

Go流程控制——条件语句

目录：

- 1. Go语言流程控制概述
- 2. if条件判断语句
- 3. switch分支语句

一、Go 语言流程控制

（一）、概述

- 1、流程控制是每种编程语言控制逻辑走向和执行次序的重要部分，流程控制是一门语言的经脉；
- 2、流程控制有条件判断语句、条件分支语句及循环语句；
- 3、Go语言的基本流程控制语句包括：
 - if条件判断语句
 - switch分支语句
 - for循环语句
 - goto跳转语句及break和continue循环控制语句

（二）、Go 语言提供了以下几种条件判断、分支语句

语句	描述
if 语句	if 语句 由一个布尔表达式后紧跟一个或多个语句组成。
if...else 语句	if 语句 后可以使用可选的 else 语句, else 语句中的表达式在 if 语句中的表达式为 false 时执行。
if 嵌套语句	你可以在 if 或 else if 语句中嵌入一个或多个 if 或 else
switch 语句	switch 语句用于基于不同条件执行不同动作。
select 语句	select 语句类似于 switch 语句，但是select会随机执行一个case，如果没有case可运行，它将阻塞，直到有case可运行。

(三)、Go 语言提供了以下几种循环语句

循环类型	描述
for 循环	重复执行语句块
循环嵌套	在 for 循环中嵌套一个或多个 for 循环

(四)、GO 语言支持以下几种循环控制语句

控制语句	描述
break 语句	经常用于中断当前 for 循环或跳出 switch 语句
continue 语句	跳过当前循环的剩余语句，然后继续进行下一轮循环
goto 语句	将控制转移到被标记的语句。

二、if条件判断语句

(一)、语法结构

1、Go 编程语言中 if 语句的语法如下：

```
if 布尔表达式 {  
    /* 在布尔表达式为 true 时执行 */  
}
```

- if 在布尔表达式为 true 时，其后紧跟的语句块执行，如果为 false 则不执行。

2、Go 编程语言中 if...else 语句的语法如下：

```
if 布尔表达式 {  
    /* 在布尔表达式为 true 时执行 */  
} else {  
    /* 在布尔表达式为 false 时执行 */  
}
```

- if 在布尔表达式为 true 时，其后紧跟的语句块执行，如果为 false 则执行

else 语句块。

3、Go 编程语言中 if...else if ... else 语句的语法如下：

```
if 布尔表达式 {  
    /* 在布尔表达式为 true 时执行 */  
} else if {  
    /* 在布尔表达式为 true 时执行 */  
} else {  
    /* 在布尔表达式为 false 时执行 */  
}
```

- if 在布尔表达式为 true 时，其后紧跟的语句块执行，如果为 false 则执行 else 语句块。

（二）、if语句中的注意细节

- 1、不需使用括号将条件包含起来；
- 2、大括号{}必须存在，即使只有一行语句；
- 3、左括号必须在if或else的同一行；
- 4、在if之后，条件语句之前，可以添加变量初始化语句，使用"; "进行分隔。

（三）、案例

1、用 if 语句判断数据奇数偶数

```
package main
```

```
import "fmt"
```

```
func main() {  
    num := 20  
    if num%2 == 0 {  
        fmt.Println(num, "偶数")  
    } else {  
        fmt.Println(num, "奇数")  
    }  
}
```

2、判断学生平均成绩。有优、良、中等、及格、不及格等五档。

```

package main

import "fmt"

func main() {
    score := 88
    if score >= 90 {
        fmt.Println("优秀")
    } else if score >= 80 {
        fmt.Println("良好")
    } else if score >= 70 {
        fmt.Println("中等")
    } else if score >= 60 {
        fmt.Println("及格")
    } else if score < 60 {
        fmt.Println("不及格")
    }
}

```

(三)、if语句特殊写法

- 1、if语句还有一个变体。它的语法是：

```

    if statement; condition {
        //代码块
    }

```

- 2、案例：判断一个数是奇数还是偶数？

```

package main

import "fmt"

func main() {
    if num := 10; num%2 == 0 {
        fmt.Println(num, "偶数")
    } else {
        fmt.Println(num, "奇数")
    }
}

