面向对象编程

# echarts图表库\*\*\*\*\*\*

<https://www.echartsjs.com/zh/index.html> 官方手册

概述：echarts是前端工程师比较出名的JavaScript函数库，它最牛的一件事情，就是可以快速开发各种图标。

比如：折线图、柱状图、饼图、散点图等等。

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <!-- 通过双闭和标签script将echarts源码引入 -->

    <!-- 第一步：需要引包，将需要使用的源码引入 -->

    <script src="./js/echarts.min.js"></script>

</head>

<body>

      <!-- 第二步:显示图表的地方 -->

     <div id="main" style="width: 600px;height:400px;"></div>

</body>

</html>

<script type="text/javascript">

    //当引入echarts函数的时候，它对外保留一个对象通过init方法创建实例

    var myChart = echarts.init(document.getElementById('main'));

    // 指定图表的配置项和数据

   Var option = {

    title : {

        text: '南丁格尔玫瑰图',

        subtext: '纯属虚构',

        x:'center'

    },

    tooltip : {

        trigger: 'item',

        formatter: "{a} <br/>{b} : {c} ({d}%)"

    },

    legend: {

        x : 'center',

        y : 'bottom',

        data:['rose1','rose2','rose3','rose4','rose5','rose6','rose7','rose8']

    },

    toolbox: {

        show : true,

        feature : {

            mark : {show: true},

            dataView : {show: true, readOnly: false},

            magicType : {

                show: true,

                type: ['pie', 'funnel']

            },

            restore : {show: true},

            saveAsImage : {show: true}

        }

    },

    calculable : true,

    series : [

        {

            name:'半径模式',

            type:'pie',

            radius : [20, 110],

            center : ['25%', '50%'],

            roseType : 'radius',

            label: {

                normal: {

                    show: false

                },

                emphasis: {

                    show: true

                }

            },

            lableLine: {

                normal: {

                    show: false

                },

                emphasis: {

                    show: true

                }

            },

            data:[

                {value:10, name:'rose1'},

                {value:5, name:'rose2'},

                {value:15, name:'rose3'},

                {value:25, name:'rose4'},

                {value:20, name:'rose5'},

                {value:35, name:'rose6'},

                {value:30, name:'rose7'},

                {value:40, name:'rose8'}

            ]

        },

        {

            name:'面积模式',

            type:'pie',

            radius : [30, 110],

            center : ['75%', '50%'],

            roseType : 'area',

            data:[

                {value:10, name:'rose1'},

                {value:5, name:'rose2'},

                {value:15, name:'rose3'},

                {value:25, name:'rose4'},

                {value:20, name:'rose5'},

                {value:35, name:'rose6'},

                {value:30, name:'rose7'},

                {value:40, name:'rose8'}

            ]

        }

    ]

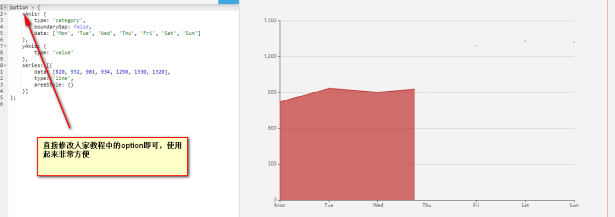
};

    // 使用刚指定的配置项和数据显示图表。

    myChart.setOption(option);

</script>

* ECHARTS函数库：搞图标开发
* 在实际工作的时候用到概率比较大的【前端工程师】，在使用的时候如下图



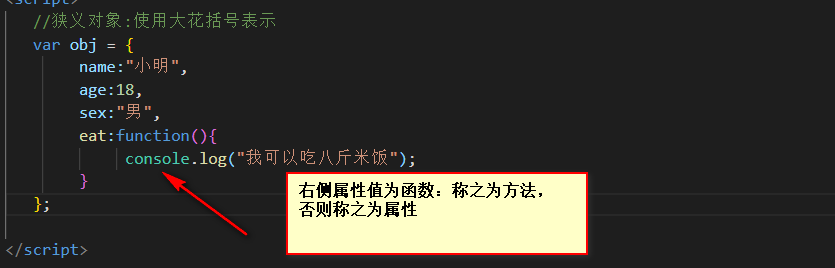
# 对象

概述：前端工程师中也有面向对象编程概念(基于面向对象编程),在JS当中对象分为两种：

* 第一种：狭义对象
* 第二种：广义对象

## 2.1狭义对象

概述：在JS当中狭义对象用一个大花括号表示{}；狭义对象可以拥有很多属性和方法；



<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //狭义对象:使用大花括号表示

   var obj = {

        name:"小明",

        age:18,

        sex:"男",

        eat:function(){

             console.log("我可以吃八斤米饭");

