## 6月29日

## 08 JSCore-Day01

1.什么是JavaScript

前端三大语言:

HTML: 是专门编写网页内容的语言

CSS: 是专门编写网页样式的语言

JavaScript: 专门编写交互行为的语言

交互:

1.用户输入数据，操作页面元素

2.程序接收并处理数据

3.程序返回处理结果

JavaScript:三大部分

ECMAScript:ECMA组织制定的JavaScript语言标准。——核心语法

JavaScript:Netscape参照ECMAScript标准实现的JavaScript语言

JScript:MS参照ECMAScript标准实现的JavaScript语言

DOM:专门操作网页内容的API ——3

BOM:专门操作浏览器的API ——2

JavaSCript:四大特点

1、运行在脚本解释引擎中

引擎：浏览器中执行专门任务的小软件

内容排版引擎：负责解析HTML和CSS，生成网页内容和样式。

脚本解释引擎：负责解释并运行 JS脚本程序。

多数JS运行在客户端的脚本引擎中

2、解释执行：

默认自上向下，边解释边执行。

编译执行：

编译：在执行程序前，自动检查错误，优化代码。

3、弱类型：

1、变量在声明时不需要提前指定数据类型

2、一个变量先后可保存不同类型的数据

3、隐式转换

4、基于/面向对象

2、变量

什么是变量：内存中存储一个数据的存储空间，再为其起一个名字

何时使用变量：只要一个数据可能被反复使用，都要先保存在变量中，再反复使用变量

如何使用变量：

声明：在内存中创建一个存储空间，再起一个名字。

如何声明：var 变量名;

同时声明多个变量：var 变量1,变量2,...;

仅声明还未赋值的变量，默认为undefined。

规定：所有变量使用前，必须先声明。

赋值：将一个数据保存到变量中

如何赋值：变量 = 值;

强调：等号左边只能是一个变量，左右顺序不能颠倒

同时为多个变量赋值：变量1 = 值1 ,变量2 = 值2 ,变量3 = 变量4 = 值3;

如果给一个未声明过的变量赋值，js会在全局范围内自动创建同名变量--严格禁止该做法

强烈建议：在声明变量的同时，就初始化变量的值

var 变量 = 值;

取值：找到变量，并取出变量中的值使用

在表达式的任何位置，只要读到变量名，js就会找到变量取出值，代替变量所在的位置。

如果表达式中试图使用一个从未声明过的变量，会报错：

ReferenceError:xxx is not defined

【**笔试题**】：var a,b = 10; //a:undefined b:10

声明提前：hoist

在开始执行程序前，都会将var声明的变量，集中到程序顶部创建，

然后才开始执行程序，赋的值留在原地

【**笔试题**】：consolg.log(a);

var a = 10;

console.log(a); //不会报错

【补：控制台的操作】：

1、多行编写：shift+回车

2、控制台中可记录之前执行过多的所有语句：

用 上下键 切换新旧代码

3、清屏：左上角园叉

设置Chrome的起始页面为空白页面：

3、数据类型

什么是：数据在内存中的存储格式

为什么：不同类型的数据，能执行的操作也不同。

包含：2大类：

1、原始类型：数据直接保存在变量本地的数据类型

包括：5个

number:js中保存一切数字的类型。即可保存整数，也可保存浮点数（小数）

何时使用：如果一个数字经常用于算数计算或比较大小，就要保存为数字类型

如何：程序中不带引号的数字直接默认就是number。

存储空间：整数占4字节

浮点数占8字节

【舍入误差】：计算机中也有计算不尽的数值。使用计算不尽的数值执行计算时，产生的误差，就叫舍入误差。

不可避免，只能近似解决：

n.toFixed(2)//将n按两位小数四舍五入

string:保存一串字符序列的数据类型

何时使用：仅用于显示的内容，就存为字符串。

如何使用：在程序中凡是引号包裹的字符都是字符串。

存储空间：js内存中，用unicode编码来表示每个字符：

unicode:人为的为全球主要语言中的每个字编一个号。

为什么：计算机不认字符只认数字

返回：汉字的编码范围：19968-40869

一个汉字占2字节

数字的编码范围：48-57

大写字母：65-90

小写字母：97-122

一个字母或数字占1字节 一个汉字占2个字节

boolean:只有两个值的数据类型

true false

何时使用：一般用作判断的结果

如何使用：只要不加引号的true或false，默认都是bool类型。

undefined:表示一个变量声明后，从未被赋值 -- 空。

专门用于程序自动初始化变量的默认值。

null:表示一个变量暂时未引用其他变量或保存其他数据 -- 空。

专门用于程序员主动清空一个变量的内容。

其实undefined和null在本质上是一样的

2、引用类型：数据无法直接保存在变量本地的数据类型

1、数据类型转换

隐式转换：无需程序员干预，js自动完成的类型转换。

强制转换：由程序员主动调用API实现的类型转换。

何时强制转换：只要隐式转换的结果不是想要的，就要强转。

分为：

补：typeof(x) -> 返回变量的数据类型

1、x to string

【var str = String(x);】 //万能

var str = x.toString(); //不能转null和undefined

eg:

function String(x){

if(x===undefined){

return "undefined";

}else if(x===null){

return "null";

}else{

return x.toString();

}

}

2、x to number:

【var num = Number(x);】

【只能转换纯数字组成的字符串】或

【bool类型(true -> 1 false -> 0)】

何时使用：将不确定的类型，转换为字符串时。

str to number:

【var num = parseInt(str);】

从str的开头依次读取每个字符。

跳过开头的空字符，仅读取'纯数字内容'。

直到碰到第一个不是数字的字符退出

缺点：'不认识小数点' -- 去掉小数部分

解决：

【var num = parseFloat(str);】

原理和parseInt完全一样

差别：'认识第一个小数点'

建议：【**首选parseFloat**:】

1、不会丢弃小数部分。

2、如果可以转为整数，parseFloat优先转为整数！

除非明确地希望去掉小数部分时，才用parseInt。

如果无法转为数字，都返回NaN：

NaN:not a number

typeof(NaN); //number

代表所有非数字的字符 -- 范围

参与算术运算，结果永远等于 NaN；

参与关系运算，不等于、不大于、不小于一切值。

NaN == NaN //false

3、x to bool:

【**var bool = Boolean(x)**】

规定：只有5个值会被转成false,其余一律转为true：

"" 0 NaN undefined null 2、【\*\*\*】运算符和表达式

程序：人的想法在计算机中的执行

运算符：程序中模拟人的想法的特殊符号

表达式：由数据、变量和运算符组成，执行一项任务的程序语句。

算数运算：+ - \* / %

% : m%n :m/n,取除不尽的余数部分。

何时使用%：

1、判断能否被整除时

2、限制运算的结果，不能超过某个值

m%100 -> 最大值99

隐式转换：

算数运算中

默认情况下：一切都转为数字，再算数运算

特例：+ 运算中，只要碰到一个字符串，就都转为字符串，

+ 法计算变为字符串拼接。

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

关系运算：作比较和做判断

> < >= <= == !=

'隐式转换：一切都转为数字再比较'

特例：

1、参与比较的两个值都是字符串

此时不再转换为数字

而是按位PK每个字符串的unicode号。如果前几位完全一样，就比长短。

2、判断NaN：

问题：NaN无法用普通的 == 判断

解决：【**var bool = isNaN(num);**】

如果返回true，说明num是NaN

返回false，说明不是NaN

反过来用：判断num是否是一个数字

【**var bool = !isNaN(num);**】

返回true:说明num是数字

返回false:说明num不是数字

3、 问题： == 无法区分null和undefined

解决：全等：=== ，不带隐式转换的 ==

首先要求类型系统，然后值再相同，才是全等

何时使用：

1、只要区分null和undefined

2、只要规定等于比较时，类型也必须相等，就用全等。

4、如果参与等于比较的两个数据，都是引用类型的对象，

则不进行任何转换，只比较两对象的地址

[1,2,3,4] == [1,2,3,4] //false,两个数组的内存地址不同

var a = {};

var b = {};

a == b; //false

比如：[]==[] false

逻辑运算：综合多个关系运算的结果，得出最终结论

&&：逻辑与，读作“且”

eg: 条件1 && 条件2

只有条件1和条件2都为true时，结果才为true。

只要有一个条件为false，结果为false

【**笔试题**】：

alert(4&&5); //5 系统先对4进行隐式转换Boolean(4)，为true之后再判断Boolean(5)，也为true，所以结果为后面的true，

alert(0&&5); //0

alert(4||5); //4

alert(0||5); //5

||: 逻辑或，读作“或”

条件1 || 条件2

只要任意一个条件为true，结果为true

除非所有条件都为false，结果为false

!: 逻辑非

颠倒判断的结果

运算符优先级：

优先级越高的运算符，优先执行

决定了js断句的顺序和位置

避免死记优先级？只要需要先执行的代码都加()

【【**短路逻辑**】】：

如果前一个条件已经可以得出最终结论，则后续条件不再执行。

&&：若前一个条件为false，则直接得出结论：false

利用此，可实现简单分支结构

||：若前一个条件为true，则直接得出结论：true

利用此，可实现简单选择结构

【**笔试题**】：

var n = 10;

n<10&&m>10; //false

n>=10&&m<10; //报错 ReferenceError: m is not defined

n<10||m<10; //报错 ReferenceError: m is not defined

n>=10||m<10; //true

**如何利用短路逻辑**：

&&：简单分支：

即一个条件一件事，满足就执行，否则什么也不做

条件%%操作：

只有满足”条件“时，才执行”操作“

eg:

商品打折

var price = 600;

price>=500&&(price=price\*0.8);

console.log(price);

||: 在两个值中挑一个有效的使用

值1 || 值2

Boolean(值1)

如果值1转为bool后为true，就用 值1

如果值1转为bool后为false，就用 值2

eg2:

输出回复内容

var reply = prompt("请输入回复内容");

alert(replay || "主人很懒，什么也没留下");