```

```
}
```

- 需要注意的是，num的定义在if里，那么只能够在该if..else语句块中使用，否则编译器会报错的。

三、if嵌套语句

可以在 if 或 else if 语句中嵌入一个或多个 if 或 else if 语句。

(一)、语法结构

1、Go 编程语言中 if...else 语句的语法如下：

```
if 布尔表达式 1 {  
    /* 在布尔表达式 1 为 true 时执行 */  
    if 布尔表达式 2 {  
        /* 在布尔表达式 2 为 true 时执行 */  
    }  
}
```

- 可以以同样的方式在 if 语句中嵌套 else if...else 语句

(二)、案例：判断学生平均成绩。有优、良、中等、及格、不及格等五档。

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    if score := 98; score >= 60 {
```

```
        if score >= 70 {
```

```
            if score >= 80 {
```

```
                if score >= 90 {
```

```
                    fmt.Println("优")
```

```
                } else {
```

```
                    fmt.Println("良")
```

```
                }
```

```
            } else {
```

```
                fmt.Println("中等")
```

```
            }
```

```
        } else {
```

```
        fmt.Println("及格")
    }
} else {
    fmt.Println("不及格")
}
}
```

四、switch分支语句

（一）、语法结构

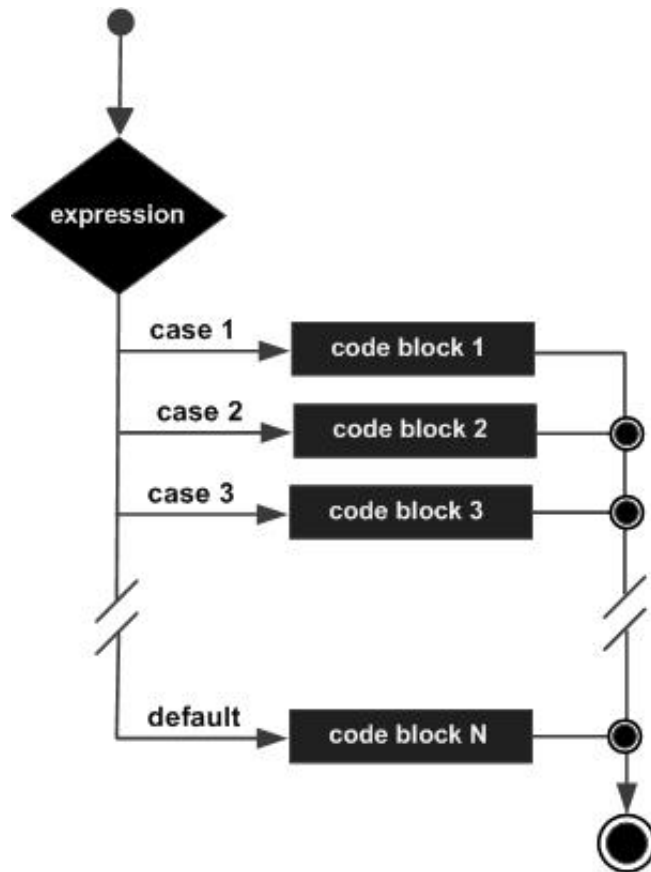
Go 编程语言中 switch 语句的语法如下：

```
switch var1 {
    case val1:
        ...
    case val2:
        ...
    default:
        ...
}
```

（二）、switch语句中的注意细节

- 1、switch 语句执行的过程自上而下，直到找到case匹配项，匹配项中无需使用break，因为Go语言中的switch默认给每个case自带break，因此匹配成功后不会向下执行其他的case分支，而是跳出整个switch。
- 2、变量 var1 可以是任何类型，而 val1 和 val2 则可以是同类型的任意值。类型不被局限于常量或整数，但必须是相同类型或最终结果为相同类型的表达式。
- 3、case后的值不能重复。
- 4、可以同时测试多个符合条件的值，也就是说case后可以有多个值，这些值之间使用逗号分割，例如：case val1, val2, val3。
- 5、Go语言中switch后的表达式可以省略，那么默认是switch true。
- 6、Go语言中的switch case因为自带break，所以匹配某个case后不会自动向下执行其他case，如需贯通后续的case，可以添加fallthrough（中文含义是：贯穿），强制执行后面的case分支。
- 7、fallthrough必须放在case分支的最后一行。如果它出现在中间的某个地方，

编译器就会抛出错误（fallthrough statement out of place, 含义是fallthrough不在合适的位置）。



(三)、案例：

1、判断学生平均成绩。有优、良、中等、及格、不及格等五档。

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    /* 定义局部变量 */
```

```
    grade := ""
```

```
    score := 78.5
```

```
    //思考： 以下代码逻辑错误在哪里？
```

```
    //switch {
```

```
    //case score < 60:
```

```
// grade = "E"  
//case score >= 60:  
// grade = "D"  
//case score >= 70:  
// grade = "C"  
//case score >= 80:  
// grade = "B"  
//case score >= 90:  
// grade = "A"  
//}
```

