

# 第7章 jQuery插件和前端常用组件



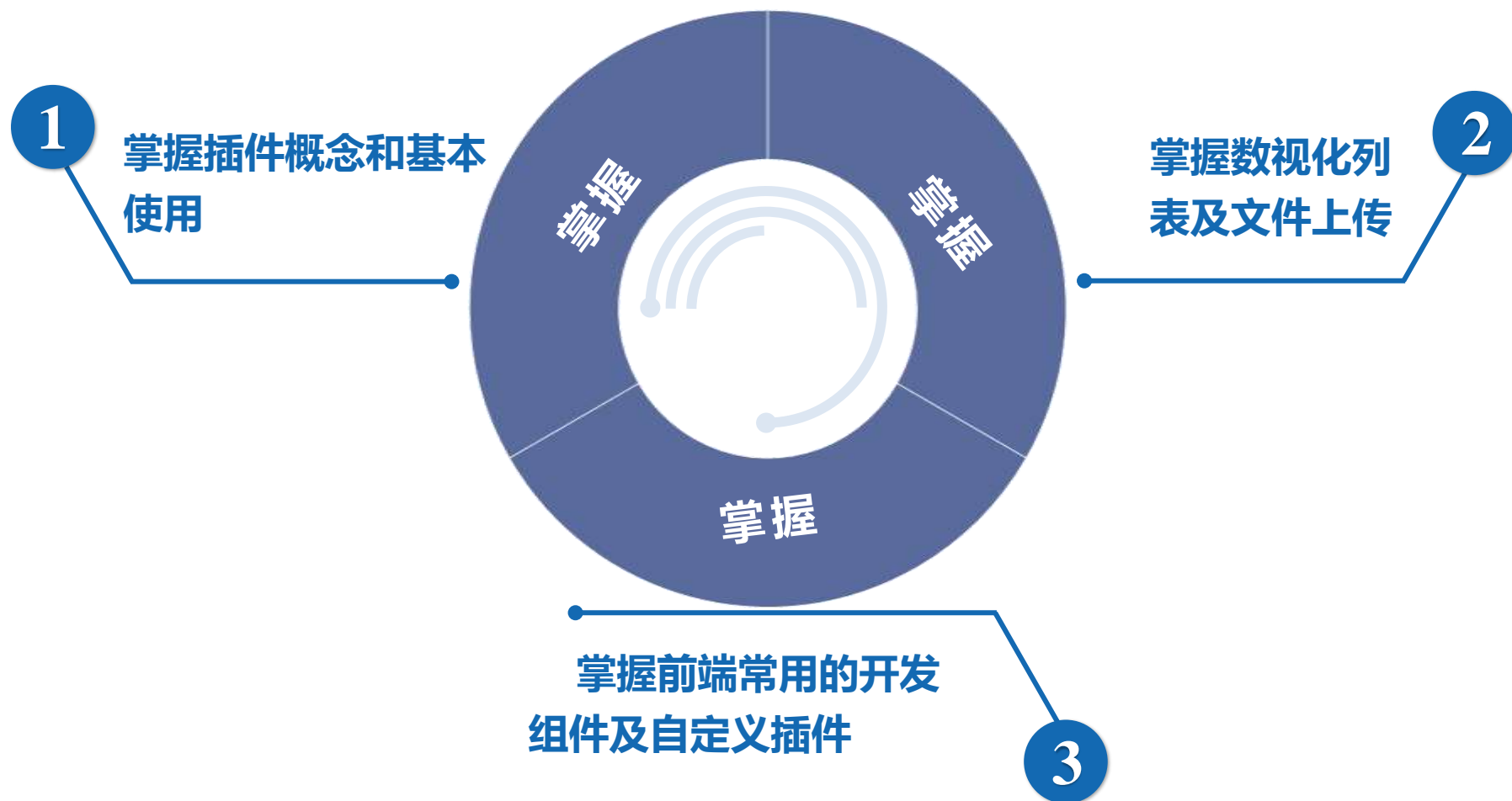
jQuery

- 插件概念和基本使用
- 模板引擎插件的使用
- 文件上传插件的使用
- 开发自定义插件
- 数据可视化列表
- 编辑器插件的使用



# 学习目标

传智播客·黑马程序员  
改变中国IT教育 我们正在行动





# 目录

传智播客·黑马程序员  
改变中国IT教育 我们正在行动

7.1

## jQuery插件的概述

[点击查看本小节知识架构](#)

7.2

## 开发自定义插件

[点击查看本小节知识架构](#)


7.3

## 模板引擎

[点击查看本小节知识架构](#)



# 目录

 传智播客. 黑马程序员  
改变中国IT教育 我们正在行动

7.4

## 数据可视化图表

 [点击查看本小节知识架构](#)

7.5

## 文件上传

 [点击查看本小节知识架构](#)

7.6

## 编辑器

 [点击查看本小节知识架构](#)



## 7.1 jQuery插件的概述

### 1. 插件的概述

**插件**：插件是一种遵循某种规范的应用程序接口编写出来的程序，只能运行在指定的环境中。

**jQuery插件**：在引入jQuery条件下运行的程序。

**jQuery插件官方网址**：<http://plugins.jquery.com/>

**jQuery插件GitHub地址**：<https://github.com/jquery-archive/plugins.jquery.com>



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

语法：

```
$.fn.插件名 = function(参数列表) {  
    // 编写插件的代码  
};
```

\$.fn是jQuery原型对象jQuery.prototype的简写



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

方法中this的指向：可通过this访问调用当前方法的jQuery对象

```
$.fn.test = function() {  
    return this === obj;  
};  
var obj = $('div');  
console.log(obj.test()); // 输出结果 : true
```

说明插件内部this与jQuery  
对象obj是同一个对象



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

注意

在编写插件时，应避免使用jQuery对象的简写“\$”，防止发生名称的冲突。





## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

#### 封装jQuery对象方法实现的插件

避免命名冲突：将插件的方法放在闭包函数中封装完插件后将其保存为一个单独的js文件，在使用插件时直接引入文件即可。推荐文件名使用“jquery.插件名.js”格式，防止与其他JavaScript库插件混淆。

```
(function($) {  
    $.fn.插件名 = function() {  
        // 编写插件的代码  
    };  
})(jQuery);
```



