45 | 使用os包中的API （下）

2018-11-23 郝林

你好，我是郝林，今天我们继续分享使用os包中的API。

我们在上一篇文章中。从“os.File类型都实现了哪些io包中的接口”这一问题出发，介绍了一系列的相关内容。今天我们继续围绕这一知识点进行扩展。

知识扩展

问题1：可应用于File值的操作模式都有哪些？

针对File值的操作模式主要有只读模式、只写模式和读写模式。这些模式分别由常量os.O_RDONLY、os.O_WRONLY和os.O_RDWR代表。在我们新建或打开一个文件的时候，必须把这三个模式中的一个设定为此文件的操作模式。

除此之外，我们还可以为这里的文件设置额外的操作模式，可选项如下所示。

- os.O_APPEND：当向文件中写入内容时，把新内容追加到现有内容的后边。
- os.O_CREATE：当给定路径上的文件不存在时，创建一个新文件。
- os.O_EXCL：需要与os.O_CREATE一同使用，表示在给定的路径上不能有已存在的文件。
- os.O_SYNC：在打开的文件之上实施同步I/O。它会保证读写的内容总会与硬盘上的数据保持同步。
- os.O_TRUNC：如果文件已存在，并且是常规的文件，那么就先清空其中已经存在的任何内容。

对于以上操作模式的使用，os.Create函数和os.Open函数都是现成的例子。

```
func Create(name string) (*File, error) {
 return OpenFile(name, O_RDWR|O_CREATE|O_TRUNC, 0666)
}
```

os.Create函数在调用os.OpenFile函数的时候，给予的操作模式是os.O_RDWR、os.O_CREATE和os.O_TRUNC的组合。

这就基本上决定了前者的行为，即：如果参数name代表路径之上的文件不存在，那么就新建一个，否则，先清空现存文件中的全部内容。并且，它返回的File值的读取方法和写入方法都是可用的。这里需要注意，多个操作模式是通过按位或操作符|组合起来的。

```
func Open(name string) (*File, error) {
return OpenFile(name, O_RDONLY, 0)
}
```

我在前面说过，os.Open函数的功能是：以只读模式打开已经存在的文件。其根源就是它在调用os.OpenFile函数的时候，只提供了一个单一的操作模式os.O_RDONLY。

以上，就是我对可应用于File值的操作模式的简单解释。在demo88.go文件中还有少许示例，可供你参考。

问题2：怎样设定常规文件的访问权限？

我们已经知道，os.OpenFile函数的第三个参数perm代表的是权限模式，其类型是os.FileMode。但实际上，os.FileMode类型能够代表的，可远不只权限模式，它还可以代表文件模式（也可以称之为文件种类）。

由于os.FileMode是基于uint32类型的再定义类型，所以它的每个值都包含了32个比特位。在这32个比特位当中，每个比特位都有其特定的含义。

比如，如果在其最高比特位上的二进制数是1，那么该值表示的文件模式就等同于os.ModeDir，也就是说，相应的文件代表的是一个目录。

又比如，如果其中的第26个比特位上的是1，那么相应的值表示的文件模式就等同于os.ModeNamedPipe，也就是说，那个文件代表的是一个命名管道。

实际上，在一个os.FileMode类型的值（以下简称FileMode值）中，只有最低的9个比特位才用于表示文件的权限。当我们拿到一个此类型的值时，可以把它和os.ModePerm常量的值做按位与操作。

这个常量的值是0777，是一个八进制的无符号整数，其最低的9个比特位上都是1，而更高的23个比特位上都是0。

所以，经过这样的按位与操作之后，我们即可得到这个FileMode值中所有用于表示文件权限的比特位，也就是该值所表示的权限模式。这将会与我们调用FileMode值的Perm方法所得到的结果值是一致。

在这9个用于表示文件权限的比特位中，每3个比特位为一组，共可分为3组。从高到低，这3组分别表示的是文件所有者（也就是创建这个文件的那个用户）、文件所有者所属的用户组，以及其他用户对该文件的访问权限。而对于每个组，其中的3个比特位从高到低分别表示读权限、写权限和执行权限。

如果在其中的某个比特位上的是1，那么就意味着相应的权限开启，否则，就表示相应的权限关闭。

因此，八进制整数0777就表示：操作系统中的所有用户都对当前的文件有读、写和执行的权限，而八进制整数0666则表示：所有用户都对当前文件有读和写的权限，但都没有执行的权限。

我们在调用os.OpenFile函数的时候，可以根据以上说明设置它的第三个参数。但要注意，只有在新建文件的时候，这里的第三个参数值才是有效的。在其他情况下，即使我们设置了此参数，也不会对目标文件产生任何的影响。

总结

为了聚焦于os.File类型本身，我在这两篇文章中主要讲述了怎样把os.File类型应用于常规的文件。该类型的指针类型实现了很多io包中的接口，因此它的具体功用也就可以不言自明了。

通过该类型的值，我们不但可以对文件进行各种读取、写入、关闭等操作，还可以设定下一次读取或写入时的起始索引位置。

在使用这个类型的值之前，我们必须先要创建它。所以，我为你重点介绍了几个可以创建，并获得此类型值的函数，包括：os.Create、os.NewFile、os.Open和os.OpenFile。我们用什么样的方式创建File值，就决定了我们可以使用它来做什么。

利用os.Create函数，我们可以在操作系统中创建一个全新的文件，或者清空一个现存文件中的全部内容并重用它。

在相应的File值之上，我们可以对该文件进行任何的读写操作。虽然os.NewFile函数并不是被用来创建新文件的，但是它能够基于一个有效的文件描述符包装出一个可用的File值。

os.Open函数的功能是打开一个已经存在的文件。但是，我们只能通过它返回的File值对相应

的文件进行读操作。

os.OpenFile是这些函数中最为灵活的一个，通过它，我们可以设定被打开文件的操作模式和权限模式。实际上，os.Create函数和os.Open函数都只是对它的简单封装而已。

在使用os.OpenFile函数的时候，我们必须要搞清楚操作模式和权限模式所代表的真正含义，以及设定它们的正确方式。

我在本文的扩展问题中分别对它们进行了较为详细的解释。同时，我在对应的示例文件中也编写了一些代码。你需要认真地阅读和理解这些代码，并在运行它们的过程当中悟出这两种模式的真谛。

我在本文中讲述的东西对于os包来说，只是海面上的那部分冰山而已。这个代码包囊括的知识众多，而且延展性都很强。

如果你想完全理解它们，可能还需要去参看操作系统等方面的文档和教程。由于篇幅原因，我在这里只是做了一个引导，帮助你初识该包中的一些重要的程序实体，并给予你一个可以深入下去的切入点，希望你已经在路上了。

思考题

今天的思考题是：怎样通过os包中的API创建和操纵一个系统进程？

戳此查看Go语言专栏文章配套详细代码。