    '</select>'
);
```

因为没有else，我们只好使用了两个if来进行判断，xindex是for内置的一个变量是从1开始索引值，记得是从1开始的哟，不是数组下标那样从0开始。

3. 简化一下判断：

```
var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}" multiple>',
        '<tpl for="options">',
            '<option value="{.:trim}"{[xindex % 2 == 1 ? " selected" : ""]}>{.:ellipsis(10)}</option>',
        '</tpl>',
    '</select>'
);
```

咱们直接哟你过一个三元判断替换了上边冗长的if判断，因为这个xindex并不是从外部提供的数据，所以要在{}里边再加上一个[]，还有几个内部变量都要用这种方式引用。

表 8.1. 内部变量

变量名	说明
values	当前范围内的变量，每次使用tpl都会进入一个子模板，就只能使用子模板对应的变量了。
parent	如果进入了子模板，就需要通过parent引用上一级模板里的变量
xindex	循环的索引值，从1开始
xcount	循环的总长度
fm	Ext.util.Format，想做什么格式化操作，就直接调用它。

呼，XTemplate的介绍到此告一段落，虽然依然存在瑕疵，比如不支持else，但也是提供了不少功能呢，其实ext-2.0中好多地方都使用到了它做数据的渲染，稍稍理解一下再去看代码也更好理解的。

代码在lingo-sample/2.0/08-05b.html。

8.4. Ext.data命名空间

Ext.data这个namespace下，定义了一系列的Store啊，reader啊，proxy啊。grid，comboxBox都是以这个为媒介获取数据的，这东西好处多多啊，什么异步加载，类型转换，分页这些功能都包含了，它用这种形式支持array，json，xml格式的数据，你可以通过Memory，Http，ScriptTag等方式获得，依照这套策略，你要是想实现什么新的协议，新的数据结构，只需要自己扩展reader和proxy就够了。像dwrproxy就是实现了自己的proxy和reader，让ext可以直接从dwr获得数据，真乃java开发者的福音呀。

8.4.1. proxy系列

代理啦，就是可以通过不同的途径获得我们想要的数据。

8.4.1.1. 人畜无害MemoryProxy

这个MemoryProxy就只能从js对象获得数据，你把一个array（数组啦），json或者是xml放到它里头就ok啦。

```
var proxy = new Ext.data.MemoryProxy([
    ['id1','name1','descn1'],
    ['id2','name2','descn2']
]);
```

8.4.1.2. 常规武器HttpProxy

使用http协议，用ajax去后台取数据的代理，构造它的时候需要设置个url:'xxx.jsp'，这样它才知道去哪里找数据。

```
var proxy = new Ext.data.HttpProxy({url:'xxx.jsp'});
```

后台就需要我们对应设计一下了，返回对应的数据呢。

```
response.setContentType("application/x-json");
Writer out = response.getWriter();
out.print("[" +
        "['id1','name1','descn1']" +
        "['id2','name2','descn2']" +
    "]");
```

多说一句呢，这个HttpProxy是不支持跨域的，它只能从同一domain下获得数据，想要跨域吗？请看下面的ScriptTagProxy。

8.4.1.3. 洲际导弹ScriptTagProxy

用法几乎跟HttpProxy一样。

```
var proxy = new Ext.data.ScriptTagProxy({url:'xxx.jsp'});
```

这个东西可不是平白就支持跨域的，要支持这个必须在后台耍些手腕了。

```
String cb = request.getParameter("callback");
response.setContentType("text/javascript");
Writer out = response.getWriter();
out.write(cb + "(");
out.print("[" +
        "['id1','name1','descn1']" +
        "['id2','name2','descn2']" +
    "]");
out.write(");");
```

奥妙就在于从请求中获得的callback参数，这个东东就叫做回调函数呢，所谓的ScriptTagProxy会在html里加上个<script type="text/javascript" src="xxx.jsp"></script>，然后你返回的是js的内容，它里边就是callback(data)的样子，而这个callback就是ext生成并且传递给后台的。

嗯，希望没把你讲晕，如果听着头疼，还是用firebug看看生成的html以及js吧，会有很大的收获滴。

最后奉上ext的api文档里给的例子，自动判断该返回ScriptTagProxy还是HttpProxy哟：

```
boolean scriptTag = false;
String cb = request.getParameter("callback");
if (cb != null) {
    scriptTag = true;
    response.setContentType("text/javascript");
} else {
    response.setContentType("application/x-json");
}
Writer out = response.getWriter();
if (scriptTag) {
    out.write(cb + "(");
}
out.print(dataBlock.toJsonString());
if (scriptTag) {
    out.write(");");
}
```

嘻嘻……，简单判断请求里是否有callback这个参数，如果有了，就返回ScriptTagProxy需要的数据，否则就当作HttpProxy啦。

8.4.2. reader系列

读取原始数据库，还要解析一下，把他们变成Record，这样才好供给Ext.data.Store使用。

8.4.2.1. 简单易行ArrayReader

这个东东就是从二维数组里挨个取数据，然后对应到Record的name里，默认是一列一列一次读取，不过你也可以考虑用mapping指定列号呢，读取数组很简单，可惜就是不能分页。

加入有二维数组是这样：

```
var data = [
    ['id1','name1','descn1'],
    ['id2','name2','descn2']
]
```

对应的ArrayReader就应该是这样：

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

呵呵~咱们演示的是字段顺序不一致的情况，如果你的字段顺序和列顺序一致，就不用另外配置mapping了。

8.4.2.2. 灵活轻便JsonReader

js里当然要用json啦，key:value的形式好理解，代码量也比xml小，我可是很倾向这种东西哟。

准备好json的数据是这样：

```
var data = {
    id:0,
    totalProperty:2,
    successProperty:true,
    root:[
        {id:'id1',name:'name1',descn:'descn1'},
        {id:'id2',name:'name2',descn:'descn2'}
    ]
};
```

json跟数组比较呢，就是支持分页了呢，我们可以多用一个totalProperty表示数据总量。successProperty是附送滴，用这个可以控制是否进行数据加载，不想让它读取数据的话，让它变成false就可以了。

现在放上JsonReader吧，看看都是咋对应的呢？

```
var reader = new Ext.data.JsonReader({
    successProperty: "successproperty",
    totalProperty: "totalProperty",
    root: "root",
    id: "id"
}, [
    {name:'id',mapping:'id'},
    {name:'name',mapping:'name'},
    {name:'descn',mapping:'descn'}
]);
```

```

```

对应的很准确啦，因为这里每一项的名字都是一样的，其实这里的mapping都可以省略了，如果你不想要json里一样的名字，那还是用mapping修理一下吧。不过呢，mapping在这里可是另有用途滴。看一下撒。

```
var data = {
    id:0,
    totalProperty:2,
    successProperty:true,
    root:[
        {id:'id1',name:'name1',descn:'descn1',person:{
            id:1,name:'man',sex:'male'
        }},
        {id:'id2',name:'name2',descn:'descn2',persion:{
            id:2,name:'woman',sex:'female'
        }}
    ]
};
var reader = new Ext.data.JsonReader({
    successProperty: "successproperty",
    totalProperty: "totalProperty",
    root: "root",
    id: "id"
}, [
    'id','name','descn',
    {name:'person_name',mapping:'person.name'},
    {name:'person_sex',mapping:'person.sex'}
]);
```

看到了没？json支持更复杂的嵌套结构，咱们的person就有自己的id, name和sex，在reader里，我们就用mapping把它里边的东西挖出来，而其他名字不变的，咱们就简写啦。

8.4.2.3. 久负盛名XmlReader

嗯，嗯，xml是非常，非常，非常通用的格式，虽然我认为它过于厚重的，但既然有人在用，咱们就要研究一下啊，可不能把市场白白丢了。

实际上，我们需要的数据大概是这样的格式。

```
<?xml version="1.0" encoding="utf-8"?>
<dataset>
    <id>1</id>
    <totalRecords>2</totalRecords>
    <success>true</success>
    <record>
        <id>1</id>
        <name>name1</name>
        <descn>descn1</descn>
    </record>
    <record>
        <id>2</id>
        <name>name2</name>
        <descn>descn2</descn>
    </record>
```

```
</dataset>
```

说实在的，真不知道为啥要用dataset做根，不过它还是用了，XmlReader里就对应这些东东。

```
var reader = new Ext.data.XmlReader({
   totalRecords: 'totalRecords',
   success: 'success'
   record: 'record',
   id: "id"
}, ['id','name','descn']);
```

呵呵~跟JsonReader里那些参数名字不太一样呢，不过意思可差不多。xml里的标签和reader里需要的名字是一样的，所以简化了呢。

呀，呀，只不过不能用string的方式做xml的原始数据了，参考localXHR.js里构造xml的方式，我们就有了下面的解决方案。

```
var data = "<?xml version='1.0' encoding='utf-8'?>" +
    "<dataset>" +
        "<id>1</id>" +
        "<totalRecords>2</totalRecords>" +
        "<success>true</success>" +
        "<record>" +
            "<id>1</id>" +
            "<name>name1</name>" +
            "<descn>descn1</descn>" +
        "</record>" +
        "<record>" +
            "<id>2</id>" +
            "<name>name2</name>" +
            "<descn>descn2</descn>" +
        "</record>" +
    "</dataset>";

var xdoc;

if(typeof(DOMParser) == 'undefined'){
    xdoc = new ActiveXObject("Microsoft.XMLDOM");
    xdoc.async="false";
    xdoc.loadXML(data);
}else{
    var domParser = new DOMParser();
    xdoc = domParser.parseFromString(data, 'application/xml');
    domParser = null;
}

var proxy = new Ext.data.MemoryProxy(xdoc);

var reader = new Ext.data.XmlReader({
    totalRecords: 'totalRecords',
    success: 'success',
    record: 'record',
    id: "id"
}, ['id','name','descn']);
```

```
var ds = new Ext.data.Store({
    proxy: proxy,
    reader: reader
});
```

8.4.3. 相信你知道怎么做加法

不是每次都必须proxy，reader，store三者全套上阵，实际上你可以选择一些整合方案，看看ext为我们提供了些什么？

1. SimpleStore = Store + MemoryProxy + ArrayReader

```
var ds = Ext.data.SimpleStore({
    data: [
        ['id1','name1','descn1'],
        ['id2','name2','descn2']
    ],
    fields: ['id','name','descn']
});
```

嘿嘿~设置上MemoryProxy需要的data和ArrayReader需要的fields就成啦。哈哈~不愧是简单store啊。

2. JsonStore = Store + HttpProxy + JsonReader

```
var ds = Ext.data.JsonStore({
    url: 'xxx.jsp',
    root: 'root',
    fields: ['id','name','descn']
});
```

嘿嘿~就是瞎搅和到一起，它内部会自动对应这些参数的，以后考虑直接用它好不好呢？

8.5. 跟我用json，每天五分钟

JSON其实是JavaScript Object Notation的缩写，用google translate翻译过来，竟然叫Javascript对象乐谱？咱们要准备好唱歌了。

先介绍一个好网址：http://www.json.org/json-zh.html。从网址的名字来看，这个地方应该是相当的专业啊，不过就是不太好懂，估计是美声唱法，所以下面我要来段通俗的了。

先搭个舞台，我们这里有老爸，老妈，三个孩子，锣鼓点儿响，好戏上场。

8.5.1. Hello 老爸。

一般都是从Hello World开始，不过咱们这里只有老爸，所以只好让他先顶上了。老爸得要个名字，否

则怎么叫他呢？

```
var dad = {
    name: 'Dad'
};
```

请忽略var data = 这些部分，这些都是js的基础，var代表局部变量，dad是变量名， = 是赋值符号。咱们现在要看的是大括号里边的内容。

在json中，大括号代表对象，大括号里的内容都是name:value的形式，冒号前头是属性名，冒号后头是属性值。这样就等于给dad对象里的name属性赋值了，我们这里把name设置成'Dad'以后就可以调用了。像这样：

```
alert('老爸的名字是：' + dad.name);
```

嘿嘿~看看弹出了什么吧？

恭喜恭喜，你已经看到json世界的一半天空了。

8.5.2. 老妈等等，孩子先上场。

咱们上边说了，老爸有三个孩子，分别是'Lala', 'Sasa', 'Dudu'，在json里怎么表示他们呢？

```
var children = [
    'Lala',
    'Sasa',
    'Dudu'
];
```

老规矩，等号和等号之前不考虑，只看等号以后的部分。这次不是大括号了，换成中括号了。在json里中括号代表数组，数组里有三个字符串，正好是三个孩子的名字。嘿嘿~有了这个数组以后我们就可以做下面的事了。

