如果要将 Ext.Element 实例赋值给一个变量，然后在后续代码中引用，请一定要使用 Ext.get 方法，不要使用 Ext.fly 方法。因为全局共享的 Ext.Element 实例有可能被后续代码修改，最终得不到你想要的效果，例如下面这段代码。

```
var el=Ext.fly('id1');
Ext.fly('id2').hide();
el.hide();
```

上面的代码本来是要隐藏 id 为“id1”和“id2”的 HTML 标签，但因为第 2 行已经修改了全局共享实例，el 变量的对象已经变为“id2”的 Ext.Element 实例，所以运行第 3 行时，并不能隐藏“id1”。

为什么有了 Ext.get 方法，还要增加一个 Ext.fly 方法呢？主要是为了减少内存的使用。试想一下，当创建一个 DOM 节点的 Ext.Element 实例时，就需要为其分配内存，如果页面需要操作许多的 DOM 节点时，内存开销就很大了。 在实际应用中，很多时候只是对某个节点执行一次性的操作，譬如修改节点的样式，这时候，如果使用全局共享的 Ext.Element 实例，就不用创建 Ext.Element 实例，从而节省内存的开销。

如果你只想返回 HTMLElement 对象，可使用 Ext.getDom 方法，其语法如下面的代码所示。

```
var el=Ext.getDom(el);
//el 可以为节点 id、DOM 节点或已存在的 Element
```

很多开发者在使用 Ext.Element 操作 DOM 节点本身的属性和方法时，经常会写以下这样的代码：

```
var el=Ext.get('elId');
el.innerHTML='Test';
```

造成这种错误的原因是：把 Ext.Element 对象当成了 HTMLElement 对象，正确的写法如下：

```
var el=Ext.get('elId');
el.dom.innerHTML='Test';
```

1.1.2 CSS 样式操作

Ext.Element 提供了 11 种操作样式方法，下面将介绍这些方法的功能和使用方法。

- addClass：为 element 增加样式类，其使用方法如下面代码所示。

```
//只增加一个样式类
Ext.fly('elId').addClass('elCss');
//增加多个样式类
Ext.fly('elId').addClass(['elCss1', 'elCss2',…, 'elCssN']);
```

- radioClass：与 addClass 类似，区别在于，该方法会移除相同的样式类，其使用方法可参考 addClass。
- removeClass：移除一个或多个样式类，其使用方法可参考 addClass。
- toggleClass：样式类开关。如果 element 已存在某个样式类，执行该方法将移除该样式类；如果不存在，则增加该样式类，其使用方法如下面的代码所示。

```
//假设 elId 不存在样式类 elCss
Ext.fly('elId').toggleClass('elCss'); //为 elId 增加类 elCss
Ext.fly('elId').toggleClass('elCss'); //删除 elId 的类 elCss
Ext.fly('elId').toggleClass('elCss'); //再次为 elId 增加类 elCss
```

- hasClass：检查 element 是否已应用指定的样式类，其使用方法如下面的代码所示。

```
If(Ext.fly('elId').hasClass('elCss')){
  //已应用类 elCss 时执行
}
```

- replaceClass：替换一个样式类，其使用方法请看下面的代码。

```
//使用样式类 elCss2 替换 elCss1
Ext.fly('elId').replaceClass('elCss1','elCss2');
```

- getStyle：返回 element 的某个样式属性值，其使用方法请看下面的代码。

```
var color=Ext.fly('elId').getStyle('color');
```

- setStyle：设置样式属性，其使用方法请看下面的代码。

```
//只设置一个属性
Ext.fly('elId').setStyle('color', '#FFFFFF');
//设置多个属性
Ext.fly('elId').setStyle({
  color: 'red',
  background: 'yellow',
  font-weight: 'bold'
})
```

- getColor：根据传送的属性返回 element 的颜色值，例如传送“background-color”，则返回背景颜色。要注意的是，无论颜色是使用 RGB 法还是 3 位十六进制法设置的，都会以 6 位十六进制格式返回。例外的是，使用预定义颜色表示法设置的颜色值，会返回预定义的颜色名称。其使用方法及其返回结果如下面的代码所示。

```
//背景颜色为 rgb(221,221,221)
var bgColor=Ext.fly('elId').getColor('background-color');
//返回"#dddddd"
//颜色为#ddd
var color=Ext.fly('elId').getColor('color');
//返回"#dddddd"
//颜色为 yellow
var color=Ext.fly('elId').getColor('color');
//返回"#yellow"
```

- setOpacity：设置 element 的 Opacity 值，其使用方法请看下面的代码。

```
Ext.fly('elId').setOpacity(.5);
//使用动画过渡
Ext.fly('elId').setOpacity(.5,true);
//使用指定动画样式过渡
Ext.fly('elId').setOpacity(.5, {duration: .35, easing: 'easeIn'});
```

- clearOpacity：清除 element 的 Opacity 设置，其使用方法请看下面的代码。

```
Ext.fly('elId').clearOpacity();
```

1.1.3 DOM 查询与遍历

Ext.Element 提供了 13 种 DOM 查询与遍历方法，下面简单介绍一下这些方法的功能及其使用方法。

- is：判断当前 element 是否与选择器选择的 element 匹配，其使用方法请看下面的代码。

```
var el=Ext.get('elId');
if(el.is('div.elCss')){
   //如果匹配，则执行该处代码
}
```

- findParent：从当前节点开始查找与选择器匹配的父节点（包含当前节点）。要注意该方法默认返回的是 HTMLElement 对象，不是 Ext.Element 对象。如果需要返回 Ext.Element 对象，则需要设置第 3 个参数为 true。其使用方法如下面的代码所示。

```
var el=Ext.fly('elId').findParent('div'); //返回 HTMLElement
//定位到在当前节点的第 4 层父节点
var el= Ext.fly('elId').findParent('div',4);
//返回 Ext.Element
var el= Ext.fly('elId').findParent('div',null,true);
```

