Day01

1. 什么是 js：

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

****

JavaScript：专门设计网页交互的语言

交互：用户输入数据-->程序接收并处理数据-返回处理结果

js 发展历程：

1995 Netscape: Navigator

LiveScript:

Sun Java: "write once run anywhere"

JavaScript: "像极了 Java，一定要比 Java 简单"

1996 JavaScript --> ECMA -->***ECMAScript***

ECMAScript：ES，ECMA 制定的 JavaScript 语言标准

规定了 js 语言的*核心语法*

JavaScript：NetScape 按照 ES 标准的规定，

实现的可使用的编程语言。

Microsoft: IE3.0,按照 ES 标准制定了自己的客户端脚本语言，JScript。

1998：W3C：DOM 标准-->专门操作网页内容的标准

随后微软和网景都宣称支持 DOM 标准

2004：Netscape-->Mozilla-->Firefox

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，没标准)

2. 如何编写 js：

1. js 如何运行的：2 种：

1. 浏览器中的 js 引擎(解释器)

浏览器中包含 2 个引擎：

1. 内容排版引擎——解析 HTML 和 CSS
2. 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——了解

可以在服务器端执行 js 脚本程序的软件

2. 如何编写：4 个位置可编写 js：

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!");

在控制台中，输出一行文字

使用 控制台 的 显示日志功能

console . log (要显示的文字)

"xxx": 字符串，程序中用于显示的一个或一串文字

js 中，如果一串文字用于显示，必须加引号

但，单双引号都行

操作：

执行程序:enter

输入多行程序：shift+enter

清屏：左上角 clear 图标

切换前/后一段已经执行的代码：上下箭头

使用 当前浏览器窗口 的 警告功能，弹出警告框

[window .] alert ("Hello World")

使用 当前网页文档 的 写入功能, 在网页中写入

document . write("Hello World")

如果字符串中出现和程序冲突的特殊符号，比如:" '

解决：2 种：1. \" 2."改为'

2. 随网页一起编写 js：2 处

1. <script>元素中：随网页解释过程而解释执行

2. 任意元素的事件属性中：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。

定义元素的事件：在开始标签中

<button on 事件名="js 脚本"

3. 外部 js 文件中编写：2 步：同 css

1. 创建.js 文件

2. 在 html 文件开头引入.js 文件

<script src="1_firstJS.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：希望的效果没反应-->F12 开控制台

控制台中：错误信息：3 部分：

错误类型：错误信息 ***出错位置的链接***

1. ***变量：

什么是变量：内存中存储一个数据的空间，再起一个名字

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

为什么使用变量：变量可以让程序更灵活

如何使用：3 件事：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; //默认值都是 undefined

变量命名：见文知义

驼峰命名：第一个单词首字母小写

之后每隔单词首字母都大写

规定：字母，数字，下划线组成

***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

强烈建议：所有变量在使用前，必须声明

2. 赋值：将等号右侧的值，保存到等号左侧的变量中

变量名=值;

对变量赋值，相当于用新值替换旧值

强调：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

var 变量名=值; //初始化:首次给变量赋值

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

特殊情况：1. 仅声明，未赋值，可以使用

*2. 未声明的变量，试图取值，报错：ReferenceError: XXX 未定义

3. 未声明，直接赋值，自动创建同名变量

但，强烈建议所有变量都要用 var 声明

****声明提前：在正式执行脚本之前，将所有 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。

***赋值留在本地。

常量：一旦创建并初始化后，值不可改变的量

何时使用：保存程序中一成不变的值

比如：π　　g

如何声明并初始化：const 常量名=值;

习惯上：常量名全大写字母组成

2. ***数据类型：2 大类：

1. 原始类型：值直接保存在变量本地的数据类型

5 种：Number　String　Boolean　undefined null

2. 引用类型：值不直接保存在变量本地的数据类型

1. Number：js 中专门存储数字的类型，

即可存整数，也可存浮点数(小数)

何时使用：计算，作比较

所占空间：整数：32Bit-->4bytes

浮点数：64Bit-->8bytes

1byte 字节=8Bit 位

内存中数字的大小和所占空间无关

***舍入误差：因为计算机中也有无法除尽的数字，

导致计算的误差　　舍入误差无法避免。

近似解决：按指定位数四舍五入：n.toFixed(2);

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，都存为字符串

所占空间：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

比如："Hello 马里奥":? 字节

111112 2 2　　11

***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！

/*鄙视题*/

```
var str="Hello";
str=str+"World";  共创建了几个字符串？
var a="a1";
var b="b1";
var c="c1";
var d=a+b+c; 共创建过 5 个字符串？
```

3. Boolean: 布尔类型：

只有两个值的数据类型：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，自动赋值使用

5. null：

```
<script>
    /*var hanbao; //undefined*/
    /*window.*/hanbao="香辣鸡腿堡";
    var drink; //undefined
    drink="可口可乐";
    console.log("15 元豪华套餐:");
    console.log(hanbao);
    console.log(drink);
    var bj=0;
    var xj=0;
```

/*鄙视题

```
    var bj,xj=0;*/
    //bj=undefined  xj=0
    console.log("北京 "+bj+" : "+xj+" 新疆");
                //北京　　0　:　0　　新疆
    //马布里,投进 3 分
    bj=bj+3;
    console.log("北京 "+bj+" : "+xj+" 新疆");
    //李根,投进 3 分
    xj=xj+3;
    console.log("北京 "+bj+" : "+xj+" 新疆");
```

/*鄙视题*/

```
    console.log(n); //undefined
    var n=100; //赋值
    console.log(n); //100
```

/*常量*/

```
    const PI=3.14;
    /*360 角度=2*PI 弧度*/
    var r30=PI/6; //30 角度是？弧度
    console.log(r30); //0.523333333333
    PI=1.14; //?出错
    console.log(r30); //0.523333333333
</script>
```

Day02

1. ***数据类型转换：2. ***运算符和表达式：

1. ***数据类型转换：

***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，先后可保存不同类型的值

3. ***js 在需要时，自动转换数据的类型

2 大类转换方式：隐式转换和强制转换

1. 隐式转换：不需要程序员干预，js 自动完成的类型转换

隐式转换无处不在，只要 js 认为有必要。

仅讨论算数计算中的隐式转换：

1. 算数计算中，一切数据都默认转为数字，再计算

Boolean 类型：true-->1　false-->0

特殊情况：如果数据无法被转为数字类型

会被转为 NaN：Not a Number

表示 不是一个数字 的 数字

(含义)　（类型）

NaN 不等于，不大于，不小于任何值

NaN 参与任何算数运算结果永远为 NaN

2. 在+运算中，碰到字符串，+就变为字符串拼接

另一个不是字符串的数据，被自动转为字符串

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

配置 EditPlus 的 ctrl+B: 快速用指定浏览器打开当前网页

开始菜单-->Google Chrome-->快捷方式点右键-->属性

复制目标地址

EditPlus-->tools-->preference-->tools

右侧下拉列表选：other browser

将地址复制到下方文本框->点 apply

2. 强制转换：程序员主动调用专门的函数，执行转换

何时使用：自动转换的类型，不是想要的

任意-->字符串：2 个 API

x.toString(); x 不是 undefined 或 null 时，才可用

String(x) --> 相当于隐式转换-->万能

typeof(x)-->返回 x 的数据类型名称

任意-->数字：3 个 API

Number(x): 专用于将非字符串类型转数字

相当于隐式转换

将字符串转为数字：2 个 API

parseInt(str): 从第一个字符开始

整数　依次读取每个数字

只要碰上第一个非数字字符就停止

自动跳过开头的空字符

***不认识小数点

何时使用：将字符串转为整数时

parseFloat(str):用法同 parseInt

浮点数　***认识第一个小数点

何时使用：将字符串转为小数时

Number(x) vs parseInt(str)

```
var b=true;
console.log(Number(b)) //1
console.log(parseInt(String(b))) //NaN
                "true"
var width="60px";
console.log(Number(width)) //NaN
console.log(parseInt(width)) //60
```

任意类型-->Boolean: Boolean(x)-->相当于隐式转换

只有 5 个值转为 false：

""　NaN undefined null 0

其余都转为 true！

快速转换：任意转 String: x+"";

任意转 Boolean: !!x

课堂练习：如何请用户输入一个数据

var input=[window.]prompt("提示信息")

弹出一个"输入框",程序暂停，等待用户输入

用户输入数据，点确定

程序将输入的数据保存到 input 变量中

程序继续向下执行

2. ***运算符和表达式：

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，数据和运算符组成的公式

所有表达式都有一个结果-->都可以当一个值用

1. 算数运算：+ - * /　%　++ --

%取余数(模运算)：

比如：n%m n/m,不取商，而取除不尽的余数

何时使用：2 大用途：

1. 判断能否整除：比如判断奇数，偶数

n%2 等于 0，可被 2 整除，——偶数

n%2 不等 0，不可被 2 整除，——奇数

2. 保证运算结果绝不超过除数

++递增　--递减

2 种：都会将变量 n 中的值+1，再存会 n 中

++n：表达式返回 n 的*新*值

n++：表达式返回 n 的*旧*值

1. ***运算符和表达式

关系运算：

***逻辑运算：

位运算：

*扩展赋值运算：

***三目运算：

1. 关系运算：将两个值做比较-->返回值：true、false

> < >= <= == !=

***关系运算中：

1. 默认将所有类型转为数字再比较
2. 如果参与比较的两值*都是字符串*

依此取出每个字符串的字符，PK unicode

比如：var name1="scott";

var name2="john";

console.log(name1>name2);

何时使用：只要判断条件，条件一般都为关系运算

2 个特殊情况：

1. NaN：NaN 不等于，不大于，不小于任何值（包括自己）

NaN 和任何数据做!=比较时，始终返回 true

问题：无法判断一个数字是否为数字

解决：isNaN(n):本意是判断一个值是否是 NaN

n 是 NaN，返回 true；否则返回 false

如何判断一个值是否为数字：

!isNaN(n)-->!true-->false-->n 不是数字

!false-->true-->n 是数字

2. undefined vs 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

undefined 和 null 其实是同一个东西

全等：=== 首先类型相同，同时值相等

不带隐式转换的等于比较

何时使用：区分 undefined 和 null 时

只要要求类型相同，且值相等时

2. 逻辑运算：将多个关系运算综合起来，得出最终结论

返回值:true、false

何时使用：程序中，只要判断条件超过 1 个时

3 种：&& || !

且 或 非

&&：两个条件必须都为 true，才返回 true

只要有一个为 false，则返回 false

||：两个条件只要有一个为 true，就返回 true

除非所有条件都为 false，才返回 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，后续条件，不再执行！

如何利用短路逻辑：

1. 1 件事，满足条件就做，不满足条件，就不错！

条件&&(操作);

何时使用：如果操作仅有一两句话时

2. 如果前一个值有效，就用前一个，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。（"" null undefined NaN 0）

3. 位运算：左移<< 右移>>

n<<m: 将 n 的二进制数所有位，左移 m 位

n* 2 的 m 次方

n>>m: n/ 2 的 m 次方

4. 扩展赋值运算：+= -= *= /= %=

n+=m: n=n+m 将 n 中的值增加 m，再存回 n 中

n*=m: n=n*m

何时使用：今后，只要直接累加，累减，累乘原变量时

5. ***三目运算：三元运算，条件运算

什么是：根据不同的条件，执行不同的操作/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: //否则，如果满足条件 2，就返回值 2 或执行操作 2

...?.....:

默认值或操作; //否则，（前边条件都不满足）

//执行默认操作，或返回默认值

总结：多个条件，多件事，多选一执行。

js：如果操作语句比较简单，建议用三目运算替代 if..else

案例**********

```
<script>
    var n1=2,n2=3,s1="2",s2="3",b1=true,b2=false;
console.log(n1+n2); //5          console.log(n1+s2); //"23"
          //2   3            //2+"3" //String(2)+"3"//"2"+"3"
console.log(s2-s1); //1          console.log(s1-n2); //-1
          //3-2                          //2-3
console.log(n1+n2+s2); //"53"
          //2   3
          //    5 +"3"
          //    "5"+"3"
    console.log(n1+s2+n2); //"233"
                //"2" "3"
                  // "23"+"3"
    console.log(s2-b1); //2
                //"3"-true
                 // 3 -1
    console.log(s2+b1); //"3true"
                //"3"+true          //    "true"
    console.log(Number(s1)*Number(b2)); //0
     //Number("3")*Number(false)       // 3 *0
    var str; //undefined
    console.log(s1+str); //"2undefined"
                //"2"+"undefined"
    console.log(s2-str); //NaN
                 //"2"-undefined        // 2 - NaN
    console.log(n1*str); //NaN
                // 2 *undefined          //       NaN
    </script>
```

```
 <script>
    var n=100;
```

```
console.log(typeof(n)); //number
console.log(typeof(n.toString())); //string
                    //100.toString()          //"100"
                   //typeof("100")
var m;
console.log(String(m));
var width="60px";
              //60.5px
                //60.5.5px
width=parseInt(width)+10+"px";
      //parseFloat(width)...
       //Number(width)...
console.log(width); //?
```

/*课堂练习*/

```
//Step1: 请用户输入工资：
var salary=parseFloat(prompt("请输入工资"));
console.log(typeof(salary)); //?
//Step2: 将工资加 1000 块钱
salary=salary+1000;
//Step3: 弹出警告框，显示加薪后的 salary
alert(salary);
var my=10;
var lp=//++my; //返回新值
          my++; //返回旧值
console.log(lp); //11
```

/*鄙视题*/

```
var n=3;
console.log(n++ + ++n +n++); //13
                // 3
                 //n=4
                      // 5
                        //n=5
                              // 5
                               //n=6
console.log(++n + n++ + ++n); //14
</script>
```

```
<script>
    var my=10;
    var lp=my++;
        //++my;
    console.log(lp);//10
    console.log(my); //11
```

/*鄙视题*/

```
    var n=2;
    var r=n++ + ++n + n++;        //++n + n++ + ++n;
        // 2   //n=3
              // 4   //n=4
                    // 4   //n=5
    console.log(r); //10
    console.log(n); //5
</script>
```

```
<script>
    var n1=2,n2=3,      s1="2",s2="3",     b1=true,b2=false;
    console.log(n1>n2); //false
    console.log(Number(s1)<n2); //2   < 3   => true
    console.log(Number(s1)>=n1); //2     >=2       //true
    console.log(Number(s1)>Number(b1)); //2   1//true
    console.log(Number(s1)==n1); //2     2       //true
    console.log(NaN>=NaN); //false
    console.log(NaN<=NaN); //false
    console.log(NaN==NaN); //false
    console.log(NaN!=NaN); //true
    /*var input=prompt("请输入一个值");
    //input 是数字，输出 true，否则输出 false
    console.log(!isNaN(input));*/
    var n; //undefined
    var m=null;
    console.log(n===null); //false
    console.log(m===null); //true
    console.log(typeof(n)); //undefined
    console.log(typeof(null)); //object
    var name1="scott";
    var name2="john";
    console.log(name1>name2); //true
                //"s"     "j"          //115 号>106 号
    var name3="smith";
    console.log(name1<name3); //true
                //"s"     "s"       //"c"     "m"       //99 号  <109 号
    var name4="scot";
    console.log(name1>name4); //true
    console.log("123"<"23"); //true/false
                //"1"     "2"              </script>
```

```
<script>
    var total=600;
    //如果 total>=500，就打八折，（不到 500，什么也不做）
    total>=500&&(total=total*0.8);
    console.log(total); //480
```

