Js(上)

一、什么是JavaScript

1、前端三大语言：

HTML，专门编写网页内容的语言

CSS，专门美化网页样式的语言

JS，专门设计网页交互的语言

2、交互：

用户输入数据→程序接受并处理数据→返回处理结果

3、js发展历程：

ECMAScript(1996年)：ES，ECMA制定的js语言标准，规定了js语言的【核心语法】

JavaScript(1996年)：NetScape【按照ES标准】的规定，实现的可使用的编程语言

JScript(1996年)：Microsoft【按照ES标准】制定的自己的客户端脚本语言

W3C(1998年)：DOM标准，专门操作网页内容的标准；随后微软和网景都宣传支持DOM标准

4、JavaScript= ECMAScript标准（核心语法）

+DOM标准（专门操作网页内容的API）

+BOM（专门操作浏览器窗口的API，无标准）

二、如何编写JS

1、JS如何运行：

(1)浏览器中的js引擎（解释器）

浏览器中包含2个引擎：

内容排版引擎→解析html和css代码

脚本解释引擎→解释并执行js脚本程序

\*脚本：不需要预编译就可直接执行的程序

\*解释执行：默认自上而下 逐行执行

(2)独立安装的js解释器：node.js

可以在服务器端执行js脚本程序的软件

2、JS如何编写：

(1)不需要网页，直接在控制台中编写和运行

\*控制台(console)：专门编写和测试js小程序的地方

比如console.log("Hello World!");

log，显示日志功能

"..."，字符串，程序中用于显示的一个或者一串文字，如果一串文字用于显示，必须加引号（单双引号都可）

操作：

执行程序：回车

输入多行程序：shift+回车

清屏：左上角clear图标

复制前一段已经执行的代码：↑

复制后一段已经执行的代码：↓

例如 当前浏览器窗口 的 警示框

window . alert("警示内容");

\*\*注：window. 可以省略不写，但是只有window.可以，不代表其他的可以省略，比如console.

\*\*注：只要弹出警告框，程序和网页都会暂停

当前网页文档 的 写入功能

document . write("...");

如果字符串中出现和程序冲突的特殊符号，比如" "

两种解决方案： I /"

II 改为 '

(2)在网页中编写js：2处

I <script>元素中

随网页解释过程而解释执行

II 任意元素的【事件】属性中，只有当事件发生时自动执行

事件：浏览器自动触发的或用户手动触发的元素状态的改变

例如<button onclick="window.alert('Hello World!')">点击此处</button>

(3)外部js文件中编写：2步，同CSS

I 创建.js文件

例如 alert("Hello World!");

II 在html文件中以src引入.js文件

例如 <script src="1.js"></script>

3、调试程序

Bug：泛指程序中出现的错误

Debug：找到并解决错误

出错的现象 ：预期的效果没反应

解决： F12控制台中错误信息包含3部分

错误类型+错误信息+出错的行号

例如ReferenceError，引用错误（找不到对象）

三、变量

1、什么是变量：内存中存储一个数据的空间，再起一个名字

2、何时使用：程序中，每个数据都要先存在变量中，再使用变量执行计算

3、为什么使用变量：变量可以让程序更灵活

4、如何使用：声明 赋值 取值

(1)声明：在内存中创建一个新变量并起名字

var 变量名;

\*\*注：默认值为undefined（所有变量的默认值）

变量命名：见文知意

驼峰命名：第一个单词首字母小写，之后每个单词首字母都大写

例如listStyleType

规则：允许字母、数字、下划线，不能有特殊符号，比如, .

\*不能以数字开头

\*不能使用保留字

保留字：ES标准中已经使用的有特殊意义的词(比如document)

\*注：强烈建议所有变量在使用前都声明

(2)赋值：将等号右侧的值保存到等号左边的变量中

变量名="值";

\*注：如果是数字，变量名=数字; 没有引号

对变量赋值，相当于用新值替换旧值

\*任何运算都无法修改变量中的值，只有赋值 = 才能修改变量中的值

强烈建议在声明同时就初始化变量的值

初始化：首次给变量赋值

var 变量名="值";

(3)取值：编写程序时，使用变量名参与运算

执行程序时，使用变量名的地方自动取出变量中的值代替

例如var drink; 或者var drink="可乐";

drink="可乐"; console.log("豪华套餐：");

console.log("豪华套餐："); console.log(drink);

console.log(drink);

