代码块

用`{}`包围起来的代码，就是代码块。

JS中的代码块，只具有**分组**的作用，没有其他的用途。

代码块中的内容，在外部是完全可见的。举例：

```
{
    var a = 2;
    alert("老胡");
    console.log("前端");
}

console.log("a = " + a);
```

打印结果：（可以看出，虽然变量 a 是定义在代码块中的，但是在外部依然可以访问）

```
前端
a = 2
```

流程控制语句

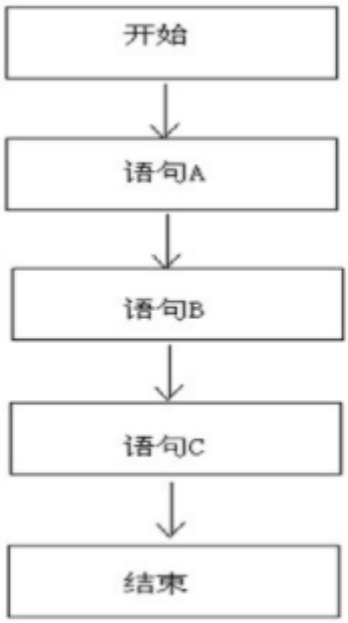
在一个程序执行的过程中，各条语句的执行顺序对程序的结果是有直接影响的。所以，我们必须清楚每条语句的执行流程。而且，很多时候我们要通过控制语句的执行顺序来实现我们要完成的功能。

流程控制语句分类

- 顺序结构
- 选择结构：if语句、switch语句
- 循环结构：while语句、for语句

顺序结构

按照代码的先后顺序，依次执行。结构图如下：



if语句

if语句有以下三种。

1、条件判断语句

> 条件成立才执行。如果条件不成立，那就什么都不做。

格式：

```
if (条件表达式) {
    // 条件为真时，做的事情

}
```

2、条件分支语句

格式1：

```
if (条件表达式) {
    // 条件为真时，做的事情

} else {
    // 条件为假时，做的事情

}
```

格式2：（多分支的if语句）

```
if (条件表达式1) {
    // 条件1为真时，做的事情

} else if (条件表达式2) {
    // 条件1不满足，条件2满足时，做的事情

} else if (条件表达式3) {
    // 条件1、2不满足，条件3满足时，做的事情

} else {
    // 条件1、2、3都不满足时，做的事情
}
```

以上所有的语句体中，只执行其中一个。

做个题目

```
根据BMI（身体质量指数）显示一个人的体型。
BMI指数，就是体重、身高的一个计算公式。公式是：
BMI =体重÷身高的平方

比如，老师的体重是81.6公斤，身高是1.71米。
那么老师的BMI就是  81.6 ÷ 1.712     等于 27.906022365856163

过轻：低于18.5
正常：18.5-24.99999999
过重：25-27.9999999
肥胖：28-32
非常肥胖，高于32

用JavaScript开发一个程序，让用户先输入自己的体重，然后输入自己的身高（弹出两次prompt框）。
计算它的BMI，根据上表，弹出用户的身体情况。比如“过轻”、“正常”、“过重”、“肥胖”、“非常肥胖”。
```

答案：

写法1：

```
//第一步，输入身高和体重
var height = parseFloat(prompt("请输入身高，单位是米"));
var weight = parseFloat(prompt("请输入体重，单位是公斤"));
//第二步，计算BMI指数
var BMI = weight / Math.pow(height, 2);
//第三步，if语句来判断。注意跳楼现象
if (BMI < 18.5) {
    alert("偏瘦");
} else if (BMI < 25) {
    alert("正常");
} else if (BMI < 28) {
    alert("过重");
} else if (BMI <= 32) {
    alert("肥胖");
} else {
    alert("非常肥胖");
}
```

写法2：

```
//第一步，输入身高和体重
var height = parseFloat(prompt("请输入身高，单位是米"));
var weight = parseFloat(prompt("请输入体重，单位是公斤"));
//第二步，计算BMI指数
var BMI = weight / Math.pow(height, 2);
//第三步，if语句来判断。注意跳楼现象
if (BMI > 32) {
    alert("非常肥胖");
} else if (BMI >= 28) {
    alert("肥胖");
} else if (BMI >= 25) {
    alert("过重");
} else if (BMI >= 18.5) {
    alert("正常")
} else {
    alert("偏瘦");
}
```