/*鄙视题*/

```
    //如果用户输入了回复，就显示用户输入的内容
    //否则显示"主人很懒，什么也没留下"
    var reply=prompt("请输入回复：");
    console.log(reply||"主人很懒，什么也没留下");     </script>
```

```
<script>
    /*请用户输入一个数字，用程序判断是奇数还是偶数*/
    //Step1: 请用户输入一个数字，保存在 num 中
    var num=prompt("请输入一个数字");
    //Step2: 判断是否是奇数，将判断结果保存在 isOdd
```

```
    var isOdd=num%2==1;
    //Step3: 判断是否是偶数，将判断结果保存在 isEven
    var isEven=num%2!=1;
    //Step4: 输出
    console.log("是奇数吗？"+isOdd);
    console.log("是偶数吗？"+isEven);
    /*判断用户输入是否为数字*/
    var input=parseInt(prompt("输入一个数据"));
    console.log("是数字吗？"+!isNaN(input));
  </script>
```

```
<script>
    /*姚*/
    //var H=226,D=2.5,B=false;
                //D>10 ||B==true
                  //false    false     -->false
    /*马*/
    //var H=166,D=218,B=false;
                //D>10
                  //true        -->true       /*短路逻辑*/
    /*贝*/
    var H=183,D=5,B=true;
              //D>10 ||B==true
                //false     true     -->true
    /*一：身高>180 而且 资产>1 而且 长的帅气*/
    console.log(H>180&&D>1&&B==true);
    /*二：资产>10 或者 长的帅气*/
    console.log(D>10||B==true);

# /*鄙视题*/

    var n=10;            var m=10;
    var r=n++>10 && ++m>10;
        //n++>10 || ++m>10;
    console.log(r); //
    console.log(n); //
    console.log(m); //
    /*请用户输入总价 total
        如果 total>=500，就打八折
        输出应收金额*/
    var total=parseFloat(prompt("输入总价"));
    //如果 total>=500，就打八折
    total>=500&&(total=total*0.8);
    console.log("应收金额："+total);
    /*请用户输入一个回复，如果用户未输入，则输出“主人很懒什么也没有留下”*/
    var input=prompt("请输入回复：");
    input!=""||(input="主人很懒什么也没有留下");
    console.log("回复："+input);          </script>
```

```
<script>
    /*请用户输入一个年份，判断是否为润年：
        条件：年份可被 4 整除且不能被 100 整除
                或者        年份可被 400 整除*/
    //Step1:请用户输入一个年份，保存在变量 year 中
    var year=prompt("输入一个年份：");
    //Step2:如果(year%4==0&&year%100!=0)||(year%400==0)
        //就输出"闰年"//否则，输出"平年"
console.log(
    (year%4==0&&year%100!=0)||(year%400==0)?"闰年": "平年");
    /*请用户输入一个字符，程序判断输入的字符类别
        条件 1：input>="0"且 input<="9"，输出"数字"
        条件 2：input>="\u4e00"&&input<="\u9fa5"，输出"汉字"
条件 3：(input>="A"&&input<="Z")||(input>="a"&&input<="z")
                        输出"字母"   否则，输出"其它字符"*/
    //Step1: 请用户输入一个字符
    var input=prompt("请输入一个字符：");
    //Step2: 根据不同的条件,输出判断结果
    console.log(
     input>="0"&&input<="9"?"数字":
      input>="\u4e00"&&input<="\u9fa5"?"汉字":
      (input>="A"&&input<="Z")
       ||(input>="a"&&input<="z")?"字母":"其它字符");
    /*根据输入的工资，判断级别
        如果工资<8000，就输出"屌丝"
否则，如果工资>=20000，就输出"土豪"    否则,输出"一般人";*/
   var salary=prompt("输入工资");
    console.log(
        //如果工资>=20000，则输出土豪
            salary>=20000?"土豪":
        //如果工资<8000，则输出屌丝
            salary<8000?"屌丝":
                //否则，输出一般人
                    "一般人" );
    /*就算可莱托指数：
        klt=体重/(身高*身高)
        klt>25   胖
            <20   瘦
                正常*/
    //Step1: 请用户输入体重(kg),保存在 weight
    var weight=prompt("输入体重(kg)");
    //Step2: 请用户输入身高(m),保存在 height
    var height=prompt("输入身高(m)");
    //Step3: 计算可莱托指数，保存在 klt 中
    var klt=weight/(height*height);
    //Step4: 根据条件判断，输出结果
    console.log(
        klt>25?"胖":
        klt<20?"瘦":"正常");
    var total=600;
    total>=500&&(total*=0.8);
    console.log(total);
</script>
```

Day03

正课：

1. *函数

2. *分支结构

1. *函数：封装一项专门任务的步骤清单的代码段，起一个名字

程序中提供的一项服务的步骤说明

何时使用函数？

当一项任务需要反复执行，但又不希望重复编写时代码重用！

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]

}

***函数名(任务名): 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

***声明时，不执行！也不读取内部的代码!

如何调用函数执行：任何位置: 函数名([参数值列表]);

***调用时，才读取内部的代码，执行

参数变量：专门接收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，必须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

何时如何为参数变量赋值：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

返回值：函数的执行结果

何时需要返回值：只要函数需要有明确的执行结果时

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

***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(window)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：1. 参数变量

2. 在函数定义中 var 的变量

***仅在函数调用时，动态创建

调用时，如果局部有，就不用全局的！

***调用后，随局部作用域一同销毁

1. *函数：***声明提前 ***按值传递

2. 全局函数：——了解

3. *分支结构：

1. *函数： ***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*顶部，集中声明

赋值留在原地

***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！

2. 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

比如：parseInt/Float(str)，isNaN(n)

比如：alert() prompt()——BOM

1. 编码解码：

/s?word=%E5%BC%A0%E4%B8%9C

张 东

问题 1：url 中不允许出现多字节字符

解决：使用 encodeURI()，将多字节字符编码为 utf-8 格式的单字节字符

使用 decodeURI(), 将收到的编码后单字节字符，解码为多字节字符原文。

/s?word=http%3A%2F%2Ftmooc.cn

: / /

问题 2：url 中不允许出现 url 的保留字符，比如：: /

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

使用 decodeURIComponent(),解码

2. eval: 执行字符串格式的语句或表达式

"alert('Hello');"

3. js 中分母是 0，不报错！返回：infinity

+∞

isFinit(n): 判断 n 是否在有效数字范围内

3. *分支结构：

程序结构：3 种：

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

操作的复杂程度

循环结构：让程序反复执行同一代码段。

分支结构：3 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)

短路逻辑：条件&&(操作 1,操作 2...)

何时使用：操作非常简单时

if 结构：如果 满足*条件*, 就执行代码段

if(条件){

满足条件时，才能执行的代码段

}

2. 一个条件，两件事：二选一执行！

如果 满足*条件*，就执行操作 1，否则，执行操作 2

三目运算：条件?操作 1:操作 2;

何时使用：操作 1 和操作 2，都非常简单时

if...else 结构：

```
if(条件){
满足条件才执行的代码段
}else{//否则
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！(有可能都不执行)

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]

三目：条件 1?操作 1:
条件 2?操作 2:
...:
默认操作——不可省略

if...else if 结构：

```
if(条件 1){
满足条件 1 才执行的操作 1;
}else if(条件 2){
满足条件 2 才执行的操作 2;
}else if(...){
...
}[else{
如果以上任何条件都不满足，则执行默认操作
}]
```

switch...case 结构:

何时使用：当条件都是*全等*比较时,才可用 switch 结构

```
switch(表达式){ //1. 计算表达式的结果
//用表达式的值和每个 case 后的值做**全等**比较
//碰到一个全等的 case 值，则进入该 case 开始执行
//并默认以此触发之后所有 case 的执行
case 值 1:
满足值 1 才执行的代码段 1;
case 值 2:
满足值 2 才执行的代码段 2;
...:
... ...
default:
如果前边的值都不满足，执行默认代码段
}
```

break: *中止*当前结构的执行，并跳出结构。

位置：在每个 case 之间

何时可以省略部分 break：上下两个 case 希望执行相同代码时

案例

```
<script>
/*<=10 10%
<=10 10% + 10~20 7.5%
<=10 10% + 10~20 7.5% + 20~40 5%
<=10 10% + 10~20 7.5% + 20~40 5% + 40~60 3%
<=10 10% + 10~20 7.5% + 20~40 5% + 40~60 3% + 60~100 1.5%
<=10 10% + 10~20 7.5% + 20~40 5% + 40~60 3% + 60~100 1.5%
                                        + >100    1%*/
//75 万
//10000+7500+10000+6000+2250
//35750
var sales=prompt("输入利润");
    var bonus=0;
//如果 sales 超过 1000000，则用(sales-1000000)*1%累加到 bonus 中，同时将 sales 设置为 1000000
sales>1000000&&(bonus+=(sales-1000000)*0.01,sales=1000000);
sales>600000&&(bonus+=(sales-600000)*0.015,sales=600000);
sales>400000&&(bonus+=(sales-400000)*0.03,sales=400000);
sales>200000&&(bonus+=(sales-200000)*0.05,sales=200000);
sales>100000&&(bonus+=(sales-100000)*0.075,sales=100000);
sales>0&&(bonus+=sales*0.1);
console.log(bonus);
    /*判断指定日期是一年中第几天*/
    //Step1:输入月份
    var y=parseInt(prompt("输入年份"));
    var m=parseInt(prompt("输入月份"));
    var d=parseInt(prompt("输入日期"));
    var days=0;
    //Step2:定义每个月的天数
    var m1=m3=m5=m7=m8=m10=m12=31;
                    //m10=31
                   //m8=31
                        //赋值运算的结果是等号右边的值
    var m4=m6=m9=m11=30;
    var m2=(y%4==0&&y%100!=0)||(y%400==0)?29:28;
    //Step3:
    m>11&&(days+=m11);
    m>10&&(days+=m10);
    m>9&&(days+=m9);
    m>8&&(days+=m8);
    m>7&&(days+=m7);
    m>6&&(days+=m6);
    m>5&&(days+=m5);
    m>4&&(days+=m4);
    m>3&&(days+=m3);
    m>2&&(days+=m2);
    m>1&&(days+=m1);
    days+=d;
    console.log(days);*/
    /*输入三个整数：x y z
```

```
        按照从大到小的顺序输出
        ***不允许声明新变量****/
    var x=prompt("输入第一个值");
    var y=prompt("输入第一个值");
    var z=prompt("输入第一个值");
    //Step1: 如果 x<y, 就交换 x 和 y 中的值
    x<y&&(x+=y,y=x-y,x-=y);
    //Step2: 如果 y<z, 就交换 y 和 z 中的值
    y<z&&(y+=z,z=y-z,y-=z);
    //Step3: 如果 x<y, 就交换 x 和 y 中的值
    x<y&&(x+=y,y=x-y,x-=y);
    console.log(x+","+y+","+z);
</script>
```

```
<script>
    /*定义一个专门的"打饭"服务——函数
        Step1: 取饭盒
        Step2: 去食堂
        Step3: 找大师傅打饭
        Step4: 回宿舍*/
    function buy(what,money){
        console.log("Step1: 取饭盒");
        console.log("Step2: 去食堂");
        console.log("Step3: 找大师傅打 "+what);
        money-=3;
        console.log("Step4: 扣款 3 元，余额："+money);
        console.log("Step5: 回宿舍");
        return "香喷喷的"+what;}
    console.log(buy("宫保鸡丁",10)); //调用一次
    var eric=buy("驴肉火烧",7); //调用两次
    console.log(eric);
</script>
```

```
<script>
    var kl=10;
    function rose(kl){
        kl--;
        console.log(kl);
    }
    function jack(){
        kl--;
        console.log(kl);
    }
    console.log(kl); //全局 kl=10
    rose(5); //4
    console.log(kl); //全局 kl=10
    jack(); //9
    console.log(kl); //9
</script>
```

```
<script>
    /*鄙视题——声明提前
        先将程序修改为声明提前之后的样子
    function fun(){
        return 1;
    }
    console.log(fun()); //2
    function fun(){
        return 2;
    }
    console.log(fun()); //2
    var fun=100;
    console.log(fun()); //出错*/
    /*鄙视题：*/
    var n=100;
    function fun(){
        var n; //undefined
        console.log(n);
        n=99;
        console.log(n);}
    fun(); //undefined  99
    console.log(n); //100
</script>
```

```
<script>
    var n=100;
    var m=n;
    m++;
    console.log(n); //100
    function buy(what,card){
        card-=3; //从饭卡里扣 3 块
        console.log("余额:"+card); //?
        return "香喷喷的"+what;}
    var card=10;
    buy("烩面",card); //7
    console.log("余额:"+card); //10
</script>
```

```
<script>
    var url="http://www.baidu.com/s?word=";
    var kword=encodeURIComponent(prompt("百度一下："));
    alert("向服务器发送:"+url+kword);
    //假设从服务器收到
    var response="%E5%BD%AD%E6%88%90";
    alert("从服务器收到:"+decodeURIComponent(response));
    /*eval*/
    eval("alert('Hello');");
    //简易计算器
    //Step1: 请用户输入算数表达式
    var stmt=prompt("输入算式");
        //"1+2-3"
    //Step2: 将算数表达式放入 eval 中
    var r=eval(stmt);
    //Step3: 输出 eval 的计算结果
    console.log(r); //0
    /*除数为 0 和 isFinite()*/
```

```
    console.log(isFinite(4/2)); //true
    console.log(4/0); //infinity
    console.log(isFinite(4/0)); //false
</script>
```

```
<script>
    /*var total=600;
    //如果 total>=500, 就 total*=0.8
    //total>=500&&(total*=0.8);
    if(total>=500){
        total*=0.8;
    }
    console.log(total);*/
```

/*模拟收银台小程序：

```
请用户输入单价(price)，数量(count)，收款金额(money)
程序：计算总价(total)=单价*数量，
      如果总价>=500,则打八折
      如果 收款金额>=总价
        计算找零(change)=收款金额-总价
        输出："应收:"+总价+"; 找零:"+找零
      否则
        计算欠款(change)=总价-收款金额
        输出："应收:"+总价+"; 还差:"+找零*/
    var price=parseFloat(prompt("输入单价"));
    var count=parseFloat(prompt("输入数量"));
    var money=parseFloat(prompt("输入收款金额"));
    var total=price*count;
    //total>=500&&(total*=0.8);
    if(total>=500){
        total*=0.8;
    }
    //如果 收款金额>=总价
    if(money>=total){
        var change=money-total;
        console.log("应收:"+total+"; 找零:"+change);
    }else{//否则
        //计算欠款(change)=总价-收款金额
        //输出："应收:"+总价+"; 还差:"+找零
        var change=total-money;
        console.log("应收:"+total+"; 还差:"+change);}*/
```

/*按照输入的分数，显示等级:

```
      如果 分数<0 或分数>100，就输出"无效分数"
否则，如果 分数>=90, 就输出"A"
否则，如果 分数>=80, 就输出"B"
否则，如果 分数>=60, 就输出"C"
                否则, 输出"D"*/
    //var score=prompt("输入分数：");
    /*console.log(
        score<0||score>100?"无效分数":
                score>=90?"A":
                  score>=80?"B":
                    score>=60?"C":
                              "D");*/
//如果 分数<0 或分数>100，就输出"无效分数"
if(score<0||score>100){
    console.log("无效分数");
}else if(score>=90){
//否则，如果 分数>=90, 就输出"A"
    console.log("A");
}else if(score>=80){
//否则，如果 分数>=80, 就输出"B"
    console.log("B");
}else if(score>=60){
//否则，如果 分数>=60, 就输出"C"
    console.log("C");
}else{//否则, 输出"D"
    console.log("D");
}
```

/*模拟电话银行程序：

