# Js基础

|  |  |  |
| --- | --- | --- |
| 修改日期 | 备注 | 修改人 |
| 2019-9-11 | 1.1~1.5js基本内容 | 王文金 |
| 2019-9-11 | 1.6~1.8js基本内容 | 杨昊 |
| 2019-9-11 | 第二章:js扩展 | 施一航 |
| 2019-9-11 | 第三章:算法 | 雷雪煜 |
| 2019-9-11 | 4.1~4.4游戏 | 刘伟业 |
| 2019-9-11 | 4.5~4.8游戏 | 王程鑫 |

# 目次

[Js基础 1](#_Toc16713)

[目次 I](#_Toc27587)

[1. Javascript基本内容 1](#_Toc20212)

[1.1. Javascript简介 1](#_Toc8050)

[1.2. Javascript三神坑 1](#_Toc6645)

[1.3. JavaScript基本语法 2](#_Toc6116)

[1.4. 数字 6](#_Toc354)

[1.5. 字符串 7](#_Toc22521)

[1.6. 数组 9](#_Toc9264)

[1.7. 对象 10](#_Toc29697)

[1.8. 函数 12](#_Toc14817)

[2. Javascript扩展 13](#_Toc4156)

[2.1. 变量 13](#_Toc32102)

[2.2. 数据类型 13](#_Toc27816)

[2.3. 作用域 15](#_Toc9301)

[2.4. Error对象 17](#_Toc12259)

[2.5. 调试方法 17](#_Toc27820)

[2.6. 计时器/定时器 18](#_Toc29285)

[3. 算法专题 19](#_Toc23982)

[3.1. 引言 19](#_Toc21611)

[3.2. 冒泡排序 19](#_Toc31899)

[3.3. 要点 19](#_Toc31808)

[3.4. 算法思想 19](#_Toc31839)

[3.5. 递归算法 22](#_Toc1995)

[4. Javascript例子\_练习 24](#_Toc26463)

[4.1. 炮弹游戏 24](#_Toc5000)

[4.2. 关灯 28](#_Toc32017)

[4.3. 水游戏 29](#_Toc19032)

[4.4. 下雨 29](#_Toc2315)

[4.5. 打飞机 30](#_Toc19663)

[4.6. 抓猪 34](#_Toc26521)

[4.7. 点名页面 35](#_Toc29670)

[4.8. 转盘 37](#_Toc19503)

# Javascript基本内容

参考网址: 网道<http://wangdoc.com/javascript/index.html>

菜鸟: <https://www.runoob.com/js/js-tutorial.html>

## Javascript简介

JavaScript :

是脚本语言, 弱类型

是一种轻量级的编程语言。

是可插入 HTML 页面的编程代码。

插入 HTML 页面后，可由所有的现代浏览器执行。

## Javascript三神坑

### 字符串与数字

由于javascript是弱类型的语言所以导致console.info(1); 与 console.info("1"); 输出的结果都是 1 ;

但是这并不代表这是相等的.

数字: 1 + 1 = 2 ;

字符串: 1” + 1 = “11”; (字符串的拼接)

== 只判断数值

=== 判断数据类型与数值

举例:

var a = 1; b = "1";

console.info(a == b) 输出为 true

console.info(a === b) 输出为 false

### 布尔值

首先接受布尔值, 它只有两个值 true 和 fasle

if( fasle){

var a = 1;

console.info(a);

}

当判断条件为 false 和 0 时, 控制台输出undefine

当判断条件为 true, 1, 非零时 控制台输出 1 ;

### 变量声明

全局变量: 声明变量时不使用var. 作用范围全局

局部变量: 声明的变量使用var 作用范围仅在局部

## JavaScript基本语法

参考网址:

[http://wangdoc.com/javascript/basic/grammar.html#%E8%AF%AD%E5%8F%A5](http://wangdoc.com/javascript/basic/grammar.html" \l "%E8%AF%AD%E5%8F%A5)

### 语句

avaScript 程序的执行单位为行（line），也就是一行一行地执行。一般情况下，每一行就是一个语句。

语句（statement）是为了完成某种任务而进行的操作，比如下面就是一行赋值语句。

var a = 1 + 3;

语句以分号结尾，一个分号就表示一个语句结束。多个语句可以写在一行内。

var a = 1 + 3 ; var b = 'abc';

### 变量

#### 概念

变量是对“值”的具名引用。变量就是为“值”起名，然后引用这个名字，就等同于引用这个值。变量的名字就是变量名。

var a = 1;

变量的声明和赋值，是分开的两个步骤，上面的代码将它们合在了一起，实际的步骤是下面这样。

var a;

a = 1;

#### 变量提升

JavaScript 引擎的工作方式是，先解析代码，获取所有被声明的变量，然后再一行一行地运行。这造成的结果，就是所有的变量的声明语句，都会被提升到代码的头部，这就叫做变量提升（hoisting）。

console.log(a);

var a = 1;

上面代码首先使用console.log方法，在控制台（console）显示变量a的值。这时变量a还没有声明和赋值，所以这是一种错误的做法，但是实际上不会报错。因为存在变量提升，真正运行的是下面的代码。

var a;console.log(a);

a = 1;

最后的结果是显示undefined，表示变量a已声明，但还未赋值。

### 注释

源码中被 JavaScript 引擎忽略的部分就叫做注释，它的作用是对代码进行解释。JavaScript 提供两种注释的写法：一种是单行注释，用//起头；另一种是多行注释，放在/\*和\*/之间。

// 这是单行注释

/\*

这是

多行

注释

\*/

### 条件语句

#### if 结构

if结构先判断一个表达式的布尔值，然后根据布尔值的真伪，执行不同的语句。所谓布尔值，指的是 JavaScript 的两个特殊值，true表示真，false表示伪。

if (布尔值)

语句;

// 或者if (布尔值) 语句;

#### if...else 结构

if代码块后面，还可以跟一个else代码块，表示不满足条件时，所要执行的代码。

if (m === 3) {

// 满足条件时，执行的语句

} else {

// 不满足条件时，执行的语句

}

#### switch 结构

多个if...else连在一起使用的时候，可以转为使用更方便的switch结构。

switch (fruit) {

case "banana":

// ...

break;

case "apple":

// ...

break;

default:

// ...

}

#### 三元运算符 ?:

JavaScript 还有一个三元运算符（即该运算符需要三个运算子）?:，也可以用于逻辑判断。

(条件) ? 表达式1 : 表达式2

### 循环语句

#### while 循环

While语句包括一个循环条件和一段代码块，只要条件为真，就不断循环执行代码块。

while (条件)

语句;

// 或者while (条件) 语句;

while语句的循环条件是一个表达式，必须放在圆括号中。代码块部分，如果只有一条语句，可以省略大括号，否则就必须加上大括号。

while (条件) {

语句;

}

下面是while语句的一个例子。

var i = 0;

while (i < 100) {

console.log('i 当前为：' + i);

i = i + 1;

}

#### for 循环

for语句是循环命令的另一种形式，可以指定循环的起点、终点和终止条件。它的格式如下。

for (初始化表达式; 条件; 递增表达式)

语句

// 或者

for (初始化表达式; 条件; 递增表达式) {

语句

}

for语句后面的括号里面，有三个表达式。

1). 初始化表达式（initialize）：确定循环变量的初始值，只在循环开始时执行一次。

2). 条件表达式（test）：每轮循环开始时，都要执行这个条件表达式，只有值为真，才继续进行循环。

3). 递增表达式（increment）：每轮循环的最后一个操作，通常用来递增循环变量。

下面是一个例子。

var x = 3;for (var i = 0; i < x; i++) {

console.log(i);

}

// 0

// 1

// 2

#### do...while 循环

do...while循环与while循环类似，唯一的区别就是先运行一次循环体，然后判断循环条件。

do

语句while (条件);

