link: null
title: 珠峰架构师成长计划
description: ECMAScript简称就是ES,你可以把它看成是一套标准,JavaScript就是实施了这套标准的一门语言,现在主流浏览器使用的是ECMAScript5。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=144 sentences=225, words=1530

ECMAScript6

ECMAScript简称就是ES,你可以把它看成是一套标准,JavaScript就是实施了这套标准的一门语言,现在主流浏览器使用的是ECMAScript5。

1. 作用域变量

作用域就是一个变量的作用范围。也就是你声明一个变量以后,这个变量可以在什么场合下使用 以前的 *JavaScript*只有全局作用域，还有一个函数作用域

在用var定义变量的时候，变量是通过闭包进行隔离的，现在用了let，不仅仅可以通过闭包隔离，还增加了一些块级作用域隔离。 块级作用用一组大括号定义一个块,使用 let 定义的变量在大括号的外面是访问不到的

```
if(true){
    let name = 'zfpx';
}
console.log(name);
```

```
if(true){
    let name = 'zfpx';
}
console.log(window.name);
```

> 结果 undefined

```
    for (let i = 0; i < 3; i++) {
        console.log("out", i);
        for (let i = 0; i < 2; i++) {
            console.log("in", i);
        }
    }
```

> 结果 out 0 in 0 in 1 out 1 in 0 in 1 out 2 in 0 in 1

```
if(true){
    let a = 1;
    let a = 2;
}
```

```
for (let i = 0; i < 2; i ++){
    console.log('inner',i);
    let i = 100;
}
```

> 结果 **i is not defined**

```
;(function () {

})();
```

```
{
}
```

2. 常量

使用 const我们可以去声明一个常量，常量一旦赋值就不能再修改了

```
const MY_NAME = 'zfpx';
MY_NAME = 'zfpx2';
```

> 注意 const限制的是不能给变量重新赋值，而变量的值本身是可以改变的,下面的操作是可以的

```
const names = ['zfpx1'];
names.push('zfpx2');
console.log(names);
```

```
const A = "0";
{
    const A = "A";
    console.log(A)
}
{
    const A = "B";
    console.log(A)
}
console.log(A)
```

> 结果 A B 0

3. 解构

解构意思就是分解一个东西的结构,可以用一种类似数组的方式定义N个变量，可以将一个数组中的值按照规则赋值过去。

```
var [name,age] = ['zfpx',8];
console.log(name,age);
```

```
    let [x, [y], z] = [1, [2.1, 2.2]];
    console.log(x, y, z);

    let [x, [y,z]] = [1, [2.1, 2.2]];
    console.log(x,y,z);

    let [json,arr,num] = [{name:'zfpx'},[1,2],3];
    console.log(json,arr,num);
```

```
1 2.1 undefined 1 2.1 2.2 { name: 'zfpx' } [ 1, 2 ] 3
```

```
let [, , x] = [1, 2, 3];
console.log(x);
```

对象也可以被解构

```
var obj = {name:'zfpx',age:8};

var {name,age} = obj;

let {name: myname, age: myage} = obj;
console.log(name,age,myname,myage);
```

在赋值和传参的时候可以使用默认值

```
let [a = "a", b = "b", c =new Error('C必须指定')] = [1, , 3];
console.log(a, b, c);

function ajax (options) {
    var method = options.method || "get";
    var data = options.data || {};

}
function ajax ({method = "get", data}) {
    console.log(arguments);
}
ajax({
    method: "post",
    data: {"name": "zfpx"}
});
```

4. 字符串

模板字符串用反引号(数字1左边的那个键)包含，其中的变量用 `${}`括起来

```
var name = 'zfpx',age = 8;
let desc = `${name} is ${age} old!`;
console.log(desc);

var str = `
a
b
`;
console.log(str);
```

```
其中的变量会用变量的值替换掉
```

```
function replace(desc){
  return desc.replace(/\$\{([^}]+)\}/g,function(matched,key){
    return eval(key);
  });
}
```

可以在模板字符串的前面添加一个标签，这个标签可以去处理模板字符串 标签其实就是一个函数,函数可以接收两个参数,一个是 `strings`,就是模板字符串里的每个部分的字符 还有一个参数可以使用 `rest`的形式 `values`,这个参数里面是模板字符串里的值

```
var name = 'zfpx',age = 8;
function desc(strings,...values){
    console.log(strings,values);
}
desc`${name} is ${age} old!`;
```

- includes()：返回布尔值，表示是否找到了参数字符串。
- startsWith()：返回布尔值，表示参数字符串是否在源字符串的头部。
- endsWith()：返回布尔值，表示参数字符串是否在源字符串的尾部。

```
var s = 'zfpx';
s.startsWith('z')
s.endsWith('x')
s.includes('p')
```

