refactoring.guru

Singleton em Go / Padrões de Projeto

5–7 minutes

O **Singleton** é um padrão de projeto criacional, que garante que apenas um objeto desse tipo exista e forneça um único ponto de acesso a ele para qualquer outro código.

O Singleton tem quase os mesmos prós e contras que as variáveis globais. Embora sejam super úteis, eles quebram a modularidade do seu código.

Você pode usar classes que dependem de singletons em algumas outras situações. Você terá que levar a classe singleton também. Na maioria das vezes, essa limitação surge durante a criação de testes de unidade.

Exemplo conceitual

Normalmente, uma instância singleton é criada quando a struct é inicializada pela primeira vez. Para que isso aconteça, definimos o método `getInstance` na struct. Este método será responsável por criar e retornar a instância singleton. Uma vez criada, a mesma instância singleton será retornada toda vez que `getInstance` for chamado.

E as goroutines? A struct singleton deve retornar a mesma instância sempre que várias goroutines estiverem tentando

acessar essa instância. Por causa disso, é muito fácil implementar incorretamente o padrão de design singleton. O exemplo abaixo ilustra a maneira correta de criar um singleton.

Alguns pontos dignos de nota:

- Há uma verificação `nil` no início para garantir que `singleInstance` esteja vazio na primeira vez. Isso evita operações de bloqueio caras toda vez que o método `getinstance` é chamado. Se esta verificação falhar, significa que o campo `singleInstance` já está preenchido.
- A struct `singleInstance` é criada dentro do bloqueio.
- Há outra verificação de `nil` após o bloqueio ser adquirido. Isso é para garantir que, se mais de uma goroutine ignorar a primeira verificação, apenas uma goroutine possa criar a instância singleton. Caso contrário, todas as goroutines criarão suas próprias instâncias da struct singleton.

single.go: Singleton

```
package main

import (
    "fmt"
    "sync"
)

var lock = &sync.Mutex{}

type single struct {
}
```

```
var singleInstance *single

func getInstance() *single {
    if singleInstance == nil {
        lock.Lock()
        defer lock.Unlock()
        if singleInstance == nil {
            fmt.Println("Creating single instance now.")
            singleInstance = &single{}
        } else {
            fmt.Println("Single instance already created.")
        }
    } else {
        fmt.Println("Single instance already created.")
    }

    return singleInstance
}
```

main.go: Código cliente

```
package main

import (
    "fmt"
)

func main() {
```

```
    for i := 0; i < 30; i++ {
        go getInstance()
    }



    fmt.Scanln()
}
```

output.txt: Resultados da execução

```
Creating single instance now.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
```

```
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
Single instance already created.
```

Outro exemplo

Existem outros métodos de criação de uma instância singleton no Go:

1. função `init`

   Podemos criar uma única instância dentro da função `init`. Isso só é aplicável se a inicialização antecipada da instância estiver ok. A função `init` é chamada apenas uma vez por arquivo em um pacote, então podemos ter certeza de que apenas uma única instância será criada.

2. `sync.Once`

   O `sync.Once` só executará a operação uma vez. Veja o código abaixo:

   syncOnce.go: Singleton

   ```
   package main
   ```

```
import (
    "fmt"
    "sync"
)

var once sync.Once

type single struct {
}

var singleInstance *single

func getInstance() *single {
    if singleInstance == nil {
        once.Do(
            func() {
                fmt.Println("Creating single instance now.")
                singleInstance = &single{}
            })
    } else {
        fmt.Println("Single instance already created.")
    }

    return singleInstance
}
```

main.go: Código cliente

```
package main
```

```
import (
    "fmt"
)

func main() {

    for i := 0; i < 30; i++ {
        go getInstance()
    }



    fmt.Scanln()
}
```

output.txt: Resultados da execução

```
Creating single instance now.
Single instance already created.
Single instance already created.
```