// 或者do {

语句

} while (条件);

不管条件是否为真，do...while循环至少运行一次，这是这种结构最大的特点。另外，while语句后面的分号注意不要省略。

下面是一个例子。

var x = 3;var i = 0;

do {

console.log(i);

i++;

} while(i < x);

## 数字

参考网址: <http://wangdoc.com/javascript/types/number.html>

### 整数和浮点数

JavaScript 内部，所有数字都是以64位浮点数形式储存，即使整数也是如此。所以，1与1.0是相同的，是同一个数。

1 === 1.0 // true

### 数值的进制

十进制：没有前导0的数值。

八进制：有前缀0o或0O的数值，或者有前导0、且只用到0-7的八个阿拉伯数字的数值。

十六进制：有前缀0x或0X的数值。

二进制：有前缀0b或0B的数值。

默认情况下，JavaScript 内部会自动将八进制、十六进制、二进制转为十进制。下面是一些例子。

0xff // 255

0o377 // 255

0b11 // 3

如果八进制、十六进制、二进制的数值里面，出现不属于该进制的数字，就会报错。

0xzz // 报错

0o88 // 报错

0b22 // 报错

上面代码中，十六进制出现了字母z、八进制出现数字8、二进制出现数字2，因此报错。

通常来说，有前导0的数值会被视为八进制，但是如果前导0后面有数字8和9，则该数值被视为十进制。

0888 // 888

0777 // 511

前导0表示八进制，处理时很容易造成混乱。ES5 的严格模式和 ES6，已经废除了这种表示法，但是浏览器为了兼容以前的代码，目前还继续支持这种表示法。

### NaN

NaN是 JavaScript 的特殊值，表示“非数字”（Not a Number），主要出现在将字符串解析成数字出错的场合。

扩展:

isNaN (is Not a Number) 常用与判断是否是数字 返回false为数字 返回true则不是数字

### parseInt()

parseInt() 方法用于将字符串转为整数。

parseInt('123') // 123

如果字符串头部有空格，空格会被自动去除。

parseInt(' 81') // 81

### parseFloat

parseFloat方法用于将一个字符串转为浮点数。

parseFloat('3.14') // 3.14

如果字符串符合科学计数法，则会进行相应的转换。

parseFloat('314e-2') // 3.14

parseFloat('0.0314E+2') // 3.14

## 字符串

参考网址: <http://wangdoc.com/javascript/types/string.html>

### **定义**

字符串就是零个或多个排在一起的字符，放在单引号或双引号之中。

'abc'

"abc"

### 字符串与数组

字符串可以被视为字符数组，因此可以使用数组的方括号运算符，用来返回某个位置的字符（位置编号从0开始）。

var s = 'hello';

s[0] // "h"

s[1] // "e"

s[4] // "o"

// 直接对字符串使用方括号运算符

'hello'[1] // "e"

如果方括号中的数字超过字符串的长度，或者方括号中根本不是数字，则返回undefined。

'abc'[3] // undefined

'abc'[-1] // undefined

'abc'['x'] // undefined

但是，字符串与数组的相似性仅此而已。实际上，无法改变字符串之中的单个字符。

var s = 'hello';

delete s[0];

s // "hello"

s[1] = 'a';

s // "hello"

s[5] = '!';

s // "hello"

上面代码表示，字符串内部的单个字符无法改变和增删，这些操作会默默地失败。

### length 属性

length属性返回字符串的长度，该属性也是无法改变的。

var s = 'hello';

s.length // 5

s.length = 3;

s.length // 5

s.length = 7;

s.length // 5

上面代码表示字符串的length属性无法改变，但是不会报错。

indexOf 判断字符串下标

字符串是由下标的

将字符串转成

小写: toLowerCase()

大写: toUpperCase()

trim() 去除首尾多余的字符

## 数组

参考网址: <https://www.w3school.com.cn/js/js_obj_intro.asp>

### 介绍

javascript数组用于在单一变量中存储多个值,数组是一种特殊的变量,它能一次存放一个以上的值.

var a=[“ ” , ” ”];

var a = new Array()

可以通过下标来引用数组中的某个元素,例子如下:

var a = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = a[0];

[0]是数组的第一个元素,[1]是第二个!

### 数组的插入和替换

替换元素，会改变原来的数组

arr.splice( 1（插入数据）,2（删除数据 一般为0）,"leno","wang");

数组是一种特殊的对象类型,在Javascript中对数组使用typeof运算符会返回”object”.

数组元素也可以是对象.可以在相同数组中存放不同类型的变量.

### 数组属性和方法.

var x = cars.length; // length 属性返回元素的数量

var y = cars.sort(); // sort() 方法对数组进行排序

length属性会返回数组的长度(数组元素的数目).length属性始终大于最高数组的下标.

### 数组的拼接

a1.concat(a2)

var a1 = [0,1,2,3,4] var a2 = [5,6,7,8,9]

a1.concat(a2); 方法不会改变数组 ，而是返回一个新的数组 // var a3 = a1.concat(a2(不限定个数));

### 数组的排序

arr.sort() //非常强大

//只适用于数字

points.sort(function(a,b){ return a - b} )

第一个参数 -

第二个参数 = 从小到大

points.sort(function(a,b){ return b - a} )

### 遍历数组元素

var fruits, text, fLen, i;

fruits = ["Banana", "Orange", "Apple", "Mango"];

fLen = fruits.length;

text = "<ul>";

for (i = 0; i < fLen; i++) {

text += "<li>" + fruits[i] + "</li>";

}

### 数组和对象的区别.

在javascript中,数组使用数字索引.

在javascript中,对象使用命名索引.

## 对象

参考网址: <https://www.w3school.com.cn/js/js_objects.asp>

### 对象属性(值对被称为属性)

var person = {firstName:"Bill", lastName:"Gates", age:62, eyeColor:"blue"};

|  |  |
| --- | --- |
| 属性 | 属性值 |
| Firstname | Bill |
| Lastname | Gates |
| Age | 62 |
| eyeColor | blue |

### This关键词

在函数中,this引用该函数的”拥有者”

1.对象的属性 是可以任意增删的

json 对象 //描述对象的格式，从文本上看，它与对象的结构没有任何区别

所有建名都是字符串！！！

### //键值对 key - value 结构

在对象中（key - value），key称之为属性，为了统一，也可以这样说： 属性名-属性值

for( var x in student ){ console.info( x(属性名) + ":" + student[x]（属性值） )}

属性的删除

delete student.xxx ( delete student.对象名 + 属性名)

// in 运算符判断对象是否具有某属性

student.hasOwnProperty("age")

和 in 操作一样，作为容错的重要手段

读取对象的属性值

var a = [0,1,2,3]; a[0];

1. 对象的属性可以是对象
2. 对象的属性可以是函数（也可以是方法）

## 函数

参考网址: <https://www.runoob.com/js/js-function-definition.html>

### 介绍

函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。函数就是包裹在花括号中的代码块，前面使用了关键词 function：(function必须是小写)

Function functionname(){

//所执行的代码

}

函数表达式(可以再变量中储存)

var x = function (a, b) {return a \* b};

function构造函数

var myFunction = new Function("a", "b", "return a \* b");  
var x = myFunction(4, 3);

### 自调用函数

函数表达式可以 "自调用"。

自调用表达式会自动调用。

如果表达式后面紧跟 () ，则会自动调用。

不能自调用声明的函数。

通过添加括号，来说明它是一个函数表达式

### 例子:

(function () {  
    var x = "Hello!!";      // 我将调用自己  
})();

函数是对象,javascript函数有属性和方法,arguments.length属性返回函数调用过程接收到的参数个数.

