Go 边看边练 -《Go 学习笔记》系列（七）

作者：88250
原文链接：https://ld246.com/article/1438260619759
来源网站：链滴

上一篇： https://hacpai.com/article/1438164538421

ToC

3.5 延迟调用

关键字 defer 用于注册延迟调用。这些调用直到 return 前才被执行，通常用用于释放资源或错误处。

```
func test() error {
    f, err := os.Create("test.txt")
    if err != nil { return err }

    defer f.Close() // 注册调用，而不是注册函数。必须提供参数，哪怕为空。

    f.WriteString("Hello, World!")
    return nil
}
```

多个 defer 注册，按 FILO 次序执行。哪怕函数或某个延迟调用发生错误，这些调用依旧会被执行。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/33128363b4bd41a492fce8073fe1c17.go?embed=true" width="99%" height="500"></iframe>

延迟调用参数在注册时求值或复制，可用指针或闭包 "延迟" 读取。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/d14360d2f3a048aad1edd09179a929d.go?embed=true" width="99%" height="500"></iframe>

滥用 defer 可能会导致性能问题，尤其是在一个 "大循环" 里。

```
var lock sync.Mutex
      func test() {
      lock.Lock()
      lock.Unlock()
}

func testdefer() {
    lock.Lock()
    defer lock.Unlock()
}

func BenchmarkTest(b *testing.B) {
    for i := 0; i < b.N; i++ {
        test()
    }
}

func BenchmarkTestDefer(b *testing.B) {
    for i := 0; i < b.N; i++ {
        testdefer()
    }
}
```

输出：

```
BenchmarkTest" 50000000 43 ns/op
BenchmarkTestDefer 20000000 128 ns/op
```

3.6 错误处理

没有结构化异常，使用 panic 抛出错误，recover 捕获错误。

```
func test() {
    defer func() {
        if err := recover(); err != nil {
            println(err.(string)) // 将 interface{} 转型为具体类型。
        }
    }()

    panic("panic error!")
}
```

由于 panic、recover 参数类型为 interface{}，因此可抛出任何类型对象。

```
func panic(v interface{})
```

```
func recover() interface{}
```

延迟调用中引发的错误，可被后续延迟调用捕获，但仅最后一个错误可被捕获。

```
func test() {
    defer func() {
        fmt.Println(recover())
    }()

    defer func() {
        panic("defer panic")
    }()

    panic("test panic")
}

func main() {
    test()
}
```

输出：

```
defer panic
```

捕获函数 recover 只有在延迟调用内直接调用才会终止错误，否则总是返回 nil。任何未捕获的错误会沿调用堆栈向外传递。

```
func test() {
    defer recover() // 无效!
    defer fmt.Println(recover()) // 无效!
    defer func() {
        func() {
            println("defer inner")
            recover() // 无效!
        }()
    }()

    panic("test panic")
}

func main() {
    test()
}
```

输出：

```
defer inner
<nil>
panic: test panic
```

使用延迟匿名函数或下面这样都是有效的。

```
func except() {
    recover()
}

func test() {
    defer except()
    panic("test panic")
}
```

如果需要保护代码片段，可将代码块重构成匿名函数，如此可确保后续代码被执行。

```
func test(x, y int) {
    var z int

    func() {
        defer func() {
            if recover() != nil { z = 0 }
        }()

        z = x / y
        return
    }()

    println("x / y =", z)
}
```

除用 panic 引发中断性错误外，还可返回 error 类型错误对象来表示函数调用状态。

```
type error interface {
    Error() string
}
```

标准库 errors.New 和 fmt.Errorf 函数用于创建实现 error 接口的错误对象。通过判断错误对象实例确定具体错误类型。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/17a23f2b07beeb2d8037ae0277fd2d2.go?embed=true" width="99%" height="630"></iframe>

如何区别使用 panic 和 error 两种方式？惯例是：导致关键流程出现不可修复性错误的使用 panic，他使用 error。

下一篇： https://hacpai.com/article/1438311936449

- **本系列是基于雨痕的《Go 学习笔记》（第四版）整理汇编而成，非常感谢雨痕的辛勤付出与分享！**
- 转载请注明：文章转载自： **黑客与画家的社区** [https://hacpai.com]
- 如果你觉得本章节做得不错，请在下面打赏一下吧~

社区小贴士

- 关注标签 [golang] 可以方便查看 Go 相关帖子
- 关注作者后如有新帖将会收到通知