Ext JS 高级程序设计

黄灯桥　徐会生　著

机械工业出版社
China Machine Press

本书特色

（1）权威：2 位资深 Ext 专家合作撰写，5 大专业网站联袂推荐

（2）新颖：基于 Ext JS 3.x 最新版撰写，各种新特性一览无余

（3）深入：透彻阐述 Ext Core、Store 和 Ext.Direct 等杀手锏应用的各种功能和特性

（4）全面：Ext UI、Ext 扩展、Ext 插件、调试等的原理与方法尽含其中

（5）实用：包含 2 个极具商业价值的完整实例，同时用 Java 和.NET 实现

本书赞誉

上将军（黄灯桥）和临远（徐会生）都是 Ext 领域的先驱和专家，而且都曾出版过 Ext 方面的专著，本书的权威性是毋庸置疑的。本书内容新颖，所有内容都基于 Ext 最新版撰写；针对性强，重点讲解了 Ext JS 的核心知识和高级技巧，以及开发者在开发过程中容易碰到的问题，值得所有中高级 Ext 开发者认真阅读。

——Ajax 中国（www.okajax.com）

Ext 的更新速度非常快，自 3.0 版以来增加了大量新特性，功能更强大，从而也更受开发者青睐。Ext Core 和 Ext.Direct 等算得上是 Ext 3.0 中的革命性变化了，它们让 Ext 脱胎换骨，然而这方面的优秀参考资料极少，本书很好地弥补了这一空白。极力推荐！

——17Ext（www.17ext.com）

如果想成为一位 Ext 高手，想要把 Ext 的作用发挥到极致，就必须熟练掌握 Ext UI、Ext 扩展、Ext 插件的原理和使用方法，以及 Ext 应用的调试方法和技巧。本书结合实例，对这些知识进行了全面而详细的讲解，建议所有 Ext 开发者都看一看。

——Dojo 中国（www.dojochina.com）

包含大量实例、可操作性强是本书的一大亮点之一。不仅所有知识点都依托实例来讲解，而且还包含两个可以复用的综合性案例，其中用单页面实现的仓库管理系统的实例颇具学习和研究价值，这可能是很多 Ext 爱好者们都希望看到的。还有一点要重点提及的是，本书所有的大型实例都以迭代的方式给出了完整的代码，而且同时包含 Java 和.NET 两个版本。

——Ext JS 中国用户组

前言

经过两年的发展，Ext JS 从 2.0 版开始，功能逐渐完善，越来越受用户欢迎。现在的框架如此之多，相比其他框架，为什么只有 Ext JS 那么火？笔者认为，其主要原因大概有以下几点：

- 架构简单，容易扩展。
- 控件比较完整。
- 数据管理与界面显示分离。
- 代码书写简单，易于掌握。

今年，Ext JS 不但推出了 3.0 版本，而且推出了 Ext Core，准备在 Web 2.0 网站开发中占一席之地。Ext Core 目前的亮点不多，但经过一段时间的发展后，应该会有不错的成绩。Ext JS 3.0 有相当大的改进，譬如在 Ext JS 2.x 版本中为人所诟病的速度问题在 Ext JS 3.0 中就有所改善。不过，最革命性的改变还是 Ext JS 中新增的 Ext.Direct 功能，它实现了服务器端的无关性。

Ext JS 的未来

根据 Ext JS 的开发路线图，在 Ext JS 3.1 版中将增加可分组的列标题——一个非常值得期待的功能。在 3.1 版中，还将增加 Tree Grid 控件，Store 将增加多字段过滤功能，这些都是非常实用的功能。这将使 Ext 的控件越来越接近 C/S 时代的 RAD 工具的控件。

在 3.2 版本中，Ext 将增加移动组件，进军移动市场，这将是一次革命性的改进。同时在 4.0 版本中，除了对 HTML 5 的支持外，还增加画布功能。

还有一点更值得期待，就是 Ext 的 RAD 开发工具也在开发当中。估计不久之后，也可以像 Delphi、VB 一样，通过拖拽的方式即可轻松开发 Web 应用。

Ext JS 在发展过程中不仅一步步地巩固着自己在 HTML、CSS、JavaScript 领域无可比拟的优势，而且已经开始向相关领域发展扩张。例如，它从 2.0.2 版开始为 Adobe 的 RIA 技术 AIR 提供支持，并且为 GWT 开发了 Ext GWT 2.0，这些都体现了 Ext JS 的强大活力和生命力。

