## 一、JS介绍

1、数据类型

Number String Boolean Undefined Null Symbol Object

2、求解数据类型

Number string boolean undefined object symbol function

3、转换

NaN 是 Number 类型中的一个特殊取值

Infinity (无穷) 也是 Number 类型中的一个特殊取值

Number()

将其它类型转换为 Number 类型

字符串 --> 数字

兼容格式 --> 对应数字

非兼容格式 --> NaN

"" --> 0

布尔 --> 数字

true --> 1

false --> 0

Undefined --> 数字

undefined --> NaN

Null --> 数字

null --> 0

对象 --> 数字

非null对象 --> NaN

Boolean()

其它类型 --> 布尔

0、-0、null、""、false、undefined 或 NaN，则该对象设置为 false

1. 求某年2月多少天

function twodays(year){

var days=(year%4==0&&year%100!==0 ||year%400==0)?29:28;

return days;

}

## 二、条件结构

1. switch方法求成绩的等级（===运算）

function scoeq(score){

var val = Math.floor(score / 10);

switch (val) {

case 8:

console.log("B");

break;

case 9:

case 10:

console.log("A");

break;

case 7:

console.log("C");

break;

case 6:

console.log("D");

break;

default:

console.log("E");

}

console.log("程序结束....");

}

1. 两个文本框相加减乘除

<input type="text" placeholder="第一个数" id="num1" >

<select id="operator">

<option value="+">+</option>

<option value="-">-</option>

<option value="\*">\*</option>

<option value="/">/</option>

<option value="%">%</option>

</select>

<input type="text" placeholder="第二个数" id="num2" >

<button id="eq">=</button>

<input type="text" placeholder="计算结果" id="result">

<script type="text/javascript">

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

var \_num1=Number(document.getElementById("num1").value);

var \_num2=Number(document.getElementById("num2").value);

var \_operator=document.getElementById("operator").value;

console.log(\_num1,\_num2,\_operator);

var result=0;

switch(\_operator){

case "+":

\_result=\_num1+\_num2;

break;

case "-":

\_result=\_num1-\_num2;

break;

case "\*":

\_result=\_num1\*\_num2;

break;

case "/":

\_result=\_num1/\_num2;

break;

case "%":

\_result=\_num1%\_num2;

break;

}

document.getElementById("result").value=\_result;

}

1. 计算机票价格

<input type="text" placeholder="月份" id="month">

<select id="seat">

<option value="高等">头等舱</option>

<option>商务舱</option>

<option>经济舱</option>

</select>

<input type="text" placeholder="价钱" id="money">

<button id="ok">应付</button>

<input type="text" placeholder="应付款" id="pay">

<script>

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

var \_month=document.getElementById("month").value;

var \_seat=document.getElementById("seat").value;

var \_money=Number(document.getElementById("money").value);

var discount=0;

if(\_month>=1&&\_month<=12){

if(\_month>=5&&\_month<11){

\_discount=(\_seat==="头等舱")?0.9:((\_seat==="商务舱")?0.8:0.7);

}else{

\_discount=(\_seat==="头等舱")?0.7:((\_seat==="商务舱")?0.5:0.3);

}

document.getElementById("pay").value=\_money\*\_discount;

}else{

alert("请如正确的月份");

}

}

</script>

4、输入年月判断某月多少天

var \_days=0;

if (\_month >= 1 && \_month <= 12) {}

switch(\_month) {

case 2:

\_days = (\_year % 4 === 0 && \_year % 100 !== 0 || \_year % 400 === 0) ? 29 : 28; break;

case 4:

case 6:

case 9:

case 11:

\_days = 30;

break;

default:

\_days = 31;

}

} else {

alert("输入月份范围有误");

}

## 三、循环

1. 通常 while 与 do-while 适用于循环次数不确定的情况，通常循环次数确定的情况使用 for 循环

2、break;

使用在循环结构中，表示退出其所在的整个循环结构

continue;

只能使用在循环结构中。表示结束其所在循环的当前次循环，还会继续判断条件执行下一次循环

3、输入一个数求出它所有的和

（1）var i = 0, j = num;

while( i <= num ) { // j >= 0

console.log(i + " + " + j + " = " + num); // 0 + 3 = 3

i++;

j--;

}

（2）var i = 0, j = num;

do {

console.log(i + " + " + j + " = " + num); // 0 + 3 = 3

i++;

j--;

} while(i <= num);\*/

（3）for(var i = 0, j = num; i <= num; i++, j--) {

console.log(i + " + " + j + " = " + num);

}

4、求最小公倍数（辗转相除法）

ar m=Number(prompt("请输入第一个数："));

var n=Number(prompt("请输入另一位数："));

do{

var r=m%n;

m=n;

n=r;

}while(r!=0);

console.log(m);

5、计算50年后的薪水

var salary=100000;

var year=0;

while(year<50) {

//salary+=salary\*0.05;

salary\*=1.05;

year++;

}

console.log(salary.toFixed(4));

6、求水仙花数（有三位数其每位数上的平方之和为本身）

for (var num = 100; num < 1000;num++) {

var a=Math.floor(num/100); //百位取整

var b=Math.floor(num/10)%10; //十位

var c=num%10; //各位

if(Math.pow(a,3)+Math.pow(b,3)+Math.pow(c,3)==num){

console.log(num);

}

}

7、三角形

1. 左下直角

ar str="";

for (var i = 0; i < 9; i++) {

var row="";

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

row+="\*";

}

str+=row+"\n";

}

console.log(str);

1. 等腰三角形

var str="";

for (var i = 1; i <= 9; i++) {

var row="";

for(var j=1;j<=9-i;j++){ //" "

row+=" ";

}

for(var j=1;j<=2\*i-1;j++){ //\*

row+="\*";

}

str+=row+"\n";

}

console.log(str);

1. 直角在右下角

var str="";

for (var i = 1; i <= 9; i++) {

var row="";

for(var j=1;j<=9-i;j++){ //\*

row+=" ";

}

for(var j=1;j<=i;j++){ // " "

row+="\*";

}

str+=row+"\n";

}

console.log(str); var str = "";

for (var i = 1; i <= 9; i++) {

var currRow = "";

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

currRow+=j+"\*"+i+"="+j\*i+"\t";

}

str += currRow + "\n";

}

console.log(str)

8、计算100以内的所有整数之和

（1）var sum=0;

for(var i=1;i<=100;i++){

sum+=i;

}

console.log("100以内所有整数和：" +sum);

（2）var sum=0,i=1;

while(i<=100){

sum+=i;

i++;

}

console.log("100以内所有整数和：" +sum);

## 四、函数

1、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。

2、函数定义：

a. 函数声明

function 函数名([参数列表]) {

// 函数主体

}

b. 函数表达式

var 变量名 = function([参数列表]){

// 函数主体

}

function() {} -- 匿名函数

c. new 构造函数() -- 不推荐使用

var 变量名 = new Function(["参数列表",] "函数主体");

3、函数调用：

函数定义好之后，如果不经调用，是不会主动起作用的

a. 主动调用

函数名([参数列表]);

b. 事件驱动调用

document.getElementById("xxx").onclick = function(){};

document.getElementById("xxx").onclick = 函数名;

c. 匿名自执行函数，IIFE（立即调用函数表达式）Immediately-Invoked Function Expression (IIFE)

(function(){})();

(function(){}());

4、返回值：

return 表达式;

返回值返回到函数调用处

函数中的 return 语句最多只能返回一个值(可不写，相当于undefined)

在函数体中遇到 return 的执行，则结束函数调用，不会执行该函数return之后的

5、跳转语句：

break; -- switch、循环

continue; -- 循环

return 表达式; -- 函数

1.猜数字游戏

（1）var rand=Math.floor(Math.random()\*100);

console.log(rand);

var count=0;

while(true){

count++;

var guess=Number(prompt("请输入1~100整数"));

if(guess>rand){

alert("猜大了哟！");

}else if(guess<rand){

alert("猜小了哟");

}else{

alert("你真棒，猜中次数："+count+"次");

break;

}

}

（2）<input type="text" id="guess\_num">

<button id="btn">猜测</button>

<p id="info"></p>

<p id="count"></p>

<script>

var rand = Math.floor(Math.random() \* 100);

var count = 0; // 猜测次数

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

var num = Number(document.getElementById("guess\_num").value);

if (num < rand){

count++;

document.getElementById("info").innerHTML = "猜小了";

} else if (num > rand) {

count++;

document.getElementById("info").innerHTML = "猜大了";

} else {

count++;

document.getElementById("info").innerHTML = "猜中了";

}

document.getElementById("count").innerHTML = "猜测次数：" + count;

}

</script>

2.逆序打印

var num=3435243524;

var val=0;

var sum=1;

do{

val=num%10;

num=Math.floor(num/10);

console.log(val); //逆序

sum+=Math.pow(val,3);

}while(num!=0);

console.log(sum);//逆序的每个位数上平方之和

3. 输入年月份得到当月第一天是本年的多少天

<input type="text" id="year" placeholder="年份">

<input type="text" id="month" placeholder="月份">

<button id="calc">计算</button>

<input type="text" id="days" placeholder="天数">

<script>

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

var \_year=Number(document.getElementById("year").value);

var \_month=Number(document.getElementById("month").value);

var \_days=1;

var day=0;

for(var i=1;i<\_month;i++){

switch(i){

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12:

day=31;break;

case 2:

day=(\_year%4==0&&\_year%100!==0 ||\_year%400==0)?29:28;break;

default:

day=30;break;

}

\_days+=day;

}

document.getElementById("days").value=\_days;

}

</script>

4.输入年月，计算当月第一天是距1900-1-1多少天

在上面基础上加上

for(var i=1900;i<\_year;i++){

\_days+=i % 4 === 0 && i % 100 !== 0 || i % 400 === 0 ? 366 : 365;

}

5.输入年月第一天是星期几（1900-1-1是星期一）

在上面基础上加上

var weekday=\_days%7;

var week=["日","一","二","三","四","五","六"];

document.getElementById("week").value=week[weekday];

6．打印年月日历

// 空格个数

var \_blank = weekday;

/\* 在控制台上打印日历 \*/

var calendar = "日\t一\t二\t三\t四\t五\t六\n";

// 在打印日期前，可能打印空格

for (var i = 0; i < \_blank; i++) {

calendar += "\t";

}

switch(\_month) {

case 4:

case 6:

case 9:

case 11:

day = 30;

break;

case 2:

day =\_year % 4 === 0 && \_year % 100 !== 0 || \_year % 400 === 0 ? 29 : 28;

break;

default:

day = 31;

} //或者switch结构不写，则有在上面switch的for要写i<=\_month;且在i!==\_month时，\_days+=day;

for (var i = 1; i <= day; i++) {

calendar+=i+"\t";

if((\_blank+i)%7==0)

calendar+="\n";

}

console.log(calendar);

1. 利用函数打印日历

//判断是否为闰年

function isLeapYear(year){

return year%4===0&&year%100!==0 ||year%400===0;

}

//判断给定年月的天数

function getMonthDay(year,month){

var day=0;

switch(month){

case 4:

case 6:

case 9:

case 11:

day=30;

break;

case 2:

day=isLeapYear(year)?29:28;

break;

default:

day=31;

break;

}

return day;

}

//打印日历

function calendar(year,month){

var totalDays=1; //总天数

//该月第一天当年第多少天

for (var i = 1; i < month; i++) {

totalDays+=getMonthDay(year,i);

};

//累加到1900-1-1的总天数

for (var i = 1900; i < year; i++) {

totalDays+=isLeapYear(i)?366:365;

};

console.log("总天数："+totalDays);

//year-month-1 是星期几

var week=totalDays%7;

//打印

var str=year+"年"+month+"月日历:\n";

str += "日\t一\t二\t三\t四\t五\t六\n";

//空格(和week一样)

for (var i = 0; i < week; i++) {

str+="\t";

};

//日期

var day=getMonthDay(year,month);

for (var i = 1; i <= day; i++) {

str+=i+"\t";

if((week+i)%7===0)

str+="\n";

};

console.log(str);

}

calendar(2017,8)

//全年日历

for (var i = 0; i < 12; i++) {

calendar(2017,i);

};

## 函数作用域

1. 某函数的阶乘

（1） function fn(num){

var result=1; //这需先定义

// for (var i = 1; i <=num; i++) {

// result\*=i;

// }

for (var i = num; i>0; i--) {

result\*=i;

}

return result;

}

var re=fn(5);

console.log(re);

（2）function fn(num) {

return num===1?1:(num\*fn(num-1));

}

console.log(fn(5));

1. 斐波拉契数列

(1) function fei(num){

if(num<=2){

return 1;

}

var result,f1=1,f2=1;

for(var i=3;i<=num;i++){

result=f1+f2;

f1=f2;

f2=result;

}

return result;

}

console.log(fei(5)) ;

(2) 递归

function fei(num){

return (num<=2)?1:(fei(num-1)+fei(num-2));

}

console.log(fei(6)) ;

## 六、 数组对象

作用：用于在单个的变量中存储多个值

元素：保存的数据

下标（索引）：保存数据的编号值，从0开始，到元素总个数-1

数组长度：数组空间的大小

数组创建：

直接量：

var arr = [];

var arr = [1, 2, 3, 4];

new 构造函数：

var arr = new Array(); // 空数组

var arr = new Array(size); // size：数字，初始数据个数

var arr = new Array(1, 2, 3, 4); // 创建同时初始化各数据

说明：JS中的数组是动态的，即数组空间的大小是可以动态发生改变的。

常用API：

属性：

length：用于获取数组长度

方法：

ES3

添加：

头部：unshift(xx, xx, xx, ...)

尾部：push(xx, xx, xx, ...)

指定索引：splice(index, 0, xx, xx, xx, ......) //0个删除，可在index.前添加xx这些数据

删除：

头部：shift()

尾部：pop()

指定索引：splice(index, howmany, xx, xx, xx, ......) //index开始howmany个删除

连接：

concat() -- 连接数组，原数组不受影响，返回连接后的新数组

join() -- 连接数组中每个元素，返回连接后的字符串， 原数组不受影响(一般默认用“，”连接，里面可用“-/\*/+......或不写”

排序：

reverse() -- 反转，会影响原数组

sort() -- 排序，升序排序，会影响原数组 （只针对首数字比较）

如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。

如果想按照其他标准进行排序，就需要提供比较函数，该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 a 和 b，其返回值如下：

若 a 小于 b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。

若 a 等于 b，则返回 0。

若 a 大于 b，则返回一个大于 0 的值。

实际就是函数返回值为a-b;

截取：

slice(startIndex, endIndex) -- 从 startIndex 索引开始到 endIndex 索引结束（会包括开始索引处的元素，不包括结束索引处元素[ )） ，不传递 endIndex 表示截取到末尾。startIndex与endIndex可取负数，表示从后向前计数

转换：

toString()

toLocaleString()

valueOf()

栈（FILO/LIFO）：first in last out

数组遍历

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

// array[i]

}

for-in 循环：

for (var i in array) {

// array[i]

}//遍历对象得到i为属性名

作用：实现数组各元素遍历

实现对象各属性遍历（属性名--属性值）

对象：属性的无序集合，每个属性存放一个原始值、对象或函数

属性就是对象的特征

var obj = {

age : 18,

name : "张三",

address : "四川成都"

}

获取属性的方法

obj.name

obj["name"] 可作用于在新数组中添加属性，并给其属性值

**改变原数组的：**  
pop：删除最后一个并返回，改变原数组  
push：向数组末尾添加元素，并返回新的长度，改变原数组  
unshift：向数组开头添加元素，并返回新的长度，改变原数组  
reverse：颠倒数组顺序，改变原数组  
shift：将第一个元素删除并且返回，空即为undefined，改变原数组  
splice:splice(start,length,item)新增删除替换数组元素，返回被删除数组，无删除则不返回，改变原数组  
sort：对数组排序，改变原数组

**不改变原数组的：**  
concat：连接多个数组，返回新的数组，不改变原数组  
join：将数组中所有元素以参数作为分隔符放入一个字符串，不改变原数组  
slice：slice(start,end)，返回选定元素，不改变原数组

数组的排序（冒泡、选择）

**冒泡排序**

相邻两个元素比较大小，按排序规则实现排序操作，不满足排序规则则交换位置。重复比较多轮，每轮比较多次将该轮中最大（小）排序放置到最后。重复的轮数，每轮比较的次数是有规则的：

二重循环

外层控制比较轮数，外层循环 N - 1

内层控制每轮比较次数，内层循环 N - 1 - i，i表示外层循环变量

function bubbleSort(arr){

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

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

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

var temp=arr[j];

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

arr[j+1]=temp;

}

}

};

return arr;

}

var arr2=[1,4,8,2,5,3,9]

bubbleSort(arr2);

console.log(arr2);

**选择排序**

把每一个数都与第一个数比较，如果小于第一个数，就把它们交换位置；这样一轮下来，最小的数就排到了最前面；重复n-1轮，就实现了选择排序

function selectSort(arr){

var k;

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

k=i;

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

if(arr[j]<arr[k]){

temp=arr[k];

arr[k]=arr[j];

arr[j]=temp;

}

}

}

return arr;

}

var arr1=[1,4,8,2,5,3,9,10]

selectSort(arr1);

console.log(arr1.join("-"));

**插入排序**

将n个元素的数列分为已有序和无序两个部分。

数列：{a1，a2，a3，a4，…，an}

将该数列的第一元素视为有序数列，后面都视为无序数列：

{{a1}，{a2，a3，a4，…，an}}

将无序数列中的元素插入到有序数列的对应位置，插入前通过比大小的方式找到其在有序数列中的对应位置。

function insertSort(arr){

var temp; //temp变量用于临时存储待插入元素

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

temp = arr[i];

//从前往后查找插入位置

for(var j=i; j>0&&arr[j-1]>temp; j--){

arr[j]=arr[j-1]; //将大于temp的arr[j]元素后移

}

arr[j]=temp;

}

}

var testArr = [35,22,1,56,88,25];

insertSort(testArr);

console.log(testArr);

**希尔排序**

希尔排序是插入排序的一种更高效率的实现。它与插入排序的不同之处在于，它会优先比较距离较远的元素。希尔排序的核心在于间隔序列的设定。既可以提前设定好间隔序列，也可以动态的定义间隔序列。动态定义间隔序列的算法是《算法（第4版》的合著者Robert Sedgewick提出的。在这里，我就使用了这种方法。

function shellSort(arr) {

var len = arr.length,

temp,

gap = 1;

while(gap < len/3) { //动态定义间隔序列

gap =gap\*3+1;

}

for (gap; gap> 0; gap = Math.floor(gap/3)) {

for (var i = gap; i < len; i++) {

temp = arr[i];

for (var j = i-gap; j > 0 && arr[j]> temp; j-=gap) {

arr[j+gap] = arr[j];

}

arr[j+gap] = temp;

}

}

return arr;

}

**快速排序**

（1）在数据集之中，选择一个元素作为"基准"（pivot）。

　　（2）所有小于"基准"的元素，都移到"基准"的左边；所有大于"基准"的元素，都移到"基准"的右边。

　　（3）对"基准"左边和右边的两个子集，不断重复第一步和第二步，直到所有子集只剩下一个元素为止。

int quicksort(vector<int> &v, int left, int right){

if(left < right){

int key = v[left];

int low = left;

int high = right;

while(low < high){

while(low < high && v[high] > key){

high--;

}

v[low] = v[high];

while(low < high && v[low] < key){

low++;

}

v[high] = v[low];

}

v[low] = key;

quicksort(v,left,low-1);

quicksort(v,low+1,right);

}

}

**字符编码：**

在人与计算机之间进行数据转换的工具

字符编码集 -- 表格

ASCII

GBK

GB2312

BIG5

unicode -- 国际化编码方式

UTF-8

UTF-16

**数据类型**

基本类型

引用类型

**函数的值传递和引用传递**

JS 中函数的参数传递只有值传递的方式（即将栈内存中的数据副本拷贝）

值传递过程中，如果传递的是基本类型的参数，则函数主体中对参数的修改，不会影响到传递之前的参数；如果传递的是引用类型的参数，则函数主体中对参数的修改，可能会影响到传递之前的参数（在堆内数组改变相当于开辟另一个空间，则完全改变栈内地址，则本身栈地址指向的堆地址不一样，改变的栈地址改变使用完后释放地址，则没改变本身栈引用）

**堆和栈及索引思维**

栈内存中主要存放基本类型变量的值，存放引用类型变量的引用(指针)

堆内存中主要存放的是对象(如通过 new 所创建的对象)

(若是对象则栈内地址指向堆内)

**JSON表示对象的方法**

在JSON中，有两种结构：对象和数组。

1. 一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间使用“,”（逗号）分隔。 名称用引号括起来；值如果是字符串则必须用括号，数值型则不需要。

2. 数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。也可[{},{}....]

