第 2 章

EasyUI表单

表单（Form）是 Web 前端开发中最常用的标记之一，通常使用表单来提交用户输入的数据以及上传文件。一个表单由三部分组成：表单标记、表单域、表单按钮。表单作为用户与服务器交互的重要枢纽，频繁受到各类恶意攻击，在设计表单时通常需要对表单域内的各类控件进行细致的过滤，这无疑增加了我们的开发难度。EasyUI 提供了丰富的表单控件，可以帮助我们快速设计一个功能强大的表单。本章将先介绍表单域内的常用组件，再将详细介绍如何初始化以及提交表单。

本章主要涉及的知识点有：

- 各类表单组件的使用方法。
- EasyUI 中值的含义。
- EasyUI 组件的依赖关系。
- 表单的提交和初始化。

2.1 文本框简介

文本框通常用来接收用户输入的数据。我们先来看一个使用 HTML 设计的登录页面，部分代码如下：

```
01  <div>账号<input  type="text"></div>
02  <div>密码<input  type="password"></div>
03  <button>提交</button>
```

运行结果如图 2.1 所示。

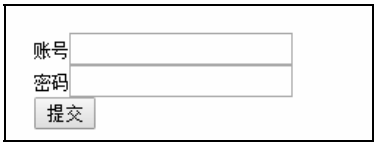

图 2.1　HTML 登录页面

读者先想一想这个登录页面有哪些问题。

首先我们没有限制账号、密码文本框为必填字段，这就导致登录用户可以什么也不输入就向服务器发送一个无效的登录请求。

其次如果后端开发人员设计的登录校验 SQL 语句为：

```
SELECT * FROM accounts WHERE username='账号' AND password = '密码'
```

正常情况下用户在账号输入框内输入"admin"，在密码输入框内输入"password"，后台执行的 SQL 语句就为：

```
SELECT * FROM accounts WHERE username='admin' AND password = 'password'
```

但是如果在账号输入框内输入"admin' AND 1=1 /*"，在密码输入框内输入任意字符串，那么后台执行的 SQL 语句就变成了：

```
SELECT * FROM accounts WHERE username='admin' AND 1=1 /* and password = 'aaa'
```

可以看到数据库实际执行的 SQL 为：

```
SELECT * FROM accounts WHERE username='admin' AND 1=1
```

“/*”后面的 SQL 语句被当作注释而忽略掉了，此时用户就可以绕开密码登录系统。

SQL 注入以及 XSS 攻击等常见的网络攻击手段，其根本原理就是用户输入的数据没有经过充分的检查和过滤，意外变成了代码被执行，我们在设计表单的各个输入控件时，一定要进行相应的过滤，如限制用户只能输入数字、用户输入长度不能超过指定范围等。

要解决上述问题我们必须限制账号、密码文本框的输入内容，例如限制账户、密码不能为空；限制用户输入的长度不能超过指定值，此外还需要限制账号文本框的输入仅能为字母、数字或者下画线。在提交表单前，我们还需要检查每个字段的输入是否合法。要做到这些我们需要设计大量的 JavaScript 逻辑进行判断，这无疑增加了前端开发的难度。

针对这个开发难点，EasyUI 提供了验证框（ValidateBox）来帮助开发者对用户输入的内容进行验证。

2.1.1　验证框（ValidateBox）

验证框是为了防止提交无效字段而设计的，当用户输入无效值时，它将改变背景颜色，并且显示警示图标和提示消息。验证框的默认配置定义在$.fn.validatebox.defaults 中。

许多初学者会误以为文本框（TextBox）为 EasyUI 中最基本的输入框，验证框扩展于文本框。其实在 EasyUI 中文本框是扩展于验证框的一个输入框，一旦混淆了两者之间的扩展关系，就会产生诸如如何给验证框加上图标等困惑。关于 EasyUI 中的依赖和扩展关系，本书将会在 2.1.2 小节详细讲解。

1. 创建验证框

使用标记创建验证框的方法如下：

```
<input id="v" class="easyui-validatebox"
data-options="required:true,validType:'email'">
```

使用 JavaScript 创建验证框的方法如下：

```
<input id="v">
    $('#v').validatebox({
        required: true,
        validType: 'email'
    });
```

2. 验证框属性

验证框常用的属性说明见表 2.1。

表 2.1　验证框常用属性说明

名称	类型	描述	默认
require	boolean	定义字段是否应被输入	false
validType	string,array	定义字段的验证类型，比如 email、url 等	null
invalidMessage	string	当文本框的内容无效时的提示文本	null
missingMessage	string	当文本框的内容为空时的提示文本	This field is required
tipPosition	string	定义当文本框的内容无效时提示消息的位置，可能的值有'left' 'right' 'top' 'bottom'	right
deltaX	number	消息提示组件在 X 方向上的偏移量	0
novalidate	boolean	当设置为 true 时，禁用验证功能	false
delay	number	延迟验证最后的输入值	200
editable	boolean	定义验证框是否可被编辑	true
disabled	boolean	定义是否禁用验证框	false
readonly	boolean	定义验证框是否为只读模式	false
validateOnCreate	boolean	定义是否在页面加载完毕后立刻进行一次验证	true
validateOnBlur	boolean	定义是否在失去焦点后进行一次验证	false

通常我们称光标进入某个组件时，该组件获得焦点；当光标离开组件时，该组件失去焦点。

validType 属性定义该字段的验证类型，例如 email、url、length 等。当验证单个规则时，validType 属性的值为字符串类型，如 validType:'email'。当验证多个规则时，validType 属性的值为数组类型，如 validType:['email','length[0,20]']。EasyUI 提供的验证规则有：

- email：检查输入是否为邮箱格式。
- url：检查输入是否为合法的地址格式。
- length[0,10]：检查输入的字符长度是否在指定范围区间。
- remote['http://.../check.php','paramName']：发送 ajax 请求来验证输入值，验证通过时返回'true'。

length 是按照字符计算长度，而非字节。字符与字节的区别在于：一个汉字和英文字母都只算一个字符，而一个汉字占两个字节以上，一个英文字母只占一个字节。

对于 EasyUI 未支持的验证规则，开发者也可以自定义验证规则，如下代码自定义一个验证两次密码输入是否一致的规则：

```
<input id="pw1" name=" pw1" type="password" class="easyui-validatebox"
data-options="required:true"><!—密码文本框 A-->
<input id="pw2" name="pw2" type="password" class="easyui-validatebox"
  required="required" validType="equals['#pw1]"> <!—密码文本框 B-->
 <script>
 $.extend($.fn.validatebox.defaults.rules, {
       equals: {
        validator: function(value,param){
        return value == $(param[0]).val();
       },
       message: '两次密码输入不一致'
    }
  });
</script>
```

其中$.extend($.fn.validatebox.defaults.rules,{})函数的意思是在 EasyUI 默认的验证规则中添加我们自定义的验证规则。equals 为我们自定义的验证规则名称，validator 函数中 value 参数为密码文本框 B 的值。param 为传递的参数；是一个数组，本例中传输的参数为#pw1，它是密码文本框 A 的 id，通过 id 可以获取密码文本框 A 的值$(param[0]).val()。最后判断其与密码文本框 B 的值是否相等，如果相等的话返回 true，此时验证通过；如果不相等的话，返回 false，同时将显示 message 中定义的验证失败提示内容。

使用自定义验证规则时，最重要的是理解 validator 函数的用法，其中 value 参数是验证字段的值，param 是其附带的数据。

3. 验证框事件

验证框常用事件说明见表 2.2。

表 2.2　验证框常用事件说明

名称	参数	描述
onBeforeValidate	none	验证字段前触发
onValidate	valid	验证字段时触发

4. 验证框方法

验证框常用方法说明见表 2.3。

表 2.3　验证框常用方法说明

名称	参数	描述
options	none	返回选项对象
destroy	none	移除并销毁该组件
validate	none	验证字段内容是否有效
isValid	none	调用 validate 方法并且返回验证结果，true 或者 false
enableValidation	none	启用验证
disableValidation	none	禁用验证
resetValidation	none	重置验证
enable	none	启用验证框
disable	none	禁用验证框
readonly	mode	启用/禁用只读模式

enableValidation 和 disableValidation 仅仅是启用/禁用验证，设置 disableValidation 为 true 后不会对用户的输入进行验证，而 disable 则禁用整个验证框，用户无法进行输入操作。

readonly 可以启用或者禁用只读模式，例如：

```
$('#v').validatebox('readonly');        // 启用只读模式
$('#v').validatebox('readonly',true);   // 启用只读模式
$('#v').validatebox('readonly',false); // 禁用只读模式
```

isValid 方法可以返回当前验证结果，通常用于提交数据前检测用户输入是否通过验证，例如：

```
$("#v").validatebox('isValid');
```

接下来我们使用验证框设计一个用户登录页面，部分代码如下：

```
    <div style="margin:20px 20px;">
            账号 <input id="account">
        </div>
        <div style="margin:20px 20px;">
            密码 <input id="password">
```

```
        </div>
        <div style="margin:20px 150px;">
            <button id='login'> 登录</button>
        </div>
        <script>
        $(function(){
            //自定义验证规则，只能输入英文和数字或者下画线
            $.extend($.fn.validatebox.defaults.rules, {
                englishOrNum : {
                validator : function(value) {
                return /^[a-zA-Z0-9_]{1,}$/.test(value);
                },
                message : '请输入英文、数字、下画线或者空格'
            }
        });
        $("#account").validatebox({
            required :true,                          //设置输入不能为空
            missingMessage :'请输入账号',            //输入为空时显示的提示
            invalidMessage:'请输入合法的账号格式',//输入验证失败时显示的提示
            validType: ['length[5,10]','englishOrNum'],
            //多个验证规则使用数组表示，长度在 5 至 10 个字符，英文、数字、下画线
            tipPosition:'bottom',                    //提示框的位置
            validateOnCreate:false,                  //页面加载完成后不进行一次验证
        });

        $("#password").validatebox({
            required :true,                          //设置输入不能为空
            missingMessage :'请输入密码',            //输入为空时显示的提示
            invalidMessage:'请输入合法的密码格式',//输入验证失败时显示的提示
            validType: 'length[6,13]',
            //单个验证规则使用字符串表示，长度在 5 至 10 个字符，英文、数字、下画线
            deltaX:-10,
            //提示框向左边便宜 10 个单位，数值为负数向左偏移，为正数向右偏移。
            validateOnCreate:false,                  //页面加载完成后不进行一次验证
        });
        $("#login").click(function(){
            //通过 isValid 方法检查是否验证通过
            if($("#account").validatebox('isValid')){
            alert("账号通过验证");
        }else{
            alert("账号未通过验证");
        }
        if($("#password").validatebox('isValid')){
            alert("密码通过验证");
        }else{
            alert("密码未通过验证");
        }
    });
});
</script>
```

最终运行结果如图 2.2 所示。

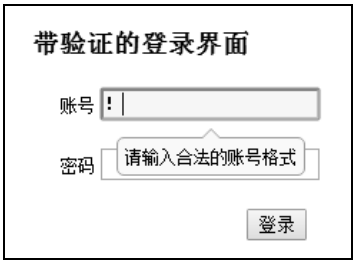

图 2.2　使用验证框设计登录页面

【本节详细代码参见随书源码：\源码\easyui\example\c2\validateLogin.html】

5. 服务器端验证用户输入

验证框可以通过 remote 规则来向服务器请求远程验证，注意当验证通过时服务器需要返回字符串'true'。通常我们会在自定义规则中使用 ajax 来请求远程验证，如下代码使用自定义规则来验证账号是否已被注册，部分代码如下：

```
//该规则用于验证账号是否已被注册
accountvalidate : {
    validator : function(value, param) {
    //获取用户输入的账户名
        var account = value.trim();
        var result;//保存验证的结果
        $.ajax({
            type : 'post',
            async : false,//设置同步请求
            url : 'server/checkaccount.php',
            data : {
                //向服务器传递的参数，php 中可以使用$_POST['account']来获取该值
                "account" : account
                },
            success : function(data) {
                //data 为服务器处理完毕后传递给客户端的值
                    result = data;
            }
        });
        //resault 为 true 时验证通过
        if(result=='0'){
            return true;
        }
        else{
            return false;
        }
    },
    message : '用户名已经被占用'
},
```

【本节详细代码参见随书源码：\源码\easyui\example\c2\remoteValidate.html】

使用服务器端验证时必须设置 ajax 为同步请求。所谓的同步请求，就是必须获取到服务器返回的值后 JavaScript 代码才会向下执行，否则会一直等待服务器处理结果。因为验证规则中必须通过服务器返回的结果来判断验证是否通过，因此此时需要设置其为同步请求。

读者可以运行实例程序，在账号中输入 sj123、xiaom11、admin、vsi1sk，此时会出现用户名已被占用的提示。

2.1.2　文本框（TextBox）

回到图 2.1 中，我们使用 HTML 创建了一个登录页面，在 2.1.1 小节中带领读者使用 EasyUI 验证框解决了如何验证用户输入的问题。使用 HTML 开发还面临着另一个巨大的挑战，就是页面美观问题，通常前端开发人员都需要编写大量的 CSS 来美化页面。由于 CSS 编写规范不统一，经常会出现不同的 CSS 文件冲突，从而导致网站整体设计无法达到预期效果。例如，在图 2.2 所示的登录页面中通过给验证框添加外边框来进行简单的页面排版。文本框会使用指定的主题样式对组件进行渲染，从而节省开发者的美化时间。

文本框的依赖关系如下：

- validatebox
- linkbutton

文本框扩展于：

- validatebox

文本框的默认配置定义在$.fn.textbox.defaults 中。

1. 创建文本框

使用标记创建文本框的方法如下：

```
<input  class= "easyui-textbox" >
```

使用 JavaScript 创建文本框的方法如下：

```
01  <input  id="tb"  type="text ">
02  $('#tb').textbox();
```

2. 文本框属性

文本框常用的属性说明见表 2.4。

表 2.4　文本框常用属性说明

名称	类型	描述	默认值
width	number	文本框的宽度	auto
height	number	文本框的高度	auto
cls	string	给文本框添加一个 CSS 类型	
prompt	string	在文本框中显示的一段提示	
value	string	默认值	
type	string	文本框类型，可能的值有'text'和'password'	text
label	string,selector	文本框标签的名称	null
labelWidth	number	标签的宽度	auto
labelPosition	string	标签的位置，可能的值有'before' 'after'和'top'	before
labelAlign	string	标签对齐方式，可能的值有'left'和'right'	left
multiline	boolean	定义文本框是否可多行输入	false
editable	boolean	定义文本框是否可被编辑	true
disabled	boolean	定义是否禁用文本框	false
readonly	boolean	定义文本框是否为只读模式	false
icons	array	定义文本框中的图标，每个图标都拥有以下属性： ● iconCls：图标类型 ● disabled：图标是否可被单击 ● handler：图标单击后的事件	[]
iconCls	string	图标类型	null
iconAlign	string	图标对齐方式，可能的值有'left'和'right'	right
iconWidth	number	图标宽度	18
buttonText	string	文本框中按钮的名称	
buttonIcon	string	文本框中按钮的图标	null
buttonAlign	string	文本框中按钮的对齐方式，可能的值有'left'和'right'	right

文本框继承了验证框全部属性、事件和方法，在实际开发中我们几乎不会直接使用到验证框，而由文本框派生出各类丰富的 EasyUI 组件。

在下面的内容中，本书将详细讲解文本框的属性、事件和方法，在本节末尾介绍属性、事件和方法的含义，并带领读者探讨 EasyUI 中的依赖关系。

首先来看一下 width、height 属性，从字面意思可以理解这是一个设置文本框尺寸的属性，可以通过比例或者固定的像素值来设置宽度，使用像素值来设置高度。例如：

```
<input  class="easyui-textbox"  data-options="width:'90%'">
```

```
<input  class="easyui-textbox"  data-options="width:360,height:20">
```

此处我们设置的 width:100%是相对于它的上一层父元素的宽度比，当设置为比例时应给值加上引号，请看下面示例：

```
01         <div  id="parent2"  style="width: 500px;">
02           <div  id="parent1"  style="width: 400px;">
03              <input  class="easyui-textbox"
data-options="width:'90%'">
04           </div>
05       </div>
```

