
Exe 入门教程

目录

EXE 入门教程	1
1 开始 EXTJS	2
1.1 获取 EXTJS	2
1.2 应用 EXTJS	2
1.3 演示 HELLOWORD	3
2 组件的介绍和应用	4
2.1 介绍.....	4
2.2 应用.....	6
2.2.1 简单示例	6
2.3 组件详解.....	8
2.3.1 EXT. Panel	8
2.3.2 布局控件	10
2.3.3 表格控件	14
2.3.4 树形控件	16
2.3.5 表单控件 Form.....	20
2.3.6 弹出窗口	23
3 EXT 核心	29
3.1 EXT.ELEMENT	29
3.2 CSS 样式操作	30
3.1 DOM 操作.....	31
3.1 AJAX 介绍	35

1 开始 extjs

1.1 获取 extjs

要使用 ExtJS，那么首先要得到 ExtJS 库文件，该框架是一个开源的，可以直接从官方网站下载，网址 <http://extjs.com/download>，把下载得到的 ZIP 压缩文件解压可以看到如下目录和文件

名称	大小	类型	修改日期
adapter		文件夹	2010-4-27 11:00
docs		文件夹	2010-4-27 11:12
examples		文件夹	2010-4-27 11:00
pkgs		文件夹	2010-4-27 11:00
resources		文件夹	2010-4-27 11:00
src		文件夹	2010-4-27 11:00
test		文件夹	2010-4-27 11:00
welcome		文件夹	2010-4-27 11:00
ext.jsb2	44 KB	JSB2 文件	2010-4-27 11:00
ext-all.js	662 KB	JScript Script 文件	2010-4-27 11:00
ext-all-debug.js	1,237 KB	JScript Script 文件	2010-4-27 11:00
ext-all-debug-w-comments.js	2,612 KB	JScript Script 文件	2010-4-27 11:00
gpl-3.0.txt	35 KB	文本文档	2010-4-27 11:00
INCLUDE_ORDER.txt	1 KB	文本文档	2010-4-27 11:00
index.html	9 KB	HTML File	2010-4-27 11:00
license.txt	2 KB	文本文档	2010-4-27 11:00

1.2 应用 extjs

应用 extjs 需要在页面中引入 extjs 的样式及 extjs 库文件，样式文件为 resources/css/ext-all.css，extjs 的 js 库文件主要包含两个，adapter/ext/ext-base.js 及 ext-all.js，其中 ext-base.js 表示框架基础库，ext-all.js 是 extjs 的核心库。

因此，要使用 ExtJS 框架的页面中一般包括下面几句：

```
<link rel="stylesheet" type="text/css"
href="extjs/resources/css/ext-all.css" />
<script type="text/javascript"
src="extjs/adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="extjs/ext-all.js"></script>
```

在 ExtJS 库文件及页面内容加载完后，ExtJS 会执行 Ext.onReady 中指定的函数，因此可以用，一般情况下每一个用户的 ExtJS 应用都是从 Ext.onReady 开始的，使用 ExtJS 应用程序的代码大致如下：

```
<script>
function fn()
{
alert('ExtJS库已加');
}
```

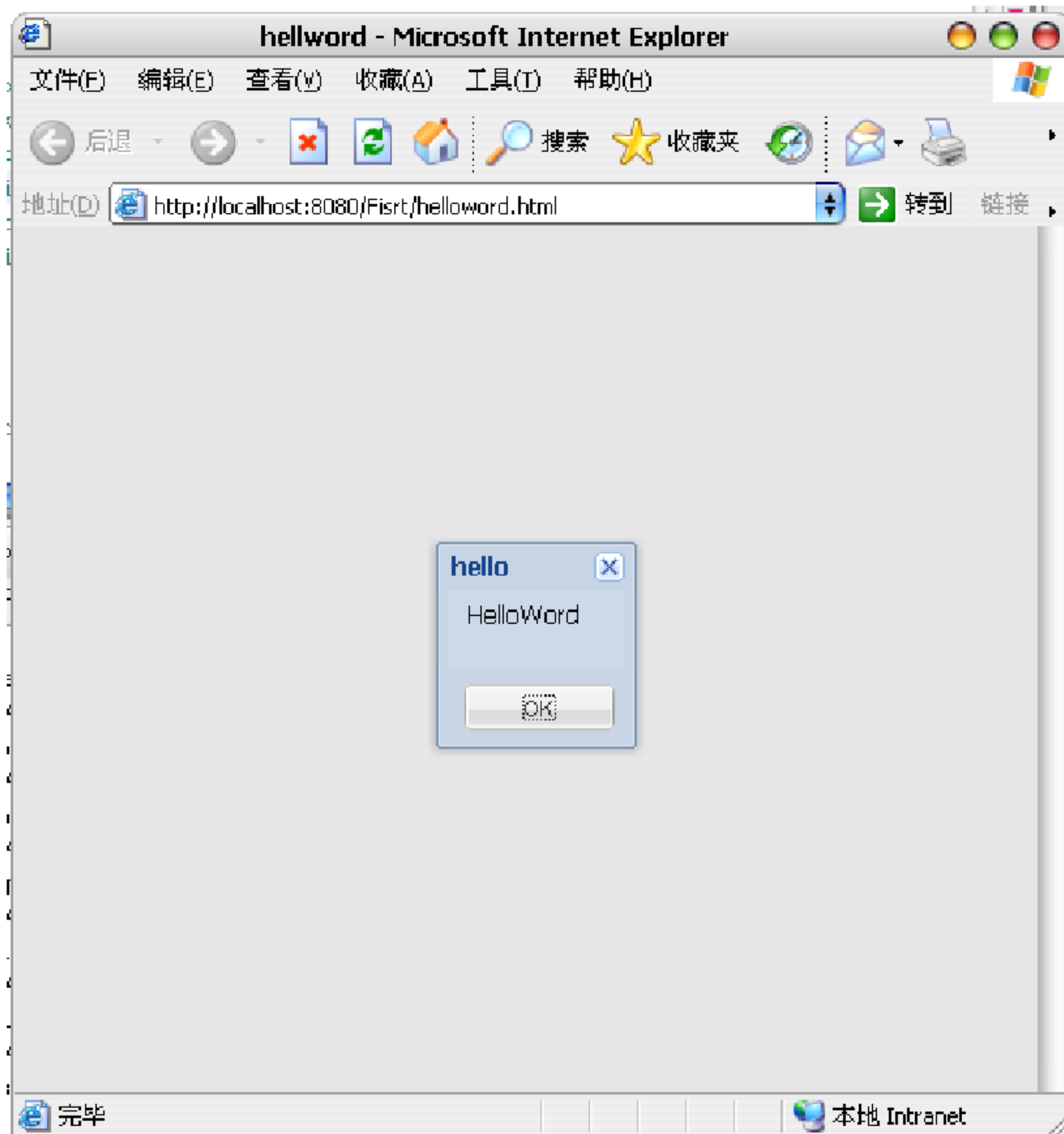
```
}  
Ext.onReady(fn);  
</script>
```

fn 也可以写成一个匿名函数的形式，因此上面的代码可以改成下面的形式：

```
<script>  
function fn()  
{  
alert('ExtJS库已加载!');  
}  
Ext.onReady(function ()  
{  
alert('ExtJS库已加载!');  
}  
);  
</script>
```

1.3 演示 HelloWorld

```
<html xmlns="http://www.w3.org/1999/xhtml" >  
  <head>  
    <meta http-equiv="Content-Type" content="text/html;  
charset=utf-8" />  
    <title>ExtJS</title>  
    <link rel="stylesheet" type="text/css"  
href="extjs/resources/css/ext-all.css" />  
    <script type="text/javascript"  
src="extjs/adaptor/ext/ext-base.js"></script>  
    <script type="text/javascript" src="extjs/ext-all.js"></script>  
    <script>  
Ext.onReady(function()  
{  
Ext.MessageBox.alert("hello", "HelloWord");  
});  
</script>  
</head>  
<body>  
</body>  
</html>
```



2 组件的介绍和应用

2.1 介绍

组件大致可以分成三大类，即基本组件、工具栏组件、表单及元素组件。
基本组件有：

xtype	Class
box	Ext. BoxComponent 具有边框属性的组件
button	Ext. Button 按钮
colorpalette	Ext. ColorPalette 调色板

component	Ext. Component 组件
container	Ext. Container 容器
cycle	Ext. CycleButton
dataview	Ext. DataView 数据显示视图
datepicker	Ext. DatePicker 日期选择面板
editor	Ext. Editor 编辑器
editorgrid	Ext. grid.EditorGridPanel 可编辑的表格
grid	Ext. grid.GridPanel 表格
paging	Ext. PagingToolbar 工具栏中的间隔
panel	Ext. Panel 面板
progress	Ext. ProgressBar 进度条
splitbutton	Ext. SplitButton 可分裂的按钮
tabpanel	Ext. TabPanel 选项面板
treepanel	Ext. tree.TreePanel 树
viewport	Ext. ViewPort 视图
window	Ext. Window 窗口

工具栏组件有

toolbar	Ext. Toolbar 工具栏
tbutton	Ext. Toolbar.Button 按钮
tbfill	Ext. Toolbar.Fill 文件
tbitem	Ext. Toolbar.Item 工具条项目
tbseparator	Ext. Toolbar.Separator 工具栏分隔符
tbspacer	Ext. Toolbar.Spacer 工具栏空白
tbsplit	Ext. Toolbar.SplitButton 工具栏分隔按钮
tbtext	Ext. Toolbar.TextItem 工具栏文本项

表单及字段组件包含:

form	Ext. FormPanel Form 面板
checkbox	Ext. form.Checkbox checkbox 录入框
combo	Ext. form.ComboBox combo 选择项
datefield	Ext. form.DateField 日期选择项
field	Ext. form.Field 表单字段
fieldset	Ext. form.FieldSet 表单字段组
hidden	Ext. form.Hidden 表单隐藏域
htmleditor	Ext. form.HtmlEditor html 编辑器
numberfield	Ext. form.NumberField 数字编辑器
radio	Ext. form.Radio 单选按钮
textarea	Ext. form.TextArea 区域文本框
textfield	Ext. form.TextField 表单文本框
timefield	Ext. form.TimeField 时间录入项
trigger	Ext. form.TriggerField 触发录入项