## 数组新增API

ES5 数组新增API

遍历

arrayObject.forEach(callback)

遍历数组中每个元素，在遍历到元素时，执行 callback 函数

callback 格式：function(currentValue, index, array){}

currentValue 表示当前遍历到的元素

index 表示当前遍历到元素在数组中的下标

array 表示调用 forEach() 方法的数组

arrayObject.map(callback)

返回一个新数组，新数组中元素为 callback 函数的返回值

arrayObject.every(callback)

arrayObject.some(callback)

arrayObject.filter(callback)

查找

arrayObject.indexOf(searchElement[, fromIndex = 0])

查找 searchElement 元素在原数组中第一次出现的索引，如果找到了，则

返回大于等于0的值，找不到返回 -1。

fromIndex 是可选参数，表示从指定fromIndex索引开始向后查找第一次出

现的索引

lastIndexOf()

归并

reduce()方法对累加器和数组中的每个元素 (从左到右)应用一个函数，将其减少为

单个值。

array.reduce(function(accumulator, currentValue, currentIndex, array),

initialValue)如果没有设置初始值，则将数组中的第一个元素作为初始值。空数组调用reduce时没有

设置初始值将会报错。

reduceRight()与array.reduce执行方向相反

字符串 String

创建：

var str = "abc"; //typeOf后是string类型

var str = new String("abc"); //typeOf后是object类型

了解：基本类型没有属性方法的调用，之所以基本类型也能调用到属性方法，是因

为 JS 做隐式转换将基本类型包装为引用类型（装箱） -- 包装对象

常用API：

属性：

length -- 获取字符串长度（包括空格）

方法：

charAt() 方法从一个字符串中返回指定的字符,返回的是字符子串。

charCodeAt() 方法返回0到65535之间的整数,返回的是位于指定位置的字

符的编码

String.fromCharCode() 方法返回使用指定的Unicode值序列创建的字符

串。String调用

indexOf() 方法返回调用 String 对象中第一次出现的指定值的索引，

开始在 fromIndex进行搜索。

lastIndexOf()

match()

replace()

search()

slice(startIndex, endIndex)

substring(start, end)

split() 用于把一个字符串分割成字符串数组。

toUpperCase()

toLowerCase()

trim() 去掉前后的空白（用于输入账号，获取字符）

1. map()函数

function map(array){

for (var i = 0,len=array.length; i < len; i++) {

array[i]\*=1.3;

};

}

arr=[3,5,6,8,1,2];

map(arr);

console.log(arr);

2.查找 value 值在 array 数组中是否存在，true:存在 false:不存在

function has(array,value){

for(var i in array) {

if(array[i]===value)

return true;

}

return false;

}

1. 去掉数组中重复元素(新数组中若在原数组中不存在则添加）

（1）function has(array,value){

for(var i in array) {

if(array[i]===value)

return true;

}

return false;

}

function norepeat(array){

newArr=[];

for(var i=0,len=array.length;i<len;i++){

if(!has(newArr,array[i]))

newArr.push(array[i]);

}

return newArr;

}

（2） var arr = [1, 3, 5,12 ,3,7, 9, 12];

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

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

if (arr[i]==arr[j]) {

arr.splice(j,1);

};

};

};

console.log(arr);

1. 查找 value 值在 array 数组中出现的第一个下标，如果存在，则返回大于等于0的值，不存在则返回 -1

function indexOf(value,array){

for(var i in array){

if(array[i]===value)

return i;

}

return -1;

}

1. 按照已有排序顺序插入新元素

（1）var arr = [1, 3, 5, 7, 9, 12];

console.log("前：" + arr.join());

var num=Number(prompt("请输入一个1-100的整数"));

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

if(arr[i]<num&&arr[i+1]>num){

arr.splice(i+1,0,num);

}

};

console.log(arr.join());

（2）var arr = [1, 3, 5, 7, 9, 12];

var insert=Number(prompt("请输入想要插入的值"));

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

if(arr[i-1]>insert){

arr[i]=arr[i-1];

}else{

break;

}

}

arr[i] = insert;//插入

//arr.splice(i,0,insert);//插入

console.log(arr.join());

1. 生成扑克牌并洗牌

var type = ["红桃", "黑桃", "方块", "梅花"],

point = [2, 3, 4, 5, 6, 7, 8, 9, 10, "J", "Q", "K", "A"],

cards = ["大王", "小王"];

// 生成

for(var i in type){

for(var j in point){

cards.push(type[i]+point[j]);

}

}

// 洗牌

for(var i=0,len=cards.length;i<len;i++){

var randIndex=Math.floor(Math.random()\*len);

var temp=cards[i];

cards[i]=cards[randIndex];

cards[randIndex]=temp;

}

// cards.sort(function(){return Math.random()-0.5;}); //乱序推荐

console.log(cards);

1. array.map()使用

var array = [3, 1, 7, 4, 6, 5, 9];

var newArray = array.map(function(curr){

return curr \* 1.3;

});

console.log(newArray);

Array.reduce()使用

var scores = [65, 73, 62, 69, 99, 62, 88, 66, 62, 65, 82];

var sum = scores.reduce(function(a, c, i, arr){

console.log(a, c, i, arr);

return a + c;

});

console.log(sum);//所有数之和

1. string的replace()方法

var str = "ssadf skjfd oqiweaujr oqireawu 09743132 jkdsajfo idaurjfq ;iewuroq herf:OIDhf oiw qehf rlkd sjfc";

//var result = str.replace("a", "打广告"); //替代第一个

var result = str.replace(/a/g, "打广告"); //替代所有a

console.log(result);

9.字符串提取

var str = "username=xiaohong&password=abc&age=15&address=成都&phone=13199887766";

var stu = {};

var values = str.split("&");

values.forEach(function(curr){

var val = curr.split("=");

var key = val.shift();//改变原数组了，key为第一个值，原数组删除了它

stu[key] = val.shift(); //或者pop

});

console.log(stu);

console.log("用户名：" + stu.username + "，密码：" + stu.password);

## 日期

Date -- 日期时间

创建对象：

var date = new Date(); // 创建对象，表示当前系统时间

var date = new Date(value); // value 代表一个毫秒值，该毫秒值是距 1970-1-1 0:0:0 以来的毫秒值

var date = new Date(dateString); // dateString 表示某日期的字符串格式，标准：yyyy-MM-ddTHH:mm:ss

var date = new Date(year, month[, date[, hours[, minutes[, seconds[, milliseconds]]]]]);

常用API：

方法：

获取：

getFullYear() -- 年

getMonth() -- 月，返回 0~11 之间的数字

getDate() -- 日

getHours() -- 时

getMinutes() -- 分

getSeconds() -- 秒

getMillseconds() -- 获取1m以内的毫秒

getDay() -- 星期，返回0~6之间的数字

getTime() -- 获取距 1970-1-1 0:0:0 以来的总毫秒数

设置：

setFullYear() -- 年

setMonth() -- 月，合理范围 0~11 之间的数字

setDate() -- 日

setHours() -- 时

setMinutes() -- 分

setSeconds() -- 秒

setMillseconds() -- 毫秒

setTime()

转换：

toString()

toUTCString()

toLocaleString()

toTimeString()

parse() -- 将日期时间字符串内容解析为距 1970-1-1 0:0:0 以来的毫秒值

日期时间处理时，通常会将日期时间对象转换为毫秒值后来运算(距1970-1-1 0:0:0以来的毫秒值)：

a. var date = new Date();

date.getTime();

b. var str = "dateString";

Date.parse(str);

c. var date = new Date()

date.valueOf()

d. var date = new Date();

var mill = +date;

e. Date.now() -- ES5新增

Math

在 JS 中不需要通过 new Math() 方式创建对象，Math 对象没有构造函数

使用 Math 对象中属性方法时，直接使用 Math 来调用

Math.min(),Math.floor()....

Number

toFixed() -- 将数字转换为字符串，保留小数点后指定位数小数

1. 统计数组中各元素出现的次数

（1）利用对象

var arr = [1, 3, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 1, 3 ,2, 4, 5];

var count = {};

arr.forEach(function(curr){

// 判断 count 对象中是否存在当前遍历到元素对应的属性

var prop=count[curr];

if(prop){

count[curr]++;

}else{

count[curr]=1;

}

})

console.log(count);

}

1. 利用数组

var arr = [1, 3, 1, 3, 4, 5, 6, 1, 2, 3, 4, 6, 1, 3 ,2, 4, 5];

var norepeat=[];

var count = [];

arr.forEach(function(curr){

var index=inArray(curr,norepeat);

//var index=norepeat.indexOf(curr);

if(index==-1){ // noRepeat 中不存在

norepeat.push(curr);

count.push(1);

}else{

count[index]++;

}

});

//输出

norepeat.forEach(function(curr,index){

console.log(curr+":"+count[index]);

})

1. 验证码

Html:

<input type="button" id="input\_code" >

<input type="text" id="code">

Script:

function genCode(length){ //产生验证码

var code="";

if (typeof length === "undefined")

length = 4;

while(code.length<length){

var rand=Math.floor(Math.random()\*75)+48;

if (rand>=48&&rand<=57

||rand>=65&&rand<=90

||rand>=97&&rand<=122)

code+=String.fromCharCode(rand);

}

return code;

}

document.getElementById("input\_code").value = genCode(6);

document.getElementById("code").onblur = function(){

var \_code = document.getElementById("code").value;

var \_input=document.getElementById("input\_code").value;

if (\_input.toUpperCase() == \_code.toUpperCase()) {

alert("正确")

} else {

alert("输入有误")

}

document.getElementById("input\_code").value = genCode(6);//再次调用

}

1. 驼峰命名法

（1）

var str="Hello xiaoming WELCOME to my HoMe";

var arr=str.split(" ");//分成数组

var arr1=[];

var result="";

//把所有变小写

arr.forEach(function(currvalue,index){

arr1[index]=currvalue.toLowerCase();

})

//处理数组

arr1.forEach(function(currvalue,index){

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

if(index>0&&i==0){//第二个单词开始，首字母变大写

result+=currvalue[0].toUpperCase();

continue; //变成大写后跳到下一个单词

}

result+=currvalue[i];

}

})

console.log(result);

（2）

var str="Hello xiaoming WELCOME to my HoMe";

var arr=str.split(" ");

arr[0]=arr[0].toLowerCase();//第一个单词小写

for(var i=1,len=arr.length;i<len;i++){ //首字母大写其它小写

arr[i]=arr[i].charAt(0).toUpperCase()+arr[i].slice(1).toLowerCase();

}

str=arr.join("");

console.log(str);

1. 字符串中字符出现的个数

var str = "adfadsfoiujewqrjes0u1alksdjflkajdsfoijarekwqeuf0XZjfasiu";

var count={};

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

var pro=str.charAt(i);

if(count[pro]){

count[pro]++;

}else{

count[pro]=1;

}

}

for(var attr in count ) {

console.log(attr + " : " + count[attr]);

}

1. 查找字符串个某个字符串出现的位置

var str="abcdadfabcdaadabcdadsafaabcdaadfaabcdadafaabcdadsfafds";

var subStr="abcd";

var index=-1;

do{

index=str.indexOf(subStr,index+1);

if (index===-1)

break;

console.log(index);

}while(1)

1. 格式化日期显示

Html:

<input type="text" id="year" value="2017" size="2">

<input type="text" id="month" value="8" size="2">

<input type="text" id="date" value="9" size="2">

<button id="format">格式化</button>

<input type="text" id="result">

Script:

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

var \_year = document.getElementById("year").value,

\_month = document.getElementById("month").value,

\_date = document.getElementById("date").value;

if(\_month<10)

\_month="0"+\_month;

\_date=("0"+\_date).slice(-2);

var \_result=\_year+"-"+\_month+"-"+\_date;

document.getElementById("result").value=\_result;

}

1. 计算输入年月的第一天是星期几

Html:

<input type="text" id="year" value="2017">年

<input type="text" id="month" value="8">月

<input type="text" value="1">日

<button id="calc">是</button>

星期<input type="text" id="result">

Script:

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

var \_year = Number(document.getElementById("year").value),

\_month = Number(document.getElementById("month").value);

var date = new Date(\_year, \_month - 1);

var weekday = date.getDay();

var \_result;

// \_result = ["天", "一", "二", "三", "四", "五", "六"][weekday];

\_result = "天一二三四五六".charAt(weekday);

document.getElementById("result").value = \_result;

}

1. 封装格式化日期

var datetime = new Date();

function format(datetime) {

var year = datetime.getFullYear(),

month = ("0" + (datetime.getMonth() + 1)).slice(-2),

date = ("0" + datetime.getDate()).slice(-2),

hour = ("0" + datetime.getHours()).slice(-2),

min = ("0" + datetime.getMinutes()).slice(-2),

sec = ("0" + datetime.getSeconds()).slice(-2);

var \_result = year + "-"+ month +"-"+ date +" "+ hour +":"+ min +":" + sec;

return \_result;

}

1. 计算到计时

Html:

<input type="text" id="deadline" value="2017-08-24 15:10:00">

<button id="calc">倒计时</button>

<input type="text" id="result" value="？天？时？分？秒">

Script:

var timer;

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

var \_deadTime=document.getElementById("deadline").value;

/\*var endtime = Date.parse(\_deadline);

var starttime = Date.now(); // 起始时间毫秒值

var diff = endtime - starttime; // 毫秒差值\*/

clearInterval(timer);// 先取消前一次计时器任务

timer=setInterval(function(){

var diff=new Date(\_deadTime)-new Date();

// 将毫秒差换算为总秒数

var seconds=Math.ceil(diff/1000);

var second=seconds%60;

var minute=Math.floor(seconds/60)%60;

var hour=Math.floor(seconds/(60\*60))%24;

var day=Math.floor(seconds/(24\*60\*60));

var \_result = day + "天"+ hour +"时"+ minute +"分"+ second +"秒";

document.getElementById("result").value = \_result;

if(seconds==0)

clearInterval(timer);

},1000)

}

两小时计时

1. var \_deadTime=date.setTime(date.getTime()+2\*60\*60\*1000); //算毫秒

（2） var timer;

clearInterval(timer);// 先取消前一次计时器任务

var seconds=2\*60\*60;

timer=setInterval(function(){

seconds--; //直接秒减

if(seconds==0)

clearInterval(timer);

var second=seconds%60;

var minute=Math.floor(seconds/60)%60;

var hour=Math.floor(seconds/(60\*60))%24;

var \_result = hour +"时"+ minute +"分"+ second +"秒";

document.getElementById("result").value = \_result;

},1000);

## BOM/DOM

**JSON**：(ES5中新增)avaScript Object Notation

作用：存储、交换文本格式的数据

JSON.parse() -- 将字符串内容解析为对应的 JS 值

JSON.stringify() -- 能够将 JS 值转换成对应格式的字符串内容

**BOM**

Browser Object Model浏览器对象模型

作用：使 JavaScript 有能力与浏览器“对话”

Window 对象 -- 代表浏览器打开的窗口

window -- 对 Window 对象的引用

location 对象 -- 代表URL（网址）

URL格式 -- 协议://主机名:端口/资源路径?查询字符串#hash

http 默认端口 80

https 默认端口 443

https://www.baidu.com:443/home/news/data/newspage?nid=570627856227622358&n\_type=0&p\_from=1&dtype=-1

属性：

hash -- #hash

host -- 主机名:端口

hostname -- 主机名

href -- 完整的 URL

pathname -- /资源路径

port -- 端口

protocol -- 协议:

search -- ?查询字符串

方法：

reload() -- 重新加载

assign() -- 相当于 location.href = "" 跳转到新页面，会向历史记录中添加访问历史

replace() -- 拿新页面替换当前页面，不会添加访问历史

history 历史记录

length -- 历史记录数

back() -- 后退

forward() -- 前进

history.go(number|URL) -- 前进/后退

navigator 浏览器本身相关信息

userAgent -- 用户代理

document 文档

DOM

name

status

alert()

confirm()

prompt()

open()

close()

计时器

var id = setTimeout(fn, time) -- 指定 time 时间到达，执行一次 fn 函数

var id = setInterval(fn, time) -- 周期性执行计时函数

clearTimeout(id)

clearInterval(id)

控制台看间隔时间

// 启动计时器

console.time('testForEach');

// (写一些测试用代码)

// 停止计时，输出时间

console.timeEnd('testForEach');

Event Loop (事件轮询)

JS 是单线程应用程序

注意：计时器并不一定是在计时时间到达就执行计时函数，会受 event loop 机制影响，可能延迟执行计时函数。

**DOM**

Document Object Model 文档对象模型

作用：在 JS 中能操作（添加、删除、修改、查询、克隆....）文档中各部分内容

document -- 指代页面中的整个文档内容

查询元素节点：

document.getElementById() -- 根据元素的 id 查找节点

如果找到，则返回找到的DOM对象，未找到，返回 null

document.getElementsByTagName() -- 根据元素的标签名查找节点

返回类数组对象（HTMLCollection）

document.getElementsByClassName() -- 根据使用的CSS类名查找节点，ES5中新增

返回集合

document.getElementsByName() --根据元素 name 属性查找节点（少用）

getElementsByTagName()、getElementsByClassName() 也可使用具体的某个 DOM 元素对象来调用，表示在该元素的后代中查找满足条件的节点。

getElementsByClassName 兼容问题？

节点筛选过滤：

element.children -- 孩子元素节点

element.childNodes -- 孩子节点（包括空白文本节点）

element.firstChild -- 第一个孩子（可能是空白文本节点）

element.lastChild

element.firstElementChild

node.parentNode -- 父节点

node.previousSibling -- 前一个兄弟节点

node.nextSibling -- 后一个兄弟节点

节点属性：

node.nodeName -- 节点名

文本节点，固定为 #text 取值

元素节点，返回标签名大写字母内容

node.nodeType -- 节点类型

返回数字

1 -- 元素节点

2 -- 属性节点

3 -- 文本节点

node.nodeValue -- 节点值

文本节点，返回文本内容值

元素节点，固定为 null

1. getElementsByClassName 兼容问题？封装函数

//className : 类名

function getElementsByClassName(className,context){

context=context||document;

if(context.getElementsByClassName){ //支持情况

return context.getElementsByClassName(className);

}else{

var result=[]; // 保存查找到结果的数组

var allElements=context.getElementsByTagName("\*");// 遍历所有元素

for (var i = 0,len=allElements.length; i < len; i++) {

// 判断当前遍历到的元素类名中是否存在待查找的类名

var classNames=allElements[i].className.split(" ");//把一组含有多个类名以空格分割成数组

if(inArray(className,classNames)!=-1)

result.push(allElements[i]);

};

}

return result;

}

//判断给定的某个值是否在数组中存在，存在则返回其下标，不存在返回-1 / indexOf第一次出现的位置,解决兼容

//value : 待查找判断的值

// array : 数组

function inArray(value,array){

if(Array.prototype.indexOf) { //如果支持indexOf函数的话

return array.indexOf(value);

}else{

for (var i = 0,len=array.length; i < len; i++) {

if(array[i]==value)

return i;

};

return -1;

}

}

1. 请计算每个学生总成绩，并按总成绩排名。统计各单科成绩第一名，输出其成绩与学号。

var stus = [

{

no : 1,

chinese : 95,

math : 99,

english : 99

},

{

no : 2,

chinese : 105,

math : 90,

english : 109

},

{

no : 3,

chinese : 85,

math : 129,

english : 111

}

];

stus.forEach(function(stu){

var sum=stu.chinese+stu.math+stu.english;

stu.sum=sum;

})

stus.sort(function(a,b){

return b.sum-a.sum; //递减

})

console.log(JSON.stringify(stus));

console.log("总分第一名学号:"+stus[0].no);

3.封装函数，实现根据id、标签、类名选择器selector查找元素

//context : 查询上下文环境，可选参数，表示一个具体的DOM对象

function $(selector,context){

context=context||document;

if(selector.charAt(0)=="#"){ // id

return context.getElementById(selector.slice(1));

//取符号之后的字符

}else if(selector.charAt(0)=="."){ // className

return getElementsByClassName(selector.slice(1),context); //是调用的函数

}else{

return context.getElementsByTagName(selector); //标签选择器

}

}

## 节点和样式设置

element.className -- 获取/设置元素对应HTML标签中的 class 属性

**添加节点：**

a. 元素节点

创建元素节点：

var \_element = document.createElement(tagName)

设置节点属性：

\_element.setAttribute(attrName, attrValue);

\_element.attrName = attrValue;

获取节点属性：

var attrValue = \_element.getAttribute(attrName);//在含有attrName才能获取

var attrValue = \_element.attrName;

【注】checked、selected、disabled：

要获取状态值，直接使用 (元素对象".") 来调用，选中后获取到true，没选中选中false

为节点内部添加文本：

\_element.innerHTML = "HTML文本内容（可包含HTML标签）";

\_element.innerText = "纯文本";

加到页面中指定位置：

\_parentNode.appendChild(\_element);

\_parentNode.insertBefore(\_element, \_existNode)

b. 文本节点

var \_txt = document.createTextNode(text)

**删除节点**：

\_parentNode.removeChild(\_child)

**克隆(复制)节点**：

\_node.cloneNode(true|false)

true是把子节点一起克隆

false只复制本身

**CSS-OM**

获取CSS

element.getAttribute("style")--获取行内样式 CSS样式声明对象，得到typeof为string

element.style -- 只能获取元素style属性中的CSS样式

getComputedStyle(element) -- 获取元素的计算 CSS样式声明 对象（最终起作用的那些样式包括一些默认样式）

IE9之前浏览器不支持，替换方案：

element.currentStyle 返回的是元素当前应用的最终CSS属性值（包括外链CSS文件，页面中嵌入的<style>属性等）

设置CSS

element.style.attrName = "attrValue"; attrName如果是复合的，则用驼峰命名法eg:borderLeftWidth

element.style["attrName"] = "attrValue";

1.// 封装函数，获取/设置CSS样式

// 使用 css(element, "width") -- 获取 element元素的 width 属性值

// 使用 css(element, "width", "100px") -- 设置 element 元素的 width 属性值为 100px(只设置一个属性)

// 使用 css(element, {width:"100px", height:"100px"}) -- 设置 element 元素的多个CSS属性，属性名与属性值使用对象表示

// 参数：

// element : DOM元素对象

// attr : CSS属性名

// value : CSS属性值，可选，不传表示获取CSS，传递表示设置CSS

// 返回值：

// 查找到的CSS属性值

function css(element,attr,value){

//获取

if(typeof attr==="string"&&value=="undefined"){

return element.getComputedStyle?element.getComputedStyle(element)[attr]:element.currentStyle[attr];

}else{

//设置一个值

if(typeof attr==="string"&&value){

return element.style[attr]=value;

}else if(typeof attr==="object"){ //设置多个值

for(var prop in attr){

element.style[prop]=attr[prop];

}

}

}

}

1. 简单的移出节点功能

html：

table border="1" cellspacing="0">

<tr>

<td>3</td>

<td>3</td>

<td><a href="javascript:void(0);">删除</a></td>

</tr>

<tr>

<td>4</td>

<td>4</td>

<td><a href="javascript:void(0);">删除</a></td>

</tr>

</table>

script:

var links = $("a");

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

links[i].onclick = function(){

// this 在事件函数中使用时，指代事件源对象

var \_tr = this.parentNode.parentNode;

\_tr.parentNode.removeChild(\_tr);

}

}

4.简单模拟购物车（在页面上操作）克隆节点和删除节点

html：

<div class="title">

<div>编号</div>

<div>商品名称</div>

<div>单价</div>

<div>数量</div>

<div>小计</div>

<div>操作</div>

</div>

<div class="main">

</div>

<div class="template" style="display: none;">

<div>1</div>

<div>小米手机</div>

<div>1999</div>

<div><span class="minus">-</span><input type="text" value="1" size="1"><span class="add">+</span></div>

<div>1999</div>

<div><a href="javascript:void(0);">删除</a></div>

</div>

<style type="text/css">

.title, .main, .row {clear: both;}

.title div, .row div{width: 100px; height: 30px;line-height: 30px; float: left; border:1px solid; text-align: center;}

.minus, .add {display:inline-block; width:20px; height:20px; line-height: 20px; text-align: center; border:1px solid; cursor: pointer;}

</style>

script：

// 先模拟部分购物车的数据

var products = [

{

id : 1,

name : "小米手机",

price : 1999,

amount : 1

},

{

id : 2,

name : "小米空调",

price : 4399,

amount : 3

},

{

id : 3,

name : "华为手机",

price : 2999,

amount : 2

}

];

// 页面加载，则显示已有购物车商品数据

products.forEach(function(product){

// 找到模板，克隆

var \_node = $(".template")[0].cloneNode(true);

// 显示节点

\_node.style.display = "block";

// 修改使用的样式类名

\_node.className = "row";

// 追加到 div.main 盒子中

$(".main")[0].appendChild(\_node);

// 处理当前节点数据

var children = \_node.children;

children[0].innerHTML = product.id;

children[1].innerHTML = product.name;

children[2].innerHTML = product.price;

children[3].children[1].value = product.amount;

children[4].innerHTML = product.amount \* product.price;

});

// 删除

var links = $("a", $(".main")[0]);

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

links[i].onclick = function(){

var \_row = this.parentNode.parentNode;

\_row.parentNode.removeChild(\_row);

}

}

// 加

var adds = $(".add", $(".main")[0]);

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

adds[i].onclick = function(){

// 修改数量

var \_amount = Number(this.previousSibling.value);

\_amount++;

this.previousSibling.value = \_amount;

