Js(下)

一、String：多个字符组成的【只读】字符数组

1、只要数组中不直接修改原数组的API，字符串都能用！

例如 str[i]，表示获得字符串中下标为i的一个字符

str.length，获得字符个数

str.slice(starti[,endi+1])，获得子数组

2、内置对象：ES标准中规定的，厂商已经实现的对象，有11个

String Number Boolean --包装类型

Array RegExp Math Date

Error

Function Object

Global (ES标准中是全局对象，但是在浏览器中被换成了window)

3、包装类型：专门封装原始类型的值，并提供对其操作的方法

(1)String Number Boolean

(2)如何创建：试图对原始类型的值调用方法时，js自动创建包装类型的对象

\*创建和【原始类型值】相对应的包装类型对象

(3)为什么要有包装类型：原始类型的值没有任何方法！

所以要先创建包装类型的对象，封装原始类型值，才能操作原始类型的值

(4)生命周期：调用方法时自动创建，方法调用完自动释放

4、String(x) 和 new String(x)

前者：将任意类型转换为字符串类型，返回值是【原始类型值】

后者：创建包装类型的对象，返回值是【引用类型的对象】

5、转义字符：将特殊意义的字符，转为普通文字

何时使用：当字符串中包含和语法冲突的特殊符号时

(1) \n，表示换行

\t，表示缩进(Tab键功能)

(2) \"，正常输出双引号

\'，正常输出单引号

\\，正常输出/，比如在需要打印路径的时候，里面包含斜线，为了防止单斜线被解析成转义字符，双斜线可以保证输出正确路径

6、String的API

\*所有API都无法修改原字符串，都是返回新字符串

(1)大小写转换

var newStr=str.toUpperCase(); 所有字母转大写

var newStr=str.toLowerCase(); 所有字母转小写

何时使用：凡是大小写字母一视同仁时，比如输入验证码

(2)获取指定位置的字符

①var char=str.charAt(i);

之前的方法是：var char=str[i];

②获得i位置字符的unicode，返回的是一个数字

var unicode=str.charCodeAt(i);

(3)检索关键字

①找右侧下一个匹配的关键词位置

var i=str.indexOf("关键词"[,fromi]);

②找左侧下一个匹配的关键词位置

var i=str.lastIndexOf("关键词"[,fromi]);

以上两种都是返回关键词中第一个字符的下标位置

fromi：都指从哪个位置开始找

省略fromi， indexOf默认从0开始找

lastIndexOf默认从length-1开始找

(4)截取子字符串：3种方法

①str.slice(starti[,endi+1]);

支持负数参数

②str.substring(starti[,endi+1]);

不支持负数参数，但是可以使用arr.length-n代替负数下标

③str.substr(starti,n);

n表示取的个数

\*例如在身份证里面取生日可用3种方法实现：

var pid="620321199010023051";

var birth=pid.slice(6,-4);

var birth=pid.substring(6,pid.length-4);

var birth=pid.substr(6,8);

(5)分隔字符串

str.split("分隔符");

分隔符：只要有规律的字符，不论长短

何时使用： ①希望将字符串按照规律分割为数组时

②只要分段处理字符串时，都要先切割，再遍历

二、正则表达式

1、概念：规定字符串中字符出现规则的表达式

简写reg

2、何时使用：查找、验证、替换、切割

3、规则：

(1)最简单的正则表达式：关键词原文----必须和原文完全匹配

(2)备选字符集：规定某【一位】字符的备选字符列表，【必须且只能多选一】

①语法：[备选字符列表]

比如 我去、我草

→我[去草] (此处意思为取'我去'或者'我草'，并不是指'我去草')

比如 我去去、我草草、我去草

→我[去草][去草]

\*②一个中括号[]，里面内容不论是什么，有多长，只能匹配【一个字符】！

③如果备选字符部分【Unicode号连续】，可用 - 连接开头字符和结尾字符

比如 要取数字0-9之间的任意一位

[0123456789] → [0-9]

比如 要取一位小写字母

[a-z]

比如 要取一位字母

[a-zA-Z]

比如 要取一位数字或字母

[a-zA-Z0-9]