(4)特殊情况： I 仅声明，未赋值 →可以使用

II 未声明的变量 →不能使用（出现ReferenceError）

III 未声明直接赋值 →可以使用（自动创建同名变量），但是强烈建议所有变量都要用var声明

(5)实例： var bj=0;

var xj=0;

等同于var bj=0,xj=0;

\*但是var bj,xj=0;只表示xj=0，bj=undefined

5、声明提前：在正式执行脚本之前，将所有的var声明的变量，提前预读到（当前作用域的）顶部，集中声明创建，然后再开始执行程序，但是【赋值留在本地】！

例如console.log(n);

var n=100;

console.log(n);

结果是什么？？？

→ var n=100;实际为var n;n=100;变量n提前到顶部，但是赋值100还在原位置

所以结果是undefined和100

四、常量

1、什么是常量：一旦创建并初始化后，值不可改变的量

2、何时使用：保存程序中一尘不变的值，比如 π(pai)，g(重力加速度)

弧度：1半径(r)=1弧度，360度角=2π弧度

3、如何声明并初始化：const 常量名="值";

\*一般常量名全部用大写字母

五、数据类型：2大类

1、原始类型【数据值】：值直接保存在变量本地的数据类型

5种：Number String Boolean Undefined Null

(1)Number：js中专门存储数字的类型，既可以是整数，也可以是浮点数（小数）

何时使用：计算、做比较

\*但是如手机号、银行卡号、身份证号、QQ号此类的→字符串，需加" "

所占空间：

整数： 32bit=4bytes

1byte字节=8bit位；1kb=1024bytes

浮点数：64bit=8bytes

\*内存中数字的大小和所占空间无关

\*但是比如数字2.0，计算机会识别为2（整数）

舍入误差：因为计算机中也有无法除尽的数字，导致计算机的误差，而且舍入误差无法避免！

近似解决：按指定位数四舍五入，比如console.log(change.toFixed(2));表示将change中的值按2位小数四舍五入

(2)String:专门保存一串字符的序列，必须带引号

何时使用：凡是仅用于显示的数据

所占空间：js中字符串中每个字符都用Unicode号存储的

unicode号：对全球主要文字中的字符，每个字符编一个号

英文字符、数字 ≤126，占1个字节

汉字字符 19968~40869，占2个字节

\u4e00~\u9fa5（十六进制表示法）

比如"Hello宝马"，5+2+2=9个字节

\*字符串的内容一旦创建，不可改变！要想改变，只能用另一个新字符串替换

例如I var str="Hello";

str=str+"World"; 问共创建了几个字符串？

→Hello一个，World一个，Hello World一个，共3个

例如II var a="a1";

var b="b1";

var c="c1";

var d=a+b+c; 问共创建了几个字符串？

\*注：a+b+c只能两两相加，先ab+c，然后等于abc

→a一个，b一个，c一个，ab一个，abc一个，共5个

(3)Boolean：布尔类型，只有两个值的数据类型，两个值为true、false

何时使用：表示二选一的时候

(4)Undefined：只有一个值，undefined

何时使用：变量未赋值时，它自动赋值使用

(5)Null

2、引用类型【对象】：值不直接保存在变量本地的数据类型

六、数据类型转换

1、js是弱类型语言：3方面

(1)声明变量时，不需要规定变量存储的数据类型

(2)赋值时，才动态判断数据的类型；同一个变量先后可以保存不同类型的值

(3)js会在需要时自动【转换】数据的类型

2、两大类转换方式：

(1)隐式转换：不需要程序员干预，js自动完成的类型转换

【算数计算】中的隐式转换：

I 算数计算中，一切数据都默认转为数字再计算

II 在+运算中，碰到字符串，+就变为拼接（如果其中一个为非字符串的数据，则被自动转为字符串）

III Boolean类型：true→1，false→0，前提是在算数运算中，如果在+中有其他字符串，则会变成"true"和"false"

IV 如果数据无法被转换为数字类型，会被转为NaN

NaN：Not a Number，表示【不是一个数字】的【数字】

\*它不等于、不大于、不小于任何值

\*其参与任何运算，运算结果都为NaN

表达式：变量、数据和运算符组成的公式

\*默认都是【从左向右】，【两两】运算

任何表达式都有1个运算结果，所以任何表达式都可以当一个值使用

(2)强制转换：程序员主动调用专门的函数执行的转换

何时使用：自动转换的类型不是想要的结果的时候

typeof(n)，返回n的数据类型名称

例如 var n=100;

console.log(typeof(n));

