day28 jquery

概述

jquery是一个轻量级的JavaScript库，它对应原生的js进行了封装，将对应的dom操作及对应的动画等做了一系列的方法封装，里面所有的内容都是方法。它的核心是链式调用及对应的jquery对象的扩展。

优点

- 丰富的选择器及筛选器 （帮助获取dom）
- 良好的动画兼容（采用也是css3的动画）
- 优秀的链式调用写法 （里面的每个方法返回的都是对应的jQuery对象）

官方网站

官网

中文网

jquery入门

引入jquery.js

```
<script src='./jquery.min.js'></script>
```

基础使用

```
//jquery对象
//使用$符号 获取jquery对象
//使用jQuery 获取jquery对象
console.log(jQuery())
console.log($())
//获取dom元素 原生获取
let box = document.querySelector('.box') //element 对象
console.log(box)
//通过jquery获取
console.log($('.box')) //jQuery对象
console.log(jQuery('.box'))
```

jQuery对象和DOM对象之间的转换

```
//jquery是不能调用dom里面的函数 dom也不可以调用jquery对象的内容
//jquery对象转为对应的dom对象 通过下标来获取里面的dom元素
let toElement = $('.box')[0]
console.log(toElement)
//dom对象转为jquery对象
//将dom对象传入jquery方法就会产生jquery对象
let toJQuery =  $(box)
console.log(toJQuery)
```

选择器

支持原生css中的所有选择器

```
// 使用jquery方法或者是对应的$方法
console.log($('#box')) //获取id为box的jquery对象
console.log($('.content')) //获取class为content的jquery对象
console.log($('.content[name=hello]')) //获取class为content里面name属性为hello的
jquery对象
//获取所有的li
console.log($('li'))
//获取第一个li
console.log($('li:first-child()'))
//获取最后一个li
console.log($('li:last-child()'))
```

内置对应的选择器

- first 获取第一个
- last 获取最后一个
- eq 获取对应下标位的
- odd 获取奇数下标位
- even 获取偶数下标位
- 支持同时传入多个选择器

```
//内置first选择器
console.log($('li:first'))
//内置last选择器
console.log($('li:last'))
//拿到第3个li 内置eq选择器 传入对应的下标
console.log($('li:eq(2)'))
//支持传入多个选择器
console.log($('.content:eq(1),li:eq(1)'))
//获取奇数下标位 1 3
console.log($('li:odd'))
//获取偶数下标位 0 2
console.log($('li:even'))
```

筛选器

针对于对应的jQuery对象来进行筛选对应的内容

筛选的相关方法

位置筛选

- first 第一个
- last 最后一个
- eq 通过下标来获取某个

关系筛选 （里面都可以传入选择器）

- 兄弟关系

  prev 上一个

  prevAll 上面的所有

  next 下一个

  nextAll 下面的所有

siblings 所有的同辈元素

- 父子关系

  parent 父元素

  children 子元素

- find 查找 根据选择器查找

```
//获取全部的li
console.log($('li'))
//获取第一个li
console.log($('li').first())
//获取最后一个li
console.log($('li').last())
//获取对应的下标的对象
console.log($('li').eq(1))
//通过关系获取 （里面都可以传入对应的选择器）
//父子关系
// 通过第一个li获取里面b标签 children如果不传参是获取所有包含的内容 如果传参那么就会进行选择器匹配
console.log($('li').eq(0).children('b'))
//获取里面包含的内容
console.log($('li').eq(0).children())
//通过li获取对应的父元素 (传入对应的父元素的选择器)
console.log($('li').parent('div')) //里面没有内容
console.log($('li').parent()) //获取所有的父元素
//兄弟关系
// 上一个
console.log($('li:eq(2)').prev())
console.log($('li:eq(2)').prev('.header')) //里面没有内容
// 下一个
console.log($('li:eq(2)').next())
// 上面的所有
console.log($('li:eq(2)').prevAll())
// 下面的所有
console.log($('li:eq(2)').nextAll())
//拿到所有的同辈元素
console.log($('li:eq(2)').siblings())
```

属性操作

- prop 获取和设置对应本身就有的属性 （a标签本身就存在对应的href属性 img本身就存在src属性 如果不是本身有的将不会显示）
- attr 获取和设置所有的属性 （底层实现setAttribute 和 getAttribute）

```
//进行属性操作 prop attr 传俩个参数就是设置 传一个参就是获取 （参数如果满了都是设置 参数没有满就是获取）
//获取元素
//获取input的value属性
console.log($('input').prop('value'));
//针对本身没有的是无法获取的
console.log($('input').prop('hello')); //undefined
//prop进行对应的设置
//针对本身就存在的可以进行设置
$('input').prop('class','input')
//针对于本身不存在的无法进行设置 无法设置到对应的元素上 但是它会设置到本身的对象上
```

```
$('input').prop('hello','hi')//element['hello'] = 'hi'
console.log($('input').prop('hello')); //hi
//attr 内部采用了setAttribute getAttribute
console.log( $('input').attr('hello')) //获取 helloWorld
$('input').attr('hello','您好')
```

