1. ECMA第一章：

1.1. let变量声明及声明特性

1.1.1. 声明变量

```
// 声明变量
let a;
let b,c,d;
let e = 100;
let f = 521,g = 'one',h = [];
```

1.1.2. 1.let变量不能重复声明

```
// let star = '测试';
// let star = '123';
```

1.1.3. 2.块级作用域，全局，函数，eval

```

// 块级作用域，全局，函数，eval
// if else while for

{
    let girl = '测试';
    // 块级外部，无法访问
    console.log(girl);
}
```

1.1.4. 3.不存在变量提升

```
let son = '儿子';
console.log(son);
```

1.1.5. 4.不影响作用域链

```

// 不影响作用域链
{
    let school = '金科';
    function fn(){
        console.log(school);
    }
    fn();
}
```

1.2. ES6案例实践：判断var为变量和let为变量的区别

```
    <style>
        .item{
            width: 100px;
            height: 50px;
            border: 1px solid black;
            display: inline-block;
        }
    </style>
</head>
<body>
    <div class="con">
        <h2>点击切换颜色</h2>
        <div class="item"></div>
        <div class="item"></div>
        <div class="item"></div>
    </div>
    <script>
        // 获取元素
        let itmes= document.querySelectorAll('.item');
```

```
        //  遍历并绑定事件
        for(let i = 0;i<itmes.length;i++){
            itmes[i].onclick = function(){
                // 修改当前背景颜色,变量为var, 指向为this
                // this.style.background = 'pink';
                // 变量为let
                itmes[i].style.background = 'pink';
            }
        }

    </script>
```

1.3. const声明常量以及特点

1.3.1. 声明常量，值不能修改的为常量

```
const SCHOOL = '金科';
```

1.3.2. 1.一定要赋初始值

```
const A;
```

1.3.3. 2.一般常量使用大写（默认规则）

```
const a = 100;
```

1.3.4. 3.常量的值不能修改

```
SCHOOL = '康博尔';
```

1.3.5. 4.块级作用域

```
{
    const PALYER = 'KEKE';
}
console.log(PALYER);
```

1.3.6. 5.对于数组和对象的元素修改，不算做对常量的修改，不会报错

```
const TEAM = ['a','b','c','d'];
TEAM.push('e');
console.log(TEAM);
```

1.4. 变量的结构赋值

1.4.1. 解构赋值：ES6 允许按照一定模式从数组和对象中提取值，对变量进行赋值

1.4.2. 1.数组的结构赋值

```
const F4 = ['赵云','典韦','司马懿','曹操'];
let [a,b,c,d] = F4;
console.log(a);
console.log(b);
console.log(c);
console.log(d);
```

1.4.3. 2.对象的结构赋值

```
const zhao = {
    name:'张三',
    age:29,
    xiaopin: function(){
        console.log('测试');
    }
}
let {name,age,xiaopin} = zhao;
console.log(name);
console.log(age);
console.log(xiaopin);
```

2. ECMA第二章：

2.1. 模板字符串

```
ES6引入新的声明字符串的方式 [``] '' ""
```

2.1.1. 1.声明

```
let str = `我是一个字符串`;
console.log(str,typeof str);
```

2.1.2. 2.内容中可以直接出现换行

```
let str = `
        <ul>
            <li>1</li>
            <li>2</li>
            <li>3</li>
        </ul>
`;
document.write(str);
```

2.1.3. 3.变量拼接

```
let star = '甲';
let out = `${star}是路人`;
console.log(out);
```

2.2. 对象的简化写法

ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法

```
<script>
    // ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法
    let name = '金科';
    let change = function(){
        console.log('这是一个函数');
    }

    const school = {
        name,
        change,
        // 方法的简写版本
        import(){
            console.log('常量对象中的函数简写版本');
        }
    }
    console.log(school);
</script>
```

2.3. 箭头函数以及声明特点：ES6允许使用 [箭头] (=>) 定义函数

2.3.1. 箭头函数的声明

```
<script>
    // 常规的函数声明和调用
    let fn = function(a,b){
        return a+b;
    }
    console.log(fn(1,2));           //3

    // 箭头函数的声明和调用
    let fn1 = (c,d) => {
        return c+d;
    };
    console.log(fn1(10,10));        //20
</script>
```

