

# Extjs4 学习指南

( 仅供学习使用、转载需注明出处 )

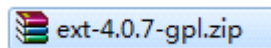
Extjs4 可用的学习资料少之又少，本文内容大部分为网络整理、  
方便学习者使用，如有出入以 extjs4 api 文档为准。

# 1 Extjs 初步

## 1.1 获取 Extjs

### 下载 extjs :

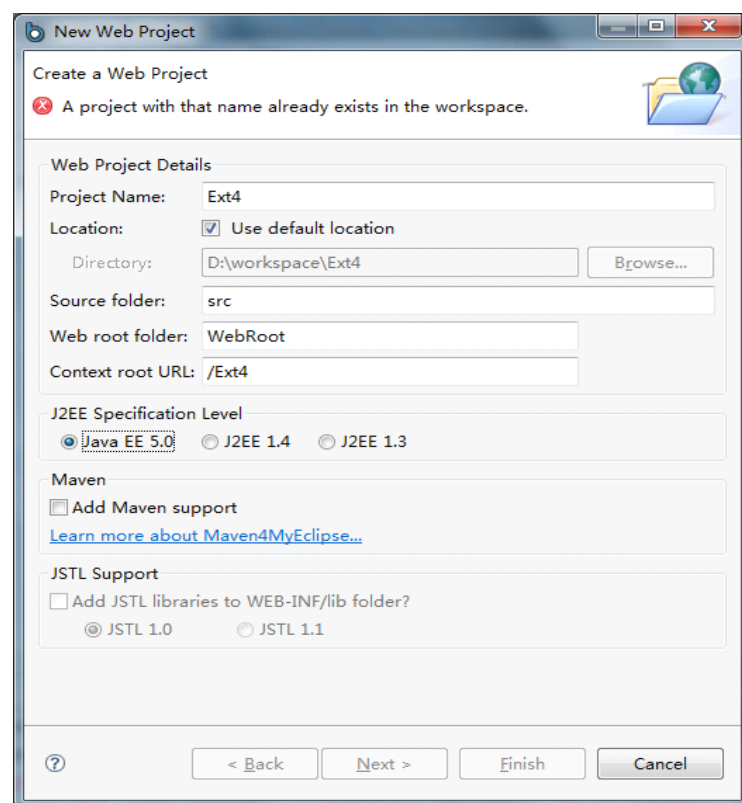
可以从 <http://extjs.org.cn/> 获得需要的 extjs 发布包及更多支持。



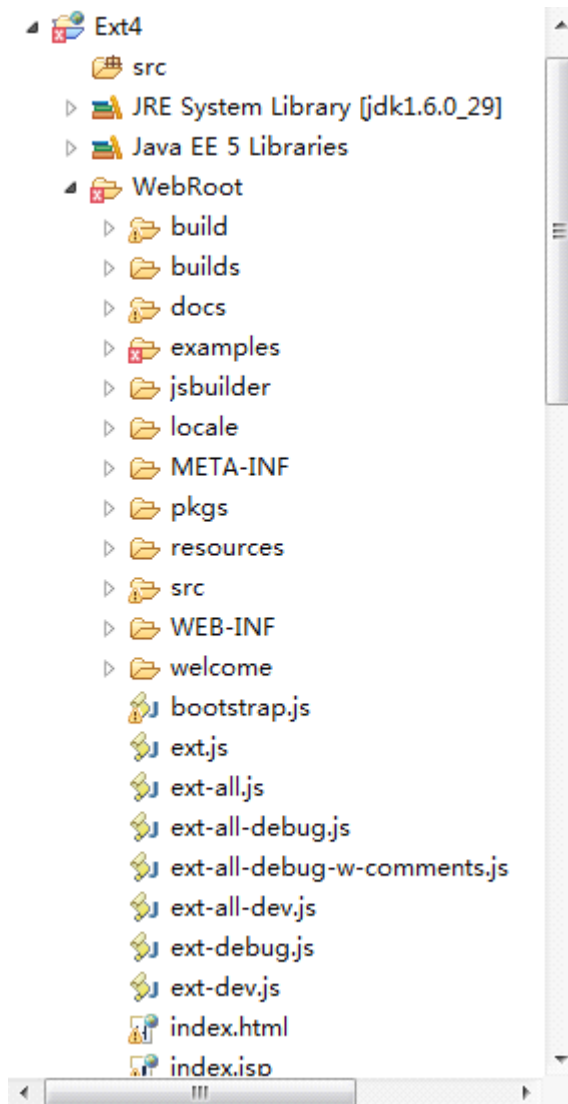
## 1.2 搭建学习环境:

假设您的机器已经安装 myeclipse 和 tomcat , 如果没有请参阅其他相关资料。

myeclipse 建立新 Web project 项目 Extjs4



并且将 extjs4.0.7 压缩包解压后的全部文件复制到项目的 Webroot 目录下



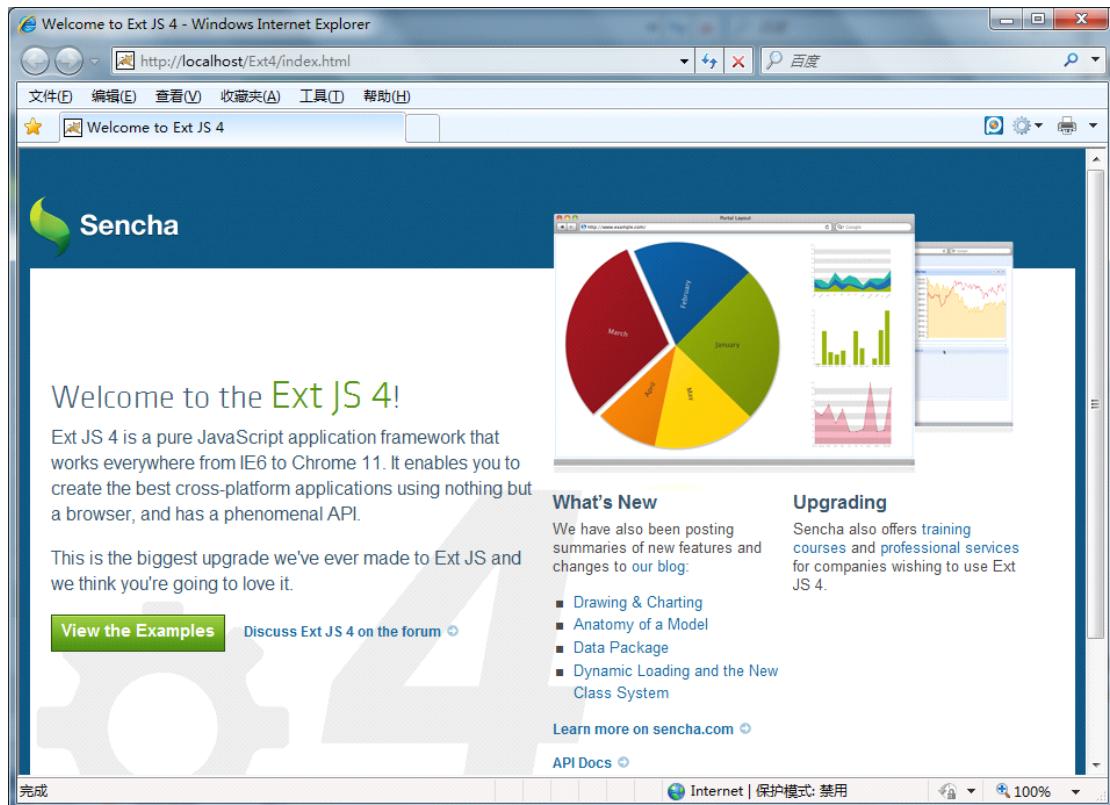
Examples 目录为 ext 官方提供的例子程序，其中可能包涵 php 的代码，错误信息可以暂时不理睬。

部署并且启动 tomcat，测试环境是否可用。

打开浏览器，输入 <http://localhost:8080/Ext4/index.html>

假设您的 tomcat 端口为 8080

您会看到以下界面，证明环境已经搭建成功！



查看 api 文档 <http://localhost:8080/Ext4/docs/index.html>

查看示例页面 <http://localhost:8080/Ext4/examples/index.html>

## 1.3 测试例子

### 开始...

Webroot 目录下建立 helloworld.js 输入如下内容:

```
Ext.application({
    name: 'HelloExt',
    launch: function() {
        Ext.create('Ext.container.Viewport', {
            layout: 'fit',
            items: [
                {
                    title: 'Hello Ext',
                    html: 'Hello! Welcome to Ext JS.'
                }
            ]
        });
    }
});
```

再建立一个 helloworld.html, 输入如下内容

```
<html>
<head>
  <title>Hello Ext</title>
  <link rel="stylesheet" type="text/css" href="resources/css/ext-all.css">
  <script type="text/javascript" src="ext-all.js"></script>
  <script type="text/javascript" src="HelloExt.js"></script>
</head>
<body></body>
</html>
```

Html 文件中只引入了一个 css 和 2 个 js 文件，注意引用路径和你建立文件路径是否能匹配，如果路径没有问题的话，打开浏览器输入

<http://localhost:8080/Ext4/helloworld.html> 您将会看到浏览器里显示一个 panel，标题是 Hello Ext，内容 Hello! Welcome to Ext JS.，如果没有，请查看是否有路径不匹配。

其他：

在 ExtJS 里最常用的，应该就是 Ext.onReady 和 Ext.application 这两个方法了，而且它也可能是你学习 ExtJS 所接触的第一个方法，这个方法在当前的 DOM 加载完毕后自动调用，保证页面内的所有元素都能被 Script 所引用。可以尝试在这个方法中添加一条语句，看看页面打开后是什么反映

（先建立 js 和 html 文件，将如下代码加入 js 文件中，html 文件相应引入对应的 js 文件，本文档所有示例代码均如此方式运行以下不再重复）

```
Ext.onReady(function() {
  alert('hello world!');
});
```

上面的代码将在页面加载完毕后弹出一对话框，打印出 'hello world!' 字样。

## 获取元素

还有一个常用的方法，就是获取页面上的元素了，ExtJS 提供了一个 get 方法，可以根据 ID 取到页面上的元素：

```
var myDiv = Ext.get('myDiv');
```

会取到页面上 ID 为 'myDiv' 的元素。如果使用 Element.dom 的方法，则可以直接操作底层的 DOM 节点，Ext.get 返回的则是一个 Element 对象。

在不能使用这种方式来获取多个 DOM 的节点，或是要获取一些 ID 不一致，但又有相同特征的时候，可以通过选择器来进行获取，比如要获取页面上所有的

标签，则可以使用：

```
var ps = Ext.select('p');
```

这样你就可以对所要获取的元素进行操作了,select()方法返回的是 Ext.CompositeElement 对象,可以通过其中的 each()方法对其所包含的节点进行遍历:

```
ps.each(function(el) {  
    el.highlight();  
});
```

当然,如果你要是获取的所有元素进行相同的操作,可以直接应用于 CompositeElement 对象上,如:

```
ps.highlight();
```

或是:

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

当然,select 参数还可以更复杂一些,其中可以包括 W3C CSS3Dom 选取器,基本的 XPath,HTML 属性等,详细情况,可以查看 DomQuery API 的文档,来了解细节.

## 事件响应

获取到了元素,则可能会对一些元素的事件进行一些处理,比如获取一个按钮,我们为它添加一个单击事件的响应:

复制代码 代码如下:

```
Ext.onReady(function() {  
    Ext.get('myButton').on('click', function() {  
        alert('You clicked the button!');  
    });  
});
```

当然,你可以把事件的响应加到通过 select()方法获取到的元素上:

复制代码 代码如下:

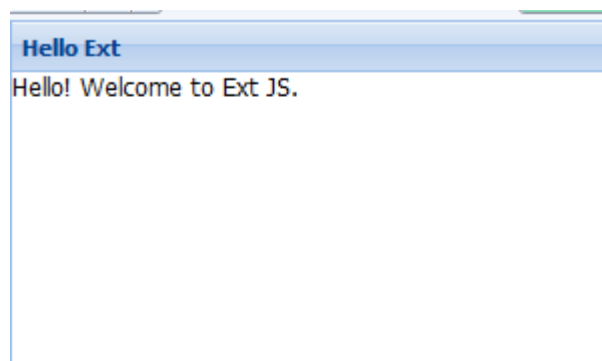
```
Ext.select('p').on('click', function() {  
    alert('You clicked a paragraph!');  
});
```

## Widgets

ExtJS 还提供了丰富的 UI 库来供大家使用。

# 2 Extjs4 布局详解

## 2.1 Fit 布局



在 **Fit** 布局中，子元素将自动填满整个父容器。注意：在 **fit** 布局下，对其子元素设置宽度是无效的。如果在 **fit** 布局中放置了多个组件，则只会显示第一个子元素

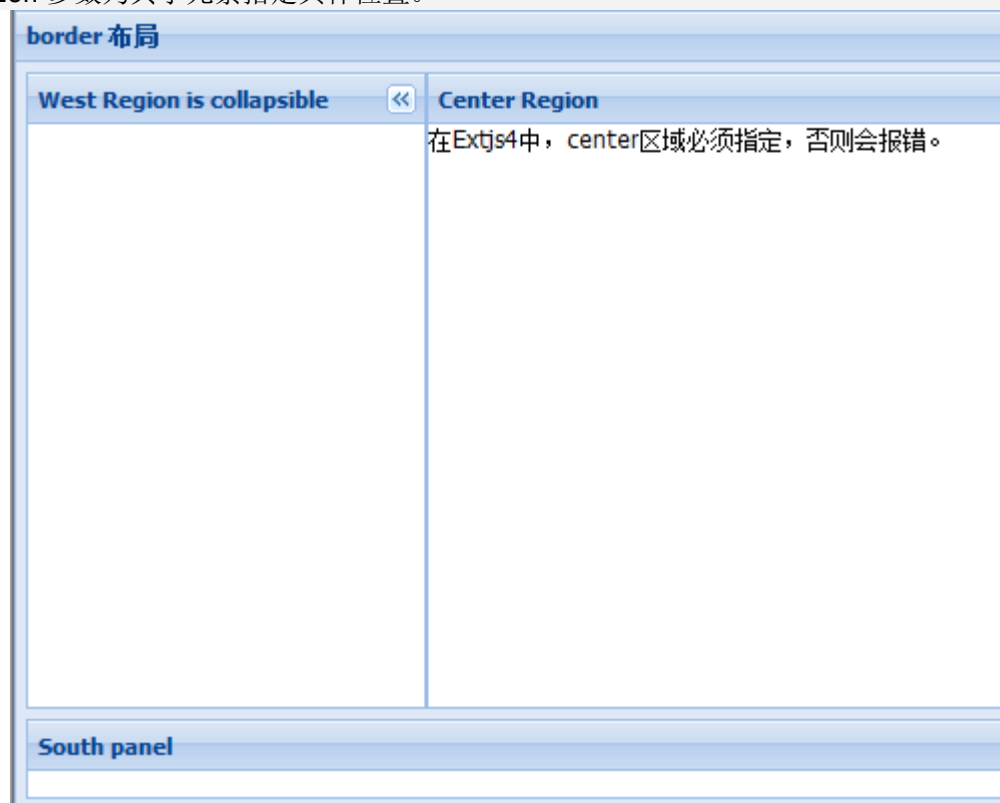
在 **Fit** 布局中，子元素将自动填满整个父容器。注意：在 **fit** 布局下，对其子元素设置宽度是无效的。如果在 **fit** 布局中放置了多个组件，则只会显示第一个子元素。典型的案例就是当客户要求一个 **window** 或 **panel** 中放置一个 **GRID** 组件，**grid** 组件的大小会随着父容器的大小改变而改变。

示例代码：

```
Ext.application({  
    name: 'HelloExt',  
    launch: function() {  
        Ext.create('Ext.container.Viewport', {  
            layout: 'fit',  
            items: [  
                {  
                    title: 'Hello Ext',  
                    html: 'Hello! Welcome to Ext JS.'  
                }  
            ]  
        });  
    }  
});
```

## 2.2 Border 布局

border 布局: border 布局也称边界布局, 他将页面分隔为 west, east, south, north, center 这五个部分, 我们需要在其 items 中指定使用 region 参数为其子元素指定具体位置。



border 布局: border 布局也称边界布局, 他将页面分隔为 west, east, south, north, center 这五个部分, 我们需要在其 items 中指定使用 region 参数为其子元素指定具体位置。

注意: north 和 south 部分只能设置高度 (height), west 和 east 部分只能设置宽度 (width)。north south west east 区域变大, center 区域就变小了。

参数 split:true 可以调整除了 center 四个区域的大小。

参数 collapsible:true 将激活折叠功能, title 必须设置, 因为折叠按钮是出现标题部分的。

**center** 区域是必须使用的, 而且 **center** 区域不允许折叠。**Center** 区域会自动填充其他区域的剩余空间。尤其在 **Extjs4.0** 中, 当指定布局为 **border** 时, 没有指定 **center** 区域时, 会出现报错信息。

示例代码:

```
Ext.application({
    name: "HelloExt",
    launch: function () {
        Ext.create('Ext.panel.Panel', {
            width: 1024,
            height: 720,
            layout: 'border',
            items: [{
                region: 'south',
                xtype: 'panel',
                height: 20,
```



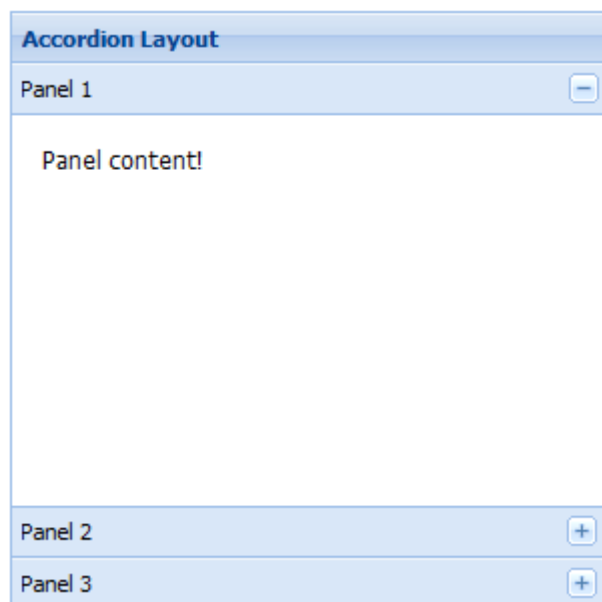
```

        split: false,
        html: '欢迎登录!',
        margins: '0 5 5 5'
    }, {
        title: 'West Region is collapsible',
        region: 'west',
        xtype: 'panel',
        margins: '5 0 0 5',
        width: 200,
        collapsible: true,
        id: 'west-region-container',
        layout: 'fit'
    }, {
        title: 'Center Region',
        region: 'center',
        xtype: 'panel',
        layout: 'fit',
        margins: '5 5 0 0',
        html: '在Extjs4中，center区域必须指定，否则会报错。'
    }
    ],
    renderTo: Ext.getBody()
});
}
});

```

## 2.3 Accordion 布局

**accordion 布局：**accordion 布局也称手风琴布局，在 accordion 布局下，在任何时间里，只有一个面板处于激活状态。其中每个面边都支持展开和折叠。



**accordion 布局：**accordion 布局也称手风琴布局，在 accordion 布局下，在任何时间里，只有一个面板处于激活状态。其中每个面边都支持展开和折叠。注意：只有 Ext.Panels 和所有 Ext.panel.Panel 子项，才可以使用 accordion 布局。

示例代码：

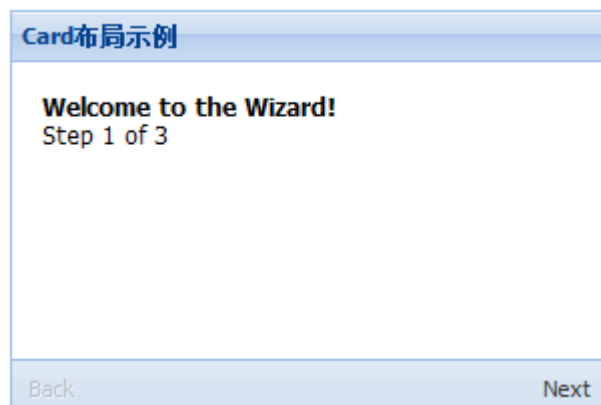
```


```

```
Ext.application({
    name: "HelloExt",
    launch: function () {
        Ext.create('Ext.panel.Panel', {
            title: 'Accordion Layout',
            width: 300,
            height: 300,
            x: 20,
            y: 20,
            layout: 'accordion',
            defaults: {
                bodyStyle: 'padding:15px'
            },
            layoutConfig: {
                titleCollapse: false,
                animate: true,
                activeOnTop: true
            },
            items: [{
                title: 'Panel 1',
                html: 'Panel content!'
            }, {
                title: 'Panel 2',
                html: 'Panel content!'
            }, {
                title: 'Panel 3',
                html: 'Panel content!'
            }
        ],
            renderTo: Ext.getBody()
        });
    });
});
```

## 2.4 Card 布局

**Card 布局：**这种布局用来管理多个子组件，并且在任何时刻只能显示一个子组件。这种布局最常用的情况是向导模式，也就是我们所说的分布提交。



**Card 布局：**这种布局用来管理多个子组件，并且在任何时刻只能显示一个子组件。这种布局最常用的情况是向导模式，也就是我们所说的分布提交。**Card** 布局可以使用 `layout: 'card'` 来创建。注意：由于此布局本身不提供分步导航功能，所以需要用户自己开发该功能。由于只有一个面板处于显示状态，那么在初始时，我们可以使用

`setActiveItem` 功能来指定某一个面板的显示。当要显示下一个面板或者上一个面板的时候，我们可以使用 `getNext()` 或 `getPrev()` 来得到下一个或上一个面板。然后使用 `setDisabled` 方法来设置面板的显示。另外，如果面板中显示的是 **FORM** 布局，我们在点击下一个面板的时候，处理 **FORM** 中提交的元素，通过 **AJAX** 将表单中的内容保存到数据库中或者 **SESSION** 中。

下面的示例代码展示了一个基本的 **Card** 布局，布局中并没有包含 **form** 元素，具体情况，还要根据实际情况进行处理：

```
Ext.application({
    name: 'HelloExt',
    launch: function() {
        var navigate = function(panel, direction){
            var layout = panel.getLayout();
            layout[direction]();
            Ext.getCmp('move-prev').setDisabled(!layout.getPrev());
            Ext.getCmp('move-next').setDisabled(!layout.getNext());
        };
        Ext.create('Ext.panel.Panel', {
            title: 'Card布局示例',
            width: 300,
            height: 202,
            layout: 'card',
            activeItem: 0,
            x:30,
            y:60,
            bodyStyle: 'padding:15px',
            defaults: {border: false},
            bbar: [{
                id: 'move-prev',
                text: 'Back',
                handler: function(btn) {
                    navigate(btn.up("panel"), "prev");
                },
                disabled: true
            },
            '->',
            {
                id: 'move-next',
                text: 'Next',
                handler: function(btn) {
                    navigate(btn.up("panel"), "next");
                }
            }
        ]],
        items: [{
            id: 'card-0',
            html: '<h1>Welcome to the Wizard!</h1><p>Step 1 of 3</p>'
        },
        {
            id: 'card-1',
            html: '<p>Step 2 of 3</p>'
        },
        {
            id: 'card-2',
            html: '<h1>Congratulations!</h1><p>Step 3 of 3 - Complete</p>'
        }
    ],
    renderTo: Ext.getBody()
});
});
```

## 2.5 Anchor 布局

**anchor** 布局将使组件固定于父容器的某一个位置，使用 **anchor** 布局的子组件尺寸相对于容器的尺寸，即父容器容器的大小发生变化时，使用 **anchor** 布局的组件会根据规定的规则重新渲染位置和大小。



**anchor** 布局将使组件固定于父容器的某一个位置，使用 **anchor** 布局的子组件尺寸相对于容器的尺寸，即父容器容器的大小发生变化时，使用 **anchor** 布局的组件会根据规定的规则重新渲染位置和大小。

**AnchorLayout** 布局没有任何的直接配置选项（继承的除外），然而在使用 **AnchorLayout** 布局时，其子组件都有一个 **anchor** 属性，用来配置此子组件在父容器中所处的位置。

**anchor** 属性为一组字符串，可以使用百分比或者是-数字来表示。配置字符串使用空格隔开，例如

**anchor:** '75% 25%', 表示宽度为父容器的 75%，高度为父容器的 25%

**anchor:** '-300 -200', 表示组件相对于父容器右边距为 300，相对于父容器的底部位 200

**anchor:** '-250 20%', 混合模式，表示组件党对于如容器右边为 250，高度为父容器的 20%

示例代码：

```
Ext.application({
    name: 'HelloExt',
    launch: function() {
        Ext.create('Ext.Panel', {
            width: 500,
            height: 400,
            title: "Anchor布局",
            layout: 'anchor',
```

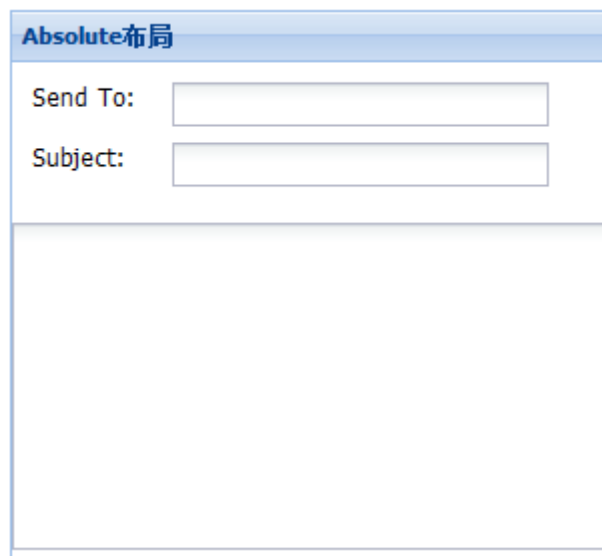
```

        x:60,
        y:80,
        renderTo: Ext.getBody(),
        items: [{
            xtype: 'panel',
            title: '75% Width and 25% Height',
            anchor: '75% 25%'
        }, {
            xtype: 'panel',
            title: 'Offset -300 Width & -200 Height',
            anchor: '-300 -200'
        }, {
            xtype: 'panel',
            title: 'Mixed Offset and Percent',
            anchor: '-250 30%'
        }
    ]
});
});

```

## 2.6 Absolute 布局

**Absolute** 布局继承 `Ext.layout.container.Anchor` 布局方式，并增加了 X/Y 配置选项对子组件进行定位，**Absolute** 布局的目的是为了扩展布局的属性，使得布局更容易使用



**Absolute** 布局继承 `Ext.layout.container.Anchor` 布局方式，并增加了 X/Y 配置选项对子组件进行定位，**Absolute** 布局的目的是为了扩展布局的属性，使得布局更容易使用。

```

Ext.application({
    name: "HelloExt",
    launch: function () {
        Ext.create('Ext.form.Panel', {
            title: 'Absolute布局',
            width: 300,
            height: 275,
            x:20,

```

```

y:90,
layout:'absolute',
defaultType: 'textfield',
items: [{
    x: 10,
    y: 10,
    xtype:'label',
    text: 'Send To:'
},{
    x: 80,
    y: 10,
    name: 'to',
    anchor: '90%'
},{
    x: 10,
    y: 40,
    xtype:'label',
    text: 'Subject:'
},{
    x: 80,
    y: 40,
    name: 'subject',
    anchor: '90%'
},{
    x:0,
    y: 80,
    xtype: 'textareafield',
    name: 'msg',
    anchor: '100% 100%'
}],
renderTo: Ext.getBody()
});
});

```

## 2.7 Column 布局

Column 布局一般被称为列布局，这种布局的目的是为了创建一个多列的格式。其中每列的宽度，可以为其指定一个百分比或者是一个固定的宽度。

Column Layout - 按比例		
Column 1	Column 2	Column 3

Column 布局一般被称为列布局，这种布局的目的是为了创建一个多列的格式。其中每列的宽度，可以为其指定一个百分比或者是一个固定的宽度。

Column 布局没有直接的配置选项（继承的除外），但 Column 布局支持一个 `columnWidth` 属性，在布局过程中，使用 `columnWidth` 指定每个面板的宽度。

注意：使用 Column 布局布局时，其子面板的所有 `columnWidth` 值加起来必须介于 0~1 之间或者是所占百分比。他们的总和应该是 1。

另外，如果任何子面板没有指定 `columnWidth` 值，那么它将占满剩余的空间。

示例代码：

```
Ext.application({
    name: "HelloExt",
    launch: function () {
        Ext.create('Ext.panel.Panel', {
            title: 'Column Layout - 按比例',
            width: 350,
            height: 250,
            x: 20,
            y: 100,
            layout: 'column',
            items: [{
                title: 'Column 1',
                columnWidth: .25
            }, {
                title: 'Column 2',
                columnWidth: .55
            }, {
                title: 'Column 3',
                columnWidth: .20
            }],
            renderTo: Ext.getBody()
        });
    }
});
```

## 3 Extjs4 文档阅读

ExtJS4 使用新的类机制进行了大量的重构。为了支撑新的架构,ext4 几乎重写了每一个类,因此最好先好好的理解一下新的架构,再开始编码。

本文适合想在 extjs4 中扩展现有类或者创建新类的开发者。其实，不管是想扩展还是使用，都建议您仔细阅读一下（如果 E 文好的，建议您还是阅读英文原文。链接地址是：<http://docs.sencha.com/ext-js/4-0/#/guide/>）。文章共分 4 个部分，建议每一部分都仔细研究下，对之后的开发工作，会有意想不到的好处。

## 3.1 系统类(class system)

Api 文档路径:

[http://localhost/Ext4/docs/index.html#!/guide/class\\_system](http://localhost/Ext4/docs/index.html#!/guide/class_system)

第一部分: 概述。说明了强大的类机制的必要性

第二部分: 编码规范。讨论类、方法、属性、变量和文件命名

第三部分: DIY。详细的编码示例

第四部分: 错误处理和调试。提供一些有用的调试和异常处理技巧

### 3.1.1 概述

ExtJS4 拥有超过 300 个的类。迄今为止,我们的社区拥有超过 20 万来自世界各地,使用不同后台语言的开发者。要在这种规模的框架上提供具有以下特点的架构,需要面临巨大的挑战:

- 1、简单易学。
- 2、快速开发、调试简单、部署容易。
- 3、良好的结构、可扩展性和可维护性。

### 3.1.2 编码和规范

**\*在所有类、命名空间(namespace)和文件名中使用一致的命名约定,有助于保持代码的良好结构和可读性。**

#### 1) Classes

类名只能包含**字母和数字**。允许包含数字,但是大部分情况下不建议使用,除非这些数字是专业术语的一部分。不要使用下划线,连字符等非数字字母符号。例如:

```
MyCompany.useful_util.Debug_Toolbar is discouraged
```

```
MyCompany.util.Base64 is acceptable
```

类名应该包含在使用点号分隔的命名空间中。至少,要有一个顶级命名空间。例如:

```
MyCompany.data.CoolProxyMyCompany.Application
```

顶级命名空间和实际的类名应使用驼峰命名(CamelCased),其他则为小写。例如:

```
MyCompany.form.action.AutoLoad
```

不是 Sencha 开发的类(即不是 Ext 自带的)不要使用 Ext 做为顶级命名空间。缩写也要遵守以上的驼峰式命名约定。例如:

```
Ext.data.JsonProxy 代替 Ext.data.JSONProxy
```

```
MyCompany.util.HtmlParser 代替 MyCompany.parser.HTMLParser
```

```
MyCompany.server.Http 代替 MyCompany.server.HTTP
```

2) 代码文件类名对应类所在的文件(包括文件名)。因此,每个文件应该只包含一个类(类名和文件名一样)。例如:



`Ext.util.Observable` 存放在 `path/to/src/Ext/util/Observable.js`  
`Ext.form.action.Submit` 存放在 `path/to/src/Ext/form/action/Submit.js`  
`MyCompany.chart.axis.Numeric` 存放  
在 `path/to/src/MyCompany/chart/axis/Numeric.js`  
`path/to/src` 是你的应用所在目录。所有类都应该在这个通用根目录下，并且使用适当的命名空间以利于开发、维护和部署。

### 3) 方法和变量

- 和类命名一样，方法和变量也只能包含字母和数字。数字同样是允许但不建议，除非属于专业术语。不要使用下划线，连字符等任何非字母数字符号。

- 方法和变量名一样使用驼峰式命名，缩写也一样。

- 举例

- 合适的方法名：

```
encodeUsingMd5() getHtml() 代替 getHTML()  
getJsonResponse() 代替 getJSONResponse()  
parseXmlContent() 代替 parseXMLContent()
```

- 合适的变量名：

```
var isGoodName  
var base64Encoder  
var xmlReader  
var httpServer
```

### 4) 属性

- 类属性名称遵循以上的变量和方法命名约定，除非是静态的常量。

- 类的静态属性常量应该全部大写。例如：

- `Ext.MessageBox.YES = "Yes"`
- `Ext.MessageBox.NO = "No"`
- `MyCompany.alien.Math.PI = "4.13"`

