```
        autoHeight: true,
        frame: true,
        items: [{
            xtype: 'easycombo',
            data: ['Boy', 'Girl']
        }],
        renderTo: document.body
    });
```

xtype 可以节省大量代码，它也让布局结构更加清晰明了，所以从某种程度上来讲 xtype 还是值得一用的。

9.3 实现一个功能完整的增、删、查、改表格控件

现在，我们已经对用户扩展组件的原理以及需要注意的事项有了一定的了解，下面就来实际操作一下，目标是实现一个自带增、删、查、改功能的表格控件，将通常需要进行上百行配置的 Grid 操作都封装在一个 EasyGrid 中。

9.3.1 扩展 GridPanel

既然我们要实现的是自带增、删、查、改全套功能的表格控件，当然就要把原本的 GridPanel 作为基类进行扩展了，下面就要使用 Ext.extend()函数通过 GridPanel 获得我们的 EasyGrid：

```
Ext.ux.EasyGrid=Ext.extend(Ext.grid.GridPanel, {});
```

然后在页面中创建一个 EasyGrid 测试一下：

```
Ext.onReady(function() {
    var EasyGrid=new Ext.ux.EasyGrid({
        title: 'EasyGrid',
        width: 400,
        renderTo: 'easyGrid'
    });
});
```

可是当我们打开页面以后，却看不到完整的表格效果（如图 9-2 所示），这是因为我们创建的 GridPanel 中缺少了 columns 参数，GridPanel 需要这个参数来定义自身的列模型。

图 9-2 刚开始 EasyGrid 无法正常显示

9.3.2 配置列模型

为了让 EasyGrid 能够正常显示，必须为它提供列模型，但是因为 GridPanel 中采用了 MVC 模型，ColumnModel 只属于用来显示数据的 View 层。之后为了在 EasyGrid 中显示数据，我们还需要设置 Model 层，并将 View 层与 Model 层的数据关联起来，这样相当于要将同样类型的数据配置两次。有没有办法一次将 View 层和 Model 层都配置好呢？

为此，我们使用了一个自定义的参数 fields，在页面中进行如下配置：

```
var EasyGrid=new Ext.ux.EasyGrid({
    title: 'EasyGrid',
```

```
    width: 400,
    fields: ['id', 'title', 'status', 'description'],
    renderTo: 'easyGrid'
});
```

在 fields 中我们设置了 4 列，分别是 id、title、status、description。

下一步将在 EasyGrid 中添加生成列模型的代码：

```
createColumns: function() {
    var cols=[];
    for (var i=0; i < this.fields.length; i++) {
        var f=this.fields[i];
        cols.push({
            header: f,
            dataIndex: f
        });
    }
    this.columns=cols;
}
```

这段代码将在组件初始化时执行，它会根据上面设置的 fields 参数生成对应的 columns，这样 EasyGrid 就知道该显示哪些列了。不过，现在我们还没有为 EasyGrid 设置显示的数据，所以页面上的表格显示还存在一些问题，如图 9-3 所示。

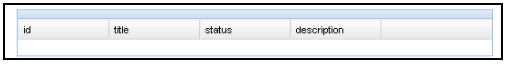

图 9-3 设置了列模型的 EasyGrid

9.3.3 配置显示数据

现在我们要为 EasyGrid 设置数据了，为简单起见，我们使用了一个二维数组作为初始数据：

```
var data=[
    ['1', 'message 1', 'normal', 'a normal description'],
    ['2', 'message 2', 'large', 'a large description'],
    ['3', 'message 3', 'small', 'a small description'],
    ['4', 'message 4', 'suspend', 'a suspend description'],
    ['5', 'message 5', 'unread', 'a unread description']
];
```

把这个数据设置到 EasyGrid 中：

```
var EasyGrid=new Ext.ux.EasyGrid({
    title: 'EasyGrid',
    width: 400,
    fields: ['id', 'title', 'status', 'description'],
    data: data,
    renderTo: 'easyGrid'
});
```

修改 EasyGrid，添加处理这部分数据的功能代码：

```
createStore: function() {
    this.store=new Ext.data.Store({
        autoLoad: true,
        proxy: new Ext.data.MemoryProxy(this.data),
        reader: new Ext.data.ArrayReader({
            fields: this.fields
        })
    });
},
```