- findParentNode：从当前节点开始查找与选择器匹配的父节点，该方法的使用方法请参考 findParent 方法。
- up：是 findParentNode 方法的缩写，不过该方法返回的是 Ext.Element 对象，其使用方法请参考 findParent 方法。
- select：通过选择器选择当前节点下的子节点，该方法的返回值是 Ext.CompositeElement 对象，其使用方法请看下面的代码。

```
Ext.fly('elId').select('div');
//如果希望返回的节点是 Ext.Element 对象，需要设置第 2 个参数为 true
Ext.fly('elId').select('div',true);
```

注意 Ext.CompositeElement 对象是不允许直接通过索引访问其内部元素的，要访问内部元素需要使用 item 方法。注意，是 item 方法，不是属性，要使用括号（而不是中括号）包含索引，要遍历对象中的元素可使用 each 方法。通过 getCount 方法可获取元素的总数。要定位元素可使用 indexOf 方法。

- query：通过选择器选择 DOM 节点，该方法返回包含 DOM 节点的数组，其使用方法请看下面的代码。

```
Ext.fly('elId').query('div');
// 上面这句也可以使用以下语句代替
Ext.query('div', Ext.getDom('elId'));
```

- child：通过选择器返回一个当前节点的子节点，默认返回的是 Ext.Element 对象，如果需要返回 HTMLElement 对象，第 2 个参数需要设置为 true，其使用方法请看下面的代码。

```
Ext.fly('elId').child('div'); //返回 Ext.Element 对象
Ext.fly('elId').child('div', true); //返回 HTMLElement 对象
```

- down：通过选择返回一个当前节点的直接子节点，该方法与 child 方法类似，唯一不同的是，选择器不能包含节点 id，其使用方法可参考 child 方法。
- parent：返回当前节点的父节点。该方法通过设置第 2 个参数可控制返回的是 Ext.Element 对象还是 HTMLElement 对象，其使用语法如下面的代码所示。

```
Ext.fly('elId').parent(); //返回 Ext.Element 对象
Ext.fly('elId').parent('',true); //返回 HTMLElement 对象
//也可以通过选择器选择
Ext.fly('elId').parent('div');
```

- next：返回与当前节点同层的、除了文本外的下一个节点。该方法的使用方法与 parent 方法类似。
- prev：返回与当前节点同层的除了文本外的上一个节点。该方法的使用方法与 parent 方法类似。
- first：返回与当前节点同层的第一个节点。该方法的使用方法与 parent 方法类似。
- last：返回与当前节点同层的最后一个节点。该方法的使用方法与 parent 方法类似。

1.1.4 DOM 操作

在 DOM 操作中会经常使用到 Ext.DomHelper 对象，因此在介绍 DOM 操作之前我们需要先了解一下 Ext.DomHelper 对象。

Ext.DomHelper 是一个用来生成 HTML 片段的类，它主要通过定义一个 JSON 格式的数据生成 HTML 片段，对开发人员来说，非常灵活方便。它的数据结构主要包括以下 4 个属性：

- tag：元素的标签，例如 div、span 之类。
- children：由元素的子元素组成的数组，可以通过该属性不断增加子元素。
- cls：元素的 CSS 类名。
- html：元素的 innerHTML 属性，如果不想使用 children 属性定义元素的内部 HTML 内容，可使用该属性代替。

譬如要生成以下的 HTML 片段：

```
<ul id='itemList' class='list'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

其数据定义如下：

```
var list={
  id: 'itemList',
  tag: 'ul',
  cls: 'list',
  children:[
    {tag: 'li',html: '1'},
    {tag: 'li',html: '2'},
    {tag: 'li',html: '3'}
  ]
}
```

当然了，如果你不喜欢使用 `children` 属性，也可以定义如下：

```
var list={
  id: 'itemList',
  tag: 'ul',
  cls: 'list',
  html: '<li>1</li><li>2</li><li>3</li>'
}
```

注意　在目前版本的 Ext Core 中不支持 createTemplate 方法，因此使用 DomHelper 的模板功能，需要加载 Ext 3 完整包中的 DomeHelpher-more.js 文件。

如果需要添加元素的其他属性，例如 id、target 等，可以直接将属性名称作为 JSON 数据的标记附加到数据结构上，请看下面这段 JSON 数据代码：

```
{
  id: 'link1',
  tag: 'a',
  href: 'url',
  target: '_blank',
  html: '链接'
}
```

该段代码将会生成以下的 HTML 代码：

```
<a target='_blank' href='url' id='link1'>链接</a>
```

从上面的例子可以看到，使用 Ext.DomHelper 的优点是代码简单明了、容易维护，是使用 Ext Core 必须掌握的知识。

下面将介绍 14 种 DOM 操作方法的功能及其使用方法。

- appendChild：在当前节点里追加子节点，其使用方法请看下面的代码。

```
var el=Ext.get('elId1')
Ext.fly('elId').appendChild('elId1'); //通过 id 追加
//与上一句作用一样，通过 Ext.Element 追加
Ext.fly('elId').appendChild(el);
//通过数组追加
Ext.fly('elId').appendChild(['elId1', 'elId2']);
//通过 HTMLElement 追加
Ext.fly('elId').appendChild(el.dom);
//通过 CompositeElement 追加
Ext.fly('elId').appendChild(Ext.select('div'));
```

- appendTo：将当前节点追加到某个节点。其使用方法请看下面的代码。

