**String 对象方法**

|  |  |
| --- | --- |
| **方法** | **描述** |
| [anchor()](http://www.w3school.com.cn/jsref/jsref_anchor.asp) | 创建 HTML 锚。 |
| [big()](http://www.w3school.com.cn/jsref/jsref_big.asp) | 用大号字体显示字符串。 |
| [blink()](http://www.w3school.com.cn/jsref/jsref_blink.asp) | 显示闪动字符串。 |
| [bold()](http://www.w3school.com.cn/jsref/jsref_bold.asp) | 使用粗体显示字符串。 |
| [charAt()](http://www.w3school.com.cn/jsref/jsref_charAt.asp) | 返回在指定位置的字符。 |
| [charCodeAt()](http://www.w3school.com.cn/jsref/jsref_charCodeAt.asp) | 返回在指定的位置的字符的 Unicode 编码。 |
| [concat()](http://www.w3school.com.cn/jsref/jsref_concat_string.asp) | 连接字符串。 |
| [fixed()](http://www.w3school.com.cn/jsref/jsref_fixed.asp) | 以打字机文本显示字符串。 |
| [fontcolor()](http://www.w3school.com.cn/jsref/jsref_fontcolor.asp) | 使用指定的颜色来显示字符串。 |
| [fontsize()](http://www.w3school.com.cn/jsref/jsref_fontsize.asp) | 使用指定的尺寸来显示字符串。 |
| [fromCharCode()](http://www.w3school.com.cn/jsref/jsref_fromCharCode.asp) | 从字符编码创建一个字符串。 |
| [indexOf()](http://www.w3school.com.cn/jsref/jsref_indexOf.asp) | 检索字符串。 |
| [italics()](http://www.w3school.com.cn/jsref/jsref_italics.asp) | 使用斜体显示字符串。 |
| [lastIndexOf()](http://www.w3school.com.cn/jsref/jsref_lastIndexOf.asp) | 从后向前搜索字符串。 |
| [link()](http://www.w3school.com.cn/jsref/jsref_link.asp) | 将字符串显示为链接。 |
| [localeCompare()](http://www.w3school.com.cn/jsref/jsref_localeCompare.asp) | 用本地特定的顺序来比较两个字符串。 |
| [match()](http://www.w3school.com.cn/jsref/jsref_match.asp) | 找到一个或多个正则表达式的匹配。 |
| [replace()](http://www.w3school.com.cn/jsref/jsref_replace.asp) | 替换与正则表达式匹配的子串。 |
| [search()](http://www.w3school.com.cn/jsref/jsref_search.asp) | 检索与正则表达式相匹配的值。 |
| [slice()](http://www.w3school.com.cn/jsref/jsref_slice_string.asp) | 提取字符串的片断，并在新的字符串中返回被提取的部分。 |
| [small()](http://www.w3school.com.cn/jsref/jsref_small.asp) | 使用小字号来显示字符串。 |
| [split()](http://www.w3school.com.cn/jsref/jsref_split.asp) | 把字符串分割为字符串数组。 |
| [strike()](http://www.w3school.com.cn/jsref/jsref_strike.asp) | 使用删除线来显示字符串。 |
| [sub()](http://www.w3school.com.cn/jsref/jsref_sub.asp) | 把字符串显示为下标。 |
| [substr()](http://www.w3school.com.cn/jsref/jsref_substr.asp) | 从起始索引号提取字符串中指定数目的字符。 |
| [substring()](http://www.w3school.com.cn/jsref/jsref_substring.asp) | 提取字符串中两个指定的索引号之间的字符。 |
| [sup()](http://www.w3school.com.cn/jsref/jsref_sup.asp) | 把字符串显示为上标。 |
| [toLocaleLowerCase()](http://www.w3school.com.cn/jsref/jsref_toLocaleLowerCase.asp) | 把字符串转换为小写。 |
| [toLocaleUpperCase()](http://www.w3school.com.cn/jsref/jsref_toLocaleUpperCase.asp) | 把字符串转换为大写。 |
| [toLowerCase()](http://www.w3school.com.cn/jsref/jsref_toLowerCase.asp) | 把字符串转换为小写。 |
| [toUpperCase()](http://www.w3school.com.cn/jsref/jsref_toUpperCase.asp) | 把字符串转换为大写。 |
| toSource() | 代表对象的源代码。 |
| [toString()](http://www.w3school.com.cn/jsref/jsref_toString_string.asp) | 返回字符串。 |
| [valueOf()](http://www.w3school.com.cn/jsref/jsref_valueOf_string.asp) | 返回某个字符串对象的原始值。 |

## Date 对象

Date 对象用于处理日期和时间。

### 创建 Date 对象的语法：

var myDate=new Date()

**注释：**Date 对象会自动把当前日期和时间保存为其初始值。

## Date 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| [constructor](http://www.w3school.com.cn/jsref/jsref_constructor_date.asp) | 返回对创建此对象的 Date 函数的引用。 |
| [prototype](http://www.w3school.com.cn/jsref/jsref_prototype_date.asp) | 使您有能力向对象添加属性和方法。 |

## Date 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [Date()](http://www.w3school.com.cn/jsref/jsref_Date.asp) | 返回当日的日期和时间。 |
| [getDate()](http://www.w3school.com.cn/jsref/jsref_getDate.asp) | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| [getDay()](http://www.w3school.com.cn/jsref/jsref_getDay.asp) | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| [getMonth()](http://www.w3school.com.cn/jsref/jsref_getMonth.asp) | 从 Date 对象返回月份 (0 ~ 11)。 |
| [getFullYear()](http://www.w3school.com.cn/jsref/jsref_getFullYear.asp) | 从 Date 对象以四位数字返回年份。 |
| [getYear()](http://www.w3school.com.cn/jsref/jsref_getYear.asp) | 请使用 getFullYear() 方法代替。 |
| [getHours()](http://www.w3school.com.cn/jsref/jsref_getHours.asp) | 返回 Date 对象的小时 (0 ~ 23)。 |
| [getMinutes()](http://www.w3school.com.cn/jsref/jsref_getMinutes.asp) | 返回 Date 对象的分钟 (0 ~ 59)。 |
| [getSeconds()](http://www.w3school.com.cn/jsref/jsref_getSeconds.asp) | 返回 Date 对象的秒数 (0 ~ 59)。 |
| [getMilliseconds()](http://www.w3school.com.cn/jsref/jsref_getMilliseconds.asp) | 返回 Date 对象的毫秒(0 ~ 999)。 |
| [getTime()](http://www.w3school.com.cn/jsref/jsref_getTime.asp) | 返回 1970 年 1 月 1 日至今的毫秒数。 |
| [getTimezoneOffset()](http://www.w3school.com.cn/jsref/jsref_getTimezoneOffset.asp) | 返回本地时间与格林威治标准时间 (GMT) 的分钟差。 |
| [getUTCDate()](http://www.w3school.com.cn/jsref/jsref_getUTCDate.asp) | 根据世界时从 Date 对象返回月中的一天 (1 ~ 31)。 |
| [getUTCDay()](http://www.w3school.com.cn/jsref/jsref_getUTCDay.asp) | 根据世界时从 Date 对象返回周中的一天 (0 ~ 6)。 |
| [getUTCMonth()](http://www.w3school.com.cn/jsref/jsref_getUTCMonth.asp) | 根据世界时从 Date 对象返回月份 (0 ~ 11)。 |
| [getUTCFullYear()](http://www.w3school.com.cn/jsref/jsref_getUTCFullYear.asp) | 根据世界时从 Date 对象返回四位数的年份。 |
| [getUTCHours()](http://www.w3school.com.cn/jsref/jsref_getUTCHours.asp) | 根据世界时返回 Date 对象的小时 (0 ~ 23)。 |
| [getUTCMinutes()](http://www.w3school.com.cn/jsref/jsref_getUTCMinutes.asp) | 根据世界时返回 Date 对象的分钟 (0 ~ 59)。 |
| [getUTCSeconds()](http://www.w3school.com.cn/jsref/jsref_getUTCSeconds.asp) | 根据世界时返回 Date 对象的秒钟 (0 ~ 59)。 |
| [getUTCMilliseconds()](http://www.w3school.com.cn/jsref/jsref_getUTCMilliseconds.asp) | 根据世界时返回 Date 对象的毫秒(0 ~ 999)。 |
| [parse()](http://www.w3school.com.cn/jsref/jsref_parse.asp) | 返回1970年1月1日午夜到指定日期（字符串）的毫秒数。 |
| [setDate()](http://www.w3school.com.cn/jsref/jsref_setDate.asp) | 设置 Date 对象中月的某一天 (1 ~ 31)。 |
| [setMonth()](http://www.w3school.com.cn/jsref/jsref_setMonth.asp) | 设置 Date 对象中月份 (0 ~ 11)。 |
| [setFullYear()](http://www.w3school.com.cn/jsref/jsref_setFullYear.asp) | 设置 Date 对象中的年份（四位数字）。 |
| [setYear()](http://www.w3school.com.cn/jsref/jsref_setYear.asp) | 请使用 setFullYear() 方法代替。 |
| [setHours()](http://www.w3school.com.cn/jsref/jsref_setHours.asp) | 设置 Date 对象中的小时 (0 ~ 23)。 |
| [setMinutes()](http://www.w3school.com.cn/jsref/jsref_setMinutes.asp) | 设置 Date 对象中的分钟 (0 ~ 59)。 |
| [setSeconds()](http://www.w3school.com.cn/jsref/jsref_setSeconds.asp) | 设置 Date 对象中的秒钟 (0 ~ 59)。 |
| [setMilliseconds()](http://www.w3school.com.cn/jsref/jsref_setMilliseconds.asp) | 设置 Date 对象中的毫秒 (0 ~ 999)。 |
| [setTime()](http://www.w3school.com.cn/jsref/jsref_setTime.asp) | 以毫秒设置 Date 对象。 |
| [setUTCDate()](http://www.w3school.com.cn/jsref/jsref_setUTCDate.asp) | 根据世界时设置 Date 对象中月份的一天 (1 ~ 31)。 |
| [setUTCMonth()](http://www.w3school.com.cn/jsref/jsref_setUTCMonth.asp) | 根据世界时设置 Date 对象中的月份 (0 ~ 11)。 |
| [setUTCFullYear()](http://www.w3school.com.cn/jsref/jsref_setUTCFullYear.asp) | 根据世界时设置 Date 对象中的年份（四位数字）。 |
| [setUTCHours()](http://www.w3school.com.cn/jsref/jsref_setutchours.asp) | 根据世界时设置 Date 对象中的小时 (0 ~ 23)。 |
| [setUTCMinutes()](http://www.w3school.com.cn/jsref/jsref_setUTCMinutes.asp) | 根据世界时设置 Date 对象中的分钟 (0 ~ 59)。 |
| [setUTCSeconds()](http://www.w3school.com.cn/jsref/jsref_setUTCSeconds.asp) | 根据世界时设置 Date 对象中的秒钟 (0 ~ 59)。 |
| [setUTCMilliseconds()](http://www.w3school.com.cn/jsref/jsref_setUTCMilliseconds.asp) | 根据世界时设置 Date 对象中的毫秒 (0 ~ 999)。 |
| [toSource()](http://www.w3school.com.cn/jsref/jsref_tosource_boolean.asp) | 返回该对象的源代码。 |
| [toString()](http://www.w3school.com.cn/jsref/jsref_toString_date.asp) | 把 Date 对象转换为字符串。 |
| [toTimeString()](http://www.w3school.com.cn/jsref/jsref_toTimeString.asp) | 把 Date 对象的时间部分转换为字符串。 |
| [toDateString()](http://www.w3school.com.cn/jsref/jsref_toDateString.asp) | 把 Date 对象的日期部分转换为字符串。 |
| [toGMTString()](http://www.w3school.com.cn/jsref/jsref_toGMTString.asp) | 请使用 toUTCString() 方法代替。 |
| [toUTCString()](http://www.w3school.com.cn/jsref/jsref_toUTCString.asp) | 根据世界时，把 Date 对象转换为字符串。 |
| [toLocaleString()](http://www.w3school.com.cn/jsref/jsref_toLocaleString.asp) | 根据本地时间格式，把 Date 对象转换为字符串。 |
| [toLocaleTimeString()](http://www.w3school.com.cn/jsref/jsref_toLocaleTimeString.asp) | 根据本地时间格式，把 Date 对象的时间部分转换为字符串。 |
| [toLocaleDateString()](http://www.w3school.com.cn/jsref/jsref_toLocaleDateString.asp) | 根据本地时间格式，把 Date 对象的日期部分转换为字符串。 |
| [UTC()](http://www.w3school.com.cn/jsref/jsref_utc.asp) | 根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数。 |
| [valueOf()](http://www.w3school.com.cn/jsref/jsref_valueOf_date.asp) | 返回 Date 对象的原始值。 |

