EXT 中文手册

开发必备

该文档内容取自互联网，详细的介绍了 EXT 的各部分内容，希望对大家有所帮助

pjq

2008-5-25

EXT 中文手册

前　言

本手册所有内容均粘贴自互联网，如有错误，请多见谅。

目　　录

EXT 中文手册 1
EXT 简介 5
目 錄 5
下载 Ext 6
开始！ 6
Element：Ext 的核心 6
获取多个 DOM 的节点 7
响应事件 7
使用 Widgets 9
使用 Ajax 11
EXT 源码概述 13
揭示源代码 13
发布 Ext 源码时的一些细节 14
我应该从哪里开始？ 15
适配器 Adapters 15
核心 Core 15
Javascript 中的作用域（scope） 15
事前准备 15
定义 15
正式开始 16
window 对象 16
理解作用域 17
变量的可见度 17
EXT 程序规划入门 18
事前准备 18
需要些什么？ 18
applayout.html 18
applayout.js 19
公开 Public、私有 Private、特权的 Privileged？ 21
重写公共变量 23
重写（Overriding）公共函数 23
DomQuery 基础 24
DomQuery 基础 24
扩展 EXT 组件 31
文件的创建 31
Let's go 35
完成 37
EXT 的布局（Layout） 39
简单的例子 40
加入内容 43
开始使用 Grid 53

步骤一 定义数据(Data Definition) 53
步骤二 列模型（Column Model） 54
Grid 组件的简易分页 55
Grid 数据 55
怎么做一个分页的 Grid 56
分页栏 Toolbar 56
EXT Menu 组件 57
创建简易菜单 57
各种 Item 的类型 59
Item 属性 59
在 UI 中摆放菜单 59
Menu 的分配方式： 60
练一练 62
动态添加菜单按钮到 Toolbar 62
更方便的是 63
下一步是 63
模板（Templates）起步 63
第一步 您的 HTML 模板 63
第二步，将数据加入到模板中 64
下一步 64
学习利用模板（Templates）的格式化功能 64
正式开始 64
下一步 66
事件处理 66
非常基础的例子 66
处理函数的作用域 66
传递参数 67
类设计 67
对象创建 67
使用构造器函数 68
方法共享 68
表单组件入门 69
表单体 69
创建表单字段 69
完成表单 70
下一步 71
为一个表单填充或提交数据 71
让我们开始吧 71
读取我们的数据 72
EXT 中的继承 73
补充资料 74
Ext 2 概述 74
组件模型 Component Model 76
容器模型 Container Model 80

布局 Layouts..... 81
Grid 84
XTemplate 84
DataView..... 85
其它新组件 85
EXT2 简介 86
下载 Ext..... 86
开始！ 87
Element：Ext 的核心 87
获取多个 DOM 的节点 88
响应事件..... 88
使用 Widgets 90
編輯 使用 Ajax 93
TabPanel 基础..... 96
Step 1: 创建 HTML 骨架..... 96
Step 2: Ext 结构的构建..... 97
Step 3: 创建 Tab 控制逻辑..... 99

EXT 简介

无论你是 Ext 库的新手，抑或是想了解 Ext 的人，本篇文章的内容都适合你。本文将简单地介绍 Ext 的几个基本概念，和如何快速地做出一个动态的页面并运行起来，假设读者已具备了一些 JavaScript 经验和初级了解 HTMLDom。

目 錄

- 1 下载 Ext
- 2 开始！
- 3 Element：Ext 的核心
- 4 获取多个 DOM 的节点
- 5 响应事件
- 6 使用 Widgets
  - 6.1 MessageBox
  - 6.2 Grid
  - 6.3 还有更多的..
- 7 使用 Ajax
  - 7.1 PHP
  - 7.2 ASP.Net
  - 7.3 Cold Fusion

下载 Ext

如果你未曾下载过，那应从这里下载最新版本的 Ext http://extjs.com/downloads 。

针对你的下载需求，有几个不同的弹性选项。通常地，最稳定的版本，是较多人的选择。下载解包后，那个 example 文件夹便是一个探索 Ext 的好地方！

开始！

Download Example File

- IntroToExt.zip

我们将使用 Ext，来完成一些 JavaScript 任务。

Zip 文件包括三个文件：**ExtStart.html**，**ExtStart.js** 和 **ExtStart.css**。解包这三个文件到 Ext 的安装目录中（例如，Ext 是在“C:\code\Ext\v1.0”中，那应该在"v1.0"里面新建目录“tutorial”。双击 **ExtStart.htm**，接着你的浏览器打开启动页面，应该会有一条消息告诉你配置已完毕。如果是一个 Javascript 错误，请按照页面上的指引操作。

在你常用的 IDE 中或文本编辑器中，打开 ExtStart.js 看看：

Ext.onReady 可能是你接触的第一个方法。这个方法是指当前 DOM 加载完毕后，保证页面内的所有元素能被 Script 引用（reference）。你可删除 alert()那行，加入一些实际用途的代码试试：

```
Ext.onReady(function() {
        alert("Congratulations!  You have Ext configured correctly!");
});
```

Element：Ext 的核心

大多数的 JavaScript 操作都需要先获取页面上的某个元素（reference）,好让你来做些实质性的事情。传统的 JavaScript 方法，是通过 ID 获取 Dom 节点的：

```
var myDiv = document.getElementById('myDiv');
```

这毫无问题，不过这样单单返回一个对象（DOM 节点），用起来并不是太实用和方便。为了要用那节点干点事情，你还将要手工编写不少的代码；另外，对于不同类型浏览器之间的差异，要你处理起来可真头大了。

进入 Ext.element 对象。元素（element）的的确确是 Ext 的心脏地带，--无论是访问元素（elements）还是完成一些其他动作，都要涉及它。Element 的 API 是整个 Ext 库的基础，如果你时间不多，只是想了解 Ext 中的一两个类的话，Element 一定是首选！

由 ID 获取一个 Ext Element 如下（首页 ExtStart.htm 包含一个 div，ID 名字为“myDiv”，然后，在 ExtStart.js 中加入下列语句）：

```
Ext.onReady(function() {var myDiv = Ext.get('myDiv');});
```

再回头看看 Element 对象，发现什么有趣的东东呢？

- Element 包含了常见的 DOM 方法和属性，提供一个快捷的、统一的、跨浏览器的接口（若使用 Element.dom 的话，就可以直接访问底层 DOM 的节点。）；
- Element.get()方法内置缓存处理（Cache），多次访问同一对象效率上有极大优势；
- 内置常用的 DOM 节点的动作，并且是跨浏览器的定位的位置、大小、动画、拖放等等（add/remove CSS classes, add/remove event handlers, positioning, sizing, animation, drag/drop）。

这意味着你可用少量的代码来做各种各样的事情，这里仅仅是一个简单的例子（完整的列表在 ElementAPI 中）。

继续在 ExtStart.js 中，在刚才我们获取好 myDiv 的位置中加入：

```
myDiv.highlight();        //黄色高亮显示然后渐退
myDiv.addClass('red');   // 添加自定义 CSS 类 (在 ExtStart.css 定义)
myDiv.center();           //在视图中将元素居中
myDiv.setOpacity(.25);   // 使元素半透明
```

获取多个 DOM 的节点

通常情况下，想获取多个 DOM 的节点，难以依靠 ID 的方式来获取。有可能因为没设置 ID,或者你不知道 ID,又或者直接用 ID 方式引用有太多元素了。这种情况下，你就会不用 ID 来作为获取元素的依据，可能会用属性（attribute）或 CSS Classname 代替。基于以上的原因，Ext 引入了一个功能异常强大的 Dom Selector 库，叫做 DomQuery。

DomQuery 可作为单独的库使用，但常用于 Ext，你可以在上下文环境中（Context）获取多个元素，然后通过 Element 接口调用。令人欣喜的是，Element 对象本身便有 Element.selcect 的方法来实现查询，即内部调用 DomQuery 选取元素。这个简单的例子中， ExtStart.htm 包含若干段落（

标签），没有一个是有 ID 的，而你想轻松地通过一次操作马上获取每一段，全体执行它们的动作，可以这样做：

```
// 每段高亮显示
Ext.select('p').highlight();
```

DomQuery 的选取参数是一段较长的数组，其中包括 W3C CSS3 Dom 选取器、基本 XPatch、HTML 属性和更多，请参阅 DomQuery API 文档以了解这功能强大的库个中细节。

响应事件

到这范例为止，我们所写的代码都是放在 onReady 中，即当页面加载后总会立即执行，功能较单一——这样的话，你便知道，如何响应某个动作或事件来执行你希望做的事情，做法是，先分配一个 function，再定义一个 event handler 事件处理器来响应。我们由这个简单的范例开始，打开 ExtStart.js，编辑下列的代

码：

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

加载好页面，代码依然会执行，不过区别是，包含 alert()的 function 是已定义好的，但它不会立即地被执行，是分配到按钮的单击事件中。用浅显的文字解释，就是：获取 ID 为'myDottom'元素的引用，监听任何发生这个元素上被单击的情况，并分配一个 function,以准备任何单击元素的情况。

正路来说，Element.select 也能做同样的事情，即作用在获取一组元素上。下一例中，演示了页面中的某一段落被单击后，便有弹出窗口：

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

这两个例子中，事件处理的 function 均是简单几句，没有函数的名称，这种类型函数称为“匿名函数（anonymous function）”，即是没有名的的函数。你也可以分配一个有名字的 event handler,这对于代码的重用或多个事件很有用。下一例等效于上一例：

```
Ext.onReady(function() {
        var paragraphClicked = function() {
                alert("You clicked a paragraph");
        }
        Ext.select('p').on('click', paragraphClicked);
});
```

到目前为止，我们已经知道如何执行某个动作。但当事件触发时，我们如何得知这个 event handler 执行时是作用在哪一个特定的元素上呢？要明确这一点非常简单，Element.on 方法传入到 even handler 的 function 中（我们这里先讨论第一个参数，不过你应该浏览 API 文档以了解 even handler 更多的细节）。在我们之前的例子中，function 是忽略这些参数的，到这里可有少许的改变，——我们在功能上提供了更深层次的控制。必须先说明的是，这实际上是 Ext 的事件对象（event object）,一个跨浏览器和拥有更多控制的事件的对象。例如，可以用下列的语句，得到这个事件响应所在的 DOM 节点：

```
Ext.onReady(function() {
        var paragraphClicked = function(e) {
                Ext.get(e.target).highlight();
        }
        Ext.select('p').on('click', paragraphClicked);
```

```
});
```

注意得到的 e.target 是 DOM 节点，所以我们首先将其转换成为 EXT 的 Elemnet 元素，然后执行欲完成的事件，这个例子中，我们看见段落是高亮显示的。

使用 Widgets

(Widget 原意为"小器件"，现指页面中 UI 控件)

除了我们已经讨论过的核心 JavaScript 库，当前的 Ext 亦包括了一系列的最前端的 JavaScirptUI 组件库。文本以一个常用的 widget 为例子，作简单的介绍。

MessageBox

比起略为沉闷的"HelloWolrd"消息窗口，我们做少许变化，前面我们写的代码是，单击某个段落便会高亮显示，现在是单击段落，在消息窗口中显示段落内容出来。
在上面的 paragraphClicked 的 function 中，将这行代码：

```
Ext.get(e.target).highlight();
```

替换为：

```
var paragraph = Ext.get(e.target);
paragraph.highlight();
Ext.MessageBox.show({
        title: 'Paragraph Clicked',
        msg: paragraph.dom.innerHTML,
        width:400,
        buttons: Ext.MessageBox.OK,
        animEl: paragraph
});
```

这里有些新的概念需要讨论一下。在第一行中我们创建了一个局部变量（Local Variable）来保存某个元素的引用，即被单击的那个 DOM 节点（本例中，DOM 节点指的是段落 paragrah,事因我们已经定义该事件与< p>标签发生关联的了）。为什么要这样做呢？嗯...观察上面的代码，我们需要引用同一元素来高亮显示，在 MessageBox 中也是引用同一元素作为参数使用。
一般来说，多次重复使用同一值（Value）或对象，是一个不好的方式，所以，作为一个具备良好 OO 思维的开发者，应该是将其分配到一个局部变量中，反复使用这变量！

现在，为了我们接下来阐述新概念的演示，请观察 MessageBox 的调用。乍一看，这像一连串的参数传入到方法中，但仔细看，这是一个非常特别的语法。实际上，传入到 MessageBox.show 的只有一个参数：一个 Object literal,包含一组属性和属性值。在 Javascript 中，Object Literal 是动态的，你可在任何时候用{和}创建一个典型的对象(object)。其中的字符由一系列的 name/value 组成的属性，属性的格式是[property name]:[property value]。在整个 Ext 中，你将会经常遇到这种语法，因此你应该马上消化并吸收这个知识点！

使用 Object Literal 的原因是什么呢？主要的原因是“可伸缩性（flexibility）”的考虑"，随时可新增、删除属性，亦可不管顺序地插入。而方法不需要改变。这也是多个参数的情况下，为最终开发者带来不少的方便（本例中的 MessageBox.show())。例如，我们说这儿的 foo.action 方法，有四个参数，而只有一个是你必须传入的。本例中，你想像中的代码可能会是这样的 foo.action(null, null, null, 'hello').,若果那方法用 Object Literal 来写，却是这样， foo.action({ param4: 'hello' })，这更易用和易读。

Grid

Grid 是 Ext 中人们最想先睹为快的和最为流行 Widgets 之一。好，让我们看看怎么轻松地创建一个 Grid 并运行。用下列代码替换 ExtStart.js 中全部语句：

```
Ext.onReady(function() {
        var myData = [
                ['Apple',29.89,0.24,0.81,'9/1 12:00am'],
                ['Ext',83.81,0.28,0.34,'9/12 12:00am'],
                ['Google',71.72,0.02,0.03,'10/1 12:00am'],
                ['Microsoft',52.55,0.01,0.02,'7/4 12:00am'],
                ['Yahoo!',29.01,0.42,1.47,'5/22 12:00am']
        ];

        var ds = new Ext.data.Store({
                proxy: new Ext.data.MemoryProxy(myData),
                reader: new Ext.data.ArrayReader({id: 0}, [
                        {name: 'company'},
                        {name: 'price', type: 'float'},
                        {name: 'change', type: 'float'},
                        {name: 'pctChange', type: 'float'},
                        {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
        ])
        });
        ds.load();

        var colModel = new Ext.grid.ColumnModel([
                {header: "Company", width: 120, sortable: true, dataIndex:
'company'},
                {header: "Price", width: 90, sortable: true, dataIndex: 'price'},
                {header: "Change", width: 90, sortable: true, dataIndex: 'change'},
                {header: "% Change", width: 90, sortable: true, dataIndex:
'pctChange'},
                {header: "Last Updated", width: 120, sortable: true,
```

```
                    renderer:      Ext.util.Format.dateRenderer('m/d/Y'),      dataIndex:
'lastChange'}
        ]);

        var grid = new Ext.grid.Grid('grid-example', {ds: ds, cm: colModel});
                    grid.render();
                    grid.getSelectionModel().selectFirstRow();
        });
```

这看上去很复杂，但实际上加起来，只有七行代码。第一行创建数组并作为数据源。实际案例中，你很可能从数据库、或者 WebService 那里得到动态的数据。接着，我们创建并加载 data store， data store 将会告诉 Ext 的底层库接手处理和格式化这些数据。接着，我们定义一个 column 模型，用来轻松地调配 Grid 的每一列参数。最后我们生成这个 Grid,传入 data store 和 column 模型两个对象，进行渲染并选好第一行。不是太困难吧？如果一切顺利，完成之后你会看到像这样的：

Company	Price	Change	% Change	Last Updated
Apple	29.89	0.24	0.81	09/01/2007
Ext	83.81	0.28	0.34	09/12/2007
Google	71.72	0.02	0.03	10/01/2007
Microsoft	52.55	0.01	0.02	07/04/2007
Yahoo!	29.01	0.42	1.47	05/22/2007

当然，你可能未掌握这段代码的某些细节（像 MemoryProxy 究竟是什么？）但先不要紧，这个例子的目的是告诉你，用少量的代码，创建一个富界面的多功能的 UI 组件而已——这是完全可能的，只要读者您有兴趣学习。这儿有许多学习 Grid 的资源。Ext Grid 教程、交叉 Gird 演示和 Grid API 文档。

还有更多的..

这只是冰山一角。还有一打的 UI Widgets 可以供调用，如 layouts, tabs, menus, toolbars, dialogs, tree view 等等。请参阅 API 文档中范例演示。

使用 Ajax

在弄好一些页面后，你已经懂得在页面和脚本之间的交互原理（interact）。接下来，你应该掌握的是，怎样与远程服务器（remote server）交换数据，常见的是从数据库加载数据（load）或是保存数据（save）到数据库中。通过 JavaScript 异步无刷新交换数据的这种方式，就是所谓的 Ajax。Ext 内建卓越的 Ajax 支持，例如，一个普遍的用户操作就是，异步发送一些东西到服务器，然后，UI 元素根据回应（Response）作出更新。这是一个包含 text input 的表单，一个 div 用于显示消息（注意，你可以在 ExtStart.html 中加入下列代码，但这必须要访问服务器）：

```
<div id="msg" style="visibility: hidden"></div>
```

```
Name: <input type="text" id="name" /><br />

<input type="button" id="oKButton" value="OK" />
```

接着，我们加入这些处理交换数据的 JavaScript 代码到文件 ExtStart.js 中(用下面的代码覆盖)：

```
Ext.onReady(function(){
        Ext.get('oKButton').on('click', function(){
                var msg = Ext.get('msg');
                        msg.load({
                        url: [server url], //换成你的 URL
                        params: 'name=' + Ext.get('name').dom.value,
                        text: 'Updating...'

                });
                msg.show();
        });
});
```

这种模式看起来已经比较熟悉了吧！先获取按钮元素，加入单击事件的监听。在事件处理器中（event handler），我们使用一个负责处理 Ajax 请求、接受响应（Response）和更新另一个元素的 Ext 内建类，称作 UpdateManager。 UpdateManager 可以直接使用，或者和我们现在的做法一样，通过 Element 的 load 方法来引用（本例中该元素是 id 为“msg“的 div）。当使用 Element.load 方法，请求（request)会在加工处理后发送，等待服务器的响应（Response），来自动替换元素的 innerHTML。简单传入服务器 url 地址，加上字符串参数，便可以处理这个请求（本例中，参数值来自“name”元素的 value),而 text 值是请求发送时提示的文本，完毕后显示那个 msg 的 div（因为开始时默认隐藏）。当然，和大多数 Ext 组件一样，UpdateManager 有许多的参数可选，不同的 Ajax 请求有不同的方案。而这里仅演示最简单的那种。

PHP

```
<? if(isset($_GET['name'])) {
    echo 'From Server: '.$_GET['name'];
}
?>
```

ASP.Net

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Request["name"] != null)
        {
```

```
                    Response.Write("From Server: " + Request["name"]);
                    Response.End();
        }
}
```

Cold Fusion

```
<cfif StructKeyExists(url, "name")>
        <cfoutput>From Server: #url.name#</cfoutput>
</cfif>
```

最后一个关于 Ajax 隐晦的地方就是，服务器实际处理请求和返回（Resposne）是具体过程。这个过程会是一个服务端页面，一个 Servlet，一个 Http 调度过程，一个 WebService,甚至是 Perl 或 CGI 脚本，即不指定一个服务器都可以处理的 http 请求。让人无法预料的是，服务器返回什么是服务器的事情，无法给一个标准的例子来覆盖阐述所有的可能性。(这段代码输出刚才我们传入'name'的那个值到客户端，即发送什么，返回什么)。

使用 Ajax 的真正挑战，是需要进行适当的手工编码，并相应格式化为服务端可用接受的数据结构。有几种格式供人们选择（最常用为 JSON/XML)。正因 Ext 是一种与服务器语言免依赖的机制，使得其它特定语言的库亦可用于 Ext 处理 Ajax 服务。只要页面接受到结果是 EXT 能处理的数据格式，Ext 绝不会干涉服务器其他的事情！要全面讨论这个问题，已超出本文的范围。推荐正在 Ajax 环境下开发的您，继续深入阅读 Ext Ajax 教程。

下一步是？
现在你已经一睹 Ext 其芳容，知道她大概能做些什么了。下面的资源有助您进一步深入了解：

EXT 源码概述

揭示源代码

Javascript 是一门解释型的语言，意味着在运行之前代码是没有经过编译的。按照这种理论，在你网站上所发播的 Ext 代码是我们看的懂的（human-readible)。我这里说“理论上”，是因为实际情况中，很多源代码是经过某些自动化步骤的处理，生成很小几行的文件最终发布的，通过剔除空白符号和注释，或混淆等的方法，以减小文件大小。

仔细看看 EXT 标准源码 **ext-core.js**，你会发现这是一段超长的源码。这是刚才提及的自动化步骤生成的结果——对浏览器来说不错！可是对于我们是难以阅读的。

ext-core.js

```
/*
 * Ext JS Library 1.1
 * Copyright(c) 2006-2007, Ext JS, LLC.
```

```
 * licensing@extjs.com
 *
 * http://www.extjs.com/license
 */

Ext.DomHelper=function(){var _1=null;var _2=/^(?:br|frame...