删除的操作

- removeProp （删除对应prop的属性）
- removeAttr （removeAttribute）

```
//removeProp
// 删除prop添加的属性 本身存在的属性或者是prop添加的属性
// delete element[key]
// $('input').removeProp('value') 无法删除
$('input').removeProp('hello')
console.log($('input').prop('hello'))
//removeAttr 内部采用的removeAttribute
$('input').removeAttr('hello')
```

class属性的操作

- addClass 追加一个class
- removeClass 删除一个class
- toggleClass 切换class

```
$('div').addClass('content')
//删除
$('div').removeClass('content')
//切换class 有变成没有 没有变成有
$('div').toggleClass('content') //没有content会添加content
$('div').toggleClass('content') //有content会删除content
```

原生js模拟实现addClass

```
//原生js实现对应的class添加
function addClass(element,className){
    //判断element的className是否为空
    if(!element.className){
        //直接设置
        element.className = className
    }else{
        //读取对应的className中的内容 空格分割
        classNames = element.className.split(' ')
        classNames.push(className)
        //设置给对应的原生
        element.className  = classNames.join(' ')
    }
}
addClass($('div')[0],'hello')
```

样式的操作

使用**css**方法 （传一个参数就是获取 传俩个参数就设置）

```
//使用css来获取字体颜色 里面使用了getComputedStyle方法 element.currentStyle
console.log($('.box').css('color'))
//设置样式 传入俩个参数 内部采用的是style赋值
$('.box').css('backgroundColor','red')
```

显示内容的操作

- html （底层实现使用了innerHTML）
- text （底层实现使用了innerText）

```
//html 还是 text方法 传一个参数设置 不传参就是获取
console.log($('.box').html())
console.log($('.box').text())
//设置相关
$('.box').html('<span>222</span>')
$('.box').text('<span>222</span>')
```

input输入框的值获取及设置 value属性设置获取

val方法

```
//获取input框的value值
console.log($('input').val());
//设置value值
$('input').val('天气很好 出去走走')
```

节点操作

增删改

- append 追加到末尾
- before 将传入的元素添加到调用的元素之前
- insertBefore 将调用的元素添加到传入的元素之前
- after 将传入的元素添加到调用的元素之后
- inertAfter 将调用的元素添加到传入的元素之后
- replaceAll 替换所有
- replaceWith 替换某一个（传入选择器）
- remove 移除本身及内容
- empty 移除里面的内容
- clone 克隆一个新的对象 （是否克隆对应的数据及事件 是否深度克隆数据及事件）

```
//获取div 给div添加一个a标签
//创建元素 创建jquery对象
let element = $('<a href="http://www.baidu.com"><span>111</span>去百度</a>')
console.log(element)
//添加给对应的div 底层实现为appendChild
$('div').append(element)
//添加到元素之前 parent.insertBefore(node,child) //父元素为body
$('div').before(element) //将element放到div之前
//将div放到element之前
$('div').insertBefore(element)
// 将element添加到div之后 after
$('div').after(element)
//将div添加到element之后
```

```
$('div').insertAfter(element)
//克隆 clone
let copy = element.clone(true,true)
//替换 用element替换div
copy.replaceAll($('div'))
//replaceWith 用传入的内容来替换调用的内容
copy.replaceWith('<b>替换的内容</b>')
//删除 可以传入选择器 （删除其中某一个内容）
// element.remove()
//清空 保留原本的标签 里面的内容被删除
element.empty()
```

尺寸

获取元素的宽度高度

- width 和 height 获取原始宽度
- innerWidth 和 innerHeight 包含填充 不包含边框 clientWidth
- outerWidth 和 outerHeight 包含边框和填充 传入对应的参数是否包含margin offsetWidth

```
//width height 不包含边框和填充
console.log($('div').width(),$('div').height());
//innerWidth innerHeight 包含填充 不包含边框
console.log($('div').innerWidth(),$('div').innerHeight());
//outerWidth outerHeight 包含边框和填充 传入对应的参数是否包含margin
console.log($('div').outerWidth(),$('div').outerWidth());
console.log($('div').outerWidth(true),$('div').outerWidth(true));
```

位置

- position 获取定位的位置
- offset 获取离页面的位置

```
console.log($('.inner').position()) //获取定位的位置 返回的是一个对象 left top
console.log($('.inner').offset()) //相对于页面的位置
```

jQuery的动画

- animate 方法
- fadeIn 淡入
- fadeOut 淡出
- slideDown 显示
- slideUp 隐藏
- slideToggle 切换
- hide 隐藏
- show 显示
- toggle 切换