## 3.1.3 DIY 亲自动手（示例代码）

---

### 3.1.3.1 声明

#### 3.1.3.1.1 Extjs4 之前的方式

如果你曾经使用过旧版本的 `extjs`，那么你一定熟悉使用 `Ext.extend` 来创建一个类：

```
1: var MyWindow=Ext.extend(Object,{...});
```

这个方法很容易从现有的类中继承创建新的类。相比直接继承，我们没有好用的 API 用于类创建的其他方面，诸如：配置、静态方法、混入（Mixins）。呆会我们再来详细的重新审视这些方面。现在，让我们来看看另一个例子：

```
1: My.cool.Window = Ext.extend(Ext.Window, { ... });
```

在这个例子中, 我们创建我们的新类, 继承 `Ext.Window`, 放在命名空间中。我们有两个问题要解决:

1, 在我们访问 `My.cool` 的 `Window` 属性之前, `My.cool` 必须是一个已有的对象。

2, `Ext.Window` 必须在引用之前加载。

第一个问题通常使用 `Ext.namespace` (别名 `Ext.ns`) 来解决。该方法递归创建 (如果该对象不存在) 这些对象依赖。比较繁琐枯燥的部分是你必须在 `Ext.extend` 之前执行 `Ext.ns` 来创建它们。

```
1: Ext.ns('My.cool');
```

```
2: My.cool.Window = Ext.extend(Ext.Window, { ... });
```

第二个问题不好解决, 因为 `Ext.Window` 可能直接或间接的依赖于许多其他的类, 依赖的类可能还依赖其它类... 出于这个原因, 在 `ext4` 之前, 我们通常引入整个 `ext-all.js`, 即使是我们只需要其中的一小部分。

### 3.1.3.1.2 Extjs4 新的方式

在 `Extjs4` 中, 你只需要使用一个方法就可以解决这些问题: **`Ext.define`**。以下是它的基本语法:

```
1: Ext.define(className, members, onClassCreated);
```

`className`: 类名

`members`: 代表类成员的对象字面量 (键值对, json)

`onClassCreated`: 可选的回调函数, 在所有依赖都加载完毕, 并且类本身建立后触发。由于类创建的新的异步特性, 这个回调函数在很多情况下都很有用。这些在第四节中将进一步讨论

例如:

```
Ext.define('My.sample.Person', {
    name: 'Unknown',
    constructor: function(name) {
        if (name) {
            this.name = name;
        }
        return this;
    },
    eat: function(foodType) {
        alert(this.name + " is eating: " + foodType);

        return this;
    }
});
var aaron = Ext.create('My.sample.Person', 'Aaron');
aaron.eat("Salad");
```

程序执行结果会弹出 `alert` 窗口显示 `"Aaron is eating: Salad"`。

- 注意我们使用 `Ext.create()` 方法创建了 `My.sample.Person` 类的一个新实例.我们也可以使用新的关键字(`new My.sample.Person()`)来创建.然而,建议养成始终用 `Ext.create` 来创建类示例的习惯,因为它允许你利用动态加载的优势.更多关于动态加载信息,请看入门指南:[入门指南](#)

### 3.1.3.2 配置

在 `ExtJS 4` ,我们引入了一个专门的配置属性,用于提供在类创建前的预处理功能.特性包括:

- 配置完全封装其他类成员
- **getter** 和 **setter**.如果类没有定义这些方法,在创建类时将自动生成配置的属性的 **getter** 和 **setter** 方法。
- 同样的,每个配置的属性自动生成一个 **apply** 方法.自动生成的 **setter** 方法内部在设置值之前调用 **apply** 方法.如果你要在设置值之前自定义自己的逻辑,那就重载 **apply** 方法.如果 **apply** 没有返回值,则 **setter** 不会设置值.看下面 `applyTitle` 的例子:

```
Ext.define('My.own.Window', {
    /** @readonly */
    isWindow: true,
    config: {
        title: 'Title Here',
        bottomBar: {
            enabled: true,
            height: 50,
            resizable: false
        }
    },
    constructor: function(config) {
        this.initConfig(config);
        return this;
    },

    applyTitle: function(title) {
        if (!Ext.isString(title) || title.length === 0) {
            alert('Error: Title must be a valid non-empty string');
        }
        else {
            return title;
        }
    },

    applyBottomBar: function(bottomBar) {
        if (bottomBar && bottomBar.enabled) {
            if (!this.bottomBar) {
                return Ext.create('My.own.WindowBottomBar', bottomBar);
            }
            else {
                this.bottomBar.setConfig(bottomBar);
            }
        }
    }
});
```

以下是它的用法:

```

var myWindow = Ext.create('My.own.Window', {
    title: 'Hello World',
    bottomBar: {
        height: 60
    }
});

alert(myWindow.getTitle()); // alerts "Hello World"

myWindow.setTitle('Something New');

alert(myWindow.getTitle()); // alerts "Something New"

myWindow.setTitle(null); // alerts "Error: Title must be a valid non-empty string"

myWindow.setBottomBar({ height: 100 }); // Bottom bar's height is changed to 100

```

### 3.1.3.3 Statics

静态成员可以使用 **statics** 配置项来定义

```

Ext.define('Computer', {
    statics: {
        instanceCount: 0,
        factory: function(brand) {
            // 'this' in static methods refer to the class itself
            return new this({brand: brand});
        }
    },
    config: {
        brand: null
    },
    constructor: function(config) {
        this.initConfig(config);

        // the 'self' property of an instance refers to its class
        this.self.instanceCount++;

        return this;
    }
});

var dellComputer = Computer.factory('Dell');
var appleComputer = Computer.factory('Mac');

alert(appleComputer.getBrand()); // using the auto-generated getter to get the
value of a config property. Alerts "Mac"

alert(Computer.instanceCount); // Alerts "2"

```

### 3.1.3.4 错误处理&调试

Extjs 4 包含一些有用的特性用于调试和错误处理。你可以使用 **Ext.getDisplayName()** 来显示任意方法的名字。这对显示抛出异常的类和方法非常有用。

```
throw new Error(['+ Ext.getDisplayName(arguments.callee) +'] 'Some message here');
```

当使用 `Ext.define()` 定义的类中的方法抛出异常后,你将在调用堆栈中看到类和方法名(如果你使用 `webkit`).例如,以下是 `chrome` 浏览器的效果:

```
✖ ▼ Uncaught Error
Ext.define.doSomething
Ext.application.launch
Ext.define.onBeforeLaunch
(anonymous function)
isEvent
fire
Ext.EventManager.onDocumentReady
Loader.Ext.Loader.onReady.fn
Loader.Ext.Loader.onReady
Ext.onReady
Ext.define.constructor
Ext.Class.Class.newClass
anonymous
Manager.Ext.ClassManager.instantiate
(anonymous function)
(anonymous function)
isEvent
fire
Ext.EventManager.onDocumentReady
Loader.Ext.Loader.onReady.fn
Loader.Ext.Loader.triggerReady
(anonymous function)
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.refreshQueue
Loader.Ext.Loader.onFileLoaded
(anonymous function)
onLoadFn
Panel.js:9
Panel.js:9
app.js:15
Application.js:185
Application.js:154
ext-debug.js:16171
ext-debug.js:16320
ext-debug.js:16478
ext-debug.js:7501
ext-debug.js:7506
ext-debug.js:17171
Application.js:148
ext-debug.js:5208
:3
ext-debug.js:6228
ext-debug.js:2159
ext-debug.js:9149
ext-debug.js:16171
ext-debug.js:16320
ext-debug.js:16478
ext-debug.js:7501
ext-debug.js:7476
ext-debug.js:2145
ext-debug.js:7085
ext-debug.js:7086
ext-debug.js:7086
ext-debug.js:7086
ext-debug.js:7086
ext-debug.js:7086
ext-debug.js:7372
ext-debug.js:2145
ext-debug.js:7104
```

`javascript` 是一种类无关 (原文: `classless`)、基于原型的语言。因此 `javascript` 最强大的特点是灵活。同样的事情可以使用不同的方式,不同的编码风格和技巧去完成。这个特点也会带来一些不可预测的风险。如果没有统一的编码规范,`javascript` 代码将很难理解、维护和复用。

相反的,基于类的编程语言拥有较为流行的面向对象模型,强类型、内建的封装机制和强制的编码约束等特点。通过强制开发人员遵守一些大的原则来使代码的行为更容易被理解,以及提高可扩展性(这里不明白,`javascript` 这类动态语言不是更容易扩展么?)和可伸缩性。但是,这类语言没有 `javascript` 的灵活性

## 3.2 MVC 应用模式

一直想写一些 `Extjs4` MVC 的东西,但由于自己的英文水平足够媲美小学 5 年纪的学生,所以在找了一些比我强一点的网友+机器翻译,总结出了以下这篇文章。但个人强烈建议去看英文原版

([http://localhost/Ext4/docs/index.html#!/guide/application\\_architecture](http://localhost/Ext4/docs/index.html#!/guide/application_architecture)

)。本段代码示例建议使用 `firefox firebug` 插件配合使用，`ie` 无法加在 `console.log` 对象。

那么，我们开始吧！

对于 `Extjs` 来说，大客户端程序一直很难写，当你为大客户端程序添加更多的功能和项目的时候，项目的体积往往迅速增长。这样的大客户端程序很难组织和维持，所以，`Extjs4` 配备了一个新的应用程序体系结构，它能结构化你的代码，那就是 `Extjs4 MVC`。

`Extjs4 MVC` 有别于其他 `MVC` 架构，`Extjs` 有他自己定义：

1、`Model` 是一个 `Field` 以及他的 `Data` 的集合，`Modes` 知道如何通过 `Stores` 来表示数据，以能用于网格和其他组件。模型的工作很像 `Extjs3` 的记录集（`Record class`），通常通过数据加载器（`Stores`）渲染至网格（`grid`）和其他组件上边。

2、`View`：用以装载任何类型的组件—`grid`、`tree` 和 `panel` 等等。

3、`Controller`—用来放使得 `app` 工作的代码，例如 `render views`，`instantiating Models` 或者其他应用逻辑。

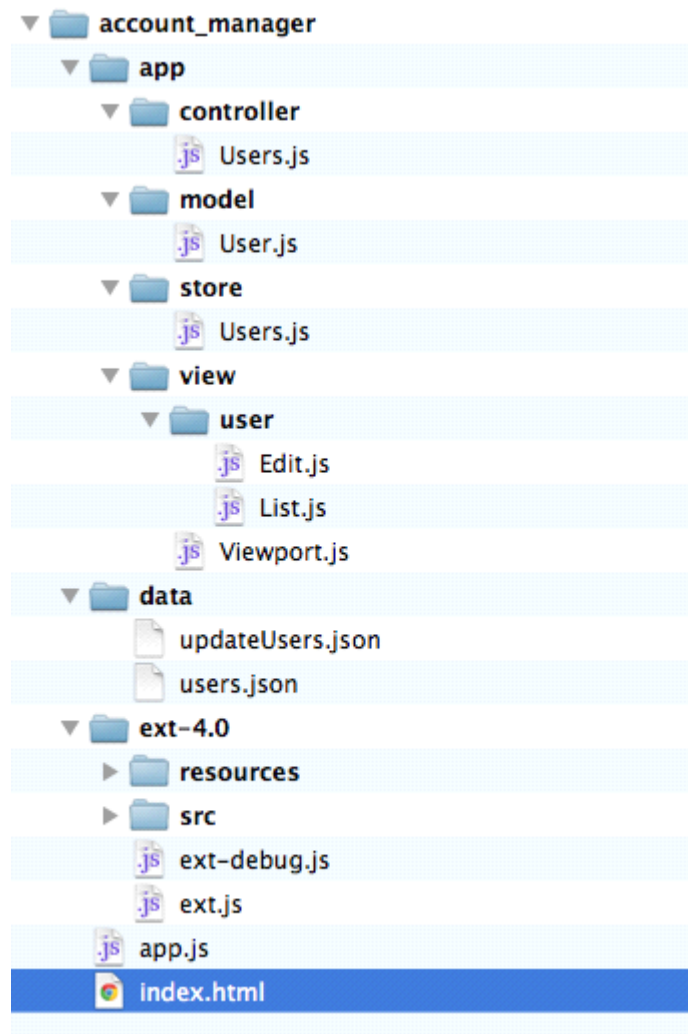
本篇文章，我们将创建一个非常简单的应用程序，即用户数据管理，最后，你就会知道如何利用 `Extjs4 MVC` 去创建简单应用程序。`Extjs4 MVC` 应用程序架构提供应用程序的结构性和一致性。这样的模式带来了一些重要的好处：

- 1、 每个应用程序的工作方式相同，我们只需要学习一次。
- 2、 应用程序之间的代码共享很容易，应为他们所有的工作方式都相同
- 3、 你可以使用 `EXTJS` 提供的构建工具创建你应用程序的优化版本。

既然是介绍 `Extjs4 MVC`，那么，我们开始创建这个应用。

### 3.2.1 文件结构（File Structure）：

`Extjs4 MVC` 的应用程序和别的应用程序一样都遵循一个统一的目录结构。在 `MVC` 布局中，所有的类放在应用程序文件夹，它的子文件夹中包含您的命名空间，模型，视图，控制器和存储器。下面来通过简单的例子来看下怎样应用。



在这个例子中，我们将整个应用程序放到一个名为“account\_manager”的文件夹下，“account\_manager”文件夹中的结构如上图。现在编辑 index.html，内容如下：

```
<html>
<head>
  <title>Account Manager</title>

  <link rel="stylesheet" type="text/css" href="ext-4.0/resources/css/ext-
all.css">

  <script type="text/javascript" src="ext-4.0/ext-debug.js"></script>

  <script type="text/javascript" src="app.js"></script>
</head>
<body></body>
</html>
```

### 3.2.2 创建 app.js 文件 (Creating the application)

所有 Extjs4 MVC 应用从 Ext.application 的一个实例开始，应用程序中应包含全局设置、以及应用程序所使用的模型(model)，视图 (view) 和控制器 (controllers)，一个应用程序还应该包含一个发射功能(launch function)。

现在来创建一个简单的账户管理应用。首先，需要选择一个命名空间（所有 extjs4 应用应该使用一个单一的全局变来来作为命名空间）。暂时，使用“AM”来作为命名空间。

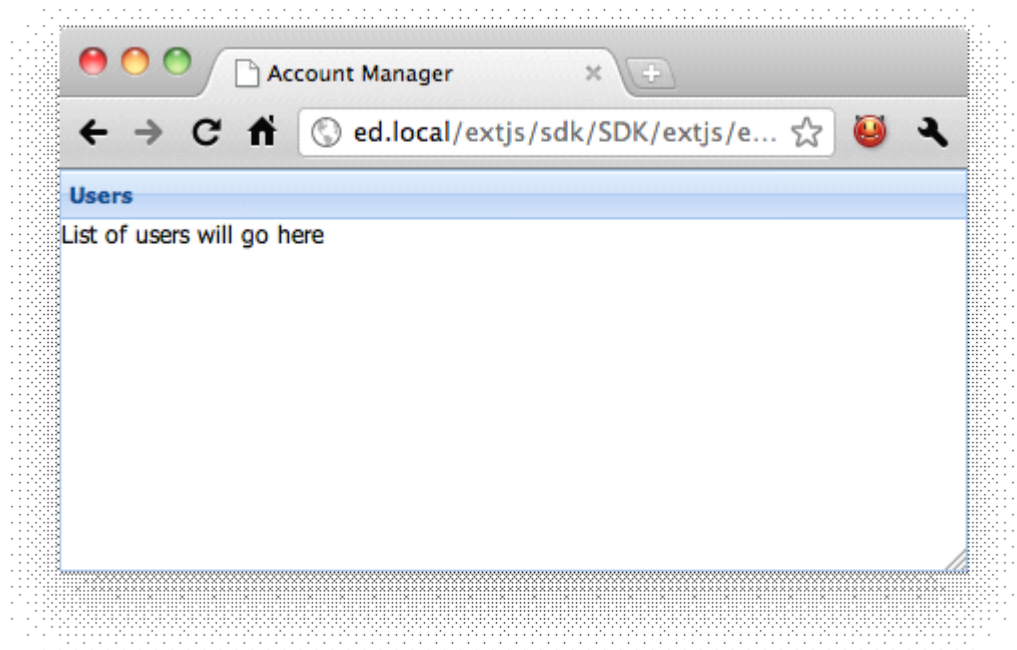
```
Ext.application({
    name: 'AM',

    appFolder: 'app',

    launch: function() {
        Ext.create('Ext.container.Viewport', {
            layout: 'fit',
            items: [
                {
                    xtype: 'panel',
                    title: 'Users',
                    html: 'List of users will go here'
                }
            ]
        });
    }
});
```

以上代码，做了如下几件事。首先，调用 Ext.application 创建一个应用程序类的实例，设置了一个“AM”的命名空间，他将作为整个应用的全局变量，也将作为 Ext.Loader 的命名空间，然后通过 appFolder 来指定配置选项设置相应的路径。最后，创建了一个简单的发射功能，这里仅仅创建了一个 Viewport，其中包含一个 panel，使其充满整个窗口。





### 3.2.3 定义一个控制器 (Defining a Controller)

控制器是整个应用程序的关键，他负责监听事件，并对某些时间做出相应的动作。现在我们创建一个控制器，将其命名为 `Users.js`，其路径是 `app/controller/Users.js`。然后，我们为 `Users.js` 添加如下代码：

```
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',

    init: function() {
        console.log('Initialized Users! This happens before the Application launch function is called');
    }
});
```

完成之后，我们将创建好的控制器添加到程序配置文件：`app.js` 中：

```
Ext.application({
    ...

    controllers: [
        'Users'
    ],

    ...
});
```

当我们访问 `index.html` 时，用户控制器(`Users.js`)自动加载，因为我们在上面的 `app.js` 中的定义中指定了。Init 函数一个最棒的地方是控制器与视图的交互，这

里的交互是指控制功能，因为他很容易就可以监听到视图类的事件处理函数，并采取相应的措施，并及时渲染相关信息到面板上来。

编写 Users.js：

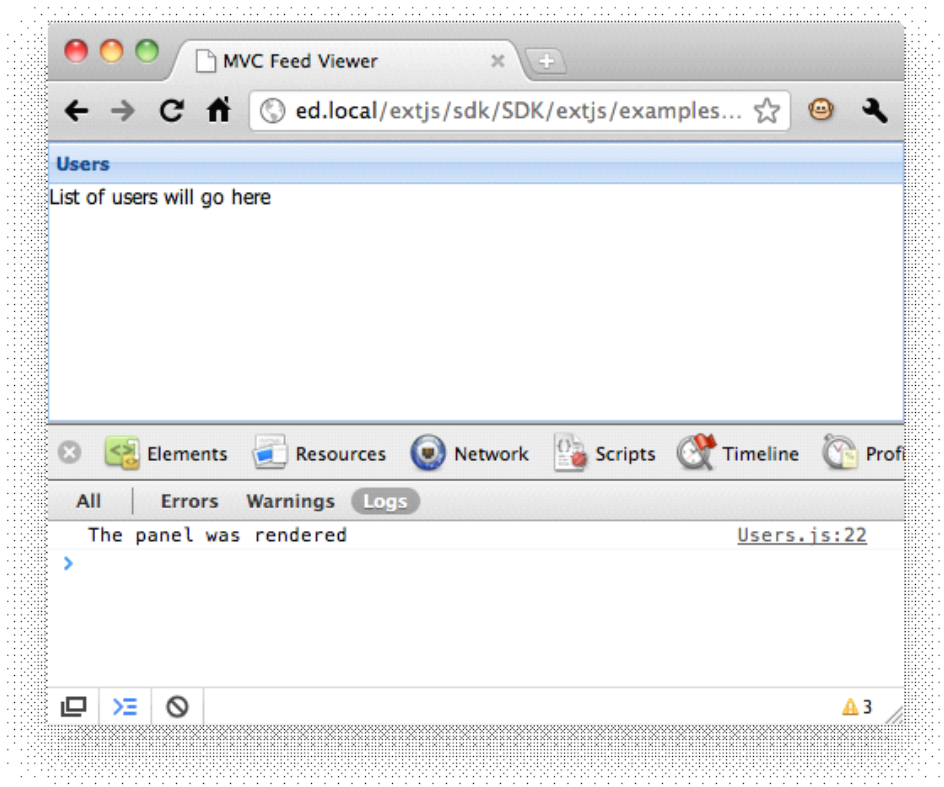
```
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',

    init: function() {
        this.control({
            'viewport > panel': {
                render: this.onPanelRendered
            }
        });
    },

    onPanelRendered: function() {
        console.log('The panel was rendered');
    }
});
```

在 Users.js 中，init 函数使用 this.control 来负责监听，由于使用了新的 [ComponentQuery](#) 引擎，所以可以快速方便的找到页面上组件的引用(viewport > panel),这有些类似 CSS 选择器，通过匹配，快速的找到相匹配的组件。

在上面的 init 函数中，我们使用 viewport > panel，来找到 app.js 中 Viewport 下的 panel 组件，然后，我们提供了一个对象的处理函数(this.onPanelRendered，注意，这里的对象是 this，他的处理函数是 onPanelRendered)。整个效果是，只要符合触发 render 事件的任何组件，那么 onPanelRendered 函数将被调用。当运行我们的应用程序，我们将看到以下内容。



### 3.2.4 定义一个视图 (Defining a View)

到目前为止，应用程序只有几行，也只有两个文件，`app.js` 和 `app/controller/Users.js`。现在我们来增加一个 `grid`，来显示整个系统中的所有用户。作为视图组件的一个子类，我们创建一个新的文件，并把他放到 `app/view/user` 目录下。命名为 `List.js`。整个路径是这样的。 `app/view/user/List.js`，下面，我们写 `List.js` 的代码：

```
Ext.define('AM.view.user.List', {
    extend: 'Ext.grid.Panel',
    alias : 'widget.userlist',

    title : 'All Users',

    initComponents: function() {
        this.store = {
            fields: ['name', 'email'],
            data : [
                {name: 'Ed',    email: 'ed@sencha.com'},
                {name: 'Tommy', email: 'tommy@sencha.com'}
            ]
        };

        this.columns = [
            {header: 'Name', dataIndex: 'name', flex: 1},
            {header: 'Email', dataIndex: 'email', flex: 1}
        ];
    }
});
```

```
        this.callParent(arguments);
    }
});
```

我们创建好的这个类，只是一个非常普通的类，并没有任何意义，为了能让我们更好的使用这个定义好的类，所以我们使用 `alias` 来定义一个别名，这个时候，我们的类可以使用 `Ext.create()` 和 `Ext.widget()` 创建，在其他组件的子组件中，也可以使用 `xtype` 来创建。

```
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',

    views: [
        'user.List'
    ],

    init: ...

    onPanelRendered: ...
});
```

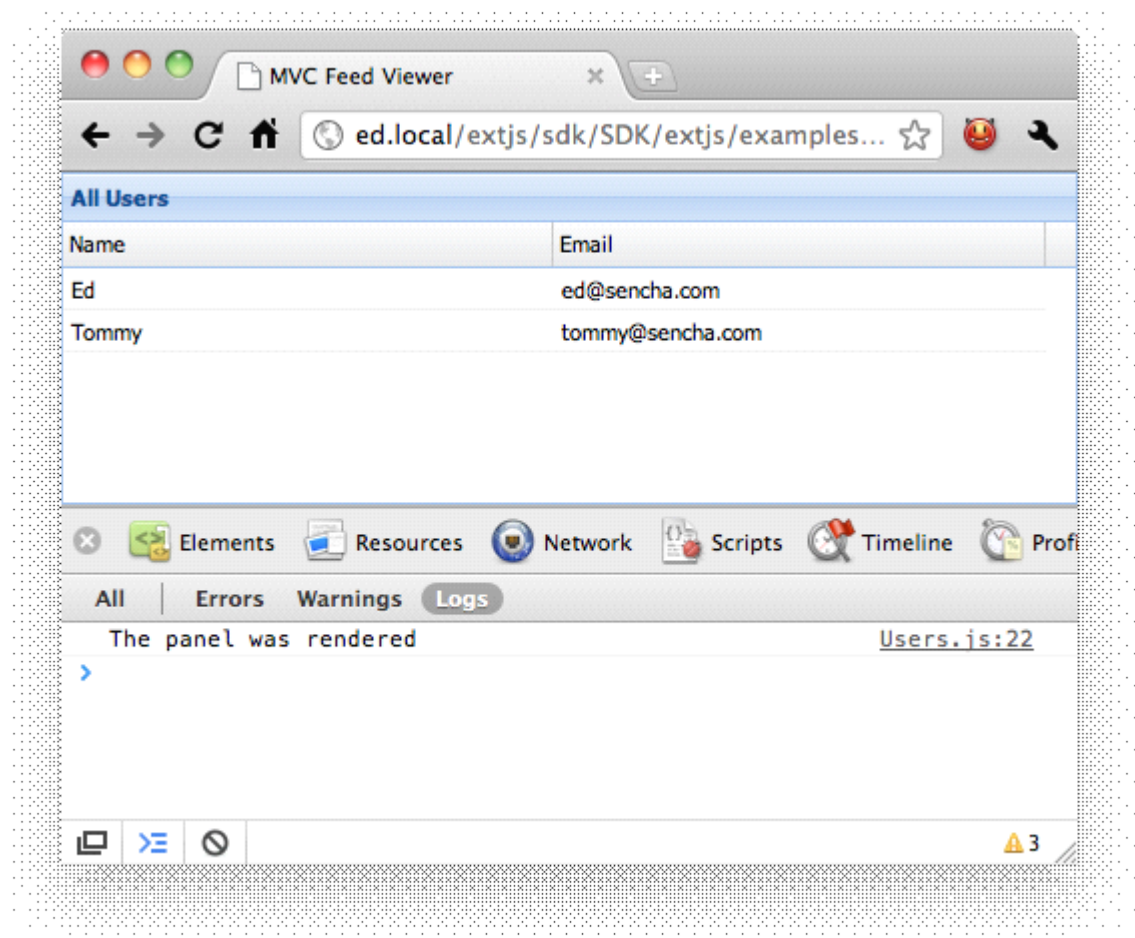
修改 `Users.js`, 增加 `views` 属性，修改 `app.js` 中的 `launch` 方法，将 `List` 渲染到 `Viewport`。

```
Ext.application({
    ...

    launch: function() {
        Ext.create('Ext.container.Viewport', {
            layout: 'fit',
            items: {
                xtype: 'userlist'
            }
        });
    }
});
```

看到这里，也许会有人开始抓狂了，这个 `user.List` 到底是怎么来的，为什么要这样写？如果你开始感到疑惑，那么不妨看看 `Ext.Loader` 是如何工作的（参见文档其他部分），在看过 `Ext.Loader` 之后，你就会明白了，`User.List` 就是 `app/view/user` 下的 `List.js` 文件。为什么 `Ext` 要从 `view` 下找呢？因为我们在控制器中定了 `views: ['user.List']`。这就是 `Extjs` 动态加载的强大之处，具体

Ext.Loader，请看本站的其他文章，你就会明白了。当我们刷新页面。



### 3.2.5 控制网格 (Controlling the grid)

要注意的是, `onPanelRendered` 功能仍然是被调用的。这是因为 `grid` 匹配 `'viewport > panel'`。  
然后我们添加一个监听，当我们双击 `grid` 中的行，就可以编辑用户。

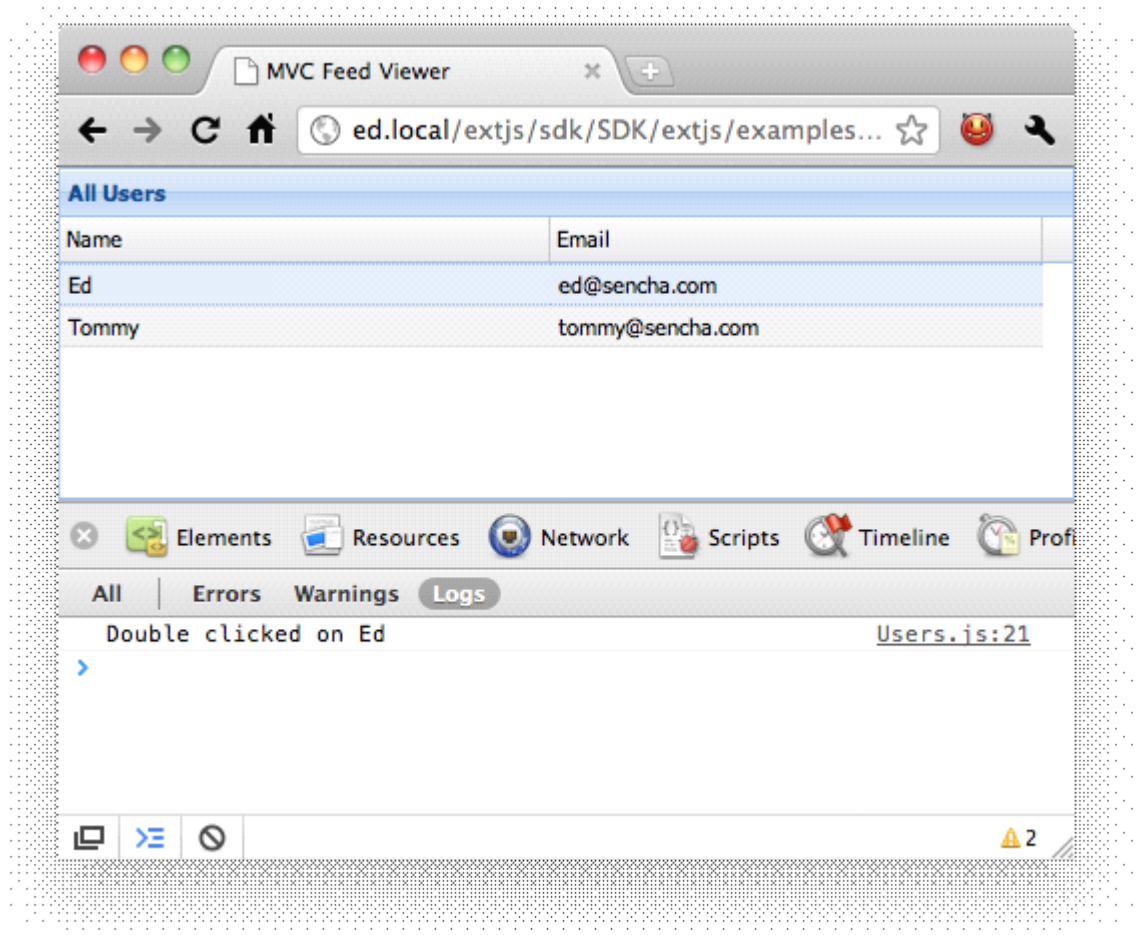
```
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',

    views: [
        'user.List'
    ],

    init: function() {
        this.control({
            'userlist': {
                itemdblclick: this.editUser
            }
        });
    },

    editUser: function(grid, record) {
        console.log('Double clicked on ' + record.get('name'));
    }
});
```

这里，我们修改了 `ComponentQuery` 的选择（`'userlist'`）和事件的名称（`'itemdblclick'`）和处理函数（`'editUser'`）。



如果要实现真正编辑用户，那么我们需要一个真正用于用户编辑 `window`，接着，创建一个 JS 文件。其路径是： `app/view/user/Edit.js`，代码是这样的：

```
Ext.define('AM.view.user.Edit', {
    extend: 'Ext.window.Window',
    alias : 'widget.useredit',

    title : 'Edit User',
    layout: 'fit',
    autoShow: true,

    initComponents: function() {
        this.items = [
            {
                xtype: 'form',
                items: [
                    {
                        xtype: 'textfield',
                        name : 'name',
                        fieldLabel: 'Name'
                    },
                    {
                        xtype: 'textfield',
                        name : 'email',
                        fieldLabel: 'Email'
                    }
                ]
            }
        ]
    }
});
```

```

    }
    ];

    this.buttons = [
        {
            text: 'Save',
            action: 'save'
        },
        {
            text: 'Cancel',
            scope: this,
            handler: this.close
        }
    ];

    this.callParent(arguments);
}
});