在可预见的未来，Ext JS 将会甩开对手，大踏步向前。不过，从 3.1 版本开始，Ext JS 开始收费，这个对开发社团会造成什么样的影响，以及是否会影响 Ext JS 的未来，还有待观察。

为什么写这本书

Ext Core 是 Ext 小组新推出的用于 Web 2.0 网站开发的核心包，其包含什么功能以及如何使

用这些功能，是大家比较关心的。本书将通过实例的形式为大家详细讲述 Ext Core 的功能及其使用方法。

在 Ext JS 3.0 版中，还增加了不少新的控件和功能，如何熟练地运用它们，也是大家比较关心的，本书也将通过实例的形式为大家详细讲述这些功能及其使用方法。尤其是 Ext.Direct，它将是本书介绍的重中之重。

在目前的 Ext 书籍中，涉及 Ext 完整示例的书少之又少，尤其是单页面应用的示例，可以说，基本上没有。鉴于这种现状，笔者萌生了写一本以示例为主的书的想法，以解读者之渴。

为了顾及.NET 和 Java 两大技术阵营的读者，本书的主要实例都将使用 C#和 Java 两种语言实现。

本书面向的读者

- 有一定使用 Ext JS 经验的中高级读者。
- 想利用 Ext Core 进行 Web 开发的读者。
- 网站建设及网站维护人员。
- 网页设计和制作人员。
- 大中专院校的学生。

如何阅读本书

如果你只对 Ext 3.0 版本中的内容感兴趣，可直接从第 5 章开始阅读。如果只对 Ext Core 有兴趣，建议重点阅读前面 4 章。如果只对 Ext JS 3.0 新增的 Ext.Direct 有兴趣，可直接从第二部分开始阅读，然后阅读第六部分。

在阅读与 Ext JS 调试相关的章节时，建议边读边在电脑上进行测试，这样可以加深对书中内容的理解。

Java 或.NET 的开发人员可根据自己掌握的语言选择相应的章节阅读。

第一部分 Ext Core

第二部分 Store 和 Direct

第三部分

第四部分 Ext 扩展和 Ext 插件

第五部分 调试

第六部分 实例

5.2 DIY 一个 Ext.Direct 实例

通过第一节的介绍，应该对如何在.NET 中使用 Ext.Direct 有了一定了解，现在我们可以尝试 DIY 一个例子来实践一下。例子要实现的功能很简单，在页面中放入一个按钮，单击该按钮后从服务器端获取字符串“Hello”，然后使用 Ext 提示窗口显示出来。

在 VS 2008 中新建一个网站“MyFirstSample”，在右边“解决方案资源管理器”的根节点上单击鼠标右键，从“添加 ASP.NET 文件夹”菜单中选择“bin”子菜单。然后将 Ext.Direct.dll 和 Newtonsoft.Json.dll 两个文件添加到 bin 目录里。Ext.Direct.dll 是 Ext.Direct 的对象库，使用 Ext.Direct 必须引用它。Newtonsoft.Json.dll 则是在.Net 中处理 JSON 数据用的，后面将会介绍到，这也是 Ext.Direct 对象中需要引用的。

现在新建一个名称为“js”目录，将 ext-base.js、ext-all.js 和 ext-lang-zh_CN.js 复制到该目录。将 Ext 包里的 resources 目录直接复制到网站目录里。

打开 default.aspx 的源代码，将 head 部分的“runat="server"”删除掉，将 body 内的源代码也删除掉，然后在 head 内加入以下代码：

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

以上代码主要是链接 Ext 库和样式的，test.ashx 随后会再创建。

在 body 内加入以下代码：

```
<input type="button" value="Hello" id="btnHello" />
<script>
   Ext.Direct.addProvider(Ext.app.REMOTING_API);
   Ext.onReady(function() {
      Ext.fly('btnHello').on('click', function() {
         MyApp.Test.SayHello(function(e, result) {
            Ext.Msg.alert('信息',result.result);
         })
      });
   });
</script>
```