设置过列模型和数据之后的 EasyGrid 已经可以正常显示了，如图 9-4 所示。

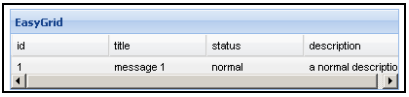

图 9-4 设置列模型和数据之后的 EasyGrid

9.3.4 点缀 EasyGrid

如果一切都采用默认配置，显示出来的 EasyGrid 并不好看，所以我们需要为 EasyGrid 再添加一些配置，让它更漂亮一些。

在 initComponent()中添加如下代码：

```
    this.autoHeight=true;
    this.stripeRows=true;
    this.viewConfig={
        forceFit: true
    };
```

- autoHeight 会让 EasyGrid 加载数据后自动计算高度，整个表格看起来就不会像是挤在一起了。
- stripeRows 表示在渲染时启用斑马线效果。
- forceFit 会自动计算每一列的宽度，保证所有列都能平均地显示在表格中。

设置完这些参数之后，我们再为 EasyGrid 上方加上工具条、下方加上分页工具条：

```
createTbar: function() {
    this.tbar=new Ext.Toolbar([{
        text: 'create',
        iconCls: 'x-tbar-loading'
    }, {
        text: 'update',
        iconCls: 'x-tbar-loading'
    }, {
        text: 'remove',
        iconCls: 'x-tbar-loading'
    }]);
},

createBbar: function() {
    this.bbar=new Ext.PagingToolbar({
        store: this.store,
    });
}
```

做完这些点缀之后，EasyGrid 就会变如图 9-5 所示。

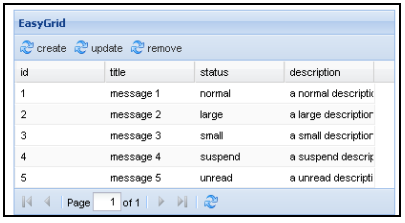

图 9-5 点缀之后的 EasyGrid

9.3.5 实现添加一条记录的功能

现在，我们希望单击工具条上的 create 按钮就可以弹出一个窗口，供我们添加一条新记录。

首先，需要把窗口和窗口中的表单都生成出来。

```
    var form=new Ext.form.FormPanel({
        frame: true,
        defaultType: 'textfield',
        buttonAlign: 'center',
        labelAlign: 'right',
        labelWidth: 70,
        trackResetOnLoad: true,
        reader: new Ext.data.ArrayReader({
            fields: this.fields
        }),
        items: items,
        buttons: [{
            text: 'submit'
        }, {
            text: 'reset',
            handler: function() {
                form.getForm().reset();
            }
        }]
    });
    return form;
},
```

这里可以看到，我们的表单也是根据最初的 fields 参数自动生成的，还是为了简单起见，我们这里把表单中所有的输入控件都当作 TextField，也没有使用任何布局，直接让所有的 field 自上而下顺序排列，最后再为 create 按钮添加一个 handler，在用户单击 create 按钮时就会弹出窗口显示编辑表单了。

为 create 按钮添加 handler 的代码如下所示：

```
    {
        text: 'create',
        iconCls: 'x-tbar-loading',
        handler: this.createRecord.createDelegate(this)
    }
```

对应的 createRecord()函数如下所示：

```
createRecord: function() {
    this.showWindow();
    var form=this.getForm();
    form.baseParams={
        create: true
    };
    form.setValues(this.getEmptyRecord());
},
```

这段代码会先显示窗口，然后获得对应的表单 form，将 form 的 baseParams 标记为 create:true。这就说明当前进行的操作是添加一条新记录，最后一步 setValues()是为了初始化表单中的数据，因为我们会一直使用同一个窗口和表单，如果不进行数据初始化，那么再次打开窗口时看到的很可能就是上次修改后的遗留数据了。

经过这些操作之后，我们单击 create 按钮就应该弹出如图 9-6 所示的窗口：

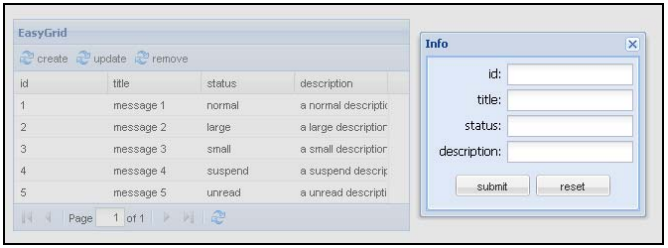

图 9-6 显示编辑表单

准备好窗口和表单之后，下一步的任务就是在用户填写完数据之后，将数据保存到 EasyGrid 的 store 中，这样 EasyGrid 就可以将新添加的数据显示到页面中。

