学习报告

这两周主要是对JS进行了复习以及对java基础进行了学习

JS：

1.BOM:

//BOM是浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心是 window

//BOM是由一系列相关的对象构成，并且，每个对象都提供了很多方法和属性

//BOM缺乏标准，js语法的标准化组织是ECMA,DOM是W3C,BOM最初是NETscape浏览器标准的一部分

//DOM和BOM的区别

//DOM:文档对象模型，把一个 文档 当做一个 对象 来看待，顶级对象是document， 主要学习的是操作页面元素， 是W3C标准规范

//BOM:浏览器对象模型， 把 浏览器 当做一个 对象 来看待，顶级对象是window， 学习的是浏览器窗口交互的一些对象，是在个浏览器上定义的，兼容性较差

//BOM比DOM更大，它包含DOM

//window对象是浏览器的顶级对象，他具有双重角色

//1.他是js访问浏览器窗口的一个接口

//2.他是一个全局对象，定义在全局作用域中的变量，函数都会变成window对象的属性和方法

//在调用的时候可以省略window，前面学习的对话框都属于window对象方法，如alert(),prompt(),等

window对象常见事件:

①：窗口加载事件：

 //窗口加载事件：window.onload 当文档内容完全加载完成会触发该事件（包括图像，脚本文件，CSS文件等），就调用的处理函数

//运用该事件 那么 script 文件 就可以写在文件内的任意位置

        window.onload = function(){

            var btn = document.querySelector('button');

            btn.addEventListener('click', function(){

                alert('点击');

            })

        }

//使用该方法我们就可以把js代码写到页面元素的上方，因为onload是等页面内容全部加载完毕再去执行

//该方式只能写一次，如果有多个，会以最后一个为准

//如果使用addEventListener就不会有限制

        window.addEventListener('load',function(){

            var btn = document.querySelector('button');

            btn.addEventListener('click',function(){

                alert('点击');

            })

        })

        window.addEventListener('load',function(){

            var btn = document.querySelector('button');

            btn.addEventListener('click',function(){

                alert('点击w');

            })

        })

//DOMContenLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等  ie9以上才支持

//如果页面的图片很多的话，从用户访问到onload触发可能需要很长事件，交互效果就不能实现，必然影响用户的体验，此时用DOMContentLoaded事件比较合适

        document.addEventListener('DOMContentLoaded',function(){

            alert(33);

        })

②：调整窗口大小事件：

/window.onresize 是调整窗口大小加载事件，当触发时就调用处理的函数

//只要窗口大小发生像素变化，就会触发这个事件

//我们经常利用这个事件完成响应式布局，

window.innerWidth 当前屏幕的宽度 innerHeight 高度

window.addEventListener('load', function () {

            var div = document.querySelector('div');

            window.addEventListener('resize', function () {

                console.log(innerWidth);

                console.log('变化了');

                if (window.innerWidth <= 800) {

//根据可视窗口大小来选择是否应隐藏对应的元素

                    div.style.display = 'none';

                }

                else {

                    div.style.display = 'block';

                }

            })

        })

③定时器：

1. setTimeout()

/setTimeout() 方法用于设置一个定时器，该定时器在定时器到期后执行调用函数

//window.setTimeout(调用函数，延时时间); window 可省略 延时时间单位是毫秒，可不写，默认为0

//这个调用函数可以直接写函数，也可以写 函数名  或 写函数字符串(不推荐)

//页面中可能有很多定时器，我们经常给定时器加标识符 （名字）

var timer1 =

            setTimeout(function () {   //提倡

                alert('888');

            }, 1000);

        function callback() {

            console.log('爆炸了');

        }

        var timer2 = setTimeout(callback(), 1000);

        var timer3 = setTimeout('callback()', 2000);

        //setTimeout()这个函数我们也称为回调函数 callback

//普通函数时按照代码顺序直接调用 而这个函数需要等待时间，时间到了才去调用这个函数，因此称为回调函数

//之前的element.onclick = function(){} 或 element.addEventListenner('click', fn) 里面的函数也是回调函数

1. setInterval()

//window.setInterval(回调函数.[间隔的毫秒数]);