Ext.Template=function(_1){if(_1 instanceof Array){_1...
...
```

接着看下去的是 **ext-core-debug.js** （注意在文件名后面加上**-debug** 的 JS 文件）， 我会发现是全部已格式化好的源代码。这个文件是配合调时器所使用的，像 Firebug 的工具能够可以让你一步一步地、一行一行地调试代码。你也会发现文件的体积将近大了一倍之多，这便是没有压缩或混淆的缘故。

ext-core-debug.js

```
/*
 * Ext JS Library 1.1
 * Copyright(c) 2006-2007, Ext JS, LLC.
 * licensing@extjs.com
 *
 * http://www.extjs.com/license
 */

Ext.DomHelper = function(){
    var tempTableEl = null;
    var emptyTags =
/^(?:br|frame|hr|img|input|link|meta|range|spacer|wbr|area|param|col)$/i;
    var tableRe = /^table|tbody|tr|td$/i;
...
```

该调试版本可以在调试阶段方便地检查 EXT 库运行到哪一步，但是你还是会错过一个有价值的...代码注释！要完整地看到代码，就要阅读真正的原始代码！

发布 Ext 源码时的一些细节

你在 download 得到的压缩文档，包含在这些文件夹之中的，有一 **source** 的子目录。在这个文件夹里面，正如所料，是全部的 EXT 的源文件，遵从 Lesser GNU （LGPL） 开源的协议。对于 EXT 开发者来说应该非常适合。

用你日常使用文本编辑器打开源代码的任意一个文件（推荐有高亮显示的编辑器，或是在这里 full-featured IDE 看看），便可以开始我们的探险！

我应该从哪里开始？

Ext 代码库里面包含了许多各种各样的文件，甚至令人觉得有点望而生畏。好在，Ext 是一个通过充分考虑后而设计的 JavaScript 库，——在底层的代码为各项应用提供稳健的基础如跨浏览器的各种 DOM 操控，使得在上层的*类 classes* 运行于一个较高级的抽象层面（*class* 一术语与我们已习惯的 Java 和 C++语言稍微有所不同，但一些概念如继承则可是如此类推去理解的——有关面向对象的 JavaScript 的更多资料，请参见 Introduction to object-oriented (OO) JavaScript）。

这意味着，当浏览源码的时候，采取"自顶向下（bottom-up）"还是"自下向顶（top-down）"的方式，都是无关紧要的。你所熟悉 API 里面的代码已经是属于最高的抽象层面的范畴，你可以根据你的兴趣，顺着这些你熟悉的 API 逐步深入。但是你若赞同我的看法，并打算深入了解其个中原理，最理想的地方是从底层代码开始。

适配器 Adapters

浏览器读取第一个源文件，当中的一个任务就是创建 Ext 对象本身。 **Ext.js**

```
Ext = {};
```

Ext 成型于 YahooUI 的 Javascript 库的扩展。在当时，Ext 须依赖 YUI 的底层代码来处理跨浏览器的问题。现在 ExtJS 已经是独立、免依赖的库了（standalone ），你可将 YUI 替换为另外你所选择 javascript 库，如 prototype、jQuery、或者是这些之中的最佳选择,—Ext 自带的底层库。负责将这些库（包括 Ext 自带的底层库）映射为 Ext 底层库的这部分代码，我们称之为适配器（Adapters）。这部分源码位于 **source/adapter** 的子目录。当项目引入 Ext 的时候便需要选择好你准备使用的适配器。

核心 Core

source/core 中的文件是构建于适配器 API 之上的"相对"最底层的源码。有些的源码甚至"底层"到直接为独立库的代码直接使用。这意味着应先了解和学习这整个库，再学习剩余的部分也不迟。要了解 Ext 的各种"Magic"和核心层面，就应该把重点放在 source/core 目录下的各个源代码。

Javascript 中的作用域（scope）

事前准备

学习本教程的最佳方法是随手准备好 Firefox 中的工具 Firebug。这样使得您可以即刻测试教程的例子。

如果机子上还没有 FireFox 和 FireBug，就应该尽快安装一套来用。

定义

作用域 scope

1.（名词）某样事物执行、操作、拥有控制权的那么一个区域 [1]
2. （名词） 编写程序时，程序之中变量的可见度；例如，一个函数能否使用另外一个函数所创建的变量。[2]

可是这能够说明什么问题呢？ 每当有人在说“这是作用域的问题”或“作用域搞错了”的时候，那就是说某个函数运行起来的时候，找不到正确变量的位置。这样我们便知道应该从哪一方面入手，查找出问题所在。

正式开始

实际上每一个你定义的函数都是某个对象的方法。甚至是这样的写法：

```
function fn() {
    alert(11);
}
```

老兄你不是故弄玄虚吧～。做一个这样的演示可真得是简单得要命。没错！本例不需要任何 Javascript 文件，服务器或 html。你只要打开 firefox，弹出 firebug,点击 console tab。在 Firefox 状态栏上面看到有>>>提示的地方就可以输入了。

输入：

```
function fn() { alert(11); };
```

然后回车。一切安然...你刚才做的实际上是定义了一个函数 fn。接着试试：

```
fn();
```

然后回车。得到 11 的警告窗口？还不错吧？接着试试：

```
window.fn();
this.fn();
```

得到一样的结果吧？这是因为函数 fn 是 window 对象的一个方法，在第二行的"this"的作用域实际指向了 windows 对象。不过多数情况中你不需要像这样 window.myFunction(...)地调用函数，这样太麻烦了，程序员工作起来会很不方便。

window 对象

window 对象总是存在的，你可理解其为一个浏览器窗口对象。它包含了其它所有的对象如 **document** 和所有的全局变量。

你可以打开 Firebug，切换到 **Script** 页面并在 Firebug 右侧的 **New watch expression...** 里面输入 **window**。观察 window 对象究竟有什么在里面。

接着，尝试找出我们之前定义过的 **fn** 函数。

另外，每个 frame 或 iframe 拥有其自身的 window 对象，其自身的全局空间。

理解作用域

接下的内容开始有点复杂了。切换到 Firebug **Console** 标签页然后输入：

```
var o1 = {testvar:22, fun:function() { alert('o1: ' + this.testvar); }};
var o2 = {testvar:33, fun:function() { alert('o2: ' + this.testvar); }};
```

结果是什么？你声明了 **o1** 和 **o2** 两个对象，分别都有一些属性和方法，但值不同。

接着试试：

```
fun();
window.fun();
this.fun();
```

出错了，是吧？因为 window 对象（等价于 this）并没有 fun 的方法。试一试下面的：

```
o1.fun();
o2.fun();
```

22 和 33 出来了？非常好！

接下来这部分的内容最复杂啦。基于这个原始的函数，如果对象的数量多的话，你必须为每个对象加上这个函数－明显是重复劳动了。这样说吧，o1.fun 写得非常清晰的而且为了搞掂它已经占用了我一个星期的开发时间。想象一下代码到处散布着 **this** 变量，怎么能不头疼？如果要将调用（执行）的 o1.fun 方法但 **this** 会执行 o2,应该怎么实现呢？试一试下面的：

```
o1.fun.call(o2);
```

明白了吗？当执行 o1 的 fun 方法时你强行将变量 **this** 指向到 o2 这个对象，换句话说，更加严谨地说:**o1.fun 的方法在对象 o2 的作用域下运行。**

当运行一个函数，一个对象的方法时，你可将作用域当作 **this 值的变量**。

变量的可见度

变量的可见度和作用域的关系非常密切。我们已经了解到，可在任何对象的外部，声明变量，或在全局的函数（函数也是变量的一种）也可以，更严格说，它们是全局对象 **window** 的属性。 **全局变量在任何地方都可见；无论函数的内部还是外部。如果你在某一个函数内修改了一个全局变量，其它函数也会得知这个值是修改过的。**

对象可以有它自己的属性（像上面的 **testvar**）,这些属性允许从内部或是外部均是可见的。试：

```
alert(o1.testvar); // 从外部访问 o1 的属性 testvar
```

从内部访问的演示可在两个测试对象的 **fun** 方法找到。

用关键字 **var** 在内部声明，相当于声明局部变量（局部声明也是在一条链上，即 Scope Chain 作用域

链上,Frank 注）：

```
i = 44;
function fn2() {
    var i = 55;
    alert(i);
}
fn2();
```

将得到什么？对了，55。声明在函数 fn2 的变量 **i** 是一个本地变量（局部变量），和等于 44 的全局变量 **i** 44 没什么关系。 But:

```
alert(i);
```

这会访问全局变量 **i**，显示 44。

希望本文能帮助读者彻底理解作用域变量可见性的含义。

EXT 程序规划入门

事前准备

本教程假设你已经安装好 ExtJS 库。安装的目录是 **extjs** 并位于你程序的上一级目录。如果安装在其它地方你必须更改路径，更改示例文件中 script 标签的 **src** 的属性。

需要些什么？

除 ExtJS 库本身外，我们还需要两个文件：

- applayout.html
- applayout.js

先看看一份 html 文档，比较精简。并附有详细说明：

applayout.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css">
```

```
    <script type="text/javascript" src="../extjs/adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../extjs/ext-all-debug.js"></script>
    <script type="text/javascript" src="applayout.js"></script>
    <!-- 本地化的脚本引用在这里 -->
    <script type="text/javascript">
        Ext.onReady(myNameSpace.app.init, myNameSpace.app);
    </script>
    <title>Application Layout Tutorial</title>
</head>
<body>
</body>
</html>
```

开头的两行声明了文档的类型。程序可以不用 doctype,但是这样的话浏览器可能默认其为 Quirks 怪僻类型，会导致处理跨浏览器这一问题上出现差异。

我们采用 HTML 4.01 Transitional 的文档类型，该类型在主流浏览器上支持得不错。当然，你也可以根据你的需求采用其它类型的 doctype，但是记住**别忘了要加上 doctype**。

接着指定 HTML 头部的 Content-Type。做这些事情其实很琐碎，但总是有益处。

然后引入 EXT 的样式，适配器和 EXTJS 本身。有两种版本的 ExtJS:

- **ext-all.js** - 不能直接阅读，加载时更快，用于产品发布
- **ext-all-debug.js** - 便于阅读，开发阶段使用,

开发阶段的时候便需要引入 debug 的版本。

applayout.js 这个文件就是我们的程序，紧跟着的是本地化的脚本，这里可以换成 Extjs 翻译好的版本

跟着我们开始分配事件句柄(event handler)，使得在文档全部加载完毕后，程序就可以初始化（运行）。

下面的这一行：

```
Ext.onReady(myNameSpace.app.init, myNameSpace.app);
```

可这样说：当文档全部加载完毕,就运行 myNameSpace.app 的 init 方法,规定的作用域是 myNameSpace.app。

然后是标题，头部的结尾，body（当前空）和结束标签。

文档的解说就说到这儿了。

applayout.js

```
/**
```

```
  * Application Layout
  * by Jozef Sakalos, aka Saki
  * http://extjs.com/learn/Tutorial:Application_Layout_for_Beginners_(Chinese)
  */

// 填充图片的本地引用
Ext.BLANK_IMAGE_URL = '../extjs/resources/images/default/s.gif';

// 创建命名空间
Ext.namespace('myNameSpace');

// 创建应用程序
myNameSpace.app = function() {
    // 元素还没创建,未能访问

    // 私有变量

    // 私有函数

    // 公共空间
    return {
        // 公共的属性,如,要转换的字符串
        // 公共方法
        init: function() {
            alert('应用程序初始化成功。');
        }
    };
}(); // 程序底部

// 文件底部
```

文件最开始的几行是注释，说明该文件的主要内容，作者，作者相关的资讯。没有任何注释的程序也可以正常的运行，一但请记住：**每次写的程序要容易给人看得懂的 Always write your application as if you would write it for another.**当你回头看你几个月前写的代码，发现你根本不记得自己写过什么的时候，就会明白这个道理，并养成编码的好习惯。接着是要指向你服务器的填充图片，如不指定则默认 extjs.com。每次运行程序的时候都访问 extjs.com，不推荐这样，应该先修改这个常量值指向到本地。

现在自定义命名空间。将所有变量和方法都划分到一个全局对象下管理，这样的好处是避免了变量名冲突和由不同函数干扰了全局变量的值。名字（namespace）可按自己的方案选择。

整段代码的重点是，我们创建了 **myNameSpace** 对象的属性 **app**，其值是一个函数立刻运行之后的返回值。

如果运行我们的代码：

```
var o = function() {
    return {p1:11, p2:22};
}();
```

实际上我们创建了一个匿名函数（没有名字的函数），经过解释（预编译？）之后让它立刻运行（注意函数后面的**()**）。最后将函数返回的对象（注意此时是一个 object 变量）分配到变量 **o**。我们的程序便是这种思路去写的。

你可以把私有变量和私有函数直接定义在 function 和 return 这两个声明之间，但是请切记：此时不能访问任何 html 页面中的元素，那会导致错误，因为这段代码在加载时页面的 head 中就运行了，而这时候 html 页面中的其它元素还没有被加载进来。

另外一方面，函数 **init**,是由匿名函数返回的对象的一个方法而已。它会在文档全部加载**后**才运行。换言之整个 DOM 树已经是可用的了。

一切都还好吧～如果能正确运行 http://yourserver.com/applayout/applayout.html，不出现什么错误的话将出现一个警告。

接下来是利用这个空白的模板，讨论本文的重点。

公开 Public、私有 Private、特权的 Privileged?

让我们加入一些有趣内容到程序中去吧。在页面 **applayout.html** 的 body 标签中加入：

```
<div id="btn1-ct"></div>
```

空白的 div 会当作按钮的容器来使用。然后在 **applayout.js** 加入下来代码：

```
/**
  * Application Layout
  * by Jozef Sakalos, aka Saki
  * http://extjs.com/learn/Tutorial:Application_Layout_for_Beginners_(Chinese)
  */

//  填充图片的本地引用
Ext.BLANK_IMAGE_URL = '../extjs/resources/images/default/s.gif';

// 创建命名空间
Ext.namespace('myNameSpace');

// 创建应用程序
```

```
myNameSpace.app = function() {
    // 元素还没创建,未能访问

    // 私有变量
    var btn1;
    var privVar1 = 11;

    // 私有函数
    var btn1Handler = function(button, event) {
        alert('privVar1=' + privVar1);
        alert('this.btn1Text=' + this.btn1Text);
    };

    // 公共空间
    return {
        // 公共的属性,如,要转译的字符串
        btn1Text: 'Button 1'

        // 公共方法
        , init: function() {
            btn1 = new Ext.Button('btn1-ct', {
                text: this.btn1Text
              , handler: btn1Handler
            });
        }
    };
}(); //程序底部

// 文件底部
```

变量 **btn1** 和 **privVar1** 是**私有的**。 那意味着在程序外部他们是不能够被访问的，而且也不可见。私有函数 **btn1Handler** 也是同样道理。

另外一个方面，**btn1Text** 是公共变量所以可以被程序外部访问或者是修改（我们稍后将会演示）。

函数 **init** 是**特权的**，即是私有变量和公共变量两者都可以被它访问到。在本例中，公共变量 **this.btn1Text** 和私有函数 **btn1Handler** 都能够被**特权函数** ***init*** *所访问。同时，相对外界来说，它也属于**公共**成员的一种。*

Ok,运行看看。能看到左上角的按钮吗？很好，点击它。得到一个 **privVar1=11** 的警告。这说明私有函数能访问私有变量。

但是在第二个警告中遇到了 **this.btn1Text=undefined** 的问题，这好像不应该这样。个中原因是因为位

于事件句柄(event handler)中的变量 **this** 没有正确指向到我们的程序。你需要用 **scope:this** 指明这个作用域（这里的 this 关键字所指示的 scope 应该是 btn1 变量）：

```
btn1 = new Ext.Button('btn1-ct', {
        text: this.btn1Text
     , handler: btn1Handler
     , scope: this
});
```

刷新一下，可以了吧？

重写公共变量

在 **applayout.js** 底部加入下列代码：

```
Ext.apply(myNameSpace.app, {
     btn1Text:'Taste 1'
});

// 文件底部
```

这代码是用来干什么的呢？首先它创建了我们的程序对象然后改变（重写）公共变量 **btn1Text** 的值。运行后将看到按钮上文字的变化。

把文本都放到公共变量，以便于以后的国际化翻译工作，而不需要修改程序的内部代码。

当然你也可以将其它的值例如尺寸、样式、等等的总之是可自定义的选项都放到公共变量中。免于在数千行代码之中为查找某个颜色而费劲。

重写（Overriding）公共函数

接着更改一下代码，让它读起来是这样的：

```
Ext.apply(myNameSpace.app, {
        btn1Text:'Taste 1'
     , init: function() {
          try {
               btn1 = new Ext.Button('btn1-ct', {
                    text: this.btn1Text
                  , handler: btn1Handler
                  , scope: this
               });
          }
```

```
        catch(e) {
            alert('错误: "' + e.message + '" 发生在行: ' + e.lineNumber);
        }
    }
});

// end of file
```

我们这里将 **init** 重写了一篇，主要是在原来代码的基础上加入异常控制，以便能够捕获异常。试运行一下看还有没有其它的变化？

嗯 是的，出现了 **btn1Handler** 未定义的错误。这是因为新的函数虽然尝试访问私有变量但它实际是不允许的。正如所见，原 init 是特权函数可访问私有空间，但新的 init 却不能。之所以不能访问私有空间，是因为：禁止外部代码 No code from outside，哪怕是尝试重写特权方法。

DomQuery 基础

本教程旨在为读者了解怎样利用单例对象 Ext.DomQuery,浏览穿梭于 DOM 树之中和获取对象，提供一个起点。

DomQuery 基础

DomQuery 的 select 函数有两个参数。第一个是选择符字符（selector string ）而第二个是欲生成查询的标签 ID(TAG ID)。
本文中我准备使用函数“Ext.query”但读者须谨记它是“Ext.DomQuery.select()”的简写方式。

这是要入手的 html：

```
<html>
 <head>
  <script type="text/javascript" src="../js/firebug/firebug.js"></script>
 </head>
 <body>
  <script type="text/javascript" src="../ext/ext-base.js"></script>
  <script type="text/javascript" src="../ext/ext-core.js"></script>
  <div id="bar"   class="foo">
   I'm a div ==> my id: bar, my class: foo
   <span class="bar">I'm a span within the div with a foo class</span>
   <a href="http://www.extjs.com" target="_blank">An ExtJs link</a>
  </div>
  <div id="foo" class="bar">
   my id: foo, my class: bar
```

```
    <p>I'm a P tag within the foo div</p>
    <span class="bar">I'm a span within the div with a bar class</span>
    <a href="#">An internal link</a>
  </div>
 </body>
</html>
```