位运算：

左移和右移：

m << n : 即左移，即m\*2的n次方

相当于：m\*Math.pow(2,n);

m >> n : 即右移，即m/2的n次方

相当于：m/Math.pow(2,n);

取整：

m>>>0 m^0 m|0

相当于：parseInt(m);

扩展赋值运算：对特定赋值运算的【简写】

如果取出变量m中的值做计算，再将结果放回到m中保存时，就可以扩展复制运算简写

m=m+n => m+=n 将n累加到m中

m=m-n => m-=n

m\*=n

m/=n

m%=n

如果每次只递增1或递减1

m++ => m+=1 => m=m+1

m-- => m-=1 => m=m-1

当递增递减运算符参与到另一个表达式内时，++和--有前后之分

【++m 和 m++】

相同：m（原值）的值一定 都加1

不同：前++，返回加之后的新值

后++，返回加之前的旧值

作业：C语言实例100题：T2、T3、T5

## 6月30日

## 08 JSCore-Day02

1、【\*\*\***函数**】：

1、什么是函数：封装一项任务步骤清单的代码段，再起一个名字

2、为什么使用函数：一项任务可能会被反复执行，而直接顺序编写的代码，无法重复利用，不便于维护。

3、何时使用函数：只要一项任务 可能被反复执行时，

都要先将任务定义在一个函数中，再反复调用函数。

4、优点：代码重用

5、如何定义：

声明：在内存中创建一个函数对象，来封装函数的定义

function 函数名(参数列表){

步骤清单;

return 返回值;

}

6、三要素：

1、 参数：接收函数执行时必要数据的变量

何时定义参数：当一项必须某些数据才能正常执行时，都要定义对应的参数。

如何定义参数："在函数名后的()中，不要var即可直接定义参数变量名"。每个参数变量之间用逗号分隔。

作用：1、提示调用者必须提供哪些数据才能正常执行

2、让函数变得更灵活

2、步骤清单

3、返回值：函数的执行结果

何时定义：如果调用者需要获得函数的执行结果时。

如何定义：return 返回值

退出函数

函数是一个"引用类型的对象"：

声明：1、用函数名定义一个变量；

2、创建函数对象，封装定义函数；

3、让函数名变量引用函数对象

调用：找到函数，并按照函数的步骤清单执行程序。

如何调用：

var 返回值 = 函数名(参数值列表);

参数值列表，尽量要和声明时的参数值列表中的个数和书序一致。

return只负责返回结果，不负责保存。

7、【\*\*\***作用域**(scope)】：一个变量的可用范围

2种：

1、全局作用域：window

全局变量：保存在全局作用域中的变量

特点：随处可用，可反复使用

2、函数作用域：

局部变量：在函数调用时，保存在函数作用域中的变量

特点：仅在函数内可用，不可重复使用

作用域规定了变量使用的顺序和优先级：

只要局部有，就不用全局的

局部没有，才去全局找

全局也没有就报错

函数调用的过程：

1、引擎找到函数定义

2、创建函数作用域

在函数作用域中保存局部变量

3、调用时，优先使用函数作用域中的局部变量。

除非局部没有，采用全局的。

4、函数调用后，函数作用域释放

其中的局部变量一同释放。

eg:

//例1、

var a=10;

function fun(){

a++; //借用全局的a,把全局的变量a递增了1

console.log(a); //11

}

fun(); //11

console.log(a); //11,全局变量已经被修改了

//例2、

var a=10;

function fun(){

var a=20;

a++; //局部

console.log(a); //21

console.log(c);

}

fun(); //21

console.log(a); //10,全局变量

//例3、

var a=10;

function fun(a){ //带参后，相当于在函数内

隐式声明了一个变量a，是局部变量

a++; //但未赋值，所以输出NaN

console.log(a);

}

fun(); //NaN

console.log(a); //10

//例4、

var a=10;

function fun(a){ //带参后，相当于在函数内

隐式声明了一个变量

a=20; //赋值

a++;

console.log(a);

}

fun(); //21

console.log(a); //10

8、【**声明提前**(hoist)】：

在正式执行程序前，都会将所有"var声明的变量和 function 声明的函数

提前到"当前作用域的顶部"集中创建

eg:

//例1、

var a=10;

function fun(a){ //传参,把a传了过来

console.log(a); //10 全局

a=100;

console.log(a); //100 局部

}

fun(10); //10 100

console.log(a); //10 全局

//例2、

var a=10;

function fun(){ //未传参

console.log(a); //undefined

var a=100;

console.log(a); //100

}

fun(); //undefined 100

console.log(a); //10

//例3、

经过运行测试，可以得到正确结果的两端代码段：

代码段： 恢复后的代码段：

fun(); //2 =>var fun; //仅声明提前

fun(); //2 =>function fun(){console.log(1);}

function fun(){console.log(1);} => functionfun(){console.log(2);}

function fun(){console.log(2);} => fun(); //2 因为函数在 执行后就被销毁了，所以函数1已经不存在了，执行了两遍函数2

var fun=100; =>fun(); //2

fun(); //TypeError: =>fun=100;//赋仍然保留在原位

//fun is not a function =>fun(); //TypeError:fun is not a function

9、**按值传递**:两个变量间赋值时，或者"将变量作为函数的参数"传入函数时。

其实，只是"将【原变量中的值】"复制一个副本给对方

后果：

【**原始类型的值**】：修改新变量，不影响原变量；

eg1:

function buy(card){

card-=3;

console.log("余额 "+card); //7

}

var card=10; //var声明的变量会被提前

buy(card); //7 "按值传递，即将实参变量的值复制一份副本给方法的形参变量"

console.log(card); //10

【**引用类型的对象**】：复制的是"【对象地址】"的一个副本，原对象依然只有一个。

结果：新旧变量同时引用同一个对象。

结果任何一方修改，另一方都受影响。

eg2:

function eat(baos){

baos.length--;

console.log(String(baos));

}

var baos=["包","包","包","包","包",]; //数组，

eat(baos); //"包","包","包","包"

console.log(String(baos)); //"包","包","包","包"

2、**全局函数**

ES标准中规定的，浏览器厂商已经实现的，不需要任何对象就可直接调用的函数

W3C离线手册-->JavaScript-->JavaScript对象-->JS Functions 中

parseInt();

parseFloat();

isNaN(num);

... //alert(); 不是全局函数，属于 BOM

//getElementById(); --> DOM

其他全局函数：

**编码解码**：(不是加密)

问题1：

URI标准不允许使用"多字节字符"，如果使用会乱码。汉字属于多字节字符

解决：（所以要）将多字节字符编码为单字节字符

【采用utf-8编码：一个汉字 3字节】

如何编码：

var code=encodeURI(str);

如何解码：

var code=decodeURI(str);

eg:

var wd=prompt("百度一下");

alert("http://www.baidu.com/s?wd"+encodeURI(wd));

alert("服务器解码"+'\n'+"http:www.baidu.com/s?wd="+decodeURI(wd));

问题2：

URI标准还不允许使用"保留字符"：

比如： : /

encodeURI 和 decodeURI 无法编码解码保留字符

解决：

encodeURIComponent() 编码

decodeURIComponent() 解码

既可编码保留字符，又可编码多字节字符

eval：执行字符串格式的js代码

isFinite：判断一个数值是否在有效范围内

js中除数为0，不会报错，而是返回infinity。

判断一个数是不是无穷大：

isFinite(num); 返回true，说明num 在有效范围内

返回false，说明num是无穷大

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

/\*\*\*利用分支结构的特点简化代码：短路逻辑、三目运算符、switch case语句\*\*\*\*\*/

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

3、**分支结构**：

什么是分支结构：让程序根据不同的条件执行不同的操作。

包括：

1、一个条件，一件事，满足就做，不满足就什么也不做。

如果操作简单：利用短路逻辑实现：

短路：条件&&操作 即只有满足"条件"，才"操作"

如果操作复杂：使用if结构：

if(条件){

操作;

}

2、一个条件，两件事，二选一执行

如果操作简单：**三目**（三元/条件）**运算**

如何使用：条件?操作1:操作2;

如果满足条件，执行操作1，否则执行操作2

"如果根据不同的条件选择不同的值"时，也可以：

三目：条件?值1:值2

如果满足条件，就用值1，否则用值2

eg：【【**判断平年闰年**】】

console.log((y%4==0&&y%100!=0||y%400==0)?"闰年":"平年");

如果操作复杂：

if(条件){

满足条件才执行的代码段

}else{

不满足条件才执行的代码段

}

3、多个条件，多件事，多选一执行：

如果操作简单：三目

条件1?操作1:

条件2?操作2:

...? ...:

默认操作; //【该步不能省略】，不同于下一个复杂操作中的 步骤

eg：

计算可莱托指数

function clt(h,w){ //传入两个参数，身高体重

var r=w/(h\*h);

console.log(r>25?"胖":r<20?"瘦":"正常");

}

clt(1.75,64);

如果操作复杂：

if(条件1){

操作1;

}else if(条件2){

操作2;

}else if(...){

...

