06

Introdução às funções

Transcrição

Queremos agora construir as funções do nosso programa, isso inclui a função que monitora os sites, que exibe os logs, que sai do programa, entre outras. Mas se implementarmos o código dentro de cada caso nosso, teremos uma centralização do nosso código na função `main`, ou seja, teremos pouco reuso de código, tornando-o pouco modularizado.

Por isso, chegou a hora de organizarmos o nosso código, quebrando-o em funções. Teremos uma função específica para exibir a mensagem de boas vindas, uma para capturar o comando do usuário, e uma função para cada caso do nosso programa.

Função para exibir a introdução do programa

Já sabemos como declarar uma função, já que já declaramos a função `main`, então vamos criar a função `exibeIntroducao`, com o nosso código de boas vindas:

```
package main

import "fmt"

func main() {

    exibeIntroducao()

    fmt.Println("1- Iniciar Monitoramento")
    fmt.Println("2- Exibir Logs")
    fmt.Println("0- Sair do Programa")

    var comando int
    fmt.Scan(&comando)
    fmt.Println("O endereço da minha variavel comando é", &comando)
    fmt.Println("O comando escolhido foi", comando)

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
    default:
        fmt.Println("Não conheço este comando")
    }
}

func exibeIntroducao() {
    nome := "Douglas"
    versao := 1.1
    fmt.Println("Olá, sr.", nome)
    fmt.Println("Este programa está na versão", versao)
}
```

É comum em Go, utilizarmos o padrão *camelCase* para nome das funções, ou seja, a primeira letra minúscula e a cada nova palavra, sua primeira letra será maiúscula, sem espaço entre elas.

Retornando um valor em uma função

Agora, vamos criar a função `leComando`, para ler o comando digitado pelo usuário e retornar o seu valor para nós. No Go, colocamos o tipo do retorno da função após o nome da mesma:

```
func leComando() int {

}
```

E para retornar um valor, utilizarmos o `return`:

```
func leComando() int {
    var comandoLido int
    fmt.Scan(&comandoLido)
    fmt.Println("O comando escolhido foi", comandoLido)

    return comandoLido
}
```

Agora, na função `main`, nós chamamos essa função, já atribuindo-a à variável `comando`, utilizando o operador de declaração de variável curta:

```
package main

import "fmt"

func main() {

    exibeIntroducao()

    fmt.Println("1- Iniciar Monitoramento")
    fmt.Println("2- Exibir Logs")
    fmt.Println("0- Sair do Programa")

    comando := leComando()

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
    default:
        fmt.Println("Não conheço este comando")
    }
}
```

```
func exibeIntroducao() {
    nome := "Douglas"
    versao := 1.1
    fmt.Println("Olá, sr.", nome)
    fmt.Println("Este programa está na versão", versao)
}

func leComando() int {
    var comandoLido int
    fmt.Scan(&comandoLido)
    fmt.Println("O comando escolhido foi", comandoLido)

    return comandoLido
}
```

O Go verá que a variável `comando` será o retorno da função `leComando`, que é um inteiro, logo a variável também será um inteiro.

Função para exibir o menu

Do mesmo jeito, vamos extrair para uma função o código que exibe o menu do nosso programa:

```
package main

import "fmt"

func main() {

    exibeIntroducao()
    exibeMenu()
    comando := leComando()

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
    default:
        fmt.Println("Não conheço este comando")
    }
}

func exibeIntroducao() {
    nome := "Douglas"
    versao := 1.1
    fmt.Println("Olá, sr.", nome)
    fmt.Println("Este programa está na versão", versao)
}

func exibeMenu() {
    fmt.Println("1- Iniciar Monitoramento")
    fmt.Println("2- Exibir Logs")
    fmt.Println("0- Sair do Programa")
```

```
}

func leComando() int {
    var comandoLido int
    fmt.Scan(&comandoLido)
    fmt.Println("O comando escolhido foi", comandoLido)

    return comandoLido
}
```

Agora que já temos uma noção de como construir funções básicas, vamos nos aprofundar nisso ao longo do treinamento, mas já podemos começar pela função que sai do programa.

Saindo do programa

Podemos criar uma função para sair do programa, mas o seu código possuiria somente uma linha, vamos fazer isso diretamente no seu caso, dentro do `switch`. Para sair do programa, é uma boa prática retornarmos um status 0 para o sistema operacional.

Para fazer isso com Go, precisamos importar o pacote que se comunica com o sistema operacional, o pacote `os`. Importando esse pacote,, chamamos a sua função `Exit`, passando o valor `0` para ela, indicando para o sistema operacional que o programa foi encerrado com sucesso. A função `main` ficará assim:

```
package main

import "fmt"
import "os"

func main() {

    exibeIntroducao()
    exibeMenu()
    comando := leComando()

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
        os.Exit(0)
    default:
        fmt.Println("Não conheço este comando")
    }
}
```

Do mesmo jeito, há uma forma de informar o sistema operacional que ocorreu algum problema na execução do programa, como por exemplo quando o usuário digita um comando desconhecido. Para isso, passamos o valor `-1` para a função `Exit`. Então a função `main` ficará assim:

```
package main

import "fmt"
import "os"

func main() {

    exibeIntroducao()
    exibeMenu()
    comando := leComando()

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
        os.Exit(0)
    default:
        fmt.Println("Não conheço este comando")
        os.Exit(-1)
    }
}
```

Quando o usuário digitar um comando desconhecido, ao invés de encerrar o programa, podemos pedir para o usuário digitar um novo comando. Veremos isso mais à frente.