```
var address, person string
address = "John Smith"
person = "john@smith.com"
emailSomeone(address, person)
```

事实上，也可以执行 emailSomeone（person，address），程序仍然可以正确编译！

但如果 emailSomeone 定义如下：

```
func emailSomeone(address email, person string) {...}
```

那么下列代码就无法编译：

```
var address email
var person string
person = "John Smith"
address = "john@smith.com"
emailSomeone(person, address)
```

这其实是一件好事——可以防止错误发生。在此不再赘述。

Go 语言还可允许程序员自定义复杂类型：

```
type Record struct {
     Name string
     Age int
 }
```

在此，定义了一个称为 Record 的新类型，这是一个包含两种值的 struct：string 型的 Name 和 int 型的 Age。

什么是一个 struct？简单而言，struct 就是一个数据结构体。Record 中的 Name 和 Age 称为 struct 的字段。

如果熟悉 Python，那么 struct 就相当于一个元组，但作用相当于一个 NamedTuple。与 JavaScript 中最相近的是对象。同理，与 Java 中最相近的是一个普通的 Java 对象；在 C#中，与之最相近的是一个普通的 CLR 对象；在 C + + 中，则等效为一个普通数据。

请注意，在此使用了最相近和等效这两个词。之所以着重介绍了 struct，是因为在读者熟悉的大多数编程语言中，可能具有某种形式的 Java - esque 对象。struct 并不是一个类。只是描述在 CPU 如何排列数据的一种定义。因此，可以等效于 Python 中的元组而不是类，或新的数据类。

给定一个 Record 型的值，可能需要提取其内部数据，可以按如下执行：

```
r := Record {
    Name: "John Smith",
    Age: 20,
}
r.Name
```

上述代码表明了：

- 如何编写一个结构体类型的值——只需给出类型名称，然后在字段中填写内容即可。
- 如何读取结构体中的字段——采用了语法 . Name。

在本书中，将采用 . FIELDNAME 作为一个符号来获取特定数据结构的字段名。希望读者能够根据上下文理解所讨论的数据结构。有时也会采用完整结构（如 r. Name）来明确所讨论的字段。