第二个参数，表示开始搜索的位置

```
var s = 'zfpx';
console.log(s.startsWith('p',2));
console.log(s.endsWith('f',2));
console.log(s.includes('f',2));
```

```
endsWith的行为与其他两个方法有所不同。它针对前n个字符，而其他两个方法针对从第n个位置直到字符串结束
```

repeat方法返回一个新字符串，表示将原字符串重复n次。

```
'x'.repeat(3);
'x'.repeat(0);
```

5. 函数

可以给定义的函数接收的参数设置默认的值 在执行这个函数的时候，如果不指定函数的参数的值，就会使用参数的这些默认的值

```
function ajax(url,method='GET',dataType="json"){
  console.log(url);
  console.log(method);
  console.log(dataType);
}
```

把...放在数组前面可以把一个数组进行展开,可以把一个数组直接传入一个函数而不需要使用 apply

```
let print = function(a,b,c){
    console.log(a,b,c);
}
print([1,2,3]);
print(...[1,2,3]);

var m1 = Math.max.apply(null, [8, 9, 4, 1]);
var m2 = Math.max(...[8, 9, 4, 1]);

var arr1 = [1, 3];
var arr2 = [3, 5];
var arr3 = arr1.concat(arr2);
var arr4 = [...arr1, ...arr2];
console.log(arr3,arr4);

function max(a,b,c) {
    console.log(Math.max(...arguments));
}
max(1, 3, 4);
```

剩余操作符可以把其余的参数的值都放到一个叫 b的数组里面

```
let rest = function(a,...rest){
    console.log(a,rest);
}
rest(1,2,3);
```

```
let destruct = function({name,age}){
    console.log(name,age);
}
destruct({name:'zfpx',age:6});
```

ECMAScript 6 给函数添加了一个 name属性

```
var desc = function descname(){}
console.log(desc.name);
```

箭头函数简化了函数的的定义方式，一般以 "=>" 操作符左边为输入的参数，而右边则是进行的操作以及返回的值 inputs=>output

```
[1,2,3].forEach(val => console.log(val)););
```

> 输入参数如果多于一个要用()包起来，函数体如果有多条语句需要用{}包起来

箭头函数根本没有自己的this，导致内部的this就是外层代码块的this。 正是因为它没有this，从而避免了this指向的问题。

```
var person = {
    name:'zfpx',
    getName:function(){
-       setTimeout(function(){console.log(this);},1000); //在浏览器执行的话this指向window
+       setTimeout(() => console.log(this),1000);//在浏览器执行的话this指向person
    }
}
person.getName();
```

将一个数组或者类数组变成数组,会复制一份

```
let newArr = Array.from(oldArr);
```

of是为了将一组数值,转换为数组

```
console.log(Array(3), Array(3).length);
console.log(Array.of(3), Array.of(3).length);
```

Array.prototype.copyWithin(target, start = 0, end = this.length) 覆盖目标的下标 开始的下标 结束的后一个的下标

```
[1, 2, 3, 4, 5].copyWithin(0, 1, 2);
```

查到对应的元素和索引

```
let arr = [1, 2 ,3, 3, 4, 5];
    let find = arr.find((item, index, arr) => {
        return item === 3;
    });
    let findIndex = arr.findIndex((item, index, arr) => {
        return item === 3;
    });

    console.log(find, findIndex);
```

就是填充数组的意思 会更改原数组 Array.prototype.fill(value, start, end = this.length);

```
let arr = [1, 2, 3, 4, 5, 6];
arr.fill('a', 1, 2);
console.log(arr);
```

6. 对象

如果你想在对象里添加跟变量名一样的属性，并且属性的值就是变量表示的值就可以直接在对象里加上这些属性

```
let name = 'zfpx';
let age = 8;
let getName = function(){
    console.log(this.name);
}
let person = {
    name,
    age,
    getName
}
person.getName();
```

对比两个值是否相等

```
console.log(Object.is(NaN,NaN));
```

把多个对象的属性复制到一个对象中,第一个参数是复制的对象,从第二个参数开始往后,都是复制的源对象

```
var nameObj = {name:'zfpx'};
var ageObj = {age:8};
var obj = {};
Object.assign(obj,nameObj,ageObj);
console.log(obj);

function clone (obj) {
  return Object.assign({}, obj);
}
```

将一个指定的对象的原型设置为另一个对象或者null

```
var obj1  = {name:'zfpx1'};
var obj2 =  {name:'zfpx2'};
var obj = {};
Object.setPrototypeOf(obj,obj1);
console.log(obj.name);
console.log(Object.getPrototypeOf(obj));
Object.setPrototypeOf(obj,obj2);
console.log(obj.name);
console.log(Object.getPrototypeOf(obj));
```

直接在对象表达式中设置prototype

```
var obj1  = {name:'zfpx1'};
var obj3 = {
    __proto__:obj1
}
console.log(obj3.name);
console.log(Object.getPrototypeOf(obj3));
```

