Golang

Go là một ngôn ngữ lập trình mới do Google thiết kế và phát triển. Nó được kỳ vọng sẽ giúp ngành công nghiệp phần mềm khai thác nền tảng đa lõi của bộ vi xử lý và hoạt động đa nhiệm tốt hơn.

Golang Basic

Kiến thức cơ bản về ngôn ngữ lập trình Golang

Overview

Go là một ngôn ngữ lập trình mới do Google thiết kế và phát triển. Nó được kỳ vọng sẽ giúp ngành công nghiệp phần mềm khai thác nền tảng đa lõi của bộ vi xử lý và hoạt động đa nhiệm tốt hơn.

Sự ra đời của ngôn ngữ Go

Vào ngày 21/9/2007 bộ ba Robert Griesemer, Rob Pike và Ken Thompson bắt đầu phát thảo những tiêu chí cho một ngôn ngữ lập trình mới trên bảng trắng. Vài ngày sau đó, những tiêu chí đó được chốt và một kế hoạch xây dựng một ngôn ngữ mới được định hình. Sau đó, công việc thiết kế Go được tiến hành song song với các công việc không có liên quan khác.

Tháng 01/2008, Kem Thompson bắt tay xây dựng một trình biên dịch ra ngôn ngữ C để thử nghiệm các ý tưởng. Vào giữa năm 2008, ngôn ngữ mới này trở thành một dự án toàn thời gian và được đầu tư đủ để trở thành một trình biên dịch dùng cho production.

Vào tháng 5/2008, Ian Taylor đã độc lập phát triển một GCC front-end cho Go dựa trên bản nháp của đặc tả ngôn ngữ này. Russ Cox tham gia vào cuối năm 2008 và giúp hiện thực hoá ngôn ngữ này cũng như các thư viện của nó.

Vào ngày 10/11/2009, Go trở thành một dự án mã nguồn mở, và sau đó có rất nhiều cá nhân trong cộng đồng đã đóng góp ý tưởng cũng như mã nguồn cho dự án này.

Việc khai thác sức mạnh của các bộ xử lý đa lõi và phần cứng thế hệ mới đối với các ngôn ngữ hiện có được xem như là việc không thể được. Bởi những giới hạn vốn có của các ngôn ngữ lập trình trên máy tính như C, C++, Java... Bấy lâu nay, các vấn đề xử lý đa lõi vẫn là chuyện của hệ điều hành.

Google đưa ra ngôn ngữ Go như là một cách tiếp cận khác về vấn đề xử lý đa lõi. Thay vì chỉ có hệ điều hành được phép cấp tài nguyên và xử lý, thì các phần mềm cũng có thể tương tác trực tiếp với nền tảng đa lõi giúp cho việc xử lý nhanh hơn.

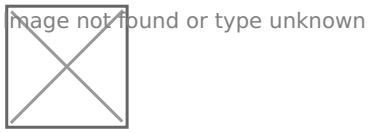
Cài đặt Go

Trước khi học một ngôn ngữ lập trình ta phải cài đặt môi trường, trình biên dịch ngôn ngữ đó. Với **Go** cũng vậy, ta sẽ tiến hành cài đặt nó trước khi bắt tay vào viết code.

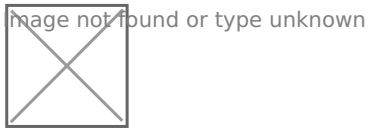
Các bạn vào website của **Go**: https://golang.org/dl/ để tải bản cài đặt tương ứng với hệ điều hành.

1. Cài đặt Go trên Windows

Sau khi tải về, các bạn cài đặt như bình thường. Sau khi cài xong, mặc định **Go** sẽ được đặt tại **C:\Go**, các bạn có thể mở ổ **C:** lên để kiểm tra. Bước tiếp theo chúng ta sẽ cài đặt một số biến môi trường, để cài đặt biến môi trường các bạn vào **Control Panel > Chọn System**. Và làm theo hướng dẫn như hình dưới:



Ta thêm tiếp biến **GOPATH**, biến môi trường này nói cho **Go** biết nơi chúng ta làm việc (nơi bạn chứa code) và chương trình mà chúng ta tạo ra sẽ đặt ở đó. Tôi chọn **C:\workspace\Go** là **GOPATH**:



Ok. Đến đây chúng ta đã xong quá trình cài đặt và cấu hình biến môi trường. Giờ chúng ta sẽ viết chương trình đầu tiên để kiểm tra quá trình cài đặt. Chương trình chúng ta sẽ viết là một chương trình kinh điển mà tất cả các lập trình viên khi học đều viết, nó là chương trình Hello World. Các bạn mở **notepad++** lên (hoặc cái gì đó để soạn thảo tùy bạn) và viết những lệnh sau:

helloworld.go

```
package main
import "fmt"
func main() {
   fmt.Printf("Hello World!")
}
```

Các bạn lưu lại vào trong **GOPATH**: `C:\workspace\Go\hello\helloworld.go`

Giờ chúng ta sẽ **build** chương trình vừa viết. Các bạn mở cửa sổ **cmd** lên và gõ lệnh như hình dưới. Nếu các bạn thấy hiện dòng chữ `Hello World!` là các bạn đã cài đặt thành công !.

```
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation.  All rights reserved.

C:\workspace\Go>go run hello\helloworld.go
Hello World!
C:\workspace\Go>
```

Chương trình Go Hello World

Chương trình Go

Trong phần này chúng ta sẽ học cách sử dụng Go bằng cách viết một chương trình quen thuộc là chương trình *Hello World.*

Chúng ta tạo một file text có tên `main.go` với nội dung sau:

```
package main
import "fmt"

// this is a comment

func main() {
    fmt.Println("Hello World")
}
```

Sau đó để dịch và chạy chương trình thì bạn vào Command Prompt (cmd) rồi gõ lệnh `go run <tên file>`. Bạn có thể phải chỉ ra cả đường dẫn đến tên file nếu thư mục hiện tại trong cmd không trùng với thư mục chứa file code. Chẳng hạn file source bạn để trong thư mục **C:/Go/main.go** thì thư mục hiện tại trong cmd phải là **C:/Go.**

```
C:\Go>go run main.go
Hello World
```

Nếu bạn làm đúng các bước trên thì sau khi gõ lệnh màn hình console sẽ in ra dòng chữ *Hello World.* Nếu không phải thì tức là bạn đã thực hiện sai ở bước nào đó, có thể là gõ sai code, trình biên dịch cũng sẽ thông báo lỗi ngay cho bạn. Và cũng giống như bất cứ trình biên dịch khác, chỉ cần trong code của bạn có lỗi thì cả chương trình sẽ không thể chạy được.

Giải thích

Một đoạn code chương trình Go được thực thi từ trên xuống dưới và từ trái sang phải giống như đọc một cuốn sách vậy.

```
package main
```

Dòng đầu tiên là `package main`, đây là câu lệnh khai báo "gói". Tất cả mọi chương trình Go đều phải được bắt đầu bởi câu lệnh khai báo gói. Tính năng Gói có tác dụng giúp chúng ta tổ chức code và tái sử dụng code dễ dàng hơn, chúng ta sẽ tìm hiểu thêm về gói trong các bài sau.

Khi biên dịch code Go thì có 2 loại là biên dịch thành chương trình chạy trực tiếp **(executable)** và biên dịch thành thư viện **(library).** Chương trình chạy trực tiếp là các file khả thi (có đuôi *.exe* trong Windows) có thể chạy một cách trực tiếp trong terminal (Command Prompt trong Windows). Còn thư viện là tập hợp code được gom lại với nhau và có thể được sử dụng trong các chương trình khác, chúng ta sẽ tìm hiểu về thư viện sau. Hiện tại chỉ bạn chỉ cần biết là cần phải có câu lệnh khai báo package trong code của mình.

Sau dòng khai báo package là một dòng trống, giống như các ngôn ngữ khác, trình biên dịch không quan tâm các các khoảng trống này, chúng ta chỉ dùng chúng để đọc code cho dễ hơn thôi.

```
import "fmt"
```

Từ khóa `import` có nghĩa là chúng ta yêu cầu được sử dụng code từ các gói khác trong chương trình của chúng ta. Ở đây là gói `fmt` (viết tắt của *format),* gói này chủ yếu chứa code thực hiện việc định dạng dữ liệu ra/vào.