第一部分：元素选择符 Selector

假设我想获取文档内所有的“span”标签：

```
// 这个查询会返回有两个元素的数组因为查询选中对整个文档的所有 span 标签。
Ext.query("span");
// 这个查询会返回有一个元素的数组因为查询顾及到了 foo 这个 id。
Ext.query("span", "foo");
```

注意刚才怎么传入一个普通的字符串作为第一个参数。

按 id 获取标签，你需要加上“#”的前缀：

```
// 这个查询会返回包含我们 foo div 一个元素的数组！
Ext.query("#foo");
```

按 class name 获取标签，你需要加上“.”的前缀：

```
/*这个查询会返回有一个元素的数组，
包含与之前例子一样的 div 但是我们使用了 class name 来获取*/
Ext.query(".foo");
```

你也可以使用关键字“*”来获取所有的元素：

```
// 这会返回一个数组，包含文档的所有元素。
Ext.query("*");
```

要获取子标签，我们只须在两个选择符之间插入一个空格：

```
// 这会返回有一个元素的数组，包含 p 标签的 div 标签
Ext.query("div p");
// 这会返回有两个元素的数组，包含 span 标签的 div 标签
Ext.query("div span");
```

还有三个的元素选择符，待后续的教程会叙述。 ""

如果朋友你觉得这里说得太简单的话，你可以选择到 DomQuery 文档看看，可能会有不少收获:)

第二部分：属性选择符 Attributes selectors

这些选择符可让你得到基于一些属性值的元素。属性指的是 html 元素中的 **href**, **id** 或 **class**。

```
// 我们检查出任何存在有 class 属性的元素。
// 这个查询会返回 5 个元素的数组。
Ext.query("*[class]"); // 结果： [body#ext-gen2.ext-gecko, div#bar.foo, span.bar, div#foo.bar, span.bar]
```

现在我们针对特定的 class 属性进行搜索。

```
// 这会得到 class 等于"bar"的所有元素
Ext.query("*[class=bar]");

// 这会得到 class 不等于"bar"的所有元素
Ext.query("*[class!=bar]");

// 这会得到 class 从"b"字头开始的所有元素
Ext.query("*[class^=b]");

//这会得到 class 由"r"结尾的所有元素
Ext.query("*[class$=r]");

//这会得到在 class 中抽出"a"字符的所有元素
Ext.query("*[class*=a]");
```

第三部分： CSS 值元素选择符

这些选择符会匹配 DOM 元素的 **style** 属性。尝试在那份 html 中加上一些颜色：

```
<html>
 <head>
  <script type="text/javascript" src="../js/firebug/firebug.js"></script>
 </head>
 <body>
  <script type="text/javascript" src="../ext/ext-base.js"></script>
  <script type="text/javascript" src="../ext/ext-core.js"></script>
  <div id="bar" class="foo" style="color:red;">
   我是一个 div ==> 我的 id 是： bar, 我的 class: foo
```

```
    <span class="bar" style="color:pink;">I'm a span within the div with a foo class</span>
    <a href="http://www.extjs.com" target="_blank" style="color:yellow;">An ExtJs link with a blank target!</a>
  </div>
  <div id="foo" class="bar" style="color:fushia;">
   my id: foo, my class: bar
   <p>I'm a P tag within the foo div</p>
    <span class="bar" style="color:brown;">I'm a span within the div with a bar class</span>
    <a href="#" style="color:green;">An internal link</a>
  </div>
 </body>
</html>
```

基于这个 CSS 的颜色值我们不会作任何查询，但可以是其它的内容。它的格式规定是这样的：

元素{属性 操作符 值}

注意我在这里是怎么插入一个不同的括号。

所以，操作符（operators）和属性选择符（attribute selectors）是一样的。

```
// 获取所以红色的元素
Ext.query("*{color=red}"); // [div#bar.foo]

// 获取所有粉红颜色的并且是有红色子元素的元素
Ext.query("*{color=red} *{color=pink}"); // [span.bar]

// 获取所有不是红色文字的元素
Ext.query("*{color!=red}");
//[html, head, script firebug.js, link, body#ext-gen2.ext-gecko,
// script ext-base.js, script ext-core.js, span.bar,
//a www.extjs.com, div#foo.bar, p, span.bar, a test.html#]

// 获取所有颜色属性是从"yel"开始的元素
Ext.query("*{color^=yel}"); // [a www.extjs.com]

// 获取所有颜色属性是以"ow"结束的元素
Ext.query("*{color$=ow}"); // [a www.extjs.com]

// 获取所有颜色属性包含"ow"字符的元素
```

```
Ext.query("*{color*=ow}"); // [a www.extjs.com, span.bar]
```

第四部分：伪类选择符 Pseudo Classes selectors

仍然是刚才的网页，但是有所不同的只是新加上了一个 UL 元素、一个 TABLE 元素和一个 FORM 元素，以便我们可以使用不同的伪类选择符，来获取节点。

```
<html>
 <head>
  <script type="text/javascript" src="../js/firebug/firebug.js"></script>
 </head>
 <body>
  <script type="text/javascript" src="../ext/ext-base.js"></script>
  <script type="text/javascript" src="../ext/ext-core.js"></script>
  <div id="bar" class="foo" style="color:red; border: 2px dotted red; margin:5px; padding:5px;">
   I'm a div ==> my id: bar, my class: foo
   <span class="bar" style="color:pink;">I'm a span within the div with a foo class</span>
   <a href="http://www.extjs.com" target="_blank" style="color:yellow;">An ExtJs link with a blank target!</a>
  </div>
  <div id="foo" class="bar" style="color:fushia; border: 2px dotted black; margin:5px; padding:5px;">
   my id: foo, my class: bar
   <p>I'm a P tag within the foo div</p>
   <span class="bar" style="color:brown;">I'm a span within the div with a bar class</span>
   <a href="#" style="color:green;">An internal link</a>
  </div>
  <div style="border:2px dotted pink; margin:5px; padding:5px;">
   <ul>
    <li>Some choice #1</li>
    <li>Some choice #2</li>
    <li>Some choice #3</li>
    <li>Some choice #4 with a <a href="#">link</a></li>
   </ul>
   <table style="border:1px dotted black;">
```

```
    <tr style="color:pink">
     <td>1st row, 1st column</td>
     <td>1st row, 2nd column</td>
    </tr>
    <tr style="color:brown">
        <td colspan="2">2nd row, colspanned! </td>
    </tr>
    <tr>
     <td>3rd row, 1st column</td>
     <td>3rd row, 2nd column</td>
    </tr>
   </table>
  </div>
  <div style="border:2px dotted red; margin:5px; padding:5px;">
   <form>
    <input id="chked" type="checkbox" checked/><label for="chked">I'm checked</label>
    <br /><br />
    <input id="notChked" type="checkbox" /><label for="notChked">not me brotha!</label>
   </form>
  </div>
 </body>
</html>
```

接着：

```
/*
 this one gives us the first SPAN child of its parent
*/
Ext.query("span:first-child"); // [span.bar]

/*
 this one gives us the last A child of its parent
*/
Ext.query("a:last-child") // [a www.extjs.com, a test.html#]

/*
```

```
 this one gives us the second SPAN child of its parent
*/
Ext.query("span:nth-child(2)") // [span.bar]

/*
 this one gives us ODD TR of its parents
*/
Ext.query("tr:nth-child(odd)") // [tr, tr]

/*
 this one gives us even LI of its parents
*/
Ext.query("li:nth-child(even)") // [li, li]

/*
 this one gives us A that are the only child of its parents
*/

Ext.query("a:only-child") // [a test.html#]

/*
 this one gives us the checked INPUT
*/
Ext.query("input:checked") // [input#chked on]

/*
 this one gives us the first TR
*/
Ext.query("tr:first") // [tr]

/*
 this one gives us the last INPUT
*/
Ext.query("input:last") // [input#notChked on]

/*
 this one gives us the 2nd TD
*/
```

```
Ext.query("td:nth(2)") // [td]

/*
 this one gives us every DIV that has the "within" string
*/
Ext.query("div:contains(within)") // [div#bar.foo, div#foo.bar]

/*
 this one gives us every DIV that doesn't have a FORM child
*/
Ext.query("div:not(form)") [div#bar.foo, div#foo.bar, div]

/*
 This one gives use every DIV that has an A child
*/
Ext.query("div:has(a)") // [div#bar.foo, div#foo.bar, div]

/*
 this one gives us every TD that is followed by another TD.
 obviously, the one that has a colspan property is ignored.
*/
Ext.query("td:next(td)") // [td, td]

/*
 this one gives us every LABEL that is preceded by an INPUT
*/
Ext.query("label:prev(input)") //[label, label]
```

扩展 EXT 组件

要创建的扩展是一个在文字前面能够显示图标的这么一个 Ext.form.Combobox。将其中一个功能举例来说，就是要在一块选择里，国家名称连同国旗一并出现。

我们先给扩展起个名字，就叫 **Ext.ux.IconCombo**。

文件的创建

首要的步骤是准备好开发中将会使用的文件。需下列文件：

- **iconcombo.html**：新扩展将会使用的 html markup

- **iconcombo.js**: 程序 javascript 代码
- **Ext.ux.IconCombo.js**: 扩展的 javascript 文件
- **Ext.ux.IconCombo.css**: 扩展样式表

iconcombo.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <link rel="stylesheet" type="text/css" href="../extjs/resources/css/ext-all.css">
    <link rel="stylesheet" type="text/css" href="Ext.ux.IconCombo.css">
    <script type="text/javascript" src="../extjs/adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../extjs/ext-all-debug.js"></script>
    <script type="text/javascript" src="Ext.ux.IconCombo.js"></script>
    <script type="text/javascript" src="iconcombo.js"></script>
    <!-- A Localization Script File comes here -->
    <script type="text/javascript">Ext.onReady(iconcombo.init, iconcombo);</script>
    <title>Ext.ux.IconCombo Tutorial</title>
</head>
<body>
<div style="position:relative;width:300px;top:24px;left:64px;font-size:11px">
    <div>Icon combo:</div>
    <div id="combo-ct"></div>
</div>
</body>
</html>
```

该文件来自教程 Ext 程序规划入门 的轻微修改。

iconcombo.js

```
/**
  * Ext.ux.IconCombo Tutorial
  * by Jozef Sakalos, aka Saki
  * http://extjs.com/learn/Tutorial:Extending_Ext_Class
  */
```

```
// 引用本地空白文件
Ext.BLANK_IMAGE_URL = '../extjs/resources/images/default/s.gif';

// 创建程序
iconcombo = function() {

    // 公共空间
    return {
        // public properties, e.g. strings to translate

        // public methods
        init: function() {
            var icnCombo = new Ext.ux.IconCombo({
                store: new Ext.data.SimpleStore({
                    fields: ['countryCode', 'countryName', 'countryFlag'],
                    data: [
                        ['US', 'United States', 'x-flag-us'],
                        ['DE', 'Germany', 'x-flag-de'],
                        ['FR', 'France', 'x-flag-fr']
                    ]
                }),
                valueField: 'countryCode',
                displayField: 'countryName',
                iconClsField: 'countryFlag',
                triggerAction: 'all',
                mode: 'local',
                width: 160
            });
            icnCombo.render('combo-ct');
            icnCombo.setValue('DE');
        }
    };
}(); // end of app

// end of file
```

我们在这个文件中创建 IconCombo，以便可以进行扩展和测试。

Ext.ux.IconCombo.js

```
// Create 创建用户的扩展（User eXtensions namespace (Ext.ux)）
Ext.namespace('Ext.ux');

/**
  * Ext.ux.IconCombo 扩展类
  *
  * @author Jozef Sakalos, aka Saki
  * @version 1.0
  *
  * @class Ext.ux.IconCombo
  * @extends Ext.form.ComboBox
  * @constructor
  * @param {Object} config 配置项参数
  */
Ext.ux.IconCombo = function(config) {

    // 调用父类的构建函数
    Ext.ux.IconCombo.superclass.constructor.call(this, config);

} // Ext.ux.IconCombo 构建器的底部

// 进行扩展
Ext.extend(Ext.ux.IconCombo, Ext.form.ComboBox, {

}); // 扩展完毕

// 文件底部
```

运行到这一步，实际这是一个没有对 Ext.form.ComboBox 新加任何东西的*空扩展*。我们正是需要这个完成好的*空扩展*，再继续下一步。

Ext.ux.IconCombo.css

```
.x-flag-us {
    background-image: url(../img/flags/us.png);
}
.x-flag-de {
```

```
    background-image: url(../img/flags/de.png);
}
.x-flag-fr {
    background-image: url(../img/flags/fr.png);
}
```

路径可能根据你所在的国旗放置目录有所不同。国旗的资源可在 here 下载。

Let's go

So far so good！如果你浏览 iconcombo.html 应该会发现一个包含三个选项的标准 combo,而德国的那个是选中的...是吧？不过还没有图标...

现在正是开始工作。在调用父类构建器之后加入下列行：

```
this.tpl = config.tpl ||
        '<div class="x-combo-list-item">'
    + '<table><tbody><tr>'
    + '<td>'
    + '<div class="{' + this.iconClsField + '} x-icon-combo-icon"></div></td>'
    + '<td>{' + this.displayField + '}</td>'
    + '</tr></tbody></table>'
    + '</div>'
;
```

在这一步，我们将默认 combox box 的模版重写为 **iconClsField** 模版。

现在加入 Ext.ux.IconCombo.css 中的样式文件：

```
.x-icon-combo-icon {
    background-repeat: no-repeat;
    background-position: 0 50%;
    width: 18px;
    height: 14px;
}
```

不错！可以测试一下了，刷新的页面，还好吧！？嗯，列表展开时那些漂亮的图标就出来了。。还有。。我们不是要在关闭时也出现图标的吗？

在构建器中加入创建模版的过程：

```
this.on({
    render:{scope:this, fn:function() {
        var wrap = this.el.up('div.x-form-field-wrap');
```

```
        this.wrap.applyStyles({position:'relative'});
        this.el.addClass('x-icon-combo-input');
        this.flag = Ext.DomHelper.append(wrap, {
            tag: 'div', style:'position:absolute'
        });
    }}
});
```

加入 **事件 render** 的侦听器，用于调整元素样式和创建国旗的 div 容器。如后按照下列方式进行扩展：

```
// 进行扩展
Ext.extend(Ext.ux.IconCombo, Ext.form.ComboBox, {

    setIconCls: function() {
        var rec = this.store.query(this.valueField, this.getValue()).itemAt(0);
        if(rec) {
            this.flag.className = 'x-icon-combo-icon ' + rec.get(this.iconClsField);
        }
    },

    setValue: function(value) {
        Ext.ux.IconCombo.superclass.setValue.call(this, value);
        this.setIconCls();
    }

}); // 扩展完毕
```

新增 **setIconCls** 函数并重写 **setValue** 函数。我们还是需要父类的 setValue 的方法来调用一下，接着再调用 setIconCls 的函数。最后，我们应该在文件 Ext.ux.IconCombo.css 加入下列代码：

```
.x-icon-combo-input {
    padding-left: 26px;
}
.x-form-field-wrap .x-icon-combo-icon {
    top: 3px;
    left: 6px;
}
```

完成

最后再刷新一下，如果一切顺利，那这个就是新的 Ext.ux.IconCombo 扩展！ 希望你能在此基础上扩展更多的组件！

谢谢 Brian Moeskau 提醒，使得能进一步精简 Ext.ux.IconCombo 代码，才称得上**最终版本**。最终代码和 CSS 为：

Ext.ux.IconCombo.js

```
// Create user extensions namespace (Ext.ux)
Ext.namespace('Ext.ux');

/**
  * Ext.ux.IconCombo Extension Class
  *
  * @author  Jozef Sakalos
  * @version 1.0
  *
  * @class Ext.ux.IconCombo
  * @extends Ext.form.ComboBox
  * @constructor
  * @param {Object} config Configuration options
  */
Ext.ux.IconCombo = function(config) {

    // call parent constructor
    Ext.ux.IconCombo.superclass.constructor.call(this, config);

    this.tpl = config.tpl ||
            '
{'
        + this.displayField
        + '}
'
    ;

    this.on({
        render:{scope:this, fn:function() {
```

```
            var wrap = this.el.up('div.x-form-field-wrap');
            this.wrap.applyStyles({position:'relative'});
            this.el.addClass('x-icon-combo-input');
            this.flag = Ext.DomHelper.append(wrap, {
                tag: 'div', style:'position:absolute'
            });
        }}
    });
} // end of Ext.ux.IconCombo constructor

// extend
Ext.extend(Ext.ux.IconCombo, Ext.form.ComboBox, {

    setIconCls: function() {
        var rec = this.store.query(this.valueField, this.getValue()).itemAt(0);
        if(rec) {
            this.flag.className = 'x-icon-combo-icon ' + rec.get(this.iconClsField);
        }
    },

    setValue: function(value) {
        Ext.ux.IconCombo.superclass.setValue.call(this, value);
        this.setIconCls();
    }

}); // end of extend

// end of file
```

Ext.ux.IconCombo.css

```
css
```

```
/* application specific styles */
.x-flag-us {
    background-image:url(../img/flags/us.png);
}
.x-flag-de {
```

```
    background-image:url(../img/flags/de.png);
}
.x-flag-fr {
    background-image:url(../img/flags/fr.png);
}

/* Ext.ux.IconCombo mandatory styles */
.x-icon-combo-icon {
    background-repeat: no-repeat;
    background-position: 0 50%;
    width: 18px;
    height: 14px;
}
.x-icon-combo-input {
    padding-left: 25px;
}
.x-form-field-wrap .x-icon-combo-icon {
    top: 3px;
    left: 5px;
}
.x-icon-combo-item {
    background-repeat: no-repeat;
    background-position: 3px 50%;
    padding-left: 24px;
}
```

EXT 的布局（Layout）

Ext 的 layout 布局对于建立 WEB 程序尤为有用。关于布局引擎（layout engine），区域管理器（region manager）的教程将分为几部分，本文是第一篇，为您介绍如何创建区域，如何增加版面到这些区域。

布局引擎(layout engine)这一功能早已在 EXT 前个 ALPHA 实现了。 Jack Slocum 对于怎样环绕某一区域，给与指定区域管理的策略，和建立界面的问题，在他的第一、第二篇关于跨浏览器的 WEB2.0 布局功能的博客中，进行过讨论。定义一个 DOM 元素的边界（edge），使之一个布局的边框（border）——这种做法使得创建“富界面”客户端 UI 的开发更进一大步。

布局管理器(layout manager)负责管理这些区域。布局管理的主要的用户组件是 BorderLayout 类。该类为 EXT 开发富界面的程序提供了一个切入点。Layout 的含意是划分好一些预定的区域。可用的区域分别有 south, east, west, north,和 center。每一个 BorderLayout 对象都提供这些区域但只有 center 要求必须使用的。如果你在单独一个区域中包含多个面板，你可通过 NestedLayoutPanel 类套嵌到 BorderLayout 实例中。

注意事项：本教程的每个文件都是.html 和.js 格式的。教程每一步都有演示,你也可以下载这些文件在编辑器（zip 格式提供在这里）中看看发生什么事。

面板（Panel）是区域管理（region management）的另外一个组件。面板提供了这么一个地方，可为您的 EXT 器件（widget）、加载的 HTML,嵌入的 IFrames、或者是你日常在 HTML 页面上摆放的随便一样东西。NestedLayoutPanel 也是一个面板，只不过用于链接多个 BorderLayout 的区域，其它的面板包括内容面板 ContentPanel,Grid 面板 GridPanel,和 Tree 面板 TreePanel。

简单的例子

下面的 layout 包含 north, south, east, west,和 center 的区域，而且每个区域包含一个 ContentPanel，各区域之间使用得了分隔条分割开。

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