// 获取当前商品的单价

var \_price = Number(this.parentNode.previousElementSibling.innerHTML)

// 计算小计

var \_sub = \_amount \* \_price;

// 显示小计

this.parentNode.nextElementSibling.innerHTML = \_sub;

}

}

## 十一、事件

**event 事件**

事件句柄（关联事件源与事件响应程序）：

onblur

onchange

onclick

onfocus

onkeydown

onkeyup

onkeypress

onload

onmousedown

onmouseup

onmouseover

onmouseout

onmouseenter

onmouseleave

onmousemove

onresize

onsubmit

onscroll -- 滚动

oncontextmenu -- 右键事件

**Event对象**

代表事件的状态

**获取 Event 对象**：

标准：事件函数的第一个参数

非标准：window.event

兼容性写法：var event = event || window.event;

**鼠标/键盘常用属性：**

event.clientX / event.clientY -- 可见视图窗口中坐标 （光标）

event.pageX / event.pageY -- 文档中的坐标（有兼容问题）（光标）

event.clientX + (document.documentElement.scrollLeft || document.body.scrollLeft);

event.clientY + (document.documentElement.scrollTop || document.body.scrollTop);

event.offsetX / event.offsetY -- 在事件源元素坐标系中的坐标位置（光标）

event.which -- 鼠标按键 1-左 2-中 3-右

event.button -- 鼠标按键 0-左 1-中 2-右 (IE低版本不一样)

event.keyCode -- 获取键盘按键，在 keydown 中获取的是虚拟键盘码，在 keypress 中获取的是字符编码

event.which -- 获取键盘按键

获取键盘按键兼容写法：var code = event.which || event.keyCode;

event.ctrlKey

event.shiftKey

event.altKey

**事件流**：

a. 事件捕获阶段

b. 处于目标阶段

c. 事件冒泡阶段

事件捕获：

在 DOM 树结构中，从 document 节点开始，依次向下捕获性处理事件，直到事件源元素

事件冒泡：

在 DOM 树结构中，在事件源元素事件处理程序执行完毕后，会将事件沿 DOM 树结构向上层（祖先）传播，将传播的过程称作是事件冒泡。

目的：集中式事件处理

所有浏览器中默认的事件处理方式都是采用事件冒泡的方式处理

利用事件冒泡处理事件，称为 事件委派(父级中子点击对象)

可通常 target 或 srcElement 获取事件源元素。

event.target -- 获取最初触发事件的事件源元素，不会在冒泡中改变（标准）

event.srcElement -- 非标准

event.target || event.srcElement

event.currentTarget -- 事件处理过程中，当前事件源

**阻止事件冒泡**：

标准：event.stopPropagation()

非标准：event.cancelBubble = true;

**事件捕获：**

element.addEventListener

target.addEventListener(type, callback[, useCapture]);

type : 事件类型（字符串）

callback : 事件响应程序（函数）

useCapture : 可选参数，表示是否使用事件捕获来处理。默认为 false（事件冒泡），设置为 true 表示事件捕获

**注册事件监听的方式：**

a. addEventListener()

b. 事件句柄

c. HTML标签中的事件属性（onclick="alert("fdf)+"）

addEventListener() 在 IE9 之前浏览器中不能使用，应使用 attachEvent() 方法替代

target.attachEvent(type, callback);()

addEventListener 与 attachEvent 区别：

a. 参数：第一个参数都表示事件类型，但 addEventListener 不需要 on 前缀，attachEvent 需要添加 on 前缀

b. 在添加事件监听时，addEventListener 是先添加则先执行，而 attachEvent 是先添加后执行

c. attachEvent 没有第三个参数，只支持事件冒泡

解除事件监听(和注册相似)

removeEventListener

detachEvent

（解除的和原函数必须要一致，可用一个函数定义再调用，而不能只是写一样的函数）

**阻止默认行为**：

标准：event.preventDefault()

非标准：event.returnValue = false;

1.解决 addEventListener 与 attachEvent 兼容问题的函数封装

（1） //注册事件

function addEvent(element,type,callback)

if(element.addEventListener){

if(type.charAt(0)==="on")//if (type.indexOf("on") === 0)

type=type.slice(2);

element.addEventListener(type,callback,false);

}else{

if(type.charAt(0)!=="on")

type="on"+type;

element.attachEvent(type,callback);

}

}

1. //解除事件

function removeEvent(element, type, callback) {

if (element.removeEventListener) { // 支持使用 removeEventListener

if (type.indexOf("on") === 0)

type = type.slice(2);

element.removeEventListener(type, callback, false);

} else { // 不支持使用 removeEventListener

if (type.indexOf("on") !== 0)

type = "on" + type;

element.detachEvent(type, callback);

}

}

1. pageX和pageY兼容

function page(e){

var \_x=e.pageX||

(e.clientX+document.documentElement.scrollLeft||document.body.scrollLeft);

var \_y=e.pageY||

(e.clientY+document.documentElement.scrollTop||document.body.scrollTop);

return {

x:\_x,

y:\_y

}

}

1. 模拟聊天界面

html：

<div id="info" style="width:300px; height:200px; border:1px solid;">

<div>xxx 说： xxxxx</div>

<div>xxx 说： xxxxx</div>

<div>xxx 说： xxxxx</div>

<div>xxx 说： xxxxx</div>

</div>

<input type="text" id="nickname" value="小明"><br>

<textarea id="message" cols="40" rows="15"></textarea>

<button id="send">发送</button>

script：

$("#send").onclick=sendMsg;

function sendMsg(){

var \_nickname=$("#nickname").value;

var \_message=$("#message").value;

$("#info").innerHTML+= "<div>"+ \_nickname +" 说："+ \_message +"</div>";

$("#message").value="";

}

$("#message").onkeydown=function(e){

e=e||event;

var code=e.which||e.keyCode; //获取键值

if(e.ctrlKey&&code==13){

sendMsg();

}

}

1. 省份的选择

css：

#select {width: 100px;height: 50px;border:1px solid; padding:10px; line-height: 50px; cursor: default;}

\* {margin: 0;padding: 0;}

#provinces {list-style: none; width:150px; height: 70px; border:1px solid; display: none;}

#provinces li {width: 40px;height: 30px;border:1px solid; line-height: 30px; text-align: center; float:left;}

html：

<div id="select">请选择省份</div>

<ul id="provinces">

<li>四川</li>

<li>北京</li>

<li>上海</li>

<li>广东</li>

<li>天津</li>

<li>陕西</li>

</ul>

script:

$("#select").onclick = function(e){

e = e || event;

// 阻止文档事件冒泡而隐藏了

e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;

// $("#provinces").style.display = "block";

// css($("#provinces"), "display", "block");

show($("#provinces"));

}

// 事件委派的方式处理 li 的点击事件

$("#provinces").onclick = function(e){

e = e || event;

var src = e.target || e.srcElement;

// console.log(this);

/\*if (src === this)

console.log("ul")

else

console.log("li")\*/

if (src.nodeName === "LI") {

$("#select").innerHTML = src.innerHTML;

}

}

// document 绑定点击事件

document.onclick = function(){

hide($("#provinces"));

}

## 十二、拖拽

默认行为：

超级链接跳转

点击提交按钮提交表单

页面内容禁止拖拽

右键菜单

阻止默认行为：

标准：event.preventDefault()

非标准：event.returnValue = false;

在事件函数中 return false; 也可以阻止默认行为

拖拽：

事件：

mousedown -- 待拖拽的目标元素

mousemove -- 文档

mouseup -- 文档

获取元素的计算宽度/高度(元素要显示占位)

element.clientWidth -- 边框以内的宽度值，去掉滚动条的宽度

clientHeight

element.offsetWidth -- 边框及内部的总宽度值

offsetHeight

获取窗口的宽度

document.documentElement.clientWidth || document.body.clientWidth;

element.offsetTop -- element元素距其有定位父元素上边沿间距

element.offsetLeft -- element元素距其有定位父元素左边沿间距

element.offsetParent -- element元素有定位的父元素

mouseover 与 mouseenter 事件区别：

mouseover 事件有事件冒泡现象，即从后代元素上经过，可能重复多次触发 mouseover 事件

mouseenter 事件没有事件冒泡，从后代元素上经过，不会重复触发事件

1.自定义右键菜单？

css:

\* {margin: 0;padding: 0;}

#contextmenu {list-style: none; width:100px; height:100px; position: absolute; top:0; left:0; border: 1px solid; display: none;}

#contextmenu li {height: 24px; border-bottom: 1px solid; text-align: center; line-height: 24px;}

html:

<ul id="contextmenu">

<li>复制</li>

<li>剪切</li>

<li>粘贴</li>

<li>全选</li>

</ul>

script:

document.oncontextmenu = function(e){

e = e || event;

// 阻止浏览器默认右键菜单显示

e.preventDefault ? e.preventDefault() : e.returnValue = false;

// 显示自定义的右键菜单

show($("#contextmenu"));

// 修改右键菜单显示位置

css($("#contextmenu"), {

top : e.pageY + "px",

left : e.pageX + "px"

});

}

document.onclick = function(){

hide($("#contextmenu"));

}

addEvent($("#contextmenu"), "click", function(e){

e = e || event;

var src = e.target || e.srcElement;

if (src.nodeName === "LI") {

console.log("你选择了 ：" + src.innerHTML);

}

});

2.窗口范围拖动

html：

<div id="box" style="width: 200px; height:200px; background: black; position: absolute; top:0; left:0;"></div>

script：

// 窗口宽度/高度

var winWidth = document.documentElement.clientWidth || document.body.clientWidth,

winHeight = document.documentElement.clientHeight || document.body.clientHeight;

// 盒子的宽度/高度

var boxWidth = $("#box").offsetWidth,

boxHeight = $("#box").offsetHeight;

$("#box").onmousedown = function(e){

e = e || event;

// 先记录光标在事件源元素坐标系中的位置

var \_offsetX = e.offsetX,

\_offsetY = e.offsetY;

// 绑定文档中移动鼠标的事件

document.onmousemove = function(e){

e = e || event;

// 获取当前窗口中光标位置

var \_clientX = e.clientX,

\_clientY = e.clientY;

// 计算拖拽元素在窗口中的定位

var \_top = \_clientY - \_offsetY,

\_left = \_clientX - \_offsetX;

// 判断是否在范围内

if (\_top < 0) // 上边

\_top = 0;

else if (\_top > winHeight - boxHeight) // 下边

\_top = winHeight - boxHeight;

if (\_left < 0) // 左边

\_left = 0;

else if (\_left > winWidth - boxWidth) // 右边

\_left = winWidth - boxWidth;

// 设置拖拽元素的定位

css($("#box"), {

top : \_top + "px",

left : \_left + "px"

});

}

// 绑定文档中弹起鼠标键的事件

document.onmouseup = function(){

document.onmousemove = null;

}

}

3.限定在窗口范围内拖拽

script:

var outerWidth = $("#outer").clientWidth,

outerHeight = $("#outer").clientHeight;

// 盒子的宽度/高度

var boxWidth = $("#box").offsetWidth,

boxHeight = $("#box").offsetHeight;

$("#box").onmousedown = function(e){

e = e || event;

// 光标在文档中定位

var \_pageX = page(e).x,

\_pageY = page(e).y;

// div#box 在文档中定位

var \_offset = offset(this);

// 记录光标在事件源元素坐标系中的位置

var \_offsetX = \_pageX - \_offset.left,

\_offsetY = \_pageY - \_offset.top;

// 绑定文档中移动鼠标的事件

document.onmousemove = function(e){

e = e || event;

// 根据光标在文档中定位及之前记录的在拖拽元素坐标系中定位计算拖拽元素在文档中的位置

offset($("#box"), {

top : page(e).y - \_offsetY,

left : page(e).x - \_offsetX

});

// 在文档中定位设置后，可能 div#box 会超出父元素范围

// 应控制其位于父元素内部

// 获取 div#box 在其父元素坐标系中坐标位置

var \_top = $("#box").offsetTop,

\_left = $("#box").offsetLeft;

// 判断范围

if (\_top < 0) // 上边

\_top = 0;

else if (\_top > outerHeight - boxHeight) // 下边

\_top = outerHeight - boxHeight;

if (\_left < 0) // 左边

\_left = 0;

else if (\_left > outerWidth - boxWidth) // 右边

\_left = outerWidth - boxWidth;

// 重新设置元素定位

css($("#box"), {

top : \_top + "px",

left : \_left + "px"

});

}

// 绑定文档中弹起鼠标键的事件

document.onmouseup = function(){

document.onmousemove = null;

}

}

html:

<div id="outer" style="width: 600px; height:400px; position: absolute; top:10px; left:100px; border: 1px solid">

<div id="box" style="width: 200px; height:200px; background: black; position: absolute; top:0; left:0;"></div>

4. 获取/设置指定元素在文档中的定位坐标

// 参数：

// element : DOM 元素

// coordinate : 坐标，可选参数，传递时表示设置元素在文档中的坐标

function offset(element,options){

if(typeof options==="undefined"){

var \_top=0,\_left=0;

while(element){

\_left+=element.offsetLeft;

\_top+=element.offsetTop;

element=element.offsetParent;

}

}

return {

top:\_top,

left:\_left

}

}

// 设置

// 先查找父元素在文档中坐标

var \_top=0,\_left=0,currelement=element.offsetParent;

while(currelement){

\_top += currentElement.offsetTop;

\_left += currentElement.offsetLeft;

currentElement = currentElement.offsetParent;

}

// 计算当前元素在文档中定位换算为距其有定位父元素坐标系中的位置

css(currelement,{

top:options.top-\_top+"px",

left:options.left-\_left+"px"

})

}

1. 放大镜效果一

html:

<div id="outer">

<img src="images/middle.webp">

<div id="inner"></div>

</div>

<div id="big">

<img src="images/big.jpg" id="big\_img">

</div>

<style type="text/css">

#outer{width: 400px;height: 400px;border: 1px solid;position: absolute;top: 50px;left: 50px;}

#outer img{width: 400px;height: 400px;}

#inner{position: absolute;top: 0;left: 0;width: 200px;height: 200px;background: #fc0;opacity: 0.3;display: none;}

#big{width: 400px;height: 400px;border: 1px solid;position: absolute;left: 550px;top: 50;display:none;overflow: hidden;}

#big img{width: 800px;height: 800px; position: absolute;top : 0;left: 0;}

</style>

script:

$("#outer").onmouseenter=function(){

show($("#inner"));

show($("#big"));

}

$("#outer").onmouseleave=function(){

hide($("#inner"));

hide($("#big"));

}

$("#outer").onmousemove=function(e){

e=e||event;

//外内宽高(写在内部，不然开始隐藏不具长宽)

//或者写在外部，但开始不能display:none;

var outWidth=$("#outer").clientWidth,

outHeigth=$("#outer").clientHeight,

innWidth=$("#inner").offsetWidth,

innHeight=$("#inner").offsetHeight;

//小方块源元素到文档的距离

offset($("#inner"),{

left:page(e).x-innWidth/2,

top:page(e).y-innHeight/2

})

console.log(innWidth);

//小方块源元素到定位父元素的的位置

var \_left=$("#inner").offsetLeft;

var \_top=$("#inner").offsetTop;

//判断条件在框内运动

if(\_left<0)

\_left=0;

if(\_top<0)

\_top=0;

if(\_left>outWidth-innWidth)

\_left=outWidth-innWidth;

if(\_top>outHeigth-innHeight)

\_top=outHeigth-innHeight;

//重新设置小方块的位置

$("#inner").style.left=\_left+"px";

$("#inner").style.top=\_top+"px";

//大方块中的图片运动

css($("#big\_img"), {

top : \_top \* -2 + "px",

left : \_left \* -2 + "px"

});

}

1. 放大镜效果二

html：

<div id="outer">

<img src="images/middle.webp">

<div id="meng"></div>

<div id="inner">

<img src="images/middle.webp" id="innimg">

</div>

</div>

<div id="big">

<img src="images/big.jpg" id="big\_img">

</div>

<style type="text/css">

#outer{width: 400px;height: 400px;border: 1px solid;position: absolute;top: 50px;left: 50px;}

#outer img{width: 400px;height: 400px;}

#meng{width: 400px;height: 400px;background: #000;position: absolute;top: 0;left: 0;display: none;opacity: 0.8;}

#inner{position: absolute;top: 0;left: 0;width: 200px;height: 200px;overflow: hidden;}

#inner img{

width: 400px;height:400px;position: absolute;

}

#big{width: 400px;height: 400px;border: 1px solid;position: absolute;left: 550px;top: 50;display:none;overflow: hidden;}

#big img{width: 800px;height: 800px; position: absolute;top : 0;left: 0;}

</style>

script:

$("#outer").onmouseenter=function(){

show($("#inner"));

show($("#big"));

show($("#meng"));

}

$("#outer").onmouseleave=function(){

hide($("#inner"));

hide($("#big"));

hide($("#meng"));

}

//外内宽高(写在内部，不然开始隐藏不具长宽)

var outWidth=$("#outer").clientWidth,

outHeigth=$("#outer").clientHeight,

innWidth=$("#inner").offsetWidth,

innHeight=$("#inner").offsetHeight;

$("#outer").onmousemove=function(e){

e=e||event;

//小方块源元素到文档的距离

offset($("#inner"),{

left:page(e).x-innWidth/2,

top:page(e).y-innHeight/2

})

console.log(innWidth);

//小方块源元素到定位父元素的的位置

var \_left=$("#inner").offsetLeft;

var \_top=$("#inner").offsetTop;

//判断条件在框内运动

if(\_left<0)

\_left=0;

if(\_top<0)

\_top=0;

if(\_left>outWidth-innWidth)

\_left=outWidth-innWidth;

if(\_top>outHeigth-innHeight)

\_top=outHeigth-innHeight;

//重新设置小方块的位置

$("#inner").style.left=\_left+"px";

$("#inner").style.top=\_top+"px";

//大方块中的图片运动

css($("#big\_img"), {

top : \_top \* -2 + "px",

left : \_left \* -2 + "px"

});

css($("#innimg"), {

top : \_top \* -1 + "px",

left : \_left \* -1 + "px"

});

}

}

## cookie

本地存储：

服务器端会话跟踪机制：session

特点：

以文本字符串内容保存信息，保存到本地的文件中

会占用网络上传带宽

最大存储容量为 4KB

保存：

document.cookie = "key=value;expires=失效时间;path=路径;domain=域名;secure";

key=value -- 最基本的cookie名值对

expires=失效时间 -- 失效时间，不设置默认会话结束失效 data.toUTCString()

path=路径 -- 通常在服务器环境下，将 path 设置为 "/"

domain=域名

secure -- 是否安全链接

修改：

覆盖保存，在同路径下保存相同 cookie 名称的内容

删除：

覆盖保存，将失效时间设置为当前时间之前的某一刻

查询：

document.cookie 读取cookie，会将网站下所有cookie以 "; "(分号+空格) 的方式连接返回

如：username=xiaoming; password=123456; age=25; address=cd

目前在firefox中可操作

1.侧边栏菜单

html:

<header class="header">头部</header>

<nav class="nav">导航</nav>

<div class="banner">banner图</div>

<div>广告</div>

<div>推荐</div>

<div id="model">滑动出现精彩部分</div>

<div id="menu">菜单</div>

<style type="text/css">

.header{height: 100px;background: #999;}

.nav{height: 300px;background: yellow;}

.banner{height: 500px;background: blue;}

#model{height: 600px;background: red;}

#menu {position: fixed; bottom:300px; right: 30px; background: black; width:50px; height:100px; display: none;}

</style>

script:

var winHeight=document.documentElement.clientHeight||document.body.clientHeight;

var \_top=offset($("#model")).top;

window.onscroll=function(){

var \_scrollTop=document.documentElement.scrollTop||document.body.scrollTop;

if(\_scrollTop>\_top-winHeight/2){

// $("#menu").style.display="block";

show($("#menu"));

}else{

hide($("#menu"));

}

}

2.记住用户名cookie

html：

<h1>用户登录</h1>

<input type="text" id="username" placeholder="用户名"><br>

<input type="password" id="password" placeholder="密码"><br>

<label><input type="checkbox" id="remember">记住用户名与密码</label><br>

<button id="btnLogin">登录</button>

script：

//页面打开，显示已保存的用户名和密码

//读取

var \_uname=cookie("uname");

var \_upwd=cookie("upwd");

if (\_uname)

$("#username").value = \_uname;

if (\_upwd)

$("#password").value = \_upwd;

if (\_uname && \_upwd)

$("#remember").checked = true;

/\*var cookies = document.cookie.split("; ");

cookies.forEach(function(curr){

var cookie = curr.split("=");

if(cookie[0] === "uname") {

$("#username").value = cookie[1];

} else if (cookie[0] === "upwd") {

$("#password").value = cookie[1];

}

});\*/

$("#btnLogin").onclick=function(){

//获取用户和密码

var \_username=$("#username").value;

var \_password=$("#password").value;

//判断是否记住用户和密码

if($("#remember").checked){ //选中

//设置

cookie("uname",\_username,{expires:10});

cookie("upwd",\_password,{expires:10});

/\*var date = new Date();

date.setDate(date.getDate() + 10);

document.cookie = "uname=" + \_username + ";expires=" + date.toUTCString();

document.cookie = "upwd=" + \_password + ";expires=" + date.toUTCString();\*/

}else{

//删除

removeCookie("uname");

removeCookie("upwd");

/\*var date = new Date();

date.setDate(date.getDate() - 1);

document.cookie = "uname=;expires=" + date.toUTCString();

document.cookie = "upwd=;expires=" + date.toUTCString();\*/

}

alert("成功");

}

3封装cookie()--读取，保存，删除

// 封装操作 cookie 的函数：获取/设置 cookie

// 参数：

// key : cookie 名

// value : cookie 值，可选，不传递表示读取cookie

// options : 可配置项 {expires:7, path:"/", domain:"", secure:true}

function cookie(key,value,options){

//读取

if(typeof value=="undefined"){

var cookies=document.cookie.split("; ");

for(var i=0,len=cookies.length;i<len;i++){

var cookie=cookies[i].split("=");

var name=decodeURIComponent(cookie.shift());

if(name==key)

return decodeURIComponent(cookie.join("="));

//防止cookie值中有“=”

}

return null;

}

//保存（ 基础 key=value 的结构）

var cookie=encodeURIComponent(key)+"="+encodeURIComponent(value);

//判断是否有可配置项

options=options||{};

if(options.experies){ //失效时间

var date=new Date();

date.setDate(date.getDate()+options.experies);

cookie+=";experies="+date.toUTCString();

}

if(options.path){ //路径

cookie+=";path="+options.path;

}

if (options.domain) // 域名

cookie += ";domain=" + options.domain;

if (options.secure) // 安全

cookie += ";secure";

document.cookie=cookie;

}

// 删除 cookie

function removeCookie(key,options){

options=options||{};

options.experies=-1;

cookie(key,"",options);

