

ExtJS 资源甘特图控件

用户手册

北京龙博中科软件有限公司

www.longboo.com

序言.....	3
简介.....	4
ExtJS 资源甘特图功能特性	4
资源 甘特图运行效果.....	5
如何安装.....	6
1. 下载运行.....	6
打开 ASP.NET 项目	6
运行示例.....	7
开发入门.....	7
资源 甘特图控件的基本原理.....	7
DataStore.....	10
GanttPanel.....	错误！未定义书签。
数据结构.....	12
任务的数据结构.....	12
依赖关系的数据结构.....	14
加载数据.....	15
保存数据.....	16
实现增删改查接口.....	16
ASP.NET 后台操作	17
自定义任务树.....	18
自定义列.....	18
自定义行.....	19
自定义单元格.....	20
自定义编辑器.....	21
编辑器只读事件.....	22
自定义条形图.....	23
自定义样式.....	23
条形图前后的标签.....	错误！未定义书签。
自定义提示框.....	24
时间刻度和自定义.....	25
任务时间基线.....	错误！未定义书签。
高级应用.....	28
大数据量和性能优化.....	28
皮肤和主题.....	错误！未定义书签。
附录一 Extjs 中的 DataStore.....	29
Record.....	29
Store	30
DataReader.....	33
1) ArrayReader	33
2) JsonReader	34
3) XmlReader	35
DataProxy 和自定义 Store	36

序言

本用户手册中文版是由ExtJS资源甘特图控件中国代理商龙博中科软件有限公司组织编写的，希望能帮助正在学习或准备学习ExtJS资源甘特图控件的朋友们快速走进资源甘特图控件的精彩世界。

手册包括ExtJS资源甘特图的新手入门、组件体系结构及使用、ExtJS资源甘特图中各功能的使用方法及示例应用等，是一个非常适合新手的ExtJS资源甘特图入门手册。本手册主要是针对ExtJS资源甘特图2.0 进行介绍，全部代码、截图等都是基于ExtJS资源甘特图2.0。

最后，希望读者朋友把阅读本手册中的发现的错误、不足及建议等反馈给我们，我们会在下一版本中及时改正。下面让我们一起进入精彩的ExtJS资源甘特图世界吧。

简介

资源甘特图又叫负荷图，其纵轴不再列出活动，而是列出整个部门或特定的资源。

Ext Scheduler 资源甘特图是基于 Extjs 核心库的开发的，基于 WEB 浏览器的甘特图解决方案。可应用于生产能力进行计划和控制系统等的开发。和 Ext Gantt 甘特图一样，Ext Scheduler 资源甘特图可与任意后端代码（.net，jsp）集成为 jsp 资源甘特图，asp.net 资源甘特图等。

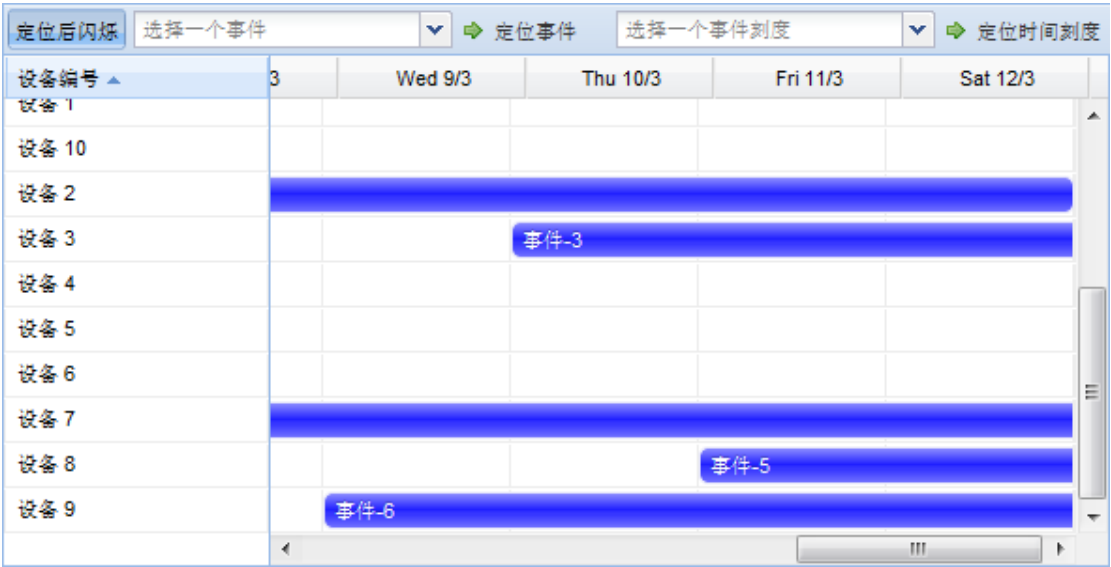
ExtJS 资源甘特图功能特性

ExtJS资源甘特图是使用Javascript开发的、基于WEB浏览器的甘特图解决方案，具备如下特点：

- 可拖拽调节条形图
- 丰富的时间线刻度：支持年/月、周/日、周/时、日/时等，用户也可以自定义时间线模式。
- 自定义列。
- 卓越的性能和大数据量支持
- 懒加载显示：通过分级加载模式，能支持超大数据量显示。
- 支持右键菜单
- 支持跟踪甘特图
- 无需安装浏览器插件