}else{ //该步也可省略

默认操作;

}

4、如果所有条件都是等于比较时：

switch case

如何使用：

switch(表达式){

case 值1:

操作1;

case 值2:

操作2;

case ...:

.... ;

default: //可省略

默认操作

}

执行原理：

先计算表达式的值，用该值依次和每个case后的值做 全等===比较。

如果表达式的值 全等于 某个case的值，则进入该case下执行内部的操作

默认情况下，执行完当前case中的操作后，会继续触发之后所有的case操作

解决：

break; //退出当前结构

一般在每个case之间用break阻断传导。

何时不需要break？

1、最后一个操作后，不加 break

2、相邻两个case执行相同的操作时，不加break。

注意：

switch case 和 else if

前者 仅用于等于比较的条件

后者可用任意条件

if (Boolean(条件)) else if()

自带隐式转换，将条件转为boolean，效率稍微低

switch case

不带隐式转换的等于比较，效率稍微高。

作业：c语言100题：T1、T11、T12、T13、T14、T16、T21、T22、

## 7月1日

## 08 JSCore-Day03

## 1、\*\*\***循环**

## 1、什么是循环：让程序反复执行同一段代码

## 2、如何使用：

## 三要素：

## 1、循环条件：让循环可以继续执行的条件。

## 2、循环变量：在循环条件中，用作比较的变量。

## 循环变量总是向着不满足循环条件的趋势变化。

## 3、循环体：循环反复执行的代码段。

## 三种循环：

## 1、当"循环变量的变化没有规律时"

## while：

## var 循环变量 = 值;

## while(循环条件){

## 循环体;

## 迭代变化循环变量

## }

## 2、先执行循环体，再判断循环条件，决定是否继续。

## var 循环变量 = 值;

## do{

## 循环体;

## 迭代变化循环变量;

## }while(循环条件);

## 区别：

## 只看第一次循环条件是否满足

## 如果第一次的条件都满足，则两者完全等效

## 如果第一次的条件不满足，则：

## while一次都不执行，

## do while至少可执行一次

## 总结：退出循环

## 1、用循环条件控制退出 -- 优雅，但是难度大 -- 仍然建议使用

## 2、用break退出 -- 野蛮，但是随意，简单

## 3、当循环条件的变化有规律时：

## for(var 循环变量=值;循环条件;迭代变化循环变量){

## 循环体;

## }

## eg：

## //1-100累加计算

## var sum=0;

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

## sum+=i;

## }

## console.log(sum);

## 特殊写法：

## 1、在第一部分中：可同时声明并初始化多个变量

## 2、第三部分中，可同时执行多个短小的操作，用逗号分隔，

## eg：

## //利用上述写法简化代码，实现1-100的累加计算

## for(var sum=0,i=0;i<=100;sum+=i++);//尤其注意;不能省略

## console.log(sum);

## 3、死循环：while(true) for(;;)

## eg：（见demo-02）

## var count=0;

## for(var n=0;n<5;y+=n==0?1:4){ //n是输出润年的个数， 输出5个即可

## count++;

## (y%4==0&&y%100!=0||y%400==0)&&(n++,console.log(y)); //若果是闰年，就 把n+1，并且输出该y

## }

## console.log(count);

## 【补】：其实，if、else、for、while之后只有一句话

## 可省略{} -- 禁止使用

## 

## 复杂循环:2步:

## 1. 截取片段，总结公式

## 2. 用循环，反复调用公式

## 2、\*\*\***数组**

## 1、什么是数组：

## 内存中连续存储多个数据的存储空间，在起一个名字。

## 数组是 引用类型 的对象

## 2、为什么：

## 程序 = 数据结构 + 算法

## 算法：解决问题的步骤

## 数据结构：数据在内存中的组织结构。好的数据结构可以极大提高程序的执行效率

## 3、何时使用：存储多个相关的数据时，用数组二不用变量

## 4、如何使用数组：

## 1、创建数组：3个场景

## 1、创建一个空数组：2种：

## var arr=[]; //创建一个新数组

## var arr=new Array();

## 创建 数组

## 何时使用：创建数组，不确定数组元素的内容。

## 2、创建数组同时，初始化数组元素：

## var arr=[值1,值2,值3,...];

## var arr=new Array[值1,值2,值3,...];

## 创建 数组 向数组中加入...

## 何时使用：创建数组时，已经知道每个元素的内容。

## 3、创建n个空元素的数组：

## var arr=new Array(n);

## 何时使用：创建数组仅知道将来元素个数不确定元素内容时。

## 2、访问数组元素

## 先找到数组的变量，再使用下标在数组内找到指定位置的元素。

## 下标：数组中唯一标识一个元素的存储位置的序号。默认从0开始， 连续递增不重复

## 比如：arr[i] 访问arr数组中下标为i位置的元素

## 每个元素的用法和普通变量，完全一样。

## 数组 又称为 一组变量的集合，再起一个统一的名字

## 【补】: **垃圾回收**：清除不再使用的对象，释放存储空间。

## 对象：内存中存储多个数据的存储空间。其占用内存比普通变量大

## 垃圾回收进程（器）：是一个专门负责释放不再使用的对象的小程序。

## 是js引擎自带的自动执行的后台程序

## 对声明周期：（垃圾回收的原理）

## 1、创建对象时，垃圾回收器会标记该对象的引用计数器为1

## 引用计数器：记录一个对象被几个变量引用着。

## 2、使用对象过程中，只要多一个变量引用该对象，计数器就+1

## 3、当主动为一个变量赋值为null时，就释放了该变量对对象的引用。计数器就-1

## 4、如果一个对象不再被任何变量引用，计数器就变为0，垃圾回收器就释放对象。

## 建议：在使用完一个较大的对象后，都要主动用null释放对该对象的引用。

## 强调：【"**JS中数组的三大差别**"】：（三个不限制）

## 1、不限制数组元素的个数

## 可随时在任何位置添加新元素

## 只要为该位置强行赋值，数组就会自动在该位置创建一个新元素

## 2、不限制数组元素的数据类型

## 3、不限制下标越界

## "JS中数组越界，不会报错! 而是返回undefined"

## 

## 数组的length属性：表示数组中元素的个数

## 固定套路：

## 1、如何获得数组最后一个元素：

## arr[arr.length-1]

## 数组的length属性，永远是最大下标+1

## 如何获得数组倒数第n个元素：

## arr[arr.length-n]

## 2、如何在数组末尾追加新元素：

## arr[arr.length]=新值

## "稀疏数组"：

## 数组元素的下标不连续的数组。但是其length属性却始终是最大下标+1

## 3、数组缩容：

## arr.length-n

## 数组遍历：依次获取数组中每个元素，对每个元素执行相同的操作。

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

## arr[i]; //当前正在遍历的元素

## }

## 【 **索引数组**】：下标都是数字的数组（即上述常规数组）

## 【\*\*\***关联数组**】：

## 什么是：可自定义下标名称的数组

## 为什么：索引数组的下标没有实际意义

## 何时使用：只要希望每个元素由特定意义就要自定义下标名。

## 如何创建：2步：

## 1、先创建一个空数组；

## 2、向数组中添加新元素，下标要"用自定义的字符串"作为名称。

## 【"关联数组中的length属性失效，永远为0"】

## 遍历关联数组中每个属性：

## **for(var key in arr){**

## **arr[key] //当前属性值**

## **}**

## in：依次获取arr中的每个属性名

## key:用来保存in本次返回的属性名

## 关联数组，也成为**hash数组**：

## hash(散列)算法：接收一个字符串，计算出一个尽量不重复的序号

## 特点：不同的字符串，尽量算出不重复的字符串。

## 同一个字符串，算出的结果一定相同。

## 所以hash中的下标(key)不可重复，是唯一的。

## 存储原理：先将该元素的下标名称交给hash算法，计算出一个尽量不重复的存储位置。将数据保存到该位置。

## 取值：将 要查找的下标名称交给hash算法，计算出和存储时完全一致的位置。直接通过地址找到元素。

## 索引数组：查找未知位置的元素，只能靠遍历。

## 缺点：受元素个数和元素存储位置的影响极大。

## HASH数组：只要知道元素的属性名，不需要遍历，即可直接获得元素的内容。

## 优点：查找速度极快，且和元素个数和存储位置无关。

## 

## **数组API**：

## 数组："封装一组连续的数据，并提供对数据进行操作的API"

## API:

## 1、to str: 2种：

## 1、**String(arr):**将数组中每个元素转为字符串并且用逗号链接。 -- 拍照

## 2、**arr.join("连接符")：**将数组中每个元素转换为字符串，用自定义的“连接符”连接每个元素

## 

## 固定套路：

## 1、无缝拼接: var str=arr.join("")

## 2、将数组中的元素拼接为页面的HTML代码

## 2、拼接和截取："都无法直接修改原数组"，而是返回新书组，用变量接住

## 拼接：

## var newArr =

## arr1.**concat**(值1,值2,arr2,值3,...)

## 将值1，值2...拼接到arr1之后，

## 强调：concat可以打散数组类型的参数为单个元素，逐个拼接。

## 截取子数组：

## var subArr=arr.**slice**(starti,endi+1);

## "复制"出arr中starti位置开始，到endi位置的元素组成新数组返回

## 特点：

## 1、slice：含头不含尾，结束位置要+1

## 2、支持负数参数：表示倒数第i个位置。其实执行的还是length-n。

## 3、如果从starti的位置直接截取到结尾，就省略第二个参数。

## **splice**：删除元素、插入元素、替换元素

## 【"直接修改原数组"】

## 删除：arr.splice(starti,n)

## 删除arr中starti位置开始的n个元素

## 其实，splice方法会返回 被删除的元素 组成的临时数组，用于后续的操作。

## 插入：arr.splice(starti,0,值1,值2,...)

## 在arr的starti位置插入值1，值2，...，原starti位置的值及其之后的值自动向后顺移

## 但是splice与concat不同，不支持数组类型的参数，无法打散数组。（会变成二维数组）

## 替换：arr.splice(starti,n,值1,值2,...)

## 先删除starti位置的n个元素，再在starti位置插入值1，值2，...

## **reverse**：颠倒数组中的元素：【直接修改原数组】

## arr.reverse();

## 只是机械的将数组颠倒

## 3、

## 7月4日

## 08 JSCore-Day04

1、\*\*\*数组

1、\*\*\*排序：

1、**冒泡排序**：

2、**arr.sort();**

默认：将数组中每个元素都转为字符串，再升序排序。

何时：仅限于排序字符串时。

（因此会遇到的问题是：当对字符串122、23排序是，会认为23>122）

解决：

步骤：

1、【"**自定义比较器函数**"】

比较器函数：专门比较任意两数大小的函数：

规定：function 函数名(a,b){ //此处函数名:compare

如果a>b就返回正数

如果a<b就返回负数

否则返回0

即 return a-b;

}

2、将比较器函数作为参数，传入sort方法中。

arr.sort(compare);

【"强调：不加()"】

sort方法的原理：

1、"如果函数执行时缺少一段逻辑（代码段），就需要用函数类型的参数，从外部传入。"

从外部传入的函数，在内部被调用。

强调：将函数作为参数传递时，不加()。

2、js的作用域 与 java的作用域

js：2级：全局、函数作用域

会导致：js中块中声明的变量，出了块，依然可用。甚至会被声明提前到块外。

java：3级：全局、函数、块(if、for ...)作用域

java中块中声明的变量，出了块，无法使用。

3、创建函数：

如果不希望被提前，可用：

var 函数名=function(...){//原为：function 函数名(){...}

...

