目 录

第一章：序

第一次看到 Extjs 的效果，我被深深震撼了。这真是一个了不起的框架，统一的效果、丰富的控件、强大的功能、能任意改变的皮肤、完美的浏览器兼容、原生态的 ajax、多种数据格式的支持、插件……我不知道该如何去表达我的兴奋和喜悦，就像年少时候看到一个心怡的姑娘，怦然心动。

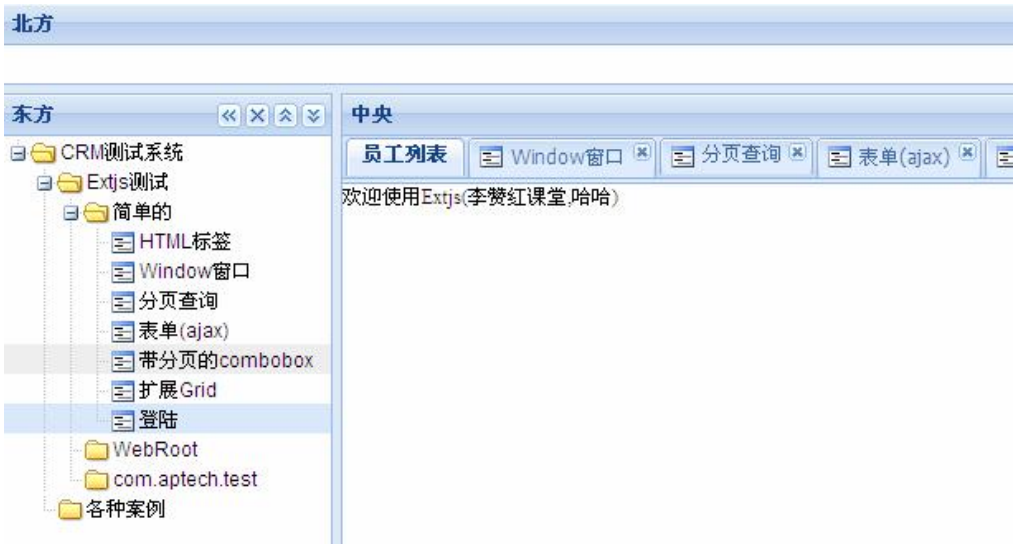
Extjs 是一个 javascript 框架，其开发团队的核心开发人员 Jack 将 OOP 应用到极致，我们不得不感叹，看起来不得要领的 javascript 语言居然也是如此优雅和无懈可击，在 Jack 的眼里，javascript 就是一位贵夫人，看上去高傲、目空一切、全无亲切感，但只要认真去研究去组织，就会得到贵夫人的欢心，但是，很抱歉，Jack 已捷足先登，并且生了一个孩子——Extjs。

在国外，许多大师值得我们敬佩，他们想象力丰富，思想前卫，长着一双鹰眼，具有无可比拟的洞察力。这些大师能觉察到将来 IT 的发展方向，走在各个领域的最前沿。我们——我、每一位老师，更包括在北大青鸟学习的莘莘学子——应该多去感受他们的精神，开拓、创新、勇往直前，培养战胜一切困难和险阻的勇气。

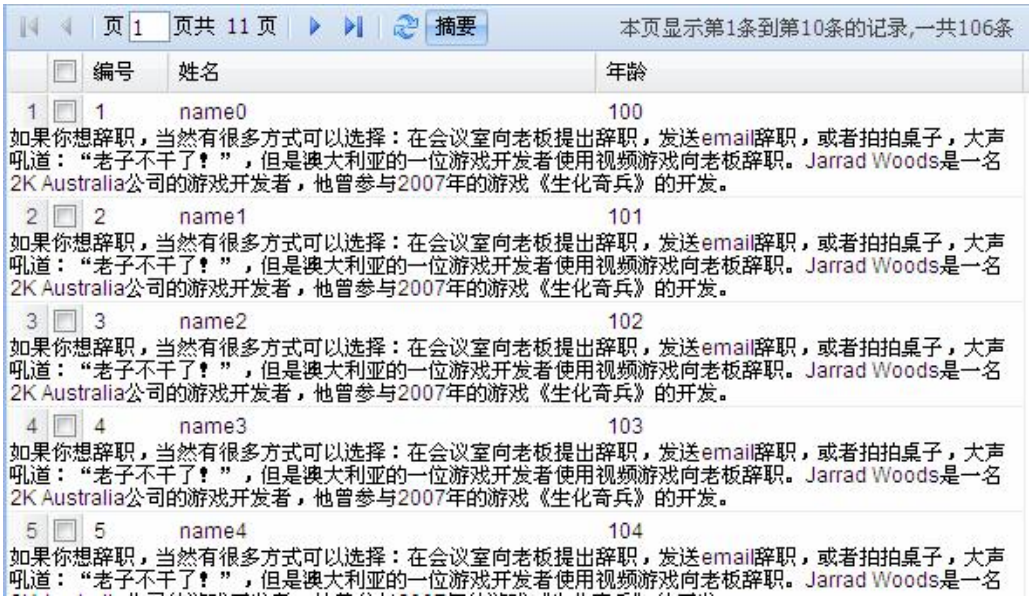
在很多很多大师们的努力下，一个个框架被开发出来并广泛应用，而我现在要和大家分享的，是 Jack 给我们带来的 Extjs。从现在开始，让我们熟悉并记住这个名字，然后，一起步入富客户端打造的美丽殿堂。

这是一个连载，因为我不可能在一夜之间将所有的东西都写出来，但我同样不想等写完之后再和大家分享，那样太迟了。我想，你的心情同我一样迫切，不过，让我们慢慢来，水滴石穿，每天进步一点点，一段时间后，回首走过的路，虽然充满艰辛，但更多的是成就，是一种成长的快意。

先看一张截图如何？



你做过这样的表格吗？



你见过能分页的下拉列表框吗？

一些看起来很复杂甚至让我们望而却步的功能，使用 Extjs 却可以轻易实现。事实上，远远不止这些，官方网站提供了非常多的示例，大部分可以直接拿到项目中使用，这些资源是一笔宝贵的财富，是提高生产力的法宝。

现在，Extjs 得到了越来越多的厂商的认同，在项目中被广泛应用。有些北大青鸟毕业的学生在公司主要负责 Extjs 技术，构建富客户端的工作。使用成熟的框架，能大大降低企业运营的风险，获取客户的高度认同，甚至，连美工都省了。

本文档作为株洲北大青鸟学生的 extjs 自学资料，在网上会同步更新，网址是：http://hi.baidu.com/ext_js。

让我们开始吧……

李赞红

2009-4-30 中国·株洲北大青鸟

第二章：准备与资源

一、下载

要开始 Extjs，必须先从网上下载所有需要的文件。目前的稳定版为 Extjs2.2，3.0 也快发布了。下载的文件解压后，包含若干个文件夹和文件，作用分别如下：

adapter 文件夹：该文件夹包含了 extjs 和其实框架如 jquery、yui、prototype 的桥接器，用于 Extjs 和这些框架的友好整合。

build 文件夹：包含了所有 js 文件的紧凑压缩版，方便网络传输，提高下载速度。

docs 文件夹：Extjs 的帮助文档，遗憾的是要先布署在服务器上才能访问。网上有人制作了无需布署的版本。

examples 文件夹：自带的示例全部在该文件夹中，一定记得去看看。

resources 文件夹：包含了 Extjs 所需要的样式表文件和图片资源。

source 文件夹：源代码文件夹，没有经过压缩的版本。

ext-base.js：基础类库。

ext-core.js：核心类库。

ext-all.js：类库完整版。

ext-all-debug.js：带格式的未压缩的类库完整版，带调试功能。

ext-core-debug.js：带格式的未压缩的核心类库。

CHANGES.html：开发及升级日志。

license.txt：协议，比较复杂，但用惯了盗版的我们来说完全不予理会。

二、拦路虎

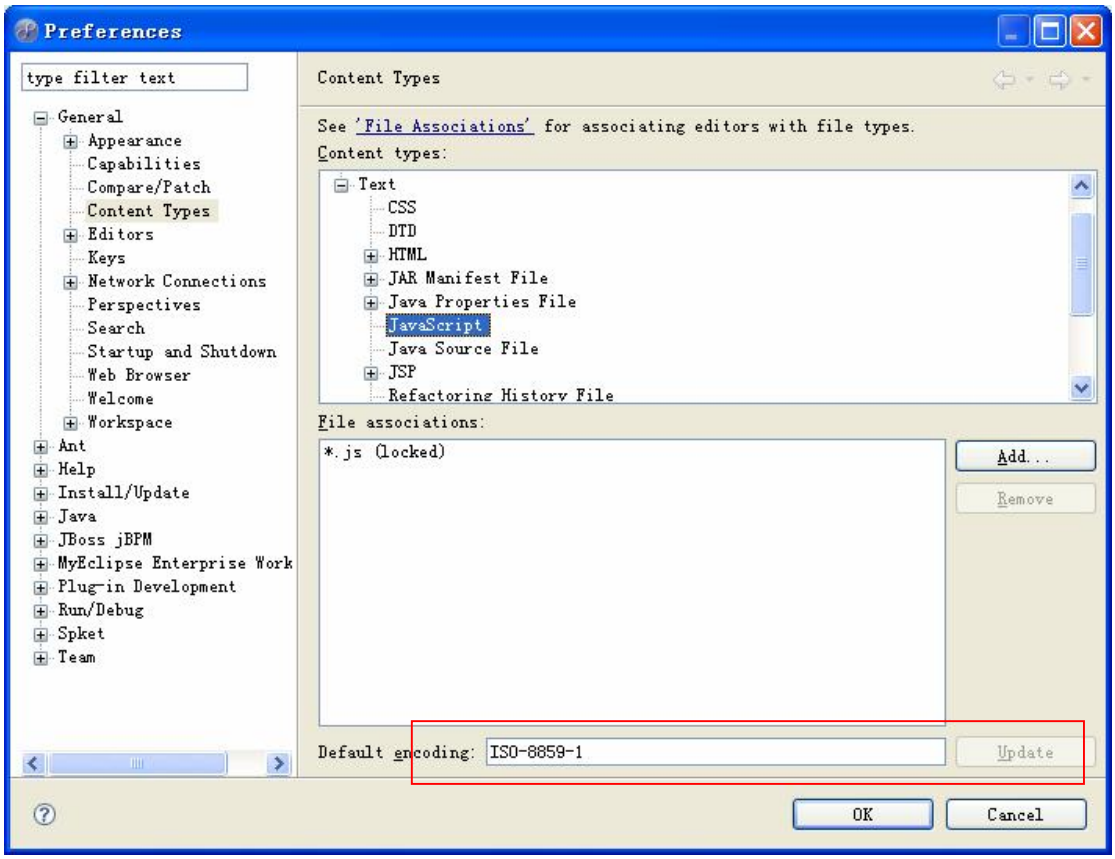
在学习之前，有一些问题需要澄清一下，不然，碰到问题的时候准会傻眼。我们还会在后面不断提醒，但事先让您知道也许是个不错的主意。

问题 1：导入 js 文件后，在 js 文件中定义的中文显示在页面后为什么会出现乱码？

问我算是问对人了，因为我为这事迷糊了一整天。乱码问题在 JavaEE 中司空见惯，但没有哪个的头疼度能和 Extjs 比。简直太伤人了。

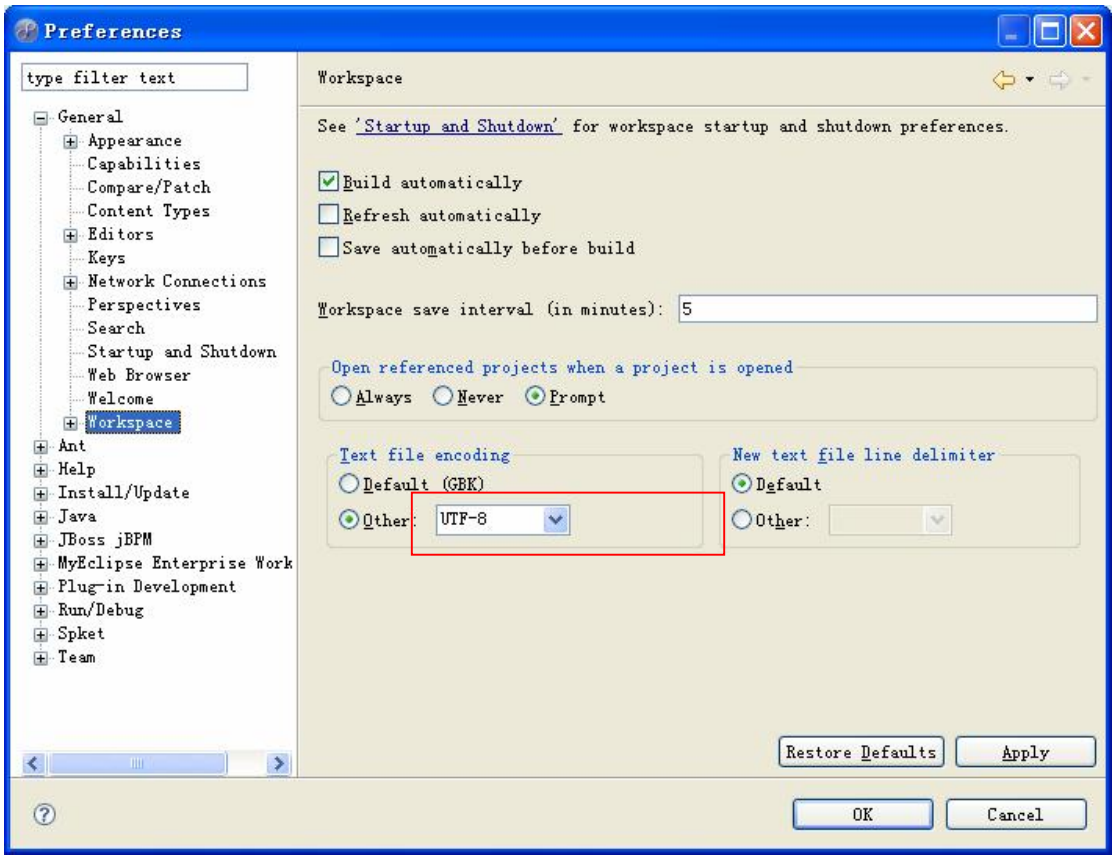
我们还得一二三逐步来，不然死了都不知道是怎么死的。

在 MyEclipse 中，js 文件的默认编码是 Iso-8859-1，这种编码和中文水火不容，从下面的界面中可以改成别的编码，如 gbk，其实我建议大家使用 utf-8，根据我的经验，utf-8 出现乱码的机率比较少。



在页面中导入 js 文件是，要指明字符集编码，形如：<script type="text/javascript" src="ViewPort.js" charset="utf-8"></script>

为了绝对的安全，将 eclipse 工作区的编码变成 utf-8，现在您也许不信，因为不改确实很好呐，可是，等到万一出现问题的时候，千万别怪我没告诉你。



问题 2：为什么同样的程序在 IE 和 Firefox 中显示不同？

Extjs 在很多情况下使用 11px 字体，11px 大小是一种边缘字体，不同的浏览器对 11px 的渲染各不相同，IE 的渲染和 12px 相似，而在 Firefox 中，则和 10px 相似，导致字体在 FF 中过小的问题。当然，英文看不出来，中文则相当别扭。像我这种追求完美的人是无法忍受的。网上有一些文章能解决一部分问题，但不能解决全部问题，一不做二不休，我们干脆打开 extjs/resources/css/ext-all.css 文件，把里面的 11px 全部替换成 12px，OK，成了。

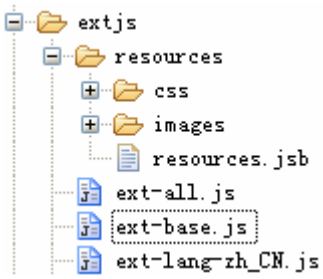
问题 3：为什么有些图片无法显示？

对不起，你的描述其实不对，不是有些，只有一张图片显示不出来，但这张图片的影响力太大了，可以说牵一发而动全身。为了配合皮肤，需要一张 1*1 大小的空白图片，变态的是，这张图片不是从本地获取的，而是访问了 http://extjs.com/s.gif，也就是说，访问了网络上的图片，如果没有连网，就会出现图片无法显示的情况。解决的方案是，在 js 文件中添加如下语句：Ext.BLANK_IMAGE_URL = "../extjs/resources/images/default/s.gif"; BLANK_IMAGE_URL 是 Extjs 中定义的一个常量，我们重新赋了新值。但是，由于环境不同，您的路径可能会有些许变化。

三、布署环境

把“布署环境”四个字拿出来，显然是吓人的，不就是脚本吗？不过，如果出错又解决不了，自信心会被严重打击，还是听我说吧。

请先将必要文件导入您的工程，以下是最精简的结构：



ext-lang-zh_CN.js 是为了本地化的需要，没别的用途。

众所周知，我们习惯将 javascript 写成独立的 js 文件，方便维护和共享，在页面中如果要访问 js 文件，需要使用 script 标签导进来，Extjs 有两个 js 文件是必须的，如下：

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

还是那句话，由于上下文环境不同，您的实际情况可能和上面的不一样，由 js 文件的实际路径决定。

同样，CSS 也同样要导入到工程中。以下是全部需要导入的文件：

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

要注意的是，不要随意调整各文件的顺序，特别是 ext-base.js 必须在 ext-all.js 文件之前，记住了！

四、SpketIDE

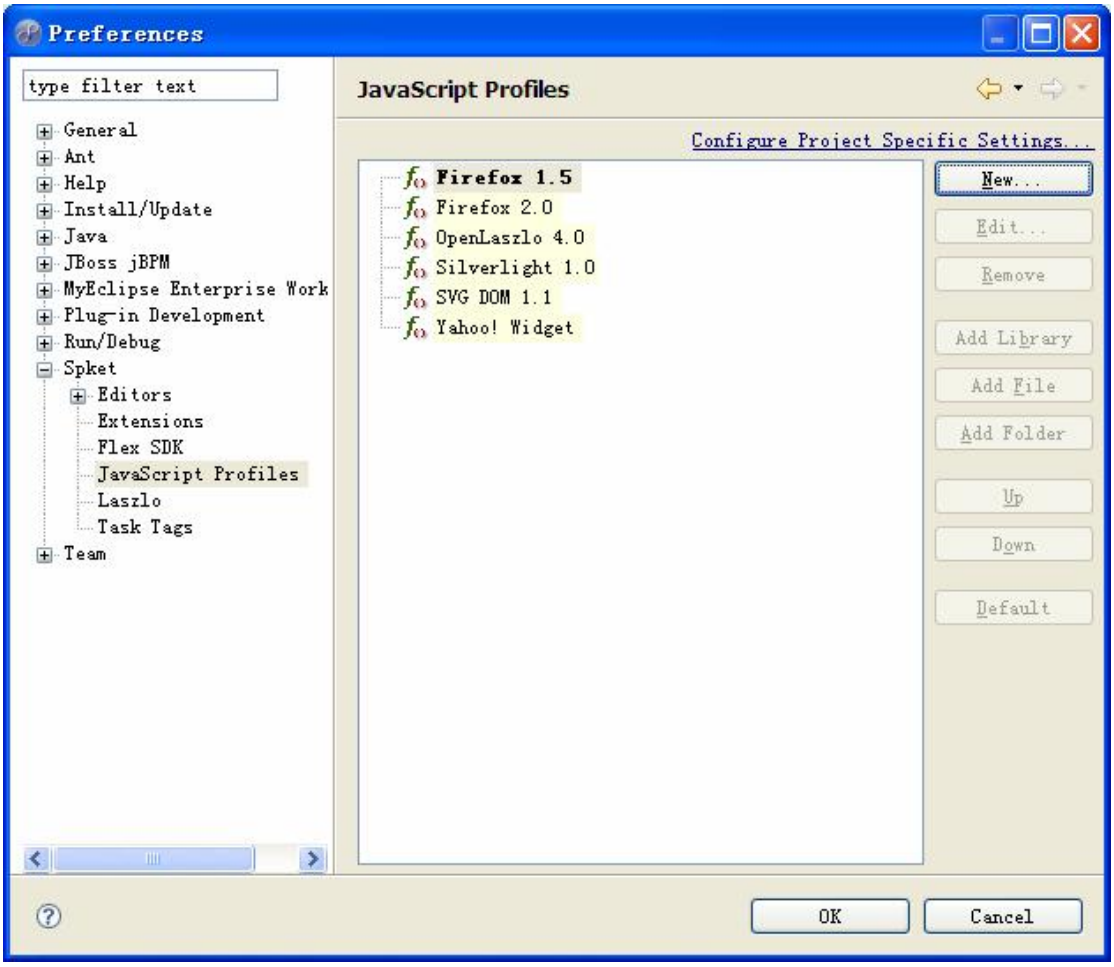
“工欲善其事,必先利其器”，再厉害的程序员，如果没有好的智能开发工具，效率也不可能提高。好的 IDE 能减少代码出错的机会，并对代码文件进行高效管理。事实上，因为 javascript 是弱弱弱类型语言，市面上没有像 eclipse 这样的集成开发环境帮助我们写 javascript 代码，但是，勉强能用的还是不少，SpketIDE 就是其一。

SpketIDE 在 Vs Stduio 和 Eclipse 平台上都有相应的插件。本人是搞 JavaEE 的，对 NET 方向无法作出交待，下面介绍 SpketIDE for Eclipse 的安装和使用。

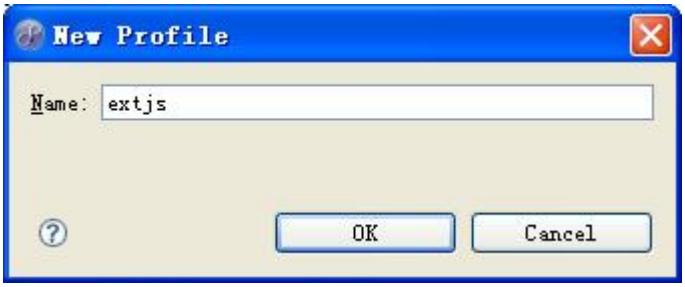
1、 首先手头上要有 spket-1.6.16.zip 文件，从网上可以下载，或者从学校 FTP 上获取。我现在使用的版本是 1.6.16；
2、 将 spket-1.6.16.zip 文件解压，复制文件夹所在目录，如：E:\LiZanhong\Devolpment\ext-2.0.2\spket-1.6.16，如果没有意外，下一级目录应该是 eclipse，千万不要复制 eclipse；
3、 切换到 eclipse 所在目录，并进入 links 目录（如果不存在，自己创建一个），如：D:\Program Files\MyEclipse 6.5\eclipse\links，在该目录中创建一个文本文件，文件名为：“ skpet.1.6.16.link ”，用记事本打开文件，输入内容： path= E:\\LiZanhong\\Devolpment\\ext-2.0.2\\spket-1.6.16;

4、 重新启动 Eclipse，如果安装成功，在 Window->Preferences 中会出现 Spket 的选项。

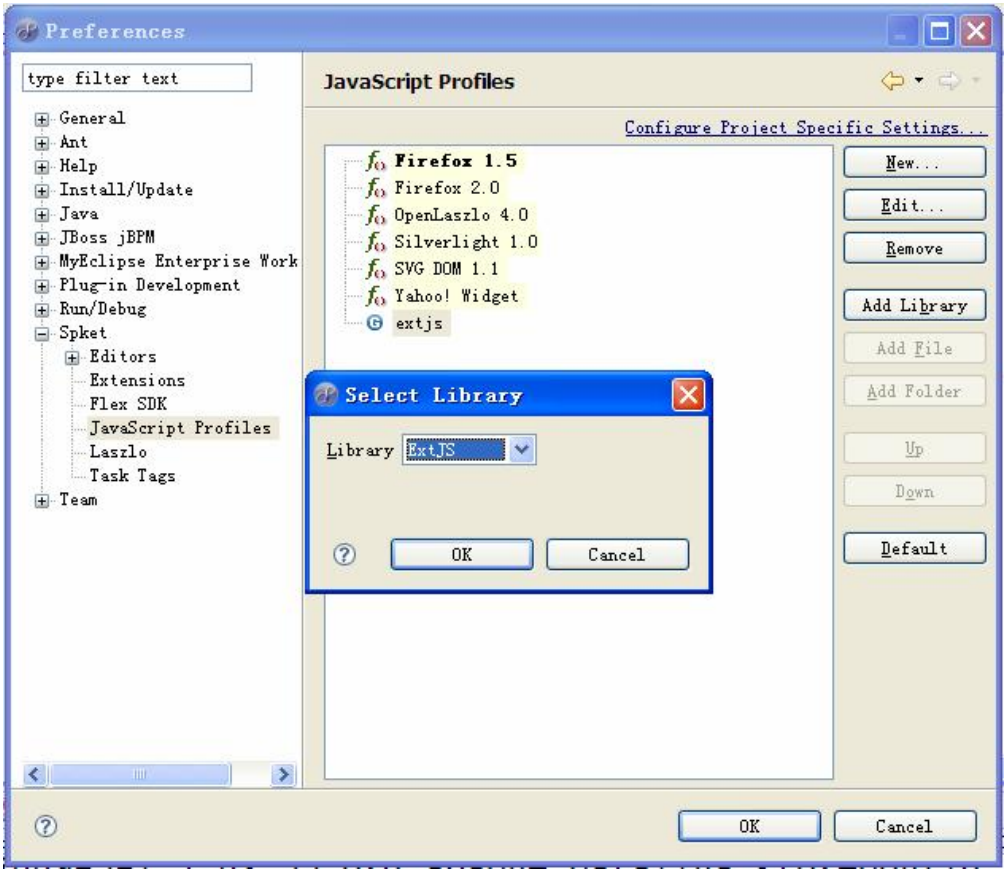
5、 下面的图能说明问题



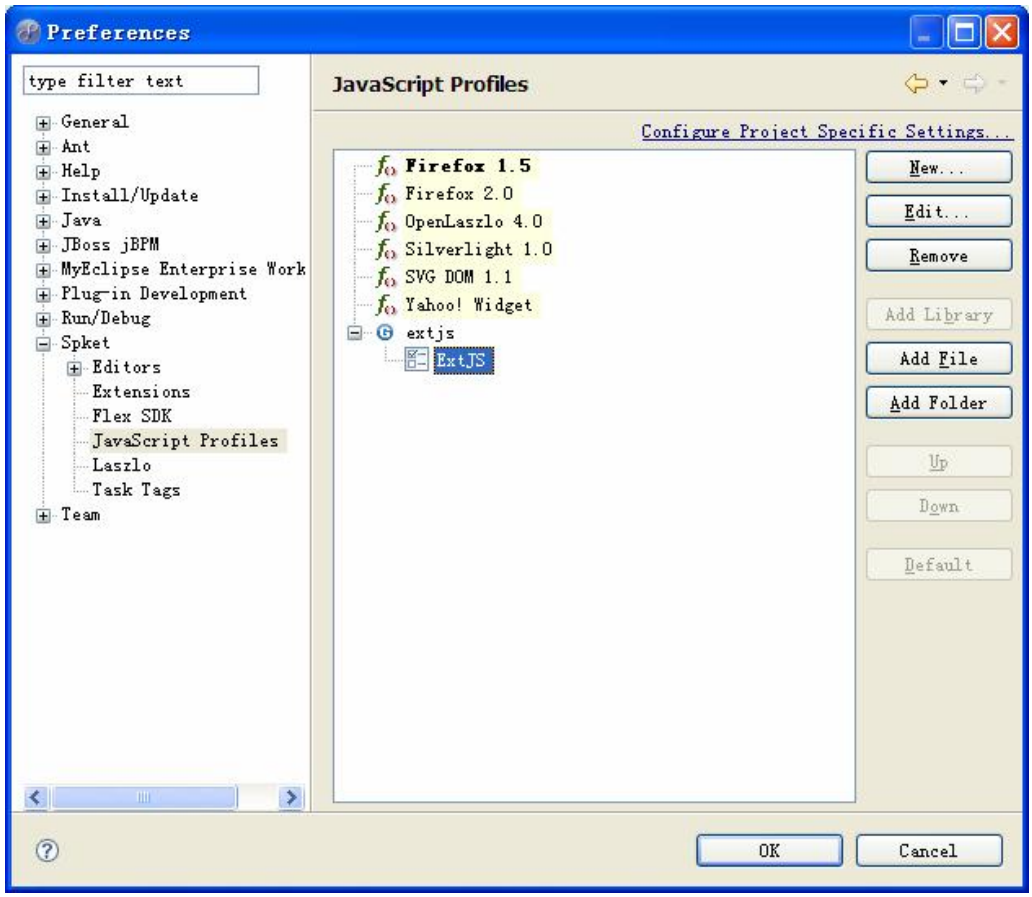
点击右上角的“new…”按钮，弹出下面的对象框：



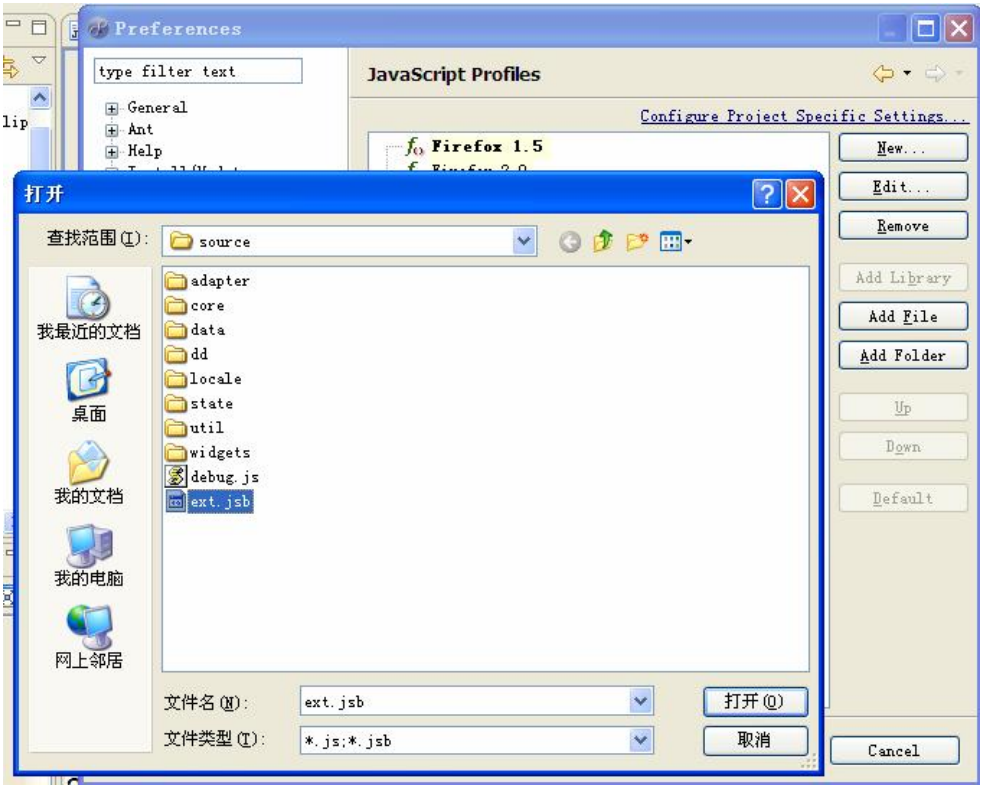
输入任意名称，如“extjs”，点击“OK”后，选中新加的“extjs”节点，单击“add library…”按钮，如下图所示：


Preferences
type filter text
General
Ant
Help
Install/Update
Java
JBoss jBPM
MyEclipse Enterprise Work
Plug-in Development
Run/Debug
Spket
Editors
Extensions
Flex SDK
JavaScript Profiles
Laszlo
Task Tags
Team
JavaScript Profiles
Configure Project Specific Settings...
Firefox 1.5
Firefox 2.0
OpenLaszlo 4.0
Silverlight 1.0
SVG DOM 1.1
Yahoo! Widget
extjs
New...
Edit...
Remove
Add Library
Add File
Add Folder
Up
Down
Default
Select Library
Library
ExtJS
OK
Cancel
OK
Cancel

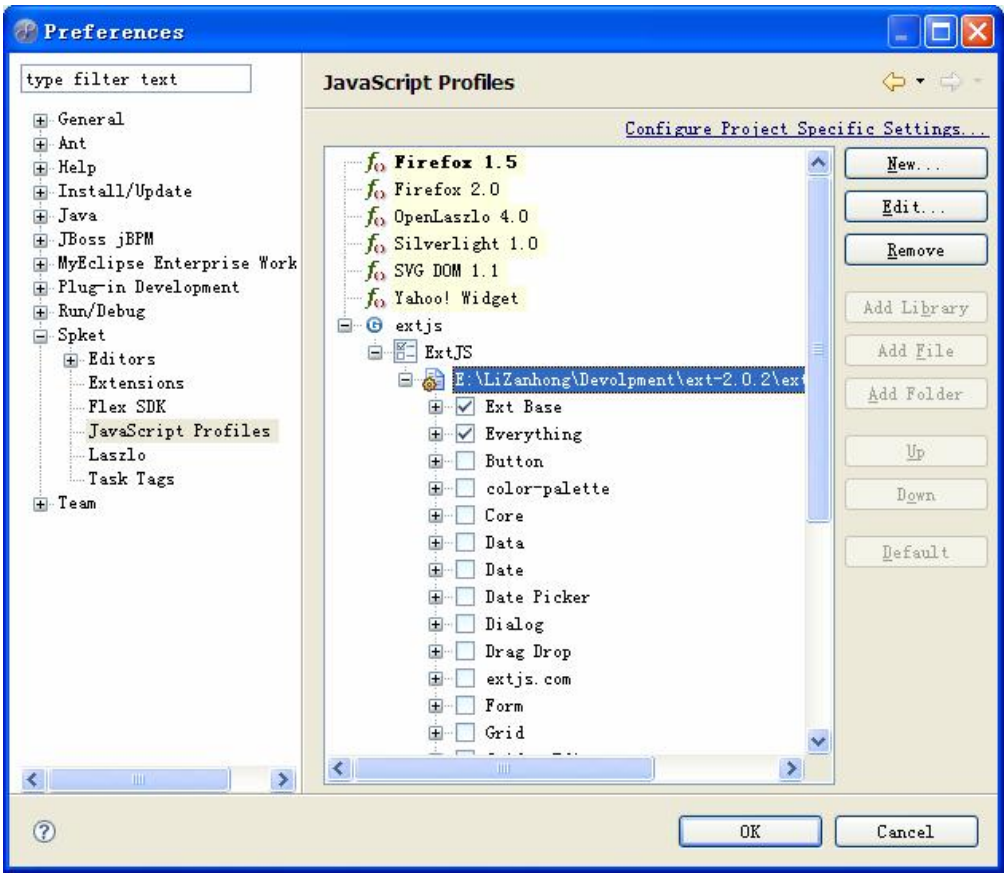
选择“ExtJS”，如下图：


Preferences
type filter text
General
Ant
Help
Install/Update
Java
JBoss jBPM
MyEclipse Enterprise Work
Plug-in Development
Run/Debug
Spket
Editors
Extensions
Flex SDK
JavaScript Profiles
Laszlo
Task Tags
Team
JavaScript Profiles
Configure Project Specific Settings...
Firefox 1.5
Firefox 2.0
OpenLaszlo 4.0
Silverlight 1.0
SVG DOM 1.1
Yahoo! Widget
extjs
ExtJS
New...
Edit...
Remove
Add Library
Add File
Add Folder
Up
Down
Default
OK
Cancel

选中“ExtJS”，单击“Add File”按钮，弹出下面的对话框：



想办法在 extjs 的文件夹中找到 ext.jsb 文件，打开后最后应该是下面这个样子：



至少，SpKetIDE 安装完成。

五、资源

最好的资源自然是官方提供的 AIP 文档和示例，这些东西很宝贵啊，虽然是洋文，但还是值得去研究。

另外，去上网吧，不是为了游戏，也不是为了聊天和娱乐，而是为了淘到更多的东西帮助我们学习——这才是正途。

这里提供一个：http://www.ajaxjs.com/ajaxjs/index/，去看看吧。

六、小结

万事开头难，把编程中一些常见的实际问题解决后，剩下的就是着手去学习。再加上一个好的开发工具，更加得心应手。

网上的资源非常丰富，免费的不要真是太浪费了。经常逛逛技术论坛，和经验老手交流学习，不愧是进步的一种方式。可以说，在网上没有找不到的东西，但学会提炼、筛选、吸收，更为重要。

第三章：Ext OOP 基础

一、javascript 类的定义

在 javascript 中，通过创建一个构造函数来定义一个类，然后通过 prototype 来扩展类的功能。假设我们定义一个螃蟹类：

```
Crab = function(){
    this.legs = 10;
}

Crab.prototype = {
    say: function(){
        alert("我是一只螃蟹，我有" + this.legs + "只脚，横行霸道是我的天性");
    }
};
//测试
var crab = new Crab();
alert(crab.legs);
crab.say();
```

prototype 是 javascript 一个非常重要的功能，能动态的为对象添加任何新的方法。Extjs 就是基于 prototype 实现的 OOP 机制。

二、Extjs 命名空间的定义

命名空间（namespace）类似于 java 中的包，用来对工程中的类进行有效管理。命名空间的层次结构使用“.”来划分。Ext 通过 namespace()方法创建命名空间。

语法：Ext.namespace(“命名空间”)

示例：Ext.namespace("com.aptech");

三、Extjs OOP

在 Extjs 中创建类和 javascript 有些不同，我们会使用他封装好的东西，而不全是基于 javascript 语法。站在巨人的肩膀上，确实有些高处不胜寒。所以，深刻了解 javascript 基础对于日后的拓深十分必要，相信我吧。

我们通常会基于命名空间创建新类，按照 java 的设计思想，会有封装、继承和多态。Extjs 也不例外，而且 Extjs 为 OOP 做了很多基础工作，使用起来非常模式化。一个类至少应该有 private 和 public 成员，且可以派生出子类，并能重写父类的方法。那么，让我们来看看 Extjs 是如何做到的。

```
Ext.namespace("com.aptech");

com.aptech.First = function(){
    //私有成员
    var kiss = "中华人民共和国";
    //私有方法

    //公有方法
    return {
        //公有成员
        init: function(){
            alert("init");
            alert(kiss);
        },
        //公有成员
        method: function(){
            alert("method");
        }
    };
};
```

我们定义了一个类 First，这实在是一个没有任何业务意义的类，只是为了说明方便。First 位于 com.aptech 命名空间中，有一个私有成员 kiss，并且向外部暴露了两个方法 init()和 method()。其实，在 function 和 return 之间定义的成员全总是 private，而在 ruturn 内部定义的成员全部是 public，如果大家的 javascipt 基础扎实的话，这段代码并不难理解，我们定义了一个匿名构造函数，函数中的变量是局部变量，外部无法访问，返回一个对象，对象是以 json 格式定义的，该对象中定义的方法自然可以访问了。

javascipt 本身不支持继承，但是我们可以模拟。继承说穿了就是子类将父类的成员据为已有，专业点就是“成员复制”，即可以复制成员变量，也可以复制成员方法。我们定义下面的方法完成此功能：

```
var extend = function(child, father){
    child.prototype = father.prototype;
}
```

现在，我们定义一个螃蟹的子类——蟹将，螃蟹成精变成了人，由原来的 10 只脚变成 2 只脚，但狗改不了吃屎，行为不会改变，依旧横行霸道。

```
GenCrab = function(){this.legs = 2;};
extend(GenCrab, Crab);
var gc = new GenCrab();
gc.say();
```

就这样，一个新类产生了。不过，在 Extjs 中有更加优雅的做法。

我们定义一个类 Second，继承自 First，看看 Extjs 是如何做的。

```
//创建子类
```

```
com.aptech.Second = function(){
    com.aptech.Second.superclass.constructor.apply(this);//调用父类构造方法
}
//com.aptech.Second 子类继承自父类 com.aptech.First
Ext.extend(com.aptech.Second, com.aptech.First, {
    //新方法
    fun: function(i){
        return i * i * i;
    },
    //重写的方法
    method: function(){
        alert("Second::method")
    }
});

//测试
var second = new com.aptech.Second();
alert(second.fun(5));
second.method();
```

哈哈，简直太优雅了，不仅可以添加新方法，还可以重写父类的方法（话外音：这不是多态的表现形式吗？）。这一切都是由 Ext.extend()搞定的，这个方法有点复杂，但他的实现原理是相同的。

四、配置(config)选项

在 Extjs 中，初始化对象时，大量使用了 config 这个参数。不要恐惧，只是一个 json 对象而已，不过，config 为 Extjs 立下了不少汗马功劳。

假设定义了一个学生类（Student），有姓名和性别两个属性，并且通过构造函数为属性初始化：

```
Student = function(name, sex){
    this.name = name;
    this.sex = sex;
}

//测试
var student = new Student("李赞红", "男");
alert("姓名：" + student.name + "\r\n 性别：" + student.sex);
```

这个一定看得懂，如果看不懂，我只能表示深深的遗憾了，您不适合地球，回你的老本营火星去吧。

如果用 json 对象作为构造函数的参数呢？

```
Student = function(config){
```

```
    this.name = config.name;
    this.sex = config.sex;
}

//测试
var student = new Student({name: "李赞红", sex: "男"});
alert("姓名：" + student.name + "\r\n 性别：" + student.sex);
```

嘿，果然万变不离其宗啊。换汤不换药的把戏骗不了咱们。但是，等等，请等等，如果类的成员特别多，十个，二十个，一百个，赋值语句岂不是很多很繁琐？你能想到这一点实在太聪明了，不过，Jack 更聪明，他早想到了，于是有了下面的代码：

```
Student = function(config){
    Ext.apply(this, config);
}

//测试
var student = new Student({name: "李赞红", sex: "男"});
alert("姓名：" + student.name + "\r\n 性别：" + student.sex);
```

Ext 定义了一个名叫 apply()的方法，作用是将第二个参数的成员赋给第一个参数。现在，不管 config 中有多少个成员都没问题了。

五、Ext.apply()和 Ext.applyIf()

前面我们知道了 Ext.apply(obj, config)方法的作用，还有另一个方法 applyIf(obj, config)，从名字上看得出来，applyIf()需要满足某种条件，实在是太棒了，这么复杂的问题都没逃过你的法眼。

事先预告一下这两个方法的区别，然后再通过例子来说明：apply 会将 config 和 obj 参数的同名属性值覆盖，并且将 config 其他属性添加到 obj 中；applyIf 不会将 config 和 obj 参数的同名属性覆盖，只将 config 其他属性添加到 obj 中。也就是说，obj 没有而 config 中有的属性最终都会复制到 obj 中，不同的是相同属性值是否会被覆盖的问题。

例子能说明一切问题。

```
Student = function(config){
    this.name = "张海军";
    this.sex = "男";
    Ext.apply(this, config);
}

//测试
var student = new Student({name: "李赞红", sex: "男", birthday: new Date()});
alert("姓名：" + student.name + "\r\n 性别：" + student.sex + "\r\n 生日：" +
student.birthday);
```

从下面结果看出，属性 name 和 sex 均被覆盖，且添加了新成员 birthday。

```
姓名：李赞红
```

```
性别：男
生日：Fri May 01 2009 07:59:39 GMT+0800
```

```
Student = function(config){
    this.name = "张海军";
    this.sex = "男";
    Ext.applyIf(this, config);
}

//测试
var student = new Student({name: "李赞红", sex: "男", birthday: new Date()});
alert("姓名：" + student.name + "\r\n 性别：" + student.sex + "\r\n 生日：" + student.birthday);
```

结果如下：

```
姓名：张海军
性别：男
生日：Fri May 01 2009 08:02:33 GMT+0800
```

哈，“张海军”终于没被“李赞红”替换了。

六、小结

讲了这么多，尽是些和 Ext 不沾边的事，是不是有点失望？

但是，我用人格担保我是个不啰嗦的人，了解我的学生肯定知道。这一章的内容是 Extjs 的基础，是我们能看懂源代码的保证。不然，看到 Jack 的代码，你以为自己进入了迷宫，或者 Jack 故意要把我们打入冷宫。

虽然是基础，却是相对的，因为就是这寥寥几行代码，蕴含了丰富的设计哲学，细细体会，才知其味。

第四章：消息框

一、话说消息框

大家对消息框并不陌生哈，消息框通常是一个模式窗口，会屏蔽掉当前窗口所有的鼠标键盘事件，关闭后才能继续后面的操作。消息框的类型也不一而足，有显示信息的，有提示输入的，有确定用户行为的，IE 和 Firefox 默认的消息框很难看啊，死气沉沉，古板枯燥，看多了令人生厌。

不过，好消息是 Ext 为我们带来了全新风格的消息框，那个舒服啊，啧啧啧……

Extjs 在实现消息框的时候，完全摈弃了传统的风格，不再弹出新的对话框，而是在当前页面跳出一个层，并将原页面完整覆盖。

原来，只是一种模拟。

在 Ext 中，定义了一个类 MessageBox，该类还有一个更精简的名字 Msg，所有消息框都定义在该类中。

二、最简单的消息框——提示框

最简单的？Extjs 有最简单的东西吗？没有。如果深入了解，发现每个细节都不简单，但是，幸好大部分时候我们只是用用，解决项目中的实际问题，到底 Jack 如何做到的，让喜欢玩的人玩去吧。

提示框的语法：

Ext.MessageBox. alert (String title, String msg, Function fn, Object scope);

参数定义如下：

1、 title：标题

2、 msg：提示内容

3、 fn：提示框关闭后自动调用的回调函数

4、 scope：作用域，用于指定 this 指向哪里，一般不用管他，特殊情况下有用

其实，通常情况下，我们用得更多的是 title 和 msg 两个参数，举例如下①：

```
extjsAlert = function(){
    Ext.MessageBox.alert("提示框", "这是一个提示框");
}
```

也可以这样：

```
extjsAlert = function(){
    Ext.MessageBox.alert("提示框", "这是一个提示框", function(){
        alert("提示框关闭了");
    });
}
```

页面代码是这样的：<input type="button" value="alert" onclick="extjsAlert();">，记住函数名不要使用和 DOM 模型相同的名字，他们犯冲。

三、输入框

输入框用来提示输入字符串，相当于 window.prompt()方法。

语法：

Ext.MessageBox.prompt(String title,String msg, Function fn, Object scope, Boolean/Number multiline)

从定义中可以看到，前四个参数和提示框一样，最后多了一个参数，如果为 true 或为数字，将允许输入多行或者指定默认高度（像素）。

示例如下②：

```
extjsPrompt = function(){
    Ext.MessageBox.prompt("输入框", "请输入您的姓名：", function(btn, txt){
        Ext.MessageBox.alert("结果", "您点击了" + btn + "按钮，<br>输入的内容为" +
txt);
    });
}
```

如果显示多行，看下面示例③：

```
extjsPrompt = function(){
    Ext.MessageBox.prompt("输入框", "请输入您的姓名：", function(btn, txt){
        Ext.MessageBox.alert("结果", "您点击了" + btn + "按钮，<br>输入的内容为" +
txt);
    }, this, 300);
}
```

注意回调函数的参数，第一个为点击的按钮的名字，如果点击确定，为“ok”，如果点击取消，为“cancel”，第二个参数就是用户输入的文本。

四、确认框

确认框提示用户作出选择，语法如下：

Ext.MessageBox.confirm (String title, String msg, Function fn, Object scope)

参数同上，示例如下：

```
extjsComfirm = function(){
    Ext.MessageBox.confirm("确认", "请点击下面的按钮作出选择", function(btn){
        Ext.MessageBox.alert("您单击的按钮是：" + btn);
    });
}
```

我们可以通过回调函数的参数 btn 采取相应的行动。

五、自定义消息框

如果上面所有的消息框都无法满足我们的需求，譬如没有图标、按钮类型太少，甚至更 BT 的，如果想要个进度条怎么办？哈哈，别急，Extjs 息数为你想到了。我们可以使用 show() 方法自定义消息框，只需要稍微配置一下就可以了。

show()方法的语法如下：

Ext.MessageBox.show (Object config)

语法是不是显得更简单？不要小瞧了他，config 这个参数可谓包罗万象，使用 json 格式可以传输很多信息到方法中去。

config 中常见属性如下：

- title：消息框标题栏
- msg：消息内容
- width：消息框的宽度
- multiline：是否显示多行文本
- closable：是否显示关闭按钮
- buttons：按钮
- icon：图标
- fn：回调函数

举例说明：

```
extjsCustom = function(){
    var config = {
        title: "自定义对话框",
        msg: "这是一个自定义对话框，想怎么搞就怎么搞",
        width: 400,
        multiline: true,
        closable: false,
        buttons: Ext.MessageBox.YESNOCANCEL,
        icon: Ext.MessageBox.QUESTION,
        fn: function(btn, txt){
            Ext.MessageBox.alert("结果", "您点击了‘yes’按钮<br>，输入的值是：" + txt);
        }
    };
    Ext.MessageBox.show(config);
}
```

在上面的例子中，不熟悉的有 buttons 和 icon，这些选项在 Ext.MessageBox 中已有定义。

buttons（按钮）的取值如下：

OK：只有“确定”按钮

CANCEL：只有“取消”按钮

OKCANCEL：有“确定”和“取消”按钮

YESNO：有“是”和“否”按钮

YESNOCANCEL：有“是”、“否”和“取消”按钮

icons（图标）取值如下：

INFO：信息图标

WARNING：警告图标

QUESTION：询问图标

ERROR：错误图标

六、进度条对话框

进度条对话框可以说是一个创举，让一个富了现实意义和使用价值的进度条轻松实现，随着进度条的滚动，我们的心情也随之畅快起来。代码赋予我们无穷的活力，让人们的视觉再次受到最强烈的冲击。

进度条对话框也是一个自定义消息框，配置 config 时添时 progress=true 即可，同时还可以设置其他相关信息，如进度提示等。Extjs 为我们提供的只是一个对话框而已，进度条的滚动还得通过代码实现。下面是进度条的代码示范：

```
extjsProgress = function(){
    Ext.MessageBox.show({
            title: '请等待',
            msg: '正在加载项目...',
            progressText: '正在初始化...',
```

```
            width:300,
            progress:true, //此属性证明这是一个进度条
            closable:false
        });

        var f = function(v){
             return function(){
                if(v == 12){
                    Ext.MessageBox.hide();
                    Ext.MessageBox.alert('完成', '所有项目加载完成!');
                }else{
                    var i = v/11;
                    Ext.MessageBox.updateProgress(i,  Math.round(100*i)+'%  已完
成');
                }
            };
        };

        for(var i = 1; i < 13; i++){
            setTimeout(f(i), i*500);
        }
}
```

在上面的代码中，progressText 属性是进度条滚动之前最初的文本，滚动进程由 updateProgress(Number value, String progressText)方法来定义，参数 value 是从 0~1 之间的小数，表示进度百分比；progressText 则表示进度条滚动过程中的文本提示信息，如 Ext.MessageBox.updateProgress(i, Math.round(100*i)+'% 已完成');。

七、让消息框飞出来

很显然，这是一个动画效果。Extjs 允许我们将消息框从指定的位置飞出来，关闭时又飞回去，很炫吧！实现也相当轻松，只要设置 animEl 选项即可，该选项指定一个标签，即消息框从标签处飞出，关闭后又飞回标签。下面是演示代码：

```
extjsAnimal = function(){
    var config = {
        title: "飞出的消息框",
        msg: "这是一个自定义对话框，是飞出来的哦。",
        width: 400,
        multiline: true,
        closable: false,
        buttons: Ext.MessageBox.YESNOCANCEL,
        icon: Ext.MessageBox.QUESTION,
        animEl: "fly"
    };
```

```
    Ext.MessageBox.show(config);
}
```

animEl 的值为“fly”，这是按钮的 id 值，在 html 页面中这样定义：<input type="button" value="Animal" id="fly" onclick="extjsAnimal();">
。

八、小结

本节是 Extjs 最简单的内容，和浏览器传统的效果相比，效果更加炫丽动感。一个小小的消息框，Extjs 如此注重细节，实在难能可贵。我们有理由相信，富客户端的蓝天，即将徐徐展开。

以下是本章使用的 html 页面内容：

```
<input type="button" value="Alert" onclick="extjsAlert();"><br>
<input type="button" value="Prompt" onclick="extjsPrompt();"><br>
<input type="button" value="Comfirm" onclick="extjsComfirm();"><br>
<input type="button" value="Custom" onclick="extjsCustom();"><br>
<input type="button" value="Progress" onclick="extjsProgress();"><br>
<input type="button" value="Animal" id="fly" onclick="extjsAnimal();"><br>
```