该段代码在页面中加入了一个 id 为 btnHello 的按钮，在“<script>”下第一句注册 Ext.Direct 对象。在 onReady 中为 btnHello 按钮增加了一个单击事件。btnHello 按钮单击后将执行服务器端 Test 对象的 SayHello 方法，然后将返回值使用提示窗口显示出来。

在“解决方案资源管理器”中右键单击，之后选择“添加新项”，在弹出窗口中选择“一般处理程序”，将文件名修改为“Test.ashx”后，单击“添加”按钮。

在引用中增加以下语句：

```
using Ext.Direct;
```

在类定义前增加以下语句：

```
[DirectAction]
```

将“IHttpHandler”修改为“DirectHandler”后，将类内原有的内容全部清除，然后增加以下代码。

```
public override string ProviderName
{
    get
    {
        return "Ext.app.REMOTING_API";
    }
}

public override string Namespace
{
    get
    {
        return "MyApp";
    }
}

[DirectMethod]
public string SayHello()
{
    return "Hello";
}
```

这样一个简单的例子就做好了。在浏览器中打开 default.aspx，单击按钮，将看到如图 5-3 所示的结果。

图 5-3 单击按钮后的结果

继续我们的测试，在“解决方案资源管理器”根目录里单击右键，之后选择“添加 ASP.NET 文件夹”中的“App_Code”子菜单，然后在“App_Code”目录中单击右键，在弹出窗口中选择“类”，将文件名修改为“MyClass.cs”后单击“添加”。

在 MyClass 中增加对 Ext.Direct 的引用，并设置类属性为 DirectAction，完成后为类添加一个 SayHello 方法，如下面的代码所示。

```
[DirectMethod]
    public string SayHello()
{
    return "Hello";
}
```

接着在项目中再增加一个“Test2.ashx”文件，同“Test.ashx”一样，增加 Ext.Direct 的引用，修改继承对象和清除代码，然后加入以下代码：

```
public override string ProviderName
{
    get
    {
        return "Ext.app.USER_API";
    }
}
```

```
public override string Namespace
{
    get
    {
        return "MyApp";
    }
}

protected override void ConfigureProvider(DirectProvider provider)
{
    this.Configure(provider,new object[] {new MyClass()});
}
```

现在修改 default.aspx 文件，首先增加对“Test2.ashx”的引用，代码如下：

```
<script type="text/javascript" src="test2.ashx"></script>
```

在页面中增加第 2 个按钮，代码如下所示：

```
    <br />
<input type="button" value="Hello2" id="btnHello2" />
```

然后在 script 中增加一个对 Ext.app.USER_API 的注册，如下所示：

```
Ext.Direct.addProvider(Ext.app.USER_API);
```

最后为 btnHello2 按钮绑定单击事件并调用 MyClass 类的 SayHello 方法，如下所示：

```
Ext.fly('btnHello2').on('click', function() {
    MyApp.MyClass.SayHello(function(e, result) {
        Ext.Msg.alert('信息', result.result);
    })
});
```

在浏览器中打开“Default.aspx”并单击“Hello2”按钮会看到与图 5-3 一样的结果。

现在我们测试一下参数的传递，打开“Test.ashx”文件，并添加一个 SayHello2 的方法，方法将带一个名称为“username”的字符串参数，如下所示。

```
[DirectMethod]
public string SayHello2(string username)
{
    return "Hello," + username;
}
```

打开“Default.aspx”，增加第 3 个按钮，如下所示。

```
<br />
<input type="button" value="Hello3" id="btnHello3" />
```

然后为其绑定单击事件，如下所示：

```
Ext.fly('btnHello3').on('click', function() {
    MyApp.Test.SayHello2('小张', function(e, result) {
        Ext.Msg.alert('信息', result.result);
    })
});
```