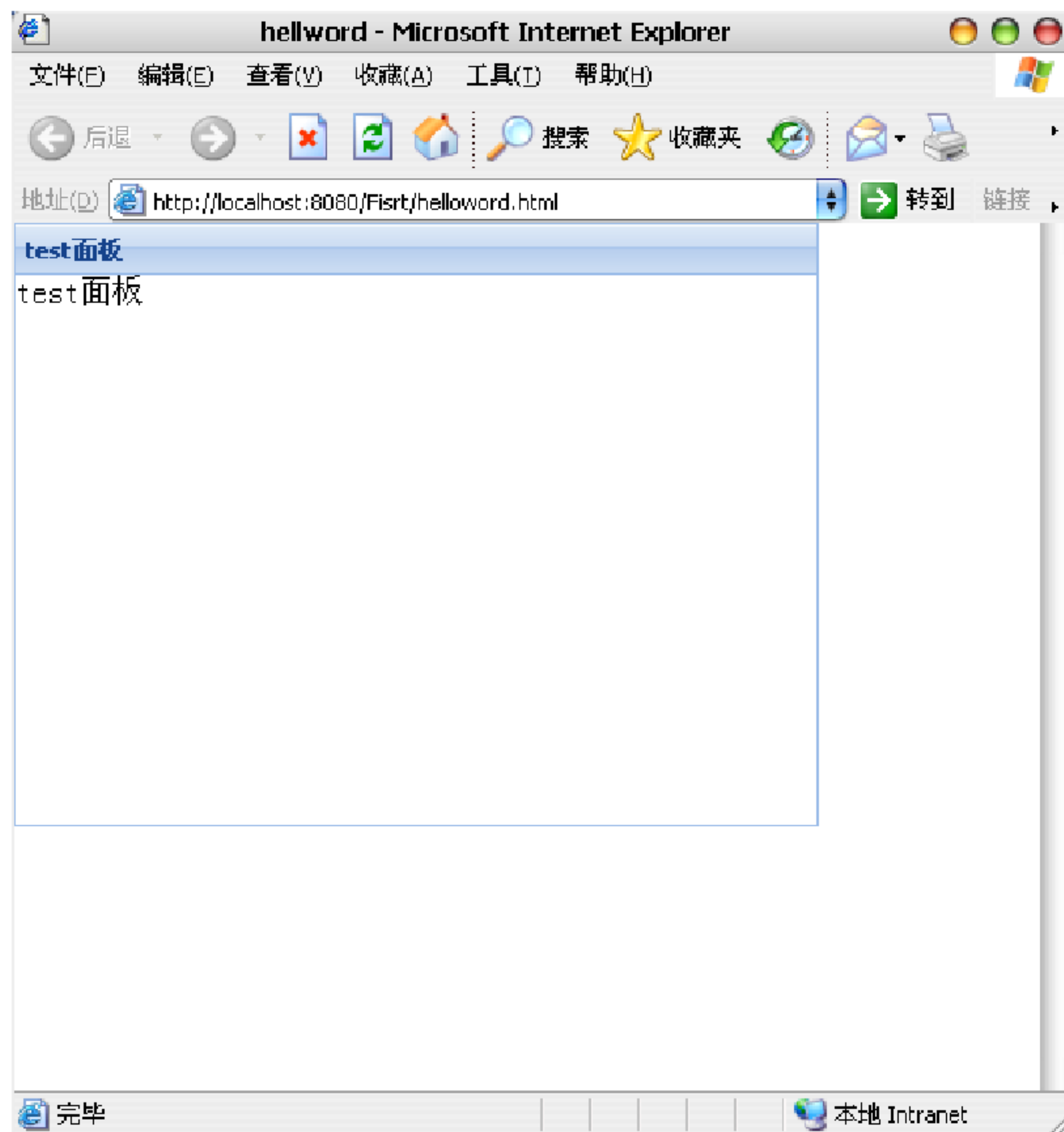
2.2 应用

2.2.1 简单示例

首先我们做一个简单的 panel

```
Ext.onReady(function()  
{  
    var myPanel=new Ext.Panel({  
        renderTo:Ext.getBody(),  
        title:'test面板',  
        width:400,  
        height:300,  
        html:'test面板'  
    });  
});
```

效果如下图



接下来我们创建一个tabpanel

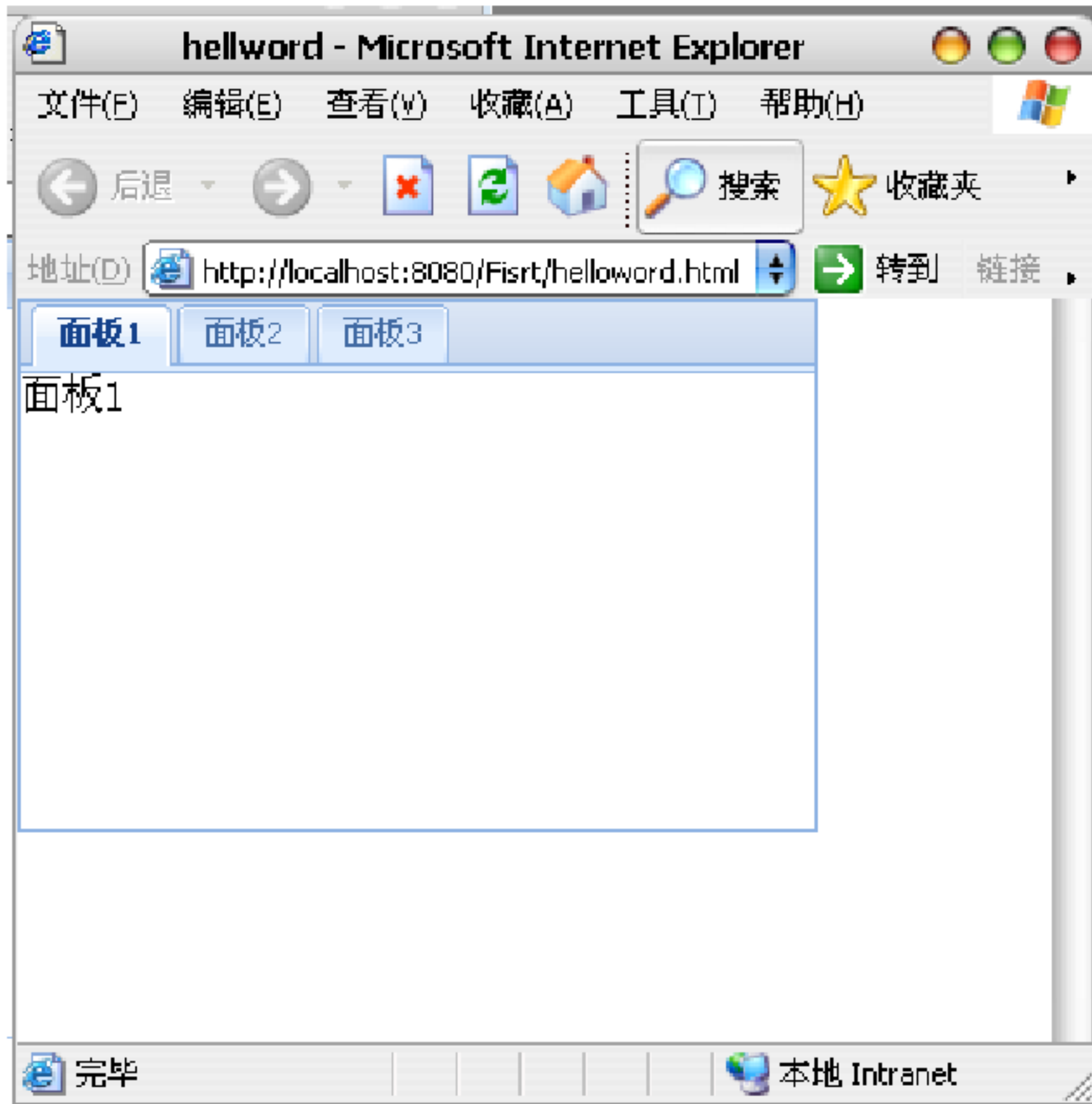
```
Ext.onReady(function() {  
    var panel = new Ext.TabPanel({  
        renderTo : Ext.getBody(),  
        width : 300,  
        height : 200,  
        activeTab:0,  
        items : [new Ext.Panel({  
            title : "面板1",  
            height : 30,  
            html:'面板1'  
        }), new Ext.Panel({  
            title : "面板2",  
            height : 30,  
            html:'面板2'  
        }), new Ext.Panel({
```

```

        title : "面板3",
        height : 30,
        html: '面板1'
    } ) ]
    } ) ;
} ) ;

```

效果如下图



2.3 组件详解

2.3.1 EXT. Panel

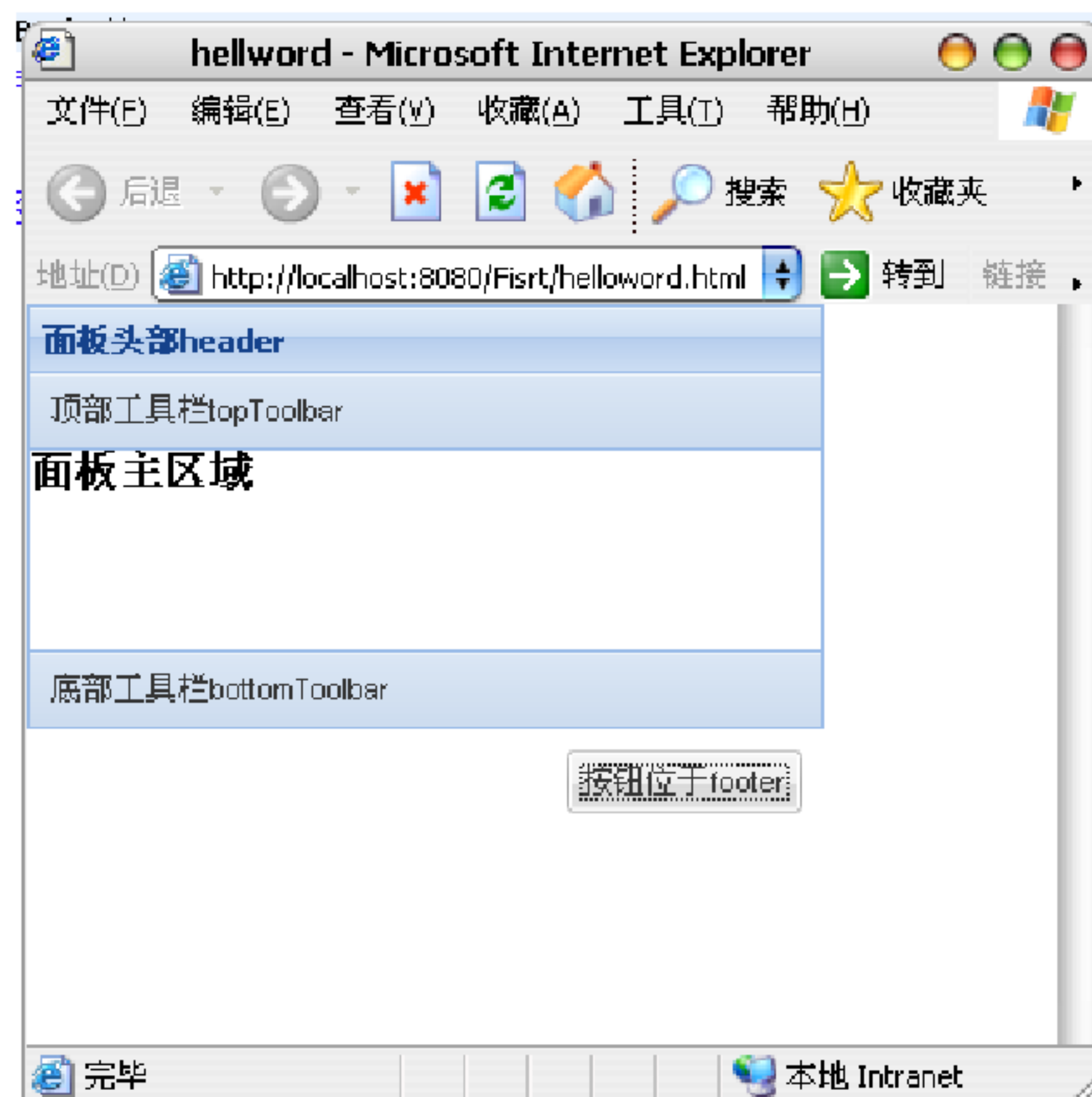
Panel 是 extjs 中最常用的一款组件，也是开发中用的最多的一款组件。最常见的 UI 组件都是继承自它,如果没有 panel，extjs 可能就没那么盛名咯

