正课:

1. 修改:

2. 按节点间关系查找:

3. 添加/删除/替换/克隆

4. 事件绑定

学习jQuery还是在学DOM

1. 修改:

内容:

.innerHTML => .html()

.textContent=> .text()

.value => .val()

.empty()

属性:

标准属性: .attr("属性名"[,"值"])

状态属性: .prop("状态"[,bool])

自定义扩展属性: .dataset.属性名 => .data("属性名"[,值])

样式:

1. 细致修改每个css样式属性:

.style/getComputedStyle(elem) => .css("css属性名"[,值])

.css({

css属性:值,

... : ... ,

})

2. 用class来简化批量修改样式:

.className="xxx" => .addClass("xxx")

全部替换 仅追加

jQuery: 4种:

.addClass("xxx xxx");

.removeClass("xxx xxx");

.hasClass("xxx xxx")

.toggleClass("xxx")

=> if(!$tar.hasClass("down"))

$tar.addClass("down");

else

$tar.removeClass("down");

2. 按节点间关系查找:

1. 父子关系:

.parentNode => .parent();

.children => .children(["selector"]) //直接子元素(可筛选)

.firstElementChild => .children().first()

.lastElementChild => .children().last()

\*\*\*.find("selector") //查找所有后代元素中符合选择要求的元素

2. 兄弟关系:

.nextElementSibling => .next(); //下一个

.nextAll() //之后所有

.previousElementSibling => .prev(); 前一个

.prevAll() //之前所有

\*\*\*.siblings(["selector"]) 除自己之外所有

.is("selector") 判断任意元素的特征

$("所有元素的集合").index("要查找的元素")

如果在同一个父元素下: $("要查找的元素").index();

3. 添加/删除/替换/克隆

1. 添加:2步:

1. var elem=$("html代码片段")

var a=$("<a href='http://tmooc.cn'>go to tmooc</a>");

2. 将新元素添加到DOM树:

father.appendChild => father.append(elem); //末尾追加

\*\*\*father.prepend(elem); //开头插入

返回值:father

$elem.appendTo/prependTo(father)

返回值:$elem，可继续对新添加的元素，做链式操作

father.inertBefore(elem,child) => child.before(elem)

\*\*\*child.after(elem)

简写: $father.append("html代码片段")

$("html代码片段").appendTo($father)

2. 删除: $(...).remove();

3. 替换: 2种:

$(old).replaceWith($new) => $old

用右边的$new替换左边的$old

$(new).replaceAll(old) => $new

用左边的$new替换右边的$old

4. 克隆: 复制一个元素对象:

$(...).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

总结: 增删改查

4. 事件绑定:

\*\*\*鄙视: jQuery中共有几种事件绑定方式，有什么差别

DOM: .addEventListener("事件名",handler);

.removeEventListener(...)

1. .bind("事件名",handler)

.unbind(...): 3种重载:

.unbind("事件名",handler) 移除指定事件上的某一个处理函数

.unbind("事件名") 移除指定事件上所有处理函数

.unbind() 移除所有事件上的处理函数

2. .one("事件名",handler)

用.one绑定的处理函数，触发一次后，自动解绑

3. .live/die

将一切处理函数都绑定在根节点document上

4. .delegate 简化版的利用冒泡

.undelegate

$father.bind("事件名",e=>{

//this->father

var $tar=$(e.target); //获取目标元素

if($tar.is("selector")){ //识别目标元素是否想要的

... ...

}

})

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

//this=>e.target

var $tar=$(this);

})

鄙视: bind vs delegate

1. 绑定位置: bind绑在每个子元素上

delegate 绑在父元素上

2. 监听个数: bind的监听个数多

delegate 监听个数少

3. 新元素: 每添加一个新元素都要重新bind

每添加一个新元素，可自动继承父元素的delegate事件处理函数

5. .on/off: 2种用法

1. 代替bind: .on("事件名",handler)

2. 代替delegate: .on("事件名","选择器",handler)

6. .事件名(handler)

强调: 仅对常用的事件名进行了简化

页面加载后触发: 2种:

1. DOMContentLoaded: 仅在DOM内容和JS加载完就可提前执行——早

何时: 所有不需要css就可执行的js代码，都应放在DOMContentLoaded中

如何: $(document).ready(()=>{...})

$().ready(()=>{...});

$(()=>{...})

其实: 放在body结尾的js，本身就是DOM加载后自动执行

问题: js文件不一定总是在结尾

2. window.onload: 必须等全部网页内容(DOM,JS,CSS,图片)加载完成才执行——晚

何时: 如果依赖于css才能正常执行的js代码，必须放在window.onload中

如何: $(window).load(()=>{...})

鄙视: $(document).ready() vs window.onload 差别

1. 触发时间: $(document).ready() DOM内容和js加载完提前触发

window.onload 所有网站内容(DOM,JS,CSS,图片)加载完成才能触发

2. 处理函数的个数:

$(document).ready() 以添加监听的方式绑定事件,可同时添加多个处理函数在DOM内容加载后执行

window.onload 以赋值方式绑定事件,只能绑定一个

鼠标事件:

mouseover mouseout : 进出子元素会频繁冒泡触发父元素的事件

mouseenter mouseleave : 即使频繁进出子元素，也不会冒泡触发父元素的事件

更简化: 如果同时绑定进入和移出事件，可使用hover

.hover(mouseenter,mouseleave)

更简化: 如果mouseenter和mouseleave可统一为一个函数

.hover(handler) 即给mouseenter，又给mouseleave

模拟触发: 用程序调用元素上的事件处理函数

$(elem).trigger("事件名")

更简化: $(elem).事件名()