Khi dùng từ khóa `import` thì tên gói được đặt trong cặp dấu nháy kép. Những kí tự được bọc trong cặp dấu nháy kép đều được gọi chung là chuỗi (hoặc string trong tiếng Anh), chúng ta sẽ tìm hiểu về string sau.

```
// this is a comment
```

Ký tự `//` cho biết những kí tự đứng sau nó là câu bình luận (comment). Các câu bình luận sẽ không được biên dịch. Trong Go có 2 loại comment, `//` là loại comment trên một dòng, tất cả các kí tự phía sau `//` sẽ không được biên dịch, và `/* */` là loại comment có thể sử dụng trên nhiều dòng, tất cả các kí tự nằm trong cặp dấu `/* */` sẽ không được biên dịch.

```
func main() {
   fmt.Println("Hello World");
}
```

Tiếp theo là phần khai báo hàm. Hàm là các thành phần xây dựng nên một chương trình. Hàm nhận dữ liệu vào, xử lý dữ liệu và xuất dữ liệu ra. Tất cả các hàm trong Go đều được định nghĩa bởi từ khóa `func,` theo sau là tên hàm (ở đây chúng ta định nghĩa hàm có tên `main),` tiếp theo là cặp dấu `()`, bên trong cặp dấu này chúng ta có thể khai báo một danh sách các tham số, tiếp theo là kiểu dữ liệu trả về (ở đây chúng ta không khai báo), rồi đến phần thân hàm nằm trong cặp dấu

ngoặc nhọn `{}`, thân hàm chứa các câu lệnh, ở đây chúng ta chỉ có duy nhất một câu lệnh. Chúng ta sẽ tìm hiểu thêm về hàm sau.

Ngoài ra cái tên `main` là một cái tên đặc biệt, hàm `main` tự động được hệ điều hành "gọi" đến đầu tiên khi chạy từ file khả thi.

> fmt.Println("Hello World");

Bên trong hàm `main` chúng ta chỉ có một câu lệnh. Câu lệnh này có 3 phần. Câu lệnh này gọi hàm `Println()` trong gói `fmt.` Hàm này nhận vào tham số là một string có tên "*Hello World*". Hàm `Println` (viết tắt của print line) thực hiện in chuỗi mà nó nhận được ra màn hình.

Tài liệu của Go được viết rất kỹ nhưng nếu bạn chưa học lập trình bao giờ thì đọc sẽ hơi thấy khó hiểu, ngoài ra tài liệu chủ yếu tiếng Anh là chính.

Cấu trúc chương trình Go

Trước khi tìm hiểu về xây dựng ứng dụng Go, chúng ta cùng tìm hiểu về cấu trúc đơn giản của một chương trình Go sau đây.

Hello World Example

Một chương trình Go cơ bản bao gồm các phần sau:

- Khai báo package
- Import các package
- Các function
- Khai báo các biến
- Các câu lệnh
- Ghi chú

hello.go

```
package main


import "fmt"


func main() {
   /* This is my first sample program. */
   fmt.Println("Hello, World!")
}
```

Trong đó:

- Dòng 1: Định nghĩa tên package của chương trình
- Dòng 2: import "fmt", sẽ yêu cầu Go compiler thực thi bao gồm cả file trong package fmt
- Dòng 3: func main() là hàm chứa các câu lệnh của chương trình.
- Dòng 4: /*...*/ sẽ được bỏ qua khi biên dịch chương trình. Mục đích của nó là ghi chú mục đích của câu lệnh cho developer. Cú pháp comment trong Go tương tự như Java và C++.
- Dong 5: fmt.Println() là function có sẵn trong Go, sẽ hiển thị nội dung "Hello, World!" trên màn hình.

Thực thi một chương trình Go

```
$ go run hello.go
Hello, World!
```

Kiểu dữ liệu trong Go

Trong ngôn ngữ lập trình Go, các kiểu dữ liệu tham chiếu đến một hệ thống mở rộng được sử dụng để khai báo các biến hoặc các hàm của các kiểu dữ liệu khác nhau. Kiểu của một biến xác định bởi không gian bộ nhớ để lưu trữ.

Các kiểu dữ liệu trong Go có thể được phân loại như sau:

#	**Kiểu dữ liệu**
1	Boolean Bao gồm 2 giá trị: (a) true - (b) false
2	**Numeric** Bao gồm các kiểu số: (a) integer - (b) float
3	**String** Kiểu dữ liệu String là một tập (set) các giá trị string. Giá trị của nó là một chuỗi các byte. String là kiểu bất biến được tạo ra một lần, không thể thay đổi nội dung của chuỗi. Loại chuỗi được khai báo trước là chuỗi.
4	**Derived** Bao gồm: 1. Pointer 2. Array 3. Structure 4. Union 5. Function 6. Slice 7. Interface 8. Map 9. Channel