面板 Panel 是 ExtJS 控件的基础，很高级控件都是在面板的基础上扩展的，还有其它大多数控件也都直接或间接有关系。应用程序的界面一般情况下是由一个一个的面板通过不同组织方式形成。面板由以下几个部分组成，一个顶部工具栏、一个底部工具栏、面板头部、面板尾部、面板主区域几个部分组件。面板类中还内置了面板展开、关闭等功能，并提供一系列可重用的工具按钮使得我们可以轻松实现自定义的行为，面板可以放入其它任何容器中，面板本身是一个容器，他里面又可以包含各种其它组件。

面板的类名为 Ext.Panel，其 xtype 为 panel，下面的代码可以显示出面板的各个组

成部分:

```
Ext.onReady(function () {
    new Ext.Panel({
        renderTo : Ext.getBody(),
        title : "面板头部header",
        width : 300,
        height : 200,
        html : '<h1>面板主区域</h1>',
        tbar : [{
            text : '顶部工具栏topToolbar'
        }],
        bbar : [{
            text : '底部工具栏bottomToolbar'
        }],
        buttons : [{
            text : "按钮位于footer"
        }]
    });
});
```

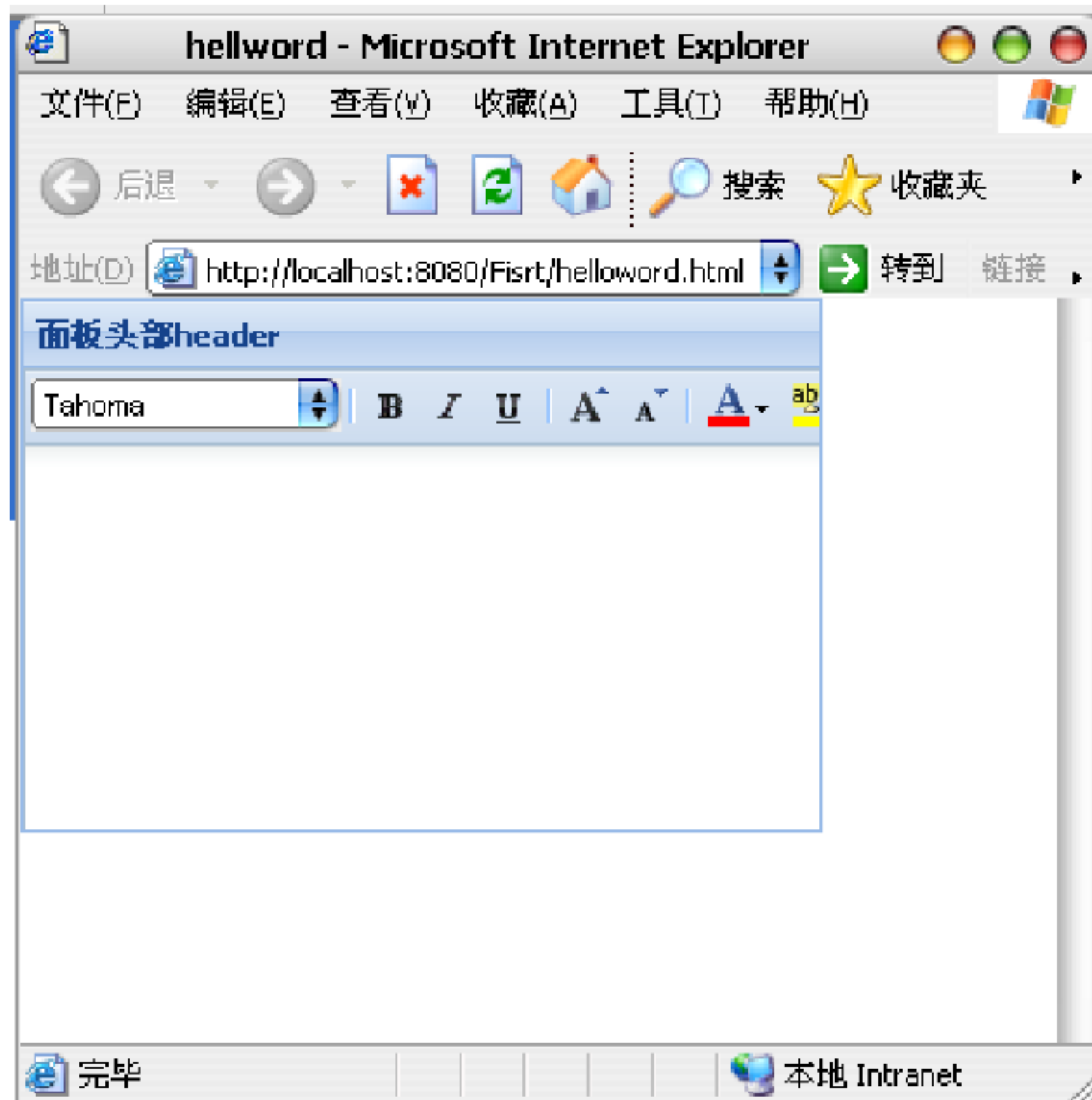


在 panel 中有个很重要的属性 Items 比较重要, 设置它的值就是给 panel 添加子项
如下代码

```
Ext.onReady(function () {
```

```
new Ext.Panel({
    renderTo : Ext.getBody(),
    title : "面板头部header",
    width : 300,
    height : 200,
    items:[{
        xtype:'htmleditor'
    }]
});
```

显示效果如下图



同时 panel 如同一张白纸任我们再它上面进行任何布局，添加任何组件

2.3.2 布局控件

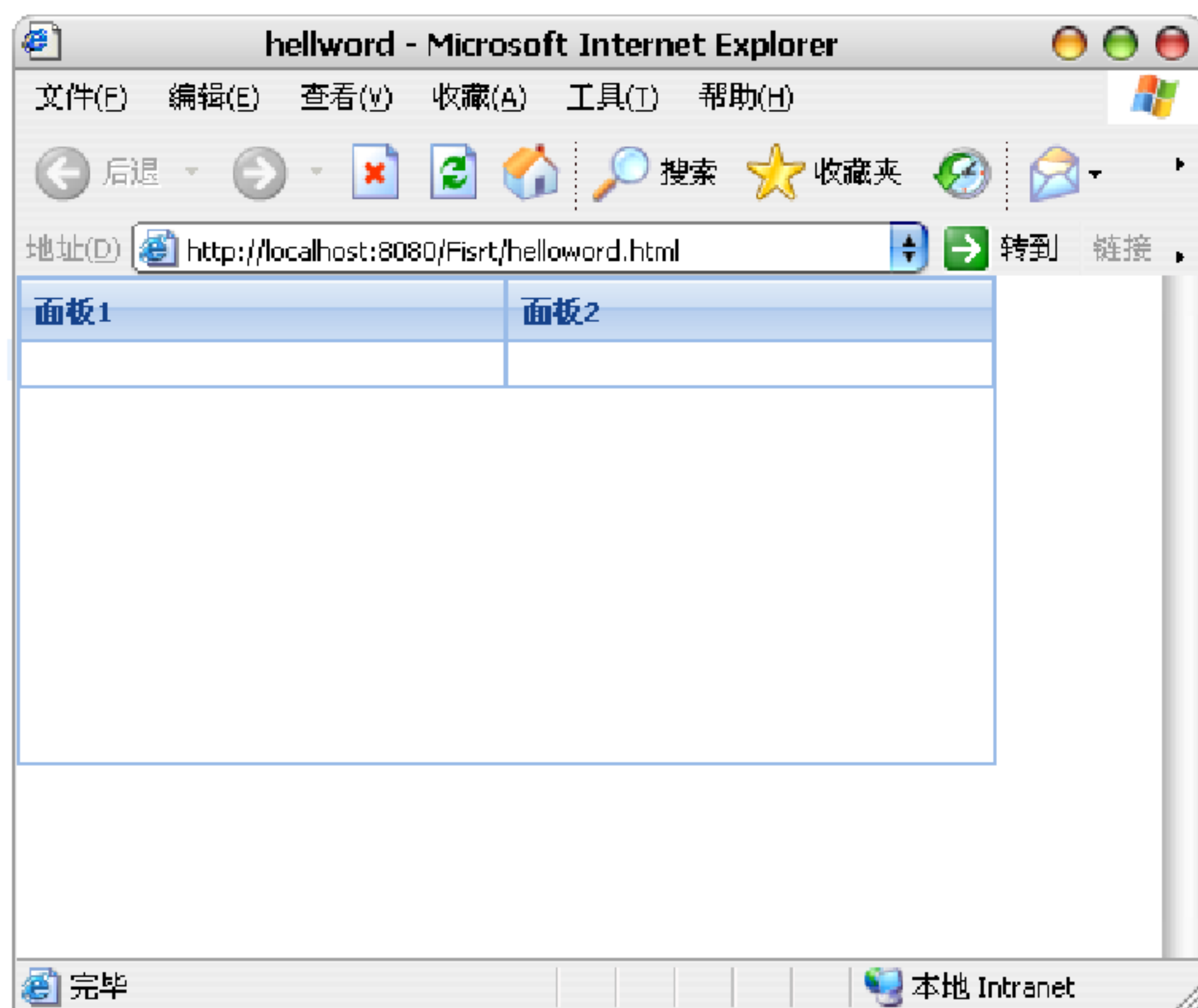
所谓布局就是指容器组件中子元素的分布、排列组合方式。Ext 的所有容器组件都支持布局操作，每一个容器都会有一个对应的布局，布局负责管理容器组件中子元素的排列、组合及渲染方式等。

ExtJS 的布局基类为 `Ext.layout.ContainerLayout`，其它布局都是继承该类。ExtJS 的容器组件包含一个 `layout` 及 `layoutConfig` 配置属性，这两个属性用来指定容器使用的布局及布局的详细配置信息，如果没有指定容器组件的 `layout` 则默认会使

用 ContainerLayout 作为布局，该布局只是简单的把元素放到容器中，有的布局需要 layoutConfig 配置，有的则不需要 layoutConfig 配置。看代码：

下面是列的布局

```
Ext.onReady(function() {  
    new Ext.Panel({  
        renderTo:Ext.getBody(),  
        width:400,  
        height:200,  
        layout:"column",  
        items:[{columnWidth:.5,  
            title:"面板1"},  
            {columnWidth:.5,  
            title:"面板2"}]  
    });  
});
```