在代码中传递了一个值为“小张”的参数。在页面中单击“Hello3”按钮，将看到如图 5-4 所示的结果。

图 5-4 单击“Hello3”按钮的显示结果

至此，我们的测试就结束了。如何在 Form 中使用

Ext.Direct，将在第 8 章 DirectLoad 与 DirectSubmit 一节讲解。

6.2 为 Ext.Direct 自定义远程方法

在上一节中我们已经了解到如何使用 directjngine 与 Ext.Direct 配合，使前台的 JavaScript 脚本可以访问后台暴露的远程方法，并使用回调函数处理远程方法的返回值。下一步我们将介绍如何使用 directjngine 编写远程方法，并将自定义的远程方法暴露给前台脚本。

假设我们已经编写了一个普通的 Java 类，如下所示：

```
package sample;

public class Hello {

   public String doHello(String msg) {
      return "Hello " + msg;
   }

   public Result getResult() {
      return new Result();
   }

   public static class Result {
      private boolean success=true;
      private String message="success";
   }
}
```

这个类中定义了两个方法：一个是拥有唯一参数的 doHello()方法，它的返回值是字符串类型；另一个是无参数的 doHello()方法，它的返回值是 Result 类型，Result 中拥有两个属性，布尔类型的 success 和字符串类型的 message。

如果希望把这个类转换成可以被 Ext.Direct 调用的远程方法，需要引入 com.softwarementors.extjs.djn.config.annotations 包中的 DirectMethod 注解，DirectMethod 用于修饰 Hello 类中的方法，只有使用 DirectMethod 修饰过的方法才会被暴露为 Ext.Direct 可以调用的远程方法，默认情况下暴露给 Ext.Direct 使用的远程方法名称就是 Java 中方法的原名，如果希望使用其他名称，也可以为 DirectMethod 指定特定的 name 属性。修改后的代码如下所示：

```
package sample;

import com.softwarementors.extjs.djn.config.annotations.DirectMethod;

public class Hello {

   @DirectMethod
   public String doHello(String msg) {
      return "Hello " + msg;
   }

   @DirectMethod
   public Result getResult() {
```

```
        return new Result();
    }

    public static class Result {
        private boolean success=true;
        private String message="success";
    }
}
```

下一步我们在 web.xml 中的 sample.classes 参数部分添加一段代码，将 Hello 类添加到 sample 模块中：

```
<init-param>
  <param-name>sample.classes</param-name>
  <param-value>
  sample.Profile,
  sample.TestAction,
  sample.Hello
  </param-value>
</init-param>
```

重启服务器之后，可以看到 directjngine 生成的 api.js 中包含了与 Hello 相关的远程方法的内容：

```
Ext.app.REMOTING_API={
  url: Ext.app.PROVIDER_BASE_URL,
  type: 'remoting',
  actions: {
    Hello: [
      {
        name: 'doHello'/*(String) => String */,
        len: 1,
        formHandler: false
      },
      {
        name: 'getResult'/*() => sample.Hello$Result */,
        len: 0,
        formHandler: false
      }
    ]
  }
}
```

上述配置中注明了 Hello 类中的 doHello()方法拥有一个参数，getResult()方法不需要传入任何参数。在 JavaScript 中使用如下代码调用这两个远程方法，并获得返回值。

```
Ext.onReady(function() {
    Ext.Direct.addProvider(Ext.app.REMOTING_API);

    Ext.get('hello').on('click', function() {
        Hello.doHello("Lingo", callback);
    });

    Ext.get('getResult').on('click', function() {
        Hello.getResult(callback);
```

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

调用 Hello.doHello()时，我们传入”Lingo”参数，然后获得”Hello Lingo”返回值，结果如图 6-4 所示。

图 6-4 调用 Hello.doHello()获得的返回值

通过 firebug 的控制台可以看到 Hello.doHello()函数调用时自动发送到后台的请求以及其中包含的请求数据，如图 6-5 所示。

图 6-5 调用 Hello.doHello()时发送的请求数据

从这里可以看到，在我们调用 Hello.doHello()方法时，Ext.Direct 会自动以 POST 方式向后台发送一条请求，请求中发送数据是我们很熟悉的 JSON：