最终运行结果如图 2.3 所示。

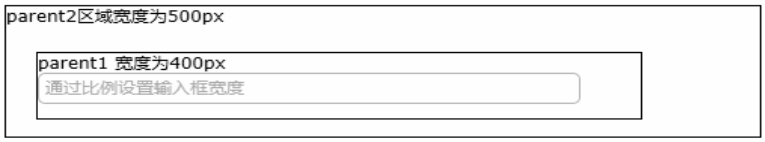

图 2.3　使用比例设置宽度

这个例子里面 parent1 为文本框的父元素，它的宽度为 400px，parent2 为 parent1 的父元素，它的宽度为 500px，我们设置文本框的宽度比为 90%，这个比例是相对于 parent1 的，因此文本框的宽度也就是 360px。如果组件通过像素值设置尺寸，我们称其为静态布局。如果使用百分比设置尺寸，我们称其为流式布局。

cls 参数用于给文本框添加一个新的风格，例如设置文本框的下外边距为 10，相关 CSS 代码如下：

```
01      <style>
02          .newStyle{
03              margin-bottom:10px;
04          }
05      </style>
```

给文本框添加该风格的相关代码如下：

```
01      $('#tb).textbox({
02          cls:'newStyle'
03      });
```

prompt 属性用于当文本框中无任何内容时显示的提示，例如：

```
$('#tb).textbox({prompt:'请输入账号'});
```

value 属性为文本框加载完毕后显示的初值。

type 属性可以设置文本框的输入类型，当设置为 password 时，用户的输入将会被替换成指定的字符以避免密码泄露。

label 为文本框中的一个标签，在图 2.2 所示的例子中，“账号”“密码”这些提示用户输

入的字符串通常会写在对应组件的前面，但是当这些字符串长度不一致时页面就会变得混乱，如图 2.4 所示。

图 2.4　一个排版混乱的页面

通过 label 属性可以解决这个问题，其中 labelWidth 为标签的宽度，labelPosition 为标签的显示位置，labelAlign 为标签的对齐方式，详细的用法示例如下：

```
        <div><input id="nickname"></div>
        <div><input id="phone"></div>
    <script>
    $(function(){
        $('#nickname').textbox({
            label:'昵称',
            labelPosition:'left',//显示在文本框的左侧
            labelAlign:'right',  //右侧对齐,字符串的最后一个字符对齐
            width: 300,
            cls:'newStyle'
        });
        $('#phone').textbox({
            label:'手机号码',
            labelPosition:'left',
            labelAlign:'right',
            width: 300,
        });
    });
    </script>
```

最终运行结果如图 2.5 所示。

图 2.5 使用标签对齐字符串

icons 可以给文本框添加图标，icons 为包含 icons 对象的数组，icons 对象有如下属性：

- iconCls：图标类型。
- disabled：定义单击图标后是否触发事件。
- handler：单击图标后触发的事件。

具体的代码示例如下：

```
        $('#tb').textbox({
            icons:[
```

```
                {
                    iconCls:'icon-man',
                    handler:function(e){
                        alert("图标被单击");
                    }
                },
            ],
        });
```

最终运行效果如图 2.6 所示。

图 2.6　带图标的文本框

iconAlign、iconWidth 属性定义了图标的对齐方式以及宽度，使用代码如下：

```
        $('#tb').textbox({
            icons:[
                {
                    iconCls:'icon-man',
                    handler:function(e){
                        alert("图标被单击");
                    }
                },
            ],
        });
```

上面的例子中使用的图标类型 icon-man 为 EasyUI 自带的图标，我们也可以添加一个自定义图标，详细步骤如下：

- 找到 EasyUI 框架下 themes 文件夹中的 icons 文件夹，将自定义的图标保存到该文件夹下。
- 打开 themes 文件夹下的 icon.css 文件，在文本末尾添加如下代码:

```
  .icon-extend-lock{
    background:url('icons/extend_lock.png') no-repeat center center;
  }
```

其中 extend_lock.png 为自定义图标的名称，icon-extend-lock 为我们自定义的图标的类型名称，使用图标类型名就可以显示我们自定义的图标，代码如下：

```
        $('#tb').textbox({
            icons:[
                {
                    iconCls:'icon-extend-lock',
                    handler:function(e){
                        alert("图标被单击");
                    }
```

```
                },
            ],
        });
```

本书在随书资料【\资源\图标\】目录下提供了大量可供读者使用的自定义图标。

文本框允许开发者为其添加一个按钮，buttonText 为按钮的名称，buttonIcon 为按钮的图标，buttonAlign 为按钮的对齐方式。示例代码如下：

```
$('#tb').textbox({
            buttonText:"按钮",
            buttonIcon:'icon-extend-lock',
            buttonAlign:'left'//左对齐
        });
```

最终运行结果如图 2.7 所示。

图 2.7 带按钮的文本框

3. 文本框事件

文本框常用事件说明见表 2.5。

表 2.5　文本框常用事件说明

名称	参数	描述
onChange	newValue,oldValue	当文本框中的内容发生改变时触发（仅在失去焦点时检查内容是否改变）
onResize	width,height	当文本框尺寸发生改变时触发
onClickButton	none	当文本框中按钮被单击时触发
onClickIcon	index	当文本框中图标被单击时触发

onChange 只能在内容发生改变且失去焦点时触发。

4. 文本框方法

文本框常用方法说明见表 2.6。

表 2.6　文本框常用方法说明

方法	参数	描述
options	none	返回选项对象

（续表）

方法	参数	描述
textbox	none	返回展示值框对象，开发者可以在这个对象上绑定任意事件
button	none	返回按钮对象
destroy	none	销毁文本框组件
resize	width	调整文本框组件的宽度
disable	none	禁用组件
enable	none	启用组件
readonly	mode	启用/禁用只读模式
clear	none	清空文本框组件的全部类型值
reset	none	重置文本框组件的全部类型值
initValue	value	初始化文本框的存储值，使用该方法不会触发 onChange 事件
setText	text	设置文本框的展示值
getText	none	获取文本框的展示值
setValue	value	设置文本框的存储值
getValue	none	获取文本框的存储值
getIcon	index	获取图标对象

textbox 返回的是展示值框对象，因此开发者无法使用该对象重新初始化文本框，关于展示值框的概念请查看 2.1.3 节内容。如下代码是错误的：

```
01  var tb = $("#tb").textbox("textbox");
02  tb.textbox({
03        width:100
04  });
```

运行后会发现在文本框中又嵌套了一个文本框。

5. 属性、方法和事件

一个手机都会拥有尺寸、音量、屏幕等元素，这些是组成一个手机的必要元素，我们称这些元素为属性。手机出厂后通常都会替消费者设置好默认的屏幕亮度以及音量，这一个过程我们称为初始化。在 EasyUI 中组件的属性在页面加载完毕后就会将其初始化好。如下代码设置文本框的初始值和初始类型属性，代码如下：

```
01  $('#tb).textbox({
02          value:'初始值',
03          type:’text’
04      });
```

在 jQuery 中可以通过$(function(){//页面加载完毕后的代码})的方式来处理页面加载完毕后的代码。

当手机接收到来电消息时，会亮起屏幕并且播放响铃，这一过程叫作事件。事件必须有特定的消息才会触发，事件也是在初始化时设置，例如当文本框中的内容发生改变时会触发 onChange 事件，写法如下：

```
$('#tb).textbox ({
    onChange:function(newValue,oldValue){
    }
});
```

使用者可以调节手机的音量大小以及屏幕显示的亮度，这一过程称为方法。方法通常是在属性初始化后改变属性的值，在文本框中的 editable、disabled、readonly、width、height、value 等属性都可以通过对应的方法将其改变，如 resize 方法改变宽度，initValue 方法重新设置初始值等。如下代码设置文本框为只读模式。

```
$('#tb').textbox('readonly',true);
```

使用者可以在手机设置功能中查看当前手机的各种配置，也就是说方法不仅可以修改属性的值也可以查看属性的值。例如，options 方法可以查看当前文本框的全部配置，我们称组件的当前配置对象为其选项对象。

使用者可以在手机上设置闹铃，闹铃只会在指定的时间才会触发，因此闹铃是一个事件，而使用者设置闹铃这一过程是一个方法，因此通过方法同样可以增加事件。文本框中 textbox 方法可以绑定任意的事件。如下代码给文本框绑定一个键盘按下的事件：

```
$('#tb').textbox("textbox").bind("keydown",function(e){
    var v = e.keyCode;//当前按键的 ASCII 码
});
```

6. 依赖与扩展

依赖更多的是含有一种组合的含义，而扩展更多的是继承含义，如果组件 A 依赖于组件 B，说明组件 A 由组件 B 组成。如果组件 A 扩展于组件 B，那么组件 A 中可以使用组件 B 的全部属性、事件、方法。关于依赖与扩展的含义本书将在第 6 章中做进一步讲解，目前读者仅需了解扩展组件可以使用被扩展组件的全部属性、方法、事件，例如文本框扩展于验证框，此时我们可以在文本框中使用验证框属性 required，例如：

```
<input class= "easyui-textbox" data-options ="required:true" >
```

所有的组件都拥有 options 方法，该方法以 JSON 格式返回一个选项对象，所谓的选项对象就是指组件初始化完毕后的配置。可以通过下面的 JavaScript 函数打印文本框的 options 对象：

```
function writeObj(obj){
    var description = "";
```

```
    for(var i in obj){
        var property=obj[i];
        description+=i+" = "+property+"\n";
    }
    alert(description);
}
writeObj($('#tb').textbox('options'));
```

运行结果如图 2.8 所示。

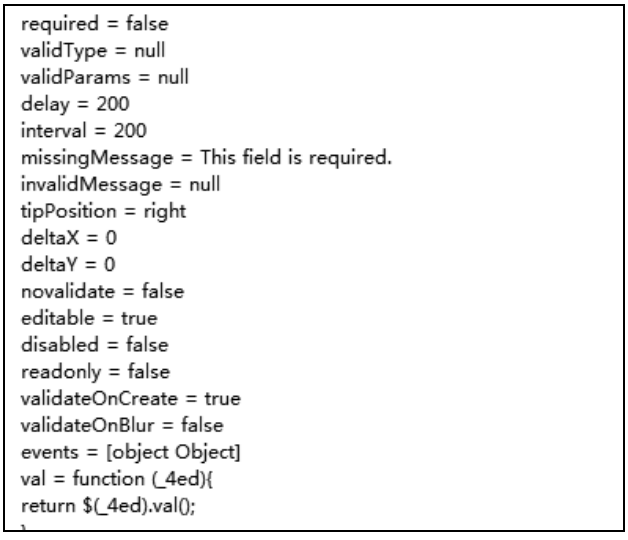

图 2.8　文本框选项对象的值

EasyUI 组件通常会使用对象作为属性、事件或者方法的参数，如果读者无法明确参数的含义，可以使用 writeObj 函数打印参数，或者使用 console.log()函数在控制台中打印参数。

options 是一个 JSON 格式的对象，我们可以通过相应的方法获取指定的字段值，例如通过 options 方法获取组件的 required 属性值，代码如下：

```
        var option = $('#tb').textbox('options');
        var required = option.required;
```

7. 默认配置

每个组件都会定义自己的默认配置，每当初始化组件时都会使用默认配置来初始化那些开发者未设置的属性或事件，例如我们可以获取文本框的默认宽度，代码如下：

```
$.fn.textbox.defaults.width
```

8. EasyUI 组件中的值

在 EasyUI 组件中有三种值，分别是：

- 输入值。用户输入的值可以是任意的字母。

- 存储值。根据用户输入不可信原则，用户输入的值必须经过相应的过滤和限制，存储值是将用户的输入进行过滤以及解析后的最终值。
- 展示值。用户并不希望显示一些枯燥的数字，例如用户更希望看到 XX 年 XX 月 XX 日格式的日期，而非一串时间戳数字，展示值就是将存储值格式化为指定的格式后的值。

我们称用户输入值为 Input，存储值为 Value，展示值为 Text。可以通过 initValue、getValue、setValue 方法来初始化存储值、获取存储值以及设置存储值。可以通过 getText、setText 方法来获取展示值、设置展示值。关于存储值和展示值的区别，下面举个简单的例子。中国的用户更希望看到例如 XX 年 XX 月 XX 日这样格式的日期，然而对于计算机而言，更希望处理时间戳格式的日期。此时我们会设计两个输入框，其中一个输入框展示 XX 年 XX 月 XX 日格式的日期，另一个输入框通常会隐藏起来，保存计算机能理解的时间戳，最终在提交表单时将时间戳传输给服务器。将存储值转化成展示值的过程称为格式化（formatter），将输入值转化成存储值的过程称为解释（parser）。EasyUI 中通常会使用包含 Value 的字符串来命名存储值的属性或者方法，使用包含 Text 的字符串来命名展示值的属性或者方法。读者在后续的学习中应当作到望文生义。

在文本框中必须先设置存储值，之后才能设置展示值。如下代码运行后会发现存储值和展示值都被设置为 2。

```
01 $('#tb').textbox('setText',"1");
02 $('#tb').textbox('setValue',"2");
```

9. EasyUI 方法的链式操作

EasyUI 组件的方法（除获取数据的方法外）返回的为该组件对象，因此可以对 EasyUI 方法使用链式操作，例如：

```
$('#tb').textbox('setValue',"2").textbox('setText',"1");
```

2.1.3 密码框（PasswordBox）

EasyUI 提供了专门用于输入密码的组件密码框。密码框提高了用户的交互性，它通过使用指定的字符来替换用户输入的密码从而防止用户密码泄露，密码框的右侧是一个眼状图标，可以通过单击该图标显示用户输入的密码。

密码框的依赖关系如下：

- textbox

密码框扩展于：

- textbox

密码框的默认配置定义在$.fn.passwordbox.defaults 中。

1. 密码框用法

使用标记创建密码框的方法如下：

```
<input  class="easyui-passwordbox" >
```

使用 JavaScript 创建密码框的方法如下：

```
<input id="pb" type="text" style="width:300px">
$(function(){
    $('#pb').passwordbox({
        prompt: 'Password',
        showEye: true
    });
});
```

2. 密码框属性

密码框常用属性说明见表 2.7。

表 2.7　密码框常用属性说明

属性名	类型	描述	默认值
passwordChar	string	密码框的展示字符	%u25CF
checkInterval	number	用户输入值转换成展示字符的时间间隔	200
lastDelay	number	用户最后一个输入值转换成展示字符的时间间隔	500
revealed	boolean	定义密码框是否直接显示用户输入值	false
showEye	boolean	定义是否显示右侧的眼状图标	true

3. 密码框事件

密码框在文本框的基础上无新增事件。

4. 密码框方法

密码框常用方法说明见表 2.8。

表 2.8　密码框常用方法说明

方法名称	参数	描述
options	none	返回选项对象
showPassword	none	显示密码框的存储值
hidePassword	none	隐藏密码框的存储值

2.1.4　数字框（NumberBox）

数字框用于过滤用户的输入值使用户仅能输入数字，可以把存储值转换为不同类型的展示值（比如：数字、百分比、货币，等等）。可以通过 formatter 方法来自定义展示格式，通过

parser 方法将输入值解析成存储。

数字框的依赖关系如下：

- textbox

数字框扩展于：

- textbox

数字框的默认配置定义在$.fn.numberbox.defaults 中。

1. 数字框的用法

使用标记创建数字框的方法如下：

```
<input type="text" class="easyui-numberbox" value="100"
data-options="min:0,precision:2">
```

使用 JavaScript 创建数字框的方法如下：

```
01  <input type="text" id="nn">
02  $('#nn').numberbox({min:0,precision:2});
```

2. 数字框属性

数字框常用属性说明见表 2.9。

表 2.9　数字框常用属性说明

名称	类型	描述	默认值
disabled	boolean	定义是否禁用组件	false
value	number	设置数字框的默认值	
min	number	允许的最小存储值	null
max	number	允许的最大存储值	null
precision	number	设置存储值小数点后的最大精度	0
decimalSeparator	string	展示值中分隔数字的整数部分和小数部分的分隔字符	
groupSeparator	string	展示值中分隔整数组合的字符	
prefix	string	展示值前缀字符串	
suffix	string	展示值后缀字符串	
filter	function(e)	过滤被按下的键	
formatter	function(value)	存储值格式化为展示值	
parser	function(s)	用户输入值转换成存储值	