Border布局由类Ext.layout.BorderLayout定义，布局名称为border。该布局把容器分成东南西北中五个区域，分别由east, south, west, north, center来表示，在往容器中添加子元素的时候，我们只需要指定这些子元素所在的位置，Border布局会自动把子元素放到布局指定的位置。看下面的代码：

```
Ext.onReady(function() {
```

```
var viewport = new Ext.Viewport({
    layout : 'border',
    renderTo:Ext.getBody(),
    items : [{
        title : 'north',
        region : 'north',
        split : true,
        border : true,
        collapsible : true,
        height : 100,
        minSize : 100,
        maxSize : 120
    }, {
        title : 'south',
        region : 'south',
        split : true,
        border : true,
        collapsible : true,
        height : 100,
        minSize : 100,
        maxSize : 120
    }, {
        title : 'east',
        region : 'east',
        split : true,
        border : true,
        collapsible : true,
        width : 120,
        minSize : 120,
        maxSize : 200
    }, {
        title : 'west',
        region : 'west',
        split : true,
        border : true,
        collapsible : true,
        width : 120,
        minSize : 120,
        maxSize : 200
    }, {
        title : 'center',
        region : 'center',
        split : true,
        border : true,
```



```

        collapsible : true
    }
});
});

```

效果如图

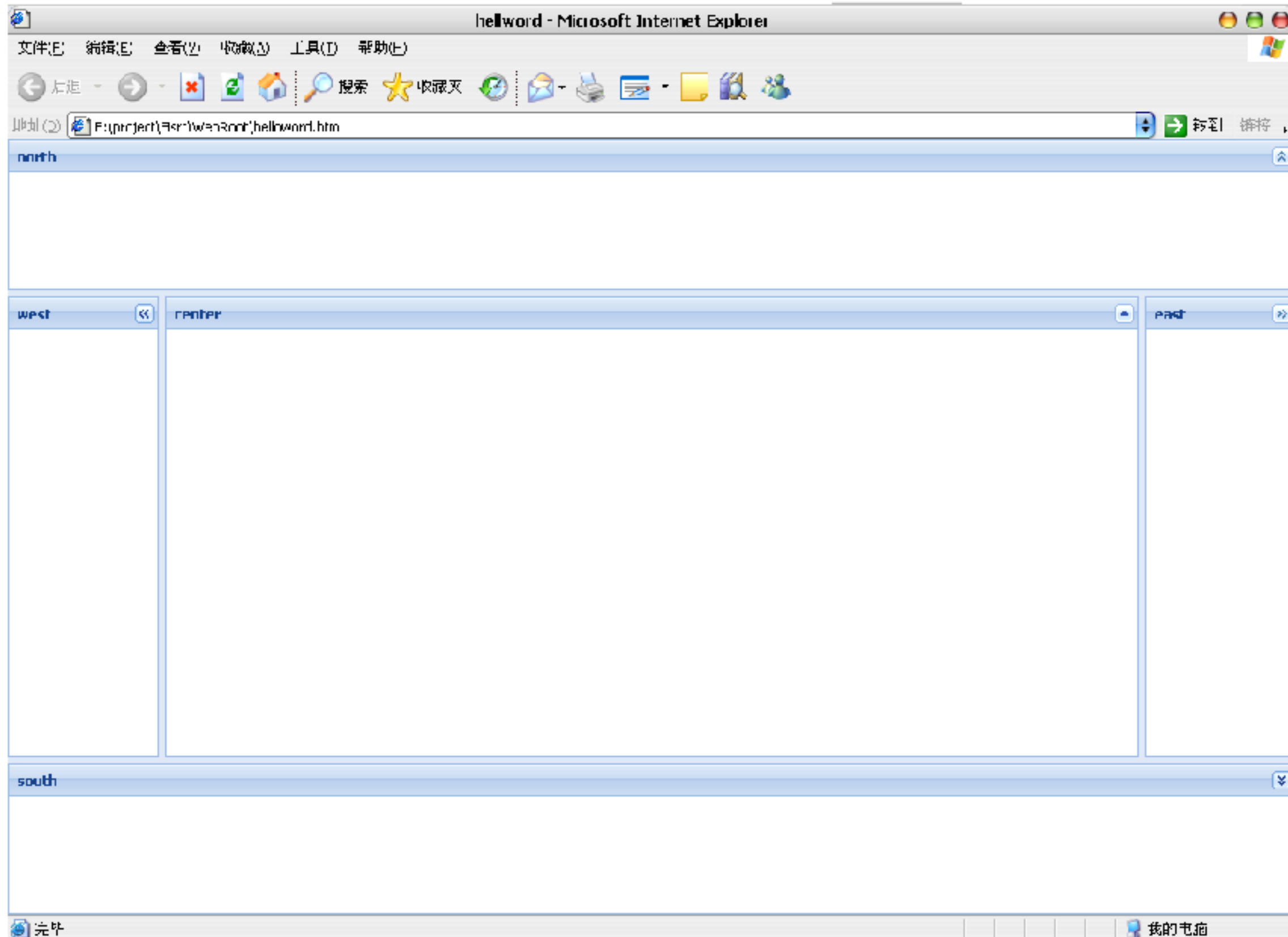


Table 布局由类 Ext.layout.TableLayout 定义，名称为 table，该布局负责把容器中的子元素按照类似普通 html 标签，看代码：

```

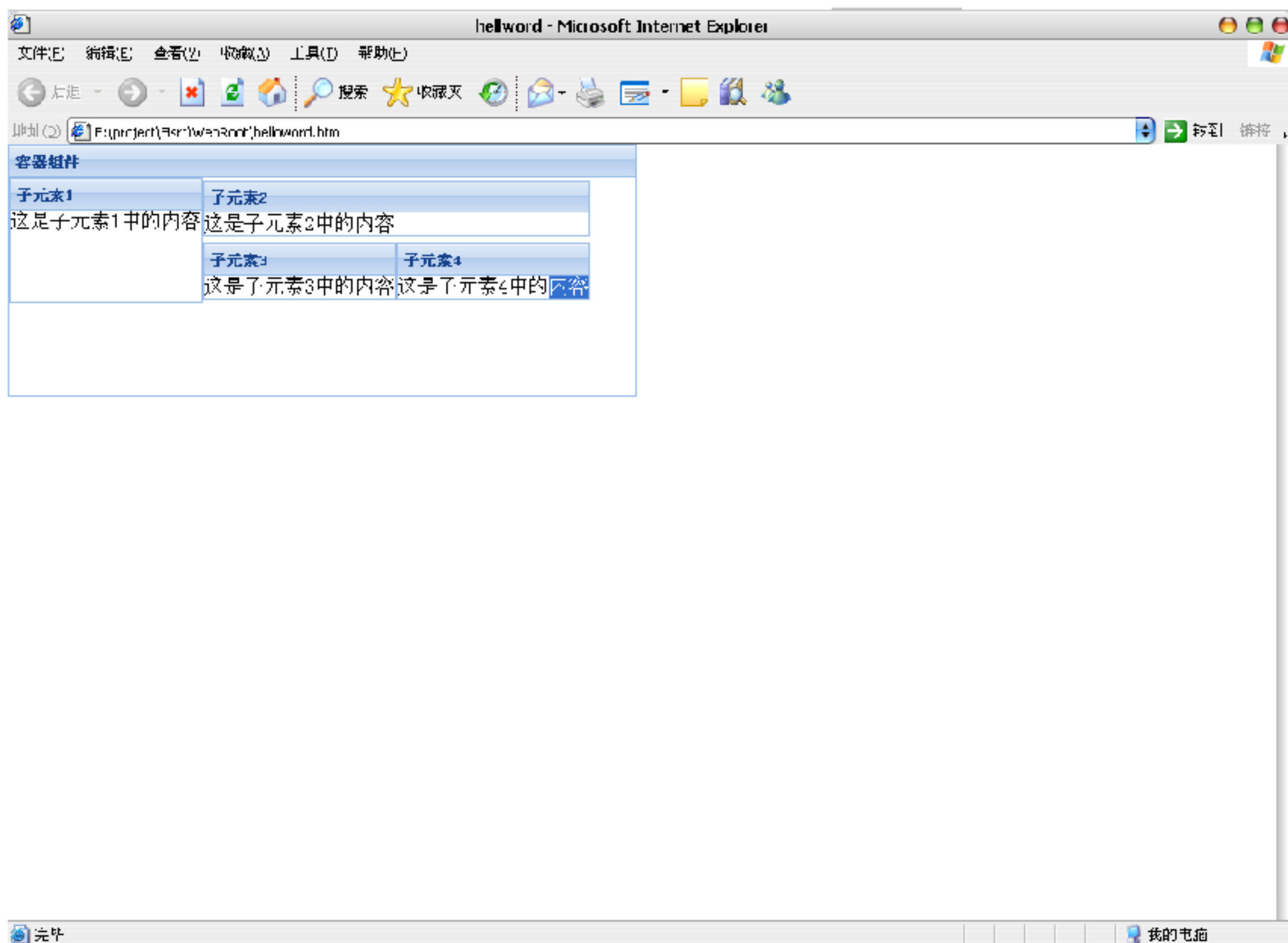
Ext.onReady(function() {
    var panel = new Ext.Panel({
        renderTo : Ext.getBody() ,
        title : "容器组件",
        width : 500,
        height : 200,
        layout : "table",
        layoutConfig : {
            columns : 3
        },
        items : [{
            title : "子元素1",
            html : "这是子元素1中的内容",
            rowspan : 2,
            height : 100
        }, {
            title : "子元素2",

```

```

        html : "这是子元素2中的内容",
        colspan : 2
    }, {
        title : "子元素3",
        html : "这是子元素3中的内容"
    }, {
        title : "子元素4",
        html : "这是子元素4中的内容"
    }
]
    );
});

