Using the Go Delve Debugger from the command line

07 Sep 2018 golang

Yesterday I had to revert to the command line to debug a program I was working on (long story as to why). I was initially apprehensive to debug from the command line but once I started I found the experience was quite nice. In fact I would compare debugging from the command line like “Vim for debugging”. Best part was I found the bug I was looking for!

I was working in Golang so I used the awesome tool Delve which is used by the VS Code Go extension. I didn’t find a simple tutorial on using the command line so I’ve created this to help me remember next time I need to fallback to the command line.

Install Delve (dlv)

The following steps assumes you have set up your go path.

```
go get -u github.com/derekparker/delve/cmd/dlv
```

Now you should be able to run (if not make sure you have your `GOPATH` configured properly):

```
dlv version
Delve Debugger
Version: 1.1.0
Build: $Id: 1990ba12450cab9425a2ae62e6ab988725023d5c $
```

Start Delve on a project

Delve can be run in many different ways but the easiest is to use `dlv debug`. This takes your current go package, builds it, then runs it. You can specify command parameters by using `-- <application parameters>` after the delve paramters you pass.

Get the example Go app and then start delve using the `hello` package:

```
$ go get github.com/golang/example/...
cd $GOPATH/src/github.com/golang/example/

# run delve
$ dlv debug ./hello
Type 'help' for list of commands.
(dlv)
```

You have now built the app and entered the debugging experience. Easy enough if you are like me, you are wondering what's next? With so many command how do I do start to do anything useful?

Navigating around at the command line

There are tons of commands you can run at this point but I like I like to start with just getting into application to see where I am and get started on the right foot. So let's set a breakpoint (`b` is short for `break`):

```
(dlv) b main.main
Breakpoint 1 set at 0x49c3a8 for main.main() ./hello/hello.go:25
(dlv)
```

Here we can see we set a breakpoint and even which file. Now execute the program until we hit it (`c` is short for `continue`):

```
(dlv) c
> main.main() ./hello/hello.go:25 (hits goroutine(1):1 total:1) (PC: 0x49c3a8)
    20:         "fmt"
    21:
    22:         "github.com/golang/example/stringutil"
    23: )
    24:
=>  25: func main() {
    26:         fmt.Println(stringutil.Reverse("!selpmaxe oG ,olleH"))
    27: }
(dlv)
```

Whoa! Now we hit the break point and can even see the code. Cool! This is fun lets keep going by stepping over the code (`n` is short for `next`):

```
(dlv) n
> main.main() ./hello/hello.go:26 (PC: 0x49c3bf)
    21:
    22:         "github.com/golang/example/stringutil"
    23: )
    24:
    25: func main() {
=>  26:         fmt.Println(stringutil.Reverse("!selpmaxe oG ,olleH"))
    27: }
```

Nice, now we can see that we have stepped to the next line of code. Now there isn't much to this code but we can see that we do have a package called `stringutil` that we are using to reverse the string. Let's step into that function (`s` is short for `step`):

```
(dlv) s
> github.com/golang/example/stringutil.Reverse() ./stringutil/reverse.go:21 (PC: 0x4
9c19b)
    16:
    17: // Package stringutil contains utility functions for working with strings.
    18: package stringutil
    19:
    20: // Reverse returns its argument string reversed rune-wise left to right.
=>  21: func Reverse(s string) string {
    22:         r := []rune(s)
    23:         for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
    24:                 r[i], r[j] = r[j], r[i]
    25:         }
    26:         return string(r)
(dlv)
```

Now we inside another package and function. Let's inspect the variable that has been passed in (`p` is short for `print`):

```
(dlv) p s
"!selpmaxe oG ,olleH"
(dlv)
```