```
    请用户输入一个数字：
        如果用户输入 1：执行查询操作
        如果用户输入 2：执行取款操作
        如果用户输入 3：执行转账操作
        如果用户输入 0：退出电话银行
                否则: 提示无效按键*/
var input=prompt("请按键:");
switch(parseInt(prompt("请按键:"))){
    case 1: //如果输入的值等于 1
        console.log("查询进行中...");
        break;
    case 2:
    case 3:
        console.log("系统维护中...");
        break;
    case 0:
        console.log("欢迎下次再来！");
        break;
    default:
        console.log("无效按键");
} </script>
```

Day04

1. ***循环结构：

While　　do while　　for　　循环嵌套

1. ***循环结构：

循环：让程序*反复*执行同一段代码

***三要素:

1. 循环条件：让循环*继续*执行的条件　　比如：圈数<3

2. 循环变量：循环条件中用于比较的变量　　比如：圈数

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

3. 循环体：循环内，反复执行的代码段

3 种：

1. while：先判断循环条件，再执行循环体

何时使用：循环变量的变化，没有规律时

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;
}
```

2. do while: 先执行循环体，再判断循环条件

何时使用：即使不满足循环条件，也希望至少执行一次时。

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);
```

while vs do while: 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，while 一次都不执行

do while 至少可以执行一次

1. ***循环结构：for - 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 的特殊用法：

1. 声明部分：可同时声明并初始化多个变量,用逗号

2. 迭代修改循环变量部分：可同时执行多个小操作

用逗号分隔

3. for(;;)-->取代 while(true)，实现死循环效果

continue: 中止本轮循环，继续下一轮循环——了解

/*猜数字:

程序内，使用变量保存一个数　　请用户反复输入一个数字

如果用户输入"exit"或猜对，就停止游戏

如果用户输入=="exit"，提示放弃了

如果用户输入>保存的数，则提示大了

如果用户输入<保存的数，则提示小了

如果用户输入==保存的数，提示猜对了

三要素：

1. 循环条件：用户输入!="exit"，且不等于保存的数

2. 循环变量：用户输入(input)

3. 循环体：根据用户的输入，输出不同的提示*/

```
//任意整数间(min~max)取 1 个随机数:
//      parseInt(Math.random()*(max-min+1)+min)
var n=parseInt(Math.random()*(9-0+1)+0);
var input="";
while(true){//不再使用循环条件控制退出
    //在循环体内部使用 break 控制退出
    //请用户反复输入一位数字
    input=prompt("你猜");
    if(input=="exit"){//如果用户输入=="exit"
        alert("放弃了");//就输出放弃了,
        break;//退出循环;
    }else if(input<"0"||input>"9"){
        //否则,如果用户输入<"0"或用户输入>"9")
        alert("玩儿我");//提示你玩儿我
    }else if(input>n){//否则,如果用户输入>保存的数,
        alert("大了");//则提示大了
    }else if(input<n){//否则,如果用户输入<保存的数,
        alert("小了");//则提示小了
    }else if(input==n){//否则,如果用户输入==保存的数
        alert("猜对了");//提示猜对了
        break;//退出循环
    }
}
```

/*计算从 1 加到 100 的和

三要素：

1. 循环条件: 加数(i)<=100

2. 循环变量: 加数(i)

从 1 开始，每次+1

3. 循环体：将当前的加数累加到和(sum)中*/

```
for(var i=1,sum=0;i<=100;sum+=i++);
/*js 中没有块级作用域*/
console.log(sum); //5050
```

/*1/1+1/3+1/5+...+1/999

三要素：

1. 循环条件：分母<1000

2. 循环变量：分母

从 1 开始，每次+2

3. 循环体：将 1/当前分母的结果累加到 sum 中*/

```
for(var i=1,sum=0;i<1000;sum+=1/i,i+=2);
console.log(sum);
```

/*打印乘法口诀表第 r 行: r=5

"1x5=5 2x5=10 3x5=15 4x5=20 5x5=25"

三要素：

1. 循环条件：乘数(i)<=r

2. 循环变量：乘数(i)　从 1 开始，每次+1

3. 循环体：拼接并计算每个子算式

i+"x"+r+"="+i*r+" "　　将子算式拼接到 str 中*/

打印乘法口诀表：

三要素:

1.循环条件：r<=9

2. 循环变量：r　　从 1 开始，每次增 1

3. 循环体：

```
for(var i=1,str="";i<=r;str+=i+"x"+r+"="+i*r+" ",i++);
    console.log(str);*/
     for(var r=1;r<=9;r++){
for(var i=1,str="";i<=r;str+=i+"x"+r+"="+i*r+" ",i++);
    console.log(str);
    }
    var bao=6;
    for(var i=1;i<9;i++){
        if(i!=bao){
            console.log("跳到 "+i);
        }
    }
```

输入任意年份，输出该年开始，之后 5 个闰年

1. 循环条件：闰年个数(n)<5 个

2. 循环变量：闰年个数，year

(n)从 0 开始，每碰上一个闰年，才+1

year 从用户输入开始，每次+1

3. 循环体：依次遍历 year 开始后的每一年

判断当前年份是否是闰年

如果是闰年　　就输出　　闰年个数+1

```
var y=parseInt(prompt("输入年份:"));
for(var n=0;n<5;(n>0?y+=4:y++)){
    if(y%4==0&&y%100!=0||y%400==0){
        console.log(y);
        n++;    }
}
/*年利率 3%，本金 10000
    几年后，本金超过 12000
    1. 循环条件：本金<=12000
    2. 循环变量：本金，年数 y
        从 10000 开始，每年*=1.03
        从 0 开始，每次+1    */
/*for(var m=10000,y=0;m<=12000;m*=1.03,y++);
console.log(y+"年后超过 12000");*/
/*反复输入密码：直到输对为止
    请用户输入密码：input
    判断 input 是否等于 pwd
    如果不等于就反复输入
    1. 循环条件：input!=pwd
    2. 循环变量: input
    3. 循环体：提示"密码错误，请重试"
    (循环退出后，)提示"验证成功"*/
    var pwd="a1s2d3f4g5";
    var input="";
    while((input=prompt("输入密码:"))!=pwd){
        alert("密码错误，请重试");
    }
    alert("验证成功");
</script>
```

/*正三角*/

```
    //打印第 r 行
    /*反复打印 r 个星星
        1. 循环条件：个数(n)<3
        2. 循环变量：个数(n), 保存当前行所有星星的 str
            n 从 0 开始，每次+1
            str 从""开始，每次拼接一个*
        3. 循环体：向 str 中，拼接一个*
        (循环退出后，)统一打印一次 str*/
    var r=1;
    for(var n=0,str="";n<r;str+="*",n++);
    console.log(str);
    /*三要素：
        1. 循环条件：r<=l
        2. 循环变量: r
            r 从 1 开始，每次+1
        3. 循环体：*/
//for(var n=0,str="";n<r;str+="*",n++);
//console.log(str);
```

/*正三角*/

```
    var l=prompt("输入行数");
    for(var r=1;r<=l;r++){
for(var n=0,str="";n<r;str+="*",n++);
console.log(str);
    }
/*反三角：第 r 行
        共需要打印 l 个字符
        先需要 l-r 个空格
        其余都是*
        三要素：
        1. 循环条件：字符个数(n)<l
        2. 循环变量：字符个数(n)　从 0 开始，每次+1
3. 循环体：如果 n<l-r,就向 str 中，拼接" "　　否则就拼接"*"*/
    var l=6;
    /*三要素：
        1. 循环条件：r<=l
        2. 循环变量：r　　r 从 1 开始，每次+1
        3. 循环体:*/
    for(n=0,str="";n<l;n++){
        str+=(n<l-r?" ":"*"); }
    console.log(str);　　//_ _ * * * *　　</script>
```

Day05

1. ****数组：一组变量的集合，起一个统一的名字
连续保存多个数据的引用类型的对象
***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，可以极大提高程序的执行效率。
相关的一组数据，都要使用数组集中管理。
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=[]; var 数组名=new Array();
何时使用：暂时不知道数据内容，先创建空数组，再追加
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,...)
数组中每个元素<==>一个变量: 用法和变量完全相同！
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+1，到元素个数-1 结束
如何取值：var 元素的值=数组名[下标];
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;
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
后果，对方修改数组的内容，访问原数字名也会受影响
null: 主动释放一个对象的引用。null 本身是原始类型的值
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
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，较大对象后，主动赋值为 null，总是好习惯
4. 获取数组的元素个数：arr.length 属性，不加圆括号
固定套路：
获得数组最后一个元素：arr[arr.length-1];
向数组末尾追加一个新元素：arr[arr.length]=新值
js 中的数组 vs 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. js 的数组越界不会出错！
赋值时：自动创建制定下标的元素，保存新值
length 属性，会随最大下标而变化
取值时：视图从没有的下标位置取值，返回 undefined
5. 遍历数组：从第 0 个元素开始，依次获取每个元素，
对每个元素执行相同的操作
下标：遍历数组时，天生的循环变量！

1. ****数组：
关联数组:
数组的 API:
1. 关联数组(hash 数组)：可自定义元素下标名称的数组
索引数组：自动从 0 开始分配连续不重复的序号下标
杨幂 81 53 89
0　1　2　3
如何创建关联数组：2 种方式：
1. 先创建普通空数组，再追加自定义下标的元素
var ym=[];
ym["sname"]="杨幂";
ym["shx"]=81;　ym["yw"]=53;　ym["yy"]=89;
关联数组的 length 属性失效
关联数组中的 key 不能重复
关联(hash)数组优势：查找极快！和元素个数无关！
2. 创建数组同时，初始化元素内容：
var ym={"sname":"杨幂","shx":81,"yw":53,"yy":89};
key　value //键　值
关联数组中保存的是多个 key/value 对儿
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]-->获得当前正在遍历的 key 中存储的值 }
2. 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,提供对数据的操作方法
1. arr 2 String: 2 个
1. var str=arr.toString();　默认只能以逗号分隔每个元素
***2. var str=arr.join("连接符");　自定义元素间的连接符
固定套路：1. 将字符数组，无缝拼接为单词：
比如: var chars=["H","e","l","l","o"];
2. 将单词数组，拼接为句子：
3. 将数组转化为页面上的列表/选择元素：
***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
2. 连接数组和获取子数组：不修改原数组对象，返回新数组对象
concat: var newArr=arr.concat(值 1,值 2,[值 3,值 4].....)

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

starti: 开始的位置下标　　endi: 获取到的位置下表

含头不含尾

slice 支持倒数下标　　省略 endi，从 starti 一直取到结尾

3. splice：删除，插入，替换任意位置的任意个元素

***直接修改原数组

删除：splice(starti,n);

插入：splice(starti,0,新值 1,新值 2,...)

替换：splice(starti,n,新值 1,新值 2,...)

4. reverse(): 颠倒数组中所有元素的位置

***直接修改原数组

/*1,2,3,4 组成多少种 3 位数*/

要求：三个位的数不能相同

```
for(var b=1;b<=4;b++){
    for(var s=1;s<=4;s++){
        for(var g=1; g<=4; g++){
            if(b!=s&&s!=g&&b!=g){
                console.log(""+b+s+g);}
            }}}
```

/*打印国际象棋棋盘：

行号从 0 开始，到 7 结束

列号从 0 开始，到 7 结束

```
for(var r=0;r<8;r++){
    for(var c=0,str="";c<8;c++){
        if((r+c)%2==0){
            str+="□";
        }else{
            str+="■";
        }
    }console.log(str);
}
```

/*每对儿兔子，第三个月，下一对儿新兔子

开始时，只有一对儿

```
var m1=1; //保存 2 个月前的对儿数
var m2=1; //保存 1 个月前的对儿数
var month=prompt("输入月份：");
for(var m=3;m<=month;m++){
    m1+=m2; //任何月份都是将前两个月相加
m1+=m2;m2=m1-m2;m1-=m2; //将合并后的 m1 和 m2 交换
}console.log("第"+month+"月:"+m2*2);
```

/*等腰三角:

共 l 行

第 r 行：*共 l+r-1 个字符*,先 l-r 个空格*/

```
var l=parseInt(prompt("输入行数"));
for(var r=1;r<=l;r++){
    for(var i=0,str="";i<l+r-1;i++){
        str+=(i<l-r)?" ":"*";
    }console.log(str);
} </script>
```

```
<script>
    //var week=['日','一','二','三','四','五','六'];
            // 0    1    2    3    4    5    6
    /*var week=[];
    week[0]='日';
    week[1]='一';
    console.log(week.length); //2 个
    week[100]="六";
    console.log(week.length); //101 个
    console.log(week);
    console.log("今天星期"+week[5]);*/
    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
    console.log(m); //100
```

/*我 一屁包子*/

```
    var my=['包','包','包','包','包'];//引用类型的对象
    var lp=my; //my 将保存到数组地址值复制给了 lp
    lp[0]=""; //修改 lp，会影响 my？
    console.log(my); //4 个
    my[1]="";
    console.log(lp); //3 个
    lp=null; //lp 释放了对象的引用
```

/*请用户反复输入员工姓名，存入数组：

定义一个空数组：emps

反复请用户输入姓名:input,*同时*判断 input!="exit"

将 input 追加到数组末尾

(循环退出后)整体输出数组的内容

```
    */
    var emps=[];
    var input="";
    while((input=prompt("输入员工姓名"))!="exit"){
        emps[emps.length]=input;
    }
    console.log(emps);
```

/*定义一个函数，找出数组中的最大值*/

```
    function getMax(arr){
        //定义一个 max 变量，初始化为数组第一个元素的值
        //下标 i 从 1 位置开始，到<arr.length 结束,每次+1
        //    如果当前 i 位置的元素值>max
        //        将当前 i 位置的元素值保存到 max 中
        return max;
    }
    console.log(getMax([3,7,4,5,8,9,6]));
  </script>
```

```
 <script>
    /*定义一个函数，indexOf()
```

```
        接收一个数组对象，和一个指定的值
        返回指定的值在数组中的位置下标
        如果没找到，就返回-1
    */
    function indexOf(arr,elem){//elem 在 arr 中的下标
//遍历数组中每个元素，i 从 0 开始，到<arr.length 结束，每次+1
for(var i=0;i<arr.length;i++){
//    如果当前元素==elem
//        返回 i
    if(arr[i]==elem){
        return i;
    }
}return -1;//(退出循环后)返回 -1
    }
    var emps=["eric","scott","tom","jerry","rose"];
console.log(indexOf(emps,prompt("输入要查找的姓名")));
/*冒泡排序*/
    function bubbleSort(arr){
        //得到的是全局变量 arr 中的数组地址
        /*第 r 轮: r 从 1 开始，到<length 结束*/
        for(var r=1;r<arr.length;r++){
        //从 0 开始遍历数组，到<length-r 结束，每次+1
            for(var i=0;i<arr.length-r;i++){
            //    如果当前元素>下一个元素
                if(arr[i]>arr[i+1]){
    //            交换当前元素和下一个元素的位置
             arr[i]+=arr[i+1];      arr[i+1]=arr[i]-arr[i+1];
        arr[i]-=arr[i+1];      }} } }
    var arr=[3,7,4,5,8,9,6];
    bubbleSort(arr);//将数组的地址复制给参数变量 arr
    console.log(arr);          </script>
```