Integer types

#	**Types and Description**
1	**uint8** Unsigned 8-bit integers (0 to 255)
2	**uint16** Unsigned 16-bit integers (0 to 65535)

3	**uint32** Unsigned 32-bit integers (0 to 4294967295)
4	**uint64** Unsigned 64-bit integers (0 to 18446744073709551615)
5	**int8** Signed 8-bit integers (-128 to 127)
6	**int16** Signed 16-bit integers (-32768 to 32767)
7	**int32** Signed 32-bit integers (-2147483648 to 2147483647)
8	**int64** Signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

Float types

#	Types and Description
1	**float32** IEEE-754 32-bit floating-point numbers
2	**float64** IEEE-754 64-bit floating-point numbers
3	**complex64** Complex numbers with float32 real and imaginary parts
4	**complex128** Complex numbers with float64 real and imaginary parts

Other Numberic types:

#	Types and Description
1	**byte** same as uint8
2	**rune** same as int32
3	**uint** 32 or 64 bits
4	**int** same size as uint
5	**uintptr** an unsigned integer to store the uninterpreted bits of a pointer value

Khai báo biến trong Go

Biến là gì?

Biến là tên được đặt cho một vị trí bộ nhớ để lưu trữ một kiểu (loại) dữ liệu cụ thể (kiểu số, chuỗi...)

Khai báo biến đơn

```
var variableName Type
```

Ví dụ:

```
var name string
```

Khai báo và khởi tạo:

```
var <tên biến> <kiểu giá trị> = <giá trị khởi tạo>
var <tên biến> = <giá trị khởi tạo>
<tên biến> := <giá trị khởi tạo>
```

Trong đó: Nếu như chúng ta không định nghĩa kiểu giá trị của biến, giá trị khởi tạo (giá trị đầu tiên mà biến nhận được) sẽ trở thành kiểu giá trị của biến đó.

Ví dụ:

```
var i = 123
//kiểu giá trị của i là int
var y = "123"
//kiểu giá trị của y là string
```

Đối với cách sử dụng ":=" kiểu giá trị của biến được khởi tạo sẽ là kiểu giá trị của giá trị khởi tạo cho biến đó.

Vì vậy các bạn cần lưu ý:

```
v := 1234
fmt.Println("v = ", v) // ok
fmt.Println("v = " + v) // error
```

Tham khảo ví dụ:

var.go

```
package main

import "fmt"

func main() {
	var s string
	s = "chuoi"
	fmt.Println("s = ", s)

	v := "abc"
	fmt.Println("v = ", v)
	fmt.Println("v = " + v)

	var i = 10
	fmt.Println("int i = ", i)

	var a, b int = 5, 10
	fmt.Println("a, b = ", a, b)

	var f = 12.3457439
	fmt.Println("f = ", f)
}
```

Hằng số trong Go

Hằng số (Constant) tham chiếu đến các giá trị cố định mà chương trình có thể không thay đổi trong quá trình thực hiện. Những giá trị cố định này cũng được gọi là **literals**.

Hằng số có thể là bất kỳ loại dữ liệu cơ bản nào như hằng số nguyên (Integer), hằng số động (Floating), hằng số ký tự (character) hoặc chuỗi ký tự (string). Cũng có các hằng số liệt kê (enum).

The `const` Keyword

```
const variableName type = value;
```

Ví dụ:

```
package main

import "fmt"

func main() {
   const LENGTH int = 10
   const WIDTH int = 5
   var area int

   area = LENGTH * WIDTH
   fmt.Printf("value of area : %d", area)
}
```

Kết quả thực hiện:

> value of area : 50

Golang Basic

Toán tử trong Go

1. Toán tử số học

Với A := 10 và B := 20

#	Operator	Description	Example
1	+	Phép cộng	A + B = 30
2	-	Phép trừ	A - B = -10
3	*	Phép nhân	A * B = 200
4	/	Phép chia	B / A = 2
5	%	Lấy phần dư	B % A = 0
6	++	Tăng giá trị thêm 1	A++ = 11
7	--	Giảm giá trị đi 1	A-- = 9

2. Toán tử so sánh

Với A := 10 và B := 20

#	Operator	Description	Example
1	==	So sánh bằng nhau	(A == B) is not true.
2	!=	So sánh khác nhau	(A != B) is true.
3	>	So sánh lớn hơn	(A > B) is not true.
4	<	So sánh nhỏ hơn	(A < B) is true.
5	>=	So sánh lớn hơn hoặc bằng	(A >= B) is not true.
6	<=	So sánh nhỏ hơn hoặc bằng	(A <= B) is true.