function myFunction(a, b) {  
    return arguments.length;  
}

### 函数的提升

程序自动提升函数的位置，让其靠前

myFunction (5)

function myFunction (a){retutn a \* a};

# Javascript扩展

## 变量

JavaScript中的变量是用于存储信息的"容器"。

与代数一样，JavaScript 变量可用于存放值（比如 x=5）和表达式（比如 z=x+y）。

变量可以使用短名称（比如 x 和 y），也可以使用描述性更好的名称（比如 age, sum, totalvolume）。

变量必须以字母开头

变量也能以 $ 和 \_ 符号开头（但不推荐这么做）

变量名称对大小写敏感（y 和 Y 是不同的变量）

JavaScript 变量还能保存其他数据类型，比如文本值 (name="Bill Gates")。

在 JavaScript 中，类似 "Bill Gates" 这样一条文本被称为字符串。

### 声明（创建） JavaScript 变量

在 JavaScript 中创建变量通常称为"声明"变量。

我们使用 var 关键词来声明变量：

var carname;

变量声明之后，该变量是空的（它没有值）。

如需向变量赋值，请使用等号：

carname="Volvo";

也可以在声明变量时对其赋值：

var carname="Volvo";

可以在一条语句中声明很多变量。该语句以 var 开头，并使用逗号分隔变量即可：

var lastname="Doe", age=30, job="carpenter";

一条语句中声明的多个不可以赋同一个值:

var x,y,z=1; //x,y 为 undefined，z 为 1

## 数据类型

值类型(基本类型)：字符串（String）、数字(Number)、布尔(Boolean)、对空（Null）、未定义（Undefined）。

引用数据类型：对象(Object)、数组(Array)、函数(Function)。

### 动态类型

JavaScript 拥有动态类型。这意味着相同的变量可用作不同的类型：

var x; // x 为 undefined

var x = 5; // 现在 x 为数字

var x = "John"; // 现在 x 为字符串

### 字符串

字符串是存储字符的变量。

字符串可以是引号中的任意文本。可以使用单引号或双引号。

可以在字符串中使用引号，只要不匹配包围字符串的引号即可：

var answer="It's alright";  
var answer="He is called 'Johnny'";  
var answer='He is called "Johnny"';

### 数字

JavaScript 只有一种数字类型。数字可以带小数点，也可以不带：

var x1=34.00; //使用小数点来写

var x2=34; //不使用小数点来写

### 布尔

布尔（逻辑）只能有两个值：true 或 false。

var x=true;

var y=false;

### 数组

创建名为 cars 的数组：

var cars=new Array();

cars[0]="Saab";

cars[1]="Volvo";

cars[2]="BMW";

或

var cars=new Array("Saab","Volvo","BMW");

或

var cars=["Saab","Volvo","BMW"];

### 对象

对象由花括号分隔。在括号内部，对象的属性以名称和值对的形式 (name : value) 来定义。属性由逗号分隔：

var person={firstname:"John", lastname:"Doe", id:5566};

上面例子中的对象 (person) 有三个属性：firstname、lastname 以及 id。

空格和折行无关紧要。声明可横跨多行：

var person={

firstname : "John",

lastname : "Doe",

id : 5566

};

对象属性有两种寻址方式：

name=person.lastname;

name=person["lastname"];

### Undefined 和 Null

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

cars=null;

person=null;

### 声明变量类型

声明新变量时，可以使用关键词 "new" 来声明其类型：

var carname=new String;

var x= new Number;

var y= new Boolean;

var cars= new Array;

var person= new Object;

## 作用域

作用域是可访问变量的集合。

在 JavaScript 中, 对象和函数同样也是变量。

在 JavaScript 中, 作用域为可访问变量，对象，函数的集合。

JavaScript 函数作用域: 作用域在函数内修改。

### 局部作用域

变量在函数内声明，变量为局部作用域：只能在函数内部访问。

// 外部不能调用 carName 变量

function myFunction() {

var carName = "Volvo";

// 函数内可调用 carName 变量

}

因为局部变量只作用于函数内，所以不同的函数可以使用相同名称的变量。

局部变量在函数开始执行时创建，函数执行完后局部变量会自动销毁。

### 全局变量

变量在函数外定义，即为全局变量。

全局变量有全局作用域: 网页中所有脚本和函数均可使用。

var carName = " Volvo";

// 外部可调用 carName 变量

function myFunction() {

// 函数内也 可调用 carName 变量

}

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

以下实例中 carName 在函数内，但是为全局变量。

// 此处可调用 carName 变量

function myFunction() {

carName = "Volvo";

// 此处可调用 carName 变量

}

### 变量生命周期

JavaScript 变量生命周期在它声明时初始化。

局部变量在函数执行完毕后销毁。

全局变量在页面关闭后销毁。

### 函数参数

函数参数只在函数内起作用，是局部变量。

### HTML 中的全局变量

在 HTML 中, 全局变量是 window 对象: 所有数据变量都属于 window 对象。

//外部可使用 window.carName

function myFunction() {

carName = "Volvo";

}

## Error对象

Error 对象在错误发生时提供了错误的提示信息。

以下代码中 try 语句块包含了未定义的函数 "adddlert" ，执行它会产生错误，catch 语句块会输出该错误的信息：

try {

adddlert("Welcome");

}

catch(err) {

document.getElementById("demo").innerHTML =

err.name + "<br>" + err.message;

}

输出信息：ReferenceError

Adddlert is not defined

### Error 对象属性

name ： 设置或返回一个错误名

Message ： 设置或返回一个错误信息(字符串)

## 调试方法

### 调试工具

浏览器内置调试工具---F12

设置断点 (代码停止执行的位置), 且可以在代码执行时检测变量。

### console.log() 方法

使用 console.log() 方法在调试窗口上可以打印你想查看变量的值

### 设置断点

在调试窗口中，可以设置 JavaScript 代码的断点。

在每个断点上，都会停止执行 JavaScript 代码，以便于检查 JavaScript 变量的值。

在检查完毕后，可以重新执行代码。

## 计时器/定时器

JavaScript 可以在时间间隔内执行。这就是所谓的定时事件。

### Timing 事件

window 对象允许以指定的时间间隔执行代码。

这些时间间隔称为定时事件。

通过 JavaScript 使用的有两个关键的方法：

setTimeout(function, milliseconds)

在等待指定的毫秒数后执行函数。

setInterval(function, milliseconds)

等同于 setTimeout()，但持续重复执行该函数。

setTimeout() 和 setInterval() 都属于 HTML DOM Window 对象的方法。

### setTimeout() 方法（定时器）

window.setTimeout(function, milliseconds);

window.setTimeout() 方法可以不带 window 前缀来编写。

第一个参数是要执行的函数。

第二个参数指示执行之前的毫秒数。

**停止执行方法：**

clearTimeout() 方法停止执行 setTimeout() 中规定的函数。

clearTimeout() 使用从 setTimeout() 返回的变量。

例：

myVar = setTimeout(function, milliseconds);

clearTimeout(myVar)；

### setInterval() 方法（计时器）

setInterval() 方法在每个给定的时间间隔重复给定的函数，即每秒执行一次函数。

window.setInterval(function, milliseconds);

可以不带 window 前缀来写。

第一个参数是要执行的函数。

第二个参数每个执行之间的时间间隔的长度。

**停止执行方法：**

clearInterval() 方法停止 setInterval() 方法中指定的函数的执行。

window.clearInterval(timerVariable)

clearInterval() 方法使用从 setInterval() 返回的变量。

例：

myVar = setInterval(function, milliseconds);

clearInterval(myVar);

# 算法专题

## 引言

我们平时对计算机中存储的数据执行的两种最常见的操作就是排序和查找，对于计算机的排序和查找的研究，自计算机诞生以来就没有停止过。如今又是大数据，云计算的时代，对数据的排序和查找的速度、效率要求更高，因此要对排序和查找的算法进行专门的数据结构设计，以便让我们对数据的操作更加简洁高效。在算法专题内将着重介绍和解析冒泡排序、递归算法，以便我们更好的了解JS算法。