其中 min、max、precision 属性主要用于控制输入值与存储值之间的转换规则，例如设置 precision 的值为 2，那么用户如果输入的是 3.12345 则会被自动过滤成 3.12。filter 属性主要用于过滤输入值，其参数 e 是一个事件对象，可以通过 e.keyCode 获取当前按下的键的 ASCII

码，返回 true 则接收该字符，返回 false 则禁止输入该字符。

decimalSeparator、groupSeparator、prefix、suffix 参数则为数字框内置的一些将存储值转换成展示值的规则，例如设置 prefix 值为美元符“$”，当存储值为 1 时，展示值则为“$1”。

formatter、parser 属性可以用来自定义输入值、存储值以及展示值之间的转化规则。formatter 用于将存储值格式化为展示值，parser 用于将输入值解析成存储值。初学者在使用这两个属性时经常会出现一系列的问题，这是因为没有理顺这两个属性触发的时机，下面我们将重点讲解，先看下面的代码：

```
        <input type="text" id="nn">
        <script>
        $(function(){
            $('#nn').numberbox({
                prefix:'$',
                //formatter 中接收的是一个存储值
                formatter:function(value){
                    alert("formatter");
                    return parseInt(value)+1;
                },
                //parser 中接收的是一个输入值
                parser:function(s){
                    alert("parser");
                    return parseInt(s)-1;
                }
            });
        });
    </script>
```

读者可以将其复制到自己的文件中运行，运行这段代码后我们可以发现：

- prefix 属性无法定义展示值前缀。
- 当页面刷新时会先执行 parser 中定义的方法，再执行一次 formatter。
- 当文本框内的内容发生改变时，会依次执行 parser、parser、formatter、formatter 属性中的方法。

第一个问题很容易理解，因为数字框默认在 formatter 属性方法中将存储值转化成展示值，在 parser 属性中将输入值转化成存储值。因此设置 prefix 属性后数字框会在默认的 parser 属性方法中检查用户输入值是否有指定前缀，有的话就将前缀移除并将处理后的值作为存储值保存，然后在 formatter 方法中的存储值前面加上前缀。上述代码中重新定义了 parser 和 formatter 属性，此时数字框默认的 parser 以及 formatter 属性将会被覆盖，因此 prefix 属性会失效。

在文本框中向读者讲解了值的概念，其实文本框在创建时会新增两个输入框，此时 HTML 如下：

```
<!–初始化框，开发者编写的标记，用于保存初始化配置和存储值，
通常也会将选项对象绑定到初始化框上->
<input type="text" id="nn" type="hidden">
<!–展示值框，文本框新增的标记，用于存放展示值-->
```

```
<input class="textbox-text">
<!---存储值框，文本框新增的标记，用于存放存储值-->
<input type="hidden" class="textbox-value">
```

读者可以发现，其实文本框向用户显示的仅仅是展示值框，而初始化框和存储值框会被隐藏，也就是说用户其实是在展示值框中进行输入的。此时我们再看这段代码：

```
        <input type="text" id="nn">
        <script>
        $(function(){
            $('#nn').numberbox({
                prefix:'$',
            });
        });
        </script>
```

这段代码的含义是在数字前面加上一个美元符$前缀，它的运行原理如下：当组件加载时会先将初始化框中的初始值使用 parser 属性中的方法进行解析，如果初始值是$111 的话会将其解析成 111，如果是其他格式的话例如 222 则仍然会解析成 222 并将解析后的值保存到存储值框中，接下来 formatter 属性会取出存储值，并在其前面加上前缀后保存到展示值框中显示。具体的过程如图 2.9 所示。

提示

由于用户输入值有可能是合法展示值，也有可能是合法的存储值，还有可能是一些非法值，所以在 parser 中需要对用户输入值进行判断和过滤。

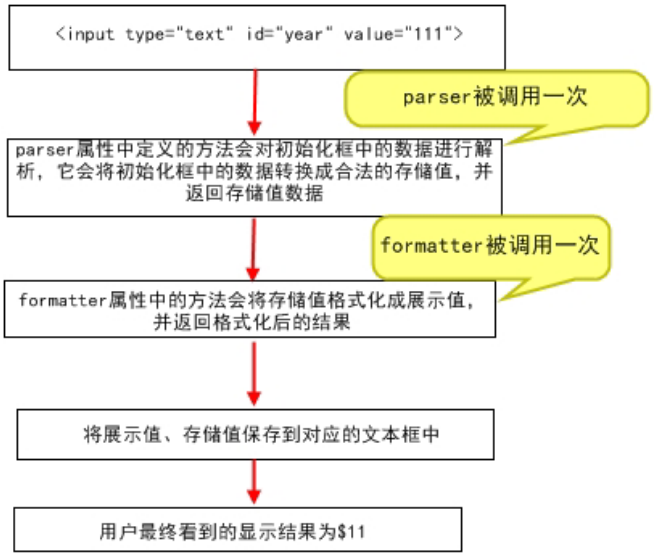

图 2.9　数字框初始化流程图

当数字框失去焦点时，数字框会调用一次 fix 方法，该方法中调用一次 parser 属性方法，将用户输入数据转换成存储值，接着该方法中会使用数字框的 setValue 方法保存存储值，但是

在 setValue 方法中也会调用一次 parser 属性方法，这是因为 setValue 方法可以被开发者直接调用，例如：

```
$('#nn').numberbox('setValue',’$11’)
```

因为该方法中仍然需要对传入的值进行解析，并将其转化成合法的存储值，这就是为什么 parser 属性会被调用两次的原因。接着会使用一次 formatter 属性方法格式化存储值，注意在处理完毕后会再调用一次 formatter 属性方法格式化存储值，这两次的调用区别是，第一次格式化的存储值是 parser 处理完毕后返回的值，第二次格式化的存储值是通过 getValue 方法获取的数字框存储值。

如果读者目前无法完全理解这两个属性的话也没关系，对于 parser 属性和 formatter 属性，读者只需要记住一句话，parser 属性是将用户输入的数据解析成合法的存储值，而 formatter 是将存储值格式化为展示值。下面我们利用这两个属性给文字添加美元符前缀，部分代码如下：

```
    <input type="text" id="nn" value="111">
    <script>
    $(function(){
        $("#nn").numberbox({
                parser:function(s){
                    s = $.trim(s.replace("$",""));
                    return s;
                },
                formatter:function(value){
                    return '$'+value;
                }
        });
    });
</script>
```

通过上述的讲解，读者必须理解如下两个知识点：

- 所有直接或间接扩展于文本框的组件向读者展示的都是其展示值框，由于初始化框被隐藏，因此除了部分样式外，一切在初始化框中设置的样式都不能生效，此时可以使用 cls 属性给展示值框添加新的风格，不过该风格只适用于展示值框，并不适用于文本框的标签。通常我们使用 标记作为文本框类组件的父容器，并在父容器中添加相关的风格。由于初始化框被隐藏，我们无法通过选择器来选中指定的文本框组件并为其绑定事件，因此文本框提供了 textbox 方法，该方法返回文本框中的展示值框对象，开发者可以为展示值框绑定相关的时间。
- parser 属性是将输入值转换成存储值，用户的输入值可能就是合法的存储值，也可能是展示值，还有可能是非法值，开发者在 parser 属性中一定要做充分的判断。

3. 数字框事件

数字框在文本框的基础上无新增事件。

4. 数字框方法

数字框常用方法的说明见表 2.10。

表 2.10　数字框常用方法说明

名称	参数	描述
options	none	返回选项对象
destroy	none	销毁数字框
disable	none	禁用数字框
enable	none	启用数字框
fix	none	把值固定为有效的值
setValue	value	设置数字框的存储值
getValue	none	获取数字框的存储值
clear	none	清除数字框的全部值
reset	none	重置数字框的全部值

5. 深入理解数字框的值

我们已经向读者讲解了 EasyUI 中的三个值，它们分别是输入值、展示值以及存储值，这里讲到数字框实际上由以下三部分组成：

- 初始化框：用于保存组件初始化的配置以及存储值。
- 存储值框：用于保存存储值。
- 展示值框：用于显示展示值，以及接收用户的输入值。

其中初始化框中保存的是初始化的配置以及存储值，一些 jQuery 开发者习惯使用例如$('#nn').val('11')的方法给数字框赋值，读者可以发现此这种方法其实只是给初始化框赋值，并不会改变数字框的值。但是通过$('#nn').val()方法可以取出数字框的存储值，这是因为存储值也会被保存在初始化框中。这样做的好处很多，例如在提交表单时，服务器端可以直接根据初始化框中的 name 属性获取数字框的存储值。

展示值框有两个作用，首先它接收用户的输入，也就是说输入值其实是被输入到展示值框中的；其次它向用户显示展示值，用户的输入值与展示值之间的转换在数字框中需要先使用 parser 属性方法将输入值转换成存储值，再使用 formatter 属性方法将存储值转化成展示值。

存储值框中会保存存储值，可以通过 getValue 方法获取其值。

接下来请读者思考自定义验证规则时到底是对数字框的哪个值进行验证呢？请看下面的代码：

```
01      <input type="text" id="nn" value="1">
02      <script>
03          $(function(){
04              $("#nn").numberbox({
```

```
                validType:"englishOrNum",
                prefix:'$',
            });
            //自定义验证规则，只能输入数字
            $.extend($.fn.validatebox.defaults.rules, {
                englishOrNum : {
                validator : function(value) {
                  return /^[0-9]{1,}$/.test(value);
                },
                message : '请输入数字'
                }
            });
        });
    </script>
```

最终运行结果如图 2.10 所示。

2.10 带验证的数字框

我们知道该示例中存储值为一个纯数字，而展示值为一个带美元符号前缀的数字，验证规则中自定义了一个验证用户输入是否为数字的规则。可以发现当用户输入一串数字后仍然无法通过验证，这是因为验证方法也会对展示值进行验证，如果我们希望仅仅验证存储值的话，那就必须在自定义验证规则中对展示值进行解析，如下代码所示。

```
$.extend($.fn.validatebox.defaults.rules, {
    englishOrNum : {
    validator : function(value) {
        value = $.trim(value.replace("$",""));//去除前缀
        return /^[0-9]{1,}$/.test(value);
    },
    message : '请输入数字'
    }
});
```

2.2 组合简介

2.2.1 组合（Combo）

组合是在页面上显示一个文本框和一个下拉面板，它是创建其他复杂组件（例如：combobox、combotree、combogrid）的基础，利用组合我们也可以自定义开发一些更加复杂的组件，例如在第 6 章中利用组合开发起止日期框组件。本节将向读者讲解组合和组合框两个组件的使用方法。

组合的依赖关系如下：

- textbox
- panel

组合扩展于：

- textbox

组合的默认配置定义在$.fn.combo.defaults 中。

1. 组合的用法

可以通过 JavaScript 从<input>或者<select>标记创建组合，注意使用标记创建组合是不合法的。例如：

```
<input id="cc" value="1">
$('#cc').combo({
    required:true,
    multiple:true
});
```

2. 组合属性

组合常用属性的说明见表 2.11。

表 2.11　组合常用属性说明

名称	类型	描述	默认值
width	number	组件的宽度	auto
height	number	组件的高度	22
panelWidth	number	下拉面板的宽度	null
panelHeight	number	下拉面板的高度	200
panelMinWidth	number	下拉面板的最小宽度	null
panelMaxWidth	number	下拉面板的最大宽度	null
panelMinHeight	number	下拉面板的最小高度	null
panelMaxHeight	number	下拉面板的最大高度	null
panelAlign	string	下拉面板的对齐方式，可能的值有 left 和 right	left
multiple	boolean	定义是否支持多选	false
multivalue	boolean	定义是否提交多个值	true
reversed	boolean	定义当输入框失去焦点时是否保存原始值	false
selectOnNavigation	boolean	定义是否可以通过键盘来选择选项	true
separator	string	多选时文本的分隔符	

（续表）

名称	类型	描述	默认值
editable	boolean	定义用户是否可以直接输入文本	true
disabled	boolean	定义是否禁用该字段	false
readonly	boolean	定义是否为只读	false
hasDownArrow	boolean	是否显示下拉的箭头按钮	true
value	number	文本框默认值	
delay	number	用户在文本框中输入某个字符后，面板会自动打开显示查找到的结果，delay 设置的就是用户输入完毕后到面板打开的延迟时间	200

keyHandler	object	当用户按下键盘上的按键后调用的函数。默认的 keyHandler 定义如下：
		keyHandler: {
		up: function(e){},
		down: function(e){},
		left: function(e){},
		right: function(e){},
		enter: function(e){},
		query: function(q,e){}
		}	

3. 组合的事件

组合常用事件的说明见表 2.12。

表 2.12　组合常用事件说明

名称	参数	描述
onShowPanel	None	显示下拉面板时触发
onHidePanel	None	隐藏下拉面板时触发
onChange	newValue,oldValue	字段值发生改变时触发

4. 组合的方法

组合常用方法的说明见表 2.13。

表 2.13　组合常用方法说明

名称	参数	描述
options	none	返回选项对象

（续表）

名称	参数	描述
panel	none	返回面板对象
textbox	none	返回文本框对象
destroy	none	销毁组件
resize	width	调整组件的宽度
showPanel	none	显示下拉面板
hidePanel	none	隐藏下拉面板
disable	none	禁用组件
enable	none	启用组件
readonly	mode	启用/禁用只读模式
validate	none	验证输入值
isValid	none	返回验证结果
clear	none	清空组件的全部类型值
reset	none	重置组件的全部类型值
getText	none	获取组件的展示值
setText	text	设置组件的展示值
getValues	none	获取组件的存储值的数组
setValues	values	设置组件的存储值的数组
getValue	none	获取组件的存储值
setValue	value	设置组件的存储值

5. 深入理解 EasyUI 依赖关系

通过前面的学习读者可以发现组合依赖于文本框和面板，其中扩展于文本框，因此我们可以使用文本框的属性来初始化组合，但是如果需要设置面板属性的话，我们需要通过组合的 panel 方法先获取面板对象再设置其属性，例如：

```
<body>
    <input id="cc" name="dept" value="aa">
    <div id="footer">底部</footer>
</body>
<script>
$(function(){
    $('#cc').combo({
        //文本框属性
        iconCls:'icon-search',
        onClickButton:function(){
            alert("11");
```

```
            },
            //组合属性
            hasDownArrow:false,
        });
        //设置面板属性
        var panel = $('#cc').combo("panel");
        panel.panel({
            footer:"#footer"
        });
    });
    </script>
```

最终运行结果如图 2.11 所示。

再看下面的代码，我们使用组合的 textbox 方法获取文本框对象，接着设置文本框属性，例如：

```
        var tb = $('#cc').combo("textbox");
        tb.textbox({
            width:300
        });
```

此时运行结果如图 2.12 所示。

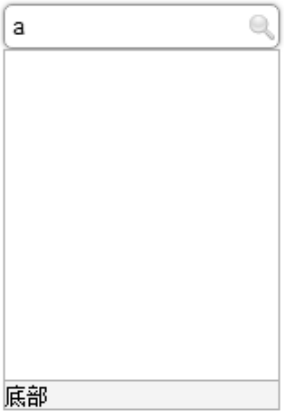

图 2.11　设置依赖组件属性

图 2.12　设置依赖组件属性

可以发现此时并没有设置成功组合中的文本框属性，相反这段代码在原先的组合上新增了一个文本框。这是因为组合本身并没有重写或新增 textbox 方法，组合使用的其实是文本框的 textbox 方法，该方法返回的是文本框中展示值框的对象，开发者可以为其绑定指定的事件。但是如果使用展示值框来初始化文本框，就会在其基础上创建一个新的文本框。

