



链滴

# Go 边看边练 - 《Go 学习笔记》系列 (十)

作者: [88250](#)

原文链接: <https://ld246.com/article/1438699210452>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

## ToC

- [Go 边看边练 - 《Go 学习笔记》系列 \(一\) - 变量、常量](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(二\) - 类型、字符串](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(三\) - 指针](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(四\) - 控制流1](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(五\) - 控制流2](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(六\) - 函数](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(七\) - 错误处理](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(八\) - 数组、切片](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(九\) - Map、结构体](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(十\) - 方法](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(十一\) - 表达式](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(十二\) - 接口](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(十三\) - Goroutine](#)
  - [Go 边看边练 - 《Go 学习笔记》系列 \(十四\) - Channel](#)
- 

## 5.1 方法定义

方法总是绑定对象实例, 并隐式将实例作为第一实参 (receiver)。

- 只能为当前包内命名类型定义方法。
- 参数 `receiver` 可任意命名。如方法中未曾使用, 可省略参数名。
- 参数 `receiver` 类型可以是 `T` 或 `*T`。基类型 `T` 不能是接口或指针。
- 不支持方法重载, `receiver` 只是参数签名的组成部分。
- 可用实例 `value` 或 `pointer` 调用全部方法, 编译器自动转换。

没有构造和析构方法, 通常用简单工厂模式返回对象实例。

```
type Queue struct {
    elements []interface{}
}

func NewQueue() *Queue { // 创建对象实例。
    return &Queue{make([]interface{}, 10)}
}
```

```

func (*Queue) Push(e interface{}) error { // 省略 receiver 参数名。
    panic("not implemented")
}

// func (Queue) Push(e int) error { // Error: method redeclared: Queue.Push
//     panic("not implemented")
// }

func (self *Queue) length() int { // receiver 参数名可以是 self、this 或其他。
    return len(self.elements)
}

```

方法不过是一种特殊的函数，只需将其还原，就知道 `receiver T` 和 `*T` 的差别。

<https://wide.b3log.org/playground/8e4b597c71c3d356ae6f8334f0d1ee7.go?embed=true>

从 1.4 开始，不再支持多级指针查找方法成员。

```

type X struct{}

func (*X) test() {
    println("X.test")
}

func main() {
    p := &X{}
    p.test()

    // Error: calling method with receiver &p (type **X) requires explicit dereference
    // (&p).test()
}

```

## 5.2 匿名字段

可以像字段成员那样访问匿名字段方法，编译器负责查找。

<https://wide.b3log.org/playground/67600a9a520592a06da510f58be64fa.go?embed=true>

通过匿名字段，可获得和继承类似的复用能力。依据编译器查找次序，只需在外层定义同名方法，就可以实现 "override"。

<https://wide.b3log.org/playground/69aa1e0de79d63b7e74bf224ea9eb7f.go?embed=true>

## 5.3 方法集

每个类型都有与之关联的方法集，这会影响到接口实现规则。

- 类型 `T` 方法集包含全部 `receiver T` 方法。

- 类型 `*T` 方法集包含全部 `receiver T + *T` 方法。
- 如类型 `S` 包含匿名字段 `T`，则 `S` 方法集包含 `T` 方法。
- 如类型 `S` 包含匿名字段 `*T`，则 `S` 方法集包含 `T + *T` 方法。
- 不管嵌入 `T` 或 `*T`，`*S` 方法集总是包含 `T + *T` 方法。

用实例 `value` 和 `pointer` 调用方法 (含匿名字段) 不受方法集约束，编译器总是查找全部方法，并自转换 `receiver` 实参。

下一篇：<https://hacpai.com/article/1438763483577>

- 
- **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