## 冒泡排序

冒泡排序是一种交换排序。

## **要点**

交换排序：两两比较待排序的关键字，并交换不满足次序要求的那对数，直到整个表都满足次序要求为止。

## **算法思想**

它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。

这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端，故名。

假设有一个大小为N的无序序列。冒泡排序就是要每趟排序过程中通过两两比较，找到第i个小（大）的元素，将其往上排。

算法演示如下图2.2 算法实例解析所示。

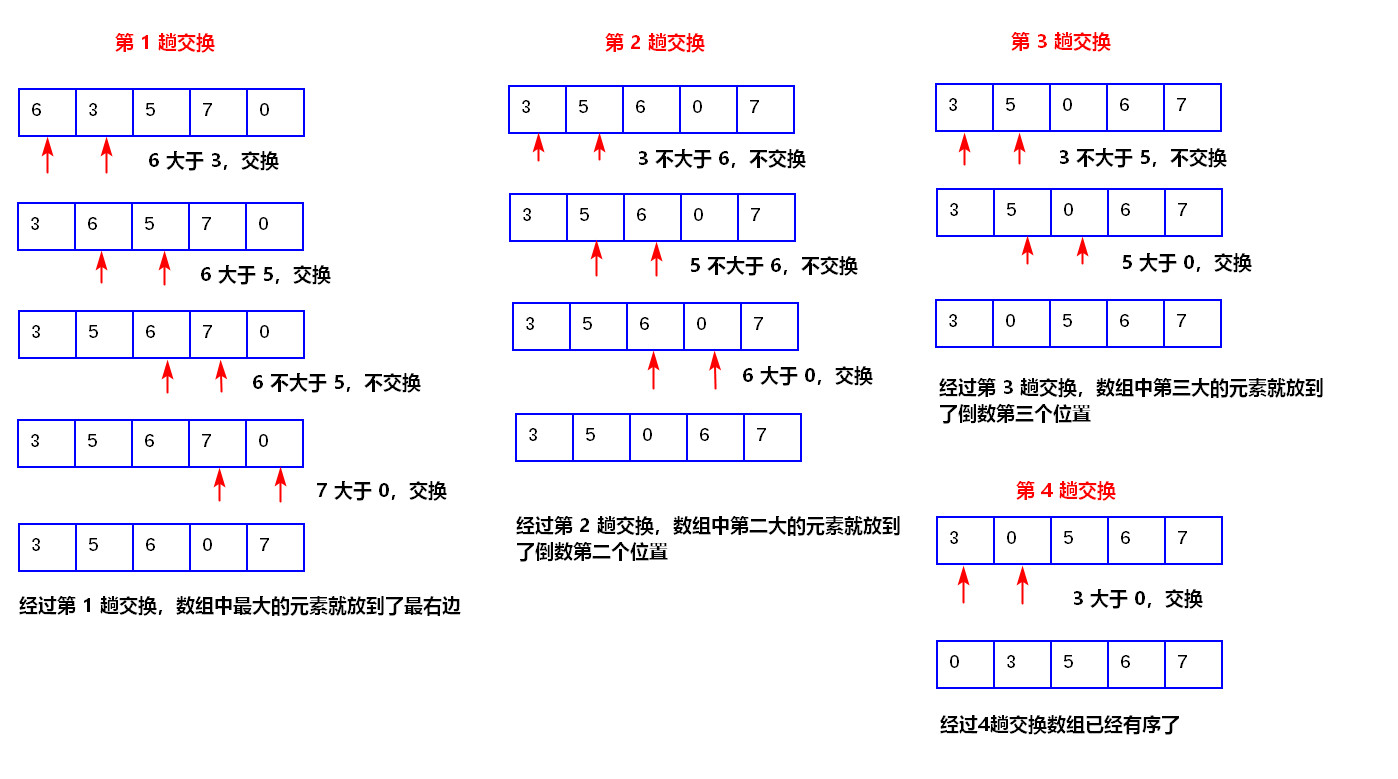


图2.2 算法实例解析

假设要对一个大小为N的无序序列进行升序排序（即从小到大）。

(1)每趟排序过程中需要通过比较找到第i个小的元素。

所以，我们需要一个外部循环，从数组首端(下标0) 开始，一直扫描到倒数第二个元素（即下标N - 2)，剩下最后一个元素，必然为最大。

(2)假设是第i趟排序可知，前i-1个元素已经有序。现在要找第i个元素，只需从数组末端开始，扫描到第i个元素，将它们两两比较即可。

所以，需要一个内部循环，从数组末端开始（下标N - 1），扫描到(下标i + 1)。

**核心代码：**

**function sortArrByParm( arr , attr){**

**arr = arr || \_data.allStudents;**

**attr = attr || "age";**

**var newArr = [];**

**for( var i = 0; i < arr.length-1; i++){**

**for(var j = 0; j < arr.length-1-i; j++){**

**if(arr[j][attr] > arr[j+1][attr]){**

**// exchange**

**var temp = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = temp;**

**}**

**}**

**}**

**\_data.allStudents = arr; // 改变全局变量(所有student的顺序)**

**fillStudents(\_data.allStudents);**

**}**

综上所述冒泡排序的算法思想如下（升序排序）：

（1）比较相邻的元素。如果第一个比第二个大，就交换它们两个；

（2）对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样最终最大数被交换到最后的位置

（3）除了最后一个元素以外，针对所有的元素重复以上的步骤

（4）重复步骤1~3，直到排序完成

### 算法分析

### 冒泡排序算法的性能

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 排序类别 | 排序方法 | 时间复杂度 | | | 空间复杂度 | 稳定性 | 复杂性 |
| 平均情况 | 最坏情况 | 最好情况 |
| 交换排序 | 冒泡排序 | O(N2) | O(N2) | O(N) | O(1) | 稳定 | 简单 |

### 时间复杂度

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数C和记录移动次数M均达到最小值：Cmin= N - 1, Mmin= 0。所以，冒泡排序最好时间复杂度为O(N)。若初始文件是反序的，需要进行N -1趟排序。每趟排序要进行N - i次关键字的比较(1 ≤ i ≤ N - 1)，且每次比较都必须移动记录三次来达到交换记录位置。在这种情况下，比较和移动次数均达到最大值：  
Cmax = N(N-1)/2 = O(N2)  
Mmax = 3N(N-1)/2 = O(N2)  
冒泡排序的最坏时间复杂度为O(N2)。因此，冒泡排序的平均时间复杂度为O(N2)。总结起来，其实就是一句话：当数据越接近正序时，冒泡排序性能越好。

### 算法稳定性

冒泡排序就是把小的元素往前调或者把大的元素往后调。比较是相邻的两个元素比较，交换也发生在这两个元素之间。

所以相同元素的前后顺序并没有改变，所以冒泡排序是一种稳定排序算法。

### 实际问题

//冒泡排序function bubbleSort ( data ) {

var temp = 0;

for ( var i = data.length ; i > 0 ; i -- ){

for( var j = 0 ; j < i - 1 ; j++){

if( data[j] > data[j + 1] ){

temp = data[j];

data[j] = data [j+1];

data[j+1] = temp;

}

}

}

return data;

}

var dataStore = [ 72 , 1 , 68 , 95 , 75 , 54 , 58 , 10 , 35 , 6 , 28 , 45 , 69 , 13 , 88 , 99 , 24 , 28 , 30 , 31 , 78 , 2 , 77 , 82 , 72 ];

console.log( '原始数据:' + dataStore );console.log( '冒泡排序:' + bubbleSort( dataStore) );