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

#### 封装jQuery对象方法实现的插件

通过一个案例进行演示封装jQuery对象方法的插件：制作tableColor插件，实现表格隔行换色功能。



ID编号	姓名	性别
001	Tom	男
002	Jim	男
003	Lucy	女
004	Lily	女



ID编号	姓名	性别
001	Tom	男
002	Jim	男
003	Lucy	女
004	Lily	女



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

制作tableColor插件页面结构

demo7-1.html 引入文件:

```
<script src="jquery-1.12.4.js"></script>  
<script src="jquery.tableColor.js"></script>
```

先引入jQuery文件，  
再引入插件文件



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

#### 封装jQuery对象方法实现的插件

制作tableColor插件页面结构

demo7-1.html 文档结构:

```
<table width="500px" border="1" cellspacing="0">
  <tr><th>ID编号</th><th>姓名</th><th>性别</th></tr>
  <tr><td>001</td><td>Tom</td><td>男</td></tr>
  <tr><td>002</td><td>Jim</td><td>男</td></tr>
  <tr><td>003</td><td>Lucy</td><td>女</td></tr>
  <tr><td>004</td><td>Lily</td><td>女</td></tr>
</table>
```



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

实现表格颜色设置:

```
var table = $('table');  
// 设置偶数行样式  
table.find('tr:even').css({background: 'lightBlue', color: 'red'});  
// 设置标题行样式  
table.find('tr:first').css({background: 'green', color: '#fff'});
```



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

jquery.tableColor.js 编写插件:

```
(function($) {  
    $.fn.tableColor = function(options) {  
        for (var i in options) {  
            this.find(i).css(options[i]);  
        }  
        return this;  
    };  
})(jQuery);
```

返回当前对象，实现链式调用



## 7.2 开发自定义插件

### 1. 封装jQuery对象方法的插件

封装jQuery对象方法实现的插件

tableColor插件使用：

```
$('#table').tableColor({  
  'tr:even': {background: 'lightBlue', color: 'red'},  
  'tr:first': {background: 'green', color: '#fff'}  
});
```



## 7.2 开发自定义插件

脚

下

留

心

### 插件书写注意事项

在编写插件时，要养成以分号（;）结尾的习惯，因为插件代码将来可能被压缩、合并，如果省略分号会导致出错。同样的，为防止他人不规范的代码对自定义插件的影响，可以在插件的头部添加一个分号（;），如下所示。

```
(function($){  
    $.fn.插件名 = function() {};  
})(jQuery);
```





## 7.2 开发自定义插件

### 2. 封装静态方法插件

封装jQuery对象方法实现的插件

语法:

```
$.插件名 = function(插件列表) {  
    // 编写插件的代码  
};
```

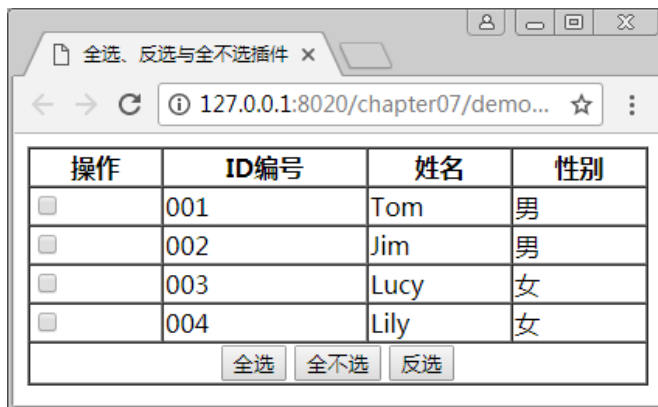


## 7.2 开发自定义插件

### 2. 封装静态方法插件

#### 封装静态方法插件：

通过一个案例进行演示封装静态方法插件：制作checkbox插件，实现表格全选、反选以及全不选操作。





## 7.2 开发自定义插件

### 2. 封装静态方法插件

#### 封装静态方法插件

jquery.checkbox.js: 定义函数Checkbox，并将其添加到\$对象上。

```
function Checkbox(ele) {  
    this.ele = ele;  
}  
$.checkbox = function(selector) {  
    return new Checkbox($(selector));  
};
```



## 7.2 开发自定义插件

### 2. 封装静态方法插件

[jquery.checkbox.js](#): 为构造函数Checkbox添加原型方法。

```
Checkbox.prototype = {  
  checkAll: function() { // 全选  
    this.ele.prop('checked', true);  
  },  
  uncheckAll: function() { // 全不选  
    this.ele.prop('checked', false);  
  },  
  checkInvert: function() { // 反选  
    this.ele.each(function() {this.checked = !this.checked;});  
  };  
};
```



## 7.2 开发自定义插件

### 2. 封装静态方法插件

[demo7-2.html](#) 页面结构:

```
<tr><th>操作</th><th>ID编号</th><th>姓名</th>...</tr>
<tr>
  <td><input type="checkbox">...男</td>
</tr>
<tr>
  <td colspan="4" align="center">
    <input id="checkAll" type="button" value="全选">...
  </td>
</tr>
```



## 7.2 开发自定义插件

### 2. 封装静态方法插件

[demo7-2.html](#) : 中使用Checkbox插件方法实现全选、反选、全不选:

```
var ele = $.checkbox(':checked');
$('#checkAll').click(function() {
    ele.checkAll();
});
$('#uncheckAll').click(function() {
    ele.uncheckAll();
});
$('#checkInvert').click(function() {
    ele.checkInvert();
});
```



## 7.2 开发自定义插件

多

学

一

招

### extend()方法的使用

为调用extend()方法的对象添加成员

“\$.extend(obj)”：表示将obj对象的成员添加到\$对象中

“\$.fn.extend(obj)”：表示将obj对象的成员添加到\$.fn对象中

```
$.extend({  
    插件方法1: function() {},  
    插件方法2: function() {}  
});
```

```
$.fn.extend({  
    插件方法1: function() {},  
    插件方法2: function() {}  
});
```