function showTime(){

var now=new Date();

var year= now.getFullYear();

var month= now.getMonth()+1;

var day = now.getDate();

var h = now.getHours();

var m = now.getMinutes();

var s = now.getSeconds();

m=m<10?"0"+m:m;

s=s<10?"0"+s:s;

var weekday='星期'+'日一二三四五六'.charAt(now.getDay());

document.getElementById("show").innerHTML=""+year+"年"+month+"月"+day+"日 "+ weekday +h+":"+m+":"+s; t=setTimeout('showTime()',500);}

# JavaScript RegExp 对象

## RegExp 对象

RegExp 对象表示正则表达式，它是对字符串执行模式匹配的强大工具。

### 直接量语法

/pattern/attributes

### 创建 RegExp 对象的语法：

new RegExp(*pattern*, *attributes*);

### 参数

参数 *pattern* 是一个字符串，指定了正则表达式的模式或其他正则表达式。

参数 *attributes* 是一个可选的字符串，包含属性 "g"、"i" 和 "m"，分别用于指定全局匹配、区分大小写的匹配和多行匹配。ECMAScript 标准化之前，不支持 m 属性。如果 *pattern* 是正则表达式，而不是字符串，则必须省略该参数。

### 返回值

一个新的 RegExp 对象，具有指定的模式和标志。如果参数 *pattern* 是正则表达式而不是字符串，那么 RegExp() 构造函数将用与指定的 RegExp 相同的模式和标志创建一个新的 RegExp 对象。

如果不用 new 运算符，而将 RegExp() 作为函数调用，那么它的行为与用 new 运算符调用时一样，只是当 *pattern* 是正则表达式时，它只返回 *pattern*，而不再创建一个新的 RegExp 对象。

### 抛出

SyntaxError - 如果 *pattern* 不是合法的正则表达式，或 *attributes* 含有 "g"、"i" 和 "m" 之外的字符，抛出该异常。

TypeError - 如果 *pattern* 是 RegExp 对象，但没有省略 *attributes* 参数，抛出该异常。

## 修饰符

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| [i](http://www.w3school.com.cn/jsref/jsref_regexp_i.asp) | 执行对大小写不敏感的匹配。 |
| [g](http://www.w3school.com.cn/jsref/jsref_regexp_g.asp) | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

## 方括号

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [[abc]](http://www.w3school.com.cn/jsref/jsref_regexp_charset.asp) | 查找方括号之间的任何字符。 |
| [[^abc]](http://www.w3school.com.cn/jsref/jsref_regexp_charset_not.asp) | 查找任何不在方括号之间的字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符。 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符。 |
| [A-z] | 查找任何从大写 A 到小写 z 的字符。 |
| [adgk] | 查找给定集合内的任何字符。 |
| [^adgk] | 查找给定集合外的任何字符。 |
| (red|blue|green) | 查找任何指定的选项。 |

## 元字符

元字符（Metacharacter）是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| [.](http://www.w3school.com.cn/jsref/jsref_regexp_dot.asp) | 查找单个字符，除了换行和行结束符。 |
| [\w](http://www.w3school.com.cn/jsref/jsref_regexp_wordchar.asp) | 查找单词字符。 |
| [\W](http://www.w3school.com.cn/jsref/jsref_regexp_wordchar_non.asp) | 查找非单词字符。 |
| [\d](http://www.w3school.com.cn/jsref/jsref_regexp_digit.asp) | 查找数字。 |
| [\D](http://www.w3school.com.cn/jsref/jsref_regexp_digit_non.asp) | 查找非数字字符。 |
| [\s](http://www.w3school.com.cn/jsref/jsref_regexp_whitespace.asp) | 查找空白字符。 |
| [\S](http://www.w3school.com.cn/jsref/jsref_regexp_whitespace_non.asp) | 查找非空白字符。 |
| [\b](http://www.w3school.com.cn/jsref/jsref_regexp_begin.asp) | 匹配单词边界。 |
| [\B](http://www.w3school.com.cn/jsref/jsref_regexp_begin_not.asp) | 匹配非单词边界。 |
| \0 | 查找 NUL 字符。 |
| [\n](http://www.w3school.com.cn/jsref/jsref_regexp_newline.asp) | 查找换行符。 |
| \f | 查找换页符。 |
| \r | 查找回车符。 |
| \t | 查找制表符。 |
| \v | 查找垂直制表符。 |
| [\xxx](http://www.w3school.com.cn/jsref/jsref_regexp_octal.asp) | 查找以八进制数 xxx 规定的字符。 |
| [\xdd](http://www.w3school.com.cn/jsref/jsref_regexp_hex.asp) | 查找以十六进制数 dd 规定的字符。 |
| [\uxxxx](http://www.w3school.com.cn/jsref/jsref_regexp_unicode_hex.asp) | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

## 量词

|  |  |
| --- | --- |
| **量词** | **描述** |
| [n+](http://www.w3school.com.cn/jsref/jsref_regexp_onemore.asp) | 匹配任何包含至少一个 n 的字符串。 |
| [n\*](http://www.w3school.com.cn/jsref/jsref_regexp_zeromore.asp) | 匹配任何包含零个或多个 n 的字符串。 |
| [n?](http://www.w3school.com.cn/jsref/jsref_regexp_zeroone.asp) | 匹配任何包含零个或一个 n 的字符串。 |
| [n{X}](http://www.w3school.com.cn/jsref/jsref_regexp_nx.asp) | 匹配包含 X 个 n 的序列的字符串。 |
| [n{X,Y}](http://www.w3school.com.cn/jsref/jsref_regexp_nxy.asp) | 匹配包含 X 或 Y 个 n 的序列的字符串。 |
| [n{X,}](http://www.w3school.com.cn/jsref/jsref_regexp_nxcomma.asp) | 匹配包含至少 X 个 n 的序列的字符串。 |
| [n$](http://www.w3school.com.cn/jsref/jsref_regexp_ndollar.asp) | 匹配任何结尾为 n 的字符串。 |
| [^n](http://www.w3school.com.cn/jsref/jsref_regexp_ncaret.asp) | 匹配任何开头为 n 的字符串。 |
| [?=n](http://www.w3school.com.cn/jsref/jsref_regexp_nfollow.asp) | 匹配任何其后紧接指定字符串 n 的字符串。 |
| [?!n](http://www.w3school.com.cn/jsref/jsref_regexp_nfollow_not.asp) | 匹配任何其后没有紧接指定字符串 n 的字符串。 |

## RegExp 对象属性

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **描述** | **FF** | **IE** |
| [global](http://www.w3school.com.cn/jsref/jsref_regexp_global.asp) | RegExp 对象是否具有标志 g。 | 1 | 4 |
| [ignoreCase](http://www.w3school.com.cn/jsref/jsref_regexp_ignorecase.asp) | RegExp 对象是否具有标志 i。 | 1 | 4 |
| [lastIndex](http://www.w3school.com.cn/jsref/jsref_lastindex_regexp.asp) | 一个整数，标示开始下一次匹配的字符位置。 | 1 | 4 |
| [multiline](http://www.w3school.com.cn/jsref/jsref_multiline_regexp.asp) | RegExp 对象是否具有标志 m。 | 1 | 4 |
| [source](http://www.w3school.com.cn/jsref/jsref_source_regexp.asp) | 正则表达式的源文本。 | 1 | 4 |

## RegExp 对象方法

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [compile](http://www.w3school.com.cn/jsref/jsref_regexp_compile.asp) | 编译正则表达式。 | 1 | 4 |
| [exec](http://www.w3school.com.cn/jsref/jsref_exec_regexp.asp) | 检索字符串中指定的值。返回找到的值，并确定其位置。 | 1 | 4 |
| [test](http://www.w3school.com.cn/jsref/jsref_test_regexp.asp) | 检索字符串中指定的值。返回 true 或 false。 | 1 | 4 |

## 支持正则表达式的 String 对象的方法

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [search](http://www.w3school.com.cn/jsref/jsref_search.asp) | 检索与正则表达式相匹配的值。 | 1 | 4 |
| [match](http://www.w3school.com.cn/jsref/jsref_match.asp) | 找到一个或多个正则表达式的匹配。 | 1 | 4 |
| [replace](http://www.w3school.com.cn/jsref/jsref_replace.asp) | 替换与正则表达式匹配的子串。 | 1 | 4 |
| [split](http://www.w3school.com.cn/jsref/jsref_split.asp) | 把字符串分割为字符串数组。 | 1 | 4 |

JQuery

jQuery是一个JavaScript脚本库，不需要特别的安装，只需要我们在页面 <head> 标签内中，通过 script 标签引入 jQuery 库即可。

**jQuery的属性与样式之html()及.text()**

读取、修改元素的html结构或者元素的文本内容是常见的DOM操作，jQuery针对这样的处理提供了2个便捷的方法.html()与.text()

**.html()方法**

获取集合中第一个匹配元素的HTML内容 或 设置每一个匹配元素的html内容，具体有3种用法：

1. .html() 不传入值，就是获取集合中第一个匹配元素的HTML内容
2. .html( htmlString )  设置每一个匹配元素的html内容
3. .html( function(index, oldhtml) ) 用来返回设置HTML内容的一个函数

**注意事项：**

.html()方法内部使用的是DOM的innerHTML属性来处理的，所以在设置与获取上需要注意的一个最重要的问题，**这个操作是针对整个HTML内容（不仅仅只是文本内容）**

**.text()方法**

得到匹配元素集合中每个元素的文本内容结合，包括他们的后代，或设置匹配元素集合中每个元素的文本内容为指定的文本内容。，具体有3种用法：

1. .text() 得到匹配元素集合中每个元素的合并文本，包括他们的后代
2. .text( textString ) 用于设置匹配元素内容的文本
3. .text( function(index, text) ) 用来返回设置文本内容的一个函数

**jQuery的属性与样式之.val()**

jQuery中有一个.val()方法主要是用于处理表单元素的值，比如 input, select 和 textarea。

**.val()方法**

1. .val()无参数，获取匹配的元素集合中第一个元素的当前值
2. .val( value )，设置匹配的元素集合中每个元素的值
3. .val( function ) ，一个用来返回设置值的函数

**注意事项：**

1. 通过.val()处理select元素， 当没有选择项被选中，它返回null
2. .val()方法多用来设置表单的字段的值
3. 如果select元素有multiple（多选）属性，并且至少一个选择项被选中， .val()方法返回一个数组，这个数组包含每个选中选择项的值

