正则表达式:

1. "ryan is not a good man" 变为"ryan5 is6 not7 a8 good9 man10"

var i=5;

"ryan is not a good man".replace(

/\b([a-zA-Z]+?)\b/g,

function($1){return $1+i++}//也可将$1换成match

);

数组降维:

1. 嵌套循环

Array.prototype.reduceDimension=function(){

for(var r=0,reduce=[];r<this.length;r++){  
 for(var c=0;c<this[r].length;c++){

reduced.push(this[r][c]);

}

}

return reduced;

}

1. 单层循环

Array.prototype.reduceDimension=function(){

for(var r=0,reduce=[];r<this.length;r++){

reduced=reduced.concat(this[r]);

}

return reduced

}

1. apply

Array.prototype.reduceDimension=function(){

return Array.prototype.concat.apply([],this)

}

函数，作用域，声明提前，闭包, 回调:

1.

function f1(){

var n=999;

nAdd=function(){n+=1}//全局

function f2(){ alert(n); }

return f2;

}

var result=f1();

result(); //999

nAdd();

result(); //1000

2.

f=function(){return true;};

g=function(){return false;};

(function(){

if((g()&&[])==![]){//false

var f=function(){return false;};

function g(){return true;};

}

})();

console.log(f()); //true

3.

var f=function(){ var a=b=1; };

f();

console.log(b); //1

console.log(a); //报错

如果改为:

var f=function(){ var a=b=1; };

setTimeout(f,0);

console.log(b);//报错

如果改为：

var a,b=0，fn=function(){ var a=b=2; }

fn();

console.log(a);//undefined

console.log(b);//2

4. 判断是否是质数：

循环方式：//都是算数计算，效率反而高

function isPrime(a) {

if(a<=3){ return true; //素数不小于等于3

}else{

for(var i=2;i<=Math.sqrt(a);i++){

if(a%i==0){ return false; } //若能被整除，说明不是素数

}//遍历结束后，都没碰上能整除的，说明是素数

return true;

}

}

正则方式： //代码简单，但因为又频繁字符串拼接，所以效率低。

function isPrime(i) {

var ones = "";

while(--i >= 0){ ones += "1"; }

return !/^1?$|^(11+?)\1+$/.test(ones);

}

alert(isPrime(7));

实现函数记忆：

什么是函数记忆: 利用闭包原理，让函数记住已经计算过的数据，从而避免重复计算，而提高程序执行效率的技巧。

见1\_memorize.html

5. 数组去重复/获得数组中不重复的数字:

见exercise09\_01.html

6. 查找字符串中出现次数最多的字符，及其出现次数

var str="Hello World";

arr=str.split("").sort().join("").match(/(.)(\1)+/g).sort(function(a,b){return b.length-a.length});

console.log(arr[0][0]+":"+arr[0].length);

7. 用正则表达式，获得一句话中首字母和末尾字母相同的所有单词。

/\b([a-zA-Z])(.\*?)\1\b/g

8. map方法的实现原理

9. 函数绑定: bind方法的实现原理

Function.prototype.bind=function(context/\*,args\*/){

var fun=this;

var args=Array.prototype.slice.call(arguments,1);

return function(/\*otherArgs\*/){

var otherArgs=Array.prototype.slice.call(arguments);

var finalArgs=args.concat(otherArgs);

return fun.apply(context,finalArgs);

}

}

10. 函数柯里化: 基于现有函数，创建一个已经设置好部分参数的新函数

类似于函数绑定。但不绑定this。

鄙视题: 实现alert(add(1)(2)(3)); //6 alert(add(1)(2)(3)(4)); //10

何时使用: 只要希望先后为函数传入不同参数，反复调用时。

如何使用:

1. 外层函数返回内层函数
2. 内层函数将外部变量和内部变量联合操作。
3. 内层函数，再返回内层函数本身，便于下次调用。
4. 如果序号输出结果，则重写内层函数的toString方法。

function add(x) { //函数柯里化

return function (y) {

x+=y;

arguments.callee.toString=function(){return x; }

return arguments.callee

};

}

alert(add(1)(2)(3)); //6

alert(add(1)(2)(3)(4)); //10

11. 号称闭包最难的题

function fun(n,o){

console.log(o);

return{

fun:function(m){

return fun(m,n);

}

};

}

var a=fun(0);a.fun(1);a.fun(2);a.fun(3);

var b=fun(0).fun(1).fun(2).fun(3);

var c=fun(0).fun(1);c.fun(2);c.fun(3);

12. 闭包

var a=0,b=0;

function A(a){

A=function(b){

alert(a+b++);

}

alert(a);

}

A(1);执行到这句：全局变量A-->function(b){ alert(a+b++); } 其中a是闭包受保护的局部变量，值为1。此次输出1

A(12); 执行到这句，其实执行的是function(b){ alert(a+b++); }，其中a是闭包中的1，b是传入的12，其实计算的是1+ 12++,结果为13.