        }

   };

 //检测当前对象类型

 console.log(typeof obj);

 //对应获取属性值

</script>

* 右侧属性值为函数：称之为对象方法，否则为属性

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //狭义对象:使用大花括号表示

   var obj = {

        name:"小明",

        age:18,

        sex:"男",

        eat:function(){

             console.log("我可以吃八斤米饭");

        }

   };

 //检测当前对象类型

 console.log(typeof obj);

 //获取属性值

 console.log(obj.name);

 console.log(obj.age);

 console.log(obj.sex);

 console.log(obj.eat);

 //重新修改属性值

 obj.name = "二哈";

 obj.age = 2;

 obj.sex = "公";

 //动态添加属性、方法

 obj.color = "白加黑";

 obj.yaoren = function(){

      console.log("我可以咬人");

 }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//同学问的问题（相忘于江湖）

 obj.eat();

 obj.eat();

 obj.eat();

 obj.yaoren();

</script>

* 在JS当中大花括号即为狭义对象
* 狭义对象可以拥有属性、方法（区分开：属性值为函数称之为方法，否则称之为属性）

记住：JS当中对象才可以通过点语法动态添加属性和方法

## 2.2广义对象

概述：如果是引用类型数据，且系统提供内置属性和方法的对象，称之为广义对象；

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

  //广义对象-----函数

  function People (a,b,c){

  }

  //JS当中：五个基本数据类型：数字、字符串、布尔、未定义、空对象

  //引用类型数据:Object

  //函数即为广义对象：因为有一些系统内置属性、方法提供给我们使用

  //name:系统提供内置属性，可以获取函数名字

  console.log(People.name);

  //length：系统提供内置属性，可以获取函数形参个数

  console.log(People.length);

  //动态添加自定义属性

  People.xingming = "我是小明";

  People.age = 18;

  People.sex = "男";

  console.log(People.xingming);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/People

//广义对象------数组

var arr = ["吃饭",'睡觉','打豆豆'];

console.log(arr.length);

console.log(arr.reverse());

console.log(arr.join());

//动态添加属性和方法

arr.xingming ="小红";

arr.age = 18;

arr.sex = "女";

console.log(arr.xingming);

</script>

在JS当中对象【狭义、广义】：可以通过点语法动态添加自定义属性、方法

狭义对象：其实狭义对象即为大花括号表示

广义对象对象:广义对象除了自定义属性、方法意外，系统给它提供了内置属性和方法使用；

大总结：其实在JS当中只要是引用类型数据即为对象

## 2.3什么不是对象

概述：在JS当中基本数据类型数值都不是对象：（如果不是对象不能通过点语法动态添加属性和方法）

比如：数字、字符串、布尔值、未定义、空对象

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //NaN不是数据类型，它只是Number（数字类型中的一个特殊值）

   console.log(5);

   console.log(NaN);

   console.log(typeof 5);

   console.log(typeof NaN);

   //在那些特定情况下回出现这个特殊值NaN【在进行数学运算符且算不出结果的时候返回数字NaN】

   console.log(0/0)

</script>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //JS当用基本数据类型数值都不是对象【数字、字符串、布尔值、未定义、空对象】

   var num = 100;

   num.xingming ="我爱你";

   console.log(num.xingming);

   var str = "今天是个好日子";

   str.xingming = "毛爷爷";

   console.log(str.xingming);

</script>

总结：在JS当中引用类型都是对象（通过点语法动态添加属性、方法）

在JS当中基本数据类型都不是对象（不能通过点语法动态添加属性、方法）

# 函数上下文\*\*\*\*\*\*

概述：函数上下文即为this，它只能在函数体中使用，函数上下文this（它代表的数值并不是一个固定数值），

函数上下文数值取决函数该如何调用执行；

问题：函数执行在JS当中有几种呀？（五种）

* 函数名+小括号 ;
* 函数作为事件处理函数执行
* 定时器回调函数执行
* 函数作为数组元素枚举出来执行
* 函数作为对象方法执行

综上所述：函数执行方式有五种，不同执行方法，函数上下文this 是不同的；

## 3.1函数名+小括号执行

概述：函数名+小括号执行，函数体中上下文为window[即为BOM对象：浏览器对象]

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

  //先认识BOM【browser object model】浏览器对象模型

  //在模拟浏览器

  //函数名+小括号执行，函数上下文即为BOM【内置window对象】

  function fun (){

      //这个this即为函数上下文

      console.log(this);