}

4.模拟购物车

**product:**

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>产品</title>

<style type="text/css">

</style>

</head>

<body>

<table border="1" id="list">

<tr>

<td>编号</td>

<td>名称</td>

<td>价格</td>

<td>操作</td>

</tr>

<tr>

<td>1</td>

<td>小米手机</td>

<td>1299</td>

<td><a href="javascript:void(0);">加入购物车</a></td>

</tr>

<tr>

<td>2</td>

<td>小米平板</td>

<td>1599</td>

<td><a href="javascript:void(0);">加入购物车</a></td>

</tr>

<tr>

<td>3</td>

<td>小米空调</td>

<td>4399</td>

<td><a href="javascript:void(0);">加入购物车</a></td>

</tr>

</table>

<a href="cart.html" target="\_blank">查看购物车</a>

<script type="text/javascript" src="js/tool.js"></script>

<script type="text/javascript">

//读取cookie

$("#list").onclick=function(e){

e=e||event;

var src=e.target||e.srcElement;

if(src.nodeName=="A"){

var \_children=src.parentNode.parentNode.children;

//将当前选购的商品保存到对象中

var product={

id:\_children[0].innerHTML,

name:\_children[1].innerHTML,

price:\_children[2].innerHTML,

amount:1

};

//读取cookie中是否之前已有保存过得

var \_products=cookie("products");

if(\_products)

\_products=JSON.parse(\_products);

else

\_products=[];

//判断是否当前商品已有选购过

var idx=index(product.id,\_products);

if(idx!==-1)

\_products[idx].amount++;

else

\_products.push(product);

//保存后的数组存回cookie中

cookie("products",JSON.stringify(\_products),{expires:10});

alert("成功");

}

}

// 查找指定 id 的商品在已选购商品数组中是否存在，返回其下标

// -1表示不存在

function index(id,products){

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

if(products[i].id==id)

return i;

}

return -1;

}

</script>

</body>

</html>

**cart:**

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>产品购物车</title>

<style type="text/css">

.title, .main, .row {clear: both;}

.title div, .row div{width: 100px; height: 30px;line-height: 30px; float: left; border:1px solid; text-align: center;}

.minus, .add {display:inline-block; width:20px; height:20px; line-height: 20px; text-align: center; border:1px solid; cursor: pointer;}

</style>

</head>

<body>

<div class="title">

<div>编号</div>

<div>商品名称</div>

<div>单价</div>

<div>数量</div>

<div>小计</div>

<div>操作</div>

</div>

<div class="main">

</div>

<div class="template" style="display: none;">

<div>1</div>

<div>小米手机</div>

<div>1999</div>

<div><span class="minus">-</span><input type="text" value="1" size="1"><span class="add">+</span></div>

<div>1999</div>

<div><a href="javascript:void(0);">删除</a></div>

</div>

<script type="text/javascript" src="js/tool.js"></script>

<script type="text/javascript">

//读取cookie中保存的购物车商品信息

var \_products=cookie("products");

if(\_products)

\_products=JSON.parse(\_products);

else{

alert("商品为空，请选购商品");

location="product.html";

}

//遍历\_products数组中每个元素

for(var i=0,len=\_products.length;i<len;i++){

var node=$(".template")[0].cloneNode(true);

node.className="row";

show(node);

// 为当前节点对象添加自定义属性，用于保存当前行所显示商品对象的信息（目的：数据缓存）

node.prod = \_products[i]

$(".main")[0].appendChild(node);

node.children[0].innerHTML=\_products[i].id;

node.children[1].innerHTML=\_products[i].name;

node.children[2].innerHTML=\_products[i].price;

node.children[3].children[1].value=\_products[i].amount;

node.children[4].innerHTML=\_products[i].price\*\_products[i].amount;

}

// 事件委派来处理“删除”，数量“+、-”

addEvent($(".main")[0], "click", function(e){

e = e || event;

var src = e.target || e.srcElement;

if (src.nodeName === "A") { // "删除"

del(src);

} else if (src.nodeName === "SPAN") { // "+" "-"

modifyAmount(src);

}

})

//+-操作

function modifyAmount(src){

// 获取 “+” 所在行

var \_row=src.parentNode.parentNode;

//获取该行缓存的商品对象

var \_prod=\_row.prod;

//数量加减

if(src.className=="add"){

\_prod.amount++;

}else if(src.className=="minus"){

if(\_prod.amount<=1)

return;

\_prod.amount--;

}

//重新保存cookie

cookie("products",JSON.stringify(\_products),{expires:10});

//显示更新后的数量

\_row.children[3].children[1].value=\_prod.amount;

//更新小计

\_row.children[4].innerHTML=\_prod.amount\*\_prod.price;

}

//输入数量修改

var \_inputs=$("input",$(".main")[0]);

for(var i=0,len=\_inputs.length;i<len;i++){

\_inputs[i].onblur=function(){

var \_row=this.parentNode.parentNode;

var \_prod=\_row.prod;

if(isNaN(this.value)||this.value<1){ //非数字

this.value=\_prod.amount;

return;

}

\_prod.amount=this.value;

//重新保存cookie

cookie("products",JSON.stringify(\_products),{expires:10});

//显示小计

\_row.children[4].innerHTML=\_prod.amount\*\_prod.price;

}

}

//删除

function del(src){

var \_row=src.parentNode.parentNode;

// 获取当前行中删除商品对象

var \_prod=\_row.prod;

// 查询该商品对象在所有商品的数组中索引

var idx=inArray(\_prod,\_products);

//从数组中删除idx索引出的元素

\_products.splice(idx,1);

// 将 \_products 数组保存回 cookie，表示从 cookie 中将当前行商品删除(重新保存)

cookie("products", JSON.stringify(\_products), {expires:10});

// 从 dom 结构中将当前行删除

\_row.parentNode.removeChild(\_row);

}

</script>

</body>

</html>

1）字符串添加html

var html = "";

for (var i = 0, len = \_products.length; i < len; i++) {

var p = \_products[i];

html += '<div class="row">' +

'<div>'+ p.id +'</div>' +

'<div>'+ p.name +'</div>' +

'<div>'+ p.price +'</div>' +

'<div><span class="minus">-</span><input type="text" value="'+ p.amount +'" size="1"><span class="add">+</span></div>' +

'<div>'+ p.price \* p.amount +'</div>' +

'<div><a href="javascript:void(0);">删除</a></div>' +

'</div>';

}

$(".main")[0].innerHTML = html;

（2）ES6添加html

var html = "";

for (var i = 0, len = \_products.length; i < len; i++) {

var p = \_products[i];

// 使用 ES6 中模板字符串来创建各行信息

html += `<div class="row">

<div>${p.id}</div>

<div>${p.name}</div>

<div>${p.price}</div>

<div><span class="minus">-</span><input type="text" value="${p.amount}" size="1"><span class="add">+</span></div>

<div>${p.price \* p.amount}</div>

<div><a href="javascript:void(0);">删除</a></div>

</div>`;

}

$(".main")[0].innerHTML = html;

（3）模板数组追加html

for (var i = 0, len = \_products.length; i < len; i++) {

var node = $(".template")[0].cloneNode(true);

node.className = "row";

show(node);

// 为当前节点对象添加自定义属性，用于保存当前行所显示商品对象的信息（目的：数据缓存）

node.prod = \_products[i];

$(".main")[0].appendChild(node);

node.children[0].innerHTML = \_products[i].id;

node.children[1].innerHTML = \_products[i].name;

node.children[2].innerHTML = \_products[i].price;

node.children[3].children[1].value = \_products[i].amount;

node.children[4].innerHTML = \_products[i].price \* \_products[i].amount;

}

## 十四、正则表达式

正则表达式：

用作于文本字符串格式校验，或文本字符串模式匹配

JS -- RegExp 对象

创建对象：

a. 直接量

var reg = /pattern/attribute eg:/a/g

b. new 构造函数()

var reg = new RegExp(pattern, attribute);

pattern 模式（规则）：

a. 简单模式

如： find child black

var reg = /find/; var reg = /black/

b. 复杂模式

方括号：

[] -- 表示匹配括号范围内的一个字符 [abcd] [a-zA-Z] [0

-58]

[^] -- 不取括号范围内的任意字符 [^a-z]

元字符：

. -- 任意字符(除换行和行结束符)

\w -- 单词字符（字母数字下划线） [a-zA-Z0-9\_]

\d -- 数字字符 [0-9]

\s -- 空白字符（空格、回车、换行、制表...）

\b -- 单词边界

\大写-对应相反的

量词：

+ -- 至少出现一次 {1,}

? -- 最多出现一次 {0,1}

\* -- 可出现任意多次 {0,}

{n} -- 固定出现n次

{n,} -- 至少出现n次

{n,m} -- 出现 n 到 m 次

^ -- 开头

$ -- 结尾

用户名：

以字母开头，其它字符可以是字母数字下划线，整个用户名字符串长度在 4 到

18 位

var reg = /^[a-zA-Z]\w{3,17}$/

var reg = new RegExp("^[a-zA-Z]\\w{3,17}$") (涉及到转义字符）

RegExp 方法：

test(string) -- 检测字符串是否符合正则对象的模式规则，符合返回 true，否则返回 false

exec(string) -- 匹配校验，返回 null 说明不匹配，返回数组说明匹配成功。返回的数组中第

一个元素是完整匹配结果，如果还有更多元素，则是正则模式中各分组匹配的结果

字符串相关的方法

search()

match()

replace()

stringObject.replace(regexp/substr,replacement)

一个新的字符串，是用 replacement 替换了 regexp 的第一次匹配或所有匹配之后得

到的。

split()

修饰符

i 修饰符用于执行对大小写不敏感的匹配。

new RegExp("regexp","i")

/regexp/i

g 修饰符用于执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

new RegExp("regexp","g")

/regexp/g

m

执行多行匹配

1. 检验检验身份证

$("#cid").onblur = function(){

var reg = /(^\d{15}$)|(^\d{17}[0-9x]$)/i;

if (reg.test(this.value)) {

$("#cid\_info").innerHTML = "身份证号码正确";

}else {

$("#cid\_info").innerHTML = "身份证号码格式有误";

}

}

2.身份证验证识别

html:

<input type="text" id="card">

<div id="info"></div>

script:

<script type="text/javascript" src="js/tools.js"></script>

<script type="text/javascript">

$("#card").onkeyup=function(){

var \_val=this.value;

if(\_val.length===18){

var reg=/^(\d{2})(\d{2})(\d{2})(\d{8})(\d{2})(\d)(\d|x)$/i;

var result=reg.exec(\_val);

console.log(result);

var html=`

<div>省份代号：${result[1]}</div>

<div>城市代号：${result[2]}</div>

<div>区县代号：${result[3]}</div>

<div>出生日期：${result[4]}</div>

<div>派出所代号：${result[5]}</div>

<div>性别代号：${result[6]}</div>

<div>校验位：${result[7]}</div>

`;

$("#info").innerHTML=html;

}

}

</script>

3.去掉字符串前后空白 String.prototype.trim()

function trim(str){

var reg=/^\s+|\s+$/g;

return str.replace(reg,"");

}

var s=" hello ";

var \_replace=trim(s);

console.log(\_replace,\_replace.length); //5

## 十五、ES5、ES6新特性

http://kangax.github.io/compat-table/es5/

http://kangax.github.io/compat-table/es6/

http://caniuse.com/

ES5、ES6新特性：

内置对象新增API

Array(ES5):

forEach(function(currentValue, index, array){})

map(fn)

some()

every()

filter()

indexOf()

lastIndexOf()

reduce()

reduceRight()

Array.isArray()

String(ES5):

trim()

Date(ES5):

toJSON()

toISOString()

Date.now()

JSON(ES5新增对象)：

parse()

stringify()

严格模式

限制 JavaScript，使得书写更规范更严谨

使用 "use strict"; 来开启严格模式

"use strict" 必须做为其作用域下的第一条可执行语句，才能启动严格模式

a. 严格模式下，不允许未声明变量直接赋值操作

b. eval() 有其自己的作用域

c. 函数中的 this，严格模式下函数直接调用，函数体中 this 为 undefined，通过 window. 调用，则指向 window 对象

d. 不能使用 8 进制

this 使用在函数中，通常指向的是调用函数的对象

尾逗号

[,,,,].length ---- 支持ES5，则返回 4，不支持则返回5

{} -- 支持ES5，对象最后一个属性之后可以有多余的逗号

bind() 与 this

this 使用在函数中，通常指向的是调用函数的对象，但可能通过函数的 bind() 方法来修改 (call()、apply()方法也能修改)

Function.prototype.bind(thisArg)

修改在函数体中的 this 指向，将函数体中 this 指向为参数 thisArg 所代表的对象，但注意，原函数不受影响，会返回修改 this 指向后的新函数

数据结构：

Symbol

Symbol 是一种特殊的、不可变的数据类型。符号类型是唯一的并且是不可修改的。

var sym = Symbol()

var left = Symbol("left");

var right = Symbol("right");

let/const

作用域：

全局作用域

局部作用域(函数级)

块级作用域：

a. IIFE(立即调用函数表达式--匿名自执行函数)

b. let -- 存在声明提升，但在变量赋值前不能使用。暂存死区

c. const -- 变量赋值后，不能再重新赋值

Map/Set

集合

Map

保存键值对

创建 Map 对象：var map = new Map();

属性：size

API:

set()

get()

forEach()

delete()

clear()

keys()

values()

entries()和输出它本身一样

has()

Set

保存不重复元素

创建 Set 对象：var set = new Set();

添加里面可直接写数组eg:var set = new Set([1,3,1,3]);

属性：size

API:

add()

forEach()

delete()

clear()

一个Map对象以插入顺序迭代其元素 — 一个 for...of 循环为每次迭代返回一个[key，value]数组。

Array.from(set)将参数装换成数组类似于数组结构

for .. of 与 for .. in

for .. of 遍历迭代值 (获取到对应的值）

for .. in 循环遍历 eg:for(var i in array) ---获取的i是索引值

箭头函数(arrow)

函数表达式简写

(参数列表) => {函数主体}

(参数列表) => 返回值表达式

单个参数 => 返回值表达式

() => {}

eg:array.map(curr => curr \* 2)

箭头函数会捕获其所在上下文的 this 值，作为自己的 this 值

解构赋值

将数组元素或对象属性解包到变量中保存

[a, b, ...rest]=arr就是a,b各得一值，rest得到后面所有的

name = stu.name;相当于({name}=stu);

({name:n}=stu); (n就是name了)

模板字符串

`adsfalsdkjfads ${exp} fasdjfkladsf`

${exp}占位获取其它值

Promise

用于一个异步操作的最终完成（或失败）及其结果值的表示

承诺异步执行成功做什么事情，执行失败做什么事情

将异步嵌套调用的方式转换为同步调用的语法

三种状态：

pending 等待

fulfilled

rejected

方法：

then(fn, fn)

使用：

a. 创建 Promise 对象

b. 添加执行器函数

c. 执行器函数中添加异步任务

d. 调用 resolve() 或 reject() 修改 Promise对象的状态

e. 调用 then() 方法，设置异步任务执行成功或失败调用的函数

生成器函数

generator function

Array.from()将参数装换成数组类似于数组结构

1. let和var 的区别与应用

html:

<div>1</div>

<div>2</div>

<div>3</div>

<div>4</div>

<div>5</div>

<style type="text/css">

div {width: 100px;height: 50px;background: #f00; margin:10px;}

</style>

script:

1. var

var divs=$("div");

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

divs[i].index=i;

divs[i].onclick=function(){

console.log("当前索引:"+this.index);

}

}

(2)let

var divs=$("div")

for(let i=0;i<divs.length;i++){

divs[i].onclick=function(){

console.log("当前索引:"+i);

}

}

1. Array.from()

const bar = ["a", "b", "c"];

var str1=Array.from(bar);

console.log(str1);

// ["a", "b", "c"]

var str=Array.from('foo');

console.log(str);

// ["f", "o", "o"]

3.实现同步效果

（1）普通的（不采用）

setTimeout(function(){

console.log("set\_timeout\_1....");

setTimeout(function(){

console.log("set\_timeout\_2.....");

setTimeout(function(){

console.log("set\_timeout\_3.....");

}, 1000)

}, 1000)

}, 1000)

（2）promise

var promise=new Promise(function(resolve,reject){

setTimeout(function(){

console.log("timer111");

resolve();

},1000);

});

promise.then(function(){

return new Promise(function(resolve){

setTimeout(function(){

console.log("timer222");

resolve();

},1000);

})

}).then(function(){

setTimeout(function(){

console.log("timer333");

},1000)

})

## 十六、运动

1.在限定框内运动

html:

<button id="btn">开始</button>

<div id="container" style="width:800px; height:500px; border:1px solid; position: relative; margin:30px auto">

<div id="box" style="width:100px; height:100px; background: red; position: absolute; top:50px; left:100px;"></div>

</div>

script:

$("#btn").onclick=function(){

//外层盒子宽高

var conWidth=$("#container").clientWidth;

var conHeight=$("#container").clientHeight;

//内层盒子宽高

var boxWidth=$("#box").offsetWidth;

var boxHeight=$("#box").offsetHeight;

//计算临界点位置

var rangeX=conWidth-boxWidth;

var rangeY=conHeight-boxHeight;

//获取div#box的初始定位

//var \_left=parseInt(css($("#box"),"left"));

var \_left=$("#box").offsetLeft;

var \_top=$("#box").offsetTop;

//速度

var speedX=2,speedY=3;

//运动

var timer=setInterval(function(){

\_left+=speedX;

\_top+=speedY;

css($("#box"),{

left:\_left+"px",

top:\_top+"px"

});

if(\_left>=rangeX||\_left<=0)

speedX\*=-1;

if(\_top>=rangeY||\_top<=0)

speedY\*=-1;

},1000/60);

}

2. 多属性运动框架

// 参数：

// element: 待添加运动动画效果的元素

// options: 对象，设置各多属性时的属性名与运动终值

// options = {width:100, height:300, left:200, top: 800}

// speed: 运动总限时

// fn: 可选，运动动画效果执行结束后再执行到的函数

function animate(element,options,speed,fn){

//先停止元素上已有的运动

clearInterval(element.timer);

var start={},range={};

//得到各属性的起始值和差值

for(var attr in options){

start[attr]=parseFloat(css(element,attr));

range[attr]=options[attr]-start[attr];

}

//记录运动起始时间

var startTime=+new Date();

//启动运动动画计时器

element.timer=setInterval(function(){

//计算运动消耗时间

var elapsed=Math.min(speed,new Date()-startTime);

//根据公式计算各属性当前步情况

for(var attr in options){

var result=elapsed\*range[attr]/speed-start[attr];

//设置css对应属性

element.style[attr]=result+(attr=="opacity"?"":"px");

}

//判断是否停止运动

if(elapsed>=speed){

clearInterval(element.timer);

fn&&fn();

}

},1000/60);

}

3.淡入运动

function fadeIn(element,speed,fn){

show(element);

element.style.opacity=0;

animate(element,{opacity:1},speed,fn);

}

4.淡出运动

function fadeOut(element,speed,fn){

animate(element,{opacity:0},speed,fn);

hide(element);

fn&&fn();

}

5.操作封装的函数

html:

<button id="go">运动</button>

<button id="in">淡入</button>

<button id="out">淡出</button>

<div id="box" style="width: 100px;height: 100px;background: red;"></div>

script:

$("#go").onclick=function(){

animate($("#box"),{width:600},3000)

}

$("#in").onclick=function(){

fadeIn($("#box"),2000)

}

$("#out").onclick=function(){

fadeOut($("#box"),2000)

}

## 十七、轮播

1.淡入淡出轮播

html:

<div id="container">

<ul>

<li style="display: block;"><img src="images/images/1.jpg"></li>

<li ><img src="images/images/2.jpg"></li>

<li ><img src="images/images/3.jpg"></li>

<li ><img src="images/images/4.jpg"></li>

<li ><img src="images/images/5.jpg"></li>

<li ><img src="images/images/6.jpg"></li>

</ul>

<div id="page">

<div class="current"></div>

<div></div>

<div></div>

<div></div>

<div></div>

<div></div>

</div>

<div id="prev">&lt;</div>

<div id="next">&gt;</div>

</div>

<style type="text/css">

#container{width: 790px;height: 340px;position: relative;margin:100px auto;}

ul{list-style:none;width: 790px;height: 34px;position: 340px;position: relative;}

ul li{width: 790px;height: 340px;position: absolute;top: 0;left: 0;display: none;}

#page{width: 180px;height: 30px;background: #999;position:absolute;left:300px;bottom: 0;box-sizing: border-box;border-radius: 5px;}

#page div{width: 20px;height: 20px;margin:auto;background: #fff;float: left;border-radius: 50%;margin:5px;}

#prev,#next{width: 45px;height: 80px;position: absolute;top:0;bottom: 0;margin: auto;background: #000;opacity: 0.3;line-height: 80px;color: red;text-align: center;display: none;font-weight: bold;font-size:50px;}

#next{right: 0;}

#page div.current{background: blue;}

</style>

**script:**

<script type="text/javascript">

//得到所有轮播的图片

var imgs=$("li",$("#container"));

//得到所有小圆点

var circles=$("div",$("#page")),

len=imgs.length, //总图片张数

currentIndex=0, //当前图片索引

nextIndex=1, //即将显示图片张数

timer=null ;//计时器id

//自动轮播切换

timer=setInterval(move,2000);

//鼠标移入停止轮播

$("#container").onmouseenter=function(){

clearInterval(timer);

show($("#prev"));

show($("#next"));

}

$("#container").onmouseleave=function(){

clearInterval(timer);

hide($("#prev"));

hide($("#next"));

}

//小圆点点击

$("#page").onclick=function(e){

e=e||event;

var src=e.target||e.srcElement;

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

if(src==circles[i]){

if(i==currentIndex)

return;

nextIndex=i;

move();

return;

}

}

}

//向前翻页

$("#prev").onclick=function(){

nextIndex=currentIndex-1;

if(nextIndex<0)

nextIndex=len-1;

move();

}

//向后翻页

$("#next").onclick=move;

//轮播切换图片（小圆点也动）

function move(){

//图片淡入淡出

fadeOut(imgs[currentIndex],1000);

fadeIn(imgs[nextIndex],1000);

//原点随着移动

circles[currentIndex].className="";

circles[nextIndex].className="current";

//修改索引

currentIndex=nextIndex;

nextIndex++;

//临界条件

if(nextIndex==len)

nextIndex=0;

}

</script>

2.无缝滑动轮播

3.封装边框内部宽度

function innerWidth(element){

var \_display=css(element,"display");

if(\_display==="none"){

return parseFloat(css(element,"width"))

+parseFloat(css(element,"paddingLeft"))

+parseFloat(css(element,"paddingRight"));

}

return element.clientWidth;

}

4.封装边框及内部宽度

function outWWidth(element){

var \_display=css(element,"display");

if(\_display=="none"){

return parseFloat(css(element,"width"))

+parseFloat(css(element,"paddingLeft"))

+parseFloat(css(element,"paddingRight"))

+parseFloat(css(element,"borderLeftWidth"))

+parseFloat(css(element,"borderRightWidth"));

}

return element.offsetWidth;

}

## 十八、瀑布流

1.吸顶效果

<div id="search"></div> （body的高为2000px）

<style type="text/css">

#search{

width: 100%;height: 100px;background: #ccc;position: fixed;top: 0;left: 0;display: none;