```
<script>
    var ym=[];
   ym["sname"]="杨幂";
   ym["shx"]=81;      ym["yw"]=53;      ym["yy"]=89;
    var ym={"sname":"杨幂","shx":81,"yw":53,"yy":89};
    console.log(ym);
    console.log(ym["shx"]);
    for(var key in ym){
        console.log(key+" : "+ym[key]);
    }
/*去掉数组中重复的元素,并统计原数组中每个值出现的次数*/
    var arr=[2,3,4,3,6,4,5,6];
    var hash=[];
    //从 0 开始，到 arr.length-1 结束,遍历 arr 中每个值
    for(var i=0;i<arr.length;i++){
    //    如果 hash 数组中，不包含当前元素对应的 key
        if(hash[arr[i]]===undefined){
    //        则将当前元素值作为 key，加入 hash 中，值默认为 1
            hash[arr[i]]=1;
        }else{//   否则
    //        将 hash 中对应 key 中的值+1
            hash[arr[i]]++;
        }
    }
    console.log(hash);
    arr=[];//清空 arr
    //依次将 hash 中每个 key，追加入 arr 中
    //for(arr[arr.length] in hash);//执行的什么操作
    for(var key in hash){
        arr[arr.length]=(key-=0);
    }
    console.log(arr); //[2,3,4,5,6]          </script>
```

```
<script>
    var chars=["H","e","l","l","o"];
    var word=chars.join("");
    console.log(word); //"Hello"
    var words=["we","two","who","and","who"];
    var str=words.join(" ");
    console.log(str);//we two who and who
    var cities=["北京","天津","石家庄","郑州","济南"];
    /*          北京</option> <option>天津</option> <option>石家庄
</option> <option>郑州</option> <option>济南*/
    var html=cities.join("</option><option>");
    html="<select><option>"+html+"</option></select>"
    document.write(html);
    /*连接*/
    var arr=[1,2,3];      var arr2=[11,22];
    var newArr=arr.concat(12,23,123,arr2);
    //1,2,3,12,23,123,11,22
    console.log(newArr);
    /*截取子数组*/
    110102198312262111
    012345678901234567
    876543210987654321-
    var pid=["1", "1", "0", "1", "0", "2", "1", "9", "8", "3", "1", "2", "2", "6",
"2", "1", "1", "1"];
    var birth=//pid.slice(6,14);      // pid.slice(6,-4);
    console.log(birth.join(""));
    console.log(pid.slice(-4));
    /*删除*/    pid.splice(6,8);      console.log(pid);
    /*插入*/    // 1101022111     // 0123456789
    pid.splice(6,0,1,9,8,3,1,2,2,6);      //console.log(pid);
    /*替换*/    // 110102198312262111
    // 012345678901234567
    pid.splice(6,8,'X','X','X','X','X','X','X','X');
    console.log(pid.join(""));
    /*颠倒数组*/        pid.reverse();        console.log(pid);
```

Day06

1. ***数组 API：

1. ***数组排序：arr.sort()　直接修改原数组

问题：默认 sort 方法，只能将所有元素转为字符串再排序

结果：无法对数字类型排序

解决：自定义比较规则：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：a,b

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;

数字比较器：

升序：function compare(a,b){return a-b};

降序：function compare(a,b){return b-a};

始终 a 是当前，b 是下一个，所以参数 a，b 顺序不能变

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！

arr.sort(compare);

1. ***数组 API：

*栈和队列：都是用数组模拟

栈：一端封闭，只能从另一端进出的数组

何时使用栈？仅从一端进出数组时

FILO：先进的后出

从数组末尾出入栈：其他元素的下标不受影响

末尾 入栈: arr.push(新值)

末尾 出栈：var last=arr.pop()

从数组开头出入栈：每出入栈一次，其余元素下标都要瞬移

开头 入栈：arr.unshift(新值);

开头 出栈：var first=arr.shift();

队列：从一端进入，从另一端出的数组

FIFO：先进的先出

***二维数组：数组中的某个元素，又指向另一个子数组对象

何时使用：2 种情况：

1. 存储横行竖列的二维数据时，比如：2048

2. 存储上下级包含关系的数据

如何创建：2 种：

1. 先创建普通空数组，再逐个元素追加子数组

2. 创建同时，初始化子数组

访问二维数组中指定位置的元素：arr[row][col]

*遍历二维数组：固定套路：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++){
        当前元素：arr[row][col]
    }
}
```

***二维数组行下标 row 不能越界：

arr[row][col]　　undefined[col]

2. ***String：字符串：多个字符组成的只读字符数组

只要数组中不直接修改原数组的 API，字符串都能用！

str[i]-->获得字符串中下标为 i 的 1 个字符

str.length-->获得字符个数

str.slice(starti[,endi+1])-->获得子数组

*内置对象：ES 标准中规定的，厂商已经实现的对象

11 个：

String Number Boolean——包装类型：

Array　RegExp　Math　Date

Error　Function　Object

Global--全局对象

浏览器中被换成了 window

*包装类型：专门封装原始类型的值，并提供对其操作的方法

3 个: String Number Boolean

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

为什么要有包装类型：原始类型的值没有任何方法！

都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！

生命周期：调用方法时自动创建　　方法调用完自动释放

String(x) vs new String(x)

String(x): 将任意类型转换为字符串类型，*返回值是原始类型值

new String(x): 创建包装类型的对象，*返回值是引用类型的对象

转义字符：将特殊意义的字符，转为普通文字

何时使用转义字符：当字符串中包含和语法冲突的特殊符号时

要用转义字符，将特殊符号转为普通文字

记 2 种：1. 功能：\n 换行　\t 一个 tab 键

2. 特殊符号：\"　\'　\\

String API: 所有 API 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的

比如：邮箱地址，用户名，验证码

2. 获取指定位置的字符：

```
var char=str.charAt(i);
```

str[i]

```
//获得 i 位置字符的 unicode-->数字
var unicode=str.charCodeAt(i);
```

3. 检索关键字：

```
//找右侧下一个匹配的关键词位置
var i=str.indexOf("关键词"[,fromi]);
//找左侧下一个匹配的关键词位置
var i=str.lastIndexOf("关键词"[,fromi]);
```

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

案例

/*乘法口诀表

```
for(var r=1;r<=9;r++){
    for(var i=1,str=[];i<=r;i++){
        str[str.length]=i+"x"+r+"="+i*r+" ";
    }console.log(str);    }
```

/*分解质因数*/

```
var n=parseInt(prompt("输入一个数")); //n 表示被除数
var result=n+"=";      var str=[];
//i 表示除数，反复除 i，直到除不尽，才+1，i<n
for(var i=2;i<n;i++){
    while(n%i==0){//只要可以被整除，就继续除 i
        n/=i; //将本轮的商，替换到被除数
        str[str.length]=i;
    }      }
i==n&&(str[str.length]=i);
//["2","3","3","5"];
console.log(result+=str.join("x"));
```

/*求最大公约数：a>=b

```
r=a%b;    a=b;    b=r    直到 b==0   最后的 a 就是最大公约数
var a=parseInt(prompt("输入第一个数")); //80
var b=parseInt(prompt("输入第二个数")); //90
a<b&&(a+=b,b=a-b,a-=b);
var ji=a*b; //如果求最小公倍数，必须先保存乘积。
//var r=0; //r 表示余数
while(b!=0){
    b=(a%(a=b)); //a?80    //   90      80
}console.log(a);
console.log(ji/a);
```

/*a+aa+aaa+...+aaaaa*/

```
var a=parseInt(prompt("输入一个 1-9 的数字")); //2
var n=parseInt(prompt("输入累加个数")); //5
    [a]   [aa]    [aaa]    [aaaa]    [aaaaa]*/
for(var arr=[],sum=0;arr.length<n; ){
    arr[arr.length]=a; //先向数组中压入 a
    sum+=parseInt(arr.join("")); //再做计算
}console.log(sum);      </script>
```

```
<script>
    var arr=[12,3,23,123,33,2,1];
    //Step1:分别定义升序和降序比较器函数对象
    //升序逻辑：只要 arr[i]-arr[i+1]>0，就交换
    //降序逻辑：只要 arr[i+1]-arr[i]>0，就交换
    function compareASC(a,b){return a-b;}
    function compareDESC(a,b){return b-a;}
   //sort 函数等待接收一个比较器函数对象，用于在 if 中比较任意两数大小
    function bubbleSort(arr,compare){
        for(var r=1;r<arr.length;r++){
            for(var i=0;i<arr.length-r;i++){
                if(compare(arr[i],arr[i+1])>0){
                    arr[i]+=arr[i+1];
                    arr[i+1]=arr[i]-arr[i+1];
                    arr[i]-=arr[i+1];}
            }    }          }
    //Step2:
    arr.sort(compareDESC);
    console.log(arr); //降序排列
    arr.sort(compareASC);
    console.log(arr); //升序排列
    arr.sort(compareDESC);
    console.log(arr);
    /*函数当做对象传递
    function card1(shicai){//任务卡*函数对象*
        console.log("加工"+shicai);}
    function card2(shicai){//任务卡*函数对象*
        console.log("买"+shicai);}
        function taskNuoyi(taskCard){//Nuoyi 的任务
        var shicai="西红柿和鸡蛋";
        taskCard(shicai);
        //nuoyi 按照任务卡的步骤，加工自己的食材
        console.log("完成西红柿炒鸡蛋");
    }
    function taskXiatian(taskCard){//Xiatian 的任务
        var shicai="肉和辣椒";
        taskCard(shicai);
        //xiatian 按照任务卡的步骤，加工自己的食材
        console.log("完成红烧肉");
    }
    taskNuoyi(card2); //Nuoyi 得到任务卡，开始执行任务
taskXiatian(card1);//xiatian 得到任务卡，开始执行任务*/ </script>
```

```
<script>
    var bus=[];
    for(var i=1;i<=5;i++){
        bus.unshift("乘客"+i);
        console.log(bus);}
    while(bus.length>0){
        var customer=bus.shift();
        console.log(customer+"下车");
        console.log(bus);}
    //Step1: 请用户输入一个整数
    var n=prompt("输入一个整数"); //75
    //定义一个空栈(数组)
    var stack=[];
    //反复用 n/2，直到 n>0
    while(n>0){
        //将 n/2 的余数，压入 stack 中
        stack.unshift(n%2);//开头入栈
```

```
        //将 n 设置为 n/2 的商,取整数部分
        n=parseInt(n/2);       }
    //stack.reverse();//翻转 stack
    //输出 stack 无缝拼接后的结果
    console.log(stack.join(""));
    //1001011*/
    var n=5;
    var queue=[];
    for(var i=1;i<=6;i++){
        queue.push("顾客"+i);
        console.log(queue);
    }
    while(n>0){
        var customer=queue.shift();
        n--;
        console.log(customer+"抢购成功，剩余 "+n);     }
    console.log(queue+"没抢上");
  </script>
```

```
<script>
    var provs=["北京市","天津市","山东省"];
    var cities=[["东城区","西城区","海淀区"],
                ["和平区","南开区","滨海新区"],
                ["济南市","德州市","青岛市"]];
    var input=prompt("输入省份名称");
   //遍历 provs 中的每个省份
    for(var i=0;i<provs.length;i++){
    //      如果当前省份==input
        if(provs[i]==input){
    //          输出 cities 数组中和当前省份位置相同的子数组
            console.log(cities[i]);
                break;// 退出循环;
        }else if(i==provs.length-1){//如果已经是最后一元素
    //      否则,如果 i==provs.length-1
    //          输出“省份不存在”
            console.log("省份不存在");
        }       }
    var data=[//0 1 2 3
              [2,4,2,4],//0
              [4,2,2,2],//1
              [2,4,2,4],//2
              [4,2,4,2] //3
             ];
    function isGameOver(){
    if(!isFull()){//如果不满
        return false;//返回 false
    }else{//否则      //      遍历 data 中每个元素
        for(var row=0;row<data.length;row++){
            for(var col=0;col<data[row].length;col++){
    //          如果不是最右侧列且当前元素的值==右侧元素值
                if(col!=data[row].length-1
                    &&data[row][col]==data[row][col+1]){
                    return false;//返回 false     }
    //          如果不是最后一行且当前元素的值==下方元素值
                if(row!=data.length-1
                    &&data[row][col]==data[row+1][col]){
                    return false;//返回 false
                }
            }
        }return true;          //(遍历结束)返回 true
            }      }
    console.log(isGameOver()); //true
    //检查 data 数组是否已满
    //      如果碰到 0，说明不满，否则，说明满了
    function isFull(){
        //遍历 data 中每个元素
        for(var row=0;row<data.length;row++){
            for(var col=0;col<data[row].length;col++){
        //      如果当前元素==0,
                if(data[row][col]==0){
            return false;// 退出函数同时返回 false
                }
            }
        }return true;       //(遍历结束)返回 true
            }console.log(isFull()); //false          </script>
```

```
<script>
    var pid="110102198312262111";
            //012345678901234567
    //获得字符串中倒数第 2 位字符
    var gender=pid[pid.length-2];
    console.log(gender%2==0?"女":"男");
    //获得生日
    //输出成"xxxx 年 xx 月 xx 日"
    console.log(pid.slice(6,10)+"年" +pid.slice(10,12)+"月"
                +pid.slice(12,14)+"日");
    /*验证码*/
    /*var code="eR7S"; //er7s
    var input="";
    //input=prompt("输入验证码:"+code);?
    //反复输入，直到 input==code,才退出
    while((input=prompt("输入验证码:"+code)).toUpperCase()
            !=code.toUpperCase()){   alert("验证码错误！ ");
    }alert("验证通过");*/ //(循环退出后)
    /*检索关键字*/
    var str="no zuo no die no can no bi bi";
            //012345678901234567890123456789012345678
    var i=str.indexOf("no"); //0              console.log(i);
    var i=str.lastIndexOf("no"); //21
    console.log(i);        </script>
```

Day07

1. ***String:

String API:

获取子字符串：3 种：

1. var subStr=str.slice(starti[,endi+1]);

2. var subStr=str.substring(starti[,endi+1]);

substring 不支持负数参数

变通: 使用 length-n,代替负数下标　都是含头不含尾

3. var subStr=str.substr(starti,n);

比如：var pid="xxx";

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

.substring(6,pid.length-4)

.substr(6,8);

分隔字符串：var strs=str.split("分隔符");

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

2. ***String 中支持正则的 API：

str.search()　str.match()　str.replace()　str.split()

1. ***正则表达式：规定字符串中字符出现规则的表达式

何时使用正则表达式：查找，验证，替换，切割

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

[备选字符列表]

比如：我去、我草、　--> 我[去草]

比如：我去去、我草草、我去草　--> 我[去草][去草]

2.1 如果备选字符的 unicode 号连续：

可用 - 连接开头字符和结尾字符

比如：1 位数字: [0123456789]-->[0-9]

1 位小写字母:[a-z]

1 位字母：[a-zA-Z]

1 位数字或字母: [a-zA-Z0-9]

3. 预定义字符集：针对常用的备选字符集提供的简化符号

比如：[0-9]-->\d : 1 位数字

[a-zA-Z0-9_]-->\w : 1 位字母，数字或下划线

空字符-->\s : 1 位空字符：换行,制表符,空格

. : 除换行外，其余所有字符！

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

6 个：2 组：

4.1： 确定数量：

{m,n}-->至少 m 个，最多 n 个　{m,}-->m 个以上

{m}-->必须 m 个

4.2： 不确定数量：

*-->可有可无，多了不限　+-->1 个以上

?-->可有可无，最多 1 个！

比如：我去,我草，我去去，我草草，我去草

我[去草][去草]?