```
var el=Ext.get('elId1')
Ext.fly('elId').appendTo('elId1'); //通过 id 追加
//与上一句作用一样，通过 Ext.Element 追加
Ext.fly('elId').appendChild(el);
```

- insertBefore：将当前节点插入某个节点之前。其使用方法可参考 appendTo 方法。
- insertAfter：将当前节点插入某个节点之后。其使用方法可参考 appendTo 方法。
- insertFirst：在当前节点中插入 1 个子节点并作为当前节点的第一个子节点。其使用方法请看下面的代码。

```
var el=Ext.get('elId1')
Ext.fly('elId').insertFirst('elId1'); //通过 id 添加
//与上一句作用一样，通过 Ext.Element 添加
Ext.fly('elId').insertFirst(el);
//通过 DomHelper 添加
Ext.fly('elId').insertFirst({
  tag: 'div',
  cls:  'box',
  html: 'hello'
})
```

- replace：使用当前节点替换某个节点。其使用方法可参考 appendTo 方法。
- replaceWith：将当前节点替换为某个已存在节点或新节点。其使用方法可参考 insertFirst 方法。

- createChild：在当前节点追加或在指定的节点前插入 1 个由 DomHelper 定义的新节点，其使用方法请看下面的代码。

```
var el= Ext.get('elId');
var c={
  tag: 'div',
  cls:  'box',
  html: 'hello'
};
//追加 1 个子节点
el.createChild(c);
//在第一个子节点之前插入
el.createChild(c,el.first());
```

- wrap：在当前节点外绑定一个由 DomHelper 创建的父节点。其使用方法请看下面的代码。

```
//假设 elid 的 html 代码为: <div id='elId'>test</div>
Ext.fly('elId').wrap();
//执行上面代码后，html 代码变成:
//<div id="随机创建的 id"><div id="t1">test </div></div>
Ext.fly('elId').wrap({
  tag: 'p',
  id: 'elId1'
  html: 'new test'
});
//执行后 html 代码变成:
// <p id='elId1'>new test <div id='elId'>test</div></p>
```

- insertHTML：在当前节点插入 HTML 代码。该方法需要指定插入位置（beforeBegin、beforeEnd、afterBegin 和 afterEnd）。该方法默认返回 HTMLElement 对象，如果需要返回 Ext.Element 对象，需要设置第 3 个参数为 true。其使用方法及插入位置请看下面的介绍。

假设当前的 html 代码如下：

```
 <ul id='elId'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

执行以下代码（位置是“beforeBegin”）：

```
Ext.fly('elId').insertHtml('beforeBegin', '<p>插入的代码</p>');
```

则结果将是以下代码：

```
<p>插入的代码</p>
<ul id='elId'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

如果执行以下代码（位置是“afterBegin”）：

```
Ext.fly('elId').insertHtml('afterBegin', '<p>插入的代码</p>');
```

则结果将是以下代码：

```
<ul id='elId'>
  <p>插入的代码</p>
  <li>1</li>
```

```
  <li>2</li>
  <li>3</li>
</ul>
```

如果执行以下代码（位置是“beforeEnd”）：

```
Ext.fly('elId').insertHtml('beforeEnd', '<p>插入的代码</p>');
```

则结果将是以下代码：

```
<ul id='elId'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <p>插入的代码</p>
</ul>
```

如果执行以下代码（位置是“afterEnd”）：

```
Ext.fly('elId').insertHtml('afterEnd', '<p>插入的代码</p>');
```

则结果将是以下代码：

```
<ul id='elId'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
  <p>插入的代码</p>
```

- remove：删除当前节点。其使用方法请看下面的代码。

```
Ext.fly('elId').remove();
```

- removeNode：在 DOM 树中删除一个节点。其使用方法如下面的代码所示。

```
Ext.removeNode(node);//node 为 HTMLElement 对象
```

- load：使用 Ajax 调用远程数据更新节点内容。其使用方法如下面的代码所示。

```
Ext.fly('elId').load({url: 'test.aspx'})
```

- getUpdater：返回当前节点的 Ext.Updater 对象。其使用方法如下面的代码所示。

```
var u= Ext.fly('elId').getUpdater();
//通过 update 方法更新节点内容
u.update({
  url: 'test.aspx'
});
```

1.1.5 事件处理

由于各浏览器的事件处理模型不同，所以要实现框架的跨浏览器特性，就需要构建一个通用的处理模型，以适应各种浏览器。Ext Core 通过 Ext.EventObject 对象实现了事件处理的跨浏览器特性。

在 Ext 中要绑定或删除事件非常简单，主要有 4 种方法，请看下面的介绍。

- addListener/on：为节点绑定事件。on 是 addListener 的简写，其使用方法如下所示。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //参数 e 是 Ext.EventObject 对象，非浏览器产生的事件对象
    //参数 t 是触发事件的 Ext.Element 对象
});
```

```
//也可以一次定义多个事件
el.on({
  'click':{
    fn:function(e,t){},
    scope:this,
    delay:100
  },
  'mouseover'{
      fn:function(e,t){}
      scope:this
  }
});
```

- removeListener/un：删除节点的监听事件。un 是 removeListener 的简写，其使用方法如下所示。

```
var el=Ext.get('elId');
el.un('click',this.handlerFn);
```