}

</style>

script:

<script src="js/tools.js"></script>

<script type="text/javascript">

var isShow=false; //吸顶标记

window.onscroll=function(){

//滚动高度

var \_scrollTop=document.documentElement.scrollTop||document.body.scrollTop;

//判断

if(\_scrollTop>1000){ //显示

if(!isShow){

isShow=true;

show($("#search"));

css($("#search"),"height",0);//初始高度

animate($("#search"),{height:100},2000);

}

}else{

if(isShow){ //隐藏

animate($("#search"),{height:0},2000,function(){

hide($("#search"));

});

isShow=false;

}

}

}

</script>

2绝对定位最短距离的瀑布流

html：

<div id="container">

<div><img src="images/imgs (10).jpg"></div>

<div><img src="images/imgs (12).jpg"></div>

<div><img src="images/imgs (13).jpg"></div>

<div><img src="images/imgs (14).jpg"></div>

<div><img src="images/imgs (15).jpg"></div>

<div><img src="images/imgs (16).jpg"></div>

<div><img src="images/imgs (17).jpg"></div>

<div><img src="images/imgs (18).jpg"></div>

<div><img src="images/imgs (19).jpg"></div>

<div><img src="images/imgs (20).jpg"></div>

<div><img src="images/imgs (21).jpg"></div>

<div><img src="images/imgs (22).jpg"></div>

<div><img src="images/imgs (23).jpg"></div>

<div><img src="images/imgs (24).jpg"></div>

<div><img src="images/imgs (25).jpg"></div>

<div><img src="images/imgs (26).jpg"></div>

<div><img src="images/imgs (27).jpg"></div>

<div><img src="images/imgs (28).jpg"></div>

</div>

<style type="text/css">

#container{width: 95%;margin:auto;position: relative;}

#container div{width: 200px;padding: 5px;border:1px solid #666;border-radius:5px;position: absolute;top: 0;left: 0;}

#container div img{width: 200px;}

</style>

script：

<script src="js/tools.js"></script>

<script type="text/javascript">

window.onload=waterfall;

window.onresize=waterfall;

function waterfall(){

var containerWidth=$("#container").clientWidth,

imgBoxs=$("div",$("#container"));

imgBoxsWidth=imgBoxs[0].offsetWidth;

cols=Math.floor(containerWidth/imgBoxsWidth); //列数

spacing=(containerWidth-cols\*imgBoxsWidth)/(cols+1); //间隙

height=[];//获取各列的高度

for(var i=0,len=imgBoxs.length;i<len;i++){

if(i<cols)

height.push(0);

var shortest=findShortest(height),

currCol=shortest.index, //当前最短的索引列

\_top=height[currCol]+10,

\_left=currCol\*imgBoxsWidth+(currCol+1)\*spacing;

css(imgBoxs[i],{

top:\_top+"px",

left:\_left+"px"

});

height[currCol]+=imgBoxs[i].offsetHeight;

}

}

//最短列的选择函数

function findShortest(height){

var min=height[0],index=0;

for(var i=1;i<height.length;i++){

if(min>height[i]){

min=height[i];

index=i;

}

}

return {

index:index,

value:min

}

}

</script>

3． 动态添加图片的瀑布流

// 动态添加更多的图片

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

// 先将动态添加图片盒子的字符串构建

var html = "";

for (var i = 11; i <= 199; i++) {

html += `<div class="add"><img src="images/imgs (${i}).jpg"></div>`;

}

// 追加到 div#container 容器中

$("#container").innerHTML += html;

// 获取新添加的所有DOM元素

var newAdd = $(".add");

// 遍历每个DOM元素

for (let i = 0; i < newAdd.length; i++) {

// 处理div.add下的图片加载完毕事件

$("img", newAdd[i])[0].onload = function(){

// 当前盒子图片已经加载完毕，放置到最短列中定位

var currCol = findShortest(height).index,

\_top = height[currCol] + 10,

\_left = currCol \* imgBoxWidth + (currCol + 1) \* spacing;

css(newAdd[i], {

top : \_top + "px",

left : \_left + "px"

});

height[currCol] += newAdd[i].offsetHeight + 10;

}

}

4.瀑布流列容器形式（和普通的瀑布流一样）

html：

<div id="container">

<div class="img\_box"><img src="images/imgs (11).jpg"></div>

<div class="img\_box"><img src="images/imgs (12).jpg"></div>

<div class="img\_box"><img src="images/imgs (13).jpg"></div>

<div class="img\_box"><img src="images/imgs (14).jpg"></div>

<div class="img\_box"><img src="images/imgs (15).jpg"></div>

<div class="img\_box"><img src="images/imgs (16).jpg"></div>

<div class="img\_box"><img src="images/imgs (17).jpg"></div>

<div class="img\_box"><img src="images/imgs (18).jpg"></div>

<div class="img\_box"><img src="images/imgs (19).jpg"></div>

<div class="img\_box"><img src="images/imgs (20).jpg"></div>

<div class="img\_box"><img src="images/imgs (21).jpg"></div>

<div class="img\_box"><img src="images/imgs (22).jpg"></div>

<div class="img\_box"><img src="images/imgs (23).jpg"></div>

<div class="img\_box"><img src="images/imgs (24).jpg"></div>

<div class="img\_box"><img src="images/imgs (25).jpg"></div>

<div class="img\_box"><img src="images/imgs (26).jpg"></div>

<div class="img\_box"><img src="images/imgs (27).jpg"></div>

<div class="img\_box"><img src="images/imgs (28).jpg"></div>

<div class="img\_box"><img src="images/imgs (29).jpg"></div>

</div>

<style type="text/css">

#container{width: 95%;margin:auto;position: relative;}

#container .img\_box{width: 200px;padding: 5px;border:1px solid #666;border-radius:5px;margin-top:10px;}

#container div img{width: 200px;}

</style>

script:

<script src="js/tools.js"></script>

<script type="text/javascript">

window.onload=function(){

var containerWidth=$("#container").clientWidth,

imgBoxs=$(".img\_box",$("#container")),

imgBoxWidth=imgBoxs[0].offsetWidth,

cols=Math.floor(containerWidth/imgBoxWidth),

spacing=(containerWidth-cols\*imgBoxWidth)/(cols+1);

//创建各列容器

var html="";

for(var i=0;i<cols;i++){

html+=`<div class="col" style="width:${imgBoxWidth}px;margin-left:${spacing}px;display:inline-block;vertical-align:top;"></div>`;

}

$("#container").innerHTML+=html;

//列容器都dom对象

var columns=$(".col");

// 将 imgBoxes 中每个对象依次添加到 columns 中的每个盒子中

var array=Array.from(imgBoxs);

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

var currCol=i%cols;

columns[currCol].appendChild(array[i]);

}

console.log(array);

}

</script>

## 十九、类和对象

面向对象：

类 -- 类型(JS中没有类的概念)

对象 -- 实体、实例

三大特性：

封装：隐藏内部实现细节，对外提供使用接口

继承：重用，复用

多态：重用

抽象

抽取对象

属性：对象的特征

对象：

属性：静态特征

方法：动态动作

对象创建：

a. 对象直接量

var obj = {}

b. new Object()

var obj = new Object()

缺点：创建大量对象时显得非常繁琐

c. 工厂方式

function createStudent(name, age, sex, height) {

var stu = new Object();

stu.name = name;

stu.age = age;

stu.sex = sex;

stu.height = height;

return stu;

}

缺点：无法具体区分出对象的类型

d. 构造函数（构造函数：每个单词首字母都大写）

obj instanceof Constructor

判断 obj 对象是否为 Constructor 创建出来的对象，或是 Constructor 后代的对象

说明：

构造函数中的this通常指向的是 new 调用构造函数所创建出来的对象

e. 构造函数 + prototype 原型

原型：共享资源

在构造函数中初始化对象的静态属性

在原型中添加对象共享的方法

函数也是特殊的对象，在所有函数对象中，都有一个 prototype 属性。prototype 通常也是一个对象。

在所有对象中（包括普通对象、函数对象）都有一个隐式的 \_\_proto\_\_ 属性，这个属性指向其构造函数中的显式属性 prototype。

在对象属性查找时，先在对象自向空间中查找，如果能够找到，则直接使用，如果找不到，到原型（prototype）中继续查找，如果原型中能找到，则使用，找不到，再到原型的原型中继续查找（原型链），直到找到第一个满足条件的属性调用使用，如果一直查找到 Object.prototype 中都未找到(因为 Object.prototype.\_\_proto\_\_ 为 null)，则返回 undefined

JS 中的对象都是基于原型的对象

## 二十一、服务器端

C/S

B/S

服务器端：

wamp 集成开发环境：

apache

mysql

php

....

将资源内容添加到 web 容器中(部署)，就是在 wamp 安装路径下的 www 目录中

部署在 web 容器中的资源，可在浏览器中通过 http://127.0.0.1:80 或 http://localhost:80 来访问到

PHP:

创建动态交互性站点的强有力的服务器端脚本语言

php 文件后缀名为 .php

标记：

<?php

// PHP服务器端脚本程序

?>

输出：

echo ""; // 向文档流中打印输出字符串内容

echo '<meta charset="utf-8">';

echo "用户名：$username"."<br>";

echo '用户名：$username'; //不能输出

单引号和双引号并用

变量：

$var = value;

PHP 是弱类型的语言

数据类型：

运算符：

流程控制语句：

条件：

if

if-else

多重if

switch

循环：

while

do-while

for

数组：

$arr = array();

数值数组

关联数组

求数组长度： count() 函数

$arr=array(1,4,6,8,9,2);

for($i=0;$i<count($arr);$i++){

echo "第$i个元素".$arr[$i]."<br>"; //$i输出不了

echo "第 $i 个元素".$arr[$i]."<br>";

}

或

$arr = array("001"=>"小明", "002"=>"小红");

echo "001号：" . $arr["001"];

$\_GET变量：

用于获取 get 请求中传递的数据

get 请求，会将向服务器传递的数据串联在 URL 后，用 ? 号串联连接。数据传输格式为 key=value 键值对的格式，如果有多个数据一起传递，则使用 & 连接。

以 get 请求方式向服务器传递的数据只能为字符串格式，并且长度受限，安全性相对较弱。

一般 GET 用于获取服务器资源.

$\_POST 变量：

获取 post 请求中传递的数据

post 请求，将向服务器传递的数据包含在请求主体中传递到服务器端，理论上可以是任意格式的内容，任意大小的资源。安全性相对较强。

一般 POST 用于向服务器提交保存的资源

$\_REQUEST变量：

获取get/post的数据

PHP $\_REQUEST 用于收集HTML表单提交的数据。

eg:var \_name=$\_GET[“username”]

数据库：

数据仓库

MySQL

Oracle

MSSQL

DB2

Access

SQL

-- 结构化查询语言

-- 用于数据库平台

-- 分类：DDL DCL DML DQL(数据定义语言，控制，操作，查询语言）

使用 MySQL 数据库：

a. 创建数据库

CREATE DATABASE `h51705` ;

b. 创建数据库表

CREATE TABLE `h51705`.`users` (

`id` INT NOT NULL AUTO\_INCREMENT PRIMARY KEY ,

`username` VARCHAR( 50 ) NOT NULL ,

`password` VARCHAR( 50 ) NOT NULL ,

UNIQUE (

`username`

)

) ENGINE = INNODB CHARACTER SET utf8 COLLATE utf8\_unicode\_ci

c. 添加数据：

INSERT INTO `h51705`.`users` (

`id` ,

`username` ,

`password`

) VALUES (

NULL , 'test', 'abcdefg'

);

INSERT INTO `h51705`.`users` (

`id` ,

`username` ,

`password`

) VALUES (

NULL , 'xiaoming', 'abcdefg'

), (

NULL , 'xiaohong', '123456'

);

d. 修改数据：

UPDATE `h51705`.`users` SET `password` = 'abc123' WHERE `users`.`id`=2;

e. 删除数据：

DELETE FROM `h51705`.`users` WHERE `users`.`id`=2

f. 简单查询：

SELECT id, username FROM `users` WHERE username='xiaohong' AND password='abcdefg'

g. 聚合函数：

SELECT COUNT(\*) FROM users;

json\_encode($row)得到

{"id":"1","username":"test","password":"abc"}

在线json.cn中可解析格式化

连接数据库库：

// 连接数据库服务器

$conn = mysql\_connect("localhost:3306", "root", "");

// 选择数据库

mysql\_select\_db("h51705", $conn);

// 编写插入的SQL语句

$sql = "INSERT INTO users (username, password) VALUES ('$uname', '$upwd')";

// 执行SQL语句

$result = mysql\_query($sql, $conn);

// 判断执行结果

if ($result) {

echo '{"status":1,"message":"success"}';

} else {

echo '{"status":0,"message":"failed"}';

}

// 关闭数据库连接

mysql\_close($conn);

## 二十二、ajax

CRUD:

添加、查询、更新、删除

ajax:

阿贾克斯

Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。

异步：

JavaScript：XMLHttpRequest - 提供客户端浏览器连接服务器的能力

XML：

JSON -- 数据存储、交换的格式

局部刷新技术（无刷新技术）

步骤（打电话）：

a. 创建核心对象(准备电话)

var xhr=new XMLHttpRequest();

b. 准备建立连接(拨号)

xhr.open(method, url, async);

c. 发送请求(呼出)

xhr.send()

d. 处理响应(通话)

xhr.onreadystatechange

xhr.readyState -- 就绪状态码，表明 xhr 状态，也就是请求到达哪个阶段了（4位请求成功））

xhr.status -- HTTP状态码,1xx,2xx,3xx,4xx,5xx （200为处理成功）

注意：

向服务器发送数据时，如果是 get 请求，则将数据以 ? 号串联在 URL 之后。如果是 post 请求，将查询字符串作为 send() 方法的参数传递，同时，在 send() 调用前，先调用响应头 xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

（使用时有时需要在wamp中的apache2下的conf下的httpd修改Deny from all

为Allow from 127.0.0.1）

API接口文档

1.封装ajax操作函数

// 参数：可配置对象

/\*options = {

type : "get", // 请求方式，get或post，默认为 get

url : "", // 请求资源的URL

data : {username:"xiaoming", age:15, address:"cd"}, // 向服务器传递的数据

dataType : "json|text", // 预期从服务器返回数据的格式

success : function(responseData){}, // 请求成功执行的函数

error : function(xhr,errMsg){} // 请求失败执行的函数

}\*/

function ajax(options) {

var url = options.url; // 请求URL

if (!url) // 没有请求的服务器资源URL

return;

var method = options.type || "get"; // 请求方式

// 处理向服务器发送的数据

// 创建查询字符串内容

var queryString = null;

if (options.data) {

queryString = [];

// 将 {username:"xiaoming", age:15, address:"cd"} 生成 username=xiaoming&age=15&address=cd

for (var attr in options.data) {

queryString.push(attr + "=" + options.data[attr]);

}

queryString = queryString.join("&");

}

// 是否向 URL 后串联查询字符串

if (method == "get" && queryString) {

url += "?" + queryString;

queryString = null;

}

// 创建核心对象

var xhr = new XMLHttpRequest();

// 打开

xhr.open(method, url, true);

// 判断是否 post 请求

if (method == "post")

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

// 发送

xhr.send(queryString);

// 处理响应

xhr.onreadystatechange = function(){

if (xhr.readyState === 4) {

if (xhr.status === 200) {

var data = xhr.responseText;

// 判断预期从服务器返回数据的格式

if (options.dataType == "json")

data = JSON.parse(data);

// 数据处理业务

options.success && options.success(data);

} else {

options.error && options.error(xhr, xhr.statusText);

}

}

}

}

// 处理 get 请求的 ajax 操作

function get(url, data, success, dataType) {

ajax({

type : "get",

url : url,

data : data,

dataType : dataType,

success : success

});

}

function post(url, data, success, dataType) {

ajax({

type : "post",

url : url,

data : data,

dataType : dataType,

success : success

});

}

// 明确返回数据格式为 json

function getJSON(url, data, success){

ajax({

type : "get",

url : url,

data : data,

dataType : "json",

success : success

});

}

2.注册的请求

（1）模拟后台数据（服务器端）

<?php

//请求数据

var uname=$\_REQUEST["username"];

var upwd=$\_REQUEST["password"];

//连接数据库服务器

$con=mysql\_connect("localhost:3306","root","");

// 选择数据库

mysql\_select\_db("h51705",$con);

//编写插入的sql语句

$sql="INSERT INTO user (username, password) VALUES ('$uname','$upwd')";

//执行sql语句

$result=mysql\_query($sql,$con);

//判断执行结果

if($result){

$arr=array("username"=>$uname,"level"=>"vip1");

echo '{"status":1,"message":"success","data":'.json\_encode($arr).'}';

}else{

echo '{"status":0,"message":"failed"}';

}

//关闭数据库连接

mysql\_close($con);

?>

（2）提交表单

<form action="register.php" method="post" id="reg\_form">

<input type="text" placeholder="用户名" name="un" id="username"><span id="username\_info"></span><br>

<input type="password" placeholder="密码" name="password" id="password"><br>

<button>注册</button>

</form>

<script>

$("#reg\_form").onsubmit=function(e){

e=e||event;

e.preventDefault?e.preventDefault():e.returnValue=false;

// ajax来提交用户注册数据

ajax({

type:"post",

url:"register.php",

data:{

username:$("#username").value,

password:$("#password").value

},

dataType:"json",

success:function(data){

var data=data.data;

if(data.status==1){

cookie("user",JSON.stringify(user));

location="success.html";

}else{

location="failed.html"

}

}

})

}

</script>

3.模拟php服务器的聊天记录

### chat.php

<?php

// 允许跨域资源访问

header("Access-Control-Allow-Origin:\*");

// 获取当前执行动作：send(发送聊天消息) receive(接收聊天消息)

$action = $\_REQUEST["action"];

// echo $action;

// 获取客户端IP地址函数

function getClientIP()

{

$ip = "Unknow";

if (getenv("HTTP\_CLIENT\_IP"))

$ip = getenv("HTTP\_CLIENT\_IP");

else if(getenv("HTTP\_X\_FORWARDED\_FOR"))

$ip = getenv("HTTP\_X\_FORWARDED\_FOR");

else if(getenv("REMOTE\_ADDR"))

$ip = getenv("REMOTE\_ADDR");

return $ip;

}

/\* 连接数据库 \*/

$conn = mysql\_connect("localhost:3306","root","");

if (!$conn) {

die ("error : " . mysql\_error());

}

/\* 指定连接的具体数据库名称 \*/

mysql\_select\_db("h51705", $conn);

/\* 设置向数据库读写数据时的编码 \*/

mysql\_query("set character set 'utf8'");

mysql\_query("set names 'utf8'");

// 判断操作

if ("receive" == $action) { // 接收聊天数据

// 查询所有聊天消息的 SQL 语句

$sql = "SELECT id, nickname, message, ip, createtime FROM chat";

// 执行查询

$result = mysql\_query($sql);

// 处理查询结果

$arr = array();

while ($row = mysql\_fetch\_array($result, MYSQL\_ASSOC)) {

$arr[] = $row;

}

// 返回响应

echo json\_encode($arr);

} else if ("send" == $action) { // 发送聊天数据

// 获取发送聊天消息

$nickname = $\_REQUEST["nickname"];

$message = $\_REQUEST["message"];

$ip = getClientIP();

// 生成 SQL 语句

$sql = "INSERT INTO chat (nickname, message, ip) VALUES ('$nickname', '$message', '$ip')";

// 执行 SQL 语句

$result = mysql\_query($sql, $conn);

// 判断执行结果

if ($result)

echo '{"status":1, "message":"success"}';

else

echo '{"status":0, "message":"failed"}';

}

// 关闭数据库连接，释放资源

mysql\_close($conn);

?>

### chat.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>局域网聊天</title>

<style type="text/css">

#info {width: 400px;height: 400px;border:1px solid; overflow:

auto;}

</style>

</head>

<body>

<div id="info"></div>

昵称：<input type="text" id="nickname" value="小明"><br>

<textarea id="message" cols="40" rows="15"></textarea>

<button id="btnSend">发送</button>

<script src="js/tools.js"></script>

<script>

$("#message").onkeydown = function(e){

e = e || event;

var code = e.which || e.keyCode;

if (e.ctrlKey && code == 13) {

sendMsg();

}

}

$("#btnSend").onclick = sendMsg;

function sendMsg(){

var \_nickname = $("#nickname").value,

\_msg = $("#message").value;

ajax({

url : "http://10.7.151.200/chat.php",

data : {

action : "send",

nickname : \_nickname,

message : \_msg

},

dataType : "json",

success : function(data){

if (data.status == 1) {

$("#info").innerHTML += `<div>

${\_nickname} 说：${\_msg}</div>`;

$("#message").value = "";

}

}

});

}

// 使用计时器，每隔 1s 从服务器读取一次数据

setInterval(function(){

getJSON("http://10.7.151.200/chat.php",

{action:"receive"}, function(data){

var html = "";

data.forEach(function(curr){

html += `<div>${curr.nickname}

(${curr.ip}) ${curr.createtime} 说：${curr.message}</div>`;

});

$("#info").innerHTML = html;

});

}, 1000);

</script>

</body>

</html>

## 二十三、跨域

https://www.showapi.com/

阿里云

在谷歌控制台中network得到数据源(看请求源)

资源跨域访问问题：

同源策略：

浏览器中的安全策略，保障浏览器中非同源资源间不能直

接访问

URL：

协议://主机名:端口/路径?查询字符串#hash

同源：协议、域名、端口完全一致

http://localhost/test.html

http://localhost/test.php

非同源：

http://127.0.0.1/test.html

http://localhost/test.php

跨域：

非同源资源间的访问

解决资源跨域访问问题：

1. cors （克服了AJAX只能同源使用的限制）

在服务器端设置响应头：

header("Access-Control-

Allow-Origin:\*");

能够处理 get/post 请求

2. jsonp

利用 <script src=""> 引入外部JS文件

的方式

(<script>使用src引入外部JS资源时，

不受同源策略影响)

如果提供的是 jsonp 跨域访问的能力，

则通常在服务器资源中，会构建一个能够在 JS 中使用到的 函数调

用 内容，函数调用时传递的参数即为需要跨域访问到的 JSON 格式

数据。

前端使用步骤：

a. 创建 <script> 元素

b. 设置 src 属性为访问资源

的路径，通常串联 callback 参数，指明回调函数名称

c. 将创建的 <script> 元素追

加到文档中，在使用完毕后从文档中移除

d. 创建回调函数，在回调函数

中处理获取到的数据（一般是全局的回调函数）

[注]JSONP只能处理GET请求方式的跨域

问题

JSONP 测试接口：http://asilu.com/

1．根据地名查天气（ajax请求）

<input type="text" id="city" placeholder="地名"><button id="btnSearch">查询</button>

<div id="info"></div>

<script src="js/tools.js"></script>

<script>

$("#btnSearch").onclick = function(){

ajax({

type : "get",

url : "http://route.showapi.com/9-2",

data : {

showapi\_appid:"29550",

showapi\_sign:"08402fce064a484baad949d9a18f75e7",

area:$("#city").value,

needMoreDay:1

},

dataType : "json",

success : function(data){

// 显示天气预报信息

var weather = data.showapi\_res\_body;

var html = "";

for (var attr in weather) {

if (attr.charAt(0) == "f") { // 预报

var forcast = weather[attr];

html += `<div>日期：${forcast.day}，白天温度：${forcast.day\_air\_temperature}，夜晚温度：${forcast.night\_air\_temperature}，白天天气：${forcast.day\_weather} <img src='${forcast.day\_weather\_pic}'>，夜晚天气：${forcast.night\_weather} <img src='${forcast.night\_weather\_pic}'></div>`;