- Action 表示期望调用的远程类；
- Method 表示期望调用的远程方法；
- Data 表示调用远程方法时传递的参数，它的类型是一个数组，数组中的参数个数与 Ext.app.REMOTE_API 中对应的 len 值相同。
- Type 表示使用 rpc 方式进行远程调用；
- Tid 表示此次调用的事务 id，它可以用来在批量调用中区分每个对应的请求和响应。

之后 MyExtDirectRouter 会处理上述请求中的 JSON 数据加以分拣，根据 action 和 method 的值决定调用哪一个远程类中的远程方法，在实际调用时会传入 data 中定义的参数，并将返回值组装为响应以返回给前台脚本处理。

图 6-6 所示为接收 Hello.doHello()发送的请求后所返回的响应内容：

图 6-6 调用 Hello.doHello()时返回的响应数据

可以看到响应内容使用的也是我们已经十分熟悉的 JSON 格式：

- Result 表示远程方法调用后返回的内容；
- Action 表示调用的远程类；
- Method 表示调用的远程方法；
- Type 表示使用 rpc 方式进行远程调用；
- Tid 表示此次调用的事务 id，它可以用来在批量调用中区分每个对应的请求和响应。

从上述响应内容中可以看到，action、method、type 和 tid 参数都是与请求中传递来的参数一一对应的，Ext.Direct 会通过这些参数来决定如何处理获得的响应数据，这样才能正确地调用我们之前所提供的回调函数。

单击 getResult 按钮之后显示的信息与单击 hello 按钮稍有不同，因为 Hello.getResult()返回的结果不再是一个简单的字符串，如图 6-7 所示：

下面我们再来观察一下调用 Hello.getResult()所发送的请求内容和对应的响应内容，如图 6-8 所示。

从图 6-8 中可以看到，与调用 Hello.doHello()时的情况不同，data 属性部分的值为 null，因为 Hello.getResult()远程方法不需要传递任何参数，其他参数的作用与上面的讨论基本相同，预期调用的远程类为 Hello，远程方法为 getResult，调用方式为 rpc，此时的事务 id 已经
变成了 6，Ext.Direct 内部会确保每次请求的 tid 都是唯一的，这样就可以通过它将对应的请求和响应对应在一起。

图 6-7 单击 getResult 按钮显示的信息

图 6-8 调用 Hello.getResult()发送的请求内容

图 6-9 所示是调用 Hello.getResult()之后返回的响应内容：

Hello 中的 getResult()方法返回的不再是一个简单的字符串类型的数值，而是我们自定义的 Result 对象，Result 类中包含两个属性：布尔类型的 success 和字符串类型的 message：

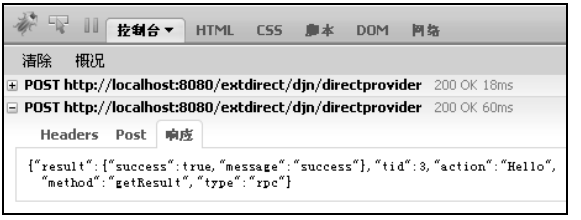

图 6-9 调用 Hello.getResult()返回的响应内容

```
    public static class Result {
       private boolean success=true;
       private String message="success";
    }
```

在 getResult()中我们直接返回了一个新建的 Result()对象，之后 gson-1.3.jar 会将它自动转换为 JSON 格式，并作为响应中的 result 属性的值，最终将完整的响应数据传递到前台进行处理。此时回调函数获得的结果将是一个 json object 的变量，为简单起见，我们直接使用 Ext.encode()将响应内容直接显示在 MessageBox 中。

```
    function callback(result) {
       Ext.Msg.alert('info', Ext.encode(result));
    }
```

完整的 hello.html 内容如下所示：

```
    <html>
      <head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
        <title>Ext JS 3.0 Direct</title>
        <link   rel="stylesheet"   type="text/css"   href="ext-3.0.0/resources/css/ext-
all.css" />
        <script type="text/javascript" src="ext-3.0.0/ext-base.js"></script>
        <script type="text/javascript" src="ext-3.0.0/ext-all.js"></script>
        <script type="text/javascript" src="sample/api.js"></script>
        <script type="text/javascript">
    Ext.BLANK_IMAGE_URL='ext-3.0.0/resources/images/default/s.gif';

    function callback(result) {
       Ext.Msg.alert('info', Ext.encode(result));
    }

    Ext.onReady(function() {
       Ext.Direct.addProvider(Ext.app.REMOTING_API);

       Ext.get('hello').on('click', function() {
          Hello.doHello("Lingo", callback);
       });

       Ext.get('getResult').on('click', function() {
          Hello.getResult(callback);
       });
    });

       </script>
      </head>
      <body>
        <button id="hello">hello</button><br>
        <button id="getResult">getResult</button>
      </body>
    </html>
```