**.html(),.text()和.val()的差异总结：**

1. .html(),.text(),.val()三种方法都是用来读取选定元素的内容；只不过.html()是用来读取元素的html内容（包括html标签），.text()用来读取元素的纯文本内容，包括其后代元素，.val()是用来读取表单元素的"value"值。其中.html()和.text()方法不能使用在表单元素上,而.val()只能使用在表单元素上；另外.html()方法使用在多个元素上时，只读取第一个元素；.val()方法和.html()相同，如果其应用在多个元素上时，只能读取第一个表单元素的"value"值，但是.text()和他们不一样，如果.text()应用在多个元素上时，将会读取所有选中元素的文本内容。
2. .html(htmlString),.text(textString)和.val(value)三种方法都是用来替换选中元素的内容，如果三个方法同时运用在多个元素上时，那么将会替换所有选中元素的内容。
3. .html(),.text(),.val()都可以使用回调函数的返回值来动态的改变多个元素的内容。

**jQuery的属性与样式之增加样式.addClass()**

通过动态改变类名（class），可以让其修改元素呈现出不同的效果。在HTML结构中里，多个class以空格分隔，当一个节点（或称为一个标签）含有多个class时，DOM元素响应的className属性获取的不是class名称的数组，而是一个含有空格的字符串，这就使得多class操作变得很麻烦。同样的jQuery开发者也考虑到这种情况，增加了一个.addClass()方法，用于动态增加class类名

**.addClass( className )方法**

1. .addClass( className ) : 为每个匹配元素所要增加的一个或多个样式名
2. .addClass( function(index, currentClass) ) : 这个函数返回一个或更多用空格隔开的要增加的样式名

**注意事项：**

.addClass()方法不会替换一个样式类名。它只是简单的添加一个样式类名到元素上

简单的描述下：在p元素增加一个newClass的样式

<p class="orgClass">

$("p").addClass("newClass")

那么p元素的class实际上是 class="orgClass newClass"样式只会在原本的类上继续增加，通过空格分隔

**jQuery的属性与样式之切换样式.toggleClass()**

在做某些效果的时候，可能会针对同一节点的某一个样式不断的切换，也就是addClass与removeClass的互斥切换，比如隔行换色效果

jQuery提供一个toggleClass方法用于简化这种互斥的逻辑，通过toggleClass方法动态添加删除Class，一次执行相当于addClass，再次执行相当于removeClass

**.toggleClass( )方法：**在匹配的元素集合中的每个元素上添加或删除一个或多个样式类,取决于这个样式类是否存在或值切换属性。即：如果存在（不存在）就删除（添加）一个类

1. .toggleClass( className )：在匹配的元素集合中的每个元素上用来切换的一个或多个（用空格隔开）样式类名
2. .toggleClass( className, switch )：一个布尔值，用于判断样式是否应该被添加或移除
3. .toggleClass( [switch ] )：一个用来判断样式类添加还是移除的 布尔值
4. .toggleClass( function(index, class, switch) [, switch ] )：用来返回在匹配的元素集合中的每个元素上用来切换的样式类名的一个函数。接收元素的索引位置和元素旧的样式类作为参数

**注意事项：**

1. toggleClass是一个互斥的逻辑，也就是通过判断对应的元素上是否存在指定的Class名，如果有就删除，如果没有就增加
2. toggleClass会保留原有的Class名后新增，通过空格隔开

**jQuery的属性与样式之样式操作.css()**

通过JavaScript获取dom元素上的style属性，我们可以动态的给元素赋予样式属性。在jQuery中我们要动态的修改style属性我们只要使用css()方法就可以实现了

**.css() 方法：获取元素样式属性的计算值或者设置元素的CSS属性**

**获取：**

1. .css( propertyName ) ：获取匹配元素集合中的第一个元素的样式属性的计算值
2. .css( propertyNames )：传递一组数组，返回一个对象结果

**设置：**

1. .css(propertyName, value )：设置CSS
2. .css( propertyName, function )：可以传入一个回调函数，返回取到对应的值进行处理
3. .css( properties )：可以传一个对象，同时设置多个样式

**注意事项：**

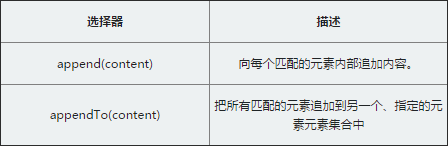
1. 浏览器属性获取方式不同，在获取某些值的时候都jQuery采用统一的处理，比如颜色采用RBG，尺寸采用px
2. .css()方法支持驼峰写法与大小写混搭的写法，内部做了容错的处理
3. 当一个数只被作为值（value）的时候， jQuery会将其转换为一个字符串，并添在字符串的结尾处添加px，例如 .css("width",50}) 与 .css("width","50px"})一样

JQ DOM

## DOM内部插入append()与appendTo()

动态创建的元素是不够的，它只是临时存放在内存中，最终我们需要放到页面文档并呈现出来。那么问题来了，怎么放到文档上？

这里就涉及到一个位置关系，常见的就是把这个新创建的元素，当作页面某一个元素的子元素放到其内部。针对这样的处理，jQuery就定义2个操作的方法

[](http://img.mukewang.com/56cc12f800017b4104480146.jpg)

append：这个操作与对指定的元素执行原生的appendChild方法，将它们添加到文档中的情况类似。

appendTo：实际上，使用这个方法是颠倒了常规的$(A).append(B)的操作，即不是把B追加到A中，而是把A追加到B中。

简单的总结就是：

.append()和.appendTo()两种方法功能相同，主要的不同是语法——内容和目标的位置不同

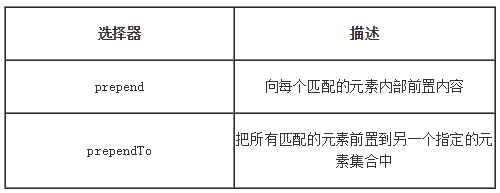
append()前面是被插入的对象，后面是要在对象内插入的元素内容

appendTo()前面是要插入的元素内容，而后面是被插入的对象

**DOM内部插入prepend()与prependTo()**

在元素内部进行操作的方法，除了在被选元素的结尾（仍然在内部）通过append与appendTo插入指定内容外，相应的还可以在被选元素之前插入，jQuery提供的方法是prepend与prependTo

选择器的描述：

[](http://img.mukewang.com/57481c3900013c6e05000193.jpg)

通过右边代码可以看到prepend与prependTo的使用及区别：

* .prepend()方法将指定元素插入到匹配元素里面作为它的第一个子元素 (如果要作为最后一个子元素插入用.append()).
* .prepend()和.prependTo()实现同样的功能，主要的不同是语法，插入的内容和目标的位置不同
* 对于.prepend() 而言，选择器表达式写在方法的前面，作为待插入内容的容器，将要被插入的内容作为方法的参数
* 而.prependTo() 正好相反，将要被插入的内容写在方法的前面，可以是选择器表达式或动态创建的标记，待插入内容的容器作为参数。

这里总结下内部操作四个方法的区别：

* append()向每个匹配的元素内部追加内容
* prepend()向每个匹配的元素内部前置内容
* appendTo()把所有匹配的元素追加到另一个指定元素的集合中
* prependTo()把所有匹配的元素前置到另一个指定的元素集合中

**DOM外部插入after()与before()**

节点与节点之前有各种关系，除了父子，祖辈关系，还可以是兄弟关系。之前我们在处理节点插入的时候，接触到了内部插入的几个方法，这节我们开始讲外部插入的处理，也就是兄弟之间的关系处理，这里jQuery引入了2个方法，可以用来在匹配I的元素前后插入内容

选择器的描述：

[](http://img.mukewang.com/57481b6b00018e3405210197.jpg)

* before与after都是用来对相对选中元素外部增加相邻的兄弟节点
* 2个方法都是都可以接收HTML字符串，DOM 元素，元素数组，或者jQuery对象，用来插入到集合中每个匹配元素的前面或者后面
* 2个方法都支持多个参数传递after(div1,div2,....) 可以参考右边案例代码

注意点：

* after向元素的后边添加html代码，如果元素后面有元素了，那将后面的元素后移，然后将html代码插入
* before向元素的前边添加html代码，如果元素前面有元素了，那将前面的元素前移，然后将html代码插

**DOM外部插入insertAfter()与insertBefore()**

与内部插入处理一样，jQuery由于内容目标的位置不同，然增加了2个新的方法insertAfter与insertBefore

[](http://img.mukewang.com/57481d230001b0f305170241.jpg)

* .before()和.insertBefore()实现同样的功能。主要的区别是语法——内容和目标的位置。 对于before()选择表达式在函数前面，内容作为参数，而.insertBefore()刚好相反，内容在方法前面，它将被放在参数里元素的前面
* .after()和.insertAfter() 实现同样的功能。主要的不同是语法——特别是（插入）内容和目标的位置。 对于after()选择表达式在函数的前面，参数是将要插入的内容。对于 .insertAfter(), 刚好相反，内容在方法前面，它将被放在参数里元素的后面
* before、after与insertBefore。insertAfter的除了目标与位置的不同外，后面的不支持多参数处理

注意事项：

* insertAfter将JQuery封装好的元素插入到指定元素的后面，如果元素后面有元素了，那将后面的元素后移，然后将JQuery对象插入；
* insertBefore将JQuery封装好的元素插入到指定元素的前面，如果元素前面有元素了，那将前面的元素前移，然后将JQuery对象插入；

## DOM节点删除之empty()的基本用法

要移除页面上节点是开发者常见的操作，jQuery提供了几种不同的方法用来处理这个问题，这里我们开仔细了解下empty方法

empty 顾名思义，清空方法，但是与删除又有点不一样，因为它只移除了 指定元素中的所有子节点。

这个方法不仅移除子元素（和其他后代元素），同样移除元素里的文本。因为，根据说明，元素里任何文本字符串都被看做是该元素的子节点。请看下面的HTML：

<div class="hello"><p>慕课网</p></div>

如果我们通过empty方法移除里面div的所有元素，它只是清空内部的html代码，但是标记仍然留在DOM中

//通过empty处理

$('.hello').empty()

//结果：<p>慕课网</p>被移除

<div class="hello"></div>

可以参考右边的代码区域：

通过empty移除了当前div元素下的所有p元素

但是本身id=test的div元素没有被删除

## DOM节点删除之remove()的有参用法和无参用法

remove与empty一样，都是移除元素的方法，但是remove会将元素自身移除，同时也会移除元素内部的一切，包括绑定的事件及与该元素相关的jQuery数据。

例如一段节点，绑定点击事件

<div class="hello"><p>慕课网</p></div>

$('.hello').on("click",fn)

如果不通过remove方法删除这个节点其实也很简单，但是同时需要把事件给销毁掉，这里是为了防止"内存泄漏"，所以前端开发者一定要注意，绑了多少事件，不用的时候一定要记得销毁

通过remove方法移除div及其内部所有元素，remove内部会自动操作事件销毁方法，所以使用使用起来非常简单

//通过remove处理

$('.hello').remove()

//结果：<div class="hello"><p>慕课网</p></div> 全部被移除

//节点不存在了,同事事件也会被销毁

**remove表达式参数：**

remove比empty好用的地方就是可以传递一个选择器表达式用来过滤将被移除的匹配元素集合，可以选择性的删除指定的节点

我们可以通过$()选择一组相同的元素，然后通过remove（）传递筛选的规则，从而这样处理

对比右边的代码区域，我们可以通过类似于这样处理

$("p").filter(":contains('3')").remove()