2.2.2　组合框（ComboBox）

组合框由一个可编辑的文本框和一个下拉面板组成。用户可以在下拉面板中选中一个或者多个值，同样可以直接在文本框内输入内容或者在下拉面板中选中一个或多个值。

组合框依赖关系如下：

- combo

组合框扩展于：

- combo

组合框的默认配置定义在$.fn.combobox.defaults 中。

1. 创建组合框

可以通过<select>标记创建一个组合框，此时可以将选项直接写入到<select>元素中。例如：

```
<select id="cc" class="easyui-combobox" name="dept" style="width:200px;">
    <option value="aa">item1</option>
    <option>item2</option>
    <option>item3</option>
    <option>item4</option>
    <option>item5</option>
</select>
```

也可以通过<input>标记创建组合框，例如：

```
<input id="cc" name="dept" value="aa">
$('#cc').combobox({
    url:'combobox_data.json',
    valueField:'id',
    textField:'text'
});
```

也可以创建两个相互依赖的组合框，例如：

```
<input id="cc1" class="easyui-combobox" data-options="
        valueField: 'id',
        textField: 'text',
        url: 'get_data1.php',
        onSelect: function(rec){
            var url = 'get_data2.php?id='+rec.id;
            $('#cc2').combobox('reload', url);
        }">
<input id="cc2" class="easyui-combobox"
data-options="valueField:'id',textField:'text'">
```

2. 组合框属性

组合框常用属性见表 2.14。

表 2.14　组合框常用属性说明

名称	类型	描述	默认值
valueField	string	设置存储值字段	value
textField	string	设置展示值字段	text

（续表）

名称	类型	描述	默认值
groupField	string	设置需要被分组的字段	null
groupFormatter	function(group)	设置分组文本的展示值	
mode	string	设置检索模式，设置为 remote 时从服务器加载数据，设置为 local 时从本地加载数据	local
url	string	提供初始化数据的服务器地址	null
method	string	定义何种方法向服务器传输参数	post
data	array	本地数据	null
queryParams	object	通过服务器加载数据时，向服务器传输的参数对象	{}
limitToList	boolean	限制用户的输入必须为下拉面板中的数据	false
showItemIcon	boolean	定义是否显示文本框右侧的下拉按钮	false
groupPosition	string	定义分组的位置，可能的值有 static 和 sticky	static
filter	function	当 mode 设置为本地加载数据时，filter 属性可以让开发者自定义接收到用户输入时如何显示下拉面板中的数据	
formatter	function	设置每一行数据的展示值	
loader	function(param,success,error)	用于从服务器端检索数据	json loader
loadFilter	function(data)	对服务器端检索后的数据进一步过滤	

下面将具体讲解如何使用组合框从服务器和本地加载数据，组合框加载数据时可以接收 JSON 格式数据，通过如下方法加载本地数据，部分代码如下：

```
$('#cc').combobox({
    valueField:'id',
    textField:'city',
    data:[
        {"id":1," country":"中国","city":"北京市"},
        {"id":2," country":"中国","city":"上海市"},
        {"id":3," country":"中国","city":"重庆市"},
        {"id":4," country":"中国","city":"天津市"},
        {"id":5," country":"美国","city":"华盛顿"},
        {"id":6," country":"美国","city":"纽约"},
        {"id":7," country":"美国","city":"旧金山"},
        {"id":8," country":"英国","city":"伦敦"},
        {"id":9," country":"英国","city":"伯明翰"},
        {"id":10," country":"英国","city":"利兹"},
        {"id":11," country":"法国","city":"巴黎"},
        {"id":12," country":"法国","city":"马赛"},
```

```
                    {"id":13," country":"法国","city":"里昂"},
                ],
            });
```

我们也可以通过服务器加载数据，服务器部分代码如下：

```
$city = array(
    array("id"=>1," country"=>"中国","city"=>"北京市"),
    array("id"=>2," country"=>"中国","city"=>"上海市"),
    array("id"=>3," country"=>"中国","city"=>"重庆市"),
    array("id"=>4," country"=>"中国","city"=>"天津市"),
    array("id"=>5," country"=>"美国","city"=>"华盛顿"),
    array("id"=>6," country"=>"美国","city"=>"纽约"),
    array("id"=>7," country"=>"美国","city"=>"旧金山"),
    array("id"=>8," country"=>"英国","city"=>"伦敦"),
    array("id"=>9," country"=>"英国","city"=>"伯明翰"),
    array("id"=>10," country"=>"英国","city"=>"利兹"),
    array("id"=>11," country"=>"法国","city"=>"巴黎"),
    array("id"=>12," country"=>"法国","city"=>"马赛"),
    array("id"=>13," country"=>"法国","city"=>"里昂")
);
echo JSON($city);
```

对应的客户端部分代码如下：

```
$('#cc').combobox({
    valueField:'id',
    textField:'city',
    url:" getData.php"
});
```

最终运行结果如图 2.13 所示。

图 2.13　使用组合框加载数据

通过本地加载数据时，需要给 data 属性赋予一个 JSON 格式的数据，使用数字 1 来存储数据要比使用“北京市”来存储数据更加节省磁盘空间，而且避免了编码问题，因此在这里使用 id 字段来表示存储值，使用 city 字段来表示展示值。组合框中的 valueField 属性指定存储值的字段，textField 属性指定展示值字段。通过服务器加载数据，其实就是将数据保存在服务器

端，其本质都是通过 JSON 格式数据来初始化组合框，url 属性提供组合框初始化数据的服务器地址。

在上述例子的 JSON 格式数据中有一个 country 字段，该字段表示每个城市所在的国家，组合框可以通过 groupField 属性对数据进行分组，使用 groupFormatter 属性设置各个分组的展示值，formatter 属性可以设置组合框中每一行数据的展示值。接下来我们将把数据按照国家进行分组，并将国家名格式化为该国家的国旗图标，然后再对各个国家首都进行加黑处理。部分代码如下：

```
$('#cc').combobox({
    //扩展自 Combo 的属性
    width: 400,
    panelHeight:450,
    //ComboBox 新增属性
    valueField:'id',
    textField:'city',
    groupField:'country',
    url:"server/getCountry.php",
    groupFormatter:function(group){
        if(group == "中国"){
            return "<img src='img/zg.png'></img>";
        }else if(group == "美国"){
            return "<img src='img/mg.png'></img>";
        }else if(group == "英国"){
            return "<img src='img/yg.png'></img>";
        }else if(group == "法国"){
            return "<img src='img/fg.png' width='36' height='27'></img>";
        }else{
            return "";
        }
    },
    formatter:function(row){
        var opts = $(this).combobox("options");
        var text = row[opts.textField];
        if(text == "北京市"||text == "华盛顿"||text == "伦敦"||text == "巴黎"){
            return "<b>"+text+"</b>";
        }
        else{
            return text;
        }
    },
});
```

在上述代码中 groupFormatter 属性的参数 group 代表的是各个分组的名称，例如中国、美国等，程序会自动检测有多少个分组，并将每个类型的分组都使用一次 groupFormatter 中的方法格式化。formatter 属性中的 row 参数代表初始化数据中的每一行数据，例如{"id":1,"country":"中国","city":"北京市"}，程序会对初始化数据使用 format 中的方法进行格式化，因此通过这个方法我们可以自定义组合框中每一行数据的展示格式。options 方法在前面的章节中曾经提及

过，它返回的是当前组件的配置，因此通过 row[opts.textField]可以获取组合框 textField 的值。最终运行结果如图 2.14 所示。

图 2.14　使用组合框对设计数据的展示值

【本节详细代码参见随书源码：　\源码\easyui\example\c2\comboboxFormat.html】

组合框是由可编辑的文本框和下拉面板组成的。用户同样可以在文本框内输入字符串来检索数据，因为当组合框中包含世界所有的国家的主要城市时，通过下拉面板一条条地查找数据显然效率低下，此时用户更希望在文本框内中输入需要查找的城市关键字，下拉面板自动将包含关键字的数据显示出来。mode 属性指定当用户在组合框内输入关键字时，组合框从何处获取数据，当设置其值为 remote 时，组合框会将用户输入的关键字通过 http 请求以参数 q 传输给服务器端，服务器端会将查询到的数据返回给组合框，mode 属性设置为 local 时，可以通过 filter 属性自定义规则在本地进行检索。服务器端检索数据较为复杂，接下来本书先讲解如何从本地检索数据，我们对上个例子进行修改，使用户输入国家时，下拉面板只显示该国家的城市，新增代码如下：

```
01       mode: "local",
02       filter:function(q,row)
03       {
04          var opts = $('#cc').combobox("options");//获取该组合框全部的属性
05          var groupname =row[opts.groupField];//获取该行数据的分组值
06          if(groupname == q){
07              return row[opts.textField];     //当用户输入的值等于分组值时则显示数据
08          }
09          else{
10              return false;
11          }
12       }
```

最终运行结果如图 2.15 所示。

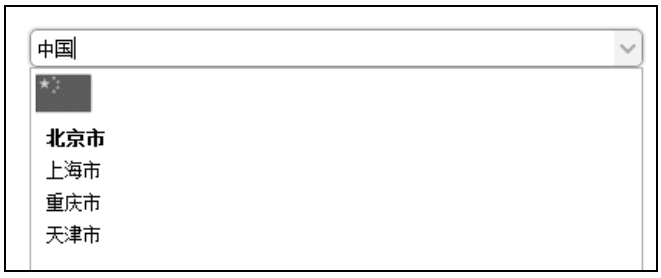

图 2.15　本地检索数据

filter 属性默认检索规则是显示包含用户输入的关键字的数据。例如输入“北京”，将会检索到“北京市”。

下面我们将探讨服务器端检索数据的方法，使用服务器端检索数据需要用到如下属性。

- mode：设置为 romote 时将从服务器端检索数据。
- loader：用于从服务器检索数据。
- loadFilter：对服务器端检索后的数据进一步过滤。

部分代码如下：

```
     mode: "remote",
     loader:function(param,success,error){
         var q = param.q;           //获取文本框中输入的数据
         if(q.length<1){
            return false;
         }
         $.ajax({                   //利用 ajax 请求获取数据
            url: ' filterCountry.php?q='+q,
            dataType: 'json',
            type:'get',
            success: function(data){
                success(data);
            },
            error:function(){
                    error();
                }
            });
     }
    loadFilter:function(data){
     //服务器端检索完毕后的回调函数，data 为服务器返回的值，
     //开发者可以进一步对数据进行过滤
    return data;
 }
```

【本节详细代码参见随书源码：\源码\easyui\example\c2\ comboboxFilter.html】

loader 属性为一个函数，其参数 param 为需要被传输到服务器的数据，通过 param.q 可以得到用户输入的关键字，然后使用 ajax 的 get 方法向服务器传输用户输入的关键字，服务器将检索后的结果通过 JSON 格式返回，ajax 传输成功后调用 loader 函数的 success 参数，该参数接收服务器返回的数据，最后会调用 loadFilter 属性中的方法，该方法可以对检索后的数据进一步过滤。服务器端代码如下：

```
$limit = $_GET['q'];
$result = db::select("select * from country where country = :country",array(
    "country"=>$limit
))->getResult();
echo Data::toJson($result);
```

运行上述程序后读者可以发现，使用服务器检索数据在组合框初始化时，并不会加载全部的数据。因此通过服务器检索数据更适合用于数据量极大时，用户并不希望显示全部的数据，仅仅希望查找自己感兴趣的数据的情况下使用。

到目前为止组合框还有两个属性没有讲解，一个为 method 属性，另一个为 queryParams 属性。在实际项目开发中，经常会遇到权限限制的问题，例如在网站上有很多个用户，每个用户下拥有多个项目，我们希望用户只能看到自己的项目。例如我们希望用户只能看到中国下的全部城市，此时我们会通过 queryParams 属性增加一些参数，这些参数将会在组合框初始化时传输到服务器，服务器根据这些参数即可限制显示给用户的数据范围，method 属性指定以何种方式发送这些参数，它可以是 get 或者 post 方法。如下代码限制用户仅能查看中国的城市：

```
    url:" getData.php",
    queryParams:{"c":'中国'},
    method: 'get'
```

服务器获取传输过来的参数并且进行处理，部分代码如下：

```
$limit = $_GET['c'];
$result = db::select("select * from country where country = :country",array(
    "country"=>$limit
))->getResult();
echo Data::toJson($result);
```

此时组合框内将仅仅显示中国范围内的城市。

loader 属性中的 param 参数是一个包含 queryParams 属性和用户输入值的对象，在上例中我们也可以通过 param.c 获取到 queryParams 属性中的限制参数，这种设计保证我们在向服务器检索数据时也可以限制在指定范围内检索。

看到这里读者可能会对组合框的属性感到繁杂，其实组合框一共有三大类属性，分别是：

（1）值类属性。例如 valueField 属性指定存储值字段，textField 属性指定展示值字段，groupField 属性指定分组值字段，由于 valueField、textField、groupField 属性仅仅是针对 JSON 格式内的数据，因此展示值只能是一段字符串，并不丰富。对此组合框提供了 groupFormatter

属性来将分组字段内容转换成任意的展示格式，提供 formatter 属性将展示值字段内容进一步格式化成任意形式。

（2）获取数据属性。组合框可以接受 JSON 格式数据，提供本地获取数据和服务器端获取数据两种形式。本地获取数据时直接设置 data 属性值即可，服务器端获取数据需要设置 url 属性为初始化数据的服务器地址。为了只显示指定范围内的数据可以在 queryParams 中增加一些限制参数，通过 method 属性指定这些参数以何种方法传输给服务器端。

（3）检索数据属性。组合框提供本地检索和服务器检索数据两种形式。本地检索数据时，只需要在 filter 属性中定义一系列的检索规则即可，本地检索数据在初始化时仍然会加载全部数据。通过服务器端检索数据时，需要设置 mode 属性值为 remote，在 loader 属性中向检索服务器发送 ajax 请求，接收到服务器返回的数据后会调用 loadFilter 属性对检索后的数据进一步过滤。

3. 组合框事件

组合框常用事件说明见表 2.15。

表 2.15　组合框常用事件说明

名称	参数	描述
onBeforeLoad	param	加载数据前触发的事件，然回 false 则取消加载
onLoadSuccess	none	加载数据成功时触发的事件
onLoadError	none	加载数据失败时触发的事件
onChange	newValue,oldValue	当文本内容发生改变时触发的事件
onClick	record	当用户单击下拉面板中的某条数据时触发的事件
onSelect	record	当用户选中下拉面板中的某条数据时触发的事件
onUnselect	record	当用户取消选中下拉面板中的某条数据时触发的事件

onBeforeLoad 是在数据加载前触发，其参数 param 表示此次数据加载请求中包含的全部参数，通常用在向服务器发送加载数据请求前检查请求的参数。

组合框事件的 record 参数为用户操作的行对象，包括用户操作某行数据的存储值、展示值以及分组值。

在 EasyUI 中很多参数都是对象，如果读者不清楚参数的具体含义，可以使用本书在 2.1.2 节中提供的 writeObj 函数打印对象。

4. 组合框方法

组合框常用方法说明见表 2.16。

表 2.16　组合框常用方法说明

名称	参数	描述
options	none	返回选项对象
getData	none	得到加载的数据
loadData	data	加载本地数据
reload	url	重新加载服务器数据
setValues	values	设置多个值
setValue	value	设置单个值
clear	none	清理全部的值
select	value	选中某个值
unselect	value	取消选中某个值

其中 setValues 方法需要在组合框设置为多选模式时才能选中多个值。设置组合框的值的含义是选中面板中指定的数据，并将选中数据的展示值显示在文本框中。

```
01          $('#cc').combobox("setValue","1");//选中单个数据
02          $('#cc').combobox("setValues","1,2");//选中多个数据
```

2.3 微调器简介

2.3.1 微调器（Spinner）

微调器由一个可编辑的文本框和两个微调按钮组成（下面称其为增量按钮和减量按钮），允许用户在一定范围内选择数据。与组合框一样，微调器也允许用户在文本框内输入值，但是它没有下拉面板。微调器是创建其他微调器组件（比如：数字微调（NumberSpinner）、时间微调器（TimeSpinner））的基础组件。

微调器的依赖关系如下：

- textbox

微调器扩展于：

- textbox

微调器的默认配置定义在$.fn.spinner.defaults 中。

1. 创建微调器

只可以通过 JavaScript 创建微调器，使用标记创建是不合法的。

```
<input id="ss" value="2">
$('#ss').spinner({
    required:true,
    increment:10
});
```