3. Toán tử logical

Với A := true và B := false

#	Operator	Description	Example
1	&&	AND	(A && B) is false.
2	||	OR	(A || B) is true.

3	!	NOT	!(A && B) is true.

4. Toán tử bitwise

#	p	q	p & q	p | q	p ^ q
1	0	0	0	0	0
2	0	1	0	1	1
3	1	1	1	1	0
4	1	0	0	1	1

Ví dụ: Với A := 60 và B:= 13

```
A = 0011 1100
B = 0000 1101
-----------------
A&B = 0000 1100
A| B = 0011 1101
A^B = 0011 0001
~A  = 1100 0011
```

#	Operator	Description	Example
1	&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
2	|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
3	^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
4	<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
5	>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

5. Toán tử gán

#	Operator	Description	Example
1	=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C

2	+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
3	-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
4	*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
5	/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
6	%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
7	<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
8	>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
9	&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
10	^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
11	|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

6. Toán tử khác

#	Operator	Description	Example
1	&	Returns the address of a variable.	&a; provides actual address of the variable.
2	*	Pointer to a variable.	*a; provides pointer to a variable.

7. Các ưu tiên của toán tử trong Go

#	Category	Operator	Associativity
1	Postfix	() [] -> . ++ - -	Left to right
2	Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
3	Multiplicative	* / %	Left to right
4	Additive	+ -	Left to right
5	Shift	<< >>	Left to right
6	Relational	< <= > >=	Left to right

7	Equality	== !=	Left to right
8	Bitwise AND	&	Left to right
9	Bitwise XOR	^	Left to right
10	Bitwise OR	|	Left to right
11	Logical AND	&&	Left to right
12	Logical OR	||	Left to right
13	Conditional	?:	Right to left
14	Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
15	Comma	,	Left to right

For trong Go

Lệnh for

Giả sử chúng ta cần in các con số từ 1 đến 10 ra màn hình, chúng ta có thể ghi 10 câu lệnh `fmt.Println()` như sau:

`main.go`

```
package main

import "fmt"

func main() {
    fmt.Println(1)
    fmt.Println(2)
    fmt.Println(3)
    fmt.Println(4)
    fmt.Println(5)
    fmt.Println(6)
    fmt.Println(7)
    fmt.Println(8)
    fmt.Println(9)
    fmt.Println(10)
}
```

Lệnh for cho phép lặp đi lặp lại các câu lệnh nhiều lần. Ví dụ:

`for.go`

```
package main

import "fmt"

func main() {
    var i int = 1
    for i <= 10 {
        fmt.Println(i)
        i  = i + 1
```

```
    }
}
```

Đoạn code trên sẽ in các chữ số từ 1 đến 10 ra màn hình, thay vì phải dùng 10 câu lệnh fmt.Println() thì bây giờ chúng ta chỉ cần dùng câu lệnh for là đủ.

Trong đoạn code trên, chúng ta khai báo biến i và gán giá trị là 1. Sau đó chúng ta sử dụng lệnh for để chạy câu lệnh fmt.Println() 10 lần bằng cách ghi từ khóa for, theo sau là một biểu thức điều kiện i <= 10, rồi tới khối lệnh nằm trong cặp dấu ngoặc nhọn {}.

Khi câu lệnh for bắt đầu chạy, đầu tiên câu lệnh này kiểm tra xem biến i có giá trị bé hơn 10 hay không, nếu đúng thì thực hiện những câu lệnh bên dưới, rồi quay ngược lại kiểm tra i cho đến khi nào i không còn bé hơn 10 thì dừng lại. Do đó chúng ta đặt lệnh i = i + 1 bên trong vòng lặp, cứ mỗi lần lặp giá trị của biến i sẽ được tăng lên 1 cho đến khi i = 10 thì vòng lặp for sẽ thoát.

Output

```
1
2
3
4
5
6
7
8
9
10
```

Chúng ta có thể đưa câu lệnh khai báo biến và lệnh tăng giá trị của biến ngay trên một dòng như sau:

for2.go

```
func main() {
    for i := 1; i <= 10 ; i++ {
       fmt.Println(i)
    }
}
```

Hầu hết các ngôn ngữ khác có rất nhiều lệnh lặp như while, do, until, foreach.... nhưng Go chỉ hỗ trợ một lệnh lặp duy nhất là lệnh for.