- 跨浏览器支持
- 跨服务端平台支持
- 语言本地化

资源甘特图运行效果



如何安装

1.下载运行

如果你是试用版的用户，请访问我们的网站

<http://www.longboo.com/>，在我们的下载频道中点击下载。我们目前提供以下的资料，更多的资料请到我们的网站及时更新。

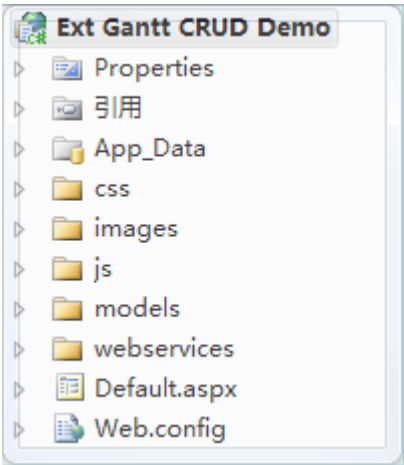
Ext Scheduler Web 资源甘特图控件下载

Web 资源甘特图 + ASP.NET 工程实例下载

ExtJS 资源甘特图控件是一个纯表现层的组件产品。下面以 ASP.NET 工程项目为例，是让您更方便、快速的掌握如何将 ExtJS 资源甘特图控件与您的系统集成开发。使用 JAVA 开发和 ASP.NET 类似，这里不再赘述。

打开 ASP.NET 项目

下载《Web 资源甘特图+ ASP.NET 工程实例下载》后，将它解压到某个文件夹下面，请用 VS 打开相应的工程文件。



运行示例

点击运行后，最终的运行结果如下图所示。



商业用户请注意，Extjs 资源甘特图授权并不包括 Extjs 本身的授权，你需要按照你的需要，购买相应的 Extjs 的授权。

开发入门

甘特图控件的基本原理

ExtJS 资源甘特图控件继承至 Extjs 的 GridPanel，如果你熟悉 Extjs，那么使用 Extjs 资源甘特图就非常的轻松；如果没有，你需

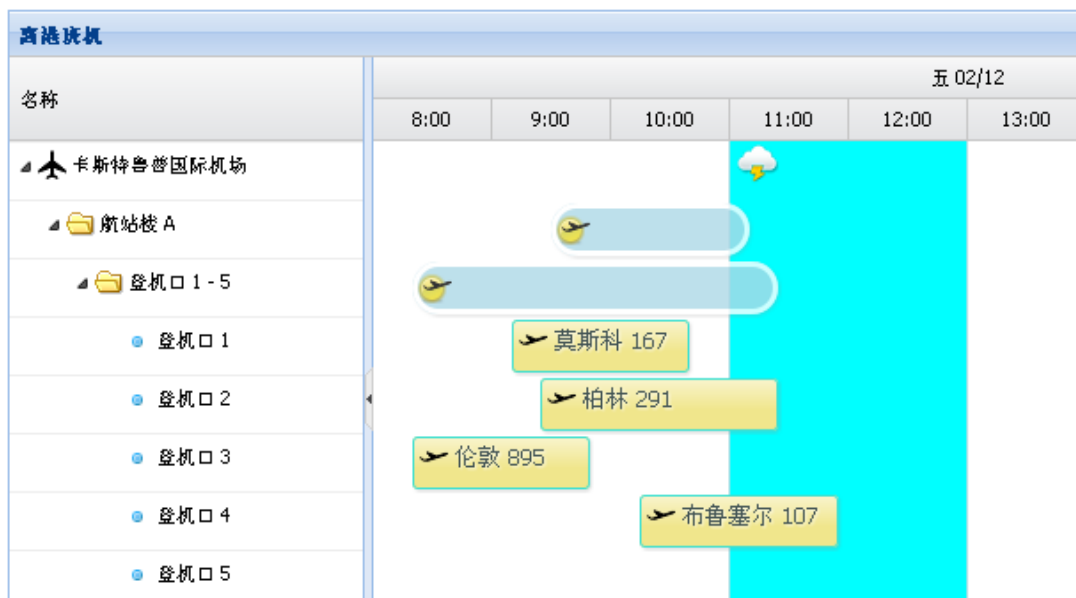
要花一点时间来先学习一下 Extjs。ExtJS 资源甘特图控件按照 MVC（模型、视图、和控制器）的设计框架设计。

1. 模型层：甘特图的数据模型包括两个方面，任务模型和依赖模型（任务之间的关联关系）。相对应的类有 `Ext.ux.maximb.tb.AdjacencyListStore` 和 `Ext.data.Store`，而前者继承于后者。
2. 界面显示组件：我们主要使用这个 `GanttPanel` 类，只需要定义一些属性就可以了。
3. 控制器：这个是由 Extjs 资源甘特图内部控制的，我们只需要在适当的时候，相应一些事件就可以定制我们自己的功能。

`GanttPanel` 是可视的组件，它用于界面显示、编辑操作、用户交互等功能。`AdjacencyListStore` 和 `Ext.data.Store` 是不可见的数据组件，负责管理数据模型和数据操作。在 `dataStore` 上的所有数据操作，比如增、删、改、过滤、排序等，都会被 ExtJS 甘特图控件监听并更新界面。

`GanttPanel` 是由两个重要的界面组件组成的：

1. 左边是数据表格或树形表格组件，继承至 Extjs 的 `Datagrid`。
所以 `Datagrid` 的操作也可以运用于 `GanttPanel`，就连 API 也是一样的。