\*但是[一 - 十]不能实现，因为Unicode号不是顺连的

(3)预定义字符集：针对常用的备选字符集提供的简化符号

①[0-9] → \d 表示一位数字

②[a-zA-Z0-9\_] → \w 表示一位数字或字母或下划线

③空字符 → \s 表示一位空字符，比如换行、Tab、空格

④ . 表示除换行外，其余所有字符

⑤大写的 \D \W \S

表示【非】，比如\D表示除了一位0-9数字的任意字符

(4)量词：规定字符集出现的次数

\*默认一个量词，仅修饰【左侧紧邻】的字符集

①确定数量：

I {m,n} → 表示至少m次，最多n次

比如 \d{3,4}表示出现3位数字要选，4位数字也要选

II {m,} → 表示至少m次，无上限（m次以上）

III {m} → 表示必须m次

②不确定数量：

I \* → 表示可有可无，多了不限

II + → 表示至少1次，多了不限（1次以上）

III ? → 表示可有可无，最多一个

比如 要同时匹配：我去，我草，我去去，我草草，我去草

→我[去草][去草]?

(5)选择和分组

①选择： | ，表示或

②分组： ()

何时使用：希望多个字符集被同一个量词修饰时

(6)手机号码规则：

+86或0086 可有可无，最多一个 (\+86|0086)? (+前面要加\转义字符，否则+用意不明)

空字符 不限制个数 \s\*

1 1

3,4,5,7,8 选其一 [34578]

9位0-9的数字 \d{9}

→综合起来(\+86|0086)?\s\*1[34578]\d{9}

(7)指定匹配位置

①开头和结尾： ^表示开头的...

$表示结尾的...

何时使用：仅匹配开头的规则和结尾的规则时使用

固定搭配：^正则表达式$ → 表示从头到尾完整匹配

何时使用：验证时，必须前加^后加$

比如 匹配字符串开头的所有(1个以上)空字符 → ^\s+

匹配字符串结尾的所有(1个以上)空字符 → \s+$

同时匹配开头或结尾的空字符 → ^\s+|\s+$

②单词边界： \b

一个单词必须前后有空字符分隔

比如 no zuo no die nothing

使用\bno\b只会匹配到两个单独的no，而nothing中的no不会匹配到

(8)密码强度验证

8位密码，字母数字组合，至少包含一位大写字母和一位数字

分3步： ①预判字符串不全由小写字母和大写字母组成

(?![a-zA-Z]+$)

②预判字符串不全由小写字母和数字组成

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

③定义正常逻辑

[a-zA-Z0-9]{8}

? 表示预判的意思

综合起来得到→^(?![a-zA-Z]+$)(?![a-z0-9]+$)[a-zA-Z0-9]{8}&

4、练习

(1)身份证号

15位数字

2位数字

1位数字、x、X

后三位可有可无，最多一次

→\d{15}(\d\d[0-9xX])?如果要验证则是^\d{15}(\d\d[0-9xX])?$

(2)汉字(1位)

[\u4e00-\u9fa5]

三、String中支持正则的API

1、查找：4种

①已知关键字原文，要求查找完整匹配的关键字位置，也就是说找固定内容的1个关键词

var i=str.indexOf("kword"[,fromi]);

var i=str.lastIndexOf("kword"[,fromi]);

这两种方法每次只能找【下一个】，如果找不到，返回-1

\*这两种方法不支持正则表达式！

②判断有没有

var i=str.search(reg);

只找第一个匹配的，如果没有返回-1

\*没有第二个参数，无法修改开始查找的位置，所以只适合判断有没有

无法继续查找之后所有关键词，只能查找第一个

③获得所有关键词的内容和个数

正则结尾 加g （g是global的缩写，意思为全部）

比如var reg=/no/g;

var kwords=str.match(reg);

\*如果 reg加g，表示查找全部匹配关键词

reg，表示默认仅查找第一个匹配的关键词

如果没找到，返回null

\*如果一个结果有可能是null时，都要先判断，再使用

比如 var str="no zuo no die no can no bi bi";

var reg=/no/g;

var kwords=str.match(reg);

if(kwords){ //这句话的意思是如果kwords有限，不是null

console.log("共找到"+kwords.length+"个");