  }

  //如果是函数名+小括号调用函数上下文为window【所谓BOM对象】

  fun();

</script>

## 3.2函数作为事件处理函数执行

概述：函数作为事件处理函数执行，函数的上下文即为触发事件的当前标签。

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <style>

       div{

            width:400px;

            height: 200px;

            background-color: black;

       }

    </style>

</head>

<body>

     <div id="box"></div>

</body>

</html>

<script>

  //获取div标签

  var div = document.getElementById("box");

  //绑定单击事件

  div.onclick = function(){

     //函数体中上下文

     this.style.background = "red";

  }

</script>

## 3.3定时器回调函数执行

概述：在JavaScript语言当中，定时为回调函数会每隔一段时间执行一次，

而定时器回调函数上下文为window；

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

  //定时器回调函数每隔一段事件执行一次

  //第一个参数：函数（称之为回调函数）

  //第二个参数：是一个数字（代表的是定时器间隔）单位MS

  setInterval(function(){

      console.log(this);

  },1500);

</script>

## 3.4函数作为数组元素枚举出来执行

概述：函数作为数组元素枚举出来执行，函数的上下文为当前数组。

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

  var arr = [1,2,3,4,5,function(){

       console.log("我是数组里面函数----我执行了");

       this.reverse();

  },6,7,8,9,10];

  //枚举出函数执行

  arr[5]();

  console.log(arr);

</script>

## 3.5函数作为狭义对象方法执行

概述：函数作为狭义对象方法执行，谁最后打点函数的上下文即为谁.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //狭义对象

   var jch = {

       name:"贾成豪",

       age:18,

       sex:"男",

       eat:function(){

           console.log(this.name+"我可以吃八级米饭");

       }

   }

//调用对象方法

jch.eat();

</script>

总结：函数上下文五大规律

1：函数名+小括号执行=>window

2:函数作为事件处理函数执行=>当前触发这个事件标签

3：定时器回调函数每隔一段事件执行=>window

4:函数作为数组元素枚举出来执行=>当前数组

5：函数作为狭义对象方法执行=>最后打点对象

# 构造函数

概述：在JavaScript语言当中也有所谓构造函数【constructor】，JS当中构造函数要符合如下两个特征：

* 函数名字首个英文字母一般需要大写【不是决定形因素】
* 构造函数务必、必须、一定是一个关键字new调用的【决定形因素】

## 4.1先认知关键字new

概述：在上一次课程当中，我们知道函数执行一共五种，关键字new其实也是调用函数另外一种方式。

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

  //认知关键字new：它也是调用函数一种方式

  //函数声明部分

  function Fun (){

   console.log("兄弟你执行呀");

  }

  //关键字new 也是调用函数一种手段

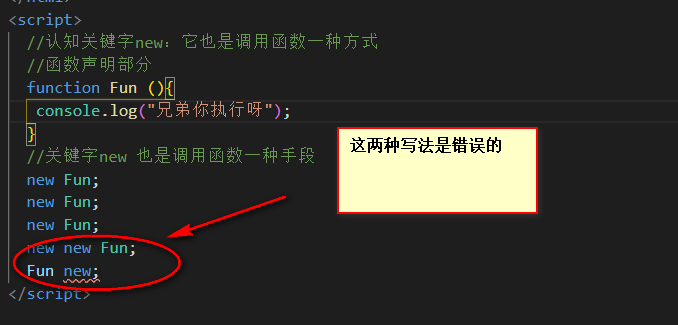
  new Fun;

  new Fun;

  new Fun;

</script>

* 关键字new也是调用函数一种方式【6种方式】
* 关键字new需要放在函数名字前面



## 4.2构造函数基本使用

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

   //构造函数两个特征:

   //第一：构造函数的首个英文字母一般大写的【不是决定性因素】

   //第二：函数如果是new调用的，称之为构造函数【决定形式因素】

   //问题：老师函数名+小括号调用函数执行、new调用函数也执行，两者也有什么区别呀？

   function People(){

     //第一步：在函数体中神秘创建了一个空的狭义对象【大的花括号】

     //第二步:函数体种上下文指向当前这个空的狭义对象

     //第三步：可以通过点语法给狭义对象动态添加属性、方法

     this.xingming = "小明";

     this.age = 18;

     this.sex = "男";

     this.hobby = ['吃饭','睡觉','打豆豆'];

