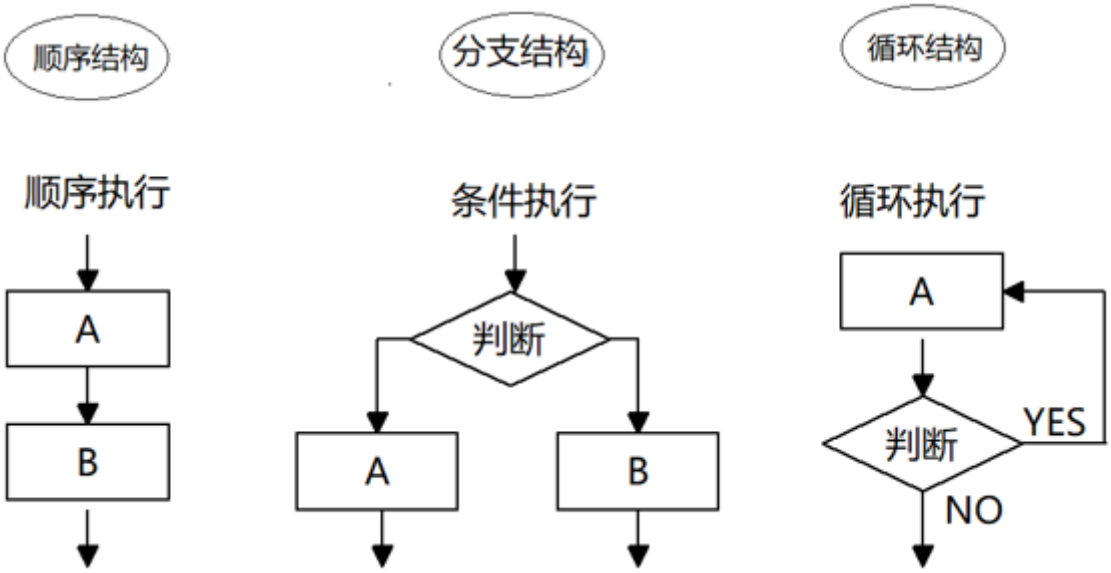
# 流程控制

## 1 、流程控制概念

在一个程序执行的过程中，各条代码的执行顺序对程序的结果是有直接影响的。很多时候我们要通过控制代码的执行顺序来实现我们要完成的功能。

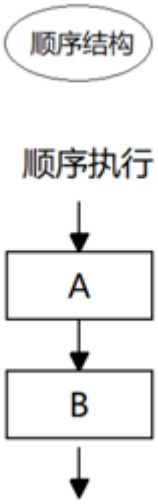
简单理解：**流程控制就是来控制代码按照一定结构顺序来执行**

流程控制主要有三种结构，分别是**顺序结构**、**分支结构**和**循环结构**，代表三种代码执行的顺序。




## 2 、顺序流程控制

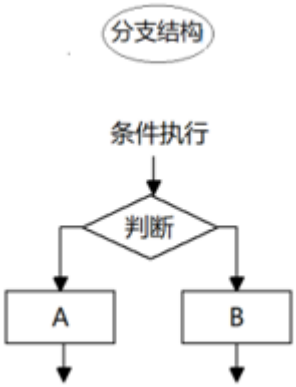
顺序结构是程序中最简单、最基本的流程控制，它没有特定的语法结构，程序会按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。




