js四

jQuery 常用方法

> 1.$(‘.class’) 最常用的DOM获取
> 2.$oDiv.attr([key], [val]) 获取/设置属性
> key是属性名，如果只传key不传val就是获取
> key & val 都传就是设置
> 如果$oDiv对象里面有多个DOM 获取的时候只能获取到第一个DOM的属性值 设置的时候所有的DOM都能被设置

```
var $oDiv = $('.hello');
$oDiv.attr('data-id');
$oDiv.attr('data-id', 'hello');
$oDiv.data('class');
```

1. $oDiv.data([key], [val])
   如果只传入key 就是获取$oDiv里面的 data-[key] 的属性值
   如果key val 都传 就是设置 \$oDiv里面的[key] = [val] 而不是data-[key]属性

```
$oDiv.data('idd', 'xxx'); // 改不了data-idd
$oDiv.data('data-idd', 'xxx'); // 改不了data-idd
$oDiv.attr('data-idd', 'xxx') // 可以改data-idd
```

1. class
   $oDiv.addClass([val]) 添加一个class为 val
   $oDiv.removeClass([val]) 删除class为 val的
   $oDiv.hasClass([val]) 判断DOM是否有class val 如果$oDiv里面有多个DOM 只判断第一个
   `$oDiv.toggleClass([val])` 判断$oDiv有没有class [val] 有就删除没有就添加

```
$oDiv.addClass('world');
$oDiv.removeClass('hello');
$oDiv.addClass('abc');
$oDiv.hasClass('world');
```

```
$oDiv.hasClass('hello');
$oDiv.toggleClass('abc')
```

1. $oDiv.html([val])
   [val] 有值就是设置$oDiv 的innerHTML
   没有值就是获取 $oDiv 的 html

```
$oDiv.html('<a href="https://www.baidu.com/">back baidu</a>');
$oDiv.html();
```

1. $input.val([val])
   [val] 有值就是设置$input 的value
   没有值就是获取 $input 的 value

```
$input.val('shanghai'); // 设置
$input.val(); // 获取
$input.val(function (index, value) {
/ 有多少值就执行多少次， index 是当前input在集合中的索引
// value是当前input的value值
return value + index;
})
```

1. $Dom.offset()
   返回一个对象 {top: val, left: val} 分别是距离body的偏移量
2. $Dom.position()
   返回一个对象 {top: val, left: val} 分别是距离父级参照物的偏移量
3. $oDiv.css(name|pre[,val|fn])
   传参两种方式 一种是两个字符串 第一个是key 第二个是value
   一种是传对象 对象里面的key/value 就是css属性的key/value
   [value] 可以是一个函数 有两个参数(index, key) jQ集合有多少项就执行多少次
   参数分别是当前DOM在集合的索引 和 当前DOM 需要更改的CSS属性的值(未修改之前的值)

```
$oDiv.css('color', 'red');
$oDiv.css('height', '300');
$oDiv.css('width', '300');
$oDiv.css('margin-top', '300px');
$oDiv.css('background', 'black');
$oDiv.css({
```

```
    color: 'red',
    height: 300,
    width: 300,
    background: 'black',
    'margin-top': '300px'
 });

$oDiv.css({
    marginTop: '20px',
    color: 'red',
    height: function (index, text) {
         console.log(index, text, 'height');
    return 300
     },
    width: function (index, text) {
         console.log(index, text, 'width');
    return 300
     },
     background: 'black'
})
$oDiv.css('color', function (index, text) {
     console.log(index);
     return 'red';
})
```

1. $oDiv.width([val]) $oDiv.height([val])
   不传val就是获取宽/高
   传val就是设置宽/高
   和$oDiv.css 获取的宽高区别就是 css获取的宽高时带单位的字符换
   width/height 获取的是不带单位的，是number类型

```
$oDiv.css('height');
$oDiv.height();
$oDiv.height('200px')
$oDiv.width();
$oDiv.width('200px')
```