```

我们定义了一个子类，继承 `Ext.window.Window`，然后使用 `initComponent` 创建了一个表单和两个按钮，表单中，两个字段分别装载用户名和电子邮件。接下来，我们修改视图控制器，使其可以载入用户数据。

```

Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',

    views: [
        'user.List',
        'user.Edit'
    ],

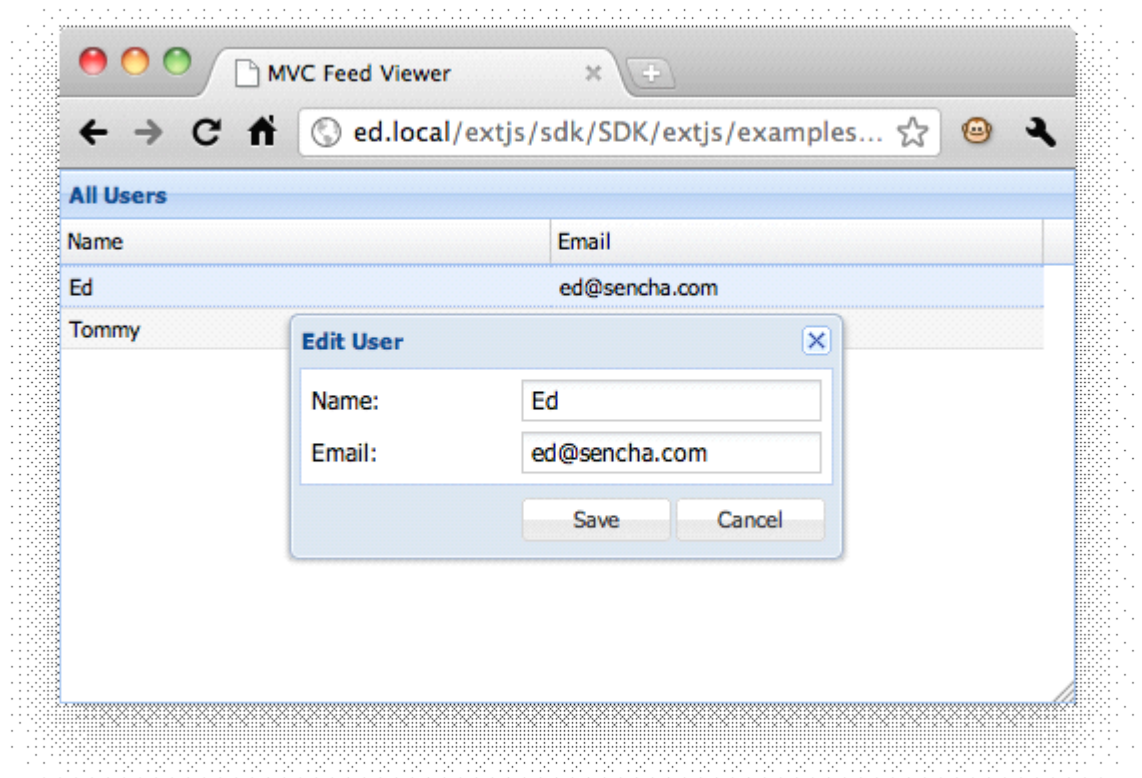
    init: ...

    editUser: function(grid, record) {
        var view = Ext.widget('useredit');

        view.down('form').loadRecord(record);
    }
});

```

首先，我们创建的视图，使用便捷的方法 `Ext.widget`，这是相当于 `Ext.create`（`'widget.useredit'`）。然后我们利用 `ComponentQuery` 快速获取编辑用户的形式引用。在 `Ext JS4` 的每个组件有一个 `down` 函数，可以使用它快速的找到任何他的子组件。当双击 `Grid` 中的行，我们可以看到如下图所示：



### 3.2.6 创建 Model 和 Store

```
Ext.define('AM.store.Users', {
    extend: 'Ext.data.Store',
    fields: ['name', 'email'],
    data: [
        {name: 'Ed', email: 'ed@sencha.com'},
        {name: 'Tommy', email: 'tommy@sencha.com'}
    ]
});
```

接下来修改两个文件:

```
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',
    stores: [
        'Users'
    ],
    ...
});
```

修改 app/view/user/List.js, 使其引用 Users

```
Ext.define('AM.view.user.List', {
    extend: 'Ext.grid.Panel',
    alias : 'widget.userlist',

    //we no longer define the Users store in the `initComponent` method
    store: 'Users',
});
```



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

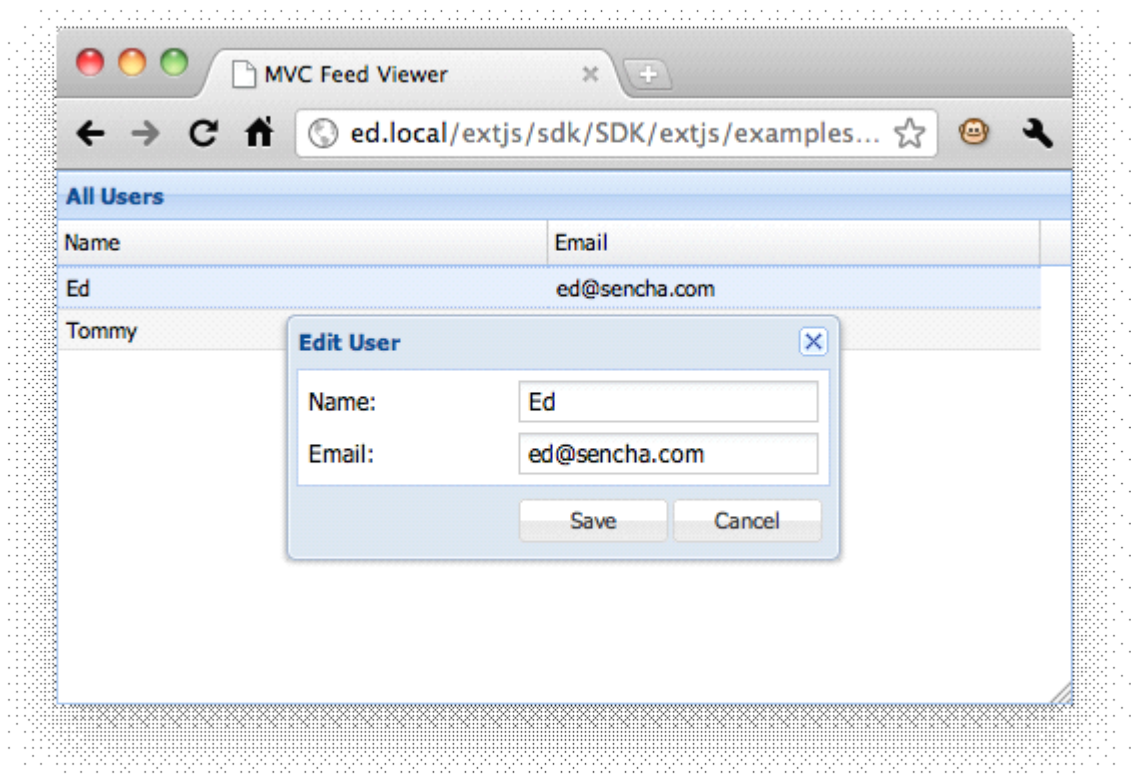
现在，我们定义的用户控制器已经能顺利的加载数据了，到目前，我们所定义的 **Store** 已经足够使用了，但是 **Extjs4** 提供了一个强大的 **Ext.data.Model** 类，不如，我们利用它来重构下我们 **Store**，创建 **app/model/User.js**

```
Ext.define('AM.model.User', {  
    extend: 'Ext.data.Model',  
    fields: ['name', 'email']  
});
```

创建好模型之后，我们将他引入用户控制：

```
//the Users controller will make sure that the User model is included on the page  
and available to our app  
Ext.define('AM.controller.Users', {  
    extend: 'Ext.app.Controller',  
    stores: ['Users'],  
    models: ['User'],  
    ...  
});  
  
// we now reference the Model instead of defining fields inline  
Ext.define('AM.store.Users', {  
    extend: 'Ext.data.Store',  
    model: 'AM.model.User',  
  
    data: [  
        {name: 'Ed',    email: 'ed@sencha.com'},  
        {name: 'Tommy', email: 'tommy@sencha.com'}  
    ]  
});
```

完成上面的代码后，刷新页面，看到的结果和以前的一样。



### 3.2.7 通过 model 保存数据

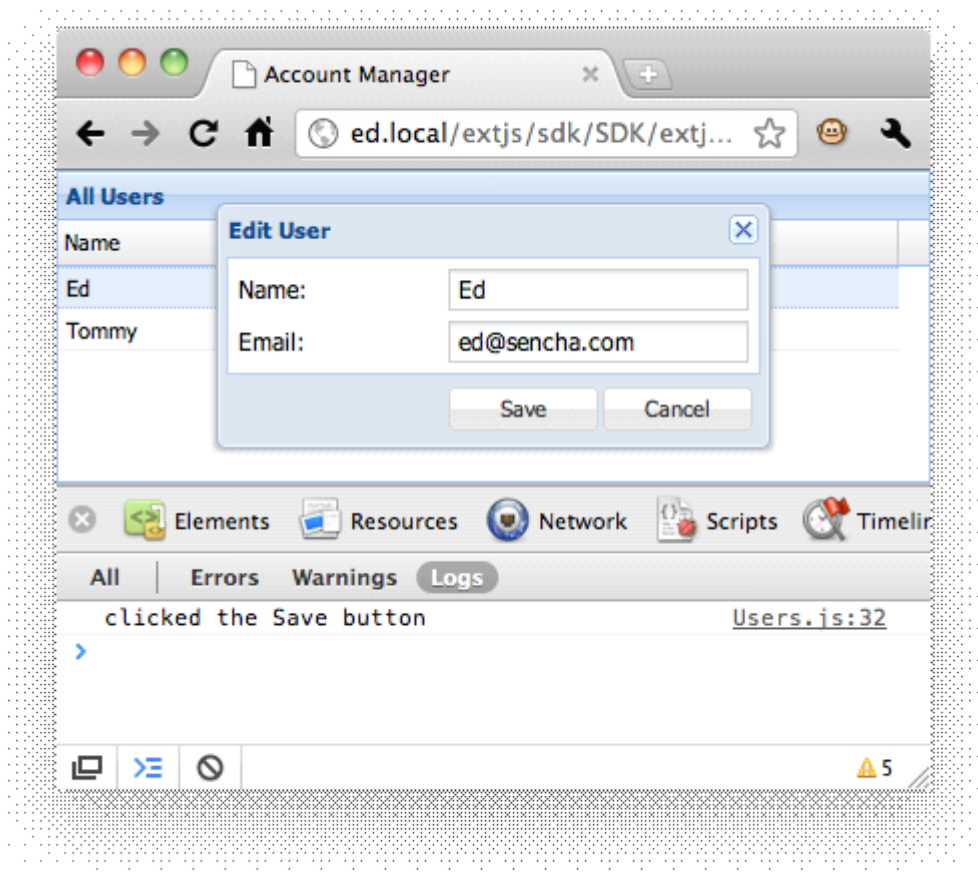
现在双击 Grid 中的行，会弹出编辑用户的 window，实现 Save 来保存用户数据，我们需要修改 init 函数。

```
Ext.define('AM.controller.Users', {
    init: function() {
        this.control({
            'viewport > userlist': {
                itemdblclick: this.editUser
            },
            'useredit button[action=save]': {
                click: this.updateUser
            }
        });
    },

    updateUser: function(button) {
        console.log('clicked the Save button');
    }
});
```

在 this.control 中，我们增加了一个选择项，'useredit button[action=save]'，当 ComponentQuery 找到符合的组件（button 按钮，并且

action 动作为 save)，给他增加一个方法 click，事件为 updateUser。如图：

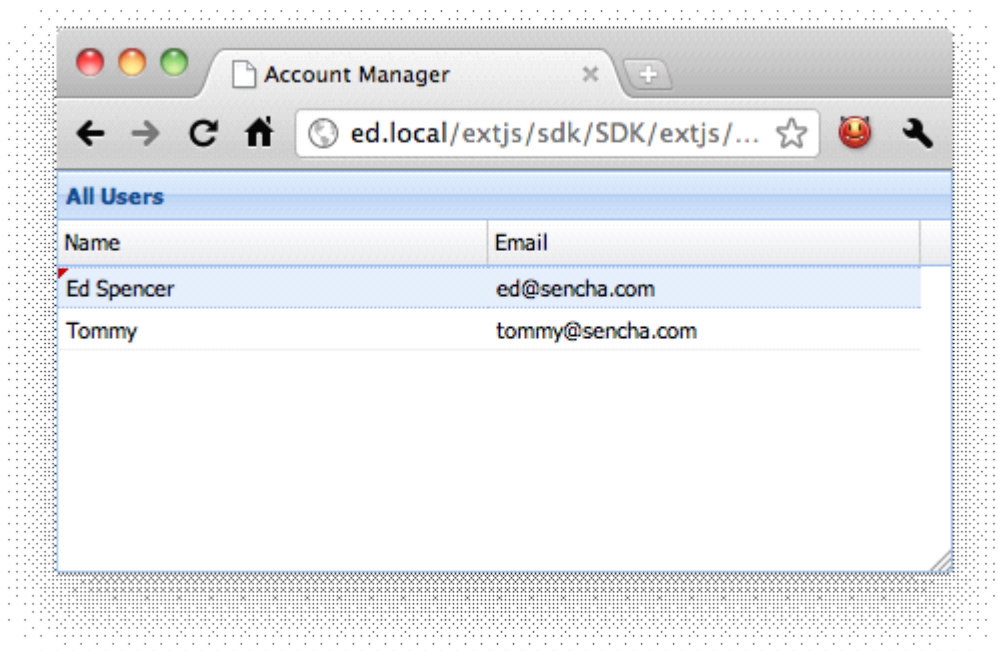


上图中，可以看到正确的 Click 事件，在 updateUser 函数中，需要一个正式的逻辑，来完成用户数据的更新。

```
updateUser: function(button) {  
    var win    = button.up('window'),  
        form   = win.down('form'),  
        record = form.getRecord(),  
        values = form.getValues();  
  
    record.set(values);  
    win.close();  
}
```

这里，当我们点击 Save 按钮，我们将按钮本身传入函数 updateUser，这时，我们使用 button.up('window') 来获取用户 window 的引用，然后使用 win.down('form') 来获取表单的引用，然后获取表单的记录，获取表单中的值，再设置记录为新的值，最后关闭 window。

修改数据后点击 **Save**，更新完成。



实际应用过程中，我们需要将数据保存到服务器，所以，需要修改文件，达到想要的目的。

### 3.2.8 将数据保存到服务器

这很容易做到。使用 **AJAX** 来实现即可。

```
Ext.define('AM.store.Users', {
  extend: 'Ext.data.Store',
  model: 'AM.model.User',
  autoLoad: true,

  proxy: {
    type: 'ajax',
    url: 'data/users.json',
    reader: {
      type: 'json',
      root: 'users',
      successProperty: 'success'
    }
  }
});
```

在 **AM.store.Users**，移除 **data**，用一个代理（**proxy**）取代它，用代理的方式来加载和保存数据。在 **Extjs4** 中，代理的方式有 **AJAX**，**JSON-P** 和 **HTML5 localStorage**，这里使用 **AJAX** 代理。数据从 **data/users.json** 中得到。

我们还用 **reader** 来解析数据，还指定了 **successProperty** 配置。具体请查看 **Json Reader**，把以前的数据复制到 **users.json** 中。得到如下形式：

```
{
  success: true,
  users: [
    {id: 1, name: 'Ed',    email: 'ed@sencha.com'},
    {id: 2, name: 'Tommy', email: 'tommy@sencha.com'}
  ]
}
```

到这里，唯一的变化是将 **Stroe** 的 **autoLoad** 设置为了 **true**，这要求 **Stroe** 的代理要自动加载数据，当刷新页面，将得到和之前一样的效果。

最后一件事情，是将修改的数据发送到服务器，对于本例，服务端只用了一个静态的 **JSON**，所以我们不会看到有任何变化，但至少可以确定这样做是可行的，相信服务端处理数据能力，大家都应该有。本例用，做个小小的变化，即新的代理中，使用 **api** 来更新一个新的 **URL**。

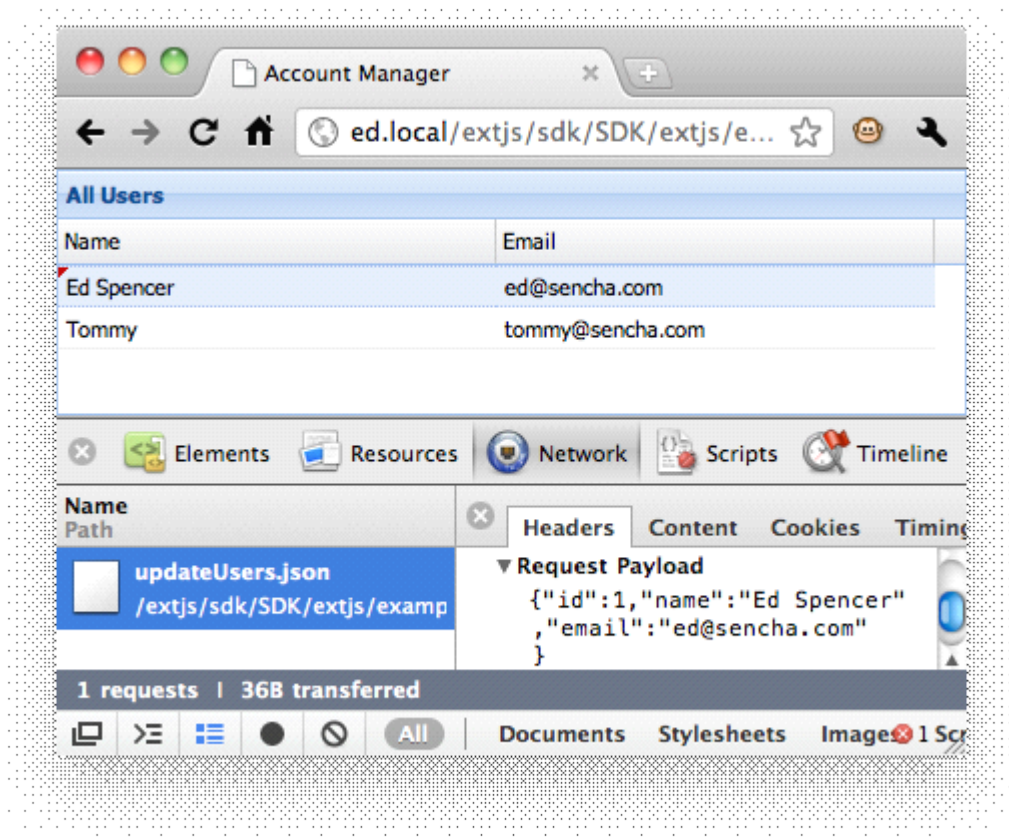
```
proxy: {
  type: 'ajax',
  api: {
    read: 'data/users.json',
    update: 'data/updateUsers.json'
  },
  reader: {
    type: 'json',
    root: 'users',
    successProperty: 'success'
  }
}
```

再来看看是如何运行的，我们还在读取 **users.json** 中的数据，而任何更新都将发送到 **updateUsers.json**，在 **updateUsers.json** 中，创建一个虚拟的回应，从而让我们知道事件确实已经发生。**updateUsers.json** 只包含了 **{"success": true}**。而我们唯一要做的事情是服务的同步编辑。在 **updateUser** 函数增加这样一个功能。

```
updateUser: function(button) {
  var win    = button.up('window'),
      form   = win.down('form'),
      record = form.getRecord(),
      values = form.getValues();

  record.set(values);
  win.close();
  this.getUsersStore().sync();
}
```

现在，运行这个完整的例子，当双击 **grid** 中的某一样并进行编辑，点击 **Save** 按钮后，得到正确的请求和回应。



至

此，整个 MVC 模式全部完成，下一节，将讨论控制器（Controller）和模式（patterns），而这些，可以使应用代码更少，更容易维护

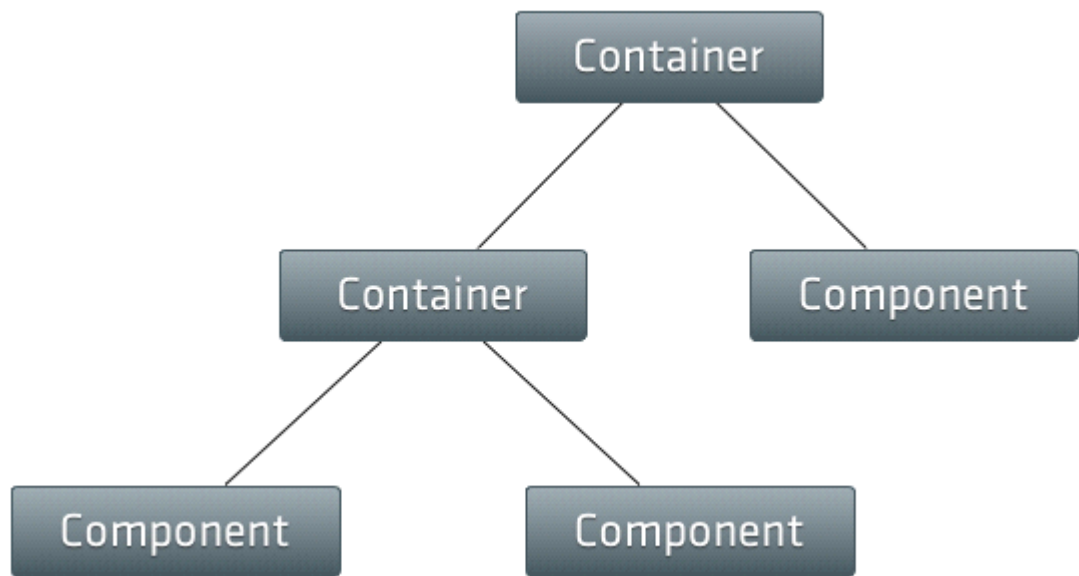
## 3.3 布局和容器

extjs4.0 布局和容器（Layouts and Containers）是 Ext JS 中最强大的部分之一。它负责控制你应用程序中每个组件的尺寸和定位。本文内容包括了如何运用布局的基础。

### 3.3.1 容器

一个 Ext JS 应用程序的图形用户界面(UI)是由许多组件（查看组件指南（[Components Guide](#)）获取更多关于组件的信息）构成的。容器是一种可以容纳一些其他组件的特殊类型组件。

典型的 Ext JS 应用程序是由一些嵌套的组件构成不同的层来组成的。如下图：



最通用的容器就是 [Panel](#)。让我们看下如何创建一个容器以允许一个 Panel 包含其他的组件：

```
Ext.create('Ext.panel.Panel', {
    renderTo: Ext.getBody(),
    width: 400,
    height: 300,
    title: 'Container Panel',
    items: [
        {
            xtype: 'panel',
            title: 'Child Panel 1',
            height: 100,
            width: '75%'
        },
        {
            xtype: 'panel',
            title: 'Child Panel 2',
            height: 100,
            width: '75%'
        }
    ]
})
```

```
});
```

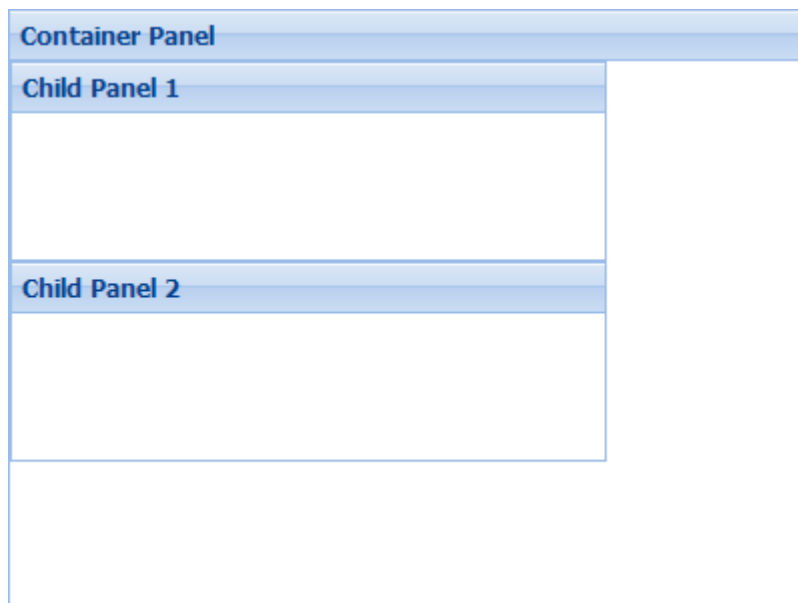
我们刚刚创建了一个 Panel，并把它渲染到 DOM 的 body 中，然后我们使用 [items](#) 配置项添加两个子 Panel 到我们的 Panel 容器中。

### 3.3.2 布局

每个容器都拥有一个布局来管理它子组件的尺寸和定位。

下面我们将讨论如何使用指定类型的布局来配置容器，以及布局系统是如何使每个组件都保持协调的。

使用布局：上面例子中我们没有为 Panel 容器指定布局。可以看到子 Panel 是一个接着一个被放置的，正如 DOM 中标准的块元素一样。

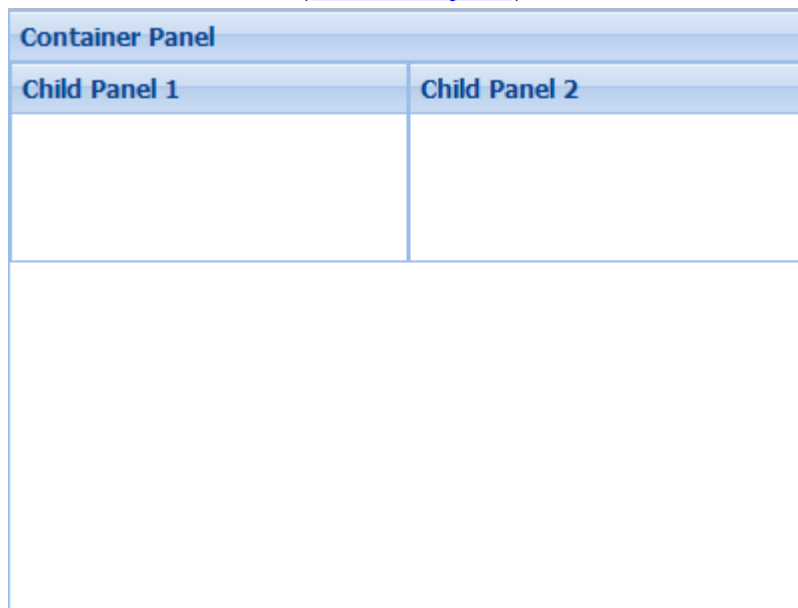


之所以这样是因为所有容器的默认布局为自动布局。自动布局并不为子元素指定任何特殊的定位和尺寸规则。

例如，我们假定想让两个子 Panel 并排放置，并且每个占据父容器宽度的 50%，我们可以简单



地使用一个列布局([Column Layout](#))，通过在父容器中配置 layout 选项来实现。



代码：

```
1. Ext.create('Ext.panel.Panel', {
2.     renderTo: Ext.getBody(),
3.     width: 400,
4.     height: 200,
5.     title: 'Container Panel',
6.     layout: 'column',
7.     items: [
8.         {
9.             xtype: 'panel',
10.            title: 'Child Panel 1',
11.            height: 100,
12.            width: '50%',
13.        },
14.        {
15.            xtype: 'panel',
16.            title: 'Child Panel 2',
17.            height: 100,
18.            width: '50%',
19.        }
20.    ]
21. });
```

Ext JS 拥有一整套可用的布局，几乎容纳了你能想象到的任何一种布局。可以查看布局的例子(Layout Examples)以了解那些布局是可行的。

### 布局系统是如何工作的呢？

父容器的布局负责其所有子元素的初始定位和尺寸大小。框架内部调用了容器的 [doLayout](#) 方法，该方法触发布局为父容器的所有子元素计算得到正确的尺寸和定位并更新 DOM。doLayout 方法是递归调用的，所以父容器的任何子元素同样也将会调用它们的 doLayout 方法。这种调

用将持续到到达组件层次的末端。通常你一般会在你的应用程序代码中调用 `doLayout` 方法，因为框架已为你调用了。

父容器的改变尺寸时，或当添加或删除子组件的 `items` 时，重新布局将被触发。通常，我们仅依赖框架为我们管理以更新布局，但是有时我们想阻止框架自动更新布局，这样我们一次可以批量地处理多个操作，完成时，我们手动地触发布局。

为了到达这个目的，我们在容器中使用延缓布局 ([suspendLayout](#)) 标志以阻止它自动布局，当我们执行那些通常会触发布局的操作时（例如添加或移除 `items`）。

当我们做完这些操作时，我们必须把延缓布局标志关闭，并且手动地调用容器的 `doLayout` 方法以触发布局：

```
1.  var containerPanel = Ext.create('Ext.panel.Panel', {
2.      renderTo: Ext.getBody(),
3.      width: 400,
4.      height: 200,
5.      title: 'Container Panel',
6.      layout: 'column',
7.      suspendLayout: true // Suspend automatic layouts while we do several different things that could trigger a layout on their own
8.      //当我们做一些能触发其自动布局的操作时，保证延缓其能自动布局。
9.  });
10. // Add a couple of child items. We could add these both at the same time by passing an array to add(),
11. //添加一对子 items。我们可以通过调用 add()并传递一个数组一次性添加这两个子组件，
12. // but lets pretend we needed to add them separately for some reason.
13. //但是我们假定需要分别地添加它们是因为其他原因。
14. containerPanel.add({
15.     xtype: 'panel',
16.     title: 'Child Panel 1',
17.     height: 100,
18.     width: '50%'
19. });
20. containerPanel.add({
21.     xtype: 'panel',
22.     title: 'Child Panel 2',
23.     height: 100,
24.     width: '50%'
25. });
26. // Turn the suspendLayout flag off.
27. //关闭延缓布局标志
28. containerPanel.suspendLayout = false;
29. // Trigger a layout.
30. //触发布局
31. containerPanel.doLayout();
```

### 3.3.3 组件布局

如容器的布局定义了一个容器如何设定尺寸和定位它的组件 items，组件的布局同样也定义了其如何设定尺寸和定位它内部的子 items。

组件的布局通过使用组件布局([componentLayout](#))来设置配置选项。一般你不需要使用该配置项，除非你正在编写一个自定义的组件，因为所有提供的组件其内部元素的大小调整和定位都拥有它们自己的布局管理器。

大多数组件使用自动布局([Auto Layout](#))，但是更多复杂的组件需要自定义的组件布局方式（例如一个有 header，footer，toolbars 的 Panel）

自此，我们了解了容器及布局，并且掌握了基本的布局方式和容器的布局方法。下章，我们将详细介绍组件的一些使用方法。

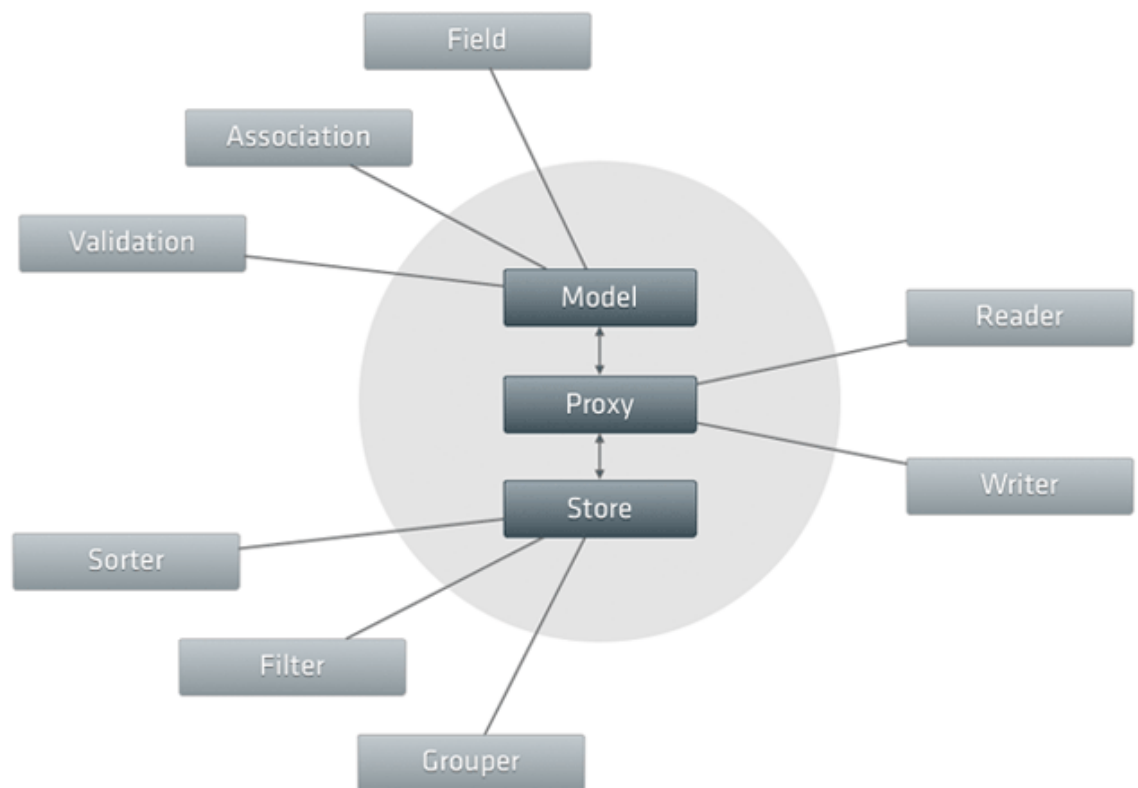
## 3.4 数据

[上一节](#)，我们介绍了下布局及容器，本节本来要介绍下组件，但是组件方面的资料暂时没有翻译完成，所以本节先介绍下数据（Data）。

### 数据

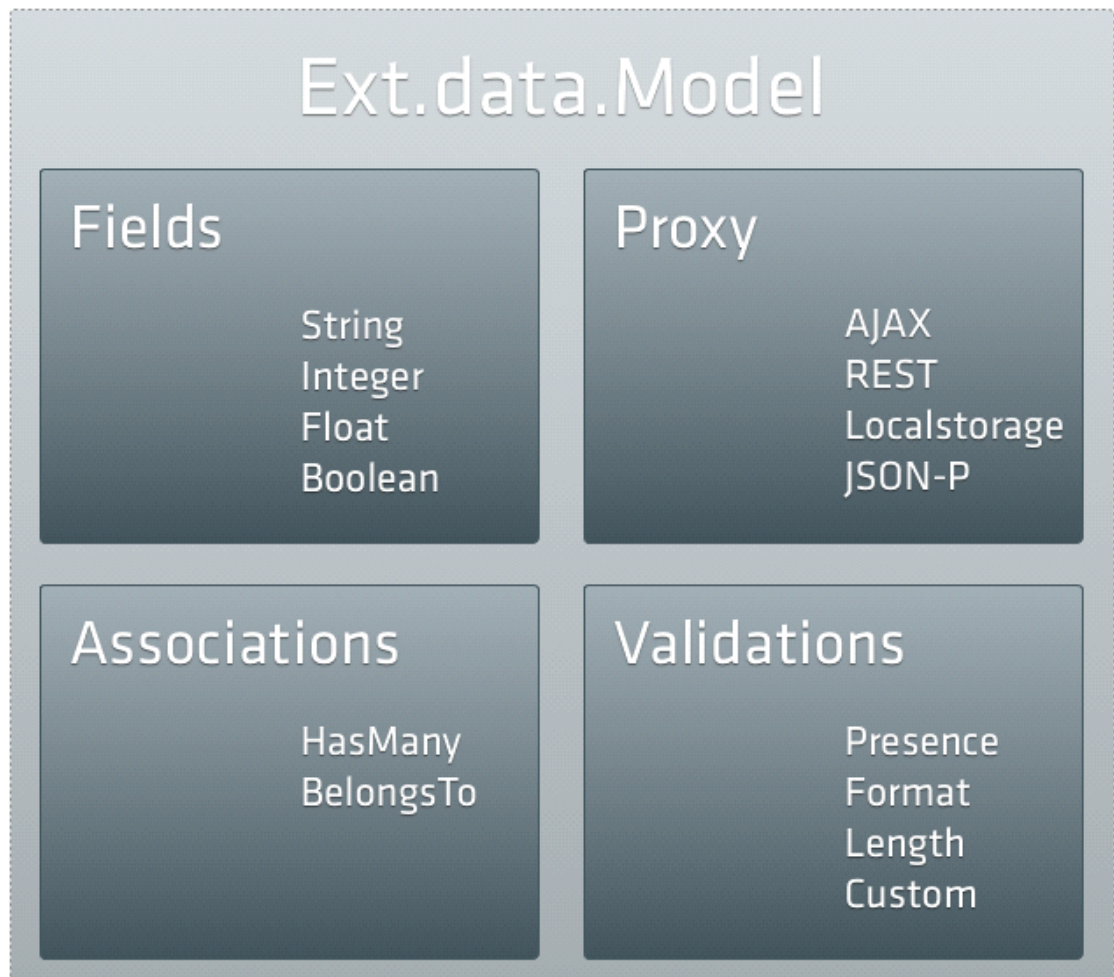
Data 包负责加载和保存你应用程序中的所有数据，由 41 个类构成，其中有三个类是最重要的，分别是模型类([Model](#))，存储类([Store](#))，代理类([Ext.data.proxy.Proxy](#))。它们几乎在每个应用程序中都被使用到，并且有很多相附类为它们提供支持。