13. 递归

var emp={work:function(){

var sum=0;

for(var i=0;i<arguments.length&&arguments[0]>0;i++){

sum+=arguments[i]+arguments.callee(--arguments[i]);

}

return sum;

}

}

console.log(emp.work(3,2,1));//10

14.

window.a=300;

function fn1(){

this.a=100;

this.b=200;

return function(){alert (this.a)}.call(arguments[0]);

}

function fn2(){

this.a=new fn1();

}

var a=new fn1().b;

//a=200;

var v=new fn1(fn2());

//a={a:100,b:200}

15.

function Human(){

var c=function(){var a=200}

return c;

}

var kang=new Human();

var feng=Human();

var x=Human();

console.log(feng==x)//false

console.log(kang===feng);//false

16

var arr=[];

function fun(){

for(var i=0;i<4;i++){

var x={};

x.no=i;

x.text=arr[i];

x.fun=function(){alert(i)}

arr.push(x);

}

}

fun()

arr[0].fun(); arr[1].fun(); arr[2].fun(); arr[3].fun();

函数+面向对象:

1.

var setObj=function(o){

o.name="xiaoming";

o={};

o.name="xiaohong";

}

var p={ name:"xixi", age:24 }

setObj(p);

console.log(p);

2.

var number=2;

var obj={

number:4,

fn1:(function(){

this.number\*=2;

number=number\*2;

var number=3;

return function(){

this.number\*=2;

number\*=3;

alert(number);

}

})(),

db2:function(){this.number\*=2}

}

var fn1=obj.fn1;

alert(number);//4

fn1();//9

obj.fn1();//27

alert(window.number);//8

alert(obj.number);//8

3.

function foo(){return this.a}

var a=2;

var o={a:3,foo:foo}

var p={a:4}

console.log(o.foo());//3

console.log((p.foo=o.foo)());//2

console.log(p.foo());//4

4.

function MyObj(){this.p.pid++;}

MyObj.prototype.p={"pid":0};

MyObj.prototype.getNum=function(num){return this.p.pid+num;}

var \_obj1=new MyObj();

var \_obj2=new MyObj();

console.log(\_obj1.getNum(1)+\_obj2.getNum(2));

5.

var a=2;

var obj={x:1,y:{z:2}};

var n=[obj,3,[4,5]];

console.log(a<<2);//8

console.log(obj["y"].z);//2

console.log(n[0].y["z"]);//2

console.log(n[2],["1"]);//[4,5]["1"]

6. 对面向对象的理解

什么是面向对象？面向对象就是在程序中，用一个对象描述现实中一个事物

为什么使用面向对象？因为面向对象更接近于人的思维方式，更便于代码维护。

面向对象三大特点：封装，继承，多态

封装：讲现实中一个事物的属性和方法，集中定义在程序中的一个对象中。

为什么要封装：更接近于人的想法，便于代码维护。

继承：父对象中的成员，子对象可以直接使用。

为什么要继承：代码重用！节约内存空间

多态：同一样东西，在不同情况下表现出不同的状态。

多态方式：重写，重载：

重载：多个同名方法，但参数列表不同，调用时，可根据传入参数的不同，动态决定调用何种匹配的方法。

为什么使用重载：便于调用，减少调用者负担。

js语法不支持重载，但可用arguments对象模拟重载效果。

重写：子对象觉得父对象的成员不好用，可在本地定义与父对象同名的成员，覆盖父对象的成员

为什么使用重写：专门定义子对象与父对象之间的差异。

7.

function MyObj(){ //构造函数，每创建一个构造函数，自动调用一次

this.p.pid++;

}

MyObj.prototype.p={"pid":0} //原型中的共有属性，所有对象共用

MyObj.prototype.getNum=function(num){

return this.p.pid+num;

}

var \_obj1=new MyObj(); //MyObj.prototype.p.pid =1

var \_obj2=new MyObj(); //MyObj.prototype.p.pid =2

console.log(\_obj1.getNum(1)+\_obj2.getNum(2)); //7

8.

function Human(){

return Human;

}

var boy=new Human();

var man=Human();

boy===man;?

如何让boy===man成立？

9.

var length = 10;

function fn() {console.log(this.length);}

var obj = {

length: 5,

method: function(fn) {

fn();

arguments[0]();

}

};

obj.method(fn, 1); //10 2

10.

var foo=function(){

console.log(this.a);

}

var obj={a:2,foo:foo};

var a=10;

var bar=obj.foo;

var bar2=foo.bind(obj);

bar();

bar2();

foo();

obj.foo();

setTimeout(bar,0);

11.

var a=1;

var b={

a:2,

b:function(){

console.log(this.a);

}(), //匿名函数自调，第一个就输出1

f:this.f=function(){

console.log(this.a);

}

};

function f(){ console.log(3); }

f();

b.f();

(b.f)();

(0,b.f)();

12

function Foo(){  
getName=function(){alert(1);};

return this;

}

Foo.getName=function(){alert(2);};

Foo.prototype.getName=function(){alert(3);};

var getName=function(){alert(4);};

function getName(){ alert(5); };

Foo.getName();//2

getName();//4

Foo().getName();//1

getName();//1

new Foo.getName();//2

new Foo().getName();//3

new new Foo().getName();//3

13.

var x=5,o={

x:10,

doit:function doit(){

var x=20;

setTimeout(function(){alert(this.x)},10);

}

}

alert(o.doit())//undefined 5