## DOM节点删除之保留数据的删除操作detach()

如果我们希望临时删除页面上的节点，但是又不希望节点上的数据与事件丢失，并且能在下一个时间段让这个删除的节点显示到页面，这时候就可以使用detach方法来处理

detach从字面上就很容易理解。让一个web元素托管。即从当前页面中移除该元素，但保留这个元素的内存模型对象。

来看看jquery官方文档的解释：

这个方法不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。与remove()不同的是，所有绑定的事件、附加的数据等都会保留下来。

$("div").detach()这一句会移除对象，仅仅是显示效果没有了。但是内存中还是存在的。当你append之后，又重新回到了文档流中。就又显示出来了。

当然这里要特别注意，detach方法是JQuery特有的，所以它只能处理通过JQuery的方法绑定的事件或者数据

参考右边的代码区域，通过 $("p").detach()把所有的P元素删除后，再通过append把删除的p元素放到页面上，通过点击文字，可以证明事件没有丢失

**DOM节点删除之detach()和remove()区别**

JQuery是一个很大强的工具库，在工作开发中，有些方法因为不常用到，或是没有注意到而被我们忽略。

remove()和detach()可能就是其中的一个，可能remove()我们用得比较多，而detach()就可能会很少了

 通过一张对比表来解释2个方法之间的不同

|  |  |  |  |
| --- | --- | --- | --- |
| 方法名 | 参数 | 事件及数据是否也被移除 | 元素自身是否被移除 |
| remove | 支持选择器表达 | 是 | 是（无参数时），有参数时要根据参数所涉及的范围 |
| detach | 参数同remove | 否 | 情况同remove |

**remove：**移除节点

* 无参数，移除自身整个节点以及该节点的内部的所有节点，包括节点上事件与数据
* 有参数，移除筛选出的节点以及该节点的内部的所有节点，包括节点上事件与数据

**detach：**移除节点

* 移除的处理与remove一致
* 与remove()不同的是，所有绑定的事件、附加的数据等都会保留下来
* 例如：$("p").detach()这一句会移除对象，仅仅是显示效果没有了。但是内存中还是存在的。当你append之后，又重新回到了文档流中。就又显示出来了。

具体可以参考右边的代码区域的对比

**DOM拷贝clone()**

克隆节点是DOM的常见操作，jQuery提供一个clone方法，专门用于处理dom的克隆

.clone()方法深度 复制所有匹配的元素集合，包括所有匹配元素、匹配元素的下级元素、文字节点。

clone方法比较简单就是克隆节点，但是需要注意，如果节点有事件或者数据之类的其他处理，我们需要通过clone(ture)传递一个布尔值ture用来指定，这样不仅仅只是克隆单纯的节点结构，还要把附带的事件与数据给一并克隆了

例如：

HTML部分

<div></div>

JavaScript部分

$("div").on('click', function() {//执行操作})

//clone处理一

$("div").clone() //只克隆了结构，事件丢失

//clone处理二

$("div").clone(true) //结构、事件与数据都克隆

使用上就是这样简单，使用克隆的我们需要额外知道的细节：

* clone()方法时，在将它插入到文档之前，我们可以修改克隆后的元素或者元素内容，如右边代码我 $(this).clone().css('color','red') 增加了一个颜色
* 通过传递true，将所有绑定在原始元素上的事件处理函数复制到克隆元素上
* clone()方法是jQuery扩展的，只能处理通过jQuery绑定的事件与数据
* 元素数据（data）内对象和数组不会被复制，将继续被克隆元素和原始元素共享。深复制的所有数据，需要手动复制每一个

**DOM替换replaceWith()和replaceAll()**

之前学习了节点的内插入、外插入以及删除方法，这节会学习替换方法replaceWith

**.replaceWith( newContent )**：用提供的内容替换集合中所有匹配的元素并且返回被删除元素的集合

简单来说：用$()选择节点A，调用replaceWith方法，传入一个新的内容B（HTML字符串，DOM元素，或者jQuery对象）用来替换选中的节点A

看个简单的例子：一段HTML代码

<div>

<p>第一段</p>

<p>第二段</p>

<p>第三段</p>

</div>

替换第二段的节点与内容

$("p:eq(1)").replaceWith('<a style="color:red">替换第二段的内容</a>')

通过jQuery筛选出第二个p元素，调用replaceWith进行替换，结果如下

<div>

<p>第一段</p>

<a style="color:red">替换第二段的内容</a>'

<p>第三段</p>

</div>

**.replaceAll( target ) ：**用集合的匹配元素替换每个目标元素

.replaceAll()和.replaceWith()功能类似，但是目标和源相反，用上述的HTML结构，我们用replaceAll处理

$('<a style="color:red">替换第二段的内容</a>').replaceAll('p:eq(1)')

总结：

* .replaceAll()和.replaceWith()功能类似，主要是目标和源的位置区别
* .replaceWith()与.replaceAll() 方法会删除与节点相关联的所有数据和事件处理程序
* .replaceWith()方法，和大部分其他jQuery方法一样，返回jQuery对象，所以可以和其他方法链接使用
* .replaceWith()方法返回的jQuery对象引用的是替换前的节点，而不是通过replaceWith/replaceAll方法替换后的节点

## DOM包裹wrap()方法

如果要将元素用其他元素包裹起来，也就是给它增加一个父元素，针对这样的处理，JQuery提供了一个wrap方法

**.wrap( wrappingElement )：**在集合中匹配的每个元素周围包裹一个HTML结构

简单的看一段代码：

<p>p元素</p>

给p元素增加一个div包裹

$('p').wrap('<div></div>')

最后的结构，p元素增加了一个父div的结构

<div>

<p>p元素</p>

</div>

**.wrap( function ) ：**一个回调函数，返回用于包裹匹配元素的 HTML 内容或 jQuery 对象

使用后的效果与直接传递参数是一样，只不过可以把代码写在函数体内部，写法不同而已

以第一个案例为例：

$('p').wrap(function() {

return '<div></div>'; //与第一种类似，只是写法不一样

})

**注意：**

.wrap()函数可以接受任何字符串或对象，可以传递给$()工厂函数来指定一个DOM结构。这种结构可以嵌套了好几层深，但应该只包含一个核心的元素。每个匹配的元素都会被这种结构包裹。该方法返回原始的元素集，以便之后使用链式方法。

## DOM包裹wrapInner()方法

如果要将合集中的元素内部所有的子元素用其他元素包裹起来，并当作指定元素的子元素，针对这样的处理，JQuery提供了一个wrapInner方法

**.wrapInner( wrappingElement )：**给集合中匹配的元素的内部，增加包裹的HTML结构

听起来有点绕，可以用个简单的例子描述下，简单的看一段代码：

<div>p元素</div>

<div>p元素</div>

给所有元素增加一个p包裹

$('div').wrapInner('<p></p>')

最后的结构，匹配的di元素的内部元素被p给包裹了

<div>

<p>p元素</p>

</div>

<div>

<p>p元素</p>

</div>

**.wrapInner( function ) ：**允许我们用一个callback函数做参数，每次遇到匹配元素时，该函数被执行，返回一个DOM元素，jQuery对象，或者HTML片段，用来包住匹配元素的内容

以上面案例为例，

$('div').wrapInner(function() {

return '<p></p>';

})

以上的写法的结果如下，等同于第一种处理了

<div>

<p>p元素</p>

</div>

<div>

<p>p元素</p>

</div>

注意：

当通过一个选择器字符串传递给.wrapInner() 函数，其参数应该是格式正确的 HTML，并且 HTML 标签应该是被正确关闭的。

## jQuery遍历之children()方法

jQuery是一个合集对象，如果想快速查找合集里面的第一级子元素，此时可以用children()方法。这里需要注意：.children(selector) 方法是返回匹配元素集合中每个元素的所有子元素（仅儿子辈，这里可以理解为就是父亲-儿子的关系）

**理解节点查找关系：**

<div class="div">

<ul class="son">

<li class="grandson">1</li>

</ul>

</div>

代码如果是$("div").children()，那么意味着只能找到ul，因为div与ul是父子关系，li与div是祖辈关系，因此无法找到。

**children()无参数**

允许我们通过在DOM树中对这些元素的直接子元素进行搜索，并且构造一个新的匹配元素的jQuery对象

**注意：jQuery是一个合集对象，所以通过children是匹配合集中每一给元素的第一级子元素**

.**children()方法选择性地接受同一类型选择器表达式**

$("div").children(".selected")

同样的也是因为jQuery是合集对象，可能需要对这个合集对象进行一定的筛选，找出目标元素，所以允许传一个选择器的表达式

具体的操作，请参考右边的代码

**jQuery遍历之find()方法**

jQuery是一个合集对象，如果想快速查找DOM树中的这些元素的后代元素，此时可以用find()方法，这也是开发使用频率很高的方法。这里要注意 children与find方法的区别，children是父子关系查找，find是后代关系（包含父子关系）

**理解节点查找关系：**

<div class="div">

<ul class="son">

<li class="grandson">1</li>

</ul>

</div>

代码如果是$("div").find("li")，此时，li与div是祖辈关系，通过find方法就可以快速的查找到了。

**.find()方法要注意的知识点：**

* find是遍历当前元素集合中每个元素的后代。只要符合，不管是儿子辈，孙子辈都可以。
* 与其他的树遍历方法不同，选择器表达式对于 .find() 是必需的参数。如果我们需要实现对所有后代元素的取回，可以传递通配选择器 '\*'。
* find只在后代中遍历，不包括自己。
* 选择器 context 是由 .find() 方法实现的；因此，$('.item-ii').find('li') 等价于 $('li', '.item-ii')(找到类名为item-ii的标签下的li标签)。

**注意重点：**

.find()和.children()方法是相似的

1.children只查找第一级的子节点

2.find查找范围包括子节点的所有后代节点

**jQuery遍历之closest()方法**

以选定的元素为中心，往内查找可以通过find、children方法。如果往上查找，也就是查找当前元素的父辈祖辈元素，jQuery提供了closest()方法，这个方法类似parents但是又有一些细微的区别，属于使用频率很高的方法

**closest()方法接受一个匹配元素的选择器字符串**

从元素本身开始，在DOM 树上逐级向上级元素匹配，并返回最先匹配的祖先元素

例如：在div元素中，往上查找所有的li元素，可以这样表达

$("div").closet("li')

**注意：jQuery是一个合集对象，所以通过closest是匹配合集中每一个元素的祖先元素**

**closest()方法给定的jQuery集合或元素来过滤元素**

同样的也是因为jQuery是合集对象，可能需要对这个合集对象进行一定的筛选，找出目标元素，所以允许传一个jQuery的对象

**注意事项：**在使用的时候需要特别注意下

粗看.parents()和.closest()是有点相似的，都是往上遍历祖辈元素，但是两者还是有区别的，否则就没有存在的意义了

1. 起始位置不同：.closest开始于当前元素 .parents开始于父元素
2. 遍历的目标不同：.closest要找到指定的目标，.parents遍历到文档根元素，closest向上查找，直到找到一个匹配的就停止查找，parents一直查找到根元素，并将匹配的元素加入集合
3. 结果不同：.closest返回的是包含零个或一个元素的jquery对象，parents返回的是包含零个或一个或多个元素的jquery对象