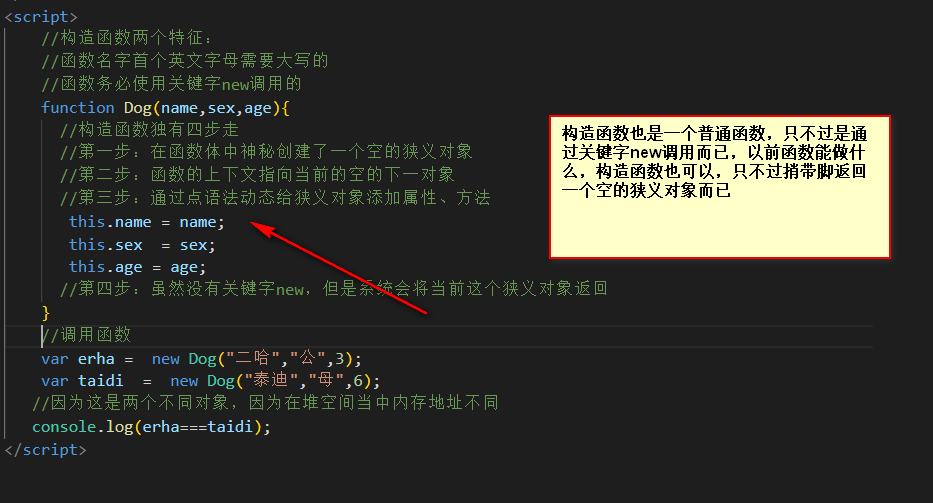
     //第四步：虽然没有关键字return，但是系统回自动将狭义对象返回

   }

   var obj = new People;

   console.log(obj);

</script>



<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

    //构造函数两个特征：

    //函数名字首个英文字母需要大写的

    //函数务必使用关键字new调用的

    function Dog(name,sex,age){

      //构造函数独有四步走

      //第一步：在函数体中神秘创建了一个空的狭义对象

      //第二步：函数的上下文指向当前的空的下一对象

      //第三步：通过点语法动态给狭义对象添加属性、方法

       this.name = name;

       this.sex  = sex;

       this.age = age;

      //第四步：虽然没有关键字new，但是系统会将当前这个狭义对象返回

      for(var i = 0 ; i < 10 ;i++){};

      if(true){};

    }

    //调用函数

    var erha =  new Dog("二哈","公",3);

    var taidi  =  new Dog("泰迪","母",6);

   //因为这是两个不同对象，因为在堆空间当中内存地址不同

   console.log(erha,taidi);

