Universidade Federal de Sergipe Departamento de Sistemas de Informação
Projeto - Análise Léxica - 2019.2
Prof. André Luis Meneses Silva

Documentação Léxica da Linguagem Golang(Go)

1. Introdução

A linguagem de programação Go é um projeto open source para fazer programadores mais produtivos.

Go é expressivo, conciso, limpo e eficiente. Seus mecanismos de concorrência tornam mais fácil escrever programas que tirarão o máximo proveito de vários núcleos e máquinas em rede, enquanto o seu novo sistema de tipos permite a construção flexível e modular do programa. Go compila rapidamente para código de máquina ainda tem a conveniência de garbage collection e o poder de reflexão em tempo de execução. É uma linguagem compilada rápida, de tipagem estática, que parece uma linguagem interpretada digitada de forma dinâmica.

2. Lexemas de Golang

É uma linguagem de alto nível, é muito usada para web mas tem a capacidade de ser desenvolvida para qualquer coisa. De paradigma imperativo, modular com encapsulamento e polimorfismo, sem herança.

Uso de ponteiros como referência, sintaxe sucinta, possui coletor de lixo (não precisando se preocupar com a memória).

2.1 - Comentários

Comentários de uma linha é fornecido por //.

Enquanto comentários de várias linhas /* comentário */.

Da mesma forma que em C/C++.

2.2 - Palavras Reservadas

As seguintes palavras-chave são reservadas e não podem ser usadas como identificadores.

break	else	type
default	switch	continue
func	const	for
case	if	return
struct	range	var

2.3 - Operadores

As seguintes sequências de caracteres representam operadores (incluindo operadores de atribuição) e pontuação:

\+ Soma
\- Subtração
* Multiplicação
/ Divisão
% Porcentagem
== igualdade
!= diferente
< Menos que
<= Menor ou igual
> Maior que
>= Maior ou igual
&& condicional AND
|| Condicional OR
! Negação

2.4 - Literais inteiros

Um literal inteiro é uma sequência de dígitos que representa uma constante inteira. Um prefixo opcional define uma base não decimal: 0b ou 0B para binário, 0, 0o, ou 0O para octal, e 0x ou 0X para hexadecimal. Um único 0 é considerado um zero decimal. Em literais hexadecimais, as letras a até f, e A até F representam os valores de 10 a 15.

Para facilitar a leitura, um caractere de sublinhado _ pode aparecer após um prefixo base ou entre dígitos sucessivos; esses sublinhados não alteram o valor do literal.

Exemplos:

```
4_2
0600
0_600
0o600
0O600          // segundo caractere é a letra maiúscula 'O'
0xBadFace
0xBad_Face
0x_67_7a_2f_cc_40_c6
```

```
170141183460469231731687303715884105727
170_141183_460469_231731_687303_715884_105727

_42     // um identificador, não um literal inteiro
42_     // inválido: _ deve separar dígitos sucessivos
4__2    // inválido: apenas um _ por vez
0_xBadFace  // inválido: _ deve separar dígitos sucessivos
```

2.5 - Literais de ponto flutuante

Um literal de ponto flutuante é uma representação decimal ou hexadecimal de uma constante de ponto flutuante.

Um literal de ponto flutuante decimal consiste em uma parte inteira (dígitos decimais), um ponto decimal, uma parte fracionária (dígitos decimais) e uma parte do expoente (e ou E seguida de um sinal opcional e dígitos decimais). Uma parte inteira ou a parte fracionária pode ser omitida; uma das vírgulas decimais ou a parte do expoente pode ser omitida. Um valor de expoente exp escala a mantissa (parte inteira e fracionária) em 10^{exp}.

Um literal de ponto flutuante hexadecimal consiste em um 0x ou 0X prefixo, uma parte inteira (dígitos hexadecimais), um ponto de raiz, uma parte fracionária (dígitos hexadecimais) e uma parte do expoente (p ou P seguida de um sinal opcional e dígitos decimais). Uma parte inteira ou a parte fracionária pode ser omitida; o ponto radix também pode ser omitido, mas a parte do expoente é necessária. (Essa sintaxe corresponde à dada no IEEE 754-2008 §5.12.3.) Um valor de expoente exp escala a mantissa (parte inteira e fracionária) em 2^{exp}.

Para facilitar a leitura, um caractere de sublinhado _ pode aparecer após um prefixo base ou entre dígitos sucessivos; esses sublinhados não alteram o valor literal.