2.3.2. 箭头函数的特性

2.3.2.1. this是静态的，this始终指向函数声明时所在作用域下的this的值。

```
<!-- `this`是静态的，`this`始终指向函数声明时所在作用域下的`this`的值。 -->
<script>
    function getName(){
        console.log(this.name);
    }
    let getName2 = () => {
        console.log(this.name);
    }
    // 设置 window对象的 name 属性
    window.name = '金科教育';
    const SCHOOL = {
        name:'康博尔'
    }

    // 直接调用
```

```
        getName();      //金科教育
        getName2();     //金科教育

        // call方法调用，call方法：更改this的指向。
        getName.call(SCHOOL);       //康博尔
        getName2.call(SCHOOL);      //金科教育
    </script>
```

2.3.2.2. 箭头函数不能作为构造实例化对象

```
    <script>
        // 不能作为构造实例化对象
        let People = (name,age) =>{
            this.name = name;
            this.age = age;
        }
        let me =new People('小胡',20);
        console.log(me);    //// Uncaught TypeError: Person is not a constructor
    </script>
```

```
Uncaught TypeError: People is not a constructor
    at 03___箭头函数的特性2.html:16:17
```

2.3.2.3. 箭头函数中，不能使用`arguments`变量

```
    <script>
        // 箭头函数中，不能使用arguments变量
        let fn = () => {
            console.log(arguments);
        }
        fn(1,2,3);
    </script>
```

```
▶Uncaught ReferenceError: arguments is not defined
    at fn (04___箭头函数的特性3.html:12:25)
    at 04___箭头函数的特性3.html:14:9
```

2.3.3. 箭头函数的简写形式

2.3.3.1. 省略小括号，当形参有且只有一个的时候

```
    let add = (n) =>{
        return n+n;
    }
    // 1.省略小括号，当形参有且只有一个的时候
    let add1 = n =>{
        return n+n;
    }
    console.log(add(10));       //20
    console.log(add1(9));       //18
```

2.3.3.2. 省略花括号，当代码体只有一条语句的时候，此时 reutrn 必须省略，而且语句的执行结果就是函数的返回值。

```
    let add2 = n =>{
        return n*n;
    }
    // 2.省略花括号，当代码体只有一条语句的时候，此时 reutrn 必须省略，而且语句的执行结果就是函数的返回值。
    let add3 = n => n*n;

    console.log(add2(8));       //64
    console.log(add3(8));       //64
```

2.3.4. 箭头函数的实践与应用场景

2.3.4.1. 案例1：点击div 2s后颜色变为 红色

```
    <style>
        #ad{
            width: 200px;
            height: 200px;
            background-color: aqua;
        }
    </style>
</head>
<body>
    <div id="ad"></div>

    <script>
```

```
        let ad = document.getElementById('ad');
        /* ad.addEventListener('click',function(){
            // this.style.background = 'red';
            setTimeout(function(){
                // console.log(123);
                console.log(this);                  //而是指向window

                // this.style.background = 'red';      //报错,this指向不到本身，而是指向window
            },2000)
        }) */

        ad.addEventListener('click',function(){
            // this.style.background = 'red';
            setTimeout( () =>{
                console.log(this);          //指向本身的元素
                this.style.background = 'red';
            },2000)
        })
    </script>
```

2.3.4.2. 从数组中返回偶数的元素

```
    <!-- 从数组中返回偶数的元素 -->
    <script>
        const ARR = [1,6,9,10,100,25]
        // 原js求偶数的方法
        // const RESULT = ARR.filter(function(item){
        //     if(item % 2 === 0){
        //         return true;
        //     }else{
        //         return false;
        //     }
        // })
        // console.log(RESULT);


        // 箭头函数精简版
        // const RESULT = ARR.filter(item =>{
        //     if(item % 2 ===0){
        //         return true
        //     }else{
        //         return false
        //     }
        // })
```

```
        // console.log(RESULT);

        // 极简版
        const RESULT = ARR.filter(item => item % 2 === 0);
        console.log(RESULT);
    </script>
```

2.3.5. 箭头函数总结：

1. 箭头函数**适合**与 `this` 无关的回调，定时器，数组的方法回调
2. 箭头函数**不适合**与 `this` 有关的回调，事件回调，对象的方法。

2.4. 函数参数的默认值设置

2.4.1. ES6 允许给函数参数赋值初始值

```
    <script>
        // ES6 允许给函数参数赋值初始值
        // 1.形参初始值，具有默认值的参数，一般位置要靠后（默认规则）
        function add(a,b,){
            return a+b+c;
        }
        console.log(add(1,2));
    </script>
```

