1. 模板字符串：

1)模板字符串的语法：`` [反引号]

2)模板字符串的用途：用来简化字符串的拼接。

3)模板字符串的特点:

在模板字符串中可以引用变量、函数调用、运算、换行。

4)模板字符串的应用：

    <script>

        //模板字符串语法： `` 反引号

        //模板字符串用途：  简化字符串拼接

        //模板字符串特点：使用变量、函数调用、运算、换行

        let age = 21;

        let uname = 'tom';

        let n = 3;

        // let str = "姓名为：<a href=''></a>"+uname+" 年龄为："+age;

        let str = `姓名为：${uname} 年龄为:${age}

                   函数调用结果为：${fn()}

                   运算结果为：${n\*9}`;

        function fn() {

            return 'Hello WEB';

        }

        console.log(str);

    </script>

1. 数组扩展：Array.from(v) 将伪数组对象或可遍历对象转换为真数组

<body>

    <div>div1</div>

    <div>div2</div>

    <div>div3</div>

    <div>div4</div>

    <div>div</div>

    <script>

        let divs = document.getElementsByTagName('div');

        console.log(divs);

        //转换成真正的数组

        let arr = Array.from(divs);

        console.log(arr);

        //注意：将可遍历的对象转换成数组时要在对象中添加"length"属性

        let obj = {

            "0": 1,

            "1": 2,

            "2": 3,

            "length": 3

        };

        console.log(obj);

        let arr2 = Array.from(obj);

        console.log(arr2);

    </script>

</body>

1. 对象简写形式：就是在对象中可以写变量和函数，来作为对象的属性和方法。

    <script>

        //对象简写形式有两种：属性名简写、方法名简写

        let age = 21;

        let usr = "demo";

        // let obj = {

        //     "age": age,

        //     "usr": usr

        // };

        //属性名简写：当属性名与属性值同名时可以简写

        let obj = {

            age,

            usr

        };

        console.log(obj);

        //方法简写：

        let obj2 = {

            usr,

            age,

            // fn: function() {

            //     console.log('fn...')

            // }

            fn() {

                console.log('fn2...');

            }

        };

        console.log(obj2);

        obj2.fn();

    </script>

1. 对象简写形式应用：

<body>

    帐号：<input type="text" id="uname" name="uname"><br> 密码：

    <input type="password" id="pwd" name="pwd"><br> 确认密码：

    <input type="password" name="repwd"><br> 邮箱：

    <input type="text" id="email" name="email"><br>

    <input onclick="sendData()" type="button" value="注册"><br>

    <script>

        function sendData() {

            let uname = $("#uname").val();

            let pwd = $("#pwd").val();

            let email = $("#email").val();

            $.post('???', {

                uname,

                pwd,

                email

            }, function(d) {}, 'json');

        }

    </script>

</body>

1. 属性名表达式： 对象的属性名可以写表达式，而在es5中对象的属性名只能是字符串。

    <script>

        //什么表达式？

        //如：2\*n

        //var m = 2\*9;

        let obj2 = {

            "m": 1,

            "n": 33

        };

        for (var i in obj2) {

            console.log(obj2[i], 777)

        }

        let str = 2;

        let y = "usr3";

        let obj = {

            "usr": "demo",

            ["usr" + str]: "tom",

            [y]: "lisi"

        };

        console.log(obj, obj[y]);

    </script>

**作业:**

产生一个对象:{A:[],B:[],C:[],D:[]...Z:[]}，提示：可以考虑使用属性名表达式

1. 扩展运算符:

1)扩展运算符语法： ...