   //setInterval()方法重复调用一个函数，每隔这个时间，就去调用一次这个函数

//其余与 settimeout类似

        window.addEventListener('load',function(){

            var timer = setInterval(function(){

                console.log('持续输出');

            },1000);

        });

//倒计时效果

        //获取元素

        var hour = document.querySelector('.hour'); //小时的盒子

        var minute = document.querySelector('.minute');  //分钟的盒子

        var second = document.querySelector('.second');  //秒的盒子

        var inputTime = +new Date('2022-4-11 21:00:00');

//返回的是用户输入时间总的毫秒数 格式要输入正确

        countDown();  //先调用一次这个函数，防止第一次刷新页面有空白

        //开启定时器

        setInterval(countDown, 1000);  //一秒执行一次

        function countDown(time) {

            var nowTime = +new Date();  //返回的是当前时间的总毫秒数

            var times = (inputTime - nowTime) / 1000;

//times 是剩余时间的总毫秒数

            var h = parseInt(times / 60 / 60 % 24) //时

            h = h < 10 ? '0' + h : h;  //补0

            hour.innerHTML = h;  //把剩余小时写到 小时盒子内

            var m = parseInt(times / 60 % 60) //分

            m = m < 10 ? '0' + m : m;  //补0

            minute.innerHTML = m;  //把剩余分钟写到 分钟盒子内

            var s = parseInt(times % 60) //分

            s = s < 10 ? '0' + s : s;  //补0

            second.innerHTML = s;

            //把剩余 秒 写到 秒盒子内

        }

1. 停止定时器

//停止定时器 clearTimerout(标识符);

        window.addEventListener('load', function () {

            var btn = document.querySelector('button');

            var timer = setTimeout(function () {

                alert('爆炸了');

            }, 5000);

            btn.addEventListener('click', function () {

                clearTimeout(timer);

                alert('阻止了爆炸');

            })

        })

        //停止定时器 clearInterval(标识符);

        window.addEventListener('load', function () {

            var begin = document.querySelector('.begin');

            var end = document.querySelector('.end');

            var timer = null;

//全局变量 null是一个空对象  定义全局变量而不定义局部变量确保开启和清除函数都可以使用

begin.addEventListener('click', function () {

                timer = setInterval(function () {

                    console.log('ni hao ma');                }, 1000);

            })

            end.addEventListener('click', function () {

                clearInterval(timer);  //清除timer的定时器

            })

        })

④JS执行队列：

//js语言的一大特点是单线程。也就是同一时间只能做一件事

        //因为JS这门脚本语言诞生的使命就是处理页面中用户的交互以及操作DOM

        //比如对某个DOM元素进行添加和删除操作，应该先进行添加，之后再删除

        //单线程就意味着 前一个任务结束才会执行后一个任务

        //这样会导致：如果js执行时间过长，这样会造成页面的渲染不连贯，导致页面渲染加载阻塞的感觉

        console.log(1);

        setTimeout(function(){

            console.log(3);

        },1000);

        console.log(2);

      //这段代码原先是 先打印了1，过了1秒才打印3，所以等1秒钟之后才能打印2

        //为解决该问题 最新的标准允许JS脚本创建多个线程， 于是JS就出现了同步和异步 （但其实还是单线程的）

        //同步：前一个任务结束后再执行下一个任务，程序执行顺序与排列顺序是一致的，同步的

        //异步：做一件事情时可能会花费很长时间，这段时间内可以去处理其他事情

        //所以因为异步上一段代码先打印了1，后打印了2 再打印了3

        console.log(1);

        setTimeout(function(){

            console.log(3);

        },0);

        console.log(2);

        //即使时间是0，也是 打印 1 2 3

        //同步任务： 同步任务都在主线程上执行，形成一个执行栈

        //异步任务： JS的异步是通过回调函数实现的

        //一般而言，异步任务有以下三种类型：

        //1.普通事件：如 click， resize 等

        //2.资源加载：如；load，error 等

        //3.定时器里的函数，包括setInterval,setTimeout 等

        //异步任务相关的回调函数添加到任务队列中(任务队列也称为消息队列)

        //所以上段代码：同步任务 console.log(1); setTimeout(function()，1000);console.log(2);