2. 右边是条形图组件，这个是 Extjs 资源甘特图控件特有的。



如上图所示，左侧的是数据表格或树形表格，右侧是条形图，他们组合起来就是一个标准的资源甘特图。

我们先帖出一个最简单的，显示一个甘特图例子的完整代码。读者朋友可以对代码有个基本的了解，然后在逐步分析各个部分的代码。

```
1. //定义资源的数据模型
2.     var resourceStore = new Ext.data.JsonStore({
3.         proxy : {
4.             type : 'ajax',
```

```

5.          //data.js 中包含资源数据
6.          url : 'data.js',
7.          reader : {
8.              type : 'json',
9.              root : 'staff'
10.         }
11.     },
12.     model : 'Sch.model.Resource'
13. });
14.     //加载事件数据
15.     var eventStore = new Ext.data.JsonStore({
16.         model : 'Sch.model.Event'
17.     });
18.
19.     resourceStore.on('load', function() {
20.         eventStore.loadData(resourceStore.proxy.reader.jsonData.tasks);
21.     });
22.
23.     //定义资源甘特图
24.     var ds = Ext.create("Sch.SchedulerPanel", {
25.         width      : 1030,
26.         height     : 400,
27.         resourceStore : resourceStore,
28.         eventStore   : eventStore,
29.         columns     : [
30.             {
31.                 header : '员工', sortable:true, width:130, dataIndex : 'Name'
32.             },
33.             //你可以增加其他字段
34.             {
35.                 header : '职务'
36.             }
37.         ]
38.     });
39.

```

DataStore

我们写程序的时候，大多数情况下，都只要去更新数据集中的数据。当我们修改了数据集中的数据甘特图控件通过监听数据改变事

件，更新自己的界面视图， 负责更新表格数据操作，例如增加行、删除行、修改单元格、移动、排序、过滤， 修改右边图表等操作。

考虑到部分读者并不十分熟悉 ExtJS，我们在附录中附上了 Extjs DataStore 的入门指南，供大家参考。详见附录一。

SchedulerPanel

SchedulerPanel 是我们要在网页上显示的可视化的类。 在定义 SchedulerPanel 的时候需要三个最基本的要素，资源的 DataStore 和资源甘特图要显示的列，如开始时间，结束时间等等。

除了以上的基本功能，我们还可以在 SchedulerPanel 中定义不同的属性，完成一些常见的扩展。例如自定义列、自定义单元格内容、自定义单元格编辑器、根据权限控制编辑器，自定义条形图外观、监听处理拖拽事件。

一个更加复杂的 SchedulerPanel 例子为：

```
1.    var s = Ext.create("Sch.SchedulerPanel", {
2.        region      : 'center', //所占位置
3.        rowHeight   : 30, //设置行高
4.        height      : 350, //资源甘特图高
5.        width       : 900, //资源甘特图宽
6.        loadMask: true, //是否显示 loading 框
7.        columns     : [
8.            {
9.                header : '员工',
10.               sortable:true,
11.               width:130,
12.               dataIndex : 'Name'
```

```

13.         }
14.     ],
15.
16.     startDate : new Date(2011, 0, 1, 6), //时间刻度开始时间
17.     endDate   : new Date(2011, 0, 1, 20), //时间刻度结束时间
18.     viewPreset: 'hourAndDay', //时间刻度显示类型
19.     resourceStore : resourceStore, //资源的 data store
20.     eventStore : eventStore, //事件的 data store
21.     //设置操作按钮
22.     tbar : [
23.         {
24.             iconCls : 'icon-previous',
25.             scale : 'medium',
26.             handler : function() {
27.                 s.shiftPrevious();//视图前移
28.             }
29.         },
30.         {
31.             iconCls : 'icon-next',
32.             scale : 'medium',
33.             handler : function() {
34.                 s.shiftNext();//视图后移
35.             }
36.         }
37.     ]
38.     });

```

数据结构

资源甘特图的数据结构分为两部分，资源的数据和事件的数据。这种设计大幅度的降低了数据之间的耦合关系，也更加接近于我们平时的数据库设计，降低了和数据库接口之间的难度。

资源的数据结构

先看一个资源的数据结构简单的 JSON 的例子。

```

1.     staff : [
2.         {

```

```
3.         Id : 'a',
4.         Name : '罗伯特',
5.         Type : 'Sales'
6.     },
7.     {
8.         Id : 'b',
9.         Name : '迈克',
10.        Type : 'Sales'
11.    },
12.    {
13.        Id : 'c',
14.        Name : '凯特',
15.        Type : 'Product manager'
16.    },
17.    {
18.        Id : 'd',
19.        Name : '莉莎',
20.        Type : 'Developer'
21.    },
22.    {
23.        Id : 'e',
24.        Name : '大卫',
25.        Type : 'Developer'
26.    },
27.    {
28.        Id : 'f',
29.        Name : '艾诺得',
30.        Type : 'Developer'
31.    },
32.    {
33.        Id : 'g',
34.        Name : '小李',
35.        Type : 'Marketing'
36.    },
37.    {
38.        Id : 'h',
39.        Name : '琼斯',
40.        Type : 'Marketing'
41.    }
42. ],
43.
44.
```