结果是Number

I 任意类型→字符串：2个API

①n.toString()

n不是undefined和null时才可使用

例如 var n=100;

console.log(typeof(n.toString()));

结果是String

②String(n)

例如 var m;

console.log(String(m));

结果是Undefined

相当于隐式转换，比如

var n1=2,n2="3";

console.log(n1+n2);

输出结果为23，程序会先把2用String(2)转换为"2"，再+"3"，所以结果为"23"

II 任意类型→数字：3个API

①Number(n)

专用于将【非字符串类型转数字】，即当n不是字符串类型时，比如Boolean类型

相当于隐式转换，比如

var n1=2,n2=true;

console.log(n1\*n2);

输出结果为2，程序会把true用Number(true)转换为1，再乘以2，所以结果为2

number(“”)输出结果为0

number(“ ”)输出结果为0

②parseInt(str)

将【字符串转数字】，从第一个字符开始，依次读取每个数字，只要碰上第一个非数字字符就停止

可以自动跳过开头的空字符

例如var width="60px";

console.log(parseInt(width));

输出结果为60，但是如果要在原来的基础上加10px，则要

width=parseInt(width)+10+"px";

console.log(width);

输出结果为70px

\*Int表示整数，所以parseInt(str)不认识小数点

何时使用：将字符串转为整数时

\*\*任意整数间(min~max)取1个随机整数：

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

例如：在0~9之间随机取一个数

var n=parseInt(Math.random()\*(9-0+1)+0);

③parseFloat(str)

\*Float表示浮点数，所以parseFloat(str)认识【第一个】小数点

何时使用：将字符串转为小数时

其他用法同parseInt(str)

例如var n=true;

console.log(Number(n));

console.log(parseInt(n));

第一行输出结果为1；第二行为NaN，因为parseInt(字符串)，n会被先String(n)转换为"true"，所以parseInt("true")的结果就是NaN

III 任意类型→Boolean

Boolean(n)，相当于隐式转换

\*只有五个值（"空字符串" NaN undefined null 0）可以转换成false，其余都转换成true！

(3)快速转换：

I 任意类型→字符串：n+"";

II 任意类型→Boolean：!!n;

3、如何请用户输入一个数据

var input=window.prompt("提示信息")

或者var input=prompt("提示信息")

程序运行到prompt时弹出一个“输入框”，程序暂停，等待用户输入，用户输入数据并点击确定，程序将输入的数据保存到input变量中，程序继续向下执行

七、运算符和表达式

什么是程序：人脑的想法在计算机中的执行

运算符：让程序模拟人类想法的特殊符号

表达式：变量、数据、运算符组成的公式

1、算数运算：+ - \* / % ++(递增) --(递减)

(1)%，取余数，也叫模运算

n%m：n除以m，不取商，而取除不尽的余数

比如17%3，商是5，余数2，所以结果为2

何时使用：

I 判断能否整除，比如判断奇偶数

例如n%2结果为0，则表示没有余数，可被整除，为偶数

例如n%2结果不为0，则表示有余数，不可被整除，为奇数

II 保证运算结果绝不超过除数

(2)++ --

I ++n:将变量n中的值+1，再存回n中，相当于n=n+1

表达式返回n的【新】值

II n++：将变量n中的值+1，再存回n中

表达式返回n的【旧】值

例如var n=3;

console.log(n++ + ++n + n++);