}else{

console.log("没找到");

}

\*正则表达式，默认都是字母【区分】大小写

如果要忽略大小写，就在结尾 加i （i是忽略的意思）

比如 var reg=/no/i;

var reg=/no/ig;

④查找每个关键词的内容和位置

regexp.exec

2、替换

str=str.replace(reg,"替换值");

替换一般都要 加g

用法见练习

3、regexp对象：封装了1条正则表达式，提供了使用正则表达式验证和查找的方法

如何创建正则表达式对象：2种

(1) var reg=/正则表达式/;

var reg=/正则表达式/ig;

何时使用：使用固定的正则表达式时

(2) var reg=new RegExp("正则表达式","ig");

\*正则表达式中所有\都要写成\\，以防止被js解析为转义字符

何时使用：正则表达式由变量拼接而成或在运行时动态改变

两大功能：

(1)查找：var arr=reg.exec(str); 同时获得关键字的内容和位置

①几乎都要加g

②exec每次只能找下一个匹配的关键词

③arr中的第一个元素arr[0]，保存了本次找到的关键词内容

如果找不到，返回null

④arr.index属性中保存了本次找到的关键词的首字母位置，下次会跳过当前关键词继续往下找，假如当前关键词为no，位置为arr[0]，no的长度为2，下次则跳过2个长度开始，即arr[2]；而内容则会被后来找到的所替换，不是累积排列起来，始终只有一个

下次从位置reg.lastIndex处开始查找

比如 while((arr=reg.exec(str))!=null){

console.log("在位置"+arr.index+"发现敏感词"+arr[0]);

console.log("下次从"+reg.lastIndex+"开始查找");

}

⑤两大模式：

①贪婪模式：默认情况下，正则表达式都会匹配尽量长的字符串

②懒惰模式：只匹配最短的符合条件的字符串

贪婪改懒惰：将.\*改(.\*?)

⑥如何获取本次查找结果中的部分内容

第一步：在正则表达式中用()包裹住需要获得的部分

第二步：在本次查找中，使用RegExp.$n获得第n个()中匹配的子内容

(2)验证：var bool=reg.test(str);

①几乎都要前加^，后加$

②验证str是否符合reg的格式规定，符合返回true，不符合返回false

③何时使用：只要验证格式时（唯一的办法）

④如何使用：可以直接放入if中，作为条件

比如 var reg=/.../;

if(reg.test(prompt("输入密码"))){

alert("密码安全");

}else{

alert("密码强度太低");

}

4、分隔

str.split(reg);

四、Math

1、概念：专门封装数学计算常用常量和计算方法的全局对象

全局对象不能new！，所有API都直接用Math.xxx调用，因为Math没有构造函数

2、何时使用：只要进行数学计算时

3、API：

(1)取整：

①上取整： Math.ceil(n)

比如15.1取16 15.9取16

②下取整: Math.floor(n)

比如15.1取15 15.9取15

③四舍五入取整: Math.round(n)

比较 参数范围 功能 返回值

Math.round(n) 任意数字 只能取整 Number

n.toFixed(d) 0-20 按任意小数位 String

(2)乘方、开平方：

①乘方： Math.pow(底数,幂);

比如 Math.pow(10,2);表示10的2次方，等于100

②开平方： Math.sqrt(n);

比如 Math.sqrt(4);表示4开平方，等于2

(3)→所以自定义round函数，按任意位数四舍五入，返回Number

比如 function round(num,d){

num\*=Math.pow(10,d); //将num\*=(10的d次幂)

num=Math.round(num.toFixed(2)); //尽量屏蔽舍入误差

num/=Math.pow(10,d); //将num/=(10的d次幂)

return num;

}

console.log(round(23.456,2)); //结果为23.46

(4)获取最大值、最小值

①Math.max(值1,值2,值3, ...);

取多个【数字】中的最大值

②Math.min(值1,值2,值3, ...);

取多个【数字】中的最小值

\*max、min默认不支持数组作为参数！

→如何让max、min支持从数组中获取最大值、最小值：

var maxValue=Math.max.apply(Math,arr);

var minValue=Math.min.apply(Math,arr);