第五章：页面与脚本完全分离

一、Extjs 是脚本的世界

编写良好的页面，往往将页面与脚本分离，而不是混杂在一起，这是一个好习惯，有利于代码维护、重构和重用。Extjs 将 javascript 的作用发挥到极致，在界面设计过程中，我们发现不再需要和 HTML 标记打交道，不再关注 div、table 和 span。如果您有 VB 或 Winform 的经验，更像这些可视化开发技术，只要以某种布局放置控件即可。所有的页面最终由 javascript 生成。

还是老调重弹，这需要一定的 javascript 基础。javascript 的确是一门神奇的语言，似乎无法驾驭。但是，只要把握了 javascript 的灵魂，实现起来根本不是难事。

二、Ext.onReady 事件

上一章中，我们曾看到这样一句话：<input type="button" value="Alert" onclick="extjsAlert();">，这样的代码好吗？好，但是不够好，因为 HTML 标记和函数还存在一定依赖关系。我们有办法将二者彻底分离，这一切，得从 Ext.onReady 事件开始。

页面加载完成后，Ext.onReady 事件被触发，基本上，除了类封装，与页面相关的操作都会写在该事件中。示范代码如下：

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

该事件触发后，匿名函数被调用，我们的代码恰恰就是写在这个匿名函数中，这样，我们便可以通过脚本为 HTML 标签绑定事件了。下面的示例您一定要看懂：

```
在 HTML 页面中，有下面的按钮，注意，该按钮没有定义任何的事件：
<input type="button" value="点击我" id="btn">
现在，我们在 js 文件中定义如下代码：
Ext.onReady(function(){
    Ext.get("btn").on("click", function(){
        Ext.MessageBox.alert("点击", "我被点击了，非常高兴");
    });
});
```

运行该页面，点击“点击我”的按钮，您发现了什么？居然弹出了一个消息框。下面是几点说明：

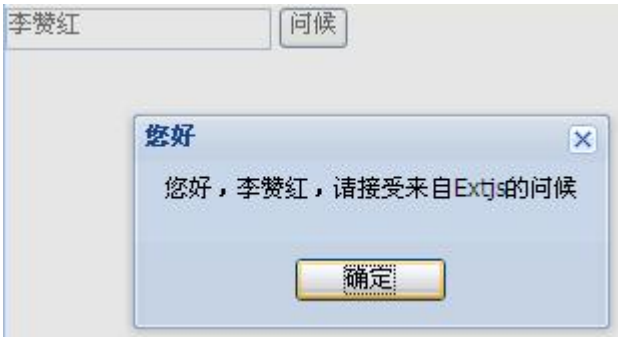
1、 Ext.get (Mixed el)：根据 HTML 标签的 id 属性获取 Ext.Element 对象，Ext.Element 是对 DOM 的封装，这个是比较常见的用法，另外参数也可以是 Ext.Element 对象或者 DOM 节点对象。

2、 on(String eventName, Function fn)：为 Ext.Element 对象定义一个事件，eventName 是事件名称，和传统的事件名称相比不以“on”开头，fn 为事件处理函数。从这里可以延伸开去，也可以是“change”、“keypress”等。

3、 上面例子的意思是为 id=“btn”的按钮定义一个 click 事件，事件触发后弹出一个消息对话框。

三、来自 Extjs 的问候

本节向您展示一个经典的问候示例，页面上有一个文本框，用于输入姓名，另外放置一个按钮，单击按钮显示问候的消息框，如下图：



哈，确实是一个众所周知的例子，代码也同样简单：

5-2-2.html

```
<body>
    <input type="text" id="name">
    <input type="button" id="btn" value="问候">
</body>
```

5_2_2.js

```
Ext.onReady(function(){
    Ext.get("btn").on("click", function(){
        var name = Ext.get("name").dom.value;
        Ext.Msg.alert("您好", "您好，" + name + "，请接受来自 Extjs 的问候");
    });
})
```

Ext.Element.dom：dom 属性是 Ext.Element 对象的 Dom 表示，获取 dom 之后，可以用传统的方式进行访问。

四、让界面动起来

界面具有动画效果最后归结为元素的动画效果。Extjs 定义了若干个方法用于完成元素动画的实现，一些是内置的，一些需要程序员自己编程。简单说，动态修改元素的样式是实现动态效果最直接最常见的方法，所以，这其中并无奥妙可言，看清楚了事实的真相，只要具备足够丰富的想象力，任何人都可以做出叹为观止的动画。

虽然动画效果应用于 Ext.Element 对象，却并不定义在 Ext.Element 类中，而是定义在另一个类 Ext.Fx 中。这让我非常疑惑，仔细查看了 Ext.Element 的源代码，愣是没发现 Ext.Fx 的身影，最终，在 Ext.Fx 中看到了玄机，该类的最后一个语句告诉了我们事情的原委：Ext.apply(Ext.Element.prototype, Ext.Fx);，看看他的解释：Ext.Fx is automatically applied to Element so that all basic effects are available directly via the Element API（Ext.Fx 自动应用于

Element 对象，所有的基本效果直接通过 Element 调用）。

五、Ext.Fx 类

正如前面所说，Extjs 的动画大部分定义在 Ext.Fx 中，尽管如此，Ext.Element 类也定义了部分动画函数。我们先来看看 Ext.Fx 类中的重要方法。

1、slideIn ([String anchor], [Object options])：

功能：滑入效果，作动画显示。

参数：

anchor：推出的方向，定义了 8 种不同的方向，值不区别大小写，可选。

值	*说明*
tl	左上角
t	顶部中央
tr	右上角
l	左边中央
r	右边中央
bl	左下角
b	底部中央
br	右下角

options：选项配置，比较典型的是 duration 属性，用于定义动画持续的时间，可选。以下是默认配置：

```
slideIn('t', {
    easing: 'easeOut',
    duration: .5
});
```

示例：在 10 秒钟之内将 div 从右边中央滑入

5_5_1.html

```
<div id="a1"> slideIn </div>
```

5_5_1.js

```
Ext.onReady(function(){
    Ext.get("a1").applyStyles({position: "absolute", top: 200, left: 200, backgroundColor: "red", width: 100, height: 100}).slideIn("r", {duration: 10});
})
```

applyStyles 是 Ext.Element 的方法，用于定义指定元素的样式。

2、slideOut ([String anchor], [Object options])：滑出效果，作动画隐藏。参数及用法同上。以下是该方法的默认配置：

```
slideOut('t', {
    easing: 'easeOut',
    duration: .5,
```

```
    remove: false,
    useDisplay: false
}
);
```

3、highlight ([String color], [Object options])：根据设置的颜色高亮显示 Element 对象，然后渐隐为原始颜色。默认情况，高亮显示的背景颜色为黄色。

参数：

color：起始颜色

options：选项配置

一个能应用在项目中的典型例子是：

```
Ext.get("a2").applyStyles({
    position: "absolute",
    top: 200,
    left: 300,
    backgroundColor: "red",
    width: 100,
    height: 100}).highlight("0000ff"/*起始颜色*/,
        {
            attr: 'background-color', /*我们改变的是背景颜色*/
            duration: 2,/*动画持续时间*/
            endColor: "ff0000" /*结束颜色*/
        }
    );
```

如果可以把 attr 属性值改成 color，我们发现颜色改变的不再是背景，而是文字颜色。endColor 的颜色值不能是形如 red 之类的英文单词，只能是 16 进制表示。起始颜色为蓝色，终止颜色为红色，中间的渐变由 Extjs 自动完成，非常自然。

4、frame ([String color], [Number count], [Object options])：展示一个展开的波纹，伴随着渐隐的边框以突出显示 Element 对象。默认情况下展示的是一个淡蓝色的波纹。

参数：

color：波纹颜色

count：波纹的个数

options：选项配置

示例：三个红色的波纹并持续 3 秒。

```
Ext.get("a3").applyStyles({
    position: "absolute",
    top: 200,
    left: 400,
    backgroundColor: "red",
    width: 100,
    height: 100}).frame("ff0000", 3, { duration: 3 });
```

5、fadeIn ([Object options])：将元素从透明渐变为不透明。结束时的透明度可以根据

"endOpacity"选项来指定。

示例：

```
Ext.get("a4").applyStyles({
    position: "absolute",
    top: 200,
    left: 500,
    backgroundColor: "red",
    width: 100,
    height: 100}).fadeIn({
        endOpacity: 1, //可以是 0 到 1 之前的任意值（例如：.5）
        duration: 4
    });
```

6、fadeOut ([Object options])：将元素从不透明渐变为透明。结束时的透明度可以根据"endOpacity"选项来指定。

示例：

```
Ext.get("a5").applyStyles({
    position: "absolute",
    top: 200,
    left: 600,
    backgroundColor: "red",
    width: 100,
    height: 100}).fadeOut({
        endOpacity: 0, //可以是 0 到 1 之前的任意值（例如：.5）
        duration: 4,
        remove: false,
        useDisplay: false
    });
```

7、scale (Number width, Number height, [Object options])：以动画展示元素从开始的高度/宽度转换到结束的高度/宽度。

参数：

width：结束宽度，如果为 undefined 则保持原始宽度

height：结束高度，如果为 undefined 为保持原始高度

示例：

```
Ext.get("a6").applyStyles({
    position: "absolute",
    top: 200,
    left: 700,
    backgroundColor: "red",
    width: 10,
    height: 10}).scale(100, 100, {duration: 2});
```

8、shift (Object options)：以动画展示元素任意组合属性的改变，如元素的尺寸、位置

坐标和（或）透明度。 如果以上属性中的任意一个没有在配置选项对象中指定则该属性不会发生改变。 为了使该特效生效，则必须在配置选项对象中设置至少一个新的尺寸、位置坐标或透明度属性。

下面是一个比较有实用性的示例：

```
Ext.get("a7").applyStyles({
    position: "absolute",
    top: 200,
    left: 800,
    backgroundColor: "red",
    width: 10,
    height: 10}).shift({
        width: 100,/*动画终止之后的宽度*/
        height: 100,/*动画终止之后的高度*/
        x: 0,/*动画终止之后的 x 坐标*/
        y: 0,/*动画终止之后的 y 坐标*/
        opacity: .5,/*动画终止之后的透明度，0-1 之间的任意值*/
        duration: 5/*动画持续的时间(5 秒)*/
    });
```

9、ghost ([String anchor], [Object options])：将元素从视图滑出并伴随着渐隐。作为可选参数传入的定位锚点将被设置为滑出特效的结束点。

参数：

anchor：同 slideIn

示例：

```
Ext.get("a8").applyStyles({
    position: "absolute",
    top: 200,
    left: 900,
    backgroundColor: "red",
    width: 100,
    height: 100}).ghost('b', {
        easing: 'easeOut',
        duration: .5,
        remove: false,
        useDisplay: false
    });
```

上面一共有 8 个示例，我把这些示例全部定义在一个页面中，下面是 html 页面的源代码，而脚本定义在 5_5_1.js 文件中。

5_5_1.html

```
<div id="a1">slideIn</div>
<div id="a2">highlight </div>
<div id="a3">frame </div>
<div id="a4">fadeIn</div>
```

```
<div id="a5">fadeOut</div>
<div id="a6">scale</div>
<div id="a7">shift</div>
<div id="a8">ghost</div>
```

看看吧，这就是 Ext.Fx 给我们带来的绚丽效果，太强悍了，Extjs 做出了我们不敢做不会做却很想做的事情，而且非常流畅自然。什么？记不住？别着急，查 API 文档吧，里面有详细的解释和示例，你看着办好了。

六、Ext.Element 类中的动画函数

除了 Ext.Fx 类之外，Ext.Element 也提供了一些动画函数，而且功能并不弱。下面请听我一一道来。

1、setWidth (Number width, Boolean/Object animate)：设置元素宽度

参数：

width：新宽度；

animate：是否以动画方式设置新的宽度，为 true 时有动画效果。也可以配置动画参数。

示例：

```
Ext.onReady(function(){
    Ext.get("e1").applyStyles({
        position: "absolute",
        left: 300,
        top: 100,
        width: 200,
        height: 200,
        backgroundColor: "red"
    }).setWidth(500, true);
})
```

元素 e1 是一个 div 元素（ setWidth ），初始宽度为 200px，通过 setWidth() 方法宽度设置为 500px，并且以动画方式将 div 拉宽。

2、setHeight (Number height, Boolean/Object animate)：设置高度，意义和使用同上。

3、setSize (Number width, Number height, Boolean/Object animate)：同时设置元素的宽度和高度，并设置是否以动画显示。

参数：

width：新的宽度

height：新的高度

animate：是否带有动画效果，或者配置动画参数

示例：

```
Ext.get("e2").applyStyles({
```

```
    position: "absolute",
    left: 100,
    top: 100,
    width: 100,
    height: 100,
    backgroundColor: "red"
}).setSize(500, 500, true);
```

div 的初始大小为 100*100,通过 setSize()将大小变成 500*500,并以动画显示。

4、setBounds (Number x, Number y, Number width, Number height, Boolean/Object animate)：设置元素的位置和大小。

参数：

x：新的左上角 x 坐标

y：新的左上角 y 坐标

width：新的宽度

height：新的高度

animate：是否以动画显示，或者配置动画参数

示例：

```
Ext.get("e3").applyStyles({
    position: "absolute",
    left: 100,
    top: 100,
    width: 100,
    height: 100,
    backgroundColor: "blue"
}).setBounds(0, 0, 500, 500, true);
```

将 div 元素的位置从 100*100 变到 0*0，并将大小从 100*100 变化到 500*500。

5、show (Boolean/Object animate)：显示元素。

6、hide (Boolean/Object animate)：隐藏元素。

七、小结

随着经验的增长，我们会发现，使用 Extjs 来定义视图将彻底摆脱 HTML 标签，HTML 标签被严严实实藏匿起来，所有页面的显示都是基于 javascript 脚本。我不知道 Jack 团队这样做，是好事还是坏事。

从我个人看来，这样做并不好，藏得越深，危害就越大，风险越难以估量。但是反过来想一想，统一的效果、强大的浏览器兼容、优雅的语言、ajax 的完美支持……这些让我忘记了一切，直至奋不顾身。

第六章：元素操作与模板

一、重要也不重要的东西

元素操作通常是指标签的创建、定位、删除和样式定义等。如果使用纯 javascript 而不用任何框架，实现交互性强的页面就必须和元素操作打交道，Extjs 也同样提供了相关的功能，但是，对于程序员来说，似乎并不那么重要了，Extjs 做了十分严密的封装。

模板是 Extjs 中一个非常重要的基础功能，因为需要生成大量的 HTML 标签，使用模板绝对是不二的最佳选择，能带来非常棒的可读性。

但我并不打算花太多时间来讲模板，因为模板被 Extjs 自己用得更多，我们反而不太用。这一章存在的目的是混个脸熟，在阅读源代码或者看别人例子的时候万一碰到不至于惊惶失措。我在课堂上不一定会讲模板这个概念，这个文档可以作为您的补充。

到底重要还是不重要，实践出真知，答案其实在你心里。

二、Ext.DomHelper 类

Ext.DomHelper 类帮助我们使用 javascript 清晰地生成 HTML 代码，他的使用非常灵活，我们应该找准他的主线，像庖丁解牛一样再慢慢剖开。

任何复杂的东西都有他存在的根基，根基就是精髓、是重心。过于浮躁搞不出什么深度，也容易让人丧失自信，掌握一套适合自己的学习方法非常必要，在学习中不断沉淀，完善自我——不仅仅是搞技术，其实任何事情都一样。只要付出了百分之一百的努力，就算没有成功也不会后悔。

下面的方法对您可能有用——

1、insertHtml (String where, HTMLElement el, String html)：在指定的元素上插件 HTML 片段。

参数：

where：插到哪里？可选值有：beforeBegin, afterBegin, beforeEnd, afterEnd

el：参照元素

html：插件的内容

示例：在页面上定义如下 div： 这是一个层

```
Ext.DomHelper.insertHtml("afterBegin", Ext.get("e").dom, "<div>中华人民共和国</div>")
```

在页面上生成的结果如下：

```
<div id="e">
<div>中华人民共和国</div>
```

```
这是一个层
</div>
```

通过修改第一个参数的值，可以总结出四个可选值的含义：
beforeBegin：插入起始标签之前
afterBegin：插入到起始标签之后，本示例就是这种情况
beforeEnd：插入到结束标签之前
afterEnd：插入到结束标签之后

2、下面的方法都差不多，我们通过一个示例来说明他们的用法。
insertBefore (*/ , Object/String o)：新节点插入到指定节点之前；
insertAfter (*/ , Object/String o)：新节点插入到指定节点之后；
insertFirst (*/ , Object/String o)：新建节点并插入到指定节点作为第一个子节点；
append(*/ , Object/String o)：新建节点并插入到指定节点作为最后一个子节点；
overwrite (*/ , Object/String o)：替代指定节点内容；
参数：
*/：指定节点，类型可以为 String/HTMLElement/Element
o：新节点，可以是 dom 对象（子孙）或裸 HTML 标记
示例：
6_6_1.html

```
<div id="parent">
    <div id="c1">第 1 个孩子</div>
    <div id="c2">第 2 个孩子</div>
    <div id="c3">第 3 个孩子</div>
    <div id="c4">第 4 个孩子</div>
    <div id="c5">第 5 个孩子</div>
</div>
```

6_6_1.js

```
//在 c2 之前插入 div
Ext.DomHelper.insertBefore("c2", "<div>c2-child-2</div>");
//在 c2 之后插入 div
Ext.DomHelper.insertAfter("c2", {tag: "div", html: "c2-child"});
//将一个新节点作为 parent 的第一个子节点
Ext.DomHelper.insertFirst("parent", "<div>parent-first-child</div>");
//将 c3 的内容更新
Ext.DomHelper.overwrite("c3", "There are new contents");
//将一个新节点作为 parent 的最后一个子节点
Ext.DomHelper.append("parent", {tag: "div", html: "parent-last-child"});
```

结果：

```
<div id="parent">
    <div>parent-first-child</div>
    <div id="c1">第 1 个孩子</div>
    <div>c2-child-2</div>
    <div id="c2">第 2 个孩子</div>
    <div>c2-child</div>
```

```
    <div id="c3">There are new contents</div>
    <div id="c4">第 4 个孩子</div>
    <div id="c5">第 5 个孩子</div>
    <div>parent-last-child</div>
</div>
```

从例子中可以看出，新建的节点可以指定多种类型：即可以是一段 HTML 标记，也可以是一个 json 对象，对于后者，可以通过 cls 属性来指定类选择器。

三、Ext.XTemplate

Templete 是模板之意，就是定义一段 HTML 代码，并放置若干个{}作为占位符，运行时将数据填充到{}中去。看来看去，和 java 中的 MessageFormat 很像。偷笑中……

XTemplate 和 DomHelper 有很深的渊源，DomHelper 是 XTemplate 的小弟，DomHelper 解决不了的事情，XTemplate 一定可以。

使用 XTemplate 一般会经历三个步骤：
1、定义 XTemplate 对象，指定一段 HTML 代码作为模板；
2、指定 XTemplate 中定义的 HTML 应该放置的位置，并填充占位符信息；
3、编译 XTemplate。

先看一个简单的例子来说明问题：

```
Ext.onReady(function(){
    var xt = new Ext.XTemplate(
        "<table border={0} width={1}>",
            "<tr>",
                "<td>{2}</td>",
                "<td>{3}</td>",
                "<td>{4}</td>",
            "</tr>",
        "</table>"
    );

    xt.append("xt", [1, 300, '单元格 1', '单元格 2', '单元格 3']);
    xt.compile();
})

页面如下：
<div id="xt"></div>
```

实例化 XTemplate 时，可以配置任意个参数，会自动连接到一起，这种写法似乎更方便更好读。append 方法同 DomHelper 的 append 方法，实际上还有 insertBefore、insertAfter、insertFirst、overwrite 等方法，不同的是第二个参数，该参数是要填充到占位符中的数据，可以是数组，也可以是 json 对象。执行完之后得到如下结果：

```
<div id="xt">
    <table border="1" width="300">
        <tbody>
            <tr>
                <td>单元格 1</td>
                <td>单元格 2</td>
                <td>单元格 3</td>
            </tr>
        </tbody>
    </table>
</div>
```

下面是改造后的例子，填充数据时用 json 对象代替了原来的数组：

```
Ext.onReady(function(){
    var xt = new Ext.XTemplate(
        "<table border={b} width={w}>",
            "<tr>",
                "<td>{v1}</td>",
                "<td>{v2}</td>",
                "<td>{v3}</td>",
            "</tr>",
        "</table>"
    );

    xt.append("xt", {b: 1, w: 300, v1: "单元格 1", v2: "单元格 2", v3: "单元格 3"});
    xt.compile();
})
```

四、小结

这一章的内容实在太少，之所以这么说，是因为 XTemplete 的内容实在太多，我们没有再深入，如果不是去扩展 Extjs，而只是单纯使用他的 API，这已经够了。

如果有兴趣，建议你去看看 API 文档，说得很详细，一定不会让你失望。

即使这样，在第七章，我会回过头来，对 XTemplete 作进一步的深入。

第七章：格式化

一、用户需要优秀体验的内容

这一章原本在我的计划之外，是我的疏忽，幸好还来得及。格式化太重要了，显示数据时，我们将面对一大堆来自数据库的数据，这些数据往往只符合外国人的口味，对中国人来说根本读不懂，日期就是一个典型的例子，还有货币、数字格式……

做好本地化工作是程序员的一项基本任务，我们可以在服务器端先格式化数据，再传到客户端显示，显然，这样加重了服务器的负担，消耗了宝贵的资源。更好的解决方案是，通过 Extjs 提供的 Ext.util.Format 类，完全可以将格式化的工作交给客户端去实现。

用户需要的是体验优秀自然的内容，请时刻记住这一点。

二、Ext.util.Format 类

严格来说，Ext.util.Format 并不是一个类，只是一个对象。所以，调用他的方法时并不需要先行实例化，直接调用即可，类似于 java 中的 static 方法。

如果打开他的源代码，基本结构是这样的：

```
Ext.util.Format = function(){
    var trimRe = /^\s+|\s+$/g;
    return {
        //方法定义在此区
    };
}();
```

我们看到，最后是以()结束的，实际上这已经是一个对象了。

Ext.util.Format 定义的方法涉及很多方面，使用起来并不难，对于稍微难理解的我写了示例，妇孺皆知的内容直接跳过演示。

1、ellipsis (String value, Number length) : String
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
参数项：
value : String：要裁剪的字符串
length : Number：允许长度
返回：
String 转换后的文本
示例：

```
var v1 = "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";
Ext.Msg.alert("ellipsis", Ext.util.Format.ellipsis(v1, 10));
```

结果：

```
对大于指定长度...
```

2、undef (Mixed value) : Mixed
检查一个值是否为 underfined，若是的话转换为空值
参数项：
value : Mixed：要检查的值
返回：
Mixed 转换成功为空白字符串，否则为原来的值
示例：

```
var v2 = "underfined";
Ext.Msg.alert("undef", Ext.util.Format.undef(v2));
```

3、defaultValue (Mixed value, String defaultValue) : String
检查一个值（引用的）是否为空，若是则转换到缺省值。
参数项：
value : Mixed：要检查的引用值
defaultValue : String：默认赋予的值（默认为""）
返回：
String
示例：

```
var v3;
Ext.Msg.alert("defaultValue", Ext.util.Format.defaultValue(v3, "这是缺省值"));
```

4、htmlEncode (String value) : String
转义(&, <, >, and ') 为能在 HTML 中显示的字符
参数项：
value : String：要编码的字符串
返回：
String 编码后的文本
示例：

```
var v4 = "<a href='http://www.zz-jb.com'>株洲北大青鸟</a>";
Ext.Msg.alert("htmlEncode", Ext.util.Format.htmlEncode(v4));
```

和下面的代码比较一下就能理解该方法的作用：

```
var v4 = "<a href='http://www.zz-jb.com'>株洲北大青鸟</a>";
Ext.Msg.alert("htmlEncode", v4);
```

5、htmlDecode (String value) : String
将（&, <, >, and '）字符从 HTML 显示的格式还原
参数项：
value : String：解码的字符串
返回：
String 编码后的文本

6、trim (String value) : String
裁剪一段文本的前后多余的空格
参数项：
value : String：要裁剪的文本
返回：
String 裁剪后的文本
关于这个方法，我实在不想再多说什么。

7、substr (String value, Number start, Number length) : String
返回一个从指定位置开始的指定长度的子字符串。
参数项：
value : String：原始文本
start : Number：所需的子字符串的起始位置
length : Number：在返回的子字符串中应包括的字符个数。
返回：
String 指定长度的子字符串
示例

```
var v5 = "中华人民共和国";
Ext.Msg.alert("substr", Ext.util.Format.substr(v5, 2, 2));
```

求子串，这方法在任何语言中似乎都不曾漏掉。

8、lowercase (String value) : String
返回一个字符串，该字符串中的字母被转换为小写字母。
参数项：
value : String：要转换的字符串
返回：
String 转换后的字符串
示例：

```
var v6 = "THE PEOPLE'S REPUBLIC OF CHINA";
Ext.Msg.alert("lowercase", Ext.util.Format.lowercase(v6));
```

9、uppercase (String value) : String
返回一个字符串，该字符串中的字母被转换为大写字母。
参数项：
value : String：要转换的字符串
返回：
String 转换后的字符串
还需要我来举例吗？

10、capitalize (String value) : String
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。
参数项：
value : String：要转换的字符串

返回：
String 转换后的字符串

11、date (Mixed value, [String format]) : Function
将某个值解析成为一个特定格式的日期。
参数项：
value : Mixed：要格式化的值
format : String：（可选的）任何有效的日期字符串（默认为“月/日/年”）
返回：
Function 日期格式函数
示例：

```
var v7 = new Date();//获取当前日期
Ext.Msg.alert("date", Ext.util.Format.date(v7, "Y-m-d H:i:s"));
```

在格式化日期时，Y 表示年，m 表示月，d 表示日，H 表示 24 小时制的小时，h 表示 12 小时制的小时，i 表示分钟，s 表示秒。和 java 不太一样。

关于日期格式化的更多内容，请参考 Ext 帮助文档 Date 部分。

12、stripTags (Mixed value) : String
剥去所有 HTML 标签
参数项：
value : Mixed：要剥去的文本
返回：
String 剥去后的 HTML 标签
示例：

```
var v8 = "<a href='http://www.zz-jb.com'>株洲北大青鸟</a>";
Ext.Msg.alert("stripTags ", Ext.util.Format.stripTags(v8));
```

结果：

```
株洲北大青鸟
```

13、stripScripts (Mixed value) : String
剥去所有脚本（Script）标签
参数项：
value : Mixed：要剥去的文本
返回：
String 剥去后的 HTML 标签

14、fileSize (Number/String size) : String
对文件大小进行简单的格式化（xxx bytes、xxx KB、xxx MB）
参数项：
size : Number/String：要格式化的数值
返回：
String 已格式化的值
示例：

```
var v9 = 2349327423;
Ext.Msg.alert("fileSize", Ext.util.Format.fileSize(v9));
```

结果：

```
2240.5 MB
```

Format 会根据字节的大小自动选择单位。

三、再谈 XTemplete

XTemplate 用于定义一个模板，并将值提供给{}占位符，XTemplate 也能和 Ext.util.Format 配合，将填充的值进行格式化，得到用户想要的任何效果。

基本格式：{index|name:method(params)}
说明：
index：索引
name：json 对象的属性名
method：Ext.util.Format 类的方法名
params：Ext.util.Format 类的方法参数

再花言巧语也需要一个例子来说明，为了迎合本节内容，我们对上一章的示例稍微做了改造，填充的数据类型更加丰富：

```
Ext.onReady(function(){
    var xt = new Ext.XTemplate(
        "<table border={b} width={w}>",
            "<tr>",
                '<td>{v1:date("Y 年 m 月 d 日 H 时 i 分 s 秒")}</td>',
                "<td>{v2:lowercase}</td>",
                "<td>{v3:ellipsis(5)}</td>",
            "</tr>",
        "</table>"
    );

    xt.append("xt", {b: 1, w: 300, v1: new Date(), v2: "CELL2", v3: "这是一段非常长的字符串"});
    xt.compile();
```

从上面代码中看出，v1 为日期类型，按符合中国人口味的日期格式输出；v2 为大写字母，变成小写字母后输出；v3 是一段较长的字符串，只显示一部分，剩余的用“…”来代替。

另外，强调的一点是，输出日期时，如果使用"<td>{v1:date('Y 年 m 月 d 日 H 时 i 分 s 秒')}</td>"（即双引号在外，单引号在内）会产生错误。

四、如果连 Format 都不能满足 XTemplete 的需要呢？

这个问题问得很好，Format 不是万能的，他最多实现了常用的格式化功能，问题是，不同的项目，不同的客户他们的需求千差万别，比如，在显示性别时，不是用文字说明，而是用图片展示，这个要求已经超过了 Format 的极限，庆幸的是，Jack 替我们想到了——使用函数来完成，实际上，我们使用的也是 Format 定义的方法。

来看下面的例子吧：

```
var xt2 = new Ext.XTemplate(
    "您是性别是：{sex:this.sexRender}"
);

xt2.sexRender = function(value){
    return value == "男" ? "<img src='../imgs/134.gif'>" :
          "<img src='../imgs/133.gif'>"
}

xt2.append("xt2", {sex: "女"});
xt2.compile();
```

显示性别时，调用 sexRender 方法，该方法的参数是实际填充的值，我们根据该参数返回不同的图片。this.sexRender 中的 this 是指 xt2 对象，所以，sexRender 必须定义在 xt2 上，否则，Extjs 会从 fm 对象中索取方法，fm 是 Extjs 自己定义的对象。

五、小结

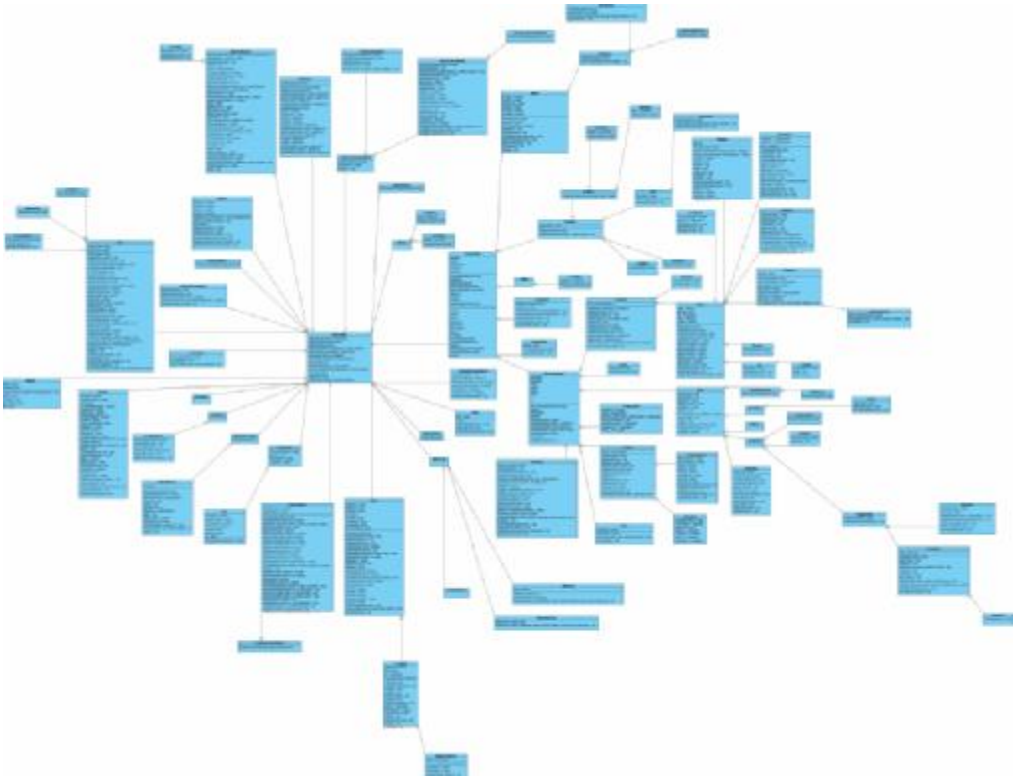
Ext.util.Format 解决了项目开发中经常遇到的格式化问题，在后面章节的学习中，数据格式化还会经常出现，我们已做好准备，迎接更大的挑战。

第八章：Extjs 组件结构

一、Extjs 的组件结构远比我们想象的复杂

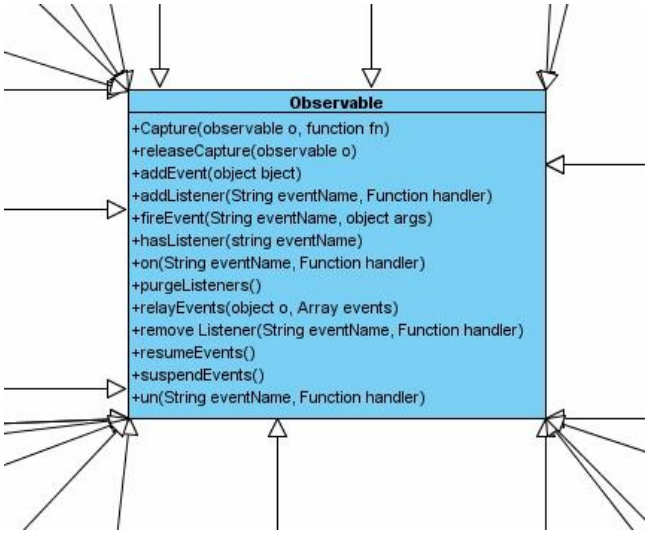
使用 Ext 设计界面，完全颠覆了传统的 HTML 标签和样式，取而代之的是一个庞大的组件体系，HTML 标签和样式被彻底隐藏，知道 VB 吗？知道 Winform 吗？对，更像这些可视化编程技术，一切控件都是预先封装的，我们只需要配置属性、调用方法、触发事件。现在，亟需解决的问题是立刻开发一个“所见即所得”的智能开发环境，目前，网上有了一些，但不足以应用于项目中，功能太少，无法达到产品的要求。

而 Extjs 的组件结构比我们想象的更复杂，希望下面这张图没把你吓倒。



Extjs 组件之间存在着错综复杂的关系，但是，最核心的莫过于 Ext.util.Observable 类，该类是一个抽象基类（Abstract base class），为事件机制的管理提供一个公共接口，是所有组件的父类，这也是“观察者模式”的一种应用。

观察者模式的作用是被观察者一旦发生变化，会通知观察者作出相应的反应。这恰恰和事件处理机制不谋而合。比如，按钮被单击，则触发单击事件；一行被选中，则选择事件被触发；页面加载完成，触发 onReady 事件。可以看看 Ext.util.Observable 类的源代码，并理清各种组件与该类的关系，一定会收获不小。



二、组件分类

大体来讲，组件有三种类型：基本组件、工具栏组件、表单组件。下面是各类型的组件清单。

基本组件

xtype	***class***	***说明***
box	Ext.BoxComponent	具有边框属性的组件
button	Ext.Button	按钮
colorpalette	Ext.ColorPalette	调色板
component	Ext.Component	组件
container	Ext.Container	容器
cycle	Ext.CycleButton	圆角按钮
dataview	Ext.DataView	数据显示视图
datepicker	Ext.DatePicker	日期选择面板
editor	Ext.Editor	编辑器
editorgrid	Ext.grid.EditorGridPanel	可编辑的表格
grid	Ext.grid.GridPanel	不可编辑的表格
paging	Ext.PagingToolbar	分页组件
panel	Ext.Panel	面板
progress	Ext.ProgressBar	进度条
splitbutton	Ext.SplitButton	可分裂的按钮
tabpanel	Ext.TabPanel	选项面板
treepanel	Ext.tree.TreePanel	树
viewport	Ext.ViewPort	视图
window	Ext.Window	窗口

工具栏组件

xtype	***class***	***说明***

toolbar	Ext.Toolbar	工具栏
tbbutton	Ext.Toolbar.Button	按钮
tbfill	Ext.Toolbar.Fill	文件
tbitem	Ext.Toolbar.Item	工具条项目
tbseparator	Ext.Toolbar.Separator	工具栏分隔符
tbspacer	Ext.Toolbar.Spacer	工具栏空白
tbsplit	Ext.Toolbar.SplitButton	工具栏分隔按钮
tbtext	Ext.Toolbar.TextItem	工具栏文本项

表单组件

xtype	***class***	***说明***
form	Ext.FormPanel	Form 面板
checkbox	Ext.form.Checkbox	复选框
combo	Ext.form.ComboBox	下拉列表框
datefield	Ext.form.DateField	日期选择器
field	Ext.form.Field	表单字段
fieldset	Ext.form.FieldSet	字段分组
hidden	Ext.form.Hidden	隐藏表单域
htmleditor	Ext.form.HtmlEditor	在线 HTML 编辑器
numberfield	Ext.form.NumberField	数字编辑器
radio	Ext.form.Radio	单选按钮
textarea	Ext.form.TextArea	区域文本框

三、组件的生命周期

通常情况下，组件架构已经够用了。它的设计目标是组件的大部分管理都对最终开发者透明。不管怎样，当我们需要定制或者扩展组件的时候，了解组件的生命周期会变得非常有用。下面的每一个阶段都是从 Component 继承下来的类的生命周期中重要的阶段。

第一个阶段：初始化

1、应用配置选项

从 Component 继承下来的类并不需要提供（通常没有提供）一个独立的构造器。Component 的构造器不仅应用从子类传过来的配置选项，同时它还做了以下的工作。

2、创建事件

任何组件都有 enabled、disable、beforeshow、show、beforehide、hide、bdforerender、render、beforedesctory、destory 事件，这些事件可以被任何组件调用。

3、在 ComponentMgr 中注册组件实例

这样就可以通过 Ext.getCmp 被获得实例引用。

4、调用 initComponent 方法

这是一个最重要的初始化步骤，它是做为一个模板方法，子类可以按需要重写这个方法。被创建的类的 initComponent 首先被调用，然后通过组件提供的 superclass.initComponent 向上调用。这个方法非常容易被实现，如果需要，你还可以任意覆盖构造器逻辑。

5、状态被始化

如果组件有状态，已保存的状态会被重新加载。

6、加载插件

如果组件有指定插件，则插件会在这时候被初始化。

7、组件呈现

如果有配置 renderTo 或 applyTo，组件会马上被呈现输出，否则，它会被延迟输出，直到组件被显式调用显示，或被它的容器所调用输出。

第二阶段：呈现

1、触发 beforerender 事件

这是一个可取消的事件，如果需要给提供处理函数来阻止组件的继续呈现输出。

2、设置容器

如果没有父容器被指定，默认它的父对象被指定为它的容器。

3、调用 onRender 方法

这是为子类执行呈现工作的一个非常重要的方法，这是一个模板方法，在子类中可以根据需来重写它的实现逻辑。直接被创建的类的 onRender 首先被调用，然后它可以通过 superclass.onRender 来调用基类的 onRender 方法。这个方法很容易被重新实现，如果需要你可以在继承关系的任意类中重写这个方法。

4、不隐藏组件

默认，大多数组件都会通过设置像 x-hidden 这个样式来使它隐藏。当 autoShow 设置为 true 时，这个隐藏功能的样式会被移除。

5、应用自定义样式

所有的 Component 子类都支持指定 cls 配置属性，通过它可以为 Component 所呈现的 HTML 元素指定 CSS 样式。通过添加组件的 cls 属性，使用标准的样式规则，是一个自定义可视组件显示效果的非常完美的方法。

6、render 方法被触发

简单的通知组件已经被成功的呈现了。

7、调用 afterRender

这是另一个模板方法，子类根据逻辑需要可以重新实现或覆盖该方法。所有的子类可以通过调 superclass.afterRender.来调用父类的方法。

8、组件隐藏或不可用

根据配置选项的值来设置。

9、状态事件被初始化

可以状态化的组件会定义一些事件来指定状态的初始化和保存。如果提供，这些事件会被添加。

第三阶段：销毁

1、触发 beforedestroy 事件

这是一个可取消的事件，如果需要，可能通过提供事件代理来阻止组件被销毁。

2、调用 beforeDestroy 方法

又一个模板方法，在子类中可以重新实现和调用父类的方法。

3、移除事件监听者（代理）

如果组件已被呈现，则移除它底层的 HTML 元素的事件监听列表，然后将元素从 DOM 中移除。

4、onDestroy 被调用

这个还是一个模板方法，在子类可以重新实现。这里需要注意的是，容器类提供了一个默认的 onDestroy 实现，它会循环销毁它的成员组件。

5、组件实例从 ComponentMgr 中反注册

不可以再通过 Ext.getCmp 获取到对象实例。

6、destroy 事件被触发

这只是一个简单的提醒，表示组件销毁成功。

7、移除 Component 上的事件代理

组件可以独立于元素，自己拥有事件代理，如果存在则移除它们。

本节是老外写的，被翻译成中文，怎么看怎么别扭。

四、组件渲染方法 render

在类 Ext.Component 中，定义了一个名为 render 的方法，该方法能将组件的最终效果在页面上呈现出来。通过对该方法源代码的分析，我们可以大概知道 Extjs 在渲染组件时的过程。

```
render : function(container, position){
  //如果还没有被渲染 并且 beforerender 方法返回值为 true,则进行渲染，这样，确保了对于组件仅进行一次渲染； position 参数指定了组件被插入容器的位置（即在 position 指定的元素前插入组件）
   if(!this.rendered     &&     this.fireEvent("beforerender", this) !== false){
       //没有传入任何参数（即未指定容器 container）并且设置了 this.el，增设配置 this.container 属性
       if(!container && this.el){
           this.el = Ext.get(this.el);
           //将父节点作为自己的容器
           container = this.el.dom.parentNode;
           this.allowDomMove = false;
       }
       this.container = Ext.get(container);
       // 如 果 配 置 了 ctCls, 对 container 进 行 ctCls 的 渲染,ctCls(Container Class)是容器的渲染类名，cls(Class)是元素的渲染类名
       if(this.ctCls){
           this.container.addClass(this.ctCls);
       }
       this.rendered = true;
       //设置 position
       if(position !== undefined){
           if(typeof position == 'number'){
               position                                    =
this.container.dom.childNodes[position];
```

```
            }else{
                position = Ext.getDom(position);
            }
        }
        this.onRender(this.container, position || null);
        //如果设置了autoShow,则移除x-hidden和x-hide-hideMode(根据
hideMode 该属性可以配置为 display,visibility,offsets 三种属性),从这
个方法可以看出，一搬来说，组件创建后缺省的模式为hidden或者none
        if(this.autoShow){
            this.el.removeClass(['x-hidden','x-hide-'          +
this.hideMode]);
        }
        //如果设置了cls,则对元素进行渲染
        if(this.cls){
            this.el.addClass(this.cls);
            delete this.cls;
        }
        //如果设置了style,则对元素Style属性进行设置
        if(this.style){
            this.el.applyStyles(this.style);
            delete this.style;
        }
        //触发fireEvent和AfterRender事件
        this.fireEvent("render", this);
        this.afterRender(this.container);
        //如果设置了hidden和disabled则进行相应的处理
        if(this.hidden){
            this.hide();
        }
        if(this.disabled){
            this.disable();
        }
```

```
        if(this.stateful !== false){
            this.initStateEvents();
        }
    }
    return this;
}
```

以下是 onReader()方法的源代码：

```
onRender : function(ct, position){
    //如果配置了autoEl属性，则根据autoEl属性生成el属性， 如果autoEl
属性为字符串，则根据字符串生成元素；否则，则在autoEl指定的元素外包裹一层
```

```
div 元素
    if(this.autoEl){
        if(typeof this.autoEl == 'string'){
            this.el = document.createElement(this.autoEl);
        }else{
            var div = document.createElement('div');
            Ext.DomHelper.overwrite(div, this.autoEl);
            this.el = div.firstChild;
        }
        if (!this.el.id) {
           this.el.id = this.getId();
        }
    }
    //把 position 元素插入到 el 元素前
    if(this.el){
        this.el = Ext.get(this.el);
        if(this.allowDomMove !== false){
            ct.dom.insertBefore(this.el.dom, position);
        }
        if(this.overCls) {
            this.el.addClassOnOver(this.overCls);
        }
    }
}
```

五、小结

本章介绍了 Extjs 组件的结构以及各组件的生命周期，并重点分析了组件的渲染方法 render()，不得不承认，这部分内容有点复杂，而且非常抽象，不过没关系，可以先跳过本章节，获取更多的感性和理性认识后，回过头来，会发现更多意想不到的惊喜。

了解组件的生命周期，有利于把握所有组件的共性，为编程提供更强的灵活性。

认真的男人是最帅的，认真的女人是最漂亮的。我真的不认为你的长相和你的衣着能决定你的内含和气质。让我们做一个认真的人。

第九章：按钮与日期选择器

一、开始组件学习之旅

哈哈，你一定很开心。

前面听我讲了这么多，可是突然发现什么也做不了，一定失望透顶。如果真是这样，你一定是个急性子。这样可不行。“万丈高楼平地起”，“冰冻三尺，非一日之寒”，踩在云朵上，迟早会重重地掉到地上。诚然，我们先前看到的全是基础，但是，请记住，基础永远最重要，评价一个人是否是牛人，不在于他知道多少框架，而在于他的基础有多牢固。

根扎得越深，站得就越稳。

现在，我要开始讲组件了，把项目中所有要用到的组件全部讲得通通透透。我无法预知你的智商，但我相信我自己。只要我会，就一定能讲清楚。因此，你没有理由不开心，不是吗？

你最好还是做点心理准备，Extjs 的控件多而杂，虽然有十足的信心，我还真怕把我自己讲糊涂。我会尽可能用简单的语言来描述复杂的逻辑，但是，同样会用复杂的代码来解释简单的概念。人生就是这样，十年一个轮回，由不得自己去控制。

二、被设计得面目全非的按钮

我敢保证，你看了按钮生成的源代码后，一定会大跌眼镜。

```
<table id="ext-comp-1001" class="x-btn-wrap x-btn" cellspacing="0"
    cellpadding="0" border="0" style="width: auto;">
    <tbody>
        <tr>
            <td class="x-btn-left">
                <i> </i>
            </td>
            <td class="x-btn-center">
                <em unselectable="on">
                    <button id="ext-gen8" class="x-btn-text" type="button">
                        确定
                    </button> </em>
            </td>
            <td class="x-btn-right">
                <i> </i>
            </td>
```

```
        </tr>
    </tbody>
</table>
```

这是一个最简单的按钮，却被层层封装，最后，才看到最关键的语句<button id="ext-gen8" class="x-btn-text" type="button">。从代码中，我们发现大量类选择器被使用，这些选择器定义在名为 resources/css/ext-all.css 的样式文件中。按钮的效果完全被重画了。

生成的这段代码其实是通过模板来定义的：

```
Ext.Button.buttonTemplate = new Ext.Template(
    '<table border="0" cellpadding="0" cellspacing="0" class="x-btn-wrap"><tbody><tr>',
        '<td class="x-btn-left"><i> </i></td><td class="x-btn-center"><em
unselectable="on"><button class="x-btn-text" type="{1}">{0}</button></em></td><td
class="x-btn-right"><i> </i></td>',
        "</tr></tbody></table>"
);
```

按钮控件用 Ext.Button 来表示，有三种类型：提交（submit）、重置（reset）和普通按钮（button）。对于按钮来说，最重要的就是触发单击事件了，有点像古代青楼里的妓女，习惯了被人指点。可怜的人！

按照 OOP 的习惯，我们会这样定义按钮：

```
var btn = new Ext.Button();
btn.setText("确定");                                    //按钮上的文字
btn.type = "submit";                                    //按钮类型
btn.setHandler(function(){                              //按钮被单击后执行本方法
    Ext.Msg.alert("按钮", "按钮测试，效果真好");
});
btn.render(Ext.getBody());                              //将按钮放在指定位置显示
```

太完美了，完全吻合面向对象编程的思想，创建对象、设置属性、响应事件，如行云流水，一气呵成。但是，等下，Extjs 更加推崇下面的做法，完全通过配置来实现。

```
var btn = new Ext.Button({
    renderTo: Ext.getBody(),
    text: "确定",
    type: "submit",
    handler: function(){
        Ext.Msg.alert("按钮", "按钮测试，效果真好");
    }
});
```

构建 Button 时，传递一个 json 对象，renderTo 是一个经常被使用的属性，用于指定当前组件渲染在什么位置，一般作为指定对象的子节点，本例中的 Ext.getBody()方法返回 document.body 对象，所以，btn 按钮将是 body 的子节点，对应的方法是 render()。而 handler 属性则对应 setHandler()方法，用来定义按钮被单击之后的处理函数。

第二种作法更加常见，第一种反而用得很少。从 API 文档中找不到特别直白的帮助，很多时候，必须结合 API 文档和源代码，才能获取有用的信息。这让人很头痛。唯一的办法是在学习和工作中不断总结经验，多看例子，多阅读源代码。还是那句话，站在巨人的肩膀上，我们会看得更远。

在 Ext.Button 的源代码中，发现有这样一句话：Ext.reg('button', Ext.Button)，第一个参数“button”是控件的 xtype 属性值，简单来说就是 Ext.Button 类的另一个名字，很多场合，通过 xtype 可以简化属性的配置。

下面的另外一些配置能增强按钮的效果：

pressed: true	使按钮处于按下状态
disabled: true	使按钮处理禁用状态
minWidth: 100	设置按钮的最小宽度
icon: "../imgs/133.gif"	设置按钮的背景图片，属性值是图片名称
iconCls: "bk"	同上，属性值是类选择器名称

三、日期选择器 Ext.DatePicker

这个组件简直让我爱不释手，一个非常不错的组件，他允许我们选择任何一天的日期，遗憾的是，也只能选择日期，不能选择时间。不过，他的优秀和强大足以抵消所有的缺点。我们应该以更加友善的眼光打量他、欣赏他。

我一直相信，代码是有生命的，组件亦然。我们应该学会倾听，学会洞察。程序员和代码是一个整体，彼此更应该惺惺相惜。只有这样，我们的代码才会有生命力，有活力。

Ext.DatePicker 类定义了一个用于选择日期的组件，他的效果就像这样：

说起来，Ext.DatePicker 并不复杂，平时用得最多的就是获取用户选择的日期，这是通过该类的 getValue()方法来得到的。其他很多配置选项主要是用来作个性化和本地化，只和表现有关，和功能无关。

看例子：

```
var dp = new Ext.DatePicker({
    renderTo: Ext.getBody(),
    minDate: Date.parseDate("2009-1-1", "Y-m-d"),
    maxDate: Date.parseDate("2009-12-30", "Y-m-d"),
    value: Date.parseDate("2009-12-30", "Y-m-d"),
    handler: function(){
        Ext.MessageBox.alert("日期", Ext.util.Format.date(this.getValue(), 'Y-m-d'));
    }
});
```

前面讲过的配置选项就不重复了，以后也不会重复。minDate 和 maxDate 是配置可选日期的最小值和最大值，value 是日期控件显示后的初始值。通过 getValue()方法得到用户选择的日期，再通过 Ext.util.Format.date 格式化。

四、小结

本节的主要目的是引导大家使用基本组件，提高熟练度，为日后应用更加复杂的组件打下良好的基础。

第十章：数据与 ComboBox

一、数据在这里是动词

我说的“数据”，其实就是如何获取数据。这本该是一个动词。

在 Extjs 中，对于选择控件来说，如同一场噩梦，刚刚接触的时候不被搞晕也会被搞傻。Jack 把所有人都当成了天才。我们不能再用传统的方法为选择控件初始化选项，而是必须使用 Ext.data 命名空间中的类来组织和管理数据项。

之所以把 Ext.data 和 ComboBox 组件放到同一个章节，是为了我在讲解 Ext.data 命名空间中的类时举例方便。Ext.data 最主要的功能是获取和组织数据结构，并和特定控件联系起来，于是，Ext.data 成了数据的来源，控件负责显示数据。他们分工是如此明确，但是又如此密不可分。以至于拆开任何一方都天理不容。

我们现在的迫切任务是理解 Ext.data 里的类，在该命名空间中，有三个东东非常厉害也非常难懂，分别是 DataProxy、DataReader 和 Store。理解这三个类成了灵活应用 Extjs 的关键，那么，我该怎么解释才能达到最理想的效果？这真是个难题。

我试着用最简单的语言来描述 DataProxy、DataReader 和 Store 的作用，不一定面面俱到，但应该能说明问题。

DataProxy：获取想要的数据，通过他能得到来自不同地方的数据，如数组、远程服务器，并组织成不同的格式。
DataReader：定义数据项的逻辑结构，一个数据项有很多列，每列的名称是什么，分别是什么数据类型，都由该类来定义。另外，还负责对不同格式的数据进行读取和解析。
Store：存储器，用于整合 Proxy 和 Reader，控件索取数据时通常和他打交道。