        //异步任务：  function(){console.log(3);}

        //执行完同步任务后会回去看异步任务(回去任务队列)里看是否有要执行的，有就拿回执行栈里执行，执行完毕后任务队列里就清除

        //主线任务执行完后还会再回头看异步任务里是否有新的要执行然后再回到主线任务 形成循环

        //由于主线程不断的重复获得任务，执行任务，再获取任务，再执行，所以这种机制被称为事件循环(event loop)

⑤this指向问题：

//this 的指向在函数定义的时候是确定不了的，只有在函数执行的时候才能确定this到底指向谁，

        //一般情况下this最终指向的是那个调用他的对象

        //全局作用域下或者普通函数中this指向全局对象window (注意定时器里面的this也指向window),因为window是可以省略的

        console.log(this);  //window

        function fn() {

            console.log(this);  //window

        }

        fn();

        window.setTimeout(function () {

            console.log(this);  //window

        }, 1000);

        //谁调用方法方法里的this就指向谁

        var o = {

            sayHi: function () {

                console.log(this);  //this 指向的是 o 这个对象

            }

        }

        //构造函数中的this指向构造函数的实例

        function Fun() {

            console.log(this);

        }

        var fun = new Fun();   //那这里this指向的就是 fun

⑥location对象：

//window对象提供了一个location属性用于获取或设置窗体的URL，并且可以用于解析URL。

        //因为这个属性返回的是一个对象，所以我们也将这个对象称为location对象

        //url 统一资源定位符 是互联网上标准资源的地址，互联网每个文件都有唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它

        //一般语法格式：protocol://host[:post]/path/[?query]#fragment

        // http://www.itcast.cn/index.html?name=andy&age=18#link

        //protocol: 通信协议 常用的http,ftp,maito等

        //host: 主机(域名)  www.itcast.cn

        //port：端口号 可选，省略时使用方案的默认端口 如http的默认端口为80

        //path: 路径 由零个或多个 '/' 符号隔开的字符串，一般用来表示主机上的一个目录或文件地址  index.html

        //query: 参数 以键值对的形式，通过 & 符号 分隔开来 name=andy&age=18

        //fragment: 片段 #后面内容 常见于链接 锚点

        //location对象属性

        //location.href  获取或者设置 整个URL

        //location.host  返回主机(域名) www.itcast.cn

        //location.port  返回端口后 如果未写返回 空字符串

        //location.pathname  返回路径

        //location.search    返回 ?参数  ?name=andy&age=18

        //location.hash      返回片段 #后面内容 常见于链接 锚点

        //优先记住href  search

⑦：navigator对象：

//navigator对象包含有关浏览器的信息，它有很多属性，我们最常用的是userAgent，该属性可以返回由客户机发送服务器的user-agent头部的值

        //下面代码可以判断用户哪个终端打开页面，实现跳转

        //if((navogator.userAgent.match(/phone|pad....))){

          //  Window.loction.href = "对应手机页面的网页 "  //手机

        //}

⑧：history对象：

//history对象：与浏览器历史记录进行交互，该对象包含用户(在浏览器窗口中)访问过的URL

        //对象方法

        //back()     后退功能

        //forward()  前进功能

        //go(参数)   前进后退功能 参数如果是1前进一个页面  -1后退一个页面

        var btn = document.querySelector('button');

        btn.addEventListener('click',function(){

            //history.back();

            //history.forward();

            history.go(1);

            //history.go(-1);

        })

2.PC端网页特效：

①元素可视区client系列

//client :客户端，我们利用client系列的相关属性来获取元素可视区的相关信息，通过client系列的相关属性可以动态的得到该元素的边框大小，元素大小等

