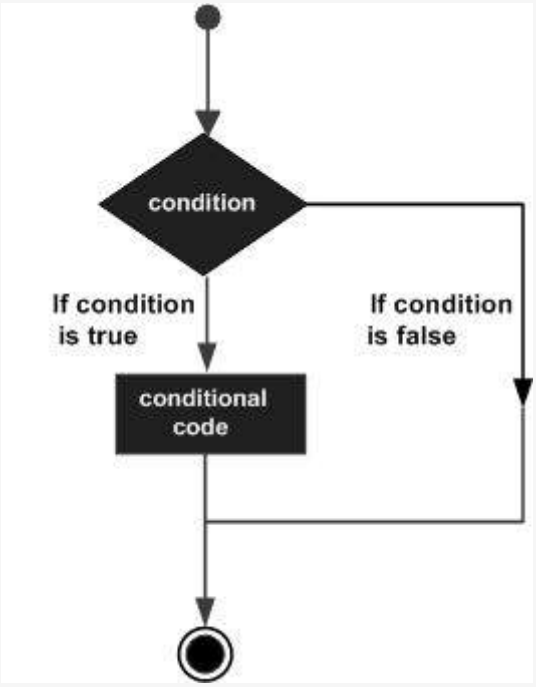
TypeScript 运算符 TypeScript 循环

TypeScript 条件语句

条件语句用于基于不同的条件来执行不同的动作。

TypeScript 条件语句是通过一条或多条语句的执行结果（True 或 False）来决定执行的代码块。

可以通过下图来简单了解条件语句的执行过程:



条件语句

通常在写代码时，您总是需要为不同的决定来执行不同的动作。您可以在代码中使用条件语句来完成该任务。

在 TypeScript 中，我们可使用以下条件语句：

- **if 语句** - 只有当指定条件为 true 时，使用该语句来执行代码
- **if...else 语句** - 当条件为 true 时执行代码，当条件为 false 时执行其他代码
- **if...else if....else 语句**- 使用该语句来选择多个代码块之一来执行
- **switch 语句** - 使用该语句来选择多个代码块之一来执行

if 语句

TypeScript if 语句由一个布尔表达式后跟一个或多个语句组成。

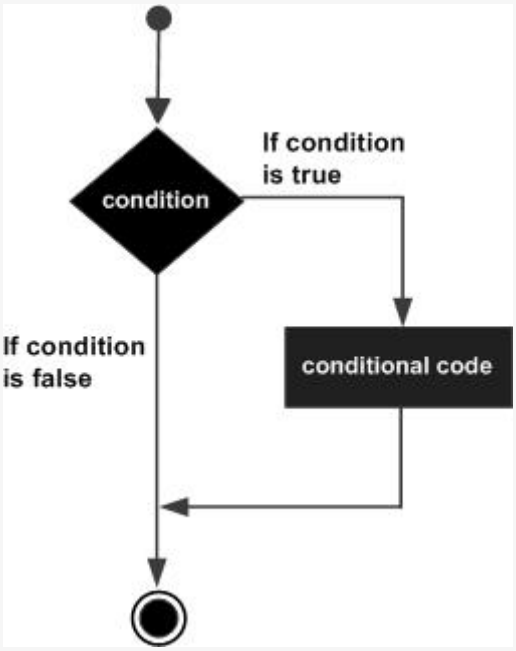
语法

语法格式如下所示：

```
if(boolean_expression){
    # 在布尔表达式 boolean_expression 为 true 执行
}
```

如果布尔表达式 boolean_expression为 true，则 if 语句内的代码块将被执行。如果布尔表达式为 false，则 if 语句结束后的第一组代码（闭括号后）将被执行。

流程图



实例

```
var num:number = 5
if (num > 0) {
console.log("数字是正数")
}
```

编译以上代码得到如下 JavaScript 代码：

```
var num = 5;
if (num > 0) {
console.log("数字是正数");
}
```

执行以上 JavaScript 代码，输出结果为：

```
数字是正数
```

if...else 语句

一个 if 语句后可跟一个可选的 else 语句，else 语句在布尔表达式为 false 时执行。

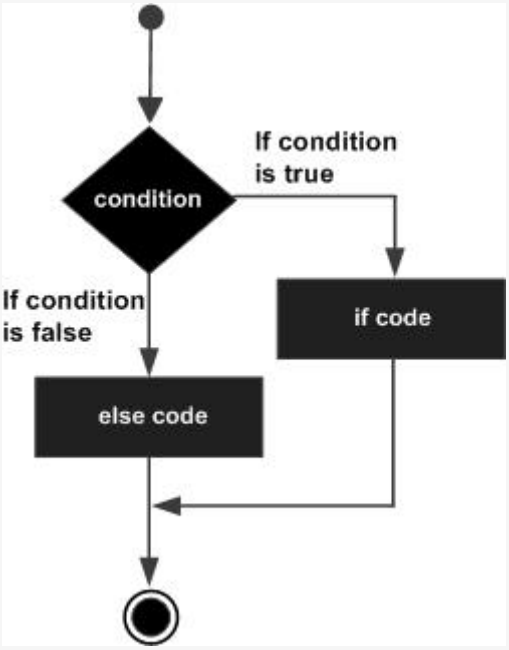
语法

语法格式如下所示：

```
if(boolean_expression){
   # 在布尔表达式 boolean_expression 为 true 执行
}else{
   # 在布尔表达式 boolean_expression 为 false 执行
}
```

如果布尔表达式 boolean_expression 为 true，则执行 if 块内的代码。如果布尔表达式为 false，则执行 else 块内的代码。

流程图



实例

TypeScript

```
var num:number = 12;
if (num % 2==0) {
console.log("偶数");
} else {
console.log("奇数");
}
```