// 原始数据:

72,1,68,95,75,54,58,10,35,6,28,45,69,13,88,99,24,28,30,31,78,2,77,82,72

// 冒泡排序:

1,2,6,10,13,24,28,28,30,31,35,45,54,58,68,69,72,72,75,77,78,82,88,95,99

## 递归算法

 在数学与计算机科学中，递归(Recursion)是指在函数的定义中使用函数自身的方法。实际上，递归，顾名思义，其包含了两个意思：递 和 归，这正是递归思想的精华所在。

### 递归概念

1.在程序中函数直接或间接调用自己；

（1）直接调用自己；

（2）简介调用自己；

2.跳出结构,有了跳出才有结。

### 递归思想

递归的基本思想就是把规模大的问题转化为规模小的相似的子问题来解决。特别地，在函数实现时，因为解决大问题的方法和解决小问题的方法往往是同一个方法，所以就产生了函数调用它自身的情况，这也正是递归的定义所在。格外重要的是，这个解决问题的函数必须有明确的结束条件，否则就会导致无限递归的情况。

1.递归的调用,最终还是要转换为自己这个函数；

2.如果有个函数foo,如果他是递归函数,到最后问题还是转换为函数foo的形式；

3.递归的思想就是将一个未知问题转换为一个已解决的问题来实。

### 递归与循环

　 递归与循环是两种不同的解决问题的典型思路。递归通常很直白地描述了一个问题的求解过程，因此也是最容易被想到解决方式。循环其实和递归具有相同的特性，即做重复任务，但有时使用循环的算法并不会那么清晰地描述解决问题步骤。单从算法设计上看，递归和循环并无优劣之别。然而，在实际开发中，因为函数调用的开销，递归常常会带来性能问题，特别是在求解规模不确定的情况下；而循环因为没有函数调用开销，所以效率会比递归高。递归求解方式和循环求解方式往往可以互换，也就是说，如果用到递归的地方可以很方便使用循环替换，而不影响程序的阅读，那么替换成循环往往是好的。问题的递归实现转换成非递归实现一般需要两步工作：

（1）自己建立“堆栈(一些局部变量)”来保存这些内容以便代替系统栈，比如树的三种非递归遍历方式；

（2）把对递归的调用转变为对循环处理。特别地，在下文中将给出递归算法的一些经典应用案例。

### 实际问题

**（1）阶乘**

/\*\*

\* Title: 阶乘的实现

\* Description:

\* 递归解法

\* 非递归解法

\* @author rico

\*/public class Factorial {

/\*\*

\* @description 阶乘的递归实现

\* @author rico

\* @created 2017年5月10日 下午8:45:48

\* @param n

\* @return

\*/

public static long f(int n){

if(n == 1) // 递归终止条件

return 1; // 简单情景

return n\*f(n-1); // 相同重复逻辑，缩小问题的规模

}

--------------------------------我是分割线-------------------------------------

/\*\*

\* @description 阶乘的非递归实现

\* @author rico

\* @created 2017年5月10日 下午8:46:43

\* @param n

\* @return

\*/

public static long f\_loop(int n) {

long result = n;

while (n > 1) {

n--;

result = result \* n;

}

return result;

}

}

**（2）查询警员**

// 查询警员

// 递归函数 -- 自己调用自己,并且层层展开

function countPoliceFromArray(policeArr){

for(var x in policeArr){

if( policeArr[x].hasOwnProperty("id") ){

\_allPolice.push(policeArr[x]);

}

if( typeof(policeArr[x]) == 'object' ){

countPoliceFromArray(policeArr[x]);

}

}

}

# Javascript例子\_练习

## 炮弹游戏

禁用鼠标右键:

function gameInit() {

// 禁用鼠标右键

document.body.oncontextmenu = function(e) {

return false;

}

}

复位:建立自定义函数设定起始位置

// 炮弹复位

function resetBom(bomId) {

document.getElementById(bomId).style.left = "5%";

document.getElementById(bomId).style.top = "80%";

}

//僵尸复位

function resetMonster(monsterId) {

document.getElementById(monsterId).style.left = "90%";

}

//僵尸移动

function monsterMove(monsterId) {

var monsterItem = document.getElementById(monsterId);

if (!monsterItem) {

console.info("获取僵尸个体出错");

return;

}

setInterval(function() {

var currX = monsterItem.style.left;

currX = currX.replace("%", "");

currX = parseFloat(currX);

currX -= 0.06; // 步幅

monsterItem.style.left = currX + "%";

// 第一次碰撞

hitCheck("monster\_hit\_canon", currX);

}, 20);

}

// 通过滚轮调整炮口 //炮口角度取值在 -60 到0 之间

window, onmousewheel = function(e) {

//console.info(e);

var currAngle = document.getElementById("stick1").style.transform;

if (!currAngle) {

currAngle = 0;

} else {

currAngle = currAngle.replace("rotate(", "");

currAngle = currAngle.replace("deg)", "");

currAngle = parseInt(currAngle);

}

if (e.shiftKey) {

if (e.deltaY < 0) {

console.info("滚轮向上");

currAngle -= 1;

} else {

console.info("滚轮向下");

currAngle += 1;

}

}

if (currAngle > -60 && currAngle < 0) {

document.getElementById("stick1").style.transform = "rotate(" + currAngle + "deg)";

formattAngle(currAngle);

}

}

// 将炮弹的角度传递出去.换算成炮弹的角度输入炮口角度,输出炮弹角度 -60 到 0

function formattAngle(currAngle) {

var trans001 = currAngle + 60; //60 ~120

var trans002 = (60 - trans001) / (120 - 60);

var trans003 = (70 - 30) \* trans002 + 30;

\_nextAngle = trans003 || 45; // 计算出错,就按45度角度

}

// 发射炮弹 炮弹角度在 30 ~ 70

var \_bomTimer = 0;

function bomMove(angle) {

// 容错

if (angle > 70) {

return;

}

clearInterval(\_bomTimer);

resetBom("bom001");

var angle = angle;

var gggg = 0.02; // 加速度

var xSpeed = 0.6; // px

var ySpeed = Math.tan(angle \* 0.017453293) \* xSpeed; //

\_bomTimer = setInterval(function() {

// x 方向的运动

var bomCurrX = document.getElementById("bom001").style.left;

bomCurrX = parseFloat(bomCurrX.replace("%", ""));

document.getElementById("bom001").style.left = bomCurrX + xSpeed + "%";

// 当前Y位置

var currY = document.getElementById("bom001").style.top;

currY = parseFloat(currY.replace("%", ""));

// 减去加速度

ySpeed -= gggg; // 这个值非常关键!!!

currY = currY - ySpeed;

// y位移

document.getElementsByClassName("bom")[0].style.top = currY + "%"

// 复位

if (currY > 80) {

hitCheck("bom\_hit\_momster", null, bomCurrX + xSpeed);

document.getElementById("bom001").style.left = "10%";

document.getElementById("bom001").style.top = "80%";

ySpeed = Math.tan(angle \* 0.017453293) \* xSpeed;

clearInterval(\_bomTimer);

}

//console.info(currY + " " + ySpeed );

}, 40)

}

//bomMove( 40);

document.getElementById("fire").onclick = function(e) {

if (!\_nextAngle) {

\_nextAngle = 45;

}

bomMove(\_nextAngle);

}

//碰撞测试

function hitCheck(type, monsterX) {

if (type == "monster\_hit\_canon") {

if (monsterX < 10) {

// 僵尸碰大炮

handle\_monster\_hit\_canon();

} else {

return;

}

}

if (type == "bom\_hit\_monster") {

// var 取出当前monsterde x 值

var monsterX = document.getElementById("monster001").style.left;

monsterX = monsterX.replace("%");

monsterX = parseFloat(monsterX);

if (Math.abs(monsterX - boms) <= 2) {

handle\_bom\_hit\_moster();