switch语句（条件分支语句）

switch语句也叫条件分支语句。

格式：

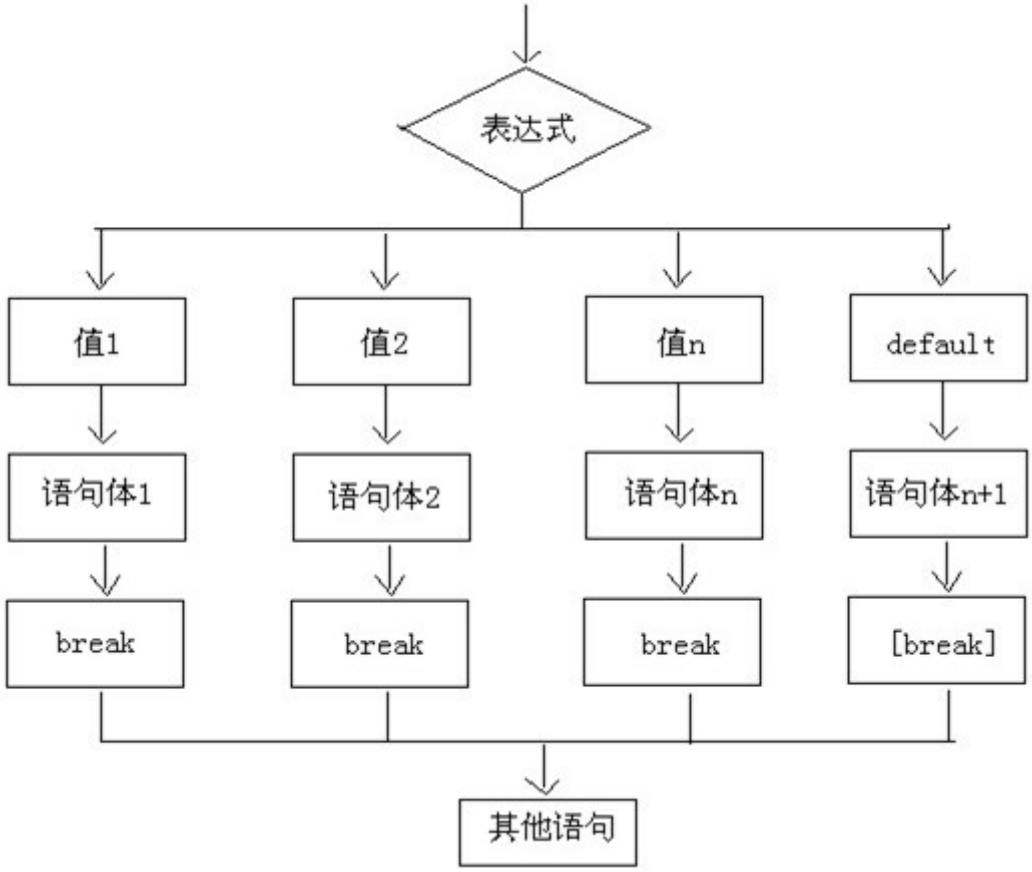
```
switch(表达式) {
    case 值1:
        语句体1;
        break;

    case 值2:
        语句体2;
        break;

    ...
    ...

    default:
        语句体 n+1;
        break;
}
```

注意：

- switch 可以理解为“开关、转换”。case 可以理解为“案例、选项”。
- switch 后面的**括号里**可以是**表达式**或者**值**，通常是一个**变量**（通常做法是：先把表达式或者值存放到变量中）。
- JS 是属于弱类型语言，case 后面的 `值1`、`值2` 可以是 `'a'`、`6`、`true` 等任意数据类型的值，也可以是表达式。

switch语句的执行流程

流程图如下：



执行流程如下：

（1）首先，计算出表达式的值，和case依次比较，一旦有对应的值，就会执行相应的语句，在执行的过程中，遇到break就会结束。

（2）然后，如果所有的case都和表达式的值不匹配，就会执行default语句体部分。

switch 语句的结束条件【非常重要】

- 情况a：遇到break就结束，而不是遇到default就结束。（因为break在此处的作用就是退出switch语句）
- 情况b：执行到程序的末尾就结束。

我们来看下面的两个例子就明白了。

case穿透的问题

switch 语句中的`break`可以省略，但一般不建议（对于新手而言）。否则结果可能不是你想要的，会出现一个现象：**case穿透。**

举例1：（case穿透的情况）

```
    var num = 4;

    //switch判断语句
    switch (num) {
        case 1:
            console.log("星期一");
            break;
        case 2:
            console.log("星期二");
            break;
        case 3:
```

```
            console.log("星期三");
            break;
        case 4:
            console.log("星期四");
        //break;
        case 5:
            console.log("星期五");
        //break;
        case 6:
            console.log("星期六");
            break;
        case 7:
            console.log("星期日");
            break;
        default:
            console.log("你输入的数据有误");
            break;
    }

```

上方代码的运行结果，可能会令你感到意外：

```
星期四
星期五
星期六
```

上方代码的解释：因为在case 4和case 5中都没有break，那语句走到case 6的break才会停止。

举例2：

```
    //switch判断语句
    var number = 5;

    switch (number) {
        default:
            console.log("我是defaul语句");
            // break;
        case (2):
            console.log("第二个呵呵:" + number);
            //break;
        case (3):
            console.log("第三个呵呵:" + number);
            break;
        case (4):
            console.log("第四个呵呵:" + number);
            break;
    }