一个资源可以有很多属性，但是必须包括以下的数据，资源甘特图才能够显示出来。

- 'Id'，资源 Id
- 'Name'，资源名称

事件的数据结构

事件的数据结构非常简单，只包括三个属性

- 'Id'，事件 Id
- 'StartDate'，开始时间
- 'EndDate'，结束时间

以下是一个事件的数据结构简单的 JSON 的例子。

```
1. tasks : [  
2.     {  
3.         ResourceId : 'a',  
4.         Title : '任务',  
5.         StartDate : '2010-05-22 10:00',  
6.         EndDate : '2010-05-22 12:00',  
7.         Location : 'Some office'  
8.     },  
9.     {  
10.        ResourceId : 'b',  
11.        Title : '其它任务',  
12.        StartDate : '2010-05-22 13:00',  
13.        EndDate : '2010-05-22 16:00',  
14.        Location : 'Home office'  
15.    },  
16.    {  
17.        ResourceId : 'c',  
18.        Title : '基本任务',  
19.        StartDate : '2010-05-22 9:00',  
20.        EndDate : '2010-05-22 13:00',
```

```
21.         Location : 'Customer office'
22.     }
23. ]
```

以上的例子都是基于 JSON 的。也就是说，甘特图可以读取静态的 JSON，或者是 ASP.NET、Java 等后端程序生成的 JSON。除了 JSON，Extjs 甘特图也支持更多的数据结构和数据格式。事实上，用户可以任意组织存储自己的项目数据，无论是服务端是 .NET 还是 JAVA，无论数据库是 ORACLE 还是 MYSQL，无论数据传输方式是 JSON 还是 XML，Web Service，Rest 等等。Extjs 自身提供的强大 Datareader，可以满足各种不同的需求。

加载数据

我们再来看看我们的 DataStore 是如何写的。除了定义数据结构以外，我们还定义了一个叫做 Proxy 的对象。

```
1. var resourceStore = new Ext.data.JsonStore({
2.     proxy : {
3.         type : 'ajax',
4.         url : 'data.js',
5.         method: GET
6.     }
7. }),
8. ...
9. });
```

在以上的例子中我们定义了一个 **HttpProxy**，**DataProxy** 字面解释就是数据代理，也可以理解为数据源，也即从哪儿或如何得到需要交给 **DataReader** 解析的数据。数据代理（源）基类由

`Ext.data.DataProxy` 定义，在 `DataProxy` 的基础，`ExtJS` 提供了 `Ext.data.MemoryProxy`、`Ext.data.HttpProxy`、`Ext.data.ScriptTagProxy` 等三个分别用于从客户端内存数据、`Ajax` 读取服务器端的数据及从跨域服务器中读取数据等三种实现。

在 `proxy` 的 `url`，我们配置了 `data.js`，也可以是一个任意的 `url` 地址，例如 `http://localhost/text.jsp` 或者 <http://localhost/text.aspx> 等等。而 `method` 则可以配置为 `get` 或者 `post` 等等。

`EventStore` 的配置和这里的类似，我们就不再赘述。

保存数据

实现增删改查接口

上一节我们讲述了如何显示甘特图，当用户增加删除了一些任务，或者改变了任务之间的依赖关系，我们如何将改动后的数据提交到服务器，在服务器中保存到数据库或文件系统中呢？

答案非常简单，只要实现 `Data Store` 的四个接口（增删改查）就可以了。我们先来看一段代码。

```
1. var resourceStore = new Ext.data.JsonStore({  
2.     proxy : {  
3.         api: {
```



```

4.         read : 'webservices/Tasks.aspx/Get',
5.         create: 'webservices/Tasks.aspx/Create',
6.         destroy: 'webservices/Tasks.aspx/Delete',
7.         update: 'webservices/Tasks.aspx/Update'
8.     }
9.
10.    },
11.    model : 'Sch.model.Resource'
12.    });

```

在以上的代码中，分别定义了四个不同的 url，分别对应着增删改查四个操作的 URL 地址。例如，用户修改了一个资源，甘特图控件就会将修改后的数据提交到 webservices/Tasks.aspx/Update。 其他的操作同理。下面我们以修改为例，看看在 ASP.NET 的服务器端是如何获取修改后的数据的。

ASP.NET 后台操作

```

1.  [WebMethod]
2.      [ScriptMethod]
3.      Public Function Update(ByVal jsonData As Employee) As Object
4.          Dim _db As New DataClasses1DataContext()
5.
6.          Dim t = _db.Employees.SingleOrDefault(Function(b) b.Id = jsonData.Id)
7.
8.          If Not t Is DBNull.Value Then
9.              t.Active = jsonData.Active
10.             t.Name = jsonData.Name
11.             t.Salary = jsonData.Salary
12.          End If
13.
14.          _db.SubmitChanges(ConflictMode.ContinueOnConflict)
15.          Return New With {.success = True}
16.      End Function

```

在以上的例子中，我们建立一个 Web Service 来实现增删改查，这个 Web Service 里面有一个 Update 的方法。请注意在该方法的前面，有[ScriptMethod]这样一个申明。 他说明这个 Web Service 提交的数据是一个 JSON 数据。