通过super可以调用prototype上的属性或方法

```
let person ={
    eat(){
        return 'milk';
    }
}
let student = {
    __proto__:person,
    eat(){
        return super.eat()+' bread'
    }
}
console.log(student.eat());
```

7. 类

使用 class这个关键词定义一个类,基于这个类创建实例以后会自动执行 constructor方法,此方法可以用来初始化

```
class Person {
    constructor(name){
        this.name = name;
    }
    getName(){
        console.log(this.name);
    }
}
let person = new Person('zfpx');
person.getName();
```

getter可以用来得获取属性， setter可以去设置属性

```
class Person {
    constructor(){
        this.hobbies = [];
    }
    set hobby(hobby){
        this.hobbies.push(hobby);
    }
    get hobby(){
        return this.hobbies;
    }
}
let person = new Person();
person.hobby = 'basketball';
person.hobby = 'football';
console.log(person.hobby);
```

在类里面添加静态的方法可以使用 static这个关键词，静态方法就是不需要实例化类就能使用的方法

```
class Person {
   static add(a,b){
       return a+b;
   }
}
console.log(Person.add(1,2));
```

一个类可以去继承其它的类里的东西

```
class Person {
   constructor(name){
     this.name = name;
   }
}
class Teacher extends Person{
    constructor(name,age){
        super(name);
        this.age = age;
    }
}
var teacher = new Teacher('zfpx',8);
console.log(teacher.name,teacher.age);
```

8.生成器(Generator)与迭代器(Iterator)

Generator是一个特殊的函数，执行它会返回一个Iterator对象。 通过遍历迭代器， Generator函数运行后会返回一个遍历器对象，而不是普通函数的返回值。

迭代器有一个next方法，每次执行的时候会返回一个对象 对象里面有两个属性，一个是 value表示返回的值，还有就是布尔值 done,表示是否迭代完成

```
function buy(books) {
    let i = 0;
    return {
        next(){
            let done = i == books.length;
            let value = !done ? books[i++] : undefined;
            return {
                value: value,
                done: done
            }
        }
    }
}

let iterators = buy(['js', 'html']);
var curr;
do {
    curr = iterators.next();
    console.log(curr);
} while (!curr.done);
```

生成器用于创建迭代器

```
function* buy(books){
    for(var i=0;iyield books[i];
    }
}
let buying = buy(['js','html']);
var curr;
do {
    curr = buying.next();
    console.log(curr);
} while (!curr.done);
```

9.集合

一个 Set是一堆东西的集合, Set有点像数组,不过跟数组不一样的是， Set里面不能有重复的内容

```
var books = new Set();
books.add('js');
books.add('js');
books.add('html');
books.forEach(function(book){
    console.log(book);
})
console.log(books.size);
console.log(books.has('js'));
books.delete('js');
console.log(books.size);
console.log(books.has('js'));
books.clear();
console.log(books.size);
```

可以使用 Map 来组织这种名值对的数据

```
var books = new Map();
books.set('js',{name:'js'});
books.set('html',{name:'html'});
console.log(books.size);
console.log(books.get('js'));
books.delete('js');
console.log(books.has('js'));
books.forEach((value, key) => {
    console.log( key + ' = ' + value);
});
books.clear();
```

10.模块

可以根据应用的需求把代码分成不同的模块 每个模块里可以导出它需要让其它模块使用的东西 在其它模块里面可以导入这些模块导出的东西

在浏览器中使用模块需要借助 导出

```
export var name = 'zfpx';
export var age = 8;
```

```
import * as school from './school.js';
console.log(school.name,school.age);
```

在页面中引用

```

```

导出时重命名

```
function say(){
    console.log('say');
}
export {say as say2};
```

导入时重命名

```
import {say2 as say3} from './school.js';
```

每个模块都可以有一个默认要导出的东西 导出

```
export default function say(){
    console.log('say');
}
```

```
import say from './school.js';
```

```
var parent = {
  age: 5,
  hobby: [1, 2, 3],
  home: {city: '&#x5317;&#x4EAC;'},
};

var child = extendDeep(parent);
child.age = 6;
child.hobby.push('4');
child.home.city = '&#x5E7F;&#x4E1C;';
console.log('child ', child); //[1, 2, 3, 4]
console.log('parent ', parent);
function extend(parent) {
  let child;
  if (Object.prototype.toString.call(parent) == '[object Object]') {
    child = {};
    for (let key in parent) {
      child[key] = extend(parent[key])
    }
  } else if (Object.prototype.toString.call(parent) == '[object Array]') {
    child = parent.map(item => extend(item));
  } else {
    return parent;
  }
  return child;
}

function extendDeep(parent, child) {
  child = child || {};
  for (var key in parent) {
    if (typeof parent[key] === "object") {
      child[key] = (Object.prototype.toString.call(parent[key]) === "[object Array]") ? [] : {};
      extendDeep(parent[key], child[key]);
    } else {
      child[key] = parent[key];
    }
  }
  return child;
}
```

作业