# Data Package in Ext JS 4



## 3.4.1 模型类和存储类

模型类([Ext.data.Model](#))是 data 包的核心部件。每个模型代表应用程序中的一些数据类型-例如一个电子商务应用程序可以有 Users、Products、Orders 等模型类。简单来说,模型仅仅是一些域和每个域对应数据的集合。我们将重点研究模型类中的四个主要部分-域([Fields](#))、代理([Proxies](#))、关联([Associations](#))和验证([Validations](#))。



现在让我们看看如何创建一个模型类：

```
Ext.define('User', {
    extend: 'Ext.data.Model',
    fields: [
        { name: 'id', type: 'int' },
        { name: 'name', type: 'string' }
    ]
});
```

模型类通常在存储类中使用，这些存储类主要是一些模型实例的集合。设置存储类和加载它的数据是很简单的：

```
Ext.create('Ext.data.Store', {
    model: 'User',
    proxy: {
        type: 'ajax',
        url: 'users.json',
        reader: 'json'
    },
    autoLoad: true
});
```

```
});
```

我们用 Ajax 代理( [Ajax Proxy](#))配置我们的存储类，告诉它加载数据的 url 地址和用来解码数据的读取器([Reader](#))。这样，我们的服务器将返回 JSON 格式的数据，我们可以使用设置的 Json 读取器( [Json Reader](#))来解析响应。上面创建的存储类实例从 url 地址 users.json 中自动加载一系列 User 模型类实例的数据。

users.json 应该返回如下所示的 JSON 字符串：

```
1.  {
2.    success: true,
3.    users: [
4.      { id: 1, name: 'Ed' },
5.      { id: 2, name: 'Tommy' }
6.    ]
7.  }
8.
```

请查看 [Simple Store](#) 获取一个演示实例。

### 3.4.2 内联数据

存储类实例也可以加载内联数据，它们转换每个传递进 [data](#) 中的对象为模型类实例：

```
1.  Ext.create('Ext.data.Store', {
2.    model: 'User',
3.    data: [
4.      { firstName: 'Ed',    lastName: 'Spencer' },
5.      { firstName: 'Tommy', lastName: 'Maintz' },
6.      { firstName: 'Aaron', lastName: 'Conran' },
7.      { firstName: 'Jamie', lastName: 'Avins' }
8.    ]
9.  });
10.
```

内联数据的例子([Inline Data example](#))

排序和分组

存储类实例能在本地执行排序、过滤和分组，同样也提供远程排序、过滤和分组：

```
1.  Ext.create('Ext.data.Store', {
2.    model: 'User',
3.    sorters: ['name', 'id'],
4.    filters: {
5.      property: 'name',
6.      value    : 'Ed'
7.    },
8.    groupField: 'age',
9.    groupDir: 'DESC'
10. });
11.
```

我们刚刚创建的存储类实例中，数据首先将按照 name 排序，其次按 id 排序；并且数据将被过滤为仅包含 name 为 'Ed' 的 Users，然后数据将按年龄进行分组且遵循由小到大的顺序。任何时候调用存储类的 API 进行排序、过滤和分组都是很轻松的。查看排序、分组、过滤存储类实例 ([Sorting Grouping Filtering Store](#)) 获取一个演示示例。

### 3.4.3 代理

代理类被存储类使用以便于管理加载和保存模型类数据。有两种类型的代理：客户端代理(Client)和服务器端代理(Server)。客户端代理包括存储数据在浏览器内存的内存方式(Memory)和使用 HTML5 本地存储器(可用时)的本地存储方式(Local Storage)。服务器端代理操作一些从远程服务器调度来的数据，例如包括 Ajax, Jsonp 和 Rest 方式。

代理方式可以直接在模型类中定义，如下：

```
1. Ext.define('User', {
2.     extend: 'Ext.data.Model',
3.     fields: ['id', 'name', 'age', 'gender'],
4.     proxy: {
5.         type: 'rest',
6.         url: 'data/users',
7.         reader: {
8.             type: 'json',
9.             root: 'users'
10.        }
11.    }
12. });
13.
14.
15.
16. // Uses the User Model's Proxy
17. Ext.create('Ext.data.Store', {
18.     model: 'User'
19. });
20.
```

这对我们有两方面的好处：首先，使得每个使用 User 模型类的存储类实例以相同方式加载数据变得可行，这样我们避免了必须为每个存储类实例复制相同代理方式的定义。其次，现在我们可以不必使用存储类来加载和保存模型数据：

```
1. // Gives us a reference to the User class
2. // 创建一个User类的引用
3. var User = Ext.ModelMgr.getModel('User');
4. var ed = Ext.create('User', {
5.     name: 'Ed Spencer',
6.     age: 25
7. });
8.
9. // We can save Ed directly without having to add him to a Store first because we
10. // 我们可以直接保存 ed 而不用先把它添加到一个存储类中，因为我们配置了
11. // configured a RestProxy this will automatically send a POST request to the url /users
12. // 一个能自动发送一个 POST 请求到指定 url 的 Rest 代理
13. ed.save({
```

```

14.     success: function(ed) {
15.         console.log("Saved Ed! His ID is " + ed.getId());
16.     }
17. });
18.
19.
20. // Load User 1 and do something with it (performs a GET request to /users/1)
21. //加载 User 1 并对其一些相关操作 (执行一个 GET 请求到 /users/1)
22. User.load(1, {
23.     success: function(user) {
24.         console.log("Loaded user 1: " + user.get('name'));
25.     }
26. });
27.

```

也有利用 HTML5 新功能 -- [LocalStorage](#) 和 [SessionStorage](#) - 的代理模式。尽管旧的浏览器不支持这些新的 HTML5 APIs，它们仍然是有用的，因为很多应用程序将从这些新特性的存在中受益。

直接在模型类中使用代理的例子([Example of a Model that uses a Proxy directly](#))

### 3.4.4 关联

模型类之间可以通过关联 API 链接在一起。大多数应用程序需要处理很多不同的模型类，并且这些模型类之间通常是相关联的。例如一个博客写作应用程序可能有 User(用户)、Post(文章)、Comment(评论)等模型类。每个用户(User)可以创建多篇文章(Posts)，并且每篇文章接受很多评论(Comments)。我们可以如下表示这些关系：

```

1.  Ext.define('User', {
2.      extend: 'Ext.data.Model',
3.      fields: ['id', 'name'],
4.      proxy: {
5.          type: 'rest',
6.          url : 'data/users',
7.          reader: {
8.              type: 'json',
9.              root: 'users'
10.         }
11.     },
12.     hasMany: 'Post' // shorthand for { model: 'Post', name: 'posts' }
13. });
14.
15.
16. Ext.define('Post', {
17.     extend: 'Ext.data.Model',
18.     fields: ['id', 'user_id', 'title', 'body'],
19.     proxy: {
20.         type: 'rest',
21.         url : 'data/posts',
22.         reader: {
23.             type: 'json',
24.             root: 'posts'
25.         }
26.     },
27.     belongsTo: 'User',
28.     hasMany: { model: 'Comment', name: 'comments' }
29. });
30.
31.
32. Ext.define('Comment', {

```



```

33.     extend: 'Ext.data.Model',
34.     fields: ['id', 'post_id', 'name', 'message'],
35.     belongsTo: 'Post'
36. });
37.

```

这将使得在你的应用程序中表示这种复杂关系变得简单。每个模型类可以和其他模型类有任意多的关联，并且你的模型类可以按任意顺序定义。一旦我们创建了一个模型类实例，我们可以很轻松地遍历与其相关联的数据 -- 例如，如果我们想记录一个给定用户的每篇文章的所有相关评论，我们可以如下这样操作：

```

1.  // Loads User with ID 1 and related posts and comments using User's Proxy
2.  //加载 User 使用 ID 1 和相关的文章和评论使用 User 的代理
3.  User.load(1, {
4.      success: function(user) {
5.          console.log("User: " + user.get('name'));
6.          user.posts().each(function(post) {
7.              console.log("Comments for post: " + post.get('title'));
8.              post.comments().each(function(comment) {
9.                  console.log(comment.get('message'));
10.             });
11.         });
12.     }
13. });
14.

```

上例我们创建每一个的 hasMany 关联将产生一个新方法添加到这个模型类上。我们声明的每个 User 模型类实例有许多(hasMany)文章(Posts)，这将为我们的 user.posts()方法，如上面代码段中使用的那样。调用 user.posts()方法将返回一个配置了 Post 模型的存储类实例。依次类推，Post 模型实例获取了一个 comments()方法，因为我们为其设置了 hasMany 评论关联。关联不仅对加载数据来说是有用的，而且对创建新记录也是有用的：

```

1.  user.posts().add({
2.      title: 'Ext JS 4.0 MVC Architecture',
3.      body: 'It\'s a great Idea to structure your Ext JS Applications using the built in MVC Architecture...'
4.  });
5.  user.posts().sync();
6.

```

这里我们实例化了一个新的 Post 模型类，该实例将自动把 User 中的 id 赋值给 Post 中的 user\_id 字段。调用 sync()方法，将通过配置的代理方式来保存新创建的 Post 模型类实例 - 再者，如果你想让操作完成时得到反馈，你可以调用异步操作并传递进一个回调函数来实现。属于(belongsTo)关联也能在模型类实例中生成一个新方法，如我们下面介绍的这个示例：

```

1.  // get the user reference from the post's belongsTo association
2.  //得到 user 实例引用从 post 实例的 belongsTo 关联配置
3.  post.getUser(function(user) {
4.      console.log('Just got the user reference from the post: ' + user.get('name'));
5.  });
6.
7.  // try to change the post's user
8.  //尝试改变文章的用户
9.  post.setUser(100, {
10.      callback: function(product, operation) {

```

```

11.         if (operation.wasSuccessful()) {
12.             console.log('Post\'s user was updated');
13.         } else {
14.             console.log('Post\'s user could not be updated');
15.         }
16.     }
17. });
18.

```

再次说明，加载函数(getUser)是异步调用的，并且需要一个回调函数作为参数来获得 user 实例。setUser 方法仅更新外键（本例中的 user\_id 字段）的值为 100，并保存这个 Post 模型类实例。通常，不管成功与否，当保存操作完成时，传递进去的回调函数都将被触发。

### 3.4.5 加载内嵌的数据

你或许想知道为什么调用 User.load 方法时，我们传递一个 success 方法，但当访问 User 的文章(Post)及评论(Comment)时我们并不需要这样做。这是因为上面的例子中，我们假定当发送请求以获取一个用户的信息时，服务器返回了该用户的数据及所有嵌套的文章和评论的数据。上例我们通过设置关联配置，框架在一次请求中就能自动解析出嵌套的数据。不是靠先发送一个请求获取用户数据，另一个请求获取文章数据，再发送其他请求以获取每篇文章的评论数据这种模式，我们可以在一次服务器响应中返回所有的数据，如下：

```

1.  {
2.      success: true,
3.      users: [
4.          {
5.              id: 1,
6.              name: 'Ed',
7.              age: 25,
8.              gender: 'male',
9.              posts: [
10.                 {
11.                     id : 12,
12.                     title: 'All about data in Ext JS 4',
13.                     body : 'One areas that has seen the most improvement...',
14.                     comments: [
15.                         {
16.                             id: 123,
17.                             name: 'S Jobs',
18.                             message: 'One more thing'
19.                         }
20.                     ]
21.                 }
22.             ]
23.         }
24.     ]
25. }
26. }
27.

```

这些数据将被框架自动解析出来。配置模型类的代理方式以用来加载任何地方的数据会变得很轻松，并且它们的阅读器模式几乎可以处理任何格式的响应数据。和 Ext JS 3 一样，模型类和存储类在整个框架中被许多组件使用，例如表格，树，表单。

查看关联和验证([Associations and Validations](#))的演示示例以获取一个可实际操作并且具有关联关系的模型实例。

当然，你可以以一种非嵌套的方式加载你的数据。如果你仅想需要时加载相关的数据，这种“懒惰加载”模式将可能是有效地。如前所做，我们仅加载 User 数据，除此之外，我们假定返回的响应仅包含 User 数据，没有任何相关联的文章(Post)数据。然后，我们在 `user.posts().load()` 方法添加回调函数中以获取相关的文章(Post)数据：

```
1. // Loads User with ID 1 User's Proxy
2. User.load(1, {
3.     success: function(user) {
4.         console.log("User: " + user.get('name'));
5.         // Loads posts for user 1 using Post's Proxy
6.         user.posts().load({
7.             callback: function(posts, operation) {
8.                 Ext.each(posts, function(post) {
9.                     console.log("Comments for post: " + post.get('title'));
10.                    post.comments().each(function(comment) {
11.                        console.log(comment.get('message'));
12.                    });
13.                });
14.            }
15.        });
16.    }
17. });
18.
```

查看懒惰关联([Lazy Associations](#))模式可以获取一个完整的示例

## 3.4.6 验证

自 Ext JS 4 起，模型类由于提供了验证数据的功能而变得更加丰富精彩了。为了证明这点，我们将在前面使用过的关联例子的基础上构建一个示例。首先让我们添加一些验证到 User 模型类中：

```
1. Ext.define('User', {
2.     extend: 'Ext.data.Model',
3.     fields: ...,
4.     validations: [
5.         {type: 'presence', name: 'name'},
6.         {type: 'length', name: 'name', min: 5},
7.         {type: 'format', name: 'age', matcher: /\d+/},
8.         {type: 'inclusion', name: 'gender', list: ['male', 'female']},
9.         {type: 'exclusion', name: 'name', list: ['admin']}
10.    ],
11.    proxy: ...
12. });
13.
```

验证和域定义遵循相同的代码格式。任何情况下，我们都可以指定一个域和一种验证类型。我们例子中的验证器配置要求 name 域必须存在，并且至少 5 个字符长，age 域必须为数字，gender 域的值只能为 male 或 female，并且用户名可以为除了 admin 外的其他任何名称。一些验证器可能具有其他的可选配置 -- 例如，长度验证可以具有最大和最小属性，格式(format)可以具有匹配(matcher)属性等。Ext JS 有五种内置的验证器，且可以轻松地添加用户自定义规则。首先让我们看看这五种类型：

- 存在(presence) 确保该域必须有确定值。0 被看作有效值，但空字符串将不被视为有效值。
- 长度(length) 确保一个字符串长度位于最大和最小值之间。两个参数都是可选的。
- 格式(format) 确保一个字符串匹配指定的正则表达式。上例中我们确保 age 域必须为数字。
- 包含(inclusion) 确保该域的值在指定的数值集合中（例如确保性别只能是男或女）。

- 排除(exclusion) 确保该域的值不在指定的数值集合中 (例如用户名不能为 admin)  
既然我们已经理解了不同验证器的功能, 让我们尝试在一个 User 实例中使用它们。

我们创建一个 user 实例并在其中运行验证器, 注意产生的任何错误:

```
1. // now lets try to create a new user with as many validation errors as we can
2. // 现在让我们尝试创建一个 user 实例, 并产生尽量多的验证错误
3. var newUser = Ext.create('User', {
4.     name: 'admin',
5.     age: 'twenty-nine',
6.     gender: 'not a valid gender'
7. });
8.
// run some validation on the new user we just created

// 在我们刚刚创建的 user 实例中运行一些验证器

var errors = newUser.validate();

console.log('Is User valid?', errors.isValid()); //returns 'false' as there
were validation errors 当有验证器错误产生时, 返回 false

console.log('All Errors:', errors.items); //returns the array of all errors
found on this model instance 返回该模型类实例所有错误组合成的数组

console.log('Age Errors:', errors.getByField('age')); //returns the errors for
the age field 返回 age 域产生的错误
```

这里的关键函数是 validate(), 该函数运行所有配置验证器并返回一个 Errors 对象。这个简单的对象为所有错误的集合, 并且添加了一些便利的方法, 例如 isValid() -- 当任何域都没有错误产生时, 返回 true, 还有 getByField()方法, 返回给定域产生的所有错误。

## 4 示例

### 4.1 Windows 的创建

Extjs4, 创建 Ext 组件有了新的方式, 就是 Ext.create

Extjs4, 创建 Ext 组件有了新的方式, 就是 Ext.create(...), 而且可以使用动态加载 JS 的方式来加快组件的渲染, 我们再也不必一次加载已经达到 1MB 的 ext-all.js 了, 本文介绍如何在 EXTJS4 中创建一个 window。

代码如下:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>窗口实例</title>
<link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css" />
<script type="text/javascript" src="../extjs/bootstrap.js"></script>
<script type="text/javascript" src="../extjs/locale/ext-lang-zh_CN.js"></script>
<script type="text/jscript">
Ext.require('Ext.window');
Ext.onReady(function(){
    Ext.create('Ext.Window',{
        width:400,
        height:230,
        //X,Y标识窗口相对于父窗口的偏移值。
        x:10,
        y:10,
        plain: true,
        //指示标题头的位置,分别为 top,bottom,left,right,默认为top
        headerPosition: 'left',
        title: 'ExtJS4 Window的创建,头在左'
    }).show();

    Ext.create('Ext.Window',{
        width:400,
        height:230,
        x:500,
        y:10,
        plain: true,
        //指示标题头的位置,分别为 top,bottom,left,right,默认为top
        headerPosition: 'right',
        title: 'ExtJS4 Window的创建,头在右'
    }).show();

    Ext.create('Ext.Window',{
        width:400,
        height:230,
        x:10,
        y:300,
        plain: true,
        //指示标题头的位置,分别为 top,bottom,left,right,默认为top
        headerPosition: 'bottom',
        title: 'ExtJS4 Window的创建,头在底'
    }).show();
    var win = Ext.create('Ext.Window',{
        width:400,
        height:230,
        x:500,
        y:300,
        plain: true,
        //指示标题头的位置,分别为 top,bottom,left,right,默认为top
        headerPosition: 'top',
        title: 'ExtJS4 Window的创建'
    });
    win.show();
});
</script>
</head>

<body>
</body>
</html>

```

运行后浏览器内可以看到四个窗口。

## 4.2 HBOX 的使用

要使用 HBox 布局方式，首先的熟悉下以下几个主要属性：

一、align: 字符类型，指示组件在容器内的对齐方式。有如下几种属性。

- 1、top（默认）：排列于容器顶端。
- 2、middle: 垂直居中排列于容器中。
- 3、stretch: 垂直排列且拉伸以填补容器高度
- 4、stretchmax: 垂直拉伸，并且组件以最高高度的组件为准。

二、flex: 数字类型，指示组件在容器中水平呈现方式，通俗的讲，就是指示组件在容器中的相对宽度。

三、pack : 字符类型，指示组件在容器的位置，有如下几种属性。

- 1、start（默认）：组件在容器左边
- 2、center: 组件在容器中间
- 3、end: 组件在容器的右边

实例代码：

### Html 代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>HBox实例</title>
<link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css" />
<script type="text/javascript" src="../extjs/bootstrap.js"></script>
<script type="text/javascript" src="../extjs/locale/ext-lang-zh_CN.js"></script>
<script type="text/javascript" src="hbox.js"></script>

</head>

<body>

</body>
<div id="d1"></div>
<div id="d2"></div>
<div id="d3"></div>
</html>
```

### Hbox.js

```
Ext.onReady(function(){
    var d1 = Ext.create('Ext.Panel',{
        title:'HBox 顶对齐，且组件在容器的左边',
        frame:true,
        width:600,
        height:100,
        items:[{
            anchor:'100%',
            layout: {
                type:'hbox',
                padding:'10',
                pack:'start',
                align:'top'
```

```

    },
    defaults:{margins:'0 5 0 0'},
    items:[{
        xtype:'button',
        text: 'Button 1'
    },{
        xtype:'button',
        text: 'Button 2'
    },{
        xtype:'button',
        text: 'Button 3'
    },{
        xtype:'button',
        text: 'Button 4'
    }
    ]
}
})

d1.render('d1');

var d2 = Ext.create('Ext.Panel',{
    title:'HBox 垂直对齐, 且组件在容器的右边',
    frame:true,
    width:600,
    height:100,
    items:[{
        anchor:'100%',
        layout: {
            type:'hbox',
            padding:'10',
            align:'middle',
            pack:'end'
        },
        defaults:{margins:'0 5 0 0'},
        items:[{
            xtype:'button',
            text: 'Button 1'
        },{
            xtype:'button',
            text: 'Button 2'
        },{
            xtype:'button',
            text: 'Button 3'
        },{
            xtype:'button',
            text: 'Button 4'
        }
        ]
    }
    ])
})

d2.render('d2');

var d3 = Ext.create('Ext.Panel',{
    title:'HBox 垂直水平居中, 并且所有控件高度为最高控件的高度',
    frame:true,
    width:600,
    height:100,
    items:[{
        anchor:'100%',

        layout: {
            type: 'hbox',
            padding:'5',
            align:'stretchmax',

```

```

        pack: 'center'
    },
    defaults: { margins: '0 5 0 0' },
    items: [
        {
            xtype: 'button',
            text: 'Small Size'
        },
        {
            xtype: 'button',
            scale: 'medium',
            text: 'Medium Size'
        },
        {
            xtype: 'button',
            scale: 'large',
            text: 'Large Size'
        }
    ]
}
})
d3.render('d3');
})

```

运行结果：



## 4.3 VBox 的使用

要使用 VBox 布局方式，首先的熟悉下以下几个主要属性：

- 一、**align**: 字符类型，指示组件在容器内的对齐方式。有如下几种属性。
  - 1、**left**（默认）：排列于容器左侧。
  - 2、**center**：控件在容器水平居中。
  - 3、**stretch**: 控件横向拉伸至容器大小
  - 4、**stretchmax**: 控件横向拉伸，宽度为最宽控件的宽。
- 二、**flex**: 数字类型，指示组件在容器中水平呈现方式，通俗的讲，就是指示组件在容器中的相对宽度。
- 三、**pack**：字符类型，指示组件在容器的位置，有如下几种属性。
  - 1、**start**（默认）：组件在容器上边
  - 2、**center**: 组件在容器中间



### 3、end: 组件在容器的下边

#### Html 代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>VBox实例</title>
<link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css" />
<script type="text/javascript" src="../extjs/bootstrap.js"></script>
<script type="text/javascript" src="../extjs/locale/ext-lang-zh_CN.js"></script>
<script type="text/javascript" src="vbox.js"></script>
<style type="text/css">
    html, body {
        font: normal 12px verdana;
        margin: 0;
        padding: 0;
        border: 0 none;
    }
</style>
</head>

<body>
</body>
</html>
```

#### Vbox.js

```
Ext.onReady(function() {
    var currentName;
    var replace = function(config, name) {
        var btns = Ext.getCmp('btns');
        if (name && name != currentName) {
            currentName = name;
            btns.remove(0);
            btns.add(Ext.apply(config));
        }
    };

    var viewport = Ext.create('Ext.Viewport', {
        layout: 'border',
        items: [{
            id: 'btns',
            region: 'west',
            baseCls: 'x-plain',
            split: true,
            width: 150,
            minWidth: 100,
            maxWidth: 250,
            layout: 'fit',
            margins: '5 0 5 5',
            items: {
                baseCls: 'x-plain',
                html: '<p style="padding:10px;color:#556677;font-size:11px;">点击右边的按钮查看效果</p>'
            }
        }, {
            region: 'center',
            margins: '5 5 5 0',
            layout: 'anchor',
            items: [{
                anchor: '100%',
```

```

        baseCls: 'x-plain',
        layout: {
            type: 'hbox',
            padding: 10
        },
        defaults: {
            margins: '0 5 0 0',
            pressed: false,
            toggleGroup: 'btns',
            allowDepress: false
        },
        items: [{
            xtype: 'button',
            text: 'Spaced / Align: left',
            handler: function() {
                replace({
                    layout: {
                        type: 'vbox',
                        padding: '5',
                        align: 'left'
                    },
                    defaults: { margins: '0 0 5 0' },
                    items: [
                        {
                            xtype: 'button',
                            text: 'Button 1'
                        },
                        {
                            xtype: 'tbspacer',
                            flex: 1
                        },
                        {
                            xtype: 'button',
                            text: 'Button 2'
                        },
                        {
                            xtype: 'button',
                            text: 'Button 3'
                        },
                        {
                            xtype: 'button',
                            text: 'Button 4',
                            margins: '0'
                        }
                    ]
                });
            }
        }, {
            xtype: 'button',
            text: 'Multi-Spaced / Align: left',
            handler: function() {
                replace({
                    layout: {
                        type: 'vbox',
                        padding: '5',
                        align: 'left'
                    },
                    defaults: { margins: '0 0 5 0' },
                    items: [
                        {
                            xtype: 'button',
                            text: 'Button 1'
                        },
                        {
                            xtype: 'tbspacer',
                            flex: 1
                        },
                        {
                            xtype: 'button',
                            text: 'Button 2'
                        },
                        {
                            xtype: 'tbspacer',
                            flex: 1
                        }
                    ]
                });
            }
        }
    ],
    'spaced');
}, {
    xtype: 'button',
    text: 'Multi-Spaced / Align: left',
    handler: function() {
        replace({
            layout: {
                type: 'vbox',
                padding: '5',
                align: 'left'
            },
            defaults: { margins: '0 0 5 0' },
            items: [
                {
                    xtype: 'button',
                    text: 'Button 1'
                },
                {
                    xtype: 'tbspacer',
                    flex: 1
                },
                {
                    xtype: 'button',
                    text: 'Button 2'
                },
                {
                    xtype: 'tbspacer',
                    flex: 1
                }
            ]
        });
    }
}

```

```

        }, {
            xtype: 'button',
            text: 'Button 3'
        }, {
            xtype: 'tbspacer',
            flex: 1
        }, {
            xtype: 'button',
            text: 'Button 4',
            margins: '0'
        }
    ],
    'multi spaced - align left');
    }, {
        xtype: 'button',
        text: 'Align: left',
        handler: function() {
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    align: 'left'
                },
                defaults: { margins: '0 0 5 0' },
                items: [
                    {
                        xtype: 'button',
                        text: 'Button 1'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 2'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 3'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 4'
                    }
                ]
            }, 'align left');
        }
    }, {
        xtype: 'button',
        text: 'Align: center',
        handler: function() {
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    align: 'center'
                },
                defaults: { margins: '0 0 5 0' },
                items: [
                    {
                        xtype: 'button',
                        text: 'Button 1'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 2'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 3'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 4'
                    }
                ]
            }, 'align center');
        }
    }, {

```

```

        xtype: 'button',
        text: 'Align: stretch',
        handler: function(){
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    align: 'stretch'
                },
                defaults: { margins: '0 0 5 0' },
                items: [{
                    xtype: 'button',
                    text: 'Button 1'
                }, {
                    xtype: 'button',
                    text: 'Button 2'
                }, {
                    xtype: 'button',
                    text: 'Button 3'
                }, {
                    xtype: 'button',
                    text: 'Button 4'
                }
            ]
        }, 'align stretch');
    }, {
        xtype: 'button',
        text: 'Align: stretchmax',
        handler: function(){
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    align: 'stretchmax'
                },
                defaults: { margins: '0 0 5 0' },
                items: [{
                    xtype: 'button',
                    text: 'Jamie'
                }, {
                    xtype: 'button',
                    text: 'Aaron'
                }, {
                    xtype: 'button',
                    text: 'Tommy'
                }, {
                    xtype: 'button',
                    text: 'Ed '
                }
            ]
        }, 'align stretchmax');
    }
    ], {
        anchor: '100%',
        baseCls: 'x-plain',
        layout: {
            type: 'hbox',
            padding: '0 10 10'
        },
        defaults: {
            margins: '0 5 0 0',
            pressed: false,
            toggleGroup: 'btns',
            allowDepress: false
        },
        items: [{

```

```

xtype: 'button',
text: 'Flex: Even / Align: center',
handler: function(){
    replace({
        layout: {
            type: 'vbox',
            padding: '5',
            align: 'center'
        },
        defaults: { margins: '0 0 5 0' },
        items: [
            {
                xtype: 'button',
                text: 'Button 1',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 2',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 3',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 4',
                flex: 1,
                margins: '0'
            }
        ]
    }, 'align flex even');
}
}, {
xtype: 'button',
text: 'Flex: Ratio / Align: center',
handler: function(){
    replace({
        layout: {
            type: 'vbox',
            padding: '5',
            align: 'center'
        },
        defaults: { margins: '0 0 5 0' },
        items: [
            {
                xtype: 'button',
                text: 'Button 1',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 2',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 3',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 4',
                flex: 3,
                margins: '0'
            }
        ]
    }, 'align flex ratio');
}
}, {
xtype: 'button',
text: 'Flex + Stretch',
handler: function(){
    replace({

```

```

        layout: {
            type: 'vbox',
            padding: '5',
            align: 'stretch'
        },
        defaults: { margins: '0 0 5 0' },
        items: [
            {
                xtype: 'button',
                text: 'Button 1',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 2',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 3',
                flex: 1
            },
            {
                xtype: 'button',
                text: 'Button 4',
                flex: 3,
                margins: '0'
            }
        ],
        'align flex + stretch');
    },
    {
        xtype: 'button',
        text: 'Pack: start / Align: center',
        handler: function() {
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    pack: 'start',
                    align: 'center'
                },
                defaults: { margins: '0 0 5 0' },
                items: [
                    {
                        xtype: 'button',
                        text: 'Button 1'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 2'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 3'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 4'
                    }
                ],
                'align pack start + align center');
        }
    },
    {
        xtype: 'button',
        text: 'Pack: center / Align: center',
        handler: function() {
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    pack: 'center',
                    align: 'center'
                },
                defaults: { margins: '0 0 5 0' },
                items: [

```