//n=4，返回旧值3

//n=5，返回新值5

//n=6，返回旧值5

运算结果为13

2、关系运算

将两个值做比较，返回值只有true和false两个结果

> < >= <= == !=

\*在关系运算中：

(1)默认将所有类型【转为数字】再比较

(2)如果参与比较的两个值【都是字符串】，依次取出每个字符串的字符，比较unicode

(3)何时使用：关系运算在程序中一般充当判断条件

(4)两个特殊情况：

I NaN，它不大于、不小于、不等于任何值，包括它自己

\*NaN和任何数据做!=比较时，始终返回true，所以这种方法无法判断一个数据是否为数字

解决方法：isNaN(数字)，本意是判断一个值是否是NaN，如果是，返回true，否则返回false

\*如何判断一个值是否为数字：

isNaN(n)→true→不是数字

isNaN(n)→false→是数字

或者在isNaN前面加！，表示反义

!isNaN(n)→true→是数字

!isNaN(n)→false→不是数字

II undefined和null

undefined：所有未赋值的变量的默认值（【自动】赋值）

null：【主动】释放或清空一个变量中的值

全等号：===

类型相同，而且值相等，只有同时满足这两个条件才可返回true，否则为false

\*本身不带隐式转换

何时使用：区分undefined和null时，以及只要类型相同，且值相等时

3、逻辑运算

&&(且) ||(或) !(非)

将多个关系运算综合起来，得出最终结论，返回值有true和false

何时使用：程序中，只要判断条件超过2个时（包括2个）

(1)&&：两个条件都必须为true，才返回true；只要其中有一个为false，则返回false

(2)||：两个条件只要有一个为true，就返回true；除非都为false，则返回false

\*\*短路逻辑：逻辑运算中，只要前一个条件已经可以得出最终结论，那么后续条件不再执行！

如何利用短路逻辑：

I 一件事，满足条件就做，不满足条件就不做

条件&&(操作);

例如var money=600;

/\*如果money>=500，就打八折（如果不到500，就什么都不做）\*/

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

console.log(money);

输出结果为480

何时使用：【操作】仅一两句话时

II 如果前一个值有效，就用前一个，不往后执行；否则就用后一个

值1||值2;

\*逻辑运算中的隐式转换：将所有参与逻辑判断的条件都转为Boolean类型

例如console.log(4||5);

输出结果为4

console.log(0||5);

输出结果为5

4、位运算：左移<< 右移>>

n<<m，将n的二进制数所有位，左移m位，\*相当于n\*2的m次方

例如1<<3，结果为8

n>>m，将n的二进制数所有位，右移m位，\*相当于n/2的m次方

例如8<<3，结果为1

5、扩展赋值运算：+= -= \*= /= %=

n+=m，相当于n=n+m，将n中的值增加m，再存回n中

n-=m，相当于n=n-m，将n中的值减少m，再存回n中

... ...

何时使用：只要直接递增或递减或... 原变量时

6、三目运算，三元运算，条件运算

根据不同的条件，执行不同的操作或者返回不同的值

语法：条件1?值1或操作1: （如果满足条件1，就返回值1或执行操作1）

条件2?值2或操作2: （条件1不满足，如果满足条件2，就返回值2或执行操作2）

...?... : ... ...

默认值（或默认操作） （以上条件都不满足，就返回默认或执行默认操作）

总结：多个条件，多件事，多选一执行

例如：console.log((year%4==0&&year%100!==0)?"闰年":

"平年");

原则：如果操作语句比较简单，建议用三目运算替代if...else

7、交换运算

比如将x y按照从大到小的顺序排列，就要交换位置才能实现

交换三部曲：x+=y , y=x-y , x-=y

如果x=5，y=6，x+=y结果为x=11

y=x-y结果为y=5

x-=y结果为x=6，实现交换

8、闰年计算条件

(year%4==0&&year%100!==0)||(year%400==0)

八、函数

1、封装一项专门任务的步骤清单的代码段，起一个名字；或者解释为程序中提供的一项服务的步骤说明

何时使用：当一项任务需要反复执行，但又不希望重复编写时，也叫作【代码重用】

如何声明函数：

function 任务名([参数变量列表]){

步骤清单代码段

[return返回值]

}

\*注：[ ]表示可有可无，但是()必须有，哪怕是空的

任务名：也叫【函数名】，是指向函数定义的一个变量

函数：封装函数定义的引用类型对象

\*\*函数在声明时，不【执行】！也不【读取】内部的代码！

如何调用函数执行：

在任何位置：任务名([参数变量列表]); 调用几次执行几次

\*\*函数在调用时，才读取内部的代码并开始执行！

2、参数变量：专门接受要传入方法中处理的数据的【变量】

何时需要定义参数：如果一个函数必需一些数据才可正常执行；需要几个数据就定义几个参数变量

何时为参数赋值：在调用函数时，按照参数定义时的顺序和个数，依次传入参数【值】

3、返回值：函数的执行结果

何时需要：只要函数需要有明确的执行结果时，或者如果调用者需要获得明确的执行结果时

如何定义返回值：函数定义内部，一般函数体结尾

return 值;