## 3、 分支流程控制

- 分支结构

由上到下执行代码的过程中，根据不同的条件，执行不同的路径代码（执行代码多选一的过程），从而得到不同的结果



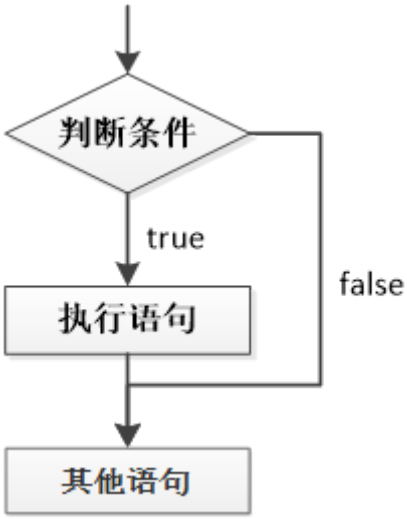

JS 语言提供了两种分支结构语句：if 语句、switch 语句

- if 语句
  - 语法结构

```
// 条件成立执行代码，否则什么也不做
if (条件表达式) {
    // 条件成立执行的代码语句
}
```

> 语句可以理解为一个行为，循环语句和分支语句就是典型的语句。一个程序由很多个语句组成，一般情况下，会分割成一个一个的语句。

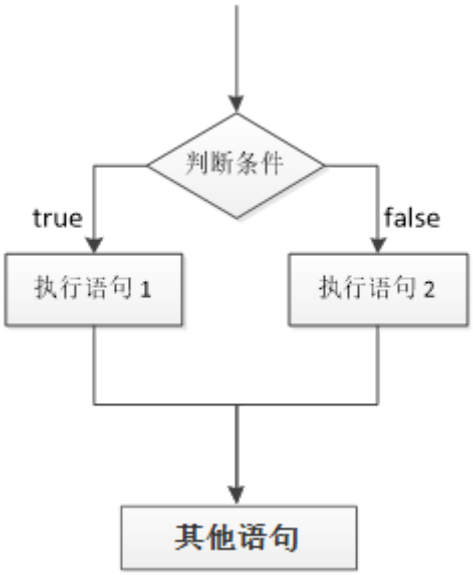
  - 执行流程




- if else语句（双分支语句）
  - 语法结构

```
// 条件成立  执行 if 里面代码，否则执行else 里面的代码
if (条件表达式) {
    // [如果] 条件成立执行的代码
} else {
    // [否则] 执行的代码
}
```

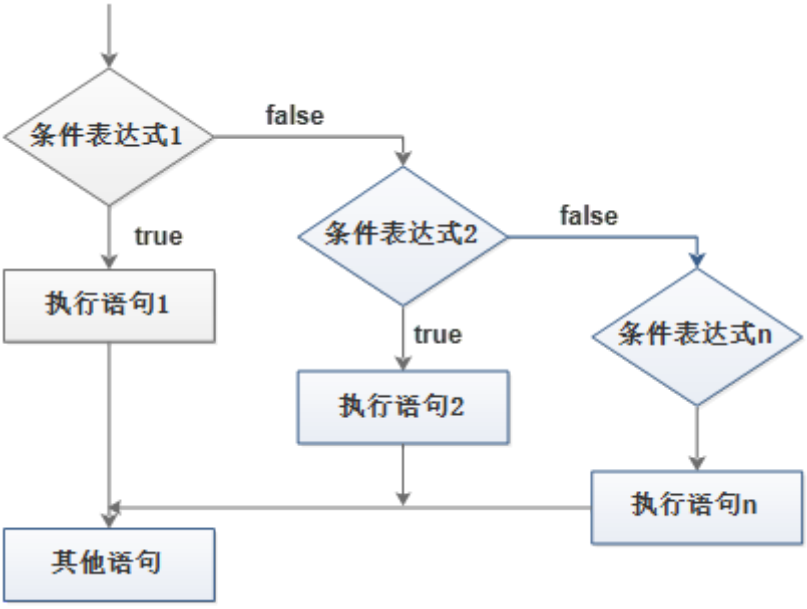
- 执行流程




- if else if 语句(多分支语句)
  - 语法结构

```
// 适合于检查多重条件。
if (条件表达式1) {
     语句1;
} else if (条件表达式2)  {
     语句2;
} else if (条件表达式3)  {
    语句3;
 ....
} else {
    // 上述条件都不成立执行此处代码
}
```

  - 执行逻辑




## 4、 三元表达式

- 语法结构

```
表达式1 ? 表达式2 : 表达式3;
```

- 执行思路
  - 如果表达式1为 true ，则返回表达式2的值，如果表达式1为 false，则返回表达式3的值
  - 简单理解： 就类似于 if else （双分支） 的简写