If Else trong Go

Lệnh if

Chúng ta sẽ thực hiện in các con số từ 1 đến 10 và cho biết số đó là chẵn hay lẻ. Để làm điều này thì chúng ta sẽ cần dùng đến câu lệnh if.

Ví dụ:

if.go

```
package main


import "fmt"


func main() {
    for i := 1; i <= 10 ; i++ {
        if i % 2 == 0 {
            fmt.Println(i, "chan")
        } else {
            fmt.Println(i, "le")
        }
    }
}
```

Ở đây chúng ta sử dụng câu lệnh if để kiểm tra xem biến i mang giá trị là số chẵn hay lẻ. Chúng ta ghi câu lệnh if, theo sau là một biểu thức điều kiện (tức là kết quả phải là true hoặc false), rồi đến khối lệnh nằm trong cặp dấu ngoặc nhọn {}, sau đó chúng ta có thể có thêm các câu lệnh else if hoặc else.

Nếu biểu thức điều kiện phía sau if là true thì thực hiện câu khối lệnh phía sau nó, ngược lại thì tiếp tục kiểm tra các biểu thức điều kiện tiếp theo nếu có.

Ở đây biểu thức điều kiện là câu lệnh i % 2 == 0, tức là chúng ta kiểm tra xem i có chia hết cho 2 hay không (hay i chia cho 2 không dư), nếu đúng thì i là số chẵn.

Output:

```
1 le
2 chan
```

```
3 le
4 chan
5 le
6 chan
7 le
8 chan
9 le
10 chan
```

Switch trong Go

Lệnh switch

Giả sử chúng ta muốn in các chữ số bằng chữ, chúng ta có thể viết đoạn code như sau:

`if.go`

```
if i == 0 {
    fmt.Println("Khong")
} else if i == 1 {
    fmt.Println("Mot")
} else if i == 2 {
    fmt.Println("Hai")
} else if i == 3 {
    fmt.Println("Ba")
} else if i == 4 {
    fmt.Println("Bon")
} else if i == 5 {
    fmt.Println("Nam")
}
```

Thay vì dùng câu lệnh if như trên, chúng ta có thể dùng câu lệnh switch như sau:

`switch.go`

```
switch i {
    case 0:  fmt.Println("Khong")
    case 1:  fmt.Println("Mot")
    case 2:  fmt.Println("Hai")
    case 3:  fmt.Println("Ba")
    case 4:  fmt.Println("Bon")
    case 5:  fmt.Println("Nam")
    default: fmt.Println("Khong biet")
}
```

Chúng ta dùng từ khóa switch, theo sau là một biểu thức điều kiện, rồi tới một danh sách các từ khóa case, ứng với mỗi từ khóa case là một giá trị nào đó, rồi tới dấu 2 chấm : và các lệnh sẽ được

thực hiện.

Ý nghĩa của lệnh switch là, nếu biểu thức điều kiện ở switch cho kết quả trùng với giá trị ở từ khóa case nào thì thực hiện các câu lệnh sau từ khóa case đó. Ngoài ra ở đây chúng ta còn có từ khóa default, có tác dụng thực hiện các câu lệnh nếu giá trị ở switch không trùng với bất kì từ khóa case nào.

Golang Basic

Defer trong Go

Trì hoãn (defer) là một khái niệm khá mới trong điều khiển luồng. Nó cho phép một câu lệnh được gọi ra nhưng không thực thi ngay mà hoãn lại đến khi các lệnh xung quanh trả về kết quả.

Ví dụ:

```
package main

import "fmt"

func main() {
    defer fmt.Println("World")   // Hoãn lệnh in ra chữ "World"
    fmt.Println("Hello")         // In ra chữ "Hello"
    // Kết quả cuối cùng là "Hello World"
}
```

Các lệnh được gọi qua từ khóa **defer** sẽ được đưa vào một **stack**, tức là hoạt động theo cơ chế vào sau ra trước (last-in-first-out). Lệnh nào **defer** sau sẽ được thực thi trước, giống như xếp 1 chồng đĩa thì chiếc đĩa sau cùng (ở trên cùng) sẽ được lấy ra trước.

Ví dụ:

```
package main

import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        defer fmt.Println(i)   // In ra giá trị của biến i
    }

    // Kết quả trả về ngược so với vòng lặp:
    // 9 8 7 6 5 4 3 2 1 0
}
```

Khi gọi lệnh **defer** thì giá trị của biến trong câu lệnh sẽ là giá trị tại thời điểm gọi chứ không phải giá trị tại thời điểm thực thi.