在 Ext Core 中还有几个配置方便的高级事件处理方法，如事件委托、缓冲和延时等。这些配置的具体使用请看下面的介绍。

- delegation：事件委托是一种减少内存消耗和防止内存泄漏的技术，其基本思路是基于 DOM 的事件传播规则（具体信息可参考相关 JavaScript 书籍），将事件绑定在一组元素的容器元素上，而不是分别为该组元素的每一个元素绑定事件。但这并不意味着可以将所有事件绑定在文档的 body 元素上，原因是页面内所有事件都执行同一个函数，只会产生消极的影响并降低性能。这项技术一般用于一个下拉日期对话框或由一个容器元素直接包含或几乎直接包含的一组元素，譬如以下 HTML 代码：

```
<ul id='elId'>
  <li id='item1'>1</li>
  <li id='item2'>2</li>
  <li id='item3'>3</li>
</ul>
```

通常的做法是为每一个 li 元素绑定事件，请看以下代码：

```
Ext.fly('item1').on('click',function(e,t){
   //处理过程
)};
Ext.fly('item2').on('click',function(e,t){
   //处理过程
)};
Ext.fly('item3').on('click',function(e,t){
   //处理过程
)};
```

使用事件委托技术则可以修改为以下代码：

```
Ext.fly('elId').on('click',function(e,t){
   switch(t.id){
      case 'item1':
       //处理过程
       break;
      case 'item2':
       //处理过程
       break;
      case 'item3':
       //处理过程
       break;
```

```
    }
)};
```

基于 DOM 的事件传播规则，容器 elId 内的子元素的事件会传播到容器 elId，这样就触发了 elId 绑定的函数，通过返回的子元素 id 就可以执行相应的代码。从例子中可以看到，事件委托的另一个优点就是无论子节点如何变化，都不需要增减事件绑定代码，只要在 switch 中添加相应处理代码即可，这样就方便了代码的维护。

- delegate：这是 on 方法的第 4 个参数的一个配置选项，其作用是通过一个选择器选择触发事件的目标。譬如以下的 HTML 代码：

```
<ul id='elId'>
  <li id='item1'>1</li>
  <li id='item2' class='list'>2</li>
  <li id='item3'>3</li>
</ul>
```

在下面的代码中，将为元素 elId 中绑定“click”事件，并设置 delgate 参数值为“.list”。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    console.log(t.id);  //结果将显示为"item2"
  },
  this,
  {
    delegate: '.list'
  }
);
```

从代码中可以清楚了解到，delegate 将不符合选择要求的项都过滤了，只有类名为“.list”的项才会执行事件代码。

- hover：该方法是 Ext Core 实现跨浏览器的悬停效果的解决方案，它将确保鼠标在进入或离开元素时执行相应的函数。在 IE 的原有事件模型里，当鼠标在子元素移动时，浏览器会过滤掉 mouserenter 和 mouserleave 事件。Ext Core 则避免了这种情况，从而实现了悬停效果的跨浏览器特性。其使用方法请看下面的代码。

```
function enter(e,t){
    t.toggleClass('red');
};

function leave(e,t){
    t.toggleClass('red');
};

  Ext.fly('elId').hover(over,out);
```

- removeAllListeners：删除节点所有的监听事件，其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.removeAllListeners('click',this.handlerFn);
```

- single：这是 on 方法第 4 个参数的一个配置选项。当设置该值为 true 时，事件在执行一次后将自动删除。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
```

```
    this,
    {
      single: true  //只会执行一次单击事件
    }
  );
```

- buffer：这是 on 方法第 4 个参数的一个配置选项。其作用是在其设置的时间范围（单位是毫秒）内，将只执行一次事件处理。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
  this,
  {
    buffer: 1000  //从事件触发开始，1 秒后才会再次响应
  }
);
```

- delay：这是 on 方法第 4 个参数的一个配置选项。其作用是在事件触发后，等待设置的时间值（单位是毫秒）后，再执行处理函数。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
  this,
  {
    delay: 1000  //从事件触发开始，1 秒后才会执行处理函数
  }
);
```

- target：这是 on 方法第 4 个参数的一个配置选项，其作用是只有在事件传播到你设定的目标元素时才执行处理函数。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
  this,
  {
    target: el.up('div')  //事件传播到第一个 div 元素时才执行处理函数
  }
);
```

- normalized：这是 on 方法的第 4 个参数的一个配置选项。当设置该参数为 false 时，传递到处理函数的将是浏览器的事件句柄，而不是 Ext.EventObject。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //e 将是浏览器事件句柄，不是 Ext.EventObject
  },
  this,
  {
    normalized: false
  }
);
```

- stopPropagation：这是 on 方法第 4 个参数的一个配置选项。当设置为 true 时，将会停止事

件传播。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
  this,
  {
    stopPropagation: true //事件将不继续往上传播
  }
);
```

- preventDefault：这是 on 方法第 4 个参数的一个配置选项。当设置为 true 时，将屏蔽默认操作。其使用方法如下面的代码所示。

```
var el=Ext.get('elId');
el.on('click',function(e,t){
    //处理过程
  },
  this,
  {
    preventDefault: true //停止默认操作
  }
);
```

- stopEvent：这是 on 方法的第 4 个参数的一个配置选项。当设置为 true 时，将停止当前事件传播和屏蔽默认操作。该项一般使用在需要使用自己右键菜单替换浏览器默认右键菜单的地方。其使用方法请看下面的代码。

```
var el=Ext.get('elId');
el.on('contentmenu',function(e,t){
    //弹出自己的右键菜单
  },
  this,
  {
    stopEvent: true //停止事件
  }
);
```

1.1.6 尺寸大小

在页面中，通常需要获取或修改页面元素的尺寸或大小。在 Ext Core 中提供了一个方法来获取或修改元素尺寸大小，而且某些方法还可以配置动画。下面开始简单介绍这些方法的使用。

- getHieght：获取元素的高度（offset height）。其使用方法如下面的代码所示。

```
var height = Ext.fly('elId').getHieght();
```