}

}

// 显示

$("#info").innerHTML = html;

}

});

}

</script>

2. 根据地名查天气（jsonp请求）

<input type="text" id="city" value="成都"><button id="search">查询</button>

<div id="result">

</div>

<script src="js/tools.js"></script>

<script>

$("#search").onclick = function(){

/\*ajax({

type:"get",

url:"https://api.asilu.com/weather/",

data: {

city : $("#city").value,

callback : "handle"

}

});\*/

var \_script = document.createElement("script");

\_script.src = "https://api.asilu.com/weather/?callback=handle&city=" + $("#city").value;

$("body")[0].appendChild(\_script);

$("body")[0].removeChild(\_script);

}

function handle(data) {

var html = "";

data.weather.forEach(function(weather){

html += `<div>日期：${weather.date}，温度：${weather.temp}，天气：${weather.weather}</div>`;

});

$("#result").innerHTML = html;

}

</script>

3．异步请求验证码和验证

<h1>用户登录</h1>

<input type="text" placeholder="用户名"><br>

<input type="password" placeholder="密码"><br>

<input type="text" placeholder="验证码" id="code"> <img src="" id="code\_img"> <span id="code\_info"></span><br>

<button>登录</button>

<script src="js/tools.js"></script>

<script>

//获取到验证码

function loadCode() {

ajax({

type : "get",

url : "http://route.showapi.com/932-2",

data : {

showapi\_appid:"29550",

showapi\_sign:"08402fce064a484baad949d9a18f75e7",

length : 6,

specials : true

},

dataType : "json",

success : function(data){

data = data.showapi\_res\_body;

$("#code\_img").src = data.image;

$("#code\_img").sid = data.sid;

}

});

}

loadCode();

// 点击更换验证码

$("#code\_img").onclick = function(){

loadCode();

$("#code").value = "";

}

// 失去焦点验证验证码输入是否正确

$("#code").onblur = function(){

getJSON(

"http://route.showapi.com/932-1",

{

showapi\_appid:"29550",

showapi\_sign:"08402fce064a484baad949d9a18f75e7",

checkcode : this.value,

sid : $("#code\_img").sid

},

function(data){

data = data.showapi\_res\_body;

if (data.valid) {

$("#code\_info").innerHTML = "正确";

} else {

$("#code\_info").innerHTML = "验证码输入错误";

}

}

);

}

</script>

## 二十四、闭包与继承

Promise 对象

用于一个异步操作的最终完成（或失败）及其结果值的表示。

承诺在异步操作成功或失败后继续完成的事情

即可以将异步操作调用的方式转换为同步调用的方式

闭包：Closures

作用域：

全局

局部（函数级）

块级

外层函数嵌套内层函数，在内层函数中有使用到外层函数中局部变量完成其功能，外部函数中返回对内层函数的引用。

闭包可以延长函数内部局部变量的生命周期

面向对象：

封装、继承、多态

继承：

prototype

\_\_proto\_\_

call()

apply()

prototype 原型：共享资源

在所有函数对象中都有 prototype 属性

在所有对象中（包括函数对象、普通对象）都有 \_\_proto\_\_ 属性

通常对象的 \_\_proto\_\_ 指向的是其构造函数中的 prototype 属性

由 \_\_proto\_\_ 与 prototype 组成的链式结构，称作是原型链，Object.prototype.\_\_proto\_\_ 为 null，即表示原型链的终点

在函数体中 this 的指向：

通常 this 指向的是调用函数的对象

修改函数体中 this 的指向：

Function.prototype.bind()

call()

语法： fn.call(thisArg, param1, param2, param3,....);

作用：

调用 fn 函数，在 fn 函数执行时，将函数体中的 this 指向 thisArg 所代表的对象。param1,

param2, param3,.... 表示 fn 函数在调用时传递的实际参数列表(使用call方法调用父构造函数)

apply()

语法： fn.apply(thisArg, []|arguments)

若thisArg指定为 null 或 undefined 时会自动指向全局对象（浏览器中就是window对象）

继承：

拷贝式继承

原型继承 ()

通常构造函数中使用 call/apply 来继承属性，在原型中使用 Xx.prototype = new Xxx() 来继承方法.

ES6 中 class

创建对象、实现继承的语法糖，底层仍然是基于原型的方式实现

创建类：

class 类名 {

constructor(){} //一个 class 中只能有一个指定的 ”constructor“ （构造）方法

method(params){}

}

继承：

class 类名 extends 父类名 {

constructor(){

super(); // 调用父类的构造函数（写在第一句）

}

method(param) {

super.methodName(); // 调用父类的方法

}

}

Object.create() -- 方法会使用指定的原型对象及其属性去创建一个新的对象。

Rectangle.prototype = Object.create(Shape.prototype);

Object.defineProperty() 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， 并返回这个对

象。

给定一个数组，各元素为数字，如何快速查找出数组中最大值？

Math.max()

var re=Math.max.apply(null,arr);

1．闭包解释

function test(){

var num=10;

function inner(){

num++;

console.log("inner中的num:"+num);

}

return inner;

}

var result=test();

result();

2.闭包使用一

html:

<div></div>

<div></div>

<div></div>

<div></div>

<div></div>

<style type="text/css">

div {width: 300px;height: 50px;margin:10px; background: #f00;}

</style>

script:

（1） var divs=$("div");

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

divs[i].index=i;

divs[i].onclick=function(){

console.log("当前索引:"+this.index);

}

}

（2）

for (let i = 0; i < divs.length; i++) {

divs[i].onclick = function(){

alert("当前盒子索引：" + i);

}

}

（3）

var divs=$("div");

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

divs[i].onclick=clk(i);

}

function clk(num){

return function(){

console.log("当前索引:"+num);

}

}

3.闭包使用3

function Student(name, sex, age) {

this.name = name;

var \_sex = "男", \_age = 18;

if (sex === "男" || sex === "女") {

\_sex = sex;

}

if (age >= 18 && age <= 25) {

\_age = age;

}

this.getSex = function(){

return \_sex;

}

this.setSex = function(sex){

if (sex === "男" || sex === "女") {

\_sex = sex;

}

}

}

var stu = new Student("小明", "男", 18);

console.log(stu.name);

console.log(stu.getSex());

stu.setSex("狗");

console.log(stu.name);

console.log(stu.getSex())

4.原型

function Student(name, age) {

this.name = name;

this.age = age;

/\*this.study = function(){

console.log(this.name + " 学习...");

}\*/

}

Student.prototype.study = function(){

console.log(this.name + " 学习...");

}

var stu1 = new Student("张三");

var stu2 = new Student("李四");

stu1.study();

stu2.study();

console.log(stu1.study === stu2.study)

（prototype是资源共享，得到结果为true,而this.study得到的是false）

5.拷贝式继承

function inherit(src,dest){

for(var attr in src){

dest[attr]=src[attr];

}

}

6.原型继承

function Role(id,name){

this.id=id;

this.name=name;

}

Role.prototype.eat=function(){

console.log(this.name+"开始吃饭了哟");

}

function Student(id,name){

Role.call(this,id,name); //继承属性

//Role.apply(this,[id,name]);

}

//原型继承

Student.prototype=new Role();

Student.prototype.constructor=Student;

var stu=new Student(1,"小兰");

stu.eat();

7.ES6中的继承

//父级

class Role{

constructor(id,name){

this.id=id;

this.name=name;

}

eat(){

console.log(this.name+"人是要吃饭的");

}

}

//子类

class Student extends Role{

constructor(id,name,score){

super(id,name); //父类的

this.score=score;

}

eat(){

super.eat();

console.log("我要去食堂吃饭");

}

}

var stu=new Student(5,"小兰","100");

stu.eat();

console.log("分数"+stu.score);

## 二十五、版本系统和飞机大战

版本控制系统：CVS 、 SVN 、GIT

SVN

是一个开放源代码的版本控制系统

（安装visiualSVN和tortoiseSVN）

使用：

服务器端：

a. 创建用户

b. 创建新的仓库（版本库）

c. 向新版本库中导入初始化项目版本

客户端：

a. checkout 、 update（在当前文件夹下右键选择）

b. 在检出的工作目录中实现代码编写、修改操作

c. 提交修改后的代码到服务器版本库中

d. 如果在提交过程中有文件冲突问题，需要解决文件冲突（通常和冲

突方同事沟通合并）

https://qianfeng-PC/svn/jsword/

**飞机大战：**

### index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>飞机大作战首页</title>

</head>

<body>

</body>

<script src="js/tools.js"></script>

<script src="js/map.js"></script>

<script src="js/game.js"></script>

<script src="js/role.js"></script>

<script src="js/self.js"></script>

<script src="js/enemy.js"></script>

<script src="js/bullet.js"></script>

<script type="text/javascript">

//工具对象

var Tool={

//判断两个矩形是否相交

intersect:function(obj1,obj2){

return !(obj1.y>obj2.y+obj2.height

||obj1.x>obj2.x+obj1.width

||obj1.y+obj1.height<obj2.y

||obj1.x+obj1.width<obj2.x

);

}

}

var game=new Game();

game.init();

game.play();

</script>

</html>

### map.js

// 地图

function Map(width,height,start\_bg,bg){

this.width=width||320;

this.height=height||568;

this.element=null; //所有地图内容的容器

this.gameContainer=null; //正式游戏界面的容器

this.self=null; //地图所关联的战机

//...

}

//定义方法，初始化创建地图dom元素

Map.prototype.init=function(){

//创建地图容器

var \_container=this.element=document.createElement("div");

//设置样式

css(\_container,{

width:this.width+"px",

height:this.height+"px",

position:"absolute",

top:0,

left:0,

bottom:0,

right:0,

margin:"auto",

border:"1px solid",

overflow:"hidden"

});

//容器中开始界面

var \_start=document.createElement("div");

css(\_start,{

height:this.height+"px",

background:"url('images/start\_bg.png')"

});

\_container.appendChild(\_start);

//容器中游戏界面

var \_game=this.gameContainer=document.createElement("div");

css(\_game,{

height:this.height+"px",

background:"url('images/bg.png')",

position:"relative"

});

\_container.appendChild(\_game);

//添加到文档中

$("body")[0].appendChild(\_container);

//为开始界面绑定点击事件

var that=this;

on(\_start,"click",function(){

hide(\_start);

that.self.move(); //让战机能够移动

});

}

### role.js

//公共角色

function Role(options){

options=options||{},

this.width=options.width,

this.height=options.heigth,

this.element=null,

this.imgSrc=options.imgSrc,

this.x=options.x,

this.y=options.y,

this.map=null, //关联地图对象

this.speed=options.speed; //移动速度

this.isAlive=true //标记当前对象在游戏中是否还存活

}

//角色初始化，创建页面dom节点

Role.prototype.init=function(){

//创建节点

var \_img=this.element=document.createElement("img");

//添加节点src属性

\_img.src=this.imgSrc;

//设置属性

css(\_img,{

position:"absolute",

top:this.y+"px",

left:this.x+"px",

width:this.width+"px",

height:this.height+"px"

});

//将角色图片添加到地图中

this.map.gameContainer.appendChild(\_img);

}

//角色在地图中移动

Role.prototype.move=function(){

//移动一步后的垂直坐标

this.y+=this.speed;

//定位dom元素

css(this.element,{

top:this.y+"px"

});

//判读当角色图片运动移除了地图范围，则删除

if(this.y<0||this.y>this.map.height){

this.isAlive=false;

this.map.gameContainer.removeChild(this.element);

}

}

/\*Role.prototype = {

constructor : Role,

init : function(){},

move : function(){}

};\*/

### self.js

//战机本身

function Self(options){

//继承属性

Role.call(this,options);

}

//继承原型中的方法

Self.prototype=new Role();

Self.prototype.constructor=Self;

//让战机移动起来

Self.prototype.move=function(){

var that=this;

this.map.element.onmousemove=function(e){

e=e||event;

offset(that.element,{

top:page(e).y-that.height/2,

left:page(e).x-that.width/2

});

that.x=that.element.offsetLeft;

that.y=that.element.offsetTop;

}

}

### bullet.js

//子弹

function Bullet(options){

Role.call(this,options);

}

Bullet.prototype=new Role();

### enemy.js

//敌机

function Enemy(options){

Role.call(this,options);

}

Enemy.prototype=new Role();

### game.js

// 游戏平台

function Game(){

this.map=null; //地图

this.self=null; //自己战机

this.enemies=[]; //保存所有敌机的数组

this.bullets=[]; //保存所有子弹的数组

}

//游戏平台初始化

Game.prototype.init=function(){

//地图初始化

this.map=new Map();

this.map.init();

//战机初始化

this.self=new Self({

width:66,

height:80,

imgSrc:"images/self.gif",

x:100,

y:300

});

this.self.map=this.map; //建立战机与地图的关联

this.map.self=this.self; //建立地图鱼战机的关联

this.self.init();

}

//组织游戏流程

Game.prototype.play=function(){

var count=0;

var that=this;

setInterval(function(){

count++;

//生成小飞机

if(count%60===0){

//创建敌机对象

var enemy=new Enemy({

width:34,

height:24,

imgSrc:"images/small\_fly.png",

x:Math.floor(Math.random()\*that.map.width),

y:0,

speed:3

});

//建立敌机与地图关联

enemy.map=that.map;

//敌机初始化

enemy.init();

//将当前敌机添加到所有敌机数组中

that.enemies.push(enemy);

}

//生成中飞机

if(count%300==0){

//创建敌机对象

var enemy=new Enemy({

width:46,

height:60,

imgSrc:"images/mid\_fly.png",

x:Math.floor(Math.random()\*that.map.width),

y:0,

speed:2

});

//建立敌机与地图关联

enemy.map=that.map;

//敌机初始化

enemy.init();

//将当前敌机添加到所有敌机数组中

that.enemies.push(enemy);

}

//生成子弹

if(count%20==0){

//创建子弹对象

var bullet=new Bullet({

width:6,

height:14,

imgSrc:"images/bullet.png",

x:that.self.x+that.self.width/2,

y:that.self.y-15,

speed:-2

});

//建立子弹与地图关联

bullet.map=that.map;

//子弹初始化

bullet.init();

//将当前敌机添加到所有子弹数组中

that.bullets.push(bullet);

}

//让敌机移动起来

for(var i=that.enemies.length-1;i>=0;i--){

that.enemies[i].move();

if(!that.enemies[i].isAlive)

that.enemies.splice(i,1);

}

//让子弹移动起来

for(var i=that.bullets.length-1;i>=0;i--){

that.bullets[i].move();

if(!that.bullets[i].isAlive)

that.bullets.splice(i,1);

}

//判断子弹与敌机是否有碰撞

for(var i=that.enemies.length-1;i>=0;i--){

var enemy=that.enemies[i];

for(var j=that.bullets.length-1;j>=0;j--){

var bullet=that.bullets[j];

//判断敌机与子弹是否相交

if(Tool.intersect(enemy,bullet)){

that.map.gameContainer.removeChild(bullet.element);

that.map.gameContainer.removeChild(enemy.element);

that.bullets.splice(j,1);

that.enemies.splice(i,1);

break;

}

}

}

},1000/60);

}

### tools.js

// 封装函数，实现根据id、标签、类名查找元素

// 参数：

// selector : 选择器，如：#id, .className, element

// context : 查询上下文环境，可选参数，表示一个具体的DOM对象

// 返回值：

// 返回满足条件的结果

function $(selector, context){

context = context || document; // 默认查询上下文为 document

if (selector.indexOf("#") === 0) // id

return document.getElementById(selector.slice(1));

if (selector.indexOf(".") === 0) // className

return getElementsByClassName(selector.slice(1), context);

return context.getElementsByTagName(selector);

}

// 解决 getElementsByClassName() 方法兼容问题

// 参数：

// className : 类名

// context : 查询上下文环境，可选参数，表示一个具体的DOM对象

// 返回值：

// 返回根据类名查找的结果

function getElementsByClassName(className, context) {

context = context || document;

// 判断浏览器是否支持使用 getElementsByClassName()

if (context.getElementsByClassName) // 支持

return context.getElementsByClassName(className);

/\* 不支持 \*/

var result = []; // 保存查找到结果的数组

// 查找在查询上下文后代中所有的元素

var allElements = context.getElementsByTagName("\*");

// 遍历所有元素

for (var i = 0, len = allElements.length; i < len; i++) {

// 判断当前遍历到的元素类名中是否存在待查找的类名

var classNames = allElements[i].className.split(" ");

if (inArray(className, classNames) !== -1) // 存在

result.push(allElements[i]);

}

// 返回查找结果

return result;

}

// 判断给定的某个值是否在数组中存在，存在则返回其下标，不存在返回-1

// 参数：

// value : 待查找判断的值

// array : 数组

// 返回值：

// 数组中的下标，-1表示不存在

function inArray(value, array) {

if (Array.prototype.indexOf) // 支持使用数组的 indexOf() 方法

return array.indexOf(value);

// 不支持使用 indexOf()

for (var i = 0, len = array.length; i < len; i++) {

if (array[i] === value)

return i;

}

return -1;

}

// 封装函数，获取/设置CSS样式

// 使用 css(element, "width") -- 获取 element元素的 width 属性值

// 使用 css(element, "width", "100px") -- 设置 element 元素的 width 属性值为 100px(只设置一个属性)

// 使用 css(element, {width:"100px", height:"100px"}) -- 设置 element 元素的多个CSS属性，属性名与属性值使用对象表示

// 参数：

// element : DOM元素对象

// attr : CSS属性名

// value : CSS属性值，可选，不传表示获取CSS，传递表示设置CSS

// 返回值：

// 查找到的CSS属性值

function css(element, attr, value) {

// 获取

if (typeof attr === "string" && typeof value === "undefined")

return window.getComputedStyle

? getComputedStyle(element)[attr]

: element.currentStyle[attr];

// 设置

if (typeof attr === "string" && value){

element.style[attr] = value;

} else if (typeof attr === "object") {

for (var prop in attr) {

element.style[prop] = attr[prop];

}

}

}

// 显示 element 元素

function show(element) {

element.style.display = "block";

}

// 隐藏 element 元素

function hide(element) {

element.style.display = "none";

}

// 注册事件监听

function on(element, type, callback) {

if (element.addEventListener) { // 支持使用 addEventListener

if (type.indexOf("on") === 0)

type = type.slice(2);

element.addEventListener(type, callback, false);

} else { // 不支持使用 addEventListener

if (type.indexOf("on") !== 0)

type = "on" + type;

element.attachEvent(type, callback);

}

}

// 解除事件监听

function off(element, type, callback) {

if (element.removeEventListener) { // 支持使用 removeEventListener

if (type.indexOf("on") === 0)

type = type.slice(2);

element.removeEventListener(type, callback, false);

} else { // 不支持使用 removeEventListener

if (type.indexOf("on") !== 0)

type = "on" + type;

element.detachEvent(type, callback);

}

}

// 获取/设置指定元素在文档中的定位坐标

// 参数：

// element : DOM 元素

// coordinate : 坐标，可选参数，传递时表示设置元素在文档中的坐标

function offset(element, coordinate) {

if (typeof coordinate === "undefined") { // 获取

var \_top = 0, \_left = 0;

while(element) {

\_top += element.offsetTop;

\_left += element.offsetLeft;

element = element.offsetParent;

}

return {

top : \_top,

left : \_left

};

}

// 设置

// 先查找父元素在文档中坐标

var \_top = 0, \_left = 0, currentElement = element.offsetParent;

while(currentElement) {

\_top += currentElement.offsetTop;

\_left += currentElement.offsetLeft;

currentElement = currentElement.offsetParent;

}

// 计算当前元素在文档中定位换算为距其有定位父元素坐标系中的位置

css(element, {

top : coordinate.top - \_top + "px",

left : coordinate.left - \_left + "px"

});

}

// 解决 event.pageX/event.pageY 兼容问题

function page(e) {

var \_x = e.pageX || (e.clientX + (document.documentElement.scrollLeft || document.body.scrollLeft)),

\_y = e.pageY || (e.clientY + (document.documentElement.scrollTop || document.body.scrollTop));

return {

x : \_x,

y : \_y

}

}

// 封装操作 cookie 的函数：获取/设置 cookie

// 参数：

// key : cookie 名

// value : cookie 值，可选，不传递表示读取cookie

// options : 可配置项 {expires:7, path:"/", domain:"", secure:true}

function cookie(key, value, options){

// 读取

if (typeof value === "undefined") {

var cookies = document.cookie.split("; ");

for (var i = 0, len = cookies.length; i < len; i++) {

var cookie = cookies[i].split("=");

var name = decodeURIComponent(cookie.shift());

if (name === key)

return decodeURIComponent(cookie.join("="));

}

return null;

}

// 保存

// 基础 key=value 的结构

var cookie = encodeURIComponent(key) + "=" + encodeURIComponent(value);

// 判断是否有可配置项

options = options || {};

if (options.expires) { // 失效时间

var date = new Date();

date.setDate(date.getDate() + options.expires);

cookie += ";expires=" + date.toUTCString();

}

if (options.path) // 路径

cookie += ";path=" + options.path;

if (options.domain) // 域名

cookie += ";domain=" + options.domain;

if (options.secure) // 安全

cookie += ";secure";

document.cookie = cookie;

}

// 删除 cookie

function removeCookie(key, options) {

options = options || {};

options.expires = -1;

cookie(key, "", options);

}

// 多属性运动框架

// 参数：

// element: 待添加运动动画效果的元素

// options: 对象，设置各多属性时的属性名与运动终值

// options = {width:100, height:300, left:200, top: 800}

// speed: 运动总限时

// fn: 可选，运动动画效果执行结束后再执行到的函数

function animate(element, options, speed, fn) {

// 先停止元素上已有运动

clearInterval(element.timer);

// 遍历 options 对象中各属性，设置初始值与运动区间值

// start = {width:100, height:100, left:100, top:50}

// range = {width:0, height:200, left:100, top:750}

var start = {}, range = {};

for (var attr in options) {

start[attr] = parseFloat(css(element, attr));

range[attr] = options[attr] - start[attr];

}

// 记录运动起始时间

var startTime = +new Date();

// 启动运动动画计时器

element.timer = setInterval(function(){

// 计算运动已消耗时间

var elapsed = Math.min(speed, new Date() - startTime);

// 根据公式计算各属性当前步情况

for (var attr in options) {

// 计算遍历到的属性当前步情况

var result = elapsed \* range[attr] / speed + start[attr];

// 设置CSS对应属性

element.style[attr] = result + (attr === "opacity" ? "" : "px");

}

// 判断是否停止运动

if (elapsed === speed) {

clearInterval(element.timer);