}

揭示了：函数其实也是一个对象

函数名其实就是引用函数对象的普通变量而已。

【颠倒比较器函数的正负号，就可改升序为降序】

function compare(a,b){

//return a-b;

return -(a-b);

}

2、**栈和队列**

js中没有专门的栈和队列类型，都是用普通数组模拟的

1、栈：

一端 封闭，只能从另一端进入的数组

FILO

何时使用：只要希望始终使用最新加入数组的元素时，就用栈结构。

如何使用：

1、结尾出入栈：

1、入栈：arr.push(值)

2、出栈：var last=arr.pop();

优点：每次出入栈，其余元素的位置不受影响。

2、开头出入栈：

1、入栈：arr.unshift(值);

2、出栈：var first=arr.shift();

缺点：每次出入栈，其余元素的位置都会受影响。

2、队列：

只能从一端进入，从另一端出

FIFO

1、结尾入队列：arr.push(值);

2、开头入队列：var.first=arr.shift();

何时使用：必须按照先来后到的顺序使用元素时。

3、二维数组

什么是二维数组：

数组中的元素，又引用了另一个子数组。

何时使用：

1、存储横行、竖列的二维数组时

2、需要对数组中的元素进行进一步细分类时

如何创建：

1、在创建时，初始化数组内容：

var arr=[[值1,值2,...],

[值1,值2,...],

...

]

2、先创建空数组，再添加子数组

var arr=[值1,值2,...];

3、访问二维数组中的元素：

arr[r][c];......

强调：二维数组，行下标不能越界，会报错

......

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

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

arr[r][c]; //当前元素

}

}

2、\*\*\***String**：一串字符组成的只读数组

vs 数组：

相同点：

1、也可以用下标访问每个字符

2、也有length属性，表示字符的个数

3、用for遍历每个字符

4、凡是'不会直接修改原数组的方法'，字符串也可使用，

如：slice、concat

字符串也叫内置对象

【"内置对象"】：ES标准中规定的，浏览器厂商已经实现的对象。

共 11 个：

String Number Boolean

Array Date Math RegExp

Error

Function Object

Global(浏览器中被window代替了)

\*\*\*"**包装类型**"：专门封装原始类型的值，并提供对原始类型的值进行操作的API

何时使用：只要试图对原始类型的值调用方法时，都会自动创建包装类型的对象，

来封装原始类型的值。调用的方法其实是包装类型对象的。而不是原始类型值的。

包装类型的对象，一旦方法执行完，就释放

eg:

var n=123.456;

console.log(n.toFixed(2));

实际上浏览器操作的是：

var n=123.456;

console.log(new Number(n).toFixed(2));

**String API** : 所有String API都无法修改原字符串，而是返回新字符串。

1、大小写转换：

何时使用：在不区分大小写时

如何转换：

**str=str.toUpperCase();** //转大写

**str=str.toLowerCase();** //转小写

2、获取指定位置的字符：

获取指定位置的字符：**str.charAt(i)** <=> str[i]

获取指定位置字符的Unicode号：**str.charCodeAt(i);**

## 7月5日

## 08 JSCore-Day05

1、\*\*\***String字符串**

截取子字符串：3种方式：

1、**str.slice(starti,endi+1)**

用法同数组的slice：

1、含头不含尾

2、"支持负数参数"

3、省略第二个参数的话，表示截到结尾

2、**str.substring(starti,endi+1)**

特点：

相同：

1、含头不含尾

2、省略第二个参数的话，表示截到结尾

不同："不支持负数参数"

如果使用倒数第n位，用 length-n

3、**str.substr(starti,n)**

从starti开始，截取n个。不用考虑含头不含尾

\*\*\*"**查找关键词**"：4种方法：

1、查找一个固定的关键词 的 【位置】

1、var i=str.indexOf("关键词",fromi);

在str中从from位置开始，找到下一个“关键词”所在位置。

返回值：

如果找到，返回关键词第一个字符的下标位置

如果没找到，返回 -1

特殊：如果省略 fromi 表示 从0位置开始。

强调：indexOf每次只找fromi 之后的 下一个 关键词

解决：用循环反复调用indexOf

每次都将fromi改为当前位置之后，直到返回-1结束

2、var i=str.lastindexOf("关键词",fromi);

在str中从fromi位置开始，找前一个“关键词”所在的位置。

返回值：同indexOf

特殊：省略fromi表示从length-1开始。

何时使用：只要查找一个固定的关键词的位置

问题：无法使用 模式匹配 的方式查找关键词。

即不支持正则

2、判断是否包含 和正则表达式匹配的 关键词：（支持正则）

1、var i=str.search(/正则表达式/i);

表示：在str中查找第一个和正则匹配的关键词的位置

说明：第二个/后+i，表示不区分大小写

不加，默认区分大小写

强调："search永远只能返回第一个关键词的位置"

"不能返回关键词的内容，也无法获得之后更多的关键词"。

返回值：如果找到，返回关键词的位置

如果没找到，返回 -1

存在问题：

永远只能找第一个且无法获得关键词的准确内容。

3、获取所有关键词的 【内容】：

var kwords=str.match(/正则/ig);

表示 获取str中所有 和正则匹配的关键词，将其 保存在结果数组 中。

说明：g表示全部。

如果不加g，默认只查找第一个。

如果没找到，返回 null ！

今后，都要先判断是不是null，再操作

eg:

console.log(kwords!=null ? "共找到"+kwords.length+"个关键词"+String(kwords) : "没找到");

如果不加g：会将完整关键词放在数组的第0个元素。同时为数组添加index属性，标记关键词的位置。

如果正则中包含分组，还会将每个分组匹配的子内容，分别放在数组的后续元素中。

存在的问题："只能获得内容，无法获得每个关键词的位置"。

解决：参见第4种方法

4、可获得每个关键词的 【内容和位置】

reg.exec()

......[下一天待续]

\*\*\*"**替换**"：

var str=str.replace(/正则/ig,"替换值")

表示：将str中所有和正则匹配的关键词都替换为“替换值”，再将生成的新字符串，存回str。

高级替换： //把函数作为参数传入

str=str.replace(/正则/ig,function(kword){

return '替换值';

});

eg：

str=str.replace(/我[去草]{1,2}/g,function(kword){

return kword.length==2 ? "\*\*" : "\*\*\*";

}

);

依次找到str中每个和正则匹配的关键词，传给函数的kword参数。

函数内根据本次找到的关键词，计算出不同的替换值。

replace会将函数返回的替换值，替换到字符串。

如果需要 部分替换关键词：

1、将正则表达式分组：

2、从函数的第二个参数开始，依次获得每次匹配的关键词的分组内容。

eg：

//该段代码实现：匹配 我去、我去去 或者 我草、我草草，然后把 去、去去 或 草、草草 字替换成 \*、\*\* 号。

str=str.replace(/我[去草]{1,2}/g,function(kword,$1,$2){

return $1+($2.length==1 ? "\*" : "\*\*");

}

);

衍生功能：

**删除**：其实就是将关键词替换成字符串""。

**格式化**：需要2步：

1、用正则将 要格式化的字符串分组

2、在replace的第二个参数中，用$n代替第n个分组的子内容，拼接格式。

\*\*\*"**分隔**"：将str按照指定的分隔符，分割为多个子字符串。

var subs=str.split("分隔符");

str.split(/正则/); //如 当天练习 05-分离邮箱账号中 用户名 与 域名

固定套路：

1、将字符串打散为"字符数组"：

var chars=str.split("");

2、按指定字符分隔：

var subs=str.split("分隔符");

3、将页面元素的内容，转化为数组

2、\*\*\*\*\***正则表达式**

什么是正则表达式：

描述一个字符串中，字符出现规律的规则

为什么：可以使用模式进行模糊匹配

何时使用：查找，替换，切割，验证

如何使用：

1、最简单的正则 就是关键词原文

2、字符集：

规定一位字符可用的 备选字符 列表

何时使用：只要某一位字符，有多个备选时

如何使用：[备选字符的列表]

强调：1个字符集，默认只能匹配一位字符

简写：如果字符集中的字符，部分unicode连续，可用 - 省略中间的简写

比如：1位数字：[0-9]

1位小写字母：[a-z]

1位字母：[A-Za-z]

1位汉字：[\u4e00-\u9fa5]

如果表示 除了xxx之外，用 [^xxx]

3、预定义字符集：

对常用字符集的简化：

1位数字：\d => [0-9]

1位字母数字或下划线：\w => [0-9A-Za-z]

1位空字符：\s => 空格，换行，制表符

1位任意字符：. => 除回车、换行外的一切字符

何时使用：

如果需求和预定义字符集完全匹配，才能使用

如果需求和预定义字符集不匹配，就必须使用普通字符集

4、量词：

规定一个字符集出现次数的规则

2类：

有明确数量限制的：

字符集{min,max} 字符集最少出现min次，最多不能超过max次

字符集{n,} 匹配字符n的出现次数 >=n 次

字符集{n} 匹配字符集必须 n 个

没有明确数量限制：

？ 0次或1次 即可有可无，最多1次

\* >=0次 即0次或多次字符

+ >=1次 即1次或多次 至少1个 ，多了不限

5、选择和分组

选择：| 表示：或 正则1|正则2 表示只要和任意一个正则匹配，就符合规则。

分组：()划分子表达式

"在线分析正则表达式的网站"：

"http://regexper.com/"