5. 选择和分组：

选择： | 读作"或"

分组： ()

何时使用：希望多个字符集被同一个两次修饰时

手机号：+86 或 0086　可有可无，最多 1 个

空字符　不限制个数

1

3,4,5,7,8　选其一

9 位数字

(\+86|0086)?\s*1[34578]\d{9}

6. 指定匹配位置：

开头和结尾：^开头的 xxx；$结尾的 xxx

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：只要验证时，必须前加^,后加$！

其余：不要都加！

比如：匹配字符串开头的所有空字符：^\s+

匹配字符串结尾的所有空字符：\s+$

同时匹配开头或结尾的空字符：

单词边界：\b:一个单词必须前后有空字符分隔才匹配

密码强度验证：8 位密码，字母数字的组合，至少包含一位大写字母和一位数字

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]+$) -->可能有数字或特殊符号

2. 预判字符串不全由小写字母和数字组成

(?![a-z0-9]+$) -->可能有大写字母或特殊符号

3. 定义正常逻辑：

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

(?![a-zA-Z]+$)(?![a-z0-9]+$)[a-zA-Z0-9]{8}

2. ***String 中支持正则的 API：

1. 查找：4 种：

1. 找固定内容的 1 个关键词：

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

每次只能找下*一个*

找不到，返回-1

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

如果没找到，返回-1

没有第二个参数，无法修改开始查找的位置

无法继续查找之后所有关键词

3. 获得所有关键词的内容和个数：reg 结尾+g

var kwords=str.match(reg)

如果 reg 结尾+g,表示查找全部匹配关键词

否则，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！

正则表达式，默认都是区分大小写

忽略大小写：reg 结尾+i

4. 查找每个关键词的内容和位置：regexp.exec(待续...)

2. 替换：str=str.replace(reg,"替换值");

替换一般都要加 g

课堂练习：3 个函数：

1. 去掉字符串开头的所有空字符: ^\s+
2. 去掉字符串结尾的所有空字符: \s+$
3. 去掉字符串开头和结尾的所有空字符: ^\s+|\s+$

去掉：将匹配的内容，替换为""

案例

```
<script>
    /*将一条消息编码为 unicode:每个字符必须编码 5 位数字*/
    var msg="今晚 308,w84u";
    for(var i=0,arr=[];i<msg.length;i++){
        var code=msg.charCodeAt(i);
        code=(100000+code+"").slice(1);
        arr.push(code);     }
    msg=arr.join(""); //将 msg 替换成了编码
    console.log(msg);
    /*解码：每隔 5 位获取一次子字符串
        将截取的 5 位 unicode，交给 String.fromCharCode(num)
        可获得 1 个字符 */
//从 0 开始，每隔 5 位遍历 msg 中的字符，到<length 结束，同时声明空数组 arr
for(var i=0,arr=[];i<msg.length;i+=5){
//    遍历过程中，截取 msg 中当前 i 位置开始，之后 5 个字符
    var code=msg.substr(i,5);
    //.slice(i,i+5);
    //.substring(i,i+5);
//    将获得的 5 个字符传入 String.fromCharCode(num)
    var char=String.fromCharCode(code);
//    将解码后的 1 个字符压入数组 arr 中
    arr.push(char);     }
   console.log(arr.join("")); //
    /*统计字符串中每种字符出现的个数
    var str="今晚 308,W84U";
    var arr=[];
    arr["num"]=0;
    arr["chs"]=0;
    arr["let"]=0;
    arr["oth"]=0;
    for(var i=0;i<str.length;i++){
        str[i]>="0"&&str[i]<="9"?arr["num"]++:
str[i]>="一"&&str[i]<="\u9fa5"?arr["chs"]++:
str[i]>="A"&&str[i]<="Z"
     ||str[i]>="a"&&str[i]<="z"?arr["let"]++:
         arr["oth"]++;}
    console.log(arr);
    /*统计字符串中每个字符重复的次数
    将当前字符作为 key，判断数组中是否有相同 key 的元素
    var str="you can you up";
    //遍历 str 中每个字符，同时创建一个空数组 arr
    for(var i=0,arr=[];i<str.length;i++){
    //    如果 arr 中以当前字符为下标的元素存在
        if(arr[str[i]]!==undefined){
    //        就为以当前字符为下标的元素值+1
            arr[str[i]]++;
        }else{//    否则
    //        为 arr 添加以当前字符为下标的新元素，值为 1
            arr[str[i]]=1;    }    }
    console.log(arr);
    /*找完数：所有因子的和等于本身
    for(var n=2,arr=[];n<1000;n++){
        //从 1 开始判断能否被 n 整除，到 n/2 结束
        for(var i=1,sum=0;i<=n/2;i++){
        n%i==0&&(sum+=i);//只要能整除，就将 i 累加到 sum
        }//(循环结束后，n==sum 说明是完数)
        n==sum&&arr.push(n);     }
    console.log(arr);
    /*100 米下落，每次弹起上次的一半
        第 n 次落地，经过的总距离
        下次弹起的高度
    var n=10;
    var h=100,up=h/2;
    for(var i=1;i<n;i++){//第一下落的 100，已经在 h 中
        h+=up*2; //将本轮下落的总距离累加到总高度 h 中
        up/=2; //下次弹起的高度
    }console.log(h+","+up);*/       </script>
```

```
<script>
    //检索指定位置右侧下*一个*关键词的位置：
    //      var i=str.indexOf("kword"[,fromi]);
    //      如果省略 fromi，默认从 0 开始找
    //检索指定位置左侧下*一个*关键词的位置：
    //      var i=str.lastIndexOf("kword"[,formi]);
    //      如果省略 fromi，默认从 length-1 开始
    //如果没找到都返回-1
```

```
var kword="我去";
var str="我去了女神家，我说：我去，你家真大";
        // 0                    12
//var index=-1;//从左向右
var index=str.length; //从右向左找
while((index=str.lastIndexOf(kword,index-1))!=-1){
//while((index=str.indexOf(kword,index+1))!=-1){
        //1. indexOf(kword,index+1)-->返回下标
        //2. 将下标存入 index 中
        //3. 将返回的下标和-1 比较
        //4. 当下一轮循环时，从 index+1 开始
    console.log("在位置"+index+"发现关键字");
    //从右向左，避免关键字在第 0 个位置时死循环
if(index==0){break;} //如果最后返回 0，则手动退出
}       </script>
```

```
<script>
    //对页面中的 html 元素按内容排序，再放回页面中
var html='<a href="#">smith</a><br><a href=
"#">eric</a><br><a href="#">jerry</a>';
    document.write(html+"<br>");
//Step1: 掐头去尾：从 12 开始到-4 结束，截取子字符串
    html=html.slice(12,-4);
    console.log(html);
//Step2: 按有规律的分隔符，分隔每个人名，得到人名数组
    var names=html.split('</a><br><a href="#">');
    console.log(names);
    //Step3: 对人名数组排序
    names.sort();
    console.log(names);
    //Step4: 怎么切割的，就再怎样 join 回去，补上开头和结尾
html='<a href="#">'    +names.join('</a><br><a href="#">')
+'</a>';
   document.write(html);
/*解析游戏角色字符串
Tom@补给兵@60%#Mary@医护兵@80%#John@特种兵@30%*/
var msg="Tom@补给兵@60%#Mary@医护兵@80%#John@特种兵@30%";
//Step1: 按#切割 msg，得到角色数组 roles
var roles=msg.split("#");
//Step2: 遍历角色数组中每个角色字符串
for(var i=0;i<roles.length;i++){
//    按@切割当前角色字符串，得到角色属性数组 props
    var props=roles[i].split("@");
//    输出: "Tom 补给兵 生命:60%"
    console.log(props[0]+" "+props[1]+" 生命:"+props[2]);}
/*鄙视题：*/
    //将一个英文句子中每个单词首字母改为大写
    var stmt="no zuo no die";
    //Step1: 按空格切割每个单词
    var words=stmt.split(" ");
    //Step2: 遍历 words 中每个单词
    for(var i=0;i<words.length;i++){
    //      将每个单词的首字母变为大写，拼接上剩余字母
    //      返回 words 中原位置
    words[i]=words[i][0].toUpperCase()+words[i].slice(1); }
    stmt=words.join(" ");
    console.log(stmt); //"No Zuo No Die";    </script>
<script>
    /*查找*/
    /*仅判断有没有
    var reply=prompt("输入回复");
    var reg=/买[了个]{0,2}表/;
    if(reply.search(reg)!=-1){ alert("发现敏感词!");
    }else{ alert("没有敏感词!");    }*/
    /*获得所有关键词的内容
    var str="No zuo no die; No can no bi bi";
    var reg=/no/ig; //g--global-->全部    //i-->忽略大小写
    var kwords=str.match(reg);
    console.log(kwords);
    if(kwords){//如果 kwords 有效(不是 null)
        console.log("共找到"+kwords.length+"个");
    }else{      console.log("没找到！");      }*/
    /*替换
    var str="\t Hello World \t";
    function ltrim(){ //去掉左侧的所有空字符
        return str.replace(/^\s+/,"");      }
    console.log(ltrim()); //"Hello World       "
    function rtrim(){ //去掉右侧的所有空字符
        return str.replace(/\s+$/,"");      }
    console.log(rtrim()); //"         Hello World"
    function trim(){ //去掉开头和结尾的空字符
        return str.replace(/^\s+|\s+$/g,"");      }
    console.log(trim()); //"Hello World"*/
    /*替换所有敏感词
        纪检    纪检监察    纪委      纪[检委](监察)?*/
    var str="纪委负责全国党员干部的纪检监察工作";
    var reg=/纪[检委](监察)?/;
    //反复使用 str.match 查找下一个关键词，直到返回 null 结束
    var kwords=null;
    while((kwords=str.match(reg))!=null){
    //      说明：match 方法不加 g，arr[0]-->本次找到的关键词
    //      替换关键词，同时判断当前关键词的字符个数
//  如果关键词是 2 个字，替换成"否则替换成"****"
    str=str.replace(reg,kwords[0].length==2?"**":"****")
     }console.log(str);
```

Day08

1. ***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/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\\ 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，

就用第二种方法。

查找：几乎都要加 g

var arr=reg.exec(str): 同时获得关键字的内容和位置

exec 每次只能找下一个匹配的关键词

arr[0]，保存了本次找到的关键词内容

如果找不到，返回 null

arr.index 属性中，保存了本次找到关键词的首字母位置

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

懒惰模式：只匹配最短的符合条件的字符串

贪婪改懒惰：.*-->(.*?)

如何获得本次查找结果中的部分内容：2 步：

1. 在正则表达式中用()，包裹住需要获得的部分

2. 在本次查找同时，使用 RegExp.$n，获得第 n 个()中匹配的子内容。

验证：几乎都要前加^，后加$

var bool=reg.test(str);

验证 str 是否符合 reg 的格式规定。

符合，返回 true，否则返回 false

何时使用：只要验证格式——唯一的办法

如何使用：可直接放入 if 中，作为条件

1. *Math：专门封装数学计算常用常量和计算方法的全局对象

***不能 new！所有 API 都直接用 Math.xxx

***Math 没有构造函数

何时使用 Math：只要进行数学计算

Math 的 API：

1. 取整：上取整 下取整 四舍五入取整

Math.ceil(n) Math.floor(n) Math.round(n)

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 按任意小数位置 String

自定义 round 函数:按任意位数四舍五入，返回 Number

2. 乘方、开平方：

乘方：Math.pow(底数,幂)：比如：Math.pow(10,2)

开平方：Math.sqrt(n): 将 n 开*平*方

3. 获取最大值,最小值：

Math.max(n1,n2,n3,...) :给定多个*数字*中的最大值

min 给定多个*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？

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

min min

4. 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)

Math.floor

比如: 60 ~ 100

Math.random() 0<=r<1

Math.random()*41 0<=r<41

Math.random()*41+60 60<=r<101

Math.floor(xxx) 60<=r<=100

2. ***Date: 封装一个 1970 年元旦至今的毫秒数，

提供了对时间操作的方法

***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

创建对象：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]");

3. //自定义任意时间： var date=new Date(xxxx,xx,xx,xx,xx,xx);

**现实的月份-1=计算机中的月份

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

var date2=new Date(date1.getTime());//取出 date1 的毫秒数

问题：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

解决：在计算前，先复制日期对象的一个副本，再计算

Date 的 API：

分量： 年 月 日 星期

FullYear Month Date Day

4 位年

时 分 秒 毫秒

Hours Minutes Seconds Milliseconds

1. 每个分量都有一对儿 get/set 方法

get 方法用来获得分量的值

set 方法用来设置分量的值 **set 方法自动调整进制！

比如：获得今天几号: date.getDate();//月中的日

设置日期的月份为 4 月: date.setMonth(3) 星期没有 set 方法。

2. 年月日星期不带 s 结尾， 时分秒毫秒带 s 结尾，

3. 返回值：只有 Date:从 1 到 31 结束

其余都是从 0 开始，到进制-1 结束 只有 Month 需要修正

比如：月份：1,2,3....11,12 星期：日,一,二,.....六

0,1,2....10,11 0 1 2 6

小时：0 1 2 3 4 ....23 分钟：0 1 2.....59

Date 计算：2 种

1. 两日期对象可直接相减,结果时毫秒差！

2. 对每个分量做加减:3 步: get 分量值，做加减，set 回去

一步概括：date.setXXX(date.getXXX()+/-n);

作业：1. 计算任意工作日后的日期

2. 自定义 format 函数

“2015 年 11 月 11 日 星期三 下午 05:55:xx”

function format(date){ var week=['日','一',...,'六']; return str; }

/*杨辉三角*/

```
var data=[ [" 1"],  [" 1"," 1"]  ];
var l=8; //总行数
function calc(){
    for(var r=2;r<l;r++){ //r 表示当前行下标
        var arr=[" 1"];
        for(var i=1;i<r;i++){//每行只计算 r-1 次
        //i 当前行中正在计算的下标，0 位置不用算
          //当前行的当前元素值
// =上一行相同位置的元素值+上一行前一位置的元素值
var n=Number(data[r-1][i])+Number(data[r-1][i-1]);
            arr.push(n<10?" "+n:n);
        }arr.push(" 1"); data.push(arr);
    }    }
function show(){
    for(var r=0;r<l;r++){
        for(var i=0,line="";i<l-r+1;line+="  ",i++);
        line+=data[r].join("  ");
        console.log(line);
    }    }    calc(); show();
```

/*猴子吃桃：倒推

```
var d1=0,d2=1;//假设前一天桃子数=0，最后一天桃子数为 1
    for(var days=9;days>0;days--){
        d1=(d2+1)*2;        d2=d1;
    }console.log(d1);
```

/*打印方片

```
    var l=9;
    for(var r=1;r<(l+1);r++){//外层循环控制行
        for(var c2=1,arr=["*"];c2<(l+1)/2;c2++){
            if(r<=(l+1)/2){//上半段逻辑
                arr.unshift(c2<r?"**":"  ");
                arr.push(c2<r?"**":"  ");
            }else{//下半段逻辑
                arr.unshift(c2<=l-r?"**":"  ");
                arr.push(c2<=l-r?"**":"  ");
            }
        }console.log(arr.join(""));
    }  </script>
```

```
<script>
    /*var arr=reg.exec(str)*/
    var reg=/no/ig;
    var str="No zuo no die;No can no bi bi";
    var arr=null;
    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");
    }
    /*获取 html 中所有 a 元素中 href 属性中的 url*/
var html="<link rel='stylesheet' href='http://index.css'/>"+
"<a name='Top'></a>"+"<a class='navi' href='http://tmooc.cn'></a>"+
"<h1 class='title1'></h1>"+
"<a href='http://tedu.cn' target='_blank'></a>"+"<div></div>";
    var reg=/<a\s+(.*?)href=['"]([^'"]*)['"]/ig;
                    // 1                2
    var urls=[]; //保存所有找到的 url
    var arr=[]; //临时保存每次 exec 查询的结果
    //反复调用 reg.exec，将结果保存到 arr 中，直到==null 退出
    while((arr=reg.exec(html))!=null){
    //    将本次查找结果中第 2 个圆括号中的子内容压入 urls
        urls.push(RegExp.$2);
    }//(循环退出后)输出 urls
    console.log(urls.join("\n"));
```