编译以上代码得到如下 JavaScript 代码：

JavaScript

```
var num = 12;
if (num % 2 == 0) {
console.log("偶数");
}
else {
```

```
console.log("奇数");
}
```

执行以上 JavaScript 代码，输出结果为：

```
偶数
```

---

if...else if....else 语句

if...else if....else 语句在执行多个判断条件的时候很有用。

语法

语法格式如下所示：

```
if(boolean_expression 1){
    # 在布尔表达式 boolean_expression 1 为 true 执行
}
else if( boolean_expression 2){
    # 在布尔表达式 boolean_expression 2 为 true 执行
}
else if(( boolean_expression 3){
    # 在布尔表达式 boolean_expression 3 为 true 执行
}
else{
    # 布尔表达式的条件都为 false 时执行
}
```

需要注意以下几点：

- 一个 **if** 判断语句可以有 0 或 1 个 **else** 语句，她必需在 **else..if** 语句后面。
- 一个 **if** 判断语句可以有 0 或多个 **else..if**，这些语句必需在 **else** 之前。
- 一旦执行了 **else..if** 内的代码，后面的 **else..if** 或 **else** 将不再执行。

实例

TypeScript

```
var num:number = 2
if(num > 0) {
console.log(num+" 是正数")
} else if(num < 0) {
console.log(num+" 是负数")
} else {
console.log(num+" 不是正数也不是负数")
}
```

编译以上代码得到如下 JavaScript 代码：

JavaScript

```
var num = 2;
if (num > 0) {
console.log(num + " 是正数");
}
else if (num < 0) {
console.log(num + " 是负数");
}
else {
console.log(num + " 不是正数也不是负数");
}
```

执行以上 JavaScript 代码，输出结果为：

```
2 是正数
```

switch...case 语句

一个 **switch** 语句允许测试一个变量等于多个值时的情况。每个值称为一个 case，且被测试的变量会对每个 **switch case** 进行检查。

switch 语句的语法：

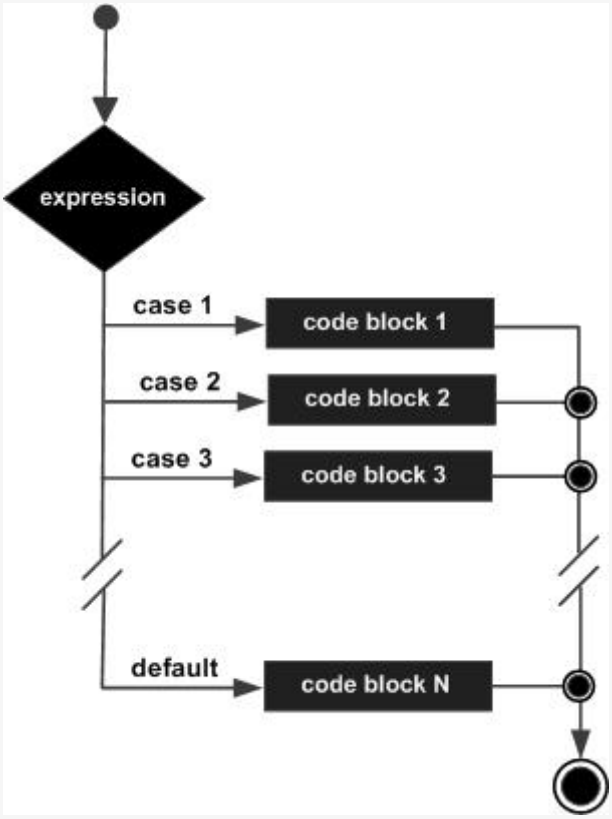
```
switch(expression){
case constant-expression :
statement(s);
break; /* 可选的 */
case constant-expression :
statement(s);
break; /* 可选的 */
/* 您可以有任意数量的 case 语句 */
default : /* 可选的 */
statement(s);
}
```

switch 语句必须遵循下面的规则：

- **switch** 语句中的 **expression** 是一个常量表达式，必须是一个整型或枚举类型。
- 在一个 switch 中可以有任意数量的 case 语句。每个 case 后跟一个要比较的值和一个冒号。
- case 的 **constant-expression** 必须与 switch 中的变量具有相同的数据类型，且必须是一个常量或字面量。
- 当被测试的变量等于 case 中的常量时，case 后跟的语句将被执行，直到遇到 **break** 语句为止。
- 当遇到 **break** 语句时，switch 终止，控制流将跳转到 switch 语句后的下一行。
- 不是每一个 case 都需要包含 **break**。如果 case 语句不包含 **break**，控制流将会 *继续* 后续的 case，直到遇到 break 为止。

- 一个 **switch** 语句可以有一个可选的 **default** case，出现在 switch 的结尾。default case 可用于在上面所有 case 都不为真时执行一个任务。default case 中的 **break** 语句不是必需的。

流程图



实例

TypeScript

```
var grade:string = "A";
switch(grade) {
case "A": {
console.log("优");
break;
}
case "B": {
console.log("良");
break;
}
case "C": {
console.log("及格");
break;
}
case "D": {
console.log("不及格");
break;
}
default: {
console.log("非法输入");
break;
```

```
}
}
```

编译以上代码得到如下 JavaScript 代码：

JavaScript

```
var grade = "A";
switch (grade) {
case "A": {
console.log("优");
break;
}
case "B": {
console.log("良");
break;
}
case "C": {
console.log("及格");
break;
}
case "D": {
console.log("不及格");
break;
}
default: {
console.log("非法输入");
break;
}
}
```

执行以上 JavaScript 代码，输出结果为：

```
优
```

← TypeScript 运算符 TypeScript 循环 →