6、匹配指定位置

^ 字符串开始位置

读作 ”以xxx开头“ 或 开头的”xxx“

强调：放在正则的开头，表示字符串的开始位置

放在字符集[]内部的开头，表示除了xxx

$ 字符串结束位置

xxx$ 读作： 以xxx结尾

强调：如果同时前加^，后加$，表示必须从头到尾完整匹配。 --验证功能

如果不加，只要部分匹配就会验证通过

\b 匹配单词边界 开头，结尾，空格，字符

\b单词\b

【"**密码强度**"】：预判 排除法

eg:

至少包含一位大写字母以及一位数字

1、预判："不都是 小写字母和数字 组成"

说明 至少包含一位 大写字母 或 其他字符

(?![a-z0-9]+$)

2、预判："不都是 字母 组成"

说明至少包含一位数字或其他字符

(?![A-Za-z]+$)

3、正则："只能由 字母和数字 组成"

排除了之前 其他字符的可能

[A-Za-z0-9]{8,10}

放在一起：在前后加上 ^ 、$

**^(?![a-z0-9]+$)(?![A-Za-z]+$)[A-Za-z0-9]{8,10}$**

## 7月6日

## 08 JSCore-Day06

1、\*\*\***RegExp**

什么是正则表达式对象：封装一条正则表达式，

提供使用该正则表达式执行验证和查找的API。

何时使用：

1、既查找每个关键词内容，又查找位置的时候。

2、格式验证

如何创建：2种方法

1、使用直接量：

**var reg=/正则/ig;** //此处变量reg成为了正则表达式对象

何时使用：如果正则表达式是固定的，不需要运行时动态生成。

字符冲突：直接量中，正则内部如果包含 / ，需要转义为 \/

2、使用new关键字创建：

**var reg=new RegExp("正则","ig");**

何时使用：如果正则表达式需要在 运行时动态生成。

强调：正则前后不需要加 / /

字符冲突：如果正则中包含 " ' \ 等字符，都需要转义。

正则API：

查找关键词：4种：

1、查找一个固定的关键词 的 【位置】

1、var i=str.indexOf("关键词",fromi);

2、var i=str.lastindexOf("关键词",fromi);

问题：不支持正则

2、判断有没有包含和正则匹配的关键词：

var i=str.search(reg);

问题：永远从 0 位置开始，始终只能找第一个

3、获得所有关键词内容：

var kwords=str.match(reg); //加g

问题：无法获得每个关键词的位置

4、既找每个关键词的位置，又找内容。

var arr=reg.exec(str);

在str中找下 一个 关键词的位置和内容，保存在数组arr中。

【补】："reg对象的lastIndex属性"：

表示下次从lastIndex位置开始匹配

默认为0

原理：

1、用reg去str中匹配一个关键词

2、将本次找到的关键词"内容"，放在arr的第一个元素中，

同时将关键词的"位置"记录在arr.index属性中。

3、自动将reg的"lastIndex属性"调整为当前位置之后

返回值：

如果找到：同 match方法不加 g 时 情况完全一样，也就是只查找第一个。

arr[0] -> 本次找到的完整关键词

如果正则中有分组，那么：

arr[1]...-> 获得第n个分组的子内容

如何使用：用循环，反复调用exec，直到返回null为止。

【课堂练习01中的问题】：

【课堂练习01中的问题】：

正则：

**贪婪模式**：默认匹配尽量长的符合规则的字符串。

**懒惰模式**：让正则仅匹配最短的符合规则的字符串。

如何将贪婪改为懒惰：

.\*? .+?

**验证格式**：

var bool=reg.test(str);

验证str是否满足reg的格式要求

满足则返true，否则返回false

强调：一般验证都是要从头到尾完整匹配

都要前加 ^，后加 $。

2、**Math**

专门封装数学计算所用的API或常量

何时使用：只要执行数学计算时

创建：Math不能new！

API：所有API都用 Math 直接调用

1、取整

1、上取整：只要小数点超过，就取下一个整数

**num = Math.ceil(num);**

2、下取整：舍弃小数部分

**num = Math.floor(num);**

只能对纯数字 下取整

对比：parseInt(str);

转换为number，去掉结尾的非数字字符。舍弃小数部分

3、四舍五入取整：

**num = Math.round(num);**

强调：只能取整，不能设定小数位数

对比：**toFixed(d)**

参数上来说：round 无法指定小数位数，只能取整

toFixed(d) 可指定小数位数，可按任意小数位数四舍五入

返回值："round 返回num，可直接参与算数计算"

"toFixed 返回str，都要提前类型转换，再做加法"

强调：小心舍入误差。-- 例 03

2、乘方和开平方：

**Math.pow(底数,幂)**

**Math.sqrt(num) 开平方**

3、最大值和最小值

**Math.max(值1,值2,...)**

**Math.min(值1,值2,...)**

强调：max和min不支持数组类型的参数

无法获得数组中的最大值

解决：

Math.max.apply(null,arr)

4、【**随机数**】

Math.random(): 产生0-1之间的随机数

公式：

如果从任意min-max之间取随机整数：

**parseInt(Math.random()\*(max-min+1)+min)**

如果从0-max之间：

parseInt(Math.random()\*(max+1))

3、**Date**:

什么是日期对象：封装一个时间

提供操作时间的API

何时使用：只要保存时间和日期，以及对时间进行操作时。

创建：3种场景

1、创建日期对象，保存当前时间：

**var now=new Date();**

强调：now封装的是客户端本地时间。

2、创建日期对象，保存自定义时间。 -- 2种

var date=new Date('yyyy/MM/dd hh:mm:ss'); //√√√√√

var date=new Date('yyyy,MM-1,dd,hh,mm,ss');

说明：

1、如果不关心时分秒，可省略

2、如果使用数字设置时间，月份要-1

3、**复制日期对象**：

为什么？

因为日期的计算，都是直接修改原日期对象。

何时复制："如果希望同时保留开始时间和截止时间时，都要先复制，再计算"。

如何复制：

var date1=new Date();

var date2=new Date(date1.getTime());

说明：.getTime()可省略

原理：日期对象中保存的其实是一个巨大的毫秒数。

是1970年1月1日0点至今的毫秒数。

**API：**

分量：FullYear Month Date Day

Hours Minutes Seconds Milliseconds

1、每个分量都有一对getXXX/setXXX方法。此处XXX即该分量名

其中，date.getXXX方法负责获取指定分量的值

date.setXXX(n)方法负责修改指定分量的值

"例外：星期Day没有setDay方法"

2、命名：年月日星期，没有s结尾

时分秒毫秒，都有s结尾

3、取值范围（返回值）：

只有月中的日 Date 从 1~31 结束

其余都是 从 0 ~ 进制-1 结束

"Month：0~11 需要修正"：

计算机中的月份比现实中的少1

Date：1~31 不用修正

Day：0~6 不用修正：

星期日是第一天，即0

Hours：0~23 不用修正

Minutes/Seconds：0~59 不用修正

应用：

日期计算：

1、两个日期对象可相减：得到毫秒差

2、对任意分量做加减：3步：

1、取分量 var n=date.getXXX();

2、做加减 n+=x; //如：n+=59

3、放回去 date.setXXX(n);

说明：

1、setXXX方法可自动调整时间

2、setXXX方法直接修改原日期

简版公式：date.setXXX(date.getXXX()+n)

日期转字符串：

date.toGMTString(); 转为中国标准时间格式

date.toLocaleString(); 转为当地时间格式

eg:

var now=new Date();

now.toGMTString();

//输出 ："Wed, 06 Jul 2016 09:51:18 GMT"

date.toLocaleDateString(); 仅保留日期

date.toLocaleTimeString(); 仅保留时间

## 7月7日

## 08 JSCore-Day07

1、\*\*\***错误处理**

什么是错误（bug）：在程序过程中导致程序异常中断的问题。

Error：在错误发生时，保存错误信息的"对象"。

JS中的6种错误：

SyntaxError：语法错误

ReferenceError：引用错误，没找到xxx

TypeError：类型错误，错误的使用了对象的方法。

RangeError：范围错误，特指参数超范围

EvalError：错误的使用了eval

URIError：URI错误

任何程序发生错误时，都会强行退出

错误处理：在程序发生错误时，保证程序不会强行退出的机制。

如何处理：

try{

可能出现错误的代码

}catch(err){

只有发生错误时，才执行的错误处理代码

}finally{

无论是否出错，都必须执行的代码

//主动保存数据

//释放资源

}

eg：例题01、02

err对象是在错误发生时JS自动创建的

"效率：一段正常代码，只要放入try中，执行效率一定降低"。

解决：

1、try中应仅放可能出现问题的代码。

2、如果可以提前预知的异常处理，都可通过 if...else 代替 try...catch...

可用于解决浏览器兼容性问题：

判断浏览器是否支持Ajax:

//用try catch判断浏览器兼容性：

try{

new XMLHttpRequest();

document.write("完美支持Ajax");

}catch(err){

document.write("您的浏览器不支持Ajax");

}

抛出自定义错误：

何时：函数的定义者向函数的调用者提醒 错误地使用了函数。

如何：

函数定义者：throw new Error("错误提示");

函数调用者：用 if...else... 或 try...catch 处理异常

eg：

例-01

2、\*\*\*\*\***Function**

什么是函数对象：在内存中封装一个 函数定义的对象。

何时使用：代码重用

创建：3种方式

1、声明：function 函数名(参数列表){

函数体;

return 返回值;

}

强调：被整体声明提前

2、函数直接量：var 函数名=function(参数列表){...};

强调："函数直接量（等号右边的部分）不会被声明提前"

