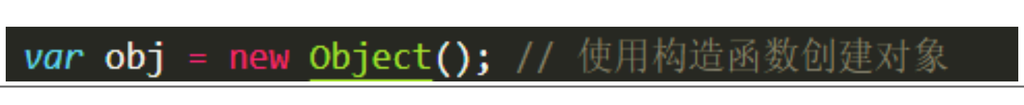
day18

1.对象创建

- 使用构造函数创建对象



```
var obj = new Object(); // 使用构造函数创建对象
```

- 使用构造函数语法糖

```
var obj2 = {}; // 构造函数的语法糖
```

- 概念
  - 1.构造函数 用于创建对象
  - 2.使用new关键字 跟上构造函数的名称进行对象创建
  - 3.构造函数(类)采用的 都是 大驼峰命名法
  - 可以使用构造函数的创建对象
    - Object
    - Array
    - Date
    - String
    - Boolean
    - Number
    - RegExp
    - Set Map
  - 注意：Math虽然不能使用构造函数，但是其还是对象

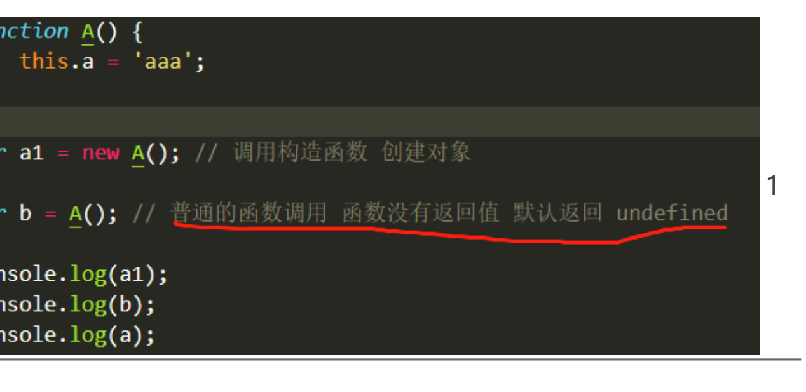
2.工厂函数

- 工厂函数的作用是生产对象 所以也采用大驼峰命名法
- 使用this，而是在函数中，创建一个对象，然后将对象设置属性

```
function Phone(cpu, ram, rom, scn) {
    var phone = new Object(); // 每一件产品都是独立的新对象
    //将原料放入产品中
    phone.cpu = cpu;
    phone.ram = ram;
    phone.rom = rom;
    phone.scn = scn;
    return phone;
}

var iphoneX1 = Phone('A12', '128G', '8G', '三星');
console.log(iphoneX1);

var iphoneX2 = Phone('A12', '128G', '8G', '三星');
console.log(iphoneX2);

var iphoneX3 = Phone('A12', '128G', '8G', '三星');
console.log(iphoneX3);

var iphoneX4 = Phone('A12', '128G', '8G', '三星');
console.log(iphoneX4);

console.log(iphoneX1 === iphoneX2);//false
```

3.构造函数

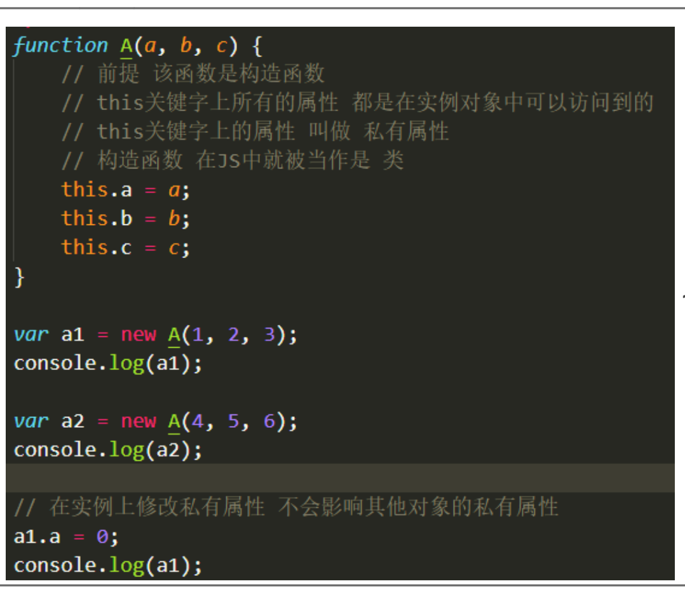
- 概念　在JS中 构造函数指的是用于创建对象的函数
- 如何判断函数是否是构造函数　看调用方式
  - 使用关键字new调用 它就是构造函数
  - 不使用关键字new调用 它就是普通函数
- 构造函数中的this
  - 在构造函数中 所创建对象的所有的属性 需要添加到this关键字上
  - 在构造函数中 this关键字 指向的是就是新创建的对象
- 构造函数的原理
  - 在内存空间中开辟一块存储空间
  - 存入新创建的对象 并返回这个对象所在的 地址
- this特殊使用
  - 当一个函数被当作构造函数使用时 函数内的this关键字 就会指向 新创建的对象
  - 如果不是当作构造函数使用 this 指向调用者
  - 如果没有调用者 则指向 window



```
function A() {
    this.a = 'aaa';
}

var a1 = new A(); // 调用构造函数 创建对象

var b = A(); // 普通的函数调用 函数没有返回值 默认返回 undefined

console.log(a1);
console.log(b);
console.log(a);
```