## 7.2 开发自定义插件

多

学

一

招

### extend()方法的使用

合并对象成员：extend()方法支持合并一个或多个对象成员，该方法的第1个参数表示目标对象，第2~N个参数表示被合并的对象

在合并时，遇到同名的成员将会覆盖，合并后，原对象的成员顺序不会改变。

```
var obj1 = {a: 1, b: 1, c: 1};  
var obj2 = {b: 2, a: 2};  
var obj3 = $.extend(obj1, obj2);  
console.log(obj1); // 输出结果 : Object {a: 2, b: 2, c: 1}  
console.log(obj2); // 输出结果 : Object {b: 2, a: 2}  
console.log(obj3 === obj1); // 输出结果 : true
```





## 7.2 开发自定义插件

多

学

一

招

### extend()方法的使用

合并对象成员：如果调用插件方法时传入的options对象中省略了可选参数，就会在options对象中自动加上这些参数。

```
(function($) {  
    var defaults = {可选参数1: 默认值1, 可选参数2: 默认值2};  
    $.插件名 = function(options) {  
        options = $.extend({}, defaults, options);  
    };  
    $.插件名.defaults = defaults; // 允许插件的使用者查看或更改可选参数  
})(jQuery);
```



## 7.2 开发自定义插件

传智播客·黑马程序员  
改变中国IT教育 我们正在行动

多

学

一

招

### extend()方法的使用

深拷贝合并：\$.extend()方法第一个参数在设为true时，表示采用递归方式合并对象。

```
var defaults = {sub: 'js', info: {id: 2, name: 'Tom'}};  
var obj = {info: {name: 'Jimmy'}, score: 96};  
var newObj = $.extend(true, {}, defaults, obj);  
// 输出结果 : {sub: "js", info: {id: 2, name: "Jimmy"}, score: 96}  
console.log(newObj);
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

#### 案例展示:

- ① 焦点图自动切换
- ② 当鼠标悬停、离开时，焦点图暂停、开启自动播放
- ③ 鼠标悬停在小圆点，修改当前圆点的显示样式，切换到对应的图片



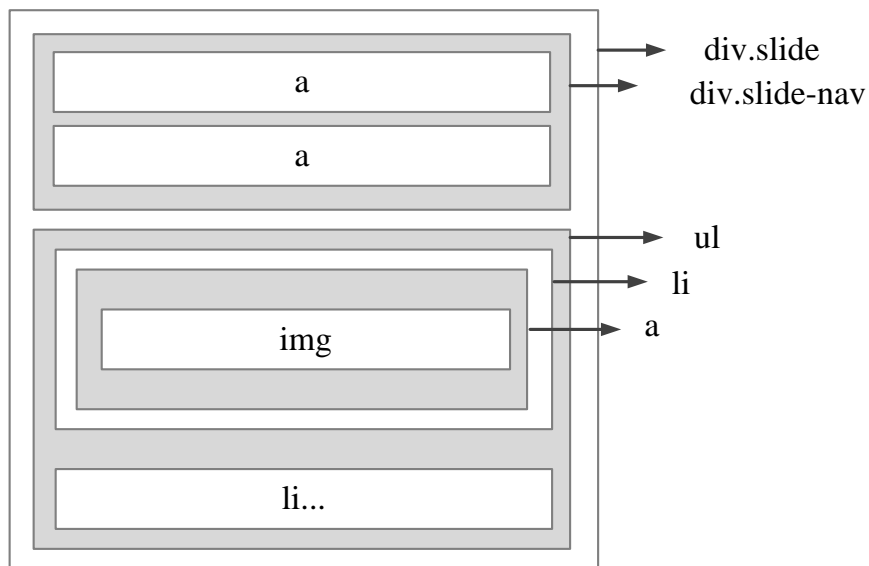


## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

#### 案例分析：

焦点图案例的HTML结构。





## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

#### 案例分析：

- ① 焦点图案例的CSS样式
- ② 根据焦点图的大小限定最外层div容器的宽高，通过浮动和定位让所有焦点图都显示在此区域内，接着设计小圆点的默认样式和鼠标滑过时的样式，从而完成整个页面的设计。





## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例分析：

焦点图案例的jQuery特效：

- ① 根据图片的个数生成小圆点
- ② 自动切换图片
- ③ 切换图片时，小圆点和图片一一对应





## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现:

编写HTML结构: slide.html 页面结构

```
<!-- 焦点图列表 -->
<ul>
  <li><a href="#"></a></li>
  ...
</ul>
<!-- 焦点图切换小圆点 -->
<div class="slide-nav"></div>
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现:

编写HTML结构: slide.html 引入文件。

```
<script src="jquery-1.12.4.js"></script>  
<script src="jquery.slide.js"></script>
```





## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现:

设计CSS样式: slide.css

```
.slide {position:relative; overflow:hidden; border:1px solid #ccc;}  
.slide ul {position:relative; list-style:none; margin:0; padding:0;}  
.slide ul li {position:absolute; display:none;}  
.slide-nav {position:absolute; right:0px; bottom:0px; height:30px;  
    line-height:34px; width:100%; background:rgba(0,0,0,0.2);  
    text-align:center;}  
.slide-nav a {background:#fff; border-radius:7px; width:14px;  
    height:14px; display:inline-block; margin:0 5px; cursor:pointer;}  
.slide-nav .slide-curr {background:#ff6700;}
```

小圆点被选中的样式



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：

封装焦点图插件 [jquery.slide.js](#) : 创建Slide构造函数

```
function Slide(obj) {} // Slide构造函数
Slide.prototype = {
  change: function(i, speed) {}, // 切换到第i张图片
  next: function() {}, // 切换到下一张图片
  start: function() {speed}, // 开启图片自动切换
  pause: function() {} // 暂停图片自动切换
};
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：

封装焦点图插件 [jquery.slide.js](#)：设置默认参数。