/*验证密码强度

```
var reg=/^(?![a-zA-Z]+$)(?![a-z0-9]+$)[0-9A-Za-z]{6}$/;
    if(reg.test(prompt("输入密码：""))){
        alert("密码格式安全");
    }else{  alert("密码强度太低")  }*/
```

/*str.split(reg)*/

```
    var html="<li>smith</li><br>"+ "<li>eric</li>"+
                "<li class='curr'>tom</li><br>"+ "<li>scott</li>";
    var reg=/<\/li>\s*.*?<li.*?>/;
    html=html.trim().slice(4,-5);
    console.log(html);
    var names=html.split(reg);
    console.log(names);
```

```
    /*自定义四舍五入的函数
        比如：234.345   2          Math.round()*/
    function round(num,d){
        //将 num*= 10 的 d 次幂
        num*=Math.pow(10,d)
        //将 num 四舍五入取整
        num=Math.round(num.toFixed(2)); //尽量屏蔽舍入误差
        //将 num/= 10 的 d 次幂
        num/=Math.pow(10,d);
        return num;
    }
    console.log(round(23.456,2)); //23.46
    console.log(round(345.56,0)); //346
    console.log(round(345.56,-1)); //350
    console.log(round(555.555,2)); //555.56  其实 555.55
```

/*使用 max 实现对数组的升序/降序排列*/

```
    var arr=[3,2,5,8,6,7,4];
    var result=[];
    //反复获得 length-1 次 max 值
```

```
    for(var c=1,len=arr.length;c<len;c++){
        var min=Math.min.apply(Math,arr);
        result.unshift(min); //将 max 从头部压入结果数组
中
        //从 arr 中删除和 max 相同的元素
        for(var i=0;i<arr.length;i++){
            if(arr[i]==min){
                arr.splice(i,1);
                break;
            }   }   }
result.unshift(arr[0]);//将 arr 中最后一个元素压入结果数组
    console.log(result);
```

```
    function doubleBall(){
    //第一个红球肯定没有重复，直接放入 reds 中即可
        var reds=[parseInt(Math.random()*33+1)];
        /*反复随机生成 6 个不重复的红球*/
    while(reds.length<6){//循环条件:红球的个数小于 6 个
    //      从 1~33 之间随机生成一个整数，保存在 r 中
            var r=parseInt(Math.random()*33+1);
    //遍历 reds,因循环内可能修改元素个数，所以先存 len
            for(var i=0,len=reds.length;i<len;i++){
                if(reds[i]==r){//如果当前元素==r
                    break;//退出遍历
                }else if(i==len-1){
        //          否则,如果当前位置已经是最后一个
                    reds.push(r);//将 r 压入 reds 中
                }   }
        }//(循环退出后)将 reds 按数字升序排列
        function compare(a,b){return a-b};
        reds.sort(compare);
        //从 1~16 之间随机生成一个整数，存入 blue 中
        var blue=parseInt(Math.random()*16+1);
        return reds.join()+"    "+blue;     }
    console.log(doubleBall()); //
    console.log(doubleBall()); //
    console.log(doubleBall()); //
    console.log(doubleBall()); //
    console.log(doubleBall()); //
    /*随机生成验证码*/
    var chars=[]; //62 个备选字符
    /*从 48 号开始，到 57 结束，顺序生成 0-9 十个字符*/
    for(var c=48;c<=57;c++){
        chars.push(String.fromCharCode(c));
    }
    /*从?号开始，到?结束，顺序生成 A-Z 十个字符*/
for(var c=65;c<=90;chars.push(String.fromCharCode(c++)));
    /*从?号开始，到?结束，顺序生成 a-z 十个字符*/
for(var    c=97;c<=122;chars.push(String.fromCharCode(c++)));

function getCode(){ //获得随机的 4 个字符组成的验证码
        var code=[];//定义一个数组 code
        //反复生成随机数,直到 code 中元素个数==4，不再生成
        while(code.length<4){
        //      从 0~61 之间，随机生成一个下标位置 i
            var i=parseInt(Math.random()*62);
        //      将 chars 中 i 位置的元素压入 code
            code.push(chars[i]);
        }return code.join(""); //XXXX
    } var code=getCode();
    var input="";
//请用户反复输入验证码(code)保存到 input，直到输入正确为止
while((input=prompt("输入验证码:"+code)).toUpperCase()
      !=code.toUpperCase()){
    alert("验证码错误");//      提示验证码错误
    code=getCode();//          重新获得验证码          }
//(退出循环)提示验证通过
alert("验证通过");
```

```
    var now=new Date();
    var birth=new Date("1983/12/26");
    var mils=now-birth; //毫秒差
    console.log(mils);
    //mils 换算成天数:
    var days=Math.floor(mils/1000/60/60/24);
    console.log("已经活了 "+days+"天");
    var age=Math.ceil(days/365);
    console.log("年龄："+age);
    /*计算机可存储的最大年数*/
    var years=Number.MAX_VALUE/1000/3600/24/365;
    console.log(years);
    /*计算合同到期时间*/
    var hiredate=new Date("2012/06/30"); //入职时间
    var enddate=new Date(hiredate.getTime());
    //3 年有效期，求到期时间：+3 年
    enddate.setFullYear(enddate.getFullYear()+3);
    //到期时间之前一个月，续签合同：-1 月
    var resdate=new Date(enddate.getTime());
    resdate.setMonth(resdate.getMonth()-1);
    //判断是否周六或周日
    if(resdate.getDay()==6){ //-1 天
        resdate.setDate(resdate.getDate()-1);
    }else if(resdate.getDay()==0){ //-2 天
        resdate.setDate(resdate.getDate()-2);     }
    //续签时间一个星期前，提醒续签：提前一周,-7 天
    var alertdate=new Date(resdate.getTime());
    alertdate.setDate(alertdate.getDate()-7);
    console.log("入职时间"+hiredate.toLocaleDateString());
    console.log("到期时间"+enddate.toLocaleDateString());
                //将日期转为字符串
    console.log("续签时间"+resdate.toLocaleDateString());
console.log("提醒时间"+alertdate.toLocaleDateString());
```

Day09

1. *错误处理：

错误：程序执行过程中导致程序异常中断的状态

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

Error 子对象：6 种：

ReferenceError: 引用错误，找不到对象

SyntaxError: 语法错误

TypeError: 类型错误，错误的使用类型或类型的方法

RangeError: 范围错误，参数超范围

EvalError、URIError

错误处理：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

err.message: 保存了错误的详细原因

err.name: 保存了错误对象的名称

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*

一般释放用过的对象资源：xxx=null

}]

其实，如果可以提前预知错误的原因：建议使用 if 代替 try catch

执行效率：try 中尽量少的包含代码

try 的执行效率低，且多创建一个 error 对象

抛出自定义错误：

throw new Error("自定义错误消息");

1. ***Function 对象

***重载：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。

js 的语法不支持重载的。可使用 arguments 实现重载效果。

何时使用重载：不同参数，不同操作，但操作名称相同

目的：减轻调用者调用时选择的负担

***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”——长得像数组的*对象*

2 点像数组：arguments.length： 参数值的个数

arguments[i]: 获得下标为 i 位置的参数值

鄙视题：类数组对象 vs 数组 差别？

类数组对象不是 Array 类型

数组是 Array 类型的对象

参数变量: 提醒调用者，如何正确使用函数

参数变量更简便

***内存中的函数对象：3 个阶段：

定义时：创建 2 个对象：函数对象和作用域链对象

函数对象：封装了函数的定义，但暂时不读取函数定义

**作用域链对象：专门保存函数可用的变量所在位置的对象

调用时：创建 1 个新对象：活动对象

活动对象：临时封装本次调用函数时使用的局部变量

并将活动对象的引用，压入作用域链对象中。

调用后：作用域链中的活动对象引用出栈

活动对象及其内部的局部变量一同释放！

***如何创建函数对象：3 种方式

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}

3. new Function：

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

鄙视题：js 中创建函数三种方式：

function compare(a,b){return a-b;}

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

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

***匿名函数：定义时没有变量引用的函数

何时使用：2 种：

1. 匿名函数自调：定义完立刻执行，执行完立刻释放

何时使用自调：一个函数只用一次！

如何使用：

(function(参数列表){

函数体;

return 值

})(参数值列表);

2. 回调：将函数作为对象传递给其他函数，

由其他函数调用。

何时使用回调：如果一个函数只被另一个函数调用时

优点：节省内存空间

缺点：无法返回使用

****闭包：

问题：全局变量：在程序任何位置都可访问(修改)

全局污染

局部变量：不可重用！

解决：即要重用变量，又要保护变量不被污染——闭包

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. 再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。

判断：3 特点：

1. 函数嵌套！
2. 内层函数使用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。

何时使用闭包：即要重用变量，又要保护变量不被污染

缺点：占用更多内存空间——因为 outer 的活动对象无法释放。

判断闭包输出: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。

/*完全平方数

```
for(var n=1;n<10000;n++){
    if((Math.sqrt(n+100)+"").indexOf(".")==-1
        &&(Math.sqrt(n+268)+"").indexOf(".")==-1){
        console.log(n);
    }
}
```

/*乒乓球

```
var t2=["x","y","z"];
for(var a=0; a<3; a++){
    for(var b=0; b<3; b++){
        if(a!=b){ //去掉 a 和 b 是同一对手的可能
            for(var c=0; c<3; c++){
            if(a!=c&&b!=c){//去掉和 c 相同对手的可能


if(t2[a]!="x"&&t2[c]!="x"&&t2[c]!="z"){

console.log(t2[a]+t2[b]+t2[c]);
                    }
            }
            }
        }
    }
}
```

/*计算 n 个工作日之后

```
var date=new Date("2015/5/29");
var days=9;
while(days>0){
    if(date.getDay()==5){
        date.setDate(date.getDate()+2);
    }else if(date.getDay()==6){
        date.setDate(date.getDate()+1);
    }else{
        date.setDate(date.getDate()+1);
        days--;
    }
}
date.setDate(date.getDate()+1);
console.log(date.toLocaleDateString());*/
```

/*自定义 format 函数，格式化日期*/

```
function format(date){
    var week=['日','一','二','三','四','五','六'];
    var result=[]; //保存每个分量的值
    result[0]=date.getFullYear();
    result[1]=date.getMonth()+1;
    result[1]<10&&(result[1]="0"+result[1]);
    result[2]=date.getDate();
    result[2]<10&&(result[2]="0"+result[2]);
    result[3]=week[date.getDay()];
    var hours=date.getHours();
    result[4]=hours<12?"上午":"下午";
    hours>12&&(hours-=12);
    hours<10&&(hours="0"+hours);
    result[5]=hours;
    result[6]=date.getMinutes();
    result[6]<10&&(result[6]="0"+result[6]);
    result[7]=date.getSeconds();
    result[7]<10&&(result[7]="0"+result[7]);
    result=result.join("");//20151112 四上午 100801
    var reg=
/(\d{4})(\d{2})(\d{2})([\u4e00-\u9fa5])([\u4e00-\u9fa5]{2})(\d{2})(\d{2})(\d{2})
/;
return result.replace(reg ,"$1 年$2 月$3 日 星期$4 $5$6:$7:$8");
}console.log(format(new Date()));
```

```
<script>
    var n=345.456;
    /*var d=prompt("输入四舍五入的位数");
    if(d>=0&&d<=20){//try{
        console.log(n.toFixed(d));
        //出错位置及其后所有代码不执行
    }else{//}catch(err){//只有发生错误才执行
        alert("位数必须在 0~20 之间");
    }//finally{//无论是否发生错误，永远执行
    console.log("释放资源");
    //}
```

/*判断浏览器是否支持 XMLHttpRequest

```
    if(window.XMLHttpRequest){//try{
        var a=new XMLHttpRequest();
        document.write("支持");
    }else{//catch(err){
        document.write("浏览器不支持");
    }//finally{
    a=null;     //}*/
```

/*鄙视题

```
    var n;
    function fun(){//不出错,finally 中没 return
                    //出错,finally 中没 return
```

```
                    //不出错,finally 中有 return
                    //出错,finally 中有 return
            try{n=1;
                //console.log(m);//ReferenceError
                //如果出错，之后的代码不执行
                return n;
            }catch(err){ n=2;
                return n;   //return 2;等待
            }finally{ n=3;
                //return n;//替换前边的所有 return
            }
        }
        console.log(fun()); //
        console.log(n);       //*/
        function round(num,d){/*程序员甲：定义函数*/
            if(!isNaN(d)){
return
Math.round(Math.pow(10,d).toFixed(2))/Math.pow(10,d);
            }else{//抛出自定义错误
                throw new Error("第二个参数必须是数字");
            }
        }
        function fun(){/*程序员乙：调用函数*/
            var num=prompt("输入要四舍五入的值");
            var d=prompt("输入位数");
            try{
                alert(round(num,d));
            }catch(err){
                alert("位数必须是数字");//用户看到，友好
的
                console.log(err.message);
                //错误消息，给程序员看
            }
        }
        fun();
    </script>
```

```
<script>
        function calc(){
            if(arguments.length==1){
                return arguments[0]*arguments[0];
            }else{
                return arguments[0]+arguments[1];
            }
        }
        console.log(calc(13));   //169
        console.log(calc(12,23));   //35
        function add(){
        //遍历 arguments 中每个参数值，同时声明 sum 变量
为 0
            for(var i=0,sum=0;i<arguments.length;i++){
                //      将当前参数值累加到 sum 上
                    sum+=arguments[i];
            }return sum;
        }console.log(add(1,2,3)); //6
        console.log(add(1,2,3,4)); //10
        console.log(add(1,2,3,4,5)); //15
```

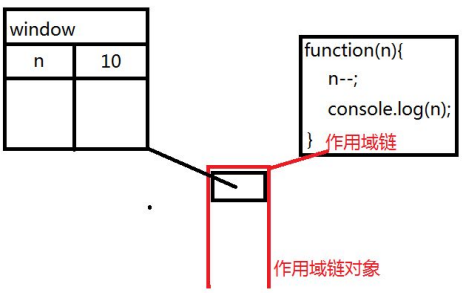
```
<script>
        var fun;
        fun=function (){
            console.log(1);
        }
        fun(); //1
        fun=function (){     console.log(2);     }
        fun(); //2
        /*fun=100;
        fun();//X*/
        var n=10;
        (function(n){
            n--;
            console.log(n);
        })(n); //9
        console.log(n); //10
        /*匿名函数回调*/
        var arr=[3,5,2,1,6,4,7];
        //内存中没有比较器函数对象
        arr.sort(function(a,b){return a-b});
        //比较器函数对象被释放
        console.log(arr);
```

```
        /*(function fun1(){    alert("页面开始加载"); })();*/
        /*(function(){    alert("页面加载完毕"); })();*/
<script>
        /*function outer(){
            var n=0;
            return function(){return ++n;}
        }
        var getNum=outer();
          //getNum:function(){return ++n;}
        console.log(getNum()); //1
        n=0;
        console.log(getNum()); //1*/
        /*汇率转换*/
        function change(rate){//rate 代表汇率
            return function(rmb){
                    return (rmb/rate).toFixed(1);
                }
        }
        var to$=change(6.1); //to$ 专门使用 rate=6.1
        console.log(to$(1000)); //163.9
        console.log(to$(5000)); //819.67
        var toEuro=change(6.9); //toEuro 专门使用 rate=6.9
```