函数 Update 的参数是一个 jsonData，根据不同的业务逻辑，更新数据库或文件系统。

自定义资源甘特图

SchedulerPanel 中默认的设置很难满足客户多变需求，我们就针对一些常用的扩展做一个说明。

一般最常用的需求是，自定义多列，自定义行，自定义单元格等等。下面我们就分别来说明。

自定义列

SchedulerPanel 的列配置对象，描述了表头、单元格、编辑器等重要配置。一般来说， 我们提供一个列配置对象，显示了“序号”、“资源名称”等比较典型的列。 但是有时我们会显示更多的资源属性，例如，资源的部门，职务，地点等等。

自定义列非常简单，只要在配置 SchedulerPanel 的列配置对象中多添加一个单元既可。见以下例子

```
40. var ds = Ext.create("Sch.SchedulerPanel", {  
41.     width      : 1030,
```

```

42.         height      : 400,
43.         resourceStore : resourceStore,
44.         eventStore    : eventStore,
45.         columns       : [
46.             {
47.                 header : '员工', sortable:true, width:130, dataIndex : 'Name'
48.             },
49.             //你可以增加其他字段
50.             {
51.                 header : '职务', sortable :true,width : 100,dataIndex : 'Types'
52.             },
53.             {
54.                 header : '更多列',sortable :true,width :100,
55.                 dataIndex:'Others'
56.             },
57.             {
58.                 header : '新增列', sortable : true, width : 100,
59.                 dataIndex : 'Others2'
60.             }
61.         ]
62.     });
63.

```

运行后的软件截图如下

时 天 周 月 年					
名字	职务 ▼	更多列	新增列	6:00	7:00
晓东	经理助理	更多列	新增列		
麦克	经理	更多列	新增列		
杰克	秘书	更多列	新增列		
彼得	员工	更多列	新增列		
琳达	员工	更多列	新增列		
凯瑞	会计	更多列	新增列		

自定义行

自定义行的需求有很多，例如，在行中显示不同的文字颜色，定义行的背景高亮显示等等。 以下我们以在行中显示不同的条形图颜

色为例，说明如何定义不同的行。

关于行的配置，我们一般在 `eventRenderer` 中配置。

具体代码如下。

```
1.   viewConfig: {  
2.     getRowClass: function(r, idx, rowParams, ds) {  
3.       if (r.get('Others')== 'test') {  
4.         return 'processed-row';  
5.       }  
6.     }  
7.   },
```

以上的代码将字段 ‘others’ 等于 ‘test’ 的 `tr` 中添加了一个 `processed-row` 的类。下一步我们需要做的是在 `css` 中定义 `processed-row` 的样式。

```
1. <style type="text/css">  
2. .processed-row td{  
3.   color: red;  
4.   font-size:11px;  
5. }  
6. </style>
```

自定义单元格

一个单元格的显示内容，取决于两点：

- `dataIndex`：如果配置了 `dataIndex`，那么单元格默认显示行对象的 `dataIndex` 属性。
- `renderer`：如果有单元格渲染器，那么单元格会忽略掉 `dataIndex` 的显示功能，而只显示从 `renderer` 函数返回的 `HTML` 字符串。

其中，单元格渲染器函数 `renderer` 是最强大的单元格内容显示工具。通过 `renderer` 函数，您可以得知是哪一行、哪一列、单元格值，并且可以结合您系统的权限，返回不同的 HTML 内容。返回的 HTML 内容是任意的，可以是图片、超链接、按钮等。下面列举一个简单的例子。

```
1. {
2.     header : '工期',
3.     sortable:true,
4.     width:50,
5.     dataIndex : 'Duration',
6.     renderer : function (value) {
7.         if(value=='test'){
8.             return value+'天'
9.         }else{
10.             return value;
11.         }
12.     }
13.     locked : true,
14. },
```

自定义编辑器

如果需要快速编辑数据，就需要使用单元格编辑器。使用单元格编辑器的代码如下：


```
1. {
2.     xtype: 'datecolumn',
3.     header : '完成时间',
4.     sortable:true,
5.     width:90,
6.     dataIndex : 'StartDate',
7.     locked : true,
8.     field:new Ext.form.DateField({
```

```

9.             allowBlank : false,
10.             format: 'Y/m/d'
11.         }),
12.         format: 'Y 年 m 月 d 日'
13.     }

```

单元格编辑器编辑时效果图如下：

名字 ▲	职务	完成时间	新增列																																																	
凯瑞	会计	2011年11月01日	新增列																																																	
凯瑞	会计	2011年11月02日	新增列																																																	
彼得	员工	2011/11/03 	新增列																																																	
彼得	员工	<div>十一月 2011</div> <table><tr><th>日</th><th>一</th><th>二</th><th>三</th><th>四</th><th>五</th><th>六</th></tr><tr><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr><tr><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td></tr><tr><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td></tr><tr><td>27</td><td>28</td><td>29</td><td>30</td><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr></table> <div>今天</div>		日	一	二	三	四	五	六	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10
日	一	二	三	四	五	六																																														
30	31	1	2	3	4	5																																														
6	7	8	9	10	11	12																																														
13	14	15	16	17	18	19																																														
20	21	22	23	24	25	26																																														
27	28	29	30	1	2	3																																														
4	5	6	7	8	9	10																																														
晓东	经理助理																																																			
晓东	经理助理																																																			
杰克	秘书																																																			
杰克	秘书																																																			
辟达	员工																																																			
辟达	员工																																																			
麦克	经理																																																			
麦克	经理																																																			
麦克	经理																																																			