```
var defaults = {  
    speed: 3000,      // 默认切换间隔时间（毫秒）  
    width: '670px',   // 默认图片宽度  
    height: '240px',  // 默认图片高度  
    prefix: 'slide '  // class前缀  
};
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 [jquery.slide.js](#) 为jQuery对象增加slide方法

```
$.fn.slide = function(options) {  
    options = $.extend({}, defaults, options);  
    this.css({width: options.width, height: options.height});  
    var slide = new Slide(this, options.prefix);  
    // 鼠标滑到焦点图区域，暂停自动切换，离开时，恢复自动切换  
    this.hover(function() {slide.pause();},  
        function(){slide.start(options.speed)});  
    return slide.change(0, 0).start(options.speed);  
};  
$.fn.slide.defaults = defaults;
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

搭建插件骨架完善插件功能。



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 jquery.slide.js：完善Slide构造函数

```
function Slide(obj, prefix) {  
    // 根据图片个数，自动生成相应数量的小圆点切换按钮  
    this.pics = obj.find('li');  
    var nav = obj.find('.' + prefix + '-nav');  
    nav.append(new Array(this.pics.length + 1).join('<a></a>'));  
    this.dots = nav.find('a'); this.currCls = prefix + '-curr';  
    var slide = this;  
    // 当鼠标滑到某个小圆点上时，切换到对应的图片  
    this.dots.mouseover(function() {slide.change($(this).index());});  
}
```

获取焦点图图片

绑定鼠标经过事件



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 jquery.slide.js：为Slide构造函数完善原型方法

```
Slide.prototype = {  
    // 切换到索引值为i的图片，speed为动画速度  
    change: function(i, speed) {  
        this.i = i; // 保存传入的索引值  
        this.dots.eq(i).addClass(this.currCls).siblings('a')  
            .removeClass(this.currCls); // 小圆点切换  
        this.pics.eq(i).stop(true, true).fadeIn(speed).siblings('li').fadeOut(speed);  
        return this; // 图片切换  
    }  
};
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 [jquery.slide.js](#) 为Slide构造函数完善原型方法

```
Slide.prototype = {  
  // 切换到下一张图片，如果已经是最后一张，则自动回到第一张  
  next: function() {  
    if (++this.i >= this.pics.length) {  
      this.i = 0;  
    }  
    return this.change(this.i, 600);  
  }  
};
```





## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 [jquery.slide.js](#) 为Slide构造函数完善原型方法。

```
Slide.prototype = {  
  start: function(speed) { // 开始自动切换  
    var slide = this;  
    this.timer = window.setInterval(function() {  
      slide.next();  
    }, speed);  
    return this;  
  }  
};
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

案例实现：封装焦点图插件 `jquery.slide.js` 为Slide构造函数完善原型方法。

```
Slide.prototype = {  
    // 暂停自动切换  
    pause: function() {  
        window.clearInterval(this.timer);  
        return this;  
    }  
};
```



## 7.2 开发自定义插件

### 3. 案例自定义焦点图插件

使用插件: slide.html中使用插件。

```
$('.slide').slide();
```



## 7.3 模板引擎

### 1. art-template快速入门

#### art-template

**描述：**轻量级的JavaScript模板引擎，具有接近JavaScript极限的运行性能、精准的调试功能、简单的语法使用规则、同时支持JavaScript的原生语法等特性。

**GitHub地址：** <https://github.com/aui/art-template/releases>



## 7.3 模板引擎

### 1. art-template快速入门

#### 编写模板

#### demo7-4.html 模板:

```
<script id="tpl" type="text/html">
  <h2>{{title}}</h2>
  <ul>
    {{each list value index}}
      <li>索引 : {{index}} , 值 : {{value}}</li>
    {{/each}}
  </ul>
</script>
```

art-template标准语法  
开始和结束的标签



## 7.3 模板引擎

### 1. art-template快速入门

渲染模板

demo7-4.json 数据:

```
{  
  "title": "first demo",  
  "list": ["Purple potato", "Sweet potato", "Potato chips"]  
}
```



## 7.3 模板引擎

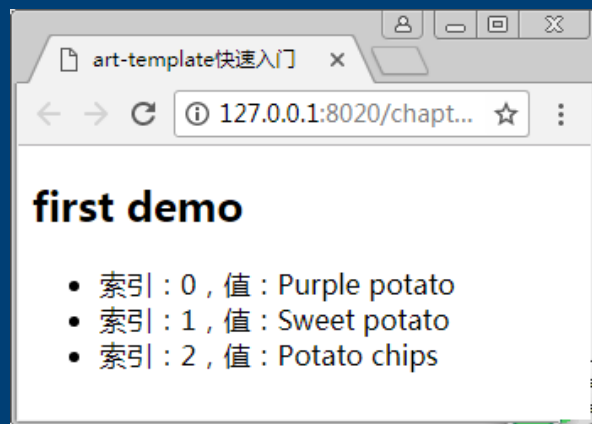
### 1. art-template快速入门

渲染模板：

demo7-4.html：

```
<script src="template-web.js"> </script>
<script src="jquery-1.12.4.js"> </script>
<script>
    $.get('demo7-4.json', function(data) {
        var con = template('tpl', data);
        $('#content').html(con);
    });
</script>
```

调用art-template模板引擎提供的template()函数





## 7.3 模板引擎

### 2. 标准语法和常用方法

art-template在使用时支持两套语法：

标准语法：支持基本的JavaScript表达式，语法简单实用，易于读写。

原始语法：支持JavaScript的所有语句，拥有强大的逻辑表达能力。





## 7.3 模板引擎

### 2. 标准语法和常用方法

不转义输出：数据原样输出在输出的数据前添加 “@” 字符