调用者何时、如何获得返回值：

(1)一个有返回值的函数调用，可以当做一个值使用

(2)调用时，可使用变量保存住函数调用的返回值

4、变量作用域：一个变量的可用范围（2种）

(1)全局作用域：window

全局变量：放在全局作用域中的变量

可在程序的任何位置访问全局变量

(2)局部作用域：在函数调用时才创建的作用域

局部变量：（2种）

I 参数变量

II 在函数定义中var的变量

\*仅在函数调用时动态创建，调用后，随局部作用域一同销毁

\*\*\*函数调用时，如果局部有，就不用全局的！！！

5、声明提前

正式开始执行程序前，先将var声明的变量和function声明的函数提前到当前作用域的顶部，集中声明，但是赋值留在原地

例如function fun(){ function fun(){

console.log(1); console.log(1);

} }

fun(); 提前后的结果为: function fun(){

function fun(){ console.log(2);

console.log(2); }

} var fun; //声明提前

fun(); fun(); //第一个function已经无效，这里为2

var fun=100; fun(); //为2

fun(); fun=100; //赋值保留

fun(); //上面fun已经赋值100，函数已丢，报错

例如 function fun(){ function fun(){

console.log(n); var n; //提到当前作用域的顶部，不会再往上去

var n=100; 提前后的结果为: console.log(n);

console.log(n); n=100;

} console.log(n);

}

6、按值传递

两变量间赋值，或将变量作为函数的参数传递时都仅将变量中的【值】，复制一个【副本】给对方

例如 var n=100;

var m=n;

m++;

console.log(n);

→结果为100，因为var m=n之后，m++已经跟n没了关系

九、全局函数

ES标准中规定的，由浏览器厂商实现的，不需要任何对象前缀就可直接访问的函数

例如parseInt/parseFloat(str) isNaN(n)此类的

1、编码 解码

\*url中不允许出现多字节字符

解决： 使用函数encodeURI()，将多字节字符编码为utf-8格式的单字节字符，再发送给服务器

使用函数decodeURI()，将收到的编码后的单字节字符，解码为多字节字符原文

\*url中不允许出现url的保留字符，比如： : /

解决： 使用函数encodeURIComponent()，编码多字节字符和保留字符

使用函数decodeURIComponent()，解码

→所以，后者是全能的，前者可以丢弃不用

2、eval：执行字符串格式的语句或表达式

例如 "alert("Hello");" 加引号后变为字符串，执行起来没有任何效果

但eval("alert("Hello");");就能执行了

3、js中，任何数除以0，【不报错】，返回infinity；但是在其他语言里面会报错

isFinite(n)，判断n是否在有效数字范围内

如果返回true，有效；如果返回false，很可能分母为0

比如 console.log(isFinite(4/2)); →true

console.log(4/0); →infinity

console.log(isFinite(4/0)); →false

十、分支结构

1、程序有三种结构：

顺序结构：默认程序都是自上向下逐行顺序执行

分支结构：根据不同的条件，选择执行不同的操作

看操作的复杂程度，简单的用三目，复杂的用分支

循环结构：让程序反复执行同一代码段

2、分支结构有三种情况

(1)一个条件，一件事：满足条件就执行，不满足就什么都不做

操作简单时：短路逻辑

操作复杂时：if结构，如果满足条件就执行代码段

语法：if(条件){

满足条件时才执行的代码段

}

(2)一个条件，两件事：如果满足条件，就执行操作1，否则，执行操作2，二选一执行

操作简单时：三目运算

操作复杂时：if...else结构

if(条件){

满足条件时才执行的代码段

}else{

不满足条件时才执行的代码段

}

(3)多个条件，多件事：如果满足条件1，就执行操作1，否则，如果满足条件2，就执行操作2，。。。 。。。[否则，默认操作]。多选一执行，也有可能都不执行

①if...else if结构：

if(条件1){

满足条件1时才执行的代码段

}else if(条件2){

满足条件2时才执行的代码段

}else if(条件3){

满足条件3时才执行的代码段

}... ...