## jQuery遍历之next()方法

jQuery是一个合集对象，如果想快速查找指定元素集合中每一个元素紧邻的后面同辈元素的元素集合，此时可以用next()方法

**理解节点查找关系：**

如下class="item-1"元素就是红色部分，那蓝色的class="item-2"就是它的兄弟元素

<ul class="level-3">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

**next()无参数**

允许我们找遍元素集合中紧跟着这些元素的直接兄弟元素，并根据匹配的元素创建一个新的 jQuery 对象。

**注意：jQuery是一个合集对象，所以通过next匹配合集中每一个元素的下一个兄弟元素**

**next()方法选择性地接受同一类型选择器表达式**

同样的也是因为jQuery是合集对象，可能需要对这个合集对象进行一定的筛选，找出目标元素，所以允许传一个选择器的表达式

**$(selector).closest(element)**

**element包括：选择器(字符串)、DOM元素(Element)、jQuery对象。**

## jQuery遍历之prev()方法

jQuery是一个合集对象，如果想快速查找指定元素集合中每一个元素紧邻的前面同辈元素的元素集合，此时可以用prev()方法

**理解节点查找关系：**

如下蓝色的class="item-2"的li元素，红色的节点就是它的prev兄弟节点

<ul class="level-3">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

**prev()无参数**

取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合

**注意：jQuery是一个合集对象，所以通过prev是匹配合集中每一个元素的上一个兄弟元素**

**prev()方法选择性地接受同一类型选择器表达式**

**同样的也是因为jQuery是合集对象，可能需要对这个合集对象进行一定的筛选，找出目标元素，所以允许传一个选择器的表达式**

## jQuery遍历之siblings()

jQuery是一个合集对象，如果想快速查找指定元素集合中每一个元素的同辈元素，此时可以用siblings()方法

**理解节点查找关系：**

如下蓝色的class="item-2"的li元素，红色的节点就是它的siblings兄弟节点

<ul class="level-3">

<li class="item-1">1</li>

<li class="item-2">2</li>

<li class="item-3">3</li>

</ul>

**siblings()无参数**

取得一个包含匹配的元素集合中每一个元素的同辈元素的元素集合

**注意：jQuery是一个合集对象，所以通过siblings是匹配合集中每一个元素的同辈元素**

**siblings()方法选择性地接受同一类型选择器表达式**

**同样的也是因为jQuery是合集对象，可能需要对这个合集对象进行一定的筛选，找出目标元素，所以允许传一个选择器的表达式**

## jQuery遍历之add()方法

jQuery是一个合集对象，通过$()方法找到指定的元素合集后可以进行一系列的操作。$()之后就意味着这个合集对象已经是确定的，如果后期需要再往这个合集中添加一新的元素要如何处理？jQuery为此提供add方法，用来创建一个新的jQuery对象 ，元素添加到匹配的元素集合中

.add()的参数可以几乎接受任何的$()，包括一个jQuery选择器表达式，DOM元素，或HTML片段引用。

简单的看一个案例：

操作：选择所有的li元素，之后把p元素也加入到li的合集中

<ul>

<li>list item 1</li>

<li>list item 3</li>

</ul>

<p>新的p元素</p>

处理一：传递选择器

$('li').add('p')

处理二：传递dom元素

$('li').add(document.getElementsByTagName('p')[0])

还有一种方式，就是动态创建P标签加入到合集，然后插入到指定的位置，但是这样就改变元素的本身的排列了

$('li').add('<p>新的p元素</p>').appendTo(目标位置)

**JQ基础之事件篇**

**jQuery鼠标事件之click与dbclick事件**

用交互操作中，最简单直接的操作就是点击操作。jQuery提供了两个方法一个是click方法用于监听用户单击操作，另一个方法是dbclick方法用于监听用户双击操作。这两个方法的用法是类似的，下面以click()事件为例

使用上非常简单：

**方法一：$ele.click()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，用的比较少

<div id="test">点击触发<div>

$("ele").click(function(){

alert('触发指定事件')

})

$("#test").click(function(){

$("ele").click() //手动指定触发事件

});

**方法二：$ele.click( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数，这样可以针对事件的反馈做很多操作了，方法中的this是指向了绑定事件的元素

<div id="test">点击触发<div>

$("#test").click(function() {

//this指向 div元素

});

**方法三：$ele.click( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").click(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

dblclick()的用法和click()的用法是类似的，可以参考以上click()的用法。

**dbclick与click事件不同的是：**

**click事件触发需要以下几点：**

* click事件其实是由mousedown与mouseup 2个动作构成，所以点击的动作只有在松手后才触发

PS:mousedown和mouseup下一节会讲到

**dblclick事件触发需要以下几点：**

dblclick又是由2个click叠加而来的，所以dblclick事件只有在满足以下条件的情况下才能被触发

* 鼠标指针在元素里面时点击。
* 鼠标指针在元素里面时释放。
* 鼠标指针在元素里面时再次点击，点击间隔时间，是系统而定。
* 鼠标指针在元素里面时再次释放。

**注意：**在同一元素上同时绑定 click 和 dblclick 事件是不可取的。各个浏览器事件触发的顺序是不同的，一些浏览器在dblclick之前接受两个 click 事件 ，而一些浏览器只接受一个 click 事件。用户往往可通过不同的操作系统和浏览器配置双击灵敏度

**jQuery鼠标事件之mousedown与mouseup事件**

用户交互操作中，最简单直接的操作就是点击操作，因此jQuery提供了一个mousedown的快捷方法可以监听用户鼠标按下的操作，与其对应的还有一个方法mouseup快捷方法可以监听用户鼠标弹起的操作。**两种方法用法类似**，下面以mousedown()为例

使用上非常简单：

**方法一：$ele.**mousedown**()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，可能一般用的比较少

<div id="test">点击触发<div>

$("ele").mousedown(function(){

alert('触发指定事件')

})

$("#test").mousedown(function(){

$("ele").mousedown() //手动指定触发事件

});

**方法二：$ele.mousedown( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">点击触发<div>

$("#test").mousedown(function() {

//this指向 div元素

});

**方法三：$ele.mousedown( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").mousedown(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

**mousedown事件触发需要以下几点：**

* mousedown强调是按下触发
* 如果在一个元素按住了鼠标不放，并且拖动鼠标离开这个元素，并释放鼠标键，这仍然是算作mousedown事件
* 任何鼠标按钮被按下时都能触发mousedown事件
* 用event 对象的which区别按键，敲击鼠标左键which的值是1，敲击鼠标中键which的值是2，敲击鼠标右键which的值是3

**mouseup事件触发需要以下几点：**

* mouseup强调是松手触发，与mousedown是相反的
* mouseup与mousedown组合起来就是click事件
* 如果用户在一个元素上按下鼠标按键，并且拖动鼠标离开这个元素，然后释放鼠标键，这仍然是算作mouseup事件
* 任何鼠标按钮松手时都能触发mouseup事件
* 用event 对象的which区别按键，敲击鼠标左键which的值是1，敲击鼠标中键which的值是2，敲击鼠标右键which的值是3

**另外需要注意的是：**

**click与mousedown的区别：**

* click事件其实是由mousedown于mouseup 2个动作构成，所以点击的动作只有在松手后才触发

**jQuery鼠标事件之mousemove事件**

用交互操作中，经常需要知道用户是否有移动的操作。基于移动的机制可以做出拖动、拖拽一系列的效果出来。针对移动事件，jQuery提供了一个mousemove的快捷方法可以监听用户移动的的操作

使用上非常简单：

**方法一：$ele.mousemove()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，用的比较少

<div id="test">点击触发<div>

$("ele").mousemove(function(){

alert('触发指定事件')

})

$("#test").click(function(){

$("ele").mousemove() //指定触发事件

});

**方法二：$ele.mousemove( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">滑动触发<div>

$("#test").mousemove(function() {

//this指向 div元素

});

**方法三：$ele.mousemove( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").mousemove(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

**mousemove事件触发需要以下几点：**

* mousemove事件是当鼠标指针移动时触发的，即使是一个像素
* 如果处理器做任何重大的处理，或者如果该事件存在多个处理函数，这可能造成浏览器的严重的性能问题

## jQuery鼠标事件之mouseover与mouseout事件

在学JS的时候，大家还记得有两个方法叫移入移出事件吗？onmouseover()与onmouseout()事件~

jQuery当中同样提供了这样的事件来监听用户的移入移出操作，mouseover()与mouseout()事件，两者用法类似，下面一mouseover为例：

**方法一：$ele.mouseover()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，用的比较少

<div id="test">点击触发<div>

$("ele").mouseover(function(){

alert('触发指定事件')

})

$("#test").click(function(){

$("ele").mouseover() //指定触发事件

});

**方法二：$ele.mouseover( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">滑动触发<div>

$("#test").mouseover(function() {

//this指向 div元素

});

**方法三：$ele.mouseover( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").mouseover(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

**jQuery鼠标事件之mouseenter与mouseleave事件**

用交互操作中，经常需要知道用户操作鼠标是否有移到元素内部或是元素外部，因此jQuery提供了一个mouseenter和mouseleave的快捷方法可以监听用户移动到内部的操作

使用上非常简单，三种参数传递方式与mouseover和mouseout是一模一样的，所以这里不再重复，主要讲讲区别，下面以mouseenter为例：

mouseenter JavaScript事件是Internet Explorer专有的。由于该事件在平时很有用，jQuery的模拟这一事件，以便它可用于所有浏览器。该事件在鼠标移入到元素上时被触发。任何HTML元素都可以接受此事件。

**mouseenter事件和mouseover的区别**

**关键点就是：冒泡的方式处理问题**

简单的例子：

mouseover为例：

<div class="aaron2">

<p>鼠标离开此区域触发mouseleave事件</p>

</div>

如果在p元素与div元素都绑定mouseover事件，鼠标在离开p元素，但是没有离开div元素的时候，触发的结果:

1. p元素响应事件
2. div元素响应事件

这里的问题是div为什么会被触发？ 原因就是事件冒泡的问题，p元素触发了mouseover，他会一直往上找父元素上的mouseover事件，如果父元素有mouseover事件就会被触发

所以在这种情况下面，jQuery推荐我们使用 mouseenter事件

mouseenter事件只会在绑定它的元素上被调用，而不会在后代节点上被触发

这就是最本质的区别，具体的对应可以参考右边的案例：

PS：同学可以根据右侧代码，自己写出mouseout与mouseleave的区别哦~

**jQuery鼠标事件之hover事件**

学了mouseover、mouseout、mouseenter、mouseleave事件，也理解了四个事件的相同点与不同点，现在可以用来给元素做一个简单的切换效果

在元素上移进移出切换其换色，一般通过2个事件配合就可以达到，这里用mouseenter与mouseleave，这样可以避免冒泡问题

$(ele).mouseenter(function(){

$(this).css("background", '#bbffaa');

})

$(ele).mouseleave(function(){

$(this).css("background", 'red');

})

这样目的是达到了，代码稍微有点多，对于这样的简单逻辑jQuery直接提供了一个hover方法，可以便捷处理

只需要在hover方法中传递2个回调函数就可以了，不需要显示的绑定2个事件

$(selector).hover(handlerIn, handlerOut)

* handlerIn(eventObject)：当鼠标指针进入元素时触发执行的事件函数
* handlerOut(eventObject)：当鼠标指针离开元素时触发执行的事件函数

## jQuery鼠标事件之focusin事件

当一个元素，或者其内部任何一个元素获得焦点的时候，例如：input元素，用户在点击聚焦的时候，如果开发者需要捕获这个动作的时候，jQuery提供了一个focusin事件

使用上非常简单：

**方法一：$ele.focusin()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，一般用的比较少

<div id="test">点击触发<div>

$("ele").focusin(function(){

alert('触发指定事件')

})

$("#test").mouseup(function(){

$("ele").focusin() //指定触发事件

});