2.4.2. 通过ES6的结构赋值相结合，可以直接访问对应的值

```
    <script>
        // 与结构赋值结合

        // function connect(options){
        //     // 使用js原生结构赋值太过于繁琐，每次都要写option
```

```
        //     // let host = options.host;
        //     // let username = options.username;
        // }

        // 通过ES6的结构赋值相结合，可以直接访问对应的值
            // 也可以当对应属性没有值的情况下，访问定值
        function connect({host='127.0.0.1',username,password,port}){
            console.log(host,username,password,port);
        }

        connect({
            host:'localhost',
            username:'root',
            password:'root',
            port:3306
        })
    </script>
```

2.5. ES6中的`rest`参数

ES6 引入 `rest` 参数，用于获取函数的实参，用来代替 `arguments`

```
    // ES5 获取实参的方式
    function date(){
        console.log(arguments);
    }
    date('甲','乙','丙');
```

2.5.1. `rest` 参数

```
    function date(...args){
        console.log(args);
    }
    date('甲','乙','丙');
```

2.5.2. rest 参数必须要放到参数最后

```
// rest 参数必须要放到参数最后
function fn(a,b,...args){
    console.log(a);
    console.log(b);
    console.log(args);
}
fn(1,2,3,4,5,6);
```

2.6. ES6扩展运算符

2.6.1. 扩展运算符介绍

[...] 扩展运算符能将[数组]转换为逗号分隔的[参数序列]

```
<script>
    // [...] 扩展运算符能将[数组]转换为逗号分隔的[参数序列]

    // 声明一个数组
    const STAR = ['路人甲','乙','丙'];      // ... 可以将数组改为 '路人甲','乙','丙'

    // 声明一个函数
    function show(){
        console.log(arguments);
    }
    show(...STAR);
</script>
```

2.6.2. 扩展运算符的应用

```
    <script>
        // 1.数组的合并
        const ARR1 = [1,2,3,4];
        const ARR2 = [0,0,0,0];
            // js,使用concat()方法
        const ARRSUM = ARR1.concat(ARR2);
        console.log(ARRSUM);
            //  ES6,扩展运算符
        const ARRES6SUN = [...ARR1,...ARR2]
        console.log(ARRES6SUN);
    </script>
```

```
    <!--  扩展运算符的应用 -->
    <script>
        // 2.数组的克隆
        const ARRZERO = ['HTML','CSS','JS'];
        const ARRZEROCLONE = [...ARRZERO];
        console.log(ARRZEROCLONE);
    </script>
```

```
    <!--  扩展运算符的应用 -->
    <div></div>
    <div></div>
    <div></div>

    <script>
        const divs = document.querySelectorAll('div');
        console.log(divs);      //  伪数组
        const DIVARR = [...divs];
        console.log(DIVARR);    // 数组类型的参数序列
    </script>
```

2.7. Symbol创建于使用

ES6引入了一种新的原始数据类型 `Symbol`，表示独一无二的值。它是 `JavaScript` 语言的第七种数据类型，是一种类似于字符串的数据类型。

- `Symbol` 的值是惟一的，用来解决命名冲突的问题。
- `Symbol` 值不能与其他数据进行运算
- `Symbol` 定义的对象属性不能使用 `for...in` 遍历循环，但是可以使用 `Reflect.ownKeys` 来获取对象的所有键名

2.7.1. Symbol的创建

```
<!-- Symbol的基本使用 -->
<script>
    // 创建 Symbol
    let s = Symbol();
    console.log(s,typeof s);        //Symbol() 'symbol'

    // Symbol值是惟一的
    let s2 = Symbol('测试');
    let s3 = Symbol('测试');
    console.log(s2 === s3);         //false

    // Symbol.for 创建 ,两个的值是一样的
    let s4 = Symbol.for('金科教育');
    let s5 = Symbol.for('金科教育');
    console.log(s4 === s5);         //true

</script>
```

2.7.2. Symbol 不能与其他数据进行运算

```
<!-- Symbol 不能与其他数据进行运算 -->
<script>
    let s = Symbol();
    // let result = s + 100;        //报错
    // let result = s > 100;        //报错
    let reslut = s + s;             //报错
    console.log(result);
</script>
```

2.7.3. JavaScript的其他数据类型

```
<script>
    // undefined
    // String
    // Object
    // Number
    // Boolean
</script>
```

2.7.4. Symbol的使用