        // 属性：

        // element.clientTop    返回元素上边框的大小

        // clientLeft           返回元素左边框的大小

        // clientWidth          返回自身包括padding，内容区的宽度，不含边框，返回数值不带单位

        // clientHeight          返回自身包括padding，内容区的高度，不含边框，返回数值不带单位

②元素偏移量offset系列：

//offset：偏移量  利用offset系列相关属性可以动态的得到该元素的位置(偏移)，大小等

        //获得元素 距离 带有定位父元素的位置

        //获得元素自身的大小（宽度高度）

        // 注意：返回的数值都不带单位

        // offset系列常用属性：

        // element.offsetParent    返回作为该元素带有定位的父级元素  如果父级都有没有定位则返回 body

        // element.offsetTop       返回元素相对 带有定位父元素 上方的偏移

        // offsetLeft              返回元素相对 带有定位父元素 左边框的偏移

        // offsetWidth             返回自身包括 padding，边框，内容区的 宽度，返回值不带单位

        // offsetHeight            返回自身包括 padding，边框，内容区的 高度，返回值不带单位

③滚动 scroll系列

//scroll 滚动的，使用scroll系列相关属性可以动态的得到该元素的大小，滚动距离等

        // scrollTop      返回被卷去的上侧距离，数值不带单位   数值包括上方被卷去内容高度+ 自身上边框的高度(从上边框下部向上就是被卷上去的部分)

        // scrollLeft     返回被卷去的左侧距离，数值不带单位

        // scrollWidth    返回自身实际的宽度，不含边框，返回数值不带单位  超出也计算超出部分

        // scrollHeight   返回自身实际的高度，不含边框，返回数值不带单位  超出也计算超出部分

        var div = document.querySelector('div');

        console.log(div.scrollHeight);

        console.log(div.offsetHeight);

        //如果浏览器的高（或宽）不足以显示整个页面时，会自动出现滚动条，

        //页面上面被隐藏掉的高度，我们就称为页面被卷去的头部，滚动条在滚动时会触发onscroll事件

        div.addEventListener('scroll',function(){

            console.log(div.scrollTop);

        })

        //如果要找页面被卷去的头部  可以通过window.pageYOffset获得，  左侧：window.pageYOffset

④：立即执行函数

function fn() {

            console.log(1);

        }

        fn(); //还需要调用才能使用

        //立即执行函数：不需要调用，立马能够自己执行的函数

        //(function() {})()    或  (function(){}());

        (function (a, b) {

            console.log(a + b);

        })(1, 2)   //第二个小括号可以是看作调用函数，第一个小括号是参数

            (function (a, b) {

                console.log(a + b);

            }(1, 2));

        //立即执行函数最大的作用是 独立创建了一个作用域，避免了命题冲突问题

⑤动画函数封装

//因为我们可能经常使用动画函数，每一次使用就写一次的话就会很麻烦，所以就对动画函数进行封装，使用的时候就引入该JS文件然后调用该动画函数就可以了

//缓动动画就是让元素运动速度有所变化，就常见的就是让速度慢慢停下来

//思路：

// 1.让每次运动的距离慢慢变小，速度就回慢慢落下来

// 2.核心算法:（目标值 - 现在的位置） / 10  作为每次移动的距离(步长)  当然/10 /9 除几都可以

// 如：原来在了left = 0处，要运动到left = 30处， 那么第一次就运动了30px, d、第二次就移动(300 - 30) / 10 < 30 ，慢慢变小

// 3.停止定时器：在当前盒子位置等于目标位置就停止

//缓动动画函数封装 object 目标对象  target 目标位置

//给动画函数添加回调函数： 在动画执行完毕后再执行一个函数 如当运动完毕后盒子变色

//给动画函数再添加一个参数，然后把对应函数传入该参数，然后在运动完毕后执行

function animate(obj, target, callback) {

    //这里使用obj的timer属性来接收定时器而不直接 声明变量var timer

    //是为了避免每个调用动画函数的元素的定时器重名，让每个调用该函数的元素的定时器是独特的

    clearInterval(obj.timer);

    obj.timer = setInterval(function () {

        //步长公式: （目标值 - 现在的位置） / 10

        //但是这样会出现小数运算，可能会有误差 ，所以我们要把他改为整数运算

        //如果是向前走   即正数 大于某数的小数就往上取整

        //但如果是向后倒 即负数 该小数应该是向下取整 运动的距离 才会是向上取整 如 -8.1 取整到 -9

        //用三元表达式

        var step = (target - obj.offsetLeft) / 10

        step = step > 0 ? Math.ceil(step) : Math.floor(step);

        if (obj.offsetLeft == target) {

            clearInterval(obj.timer);

            //回调函数写到清除定时器里面，不然写在外边不构成回调函数，只是同步任务，而定时器是异步任务，

            //这样子会导致还没有运动完就已经调用函数了，所以要写在定时器 清除定时器里面

            // if (callback) {   //判断是否有回调函数 有就执行，没有就退出

            //     callback();