**方法二：$ele.focusin( handler )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">点击触发<div>

$("#test").focusin(function() {

//this指向 div元素

});

**方法三：$ele.focusin( [eventData ], handler )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").focusin(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

## jQuery鼠标事件之focusin事件

当一个元素，或者其内部任何一个元素获得焦点的时候，例如：input元素，用户在点击聚焦的时候，如果开发者需要捕获这个动作的时候，jQuery提供了一个focusin事件

使用上非常简单：

**方法一：$ele.focusin()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，一般用的比较少

<div id="test">点击触发<div>

$("ele").focusin(function(){

alert('触发指定事件')

})

$("#test").mouseup(function(){

$("ele").focusin() //指定触发事件

});

**方法二：$ele.focusin( handler )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">点击触发<div>

$("#test").focusin(function() {

//this指向 div元素

});

**方法三：$ele.focusin( [eventData ], handler )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").focusin(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

## jQuery鼠标事件之focusout事件

当一个元素，或者其内部任何一个元素失去焦点的时候，比如input元素，用户在点击失去焦的时候，如果开发者需要捕获这个动作，jQuery提供了一个focusout事件

使用上非常简单：

**方法一：$ele.focusout()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，可能一般用的比较少

<div id="test">点击触发<div>

$("ele").focusout(function(){

alert('触发指定事件')

})

$("#test").mouseup(function(){

$("ele").focusout() //指定触发事件

});

**方法二：$ele.focusout( handler )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<div id="test">点击触发<div>

$("#test").focusout(function() {

//this指向 div元素

});

**方法三：$ele.focusout( [eventData ], handler )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<div id="test">点击触发<div>

$("#test").focusout(11111,function(e) {

//this指向 div元素

//e.date => 11111 传递数据

});

## jQuery表单事件之blur与focus事件

在之前2.8与2.9节我们学过了表单处理事件focusin事件与focusout事件，同样用于处理表单焦点的事件还有blur与focus事件

它们之间的本质区别:

是否支持冒泡处理

举个简单的例子

<div>

<input type="text" />

</div>

其中input元素可以触发focus()事件

div是input的父元素，当它包含的元素input触发了focus事件时，它就产生了focusin()事件。

focus()在元素本身产生，focusin()在元素包含的元素中产生

blur与focusout也亦是如此

## jQuery表单事件之change事件

<input>元素，<textarea>和<select>元素的值都是可以发生改变的，开发者可以通过change事件去监听这些改变的动作

**input元素**

监听value值的变化，当有改变时，失去焦点后触发change事件。对于单选按钮和复选框，当用户用鼠标做出选择时，该事件立即触发。

**select元素**

对于下拉选择框，当用户用鼠标作出选择时，该事件立即触发

**textarea元素**

多行文本输入框，当有改变时，失去焦点后触发change事件

change事件很简单，无非就是注意下触发的先后行为，可以看看右边代码参考

## jQuery表单事件之select事件

当 textarea 或文本类型的 input 元素中的文本被选择时，会发生 select 事件。  
这个函数会调用执行绑定到select事件的所有函数，包括浏览器的默认行为。可以通过在某个绑定的函数中返回false来防止触发浏览器的默认行为。

select事件只能用于<input>元素与<textarea>元素

使用上非常简单：

**方法一：.select()**

触发元素的select事件:

$("input").select();

**方法二：$ele.select( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<input id="test" value="文字选中"></input>

$("#test").select(function() { //响应文字选中回调

//this指向 input元素

});

**方法三：$ele.select( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<input id="test" value="文字选中"></input>

$("#test").select(11111,function(e) {//响应文字选中回调

//this指向 div元素

//e.date  => 11111 传递数据

});

**jQuery表单事件之submit事件**

提交表单是一个最常见的业务需求，比如用户注册，一些信息的输入都是需要表单的提交。同样的有时候开发者需要在表单提交的时候过滤一些的数据、做一些必要的操作（例如：验证表单输入的正确性，如果错误就阻止提交，从新输入）此时可以通过submit事件，监听下提交表单的这个动作

使用上非常简单，与基本事件参数处理保持一致

**方法一：$ele.submit()**

绑定$ele元素，不带任何参数一般是用来指定触发一个事件，用的比较少

<div id="test">点击触发<div>

$("ele").submit(function(){

alert('触发指定事件')

})

$("#text").click(function(){

$("ele").submit() //指定触发事件

});

**方法二：$ele.submit( handler(eventObject) )**

绑定$ele元素，每次$ele元素触发点击操作会执行回调 handler函数

这样可以针对事件的反馈做很多操作了

<form id="target" action="destination.html">

  <input type="submit" value="Go" />

</form>

$("#target").submit(function() { //绑定提交表单触发

//this指向 from元素

});

**方法三：$ele.submit( [eventData ], handler(eventObject) )**

使用与方法二一致，不过可以接受一个数据参数，这样的处理是为了解决不同作用域下数据传递的问题

<form id="target" action="destination.html">

  <input type="submit" value="Go" />

</form>

$("#target").submit(11111,function(data) { //绑定提交表单触发

//data => 1111 //传递的data数据

});

通过在<form>元素上绑定submit事件，开发者可以监听到用户的提交表单的的行为

具体能触发submit事件的行为：

* <input type="submit">
* <input type="image">
* <button type="submit">
* 当某些表单元素获取焦点时，敲击Enter（回车键）

上述这些操作下，都可以截获submit事件。

这里需要特别注意：

form元素是有默认提交表单的行为，如果通过submit处理的话，需要禁止浏览器的这个默认行为

传统的方式是调用事件对象 e.preventDefault() 来处理， jQuery中可以直接在函数中最后结尾return false即可

jQuery处理如下：

$("#target").submit(function(data) {

return false; //阻止默认行为，提交表单

});

**键盘事件**

**jQuery键盘事件之keydown()与keyup()事件**

鼠标有mousedown,mouseup之类的事件，这是根据人的手势动作分解的2个触发行为。相对应的键盘也有这类事件，将用户行为分解成2个动作，键盘按下与松手，针对这样的2种动作，jQuery分别提供了对应keydown与keyup方法来监听

**keydown事件：**

当用户在一个元素上第一次按下键盘上字母键的时候，就会触发它。使用上非常简单，与基本事件参数处理保持一致，这里使用不在重复了，列出使用的方法

//直接绑定事件

$elem.keydown( handler(eventObject) )

//传递参数

$elem.keydown( [eventData ], handler(eventObject) )

//手动触发已绑定的事件

$elem.keydown()

**keyup事件：**

当用户在一个元素上第一次松手键盘上的键的时候，就会触发它。使用方法与keydown是一致的只是触发的条件是方法的

**注意：**

* keydown是在键盘按下就会触发
* keyup是在键盘松手就会触发
* 理论上它可以绑定到任何元素，但keydown/keyup事件只是发送到具有焦点的元素上，不同的浏览器中，可获得焦点的元素略有不同，但是表单元素总是能获取焦点，所以对于此事件类型表单元素是最合适的。

**jQuery键盘事件之keypress()事件**

在input元素上绑定keydown事件会发现一个问题：

每次获取的内容都是之前输入的，当前输入的获取不到

keydown事件触发在文字还没敲进文本框，这时如果在keydown事件中输出文本框中的文本，得到的是触发键盘事件前的文本，而keyup事件触发时整个键盘事件的操作已经完成，获得的是触发键盘事件后的文本

当浏览器捕获键盘输入时，还提供了一个keypress的响应，这个跟keydown是非常相似，这里使用请参考keydown这一节，具体说说不同点

keypress事件与keydown和keyup的主要区别

* 只能捕获单个字符，不能捕获组合键
* 无法响应系统功能键（如delete，backspace）
* 不区分小键盘和主键盘的数字字符

总而言之，

KeyPress主要用来接收字母、数字等ANSI字符，而 KeyDown 和 KeyUP 事件过程可以处理任何不被 KeyPress 识别的击键。诸如：功能键（F1-F12）、编辑键、定位键以及任何这些键和键盘换档键的组合等。

## on()的多事件绑定

之前学的鼠标事件，表单事件与键盘事件都有个特点，就是直接给元素绑定一个处理函数，所有这类事件都是属于快捷处理。翻开源码其实可以看到，所有的快捷事件在底层的处理都是通过一个"on"方法来实现的。jQuery on()方法是官方推荐的绑定事件的一个方法。

**基本用法：.on( events ,[ selector ] ,[ data ] ) 基本用法：.on( eventtypes ,[ selector ] ,[ data ] ,function)**

最常见的给元素绑定一个点击事件，对比一下快捷方式与on方式的不同

$("#elem").click(function(){}) //快捷方式

$("#elem").on('click',function(){}) //on方式

最大的不同点就是on是可以自定义事件名，当然不仅仅只是如何，继续往下看

**多个事件绑定同一个函数**

$("#elem").on("mouseover mouseout",function(){ });

通过空格分离，传递不同的事件名，可以同时绑定多个事件

**多个事件绑定不同函数**

$("#elem").on({

mouseover:function(){},

mouseout:function(){},

});

通过空格分离，传递不同的事件名，可以同时绑定多个事件，每一个事件执行自己的回调方法

**将数据传递到处理程序**

function greet( event ) {

alert( "Hello " + event.data.name ); //Hello 慕课网

}

$( "button" ).on( "click", {

name: "慕课网"

}, greet );

可以通过第二参数（对象），当一个事件被触发时，要传递给事件处理函数的

**卸载事件off()方法**

* 通过.on()绑定的事件处理程序
* 通过off() 方法移除该绑定

根据on绑定事件的一些特性，off方法也可以通过相应的传递组合的事件名，名字空间，选择器或处理函数来移除绑定在元素上指定的事件处理函数。当有多个过滤参数时，只有与这些参数完全匹配的事件处理函数才会被移除

绑定2个事件

$("elem").on("mousedown mouseup",fn)

删除一个事件

$("elem").off("mousedown")

删除所有事件

$("elem").off("mousedown mouseup")

快捷方式删除所有事件，这里不需要传递事件名了，节点上绑定的所有事件讲全部销毁

$("elem").off()

## jQuery事件对象的属性和方法

事件对象的属于与方法有很多，但是我们经常用的只有那么几个，这里我主要说下作用与区别

**event.type：获取事件的类型**

触发元素的事件类型

$("a").click(function(event) {

alert(event.type); // "click"事件

});

**event.pageX 和 event.pageY：获取鼠标当前相对于页面的坐标**

通过这2个属性，可以确定元素在当前页面的坐标值，鼠标相对于文档的左边缘的位置（左边）与 （顶边）的距离，简单来说是从页面左上角开始,即是以页面为参考点,不随滑动条移动而变化

**event.preventDefault() 方法：阻止默认行为**

这个用的特别多，在执行这个方法后，如果点击一个链接（a标签），浏览器不会跳转到新的 URL 去了。我们可以用 event.isDefaultPrevented() 来确定这个方法是否(在那个事件对象上)被调用过了

**event.stopPropagation() 方法：阻止事件冒泡**

事件是可以冒泡的，为防止事件冒泡到DOM树上，也就是不触发的任何前辈元素上的事件处理函数

**event.which：获取在鼠标单击时，单击的是鼠标的哪个键**

event.which 将 event.keyCode 和 event.charCode 标准化了。event.which也将正常化的按钮按下(mousedown 和 mouseupevents)，左键报告1，中间键报告2，右键报告3

**event.currentTarget : 在事件冒泡过程中的当前DOM元素**

冒泡前的当前触发事件的DOM对象, 等同于this.