```

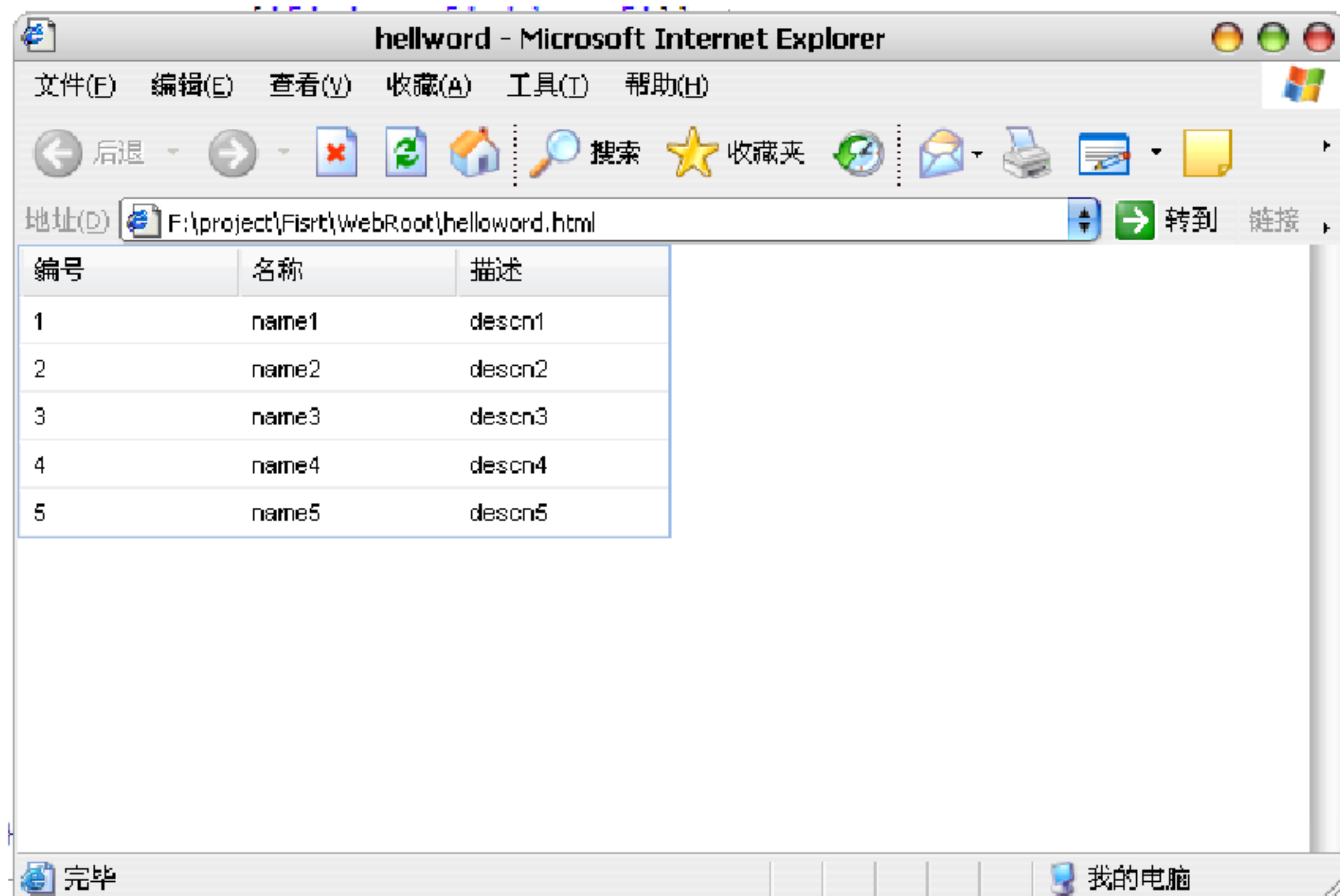


Ext 中除了上述讲的这几种布局以外还有 Fit 布局、Form 布局、Accordion 布局、Card 布局，这里就不一一描述，请查看相关资料吧

2.3.3 表格控件

ExtJS 中的表格功能非常强大，包括了排序、缓存、拖动、隐藏某一列、自动显示行号、列汇总、单元格编辑等实用功能。表格由类 Ext.grid.GridPanel 定义，继承自 Panel，其 xtype 为 grid。ExtJS 中，表格 Grid 必须包含列定义信息，并指定表格的数据存储器 Store。表格的列信息由类 Ext.grid.ColumnModel 定义、而表格的数据存储器由 Ext.data.Store 定义，数据存储器根据解析的数据不同分为 JsonStore、SimpleStore、GroupingStore 等。废话不多说了，直接看代码：

```
Ext.onReady(function() {
    //定义列
    var cm = new Ext.grid.ColumnModel([
        {header: '编号', dataIndex: 'id'},
        {header: '名称', dataIndex: 'name'},
        {header: '描述', dataIndex: 'descn'}
    ]);
    //数据定义
    var data = [['1', 'name1', 'descn1'],
        ['2', 'name2', 'descn2'],
        ['3', 'name3', 'descn3'],
        ['4', 'name4', 'descn4'],
        ['5', 'name5', 'descn5']];
    //数据源定义
    var ds = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
    ds.load();
    var grid = new Ext.grid.GridPanel({
        renderTo: Ext.getBody(),
        ds: ds,
        cm: cm,
        width: 300,
        autoHeight: true
    });
});
```

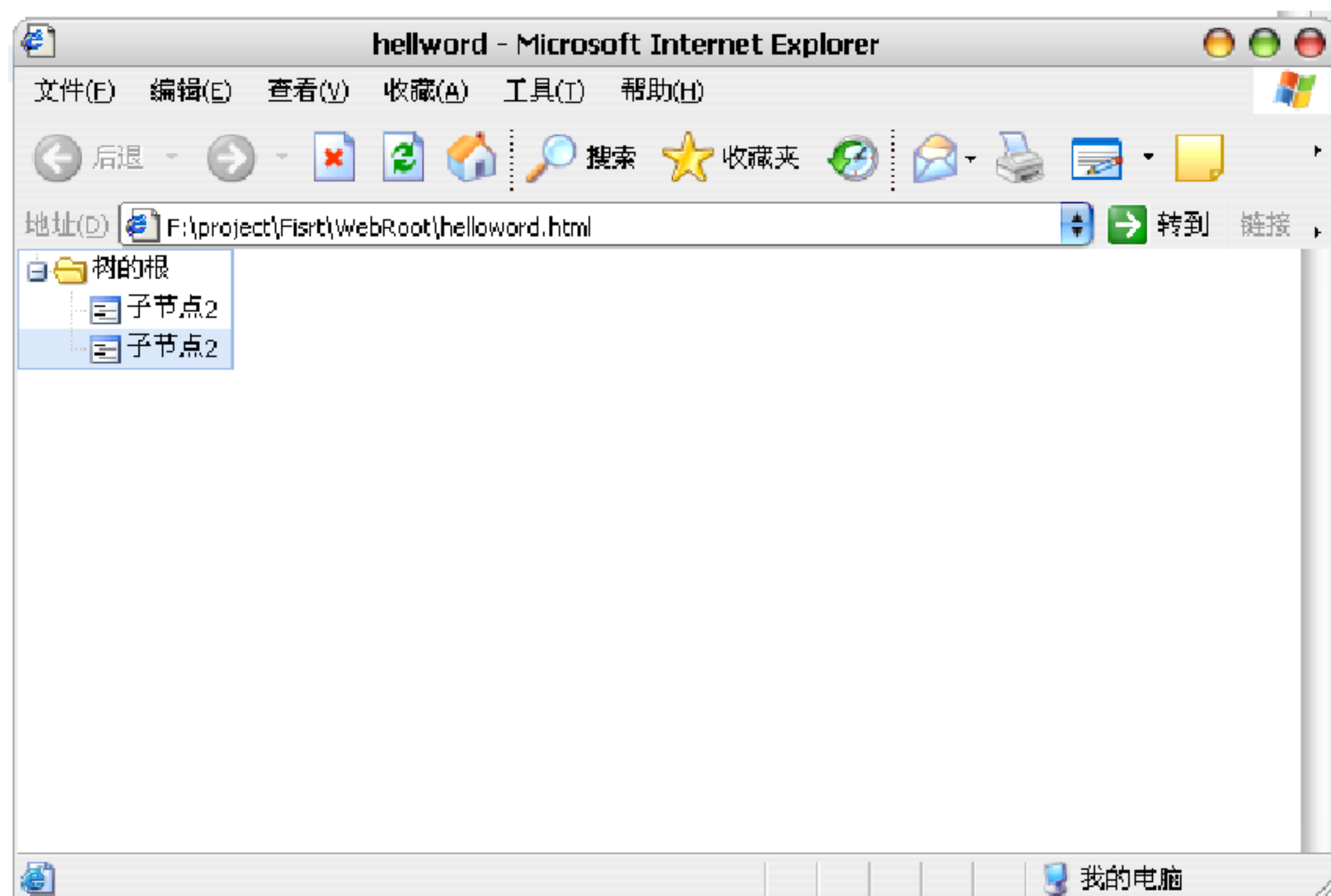


这里只做了些简单的示例演示，如果想详细了解表格控件请查阅 EXT 相关资料和书籍

2.3.4 树形控件

树控件由 Ext.tree.TreePanel 类定义，控件的名称为 treepanel，TreePanel 类继承自 Panel 面板。在 ExtJS 中使用树控件其实非常简单，我们先来看下面的代码
一个简单的树

```
Ext.onReady(function() {
    var root=new Ext.tree.TreeNode({
        id:"root",
        text:"树的根"});
    root.appendChild(new Ext.tree.TreeNode({
        id:"c1",
        text:"子节点2"
    }));
    root.appendChild(new Ext.tree.TreeNode({
        id:"c2",
        text:"子节点2"
    }));
    var tree=new Ext.tree.TreePanel({
        renderTo:Ext.getBody(),
        root:root,
        width:100
    });
});
```

树的节点信息。ExtJS 的树控件提供了对这种功能的支持，你只需要在创建树控件的时候，通过给树指定一个节点加载器，可以用来从服务器端动态加载树的节点信息。我们来看下面的代码：

```
Ext.onReady(function() {  
    Ext.BLANK_IMAGE_URL='images/default/s.gif';  
    var root = new Ext.tree.AsyncTreeNode({  
        expanded:true,  
        text:'系统菜单',  
        id:"101",//默认为0  
        draggable : false,  
        iconCls:"b"  
    });  
    var tree = new Ext.tree.TreePanel({  
        title:'树形结构',  
        root : root,  
        border:false,  
        autoHeight:true,  
        renderTo : Ext.getBody(),  
        autoScroll : true,//显示滚动条  
        animate : true,//true表示使用动画展开/折叠  
        enableDD : true,//允许拖放  
        containerScroll : true,//登记本容器ScrollManager  
        listeners : {//在加载之前的一个监听事件  
            'beforeload' : function(node) {  
                //alert(node.id);//得到的节点
```

```

        node.loader = new Ext.tree.TreeLoader({ //树节点的数据
来源
            url : "viewtree.do?id="+node.id, //请求的路径
            baseParams : { //请求参数
                //id : node.id
            }
        });
    }
}
});
});
});

```

```

-----后台 action 代码-----
package action;
import java.io.IOException;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;
import util.ExtHelper;
import dao.UserDAO;

public class TreeAction extends DispatchAction {
    public ActionForward getTreeDate(ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request, HttpServletResponse
        response) throws IOException {
        response.setContentType("charset=UTF-8");
        String parentId=request.getParameter("id");
        UserDAO userDao=new UserDAO();
        List lst=userDao.getTree(parentId);
        String json=ExtHelper.listToJsonArray(lst);
        System.out.println(json);
        response.getWriter().write(json);
        return null;
    }
}

```

这是把 Bean 转换成 JSON 和 XML 的帮助类，详细内容就不做解释了，看代码就能明白一切，此类需要加载的 jar 包
commons-beanutils.jar

```
commons-lang.jar
ezmorph-1.0.4.jar
xstream-1.3.1.jar
commons-logging.jar
commons-collections-3.2.jar
json-lib-2.2.3-jdk13.jar
```

```
package util;
import java.util.ArrayList;
import java.util.List;
import net.sf.json.JSONArray;
import net.sf.json.JSONObject;
import bean.TotalJson;
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

public class ExtHelper {
    //把list数据转换成json格式的数据
    public static String getJsonFromList(long recordTotal, List
beanList){
        TotalJson total=new TotalJson();
        total.setResults(recordTotal);
        total.setItems(beanList);
        JSONObject jsonArray=JSONObject.fromObject(total);
        return jsonArray.toString();
    }
    //把list数据转换成xml格式的数据
    public static String getXmlFromList(long recordTotal, List
beanList){
        TotalJson total=new TotalJson();
        total.setResults(recordTotal);
        List results=new ArrayList();
        results.add(total);
        results.addAll(beanList);
        XStream xs=new XStream(new DomDriver());
        for (int i = 0; i <results.size(); i++) {
            Class c=results.get(i).getClass();
            String b=c.getName();
            String[] temp=b.split("\\.");
            xs.alias(temp[temp.length-1], c);
        }
        String xml="<?xml version=\"1.0\"
encoding=\"utf-8\"?>\n"+xs.toXML(results);
        return xml;
    }
}
```

```
    }  
    //把list数据转换成JsonArray格式的数据  
    public static String listToJsonArray(List obj){  
        JSONArray json=JSONArray.fromObject(obj);  
        return json.toString();  
    }  
}
```