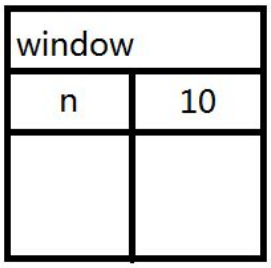
```
console.log(toEuro(1000)); //144.9
/*鄙视题
function outer(){
    for(var i=0,arr=[];i<3;i++){
        arr[i]=function(){return i};//i 是受保护的变量
    }
    return arr;//循环结束后，i=3
}
var getFuns=outer();*/
//外层函数调用了 1 次，只有 1 个受保护的变量 i=3
/*getFuns: [function(){return i},function(){return i},function(){return i}
]*/
//console.log(getFuns[0]()); //3
//console.log(getFuns[1]()); //3
//console.log(getFuns[2]()); //3
/*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);*/
var getSecret,setSecret;//等待接收访问器函数对象
(function(){//只在开始时，创建访问器函数
    var secret=0;
    getSecret=function(){ return secret; };
    setSecret=function(value){ secret=value; }
})(); //只有一个 secret=0
//getSecret: function(){ return secret; }
//setSecret: function(value){ secret=value; }
console.log(getSecret()); //0
setSecret(10); //secret=10
console.log(getSecret()); //10
```

anonyFunc

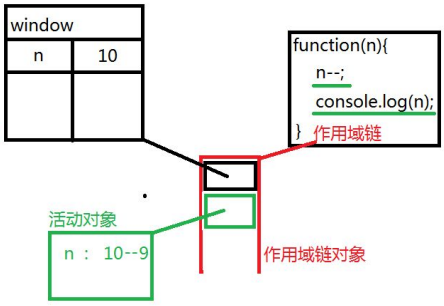
```
var n=10;
(function(n){
    n--;
    console.log(n);
})(n); //定义时
console.log(n);
```



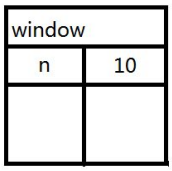

```
var n=10;
(function(n){
    n--;
    console.log(n);
})(n);
console.log(n);
```




```
var n=10;
(function(n){
    n--;
    console.log(n);
})(n); //调用时 -->9
console.log(n);
```

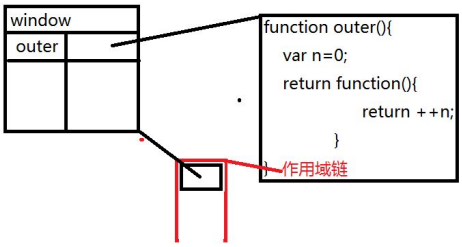



```
var n=10;
(function(n){
    n--;
    console.log(n);
})(n); //调用后
console.log(n);
```

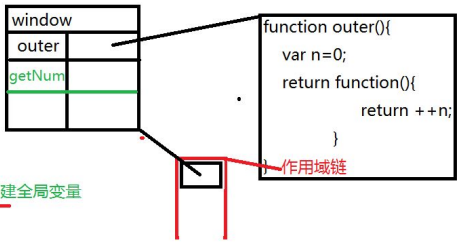


closure

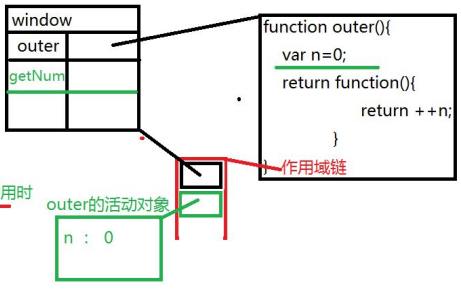
```
function outer(){
    var n=0;
    return function(){
        return ++n;
    }
}
var getNum=outer();
console.log(getNum());
console.log(getNum());
n=0;
console.log(getNum());
```

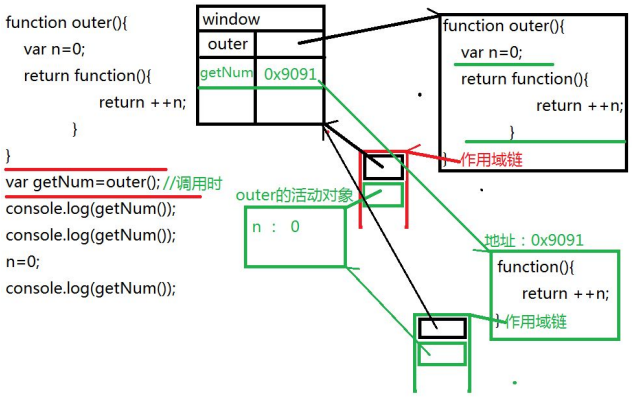


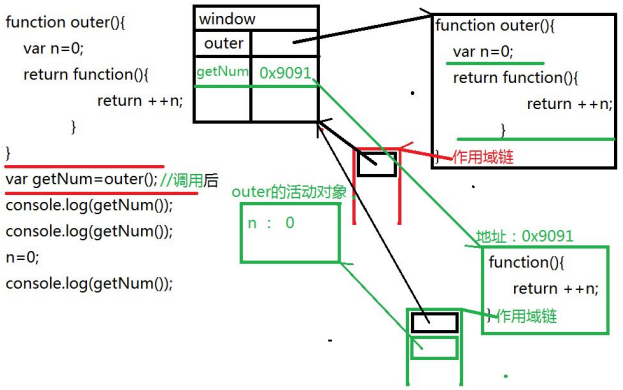

```
function outer(){
    var n=0;
    return function(){
        return ++n;
    }
}
var getNum=outer();//创建全局变量
console.log(getNum());
console.log(getNum());
n=0;
console.log(getNum());
```

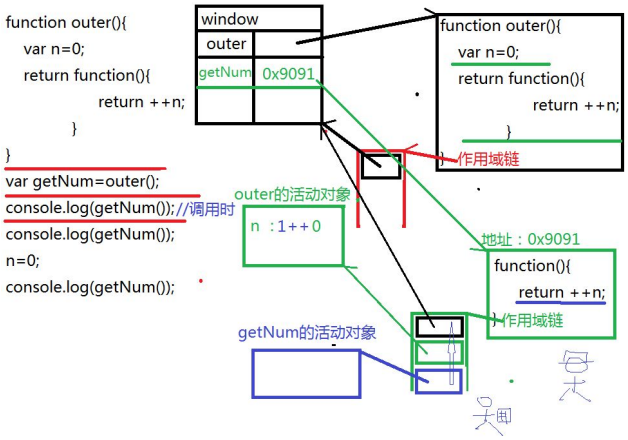


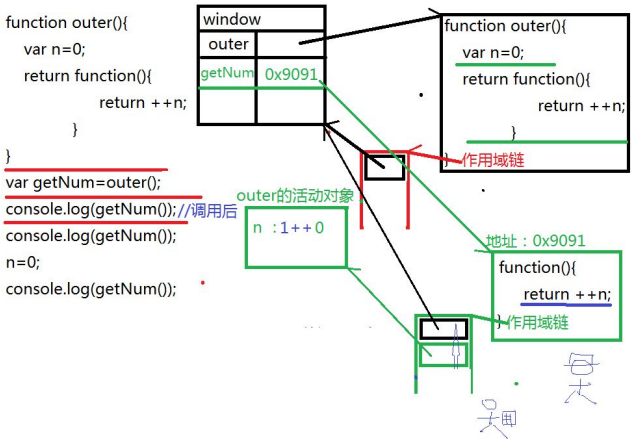

```
function outer(){
    var n=0;
    return function(){
        return ++n;
    }
}
var getNum=outer();//调用时
console.log(getNum());
console.log(getNum());
n=0;
console.log(getNum());
```

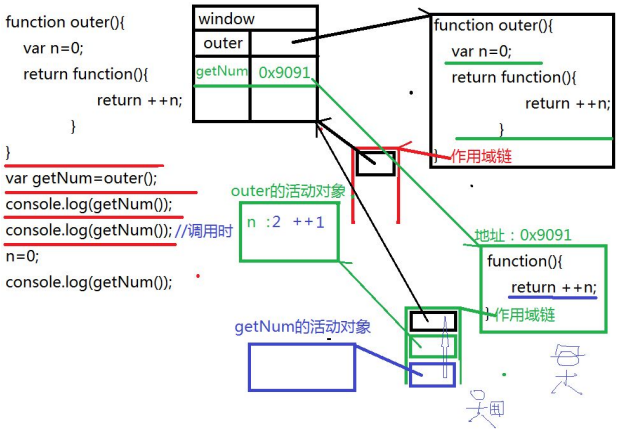


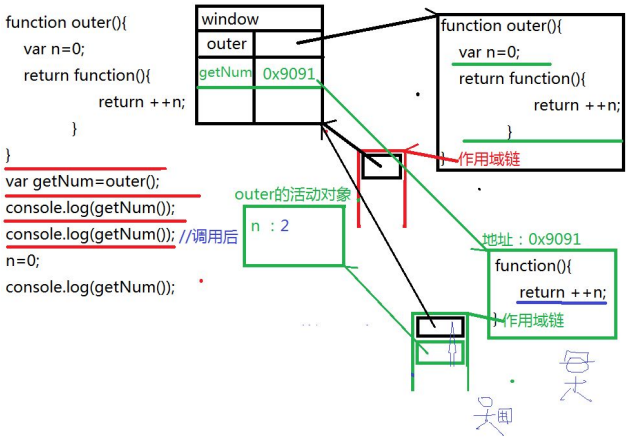




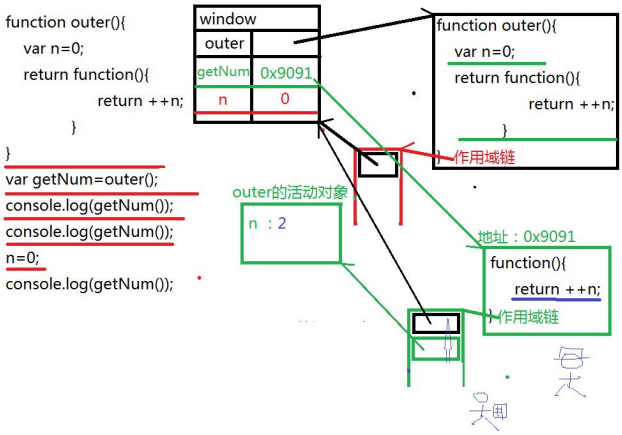




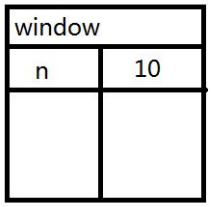




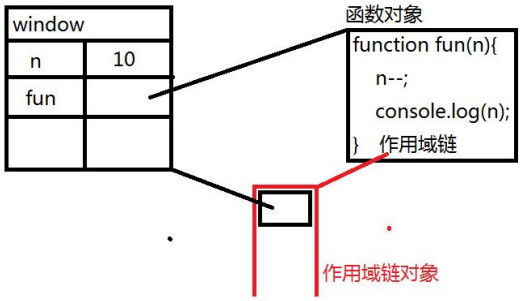
func

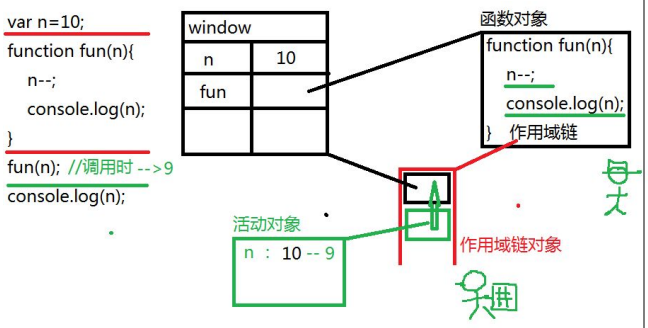


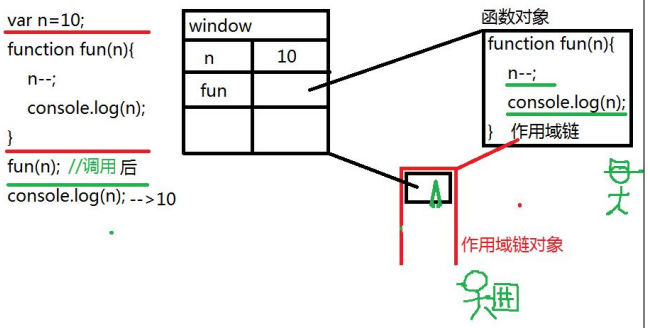

```
var n=10;
function fun(n){
    n--;
    console.log(n);
}
fun(n);
console.log(n);
```




```
var n=10;
function fun(n){
    n--;
    console.log(n);
}
fun(n);
console.log(n);
```







Day10

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

属性和方法统称为对象的成员。

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. 创建对象：3 种：

1. 创建一个单独的对象：2 种：

1. 对象直接量方式：

var obj={

属性名:属性值,

方法名:function(参数列表){方法体;return 返回值}

};

*****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。

如何访问对象中的属性和方法？点-->的

比如：对象.属性-->一个变量

对象.方法()-->一个函数

2. 使用 new 创建：

2 步：先创建空对象；再为空对象追加新*成员*

var obj=new Object(); -->{}

obj.属性名=属性值;

obj.方法名=function(){xxx};

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：

什么是构造函数：

专门定义一类对象的统一属性和方法的结构的函数

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(){...}

}

*****什么是 this：值随时执行当前正在使用的对象：

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.方式访问当前对象的属性

this 默认为 window　所有无主的都是 window 的

***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);

new: 4 件事：

1. 创建一个空对象

2. 调用构造函数,为空对象添加新属性和方法。

this 指上一步刚创建的空对象

3. ？

4. 将新对象的地址保存到 obj 变量中

1. ***原型和原型链

原型(prototype):保存所有子对象共有成员的对象

***每个构造函数都有一个原型属性

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

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。

找不到，再去构造函数的原型中查找。

new：1. 创建空对象

2. 调用构造函数，为新对象添加成员

3. 设置当前对象的__proto__属性为构造函数的原型对象　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

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

原型相关 API：

1. 判断对象的属性是自有，还是共有：3 种情况

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：

1. 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

2. 使用===undefined：说明不包含！

简写为(!对象.成员名)

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,没有返回 false

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)

2. 获得任意对象的__proto__属性：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

删除对象中的属性：delete 对象.成员

***只能删除自有的成员

***只有 var 声明的全局变量不让 delete

使用 window.或 window[""]增加的全局成员可以 delete

如何判断一个对象是数组，几种方法：4 种

***typeof 无法区分数组和一般对象，不能用来识别数组！

1. Array.prototype.isPrototypeOf(obj)

如果返回 true，说明是数组，否则不是

2. instanceof: 判断一个对象是否由某个类型创建出来

对象 instanceof 类型名

obj instanceof Array 返回 true，说明是数组

***instancof 会查找原型链上所有原型的构造函数

3. getPrototypeOf:

Object.getPrototypeOf(obj)==Array.prototype

4. Object.prototype.toString()方法：

***重写：override，子对象觉得父对象的成员不好用，

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)

apply(arr)

call vs apply: 差在第二个参数上！

Math.max.apply(Math,[1,2,3,4]);——才可获得数组最大值

apply，第二个参数要求将所有要传入的参数放在一个数组中

Math.max.call(Math,1,2,3,4)——不能用于数组

call,第二个参数要求将所有的参数，必须一个一个独立传入

5. Array.isArray(obj): 专门判断对象是否是数组！

*****继承：

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！

继承：3 种方式：

1. 一个对象，直接继承另一对象：

Object.setPrototypeOf(子对象,父对象)

