# 代码块

在JS中，我们可以使用{}来为语句进行分组，可以将同一组的语句放到一对{}中

一对大括号中的语句，我们称为是一个代码块，代码块后边不用加;

JS中的代码块仅有分组的作用没有其他的作用，代码块中的内容对于代码块外是可见的

{

alert("hello");

console.log("你好");

document.write("哈哈哈哈哈哈");

}

# prompt()函数

可以在页面中弹出一个提示框，这个提示框会带有一个输入框

\* 需要指定一个字符串作为参数，这个字符串将会作为提示框中的提示文字显示

\* 该函数将会将用户输入的内容作为返回值返回，我们可以定义一个变量来接收用户输入的内容

注意：它返回的值的类型永远都是String

如果需要使返回的值为Number类型，则在函数前面加一个+，转换为Number

var num = +prompt("请输入一个数:");

# 流程控制语句

- JS中程序的执行顺序是按照自上到下一行一行执行的

- 我们可以通过流程控制语句，来改变程序执行的顺序，或者让某一段程序反复执行多次

- 流程控制语句的分类：

1.条件判断语句

2.条件分支语法

3.循环语句

## 条件判断语句（if语句）

### 语法一：if

if(条件表达式) {

语句...

}

执行流程：

if语句在执行时，会先对条件表达式进行求值判断，

如果判断结果为true，则执行if后的语句

如果判断结果为false，则不执行

- if语句只对紧随其后的那个语句起作用，如果希望if语句可以控制多条语句，则可以将语句放到一个代码块中，一般情况为了方便阅读，即使if后仅有一条语句，我们也要加{}

- 如果if后的条件表达式的结果不是布尔值，则将其转换为布尔值然后再判断

### 语法二： if...else...语句：

if(条件表达式){

语句...

}else{

语句...

}

执行流程：

if...else...语句执行时，会先对if后的条件表达式进行求值判断，

如果判断结果为true，则执行if后的语句

如果判断结果为false，则执行else后的语句

### 语法三： if...else if...else...语句

if(条件表达式){

语句...

}else if(条件表达式){

语句...

}else if(条件表达式){

语句...

}else if(条件表达式){

语句...

}else{

语句...

}

执行流程：

if...else if...else语句执行时,会从上到下依次对条件表达式进行求值判断，

如果遇到判断结果为true的，则执行当前if后的语句，语句执行完成后直接结束不再判断

如果判断结果为false时，则继续向下判断，直到找到为true的条件表达式为止

如果所有的条件表达式都为false，则执行最后的else

if...else if...else语句中只会有一个代码块会执行，一旦条件判断成立，则立即执行其后的代码块，执行完成后立即退出，不再判断其他条件

## 条件分支语句(switch语句)

### 语法

switch(条件表达式){

case 表达式:

语句...

break;

case 表达式:

语句...

break;

default:

语句...

break;

}

default写在最前面也行，写在最前面必须写break

- 执行流程：

switch...case...语句在执行时，会将switch后的条件表达式自上向下依次和case后的表达式进行全等比较，

如果全等则会执行当前case后的所有语句，可以在case的最后添加一个break关键字，来避免其他的case执行

如果不全等，则继续向下比较，直到找到全等的case为止，

如果所有的case都不符合要求，则执行default后的语句

switch...case...语句的功能和if...else...功能类似， switch能干的if也能干，if能干的switch也可以，他们是可以互相替换的，实际开发中习惯用哪个就用哪个，一般情况使用if比较多

当多个case后的语句是一样的时候，switch语句可以简化

switch(条件表达式){

case 表达式:

case 表达式:

case 表达式:

语句...

break;

default:

语句...

break;

}

### 练习：

对于成绩大于60分的，输出'合格'；低于60分的，输出'不合格'

解法一：

var score=prompt(“请输入成绩：”);

switch(parseInt(score / 10)){

case 10:

case 9:

case 8:

case 7:

case 6:

console.log("合格");

break;

default:

console.log("不合格");

break;

}

解法二：

switch(true){

case score >= 60:

console.log("合格");

break;

default:

console.log("不合格");

break;

}

# 循环语句

- 通过循环语句，我们可以使一些语句反复的执行多次

- 分类

- while语句

- do...while语句

- for语句

## while语句（while循环）

- 语法：

while(条件表达式){

语句...

}

- 执行流程：

while语句在执行时，会先对条件表达式进行求值判断

如果结果为true，则执行循环体，执行完毕以后，继续对条件表达式进行判断

如果为true，则继续执行循环体，以此类推

如果结果为false，则退出循环

创建一个执行10次的循环

1.初始化的表达式(初始化一个变量)

var i = 0;

2.条件表达式（循环执行的条件）

while(i < 10){

3.更新表达式(对初始化变量进行更新)

i++;

document.write(i + "<br />");

}

像这种将条件表达式写死为true，我们称为死循环

死循环就是会一直执行的循环，慎用

var n = 0;

while(true){

alert(n++);

}

当满足一定条件后，可以使用break来结束循环

while(true){

alert(n++);

if(n == 10){

break;

}

}

## do...while语句

语法：

do{

语句...