```

        xtype: 'button',
        text: 'Button 1'
    }, {
        xtype: 'button',
        text: 'Button 2'
    }, {
        xtype: 'button',
        text: 'Button 3'
    }, {
        xtype: 'button',
        text: 'Button 4',
        margins: '0'
    }
    ], 'align pack center + align center');
    }, {
        xtype: 'button',
        text: 'Pack: end / Align: center',
        handler: function() {
            replace({
                layout: {
                    type: 'vbox',
                    padding: '5',
                    pack: 'end',
                    align: 'center'
                },
                defaults: { margins: '0 0 5 0' },
                items: [
                    {
                        xtype: 'button',
                        text: 'Button 1'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 2'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 3'
                    },
                    {
                        xtype: 'button',
                        text: 'Button 4',
                        margins: '0'
                    }
                ]
            }, 'align pack end + align center');
        }
    }
    ]
    });
});

```

## 4.4 Grid

Extjs4 Grid 创建还是比较容易的，难记、难理解的地方，也就是数据的获取。下面，就创建一个最简单的 Grid 组件，后面，我们会逐渐丰富这个 Grid 的功能。

HTML 代码：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Grid</title>
<link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css" />

```

```

<script type="text/javascript" src="../../extjs/bootstrap.js"></script>
<script type="text/javascript" src="../../extjs/locale/ext-lang-
zh_CN.js"></script>
<script type="text/javascript" src="grid.js"></script>
</head>

<body>
<div id="demo"></div>
</body>
</html>

```

## Grid.js

```

Ext.require([
    'Ext.grid.*',
    'Ext.data.*'
]);
Ext.onReady(function(){
    Ext.define('MyData',{
        extend: 'Ext.data.Model',
        fields: [
            //第一个字段需要指定mapping, 其他字段, 可以省略掉。
            {name: 'UserName', mapping: 'UserName'},
            'Sex',
            'Age',
            'XueHao',
            'BanJi'
        ]
    });

    //创建数据源
    var store = Ext.create('Ext.data.Store', {
        model: 'MyData',
        proxy: {
            //异步获取数据, 这里的URL可以改为任何动态页面, 只要返回JSON数据即可
            type: 'ajax',
            url: 'mydata.json',

            reader: {
                type: 'json',
                root: 'items',
                //totalProperty : 'total'
            }
        },
        autoLoad: true
    });

    //创建Grid
    var grid = Ext.create('Ext.grid.Panel',{
        store: store,
        columns: [
            {text: "姓名", width: 120, dataIndex: 'UserName', sortable: true},
            {text: "性别", flex: 1, dataIndex: 'Sex', sortable: false},
            {text: "年龄", width: 100, dataIndex: 'Age', sortable: true},
            {text: "学号", width: 100, dataIndex: 'XueHao', sortable: true},
            {text: "班级", width: 100, dataIndex: 'BanJi', sortable: true}
        ],
        height:400,
        width:480,
        x:20,

```



```
y:40,  
title: 'ExtJS4 Grid示例',  
renderTo: 'demo',  
viewConfig: {  
    stripeRows: true  
}  
})  
})
```

Mydata.json

```
{  
  "items": [  
    {  
      "UserName": "李彦宏",  
      "Sex": "男",  
      "Age": 25,  
      "XueHao": 00001,  
      "BanJi": "一班"  
    },  
    {  
      "UserName": "马云",  
      "Sex": "男",  
      "Age": 31,  
      "XueHao": 00002,  
      "BanJi": "二班"  
    },  
    {  
      "UserName": "张朝阳",  
      "Sex": "男",  
      "Age": 30,  
      "XueHao": 00003,  
      "BanJi": "一班"  
    },  
    {  
      "UserName": "朱俊",
```

```
        "Sex": "男",
        "Age": 28,
        "XueHao": 00004,
        "BanJi": "一班"
    },
    {
        "UserName": "丁磊",
        "Sex": "男",
        "Age": 29,
        "XueHao": 00005,
        "BanJi": "二班"
    },
    {
        "UserName": "李国军",
        "Sex": "男",
        "Age": 30,
        "XueHao": 00006,
        "BanJi": "二班"
    },
    {
        "UserName": "王志宝",
        "Sex": "男",
        "Age": 25,
        "XueHao": 00007,
        "BanJi": "一班"
    }
]
}
```

这里需要注意的是 json 文件的编码方式必须要跟 html 文件的编码方式一致，如本例中 html 的编码方式为 utf-8，则 json 必须也要是 utf-8 格式，否则页面数据会出现乱码。Json 仅仅是个文本文件而已，转码方式有很多。

Extjs4 Grid 组件的数据需要几点注意。

第一，就是数据类型，我们可以指定数据类型，比如：

```
var store = Ext.create('Ext.data.ArrayStore', {
    fields: [
        {name: 'company'},
        {name: 'price',      type: 'float'},
        {name: 'change',     type: 'float'},
        {name: 'pctChange',  type: 'float'},
        {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
    ],
    data: myData
});
```

数据类型分为以下几种：

- 1、auto（默认）
- 2、string
- 3、int
- 4、float
- 5、boolean
- 6、date

第二：Ext.data.Model，示例中，只指定了一个 mapping，那么第一个 mapping 是必须要指定的，从第二个开始，我们就不需要再去指定 mapping 了，Extjs 很聪明，他会根据数据来判断需要的 mapping。

运行结果：

ExtJS4 Grid示例				
姓名	性别	年龄	学号	班级
李彦宏	男	25	1	一班
马云	男	31	2	二班
张朝阳	男	30	3	一班
朱俊	男	28	4	一班
丁磊	男	29	5	二班
李国军	男	30	6	二班
王志宝	男	25	7	一班

## 4.5 Grid 分页

Grid 组件，当数据量很大的时候，就需要分页显示，本文介绍如何实现 Extjs4 Grid 的分页功能。

先看 THML 代码：

```
1. <!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
   11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>Paging Grid-MHZG.NET</title>
6. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
   all.css" />
7. <script type="text/javascript" src="../../bootstrap.js"></script>
8. <script type="text/javascript" src="../../locale/ext-lang-
   zh_CN.js"></script>
9. <script type="text/javascript" src="paing.js"></script>
10. </head>
11.
12. <body>
13. <div id="demo"></div>
14. </body>
15. </html>
```

这里引用的 JS 文件，都是相对于 Extjs4 整个目录的。如果已经将 JS 和 CSS 文件剥离并分目录放置了，那么一定要注意修改 bootstrap.js。

JS 代码：

```
Ext.require([

    'Ext.grid.*',

    'Ext.toolbar.Paging',

    'Ext.data.*'

]);
```

```
Ext.onReady(function(){

    Ext.define('MyData',{

        extend: 'Ext.data.Model',

        fields: [

            'title','author',

            //第一个字段需要指定 mapping，其他字段，可以省略掉。

            {name:'hits',type: 'int'},

            'addtime'

        ]

    });
```

```
});

//创建数据源
var store = Ext.create('Ext.data.Store', {
    //分页大小
    pageSize: 50,
    model: 'MyData',
    //是否在服务端排序
    remoteSort: true,
    proxy: {
        //异步获取数据，这里的URL可以改为任何动态页面，只要返回JSON
        //数据即可
        type: 'ajax',
        url: 'mydata.asp',

        reader: {
            root: 'items',
            totalProperty : 'total'
        },
        simpleSortMode: true
    },
    sorters: [{
        //排序字段。
        property: 'hits',
        //排序类型，默认为 ASC
        direction: 'DESC'
    }]
});
```

```
//创建 Grid

var grid = Ext.create('Ext.grid.Panel',{
    store: store,
    columns: [
        {text: "标题", width: 120, dataIndex: 'title', sortable: true},
        {text: "作者", flex: 200, dataIndex: 'author', sortable: false},
        {text: "点击数", width: 100, dataIndex: 'hits', sortable: true},
        {text: "添加时间", width: 100, dataIndex: 'addtime', sortable: true}
    ],
    height:400,
    width:520,
    x:20,
    y:40,
    title: 'ExtJS4 Grid 分页示例',
    disableSelection: true,
    loadMask: true,
    renderTo: 'demo',
    viewConfig: {
        id: 'gv',
        trackOver: false,
        stripeRows: false
    },

    bbar: Ext.create('Ext.PagingToolbar', {
        store: store,
        displayInfo: true,
```

```

        displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',

        emptyMsg: "没有数据"

    })

})

store.loadPage(1);

})

```

关于数据，任何服务端都可以，只要返回相应的数据就可以了。这里简单使用 ASP 代码做了一些测试用的数据，但是这些测试代码包含了参数接收、基本判断以及分页方法。  
具体情况具体实施，这些代码只作为抛砖引玉的作用。

ASP 代码：

```

1.  <%
2.      Response.ContentType = "text/html"
3.      Response.Charset = "UTF-8"
4.  %>
5.  <%
6.  '返回 JSON 数据, 自定义一些测试数据。。
7.  '这里的参数与 EXT3.x 相同, 区别在于排序字段和排序方式使用了新的属性。
8.  '由于这里是测试数据, 接收的参数只用到了 start, limit。sorts 和 dir 在实际操作过程中,
    将之加入 SQL 的 ORDER BY 里即可。
9.  start = Request("start")
10. limit = Request("limit")
11. If start = "" Then
12.     start = 0
13. End If
14. If limit = "" Then
15.     limit = 50
16. End If
17. sorts = Replace(Trim(Request.Form("sort")), "", "")
18. dir = Replace(Trim(Request.Form("dir")), "", "")
19.
20. Dim counts:counts=300
21. '注意, 这里的 counts 相当于 Rs.RecordCount, 也就是记录总数。
22.
23. Dim Ls:Ls = Cint(start) + Cint(limit)
24. If Ls >= counts Then
25.     Ls = counts
26. End If
27.
28. Echo("{")
29. Echo("total":)
30. Echo(""&counts&"",")
31. Echo("items":[")
32. For i = start+1 To Ls
33.     Echo("{")
34.     Echo("title":"newstitle"&i&"")
35.     Echo(",")
36.     Echo("author":"author"&i&"")
37.     Echo(",")
38.     Echo("hits":"","&i&"")
39.     Echo(",")
40.     Echo("addtime":"","&Now()&"")
41.     Echo("}")
42.     If i<Ls Then
43.         Echo(",")
44.     End If

```

```

45. Next
46. Echo("{}")
47. Function Echo(str)
48.     Response.Write(str)
49. End Function
50. %>

```

最后，来张效果图：

标题	作者	点击数 ▼	添加时间
newstitle51	author51	51	2011/5/18 11:03...
newstitle52	author52	52	2011/5/18 11:03...
newstitle53	author53	53	2011/5/18 11:03...
newstitle54	author54	54	2011/5/18 11:03...
newstitle55	author55	55	2011/5/18 11:03...
newstitle56	author56	56	2011/5/18 11:03...
newstitle57	author57	57	2011/5/18 11:03...
newstitle58	author58	58	2011/5/18 11:03...
newstitle59	author59	59	2011/5/18 11:03...
newstitle60	author60	60	2011/5/18 11:03...
newstitle61	author61	61	2011/5/18 11:03...
newstitle62	author62	62	2011/5/18 11:03...
newstitle63	author63	63	2011/5/18 11:03...
newstitle64	author64	64	2011/5/18 11:03...
newstitle65	author65	65	2011/5/18 11:03...

第 2 页,共 6 页 显示 51 - 100 条, 共计 300 条

## 4.6 多表头 grid

做项目的时候，有时候会遇到多表头的 Grid，在 EXTJS4 中，多表头的实现已经很简单了，本文介绍如何实现多表头 grid 的功能。



先看下效果图：

ExtJS4 多表头Grid带分页示例			
基本信息			添加时间
标题	作者	点击数 ▼	
newstitle1	author1	1	2011/5/19 11:2
newstitle2	author2	2	2011/5/19 11:2
newstitle3	author3	3	2011/5/19 11:2
newstitle4	author4	4	2011/5/19 11:2
newstitle5	author5	5	2011/5/19 11:2
newstitle6	author6	6	2011/5/19 11:2
newstitle7	author7	7	2011/5/19 11:2
newstitle8	author8	8	2011/5/19 11:2
newstitle9	author9	9	2011/5/19 11:2
newstitle10	author10	10	2011/5/19 11:2
newstitle11	author11	11	2011/5/19 11:2
newstitle12	author12	12	2011/5/19 11:2
newstitle13	author13	13	2011/5/19 11:2

第 1 页,共 6 页 显示 1 - 50 条, 共计 300 条

实现代码如下：

HTML 代码：

```
1. <!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>GroupHeaderGrid-MHZG.NET</title>
6. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
all.css" />
7. <script type="text/javascript" src="../../bootstrap.js"></script>
8. <script type="text/javascript" src="../../locale/ext-lang-
zh_CN.js"></script>
9. <script type="text/javascript" src="group-header.js"></script>
10. </head>
11.
12. <body>
13. <div id="demo"></div>
14. </body>
15. </html>
```

group-header.js：

```
1. Ext.require([
2.     'Ext.grid.*',
```

```

3.     'Ext.toolbar.Paging',
4.     'Ext.util.*',
5.     'Ext.data.*'
6.   ]});
7.
8.   Ext.onReady(function(){
9.       Ext.define('MyData',{
10.           extend: 'Ext.data.Model',
11.           fields: [
12.               'title','author',
13.               //第一个字段需要指定 mapping , 其他字段 , 可以省略掉。
14.               {name:'hits',type: 'int'},
15.               'addtime'
16.           ]
17.       });
18.
19.       //创建数据源
20.       var store = Ext.create('Ext.data.Store', {
21.           //分页大小
22.           pageSize: 50,
23.           model: 'MyData',
24.           //是否在服务端排序
25.           remoteSort: true,
26.           proxy: {
27.               //异步获取数据, 这里的 URL 可以改为任何动态页面, 只要返回 JSON 数据即可
28.               type: 'ajax',
29.               url: 'mydata.asp',
30.
31.               reader: {
32.                   root: 'items',
33.                   totalProperty : 'total'
34.               },
35.               simpleSortMode: true
36.           },
37.           sorters: [{
38.               //排序字段。
39.               property: 'hits',
40.               //排序类型, 默认为 ASC
41.               direction: 'DESC'
42.           }]
43.       });
44.
45.       //创建 Grid
46.       var grid = Ext.create('Ext.grid.Panel',{
47.           store: store,
48.           columnLines: true,
49.           columns: [{
50.               text:"基本信息",
51.               columns:[
52.                   {text: "标题",
53.                   ", width: 120, dataIndex: 'title', sortable: true},
54.                   {text: "作者",
55.                   ", width: 200, dataIndex: 'author', sortable: false},
56.                   {text: "点击数",
57.                   ", width: 100, dataIndex: 'hits', sortable: true}]
58.               },
59.               {text: "添加时间",
60.               ", width: 100, dataIndex: 'addtime', sortable: true}
61.           ],
62.           height:400,

```

```

59.         width:520,
60.         x:20,
61.         y:40,
62.         title: 'ExtJS4 多表头 Grid 带分页示例',
63.         disableSelection: true,
64.         loadMask: true,
65.         renderTo: 'demo',
66.         viewConfig: {
67.             id: 'gv',
68.             trackOver: false,
69.             stripeRows: false
70.         },
71.
72.         bbar: Ext.create('Ext.PagingToolbar', {
73.             store: store,
74.             displayInfo: true,
75.             displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',
76.             emptyMsg: "没有数据"
77.         })
78.     })
79.     store.loadPage(1);
80. })

```

JS 代码要注意的地方：

- 1、记得载入 'Ext.util.\*'，
- 2、二级表头必须指定宽度，如果不指定的话，会提示错误。如图所示，红框里的项，必须要指定宽度。

```

columns: [{
    text: "基本信息",
    columns: [
        {text: "标题", width: 120, dataIndex: 'title', sortable: true},
        {text: "作者", width: 200, dataIndex: 'author', sortable: false},
        {text: "点击数", width: 100, dataIndex: 'hits', sortable: true}
    ],
    {text: "添加时间", width: 100, dataIndex: 'addtime', sortable: true}
],

```

服务端代码，这里使用 ASP 编写，具体解释请参考前一篇文章。

```

1.  <%
2.      Response.ContentType = "text/html"
3.      Response.Charset = "UTF-8"
4.  %>
5.  <%
6.      '返回 JSON 数据,自定义一些测试数据。。
7.      '这里的参数与 EXT3.x 相同，区别在于排序字段和排序方式使用了新的属性。
8.      '由于这里是测试数据，接收的参数只用到了 start,limit。sorts 和 dir 在实际操作过程中，
        将之加入 SQL 的 ORDER BY 里即可。
9.      start = Request("start")
10.     limit = Request("limit")
11.     If start = "" Then
12.         start = 0
13.     End If
14.     If limit = "" Then
15.         limit = 50

```

```

16. End If
17. sorts = Replace(Trim(Request.Form("sort")), "','', "'")
18. dir = Replace(Trim(Request.Form("dir")), "','', "'")
19.
20. Dim counts:counts=300
21. '注意, 这里的 counts 相当于 Rs.RecordCount, 也就是记录总数。
22.
23. Dim Ls:Ls = Cint(start) + Cint(limit)
24. If Ls >= counts Then
25.     Ls = counts
26. End If
27.
28. Echo("{")
29. Echo("total:")
30. Echo("&counts&",")
31. Echo("items:")
32. For i = start+1 To Ls
33.     Echo("{")
34.     Echo("title":"newstitle"&i&")
35.     Echo(",")
36.     Echo("author":"author"&i&")
37.     Echo(",")
38.     Echo("hits":"&i&")
39.     Echo(",")
40.     Echo("addtime":"&Now()&")
41.     Echo("}")
42. If i<Ls Then
43.     Echo(",")
44. End If
45. Next
46. Echo("}")
47. Function Echo(str)
48.     Response.Write(str)
49. End Function
50. %>

```

## 4.7 带搜索的 grid

项目开发中, Grid 组件少不了搜索功能, 在 Extjs4 中, 搜索组件以插件的形式出现, 而且实现也非常简单, 搜索组件位于 examples/ux/form 目录下, JS 文件是 SearchField.js。

Grid 加载搜索功能, 要注意的是:

- 1、开启延迟加载, 即 Ext.Loader.setConfig({enabled: true});
- 2、设置插件的目录: Ext.Loader.setPath('Ext.ux',  
'../..../examples/ux');, 需要注意, 插件所在目录一定要正确, 否则加载失败, 就实现不了所要功能了。

效果图:

ExtJS4 SearchGrid-Grid 搜索			
搜索: <input type="text"/>			
标题	作者	点击数 ▼	添加时间
newstitle1	author1	1	2011/5/21 9:18:25
newstitle2	author2	2	2011/5/21 9:18:25
newstitle3	author3	3	2011/5/21 9:18:25
newstitle4	author4	4	2011/5/21 9:18:25
newstitle5	author5	5	2011/5/21 9:18:25
newstitle6	author6	6	2011/5/21 9:18:25
newstitle7	author7	7	2011/5/21 9:18:25
newstitle8	author8	8	2011/5/21 9:18:25
newstitle9	author9	9	2011/5/21 9:18:25
newstitle10	author10	10	2011/5/21 9:18:25
newstitle11	author11	11	2011/5/21 9:18:25
newstitle12	author12	12	2011/5/21 9:18:25
newstitle13	author13	13	2011/5/21 9:18:25
newstitle14	author14	14	2011/5/21 9:18:25

第 1 页,共 6 页 显示 1 - 50 条, 共计 300 条

初始加载

ExtJS4 SearchGrid-Grid 搜索			
搜索: newstitle			
标题	作者	点击数 ▼	添加时间
newstitle	author	100	2011/5/21 9:20:23

第 1 页,共 1 页 显示 1 - 1 条, 共计 1 条

输入查询条件后

HTML 代码:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>SearchGrid-MHZG.NET</title>
<link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
<style type="text/css">
    #search-results a {
        color: #385F95;
        font:bold 11px tahoma, arial, helvetica, sans-serif;
        text-decoration:none;
    }

    #search-results a:hover {
        text-decoration:underline;
    }

    #search-results p {
        margin:3px !important;
    }

    .search-item {
        font:normal 11px tahoma, arial, helvetica, sans-serif;
        padding:3px 10px 3px 10px;
        border:1px solid #fff;
        border-bottom:1px solid #eeeeee;
        white-space:normal;
        color:#555;
    }

    .search-item h3 {
        display:block;
        font:inherit;
        font-weight:bold;
        color:#222;
    }

    .search-item h3 span {
        float: right;
        font-weight:normal;
        margin:0 0 5px 5px;
        width:100px;
        display:block;
        clear:none;
    }

    /*这里要注意这两张图片的路径要正确*/
    .x-form-clear-trigger {
        background-image: url(../../resources/themes/images/default/form/clear-
trigger.gif);
    }

    .x-form-search-trigger {
        background-image:
url(../../resources/themes/images/default/form/search-trigger.gif);
    }
</style>
<script type="text/javascript" src="../../bootstrap.js"></script>
<script type="text/javascript" src="../../locale/ext-lang-zh_CN.js"></script>
<script type="text/javascript" src="searchgrid.js"></script>
</head>

<body>
<div id="demo"></div>
</body>
</html>

```

## SearchGrid.js:

```
1. Ext.Loader.setConfig({enabled: true});
2. Ext.Loader.setPath('Ext.ux', '../..//examples/ux');
3. Ext.require([
4.     'Ext.grid.*',
5.     'Ext.toolbar.Paging',
6.     'Ext.util.*',
7.     'Ext.data.*',
8.     'Ext.ux.form.SearchField'
9. ]);
10.
11. Ext.onReady(function(){
12.     Ext.define('MyData',{
13.         extend: 'Ext.data.Model',
14.         fields: [
15.             'title','author',
16.             //第一个字段需要指定 mapping, 其他字段, 可以省略掉。
17.             {name:'hits',type: 'int'},
18.             'addtime'
19.         ]
20.     });
21.
22.     //创建数据源
23.     var store = Ext.create('Ext.data.Store', {
24.         //分页大小
25.         pageSize: 50,
26.         model: 'MyData',
27.         //是否在服务端排序
28.         remoteSort: true,
29.         proxy: {
30.             //异步获取数据, 这里的 URL 可以改为任何动态页面, 只要返回 JSON 数据即可
31.             type: 'ajax',
32.             url: 'searchgrid.asp',
33.
34.             reader: {
35.                 root: 'items',
36.                 totalProperty : 'total'
37.             },
38.             simpleSortMode: true
39.         },
40.         sorters: [{
41.             //排序字段。
42.             property: 'hits',
43.             //排序类型, 默认为 ASC
44.             direction: 'DESC'
45.         }]
46.     });
47.
48.     //创建 Grid
49.     var grid = Ext.create('Ext.grid.Panel',{
50.         store: store,
51.         columnLines: true,
52.         columns: [
53.             {text: "标题", width: 120, dataIndex: 'title', sortable: true},
54.             {text: "作者", width: 140, dataIndex: 'author', sortable: false},
55.             {text: "点击数", width: 100, dataIndex: 'hits', sortable: true},
56.             {text: "添加时间", width: 150, dataIndex: 'addtime', sortable: true}
57.         ],
58.         height:400,
```

```

59.         width:520,
60.         x:20,
61.         y:40,
62.         title: 'ExtJS4 SearchGrid-Grid 搜索',
63.         disableSelection: true,
64.         loadMask: true,
65.         renderTo: 'demo',
66.         viewConfig: {
67.             id: 'gv',
68.             trackOver: false,
69.             stripeRows: false
70.         },
71.         dockedItems: [{
72.             dock: 'top',
73.             xtype: 'toolbar',
74.             items: {
75.                 width: 300,
76.                 fieldLabel: '搜索',
77.                 labelWidth: 50,
78.                 xtype: 'searchfield',
79.                 store: store
80.             }
81.         }, {
82.             dock: 'bottom',
83.             xtype: 'pagingtoolbar',
84.             store: store,
85.             pageSize: 25,
86.             displayInfo: true,
87.             displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',
88.             emptyMsg: '没有数据'
89.         }]
90.     })
91.     store.loadPage(1);
92. })
93. })

```

代码完成了 Grid 组件异步加载信息、分页和搜索功能，可以满足一般使用情况了。

服务端代码，由于使用测试数据，这里只使用了最简单的方法来实现搜索效果，实际操作中，需要将查询条件置于 SQL 语句中，达到搜索效果。

SearchGrid.asp:

```

1.  <%
2.      Response.ContentType = "text/html"
3.      Response.Charset = "UTF-8"
4.  %>
5.  <%
6.      '返回 JSON 数据,自定义一些测试数据。。
7.      '这里的参数与 EXT3.x 相同,区别在于排序字段和排序方式使用了新的属性。
8.      '由于这里是测试数据,接收的参数只用到了 start,limit.sorts 和 dir 在实际操作过程中,
      将之加入 SQL 的 ORDER BY 里即可。
9.      start = Request("start")
10.     limit = Request("limit")
11.     '查询时获取的参数。
12.     query = Request("query")
13.     If start = "" Then
14.         start = 0
15.     End If
16.     If limit = "" Then
17.         limit = 50
18.     End If
19.     sorts = Replace(Trim(Request.Form("sort")), "','','")
20.     dir = Replace(Trim(Request.Form("dir")), "','','")
21.
22.     '测试数据,这里直接输出结果,实际应用中,应该把查询条件放到 SQL 语句中。

```



```

23. If query = "newstitle" Then
24.     Echo("{")
25.     Echo("total:")
26.     Echo("1")
27.     Echo("","items":[")
28.     Echo("{")
29.     Echo("title":"newstitle")
30.     Echo(",")
31.     Echo("author":"author")
32.     Echo(",")
33.     Echo("hits":"100")
34.     Echo(",")
35.     Echo("addtime":"&Now()&")
36.     Echo("}")
37.     Echo("}]")
38. Else
39.     Dim counts:counts=300
40.     '注意，这里的 counts 相当于 Rs.RecordCount, 也就是记录总数。
41.
42.     Dim Ls:Ls = Cint(start) + Cint(limit)
43.     If Ls >= counts Then
44.         Ls = counts
45.     End If
46.
47.     Echo("{")
48.     Echo("total:")
49.     Echo(""&counts&"",")
50.     Echo("items":[")
51.     For i = start+1 To Ls
52.         Echo("{")
53.         Echo("title":"newstitle"&i&")
54.         Echo(",")
55.         Echo("author":"author"&i&")
56.         Echo(",")
57.         Echo("hits":"&i&")
58.         Echo(",")
59.         Echo("addtime":"&Now()&")
60.         Echo("}")
61.         If i<Ls Then
62.             Echo(",")
63.         End If
64.     Next
65.
66.     Echo("}]")
67. End If
68. Function Echo(str)
69.     Response.Write(str)
70. End Function
71. %>

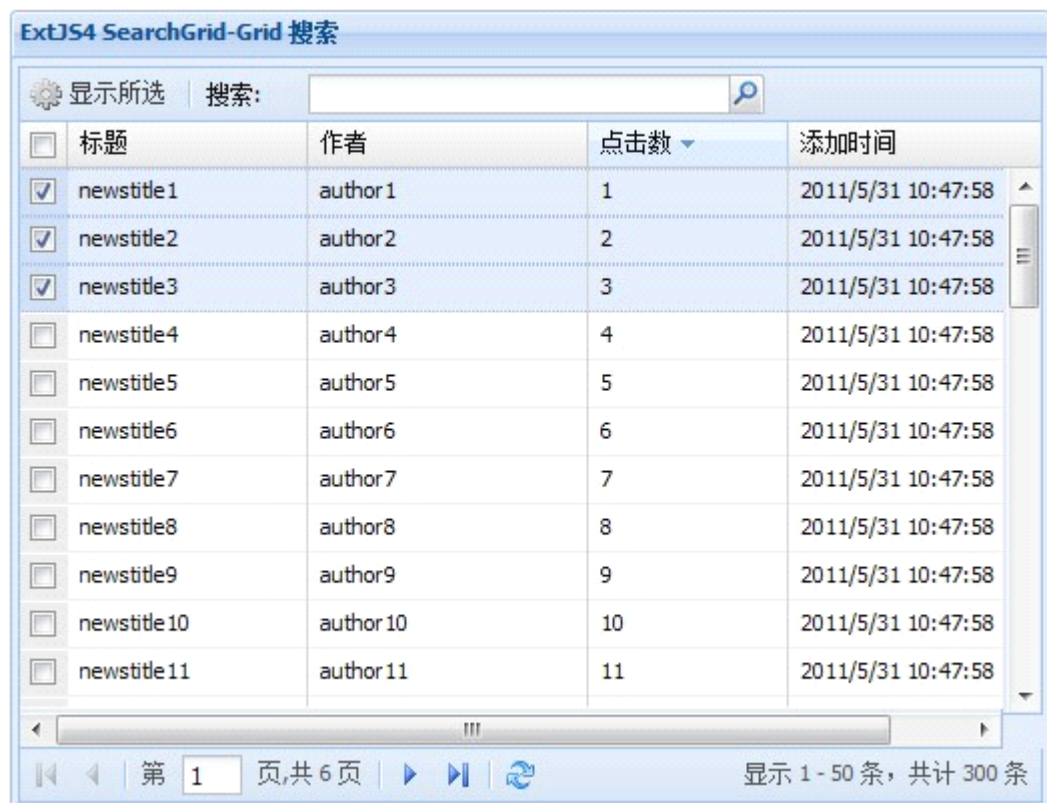
```

至此，带搜索功能的 Grid 就全部完成了。

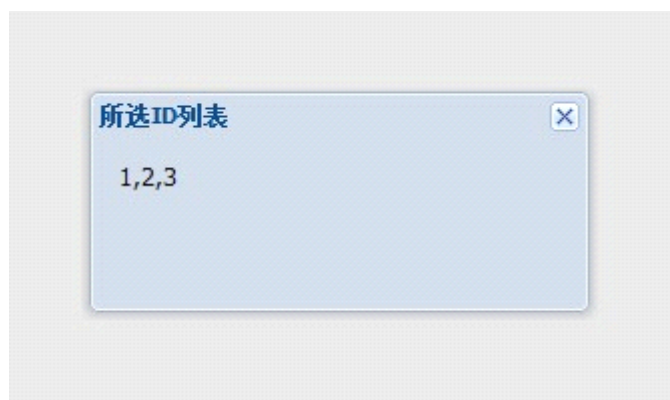
## 4.8 Grid 多选/全选

但大量数据需要删除操作的时候，总不能一条一条去删除吧，本文介绍如何在 Extjs4 Grid 中加入全选功能。

先来张效果图：



点击显示所选后：



注意点：

- 1、需要在 JS 文件中动态加载“'Ext.selection.CheckboxModel'”
- 2、服务端要返回数据的 id 值。
- 3、Ext.data.Model 中，要有 id 的信息，就是因为 JS 代码中忘记了写 id，导致调试了很久都无法获取 id 值，从头检查代码的时候才发现错误。

正文：

html 代码：

```

1. <!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
   11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>Grid 多选/全选-MHZG.NET</title>
6. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
   all.css" />
7. <style type="text/css">
8.     #search-results a {
9.         color: #385F95;
10.        font:bold 11px tahoma, arial, helvetica, sans-serif;
11.        text-decoration:none;
12.    }
13.    .add {
14.        background-
15.        image:url(../../examples/shared/icons/fam/cog.gif) !important;
16.    }
17.    #search-results a:hover {
18.        text-decoration:underline;
19.    }
20.    #search-results p {
21.        margin:3px !important;
22.    }
23.
24.    .search-item {
25.        font:normal 11px tahoma, arial, helvetica, sans-serif;
26.        padding:3px 10px 3px 10px;
27.        border:1px solid #fff;
28.        border-bottom:1px solid #eeeeee;
29.        white-space:normal;
30.        color:#555;
31.    }
32.    .search-item h3 {
33.        display:block;
34.        font:inherit;
35.        font-weight:bold;
36.        color:#222;
37.    }
38.
39.    .search-item h3 span {
40.        float: right;
41.        font-weight:normal;
42.        margin:0 0 5px 5px;
43.        width:100px;
44.        display:block;
45.        clear:none;
46.    }
47.
48.    .x-form-clear-trigger {
49.        background-
50.        image: url(../../resources/themes/images/default/form/clear-trigger.gif);
51.    }
52.
53.    .x-form-search-trigger {
54.        background-
55.        image: url(../../resources/themes/images/default/form/search-trigger.gif);
56.    }
57. </style>
58. <script type="text/javascript" src="../../bootstrap.js"></script>

```

```

57. <script type="text/javascript" src="../../locale/ext-lang-
    zh_CN.js"></script>
58. <script type="text/javascript" src="selectgrid.js"></script>
59. </head>
60.
61. <body>
62. <div id="demo"></div>
63. </body>
64. </html>

```

selectgrid.js :

```

1. Ext.Loader.setConfig({enabled: true});
2. Ext.Loader.setPath('Ext.ux', '../../examples/ux');
3. Ext.require([
4.     'Ext.grid.*',
5.     'Ext.toolbar.Paging',
6.     'Ext.util.*',
7.     'Ext.data.*',
8.     'Ext.ux.form.SearchField',
9.     'Ext.selection.CheckboxModel'
10. ]);
11.
12. Ext.onReady(function(){
13.     Ext.define('MyData',{
14.         extend: 'Ext.data.Model',
15.         fields: [
16.             'id','title','author',
17.             {name:'hits',type: 'int'},
18.             'addtime'
19.         ]
20.     });
21.
22.     //创建数据源
23.     var store = Ext.create('Ext.data.Store', {
24.         //分页大小
25.         pageSize: 50,
26.         model: 'MyData',
27.         //是否在服务端排序
28.         remoteSort: true,
29.         proxy: {
30.             //异步获取数据，这里的 URL 可以改为任何动态页面，只要返回 JSON 数据即可
31.             type: 'ajax',
32.             url: 'selectgrid.asp',
33.
34.             reader: {
35.                 root: 'items',
36.                 totalProperty : 'total'
37.             },
38.             simpleSortMode: true
39.         },
40.         sorters: [{
41.             //排序字段。
42.             property: 'hits',
43.             //排序类型，默认为 ASC
44.             direction: 'DESC'
45.         }]

```