相当于:子对象.__proto__=父对象

2. 修改构造函数的原型对象，统一修改之后创建的所有子对象的公共原型：

构造函数.prototype=父对象

***必须在首次实例化对象之前！

案例

```
<script>
    function Student(sname,age){
        this.sname=sname;
        this.age=age;
        /*this.intrSelf=function(){
            xxx
        }*/
    }
    var father={car:"BMW",balance:1000000000}
    Student.prototype=father;
//构造函数的原型对象 添加 intrSelf 函数
    Student.prototype.intrSelf=function(){
        console.log("I'm "+this.sname+",I'm "+this.age);
    }
    var lilei=new Student("Li Lei",18);
    var hmm=new Student("Han Meimei",19);
    //判断 hmm 能不能用 intrSelf
    console.log("intrSelf" in hmm);
    //判断 lilei 能不能用 intrSelf
    console.log("intrSelf" in lilei);
    lilei.intrSelf();
    hmm.intrSelf();
    /*hmm.__proto__=father;
    console.log(hmm.car);
    console.log(hmm.balance);
    console.log(lilei.car); //undefined*/
    /*hmm.money=5;
    console.log(lilei.hasOwnProperty("money")); //false
    console.log(hmm.hasOwnProperty("money"));//true
    delete hmm.money;
    console.log(hmm.hasOwnProperty("money"));//false*/
    /*console.log(!(hmm.hasOwnProperty("intrSelf"))
                &&"intrSelf" in hmm); //true
    console.log(!(hmm.hasOwnProperty("sname"))
                &&"sname" in hmm); //false
```

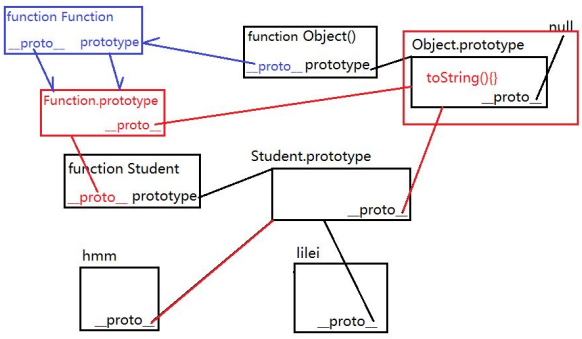
```
        console.log(
            Object.getPrototypeOf(hmm)//获得 hmm 的父对象
            ==Object.getPrototypeOf(lilei)//获得 lilei 的父对象
        ); //true*/
    </script>



<script>
    /*当前浏览器中 String 类型的原型对象是否包含 trim 方法*/
    if(!("trim" in String.prototype)){//如果不包含
        //只有放在原型中的方法才能被所有子对象调用
        String.prototype.trim=function(){
            return this.replace(/^\s+|\s+$/g,"");
        }
    }
    var str="\t hello world \t";
    document.write(str.trim());
    /*判断并自定义数组对象的 indexOf 方法*/
    /*indexOf 不在数组类型的原型对象中*/
    if(!Array.prototype.indexOf/*===undefined*/){
//在数组类型的原型对象中添加 indexOf 成员赋值为函数：
                //两个参数：kword,fromi
        Array.prototype.indexOf=function(kword,fromi){
//      如果 formi===undefined，则 fromi=0;
        fromi===undefined&&(fromi=0);
//      从 fromi 开始，遍历当前正在调用 indexOf 方法的数组对象
            for(;fromi<this.length;fromi++){
                //如果当前元素==kword
                if(this[fromi]==kword){
                    return fromi;//返回当前位置 i
                }
            }return -1;//(遍历结束)返回-1;
        }
    }
    var arr=["no","zuo","no","die"];
            // 0      1       2       3
    document.write(arr.indexOf("no")); //0
    document.write("<br>");
    document.write(arr.indexOf("no",1)); //2
    document.write("<br>");
    document.write(arr.indexOf("no",3)); //-1
```

```
    /*判断一个对象是不是数组*/
    var arr=[1,2,3]; //Array.prototype
    var obj={}; //Object.prototype
    var fun=function(){}; //Function.prototype
    /*typeof 无法区分普通对象和数组*/
    console.log(typeof arr); //object  判断失败
/*isPrototypeOf():判断数组类型的原型是不是 arr 的父级对象*/
    if(Array.prototype.isPrototypeOf(arr)){
        console.log("arr 是数组");
    }
/*instanceof 判断 arr 是否由数组类型创建出来*/
    if(arr instanceof Array){
        console.log("arr 是数组");
    }
/*getPrototypeOf:判断 arr 的父级对象是不是数组类型的原型*/
    if(Object.getPrototypeOf(arr)==Array.prototype){
        console.log("arr 是数组");
    }
/*调用 Object 的原型中的原生 toString 输出对象的类型名*/
    console.log(Object.prototype.toString.call(arr));
</script>
<script>
    function Student(sname,age){
        this.sname=sname;
        this.age=age;
    }
    var father={car:"BMW"};

    Student.prototype.intrSelf=function(){
        console.log("I'm "+this.sname+"I'm "+this.age);
    }
    /*"实例化"两个学生类型的对象*/
    var lilei=new Student("Li Lei",18);
    /*修改构造函数的原型对象为 father*/
    Student.prototype=father;
    var hmm=new Student("Han Meimei",19);
    lilei.intrSelf();//this-->lilei
    /*只让 hmm 继承 father
    Object.setPrototypeOf(hmm,father);*/
    console.log(hmm.car); //
    console.log(lilei.car); //
```

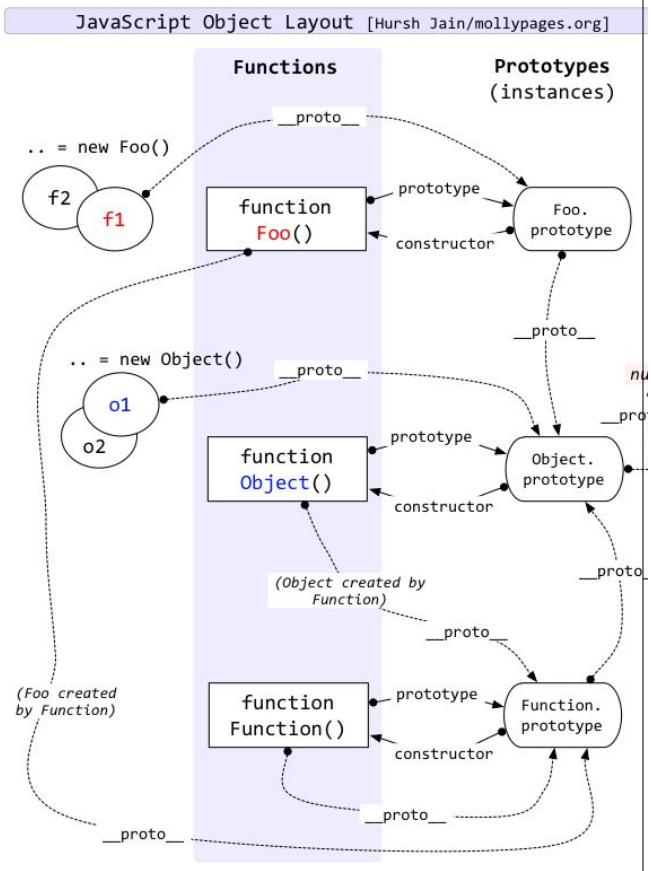



```
<title>自我介绍</title>
        <div id="parentDiv">
<div id=
"lilei"onmouseover="lilei.intrSelf()"
onmouseout="dialog.style.display='none'">
</div>
<div id="hmm" onmouseover=
"hmm.intrSelf()" onmouseout="dialog.style.display='none'">
</div>
    <div id="dialog"><!--innerHTML--></div>
        </div>
        <script>
            /*只描述李雷一个同学:
                姓名: Li Lei
                年龄: 18
                自我介绍: 说："I'm Li Lei,I'm 18";
            var lilei={//对象
                "sname":"Li Lei",
                "age":18,
                "intrSelf":function(){
                    alert("I'm Li Lei,I'm 18");}   }*/
            /*描述韩梅梅:
            var hmm=new Object();//关联数组
            hmm.sname="Han Meimei";
            hmm.age=19;
            hmm.intrSelf=function(){
                    alert("I'm Han Meimei,I'm 19");
            }*/
            /*对象 vs 关联数组
            console.log(lilei.constructor); //Object
            console.log(hmm.constructor); //Array
            console.log(lilei["sname"]);
            console.log(hmm.sname);
            lilei.intrSelf();//调用 lilei 对象中的方法*/

            /*构造函数*/
            function Student(sname,age){
                var self=this;
                self.sname=sname;
                //当前对象的
                self.age=age;
                self.intrSelf=function(){
                var msg="I'm "+self.sname+",I'm "+self.age;
                //找到页面上 id 为 dialog 的 div 元素对象
                var div=document.getElementById("dialog");
                //将 div 的 innerHTML 设置为 msg
                div.innerHTML=msg;
                //将 div 的 style 的 display 属性设置为"block";
                div.style.display="block";   }
            }
            var lilei=new Student("Li Lei",18);
            var hmm=new Student("Han Meimei",19);
            console.log(lilei.sname);
            console.log(hmm.sname);
            //lilei.intrSelf(); //this-->lilei
              //this.sname-->lilei.sname
            /*鄙视题：*/
            function foo(){return this.a};
            var a=2;
            var o={a:3,foo:foo};
            var p={a:4};
            console.log(o.foo()); //?
            p.foo=o.foo;
            console.log(p.foo()); //?
            console.log(foo()); //?
        </script>
```

```
#parentDiv{
    width:492px; height:382px;
    background-image:url("images/hanmeimei.jpg");
    background-repeat:no-repeat;
    position:relative;
    overflow:hidden;
}
#hmm{
    width:50px; height:60px;
    cursor: pointer;
    margin-top:135px; margin-left:190px;
    position:absolute;
}
#lilei{
    width:80px; height:70px;
    cursor: pointer;
    margin-top:90px; margin-left:65px;
    position:absolute;
}
#dialog{
    position:absolute;
```

```
        width:160px; height:120px;
        padding:12px;
        background-image:url("images/dialog.png");
        background-repeat:no-repeat;
        font-size:1.5em;
        top:0px;left:280px;
        display:none;
}
```



回顾：

1.*****继承：3 种：

1. 一个对象，只继承另一个对象：

   修改子对象的__proto__属性，指向父对象

   Object.setPrototypeOf(子对象,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，实现统一修改将要创建的所有子对象的父对象。

   ***在创建所有子对象之前设置。

   构造函数.prototype=父对象

   何时使用：批量修改未来所有子对象的父对象时

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

对象的克隆：在 Object 的原型对象中添加 clone 方法

1. constructor 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]
}
```

如何克隆原始类型的包装对象

3. 如何获得包装类型对象中的值：obj.valueOf

****toString vs valueOf 的差别:

返回值类型的差别：

1. toString 一定将所有内容转为字符串

2. valueOf 取出对象内部的值，不进行类型转换

用途的差别：

1. valueOf 专用于算数计算和关系运算

2. toString 专用于输出字符串

共同的缺点：无法获取 null 和 undefined 的值

```
Number=function(){
if(this==null){
        return 0;
    }else{
    return this.valueOf();
    }
}
```

3. 仅继承结构：最像 Java 中的 extends

何时使用继承结构：两种类型间形成继承关系

不要求两种类型的实例对象间有继承

如何继承结构：2 步：

1. 子类型构造函数开始位置，先调用父类型构造函数
2. 设置子类型的原型对象，继承父类型的原型对象。

```
function Student(sname,age){
    this.sname=sname;
    this.age=age;
}
//重写 toString 方法
Student.prototype.toString=function(){
    return "I'm "+this.sname+",I'm "+this.age;}
var lilei=new Student("Li Lei",18);
//var father={car:"BMW"};
//Student.prototype=father;
var hmm=new Student("Han Meimei",19);
//自动调用 toString，但因为被重写，所以优先用原型中的
document.write(lilei);
document.write("<br>");
document.write(hmm);
```

```
Object.prototype.clone=function(){
    //this-->表示要克隆的旧对象
    var newObj;
    if(!(this instanceof Number)
        &&!(this instanceof String)
            &&!(this instanceof Boolean)){
        //如果不是包装类型
        newObj=new this.constructor();
    }else{//否则，必须向构造函数中传值
```

```
                newObj=new
this.constructor(this.valueOf());}
            for(var prop in this){//遍历旧对象中每个属性
                if(this.hasOwnProperty(prop)){//只复制自有
属性
                    newObj[prop]=this[prop];
                }       } return newObj;
    }var arr=[1,2,3];     var newArr=arr.clone();
    console.log(newArr);
    function Student(sname,age){
        this.sname=sname;         this.age=age;       }
    Student.prototype.toString=function(){
        return "I'm "+this.sname+",I'm "+this.age;       }
    var lilei=new Student("Li Lei",18);
    var cloneLilei=lilei.clone();
    console.log(lilei==cloneLilei); //false
    //存在不同地方的两个对象
    console.log(cloneLilei);
    var str="Hello World";
    var newStr=str.clone();//new String(str).clone();
    console.log(newStr);
    console.log(newStr.length);
    /*var n=true;
    console.log(typeof(n.toString())); //String
    console.log(typeof(n.valueOf())); //?
    var obj={
        n:5,
        //重写了 toString 和 valueOf
        toString:function(){
            console.log("调用了 toString");
            return this.n.toString();    },
        valueOf:function(){
            console.log("调用了 valueOf");
            return this.n;    }   }*/
    //console.log(10-obj); //valueOf
    //console.log("10"+obj); //valueOf
    //console.log(10>obj); //valueOf
    //alert(obj); //toString
```

```
    //定义一个飞行物的构造函数 Flyer
    function Flyer(fname,speed){
    //      规定 2 个属性：fname,speed
        this.fname=fname;          this.speed=speed;       }

    //构造函数的原型中，定义公共方法 fly
    //      输出: fname 以 speed 公里/时速 飞行
    Flyer.prototype.fly=function(){
    console.log(this.fname+" 以 "+this.speed+" 公里/时速飞
行"); }
    var bird=new Flyer("麻雀",40);
    bird.fly();//"麻雀以 40 公里/时速飞行"

    //定义飞机类型构造函数 Plane
    function Plane(fname,speed,capacity){
    //      规定 3 个属性：fname,speed,capacity
        //使用当前调用 Flyer 构造函数
        //Flyer 中 this-->当前对象
        //同时传入必要的参数
        Flyer.call(this,fname,speed); //像 Java
        this.capacity=capacity;     }
    //让子类型原型继承父类型原型
    //-->子类型对象可使用父类型的共有方法。
Object.setPrototypeOf(Plane.prototype,Flyer.prototype);
    //飞机类型的原型中增加 fly 方法
    Plane.prototype.fly=function(){
    //     输出:fname 搭载 capacity 名乘客以 speed 公里/时速飞行
        console.log(this.fname+"搭载"+this.capacity
                +"名乘客以"+this.speed+"公里/时速飞行"); }
    var A380=new Plane("A380",1000,555);
    console.log(A380.fname);
    console.log(A380.speed);
    //Flyer.prototype.fly.call(A380);
    A380.fly();
    //"A380 搭载 555 名乘客以 1000 公里时速飞行"
</script>
```