用户通过表单填写数据的情景如图 9-7 所示：

用户通过表单填写数据后 EasyGrid 中的数据显示如图 9-8 所示：

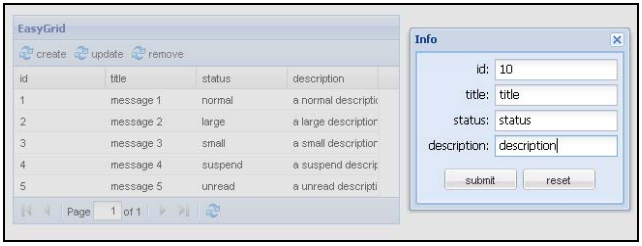

图 9-7 在表单中填写数据

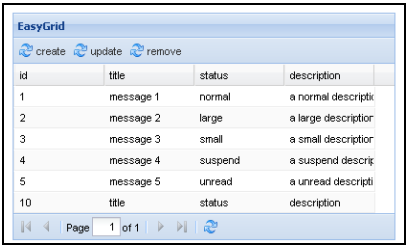

图 9-8 提交后数据将附加到 EasyGrid 中

为窗口中的 submit 按钮添加 handler：

```
{
    text: 'submit',
    handler: this.submitRecord.createDelegate(this)
}
```

对应的 submitRecord()函数如下所示：

```
submitRecord: function() {
    var form=this.getForm();
    var values=form.getFieldValues();
    if (form.baseParams.create) {
        var data=[];
        for (var name in values) {
            data.push(values[name]);
        }
        this.store.loadData([data], true);
    } else {
    }
    this.hideWindow();
},
```

submitRecord()函数中会依次执行如下几步操作：

- 通过 getForm()获得当前的表单 form；
- 通过 getFieldValues()获得表单中的所有数据；
- 判断当表单的 baseParams.create 为 true 时，执行添加一条新记录的操作；
- 创建一个数组，把表单中的数据依次放入数组；

- 将包含表单数据的数组加载到 store 里，因为第二个参数使用了 true，所以 store 会将数组中的数据附加到原有数据的后面；
- 隐藏操作窗口。

9.3.6 实现修改一条记录的功能

EasyGrid 中的修改操作与添加操作十分相似，都是弹出一个窗口，用户在窗口包含的表单中对数据进行编辑，最后单击 submit 按钮提交数据。上面已经介绍了窗口以及表单的创建，因此这里就不再重复了，下面来看一下修改操作和添加操作在功能上的区别。

首先要为 update 按钮设置 handler 参数，这样才能在用户单击按钮时执行对应的操作，代码如下所示：

```
{
    text: 'update',
    iconCls: 'x-tbar-loading',
    handler: this.updateRecord.createDelegate(this)
}
```

添加操作在每次弹出窗口时都是清空表单中的数据，修改操作在每次弹出窗口时也需要为表单赋值，这里就隐含着一个前提条件：用户必须事先选中一条记录，然后才能单击 update 按钮对这条记录进行修改。

为了判断用户是否已经选中了一条记录，我们编写了 getSelectedRecord()来进行判断，代码如下所示：

```
getSelectedRecord: function() {
    var sm=this.getSelectionModel();
    if (sm.getCount() == 0) {
        Ext.Msg.alert('Info', 'Please select one row.');
        return false;
    } else {
        return sm.getSelections()[0];
    }
},
```

这里获取了 EasyGrid 的 SelectionModel，当 sm.getCount()返回 0 时，表示当前用户还没有选中任何一条记录，这时就不能执行更新操作，为了让用户知道原因，我们会使用 MessageBox 显示一条提示信息，如图 9-9 所示。

图 9-9 未选中记录时的提示信息

在 updateRecord() 函数中，首先会判断 getSelectedRecord() 的返回值是否为 false，如果为 false，表示用户尚未选中需要修改的记录，这时就要中断操作，代码如下所示：

```
updateRecord: function() {
    var r=this.getSelectedRecord();
    if (r != false) {
        this.showWindow();
        var form=this.getForm();
        form.baseParams={
```

```
                create: false
            };
            form.loadRecord(r);
        }
    },
```