```

46.     });
47.
48.     //创建多选
49.     var selModel = Ext.create('Ext.selection.CheckboxModel');
50.     //创建 Grid
51.     var grid = Ext.create('Ext.grid.Panel',{
52.         store: store,
53.         selModel: selModel,
54.         columnLines: true,
55.         columns: [
56.             {text: "标题",
57.             ", width: 120, dataIndex: 'title', sortable: true},
58.             {text: "作者",
59.             ", width: 140, dataIndex: 'author', sortable: false},
60.             {text: "点击数",
61.             ", width: 100, dataIndex: 'hits', sortable: true},
62.             {text: "添加时间",
63.             ", width: 150, dataIndex: 'addtime', sortable: true}
64.         ],
65.         height:400,
66.         width:520,
67.         x:20,
68.         y:40,
69.         title: 'ExtJS4 SearchGrid-Grid 搜索',
70.
71.         disableSelection: false, //值为 TRUE , 表示禁止选择行
72.         frame: true,
73.         loadMask: true,
74.         renderTo: 'demo',
75.         viewConfig: {
76.             id: 'gv',
77.             trackOver: false,
78.             stripeRows: false
79.         },
80.         dockedItems: [{
81.             dock: 'top',
82.             xtype: 'toolbar',
83.             items: [{
84.                 itemId: 'Button',
85.                 text: '显示所选',
86.                 tooltip: 'Add a new row',
87.                 iconCls: 'add',
88.                 handler: function(){
89.                     var record = grid.getSelectionModel().getSelection();
90.                     if(record.length == 0){
91.                         Ext.MessageBox.show({
92.                             title: "提示",
93.                             msg: "请先选择您要操作的行!",
94.                             //icon: Ext.MessageBox.INFO
95.                         })
96.                     }
97.                     return;
98.                 }else{
99.                     var ids = "";
100.                     for(var i = 0; i < record.length; i++){
101.                         ids += record[i].get("id")
102.                         if(i<record.length-1){
103.                             ids = ids + ",";
104.                         }
105.                     }
106.                 }
107.             }
108.         ]
109.     });
110.     Ext.MessageBox.show({

```

```

102.                                     title:"所选 ID 列表",
103.                                     msg:ids
104.                                     //icon: Ext.MessageBox.INFO
105.                                 })
106.                            }
107.                    }
108.            }, '-', {
109.                width: 300,
110.                fieldLabel: '搜索',
111.                labelWidth: 50,
112.                xtype: 'searchfield',
113.                store: store
114.            }]
115.        }, {
116.            dock: 'bottom',
117.            xtype: 'pagingtoolbar',
118.            store: store,
119.            pageSize: 25,
120.            displayInfo: true,
121.            displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',
122.            emptyMsg: '没有数据'
123.        }]
124.
125.    })
126.    store.loadPage(1);
127.})

```

服务端 selectgrid.asp :

```

1.  <%
2.      Response.ContentType = "text/html"
3.      Response.Charset = "UTF-8"
4.  %>
5.  <%
6.      '返回 JSON 数据,自定义一些测试数据。。
7.      '这里的参数与 EXT3.x 相同,区别在于排序字段和排序方式使用了新的属性。
8.      '由于这里是测试数据,接收的参数只用到了 start,limit。sorts 和 dir 在实际操作过程中,
        将之加入 SQL 的 ORDER BY 里即可。
9.      start = Request("start")
10.     limit = Request("limit")
11.     '查询时获取的参数。
12.     query = Request("query")
13.     If start = "" Then
14.         start = 0
15.     End If
16.     If limit = "" Then
17.         limit = 50
18.     End If
19.     sorts = Replace(Trim(Request.Form("sort")), "','','")
20.     dir = Replace(Trim(Request.Form("dir")), "','','")
21.
22.     '测试数据,这里直接输出结果,实际应用中,应该把查询条件放到 SQL 语句中。
23.     If query = "newstitle" Then
24.         Echo("{")
25.         Echo("total:")
26.         Echo("1")
27.         Echo("","items":["")

```

```

28.     Echo("{")
29.     Echo("title":"newstyle")
30.     Echo(",")
31.     Echo("author":"author")
32.     Echo(",")
33.     Echo("hits":"100")
34.     Echo(",")
35.     Echo("addtime":"&Now()&")
36.     Echo("}")
37.     Echo("]")
38. Else
39. Dim counts:counts=300
40. '注意，这里的 counts 相当于 Rs.RecordCount,也就是记录总数。
41.
42. Dim Ls:Ls = Cint(start) + Cint(limit)
43. If Ls >= counts Then
44.     Ls = counts
45. End If
46.
47. Echo("{")
48. Echo("total:")
49. Echo("&counts&","")
50. Echo("items:")
51. For i = start+1 To Ls
52.     Echo("{")
53.     Echo("id":"&i&")
54.     Echo(",")
55.     Echo("title":"newstyle&i&")
56.     Echo(",")
57.     Echo("author":"author&i&")
58.     Echo(",")
59.     Echo("hits":"&i&")
60.     Echo(",")
61.     Echo("addtime":"&Now()&")
62.     Echo("}")
63.     If i<Ls Then
64.         Echo(",")
65.     End If
66. Next
67.
68. Echo("]")
69. End If
70. Function Echo(str)
71.     Response.Write(str)
72. End Function
73. %>

```

## 4.9 可编辑的 grid

实际运用中，我们可能需要动态修改 Grid 中的数据，也就是要实现 EditGrid 功能。本文介绍如何实现 Extjs4 EditGrid 功能。先发几张效果图。看图说话，更有说服力哦。

<div>  显示所选         <div>           搜索:           <input type="text"/> </div> </div>					
<input type="checkbox"/>	标题	作者	点击数 ▼	添加时间	审核
<input checked="" type="checkbox"/>	newstitle1	管理员	5	2011-06-08	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle2	author2	2	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle3	author3	3	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle4	author4	4	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle5	author5	5	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle6	author6	6	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle7	author7	7	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle8	author8	8	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle9	author9	9	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle10	author10	10	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle11	author11	11	2011-06-09	<input type="checkbox"/>
<div> <div>           第 1 页,共 1 页         </div> <div> </div> </div> <div>显示 1 - 20 条, 共计 20 条</div>					

<div>  显示所选         <div>           搜索:           <input type="text"/> </div> </div>					
<input type="checkbox"/>	标题	作者	点击数 ▼	添加时间	审核
<input checked="" type="checkbox"/>	newstitle1	<div><input type="text"/></div>	5	2011-06-08	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle2	佚名	2	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle3	管理员	3	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle4	编辑	4	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle5	总编辑	5	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle6	测试员	6	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle7	author7	7	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle8	author8	8	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle9	author9	9	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle10	author10	10	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle11	author11	11	2011-06-09	<input type="checkbox"/>
<div> <div>           第 1 页,共 1 页         </div> <div> </div> </div> <div>显示 1 - 20 条, 共计 20 条</div>					



ExtJS4 EditGrid(可编辑的Grid)

显示所选 搜索:

<input type="checkbox"/>	标题	作者	点击数	添加时间	审核
<input checked="" type="checkbox"/>	newstitle1	author1	5	2011-06-08	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle2	author2	2	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle3	author3	3	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle4	author4	4	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle5	author5	5	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle6	author6	6	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle7	author7	7	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle8	author8	8	2011-06-09	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle9	author9	9	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle10	author10	10	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle11	author11	11	2011-06-09	<input type="checkbox"/>

第 1 页,共 1 页 显示 1 - 20 条, 共计 20 条

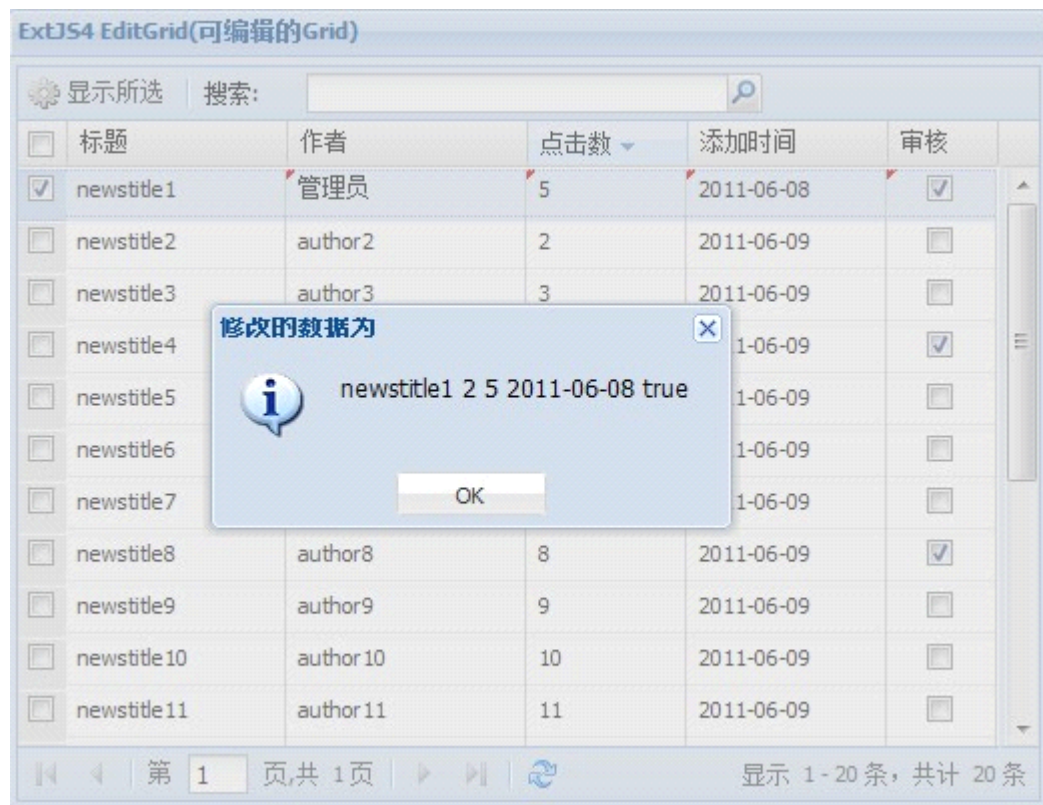
ExtJS4 EditGrid(可编辑的Grid)

显示所选 搜索:

<input type="checkbox"/>	标题	作者	点击数	添加时间	审核
<input checked="" type="checkbox"/>	newstitle1	管理员	5	11-06-08	<input checked="" type="checkbox"/>
<input type="checkbox"/>	newstitle2	author2	2		
<input type="checkbox"/>	newstitle3	author3	3		
<input type="checkbox"/>	newstitle4	author4	4		
<input type="checkbox"/>	newstitle5	author5	5		
<input type="checkbox"/>	newstitle6	author6	6		
<input type="checkbox"/>	newstitle7	author7	7		
<input type="checkbox"/>	newstitle8	author8	8		
<input type="checkbox"/>	newstitle9	author9	9		
<input type="checkbox"/>	newstitle10	author10	10	2011-06-09	<input type="checkbox"/>
<input type="checkbox"/>	newstitle11	author11	11	2011-06-09	<input type="checkbox"/>

第 1 页,共 1 页 显示 1 - 20 条, 共计 20 条

六月 2011  
今天  
Date Picker 六月



HTML 代码：

```

1. <!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
   11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>ExtJS4 EditGrid(可编辑的 Grid)-MHZG.NET</title>
6. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
   all.css" />
7. <link rel="stylesheet" type="text/css" href="../../examples/ux/css/CheckHead
   er.css" />
8. <style type="text/css">
9.     #search-results a {
10.         color: #385F95;
11.         font:bold 11px tahoma, arial, helvetica, sans-serif;
12.         text-decoration:none;
13.     }
14.     .add {
15.         background-
16.         image:url(../../examples/shared/icons/fam/cog.gif) !important;
17.     }
18.     #search-results a:hover {
19.         text-decoration:underline;
20.     }
21.     #search-results p {
22.         margin:3px !important;
23.     }

```

```

24.
25.     .search-item {
26.         font:normal 11px tahoma, arial, helvetica, sans-serif;
27.         padding:3px 10px 3px 10px;
28.         border:1px solid #fff;
29.         border-bottom:1px solid #eeeeee;
30.         white-space:normal;
31.         color:#555;
32.     }
33.     .search-item h3 {
34.         display:block;
35.         font:inherit;
36.         font-weight:bold;
37.         color:#222;
38.     }
39.
40.     .search-item h3 span {
41.         float: right;
42.         font-weight:normal;
43.         margin:0 0 5px 5px;
44.         width:100px;
45.         display:block;
46.         clear:none;
47.     }
48.
49.     .x-form-clear-trigger {
50.         background-
51.         image: url(../../resources/themes/images/default/form/clear-trigger.gif);
52.     }
53.     .x-form-search-trigger {
54.         background-
55.         image: url(../../resources/themes/images/default/form/search-trigger.gif);
56.     }
57. </style>
58. <script type="text/javascript" src="../../bootstrap.js"></script>
59. <script type="text/javascript" src="../../locale/ext-lang-
60. zh_CN.js"></script>
61. <script type="text/javascript" src="editgrid.js"></script>
62. </head>
63. <body>
64. <div id="demo"></div>
65. </body>
66. </html>

```

editgrid.js :

```

1. Ext.Loader.setConfig({enabled: true});
2. Ext.Loader.setPath('Ext.ux', '../../examples/ux');
3. Ext.require([
4.     'Ext.grid.*',
5.     'Ext.toolbar.Paging',
6.     'Ext.util.*',
7.     'Ext.data.*',
8.     'Ext.state.*',
9.     'Ext.form.*',
10.    'Ext.ux.form.SearchField',
11.    'Ext.selection.CellModel',
12.    'Ext.ux.CheckColumn',
13.    'Ext.selection.CheckboxModel'
14. ]);
15.
16. Ext.onReady(function(){

```

```

17.     var isEdit = false;
18.
19.     function formatDate(value){
20.         return value ? Ext.Date.dateFormat(value, 'Y-m-d') : '';
21.     }
22.
23.     Ext.define('MyData',{
24.         extend: 'Ext.data.Model',
25.         fields: [
26.             {name:'id'},
27.             {name:'title',type:'string'},
28.             {name:'author'},
29.             {name:'hits',type: 'int'},
30.             {name:'addtime',type:'date',dateFormat:'Y-m-d'},
31.             {name:'checked',type:'bool'}
32.         ]
33.     });
34.
35.     //创建数据源
36.     var store = Ext.create('Ext.data.Store', {
37.         //分页大小
38.         pageSize: 50,
39.         model: 'MyData',
40.         //是否在服务端排序
41.         remoteSort: true,
42.         autoDestroy: true,
43.         proxy: {
44.             //异步获取数据，这里的URL 可以改为任何动态页面，只要返回 JSON 数据即可
45.             type: 'ajax',
46.             url: 'editgrid.asp',
47.
48.             reader: {
49.                 root: 'items',
50.                 totalProperty : 'total'
51.             },
52.             simpleSortMode: true
53.         },
54.         sorters: [{
55.             //排序字段。
56.             property: 'hits',
57.             //排序类型，默认为 ASC
58.             direction: 'DESC'
59.         }]
60.     });
61.     //下拉框数据,测试数据。
62.     var cbstore = Ext.create('Ext.data.Store', {
63.         fields: ['id', 'name'],
64.         data : [
65.             {"id":"1","name":"佚名"},
66.             {"id":"2","name":"管理员"},
67.             {"id":"3","name":"编辑"},
68.             {"id":"4","name":"总编辑"},
69.             {"id":"5","name":"测试员"}
70.         ]
71.     });
72.
73.
74.     //创建多选
75.     var selModel = Ext.create('Ext.selection.CheckboxModel');
76.     var cellEditing = Ext.create('Ext.grid.plugin.CellEditing', {

```

```

77.         clicksToEdit: 1
78.     });
79.     //创建 Grid
80.     var grid = Ext.create('Ext.grid.Panel',{
81.         store: store,
82.         selModel: selModel,
83.         columnLines: true,
84.         columns: [{
85.             id:"title",
86.             header: "标题",
87.             width: 110,
88.             dataIndex: 'title',
89.             sortable: true,
90.             field: {
91.                 allowBlank: false
92.             }
93.         },{
94.             header: "作者",
95.             width: 120,
96.             dataIndex: 'author',
97.             id:'gc',
98.             sortable: false,
99.             field: {
100.                 xtype: 'combobox',
101.                 id:'authors',
102.                 typeAhead: true,
103.                 triggerAction: 'all',
104.                 queryMode: 'local',
105.                 selectOnTab: true,
106.                 store: cbstore,
107.                 lazyRender: true,
108.                 displayField:'name',
109.                 valueField:'id',
110.                 listClass: 'x-combo-list-small',
111.                 listeners:{
112.                     select : function(combo, record,index){
113.                         isEdit = true;
114.                     }
115.                 }
116.             },
117.             renderer:rendererData
118.         },{
119.             header: "点击数",
120.             width: 80,
121.             dataIndex: 'hits',
122.             sortable: true,
123.             field: {
124.                 xtype: 'numberfield',
125.                 allowBlank: false,
126.                 minValue: 0,
127.                 maxValue: 100000
128.             }
129.         },{
130.             header: "添加时间",
131.             width: 100,
132.             dataIndex: 'addtime',
133.             sortable: true,
134.             renderer: formatDate,
135.             field: {
136.                 xtype: 'datefield',
137.                 format: 'y-m-d',
138.                 minValue: '01/01/06'
139.             }

```

```

140.     }
141. }, {
142.     xtype: 'checkcolumn',
143.     header: '审核',
144.     dataIndex: 'checked',
145.     width: 55
146. }],
147. height: 400,
148. width: 520,
149. x: 20,
150. y: 40,
151. title: 'ExtJS4 EditGrid(可编辑的 Grid)',
152.
153. disableSelection: false, // 值为 TRUE, 表示禁止选择
154. frame: true,
155. selType: 'cellmodel',
156. loadMask: true,
157. renderTo: 'demo',
158. viewConfig: {
159.     id: 'gv',
160.     trackOver: false,
161.     stripeRows: false
162. },
163. dockedItems: [{
164.     dock: 'top',
165.     xtype: 'toolbar',
166.     items: [{
167.         itemId: 'Button',
168.         text: '显示所选',
169.         tooltip: 'Add a new row',
170.         iconCls: 'add',
171.         handler: function() {
172.             var record = grid.getSelectionModel().getSelection();
173.             if (record.length == 0) {
174.                 Ext.MessageBox.show({
175.                     title: "提示",
176.                     msg: "请先选择您要操作的行!",
177.                     icon: Ext.MessageBox.INFO,
178.                     buttons: Ext.Msg.OK
179.                 })
180.                 return;
181.             } else {
182.                 var ids = "";
183.                 for (var i = 0; i < record.length; i++) {
184.                     ids += record[i].get("id")
185.                     if (i < record.length - 1) {
186.                         ids = ids + ",";
187.                     }
188.                 }
189.                 Ext.MessageBox.show({
190.                     title: "所选 ID 列表",
191.                     msg: ids
192.                     // icon: Ext.MessageBox.INFO
193.                 })
194.             }
195.         }
196.     }],
197.     width: 300,
198.     fieldLabel: '搜索',
199.     labelWidth: 50,
200.     xtype: 'searchfield',
201.     store: store

```

```

202.         }]
203.     }, {
204.         dock: 'bottom',
205.         xtype: 'pagingtoolbar',
206.         store: store,
207.         pageSize: 25,
208.         displayInfo: true,
209.         displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',
210.         emptyMsg: '没有数据'
211.     }],
212.     plugins: [cellEditing]
213.
214. })
215. store.loadPage(1);
216. grid.on('edit', onEdit, this);
217.
218. function onEdit(){
219.     var record = grid.getSelectionModel().getSelection()[0];
220.     //这里进行异步操作, 关于 Extjs 的异步操作这里不做演示, 仅列出所有值。
221.     var title = record.get('title');
222.     var author = record.get('author');//注意, 这里得到的是 id 值, 而不是
    name 值, 如果没有修改作者, 那么得到的值是默认显示的字符串, 这个需要在服务端进行判断并
    处理。
223.     var clk = record.get('hits');
224.     var addtime = Ext.Date.dateFormat(record.get('addtime'), 'Y-m-d');
225.     var checked = record.get('checked');
226.     Ext.MessageBox.show({
227.         title: "修改的数据为",
228.         msg: title + "\r\n" + author + "\r\n" + clk + "\r\n" + addtime + "\r\n" + checked,
229.         icon: Ext.MessageBox.INFO,
230.         buttons: Ext.Msg.OK
231.     })
232. }
233.
234. function rendererData(value, metadata, record){
235.     if(isEdit){
236.         var index = cbstore.find(Ext.getCmp('authors').valueField, value);
237.         var record = cbstore.getAt(index);
238.         return record.data.name;
239.     }else{
240.         return value;
241.     }
242. }
243. }
244. })

```

editgrid.asp : 时间关系, 只简单做了些测试数据。

```

1.  <%
2.      Response.ContentType = "text/html"
3.      Response.Charset = "UTF-8"
4.  %>
5.  <%
6.      '返回 JSON 数据, 自定义一些测试数据。。
7.      '这里的参数与 EXT3.x 相同, 区别在于排序字段和排序方式使用了新的属性。
8.      '由于这里是测试数据, 接收的参数只用到了 start, limit。 sorts 和 dir 在实际操作过程中,
    将之加入 SQL 的 ORDER BY 里即可。

```

```

9. start = Request("start")
10. limit = Request("limit")
11. '查询时获取的参数。
12. query = Request("query")
13. If start = "" Then
14.     start = 0
15. End If
16. If limit = "" Then
17.     limit = 50
18. End If
19. sorts = Replace(Trim(Request.Form("sort")), ",", "")
20. dir = Replace(Trim(Request.Form("dir")), ",", "")
21.
22. '测试数据，这里直接输出结果，实际应用中，应该把查询条件放到 SQL 语句中。
23. If query = "newstitle" Then
24.     Echo("{")
25.     Echo("total":")
26.     Echo("1")
27.     Echo("","items":[")
28.     Echo("{")
29.     Echo("title":"newstitle")
30.     Echo(",")
31.     Echo("author":"author")
32.     Echo(",")
33.     Echo("hits":"100")
34.     Echo(",")
35.     Echo("addtime":"&Now()&")
36.     Echo("}")
37.     Echo("]})")
38. Else
39. Dim counts:counts=20
40. '注意，这里的 counts 相当于 Rs.RecordCount,也就是记录总数。
41.
42. Dim Ls:Ls = Cint(start) + Cint(limit)
43. If Ls >= counts Then
44.     Ls = counts
45. End If
46.
47. Echo("{")
48. Echo("total":")
49. Echo(""&counts&"",")
50. Echo("items":[")
51. For i = start+1 To Ls
52.     Echo("{")
53.     Echo("id":"&i&")
54.     Echo(",")
55.     Echo("title":"newstitle"&i&")
56.     Echo(",")
57.     Echo("author":"author"&i&")
58.     Echo(",")
59.     Echo("hits":"&i&")
60.     Echo(",")
61.     Echo("addtime":"&Now()&")
62.     Echo(",")
63.     If i Mod 4 = 0 Then
64.         Echo("checked":"true")
65.     Else
66.         Echo("checked":"false")
67.     End If
68.     Echo("}")
69.     If i<Ls Then
70.         Echo(",")
71.     End If
72. Next

```



```

73.
74. Echo("]}")
75. End If
76. Function Echo(str)
77.     Response.Write(str)
78. End Function
79. %>

```

由于 4.x 有了重大更新，所以很多在 3.x 中的东西在 4.x 里并不好使，直接导致了本文迟迟没有发表，原因就出在了更新下拉框后，Grid 中显示的是 ID 号而不是 name 值。一番查看 API 及研究，终于发现问题所在，3.x 中 renderer combobox 在 4.x 并不好用，最后重新读了一遍自己写的代码，才有所顿悟。希望文章能起到一个抛砖引玉的作用，大家一起进步。

## 4.10 图片验证码的实现

上几篇文章，主要学习了 Extjs4 Grid 的使用方法，从本篇开始，我们开始其他组件的学习，使用。在登录、注册甚至是发表文章或帖子的时候，都会用到验证码这个东西，那么在 EXTJS 中，可以使用验证码功能么？答案是肯定的，在 EXTJS4 之前，也有很多验证码的实现，在 Extjs4 中，验证码到底如何实现呢？

暂时，我们将验证码组件，命名为 CheckCode。此组件继承自 Ext.form.field.Text，在实现之前，我们需要写两个样式，分别用来控制验证码的输入框和验证码图片的大小。

CSS 样式为：

```

1. #CheckCode{ float:left;}
2. .x-form-code{width:73px;height:20px;vertical-align:middle;cursor:pointer; float:left; margin-left:7px;}

```

记住这两个样式的定义，后面，我们会用到它。

验证码的 JS 代码：

```

1. Ext.define('SMS.view.CheckCode',{
2.     extend: 'Ext.form.field.Text',
3.     alias: 'widget.checkcode',
4.     inputType: 'codefield',
5.     codeUrl: Ext.BLANK_IMAGE_URL,
6.     isLoader: true,
7.     onRender: function(ct,position){
8.         this.callParent(arguments);
9.         this.codeEl = ct.createChild({ tag: 'img', src: Ext.BLANK_IMAGE_URL})
10.    };
11.    this.codeEl.addCls('x-form-code');
12.    this.codeEl.on('click', this.loadCodeImg, this);
13.    if (this.isLoader) this.loadCodeImg();
14.    },
15.    alignErrorIcon: function() {
16.        this.errorIcon.alignTo(this.codeEl, 'tl-tr', [2, 0]);
17.    },
18.
19.    loadCodeImg: function() {
20.        this.codeEl.set({ src: this.codeUrl + '?id=' + Math.random() });
21.    }
22.
23. })

```

以上代码中，定义了一个“类”，名字是：SMS.view.CheckCode，其实这个名字，相当于 extjs 3.x 之中的命名空间，以前也提到过。它继承自 Ext.form.field.Text，在它的 onRender 中，我们写了一些代码。其中 **this.callParent(arguments);** 代替了 xxxx.superclass.onRender.call(this, ct, position); 在 Ext.form.field.Text 的基础上，使用 createChild 方法，创建了一个图片，并为其添加了一个名为 x-form-code，而后，给其创建了一个 click 事件，这个事件实现的功能是，当我们点击验证码图片时，换另外一张图片，也就是常说的：“看不清？换一张功能。”，最后，如果 isLoader 为 True 时，调用 loadCodeImg 方法。至此，验证码功能全部完成了。下面，我们看看如何使用。

新建 Login.js 文件，定义“类”SMS.view.Login，其全部代码为：

```

1. Ext.define('SMS.view.Login',{
2.     extend:'Ext.window.Window',
3.     alias: 'widget.loginForm',
4.     requires: ['Ext.form.*','SMS.view.CheckCode'],
5.     initComponents:function(){
6.         var checkcode = Ext.create('SMS.view.CheckCode',{
7.             cls : 'key',
8.             fieldLabel : '验证码',
9.             name : 'CheckCode',
10.            id : 'CheckCode',
11.            allowBlank : false,
12.            isLoading:true,
13.            blankText : '验证码不能为空',
14.            codeUrl: '/include/checkCode.asp',
15.            width : 160
16.        })
17.        var form = Ext.widget('form',{
18.            border: false,
19.            bodyPadding: 10,
20.            fieldDefaults: {
21.                labelAlign: 'left',
22.                labelWidth: 55,
23.                labelStyle: 'font-weight:bold'
24.            },
25.            defaults: {
26.                margins: '0 0 10 0'
27.            },
28.            items:[{
29.                xtype: 'textfield',
30.                fieldLabel: '用户名',
31.                blankText : '用户名不能为空',
32.                allowBlank: false,
33.                width:240
34.            },{
35.                xtype: 'textfield',
36.                fieldLabel: '密    码',
37.                allowBlank: false,
38.                blankText : '密码不能为空',
39.                width:240,
40.                inputType : 'password'
41.            },checkcode],
42.            buttons:[{
43.                text:'登录',
44.                handler:function(){
45.
46.                }
47.            },{
48.                text:'取消',
49.                handler:function(){
50.
51.            }

```

```

52.         }]
53.     })
54.     Ext.apply(this,{
55.         height: 160,
56.         width: 280,
57.         title: '用户登陆',
58.         closeAction: 'hide',
59.         closable : false,
60.         iconCls: 'login',
61.         layout: 'fit',
62.         modal : true,
63.         plain : true,
64.         resizable: false,
65.         items:form
66.     });
67.     this.callParent(arguments);
68. }
69. });

```

然后在主页面的代码中调用此 LoginWindow。

```

1.  requires:['SMS.view.Login']
2.  var win;
3.  win = Ext.create('SMS.view.Login').show();

```

最后效果图：



## 4.11tabpanel

创建一个 `tabpanel` 有两种方法：

- 一： `Ext.createWidget('tabpanel',{...})`
- 二： `Ext.create('Ext.tab.Panel',{...})`

本文分别介绍这两种创建方法。

HTML 代码：

```

1.  <!DOCTYPE html PUBLIC "-
    //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtm
    l1-transitional.dtd">
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head>
4.  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.  <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
    all.css" />
6.  <script type="text/javascript" src="../../bootstrap.js"></script>

```

```

7. <script type="text/javascript" src="../../locale/ext-lang-
   zh_CN.js"></script>
8. <script type="text/javascript" src="tabs.js"></script>
9. <title>MHZG.NET--Tabs</title>
10.</head>
11.
12.<body>
13.<div id="tab"></div>
14.</body>
15.</html>
   tabs.js:

```

```

1. Ext.require('Ext.tab.*');
2. Ext.onReady(function(){
3.     //第一种方式创建
4.     var tabs = Ext.createWidget('tabpanel', {
5.         renderTo: 'tab',
6.         width: 450,
7.         activeTab: 0,
8.         margin: '50 10 50 80',
9.         defaults :{
10.             bodyPadding: 10
11.         },
12.         items: [{
13.             //contentEl: 'script', //将指定容器中的内容加载到 tabPanel 的内容区
14.             title: 'Tabs-1',
15.             closable: true,
16.             html: 'Tabs-1 内容。'
17.         }, {
18.             title: 'Tabs-2',
19.             closable: false,
20.             html: 'Tabs-2 内容'
21.         }
22.     ]});
23.
24.     //第二种方式创建
25.     var tabs2 = Ext.create('Ext.tab.Panel', {
26.         renderTo: document.body,
27.         activeTab: 0,
28.         width: 600,
29.         height: 250,
30.         plain: true,
31.         margin: '0 10 0 80',
32.         defaults :{
33.             autoScroll: true,
34.             bodyPadding: 10
35.         },
36.         items: [{
37.             title: 'Tabs-1',
38.             html: "这里显示内容"
39.         }, {
40.             title: '异步加载内容',
41.             loader: {
42.                 url: 'ajax.htm',
43.                 contentType: 'html',
44.                 loadMask: true
45.             },
46.             listeners: {
47.                 activate: function(tab) {
48.                     tab.loader.load();
49.                 }
50.             }
51.         }, {
52.             title: '异步加载内容 1',
53.             loader: {

```

```

54.         url: 'ajax1.htm',
55.         contentType: 'html',
56.         autoLoad: true,
57.         params: 'foo=123&bar=abc'
58.     }, {
59.         title: '点击触发事件',
60.         listeners: {
61.             activate: function(tab){
62.                 alert(tab.title);
63.             }
64.         },
65.         html: "点击 Tab 之后，触发事件，监听事件: activate。activate 可传
        递两个参数。1、Ext.Component this。2、Object options "
66.     }, {
67.         title: 'Tabs 不可能',
68.         disabled: true
69.     }
70. ]
71. })
72. });
73. });

```

第二个 `tabpanel` 中有两个 `html` 文件，分别是 `ajax.htm` 和 `ajax1.htm`，这两个文件代码就不写了，里面就是敲了一些字，而这些字就是 `tabpanel` 内容区的那些文字，不过需要注意的一点就是，在异步获取其他文件中的内容时，这些文件返回的编码格式应该是 UTF-8...

## 4.12 选项卡(tabs)

`tab` 选项卡，是 `Extjs` 中常用的组件，`tab` 一般依附于 `tabpanel`，很多时候我也认为 `tab` 就是 `tabpanel`，但在官方 `api` 中，确实有 `Ext.tab.Panel` 和 `Ext.tab.Tab` 之分，具体属性、事件、方法，请[参考 API](#)，`tab` 选项卡可以单独渲染，使用方法是：`xtype: 'tab'`，本文介绍 `tab` 的基本用法。

HTML 代码：除了加载基本库以外，定义了一些 CSS 样式，这些 CSS 基本没用，就是我看所有组件靠着 `BODY` 太不舒服了。。