} else {

console.info("未命中");

}

}

}

// 处理monster\_hit\_canon

function handle\_monster\_hit\_canon() {

alert("monster\_hit\_canon");

resetMonster("monster001");

}

function handle\_bom\_hit\_momster(){

alert("bom\_hit\_momster - 写样式");

resetMonster("monster001");

## 关灯

var counter1 = 5;

var timer1 = setInterval(function(){

counter1 --;

document.getElementById("countdown1").innerText = counter1;

if( counter1 < 0){

document.getElementById("mask").style.display = "none";

clearInterval( timer1);

}

}, 1000);

**点亮:灰度每次减15 当等于0时变为颜色#FFFF00**

document.getElementById("bt1").onclick = function( e ){

document.getElementById("light1").style.filter = "grayscale(100%)";

document.getElementById("light1").style.visibility = "visible";

var grayscale001 = 100;

timer2 = setInterval(function(){

grayscale001 -= 15;

document.getElementById("light1").style.filter = "grayscale("+ grayscale001 +"%)";

if( grayscale001 <= 0){

document.body.style.background = "#FFFF00";

clearInterval(timer2);

}

}, 400);

}

**关灯:**

document.getElementById("bt2").onclick = function( e ){

clearInterval(timer2); // 终止点亮过程, 进入关灯

document.getElementById("light1").style.filter = "grayscale(100%)";

//

// 反操作 clearTimeout()

setTimeout(function(){

document.getElementById("light1").style.visibility = "hidden";

document.body.style.background = "#000000";

}, 1000);

};

## 水游戏

水游戏:我不会

拖放:

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">

function allowDrop(ev) {

ev.preventDefault(); // 固定写法, 还原其默认行为

}

function drag(ev) {

ev.dataTransfer.setData("Text", ev.target.id);

}

function drop(ev) {

console.info(ev.target.id)

ev.preventDefault();

var data = ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(data));

}

// document.getElementById("div2").appendChild( document.getElementById("drag1") );

// 将页面上的某节点搬运到目标节点下

// ondragend 事件函数/ 事件响应

function end( e ){

// ev.preventDefault(); // 固定写法, 还原其默认行为

// console.info(e.clientX );

e.target.style.left = e.clientX + "px";

e.target.style.top = e.clientY + "px";

}

## 下雨

var rainCount = 0; // 当前雨滴数量

var rainMax = 200; // 雨滴总数量

function doRainFall() {

if( rainCount >= rainMax ){

return;

}

var rainFall = document.createElement('div'); //创建div

rainFall.setAttribute("class", "rain");

rainFall.style.background = "url(\"xuehua2.png\")no-repeat"

rainFall.style.top = "0px";

rainFall.style.left = Math.random()\*100 + "%";

document.body.appendChild(rainFall);

rainCount += 1;

var rainTimer = setInterval(function() {

var newTop = rainFall.style.top.replace("px", "");

newTop = parseFloat(newTop);

newTop += Math.floor( Math.random()\* 15 + 5 ); //newTop += 10;

// 落到地面的时候让它消失

if( newTop >= 800){

rainFall.remove();

rainCount -= 1;

clearInterval( rainTimer );

}

rainFall.style.top = newTop + "px";

}, 100);

}

var doRain = setInterval(function(){

doRainFall();

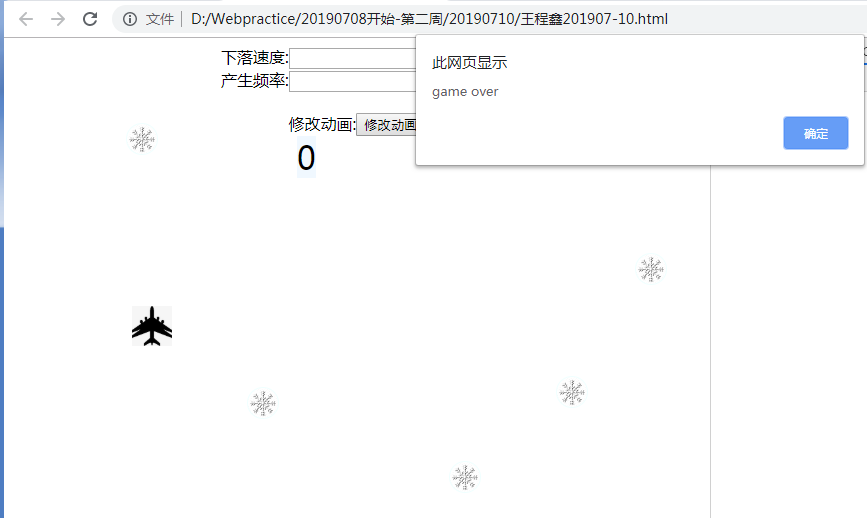
} , 800);

## 打飞机

打飞机的讲解内容:

页面上有两个input,type为text,一个输入下落速度的参数,另一个输入产生频率的参数,在两个input文本框下有一个修改动画的按钮input,type为button,两个文本框输入参数后点击修改动画才会生效,修改动画下还有一个简单的计分板,这个计分板是通过鼠标左击屏幕中下落的雪花消失后才会计分.而图下面还有个飞机一样的图标,可以通过键盘监听事件绑定进行在页面上移动,w,a,s,d,分为上左下右.而雪花和飞机的会产生碰撞事件,碰撞后会出现弹窗提示.点击确定之后再次开始游戏.

在页面上呈现的出的视图如下:



获取input的值(传进来目标input标签的id值)

修改动画,需要两个参数1.下落速度,2.产生速率

function getTextValue() {

var fallingSpeed = document.getElementById("text1").value;

var createHZ = document.getElementById("text2").value;

//容错

if (!fallingSpeed || isNaN(fallingSpeed) ||

!createHZ || isNaN(createHZ)) {

console.info("参数非数字");

//没有值不处理

return;

} else {

//调整动画

settingAnimate(fallingSpeed, createHZ);

}

}

雪花部分的代码如下:

雪花和计分板的初始值:

var rainCount = 0; // 当前雪花数量

var rainMax = 300; // 雪花总数量

var yourTotal = 0;

document.getElementById("total").innerText = yourTotal;

每执行一次,一个雪花产生和消亡:

function doRainFall(fallingSpeed) {

if (rainCount >= rainMax) {

return;

}

if (!fallingSpeed) {

fallingSpeed = 100;

}

var rainFall = document.createElement('div'); //创建div

rainFall.setAttribute("class", "rain");

rainFall.style.background = "url(\"1.jpg\")no-repeat"

rainFall.style.top = "0px";

rainFall.style.left = Math.random() \* 1200 + "px";

rainFall.onclick = function() {

rainCount -= 1;

rainFall.remove();

yourTotal += 1;

document.getElementById("total").innerText = yourTotal;

if (rainTimer) {

clearInterval(rainTimer);

}

};

// 产生雪花

document.body.appendChild(rainFall);

rainCount += 1;

var rainTimer = setInterval(function() {

var newTop = rainFall.style.top.replace("px", "");

newTop = parseFloat(newTop);

newTop += Math.floor(Math.random() \* 15 + 5); //newTop += 10;

rainFall.style.top = newTop + "px";

// 取下落物体的x

var newLeft = rainFall.style.left.replace("px", "");

newLeft = parseFloat(newLeft);

// 碰撞测试

var planX = document.getElementById("plan").style.left;

planX = planX.replace("px", "");

planX = parseInt(planX);

var planY = document.getElementById("plan").style.left;

planY = planY.replace("px", "");

planY = parseInt(planY);

// 小于40发生碰撞

if (Math.abs(newLeft - planX) < 40) {

if (Math.abs(newTop - planY) < 40) {

alert("game over");

location.reload();