如果 getSelectedRecord()的返回值不为 false，说明用户已经选择了一条需要修改的记录，这时就要弹出窗口，并在 baseParams 中设置 create:false，表示当前进行的操作是对一条记录进行修改，最后调用 form 的 loadRecord()函数将用户选中的记录显示在表单中，如图 9-10 所示。

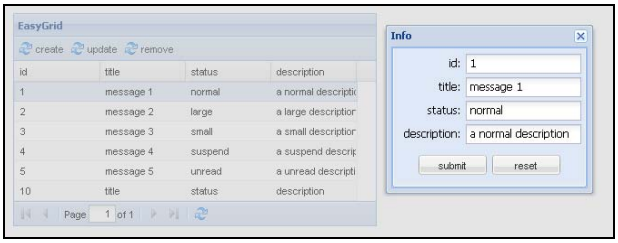

图 9-10 选中一条记录在弹出窗口中显示

这时可以对表单中的数据进行修改，如果对修改的数据不满意，还可以单击 reset 按钮将表单中的数据恢复到初始状态，重新修改。修改完成之后，可以单击 submit 按钮提交修改后的结果。

修改表单数据的效果如图 9-11 所示：

下面修改 submitRecord() 函数中的代码，让它支持修改操作，代码如下所示：

```
    submitRecord: function() {
        var form=this.getForm();
        var values=form.getFieldValues();
        if (form.baseParams.create) {
            var data=[];
            for (var name in values) {
                data.push(values[name]);
            }
            this.store.loadData([data], true);
        } else {
            var r=this.getSelectedRecord();
            r.beginEdit();
            for (var name in values) {
                r.set(name, values[name]);
            }
            r.endEdit();
        }
        this.hideWindow();
    },
```

在上述的 submitRecord 函数中，首先判断当前进行的是添加还是修改操作，当 baseParams 中的 create 参数不为 true 时，就执行修改操作。

提交修改结果时将执行如下步骤：

（1）通过 getSelectionRecord()获得用户选中的记录；

（2）执行 r.beginEdit()函数，开始修改 record 中的数据；

（3）将表单中的数据依次复制到 record 中；

（4）执行 r.endEdit()函数，结束修改操作。

这样就可以看到 EasyGrid 中的数据已经修改为对应的结果，如图 9-12 所示。

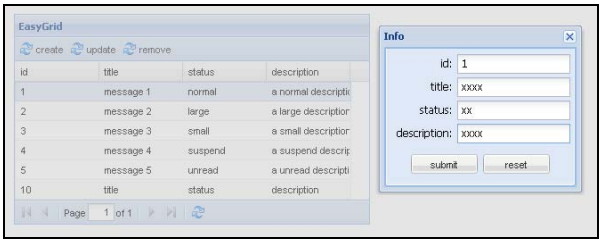

图 9-11 修改表单中的数据

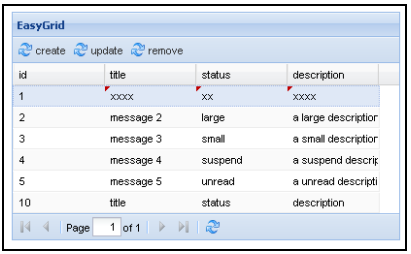

图 9-12 提交后 EasyGrid 中的显示结果

9.3.7 实现删除一条记录的功能

删除操作与更新操作有些类似，都需要用户预先选中一条记录，然后才能执行。下面可以看到，删除操作中也用到了更新操作中见过的 getSelectedRecord()函数。因此，在用户没有选择任何记录之前就去单击 remove 按钮时，也会弹出提示用户选择记录的 MessageBox。

首先为 remove 按钮添加 handler，代码如下所示：

```
{
    text: 'remove',
    iconCls: 'x-tbar-loading',
    handler: this.removeRecord.createDelegate(this)
}
```

对应的 removeRecord()函数代码如下所示：

```
removeRecord: function() {
    var r=this.getSelectedRecord();
    if (r != false) {
        Ext.Msg.confirm('Info', 'Are you sure?', function(btn) {
            if (btn == 'yes') {
                this.getStore().remove(r);
            }
        }, this);

    }
},
```