Kiểu Array trong Go

`Array` (mảng) trong Go tương tự các ngôn ngữ khác, tuy nhiên nó có kích thước cố định (fixed size) và các phần tử bên trong phải cùng loại dữ liệu.

Ví dụ:

```
// Khoi tao mang string voi kich thuoc 2
var a [2]string

// Gan gia tri
a[0] = "Hello"
a[1] = "World"

// print
fmt.Println(a[0], a[1])
fmt.Println(a)

// Khoi tao mang va dong thoi gan gia tri
primes := [6]int{2, 3, 5, 7, 11, 13}
fmt.Println(primes)

// Khi tạo mảng nhưng không ghi rõ kích thước (thay bằng dấu ba chấm), trình biên dịch sẽ tự hi
dựa vào số phần tử đã khai báo
numbers := [...]int{12, 78, 50}
fmt.Println(numbers)
```

Không giống đa số các ngôn ngữ khác, Array trong Go không phải là dạng tham chiếu (reference types) mà là dạng tham trị (value types). Khi gán giá trị nó cho một biến mới thì nó sẽ tạo ra một bản copy của Array cũ, và mọi thay đổi ở Array mới không ảnh hưởng gì đến Array cũ.

Ví dụ:

```
// Khởi tạo và gán giá trị
a := [...]int{1, 2, 3, 4, 5}
b := a   // b là một array mới có giá trị giống a
b[0] = 9   // Thay đổi giá trị một phần tử của b
```

```
fmt.Println("a is ", a)   // K□t qu□: 1 2 3 4 5
fmt.Println("b is ", b)   // K□t qu□: 9 2 3 4 5
```

Golang Basic

Kiểu Slice trong Go

`Slice` là một tham chiếu đến Array, nó mô tả một phần (hoặc toàn bộ) Array. Nó có kích thước động nên thường được sử dụng nhiều hơn Array.

Xem thêm về Array

1. Khởi tạo Slice

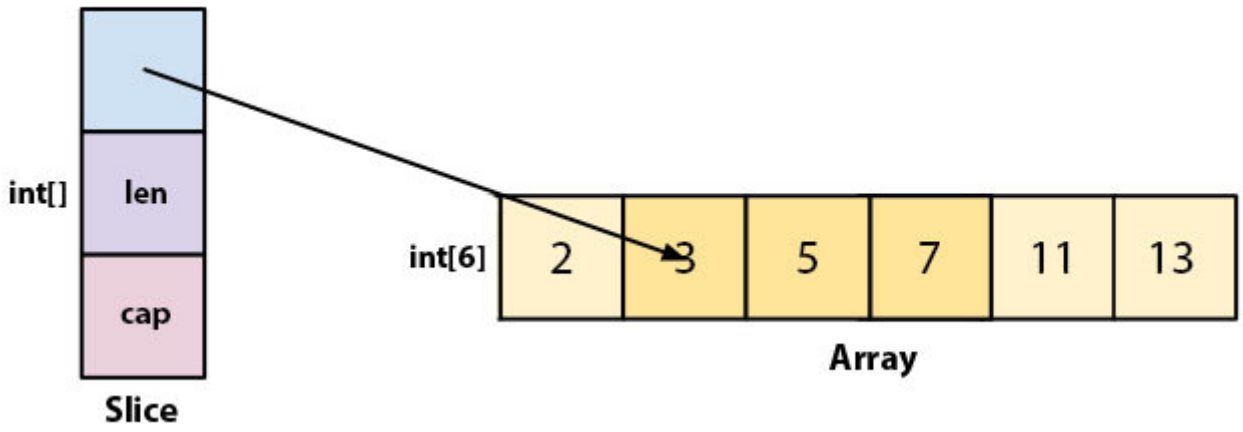
- Slice có thể tạo ra từ một Array bằng cách cung cấp 2 chỉ số (low và high) xác định vị trí phần tử trong Array.

Ví dụ:

```
// Khởi tạo Array primes
primes := [6]int{2, 3, 5, 7, 11, 13}

// Khởi tạo Slice s bằng cách cắt từ phần tử ở vị trí 1 (low) đến phần tử ở vị trí 3 (high - 1) c
Array primes
var s []int = primes[1:4]

// In ra giá trị của Slice s
fmt.Println(s) // Giá trị của s là [3, 5, 7]
```



- Một Slice sẽ có 2 thuộc tính là length (len) và capacity (cap). Length là số phần tử chứa trong Slice, còn capacity là số phần tử chứa trong Array mà Slice tham chiếu đến (bắt đầu tính từ phần tử đầu tiên của Slice). Để lấy ra length của Slice ta dùng hàm len(), còn để lấy ra capacity thì ta dùng

hàm cap().