(5)随机数

Math.random(); 0<=r<1

\*任意max和min之间取随机整数：

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

将parseInt换成Math.floor是一样的效果

比如在10到50之间取随机整数：

parseInt(Math.random()\*(50-10+1)+10);

→parseInt(Math.random()\*51;

五、Date

1、概念：封装一个时间值（1970年元旦，至今的毫秒数），提供对时间操作的方法的对象

2、js中的Date对象里封装的是一个巨大的毫秒数

从1970年1月1日0点0分0秒——到当前时间的毫秒差

3、如何创建：4种

(1)创建一个日期对象，同时保存当前时间

var now=new Date();

\*(2)自定义任意时间

var date=new Date("xxxx/xx/xx xx:xx:xx");

年 月 日 小时 分 秒

其中xx:xx:xx小时、分、秒可以省略

(3)自定义任意时间

var date=new Date(xxxx,xx,xx,xx,xx,xx);

年 月 日 小时 分 秒

其中现实的月份=计算机中的月份+1

(4) var date1=new Date("xxxx/xx/xx");

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

date1.getTime()的意思为取出date1的毫秒数

解释：所有日期的set操作都直接修改原日期对象，一旦计算完，旧日期被覆盖

所以在计算前，先复制日期对象的一个副本，再计算

4、API

(1)每个分量都有一对get/set方法

分量： 年：FullYear(4位) 月：Month 日：Date

星期：Day 时：Hours 分：Minutes

秒：Seconds 毫秒：Milliseconds

get方法用来获取分量的值；set方法用来设置分量的值

set方法会自动调整进制！

比如月份set设置14，因为月份没有14月，所以会自动跳到明年的2月份

\*星期没有set方法，因为星期根据日期推算出来，不需要set

比如 要获得今天是几号

date.getDate(); 结果只会返回今天的号数，并无月份

设置日期的月份为4月

date.setMonth(3);

(2)年、月、日、星期不带s结尾；时、分、秒、毫秒带s结尾（见上面分量表）

(3)返回值

只有Date是从1到31结束，其余都是从0到进制-1结束

\*只有Month需要修正，其余都不需要

因为现实中月份： 1,2,...,11,12

而程序中： 0,1,...,10,11

5、Date计算：2种

(1)两日期对象可以直接相减

结果是【毫秒差】

(2)对每个分量做加减：3步

get分量；做加减；set回去

固定套路：

date.setXXX(date.getXXX()+n);

date.setXXX(date.getXXX()-n);

比如 enddate.setFullYear(enddate.getFullYear()+3);

意思为在入职年份的基础上加3年，即合同到期年份

6、日期转字符串：4个

(1)date.toString();

GMT格式显示，即国际标准化格式

(2)date.toLocaleString();

以操作系统当地时间格式显示，包含时间和日期

(3)date.toLocaleDateString();

以操作系统当地时间格式显示，仅包含日期

(4)date.toLocaleTimeString();

以操作系统当地时间格式显示，仅包含时间

六、利用replace和正则表达式，格式化字符串：2步

首先，利用正则表达式将原字符串分组匹配

然后，将正则表达式传入replace中，使用替换值格式化字符串

替换值中如何获得每个子字符串：$n

七、错误处理

1、错误：程序执行过程中导致程序异常中断的状态

2、Error对象：封装了错误信息的对象

何时创建：发生错误时，js自动创建

3、Error子对象：6种

(1)ReferenceError：引用错误，找不到对象

(2)SyntaxError：语法错误(写的有问题)

(3)TypeError：类型错误，错误的使用类型或类型的方法

(4)RangeError：范围错误，参数超范围

(5)EvalError

(6)URIError

4、错误处理：如何在发生错误时，保证程序不中断

如何错误处理：

try{

可能发生错误的代码

}catch(err){ //err中发生错误时会自动收到创建的error对象

如果发生错误，才执行的错误处理代码

}finally{

无论是否发生错误，都执行的代码

\*一定会在【退出之前】执行！

一半释放用过的对象资源：xxx=null

}

其中finally{...}可以省略

\*如果可以提前预知错误的原因，建议使用if...else...代替try...catch...

