Java Script

什么是JS

是一种弱类型脚本语言，其源代码不需经过编译，而是浏览器解释运行，用于控制网页的行为。

也是世界上最流行的脚本语言。一个合格的后端人员，必须要精通JS

记得再ide中将js设置为ES6版本 不然后面有可能报错嗷

Hello World

1.创建html文件

2.引入Script标签

3.一个最简单的弹窗

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    <script>
        alert("hello world");
    </script>
</head>
<body>

</body>
</html>
```

script标签一般放在head里或body最下方

我们也可以将js文件摘出，从外部引入js。这样做的话就要在html同级目录下创建js目录，书写js文件，我们这里就创建了JavaScript标签 书写了a.js.a.js的内容和原本Script框里的内容一致。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    //尽量不要自闭和哦
    <script src="JavaScript/a.js">
    </script>

</head>

<body>

</body>
```

```
</html>
```

js基本语法入门及数据类型

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        // 1 定义变量 变量类型 变量名 = 变量值
        // 变量名不能以数字开头
        var score = 75;

        // 2 条件控制
        if(score>60 && score<70 ){
            alert("60~70")
        }else if(score>70 && score<80){
            alert('70~80')
        }else{
            alert('other')
        }
        // console.log(score) 在浏览器控制台中打印变量值
        // == 等于（值一样） ===绝对等于（类型、值一样）避免使用==来判断
        //NaN与所有的数值不相等 只能通过isNaN（）来判断是否为NaN
        //尽量避免浮点数运算（精度问题 ）
        //java中同一数组的数据类型需要一致，但是JavaScript中可以不一致
        // 数组中超过范围会抱udifiend
        var arr =[1,2,3,4,'hello',5.5,null,true];
        // Person person = new Person(1,2,3,4,5);
        var person = {
            name: 'fanjuncheng',
            age:3,
            tags:[1,2,3,'js',2.5]
        }

    </script>
<body>

</body>
</html>
```

严格检查模式

因为js是一门很随便的语言，声明也能使用，例如以下

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
            //在控制台中，g依然可以打印出来
```

```
            g = 8;

    </script>
<body>

</body>
</html>
```

这种方式显的十分不严谨，所以ES5开始支持严格检查模式，在写入了"use strict"之后，未声明的变量将无法使用。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        //‘use strict’的目的是指定代码在严格条件下执行。严格模式下你不能使用未声明的变量。
        //局部变量建议使用 let 全局变量为var
        // 'use strict'
        var i = 8;
            g = 8;
    </script>
<body>

</body>
</html>
```

字符串

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        //正常字符串使用单引号或双引号包裹，\为转义字符
        'use strict';
        let  i=8;
        let  name = 'mouermou'
        //长字符串 `` (esc下 tab上)
        let msg = `1 2
        3
        4
        5,${name}
        `
        //字符串长度 str.length
        //字符串不可变
        //大小写转化 toUpperCase toLowerCase
        //获取下标indexOf
        //substring（1）/ （1，3） 从第一个字符截取到最后一个字符/第一个到第二个，包含前面不包含后面
```

```
        let g = `123+${name}`
    </script>
<body>

</body>
</html>
```

数组

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<script>
    'use strict'
    //Array 可以包含任意的数据类型
    //长度 arr.length 若给arr.length 复制 数组大小会发生变化 赋值过小会导致元素丢失
    let arr= ['a',2,3,4,5.5,1,'abc'];
    //indexOF 通过元素获得下标索引
    //slice() 截取Array的一部分，返回一个新数组
    // push 压入元素到尾部
    // pop  弹出尾部元素
    // unshift(),shift() 头部压入与弹出
    //sort 元素排序
    let array = [6,5,4,3];
    //reverse 元素反转
    //concat() 拼接数组 但是没有改变原数组
    //join() 打印拼接数组，使用特定的字符串连接

    //二维数组
    let arr1=[[1,2,3][3,2,1][5,6,7]];


</script>
<body>

</body>
</html>
```

对象

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        'use strict'
        // var 对象名 = {
        //      属性名：属性值
        //      属性名：属性值
        // }
```

```
        //key都是字符串，值是任意对象
        var person = {
            name:"wuwuwu",
            age: 18,
            email: "22222",
            score: 0
        }
        //使用一个不存在的属性不会报错 underfined
        //delete 删除对象属性
        //动态添加 直接给新的属性添加值即可
        // 判断属性值是否存在这个对象中 xxx in xxx
        // 判断一个属性是否是这个对象自身拥有的hasOwnProperty()
    </script>
<body>

</body>
</html>
```