[else{

如果以上条件都不满足才执行的（默认）代码段

}]

\*结束时跟三目不同的是: 可以没有默认操作

②switch...case结构：

switch(表达式){

case 值1:

满足值1时才执行的代码段

case 值2:

满足值2时才执行的代码段

...

...

default:

如果前面的值都不满足，执行默认代码段

}

何时使用：当条件都是【全等比较】时，才可使用switch结构

\*执行原理：计算表达式的结果，用表达式的值和每个case的值做【全等】比较，碰到一个全等的case值，则进入该case开始执行，并默认依次触发之后所有case的执行！

break：【中止】当前结构的执行，并跳出结构

使用位置：每个case之间，最后一个case后面没必要加

何时可以省略：上下两个case希望执行相同代码时

例如 case 2:

case 3:

console.log("维护中。。。");

break;

continue：中止【本轮】循环，继续下一轮循环

十一、循环结构

1、循环：让程序反复执行同一段代码

2、循环三要素：

(1)循环条件：让循环【继续】执行的条件（不是让循环退出的条件！）

(2)循环变量：循环条件中用于比较的变量

从几开始，（到几结束），每次变化几

循环变量总要向着不满足循环条件的趋势不断变化，否则无法结束

例如 游乐场坐小火车，循环条件：圈数<3

循环变量：圈数

(3)循环体：循环内反复执行的代码段

3、三种语法结构：

(1)while语法： var 循环变量=初始值;

while(循环条件){

循环体;

迭代修改循环变量;

}

→先判断循环条件，再执行循环体

何时使用：当循环变量的变化没有规律时

(2)do...while语法： var 循环变量=初始值;

do{

循环体;

迭代修改循环变量;

}while(循环条件);

→先执行循环体，再判断循环条件

\*while和do...while区分：就看第一次条件是否满足

\*\*【第一次条件满足，两者完全等效】

第一次条件不满足，while一次都不执行

do...while至少可以执行一次

何时使用：即使不满足循环条件，也希望至少执行一次时

(3)for循环：

for和while完全等效

语法： for(var 循环变量=初始值;循环条件;迭代修改循环变量;){

循环体;

}

for语法简写： var sum=0;

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

sum+=i;

}

可以改为 for(var i=1,sum=0;i<=100;sum+=i,i++);

或 for(var i=1,sum=0;i<=100;sum+=i++);

\*sum+=i++这样写是看题目的要求，如果是++i就不一定能这样简写

\*for(;;)和while(true)完全等效，表示死循环，只能在循环体中用break打断循环！

十二、数组

1、概念： 一组变量的集合，起一个统一的名字(用法)

保存多个数据的引用类型的对象(本质)

2、程序=数据结构+算法

算法：程序执行的步骤

数据结构：数据在内存中的存储结构

好的数据结构，可以极大的提高程序的执行效率

相关的一组数据，都要用数组集中管理

3、如何使用数组：创建 取值 赋值

(1)创建：4种

①创建空数组：var 数组名=[];

或者var 数组名=new Array();

何时使用：暂时不知道数组内容

②创建同时初始化数组内容：var 数组名=[值1，值2，...];

何时使用：创建时就已经知道数组的内容

③创建指定元素个数的数组：var 数组名=new Array(n);

创建一个新数组对象，同时初始化n个空元素

④创建同时初始化数组内容：var 数组名=new Array(值1，值2，...);

(2)取值：获取数组中指定位置的元素内容

下标：数组中标识每个元素位置的序号

默认，自动从0开始，每个加1，到元素个数-1结束

如何取值：数组名[下标];

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

//下标：0 1 2 3 4 5 6

console.log("今天是星期："+week[5]);

(3)赋值：将新值放入指定位置的元素中保存

如何赋值：数组名[下标]=新值;

4、数组中每个元素【完全等效】于一个变量，用法和变量完全相同

5、数组是引用类型的对象：

数组名作为变量赋值或参数传递时，都是将地址值复制给对方

如果对方修改数组的内容，访问原数组名也会受影响

例如 var my=["包子","包子","包子"]; //【引用类型的对象】

var your=my; //my将保存到数组【地址值】复制给了your

your[0]=""; //吃掉第一个包子

console.log(my); //结果还剩2个

而 var m=100; //【原始类型的值】

var n=m; //将m中的值【复制一个副本】给n

n--; //修改n，不影响m

6、null：主动释放一个对象的引用，null本身是一个原始类型的值

垃圾回收器：伴随主程序，同时运行的后台程序

记录每个对象当前被引用的次数，如果被引用的次数==0，垃圾回收器自动释放对象内存空间

强烈建议：在使用完较大的对象后，主动赋值为null

7、获取数组的元素个数：

arr.length属性（是一个值），不加圆括号

固定套路：

(1)获取数组最后一个元素：arr[arr.length-1];

