类的继承

```
<!-- 类继承 -->
   <script>
       // ES5构造函数的继承

       // 手机
       function Phone(brand, price) {
           this.brand = brand;
           this.price = price;
       };

       Phone.prototype.call = function () {
           console.log('I can cell somebody');
       };

       // // 智能手机
       function SmartPhone(brand, price, color, size) {
           Phone.call(this, brand, price);
           this.color = color;
           this.size = size;

       };

       // // 设置子级构造函数的原型
       SmartPhone.prototype = new Phone;
       // // 校正
       SmartPhone.prototype.constructor = SmartPhone;

       // // 声明子类的方法
       SmartPhone.prototype.photo = function () {
           console.log('I can take photograph');
       };

       SmartPhone.prototype.playGame = function () {
           console.log('I can play game');
       };

       const iphone = new SmartPhone('iphone', 4999, 'black', '4.7inch');
       console.log(iphone);


       // 类继承
       class Phone{
           constructor(brand,price){
               this.brand=brand;
               this.price=price;
           }

           // 父类的成员属性
           call(){
               console.log('I can cell somebody');
           }
       };

       class SmartPhone extends Phone{
            // 构造方法
            constructor(brand,price,color,size){
                super(brand,price);
                this.color=color;
                this.size=size;
            }

            photo(){
                console.log('I can take aphotograph');
            }

            playGame(){
                console.log('I can play game');
            }

            // 子类对父类方法的重写
            // 子类不能直接调用父类的同名方法
            call(){
                console.log('I can make a video call');
            }
        };

        const xiaomi= new SmartPhone('xiaomi',1999,'red','5.5inch');
        console.log(xiaomi);
        xiaomi.call();
        xiaomi.photo();
        xiaomi.playGame();
    </script>
```

class 的 get 和 set

```
<script>
        // get & set
       class Phone{
        //    get 通常对对象的动态属性做封装
           get price(){
               console.log('The price attibute is read');
               return '价格';
           }

        //    set可以添加更对的控制和判断
           set price(newVal){
               console.log('The price has been modified');
           }
       }

    //    实例化对象
    let s=new Phone();
    // console.log(s.price);
    s.price='free';
    </script>
```

数值扩展

```
<script>
    // 数值扩展
    // Number.EPSILON 是JavaScript表示的最小的精度  是一个非常小的数
    function equal(a, b) {
        if (Math.abs(a - b) < Number.EPSILON) {
            return true;
        } else {
            return false;
        }
    };
    console.log(0.1 + 0.2 === 0.3);
    console.log(equal(0.1 + 0.2, 0.3));


    // 1. 二进制和八进制
    let b = 0b1010;
    console.log(b);
    let o = 0o777;
    console.log(o);
    let d = 100;
    console.log(d);
    let x = 0xff;
    console.log(x);


    // 2.Number.isFinite   检测一个数值是否为有限数
    console.log(Number.isFinite(100));
    console.log(Number.isFinite(100 / 0));
    console.log(Number.isFinite(Infinity));


    // 3. Nubmer.isNaN  检测一个数值是否为NaN
    console.log(Number.isNaN(123));


    // 4. Number.parseInt   Number.parseFloat  字符串转整数/浮点数
    console.log(Number.parseInt('5211314love')); //会截断  输出5211314
    console.log(Number.parseFloat('1.23456789神奇')); //会截断   输出1.23456789


    // 5. Nubmer.isInteger   判断一个数是否为整数
    console.log(Number.isInteger(5));
    console.log(Number.isInteger(5.1));


    // 6. Math.trunc    将数字的小数部分抹掉
    console.log(Math.trunc(3.1415));


    // 7. Math.sign    检测一个数到底是正数 负数 还是0
    console.log(Math.sign(100));
    console.log(Math.sign(0));
    console.log(Math.sign(-10));
</script>
```