流程控制

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<script>
    //if 判断
    // let age = 3;
    // if(age>3){
    //     alert("haha");
    // }else if(age<5){
    //     alert("kuwa");
    // }else{
    //     alert("dudu")
    // }

    //while 循环
    // while(age<100){
    //     age  = age+1;
    // }
    //     do{
    //         age = age+1;
    //     }while(age<100)

    //for循环
    //     for(let i = 0; i<100;i++){
    //         console.log(i)
    //     }

    // foreach循环
    let age = [12,432,5432,341,54];
    //   age.forEach(function (value) {
    //       console.log(value);
    //   }
    //for in 循环
```

```
    for(let num in age){
        console.log(age[num])
    }
</script>
<body>

</body>
</html>
```

Map and Set

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        'use strict'
        let map = new Map([['tom',100],['jack',60],['alex',80]]);
        let score = map.get('tom');
        //map.set('liwei',200) 新增
        //map.delete('liwei') 删除
        console.log(score)
        //set 无序不重复的集合->set可以去重
        let set = new Set([3,1,1,1]);
        //set.add(2); 增
        //set.delete(1); 删
        // console.log(set.has(3)); 是否包含

        // 遍历数组 map set 均可以用 for of的方式
        let arr = [3,4,5];
        for(let x of arr){
            console.log(x)
        }

    </script>
<body>

</body>
</html>
```

自定义函数

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        //定义方式一
        //一旦执行到return代表函数结束，返回结果
        //如果没有执行return,函数执行完也会返回结果，结果就是undefined
        // function abs(x) {
```

```
        //     if(x>=0){
        //         return x;
        //     } else{
        //         return -x;
        //     }
        // }

        //定义方式二
        let abs2 = function(x){
            if(x>=0){
                return x;
            }else{
                return -x;
            }
        }

        //函数可以被传递多个值，但只会使用第一个符合规定的值
        //arguments是js的关键字，代表传递进来的所有参数，是一个数组
        //rest获取除了已经定义的参数之外的参数
        //rest参数只能写在最后 必须用...标识
        function aaa(a,b,...rest){
            console.log('a->'+a);
            console.log('b->'+b);
            console.log(rest);

        }

    </script>
<body>

</body>
</html>
```

变量作用域

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>
        // 在函数体中声明，则在函数体外不可使用
        // function f() {
        //     let x = 1;
        //     x = x + 1;
        // }
        // x = x+2
        //如果两个函数使用了相同的变量名，只要在函数内部，就不会冲突
        //内部函数可以访问外部函数的成员，反之则不行
        //假设内部函数变量和外部重名 ，js中函数查找从自身函数开始，由内向外查找
        //假设外部存在这个同名的函数变量，则内部函数会屏蔽外部函数的变量
        // function f1() {
        //     let x = 1;
        //
        //     function f2() {
```

```
        //         let x = 2;
        //         console.log('inner'+x);
        //     }
        //
        //     console.log('outer'+x);
        //     f2()
        // }

        //默认的所有全部变量都会绑定在window对象下
        // alert本身也是window下的一个变量
        // var x = 'xxx';
        // alert(x);
        // alert(window.x);
        // window.alert(x);

        // 由于我们所有的全局变量会默认绑到window上,所以如果有不同的js文件使用了相同的
        // 全部变量，就容易引起冲突，故把自己的代码全部放入自己定义的唯一空间名字中，降低冲突
的可能性
        // var funi = {}; //唯一全局变量
        // funi.name = "fjc" //定义全局变量
        // funi.add = function (a,b) {
        //     return a+b;
        // }


    </script>
<body>

</body>
</html>
```

函数方法

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
    <script>

        function getAge() {
            var now = new Date().getFullYear();
            return now-this.brith
        };

        var funi = {};
        funi.name = 'funi';//属性
        funi.brith = 1998;
        funi.age = getAge()
        //     function () {//方法
        //     var now = new Date().getFullYear();
        //     return now-this.brith
        // }
        //函数方法中的 apple方法可以重定向this指向的值
```

```
        getAge.apply(funi,[]);

    </script>
<body>
```