```
    },
    west: {
        split: true, initialSize: 100
    },
    center: {
    }
});
```

这是一个非常基本的 layout,只是分配了东南西北中间的区域、分隔条、设置一下初始尺寸，并最迟定义中间区域。本例中， BorderLayout 被绑定到"document.body"这个 DOM 元素，其实 BorderLayout 还可以绑定到任何一个封闭的 DOM 元素。定义好 BorderLayout 之后，我们加入 ContentPanel 对象（基于本例）。

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

当前的例子是将 ContentPanel 加入到所有区域中。由调用 mainLayout.beginUpdate()开始。beginUpdate()告诉 BorderLayout 对象在执行 endUpate()方法之前，先不要对加入的对象排版布局。这样的好处是避免了 ContentPanel 有对象加入时，导致 UI 的刷新，改进了整体的用户体验。执行 beginUpdate()之后，加入五个 ContentPanel 对象到区域。所有的 ContentPanel 对象（除中间的那个外），都设置是可关闭的（closbale）。所有的 ContentPanel 对象也都设置为自动适配它们的父元素。最后执行 endUpdate()渲染 layout。

InternetExploer 注意事项：BorderLayout 所容纳的元素必须有一个 SIZE 以便正确渲染。典型地你无须为 document.body 指明 size，因为 document.body 通常是有 size 的了（大多数情况，一除非你在浏览器上什么也看不到）。但是如果你将 layout 连同容器放到现有的 web 页面上（‘可能是 DIV）,那么 DIV 的 size 应该先指明以便正确渲染。如下列显示正常：

好，让我们趁热打铁，看看完整的 layout 是怎样的。假设 ext 是一子目录叫做 ext-1.0，父目录下面的代码。

simple.html:

simple.js:

```
Simple = function() {
    return {
        init : function() {
            var mainLayout = new Ext.BorderLayout(document.body, {
                north: {
                    split: true, initialSize: 50
                },
                south: {
```

```
                    split: true, initialSize: 50
                },
                east: {
                    split: true, initialSize: 100
                },
                west: {
                    split: true, initialSize: 100
                },
                center: {
                }
            });
            mainLayout.beginUpdate();
            mainLayout.add('north', new Ext.ContentPanel('north-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('south', new Ext.ContentPanel('south-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('east', new Ext.ContentPanel('east-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('west', new Ext.ContentPanel('west-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('center', new Ext.ContentPanel('center-div', {
                fitToFrame: true
            }));
            mainLayout.endUpdate();
        }
    };
}();
Ext.EventManager.onDocumentReady(Simple.init, Simple, true);
```

加入内容

上面的例子做的 layout,除了可移动分割栏外，功能还不强大。需要加入些内容。有几种的办法加入内容。如果您直接加入内容到 DIV 中（ContentPanel 绑定的那个），ContentPanel 对象会对 div 里面的内容进行渲染。尽管试试！我们会更改 html 内容加入 center-div 中。

simple2.html:

This is some content that will display in a panel
when a ContentPanel object is attached to the div.

除此之外，还可以利用 ContentPanel 对象带有的 function 加载数据。可用的方法有几种，这里我们使用其中两种：setContent() 与 setUrl()。setContent()允许您直接从 JavaScipt 程序中插入 HTML。setUrl()，允许您从服务端得到数据加入 ContentPanel 中。

我们原来的例子中，ContentPanel 对象创建的时候是匿名的（anonymous）。这没问题，但要引用它们，你需要遍历区域管理器所分配的对象以获得引用的对象。这不是最好的办法，所有我的做法是分配一个变量给 ContentPanel 然后便可直接引用。

simple3.js:

```
Simple = function() {
    var northPanel, southPanel, eastPanel, westPanel, centerPanel;
    return {
        init : function() {
            var mainLayout = new Ext.BorderLayout(document.body, {
                north: {
                    split: true, initialSize: 50
                },
                south: {
                    split: true, initialSize: 50
                },
                east: {
                    split: true, initialSize: 100
                },
                west: {
                    split: true, initialSize: 100
                },
                center: {
                }
            });
            mainLayout.beginUpdate();
            mainLayout.add('north', northPanel = new Ext.ContentPanel('north-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('south', southPanel = new Ext.ContentPanel('south-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('east', eastPanel = new Ext.ContentPanel('east-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('west', westPanel = new Ext.ContentPanel('west-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('center', centerPanel = new Ext.ContentPanel('center-div', {
                fitToFrame: true
            }));
            mainLayout.endUpdate();
```

```
            northPanel.setContent('This panel will be used for a header');

            westPanel.setContent('        ');
            centerPanel.setUrl('index.html');
            centerPanel.refresh();
        }
    };
}();
Ext.EventManager.onDocumentReady(Simple.init, Simple, true);
```

我们现在从现有的页面动态加载内容。但是这里有个问题。若果内容页面积过大而撑破页面的话将没有意义了。我们提供了一些配置属性以解决这类问题。当 fitToFrame 为 true 时，就自动配置 autoScroll。内容一旦溢出就会出现滚动条。另外一个涉及 InternetExploer 的问题，是中间的内容的样式没有生效，原因是一些浏览器支持动态样式而一些不支持，要较好地解决上述问题，推荐使用 Iframe 标签。

用 IFRAME 标签做布局可灵活地处理，我们准备在 DOM 中直接操纵 IFRAME.这里 IFRAME 成为面板的容器，以填入中间区域的内容

设置一下 IFRAME 的滚动条并放到中间的页面。.

simple4.html:

simple4.js:

```
Simple = function() {
    var northPanel, southPanel, eastPanel, westPanel, centerPanel;
    return {
        init : function() {
            var mainLayout = new Ext.BorderLayout(document.body, {
                north: {
                    split: true, initialSize: 50
                },
                south: {
                    split: true, initialSize: 50
                },
                east: {
                    split: true, initialSize: 100
                },
                west: {
                    split: true, initialSize: 100
                },
                center: {
                }
            });
            mainLayout.beginUpdate();
            mainLayout.add('north', northPanel = new Ext.ContentPanel('north-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('south', southPanel = new Ext.ContentPanel('south-div', {
                fitToFrame: true, closable: false
            }));
```

```
            mainLayout.add('east', eastPanel = new Ext.ContentPanel('east-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('west', westPanel = new Ext.ContentPanel('west-div', {
                fitToFrame: true, closable: false
            }));
            mainLayout.add('center', centerPanel = new Ext.ContentPanel('center-div', {
                fitToFrame: true, autoScroll: true, resizeEl: 'center-iframe'
            }));
            mainLayout.endUpdate();
            northPanel.setContent('This panel will be used for a header');
            Ext.get('center-iframe').dom.src = 'index.html';
        }
    };
}();
Ext.EventManager.onDocumentReady(Simple.init, Simple, true);
```

当前的进展不错。大多数情况滚动条工作起来是很好的，但留意一样的东西， Internet Explorer 7 之前的版本，如果文档完整指明 DTD 的 DOCTYPE 标签，IE 很可能出现垂直滚动条的同时也显示水平滚动条。这个 IE 布局的一个 BUG。

现在这是个基本的 LAYOUT 和几个 ContentPanel 对象。接着我们加入一条工具栏（toolbar）到中间的 ContentPanel 对象。创建过程非常简单。由于主题的关系，我并不准备在这里详细介绍如何创建 toolbat。这是简单的创建 toolbar 的过程：

```
var simpleToolbar = new Ext.Toolbar('simple-tb');
simpleToolbar.addButton({ text: 'Scroll Bottom', cls: 'x-btn-text-icon scroll-bottom'});
simpleToolbar.addButton({ text: 'Scroll Top', cls: 'x-btn-text-icon scroll-bottom'});
```

要加入 toolbar,需要先加入 HTML 的容器，我们需要加入一些代码以创建 toolbar，然后绑定到中间的区域。toolbar 有两个按钮： Scroll Bottom 和 Scroll Top。这些按钮会滚动 IFRAME 内容到底部或是顶部。为了尽可能兼容多浏览器，我们的加入一个 function 来控制 IFRAME 文档。

simple5.html:

```
```

simple5.js:

```
function getIframeDocument(el) {
    var oIframe = Ext.get('center-iframe').dom;
    var oDoc = oIframe.contentWindow || oIframe.contentDocument;
    if(oDoc.document) {
        oDoc = oDoc.document;
    }
    return oDoc;
}

Simple = function() {
    var northPanel, southPanel, eastPanel, westPanel, centerPanel;
    return {
        init : function() {
            var simpleToolbar = new Ext.Toolbar('center-tb');
```

```
simpleToolbar.addButton({
    text: 'Scroll Bottom', cls: 'x-btn-text-icon scroll-bottom', handler: function(o, e) {
        var iframeDoc = getIframeDocument('center-iframe');
        iframeDoc.body.scrollTop = iframeDoc.body.scrollHeight;
    }
});
simpleToolbar.addButton({
    text: 'Scroll Top', cls: 'x-btn-text-icon scroll-top', handler: function(o, e) {
        var iframeDoc = getIframeDocument('center-iframe');
        iframeDoc.body.scrollTop = 0;
    }
});
var mainLayout = new Ext.BorderLayout(document.body, {
    north: {
        split: true, initialSize: 50
    },
    south: {
        split: true, initialSize: 50
    },
    east: {
        split: true, initialSize: 100
    },
    west: {
        split: true, initialSize: 100
    },
    center: { }
});
mainLayout.beginUpdate();
mainLayout.add('north', northPanel = new Ext.ContentPanel('north-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add('south', southPanel = new Ext.ContentPanel('south-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add('east', eastPanel = new Ext.ContentPanel('east-div', {
    fitToFrame: true, closable: false
}));
mainLayout.add('west', westPanel = new Ext.ContentPanel('west-div', {
```

```
                fitToFrame: true, closable: false
            }));
            mainLayout.add('center', centerPanel = new Ext.ContentPanel('center-div', {
                fitToFrame: true, autoScroll: true, resizeEl: 'center-iframe', toolbar: simpleToolbar
            }));
            mainLayout.endUpdate();
            northPanel.setContent('This panel will be used for a header');
            Ext.get('center-iframe').dom.src = 'index.html';
        }
    };
}();
Ext.EventManager.onDocumentReady(Simple.init, Simple, true);
```

一个标准的 layout 已经差不多了。区域可设置标题，这样可以把几个区域区分开来，创建该区域面板的时候指定属性即可。

所有的区域都可以收缩和展开。要使一个区域可收缩，你应在区域配置项中指定 collapsible 属性。属性 collapsedTitle 是用于区域收缩之后显示的文字，collapsedTitle 属性只用于 north 和 south 区域。

simple6.js:

```
function getIframeDocument(el) {
    var oIframe = Ext.get('center-iframe').dom;
    var oDoc = oIframe.contentWindow || oIframe.contentDocument;
    if(oDoc.document) {
        oDoc = oDoc.document;
    }
    return oDoc;
}

Simple = function() {
    var northPanel, southPanel, eastPanel, westPanel, centerPanel;
    return {
        init : function() {
            var simpleToolbar = new Ext.Toolbar('center-tb');
            simpleToolbar.addButton({
                text: 'Scroll Bottom', cls: 'x-btn-text-icon scroll-bottom', handler: function(o, e) {
                    var iframeDoc = getIframeDocument('center-iframe');
                    iframeDoc.body.scrollTop = iframeDoc.body.scrollHeight;
```

```
        }
    });
    simpleToolbar.addButton({
        text: 'Scroll Top', cls: 'x-btn-text-icon scroll-top', handler: function(o, e) {
            var iframeDoc = getIframeDocument('center-iframe');
            iframeDoc.body.scrollTop = 0;
        }
    });
    var mainLayout = new Ext.BorderLayout(document.body, {
        north: {
            split: true, initialSize: 50
        },
        south: {
            split: true, initialSize: 125, titlebar: true,
            collapsedTitle: 'Status', collapsible: true
        },
        east: {
            split: true, initialSize: 100
        },
        west: {
            split: true, initialSize: 100, titlebar: true, collapsible: true
        },
        center: { titlebar: true}
    });
    mainLayout.beginUpdate();
    mainLayout.add('north', northPanel = new Ext.ContentPanel('north-div', {
        fitToFrame: true, closable: false
    }));
    mainLayout.add('south', southPanel = new Ext.ContentPanel('south-div', {
        fitToFrame: true, closable: false, title: 'Status'
    }));
    mainLayout.add('east', eastPanel = new Ext.ContentPanel('east-div', {
        fitToFrame: true, closable: false
    }));
    mainLayout.add('west', westPanel = new Ext.ContentPanel('west-div', {
        fitToFrame: true, closable: false, title: 'Navigation'
    }));
    mainLayout.add('center', centerPanel = new Ext.ContentPanel('center-div', {
```

```
                fitToFrame: true, autoScroll: true, resizeEl: 'center-iframe',
                toolbar: simpleToolbar, title: 'Content'
            }));
            mainLayout.endUpdate();
            northPanel.setContent('This panel will be used for a header');
            Ext.get('center-iframe').dom.src = 'index.html';
        }
    };
}();
Ext.EventManager.onDocumentReady(Simple.init, Simple, true);
```

注意我们收藏起 west 区域时，是没有 title 的。当前的 HTML 没有提供对一个元素的 90 度的旋转。我们只好用一张透明的图片来实现，上面的文字是'Navigation',宽 18p,高 150p,然后 90 度旋转。

为了显示图片，我们需要增大 EXT 原先的 widget 样式，只须在 HTML 头样式表中加入下列样式便可得到适合的样式效果。如 simple7.html 示。

```
.x-layout-collapsed-west {
    background-image: url(navigation.gif);
    background-repeat: no-repeat;
    background-position: center;
}
```

开始使用 Grid

本文涉及的范例代码，可以在 这里下载。一个有效的例子可在 这里找到。

步骤一 定义数据(Data Definition)

首先要让 Grid 知道 XML 文档定义了每一行是什么数据。正如所见，我们命名了"item"在下面的 XML 样本中。

单行的 XML 样本数据

```
<Item>
    <ASIN>0446613657</ASIN>
    <DetailPageURL>
        http://www.amazon[*SNIP*]JVQEG2
    </DetailPageURL>
    <ItemAttributes>
```

```
        <Author>Sidney Sheldon</Author>
        <Manufacturer>Warner Books</Manufacturer>
        <ProductGroup>Book</ProductGroup>
        <Title>Are You Afraid of the Dark?</Title>
    </ItemAttributes>
</Item>
```

接着需要定义某一列为“统一标识（Unique Identifier）”,即 ID,根据“id"所设置的那个节点来读取值（样本代码第九行）。本例中的样本数据是”ASIN"列。

数据定义的最后部分就是指定你需要显示的字段(Fields),把这些字段放到一个数组之中，并保证这些字段与你的 XML 数据中元素名称是一致的，而且还要注意先后顺序，这里的顺序不需要和 XML 文件中顺序一致。

```
var dataStore = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url: 'sampledata-sheldon.xml'}),
    reader: new Ext.data.XmlReader({
        record: 'Item',
        id: 'ASIN'
    }, [
        'Author', 'Title', 'Manufacturer', 'ProductGroup'
    ])
});
```

步骤二 列模型（Column Model）

下一步便是定义 Column Model 列模型。简单地说，就是通过一些属性的设置，决定每一列怎么控制或怎么显示，这是一个由每列的配置参数组成的数组。注意，出现的顺序应该与刚才定义“字段”的数组一致。较常见的参数通常是 header 和 width，所以你会觉得这两项是必须要设置的，然而宽度（width）其实不总是需要的，因为稍后我们将使用 autoWidth/Height 来代替。

列模型样本

```
var colModel = new Ext.grid.ColumnModel([
        {header: "Author", width: 120, dataIndex: 'Author'},
        {header: "Title", width: 180, dataIndex: 'Title'},
        {header: "Manufacturer", width: 115, dataIndex: 'Manufacturer'},
        {header: "Product Group", width: 100, dataIndex: 'ProductGroup'}
]);
```

最后是将 DataStore 和 Column Model 两样东西传入到 Grid，渲染结果，然后加载来自 DataStore 的数据，这些就是你让 Grid 工作起来的最基本因素！

进行渲染！

```
var grid = new Ext.grid.Grid('mygrid', {
        ds: dataStore,
        cm: colModel
});
grid.render();

dataStore.load();
```

Grid 组件的简易分页

读者应先下载本例涉及的示范代码。这里是一个已经完成好的例子。

Grid 数据

Grid 的分页必须依靠服务端（Server Side）来划分好每一页的数据才可以完成。

本例中的服务端事 PHP,数据库是 MySQL，用来导出一些随机的数据。下列脚本的作用是，获取我们想要的数据，同时这些数据事已分好页的数据。分页的参数是由 Page Toolbar 传入的变量 limit 和 start 所决定的。

```
$link = mysql_pconnect("test-db.vinylfox.com", "test", "testuser")
    or die("Could not connect");
mysql_select_db("test") or die("Could not select database");

$sql_count = "SELECT id, name, title, hire_date, active FROM
random_employee_data";
$sql = $sql_count . " LIMIT ".$_GET['start'].", ".$_GET['limit'];

$rs_count = mysql_query($sql_count);

$rows = mysql_num_rows($rs_count);

$rs = mysql_query($sql);

while($obj = mysql_fetch_object($rs))
{
    $arr[] = $obj;
}
```

```
Echo
$_GET['callback'].'({"total":"'.$rows.'","results":'.json_encode(
$arr).'})';
```

由于每个后台开发的环境都不尽相同，所以这里的服务端代码就不细究了。

怎么做一个分页的 Grid

本例采用的是 ScriptTagProxy，原因是 范例代码 和 服务端代码 不是在同一个服务器上（译注：即“跨域”），而大多数的情况是，在同一个服务器上得到数据，直接用 HttpProxy 就可以了。

使用 DataStore 与平时唯一不同的地方，便是需要设置 totalProerty 属性。本例中，我们从服务端的脚本计算出“total”这个值，告诉 DataStore 总共有多少个记录，这里指的是所有的记录数。

```
    var ds = new Ext.data.Store({

        proxy: new Ext.data.ScriptTagProxy({
            url:
'http://www.vinylfox.com/yui-ext/examples/grid-paging/grid-paging
-data.php'
        }),

        reader: new Ext.data.JsonReader({
            root: 'results',
            totalProperty: 'total',
            id: 'id'
        }, [
            {name: 'employee_name', mapping: 'name'},
            {name: 'job_title', mapping: 'title'},
            {name:'hire_date', mapping:'hire_date', type:'date',
dateFormat: 'm-d-Y'},
            {name: 'is_active', mapping: 'active'}
        ])

    });
```

分页栏 Toolbar

这里加入一个分页栏到 Grid 的面板中，--差不多完成喽。

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

```
var paging = new Ext.PagingToolbar(gridFoot, ds, {
    pageSize: 25,
    displayInfo: true,
    displayMsg: 'Displaying results {0} - {1} of {2}',
    emptyMsg: "No results to display"
});
```

最后传入 start 和 limit 参数以初始化数据。

```
ds.load({params:{start:0, limit:25}});
```

花时间较多的地方是，在后台如何生成数据，以配合 Grid 的运作，一旦这些工作 OK 后，分页 Grid 再不是一件难事了。

EXT Menu 组件

这篇教程中，我们将学习使用 Ext 的菜单器件（Menu Widgets）。假设读者已经阅读过 Ext 简介 一文，并懂得一些 Ext 的相关基本知识。菜单器件是 Ext 库中较晚实现的组件。它由几个类联合构成，使得创建一个菜单只需若几行代码。

开始！

第一步要做的是，下载本教程的示例 zip 文件 Zip 文件包括三个文件：ExtMenu.html,、ExtMenu.js、ExtMenu.css、和 list-items.gif。解包这四个文件到 Ext 的安装目录中（例如，Ext 是在“C:\code\Ext\v1.0”中，那应该在"v1.0"里面新建目录“menututorial”。双击 ExtMenu.htm，接着你的浏览器打开启动页面，应该会有一条消息告诉你配置已完毕。如果是一个 Javascript 错误，请按照页面上的指引操作。

在你常用的 IDE 中或文本编辑器中，打开 ExtMenu.js 看看：

```
Ext.onReady(function() {
    alert('You have Ext configured correctly! We are now ready to do some Ext Menu Magic!');
 });
```

在 Ext 的介绍中，我们讨论过使用 Ext 的原因和 Ext.onReady()函数的功能。

创建简易菜单

先看看怎么做一个基本的菜单，然后再讨论代码中的各个组件和知识点。加入下列代码到 onReady 函数中：