4.构造函数2

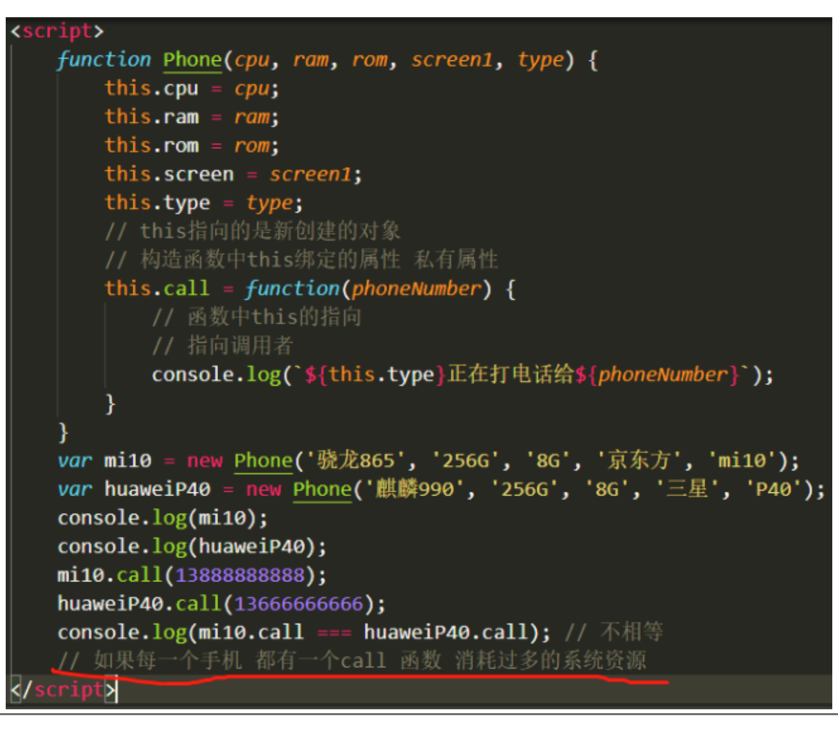
- 带参构造函数的创建
  - 前提：该函数是构造函数
  - this关键字上所有的属性 都是在实例对象中可以访问到的
  - this关键字上的属性 叫做 私有属性
  - 构造函数 在JS中就被当作是 类
- 注意　在实例上修改私有属性 不会影响其他对象的私有属性



```
function A(a, b, c) {
    // 前提 该函数是构造函数
    // this关键字上所有的属性 都是在实例对象中可以访问到的
    // this关键字上的属性 叫做 私有属性
    // 构造函数 在JS中就被当作是 类
    this.a = a;
    this.b = b;
    this.c = c;
}

var a1 = new A(1, 2, 3);
console.log(a1);

var a2 = new A(4, 5, 6);
console.log(a2);

// 在实例上修改私有属性 不会影响其他对象的私有属性
a1.a = 0;
console.log(a1);
```

5.构造函数3

- 私有方法
  - this指向的是新创建的对象
  - 构造函数中this绑定的属性 私有属性
  - 函数中this的指向，指向调用者



```
<script>
    function Phone(cpu, ram, rom, screen1, type) {
        this.cpu = cpu;
        this.ram = ram;
        this.rom = rom;
        this.screen = screen1;
        this.type = type;
        // this指向的是新创建的对象
        // 构造函数中this绑定的属性 私有属性
        this.call = function(phoneNumber) {
            // 函数中this的指向
            // 指向调用者
            console.log(`${this.type}正在打电话给${phoneNumber}`);
        }
    }
    var mi10 = new Phone('骁龙865', '256G', '8G', '京东方', 'mi10');
    var huaweiP40 = new Phone('麒麟990', '256G', '8G', '三星', 'P40');
    console.log(mi10);
    console.log(huaweiP40);
    mi10.call(13888888888);
    huaweiP40.call(13666666666);
    console.log(mi10.call === huaweiP40.call); // 不相等
    // 如果每一个手机 都有一个call 函数 消耗过多的系统资源
</script>
```