```
alert('一共有' + children.length + '个孩子。');
alert('第一个孩子是：' + children[0]);
```

这个数组是绝对符合任何一个数组的基本法则，你可以通过children.length获得它里边有几个元素，用children[0]来获得指定位置的元素，稍微注意一下索引是从0开始的，而不是1。

恭喜恭喜，json世界的两种结构你都看过了，现在我们来总结一下吧，json包含两种结构，一种是大括号里包着的name:value组合，一种是方括号里包着的数组，再没有其他的了。

8.5.3. 老妈来了，老妈来啦。

请母亲大人登场。鼓乐齐鸣，净水泼街，黄土垫道。母亲大人可不像老爸那么不修边幅，老妈可精细呢。

```
var mum = {
    name: 'Mum',
    birthday: new Date(1982, 9, 2),
    age: 26,
    hasJob: true,
    husband: dad,
    likes: children
};
```

别眼花啊，大括号里这么多name:value组合，每组之间都用逗号分隔的。请注意哟，最后一组name:value后面不能有逗号，当然我们非常可能顺手多写一个逗号，这样在firefox下不会有问题，firefox会忽略这个疏忽，但ie下会立即报错，搞垮整个js脚本。切记切记。

这个问题是非常非常，常见的问题，ext官网上甚至有人建议这样写：

```
var mum = {
    name: 'Mum'
  , birthday: new Date(1982, 9, 2)
  , age: 26
```

```
  , hasJob: true
  , husband: dad
  , children: children
};
```

嗯，不说这个了，让我们仔细瞧瞧老妈的个人档案上都写了些什么。

1. name名字，字符串

2. birthday生日，日期类型

3. age年龄，数字

4. hasJob有工作吗？布尔型true/false

5. husband丈夫，自定义对象，就是老爸啦。

6. chlidren孩子们，是个数组。

当然当然，如果其中某一项不存在，也可能是null。呼呼，老妈真是不简单，基本包含了json对象中所有可以用到的数据类型了。看一遍也很简单，熟练应用可不会是什么问题呢。

8.5.4. Ext对json的支持力度

如果目前我们有一下数据：

```
var dadData = '{name:"Dad"}';
var childrenData = '["Lala", "Sasa", "Dudu"]';
```

请注意，他们两个都是字符串，不知道你有没有觉得他们两个的内容似曾相识，对了，跟前面出现的json差不多呢，但现在这两个是字符串，所以不能用dad.name呀，children.length这样的调用，有没有想过如果把他们变成json就更好操作了呢？

Ext为我们提供了decode函数，可以把字符串转换成json。不过不是任意字符串，这个字符串必须是符合json格式的，即大括号，中括号，冒号，逗号这样的。看看具体要怎么做吧。

```
dad = Ext.decode(dadData);
children = Ext.decode(childrenData);
alert(dad.name);
alert(children);
```

就这样，Ext.decode(dadData)然后直接赋值给一个变量，然后就想怎么用就怎么用啦。哈哈~

或许你又问啦，为什么要把数据写到字符串里呢？这么麻烦，为啥不直接写成json呢？这个主要是因为http协议只能传输字符串，你用Ext.lib.Ajax.request的时候，回调函数success(response)如果返回的是json，你就要用到了：

```
Ext.lib.Ajax.request(
```

```
    'POST',
    'grid2.jsp',
    {success: function(response){
        var dad = Ext.decode(response.responseText);
    },failure: function(){}},
    'param=1'
);
```

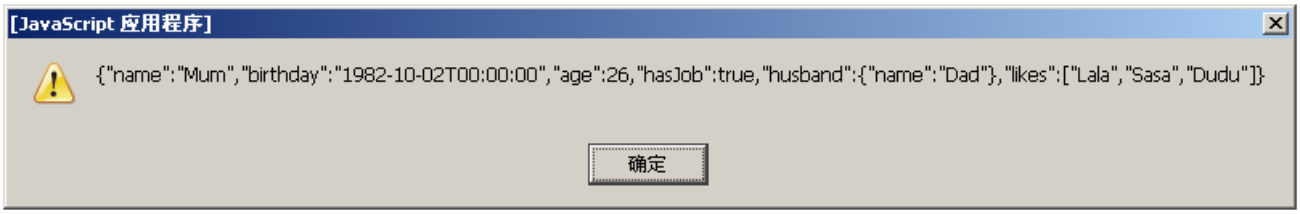
好的，response.responseText就是从后台返回的字符串，我们用Ext.decode把字符串解析成json，呵呵~

8.5.5. 反向操作，ext把json变成字符串

既然http需要字符串，如果我们想把json发送到后台，也需要先组装一下，你当然可以用循环，手工把json里的元素一个一个拼到字符串里，不过我还是建议你使用ext提供的函数。

```
var str = Ext.encode(mum);
alert(str);
```

生成的结果会是这样：



看到了吧？自动帮你生成一个符合json格式的字符串，里边包含了mum的所有数据，注意一下husband和children部分吧，引用部分的数据也都在。吼吼，方便呀。

你可能又要问我为啥这些json对象的name:value组合中的name部分要使用引号了，咱们之前不是一直没用吗？我解释一下，这纯粹是为了安全，name部分可能有空格啊，如果不用引号包起来绝对会有问题，人可以判断什么时候需要加引号，可程序没那么聪明，所以都加上引号以防万一啦。如下：

```
var spec = {
    'first name': 'Lingo',
    名字: '临远'
};
var specData = Ext.encode(spec);
alert(specData);
```

还要注意一点，http协议规定，url传递的参数只能包含ascii字符，中文是不能通过url正常传输的，所以就需要在发送url前，对数据进行编码，把中文这种非法字符转换成url规定的合法字符。

```
alert(encodeURIComponent(Ext.encode(spec)));
```

经它一转换，咱们的中文就变成了这个样子，虽然看着很晕，但是这样才能保证数据通过http正常传输的。

最后，Ext.lib.Ajax.request就变成了这个样子，同时调用Ext.decode和Ext.encode传输json和接收json。

```
Ext.lib.Ajax.request(
    'POST',
    'grid2.jsp',
    {success: function(response){
        var dad = Ext.decode(response.responseText);
    },failure: function(){}},
    'param=' + encodeURIComponent(Ext.encode(spec))
);
```

例子在lingo-sample/1.1.1/08-06.html和lingo-sample/2.0/08-06.html，东西太简单了，不看也罢。

json真的很简单，咱们这里说了一点儿基础，再附加上一点儿ext操作json的东东，应该足够啦，以后就看你自由发挥了。Good Luck。

8.6. 小声说说scope

js中this的使用是由来已久的问题了，不过我们这里可不会去讲什么大道理，我们只会结合具体的例子，告诉你们可能会遇到什么问题，在遇到这些问题的时候ext是怎么解决的。

一个超级常见的场景是使用Ajax回调函数的时候。

现在我们的html中有一个text输入框，一个按钮：

```
<input type="text" name="text" id="text">
<input type="button" name="button" id="button" value="button">
```

我们希望按这个按钮之后，使用ajax去后台读取数据，然后把后台响应的数据放到text里，流程是很简单的。一般都会想当然的这样做吧？

```
function doSuccess(response) {
    text.dom.value = response.responseText;
}

Ext.onReady(function(){
    Ext.get('button').on('click', function(){
```

```
        var text = Ext.get('text');
        Ext.lib.Ajax.request(
            'POST',
            '08-07.txt',
            {success:doSuccess},
            'param=' + encodeURIComponent(text.dom.value)
        );
    });
});
```

大家看到ajax之前已经使用Ext.get('text')获得了text，就以为后面也可以直接使用，就没想到success这种所谓的回调函数，是不会按照你写的顺序去执行的，当然也不会像你所想的那样，可以把局部变量text拿来就用，实际上，如果你什么都不做，光是这样用回调函数，你不得不再次使用Ext.get('text')重新获得元素，才能继续下面的错误，要不然浏览器就会告诉你text未定义。

在此使用Ext.get('text')重新获取对象还比较简单，不过有的情况下很不容易获得需要处理的对象，如何保存ajax之前获得的操作对象呢？我们有两个选择：scope和createDelegate。

1. 方法一：给ajax设置scope。

```
function doSuccess(response) {
    this.dom.value = response.responseText;
}

Ext.lib.Ajax.request(
    'POST',
    '08-07.txt',
    {success:doSuccess,scope:text},
    'param=' + encodeURIComponent(text.dom.value)
);
```

在ajax的callback参数部分添加一个scope:text，把回调函数的scope指向text，它的作用就是把doSuccess函数里的this指向text对象。好了，再把doSuccess里改成this.dom.value就大功告成了。

记得下次再想让回调函数里用什么对象，配上scope，然后就能用this随便乱用啦。

2. 方法二：为success添加createDelegate()。

```
function doSuccess(response) {
    this.dom.value = response.responseText;
}

Ext.lib.Ajax.request(
    'POST',
    '08-07.txt',
    {success:doSuccess.createDelegate(text)},
    'param=' + encodeURIComponent(text.dom.value)
);
```

createDelegate只能在function上调用，它把函数里的this强行指向我们需要的对象，然后

doSuccess里一样用this就可以了。嗯，作为你的备选答案咯。

如果让我选择的话，我会尽量选择scope，因为createDelegate毕竟还是要生成一大堆指向的东西，简单环境下scope就够用了，倒是createDelegate还有其他特技，比如修改调用参数什么的。呵呵~

例子在lingo-sample/1.1.1/08-07.html和lingo-sample/2.0/08-07.html。

8.7. 菜单和工具条

在一个面板的顶端或者底端放上一横条的按钮，点下每个按钮执行不同的操作。我们就管这个横条叫工具条了，放在工具条上面的那些东西，就是菜单了。

8.7.1. 至简至廉的菜单

虽然有点儿太简陋了，不过我们确实在一个工具条上放了4个按钮，看看代码，然后再听说书人娓娓道来呀。

```
// 创建工具条
var tb = new Ext.Toolbar();
tb.render('toolbar');

// 为工具条添加4个按钮
tb.add({
    text: '新建'
},{
    text: '修改'
},{
    text: '删除'
},{
    text: '显示'
});
```

先创建一个工具条，然后把工具条渲染到div上，调用Toolbar的add函数，添加4个按钮，咱们不需要特别指定是什么类型，默认的{}会转换成按钮的。

这个最最最简单的例子就在lingo-sample/1.1.1/08-08.html和lingo-sample/2.0/08-08.html里。

可惜，现在点击菜单上的按钮也不会有什么效果，想处理单击按钮的事件吗？给它们设置上handler就好了。

```
tb.add({
    text: '新建',
    handler: function() {
        Ext.Msg.alert('提示', '新建');
    }
},{
    text: '修改',
```

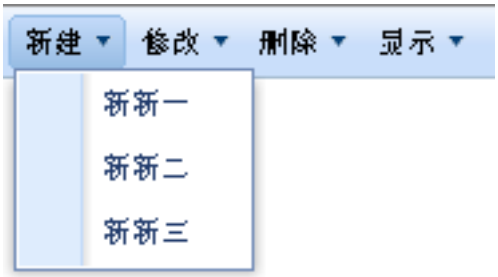
```
    handler: function() {
        Ext.Msg.alert('提示', '修改');
    }
},{
    text: '删除',
    handler: function() {
        Ext.Msg.alert('提示', '删除');
    }
},{
    text: '显示',
    handler: function() {
        Ext.Msg.alert('提示', '新建');
    }
});
```

给每个按钮放上handler处理函数，每点击一个按钮，都会弹出对应的对话框。哈哈~这不就好了吗？虽然我们只是弹出了个对话框，而实际上，你可以做在这个处理函数里做任何事情撒，只要你能想像到的。

例子在lingo-sample/1.1.1/08-09.html和lingo-sample/2.0/08-09.html上呢。

8.7.2. 丰富一点儿的多级菜单

我们上面做的就只是在工具条上装了几个按钮，如果要实现更多多的菜单选项该怎么做？就好像下面这样的效果呢：



其实，现在你看到的这个方方的东西才是真正的菜单呢，代码部分就要要先创建一个Ext.menu.Menu，然后把这个menu放到工具条里，就像这样：