似乎还是很抽象，接下来我会讲得更详细点。

二、Ext.data.DataProxy 类

proxy 是代理的意思，很多时候，聪明的架构师和设计者为了屏蔽底层的差异，给用户提供一个统一的访问接口，会设计一个名为“proxy”的类。这种设计哲学在架构中普通应用，并且解决了很多实际问题，代码也变得更加优雅，并有效降低代码侵入。

Ext.data. DataProxy 就是来源于这样一种灵感。

Ext.data.DataProxy 是获取数据的代理，数据可能来自于内存，可能来自于同一域的远程服务器数据，更有可能来自于不同域的远程服务器数据。

但是，在实际应用中，我们不会直接使用 Ext.data.DataProxy，而是使用他的子类：MemoryProxy、HttpProxy 和 ScriptTagProxy，他们的作用分别是：

MemoryProxy：获取来自内存的数据，可以是数组、json 或者 xml。

HttpProxy：使用 HTTP 协议通过 ajax 从远程服务器获取数据的代理，需要指定 url。

ScriptTagProxy：功能和 HttpProxy 一样，但支持跨域获取数据，只是实现时有点偷鸡摸狗。

在本章中，我们只学习 MemoryProxy 的使用，更多的内容在未来的章节中会慢慢出现，目的是为了不吓到您。其实我们的章节很短也是因为这个原因。从初学者的角度看，大家更愿意接受简言意赅的知识。希望我的这个出发点没错。

因为数据最终要显示在 ComboBox 中，所以我们以 ComboBox 为中心进行设计。大家应该知道，下拉列表框有两个值，一个是显示值，另一个是实际值，也就是说，一个数据项包含两列：显示值列和实际值列。我们定义一个用来保存城市名称的二维数组：

```
var cities = [
    [1, "长沙市"],
    [2, "株洲市"],
    [3, "湘潭市"],
    [4, "邵阳市"]
];
```

在二维数组中，每一个城市保存了两个值：值一表示城市编号，作为实际值，值二表示城市名称，作为显示值。然后，我们将 data 构建出一个 MemoryProxy 对象：

```
var proxy = new Ext.data.MemoryProxy(data);
```

OK，就是这样。没有太多的技巧。

三、Ext.data.DataReader 类

DataReader 从来不是单独行动的，他没有太多的自主权，总是看 DataProxy 行事。总体来说，DataReader 用来定义数据项（行）的逻辑结构，主要信息有：列的逻辑名称（name）、列的数据类型（type）、列与数据源的索引映射（mapping）等，另外，还包含一些元数据，如分页信息。

实际上，每一个数据项都是一个 Ext.data.Record（记录）对象，而数据项的列信息则是通过 Ext.data.Record 来定义的。Ext.data.Record 并没有固定的结构，他保存的是一个 json 对象数组，数组的元素个数由列的数量来决定。在上面的例子中，城市包含 ID 和名称，所以，必须在数组中定义两个元素，基本就是这个样子：

```
var City = Ext.data.Record.create([
    {name: "cid", type: "int", mapping: 0},
    {name: "cname", type: "string", mapping: 1}
]);
```

我们定义了一个 City 的结构，通过 Ext.data.Record.create 创建，参数是一个 json 对象数组，name 和 type 分别表示每一列的名称和数据类型，mapping 是列值与数组元素的映射关

系。

Record 创建好后，必须和 DataReader 关联，DataReader 也同样有三个子类：Ext.data.ArrayReader、Ext.data.JsonReader、Ext.data.XmlReader。我之前说过 DataReader 从来不单独行动，使用哪一个子类主要取决于 DataProxy 中封装的数据类型，如果是数组，则使用 Ext.data.ArrayReader；如果是 json，则使用 Ext.data.JsonReader；如果是 xml，则使用 Ext.data.XmlReader。在本教程中，我打算将 xml 封杀。我不喜欢这个东西（尽管有时候不得不面对）。相对而言，我更热衷于轻量级的 jsonObject。

在本例中，我们处理的数据类型是数组，所以自然要使用 ArrayReader。

```
var reader = new Ext.data.ArrayReader({}, City);
```

构造 ArrayReader 对象时，构造函数的第一个参数就是元数据，第二个参数则是 Record。也可以一步到位：

```
var reader = new Ext.data.ArrayReader({}, [
    {name: "cid", type: "int", mapping: 0},
    {name: "cname", type: "string", mapping: 1}
]);
```

四、Ext.data.Store 类

这个类相对简单，不需要面对数据和结构，只是把 DataProxy 和 DataReader 整合在一起，这样一来，数据有了，结构有了，俨然就是一张数据表，想一想数据库中的物理表是不是就是这样的呢？嗯，非常像。

典型的写法像这样：

```
var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader
});
```

到了这一步，别以为大功告成了，其实这时候数据并没有加载到 Store 中，默认情况下，Store 采取延时加载，必须显式调用 load()方法，当然，我们也可以采取即时加载策略，按如下配置即可：

```
var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader,
    autoLoad: true //即时加载数据
});
```

我们画张图，用来描述这三个类之间的关系。

可以看出，Ext.data.Store 的主要目的是在内存中建立一张数据表，填充到组件中，这些组件形态也千差万别，最典型的就是 ComboBox 和 GridPanel。

五、下拉列表框

下拉列表框被定义成 ComboBox 类，位于 Ext.form 命名空间，和 Button 不同，他是一个表单域组件，常用于表单中。

如果我们坚持认为 Extjs 把按钮改得面目全非，可能是您还没有见识到 ComboBox 的厉害，传统的页面设计使用<select>标记作为下拉列表框，但 Extjs 中的 ComboBox 和 select 标记一点儿关系也没有了。不得不佩服人类的创造力。

我们有一个这样的 ComboBox，该组件用来显示湖南的城市。

看起来和 select 标记别无二致，只是更加漂亮了。来看看 Extjs 帮我们生成的静态代码，但愿您看了后依旧安然无恙。

```
<div id="ext-gen7" class="x-form-field-wrap" style="width: 164px;">
    <input id="ext-comp-1001"
        class="x-form-text x-form-field x-form-empty-field" type="text"
```

```
        name="ext-comp-1001" autocomplete="off" size="24" />
    <img id="ext-gen8" class="x-form-trigger x-form-arrow-trigger"
        src="http://extjs.com/s.gif" />
</div>
<div id="ext-gen32" class="x-shadow" style="z-index: 10999; left: 1px; top: 25px; width: 172px; height: 86px; display: none;">
    <div class="xst">
        <div class="xstl" />
        <div class="xstc" style="width: 160px;" />
        <div class="xstr" />
    </div>
    <div class="xsc" style="height: 74px;">
        <div class="xsml" />
        <div class="xsmc" style="width: 160px;" />
        <div class="xsmr" />
    </div>
    <div class="xsb">
        <div class="xsbl" />
        <div class="xsbc" style="width: 160px;" />
        <div class="xsbr" />
    </div>
</div>
<div id="ext-gen22" class="x-layer x-combo-list" style="position: absolute; z-index: 11000; visibility: hidden; left: -10000px; top: -10000px; width: 162px; height: 84px;">
    <div id="ext-gen24" class="x-combo-list-inner" style="overflow: auto; width: 162px; height: 84px;">
        <div class="x-combo-list-item x-combo-selected">
            长沙市
        </div>
        <div class="x-combo-list-item">
            株洲市
        </div>
        <div class="x-combo-list-item">
            湘潭市
        </div>
        <div class="x-combo-list-item">
            邵阳市
        </div>
    </div>
</div>
```

生成的代码和 select 扯不上任何关系了，取而代之的是 N 个 div，一下变得无比复杂，这就是美丽的代价。

一起来熟悉一个比较典型的 ComboBox 的定义方法：

```
var combobox = new Ext.form.ComboBox({
    renderTo: Ext.getBody(),
    triggerAction: "all",
    store: store,
    displayField: "cname",
    valueField: "cid",
    mode: "local",
    emptyText: "请选择湖南城市"
});
```

嘿嘿，如果不解释您一定会愣上半天，有时候真不习惯 Extjs 中属性名称的定义风格，也许只怪我们的头发太长见识太少，来认识一下吧。

triggerAction：是否开启自动查询的功能，为 all 表示不开启，为 query 表示开启，默认为 query；

store：不用解释了，和前面的 Ext.data.Store 对象挂勾，定义数据源；

displayField：关联 Record 的某一个逻辑列名作为显示值，本例为城市名称；

valueField：关联 Record 的某一个逻辑列名作为实际值，本例为城市 ID；

mode：可选值有 local 和 remote，如果数据来自本地，用 local，如果数据来自远程服务器，必须用 remote，默认为 remote；

emptyText：没有选择任何选项的情况文本框中的默认文字。

请理解并记住这个基本的用法，很多东西都是这样巩固消化的。当我们面对如海水般涌来的信息，第一感觉就是无所适从，继而惊惶失措，夺路而逃，于是被永远挡在真理的大门之外。成功和失败，只在一念之间。

抓住核心点之后，慢慢扩展，知识架构便会越来越丰满，因为把握好了整个知识点的精髓，再怎么变化，都不至于迷路。

六、得到下拉列表框的值

下拉列表框（ComboBox）有两个值，显示值和实际值。显示值为用户提供了更为直观和理想的体验，而实际值则会传送到服务器，供服务器处理。

ComboBox 定义了两个方法，其中，getValue()用于返回实际值，getRawValue()用于返回显示值。如果让我们自己来猜，估计永远想不到显示值和 getRawValue()方法有联系，太神奇了！

我在页面上放置了一个按钮，点击按钮显示下拉列表框的值：

```
var btn = new Ext.Button({
    text: "列表框的值",
    renderTo: Ext.getBody(),
    handler: function(){
```

```
        Ext.Msg.alert("值", "实际值：" + combobox.getValue() + "；显示值：" + combobox.getRawValue());
    }
});
```

七、源代码

下面是本章示例的源程序：

```
Ext.onReady(function(){
    var cities = [
        [1, "长沙市"],
        [2, "株洲市"],
        [3, "湘潭市"],
        [4, "邵阳市"]
    ];

    var proxy = new Ext.data.MemoryProxy(cities);

    var City = Ext.data.Record.create([
        {name: "cid", type: "int", mapping: 0},
        {name: "cname", type: "string", mapping: 1}
    ]);

    //var reader = new Ext.data.ArrayReader({}, City);

    var reader = new Ext.data.ArrayReader({}, [
        {name: "cid", type: "int", mapping: 0},
        {name: "cname", type: "string", mapping: 1}
    ]);

    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader,
        autoLoad: true //即时加载数据
    });

    //store.load();

    var combobox = new Ext.form.ComboBox({
        renderTo: Ext.getBody(),
        triggerAction: "all",
        store: store,
        displayField: "cname",
```

```
        valueField: "cid",
        mode: "local",
        emptyText: "请选择湖南城市"
    });

    var btn = new Ext.Button({
        text: "列表框的值",
        renderTo: Ext.getBody(),
        handler: function(){
            Ext.Msg.alert("值", "实际值：" + combobox.getValue() + "；显示值：" +
combobox.getRawValue());
        }
    });
})
```

八、小结

本章真的很重要，一定要想方设法看明白，然后一字一字将例子敲出来，再理解其中的意思。以后，只要碰到数据显示的问题，我就会把这章的东西搬出来，稍微改动应用到另一个知识点中去。

而且，ComboBox 的东西也不止这些，应该说，ComboBox 算是一个比较复杂的组件，我们现在接触的只是最基本的用法，在后面我会继续和您一起深入探讨。

熟练掌握本章，就是您现在的使命。如果依旧一知半解，建议您折回去，重新开始……

第十一章：Ajax 与 ComboBox

一、Ajax

ajax 风靡已久，google 在 gmail 中掀起了一股无页面刷新运动，现在成民皆 ajax。ajax 给用户带来了前所未有的用户体验，借助浏览器自带的 ajax 引擎，在不刷新页面的情况下，能够从服务器获取数据，且“按需所取”，减少了宽带流量，增强了使用感受。

我没有在这里讲解 ajax 的计划，您可以参考其他书籍，网上的教程遍地开花，ajax 本身并不高深，只需要花少量的时间便可学会。多实践，有利于更好的理解和消化。

Extjs 对 ajax 的支持相当全面而完善，只要涉及到与服务器的交互，都是通过 ajax 来完成的。如果没有 ajax，Extjs 也会失去他现有的光彩。当然，在提交表单的时候，Extjs 并不强迫我们使用 ajax，但是，使用 ajax 仍然是一个不错的选择。

Extjs 对 ajax 的支持主要体现在两个方面，其一，使用专门的类封装 ajax 底层，提供一个简单的接口供用户调用，能大大简化 ajax 技术的实现；第二种方法更直接，很多组件本身就内置对 ajax 的支持，只要设置几个配置选项就万事大吉，当然，HttpProxy 功不可没。

我在本章会重点讲述 ComboBox 如何通过 ajax 从服务器获取数据并初始化选项，这是一个激动人心的功能，前提是您应该先掌握第十章的内容。

二、Ext.Ajax 类

如果您曾经见过 Ext.lib.Ajax 类，一定会跳起来，为什么不是 Ext.lib.Ajax 而是 Ext.Ajax 呢？我们都没错，二者都是对 ajax 技术的封装，如果从服务器返回的是 text/html 数据，用哪一个都没关系。在 Ext2.x 中，Ext.lib.Ajax 已被定义成为最基本的功能，在 ext-base.js 文件里可以找到源代码。而 Ext.Ajax 则是对 Ext.lib.Ajax 的封装和扩展，功能更加强大，已经支持文件上传了——真应该狂笑三声。

具体说来，Ext.Ajax 并不是一个类，而是一个对象，所以，我们并不需要实例化。如果不理解我说的这句话，建议您去看看源代码，所有的真相都会水落石出。可以告诉您的是，Ext.Ajax 是 Ext.data.Connection 类型的对象，很神奇吧？

纵然迂回婉转，我们并不需要关注太多细节，Ext.Ajax 有一个非常重要的方法：request()，该方法是我们实现 ajax 的基础，他的基本使用方法如下：

```
Ext.Ajax.request({
    url: "../TimeServlet",
    success: function(response, config){
        alert(config.url + "," + config.method);
```

```
            Ext.MessageBox.alert("result", response.responseText);
        },
        failure: function(){
            Ext.MessageBox.alert("result", "请求失败");
        },
        method: "post",
        params: {name: "李赞红"}
});
```

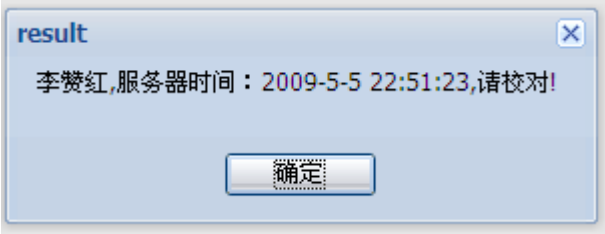
```
//TimeServlet.java
response.setContentType("text/html;charset=utf-8");
PrintWriter out = response.getWriter();
out.println(request.getParameter("name") + ",服务器时间：" + new Date().toLocaleString()
+ ",请校对!");
out.flush();
out.close();
```

这几个配置选项很容易理解，url 是接受请求的组件，此处为 Servlet，请求将发送到该组件进行处理；method 是请求的类型，不是 get 就是 post；params 则是请求参数，是一个 json 对象，可以传递更多的参数；而 success 和 failure 是回调函数，如果请求成功，自动调用 success()方法，失败则调用 failure()方法。

在 success 回调方法中，有两个参数 response 和 config，其中，从服务器返回的数据保存在 response.responseText 属性中，而 config 则保存 Ext.Ajax.request()方法的配置选项信息。

建议大家使用 post 发送请求，中文更容易处理，如果使用 get 可能会面临更多的问题。

示例中的结果如下图：



接下来演示服务器返回 json 对象的情形：

```
Ext.Ajax.request({
    url: "../TimeServlet",
    success: function(response, config){
        alert(config.url + "," + config.method);
        var json = Ext.util.JSON.decode(response.responseText);
        Ext.MessageBox.alert("result", json.author + "," + json.time);
    },
```

```
    failure: function(){
        Ext.MessageBox.alert("result", "请求失败");
    },
    method: "post",
    params: {name: "李赞红"}
});
```

```
//TimeServlet.java
String name = request.getParameter("name");
response.setContentType("text/html;charset=utf-8");
PrintWriter out = response.getWriter();
out.println("{author: '" + name +"', time: '" + new Date().toLocaleString() +"'}");
out.flush();
out.close();
```

和上面的基本差不多，讲讲不同点。从服务器返回的数据格式不再是一段简单的文本，而是一段符合 json 格式的字符串，这段字符串返回到客户端后，由 Ext.util.JSON.decode 负责转成 json 对象。看看 Ext.util.JSON.decode()方法的源代码，直白得超乎我们的想象。

```
this.decode = function(json){
    return eval("(" + json + ')');
};
```

三、Ajax 文件上传

似乎应该澄清一个误区，ajax 并不支持传送流数据，这实在是一个并不高明的设计，我固执的以为，不管从哪个角度来看，支持流数据的传递应该成为浏览器的标准，可惜我不是标准的制定者和开发者。

即使这样，却并不代表上传文件必须刷新页面，睿智的开发者自然能探索出一条可行的路子，比较普通的做法是使用一个不可见的 iframe 内嵌框架完成这个艰巨的任务，iframe 比较难以控制，庆幸的是，Extjs 内置了这种思想的实现，下面是 Extjs 实现的源码：

```
doFormUpload : function(o, ps, url){
    var id = Ext.id(); //生成一个唯一的 ID 值
    var frame = document.createElement('iframe'); //创建 iframe 标签
    //将 id 和 name 设置成相同的值
    frame.id = id;
    frame.name = id;
    frame.className = 'x-hidden';//设置样式隐藏 iframe，使之不可见
    if(Ext.isIE){
        frame.src = Ext.SSL_SECURE_URL;
    }
```

```
    document.body.appendChild(frame);//将创建的 iframe 添加到页面

    if(Ext.isIE){
        document.frames[id].name = id;
    }

    var form = Ext.getDom(o.form);//获取表单对象
    form.target = id; //这句话很关键，是连接表单和 iframe 的桥梁
    form.method = 'POST';//文件上传必须设置成 post 请求
    form.enctype = form.encoding = 'multipart/form-data';//这也是文件上传必须设置
的
    if(url){
        form.action = url;//表单提交的地址
    }

  //下面的代码段用于将 params 中的参数保存到隐藏字段中，随表单一起提交
    var hiddens, hd;
    if(ps){ // add dynamic params
        hiddens = []; //保存所有创建的隐藏表单域
        //ps 其实就是 params，从 Ext.Ajax.request 方法中传过来
        ps = Ext.urlDecode(ps, false);
        //循环遍历每一个参数
        for(var k in ps){
            if(ps.hasOwnProperty(k)){
                //创建隐藏标签
                hd = document.createElement('input');
                hd.type = 'hidden';
                hd.name = k;//设置隐藏标签的名称
                hd.value = ps[k];//设置隐藏标签的值
                form.appendChild(hd);//将新创建的标签添加到表单中
                hiddens.push(hd);//将新建的隐藏表单域放入数组中方便后面移除
            }
        }
    }

    function cb(){
        var r = {   // bogus response object
            responseText : '',
            responseXML : null
        };

        r.argument = o ? o.argument : null;

        try { //
```

```
                var doc;
                if(Ext.isIE){
                    doc = frame.contentWindow.document;
                }else {
                    doc = (frame.contentDocument || window.frames[id].document);
                }
                //设置返回到客户端的值（字符串）
                if(doc && doc.body){
                    r.responseText = doc.body.innerHTML;
                }
                //设置返回到客户端的值（XML）
                if(doc && doc.XMLDocument){
                    r.responseXML = doc.XMLDocument;
                }else {
                    r.responseXML = doc;
                }
            }
            catch(e) {
                // ignore
            }

            Ext.EventManager.removeListener(frame, 'load', cb, this);
            //触发 requestcomplete（请求完成）事件
            this.fireEvent("requestcomplete", this, r, o);
            //上传成功调用 success 方法
            Ext.callback(o.success, o.scope, [r, o]);

            Ext.callback(o.callback, o.scope, [o, true, r]);
            //延时 0.1 秒后，删除前面创建的 frame，延时可能是为了安全起见
            setTimeout(function(){Ext.removeNode(frame);}, 100);
        }

        Ext.EventManager.on(frame, 'load', cb, this);
        form.submit();//提交表单

        if(hiddens){ // 移除前面创建的隐藏表单域
            for(var i = 0, len = hiddens.length; i < len; i++){
                Ext.removeNode(hiddens[i]);
            }
        }
    }
```

哈，又开始摆源码了，不过我加了注释，过去我在做 ajax 文件上传时，正是使用了这种方法，所以，这些代码对我来说很熟悉。比较而言，extjs 做得更好，他的很多东西都是

自动创建删除的，我喜欢这种方式——要的时候创建，不要的时候删除，而不是像小偷一样躲来躲去。

如果不太明白源码的真实含义，能熟练使用依然难能可贵。ajax 文件上传一般遵循下面三个步骤：

1、创建文件上传表单；

2、调用 Ext.Ajax.request()方法实现文件上传；

3、定义文件上传处理器，并结合开源的文件上传组件（如 cos），将数据流转换成文件和参数。

但在开发的时候，上面的顺序是倒过来的。我打算结合代码来说明整个实现过程。

第一步：定义 Web 组件，完成文件上传的请求，当然，在上传文件时候，也会有其他的数据，我们照单全收。要注意的是，Extjs 不负责将流数据保存为物理文件，我们得自己写代码解析。为了简化代码，使用开源的文件上传组件值得推荐。我手头有一个 cos 组件，完全可以满足这个需求。

去花点时间了解 cos 吧，或者别的也行。我没有讲解 cos 的计划，哈哈哈。

```
package com.aptech.ajax;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.oreilly.servlet.MultipartRequest;

public class FileUpServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        MultipartRequest multi = new MultipartRequest(
                request,
                "c:\\", //文件上传后保存的位置
                10 * 1024 * 1024, //允许的最大文件大小
                "utf-8"  //编码
        );

        System.out.println("文件名：" + multi.getFilesystemName("f"));
        System.out.println("文件描述：" + multi.getParameter("desc"));
    }
```

```
}
```

这个……我们平时不也是这样做的吗？没错，服务器组件和视图组件根本就扯不上关系。如果换掉 extjs，用别的技术作视图，对服务器组件没有任何影响。

第二步：设计上传界面

```
<form id="upform">
    请选择文件：<br><input type="file" name="f"><br>
    文件描述：<br><textarea rows="5" cols="70" name="desc"></textarea><br>
    <input type="button" id="btn" value="上传">
</form>
```

这个表单貌似精简了许多，没有为 form 指定 method 和 action，更没有指定 enctype，如果您细心点，肯定会在 doFormUpload 方法中发现玄机，这些属性通通在代码中设定了，我们单独提取出来看个究竟：

```
    var form = Ext.getDom(o.form);//获取表单对象
    form.target = id; //这句话很关键，是连接表单和iframe的桥梁
    form.method = 'POST';//文件上传必须设置成post请求
    form.enctype = form.encoding = 'multipart/form-data';//这也是文件上传必须设置的
    if(url){
        form.action = url;//表单提交的地址
    }
```

第三步：这一步当然是关键，他衔接了第一步的 Web 组件和第二步的界面。从界面获取数据并提交到服务器，这个操作由 Ext.Ajax.request()方法完成。

```
Ext.get("btn").on("click", function(){
    Ext.Ajax.request({
        url: "../FileUpServlet",
        isUpload: true,
        form: "upform",
        success: function(response, config){
            Ext.MessageBox.alert("fileup", "文件上传成功");
        }
    });
});
```

isUpload 属性表明这是一个文件上传的请求，form 是表单标签的 id，请求成功调用 success 方法。

四、你来自远方

总体来说，Extjs 对 Ajax 的封装已出神入化、无可挑剔，特别是对文件上传的支持，令 Extjs 大放异彩。不过，还有更牛的，如果将 ajax 和某些组件关联，连 Ext.Ajax.request 方法都不需要调用了，我们甚至感觉不到 ajax 正在工作。

ComboBox 的选项数据可以来自于本地（local），同样也可以从服务器获取数据并填充至选项中，这一切当然归功于 ajax，设计得如此简洁和巧妙，无任何框架能出 Extjs 其右。

问题又回到第十章，我们使用 Store 来创建一个类似于数据表的结构，DataProxy 负责取到数据，DataReader 负责定义记录结构。因为要远程获取数据，所以应该使用 HttpProxy，假设我们从服务器返回的是 json 格式的数据，用于保存城市信息，因此必须使用 JsonReader 来解析，于是就有了下面的代码：

```
var proxy = new Ext.data.HttpProxy({url: "../CityJsonServlet"});

var City = Ext.data.Record.create([
    {name: "cid", type: "int", mapping: "cid"},
    {name: "cname", type: "string", mapping: "cname"}
]);
var reader = new Ext.data.JsonReader({}, City);

var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader
});
```

```
// CityJsonServlet.java
public class CityJsonServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        String json = "[{cid: 1, cname: '长沙市'},{cid: 2, cname: '株洲市'},{cid: 3, cname: '湘潭市'},{cid: 4, cname: '邵阳市'}, {cid: 5, cname: '娄底市'}]";
        out.println(json);
        out.flush();
        out.close();
    }
}
```

在 City 结构中，mapping 不再是索引，而是 json 对象的属性名，在逻辑列名与 json 之间建立映射。最后，将数据与结构整合到 Store。

剩下的工作就是将数据变成 ComboBox 的数据项了。大家可以比较一下下面的代码与前面学习过的代码之间的区别：

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: "请选择城市",
    mode: "remote",
    triggerAction: "all",
    displayField: "cname",
    valueField: "cid",
    renderTo: Ext.getBody()
});
```

这些配置选项是那么熟悉。现在，让我们擦亮眼睛找出前后的差别：mode 的值由原来的 local 变成了 remote；没有 autoLoad 选项， ComboBox 自动加载数据。太强了，聪明得让我吃惊。

不管数据来自哪里，ComboBox 的定义都没有太大的出入，最大的不同就是 mode 配置项的区别，如果数据来自内存，设置成 local，如果数据来自远程服务器，设置成 remote，记住啊，默认是 remote。

如果希望关闭 ComboBox 的编辑功能，设置 readOnly: true 即可。

五、小结

一不小心把 Extjs 的 ajax 讲了个遍，这真是一个体贴的功能，其实 Extjs 对 xml 格式的支持也很强大的，不过人的惰性限制了他的使用。WebQQ 和开心网大量使用了 json 格式，相信这种轻量级的数据格式会越来越得到程序员的喜爱。

我对 Extjs 的 ajax 文件上传功能非常敬仰，Extjs 总是恰到好处地帮我们想好所有的事情，从源代码中能感受到 Extjs 的无穷魅力。

重新设计的 ComboBox 太强大了，完全颠覆了下拉列表框最原始的定义，我们还真没想到，这东东居然支持分页。下一章我将会和大家一起分享。

第十二章：分页与 ComboBox

一、关于分页

分页是查询最常见的操作之一，比较普遍的作法是要多少数据从数据库中找多少数据。Extjs 对分页的支持十分强大，为了防止刷新，都采用 ajax 实现，服务器负责将数据从数据库中取出，传到客户端，Extjs 再根据服务器返回的信息自动分页。

前面说过，Extjs 支持三种格式的数据：数组、json 和 xml，后两者支持分页查询，而使用 json 是大家喜欢和熟悉的作法。

我们使用 ComboBox 来演示分页的例子，可以说是前人古人后无来者，哈，其实都一样，后面将使用 GridPanel 分页显示数据大同小异。这一章懂了，我就不相信后面的你不懂——简直是不可能的事。

Extjs 设计的分页导航条很漂亮，功能一应俱全，更有意思的是，如果您的要求超高，还可以在分页导航条上放置自己的控件，这是后话。

二、从 Servlet 获取当前页数据

数据终究是从服务器获取的，至于使用 Servlet，或者是 jsp 甚至 Action 都无关紧要。而我也并不想和 JDBC 攀亲带故，无意从数据库取数据，我们力求保证代码的清晰和简洁，在一个问题没有弄明白之前，再扯出另一个问题是不明智的行为。

但必须明确取的是什么数据，这是基本。为了方便，我们取出部门信息，包含部门 ID 和部门名称，ID 作实际值，名称作显示值。为了区别不同的部门，在名称后面添加序号。这是一个好主意。

为了能够让 ComboBox 认识 Servlet 返回的数据，必须依照 ComboBox 规定的格式来定义 json 对象，就相当于二者之间达成的协议，只有遵循共同的标准大家才能互通互话。返回数据的格式应该像下面这样定义：

```
{totalProperty:17, root:[{did: 0, dname: '部门 0'},{did: 1, dname: '部门 1'},{did: 2, dname: '部门 2'},{did: 3, dname: '部门 3'},{did: 4, dname: '部门 4'}]}
```

totalProperty 是分页中必须要用的总记录数，如果提供一个页大小就好了，这样就能通过“总记录数%页大小 == 0？总记录数 / 页大小 : 总记录数 / 页大小 + 1”得到总页数，如此一来，天兵天将全齐了，分页也就应声而出。

root 是一个 json 对象的数组，包含了详细的数据。root 中元素的个数由页大小决定。如

果写简单点就是这样啦：

```
{totalProperty:17, root:[{ },{ },{ },{ },{ }]}
```

另外，既然是分页，提供分页参数是不可避免的，ComboBox 会自动提供两个参数：start 和 limit，start 是从第几条开始，limit 是取出多少条。如果页大小是 5,第一页的参数值应该是 start = 0，limit = 5；第二页的参数值应该是 start = 5, limit = 5；第三页的参数值应该是 start = 10, limit = 5……依此类推。事实上，这只是我帮助大家理解才讲得如此啰嗦，ComboBox 已经完成了绝大部分工作，我们只要指定页大小（pageSize）就可以了。但是，在 Servlet 中，我们必须得到 start 和 limit 的值，不然怎么分页呢？

```
public class ComboBoxServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        //起始索引
        int start = Integer.parseInt(request.getParameter("start"));
        //页大小
        int limit = Integer.parseInt(request.getParameter("limit"));
        System.out.println(start + "," + limit);
        //总记录条数，如果是数据库则要通过 count 计算出来
        int totalProperty = 17;

        String fmt = "{did: %d, dname: '%s'}";
        StringBuffer s = new StringBuffer("{totalProperty:");
        s.append(totalProperty).append(", root:[");
        int end = start + limit;

        //因为不是查找数据库，所以需要多加一个判断
        if(end > totalProperty) end = totalProperty;//如果是数据库，本行要删除

        for(int i = start; i < end; i ++){
            s.append(String.format(fmt, i, "部门" + i));
            if(i < end - 1){
                s.append(",");//各 json 对象用“,”隔开，最后一个不要
            }
        }
        s.append("]}");
        out.println(s.toString());
        out.flush();
        out.close();
    }
```

如果前面的文字看起来很迷糊，建议你配合上面的代码阅读，思路会瞬间清晰。

三、创建 ComboBox

如果再重复讲解 ComboBox 的用法，相信您会睡着去吧，为了保持您一如既往的激情，我决定不做傻事。我把代码贴出来，不同点作个标记，再详细向您解释。

```
Ext.onReady(function(){
    var proxy = new Ext.data.HttpProxy({url: "../ComboBoxServlet"});

    var City = Ext.data.Record.create([
        {name: "did", type: "int", mapping: "did"},
        {name: "dname", type: "string", mapping: "dname"}
    ]);
    var reader = new Ext.data.JsonReader({
            totalProperty: "totalProperty", //总记录数
            root: "root" //所有的数据（json 对象数组）
        }, City);

    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader
    });

    var combo = new Ext.form.ComboBox({
        store: store,
        emptyText: "请选择部门",
        mode: "remote",
        pageSize: 5,
        triggerAction: "all",
        displayField: "dname",
        valueField: "did",
        renderTo: Ext.getBody(),
        readOnly: true,
        listWidth: 300
    });
})
```

粗斜体部分是 ComboBox 分页功能额外加的，构造 JsonReader 对象的第一个参数以前都为空，现在我放了一个 json 对象，明眼人一看就知道了，totalProperty 和 root 两个属性与 Servlet 返回的 json 对象的两个属性一一对应，还记得下面的格式吗？

```
{totalProperty:17, root:[{did: 0, dname: '部门 0'},{did: 1, dname: '部门 1'},{did: 2, dname: '
```

```
部门 2'},{did: 3, dname: '部门 3'},{did: 4, dname: '部门 4'}]}
```

记住，是属性名，不是属性值。ComboBox 会自动根据属性名将属性值读出来。也可以这样返回：

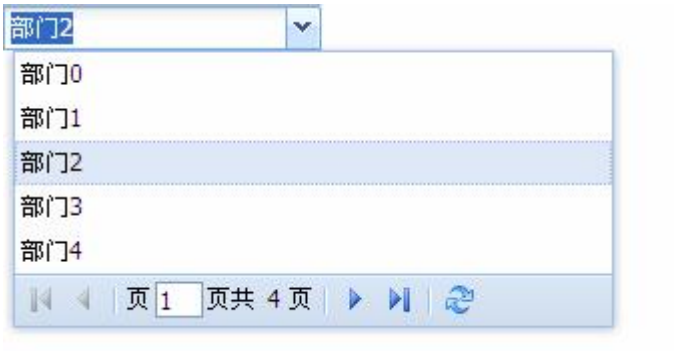
```
{count:17, data:[{did: 0, dname: '部门 0'},{did: 1, dname: '部门 1'},{did: 2, dname: '部门 2'},{did: 3, dname: '部门 3'},{did: 4, dname: '部门 4'}]}
```

totalProperty 和 root 分别变成了 count 和 data，JsonReader 应该这样构造：

```
var reader = new Ext.data.JsonReader({
        totalProperty: "count", //总记录数
        root: "data" //所有的数据（json 对象数组）
    }, City);
```

我都觉得自己快变成老太婆了，哪有这么啰嗦的人哦。

pageSize 就不用说啦，一看名字就知道是页大小，ComboBox 列表一页显示多少个选项全由他说了算，而 listWidth 与分页无关，用来控制列表的宽度，效果如下图：



四、小结

ComboBox 到此告一段落，别小看第十章至第十二章啊，这里面有很多东西在以后的学习中依旧有用，特别在显示表格数据时，简直一个模板刻出来的。所以，这样的热身绝对不是耍花招，是真枪实战，如果这三章没学懂，我建议您暂时停下来，看看路边的风景，学懂后再继续前行。

第十三章：面板（Panel）

一、漂亮的窗格从这里开始

如果你需要一款优雅清淡的面板，Extjs 绝对是不二选择。Ext.Panel 类定义的面板令我们对美的追求达到极限，再苛刻的用户都没有理由表示他的否定。Panel 不仅拥有美丽的外表，其功能也绝不忽悠。

我很奇怪为什么在 Extjs 自带的例子中关于 Panel 的演示仅廖廖数笔，只讲了该组件的最简单用法，更多的高级功能绝笔未提，这似乎不是 Extjs 的风格，或者这从另一个侧面说明 Panel 的使用面并不广。实际上，他的子类用得更加频繁，如：TreePanel、TabPanel、FormPanel、GridPanel、EditGridPanel 等，都是非常重要的子类。

确切地说，面板更应该是一个容器，在面板中，我们可以放置任何组件和文字，而面板本身也可以放到其他面板中。

二、Ext.Panel 类

Panel 的结构比我们想象中要复杂一点，一个完整的 Panel 包括标题栏、顶部工具栏、底部工具栏、footer 和面板主区域组成。footer 没办法用中文描述，一般放置诸如“确定”或“取消”之类的按钮。

另外，从图上可以看到，标题栏右侧还可以放一排小按钮，实现一些体贴的功能，如关闭、刷新、最小化、最大化、设置等。

我们先来看一个最简单的面板代码：

```
var panel2 = new Ext.Panel({
    title: "最简单的 Panel",
    width: 400,
    titleCollapse: true,
    collapsible: true,
    html: "“误炸”是美国的最终解释和态度，只有在遇到一些中美军事对峙事件时，“炸馆事件”才会被偶尔提及。美国总统中国顾问团的一位成员说，10 年后中国已经崛起，中美关系也稳定并朝着良好势头发展，当年“误炸”已成历史一瞬。",
    renderTo: "d"
});
```

title 决定 Panel 是否有标题栏，属性值用于指定标题栏的名字，collapsible 值为 true 表示 Panel 可收缩，此时标题栏右侧会出现一个方向按钮，titleCollapse 使标题栏也具备收缩能力，html 指定正文区域的内容，在这里是一段静态文字，如果想将一个页面的内容显示出来，autoLoad: {url: "inner.html"}可以达到目的，当 html 与 autoLoad 属性并存，autoLoad 优先显示。

Panel 包含顶部和底部两个工具栏，属性分别为 tbar（Top bar）和 bbar（Bottom bar），他们都是 Ext.Toolbar 类型的对象，一个典型的 Ext.Toolbar 这样定义：

```
var toolBar1 = new Ext.Toolbar({
    buttons: [{
        text: "新建",
        handler: function(){
            Ext.MessageBox.alert("新建", "新建操作演示");
        }
    },{
        xtype: "tbseparator"
    },{
        text: "打开"
    },{
        text: "保存"
    }]
});
```

在 buttons 属性中，定义了一个数组，数组中的元素为 json 对象，通常情况下代表一个按钮，text 是按钮上的文字，handler 是单击按钮时触发的回调函数。不过，除了按钮之外还可以是其他类型的元素，如：tbseparator（简写为“-”）、tbspacer（简写为“ ”）、tbfill（简写为“->”）、tbsplit（带菜单的按钮）等。

在上面的代码中，定义了一个分隔线，这是通过 tbseparator 实现的，分隔线是为了能够更好的区别按钮类别而定义的竖线，也可以使用“-”代替，像下面这样：

```
var toolBar2 = new Ext.Toolbar({
```

```
    buttons: [{
        text: "上一步"
    },{
        text: "下一步"
    },"-",{
        text: "退出"
    }]
});
```

通过 icon 或 iconCls 可以在按钮前面放置图标，icon 的值为图片名称，iconCls 的值是一个类选择器，必须通过 css 声明。因为按钮图片通过背景显示，所以在文字之前一定记得留出图片空隙，cls: "x-btn-text-icon"帮我们完成了这个工作，不然我们会发现图片和文字叠加在一起了。我将在“上一步”和“下一步”按钮上放置图标，修改后代码如下：

```
var toolBar2 = new Ext.Toolbar({
    buttons: [{
        text: "上一步",
        icon: "../extjs/resources/images/default/dd/drop-add.gif",
        cls: "x-btn-text-icon"
    },{
        text: "下一步",
        icon: "../extjs/resources/images/default/dd/drop-yes.gif",
        cls: "x-btn-text-icon"
    },"-",{
        text: "退出"
    }]
});
```

工具栏创建好后，接下来放到 Panel 中，分别设置 tbar 和 bbar 属性即可：

```
var panel2 = new Ext.Panel({
    title: "最简单的 Panel",
    width: 400,
    titleCollapse: true,
    collapsible: true,
    html: "“误炸”是美国的最终解释和态度，只有在遇到一些中美军事对峙事件时，“炸馆事件”才会被偶尔提及。美国总统中国顾问团的一位成员说，10 年后中国已经崛起，中美关系也稳定并朝着良好势头发展，当年“误炸”已成历史一瞬。",
    autoLoad: {url: "inner.html"},
    renderTo: "d",

    tbar: toolBar1,
    bbar: toolBar2
});
```

标题栏的右侧，还可以放置一排小按钮，如果设置了 collapsible 属性，我们可以看到一个向上的箭头，事实上，可以放置更多的按钮，这些按钮通常是一个小图标，没有文字。在 Ext.Panel 中通过 tools 来定义：

```
tools: [{
    id: "refresh",
    handler: function(event, toolEl, p){
        p.getUpdater().update({url: "inner.html", scripts: true});
    }
}]
```

tools 中的按钮都已预先定义，通过 id 来区别，比如 id 为 refresh 时，表示这是刷新按钮，但是并没有定义单击事件，具体的响应过程还需要通过 handler 来定义。在上面的例子中，handler 函数有三个参数，event 是 Ext.EventObject 对象，toolEl 指当前按钮对象，p 则指当前面板对象，通过 p.getUpdater().update({url: "inner.html", scripts: true});可以更新面板正文区域的内容，注意内容是从 inner.html 文件中加载的，scripts 为 true 表示执行该文件中的脚本，否则脚本不会执行。

Extjs 赋予了 tools 更多的内容，使用时将 id 设置成下面列表的任意一个值即可。我们列个清单一起展示：

```
toggle
close
minimize
maximize
restore
gear
```

```
pin
unpin
right
left
up
down
refresh
minus
plus
help
search
save
print
```

在下面的示例中，我会在标题栏上放置五个小按钮，分别完成“刷新”、“收缩面板”、“打开面板”、“帮助”和“关闭面板”的功能：

```
var panel2 = new Ext.Panel({
    title: "最简单的 Panel",
    width: 400,
    titleCollapse: true,
    collapsible: true,
    html: "“误炸”是美国的最终解释和态度，只有在遇到一些中美军事对峙事件时，“炸馆事件”才会被偶尔提及。美国总统中国顾问团的一位成员说，10 年后中国已经崛起，中美关系也稳定并朝着良好势头发展，当年“误炸”已成历史一瞬。",
    autoLoad: {url: "inner.html"},
    renderTo: "d",

    tbar: toolBar1,
    bbar: toolBar2,

    tools: [{
        id: "refresh",
        handler: function(event, toolEl, p){
            p.getUpdater().update({url: "inner.html", scripts: true});
        }
    },{
        id: "up",
        handler: function(event, toolEl, p){
            p.collapse(true);
        }
    },{
        id: "down",
        handler: function(event, toolEl, p){
```

```
                p.expand(true);
            }
        },{
            id: "help",
            handler: function(event, toolEl, p){
                Ext.Msg.alert("关于", "本软件系椰子林软件工作室出品，盗版必究！");
            }
        },{
            id: "close",
            handler: function(event, toolEl, p){
                p.hide();
            }
        }]
});
```

代码中涉及到 Panel 的一些方法，collapse()方法可以收缩面板，expand()方法可以展开面板，这两个方法都带一个 boolean 类型的参数，为 true 时面板以动画方式徐徐展开或收缩。hide()自然是将面板隐藏。运行后效果如图：

三、小结

从 Ext.Panel 的源代码来看，他的实现并不复杂。但是这个类在 Extjs 中有不可替代的重要性，树形面板、表单面板、选项卡面板、网格面板等组件都是在该类的基础上扩展而成的。

大家应该养成阅读源代码的习惯，因为我发现 Extjs 的 API 写得并不详细，很多配置、属性和方法都找不到，而在源代码中，却有详细的注释和讲解。直白地说，看源代码带来我好处远远大于 API 文档。同时，尽量保持阅读英文的习惯，一个一个单词去读，根据上下

文环境大概了解意思就够了。

标记一下，Ext.Panel 的 frame 配置为 true 时，工具栏会出现错位并位置不准确的问题，我不知道是个别现象还是普通现象，大家可以自己试一下，并反馈给我，反馈结果可以发到我的邮箱，或者在学校内部网上留言。

第十四章：Panel 的子类——Window 窗口

一、概述

Window 是 Panel 的子类，Ext.Window = Ext.extend(Ext.Panel, {})，从源代码中可以看出，Window 继承自 Panel，意味着 Window 具备 Panel 的所有特征，并且有他自己的个性。

从外观上来看，Window 对象更像一个 Window 操作系统中的窗口，有标题栏、关闭按钮、默认显示在页面最中间、能够拖动、改变窗口大小、自动激活选中窗口，不仅如此，Window 还有他自己的特定行为，必须使用 show()方法才会显示出来，render 方法和 renderTo 属性对他来说已失去魅力。

Window 有两种类型：模式窗口和非模式窗口，默认为后者。模式窗口在窗口没有关闭之前无法对其他组件作任何操作，等同于 Ext.Message.alert()的效果。

默认情况下，Window 接受 ESC 按键，按下 ESC 键后，窗口会自动关闭。

二、Ext.Window 类

因为 Ext.Window 继承自 Ext.Panel，所以很多用法都非常类似，一个典型的 Ext.Window 对象包含标题栏、关闭按钮，这也是默认行为。当然，习惯上，我会为为窗口指定一个初始宽度和高度。

```
Ext.onReady(function(){
    new Ext.Window({
        title: "窗口",
        width: 400,
        height: 300
    }).show(Ext.getBody());
})
```

Window 要通过 show()方法才能显示，参数为动画参照物，窗口会从参照物上飞出。如果不指定参数，则没有动画效果。

Window 是一个容器，可以通过下面三种方式往 Window 正文区域放置内容：

1、直接配置 html 属性

```
Ext.onReady(function(){
    new Ext.Window({
        title: "窗口",
        width: 400,
```

```
        height: 300,
        html: "今天是双休,可是我不能休息",
        bodyStyle: "padding: 5px"
    }).show(Ext.getBody());
})
```

bodyStyle 配置选项是 Window 正文区域的样式，在上面的示例中，填充间距被设置成 5 个像素，防止内容和窗口边框挨得太近。

2、将另一个文件中的内容加载到 Window 中

```
Ext.onReady(function(){
    new Ext.Window({
        title: "窗口",
        width: 400,
        height: 300,
        bodyStyle: "padding: 5px",
        autoLoad: {url: "content.html", scripts: true}
    }).show(Ext.getBody());
})
```

3、使用 items 属性往 Window 中添加组件

```
Ext.onReady(function(){
    new Ext.Window({
        title: "窗口",
        width: 400,
        height: 300,
        bodyStyle: "padding: 5px",
        items: [
            new Ext.Button({text: "按钮哦"}),
            new Ext.DatePicker({})
        ]
    }).show(Ext.getBody());
})
```

所有容器都有 items 配置选项，该选项就是我们要放到容器中的组件集合，默认情况下，使用 form 布局对控件进行排列（即纵向排列）。在上面的示例中，我们放了两个控件，分别为按钮和日期选择器。

在没有显式配置的情况下，Window 没有最大化和最小化按钮，这两个按钮可以通过 maximizable 和 minimizable 来指定，遗憾的是，Extjs 只实现了最大化功能，最小化的功能没有实现，我们只能根据自己的需要来实现最小化窗口。

庆幸的是，关闭按钮已经帮我们做了，而且有两种选择：关闭后如果只是隐藏，将 closeAction 配置选项设置为“hide”，这样做不会释放内存；如果要从 DOM 树中彻底删除，closeAction 必须为“close”，close 是默认设置。

和 Ext.Panel 一样，可以在 Window 中定义工具栏，方法与前者相差无几。

三、实现 Window 的最小化功能

Window没有实现最小化，程序员可以根据客户和业务需求设计更加实用的最小化效果。这一小节，我们完成这个任务。

我打算模拟 Windows 操作系统中的桌面，这似乎是一个巨大的挑战。在桌面下方（或上方）有一个任务栏，窗口最小化之后会在任务栏生成一个按钮，点击按钮后窗口随即还原。

为了降低实现难度，我将任务栏放在页面上方，坦白说这不是真正的任务栏，只是工具栏而已。工具栏上有一个固定不变的按钮，用来创建新窗口，新创建的窗口从该按钮飞出，窗口创建后在工具栏上动态生成一个对应的按钮，这样，每次新创建的窗口都有唯一的按钮与之对应。不知道你有没有明白我说的话，再简单点解释就是桌面任务栏的模拟。

工具栏这样创建：

```
var toolBar = new Ext.Toolbar({
    height: 25,
    renderTo: Ext.getBody(),
    items: [
        newWin
    ]
});
```

工具栏上放置了一个名为 newWin 的按钮，该按钮像下面这样定义：

```
var newWin = new Ext.Button({
    text: "新建窗口",
    pressed: true,
});
```

pressed 选项表示按钮处于按压状态，我感觉这种效果似乎更像按钮一些，当然，您可以设置一个图标使之更加漂亮。

点击“新建窗口”按钮后，通过 handler 回调函数来创建一个新的窗口，实际上，我们还会创建一个按钮，该按钮与新建窗口对应，并且放到工具栏中。

```
var newWin = new Ext.Button({
```

```
    text: "新建窗口",
    pressed: true,

    handler: function(){
        var win = new Ext.Window({
            title: "窗口" + (i ++),
            width: 400,
            height: 300,
            minimizable: true,
            listeners: {
                //最小化窗口事件
                minimize: function(window){
                    window.hide(win.button.id);
                },

                close: function(window){
                    var delObj = document.getElementById(window.button.id);
                    delObj.parentNode.parentNode.removeChild(delObj.parentNode);
                }
            }
        });
        win.show(newWin.id);
        win.getUpdater().update({url: "content.html"});

        win.button = new Ext.Button({
            win: win,
            id: "id" + i ,
            text: win.title,
            pressed: true,
            handler: function(btn){
                btn.win.show(btn.id);
            }
        });
        toolBar.addButton(win.button);
    }
});
```

相对而言，这段代码长了许多，我将代码所做的工作简单描述一下：单击“新建窗口”后，创建一个新的 Window 对象，标题以步长为 1 依次累加，长和宽分别为 400 和 300 像素，显示“最小化”按钮，配置完毕后，调用 show()方法显示，显示的动画参照物为“新建窗口”按钮，也就是说，所有新的窗口都从一个位置飞出。然后，通过更新器对象加载 content.html 中的内容到窗口中。接下来的任务就是创建一个与窗口对应的按钮，该按钮的标题和窗口标题一致。要注意的是，在程序中出现了按钮和窗口相互引用的情况，这样做的目的是为了在操作窗口时能够访问按钮，操作按钮时也同样能够访问窗口。按钮引用窗口通过配置 win:win

实现，窗口引用按钮则通过 win.button 来实现。

那么，我们究竟是如何最小化窗口的呢？大家可能发现了一个名为 listeners 的配置，这个配置很有用，能够定义一系列的组件事件，这样事件的管理更加集中。在代码中，minimize 事件在点击“最小化”按钮时触发，我们的实现非常简单：window.hide(win.button.id);，该语句以 win.button.id 作为参照物隐藏窗口。单击对应的按钮后，隐藏的窗口再次显示出来。

最后要关注的是窗口关闭后的一个连带责任的问题。我们都知道，每个窗口都有一个按钮与之对应。也就是说，窗口关闭了，工具栏上的按钮也必须删除。我在 Toolbar 的源代码中找了半天，并没有一个合适的方法来实现删除按钮的功能，所以，只好采用最原始的办法，通过操作 DOM 实现。

```
<table cellspacing="0">
  <tbody>
    <tr>
      <td id="ext-gen7">
      <td id="ext-gen190">
        <table id="id4" class="x-btn-wrap x-btn x-btn-
          pressed" cellspacing="0" cellpadding="0" border="0" style="width: auto;">
        </td>
      <td id="ext-gen281">
        <table id="id5" class="x-btn-wrap x-btn x-btn-
          pressed" cellspacing="0" cellpadding="0" border="0" style="width: auto;">
        </td>
    </tr>
  </tbody>
</table>
```

为了配合讲解，我截了一张图，上面的图是工具栏在浏览器中的静态代码，工具栏是通过使用一个一行多列的 table 实现的，一个按钮占用一个单元格 td，所以，按钮删除后，该按钮所在的单元格也应该删除。

问题在于，我在代码中只为按钮设置了 ID，所以，按钮我能控制。图中的“id4”和“id5”就是生成的两个按钮，注意按钮也是通过表格做出来的。删除一个节点的规则是：父节点.removeNode(子节点)。比如，我们能找到 id4 按钮，该按钮位于 ext-gen190 单元格中，所以，删除该按钮时必须删除 ext-gen190 单元格，现在问题转换成了先找到 ext-gen190 的父节点，再调用 removeNode()方法删除 ext-gen190 单元格。

哈，您一定晕了，不过这是没办法的事，您可以循环往复多看几遍。顺便提醒一下，写脚本真的离不开 Firefox 浏览器，更离不开 Firebug 插件。如果您死守 IE 不放，我只能说您是死脑筋。在很久很久以前，Firefox 便成了我桌面浏览器的首选，我忠心希望您能改变以往的习惯。让我们一起 Firefox 吧！

继续前面的话题，当用户单击关闭按钮时，马上会触发 close 事件，在该事件中，我们有机会删除工具栏上对应的按钮。代码如下：

