包命名

介绍

Go代码被组织到包中。在包内，代码可以引用其中定义的任何标识符（名称），而包的客户端只能引用包的导出类型，函数，常量和变量。这样的引用始终将包名称作为前缀：`foo.Bar`引用名为`foo`包中的导出名称`Bar`。

好的程序包名称会使代码更好。程序包的名称为其内容提供了上下文，从而使客户更容易了解程序包的用途以及如何使用它。该名称还可以帮助程序包维护者确定其在演化过程中什么属于以及什么不属于它。具名的包使查找所需代码变得更加容易。

《Effective Go》提供 指导方针来命名包，类型，函数和变量。本文在此讨论的基础上进行了扩展，并调查了标准库中的名称。本文还讨论了错误的程序包名称以及如何修复它们。

包装名称

好的包名称简短明了。它们是小写字母，没有`下划线`或`大小写混合`。它们通常是简单的名词，例如：

- `time`（提供用于测量和显示时间的功能）
- `list`（实现一个双向链接列表）
- `http`（提供HTTP客户端和服务器实现）

其他语言的典型名称样式在Go程序中可能不是惯用的。以下是两个名称示例，这些名称在其他语言中可能是不错的样式，但不适用于Go：

- `computeServiceClient`
- `priority_queue`

Go包可能会导出几种类型和功能。例如，`compute`程序包可以导出`Client`类型，其中包含使用服务的方法，以及用于跨多个客户端划分计算任务的功能。

明智地缩写。 当程序员熟悉缩写名称时，可以使用缩写名称。广泛使用的包通常具有压缩名称：

- `strconv`（字符串转换）
- `syscall`（系统调用）
- `fmt`（格式化的I/O）

另一方面，如果缩写程序包名称使它含糊不清或不清楚，则不要这样做。

不要窃用客户的好名称。 避免给包指定客户端代码中常用的名称。例如，缓冲的I/O包称为`bufio`，而不是`buf`，因为它`buf`是缓冲区的一个很好的变量名。

命名包装内容

包名称及其内容的名称是耦合的，因为客户端代码将它们一起使用。设计包时，请考虑客户的观点。

避免口吃。 由于客户端代码在引用包内容时将包名称用作前缀，因此这些内容的名称无需重复包名称。`http`程序包提供的HTTP服务器称为`Server`，而不是`HTTPServer`。客户端代码将此类型称为`http.Server`，因此没有歧义。

简化函数名称。 当pkg包中的函数返回类型`pkg.Pkg`（或`*pkg.Pkg`）的值时，函数名称通常可以省略类型名称而不会引起混淆：

```
start := time.Now()                                  // start is a time.Time
t, err := time.Parse(time.Kitchen, "6:06PM")         // t is a time.Time
ctx = context.WithTimeout(ctx, 10*time.Millisecond)  // ctx is a
context.Context
ip, ok := userip.FromContext(ctx)                    // ip is a net.IP
```

在`pkg`包中的命名函数`New`返回类型是`pkg.Pkg`。这是使用该类型的客户端代码的标准入口点：

```
 q := list.New()  // q is a *list.List
```

当函数返回类型是`pkg.T`，这里`T`不是`Pkg`，函数名可以包括`T`使客户端代码更容易理解。常见的情况是一个包具有多个类似于New的功能：

```
d, err := time.ParseDuration("10s")  // d is a time.Duration
elapsed := time.Since(start)         // elapsed is a time.Duration
ticker := time.NewTicker(d)          // ticker is a *time.Ticker
timer := time.NewTimer(d)            // timer is a *time.Timer
```

不同包中的类型可以具有相同的名称，因为从客户端的角度来看，此类名称由包的名称来区分。例如，标准库中包括一个名为几种类型`Reader`，其中包括`jpeg.Reader`，`bufio.Reader`，和`csv.Reader`。每个包名称都使用`Reader`作为好的类型名称。

如果您不能给出一个对于该包内容而言是有意义的前缀作为包名称，则该包抽象边界可能是错误的。编写将包作为客户端使用的代码，并在结果不佳时重新构建包。这种方法将产生易于客户理解和易于开发人员维护的包。

