Linguagem Go

Sem exageros

Projetada para a programação de sistema, a linguagem Go, criada pelo Google, faz o trabalho sem grandes sofisticações. Uma boa opção para quem é alérgico a exageros.

por Oliver Frommel

A linguagem Go não faz falsas promessas. Talvez seja exatamente o que a torna tão interessante a longo prazo. Em 2007, Robert Griesemer, Ken Thompson e Rob Pike inventaram o Go por estarem descontentes com as linguagens de programação de sistemas existentes. Eles não cederam a qualquer das tendências atuais como programação online assíncrona ou computação em nuvem; ao invés disso, aprenderam com a experiência de 30 anos com o C e criaram uma linguagem de programação capaz de se tornar sua sucessora. Como o C, o Go [1] mostra a sua força na programação do sistema, embora a linguagem possa ser implementada para praticamente qualquer finalidade. Os inventores da linguagem são funcionários do Google, e o Go fez o seu caminho até o *Google App Engine* assim como o Java e o Python; no entanto, o suporte ao Go está atualmente em fase experimental [2].

Promessa futura

No início de 2008, Thompson completou seu primeiro compilador experimental, que gera o código C. Ian Taylor começou um pouco mais tarde, trabalhando em uma interface Go para o compilador GCC. Perto do fim daquele ano, Russ Cox juntou-se ao projeto Go e o trabalho caminhou um pouco mais rápido. Em novembro de 2009, a equipe finalmente apresentou a primeira versão pública do compilador Go. Em março de 2012, a versão 1.0 e a especificação foram liberados, prometendo compatibilidade com as futuras versões do Go [3]. Portanto, a linguagem agora é adequada para projetos de software genuíno e não apenas experiências.

Os objetivos declarados do projeto Go são: compilação eficiente, execução rápida e programação simples. As linguagens existentes não conseguem combinar os três, dizem os inventores do Go, que se propõe a combinar a programação simples oferecida por linguagens cada vez mais populares como Python e Ruby com a eficiência e confiabilidade de outras mais "veteranas" como C, C++ e Java. Ao fazer isso, a compilação não vai demorar tanto tempo como em projetos Java, por exemplo. Além disso, a linguagem Go visa lidar com dependências entre bibliotecas externas de uma forma superior.

A simplicidade é uma das principais características do Go; seus inventores a criaram sem muitas construções. Em primeiro lugar, o Go tem como premissa uma sintaxe consistente e inequívoca. O mesmo não pode ser dito de linguagens como Perl, Ruby ou Scala, que usam uma variedade de construções sintáticas ou métodos para um único e mesmo propósito.

O Go é orientado em C mas omite muitos elementos que, de forma redundante, cumprem a mesma finalidade. Por exemplo, o Go utiliza apenas a variante Postfix