```
<div id="show"></div> <!-- 展示模板渲染数据后的内容 -->
```

```
<!-- 编写模板 -->
```

```
<script id="test" type="text/html">
```

```
    默认输出：{{title}}<hr>
```

```
    不转义输出：{{@title}}
```

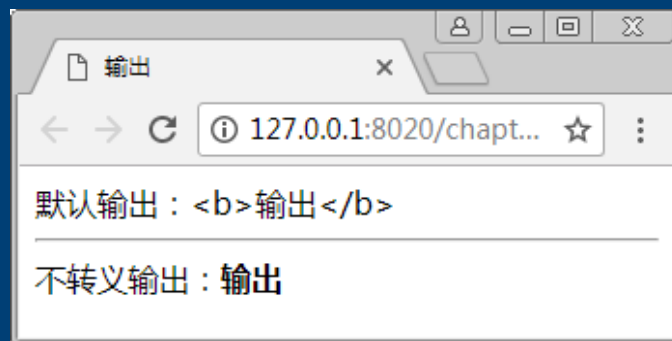
```
</script>
```

```
<!-- 渲染模板 -->
```

```
<script>  var con = template('test', {title: '<b>输出</b>'});
```

```
    $('#show').html(con);
```

```
</script>
```





## 7.3 模板引擎

### 2. 标准语法和常用方法

print输出字符串在模板中输出字符串

```
<div id="show"></div><!-- 展示模板渲染数据后的内容 -->
<script id="test" type="text/html"><!-- 编写模板 -->
    {{print flag? opt.num1 + opt.num2 : '此时flag为false' }}
</script>
<script><!-- 渲染模板 -->
    var data = {flag: false, opt: {num1: 3, num2: 8}};
    var con = template('test', data);
    $('#show').html(con);
</script>
```

flag为真时，在id等于show的元素内输出num1与num2的和11，否则输出“此时flag为false”的提示信息

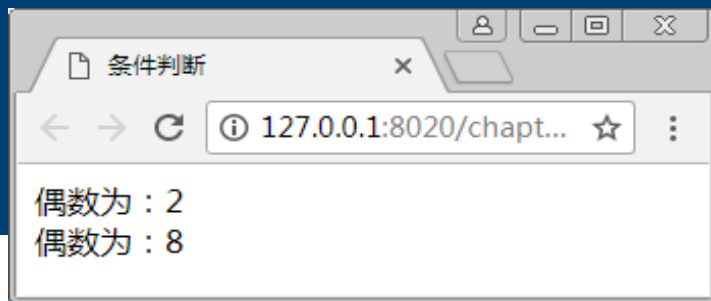


## 7.3 模板引擎

### 2. 标准语法和常用方法

#### 条件判断实现条件判断

```
<div id="show"></div> <!-- 展示模板渲染数据后的内容 -->
<script id="test" type="text/html"> <!-- 编写模板 -->
    {{each num v}} {{if v%2 == 0}} 偶数为 : {{v}} <br> {{/if}} {{/each}}
</script>
<script> <!-- 渲染模板 -->
    var con = template('test', {num: [2, 9, 8, 7, 1]});
    $('#show').html(con);
</script>
```





## 7.3 模板引擎

### 2. 标准语法和常用方法

嵌入子模板模板中的内容过于繁杂时，可以将其分块实现，通过嵌入子模板的方式引入。

编写模板1、模板2

```
<div id="show"></div> <!-- 展示模板渲染数据后的内容 -->
<script id="test" type="text/html"> <!-- 编写模板1 -->
    <h4>{{title}}</h4> {{include 'tpl'}}
</script>
<script id="tpl" type="text/html"> <!-- 编写模板2 -->
    <ul> {{each list v i}} <li>第 {{i + 1}}个 : {{v}}</li> {{/each}} </ul>
</script>
```

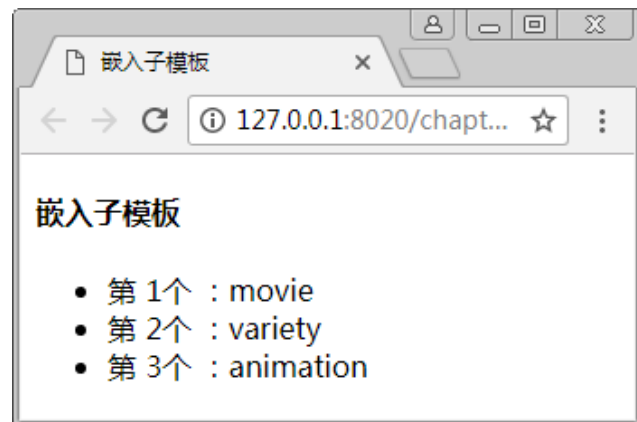


## 7.3 模板引擎

### 2. 标准语法和常用方法

嵌入子模板模板中的内容过于繁杂时，可以将其分块实现，通过嵌入子模板的方式引入。

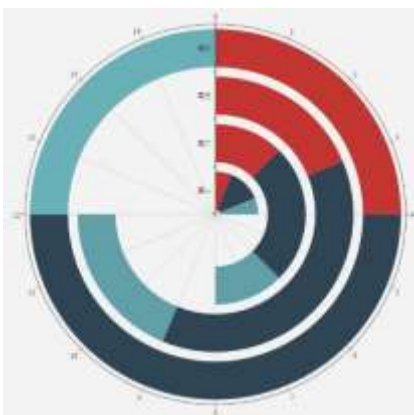
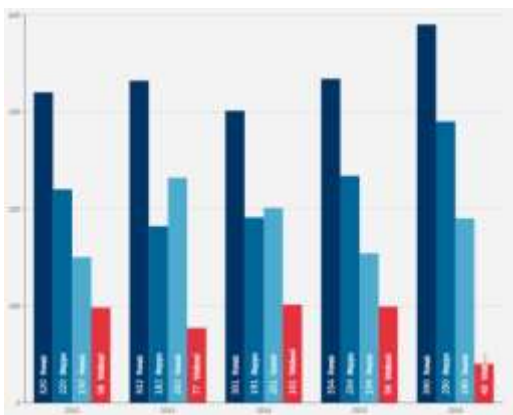
```
<script> <!-- 渲染模板 -->
  var data = {
    title: '嵌入子模板',
    list: ['movie', 'variety', 'animation']
  };
  var con = template('test', data);
  $('#show').html(con);
</script>
```





## 7.4 数据可视化图表

在实际开发中，根据实际需求，除了直接将后端返回的数据展示给用户外，还可以将其制作成可视化的图表（如折线图、饼状图等）展示，让数据信息变得更加直观。





## 7.4 数据可视化图表

### 1. 快速体验ECharts

**Echarts:** 一个纯JavaScript开源可视化库，具有丰富的可视化类型、绚丽的特效、支持交互式数据、跨平台应用、可高度个性化定制等多种特性。

下载地址: <http://echarts.baidu.com/download.html>



## 7.4 数据可视化图表

### 1. 快速体验ECharts

绘制图表：demo7-5.html 引入文件、准备容器