包路径

Go软件包同时具有名称和路径。包名称在其源文件的package语句中指定；客户端代码将其用作包导出名称的前缀。导入程序包时，客户端代码使用程序包路径。按照惯例，包路径的最后一个元素是包名称：

```
import (
    "context"                // package context
    "fmt"                    // package fmt
    "golang.org/x/time/rate" // package rate
    "os/exec"                // package exec
)
```

构建工具将包路径映射到目录。go工具使用GOPATH 环境变量
在`$GOPATH/src/github.com/user/hello`目录中查找`"github.com/user/hello"`的源文件。（这种情况当然应该很熟悉，但是了解包的术语和结构很重要。）

目录。 标准库使用的目录，如`crypto`，`container`，`encoding`，和`image`将相关的协议和算法聚在一起。这些目录之一中的包之间没有实际关系；目录仅提供一种整理文件的方法。只要导入不创建循环，任何包都可以导入任何其他包。

就像不同包中的类型可以具有相同的名称一样，不同目录中的包可以具有相同的名称。例如，_runtime/pprof以pprof工具期望的格式提供分析数据，而net/http/pprof 提供HTTP端点显示分析数据。客户端代码使用包路径导入包，因此不会造成混淆。如果源文件需要导入两个`pprof`包，则可以 在本地重命名一个或两个。重命名导入的包时，本地名称应遵循与包名称相同的准则（小写，无`下划线`或`大小写混合`）。

不好的包名称

错误的程序包名称使代码难以导航和维护。这儿是一些识别和修复不良名称的准则。

避免无意义的程序包名称。 名为`util`，`common`或`misc`的程序包的名称毫无意义，因为它不能让客户清楚地明白该包所包含的内容。这使客户更难以使用该程序包，并使维护人员更难以保持程序包的重点。随着时间的流逝，它们会累积依赖关系，特别是在大型程序中，从而使编译工作变得不必要地缓慢。而且，由于此类包名称太通用，因此它们更有可能与客户端代码导入的其他包发生冲突，从而迫使客户端创造名称来区分它们。

分解通用包。 要修复此类包，请查找具有通用名称元素的类型和函数，然后将其放入自己的包中。例如，如果您有

```
package util
func NewStringSet(...string) map[string]bool {...}
func SortStringSet(map[string]bool) []string {...}
```

然后客户端代码看起来像

```
set := util.NewStringSet("c", "a", "b")
fmt.Println(util.SortStringSet(set))
```

将这些功能拉出`util`到新的包中，选择适合内容的名称：

```
package stringset
func New(...string) map[string]bool {...}
func Sort(map[string]bool) []string {...}
```

然后客户端代码变成

```
set := stringset.New("c", "a", "b")
fmt.Println(stringset.Sort(set))
```

进行此更改后，可以更轻松地了解如何改进新包：

```
package stringset
type Set map[string]bool
func New(...string) Set {...}
func (s Set) Sort() []string {...}
```

这产生了更简单的客户端代码：

```
set := stringset.New("c", "a", "b")
fmt.Println(set.Sort())
```

包装的名称是其设计包的关键步骤。尽可能从你的项目中消除无意义的包名称。

不要为所有API使用单个程序包。 许多好心的程序员把所有的程序的导出接口组织到同一个单一的名为`api`，`types`或者`interfaces`包中，以为依靠它能更容易找到代码库的入口点。这是个错误的想法。此类程序包与命名为`util`或`common`的程序包存在相同的问题，它们会无边界地增长，无法为用户提供任何指导，积累依赖关系，并与其他导入冲突。分解它们，也许可以使用目录将公共包从它们的实现的地方分开。

避免不必要的程序包名称冲突。 尽管不同目录中的程序包可能具有相同的名称，但经常一起使用的程序包应具有不同的名称。这样可以减少混乱，并减少客户端代码中本地重命名的需要。出于同样的原因，避免使用与流行的标准包，如`io`或`http`，相同的名称。

结论

包名称对于Go程序中的良好命名至关重要。花点时间选择好的包名称并组织好代码。这可以帮助客户理解和使用您的包，并帮助维护人员优雅地扩展它们。