```
var menu1 = new Ext.menu.Menu({
    items: [
        {text: '新新一'},
        {text: '新新二'},
        {text: '新新三'}
    ]
});

tb.add({
    text: '新建',
    menu: menu1
});
```

看到了没？text还是工具条上显示的文字，你设置了对应的menu参数，就会自动生成对应的菜单，这样工具条上就多了黑色的小小三角，点击之后蹦出来上边定义的菜单咯。

最有趣的是这些菜单都是可以嵌套滴，你任意搞几层都可以，咱们先弄个两层的看一看。

实现起来很简单，给menu里再加上menu就行了。

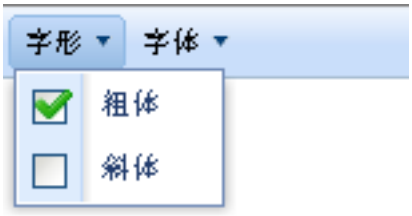
```
var menuHistory = new Ext.menu.Menu({
    items: [
        {text: '今天'},
        {text: '昨天'},
        {text: '一周'},
        {text: '一月'},
        {text: '一年'}
    ]
});

var menuFile = new Ext.menu.Menu({
    items: [
        {text: '新建'},
        {text: '打开'},
        {text: '保存'},
        {text: '另存...'},
        '-',
        {text: '历史', menu: menuHistory},
        '-',
        {text: '关闭'}
    ]
});
```

看到啦？想在哪里加上子菜单，直接使用menu指定就行。现在你可以随便打造自己的菜单啦，如果想看看我们的，1.x的例子在lingo-sample/1.1.1/08-10.html和08-10.html，2.0里例子在lingo-sample/2.0/08-10.html和08-11.html。

8.7.3. 单选多选，菜单里搞这套

别害怕，并不是要把checkbox或者radio放到工具条上，咱们看到的都是蛮漂亮的图片。

这个是多选：




```
var menuCheckbox = new Ext.menu.Menu({
    items: [
        new Ext.menu.CheckItem({
            text: '粗体',
            checked: true,
            checkHandler: function(item, checked) {
                Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '粗体');
            }
        }),
        new Ext.menu.CheckItem({
            text: '斜体',
            checkHandler: function(item, checked) {
                Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '斜体');
            }
        })
    ]
});
```

看一下啦，我们使用的是Ext.menu.CheckItem，text就是显示的文字，咱们早就司空见惯了，checked用来指定是不是被选中了，最后看看checkHandler这个不太一样的地方，与普通菜单相比多了一个checked参数，可以告诉我们这次点击是选中，还是取消。

下面我们再来看看单选：

其实呢，两种情况都是用的Ext.menu.CheckItem，唯一不同的是那个group参数，就好像这样。

```
var menuRadio = new Ext.menu.Menu({
    items: [
        new Ext.menu.CheckItem({
            text: '宋体',
            group: 'font',
            checked: true,
            checkHandler: function(item, checked) {
                Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '宋体');
```

```
            }
        }),
        new Ext.menu.CheckItem({
            text: '黑体',
            group: 'font',
            checkHandler: function(item, checked) {
                Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '黑体');
            }
        }),
        new Ext.menu.CheckItem({
            text: '楷体',
            group: 'font',
            checkHandler: function(item, checked) {
                Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '楷体');
            }
        })
    ]
});
```

嘿嘿~其实成了单选菜单，checkHandler中的checked肯定一直是true啦。group相同的那些CheckItem会自动切换，保证每次只选中一个，厉害吧？

例子在lingo-sample/1.1.1/08-12.html和lingo-sample/2.0/08-12.html。

8.7.4. 小把戏，定制好的菜单

选择日期呢，Ext.menu.DateMenu这个东西可以让咱们直接把日期选择加到菜单里。

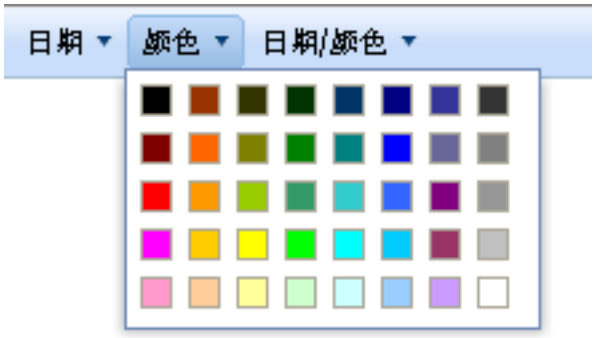
注意啦，这是一个Menu而不是Item，直接对应到menu就可以啦。

```
tb.add({
    text: '日期',
    menu: new Ext.menu.DateMenu({
        handler : function(dp, date){
            Ext.Msg.alert('选择日期', '选择的日期是 {0}.', date.format('Y年m月d日'));
        }
    })
```

```
});
```

咱们看看DateMenu对应的handler就行啦，它有两个参数DatePicker和date，第二个就是你选中的时间啦，好消息是这是一个date对象，你可以使用它定义的函数，坏消息是，你需要转换成你需要的形式才能正常显示。幸好ext内置了format函数，让我们更容易得到需要的格式。

选择颜色呢，Ext.menu.ColorMenu上场。



颜色选择框呢，不常用，但是很绚丽啦，用法跟日期菜单差不多，也是有个特定的handler，让我们可以直接获得选中的颜色，酷啊。

```
tb.add({
    text: '颜色',
    menu: new Ext.menu.ColorMenu({
        handler : function(cm, color){
            if (typeof color == 'string') {
                Ext.Msg.alert('选择颜色', '选择的颜色是 ' + color);
            }
        }
    })
});
```

不清楚为啥，这个handler会执行两次，第二次会输出成一个event，所以要加上个typeof进行判断，最后得到的颜色是一个6位的字符串，在它前头加个#就可以直接放到css里用啦。

例子在lingo-sample/1.1.1/08-13.html和lingo-sample/2.0/08-13.html里。

8.7.5. SplitButton让按钮和菜单结合

一般情况下，无论你按下菜单本身或者是他旁边那个小小黑色按钮，都会弹出menu。为了增强用户体验，我们可能实现一种更便捷的方法，让menu自动记忆上次咱们选择的项目，下次直接点这个菜单不会弹出让你选择，而是直接执行上次选择的效果。嘿嘿~如何啊？

这里我们就向您隆重介绍SplitButton，实际上也叫MenuButton，让我们来看看他是如何工作的。

```
var menuButton = new Ext.Toolbar.MenuButton({
    text: '选择一个工具',
    handler: function() {
        if(menuButton.menu && !menuButton.menu.isVisible()){
            menuButton.menu.show(this.el, menuButton.menuAlign);
        }
    },
    menu: {
        items: [
            {text: '直线', handler: line},
            {text: '矩形', handler: rect},
            {text: '圆形', handler: circle}
        ]
    }
});
```

描述一下咱们所做的：

1. 先构造一个Ext.Toolbar.MenuButton，它本身有两种状态，单击按钮本身会执行handler函数，单击右边的那个小箭头，会弹出菜单。

   我们在这里边用text指定显示的文字，用menu外加items告诉应该弹出什么来。好的，万事俱备，我们可以进入下一步了。

2. 第二步，让我们看看handler这个处理函数里边做了些什么？刚开始的时候因为咱们还什么都没选，所以让它直接把menu弹出来，一旦我们选择了任意一项，就在对应的菜单项中修改MenuButton的handler，下一次点它的时候就会直接执行咱们最后一次选择的操作了。

   比如，直线的handler就是这个了：

```
function line() {
    menuButton.handler = line;
    menuButton.setText('直线');
    Ext.Msg.alert('工具', '直线');
}
```

我是觉得SplitButton的功能很不错呢，你也可以完全不像我这样的用，看你的咯。例子都在lingo-sample/1.1.1/08-14.html和lingo-sample/2.0/08-14.html。

8.8. 蓝与灰，切换主题

其实，俺在这里说的不太准确呢。1.1.1里有四个主题，default, aero, vista, gray。而2.0就只有两种default和gray。

即使如此，动态修改css的原理依然相同，都是利用Ext.util.CSS.swapStyleSheet()函数，它会动态修改指定id的css标签，修改它指向外部css文件，借此修改整个项目的样式，于是主题也就改变了。

1. 第一步：制作一个select可以选择那几个主题呢，在里边我们可以选择这几个主题，嗯，其实就是主题对应css的名称。

```
<select id="themeSelect">
    <option value="default">default</option>
    <option value="aero">aero</option>
    <option value="vista">vista</option>
    <option value="gray">gray</option>
</select>
```

2. 第二步：为这个select添加监听change事件的函数。

```
Ext.get("themeSelect").on("change", function(e) {
    var v = e.target.value;
    if (v == 'default') {
        Ext.util.CSS.swapStyleSheet("theme", "");
    } else {
        Ext.util.CSS.swapStyleSheet("theme", "../../resources/css/xtheme-" + e.target.value + ".css");
    }
});
```

咱们使用的还是Ext.get()和on()，每次select的值发生修改时，先把主题名传递过来，然后swap吧。你这里也看到咱们是怎么拼这个文件名的。

3. 第三步，给html添加上个空的css标签，以备后用，这个东西现在虽然是空的，等到调用swap函数的时候就会乱变了。

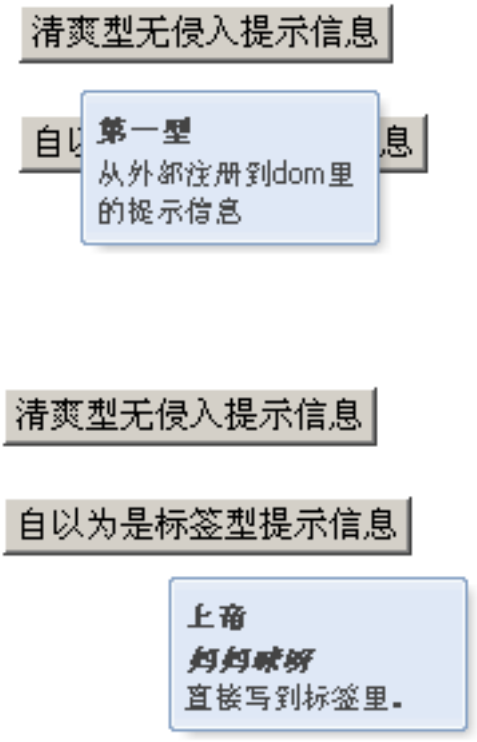
```
<link id="theme" rel="stylesheet" type="text/css" href="" />
```

行了，东西都有了，自己去乱换吧，我不管你啦。

1.x的代码在lingo-sample/1.1.1/08-15.html。2.0的代码在lingo-sample/2.0/08-15.html。

8.9. 悬停提示

当某人把鼠标傻傻的停在某个东东上，突然会发现跳出一个框框，框框里写了一些乱七八糟，不明就里的文字，也许是解释，也许是说明，也许是什么。



其实哩，QuickTip已经完渗入到ext控件体系了，grid的header，tree的node，form的field都可以见到它的身影。但要是它仅仅只能弹出个蓝色小窗窗，咱们就真没有讨论的必要了，实际上qtip的搞怪程度绝对超乎你的想像。

8.9.1. 起初，初始化

如果想使用这个提示功能，请务必记得先使用Ext.QuickTips.init();函数对整个系统进行初始化，有了它我们才可以激活grid, tree, form还有咱们自己定义的那些qtip。

```
Ext.QuickTips.init();
```

这个函数对ext的提供体系做初始化后立刻激活这些提示功能，必须必须在其他操作之前执行，其他操作指的disable()禁用提示功能，enable启用提示功能，register给一个dom元素注册提示功能，unregister取消提示功能，所有的这些都需要在初始化之后进行呢。当然咱们可能不会用到这么多东西，只要记得先执行一下init()就好了。

8.9.2. 诞生，注册提示

既然已经说到了register()，咱们就来看看如何给一个dom元素添加上这些提示功能。

咱们首先需要一个<input id="q1" type="button" value="清爽型无侵入提示信息" />，说实话，我们需要的只有id而已，其他的都是摆设。

现在我们就给这个button弄个提示框玩玩，对了，记得先初始化哟。

```
Ext.QuickTips.init();
```

```
Ext.QuickTips.register({
    target: 'q1',
    title: '第一型',
    text: '从外部注册到dom里的提示信息'
});
```

register()函数的最简形式只有三个参数，target指定给哪个dom元素补充提示，title和text都显示在提示框了，只不过title默认加粗并且放到上边，text算是内容啦，不用太在意所谓的位置，你可以在它里边使用html控制格式呢。

经此磨练，只要把鼠标放到按钮上一会儿，就会出现这个提示啦。

8.9.3. 分支，标签提示

统一配置易于维护一直我们追求的东西，以咱们这些正常人的眼光看来，把html和提示信息分开写明显是不智的行为，还是让咱们把这些放到一起吧。

即使我们可以把提示信息结合到html中，初始化过程也是必要的，别忘了啊。然后我们就可以来看看怎么直接给html加提示了。