何时使用：只要不希望被声明提前时

揭示：函数其实是引用类型的对象

函数名其实仅是一个引用函数对象的普通变量

3、用 new 关键字：

var 函数名=new Function("参数1","参数2", ... ,

"函数体 ; return 返回值");

eg：例-04

1、**重载**

什么是重载：

相同函数名、不同参数列表的多个函数，在调用时可【自动】根据传入的参数不同，调用不同的函数执行。

为什么使用：

相同的任务就应该起相同的函数名来减轻调用者的负担。

何时使用：

【"同一件事，根据传入参数的不同，执行不同的逻辑时"】。

如何实现：

JS的语法不支持重载！

因为JS中不允许包含多个同名函数。最后定义的同名函数会覆盖之前定义的。

解决：arguments

arguments：是每个函数中"【自动创建的】"、用于保存所有传入函数的参数值的【"类数组对象"】。

==> 【类数组对象】：长得像数组的对象，"但不是数组" vs 数组：

相同：1、可用下标访问每个元素 ==> 例05

2、都有length属性

3、使用for遍历

不同：类型不同

数组：Array

类数组对象：Object

无法使用数组类型的API

arguments.callee ：（自己） 引用了当前函数对象。专门实现递归调用。

eg:例-06

arguments.callee.caller ：（调用自己的别人） 表示调用当前函数的外层函数

2、**匿名函数**

什么是匿名函数：

定义函数时，不使用任何变量引用的函数。

function (){

return a-b;

}

为什么使用：节约内存

因为没有任何变量使用，使用完立刻释放。

何时使用："如果一个函数只用一次，就必须用匿名函数"。

如何使用：

1、匿名函数自调：创建完函数，立刻执行。

(function(...){...})()

执行后，匿名函数被释放。

作用：1、节约内存

2、充当临时作用域

2、匿名函数回调：将函数传给另一个函数使用

比如：【**比较器函数**】：

var cmp=function(a,b){return a-b}

arr.sort(cmp);

==>

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

//cmp 没用了

//"今后一律用这种方式写比较器函数"

3、\*\*\*\***作用域和作用域链**

函数的生命周期：4个阶段：

1、程序开始执行时：(步骤：)

1、创建一个 Execution Context Stack(ECS)，

是 依次保存正在调用的函数的 "执行环境"的栈结构；

2、创建全局作用域对象：window；

3、在ECS中压入第一个全局执行环境。

2、定义函数时：

1、创建函数对象，封装函数的定义；

2、在函数对象中，设置scope属性，引用函数来自的作用域，通常scope都是window；

3、用函数名创建全局变量，引用函数对象。

3、调用函数时：

1、创建一个活动对象 Actived Object(AO):

活动对象：保存函数的局部变量的函数作用域对象；

2、向ECS中压入本次函数调用的执行环境EC；

3、EC引用AO；

4、设置AO引用函数的scope(window)。

4、函数调用后：

1、EC出栈，导致AO无人使用，而被释放。

导致AO中的局部变量一同被释放！

作用域：一个变量的可用范围

其实window对象就是全局作用域

"AO对象就是函数作用域"

AO对象又引用了window对象

作用域链：由各级作用域对象，逐级引用形成的链式结构，就是作用域链。

作用域链的末尾是window对象。

作用域链控制着变量的使用顺序：

优先使用AO中的局部变量，

如果AO中没有，就沿着作用域链向下找，

如果到window还没找到，就报错。

【【**笔试题**】】：

......

4、\*\*\*\*\***闭包**

什么是闭包："既重用变量，又保护变量不受污染的机制"。

为什么使用：

全局变量的优点：随处可用、可反复使用

缺点：容易被全局污染

局部变量的优点：仅在函数内使用，不会被污染

缺点：无法反复使用

何时使用：如果希望重用一个变量，且不希望该变量被随意篡改时。

如何实现：3步：

1、用外层函数将 【受保护的变量和操作变量的函数】 封装在内部。

2、外层函数将内层函数返回。

3、调用外层函数，获得返回的内层函数对象

说明：

1、两次外层函数调用返回的闭包中，受保护的变量是各自独立的，没有任何关系。

**闭包为什么能留住局部变量？：**

**外层函数的作用域对象无法释放（即没有被释放）**

## 7月8日

## 08 JSCore-Day08

回顾：

1、闭包：

什么是：重用并保护一个局部变量的机制

'因为外层函数的作用域无法被释放。'

为什么：

全局变量、局部变量

何时：希望重用一个变量又不希望被篡改时

如何使用：3步

1、用外层函数包裹受保护的变量和操作变量的函数

2、外层函数将内层函数返回

3、使用者调用外层函数调用内层函数的返回对象。

笔试题：（步骤）

1、找受保护的变量是谁。

2、找操作变量的函数。

3、同一次外层函数调用返回的所有内层函数对象，共用同一个受保护的局部变量

4、两次外层函数调用，返回的不同内层函数的对象，所操作的变量没有任何关系。

闭包返回内层函数对象：

1、return

2、直接给全局变量赋值

3、将内层函数封装在一个对象中，然后返回对象。

闭包的缺点：

占用更多内存空间，且一旦形成闭包，无法自动释放。

1、\*\*\*\*\*面向对象（OOP）

1、对象：

1、专门描述现实中一个具体事物的属性和功能的一种程序结构

事物的属性会成为对象的属性，事物的功能会成为对象的方法。

2、其实质：内存中同时存储多个数据和方法的一块存储空间

2、面向对象：

在程序中，都是先用对象来封装一个事物的属性和功能。然后再调用对象的方法来执行任务。

为什么面向对象如此流行？：符合人的日常习惯。

3、面向对象三大特点：封装、继承、多态

1、封装: 将现实中一个事物的属性和功能集中定义在一个对象中。

如何使用：2步

1、创建对象 -- 封装：2个场景

1、创建一个单独的对象：

1、对象直接量：【'习惯于使用第一种'】

var obj={

'属性名':属性值,

... :...;

'方法名':function(){

... this.属性 ...

}

}

【何时使用】：

"如果创建对象时，已经知道对象的所有属性和方法时"。

对象的属性和方法统称为对象的成员

对象中的每个成员名，都是字符串类型

但是，可省略""。

问题：对象的方法中，不能写死对象的属性

解决：在对象的方法中，直接访问对象当前对象自己的属性

this 关键词：专门用在对象的方法中，用来指代正在调用方法的当前对象本身。

其实就是调用方法时，点"."前的对象

总结：只要在对象 的方法中访问对象自己的属性，必须用this.属性名

2、使用new关键词：**（工厂模式）**

var obj=new Object(); //先创建空对象

var obj={};

obj.属性名=值; //再添加属性

obj.方法名=function(){

...this.属性...

}

补：new和()都可省略，但不可同时省略

new Object()

Object()

new Object

Object

【何时使用】：

"在创建对象时，还不知道对象的属性和方法，需要后续动态添加"

\*\*\*\*\*【说明：JS中一切对象都是关联数组】

对象与数组的相同：

1、属性名都是字符串，不可重复

2、随时添加新属性和方法

3、可用for in 遍历

2、批量创建多个相同结构的对象：2步：

1、定义**构造函数**：**（构造函数不需要返回值）**

构造函数：专门定义类型对象统一结构的特殊函数。

为什么使用：

为代码重用

何时使用：今后只要反复创建多个相同结构的对象前，都要用构造函数先定义统一的结构。

如何定义：

function 类型名/构造函数名(属性参数,...){

//this：当前正在创建的空对象

//向当前空对象中添加新的属性

this.属性名=属性参数;

//向当前空对象中添加新的方法

this.方法名=function(){

...

}

}

2、用new关键字调用构造函数创建新对象：

var obj=new 构造函数名(属性值);

这里new做了4件事：

1、创建一个空对象

2、"设置新对象的 \_proto\_ 继承构造函数的原型对象"

3、用新对象调用构造函数

构造函数会向新对象中添加属性和方法。

4、将新对象地址返回给obj变量保存。

2、访问对象的成员：

访问属性：对象.属性

单个对象属性的用法和变量完全一样。

调用方法：对象.方法(参数)

找到“对象”的“方法”，执行。

单个对象方法的使用和函数完全一样。

2、继承：父对象的成员，子对象不必重新创建就可以直接使用。

JS中的继承都是通过原型对象实现的。

原型和原型链：

原型：保存一类对象共有成员的父级对象

为什么使用原型对象：为了实现继承

"优点：代码重用、节约内存。"

何时使用原型对象：

只要一类对象共有的成员，都必须集中定义在原型对象中一次即可。

如何使用原型对象：

创建：在定义构造函数时，JS会自动创建该类型的原型对象

向原型对象中添加其他成员：

构造函数.prototype.成员名=值

内置对象的原型对象：

可解决浏览器的兼容性问题：

如果需要的API，在指定类型的原型对象中不存在，说明不支持

就要在该类型的原型对象中添加所需的共有成员。

其中在API内部，用this获得当前正在调用API的点前的对象

原型链：由各级父对象，各级继承形成的链式结构

控制着对象成员（属性和方法）的使用顺序：

优先使用对象本地的成员 ---- 自有属性

如果本地没有，才沿着原型链向上查找各级父对象。直到找到为止。 ---- 共有属性

如果整个原型链上没有，才返回undefined

怎样判断成员是 自有属性 还是 共有属性？

1、判断自有属性：

obj.hasOwnProperty("属性名")

用于判断“属性名”是否是obj的自有属性

保存在obj对象本地

是自有属性就返回true，否则返回false

2、判断共有属性

问题：如果不是自有属性：

1、可能在原型链上

2、也可能根本没有

解决：不是自有，且可以访问到

!obj.hasOwnProperty('属性名') && obj.属性名!=undefined

"如何修改自有属性和共有属性"：

自有属性，只能用所在对象去【修改】

共有属性，必须通过原型对象去【修改】

如果强行使用某个子对象，修改共有属性，

后果：仅在当前子对象添加同名自有属性

导致：当前子对象无法再使用共有属性。

删除属性：

delete 对象.属性

【亲子鉴定】：给对象做亲子鉴定

1、用原型对象检查：

var bool=father.isPrototypeOf(child)

判断child是否继承自father

father是否在child的原型链上

如果father在child的原型链上，返回true

否则 返回false

eg：

//验证：

//采用匿名函数自调的方法

(function(){

var arr=[];

//验证arguments是数组类型的对象吗？

console.log( Array.prototype.isPrototypeOf(arguments)

); //false

//验证arr是数组类型的对象吗？

console.log(

Array.prototype.isPrototypeOf(arr)

); //true

})();