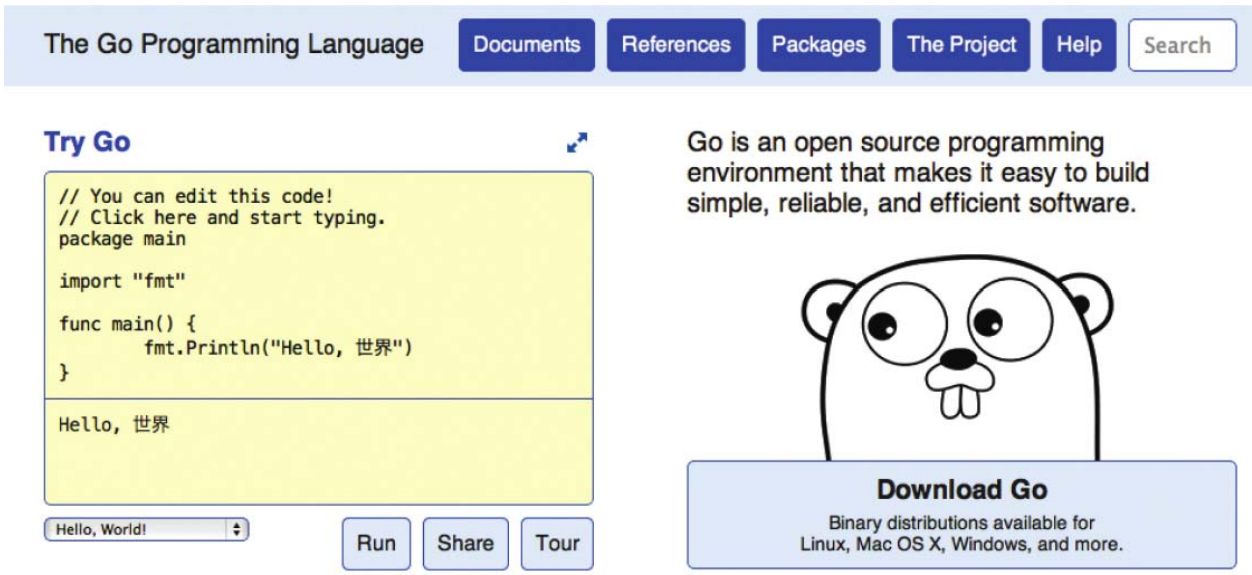

Figura 1 Um tour no site oferece aos usuários uma boa primeira impressão da linguagem Go.

do operador de incremento ++, que fica atrás da variável. Ao mesmo tempo, esta é apenas uma indicação e não uma expressão que pode ser usada imediatamente abaixo. Embora isto leve a um pouco mais de tipagem, garante semânticas claras e menos confusão.

Programas Go são feitos de modo ainda mais claro por meio de empréstimos de linguagens estruturadas como Pascal, Modula e Oberon. Em parte, a sintaxe Go foi feita em *Newsqueak* e *limbo*. Esta última é a linguagem de programação do sistema operacional Inferno que, por sua vez, é uma ramificação do sistema *Plan9*, no qual Thompson e Pike trabalharam anteriormente.

Claramente formatada

Por exemplo, o Go trabalha sem ponto e vírgula para completar instruções (**figura 1**). Embora vírgulas sejam parte da especificação da linguagem, o analisador as adiciona de forma independente, da forma como a especificação do JavaScript faz. Para que isto funcione, os programadores precisam manter o estilo de abertura de chaves especificado, a qual afirma que a abertura de chaves em um bloco aparece sempre no final de uma linha e não no início da linha seguinte. A ferramenta `gofmt` fornece com os pacotes a verificação desta formatação, eliminando a necessidade de o programador fazê-la e garantindo que o código Go tenha a mesma aparência em todos os projetos.

As variáveis são declaradas com a palavra-chave `var`, seguida do nome da variável e do tipo, isto é, na ordem contrária do C, C++, ou Java:

```
var x float64
```

Os nomes de variáveis podem começar com qualquer caractere considerado como uma letra no padrão Unicode. O operador `:=` permite definir e inicializar variáveis de uma só vez. A palavra-chave `var` pode ser omitida aqui, como na maioria das especificações de tipo:

```
i := 1
pi := 3.142
```

Os desenvolvedores Go também optaram por uma radical limpeza quando se trata de loops – se olharmos mais de perto, normalmente todos os `whiles`, `do untils`, `foreachs`, e assim por diante podem ser reformulados. O Go tem apenas um ciclo: o bom e velho loop `for`.

Tipos

A partir da crescente popularidade de linguagens como Python e JavaScript, os desenvolvedores deduziram que um sistema de tipos simples tende a promover a popularidade de uma linguagem de programação. Como Robert Griesemer coloca: "os sistemas de tipo desajeitado conduzem as pessoas para linguagens com tipagem dinâmica" [4]. No entanto, eles não estavam preparados para sacrificar a confiabilidade que as linguagens fortemente tipadas oferecem. O compromisso clássico no Go é que tipagem forte seja implementada com a inferência do tipo – isto é, qualquer variável tem um tipo fixo, mas o programador não precisa especificá-lo se o compilador naturalmente entender o tipo.