7.6 Store 的数据操作

7.6.1 添加数据

要给 Store 添加数据，可以使用 insert 方法、add 方法、addSorted 方法或 loadData 方法。insert 方法将在指定位置开始插入记录，一次可以插入多条记录；add 方法则直接增加一条或多条记录；addSorted 方法则在其排序后的位置插入记录，该方法一次只能插入一条记录。如果要使用 loadData 方法添加记录，则需要指定第 2 个参数为 true，不然会清空原有数据再追加数据。

它们的使用方法请看下面代码：

```
//添加单个记录
var data={id:1001,name: '张三'};
var p=new store.recordType(data,data.id);

store.insert(2,p);              //在第一条记录后插入

store.add(p);

store.addSorted(p);

store.loadData(p,true);

//添加多个记录
var data1={id:1001,name: '张三'};
var p1=new store.recordType(data1,data1.id);
var data2={id:1002,name: '李四'};
var p2=new store.recordType(data2,data2.id);
store.insert(0,[p1,p2]);
store.add([p1,p2]);
store.loadData([p1,p2],true);
```

7.6.2 删除数据

在 Store 中删除数据有 remove、removeAll 和 removeAt 这 3 种方法。它们的使用方法请看下面代码：

```
store.remove(rec);              //rec 为一个记录
store.removeAll();              //删除所以记录
store.removeAt(10);
```

从代码中可以看到：remove 方法需要知道具体的记录才允许删除；removeAll 删除全部记录，并触发 clear 事件；removeAt 方法则删除指定位置的记录。

7.6.3 搜索、定位和统计

Store 提供了以下搜索和定位记录的方法：

- each：枚举所有记录，当枚举函数返回 false 时，终止枚举操作。其使用方法请看下面代码：

```
store.each(function(rec){
  //处理过程
});
```

- filter：根据指定属性过滤记录。其使用方法请看下面代码：

```
//过滤掉不是 name 不包含"张"的记录
store.filter('name','张',true,false);
```

代码中的过滤规则可使用字符串，也可以使用正则表达式。第 3 个参数如果为 false，则表示只匹配开始位置，为 true 则表示匹配任何位置；第 4 个参数如果为 false，则表示不区分大小写，为 true 则表示要匹配大小写。

- filterBy：通过一个函数过滤记录。该函数会枚举每一个记录，然后根据函数返回值判断记录是否被过滤。如果函数返回 true，则包含该记录，如果返回 false，则过滤掉该记录。其使用方法请看下面代码：

```
//返回 id 为单数的记录
store.filterBy(function(rec,id){
  if(id %2==0){
    return true;
    }else{
      retrun fase;
    }
});
```

- isFiltered：判断 Store 当前是否处于过滤状态。如果是，返回 true；否则，返回 false。其使用方法请看下面代码：

```
if(store.isFiltered()){
  store.clearFilter();
}
```

- clearFilter：将 Store 恢复到没有进行过滤的状态，其使用方法可参考 isFiltered 方法中的代码。
- find：根据指定属性查找匹配的记录，并返回匹配的第一个记录的索引值。如果没有找到匹配的记录，则返回-1。其使用方法请看下面代码：

```
var index=store.filter('name','张',2,true,false);
```

与 filter 方法一样，搜索值可以是字符串，也可以是正则表达式。第 3 个参数为搜索开始位置，默认值为 0。第 4 个参数如果为 false，则表示只匹配开始位置，为 true 则表示匹配任何位置。第 5 个参数如果为 false，表示不区分大小写，如果为 true 则表示要区分大小写。