2. 微调器属性

微调器常用属性说明见表 2.17。

表 2.17　微调器的常用属性

名称	类型	描述	默认值
width	number	组件的宽度	auto
height	number	组件的高度	22
value	string	微调器的初始值	
min	string	微调器允许的最小值	null
max	string	微调器允许的最大值	null
increment	number	单击微调器按钮时的增量值	1
editable	boolean	定义是否可以直接向微调器中输入值	true
disabled	boolean	禁用微调器	false
readonly	boolean	是否开启只读模式	false
spinAlign	string	定义微调器按钮的对齐方式，可能的值有'left' 'right' 'horizontal'和'vertical'	right
spin	function(down)	当用户单击微调按钮时调用的函数。'down'参数表示用户单击了何种按钮，值为 true 表示用户单击了增量按钮，值为 false 时表示用户单击了减量按钮	

3. 微调器事件

微调器常用事件的说明见表 2.18。

表 2.18　微调器的常用事件

名称	参数	描述
onSpinUp	None	当单击向上微调按钮时触发
onSpinDown	None	当单击向下微调按钮时触发

4. 微调器方法

微调器常用方法的说明见表 2.19。

表 2.19　微调器的常用方法

名称	参数	描述
options	none	返回选项对象
destroy	none	销毁微调器组件
resize	width	调整微调器组件的尺寸
enable	none	启用微调器组件
disable	none	禁用微调器组件
getValue	none	获取组件的值
readonly	mode	启用/禁用只读模式
setValue	value	设置组件的值
clear	none	清理组件全部类型的值
reset	none	重置组件的全部类型的值

2.3.2　数字微调器（NumberSpinner）

数字微调器由数字框和微调器组成，可以将用户输入的数据转换成不同的展示格式，例如数字、百分比、货币等。它允许用户使用向上/向下微调按钮滚动到一个期望值。数字微调器的使用与数字框一样，不同的是开发者可以通过增量按钮或减量按钮来限制每次数值的变化量。

数字微调器的依赖关系如下：

- spinner
- numberbox

数字微调器扩展于：

- spinner

数字微调器的默认配置定义在$.fn.numberspinner.defaults 中。

1. 创建数字微调器

使用标记创建数字微调器的方法如下：

```
<input id="ss" class="easyui-numberspinner" >
```

使用 JavaScript 创建数字微调器的方法如下：

```
01      $('#ss').numberspinner({
02          min: 10,
03          max: 100,
04          editable: false
05  });
```

2. 数字微调器属性

数字微调器本身无重写/新增属性。

3. 数字微调器事件

数字微调器本身无重写/新增事件。

4. 数字微调器方法

数字微调器常用方法说明见表 2.20。

表 2.20　数字微调器的常用方法

名称	参数	描述
options	none	返回方法对象
setValue	value	设置数字微调器的值

2.3.3　时间微调器（TimeSpinner）

时间微调器是基于微调器的一个组件，与数字微调器一样。不过时间微调器可以用来显示时间，用户可以通过单击时间微调器右侧的增量或者减量按钮来调整时间。

时间微调器的依赖关系如下：

- spinner

时间微调器扩展于：

- spinner

时间微调器的默认配置定义在$.fn.timespinner.defaults 中。

1. 创建时间微调器

使用标记创建时间微调器的方法如下：

```
<input id="ss" class="easyui-timespinner"  style="width:80px;"
       required="required" data-options="min:'06:30',showSeconds:true">
```

使用 JavaScript 创建时间微调器的方法如下：

```
<input id="ss" style="width:80px;">
    $('#ss').timespinner({
        min: '06:30',
        required: true,
        showSeconds: true
});
```

2. 时间微调器的属性

时间微调器常用属性说明见表 2.21。

表 2.21　时间微调器的常用属性

名称	类型	描述	默认值
separator	string	时分秒之间的分隔符	:
showSeconds	boolean	是否精确显示到秒	false
highlight	number	初始化时高亮的字段，值 1 代表小时、2 代表分钟、3 代表秒	
formatter	function(data)	将存储值格式化成展示值	
parser	function(s)	将初始值或输入值解析成存储值	
selections	array	该属性用于设置组件高亮的部分，例如[[0,2]、[3,5]、[6,8]]	

3. 时间微调器的事件

时间微调器无重写/新增事件。

4. 时间微调器的方法

时间微调器常用方法说明见表 2.22。

表 2.22　时间微调器的常用方法

名称	参数	描述
options	none	返回选项对象
setValue	value	设置存储值
getHours	none	获取小时数据
getMinutes	none	获取分钟数据
getSeconds	none	获取秒数据

2.3.4　日期微调器（DateTimeSpinner）

日期微调器由时间微调器扩展而来，不仅可以微调时间还可以微调日期。
日期微调器的依赖关系如下：

- timespinner

日期微调器扩展于：

- timespinner

日期微调器的默认配置定义在$.fn.datetimespinner.defaults 中。

1. 创建日期微调器

使用标记创建日期微调器的方法如下：

```
<input class="easyui-datetimespinner" style="width:300px">
```

使用 JavaScript 创建创建日期微调器的方法如下：

```
<input id="dt" type="text" style="width:300px">
$('#dt').datetimespinner({
    //...
})
```

2. 日期微调器属性

日期微调器重写了时间微调器的 selections 属性，可以通过为其设置一个数组值来调整日期微调器的高亮部分，例如：[[0,2],[3,5],[6,10],[11,13],[14,16],[17,19]]。

3. 日期微调器事件

日期微调器无重写/新增事件。

4. 日期微调器方法

日期微调器无重写/新增方法。

2.4 菜单和按钮

2.4.1 菜单（Menu）

菜单由菜单域与菜单元素组成，一个菜单域内有多个菜单元素，每一个菜单元素都拥有相关的属性，而菜单域也拥有属性、事件以及方法，菜单的创建较为麻烦，但是使用却很简单，本节将先介绍菜单元素的用法再详细介绍菜单域的用法。

菜单的默认配置定义在$.fn.menu.defaults 中。

1. 创建菜单

第一步需要创建一个菜单区域，并根据页面需要设置菜单区域的宽度，我们可以通过在 class 中增加 easyui-menu 属性来创建菜单区域，代码如下：

```
<div  class="easyui-menu" style="width:150px;" id='mm'>
菜单区域…
</div>
```

第二步需要设置菜单元素，可以通过在菜单域中添加 标签来创建菜单元素，代码如下：

```
    <div  class="easyui-menu" style="width:150px;">
        <div>菜单元素 1</div>
        <div>菜单元素 2</div>
        <div>菜单元素 3</div>
```

```
05      </div>
```

第三步需要通过 JavaScript 初始化菜单域，代码如下：

```
    $('#mm').menu();
```

第四步是显示菜单区域，菜单区域创建完成后默认是不显示的，开发人员需要通过如下代码显示菜单：

```
01  $('#mm').menu('show', {
02      left: 200,
03      top: 100
04  });
```

2. 菜单元素

菜单元素的属性见表 2.23。

表 2.23　菜单元素的属性

名称	类型	描述	默认值
id	string	菜单元素的 id 属性	
text	string	菜单元素显示的文本	
iconCls	string	在菜单元素左侧显示一个 16×16 的图标	
href	string	当单击菜单元素时跳转的页面	
disabled	boolean	定义是否显示菜单元素	false
onclick	function	单击菜单元素时被调用的函数	

我们可以给菜单元素添加一个图标和单击事件，代码如下：

```
<div data-options="iconCls:'icon-save'" onclick="alert('1111');">保存</div>
```

菜单元素本身也可以作为一个菜单域，可以在菜单元素中添加新的菜单元素，此时就可以创建二级菜单，如下代码所示。

```
01      <div id="mm" class="easyui-menu" >
02          <div>菜单元素一</div>
03          <div>
04              <span>二级菜单</span>
05              <div >
06                  <div>二级菜单元素 1</div>
07                  <div>二级菜单元素 2</div>
08              </div>
09          </div>
10          <div>菜单元素二</div>
11          <div>菜单元素三</div>
12      </div>
```

由于每一个菜单元素都可以作为一个菜单域，因此我们可以根据需要创建任意多级菜单。

菜单元素也可以作为一个菜单分隔符，例如下面的代码可以创建一个菜单分隔符：

```
<div class="menu-sep"></div>
```

3. 菜单域的属性

菜单域常用属性的说明见表 2.24。

表 2.24　菜单域常用属性

名称	类型	描述	默认值
zIndex	number	菜单域的堆叠顺序值，堆叠顺序高的元素总是会处于堆叠顺序较低的元素的前面	110000
left	number	菜单域距离左侧的位置	0
top	number	菜单域距离顶部的位置	0
align	string	定义菜单的对齐方式可能是'left'和'right'	left
minWidth	number	菜单域的最小宽度	120
itemHeight	number	菜单元素的高度	22
duration	number	定义鼠标离开菜单域后多少毫秒后隐藏菜单域	100
hideOnUnhover	boolean	定义鼠标离开菜单域后是否自动隐藏菜单域	true
inline	boolean	定义是否设置为内联菜单	false
fit	boolean	定义是否自动适应其父元素尺寸	false

inline 属性设置菜单是否为内联菜单。所谓的内联，就是指显示位置是否以其父元素的起点为起点，例如：

```
<style>
    #border{
            border-style: solid;
            border-width: 1px;
            height: 100px;
            width: 200px;
            margin-left:500px;
            margin-top:200px;
        }
    </style>
    <div id="border">
        <div id="mm" class="easyui-menu" >
            <div>菜单元素一</div>>
            <div>菜单元素二</div>
            <div>菜单元素三</div>
        </div>
    </div>
    <script>
    $('#mm').menu({
        inline:true
```

```
21      });
22      $('#mm').menu('show', {
23          left: 200,
24          top: 100
25      });
26      </script>
```

最终运行结果如图 2.16 所示。

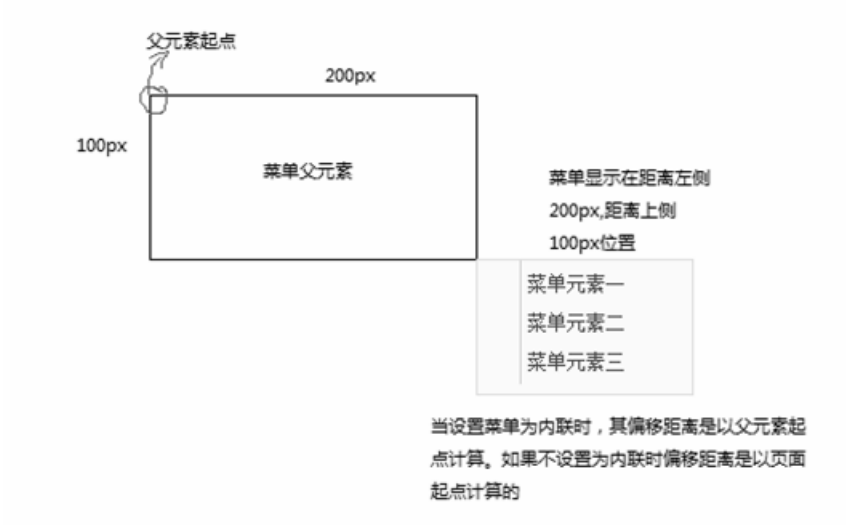

图 2.16　内联菜单

4. 菜单域的事件

菜单域常用事件的说明见表 2.25。

表 2.25　菜单域常用事件

名称	参数	描述
onShow	none	当菜单显示时触发
onHide	none	当菜单隐藏时触发
onClick	item	当菜单元素被单击时触发

当菜单域内有菜单元素被单击时会触发 onClick 事件。onClick 事件参数 item 为被单击的菜单元素对象，可以通过 item.name 或者 item.id 来获取被单击元素的 name 和 id 属性。例如，下面的代码当菜单元素一被单击时会显示该菜单元素名称，开发者可以使用该事件为菜单域内的各类菜单元素添加不同的处理逻辑：

```
01      <div id="mm" class="easyui-menu" >
02          <div name='菜单元素 1'>菜单元素一</div>
03          <div>菜单元素二</div>
04          <div>菜单元素三</div>
05      </div>
```

```
<script>
$('#mm').menu({
    onClick:function(item){
        alert(item.name);
    }
});
$('#mm').menu('show', {
    left: 200,
    top: 100
});
</script>
```

5. 菜单域的方法

菜单域常用方法的说明见表 2.26。

表 2.26　菜单域常用方法

名称	参数	描述
options	none	返回选项对象
show	pos	显示菜单域
hide	none	隐藏菜单域
destroy	none	销毁菜单域
getItem	itemEl	获取指定菜单元素包含 target 的属性
setText	param	设置指定菜单元素的文本
setIcon	param	设置指定菜单元素的图标
findItem	text	通过菜单元素的文本获取菜单元素对象
appendItem	options	在菜单域内添加一个新的菜单元素
removeItem	itemEl	移除指定的菜单元素
enableItem	itemEl	使用指定的菜单元素
disableItem	itemEl	禁用指定的菜单元素
showItem	itemEl	显示指定的菜单元素
hideItem	itemEl	隐藏指定的菜单元素
resize	menuEl	调整菜单域的大小

掌握菜单域的使用方法前，读者首先应该了解 itemEl 参数的含义，它指的是一个菜单元素对象，我们可以通过如下方法获取菜单元素对象：

```
<div id="mm" class="easyui-menu" >
    <div id='m1'>菜单元素一</div>
    <div>菜单元素二</div>
    <div>菜单元素三</div>
</div>
```

```
    <script>
        var itemEl = $('#m1');  //获取菜单元素一对象
    </script>
```

findItem 可以根据菜单元素名称获取菜单元素对象。

setText 可以更改指定的菜单元素名称。

setText 的参数 param 包含两个属性，第一个为 target 属性，它是指定的菜单元素的 DOM 对象；另一个为 text 属性，它是更改后的菜单元素名称，例如我们可以重新设置菜单元素的名称，部分代码如下：

```
    var item = $('#mm').menu('findItem', '菜单元素一');
    $('#mm').menu('setText', {
        target: item.target,
        text: '菜单元素新'
    });
```

setIcon 的参数 param 包含两个属性，第一个为 target 属性，它是指定的菜单元素的 DOM 对象，另一个为 iconCls 属性，它为指定图标的类型。我们可以使用该方法设置菜单元素的图标，部分代码如下：

```
$('#mm').menu('setIcon', {
    target: $('#mm').menu('findItem', '菜单元素一'),
    iconCls: 'icon-closed'
});
```

2.4.2 链接按钮（LinkButton）

链接按钮由一个<a>标签表示，它可以同时显示文本和图标，也可以只显示其中的一个，链接按钮的宽度会根据文本的长度自动适应。链接按钮可以直接跳转页面，也可以通过程序获取其单击事件后进行相关逻辑处理。

链接按钮的默认配置定义在$.fn.linkbutton.defaults 中。

1. 创建链接按钮

使用标记创建链接按钮的方法如下：

```
<a  id="btn"  href="#"  class="easyui-linkbutton">链接按钮</a>
```

通过 JavaScript 创建链接按钮的方法如下：

```
<a id="btn" href="#">链接按钮</a>
$('#btn').linkbutton();
```

2. 链接按钮属性

链接按钮常用属性的说明见表 2.27。

表 2.27　链接按钮常用属性

名称	类型	描述	默认
width	number	该组件的宽度	null
height	number	该组件的高度	false
id	string	该组件的 id 属性	false
disabled	boolean	定义是否禁用按钮	false
toggle	boolean	定义是否允许用户切换按钮的选中状态	false
selected	boolean	定义按钮的状态是否为选中状态	false
group	string	按钮所属的分组名称	null
plain	boolean	是否隐藏按钮边界	false
text	string	按钮上的文本	‘’
iconCls	string	按钮上的图标	null
iconAlign	string	图标的对齐方式，可能的值有'left' 'right' 'top'和'bottom'	left
size	string	按钮的尺寸，可能的值有'small'和'large'	small