这里首先获得用户选择的记录，然后调用 Ext.Msg.confirm 弹出一个可以等待用户选择的对话框，如图 9-13 所示：

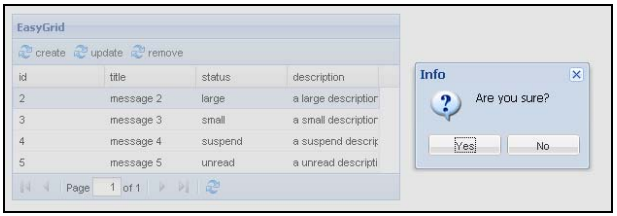

图 9-13 等待用户决定是否删除

这是为了避免用户执行误操作，所以再次让用户确认是否希望删除当前记录：如果用户选择

Yes，就会删除当前记录；如果用户选择 No，就会取消当前的操作。

到此为止，EasyGrid 已经完成了所有的增、删、查、改的基本功能。从中我们可以看到自定义扩展组件的威力，只需要了解组件的功能实现，就可以继承原有组件并将这些功能任意组合成自己需要的形式。一旦封装完成，之后只需要很少的代码就可以实现以前需要经过非常复杂的配置才能实现的功能。

EasyGrid 的完整代码如下所示：

```
Ext.ux.EasyGrid=Ext.extend(Ext.grid.GridPanel, {
    initComponent: function() {
        this.autoHeight=true;
        this.stripeRows=true;
        this.viewConfig={
            forceFit: true
        };

        this.createStore();
        this.createColumns();
        this.createTbar();
        this.createBbar();

        Ext.ux.EasyGrid.superclass.initComponent.call(this);
    },

    createStore: function() {
        this.store=new Ext.data.Store({
            autoLoad: true,
            proxy: new Ext.data.MemoryProxy(this.data),
            reader: new Ext.data.ArrayReader({
                fields: this.fields
            })
        });
    },

    createColumns: function() {
        var cols=[];
        for (var i=0; i < this.fields.length; i++) {
            var f=this.fields[i];
            cols.push({
                header: f,
                dataIndex: f
            });
        }
        this.columns=cols;
    },

    createTbar: function() {
        this.tbar=new Ext.Toolbar([{
            text: 'create',
            iconCls: 'x-tbar-loading',
            handler: this.createRecord.createDelegate(this)
        }, {
            text: 'update',
            iconCls: 'x-tbar-loading',
            handler: this.updateRecord.createDelegate(this)
        }, {
            text: 'remove',
            iconCls: 'x-tbar-loading',
            handler: this.removeRecord.createDelegate(this)
```

```
        }]);
    },
    createBbar: function() {
        this.bbar=new Ext.PagingToolbar({
            store: this.store,
        });
    },

    createRecord: function() {
        this.showWindow();
        var form=this.getForm();
        form.baseParams={
            create: true
        };
        form.setValues(this.getEmptyRecord());
    },

    updateRecord: function() {
        var r=this.getSelectedRecord();
        if (r != false) {
            this.showWindow();
            var form=this.getForm();
            form.baseParams={
                create: false
            };
            form.loadRecord(r);
        }
    },

    removeRecord: function() {
        var r=this.getSelectedRecord();
        if (r != false) {
            this.getStore().remove(r);
        }
    },

    getSelectedRecord: function() {
        var sm=this.getSelectionModel();
        if (sm.getCount() == 0) {
            Ext.Msg.alert('Info', 'Please select one row.');
            return false;
        } else {
            return sm.getSelections()[0];
        }
    },

    getEmptyRecord: function() {
        var r={};
        for (var i=0; i < this.fields.length; i++) {
            var f=this.fields[i];
            r[f]='';
        }
        return r;
    },

    submitRecord: function() {
        var form=this.getForm();
        var values=form.getFieldValues();
        if (form.baseParams.create) {
```

```
            var data=[];
            for (var name in values) {
                data.push(values[name]);
            }
            this.store.loadData([data], true);
        } else {
            var r=this.getSelectedRecord();
            r.beginEdit();
            for (var name in values) {
                r.set(name, values[name]);
            }
            r.endEdit();
        }
        this.hideWindow();
    },

    getForm: function() {
        return this.getFormPanel().getForm();
    },

    getFormPanel: function() {
        if (!this.gridForm) {
            this.gridForm=this.createForm();
        }
        return this.gridForm;
    },

    createForm: function() {
        var items=[];
        for (var i=0; i < this.fields.length; i++) {
            var f=this.fields[i];
            items.push({
                fieldLabel: f,
                name: f
            });
        }

        var form=new Ext.form.FormPanel({
            frame: true,
            defaultType: 'textfield',
            buttonAlign: 'center',
            labelAlign: 'right',
            labelWidth: 70,
            trackResetOnLoad: true,
            reader: new Ext.data.ArrayReader({
                fields: this.fields
            }),
            items: items,
            buttons: [{
                text: 'submit',
                handler: this.submitRecord.createDelegate(this)
            }, {
                text: 'reset',
                handler: function() {
                    form.getForm().reset();
                }
            }]
        });
        return form;
```

```
    },
    showWindow: function() {
        this.getWindow().show();
    },

    hideWindow: function() {
        this.getWindow().hide();
    },

    getWindow: function() {
        if (!this.gridWindow) {
            this.gridWindow=this.createWindow();
        }
        return this.gridWindow;
    },
    createWindow: function() {
        var formPanel=this.getFormPanel();

        var win=new Ext.Window({
            title: 'Info',
            closeAction: 'hide',
            modal: true,
            items: [
                formPanel
            ]
        });

        return win;
    }
});
```