TotalJson 类

```
package bean;  
import java.util.List;  
  
public class TotalJson {  
    private long results;  
    private List items;  
    public long getResults() {  
        return results;  
    }  
    public void setResults(long results) {  
        this.results = results;  
    }  
    public List.getItems() {  
        return items;  
    }  
    public void setItems(List items) {  
        this.items = items;  
    }  
}
```

2.3.5 表单控件 Form

表单在 web 应用中处于非常重要的地位，时刻都在使用表单收集用户信息与交互，并将收集到的有用信息提交到后台服务器，表单是客户端与服务器之间通信的主要桥梁。表达能力有限，描述性的东西我就不多说了，如果你用上了 Ext 你就慢慢能理解它了。我们还是先看代码：

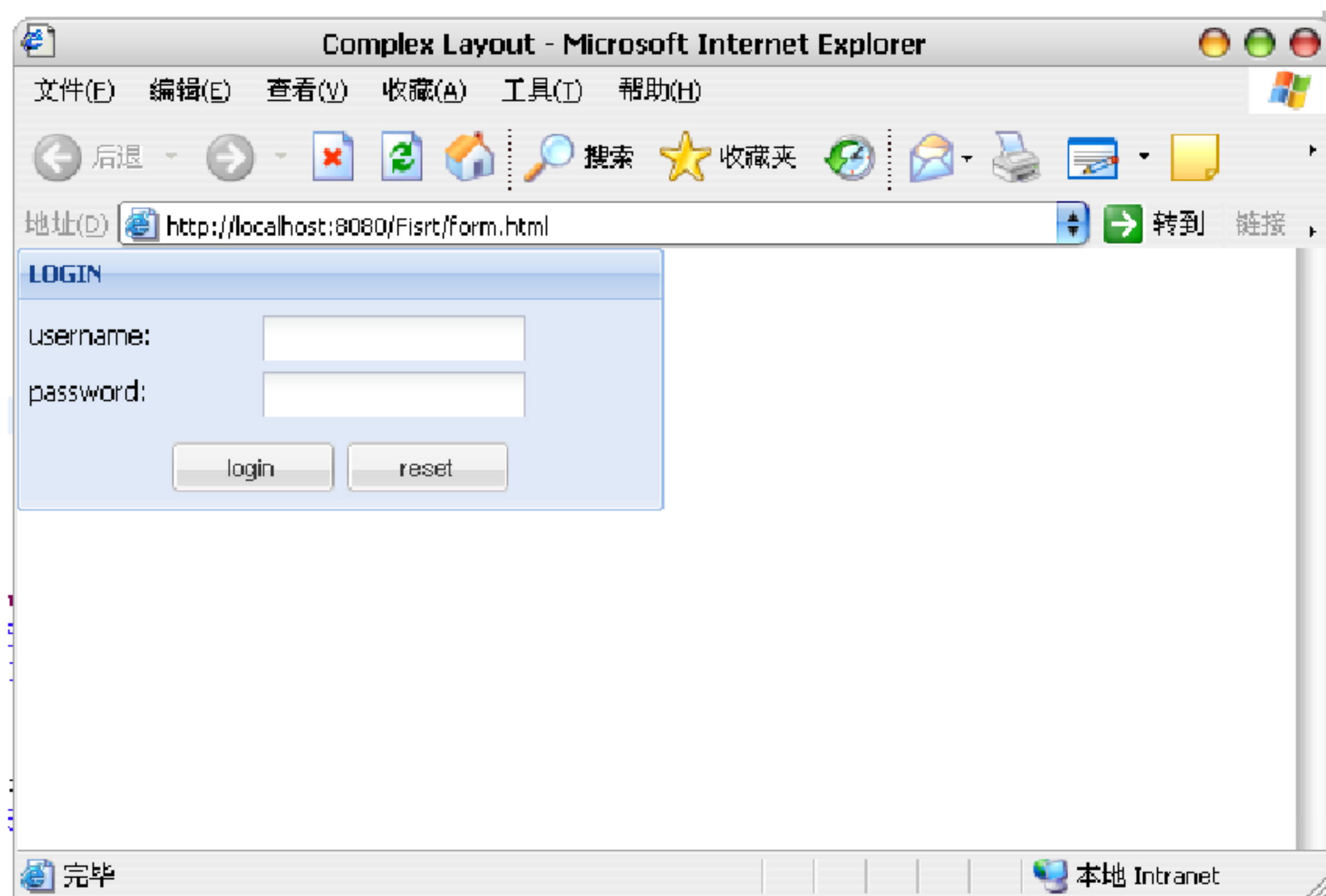
```
Ext.onReady(function() {  
    Ext.BLANK_IMAGE_URL='images/default/s.gif';  
    var formPanel=new Ext.form.FormPanel({  
        title:'LOGIN',  
        id:'elId',
```

```
        autoHeight:true,
        x:200,
        y:200,
        width:300,
        renderTo:Ext.getBody(),
        frame:true,
        cls:'text-align:center',
        labelAlign:'center',
        items:[{
            xtype:'textfield',
            name:'username',
            fieldLabel:'username',
            allowBlank:false,
            blankText:'请输入用户名',
            msgTarget:'under'
        },{
            xtype:'textfield',
            fieldLabel:'password',
            name:'pwd',
            allowBlank:false,
            inputType:'password',
            blankText:'密码不能为空',
            msgTarget:'under'
        }],
        buttonAlign:'center',
        buttons:[{
            xtype:'button',
            text:'login',
            scope:this,
            handler:login
        },{
            xtype:'button',
            text:'reset',
            scope:this,
            handler:reset
        }
    ]
});
//登录方法
function login() {
    formPanel.form.submit({
        clientValidation:true,
        waitMsg:'正在登录系统请稍候.....',
        url:'getDate.do?method=login',//去到后台服务器的地址
        method:'POST',
```

```
        //成功
        success:function(form, action){
            Ext.Msg.alert('提示','系统登录成功');
        },
        //失败
        failure:function(form,action){
            Ext.Msg.alert('提示','系统登录失败');
        }
    });
}
//重置的方法
function reset(){
    formPanel.form.reset();
}
});

//后台action的方法
public ActionForward login(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
    throws IOException{
    response.setContentType("charset=UTF-8");
    String username=request.getParameter("username");
    String pwd=request.getParameter("pwd");

    UserDao userDao=new UserDao();
    //验证数据的正确与否
    boolean flag=userDao.login(username, pwd);
    if(flag){
        response.getWriter().write("{success:true}");
    }else{
        response.getWriter().write("{success:false}");
    }
    return null;
}
```