执行效率：

try的执行效率低，且(跟if相比)多创建一个error对象

try中尽量包含更少的代码

5、Error对象属性：

(1)err.message，保存了错误的详细原因

(2)err.name，保存了错误对象的名称

6、抛出自定义错误：

见练习 实例

八、重载：

(1)概念：程序中同时定义多个相同函数名，不同参数列表的函数，调用时根据传入参数的不同，动态选择匹配的函数执行

(2)js的语法是不支持重载的，可以使用arguments实现重载效果

(3)何时使用：不同参数，不同操作，但操作名称相同时

(4)目的：减轻调用者调用时选择的负担

(5)arguments：所有函数对象中默认都有的，专门接收调用时传入参数的值的【类数组对象】

类数组对象：长的像数组的对象

①arguments.length：参数值的个数

②arguments[i]：获得下标为i位置的参数值

(6)类数组对象 和 数组：

类数组对象不是Array类型

数组是Array类型的对象

(7)参数变量：提醒调用者，如何正确使用函数

参数变量更简便

九、内存中的函数对象

1、三个阶段

(1)定义时：创建2个对象

①函数对象：封装了函数的定义，但暂时不读取函数定义

②作用域链对象：专门保存函数可用的变量所在位置的对象

(2)调用时：创建1个新对象，并将活动对象的引用，压入作用域链对象中

活动对象：专门临时封装本次调用函数时使用的局部变量

(3)调用后：作用域链中的活动对象引用出栈，活动对象以及其内部的局部变量一同释放

2、如何创建函数对象：3种

(1)声明方式： function 函数名(参数列表){

函数体;

return 值;

}

\*\*只有声明方式才能被提前！

(2)函数直接量： var 函数名=function(参数列表){

函数体;

return 值;

}

等号右边的部分就是函数直接量

(3)new Function: var 函数名=new Function("参数1","参数2",...,"函数体");

例如： ①function compare(a,b){retrun a-b;}

②var compare=function(a,b){retrun a-b;}

③var compare=new Function("a","b","return a-b;");

"a","b"是new Function的一种特殊的语法，并不是表示必须得是字符串类型

3、匿名函数

(1)概念：定义时没有变量引用的函数

(2)何时使用：2种情况

①匿名函数自调：定义完立刻执行，执行完立刻释放

何时使用：一个函数只用【一次】时

如何使用： (function(参数列表){

函数体;

return 值;

})(参数值列表);

②匿名函数回调：将函数作为对象传递给其他函数，由其他函数调用

何时使用：如果一个函数【只被】另一个函数调用时

(3) 优点：节省内存空间

缺点：无法返回使用

4、闭包

(1)问题：

①全局变量：在程序任何位置都可以访问（修改），这种情况也叫作【全局污染】

②局部变量：不可重复用！

解决：既要重用变量，又要保护变量不被污染--闭包

(2)如何创建闭包：3步

①先用外层函数封装一个受保护的局部变量

②在内层函数中操作外层函数的变量

③外层函数将内层函数返回到外部，在外部反复调用

(3)如何判断闭包：3特点

①一定要有函数嵌套

②内层函数使用了外层函数的局部变量

③内层函数被返回到外部，在外部调用

(4)何时使用：既要重用变量，又要保护变量不被污染时

(5)缺点：浪费内存空间

因为outer的活动对象无法释放

(6)判断闭包输出的2句话：

①同一次外层函数调用返回的内层函数，操作同一个变量

②外层函数调用了几次，就有几个受保护的变量副本

(7)get/set访问器：

get访问器：专门从变量中取值的方法

set访问器：专门修改变量值的方法

为什么使用get/set访问器：保护变量不被随意赋值或篡改

var value=get变量名();

set变量名(value);