Exemplos:

```
0.
72,40
072.40                  // == 72,40
2,71828
1.e + 0
6.67428e-11
1E6
0,25
.12345E + 5
1_5.    // == 15.0
0.15e + 0_2   // == 15.0
```

```
0x1p-2                  // == 0.25
0x2.p10         // == 2048.0
0x1.Fp + 0      // == 1.9375
0X.8p-0         // == 0.5
0X_1FFFP-16                 // == 0.1249847412109375
0x15e-2         // == 0x15e - 2 (subtração número inteiro)

0x.p1    // inválido: mantissa não tem dígitos
1p-2     // inválido: o expoente p requer expoente hexadecimal
0x1.5e-2         // inválido: mantissa hexadecimal requer p expoente
1_.5     // inválido: _ deve separar dígitos sucessivos
1._5     // inválido: _ deve separar dígitos sucessivos
1.5_e1                  // inválido: _ deve separar dígitos sucessivos
1.5e_1                  // inválido: _ deve separar dígitos sucessivos
1.5e1_                  // inválido: _ deve separar dígitos sucessivos
```

2.6 - Literais de string

Um literal de string representa uma constante de string obtida da concatenação de uma sequência de caracteres. Existem duas formas: literais de string brutos e literais de string interpretados.

Literais de cadeia bruta são sequências de caracteres entre aspas, como em ‘foo’. Dentro das aspas, qualquer caractere pode aparecer, exceto as aspas. O valor de uma literal de sequência bruta é a sequência composta pelos caracteres não interpretados (implicitamente codificados em UTF-8) entre as aspas; em particular, as barras invertidas não têm significado especial e a sequência pode conter novas linhas. Caracteres de retorno de carro ('\ r') dentro de literais de sequência bruta são descartados do valor da sequência bruta.

Literais de sequência de caracteres interpretados são sequências de caracteres entre aspas duplas, como em "bar". Dentro das aspas, qualquer caractere pode aparecer, exceto nova linha e aspas duplas sem escape. O texto entre as aspas forma o valor do literal, com escapes de barra invertida interpretadas como estão em literais de runas (exceto que \' são ilegais e \" legais), com as mesmas restrições. Os escapes octais de três dígitos (\nnn) e hexadecimais de dois dígitos (\xnn) representam bytes individuais da sequência resultante; todas os outros escapes representam a codificação UTF-8 (possivelmente de vários bytes) de caracteres individuais.

Exemplos:

```
‘abc’   // o mesmo que "abc "
‘\n
\n’       // o mesmo que "\\ n \ n \\ n"
```

```
"\n"
"\" "      // o mesmo que ‘ " ’
"Olá, mundo! \n"
```

2.7 - Identificadores

2.7.1 Identificadores pré-declarados

Os seguintes identificadores são declarados implicitamente no bloco do universo :

Types:

bool byte float32 float64
int int8 int16 int32 int64 string
uint uint8 uint16 uint32 uint64 uintptr

Constants:

true false

Zero value:

nil

Functions:

print println

2.8 Variável

Uma variável é um local de armazenamento para armazenar um *valor* . O conjunto de valores permitidos é determinado pelo *tipo* da variável .

Uma declaração de variável ou, para parâmetros e resultados da função, a assinatura de uma declaração de função ou literal da função reserva armazenamento para uma variável nomeada. Chamar a função new interna ou pegar o endereço de um literal composto aloca armazenamento para uma variável em tempo de execução. Essa variável anônima é referida por meio de um indireto de ponteiro (possivelmente implícito) .

Variáveis *estruturadas* de tipos de matriz , fatia e estrutura têm elementos e campos que podem ser endereçados individualmente. Cada um desses elementos atua como uma variável.

O *tipo estático* (ou apenas o *tipo*) de uma variável é o tipo fornecido em sua declaração, o tipo fornecido na newchamada ou no literal composto ou o tipo de um elemento de uma variável estruturada. As variáveis do tipo de interface também têm um *tipo dinâmico* distinto , que é o tipo concreto do valor atribuído à variável no tempo de execução (a menos que o valor seja o identificador pré-declarado nil, que não tem tipo). O tipo dinâmico pode variar durante a execução, mas os valores armazenados nas variáveis da interface são sempre atribuíveis ao tipo estático da variável.

```
var x interface{}  // x is nil and has static type interface{}
var v *T           // v has value nil, static type *T
x = 42             // x has value 42 and dynamic type int
x = v              // x has value (*T)(nil) and dynamic type *T
```

O valor de uma variável é recuperado fazendo referência à variável em uma expressão ; é o valor mais recente atribuído à variável. Se uma variável ainda não tiver sido atribuída a um valor, seu valor será o valor zero para seu tipo.