对象方法扩展

```
<script>
    // 1. object.is  判断两个值是否完全相等
    console.log(Object.is(120, 120));
    console.log(Object.is(NaN, NaN)); //true
    console.log(NaN === NaN); //false


    // 2. Object.assign   对象的合并
    const config1 = {
        host: 'localhost',
        port: 3306,
        name: 'root',
        password: 'root',
        test:'test'
    }

    const config2 = {
        host: '127.0.0.1',
        port: 33060,
        name: 'Spongebob',
        password: 'password'

    }

    const config = Object.assign(config1, config2);
    console.log(config);


    // 3. Object.setPrototypeOf设置原型对象   Object.getPrototypeOf
    const school={
        name:'CUGB',
    }

    const cities={
        xiaoqu:['Beijing','Wuhan']
    }

    Object.setPrototypeOf(school,cities);
    console.log(Object.getPrototypeOf(school));
    console.log(school);

</script>
```

模块化

```
<script type="module">
    // 通用的导入方式
    // 引入m1.js模块内容
    import * as m1 from"./m1.js";
    console.log(m1);

    // 引入m2.js的模块内容
    import * as m2 from "./m2.js";
    console.log(m2);

    // 引入m3.js内容
    // import * as m3 from "./m3.js";
    // console.log(m3);
    // m3.default.workplace();

    // 2. 解构赋值的形式
    import {school,teach} from "./m1.js";
    console.log(school);
    console.log(teach);

    import {school as bd,study} from "./m2.js";
    console.log(bd);
    console.log(study);

    // import {default as m3} from "./m3.js";
    // console.log(m3);


    // 3. 简便形式  只能针对默认暴露
    import m3 from "./m3.js";
    console.log(m3);

</script>
```

ES7新特性

```
 <script>
        // includes indexOf
        const mingzhu=['西游记','红楼梦','三国演义','水浒传'];

        // 判断
        console.log(mingzhu.includes('西游记'));

        // **
        console.log(2**10);    //相当于Math.pow(2,10)
    </script>
```

ES8对象方法扩展

```
<script>
    // 声明对象
    const school={
        name:'CUGB',
        cities:['Beijing','Wuhan'],
        xueke:['dizhi','tumu','zhubao']
    };

    // 获取对象所有的键
    // console.log(Object.keys(school));

    // 获取对象所有的值
    // console.log(Object.values(school));

    // entries
   const m=new Map(Object.entries(school));
   console.log(m.get('cities'));

    // 对象属性的描述对象
   console.log(Object.getOwnPropertyDescriptors(school));
</script>
```

ES9 新特性

```
<script>
    // Rest 参数与spread扩展运算符在ES6中已经引入，不过ES6中只针对于数组，在ES9中为对象提供了像

    function connect({
        host,
        port,
        ...user
    }) {
        console.log(host);
        console.log(port);
        console.log(user);


    }

    connect({
        host: '127.0.0.1',
        port: 3306,
        username: 'root',
        password: 'root'
    })


    const skillOne = {
        q: '天音波'
    };
    const skillTwo = {
        w: '金钟罩'
    };
    const skillThree = {
        e: '天雷破'
    };
    const skillFour = {
        r: '猛龙摆尾'
    }

    const mangseng={...skillOne,...skillTwo,...skillThree,...skillFour};

    console.log(mangseng);
</script>
```

ES9正则扩展

```
<script>
    // 命名捕获分组

    // 声明一个字符串
    let str = '<a href="http://www.baidu.com/">Baidu</a>';

    // 提取URL与标签文本
    // 之前的处理
    const reg=/<a href="(.*)">(.*)<\/a>/;

    const result=reg.exec(str);
    console.log(result[1]);
    console.log(result[2]);

    // 现在的做法
    const reg = /<a href="(?<url>.*)">(?<text>.*)<\/a>/;

    const result = reg.exec(str);
    console.log(result);



    // 反向断言
    // 正向断言 根据当前匹配的后面的内容判断前面的内容是否满足条件
    let str = 'abcdefg1234567测试测试789结尾';
    const reg = /\d+(?=结)/;
    const result = reg.exec(str);
    console.log(result);

    // 反向断言 根据当前匹配的前面的内容判断后面的内容是否满足条件
    const reg=/(?<=测试)\d+/;
    const result=reg.exec(str);
    console.log(result);



    // 正则dotAll模式
    // dot .  元字符   除换行符以外的任意单个字符
    // （先略过）
</script>
```