```
var menu = new Ext.menu.Menu({
    id: 'basicMenu',
    items: [{
            text: 'An item',
            handler: clickHandler
        },
        new Ext.menu.Item({
            text: 'Another item',
            handler: clickHandler
        }),
        '-',
        new Ext.menu.CheckItem({
            text: 'A check item',
            checkHandler: checkHandler
        }),
        new Ext.menu.CheckItem({
            text: 'Another check item',
            checkHandler: checkHandler
        })
    ]
});

var tb = new Ext.Toolbar('toolbar', [{
        text:'Our first Menu',
        menu: menu   // 分配 menu 到按钮
    }
]);

function clickHandler() {
    alert('Clicked on a menu item');
}

 function checkHandler() {
    alert('Checked a menu item');
}
```

OK 仔细看看这里的代码,首先实例化一个 Menu 类为变量“menu”。然后 Menu 的构建函数接受一串的 Object Literal 作为参数，在先前的 Ext 教程中我们已经讨对 ObjectLiteral 进行了解。当前的 ObjectLiteral 就包含了我们菜单中想要的属性。第一个属性是我们分配的 ID，稍后我们可用 ID 来获取 Menu 的引用。

各种 Item 的类型

属性“items”可能是最重要的属性当中的一个。留意一下语法其实我们是将一个数组作为值（value）传到属性中去。数组里的数据就是我们想要在菜单中出现的每一项。每本例中我们放了六个菜单项，但何解每项的语法(Syntax)都不尽相同呢？第一项是一串 Object Literal,包含一组我们可配置的 name/value。Ext 的底层库会为这串 Object Literal 按其配置默认地创建 item 对象。接着第二项是 Item 对象本身，后面跟着分隔符，最后两个是单选框项。整个过程演示了 Ext Widget 如何动态的运行。下例各项可填入到数组中：

```
{text: 'Foo'}                          // Ext 转换这个配置对象到 menu item
'Straight Text'                        //文本或原始 html (纯文字)
'-'                                    // 创建分隔符
new Ext.menu.Item({text: 'Foo'})       // 创建一个标准 item (同{text: 'Foo'})
new Ext.menu.CheckItem()               // 创建单选框 item
new Ext.menu.DateItem()                // 创建 menu 内置的日历控件
new Ext.menu.ColorItem()               //创建一个颜色采集器
new Ext.menu.BaseItem(document.getElementById('my-div')) //允许你加入任何元素
```

Item 属性

Item 接受什么类型的属性呢？本例中我们使用了这两属性：

```
text: 'An item',
handler: clickHandler
```

第一个是 Item 的文本。第二个是当用户单击一单击 Item 所触发的事件处理函数（event handler function）。本例中我们将函数 clickHandler()和 checkHandler()定义在代码的最后。作为演示用途，这仅仅是用 alert()通知你有一个单击的消息。
其它有用的属性是：

```
cls: 'a-class-name'  ,        // 为标准 Item 手动设置样式和图标 ICON
icon: 'url',                  // 如不想用 CSS,可直接设置图标的 URL
group: 'name of group',       //只适用多选项，保证只有一项被选中
```

完整的 item 属性列表在 Menu Item 文档中。

在 UI 中摆放菜单

So,创建好的 Menu 对象已经有两个基本的 item 了，然而我们怎么把它们摆放到页面中呢?在 UI 中一个 menu 可以分配到多个位置（同一个对象，不同位置多次使用），这正是 Ext 如此强大的原因：每一个器件（widget）切成“一块一快”的类，来构建整个面向对象的动态结构（Structure）。这意味着 menu 可

用于不同的场合。我们可将 menu 放到有按钮的 toolbar 中，或用页面中的一个按钮来展开 menu，甚至可在 Ext 的其它器件中（widgets）使用 menu 作为上下文菜单（Context Menu）。

本例中分配一个 menu 到 toolbar 中：

```
var tb = new Ext.Toolbar('toolbar', [{
        text:'Our first Menu',
        menu: menu   // 分配 menu 到 toolbar
    }
]);
```

Ext.Toolbar 构建函数接受两个参数，第一个是指定 toolbar 的容器（contrainer）；第二个参数是包含全部按钮的数组。这里，我们只使用一个按钮，正如所见，按钮实质上是一串由数组组成的 Object Literal，同时这个 Object Litetal 亦包含了不同的属性。下面是一组按钮对象的 Object Litetal 属性：

```
cls: 'a-class-name'   ,        //手动设置样式和图标 ICON
icon: 'url',                   // 如不想用 CSS,可直接设置图标的 URL
text:'Our first Menu',         // 按钮提示之文字
menu: menu                     // 可以是 menu 之 id 或配置对象
```

Menu 的分配方式：

刚才谈到如何分配 Menu 到工具条中（toolbar），继而亦讨论 menu 分配的不同方式，看看有关的细节是怎样的。So，因为 menu 属性可以以不同方式地分配，即是可以赋予一个 menu 对象，或是已经建好的 menuID，或直接是一个 menu config 对象。你应该有机会就尝试一下以多种方式创建 menu，因为类似的方式方法在 Ext 的 Widgets 随处可见。下面的代码演示了如何用不同的方法来做出跟范例一样的效果，唯一相同的就是 menu 对象的 config。菜单中包括两个子菜单、一个 DataItem 容器、一个 ColorItem 容器。另外注意 event handing 函数需两个参数来获取事件更多的信息和知道哪个 item 被单击了。可以的话，把下面的代码也加入到 onReady 函数中，亲身体验一下：

```
var dateMenu = new Ext.menu.DateMenu({
    handler : function(datepicker, date){
        alert('Date Selected', 'You chose: '+ date.format('M j, Y'));
    }
});

var colorMenu = new Ext.menu.Menu({
    id: 'colorMenu', // the menu's id we use later to assign as submenu
    items: [
        new Ext.menu.ColorItem({
            selectHandler: function(colorpicker, color){
                alert('Color Selected', 'You chose: ' + color);
            }
        })
```

```
    ]
});

var tb = new Ext.Toolbar('toolbar', [{
        text:'Our first Menu',
        menu: {
            id: 'basicMenu',
            items: [{
                    text: 'An item',
                    handler: clickHandler
                }, {
                    text: 'Another item',
                    handler: clickHandler
                },
                '-',
                new Ext.menu.CheckItem({
                    text: 'A check item',
                    checkHandler: checkHandler
                }),
                new Ext.menu.CheckItem({
                    text: 'Another check item',
                    checkHandler: checkHandler
                }),
                '-', {
                    text: 'DateMenu as submenu',
                    menu: dateMenu, // assign the dateMenu we created above by
variable reference,
                    handler: date
                }, {
                    text: 'Submenu with ColorItem',
                    menu: 'colorMenu'     // we assign the submenu containing a
ColorItem using it's id
                }
            ]
        }
    }
]);

function clickHandler(item, e) {
```

```
    alert('Clicked on the menu item: ' + item.text);
}

function checkHandler(item, e) {
    alert('Checked the item: ' + item.text);
}
```

注意： 留意几种不同的方法加入子菜单！还有 even handing 函数和 Coloritem、DataMenu 匿名函数之间的区别。

练 一 练

Ok 我们用上述的方法，创建了 toolbar 和 menu，看起来应该是这样的：

上文提及 menu 可摆放在 UI 的任何位置，这里将为你演示 menu 如何与 Toolbars、MeneuButtons、Context Menus's 配合工作，包括一些有用的方法和动态添加的功能。

MenuButton

```
new Ext.MenuButton('menubutton', {text:'Menu Button 1', menu: dateMenu});
```

动 态 添 加 菜 单 按 钮 到 Toolbar

这条 Toolbar 有两个按钮。一个分隔符，和一个纯图标的按钮（附 Quicktips）。你可尝试这样做，把 zip 文件中.gif 加入

```
Ext.QuickTips.init();

tb.add('-', {
    icon: 'list-items.gif', //图标可单行显示
    cls: 'x-btn-icon',        // 纯图标
```

```
    tooltip: 'Quick Tips
提示文字'
});
```

更 方 便 的 是

一些代码片段，有助你提高效率，留意注释！

```
// Menus 更多的 API 内容
// 动态 增、减元素

menu.addSeparator(); //动态加入分隔符

var item = menu.add({
    text: 'Dynamically added Item'
});

// items 完整支持 Observable API
item.on('click', onItemClick);

// items can easily be looked up
menu.add({
    text: 'Disabled Item',
    id: 'disableMe'  // <-- 设置 ID 便于查找 lookup
    // disabled: true    <-- 先不 disabled 而采用下面的方式
});

// 用 id 或 index 访问
menu.items.get('disableMe').disable();
```

下 一 步 是

现在你已经了解菜单组件是如何工作了。下面的资源有助您进一步更深入学习菜单：

模板（Templates）起步

首先建议读者先下载本例的代码，以配合文本的说明。有效的例子在这里。

第一步 您的 HTML 模板

第一个步骤没有任何特别，这里的 HTML 可以说是用来格式化你的数据。花括号里面的关键字就是你数据中的{id},{url} 和{text}的容器（placeholder） 。或者用纯数字{0},{1},{2}来表示,但是关键字的命名

方式会使你的代码更可读。

现在我们加载 html 模板，创建一个模板对象（第五行），然后进行编译（第六行）。尽管编译模板不是必须的，但是一般情况下总能改善性能的。

```
var html = '<a id="{id}" href="{url}" class="nav">{text}</a><br />';

var tpl = new Ext.Template(html);
tpl.compile();
```

第二步，将数据加入到模板中

这里我们将使用 append 方法加入两行的数据。正如你所见，元素的“id”、“url”和“text”相对应于上述模板的容器。

```
tpl.append('blog-roll', {
    id: 'link1',
    url: 'http://www.jackslocum.com/',
    text: "Jack's Site"
});
tpl.append('blog-roll', {
    id: 'link2',
    url: 'http://www.extjs.com/',
    text: "Jack's New Site"
});
```

这就是模板的基本知识点，对于你来这说非常简单吧？

下一步

如果你想换个地方，文档区便是一个绝佳的好地方。看看范例 Feed Viewer，里面就大量使用了模板。

学习利用模板（Templates）的格式化功能

本教程基于 Ext 的模板引擎展开详述，亦是对 Shea Frederick“模板入门”教程一文的补充。假设读者已经初步接触过模板（Templates）,和格式化函数的基本语法为“{VARIABLE:[(可选的参数)]}”。

正式开始

假设我们打算从一变量中，打印出内容，但当中的内容有可能会占用过多的空间。对于这种情况，

通过的办法是对该内容截取，限制在 50 个英文字符内，然后做成连接让用户点击后观察全文。函数"ellipsis"的功能正是这样，可限制在任意字符数内。另外，在截取字符串的后面，该函数还会加上“...“，以示实际的内容还有更多。

一个模板如下示

```
var myTpl = new Ext.Template(’
{content:ellipsis(50)}
Read More
’);
```

通过处理，其中有 47 个字符是属于内容本身的，另外三个字符是”...“, 一共 50 个字符。

这是一份可用于模板格式化函数的列表：

- ellipsis(length) - 对大于指定长度部分的字符串，进行裁剪，增加省略号（“...”）的显示。适用于只显示前 N 位的字符，然后提供详细页面的场合。
- undef -检查一个值是否为 underfined，如果是的转换为空值
- htmlEncode - 转换(&, <, >, and ') 字符
- trim - 对一段文本的前后多余的空格裁剪
- substr(start, length) - 返回一个从指定位置开始的指定长度的子字符串。
- lowercase - 返回一个字符串，该字符串中的字母被转换为小写字母。
- uppercase - 返回一个字符串，该字符串中的字母被转换为大写字母。
- capitalize - 返回一个字符串，该字符串中的第一个字母转化为大写字母，剩余的为小写。
- usMoney - 格式化数字到美元货币。如：$10.97
- date[(format)] - 将一个日期解析成为一个特定格式模式的日期。如日期字符串不输入，默认为“月/日/年”
- stripTags - 剥去变量的所有 HTML 标签

您亦可以创建自定义的格式化函数，具体做法是，在模板的实例上加入新的方法，继而在模板上调用，格式化的函数应该像这样的：{VARIABLE:this.}"

这是一个简单的实例，对模板实例加入一个"yesNoFormat "的新函数。yesNoFormat 与 ColdFusion 转换”truthy“函数相类似 ，如果是真的输出"Yes",假的输出"No"。

```
var testCustomTpl = new Ext.Template('
User: {username} IsRevoked: {revoked:this.yesNoFormat}
');
testCustomTpl.yesNoFormat = function(value) {
        return value ? 'Yes' : 'No';
};
testCustomTpl.append(document.body, {username: 'aconran', revoked: 1});
```

下一步

关于怎么学好 EXT 这个框架我的看法是，在您熟悉的 IDE 中打开源码进行阅读。保证阁下一定会收获不少技巧和写代码的好习惯，而且极有可能发现新的大陆，还是没有归档的。熟悉模板 Templates 的简单用法和格式化功能后，就可进入下一步的学习：MasterTemplates。 MasterTemplates 提供了处理”子模板“的功能，以方便从数据库循环数据，同时亦包含模板（Templates）的所有功能。

事件处理

在 Javascript 中，你将不得不经常进行事件的处理。这有时很难顺利进行，因为你需要进行不同的跨浏览器标准化事件处理。而 ExtJs 使得处理事件变得非常容易，有时候甚至还富于乐趣(!)。

非常基础的例子

想象一下这样一个例子，当用户点击一个链接时，你想向他显示一则警告信息。请继续往下看，因为在开始处理事件前你也许想知道更多。

```
var el = Ext.get('somelink');
el.on('click', function(){
    alert('you click on a link');
});
```

注意，在这里我们使用了一个匿名处理函数。另外，你应该在 DOM 初始化后才执行上述代码（使用 Ext.onReady()方法）

处理函数的作用域

好了，我们刚刚学习了最基础的事件处理。让我们看看其他一些我们能做的事。默认情况下，处理函数内的作用域是你绑定事件的元素。

```
var el = Ext.get('somelink');
el.on('click', function(){
    alert(this.id); // 这里将显示'somelink'
});
```

注意 this 不是 Ext Element 对象。如果你想使用 Ext 的方法你必须使用"var el = Ext.get(this);"

但如果我们想要改变处理函数内的作用域呢？你可以把那个对象作为作用域参数。

```
function onClick = function(){
    alert(this.someProperty); // 这里将显示'someValue'
};

var scope = {
    someProperty : 'someValue'
}
```

```
var el = Ext.get('somelink');
el.on('click', onClick, scope);
```

提示：更多关于作用域的信息请参见这里

传 递 参 数

在前面的例子中我们看到了如何改变处理函数内的作用域。但如果我们仍然想访问（与之相绑定的）元素呢？我们可以使用传递给处理函数的参数来进行操作。

```
function onClick = function(ev, target){
    alert(this.someProperty); // 这里将显示'someValue'
    alert(target.id); // 这里将显示'somelink'
};

var scope = {
    someProperty : 'someValue'
}

var el = Ext.get('somelink');
el.on('click', onClick, scope);
```

如你所见，在这个例子中我们使用了第二个参数（target）。第一个参数是 Ext Event 对象，我们可以使用此对象来做很多事情。

类设计

Javascript 与其他的面向对象语言不同，如 C++，Java 或 PHP 等。它并不是基于类的，而是基于原型的一种语言。

对 象 创 建

在 Javascript 中创建一个类是非常容易的：

```
var myObject = {
  aVar: 15,
  aMethod: function() {
    alert("I'm a method of the object myObject." + "aVar: " + this.aVar);
  }
}
```

你不必通过定义一个类然后实例化该类来创建一个对象。我们在这里使用了一个对象构造器。它满足了使用单个对象的场合。如果我们需要使用同一个类型的多个对象，我们必须使用一个构造器函数和 *new* 关键字。

使用构造器函数

在 Javascript 中没有类的概念，但是构造器是存在的。你可以编写一个函数，然后通过 *new* 关键字来创建一个对象。

```
// 首先，我们为我们的类定义一个空的构造器
function myClass() {
  this.aVar = 15;
  this.aMethod = function() {
    alert("I'm a method of the object myObject.");
  }
}

// 创建类的实例
var A = new myClass();

// 显示 15
alert(A.aVar);

// 第二个实例
var B = new myClass();
```

方法共享

你必须使用 prototype 对象：

```
// 我们定义了一个 prototype 对象的一个方法
myClass.prototype.sharedMethod = function() { alert("I'm a shared method") }

// 显示我们的信息
A.sharedMethod();

// 相同的信息
B.sharedMethod();
```

There is no method named *sharedMethod* in the myClass definition. Javascript looks for a method with this name in the prototype object of myClass and calls it if it exists. 在 myClass 定义中并没有一个名为 *sharedMethod* 的方法。Javascript 会在 myClass 相关联的 prototype 对象中寻找与该方法名相同的方法，如果存在的话，Javascript 则调用该方法。

表单组件入门

我建议下载用于这个例子的一段程序，这样可能对你有一些帮助。你也可以找一个有效的例子。

表单体

首先要做的第一件事就是创建一个表单体，这相当于在 HTML 中书写一个 <form></form>标识。

```
var form_employee = new Ext.form.Form({
        labelAlign: 'right',
        labelWidth: 175,
        buttonAlign: 'right'
});
```

创建表单字段

表单示例由 name、title、hire_date 和 active 四个表单字段构成。开头的两个表单字段 name 和 title，只是简单的文本字段，我们会用 TextField 方法来创建它们。

重要的配置选项是 name，定义该选项与 HTML 中定义一个表单字段名几乎一样。

```
var employee_name = new Ext.form.TextField({
        fieldLabel: 'Name',
        name: 'name',
        width:190
});

var employee_title = new Ext.form.TextField({
        fieldLabel: 'Title',
        name: 'title',
        width:190
});
```

跟着的 hire_date 字段是一个日期字段，我们会用 **DateField** 方法来创建，它会为我们弹出一个别致的日期选择器来让我们选择日期。

format 配置选项被用来为 PHP 指定日期格式标准（PHP 的日期格式）。日期格式字符串的调整须与你所用的日期格式相匹配。

```
var employee_hire_date = new Ext.form.DateField({
        fieldLabel: 'Hire Date',
        name: 'hire_date',
```

```
        width:90,
        allowBlank:false,
        format:'m-d-Y'
});
```

最后一个表单元素 active 是一个布尔值，我们使用 **Checkbox** 方法来创建。

```
var employee_active = new Ext.form.Checkbox({
        fieldLabel: 'Active',
        name: 'active'
});
```

完成表单

现在，我们把表单里的所有表单字段加入到 **fieldset** 中去。当然了，如果你想在 **fieldset** 的外面进行，可以选择使用 **add** 方法。

```
form_employee.fieldset(
        {legend:'Employee Edit'},
        employee_name,
        employee_title,
        employee_hire_date,
        employee_active
)
```

最后，最不能少的就是 **submit** 按钮，它与一小段点击时进行错误检测的代码块一起被 **addButton** 方法加进来。调用 **render** 方法，传入 **div** 标识的“**id**”，然后在网页的 div 里把表单显示出来。

```
form_employee.addButton('Save', function(){
        if (form_employee.isValid()) {
                Ext.MessageBox.alert('Success', 'You have filled out the form correctly.');
        }else{
                Ext.MessageBox.alert('Errors', 'Please fix the errors noted.');
        }
}, form_employee);

form_employee.render('employee-form');
```

下一步

虽然本教程让你懂得了如何去创建一个表单，但创建出来的表单什么事情也干不了。就像一部没有引擎的小汽车——它看起来可能很漂亮，但不能让你走得更远。

为一个表单填充或提交数据

这个教程使用了在表单入门教程中使用过的 雇员信息编辑表单。如果你仍然不熟悉如何创建一个表单，你可以首先看一下这个例子。我建议下载用于这个例子的一段程序，这样可能对你有一些帮助。你也可以找一个有效的例子。

我们将经历使用表单的整个过程，从最初的从服务器获取数据填入表单，到将数据返回给服务器。在后端我使用 PHP 和 MySQL，然而这个例子对于 PHP 和 MySQL 来说并不是特殊的，而是只要求你能够从你的服务器读取和输出 JSON 数据。

让我们开始吧

首先我们必须设置表单的 url，这是一个能获得 POST 数据并将其写进我们的数据库的 PHP 脚本.