```
var delObj = document.getElementById(window.button.id);
delObj.parentNode.parentNode.removeChild(delObj.parentNode);
```

第一行代码根据 id 找到按钮，第二行代码则删除按钮。

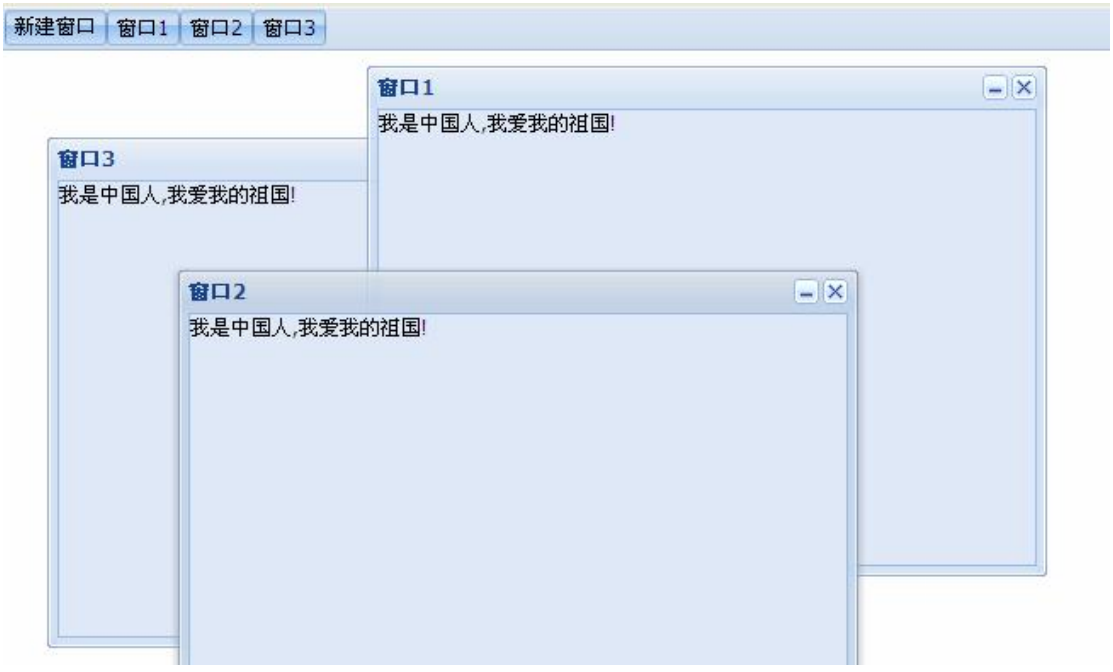
delObj(id4).parentNode(ext-gen190).parentNode(tr).removeChild(delObj.parentNode); 括 号内的文字是我作的说明，代表某一个标签。

完整的代码如下：

```
var i = 1;

Ext.onReady(function(){
    var newWin = new Ext.Button({
        text: "新建窗口",
        pressed: true,

        handler: function(){
            var win = new Ext.Window({
                title: "窗口" + (i ++),
                width: 400,
                height: 300,
                minimizable: true,
                listeners: {
                    //最小化窗口事件
                    minimize: function(window){
                        window.hide(win.button.id);
                        window.minimizable = true;
                    },

                    close: function(window){
                        var delObj = document.getElementById(window.button.id);

    delObj.parentNode.parentNode.removeChild(delObj.parentNode);
                    }
                }
            });
            win.show(newWin.id);
            win.getUpdater().update({url: "content.html"});

            win.button = new Ext.Button({
                win: win,
                id: "id" + i ,
                text: win.title,
                pressed: true,
                handler: function(btn){
                    if(btn.win.minimizable){
                        btn.win.show(btn.id);
```

```
                        btn.win.minimizable = false;
                    }else{
                        btn.win.hide(btn.id);
                        btn.win.minimizable = true;
                    }

                }
            });
            toolBar.addButton(win.button);
        }
    });

    var toolBar = new Ext.Toolbar({
        height: 25,
        renderTo: Ext.getBody(),
        items: [
            newWin
        ]
    });
})
```

动画无法截图，下面是打开多个窗口后的效果：



四、小结

本章的第三节，是本章的难点，可能有些同学甚至无法理解。我的建议是结合图片一起

看，多看几遍，也许会豁然开朗。

万一还是看不懂，别浪费时间了，进入下一章吧……

第十五章：Panel 的子类——FormPanel

一、无处不在的表单

作为一个 Web 开发程序员，表单再熟悉不过。我们经常会通过表单提交数据到服务器进行处理，在传统的 Web 开发中，使用 HTML 标记实现，使用一个<form>标签包裹若干个表单域标签，借助提交按钮完成数据的提交操作。在提交之前，可以触发 onsubmit 事件，在该事件中有机会对提交的数据进行验证，如果验证失败，则阻止表单提交。

一个多么简单的过程啊!

但是，如果您认为在 Extjs 中会一样简单，那就错了。在熟悉和掌握之前，可能真的会被 Extjs 表单打倒。我们必须学习他一系列的表单控件、验证方法，以及看起来简单却极易出错的布局。

思路总是在纷繁复杂中迷离，最后又在思考中简化。

二、Ext.form.FormPanel 类

Ext.form.FormPanel 继承自 Ext.Panel，具备面板的特征，但在布局上有他自己的优势。总体来说，FormPanel 为表单提供了一个完整的界面，但不包含表单操作，如提交和重置，这些操作由 Ext.form.BasicForm 类完成。

我们得出一个结论：Ext.form.FormPanel 类为表单提供一个统一的界面，而 Ext.form.BasicForm 类为表单提供操作，如提交、重置、检查脏数据、数据合法性验证等等。

Ext.form.FormPanel 和 Ext.form.BasicForm 的关系如下：

```
Ext.form.BasicForm = Ext.form.FormPanel.getForm()
```

不管怎么说，表单是由一系列表单域组成的，Extjs 对表单域也同样做了封装，这些表单域以某种特定布局存在于 FormPanel 中，FormPanel 有一个叫 items 的配置，这是一个 MixedCollection 类型的对象，也可以认为是一个集合，而表单域就是该集合中的一个个元素。

Ext.form.FormPanel 规定至少要有一个表单域元素，事实上没有表单域的表单是不存在的，下面是一个最简单的表单：

```
Ext.onReady(function(){
    var f = new Ext.form.FormPanel({
        title: "欢迎登陆",//表单面板的标题
        width: 300,//面板宽度
        height: 130,//面板高度
```

```
        bodyStyle: "padding: 6px",//正文区域的样式
        labelAlign: "right",//统一的提示信息对齐方式，此处为右对齐
        frame: true,//此处为 true 可以得到更好的效果
        items: [
            new Ext.form.TextField({
                name: "userName",//用于访问的 name
                allowBlank: false, //不允许为空
                fieldLabel: "用户名",//表单域提示信息
            }),{
                name: "password",
                xtype: "textfield",//表示表单域的类型，此处即为 Ext.form.TextField
                inputType: "password",//输入框类型，有三种类型：文本框、密码框、
                                      //文件域
                fieldLabel: "密码",//提示信息
                allowBlank: false //不允许为空
            }
        ],
        buttons:[{
            text: "确定",
            handler: function(){
                //得到 name 为 userName 的表单域控件的值
                var userName = f.getForm().findField("userName").getValue();
                var password = f.getForm().findField("password").getValue();
                Ext.Msg.alert("", "用户名：" + userName + "<br>密码：" + password);
            }
        },{
            text: "重置",
            handler: function(){
                f.getForm().reset();//重置表单面板中所有表单组件的值
            }
        }]
    });

    f.render("a");
})
```

这是大家非常熟悉的用户登陆。一个表单的基础配置应该包含表单标题、宽度、高度，分别使用 titile、width、height 选项进行配置。bodyStyle 为正文区域样式。frame 为 true 可以得到一个更加漂亮的效果。

我们最应该关注的可能是表单域了，从外观来说，一个表单域包含提示信息和输入/选择控件，这两部分被封装在表单域控件中，如文本框由 Ext.form.TextField 类来定义，fieldLabel 选项指定提示信息，hideLabel 可以将提示信息隐藏，labelAlign 指定提示信息的对齐方式。另外，密码框是文本框的变种，设置 inputType: "password"后文本框中的内容将以密文显示。

allowBlank 选项表示是否允许为空，如果为 false，表示为必填项。

在示例中，我特意使用两种不同的风格定义用户名和密码两个文本框，用户名使用 new 关键字创建；密码框直接定义成了 json 对象，但多定义了一个选项 xtype，在第八章具体介绍了每一个组件的 xtype 值，该值是组件的另一个名字的字符串表示，如；xtype=”textfield” 表示文本框，xtype=”form”表示表单。xtype 的值通常由类似于 Ext.reg('form', Ext.FormPanel); 的代码来注册，并统一使用小写表示。

在 FormPanel 中，需要两个按钮——提交和重置。按钮定义在 buttons 选项中，是一个数组类型的对象。按钮的定义在前面说了很多，在此不再赘述。我们要强调的是当单击“提交”按钮时，如何得到表单域的值。

要得到表单域的值，自然必须先得到表单域对象，FormPanel 类定义了一个 getFom()方法，返回 BasicForm 对象，该对象是表单面板关联的表单对象。BasicForm 类有一个方法叫 findField（），该方法能根据表单域的 id 或 name 得到表单域对象，然后通过表单域对象的 getValue()方法得到用户输入的值。代码如下：

```
var userName = f.getForm().findField("userName").getValue();
var password = f.getForm().findField("password").getValue();
```

我们可以看看 findField()方法的源码：

```
findField : function(id){
    var field = this.items.get(id);
    if(!field){
        this.items.each(function(f){
            if(f.isFormField && (f.dataIndex == id || f.id == id || f.getName() == id)){
                field = f;
                return false;
            }
        });
    }
    return field || null;
},
```

最开始 findField()方法根据 id 来查找表单域对象，如果没找到，再根据 name 查找。所以，我们定义 name 就行了，name 还有另外一个功能，服务器在接收表单域的数据时，必须提供该属性。

重置功能的实现更加简单，BasicForm 类定义了一个名为 reset()的方法，该方法能将表单中的所有表单域重置。我们也可以看看他的源代码：

```
reset : function(){
    this.items.each(function(f){
```

```
            f.reset();
        });
        return this;
},
```

从代码中可以看出，reset()方法循环遍历了表单中的所有表单域，然后调用表单域的reset()方法。BasicForm 类的 items 属性是表单域的集合，通过 each 方法对集合中的元素进行遍历，遍历到当前元素时，通过回调函数对当前元素进行处理，而回调函数中的参数就是当前元素。这个是 Extjs 的特定循环语法。

下面的代码能完成一样的功能：

```
{
    text: "重置",
    handler: function(){
        var fields = f.getForm().items.items;
        for(var i = 0; i < fields.length; i ++){
            fields[i].reset();
        }
    }
}
```

在代码 var fields = f.getForm().items.items 中，您可能会疑惑为什么会有两个 items，其实我们知道 f.getForm().items 是 MixedCollection 类型，MixedCollection 类型又关联了一个 Array 类型，而表单域是放在该 Array 类型中的，所以 items 必须连续引用两次。

下面关于 MixedCollection 的源代码也许对理解有帮助：

```
Ext.util.MixedCollection = function(allowFunctions, keyFn){
    this.items = [];
    this.map = {};
    this.keys = [];
    this.length = 0;
}
```

FormPanel 中定义了一个名为 defaultType 的属性，该属性用于在没有指定 xtype 时默认创建的组件类型。下面的代码简单地说明了他的用法：

```
var f = new Ext.form.FormPanel({
    //数据传送到服务器的配置
url: "../FormServlet",
method: "post",
baseParams: {extra: "中华人民共和国", id: 100},
title: "欢迎登陆",
```

```
width: 300,
autoHeight: true,
bodyStyle: "padding: 6px",
labelAlign: "right",
defaultType: "textfield",//默认组件类型
frame: true,
items: [
    new Ext.form.TextField({
        name: "userName",
        allowBlank: false, //不允许为空
        fieldLabel: "用户名"
    }),{
        name: "password",
        inputType: "password",
        fieldLabel: "密码",
        allowBlank: false //不允许为空
    },{
        fieldLabel: "验证码"
    }
],
……
……
```

指定了 defaultType 后，密码和验证码输入框在没有指定 xtype 的情况下，自动创建该属性定义的组件类型 textfield。

三、提交表单至服务器

表单数据最终还是要提交到服务器，在 Extjs 中，通常使用 ajax 实现。BasicForm 类定义了一个名为 submit()的方法，该方法用于提交表单。我们在提交按钮的 handler 中定义如下代码：

```
f.getForm().submit();
```

f 为 Ext.form.FormPanel 类型的对象，通过 getForm()方法可以获取 BasicForm 对象，该对象的 submit()方法通过 ajax 提交表单。这样做已经很好了，但有一点不人性化，用户无法知道提交之后的反馈结果，为了能够判别提交成败与否，在 submit()方法中传递一个 json 对象，该对象有两个回调方法：success 和 failure，分别在表单提交成功和失败的时候自动调用。

```
f.getForm().submit({
    success: function(){
        Ext.Msg.alert("成功", "恭喜，表单提交成功");
```

```
    },
    failure: function(){
        Ext.Msg.alert("失败", "对不起，表单提交失败");
    }
});
```

现在，用户终于知道自己的工作是什么结果了。

当然，基础工作依然要做，我们还要设置请求提交到哪个 Web 组件以及使用什么方法提交，像下面这样：

```
url: "../FormServlet",
method: "post",
```

完整的源代码如下：

```
Ext.onReady(function(){
    var f = new Ext.form.FormPanel({
        //数据传送到服务器的配置
        url: "../FormServlet",
        method: "post",
        baseParams: {extra: "attach", id: 100},
        title: "欢迎登陆",
        width: 300,
        height: 130,
        bodyStyle: "padding: 6px",
        labelAlign: "right",
        frame: true,
        items: [
            new Ext.form.TextField({
                name: "userName",
                allowBlank: false, //不允许为空
                fieldLabel: "用户名",
                labelAlign: "right"
            }),{
                name: "password",
                xtype: "textfield",
                inputType: "password",
                fieldLabel: "密码",
                allowBlank: false //不允许为空
            }
        ],
        buttons:[{
            text: "确定",
```

```
            handler: function(){
                var userName = f.getForm().findField("userName").getValue();
                var password = f.getForm().findField("password").getValue();
                Ext.Msg.alert("", "用户名：" + userName + "<br>密码：" + password);

                //提交表单
                f.getForm().submit({
                    success: function(){
                        Ext.Msg.alert("成功", "恭喜，表单提交成功");
                    },
                    failure: function(){
                        Ext.Msg.alert("失败", "对不起，表单提交失败");
                    }
                });
            }
        },
        {
            text: "重置",
            handler: function(){
                f.getForm().reset();
            }
        }]
    });

    f.render("a");
})
```

有些数据不一定非得通过表单域传输到服务器，传统的做法通过隐藏表单域实现，在 Extjs 中，可以在 FormPanel 配置里通过 baseParams 实现，该配置是一个 json 对象，可以传送中文。代码形如：baseParams: {extra: "中华人民共和国", id: 100}。

有时候，表单提交成功后，希望获取从服务器返回的信息。这个需要服务器组件以 json 格式返回，在本例中，FormServlet 是这样定义的：

```
package com.aptech.ajax2;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class FormServlet extends HttpServlet {

    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();

        System.out.println(request.getParameter("userName"));
        System.out.println(request.getParameter("password"));

        System.out.println(request.getParameter("extra"));
        System.out.println(request.getParameter("id"));

        out.println("{success: true, msg: '服务器结果，成功!', time: '2009 年 5 月 12 日
'}");
        out.flush();
        out.close();
    }
}
```

我们注意粗斜体的那一句，为了标记这是一个成功的请求处理，***success: true*** 必不可少，如果没有这个属性，Extjs 会一直认为处理失败。而 msg 和 time 是自定义的属性，返回到浏览器后按下面的方法得到：

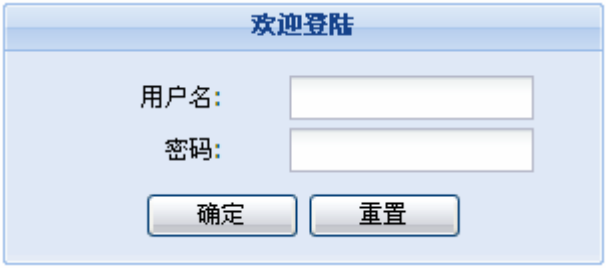
```
f.getForm().submit({
    success: function(f, action){
        Ext.Msg.alert("成功", "恭喜，表单提交成功,服务器反馈的结果是：" +
action.result.msg + ",时间" + action.result.time);
    },
    failure: function(){
        Ext.Msg.alert("失败", "对不起，表单提交失败");
    }
});
```

在 success 回调方法的参数中，f 为当前表单对象，action 对象中的 result 属性保存的就是从服务器返回的 json 对象。

四、小结

本章仅仅介绍了表单的基本使用，表单域也仅限于文本框和密码框，还有更多的选择控件没有提及，但是我们不能逃避，所以在下一章中将会对其他控件进行详细介绍。

先把基本的内容理解，这是必要的。好的学习方法是在简单的基础上逐渐深化，在模仿的基础上创新。我要强调的是，基础的位置是独一无二的，如果想在某一个领域有所建树，深厚的基础对于以后的学习和发展大有裨益。



第十六章：更多表单组件

一、您能说出哪些表单组件呢？

上一章我们介绍了 Ext.form.TextField 类，该类是 Extjs 封装的文本框，与传统的文本框相比，Extjs 赋予了他更丰富的意义，不仅美化了显示效果，提示信息也被包裹起来，所有的东西都不再熟悉，取而代之的是一副副崭新的面孔。

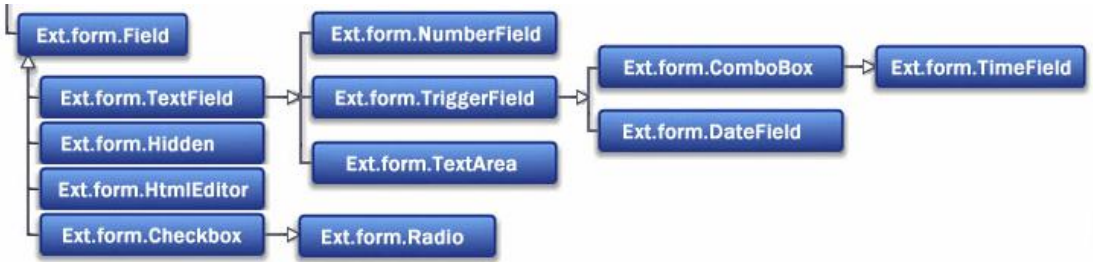
Ext.form.TextField 不只是用于输入文本，还能接收密码以及文件域，由 inputType 选项来决定，该选项有三个值：text、password 和 file，默认为 text。

除了文本框，还有哪些组件呢？下面的表列出了 Extjs 所有的表单组件。

xtype	***class***	***说明***
form	Ext.FormPanel	Form 面板
checkbox	Ext.form.Checkbox	复选框
combo	Ext.form.ComboBox	下拉列表框
datefield	Ext.form.DateField	日期选择器
field	Ext.form.Field	表单字段，表单组件的父类
fieldset	Ext.form.FieldSet	字段分组
hidden	Ext.form.Hidden	隐藏表单域
htmleditor	Ext.form.HtmlEditor	在线 HTML 编辑器
numberfield	Ext.form.NumberField	数字编辑器
radio	Ext.form.Radio	单选按钮
textarea	Ext.form.TextArea	区域文本框
timefield	Ext.form.TimeField	时间组件
checkboxgroup	Ext.form.CheckboxGroup	复选框分组控件
radiogroup	RadioGroup	单选框分组控件

这个表在第八章出现过，当时没有详细解释，现在应该给出一个答案了。

二、表单组件关系图



从图中可以看出，所以表单组件都继承自 Ext.form.Field，其中，TestField、Hidden、HtmlEditor 和 Checkbox 是 Field 的直接子类，其他组件又从这些类中派生而成。

三、组件配置选项介绍

所有组件都是 Field 的直接或间接子类，所以大部分用法都大同小异，我们要习惯从变化的东西中找出不变的东西，帮助我们记忆和理解，死记硬背解决不了根本问题。

- 单选按钮和单选分组控件

```
//性别男
var rdaSexBoy = new Ext.form.Radio({
    name: "rdaSex",
    inputValue: "男",//实际值
    boxLabel: "男",//显示值
    checked: true //默认为男
});

//性别女
var rdaSexGirl = new Ext.form.Radio({
    name: "rdaSex",
    inputValue: "女",
    boxLabel: "女"
});

//性别分组
var grpSex = new Ext.form.RadioGroup({
    name: "sex",
    fieldLabel: "性别",
    items: [rdaSexBoy, rdaSexGirl],
    width: 100
})
```

和文本框不同，选择框通过有两个值：实际值和显示值。显示值通过 boxLabel 指定，而实际值由 inputValue 指定，inputValue 选项值将传送到服务器。如果希望 Radio 默认处理选择状态，设置 checked 即可。

RadioGroup 将 Radio 分组，即使这样，我们也要注意，如果要同组互斥，必须将 Radio 的 name 属性设置成相同的值。而 RadioGroup 仅仅将单选按钮从逻辑上绑定到一起，并且提供一个提示信息。另外，该类的 width 选项很有用，可以适当调整所占的宽度，而 Radio 的 width 则起不到相应的效果。

- 日期选择器

```
//出生日期
var dateBirthday = new Ext.form.DateField({
    name: "dateBirthday",
    fieldLabel: "出生日期",
    width: 200,
    format: "Y-m-d",//指定日期格式,Y 表示四位数的年，m 表示月，d 表示日
    value: new Date() //默认为当前日期
});
```

我们介绍过 Ext.DatePicker，该类为我们提供了一个占地面积很广的日期选择器，Ext.form.DateField 则在原来 Ext.DatePicker 的基础上多了一个文本框，文本框用来保存用户选择的日期。format 指定日期格式，可用的日期格式有"m/d/Y|n/j/Y|n/j/y|m/j/y|n/d/y|m/j/Y|n/d/Y|m-d-y|m-d-Y|m/d|m-d|md|mdy|mdY|d|Y-m-d，如果不指定，默认格式为 m/d/Y。value 则指定日期选择器弹出后的默认值，注意必须是 Date 对象。

可以通过两个方法得到 Ext.form.DateField 的选择值，getValue()返回的是 Date 对象，而 getRawValue()则返回 String 数据，和文本框中的值一致，后者可能更符合我们的需求。

遗憾的是，Ext.form.DateField 无法选择时间，但网上已经出现带时间选择和输入的扩展。

- 复选框及复选框分组

```
//爱好
var chkHobby1 = new Ext.form.Checkbox({
    name: "chkHobby",
    inputValue: "钓鱼",
    boxLabel: "钓鱼",
    checked: true
});

var chkHobby2 = new Ext.form.Checkbox({
    name: "chkHobby",
    inputValue: "网友聚会",
    boxLabel: "网友聚会"
});

var chkHobby3 = new Ext.form.Checkbox({
    name: "chkHobby",
    inputValue: "洗桑拿",
    boxLabel: "洗桑拿"
});

var chkHobby4 = new Ext.form.Checkbox({
```

```
    name: "chkHobby",
    inputValue: "打保龄球",
    boxLabel: "打保龄球"
});

//爱好分组
var grpGobby = new Ext.form.CheckboxGroup({
    name: "hobby",
    fieldLabel: "您的爱好",
    items: [chkHobby1, chkHobby2, chkHobby3, chkHobby4],
    width: 300
});
```

Ext.form.Checkbox 类是复选框的再次封装，选项配置和 Radio 基本相同。Ext.form.CheckboxGroup 为复选框提供逻辑分组，用法和 RadioGroup 相同。

Ø 数字输入框

```
//最喜欢的数字
var numLove = new Ext.form.NumberField({
    name: "numLove",
    fieldLabel: "最喜欢的数字"
});
```

Ext.form.NumberField 其实就是文本框，特殊性在于限定了用户的输入，只能输入数字，包括小数和整数。如果用户不小心输入了两个或两个以上的小数字点，从第二个小数点开始到最后都会被自动清除。

Ø 文本区

```
//家庭住址
var areaAddress = new Ext.form.TextArea({
    name: "areaAddress",
    fieldLabel: "家庭住址",
    width: 500,
    height: 50
});
```

Ext.form.TextArea 对应 HTML 的 textarea 标记，允许用户输入多行文本。

Ø 时间点选择器

```
//上班时间
var timeWork = new Ext.form.TimeField({
```

```
    name: "timeWork",
    fieldLabel: "上班时间"
});
```

Ext.form.TimeField 是一个特殊的 ComboBox，方便用户选择某个时间点，默认情况下每隔 15 分钟一个选择点，用户可以自定义，包括时间选择点的时长和格式。

format 用于定义时间格式，默认为 12 小时制，如果要符合生活习惯，一般采用 24 小时制，该选项必须手动设置为“H:i”，常用的有 g:ia|g:iA|g:i a|g:i A|h:i|g:i|H:i|ga|ha|gA|h a|g a|g A|gi|hi|gia|hia|g|H。increment 选项用于配置时间选择点的间隔，默认为 15 分钟，下面的代码对以上两个选项进行了重写：

```
//上班时间
var timeWork = new Ext.form.TimeField({
    name: "timeWork",
    fieldLabel: "上班时间",
    increment: 30,
    format: "H:i"
});
```

如果希望限制时间范围，minValue 和 maxValue 可以做到，分别用于设置时间范围的最小值和最大值，默认为 00:00 到 24:00。

- 在线编辑器

```
var htmlIntro = new Ext.form.HtmlEditor({
    name : "htmlIntro",
    fieldLabel: "个人简介",
    enableLists: false,
    enableSourceEdit: false,
    height: 150
});
```

Ext.form.HtmlEditor 是一个在线编辑器，和文本区最大的不同在于能为输入的文字指定格式，虽然格式定义不太丰富，但能满足一般性需求。内置的格式有：设置字体大小、设置字体颜色、文字对齐方式、列表、定义超链接、设置字体名称、查看源代码等等。

我们还能够对上述格式显示和隐藏，只将需要的格式显示出来。下面的选项用于设置格式是否可用：

```
    enableFontSize : true,
    enableColors : true,
    enableAlignments : true,
    enableLists : true,
```

```
enableSourceEdit : true,
enableLinks : true,
enableFont : true,
```

定义 Ext.form.HtmlEditor 对象时，将上述配置的值成 false 即可隐藏，默认情况下全部显示。

- 文件域

```
//照片
var txtPhoto = new Ext.form.TextField({
    name: "txtPhoto",
    inputType: "file",
    fieldLabel: "照片",
    width: 500
});
```

一个 TextField 而已，将 inputType 的值设置成了 file。

四、完整源代码

```
//本示例用于添加员工，演示各种表单组件的用法
Ext.onReady(function(){
    //姓名
    var txtName = new Ext.form.TextField({
        name: "txtName",
        fieldLabel: "姓名",
        width: 200
    });

    //密码
    var txtPassword = new Ext.form.TextField({
        name: "txtPassword",
        fieldLabel: "密码",
        inputType: "password",
        width: 200
    });

    //性别男
    var rdaSexBoy = new Ext.form.Radio({
        name: "rdaSex",
        inputValue: "男",//实际值
        boxLabel: "男",//显示值
```

```
        checked: true //默认为男
    });

    //性别女
    var rdaSexGirl = new Ext.form.Radio({
        name: "rdaSex",
        inputValue: "女",
        boxLabel: "女",
        width: 150
    });

    //性别分组
    var grpSex = new Ext.form.RadioGroup({
        name: "sex",
        fieldLabel: "性别",
        items: [rdaSexBoy, rdaSexGirl],
        width: 100
    })

    //出生日期
    var dateBirthday = new Ext.form.DateField({
        name: "dateBirthday",
        fieldLabel: "出生日期",
        width: 200,
        format: "Y-m-d",//指定日期格式,Y 表示四位数的年，m 表示月，d 表示日
        value: new Date() //默认为当前日期
    });

    //爱好
    var chkHobby1 = new Ext.form.Checkbox({
        name: "chkHobby",
        inputValue: "钓鱼",
        boxLabel: "钓鱼",
        checked: true
    });

    var chkHobby2 = new Ext.form.Checkbox({
        name: "chkHobby",
        inputValue: "网友聚会",
        boxLabel: "网友聚会"
    });

    var chkHobby3 = new Ext.form.Checkbox({
        name: "chkHobby",
```

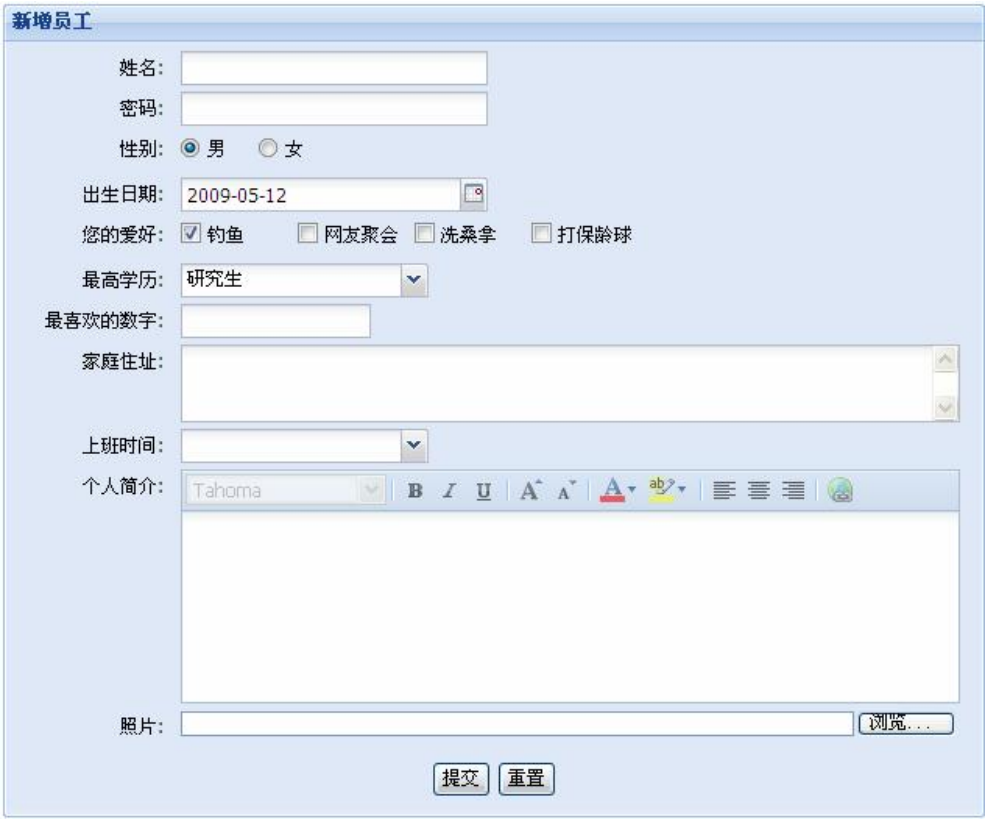
```
        inputValue: "洗桑拿",
        boxLabel: "洗桑拿"
});

var chkHobby4 = new Ext.form.Checkbox({
        name: "chkHobby",
        inputValue: "打保龄球",
        boxLabel: "打保龄球"
});

//爱好分组
var grpGobby = new Ext.form.CheckboxGroup({
        name: "hobby",
        fieldLabel: "您的爱好",
        items: [chkHobby1, chkHobby2, chkHobby3, chkHobby4],
        width: 300
});

//最高学历
//准备数据
var data = [[1, "博士"],
        [2, "硕士"],
        [3, "研究生"],
        [4, "本科"],
        [5, "专科"],
        [6, "大学肄业"],
        [7, "文盲"]];
var proxy = new Ext.data.MemoryProxy(data);
//定义学历结构
var Edu = Ext.data.Record.create([
        {name: "eid", type: "int", mapping: 0},
        {name: "ename", type: "string", mapping: 1}
]);
var reader = new Ext.data.ArrayReader({}, Edu);
var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader
});
store.load(); //加载数据
//创建下拉列表框
var chkEdu = new Ext.form.ComboBox({
        name: "chkEdu",
        fieldLabel: "最高学历",
        store: store,
```

```
        mode: "local",
        triggerAction: "all",
        emptyText: "请选择最高学历",
        displayField: "ename", //显示值
        valueField: "eid", //实际值,
        value: 3 //缺省值
});

//最喜欢的数字
var numLove = new Ext.form.NumberField({
        name: "numLove",
        fieldLabel: "最喜欢的数字"
});

//家庭住址
var areaAddress = new Ext.form.TextArea({
        name: "areaAddress",
        fieldLabel: "家庭住址",
        width: 500,
        height: 50
});

//上班时间
var timeWork = new Ext.form.TimeField({
        name: "timeWork",
        fieldLabel: "上班时间",
        increment: 30,
        format: "H:i"
});

/*var timeWork = {
        xtype: "timefield",
        name: "timeWork",
        fieldLabel: "上班时间"
};*/

//个人简介
var htmlIntro = new Ext.form.HtmlEditor({
        name : "htmlIntro",
        fieldLabel: "个人简介",
        enableLists: false,
        enableSourceEdit: false,
        height: 150
```

```
});

//照片
var txtPhoto = new Ext.form.TextField({
	name: "txtPhoto",
	inputType: "file",
	fieldLabel: "照片",
	width: 500
});

//提交按钮
var btnSubmit = new Ext.Button({
	text: "提交",
	handler: function(){
		f.getForm().submit({
			success: function(form, action){
				form.items.each(function(field){
					if(field.isFormField){
						alert(field.getName() + "=" + field.getValue());
					}
				});
			},
			failure: function(){
				Ext.MessageBox.alert("", "对不起，表单提交失败！");
			}
		});
	}
});

//重置按钮
var btnReset = new Ext.Button({
	text: "重置" ,
	handler: function(){
		f.getForm().reset();
	}
});

var f = new Ext.form.FormPanel({
	url: "../FormServlet",
	method: "post",
	renderTo: "a",
	title: "新增员工",
	style: "padding: 10px",
	frame: true,
```

```
            labelAlign: "right",
            width: 650,
            autoHeight: true,

            items:[txtName, txtPassword, grpSex, dateBirthday, grpGobby, chkEdu, numLove,
areaAddress, timeWork, htmlIntro, txtPhoto],
            buttons:[btnSubmit, btnReset]
    });
})
```

下面是效果图：



五、小结

本章讨论了各种表单组件的基本用法，局限于组件的使用和值的获取。大家应该好好去总结每一个组件的相同点和不同点。

总体来说，表单组件分成两大类型：输入框和选择框。这两种类型在配置的时候细节上有些出入，将这些重点记在心里，做到心中有数，才能处事不惊！

接下来的章节我们将继续讨论表单组件。

第十七章：悬停提示与验证

一、悬停提示

悬停提示在鼠标悬停一小段时间后，会出现一段提示信息。HTML 标准对悬停提示做了实现，效果比较简单。Extjs 为悬停提示提供了一种更加华丽的效果。

表单数据一般都会先通过客户端验证，如果数据无效，可以防止提交，这样能减少客户端与服务器的交互，降低宽带流量，增强用户体验。Extjs 默认采用悬停提示来告知用户。所以，将悬停提示和验证放到同一个章节自然有他的道理。

Extjs 支持五种悬停提示，分别为 qtip、side、under、title 和 around，其中，qtip 是默认的支持方式。但是，真正实现的是前四种，around 只定义了，并没有实现，程序员可以根据自己的需要去完成。

我们一起来看看前四种效果的截图：

上述四种悬停提示的效果有些差异，qtip 是由 Extjs 自定义的层，side 多了一个错误图标，under 最直观，提示信息显示在组件的下方，而 title 又回到了原始时代。

Extjs 的悬停提示是由 Ext.form.MessageTargets 类来定义的，而且只针对表单组件，如果想知道具体的实现过程，可以查看 Field.js 文件第 565 行至 679 行的源代码。

二、悬停提示的用法

默认情况下，悬停提示并没有启用，在代码中必须先执行下列语句：

```
//启动悬停提示
Ext.QuickTips.init();
```

一般我们会碰到三种情况需要悬停提示：HTML 标记、表单组件和非表单组件，不同的情况需要指定的提示信息属性不一定全部相同。我们先对 HTML 标记和非表单组件进行示范。

- 非表单组件

按钮(Ext.Button)是一个典型的非表单组件，虽然在表单中也有他的使用价值，但是，Extjs 对组件分类时，还是将他划分到了基本组件中。

我们将以按钮为例演示非表单组件的悬停提示，如果想彻底了解，最好的办法就是阅读源代码，因为我发现，Extjs 的源代码并不复杂，而且其中包含的信息量比任何文档都丰富，可以很轻易找到相关问题的解决方案。

我不知道这是第几次呼吁您看源码了，不管您持什么态度，我还是坚信我的提议。

Button 支持两种类型的悬停提示：qtip 和 title，默认是前者，通过 tooltipType 选项进行配置，而提示信息保存在 tooltip 选项中。

```
Ext.onReady(function(){
    Ext.QuickTips.init();

    var btn = new Ext.Button({
        text: "悬停演示",
        renderTo: Ext.getBody(),
        tooltip: "这是一个悬停演示的按钮，怎么样？<br>是不是很漂亮？",
        tooltipType: "qtip"
    });
});
```

有时候，悬停提示信息过长，我们愿意换行显示，对于 qtip 类型来说，在提示信息中使用
标签可以做到，如果是 title 类型，使用\r\n 即可。

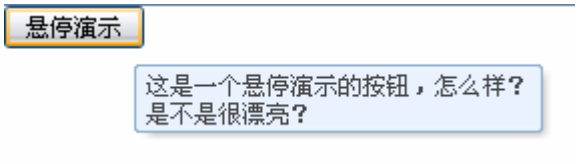
事实上，qtip 类型的悬停提示完全通过 div 实现，所以，在视觉上做得更加到位，我们

可以修改样式轻易实现不同的效果。下面是 qtip 类型的悬停提示生成的代码：

```
<div id="ext-gen25" class="x-shadow" style="z-index: 19999; left: 76px; top: 36px; width: 204px; height: 34px; display: none;">
<div class="xst">
<div class="xstl"/>
<div class="xstc" style="width: 192px;"/>
<div class="xstr"/>
</div>
<div class="xsc" style="height: 22px;">
<div class="xsml"/>
<div class="xsmc" style="width: 192px;"/>
<div class="xsmr"/>
</div>
<div class="xsb">
<div class="xsbl"/>
<div class="xsbc" style="width: 192px;"/>
<div class="xsbr"/>
</div>
</div>
<div id="ext-comp-1001" class="x-tip" style="position: absolute; z-index: 20000; visibility: hidden; width: 204px; left: 72px; top: 33px; display: none;">
<div class="x-tip-tl">
<div class="x-tip-tr">
<div class="x-tip-tc">
<div id="ext-gen10" class="x-tip-header x-unselectable" style="-moz-user-select: none;">
<span id="ext-gen24" class="x-tip-header-text"/>
</div>
</div>
</div>
</div>
<div id="ext-gen11" class="x-tip-bwrap">
<div class="x-tip-ml">
<div class="x-tip-mr">
<div class="x-tip-mc">
<div id="ext-gen12" class="x-tip-body" style="height: auto; width: 192px;">
这是一个悬停演示的按钮，怎么样？
<br/>
是不是很漂亮？
</div>
</div>
</div>
</div>
<div class="x-tip-bl x-panel-nofooter">
```

```
<div class="x-tip-br">
<div class="x-tip-bc"/>
</div>
</div>
</div>
</div>
```

可能有人会疑惑为什么生成这么多的代码。在这段冗长的代码中，主要包括四大内容：阴影、圆角、边框和提示内容。所以，为了迎合客户的审美需求，真是害苦了我们这些程序员；所以，满足了客户的审美需求，真是美死了我们这些程序员。程序员永远都是苦与爽的共同体。



Ø HTML 标记

如果要为 HTML 标记定义悬停提示，有更多的方式可以选择。但是和 Extjs 组件相比，实现方式有了不一样的改变。

假设在页面上有一个文本框：

```
<div id="t" style="margin: 100px;">
    <input name="ta">
</div>
```

现在，当鼠标悬停在该文档框上时，出现一段悬停提示，我们可以按照下面的方式进行定义：

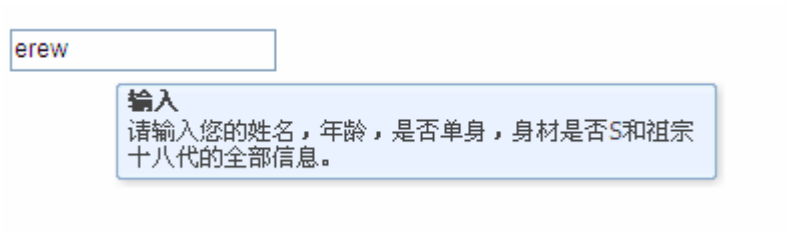
```
Ext.QuickTips.register({
    text: "请输入您的姓名，年龄，是否单身，身材是否 S 和祖宗十八代的全部信息。
",
    target: "ta",
    title: "输入"
});
```

不一样吧，呵呵。register 类似于一个 static 方法，该方法为指定的标记注册悬停提示。默认情况下，提示框最小宽度为 40，最大宽度为 300，如果文字长度超过最大宽度将自动换行。title 是标题文字，text 就是提示内容了，而 target 则为应用的目标对象。什么？ta 是什么？你说呢？

我们还可以通过 width 选项指定提示框的宽度，最大支持 500，dismissDelay 指定提示

框悬停的时间，以毫秒计算。如果您想让提示框长命百岁，设置 autoHide: false 吧。

来看看效果图：



哎哟，怎么越看越像一个面板啊？哈，对了，这就是一个 Panel。到底怎么回事？其实，提示效果的绘制是通过 Ext.Tip 类实现的，而 Ext.Tip 的父类就是 Ext.Panel，你说这不是面板是什么？

如果我们对 Ext.Tip 提供的默认选项值不满意，绝对可以偷梁换柱，变成你想要的效果。对于 Ext.tip，可以重写的选项配置包括：

hideDelay：消失的时间，试了一下，好像没效果
maxWidth：最大宽度
minWidth：最小宽度
showDelay：显示时间
trackMouse：提示框是否跟着鼠标一起走。

典型的实现如下：

```
Ext.apply(Ext.QuickTips.getQuickTip(), {
        maxWidth: 200,
        minWidth: 100,
        showDelay: 50,
        trackMouse: true,
        hideDelay: 5000
});
```

注意啦，这个代码是全局定义，也就是说，他不是针对单个 HTML 标记或组件，而是对所有的标记和组件都有效。

下面是本节的完整代码，便于您参考：

```
Ext.onReady(function(){
    Ext.QuickTips.init();

    /*Ext.apply(Ext.QuickTips.getQuickTip(), {
            maxWidth: 200,
            minWidth: 100,
```

```
        showDelay: 50,
        trackMouse: true,
        hideDelay: 5000,
        animate: true
    });*/

    var btn = new Ext.Button({
        text: "悬停演示",
        renderTo: Ext.getBody(),
        tooltip: "这是一个悬停演示的按钮，怎么样？<br>是不是很漂亮？",
        tooltipType: "qtip"
    });

    Ext.QuickTips.register({
        text: "请输入您的姓名，年龄，是否单身，身材是否 S 和祖宗十八代的全部信
息。",
        target: "ta",
        title: "输入",
        autoHide: false
    });
});
```

还有个更好玩的东东，可以直接将悬停选项配置在 HTML 标记上，像下面这样：

```
<input type="button" value="OK" ext:qtitle="OK Button" ext:qwidth="100"
     ext:qtip="This is a quick tip from markup!"></input>
```

这个不是我自己写的哦！源代码里自带的，我运行了一下，没问题。

三、表单组件验证

表单组件自身有验证功能，而且能满足一般性需求，一旦验证出错，将会出现非常醒目的提示，并且，表单提交被阻止。我们并不需要写很多代码，更多的时候只需要配置几个选项，如果自带的验证功能无法满意您变态的要求，还支持正则表达式，有了正则表达式，相信一切都能搞定了。

Ext.form.Field 定义了几个属性用于配置是否启用验证功能：validationEvent 指定什么事件触发时验证，默认为 keyup，如果将该属性设为 false，即使有键盘输入也不再验证；validateOnBlur 指组件失去焦点后是否验证，默认为 true。默认情况下，用户在输入数据和失去焦点时都会验证数据是否合法。

Ext.form.Field 还可以自定义验证未通过的显示效果，invalidClass 是验证没有通过时表单组件采用的类选择器；focusClass 是组件获得焦点时采用的类选择器；invalidText 是输入

数据并没有通过验证时显示的提示信息。

Ø 非空验证

如果组件内容必填，可以采用非空验证，该验证能保证输入内容不会为空。定义表单组件时，配置 allowBlank 选配为 false 即可。

上面的内容通过一个统一的示例来说明：

```
var txtName = new Ext.form.TextField({
    name: "txtName",
    fieldLabel: "姓名",
    width: 200,
    allowBlank: false,
    regex: /^\w{6,}$/, //为了配置 invalidText，此处使用正则表达式，即至少要有 6 个字符
    invalidClass: "myvalid",
    invalidText: "温馨提示：通过验证，结果为无效数据",
    focusClass: "myfocus",
    validationEvent: false,
    validateOnBlur: false,
    validationDelay: 2000
});
```

Ø 长度验证

minLength 指定输入的最小长度，maxLength 指定输入的最大长度，minLengthText 指定没有达到最小长度时的提示信息，maxLengthText 指定超过最大长度时的提示信息。

```
//密码
var txtPassword = new Ext.form.TextField({
    name: "txtPassword",
    fieldLabel: "密码",
    inputType: "password",
    minLength: 6,
    maxLength: 10,
    minLengthText: "温馨提示：最小长度必须为 6 个字符",
    maxLengthText: "温馨提示：最大长度只能为 10 个字符",
    width: 200
});
```

Ø vtype 验证

vtype 验证是一种预先定义好的验证，可用于验证邮箱、网址、字符与下划线、字符数

字与下划线。

vtype 验证由 Ext.form.Vtypes 定义，其实也是对正则表达式的封装，我们可以看看是如何定义正则表达式的：

```
var alpha = /^[a-zA-Z_]+$/; //字母和下划线
var alphanum = /^[a-zA-Z0-9_]+$/; //字母、数字和下划线
var email = /^([\w]+)(.[\w]+)*@([\w-]+\.){1,5}([A-Za-z]){2,4}$/;//邮箱
var url = /(((https?)|(ftp)):\/\/([\-\w]+\.)+\w{2,3}(\/[%\-\w]+(\.\w{2,})?)*(([\w\-\.\?\\\/+@&#;`~=%!]*)(\.\w{2,})?)*\/?)/i;//网址
```

上面的四个变量名即为 vtype 的名称，注意 vtype 最先定义在 Ext.form.TextField 类中，如果组件不是 Ext.form.TextField 的子类，则不接受 vtype 验证。看示例：

```
//个人喜欢的网站
var txtUrl = new Ext.form.TextField({
    name: "txtUrl",
    fieldLabel: "个人喜欢的网站",
    vtype: "url",
    width: 500
});
```

如果要验证邮箱，将 vtype 的值设为“email”，其他的一样。

- 正则表达式验证

正则表达式也是定义在 Ext.form.TextField 类，所以，同样只有继承该类的组件才能使用。

regex 选项对正则表达式提供支持，他是一个 RegExp 对象，也可以使用 javascript 的特有语法/^$/，如 regex: /^\w{6,}$/即是一个正则表达式，至少为 6 个字符。regexText 选项为未通过时的提示文本，如果同时设置了 invalidText，则以 regexText 为准。

关于正则表达式的更多内容，请阅读相关资料。

四、小结

其实 Extjs 内置的验证功能并不多，但是有了正则表达式的支持，不管有多复杂的验证都所向无敌了。

如果您未接触过正则表达式，赶紧学吧。还来得及！

第十八章：FormPanel 布局与初始化

一、布局概述

客户的需求不会总是一成不变，我们更愿相信天下一定存在千奇百怪的需求，默认情况下，FormPanel 内的组件布局排列从上而下，这样太单调太古板了。如果客户不愿意总是这样，我们该怎么办？放弃 Extjs 吗？

当然不，Extjs 绝对不会这么简单，事实上，通过 FormPanel 的布局控制，可以做出客户想要的任何布局，而且和 swing 相比，简单了很多。

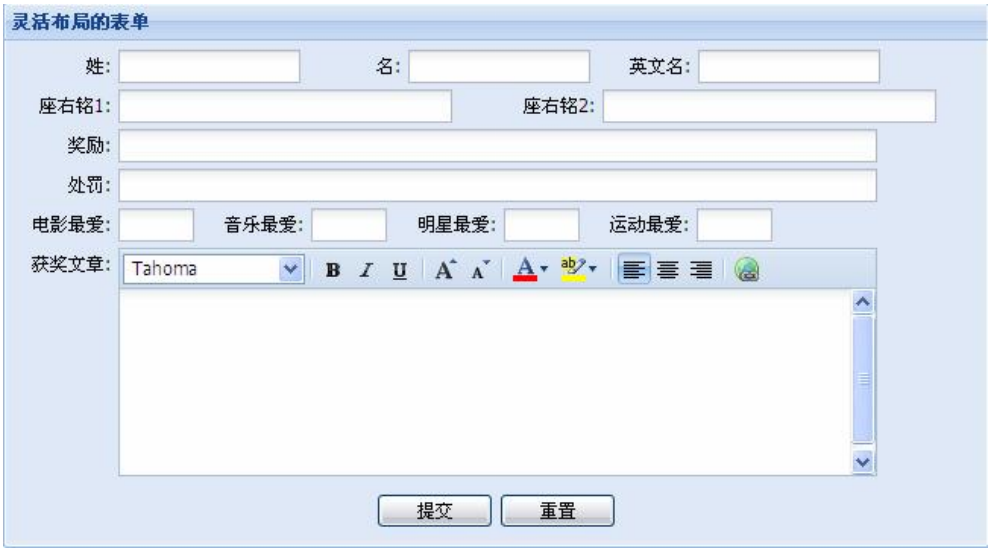
FormPanel 有两种布局：form 和 column，form 是纵向布局，column 为横向布局。默认为后者。使用 layout 属性定义布局类型。对于一个复杂的布局表单，最重要的是正确分割，分割结果直接决定布局能否顺利实现。

如果不再使用默认布局，那么我们必须为每一个元素指定一种布局方式，另外，还必须遵循以下几点：

- 落实到任何一个表单组件后，最后总是 form 布局
- defaultType 属性不一定起作用，必须显式为每一个表单组件指定 xtype 或 new 出新对象
- 在 column 布局中，通过 columnWidth 可以指定列所占宽度的百分比，如占 50%宽度为.5。

二、分割吧！

学会分割，我们从一个样例开始：



哦，真要命，太复杂了。

但是别慌，我们要静下心来，将他大卸八块，剖析出一个合理的结构，像下面这样：

我们发现，布局其实是由行和列组件成，分成由左往右和由上往下两个方向，由左往右叫 column，由上往下叫 form。

整个大的表单是 form 布局，从上往下放置了五个小布局，在这里我以行 n 标记，我们以行 1 为例进行分析。行 1 从左往右有三个表单组件，所以是 column 布局，行 1 我们用结构这样定义：

```
{
    layout: “column”,
    items:[{},{},{}] //items 表示指定布局内的表单组件集合，在此有三个
}
```

行 1 内其实还有三个 form 布局，因为每个布局中只有一个表单组件，所以看起来并不是那么明显，我们完全可以放置多个表单组件到布局中。每一个布局使用下面的结构定义：

```
{
    layout: “form”,
    items:[{}] //只有一个表单组件
}
```

上面的两个结构最终要组装到一起：

```
{
    layout: “column”,
    items:[{
        layout: “form”,
```

```
            items:[{}]
        },{
            layout: “form”,
            items: [{}]
        },{
            layout: “form”,
            items: [{}]
}]
}
```

其他行的分析是同样的道理，在此不再赘述。图中的效果对应的代码如下：

```
Ext.onReady(function(){
    var form = new Ext.form.FormPanel({
        title: "灵活布局的表单",
        width: 650,
        autoHeight: true,
        frame: true,
        renderTo: "a",
        layout: "form", //整个大的表单是 form 布局
        labelWidth: 65,
        labelAlign: "right",

        items:[{ //行 1
            layout: "column",   //从左往右的布局
            items:[{
                columnWidth: .3, //该列有整行中所占百分比
                layout: "form",      //从上往下的布局
                items: [{
                    xtype: "textfield",
                    fieldLabel: "姓",
                    width: 120
                }]
            },{
                columnWidth: .3,
                layout: "form",
                items: [{
                    xtype: "textfield",
                    fieldLabel: "名",
                    width: 120
                }]
            },{
                columnWidth: .3,
                layout: "form",
```

