先看一个代码A：  
Ext.define('MyApp.view.MyWindow', {  
    extend: 'Ext.window.Window',  
    height: 451,  
    width: 642,  
    title: 'My Window',  
  
  
    initComponent: function() {  
        var me = this;  
        var button=Ext.create('Ext.button.Button',{id:'button'});  
        var obj={};   
        me.callParent(arguments);  
    }  
  
  
});  
以上代码声明了一个window类，在这个类的initComponent方法中声明了一个局部变量，引用了一个button对象。initComponent方法我们可以类比理解成一个类的构造方法。  
那么假使我现在create出'MyApp.view.MyWindow'这个类，则在内存中可以检测出button这个对象也是存在的。检测方法可以是 在firebug的控制台中输入代码alert(Ext.getCmp('button'));结果是 object Object。当然我们alert(obj)的话，结果肯定是undifined。  
那么当我关闭这个window，因为window的默认closeAction是destroy，因此会从内存中销毁这个window。但是很不幸，我在window的initComponent()中声明的那个button对象并不会销毁，那么这就导致了内存泄露。  
如果你用的是ext-all.js，那么虽然你内存泄露了，但是当你第二次打开这个window的话，并不会导致程序出错。如果你用的是ext-all-dev.js，那么第二次打开这个window就会报错，大概是duplicated id之类的错误。  
通常在开发阶段，用的是ext-all-dev.js。在商业部署时才会用ext-all.js。  
  
  
下面再来看一个代码B，这个代码是不会导致内存泄露的。  
Ext.define('MyApp.view.MyWindow', {  
    extend: 'Ext.window.Window',  
  
  
    height: 451,  
    width: 642,  
    title: 'My Panel',  
  
  
    initComponent: function() {  
        var me = this;  
  
  
        Ext.applyIf(me, {  
            items: [  
                {  
                    xtype: 'button',  
                    id: 'button',  
                    text: 'MyButton'  
                }  
            ]  
        });  
  
  
        me.callParent(arguments);  
    }  
  
  
});  
  
  
那么当这个window被关闭时，alert(Ext.getCmp('button'));结果是 undefined，意味着它也被销毁了。  
  
  
我们也可以这么写代码C：  
Ext.define('MyApp.view.MyWindow', {  
    extend: 'Ext.window.Window',  
  
  
    autoShow: true,  
    height: 250,  
    width: 400,  
    title: 'My Panel',  
  
  
    initComponent: function() {  
        var me = this;  
        var button=Ext.create('Ext.button.Button',{id:'button',text:'button'});  
        var obj={};   
         
        me.callParent(arguments);  
         me.add(button);  
    }  
  
  
});  
则当window关闭以后，执行alert(Ext.getCmp('button'));结果依然是undefined。  
  
  
那么如果我们非要写出A那种代码，该怎么去销毁button对象？请看代码D：  
Ext.define('MyApp.view.MyWindow', {  
    extend: 'Ext.window.Window',  
  
  
    autoShow: true,  
    height: 250,  
    width: 400,  
    title: 'My Panel',  
  
  
    initComponent: function() {  
        var me = this;  
var button=Ext.create('Ext.button.Button',{id:'button'});  
        Ext.applyIf(me, {  
             
            listeners: {  
                close: {  
                    fn: me.onWindowClose,  
                    scope: me  
                }  
            }  
        });  
  
  
        me.callParent(arguments);  
    },  
  
  
    onWindowClose: function(panel, eOpts) {  
        Ext.ComponentManager.unregister(Ext.getCmp('button'));  
    }  
  
  
});  
关掉window之后执行alert(Ext.getCmp('button'));显示undefined。  
  
  
相信看到这里，要是extjs还算熟悉的话，应该知道问题的关键在哪了，其实就是Ext.ComponentManager这个类在作怪。因为我声明button时指定了id，所以Ext.ComponentManager会在自身的hashmap中托管这个button。而由于代码A中button并不依附于任何视图组件而单独存在，它只是在某个视图组件的initComponent()中被创建而已，所以它事实上成为了全局的东西，虽然我们主观上会觉得这个button对象应该要被销毁才合理，因为一眼看去它并不是被全局变量所引用，那么它理所当然的应该被javascript的垃圾回收机制来释放掉内存，不过暗地里它并不是一个孤立的对象，它被Ext.ComponentManager引用了，因此它不会被垃圾回收。  
  
  
那么假使我不指定id这个config呢？我猜想那么当initComponent()结束后它就会被垃圾回收掉了。不过我暂时还不知道怎么去验证。  
  
  
也许有人会说，为什么要写A那种代码，意义何在？不过在实际开发中由于各种各样的原因，有很大可能会存在类似A那样的代码，比如说右击事件菜单这种情况。  
  
  
结论：没事不要闲的蛋疼去配置id，万一配置了id而且又是有内存泄露的风险，则记得在使用完了后从Ext.ComponentManager中销毁它。store的话也有一个store的管理器，我就没具体去试了。也许大概差不多也是一样的情况。还有就是并不是只在4.2才会这样，4.1也是一样的，4.0以前的就没有去尝试了