```
var form_employee = new Ext.form.Form({
        ...
        url:'forms-submit-process.php',
        ...
});
```

我们的数据包含 5 个字段：id, name, title, hire_date 和 active,这些字段可以被取回并放置到一个数据存储对象（Store）中。

以下的程序构造了一个数据存储对象，在这个时候没有数据被取回，我们的数据代理（Proxy）对象提交到一个 PHP 脚本，用来取回数据库 id 为 14 的行 并将它转换成一个 JSON 字符串。

```
employee_data = new Ext.data.Store({
proxy: new Ext.data.HttpProxy({url: 'forms-submit-getdata.php?id=14'}),
reader: new Ext.data.JsonReader({},[ 'id', 'name', 'title', 'hire_date', 'active']),
remoteSort: false
});
```

接下来要做的是设定我们的事件监听者来监察什么时候数据被载入， 这个将保证我们不会在数据被载入之前填入表单。

```
employee_data.on('load', function() {

        // data loaded, do something with it here...
```

```
});
```

当数据被载入后，我们可以取回数据并用 setValue 方法将其填入表单。这里我们用 getAt(0) 从我们的数据存储对象里重新取回第一行数据（行 zero）。

注意：这里使用的表单和表单字段在 getting started tutorial 中有定义和解释。

```
employee_name.setValue(employee_data.getAt(0).data.name);
employee_title.setValue(employee_data.getAt(0).data.title);
employee_hire_date.setValue(employee_data.getAt(0).data.hire_date);
employee_active.setValue(Ext.util.Format.boolean(employee_data.getAt(0).data.active));
```

我们将要创建提交按钮并添加到表单，记得给来源于表单字段的 POST 数据设定扩展参数。你将会发现通过行确定字段（id）对于让 php 脚本找到需要更新的行非常有用，同时为了更好的判断，还需要一个 action 判定。

我还使用 isValid 参数来保证表单在提交前符合每一个字段的要求。

```
form_employee.addButton('Save', function(){
        if (form_employee.isValid()) {
                form_employee.submit({
                        params:{
                                action:'submit',
                                id:employee_data.getAt(0).data.id
                        },
                        waitMsg:'Saving...'
                });
        }else{
                Ext.MessageBox.alert('Errors', 'Please fix the errors noted.');
        }
}, form_employee);

form_employee.render('employee-form');
```

读取我们的数据

现在我们来读取数据

```
employee_data.load();
```

这样真的能够惊人简单的创建一个可用的表单，与成对的服务器段脚本接合，就能够将数据从数据

库获取并写入修改后的数据。这些服务端脚本可以简单到只需几行而已。

EXT 中的继承

Ext 提供了这样的一个实用函数 Ext.extend（API 参考）在 EXT 框架中实现类继承的机制。这赋予了你扩展任何 JavaScript 基类的能力，而无须对类自身进行代码的修改（这里通常指的是子类，或是从它继承的，一个基类）扩展 Ext 组件这是个较理想的方法。

要从一个现有的类创建出一个新类，首先要通过一个函数声明新类的构造器，然后调用新类属性所共享的扩展方法。这些共享的属性通常是方法，但是如果要在实例之间共享数据（例如，Java 中的静态类变量），应该也一同声明。

JavsScript 并没有提供一个自动的调用父类构造器的机制，所以你必须通过属性 superclass 在你的构造器中显式调用父类。第一个参数总是 this，以保证构造器工作在调用函数的作用域。

```
MyNewClass = function(arg1, arg2, etc) {
    // 显式调用父类的构造函数
    MyNewClass.superclass.constructor.call(this, arg1, arg2, etc);
};

Ext.extend(MyNewClass, SomeBaseClass, {
  theDocument: Ext.get(document),
  myNewFn1: function() {
    // etc.
  },
  myNewFn2: function() {
   // etc.
  }
});
```

下面的一个例子是 Ext 的实际案例，用于可缩放，拖动元素，X、Y 的坐标值指定了对象可在垂直、水平方向拖动的距离。

```
// 创建新类的构造函数
Ext.ResizableConstrained = function(el, config){
     Ext.ResizableConstrained.superclass.constructor.call(this, el, config);
};

// 扩展基类
Ext.extend(Ext.ResizableConstrained, Ext.Resizable, {
     setXConstraint : function(left, right){
```

```
        // 得到父类的属性 dd 和 setXConstraint 的引用
        this.dd.setXConstraint(left, right);
    },

    setYConstraint : function(up, down){
      // 得到父类的属性 dd 和 setYConstraint 的引用
      this.dd.setYConstraint(up, down);
    }
});

// 创建新类的实例
var myResizable = new Ext.ResizableConstrained('resize-el', {
    width: 200,
    height: 75,
    minWidth:100,
    minHeight:50,
    maxHeight:150,
    draggable:true
});

//调用新方法
myResizable.setYConstraint(0,300);
myResizable.setXConstraint(0,300);
```

按照直白语言，你可以把上面的代码理解成为：”Ext.ResizableConstrained 扩展了 Ext.Resizable 并实现了这些方法”。

补充资料

Ext 2 概述

欢迎来到 Ext 2.0。在下列各章节中，你将会接触到 Ext 2.0 最新的改进，你也将会学习，有哪些新功能是为你所用的。虽然作为一份概述性的内容，本文不会讨论如何编写 Ext 2.0 应用程序个中细节，但是你可在下面提供的资源，找到你所需的内容：

- Ext 1.x to 2.0 升级指南
- Ext 2.0 API 参考
- Ext 2.0 范例

- Ext 2.0 Change Log (Coming soon)
- Ext 社区论坛

有关重大改变的几个要点

文章内容是对 2.0 新变化的综合简述。请留意 Ext 框架在从 1.x 跨越到 2.0 的过程中，经历了无数的细微改进、臭虫修正和其他的改动。 要逐一列出尚难为之，所以本文着重提及架构上有转换的主要地方，和一些全新加入的功能。本文下列的各部分将完整解释这每一项的细节。

- **组件模型 Component Model**

  在 1.x 中就有 Component 和 BoxComponent 两个类了，但却没有深入整合到框架中去。到 2.0，这两个类得到极大的改进并是一切主要组件的基础。 尽管这些类对于开发者而言一般是尽量隐藏细节的，不过打好组件生存周期的基础知识有利于下一步的 Ext 学习。参阅详细。

- **容器模型 Container Model**

  有几个核心类可用于器件（widgets）的构建和包含其它组件的布局。 **容器 Container** 为可容纳对象和组件的布局提供一个基础性的构成方式，对于整个 Ext 框架可视化必不可少。 **面板 Panel** 扩展自容器类，为用户程序提供特定功能的 UI 基类，而且可以说是容器结构层次中最常用的类。 **窗口 Window** 是面板的一种特殊类型，使得 web 应用程序如桌面式（desktop-style）那样。**视见区 Viewport** 是专为全屏幕 web 程序应用而设计的实用容器 。参阅详细

- **布局 Layouts**

  1.x 中的布局方式围绕在 BorderLayout 和其相关的几个类。2.0，布局的整体架构建立在新容器类和崭新的布局类上。 BorderLayout 现加入到九种风格布局之中。布局类已经是全部重写设计并考虑最大的可扩展性。 布局的管理亦受益于 2.0 的框架，去掉一些开发者之前需要面对的复杂实现。参阅详细

- **Grid**

  Grid 组件往往都被认为是 Ext 的核心组件之一，在 2.0 的版本同时继续演进。新版的用户界面更友好，性能更佳，功能上新加了行摘要、行归组、和一些依靠插件实现的功能如 expandable rows 和 row numbering 等等更多。参阅详细

- **模板 XTemplate**

  1.x 的模版类处理一些简单的模版时令人放心，但对于高级的输出任务就缺乏关键的支持。 在 2.0 中，全新的 XTemplate 可支持子模版，数组处理，直接代码执行，逻辑判断和更多有用的功能。参阅详细

- **数据视图 DataView**

  1.x 的模版将数据绑定到模版以生成制定的 UI 视图。JsonView 是快速绑定 JSON 数据辅助类。2.0 的 DataView 把以上两种方式作统一的处理，不同的是它继承自 BoxComponent，可更好地支持各种布局方式，新的 XTemplate 类为模版处理提供强大的支持。参阅详细

- **其它新组件**

  这些新组件包括动作 Action、CycleButton、 Hidden (field)、 ProgressBar 和 TimeField。参阅详细

补充说明

- **主题**

  2.0 支持开箱即用的主题，使用为更简化。Ext 1.x 支持四套主题，但 2.0 减少到两套。 如打算

自定义 Ext 的主题，那么 Gray 主题就是一份不错的蓝本，另外一些 2.0 社区主题也可以提供一些思路或直接使用。 这不是 API 改动的一部分，但是有需要在这里提及一下。

- **突破性进展**

  令人，2.0 的一些改动无法做到向后兼容。因为相关的组件和渲染模型已经是从根本上进行修改，许多现有的组件必须舍弃旧 1.x 的方式重写编写，与 1.x 的差别较大。 我们提供的 1.x 到 2.0 升级指南希望能解决升级现有 Ext 1.x 程序的困难。

组件模型 Component Model

组件概述

2.0 的一个目标就是希望能以简单的代码块构建程序，甚至比以前更简单。 组件 Component 类最初在 1.x 引入，却没有全面整合到框架中去。在 2.0 中，组件所赋予的能力有长足的改进和提升，使得其成为整个架构里最为基础的一个类。组件对象为组件的创建、渲染、事件处理、状态管理和销毁提供了统一的模型，Ext 下面的每一个组件具备了这些由组件对象扩展出来的特性。这是 2.0 组件对象的关键特性：

- **显式声明构建器链和重写 Explicit constructor chaining and overriding**

  组件会将一个基础构造器连同配置传入到子类中。函数 `initComponent` 用于提供制定的构造器逻辑，只要在继承链上的某一个子类实现便可，所有的组件都遵从此方式。此时的子类就可在 `initComponent` 中对其设置相关的属性，实现具体的功能。

- **可调控的渲染 Managed rendering**

  2.0 中，每个组件都支持延时渲染（lazy rendering），又称按需渲染（on-demand rendering）。渲染的调控是自动为你完好的。即使如此，你亦可以通过的 beforerender 和 render 事件控制渲染发生、结束，达到最为灵活的自定义调控。

- **可调控的销毁 Managed destruction**

  每一个组件具有 destroy 的函数，当组件不再需要时，Ext 就负责组件的结束调控，如自动垃圾回收和摧毁组件元素。当然，销毁亦提供相应的事件，如 beforedestroy 和 destroy 可按照实际的情况作出逻辑处理。

- **管理声明自动化 Automatic state management**

  组件内建设置和获取状态（State）的功能，只要是全局对象 StateManager 和一个状态 Provider 都初始化好，那么多数的组件都具有自动状态管理的能力。

- **组件常规行为的统一接口 Consistent interface for basic component behavior**

  一般常规的行为如隐藏、显示和激活、禁用均是组件的基本特性。如需要，这些都可由子类去重写或制定。

- **由组件管理器负责的可用调配 Availability via ComponentMgr**

  Ext 的每一个组件在创建的时候就会由组件管理器登记注册，即你可随时获取任何组件，只需调用 `Ext.getCmp('id')`。

- **支持插件 Plugin support**

  现在任何的组件可以通过插件的形式来扩展了。插件实质是带有 `init` 方法的一种类。该方法会有一个单独的参数（类型为 Ext.Component）传入到其中。插件可通过组件的 plugins 配置项

指定。当组件创建时，如果有插件可用，组件就会调用每个插件上的 init 方法，传递自身的引用作为参数。 每个插件之后可调用方法或响应组件的事件以实现自身的功能。

组件的生存周期 Component Life Cycle

一般来说，组件的对象架构满足了“能运行（Just Works）”这一基本要求。架构在设计上已是调控好了大多数组件是怎样处理的，而且对最终开发者是透明的。 不过，若对组件对象扩展，或是有些需要制定的地方，就要利用一定的时间去实现。 深入理解组件对象的生存周期会是非常好的帮助。下面的内容就是对基于组件的每个类，一个周期内各个重要阶段作出解释：

初始化 Initialization

1. **配置项对象生效了**

   The config object is applied.

   组件对象的构造器会把全部的配置项传入到其子类中去，并且进行下列所有的步骤。

2. **组件的底层事件创建了**

   The base Component events are created

   这些事件由组件对象负责触发。事件有 enable, disable, beforeshow, show, beforehide, hide, beforerender, render, beforedestroy, destroy （参阅 Component API 文档完整的内容）。

3. **组件在组件管理器里登记了**

   The component is registered in ComponentMgr

   initComponent 这方法总是使用在子类中，就其本身而论，该方法是一个模板方法（template method），用于每个子类去现实任何所需的构造器逻辑（any needed constructor logic）。首先会创建类，然后组件对象各层次里面的每个类都应该调用 superclass.initComponent。通过该方法，就可方便地实现（implement），或重写（Override）任意一层构造器的逻辑。

4. **状态感知进行初始化（如果有的话）**

   State is initialized (if applicable)

   如果组件是状态感知的，其状态会进行刷新。

5. **加载好插件（如果有的话）**

   Plugins are loaded (if applicable)

   如果该组件有指定任何插件，这时便会初始化。

6. **渲染好插件（如果有的话）**

   The component is rendered (if applicable)

   如果指定了组件的 renderTo 或 applyTo 配置属性，那么渲染工作就会立即开始，否则会延时渲染，即等待到显式控制显示，或是其容器告知其显示的命令。

渲染过程 Rendering

1. **触发 beforerender 事件 The beforerender event is fired**

   这是个可取消的事件，指定的 handler 可阻止组件进行渲染

2. **设置好容器 The container is set**
   如果没有指定一个容器，那么将使用位于 DOM 元素中组件的父节点作为容器。

3. **调用 onRender 方法 he onRender method is called**
   这是子类渲染最重要的一个步骤，由于该方法是一个模板方法（template method），用于每个子类去现实任何所需的渲染逻辑（any needed render logic）。首先会创建类，然后组件对象各层次里面的每个类都应调用 superclass.onRender。通过该方法，就可方便地实现（implement），或重写（Override）任意一层渲染的逻辑。

4. **组件是“隐藏”状态的 The Component is "unhidden"**
   默认下，许多组件是在 CSS 样式类如"x-hidden"设置隐藏的。如果 autoShow 所配置的值为 true，这时就不会有这个"hide"样式类作用在该组件上

5. **自定义的类、样式生效了 Custom class and/or style applied**
   一切组件的子类都支持 cls 和 style 两种特殊的配置属性，分别用于指定用户自定义的 CSS 样式类和 CSS 规则。 推荐指定 cls 的方法来制定组件及其各部分的可视化设置。由于该样式类会套用在组件 makeup 最外的一层元素，所以标准 CSS 规则会继承到组件下任何子元素的身上。

6. **触发 render 事件 The render event is fired**
   这是组件通知成功渲染的一个步骤。这时，你可肯定认为组件的 DOM 元素已经是可用的了。如果尝试在组件之前访问组件，会报告一个不可用的错误。

7. **调用了 afterRender 方法 The afterRender method is called**
   这是另外一个实现或重写特定所需的“后渲染”逻辑的模板方法。每个子类应调用 superclass.afterRender.

8. **组件被隐藏或禁用（如果有的话） The Component is hidden and/or disabled (if applicable)**
   配置项 hidden 和 disabled 到这步生效

9. **所有状态感知的事件初始化(如果有的话） Any state-specific events are initialized (if applicable)**
   状态感知的组件可由事件声明特殊加载和保存状态。如支持，加入此类的事件。

销毁过程 Destruction

1. **触发 beforedesroy 事件 The beforedestroy event is fired**
   这是个可取消的事件，指定的 handler 可阻止组件被销毁。

2. **调用了 beforeDestroy 方法 The beforeDestroy method is called**
   这是另外一个实现或重写预定销毁逻辑的模板方法。每个子类应调用 superclass.beforeDestroy。

3. **元素及其侦听器被移除 Element and its listeners are removed**
   若组件是渲染好的，所属的元素的事件侦听器会被移除和元素本身会从 DOM 那里移除。

4. **调用了 onDestroy 方法 The onDestroy method is called**
   这是另外一个实现或重写特定所需的“后销毁”逻辑的模板方法。每个子类应调用 superclass.onDestroy。**注意** 容器类（Container class，和一切容器的子类）提供了一个默认的 onDestroy 实现，自动遍历其 items 集合里的每一项，分别地执行子组件身上的 destroy 方法。

5. **在组件管理器中撤销组件对象的登记 Component is unregistered from ComponentMgr**
   I 使得不能再从 Ext.getCmp 获取组件对象
6. **触发 destroy 事件 The destroy event is fired**
   这是组件成功销毁的一个简单通知。此时组件已经 DOM 中已是没有的了
7. **组件上的事件侦听器被移除 Event listeners on the Component are removed**
   组件本身可允许从所属的元素得到事件侦听器。如有的话，连同删除。

组件的 X 类型 XTypes

XTypes 是 Ext 2.0 中新的概念，被认为是 Ext 组件的特定类型。可用的 xtype 摘要可在 Component 类 API 开始的地方找到。与一般 JavaScript 对象用法相似，XTypes 可用于查找或比较组件对象，如 isXType 和 getXType 的方法。 你亦可以列出任意组件的 xtpye 层次表，用方法 getXTypes。

然而，如何把 Xtypes 用于优化组件的创建和渲染过程才是 XTypes 发挥威力的地方。 通过指定一个 xtype 的配置对象的写法，可隐式声明的方式创建组件，使得如果没有渲染的需要就只是一个对象而免去实例化的步骤，这时不仅渲染的动作是延时的，而且创建实际对象的这一步也是延时的，从而节省了内存和资源。 在复杂的布局中，这种性能上的改进尤为明显。

```
//显式创建所容纳的组件
var panel = new Ext.Panel({
    ...
    items: [
        new Ext.Button({
            text: 'OK'
        })
    ]
};