```
<input type="button" value="自以为是标签型提示信息"
    ext:qtitle="上帝"
    ext:qtip="<b><i>妈妈咪呀</i></b><p>直接写到标签里。</p>" />
```

这些配置是以ext:作为前缀的，qtitle代表提示标题，qtip代表提示内容，这些配置会在初始化的时候解析到ext中，并在需要的时候弹出来。

在看到QuickTips之前，我一直认为dojo中这些不符合html标准的attribute是极不合理的，你每次都要先设置html，再去编写js，让他们两个对应起来的同时，让你在开发和维护的时候都要付出双倍，甚至更多的努力。但在extjs使用这种方法配置提示的时候，我却觉得异常欣慰，在正确的时机做正确的事情才是最重要的。

8.9.4. 发展，全局配置

默认情况下，提示信息在鼠标悬停0.5后显示，显示5秒后消失。很明显有的时候我们需要修改它的默认配置，最简单的办法就是使用Ext.apply()覆盖QuickTips的默认属性：

```
Ext.apply(Ext.QuickTips, {
    maxWidth: 200,
    minWidth: 100,
    showDelay: 50,
    dismissDelay: 1000,
    hideDelay: 500,
    trackMouse: false,
    animate: true
});
```

从上往下看看这些设置都是些什么意思：

1. maxWidth，minWidth，提示框的最大宽度和最小宽度，这会导致提示内容的自动换行哟。
2. showDelay是鼠标悬停多久后显示提示，默认是0.5秒，这里的时间都是用ms做单位的，1000才相当于一秒。
3. dismissDelay告诉咱们这个提示框一共能显示多长时间，默认5秒，如果打算看到永不消失的电波，请直接设置成0。
4. hideDelay是提示逐渐消失的时间，默认0.2秒。
5. trackMouse可有意思了，如果你把它设置成true，提示窗蹦出来以后也不会老实，它会跟着鼠标乱动滴。
6. autoHide一般都是true的，它会根据dismissDelay和hideDelay逐渐消隐提示框，初看起来似乎也是一条让提示框永不消失的捷径，不过你用了以后就知道了，autoHide:false的结果是鼠标离开了dom提示也不消失，这个提示就这么一直挂在上边了，太可怕了。
7. animate:true动画效果，喜欢玩就打开吧。

8.9.5. 进化，个体配置

这些参数既可以全局配置，也可以单独指定，用来为某一个dom的提示信息设置自己的配置呢。

既然说到单独配置，当然也要分成使用register()和直接写到html里两种方式了，名称稍稍有些不同，但也有规律可循。

1. register()里用的参数跟全局配置一样，毕竟都是js嘛。

```
Ext.QuickTips.register({
    target: 'q1',
    title: '第一型',
    text: '从外部注册到dom里的提示信息',
    maxWidth: 200,
    minWidth: 100,
    showDelay: 50,
    dismissDelay: 1000,
    hideDelay: 500,
    trackMouse: false,
    autoHide: false,
    animate: true
});
```

   这里的autoHide:false的情况有所不同，单独设置autoHide:false，出现的提示不会自动消失，但会提供给用户一个关闭按钮，唉，全局设置这个参数就不行。

2. 标签写法就有些不同了，其实我倒觉得搞成一样就好了，何必另做一套呢？可是它已经这么做了，咱们也没办法。

```
```

```
<input type="button" value="自以为是标签型提示信息"
    ext:hide="user"
    ext:qclass=""
    ext:qwidth="300"
    ext:qtitle="上帝"
    ext:qtip="<b><i>妈妈咪呀</i></b><p>直接写到标签里。</p>" />
```

标签式设置有限制呢，咱们只能使用5个参数，ext:hide="user"的时候跟autoHide:false是一个效果，其他的倒是都容易理解，不用讲也明白了。

呼，绕了一圈也算把咱们需要的功能都看到了，怎么样，是不是对提示系统有了改观呢？例子在lingo-sample/1.1.1/08-16.html和lingo-sample/2.0/08-16.html，代码一模一样，不多说了。

8.10. 灵异事件，Ext.state

奉劝大家，如果搞不明白state是做什么的，千万别乱用啊，要不然就像是撞到鬼了，蹦出来一大堆诡异的问题。

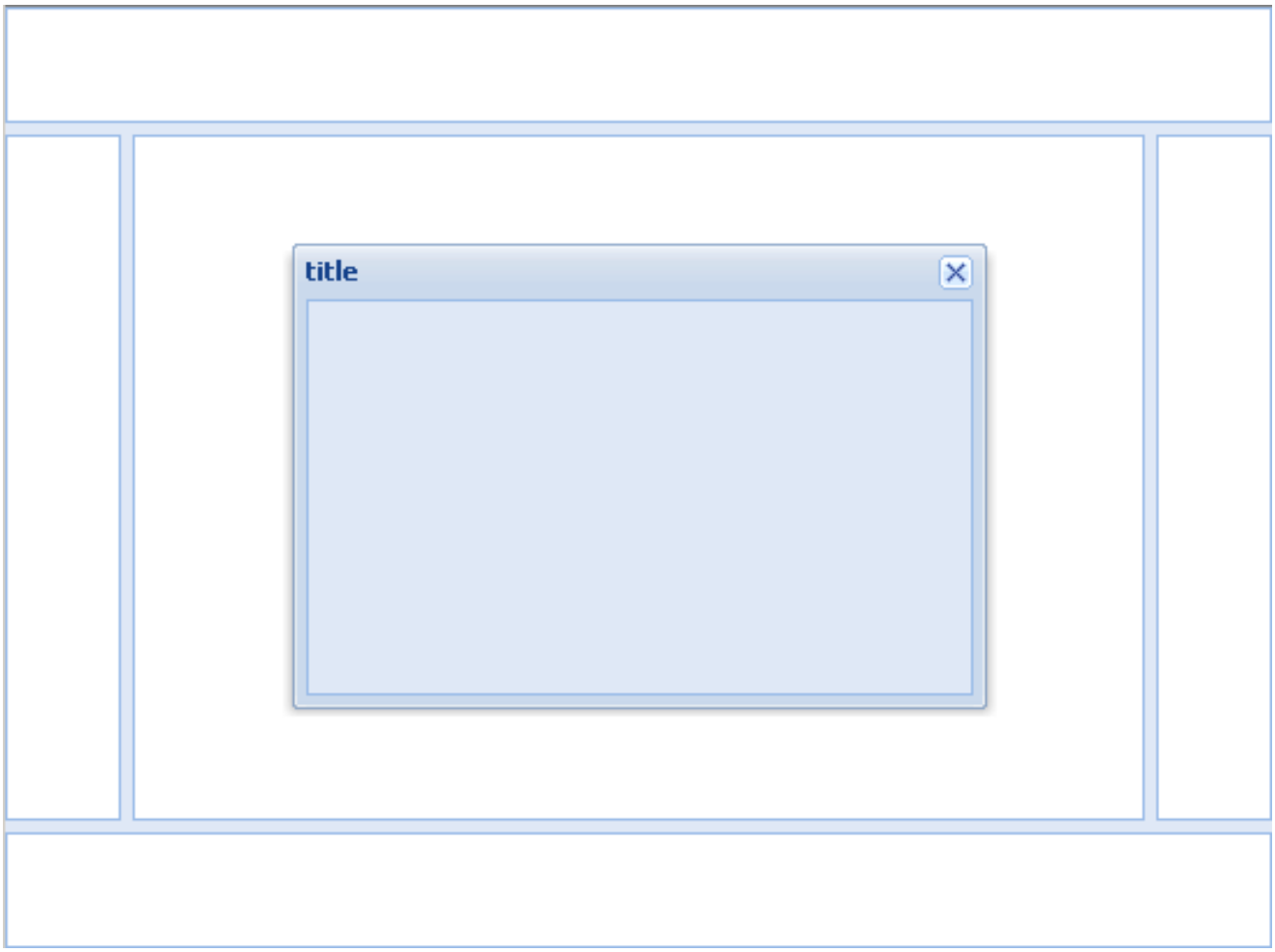
这个state用讲的还是太难理解了，但是只要见过它造成的事故，估计不用我说什么你也忘不掉了。

嗯，作为比对，我们在页面中放一个分成五块的ViewPort布局，再弹出一个弹出窗口来。

```
Ext.onReady(function(){
    var viewport = new Ext.Viewport({
        layout:'border',
        items:[{
            region: 'north',
            height: 50,
            split: true
        },{
            region: 'south',
            height: 50,
            split: true
        },{
            region: 'west',
            width: 50,
            split: true
        },{
            region: 'east',
            width: 50,
            split: true
        },{
            region: 'center'
        }]
    });

    var win = new Ext.Window({
        title: 'title',
        width: 300,
        height: 200
    });
    win.show();
```