- getWidth：获取元素的宽度（offset width）。其使用方法如下面的代码所示。

```
var width = Ext.fly('elId').getWidth();
```

- setHeight：设置元素高度。其使用方法如下面的代码所示。

```
Ext.fly('elId').setHeight(200);
//使用默认动画方式修改高度
Ext.fly('elId').setHeight(200，true);
//自定义动画方式修改高度
Ext.fly('elId').setHeight(200，{
```

```
    duration :.5 //动画的持续时间
    callback :function(){//处理} //动画结束回调函数
});
```

- setWidth：设置元素宽度。其具体使用方法请参考 setHeight 方法。
- getBorderWidth：获取元素指定边的宽度。4 条边分别用 t（上）、l（左）、r（右）、b（下）4 个字母表示，也可以获取由这 4 个字母组合而成的宽度合值，例如“lr”，表示可获得左边和右边宽度的合值。其使用方法如下面的代码所示。

```
var borderWidth = Ext.fly('elId').getBorderWidth('lr');
```

- getPadding：获取元素指定边的边距。与 getBorderWidth 一样，使用 t、l、r、b 4 个字母表示 4 条边，也可以通过组合获取合值。其使用方法可参考 getBorderWidth 方法。
- clip：保存元素当前的 overflow 设置和 clip 设置。可使用 unclip 取消。其使用方法请看下面的代码。

```
Ext.fly('elId').clip ();
```

- unclip：将执行 clip 方法之后的 clip 设置还原回原来的设置。其使用方法请看下面的代码。

```
Ext.fly('elId').unclip();
```

- isBorderBox：检查浏览器是否使用了 border box 规则。如果返回 true，表示浏览器是 IE 且使用了非严格模式盒子模型。其使用方法请看下面的代码。

```
if(Ext.isBorderBox){
  //处理过程
}
```

1.1.7 定位功能

通过 Ext Core 的定位 API，可方便实现跨浏览器的获取或设置元素的位置。类似尺寸 API，可在方法中加入动画效果。下面开始简单介绍这 22 个方法的使用方法。

- getX：获取元素基于页面坐标的 X 轴位置。元素只有在 DOM 树下才可以获取坐标值，如果 display 属性为 none 或还没插入 DOM 树，则返回 false。其使用方法请看下面的代码。

```
var x=Ext.fly('elId').getX();
```

- getY：获取元素基于页面坐标的 Y 轴位置。与 getX 方法一样，元素只有在 DOM 树下才可以获取坐标值。其使用方法请参考 getX 方法。
- getXY：获取元素基于页面坐标的坐标值，返回值为数组。该方法与 getX 方法一样，元素只有在 DOM 树下才可以获取坐标值。其使用方法请参考 getX 方法。
- setX：设置元素 X 坐标。该方法与 getX 方法一样，元素只有在 DOM 树下才可以设置。其使用方法请看下面代码。

```
Ext.fly('elId').setX(10);
```

- setY：设置元素 Y 坐标。该方法与 getX 方法一样，元素只有在 DOM 树下才可以设置坐标值。其使用方法请参考 setX 方法。
- setXY：设置元素坐标。该方法与 getX 方法一样，元素只有在 DOM 树下才可以设置坐标值。其使用方法请看下面代码。

```
Ext.fly('elId').setXY([10,10]);
```

- getOffsetsTo：返回元素相对于另一元素的间距。该方法返回的是数组值。该方法与 getX 方法一样，元素只有在 DOM 树下才可以设置坐标值。其使用方法请看下面代码。

```
var offset = Ext.fly('elId').getOffsetsTo(Ext.fly('elId1'));
```

- getLeft：返回元素左边的 X 坐标。如果设置参数为 true，则返回基于 css 定义的坐标值，而不是基于页面的坐标值。其使用方法请看下面代码。

```
var x = Ext.fly('elId').getLeft();
```

- getRight：返回元素右边的 X 坐标，实际值为元素的 X 坐标加上元素宽度。与 getleft 方法一样，设置参数为 true 则返回基于 css 定义的坐标值。其使用方法可参考 getLeft 方法。
- getTop：返回元素顶部的 Y 坐标。与 getleft 方法一样，设置参数为 true 则返回基于 css 定义的坐标值。其使用方法可参考 getLeft 方法。
- getBottom：返回元素底部的 Y 坐标，实际值为元素 Y 坐标加上元素高度。与 getleft 方法一样，设置参数为 true 则返回基于 css 定义的坐标值。其使用方法可参考 getLeft 方法。
- setLeft：设置元素样式 left 属性的值。其使用方法请看下面代码。

```
Ext.fly('elId').setLeft(100);
```

- setRight：设置元素样式 right 属性的值。其使用方法请参考 setLeft 方法。
- setTop：设置元素样式 top 属性的值。其使用方法请参考 setLeft 方法。
- setBottom：设置元素样式 bottom 属性的值。其使用方法请参考 setLeft 方法。
- setLocation：设置元素基于页面坐标的坐标值。其使用方法请看下面代码。

```
Ext.fly('elId').setLocation(100，200);
```

- moveTo：设置元素基于页面坐标的坐标值。该方法可设置位置改变时是否使用动画。其使用方法请看下面代码。

```
Ext.fly('elId').moveTo(100，200);
//使用默认动画
Ext.fly('elId').moveTo(100，200,true);
//使用自定义动画
Ext.fly('elId').moveTo(100，200,{
  duration :.5 //动画的持续时间
  callback :function(){//处理} //动画结束回调函数
});
```

