GDB 调试 golang 代码

作者：cuua
原文链接：https://ld246.com/article/1618023283155
来源网站：链滴

11.2 使用GDB调试

开发程序过程中调试代码是开发者经常要做的一件事情，Go语言不像PHP、Python等动态语言，只修改不需要编译就可以直接输出，而且可以动态的在运行环境下打印数据。当然Go语言也可以通过Pritln之类的打印数据来调试，但是每次都需要重新编译，这是一件相当麻烦的事情。我们知道在Pytho中有pdb/ipdb之类的工具调试，Javascript也有类似工具，这些工具都能够动态的显示变量信息，单调试等。不过庆幸的是Go也有类似的工具支持：GDB。Go内部已经内置支持了GDB，所以，我们可通过GDB来进行调试，那么本小节就来介绍一下如何通过GDB来调试Go程序。

GDB调试简介

GDB是FSF(自由软件基金会)发布的一个强大的类UNIX系统下的程序调试工具。使用GDB可以做如下情：

启动程序，可以按照开发者的自定义要求运行程序。

可让被调试的程序在开发者设定的调置的断点处停住。（断点可以是条件表达式）

当程序被停住时，可以检查此时程序中所发生的事。

动态的改变当前程序的执行环境。

目前支持调试Go程序的GDB版本必须大于7.1。

编译Go程序的时候需要注意以下几点

传递参数-ldflags "-s"，忽略debug的打印信息

传递-gcflags "-N -l" 参数，这样可以忽略Go内部做的一些优化，聚合变量和函数等优化，这样对于DB调试来说非常困难，所以在编译的时候加入这两个参数避免这些优化。

常用命令

GDB的一些常用命令如下所示

list

简写命令l，用来显示源代码，默认显示十行代码，后面可以带上参数显示的具体行，例如：list 15，示十行代码，其中第15行在显示的十行里面的中间，如下所示。

```
10          time.Sleep(2 * time.Second)
11          c <- i
12      }
13      close(c)
14  }
15
16  func main() {
17      msg := "Starting main"
18      fmt.Println(msg)
19      bus := make(chan int)
```

break

简写命令 b,用来设置断点，后面跟上参数设置断点的行数，例如b 10在第十行设置断点。

delete 简写命令 d,用来删除断点，后面跟上断点设置的序号，这个序号可以通过info breakpoints获相应的设置的断点序号，如下是显示的设置断点序号。

```
·       Num     Type           Disp Enb Address            What
```

· 2 breakpoint keep y 0x0000000000400dc3 in main.main at /home/xiemengjun/gb.go:23

breakpointalready hit 1 time

backtrace

简写命令 bt,用来打印执行的代码过程，如下所示：

#0 main.main () at /home/xiemengjun/gdb.go:23

#1 0x000000000040d61e in runtime.main () at/home/xiemengjun/go/src/pkg/runtime/proc.c244

#2 0x000000000040d6c1 in schedunlock () at/home/xiemengjun/go/src/pkg/runtime/proc.c:67

#3 0x0000000000000000 in ?? ()

info

info命令用来显示信息，后面有几种参数，我们常用的有如下几种：

info locals

显示当前执行的程序中的变量值

info breakpoints

显示当前设置的断点列表

info goroutines

显示当前执行的goroutine列表，如下代码所示,带*的表示当前执行的

- 1 running runtime.gosched
- 2 syscall runtime.entersyscall

3 waiting runtime.gosched

4 runnable runtime.gosched

print

简写命令p，用来打印变量或者其他信息，后面跟上需要打印的变量名，当然还有一些很有用的函数le()和cap()，用来返回当前string、slices或者maps的长度和容量。

whatis

用来显示当前变量的类型，后面跟上变量名，例如whatis msg,显示如下：

type = structstring

next

简写命令 n,用来单步调试，跳到下一步，当有断点之后，可以输入n跳转到下一步继续执行

coutinue

简称命令 c，用来跳出当前断点处，后面可以跟参数N，跳过多少次断点

set variable

该命令用来改变运行过程中的变量值，格式如：set variable <var>=<value>

调试过程

我们通过下面这个代码来演示如何通过GDB来调试Go程序，下面是将要演示的代码：

package main

```
import (
"fmt"
"time"
)

func counting(cchan<- int) {
for i := 0; i < 10; i++ {
time.Sleep(2 * time.Second)
c <- i
}
close(c)
}

func main() {
msg := "Starting main"
fmt.Println(msg)
bus := make(chan int)
msg = "starting a gofunc"
go counting(bus)
for count := range bus {
fmt.Println("count:", count)
}
}
```

编译文件，生成可执行文件gdbfile:

```
go build-gcflags "-N -l" gdbfile.go
```

通过gdb命令启动调试：

```
gdb gdbfile
```

启动之后首先看看这个程序是不是可以运行起来，只要输入run命令回车后程序就开始运行，程序正的话可以看到程序输出如下，和我们在命令行直接执行程序输出是一样的：

```
(gdb) run
Startingprogram: /home/xiemengjun/gdbfile
Starting main
count: 0
count: 1
count: 2
count: 3
count: 4
count: 5
count: 6
```

```
count: 7
count: 8
count: 9
[LWP 2771exited]
[Inferior 1(process 2771) exited normally]
```