2)扩展运算符主要功能:

(1)类似于arguments，收集数据功能

(2)类似于apply，数据展开功能

    <script>

        //数据展开：

        let arr = [1, 3, 5];

        // let arr2 = [7, 9, ...arr];

        let arr2 = [...arr, 7, 9];

        //  console.log(arr2);

        let arr3 = [...arr];

        //  console.log(arr3, arr);

        //注意：最好在谷歌浏览器中运行下面的代码

        let obj = {

            "x": 1,

            "y": 21

        };

        let obj2 = {

            fn() {

                console.log('fn...');

            }

        };

        let obj3 = {

            "data": {...obj

            },

            "method": {...obj2

            }

        }

        console.log(obj3);

        //收集数据：

        //  function fn(x, y, m, n) {

        function fn(...arg) { //收集数据

            console.log(arg);

            let xy = [...arg];

            //es5

            //    demo(arg[0], arg[1], arg[2], arg[3]);

            demo(...arg); //展开数据

        }

        function demo(m, n, i, p) {

            console.log(m, n, i, p);

        }

        fn(1, 2, 5, 8)

    </script>

1. 扩展运算符应用：

    <script>

        //在数组解构赋值中应用

        let arr = [1, 2, 3, 4];

        let [, x, ...y] = arr;

        console.log(x, y);

        //

        let str = "testing";

        let arr2 = [...str];

        console.log(arr2);

    </script>

1. 箭头函数主要用途：箭头函数一般用于回调函数，而回调函数通常是匿名函数

1. 箭头函数语法： ([形参...]) =>{}
2. 箭头函数特点：

1)箭头函数不同于一般函数:

(1)一般函数可以先调用后定义，箭头函数只能先定义后调用；

(2)一般函数有arguments，箭头函数没有arguments，但可以使用rest参数；

(3)一般函数可以当作构造函数，箭头函数不能当作构造函数，因为箭头函数没有自己的this；

    <script>

        //箭头函数不同于一般函数：

        //1)一般函数可以先调用后定义，而箭头函数只能先定义使用调用

        //2)一般函数有arguments，而箭头函数没有arguments，但可以使用rest参数

        //3)一般当作构造函数，而箭头函数不能当作构造函数,因为箭头函数没有自己的this

        //一般函数可以先调用后定义

        //  fn(); //调用函数

        function fn() {

            console.log('testing..', arguments);

        }

        //实例化

        // new fn();

        //箭头函数不能先调用

        //  demo(); // demo is not a function

        var demo = (...x) => {

            console.log(x);

            //arguments is not defined

            //箭头函数没有arguments

            //console.log('demo...', arguments);

        }

        // demo(1, 3, 4);

        //实例化

        //箭头函数不能当作构造函数

        //  new demo(1, 3, 4); //demo is not a constructor

    </script>

2)箭头函数简写形式：

(1)如果箭头函数只有一个形参时可以省略圆括号；

(2)如果箭头函数体只有一条语句并且该语句会作为返回值返回时，可以省略花括号及return关键词；

   <script>

        //箭头函数语法：

        //   ([形参...])=>{函数体}

        //箭头函数简写形式：

        //1)当箭头函数只有一个形参时，可以省略圆括号;

        //2)当箭头函数体只有一条语句并且该语句的结果会作为返回值返回时，可以省略花括号及return

        // var fn = (x) => {

        //     return x \* 9;

        // }

        var fn = x => x \* 9;

        let rtn = fn(9);

        console.log(rtn);

    </script>

3)箭头函数中的this依赖外层函数中的this指向，也就是说箭头函数中的this指向定义时所在的对象，而不绑定当前调用者；

    <script>

        //箭头函数this指向 ：箭头函数中的this指向依赖外层函数中的this指向,箭头函数中的this不绑定当前调用者

        function fn() {

            console.log(this, 111);

            setTimeout(function() {

                console.log(this);

            }, 1000);

            setTimeout(() => {

                console.log(this);

            }, 2000);

        }

        //   fn(); // window   window

        //   new fn(); // window  fn{}

        var brand = "苹果";

        var obj = {

            brand: "华为",

            fn: function() {

                setTimeout(function() {

                    console.log(this, this.brand);

                }, 1000);

                setTimeout(() => {

                    console.log(this, this.brand);