ES10 Object.fromEntries

```
<script>
    // 接收二维数组
    const result=Object.fromEntries([
        ['name','Sponge'],
        ['friend','派大星,章鱼哥,蟹老板,珊迪']
    ]);

    console.log(result);

    // Map
    const m=new Map();
    m.set('name','spongebob')
    const result=Object.fromEntries(m);
    console.log(result);

    // ES8中 Object.entries可以将一个对象转化为一个数组
    // Object.fromEntries可以将数组转化为二维对象
    // 所以Object.entries和Object.fromEntries可以算作一个逆运算
    const arr=Object.entries({
        name:'Spongebob'
    })
    console.log(arr);
</script>
```

ES10字符串扩展方法

```
<script>
    // 指定清除一个字符串左侧或者右侧的空白字符
    let str='                   海绵宝宝               ';
    console.log(str);
    console.log(str.trimStart());    //清除左侧空格
    console.log(str.trimEnd());      //清除右侧空格
</script>
```

```
<script>
    // flat  将多维数组转化为低维数组
    // const arr=[1,2,3,4,[5,6]];
    // console.log(arr.flat());

    // const arr1=[1,2,3,4,[5,6,[7,8,9]]];
    // 转换为2维数组
    // console.log(arr1.flat());
    // 转换为1维数组  参数为深度
    // console.log(arr1.flat(2));

    // flatMap
    const arr2 = [1, 2, 3, 4];
    // const result=arr2.map(item=>item*10);
    // console.log(result); //[10,20,30,40]
    const result = arr2.flatMap(item => [item * 10]);
    console.log(result);
</script>
```

symbol扩展

```
<script>
    // 创建symbol
    let s=Symbol('Spongebob');

    console.log(s.description);
</script>
```

ES11 私有属性

```
<script>
    class Person {
        // 公有属性
        name;
        // 私有属性
        #age;
        #weight;
        // 构造方法初始化
        constructor(name, age, weight) {
            this.name = name;
            this.#age = age;
            this.#weight = weight;
        }

        intro(){
            console.log(this.name);
            console.log(this.#age);
            console.log(this.#weight);
        }
    }

    // 实例化
    const girl = new Person('Mary', 18, '55kg');
    console.log(girl);
    console.log(girl.name);
    // console.log(girl.#age);
    // console.log(girl.#weight);
    girl.intro(); //可以调用
</script>
```

Promise.allSettled

```
<script>
    // 声明两个promise对象
    const p1=new Promise((resolve,reject)=>{
        setTimeout(()=>{
            resolve('商品数据--1');
        },1000)
    });

    const p2=new Promise((resolve,reject)=>{
        setTimeout(()=>{
            // resolve('商品数据--2');
            reject('出错了');
        },1000)
    });

    // 调用allsettled方法   返回的结果始终是成功的
    // const result=Promise.allSettled([p1,p2]);
    // console.log(result);

    // 区别all 方法 都用在做一些批量异步任务的场景，但是all方法根据每个对象的状态返回结果，都成
    const res=Promise.all([p1,p2]);
    console.log(res);
</script>
```

可选链操作符

```
<script>
    // 对象层级比较深，可以用可选链操作符
    // ?.

    function main(config) {
        const dbHost = config?.db ?.host;
        console.log(dbHost);
    }

    main({
        db: {
            host: '127.0.0.1',
            username: 'root'
        },
        cache: {
            host: '123.234.567',
            username: 'admin'
        }
    })
</script>
```

BigInt

```
<script>
    // 大整型
    let n=123n;
    console.log(n,typeof(n));

    // 函数
    let n1=123;
    console.log(BigInt(n1));
    // console.log(BigInt(1.2));  //不可以使用浮点数

    // BigInt主要用于大数值运算
    // 最大安全整数
    let max=Number.MAX_SAFE_INTEGER;
    console.log(max+2);  //不能正常运算
    // BigInt不能与正常的数值做运算，必须与BigInt做运算
    console.log(BigInt(max)+BigInt(1));
    console.log(BigInt(max)+BigInt(2));
</script>
```

GlobalThis

```
<script>
    // globalThis 始终指向全局对象
    console.log(globalThis);
</script>
```