```
});
```

上下默认高度50，左右默认宽度50，弹出窗口宽300，高200，默认出现在浏览器中央。看起来就像这样：



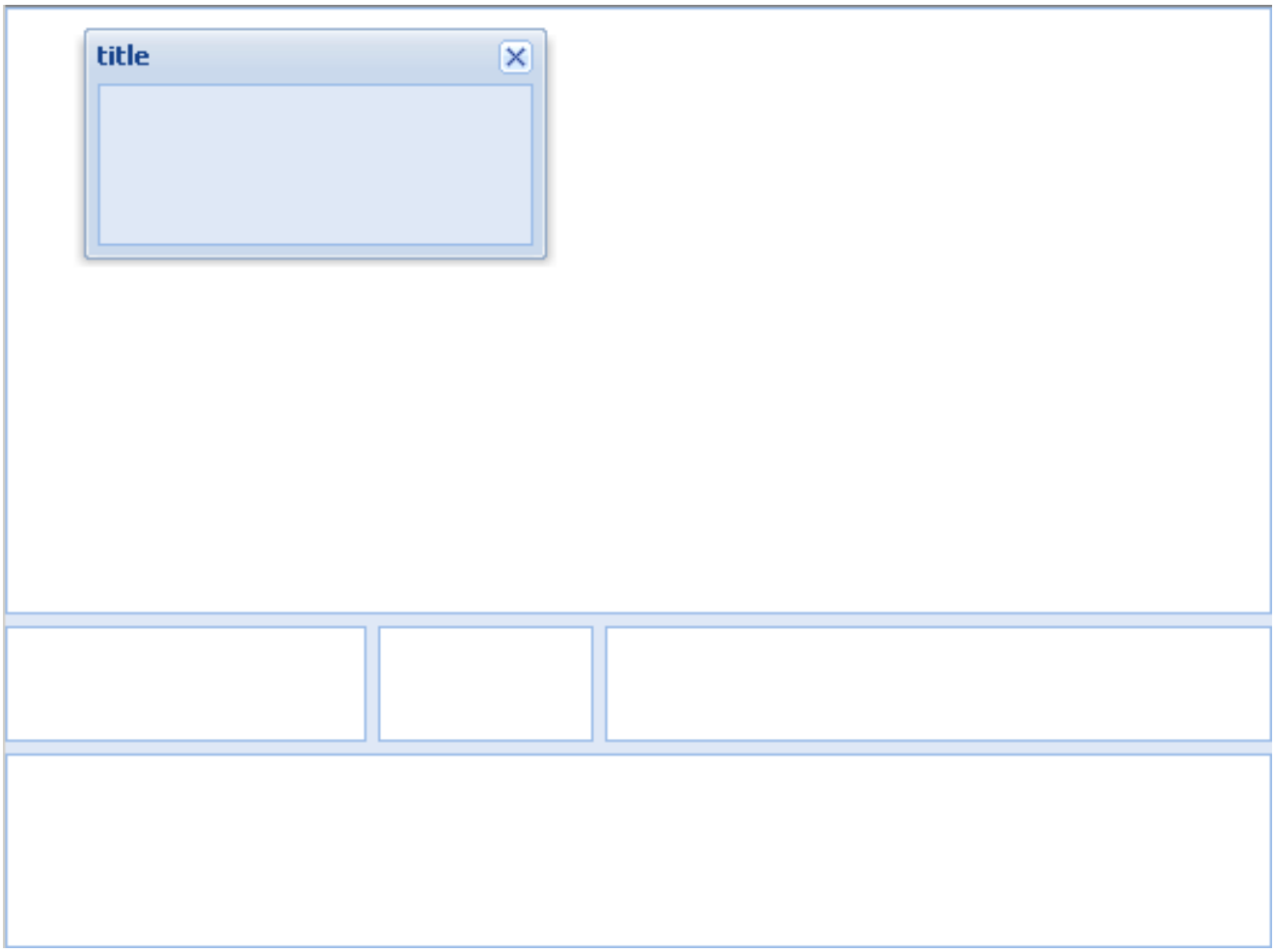
上下左右都设置上了split，宽高就可以让用户拖拽着修改了，弹出窗口本身既修改大小又可以拖拽移动。不过你再怎么拖拽，再怎么修改，刷新一下页面所有东西就又回复原装了。

嗯，至此为止一切都还正常，现在我们加上state。

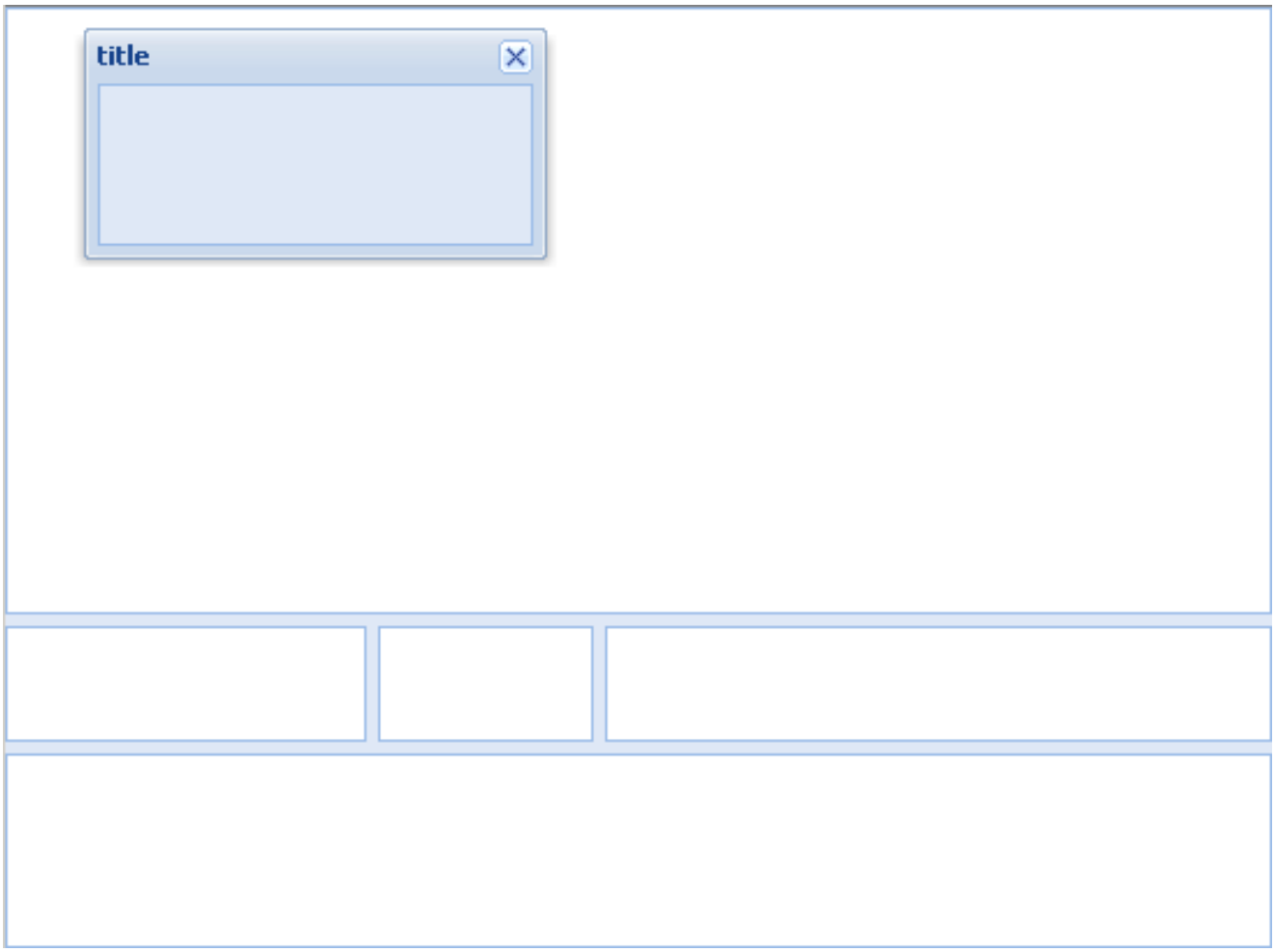
```
Ext.state.Manager.setProvider(new Ext.state.CookieProvider({}));
```

记得这句要加载viewport和window之前，也就是说初始化必须放在最前面，这个跟Ext.Quicktips用法差不多。

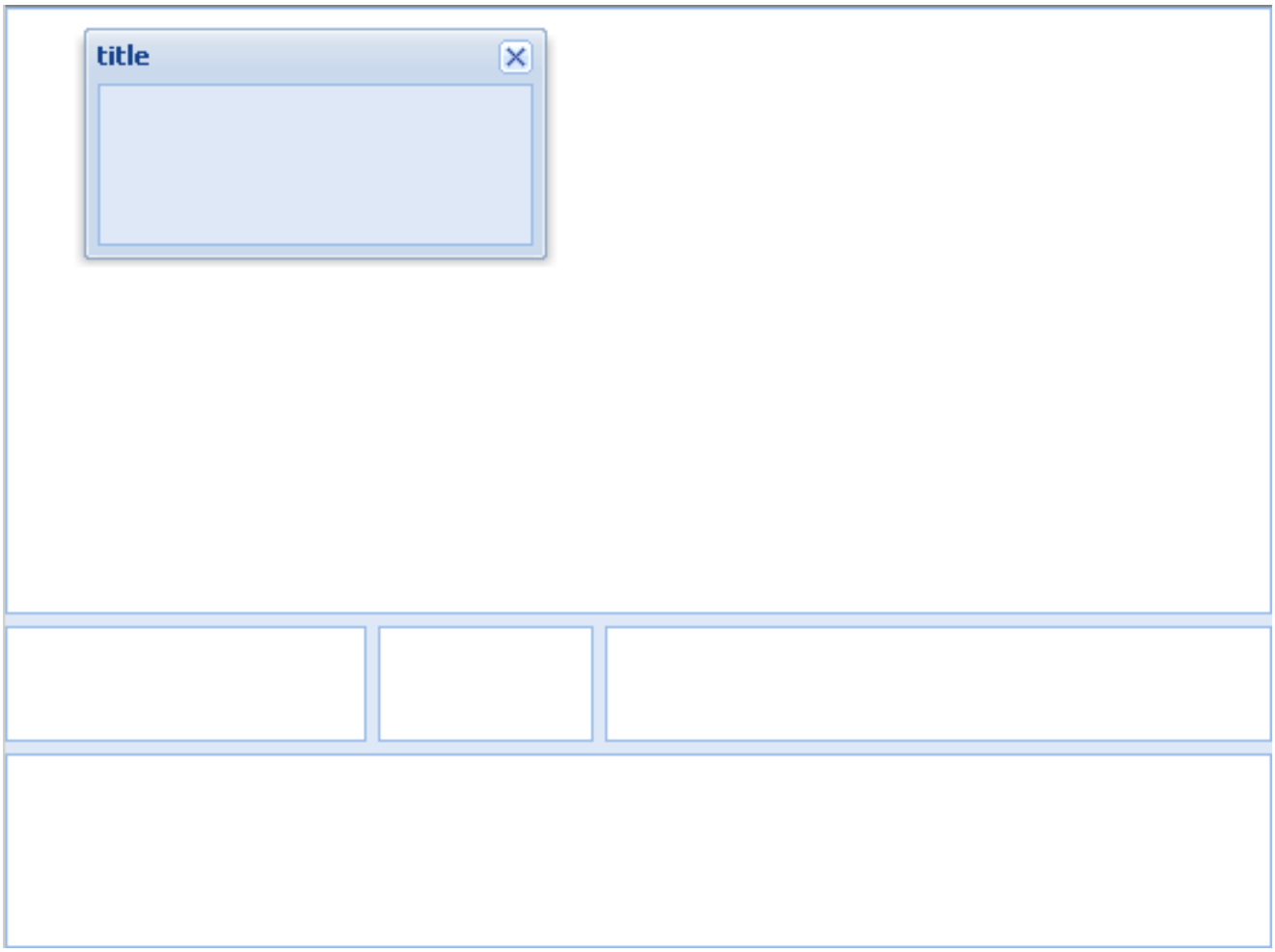
好了，现在咱们再拖拽几下吧，嗯，也是没什么问题。



现在刷新一下页面呢？



竟然，没变？是不是眼花了？还是没刷新好？那咱们把浏览器关掉再打开这个页面试试。



见鬼了，显示的还是上次修改的布局，修改后的高和宽，修改后的窗口大小，修改后的窗口位置，初始化的数据都没用了。

有经验的朋友估计怀疑到页面缓存了，慌忙跑去清楚缓存，哈哈~不管你怎么清除缓存，只要不清空cookie，这个页面的布局就不会变啦，只要cookie还在，你的初始化数据就不会起作用，看到的都是最后修改的样子。

看看cookie里有什么玄机哟，咱们用的firefox查看cookie哟。

这些ys开头的就是ext为咱们保存的数据了，删掉他们试试，再刷新一下，看是不是布局窗口都回到原位了？唉，知道了就没什么神秘的咯。

实际上蛮简单的，Ext.state是由Manager和Provider两部分组成滴，manager是一个单例，你就不用再去创建他了，啥时候需要它，直接往它里边设置provider就好。这个provider是实际进行保存和读取数据的东东，不过ext只给咱们提供了一个Ext.state.CookieProvider，把状态都保存到用户浏览器的cookie里。默认呢cookie会保存一周，path和domain都取当前位置的path和domain，咱们刚才是在本地打开的网页，domain就是空的咯。

如果你需要修改设置，就这样做呢。

```
var cp = new Ext.state.CookieProvider({
    path: "/cgi-bin/",
    expires: new Date(new Date().getTime()+(1000*60*60*24*30)), //30天
    domain: "extjs.com"
});
Ext.state.Manager.setProvider(cp);
```

path啦，domain啦，是字符串呢，expires这个过期时间直接是当前时间加上了七天，需要多长时间你都可以自己算啦。

ext在运行的时候，支持state的组件都回去判断Ext.state.Manager是否进行了初始化，如果有必要，就去调用state中的数据对自己的属性进行修改呢，顺便提一句，所有Ext.Component都支持state的。

你也看到啦，state中保存的是以ys加上组件id作为cookie的key来保存数据的，这个id怎么看都是自动生成的，既然是自动生成的，就可能重复，重复的话，就会出错。想想吧，如果以前用过一次CookieProvider，然后没注意，又在其他页面里用上了它，结果第二页里就会把上次保存的状态也都取出来了，要是正好碰上id一样的情况，那就好玩了。你的window可能得到以前设置的ViewPort的属性，这个宽度，高度就错的离谱了。

为了规避这个问题，别让ext自己去胡乱匹配，咱们给你推荐一个stateId参数，好好想上一个不容易

重复的名字吧。这样就不容易出问题咯。

```
{
    stateId: 'viewPortNorth',
    region: 'north',
    height: 50,
    split: true
}
```

配上个viewPortNorth，cookie里对应的key就变成ys-viewPortNorth咯，估计不容易冲突了呢。

这个state造成的灵异事件可是非常诡异的，不过又不失为一种客户自定义模板的办法呢。不管这些了，先看看例子咯。

1.x的例子在lingo-sample/1.1.1/08-17.html，2.0的例子在lingo-sample/2.0/08-17.html。

8.11. 所谓的事件

ext中遵循一种单根的事件模型，你去看吧，只要有哪个控件继承了Ext.util.Observable，那么它肯定是可以支持事件了。你可以给这个对象定义上一堆事件，然后给这些事件配置上监听器，到某个事件被触发的时候，会自动去调用对应的监听器，所有这些就是ext的事件模型了。

咱们也继承这个Ext.util.Observale，实现一个支持事件的对象好不？