            // }

            callback && callback();   //和上面等价 利用短路与： 如果有callback 左边为真，那就会执行，否则就结束

        }

        obj.style.left = obj.offsetLeft + step + 'px';

    }, 30);   //30执行一次

}

1. jQuery

//js库：即library 是一个封装好的特定的集合(方法和函数)。从封装一大堆函数的角度理解库，

        // 就是在这个库中，封装了很多预先定义好的函数再里面，比如动画animate，hide，show，比如获取元素等

        // 简单理解就是一个JS问价，里面对我们原生JS代码进行了封装，存放到里面，这样我们可以快速高效的使用这些封装好的功能了

        //jQuery就是为了快速方便的操作DOM,里面基本都是函数(方法)

// jQuery 封装了JS常用的功能代码，优化了DOM，事件处理，动画设计和Ajax交互

        // 学习jQuery本质就是学习调用这些函数(方法)

        // 轻量级，核心文件在几十KB，不会影响页面加载速度

        // 跨浏览器兼容，基本兼容了现在主流的浏览器

        // 链式编程，隐式迭代

        // 对事件，样式，动画支持，大大简化了DOM操作

        // 支持插件扩展开发。有着方法的第三方的插件，例如：树形菜单，日期控件，轮播图等

        // 免费，开源

②基本使用

<script src="jQuery.min.js"></script>

    <!-- 在官网把jQuery代码复制到JS文件中，使用时再引入就可以了 -->

//等着页面DOM加载完毕再去执行

        // $(document).ready(function(){

        //     //利用jQuery将div盒子隐藏

        // $('div').hide();

        // })

        //第二种，更方便 入口函数

        $(function () {

            $('div').hide();

        })

        //jQuery的顶级对象$  是jQuery的别称，在代码中可以代替jQuery，一般方便都使用$

        //它相当于原生JS库中的window，把元素利用$包装成jQuery对象，就可以调用jQuery的方法

③jQuery对象和DOM对象的区别

//DOM对象： 用原生JS获取过来的对象就是DOM对象

        var mydiv = document.querySelector('div');  //DOM对象

        console.log(mydiv);

        $('div');   //jQuery对象

        console.log($('div'));

        //jQuery对象：用jQuery方式获取过来的对象就是jQuery对象  本质就是利用$对DOM对象包装后产生的对象(伪数组形式存储)

        mydiv.style.display = 'none';

        //mydiv.hide();   //报错

        $('div').hide();

        //$('div').style.display = 'none';   //报错

        //jQuery对象和DOM对象一定不能混用，是什么对象就用什么对象有的属性和方法

        //DOM对象和jQ对象之间是可以相互转换的，因为原生JS比jQuery大，

        //原生的一些属性和方法JQ没有封装，所以要使用这些属性和方法需要把jQ对象转换为DOM对象

        //DOM 转化为jQuery: $(DOM对象)

        $(mydiv);

        //jQuery 转化为DOM(两种方式)

        //$(mydiv)[index];

        //$(mydiv).get(index);    index是索引号

        $('div')[0];

        $('div').get(0);

④jQuery选择器

//选择器来选择元素

    //基础选择器

    // ID选择器   $('#id');        获取指定ID元素

    // 全选选择器    $('\*');          匹配所有元素

    // 类选择器      $('.class');     获取同一类class的元素

    // 标签选择器    $('div');        获取同一类标签的所有元素

    // 并集选择器    $('div,p,li');   选取多个元素

    // 交集选择器    $('li.current'); 交集元素

    //单引号双引号都可以

    //层级选择器

    // 子代选择器    $('ul>li');      使用 > 获得亲儿子层级的元素，不会获得孙级元素

// 后代选择器    $('ul li');     使用空格  代表后代选择器，获得ul下的所有li元素，包括孙子

//筛选选择器

// :first       $('li:first')      获取第一个li元素

    // :last        $('li:last')       获取最后一个li元素

    // :eq(index)   $('li:eq(2)')      获取到的li元素中，选择索引号为2的元素，索引号从0开始

    // :odd         $('li:odd')        获取到的li元素中，选择索引号为奇数的元素

    // :even        $('li:even')       获取到的li元素中，选择索引号为偶数的元素