**this和event.target的区别：**

js中事件是会冒泡的，所以this是可以变化的，但event.target不会变化，它永远是直接接受事件的目标DOM元素；

**.this和event.target都是dom对象**

如果要使用jquey中的方法可以将他们转换为jquery对象。比如this和$(this)的使用、event.target和$(event.target)的使用；

**jQuery自定义事件之trigger事件**

众所周知类似于mousedown、click、keydown等等这类型的事件都是浏览器提供的，通俗叫原生事件，这类型的事件是需要有交互行为才能被触发。

在jQuery通过on方法绑定一个原生事件

$('#elem').on('click', function() {

alert("触发系统事件")

});

alert需要执行的条件：必须有用户点击才可以。如果不同用户交互是否能在某一时刻自动触发该事件呢？ 正常来说是不可以的，但是jQuery解决了这个问题，提供了一个trigger方法来触发浏览器事件

所以我们可以这样：

$('#elem').trigger('click');

在绑定on的事件元素上，通过trigger方法就可以调用到alert了，挺简单！

**再来看看.trigger是什么？**

**简单来讲就是：根据绑定到匹配元素的给定的事件类型执行所有的处理程序和行为**

trigger除了能够触发浏览器事件，同时还支持自定义事件，并且自定义时间还支持传递参数

$('#elem').on('Aaron', function(event,arg1,arg2) {

    alert("自触自定义时间")

 });

$('#elem').trigger('Aaron',['参数1','参数2'])

**trigger触发浏览器事件与自定义事件区别？**

* **自定义事件对象，是jQuery模拟原生实现的**
* **自定义事件可以传递参数**

**jQuery自定义事件之triggerHandler事件**

trigger事件还有一个特性：**会在DOM树上冒泡**，所以如果要阻止冒泡就需要在事件处理程序中返回false或调用事件对象中的.stopPropagation() 方法可以使事件停止冒泡

trigger事件是具有触发原生与自定义能力的，但是存在一个不可避免的问题**： 事件对象event无法完美的实现**，毕竟一个是浏览器给的，一个是自己模拟的。尽管 .trigger() 模拟事件对象，但是它并没有完美的复制自然发生的事件，若要触发通过 jQuery 绑定的事件处理函数，而不触发原生的事件，使用.triggerHandler() 来代替

triggerHandler与trigger的用法是一样的，重点看不同之处：

* triggerHandler不会触发浏览器的默认行为，.triggerHandler( "submit" )将不会调用表单上的.submit()
* .trigger() 会影响所有与 jQuery 对象相匹配的元素，而 .triggerHandler() 仅影响第一个匹配到的元素
* 使用 .triggerHandler() 触发的事件，并不会在 DOM 树中向上冒泡。 如果它们不是由目标元素直接触发的，那么它就不会进行任何处理
* 与普通的方法返回 jQuery 对象(这样就能够使用链式用法)相反，.triggerHandler() 返回最后一个处理的事件的返回值。如果没有触发任何事件，会返回 undefined

Jq动画

## jQuery中隐藏元素的hide方法

让页面上的元素不可见，一般可以通过设置css的display为none属性。但是通过css直接修改是静态的布局，如果在代码执行的时候，一般是通过js控制元素的style属性，这里jQuery提供了一个快捷的方法.hide()来达到这个效果

$elem.hide()

**提供参数：**

.hide( options )

当提供hide方法一个参数时，.hide()就会成为一个动画方法。.hide()方法将会匹配元素的宽度，高度，以及不透明度，同时进行动画操作

**快捷参数：**

.hide("fast / slow")

这是一个动画设置的快捷方式，'fast' 和 'slow' 分别代表200和600毫秒的延时，就是元素会执行200/600毫秒的动画后再隐藏

**注意：**

jQuery在做hide操作的时候，是会保存本身的元素的原始属性值，再之后通过对应的方法还原的时候还是初始值。比如一个元素的display属性值为inline，那么隐藏再显示时，这个元素将再次显示inline。一旦透明度 达到0，display样式属性将被设置为none，这个元素将不再在页面中影响布局

**jQuery中显示元素的show方法**

css中有display:none属性，同时也有display:block，所以jQuery同样提供了与hide相反的show方法

方法的使用几乎与hide是一致的，hide是让元素显示到隐藏，show则是相反，让元素从隐藏到显示

看一段代码：使用上一致，结果相反

$('elem').hide(3000).show(3000)

让元素执行3秒的隐藏动画，然后执行3秒的显示动画。

show与hide方法是非常常用的，但是一般很少会基于这2个属性执行动画，大多情况下还是直接操作元素的显示与隐藏为主

**注意事项：**

* show与hide方法是修改的display属性，通过是visibility属性布局需要通过css方法单独设置
* 如果使用!important在你的样式中，比如display: none !important，如果你希望.show()方法正常工作，必须使用.css('display', 'block !important')重写样式
* 如果让show与hide成为一个动画，那么默认执行动画会改变元素的高度，高度，透明度

**jQuery中显示与隐藏切换toggle方法**

show与hide是一对互斥的方法。需要对元素进行显示隐藏的互斥切换，通常情况是需要先判断元素的display状态，然后调用其对应的处理方法。比如显示的元素，那么就要调用hide，反之亦然。 对于这样的操作行为，jQuery提供了一个便捷方法toggle用于切换显示或隐藏匹配元素

**基本的操作：toggle();**

这是最基本的操作，处理元素显示或者隐藏，因为不带参数，所以没有动画。通过改变CSS的display属性，匹配的元素将被立即显示或隐藏，没有动画。

* 如果元素是最初显示，它会被隐藏
* 如果隐藏的，它会显示出来

display属性将被储存并且需要的时候可以恢复。如果一个元素的display值为inline，然后是隐藏和显示，这个元素将再次显示inline

**提供参数：.toggle( [duration ] [, complete ] )**

同样的提供了时间、还有动画结束的回调。在参数对应的时间内，元素会发生显示/隐藏的改变，在改变的过程中会把元素的高、宽、不透明度进行一系列动画效果。这个元素其实就是show与hide的方法

**直接定位：.toggle(display)**

直接提供一个参数，指定要改变的元素的最终效果

其实就是确定是使用show还是hide方法

if ( display === true ) {

$( "elem" ).show();

} else if ( display === false ) {

$( "elem" ).hide();

}

toggle方法就是show与hide的相互切换的一个快捷方法

**jQuery中下拉动画slideDown**

对于隐藏的元素，在将其显示出来的过程中，可以对其进行一些变化的动画效果。之前学过了show方法，show方法在显示的过程中也可以有动画，但是.show()方法将会匹配元素的宽度，高度，以及不透明度，同时进行动画操作。这里将要学习一个新的显示方法slideDown方法

**.slideDown()：用滑动动画显示一个匹配元素**

.slideDown()方法将给匹配元素的高度的动画，这会导致页面的下面部分滑下去，弥补了显示的方式

常见的操作，提供一个动画是时间，然后传递一个回调，用于知道动画是什么时候结束

**.slideDown( [duration ] [, complete ] )**

持续时间（duration）是以毫秒为单位的，数值越大，动画越慢，不是越快。字符串 'fast' 和 'slow' 分别代表200和600毫秒的延时。如果提供任何其他字符串，或者这个duration参数被省略，那么默认使用400 毫秒的延时。

具体使用：

$("ele").slideDown(1000, function() {

//等待动画执行1秒后,执行别的动作....

});

**注意事项：**

* 下拉动画是从无到有，所以一开始元素是需要先隐藏起来的，可以设置display:none
* 如 果提供回调函数参数，callback会在动画完成的时候调用。将不同的动画串联在一起按顺序排列执行是非常有用的。这个回调函数不设置任何参数，但是 this会设成将要执行动画的那个DOM元素，如果多个元素一起做动画效果，那么要非常注意，回调函数会在每一个元素执行完动画后都执行一次，而不是这组 动画整体才执行一次

## jQuery中上卷动画slideUp

对于显示的元素，在将其隐藏的过程中，可以对其进行一些变化的动画效果。之前学过了hide方法，hide方法在显示的过程中也可以有动画，但 是.hide()方法将为匹配元素的宽度，高度，以及不透明度，同时进行动画操作。这里将要学习一个新的显示方法slideUp方法

最简单的使用：不带参数

**$("elem").slideUp();**

这个使用的含义就是：找到元素的高度，然后采用一个下滑动画让元素一直滑到隐藏，当高度为0的时候，也就是不可见的时，修改元素display 样式属性被设置为none。这样就能确保这个元素不会影响页面布局了

带参数：

**.slideUp( [duration ] [, easing ] [, complete ] )**

同样可以提供一个时间，然后可以使用一种过渡使用哪种缓动函数，jQuery默认就2种，可以通过下载插件支持。最后一个动画结束的回调方法。

**因为动画是异步的，所以要在动画之后执行某些操作就必须要写到回调函数里面，这里要特别注意**

**jQuery中上卷下拉切换slideToggle**

slideDown与slideUp是一对相反的方法。需要对元素进行上下拉卷效果的切换，jQuery提供了一个便捷方法slideToggle用滑动动画显示或隐藏一个匹配元素

**基本的操作：slideToggle();**

这是最基本的操作，获取元素的高度，使这个元素的高度发生改变，从而让元素里的内容往下或往上滑。

**提供参数：.slideToggle( [duration ] ,[ complete ] )**

同样的提供了时间、还有动画结束的回调。在参数对应的时间内，元素会完成动画，然后出发回调函数

同时也提供了时间的快速定义，字符串 'fast' 和 'slow' 分别代表200和600毫秒的延时

slideToggle("fast")

slideToggle("slow")

**注意：**

* display属性值保存在jQuery的数据缓存中，所以display可以方便以后可以恢复到其初始值
* 当一个隐藏动画后，高度值达到0的时候，display 样式属性被设置为none，以确保该元素不再影响页面布局

## jQuery中淡出动画fadeOut

让元素在页面不可见，常用的办法就是通过设置样式的display:none。除此之外还可以一些类似的办法可以达到这个目的。这里要提一个透明度的方法，设置元素透明度为0，可以让元素不可见，透明度的参数是0~1之间的值，通过改变这个值可以让元素有一个透明度的效果。常见的淡入淡出动画正是这样的原理。

**fadeOut()函数用于隐藏所有匹配的元素，并带有淡出的过渡动画效果**

所谓"淡出"隐藏的，元素是隐藏状态不对作任何改变，元素是可见的，则将其隐藏。

.fadeOut( [duration ], [ complete ] )

通过不透明度的变化来实现所有匹配元素的淡出效果，并在动画完成后可选地触发一个回调函数。这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。

字符串 'fast' 和 'slow' 分别代表200和600毫秒的延时。如果提供任何其他字符串，或者这个duration参数被省略，那么默认使用400毫秒的延时

## jQuery中淡入效果fadeTo

淡入淡出fadeIn与fadeOut都是修改元素样式的opacity属性，但是他们都有个共同的特点，变化的区间要么是0，要么是1

fadeIn：淡入效果，内容显示，opacity是0到1

fadeOut：淡出效果，内容隐藏，opacity是1到0

如果要让元素保持动画效果，执行opacity = 0.5的效果时，要如何处理？

如果不考虑CSS3，我们用JS实现的话，基本就是通过定时器，在设定的时间内一点点的修改opacity的值，最终为0.5，原理虽说简单，但是总不如一键设置这么舒服，jQuery提供了fadeTo方法，可以让改变透明度一步到位

**语法**

**.fadeTo( duration, opacity ,callback)**

必需的 duration参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。fadeTo() 方法中必需的 opacity 参数将淡入淡出效果设置为给定的不透明度（值介于 0 与 1 之间）。可选的 callback 参数是该函数完成后所执行的函数名称。