```
// 引入ECharts的JavaScript文件
<script src="echarts.js"> </script>
// 为ECharts准备一个DOM容器
<div id="box" style="width:400px;height:300px"> </div>
<script>
    // 编写图表代码
</script>
```





## 7.4 数据可视化图表

### 1. 快速体验ECharts

绘制图表: demo7-5.html 编写图表代码

初始化echarts对象

```
var myChart = echarts.init(document.getElementById('box'));  
myChart.setOption({  
  title: {text: '整点温度实况'},  
  color: '#675bba',  
  legend: {data: ['温度']}, // 图例数组的名称  
});
```

使用echarts对象调用setOption()方法设置图表的各项特性。



## 7.4 数据可视化图表

### 1. 快速体验ECharts

绘制图表: `setOption()`方法的参数项

参数项	描述
title	标题组件，包含主标题和副标题
color	调色盘颜色列表
legend	图例组件，用于展现不同series系列的标记、颜色和名字
tooltip	提示框组件
xAxis	设置直角坐标系grid中的X轴相关配置
yAxis	设置直角坐标系grid中的Y轴相关配置
series	设置图表的系列列表，每个系列通过type决定自己的图表类型



## 7.4 数据可视化图表

### 1. 快速体验ECharts

加载异步数据:

demo7-5.json

```
{  
  "h": [6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 2, 4],  
  "c": [3, 8, 12, 12, 16, 18, 16, 14, 10, 5, 4, 3]  
}
```

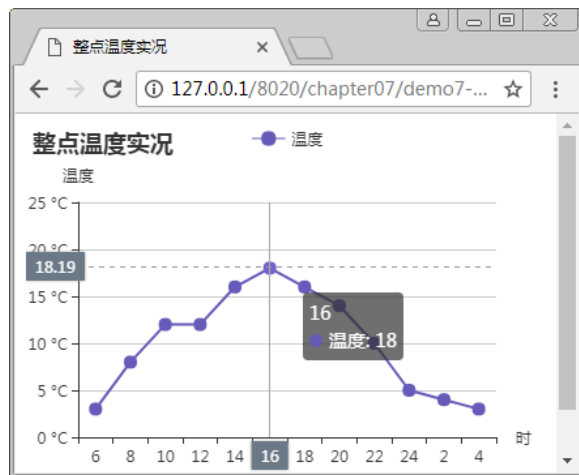


## 7.4 数据可视化图表

### 1. 快速体验ECharts

加载异步数据：发送Ajax请求的代码，将获取数据填入到echarts对应的参数项中。

```
$.get('demo7-5.json', function(data) {  
    myChart.setOption({  
        xAxis: {data: data.h},  
        series: [{name: '温度', data: data.c}]  
    });  
});
```





## 7.4 数据可视化图表

### 2. ECharts的常用配置项

图表类型：`series`配置项可同时指定一个或多个图表类型及相关配置，从而形成系列列表，每个系列都是通过`type`属性决定自己的图表类型。

例如：

参数项	描述
line	默认为折线图，设置 <code>areaStyle</code> 后可以绘制面积图。通过 <code>visualMap</code> 组件可通过不同颜色区分每个区间内的数据。可应用在直角坐标系和极坐标系上
bar	柱状或条形图，通过柱形的高度或条形的宽度表示数据的大小，要求直角坐标系中至少有一个类目轴或时间轴
pie	饼图，设置 <code>roseType</code> 可显示成南丁格尔图（通过半径大小区分数据的大小）

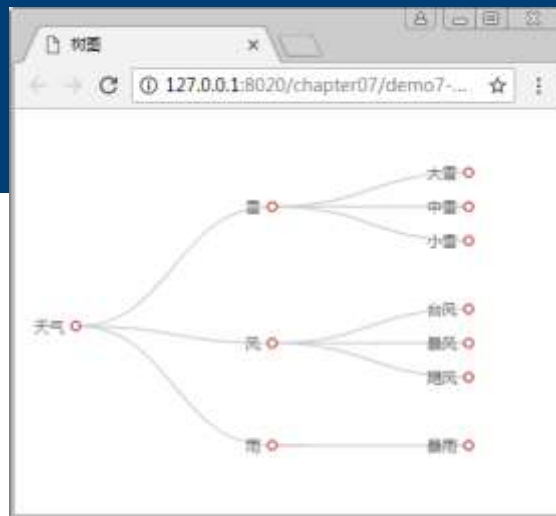


## 7.4 数据可视化图表

### 2. ECharts的常用配置项

图表类型：以具有层级结构的天气数据生成树状图表。

```
{  
  "name": "天气",  
  "children": [ {"name": "雪",  
    "children": [ {"name": "大雪"}, {"name": "中雪"}, {"name": "小雪"} ] }...  
]
```





## 7.4 数据可视化图表

### 2. ECharts的常用配置项

数据集：组件实现对数据的单独管理。

```
dataset: {
```

```
  source: [
```

```
    ['商品', '2017', '2018'],
```

```
    ['图书', 56.3, 85.9],
```

```
    ['服饰', 90, 95.6]
```

```
  ]
```

```
}
```

第一行识别为纬度数据

识别为图表具体的数据



## 7.4 数据可视化图表

### 2. ECharts的常用配置项

数据集：组件实现对数据的单独管理