编辑器只读事件

我们在平时的应用中，经常会控制甘特图的编辑权限。 例如让甘特图为只读或部分只读。

控制整个甘特图只读非常简单，只需要下面一行语句就能控制。

```

1. gantt.setReadOnly(true);

```

在实际的应用中，往往要控制甘特图的某些行只读或者是某一个单元格只读，这时候我们需要编写 **beforeedit** 回调函数了。请看下面的例子。

```
1. listeners : {  
2.     beforeedit : function(e) {  
3.         if(e.record.get('other')== '资源 1'){  
4.             return false;  
5.         }  
6.     }  
7. }
```

这是一个 **beforeedit** 回调函数的简单例子。 **beforeedit** 回调函数会在用户单击单元格之后，用户输入内容之前调用，如果返回 **false**，则禁止在单元格中输入内容。

自定义条形图

以上我们讨论的自定义内容都是对于左边的树形表格，**Extjs** 甘特图也提供各种机制自定义右边的条形图。下面我们讨论的自定义，都是定义某一个或几个任务的条形图。

自定义样式

基于条形图的 **HTML** 结构和样式，通过修改样式的边框线、背景图片和背景色，达到修改外观的目的。

GanttPanel 提供一个 eventRenderer 回调函数，该函数在渲染条形图之前被调用，返回一个 css 样式或者一个 css 类。以下的例子演示了，当任务的周期超过 7 天，条形图高亮（背景变为红色）。

```
1. eventRenderer : function (item, resourceRec, tplData, row, col, ds) {
2.     if (Sch.util.Date.getDurationInDays(item.get('StartDate'), item.get('EndDate')) < 1) {
3.         return {
4.             tplData.style = 'background-color:red'; //条形图的样式
5.             return resourceRec.get('Name');
6.         };
7.     }
8. }
```

自定义提示框

ExtJS 甘特图控件有很多提示框效果，当鼠标移上，或进行某些操作的时候，提供更多丰富的信息帮助用户更好的操作。

```
1. tooltipTpl : new Ext.XTemplate(
2.     '<dl class="eventTip">',
3.         '<dt class="icon-clock">时间
4.         </dt><dd>[[Ext.Date.format(values.StartDate, "Y-m-d G:i)]]</dd>',
5.         '<dt class="icon-task">任务</dt><dd>{Title}</dd>',
6.         '<dt class="icon-earth">地址
7.         </dt><dd>{Location}&nbsp;</dd>',
8.         '</dl>'
9. ).compile(),
```

运行结果



时间刻度和自定义

甘特图提供多种不同的时间刻度的定义，用户也可以自定义自己的时间刻度。系统内置的时间刻度包括以下六种：

- hourAndDay 小时和天刻度
- dayAndWeek 天和周刻度
- weekAndDay 周和天刻度
- weekAndMonth 周和月刻度
- monthAndYear 月和年刻度
- year 年刻度

以下是自定义时间刻度的代码。

```
1.      Sch.preset.Manager.registerPreset("dayNightShift", {
2.          timeColumnWidth : 35,
3.          rowHeight : 32,
4.          displayDateFormat : 'G:i',
5.          shiftIncrement : 1,
6.          shiftUnit : "DAY",
7.          timeResolution : {
8.              unit : "MINUTE",
9.              increment : 15
10.         },
11.         defaultSpan : 24,
12.         headerConfig : {
13.             bottom : {
14.                 unit : "HOUR",
15.                 increment : 1,
16.                 dateFormat : 'G'
17.             },
18.             middle : {
19.                 unit : "HOUR",
20.                 increment : 12,
21.                 renderer : function(startDate, endDate, headerConfig, cellIdx)
22.                 {
23.                     // Setting align on the header config object
24.                     headerConfig.align = 'center';
25.
26.                     if (startDate.getHours()===0) {
27.                         // Setting a custom CSS on the header cell element
28.                         headerConfig.headerCls = 'nightShift';
29.                         return Ext.Date.format(startDate, 'M d') + ' Night
30.                         Shift';
31.                     }
32.                     else {
33.                         // Setting a custom CSS on the header cell element
34.                         headerConfig.headerCls = 'dayShift';
35.                         return Ext.Date.format(startDate, 'M d') + ' Day
36.                         Shift';
37.                     }
38.                 }
39.             },
40.             top : {
41.                 unit : "DAY",
42.                 increment : 1,
```

```
40.         dateFormat : 'd M Y'
41.     }
42. }
43. });
```

高级应用

大数据量和性能优化

甘特图所能支持的数据量，以及界面的渲染和操作性能，是衡量一个甘特图组件是否符合项目需要的重要指标。

为了支持超过万级超大数据量的甘特图，Extjs 提供一个数据缓冲视图的解决方案。使用这个方案非常简单，只需要在创建甘特图的时候，加入以下代码。

```
1. var sched = Ext.create("Sch.panel.SchedulerGrid", {
2.     verticalScroller: {
3.         xtype: 'paginggridscroller',
4.         activePrefetch: false
5.     },
6.
7. });
8.
9. var store = Ext.create('Ext.data.Store', {
10.    model: 'Employee',
11.    pageSize: 50,
12.    // allow the grid to interact with the paging scroller by buffering
13.    buffered: true,
14.    purgePageCount: 0,
15.    proxy: {
16.        type: 'memory'
17.    }
18. });
```