2、用构造函数检查

var bool=child instanceOf 构造函数

判断child是否是构造函数创建出来的子对象

补：instance实例：（实例化）

用一个构造函数创建出来的一个子对象，就称为子对象时构造函数的实例

eg:

var obj=new 构造函数();

实例化一个 构造函数 类型的对象

eg：实例化一个数组类型的对象arr：

var arr=new Array();

【笔试题】：

判断一个对象是否是数组类型，有几种办法：

3、多态：同一个函数，在不同情况下，表现出不同的状态

重写(override)：如果子对象觉得父对象的成员不好用，

就可在子对象本地重新定义同名成员，覆盖父对象的成员。

call：强行借用一个本来无法调用到的函数

目标函数.call(obj)

执行时：相当于obj.目标函数

## 7月11日

## 08 JSCore-Day09

回顾：【**笔试题**】：**（谈谈你对面向对象的理解）**

面向对象：

三大特点：封装、继承、多态

**封装**：创建一个对象，集中保存现实中一个事物的属性和功能

3种：

1、对象直接量：

何时使用：只创建一个单独的对象

已经知道所有的属性和功能

2、用new关键字：

何时使用：只创建一个单独的变量

在创建时，暂时不知道对象的属性和功能

3、构造函数：

何时使用：反复创建多个相同类型的对象时

如何使用：2步

1、调用构造函数（类型），描述一个对象统一的数据结构

2、用new实例化构造函数的对象

new：

1、创建新对象

2、设置新对象继承构造函数的原型对象

3、用新对象调用构造函数，添加属性和方法

4、返回新对象地址

优点：代码重用

缺点：不能节约对象

**继承**：父对象的成员，子对象无需重复创建就可直接使用

为什么（优点）：代码重用、节约内存

何时使用：如果多个子对象拥有相同的成员（属性值和方法）时，就都要定义父对象，统一保存一份。

如何使用：JS中的继承都是继承原型

原型对象：集中保存同一类型的多个子对象共有的属性值和方法的对象

创建：定义构造函数时，自动创建。

每个构造函数都有一个prototype属性指向该类型的原型对象。

默认情况下：每实例化一个子对象，就自动设置子对象的\_\_proto\_\_继承构造函数的原型对象

原型链：由各级父对象逐级继承，形成的链式结构。

控制着对象的属性的访问顺序

VS 作用域链：

后者控制变量的使用顺序

**多态**：同一个方法在不同情况下，表现出不同的状态。

重写（override）:如果子对象觉得父对象的成员不好用，可在本地创建同名自有成员，覆盖父对象中的成员。

为什么存在多态：表现子对象和父对象之间的差异

正课：

1、\*\*\*\*\***自定义继承关系**：

1、仅修改一个对象的父对象：

obj.\_\_proto\_\_ = father

问题：\_\_proto\_\_是内部 属性，本不能直接使用。

解决：

Object.setPrototypeOf(child,father);

设置child对象继承father

Object.getPrototypeOf(child)

获得child的父对象

eg：例-02

问题：每次只能修改一个对象的父对象

2、批量修改所有对象的父对象

通过修改构造函数的原型对象：

构造函数.prototype = father

eg：例-01

时机：在定义完构造函数后，在"开始创建第一个子对象之前修改构造函数的原型对象"，才能批量修改所有子元素的父对象。

3、【**两种类型间的继承**】：

问题：当两种类型间有相同的 属性定义 和 方法 时

解决：抽象/创建 出一个公共的父类型，集中保存两种类型相同的属性结构和方法

如何实现:3步

1、定义父类型 构造函数和原型对象

2、在子类型构造函数中"借用父类型构造函数"

如何借用：call 和 apply

父类型构造函数.apply(this,arguments)

call和apply：专门用于借用一个函数，并替换函数 中的this为指定对象

何时使用：今后只要函数中的this不是想要的，都要用call和apply替换。

"【**笔试题**】**call VS apply "的相同点与不同点**

相同：借用构造函数，替换this

不同：call要求传入函数的参数必须独立传入。

apply要求传入函数的参数必须放入一个数组或集合中，整体传入

如何使用：

函数.call(替换this的对象,要传给函数的参数1,参数2)

eg：例-02 飞机大战

Flyer.call(this,name,speed);

3、让子类型的原型对象继承父类型的原型对象。

Object.setPrototypeOf(){

子类型的原型对象,

父类型的原型对象

}

2、\*\*\*\*\***ECMAScript5：(ES5)**

对象的属性：

ES5中提供了对对象属性的保护

划分；两类：

1、命名属性：凡是可用 . 直接访问的属性

1、**数据属性**：直接存储属性值的属性

ES5规定每个数据属性都有四大特性

value：实际存储属性值

writable：true/false 是否可修改

enumerable：true/false 是否可遍历

configurable：true/false 是否可修改或删除其他特性

如何【读取】一个属性的特性：

**Object.getOwnPropertyDescriptor(obj,'属性名');**

//返回一个对象，包含四大特征

特性的默认值：

用传统方式添加的属性，四大特性都是true

如何【设置】属性的特性：

**Object.defineProperty(obj,"属性名"{**

**特性:值,**

**... :...**

**})**

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

如果obj中没有指定的属性名，则自动添加同名属性。"但是特性默认都是false！"

建议：用 defineProperty 添加同名属性时，尽量 显式定义四大特性 。

问题：defineProperty依次只能设置一个属性的四大特性

解决：

Object.defineProperties(obj,{

属性:{四大特性},

... : ...

})

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

问题：任何时候都可用definedProperty将特性恢复原样

解决："修改特性同时，都要将configurable设置为false"。

configurable一旦被改为false，则不可逆（不可恢复）

2、**访问器属性**：不直接存储属性值，仅提供对其他属性的保护。

不直接保存数据，"仅为其他数据属性提供验证和保护 的特殊属性"。

四大特性：

get:function(){return xxx},

set:function(val){

验证val

xx=val;

},

enumerable:...,

configurable:...

何时使用：只要用自定义的规则保护属性时，都要用访问器属性。

如何使用：2步：

1、定义数据属性，实际存储属性值

问题：使用者可直接操作数据属性，而绕过访问器属性的保护。

"解决：闭包！"

2、定义访问器属性，保护数据属性

只能用defineProperty()

Object.defineProperty(obj,"属性名",{

get:function(){return xxx},

set:function(val){xxx=val},

enumerable:...,

configurable:...

})

get和set方法都 不用手动调用：

只要用访问器属性取值时，自动调用get()

只要用访问器属性赋值时，自动调用set()

"set中的参数val，会自动获得要赋的值"

eg：例04

构造函数中的访问器属性：

构造函数中的访问器属性：

"【**笔试题**】"：

**实现一个类型，同时包含public公有属性和private私有属性**

private私有属性：仅在对象内部使用的属性，无法通过 . 访问到

public 公有属性：直接用 对象. 可访问到的

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

eg：例-05

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

【【防篡改】：禁止添加、删除、修改对象的属性】

1、防扩展：

每个对象中，都有一个默认的内部属性：extensible，默认为true。

说明随时可扩展新属性。

如果修改extensible属性为false，则禁止扩展新属性。

如何修改：

**Object.preventExtensions(obj);**

阻止对obj对象的一切扩展

问题：

只限制添加新属性，不限制删除旧属性。

2、密封：

是在"防扩展的基础上"修改所有属性的configurable特性为false。

如何密封：

**Object.seal();**

3、冻结：

就是"在密封的基础上"禁止修改所有属性的值。

**Object.freeze(obj);**

**Object.create():**"创建一个新对象，继承指定的父对象，同时为新对象扩展新属性"。

何时使用：

只要基于一个现有父对象，创建一个子对象时。

如何使用：

var child=Object.create(father,{

新属性:{四大特性},

... :...

})

2、内部属性：不允许打.，直接访问的属性

比如：class \_\_proto\_\_

【**严格模式**】：比普通的JS代码执行要求更严格

为什么：为了统一开发的写法。

何时使用：新项目，都要在严格模式下开发。

旧项目，逐个功能向严格模式迁移。

如何启用：在当前代码段的顶部加：

"use strict";

规定：

1、将静默失败升级为错误！

**数组API：**

1、判断：数组中的元素是否符合要求：

**arr.every(...)**：判段arr中所有元素是否都符合要求

**arr.some(...)**：判段arr中是否包含符合要求的元素。

如何使用：

var bool=arr.every(function(){

function(val,idx,arr){

return 判段条件;

}

});

every会用functon去arr中每个元素执行检查。只有每个元素的执行结果都为true时，才返回true。只要任意一个不返回true，则every立刻返回false

说明：val：自动获得当前正在检查的元素值

idx：自动获得当前正在检查的元素位置

arr：自动获得当前正在检查的元素

2、遍历：对数组中的每个元素执行相同的操作

arr.forEach(...);

依次对arr中每个元素执行相同的操作，再保存回原数组。

直接修改原数组

arr.map(...);

依次取出arr中的每个元素的值，执行相同操作再保存到新数组中。

## 7月12日

## 08 JSCore-Day10 : am

1、\*\*\*\*\*ES5：

1、**数组API**（续）：

1、...

2、...

3、过滤和汇总：

**过滤**：筛选出arr中符合条件的元素组成新数组。

**var sub=arr.filter(function(val,idx,arr){**

**return 条件;**

**});**

这种做法不修改原数组。

**汇总**：将当前数组中的所有元素统计出一个结果。

**var r=arr.reduce(function(prev,val,idx,arr){**

**return 将val汇总到prev中的结果;**

**},base);**

base：用来设置起始的汇总值。

表示依次将arr中的每个值 ，汇总到prev中。

说明：prev获得的是截止idx位置之前所有元素值的临时汇总。

2、\*\*\*\*\***bind**:

bind：基于一个现有函数，创建一个新函数，同时永久绑定this对象和部分参数。

为什么：

**call和apply:**

临时借用函数，替换其中的this，传入参数。

问题：都是临时借用，临时替换this。

解决：用bind

何时使用：希望永久绑定this和部分参数时

如何使用：

**var fun=Fun.bind(obj,参数,参数2,...);**

意思是：基于已有的Fun函数，返回一个新函数，

同时永久绑定新函数中的this为obj，再同时绑定部分参数值为参数1,参数2,...