- findBy：通过一个函数查找匹配的记录并返回匹配的第一个记录的索引值。该函数会从指定的开始位置枚举每一个记录，然后根据返回值判断记录是否匹配。如果函数返回 true，则表示已找不到匹配记录，枚举操作结束并返回匹配记录的索引值。如果没有找到匹配的记录，方法返回值为-1。其使用方法请看下面代码：

```
store.findBy(function(rec,id){
  if(id %2==0){
    return true;
```

```
    }else{
       retrun fase;
    }
},2);
```

在代码中，第 2 个参数“2”表示查找的开始位置。

- findExact：与 find 方法作用一样，只是没有匹配位置和区分大小写参数。其使用方法可参数 find 方法。
- getAt：根据索引值获取一个记录，其使用方法请看下面代码。

```
//获得第 2 条记录
var rec=store.getAt(2);
```

- getById：根据记录 id 返回记录，其使用方法请看下面代码。

```
//获得 id 为 2 的记录
var rec=store.getById(2);
```

- getCount：返回 Store 的记录总数。如果没有采用分页方式，则其返回结果与 getTotalCount 方法一样。如果采用了分页方式，则 getCount 返回的是 Store 的记录总数，getTotalCount 返回的才是数据库的记录总数，不过前提是要在 Reader 定义中包含有记录总数的属性。其使用方法请看下面代码：

```
var count=store.getCount();
```

- getTotalCount：返回数据库的记录总数，不过前提是 Reader 必须包含记录总数的属性。其使用方法请参考 getCount 方法。
- getModifiedRecords：获取执行 commit 后被修改的记录。要注意的是，返回的记录不包括被删除的记录。其使用方法请看下面代码：

```
var recs=store.getMpdifiedRecords ();
```

- getRange：返回指定范围内的记录。其使用方法请看下面代码：

```
//返回所以记录
var recs=store.getRange();

//返回第 2 条到第 10 条之间的记录。
var recs=store.getRange(2,10);
```

- indexOf：返回记录的索引值。如果记录在 Store 中不存在，则返回-1。其使用方法请看下面代码：

```
var index=store.indexOf(rec);
```

- indexOfId：根据记录 id 返回记录的索引值。如果记录 id 在 Store 中不存在，则返回-1。其使用方法请看下面代码。

```
var index=store.indexOfId(2);                    //返回 id 为 2 的记录索引值
```

- query：获取符合指定条件的记录。其返回值为 Ext.MixedCollectionxt 对象。其使用方法可参考 filter 方法。
- queryBy：通过一个函数获取指定条件的记录。该函数会枚举每一个记录，然后根据函数返回值判断记录是否被选取。如果函数返回 true，则包含该记录，如果返回 false，则不选取该记录。其使用方法可参考 filterBy 方法。

- sum：合计指定范围内指定字段的值。其使用方法请看下面代码。

```
//计算 total 字段从第 2 个记录到第 10 个记录的和
var total=store.sum('total',2,10);
```

7.6.4 更新数据

当使用 Grid 时我们会发现，某个数据被修改后，会在其单元格左上角显示一个红色标记，表示该数据被修改过，而包含该数据的记录的 dirty 属性将被标记为 true，说明该记录处于已被更改状态。如果这时需要取消所有记录的更改，可使用 Store 的 rejectChanges 方法，其使用方法请看下面的代码。

```
store.rejectChanges();
```

如果你想确认更改，可使用 commitChanges 方法或 save 方法。

commitChanges 方法的使用方法请看下面代码。

```
store.commitChanges();
```

commitChanges 方法会触发 Store 的 update 事件。

save 方法的使用方法请看下面代码。

```
store.save();
```

save 方法与 commitChanges 方法不同的地方是会将更新发送到服务器端。如果配置了 Ext.data.Api.actions 对象，则根据该定义提交数据。如果没有定义，则使用 Store 定义的 url 属性提交数据。