```
Person = function(name) {
    this.name = name;
    this.addEvents("walk", "eat", "sleep");
}
Ext.extend(Person, Ext.util.Observable, {
    info: function(event) {
        return this.name + ' is ' + event + 'ing.';
    }
});
```

这里对象叫做Person，有一个属性name，初始化的时候调用this.addEvents()函数定义上三个事件，walk, eat, sleep，最后使用Ext.extend()让Person继承Ext.util.Observable的所有属性，对了，咱们还加上了一个函数info()，让它返回Person的信息。

嗯，工作准备就绪了，接下来如何使用呢？

让我们在html创建一个Person的实例，然后为它的事件都配置好监听器吧。

```
var person = new Person('Lingo');
person.on('walk', function() {
    Ext.Msg.alert('event', person.name + "在走啊走啊。");
});
person.on('eat', function(breakfast, lunch, supper) {
    Ext.Msg.alert('event', person.name + "要吃" + breakfast + ", " + lunch + "和" + supper + "。");
});
person.on('sleep', function(time) {
```

```
    Ext.Msg.alert('event', person.name + "从" + time.format("H") + "点开始睡觉啦。");
});
```

这里的on()是addListener()的简写形式，功能完全一样，第一个参数传递事件名，第二个参数是事件发生时执行的函数，咱们这里为求简便全部写成alert形式了。

好了，准备工作都已就绪，现在等这些事件发生就可以看到效果了。可怎么让事件发生呢？为了便于控制咱们弄三个按钮，按钮按下后就触发对应的事件。

```
Ext.get('walk').on('click', function() {
    person.fireEvent('walk');
});

Ext.get('eat').on('click', function() {
    person.fireEvent('eat', '早餐', '中餐', '晚餐');
});

Ext.get('sleep').on('click', function() {
    person.fireEvent('sleep', new Date());
});
```

好的，调用下fireEvent()就会触发事件了，传入一个事件名作为参数就好，这个事件对应的那些监听函数就去执行啦。

哼哼，下面是只有动态语言才能享受到的大优惠，你想给监听方法传递什么参数，直接写到fireEvent()里就行了，不用管参数数量，不用管参数类型，字符串，数字，日期，数组，想传什么就可以传什么，不过你可要确定监听函数里可以处理你传递过去的参数呢，灵活的同时也暴露出确定，万一监听参数需要某个参数，触发事件的时候却忘了传递过去，十有八九就报错了，这种非直观错误才就难查啦。

正如on是addListener的简写一样，removeListener的简写形式叫做un。你可以用它删除某个事件对应的监听函数。

```
var fn = function() {
    // TODO:
}
person.on('walk', fn);
person.un('walk', fn)
```

还有一个purgeListeners()函数，可以把所有的监听器都删掉，注意啦，是所有的监听器，你甚至不能指定清除某个事件的监听器，一旦调用了这个函数可就轻松啦，所有的事件都不起作用了。慎用慎用。

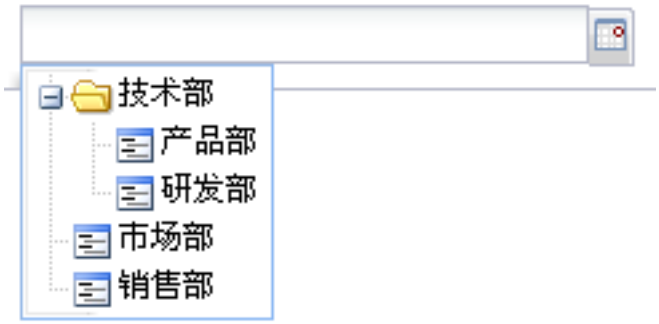
例子在lingo-sample/1.1.1/08-17.html和lingo-sample/2.0/08-17.html。

第 9 章 沉寂吧！我们要自己的控件。

免费给的东西没人珍惜。介于此，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。

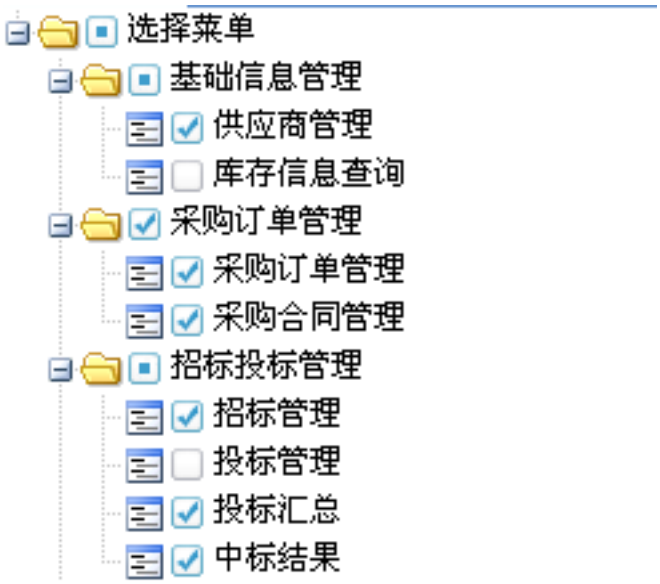
9.1. 下拉树形选择框TreeField

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。

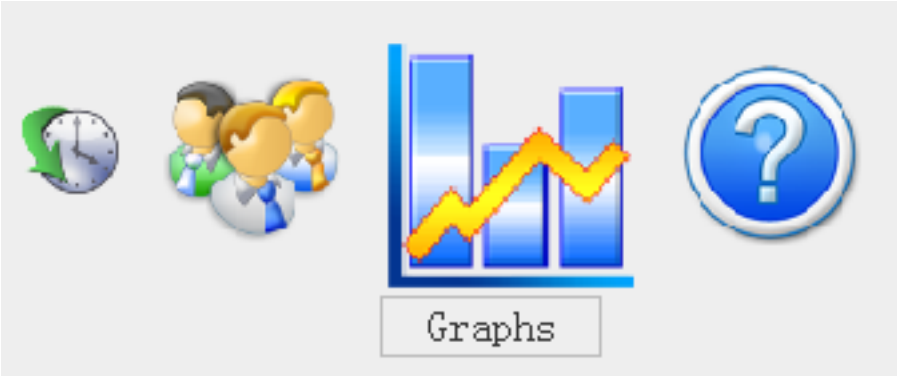


9.3. 带全选的checkbox的grid

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

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

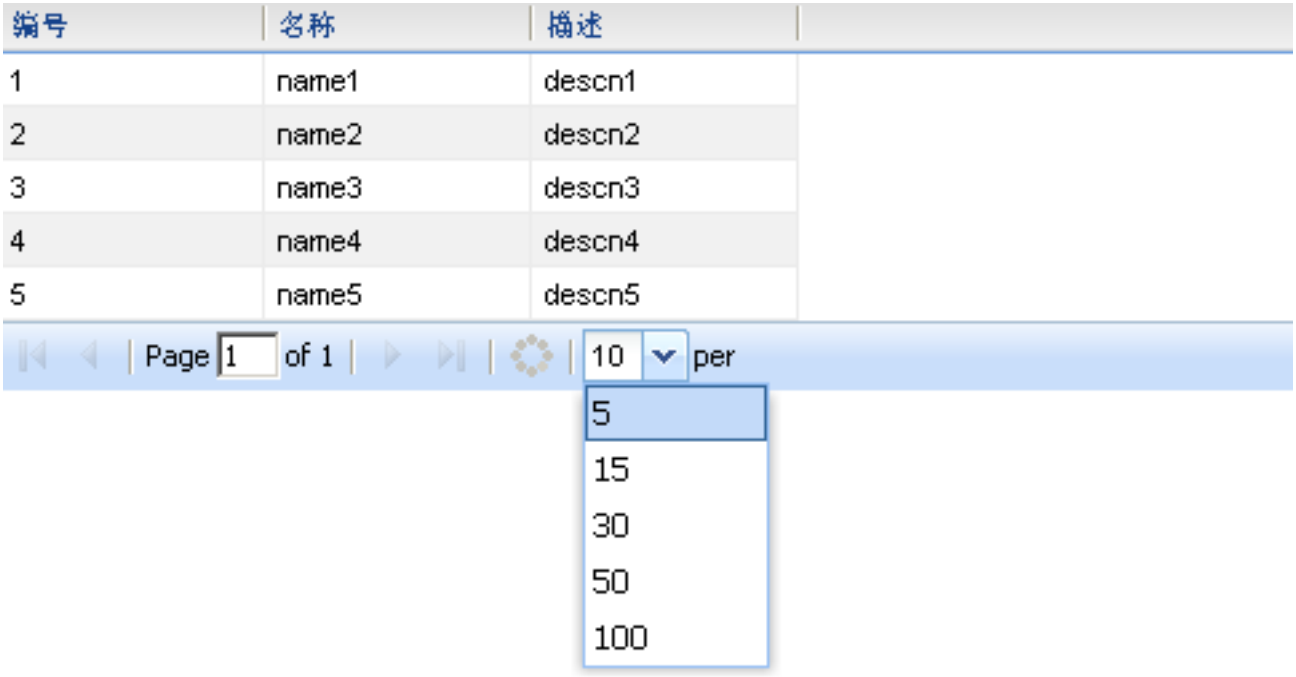
2007-12-05土豆新货上架

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

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

第 10 章 撕裂吧！邪魔外道与边缘学科。

这里都是不能够通用的东西，本来extjs应该与后台开发语言无关的，只要后台能处理返回ext需要的数据库格式，它就可以处理。不过，通用的毕竟没有专用的方便，所以我们还是要说说咋跟平台绑定。

以下的内容不会适合所有读者，请自行选择阅读。

10.1. dwr与ext整合

据统计，玩ajax的java程序员，有一大半都使用dwr。嗯，如果你没听说过dwr可以去网上随便搜一下。dwr的官方网站是：http://getahead.org/dwr/。

下面的部分不包括讲解如何使用dwr，如果你还不能熟练使用dwr，请先去补习。

10.1.1. 无侵入式整合dwr和ext

因为dwr在前台完全就是一个普通的js，所以我们不需要特地去做些什么，直接使用ext调用dwr的函数就可以了。

以grid为例，现在我们要显示一个通讯录的信息，后台记录的数据有：自动生成的id，名字，性别，邮箱，电话，加入时间，备注。

编写对应的pojo：

```
public class Info {
    long id;
    String name;
    int sex;
    String email;
    String tel;
    Date addTime;
    String descn;
}
```

然后写个操作pojo的manager类：

```
public class InfoManager {
    private List infoList = new ArrayList();

    public List getResult() {
        return infoList;
    }
}
```

代码部分有些删节，只保留关键部分让咱们讨论就可以了，就这样把这两个类配置到dwr.xml中，让前台可以对他俩进行调用。

下面是ext与dwr配合的关键部分，我们要对js部分做如下修改：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'性别',dataIndex:'sex'},
    {header:'邮箱',dataIndex:'email'},
    {header:'电话',dataIndex:'tel'},
    {header:'添加时间',dataIndex:'addTime'},
    {header:'备注',dataIndex:'descn'}
]);

var ds = new Ext.data.JsonStore({
    fields: ["id","name","sex",'email','tel','addTime','descn']
});

// 调用dwr取得数据
infoManager.getResult(function(data) {
    ds.loadData(data);
});

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

注意力放在中间的部分吧，执行infoManager.getResult()函数的时候，dwr就是在ajax去后台取数据了，操作成功后调用咱们定义的匿名回调函数，在这里我们只做一件事，那就是将返回的data直接注入到ds中。

非常幸运的是，dwr返回的data可以被JsonStore直接读取，设置了fields参数告诉我们需要哪些属性，最后就大功告成了。

在这里，ext和dwr两者之间毫无瓜葛，你将dwr或者ext中的任何一方替换掉都是可行的。实际上他们只是运行在一起，并没有整合的意思呢，不过在此给出这个例子，也是说明了一种松耦合的可能性，实际操作中完全可以使用这种方式呢。

10.1.2. DWRProxy

ext和dwr完全可以不做任何修改的袖筒合作，不过你还要考虑很多细节，比如grid分页，刷新，排序，搜索，考虑到这些通用的操作，extjs.com上有人放出了DWRProxy，借助它让dwr和ext衔接的更加紧密，不过需要在后台添加DWRProxy所需要的java类，这可能不是最好的解决方案，但相信通过对它内在实现的讨论，可以给我们更多的选择和想像空间。

注意啦，注意啦，这个DWRProxy.js可是要放在ext-base.js和ext-all.js后边滴，要会报某某不是构造方法的错误哟。

我们现在就用DWRProxy来做个分页看看，除了准备好DWRProxy.js这个插件之外还要在后台专门准备一个分页的封装类，因为我们不只要告诉前台显示哪些数据，还要告诉前台咱们一共有多少条数据。现在让我隆重向您介绍ListRange.java。