6.公有属性

- 原型对象
  - 原型是函数的一个属性
  - 所有函数 都拥有这个属性 prototype
  - 所有的prototype属性中都拥有一个 constructor 属性
  - constructor 在这里是一个指针 它指向了 该原型对象的父级函数
- prototype
  - prototype 用来保存 构造函数 实例的 公有属性
  - 如果不被当作构造函数使用 它就没有意义
  - prototype 中的所有成员 都可以被 实例对象访问
- 使用对象.prototype.方法名　创建公有属性方法

```
Phone.prototype.call = function(phoneNumber) {
    console.log(`${this.type}正在打电话给${phoneNumber}`);
}

Phone.prototype.play = function(gameName) {
    console.log(`正在使用${this.type}玩${gameName}`);
}

var mi10 = new Phone('骁龙865', '256G', '8G', '京东方', 'mi10');
var huaweiP40 = new Phone('麒麟990', '256G', '8G', '三星', 'P40');

mi10.call(13666666666);
huaweiP40.call(13666688866);
```

7.深入原型

- prototype 原型
  - 用于保存 构造函数 实例对象的公有属性
  - 原型下所有的函数和属性都可以在实例对象(构造函数创建的对象)中被访问
- 对象的创建
  - new 关键字
  - 构造函数
- JS中的对象 可以访问到原型中的属性和方法
  - JS中的每一个对象 都拥有一个 指针 __proto__
  - __proto__ 指向的是这个对象的构造函数的原型
- 对象是如何查找属性的
  - 1. 首先找私有属性，如果没有找到，则转到2
  - 2. 向上查找公有属性，如果没有找到，则转到3
  - 3. 继续沿原型链向上查找 直到 Object.prototype中都没有，找到原型链中的null， 返回undefined
- 特殊
  - JS中所有的原型对象 都是由 Object 创建的
  - 所有的函数 都是由 Function 创建的
  - 而Function 是由自己创建的

8.toString

- Object.prototype.toString()　自定义对象的toString方法是通过原型链的查找获得的

```
function Foo() {
    // this.toString = function() {
    //     return 'abc';
    // }
}
Foo.prototype.toString = function() {
    return 'hello world';
}
var foo = new Foo();
console.log(foo.toString()); // 自定义对象的toString方法是通过原型链的查找获得的
console.log(foo.__proto__.__proto__ === Object.prototype); //返回true//Object.prototype
```

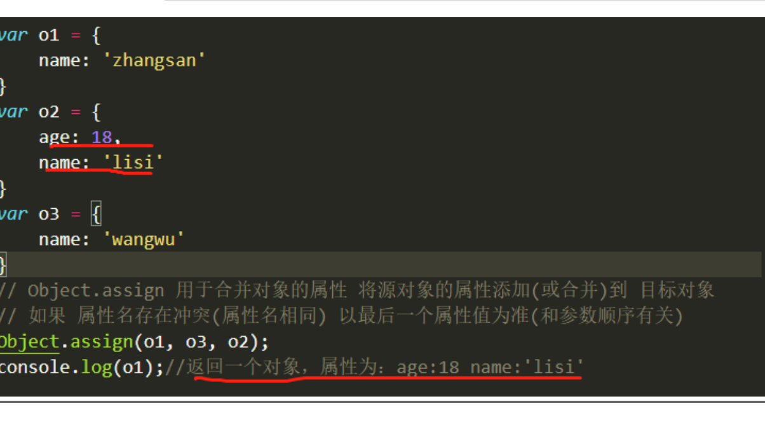
- 函数重载
  - 将功能相同的或相近的函数 设置成相同的函数名
  - 参数的类型 或 数量不同 来区分不同的函数
- 判断数据类型

```
// 判断数据类型
function getClass(o) {
    return Object.prototype.toString.call(o).slice(8, -1);
}

console.log(getClass(123));//Number
console.log(getClass('abc'));//String
console.log(getClass(null));//Null
console.log(getClass(undefined));//Undefined
console.log(getClass({}));//Object
console.log(getClass([]));//Array
```