好了，现在我们已经知道怎么让程序跑起来了，接下来开始给代码设置断点：

```
(gdb) br main.main
Breakpoint 1 at0x4010b0: file /data/code/gowork/src/my_code/d5.go, line 13.
(gdb) br main.counting
Breakpoint 2 at0x401000: file /data/code/gowork/src/my_code/d5.go, line 6.
(gdb) list main.main
time.Sleep(2 * time.Second)
c <- i
}
close(c)
}
func main() {
msg := "Starting main"
fmt.Println(msg)
bus := make(chan int)
msg = "Starting a fo gofunction"
(gdb) list main.counting
package main
import (
"fmt"
"time"
)
func counting(c chan<- int){
for i:=0;i < 10;i++ {
time.Sleep(2 * time.Second)
c <- i
}
(gdb) run
Startingprogram: /home/xiemengjun/gdbfile
Starting main
[New LWP 3284]
[Switching toLWP 3284]
```

```
Breakpoint 1,main.main () at /home/xiemengjun/gdbfile.go:23
23          fmt.Println("count:",count)
```

上面例子b 23表示在第23行设置了断点，之后输入run开始运行程序。现在程序在前面设置断点的地停住了，我们需要查看断点相应上下文的源码，输入list就可以看到源码显示从当前停止行的前五行始：

```
(gdb) list
18      fmt.Println(msg)
19      bus := make(chan int)
20      msg = "starting a gofunc"
21      go counting(bus)
22      for count := range bus {
23          fmt.Println("count:",count)
24      }
25  }
```

现在GDB在运行当前的程序的环境中已经保留了一些有用的调试信息，我们只需打印出相应的变量，看相应变量的类型及值：

```
(gdb) infolocals
count = 0
bus =0xf840001a50
(gdb) p count
$1 = 0
(gdb) p bus
$2 = (chan int)0xf840001a50
(gdb) whatis bus
type = chan int
```

接下来该让程序继续往下执行，请继续看下面的命令

```
(gdb) c
Continuing.
count: 0
[New LWP 3303]
[Switching toLWP 3303]

Breakpoint 1,main.main () at /home/xiemengjun/gdbfile.go:23
23fmt.Println("count:", count)
(gdb) c
Continuing.
count: 1
[Switching toLWP 3302]
```

Breakpoint 1,main.main () at /home/xiemengjun/gdbfile.go:23

23fmt.Println("count:", count)

每次输入c之后都会执行一次代码，又跳到下一次for循环，继续打印出来相应的信息。

设想目前需要改变上下文相关变量的信息，跳过一些过程，并继续执行下一步，得出修改后想要的结：

(gdb) info locals

count = 2

bus =0xf840001a50

(gdb) set variable count=9

(gdb) info locals

count = 9

bus =0xf840001a50

(gdb) c

Continuing.

count: 9

[Switching toLWP 3302]

Breakpoint 1,main.main () at /home/xiemengjun/gdbfile.go:23

23fmt.Println("count:", count)最后稍微思考一下，前面整个程序运行的过程中到底创建了多少个goroutine，每个goroutine都在做什么：

(gdb) info goroutines

- 1 runningruntime.gosched
- 2 syscallruntime.entersyscall

3 waitingruntime.gosched

4 runnableruntime.gosched

(gdb) goroutine 1 bt

#00x000000000040e33b in runtime.gosched () at/home/xiemengjun/go/src/pkg/runtime/proc.c:927

#10x0000000000403091 in runtime.chanrecv (c=void, ep=void, selected=void,received=void)

at/home/xiemengjun/go/src/pkg/runtime/chan.c:327

#20x000000000040316f in runtime.chanrecv2 (t=void, c=void)

at/home/xiemengjun/go/src/pkg/runtime/chan.c:420

#30x0000000000400d6f in main.main () at /home/xiemengjun/gdbfile.go:22

#40x000000000040d0c7 in runtime.main () at/home/xiemengjun/go/src/pkg/runtime/proc.c:44

#5 0x000000000040d16ain schedunlock () at /home/xiemengjun/go/src/pkg/runtime/proc.c:67

#60x0000000000000000 in ?? ()

通过查看goroutines的命令我们可以清楚地了解goruntine内部是怎么执行的，每个函数的调用顺序

经明明白白地显示出来了。

小结

本小节我们介绍了GDB调试Go程序的一些基本命令，包括run、print、info、set variable、coutinu、list、break等经常用到的调试命令，通过上面的例子演示，我相信读者已经对于通过GDB调试Go序有了基本的理解，如果你想获取更多的调试技巧请参考官方网站的GDB调试手册，还有GDB官方网的手册