十、面向对象：OPP

1、什么是对象：程序中描述现实中一个具体事物的属性和功能的结构，一块可以存储多个数据的存储空间

面向对象：程序中都是用一个对象来描述现实中一个事物

事物的属性就会成为程序中对象的属性变量；事物的功能就会成为程序中对象的方法变量

方法就是一个函数！只不过被包含在一个对象中才成为方法

封装：将多个相关的属性和函数集中定义在一个对象中

2、如何创建对象：3种

\*\*js中的对象，底层都是一个关联数组

对象的属性和方法都是关联数组中的元素

window就是最大的一个关联数组对象

所有全局变量和函数都是window的成员

\*方法，其实就是一个保存函数地址的属性而已

(1)创建一个单独的对象：2种

①对象直接量方式：（等号右边大括号里面的内容即就是对象直接量）

var obj={

属性名:属性值,

方法名:function(参数列表){

方法体;

return 返回值

}

}

②使用new创建：2步

先创建空对象；再为空对象追加新成员

\*无论是属性变量还是方法变量，统称为对象的成员

var obj=new Object();

obj.属性名=属性值;

obj.方法名=function(){... ...};

(2)反复创建多个相同结构的对象（利用构造函数）：2步

①定义构造函数

构造函数：专门定义一类对象的统一属性和方法结构的函数

如何定义：

function 类型名/构造函数名(参数列表){

this.属性名=参数;

this.方法名=function(){... ...}

}

\*\*this：在构造函数中表示正在创建的对象；在方法中表示正在调用方法的对象

\*方法中都要使用this.xxx的方式访问当前对象的属性

默认为window

所有无主的都是window的

它的值随时指向当前正在使用的对象

this和定义在哪儿无关，只和调用时的对象有关

②调用构造函数创建对象

var obj=new 类型名/构造函数名(属性值列表);

其中new做了4件事：

I 创建一个空对象

II 调用构造函数，为空对象添加新属性和方法(此时的this指上一步刚创建的空对象)

III 设置当前对象的\_\_proto\_\_属性为构造函数的原型对象

IV 将新对象的地址保存到obj变量中

例如 function Student(sname,age){

this.sname=sname;

this.age=age;

this.intrSelf=function(){

alert("I'm "+this.sname+",I'm "+this.age);

}

}

var lilei=new Student("Li Lei",18);

var hmm=new Student("Han Meimei",19);

cnosole.log(lilei.sname);

cnosole.log(hmm.sname);

lilei.intrSelf(); //this→lilei

//this.sname→lilei.sname

3、如何访问对象中的属性和方法

比如 对象.属性 → 一个变量

对象.方法() → 一个函数

4、原型和原型链

(1)原型(prototype)：保存多个子对象共享的成员的对象

\*每个构造函数都有一个原型对象属性

属性引用了该构造函数对应的原型对象

\*由构造函数创建的每个对象中都有一个\_\_proto\_\_属性

属性指向构造函数的原型对象

在访问子对象的成员时，优先在成员本地找，如果找不到，再去构造函数的原型中查找

(2)原型链：由各级对象的\_\_proto\_\_逐级向上引用形成的多级继承关系

\*所有的对象都有一个\_\_proto\_\_属性，指向自己的父对象

(3)何时放在构造函数中：每个对象自有的成员

何时放在原型对象中：所有子对象共有的成员

(4)原型相关API

①判断对象的属性是自有，还是共有

自有属性：保存在对象本地的属性

共有属性：保存在公共原型对象中的属性

如何判断一个属性是否可用：2种

I in关键字："属性名" in 对象

如果"属性名"在当前对象的原型链中，返回true；如果在整条原型链上都没找到，返回false

比如console.log("intrSelf" in lilei);

比如if(!("indexOf" in Array.prototype)){...}

II 使用===undefined

如果等于undefined，说明不包含

可以简写为： !对象.成员名

比如if(Array.prototype.indexOf===undefined){...}

比如if(!Array.prototype.indexOf){...}

专门判断是否自有属性：

obj.hasOwnProperty("成员");

如果obj本地有指定"成员"，则返回true；否则返回false

比如 hmm.money=5;

console.log(lilei.hasOwnProperty("money"));

console.log(hmm.hasOwnProperty("money"));

第一个结果返回false；第二个结果返回true

判断共有属性：

不是自有的，还要存在于原型链上

if(!(obj.hasOwnProperty("成员"))&&"成员" in obj)

比如 console.log(!(hmm.hasOwnProperty("intrSelf"))&&"intrSelf" in hmm);