// 有传递运动结束后的函数参数，则调用

fn && fn();

}

}, 1000/60);

}

// 淡入

function fadeIn(element, speed, fn) {

show(element);

element.style.opacity = 0;

animate(element, {opacity:1}, speed, fn);

}

// 淡出

function fadeOut(element, speed, fn) {

animate(element, {opacity:0}, speed, function(){

hide(element);

fn && fn();

});

}

// 边框内部宽度

function innerWidth(element) {

var \_display = css(element, "display");

if (\_display === "none") {

return parseFloat(css(element, "width"))

+ parseFloat(css(element, "paddingLeft"))

+ parseFloat(css(element, "paddingRight"))

}

return element.clientWidth;

}

function innerHeight(element) {

var \_display = css(element, "display");

if (\_display === "none") {

return parseFloat(css(element, "height"))

+ parseFloat(css(element, "paddingTop"))

+ parseFloat(css(element, "paddingBottom"))

}

return element.clientHeight;

}

// 边框及内部宽度

function outerWidth(element) {

var \_display = css(element, "display");

if (\_display === "none") {

return parseFloat(css(element, "width"))

+ parseFloat(css(element, "paddingLeft"))

+ parseFloat(css(element, "paddingRight"))

+ parseFloat(css(element, "borderLeftWidth"))

+ parseFloat(css(element, "borderRightWidth"))

}

return element.offsetWidth;

}

function outerHeight(element) {

var \_display = css(element, "display");

if (\_display === "none") {

return parseFloat(css(element, "height"))

+ parseFloat(css(element, "paddingTop"))

+ parseFloat(css(element, "paddingBottom"))

+ parseFloat(css(element, "borderTopWidth"))

+ parseFloat(css(element, "borderBottomWidth"));

}

return element.offsetHeight;

}

// 封装 ajax 操作函数

// 参数：可配置对象

/\*options = {

type : "get", // 请求方式，get或post，默认为 get

url : "", // 请求资源的URL

data : {username:"xiaoming", age:15, address:"cd"}, // 向服务器传递的数据

dataType : "json|text", // 预期从服务器返回数据的格式

success : function(responseData){}, // 请求成功执行的函数

error : function(xhr,errMsg){} // 请求失败执行的函数

}\*/

function ajax(options) {

var url = options.url; // 请求URL

if (!url) // 没有请求的服务器资源URL

return;

var method = options.type || "get"; // 请求方式

// 处理向服务器发送的数据

// 创建查询字符串内容

var queryString = null;

if (options.data) {

queryString = [];

// 将 {username:"xiaoming", age:15, address:"cd"} 生成 username=xiaoming&age=15&address=cd

for (var attr in options.data) {

queryString.push(attr + "=" + options.data[attr]);

}

queryString = queryString.join("&");

}

// 是否向 URL 后串联查询字符串

if (method == "get" && queryString) {

url += "?" + queryString;

queryString = null;

}

// 创建核心对象

var xhr = new XMLHttpRequest();

// 打开

xhr.open(method, url, true);

// 判断是否 post 请求

if (method == "post")

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

// 发送

xhr.send(queryString);

// 处理响应

xhr.onreadystatechange = function(){

if (xhr.readyState === 4) {

if (xhr.status === 200) {

var data = xhr.responseText;

// 判断预期从服务器返回数据的格式

if (options.dataType == "json")

data = JSON.parse(data);

// 数据处理业务

options.success && options.success(data);

} else {

options.error && options.error(xhr, xhr.statusText);

}

}

}

}

// 处理 get 请求的 ajax 操作

function get(url, data, success, dataType) {

ajax({

type : "get",

url : url,

data : data,

dataType : dataType,

success : success

});

}

function post(url, data, success, dataType) {

ajax({

type : "post",

url : url,

data : data,

dataType : dataType,

success : success

});

}

// 明确返回数据格式为 json

function getJSON(url, data, success){

ajax({

type : "get",

url : url,

data : data,

dataType : "json",

success : success

});

}

## 二十六、设计模式

设计模式

代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案

单例（Singleton Pattern）

这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建

1、单例类只能有一个实例。

2、单例类必须自己创建自己的唯一实例。

3、单例类必须给所有其他对象提供这一实例

主要解决：一个全局使用的类频繁地创建与销毁

代理（Proxy Pattern）

一个类代表另一个类的功能。

主要解决：在直接访问对象时带来的问题。

工厂（Factory Pattern）

在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

提供对象创建的最佳方式

主要解决：主要解决接口选择的问题。

适配器（Adapter Pattern）

两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能

主要解决：主要解决在软件系统中，常常要将一些"现存的对象"放到新的环境中，而新环境要求的接口是现对象不能满足的。

组合（Composite Pattern）

将对象组合成树形结构以表示"部分-整体"的层次结构。

主要解决：它在我们树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以向处理简单元素一样来处理复杂元素，从而使得客户程序与复杂元素的内部结构解耦。

策略（Strategy Pattern）

定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。

主要解决：在有多种算法相似的情况下，使用 if...else 所带来的复杂和难以维护。

观察者（Observer Pattern）

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

主要解决：一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。

## 二十八、jquery

carousel封装

轮播图

面向对象

ES6

《锋利的jQuery》

jQuery:

JS库:jQuery、prototype.js、mooltools、YUI.......

http://jquery.com/

write less do more

特点：

1. 小巧

2. DOM操作、事件处理、运动动画、AJAX等操作更简便

3. 支持所有 CSS 选择器(包括所有的 CSS3 选择器)

4. 跨浏览器使用（不用再考虑浏览器兼容问题）

5. 链式风格

6. 隐式迭代

7. 完善的API文档

8. 丰富插件

...........

下载：

1.x -- 能够解决IE9之前浏览器兼容问题

3.+++++++++++++++++x -- 不考虑IE9之前浏览器兼容问题

未经压缩文件

压缩文件

引入 JS 库文件：

a. 本地

b. 网络 CDN

jQuery 是以面向对象的方式设计的JS库

$ 和 jQuery 是等价的

$()/jQuery() -- jQuery 对象生产的工厂

jQuery 对象中通常包装的是 DOM 对象

使用过程中必须严格区分 jQuery 对象与 DOM 对象。jQuery 对象与 DOM 对象的方法调用不能混淆

jQuery 与 dom 对象转换：

jQuery --> dom：

jQueryObject[index]

jQueryObject.get(index)

dom --> jQuery:

$(domObject)

eg:DOM对象转jQuery对象

$(document.getElementById("msg"))

jQuery对象转DOM对象

$("#msg")[0]，$("div").eq(1)[0]，$("div").get()[1]，$("td")[5]

框架：

$(document).ready(function(){});

window.onload = function(){} -- 页面内容加载完毕执行

说明：

a. ready() 方法中传递的函数会在 DOM 结构加载结束就执行

b. ready() 方法在同一个文件中可以调用多次，追加

c. 简写：$(function(){});

API 说明：

选择器：-- 作用：实现元素查询，$(selector)

基本

层级

过滤

属性

表单

核心：$()

DOM(文档处理、筛选、属性、CSS)：

事件、事件对象：

运动动画：

ajax、延迟对象（类似Promise）：

工具：

插件：

cookie 插件

先引入 jQuery 库，再引入 cookie 插件库

原生 JS 中购物车 --> jQuery

面向对象轮播图

html：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>轮播图对象</title>

<link rel="stylesheet" href="css/carousel.css">

</head>

<body>

<div id="car1">

</div>

<div id="car2">

</div>

<script src="js/tools.js"></script>

<script src="js/carousel.js"></script>

<script>

//car1的轮播图

new Carousel({

imgs:[

{src:"images/1.jpg",href:"http://www.baidu.com"},

{src:"images/2.jpg", href:"http://www.163.com"},

{src:"images/3.jpg", href:"http://www.sina.com.cn"},

{src:"images/4.jpg", href:"http://baike.baidu.com"}

],

container:$("#car1"),

width:790,

height:340

});

//car2

new Carousel({

imgs : [

{src : "images/sm\_1.png", href:"#"},

{src : "images/sm\_2.png", href:"#"}

],

container : $("#car2"),

width : 180,

height : 225

})

</script>

</body>

</html>

js:

function Carousel(options){

options=options||{};

this.imgs=options.imgs;

this.container=options.container;

this.width=options.width;

this.height=options.height;

this.imgBoxs=[]; //存放所有li的元素

this.circles=null; //存放所有小圆点的元素

this.currentIndex=0;

this.nextIndex=1;

this.timer=null; //计时器id

this.init();

}

Carousel.prototype={

constructor:Carousel,

init:function(){

// 创建轮播图外层的节点

var \_container=document.createElement("div");

\_container.className="carousel\_container";

//设置样式

css(\_container,{

width:this.width+"px",

height:this.height+"px"

});

//放置轮播图片的ul

var \_ul=document.createElement("ul");

css(\_ul,{

width:this.width+"px",

height:this.height+"px"

});

//创建li

for(var i=0,len=this.imgs.length;i<len;i++){

var \_img=this.imgs[i];

var \_li=document.createElement("li");

\_li.innerHTML="<a href='"+\_img.href+"' target='\_blank'><img src='"+\_img.src+"'></a>";

if(i==0)

show(\_li);

\_ul.appendChild(\_li);

//将当前创建的li元素保存到imgBoxs数组

this.imgBoxs.push(\_li);

}

\_container.appendChild(\_ul);

//小圆点

var \_pages=document.createElement("div");

\_pages.className="pages";

css(\_pages,{

width:this.width+"px"

});

for(var i=0;i<len;i++){

\_pages.innerHTML+="<div class='"+(i==0?"current":"")+"'></div>";

}

//保存所有小圆点到对象属性中

this.circles=$("div",\_pages);

\_container.appendChild(\_pages);

//翻页

var \_prev=document.createElement("div");

\_prev.className="prev";

\_prev.innerText="<";

var \_next=document.createElement("div");

\_next.className="next";

\_next.innerText=">";

\_container.appendChild(\_prev);

\_container.appendChild(\_next);

//将当初轮播图的布局结构添加到页面的容器中

this.container.appendChild(\_container);

//自动轮播

var cb=this.move.bind(this);

this.timer=setInterval(cb,3000);

//鼠标移入移出

on(\_container,"mouseenter",()=>{

this.over();

});

on(\_container,"mouseleave",()=>{

this.out();

});

//小圆点切换，事件委派

on(\_pages,"click",(e)=>{

e=e||event;

var src=e.target||e.srcElement;

if(src!=\_pages){

var index=inArray(src,Array.from(this.circles));

if(this.currentIndex!=currentIndex){

this.nextIndex=index;

this.move();

}

}

});

//上下页切换

on(\_prev,"click",()=>{

this.nextIndex=this.currentIndex-1;

if(this.nextIndex<0)

this.nextIndex=this.imgBoxs.length-1;

this.move();

});

on(\_next,"click",()=>{

this.move();

});

},

move:function(){

// 图片的淡入与淡出

fadeIn(this.imgBoxs[this.nextIndex],600);

fadeOut(this.imgBoxs[this.currentIndex],600);

//小圆点样式移动

this.circles[this.currentIndex].className="";

this.circles[this.nextIndex].className="current";

//修改索引

this.currentIndex=this.nextIndex;

this.nextIndex++;

if(this.nextIndex>=this.imgBoxs.length)

this.nextIndex=0;

},

over:function(){

clearInterval(this.timer);

},

out:function(){

this.timer=setInterval(this.move.bind(this),3000);

}

}

css:

.carousel\_container {

position: relative;

overflow: hidden;

}

.carousel\_container ul {

list-style: none;

position: relative;

}

.carousel\_container ul li {

position: absolute;

top : 0;

left: 0;

display: none;

}

.carousel\_container .pages {

height:30px;

background: #000;

position: absolute;

left:0;

bottom: 0;

}

.carousel\_container .pages div {

width: 20px;

height: 20px;

margin:5px;

background: #fff;

float: left;

border-radius: 20px;

}

.carousel\_container .pages div.current {

background: #f00;

}

.carousel\_container .prev,.carousel\_container .next {

width: 45px;

height: 100px;

position: absolute;

top: 0;

bottom: 0;

margin:auto;

background: #000;

color:#fff;

font-size:30px;

font-weight: bold;

text-align: center;

line-height: 100px;

opacity: 0.5;

cursor:default;

}

.carousel\_container .next {

right: 0;

}

## 二十九、jquery购物车以及模板

破坏性操作：

jQuery 对象中所包装的 DOM 对象发生了变化

end() -- 回到最近一次破坏性操作之前的状态

模板引擎

artTemplate

创建模板

使用 <script type="text/html"></script>

模板中通常是 JS 代码与 HTML 代码的混合

HTML 代码直接书写

JS 代码通常使用 {{ }} 或 <% %> 包含

**1.购物车基本功能的实现**

**cart html：**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>购物车</title>

<style type="text/css">

.title, .main, .row , .footer{clear: both;}

.title div, .row div{width: 100px; height: 30px;line-height: 30px; float: left; border:1px solid; text-align: center; overflow: hidden;}

.minus, .add {display:inline-block; width:20px; height:20px; line-height: 20px; text-align: center; border:1px solid; cursor: pointer;}

.prod {width: 100px;height: 130px;line-height: 25px; border:1px solid; margin:10px; float:left;}

</style>

</head>

<body>

<div class="title">

<div><label><input type="checkbox" class="ck\_all">全选</label></div>

<div>编号</div>

<div>商品名称</div>

<div>商品描述</div>

<div>单价</div>

<div>数量</div>

<div>小计</div>

<div>操作</div>

</div>

<div class="main">

</div>

<div class="footer">

合计：<span class="total"> 0</span>

<div>

<a href="javascript:void(0);" class="del\_ck">删除选中</a>

<a href="javascript:void(0);" class="clear">清空购物车</a>

</div>

</div>

<div class="template" style="display: none;">

<div><input type="checkbox" class="ck\_prod"></div>

<div class="id">1</div>

<div class="name">小米手机</div>

<div class="desc">小米手机</div>

<div class="price">1999</div>

<div><span class="minus">-</span><input class="amount" type="text" value="1" size="1"><span class="add">+</span></div>

<div class="sub">1999</div>

<div class="oper"><a href="javascript:void(0);">删除</a></div>

</div>

<h1>推荐购买</h1>

<div class="buy">

</div>

<script type="text/html" id="prod\_temp">

<%

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

var prod=newProducts[i];

%>

<div class="prod">

<div class="id"><%= prod.id %></div>

<div class="name"><%= prod.name %></div>

<div class="price"><%= prod.price %></div>

<div class="desc"><%= prod.desc %></div>

<div><a href="javascript:void(0);" class="addToCart">加入购物车</a></div>

</div>

<%

}

%>

</script>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script src="lib/jquery\_plugins/jquery.cookie.js"></script>

<script src="lib/arttemplate/template-native.js"></script>

<script>

$(document).ready(function(){

//读取购物车的数据

$.cookie.json=true;

var \_products=$.cookie("products")||[];

if(\_products.length==0){

alert("购物车为空");

location="producta.html";

}

//显示购物车已有商品的页面数据

$.each(\_products,function(index,element){

//克隆模板节点

$(".template").clone(true)

.data("product",this) //遍历道德商品对象数据缓存起来

.removeClass("template")

.addClass("row")

.show()

.appendTo(".main")

.children(".id").text(this.id).parent()

.children(".name").text(this.name).end()

.children(".desc").text(this.desc).end()

.children(".price").text(this.price).end()

.find(".amount").val(this.amount).end()

.children(".sub").text(this.price\*this.amount);

});

//删除单件商品

$(".main").on("click","a",function(){

var row=$(this).parents(".row");

//获取当前行所缓存的商品对象

var \_prod=row.data("product");

var \_index=$.inArray(\_prod,\_products);

//数组和cookie中删除

\_products.splice(\_index,1);

$.cookie("products",\_products,{expires:10,path:"/"});

row.remove();

})

//数量加/减(事件委派)

$(".main").delegate(".add,.minus","click",function(){

var row=$(this).parents(".row");

var \_prod=row.data("product");

if($(this).is(".add")){

\_prod.amount++;

}else{

if(\_prod.amount<=1)

return;

\_prod.amount--;

}

//显示加之后的数量

row.find(".amount").val(\_prod.amount);

row.children(".sub").text(\_prod.price\*\_prod.amount);

//将数组保存到cookie中

$.cookie("products",\_products,{expires:10,path:"/"});

})

//数量输入的修改

$(".amount").blur(function(){

var row=$(this).parents(".row");

var \_prod=row.data("product");

if(!/^[1-9]\d\*$/.test($(this).val())){

$(this).val(\_prod.amount);

return;

}

\_prod.amount=$(this).val();

//显示小计

row.children(".sub").text(\_prod.price\*\_prod.amount);

$.cookie("products",\_productsj,{expires:7,path:"/"});

calcTotal();

})

//全选操作

$(".ck\_all").click(function(){

//获取复选框选中状态

var status=$(this).prop("checked"); //获取第一个属性值

//将商品行前的复选框选中状态设置为全选框的状态

$(".ck\_prod").prop("checked",status);

calcTotal();

})

//所有商品选中全选同样操作

$(".ck\_prod").click(function(){

var len=$(".ck\_prod:checked").length;

$(".ck\_all").prop("checked",len===\_products.length);

calcTotal();

})

//定义函数，计算合计金额

function calcTotal(){

//获取选中的行

var rows=$(".ck\_prod:checked").parents(".row");

var sum=0;

rows.each(function(index,element){

sum+=Number($(this).children(".sub").text());

});

//显示合计

$(".total").text(sum);

}

//选中行的删除操作

$(".del\_ck").click(function(){

var rows=$(".ck\_prod:checked").parents(".row");

rows.each(function(){

var \_prod=$(this).data("product");

var \_index=$.inArray(\_prod,\_products);

\_products.splice(\_index,1);

});

$.cookie("products",\_products,{expires:10,path:"/"});

rows.remove();

calcTotal();

})

//清空购物车

$(".clear").click(function(){

$.removeCookie("products",{path:"/"});

$(".main").empty();

$(".total").text(0);

})

//推荐购买商品

$(".buy").on("click",".addToCart",function(){

$(this).parents(".prod");

})

//加载推荐都买模板数据

$.getJSON("mock/products.json",function(data){

//渲染模板

var html=template("prod\_temp",{newProducts:data});

//显示

$(".buy").html(html);

})

})

</script>

</body>

</html>

**producta.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>商品列表</title>

</head>

<body>

<table border="1">

<tr>

<td>商品编号</td>

<td>名称</td>

<td>价格</td>

<td>描述</td>

<td>操作</td>

</tr>

<tr>

<td class="id">1</td>

<td class="name">红米 note4</td>

<td class="price">1299</td>

<td class="desc">小米系列.....</td>

<td class="addToCart"><a href="javascript:void(0);">加入购物车</a></td>

</tr>

<tr>

<td class="id">2</td>

<td class="name">红米 note5</td>

<td class="price">1399</td>

<td class="desc">小米系列.....</td>

<td class="addToCart"><a href="javascript:void(0);">加入购物车</a></td>

</tr>

<tr>

<td class="id">3</td>

<td class="name">红米 note6</td>

<td class="price">1199</td>

<td class="desc">小米系列.....</td>

<td class="addToCart"><a href="javascript:void(0);">加入购物车</a></td>

</tr>

<tr>

<td class="id">4</td>

<td class="name">红米 note7</td>

<td class="price">1599</td>

<td class="desc">小米系列.....</td>

<td class="addToCart"><a href="javascript:void(0);">加入购物车</a></td>

</tr>

</table>

<a href="carta.html" target="\_blank" class="showCart">查看购物车</a>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script src="lib/jquery\_plugins/jquery.cookie.js"></script>

<script>

$(document).ready(function(){

//点击购物车情况

$(".addToCart>a").click(function(){

var row=$(this).parents("tr");

//将当前所在行的商品信息保存到对象

var product={

id:row.children(".id").text(),

name:row.children(".name").text(),

price:row.children(".price").text(),

desc:row.children(".desc").text(),

amount:1

}

//从cookie中读取已保存的购物车信息

$.cookie.json=true;

var \_products=$.cookie("products")||[];

//查找当前商品在数组中是否存在

var \_index=indexOf(product.id,\_products);

if(\_index==-1)

\_products.push(product);

else

\_products[\_index].amount++;

$.cookie("products",\_products,{expires:10,path:"/"});

alert("添加成功");

});

//查找id所在的商品索引是否存在products

function indexOf(id,products){

for(var i=0,len=products.length;i<len;i++){

if(products[i].id==id)

return i;

}

return -1;

}

})

</script>

</body>

</html>

**products.json**

[

{"id":10,"name":"金话筒","price":99,"desc":"自营商品。。。"},

{"id":10,"name":"金话筒","price":99,"desc":"自营商品。。。"},

{"id":10,"name":"金话筒","price":99,"desc":"自营商品。。。"},

{"id":10,"name":"金话筒","price":99,"desc":"自营商品。。。"},

{"id":10,"name":"金话筒","price":99,"desc":"自营商品。。。"}

]

**2.查询天气**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>查询天气</title>

</head>

<body>

<input type="text" id="city"><button>查询</button>

<div></div>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script>

$(function(){

$("button").click(function(){

$.ajax({

type:"get",

url:"http://route.showapi.com/9-2",

data:{

showapi\_appid : "29550",

showapi\_sign : "08402fce064a484baad949d9a18f75e7",

area : $("#city").val(),

needMoreDay : 1

},

dataType:"json",

success:function(data){

data=data.showapi\_res\_body;

var html="";

for(var attrName in data){

if(attrName.indexOf("f")===0){

var forcast=data[attrName];

html += `<div>日期：${forcast.day}，气温：${forcast.day\_air\_temperature} - ${forcast.night\_air\_temperature}，天气：${forcast.day\_weather}</div>`;

}

}

$("div").html(html);

}

})

})

})

</script>

</body>

</html>

**3.jsonp查号码归属地以及推荐**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<button>查询手机号码归属地</button><br>

输入关键词<input type="text" id="sug">

<div class="place"></div>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script>

$(function(){

//打印当前号码的归属地

$("button").click(function(){

var phone="15881272563";

$.getJSON("https://api.asilu.com/phone/?callback=?&phone="+phone,function(data){

console.log(data);

$(".place").text(data.province+"-"+data.city+"-"+data.sp);

})

})

//输入关键词得到推荐

// https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?cb=handle&wd=

$("#sug").keyup(function(){

$.ajax({

type:"get",

url:"https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su",

data:{wd:$(this).val()},

dataType:"jsonp",

jsonp:"cb",

success:function(data){

console.log(data);

}

})

})

});

</script>

</body>

</html>

**4.省份以及地区的选择**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>确认订单</title>

</head>

<body>

<h1>请填写收货信息</h1>

<div>

地址：

<select class="province">

<option value="-1">请选择省份</option>

</select>

<select class="city">

<option value="-1">请选择城市</option>

</select>

<select class="district"></select>

</div>

<div>详细地址：

<textarea name="" id="" cols="30" rows="10"></textarea>

</div>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script>

$(function(){

var url1 = "http://route.showapi.com/1149-1?showapi\_appid=29550&showapi\_sign=08402fce064a484baad949d9a18f75e7&level=1",

url2 = "http://route.showapi.com/1149-1?showapi\_appid=29550&showapi\_sign=08402fce064a484baad949d9a18f75e7&level=1&page=2";