链接按钮属性的详细介绍如图 2.17 所示。

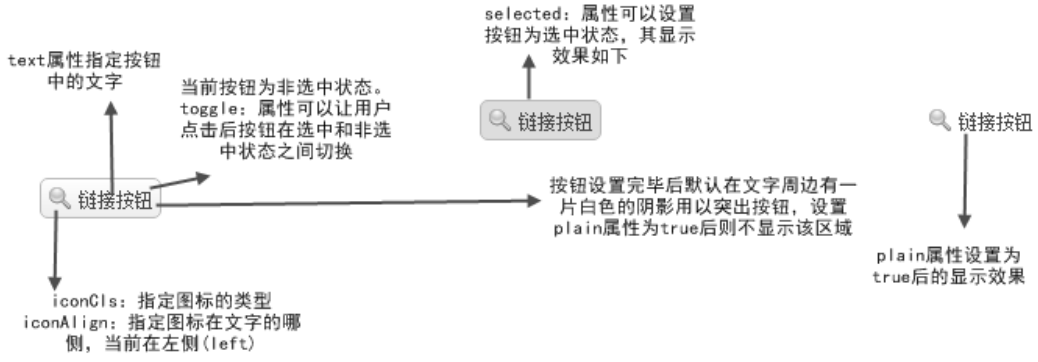

图 2.17　链接按钮属性介绍

3. 链接按钮事件

链接按钮常用事件说明见表 2.28。

表 2.28　链接按钮常用事件

名称	参数	描述
onClick	none	单击按钮后的事件

4. 链接按钮方法

链接按钮常用方法说明见表 2.29。

表 2.29　链接按钮常用方法

名称	参数	描述
options	none	返回选项对象
resize	param	调整按钮的尺寸
disable	none	禁用按钮
enable	none	启用按钮
select	none	设置按钮为选中状态
unselect	none	设置按钮为非选中状态

链接按钮方法的使用非常简单，如下代码可以调整链接按钮的尺寸：

```
$('#btn').linkbutton('resize', {
    width: '100%',
    height: 32
});
```

5. 处理链接按钮上的单击事件

链接按钮的主要作用就是在其被单击后可以处理一系列的事务，有些链接按钮被单击后需要直接跳转到一个新的页面，例如：

```
<a id="btn" href="test.html" class="easyui-linkbutton">链接按钮</a>
```

大部分的链接按钮在被单击后需要处理一系列的逻辑，例如表单中的提交按钮被单击后，需要判断用户的输入是否验证通过，此时我们需要通过程序来获取单击事件，例如：

```
    $('#btn').click(function(){
        alert('easyui');
    });
```

我们也可以使用链接按钮提供的单击事件方法，例如：

```
        $('#btn').linkbutton({
            iconCls: 'icon-search',
            onClick:function(){
                alert('easyui');
            }
        });
```

6. 链接按钮组

我们通常会将一组关联的链接按钮分到一组中，当链接按钮被分组后，用户每次只能选中链接按钮组中的一个按钮，例如：

```
    <div>
```

```
        <a id="btn1" href="#">链接按钮 1</a>
        <a id="btn3" href="#">链接按钮 2</a>
        <a id="btn2" href="#">链接按钮 3</a>
    </div>
    <script>
    $(function(){
        $('#btn1').linkbutton({
            toggle:true,
            group:"btn-group"
        });
        $('#btn2').linkbutton({
            toggle:true,
            group:"btn-group"
        });
        $('#btn3').linkbutton({
            toggle:true,
            group:"btn-group"
        });
    });
    </script>
```

最终运行结果如图 2.18 所示。

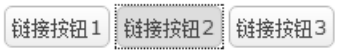

图 2.18　链接按钮组

2.4.3　菜单按钮（MenuButton）

上一节我们讲解了链接按钮的使用方法。链接按钮是一个基础按钮，它用于被单击后处理一系列的事务，但是在实际开发中特别是在列表中往往需要用到大量的链接按钮，例如某个页面显示 10 条用户的基本信息，每个用户的基本信息后面都需要有查看、编辑、删除这三个链接按钮，那么整个页面至少需要用到 30 个链接按钮。使用菜单按钮可以减少页面实际的按钮数量，它仅显示一个链接按钮，当链接按钮被单击后会在其下方显示一个菜单。

菜单按钮的依赖关系如下：

- linkbutton
- menu

菜单按钮扩展于：

- linkbutton

菜单按钮的默认配置定义在$.fn.menubutton.defaults 中。

1. 创建菜单按钮

创建菜单按钮时需要创建一个菜单按钮和一个菜单，并且设置菜单按钮的 menu 属性为菜单的一个选择器。

所谓的选择器就是如何选中某个元素，可以通过元素的 id 来选择此时的写法为#xx，也可以根据元素的 class 来选择写法为.xx。还有一些高级的选择器就不在这里介绍了，详细情况请查看 jQuery 选择器。

通过标记创建菜单按钮的方法如下：

```
<a href="JavaScript:void(0)" id="mb" class="easyui-menubutton"
       data-options="menu:'#mm',iconCls:'icon-edit'">编辑</a>
<div id="mm" style="width:150px;">
    <div>详情</div>
    <div>修改</div>
    <div>删除</div>
</div>
```

通过 JavaScript 创建菜单按钮的方法如下：

```
    <a href="JavaScript:void(0)" id="mb" >编辑</a>
    <div id="mm" style="width:150px;">
        <div>详情</div>
        <div>修改</div>
        <div>删除</div>
    </div>
    <script>
    $(function(){
        $('#mb').menubutton({
            iconCls: 'icon-edit',
            menu:'#mm'
        });
    });
    </script>
```

此时我们就创建了一个菜单按钮，当将鼠标放到编辑按钮上时，会在按钮下方显示一个菜单，运行效果如图 2.19 所示。

图 2.19　菜单按钮显示效果

2. 菜单按钮属性

菜单按钮常用的属性见表 2.30。

表 2.30　菜单按钮的常用属性

名称	类型	描述	默认值
plain	boolean	是否隐藏按钮边界	true
menu	string	菜单选择器	null
menuAlign	string	菜单的对齐方式，可能的值有'left'和'right'	left
duration	number	鼠标放到菜单按钮上时，显示菜单的延迟时间，单位为毫秒	100
hasDownArrow	boolean	菜单按钮右侧是否显示下拉图标	true

3. 菜单按钮事件

菜单按钮本身无新增和重写事件。

4. 菜单按钮方法

菜单按钮常用方法见表 2.31。

表 2.31　菜单按钮的常用方法

名称	参数	描述
options	none	返回选项对象
disable	none	禁用菜单按钮
enable	none	启用菜单按钮
destroy	none	销毁菜单按钮

2.4.4　分割按钮（SplitButton）

分割按钮与菜单按钮一样都是由链接按钮和菜单组成的，不同的是当鼠标移动到菜单按钮任意位置上时会显示一个菜单，只有当鼠标移动到分割按钮右侧的图标上时才会显示菜单。也就是说，使用分割按钮可以直接单击处理事务而不需要显示菜单。例如，我们上一节展示的一个菜单按钮，它有详情、修改、删除这三个功能，通过编辑文本来提示用户单击并显示菜单，但是对于用户来说详情功能用得最多，而修改、删除功能极少使用，这时我们更希望菜单按钮单击后，可以直接跳转到详情页面而不是显示菜单，分割按钮就是为了这一问题而设计的。

分割按钮的依赖关系如下：

- menubutton

分割按钮扩展于：

- menubutton

分割按钮的默认配置定义在$.fn.splitbutton.defaults 中。

1. 创建分割按钮

通过标记创建分割按钮的方法如下：

```
<a href="JavaScript:void(0)" id="mb" class="easyui-splitbutton"
      data-options="menu:'#mm'">详情</a>
<div id="mm" style="width:150px;">
    <div>详情</div>
    <div>修改</div>
    <div>删除</div>
</div>
```

通过 JavaScript 创建分割按钮的方法如下：

```
    <a href="JavaScript:void(0)" id="mb" >编辑</a>
    <div id="mm" style="width:150px;">
        <div>详情</div>
        <div>修改</div>
        <div>删除</div>
    </div>
    <script>
    $(function(){
        $('#mb').splitbutton({
            menu:'#mm',,
            onClick:function(){
                //处理详情功能业务
            }
        });
    });
    </script>
```

2. 分割按钮属性

分割按钮常用属性见表 2.32。

表 2.32　分割按钮的常用属性

名称	类型	描述	默认值
plain	boolean	是否隐藏按钮边界	true
menu	string	菜单选择器	null
menuAlign	string	菜单的对齐方式，可能的值有'left'和'right'	left
duration	number	鼠标放到分割按钮右侧下拉图标上时，显示菜单的延迟时间，单位为毫秒	100

3. 分割按钮事件

分割按钮本身无新增和重写事件。

4. 分割按钮方法

分割按钮常用方法见表 2.33。

表 2.33　分割按钮的常用方法

名称	参数	描述
options	none	返回选项对象
disable	none	禁用分割按钮
enable	none	启用分割按钮
destroy	none	销毁分割按钮

2.4.5　切换按钮（SwitchButton）

切换按钮通常用来提供两种状态的切换，例如开启/关闭状态，用户可以通过单击切换按钮的滑块来切换状态。

切换按钮的默认配置定义在$.fn.switchbutton.defaults 中。

1. 创建切换按钮

通过标记创建切换按钮的方法如下：

```
<input class="easyui-switchbutton" >
```

通过 JavaScript 创建切换按钮的方法如下：

```
<input id="sb" >
<script type="text/javascript">
    $(function(){
        $('#sb').switchbutton({
            checked: true,
        })
    })
</script>
```

2. 切换按钮属性

切换按钮常用属性说明见表 2.34。

表 2.34　切换按钮常用属性说明

名称	类型	描述	默认值
width	number	切换按钮宽度	60
height	number	切换按钮高度	26
handleWidth	number	按钮中滑块的宽度	auto
checked	boolean	定义按钮是否被选择	false
disabled	boolean	禁用按钮	false
readonly	boolean	定义按钮是否为只读模式	false
reversed	boolean	定义为 true 时将会调换开启和关闭状态的位置	false

（续表）

名称	类型	描述	默认值
onText	string	滑块左侧的文本	ON
offText	string	滑块右侧的文本	OFF
handleText	string	滑块上的文本	''
value	string	给切换按钮绑定一个值	on

3. 切换按钮事件

切换按钮常用事件说明见表 2.35。

表 2.35　切换按钮常用事件说明

名称	参数	描述
onChange	checked	当滑块选中的值发生改变时触发

4. 切换按钮方法

切换按钮常用方法说明见表 2.36。

表 2.36　切换按钮常用方法说明

名称	参数	描述
options	none	返回选项对象
resize	param	调整切换按钮的尺寸
disable	none	禁用切换按钮
enable	none	启用切换按钮
readonly	mode	开启/关闭只读模式
check	none	选中切换按钮
uncheck	none	取消选中切换按钮
clear	none	清除切换按钮的选中值
reset	none	重置切换按钮的选中值
setValue	value	设置切换按钮值

2.5 快速输入日期

在之前的章节中已经简单介绍了部分 EasyUI 中的日期控件，如时间微调器、日期微调器等。本节将向读者介绍 EasyUI 中功能更加强大的日期控件，使用这些日期控件非常简单，但是如果要完全掌握它们却非常困难。在学习本节的过程中读者应该牢牢把握住存储值和展示值这一核心概念。

2.5.1　日历（Calendar）

在前面的章节中我们学习了日期微调器控件，这个控件尽管也可以设置日期，但是使用起来却比较麻烦，例如无法快速地选择一个日期。日历提供了一个可供用户单击选择日期的界面，用户可以使用日历控件快速选择日期，日历控件默认星期天为每周的第一天。

日历的默认配置定义在$.fn.calendar.defaults 中。

1. 创建日历

使用标记创建日历的方法如下：

```
<div id="cc" class="easyui-calendar" style="width:180px;height:180px;"></div>
```

使用 JavaScript 创建日历的方法如下：

```
<div id="cc" style="width:180px;height:180px;"></div>
$('#cc').calendar({
    current:new Date()
});
```

2. 日历属性

日历常用属性的说明见表 2.37。

表 2.37　日历的常用属性

名称	类型	描述	默认
width	number	日历组件的宽度	180
height	number	日历组件的高度	180
fit	boolean	日历的尺寸是否自动适应其父元素	false
border	boolean	定义是否显示日历的边界	true
showWeek	boolean	定义是否显示星期数	false
weekNumberHeader	string	显示在星期数头部的标签	
getWeekNumber	function(date)	这个函数返回当前的星期数	
firstDay	number	定义每周的第一天。周末为 0，周一为 1…..	0
weeks	array	显示的星期列表	['S','M','T','W','T','F','S']
months	array	显示的月份列表	['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
year	number	设置日历当前显示的年份	当前年份（四位数表示）
month	number	设置日历当前显示的月份	当前月份，从 1 开始

（续表）

名称	类型	描述	默认
current	Date	设置日历当前显示的日期	当前日期
formatter	function(date)	设置日历中每日的展示值	
styler	function(date)	为日历中的展示值设置一个风格	
validator	function(date)	验证用户单击的日期是否可以被选择，返回 false 可以禁止用户选择该日期	

在详细讲解日历的属性前，先简单介绍 JavaScript 中的 Date 对象。Date 对象是 JavaScript 语言中内置的数据类型，用于提供日期和时间的操作接口。Date 类型使用自 UTC 1970 年 1 月 1 日 0 点开始经过的毫秒数来保存日期。Date 默认输出值格式为：

Tue Feb 13 2018 09:33:27 GMT+0800 (中国标准时间)

Date 对象的常用方法如下：

- 创建 Date 对象：var date = new Date()
- 从 Date 对象以四位数字返回年份：date.getFullYear()
- 从 Date 对象返回月份(0~11)：date.getMonth()
- 从 Date 对象返回一个月中的某一天(1~31)：date.getDate()
- 从 Date 对象返回一周中的某一天(0~6)：date.getDay()
- 返回 Date 对象的小时(0~23)：date.getHours()
- 返回 Date 对象的分钟(0~59)：date.getMinutes()
- 返回 Date 对象的秒数(0~59)：date.getSeconds()
- 返回 Date 对象的毫秒(0~999)：date.getMilliseconds()
- 返回 1970 年 1 月 1 日至今的毫秒数：date.getTime()
- 返回 1970 年 1 月 1 日午夜到指定日期（字符串）的毫秒数：date.parse()

Data 对象是以毫秒数进行创建的，它以毫秒数返回当前的日期，但是一些后台语言（如 PHP 等）是通过秒数创建日期对象的，读者要注意两者之间的转换。Date 对象也可以通过日期格式进行创建，如 new Date('2018/1/1')，但是无法使用中文格式进行创建，例如 new Date('2018 年 1 月 1 日')是非法的。

日历默认以英文符号来标注月份以及星期，下面我们将对日历控件进行一次汉化，并演示日历属性的用法，部分代码如下：

```
01      $('#cc').calendar({
02          /*初始化为当前日期，Date 对象可以通过毫秒数以及日期格式进行创建，如果不填写
03  创建条件的话默认以当前日期进行创建*/
04          current:new Date(),
05          width:400,//日历控件宽度
06          height:300,//日历控件的高度
```

```
07          showWeek:true,//在日历控件的最左侧显示当前星期是当年的第几个星期
08          weekNumberHeader:"星期数",
09          firstDay:"1",//设置星期一为每周的第一天
10          months:['1 月', '2 月', '3 月', '4 月', '5 月','6 月','7 月','8 月','9 月','10
月','11 月','12 月'],
11          //使用中文标注月份
12          weeks:['周日','周一','周二','周三','周四','周五','周六'],
13          //设置每一天的展示值
14          formatter:function(date){
15              return date.getDate()+"日";
16          },
17          //为展示值添加一个风格
18          styler:function(date){
19              if (date.getDay() == 6){
20                  return 'color:red';//将日历上周六的日期设置成红色
21              }
22              if (date.getDay() == 0){
23                  return 'color:blue';//将日历上周日的日期设置成蓝色
24              }
25          }
26      });
```