}while(条件表达式)

- 执行流程：

do...while语句在执行时，会先执行do后的循环体，然后再判断while后的条件表达式，

如果判断结果为false，则终止循环

如果判断结果为true，则继续执行循环体，执行完成以后继续判断，依此类推

它和while的区别就是，while是先判断后执行，而do...while是先执行后判断，只不过do...while可以保证循环体至少执行一次。，如果需要确保循环体至少执行一次，则使用do...while，否则使用while。

## for循环（for语句）

for循环和while循环本质上没有区别，只不过for循环将循环三个要件，统一写到一起

- 语法：

for(①初始化表达式;②条件表达式;④更新表达式){

③语句...

}

- 执行流程：

①执行初始化表达式，来初始化变量

②执行条件表达式，来判断是否执行循环体

如果条件表达式为false，则直接退出循环

如果条件表达式为true，则执行循环体

③执行循环体

④执行更新表达式

重复②

- 在for循环中初始化表达式只会执行一次

- for循环的功能和while循环的功能是重复，可以互相代替，实际开发中for循环用的会多一些

var i = 0;

for( ; i < 10 ; ){

document.write(i + "<br />");

i++;

} 这么一改就和while循环是一样的

如果for循环中将所有的表达式都省略，就写两个; 则此时它就是一个死循环会一直执行，或者条件表达式为true，也是一个死循环

for(;;){ for(;true;){

alert("hello"); alert("hello");

} }

## 嵌套的for循环

for(var i = 0 ; i < 5 ; i++){

//外层循环每执行1次，内层循环就要执行5次

for(var j = 0 ; j <5 ; j++){

document.write("\*&nbsp;&nbsp;&nbsp;");

}

}

当用嵌套的for循环绘制图形时，外层for循环控制图形的高度，内层for循环控制图形的宽度

//创建一个外层循环来控制图形的高度， i < 几，图形的高度就是几

for(var i = 0 ; i < 5 ; i++){

//创建一个内层循环来控制图形的宽度，循环执行几次图形的宽度就是多少

for(var j = 0 ; j < i+1 ; j++){

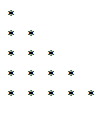
document.write("\*&nbsp;&nbsp;&nbsp;");

}

//输出一个换行

document.write("<br />");

}



关于外层循环控制高度，内层循环控制宽度的解释：

上面代码中，当外层循环每执行一次后，就会输出一个换行<br />，执行几次就换几行，所以外层循环控制高度；而当外层循环执行一次时，内层循环要执行一个完整的循环，由于没有换行<br />，图形\*默认是从左往右排列，所以内层循环控制宽度

# break和continue

## break

break关键字只可以用来退出switch语句和循环语句，不可以单独在if中使用，但如果在switch或循环中的if语句中可以使用，本质上还是对switch和循环起作用

for(var i=0 ; i<5 ; i++){

console.log(i);

if(i == 2){

break;

}

}

break可以用来立刻退出离他最近的循环语句

for(var i=0 ; i<5 ; i++){

console.log("外层循环："+i);

for(var j=0 ; j<5 ; j++){

console.log("----->内层循环："+j)

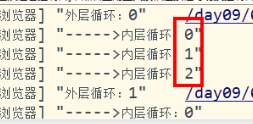
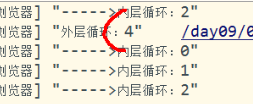
if(j == 2){

break;

}

}

}

图没有截取完整，但可以看出内层循环只到2没了，而外层循环没有受到影响，一直执行到循环完成

还可以为循环指定一个label来标识循环，使用break时，可以通过这个label来指定要结束的循环。语法：label名： label名可以随便起

hello: // hello:就是给外层循环指定的label

for(var i=0 ; i<5 ; i++){

console.log("外层循环："+i);

for(var j=0 ; j<5 ; j++){

console.log("----->内层循环："+j)

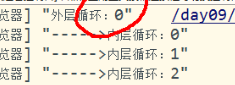
if(j == 2){

break hello;

}

}

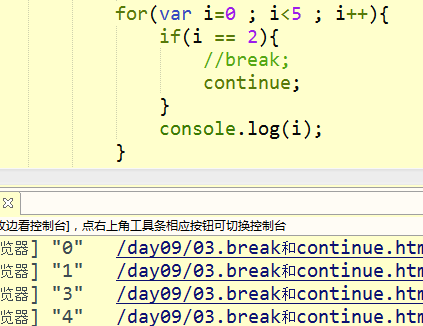
}



当j==2时，直接结束外层循环，所以后面的循环都没有了

## continue

continue可以用来跳过当次循环



执行完continue后直接跳到更新表达式，不再执行continue后的语句

continue也是只对离他最近的循环语句起作用，也可以为循环指定一个label来跳过指定循环的当次循环

# 计时器

console.time()可以用来开启一个计时器

\* 这个计时器可以用来记录程序的执行的时间

\* 该方法需要一个字符串作为参数，这个字符串将会作为计时器的标识

console.timeEnd()可以用来停止一个计时器

\* 该方法需要一个字符串作为参数，这个字符串是要停止计时器的标识，开始计时器的字符串和停止计时器的字符串要相同

例如：console.time(“test”)

程序

console.timeEnd(“test”)