1. on/off 事件

```
$oDiv.click(function () {
    alert('world')
});
$oDiv.on('click', function () {
```

```
    alert('hello')
});
```

> jQuery两种绑定事件的方式
>
> 1. $oDiv.[event](fn)
> 2. $oDiv.on([event], fn)
>   event不用加on, 例如元素JS用onclick 咱们只用click就行
>   都可以绑定多次 一般用第二种
>
> $oDiv.off([event], [name])
>
> 1. 两个参数都有 解除event事件的name方法
> 2. 如果name参数为空 解除event事件绑定的所有方法
> 3. 如果两个参数都没有 解除当前DOM绑定的所有事件

```
$oDiv.off(); // 解除$oDiv绑定的所有事件
$oDiv.off('click'); // 解除$oDiv绑定的所有click事件
$oDiv.off('click', fn2); // 解除$oDiv绑定的click事件中的fn2方法
```

> $(this).index()

```
$focusLis.on('mouseenter', function (){
$(this).index();
// 获取当前DOM在 所有兄弟元素 里面的索引
})
```

less

[less文档](http://lesscss.cn/)

less用法

- 开发环境(dev) 生产环境(pro 就是线上)
- dev 中可以引入less.min.js 去解析 引入less的link 的rel=”stylesheet/less”
- dev 用解析工具直接生成(webpack)
- pro 会讲less先解析成css再上线
- 在index.less所在的文件夹下运行 lessc index.less index.css

- 将index.less 编译成 index.css（这个css名是自己起的）
- lessc index.less index.min.css -x
- 编译并且压缩index.less

常用用法

嵌套

```
.outer {
    .inner {
    }
}
```

变量

```
@themeColor:red;
.outer {
  background: @themeColor;
}
```

- 通过@开头定义一个变量 可以用 - (中杆)
- 使用的时候直接给对应的css属性赋值就行
  变量的常见用途
- 主体更改 改变变量值即可改变主题
- 路径提升

函数 混用

```
.box {
  width: 100px;
  height: 200px;
  background-color: aquamarine;
}
.box2 {
  .box;
  background-color: bisque;
}
```

- 在一个样式（.box2）可以直接写上（.box;）
  也就是把 .box 的样式完全复制一份过来 如果需要不同，再写上不同的样式去覆盖
- 不带小括号，编译后 .box 依旧存在

```
.box(@num) {
  width: 100px;
  height: @num;
  background-color: aquamarine;
}
.box2 {
  .box(300px);
  background-color: bisque;
}
```

- 带小括号 可以传参 .box就完全变成了一个函数 编译后不会存在

函数返回值

```
.transform1(@totate) {
  transform: @totate;
  -webkit-transform: @totate;
  -moz-transform: @totate;
  -ms-transform: @totate;
  -o-transform: @totate;
}
.box(@n, @m) {
  @result: @n + @m;
}
.fontS(@n, @m) {
  @resultFontS: @n + @m;
}
.box1 {
  .box(0.1, 0.2);
  //执行多次如果有返回值 以第一次执行结果为标准
  .fontS(14, 14);
  .fontS(24, 24);
  .fontS(34, 34);
  background: cadetblue;
  font-size: unit(@resultFontS, px);
  opacity: @result;
}
.box2 {
  .box(0.2, 0.2);
  .fontS(24, 14);
```

```
    //执行多次没有返回值的函数 每次都编译 所以以最后一次为准
    .transform1(rotate(100deg));
    .transform1(rotate(200deg));
    .transform1(rotate(300deg));
    .transform1(rotate(400deg));
    background: chartreuse;
    font-size: unit(@resultFontS, px);
    opacity: @result;
  }
```

- 执行多次如果有返回值 以第一次执行结果为标准
- 执行多次没有返回值的函数 每次都编译 所以以最后一次为准

函数参数的默认值

```
.box(@n:0.2, @m:0.1) {
  @result: @n + @m;
}
.fontS(@n:15px, @m:15px) {
  @resultFontS: @n + @m;
}
```