//使用 xtype 创建
var panel = new Ext.Panel({
    ...
    items: [{
        xtype: 'button',
        text: 'OK'
    }]
};
```

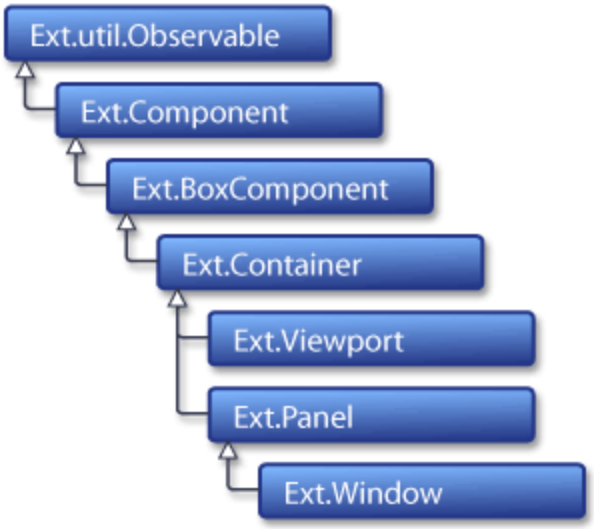
第一个例子中，面板初始化的同时，按钮总是会立即被创建的。如果加入较多的组件，这种方法很可能界面的渲染速度。第二例子中，按钮直到面板真正在浏览器上显示才会被创建和渲染。

如果面板从未显示（例如有个 tab 一直是隐藏的），那么按钮就不会被创建，不会消耗任何资源了。

BoxComponent

BoxComponent 是另外一个重要的基类，该类从组件 Component 扩展，为任何要进行可视渲染和参与布局的组件提供了一致的、跨浏览器的箱子模型（Box Model）实现。BoxComponent 负责调节大小和定位，自动处理各浏览器之间的差异，如外/内补丁、边框的问题，形成一个统一的箱子模型，以支持各种浏览器。2.0 的一切容器类（container）扩展自 BoxComponent。

容器模型 Container Model



Ext 2.0 Component/Container Class Hierarchy

容器 Container

一个组件如果有*包含其它的组件*，那么，容器 Container 便是这个组件奠基之石。该类提供了布局方面和调节大小、嵌套组所需的逻辑，并且提供一个基础性的加入组件协调机制。容器类不应该直接使用，其目的在于为一切可视的容器组件提供一个基类。

面板 Panel

面板 Panel 是 2.0 最常用的容器，90%布局任务都离不开面板。面板用在排版布局上，如同一张白纸，完全是空白的矩形，没有可视内容。虽然这样，面板也提供了一些预留区域， 方便加入程序所需的 UI 界面，包括顶部或底部的工具条、菜单栏、头部区域、底部区域和躯干区域。同时内建可展开和可收缩行为的按钮，和其它不同任务预设的按钮。面板可轻松地下降到任意的容器或布局，当中定位或渲染的逻辑全部由 Ext 调控，

下列是 Ext 2.0 面板最主要的几个子类：

- GridPanel
- TabPanel

- TreePanel
- FormPanel

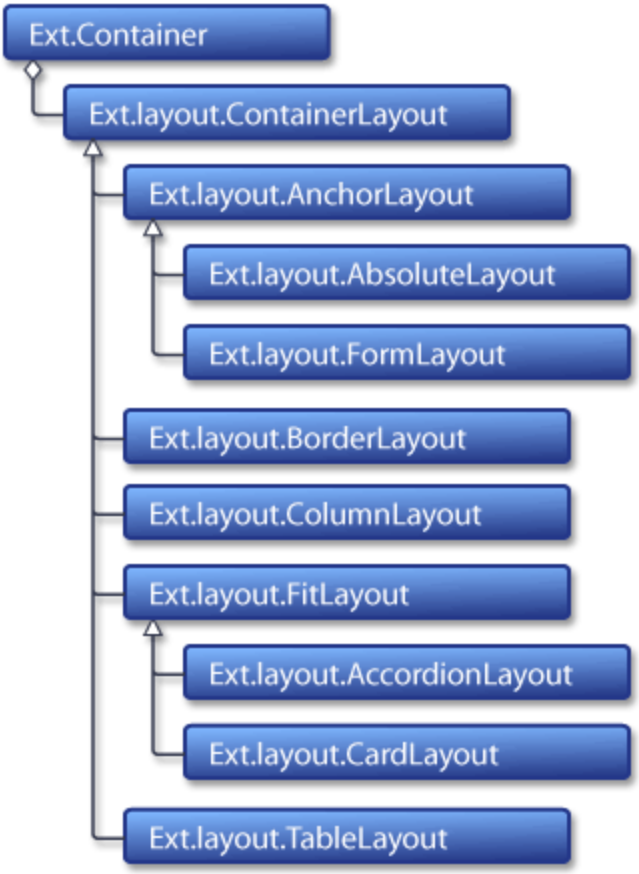
Window

Window 是一种可浮动的、可最小/最大化的、可复原的、可拖动的..等此类的特殊面板。其目的在于实现一种具有桌面风格的程序界面的基类，像 Ext 桌面演示看到的那样。

视见区 Viewport

视见区 Viewport 是以 document.body 作容器的实用类，它会与浏览器视图自适应尺寸，是全屏幕应用的基础，如浏览器大小调节、布局重新计算的问题由该类自动完成。 **注意视见区 Viewport 除了 document.body 不能渲染在其它的容器上，所以一个页面只能有一个视见区。**

布局 Layouts



Ext 2.0 Layout Class Hierarchy

引言

2.0 中最具意义的改进之一。在创建优雅的程序布局时感受到易用性或灵活性方面带来的好处。在 Ext 1.x，布局的开发集中围绕在 BorderLayout、Region 和 ContentPanel 几个类。 1.x BorderLayout 已经可以方便地生成 UI，但要真正创建属于自己的布局，还是没有足够的支持。 创建复杂的布局通常需要手工编写一些代码应付滚动条、嵌套和某些怪癖的问题。

Ext 2.0 带来了一个重写编写的、企业级应用的布局管理系统。 共有 10 种风格的布局管理器，分别提供构建各种可能的程序布局基础。Ext 调控了布局诸如 size、定位、滚动条和其他的属性方面的问题，一如既往地简单，开箱即用。在容器也可无限嵌套布局、混合其他不同风格的布局。

布局由容器内置创建，所以布局不应通过关键字 **new** 实例化这种方式直接使用。 有一种内部的机制，容器与布局能够很好地协调工作—只需配置好相关的参数，容器就会委托其负责的布局类工作。创建容器的时候，你应选定一种布局的风格以及相关的配置，这两个配置是属性 layout 和属性 layoutConfig。 举例：

```
var panel = new Panel({
    title: 'My Accordion',
    layout: 'accordion',  //在这个面板中所使用的布局样式
    layoutConfig: {
        animate: true       //布局指定的配置项写在这里
    }
    // 其他 Panel 的选项
});
```

当你创建嵌套布局时，明白面板包含其他面板是很重要的，*布局中的每个面板必须指定一个布局管理器*。 多数情况你不需要指定布局的风格如“border”或“accordion”，较常见的是“fit”那一种，会自动调整大小自适应它的容器。 如果你不指定某个布局管理器，默认的是 ContainerLayout 类，不过这样很可能导致一些意料不到的情况发生，所以最好精确声明一下。

每种布局类都支持其特定的配置选项。关于布局每种配置选项可参考 API 文档。

布局管理器 Layout Managers

ContainerLayout

其它一切布局管理器的基类，容器若不指定某个布局管理器，则默认的管理器就是这个 ContainerLayout 。 ContainerLayout 没有任何的外观表示— 其主要的职责是容纳子项目、控制渲染和一些常见任务，如调节大小缓冲

CardLayout

CardLayout 将容器中的每个组件当作一个卡片来处理。在某一时间，只有一个卡片是可见的，容器象一个卡片堆栈一样工作。大多数的情况，用于向导（Wizards），制定的 tab 实现或其它多页面信息

（resize buffering）。ContainerLayout 常用于扩展制定的布局，很少实例化直接使用。详细在 API 参考.

的场合。参阅 API 参考。

AbsoluteLayout

这是一个非常简单的布局，通过 X/Y 坐标精确来定位包含各项的相关容器。参阅 API 参考.

ColumnLayout

适用于多个列并排结构的布局风格，每个列的宽度须由像素值或百分比指定，但高度自适应于内容的高度。详细在 API 参考.

AccordionLayout

AccordionLayout 包含了一组像卡片垂直方向堆栈的面板，同通过展开或收缩来显示内容在某一时间，只有一个卡片是可见的。详细在 API 参考.

FitLayout

这是一个简单的布局，主要是创建一个适应容器大小的布局区域。如没有特定的布局要求这是容器最好的默认布局。详细在 API 参考.

AnchorLayout

这是为一些固定元素相对于容器四条边的布局。元素可通过与边缘的百分比或便宜一个值来定位， and it also supports a virtual layout canvas that can have different dimensions than the physical container. 详细在 API 文档。

FormLayout

FormLayout 是为创建一张要提交数据条目的表单而设计的布局风格。注意，一般来讲，和 FormPanel 相似，该布局类都有表单提交的自动处理，你会更倾向使用前者。FormPanels **必须**指定 layout:'form'（只能一定是这样）,所以表单额外需要的一个布局将其嵌套。 参阅 API 文档。

BorderLayout

与 1.x 的 BorderLayout 的布局完全一致。布局区域支持嵌套， 滑动条面板和可关闭、微调的分隔区

TableLayout

主要目的是通过一个表格的形式划分区域。实际上也是生成一个 table 的 HTML makeup 详细在

域。对于一些典型的业务程序的首要 UI 尤为适用。详细 API 文档。

API 参考。

Grid

概述

2.0 中的 GridView 有极大的改进，而 Grid 的 UI 部分，正是由 GridView 这个类来实现的。2.0 中 GridView 最主要的新功能有：

- **效能的提升**
  GridView 的底层结构和渲染代码已在 2.0 完整重构过，并侧重考虑了效能部分。正因性能的原因，锁定列的这一功能已经取消（参阅下一节）。
- **感观（look and feel）的改进**
  和新 2.0 的主题一起， grid 的外观控制也有新变化，使得 Grid 比以前更时尚和看上去更具吸引力。
- **单行归组**
  多个行可被归组到某一指定列，由用户重新归组亦可。
- **多行组摘要**
  每一组可相应的提供一个数据摘要组
- **进阶插件支持**
  在 2.0 中，新加入插件机制。GridView 就是这种插件机制的一个典型例子。如范例中所示，Grid 优秀的功能便是依靠几个预先做好的插件。插件 RowExpander 提供了展开、收缩行的功能，插件 RowNumberer 就提供了行中数字的支持。

列锁定的注意事项（Column Locking）

有些用户或许发现 Ext 1.x 中列锁定的功能，到 2.0 因为已经取消，并可以说以后也不再支持了。列锁定（Column locking），虽然对某些用户来说有其的用途，但与 2.0 GridView 的新功能有不兼容的地方（如归组、摘要等），而且为了实现锁定会使得 Grid 渲染性能开销增大。 因此，1.x gridView 这功能的向上升级，或打补丁，均不会由官方支持。

注意：当前有为 2.0 而做的用户扩展在进行中，以实现 2.0 的列锁定，而且看上去写得还蛮不错。更多有用资讯可论坛的帖子找到。

XTemplate

XTemplate 使用了多种标签和特殊操作符支持模板，使得模板在应付复杂的数据结构时尤为健壮。这里所列出的高度概括的几项功能，欲了解完整的细节和使用方法，请参阅 XTemplate API 文档.

- 自动数组填充和作用域切换
- 可在子模板作用域内访问父级对象

- Array item index variable
- 支持数据值的简单匹配
- 自动渲染浮点型数组（不包含非对象的值）
- 基础性的条件逻辑符号 **if**
- 可执行模板中直接写好的任意语句
- 支持模板的配置属性
- 可通过配置项对象自定义模板方法
- 可用于服务端的 JavaScript 模板

DataView

从表面上看，DataView 跟 1.x 的 View 类非常相似。两者都支持模版数据渲染，data store 数据绑定和内建的选区模型和事件。但是， 随着 2.0 新架构的设计，DataView 赋予了更强大的功能。 下面是这次最重要的改动：

- **来自 BoxComponent 的继承**
  1.x View 类继承自 Observable, 作为独立组件而言工作不错， 但是它并没提供内建的机制与其他组件融合的能力。而 DataView 就是针对这种不足作出的改进，该类从 BoxComponent 继承，因此如前文所述，也具备一般组件的生存周期控制。
- **发挥了 XTemplate 之功效**
  1.x View 类采用了 1.x 本身的模版类 Template 。较好满足了 view 自身的需求，但是难以满足一些复杂的渲染任务。DataView 采用的模版类也升级到 2.0 的 XtTemplate，以应付制定 UI 可轻松应付复杂的数据。
- **新增的配置选项**
  DataView 提供了更为灵活的一几个新选项：
  - **itemSelector:** 必须是一个 DomQuery 选择符告知该类究竟如何分辨出每个 item。相比 1.x 的做法带来灵活性和速度的提高。
  - **simpleSelect:** This is a new selection mode option that enables multi-selection by clicking on multiple items without requiring the user to hold Shift or Ctrl.
  - **overClass:** 一个 CSS 的样式类，每个元素 onmouseover 和 onmouseout 时生效。
  - **loadingText:**像其他绑定 store 的 Ext 组件一样，DataView 支持标准的遮罩效果。

其它新组件

一些有趣的新组件也加入到 2.0 中去了。要了解这些新组件到底有什么新功能，最好还是看看 API 文档完整的介绍。

动作 Action

动作 Action 是一种从组件中抽象出来的可复用的“功能块”，即多个组件之间的同一功能都来自这个 ACTION 的实现。动作允许你共享句柄（handlers），配置选项和 UI 的更新，所有组件均支持动作的接口（主要是 Toolbar，Button 和 Menu 组件）。 详细在 API 文档。

CycleButton

这是一个包含复选元素菜单的特制特制的 SplitButton。当菜单子项每次被单击，按钮都会轮回一次状态，触发 change 的事件（或调用按钮的 changeHandler 函数，如果有的话）以激活菜单项。 FeedViewer 演示程序就包含了该例子— 预览窗口地址的那个按钮就是一个 SplitButton。详细在 API 文档。

Hidden (field)

这个便是 HTML 表单中隐藏域的一个简单的实现，能够以 EXT FORM 组件般操控。详细在 API 参考。

进度条 ProgressBar

1.x 中的进度条是简单地内建在 MessageBox 类中。现在已重构为单独的器件并有进一步的改进。它支持两种不同的模式（手动和自动进度），而且 LOOK and FEEL 方面可轻松制定。详细在 API 参考。

TimeField

这是下拉列表时间选取器的简单实现。详细在 API 参考。

EXT2 简介

本教程适用于 Ext 2.0 的版本，而版本 1.x 仍可找到。

无论你是 Ext 库的新手，抑或是想了解 Ext 的人，本篇文章的内容都适合你。本文将简单地介绍 Ext 的几个基本概念，和如何快速地做出一个动态的页面并运行起来，假设读者已具备了一些 JavaScript 经验和简单了解 HTML 的文档对象模型（document object model ，DOM）。

下载 Ext

如果你未曾下载过，那应从官方网站那里下载最新版本的 Ext http://extjs.com/downloads 。

因应各种的下载需求，有几种不同的可选项。通常地，最稳定的版本，是较多人的选择。下载解包后，那个 **example** 文件夹便是一个探索 Ext 的好地方！

开始！

下载示例文件

- IntroToExt2.zip

我们将讲讲怎么使用 Ext，来完成一些 JavaScript 常见的任务。如果你想自己试试，就应该先下 IntroToExt2.zip ，用来构建已下面的 Ext 代码。

Zip 包里有三个文件：**ExtStart.html**，**ExtStart.js** 和 **ExtStart.css**。解包这三个文件到 Ext 的安装目录中（例如，Ext 是在“C:\code\Ext\v2.0”中，那应该在"v2.0"里面新建目录“tutorial”。双击 **ExtStart.htm**，接着你的浏览器打开启动页面，应该会有一条消息告诉你配置已完毕。如果出现了 Javascript 错误，请按照页面上的指引操作。

现在在你常用的 IDE 中或文本编辑器中，打开 ExtStart.js 看看。

```
Ext.onReady(function() {
        alert("Congratulations!  You have Ext configured correctly!");
});
```

Ext.onReady 可能是你接触的第一个也可能是在每个页面都要使用的方法。这个方法会在 DOM 加载全部完毕后，保证页面内的所有元素能被 Script 引用（reference）之后调用。你可删除 alert()那行，加入一些实际用途的代码试试。

Element：Ext 的核心

大多数的 JavaScript 操作都需要先获取页面上的某个元素的引用（reference）,好让你来做些实质性的事情。传统的 JavaScript 做法，是通过 ID 获取 Dom 节点的：

```
var myDiv = document.getElementById('myDiv');
```

这毫无问题，不过这样单单返回一个对象（DOM 节点），用起来并不是太实用和方便。为了要用那节点干点事情，你还将要手工编写不少的代码；另外，对于不同类型浏览器之间的差异，要处理起来可真头大了。

进入 Ext.element 对象。元素（element）的的确确是 Ext 的心脏地带，--无论是访问元素（elements）还是完成一些其他动作，都要涉及它。Element 的 API 是整个 Ext 库的基础，如果你时间不多，只是想了解 Ext 中的一两个类的话，Element 一定是首选！

由 ID 获取一个 Ext Element 如下（首页 ExtStart.htm 包含一个 div，ID 名字为“myDiv”，然后，在 ExtStart.js 中加入下列语句）： The corresponding code to get an Ext Element by ID looks like this (the starter page ExtStart.html contains a div with the id "myDiv," so go ahead and add this code to ExtStart.js):

```
Ext.onReady(function() {
        var myDiv = Ext.get('myDiv');
});
```

再回头看看 Element 对象，发现什么有趣的东东呢？

- Element 包含了常见的 DOM 方法和属性，提供一个快捷的、统一的、跨浏览器的接口（若使用 Element.dom 的话，就可以直接访问底层 DOM 的节点。）；
- Element.get()方法提供内置缓存（Cache），多次访问同一对象效率上有极大优势；
- 内置常用的 DOM 节点的动作，并且是跨浏览器的定位的位置、大小、动画、拖放等等（add/remove CSS classes, add/remove event handlers, positioning, sizing, animation, drag/drop）。

这意味着你可用少量的代码来做各种各样的事情，这里仅仅是一个简单的例子（完整的列表在 Element API 文档中）。

继续在 ExtStart.js 中，在刚才我们获取好 myDiv 的位置中加入：

```
myDiv.highlight();        //黄色高亮显示然后渐退
myDiv.addClass('red');   // 添加自定义 CSS 类 (在 ExtStart.css 定义)
myDiv.center();           //在视图中将元素居中
myDiv.setOpacity(.25);   // 使元素半透明
```

获取多个 DOM 的节点

通常情况下，想获取多个 DOM 的节点，难以依靠 ID 的方式来获取。有可能因为没设置 ID,或者你不知道 ID,又或者直接用 ID 方式引用有太多元素了。这种情况下，你就会不用 ID 来作为获取元素的依据，可能会用属性（attribute）或 CSS Classname 代替。基于以上的原因，Ext 引入了一个异常强大的 Dom Selector 库，叫做 DomQuery。

DomQuery 可作为单独的库使用，但常用于 Ext，你可以在上下文环境中（Context）获取多个元素，然后通过 Element 接口调用。令人欣喜的是，Element 对象本身便有 Element.selcect 的方法来实现查询，即内部调用 DomQuery 选取元素。这个简单的例子中，ExtStart.htm 包含若干段落（

标签），没有一个是有 ID 的，而你想轻松地通过一次操作马上获取每一段，全体执行它们的动作，可以这样做：

```
// 每段高亮显示
Ext.select('p').highlight();
```

Element.select 在这个例子中的方便性显露无疑。它返回一个复合元素，能通过元素接口（Element interface）访问每个元素。这样做的好处是可不用循环和不分别访问每一个元素。

DomQuery 的选取参数可以是一段较长的数组，其中包括 W3C CSS3 Dom 选取器、基本 XPatch、HTML 属性和更多，请参阅 DomQuery API 文档以了解这功能强大的库个中细节。

响应事件

到这范例为止，我们所写的代码都是放在 onReady 中，即当页面加载后总会立即执行，功能较单一——这样的话，你便知道，如何响应某个动作或事件来执行你希望做的事情，做法是，先分配一个 function，再定义一个 event handler 事件处理器来响应。我们由这个简单的范例开始，打开 ExtStart.js，编辑下列的代码：

```
Ext.onReady(function() {
    Ext.get('myButton').on('click', function(){
        alert("你单击了按钮");
    });
});
```

代码依然会加载好页面后执行，不过重要的区别是，包含 alert()的 function 是已定义好的，但它不会立即地被执行，是分配到按钮的单击事件中。用浅显的文字解释，就是：获取 ID 为'myDottom'元素的引用，监听任何发生这个元素上被单击的情况，并分配一个 function,以准备任何单击元素的情况。

一般来说，Element.select 也能做同样的事情，即作用在获取一组元素上。下一例中，演示了页面中的某一段落被单击后，便有弹出窗口：

```
Ext.onReady(function() {
        Ext.select('p').on('click', function() {
                alert("你单击一段落;");
        });
});
```

这两个例子中，事件处理的 function 均是简单几句，没有函数的名称，这种类型函数称为“匿名函数（anonymous function）”，即是没有名的的函数。你也可以分配一个有名字的 event handler,这对于代码的重用或多个事件很有用。下一例等效于上一例：