9.4 从头实现 Ext 扩展

我们已经讨论过如何在原有组件的基础上实现 Ext 扩展组件，实际上我们也可以不基于任何组件实现自定义扩展。

比如下面这个 Ext.ux.Fisheye 鱼眼菜单就没有基于任何 Ext 组件，完全是从零开始编写的，其效果如图 9-14 和图 9-15 所示。

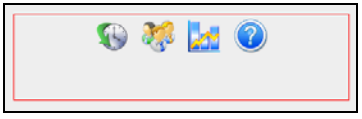

图 9-14 fisheye 菜单的初始状态

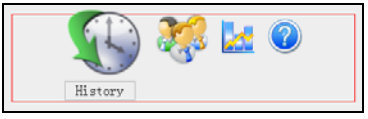

图 9-15 鼠标移上 fisheye 菜单的显示效果

实现代码如下所示：

```
/*-------------------------------------------------------------------------
 * Basic fisheye contributed by Ronald van Raaphorst
 *
 * Modified for Ext by Matja□ Lipuš
 *
 * Fisheye 1.1
 * version d.d. 8/8/2007
 *-------------------------------------------------------------------------*/
```

```
    Ext.namespace("Ext.ux");
    /**
     * @class Ext.ux.Fisheye
     * Fisheye
     * @cfg {Number} startSize Initial icon size
     * @cfg {Number} endSize Max icon size
     * @cfg {Number} hSpace Initial horizontal space between items
     * @cfg {Number} vDiff Vertical space factor to move images vertically
     * @constructor
     * @param {String/HTMLElement/Ext.Element} el The container element or DOM node,
or its id
     * @param {Object} config Configuration options
     */
    Ext.ux.Fisheye=function(el, config)
    {
            el=Ext.fly(el);

            config=Ext.applyIf(config || {}, {
                  startSize: 40,
                  endSize:   100,
                  hSpace:    15,
                  vDiff:     -25
            });
            Ext.apply(this, config);

            el.wrap({cls: el.dom.id + "-wrap fisheye"});

            var coord     =el.getXY();
            this.dSize    =this.endSize - this.startSize;
            this.top      =coord[1];
            this.left     =coord[0];
            this.hCenter=el.getWidth()/ 2; // horizontal center of the fisheye
            this.vCenter=this.top + this.dSize / 2; // height at which the images are
maximized
            this.vTop =this.vCenter - 100;   // top vertical influence sphere
            this.vBottom=this.vCenter + 100;  // bottom vertical influence sphere

            this.labelTresholdFactor=0.6; // Size factor [0-1] above which to display
the label
            this.items=Ext.query("img", el.dom);

            Ext.select("span", false, el.dom).setDisplayed(false);
            this.initElements();

            Ext.get(document).on("mousemove", this.onMouseMove, this);
    };

    Ext.ux.Fisheye.prototype={

            initElements: function()
            {
                  var itemWidth=this.startSize * this.items.length;
                  var itemSpace=this.hSpace * (this.items.length-1);
                  var half=(itemWidth + itemSpace) / 2;
                  var p   =this.hCenter - half;
                  for (var i=0, len=this.items.length; i<len; i++) {
                        this.setSize(this.items[i], this.startSize, this.startSize);
                        this.setPos(this.items[i], p, 0);
```

```
                        this.items[i].centerXPos=p + this.startSize / 2;
                        this.items[i].centerYPos=0 + this.startSize / 2;
                        p=p + this.startSize + this.hSpace;
                }
        },

        setSize: function(element, height, width)
        {
                element.style.width =width + "px";
                element.style.height=height + "px";
        },

        setPos: function(element, x, y)
        {
                element.style.left=x + "px";
                element.style.top =y + "px";
        },

        onMouseMove: function(event)
        {
                 event=event.browserEvent;
                 if (event.clientY < this.vTop || event.clientY > this.vBottom) {
                        return;
               }

                var vFactor;
                if (event.clientY < this.vCenter) {
                        vFactor=(event.clientY  -  this.vTop)  /  (this.vCenter  -
this.vTop);
                 } else {
                        vFactor=(this.vBottom  -  event.clientY)  /  (this.vBottom  -
this.vCenter);
                }

                var distance=0;
                var item;
                var totalWidth=-1 * this.hSpace;

                var hFactor, sizeFactor;

                for (var i=0, len=this.items.length; i<len; i++) {
                        item=this.items[i];
                        distance=Math.abs(item.centerXPos + this.left – event
.clientX);
                        if (distance > 100) {
                                hFactor=0;
                        } else {
                                hFactor=(100 - distance) / 100;
                        }

                        sizeFactor=vFactor * hFactor;
                        item.sizeFactor=sizeFactor;
                        item.newSize=this.startSize + sizeFactor * this.dSize;
                        item.newTop =this.vDiff * vFactor * item.sizeFactor;
                        totalWidth  =totalWidth + item.newSize + this.hSpace;
                }
                this.paint(totalWidth);
        },
```

```
        paint: function(totalWidth)
        {
                var xPos=this.hCenter - (totalWidth / 2) ;
                for (var i=0, len=this.items.length; i<len; i++) {
                        var item=this.items[i];

                        this.setPos(item, xPos, item.newTop );
                        this.setSize(item, item.newSize, item.newSize);

                        var x=Ext.query("span", item.parentNode);
                        if (item.sizeFactor > this.labelTresholdFactor) {
                                this.setPos(x[0], xPos, item.newTop + item.newSize);
                                x[0].style.display="block";
                        } else {
                                x[0].style.display="none";
                        }
                        xPos=xPos + item.newSize + this.hSpace;
                }
        }
};
```