                }, 2000);

            }

        }

        // obj.fn(); //苹果   华为

        var price = 5000;

        var obj2 = {

            price: 8000,

            fn: () => {

                setTimeout(function() {

                    console.log(this, this.price);

                }, 1000);

                setTimeout(() => {

                    console.log(this, this.price);

                });

            }

        }

        obj2.fn();

    </script>

    <script>

        let obj = {

            age: 20,

            demo: function() {

                console.log(this);

            }

        }

        //obj.demo();  // Ojbect

        //  obj.demo.call(window);

        function demo() {

            console.log(this);

            setTimeout(() => {

                console.log(this);

            }, 1000);

        }

        // demo();  //window   window

        //    demo.call(new Object()); // {} {}

        var demo = () => {

            console.log(this);

            setTimeout(() => {

                console.log(this);

            }, 2000);

        }

        demo.call(new Object());

    </script>

作业：参考着上面的例子定义:

1. 当外层是一般函数、内层也是一般函数，
2. 当外层是箭头函数、内层是一般函数，
3. 箭头函数不适用的场景:

<body>

    <button id="btn">测试</button>

    <script>

        //对象中的方法不能使用箭头函数

        let obj = {

            age: 22,

            usr: "lisi",

            fn: function() {

                console.log(`姓名:${this.usr}  年龄:${this.age}`);

            },

            fn2: () => {

                console.log(`姓名:${this.usr}  年龄:${this.age}`);

            }

        };

        // obj.fn();

        // obj.fn2();

        //给dom绑定事件函数时不要使用箭头函数

        var curbtn = document.getElementById('btn');

        // curbtn.addEventListener('click', function(e) {

        //     console.log(this);

        // });

        curbtn.addEventListener('click', (e) => {

            console.log(this);

        });

    </script>

</body>

1. 什么是Symbol

是es6 新增的数据类型，和原有的数据类型(String、Number、Boolean、Object、null、undefined)一样。

1. Symbol类型的特点:

唯一性、不能参与运算(包括字符串拼接)、for...in或for...of不能遍历

    <script>

        //Symbol简介：

        //Symbol是es6新增的数据类型，和原有数据类型(String、Number、boolean、null、undefined、Object)一样

        //Symbol功能：用来产生唯一值

        // let x = Symbol();

        // let y = Symbol();

        // console.log(x);

        // console.log(x === y);

        //Symbol('备注')

        let price = Symbol('价格');

        let brand = Symbol('品牌');

        console.log(price, brand);

    </script>

1. Symbol应用：

    <script>

        let obj = {

            'brand': '苹果',

            'price': 8000

        };

        obj.brand = '梨';

        console.log(obj.brand);

        let obj2 = {};

        let brand = Symbol('品牌');

        obj2[brand] = '苹果手机';

        obj2.brand = '梨';

        console.log(obj2, obj2[brand]);

        //Symbol不能使用for...in或for...of来遍历

        for (var i in obj2) {

            console.log(i);

        }

        //Symbol不能参与运算

        let s = brand + 'ok';

    </script>

1. Promise用途： 主要用来解决异步编程中数据传递的问题（可以解决回调地狱问题）
2. 什么是同步与异步？

什么是同步?

        同步就是当调用发生之后在没有得到结果之前，后续代码不能正常执行。

        什么是异步？  
        异步就是发调用发生之后在没有得到结果之前，后续代码可以正常执行，而这次调用结果会以通知或回调函数来告诉调用者。

    <script>

        //异步：

        console.log(1);

        setTimeout(function() {

            console.log(2);

        }, 1000);

        console.log(3);

    </script>

1. 异步编程中的数据传递:

    <script>

        //同步编程中的数据传递：

        function demo(x, y) {

            return x \* y;

        }

        // demo(2, 6);

        //异步编程中的数据传递:使用回调函数来解决

        function fn(v, calls) {

            //console.log(v, calls);

            setTimeout(function() {

                calls(v \* 8);