$.when($.ajax(url1),$.ajax(url2)).then(function(data1,data2){

var provinces=data1[0].showapi\_res\_body.data.concat(data2[0].showapi\_res\_body.data);

var html="";

$.each(provinces,function(){

html+=`<option value='${this.id}'>${this.areaName}</option>`;

});

$(".province").append(html);

})

//when提供一种方法来执行一个或多个对象的回调函数，延迟对象通常表示异步事件

//省份/城市选择改变

$(".province,.city").change(function(){

//选中项对应的id

var val=$(this).val();

if(val==-1)

return;

var url = "http://route.showapi.com/1149-2?showapi\_appid=29550&showapi\_sign=08402fce064a484baad949d9a18f75e7&parentId=" + val;

//异步请求

$.getJSON(url,(data)=>{

var cities=data.showapi\_res\_body.data;

var html="<option val='-1'>请选择</option>";

$.each(cities,function(){

html+=`<option value='${this.id}'>${this.areaName}</option>`;

})

if($(this).is(".province"))

$(".city").html(html);

else

$(".district").html(html);

});

})

});

</script>

</body>

</html>

## 三十、运动以及插件的运用

animate() 实现的运动动画效果，在其底层使用的是运动队列的结构来保存各动画效果

css设置宽高不用加单位，就默认为px

插件

插件编写：

$.fn.extend() <==> jQuery.prototype.extend()

==> 向 jQuery 对象中添加扩展功能

==> 调用时，通过 $("xxx").xxx() 方式来调用扩展的功能

$.extend()

==> 向 jQuery 函数本身添加扩展功能

==> 调用时，直接通过 $.xxx() 调用

淘宝镜像（先安装node.js后才在cmd中安装npm）

npm install -g cnpm --registry=https://registry.npm.taobao.org

1、无缝轮播图

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>无缝轮播</title>

<style type="text/css">

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>无缝轮播</title>

<style type="text/css">

\*{margin: 0;padding: 0;}

#container {

width: 790px;

height: 340px;

position: relative;

margin:100px auto;

overflow: hidden;

border:5px solid red;

}

ul {

list-style: none;

width: 790px;

height: 340px;

position: absolute;

top:0;

left: 0;

}

ul li {

width: 790px;

height: 340px;

float: left;

}

#pages {

width:790px;

height: 30px;

background: #000;

position: absolute;

left:0;

bottom: 0;

}

#pages div {

width: 20px;

height: 20px;

margin:5px;

background: #fff;

float: left;

border-radius: 20px;

}

#pages div.current {

background: #f00;

}

#prev,#next {

width: 45px;

height: 100px;

position: absolute;

top: 0;

bottom: 0;

margin:auto;

background: #000;

color:#fff;

font-size:30px;

font-weight: bold;

text-align: center;

line-height: 100px;

opacity: 0.5;

display: none;

}

#next {

right: 0;

}

</style>

</head>

<body>

<div id="container">

<ul id="imgs">

<li><img src="images/1.jpg"></li>

<li><img src="images/2.jpg"></li>

<li><img src="images/3.jpg"></li>

<li><img src="images/4.jpg"></li>

<li><img src="images/5.jpg"></li>

</ul>

<div id="pages"></div>

<div id="prev">&lt;</div>

<div id="next">&gt;</div>

</div>

</body>

<script src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>

<script type="text/javascript">

$(function(){

var imgBoxs=$("#imgs li"),//待轮播的图片

len=imgBoxs.length, //图片的张数

imgWidth=imgBoxs.outerWidth(),//图片的宽度

currentIndex=1,//当前显示图片索引

nextIndex=2,//即将显示图片索引

timer=null; //轮播计时器

//复制第一张和最后一张图片盒子

var first=imgBoxs.eq(0).clone(true),

last=imgBoxs.last().clone(true);

//添加到ul

$("#imgs").append(first).prepend(last);

len+=2;

//设置初始状态

$("#imgs").width(imgWidth\*len).css({"left":-imgWidth});

//动态添加小圆点

var html="";

for(var i=0;i<len-2;i++){

html+="<div></div>";

}

// $("#pages").append(html).children(":first").addClass("current");

$(html).appendTo("#pages").first().addClass("current");

//自动轮播

//鼠标移入/移除容器

$("#container").hover(function(){

// mouseenter

clearInterval(timer);

$("#prev,#next").stop().fadeIn();

},function(){

// mouseleave

timer=setInterval(move,3000);

$("#prev,#next").stop().fadeOut();

}).mouseleave(); //.trigger("mouseleave");

//处理小圆点点击事件

$("#pages").on("click","div",function

(){

var \_index=$(this).index();

if(\_index+1==currentIndex)

return;

nextIndex=\_index+1;

move();

})

//向上下翻页

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

nextIndex=currentIndex-1;

move();

})

$("#next").click(move);

//轮播切换的函数

function move(){

//计算轮播运动定位位置

var \_left=-1\*nextIndex\*imgWidth;

//小圆点切换

var circleIndex=nextIndex-1;

if(nextIndex==0)

circleIndex=len-3;

else if(nextIndex==len-1)

circleIndex=0;

$("#pages div").eq(circleIndex).addClass("current").siblings().removeClass("current");

//运动定位

$("#imgs").stop().animate({left:\_left},function(){

if(nextIndex>=len){

currentIndex=1;

nextIndex=2;

$("#imgs").css({"left":-imgWidth});

}

if(currentIndex==0){

currentIndex=len-2;

nextIndex=len-1;

$("#imgs").css({"left":-(len-2)\*imgWidth});

}

})

//修改索引

currentIndex=nextIndex;

nextIndex++;

}

})

</script>

</html>

2．简单的运动

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<div></div>

<div></div>

<div></div>

<div></div>

<script src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>

<script>

$(function(){

$("div").css({

width:200,

height:100,

margin:10,

background:"#f00"

}).mouseenter(function(){

$(this).stop().animate({width:500},100);

}).mouseleave(function(){

$(this).stop().animate({width:200},1000);

})

})

</script>

</body>

</html>

3.楼层导航效果

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title></title>

<style type="text/css">

.header, .nav, .main, .footer {

width: 800px;

margin:auto;

}

.header {

height : 1000px;

background: #666;

}

.nav {

height: 500px;

background: #ccc;

}

.floor {

height:600px;

}

.footer {

height:300px;

background: #666;

}

.menu {

width: 60px;

list-style: none;

background: #fff;

position: fixed;

bottom: 100px;

left:50px;

display: none;

}

.menu li {

height:50px;

line-height: 50px;

text-align: center;

border-bottom: 1px solid #666;

cursor: pointer;

position: relative;

}

.menu li span {

display: none;

width:60px;

height: 50px;

line-height: 50px;

position: absolute;

top:0;

left:0;

background: #f00;

}

</style>

</head>

<body>

<div class="header">头部</div>

<div class="nav">导航</div>

<div class="main">

<div class="floor" style="background: #f00">服饰</div>

<div class="floor" style="background: #ff0">家电</div>

<div class="floor" style="background: #ddd">数码</div>

<div class="floor" style="background: #f0f">运动</div>

<div class="floor" style="background: #0f0">零食</div>

<div class="floor" style="background: #0ff">图书</div>

</div>

<div class="footer"></div>

<ul class="menu">

<li>1L<span style="display: block;">服饰</span></li>

<li>2L<span>家电</span></li>

<li>3L<span>数码</span></li>

<li>4L<span>运动</span></li>

<li>5L<span>零食</span></li>

<li>6L<span>图书</span></li>

<li>TOP</li>

</ul>

</body>

<script src="http://apps.bdimg.com/libs/jquery/2.1.4/jquery.min.js"></script>

<script type="text/javascript">

$(function(){

//获取窗口高度

var winHeight=$(window).height();

//是否点击菜单后的滚动

var flag=false;//true:点击菜单的滚动 false:鼠标滚动

//处理窗口滚动事件

$(window).on('scroll',function(){

if(!flag){

//获取滚动高度

var \_scrollTop=$(window).scrollTop();

//获取1F楼层的结构高度

var \_top=$(".floor").offset().top;

//判断菜单的隐藏显示

if(\_scrollTop>\_top-winHeight/2){

$(".menu").stop().fadeIn(400);

}else{

$(".menu").stop().fadeOut(400);

}

//楼层滚动汉子的显示与隐藏

$(".floor").each(function(index,element){

// 获取当前遍历到楼层前所有布局的高度

var \_top=$(this).offset().top;

if(\_scrollTop>=\_top-winHeight/2)

$(".menu li").eq(index).addClass("current").children("span").show().

end().

siblings().removeClass("current").children("span").hide();

})

}

})

//处理菜单中菜单项的点击事件

$(".menu").on("click","li:not(:last)",function(){

flag=true;

// 将当前点击li下的span显示，其它隐藏

$(this).addClass("current").children("span").show().end()

.siblings().removeClass("current").children("span").hide();

//获取当前点击菜单项的索引

var \_index=$(this).index();

//对应层的高度

var \_top=$(".floor").eq(\_index).offset().top;

//运动动画效果

$("html,body").stop().animate({scrollTop:\_top},function(){

flag=false;

});

//hover效果

$(".menu li").hover(function(){

$(this).children("span").show();

},function(){

if(!$(this).is(".current"))

$(this).children("span").hide();

});

$(".menu li:last").click(function(){

// $("html, body").stop().animate({scrollTop:0});

$(window).scrollTop(0);

})

})

})

</script>

</html>

3.抛物线插件

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style type="text/css">

.cart {width: 300px;height: 100px;border:1px solid; margin:10px; float: left;}

.side {width: 100px;height: 100%;position: fixed; right: 0; background: #000;}

.cart\_info {width: 80px;height: 100px;position: absolute; top:200px; left:10px; background: #fff;}

</style>

</head>

<body>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="cart"><a href="#">加入购物车</a></div>

<div class="side">

<div class="cart\_info">购物车</div>

</div>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script src="lib/jquery\_plugins/jquery.fly.min.js"></script>

<script>

$(".cart a").click(function(e){

var offset = $(".cart\_info").offset();

var flyer = $("<img src='images/middle.webp' style='width:40px;height:40px;'>");

flyer.fly({

start:{

left : e.clientX,

top : e.clientY

},

end:{

left: offset.left - $(window).scrollLeft(),

top: offset.top - $(window).scrollTop(),

width : 0,

height : 0

}

});

// e.preventDefault();

return false;

});

</script>

</body>

</html>

4.自定义轮播图插件

**html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<link rel="stylesheet" href="css/carousel.css">

</head>

<body>

<div id="car1">

</div>

<div id="car2">

</div>

<script src="lib/jquery/jquery-1.12.4.min.js"></script>

<script src="js/carousel.js"></script>

<script>

$("#car1").carousel({

imgs : [

{src:"images/1.jpg", href:"http://www.baidu.com"},

{src:"images/2.jpg", href:"http://www.163.com"},

{src:"images/3.jpg", href:"http://www.sina.com.cn"},

{src:"images/4.jpg", href:"http://baike.baidu.com"}

],

width : 790,

height : 340

});

new Carousel({

imgs : [

{src : "images/sm\_1.png", href:"#"},

{src : "images/sm\_2.png", href:"#"}

],

container : $("#car2"),

width : 180,

height : 225

});

var result = $.min([3, 5, 1, 7, 9, 2, 6]);

console.log(result);

</script>

</body>

</html>

**js:**

;

(function($){

function Carousel(options) {

options = options || {};

this.imgs = options.imgs;

this.width = options.width;

this.height = options.height;

this.imgBoxes = []; // 存放所有 li 的元素

this.circles = null; // 存放所有小圆点的元素

this.currentIndex = 0;

this.nextIndex = 1;

this.timer = null; // 计时器id

}

Carousel.prototype = {

constructor : Carousel,

init : function(container){

var \_container = $("<div>").addClass("carousel\_container").css({

width:this.width,

height:this.height

}).appendTo(container);

var \_ul = $("<ul>").css({

width:this.width,

height:this.height

}).appendTo(\_container);

for (var i = 0, len = this.imgs.length; i < len; i++) {

var \_img = this.imgs[i],

\_li = $("<li>"),

html = "<a href='"+ \_img.href +"' target='\_blank'><img src='"+ \_img.src +"'></a>";

\_li.html(html);

if (i === 0)

\_li.css("display", "block");

\_li.appendTo(\_ul);

this.imgBoxes.push(\_li);

}

/\* 自动轮播 \*/

var cb = this.move.bind(this);

this.timer = setInterval(cb, 3000);

},

move : function(){ // 轮播切换的函数

$(this.imgBoxes[this.currentIndex]).stop().fadeOut(600);

$(this.imgBoxes[this.nextIndex]).stop().fadeIn(600);

// 修改currentIndex与nextIndex的值

this.currentIndex = this.nextIndex;

this.nextIndex++;

if (this.nextIndex >= this.imgBoxes.length)

this.nextIndex = 0;

}

}

/\* 向 jQuery 对象添加功能 \*/

/\*$.fn.extend({

carousel : function(options){

var car = new Carousel(options);

car.init(this);

}

});

$.fn.carousel = function(options){

new Carousel(options).init(this);

}\*/

$.prototype.carousel = function(options){

new Carousel(options).init(this);

}

/\* 向jQuery本身添加功能 \*/

/\*$.max = function(array){

return Math.max.apply(null, array);

}\*/

$.extend({

max : function(array){

return Math.max.apply(null, array);

},

min : function(array){

return Math.min.apply(null, array);

}

});

})(jQuery);

**css:**

.carousel\_container {

position: relative;

overflow: hidden;

}

.carousel\_container ul {

list-style: none;

position: relative;

}

.carousel\_container ul li {

position: absolute;

top : 0;

left: 0;

display: none;

}

.carousel\_container .pages {

height:30px;

background: #000;

position: absolute;

left:0;

bottom: 0;

}

.carousel\_container .pages div {

width: 20px;

height: 20px;

margin:5px;

background: #fff;

float: left;

border-radius: 20px;

}

.carousel\_container .pages div.current {

background: #f00;

}

.carousel\_container .prev,.carousel\_container .next {

width: 45px;

height: 100px;

position: absolute;

top: 0;

bottom: 0;

margin:auto;

background: #000;

color:#fff;

font-size:30px;

font-weight: bold;

text-align: center;

line-height: 100px;

opacity: 0.5;

cursor:default;

}

.carousel\_container .next {

right: 0;

}

## 三十一、前端自动化构建工具

SASS/LESS

CSS预处理器，

使得 CSS 的开发可以像 JS 一样有变量、流程控制等使用

SASS：

https://www.sass.hk/

sass基于Ruby语言开发而成，因此安装sass前需要安装Ruby（使用koala）

SASS使用：

保存文件：

后缀名为 .scss

变量：

嵌套的CSS规则：

导入：@import

注释：

混合器：

@mixin

@include

继承：

@extend

流程控制：

@if

@if - @else

@if - @else if - @else

@for $var from <start> through <end> ---- 包括开始与结束

@for $var from <start> to <end> ---- 包括开始，不包括结束

@each $var in <list>

@while condition {statements;}

函数：

@function fn(args) {

// statements

@return exp;

}

前端自动化构建工具

gulp、grunt、fis3

gulp

http://www.gulpjs.com.cn/

http://www.ydcss.com/archives/18

gulp是基于Nodejs的自动任务运行器

大致使用流程：

安装nodejs -> 全局安装gulp -> 项目安装gulp以及gulp插件 -> 配置gulpfile.js -> 运行任务

具体：

a. 安装 NodeJS，npm

npm 安装插件时语法：（npm是国外下载，cnpm是国内下载，速度快一点）

npm install <name> [-g] [--save-dev]

b. 选择安装 cnpm，cnpm能够替换npm的功能（淘宝镜像（https://npm.taobao.org/）安装）

npm install -g cnpm --registry=https://registry.npm.taobao.org

可使用 cnpm 安装第三方插件：

cnpm install <name> [-g] [--save-dev]

c. 全局安装 gulp

cnpm install gulp –g （查看用gulp -v）

前 a,b,c 三步执行一次即可

以下步骤是在项目目录下执行：

d. 进入项目目录下(命令提示符下shift+右键)，创建 package.json 文件：

cnpm init

package.json是基于nodejs项目必不可少的配置文件，它是存放在项目根目录的普通json文件

e. 项目本地安装 gulp：

cnpm install gulp --save-dev

项目目录下出现 node\_modules 文件夹

f. 项目本地安装 gulp 插件：

https://www.npmjs.com 可查询各插件使用的方式

CSS压缩：

cnpm install gulp-clean-css --save-dev

JS压缩：

cnpm install gulp-uglify --save-dev

html压缩：

cnpm install gulp-htmlmin --save-dev

sass编译：

cnpm install gulp-sass --save-dev

自动刷新：

cnpm install gulp-livereload --save-dev

安装的gulp插件要配合浏览器中的livereload插件一起使用。在访问资源时要能够自动刷新，需要在服务器环境下，即资源访问的协议为http

文件合并：

gulp-concat

重命名：

gulp-rename

g. 项目根目录下创建 gulpfile.js 文件：

引入方式：var gulp=require(‘gulp);

var minify=require(‘gulp-clean-css’)

创建任务：gulp.task(‘css’,function(){

gulp.src(“路径”)

.pipe(minifycss()) //可有配置选项

.pipe(gulp.dest(‘目录’));//它创建这个目录

})

创建gulp默认任务

gulp.task(“default”,[‘css’,’js’])

gulp监听任务（此处代码。。）运行：gulp watch

h. 运行任务：

命令提示符下：gulp 任务名（eg:gulp css）

零配置服务器WEB容器插件：

http-server -- 只能部署静态资源 (把当前文件改为根目录）

可以将任意目录变成web容器目录

安装：

cnpm install http-server –g

## 三十二、GIT应用

CVS --> SVN --> GIT

GIT

Git是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

使用：

1. 安装 git-bash

2. 全局配置

$ git config --global user.name "runoob"

$ git config --global user.email test@runoob.com

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

// 进入项目目录下操作（在当前项目目录中右键-->git bash here）

a. 创建仓库（版本库）：git init （生成一个隐藏的文件.git ）

b. 将文件添加到缓存

语法：git add <file>

git add命令可将该文件添加到缓存（暂存区在.git中生成index文件）

git add . (所有文件加之缓存)

git status -- 查看状态

git reset HEAD -- 将暂存区内的缓存资源清空

c. 将暂存区缓存资源提交到版本库：

语法 git commit -m "message"

git log -- 查看提交的版本日志

d. git reflog(查看版本号）

e. git reset 版本号（回到某个版本） 没有全局版本，但仍然有版本

注册账户连接远程仓库（github）----国内有时用coding.net：

a. 本地创建 ssh key（在c盘用户里的.ssh文件）

本地命令：ssh-keygen -t rsa -C "youremail@example.com"

youremail@example.com为 github 注册邮箱

b. 将 rsa 公私(pub)钥对中公钥文本内容复制到 github 账户配置中，创建 github 服务器上的 ssh key （在电脑用户中生成一个.ssh文件中的id\_rsa私钥信息，和公钥信息id\_rsa.pub）

c. 测试（判断连接是够成功）：

ssh -T [git@github.com](mailto:git@github.com)

git remote add origin https://github.com/fengxiaolan/music.git

git push -u origin master

可在忽略.git下改变url为ssh还是git

前三步配置，在同一台电脑上只需要做一次

d. 在 github 上创建远程空白仓库 （在github账户中加一个repository）

e. 将本地仓库与远程仓库建立关联：（点击已经存在的仓库）

复制第一句，例如

git remote add origin git@github.com:itrainhub/h51705.git

f. 将本地仓库内容推送到远程仓库：

复制第二句，例如

git push -u origin master

之后提交的同步到网页上直接git push

工作流程：

a. git clone + （clone or down中的协议ssh的协议）-- 第一次克隆远程仓库的版本库资源

git pull -- 将远程仓库中的版本资源更新同步到本地仓库

b. 在工作空间中修改、添加、删除资源

c. 在本地提交版本库

git add .

git commit -m "xxxxx"

d. 将本地版本库推送到远程仓库

git push 命令用于将本地分支的更新，推送到远程主机

git pull命令的作用是：取回远程主机某个分支的更新，再与本地的指定分支合并，它的完整格式稍稍有点复杂

用tortoisegit 的clone 需要用http协议（这个有时会用用户名密码）

在git bash here 中需要ssh

（在.git 中的config可以修改url）

touch .gitignore (忽略某个文件）

RequireJS -- 模块化开发

http://requirejs.org/

RequireJS是一个非常小巧的JavaScript模块载入框架

菜鸟：http://www.runoob.com/w3cnote/requirejs-tutorial-1.html

http://www.runoob.com/w3cnote/requirejs-tutorial-2.html

模块化开发规范：

AMD（异步模块定义）

配置的话路径有问题，可以把项目改为根目录http-server

AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。

CMD（通用模块定义）

CMD 是 SeaJS (玉伯)在推广过程中对模块定义的规范化产出。

CommonJS

BravoJS

requireJS：

1.引入require.js

2.定义函数模块module1.js🡺 define(function(){

function $(selector){

return document.querySelectorAll(selector);

}

function css(element, attr) {

return getComputedStyle(element)[attr];

}

return {

$ : $,

css : css

}

})

3．配置文件config.js🡺require.config（{

baseUrl:’/’

path:{

“jquery”:”lib/jquery/jquery-1.12.4.min”,

“cookie”:”lib/cookie.min”,

“fly”:”lib/fly.min”

“module1”:”js/module”

}，

shim:{

“fly”:{

deps:[‘jquery’]

}

}

}）

4.请求文件index.js🡺require([‘config’]，function（）{

require([‘jquery’,” 依赖模块js/module”,”cookie”]，function（）{

//待完成业务代码

})

})

注：cookie遵循jquery规范

fly不遵循jquery规范，需要配置垫片shim

PC项目：

技术点：

HTML+CSS、H5+C3

原生JS

jQuery

SASS

GULP

GIT

requireJS

artTemplate

页面：

首页

分类导航

轮播图

Tab切换

楼层导航

.......

登录

注册

表单验证

验证码

.......

列表/专题

模板

排序

分页

.......

详情

放大镜

Tab切换

吸顶

加入购物车

....

购物车

购物车商品信息维护

cookie

....

确认订单

弹出层：收货地址：省市级联

cookie

商品确认

........

规划：

https://github.com/itrainhub?tab=repositories

业务：

项目结构：

## 三十三、bootstrap

bootstrap

Bootstrap 是最受欢迎的 HTML、CSS 和 JS 框架，用于开发响应式布局、移动设备优先的 WEB 项目。

Bootstrap 提供了一套功能丰富的CSS、JS插件

全局CSS：

.container -- 固定宽度

.container-fluid -- 100%宽度

.row -- 栅格系统行

.col-xs/sm/md/lg-\*

.col-xs/sm/md/lg-offset-\*

表格：

.table

.table-striped

.table-bordered

.table-hover

表单：

.form-control

.form-group

.....

组件