```
dataset: {  
  dimensions: ['商品', '2017', '2018'],  
  source: [  
    {商品: '图书', '2017': 56.3, '2018': 85.9},  
    {商品: '服饰', '2017': 90, '2018': 95.6}  
  ]  
}
```

当指定dimensions属性时，该属性的数据识别为纬度数据，source属性中的数据都会被识别为具体数据





## 7.4 数据可视化图表

### 2. EChars的常用配置项

#### 工具栏

```
toolbox: {
```

```
  show: true, orient: 'vertical',
```

图标以垂直的方式展示

```
  feature: {
```

```
    dataZoom: {yAxisIndex: 'none'},
```

```
    dataView: {readOnly: false},
```

```
    magicType: {type: ['line', 'bar'] },
```

```
    restore: {},saveAsImage: {},
```

工具的配置项

```
  }
```

```
}
```



## 7.5 文件上传

### 1. 快速体验WebUploader

#### WebUploader

**描述：**采用AMD（Asynchronous Module Definition，异步模块定义）规范组织的文件上传组件

**地址：** <https://github.com/fex-team/webuploader/releases>



## 7.5 文件上传

### 1. 快速体验WebUploader

WebUploader包含了两个版本，分别为压缩版本（webuploader-0.1.5）和源码版本（Source code）。





## 7.5 文件上传

### 1. 快速体验WebUploader

参数项	描述
Uploader.swf	SWF文件，当使用Flash运行时需要引入
webuploader.css	Web Uploader提供的CSS样式文件
webuploader.html5only.js	仅适用于HTML5实现的版本
webuploader.html5only.min.js	仅适用于HTML5实现的迷你版本
webuploader.flashonly.js	仅适用于FLASH实现的版本
webuploader.flashonly.min.js	仅适用于FLASH实现的迷你版本
webuploader.js	WebUploader的完全版本
webuploader.min.js	WebUploader的完全迷你版本
webuploader.nolog.js	不开启日志功能的完全版本
webuploader.nolog.min.js	不开启日志功能的完全迷你版本
webuploader.withoutimage.js	去除图片处理版本，适用于HTML5和FLASH
webuploader.withoutimage.min.js	去除图片处理迷你版本，适用于HTML5和FLASH



## 7.5 文件上传

### 1. 快速体验WebUploader

使用WebUploader的文件引入方式:

```
<link rel="stylesheet" href="webuploader-0.1.5/webuploader.css">  
<script src="jquery-1.12.4.js"> </script>  
<script src="webuploader-  
0.1.5/webuploader.html5only.min.js"> </script>
```

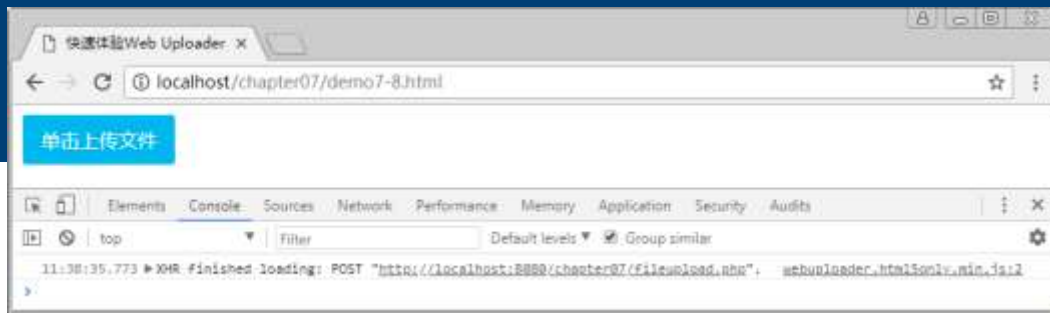


## 7.5 文件上传

### 1. 快速体验WebUploader

上传案例：

```
var uploader = WebUploader.create({  
    auto: true, // 选完文件后，是否自动上传  
    server: 'fileupload.php', // 文件接收服务端  
    pick: { // 指定选择按钮的容器  
        id: '#file_picker',  
        innerHTML: '单击上传文件',  
        multiple: false  
    }  
})
```





## 7.5 文件上传

### 1. 快速体验WebUploader

在服务器端接收上传文件: fileupload.php

```
<?php  
    var_dump($_FILES);  
?>
```



## 7.5 文件上传

### 2. 上传文件进度

**fileQueued事件：** 监控文件是否放入到上传的队列。

**uploadProgress事件：** 获取文件上传的进度。

// 上传文件被加入到队列后触发

```
uploader.on('fileQueued', function(file) {
```

```
.....
```

```
});
```

// 上传过程中触发，并且携带上传的进度值

```
uploader.on('uploadProgress', function( file, percentage) {
```

```
.....
```

```
});
```

上传的文件对象

数值类型的上传进度





## 7.5 文件上传

### 2. 显示上传文件进度

上传事件:

参数项	描述
uploadSuccess	上传成功
uploadError	上传失败
uploadComplete	上传完成（不论文件是否上传成功）



## 7.5 文件上传

### 3. 图片上传操作

上传事件：thumb可再详细设置缩略图的样式（大小、是否可裁剪等等）。

```
accept: { // 只允许选择图片文件
    .....
},
duplicate: true, // 允许重复上传同一张图片
thumb: { // 配置生成缩略图的选项
    .....
}
```



## 7.5 文件上传

### 3. 图片上传操作

上传事件：thumb可再详细设置缩略图的样式（大小、是否可裁剪等等）。