"【eg：例-02; bind】"

3、**严格模式**：

启用：代码段顶部：**"use strict";**

2处：

1、script的开始

2、函数的开始

规则：

1、**将静默失败升级为错误**

2、**禁止对未声明过的变量赋值**

(function(){

"use strict";

m=12; //报错：m is not defined

console.log(m);

})();

3、**禁用了arguments.callee**

(function(){

"use strict";

arguments.callee(); //报错： 'caller', 'callee', and 'arguments' properties

//may not be accessed on strict mode functions or

//the arguments objects for calls to them.

})();

4、**多了eval作用域**

(function(){

'use strict';

eval("var sum=2+3");

console.log(sum); //报错：sum is not defined

})();

## 7月14日

## 08 DOM-01

1、DOM：

1、什么是DOM：专门操作网页内容的API

JS：三大部分：

ECMAScript（核心语法 9day）+

DOM（专门操作网页内容的API 3day）+

BOM（专门操作浏览器窗口的API 2day）

DHTML： 【笔试题】 HTML XHTML DHTML XML

HTML：专门编写网页内容的语言

XHTML：一个更严格的XHTML标准

DHTML：一切可以实现网页动态效果的技术的统称

包含：HTML + CSS + JS

XML：Extensible Markup Language

专门存储和传输格式化数据的文档结构。HTML是XML的一个子集

JSON：JavaScript Object Notation

3种API：

1、"核心DOM"：万能的。能够操作一切结构化文档。

包括：HTML XML

问题：API繁琐

解决：用专门的HTML DOM

2、HTML DOM：是专门操作HTML网页内容的API。

对核心DOM中'部分常用的'API提供的简化版本。 -- 不是万能的。

实际开发中选择最简单的用，如果简单的无法实现，就用会用的。

3、XML DOM：专门操作XML文档的API。

2、\*\*\*DOM Tree：

1、什么是DOM树：网页的所有内容在内存中其实是以树形结构存储的。

1、何时创建：当浏览器读取到HTML内容时，会自动创建DOM树。

2、如何创建：

1、读到HTML文档时，先创建根节点：

document 可以查找元素、创建元素

2、开始顺序读取HTML文档的内容，创建子节点。

网页中的"一切内容都是DOM树上的节点对象。"

节点对象：Node类型是所有节点类型的父类型。Node类型提供了所有节点对象共有的基本属性和方法。

三大属性：

1、node.nodeType:节点的类型。Number类型的属性

何时使用：判断或区分节点的类型时

如何使用：值是数字

document：9

元素节点：1

属性节点：2

文本节点：3

DOCTYPE: 10

问题：nodeType无法细致区分每种元素

解决：

2、node.nodeName: 节点的名称：

何时使用：判断或区分节点名时

更多用于细致区分每种元素标签名

document：#document

元素节点：标签名(全大写)

属性节点：属性名

文本节点：#text

DOCTYPE: html

3、node.nodeValue: 节点属性值。字符串类型的属性

何时使用: 只用于获得属性的值和文本的内容

document：null/undefined

元素节点：null/undefined

属性节点：属性值

文本节点：文本内容

DOCTYPE: null

节点间关系：2大类

【节点树】：包含所有节点的树结构

任何文本也是节点，所以，节点树会受到看不见的空文本的干扰

1、父子关系：4个

1、elem.parentNode => 获得ele的父节点

2、elem.childNodes -> 是类数组对象，不是数组

=> 获得elem的所有"直接子对象"

3、elem.firstChild => 获得elem的第一个子节点

4、elem.lastChild => 获得elem的最后一个子节点

2、兄弟关系：2个：

1、elem.previousSibling =>

获得elem的前一个兄弟节点

2、elem.nextSibling => 获得elem的后一个兄弟节点

何时使用：站在现有节点，查找周边节点。建议查找节点要就近。

【元素树】：仅包含元素节点的树结构。不是一颗新的树，仅是节点树的子集。

优点：仅包含元素节点，不受空文本的干扰

缺点：无法访问文本节点

变通：elem.innerHTML

包含：2大类：

节点树 元素树

parentNode parentElement

childNodes children

firstChild firstElementChild

lastChild lastElementChild

previousSibling previousElementSibling

nextSibling nextElementSibling

3、递归遍历：遍历指定父节点下的所有子节点

2步：

1、先定义一个函数，仅遍历"直接子节点"

2、对每个直接子节点，再次调用当前遍历函数本身。

childNodes：返回所有直接子节点对象，保存在一个类数组对象中。

是一个【"动态集合"】：不实际存储对象的完整属性。

childNodes存在的问题：

"问题：每次访问动态集合时，都会反复去DOM数查找。"

解决：在遍历的时候，首先将length属性的值保存在变量中，用变量作为循环的条件。

eg: 例 - 3\_2

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

算法：深度优先遍历

每次都优先遍历子节点。"所有子节点"遍历完毕，再遍历兄弟节点。

2、遍历API：

1、NodeIterator:（节点遍历器）"按照深度优先的原则"，依次读取指定父节点下的每个节点

如何使用：2步

1、创建NodeIterator对象：

var iterator = document.createNodeIterator(parent,NodeFilter.SHOW\_ALL,

.SHOW\_ELEMENT

null,false

);

2、反复调用：

var currNode = iterator.nextNode();

//让iterator跳到下一个节点，并返回

到头返回null

强调：调用一次nextNode，只会向前跳一个节点

iterator.previousNode();

//退一个节点

2、TreeWalker：基本用法和NodeIterator完全一样。

强调：

1、"不遍历根节点，只遍历子节点"

2、又提供了向四周跳转的方法：

walker.parentNode();

.firstChild();

.lastChild();

.nextSibling();

.previousSibling();

3、查找：

DOM：可以查找、修改、创建、删除元素

1、按HTML属性查找：

1、按id查找一个元素：

var elem = document.getElementById('id');

强调："ById只能用在document对象上"。

2、按标签名查找多个子元素：

var elems = parent.getElementsByTagName('标签名');

强调：1、"ByTagName可用在任意父元素上"。

2、ByTagName不止查询直接子节点，且查询所有后代节点。

3、按class查找：

var elems=parent.getElementByClassName('class');

4、按name查找：

var elems=parent.getElementByName('name');

强调：返回值elems即使返回的是一个元素，也会放在集合中返回，

所以必须加[0]，才能取出唯一的元素

返回值elems：也是动态集合

2、用选择器查找： -- "Selector API"，是 jQuery 的核心

1、找1个：

var elem=parent.querySelector('select');

2、找多个：

var elems=parent.querySelectorAll('select');

强调：elems非动态集合，不会导致反复遍历DOM树。

何时使用：

如果需要多次getXXX才能找到目标元素时。

不适合使用：

基于现有节点向上或向周围爬树时。

【笔试题】：getXXX vs Selector API

效率：getXXX的效率 比 Selector API 高！

因为前者返回动态集合，不需要准备就可返回结果。 -- 快

后者准备好所有相关数据再返回。 -- 慢

返回值：前者 -> 动态集合，可能造成反复遍历DOM树。

后者 -> 非动态集合，不会造成反复遍历DOM树。

如何选择：

如果一次getXXX查找即可找到结果时，首选getXXX

如果需要多次getXXX才能找到目标元素时，首选selectorAPI

不需要查找，可直接获得的元素：

document.body -> <body>

document.head -> <head>

document.documentElement -> <html>

## 7月15日

## 08 DOM-02

1、\*\*\*修改：

1、修改内容：

1、获取或修改元素开始标签和结束标签之间的html内容：

elem.innerHTML

2、获取或修改元素开始标签和结束标签之间的纯文本内容（去掉标签，将特殊符号转为正文）：

elem.textContent

专门用途：

去掉内容中的标签，仅保留文字

IE8中使用innerText

2、修改属性：

1、"核心DOM"：get、set、remove。万能，但是繁琐

1、获取属性值：【【月考题！！】】

1、//获得属性节点对象

var attrNode=elem.attributes[i]; 或 下标i写成属性名

elem.attributes('属性名');

attrNode.value //获得属性节点的值

eg:

a1.attributes['href'].value //获取到a标签的href的值

2、//直接获得属性值

var value=elem.getAttribute('属性名');

2、设置属性值：

elem.setAttribute('属性名',属性值); //属性值会隐式转换成字符串

3、移除属性：

elem.removeAttribute('属性名');

4、判断是否包含指定属性：

elem.hasAttribute('属性名');

2、"HTML DOM"：

1、所有html标准的属性，自动成为元素对象的属性。可用 . 访问。

元素对象的属性和普通对象的属性用法完全相同。

强调：HTML DOM只能访问标准属性，

自定义属性只能用核心DOM访问

特殊：HTML class --> js className

建议：修改样式，首选className修改

"自定义属性：HTML5规定"

什么是：开始标签中，data-开头 的属性

data-i data-age data- 是HTML5中自定义属性的标志

如何获取自定义属性值：

elem.dataset.i

3、修改样式：

1、内联样式：

1、获取或设置：

elem.style.css属性名 = 值

强调：只能获得内联样式

优点：修改一个元素的样式时，又不影响其他元素。

何时使用：只要修改一个元素的样式时，首选 style.css属性名 的方式

强调：css属性名：

将css中的属性名去掉 - ，变 驼峰命名法

font-size -> fontSize

background-color -> backgroundColor

list-style-type -> listStyleType

2、内部/外部：

1、获取最终应用到当前元素的计算后的样式

var style=getComputedStyle(elem) //返回的是一个style对象

style.css属性名

何时使用：只要想获得一个元素完整的样式，就要用getComputedStyle(elem)

2、修改样式表中的样式：强烈不建议

样式表对象模型：COM

步骤：

1、每个样式表都是一个sheet对象：

var sheet = document.styleSheets[i];

2、每个选择器后的 {...}，就是一个cssRule规则。

var cssRule = sheet.cssRules[i];

3、访问每个cssRule中的一个样式属性：

cssRule.style.css属性