```
```

```
public class ListRange {
    Object[] data;
    int totalSize;
}
```

其实ListRange非常单纯，只有两个属性，提供数据的data和提供数据总量的totalSize。再看一下InfoManager.java，为了实现分页我们专门写上一个getItems方法，它大概是这个样子：

```
public ListRange getItems(Map conditions) {
    int start = 0;
    int pageSize = 10;
    int pageNo = (start / pageSize) + 1;

    try {
        start = Integer.parseInt(conditions.get("start").toString());
        pageSize = Integer.parseInt(conditions.get("limit").toString());
        pageNo = (start / pageSize) + 1;
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    List list = infoList.subList(start, start + pageSize);
    return new ListRange(list.toArray(), infoList.size());
}
```

getItems()的参数是Map，我们从它里边获得咱们需要的参数，比如，start啦，limit啦。不过你也知道http里的参数都是字符串，咱们需要的是数字，所以对类型要转换下，最后根据start和limit两个属性从全部数据中截取一部分出来，放进新建ListRange中，简单把ListRange推回前台，于是一切都解决了。

现在重头戏来了，我们要确实用上这个传说中的Ext.data.DWRProxy了，嗯，还有那个Ext.data.ListRangeReader，通过这两个扩展，我们的ext可以完全支持dwr的数据传送协议了，实际上这正是ext要把数据和显示分离设计的原因，这样你改个proxy改个reader，不改其他的就实现了功能哟，多好呢？后台还是你熟悉的dwr，So...至少在grid部分，你可以享受人生了。

唉，说了这么多废话，还是让主角上场吧。

```
var ds = new Ext.data.Store({
    proxy: new Ext.data.DWRProxy(infoManager.getItems, true),
    reader: new Ext.data.ListRangeReader({
        totalProperty: 'totalSize',
        root: 'data',
        id: 'id'
    }, info),
    remoteSort: true
});
```

跟我们上头说的一样，我们改了改proxy，再改了改reader，其他地方不用管grid就正常运转了。需要提醒的只有DWRProxy的用法，两个参数哦，第一个叫dwrCall，把一个dwr函数放进去，这个东东对应的是后台的getItems方法，第二个参数叫pagingAndSort，说真的，这个参数应该能控制咱们的dwr是否需要分页和排序，但在js代码里怎也没看到对应的功能，莫非是我眼花了不成？？不过也可能是我

自己改错了，我都不知道哪个才是最原版的DWRProxy了，呼呼，以后等DWRProxy这个扩展升级了再看看吧。

ListRangeReader部分与我们后台的那个ListRange.java对应，你没看到totalProperty吗？那个就是总数了，我们告诉它ListRange里totalSize这个属性就是总数了。还有root，我们也告诉它在ListRange里数据变量的名字叫data。嘿嘿~然后它会自己对应起来滴，这个主要是为了人类的方便，如果你哪天不想用咱们给你的ListRange.java，等你造好自己的轮子以后，把这两个属性对应起来就好咯。吼吼。

10.1.3. DWRTreeLoader

第一次撒，第一次，尝试让树形也支持支持dwr，虽然说不上稀奇，不过扩展总是让人心动，不是吗？嘿嘿~即使只报着猎奇的心思，也要探个究竟，大不了不好用再改嘛，我们可一定要拥抱变化。

啦啦啦~这个更简单啦，后台我们需要树形节点对应的TreeNode.java，目前呢，只要id，text，leaf三项就可以了。

```
public class TreeNode {
    String id;
    String text;
    boolean leaf;
}
```

id是节点的唯一标记，唯一哟，知道了id就知道你在搞哪个点了。text是显示的标题，呼呼，leaf比较重要呢，怎么用它标记这个节点是不是叶子。

啦啦啦，还是异步树，TreeNodeManager.java里的getTree()方法将获得一个节点的id做参数，然后返回这个节点下的所有子节点，咱们这里没节制的乱生成啦，别太在意，等轮到你的时候，你可以做任何想做的东西呢。TreeNodeManager.java里大概像这个样子：

```
public List getTree(String id) {
    List list = new ArrayList();
    String seed1 = id + 1;
    String seed2 = id + 2;
    String seed3 = id + 3;
    list.add(new TreeNode(seed1, "" + seed1, false));
    list.add(new TreeNode(seed2, "" + seed2, false));
    list.add(new TreeNode(seed3, "" + seed3, true));

    return list;
}
```

看，很简单吧，其实List和数组都一样啦，反正到了前台都是一样。嘿嘿~

js那部分更简单，引入DWRTreeLoader.js，把TreeLoader一换就行了。看看吧：

```
var tree = new Ext.tree.TreePanel('tree', {
    loader: new Ext.tree.DWRTreeLoader({dataUrl: treeNodeManager.getTree})
});
```

```

```

还是dataUrl，不过这次对待的是一个dwr的函数哩，不过不用太在意啦，反正用的时候它内部会自动处理。嘿嘿~感觉dwr有时候还真是不错呢。

啦啦啦~不才在下有时候也想总结总结，dwr虽然绑定到了java平台上，但是带来的便利可不是一点儿半点儿，说实话，dwr可比json-lib好用多了，而且dwr返回的东东可以直接当json用。如果你用json-lib还要面对无休无止的循环引用，无限循环，唉，有时候觉得还是搞成dwr好啦。不过我用得还是少，跨平台的恶念总是侵扰偶的思想，土豆也说DWRProxy不是好东东，路遥知马力，日久见人心，咱们还是静观后效吧。

这次的例子复杂一些，因为包括lib呀，class呀，xml呀，jsp呀，我统统放在lingo-sample/1.1.1/dwr1/下了，自己看看呗。感谢上帝2.0只需要改改js，后台完全不用动，扩展也没啥可改的，对应的代码放在lingo-sample/2.0/dwr2/下。

10.2. localXHR让你在不用服务器就玩ajax

ajax是不能在本地文件系统使用的，你必须把东东放到服务器上，不管是iis, apache, tomcat，或者是你熟悉的什么东西，反正只要是支持http协议，就可以用ext里的ajax。

至于本地为啥用不了ajax，其实主要是因为ajax要判断http返回的相应状态，只有status=200的时候才认为这次请求是成功的，所以呢，localXHR做的就是强行修改响应状态，假装一个成功的状态就让ajax老老实实干活了。呵~

介于天外小人的意愿，咱们把localXHR.js开膛破腹地研究一下，就让我们来瞧瞧这212行代码里有什么奥秘。

1. 加入了一个forceActiveX属性，默认是false，估计是控制是否强制使用activex，暂时不知啥用处。

2. 修改createXhrObject函数，只是在最开头加了一句判断的语句：

```
if(Ext.isIE7 && !!this.forceActiveX){throw("IE7forceActiveX");}
```

   这个函数本来是创建ajax对象的，看起来似乎是ie下不支持activex，会抛出异常。没用过ie7，不晓得呢。

3. 增加了getHttpStatus函数，看来就是为了集中搞定http的响应状态。

```
getHttpStatus: function(reqObj){
    var statObj = {
        status:0
        ,statusText:''
        ,isError:false
        ,isLocal:false
        ,isOK:false
    };
    try {
        if(!reqObj)throw('noobj');
        statObj.status = reqObj.status || 0;
```

```
        statObj.isLocal = !reqObj.status && location.protocol == "file:" ||
                          Ext.isSafari && reqObj.status == undefined;

        statObj.statusText = reqObj.statusText || '';

        statObj.isOK = (statObj.isLocal ||
                        (statObj.status > 199 && statObj.status < 300) ||
                         statObj.status == 304);

    } catch(e){
        //status may not avail/valid yet.
        statObj.isError = true;
    }

    return statObj;
},
```

它为状态增添了更多语义，status状态值，statusText状态描述，isError是否有错误，isLocal是否在本地，isOK是否成功。

对isLocal是否本地的判断分成两种，如果reqObj没有status，而且请求协议是file:。或者浏览器是safari而且reqObj.status没有定义的时候，就是在本地了。

是否成功的条件挺多，如果在本地，或者响应状态在199到300之间，或者是304，多说一句304是not modified，就是成功啦。

如果出现异常，就是错误。最后返回去就行。

4. 对handleTransactionResponse的修改，参数数量都是三个，但是第一个参数的含义肯定不同，你可以在ext-base.js里看到很多状态值200啦，300呀。可在localXHR.js里只能看到isError，isOk。哼，我想localXHR.js里用的应该就是getHttpStatus返回的对象了。

   从体积来看，这个函数是被大大简化了，估计对本地来说这点儿内容就够用了呢。

5. createResponseObject方法倒是被大大强化了，你看现在多少行？其实前半段都是一样，localXHR.js中对isLocal的情况做了大量的处理，响应中的responseText看来是可以从连接中获得的，然后如果需要xml，它就使用ActiveXObject("Microsoft.XMLDOM")或者是new DOMParser()把responseText解析成xml放到response里，响应状态也是重新计算的，这样就能让ajax正常调用了。

6. 最后修理的是asyncRequest方法，异步请求的时候如果出现异常，就调用handleTransactionResponse返回响应，然后根据各种情况，稍微搞一搞header属性啦。

7. 最后212行有一个小插曲，

```
Ext.lib.Ajax.forceActiveX = (document.location.protocol == 'file:');
```

   如果协议是file:，就强制使用activex撒。吼吼。

对localXHR.js的超简单研究就此告一段落，偶认为他的主要功能还是在判断协议为file:的时候，对响应状态进行伪造，似乎不支持ie7，如果能测试一下就好了。

10.3. 在form中使用fckeditor

ext自带的轻量级htmleditor功能有些简陋了，如果想换成fckeditor可以考虑先制作一个textarea，然后把这个textarea换成fckeditor。

首先我们有一个form，里边有个textarea。

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 50,
    width: 600,
    title: 'form',
    frame: true,
    items: [{
        name: 'textarea',
        xtype: 'textarea',
        fieldLabel: 'textarea'
    }]
});
form.render("form");
```

假设fckeditor放在./fckeditor/目录下，两步实现textarea到fckeditor的转换。

1. 导入fckeditor.js。

```
<script type="text/javascript" src="./fckeditor/fckeditor.js"></script>
```

2. form.render()后才执行fckeditor，让dom生成完再继续。

```
var oFCKeditor = new FCKeditor( 'textarea' );
oFCKeditor.BasePath = "./fckeditor/";
oFCKeditor.ReplaceTextarea();
```

不过这样修改后的form，即使调用submit()也不会包含fckeditor编辑的部分。你只能用form.el.dom.submit();或者自己手工获得这些对象，使用ajax发送到后台。

10.4. 健康快乐动起来，fx里的动画效果

如果看过sources目录下的源代码话，就会在core目录下看到一个叫fx.js的东西，这个东西只专管动画效果的，只不过ext里已经将动画效果合并到Element中了，任何时候得到的el都可以直接调用fx中定义的动画效果函数的。

Fx.js中一共定义了几种动画效果，包括渐入渐出，高亮，改变大小，闪烁什么的。每个动画效果都可以在options里指定duration持续时间，remove是否在消失后彻底删除dom，callback在执行完效果后执行的回调函数，等等等等等。

在这些参数里，anchor参数是个好玩的东西呢，它表示着有可能实现动画效果的八个方向。

表 10.1. anchor方向

数值	说明
tl	左上
t	上
tr	右上
l	左
r	右
bl	左下
b	下
br	右下

具体实现的效果会根据定义的方向展开，这八个方向任选一个也足够用了。

Fx还包括一些其他方法，比如pause()可以指定暂停几秒后执行下一个动画效果，syncFx()表示立即执行所有动画效果。

千言万语也不如实际看看这些效果，例子在lingo-sample/1.1.1/10-01.html和lingo-sample/2.0/10-01.html。

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：

1. 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。

2. 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，不可能让你随便用。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，就可以在LGPL协议下免费用。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。因为它的内容实在太多了，我就简单说一下LGPL的含义。LGPL是被认为能够较好保护开发者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。