```

1. <!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
all.css" />
6. <script type="text/javascript" src="../../bootstrap.js"></script>
7. <script type="text/javascript" src="../../locale/ext-lang-
zh_CN.js"></script>
8. <script type="text/javascript" src="tabs1.js"></script>
9. <title>MHZG.NET--Tabs</title>
10. <style>
11.     .main{ margin:50px auto;}
12.     #add{ padding-left:10px;}
13. </style>
14. </head>
15.

```

```
16. <body>
17. <div class="main">
18. <div id="add"></div>
19. <div id="tab"></div>
20. </div>
21. </body>
22. </html>
```

tabs1.js :

```
1. Ext.require('Ext.tab.*');
2. Ext.onReady(function(){
3.     var currentItem;
4.     var tabs = Ext.createWidget('tabpanel', {
5.         renderTo: 'tab',
6.         resizeTabs: true,
7.         enableTabScroll: true,
8.         margin: '10',
9.         width: 600,
10.        height: 250,
11.        defaults: {
12.            autoScroll: true,
13.            bodyPadding: 10
14.        },
15.        items: [{
16.            title: '选项卡',
17.            html: '选项卡内容',
18.            closable: true
19.        }]
20.    });
21. });
22.
23. var index = 0;
24.
25. function addTab (closable) {
26.     ++index;
27.     tabs.add({
28.         title: '新选项卡- ' + index,
29.         html: '新选项卡内容 ' + index + '<br/><br/>',
30.         closable: !!closable
31.     }).show();
32. }
33.
34. Ext.createWidget('button', {
35.     renderTo: 'add',
36.     text: '创建可关闭选项卡',
37.     handler: function () {
38.         addTab(true);
39.     }
40. });
41.
42. Ext.createWidget('button', {
43.     renderTo: 'add',
44.     text: '创建不可关闭选项卡',
45.     handler: function () {
46.         addTab(false);
47.     },
48.     style: 'margin-left: 8px;'
49. });
50. });
```

## 4.13 上传文件

本文介绍 Extjs4 文件上传示例，Extjs4 中，主要使用 `up('form').getForm().submit()` 方法来实现文件的上传，在 `submit` 方法中，指定其 `type` 属性，这点很重要，如果不指定，那么在上传完成后的处理中，除非服务端不返回数据，否则客户端就会报错。实例代码如下：

upload.html:

```
1. <!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
   11-transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
   all.css" />
6. <script type="text/javascript" src="../../bootstrap.js"></script>
7. <script type="text/javascript" src="../../locale/ext-lang-
   zh_CN.js"></script>
8. <script type="text/javascript" src="upload.js"></script>
9. <title>MHZG.NET--upload</title>
10. </head>
11.
12. <body>
13. <div id="upload"></div>
14. </body>
15. </html>
```

upload.js:

```
1. Ext.onReady(function(){
2.     Ext.create('Ext.form.Panel', {
3.         title: '文件上传示例--MHZG.NET',
4.         width: 400,
5.         bodyPadding: 10,
6.         margin: '50 10 50 80',
7.         frame: true,
8.         renderTo: 'upload',
9.         items: [{
10.             xtype: 'filefield',
11.             name: 'fileName',
12.             fieldLabel: '上传',
13.             labelWidth: 50,
14.             msgTarget: 'side',
15.             allowBlank: false,
16.             anchor: '100%',
17.             buttonText: '选择文件'
18.         }],
19.
20.         buttons: [{
21.             text: '上传',
22.             handler: function() {
23.                 var form = this.up('form').getForm();
24.                 if(form.isValid()){
25.                     form.submit({
26.                         url: 'upload.asp',
27.                         type: 'ajax',
28.                         waitMsg: '正在保存文件...',
29.                         success: function(fp, o) {
30.                             // Ext.Msg.alert('提示信息', '文件成功上传,文件名字
   为: '+o.result.file);
31.                             Ext.Msg.show({
32.                                 title: '提示信息',
```

```

33.                msg: '文件上传成功<br>上传文件名为: '+o.result.file,
34.                minWidth:200,
35.                modal:true,
36.                buttons:Ext.Msg.OK
37.            })
38.            form.findField('fileName').setRawValue('');
39.        }
40.    });
41.    }
42.    }
43.    }]
44.    });
45.    });

```

服务端文件 `upload.asp` 的内容就不写了，由于在文件中用到了各种无组件上传代码，所以比较杂乱，注意一点就可以，在任何类型的服务端处理文件中，在处理完上传之后，给客户端返回一段 JSON，客户端就可以通知客户上传完成。服务端处理完上传之后返回的 JSON 字符串为如下类型：{success:true,file:""&fileName&""}，JSON 字符串可以任意组合，这里的 `fileName` 为上传后文件地址+文件名（/UploadFile/2011830.gif），以便于客户端进行其他操作。

最后，在代码中用到了 `form.findField('fileName').setRawValue('');`，这句是清空 `file` 选择框，因为用 `form.reset();` 根本无法清除 `file` 选择框中的内容，无奈之下用了这个办法，如果大家有更好的办法，欢迎留言指正。。

## 4.14 ComponentQuery

对于 Extjs3.x 来说,Extjs4.0 组件查找使用了 ComponentQuery 类，这个类是用来查找容器内的组件的。实现方式类似于 CSS 的 Selector。在 extjs3.x 中，查找组件的方式有很多，例如：（ID 组件 ID）、（ref 组件引用）、（items.get(0)组件的层级）。在 extjs4.0 中，则多了 ComponentQuery。利用这个类，将更加方便的查找组件。下面，则是将 extjs3 和 extjs4 各种查找组件的方式做下对比。

**Extjs3.x:**

**ID:** 这就是所熟知的 `Ext.getCmp("组件 ID")`，虽然说这种方式查找组件最直接，也最容易，但是如果随着程序体积变大，组件越来越多，那么可能会发生组件 ID 相同的事情发生，这样一来，就可能发生显示的问题和获取对象不正常。

**ref:** 在 EXTJS3 中，所有的组件都会有一个 `ref` 属性，也就是 `reference` 的意思。这种方式是通过组件的引用而得到组件对象。例如：`ref:'mypanel'`，但是这种方式的局限性在于，他只能查找不同层级之间的组件。也就是说 `A[a,b,c] C[d,e,f]`，这样的方式，使用 `ref` 非常方便，但如果是 `B[c,c,d]` 这样的方式，就会出现问題，因为在某一层级上，出项了两个相同的组件。

**items.get(0):** 这种方式是通过首先获得组件中元素的一个数组，然后通过数组获得想要的组件对象，这种方式，我们一般不会用到，因为，他非常不灵活，如果我们的元素层级发生改变的话，我们将不能获取我们想要的组件，维护起来也非常困难。所以，这种方式，是用的最少的一种方式。

在 Extjs4.0 中，我们依然可以沿用 Extjs3.x 中查找组件的方式，但是在 Extjs4.0 中，我们可以利用 ComponentQuery，方便找到对应组件。

1、通过组件 ID 获取组件: "#组件 ID"，如果通过这种方式，那么一定要记住在组件 ID 前添加#号。

2、得到某一组件下所有的指定类型的组件: "panel>button"，这种方式是查找所有 panel 组件下的所有 button 组件。

3、通过 xtype: "treepanel"或".treepanel



4、如果想获取所有 button 并且 action 为 save 的 button，则可以使用 "button[action=save] "，又或者获取所有 panel，并且 autoscroll 属性为 true 的 panel，则可以使用 "panel[autoScroll=true]"

还有两种方式，是查找某一组件的子组件或上级组件，例如：

- 1、查找 window 下的 form: win.down("form")
- 2、查找 button 的父组件 window: button.up("window");

最明显的例子就是我们在 extjs4.0 使用 MVC 模式进行开发的时候，经常会在 control 控制中大量使用 'viewport > panel', 'edit button[action=save]' 这类查找，当我们点击 button 进行数据保存的时候，我们会使用

```
1. var win    = button.up('window'),
2.     form   = win.down('form'),
3.     record = form.getRecord(),
   这类型的查找进行数据更新。
```

最后，在 extjs4.0 MVC 模式中，经常会碰到

```
1. refs:[
2.     {ref: 'menu',selector: 'tablepanel'},
3.     {ref: 'feedList', selector: 'feedlist'},
4.     {
5.         ref: 'feedWindow',
6.         selector: 'feedwindow',
7.         autoCreate: true,
8.         xtype: 'feedwindow'
9.     }
10. ]
```

这样的书写方式。有很多人问我这是什么意思，查 API 也找不到。网络上也找不到，其实看 extjs3.x 中查找组件的方式，就会明白了。Refs 是一个查找并匹配组件的集合，集合里面则指定了需要的操作，即查找一个组件，而找到的这个组件所匹配组件就是 selector 指定的，用上面代码一条 {ref: 'menu',selector: 'tablepanel'} 来稍做解释，即查找 menu 组件（其实是一颗树），点击树节点，结果将在 'tablepanel' 中显示。这样解释可能比较含糊，那么就用官方例子做个解释吧。{ref: 'feedData', selector: 'feedlist dataview'}。这条的含义是查找 'feedData'，将 'feedData' 显示在 'feedlist' 的子组件 dataview 上。

总结下，就是在 ExtJS4 中所有的组件都有一个 query 方法，这个方法就是 ComponentQuery 的实现。当然，如果还是不明白的话，就可以使用：

```
Ext.ComponentQuery.query("tabpanel");
Ext.ComponentQuery.query("#mytree")
```

等等。实在 extjs4.0 实际使用过程中，如果要调用或查找组件，extjs 会自动调用 ComponentQuery 的 query 方法来查找对应组件。

例子：

```
1. init:function(){
2.     this.control({
3.         'smsmenu': { //这里不必写全部的代码 Ext.ComponentQuery.query(...);
4.             itemmousedown: this.loadMenu
5.         }
6.     })
7. }
```

这就是本节所学到的内容，如果文中有错误或有更好的方法，欢迎留言交流。我们一起学习，共同进步！！

## 4.15 Ext.data.model

extjs4.0 中，在 data 类中新增加了一个 Model（模型类），这个类有点类似于 extjs3.x 中的 record。Model 类功能非常强大，尤其在 extjs4.0 MVC 开发中，非常实用。

如何定义 Model 类：

```
1. Ext.regModel("Student",{
2.
3.     fields:[
4.
5.         {name:"name",type:"string"},
6.         {name:"class",type:"string"}
7.     ]
8. });
```

Model 类中最重要的属性就是 Fields，这个属性接受一个数组，用来设置显示数据中所包含的字段，"Student"设置了该类的名字。

Model 类的功能：

```
1. Ext.regModel("Student",{
2.     fields:[
3.         {name:"name",type:"string"},
4.         {name:"class",type:"string",convert:function(val){
5.             if(val=="1"){return
6.                 "一班"};
7.             if(val=="2"){return
8.                 "二班"};
9.             if(val=="3"){return
10.                 "三班"};
11.         }}
12.     ]
13. });
```

首先我们来看一下 fields 属性中的功能，例如：我们现在定义了一个如上的 Model 类，然后，数据源返回班级（class）这个属性时。格式为（1、2、3），如果我们把这样的数据呈现出来，那么将显示的很不友好。所以，我们要在呈现之前，对数据做一个转换。把原本不友好的数据，转换成有好的数据，这就用到了。fields 中的 convert 属性。

```
1. var student=new Student({
2.     name:"test",
3.
4.
5.     class:"3"
6.
7.
8. });
```

2、定义好一个 Model 类之后，我们就可以使用这个 Model 类。最简单的方式，我们是直接 new 一个 Model 类的对象，然后将我们的数据信息加载到对象中，有点类似于 Ext3.X 中的 record 的对象的使用。

上述代码我们根据 Student 的模型类，定义了一个该类的对象，但在实际应用程序中，像这样的情况很少，毕竟我们的数据不一定是这种一成不变的数据，有时候我们需要从后台加载我们的数据，然后给予我们的 model 类。这就需要我们的模型类有能够请求后台的能力，这也是 Model 中提供的第二个功能，Model 中提供一个属性 proxy（代理）。这个 proxy 中有几个比较重要的属性（type、url、reader），type 属性值是一个字符串形式，用来表明。该代理的一种类型，具体的可以查看 API 中了解有哪些类型，

url 也就是请求后台的 url。reader，也就是我们要用什么样的阅读器，来解析我们的数据：

```
1. Ext.regModel("Student",{
2.
3.
4.     fields:[
5.
6.
7.     {name:"name",type:"string"},
8.     {name:"class",type:"string",convert:function(val){
9.     if(val=="1"){return "一班"};
10.    if(val=="2"){return "二班"};
11.    if(val=="3"){return "三班"};
12.    }}
13.    ],
14.    proxy:{
15.    type:"rest",
16.    url:"server/service.aspx",
17.    reader:"json"
18.    }
19. });
20.
21.
22. Student.load(001,{
23.
24.
25.    success:function(student){
26.    //处理加载完成的逻辑
27.    }
28. });
```

service.aspx 返回的数据格式：

```
{id:001,name:"zhangsan",class:"2"}
```

经过上述的设置，我们的 Model 就可以与后台交互，并要求后台返回我们想要的數據了，这个也是之前 record 类所办不到的。

3、在我们的应用程序中，可能我们定义的 Model 不止一个。但是，他们之间可能都是独立的，没有任何的关系，但是，在我们的应用程序中可能在后台返回的数据中存在一定的联系，而且我们又想让这些 Model 之间存在一定的联系，这样我们在处理起来，会比较简单一些。大家看下边的一段数据。

```
1. {
2.   id:"009",
3.   name:"Jerry",
4.   subjects:[
5.     {id:"001",name:"English"},
6.
7.
8.     {id:"002",name:"Mathematics"}
9.   ]
10. }
```

在上述的数据中，科目和学生之间是有一定的联系的。所以，我们也在想，解析数据的时候，让他们保持这种联系，以便于我们更好的解析数据。这样我们就用到了 Model 中的 belongsTo 和 hasMany 这样两个属性。首先要解析这样的数据，我们需要定义好我们的 Model 类。如下：

```
1. Ext.regModel("subject",{
```

```

2.
3.
4.  fields:[
5.  {name:"id",type:"string"},
6.  {name:"name",type:"string"}
7.  ],
8.  belongsTo:"Stdudent"
9.  });
10.
11.
12.
13.
14. Ext.regModel("Student",{
15.  fields:[
16.  {name:"id",type:"string"},
17.  {name:"name",type:"string"}
18.  ],
19.
20.  ],
21.
22.
23.  proxy:{
24.  type:"rest",
25.  url:"servicee/servicee.aspx",
26.  reader:"json"
27.  },
28.  hasMany:[{model:"subject",name:"subjects"}]
29.
30.
31. });

```

在我们定义好 Model 后，下面我们就可以加载并解析我们的数据了。

```

1.  Student.load("009",{
2.
3.
4.  success:function(student){
5.
6.
7.  alert(student.get("id"));
8.
9.
10. alert(student.subjects().getCount());//我们可以直接访问该学生的科目
11. }
12. })
13. });

```

4、在 ExtJS4 的 Model 中还提供了对字段列的验证功能。我们想验证字段只需要设置 Model 类的 validations 属性即可，代码如下：

```

1.  Ext.regModel("Student",{
2.  fields:[
3.  {name:"id",type:"string"},
4.  {name:"name",type:"string"}
5.
6.
7.  ],
8.
9.
10. proxy:{
11. type:"rest",
12. url:"data/1.aspx",
13. reader:"json"
14. },

```

```

15. hasMany:[{model:"subject",name:"subjects"}],
16.
17.
18. validations:[
19. {type:"presence",field:"id"},
20.
21.
22. {type:"length",field:"name",min:3}
23. ]
24.
25.
26. });
27.
28.
29. var student=new Student({id:"001",name:"z"});
30.
31.
32. var stuvalidate=student.validate();//得到验证类
33.
34.
35. stuvalidate.isValid();//返回验证结果 true 或 false
36.
37.
38. stuvalidate.each(function(error){
39.
40.
41. alert(error.field+" "+error.message);//遍历验证的信息
42. });

```

## 4.16 Combobox 三级联动

很多网友在问，Extjs4.0 ComboBox 如何实现，好在之前用 3.x 实现过一个三级联动，如今用 Extjs4.0 来实现同样的联动效果。其中注意的一点就是，3.x 中的 `model:'local'` 在 Extjs4.0 中用 `queryMode: 'local'` 来表示，而且在 3.x 中 Load 数据时用 `reload`，但是在 extjs4.0 中要使用 `load` 来获取数据。如下图：

### 代码部分

先看 HTML 代码：

```

1. <!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml
11-transitional.dtd">

```

```

2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>MHZG.NET-城市三级联动实例</title>
6. <link rel="stylesheet" type="text/css" href="../../resources/css/ext-
  all.css" />
7. <script type="text/javascript" src="../../bootstrap.js"></script>
8. <script type="text/javascript" src="../../locale/ext-lang-
  zh_CN.js"></script>
9. <script type="text/javascript" src="combobox.js"></script>
10. </head>
11.
12. <body>
13. </body>
14. </html>

```

简单的很，就是加载了基本的 CSS 文件和 JS 文件，并且加载自定义的 combobox.js 文件。

combobox.js:

```

1. Ext.require('Ext.*');
2. Ext.onReady(function(){
3.     //定义 ComboBox 模型
4.     Ext.define('State', {
5.         extend: 'Ext.data.Model',
6.         fields: [
7.             {type: 'int', name: 'id'},
8.             {type: 'string', name: 'cname'}
9.         ]
10.    });
11.
12.    //加载省数据源
13.    var store = Ext.create('Ext.data.Store', {
14.        model: 'State',
15.        proxy: {
16.            type: 'ajax',
17.            url: 'city.asp?act=sheng&n='+new Date().getTime()+''
18.        },
19.        autoLoad: true,
20.        remoteSort:true
21.    });
22.    //加载市数据源
23.    var store1 = Ext.create('Ext.data.Store', {
24.        model: 'State',
25.        proxy: {
26.            type: 'ajax',
27.            url: 'city.asp?act=shi&n='+new Date().getTime()+''
28.        },
29.        autoLoad: false,
30.        remoteSort:true
31.    });
32.    //加载区数据源
33.    var store2 = Ext.create('Ext.data.Store', {
34.        model: 'State',
35.        proxy: {
36.            type: 'ajax',
37.            url: 'city.asp?act=qu&n='+new Date().getTime()+''
38.        },
39.        autoLoad: false,
40.        remoteSort:true
41.    });
42.
43.
44.
45.    Ext.create("Ext.panel.Panel",{

```

```

46.     renderTo: document.body,
47.     width:290,
48.     height:220,
49.     title:"城市三级联动",
50.     plain: true,
51.     margin:'30 10 0 80',
52.     bodyStyle: "padding: 45px 15px 15px 15px;",
53.     defaults :{
54.         autoScroll: true,
55.         bodyPadding: 10
56.     },
57.     items:[{
58.         xtype:"combo",
59.         name:'sheng',
60.         id : 'sheng',
61.         fieldLabel:'选择省',
62.         displayField:'cname',
63.         valueField:'id',
64.         store:store,
65.         triggerAction:'all',
66.         queryMode: 'local',
67.         selectOnFocus:true,
68.         forceSelection: true,
69.         allowBlank:false,
70.         editable: true,
71.         emptyText:'请选择省',
72.         blankText : '请选择省',
73.         listeners:{
74.             select:function(combo, record,index){
75.                 try{
76.                     //userAdd = record.data.name;
77.                     var parent=Ext.getCmp('shi');
78.                     var parent1 = Ext.getCmp("qu");
79.                     parent.clearValue();
80.                     parent1.clearValue();
81.                     parent.store.load({params:{param: this.value}});
82.                 }
83.                 catch(ex){
84.                     Ext.MessageBox.alert("错误","数据加载失败。");
85.                 }
86.             }
87.         },
88.         {
89.             xtype:"combo",
90.             name:'shi',
91.             id : 'shi',
92.             fieldLabel:'选择市',
93.             displayField:'cname',
94.             valueField:'id',
95.             store:store1,
96.             triggerAction:'all',
97.             queryMode: 'local',
98.             selectOnFocus:true,
99.             forceSelection: true,
100.            allowBlank:false,
101.            editable: true,
102.            emptyText:'请选择市',
103.            blankText : '请选择市',
104.            listeners:{
105.                select:function(combo, record,index){
106.                    try{
107.                        //userAdd = record.data.name;
108.                        var parent = Ext.getCmp("qu");
109.                        parent.clearValue();
110.

```

```

111.         parent.store.load({params:{param: this.value}});
112.     }
113.     catch(ex){
114.         Ext.MessageBox.alert("错误", "数据加载失败。");
115.     }
116. }
117. },
118. },
119. {
120.     xtype: "combo",
121.     name: 'qu',
122.     id : 'qu',
123.     fieldLabel: '选择区',
124.     displayField: 'cname',
125.     valueField: 'id',
126.     store: store2,
127.     triggerAction: 'all',
128.     queryMode: 'local',
129.     selectOnFocus: true,
130.     forceSelection: true,
131.     allowBlank: false,
132.     editable: true,
133.     emptyText: '请选择区',
134.     blankText : '请选择区',
135. }
136. ]
137. })
138. });

```

上述代码中，如果在 **ComboBox** 直接定义 **store** 数据源，会出现这样一种情况，那就是当选择完第一个省，点击第二个市的时候，会闪一下，再点击，才会出现市的数据。那么要解决这样的情况，那么必须先要定义好省、市、区的数据源。那么再点击的时候，就不会出现上述情况了。

代码中，使用 **store** 为省的数据，设置其 **autoLoad** 为 **true**，那么页面第一次加载的时候，就会自动加载省的数据，然后给省和市添加了监听 **select**，作用在于当点击省的时候，要清空市和区的数据，并根据当前选定的值去加载对应的数据到市的数据源中。当然 **store1** 和 **store2** 原理是一样的。

最后，服务端要根据传的值进行数据检索及返回正确数据，这里没有从数据库中查询数据，而只是简单的写了一些测试代码，相信 **extjs** 代码没有任何的问题了，那么服务端返回数据，就不是一件很重要的事情了。

City.asp:

```

1.  <%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
2.  <%
3.      Response.ContentType = "text/html"
4.      Response.Charset = "UTF-8"
5.  %>
6.  <%
7.      Dim act:act = Request("act")
8.      Dim param : param = Request("param")
9.      If act = "sheng" Then
10.         Response.Write("[")
11.         Response.Write("{\"cname\":\"北京市\",\"id\":\"110000\"},")
12.         Response.Write("{\"cname\":\"内蒙古自治区\",\"id\":\"150000\"}")
13.         Response.Write("]")
14.      End If
15.      If act = "shi" Then
16.         If param = "110000" Then
17.             Response.Write("[")
18.             Response.Write("{\"cname\":\"市辖区\",\"id\":\"110100\"},")

```



```

19.         Response.Write("{\"cname\":\"市辖区\",\"id\":\"110200\"}")
20.         Response.Write("]")
21.     ElseIf param = "150000" Then
22.         Response.Write("[")
23.         Response.Write("{\"cname\":\"呼和浩特市\",\"id\":\"150100\"},")
24.         Response.Write("{\"cname\":\"包头市\",\"id\":\"150200\"}")
25.         Response.Write("]")
26.     End If
27. End If
28. If act = "qu" Then
29.     If param = "110100" Then
30.         Response.Write("[")
31.         Response.Write("{\"cname\":\"朝阳区\",\"id\":\"110101\"},")
32.         Response.Write("{\"cname\":\"昌平区\",\"id\":\"110102\"}")
33.         Response.Write("]")
34.     ElseIf param = "110200" Then
35.         Response.Write("[")
36.         Response.Write("{\"cname\":\"密云县\",\"id\":\"110201\"},")
37.         Response.Write("{\"cname\":\"房山县\",\"id\":\"110202\"}")
38.         Response.Write("]")
39.     ElseIf param = "150100" Then
40.         Response.Write("[")
41.         Response.Write("{\"cname\":\"回民区\",\"id\":\"150101\"},")
42.         Response.Write("{\"cname\":\"新城区\",\"id\":\"150102\"}")
43.         Response.Write("]")
44.     ElseIf param = "150200" Then
45.         Response.Write("[")
46.         Response.Write("{\"cname\":\"青山区\",\"id\":\"150201\"},")
47.         Response.Write("{\"cname\":\"东河区\",\"id\":\"150202\"}")
48.         Response.Write("]")
49.     End If
50. End If
51. %>

```

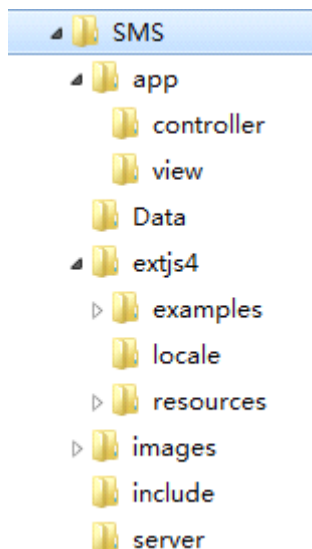
## 5 员工管理系统

### 5.1 准备工作

重写原因：由于开始准备的时候，就是按照传统开发去做的，写了一部分之后，有网友和同事提出：“为什么不用 MVC 模式”呢？这样的问题让我对目前传统开发的心发生了一些细微变法，是啊，为什么不用 MVC 模式呢？我征求了一下同事及热心网友的意见，都同意使用 MVC 模式开发。从而，我删掉了原来所有目录，重新按照 MVC 模式去组织目录结构，很快的，目录结构准备好了。

那。。我们重新来过，使用 EXTJS4.0 的 MVC 模式，开发这套员工管理系统。给个简称吧。SMS（你懂得。呵呵！）。

#### 一、建立环境：



**Data:** 数据库文件夹，里面放着管理系统用的数据库文件。数据库目前只有三张表。分别是：

**Menu:** 菜单项

**user:** 员工注册信息

**userinfo:** 员工个人资料信息

**Images:** 图片目录，一些自定义的图片文件

**Include:** 服务端文件目录，里面包含 ASP 所用到的 `Conn.asp`、`Function.asp` 等文件

**App:** 整个 SMS 所用到的自定义 JS 文件，里面有一个 `controller` 文件夹，一个 `view` 文件夹。`controller` 文件夹放置主代码，`view` 文件夹放置各组件。这几个文件夹中的内容会在第二章进行介绍。

**Extjs4:** 此目录放置 Extjs4 的库文件。

**Server:** 服务端目录，里面包含 ASP 服务端获取数据的各种 `.ASP` 文件。目前里面建立了一个叫 `MenuLoader.asp` 的文件，从名字上来看，这个文件是加载菜单使用。

## 5.2 框架的搭建

废话不多说了，上篇文章建立了比较基础的文件。今天开始搭建大体的框架，由于 Extjs4 在组件建立方面有了很大的改变，所以第一次建立的框架页面还是费了比较长的时间。本章内容增加了一些图片及 CSS 文件，目的是为了美化整个界面。增加的 CSS 文件：

**注意事项：**layout、region 的使用，如果没有看 API 及相关文档的话，那么面对错误对话框的时候，还不知道是怎么回事。

本文将 `main.js` 放到了 `/app/controller` 文件夹下，这将是整个项目的主体文件。

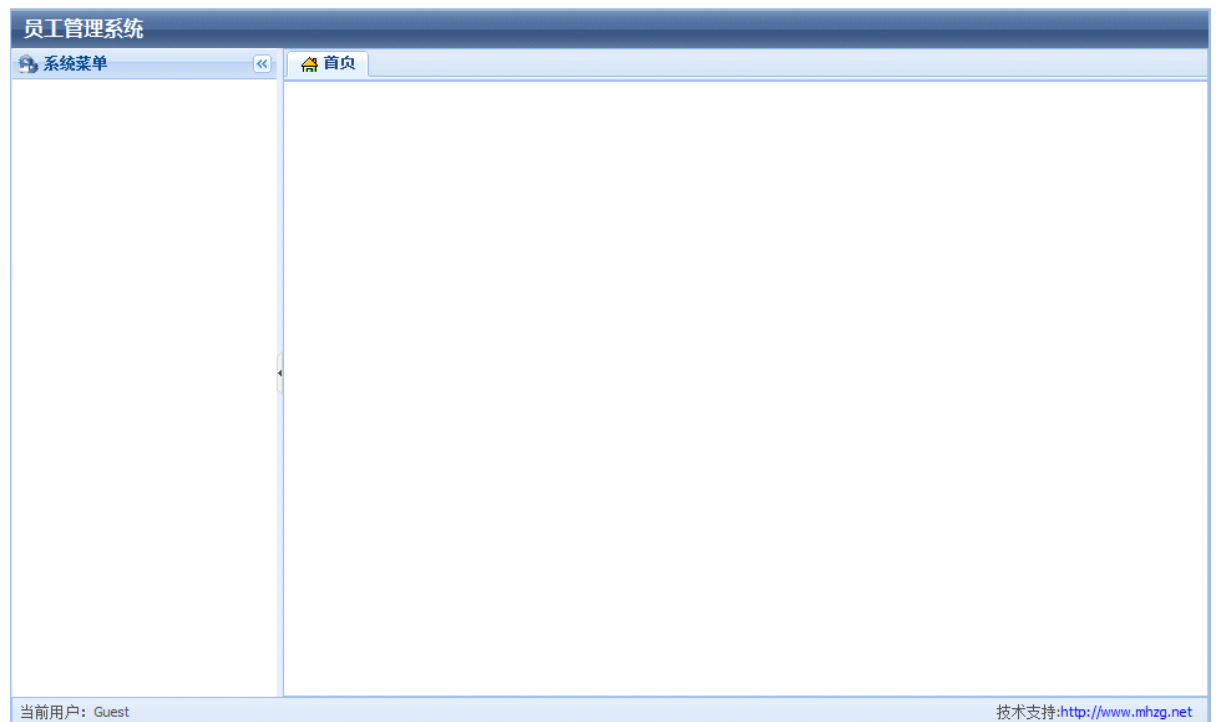
而头部、菜单、内容区及底部则完全分离成 4 个 JS 文件，我们将先实现这几个文件的基础功能，之后我们会慢慢完善这些组件。而整个页面的填充，也使用一个 JS 文件来完成。也许有人会问，这么多文件，是不是要都在 `index.html` 中引入啊。这样想的话，就错了哦。因为我们使用的是 Extjs4，所以我们一定要使用 **Extjs4 动态加载** 功能。

先来看下自定义 CSS ( sytle.css ) :

1. #header { background: #7F99BE url(/images/layout-browser-hd-bg.gif) repeat-x center;}
2. #header h1 {font-size: 16px;color: #fff;padding: 3px 10px; font-family:"微软雅黑","黑体"}
- 3.
4. .tabs{}
5. .tabs{background-image: url(../images/menuPanel/bulletin\_manager.gif) !important;}
6. .manage{background-image: url(../images/menuPanel/admin.gif) !important;}
7. .home{background-image: url(../images/home.gif) !important; line-height:30px;}
8. .icon-menu{background-image: url(../images/menuPanel/sys.gif) !important;}

图片文件夹就不放上来了。从以前的项目中拷贝了一些比较靠谱的图片，大家完全可以自己去下载一些 ICON 图标文件而为己所用。

搭建的框架是经典的 EXTJS 布局模式，如图所示：



首先，我们建立 index.html 和 app.js，index.html 代码为：

```
1. <!DOCTYPE html PUBLIC "-
   //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5. <title>员工管理系统</title>
6. <link rel="stylesheet" type="text/css" href="extjs4/resources/css/ext-
   all.css" />
```

```

7. <!--引入自定义 CSS-->
8. <link rel="stylesheet" type="text/css" href="extjs4/resources/style.css" />
9. <script type="text/javascript" src="extjs4/ext-all-debug.js"></script>
10. <script type="text/javascript" src="app.js"></script>
11. </head>
12.
13. <body>
14. </body>
15. </html>

```

App.js :

```

1. Ext.Loader.setConfig({enabled: true});
2. Ext.application({
3.     name: 'SMS',
4.     appFolder: 'app',
5.     controllers: [
6.         'Main'
7.     ]
8. });

```

稍做解释：

Ext.Loader.setConfig({enabled: true}); //意思是开启 Ext.Loader。  
Ext.Loader 是动态加载的核心哦。。

Ext.application({...}); 看字面意思吧，不解释。

配置中的 name，我理解为是 Extjs3.x、Extjs2.x 中的命名空间。

appFolder，应用文件夹名字。

controllers，控制单元的名字，这里我们定义为 Main。那么根据 Extjs4 动态加载的要求，我们需要在 /app/controllers 文件夹下建立 Main.js 文件，作为控制单元。有关 Extjs4 动态加载机制，请参考：[www.mhzg.net/a/20117/2011721040290.html](http://www.mhzg.net/a/20117/2011721040290.html)

Main.js :