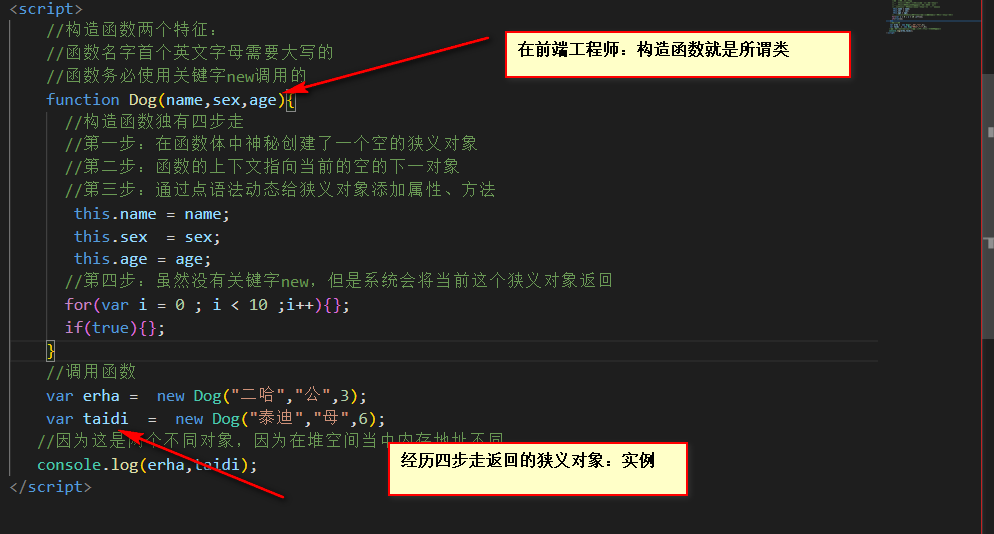
</script>

* 构造函数也是一个函数，以前函数能做的构造函数也能在【循环、条件】
* 只不过构造函数可以捎带脚返回一个狭义对象而已

## 4.3原型----prototype

概述：在前端工程师世界当中，构造函数就是所谓‘类’，通过四步走出来的狭义对象，称之为‘实例’：

比如上一个案例：Dog构造函数他们一般称之为Dog类 erha、taidi:一般称之为Dog类的实例；



任意构造函数天生都有一个prototype属性，指向一个空的狭义对象

实例天生拥有一个\_\_proto\_\_属性，向上面的狭义对象借用方法使用，需要的实例进行借用，省内存

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

</head>

<body>

</body>

</html>

<script>

      //任意构造函数 天生都有一个 prototype属性 ，指向一个空的狭义对象

      //创建出来的时候天生拥有一个\_\_proto\_\_属性，可以向上面的狭义对象借用方法使用

      function Dog(name,age,sex){

         //经历四步走

         this.name = name;

         this.age = age;

         this.sex  = sex;

      }

      Dog.prototype.yaoren = function(){

          console.log("我可以咬人");

      }

      //创建二哈实例

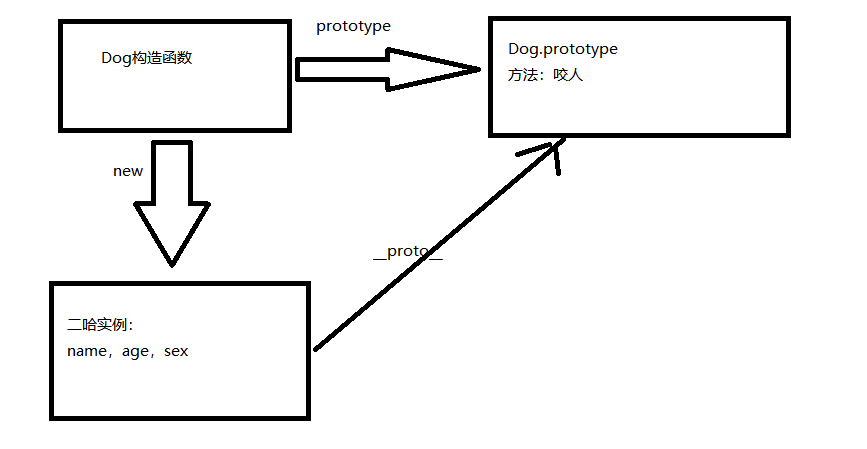
      var erha = new Dog("二哈",18,"公");

      console.log(erha);

      erha.yaoren();

      erha.yaoren();

</script>



# 士兵行走

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Document</title>

    <style>

        \* {

            margin: 0;

            padding: 0;

        }

        div {

            position: absolute;

            width: 50px;

            height: 100px;

            background: url('./img/1.png');

            /\* 背景图定位 \*/

            background-position: 0px -200px;

        }

    </style>

</head>

<body>

</body>

</html>

<script>

    //将来我需要10000士兵行走

    function Bing() {

        //控制背景图发生变化属性

        this.x = 0;

        //top属性值

        this.t = parseInt(Math.random() \* 500);

        //修改left属性值

        this.l = 0;

        //控制小兵是否行走

        this.isMove = false;

        //经历四步走

        this.init();

        //将每隔一秒创建实例添加到数组进行同一管理

        arr.push(this);

        //控制是否行走

        this.bind();

    }

    Bing.prototype.init = function () {

        //在JS当中创建div标签

        this.dom = document.createElement("div");

        this.dom.style.top = this.t + "px";

        //标签上树

        document.body.appendChild(this.dom);

    }

    //跟新脚步方法

    Bing.prototype.update = function () {

        if (this.isMove) return;

        //控制背景图发生变化属性值

        this.x++;

        //修改控制left变化属性值

        this.l += 10;

        if (this.x > 3) this.x = 0;

        //修改div背景图定位

        this.dom.style.backgroundPosition = -this.x \* 50 + "px -200px";

        this.dom.style.left = this.l + "px";

        //当达到一定条件删除标签

        if (this.l > 500) {

            //将数组里面实例移出

            arr.splice(arr.indexOf(this), 1);

            //节点移出

            document.body.removeChild(this.dom);

        }

    }

    Bing.prototype.bind = function () {

        var self = this;

        this.dom.onclick = function () {

            self.isMove = !self.isMove;

        }

    }

    //向每隔一秒创建一个士兵

    var time = 0;

    //创建数组接受全部实例

    var arr = [];

    //开启定时器一直运动

    setInterval(function () {

        time++;

        if (time % 10 == 0) new Bing;

        //让每一个士兵向右侧行走---遍历数组

        for (var i = 0; i < arr.length; i++) {

            arr[i].update();

        }

    }, 100);

</script>