}

}

// 落到地面的时候让它消失

if (newTop >= 800) {

rainFall.remove();

rainCount -= 1;

clearInterval(rainTimer);

}

rainFall.style.top = newTop + "px";

}, fallingSpeed);

}

初始下落频率和产生频率:

var doRain = setInterval(function() {

doRainFall(100); //下落频率

}, 400); //产生的频率

调整动画的函数:调整之前要先做清除计时器,

function settingAnimate(fallingSpeed, createHZ) {

clearInterval(doRain); //清除计时器

doRain = setInterval(function() {

doRainFall(fallingSpeed);

}, createHZ);

}

飞机的初始位置:

document.getElementById("plan").style.left = "200px";

document.getElementById("plan").style.top = "500px";

键盘监听部分通过w,s,a,d上下左右操作飞机的移动:

window.onkeypress = function(e) {

var planX = document.getElementById("plan").style.left;

planX = planX.replace("px", "");

planX = parseFloat(planX);

var planY = document.getElementById("plan").style.top;

planY = planY.replace("px", "");

planY = parseInt(planY);

if (e.key == "w") {

planY -= 20;

}

if (e.key == "s") {

planY += 20;

}

if (e.key == "a") {

planX -= 20;

}

if (e.key == "d") {

planX += 20;

}

document.getElementById("plan").style.top = planY + "px";

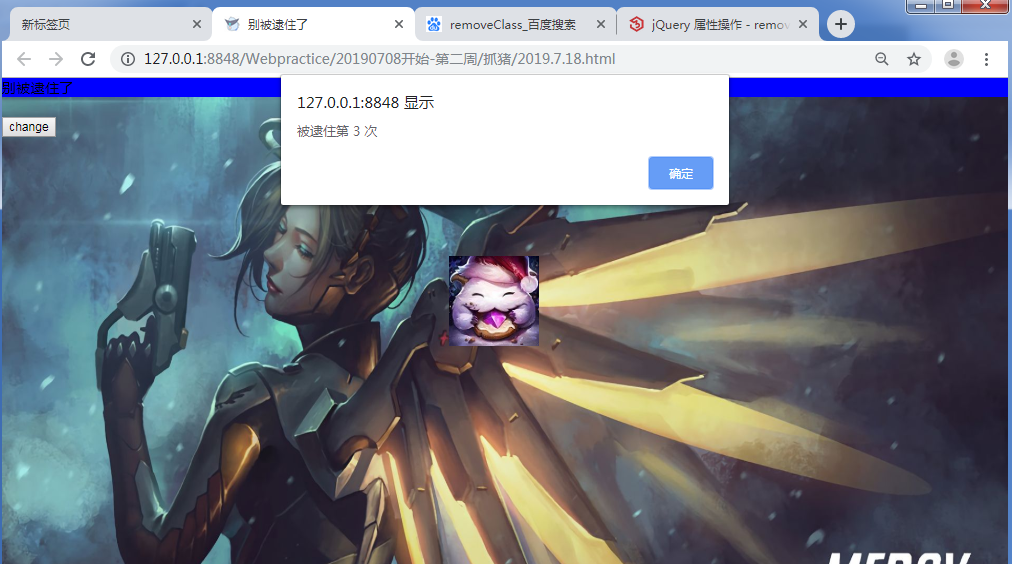
document.getElementById("plan").style.left = planX + "px";

}

## 抓猪

抓猪项目讲解:页面上有一个div里面的内容时别被逮住了,这不是重点,div下面有个input按钮,点击会改变原来div的内填充的颜色,正题来了,页面上有个一直在移动的div,里面放猪的图片,当你在页面上用鼠标点击到了猪的div时会跳出一个弹窗提示,说你抓到了然后计数你抓到的次数.

页面呈现的效果视图:



Body内的代码如下:

<div id="catch">别被逮住了</div>

<br />

<input type="button" value="change" id="bt1" />

<div id="catcher" style="background:url(polo2.jpg);height:100px;width:100px;position:absolute;">

</div>

div别被逮住了初始样式用jquery的addClass()方法获取样式

$(document).ready(function(){

$("#catch").addClass("bgRed");

});

获取了样式之后点击input按钮修改样式,先删除原样式,再添加新样式,用的方法是jquery的addClass()和removeClass()方法

$("#bt1").click(function(e){

$("#catch").removeClass("bgRed");

$("#catch").addClass("bgBlue");

});

关于抓猪部分:

这是猪在浏览器页面上移动的代码:

$(document).ready(function() {

var catchIt = false;

var bskerer;

var catchTimes = 0;

var hz = 50; // 默认 50

function goOnDoIt() {

bskerer = setInterval(function() {

var situationRam = parseInt(Math.random() \* 10);

if (situationRam % 2 == 0) {

$("#catcher").animate({

left: parseInt(Math.random() \* 1000) + 'px'

});

} else {

$("#catcher").animate({

top: parseInt(Math.random() \* 800) + 'px'

});

}

}, hz);

}

goOnDoIt();

这是抓猪事件,猪被抓后出现弹窗提示,用到的方法有鼠标点击事件click,

$("#catcher").click(function() {

catchIt = true;

catchTimes++;

alert('被逮住第 ' + catchTimes + " 次");

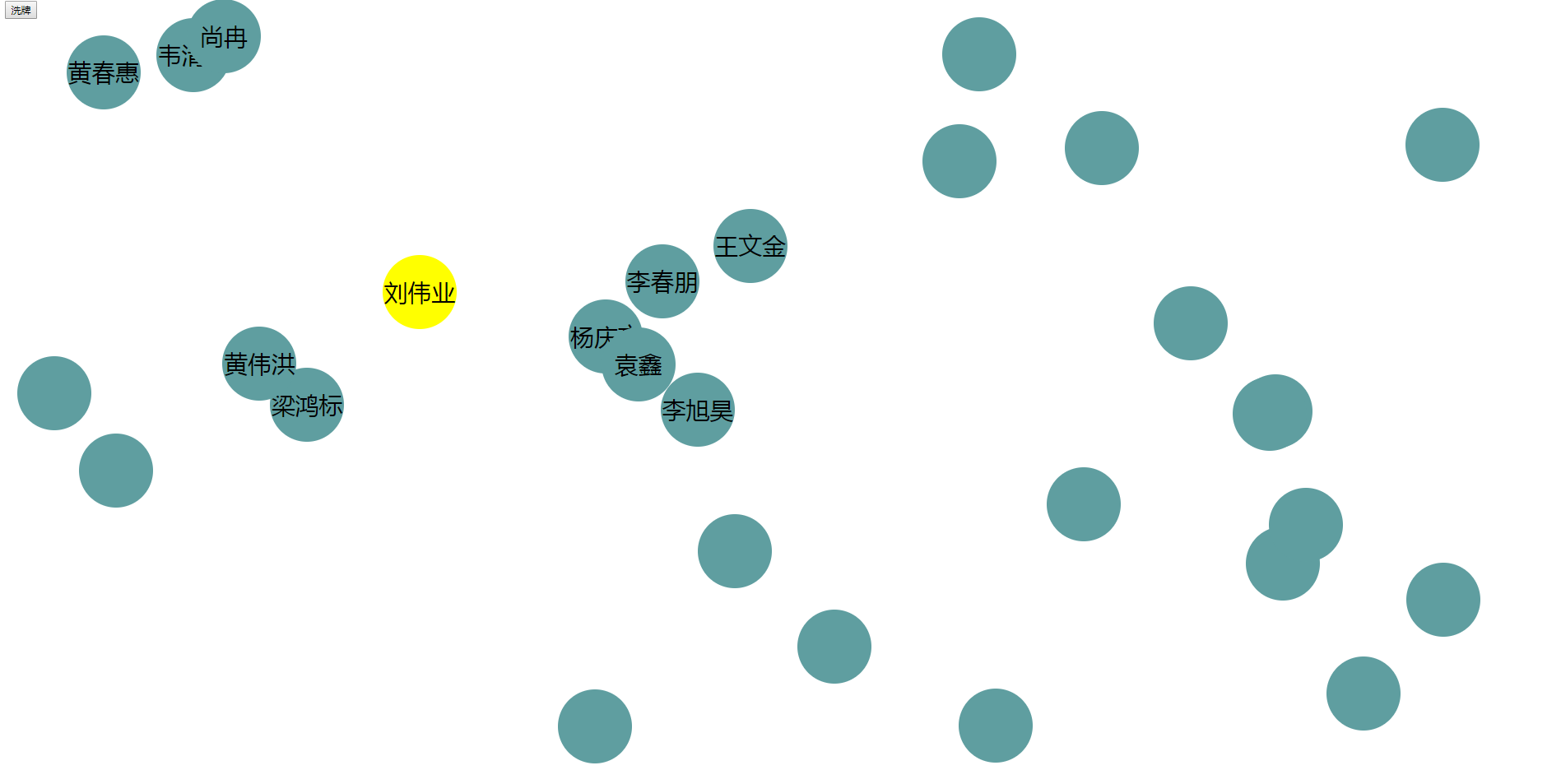
catchIt = false;