```

1. Ext.define('SMS.controller.Main',{
2.     extend: 'Ext.app.Controller',
3.     init : function(){
4.     }
5. });

```

这里的 Main.js 中只是定义了 Main 这个类，且继承了 Ext.app.Controller，其余都没有写。看到这里，会有人很奇怪了，index.html 中引入了 app.js，而 app.js 只是创建了 Main 这个类，但 Main.js 什么都没有，那么页面中为什么会显示出框架页面呢？这也是最多人所疑惑的。下面我来解释下这个问题。所有的原因就在于 app.js 这个文件中，app.js 文件定义了 Ext.application。而 Ext.application 中有个属性是 `autoCreateViewport`，这个属性是 Boolean 类型，如果值为 true，那么 Extjs4 会自

动加载 view/Viewport.js 文件，如果值为 false，那么必须要自己去创建一个 View，这就是为什么 app.js 和 Main.js 文件都没有写相关代码，也会有界面出现。

整理下思路，由于 Extjs4 自动加载了 view/Viewport.js，而 Viewport.js 文件包含了头部、菜单、内容区及底部这 4 个组件，那么我们必须先完成这 4 个文件的编写，同样，由于这 4 个文件是界面型的，我们将这 4 个文件都放到 view 文件夹下。

view 文件夹下共 5 个 JS 文件，分别为：

Header.js、Menu.js、South.js、TabPanel.js 及 Viewport.js

这 5 个 js 文件的作用，我们一一介绍。

5 个 js 文件都包含了 Ext.applyIf、callParent。由于篇幅问题，Ext.applyIf、callParent 等方法，请参考 Extjs4 相关 API。

Header.js：这个是头部，也就是深蓝色底子。白色字体，那块，上面写着员工管理系统。

代码为：

```
1. Ext.define('SMS.view.Header', {
2.     extend: 'Ext.Component',
3.     initComponents: function() {
4.         Ext.applyIf(this, {
5.             xtype: 'box',
6.             cls: 'header',
7.             region: 'north',
8.             html: '<h1>员工管理系统</h1>',
9.             height: 30
10.        });
11.        this.callParent(arguments);
12.    }
13. });
```

Menu.js:

```
1. Ext.define('SMS.view.Menu',{
2.     extend: 'Ext.tree.Panel',
3.     initComponents : function(){
4.         Ext.apply(this,{
5.             id: 'menu-panel',
6.             title: '系统菜单',
7.             iconCls: 'icon-menu',
8.             margins : '0 0 -1 1',
9.             region: 'west',
10.            border : false,
11.            enableDD : false,
12.            split: true,
13.            width : 212,
14.            minSize : 130,
15.            maxSize : 300,
16.            rootVisible: false,
```

```

17.         containerScroll : true,
18.         collapsible : true,
19.         autoScroll: false
20.     });
21.     this.callParent(arguments);
22. }
23. })

```

TreePanel 并没有加载菜单项，关于 Extjs4 Tree，我们后面会介绍。

TabPanel.js :

```

1. Ext.define('SMS.view.TabPanel',{
2.     extend: 'Ext.tab.Panel',
3.     initComponent : function(){
4.         Ext.apply(this,{
5.             id: 'content-panel',
6.             region: 'center',
7.             defaults: {
8.                 autoScroll:true,
9.                 bodyPadding: 10
10.            },
11.            activeTab: 0,
12.            border: false,
13.            //plain: true,
14.            items: [{
15.                id: 'HomePage',
16.                title: '首页',
17.                iconCls: 'home',
18.                layout: 'fit'
19.            }]
20.        });
21.        this.callParent(arguments);
22.    }
23. })

```

South.js :

```

1. Ext.define('SMS.view.South',{
2.     extend: 'Ext.Toolbar',
3.     initComponent : function(){
4.         Ext.apply(this,{
5.             id:"bottom",
6.             //frame:true,
7.             region:"south",
8.             height:23,
9.             items:["当前用户 : Guest", '->', "技术支持:
持:<a href='http://www.mhzg.net' target='_blank' style='text-
decoration:none;'><font color='#0000FF'>http://www.mhzg.net</font></a>&nbsp;&
nbsp;"]
10.        });
11.        this.callParent(arguments);
12.    }
13. })

```

文件都创建好了。我们进行最后一部，布局。

Viewport.js :

```

1. Ext.define('SMS.view.Viewport',{
2.     extend: 'Ext.Viewport',
3.     layout: 'fit',
4.     hideBorders: true,
5.     requires : [
6.         'SMS.view.Header',
7.         'SMS.view.Menu',
8.         'SMS.view.TabPanel',
9.         'SMS.view.South'
10.    ],
11.    initComponents : function(){
12.        var me = this;
13.        Ext.apply(me, {
14.            items: [{
15.                id: 'desk',
16.                layout: 'border',
17.                items: [
18.                    Ext.create('SMS.view.Header'),
19.                    Ext.create('SMS.view.Menu'),
20.                    Ext.create('SMS.view.TabPanel'),
21.                    Ext.create('SMS.view.South')
22.                ]
23.            }]
24.        });
25.        me.callParent(arguments);
26.    }
27. })

```

重点：requires 属性，这个我理解为创建引用。稍懂编程语言的人应该都明白。但是光引用还不够，我们还需要去实例化它，也就是 Ext.create。至此，我们的框架已经顺利搭建完毕。

今天的工作也就是这么多，搭建完框架之后，会慢慢丰富整个系统。本来想连菜单的树也完成，最后想了想，这工作还是留到明天吧。因为树涉及到了异步获取，需要有服务端程序，今天弄好框架之后，把服务端代码写好了，明天来完成这棵树的实现吧。

需要注意的一点，在 extjs4 中，只要定义了布局为 border，那么他的 items 中必须要有 **region: 'center'** 的组件，否则将会提示错误。貌似在 extjs3.x 甚至是以前的版本都没发现有这样的要求，因为这个，费了老大的劲才调整过来，再一看，代码全部变了，已经跟 extjs3.x 的风格完全不同了。令人欣喜的是，现在的代码完全符合 extjs4 的风格，也完全符合我的预期。

最近有网友跟我反映，说按照文章内容照着完成后，不显示框架，只有空白一片，问了几位网友下，发现他们都是使用的 Extjs4.02a 的版本，最后，在官方最新的 API 中发现，Ext.application 下的属性 autoCreateViewport 默认变成了 false，如此一来，就不能自动创建 Viewport 了。这里提供下解决办法。

如果你的 Extjs4 版本为 Extjs4.02a 以下版本，则不需要任何改动。完全可以显示完整框架。

如果你的 Extjs4 版本为 Extjs4.02a 或以上版本，则需要修改 app.js 文件，其内容为：

```

1. Ext.Loader.setConfig({enabled: true});
2. Ext.application({

```

```
3.     name: 'SMS',
4.     autoCreateViewport: true,
5.     appFolder: 'app',
6.     controllers: [
7.         'Main'
8.     ]
9. });
```

注意，此 `app.js` 加入了 `autoCreateViewport: true` 设置。

## 5.3 菜单的实现

上篇文章介绍了搭建一个空的框架，使得管理系统有了大致的模样，今天工作的主要内容就是菜单的实现以及点击菜单后在右边内容区打开一个新的 Panel。本篇文章的内容主要包括两个方面，Extjs4 Tree 及 Extjs4 tabPanel。

在 Extjs 应用中实现菜单，无疑 Tree 是最好的选择，将菜单项直接写到 Tree 中，也未尝不可，但后期的维护会非常麻烦，那么最好的选择就是异步获取菜单节点，这样既有利于后期维护，也可以节省 JS 代码的编写量。

实现异步加载，那必须要有数据库和服务端程序。管理系统使用的是 ACCESS 数据库。在数据库中建立 Menu 表，表一共有 4 个字段，分别是 ID、MenuName、ParentID 和 cls。

ID：自增长类型

MenuName：代表菜单的名字。ParentID

ParentID：父 ID

cls：样式名（这个需要在样式表中体现出来，才会有效果）

数据库有了，接下来就是服务端的编写，其实 JS 框架的好处在于服务端无论用什么都可以，这里我就使用 ASP 来实现了。由于服务端涉及的方面比较多，较多代码帖出来会比较乱，这里只说下返回的形式。

菜单需要的 JSON 数据格式如下：

```
1. [{ "text": "管理员管理", "id": "1", "iconCls": "manage", "leaf": true }]
```

这里注意一点：由于整棵菜单树都是异步获取的，所以节点并不需要递归，而树的 `node.id` 就是 JSON 数据中的 ID。点击父节点展开子节点的时候，发送的数据也是 `node.id`，这样正好解决获取子节点的问题。所有根节点的 ParentID 都为 0。那么第一次加载菜单的时候，正好可以获取所有的根节点了。

重点，菜单树的数据源编写，根据 MVC 原则，在 app 目录下建立 store 文件夹，这个文件夹下放置所有的获取数据的 js 文件，在 store 文件夹下建立 Menus.js，编写如下代码：



```

1. Ext.define('SMS.store.Menu',{
2.     extend: 'Ext.data.TreeStore',
3.     root: {
4.         expanded: true
5.     },
6.     proxy: {
7.         type: 'ajax',
8.         url: '/server/MenuLoader.asp'
9.     }
10.
11. })

```

然后修改 view 文件夹下的 Menu.js 文件：

```

1. Ext.define('SMS.view.Menu',{
2.     extend: 'Ext.tree.Panel',
3.     alias: 'widget.smsmenu',
4.     requires:['SMS.store.Menu'],
5.     initComponents : function(){
6.         Ext.apply(this,{
7.             id: 'menu-panel',
8.             title: '系统菜单',
9.             iconCls:'icon-menu',
10.            margins : '0 0 -1 1',
11.            region:'west',
12.            border : false,
13.            enableDD : false,
14.            split: true,
15.            width : 212,
16.            minSize : 130,
17.            maxSize : 300,
18.            rootVisible: false,
19.            containerScroll : true,
20.            collapsible : true,
21.            autoScroll: false,
22.            store:Ext.create('SMS.store.Menu'),
23.        });
24.        this.callParent(arguments);
25.    }
26. })

```

如此，当页面打开时，就会自动加载菜单项了。但是目前来说，点击菜单的任何节点都没有任何作用，那么由于整个项目都要使用 Extjs 来完成，那么必须要实现点击节点在右边内容区显示相应的 Grid 或 Panel 等等。下面，我们编写代码实现这个功能。

原理：当点击菜单树的节点时，先要判断要打开的组件是否存在，如果存在，则在右边内容区激活当前组件，如果不存在，则创建一个组件，然后在右边内容区增加一个组件。当然。这里为了通用性更高，创建出的组件一律按 panel 为准。为了实现在右边内容区的 tabPanel 上增加或删除对应的 table，根据分离原则，我们需要在控制器上完成该操作，接下来，我们在 controller 文件夹下建立 Menu.js 文件，Menu.js 会完成这一系列的工作，具体代码如下：

```

1. Ext.define('SMS.controller.Menu',{
2.     extend: 'Ext.app.Controller',
3.     refs:[

```

```

4.         {ref: 'smsmenu',selector: 'smstablepanel'},
5.         {ref: 'tabPanel',selector: 'smstablepanel'}
6.     ],
7.     init:function(){
8.         this.control({
9.             'smsmenu': {
10.                 itemmousedown: this.loadMenu
11.             }
12.         })
13.     },
14.     loadMenu:function(selModel, record){
15.         if (record.get('leaf')) {
16.             var panel = Ext.getCmp(record.get('id'));
17.             if(!panel){
18.                 panel = {
19.                     title: 'New Tab ' + record.get('id'),
20.                     iconCls: 'tabs',
21.                     html: 'Tab Body ' + record.get('id') + '<br/><br/>',
22.                     closable: true
23.                 }
24.                 this.openTab(panel,record.get('id'));
25.             }else{
26.                 var main = Ext.getCmp("content-panel");
27.                 main.setActiveTab(panel);
28.             }
29.         }
30.     },
31.     openTab : function (panel,id){
32.         var o = (typeof panel == "string" ? panel : id || panel.id);
33.         var main = Ext.getCmp("content-panel");
34.         var tab = main.getComponent(o);
35.         if (tab) {
36.             main.setActiveTab(tab);
37.         } else if(typeof panel!="string"){
38.             panel.id = o;
39.             var p = main.add(panel);
40.             main.setActiveTab(p);
41.         }
42.     }
43. }
44.
45. })

```

关键点：refs 和 this.control，refs 在官方 API 中没有找到其解释，网上查了下，对该属性的解释是：凡是 component 都可以使用该属性在它的归属容器及归属容器的父节点中注入一个对该属性的引用名称。有了该引用名，和该组件有共同父节点的组件就可以比较方便的引用该组件。

而 this.control 则很容易理解了，即通过 Ext.ComponentQuery 为选定的组件添加监听。上面代码为我们的菜单节点添加了一个事件 loadMenu，loadMenu 中，先获取对应的 panel，如果该 panel 不存在，则使用 openTab 方法在内容区的 tabPanel 上增加一个 panel，如果存在，则激活该 panel。而 panel 不存在的话，这里只简单的创建了一个 panel，并没有任何意义，下篇文章，我们将详细讨论这个问题。

最后，我们需要修改 app.js，将我们创建的控制加载进来。app.js：

```

1. Ext.Loader.setConfig({enabled: true});
2. Ext.application({

```

```
3.     name: 'SMS',
4.     appFolder: 'app',
5.     autoCreateViewport:true,
6.     controllers: [
7.         'Menu'
8.     ]
9. });
```

这里唯一修改的地方就是将 Main 改为了 Menu。后面，我们会逐步完善。

今天的工作比较多，整理下今日的工作内容：

1、在 app 目录下建立了 store 文件夹，并建立了 Menus.js 文件，这个文件负责菜单数据的加载。

2、修改了 view 文件夹下的 Menu.js,使其可以加载菜单数据。

3、在 controller 文件夹下建立了 Menu.js，使其可以在内容区的 tabPanel 上增加新的 panel 组件。

4、修改 app.js，使其可以使菜单项正常使用。

## 5.4 实现登录

上篇文章介绍了如何实现菜单功能（[点击查看](#)），但是有个问题，就是管理系统必须是登录后才会显示菜单，而且菜单还要实现不同权限有不同的菜单项，本文将实现这个功能。

首先，将 server/MenuLoader.asp 修改，增加管理员验证功能。即

```
1. If Session("Manage") <> "" Then
2. '显示菜单项
3. End If
```

这时，重新打开页面，由于有了基本的管理员验证，菜单不显示了。



接下来，开始制作登录，在 view 文件夹下建立 Login.js，checkcode.js，其中 Login.js 实现登录功能，有用户名、密码和验证码，验证码的实现，就是 checkcode.js，由于篇幅问题，checkcode.js 请查看本站另一篇文章，[ExtJS4 学习笔记\(十\)---ExtJS4 图片验证码的实现](#)。

主要是 Login.js：

```
1. Ext.define(SMS.view.Login',{
2.     extend:'Ext.window.Window',
3.     alias: 'widget.loginForm',
4.     requires: ['Ext.form.*','SMS.view.CheckCode'],
5.     initComponents:function(){
6.         var checkcode = Ext.create('SMS.view.CheckCode',{
7.             cls : 'key',
8.             fieldLabel : '验证码',
9.             name : 'CheckCode',
10.            id : 'CheckCode',
11.            allowBlank : false,
12.            isLoader:true,
13.            blankText : '验证码不能为空',
14.            codeUrl: '/include/checkCode.asp',
15.            width : 160
16.        })
17.        var form = Ext.widget('form',{
18.            border: false,
19.            bodyPadding: 10,
20.            fieldDefaults: {
21.                labelAlign: 'left',
22.                labelWidth: 55,
23.                labelStyle: 'font-weight:bold'
24.            },
25.            defaults: {
26.                margins: '0 0 10 0'
27.            },
28.            items:[{
29.                xtype: 'textfield',
30.                fieldLabel: '用户名',
```

```

31.         blankText : '用户名不能为空',
32.         name: 'UserName',
33.         id: 'UserName',
34.         allowBlank: false,
35.         width: 240
36.     }, {
37.         xtype: 'textfield',
38.         fieldLabel: '密 码',
39.         allowBlank: false,
40.         blankText : '密码不能为空',
41.         name: 'PassWord',
42.         id: 'PassWord',
43.         width: 240,
44.         inputType : 'password'
45.     }, checkcode],
46.     buttons: [{
47.         text: '登录',
48.         handler: function() {
49.             var form = this.up('form').getForm();
50.             var win = this.up('window');
51.             if (form.isValid()) {
52.                 form.submit({
53.                     clientValidation: true,
54.                     waitMsg: '请稍后',
55.                     waitTitle: '正在验证登录',
56.                     url: '/server/checklogin.asp',
57.                     success: function(form, action) {
58.                         // 登录成功后。
59.                         // 隐藏登录窗口，并重新加载菜单
60.                         win.hide();
61.                         Ext.getCmp('SystemMenus').store.load();
62.                     },
63.                     failure: function(form, action) {
64.                         Ext.MessageBox.show({
65.                             width: 150,
66.                             title: "登录失败",
67.                             buttons: Ext.MessageBox.OK,
68.                             msg: action.result.msg
69.                         });
70.                     }
71.                 });
72.             });
73.         }
74.     }
75.     ]
76. })
77. Ext.apply(this, {
78.     height: 160,
79.     width: 280,
80.     title: '用户登陆',
81.     closeAction: 'hide',
82.     closable : false,
83.     iconCls: 'win',

```

```

84.         layout: 'fit',
85.         modal : true,
86.         plain : true,
87.         resizable: false,
88.         items:form
89.     });
90.     this.callParent(arguments);
91. }
92. });

```

最终效果：



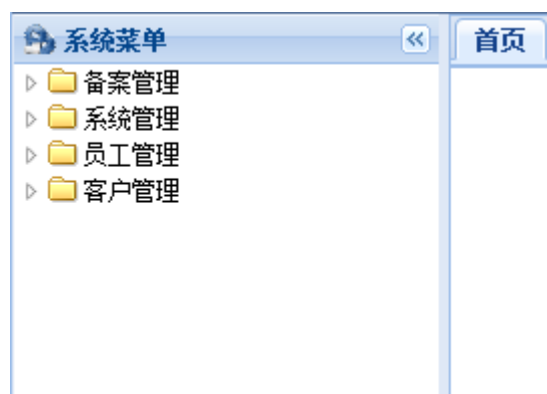
修改 controller 目录下的 Main.js:

```

1. Ext.define(SMS.controller.Main',{
2.     extend: 'Ext.app.Controller',
3.     requires:['SMS.view.Login'],
4.
5.     onLaunch : function(){
6.         var win;
7.         if(!win){
8.             win = Ext.create('SMS.view.Login').show();
9.         }
10.    }
11. })

```

这时，当页面加载的时候，会显示登录窗口，而登录成功后，会隐藏登录窗口并加载菜单。最后附上登录成功后页面效果。



## 5.5 动态 grid

上一节我们说了如何实现登录，现在为止，框架有了，菜单有了。下一步工作就是实现每个菜单项目的数据显示，一般来讲，我们都是固定思维，通过编写大量的 grid 来实现各种数据的显示，在网络上有一些案例，都是每一个菜单项目，写一个 grid，而且点击树节点，都是通过判断或其他方法来创建相应 grid 的实例，例如：

```
1. if(node.id=="depmanage"){
2.     panel = new depManagePanel();
3.     main.openTab(panel);
4. }else if(node.id=="telmanage"){
5.     panel = new telManagePanel();
6.     main.openTab(panel);
7. }else if(node.id=="usermanage"){
8.     panel = new userManagePanel();
9.     main.openTab(panel);
10. }else if(node.id=="userlog"){
11.     panel = new operationPanel();
12.     main.openTab(panel);
13. }else{....}}
```

但是在很多优秀的案例中，也不排除动态生成 grid 的可能，至少目前为止，我没见过此种案例，可能是孤陋寡闻吧。。

在这个项目的开发过程中，我不断思考，是否能动态生成 grid，使得工作一劳永逸。在查阅一些资料和自己动手尝试后，发现这样的办法可行，但前提是，需要服务端支持。单纯从开发角度来讲，我感觉这样的方法更加适合实际的开发。

在本节开始前，我们先来看一个问题。

树节点服务端返回的 json 数据：看了 API 之后，发现树的 store 返回 json 数据后，如果不指定其 fields，那么再整个树的节点集合中，只有固定的几种属性值，即 text, id、iconCls 和 leaf（注意：这里只是例举了这几种常见的属性值，至于其他的属性值，还是留给大家去探究吧）。所以大多数网友问我，为什么自定义的属性值，在 extjs4.0 中使用 record.get("xxxx") 为什么获取不到？这其中的原因就是 store 中没有指定 fields。也就是说，在本项目中，如果菜单树不指定 fields，那么是无法往下进行开发的。

回到正题，要实现动态创建 grid，我们先要可以让树有很多属性供我们所用，新建 Menu.js，将这个 js 文件放置到 app/model 文件夹下。其代码为：

```
1. Ext.define('SMS.model.Menu', {
2.     extend: 'Ext.data.Model',
3.     fields: ['id', 'text', 'iconCls', 'stores', 'columns'],
4.     root: {
5.         expanded: true
6.     },
```

```

7.  proxy: {
8.    type: 'ajax',
9.    url: '/server/MenuLoader.asp'
10. }
11. })
12.

```

MenuLoader.asp 返回如下 json :

```

1.  [{"text": "备案列表
    ", "id": "4", "iconCls": "Folder", "stores": "bastore", "columns": [{text: '序号
    ', dataIndex: 'ID'}, {text: '公司名称
    ', dataIndex: 'kehu_name'}], "leaf": true}, {"text": "新增备案
    ", "id": "5", "iconCls": "Folder", "stores": "", "columns": [], "leaf": true}]
2.

```

修改 app/store 下的 Menus.js , 更改为如下代码 :

```

1.  Ext.define('SMS.store.Menus',{
2.    extend: 'Ext.data.TreeStore',
3.    requires: 'SMS.model.Menu',
4.    model: 'SMS.model.Menu'
5.  })

```

新建 bastore.js , 将其放到 app/store 目录下, 代码为 :

```

1.  Ext.define('SMS.store.bastore', {
2.    extend: 'Ext.data.Store',
3.    requires: 'SMS.model.beianlistmodel',
4.    model: 'SMS.model.beianlistmodel'
5.  });
6.
7.
8.

```

新建 beianlistmodel.js , 将其放到 app/model 目录下, 其代码为 :

```

1.  Ext.define('SMS.model.beianlistmodel', {
2.    extend: 'Ext.data.Model',
3.    fields: ['ID', 'kehu_name']//,这里只列出了自增值 id 和客户名称。
4.
5.    //proxy: {
6.    //      type: 'ajax',
7.    //      url: '/server/getbeianlist.asp',
8.    //      reader: {
9.    //        type: 'json',
10.    //        root: 'results'
11.    //      }
12.    //    }
13.  });

```

服务端代码没有编写, 所以将获取数据的代码暂时注释掉。



接下来，我们修改 app/controller 目录下的 Menu.js，增加一个 stores 属性，其代码为

```
stores: ['bastore'],
```

然后将测试代码

```
1.  if (record.get('leaf')) {
2.      var panel = Ext.getCmp(record.get('id'));
3.      if(!panel){
4.          panel = {
5.              title: 'New Tab ' + record.get('id'),
6.              iconCls: 'tabs',
7.              html: 'Tab Body ' + record.get('id') + '<br/><br/>',
8.              closable: true
9.          }
10.         this.openTab(panel,record.get('id'));
11.     }else{
12.         var main = Ext.getCmp("content-panel");
13.         main.setActiveTab(panel);
14.     }
15. }
16.
```

全部删除，重新写如下代码：

```
1.  if (record.get('leaf')) {
2.      var panel = Ext.getCmp(record.get('id'));
3.      if(!panel){
4.          panel = Ext.create("Ext.grid.Panel",{
5.              store:record.get('stores'),
6.              columns:record.get('columns'),
7.              title: record.get('text'),
8.              id:record.get('text')+record.get('id'),
9.              viewConfig: {
10.                  stripeRows: true
11.              },
12.              closable: true
13.          })
14.          this.openTab(panel,record.get('id'));
15.      }else{
16.          var main = Ext.getCmp("content-panel");
17.          main.setActiveTab(panel);
18.      }
19.  }
20.
```

稍做说明：Ext.create("Ext.grid.Panel"..... 首先创建了一个 gridpanel，由于我们增加了菜单树的 fields 属性，那么我们可以获取所有 fields 指定的属性值，通过上面的 json 数据

```
{"text": "备案列表"
```

```
", "id": "4", "iconCls": "Folder", "stores": "bastore", "columns": [{text: '序号', dataIndex: 'ID'}, {text: '公司名称', dataIndex: 'kehu_name'}], "leaf": true}
```

我们就实现了动态生成 grid。app/controller 目录下的 Menu.js 全部代码为：

```

1. Ext.define('SMS.controller.Menu',{
2.   extend: 'Ext.app.Controller',
3.   refs:[
4.     {ref: 'smsmenu',selector: 'smstablepanel'},
5.     {ref: 'tabPanel',selector:'smstablepanel'}
6.   ],
7.   stores: ['bastore'],
8.   init:function(){
9.     this.control({
10.      'smsmenu': {
11.        itemmousedown: this.loadMenu
12.      }
13.    })
14.  },
15.  loadMenu:function(selModel, record){
16.    if (record.get('leaf')) {
17.      var panel = Ext.getCmp(record.get('id'));
18.      if(!panel){
19.        panel = Ext.create("Ext.grid.Panel",{
20.          store:record.get('stores'),
21.          columns:record.get('columns'),
22.          title: record.get('text'),
23.          id:record.get('text')+record.get('id'),
24.          viewConfig: {
25.            stripeRows: true
26.          },
27.          closable: true
28.        })
29.        this.openTab(panel,record.get('id'));
30.      }else{
31.        var main = Ext.getCmp("content-pane");
32.        main.setActiveTab(panel);
33.      }
34.    }
35.  },
36.  },
37.  openTab : function (panel,id){
38.    var o = (typeof panel == "string" ? panel : id || panel.id);
39.    var main = Ext.getCmp("content-pane");
40.    var tab = main.getComponent(o);
41.    if (tab) {
42.      main.setActiveTab(tab);
43.    } else if(typeof panel!="string"){
44.      panel.id = o;
45.      var p = main.add(panel);
46.      main.setActiveTab(p);
47.    }
48.  }
49. })
50.
51. })
52.

```

声明，进行一个 Extjs4.0MVC 模式开发，是一个尝试的过程，随着 Extjs4.x 的不断升级、改进和修复，可能会造成当前项目无法运行或者报错，mhzg 会持续关注 extjs 的最新动态，如果有大幅改动，mhzg.net 会将最新的 extjs4 版本下载到本地进行测试，如果本项目无法运行，那么 mhzg.net 会进行修复并在 mhzg.net 上发布补充文档进行说明和修正。而在整个开发过程中，难免会有一些地方不尽人意，欢迎大家相互探讨、研究，共

同进步，extjs4 开发笔记将在第六章更名为 extjs4 MVC 开发笔记，请大家持续关注，谢谢！！

## 5.6 数据的增删改

上一章，我们介绍了动态 Grid 的显示，其地址是：[Extjs4 开发笔记\(五\)—动态 grid](#)，在上一章，我们只做了数据的显示，并没有添加、删除和修改功能，本章内容，介绍如何进行添加、删除和修改。

一般的项目中，Grid 功能不是很复杂的话，都会使用 window 来实现数据的添加、修改功能，而本实例中，由于使用了动态 grid 功能，这样就使得再使用动态 window 来实现数据的添加和修改就会变的非常困难。

幸好，Grid 有 RowEditing，下面，我们就用 RowEditing 来实现数据的添加和修改功能。在开始之前，我们先回顾下上一章的一些重点内容：

1、给 Menu 增加了 field 属性，使得我们在数据库中的一些字段可以被当做是 Menu 的节点集合中的对象来调用。

2、给菜单表增加了两个字段，分别是 store 和 columns。在 app/store/文件夹下，我们新建了 bastore.js 文件，那么再数据库中对应的字段中，填写内容为 bastore，在 columns 字段中，我们添加了内容为 {text: '序号', dataIndex: 'ID'}, {text: '公司名称', dataIndex: 'kehu\_name'} 的数据。

最后，我们修改了 Menu.js 文件，使得 Grid 可以显示数据。

现在，我们修改 columns 中的数据为：

```
1. {text: '序号', dataIndex: 'ID', width: 50}, {text: '公司名称',  
  ', dataIndex: 'kehu_name', width: 260, editor: {allowBlank: false}}, {text: '备案号',  
  ', dataIndex: 'beianhao', width: 140, editor: {allowBlank: false}}, {text: '备案密码',  
  ', dataIndex: 'beianpass', width: 100, editor: {allowBlank: false}}, {text: '备案邮箱',  
  ', dataIndex: 'beianemail', width: 160, editor: {allowBlank: false}}, {text: '备案邮  
箱密码', dataIndex: 'emailpass', width: 120, editor: {allowBlank: false}}, {text: '备  
案账号', dataIndex: 'beianzh', width: 160, editor: {allowBlank: false}}, {text: '账号  
密码', dataIndex: 'beianzhpa', width: 120, editor: {allowBlank: false}}
```

在这些数据中，将所有字段都列了出来，并且制定了 editor 中 allowBlank 的数据位 false，就是说，这些内容必须填写。

接下来，我们需要修改 app/controller 下的 menu.js 文件，增加一些功能，使其可以添加、删除信息。修改内容如下：

```

1.  if (record.get('leaf')) {
2.      var panel = Ext.getCmp(record.get('id'));
3.      if(!panel){
4.          Ext.require(['Ext.grid.*']);
5.          var rowEditing = Ext.create('Ext.grid.plugin.RowEditing',{
6.              clicksToMoveEditor: 1,
7.              autoCancel: true
8.          });
9.          panel = Ext.create("Ext.grid.Panel",{
10.              store:record.get('stores'),
11.              columns:record.get('columns'),
12.              errorSummary:false,
13.              title: record.get('text'),
14.              id:record.get('text')+record.get('id'),
15.              columnLines: true,
16.              bodyPadding:0,
17.              closable: true,
18.              bbar: Ext.create('Ext.PagingToolbar', {
19.                  store: record.get('stores'),
20.                  displayInfo: true,
21.                  displayMsg: '显示 {0} - {1} 条, 共计 {2} 条',
22.                  emptyMsg: "没有数据"
23.              }),
24.              dockedItems: [{
25.                  xtype: 'toolbar',
26.                  items: [{
27.                      text: '增加信息',
28.                      iconCls: 'icon-add',
29.                      handler: function(){
30.                          rowEditing.cancelEdit();
31.                          var panelStore = this.up("panel").getStore();
32.                          var panelModel = this.up("panel").getSelecti
onModel();
33.                          panelStore.insert(0,panelModel);
34.                          rowEditing.startEdit(0, 0);
35.                      }
36.                  }, '-', {
37.                      itemId: 'delete',
38.                      text: '删除信息',
39.                      iconCls: 'icon-delete',
40.                      disabled: true,
41.                      handler: function(){
42.                          var selection = panel.getView().getSelection
Model().getSelection()[0];
43.                          var panelStore = this.up("panel").getStore();
44.                          if (selection) {
45.                              panelStore.remove(selection);
46.                          }
47.                      }
48.                  }, '-', {
49.                      text: '刷新数据',
50.                      iconCls: 'icon-refresh',
51.                      handler: function(){
52.                          var panelStore = this.up("panel").getStore();
53.                          var currPage = panelStore.currentPage;
54.                          panelStore.removeAll();
55.                          panelStore.load(currPage);
56.                      }
57.                  }]
58.              }],
59.              plugins: [rowEditing],
60.              listeners: {
61.                  'selectionchange': function(view, records) {

```



```

32.         },
33.         success: function(response){
34.             store.removeAll();
35.             store.load(currPage);
36.         }
37.     });
38. },
39.     remove: function(store, record){
40.         var currPage = store.currentPage;
41.         //alert(record.get("ID"))
42.         Ext.Ajax.request({
43.             url: '/server/getbeian.asp?action=del',
44.             params: {
45.                 id : record.get("ID")
46.             },
47.             success: function(response){
48.                 store.removeAll();
49.                 store.load(currPage);
50.             }
51.         });
52.     }
53. },
54.     sorters: [{
55.         property: 'ID',
56.         direction: 'DESC'
57.     }]
58. });

```

代码中，为 store 增加了两个监听，update 和 remove，并且将数据通过 AJAX 发送到服务端，在服务端进行处理，这里，只使用了 update 和 remove。store 中还有个 add 方法，此方法也是增加一条数据，按照常理来说。这个方法才是增加数据，但是我使用了 add 方法之后，点击添加信息，就会添加一条空数据，索性就使用 update 方法，将 id 值也发送到服务端，在服务端进行处理，服务端处理流程是：接收 id 值，判断 id 值是否为空，如果为空，则新增数据，如果不为空，则更新数据。

至此，一个 grid 的功能全部完成了。而且本项目所有的功能，都是如此，这样，只要再数据库中插入相应的行，在 app/store 和 app/model 中建立相关的 js 就可以了。至于其他功能，就不在此——例举了。

声明一点，这个开发笔记实施到最后，有一些东西已经脱离了 MVC 的范畴，而且也没想到六章内容就结束了这个项目。从文章一到六，只是起一个抛砖引玉的作用。由于 Extjs4 有很大的变动，所以任何基于 Extjs4.x MVC 的项目，都是摸着石头过河，一点一点积累起来的，并不是说我的这些文章起到了指导性的作用，而是实际开发过程中的一些体会和经验。所有项目，有很多种方法可以实现需求。