使用个数据缓冲视图的 Extjs 甘特图能支持 10 万条以上的数据量。在我们的测试中，内存开销和性能体验都非常好。

附录一 Extjs 中的 DataStore

和甘特图打交道，就不得不和 DataStore 打交道，甘特图中的数据是存放类型为 Store 的数据存储器中，通过指定甘特图中的 store 属性来设置表格中显示的数据，通过调用 store 的 load 或 reload 方法可以重新加载表格中的数据。ExtJS 中用来定义控件中使用数据的 API 位于 Ext.Data 命名空间中，本附录我们重点对 ExtJS 中的数据存储 Store 进行介绍。

Record

首先需要明确是，ExtJS 中有一个名为 Record 的类，表格等控件中使用的数据是存放在 Record 对象中，一个 Record 可以理解为关系数据表中的一行，也可以称为记录。Record 对象中即包含了记录（行中各列）的定义信息（也就是该记录包含哪些字段，每一个字段的数据类型等），同时又包含了记录具体的数据信息（也就是各个字段的值）。

我们来看直接使用 Record 的代码：

```
1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title'},
3.   {name: 'username', mapping: 'author'},
4.   {name: 'loginTimes', type: 'int'},
5.   {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
6. ]);
7. var r=new MyRecord({
8.   title:"日志标题",
9.   username:"easyjf",
10.  loginTimes:100,
```

```
11. loginTime:new Date()  
12. });  
13. alert(MyRecord.getField("username").mapping);  
14. alert(MyRecord.getField("lastLoginTime").type);
```

首先使用 Record 的 create 方法创建一个记录集 MyRecord, MyRecord 其实是一个类, 该类包含了记录集的定义信息, 可以通过 MyRecord 来创建包含字段值的 Record 对象。在上面的代码中, 最后的几条语句用来输出记录集的相关信息, MyRecord.getField("username") 可以得到记录中 username 列的字段信息, r.get("loginTimes") 可以得到记录 loginTimes 字段的值, 而 r.data.username 同样能得到记录集中 username 字段的值。

对 Record 有了一定的了解, 那么要操作记录集中的数据就非常简单了, 比如 r.set(name, value) 可以设置记录中某指定字段的值, r.dirty 可以得到当前记录是否有字段的值被更改过等等。

Store

Store 可以理解为数据存储器, 可以理解为客户端的小型数据表, 提供缓存等功能。在 ExtJS 中, GridPanel、ComboBox、DataView 等控件一般直接与 Store 打交道, 直接通过 store 来获得控件中需要展现的数据等。一个 Store 包含多个 Record, 同时 Store 又包含了数据来源, 数据解析器等相关信息, Store 通过调用具体的数据解析

器(DataReader)来解析指定类型或格式的数据(DataProxy),并转换成记录集的形式保存在Store中,作为其它控件的数据输入。

数据存储器由Ext.data.Store类定义,一个完整的数据存储器要知道数据源(DataProxy)及数据解析方式(DataReader)才能工作,在Ext.data.Store类中数据源由proxy配置属性定义、数据解析(读取)器由reader配置属性定义,一个较为按部就班创建Store的代码如下:

```
1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title'},
3.   {name: 'username', mapping: 'author'},
4.   {name: 'loginTimes', type: 'int'},
5.   {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
6. ]);
7. var dataProxy=new Ext.data.HttpProxy({url:"link.ejf"});
8. var theReader=new Ext.data.JsonReader({
9.   totalProperty: "results",
10.  root: "rows",
11.  id: "id"
12. },MyRecord);
13. var store=new Ext.data.Store({
14.  proxy:dataProxy,
15.  reader:theReader
16. });
```

当然,这样的难免代码较多,Store中本身提供了一些快捷创建Store的方式,比如上面的示例代码中可以不用先创建一个HttpProxy,只需要在创建Store的时候指定一个url配置参数,就会自动使用HttpProxy来加载参数。比如,上面的代码可以简化成:

```
1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title'},
```

```

3.  {name: 'username', mapping: 'author'},
4.  {name: 'loginTimes', type: 'int'},
5.  {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
6.  ]});
7.  var theReader=new Ext.data.JsonReader({
8.    totalProperty: "results",
9.    root: "rows",
10.   id: "id"
11.  },MyRecord);
12.  var store=new Ext.data.Store({
13.    url:"link.ejf",
14.    proxy:dataProxy,
15.    reader:theReader
16.  });

```

虽然不再需要手动创建 `HttpProxy` 了，但是仍然需要创建 `DataReader` 等，毕竟还是复杂，`ExtJS` 进一步把这种常用的数据存储器进行了封装，在 `Store` 类的基础上提供了 `SimpleStore`、`SimpleStore`、`GroupingStore` 等，直接使用 `SimpleStore`，则上面的代码可以进一步简化成下

面的内容：

```

1.  var store=new Ext.data.JsonStore({
2.    url:"link.ejf?cmd=list",
3.    totalProperty: "results",
4.    root: "rows",
5.    fields:['title', {name: 'username', mapping: 'author'},
6.    {name: 'loginTimes', type: 'int'},
7.    {name: 'lastLoginTime', mapping: 'loginTime', type: 'date'}
8.  ]
9.  });

```


DataReader

DataReader 表示数据读取器，也就是数据解析器，其负责把从服务器或者内存数组、xml 文档中获得的杂乱信息转换成 ExtJS 中的记录集 Record 数据对象，并存储到 Store 里面的记录集数组中。

