Go语言条件语句

条件语句是用来判断给定的条件是否满足(表达式值是否为`true`或者`false`)，并根据判断的结果(真或假)决定执行的语句，go语言中的条件语句也是这样的。

Go 语言提供了以下几种条件判断语句：

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 语句中的表达式在布尔表达式为 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 if** 语句。
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会随机执行一个可运行的case。如果没有case可运行，它将阻塞，直到有case可运行。

> 注意：Go 没有三目运算符，所以不支持 **?:** 形式的条件判断。

Go 语言 if 语句

if 语句由布尔表达式后紧跟一个或多个语句组成。**if语句**和其他语言中的类似，都是根据给定的条件表达式运算结果来，判断执行流程。

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。

> 注意：在go语言中 布尔表达式不用使用括号。

• 可省略条件表达式括号。
• 初始化语句，可定义代码块局部变量。
• 代码块左括号必须在条件表达式尾部。

示例：

```
package main

import "fmt"

func main() {
   /* 定义局部变量 */
```

```
   var a int = 10

   /* 使用 if 语句判断布尔表达式 */
   if a < 20 {
       /* 如果条件为 true 则执行以下语句 */
       fmt.Printf("a 小于 20\n" )
   }
   fmt.Printf("a 的值为 : %d\n", a)
}
#结果
a 小于 20
a 的值为 : 10
```

Go 语言 if...else 语句

if 语句 后可以使用可选的 else 语句, else 语句中的表达式在布尔表达式为 false 时执行。

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。

示例：

```
package main

import "fmt"

func main() {
   /* 局部变量定义 */
   var a int = 100;

   /* 判断布尔表达式 */
   if a < 20 {
       /* 如果条件为 true 则执行以下语句 */
       fmt.Printf("a 小于 20\n" );
   } else {
       /* 如果条件为 false 则执行以下语句 */
       fmt.Printf("a 不小于 20\n" );
   }
   fmt.Printf("a 的值为 : %d\n", a);

}
#结果
a 不小于 20
a 的值为 : 100
```

Go 语言 if 语句嵌套

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

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 */
   }
}

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

你可以以同样的方式在 if 语句中嵌套 **else if...else** 语句。

示例：

```
package main

import "fmt"

func main() {
   /* 定义局部变量 */
   var a int = 100
   var b int = 200

   /* 判断条件 */
   if a == 100 {
       /* if 条件语句为 true 执行 */
       if b == 200 {
          /* if 条件语句为 true 执行 */
          fmt.Printf("a 的值为 100 ， b 的值为 200\n" );
       }
   }
   fmt.Printf("a 值为 : %d\n", a );
   fmt.Printf("b 值为 : %d\n", b );
}

#结果
a 的值为 100 ， b 的值为 200
a 值为 : 100
b 值为 : 200
```

Go 语言 switch 语句

switch 语句用于基于不同条件执行不同动作，每一个 case 分支都是唯一的，从上至下逐一测试，直到匹配为止。

switch 语句执行的过程从上至下，直到找到匹配项，匹配项后面也不需要再加 break。

switch 默认情况下 case 最后自带 break 语句，匹配成功后就不会执行其他 case，如果我们需要执行后面的 case，可以使用 **fallthrough** 。

Go 编程语言中 switch 语句的语法如下：

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

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，但必须是相同的类型；或者最终结果为相同类型的表达式。

您可以同时测试多个可能符合条件的值，使用逗号分割它们，例如：case val1, val2, val3。多条件匹配。

示例：

```
package main

import "fmt"

func main() {
   /* 定义局部变量 */
   var grade string = "B"
   var marks int = 90

   switch marks {
      case 90: grade = "A"
      case 80: grade = "B"
      case 50,60,70 : grade = "C"
      default: grade = "D"
   }

   switch {
      case grade == "A" :
         fmt.Printf("优秀!\n" )
      case grade == "B", grade == "C" :
         fmt.Printf("良好\n" )
      case grade == "D" :
         fmt.Printf("及格\n" )
      case grade == "F":
         fmt.Printf("不及格\n" )
      default:
         fmt.Printf("差\n" );
   }
   fmt.Printf("你的等级是 %s\n", grade );
}

#结果
优秀!
```

```
你的等级是 A
```

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。

Type Switch 语法格式如下：

```
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);
}
```

示例：

```
package main

import "fmt"

func main() {
   var x interface{}

   switch i := x.(type) { // 带初始化语句
      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>
```

fallthrough

使用 fallthrough 会强制执行后面的 case 语句，fallthrough 不会判断下一条 case 的表达式结果是否为 true。

示例：

```
package main
```

```
import "fmt"

func main() {

    switch {
    case false:
            fmt.Println("1、case 条件语句为 false")
            fallthrough
    case true:
            fmt.Println("2、case 条件语句为 true")
            fallthrough
    case false:
            fmt.Println("3、case 条件语句为 false")
            fallthrough
    case true:
            fmt.Println("4、case 条件语句为 true")
    case false:
            fmt.Println("5、case 条件语句为 false")
            fallthrough
    default:
            fmt.Println("6、默认 case")
    }
}

#结果
2、case 条件语句为 true
3、case 条件语句为 false
4、case 条件语句为 true
```

从以上代码输出的结果可以看出：switch 从第一个判断表达式为 true 的 case 开始执行，如果 case 带有 fallthrough，程序会继续执行下一条 case，且它不会去判断下一个 case 的表达式是否为 true。

Go 语言 select 语句

select 是 Go 中的一个控制结构，类似于用于通信的 switch 语句。每个 case 必须是一个通信操作，要么是发送要么是接收。

select 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。一个默认的子句应该总是可运行的。

> select中的case语句必须是一个channel操作

> select中的default子句总是可运行的。
>
> 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。

Go 编程语言中 select 语句的语法如下：

```
select {
    case communication clause  :
       statement(s);
    case communication clause  :
       statement(s);
    /* 你可以定义任意数量的 case */
    default : /* 可选 */
       statement(s);
}
```

以下描述了 select 语句的语法：

- 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：

  1. 如果有 default 子句，则执行该语句。
  2. 如果没有 default 子句，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。

示例：

```
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")
   }
}
#结果
no communication
```

> 当我们讲到channel 时将着重介绍一下