Ví dụ:

```
s := []int{2, 3, 5, 7, 11, 13}

s = s[0:0] // s = [], len(s) = 0, cap(s) = 6
s = s[0:4] // s = [2, 3, 5, 7], len(s) = 4, cap(s) = 6
s = s[2:4] // s = [5, 7], len(s) = 2, cap(s) = 4, cap được tính từ vị trí s□ 2 tr□ đi
s = s[0:4] // s = [5, 7, 11, 13], len(s) = 4, cap(s) = 4
```

- Khi tạo Slice ta có thể bỏ qua các chỉ số low và high, khi đó Go sẽ tự sử dụng giá trị mặc định: 0 cho low và length của Slice cho high.

Ví dụ:

```
s := []int{2, 3, 5, 7, 11, 13}

s = s[:0] // s = [0:0]
s = s[:4] // s = [0:4]
s = s[2:] // s = [2:len(s)] => s = [2:4]
s = s[:4] // s = [0:4]
```

- Ngoài việc tạo Slice như trên, chúng ta có thể tạo theo các cách sau:

- Khai báo như một mảng nhưng không chỉ ra kích thước mảng:

```
q := []int{2, 3, 5, 7, 11, 13}
```

- Sử dụng hàm make() với công thức sau:

```
func make([]T, len, cap) []T

a := make([]int, 5) // len(a)=5
b := make([]int, 0, 5) // len(b)=0, cap(b)=5
```

- Slice có zero value là nil (length = 0 và capacity = 0), nil tương đương với giá trị null trong các ngôn ngữ lập trình khác.

- Do Slice chỉ là tham chiếu đến Array, do đó thay đổi giá trị của Slice sẽ làm thay đổi giá trị của Array mà nó tham chiếu đến. Nếu có nhiều Slice cùng tham chiếu đến một Array thì khi thay đổi giá trị một Slice có thể làm thay đổi giá trị các Slice khác.

Ví dụ:

```
numbers := [4]int{1, 2, 3, 4}

a := numbers[0:2] // a = [1, 2]
b := numbers[1:3] // b = [2, 3]

b[0] = 5 // Thay đổi giá trị phần tử đầu tiên của Slice b

fmt.Println(a, b) // a = [1, 5], b = [5, 3]
fmt.Println(numbers) // numbers = [1, 5, 3, 4]
```

2. Append Slice

- Để bổ sung thêm phần tử cho slice, ta dùng hàm append() với công thức sau:

```
func append(s []T, vs ...T) []T
```

- Hàm này sẽ trả về một slice có chứa toàn bộ các phần tử của slice ban đầu và các phần tử mới thêm vào. Trong trường hợp slice ban đầu có sức chứa nhỏ (Array mà nó tham chiếu đến có size nhỏ), một Array mới có kích thước lớn hơn sẽ được tạo ra và slice mới sẽ tham chiếu đến Array đó.

```
var s []int

// Append có thể hoạt động với nil slice.
s = append(s, 0) // s = [0]

// Append thêm một phần tử vào slice.
s = append(s, 1) // s = [0, 1]

// Append thêm nhiều phần tử vào slice.
s = append(s, 2, 3, 4) // s = [0, 1, 2, 3, 4]
```

3. Range Slide

- Range là một hình thức của vòng lặp for dùng để duyệt qua một slice hoặc map (sẽ nhắc đến ở phần sau). Mỗi một vòng lặp sẽ trả về 2 giá trị: Giá trị đầu tiên là chỉ số (vị trí) của phần tử, và giá trị thứ hai là bản sao của phần tử đó (cùng giá trị).

Ví dụ:

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}
for i, v := range pow {
    fmt.Printf("i = %d, v = %d \n", i, v)
}
```

- Trong trường hợp khi lặp chỉ sử dụng 1 trong 2 giá trị trả về thì ta sẽ bỏ qua giá trị còn lại bằng cách thay tên biến bằng ký tự gạch dưới (vì nếu không thì khi biên dịch sẽ báo lỗi biến được định nghĩa mà không sử dụng).

Ví dụ:

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}
for _, v := range pow {
    fmt.Printf("v = %d \n", v)
}
```