switch { //switch后面省略不写，默认相当于：switch true

case score >= 90:

grade = "A"

case score >= 80:

grade = "B"

case score >= 70:

grade = "C"

case score >= 60:

grade = "D"

default:

grade = "E"

}

fmt.Printf("你的等级是: %s\n", grade)

fmt.Print("最终评价是: ")

switch grade {

case "A":

fmt.Printf("优秀!\n")

case "B":

fmt.Printf("良好\n")

case "C":

fmt.Printf("中等\n")

case "D":

fmt.Printf("及格\n")


```

default:
    fmt.Printf("差\n")
}
}

```

2、判断某年某月的天数

```

package main
import "fmt"
func main() {
    /* 定义局部变量:年、月、日 */

    year := 2008
    month := 2
    days := 0
    switch month {
    case 1, 3, 5, 7, 8, 10, 12:
        days = 31
    case 4, 6, 9, 11:
        days = 30
    case 2:
        if (year%4 == 0 && year%100 != 0) || year%400 == 0 {
            days = 29
        } else {
            days = 28
        }
    default:
        days = -1
    }

    fmt.Printf("%d 年 %d 月的天数为: %d\n", year, month, days)
}

```

(四) 、Type Switch

- switch 语句还可以被用于 type-switch 来判断某个 interface 变量中实际

存储的变量类型。

1、语法结构：

```
switch x.(type){
    case type:
        statement(s);
    case type:
        statement(s);
    /* 你可以定义任意个数的case */
    default: /* 可选 */
        statement(s);
}
```

2、示例代码：

```
package main
import "fmt"

func main() {
    var x interface{}
    switch i := x.(type) {
        case nil:
            fmt.Printf(" x 的类型 :%T",i)
        case int:
            fmt.Printf("x 是 int 型")
        case float64:
            fmt.Printf("x 是 float64 型")
        case func(int) float64:
            fmt.Printf("x 是 func(int) 型")
        case bool, string:
            fmt.Printf("x 是 bool 或 string 型" )
        default:
            fmt.Printf("未知型")
    }
}

结果
x 的类型 :<nil>
```

五、select 语句（后续讲解）

- select 语句类似于 switch 语句，但是select会随机执行一个可运行的

case。如果没有case可运行，它将阻塞，直到有case可运行。

(一)、示例代码：

```
package main
import "fmt"

func main() {
    var c1, c2, c3 chan int
    var i1, i2 int
    select {
        case i1 = <-c1:
            fmt.Printf("received ", i1, " from c1\n")
        case c2 <- i2:
            fmt.Printf("sent ", i2, " to c2\n")
        case i3, ok := (<-c3): // same as: i3, ok := <-c3
            if ok {
                fmt.Printf("received ", i3, " from c3\n")
            } else {
                fmt.Printf("c3 is closed\n")
            }
        default:
            fmt.Printf("no communication\n")
    }
}
```

运行结果：

no communication

- 每个case都必须是一个通信
- 所有channel表达式都会被求值
- 所有被发送的表达式都会被求值
- 如果任意某个通信可以进行，它就执行；其他被忽略。
- 如果有多个case都可以运行，Select会随机公平地选出一个执行。其他不会执行。
- 否则：

如果有default子句，则执行该语句。

如果没有default字句，select将阻塞，直到某个通信可以运行；Go不会重新对channel或值进行求值。