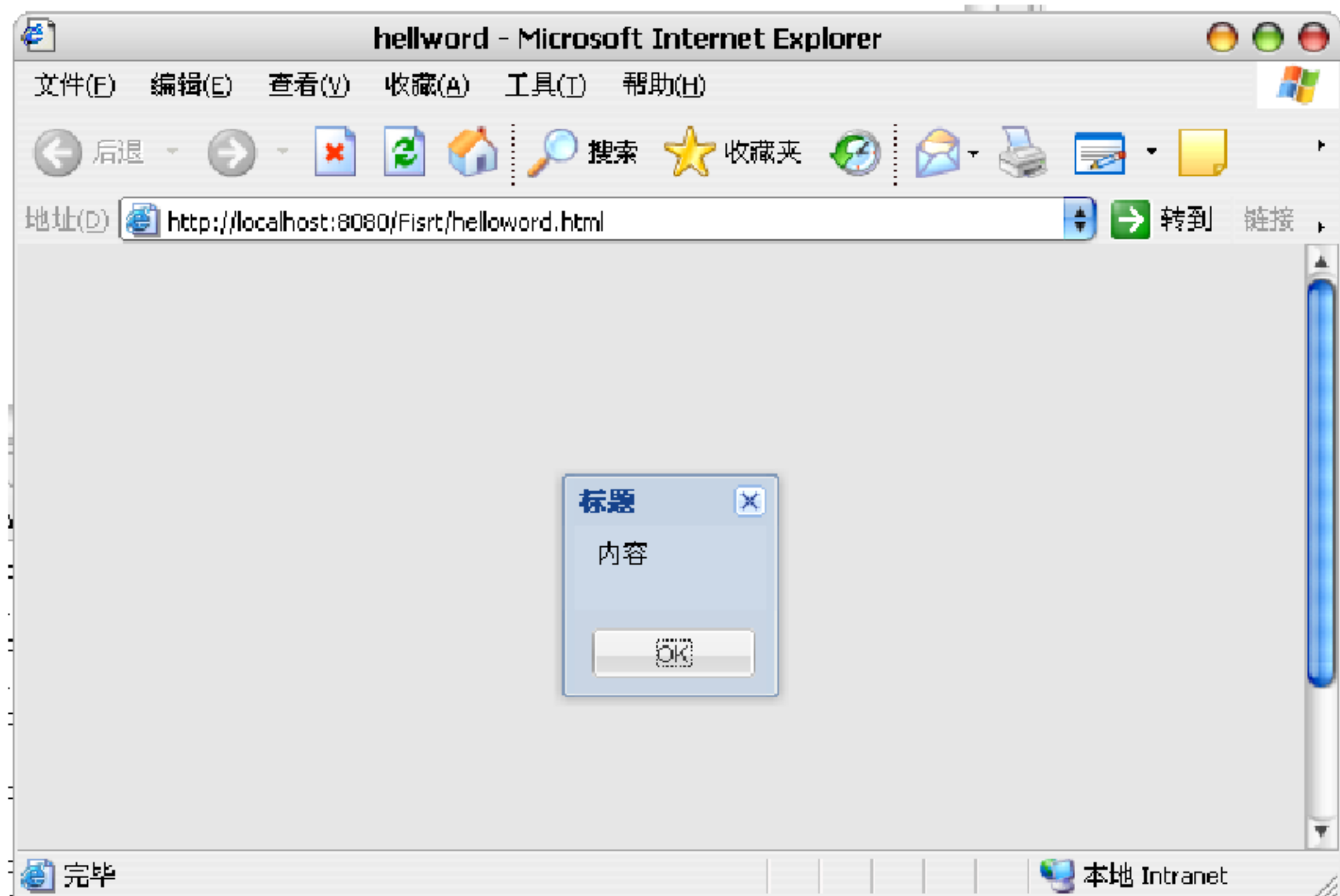
这是简单的一个登录示例，进行了后台和数据的验证

2.3.6 弹出窗口

对浏览器原声的 `alert()`, `confirm()`, `prompt()`进行了一些改进，更灵活了些
看代码：

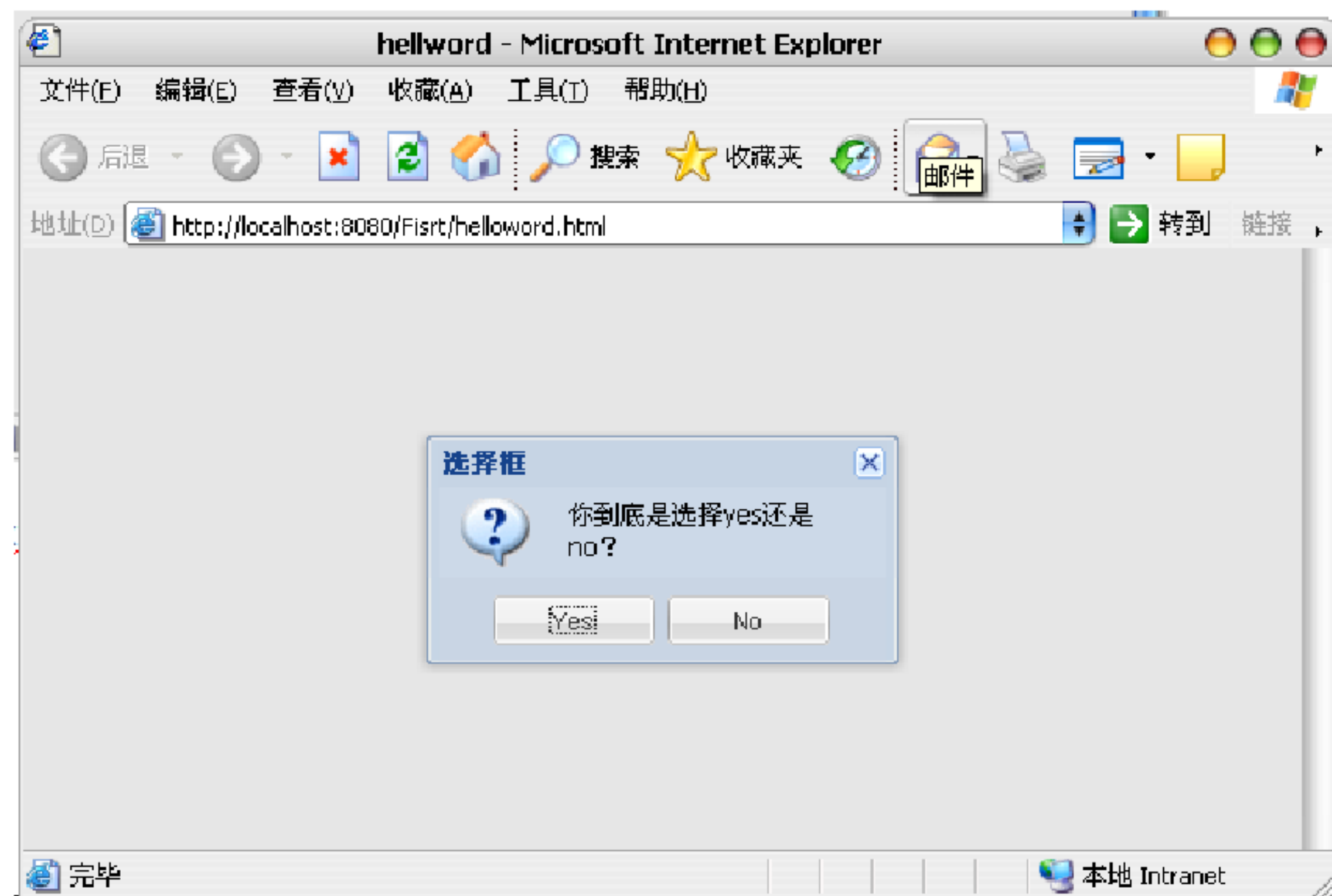
`Ext.MessageBox.alert()`

```
Ext.onReady(function() {  
    Ext.MessageBox.alert('标题', '内容', function(btn) {  
        alert('你刚刚点击了 ' + btn);  
    });  
});
```



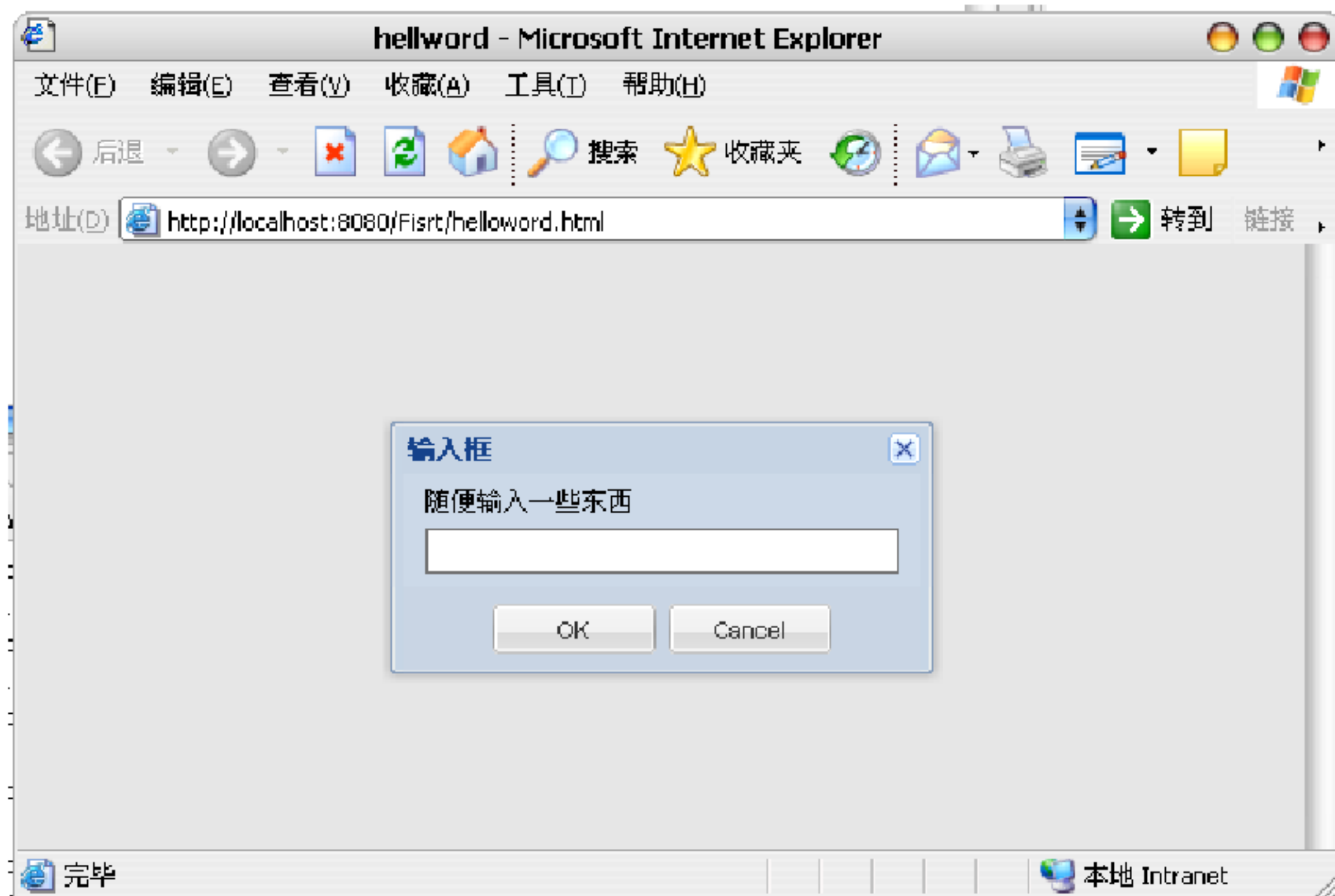
Ext.MessageBox.confirm()

```
Ext.onReady(function() {  
    Ext.MessageBox.confirm('选择框', '你到底是选择yes还是no?',  
        function(btn) {  
            alert('你刚刚点击了 ' + btn);  
        });  
});
```

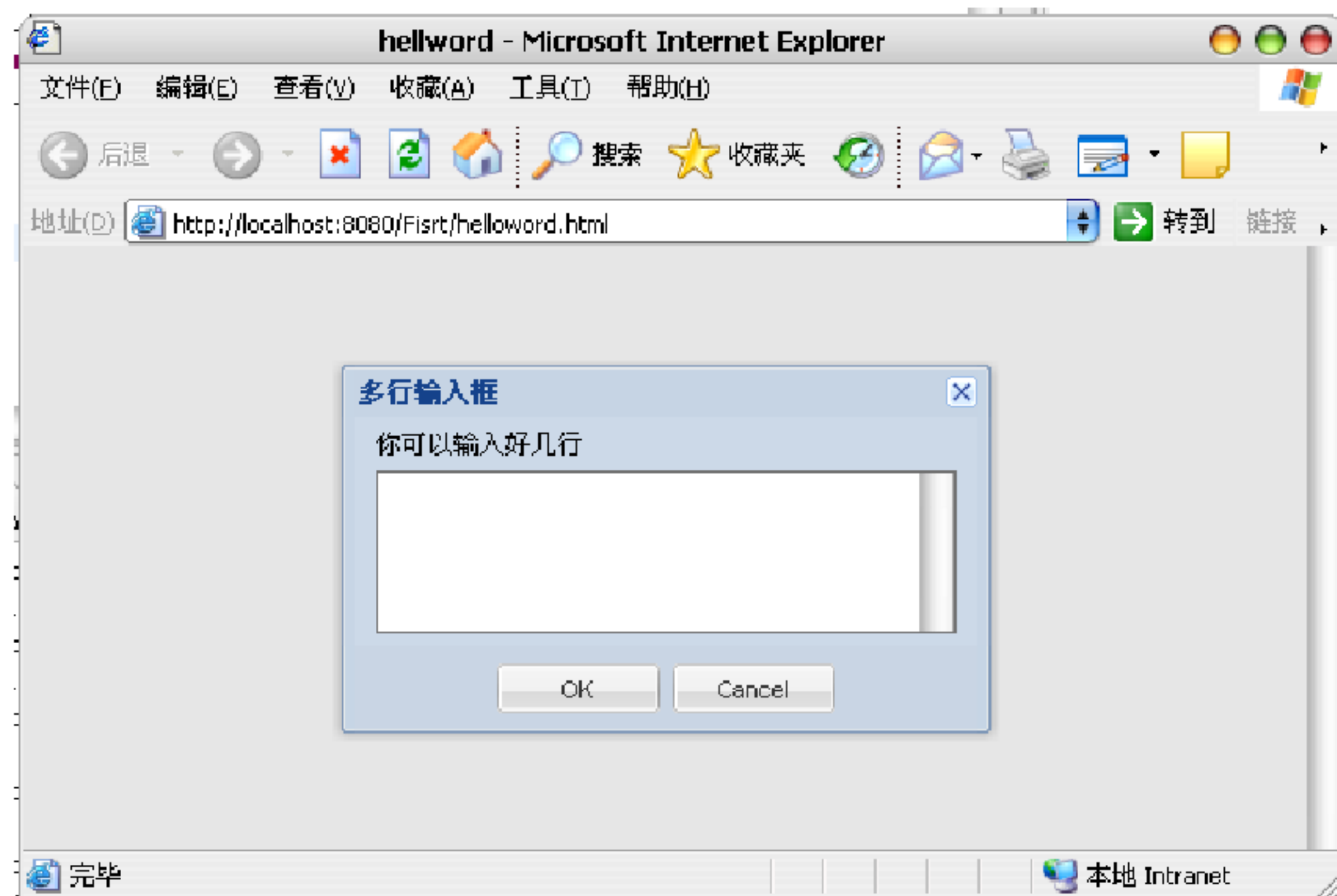
Ext.MessageBox.prompt()

```
Ext.onReady(function() {  
    Ext.MessageBox.prompt('输入框', '随便输入一些东西', function(btn,  
text) {  
        alert('你刚刚点击了 ' + btn + ', 刚刚输入了 ' + text);  
    });  
});
```