```
uploader.makeThumb(file, function(error, src) {  
  if (error) {  
    .....  
  }  
}, 100, 100);
```

生成缩略图错误



## 7.6 编辑器

### 1. 快速体验UEditor

#### UEditor

**描述：**UEditor是百度推出的一款编辑器，它的开源代码基于MIT协议，允许用户自由使用与修改。

**地址：** <http://ueditor.baidu.com/website/download.html>

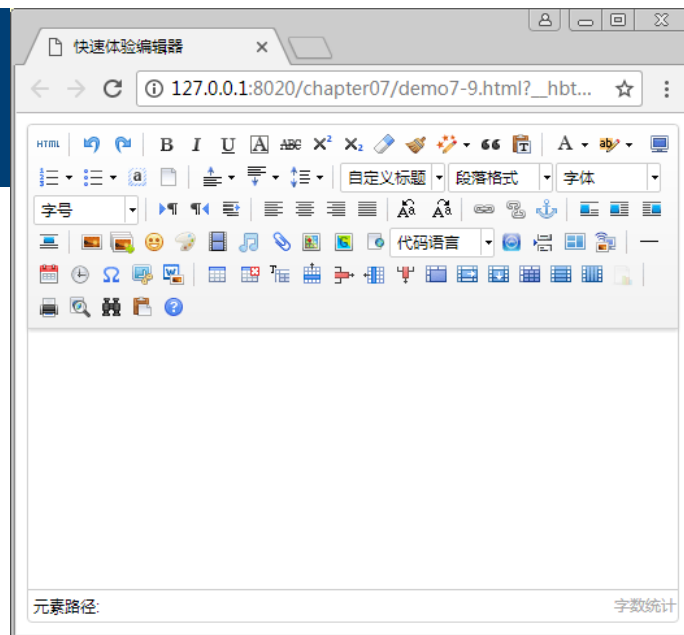


## 7.6 编辑器

### 1. 快速体验UEditor

调用**UE.getEditor()**方法将实例化的编辑器存放到id为content的元素容器内。

```
var ue = UE.getEditor('content');
```





## 7.6 编辑器

### 2. 定制工具栏图标

选项	含义	选项	含义	选项	含义
backcolor	背景色	blockquote	引用	bold	加粗
emotion	表情	fontsize	字号	fontfamily	字体
forecolor	字体颜色	formatmatch	格式刷	fullscreen	全屏
help	帮助	insertorderedlist	有序列表	insertrow	前插入行
italic	斜体	insertunorderedlist	无序列表	insertcol	前插入列
inserttable	插入表格	justifyjustify	两端对齐	justifyleft	居左对齐
justifyright	居右对齐	justifycenter	居中对齐	link	超链接
paragraph	段落格式	redo	重做	strikethrough	删除线
source	源代码	underline	下划线	undo	撤销

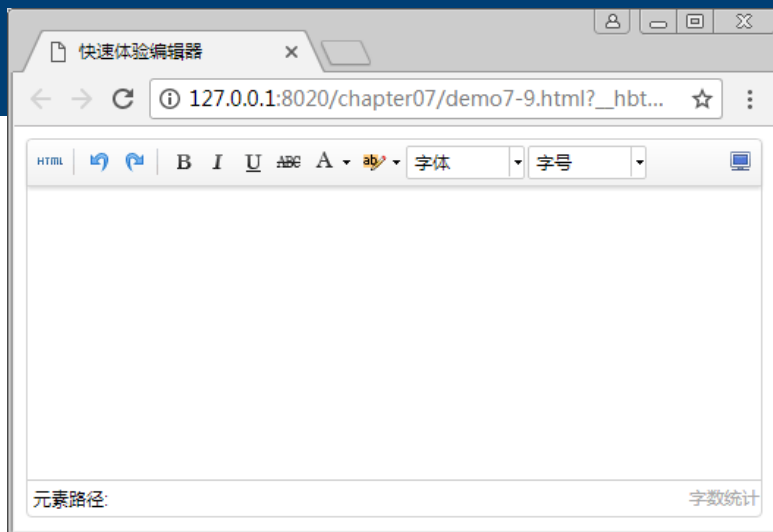


## 7.6 编辑器

### 2. 定制工具栏图标

修改`ueditor.config.js`配置项文件里面的`toolbars`。

```
UEEDITOR_CONFIG['toolbars'] = [['fullscreen', 'source', '|',  
    'undo', 'redo', '|', 'bold', 'italic', 'underline', 'strikethrough',  
    'forecolor', 'backcolor', 'fontfamily', 'fontsize'  
    ]];
```





## 7.6 编辑器

### 2. 定制工具栏图标

在实例化编辑器时设置**toolbars**参数。

```
var ue = UE.getEditor('content', {  
    'toolbars': [  
        ['fullscreen', 'source', '|', 'undo', 'redo', '|', 'bold', 'italic',  
        'underline', 'strikethrough', 'forecolor', 'backcolor', 'fontfamily',  
        'fontsize']  
    ]  
});
```





## 7.6 编辑器

### 3. UEditor常用方法

在单击按钮时，获取获取编辑器的内容。

```
ue.ready(function() {  
    ue.setContent('<i>请在此处编写内容!</i>'); // 设置编辑器的默认值  
    ue.setContent('<p>雄关漫道真如铁，而今迈步从头越。</p>', true));  
    $('#btn').click(function() {  
        var html = ue.getContent(); alert(html); // 为编辑器追加内容  
    });  
});
```





## 7.6 编辑器

### 3. UEditor常用方法

获取内容方法:

参数项	描述
getContent()	获取编辑器的内容
setContent()	设置编辑器的内容，可修改编辑器当前的html内容
getContentTxt()	获取纯文本内容
getPlainTxt()	获取保留段落格式的纯文本内容
hasContents()	判断编辑器是否有内容
selection.getText()	获得当前选中的文本



## 7.6 编辑器

### 3. UEditor常用方法

设置编辑器行为:

参数项	描述
focus()	让编辑器获得焦点
blur()	让编辑器失去焦点
isFocus()	判断编辑器是否获得焦点
setDisabled()	设置当前编辑区域不可编辑
setEnabled()	设置当前编辑区域可以编辑
setHide()	隐藏编辑器
setShow()	显示编辑器
execCommand()	执行命令的通用接口。例如，设置当前选区背景色



## 本章小结

本章首先讲解了什么是插件，以及jQuery中如何自定义插件的两种方式。然后以方便前端开发为目的，讲解了一系列的常用组件，包括模板引擎art-template、数据可视化图表ECharts、文件上传组件WebUploader以及编辑器UEditor。学习本章内容后，读者需要掌握至少一种插件的自定义方法，并能够根据项目需求灵活运用常用的开发组件。



# Thank You!

[yx.boxuegu.com](http://yx.boxuegu.com)