(2)向数组末尾追加一个新元素：arr[arr.length]=新值;

8、js中的数组 与 其他语言中的数组：3个区别

(1)不限制元素的数据类型

(2)不限制数组长度

(3)允许越界

赋值时：自动创建指定下标的元素，保存新值，length属性会随最大下标而变化

取值时：试图从没有下标的位置取值，会返回undefined

9、遍历数组：从第0个元素开始，依次获取每个元素，然后对每个元素执行相同的操作

下标是天生的循环变量！

常用写法：for(var i=0;i<arr.length;i++){}

10、冒泡排序

从数组第一个元素开始，依次比较前后两个元素的大小，如果当前元素>下一个元素，则交换两元素中值的位置

for(var r=1;r<=arr.length-1;r++){ //r表示轮数

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

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

arr[i]+=arr[i+1];

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

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

}

}

}

11、关联数组，又叫hash数组

(1)概念：可自定义元素下标名称的数组

(2)索引数组：自动从0开始分配连续不重复的序号下标

(3)如何创建关联数组：2种

①先创建普通空数组，再追加自定义下标的元素

例如 孙大圣的考试成绩数据如下：

var xx=[];

xx["name"]="孙大圣";

xx["sx"]=81;

xx["yw"]=53;

xx["yy"]=89;

②创建数组的同时，初始化元素内容

例如 孙大圣的考试成绩数据如下：

var xx={"name":"孙大圣","sx":81,"yw":53, ...};

↑ ↑

key value

所以也称为：关联数组中保存的是多个key(键)/value(值)对

\*(4)关联数组中的key不能重复！！

\*(5)关联数组的length属性无效！！ 【length=0】

(6)优势(何时使用)：查找极快，并且和元素个数无关

(7)如何遍历关联数组：

for(var key in arr){ //依次取出arr中每个key，存在变量key中

arr[key] //获取当前正在遍历的key中存储的值

}

12、数组API

什么是对象：封装数据，并提供对数据的操作方法的整体

什么是数组对象：封装了一组相关的数据，提供了对数据的操作方法

(1)数组转字符串：2种

①var str=arr.toString();

默认只能以【逗号】分隔每个元素

②var str=arr.join("连接符");

可以自定义元素间的连接符（分隔符）

固定套路：

I 将字符数组，无缝拼接为单词，比如：

var chars=["H","e","l","l","o",];

console.log(chars.join(""));

II 将单词数组拼接为句子：

chars.join(" ");

III 将数组转化为页面上的列表或选择元素

见实例

(2)连接数组

不修改原数组对象，返回新数组对象

concat:var newArr=arr.concat(值1,值2,数组, ...);

例如 var arr=[1,2,3];

var arr2=[11,22];

var newArr=arr.concat(12,23,123,arr2);

console.log(newArr);

→第三行var newArr必不可少，输出结果为[1,2,3,12,23,123,11,22]

\*(3)获取子数组

不修改原数组对象，返回新数组对象

slice:var subArr=arr.slice(starti,[endi+1]);

starti:开始的位置下标

endi:获取到的位置下标

含头不含尾！

slice支持倒数下标

如果省略endi，则从starti一直取到结尾

(4)splice：删除、插入、替换

①删除：var deletes=arr.splice(starti,n);（n表示个数）

\*直接修改原数组对象

返回：被删除元素组成的临时数组

注：如果直接调用arr.splice(starti,n)，则输出被删除后剩下的元素；如果新定义var deletes，则输出被删除元素组成的临时数组

例如 var arr=[1,2,3,4,5];

// 0 1 2 3 4

arr.splice(2,2);

console.log(arr);

→输出剩下的元素，结果为[1,2,5]

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

var deletes=arr.splice(2,2);

console.log(deletes);

→输出被删除元素组成的数组，结果为[3,4]