最终运行结果如图 2.20 所示。

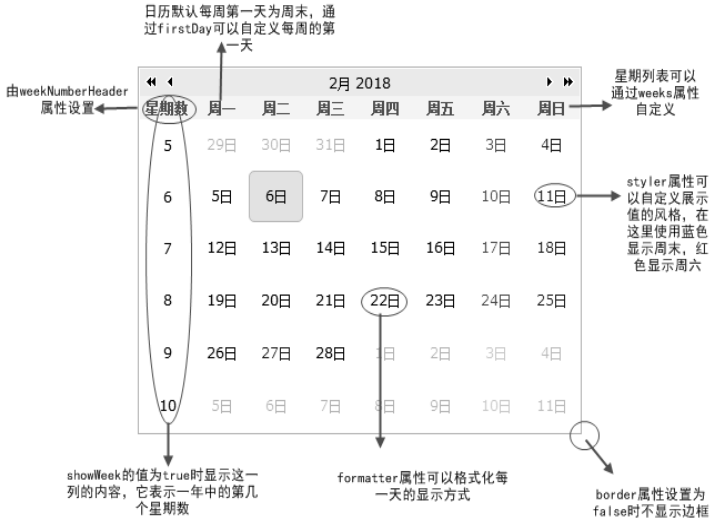

图 2.20　日历演示

3. 日历事件

日历常用事件说明见表 2.38。

表 2.38　日历的常用事件说明

名称	参数	描述
onSelect	Date	用户选择一个日期时触发
onChange	newDate, oldDate	日期改变时触发

4. 日历方法

日历常用方法说明见表 2.39 所示。

表 2.39　日历的常用方法说明

名称	参数	描述
options	none	返回日历选项对象
resize	none	调整日历尺寸
moveTo	date	指定日历选中的日期

2.5.2　日期框（DateBox）

日期框由组合和日历组成，用户在编辑框中输入的字符串被转换成有效的日期显示在日历面板上，在日历上选中的日期也将被转换成字符串显示在编辑文本框内。

日期框的依赖关系如下：

- combo
- calendar

日期框扩展于：

- combo

日期框的默认配置定义在$.fn.datebox.defaults 中。

1. 日期框的用法

使用标记创建日期框的方法如下：

```
<input id="db" type="text" class="easyui-datebox" required="required">
```

使用 JavaScript 创建日期框的方法如下：

```
01      <input id="db" type="text">
02      $('#db').datebox({
03          required:true
04       });
```

2. 日期框属性

日期框常用属性的说明见表 2.40。

表 2.40　日期框常用属性

名称	类型	描述	默认值
panelWidth	number	下拉日历面板的宽度	180
panelHeight	number	下拉日历面板的高度	auto
currentText	string	当前日期按钮上显示的文本	Today
closeText	string	关闭按钮上显示的文本	Close
okText	string	确定按钮上显示的文本	Ok
disabled	boolean	设置为 true 时禁用日期框	false
buttons	array	日历面板下的按钮	
sharedCalendar	string,selector	多个日期框组件使用的共享日历	
formatter	function	格式化日期的函数，并返回一个格式化的字符串值	
parser	function	解析日期字符串的函数，返回一个日期值	

下面我们先创建一个日期框，并将日期框的按钮进行相应汉化，部分代码如下：

```
<input id="db" type="text" class="easyui-datebox" required="required">
    <script>
        $(function(){
            $('#db').datebox({
                currentText:"今天",
                closeText:"关闭",
            });
        });
    </script>
```

最终运行结果如图 2.21 所示。

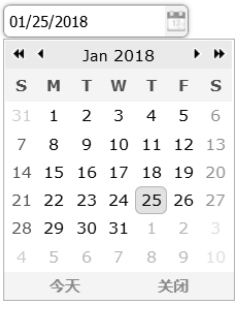

图 2.21　日期框演示

此时我们已经将按钮的默认名称改为对应的中文名称。EasyUI 允许开发者自由地改变按钮数量以及按钮的消息响应（消息响应简单的来说就是按下该按钮后所运行的一段程序，一般使用 handler 函数来处理，开发者可以在 handler 函数体内编写相应的响应程序），下面我们在“今天”和“关闭”两个按钮之间添加一个“确定”按钮，部分代码如下：

```
<input id="db" type="text" class="easyui-datebox" required="required">
<script>
    $(function(){
        $('#db').datebox({
            currentText:"今天",
            closeText:"关闭",
        });
        var buttons = $.extend([], $.fn.datebox.defaults.buttons);
        buttons.splice(1, 0, {
            text: '确定',
            handler: function(target){
            alert('当前选择的日期是'+$(target).val());
            }
        });
        $('#db').datebox({
            buttons: buttons
        });
    });
</script>
```

该例我们创建了一个带“确定”按钮的日期框，并且在单击“确定”按钮时显示当前选择的时间，最终运行结果如图 2.22 所示。

图 2.22　设置日期框按钮

对按钮的操作在实际开发中十分常用，上述改变日期框按钮的操作是一个十分通用的方法，下面将对上述代码进行详细解析。

先看这段代码：

```
var buttons = $.extend([], $.fn.datebox.defaults.buttons);
```

这段代码的含义是获取日期框控件的默认按钮对象并以数组形式返回。在这里指定控件是日期框（datebox），日期框的默认按钮是'Today'和'Close'按钮，因此 buttons 的值也就是数组['Today 按钮对象','Close 按钮对象']。

所谓的按钮对象，读者可以简单地理解为包含按钮名称、消息响应函数等属性的集合。

接下来看这段代码：

```
01  buttons.splice(1, 0, {
02      text: '确定',
03      handler: function(target){
04      alert('当前选择的日期是'+$(target).val());
05      }
06  });
```

我们先简单地学习一下 splice 函数的用法，splice 是 JavaScript 的一个方法，用于向数组添加或者删除元素，它的基本语法是：

```
arrayObject.splice(index,howmany,item1,.....,itemX)
```

- index：规定添加/删除元素的位置，使用负数可从数组结尾处规定位置。
- howmany：规定要删除的元素数量。如果设置为 0，则不会删除元素。
- item1：向数组添加的新元素。
- arrayObject：需要被进行添加/删除的数组。

index 规定的元素位置是从 0 开始计数的，本例中的位置顺序是['0'=>'Today 按钮对象', '1'=>'Close 按钮对象']

设置 index 参数值为 1, howmany 参数值为 0 意味着在第二个元素的位置上新增一个 item1 元素，原先位置上的 Close 按钮对象保留并向后退一位，此时 buttons.数组的值就变成了['Today 按钮对象'，'新增的按钮对象'，'Close 按钮对象']。如果设置 howmany 的值为 1，就会删除原先位置上的 Close 按钮对象，此时 buttons.数组的值就变成了['Today 按钮对象'，'新增的按钮对象']。在本例中新增的按钮对象为：

```
01  {
02          text: '确定',
03          handler: function(target){
04          alert('当前选择的日期是'+$(target).val());
05          }
06  }
```

在 JavaScript 中使用{}来表示一个对象。

其中 text 表示按钮的名称，handler 为按钮单击消息的响应函数，参数 target 为日期框中的编辑框对象，这样通过 jQuery 的取值方法就能轻松获取当前选中的日期。

图 2.21 中在文本框内显示的日期格式为 X（月）/X（日）/X（年），这并不符合中国人的阅读模式，我们更希望将日期格式改为 X 年 X 月 X 日，此时需要使用 formatter 和 parser 属性来设置，许多初学者可能会写下如下代码：

```
01  <input id="db" type="text" class="easyui-datebox" required="required">
02          <script>
03              $(function(){
```

```
                $('#db').datebox({
                    parser:function(s){
                        var t =Date.parse(s);
                        if (!isNaN(t)){
                            return new Date(t);
                        } else {
                            return new Date();
                        }
                    },
                    formatter:function(date){
                        var y = date.getFullYear();
                        var m = date.getMonth()+1;
                        var d = date.getDate();
                        return y+'年'+m+'月'+d+'日';
                    }
                });
            });
        </script>
```

最终运行结果如图 2.23 所示。

图 2.23　设置日期框的展示值

此时会发现日期的格式虽然显示正常，但是无论单击哪一天总是会显示当前日期，这是因为 parser 属性是对初始值以及输入值进行解析的。我们在数字框章节中讲到，parser 属性是将初始值以及输入值解析成合法的存储值。例如，上述代码中通过 if(!isNaN(t))来判断输入框中的值是否是一个合法的日期格式，如果不是合法日期格式就将当前的日期作为存储值返回。Date 对象的 parse 方法无法解析中文格式的日期，始终返回当前的日期对象，所以我们会发现无论在日历中选中哪一天，文本框中始终显示当前的日期。解决这个问题的方法就是，在 parser 属性中将中文格式的日期转化成 Date 对象的 parse 方法所接收的日期格式，例如 X（月）/X（日）/X（年）。

使用中文格式日期还存在一个问题，我们无法限制用户在文本框中按照我们规定的格式输入日期。例如，我们使用 parser 属性的方法对 X 年 X 月 X 日格式日期进行解析，此时如果用户输入 X（年）/X（月）/X 日），那么日期框仍然无法正常运作。解决这个问题的方法就是在对用户输入进行解析时判断多种情况，相关代码如下：

```
    parser: function(s){
        //使用正则表达式解析用户输入
        //第一种解析的格式 X 年 X 月 X 日
        var m1 = /\d 年\d 月\d 日/;
        //第二种解析的格式 X/X/X;
        var m2 = /\d\/\d\/\d/;
        //第三种解析的格式 X.X.X
        var m3 = /\d\.\d\.\d/;
        if (m1.test(s)){
        var tmp1 = s.split("年");
            var year = tmp1[0];
            var tmp2 = tmp1[1].split("月");
            var month= tmp2[0];
            var tmp3 = tmp2[1].split("日");
            var day  = tmp3[0]
            return new Date(year+"/"+month+"/"+day);
        }else if(m2.test(s)){
            return new Date(s);
        } else if(m3.test(s)){
            return new Date(s);
        }
        else {
            return new Date();
        }
    },
    formatter:function(date){
        var y = date.getFullYear();
        var m = date.getMonth()+1;
        var d = date.getDate();
        return y+'年'+m+'月'+d+"日";
    }
```

最终运行结果如图 2.24 所示。

图 2.24　设置中文格式的日期框

【本节详细代码参见随书源码：\源码\easyui\example\c2\dateboxCHN.html】

3. 日期框事件

日期框常用事件说明见表 2.41。

表 2.41　日期框常用事件

名称	参数	描述
onSelect	date	当用户选中一个日期后触发

4. 日期框方法

日期框常用方法说明见表 2.42。

表 2.42　日期框常用方法

名称	参数	描述
options	none	返回选项对象
calendar	none	获取日历对象
setValue	value	设置日期框的值
cloneFrom	from	将日期框从一个地方复制到另一个地方

5. 日历的共用

在网站开发中经常会遇到一个页面中用到多个日期时间框的情况，通常情况下需要通过日期框的 calendar 方法获取其日历对象，然后对日历对象进行汉化，但是对每个日期框都使用这种方式会使开发变得十分烦琐，因此我们可以采用多个日期框共用一个日历的方法来减少开发的难度。日期框可以通过 sharedCalendar 属性和 cloneFrom 方法来共用日历。其中 sharedCalendar 属性是对日历进行复用，使用方法如下：

```
<input id="dt1" type="text">
<input id="dt2" type="text">
<div id="cc"></div>
<script>
    $(function(){
        $('#dt1').datebox({
            sharedCalendar:"#cc"
        });
        $('#dt2').datebox({
            sharedCalendar:"#cc"
        });
        $('#cc').calendar({
            current:new Date(),
            width:400,
            height:300,
            firstDay:"1",
            months:['1 月','2 月','3 月','4 月','5 月','6 月','7 月','8 月',
            '9 月','10 月','11 月', '12 月'],
            weeks:['周日','周一','周二','周三','周四','周五','周六'],
```

```
20                  });
21              });
22          </script>
```

使用该方法时需要先初始化一个日历，然后使用 sharedCalendar 属性使两个日期框共同使用这个日历。

使用日期框的 cloneFrom 方法也可以完成日历的共享，与 sharedCalendar 属性不同的是，cloneFrom 方法会共用指定日期框的全部属性，也可以称它是完成日期框的复制操作，例如：

```
01          <input id="dt1" type="text">
02          <input id="dt2" type="text">
03          <script>
04              $(function(){
05                  $('#dt1').datebox({
06                      width:300,
07                      currentText:"今天",
08                      closeText:"关闭",
09
10                  });
11                  $('#dt1').datebox('calendar').calendar({
12                      current:new Date(),
13                      width:400,
14                      height:300,
15                      firstDay:"1",
16                      months:['1 月','2 月','3 月','4 月','5 月','6 月','7 月','8 月',
17                      '9 月','10 月',  17   '11 月', '12 月'],
18                      weeks:['周日','周一','周二','周三','周四','周五','周六'],
19                  });
20                  $('#dt2').datebox("cloneFrom","#dt1");
21              });
22          </script>
```

最终运行结果如图 2.25 所示。

图 2.25　日期框的复制

【本节详细代码参见随书源码：\源码\easyui\example\c2\shareCalendar.html】

2.5.3 日期时间框（DateTimeBox）

日期时间框的使用与日期框一样，与日期框不同的是日期时间框增加了一个时间微调器用于显示时间。

日期时间框的依赖关系如下：

- datebox
- timespinner

日期时间框扩展于：

- datebox

日期时间框的默认配置定义在$.fn.datetimebox.defaults 中。

1. 日期时间框的用法

使用标记创建日期时间框的方法如下：

```
<input class="easyui-datetimebox" >
```

使用 JavaScript 创建日期时间框的方法如下：

```
01  <input id="dt" type="text">
02  $('#dt').datetimebox();
```

2. 日期时间框的属性

日期时间框常用属性见表 2.43。

表 2.43 日期时间框常用属性

名称	类型	描述	默认值
currentText	string	当前日期按钮上显示的文本	Today
closeText	string	关闭按钮上显示的文本	Close
okText	string	确定按钮上显示的文本	Ok
spinnerWidth	number	嵌入在日期时间框中时间微调器的宽度	100%
showSeconds	boolean	时间是否精确显示到秒	true
timeSeparator	string	时间分隔符，分割小时、分钟以及秒数的分割符	:

3. 日期时间框的事件

日期时间框无新增和重写的事件。

4. 日期时间框的方法

日期时间框常用方法见表 2.44。

表 2.44　日期时间框常用方法

名称	参数	描述
options	none	返回选项对象
spinner	none	获取微调器对象
setValue	value	设置日期时间框的值
cloneFrom	from	将日期时间框从一个地方复制到另一个地方

2.6 其他高级组件

本节将讲解 EasyUI 中的一些高级组件，这些组件都是由一些基础组件组合而来，它们的使用非常简单，但是功能却非常强大。

2.6.1 标签框（TagBox）

在前面的章节中我们介绍了组合框的使用方法，当设置组合框属性 multiple 为 true 时，组合框支持多选，此时如果需要修改选中内容时，可以在文本框中删除指定内容或者在下拉面板中重新选择。标签框由组合框扩展而来，它可以在文本框内显示标签而不是展示值，此时如果用户需要删除选中的内容，只需在文本框中将标签删除即可，而不需要将内容一个个地删除。

标签框的依赖关系如下：

- combobox

标签框扩展于：

- combobox

标签框的默认配置定义在$.fn.tagbox.defaults 中。

1. 创建标签框

使用标记创建标签框的方法如下：

```
<input class="easyui-tagbox" value="标签元素 1,标签元素 2" label="请选择">
```

使用 JavaScript 创建标签框的方法如下：

```
<input id="tb" type="text" style="width:300px">
$('#tb').tagbox({
    label: '请选择',
    value: ['标签元素 1','标签元素 2']
});
```

2. 标签框属性

标签框常用属性说明见表 2.45。

表 2.45 标签框常用属性

名称	类型	描述	默认值
hasDownArrow	boolean	标签框右侧是否显示下拉图标	false
tagFormatter	function(value,row)	设置标签的展示值	
tagStyler	function(value,row)	给标签的展示值添加风格	

3. 标签框事件

标签框常用事件见表 2.46。

表 2.46 标签框常用属性

名称	类型	描述
onClickTag	value	标签被单击时触发
onBeforeRemoveTag	value	删除标签前触发，如果返回 false 则阻止删除行为
onRemoveTag	value	删除标签时触发