```
                items: [{
                    xtype: "textfield",
                    fieldLabel: "英文名",
                    width: 120
                }]
            }]
        },{//行 2
            layout: "column",
            items:[{
                columnWidth: .5,
                layout: "form",
                items:[{
                    xtype: "textfield",
                    fieldLabel: "座右铭 1",
                    width: 220
                }]
            },{
                columnWidth: .5,
                layout: "form",
                items:[{
                    xtype: "textfield",
                    fieldLabel: "座右铭 2",
                    width: 220
                }]
            }]
        },{//行 3
            layout: "form",
            items:[{
                xtype: "textfield",
                fieldLabel: "奖励",
                width: 500
            },{
                xtype: "textfield",
                fieldLabel: "处罚",
                width: 500
            }]
        },{//行 4
            layout: "column",
            items:[{
                layout: "form",
                columnWidth: 0.2,
                items:[{
                    xtype: "textfield",
                    fieldLabel: "电影最爱",
```

```
                    width: 50
                }]
            },{
                layout: "form",
                columnWidth: 0.2,
                items:[{
                    xtype: "textfield",
                    fieldLabel: "音乐最爱",
                    width: 50
                }]
            },{
                layout: "form",
                columnWidth: 0.2,
                items:[{
                    xtype: "textfield",
                    fieldLabel: "明星最爱",
                    width: 50
                }]
            },{
                layout: "form",
                columnWidth: 0.2,
                items:[{
                    xtype: "textfield",
                    fieldLabel: "运动最爱",
                    width: 50
                }]
            }]
        },{//行 5
            layout: "form",
            items:[{
                xtype: "htmleditor",
                fieldLabel: "获奖文章",
                enableLists: false,
                enableSourceEdit: false,
                height: 150
            }]
        }],
        buttonAlign: "center",
        buttons:[{
            text: "提交"
        },{
            text: "重置"
        }]
    });
```

```
})
```

表单布局看起来是复杂的，只要找到了其中的规律，所有的侵略者都是纸老虎，怎么做？你自己看着办吧。去仔细研究一下，等到你真正懂得的时候，才是功成名就的时候。

三、表单初始化

把表单初始化放到这一章，确实有点不伦不类哈，当初我觉得布局内容太少，而本节又不适合单独成章，所以干脆把这两个东东放到一起讲咯，所以就不伦不类了。

但是，内容的安排无法阻止我们学习的热情，继续吧。

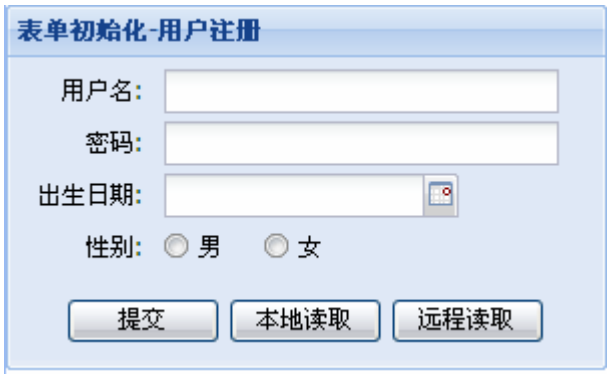
为什么要初始化表单呢？这个操作其实很常见，在修改数据时，必须先将已有的值填充到表单组件中，然后在原来的基础上进行修改，别告诉我你没做过修改，哈哈。

FormPanel 通常支持两种初始化表单组件的方法：

- Ø 本地初始化：创建二维数据或 json 对象，使用 Ext.form.BasicForm 的 setValues()方法填充；
- Ø 远程初始化：从远程服务回获取 json 对象数组，通过 Ext.form.FormPanel 的 load()方法填充。

可能还有别的方法，暂时还没有去研究哈。这段源代码只是粗略看了一下，似乎没别的办法了，嘿嘿。

下面是一个注册表单，用户输入用户名、密码、生日和性别，以便注册，没什么实际业务功能，为了演示而演示，请勿对号入座，谢谢！



生成该表单的代码如下：

```
Ext.onReady(function(){
    var form = new Ext.form.FormPanel({
        title: "表单初始化-用户注册",
        width: 300,
```

```
        autoHeight: true,
        frame: true,
        renderTo: "a",
        labelWidth: 65,
        labelAlign: "right",
        defaultType: "textfield",

        items:[{
            name: "userName",
            fieldLabel: "用户名",
            width: 200
        },{
            name: "password",
            fieldLabel: "密码",
            inputType: "password",
            width: 200
        },{
            name: "birthday",
            fieldLabel: "出生日期",
            xtype: "datefield",
            format: "Y-m-d",
            width: 150
        },{
            name: "sexGroup",
            fieldLabel: "性别",
            xtype: "radiogroup",
            width: 100,
            items:[{
                    name: "sex",
                    xtype: "radio",
                    boxLabel: "男",
                    inputValue: "男"
                },{
                    name: "sex",
                    xtype: "radio",
                    boxLabel: "女",
                    inputValue: "女"
                }
            ]
        }],
        buttons:[{
            text: "提交"
        },{
            text: "本地读取",
```

```
                handler: function(){
                    var json = {
                        userName: "张海军",
                        password: "admin",
                        birthday: "1980-08-08",
                        sex: "女"
                    };
                    form.getForm().setValues(json);
                }
            },{
                text: "远程读取"
            }]
    });
})
```

点击“本地读取”的按钮后，结果是前面三个组件的初始化一切正常，唯独单选按钮丝毫没半点反应。于是，我改代码，看源代码，改呀改，看呀看，最后得出结论，Extjs 根本没做好，这可能是 Extjs 的 bug。哈，终于看到 Extjs 弱智的一面了。

于是，Google，看别人怎么说的，果然不出所料啊，幸好 Ext 官方论坛用户 vtswingkid 开发的 Ext.ux.RadioGroup 扩展解决了这一问题。您可以去 Ext 的官方页面 http://extjs.com/forum/showthread.php?t=23250 下载该组件，不过要先注册啊，而且必须邮箱验证，总之很麻烦，如果您不嫌弃，用我的账号吧：lizanhong/aaaaaa。下载的压缩包有一个 js 文件和 test.html 文件，后者演示了该组件的基本用法。

现在，我们用 Ext.ux.RadioGroup 代替 Ext.form.RadioGroup，注意，要改的地方很多，您最好一行行比较看清楚。

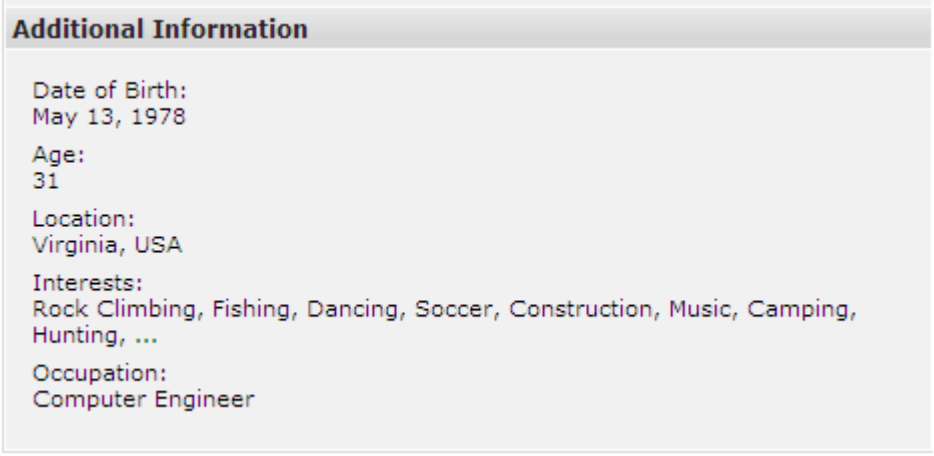
```
{
    name: "sexGroup",
    fieldLabel: "性别",
    xtype: "ux-radiogroup",
    horizontal:true, //水平放置
    radios:[{
                name: "sex",
                boxLabel: "男",
                value: "男"
            },{
                name: "sex",
                boxLabel: "女",
                value: "女"
            }
    ]
}
```

新增的配置选项我用下划线标记出来了。在 RadioGroup 中，分组和 Radio 组件被整合到了一起，水平排列还是垂直排列不再通过布局实现，而是由 horizontal 配置决定，如果不设置该选项则垂直排列。items 也被 radio 替换了，实际值不再是 inputType，取而代之的是 value。

接下来将“本地读取”按钮的单击事件修改一下：

```
{
    text: "本地读取",
    handler: function(){
        var json = {
            userName: "张海军",
            password: "admin",
            birthday: "1980-08-08",
            sexGroup: "女"
        };
        form.getForm().setValues(json);
        alert(form.getForm().findField("sexGroup").getValue());
    }
}
```

OK，大功告成了。用十二万分的诚意感谢 vtswingkid，您是优质的美国公民，以下是 vtswingkid 在论坛中的基本资料：



远程读取相对来说复杂了些。从服务器返回的 json 对象必须和 FormPanel 建立映射关系，即哪一个值填充到哪个表单组件中。映射关系信息保存在 DataReader 对象中。因为我们返回的 json 数组，所以使用他的子类 JsonReader。另外，JSonReader 对象必须是 BasicForm 的 reader 选项值。

我们依然采用 Servlet 从服务器返回数据。

```
public class InitValuesServlet extends HttpServlet {
```

```
    public void service(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();

        String json = "[{userName: '张海军',password: 'admin',birthday:
'1980-08-08',sexGroup: '男'}]";
        out.println(json);
        out.flush();
        out.close();
    }
}
```

创建 JsonReader 对象，用来定义记录结构，并建立和 json 对象的映射关系。

```
var reader = new Ext.data.JsonReader({},[
    {name: "userName", type: "string", mappging: "userName"},
    {name: "password", type: "string", mappging: "password"},
    {name: "birthday", type: "string", mappging: "birthday"},
    {name: "sexGroup", type: "string", mappging: "sexGroup"}
]);
```

在这里，name 是表单组件的 name 属性值，mapping 是服务器返回的 json 对象映射的属性名。即 name 取决于表单组件的 name 属性，而 mapping 取决于 json 对象的属性名，我画了个箭头表明这种关系。

最后来看看“远程读取”按钮的代码：

```
{
    text: "远程读取",
    handler: function(){
        form.getForm().reader = reader;
        form.load({url: "../InitValuesServlet"});
    }
}
```

注意啦注意啦，reader 是赋给 form.getForm().reader，而不是给 form。load 方法负责通过 ajax 从指定的 Web 组件获取数据，然后，自动填充。

本章的完整源代码请看附带的示例。太长了，复制到教程中我怕被人说滥竽充数。表单初始化这一节的示例是同一个，只是在原来的基础上作了些修改，所以请从头到尾依次看完，别拦腰阅读。

四、小结

表单布局最终可以分割成行与列的组合，行叫 column 布局，列叫 form 布局。学会分割复杂结构的表单能帮助我们理解，并且构造出更复杂的表单结构。

其实默认的 form 布局能满足大部分需求，但是不是全部，所以，搞懂表单布局刻不容缓。

表单初始化也是一个基本功能，Extjs 为我们提供了两种不同的方法来初始化表单控件，事实上，GridPanel 中的行也可以和 Form 绑定，在 BasicForm 类中定义了 loadRecord()负责此工作，甚至能通过表单来更新 GridPanel 中的行。这是后话。

第十九章：叹为观止的表格组件——GridPanel

一、表格、表格面板

在项目中会大量用到表格，表格用于显示一系列有规则的数据，排列整齐，便于阅读。在 C/S 架构中，ListView 就是表格的一种表现，B/S 中传统的页面设计直接使用 Table 标记，但是该标记功能太有限了，列不能拖动，不能排序，需要大量的样式才能得到好的效果，不同的浏览器渲染各不相同，不同筛选列……很多客户喜欢的功能他都不具备，所以，一个功能更加强大更符合项目实际需求的表格组件应运而生。他就是令人叹为观止的 GridPanel。

表格相关的组件很多，大多定义在 Ext.grid 命名空间中，GridPanel 就是其中的一个，当然在该命名空间中，还有许多周边组件和类，而且采取分层结构构建，各层职责非常清晰，理解这些复杂组件的结构是驾驭他们并灵活运用的基础。

GridPanel 可以说是众星捧月，为了让他更好的工作，很多人在背后默默地付出，下面的类都是他的支持者：

Ext.grid.ColumnModel：列模型，定义 GridPanel 的表头；

Ext.grid.RowSelectionModel：行选择模型，定义行的选择操作；

Ext.grid.GridView：视图，负责表格面板效果的渲染；

Ext.data.Store：数据源，负责为表格面板提供各种格式的数据；

Ext.grid.RowNumberer()：序号生成器，负责为每一行生成一个从 1 开始的序号；

Ext.grid.CheckboxSelectionModel：带复选框的行选择模型，为每行生成一个复选框，便于多选；

在第 10 章、第 11 章、第 12 章学过 ComboBox 后，本章将不再困难。

二、列模型与数据

Ext.grid.GridPanel 其实就是一 Panel，Ext.Panel 是他的父类。在面板上放置表格数据就成了 Ext.grid.GridPanel。掌握好 Ext.grid.GridPanel 组件，必须明白他自身的结构。

从外观上看，表格分成三大部分：表头、数据行和分页栏。分页栏我们会独自成章，先来看表头和数据行。

表头为表格列提供一系列信息，包含了列的说明、列的宽度、是否可以改变列的大小、是否排序、是否出现菜单等数据。我们可以根据客户需要完成相应的定制。

表头由列模型 Ext.grid.ColumnModel 定义，是 Ext.util.Observable 的子类，所以，列模型也会触发相应的事件。列模型通常会包含若干个列的信息，每一列的信息保存在 json 结构中，一个列的配置选项有：

配置选项名	*类型*	*作用*
header	String	列头文字说明
dataIndex	String	记录结构中的 name 属性值
width	Number	列的宽度
sortable	Boolean	是否排序
fixed	Boolean	是否固定宽度
resizable	Boolean	是否能改变列的宽度
menuDisabled	Boolean	单击列头后是否出现菜单
tooltip	String	悬停提示
renderer	Function	自定义单元格内容
align	String	列的对齐方式，有 left、center 和 right

下面的图是我们的任务：

万丈高楼平地起，我们先定义好列模型：

```
//列模型
var cm = new Ext.grid.ColumnModel([
    {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
    {header: "性别", width: 40, dataIndex: "Sex", align: "center"},
    {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
    {header: "学历", width: 80, dataIndex: "Education", align: "center"},
    {id: "memo", header: "备注", dataIndex: "Memo"}
]);
```

在上述代码中，我们看到，每一列的信息都被封装在 json 对象中，并且组成一个数组传送到 Ext.grid.ColumnModel。其中，dataIndex 属性最需要注意，表示记录结构中的 name 属性值。

我们必须用到第 10 章的内容创建 DataProxy、DataReader 和 Store 对象，Store 对象和 GridPanel 绑定，GridPanel 就会显示 Store 中的数据了。如果这些内容不太明白，请返回第十章继续操练内功。

这里我们假设采用 json 格式来提供测试数据。

```
//准备数据
var data = [{
```

```
    name: "李赞红",
    sex: "男",
    birthday: Date.parseDate("1979-04-11", "Y-m-d"),
    edu: "本科",
    memo: "无备注"
},{
    name: "陈南",
    sex: "男",
    birthday: Date.parseDate("1987-08-06", "Y-m-d"),
    edu: "本科",
    memo: "一个小帅哥哈"
},{
    name: "易珊静",
    sex: "女",
    birthday: Date.parseDate("1980-05-12", "Y-m-d"),
    edu: "本科",
    memo: "无备注"
},{
    name: "张海军",
    sex: "男",
    birthday: Date.parseDate("1980-12-11", "Y-m-d"),
    edu: "本科",
    memo: "北大青鸟最帅的老师"
}];

//Proxy
var proxy = new Ext.data.MemoryProxy(data);

//Record 定义记录结果
var Human = Ext.data.Record.create([
    {name: "Name", type: "string", mapping: "name"},
    {name: "Sex", type: "string", mapping: "sex"},
    {name: "Birthday", type: "string", mapping: "birthday"},
    {name: "Education", type: "string", mapping: "edu"},
    {name: "Memo", type: "string", mapping: "memo"}
]);

//Reader
var reader = new Ext.data.JsonReader({}, Human);

//Store
var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader
```

```
});

store.load();
```

列模型中的 dataIndex 必须和 Human 结构中的 name 属性值一一对应，这样就知道每一列显示什么值了。

现在，列模型和数据都准备好了，将这些数据做为 GridPanel 的选项配置传递到 GridPanel 的构造方法中，并适当设置 GridPanel 的外观参数，效果就出来了。

```
var grid = new Ext.grid.GridPanel({
    title: "中国公民",
    width: 600,
    autoHeight: true,
    cm: cm,
    store: store,
    renderTo: "a",
    autoExpandColumn: "memo" //自动伸展，占满剩余区域
});
```

很多都是 Ext.Panel 中非常熟悉的参数，需要指出的是 autoExpandColumn 选项，该选项很有用，他能自动占用剩余的空间，将面板填满，autoExpandColumn 的属性值是列模型中的 id 属性值，为了配合该功能，“备注”列和别的列不一样，多了一个“id”属性，千万谨记。

三、加强版的列模型

除了完成上述的基本功能之外，我们可以让 GridPanel 更加符合实际需求。上节的示例其实并不完美：数据行没有序号、日期内容不符合习惯、性别太古板，用图片表示更好、没有操作按钮或链接……

现在，针对上面的需求，我们加以改善。

Ø 显示序号

这可以说是一个基本功能，通过序号我们一眼就能看出当前表格的行数。Extjs 为此专门写了一个叫 Ext.grid.RowNumberer 的类，该类通过 prototype 添加了一个 json 类型的属性，其内容和列模型中的配置如出一辙。我们不如看看他的源码：

```
Ext.grid.RowNumberer.prototype = {
      header: "",
      width: 23,
      sortable: false,
      fixed:true,
```

```
    menuDisabled:true,
    dataIndex: '',
    id: 'numberer',
    rowspan: undefined,
    renderer : function(v, p, record, rowIndex){
        if(this.rowspan){
            p.cellAttr = 'rowspan="'+this.rowspan+'"';
        }
        return rowIndex+1;
    }
};
```

从代码中看出，这是一个特殊的列，没有列头说明性文字，宽度为 23 个像素且被固定，不能排序，也没有菜单。这就是 RowNumberer。

Ø 自定义显示内容

在例子中显示性别时，我们使用文字，这样够清楚了，但我希望更花哨点，用图标代替文字，让效果更具冲击力。

列模型提供了一个回调函数 renderer，该函数可以根据当前列的信息进行再加工，获得更加有用的结果。该函数的完整定义如下：

```
renderer：function(value, metadata, record, rowIndex, colIndex, store){

}
```

这个方法的参数真丰富啊，听我慢慢解释。

value：原始值

metadata：可能的值为 css 或 attr

record：Ext.data.Record，GridPanal 的记录结构定义

rowIndex：行索引

colIndex：列索引

store：Ext.data.Store，数据源

我们将性别变成对应图标，代码如下：

```
renderer: function(v){
    if(v == "男"){
        return "<img src='../imgs/134.gif'>";
    }else{
        return "<img src='../imgs/133.gif'>";
    }
}
```

参数 v 就是单元格中的原始值，如果为“男”，返回名为 134.gif 的图标，如果为女，则返回名为 133.gif 的图标，通过这种方式我们可以实现更多的效果，如产品的预览图等。该方法中其他参数被我忽略了，没有关系。

Ø 日期格式化

如果列模型中定义的列类型为 date，显示的结果会把我们搞晕，更多的时候，我们必须对 Date 类型的数据进行格式化，达到我们的阅读要求。最直观的方法是定义 renderer 方法，在方法中返回格式化后的数据，但是 Extjs 并不要求我们必须这样做，而是提供更加简洁的作法：renderer: Ext.util.Format.dateRenderer("Y-m-d")

最后效果如下图：

列模型的改动比较大，我们一起来看看完整的源代码：

```
//列模型（加强版）
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
    {header: "性别", width: 40, dataIndex: "Sex", align: "center",
        renderer: function(v){
            if(v == "男"){
                return "<img src='../imgs/134.gif'>";
            }else{
                return "<img src='../imgs/133.gif'>";
            }
        }
    },
    {header: "生日", width: 100, format: "Y-m-d", dataIndex: "Birthday", renderer: Ext.util.Format.dateRenderer("Y-m-d")},
    {header: "学历", width: 80, dataIndex: "Education", align: "center"},
    {id: "memo", header: "备注", dataIndex: "Memo"},
    {header: "操作", width: 150, dataIndex: '', menuDisabled:true, renderer: function(v){return "<span style='margin-right:10px'><a href='#'>修改</a></span><span><a href='#'>删除</a></span>";}}
]);
```

四、小结

学完本章您一定欣喜若狂，Extjs 为我们提供了一个非常完美的表格实现，不管从界面上还是从功能上都无懈可击。其实，GridPanel 的好远远不止这些，好戏在后头。

掌握每一个章节都是不小的进步，我们就是在学习中，一天天成长……

第二十章：行模型与 Grid 视图

一、行选择模型

行选择模型提供对数据行的选择操作，从来没有见过像 GridPanel 这样支持如此直观和多样的行选择，默认情况下，GridPanel 使用 Ext.grid.RowSelectionModel 作为行选择模型，该选择模型在不经任何设置的情况下即支持多选。对于选择的行，我们可以通过与其他兄弟的配合访问、操作甚至修改数据。行选择模型还定义了一系列事件，方便程序员与 Ext.grid.RowSelectionModel 交互。

行选择模型与数据无关，所以我们不能试图通过 Ext.grid.RowSelectionModel 去读取单元格数据。Ext.grid.RowSelectionModel 为我们定义的主要是两个方面的内容：一是提供大量的方法用于选择行，如选择上一行，选择下一行，选择所有行等等；二是获取行的选择信息，如某一行是否被选中、选中的行数等等。

总之，Ext.grid.RowSelectionModel 中只有与行选择相关的信息，其他的不要作任何幻想。数据操作还得由 Store 负责，而界面更新则得由 GridView 出马了。

再次提醒，行选择模型只负责行的选择，与其他无关。

我们来看看 Ext.grid.RowSelectionModel 提供了哪些方法：

- selectFirstRow()：选择第一行；
- selectLastRow(keepExisting)：选择最后一行，参数 keepExisting 为 true 保留原先选择的行，为 false 则清除原来选择的行；
- selectNext(keepExisting)：选择下一行，参数 keepExisting 为 true 保留原先选择的行，为 false 则清除原来选择的行；
- selectPrevious(keepExisting)：选择上一行，参数 keepExisting 为 true 保留原先选择的行，为 false 则清除原来选择的行；
- hasNext()：判断是否还有下一行；
- hasPrevious()：判断是否还有上一行；
- selectAll()：选择所有行；
- isSelected(index)：判断指定索引的行是否被选择
- selectRows(rows, keepExisting)：选择数组 rows 中指定的行（索引），参数 keepExisting 为 true 保留原先选择的行，为 false 则清除原来选择的行；
- selectRange(startRow, endRow, keepExisting)：选择从 startRow 开始到 endRow 结束的行，参数 keepExisting 为 true 保留原先选择的行，为 false 则清除原来选择的行；
- deselectRow(index)：不选指定索引的行；
- deselectRange(startRow, endRow)：不选从 startRow 开始到 endRow 结束的行；

- getCount()：返回选择的行数；

我们通过一个示例来演示上面部分方法的使用，不一定很全面，请举一反三。

```
var grid = new Ext.grid.GridPanel({
    title: "中国公民",
    width: 700,
    autoHeight: true,
    cm: cm,
    store: store,
    renderTo: "a",
    autoExpandColumn: "memo", //自动伸展，占满剩余区域
    buttonAlign: "center",
    buttons: [{
        text: "第一行",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            rsm.selectFirstRow();
        }
    },{
        text: "上一行",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            if(!rsm.hasPrevious()){
                Ext.Msg.alert("警告", "已到达第一行");
            }else{
                rsm.selectPrevious();
            }
        }
    },{
        text: "下一行",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            if(!rsm.hasNext()){
                Ext.Msg.alert("警告", "已到达最后一行");
            }else{
                rsm.selectNext();
            }
        }
    },{
        text: "最后一行",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            rsm.selectLastRow();
        }
    },{
        text: "全选",
```

```
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            rsm.selectAll();
        }
    },{
        text: "全不选",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            rsm.deselectRange(0, grid.getView().getRows().length - 1);
        }
    },{
        text: "反选",
        handler: function(){
            var rsm = grid.getSelectionModel();//得到行选择模型
            for(var i = grid.getView().getRows().length - 1; i >= 0; i --){
                if(rsm.isSelected(i)){
                    rsm.deselectRow(i);
                }else{
                    rsm.selectRow(i, true);//必须保留原来的，否则效果无法实现
                }
            }
        }
    }]
});
```

我们在 GridPanel 的正文放置一排按钮，分别实现第一行、上一行、下一行、最后一行、全选、全不选、反选的效果，在上面的代码中，反选费了一番周折，在做这个功能时，没有考虑到原来的选择行状态是否被保留，导致单击后总是选择第一行，rsm.selectRow(i, true)方法的第二个参数设置为 true 解决了该问题。通过反选这个例子，我们更深刻体会了 keepExisting 的实际意义。

grid.getView().getRows().length 可以获得总行数，和 Ext.grid.RowSelectionModel 的 getCount()方法不同，后者返回的是选择的总行数，而前者是表格中所有的行。

下面是截图：

我们常常希望在每一行添加复选框，通过复选框来选择行，这样就不需要按 ctrl 或 shift 键实现多选了，使用鼠标就能直接操作。项目中的批量删除往往都做成这种效果，在 GridPanel 中，可以轻易实现，我们只要创建一个 Ext.grid.CheckboxSelectionModel 对象即能实现。可以这么认为，Ext.grid.CheckboxSelectionModel 为 GridPanel 添加了一个特殊的列，列中填充的内容不是文字或 HTML 标记，而是一个复选框，并且只占 20 个像素的宽度，禁用排序和列菜单，效果如下图：

下面是实现过程：

1. 创建 Ext.grid.CheckboxSelectionModel 对象；

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

2. 修改列模型，确定复选框放在哪一列；

```
//列模型
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    sm, //放在第二列
    {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
    ……（后面代码在前面出现过，在此省略 N 字）
```

3. 修改 GridPanel 的配置选项，添加一个 sm 的配置并为 Ext.grid.CheckboxSelectionModel()对象。

```
var grid = new Ext.grid.GridPanel({
    title: "中国公民",
    width: 700,
    autoHeight: true,
    cm: cm,
    sm: sm,
    ……（后面代码在前面出现过，在此省略 N 字）
```

顺便说一句，Extjs 中的 checkbox 是图片做的，不是传统的<input type=checkbox/>标记。如果要实现单选，为行选择模型配置 singleSelect: true 选项吧。

二、Grid 视图

grid 视图被定义成 Ext.grid.GridView 类，该类的源码达 1600 行，是一个相对比较复杂的类，grid 视图根据用户提供的参数渲染表格，并能得到原始的 TR 或 TD 信息。

Grid 视图渲染的表格与我们传统的 table 标记存在很大不同，GridPanel 的每一行是一个单独的一行多列的 table，他的行以 table 为单位，而不是以 tr 为单位，table 放在 div 中。

在 Grid 视图中，定义了很多模板，如主体模板、列头模板、行模板等等，最后的效果由这些模板按照特定方式组合而成。我们看看他的主体模板：

```
ts.master = new Ext.Template(
        '<div class="x-grid3" hidefocus="true">',
        '<div class="x-grid3-viewport">',
            '<div class="x-grid3-header"><div class="x-grid3-header-inner"><div class="x-grid3-header-offset">{header}</div></div><div class="x-clear"></div></div>',
            '<div class="x-grid3-scroller"><div class="x-grid3-body">{body}</div><a href="#" class="x-grid3-focus" tabIndex="-1"></a></div>',
        "</div>",
        '<div class="x-grid3-resize-marker"> </div>',
        '<div class="x-grid3-resize-proxy"> </div>',
    "</div>"
);
```

该模板将表格分成两个部分：header 和 body，header 就是列头，body 即为数据行区。我们说每一行都是一个 table，从下面的模板中可见一斑：

```
ts.row = new Ext.Template(
    '<div class="x-grid3-row {alt}" style="{tstyle}"><table class="x-grid3-row-table" border="0" cellspacing="0" cellpadding="0" style="{tstyle}">',
    '<tbody><tr>{cells}</tr>',
    '</tbody></table></div>'
);
```

一行包含若干个列，这些列放在{cells}中，列模板是这样定义的：

```
ts.cell = new Ext.Template(
        '<td class="x-grid3-col x-grid3-cell x-grid3-td-{id} {css}" style="{style}" tabIndex="0" {cellAttr}>',
        '<div class="x-grid3-cell-inner x-grid3-col-{id}" unselectable="on" {attr}>{value}</div>',
        "</td>"
);
```

{value}就是单元格中最后要放的值了。

Ext.grid.GridView 中常用的方法不多，但有些方法特别有用，下面是 Ext.grid.GridView 最重要的方法：

- getRows()：返回所有的行，这些行不是 Extjs 的封装类对象，而是 TR 元素对象，只能通过 dom 访问他的属性；
- getRow(row)：返回指定的行，不是 Extjs 的封装类对象，而是 TR 元素对象，只能通过 dom 访问他的属性；
- getCell(row, col)：返回第 row 行第 col 列的单元格，返回的单元格不是 Extjs 的封装类对象，而是 TD 元素对象，只能通过 dom 访问他的属性；
- refresh(headersToo)：刷新，如果 headersToo 为 true，则表头也一起刷新；

在实际应用中，可能会获取被选行的某一列的值，或者动态添加行和删除行，这些操作光操 Ext.grid.GridView 无法完成，不管是获取值，还是动态操作行，都和 Ext.data.Store 有关系，我们只要操作 Ext.data.Store 中的数据就行了，数据一改变，调用 Ext.grid.GridView 类的 refresh()方法刷新界面即可，refresh()的另一个作用是能对序号重新编号。

Ø 读取被选行的值

Store 由结构和数据构成，每一行都是一个 Ext.data.Record，Record 的每一个值都由 name 标识，通过 name 属性值可以读取出指定 Record 的某一列的值。先看下面的代码：

```
grid.getStore().getAt(i).get("Memo");
```

grid.getStore()获取 GridPanel 的 Store，getAt(i)获取第 i 个 Record，get("Memo")获取 Record 的 Memo 的值。应该还记得，我们曾经这样定义过 Record：

```
var Human = Ext.data.Record.create([
    {name: "Name", type: "string", mapping: "name"},
    {name: "Sex", type: "string", mapping: "sex"},
    {name: "Birthday", type: "string", mapping: "birthday"},
    {name: "Education", type: "string", mapping: "edu"},
    {name: "Memo", type: "string", mapping: "memo"}
]);
```

下面的示例将打印出所有被选行的姓名：

```
var view = grid.getView();
var rsm = grid.getSelectionModel();//得到行选择模型
var r = "";
for(var i = 0; i < view.getRows().length; i ++){
    if(rsm.isSelected(i)){//找到选中的行
        r += grid.getStore().getAt(i).get("Name") + "<br>";
    }
```

```
}

Ext.MessageBox.alert("选择", "您选择的数据有：<br>" + r);
```

Ø 增加新行

我们可以把新行添加到任何位置，这需要调用 Ext.data.Store 中定义的方法：

- add(records)：将记录添加到行尾，可以一次添加多行，records 为 Ext.data.Record[]类型的参数；
- insert(index, records)：将记录添加到指定索引处，可以一次添加多行，records 为 Ext.data.Record[]类型的参数；

本章示例中，我们定义了一个描述人类的结构 Human，以 Human 为基础，我们创建数据：

```
var obj = {
    Name: "新人物",
    Sex: "女",
    Birthday: Date.parseDate("1980-05-12", "Y-m-d"),
    Education: "本科",
    Memo: "是新的"
};
```

这是一个 json 对象，也可以是数组，由 Reader 的类型决定，json 对象中的属性名必须和 Human 的 name 属性值一致，这是一种映射关系。

我们将 obj 与 Human 合并：

```
    var human = new Human(obj);
```

将 human 添加到 Store 中就可以了。最后，刷新 GridView，让界面重新显示，GridView 会重新从 Store 读取数据并更新界面。下面是本操作完整的源代码：

```
var view = grid.getView();
var obj = {
    Name: "新新人类",
    Sex: "男",
    Birthday: Date.parseDate("2009-05-16", "Y-m-d"),
    Education: "本科",
    Memo: "我是新新人类，所向无敌！"
};
var human = new Human(obj);
grid.getStore().insert(0, human);//添加到行首，修改第一个参数可以添加到任何位置
view.refresh();
```

如果要添加到行尾，使用下面的语句，其他代码完全相同：

```
grid.getStore().add(human);
```

Ø 删除选择行

删除行的思路和添加行大同小异，基本实现步骤如下：

1. 循环遍历所有行
2. 判断当前行是否被选中
3. 删除选中的行

删除行同样要使用 Ext.data.Store 的方法 remove()，如果是逐行删除，则应该遵循从后往前的方向，remove()这样定义：

```
remove (record)：删除指定的记录，record 参数为 Ext.data.Record 类型对象
```

另外，如果要删除所有行，可以调用 removeAll()方法。

看看完整的源代码：

```
{
    text: "删除选定行",
    icon: "../extjs/resources/images/default/dd/drop-no.gif",
    cls: "x-btn-text-icon",
    handler: function(){
        var rsm = grid.getSelectionModel();//得到行选择模型
        var view = grid.getView();
        var store = grid.getStore();
        for(var i = view.getRows().length - 1; i >= 0 ; i --){
            if(rsm.isSelected(i)){
                store.remove(store.getAt(i));
            }
        }
        view.refresh();
    }
}
```

删除所有行则简单了很多：

```
var store = grid.getStore();
var view = grid.getView();
store.removeAll();
view.refresh();
```

以下是本小节的效果图：

三、小结

至此，我们已经学习了三个非常重要的类：Ext.grid.ColumnModel、Ext.grid.RowSelectionModel 和 Ext.grid.GridView，这三个类分工明确，职责清晰，是我们在设计的时候一个很好的借鉴和典范。

Ext.grid.ColumnModel 定义表头，包含了列的所有信息，并提供丰富的高级功能，如排序、菜单、列是否被隐藏等等。默认情况下，排序被禁用，需要将 sortable 设置为 true 手动打开。

Ext.grid.RowSelectionModel 负责行的选择，通过他我们可以实现任何的选择操作，相对于 Table，他的功能强大太多了。

Ext.grid.GridView 则完成表格的渲染，我们能看到那么漂亮的表格，完全是他的功劳。

另外，Ext.data.Store 也功不可没，数据操作最终还得通过他来实现，该类负责更新数据，Ext.grid.GridView 负责显示数据，Ext.grid.RowSelectionModel 负责选择数据。

第二十一章：GridPanel 分页

一、跑跑题——JSON-LIB

在讲分页之前，让我们走走神，熟悉一个 java 对象与 json 对象的转换工具：json-lib。

json-lib 提供了一种快捷的在 java 对象与 json 对象之间进行相互转换的方法，使用非常简单，没有复杂的配置，也没有太多的 API，支持 javabean、List、Map 等常用对象，甚至支持这些对象的嵌套。

在前面的章节中，我一直在拼凑 json 格式的字符串，不仅麻烦，还容易出错，json-lib 能很好的解决这个问题。

您可以去网上下载 json-lib 包，但是还需要其他包的支持，完整的包应该像下面这样：

json-lib 中主要的 API 有：

- JSONObject：实现 JavaBean 和 json 对象之间的相互转换，基本用法有：
  1. JSONObject jObject = JSONObject.fromObject(javaBean); 将 javaBean 转换成 json 对象
  2. Object javaBean = JSONObject.toBean(jsonObject);将 json 对象转换成 javaBean
- JSONArray：实现 java 集合与 json 对象之间的相互转换，基本用法有：
  1. JSONArray jArray = JSONArray.fromObject(java 集合);将 java 集合转换成 json 对象
  2. Object array = JSONArray.toArray(jsonArray)：将 json 对象数组转换为 java 数组
  3. Object array = JSONArray.toList(jsonArray)：将 json 对象数组转换为 List

根据第 12 章的理论，实现分页我们应该返回形如下面格式的 json 字符串：

```
{"root":
    [
        {"birthday":"2009-05-17","edu":"博士","hid":0,"memo":"这是一行测试数据","name":"无名氏 0","sex":"男"},
        {"birthday":"2009-05-17","edu":"博士","hid":1,"memo":"这是一行测试数据","name":"无名氏 1","sex":"男"}
    ],
```

```
"totalProperty":86}
```

这段字符串保存了查询数据与分页信息，其中，totalProperty 是本次分页的总记录条数，root 是一个 json 对象数组，表示本次查询的结果，请注意他的格式。

我们定义一个 Human 类来保存人的信息：

```
package com.aptech.ajax2;
import java.util.Date;
public class Human {
    private int hid;    //编号
    private String name;//姓名
    private String sex;//性别
    private Date birthday;//出生年月
    private String edu;//学历
    private String memo;//备注

    public Human() {
    }

    public Human(int hid, String name, String sex, Date birthday, String edu,
            String memo) {
        super();
        this.hid = hid;
        this.name = name;
        this.sex = sex;
        this.birthday = birthday;
        this.edu = edu;
        this.memo = memo;
    }

    public int getHid() {
        return hid;
    }

    public void setHid(int hid) {
        this.hid = hid;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public String getEdu() {
        return edu;
    }

    public void setEdu(String edu) {
        this.edu = edu;
    }

    public String getMemo() {
        return memo;
    }

    public void setMemo(String memo) {
        this.memo = memo;
    }
}
```

可以看到，Human 的属性与我们要输出的 json 对象的属性是一致的，json-lib 会根据 Human 的属性名来定义 json 对象的属性名。因为我们返回的是多个 Human 的集合，因此，使用 List 保存是个不错的主意，于是有了下面的代码：

```
// 本 Servlet 为表格分页准备数据
List<Human> list = new ArrayList<Human>();
for (int i = start; i < end; i++) {
    list.add(new Human(i, "无名氏" + i, "男", new Date(), "博士", "这是一行测试数据"));
```

```
}
```

变量 start 和 end 是分页信息，根据实际情况会有所变动。

接下来定义 PageBean 类，封装 totalProperty 和 root 属性，其中，root 是上面的 List 集合：

```
package com.aptech.ajax2;

import java.util.List;

public class PageBean {
    private int totalProperty;
    private List root;

    public PageBean() {
        // TODO Auto-generated constructor stub
    }

    public PageBean(int totalProperty, List root) {
        super();
        this.totalProperty = totalProperty;
        this.root = root;
    }

    public int getTotalProperty() {
        return totalProperty;
    }

    public void setTotalProperty(int totalProperty) {
        this.totalProperty = totalProperty;
    }

    public List getRoot() {
        return root;
    }

    public void setRoot(List root) {
        this.root = root;
    }

}
```

实例化该对象，并且赋值：

```
int totalProperty = 86;
PageBean pageBean = new PageBean(totalProperty, list);
```

现在，所有的信息都封装在 pageBean 对象中了。为了生成 json 对象，使用下面的代码转换即可：

```
JSONObject jObject = JSONObject.fromObject(pageBean, JsonUtil.configJson("yyyy-MM-dd"));
out.println(jObject.toString());
```

最终，我们获得了想要的结果，通过创建普通的 java 对象，再转换成 json 格式，和过去相比，实在有了不小的进步。我们只要维护一系列的 java 对象就行了，不再进行频繁的字符串拼接，后者通过拼接生成 json 不仅容易出错，还不容易维护，并且破坏了代码风格，怎么看怎么不爽。

json-lib 对 java.util.Date 的输出非常奇怪，似乎没作任何处理，执行 System.out.println(JSONObject.fromObject(new java.util.Date())); 代码后，输出结果是 {"date":17,"day":0,"hours":17,"minutes":41,"month":4,"seconds":9,"time":1242553269015,"timezoneOffset":-480,"year":109}，显然，这样无法满足客户端的要求，加重了对日期解析的负担。而对于 java.sql.Date、java.sql.Time 和 java.sql.Timestamp，则根本不支持，会抛出 java.lang.IllegalArgumentException 异常。

为了得到正常的日期信息，在 fromObject 方法中必须指定一个 JsonConfig 的参数，我们通过一个静态方法返回该对象：

```
package com.aptech.util;

import java.util.Date;

import net.sf.json.JsonConfig;
import net.sf.json.util.CycleDetectionStrategy;

public class JsonUtil {
    public static JsonConfig configJson(String datePattern) {
        JsonConfig jsonConfig = new JsonConfig();
        jsonConfig.setExcludes(new String[] { "" });
        jsonConfig.setIgnoreDefaultExcludes(false);
        jsonConfig.setCycleDetectionStrategy(CycleDetectionStrategy.LENIENT);
        jsonConfig.registerJsonValueProcessor(Date.class,
                new JsonDateValueProcessor(datePattern));

        return jsonConfig;
    }
```

```
}
```

configJson()方法的参数 datePattern 用于指定日期的输出格式，符合 java 中的定义标准，如 yyyy-MM-dd HH-mm-dd 。 configJson() 内 使 用 了 JsonDateValueProcessor 对 象 ，JsonDateValueProcessor 用于处理日期：

```
package com.aptech.util;

import java.text.SimpleDateFormat;
import java.util.Date;

import net.sf.json.JsonConfig;
import net.sf.json.processors.JsonValueProcessor;

/**
 * 本代码来自网上，感谢！
 *
 */
public class JsonDateValueProcessor implements JsonValueProcessor {
    private String format = "yyyy-MM-dd HH:mm:ss";
    public JsonDateValueProcessor() {
    }
    public JsonDateValueProcessor(String format) {
        this.format = format;
    }
    public Object processArrayValue(Object value, JsonConfig jsonConfig) {
        String[] obj = {};
        if (value instanceof Date[]) {
            SimpleDateFormat sf = new SimpleDateFormat(format);
            Date[] dates = (Date[]) value;
            obj = new String[dates.length];
            for (int i = 0; i < dates.length; i++) {
                obj[i] = sf.format(dates[i]);
            }
        }
        return obj;
    }
    public Object processObjectValue(String key, Object value,
            JsonConfig jsonConfig) {
        if (value instanceof Date) {
            String str = new SimpleDateFormat(format).format((Date) value);
            return str;
        }
        return value == null ? null : value.toString();
```

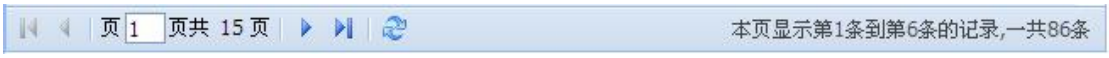
```
    }
    public String getFormat() {
        return format;
    }
    public void setFormat(String format) {
        this.format = format;
    }
}
```

JsonDateValueProcessor 类为 json-lib 不认识的数据类型提供一种处理办法，必须实现 JsonValueProcessor 接口，processArrayValue()处理数组，processObjectValue()处理单个对象，这两个方法被 json-lib 回调，我们负责提供解决方案。

如果要处理 java.sql.Date、java.sql.Time 和 java.sql.Timestamp，仿照上面的例子扩展 JsonValueProcessor 接口，这真是一件简单的工作。

二、分页工具栏

Extjs 为我们提供了一个非常棒的分页工具栏，并且显示的信息很丰富。内置第一页、上一页、下一页、最后一页、刷新当前页等按钮，支持直接输入页码，并显示当前页面的信息。下面是一个截图：



很漂亮吧，分页工具栏通过 Ext.PagingToolbar 类实现。下面是基本的用法：

```
new Ext.PagingToolbar({
    store: store, //这个不用介绍了吧？
    pageSize: 6,//页大小
    displayInfo: true,//是否显示 displayMsg
    displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
    emptyMsg: "没有记录"//如果没有记录，则显示该文本
})
```

Panel 可以在顶部和底部各放置一个工具栏，tbar 表示顶部工具栏，bbar 表示底部工具栏，我们可以根据需要放置在任一位置。

三、分页

前面将理论基础讲完了，这一节我们没有太多的新内容，下面的 Servlet 用来处理客户端的分页请求，您一定会感觉似曾相识。

```
package com.aptech.ajax2;
……省略了部分包
import net.sf.json.JSONObject;
import com.aptech.util.JsonUtil;
public class PageServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        // 获取参数
        int start = Integer.parseInt(request.getParameter("start"));
        int limit = Integer.parseInt(request.getParameter("limit"));
        int totalProperty = 86;
        int end = start + limit;
        if(end > totalProperty) end = totalProperty;
        // 本 Servlet 为表格分页准备数据
        List<Human> list = new ArrayList<Human>();
        for (int i = start; i < end; i++) {
            list.add(new Human(i, "无名氏" + i, "男", new Date(), "博士", "这是一行测
试数据"));
        }

        //定义符合分页所需要的格式
        PageBean pageBean = new PageBean(totalProperty, list);
        JSONObject jObject = JSONObject.fromObject(pageBean,
JsonUtil.configJson("yyyy-MM-dd"));
        out.println(jObject.toString());
        out.flush();
        out.close();
    }
}
```

得到数据：

```
//Proxy
var proxy = new Ext.data.HttpProxy({url: "../PageServlet"});

//Record 定义记录结果
var Human = Ext.data.Record.create([
    {name: "Hid", type: "int", mapping: "hid"},
    {name: "Name", type: "string", mapping: "name"},
    {name: "Sex", type: "string", mapping: "sex"},
    {name: "Birthday", type: "string", mapping: "birthday"},
    {name: "Education", type: "string", mapping: "edu"},
```

```
    {name: "Memo", type: "string", mapping: "memo"}
]);

//Reader
var reader = new Ext.data.JsonReader(
    {totalProperty: "totalProperty", root: "root"},
    Human
);

//Store
var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader
});

store.load({params: {start: 0, limit: 6}});
```

我没打算再重复解释这段代码，如果不理解，请看第 10-12 章的内容。要注意的是 store.load()方法，该方法和以前不一样，多了一个 json 参数，该参数保存了第一次加载的分页信息，从第二次开始，start 的值被自动计算，属性 params 则是一个 json 对象，start 表示从哪一个索引开始查找，limit 表示查找多少个记录。

在 PageServlet 中，我们通过下面两个语句获取这两个参数的值：

```
        int start = Integer.parseInt(request.getParameter("start"));
        int limit = Integer.parseInt(request.getParameter("limit"));
```

别小看他们，他们可是我们分页最重要的数据。

按照国际惯例，我们创建列模型，告诉 GridPanel 该如何显示数据。

```
//列模型
var cm = new Ext.grid.ColumnModel([
    {header: "ID", width: 40, dataIndex: "Hid"},
    {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
    {header: "性别", width: 40, dataIndex: "Sex", align: "center"},
    {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
    {header: "学历", width: 80, dataIndex: "Education", align: "center"},
    {id: "memo", header: "备注", dataIndex: "Memo"}
]);
```

最后，主角登场了，开始创建 GridPanel：

```
var grid = new Ext.grid.GridPanel({
```

```
    title: "中国公民",
    width: 650,
    autoHeight: true,
    cm: cm,
    store: store,
    renderTo: "a",
    autoExpandColumn: "memo", //自动伸展，占满剩余区域
    bbar:
    new Ext.PagingToolbar({
        store: store,
        pageSize: 6,
        displayInfo: true,
        displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
        emptyMsg: "没有记录"
    })
});
```

看出来了吗？分页工具栏被我放到了哪里？别告诉我你不知道。最后一起来看看效果吧！

四、小结

分页在 extjs 中不是一个很复杂的技术，可以说将分页的难度降到了最底，并且一律采用 ajax，不得不感叹，ajax 真是个好东西。extjs 将这个技术发挥到了极致。

如果您不喜欢使用乱七八糟的开源组件，依旧可以自己拼接 json，其实只要找到了恰当的方法，代码依旧优雅。不过我还是推荐您能主动接受新鲜事物，我们应该对任何好的东西都充满好奇并为已所用，您说呢？

当我们大量用到 json 的时候，使用 json-lib 吧，因为他的简洁，因为我们的惰性。json-lib 是一位忠诚的魔法师，至始至终兢兢业业一心一意为我们服务。

第二十二章：GridPanel 扩展

一、学会自学吧，朋友

这一章的内容非常有意思，因为我要带大家实现一些更加有用的功能，这些功能并非我的独创，我想我没有足够的想象力达到 extjs 团队的水平，我想到的他们早已想到，我不曾想到的他们也帮我们想到了。所以，本章的示例基本上来自 extjs 自带的示例，不过，我可能会做少许修改。

但是，在这一章，我无意再使用详细的讲解去分析各个语句的含义，只是做一些辅助说明，您必须亲自去分析和调试，最终将效果实现。希望您的自学能力能助您腾飞。

千万不要以为我在偷懒，留点想象空间给您，这也是十分必要的，是吗？

二、带摘要的 GridPanel

“带摘要的 GridPanel”，如果我不说明白，您一定会更糊涂，不过事情没有想象的复杂，下面的图会告诉您所有的真相：

标题中的“摘要”在图中变成了“备注”，哦，这并不重要，一种表达而已。重要的是和上一章的示例比较，备注不再以列的形式出现，而是单独成行，并用一个方框包围起来。不知您注意到没有，分页工具栏上出现了一个新的按钮，现在您至少明白，原来分页工具栏

上的按钮是可以自定义的。

需要提醒的是，该示例服务器端的代码依然是上一章的代码，没作任何修改，如果非要说修改了的话，就是备注内容被我复制了 7 次，别的我发誓绝对没有改变。

在 GridPanel 中，每一行都有一个默认被隐藏的 RowBody，该如何翻译 RowBody 呢？大概就是一行的主体部分的意思。我们将在 RowBody 中放上备注，所以必须先启用 RowBody。RowBody 由 GridView 来管理，启用的配置自然得交给他，但是 GridView 并不需要实例化，我们无法控制，所以配置 GridView 的工作就只能委托给 GridPanel 了。GridPanel 有一个叫 viewConfig 的选项，该选项正是 GridView 的配置场所。下面的代码启用了 RowBody：

```
viewConfig: {
    forceFit:true,
    enableRowBody:true,
    showPreview:true,
    getRowClass : function(record, rowIndex, p, store){
        if(this.showPreview){
            p.body = '<p style="padding:5px;border:1px #DFE8F6 solid;margin:2px;"><span style="color:#15428B; font-weight:bold;">备注：</span>'+record.data.Memo+'</p>';
            return 'x-grid3-row-expanded';
        }
        return 'x-grid3-row-collapsed';
    }
}
```

forceFit 属性是我一直避讳的问题，因为在前面任何一个地方解释都觉得有点不合时宜。现在，我将给出明确答案：forceFit 用于按照一定比例拉伸列的宽度，填满整个 GridPanel。在 extjs2 中，这个选项有了很大的改进，用起来很舒服。不过，老实说，我更喜欢 autoExpandColumn。

enableRowBody 就是启用 RowBody 的选项，这个选项很关键。

showPreview 并不是 GridView 的属性，我在源代码中没找到，而且我相信 GridView 并不需要 showPreview，这里是一个特例。showPreview 能控制摘要内容是否显示还是隐藏。

getRowClass()方法用来设置 RowBody 的内容，我加了些样式，显示了一个方框，看起来似乎很不错。

自定义的分页工具栏也是我们关心的重点，下面的代码看起来很容易理解：

```
new Ext.PagingToolbar({
    store: store,
```

```
    pageSize: 6,
    displayInfo: true,
    displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
    emptyMsg: "没有记录",
    items:["-",{
        icon: "../extjs/resources/images/default/dd/drop-yes.gif",//图标
        cls: "x-btn-text-icon",//按钮文字前为放图标留的一段空隙
        pressed: true,
          enableToggle:true,
          text: '隐藏备注',
          cls: 'x-btn-text-icon details',
          toggleHandler: function(btn, pressed){
                var view = grid.getView();
                if(pressed){
                    btn.setText("隐藏备注");
                }else{
                    btn.setText("显示备注");
                }
                view.showPreview = pressed;
                view.refresh();
          }
    }]
})
```

enableToggle 和 toggleHandler()是相辅相成的，enableToggle 决定是否启用状态切换功能，而 toggleHandler()是状态切换后的处理函数，在 toggleHandler()方法中，我们根据按钮是否被按下修改了显示文本，这样貌似更人性化。

下面给出 js 的完整源代码，服务器代码请参考上一章：