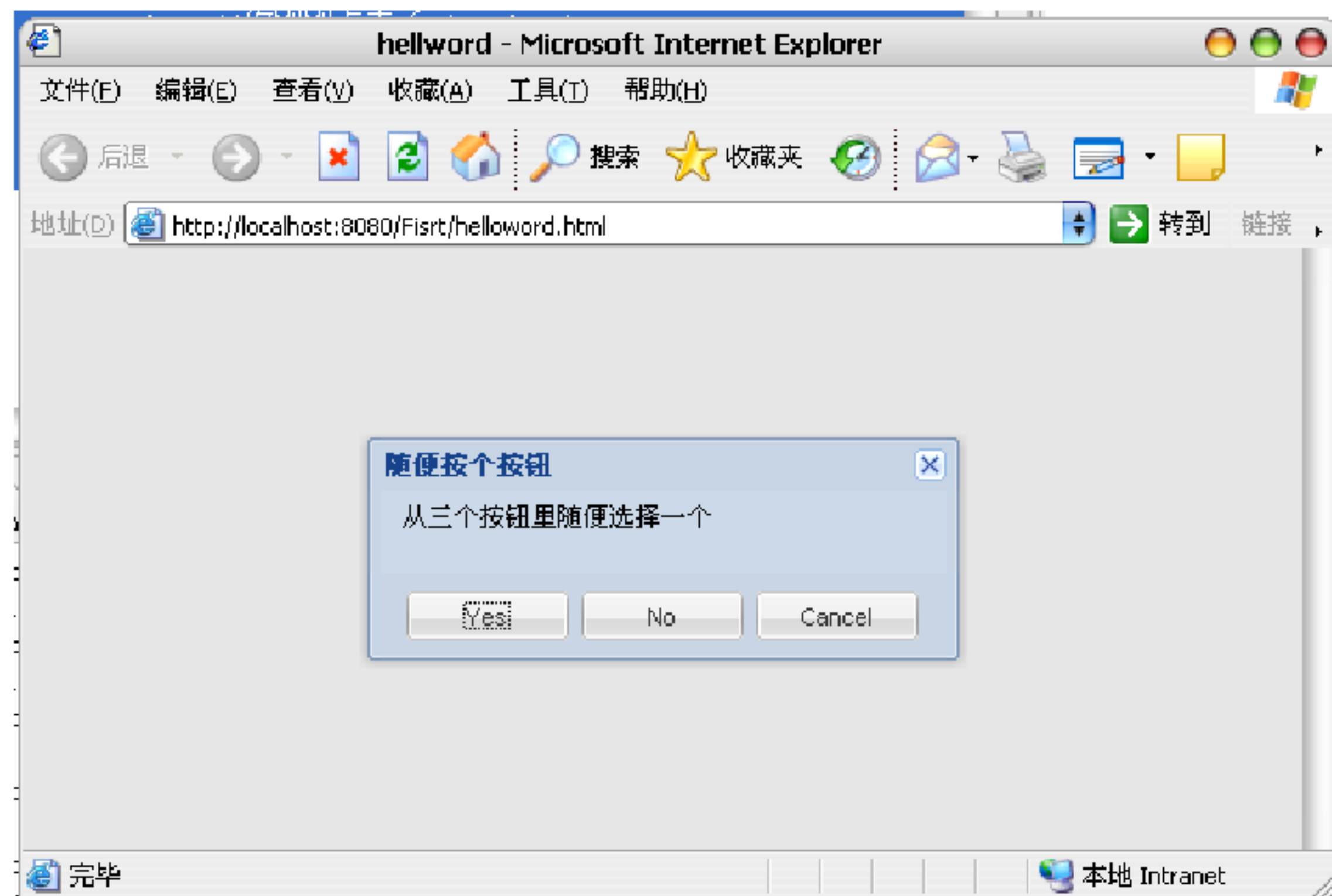


可以输入多行的输入框

```
Ext.onReady(function() {  
    Ext.MessageBox.show({  
        title: '多行输入框',  
        msg: '你可以输入好几行',  
        width: 300,  
        buttons: Ext.MessageBox.OKCANCEL,  
        multiline: true,  
        fn: function(btn, text) {  
            alert('你刚刚点击了 ' + btn + ', 刚刚输入了 ' + text);  
        }  
    });  
});
```

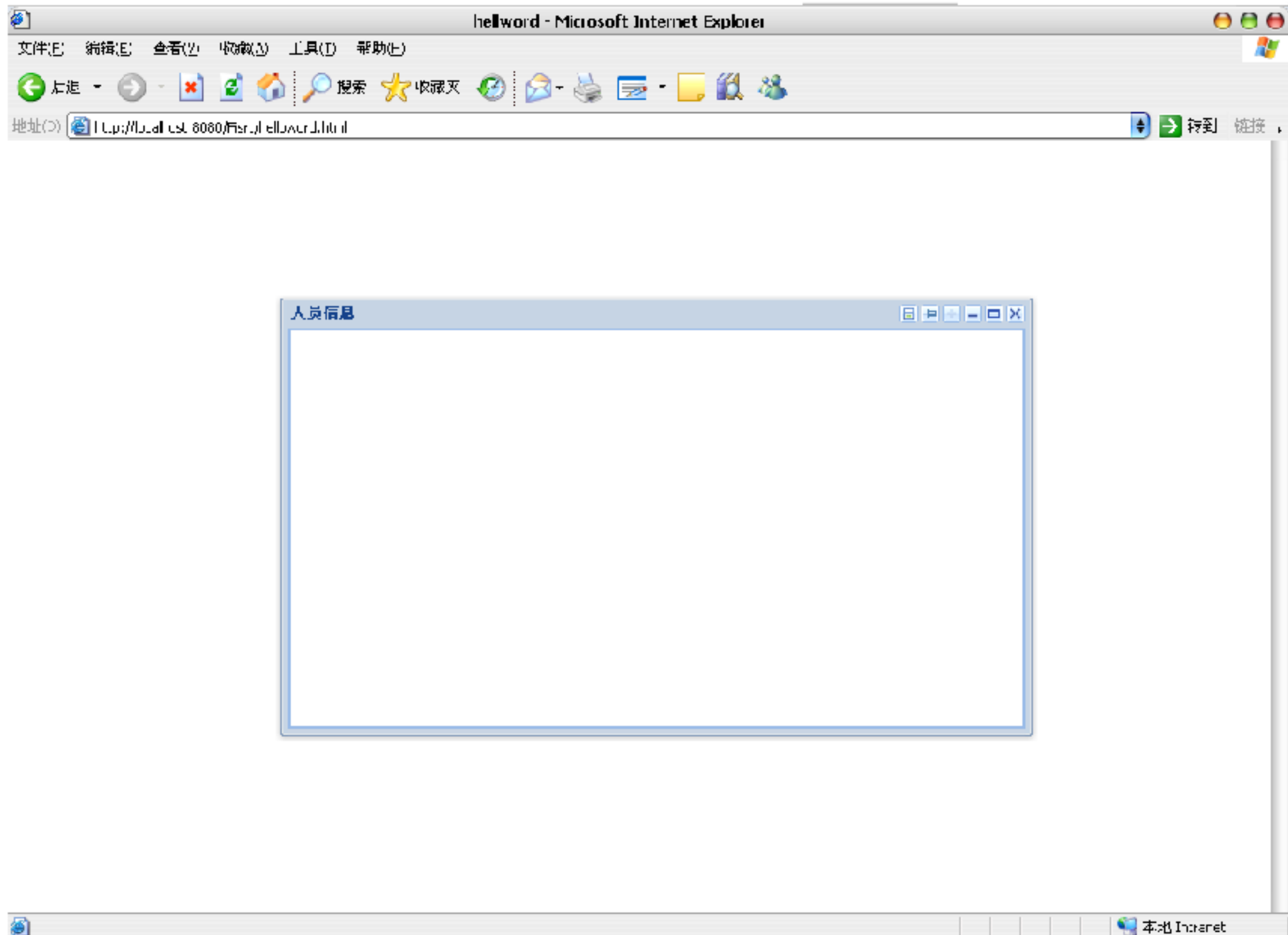



```
Ext.onReady(function() {  
    Ext.MessageBox.show({  
        title : '随便按个按钮',  
        msg : '从三个按钮里随便选择一个',  
        buttons : Ext.MessageBox.YESNOCANCEL,  
        fn : function(btn) {  
            alert('你刚刚点击了 ' + btn);  
        },  
        animEl : 'dialog'  
    });  
});
```



弹出一个 window

```
Ext.onReady(function() {  
    var win=new Ext.Window({  
        title:'人员信息',  
        closable:true,  
        width:600,  
        height:350,  
        minimizable:true,  
        maximizable:true,  
        //border:false,  
        //plain:true,  
        layout:'fit',  
        tools:[{id:'save'},{id:'unpin'},{id:'plus'}],  
        items:[{}]  
    });  
    win.show(this);  
});
```

3 Ext 核心

3.1 Ext.Element

在一个网页文档里包含了许多HTML 标签，而这些HTML 标签在DOM 树中会转换成一个一个的HTMLElement，从而便于脚本引用。由于浏览器DOM 操作的差异，为了实现Web 框架的跨浏览器特性，每个框架都会构建一个新类，用来操作和遍历DOM 树，而在Ext 框架中实现该功能的类就是Ext.Element。

在 Ext 中，可通过Ext.get 方法正获取HTMLElement 节点的Ext.Element 实例，其语法如下面的代码所示，其中myElementId 为HTMLElement 的id 属性。

```
var el = Ext.get(el);
```

//el 可以为节点id、DOM 节点或已存在的Element

使用 Ext.get 会创建一个Ext.Element 的实例，并可在后期引用时使用。如果不需要创建这个实例，只是对HTMLElement 执行一次性的操作，那么可以使用Ext.fly 方法。Ext.fly 方法并不创建Ext.Element 实例，只是利用全局共享的Ext.Element 实例进行操作，其语法如下面的代码所示。

```
Ext.fly(el).getHeight();
```

//el 可以为节点id、DOM 节点

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';
```

3.2 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']);
```

removeClass：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();
```

3.1 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DomeHelper-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'
});
```

3.1 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。