4. 标签框方法

标签框无新增和重写的方法。

2.6.2 搜索框（SearchBox）

搜索框由文本框和菜单按钮组成，用户可以在菜单按钮中选择不同的搜索类型，当用户单击搜索框右侧的图标时就会触发搜索行为。

搜索框的依赖关系如下：

- textbox
- menubutton

搜索框扩展于：

- textbox

搜索框的默认配置定义在$.fn.searchbox.defaults 中。

1. 创建搜索框

使用标记创建搜索框的方法如下：

```
<input id="ss" class="easyui-searchbox" style="width:300px"
       data-options="searcher:qq,prompt:'请输入搜索内容',menu:'#mm'"></input>
<div id="mm" style="width:120px">
   <div data-options="name:'all',iconCls:'icon-ok'">类型一</div>
```

```
    <div data-options="name:'sports'">类型二</div>
</div>
<script type="text/JavaScript">
  function qq(value,name){
     alert(value+":"+name)
  }
</script>
```

通过 JavaScript 创建搜索框的方法如下：

```
<input id="ss"></input>
<div id="mm" style="width:120px">
    <div data-options="name:'all',iconCls:'icon-ok'">类型一</div>
    <div data-options="name:'sports'">类型二</div>
</div>
$('#ss').searchbox({
    searcher:function(value,name){
        alert(value + "," + name)
    },
    menu:'#mm',
    prompt:'请输入搜索内容'
});
```

2. 搜索框属性

搜索框常用属性说明见表 2.47。

表 2.47　搜索框常用属性说明

名称	类型	描述	默认值
width	number	组件的宽度	auto
height	number	组件的高度	22
prompt	string	显示在文本框内的提示消息	''
value	string	输入值	''
menu	selector	菜单选择器	null
searcher	function(value,name)	用于相应用户搜索行为的函数	null
disabled	boolean	禁用搜索框	false

searcher 属性是搜索框最重要的一个属性，当用户单击搜索图标时，就会运行该属性内定义的方法。其中的参数 value 为文本框内的值，name 参数为用户选择的菜单元素的 name 属性。

3. 搜索框事件

搜索框无新增和重写事件。

4. 搜索框方法

搜索框常用方法说明见表 2.48。

表 2.48　搜索框常用属性

名称	类型	描述
options	none	返回选项对象
menu	none	返回菜单对象
textbox	none	返回文本框对象
getValue	none	返回当前搜索值
setValue	value	设置一个新的搜索值
getName	none	获取当前选中的搜索类型名称
selectName	none	设置当前的搜索类型
destroy	none	销毁组件
resize	width	调整组件尺寸
disable	none	禁用组件
enable	none	启用组件
clear	none	清除搜索内容
reset	none	重置搜索内容

2.6.3　文件框（FileBox）

文件框用于用户上传表单文件，扩展自文本框。文件框中可以使用文本框的属性、事件以及方法，但是出于浏览器安全的考虑，部分方法如'setValue'可能无法在文件框上使用。本节将简单介绍文件框的使用方法，在下一节表单中将向读者演示如何利用表单上传文件。

文件框的依赖关系如下：

- textbox

文件框扩展于：

- textbox

文件框的默认配置定义在$.fn.filebox.defaults 中。

1. 创建文件框

使用标记创建文件框的方法如下：

```
<input class="easyui-filebox" style="width:300px">
```

通过 JavaScript 创建文件框的方法如下：

```
<input id="fb" type="text" style="width:300px">
$('#fb').filebox({
    buttonText: 'Choose File',
    buttonAlign: 'left'
```

```
05  })
```

2. 文件框属性

文件框常用属性说明见表 2.49。

表 2.49　文件框常用属性

名称	类型	描述	默认
buttonText	string	文件框右侧按钮上的显示文本	Choose File
buttonIcon	string	文件框右侧按钮上的显示图标	null
buttonAlign	string	按钮的位置，可能是'left'和'right'	right
accept	string	指定服务器接收的类型	
multiple	boolean	定义是否允许上传多个文件	false
separator	string	多文件上传时各个文件名称之间的分隔符	,

accept 属性可以限制用户选择的文件类型，例如限制用户只能选择图片，代码如下：

```
01  $('#file').filebox({
02      accept: 'image/*'
03  });
```

其中符号*的含义是允许用户选择所有的 image 类型文件，常见的 accept 值见表 2.50。

表 2.50　常见的accept值

类型	accept 值
*.3gpp	audio/3gpp, video/3gpp
*.ac3	audio/ac3
*.asf	allpication/vnd.ms-asf
*.au	audio/basic
*.css	text/css
*.csv	text/csv
*.doc	application/msword
*.dot	application/msword
*.dtd	application/xml-dtd
*.dwg	image/vnd.dwg
*.dxf	image/vnd.dxf
*.gif	image/gif
*.htm	text/html
*.html	text/html

（续表）

类型	accept 值
*.jp2	image/jp2
*.jpe	image/jpeg
*.jpeg	image/jpeg
*.jpg	image/jpeg
*.js	text/JavaScript, application/JavaScript
*.json	application/json
*.mp2	audio/mpeg, video/mpeg
*.mp3	audio/mpeg
*.mp4	audio/mp4, video/mp4
*.mpeg	video/mpeg
*.mpg	video/mpeg
*.mpp	application/vnd.ms-project
*.ogg	application/ogg, audio/ogg
*.pdf	application/pdf
*.png	image/png
*.pot	application/vnd.ms-powerpoint
*.pps	application/vnd.ms-powerpoint
*.ppt	application/vnd.ms-powerpoint
*.rtf	application/rtf, text/rtf
*.svf	image/vnd.svf
*.tif	image/tiff
*.tiff	image/tiff
*.txt	text/plain
*.wdb	application/vnd.ms-works
*.wps	application/vnd.ms-works
*.xhtml	application/xhtml+xml
*.xlc	application/vnd.ms-excel
*.xlm	application/vnd.ms-excel
*.xls	application/vnd.ms-excel
*.xlt	application/vnd.ms-excel
*.xlw	application/vnd.ms-excel
*.xml	text/xml, application/xml
*.zip	aplication/zip

3. 文件框事件

文件框无新增或重写属性。

4. 文件框方法

文件框常用方法说明见表 2.51。

表 2.51　文件框常用方法

名称	参数	描述
files	none	返回选择的文件列表对象

2.7 表单

在本章的开头部分，我们讲到一个表单由三部分组成，即表单标签、表单域、表单按钮，并且学习了 EasyUI 表单域中的部分控件以及各类按钮的使用方法。在前面的章节中，我们介绍的各类控件大都是用来处理用户的各类操作，其仅仅属于客户端的操作，而一个网站更需要服务器端来处理客户端的各种操作，表单就是客户端与服务器端之间交互的一个中介，通过表单我们可以将各类控件的值传输到服务器端。本节将先介绍表单的使用方法，随后将向读者演示一些实用的程序。

2.7.1　表单的基本使用方法

本节将先带领读者学习表单的基本使用方法，接着会通过几个例子进一步讲解表单的使用。

表单的默认配置定义在$.fn.form.defaults 中。

1. 创建表单

表单只能通过标记来创建：

```
<form id="ff">
    ...表单域内的各类控件以及表单按钮
</form>
```

2. 表单属性

表单常用属性说明见表 2.52。

表 2.52　表单常用属性

名称	类型	描述	默认
novalidate	boolean	设置为 false 时可以对表单字段进行验证	false
iframe	boolean	是否使用 iframe 提交表单，设置为 true 时将在 iframe 中提交表单，设置为 false 时将通过 ajax 提交表单	true

（续表）

名称	类型	描述	默认
ajax	boolean	定义是否通过 ajax 提交表单	true
dirty	boolean	定义是否仅提交值发生改变的字段	false
queryParams	object	提交表单时向服务器传输的一些额外参数	{}
url	string	服务器端地址	null

3. 表单事件

表单常用事件说明见表 2.53。

表 2.53　表单常用事件说明

名称	参数	描述
onSubmit	param	提交表单前触发，返回 false 则阻止表单提交
onProgress	percent	提交数据时触发，percent 为当前数据上传的进度，使用该事件必须设置 iframe 为 false
success	data	表单成功提交后触发
onBeforeLoad	param	初始化数据加载前触发
onLoadSuccess	data	初始化数据加载成功时触发
onLoadError	none	初始化数据加载失败时触发
onChange	target	字段的值发生改变时触发

4. 表单方法

表单常用方法说明见表 2.54。

表 2.54 表单常用方法说明

名称	参数	描述
submit	options	提交表单，参数 options 是一个包含以下属性的对象 url: 服务器地址 onSubmit: 提交表单前触发的事件 success: 服务器处理完毕后回调的函数
load	data	加载数据初始化表单
clear	none	清除全部加载的数据
reset	none	重置全部加载的数据
validate	none	检查组件是否验证通过
enableValidation	none	启用验证
disableValidation	none	禁用验证
resetValidation	none	重置验证
resetDirty	none	重置被改变的标记

2.7.2　提交表单

通常我们会在控件的标记内定义一个 name 属性，在向服务器提交数据时，服务器可以通过该属性的值来获取数据，例如：

```
<form id="ff" method="post">
        <div id="nb" name="nb"></div>
        <a id="btn" href="#" class="easyui-linkbutton">提交</a>
</form>
<script>
    $(function(){
        $('#nb').numberbox({
            prefix:'$',
        });
        $('#ff').form({
            url:"form.php",
            onSubmit: function(){
                //提交前触发的事件
            },
            success:function(data){
                //服务器返回数据后触发
            }
        });
        $("#btn").click(function(){
            $('#ff').submit();
        });
    });
</script>
```

相关的服务器代码如下：

```
<?php
    //获取前端数据
    $name = $ POST["nb"];
    //注意$name 的值是存储值
?>
```

服务器接收的存储值，本例中如果用户输入数字 1 时数字框会显示$1，但是提交到服务器的则是存储值 1。

我们也可以使用表单的 submit 方法动态提交表单，例如上述代码等同于：

```
<form id="ff" method="post">
        <div id="nb" name="nb"></div>
        <a id="btn" href="#" class="easyui-linkbutton">提交</a>
</form>
<script>
    $(function(){
            $('#nb').numberbox({
                prefix:'$',
```

```
                });
            $("#btn").click(function(){
                $('#ff').form('submit',{
                    url:"form.php",
                    onSubmit: function(){
                        //提交前触发的事件
                    },
                    success:function(data){
                        //服务器返回数据后触发
                        alert(data);
                    }
                });
            });
        });
    </script>
```

在提交表单的过程中，可以使用 queryParams 属性添加一些额外的参数，例如：

```
queryParams:{param1:'1',param2:'2'},
```

服务器端获取这些参数的方法如下：

```
$param1 = $_POST["param1"];
```

在提交表单前通常需要检查组件中的输入值是否通过验证，此时只需在 onSubmit 事件中使用 validate 方法进行检查即可，该方法会自动扫描表单内的全部组件，只有所有组件都验证通过时它才会返回 true。例如：

```
        onSubmit: function(){
                    if($('#ff').form('validate')){
                        alert('通过验证');
                        return true;
                    }
                    else{
                        alert('未通过验证');
                        return false;
                    }
        }
```

2.7.3 初始化表单字段

在实际开发中经常会遇到对某些信息进行编辑的情况，例如编辑某人的个人信息等，此时需要先从服务器将用户数据取出并初始化指定的组件，EasyUI 表单提供了数据加载功能，它可以方便地帮助我们完成数据的初始化功能，它支持本地加载数据和从服务器加载数据两种方式。下面先向读者展示如何加载本地数据，相关代码如下：

```
    <form id="ff" method="post">
            <input id="nickname" name="nickname"><br/><br/>
            <input id="age" name="age"><br/><br/>
            <input id="birthday" name='birthday'>
    </form>
```

```
    <script>
        $(function(){
            $('#nickname').textbox({
                label:"姓名",
                labelWidth:150,
                width:350
            });
            $('#age').numberspinner({
                label:"年龄",
                labelWidth:150,
                width:350
            });
            $('#birthday').datebox({
                label:"出生日期",
                labelWidth:150,
                width:350
            });
            $("#ff").form('load',{
                nickname:'张三',
                age:'18',
                birthday:'7/3/2000'
            })
        });
    </script>
```

使用表单的 load 方法可以加载本地数据，可以发现在它的参数中仍然是使用字段的 name 属性进行赋值。最终运行结果如图 2.26 所示。

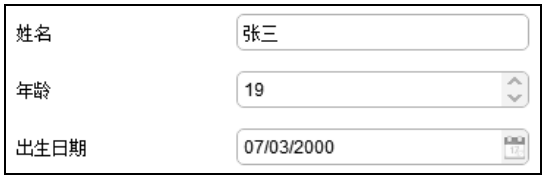

图 2.26　使用本地数据初始化表单

同样可以使用 load 方法加载服务器端数据来初始化表单，例如：

```
$("#ff").form('load','initData.php');
```

其中服务器端代码如下：

```
    $data = array('nickname'=>'zhangsan',
              'age'=>'18',
              'birthday'=>'7/3/2000'
            );
    echo json_encode($data);
```

服务器端的数据结构与本地加载时一致，也是通过字段的 name 属性来赋值。

2.7.4 文件上传

本节将讲解如何利用表单上传文件，先看一个例子：

```
<form id="ff" method="post">
    <!-- 只接受 pdf 类型文件 -->
    <input class="easyui-filebox" name='file' style="width:300px"
     data-options="accept:'application/pdf'" ><br/><br/>
    <div id="p" class="easyui-progressbar"
style="width:300px;"></div><br/><br/>
    <a id="btn" href="#" class="easyui-linkbutton">上传文件</a>
</form>
<script>
    $(function(){
        $("#ff").form({
            iframe:false,
            onProgress:function(percent){
                $("#p").progressbar("setValue",percent);
            }
        });
        $("#btn").click(function(){
            $('#ff').form('submit',{
                url:"server/upload.php",
            });
        });
    });
</script>
```

这个例子里面我们使用了表单的 onProgress 事件，该事件返回文件上传过程中的进度，使用该事件先要设置表单的 iframe 属性为 false。我们使用了一个进度条组件（progressbar），该组件将会动态显示文件上传的进度。

读者也可以使用 $.messager.progress() 方法显示默认的进度条、使用 $.messager.progress('close')方法关闭默认进度条。不同的是默认进度条并不会根据表单处理的进度实时更新，它会循环地更新进度直到被关闭为止。

服务器端同样是使用文件框的 name 来获取上传的文件，其代码如下：

```
// 文件的保存目录,默认文件会被保存到 D 盘根目录下
$path = "D://";
if (file_exists($path . $_FILES["file"]["name"])) {
    echo $_FILES["file"]["name"] . " 已存在. ";
} else {
    move_uploaded_file($_FILES["file"]["tmp_name"], $path .
$_FILES["file"]["name"]);
}
```

最终运行结果如图 2.27 所示。

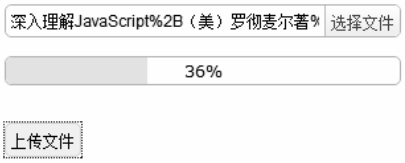

图 2.27　文件上传演示

【本节详细代码参见随书源码：\源码\easyui\example\c2\uploadFile.html】

本书仅仅是简单地向读者演示服务器端的相关代码，由于服务器端开发语言较多且相关框架众多，为减轻读者的学习成本，在本书中将不重点介绍服务器端的相关代码。读者在学习服务器端代码时只需关注两点即可：（1）服务器端如何获取数据；（2）服务器端如何返回数据。在本节中服务器端主要是通过标记的 name 属性获取数据，返回的是一个 JSON 格式数据。

2.8 小结

本章重点向读者介绍了 EasyUI 中的各类组件，它们绝大部分都是扩展于文本框，因此它们大部分都拥有文本框的属性、事件和方法。本章中的内容将在第 6 章进一步深入讲解，目前读者仅需掌握如下知识点。

- 组件之间的依赖和扩展关系。
- 组件值的含义，能区分存储值、展示值和初始值。
- 通过表单将组件的存储值上传至服务器。