```
Ext.onReady(function(){
    //准备数据

    //Proxy
    var proxy = new Ext.data.HttpProxy({url: "../PageServlet"});

    //Record 定义记录结果
    var Human = Ext.data.Record.create([
        {name: "Hid", type: "int", mapping: "hid"},
        {name: "Name", type: "string", mapping: "name"},
        {name: "Sex", type: "string", mapping: "sex"},
        {name: "Birthday", type: "string", mapping: "birthday"},
        {name: "Education", type: "string", mapping: "edu"},
        {name: "Memo", type: "string", mapping: "memo"}
```

```
    ]);

    //Reader
    var reader = new Ext.data.JsonReader(
        {totalProperty: "totalProperty", root: "root"},
        Human
    );

    //Store
    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader
    });

    store.load({params: {start: 0, limit: 6}});

    //列模型
    var cm = new Ext.grid.ColumnModel([
        {header: "ID", width: 40, dataIndex: "Hid"},
        {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
        {header: "性别", width: 40, dataIndex: "Sex", align: "center"},
        {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
        {id: "edu", header: "学历", width: 80, dataIndex: "Education", align: "center"}
    ]);

    var grid = new Ext.grid.GridPanel({
        title: "中国公民",
        width: 650,
        autoHeight: true,
        cm: cm,
        store: store,
        renderTo: "a",
        autoExpandColumn: "edu", //自动伸展，占满剩余区域
        viewConfig: {
            forceFit:true,
            enableRowBody:true,
            showPreview:true,
            getRowClass : function(record, rowIndex, p, store){
                if(this.showPreview){
                    p.body = '<p style="padding:5px;border:1px #DFE8F6
solid;margin:2px;"><span style="color:#15428B; font-weight:bold;"> 备 注 ：
</span>'+record.data.Memo+'</p>';
                    return 'x-grid3-row-expanded';
                }
```

```
                return 'x-grid3-row-collapsed';
            }
        },
        bbar:
        new Ext.PagingToolbar({
            store: store,
            pageSize: 6,
            displayInfo: true,
            displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
            emptyMsg: "没有记录",
            items:["-",{
                icon: "../extjs/resources/images/default/dd/drop-yes.gif",
                cls: "x-btn-text-icon",
                pressed: true,
                enableToggle:true,
                text: '隐藏备注',
                cls: 'x-btn-text-icon details',
                toggleHandler: function(btn, pressed){
                    var view = grid.getView();
                    if(pressed){
                        btn.setText("隐藏备注");
                    }else{
                        btn.setText("显示备注");
                    }
                    view.showPreview = pressed;
                    view.refresh();
                }
            }]
        })
    });

});
```

三、RowExpander

我将 RowExpander 翻译成为“行扩展器”，这不是什么高深玩意儿，网上论坛随处可见，但我依旧认为这是一个不错的创意。RowExpander 就是在一行的某一列添加一个“+”按钮，单击该按钮后，显示 RowBody，按钮变成“-”，我们可以随意切换。先来看一下效果图：

虽然我说+按钮可以放在任意列，但我强烈建议您放在第一列，放在其他列会出现意想不到的效果——那不是我们期望的。

事实上，这个功能并不是 Extjs 原生态的，而是 Extjs 自带的示例程序中的一个小插件。该插件位于目录<Extjs/examples/grid>下，名为 RowExpander.js，这个文件定义了一个名为 RowExpander 的类，该类实现的就是“行扩展器”的功能。

RowExpander 没有太多的选项可以设置，但在展开和收缩的时候，分别会触发 expand 和 collapse 事件，参数列表为（this, record, body, row.rowIndex），或许您用得着。

RowExpander 的使用比较简洁，我将贴上代码，注意看粗体部分：

```
Ext.onReady(function(){
    //准备数据
    //Proxy
    var proxy = new Ext.data.HttpProxy({url: "../PageServlet"});

    //Record  定义记录结果
    var Human = Ext.data.Record.create([
            {name: "Hid", type: "int", mapping: "hid"},
            {name: "Name", type: "string", mapping: "name"},
            {name: "Sex", type: "string", mapping: "sex"},
            {name: "Birthday", type: "string", mapping: "birthday"},
            {name: "Education", type: "string", mapping: "edu"},
            {name: "Memo", type: "string", mapping: "memo"}
    ]);

    //Reader
    var reader = new Ext.data.JsonReader(
```

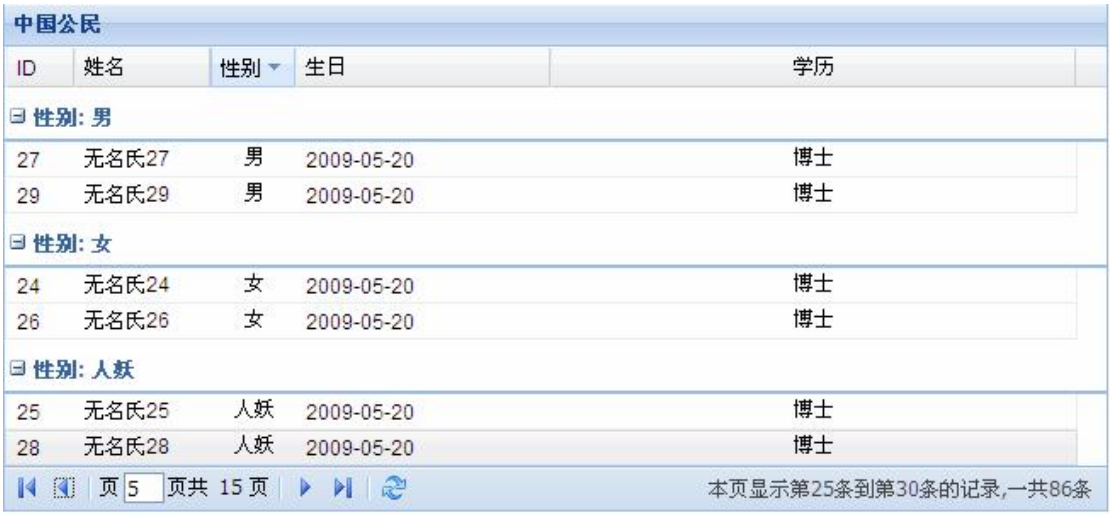
```
        {totalProperty: "totalProperty", root: "root"},
        Human
    );

    //Store
    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader
    });

    store.load({params: {start: 0, limit: 6}});

    var expander = new Ext.grid.RowExpander({
            tpl: new Ext.Template("<p style='padding:5px;border:1px #DFE8F6 solid;margin:2px;'><span style='color:#15428B; font-weight:bold;'> 备 注 ：</span>{Memo}</p>")
        }
    );

    //列模型
    var cm = new Ext.grid.ColumnModel([
        expander,
        {header: "ID", width: 40, dataIndex: "Hid"},
        {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
        {header: "性别", width: 40, dataIndex: "Sex", align: "center"},
        {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
        {id: "edu", header: "学历", width: 80, dataIndex: "Education", align: "center"}
    ]);

    var grid = new Ext.grid.GridPanel({
        title: "中国公民",
        width: 650,
        autoHeight: true,
        cm: cm,
        store: store,
        renderTo: "a",
        plugins: [expander],
        autoExpandColumn: "edu", //自动伸展，占满剩余区域
        bbar:
        new Ext.PagingToolbar({
            store: store,
            pageSize: 6,
            displayInfo: true,
            displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
```

```
                emptyMsg: "没有记录"
            })
    });

});
```

实例化 RowExpander 时，我们定义了一个填充 RowBody 的模板，模板中访问了 Record 的 Memo 值，定义列模型时，将 RowExpander 的实例设置成了一个新列，最后，创建 Ext.grid.GridPanel 时，增加了 plugins: [expander]配置选项。

四、分组 GridPanel

分组 GridPanel 也很实用，他能将指定字段进行分组，将同一组记录显示在一起。效果如下图所示：



上图中，我们将性别分组，本示例设定了三种性别：男、女和人妖。为了配合该效果，PageServlet 修改如下（粗体为修改部分）：

```
package com.aptech.ajax2;
……导包语句省略
public class PageServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        // 获取参数
        int start = Integer.parseInt(request.getParameter("start"));
        int limit = Integer.parseInt(request.getParameter("limit"));
        int totalProperty = 86;
        int end = start + limit;
```

```
        if(end > totalProperty) end = totalProperty;
        // 本 Servlet 为表格分页准备数据
        List<Human> list = new ArrayList<Human>();
        Random rnd = new Random();
        for (int i = start; i < end; i++) {
            int sexId = rnd.nextInt(10);
            list.add(new Human(i, "无名氏" + i, sexId % 2 == 0 ? "男" : sexId % 3 == 0 ? "女" : "人妖", new Date(), "博士", "这是一行测试数据这是一行测试数据这是一行测试数据这是一行测试数据这是一行测试数据这是一行测试数据这是一行测试数据"));
        }

        //定义符合分页所需要的格式
        PageBean pageBean = new PageBean(totalProperty, list);
        JSONObject jObject = JSONObject.fromObject(pageBean, JsonUtil.configJson("yyyy-MM-dd"));

        out.println(jObject.toString());

        out.flush();
        out.close();
    }
}
```

有两个类用来支持分组，一个是继承自 Ext.data. Store 的 Ext.data.GroupingStore，另一个是继承自 Ext.grid.GridView 的 Ext.grid.GroupingView，前者支持数据数组，后者支持界面效果分组。

下面的源码基本是重复的，修改部分用粗体标出，分组立即出现：

```
Ext.onReady(function(){
    var proxy = new Ext.data.HttpProxy({url: "../PageServlet"});
    //Record 定义记录结果
    var Human = Ext.data.Record.create([
        {name: "Hid", type: "int", mapping: "hid"},
        {name: "Name", type: "string", mapping: "name"},
        {name: "Sex", type: "string", mapping: "sex"},
        {name: "Birthday", type: "string", mapping: "birthday"},
        {name: "Education", type: "string", mapping: "edu"},
        {name: "Memo", type: "string", mapping: "memo"}
    ]);

    //Reader
    var reader = new Ext.data.JsonReader(
```

```
        {totalProperty: "totalProperty", root: "root"},
        Human
    );

    var groupStore = new Ext.data.GroupingStore({
        proxy: proxy,
        reader: reader,
        groupField: "Sex",
        sortInfo: {field: "Sex", direction: "DESC"}
    });

    groupStore.load({params: {start: 0, limit: 6}});

    //列模型
    var cm = new Ext.grid.ColumnModel([
        {header: "ID", width: 40, dataIndex: "Hid"},
        {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
        {header: "性别", width: 50, dataIndex: "Sex", align: "center"},
        {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
        {id: "edu", header: "学历", width: 80, dataIndex: "Education", align: "center"}
    ]);

    var grid = new Ext.grid.GridPanel({
        title: "中国公民",
        width: 650,
        autoHeight: true,
        cm: cm,
        store: groupStore,
        view: new Ext.grid.GroupingView(),//使用继承自 GridView 的分组视图
        renderTo: "a",
        autoExpandColumn: "edu", //自动伸展，占满剩余区域
        bbar:
        new Ext.PagingToolbar({
            store: groupStore,
            pageSize: 6,
            displayInfo: true,
            displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
            emptyMsg: "没有记录"
        })
    });

});
```

GroupingStore 中的 groupField 是分组字段，对应 Record 中的 name 值，sortInfo 为排序

信息，ASC 表示升序，DESC 表示降序。

Ext.grid.GridPanel 的 view 配置选项在默认情况下为 GridView，为了配合分组，在这里被 Ext.grid.GroupingView 替换。

五、将带摘要的 GridPanel 和分组 GridPanel 合二为一

本节的效果图如下所示：

GridPanel 的 view 默认为 GridView，用于渲染表格效果，由于 GridView 不是我们自己实例化，所以无法对其控制，因此 GridPanel 提供了一个名为 viewConfig 的配置，用于为 GridView 提供配置信息。

但是，在分组的 GridPanel 中，view 被 GroupingView 替换了，我们创建了一个 GroupingView 对象，因为该对象被我们显示创建，所以可以直接将选项配置交给 GroupingView。您可以比较第 2 节与本节的区别，注意粗体部分：

```
Ext.onReady(function(){
    //准备数据
```

```
//Proxy
var proxy = new Ext.data.HttpProxy({url: "../PageServlet"});

//Record 定义记录结果
var Human = Ext.data.Record.create([
    {name: "Hid", type: "int", mapping: "hid"},
    {name: "Name", type: "string", mapping: "name"},
    {name: "Sex", type: "string", mapping: "sex"},
    {name: "Birthday", type: "string", mapping: "birthday"},
    {name: "Education", type: "string", mapping: "edu"},
    {name: "Memo", type: "string", mapping: "memo"}
]);

//Reader
var reader = new Ext.data.JsonReader(
    {totalProperty: "totalProperty", root: "root"},
    Human
);

var groupStore = new Ext.data.GroupingStore({
    proxy: proxy,
    reader: reader,
    groupField: "Sex",
    sortInfo: {field: "Sex", direction: "DESC"}
});

groupStore.load({params: {start: 0, limit: 6}});

//列模型
var cm = new Ext.grid.ColumnModel([
    {header: "ID", width: 40, dataIndex: "Hid"},
    {header: "姓名", width: 80, dataIndex: "Name", tooltip: "这是您的姓名"},
    {header: "性别", width: 50, dataIndex: "Sex", align: "center"},
    {header: "生日", width: 150, format: "Y-m-d", dataIndex: "Birthday"},
    {id: "edu", header: "学历", width: 80, dataIndex: "Education", align: "center"}
]);

var grid = new Ext.grid.GridPanel({
    title: "中国公民",
    width: 650,
    autoHeight: true,
    cm: cm,
    store: groupStore,
    view: new Ext.grid.GroupingView({
```

```
            forceFit:true,
            enableRowBody:true,
            showPreview:true,
            getRowClass : function(record, rowIndex, p, store){
                if(this.showPreview){
                    p.body = '<p style="padding:5px;border:1px #DFE8F6 solid;margin:2px;"><span style="color:#15428B; font-weight:bold;">备 注 ：</span>'+record.data.Memo+'</p>';
                    return 'x-grid3-row-expanded';
                }
                return 'x-grid3-row-collapsed';
            }
        }),//使用继承自 GridView 的分组视图
        renderTo: "a",
        autoExpandColumn: "edu", //自动伸展，占满剩余区域
        bbar:
        new Ext.PagingToolbar({
            store: groupStore,
            pageSize: 6,
            displayInfo: true,
            displayMsg: "本页显示第{0}条到第{1}条的记录,一共{2}条",
            emptyMsg: "没有记录",
            items:["-",{
                icon: "../extjs/resources/images/default/dd/drop-yes.gif",
                cls: "x-btn-text-icon",
                pressed: true,
                enableToggle:true,
                text: '隐藏备注',
                cls: 'x-btn-text-icon details',
                toggleHandler: function(btn, pressed){
                    var view = grid.getView();
                    if(pressed){
                        btn.setText("隐藏备注");
                    }else{
                        btn.setText("显示备注");
                    }
                    view.showPreview = pressed;
                    view.refresh();
                }
            }]
        })
    });

});
```

六、小结

本章没有太多的描述，但实现了一些非常实用的功能，稍做修改便可应用到项目中。使用 GridPanel 时，涉及到很多的类，多去了解这些类之间的关系，对于理解整个体系结构大有好处。

好啦，不多说，继续下一章。

第二十三章：可编辑的 GridPanel——EditGridPanel

一、EditGridPanel

Ext.grid. EditGridPanel 是 Ext.grid.GridPanel 的子类，不同的是，EditGridPanel 的功能得到了进一步增强，能对表格内的单元格内容进行修改，配合第 20 章的添加行和删除行功能，我们将具备对表格实现任何操作的能力。

这真是一个令人兴奋的功能，为了写好本章的例子，我花了整整 5 个小时，我在琢磨如何才能让本章的例子尽善尽美，尽可能减少您的学习时间，最终，我做到了。虽然参考了网上的一些示例，但我更多的是针对实际项目的需求，不一定是最好的，但至少提供了一个完整的解决方案，不是吗？

Ext.grid.EditorGridPanel 和 GridPanel 一样，本身的使用是简单的，更多的时候，我们都是在驾驭他之前做着无数的准备——渲染前的数据准备、操作后的事件处理准备、与数据库服务器的交互准备。这些，都是我们必须面对的。

从名字就可以知道，Ext.grid.EditorGridPanel 是一个具有编辑功能的表格面板，曾经以为，这个功能实现是如何的艰难，但是，幸好有 Jack，幸好有 Extjs，这些天才们将这一功能作了最优雅的封装，我们可以在单元格中放置任何组件，并自动更新单元格的内容，使用 ajax 还能将修改后的数据上传到服务器，与数据库同步。有了这些，还有什么不能满足的呢？

我们来了解一下 EditorGridPanel 的工作过程：

1、用户点击单元格

2、单元格按照预设的组件显示单元格的内容并处于编辑状态

3、离开单元格的编辑状态

4、单元格内容更新为编辑后的内容，并在单元格的左上角出现一个小三角形符号，提示该单元格已修改过

5、程序内部的变化：将记录设置为脏数据状态，并将修改后的记录存放在 Record 类型的数组 modified 中。

默认情况下，双击才会使 EditorGridPanel 处于编辑状态，我们可以改变 clicksToEdit 选项的值为 1，只需要单击就行了。EditorGridPanel 默认的选择模型与 GridPanel 也不同，也许您还记得 GridPanel 的选择模型是行选择模型 RowSelectionModel，而 EditorGridPanel 变成了 CellSelectionModel 选择模型，这二者的区别在于前者一次选择一行，支持多选；而后者一次只选择一个单元格，并且不支持多选，这似乎非常符合单元格单击编辑的逻辑。当然，如果您不喜欢 CellSelectionModel，我们可以很轻松的替换掉，只要设置 selModel 选项即可。这两个选择模型中定义的方法也有点不同，所以要注意区别。

二、编辑订单数据

为了配合本章内容，或者说为了应用尽可能多的组件，我打算对订单数据进行编辑，订单包括如下信息：订单编号、下单日期、收货人、收货人地址、交易银行和是否使用快递等，这些信息在修改的时候分别使用 NumberField、DateField、TextField、TextField、ComboBox、Checkbox 组件，一个示例不可能将所有组件全部用到，我只是尽可能多的满足您的胃口，万一碰到特殊的情况，您还得自己仔细斟酌。

本章示例我们访问 json 格式的数据，为了不至于弄得太复杂，数据不是从数据库中获取，而是来自于一个 json 数组。

```
//定义用于表示订单的数组
var data = [{id: 1, date: new Date(), name: "李世民", address: "中华人民共和国", bank: "中国人民银行", isFast: false},
    {id: 2, date: new Date(), name: "搜狗五笔", address: "北京天安门", bank: "中国建设银行", isFast: false},
    {id: 3, date: new Date(), name: "宋江", address: "中国古代某处", bank: "中国建设银行", isFast: true},
    {id: 4, date: new Date(), name: "李宝田", address: "湖南省株洲市红旗广场", bank: "中国工商银行", isFast: false}];
```

按下来定义记录结构，记录的结构很重要，程序在运行过程中可以从结构中得到很多有用的信息：

```
var Order = Ext.data.Record.create(
    [
        {name: "ID", type: "int", mapping: "id"},//编号
        {name: "DATE", type: "date", mapping: "date"},//下单日期
        {name: "NAME", type: "string", mapping: "name"},//收货人姓名
        {name: "ADDRESS", type: "string", mapping: "address"},//收货人地址
        {name: "BANK", type: "string", mapping: "bank"},//交易银行
        {name: "ISFAST", type: "boolean", mapping: "isFast"}//交易银行
    ]
);
```

一个记录结构包含了逻辑名称(name)，并且和 json 对象的属性对应(mapping)，通过 type 指定数据类型，他的值是类型名称的字符串表示。

交易银行我们从列表中取，数据可以来自本地，也可以从数据库服务器读回，我们采用前者：

```
var banks = [
    ["中国建设银行", "中国建设银行"],
```

```
    ["中国银行", "中国银行"],
    ["中国工商银行", "中国工商银行"],
    ["中国人民银行", "中国人民银行"]
];
```

最重要的部分到了，EditGridPanel 的列模型和 GridPanel 相比，要多配置一个名为 editor 选项，该选项指定当前列的数据使用什么组件编辑，这些组件都必须封装在 Ext.grid.GridEditor 类中，Ext.grid.GridEditor 是 Ext.Editor 的子类，用来提供统一的方法完成单元格的编辑。

下面的代码创建了列模型：

```
//定义列模型
var cm = new Ext.grid.ColumnModel([
    {header: "订单编号", dataIndex: "ID", width: 60,
        editor: new Ext.grid.GridEditor(new Ext.form.NumberField())},
    {header: "下单日期", dataIndex: "DATE",width: 140,
        renderer: Ext.util.Format.dateRenderer("Y-m-d"),
        editor: new Ext.grid.GridEditor(new Ext.form.DateField({
            format: "Y-m-d"
        }))},
    {header: "收货人姓名", dataIndex: "NAME",width: 120,
        editor: new Ext.grid.GridEditor(new Ext.form.TextField())},
    {header: "收货人地址", dataIndex: "ADDRESS",id: "ADDRESS",
        editor: new Ext.grid.GridEditor(new Ext.form.TextField())},
    {header: "交易银行", dataIndex: "BANK",width: 120,
        editor: new Ext.grid.GridEditor(new Ext.form.ComboBox({
            store: new Ext.data.SimpleStore({
                fields: ["value", "text"],
                data: banks
            }),
            displayField: "text",
            valueField: "value",
            mode: "local",
            triggerAction: "all",
            readOnly: true,
            emptyText: "请选择银行"
        }))},
    {header: "快递", dataIndex: "ISFAST",width: 70,
        renderer: function(v){return v ? "快递" : "非快递"},
        editor: new Ext.grid.GridEditor(new Ext.form.Checkbox())}
]);
```

在 Ext.grid.GridEditor 中，我们封装的实际上都是表单组件，如果对表单组件不太熟悉，

请阅读第 16 章，可以获取更多有用的内容。

最后，来看看 Ext.grid.EditorGridPanel 的定义：

```
var grid = new Ext.grid.EditorGridPanel({
    store: store,
    cm: cm,
    autoExpandColumn: "ADDRESS",
    width: 800,
    autoHeight: true,
    renderTo: "a",
    autoEncode: true, //提交时是否自动编码
})
```

和 GridPanel 基本一样，唯一多了一个 autoEncode 选项，该选项用于提交数据到服务回时对中文编码，真是人性化啊，什么都替我们想到了。

综上所述（好像教科书上都是这样说的，^_^），EditGridPanel 和 GridPanel 最大的区别主要体现在列模型上，editor 选项可以指定单元格的编辑组件。下面是效果图，看起来和 GridPanel 一模一样，不过点击单元格后则完全不同了，您最好自己将结果运行一下：

下面是编辑了部分单元格的效果：

通过一个棕色的三角形标记，我们很容易看出哪些数据已被修改，这些修改后的数据我们称之为脏（dirty）数据。同时，选择模型成了单元格选择模型，每次只能选择一个单元格而不是一行。

在表格面板的顶部工具栏，定义了三个按钮，分别实现添加一行、删除一行和保存数据的功能，本章的主要亮点其实就在这里，下面我们分别讲述：

- 添加一行

在第 20 章学习 GridPanel 的时候，添加一行的代码我们基本熟悉了，但有还是有些区别，我们必须在新添一行后为数据设脏，这一步很重要，不然使用 ajax 提交时不会传送到服务器，因为非脏数据不会自动保存到 Store 的 modified 属性中，而 modified 中保存的正是被修改的数据。

Store 中的 modified 属性是这样定义的（位于 Store.js 的第 68 行）：

```
this.modified = [];
```

从这段代码中我们看不到任何有用的信息，只知道是一个数组，经过分析，我们还是知道了真实的情况，该属性保存了所有被修改的记录 Record，也就是说，modified 数组中的元素类型为 Record。在本示例中，他其实就是 Order 的集合。

执行添加操作之前，我们最好使用 EditGridPanel 的 stopEditing()方法停止原来的编辑状态，添加操作完成之后，将新增的记录设置成脏数据。

需要的注意的是，一个记录中的任何字段值发生了变化，整行的所有字段值都将设置成脏数据，我个人认为这是一个好的做法。

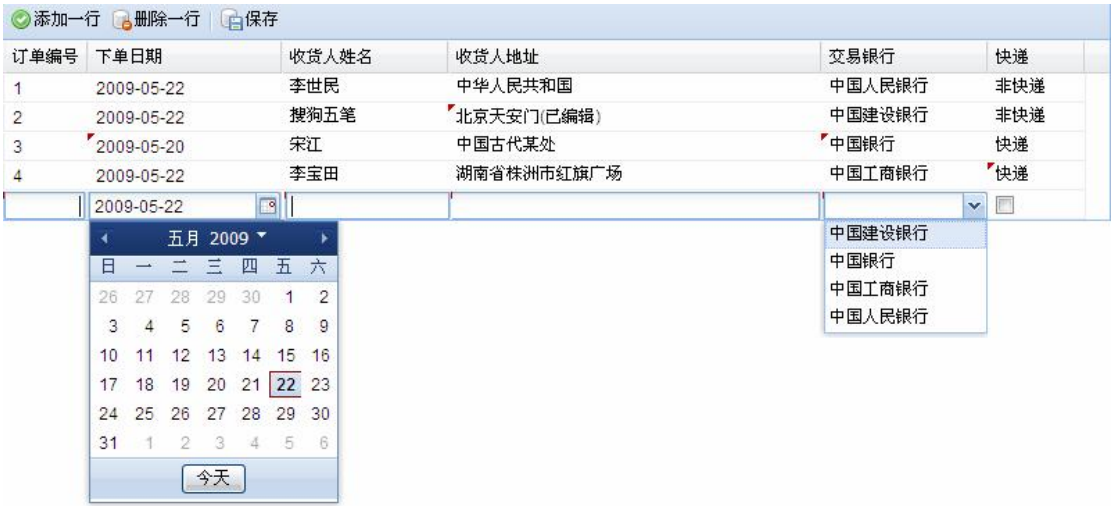
下面是本功能的源代码：

```
var initValue = {
    ID: "",
    DATE: new Date(),
    NAME: "",
    ADDRESS: "",
    BANK: "",
    ISFAST: false
};
var order = new Order(initValue);
grid.stopEditing();
grid.getStore().add(order);

//设置脏数据
order.dirty = true;
//只要一个字段值修改了，该行的所有值都设置成脏标记
order.modified = initValue;
if(grid.getStore().modified.indexOf(order) == -1){
    grid.getStore().modified.push(order);
}
```

initValue 为新记录提供了初始值，在这里我基本置空了，时间是当前时间。

下面的效果图并不是真实的，因为每一个单元格需要双击才能进入编辑状态，而另外的

单元格将退出编辑状态，也就是在任何时候，只有一个单元格才能处理编辑状态，我将所有单元格的编辑状态效果使用 photoshop 整合到了一起。



Ø 删除一行

在 Store 的 modified 数组中，可以保存被修改的记录，也可以保存新添加记录，遗憾的是，删除的记录他坐视不管，所以，删除一行时，我们除了把 EditGridPanel 中的记录行删除，还要将删除的数据传送到服务器，以便同步数据库。

我们先不管数据库那一块，看看如何才能删除 EditGridPanel 的记录行。因为 EditGridPanel 的选择模型不同，是单元格选择模型，该模型默认只支持单选，所以，获取单元格数据时有些许差异，通常我们使用 Ext.grid.CellSelectionModel 的 getSelectedCell()方法获取单元格信息，该方法返回的不是单元格的内容，而是单元格的行索引与列索引，譬如我们单击了第二行第三列的单元格，该方法返回[3，4]。对，就是一个数组。

只要知道了单元格的行，其实就可以从 Store 中获取该行的 Record 对象了（使用 Store.getAt（）方法获取），这正是我们需要的，因为 Store.remove()删除的正是一个 Record 对象。

```
var sm = grid.getSelectionModel();
if(sm.hasSelection()){
    Ext.Msg.confirm("提示", "真的要删除选中的行吗？", function(btn){
        if(btn == "yes"){
            var cellIndex = sm.getSelectedCell();//获取被选择的单元格的行和列索引
            //alert(cellIndex[0] + "," + cellIndex[1])//打印被选择的单元格的行和列索引
            var record = grid.getStore().getAt(cellIndex[0]);
            grid.getStore().remove(record);
        }
    });
}else{
```

```
    Ext.Msg.alert("错误", "请先选择删除的行，谢谢!");
}
```

CellSelectionModel 的 hasSelection()方法用于判断是否有单元格被选择，其他的代码在上述解释的帮助下，应该没多大问题了。

三、保存修改的数据至服务器

添加数据、修改数据和删除数据后，应该与数据库进行同步，这才是关键。添加和修改的数据可以一次提交至服务器，但删除的数据只能单独操作。

Ø 将新增与修改的数据传送到服务器

```
var store = grid.getStore();
//得到修改过的 Recored 的集合
var modified = store.modified.slice(0);
//将数据放到另一个数组中
var jsonArray = [];
Ext.each(modified, function(m){
    //alert(m.data.ADDRESS);//读取当前被修改的记录的地址
    //m.data 中保存的是当前 Recored 的所有字段的值(json)，不包含结构信息
    jsonArray.push(m.data);
});
//通过 ajax 请求将修改的记录发送到服务器，最终影响数据库
Ext.Ajax.request({
    method: "post", //最好不要用 get 请求
    url: "../EditGirdServlet",
    success: function(response, config){
        var json = Ext.util.JSON.decode(response.responseText);
        Ext.Msg.alert("提交成功", json.msg);
    },
    params: {data: Ext.util.JSON.encode(jsonArray)}
});
```

Ø 将被删除的数据传送到服务器

每删除一行数据，都会触发“remove”事件，我们从这里入手，完成删除数据库中的数据的操作。

我们将调用 Store 类的 remove()方法删除指定行的数据，从下面的源代码中可以看出 remove()方法与 remove 事件的关系：

```
remove : function(record){
```

```
        var index = this.data.indexOf(record);
        this.data.removeAt(index);
        if(this.pruneModifiedRecords){
            this.modified.remove(record);
        }
        if(this.snapshot){
            this.snapshot.remove(record);
        }
        this.fireEvent("remove", this, record, index);
},
```

删除一行数据后，下面的操作将被执行：
1、删除 Store 中的指定元素
2、删除 modified 数组中的指定元素
3、触发 remove 事件
4、更新 EditGridPanel 视图

其中，remove 事件处理函数将得到三个参数：Store、当前被删除的 Record 和索引号。第二个参数恰好是我们需要的，我们将把被删除的 Record 传送到删除器，以便删除。

```
//删除一行时，同步数据库
grid.getStore().on("remove", function(s, record, index){
    var jsonArray = [record.data]; //因为 Servlet 只处理数组，所以先变成数组
    Ext.Ajax.request({
        method: "post",
        url: "../EditGirdServlet",
        params: {data: Ext.util.JSON.encode(jsonArray), action: "delete"},
        success: function(response, config){
            var msg = Ext.util.JSON.decode(response.responseText);
            Ext.Msg.alert("", msg.msg);
        }
    });
});
```

因为上面两个操作我使用同一个 Servlet 来处理，所以在删除时，多加了一个名为 action 的参数，表示这是一个删除操作。Ext.util.JSON.encode()方法将 json 对象转成字符串，Ext.util.JSON.decode 则反之。

四、处理请求

上一节我们使用 ajax 将操作的数据上传到了服务器，服务器使用一个名为 EditGirdServlet 的 Servlet 来处理请求，因为数据以 json 格式传输，所以在 Servlet 中必须对其进行转换，我们继续使用 json-lib 来完成。

```
public class EditGirdServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();

        String data = request.getParameter("data");
        String action = request.getParameter("action");//操作类型

        JSONObject a = null;
        Object bean = null;
        JSONArray array = JSONArray.fromObject(data);
        Object[] objArray = array.toArray();
        for(Object obj : objArray){
            a = JSONObject.fromObject(obj);
            bean = a.toBean(a);
            //经过查看，发现 bean 为 MorphDynaBean 类型
            MorphDynaBean morphDynaBean = (MorphDynaBean) bean;
            Integer id = (Integer) morphDynaBean.get("ID");
            String name = (String) morphDynaBean.get("NAME");
            String address = (String) morphDynaBean.get("ADDRESS");
            String bank = (String) morphDynaBean.get("BANK");
            Boolean isFast = (Boolean) morphDynaBean.get("ISFAST");
            String date = (String) morphDynaBean.get("DATE");


            System.out.println(id + "\t" + name + "\t" + address + "\t" + bank + "\t" +
isFast + "\t" + date);
        }

        if("delete".equals(action)){
            out.println("{msg: '恭喜，删除的行已同步数据库!'}");
        }else{
            out.println("{msg: '恭喜，数据修改成功!'}");
        }
        out.flush();
        out.close();
    }
}
```

请求参数 data 就是从客户端传过来的新添、修改或删除的数据，比如我现在添加一行数据同时修改一行数据，这样就会有两个 json 对象传过来，格式像下面这样：

```
[
    {"ID":5,"DATE":"2009-05-13T00:00:00","NAME":"李赞红","ADDRESS":"湖南株洲北大青鸟","BANK":"中国工商银行","ISFAST":true},
    {"ID":4,"DATE":"2009-05-22T13:14:36","NAME":"李宝田","ADDRESS":"湖南省株洲市红旗广场 已修改啦","BANK":"中国工商银行","ISFAST":false}
]
```

很显然，这是一个字符串， 我们应该将数据提取出来。提取数据的思路如下：

1、将上述 json 字符串转换成 JSONArray 对象，注意，此处是数组，如果不是数组，应该使用 JSONObject 类

```
JSONArray array = JSONArray.fromObject(data);
```

2、将 JSONArray 对象转换成 Object 数组，数组中的元素是形如：{p:v, p2:v2}的字符串

```
Object[] objArray = array.toArray();
```

3、循环遍历 objArray 数组，并将数组中的每一个元素转换成 JSONObject 对象

```
a = JSONObject.fromObject(obj);
```

4、将 JSONObject 对象转变成 Java 对象

```
bean = a.toBean(a);
```

5、上一步的 bean 其实是 MorphDynaBean，该类是 json-lib 自带的，我们强制转换

```
MorphDynaBean morphDynaBean = (MorphDynaBean) bean;
```

6、使用 MorphDynaBean 的 get 方法取出对应的值，如：

```
Integer id = (Integer) morphDynaBean.get("ID");
```

数据全部得到了，接下来如何做不用我说了吧？您可以将数据保存到任何地方或者随你处置。故事的结尾，我们把本章的源代码一起贴出来。

五、完整源代码

```
Ext.onReady(function(){
    //定义用于表示订单的数组
    var data = [{id: 1, date: new Date(), name: "李世民", address: "中华人民共和国", bank: "中国人民银行", isFast: false},
            {id: 2, date: new Date(), name: "搜狗五笔", address: "北京天安门", bank: "中国建设银行", isFast: false},
            {id: 3, date: new Date(), name: "宋江", address: "中国古代某处", bank: "中国建设银行", isFast: true},
            {id: 4, date: new Date(), name: "李宝田", address: "湖南省株洲市红旗广场", bank: "中国工商银行", isFast: false}];

    //定义 Proxy
    var proxy = new Ext.data.MemoryProxy(data);
```

```
    var Order = Ext.data.Record.create(
        [
            {name: "ID", type: "int", mapping: "id"},//编号
            {name: "DATE", type: "date", mapping: "date"},//下单日期
            {name: "NAME", type: "string", mapping: "name"},//收货人姓名
            {name: "ADDRESS", type: "string", mapping: "address"},//收货人地址
            {name: "BANK", type: "string", mapping: "bank"},//交易银行
            {name: "ISFAST", type: "boolean", mapping: "isFast"}//交易银行
        ]
    );

    //定义 Reader
    var reader = new Ext.data.JsonReader({}, Order);

    //定义 Store
    var store = new Ext.data.Store({
        proxy: proxy,
        reader : reader,
        autoLoad: true,
        pruneModifiedRecords: true
    });

    var banks = [
        ["中国建设银行", "中国建设银行"],
        ["中国银行", "中国银行"],
        ["中国工商银行", "中国工商银行"],
        ["中国人民银行", "中国人民银行"]
    ];

    //定义列模型
    var cm = new Ext.grid.ColumnModel([
        {header: "订单编号", dataIndex: "ID", width: 60,
            editor: new Ext.grid.GridEditor(new Ext.form.NumberField({allowBlank:
false}))},
        {header: "下单日期", dataIndex: "DATE",width: 140,
            renderer: Ext.util.Format.dateRenderer("Y-m-d"),
            editor: new Ext.grid.GridEditor(new Ext.form.DateField({
                format: "Y-m-d"
            }))},
        {header: "收货人姓名", dataIndex: "NAME",width: 120,
            editor: new Ext.grid.GridEditor(new Ext.form.TextField())},
        {header: "收货人地址", dataIndex: "ADDRESS",id: "ADDRESS",
            editor: new Ext.grid.GridEditor(new Ext.form.TextField())},
        {header: "交易银行", dataIndex: "BANK",width: 120,
```

```
            editor: new Ext.grid.GridEditor(new Ext.form.ComboBox({
                store: new Ext.data.SimpleStore({
                    fields: ["value", "text"],
                    data: banks
                }),
                displayField: "text",
                valueField: "value",
                mode: "local",
                triggerAction: "all",
                readOnly: true,
                emptyText: "请选择银行"
            }))},
        {header: "快递", dataIndex: "ISFAST",width: 70,
            renderer: function(v){return v ? "快递" : "非快递"},
            editor: new Ext.grid.GridEditor(new Ext.form.Checkbox())}
    ]);

    //定义 EditGridPanel
    var grid = new Ext.grid.EditorGridPanel({
        store: store,
        cm: cm,
        autoExpandColumn: "ADDRESS",
        width: 800,
        autoHeight: true,
        renderTo: "a",
        autoEncode: true, //提交时是否自动编码
        //clicksToEdit: 1,
        tbar: [{
            text: "添加一行"    ,
            icon: "../extjs/resources/images/default/dd/drop-yes.gif",
            cls: "x-btn-text-icon",
            handler: function(){
                var initValue = {
                    ID: "",
                    DATE: new Date(),
                    NAME: "",
                    ADDRESS: "",
                    BANK: "",
                    ISFAST: false
                };
                var order = new Order(initValue);
                grid.stopEditing();
                grid.getStore().add(order);
```

```
                    //设置脏数据
                    order.dirty = true;
                    //只要一个字段值修改了，该行的所有值都设置成脏标记
                    order.modified = initValue;
                    if(grid.getStore().modified.indexOf(order) == -1){
                        grid.getStore().modified.push(order);
                    }
                }
            },{
                text: "删除一行",
                icon: "../imgs/database_delete.png",
                cls: "x-btn-text-icon",
                handler: function(){
                    var sm = grid.getSelectionModel();
                    if(sm.hasSelection()){
                        Ext.Msg.confirm("提示", "真的要删除选中的行吗？",
function(btn){
                            if(btn == "yes"){
                                var cellIndex = sm.getSelectedCell();//获取被选择的
单元格的行和列索引
                                //alert(cellIndex[0] + "," + cellIndex[1])//打印被选择
的单元格的行和列索引
                                var record = grid.getStore().getAt(cellIndex[0]);
                                grid.getStore().remove(record);
                            }
                        });
                    }else{
                        Ext.Msg.alert("错误", "请先选择删除的行，谢谢!");
                    }
                }
            },"-", {
                text: "保存",
                icon: "../imgs/database_save.png",
                cls: "x-btn-text-icon",
                handler: function(){
                    var store = grid.getStore();
                    //得到修改过的 Recored 的集合
                    var modified = store.modified.slice(0);
                    //将数据放到另一个数组中
                    var jsonArray = [];
                    Ext.each(modified, function(m){
                        //alert(m.data.ADDRESS);//读取当前被修改的记录的地址
                        //m.data 中保存的是当前 Recored 的所有字段的值(json)，不包含
结构信息
```

```
                jsonArray.push(m.data);
            });

            var r = checkBlank(modified);
            if(!r){
                return;
            }else{
                //通过 ajax 请求将修改的记录发送到服务器，最终影响数据库
                Ext.Ajax.request({
                    method: "post", //最好不要用 get 请求
                    url: "../EditGirdServlet",
                    success: function(response, config){
                        var json = Ext.util.JSON.decode(response.responseText);
                        //grid.getStore().reload();
                        Ext.Msg.alert("提交成功", json.msg);
                    },
                    params: {data: Ext.util.JSON.encode(jsonArray)}
                });
            }
        }
    }]
});

//删除一行时，同步数据库
grid.getStore().on("remove", function(s, record, index){
    var jsonArray = [record.data]; //因为 Servlet 只处理数组，所以先变成数组
    if(record.data.ID != ""){
        Ext.Ajax.request({
            method: "post",
            url: "../EditGirdServlet",
            params: {data: Ext.util.JSON.encode(jsonArray), action: "delete"},
            success: function(response, config){
                var msg = Ext.util.JSON.decode(response.responseText);
                //grid.getStore().reload();
                Ext.Msg.alert("", msg.msg);
            }
        });
    }
});

//验证是否输入的数据是否有效
var checkBlank = function(modified/*所有编辑过的和新增的 Record*/){
    var result = true;
    Ext.each(modified, function(record){
```

```
                var keys = record.fields.keys; //获取 Record 的所有名称
                Ext.each(keys, function(name){
                    //根据名称获取对应的值
                    var value = record.data[name];
                    //找出指定名称所在的列索引
                    var colIndex = cm.findColumnIndex(name);
                    //var rowIndex = grid.getStore().indexOfId(record.id);

                    //根据行列索引找出组件编辑器
                    var editor = cm.getCellEditor(colIndex).field;
                    //验证值是否合法
                    var r = editor.validateValue(value);
                    if(!r){
                        Ext.MessageBox.alert("验证", "对不起，您输入的数据非法");
                        result = false;
                        return;
                    }
                });
            });
            return result;
        }
});
```

六、验证

虽然不是表单，数据还是要验证的。但在前面的文字中我只字未提，这是为了保证代码的简单。事实上，在 EditGridPanel 中的验证方式与表单验证方式完全相同，我们依旧可以使用 allowBlank 来验证是否为必填项。不过，这种方式只对修改有效，如果是新增的行，只能手动验证。

在上面完整的源代码中，checkBlank()方法就是用来验证的的，思路是循环找出所有的单元格编辑组件，并调用 validateValue 方法进行验证。

执行“保存”操作时，在提交数据到服务器之前调用该方法即可。

```
var r = checkBlank(modified);
```

创建 Store 时，新添加了 pruneModifiedRecords 配置，该配置能解决新增一行马上又把该行删除时没有从 modified 删去的问题，该问题将导致提交时无法通过验证。源代码是这样的：

```
remove : function(record){
    var index = this.data.indexOf(record);
```

```
        this.data.removeAt(index);
        if(this.pruneModifiedRecords){
            this.modified.remove(record);
        }
        if(this.snapshot){
            this.snapshot.remove(record);
        }
        this.fireEvent("remove", this, record, index);
},
```

语句 this.modified.remove(record)必须在 pruneModifiedRecords 为 true 时才执行，但默认情况下 pruneModifiedRecords 为 false，所以需要在 Store 创建时配置。

七、替换选择模型

如果不想使用单元格选择模型，可以使用行选择模型进行替代，只需要配置 selModel 选项就能达到我们的要求：

```
var grid = new Ext.grid.EditorGridPanel({
    store: store,
    cm: cm,
    autoExpandColumn: "ADDRESS",
    width: 800,
    autoHeight: true,
    renderTo: "a",
    autoEncode: true, //提交时是否自动编码
    clicksToEdit: 1,
    selModel: new Ext.grid.RowSelectionModel({singleSelect: true}),
    ……（省略）
```

八、小结

本章算是篇幅最长的一章了，但是其作用也不可估量，您可以将源代码复制到环境中运行，通过运行结果来理解源代码，可能更容易掌握。

第二十四章：树与选择模型

一、树——TreePanel

在 Extjs 中，树结构是放在面板中的，将树与面板结合在一起，就成了我们所看到的 TreePanel。TreePanel 是一个功能强大完备的组件，也是一个比较复杂的组件。我在阅读他的源码时，花了不少的时间。不过还好，Extjs 优雅的代码和清晰的组件关系没有让我在代码海洋中迷路，我顺利从浩瀚的代码中折了回来。

关于 TreePanel 我会花上好几个章节来讲述，基于他的复杂，也基于他的优秀。在阅读源代码的过程中，我急于和大家分享这份喜悦，苦于笔墨不济，我似乎无法完整描述，但我依然会通过源代码来解释一些实现过程。

树是由一个一个节点组成的，树有唯一的根节点。树的节点分成三种：根节点，叶子节点和非叶子节点。这些节点的名称虽然不同，但他们的类型是一样的，都叫 TreeNode。根节点唯一；叶子节点没有子节点，但有唯一的父节点；非叶子节点有一个父节点和若干个子节点。

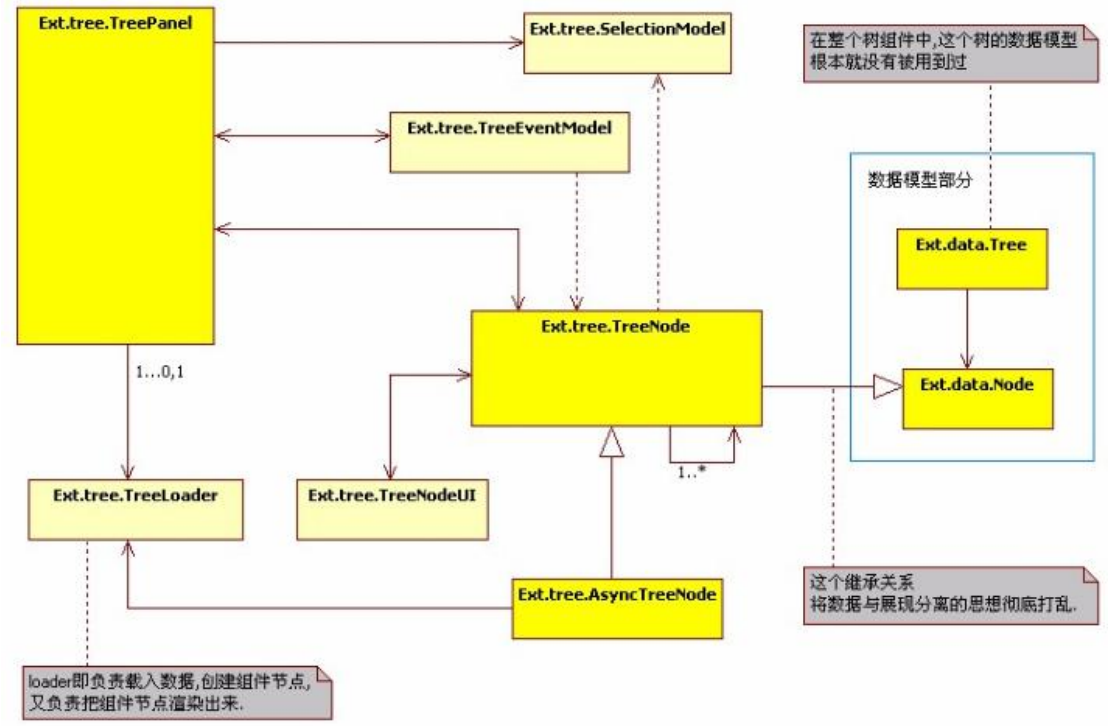
Extjs 将树在浏览器中渲染后，每一个节点都是通过<li>标签实现的，同一级节点放在一个<ul>中，各种标记通过巧妙的组合，最后成了我们所看到的效果。

舞台上神采飞扬的是 TreePanel，这个是和我们的视线直接造成冲击的部分。但幕后的工作者功劳更大，他们默默无闻却扮演着无比重要的角色。下面是所有演员和幕后工作人员名单：

- Ext.data.Node：节点的基础定义，注意是在 Ext.data 命名空间中，保存节点信息并提供节点的基本操作，如添加相对节点、判断节点状态、遍历节点、扩展与收缩节点。
- Ext.tree.TreeNode：树节点定义，从 Ext.data.Node 继承而来，定义了一个名为 text 的属性，用于指定节点的显示文本，与该类关联的类有节点 UI 渲染器、节点信息远程加载器和节点选择模型。另外，重写了 Ext.data.Node 一部分节点操作方法。
- Ext.tree.AsyncTreeNode：异步树节点，保存远程获取的节点信息，是 Ext.tree.TreeNode 的子类。
- Ext.tree.TreeLoader：树加载器，通过 ajax 技术从服务器获取节点信息。
- Ext.tree.TreeNodeUI：节点 UI 渲染器，该类负责节点在浏览器中显示效果的绘制。
- Ext.tree.DefaultSelectionModel：默认的选择器，定义了如何选择树中的节点。一次只能选择一个节点。
- Ext.tree.MultiSelectionModel：多选选择器，一次可以选择多个节点。
- Ext.data.Tree：定义了树的结构，所有节点都保存在该结构中。
- Ext.tree.TreePanel：树形面板，树和面板的结合产物。
- Ext.tree.TreeEditor：使树节点具备修改功能。

以下是各主要类之间的类图，这张图是从网上论坛下载的，本来我自己也画了一张，但

是我认为这张图更加详细，不过这其中掺杂了作者的一些看法，我们持旁观态度，不敢苟同。



二、创建简单的 TreePanel

我们从最简单的开始，现在，您可能很急切地想知道如何创建一个 TreePanel，这并不是一件具有挑战性的工作，因为我很可以很轻易地完成。显然，树是由一系列具有层次结构的节点组成，不管是根节点，还是叶子节点，或者是非叶子节点，他们都是节点，区别在于与其他节点的关系。所以，要创建一棵树，只需要创建一系列节点，并建立各节点的关系就可以了。

创建一个节点，我们使用下面的语法：

```
var node    = new Ext.tree.TreeNode({text: "节点显示文本"})
```

构造函数的参数是一个 config，一般来说有两个属性：id 和 text，id 唯一标识树中的节点，text 则是节点的显示文本。我们可以添加任何你需要的属性，最后会统一保存在 attributes 属性中，比如我希望保存一个名为 hobby（爱好）的属性，我们可以这样定义：var node = new Ext.tree.TreeNode({text: "节点名", hobby:"跳伞"})，使用 node. attributes.hobby 即可取出 hobby 的值。我们经常会在节点对象上保存一些信息，这个小知识点一定要掌握好。

一般来说，构造一棵树，完全可以通过添加子节点这个操作来完成。添加子节点的方法是 appendChild(节点对象或者节点对象数组)，通过该方法可以为节点添加一个子节点，也可以为节点一次添加多个子节点，参数为节点数组时，该数组中的所有节点都会成为子节点。

```
level_1_1.appendChild([level_1_1_1, level_1_1_2, level_1_1_3]);
```

上面的代码表示将 level_1_1_1, level_1_1_2, level_1_1_3 作为子节点添加到 level_1_1 中。

最后，要特别为 TreePanel 指定一个根节点，调用 TreePanel 的 setRootNode(root)方法完成。

```
Ext.onReady(function(){
    Ext.QuickTips.init();

    //定义根节点
    var root = new Ext.tree.TreeNode({text: "根节点", iconCls:"me-iconCls"});
    var level_1_1 = new Ext.tree.TreeNode({text: "一级_1", iconCls:"me-iconCls"});
    var level_1_2 = new Ext.tree.TreeNode({text: "一级_2", iconCls:"me-iconCls"});

    var level_1_1_1 = new Ext.tree.TreeNode({text: "二级_1", iconCls:"me-iconCls"});
    var level_1_1_2 = new Ext.tree.TreeNode({text: "二级_2", iconCls:"me-iconCls"});
    var level_1_1_3 = new Ext.tree.TreeNode({text: "二级_3", iconCls:"me-iconCls"});

    var level_1_1_3_1 = new Ext.tree.TreeNode({text: "二级_3_1",
iconCls:"me-iconCls"});
    var level_1_1_3_2 = new Ext.tree.TreeNode({text: "二级_3_2",
iconCls:"me-iconCls"});
    var level_1_1_3_3 = new Ext.tree.TreeNode({text: "二级_3_3",
iconCls:"me-iconCls"});

    level_1_1.appendChild([level_1_1_1, level_1_1_2, level_1_1_3]);
    level_1_1_3.appendChild([level_1_1_3_1,level_1_1_3_2,level_1_1_3_3]);
    root.appendChild([level_1_1, level_1_2]);