## 5、 switch分支流程控制

- 语法结构

  switch 语句也是多分支语句，它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的特定值的选项时，就可以使用 switch。

```
switch( 表达式 ){
    case value1:
        // 表达式 等于 value1 时要执行的代码
        break;
    case value2:
        // 表达式 等于 value2 时要执行的代码
        break;
    default:
        // 表达式 不等于任何一个 value 时要执行的代码
}
```

  - switch ： 开关 转换 ， case ： 小例子 选项
  - 关键字 switch 后面括号内可以是表达式或值， 通常是一个变量
  - 关键字 case , 后跟一个选项的表达式或值，后面跟一个冒号
  - switch 表达式的值会与结构中的 case 的值做比较
  - 如果存在匹配全等(===) ，则与该 case 关联的代码块会被执行，并在遇到 break 时停止，整个 switch 语句代码执行结束
  - 如果所有的 case 的值都和表达式的值不匹配，则执行 default 里的代码

    **注意： 执行case 里面的语句时，如果没有break，则继续执行下一个case里面的语句。**

- switch 语句和 if else if 语句的区别
  - 一般情况下，它们两个语句可以相互替换
  - switch...case 语句通常处理 case为比较确定值的情况， 而 if...else...语句更加灵活，常用于范围判断(大于、等于某个范围)
  - switch 语句进行条件判断后直接执行到程序的条件语句，效率更高。而if...else 语句有几种条件，就得判断多少次。
  - 当分支比较少时，if... else语句的执行效率比 switch语句高。
  - 当分支比较多时，switch语句的执行效率比较高，而且结构更清晰。

## 6、循环 - for循环

- 语法结构

```
for(初始化变量; 条件表达式; 操作表达式 ){
    //循环体
}
```

| 名称 | 作用 |
| --- | --- |
| 初始化变量 | 通常被用于初始化一个计数器，该表达式可以使用 var 关键字声明新的变量，这个变量帮我们来记录次数。 |
| 条件表达式 | 用于确定每一次循环是否能被执行。如果结果是 true 就继续循环，否则退出循环。 |
| 操作表达式 | 用于确定每一次循环是否能被执行。如果结果是 true 就继续循环，否则退出循环。 |

执行过程：

1. 初始化变量，初始化操作在整个 for 循环只会执行一次。

- 执行条件表达式，如果为true，则执行循环体语句，否则退出循环，循环结束。

1. 执行操作表达式，此时第一轮结束。
2. 第二轮开始，直接去执行条件表达式（不再初始化变量），如果为 true ，则去执行循环体语句，否则退出循环。
3. 继续执行操作表达式，第二轮结束。
4. 后续跟第二轮一致，直至条件表达式为假，结束整个 for 循环。

断点调试：

> 断点调试是指自己在程序的某一行设置一个断点，调试时，程序运行到这一行就会停住，然后你可以一步一步往下调试，调试过程中可以看各个变量当前的值，出错的话，调试到出错的代码行即显示错误，停下。断点调试可以帮助观察程序的运行过程

> 断点调试的流程：
> 1、浏览器中按 F12--> sources -->找到需要调试的文件-->在程序的某一行设置断点
> 2、Watch：监视，通过watch可以监视变量的值的变化，非常的常用。
> 3、摁下F11，程序单步执行，让程序一行一行的执行，这个时候，观察watch中变量的值的变化。

- for 循环重复相同的代码

  比如输出10句“媳妇我错了”

```
//  基本写法
for(var i = 1; i <= 10; i++){
    console.log('媳妇我错了~');
}
// 用户输入次数
var num = prompt('请输入次数:');
for ( var i = 1 ; i <= num; i++) {
    console.log('媳妇我错了~');
}
```

- for 循环重复不相同的代码

  例如，求输出1到100岁：

```
//  基本写法
for (var i = 1; i <= 100; i++) {
      console.log('这个人今年' + i + '岁了');
}
```