## jQuery中淡入淡出切换fadeToggle

fadeToggle()函数用于切换所有匹配的元素，并带有淡入/淡出的过渡动画效果。之前也学过toggle、slideToggle 也是类似的处理方式

**fadeToggle切换fadeOut与fadeIn效果，所谓"切换"，即如果元素当前是可见的，则将其隐藏(淡出)；如果元素当前是隐藏的，则使其显示(淡入)。**

**常用语法：.fadeToggle( [duration ] ,[ complete ] )**

可选的 **duration**参数规定效果的时长。它可以取以下值："slow"、"fast" 或毫秒。 可选的 callback 参数是 fadeToggle完成后所执行的函数名称。

fadeToggle() 方法可以在 fadeIn() 与 fadeOut() 方法之间进行切换。如果元素已淡出，则 fadeToggle() 会向元素添加淡入效果。如果元素已淡入，则 fadeToggle() 会向元素添加淡出效果。

## jQuery中动画animate(上)

有些复杂的动画通过之前学到的几个动画函数是不能够实现，这时候就需要强大的animate方法了

操作一个元素执行3秒的淡入动画，对比一下2组动画设置的区别

$(elem).fadeOut(3000)

$(elem).animate({

opacity:0

},3000)

显而易见，animate方法更加灵活了，可以精确的控制样式属性从而执行动画

**语法：**

.animate( properties ,[ duration ], [ easing ], [ complete ] )

.animate( properties, options )

.animate()方法允许我们在任意的数值的CSS属性上创建动画。2种语法使用，几乎差不多了，唯一必要的属性就是一组CSS属性键值对。这组属性和用于设置.css()方法的属性键值对类似，除了属性范围做了更多限制。第二个参数开始可以单独传递多个实参也可以合并成一个对象传递了

**参数分解：**

**properties**：一个或多个css属性的键值对所构成的Object对象。要特别注意所有用于动画的属性**必须是数字的**，除非另有说明；这些属性如果不是数字的将不能使用基本的jQuery功能。比如常见的，border、margin、padding、width、height、font、left、top、right、bottom、wordSpacing等等这些都是能产生动画效果的。background-color很明显不可以，因为参数是red或者GBG这样的值，非常用插件，否则正常情况下是不能只用动画效果的。注意，CSS 样式使用 DOM 名称（比如 "fontSize"）来设置，而非 CSS 名称（比如 "font-size"）。

特别注意单位，属性值的单位像素（px）,除非另有说明。单位em 和 %需要指定使用

.animate({

left: 50,

  width: '50px'

  opacity: 'show',

  fontSize: "10em",

}, 500);

除了定义数值，每个属性能使用'show', 'hide', 和 'toggle'。这些快捷方式允许定制隐藏和显示动画用来控制元素的显示或隐藏

.animate({

width: "toggle"

});

如果提供一个以+= 或 -=开始的值，那么目标值就是以这个属性的当前值加上或者减去给定的数字来计算的

.animate({

    left: '+=50px'

}, "slow");

**duration时间**

动画执行的时间，持续时间是以毫秒为单位的；值越大表示动画执行的越慢，不是越快。还可以提供'fast' 和 'slow'字符串，分别表示持续时间为200 和 600毫秒。

**easing动画运动的算法**

jQuery库中默认调用 swing。如果需要其他的动画算法，请查找相关的插件

**complete回调**

动画完成时执行的函数，这个可以保证当前动画确定完成后发会触发

<script type="text/javascript">

$("#exec").click(function() {

var v = $("#animation").val();

var $aaron = $("#aaron");

if (v == "1") {

//观察每一次动画的改变

$aaron.animate({

height: '50'

}, {

duration :2000,

//每一个动画都会调用

step: function(now, fx) {

$aaron.text('高度的改变值:'+now)

}

})

} else if (v == "2") {

//观察每一次进度的变化

$aaron.animate({

height: '50'

}, {

duration :2000,

//每一步动画完成后调用的一个函数，

//无论动画属性有多少，每个动画元素都执行单独的函数

progress: function(now, fx) {

$aaron.text('进度:'+arguments[1])

// var data = fx.elem.id + ' ' + fx.prop + ': ' + now;

// alert(data)

}

})

}

});

</script>

**主要在fx这个参数上：**

**step：**fx返回的是和css相关的属性；

**progress：** fx返回的是动画进度本身的属性；

now:是当前动画正在改变的属性的实时值；

fx: jQuery.fx 原型对象的一个引用,其中包含了多项属性,比如

 执行动画的元素：elem;

动画正在改变的属性：prop;

正在改变属性的当前值：now;

正在改变属性的结束值：end;

 正在改变属性的单位：unit;等

   step:function(now,fx){

        console.log("返回的CSS属性是："+fx.prop);

        console.log("属性初始值："+fx.start);

        console.log("属性结束值："+fx.end);

        console.log("属性当前值："+fx.now);

    }

**jQuery中停止动画stop**

动画在执行过程中是允许被暂停的，当一个元素调用.stop()方法，当前正在运行的动画（如果有的话）立即停止

**语法：**

.stop( [clearQueue ], [ jumpToEnd ] )

.stop( [queue ], [ clearQueue ] ,[ jumpToEnd ] )

stop还有几个可选的参数，简单来说可以这3种情况

* .stop(); 停止当前动画，点击在暂停处继续开始
* .stop(true); 如果同一元素调用多个动画方法，尚未被执行的动画被放置在元素的效果队列中。这些动画不会开始，直到第一个完成。当调用.stop()的时候，队列中的下一个动画立即开始。如果clearQueue参数提供true值,那么在队列中的动画其余被删除并永远不会运行
* .stop(true,true); 当前动画将停止，但该元素上的 CSS 属性会被立刻修改成动画的目标值

简单的说：参考下面代码、

$("#aaron").animate({

height: 300

}, 5000)

$("#aaron").animate({

width: 300

}, 5000)

$("#aaron").animate({

opacity: 0.6

}, 2000)

1. stop()：只会停止第一个动画，第二个第三个继续
2. stop(true)：停止第一个、第二个和第三个动画
3. stop(true ture)：停止动画，直接跳到第一个动画的最终状态

## jQuery中each方法的应用

jQuery中有个很重要的核心方法each，大部分jQuery方法在内部都会调用each，其主要的原因的就是jQuery的实例是一个元素合集

如下：找到所有的div，并且都设置样式，css只是一个方法，所以内部会调用each处理这个div的合集，给每个div都设置style属性

$('div').css(...)

jQuery的大部分方法都是针元素合集的操作，所以jQuery会提供$(selector).each()来遍历jQuery对象

.each只是处理jQuery对象的方法，jQuery还提供了一个通用的jQuery.each方法，用来处理对象和数组的遍历

**语法**

jQuery.each(array, callback )

jQuery.each( object, callback )

第一个参数传递的就是一个对象或者数组，第二个是回调函数

$.each(["Aaron", "慕课网"], function(index, value) {

//index是索引,也就是数组的索引

//value就是数组中的值了

});

each就是for循环方法的一个包装，内部就是通过for遍历数组与对象，通过回调函数返回内部迭代的一些参数，第一个参数是当前迭代成员在对象或数组中的索引值(从0开始计数)，第二个参数是当前迭代成员(与this的引用相同

jQuery.each()函数还会根据每次调用函数callback的返回值来决定后续动作。如果返回值为false，则停止循环(相当于普通循环中的break)；如果返回其他任何值，均表示继续执行下一个循环。

$.each(["Aaron", "慕课网"], function(index, value) {

return false; //停止迭代

});

jQuery方法可以很方便的遍历一个数据，不需要考虑这个数据是对象还是数组

## jQuery中查找数组中的索引inArray

在PHP有in\_array()判断某个元素是否存在数组中，JavaScript却没有，但是jQuery封装了inArray()函数判断元素是否存在数组中。注意了：在ECMAScript5已经有数据的indexOf方法支持了，但是jQuery保持了版本向下兼容，所以封装了一个inArray方法

jQuery.inArray()函数用于在数组中搜索指定的值，并返回其索引值。如果数组中不存在该值，则返回 -1。

**语法：**

jQuery.inArray( value, array ,[ fromIndex ] )

用法非常简单，传递一个检测的目标值，然后传递原始的数组，可以通过fromIndex规定查找的起始值，默认数组是0开始

例如：在数组中查找值是5的索引

$.inArray(5,[1,2,3,4,5,6,7]) //返回对应的索引：4

注意：

**如果要判断数组中是否存在指定值，你需要通过该函数的返回值不等于(或大于)-1来进行判断**

**jQuery中去空格神器trim方法**

页面中，通过input可以获取用户的输入值，例如常见的登录信息的提交处理。用户的输入不一定是标准的，输入一段密码：'  1123456  "，注意了： 密码的前后会留空，这可能是用户的无心的行为，但是密码确实又没错，针对这样的行为，开发者应该要判断输入值的前后是否有空白符、换行符、制表符这样明显的无意义的输入值。

**jQuery.trim()函数用于去除字符串两端的空白字符**

**这个函数很简单，没有多余的参数用法**

**需要注意：**

* 移除字符串开始和结尾处的所有换行符，空格(包括连续的空格)和制表符（tab）
* 如果这些空白字符在字符串中间时，它们将被保留，不会被移除

通过右边的代码可以明显看到trim使用后的效果

## jQuery中DOM元素的获取get方法

jQuery是一个合集对象，如果需要单独操作合集中的的某一个元素，可以通过**.get()**方法获取到

以下有3个a元素结构：

<a>1</a>

<a>2</a>

<a>3</a>

通过jQuery获取所有的a元素合集$("a")，如果想进一步在合集中找到第二2个dom元素单独处理，可以通过get方法

**语法：**

.get( [index ] )

**注意2点**

1. get方法是获取的dom对象，也就是通过document.getElementById获取的对象
2. get方法是从0开始索引

所以第二个a元素的查找： $**(a).get(1)**

**负索引值参数**

get方法还可以从后往前索引，传递一个负索引值，**注意的负值的索引起始值是-1**

同样是找到第二元素，可以传递 $**(a).get(-2)**

**通过.get()方法获取到的元素会转换为DOM元素。**

**.css是JQ方法要用eq()获取JQ对象。**

## jQuery中DOM元素的获取index方法

get方法是通过已知的索引在合集中找到对应的元素。如果反过来，已知元素如何在合集中找到对应的索引呢？

.index()方法，从匹配的元素中搜索给定元素的索引值，从0开始计数。

**语法：参数接受一个jQuery或者dom对象作为查找的条件**

.index()

.index( selector )

.index( element )

* 如果不传递任何参数给 .index() 方法，则返回值就是jQuery对象中第一个元素相对于它同辈元素的位置
* 如果在一组元素上调用 .index() ，并且参数是一个DOM元素或jQuery对象， .index() 返回值就是传入的元素相对于原先集合的位置
* 如果参数是一个选择器， .index() 返回值就是原先元素相对于选择器匹配元素的位置。如果找不到匹配的元素，则 .index() 返回 -1

简单来说：

<ul>

    <a></a>

    <li id="test1">1</li>

    <li id="test2">2</li>

    <li id="test3">3</li>

</ul>

$("li").index() 没有传递参数，反正的结果是1，它的意思是返回同辈的排列循序，第一个li之前有a元素,同辈元素是a开始为0，所以li的开始索引是1

如果要快速找到第二个li在列表中的索引,可以通过如下2种方式处理

$("li").index(document.getElementById("test2")) //结果：1

$("li").index($("#test2")) //结果:1