            }, 1000);

        }

        var callback = function(d) {

            console.log(d, 8888)

        };

        fn(8, callback)

    </script>

1. 在es5中使用回调函数可以解决异步编程中数据传递问题，但当业务比较复杂时代码的可读性、可维护性就变差。

    <script>

        $.ajax({

            url: "http://106.13.114.114:5000/api/firstCategory", // 所有一级分类

            dataType: "json",

            success(res) {

                $.ajax({

                    url: `http://106.13.114.114:5000/api/secondCategory`, // 传递一级ID换取下属的二级分类列表

                    data: {

                        firstId: res['list'][0][0]['firstId']

                    },

                    dataType: "json",

                    success(res2) {

                        $.ajax({

                            url: `http://106.13.114.114:5000/api/thiredCategory`, // 传递二级分类ID, 获取下属的三级分类列表

                            data: {

                                secondId: res2['list'][0]['secondId']

                            },

                            dataType: "json",

                            success(res3) {

                                $.ajax({

                                    url: `http://106.13.114.114:5000/api/categoryList`, // 传递三级分类ID, 获取下属的商品数据列表

                                    data: {

                                        thiredId: res3['list'][0]['thiredId']

                                    },

                                    dataType: "json",

                                    success(result) {

                                        console.log(result);

                                    }

                                })

                            }

                        })

                    }

                })

            }

        })

    </script>

1. 什么是Promise?

Promise实际上是一个许诺器，因些promise里面的代码通常是将来要执行的代码，而这些将来要执行的代码一般都是异步的I[input输入]/O[output输出]操作(比如：操作文件、操作数据库)，这些操作执行完成后会有两种结果:成功或失败，因此promise有三种状态：resolve(成功)、reject(失败)、pending(初始)

1. Promise的基本用法：

    <script>

        //注意：promise里面的代码会以同步方式来执行

        // console.log(1);

        // let prm = new Promise((resove, reject) => {

        //     //将来要执行的异步操作代码

        //     console.log(2); //同步执行

        // });

        // console.log(3);

        //  console.log(prm);

        //resolve:成功时要执行的回调函数

        //reject:失败时要执行的回调函数

        let prm = new Promise((resolve, reject) => {

            setTimeout(function() {

                //console.log('Hello...');

                //需求：把"Hello..."字符串传递出来？

                // resolve('Hello...'); //调用方法

                //   reject('error msg');

            }, 2000);

        });

        prm.then(function(d) {

            console.log('成功...', d);

        }, function(err) {

            console.log('失败...', err);

        })

    </script>

**作业**：使用Promise+ajax技术调用接口：[http://106.13.114.114:5000/api/firstCategory，并使](http://106.13.114.114:5000/api/firstCategory，并敷衍)用Promise实现异步编程数据传递。

1. Promise异常处理：

如果把失败时的回调函数定义在then()方法中的第二个参数中，那么可以捕获到Promise里面的代码异常，但不能捕获到then()中第一个参数中的成功时回调函数里面的代码异常；如果把失败时的回调函数定义在catch()方法中，不仅可以捕获到Promise里面的代码异常，还可以捕获到then()中第一个参数中的成功时回调函数里面的代码异步。

    <script>

      let prm = new Promise((resolve, reject) => {

            // console.log(x);

            // let x = 3;

            // resolve('成功');

            try {

                //  console.log(x);

                let x = 3;

                resolve(x);

            } catch (error) {

                reject('失败' + error);

            }

        });

        // prm.then(function(d) {

        //     console.log(d);

        //     console.log(y);

        //     let y = 8;

        // }, function(err) {

        //     console.log(err, 777);

        // });

        prm.then(function(d) {

            console.log(d);

            console.log(y);

            let y = 10;

        }).catch(function(err) {

            console.log(err, 666);

        });

        //promise的then()调用完成后返回的是Promise对象

        // console.log(prm.then(), 999);

    </script>