- position：预置元素的 position 属性。如果没有设置 position 属性，该方法会设置 position 属性为“relative”。其使用方法请看下面代码。

```
//设置为 relative
Ext.fly('elId').position('relative');
//设置为 absolute，z-Index 为 1000，x 坐标为 100，坐标为 200
Ext.fly('elId').position('absolute',1000，100,200);
```

- clearPositioning：当文档已经加载完成，将元素 position 属性并设置回默认值。其使用方法请看下面代码。

```
Ext.fly('elId').cearPositioning();
Ext.fly('elId').cearPositioning('top');
```

- getPositioning：获取元素的 position 属性。通常与 setPostitioning 方法一起使用，在更新元素后用来恢复设置。其使用方法请看下面代码。

```
var pos=Ext.fly('elId').getPositioning();
```

- setPositioning：设置元素 position 属性。其使用方法请看下面代码。

```
Ext.fly('elId').setPositioning({
  left: 'static',
  right: 'auto'
});
```

- translatePoints：修改元素的 left 属性值和 top 属性。其使用方法请看下面代码。

```
Ext.fly('elId').translatePoints([100,200]);
Ext.fly('elId').translatePoints(100,200);
```

1.1.8 动画功能

Ext Core 通过预置的动画功能，可以让开发人员轻松实现动画功能。通过自定义动画配置，可实现更多的动画效果。在动画完成后，开发人员可通过回调函数实现后续处理。

Ext Core 动画插件可选择 8 个不同定位点来开始或结束动画。这 8 个定位点的值与描述请看表 1-1。

表 1-1 Ext Core 动画 8 个定位点的值与描述

值	描述
tl	左上角
t	顶边中心
tr	右上角
l	左边中心
r	右边中心
bl	左下角
b	底边中心
br	右下角

下面开始介绍 Ext Core 预置的 12 个动画功能。

- slideIn/slideOut：元素的滑进或滑出效果。默认 slideIn 是从顶部滑进的，而 slideOut 是从底部滑出的，可通过修改第 1 个参数设置滑进或滑出位置。其使用方法请看下面代码。

```
var el = Ext.get('elId');
//从顶部滑进
el.slideIn();
//从底部滑出
el.slideOut();
//从左边滑进
el.slideIn('l',{
  easing: 'easeOut',
  duration: .5
});
```

- puff：元素慢慢向四周扩大并逐渐消失。当效果完成后，元素将隐藏（visibility 属性为 hidden），不过元素原来所占位置将继续存在。如果需要删除元素，请设置 remove 属性为

true。其使用方法请看下面代码。

```
var el = Ext.get('elId');
//默认方式
el.puff();
//自定义方式，元素消失后删除元素
el.puff({
  easing: 'easeOut',
  duration: .5,
  remove: true,
  useDisplay: false
})
```

- switchOff：元素闪烁一下，然后往中心折叠（类似关掉电视）。当效果完成后，元素将隐藏（visibility 属性为 hidden），不过元素原来所占位置将继续存在。如果需要删除元素，请设置 remove 属性为 true。其使用方法请看下面代码。

```
var el = Ext.get('elId');
//默认方式
el.switchOff();
//自定义方式，元素消失后删除元素
el.switchOff({
  easing: 'easeOut',
  duration: .5,
  remove: true,
  useDisplay: false
});
```

- highlight：利用设置的颜色高亮显示元素，然后逐渐消隐回原有颜色。默认是设置元素背景颜色，可通过设置 attr 属性设置高亮显示方式。如果没有初始颜色，可以使用 endColor 属性设置消隐后的颜色。其使用方法请看下面代码。

```
var el = Ext.get('elId');
//默认方式
el.highlight();
//自定义方式
el.highlight('0f0f0f',{
  easing: 'easeOut',
  attr: 'color'
  duration: .5,
  endColor: 'dddddd'
});
```

- frame：从元素边界开始以水波扩散的方式提示用户。其使用方法请看下面代码。

```
var el = Ext.get('elId');
//默认方式

el.frame();
//自定义方式
el.frame('0f0f0f',{
  duration: .5,
});
```

- pause：在队列中的动画开始之前产生一个停顿。其使用方法请看下面代码。

```
el.pause(1); //停顿 1 秒
```

- fadeIn/fadeout：实现渐变效果。fadeIn 方法从透明渐变到不透明。fadeOut 方法从不透明渐变到透明。通过 endOpacity 属性可设置渐变结束后的不透明度。要注意在 IE 中可能要设置

useDisplay 属性为 true。其使用方法请看下面代码。

```
var el = Ext.get('elId');
el.fadeIn();
el.fadeOut();
//自定义方式
el.fadeIn({
  endpacity: 1,
  easing: 'easeOut',
  duration: .5
});
```

- scale：将元素的尺寸从原有尺寸过渡到设置的尺寸。其使用方法请看下面代码。

```
var el = Ext.get('elId');
el.scale(100，200);
//自定义方式
el.fadeIn(100，200，{
  easing: 'easeOut',
  duration: .5
});
```

- shift：元素渐变到新的位置、新的尺寸和不透明度。该方法至少需要位置、尺寸和不透明度其中一项设置，不然元素不会被改变。其使用方法请看下面代码。

```
var el = Ext.get('elId');
el.shift({
  width:100, //元素的新宽度
  height: 100, //元素的新高度
  x: 10 , //元素的新 x 坐标
  y: 10 , //元素的新 y 坐标
  opacity: .8， //元素的新不透明度
  easing: 'easeOut',
  duration: .5
});
 //以上属性 width、height、x、y、opacity 至少必须有一项
```

- ghost：当元素渐隐时，元素同时滑出。可通过第 1 个参数设置滑出位置。其使用方法请看下面代码。

```
var el = Ext.get('elId');
el.ghost();
//自定义方式，从左边滑出
el.ghost('l'，{
  easing: 'easeOut',
  duration: .5,
  remove:false,
  userDisplay:fase
});
```