```
Ext.onReady(function() {
        var paragraphClicked = function() {
                alert("You clicked a paragraph");
        }
        Ext.select('p').on('click', paragraphClicked);
});
```

到目前为止，我们已经知道如何执行某个动作。但当事件触发时，我们如何得知这个 event handler 执行时是作用在哪一个特定的元素上呢？要明确这一点非常简单，Element.on 方法传入到 even handler 的 function 中（我们这里先讨论第一个参数，不过你应该浏览 API 文档以了解 even handler 更多的细节）。在我们之前的例子中，function 是忽略这些参数的，到这里可有少许的改变，——我们在功能上提供了更深层次的控制。必须先说明的是，这实际上是 Ext 的事件对象（event object）,一个跨浏览器和拥有更多控制的事件的对象。例如，可以用下列的语句，得到这个事件响应所在的 DOM 节点：

```
Ext.onReady(function() {
        var paragraphClicked = function(e) {
                Ext.get(e.target).highlight();
        }
        Ext.select('p').on('click', paragraphClicked);
});
```

注意得到的 e.target 是 DOM 节点，所以我们首先将其转换成为 EXT 的 Elemnet 元素，然后执行欲完

成的事件，这个例子中，我们看见段落是高亮显示的。

使用 Widgets

(Widget 原意为"小器件"，现指页面中 UI 控件)

除了我们已经讨论过的核心 JavaScript 库，当前的 Ext 亦包括了一系列的最前端的 JavaScirptUI 组件库。文本以一些常用的 widget 为例子，作简单的介绍。

MessageBox

比起略为沉闷的"HelloWolrd"消息窗口，我们做少许变化，前面我们写的代码是，单击某个段落便会高亮显示，现在是单击段落，在消息窗口中显示段落内容出来。
在上面的 paragraphClicked 的 function 中，将这行代码：

```
Ext.get(e.target).highlight();
```

替换为：

```
var paragraph = Ext.get(e.target);
paragraph.highlight();
Ext.MessageBox.show({
        title: 'Paragraph Clicked',
        msg: paragraph.dom.innerHTML,
        width:400,
        buttons: Ext.MessageBox.OK,
        animEl: paragraph
});
```

这里有些新的概念需要讨论一下。在第一行中我们创建了一个局部变量（Local Variable）来保存某个元素的引用，即被单击的那个 DOM 节点（本例中，DOM 节点指的是段落 paragrah,事因我们已经定义该事件与

标签发生关联的了）。为什么要这样做呢？嗯...观察上面的代码，我们需要引用同一元素来高亮显示，在 MessageBox 中也是引用同一元素作为参数使用。
一般来说，多次重复使用同一值（Value）或对象，是一个不好的方式，所以，作为一个具备良好 OO 思维的开发者，应该是将其分配到一个局部变量中，反复使用这变量！

现在，为了我们接下来阐述新概念的演示，请观察 MessageBox 的调用。乍一看，这像一连串的参数传入到方法中，但仔细看，这是一个非常特别的语法。实际上，传入到 MessageBox.show 的只有一个参数：一个 Object literal,包含一组属性和属性值。在 Javascript 中，Object Literal 是动态的，你可在任何时候用{和}创建一个典型的对象(object)。其中的字符由一系列的 name/value 组成的属性，属性的格式是[property name]:[property value]。在整个 Ext 中，你将会经常遇到这种语法，因此你应该马上消化并吸收这个知识点！使用 Object Literal 的原因是什么呢？主要的原因是"可伸缩性（flexibility）"的考虑"，随时可新增、删除属性，亦可不管顺序地插入。而方法不需要改变。这也是多个参数的情况下，为最终开发者带来不少的方便（本例中的 MessageBox.show())。例如，我们说这儿的 foo.action 方法，有四个参数，而只有一个是你必须

传入的。本例中，你想像中的代码可能会是这样的 foo.action(null, null, null, 'hello')。若果那方法用 Object Literal 来写，却是这样， foo.action({ param4: 'hello' })，这更易用和易读。

Grid

Grid 是 Ext 中人们最想先睹为快的和最为流行 Widgets 之一。好，让我们看看怎么轻松地创建一个 Grid 并运行。用下列代码替换 ExtStart.js 中全部语句：

```
Ext.onReady(function() {
        var myData = [
                ['Apple',29.89,0.24,0.81,'9/1 12:00am'],
                ['Ext',83.81,0.28,0.34,'9/12 12:00am'],
                ['Google',71.72,0.02,0.03,'10/1 12:00am'],
                ['Microsoft',52.55,0.01,0.02,'7/4 12:00am'],
                ['Yahoo!',29.01,0.42,1.47,'5/22 12:00am']
        ];

        var myReader = new Ext.data.ArrayReader({}, [
                {name: 'company'},
                {name: 'price', type: 'float'},
                {name: 'change', type: 'float'},
                {name: 'pctChange', type: 'float'},
                {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
        ]);

        var grid = new Ext.grid.GridPanel({
                store: new Ext.data.Store({
                        data: myData,
                        reader: myReader
                }),
                columns: [
                        {header: "Company", width: 120, sortable: true, dataIndex:
'company'},
                        {header: "Price", width: 90, sortable: true, dataIndex:
'price'},
                        {header: "Change", width: 90, sortable: true, dataIndex:
'change'},
                        {header: "% Change", width: 90, sortable: true, dataIndex:
'pctChange'},
                        {header: "Last Updated", width: 120, sortable: true,
                                renderer: Ext.util.Format.dateRenderer('m/d/Y'),
                            dataIndex: 'lastChange'}
```

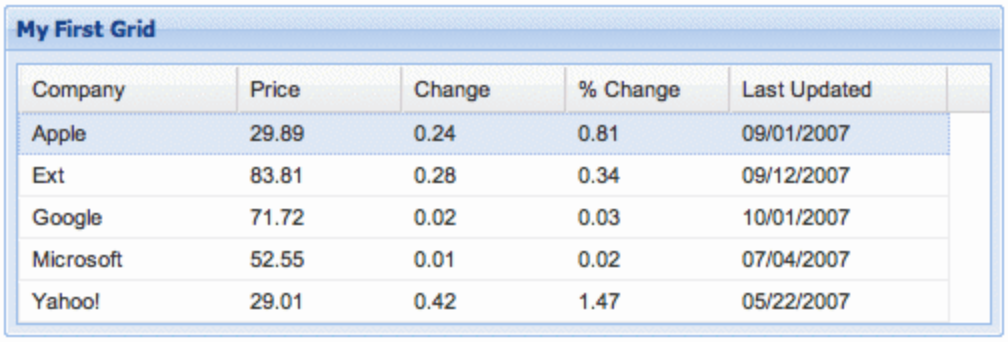
```
                ],
                viewConfig: {
                        forceFit: true
                },
                renderTo: 'content',
                title: 'My First Grid',
                width: 500,
                frame: true
        });

        grid.getSelectionModel().selectFirstRow();
});
```

这看上去很复杂，但实际上加起来，只有**四行**代码（不包含测试数据的代码）。

- 第一行创建数组并作为数据源。实际案例中，你很可能从数据库、或者 WebService 那里得到动态的数据。
- 接着，我们创建并加载 data store， data store 将会告诉 Ext 的底层库接手处理和格式化这些数据。不同的数据类型须在类 Reader 中指明。
- 接着，我们创建一个 Grid 的组件，传入各种的配置值，有：
  - 新的 data store, 配置好测试数据和 reader
  - 列模型 column model 定义了 列 columns 的配置
  - 其他的选择指定了 Grid 所需的功能
- 最后，通过 SelectionModel 告诉 Grid 高亮显示第一行。

不是太困难吧？如果一切顺利，完成之后你会看到像这样的：



当然，你现在还未掌握这段代码的某些细节，但先不要紧，这个例子的目的是告诉你，只要学习了少量的几行代码，创建一个富界面的多功能的 UI 组件是可能的。更多的 grid 细节读者可作为一种练习去学习。这儿有许多学习 Grid 的资源，Ext Grid 教程、Grid 交互演示交和 Grid API 文档。

还有更多的..

这只是冰山一角。还有一打的 UI Widgets 可以供调用，如 layouts, tabs, menus, toolbars, dialogs, tree view 等等。请探索 范例演示。

編輯 使用 Ajax

在弄好一些页面后，你已经懂得在页面和脚本之间的交互（interact）原理。接下来，你应该掌握的是，怎样与远程服务器（remote server）交换数据，常见的是从数据库加载数据（load）或是保存数据（save）到数据库中。通过 JavaScript 异步无刷新交换数据的这种方式，就是所谓的 Ajax。Ext 内建卓越的 Ajax 支持，例如，一个普遍的用户操作就是，异步发送一些东西到服务器，然后，UI 元素根据回应（Response）作出更新。这是一个包含 text input 的表单，一个 div 用于显示消息（注意，你可以在 ExtStart.html 中加入下列代码，但这必须要访问服务器）：

```
<div id="msg"></div>
<div>
   Name: <input type="text" id="name" />
   <input type="button" id="okButton" value="OK" />
</div>
<div id="msg"></div>
```

接着，我们加入这些处理交换数据的 JavaScript 代码到文件 ExtStart.js 中(用下面的代码覆盖)：

```
Ext.onReady(function(){
        Ext.get('okButton').on('click', function(){
                var msg = Ext.get('msg');
                msg.load({
                        url: 'ajax-example.php', // <-- 按实际改动
                        params: 'name=' + Ext.get('name').dom.value,
                        text: 'Updating...'
                });
                msg.show();
        });
});
```

注意： 这个例子需要 web server 才可运行。 浏览器的 URL 地址不应是以 *file://*开头，而是 *http://*开头，否则的话 Ajax 的数据交互将不会工作。Localhost 就可以工作得很好，但必须是通过 http 的。

这种模式看起来已经比较熟悉了吧！先获取按钮元素，加入一个匿名函数监听单击。在事件处理器中（event handler），我们使用一个负责处理 Ajax 请求、接受响应（Response）和更新另一个元素的 Ext 内建类，称作 UpdateManagerUpdater。UpdateManager 可以直接使用，或者和我们现在的做法一样，通过 Element 的 load 方法来引用（本例中该元素是 id 为“msg“的 div）。当使用 Element.load 方法，请求（request)会在加

工处理后发送，等待服务器的响应（Response），来自动替换元素的 innerHTML。简单传入服务器 url 地址，加上字符串参数，便可以处理这个请求（本例中，参数值来自“name”元素的 value),而 text 值是请求发送时提示的文本，完毕后显示那个 msg 的 div（因为开始时默认隐藏）。当然，和大多数 Ext 组件一样，Ext.Ajax 有许多的参数可选，不同的 Ajax 请求有不同的方案。而这里仅演示最简单的那种。

最后一个关于 Ajax 隐晦的地方就是，服务器实际处理请求和返回（Resposne）是具体过程。这个过程会是一个服务端页面，一个 Servlet，一个 Http 调度过程，一个 WebService,甚至是 Perl 或 CGI 脚本，即不指定一个服务器都可以处理的 http 请求。让人无法预料的是，服务器返回什么是服务器的事情，无法给一个标准的例子来覆盖阐述所有的可能性。。

下面的例子是一些常见的语言以方便开始测试(这段代码输出刚才我们传入'name'的那个值到客户端，即发送什么，返回什么，然后在我们刚才写的'msg' div 中加入该文本）。PHP 的已经包含在下载文件中，文件名为'ajax-example.php'，可换成你自己服务端的代码：

Plain PHP

```
<?php if(isset($_POST['name'])) {
                echo 'From Server: '.$_POST['name'];
        }
?>
```

CakePHP

```
<?php if(isset($this->data['name'])) {
                $this->flash('From Server: '.$this->data['name']);
        }
?>
```

Django

```
from django.http import HttpResponse

def ajax_request(request):
    return HttpResponse('From Server: %s' % request.POST.get('name', 'nada'))
```

Perl

```
#!/usr/bin/perl
use strict;
```

```
use warnings;
use CGI;

my $Query = new CGI;

print $Query->header();
print "Hello from : ".$Query->param('name');

exit;
```

ASP.Net

```
protected void Page_Load(object sender, EventArgs e)
{
        if (Request.Form["name"] != null)
        {
                Response.Write("From Server: " + Request.Form["name"]);
                Response.End();
        }
}
```

ColdFusion

```
<cfif StructKeyExists(form, "name")>
    <cfoutput>From Server: #form.name#</cfoutput>
</cfif>
```

JSP

```
From Server: ${param.name}
```

使用 Ajax 的真正挑战，是需要进行适当的手工编码，并相应格式化为服务端可用接受的数据结构。有几种格式供人们选择（最常用为 JSON/XML）。 Ext 没有跟任何服务器语言有独家联系，因此其它特定语言的库亦可用于 Ext 处理 Ajax 服务。只要页面接受到结果是 EXT 能处理的数据格式，Ext 绝不会干涉服务器其他的事情！推荐参阅我们提供的各个平台资源以了解服务端框架的更多资讯和辅助。

TabPanel 基础

这份教程目的在于对 TabPanel 类进行一次快速介绍。所提及的知识都是来自我对 TabPanel 范例、Ext 源码和 API 文档的学习。到本文最后，你应该完成好一个 Tab Panel，这个 TabPanel 能够：

- 创建新 tab，其内容来自一个 URL。
- 判断某个 tab 是否存在，有的话加载新内容。

Step 1: 创建 HTML 骨架

我们将会用下列 HTML，和 Ext 一齐构建一个基本的结构。复制这些内容到一个叫 *tptut.heml* 的文件，并要求是运行在服务端的，当然 也要保证 *ext-all.css*, *ext-base.js*, 和 *ext-all.js* 这些 Ext 安装路径的正确。然后按照以下步骤创建 tab_actions.js:

```
<html>
<head>
<title>TabPanel 教程</title>
<!-- Ext CSS and Libs -->
<link rel="stylesheet" type="text/css" href="../include/ext2/resources/css/ext-all.css"
/>
<script                                                    type="text/javascript"
        src="../include/ext2/adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../include/ext2/ext-all.js"></script>

<!-- Custom CSS and Libs -->
<script type="text/javascript" src="./tab_actions.js"></script>
<style>
#actions li {
        margin:.3em;
}
#actions li a {
        color:#666688;
        text-decoration:none;
}
</style>
</head>
<body>


<ul id="actions" class="x-hidden">
```

```
        <li>
                <a id="use" href="#">Use Existing Tab</a>
        </li>
        <li>
                <a id="create" href="#">Create New Tab</a>
        </li>
</ul>

<div id="tabs"></div>
</body>
</html>
```

以上代码有两个地方的元素需要注意。我们将使用"actions"(动作列表)这种简易的实现来执行 tab 的创建。"tabs"的那个 div 将用于 Tab 面板中第一个默认 tab 的容器。

Step 2: Ext 结构的构建

在刚才那个目录中创建一个文文件。就叫做作 *tab_actions.js*，加入下面 JavaScript:

```
Ext.onReady(function(){
   // 包含 actions 的菜单
    var tabActions = new Ext.Panel({
        frame:true,
        title: 'Actions',
        collapsible:true,
        contentEl:'actions',
        titleCollapse: true
    });

    // 保持 actions 菜单的父面板
    var actionPanel = new Ext.Panel({
        id:'action-panel',
        region:'west',
        split:true,
        collapsible: true,
        collapseMode: 'mini',
        width:200,
        minWidth: 150,
        border: false,
```

```
        baseCls:'x-plain',
        items: [tabActions]
    });

    // 主面板（已有 tab）
    var tabPanel = new Ext.TabPanel({
                region:'center',
                deferredRender:false,
                autoScroll: true,
                margins:'0 4 4 0',
                activeTab:0,
                items:[{
                        id:'tab1',
                        contentEl:'tabs',
                title: 'Main',
                closable:false,
                autoScroll:true
                }]
    });

    //  配置视图 viewport
    viewport = new Ext.Viewport({
        layout:'border',
        items:[actionPanel,tabPanel]});
});
```

上面的代码被套上一个 Ext.onReady 的函数,以防止页面元素未全部加载就执行代码了。接着要做的事情是将我们的动作列表（Action list）转换到名字为 **tabActions** 的那个面板，该命名是由 contentEl (content element)（内容元素）这个配置项参数所指定的。

接着，创建一个父面板 **actionPanel** 来保持菜单面板。我们已 **tabActions** 作为一个 item 项的参数。由于 **actionPanel** 会由视图 Viewport 的 LayoutManager 来页面定位，所以我们须在配置项对象中指定一个区域。

第三个步骤是创建 TabPanel(Tab 面板)本身。我们想在页面居中，即是对应于视图的中部。还要将一系列的 tab 配置项对象参数传入到面板中。在这里例子中,参与内置渲染的只有一个 tab，但是多个也是可以的。如能确定每个面板在页面上能够被当作容器使用，便可以成为该数组的元素。像当前的情况，我们是把 *tabs* 作为第一个面板的内容元素。 要注意，我们这指出了 tab 的 Id。这就是我们稍后获取的 tab 的依据。

最后，我们设置视图， 用于浏览器可视区域的控制。 所需要做的就是指定一个布局（layout）和什么要显示的组件。组件已经由视图的 LayoutMangager（视图管理器）配置好适合放置的区域。

这时，你应该在浏览器观察到，包含 Acitons 菜单的两个格式化列在左边，tab 面板占据了屏幕的其

余位置。

Step 3: 创建 Tab 控制逻辑

现在我们所需的元素已经创建好了， 可以增加 Tab 面板的创建（creating）和更新（updating）方法。在当前目录中新建三个页面以便我们的测试：

- *loripsum.html*
- *sample0.html*
- *sample1.html*

这三分文件的实际内容无关紧要， 但最好是每份的内容应该有所不同，好让 tab 加载内容后区别开来。

打开 **tab_actions.js**，在 **viewport** 创建的位置插入下列代码：

```
// 在中间的面板加入 tab
    function addTab(tabTitle, targetUrl){
        tabPanel.add({
            title: tabTitle,
            iconCls: 'tabs',
            autoLoad: {url: targetUrl, callback: this.initSearch, scope: this},
            closable:true
        }).show();
    }

    // 更新 tab 内容，如不存在就新建一个
    function updateTab(tabId,title, url) {
        var tab = tabPanel.getItem(tabId);
        if(tab){
                tab.getUpdater().update(url);
                tab.setTitle(title);
        }else{
                tab = addTab(title,url);
        }
        tabPanel.setActiveTab(tab);
    }

    // 映射连接的 id 到函数
    var count = 0;
    var actions = {
        'create' : function(){
```

```
                addTab("New Tab",'loripsum.html');
        },
        'use' : function(){
                // 演示页之间的轮换
                updateTab('tab1','替换' + count + ' 次','sample'+(count%2)+'.html');
                count++;
        }
};

function doAction(e, t){
        e.stopEvent();
        actions[t.id]();
}

// body 初始化后，viewport setup 过后才能执行下面的代码
actionPanel.body.on('mousedown', doAction, null, {delegate:'a'});
```

函数 **addTab(...)** 需要标题 title 和 URL 字符串两个参数，然后传递到 **tabPanel.add(...)**里的配置项对象。 这会返回创建好的面板对象，上面的代码立即调用了 **show()**方法显示内容。

函数 **updateTab(...)** 需要多个 tabId 的参数，以便能检测这个 tab 是否存在。如果是，面板会获取 Updater 进行面板内容的更新。若然不是，会调用 addTab(...)创建 tab。

最后一个步骤， 增加一个 **actionPanel** 的监听器，来响应选区动作之后的事件后，执行相应的函数。先要说明的是，**actions** 是一创建好的对象，对象可被认为是一种哈希表（HashTable）或是字典（Dictionary），一一映射了动作和方法两者。注意“键 key”对应于 HTML list 项。 由于方法比较简单，我们就直接这里写好了。变量 **counter** 用于清晰显示 tab 之间的切换，没有其他特定的功能。

事件处理器（event handler）**doAction(...)**执行时会有两个参数传入：事件对象和目标对象，**actions** 函数查找目标对象的 id 然后执行相应的方法。 当任意一个 **actionPanel's** 的组件被按下，将会触发 mousedown 的事件，侦听器立即通知已登记的事件处理器 **doAction(...)**运行。 被按下的那个组件便是事件对象的目标。