- 形参:默认值 (@n:0.2)

less 作用域 与 变量提升 即声明也定义

> 函数也会提升 和JS一样

```
@fontSize: 15px;
.box3 {
// 结果是30px
  font-size: @fontSize;
  color: white;
  background: blue;
  @fontSize: 30px;
  .box4 {
    .hello(20, 20);
    // 函数也是提升
    font-size: unit(@result, px);
    color: white;
    background: blue;
```

```
    }
}
.hello(@n, @m) {
    @result: @n + @m;
}
```

继承 extend

```
.box {
    width: 100px;
    height: 100px;
}
.box1:extend(.box) {
    background: chartreuse;
}
.box2 {
    .box;
    background: cadetblue;
}
```

- .box1:extend(.box) {} box1以这种方式继承box的样式，编译后是box,box1这样的形式，不是复制一份box的代码给box1
- .box1{.box;} 这种方式就是复制 把box里面的样式复制一份给box1
- .box1:extend(.box, .com) {} 这种写法box1可以同时继承box 和 com 的样式

条件判断

```
.box(@num) when (@num >= 10) {
    font-size: 40px;
}
.box(@num) when (@num < 10) {
    font-size: 20px;
}
.box(@n, @m) when (@n > 10) and (@m > 10) {
    font-size: 80px;
}
.box1 {
    .box(100);
    background: cadetblue;
}
```

less 连接符 &

```
.box4 {
  background: red;
  &.boxa {
    font-size: 40px;
  }
  &:hover {
    font-size: 20px;
  }
}
```

> 块级作用域里面的 & 连接符 代表的就是父级标签

es6

- 解构赋值 数组和对象
  let ary = [10, 20, 15];
  由于数组是有顺序的，x表示数组的第一项，y表示数组第二项….
  x,y,z即声明又定义
  let [x,y,z] = ary;
  let [x,,z]=ary;//中间这一项表示空项
  给b赋初始值30 = 表示设置默认值
- 对象的解构赋值 = 表示设置默认值 :表示更改变量名
  let obj = {name: “zf”, age: 8};
  由于对象没有顺序，解构时变量名跟属性名得一样
  let {name: n, age = 9, hobby = “sleep”} = obj;
- 嵌套的解构赋值
  let ary2 = {a1: “同仁堂”, b1: [“珠峰”, “孟记粥铺”], c1: “回龙观东大街”};
  let {a1, b1: [e, f], c1} = ary2;
  console.log(a1, e, f, c1);
- … 扩展运算符，展开运算符，剩余运算符(rest element)
  - 合并
    let ary5 = [10, 20, 56, 7];
    let ary6 = [30, 40];
    let ary7 = […ary5, …ary6];
    let [, …arr1] = ary5;
    let […arr] = ary;//克隆数组
    let o1 = {n: 10, m: 20};
    let o2 = {a: 30, b: 40};

```
let o = {…o1, …o2};//es7
Object.assign(o1, o2);//合并对象
let {…o3} = o1;//克隆对象
```

箭头函数

- 箭头函数 参数部分=>函数体部分
  function fn(a){return a+10;}
  let fn = a => a+10; //若只有一行，并且有返回值 ,直接写return后面的内容，return可以省略
  若有多个参数拿小括号包起来 ，若函数体内容有多行，得用{}包起来
  let fn = (a,b)=>{
  let total = null;
  total = a+b;
  return a+b;
  }
- 箭头函数没有this,会往上级作用查找，若没找到继续往上查找，直到找到window
- 箭头函数中没有arguments
  不要定义成构造函数
- 不支持call,apply,bind改变箭头函数的this关键字，因为箭头函数没有this

定时器里的this

- 定时器里的this指的是sum方法中this
  function sum(){
  var that = this;
  window.setTimeout(function(){ //定时器里的this指的是sum方法中this
  },1000);
  }