- animate：通过该方法可自定义复杂的动画效果。其使用方法请看下面代码。

```
var el = Ext.get('elId');
el.animate(
{
    borderWith:{to : 3,from:0},
    opacity: {to: .5, from:1}
    height: {to 100 , from:el.getHeight()},
    widht: {to 200 , from:el.getwidth()},
    top:{by:-100,unit: 'px'}
```

```
    },
      1,//动画长度，单位为秒，该参数可选
      null, //回调函数，该参数可选
      'easeOut',//渐变方式，该参数可选
      'run' //动画类型
    );
```

在定义中，borderWidth、opactiy 等属性可以为元素的任何样式属性。动画类型包括以下几个类型：

- run：默认类型。
- color：动画渐变背景、文本或边界颜色。
- motion：在渐变过程中使用 Bezier 曲线作为运动轨迹。
- scroll：已垂直或水平滚动方式运动。

1.1.9 杂项

Ext.Element 还有 4 个常用的方法，请看下面的介绍。

- focus：使元素获得焦点。其使用方法请看下面代码。

```
Ext.fly('elId').focus();
```

- blur：使元素失去焦点。其使用方法请看下面代码。

```
Ext.fly('elId').blur();
```

- getValue：返回 value 属性的值。其使用方法请看下面代码。

```
var v1= Ext.fly('elId').getValue();
var v2= Ext.fly('elId').getValue(true); //返回数字值
```

- getAttributeNS：返回指定属性名称的值。其使用方法请看下面代码。

```
//返回属性 name 的值
var v1= Ext.fly('elId').getAttributeNS('', 'name');
```

1.2 Ajax 介绍

Ajax 功能是 Ext Core 的一个核心功能，其使用非常简单，主要有两种使用方式（使用不同回调方式），具体请看下面的例子。

```
//使用 success 属性和 failure 作为回调函数
Ext.Ajax.request({
  url: 'test.aspx',
  success:function(response,opts){},
  failure:function(response,opts){},
  params:{page:1}
});

//使用 callback 属性作为回调函数
Ext.Ajax.request({
  url: 'test.aspx',
  callback:function(opts,success,response){},
  params:{page:1}
});
```

两种方式的主要区别在于，callback 方式需要自己根据 success 参数判断请求是否成功，而使用 success 和 failure 方式则不需要做这一步。具体选择哪种方式主要还是根据自己喜好，而且最好是选择一种方式后，就保持这种方式，不要混合使用。

无论 success、failure 或 callback 方式，都会返回 XMLHttpRequest 对象，要访问返回的数据，需要使用该对象的 responseText 属性。如果返回的 JSON 数据需要使用 Ext 的 decode 方法解码，其使用方法请看下面的代码。

```
var datas = Ext.util.JSON.decode(response.responseText);
```

在使用 Ajax 时，通常会使用以下几个属性：

- url：要访问的地址。
- params：由 JSON 数据格式组成的提交参数。
- method：默认使用 post 方式，如果需要，可以设置为“GET”，使用 get 方式提交。
- timeout：请求时超时的时间（单位是秒），默认是 30 秒。
- form：form 元素的 id，通常在 form 提交时使用。
- disableCaching：如果设置为 true，在请求时会增加一个唯一的缓存参数，以防止返回缓存数据。

在 Ext 的 Ajax 中，有 3 个事件可以让用户处理请求过程。具体请看下面介绍。

- beforerequest：在请求发送前会触发该事件。
- requestcomplete：请求发送成功时触发该事件。
- requestexception：服务器返回 HTTP 状态错误代码时触发该事件。

1.3 DomQuery 介绍

DomQuery 是 Ext Core 提供的 HTML 或 XML 文档选择器，它支持大部分的 CSS 3 选择器规则，同时提供了一些自定义方式。

DomQuery 主要有以下 4 种选择方式：

- 通过元素标记选择，主要有以下 6 种方法：

（1）*****：选择任何元素。其使用方法请看下面代码。

```
Ext.select('*');
```

（2）**E**：元素的标记为 E。其使用方法请看下面代码。

```
Ext.select('div');
```

（3）**E F**：选择包含在标记 E 中的标记 F。其使用方法请看下面代码。

```
Ext.select('div a');//将选择 div 下的 a 元素
```

（4）**E>F**：选择包含在标记 E 中的直接子标记 F。其使用方法请看下面代码。

```
Ext.select('div>a');//将选择 div 下的直接子元素 a
```

（5）**E+F**：选择所有紧接在元素 E 后的元素 F。其使用方法请看下面代码。

```
Ext.select('div+a');//将选择紧接在 div 下的元素 a
```

（6）**E~F**：选择所有紧接在元素 E 后的同层元素 F。其使用方法请看下面代码。

```
Ext.select('div~a');//将选择紧接在 div 下的同层元素 a
```

● 通过元素属性选择，主要有以下 7 种语法。

（1）**E[foo]**：选择带有属性 foo 的元素。其使用语法请看下面代码。

```
Ext.select('div[id]');//选择有 id 属性的 div 元素
```

（2）**E[foo=bar]**：选择 foo 的属性值为 bar 的元素。其使用语法请看下面代码。

```
Ext.select('input[checked=true]');//选择 checked 属性值为 true 的元素
```

（3）**E[foo^=bar]**：选择 foo 的属性值以 bar 开头的元素。其使用语法请看下面代码。

```
Ext.select('input[name^=form1]'); //选择 name 属性值以 form1 开头的元素
```