});

## 点名页面

点名页面讲解内容大概:页面上有一个input按钮,value=洗牌,点击洗牌出现一个数组里的所有数据,每个数据都设置了相同的样式呈现到页面上,每个数据有name和id两个属性,然后再绑定事件鼠标点击页面上呈现数据的div时,会出现数据的name属性,这就完成了点名.

页面上呈现的视图:



body里的内容如下:

<input type="button" id="Shuffle" value="洗牌" />

一个数组,里面有多个数据代码如下:数据太多不一一写出,写个例子

var allStudents = [{ {

name: "李四",

id: 1

},. . . . . . .}];

// 洗牌事件开始

function createRan() {

// 先删除页面上所有数据(div)通过数据源进行删除

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

var targetHtmlNode = document.getElementById(allStudents[i].id);

if (targetHtmlNode) {

targetHtmlNode.remove();

}

}

// 还有一个错误代码例子,这段代码不可用:

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

allguys[i].remove();//注意,IE不支持这个方法!

}

// 遍历allStudents,将他们都放到页面上,方法有setAttribute()为div添加指定的样式

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

var oneStudent = document.createElement("div");// 创建div节点

oneStudent.setAttribute("class", "student");

oneStudent.id = allStudents[i].id;

oneStudent.onclick = function(evetxxx) {

showName(evetxxx);

}

var xRan = Math.random() \* 90;

var yRan = Math.random() \* 90;

oneStudent.style.left = xRan + "%";

oneStudent.style.top = yRan + "%";

document.body.appendChild(oneStudent);

}

}

这部分代码是遍历allStudents数组,然后输出数据

function showName(evetxxx) {

var stuId = evetxxx.target.id;

console.info(stuId);

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

document.getElementById(allStudents[i].id).style.zIndex = 9;//降低其他人

if (allStudents[i].id == stuId) {

evetxxx.target.innerText = allStudents[i].name;

break;

}

}

}

//事件注册,这里绑定按钮点击洗牌出现所有数据

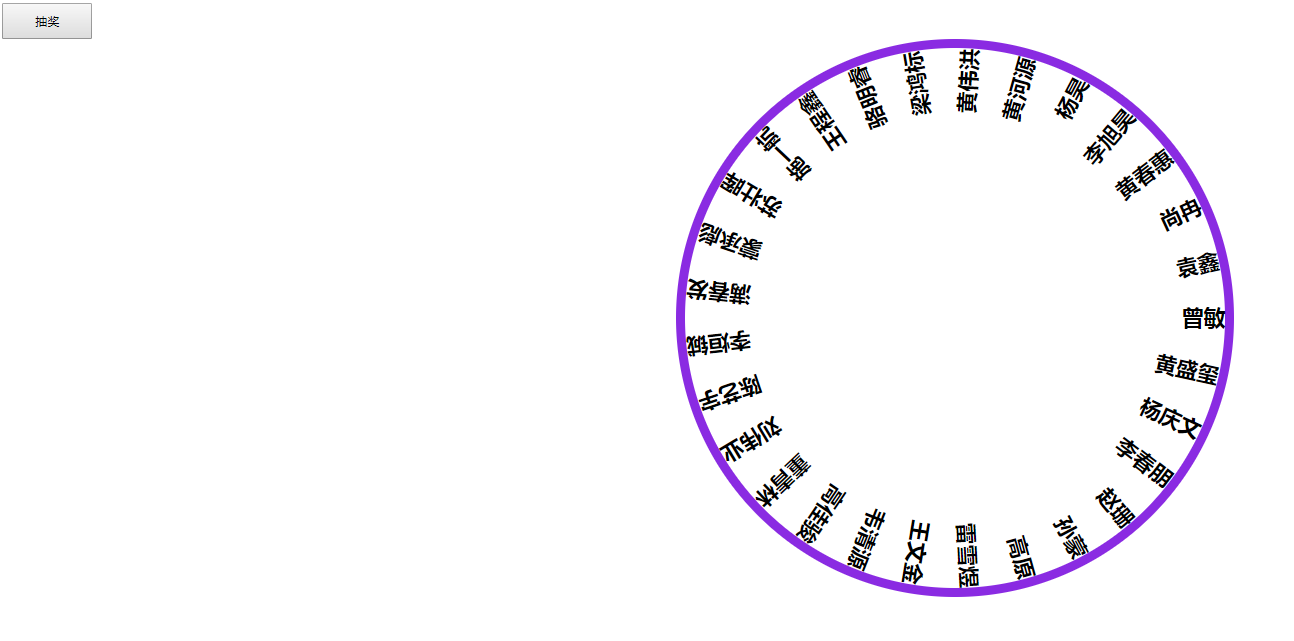
document.getElementById("Shuffle").onclick = function(e) {

createRan();

}

## 转盘

页面上呈现的视图:



Body代码如下:

<input type="button" id="doRnllingBt" class="button" value="抽奖" />

<div class="rollingpan" id="rollingpan1111">

Js代码如下:

var timerPan = 0;

// 转动 或者停止转盘

function doRolling(flag) {

if (timerPan) {

console.info(document.getElementById("rollingpan1111").style.transform)

clearInterval(timerPan);

}

if (flag) {

var anble = 0;

timerPan = setInterval(function() {

anble += 5; // 每次转动的角度

if (anble > 360) {

anble = 0;

}

document.getElementById("rollingpan1111").style.transform = "rotate(" + anble + "deg)";

}, 10);

}

}

var \_isRollong = false; //转盘状态标志位

document.getElementById("doRnllingBt").onclick = function() {

console.info(3);

if (\_isRollong == false) {

\_isRollong = true;

} else {

\_isRollong = false;

}

doRolling(\_isRollong);

}

数组的数据如下,数据太多不一一写出:

var allStudents = [{ {

name: "李四",

id: 1

},. . . . . . .}];

// 此部分是转盘被点击触发后发生的一系列变化

function fillRollingPan(studentsArr) {

var oneAngle = 360 / studentsArr.length;

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

var oneStudent = document.createElement("div");

oneStudent.setAttribute("class", "itemX");

oneStudent.id = "item" + studentsArr[i].id;

oneStudent.innerText = studentsArr[i].name;

var currAngle = oneAngle \* i;

oneStudent.style.transform = "rotate(" + currAngle + "deg)";

document.getElementById("rollingpan1111").appendChild(oneStudent);

}

}

fillRollingPan(allStudents);