数据解析器的基类由 Ext.data.DataReader 定义，其它具体的数据解析器都是该类的子类，ExtJS 中提供了读取二维数组、JSON 数据及 Xml 文档的三种数据解析器，分别用于把内存中的二维数组、JSON 格式的数据及 XML 文档信息解析成记录集。下面简单的介绍：

1) ArrayReader

Ext.data.ArrayReader 一数组解析器，用于读取二维数组中的信息，并转换成记录集 Record

对象。首先看下面的代码：

```
1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title', mapping:1},
3.   {name: 'username', mapping:2},
4.   {name: 'loginTimes', type:3}
5. ]);
6. var myReader = new Ext.data.ArrayReader({
7.   id: 0
8. }, MyRecord);
```

这里定义的 myReader 可以读取下面的二维数组：

```
1. [ [1, '测试', '小王',3], [2, '新年好', 'williamraym',13] ]
```

2) JsonReader

Ext.data.JsonReader—Json 数据解析器，用于读取 JSON 格式的数据信息，并转换成记录集 Record 对象。看下面使用 JsonReader 的代码：

```
1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title'},
3.   {name: 'username', mapping: 'author'},
4.   {name: 'loginTimes', type: 'int'}
5. ]);
6. var myReader = new Ext.data.JsonReader({
7.   totalProperty: "results",
8.   root: "rows",
9.   id: "id"
10. }, MyRecord);
```

这里的 JsonReader 可以解析下面的 JSON 数据：

```
1. { 'results': 2, 'rows': [
2.   { id: 1, title: '测试', author: '小王', loginTimes: 3 },
3.   { id: 2, title: 'Ben', author: 'williamraym', loginTimes:13} ]
4. }
```

JsonReader 还有比较特殊的用法，就是可以把 Store 中记录集的配置信息存放直接保存在从服务器端返回的 JSON 数据中，比如下面的例子：

```
1. var myReader = new Ext.data.JsonReader();
```

这一个不带任何参数的 myReader，可以处理从服务器端返回的下面 JSON 数据：

```

1. {
2.   'metaData': {
3.     totalProperty: 'results',
4.     root: 'rows',
5.     id: 'id',
6.     fields: [
7.       {name: 'title'},
8.       {name: 'username', mapping: 'author'},
9.       {name: 'loginTimes', type: 'int'} ]
10.  },
11.  'results': 2, 'rows': [
12.    { id: 1, title: '测试', author: '小王', loginTimes: 3 },
13.    { id: 2, title: '新年好', author: 'williamraym', loginTimes:13}]
14. }

```

3) XmlReader

Ext.data.XmlReader—XML 文档数据解析器，用于把 XML 文档数据转换成记录集 Record 对象。看下面的代码：

```

1. var MyRecord = Ext.data.Record.create([
2.   {name: 'title'},
3.   {name: 'username', mapping: 'author'},
4.   {name: 'loginTimes', type: 'int'}
5. ]);
6. var myReader = new Ext.data.XmlReader({
7.   totalRecords: "results",
8.   record: "rows",
9.   id: "id"
10. }, MyRecord);

```

上面的 myReader 能够解析下面的 xml 文档信息：

```

1. <topics>
2. <results>2</results>
3. <row>
4. <id>1</id>
5. <title>测试</ title >

```

```
6. <author>小王</ author >
7. <loginTimes>3</ loginTimes >
8. </row>
9. <row>
10. <id>2</id>
11. <title>新年好</ title >
12. <author> williamraym </ author >
13. <loginTimes>13</ loginTimes >
14. </row>
15. </topics>
```

DataProxy 和自定义 Store

DataProxy 字面解释就是数据代理，也可以理解为数据源，也即从哪儿或如何得到需要交给 DataReader 解析的数据。数据代理（源）基类由 Ext.data.DataProxy 定义，在 DataProxy 的基础，ExtJS 提供了 Ext.data.MemoryProxy、Ext.data.HttpProxy、Ext.data.ScriptTagProxy

等三个分别用于从客户端内存数据、Ajax 读取服务器端的数据及从跨域服务器中读取数据等三种实现。

比如像 SimpleStore 等存储器是直接从从客户端的内存数组中读取数据，此时就可以直接使用 Ext.data.MemoryProxy，而大多数需要从服务器端加载的数据直接使用 Ext.data.HttpProxy，HttpProxy 直接使用 Ext.Ajax 加载服务器的数据，由于这种请求是不能跨域的，所以要读取跨域服务器中的数据时就需要使用到 Ext.data.ScriptTagProxy。

在实际应用中，除了基本的从内存中读取 javascript 数组对象，从服务器读取 JSON 数组，从服务器取 xml 文档等形式的数据外，有时候还需要使用其它的数据读取方式。比如熟悉 EasyJWeb 中远程 Web 脚本调用引擎或 DWR 等框架的都知道，通过这些框架我们可以直接在客户端使用 javascript 调用服务器端业务组件的方法，并把服务器端的结果返回到客户端，客户端得到的是一个 javascript 对象或数组。由于这种方式的调用是异步的，因此，

相对来说有点特殊，即不能直接使用 Ext.data.MemoryProxy，也不能直接使用 Ext.data.HttpProxy，当然更不需要 Ext.data.ScriptTagProxy，这时候就需要创建自定义的 DataProxy 及 Store，然后使用这个自定义的 Store 来实现这种基于远程脚本调用引擎的框架得到数据。