- 注意，数组有自己的toString（）方法，不继承Object的

9.Object.assign

- 函数使用
  - 方法用于将所有可枚举属性的值从一个或多个源对象复制到目标对象。它将返回目标对象。
  - 语法: Object.assign(target, ...sources)
  - // 返回值: target 目标对象
  - 使用注意
    - Object.assign 用于合并对象的属性 将源对象的属性添加(或合并)到 目标对象
    - 如果 属性名存在冲突(属性名相同) 以最后一个属性值为准(和参数顺序有关)



```
var o1 = {
    name: 'zhangsan'
}
var o2 = {
    age: 18,
    name: 'lisi'
}
var o3 = {
    name: 'wangwu'
}
// Object.assign 用于合并对象的属性 将源对象的属性添加(或合并)到 目标对象
// 如果 属性名存在冲突(属性名相同) 以最后一个属性值为准(和参数顺序有关)
Object.assign(o1, o3, o2);
console.log(o1);//返回一个对象，属性为：age:18 name:'lisi'
```

- 应用场景　参数合并（为函数提供默认参数）

10.Object.create

- 函数使用
  - 方法创建一个新对象，使用现有的对象来提供新创建的对象的__proto__。
  - 创建一个新对象，使用一个现有对象作为新对象的原型
- 作用
  - 克隆对象
  - 实现继承
  - 新对象可以访问到现有对象的属性(对象都可以访问到原型中的属性)
- 通俗描述
  - 基于原型
  - new 创建对象
  - 你创建的每一个对象 都可以访问原型对象
  - 你创建的每一个对象 都是拷贝了一次原型对象

```
var obj = {
    name: 'zhangsan',
    age: 20
};
var obj2 = Object.create(obj);
obj2.age = 25; // any obj2 添加 私有属性
console.log(obj2.age);//如果不重新赋值，那么age就是20
console.log(obj2);
```

11.Object.defineProperty()

- 函数的使用
  - 方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性，并返回此对象
  - 语法: Object.defineProperty(obj, prop, descriptor)
  - 参数
    - obj 需要定义属性的对象
    - prop 需要定义或修改的属性名 可以是字符串 也可以是Symbol
    - descriptor 定义或修改属性的描述符

```
var obj = {
    name: 'zhangsan'
};

Object.defineProperty(obj, 'age', {//设置age属性
    configurable: true, // 属性可删除  默认false
    enumerable: true, // 属性可枚举  默认false
    writable: true, // 属性可赋值  默认 false
    value: 20
});
Object.defineProperty(obj, 'name', {
    enumerable: false
})
obj.age = 21;
for (let key in obj) {
    console.log(key, obj[key]);
}
```

12.class

- ES6 关键字 class，本质是函数 构造函数的语法糖
- 正常构造类

```
function Student(name, age) {
    this.name = name;
    this.age = age;
    Object.defineProperty(this, 'age', {//设置不可修改的属性
        value: age
    });
}

var s1 = new Student('zhangsan', 12);
console.log(s1);
```

- 语法糖构造

```
class Student {
    constructor(name, age) {
        // 私有属性
        this.name = name;
        this.age = age;
    }
    // 公有属性
    // 所有的公有属性依旧是在原型上
    // 通过实例访问
    sayName() {
        console.log(this.name);
    }
    // 静态属性通过类名访问
    static className = 'html5-2009';
}
var s2 = new Student('lisi', 15);
console.log(s2);
// console.log(typeof Student);
s2.sayName();

console.log(Student.className);
```

- 注意：公有属性依旧在原型上，通过实例访问　静态属性通过类名访问

13.class继承

- 使用extend继承父类所有公有属性
- 使用super继承父类所有私有属性　super必须在constructor开头的位置，和use strick一样

```
class F {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}
class S extends F { // extends 用于继承父类 继承所有的公有属性
    constructor(name, age) {
        // super实现的功能是从父类继承私有属性
        // super必须在constructor开头的位置
        super(name); // super是父类构造函数
        this.age = age;
    }
}
var s = new S('zhangsan', 18);
console.log(s);
// console.log(s.getName());
```

14.test

- 多参数、多情况的处理
- 函数重载

```
function fn() {
    switch (arguments.length) {
        case 1:
            console.log(arguments[0].split(''));
            break;
        case 2:
            console.log((arguments[0] + arguments[1]).split(''));
            break;
        case 3:
            // console.log(arguments[0].concat(arguments[1], arguments
            [2]));

            console.log(Array.from(arguments).join(''));
            break;
    }
}
```

扩展

- 支持for-of遍历的数据结构

原生具备 Iterator 接口的数据结构如下。

- Array
- Map
- Set
- String
- TypedArray
- 函数的 arguments 对象
- NodeList 对象

15.URLSearchParams

- 阿里面试题，具体看代码