正课:

1. 查找

2. 修改

3. 按节点间关系查找

4. 添加/删除/替换/克隆

5. 事件绑定

jQuery API三大特性:

1. 一个函数两用: 没给新值，默认读取原值

给了新值，则变为修改操作

2. 自带遍历效果: 对整个jQuery类数组对象调用简化版API，等效于自动对jQuery对象中每个DOM对象分别调用简化版API

3. 大多数API都返回正在操作的jQuery对象本身

1. 查找:

内容过滤: jQuery新增:

:contains(文本) 以元素的文本内容作为筛选条件

:has(选择器) 以元素的子元素特征作为筛选条件

:parent 非空的元素

:empty 空元素

可见性过滤:

:hidden 选择所有隐藏的元素

只能选择: display:none input type="hidden"

:visible

表单元素过滤:

:input 选择所有表单元素,包括:input select textarea button

:type: 每种type都有对应的选择器:

:text :password :checkbox :radio :submit ...

状态过滤: :disabled :checked :selected :enabled

2. 修改:

内容:

HTML代码片段: $(...).html() .innerHTML

纯文本内容: $(...).text() .textContent

表单元素的内容: $(...).val() .value

清空内容: $(...).empty()

属性:

HTML标准属性: $(...).attr("标准属性名"[,"值"])

$(...).attr({

标准属性:值,

标准属性:值,

})

状态属性: $(...).prop("状态属性"[,bool])

自定义扩展属性:

$(...).data("属性名",值) .dataset.属性名

样式:

1. 获取或修改内联样式: $(...).css("css属性名",值)

问题: 一次只能修改一个css属性的值

解决: 1. 链式操作:

2. 一个css同时修改多个css属性:

$(...).css({

css属性名:值,

css属性名:值,

})

强调: .css()在修改时，自动修改内联样式

在获取时，自动获取计算后的样式

2. 用class批量应用样式:

$(...).addClass("...")

$(...).removeClass("...")

$(...).hasClass("...")

$(...).toggleClass("...")

等效于:

if($btn.hasClass("down"))

$btn.removeClass("down")

else

$btn.addClass("down");

3. 按节点间关系查找:

1. 父子:

$(...).parent()

$(...).children(["selector"])

$(...).find("selector")

$(...).children().first() .firstElementChild

$(...).children().last() .lastElementChild

2. 兄弟:

$(...).next()

$(...).nextAll()

$(...).prev()

$(...).prevAll()

$(...).siblings(["selector"])

$(...).index() 查找任意元素在当前父元素下的下标位置

4. 添加/删除/替换/克隆

添加:

1. 用HTML代码片段创建新元素:

var $a=$("<a href='url'>text</a>");

2. 将新元素添加到DOM树

末尾追加

$parent.append($a) //return $parent

$a.appendTo($parent) //return $a

开头插入

$parent.prepend($a)

$a.prependTo($parent)

在当前元素前后插入

$child.after($a) 将a插入到child之后

$child.before($a) 将a插入到child之前

删除: $().remove()

替换:

$(旧).replaceWith($新) //return $旧

$新.replaceAll(旧) //return $新

克隆:

$().clone() 浅克隆: 仅克隆属性和样式，不克隆行为

$().clone(true) 深克隆: 即克隆属性和样式，又克隆行为

jQuery中$的原理: 举例说明，你阅读过的常用库/框架的代码部分

$是一个工厂函数, 内部自动new jQuery创建jQuery类型的子对象.

3种情况:

1. 参数为选择器时:

先查找DOM元素，再将找到的DOM元素保存在新创建的jQuery对象中

优化(Speed-up):

1. 如果选择器是id，则自动调用getElementById

2. 如果选择器是标签名,则自动调用getElementsByTagName

3. 如果选择器是class,则自动调用getElementsByClassName

4. 如果选择器复杂，自动调用querySelectorAll

2. 参数为DOM元素:

直接将DOM元素保存到新创建的jQuery对象中

3. 参数为HTML代码片段:

用html代码片段创建一个/多个新DOM元素，再将创建的DOM元素保存到新创建的jQuery对象中

5. 事件绑定:

jQuery中共有几种事件绑定方式，有什么不同:

1. bind/unbind() 代替addEventListener/removeEventListener

unbind() 三种重载:

unbind("事件名",处理函数): 移除当前元素指定事件上的一个指定的处理函数

unbind("事件名"): 移除当前元素指定事件上所有处理函数

unbind(): 移除当前元素上所有事件的处理函数

2. one() 只触发一次，触发后自动解绑

3. live/die() 已被废弃, 将所有元素的所有事件处理函数，都集中绑定在document根节点上

4. delegate/undelegate() 其实就是利用冒泡

$parent.delegate("selector","事件名",function(){

this->e.target->目标元素

执行操作!

})

鄙视: 简述jQuery中事件委托的原理

$parent.bind("事件名",e=>{

e.target->目标元素

if($(e.target).is(selector))

执行操作!

})

总结: bind vs delegate(利用冒泡)

1. 绑定位置: bind直接绑在目标元素(子元素上)

delegate(利用冒泡) 绑在父元素上

2. 监听个数: bind 监听个数多

delegate(利用冒泡) 监听个数少

3. 动态生成子元素:

delegate(利用冒泡) 动态生成的新子元素可自动获得父元素上的事件处理函数

bind 动态生成的新子元素必须反复绑定

5. on/off()

on("事件名",处理函数) 代替bind()

on("事件名","选择器",处理函数) 代替delegate()

6. 终极简化:

对常用的事件，进行了终极简化:

$(...).事件名(处理函数)