    //定义 TreePanel
    var tree = new Ext.tree.TreePanel({
        width: 200,
        height: 300,
        title: "树",
        lines: true
    });
    tree.setRootNode(root);
    tree.render("a");
    tree.expandAll();
})
```

代码中有些东西需要解释一下，lines 选项为 true 时，将显示连接线，为 false 时，不会

显示连接线，默认是 true 值。tree.expandAll()方法能将节点全部展开，是动画效果哦。与之对应的 collapseAll()方法则能将所有节点收缩。

下面是 TreePanel 类其他的常用方法：

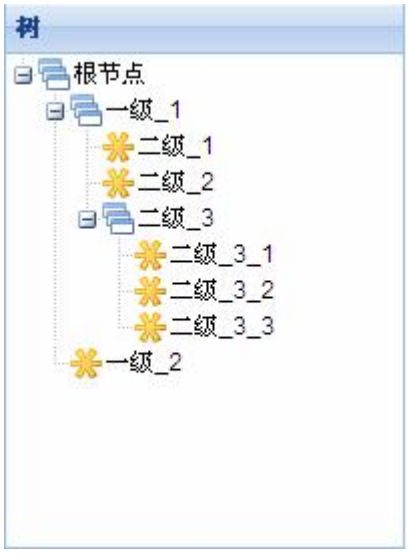
- getNodeById(String id) : Node 根据 ID 找到节点对象，如果没有设置 ID，Ext 会为节点自动生成一个，不过我们将无法控制，最好为每一个节点指定一个唯一的 ID
- getRootNode() : Node 得到树的根节点
- getSelectionModel() : TreeSelectionModel 得到 TreePanel 的选择模型

还有一个非常重要的功能要声明，我使用 iconCls 配置选项改变了默认的节点图标。节点图标无法通过配置选项指定（至少在源代码中没有找到相关部分，只能重写 UI 实现），只能定义 CSS 样式表，me-iconCls 是自定义的类选择器，必须按照下面的格式定义：

```
.x-tree-node-collapsed .me-iconCls{
    background-image: url(../imgs/application_double.png)
}
.x-tree-node-expanded .me-iconCls{
    background-image: url(../imgs/application_cascade.png)
}
.x-tree-node-leaf .me-iconCls{
    background-image: url(../imgs/asterisk_yellow.png)
}
```

类选择器名称的粗体部分是自定义名称，您可以根据自己的喜爱定义，.x-tree-node-collapsed 表示非叶子节点收缩状态下的图标，.x-tree-node-expanded 表示非叶子节点扩展状态下的图标，.x-tree-node-leaf 为叶子节点图标。

该代码运行后的效果如下所示：



三、选择模型

和表格面板一样，树也有选择模型，选择模型用于选择特定节点，并使用色块标注。树默认的选择模型是 Ext.tree.DefaultSelectionModel，该类并没有定义太多的方法，但基本都与选择有关。

下面是 Ext.tree.DefaultSelectionModel 中定义的主要方法：

- select(TreeNode node) : TreeNode 选择指定节点
- unselect(TreeNode node) : void 不选择指定节点
- selectPrevious() : TreeNode 选择上一个节点，并返回上一个节点
- selectNext() : TreeNode 选择下一个节点，并返回下一个节点
- getSelectedNode() : TreeNode 获取选择的节点
- isSelected(TreeNode node) : Boolean 判断指定节点是否被选择
- selectPath(String path, [String attr], [Function callback]) : void 根据路径选择任意一个节点，path 为路径，如：/根节点/一级_1，必须以/开头。attr 是选择依据，默认根据 id 选择。

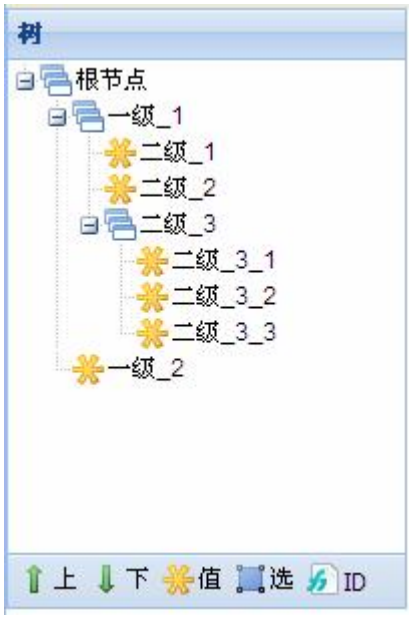
下面的示例是对上面的部分方法的演示，似乎没什么好解释的了。仔细去研究吧。

```
Ext.onReady(function(){
    Ext.QuickTips.init();

    //定义根节点
    var root = new Ext.tree.TreeNode({id: 1, text: "根节点", iconCls:"me-iconCls"});
    var level_1_1 = new Ext.tree.TreeNode({id: 2, text: "一级_1", iconCls:"me-iconCls"});
    var level_1_2 = new Ext.tree.TreeNode({id: 3, text: "一级_2", iconCls:"me-iconCls"});

    var level_1_1_1 = new Ext.tree.TreeNode({id: 4, text: "二级_1",
iconCls:"me-iconCls"});
    var level_1_1_2 = new Ext.tree.TreeNode({id: 5, text: "二级_2",
iconCls:"me-iconCls"});
    var level_1_1_3 = new Ext.tree.TreeNode({id: 6, text: "二级_3",
iconCls:"me-iconCls"});

    var level_1_1_3_1 = new Ext.tree.TreeNode({id: 7, text: "二级_3_1",
iconCls:"me-iconCls"});
    var level_1_1_3_2 = new Ext.tree.TreeNode({id: 8, text: "二级_3_2",
iconCls:"me-iconCls"});
    var level_1_1_3_3 = new Ext.tree.TreeNode({id: 9, text: "二级_3_3",
iconCls:"me-iconCls"});

    level_1_1.appendChild([level_1_1_1, level_1_1_2, level_1_1_3]);
    level_1_1_3.appendChild([level_1_1_3_1,level_1_1_3_2,level_1_1_3_3]);
```

```
root.appendChild([level_1_1, level_1_2]);

//定义 TreePanel
var tree = new Ext.tree.TreePanel({
    width: 200,
    height: 300,
    title: "树",
    //lines: false,

    bbar: [{
        icon: "../imgs/arrow_up.png",
        cls: "x-btn-text-icon",
        text: "上",
        tooltip: "向上选择一个节点",
        tooltipType: "qtip",
        handler: function(){
            var model = tree.getSelectionModel();//获取选择模型
            model.selectPrevious();
        }
    },{
        icon: "../imgs/arrow_down.png",
        cls: "x-btn-text-icon",
        text: "下",
        tooltip: "向下选择一个节点",
        tooltipType: "qtip",
        handler: function(){
            var model = tree.getSelectionModel();//获取选择模型
            model.selectNext();
        }
    },{
        icon: "../imgs/asterisk_yellow.png",
        cls: "x-btn-text-icon",
        text: "值",
        tooltip: "获取被选择的节点的显示文本",
        tooltipType: "qtip",
        handler: function(){
            var model = tree.getSelectionModel();//获取选择模型
            var selectedNode = model.getSelectedNode();//获取选择的节点
            if(selectedNode){
                Ext.MessageBox.alert("", selectedNode.text);
            }
        }
    },{
        icon: "../imgs/shape_handles.png",
```

```
            cls: "x-btn-text-icon",
            text: "选",
            tooltip: "自定义选择任何节点",
            tooltipType: "qtip",
            handler: function(){
                Ext.Msg.prompt("路径", "请输入要选择的节点路径（从根节点开始，
如“/根节点/一级_1”）", function(btn, txt){
                    if(btn == "ok"){
                        tree.selectPath(txt, "text");
                    }
                });
            }
        },{
            icon: "../imgs/page_white_freehand.png",
            cls: "x-btn-text-icon",
            text: "ID",
            tooltip: "根据 ID 进行选择",
            tooltipType: "qtip",
            handler: function(){
                Ext.Msg.prompt("路径", "请输入要选择的节点的 ID", function(btn,
txt){
                    if(btn == "ok"){
                        var model = tree.getSelectionModel();//获取选择模型
                        var selNode = tree.getNodeById(txt);//根据 ID 得到节点对象
                        model.select(selNode);//选择该节点对象
                    }
                });
            }
        }]
    });

    tree.setRootNode(root);
    tree.render("a");
    tree.expandAll();
})
```

效果如下图所示：



四、MultiSelectionModel

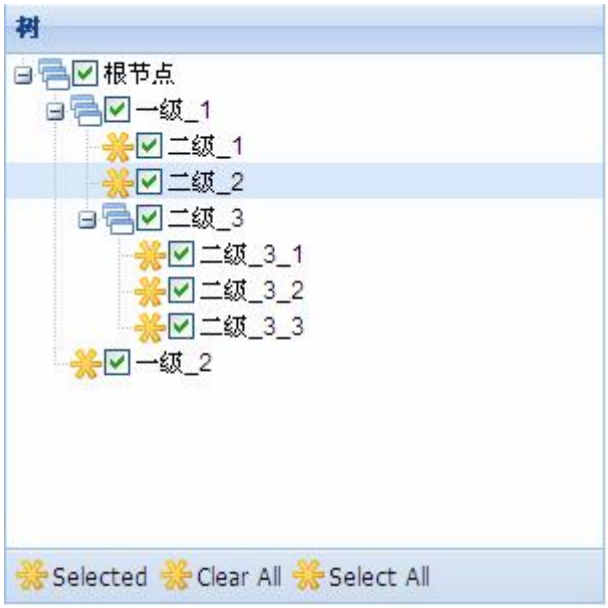
DefaultSelectionModel 只支持单选，如果要支持多选，必须将 TreePanel 的选择模型替换成 Ext.tree. MultiSelectionModel，下面的代码完成了该任务：

```
//定义 TreePanel
var tree = new Ext.tree.TreePanel({
    width: 200,
    height: 300,
    title: "树",
    selModel: new Ext.tree.MultiSelectionModel(), //支持多选的选择模型
}
```

DefaultSelectionModel 中获取所有选择的节点的方法有了改变，必须使用 getSelectedNodes()方法得到，该方法返回 TreeNode 类型数据，遍历一下就行了。另外，select()的定义也变成了 select(TreeNode node, [EventObject e], Boolean keepExisting) : TreeNode，keepExisting 表示在选择新节点的同时要不要删除原先已经选择的节点。

五、带复选框的节点

树节点前面带上一个复选框用于多选并不是什么稀罕事，这种多选方式与 MultiSelectionModel 并不相同，MultiSelectionModel 选择模型通过节点反白配合 Ctrl 键实现，而这一小节，我们将介绍通过复选框实现多选，就像下面的效果图：



首先我们急需解决的是如何在节点前面放置复选框，TreePanel 内置作了实现，创建 Node 时指定 checked 配置选项即可，其值为 true 或 false，这也是节点的初始状态，下面创建了一个带复选框的 Node：

```
var root = new Ext.tree.TreeNode({id: 1, checked: false, text: "根节点", iconCls:"me-iconCls"});
```

TreePanel 定义了一个方法 getChecked()，能得到所有处于选中状态的节点，该方法非常有意思，如果带参数，返回的结果会有所不同，参数可以设置为节点的某一个属性名称，如果并不想得到节点对象，而只想得到节点的 id，将“id”作为参数调用 getChecked()方法，如果只想得到被选择的节点的显示文本，则使用 getChecked(“text”)调用。

我们也可以通过代码来设置选择的状态，这需要 TreeNodeUI 类的配合，每一个 Node 都有一个独立的 TreeNodeUI 类与之对应，负责该节点的渲染和显示。通过 Node 的 ui 属性可以得到他自己的 TreeNodeUI 对象。同时，TreeNodeUI 类有一个 checkbox 的属性，该属性即为复选框的 dom 对象，设置 checked 即可修改复选框状态。但是要注意的是，虽然复选框的外观效果改变了，但模型数据并没有随之改变，用户获取被选择的节点时，并不会因为复选框取消选择而改变，我们必须调用 TreeNodeUI 的 onCheckChange()方法使外观界面与模型数据保持同步。

复选框的状态发生变化后，会触发 checkchange 事件，当前 Node 会自动传送到事件处理函数中。关于事件，我基本没花篇幅去描述，源代码中关于事件的定义、事件处理函数的参数列表都写得很清楚。如果有机会，我会独自成篇。

我们将为您提供一个示例，该示例主要完成四个功能：获取所有被选节点、全选、全不选、选择父节点连带选择所有子节点。效果如上图，代码如下所示：

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

```
    Ext.BLANK_IMAGE_URL = "../extjs/resources/images/default/s.gif";
    Ext.QuickTips.init();

    //定义根节点
    var root = new Ext.tree.TreeNode({id: 1, checked: false, text: "根节点", iconCls:"me-iconCls"});
    var level_1_1 = new Ext.tree.TreeNode({id: 2, checked: false, text: "一级_1", iconCls:"me-iconCls"});
    var level_1_2 = new Ext.tree.TreeNode({id: 3, checked: false, text: "一级_2", iconCls:"me-iconCls"});

    var level_1_1_1 = new Ext.tree.TreeNode({id: 4, checked: true, text: "二级_1", iconCls:"me-iconCls"});
    var level_1_1_2 = new Ext.tree.TreeNode({id: 5, checked: true, text: "二级_2", iconCls:"me-iconCls"});
    var level_1_1_3 = new Ext.tree.TreeNode({id: 6, checked: true, text: "二级_3", iconCls:"me-iconCls"});

    var level_1_1_3_1 = new Ext.tree.TreeNode({id: 7, checked: true, text: "二级_3_1", iconCls:"me-iconCls"});
    var level_1_1_3_2 = new Ext.tree.TreeNode({id: 8, checked: true, text: "二级_3_2", iconCls:"me-iconCls"});
    var level_1_1_3_3 = new Ext.tree.TreeNode({id: 9, checked: true, text: "二级_3_3", iconCls:"me-iconCls"});

    level_1_1.appendChild([level_1_1_1, level_1_1_2, level_1_1_3]);
    level_1_1_3.appendChild([level_1_1_3_1,level_1_1_3_2,level_1_1_3_3]);
    root.appendChild([level_1_1, level_1_2]);

    //定义 TreePanel
    var tree = new Ext.tree.TreePanel({
        width: 300,
        height: 300,
        title: "树",

        bbar: [{
            text: "Selected",
            icon: "../imgs/asterisk_yellow.png",
            cls: "x-btn-text-icon",
            tooltip: "获取被选择的节点的显示文本",
            tooltipType: "qtip",

            handler: function(){
                var nodes = tree.getChecked();//获取所有节点
```

```
                for(var i = 0; i < nodes.length; i ++){
                    alert(nodes[i].text);
                }
            }
        },{
            text: "Clear All",
            icon: "../imgs/asterisk_yellow.png",
            cls: "x-btn-text-icon",
            tooltip: "清除",
            tooltipType: "qtip",

            handler: function(){
                var nodes = tree.getChecked();
                var selModel = tree.getSelectionModel();
                for(var i = 0; i < nodes.length; i ++){
                    nodes[i].ui.checkbox.checked = false;//将选中的节点取消
                    nodes[i].ui.onCheckChange();//与模型数据同步
                }
            }
        },{
            text: "Select All",
            icon: "../imgs/asterisk_yellow.png",
            cls: "x-btn-text-icon",
            tooltip: "选择所有",
            tooltipType: "qtip",

            handler: function(){
                tree.iteratorCheck(tree.getRootNode(), true);
            }
        }]
});

/**
 * 触发复选框状态改变事件
 */
tree.on("checkchange", function(node){
    var children = node.childNodes;
    for(var i = 0; i < children.length; i ++){
        children[i].ui.checkbox.checked = node.ui.checkbox.checked;
        children[i].ui.onCheckChange();
    }
});
```

```
    /**
     * 递归遍历所有节点
     */
    tree.iteratorCheck = function(node, checked){
        if(node.hasChildNodes()){
            node.eachChild(function(currentNode){
                tree.iteratorCheck(currentNode, checked);
            });
        }

        node.ui.checkbox.checked = checked;
        node.ui.onCheckChange();
    }

    tree.setRootNode(root);
    tree.render("a");
    tree.expandAll();
})
```

六、小结

本章介绍了 TreePanel 的使用，这是一个绝对值得去使用的组件，其本身也决定了他的复杂，但是，真正去使用的时候，还是有章可循的。我们要习惯从偶尔性中找出必然性，以不变应万变。

TreePanel 的默认选择模型是 Ext.tree.DefaultSelectionModel，经过前面章节的洗礼，大家对这东东不再陌生了，Extjs 的设计确实是可圈可点，结构和职责都很清晰，并且还应用了一些喜闻乐见的设计模式，在讲解时，我都忽略了，因为我怕添加技术难度，不去关注也许能减少您的心理障碍。

第二十五章：动态操作树节点

一、概述

树节点在树结构中的位置一般是固定的，我们并不需要对节点进行操作，但确实存在一些需求必须操作节点，比如在显示产品类型时，为了体现出产品类型的层级关系，不会使用 GridPanel 来显示，而是使用 TreePanel；又比如显示部门时，部门之类同样存在上下级关系，使用树是一个理想的选择。所以，树通常有两种用途：一是作为导航结构为用户提供一个结构清晰容易操作的菜单；二是代替 GridPanel 显示具有层次结构的数据。前者一般是固定的，后者则必须实现常见的 CRUD 等功能。

树的节点操作主要有：追加子节点，添加同级节点，删除节点，显示节点的相关信息。这些功能 TreePanel 在其他同仁的一致努力下，得到了非常好的实现，我们只要坐享其成就可以了。真的很少看到有这么强大的树，他不是树，是树精了。

节点由 TreeNode 来定义，TreeNode 是 Node 的子类，实际上，Node 定义的是结构和数据，TreeNode 在 Node 的基础上还负责对节点进行渲染，二者分工明确，职责清晰，这种设计理念值得借鉴。

Node 内部定义了五个属性：parentNode，父节点；firstChild，第一个子节点；lastChild，最后一个子节点；previousSibling，前一个兄弟节点；nextSibling，后一个兄弟节点。这几个属性相互联系，通过一个节点就能找到与该节点相关的其他节点。childNodes 则是节点的所有子节点。

新增节点有两种方式：一是成为一个子节点；二是成为兄弟节点。这两种添加方式分别使用 appendChild()与 insertBefore()方法完成，这两个方法的定义如下：

- appendChild(Node/Array node) : Node：添加一个子节点，如果参数为数组，则添加多个子节点
- insertBefore(Node node, Node refNode) : Node：在子参照节点之前添加子节点，参数 node 表示新的子节点，refNode 表示参数子节点，二者都是当前节点的子节点。如果不指定参照节点则追加为子节点。

删除节点同样有两种做法：一是自杀式，自己把自己删除；二是父杀子式，父节点删除子节点，这两个方法的定义分别如下：

- remove() : Node：删除自己
- removeChild(Node node) : Node：父节点删除子节点

TreePanel 支持节点显示文本的编辑，用户只要进行稍有间隔的两次单击就能使节点处理编辑状态，这个工作是由 Ext.tree.TreeEditor 完成的。Ext.tree.TreeEditor 对 TreePanel 进行了二次处理，但做得很巧妙。基本使用方法如下：

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

代码中，我们创建了一个 Ext.tree.TreeEditor 对象，tree 表示前面已经创建好的 TreePanel 对象，而第二个参数则表示节点的编辑组件，默认为 TextField，我们可以定义任何组件，不过似乎很少这样做，我们还能借助 TextField 自身的功能验证用户的输入是否合法，allowBlank 为 false 表示为必填项，您可以在前面的章节中找到验证的详细解释。

二、基本操作

Ø 在同级节点之前添加节点

事实上，并没有提供直接的方法来实现该功能，选中一个节点后，要在该节点之前添加兄弟节点，必须转化成在选中节点的父节点上添加子节点，并指定子节点的参照节点，而参照节点恰恰是选中节点。

我们可以这样定义：父节点. insertBefore(新节点，选中节点)，选中节点其实就是我们的参照节点。看下面的代码：

```
sameLevelBefore: function(tree){
    Ext.MessageBox.prompt("输入", "请输入新节点的名称：", function(btn, txt){
        if(btn == "ok"){
            var newNode = new Ext.tree.TreeNode({text: txt});
            var selNode = tree.getSelectionModel().getSelectedNode();
            if(!selNode){
                Ext.Msg.alert("错误", "在添加新节点之前请先选择参照节点！");
            }else if(selNode.id == tree.getRootNode().id){
                Ext.Msg.alert("错误", "根节点不能添加同级节点！");
            }else{
                selNode.parentNode.insertBefore(newNode, selNode);
            }
        }
    });
},
```

selNode 是选择的节点，newNode 为新节点，下面将会不断出现这两个对象名称，请仔细理解。

Ø 在同级节点之后添加节点

在同级节点之后添加节点这个功能就更没有提供直接支持了，我们只能通过调用 insertBefore()委婉实现。我们转化成另一种方式：即将新节点添加到下一个兄弟节点的前面，这样也就处于指定节点的后面了。

```
selNode.parentNode.insertBefore(newNode, selNode.nextSibling);
```

其他代码与上面相差无几。

Ø 追加子节点

这个操作简单得出乎我们的想象，前面一直是这样做的。

```
selNode.appendChild(newNode);//增加子节点
selNode.expand();//展开子节点
```

这里有一点要注意，新增子节点之后，子节点并不会自动展开，调用节点的 expand()方法能自动展开子节点。

Ø 删除节点

我们采用自杀式删除，selNode.remove()立马完成该功能。但是，根节点不能删除，所以在删除之前最好判断一下用户选择的是否是根节点。

```
if(selNode.id == tree.getRootNode().id){
    Ext.Msg.alert("", "根节点不能删除");
}else{
    selNode.remove();
}
```

我们根据用户选择的节点的 id 与树的根节点 id 进行比较，因为 id 是唯一的，所以如果相同则表示用户选择了根节点。

Ø 修改节点信息

为了能修改节点，必须使用 TreeEditor，当然，使用的方法非常简单：

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

这样，节点都具备修改功能了，同时，会验证节点显示内容是否为空。

Ø 显示节点信息

我们打算显示选中节点的父节点、兄弟节点、第一和最后一个子节点，以及节点的完整路径。

```
if(!selNode){
        Ext.Msg.alert("警告", "没有选择任何节点！");
        return;
}
```

```
var parentNode = selNode.parentNode;
var firstChild = selNode.firstChild;
var lastChild = selNode.lastChild;
var previousSibling = selNode.previousSibling;
var nextSibling = selNode.nextSibling;

Ext.MessageBox.alert("INFO", (parentNode ? "父节点：" + parentNode.text : "无父节点")
    + "<br>" + (firstChild ? "第一个子节点：" + firstChild.text : "无第一个子节点")
    + "<br>" + (lastChild ? "最后一个子节点：" + lastChild.text : "无最后一个子节点")
    + "<br>" + (previousSibling ? "上一个兄弟节点：" + previousSibling.text : "无上一个
兄弟节点")
    + "<br>" + (nextSibling ? "下一个兄弟节点：" + nextSibling.text : "无下一个兄弟节点
")
    + "<br>节点路径：" + selNode.getPath("text")
);
```

三、事件

添加、删除或者修改了节点信息后，可能最终要和数据库同步，这需要使用 ajax 技术来完成。但更重要的是，如何得到添加、删除和修改后的节点信息才是问题的关键。最直接的办法是触发操作完成后的事件，在事件中使用 ajax 将修改结果传送到服务器。

使用 appendChild()添加子节点后，将触发 TreePanel 的 append 事件，而 insertBefore()方法执行后会触发 insert 事件，用 remove()删除节点后则触发 remove 事件，修改了节点内容后会触发 TreeEditor 的 complete 事件。更多事件请参考源代码。

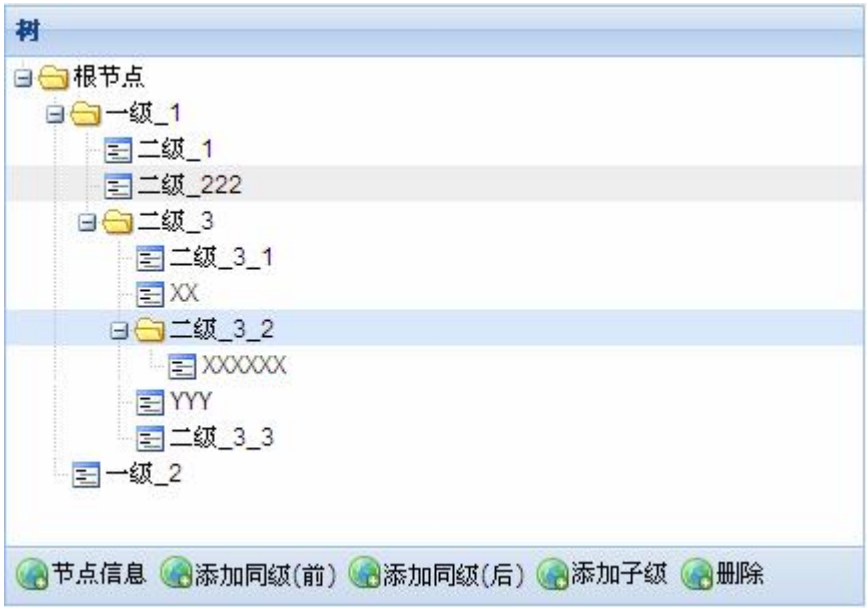
下面是事件处理的基本代码，没有使用 ajax 上传到服务器，因为这个问题已经很简单了。

```
//处理事件
editor.on("complete", function(editor, value, startValue){
    alert("原值：" + startValue + ",新值：" + value);
    alert("被修改节点：" + editor.editNode);
});

tree.on("insert", function(tree, parent, node, refNode){
    alert("inserted");
});

tree.on("remove", function(tree, parent, node){
    alert("removed");
});
```

```
tree.on("append", function(tree, parent, node){
    alert("appended");
});
```

将事件回调函数中的参数大概说明一下：tree 自然是 TreePanel，parent 是操作节点的父节点，node 是操作节点本身，而 refNode 是指参照节点。

本章的示例运行结果如图：



四、小结

本章从实际项目开发角度阐述了节点的动态操作，涉及到的方法并不多，如果我们熟悉 DOM 模型，会发现很多属性和方法都似曾相识，理解起来并不困难。总体来说，Extjs 确实体现出了大师的风格，结构清晰，代码易读，并有详细的说明注释。网上容易上手又很全的资料并不多，但是如果弄清楚了 Extjs 的编程风格，仔细阅读源代码，一定能取得事半功倍的效果。

第二十六章：远程获取节点数据

一、概述

对于规模很大的项目，导航树可能很复杂，或者如果数据特别多，也会导致树的级别或节点很多，而很多时候，不一定要操作所有节点，用户只关注他们感兴趣的节点，因此我们可以做成异步形式，点击某一个节点时，通过 ajax 从服务器拉取该节点的子节点，而不是将树的所有节点一次性读出来。

TreeNode 不支持异步读取子节点，该操作由 Ext.tree.AsyncTreeNode 类实现，Ext.tree.AsyncTreeNode 是 Ext.tree.TreeNode 的子类，重写了部分方法，譬如最关键的 expand()、hasChildNodes()等等。AsyncTreeNode 的 expand()方法先将数据从服务器读出，再调用 TreeNode 的 expand()方法来展开，所以，异步读取节点是由 TreeNode 和 AsyncTreeNode 配合完成的。

从服务器获取数据由 Ext.tree.TreeLoader 负责。该类通过 ajax 将数据从服务器取回，并创建新节点作为子节点。Ext.tree.TreeLoader 只支持 json 对象，不支持数组和 xml，如果要支持数组和 xml，必须对该类进行重写。但是，json 格式确实是一种容易理解并且高效的格式，如果没有什么特别的要求，最好还是不要改罢。

Ext.tree.TreeLoader 从服务器读取的数据应该像下面这样定义：

```
public class SyncTreeServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();

        String id = request.getParameter("node");
        if(id != null){
            if("1".equals(id)){
                out.println("[{id:'2', text: '层级_1', leaf: false},{id:'3', text: '层级_2', leaf: true}]");
            }else if("2".equals(id)){
                out.println("[{id:'4', text: '层级_1_1', leaf: false},{id:'5', text: '层级_1_2', leaf: true}]");
            }else if("4".equals(id)){
                out.println("[{id:'6', text: '层级_1_1_1', leaf: false},{id:'7', text: '层级_1_1_2', leaf: true}]");
            }else if("6".equals(id)){
                out.println("[{id:'8', text: '层级_1_1_1_1', leaf: true},{id:'9', text: '层级_1_1_1_2', leaf: true}]");
```

```
                }
            }

            out.flush();
            out.close();
        }
}
```

注意下面几个问题：

1) 所有的节点都必须定义一个唯一的 id，因为 TreeLoader 发请求时会传送一个名为 node 的参数，该参数正是异步节点的 id，并根据 id 读取该节点下的所有子节点。
2) 必须明确声明节点的类型，如果节点为叶子节点，leaf 设置为 true，如果为非叶子节点，leaf 设置为 false。至于为什么要这样，请继续看下文。
3) 如果节点由数据库管理和维护，则最好设计一个能满足要求的表结构，不要冒然为之。

二、异步加载解析

如果您对阅读源代码不感兴趣，本小节可以跳过。不过，我还是强烈建议您多了解相关的工作机制。

我们从根节点开始。

一棵树至少要提供一个根节点，其他节点都可以异步加载生成。根节点应该是 AsyncTreeNode 类型的节点，这样才能从服务器获取数据。我们这样定义根节点：

```
var root = new Ext.tree.AsyncTreeNode({id: "1", text: "根节点", leaf: false});
```

根节点的 id 被设置成了 1，text 是“根节点”，leaf 为 false 表示该节点下还有子节点，即不是叶子节点。

TreePanel 也有了不一样的配置，dataUrl 选项是必须的配置的，该选项指定返回子节点数据的服务器资源，本例为 Servlet 组件。

```
var tree = new Ext.tree.TreePanel({
    title: "异步加载节点",
    width: 300,
    height: 300,
    dataUrl: "../SyncTreeServlet"
});

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

点击节点后，触发节点的单击事件，在 Ext.tree.TreeNodeUI 中使用 this.node.toggle()语句开展开或收缩子节点，其实就是调用 TreeNode 的 toggle()方法，该方法在 Ext.tree.TreeNode 如下定义：

```
toggle : function(){
    if(this.expanded){
        this.collapse();
    }else{
        this.expand();
    }
},
```

这段代码很好理解，如果节点是展开的，则收缩，反之亦然。

异步加载的重头戏自然是 expand()方法，如果节点类型为 Ext.tree.AsyncTreeNode，则调用 Ext.tree.AsyncTreeNode 中重写的 expand()方法，该方法的源代码如下：

```
expand : function(deep, anim, callback){
    if(this.loading){ // if an async load is already running, waiting til it's done
        var timer;
        var f = function(){
            if(!this.loading){ // done loading
                clearInterval(timer);
                this.expand(deep, anim, callback);
            }
        }.createDelegate(this);
        timer = setInterval(f, 200);
        return;
    }
    if(!this.loaded){
        if(this.fireEvent("beforeload", this) === false){
            return;
        }
        this.loading = true;
        this.ui.beforeLoad(this);
        var loader = this.loader || this.attributes.loader || this.getOwnerTree().getLoader();
        if(loader){
            loader.load(this, this.loadComplete.createDelegate(this, [deep, anim,
callback]));
            return;
        }
    }
    Ext.tree.AsyncTreeNode.superclass.expand.call(this, deep, anim, callback);
},
```

假设一个异步节点正在加载，则以 200 毫秒为间隔进行等待，即每隔 200 毫秒去判断一下原来的节点是否加载完成，直到加载完成为止，才进行新一轮加载。

如果单击的节点未曾加载过，则开始从服务器获取子节点数据，但是，在获取数据之前，必须先做一些准备工作，比如执行 this.ui.beforeLoad(this)将节点变成“正在进行时”的状态，然后调用 loader.load(this, this.loadComplete.createDelegate(this, [deep, anim, callback]))语句从服务器加载子节点数据。load()方法定义在 Ext.tree.TreeLoader 中：

```
load : function(node, callback){
    if(this.clearOnLoad){
        while(node.firstChild){
            node.removeChild(node.firstChild);
        }
    }
    if(this.doPreload(node)){ // preloaded json children
        if(typeof callback == "function"){
            callback();
        }
    }else if(this.dataUrl||this.url){
        this.requestData(node, callback);
    }
},
```

属性 clearOnLoad 默认为 true 的，也就是说在加载数据之前，先删除已经存在的子节点。如果子节点已存在，即已被加载过，不再从服务器获取数据，而是直接从缓存中读取，这样可以有效减少和服务器交互的次数，this.doPreload(node)方法正是完成些功能，该方法源代码如下：

```
doPreload : function(node){
    if(node.attributes.children){
        if(node.childNodes.length < 1){ // preloaded?
            var cs = node.attributes.children;
            node.beginUpdate();
            for(var i = 0, len = cs.length; i < len; i++){
                var cn = node.appendChild(this.createNode(cs[i]));
                if(this.preloadChildren){
                    this.doPreload(cn);
                }
            }
            node.endUpdate();
        }
        return true;
    }else {
```

```
            return false;
        }
},
```

从节点的 attributes 属性中读取出他的所有子节点 children，children 是保存在内存中的子节点集合数据（只是节点的数据，不是渲染后的节点），即使子节点全部被删除了，依旧可以重新将节点数据从 children 中读出来，这样能防止第二次和服务器交互。node.beginUpdate()方法内执行 this.childrenRendered = false 语句，表示子节点尚未渲染，接下来遍历 children，将节点的数据一个个读出（第一个节点数据皆为 json 对象），并通过 createNode(cs[i])创建节点，createNode()的定义如下：

```
createNode : function(attr){
        // apply baseAttrs, nice idea Corey!
        if(this.baseAttrs){
            Ext.applyIf(attr, this.baseAttrs);
        }
        if(this.applyLoader !== false){
            attr.loader = this;
        }
        if(typeof attr.uiProvider == 'string'){
           attr.uiProvider = this.uiProviders[attr.uiProvider] || eval(attr.uiProvider);
        }
        if(attr.nodeType){
            return new Ext.tree.TreePanel.nodeTypes[attr.nodeType](attr);
        }else{
            return attr.leaf ?
                            new Ext.tree.TreeNode(attr) :
                            new Ext.tree.AsyncTreeNode(attr);
        }
},
```

创建子节点时，为了让子节点也能异步加载，同样要设置子节点的 loader 属性，并且通过 attr.uiProvider = this.uiProviders[attr.uiProvider] || eval(attr.uiProvider)指定一个同样的渲染器。最后，也是很关键的一步，根据节点的 leaf 属性创建不同类型的子节点，如果 leaf 为 true，表示是叶子节点，不再异步加载，创建 Ext.tree.TreeNode 类型的子节点就可以了，如果 leaf 为 false，表示为非叶子节点，必须通过异步加载从服务器获取子节点，所以要创建 Ext.tree.AsyncTreeNode 类型的节点。这就是为什么从服务器返回节点数据时要设置 leaf 的原因，我们要明确告诉他节点的类型，因为 extjs 并不会主动判断。

子节点全部创建后，node.endUpdate()被执行，该方法负责渲染内存中的子节点，在界面上显示出来。

如果程序判断发现单击节点没有被加载过，就必须从服务器读取子节点数据了，这是通过 this.requestData(node, callback)完成的，该方法的源代码如下：

```
requestData : function(node, callback){
    if(this.fireEvent("beforeload", this, node, callback) !== false){
        this.transId = Ext.Ajax.request({
            method:this.requestMethod,
            url: this.dataUrl||this.url,
            success: this.handleResponse,
            failure: this.handleFailure,
            scope: this,
            argument: {callback: callback, node: node},
            params: this.getParams(node)
        });
    }else{
        // if the load is cancelled, make sure we notify
        // the node that we are done
        if(typeof callback == "function"){
            callback();
        }
    }
},
```

这是一个典型的 ajax 请求了，此刻，在 TreePanel 中配置的 dataUrl 将派上用场，其实配置 TreeLoader 的 url 也是一样的，如果 dataUrl 没有配置就会使用 url，注意看 params 属性，该属性表示要传送到服务器的请求参数，this.getParams(node)方法负责生成请求参数，源代码如下：

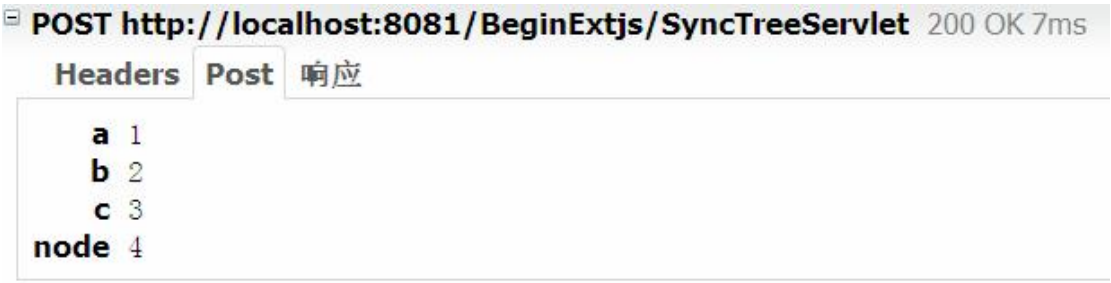
```
getParams: function(node){
    var buf = [], bp = this.baseParams;
    for(var key in bp){
        if(typeof bp[key] != "function"){
            buf.push(encodeURIComponent(key), "=", encodeURIComponent(bp[key]),
"&");
        }
    }
    buf.push("node=", encodeURIComponent(node.id));
    return buf.join("");
},
```

上面代码中，最重要的就是 buf.push("node=", encodeURIComponent(node.id))，该语句将生成形如 node=1 的请求参数，baseParams 也是请求参数列表，我们可以在 baseParams 中配置多个请求参数，将 TreePanel 的创建修改成下面这样：

```
var tree = new Ext.tree.TreePanel({
   title: "异步加载节点",
```

```
    width: 300,
    height: 300,
    //dataUrl: "../SyncTreeServlet"
    loader: new Ext.tree.TreeLoader({baseParams:{a:1,b:2,c:3}, dataUrl: "../SyncTreeServlet"})
});
```

现在，我们将 dataUrl 替换成 loader 选项，loader 是一个 TreeLoader 对象，并配置了 baseParams:{a:1,b:2,c:3}，每次发送异步请求时，a、b、c 三个参数都会被发送到服务器中。如果我们单击了根节点，则会将下面四个参数传送到服务器：



参数设定好后，通过 Ext.Ajax.request()方法发送 ajax 请求，请求成功，交给 handleResponse()方法处理：

```
handleResponse : function(response){
    this.transId = false;
    var a = response.argument;
    this.processResponse(response, a.node, a.callback);
    this.fireEvent("load", this, a.node, response);
},
```

该方法触发“load”事件，而处理响应数据又是通过 this.processResponse(response, a.node, a.callback)语句实现的：

```
processResponse : function(response, node, callback){
    var json = response.responseText;
    try {
        var o = eval("("+json+")");
        node.beginUpdate();
        for(var i = 0, len = o.length; i < len; i++){
            var n = this.createNode(o[i]);
            if(n){
                node.appendChild(n);
            }
        }
        node.endUpdate();
```

```
            if(typeof callback == "function"){
                callback(this, node);
            }
        }catch(e){
            this.handleFailure(response);
        }
},
```

这段代码是如此熟悉，var o = eval("("+json+")");将从服务器返回的字符串转换成 json 对象或数组，然后循环找出每一个 json 对象，并转化成 Node，代码和前面如出一辙。

node.endUpdate()语句渲染所有子节点。

到此为止，从异步加载数据，到创建新节点，到追加为子节点，到渲染，一个流程下来，子节点全部创建好了，最后，Ext.tree.AsyncTreeNode 类的 expand()方法调用 Ext.tree.AsyncTreeNode.superclass.expand.call(this, deep, anim, callback)将子节点展开，该方法其实就是调用了 Ext.tree.AsyncTreeNode 的父类 Ext.tree.TreeNode 的方法。

解析起来似乎有点复杂，但还不至于无法接受，使用起来就更简单了。下面是本章示例的完整源代码：

```
Ext.onReady(function(){
    Ext.QuickTips.init();

    var root = new Ext.tree.AsyncTreeNode({id: "1", text: "根节点", leaf: false});

    var tree = new Ext.tree.TreePanel({
        title: "异步加载节点",
        width: 300,
        height: 300,
        //dataUrl: "../SyncTreeServlet"
        loader: new Ext.tree.TreeLoader({baseParams:{a:1,b:2,c:3}, dataUrl:
"../SyncTreeServlet"})
    });

    tree.setRootNode(root);
    tree.render("a");
})
```

三、小结

本章貌似第一次分析源代码，其实一开始并没打算这样做，但我很想让大家知道 Ext.tree.AsyncTreeNode 与 Ext.tree. TreeNode 的区别，并且让您明白 Ext.tree.AsyncTreeNode

是如何从服务器拉取数据的，一路分析下来，我把重要的语句变成了粗体，方便理解和阅读。

总而言之，Ext.tree.AsyncTreeNode 离不开 Ext.tree.TreeLoader 的支持，甚至可以说，Ext.tree.AsyncTreeNode 只负责展开子节点，其他工作则交给 Ext.tree.TreeLoader 去做了，我认为 Ext.tree.TreeLoader 负责读取数据就行了，不必连子节点创建和渲染的工作也包揽，不知 Extjs 是作何考虑。

第二十七章：选项卡面板——Ext.TabPanel

一、关于魅族和 M8

迷迷糊糊居然好几天没有写关于 Extjs 的文章了，这一章写了个标题后一直被我放在角落里。最近工作的事特多，忙着备课，extjs、xml 和 oracle，这些都是不曾带过的课程，整理思路，写教案，做 PPT，事情虽然不难，但很琐碎，需要花费很多时间。

生活貌似太平静了，整天上班，在家也不闲着，虽然不觉得无聊，但至少无趣。所以，决定买一个新手机犒劳自己。先前那个 Nokia3230 被我用了两年了吧，磨损得比较严重，电池也不好使了，但并不影响使用，即使偶尔出战状况，似乎也无关痛痒，我不跑业务，没有客户，和同事朋友打个电话就算是突然断了或者没接到也无所谓。被女儿不知摔了多少次，一直半伤半残地呆在我笔记本包里，而一直关注的魅族 M8 在我的心里却无形中放大了，这是我心怡的手机，不为别的，只为魅族的专注与执着，还有对用户的尊重。

魅族 M8 是款好手机，我不是托，动不动就挂人之托的人去死，魅族以前出 mp3 的时候我压根不知道，他出手机才开始关注，才知道有魅族这个公司。M8 手机的发展非常曲折，传言也很多，但都被黄章同学（魅族的掌门人）一一破解，非常佩服魅族的精神，国产手机厂商无一能做到。现在，魅族一个偌大的公司，专心专意经营这一款产品，并且是绝对的精品，这种勇气值得我们敬佩。当然，从 MP3 到手机，这是一个截然相反的转变，有很多问题需要解决，产品依然有瑕疵，但我看到了魅族的努力，这就够了。

所以，决定抽个时间，去专卖店一趟，把 M8 买回来，非常期待，也希望魅族不要让我失望。我支持魅族，是希望能用上一个好的产品，希望魅族的精神能赋予 M8 更多的意义。

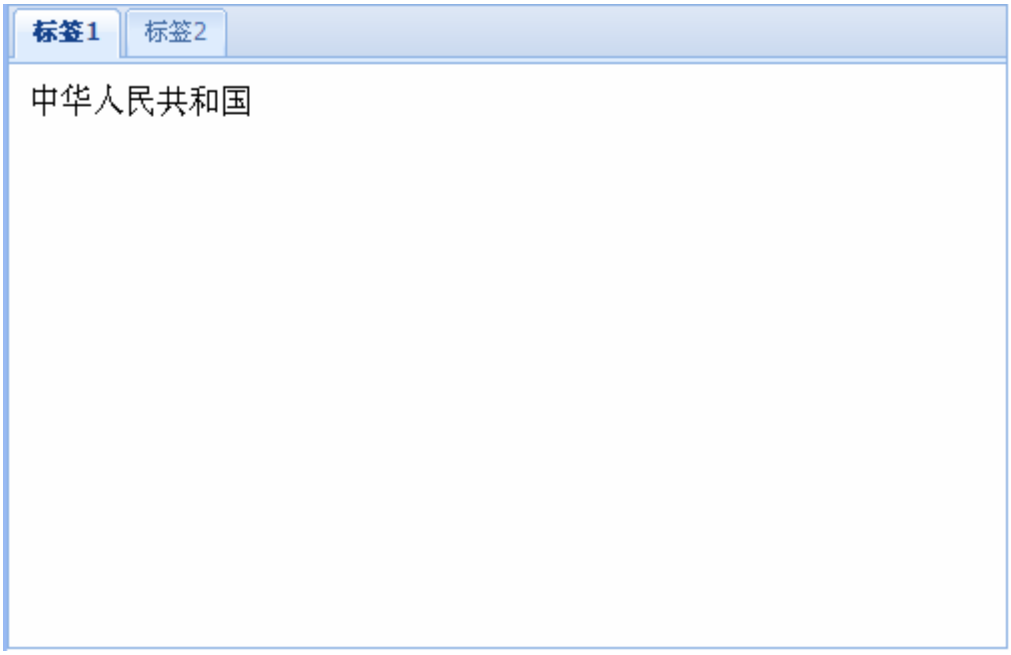
二、TabPanel 概述

TabPanel 同样是个好东西，他为我们提供了一个非常优质的选项面板，这似乎是我见过的最灵活最容易使用的选项面板了。我们看到的 extjs API 文档就是使用了这个组件，第一次看到的时候是不是觉得很了不起？

TabPanel 是 Panel 的子类，即 TabPanel 本身就是一个面板，而 TabPanel 的每一个标签又是一个面板，明白了这个道理，TabPanel 的使用就不难了。当然，效果被重画了，如下图所示。

图中我们定义了一个 TabPanel，而 TabPanel 上又定义了两个 Panel，分别是标签 1 和标签 2，最终的理解就是：我们定义了三个面板而已，其中 TabPanel 是父面板，他有两个子面板。

在第 13 章我们详细分析了 Panel 的使用，如果不清楚请返回温习。



首先，我们创建了两个普通的 Panel，内容通过 html 选项指定，这是静态内容，如果试图动态加载某一个页面的内容，配置 autoLoad 选项即可，下面的代码看起来没有任何的特别之处：

```
var panel1 = new Ext.Panel({
    title: "选项卡 1",
    html: "中华人民共和国",
    bodyStyle: "padding: 10px"
});

var panel2 = new Ext.Panel({
    title: "选项卡 2",
    html: "美利坚合众国",
    bodyStyle: "padding: 10px"
});
```

然后，创建 TabPanel，因为 TabPanel 是 Panel 的子类，所以代码看起来是如此相似：

```
var tabPanel = new Ext.TabPanel({
    renderTo: "d",
    width: 500,
    height: 300
});
```

这三个面板还没有任何关系，按照正常的逻辑，我们应该将 panel1 和 panel2 添加到 tabPanel 中，这是通过从 Container 类中继承下来的方法 add()实现的。

```
tabPanel .add(panel1);
tabPanel .add(panel2);
```

最后，将 panel1 设置为默认的标签：

```
tabPanel .setActiveTab(panel1);
```

至此，我们能做一个具备基本功能的选项面板了。上面的讲述是分步进行的，目的是为了让您更详细了解其中的内幕，事实上，很多人更喜欢写成下面的样子：

```
Ext.onReady(function(){
    var tabPanel    = new Ext.TabPanel({
        renderTo: "d",
        width: 500,
        height: 300,
        items:[{
            title: "选项卡 1",
            html: "中华人民共和国",
            bodyStyle: "padding: 10px"
        },{
            title: "选项卡 2",
            html: "美利坚合众国",
            bodyStyle: "padding: 10px"
        }],
        activeItem: 0
    });
})
```

我们将标签放到了 items 选项中，通过 activeItem：0 指定索引为 0 的标签为当前标签。

三、TabPanel 标签操作

在实际项目中，我们经常会对标签进行创建与删除操作，结合 Viewport，给应用程序一种全新的视角。在使用 Ext 的 API 文档时，大家一定被他独特的效果所吸引，最引人注目的便是他独一无二的选项卡，每一个 API 页面占用一个唯一的标签，还能对标签进行关闭操作，这让我们有一种强烈的欲望去实现同样的功能。

上一节我们讲述了标签的创建，对于标签的关闭也同样简单。从 Container 类中继承下来的 remove()方法能删除 TabPanel 的标签，如果是删除当前标签，代码如下：

```
var activeTab = tabPanel.getActiveTab();
tabPanel.remove(activeTab);
```

其中，getActiveTab()方法能获取 TabPanel 的活动标签，然后调用 remove 方法删除。

基于上述原理，我们能实现“关闭其他标签”的功能，即删除除了当前标签以外的其他所有标签，通过 TabPanel 的 items 属性可以得到所有打开的标签，遍历每一个标签，判断当前标签是否是活动标签，如果不是，则删除。当然，通常情况下，不带关闭按钮的标签不删除。

```
var activeTab = tabPanel.getActiveTab();
tabPanel.items.each(function(item){
    if(item.closable && item != activeTab){
        tabPanel.remove(item);
    }
});
```

如果要删除所有标签，将上面的代码稍做修改：

```
var activeTab = tabPanel.getActiveTab();
tabPanel.items.each(function(item){
    if(item.closable){
        tabPanel.remove(item);
    }
});
```

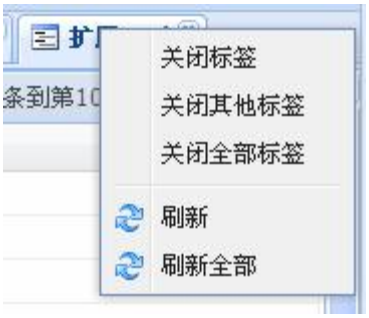
Panel 类有一个更新器，更新器用于刷新面板内容，而 TabPanel 的标签正好是 Panel，所以，我们可以通过更新器再次更新标签的内容，如果要更新当前活动标签的内容，代码如下：

```
ctxItem.getUpdater().update(ctxItem.autoLoad.url);
```

其中，ctxItem 即为活动标签。

四、标签弹出菜单

下面是一个标签弹出菜单，以插件的形式提供给 TabPanel 使用，这个插件从网上找到的，来源无从考究，我做了部分修改，帖出来以飨读者。



```
// Very simple plugin for adding a close context menu to tabs
Ext.ux.TabCloseMenu = function(){
    var tabs, menu, ctxItem;
    this.init = function(tp){
        tabs = tp;
        tabs.on('contextmenu', onContextMenu);
    }

    function onContextMenu(ts, item, e){
        if(!menu){ // create context menu on first right click
            menu = new Ext.menu.Menu([{
                id: tabs.id + '-close',
                text: '关闭标签',
                handler : function(){
                    tabs.remove(ctxItem);
                }
            },{
                id: tabs.id + '-close-others',
                text: '关闭其他标签',
                handler : function(){
                    tabs.items.each(function(item){
                        if(item.closable && item != ctxItem){
                            tabs.remove(item);
                        }
                    });
                }
            },{
                id: tabs.id + '-close-all',
                text: '关闭全部标签',
                handler : function(){
                    tabs.items.each(function(item){
                        if(item.closable){
                            tabs.remove(item);
                        }
                    });
                }
            },"-",{
                id: tabs.id + "-fresh",
                text: "刷新",
                iconCls: "x-tbar-loading",
                handler: function(){
                    ctxItem.getUpdater().update(ctxItem.autoLoad.url);
                }
```

```
                },{
                    id: tabs.id + "-fresh-all",
                    text: "刷新全部",
                    iconCls: "x-tbar-loading",
                    handler: function(){
                        tabs.items.each(function(item){
                            item.getUpdater().update(item.autoLoad.url);
                        });
                    }
                }]);
            }
            ctxItem = item;
            var items = menu.items;
            items.get(tabs.id + '-close').setDisabled(!item.closable);
            var disableOthers = true;
            tabs.items.each(function(){
                if(this != item && this.closable){
                    disableOthers = false;
                    return false;
                }
            });
            items.get(tabs.id + '-close-others').setDisabled(disableOthers);
            var disableAll = true;
            tabs.items.each(function(){
                if(this.closable){
                    disableAll = false;
                    return false;
                }
            });
            items.get(tabs.id + '-close-all').setDisabled(disableAll);
            menu.showAt(e.getPoint());
        }
};
```