```

上方代码的运行结果，你也许会意外：

```
我是defaul语句
第二个呵呵:5
第三个呵呵:5
```

上方代码的解释：代码走到 default 时，因为没有遇到 break，所以会继续往下走，直到遇见 break 或者走到程序的末尾。 从这个例子可以看出：switch 语句的结束与 default 的顺序无关。

switch 语句的实战举例：替换 if 语句

我们实战开发中，经常需要根据接口的返回码 retCode ，来让前端做不同的展示。

这种场景是业务开发中经常出现的，请一定要掌握。然而，很多人估计会这么写：

写法1：（不推荐。这种写法太挫了）

```
let retCode = 1003; // 返回码 retCode 的值可能有很多种情况

if (retCode == 0) {
    alert('接口联调成功');
} else if (retCode == 101) {
    alert('活动不存在');
} else if (retCode == 103) {
    alert('活动未开始');
} else if (retCode == 104) {
    alert('活动已结束');
} else if (retCode == 1001) {
    alert('参数错误');
} else if (retCode == 1002) {
    alert('接口频率限制');
} else if (retCode == 1003) {
    alert('未登录');
} else if (retCode == 1004) {
    alert('（风控用户）提示 活动太火爆啦~军万马都在挤，请稍后再试');
} else {
    // 其他异常返回码
    alert('系统君失联了，请稍候再试');
}
```

如果你是按照上面的 `if else` 的方式来写各种条件判断，说明你的代码水平太初级了，会被人喷的，千万不要这么写。那要怎么改进呢？继续往下看。

写法2：（推荐。通过 return 的方式，将上面的写法进行改进）

```

let retCode = 1003; // 返回码 retCode 的值可能有很多种情况
handleRetCode(retCode);

// 方法：根据接口不同的返回码，处理前端不同的显示状态
function handleRetCode(retCode) {
    if (retCode == 0) {
        alert('接口联调成功');
        return;
    }

    if (retCode == 101) {
        alert('活动不存在');
        return;
    }

    if (retCode == 103) {
        alert('活动未开始');
```

```
        return;
    }

    if (retCode == 104) {
        alert('活动已结束');
        return;
    }

    if (retCode == 1001) {
        alert('参数错误');
        return;
    }

    if (retCode == 1002) {
        alert('接口频率限制');
        return;
    }

    if (retCode == 1003) {
        alert('未登录');
        return;
    }

    if (retCode == 1004) {
        alert('（风控用户）提示 活动太火爆啦~军万马都在挤，请稍后再试');
        return;
    }

    // 其他异常返回码
    alert('系统君失联了，请稍候再试');
    return;
}

```

上面的写法2，是比较推荐的写法：直接通过 return 的方式，让 function 里的代码不再继续往下走，这就达到目的了。对了，因为要用到 return ，所以需要单独封装到一个 function 里面。

如果你以后看到有前端小白采用的是**写法1**，请一定要把**写法2**传授给他：不需要那么多的 if else，直接用 return 返回就行了。

写法3：（推荐。将 if else 改为 switch ）

```
let retCode = 1003; // 返回码 retCode 的值可能有很多种情况

switch (retCode) {
    case 0:
        alert('接口联调成功');
        break;
    case 101:
        alert('活动不存在');
        break;

    case 103:
        alert('活动未开始');
        break;

    case 104:
```

```
        alert('活动已结束');
        break;

    case 1001:
        alert('参数错误');
        break;

    case 1002:
        alert('接口频率限制');
        break;

    case 1003:
        alert('未登录');
        break;

    case 1004:
        alert('（风控用户）提示 活动太火爆啦~军万马都在挤，请稍后再试');
        break;

    // 其他异常返回码
    default:
        alert('系统君失联了，请稍候再试');
        break;
}
```

在实战开发中，方式3是非常推荐的写法，甚至比方式2还要好。我们尽量不要写太多的 if 语句，避免代码嵌套过深。

switch 语句的优雅写法：适时地去掉 break

我们先来看看下面这段代码：（不推荐）

```
let day = 2;

switch (day) {
    case 1:
        console.log('work');
        break;

    case 2:
        console.log('work');
        break;

    case 3:
        console.log('work');
        break;

    case 4:
        console.log('work');
        break;

    case 5:
        console.log('work');
        break;

    case 6:
        console.log('relax');
```

```
        break;

    case 7:
        console.log('relax');
        break;

    default:
        break;
}
```

上面的代码，咋一看，好像没啥毛病。但你有没有发现，重复代码太多了？

实战开发中，凡是有重复的地方，我们都必须要想办法简化。写代码就是在不断重构的过程。

上面的代码，可以改进如下：（推荐，非常优雅）

```
let day = 2;

switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        console.log('work');
        break; // 在这里放一个 break

    case 6:
    case 7:
        console.log('relax');
        break; // 在这里放一个 break

    default:
        break;
}
}
```

你没看错，就是上面的这种写法，能达到同样的效果，非常优雅。

小白可能认为这样的写法可读性不强，所以说他是小白。我可以明确告诉你，改进后的这种写法，才是最优雅的、最简洁、可读性最好的。