commitChanges 方法和 rejectChanges 方法经常会用于 Grid 中的 Checkbox 修改。譬如：用户单击 Checkbox 后，会触发 Store 的 update 事件，在 update 事件里，我们使用 Ajax 将更新提交到服务器端以更改记录。然后在 Ajax 的回调函数中，如果服务器端返回的信息表示更新成功，就使用 commitChanges 方法更新 Store；如果不成功，则使用 rejectChanges 方法取消更改，使客户端的数据与服务器端保持一致，也避免了在 Grid 的单元格中左上角显示数据更改信息，具体使用方法请看下面代码。

```
var store=new Ext.data.Store({
  url:'test.ashx',
  reader:new Ext.data.JsonReader({
     totalProperty: "results",
     root:"rows",
     id:"id"
     },[
       {name: 'id',type:'int'},
       {name:'title'},
       {name:'ontop',type:'bool'},
       {name:'posttime',type: 'date',dateFormat:'Y-m-d H:i:s'}
     ]),
remoteSort: true,
listeners:{
     update:function(store,rec,op){
     //判断是否有记录被编辑
```

```
            if(op==Ext.data.Record.EDIT){
              Ext.Ajax.request({
                params:{field: 'ontop',id:rec.data.id},
                url: 'action.ashx?act=check',
                scripts:true,
                success: function(response ,options){
                var msg=response.responseText;
                var obj=Ext.decode(msg);
                if(obj){
                  if(obj.success){
                    //服务器端更新成功更新数据
                    app.store.commitChanges();
                    return;
                }else{
                    //服务器端更新取消修改
                  app.store.rejectChanges();
                }
                if(msg!="")
                  Ext.Msg.alert("信息",msg);
                },
                failure: function(response ,options){
                    //传输数据失败，取消修改
                    app.store.rejectChanges();
                    Ext.Msg.alert("错误","改变文章状态失败！<br>出错信息："+response.
responseText);
                    }
                });
               }
            }
          }
        }
    }) //store
```

7.6.5 排序

Store 的排序分远程排序和本地排序两种方式。如果是远程排序，需要在定义 Store 时设置 http://localhost/control/ext3/docs/index.html - expandremoteSort 属性，其使用方法请看下面的代码。

```
    var store=new Ext.data.Store({
       remoteSort:true,
       reader: new Ext.data.JsonReader(
          {
             idProperty: 'key',
             root: 'daRoot',
             totalProperty: 'total'
          },
          Dude  // recordType
       )
    });
```

实行远程排序时，每当顺序改变时，例如单击 Grid 的列标题，会向服务器提交 sort 和 dir 两个

排序参数。其中：sort 参数表示排序的字段名称，要注意的是，这是 Store 中定义的字段名称，不是数据库中的字段名称；而 dir 参数表示排序顺序，字符串“ASC”表示顺序排序，字符串“DESC”表示降序排序，要注意，字符串中的字母必须全部为大写字母。

在本地对记录进行排序，可使用 sort 方法，其使用方法请看下面代码。

```
store.sort('name', 'ASC');
```

代码表示根据字段“name”对记录进行排序。如果要执行降序排序，则将“ASC”修改为“DESC”。要注意，“ASC”与“DESC”所有字母必须为大写字母。

如果想获取当前排序状态，可使用 getSortState 方法，其使用方法请看下面代码。

```
var sortstate=store.getSortState();
```

该方法将返回一个 JSON 数据，其结构请看下面代码。

```
{
  field: '当前排序的字段名称',
  direction: '当前的排序顺序'                    //值为 ASC 或 DESC
}
```

如果想设置每次调用 load 方法后默认的排序方式，可使用 setDefaultSort 方法，其使用方法请看下面代码。

```
store.setDefaultSort('name','DESC');
```

该段代码表示在每次调用 load 方法之后，默认的排序字段是 name，排序顺序为降序。如果要升序排序，请将“DESC”修改为“ASC”，所有字母都必须为大写字母。

在本地排序中，Ext 3.0 还是没有将字符串排序使用 localeCompare 方法进行排序，所以在对中文排序时还是存在问题，因而需要重载一下 Store 的 sortData 方法，具体请看下面的代码。

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

代码基本是按 Ext 的源代码修改过来的，修改的地方是先判断 v1 的值是否是字符串，如果是，则使用 localeCompare 方法比较数据，否则按照原来的方式比较。

笔者比较喜欢将该段代码放在本地化文件（ext-lang-zh_CN.js）中，这样就不需要根据页面情

况加入这段代码了。