（4）**E[foo$=bar]**：选择 foo 的属性值以 bar 结尾的元素。其使用语法请看下面代码。

```
Ext.select('input[name$=form1]'); //选择 name 属性值以 form1 结尾的元素
```

（5）**E[foo*=bar]**：选择 foo 的属性值包含字符串 bar 的元素。其使用语法请看下面代码。

```
Ext.select('input[name*=form1]'); //选择 name 属性值包含字符串 form1 的元素
```

（6）**E[foo%=2]**：选择 foo 的属性值能整除 2 的元素。其使用语法请看下面代码。

```
Ext.select('input[value%=2]'); //选择 value 属性值能整除 2 的元素
```

（7）**E[foo!=bar]**：选择 foo 的属性值不等于 bar 的元素。其使用语法请看下面代码。

```
Ext.select('input[name!=form1]'); //选择 name 属性值不等于 form1 的元素
```

● 通过伪对象选择，主要有以下 18 种语法。

（1）**E:first-child**：选择所有是父节点的第一子节点的 E。其使用语法请看下面代码。

```
Ext.select('ul li:first-child'); //选择所有 ul 下的第一个 li 子节点
```

（2）**E:last-child**：选择所有是父节点的最后一个子节点的 E。其使用语法请看下面代码。

```
Ext.select('ul li:last-child'); //选择所有 ul 下的最后一个 li 子节点
```

（3）**E:nth-child(n)**：选择所有是父节点的第 n（n≥1）个子节点的 E。其使用语法请看下面代码。

```
Ext.select('ul li:nth-child(2)'); //选择所有 ul 下的第 2 个 li 子节点
```

（4）**E:nth-child(odd)**：选择所有索引值为奇数的子节点 E。其使用语法请看下面代码。

```
Ext.select('ul li:nth-child(odd)'); //选择所有 ul 下的奇数行 li 子节点
```

（5）**E:nth-child(evan)**：选择所有索引值为偶数的子节点 E。其使用语法请看下面代码。

```
Ext.select('ul li:nth-child(evan)'); //选择所有 ul 下的偶数行 li 子节点
```

（6）**E:only-child**：选择所有父节点下只有 1 个子节点的节点 E。其使用语法请看下面代码。

```
Ext.select('ul li:only-child'); //选择所有 ul 下只有一个子节点的 li 节点
```

（7）**E:checked**：选择所有 checked 属性值为 true 的元素 E。其使用语法请看下面代码。

```
Ext.select('input:checked');
```

（8）**E:first**：选择匹配的第一个元素 E。其使用语法请看下面代码。

```
Ext.select('input:first'); //选择第一个 input 元素
```

（9）**E:last**：选择匹配的最后一个元素 E。其使用语法请看下面代码。

```
Ext.select('input:last'); //选择最后一个 input 元素
```

（10）**E:nth(n)**：选择匹配的第 n（n≥1）个元素 E。其使用语法请看下面代码。

```
Ext.select('input:nth(2)'); //选择第 2 个 input 元素
```

（11）**E:odd**：是语法“:nth-child(odd)”的简写。

（12）**E:evan**：是语法“:nth-child(evan)”的简写。

（13）**E:contains(foo)**：选择 innerHTML 属性包含字符串 foo 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:contains(list)'); //选择 innerHTML 属性包含“list”的 div
```

（14）**E:nodeValue(foo)**：选择包含一个文本节点且节点值等于 foo 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:nodeValue(test)'); //选择包含文本节点且值为“test”的 div
```

（15）**E:not(S)**：选择与简单选择器 S 不匹配的元素 E。其使用语法请看下面代码。

```
Ext.select('input:not(:checked)'); //选择不包含 checked 属性的 input
```

（16）**E:has(S)**：选择与简单选择器 S 匹配的元素 E。其使用语法请看下面代码。

```
Ext.select('div:has(p)'); //选择包含 p 的 div
```

（17）**E:next(S)**：选择与简单选择器 S 匹配的元素 E 同层的下一个元素。其使用语法请看下面代码。

```
Ext.select('div:next(p)'); //选择与包含 p 的 div 同层的下一个 div
```

（18）**E:prev(S)**：选择与简单选择器 S 匹配的元素 E 同层的上一个元素。其使用语法请看下面代码。

```
Ext.select('div:prev(p)'); //选择与包含 p 的 div 同层的下一个 div
```

● 通过 CSS 值进行选择。主要有以下 6 种语法。

（1）**E:{display=none}**：选择 display 值为 none 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display=none}');
```

（2）**E:{display^=none}**：选择 display 值以 none 开始的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display^=none}');
```

（3）**E:{display$=none}**：选择 display 值以 none 结尾的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display$=none}');
```

（4）**E:{display*=none}**：选择 display 值包含字符串 none 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display*=none}');
```

（5）**E:{display%=2}**：选择 display 值整除 2 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display%=none}');
```

（6）**E:{display!=none}**：选择 display 值不等于 none 的元素 E。其使用语法请看下面代码。

```
Ext.select('div:{display!=none}');
```

以上语法可以单独使用，也可以组合在一起使用，例如以下代码：

```
Ext.select('div,span'); //选择所有 div 元素与 span 元素
//选择 class 为 red，且是第 1 个子节点，display 属性为 none 的 div
Ext.select('div.red:first-child[display=none]');
```

如果没有指定选择器开始搜索的根节点，默认是从 body 节点开始，这就等于每次都要做全文搜索，其性能可想而知，是相当低效的，因此，建议大家在每次搜索时都指定根节点，减少搜索范围。设置搜索根节点的语法请看下面代码。

```
Ext.select('div',true, 'elId'); // elId 为根节点 id
```

```
    //下面这句与上面的效果是一样的
    Ext.fly('elid').select('div');
```