3. 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：

1. 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。

2. 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。

1. 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。

2. 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。

3. 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" />
```

4. 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.3.1. 顺便说说常见的Ext is not defined错误

请注意，如果上述的js引入顺序不是按照ext-base.js，ext-all.js，就会出现这个错误。

原因是因为Ext这个顶级命名空间是在adapter里定义的，你可以选择ext-base，prototypejs，jquery或是yui作为adapter，但这个adapter必须放在ext-all.js前面。你要是错把ext-all.js放到最前面就会出现ext未定义的错误。

多说一句，要是你把自己扩展ext的js放到这些东西的前面，也会出现xxx is not a constructor的错误。所以说顺序可是很重要滴。

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.huddletogether.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);
```

A.9. 有关grid的一些小问题

A.9.1. 如何让grid总所有的列都支持排序

当然，你可以在每个列上都写上sortable: true，然而呢，我们也可以修改ColumnModel的一个属性defaultSortable，默认情况下它是false，如果希望所有列默认都可以排序，就在构造ColumnModel之后，设置cm.defaultSortable = true，这样，这个cm下的所有列默认都是可排序的了。

A.9.2. 修改一个grid的ColumnModel和Store

cm和ds决定了一个grid的显示方式和显示数据，有的同志提出了希望用一个grid显示不同形式的数据，这些数据的列模型与数据不一定相同，这个时候我们可以使用reconfigure()函数，它的第一个参数Ext.data.Store，第二个参数Ext.grid.ColumnModel，只需要把新的ds和cm传入，就可以让grid显示新的数据了。

A.10. 如何选中树中的某个节点

1. 第一，每个TreeNode都有select()函数，调用这个函数就是选中节点。
2. 第二，使用TreePanel的selectPath()函数，传入的参数是想要选中的节点的path值，你可以使用node.getPath()获得，举个例子，如果根节点的id是root，那么就使用selectPath('/root');如果根节点下有一个id为leaf的节点，我们要选中这个节点就要使用selectPaht('/root/leaf');以此类推。
3. 第三，通过SelectionModel选择节点，treePanel.getSelectionModel()便可以获得属性的选择模型，sm有一个select()函数，传入TreeNode就可以选中这个节点了。它还提供selectPrevious(), selectNext()方法，选中当前选中节点的上一个，或下一个节点。

A.11. 如何使用input type="image"

关于input type="password"，input type="submit"，input type="file"，input type="image"在Ext中没有提供对应的控件，我们可以根据需要使用inputType对Ext.form.TextField进行改装。

这里input type="image"比较特殊，因为需要通过src属性指定显示的图片，默认情况下Ext.form.TextField生成的dom没有这个属性，我们需要修改一下autoCreate参数：

```
{
    fieldLabel: '证件照片',
    name: 'smallimg',
    inputType: 'image',
    autoCreate: {tag: "input", type: "image", src:'http://jstang.5d6d.com/images/avatars/noavatar.gif' },
    width: 120,
    height: 140
```

```
}
```

autoCreate使用的是DomHelper的语法，这样它就会生成一个带有src的dom了。

附录 B. 修改日志

修订历史

修订 0.1.2 2008-01-11

1. 补充TreeLoader读取本地json：第 3.5.2 节 “TreeLoader外传 之 读取本地json”。

2. 添加：第 6.7 节 “向诸位介绍一下2.0里各具特色的布局”。

3. 添加：第 7.4 节 “yui自远方来，不亦乐乎”。

4. 添加：第 8.10 节 “灵异事件，Ext.state”，第 8.11 节 “所谓的事件”。

5. 添加：第 10.4 节 “健康快乐动起来，fx里的动画效果”。

6. 补充了一些FAQ：第 A.9 节 “有关grid的一些小问题” ~ 第 A.11 节 “如何使用input type=″image″”。

修订 0.1.1 2008-01-07

1. 补充中文排序：第 2.4.3 节 “中文排序是个大问题”。

2. 补充重新计算行号：第 2.6.1 节 “自动行号”。

3. 补充前台排序：第 2.7.5 节 “谣言说ext不支持前台排序”。

4. 补充一系列对PropertyGrid的操作：第 2.10.1.2 节 “小舞曲：强制对name列排序”。

5. 添加：第 2.12 节 “大步流星，后台排序”。

6. 添加：第 4.11 节 “自动把数据填充到form里”。

7. 添加：第 8.3.3 节 “醍醐灌顶，MasterTemplate和XTemplate。”。

8. 添加：第 8.8 节 “蓝与灰，切换主题”。

9. 添加：第 8.9 节 “悬停提示”。

修订 0.1.0 2007-12-31

1. 添加：第 4.8 节 “关于表单内部控件的布局问题”。

2. 添加：第 4.9 节 “还要做文件上传哟”。

3. 添加：第 4.10 节 “非想非想，单选框多选框”。

4. 补充更多菜单内容：第 8.7 节 “菜单和工具条”。

修订 0.0.9 2007-12-22

1. 补充修改表格列宽。第 2.4.1 节 “自主决定每列的宽度”。

2. 补充让TreeLoader支持json对象。第 3.5 节 “这种装配树节点的形式，真是让人头大。”。

3. 补充给节点设置超链接。第 3.7.6 节 “给树节点设置超链接”。

4. 添加：第 4.6 节 “form提交数据的三重门”。

5. 添加：第 4.7 节 “验证苦旅”。

6. 添加：第 8.5 节 “跟我用json，每天五分钟”。

7. 添加：第 8.6 节 “小声说说scope”。

8. 起个头：第 8.7 节 “菜单和工具条”。

9. 添加：第 10.3 节 “在form中使用fckeditor”。

修订 0.0.8 2007-12-18

1. 补充转换date。第 2.5 节 “让单元格里显示红色的字，图片，按钮，你还能想到什么？”。

2. 补充点击获得选中信息。第 2.9 节 “连坐法，关于选择模型”。

3. 添加第 2.11 节 “午夜怪谈，论可以改变大小，可以拖拽的表格”。

4. 添加第 5.4.4 节 “把form放进对话框里”。

5. 补充TabPanel的部分。第 6.5 节 “脑袋上有几个标签的tabPanel”。

6. 添加第 6.6 节 “让布局复杂一点儿”。

7. 添加第 8.4 节 “Ext.data命名空间”。

修订 0.0.7 2007-12-13

1. 添加第 2.8 节 “爱生活，EditorGrid。”。

2. 添加第 3.9.2.3 节 “裟椤双树，常与无常”。

3. 添加第 5.4.3 节 “1.x里的叫做BasicDialog”。

4. 添加第 6.5 节 “脑袋上有几个标签的tabPanel”。

修订 0.0.6 2007-12-12

1. 添加第 3.9.2.2 节 “把节点扔到哪里啦”。

2. 添加第 4.5 节 “把form里的那些控件全部拿出来看看”

3. 添加第 10 章 撕裂吧！邪魔外道与边缘学科。

修订 0.0.5 2007-12-10

1. docbook-dtd升级到5.0b5，docbook-xsl升级到1.73.2，于是我们得到了代码高亮功能。

2. 添加第 3.6 节 “jsp的例子是一定要有的”

3. 添加第 5.4 节 “让弹出窗口，显示我们想要的东东，比如表格”

4. 添加第 A.1 节 “ext到底是收费还是免费”
修订 0.0.4 2007-12-09

1. 添加第 1.7 节 “入门之前，都看helloworld。”

2. 添加第 4.4 节 “起点高撒，从comboBox往上蹦”
修订 0.0.3 2007-12-06

1. 添加第 2.7 节 “分页了吗？分页了吗？如果还没分就看这里吧。”

2. 添加第 6.4 节 “2.0的ViewPort是完全不同的实现”
修订 0.0.2 2007-12-04

1. ext-2.0正式发布，2.0-rc1相关的代码升级到 2.0。
修订 0.0.1 2007-11-28

1. 初稿完成。说在前头的

附录 C. 后记

C.1. 2007年12月5日，迷茫阶段

写这东西是想解决眼前的问题，说实在的烦死了。

C.1.1. 仇恨

超级仇恨那些有问题就问的新手，他简直就认为这世界上只有自己是在做程序员，而其他人的职业都是培训师和咨询师，而且这些培训师和咨询师竞争超激烈，必须讨好程序员才能活下去。

于是他可以不用整理思路就胡乱提问，让别人帮他整理思路。他脑子里都是糨糊吗？

于是他可以不用整理语言就说话，他说的是汉语吗？天晓得当初他是怎么学语文的。

于是他可以骂对方态度不好，因为整个世界都是绕着他转的。他是主人，别人都是仆人。

于是，他想什么时候问问题，就什么时候问，对方回答的方式必须照他喜好来，因为他的要求就必须满足。

于是，你给他解决问题，必须是免费的，必须不能谈任何条件。你要是不这么做，就要遭天谴似的。

C.1.2. 反省

自己有一些强迫症，正常生活没什么朋友，每天有事没事都挂在网上，天天聊天，有时候也帮别人解决问题，当然偶的实力很差啦。 沉迷于虚拟社区是现在要面对的最大难题。

在下性格暴躁，对社会有错误的认识，以为别人都会为对方着想，又不知道怎么好好与人相处，其实遇到自己不喜欢的人，除了吵架就是不理他，后者当然更有利于社会和谐，我在逐渐学习这一点了。

实际上并非所有人都投缘，谨记要同不喜欢的家伙保持距离。现实中也要意识到这一点。

昨天又吵了一架，对方认为我回答问题的态度不好，我脾气上来就控制不住的一通大骂，嘿嘿，网上人们都不为自己的言行负责呢。这让我明白了一件事，免费给的东西没人珍惜。介于此，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。

C.2. 关于ext与dwr整合部分的讨论

啪啦啪啦同志提出，希望文档加入ext与dwr整合的部分。

ext是与后台无关的，而dwr是java平台的产物，讨论两者整合是否会与平台绑定，失去可移植性？

dwr在前台的调用与普通js无异，谈不上整合的问题，完全可以先调用dwr返回需要的数据，然后交给ext处理。

考虑dwr在java平台的占有率巨大，而且应用上比json更简便，还是有涉足的价值的。

目前接触到的ext与dwr整合部分的扩展，仅有DwrProxy和DwrTreeProxy，分别都要求后台有对应的

javabean辅助，计划先涉及这两个扩展。

C.3. 怎么看文档附件里的范例

购买了本文档的同志，在邮件附件中除了打包好的电子书之外，还有文档中所有的例子，例子分为1.x和2.0版本，分别放在lingo-sample目录下的1.1.1目录和2.0目录下，请根据自己使用的版本进行选择。

使用的时候，请将对应目录放在ext发布包的examples目录下，既可以用浏览器打开示例html文件观看效果。如下图所示：

示例中使用了localXHR.js，无需服务器就可以读取json数据，直接放在本地就可以浏览大部分例子。

对于需要后台jsp提供数据的例子，最简单的方法是，将整个ext发布包复制到tomcat的webapps目录下，启动tomcat后，通过浏览器访问examples下的例子。

附录 D. 贡献者列表

D.1. 感谢[飘]的大力支持

感谢[飘]的大力支持，推进了ext-2.0版TreeField与CheckboxTree的开发进程。

祝福飘同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.2. 感谢[吧啦吧啦]的大力支持

感谢[吧啦吧啦]的大力支持，推进了文档转换为pdf的进程，对dwr与ext整合的建议。

祝福吧啦吧啦同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.3. 感谢[游戏人生]的大力支持

感谢[游戏人生]的大力支持，推进了文档服务的改进和ext主题的开发进程，推进了树形章节的编写。

祝福游戏人生同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.4. 感谢[綄帥]的大力支持

感谢[綄帥]的大力支持，推进了文档的开发。

祝福綄帥同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.5. 感谢[葡萄]的大力支持

感谢[葡萄]的大力支持，推进了文档的开发。

祝福葡萄同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美

。

D.6. 感谢[天外小人]的大力支持

感谢[天外小人]的大力支持，推进了文档的开发，对localXHR方面研究的建议。

祝福天外小人同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.7. 感谢[我想我是海]的大力支持

感谢[我想我是海]的大力支持，推进了文档的开发。

祝福我想我是海同志：

一帆风顺 两全其美 三羊开泰 四通八达 五福同寿 六六大顺 七星高照 八面玲珑 九九归一 十全十美。

D.8. 还要感谢：

1. 我爱小宝宝

2. Bourne

3. \(- 瓜 -)/

4. 恋

5. 真神

6. 杨涛

7. Wayne

8. 刘涛

9. Bear&か銘

10.Martin

11./love尛晞凌

12.亚马逊蝌蚪

13.jasonm

14.不迷不悟

15.笨笨熊

16.小鸣哥