②插入：arr.splice(starti,n,新值1,新值2, ...);（n表示要删除的个数）

\*直接修改原数组对象

例如 var arr=[1,2,3,4,5];

// 0 1 2 3 4

arr.splice(3,0,11,22);

console.log(arr);

→从下标为3的位置开始，一个也不删除，结果为[1,2,3,11,22,4,5]

③替换：删除旧值，插入新值

var deletes=arr.splice(starti,n,新值1,新值2, ...);

例如 var arr=[1,2,3,4,5];

var deletes=arr.splice(2,2,11,22,33);

console.log(arr);

console.log(deletes);

→第一行输出[1,2,11,22,33,5]

第二行输出[3,4]

(5)颠倒数组的元素：

arr.reverse();

例如 var arr=[1,2,3,4,5];

arr.reverse();

console.log(arr);

→输出结果为[5,4,3,2,1]

(6)数组排序

arr.sort();

直接修改原数组

\*默认sort方法，只能将所有元素转为字符串再排序，无法对数字类型排序

解决办法：自定义排序规则（2步）

①先定义比较器函数对象

比较器函数：专门封装任意两数比较逻辑的函数

参数必须是2个

函数体：比较两个数的大小

返回值：如果a>b，必须返回>0的数；如果a<b，必须返回<0的数；如果a=b，必须返回0

例如 数字比较器： function shuzi(a,b){

return a-b; //a-b为升序，b-a为降序

}

a始终是当前数，b是下一个，不能调换位置

\*\*\*②将比较器函数对象传入sort方法内

\*js中函数也是一个对象！

函数对象可以被当做参数传递！传递到哪儿，就可以在哪儿调用

固定套路： function shuzi(a,b){return a-b;}

arr.sort(shuzi);

(7)栈 和 队列（都是用数组模拟）

①栈：一端封闭，只能从另一端进出的数组

先进的后出，后进的先出，比如坐地铁（先上的走到车厢里面，后上的站门口，先上的只能后出）

I 从数组【末尾】出入栈：其他元素的下标不受影响

入栈：arr.push(新值);

出栈：var last=arr.pop();

II 从数组【开头】出入栈：每出入栈一次，其余下标都要顺移

入栈：arr.unshift(新值);

出栈：var first=arr.shift();

何时使用：【仅从一端】进出数组时，不论开头或结尾

②队列：从一端进入，从另一端出的数组

先进的先出，后进的后出，比如排队买手机

\*如果入栈用push，出栈则用shift；如果入栈用unshift，出栈则用pop

13、优化

频繁字符串拼接，推荐使用数组的join方法作为替代

步骤：

(1)先将要拼接的子字符串，放入一个数组中，临时保存

(2)遍历结束后，一次性join成一个完整字符串

例如 for(var i=0,str=[];i<l+r-1;i++){

str[str.length]=(i<l-r)?" ":"\*";

}

str=str.join("");

console.log(str);

14、求最大公约数

固定套路： a>=b;

r(余数)=a%b;

a=b;

b=r;

直到b==0结束

最后的a就是最大公约数，即console.log(a)

十三、二维数组

1、概念：数组中的某个元素，又指向另一个子数组对象

2、何时使用：两种情况

(1)存储横行竖列的二维数据时，比如2048游戏

(2)存储上下级包含关系的数据，比如省份和城市

3、如何创建：两种方法

(1)先创建普通空数组，再逐个元素追加子数组

例如 var data=[];

data[0]=[0,0,0,0];

data[1]=[0,0,0,0];

data[2]=[0,0,0,0];

data[3]=[0,0,0,0];

(2)创建的同时就初始化子数组 (用的比较多)

例如 var data=[

[0,0,0,0],

[0,0,0,0],

[0,0,0,0],

[0,0,0,0]

];

4、访问二维数组中指定位置的元素

arr[row行][col列]

比如 var data=[

// 0 1 2 3

[0,0,0,0], //0

[0,0,0,0], //1

[0,0,0,0], //2

[0,0,0,0] //3

];

→如果要输出第二行，第三列就是console.log(data[1][2]);

→如果要修改第三行，第二列的元素为4就是data[2][1]=4;

5、遍历二维数组

\*\*外层循环遍历【行】，内层循环遍历【列】

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

for(var col=0;col<arr[row].length;col++){

arr[row][col]; //当前元素

}

}

6、二维数组的【行下标row】不能越界