Ao mesmo tempo, o Go não tem uma hierarquia de tipo rigorosa. Na visão dos desenvolvedores, isso torna a implementação de compiladores e ferramentas mais complicado e leva a discussões intermináveis sobre a configuração concreta da hierarquia. Em vez disso, o Go oferece tipos

importantes, como arrays e mapas (*hashes*) no núcleo da linguagem. Ponteiros também existem, mas são mais parecidos com aqueles em Pascal (isto é, sem a aritmética típica do ponteiro da linguagem C).

Tipos genéricos que se tornaram populares no C++, Java, e Python não são suportados no Go, mesmo que os desenvolvedores admitam que possam ser úteis, e sua implementação não está inequivocamente descartada em algum momento no futuro distante. No entanto, o Go suporta programação orientada a objetos; a abstração para isto são interfaces, mas a interface em Go difere de suas contrapartes em Java ou Objective C. Uma interface especifica um conjunto de métodos que implementam "objetos" para cumprir uma determinada função, sem pertencer a uma classe comum.

Além disso, o Go não tem exceções porque elas causam mudanças incontroláveis no fluxo do programa. Como alternativa, a linguagem oferece valores de retorno múltiplos para funções, que podem usar um mecanismo curinga para capturar valores de erro. A **linha 2** da **listagem 1** mostra uma chamada para `os.Create()`, que ao mesmo tempo retorna o identificador de arquivo e um código de erro.

Ferramentas

O interessado em programar na linguagem Go normalmente só precisa instalar o compilador; as distribuições mais recentes do Linux já o possuem em seus sistemas de gerenciamento de pacotes. Para instalar no Ubuntu 12.04, digite:

```
apt-get install golang
```

Outros sistemas operacionais, incluindo Windows e Mac OS X, têm distribuições binárias do pacote Go.

Após a instalação, seu disco rígido irá conter (dependendo da arquitetura do processador) os comandos `6a`, `6c`, `6g` e `6l` - ou os seus homólogos com os dígitos `5` ou `8`. As seis ferramentas são para processadores AMD64, a série 5 são para processadores 368, e a série 8 são para processadores ARM. Os nomes estranhos são herdados do Plan9, que utiliza esses dígitos para identificar as arquiteturas de processadores.

Convenientemente, na prática, não temos que nos preocupar com esses detalhes e podemos simplesmente utilizar o `frontend go`. Também vale a pena mencionar que objetos gerados pelo compilador com a extensão `.6` contêm referências a todos os módulos utilizados, que o vinculador segue depois. Isto evita o problema de ter de encontrar as bibliotecas adequadas, além dos arquivos de cabeçalho, e especificá-los durante o processo de vinculação, como é o caso do C e do C++. Se um programa Go é construído, o compilador pode também vinculá-lo. Isto significa que os problemas de vinculador bem conhecidos são coisa do passado. O Go também lida com a tarefa de `makefiles`.

Se o usuário tiver instalado o Go corretamente, deve ser capaz de executar o comando `go` no console:

```
$ go version
go version go1
```

Directory /src/pkg

Name	Synopsis
..	
archive	
tar	Package tar implements access to tar archives.
zip	Package zip provides support for reading and writing ZIP archives.
bufio	Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.
builtin	Package builtin provides documentation for Go's predeclared identifiers.
bytes	Package bytes implements functions for the manipulation of byte slices.
compress	
bzip2	Package bzip2 implements bzip2 decompression.
flate	Package flate implements the DEFLATE compressed data format, described in RFC 1951.
gzip	Package gzip implements reading and writing of gzip format compressed files, as specified in RFC 1952.
lzw	Package lzw implements the Lempel-Ziv-Welch compressed data format, described in T. A. Welch, ``A Technique for High-Performance Data Compression'', Computer, 17(6) (June 1984), pp 8-19.
zlib	Package zlib implements reading and writing of zlib format compressed data, as specified in RFC 1950.
container	
heap	Package heap provides heap operations for any type that implements heap.Interface.

Figura 2 Alguns dos pacotes Go.