在实例当中，intrSelf是共有属性，返回true

②获得任意对象的\_\_proto\_\_属性(目的是获得父级对象)

var 父对象=Object.getPrototypeOf(对象);

③判断父对象是否处在子对象原型链的上级

父对象.isPrototypeOf(子对象);

只要父对象在子对象的原型链上，就返回true；否则返回false

④删除对象中的属性

delete 对象.成员;

\*只能删除【自有】的成员

特例 var a=100;

或window.a=100;

或window["a"]=100;

delete a;

console.log(a);

→前三行的定义方式是等效的，但是经过delete之后，var a输出结果还是100，其他两个报错，这就说明：var a不能被delete；其他两种是以对象写入的，可以被delete，所以输出的时候已经找不到a，就报错

(5)如何判断一个对象是不是数组：4种方法（见练习）

\*typeof无法区分数组和一般对象，不能用来识别数组

比如 var arr=[];

console.log(typeof arr); //输出object，判断失败

①Array.prototype.isPrototypeOf(obj);

返回true是数组，否则不是

比如console.log(Array.prototype.isPrototypeOf(arr));

②对象 instanceof 类型名

instanceof：判断一个对象是否由某个类型创建出来

比如console.log(arr instanceof Array);

\*instanceof会查找原型链上所有原型的构造函数

③Object.getPrototypeOf()

比如console.log(Object.getPrototypeOf(arr)==Array.prototype);

返回true是数组，否则不是

④Object.prototype.toString方法

重写：override，指如果子对象觉得父对象的成员不好用，则可自己定义同名的自有成员覆盖父对象的

call和apply：在调用方法的一瞬间，替换调用方法的对象

Object.prototype.toString.call(arr);

Object.prototype.toString.apply(arr);

此种情况下两种等效

call vs apply：差在第二个参数上！

Math.max.apply(Math,[1,2,3]);

apply第二个参数要求将所有要传入的参数放在一个数组中

\*可获得数组最大值

Math.max.call(Math,1,2,3);

call第二个参数要求将所有的参数，必须一个一个独立传入

\*不可获得数组最大值，不能用于数组

⑤Array.isArray(obj)：专门判断对象是否是数组的函数

(此种写法比较新，笔试面试的时候不推荐，后期开发可以使用)

5、继承

(1)面向对象3大特点：封装、继承、多态

封装：将相关的属性和方法，集中定义在一个对象中

多态：js仅变相支持重载

继承：父对象中定义的成员，子对象不用定义，可直接访问

(2)3种继承方式

①一个对象，直接继承另一个对象

Object.setPrototypeOf(子对象,父对象);

等效于：子对象.\_\_proto\_\_=父对象

何时使用：只修改一个对象的父对象时使用

②修改构造函数的原型对象，统一修改之后将要创建的所有子对象的公共原型

构造函数.prototype=父对象

\*在创建所有子对象之前设置

\*一旦对象创建后，修改构造函数的原型，子对象无法继承新父对象

何时使用：批量修改未来所有子对象的父对象时

③仅继承结构(类似Java中的extends)

何时使用：两种【类型间】形成继承关系，并不一定要求两种类型的实例对象间有继承

如何继承结构：2步

I 子类型构造函数开始位置，先调用父类型构造函数

II 设置子类型的原型对象，继承父类型的原型对象

(3)对象的克隆

在Object的原型对象中添加clone方法

①constructor属性

只有构造函数的原型对象才有！

从构造函数的原型对象指回构造函数对象

②如何遍历对象中所有属性

for(var key in obj){

obj[key];

}

③如何克隆原始类型的包装对象

如何取出包装类型对象中的值：

obj.valueOf()

valueOf 和 toString的差别：

返回值类型的差别：

I toString一定是将所有内容转为字符串

II valueOf会取出对象中内部的值，而不进行类型转换

用途的差别：

I valueOf专用于算术计算和关系运算

II toString专用于输出字符串

在后台有如下这样的区别：

String=function(){ Number=function(){

if(this==null){ if(this==null){

return "null"; return 0;

}else{ }else{

return this.toString(); return this.valueOf();

} }

} }

valueOf 和 toString共同的缺点：

无法获取null和undefined的值