例如，求输出1到100岁，并提示出生、死亡

```
// for 里面是可以添加其他语句的
for (var i = 1; i <= 100; i++) {
 if (i == 1) {
    console.log('这个人今年1岁了， 它出生了');
 } else if (i == 100) {
    console.log('这个人今年100岁了，它死了');
  } else {
      console.log('这个人今年' + i + '岁了');
  }
}
```

for循环因为有了计数器的存在，还可以重复的执行某些操作，比如做一些算术运算。

## 7、循环-双重for循环

- 双重 for 循环概述

> 循环嵌套是指在一个循环语句中再定义一个循环语句的语法结构，例如在for循环语句中，可以再嵌套一个for 循环，这样的 for 循环语句我们称之为双重for循环。

- 双重 for 循环语法

```
for (外循环的初始; 外循环的条件; 外循环的操作表达式) {
    for (内循环的初始; 内循环的条件; 内循环的操作表达式) {
        需执行的代码;
   }
}
```

  - 内层循环可以看做外层循环的循环体语句
  - 内层循环执行的顺序也要遵循 for 循环的执行顺序
  - 外层循环执行一次，内层循环要执行全部次数
- 打印五行五列星星

```
var star = '';
for (var j = 1; j <= 5; j++) {
    for (var i = 1; i <= 5; i++) {
      star += '☆'
    }
    // 每次满 5个星星 就 加一次换行
    star += '\n'
}
console.log(star);
```

核心逻辑：

1.内层循环负责一行打印五个星星

2.外层循环负责打印五行

- for 循环小结
  - for 循环可以重复执行某些相同代码
  - for 循环可以重复执行些许不同的代码，因为我们有计数器
  - for 循环可以重复执行某些操作，比如算术运算符加法操作

- 随着需求增加，双重for循环可以做更多、更好看的效果
- 双重 for 循环，外层循环一次，内层 for 循环全部执行
- for 循环是循环条件和数字直接相关的循环

## 8、循环- while循环

while语句的语法结构如下：

```
while (条件表达式) {
    // 循环体代码
}
```

执行思路：

- 1 先执行条件表达式，如果结果为 true，则执行循环体代码；如果为 false，则退出循环，执行后面代码
- 2 执行循环体代码
- 3 循环体代码执行完毕后，程序会继续判断执行条件表达式，如条件仍为true，则会继续执行循环体，直到循环条件为 false 时，整个循环过程才会结束

注意：

- 使用 while 循环时一定要注意，它必须要有退出条件，否则会成为死循环

## 9、循环- do-while循环

do... while 语句的语法结构如下：

```
do {
    // 循环体代码 - 条件表达式为 true 时重复执行循环体代码
} while(条件表达式);
```

执行思路

- 1 先执行一次循环体代码
- 2 再执行条件表达式，如果结果为 true，则继续执行循环体代码，如果为 false，则退出循环，继续执行后面代码

  注意：先再执行循环体，再判断，do...while循环语句至少会执行一次循环体代码

## 10、 continue、break

continue 关键字用于立即跳出本次循环，继续下一次循环（本次循环体中 continue 之后的代码就会少执行一次）。

例如，吃5个包子，第3个有虫子，就扔掉第3个，继续吃第4个第5个包子，其代码实现如下：

```
for (var i = 1; i <= 5; i++) {
    if (i == 3) {
        console.log('这个包子有虫子，扔掉');
        continue; // 跳出本次循环，跳出的是第3次循环
     }
     console.log('我正在吃第' + i + '个包子呢');
}
```

运行结果：


break 关键字用于立即跳出整个循环（循环结束）。

例如，吃5个包子，吃到第3个发现里面有半个虫子，其余的不吃了，其代码实现如下：

```
for (var i = 1; i <= 5; i++) {
   if (i == 3) {
       break; // 直接退出整个for 循环，跳到整个for下面的语句
   }
   console.log('我正在吃第' + i + '个包子呢');
}
```

运行结果：