Listagem 1: Tratamento de erros

```
01 for try := 0; try < 2; try++ {
02 file, err = os.Create(filename)
03 if err == nil {
04 return
05 }
06 if e, ok := err.(*os.PathError); ok && e.Err == syscall.ENOSPC {
07 deleteTempFiles() // Recover some space.
08 continue
09 }
10 return
11 }
```

Se a instalação do Go não foi adicionada automaticamente ao `PATH` para arquivos executáveis, devemos apontar a variável de ambiente `GOROOT` para o diretório pai e em seguida adicionar `$GOROOT/bin` ao `PATH`.

Pelo fato de programas Go compilarem muito rapidamente, eles são bem adaptados para chamar scripts, com certas limitações – isto é, como uma espécie de "faça-você-mesmo", compilação no tempo exato. A ferramenta Gorun oferece suporte a este processo [5].

Avançado

Apesar de remover muitas características encontradas em outras linguagens, o Go acrescenta características modernas desejadas pelos programadores, tais como fechamentos, que no Go são funções anônimas que salvam o ambiente.

A linguagem atinge o objetivo declarado de programação simples para sistemas multicore – o Go foi desenvolvido na era pré-nuvem, quando processadores multicore eram o considerados o "Santo Graal" – com uma abstração chamada `goroutines`, que segue o modelo de processos de comunicação sequenciais (*communicating sequential processes*, ou CSP) para facilitar a programação sujeita a erros com threads [6].

Para ler mais, confira o documento online "Go Efetivo", que mostra algumas soluções "idiomáticas" [7]. Uma variedade de livros sobre Go foram publicados, e mais ajuda está disponível na lista de discussão *golang-nuts* (a lista *golang-dev* é destinada a desenvolvedores do compilador). Por fim, uma dica para a pesquisa online para recursos Go: em vez de digitar "go" como o termo a ser pesquisado, use sempre "golang". Localmente, poderemos ler toda a documentação no navegador se iniciarmos o servidor de documentação, digitando

```
godoc -http=:8000
```

O Go vem com uma variedade de bibliotecas (**figura 2**); é possível encontrar a lista completa em vários sites [8][9][10]. Os maiores projetos implementados em Go incluem o servidor online Falcore [11] e o software StatHat [12].

Conclusão

A linguagem Go é madura e, graças à compatibilidade futura prometida pelos desenvolvedores, é adequada para projetos de software profissionais. Além disso, os tempos curtos de compilação são úteis para grandes projetos. A redução no escopo da linguagem a torna relativamente fácil de aprender e mantém código de terceiros fácil de ler, o que acabará por beneficiar a manutenção de qualquer parte do software.

O preço para essa redução ao essencial é a ausência de recursos como classes baseadas na orientação a objetos, que muitos programadores estão acostumados a ter. O Go oferece vários recursos que permitem desenvolvimento orientado a objeto, mas requer alguma acomodação de novos conceitos. O Go está previsto para ser implementado em um número crescente de projetos, especialmente na área de programação de sistema, mas suporte no Google App Engine também pode torná-la interessante para uso em programação online. ■

Mais informações

[1] Go: http://golang.org/

[2] Go no Google App Engine: https://developers.google.com/appengine/docs/go/

[3] Post sobre o lançamento da primeira versão do Go: http://blog.golang.org/2012/03/go-version-1-is-released.html

[4] Robert Pike no GoogleTechTalks: http://www.youtube.com/watch?v=rKnDgT73v8s

[5] Gorun: http://wiki.ubuntu.com/gorun/

[6] CSP: http://www.usingcsp.com/cspbook.pdf

[7] Go Effective: http://golang.org/doc/effective_go.html

[8] Bibliotecas escritas em Go puro: http://go-lang.cat-v.org/pure-go-libs/

[9] Biblioteca bindings para Go: http://go-lang.cat-v.org/library-bindings/

[10] Projetos externos Go: http://godashboard.appspot.com/

[11] Falcore: http://ngenuity.ngmoco.com/2012/01/introducing-falcore-and-timber.html

[12] Construindo StatHat com Go: http://blog.golang.org/2011/12/building-stathat-with-go.html

Gostou do artigo?

Queremos ouvir sua opinião.
Fale conosco em
cartas@linuxmagazine.com.br

Este artigo no nosso site:
http://lnm.com.br/article/7949