将上面的代码保存为 TabCloseMenu.js 文件，在创建 TabPanel 时，配置 plugins 选项：

```
var tabPanel = new Ext.TabPanel({
    renderTo: "center_div",
    enableTabScroll: true,
    border:false,
    frame: true,
    autoWidth: true,
    layoutOnTabChange:true,
    plugins: [new Ext.ux.TabCloseMenu()],
```

```
    defaults: {autoScroll:true} //如果内容超出范围,则自动出现滚动条
    //deferredRender:false //一次性将选项卡内容全部加载,不推荐
});
```

五、小结

TabPanel 是一个非常有用且常用的组件，能将更多的内容在有限的空间中展现出来，用户通过鼠标便可在不同的页面之间进行切换。从 windows95 开始，选项卡控件大行其道，移植到 B/S 中，这么方便并且效果如此出众，除了 Extjs，无人能出其右。

第二十八章：Viewport 类

一、概述

前面学过表单面板布局，通过布局有规则地将组件整到一起，即合理又养眼。页面同样需要布局，特别是首页，一个好的首页能将系统功能有效整合，并显示尽可能多的信息，用户使用极易上手，这是终极目标，而 Viewport 类让我们的理想成为可能。在 B/S 架构中，有一个像 Viewport 一样的组件提供如此优良的页面布局，实在是我辈之大幸。

如果熟悉 Swing，一定也熟悉 Jframe 和 JPanel 的边界布局（BorderLayout），边界布局将画面分成东、西、南、北、中五个区域，我们可以将控件放在指定区域，达到符合客户需要的界面效果。Viewport 使用的也是边界布局，虽然这不是必须的，但是，使用边界布局是最常见的方式。

使用边界布局在注意下面几个问题：

- Ø 一个页面只能有一个 Viewport 对象
- Ø Viewport 必须要指定中间区域，其他区域可有可无，如果没有指定中间区别，会无情地抛出异常

```
if(!c){
    throw 'No center region defined in BorderLayout ' + ct.id;
}
```

二、Viewport 的基本使用

Viewport 的完整类名为 Ext.Viewport。为了让 Viewport 实现边界布局，必须设置 layout 选项为“border”，表示边界布局，边界布局是通过 Ext.layout.BorderLayout 类实现的，这是一个比较复杂的类，布局效果由该类绘制，并负责计算区域的大小和位置以及显示风格。

Ext.layout.BorderLayout.Region 类定义每一个区域，其中，有一种特殊的区域，会绘制一个分割条，通过鼠标的拖动可以改变相邻两个区域的大小，但总大小不变，这个类叫 Ext.layout.BorderLayout.SplitRegion，显然，他是 Ext.layout.BorderLayout.Region 的子类。

Viewport 的每一个区域都是一个面板（Ext.Panel），当然，也可以是他的子类，如 TreePanel、TabPanel、FormPanel 或者 GridPanel。总之，创建一个面板后为该面板指定一个 region 即可。region 配置了面板放置的位置，一个有五个值，分别是 west、east、south、north、center，如果要为面板增加一个分割条，配置 split 选项为 true。

下面的示例创建了一个简单的 Viewport，简单得除了布局别的什么都没有。

```
Ext.onReady(function(){
    var north = new Ext.Panel({
```

```
        title: "北方",
        region: "north",
        height: 80
    });

    var south = new Ext.Panel({
        title: "南方",
        region: "south",
        height: 80
    });

    var center = new Ext.Panel({
        title: "中间",
        region: "center"
    });

    var west = new Ext.Panel({
        title: "西方",
        region: "west",
        width: 200,
        split: true
    });

    var east = new Ext.Panel({
        title: "东方",
        region: "east",
        width: 200,
        split: true
    });


    var vp = new Ext.Viewport({
        layout : 'border',
        items:[north, west, south, east, center]
    });
})
```

我不知道还有什么可以解释的，直接看效果图吧。

面板有了，要放什么东西完全取决于您的设计与想象。通常情况下，西方会放置树形面板（TreePanel），用于功能导航，中间放置选项卡面板（TabPanel），用于显示各个页面，北方放置系统 LOGO，南方放置版权信息，东方不需要定义，直接无视。

在面板中增设 collapsible: true 属性可以自动收缩面板，效果非常炫，您不妨试试。

```
var west = new Ext.Panel({
    title: "西方",
    region: "west",
    width: 200,
    split: true,
    collapsible: true
});
```

事实上，collapsible、split、region 都不是面板的属性，而是 Ext.layout.BorderLayout.Region 类中定义的，这里移花接木了。Ext.layout.BorderLayout.Region 除了上面讲到的几个选项，下面的选项也会经常被用到：

collapseMode：设置面板收缩模式，有默认和“mini”两种模式，mini 模式我们平时见得比较多，基本完全隐藏。但这两种模式都有一个共同特点，即在分割条上有个小三角形按钮，点击该按钮面板马上收缩。自己写代码体验一下实际的效果比我在这里费 N 多口水绝对有效得多。

minWidth：设置面板的最小宽度，默认为 50 像素。通常用于设置东西两个区域

minHeight：设置面板的最小调试，默认为 50 像素。通常用于设置南北两个区域

defaultMargins：设置面板四边离区域的边界，默认为 defaultMargins：{left:0,top:0,right:0,bottom:0}

下面的示例不再单纯使用 Ext.Panel，而是将 TreePanel 和 TabPanel 结合起来，该示例完成了应用程序主页面的雏形。

```
Ext.BLANK_IMAGE_URL = "../extjs/resources/images/default/s.gif";
Ext.onReady(function() {
    var top = new Ext.Panel({
                    region : 'north',
                    title : "标题",
                    height : 80,
                    border : true,
                    html: "LOGO",
                    margins : '0 0 5 0'
                });

    var left = new Ext.tree.TreePanel({
                    region : 'west',
                    collapsible : true,
                    title : 'Navigation',
                    width : 200,
                    autoScroll : true,
                    split : true,
                    listeners : {
                        click : function(n) {
                            var url = n.attributes.url;
                            var id = n.attributes.id;//id 如果没有定义，则自动生成一个唯一的 ID
                            var p = center.getItem(id);//获取节点 ID 对应的标签面板
                            if(url){
                                if(p){
                                    //如果已经存在，则激活
                                    center.setActiveTab(p);
                                }else{
                                    //如果不存在，则创建新的并激活
                                    p = new Ext.Panel({
                                        title: n.attributes.text,
                                        autoLoad: {url: url, scripts: true},
                                        closable: true,//标签上出现关闭按钮
                                        id: id //这里一定要设置
                                    });
                                    center.add(p);
                                    center.setActiveTab(p);
                                }
                            }
                        }
```

```
                }
            });

    // 定义根节点
    var root = new Ext.tree.TreeNode({
                text : "根节点"
            });

    var child1 = new Ext.tree.TreeNode({
                text : "子节点 1",
                url: "s.jsp"
            });
    var child2 = new Ext.tree.TreeNode({
                text : "子节点 2",
                url: "grid_2.html"
            });
    var child3 = new Ext.tree.TreeNode({
                text : "子节点 3",
                url: "editGrid_2.html"
            });

    root.appendChild([child1, child2, child3]);
    left.setRootNode(root);

    var center = new Ext.TabPanel({
                region : 'center',
                items : {
                    id : "opt1",
                    title : 'Default Tab',
                    html : '欢迎使用北大青鸟办公管理系统'
                },
                enableTabScroll: true
            });

    center.setActiveTab("opt1");

    var bottom = new Ext.Panel({
                region : 'south',
                title : 'Information',
                collapsible : true,
                html : '版权所有，翻版必究!',
                split : true,
                height : 100,
                minHeight : 100,
```

```
                bodyStyle : "padding: 10px; font-size: 12px; text-align:center;"
            });

    var vp = new Ext.Viewport({
                layout : 'border',
                items : [top, left, center, bottom]
            });

    // 展开所有节点
    left.expandAll();
})
```

TreePanel 中的每一个节点都有一个 url 配置选项，该配置用来指定 TabPanel 要打开的页面地址，单击节点后，判断 TabPanel 中是否已经创建好对应标签，如果已创建，激活就行了，如果没有创建，则创建新标签并将该标签激活，成为当前标签。

一起看看效果图：

三、小结

无语 ING……

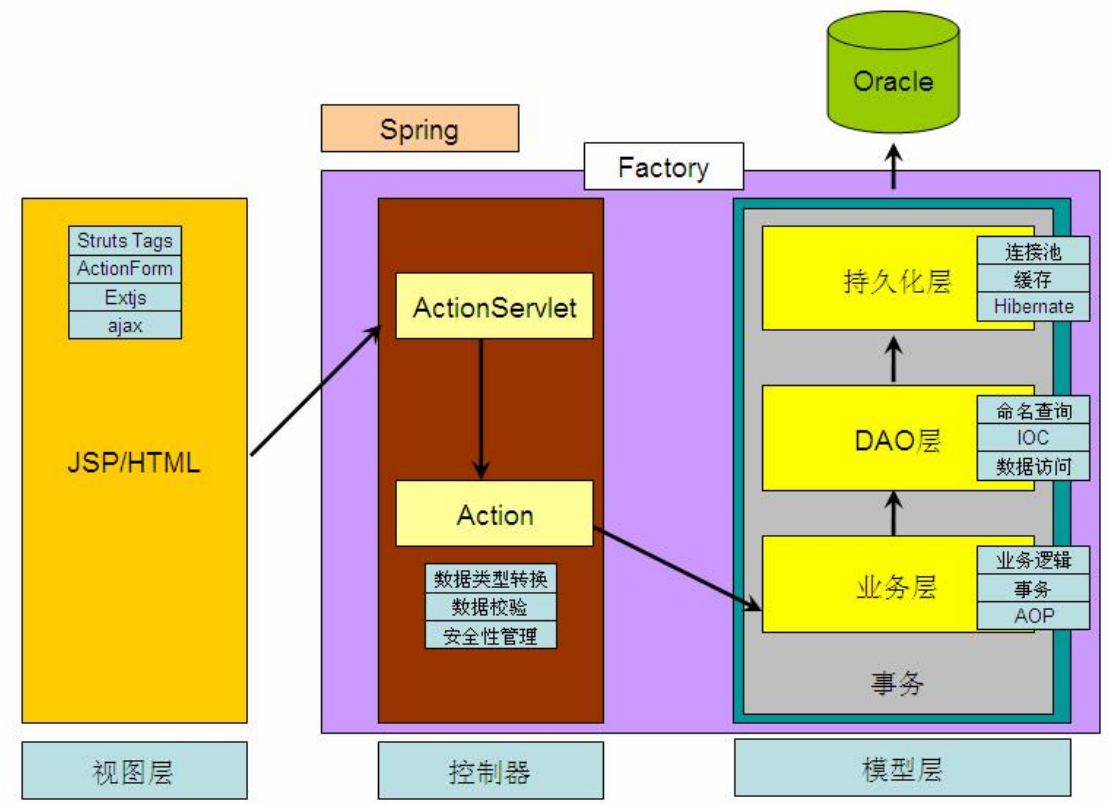
第二十九章：综合项目

一、概述

最近太忙，忙得没时间写文章，这一章为大家提供一个综合项目，希望能为您在学习过程中的难点拨开迷雾。虽然叫综合项目，实际是一个小练习而已，“麻雀虽小，五脏俱全”，项目中的很多地方在前面并没有讲述，而且是一些关键的细节，仔细阅读，必定能带来不少收获。

本项目数据库采用 Oracle10g，持久化层使用 Hibernate3.0，WEB 层使用 Struts1.2，客户界面使用 Extjs2.2，最后使用 Spring2 管理 Bean 和事务，这是一个不折不扣的 SSH+Extjs 的项目。

项目的体系结构如下图：



考虑到这只是一个演示项目，并没有实现太多的功能，我们创建了部门表和员工表，功能上实现了员工的添加、修改、批量删除、列表显示和条件查询等等，基本包含了常见的功能。

本章没有过多的解释，主要是自己的时间有限，希望哪天等我闲点的时候能做更好的补充。所以，只好您自己多看多理解了。另外，本项目没有考虑任何性能、代码重构的问题，有欠妥的地方在所难免，说得不好听点是为技术而技术。

二、数据库设计

本项目共有两张表：部门表和员工表，才者为主外键关系。下面的 SQL 脚本使用 PLSQL Developer 自动生成，并带了一些测试数据：

```
prompt PL/SQL Developer import file
prompt Created on 2009 年 6 月 12 日  by Administrator
set feedback off
set define off
prompt Creating DEPT...
create table DEPT
(
  DID      NUMBER not null,
  DNAME VARCHAR2(20)
)
;
alter table DEPT
  add primary key (DID);

prompt Creating EMPLOYEE...
create table EMPLOYEE
(
  EID          NUMBER not null,
  ENAME       VARCHAR2(20),
  EADDRESS VARCHAR2(200),
  E_DID       NUMBER
)
;
alter table EMPLOYEE
  add primary key (EID);
alter table EMPLOYEE
  add foreign key (E_DID)
  references DEPT (DID);

prompt Deleting EMPLOYEE...
delete from EMPLOYEE;
commit;
prompt Deleting DEPT...
delete from DEPT;
commit;
prompt Loading DEPT...
insert into DEPT (DID, DNAME)
values (1, '市场部');
```

```
insert into DEPT (DID, DNAME)
values (2, '宣传部');
insert into DEPT (DID, DNAME)
values (3, '人力资源部');
insert into DEPT (DID, DNAME)
values (9, '经 XX');
insert into DEPT (DID, DNAME)
values (10, '经 XX');
insert into DEPT (DID, DNAME)
values (12, '经 XX');
commit;
prompt 6 records loaded
prompt Loading EMPLOYEE...
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (143, 'CCCC', 'CCCC', 3);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (3, '王五 SS', '株洲 SSS', 2);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (4, '李世民', 'XXXXXXX', 2);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (16, '中国', '中华人民共和国', 2);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (33, '宋江 XX', '中华人民共和国', 2);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (85, 'DDD', 'SSSSS', 3);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (89, '43543XX', '345435', 3);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (117, '222', '2222XXX', 3);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values (118, 'AAA', 'AAAA33', 3);
insert into EMPLOYEE (EID, ENAME, EADDRESS, E_DID)
values  (120,  '编辑的',  '编辑的员工编辑的员工 WWW 编辑的员工编辑的员工 WWW',
3);
commit;
prompt 10 records loaded
set feedback on
set define on
prompt Done.
```

三、持久层封装

我们定义了一个 Common 类，用于封装数据访问的通用操作，所有的 Hibernate API 全

部封装在这里，而 NamedQueryCommon 类是为了支持 Hibernate 命名查询提供的，本项目查询功能全部使用命名查询实现。

Common.java

```
package com.aptech.presist;

import java.io.Serializable;
import java.sql.SQLException;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.springframework.orm.hibernate3.HibernateCallback;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

/**
 * <b>数据库访问操作：</b><br>
 * 添加<br>
 * 修改<br>
 * 删除<br>
 * 加载<br>
 * 查询（不分页）<br>
 * 分页查询<br>
 * 计算总记录条数<br>
 * 计算总页数<br>
 *
 * @author Administrator
 *
 */
public class Common<POJO> extends HibernateDaoSupport implements
ICommon<POJO> {
    /*
     * (non-Javadoc)
     *
     * @see com.aptech.presist.ICommon#insertPOJO(POJO)
     */
    public void insertPOJO(POJO pojo) throws CommonException {
        this.getHibernateTemplate().save(pojo);
    }

    /*
     * (non-Javadoc)
     *
```

```
 * @see com.aptech.presist.ICommon#updatePOJO(POJO)
 */
public void updatePOJO(POJO pojo) throws CommonException {
	this.getHibernateTemplate().update(pojo);
}

/*
 * (non-Javadoc)
 *
 * @see com.aptech.presist.ICommon#deletePOJO(java.lang.Class,
 *		java.io.Serializable)
 */
public void deletePOJO(Class clazz, Serializable id) throws CommonException {
	this.getHibernateTemplate().delete(
			this.getHibernateTemplate().load(clazz, id));
}

/*
 * (non-Javadoc)
 *
 * @see com.aptech.presist.ICommon#loadPOJO(java.lang.Class,
 *		java.io.Serializable)
 */
public POJO loadPOJO(Class clazz, Serializable id) throws CommonException {
	return (POJO) this.getHibernateTemplate().load(clazz, id);
}

/**
 * 绑定参数值,调用: bindParameters(query, 1 ,5)
 *
 * @param query
 *			"from Dept as d where d.id > ? and d.id < ?"
 * @param objects
 */
void bindParameters(Query query, Object... objects) {
	if (objects == null)
		return;

	int i = 0;
	for (Object object : objects) {
		if (object instanceof java.sql.Date) {
			java.sql.Date sqlDate = (java.sql.Date) object;
			java.util.Date date = new java.util.Date(sqlDate.getTime());
```

```
                query.setParameter(i, date);
            } else {
                query.setParameter(i, object);
            }
            i++;
        }
    }

    List commonQuery(Query query, Object... objects) {
        query.setCacheable(true);// 使用二级缓存
        this.bindParameters(query, objects);
        List list = query.list();
        return list;
    }

    /*
     * (non-Javadoc)
     *
     * @see com.aptech.presist.ICommon#queryPOJOs(java.lang.String,
     *      java.lang.Object)
     */
    public List queryPOJOs(final String hql, final Object... objects)
            throws CommonException {
        return this.getHibernateTemplate().executeFind(new HibernateCallback() {
            public Object doInHibernate(Session session)
                    throws HibernateException, SQLException {
                try {
                    Query query = session.createQuery(hql);
                    return commonQuery(query, objects);
                } catch (HibernateException e) {
                    e.printStackTrace();
                    throw new CommonException(e);
                }
            }
        });
    }

    /*
     * (non-Javadoc)
     *
     * @see com.aptech.presist.ICommon#queryPOJOs(java.lang.Class)
     */
    public List<POJO> queryPOJOs(Class clazz) throws CommonException {
        return this.queryPOJOs("from " + clazz.getName());
```

```
}

List commonQuery(Query query, int curpage, int pagesize, Object... objects) {
	query.setCacheable(true);// 使用二级缓存
	this.bindParameters(query, objects);// 绑定参数
	// 设置分页参数
	query.setFirstResult((curpage - 1) * pagesize);
	query.setMaxResults(pagesize);

	List list = query.list();
	return list;
}

/*
 * (non-Javadoc)
 *
 * @see com.aptech.presist.ICommon#queryPOJOsByPage(java.lang.String, int,
 *      int, java.lang.Object)
 */
public List queryPOJOsByPage(final String hql, final int curpage,
		final int pagesize,
		final Object... objects) throws CommonException {
	return this.getHibernateTemplate().executeFind(new HibernateCallback() {

		public Object doInHibernate(Session session)
				throws HibernateException, SQLException {
			try {
				Query query = session.createQuery(hql);
				return commonQuery(query, curpage, pagesize, objects);
			} catch (HibernateException e) {
				e.printStackTrace();
				throw new CommonException(e);
			}
		}
	});

}

/*
 * (non-Javadoc)
 *
 * @see com.aptech.presist.ICommon#totalPOJOsCount(java.lang.String,
 *      java.lang.Object)
 */
```

```
    public long totalPOJOsCount(String hql, Object... objects)
                throws CommonException {
        List list = this.queryPOJOs(hql, objects);
        return Long.parseLong(list.get(0).toString());
    }

    /*
     * (non-Javadoc)
     *
     * @see com.aptech.presist.ICommon#totalPages(long, int)
     */
    public long totalPages(long totalCount, int pagesize) {
        return totalCount % pagesize == 0 ? totalCount / pagesize : totalCount
                / pagesize + 1;
    }

    ////////////////////////配合 extjs 的查询，因为分页信息和平时有所出入//////////////////
    public List queryPOJOsByPage4Extjs(final String hql, final int start,
                final int limit,
                final Object... objects) throws CommonException{
        int curpage = start / limit + 1;
        return this.queryPOJOsByPage(hql, curpage, limit, objects);
    }
}
```

NamedQueryCommon.java

```
package com.aptech.presist;

import java.sql.SQLException;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.springframework.orm.hibernate3.HibernateCallback;

/**
 * 支持命名查询
 * @author Administrator
 *
 */
public class NamedQueryCommon<POJO> extends Common<POJO> implements
ICommon<POJO>{
    /**
```

```
     * 命名查询
     */
    @Override
    public List queryPOJOs(final String namedQuery, final Object... objects)
            throws CommonException {
        return this.getHibernateTemplate().executeFind(new HibernateCallback(){
            public Object doInHibernate(Session session)
                    throws HibernateException, SQLException {
                Query query = session.getNamedQuery(namedQuery);
                return commonQuery(query, objects);
            }});


    }

    /**
     * 支持分页的命名查询
     */
    @Override
    public List queryPOJOsByPage(final String namedQuery,
            final int curpage, final int pagesize,
            final Object... objects) throws CommonException {
        return this.getHibernateTemplate().executeFind(new HibernateCallback(){
            public Object doInHibernate(Session session)
                    throws HibernateException, SQLException {
                Query query = session.getNamedQuery(namedQuery);
                return commonQuery(query, curpage, pagesize, objects);
            }});
    }
}
```

四、DAO

DAO 定义基本的数据访问操作，简洁但很重要。DeptDao 实现了部门的基本数据访问操作，EmployeeDao 则实现员工的基本数据访问操作。

DeptDao.java

```
package com.aptech.dao.impl;

import java.io.Serializable;
import java.util.List;

import com.aptech.dao.IDeptDao;
```

```
import com.aptech.pojo.Dept;
import com.aptech.presist.ICommon;

public class DeptDao implements IDeptDao {
    private ICommon<Dept> common;

    public void setCommon(ICommon<Dept> common) {
        this.common = common;
    }

    public Dept loadDept(Serializable id){
        return common.loadPOJO(Dept.class, id);
    }

    public List<Dept> queryAll(){
        return common.queryPOJOs("query.dept");
    }
}
```

EmployeeDao.java

```
package com.aptech.dao.impl;

import java.io.Serializable;
import java.util.List;

import com.aptech.dao.IEmployeeDao;
import com.aptech.pojo.Employee;
import com.aptech.presist.ICommon;

public class EmployeeDao implements IEmployeeDao {
    private ICommon<Employee> common;

    public void setCommon(ICommon<Employee> common) {
        this.common = common;
    }

    public void insertEmployee(Employee employee){
        common.insertPOJO(employee);
    }

    public void updateEmployee(Employee employee){
        common.updatePOJO(employee);
    }
```

```
    public void deleteEmployee(Serializable id){
        common.deletePOJO(Employee.class, id);
    }

    public Employee loadEmployee(Serializable id){
        return common.loadPOJO(Employee.class, id);
    }

    public List<Employee> queryByPage(int start, int limit){
        return common.queryPOJOsByPage4Extjs("query.employee.by.page.for.extjs",
start, limit);
    }

    public long totalLines(){
        return common.totalPOJOsCount("query.employee.count");
    }

    public List<Employee> queryByEname(int start, int limit, String ename){
        return common.queryPOJOsByPage4Extjs("query.employee.by.ename.for.extjs",
start, limit, "%" + ename + "%");
    }

    public long totalLinesByEname(String ename){
        return common.totalPOJOsCount("query.employee.count.by.ename", ename);
    }
}
```

DAO 使用了若干个命名查询，命名查询定义在 query.hbm.xml 文件中。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <query name="query.employee.by.page.for.extjs">
        <![CDATA[
            from Employee as e left outer join fetch e.dept
        ]]>
    </query>

    <query name="query.employee.count">
        <![CDATA[
            select count(*) from Employee e
        ]]>
```

```
    </query>

    <query name="query.employee.by.ename.for.extjs">
        <![CDATA[
            from Employee as e left outer join fetch e.dept where e.ename like ?
        ]]>
    </query>

    <query name="query.employee.count.by.ename">
        <![CDATA[
            select count(*) from Employee e where e.ename like ?
        ]]>
    </query>

    <query name="query.dept">
        <![CDATA[
            from Dept
        ]]>
    </query>
</hibernate-mapping>
```

五、业务层

业务层实现了与员工有关的基本业务，如添加员工，修改员工信息，删除员工信息和查询。

EmployeeService.java

```
package com.aptech.service.impl;

import java.util.ArrayList;
import java.util.List;

import com.aptech.dao.IDeptDao;
import com.aptech.dao.IEmployeeDao;
import com.aptech.pojo.Dept;
import com.aptech.pojo.Employee;
import com.aptech.service.IEmployeeService;
import com.aptech.util.TypeUtils;
import com.aptech.util.VoPoConverter;
import com.aptech.vo.DeptVo;
import com.aptech.vo.EmployeePager;
import com.aptech.vo.EmployeeVo;
```

```
public class EmployeeService implements IEmployeeService {
    private IEmployeeDao edao;
    private IDeptDao ddao;

    public void setDdao(IDeptDao ddao) {
        this.ddao = ddao;
    }

    public void setEdao(IEmployeeDao edao) {
        this.edao = edao;
    }

    /**
     * 查找所有的部门
     * @return
     */
    public List<DeptVo> queryAllDept(){
        List<Dept> listPo = ddao.queryAll();
        List<DeptVo> listVo = new ArrayList<DeptVo>();

        DeptVo deptVo = null;
        for(Dept dept : listPo){
            listVo.add(VoPoConverter.copyProperties(dept, DeptVo.class));
        }
        return listVo;
    }

    /**
     * 保存员工
     * @param employeeVo
     */
    public void insertEmployee(EmployeeVo employeeVo){
        Dept dept = ddao.loadDept(employeeVo.getDid());
        Employee employee = VoPoConverter.copyProperties(employeeVo,
Employee.class);
        employee.setDept(dept);

        edao.insertEmployee(employee);
    }

    /**
     * 分页查询所有的数据
     * @param start
     * @param limit
```

```
     * @return EmployeePager,包含了当前页的数据和总记录条数
     */
    public EmployeePager queryByPage(int start, int limit){
        List<Employee> listPo = edao.queryByPage(start, limit);
        List<EmployeeVo> listVo = this.po2vo(listPo);
        long totalLines = edao.totalLines();
        return new EmployeePager(totalLines, listVo);
    }

    /**
     * 根据员工姓名进行查询
     * @param start
     * @param limit
     * @param ename
     * @return
     */
    public EmployeePager queryByEname(int start, int limit, String ename){
        List<Employee> listPo = edao.queryByEname(start, limit, ename);
        List<EmployeeVo> listVo = this.po2vo(listPo);
        long totalLines = edao.totalLinesByEname(ename);
        return new EmployeePager(totalLines, listVo);
    }

    /**
     * 根据 ID 删除员工
     * @param id
     */
    public void deleteEmployeeById(String[] ids){
        long[] longIds = TypeUtils.stringArray2LongArray(ids);
        for(int i = 0; i < longIds.length; i ++){
            edao.deleteEmployee(longIds[i]);
        }
    }

    /**
     * 修改员工信息
     * @param employeeVo
     */
    public void updateEmployee(EmployeeVo employeeVo){
        Dept dept = ddao.loadDept(employeeVo.getDid());
        Employee employee = VoPoConverter.copyProperties(employeeVo,
Employee.class);
        employee.setDept(dept);
```

```
            edao.updateEmployee(employee);
        }


        private List<EmployeeVo> po2vo(List<Employee> listPo){
            List<EmployeeVo> listVo = new ArrayList<EmployeeVo>();
            if(listPo == null) return listVo;
            EmployeeVo employeeVo = null;

            for(Employee employee : listPo){
                employeeVo = VoPoConverter.copyProperties(employee, EmployeeVo.class);
                employeeVo.setDid(employee.getDept().getDid());
                employeeVo.setDname(employee.getDept().getDname());

                listVo.add(employeeVo);
            }
            return listVo;
        }

}
```

EmployeePager 类是为了迎合 Extjs 分页定义的，保存了 totalProperty 和 root 两个属性。

EmployeePager.java

```
package com.aptech.vo;

import java.util.List;

/**
 * 保存当前页的员工信息和总记录条数
 *
 * @author Administrator
 *
 */
public class EmployeePager {
    private long totalProperty;
    private List<EmployeeVo> root;

    public EmployeePager() {
        // TODO Auto-generated constructor stub
    }

    public EmployeePager(long totalProperty, List<EmployeeVo> root) {
        super();
```

```
        this.totalProperty = totalProperty;
        this.root = root;
    }

    public long getTotalProperty() {
        return totalProperty;
    }

    public void setTotalProperty(long totalProperty) {
        this.totalProperty = totalProperty;
    }

    public List<EmployeeVo> getRoot() {
        return root;
    }

    public void setRoot(List<EmployeeVo> root) {
        this.root = root;
    }

}
```

六、控制器 Action

功能有限，所有只定义了两个 Action，一个为部门提供控制访问，一个为员工提供控制访问，都继承自 DispatchAction。查询时，控制器返回 json 格式的数据，这是通过 json-lib 开源工具进行转换的。

DeptAction.java

```
package com.aptech.struts.actions;

import java.io.PrintWriter;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import net.sf.json.JSONArray;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;
```

```
import com.aptech.service.IEmployeeService;
import com.aptech.vo.DeptVo;

public class DeptAction extends DispatchAction {
private IEmployeeService es;

    public void setEs(IEmployeeService es) {
        this.es = es;
    }

    public ActionForward queryDepts(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        List<DeptVo> list = es.queryAllDept();
        JSONArray jArray = JSONArray.fromObject(list);
        String json = jArray.toString();

        PrintWriter out = response.getWriter();
        out.println(json);
        out.flush();
        out.close();
        return null;
    }
}
```

EmployeeAction.java

```
package com.aptech.struts.actions;

import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import net.sf.json.JSONObject;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import com.aptech.service.IEmployeeService;
import com.aptech.struts.form.EmployeeForm;
import com.aptech.vo.EmployeePager;
```

```
public class EmployeeAction extends DispatchAction {
    private IEmployeeService es;

    public void setEs(IEmployeeService es) {
        this.es = es;
    }

    /**
     * 添加新员工
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return
     * @throws Exception
     */
    public ActionForward add(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        PrintWriter out = response.getWriter();
        try {
            EmployeeForm ef = (EmployeeForm) form;
            es.insertEmployee(ef.getEmployeeVo());
            out.println("{success:true, msg:'恭喜，员工信息添加成功！'}");
        } catch (Exception e) {
            e.printStackTrace();
            out.println("{success:false, msg:'对不起，员工信息添加失败！'}");
        }finally{
            out.flush();
            out.close();
        }

        return null;
    }

    /**
     * 分页查询所有的员工数据
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return
     * @throws Exception
```

```
     */
    public ActionForward query(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
            PrintWriter out = response.getWriter();
            int start = Integer.parseInt(request.getParameter("start"));
            int limit = Integer.parseInt(request.getParameter("limit"));
            EmployeePager employeePager = es.queryByPage(start, limit);
            String json = JSONObject.fromObject(employeePager).toString();
            out.println(json);
            out.flush();
            out.close();

            return null;
    }

    /**
     * 根据员工姓名查询
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return
     * @throws Exception
     */
    public ActionForward queryByEname(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
            PrintWriter out = response.getWriter();
            int start = Integer.parseInt(request.getParameter("start"));
            int limit = Integer.parseInt(request.getParameter("limit"));
            String ename = request.getParameter("ename");
            //ename = new String(ename.getBytes("iso-8859-1"), "utf-8");
            System.out.println(ename);

            EmployeePager employeePager = es.queryByEname(start, limit, ename);
            String json = JSONObject.fromObject(employeePager).toString();
            out.println(json);
            out.flush();
            out.close();

            return null;
    }
```

```
/**
 * 批量删除员工
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward deleteByIds(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
    PrintWriter out = response.getWriter();
    try {
        String[] ids = request.getParameterValues("ids");
        es.deleteEmployeeById(ids);
        out.println("{success:true, msg:'恭喜，员工信息删除成功！'}");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        out.println("{success:false, msg:'对不起，员工信息删除失败！'}");
    } finally{
        out.flush();
        out.close();
    }

    return null;
}

/**
 * 修改员工信息
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward edit(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
    PrintWriter out = response.getWriter();
    try {
        EmployeeForm ef = (EmployeeForm) form;
```

```
                es.updateEmployee(ef.getEmployeeVo());
                out.println("{success:true, msg:'恭喜，员工信息修改成功！'}");
            } catch (Exception e) {
                e.printStackTrace();
            }finally{
                out.flush();
                out.close();
            }

            return null;
        }
}
```

EmployeeAction 类关联的 ActionForm 叫 EmployeeForm，该类代码如下：

EmployeeForm.java

```
package com.aptech.struts.form;

import org.apache.struts.action.ActionForm;

import com.aptech.vo.EmployeeVo;

public class EmployeeForm extends ActionForm {
    private EmployeeVo employeeVo = new EmployeeVo();

    public EmployeeVo getEmployeeVo() {
        return employeeVo;
    }

    public void setEmployeeVo(EmployeeVo employeeVo) {
        this.employeeVo = employeeVo;
    }

}
```

EmployeeForm 类关联 EmployeeVo，这样可以将数据直接放到 VO 对象中，减少了一次转换。

EmployeeVo.java

```
package com.aptech.vo;

public class EmployeeVo {
    private Long eid;
    private String ename;
```

```
    private String eaddress;

    private Long did; //  部门 ID
    private String dname; //  部门名称

    public EmployeeVo() {
        // TODO Auto-generated constructor stub
    }

    public EmployeeVo(String ename, String eaddress, Long did) {
        super();
        this.ename = ename;
        this.eaddress = eaddress;
        this.did = did;
    }

    public EmployeeVo(Long eid, String ename, String eaddress, Long did,
            String dname) {
        super();
        this.eid = eid;
        this.ename = ename;
        this.eaddress = eaddress;
        this.did = did;
        this.dname = dname;
    }

    public Long getEid() {
        return eid;
    }

    public void setEid(Long eid) {
        this.eid = eid;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public String getEaddress() {
        return eaddress;
```

```
    }

    public void setEaddress(String eaddress) {
        this.eaddress = eaddress;
    }

    public Long getDid() {
        return did;
    }

    public void setDid(Long did) {
        this.did = did;
    }

    public String getDname() {
        return dname;
    }

    public void setDname(String dname) {
        this.dname = dname;
    }

}
```

七、Spring 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xmlns:tx="http://www.springframework.org/schema/tx"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="
            http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
            http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
            http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">


    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
        <property name="configLocation"
            value="classpath:hibernate.cfg.xml">
        </property>
    </bean>

    <bean id="hibernateTemplete"
class="org.springframework.orm.hibernate3.HibernateTemplate">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>

    <bean id="comm" class="com.aptech.presist.NamedQueryCommon">
        <property name="hibernateTemplate">
            <ref bean="hibernateTemplete"/>
        </property>
    </bean>

    <bean id="ddao" class="com.aptech.dao.impl.DeptDao">
        <property name="common">
            <ref bean="comm"/>
        </property>
    </bean>

    <bean id="edao" class="com.aptech.dao.impl.EmployeeDao">
        <property name="common">
            <ref bean="comm"/>
        </property>
    </bean>

    <bean id="employeeService" class="com.aptech.service.impl.EmployeeService">
        <property name="ddao">
            <ref bean="ddao"/>
        </property>
        <property name="edao">
            <ref bean="edao"/>
        </property>
    </bean>

    <bean name="/employee" class="com.aptech.struts.actions.EmployeeAction">
        <property name="es">
            <ref bean="employeeService"/>
        </property>
    </bean>
```

```
    <bean name="/dept" class="com.aptech.struts.actions.DeptAction">
        <property name="es">
            <ref bean="employeeService"/>
        </property>
    </bean>

    <!-- 事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>

    <!-- 指定环绕通知并指定事务的传播行为 -->
    <tx:advice id="advice-id" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>

    <aop:config>
        <aop:pointcut id="pointcut-id" expression="execution(public *
com.aptech.service.*.*(..))"/>
        <aop:advisor advice-ref="advice-id" pointcut-ref="pointcut-id"/>
    </aop:config>

</beans>
```

八、主界面

主界面通过 Viewport 来布局，左边是功能导航树，主体部分是 TabPanel，上下两个区域与功能无关。主界面充分利用了 Extjs 提供的面板组件构造出一个大同世界哈。

viewport.js

```
Ext.BLANK_IMAGE_URL = "extjs/resources/images/default/s.gif";
Ext.QuickTips.init();//启动悬停提示
var left;

Ext.onReady(function(){
    //创建北面的面板
    var top = new Ext.Panel({
```

```
        region: "north",
        title: "LOGO",
        height: 80,
        html: "这里放 LOGO"
    });

    //左边的树
    left = new Ext.tree.TreePanel({
        region: "west",
        title: "功能导航",
        collapsible: true,
        split: true,
        containerScroll: true,
        autoScroll: true,
        width: 200,
        listeners: {
            dblclick: function(n){
                //alert(n.attributes.url);
                var url = n.attributes.url;
                var id = n.attributes.id;
                if(url){
                    if(center.getItem(id)){
                        //表示标签已打开,则激活
                        center.setActiveTab(id);
                    }else{
                        //表示标签还没有打开,创建新页面
                        //有 url 才打开页面
                        var p = new Ext.Panel({
                            title: n.attributes.text, //标题就是节点的文本
                            id: id, //标签的 ID 和节点的 ID 一样
                            autoLoad: {url: url, scripts: true},
                            closable: true
                        });
                        center.add(p);
                        center.setActiveTab(p);
                    }
                }
            }
        }
    });

    var root = new Ext.tree.TreeNode({id: "1", text: "员工管理系统", leaf: "false"});
    var addEmp = new Ext.tree.TreeNode({text: "新增员工", url: "addemp.html"});
    var emplist = new Ext.tree.TreeNode({text: "员工信息维护", url: "emplist.html"});
```

```
    root.appendChild([addEmp, emplist]);
    left.setRootNode(root);

    //中间的选项面板
    var center = new Ext.TabPanel({
        region: "center",
        defaults: {autoScroll:true},//自动出现滚动条
        items:[{
            title: "首页",
            html: "欢迎使用本系统！",
            id: "index"
        }],
        enableTabScroll: true
    });
    center.setActiveTab("index");

    //底部的面板
    var bottom = new Ext.Panel({
        region: "south",
        html: "版权所有，翻版必究",
        bodyStyle: "padding: 10px;text-align: center; font-size:12px"
    });


    var vp = new Ext.Viewport({
        layout: "border",
        items: [top, left, center, bottom]
    });

    left.expandAll();
})
```

注意理解加粗部分的代码，这是一段打开选项卡控件标签的代码，如果标签已打开，则激活，如果没有打开，则重新创建。

九、添加新员工

定义一个表单面板，通过 ajax 提供到控制器进行处理。

addemp.js

```
Ext.onReady(function(){
    var txtName = new Ext.form.TextField({
        fieldLabel: "员工姓名",
```

```
        name: "employeeVo.ename",
        allowBlank: false
    });

    var areaAddress = new Ext.form.TextArea({
        fieldLabel: "现住地址",
        name: "employeeVo.eaddress",
        allowBlank: false,
        width: 500
    });

    //员工所在的部门
    var proxy = new Ext.data.HttpProxy({url: "dept.do?method=queryDepts"});
    var reader = new Ext.data.JsonReader({}, [
        {name: "rec_did", type: "int", mapping: "did"},
        {name: "rec_dname", type: "string", mapping: "dname"}
    ]);
    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader,
        autoLoad: true
    });

    var cmbDept = new Ext.form.ComboBox({
        store: store,
        fieldLabel: "所在部门",
        displayField: "rec_dname",
        valueField: "rec_did",
        mode: "remote",
        triggerAction: "all",
        emptyText: "请选择所在的部门",
        allowBlank: false,
        editable: false,
        hiddenName: "employeeVo.did" //请注意，设置该选项才能将实际值传送到服务
器
    });

    var form = new Ext.form.FormPanel({
        title: "新增员工",
        frame: true,
        items: [txtName, cmbDept, areaAddress],
        url: "employee.do",
        baseParams: {method: "add"},
        method: "post",
```

```
        renderTo: "addemp",
        buttons: [{
            text: "提交数据",
            handler: function(){
                var json = {
                    success: function(f, action){
                        Ext.Msg.alert("成功", action.result.msg);
                    },
                    failure: function(){
                        Ext.Msg.alert("失败", "对不起，操作失败，请检查数据是否
完整！");
                    }
                };
                form.getForm().submit(json);
            }
        },{text: "重置",
            handler: function(){
                var basicForm = form.getForm();
                basicForm.reset();
            }
        }]
    });

})
```

十、员工信息维护

这个 js 文件中包括的内容很多，包括修改、删除和查询。修改时，为了便于理解，将表单面板的代码重复了一次。您可以使用 OOP 对该代码重构，使之更加优雅。

emplist.js

```
Ext.onReady(function(){
    var proxy = new Ext.data.HttpProxy({url: "employee.do?method=query"});
    var Employee = Ext.data.Record.create([
        {name: "rec_eid", type: "int", mapping: "eid"},
        {name: "rec_ename", type: "string", mapping: "ename"},
        {name: "rec_eaddress", type: "string", mapping: "eaddress"},
        {name: "rec_did", type: "int", mapping: "did"},
        {name: "rec_dname", type: "string", mapping: "dname"}
    ]);

    var reader = new Ext.data.JsonReader({
        totalProperty: "totalProperty",
```

```
        root: "root"}, Employee
    );

    var store = new Ext.data.Store({
        proxy: proxy,
        reader: reader
    });

    store.load({params: {start:0, limit: 5}});

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

    var cm = new Ext.grid.ColumnModel([
        new Ext.grid.RowNumberer(),
        sm,
        {header: "姓名", width: 110, dataIndex: "rec_ename"},
        {id: "address", header: "地址", dataIndex: "rec_eaddress"},
        {header: "所属部门", width: 110, dataIndex: "rec_dname"}
    ]);

    var pageToolbar = new Ext.PagingToolbar({
        store: store,
        pageSize: 5,
        displayInfo: true,
        displayMsg: "当前显示从{0}条到{1}条,共{2}条",
        emptyMsg:  "<span  style='color:red;font-style:italic;'>对不起,没有找到数据
</span>"
    });

    var grid = new Ext.grid.GridPanel({
        store: store,
        autoExpandColumn: "address",
        cm: cm,
        sm: sm,
        bbar: pageToolbar,
        autoHeight: true,
        bodyStyle: "width: 100%",
        autoWidth: true,
        tbar:[{
            text: "新建员工",
            icon: "images/add.png",
            cls: "x-btn-text-icon",
            handler: function(){
                var path = "/员工管理系统/新增员工";
```

```
                left.selectPath(path, "text", function(x, node){
                    left.fireEvent("dblclick", node);//触发双击事件
                });
            }
        },{
            text: "修改员工",
            icon: "images/application_edit.png",
            cls: "x-btn-text-icon",
            handler: function(){
                //判断是否有选择行
                var selModel = grid.getSelectionModel();
                var record;//选择的行的数据
                if(!selModel.hasSelection()){
                    Ext.Msg.alert("错误", "请选择要修改的行!");
                }else if(selModel.getSelections().length > 1){
                    Ext.Msg.alert("错误", "一次只能修改一行,不行同时选择多行!");
                }else{
                    record = selModel.getSelected();


                    ////////////////////////////////////////////
                    //定义隐藏表单域保存修改员工的 ID
                    var hiddenEid = new Ext.form.Hidden({
                        name: "employeeVo.eid"
                    });

                    var txtName = new Ext.form.TextField({
                        fieldLabel: "员工姓名",
                        name: "employeeVo.ename",
                        allowBlank: false
                    });

                    var areaAddress = new Ext.form.TextArea({
                        fieldLabel: "现住地址",
                        name: "employeeVo.eaddress",
                        allowBlank: false,
                        width: 330
                    });

                    //员工所在的部门
                    var proxy = new Ext.data.HttpProxy({url:
"dept.do?method=queryDepts"});
                    var reader = new Ext.data.JsonReader({}, [
                        {name: "rec_did", type: "int", mapping: "did"},
```

```
                {name: "rec_dname", type: "string", mapping: "dname"}
            ]);
            var store2 = new Ext.data.Store({
                proxy: proxy,
                reader: reader,
                autoLoad: true,
                listeners:{
                    load: function(){
                        cmbDept.setValue(record.get("rec_did"));
                    }
                }
            });

            var cmbDept = new Ext.form.ComboBox({
                store: store2,
                fieldLabel: "所在部门",
                displayField: "rec_dname",
                valueField: "rec_did",
                mode: "remote",
                triggerAction: "all",
                allowBlank: false,
                emptyText: "请选择所在的部门",
                allowBlank: false,
                editable: false,
                name: "employeeVo.did",
                hiddenName: "employeeVo.did" //请注意，设置该选项才能将
实际值传送到服务器
            });

            var form = new Ext.form.FormPanel({
                frame: true,
                items: [hiddenEid, txtName, cmbDept, areaAddress],
                url: "employee.do",
                baseParams: {method: "edit"},
                method: "post",
                buttons: [{
                    text: "修改",
                    handler: function(){
                        var json = {
                            success: function(f, action){
                                Ext.Msg.alert("成功", action.result.msg);
                                store.reload();
                                win.close();
                            },
```

```
                                        failure: function(){
                                            Ext.Msg.alert("失败", "对不起，操作失败，
请检查数据是否完整！");
                                        }
                                    };
                                    form.getForm().submit(json);
                                }
                            },{text: "关闭",
                                handler: function(){
                                    win.close();
                                }
                            }]
                        });

                        var win = new Ext.Window({
                            title: "修改员工",
                            id: "edit",
                            width: 500,
                            modal: true,
                            autoHeight: true,
                            items: [form]
                        });

                        win.show("editEmployee");

                        //要等窗口出现之后才能初始化
                        form.getForm().setValues({
                            "employeeVo.eid": record.get("rec_eid"),
                            "employeeVo.ename": record.get("rec_ename"),
                            "employeeVo.eaddress": record.get("rec_eaddress")
                        });
                    }
                    ////////////////////////////////////////////
                }
            },{
                icon: "images/cross.png",
                cls: "x-btn-text-icon",
                text: "批量删除",
                handler: function(){
                    var selModel = grid.getSelectionModel();
                    if(selModel.hasSelection()){
                        Ext.Msg.confirm("确认", "您确定要删除选择的记录吗?",
function(btn){
                            if(btn == "yes"){
```

```
                                //选择了行
                                var records = selModel.getSelections();
                                var ids = [];
                                for(var i = 0; i < records.length; i ++){
                                    ids.push(records[i].get("rec_eid"));
                                }
                                Ext.Ajax.request({
                                    url: "employee.do?method=deleteByIds",
                                    params: {ids: ids},
                                    method: "post",
                                    success: function(response){
                                        var json = Ext.util.JSON.decode(response.responseText);
                                        Ext.MessageBox.alert("结果", json.msg);
                                    },
                                    failure: function(){
                                        Ext.Msg.alert("结果", "对不起,删除失败!!!!!");
                                    }
                                });

                                store.reload();//更新页面
                            }
                        });
                    }else{
                        Ext.Msg.alert("错误", "请选择要删除的行!");
                    }
                }
            },{
                icon: "images/arrow_refresh.png",
                cls: "x-btn-text-icon",
                text: "查看全部",
                handler: function(){
                    store.proxy = proxy;
                    //store.load({params: {start:0, limit: 5}});
                    store.reload();
                }
            }, new Ext.Toolbar.Fill(),
            new Ext.Toolbar.TextItem("搜索:"),
            new Ext.form.TriggerField({
                id: "keyword",
                triggerClass: "x-form-search-trigger", //在文本框后添加搜索按钮
                emptyText: "请输入员工姓名",
                onTriggerClick: function(){
                    //alert("dd");
```

```
                    var v = Ext.get("keyword").getValue();
                    var searchProxy = new Ext.data.HttpProxy({url:
"employee.do?method=queryByEname"});
                    store.proxy = searchProxy;
                    store.load({params: {start:0, limit: 5, ename: v}});
                    //store.reload();
                }
            })]
    });

    grid.render("emplist");
})
```

注意加粗部分，这是条件查找的关键代码。

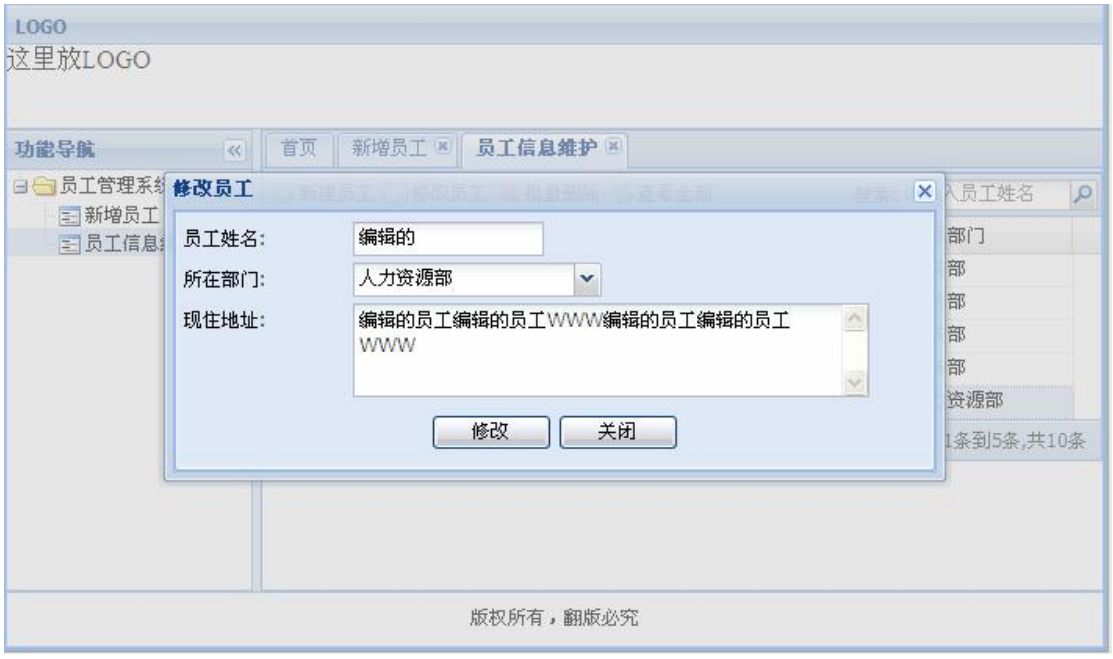
十一、效果图

这一章弄得太简陋了，我都不好意思面对江东父老，上面的代码并不完整，只是把实现思路整理出来了，如果想要源代码的，给我发邮件吧，我把这一章的综合项目源代码给您发过去。邮箱地址在页眉上。下面上几个图让您了解一下我做了什么。

新增员工界面：

员工信息维护界面：

修改界面：



十二、小结

废话不多说了，把项目中要注意的问题简单说一下。

1、ComboBox 将值上传到服务器的问题：一般的组件设置 name 选项后，在服务器通过 request.getParameter()方法即可获取到值，但 ComboBox 不一样，设置 name 后，传到服务器的总是显示值，实际值必须指定 hiddenName。代码如下：

```
var cmbDept = new Ext.form.ComboBox({
    store: store,
    fieldLabel: "所在部门",
    displayField: "rec_dname",
    valueField: "rec_did",
    mode: "remote",
    triggerAction: "all",
    emptyText: "请选择所在的部门",
    allowBlank: false,
    editable: false,
    name: "employeeVo.did",
    forceSelection: true,
    hiddenName: "employeeVo.did" //请注意，设置该选项才能将实际值传送到服务器
});
```

2、修改操作时初始化 ComboBox：当 ComboBox 异步加载数据时，初始化的结果总是不对，原因是因为 Store 的数据尚未加载完成便已执行初始化动作，这和逻辑不吻合，解决方案是处理 Store 的 load 事件。

```
var store = new Ext.data.Store({
    proxy: proxy,
    reader: reader,
    autoLoad: true,
    listeners: {
        load: function(){
            if(record){
                cmbDept.setValue(record.get("rec_did"));
            }
        }
    }
});
```

3、面板工具栏的搜索功能，这个做得很到位，当然换来的结果是千篇一律。

```
new Ext.form.TriggerField({
id: "keyword",
triggerClass: “x-form-search-trigger”, //出现搜索按钮
emptyText: "请输入员工姓名",
onTriggerClick: function(){
var value = Ext.fly(“keyword”).getValue(); //获取文本框内的值
//下面的语句完成搜索并将 GridPanel 中的值重新加载
var searchProxy = new Ext.data.HttpProxy({url:
    "employee.do?method=queryByEname" });
store.proxy = searchProxy;
```

```
store.load({params: {start: 0, limit: 5, ename: value}});
}
})
```

效果图如下：

4、使用代码触发事件，如触发树形面板的双击事件：

```
doubleClickNode: function(xpath){
    left.selectPath(xpath, "text", function(f, node){
        left.fireEvent("dblclick", node); //模拟树的双击事件
    });
},
```