从代码中可以看到，fisheye 中甚至没有使用 Ext.extend 实现 JavaScript 的对象继承。它使用的是最基本的 JavaScript 继承方式，仅仅依靠对 prototype 的操作定义了 Ext.ux.Fisheye 类。Fisheye 内部也仅仅使用了 Ext 提供的 fly()和 select()等函数实现对 DOM 的获取与操作。这也表明，即使不使用 Ext 中已提供的大量组件，只是依靠对 DOM 的基本操作也可以实现注入 Fisheye 之类的绚丽效果。

虽然 Fisheye 完全没有使用 Ext 提供的继承方式，但我们依然可以在页面中像使用其他组件一样来使用它，如下面的代码所示：

```
<html>
<head>
    <title>Fisheye example</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <style type="text/css">
        body {
            padding: 0;
            margin: 0;
            background: #eee;
        }

        div.menu-wrap {
            width: 400px;
        }
    </style>
    <link href="fisheye.css" type="text/css" rel="stylesheet"/>
    <script                                  src="../../../adapter/ext/ext-base.js"
type="text/javascript"></script>
    <script src="../../../ext-all.js" type="text/javascript"></script>
    <script src="fisheye.js" type="text/javascript"></script>
    <script type="text/javascript">
        Ext.onReady(function(){
            new Ext.ux.Fisheye("menu");
        });
    </script>
</head>
<body>
    <div style="padding: 0; margin-left: 300px; margin-top: 200px;">
        <ul id="menu">
```

```
            <li>
                <a href="#1">
                    <img src="images/history.png" alt="History"/>
                    <span>History</span>
                </a>
            </li>
            <li>
                <a href="#2">
                    <img src="images/users.png" alt="Users"/>
                    <span>Users</span>
                </a>
            </li>
            <li>
                <a href="#3">
                    <img src="images/graphs.png" alt="Graphs"/>
                    <span>Graphs</span>
                    </a>
            </li>
            <li>
                <a href="#4">
                    <img src="images/help.png" alt="Help"/>
                    <span>Help</span>
                </a>
            </li>
        </ul>
    </div>
</body>
</html>
```