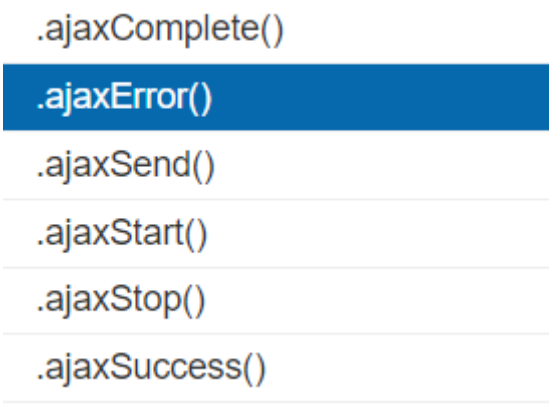
```
//通过回调函数来解决异步问题 淡入淡出变化的是透明的
$('div').fadeIn(2000,function(){
    //dom对象 指向调用的元素
    console.log(this)
    $(this).fadeOut(3000)
})
```

```
// $('div').fadeTo(2000,1) 到达某个透明的
//传入目标对象（只支持传入number类型的值 如果传入的是颜色需要使用Color这个类） 时间  回调
$('div').animate({
    left:50,
    top:50,
    backgroundColor:'yellow' //无效
},3000,()=>{
    console.log('完成');
})
//切换
$('div').toggle(2000)
$('div').slideDown(3000,()=>{ //滑下来显示
    $('div').slideUp(3000,()=>{ //移上去隐藏
        $('div').slideToggle(3000,()=>{ //切换

        })
    })
})
$('div').show(2000, () => {
    $('div').hide(2000, () => {

    })
})
```

jQuery的ajax

- ajax方法 发送任何请求
- get方法 发送get请求
- post方法 发送post请求

```
for(let i=0;i<$('button').length;i++){
    $('button')[i].onclick = function(){
        switch(i){
            case 0: $.get('https://jsonplaceholder.typicode.com/todos',
(data,status,xhr)=>{
                console.log(data,status,xhr)
            })
                break;
            case 1: $.post('https://jsonplaceholder.typicode.com/posts',
(data,status,xhr)=>{
                console.log(data,status,xhr)
            })
                break;
            case 2: $.ajax('https://jsonplaceholder.typicode.com/todos',{
                data:{},
                method:'get',
                success(data){
                    console.log(data)
                }
            })
                break;
        }
    }
}
```



- ajaxComplete 请求完成
- ajaxError 请求出错
- ajaxSend 请求发送
- ajaxStart 请求开始发送
- ajaxStop 请求停止发送
- ajaxSuccess 请求成功

jQuery的事件

jQuery中实现了观察者模式

- on 监听
- off 取消
- one 执行一次

对应原有的事件进行增强

将每个事件的事件名定义为方法

- click 方法
- mousedown 方法
- mousemove 方法
- mouseenter 方法
- mouseleave 方法
- hover 传入俩个处理函数一个处理鼠标进入 一个处理鼠标移出
- ....

jQuery的多库共存

避免关键词（全局污染）冲突，修改原本的$和对应的jQuery，变成自定义的内容。

noConfict

```
//关键词
console.log($)
console.log(jQuery)
// 全局污染
// var $ = 'hello'
// var jQuery = 'hhaha'
// console.log($)
// console.log(jQuery)
// var a = $.noConflict() //$符号没有意义 jQuery可以使用
// console.log($)
// console.log(a())
// console.log(jQuery)
```

```
var b = $.noConflict(true) //$符号没有意义 jQuery也没有意义
console.log($)
console.log(jQuery)
console.log(b)
```

jQuery的插件扩展

使用extend关键词

自定义函数到jQuery中

- **自定义静态函数**

  $.extend

- 自定义原型函数

  $.fn.extend

```
//静态方法扩展
$.extend({
    sayHello() {
        console.log('hello')
    }
})
$.sayHello()
//原型方法扩展
$.fn.extend({
    rmClass(className) {
        console.log(this) //当前访问的jQuery对象
        //返回的数组
        let result = []
        //完成removeClass的操作
        for (var i = 0; i < this.length; i++) {
            //获取本身的class
            let element = this[i]
            let classStr = element.className
            //再使用对应的空格进行分割
            let classNames = classStr.split(' ')
            //查询对应的className是否存在
            let index = classNames.findIndex(name => {
                return name == className
            })
            //判断是否存在
            if (index != -1) {
                //存在从数组中删除
                classNames.splice(index, 1)
            }
            //判断classNames是否个数为0
            if (!classNames.length) {
                //移出class
                element.removeAttribute('class')
            } else {
                //设置对应的className
                element.className = classNames.join(" ")
            }
            result.push(element)
        }
```

```
        //返回jquery对象
        return jQuery([]).pushStack(result)
    }
})
console.log( $('div').rmClass('hello'))
```