Yup, that's the string value we were expecting! Next let's setting another breakpoint at line 24 and if we hit continue we should end up there:

```
(dlv) b 24
Breakpoint 2 set at 0x49c24e for github.com/golang/example/stringutil.Reverse() ./st
ringutil/reverse.go:24
(dlv) c
> github.com/golang/example/stringutil.Reverse() ./stringutil/reverse.go:24 (hits go
routine(1):1 total:1) (PC: 0x49c24e)
    19:
    20: // Reverse returns its argument string reversed rune-wise left to right.
    21: func Reverse(s string) string {
    22:         r := []rune(s)
```

```
    23:         for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
=>  24:                 r[i], r[j] = r[j], r[i]
    25:         }
    26:         return string(r)
    27: }
(dlv)
```

Sweet, but what's the values for `i`, `j`, and even `r`? Let's inspect the locals:

```
(dlv) locals
r = []int32 len: 19, cap: 32, [...]
i = 0
j = 18
```

Now we could do some tweaking of the local values and see what happens:

```
(dlv) set i = 1
(dlv) locals
r = []int32 len: 19, cap: 32, [...]
i = 1
j = 18
```

Nice that changed the value but will likely mess up our application. Let's start over (`r` is short for `restart`) just incase:

```
(dlv) r
Process restarted with PID 9429
```

Sweet but let's take a look at the breakpoints we have set, and remove the main entry point breakpoint:

```
(dlv) bp
Breakpoint unrecovered-panic at 0x42a7f0 for runtime.startpanic() /usr/local/go/src/r
untime/panic.go:591 (0)
        print runtime.curg._panic.arg
Breakpoint 1 at 0x49c3a8 for main.main() ./hello/hello.go:25 (0)
Breakpoint 2 at 0x49c24e for github.com/golang/example/stringutil.Reverse() ./stringu
til/reverse.go:24 (0)
(dlv) clearall main.main
Breakpoint 1 cleared at 0x49c3a8 for main.main() ./hello/hello.go:25
(dlv)
```

Alright if you're like me your feeling pretty good but are you ready to get fancy? We know the name of the function we want to jump to so let's search for the function name, set a breakpoint right were we want it, then execute to it:

```
(dlv) funcs Reverse
github.com/golang/example/stringutil.Reverse
```

```
(dlv) b stringutil.Reverse
Breakpoint 3 set at 0x49c19b for github.com/golang/example/stringutil.Reverse() ./st
ringutil/reverse.go:21
(dlv) c
> github.com/golang/example/stringutil.Reverse() ./stringutil/reverse.go:21 (hits gor
outine(1):1 total:1) (PC: 0x49c19b)
    16:
    17: // Package stringutil contains utility functions for working with strings.
    18: package stringutil
    19:
    20: // Reverse returns its argument string reversed rune-wise left to right.
=>  21: func Reverse(s string) string {
    22:         r := []rune(s)
    23:         for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
    24:                 r[i], r[j] = r[j], r[i]
    25:         }
    26:         return string(r)
(dlv)
```

Finally we might want to evaluate the checks on the for loop:

```
(dlv) c
> github.com/golang/example/stringutil.Reverse() ./stringutil/reverse.go:24 (hits go
routine(1):1 total:1) (PC: 0x49c24e)
    19:
    20: // Reverse returns its argument string reversed rune-wise left to right.
    21: func Reverse(s string) string {
    22:         r := []rune(s)
    23:         for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
=>  24:                 r[i], r[j] = r[j], r[i]
    25:         }
    26:         return string(r)
    27: }
(dlv) p len(r)-1
18
(dlv)
```

Everything looks fine, so I guess I work is done here:

```
(dlv) clearall reverse.go:24
Breakpoint 2 cleared at 0x49c24e for github.com/golang/example/stringutil.Reverse()
./stringutil/reverse.go:24
(dlv) c
Hello, Go examples!
Process 10956 has exited with status 0
(dlv) exit
Process 10956 has exited with status 0
```

That just touched the surface of what you can do but I hope it made you more comfortable with debugging on the command line. I think you can see how fast and efficient you can be. Good luck on you bug hunting!

Related Posts

Running Windows Unit tests for Kubernetes on Windows 01 Nov 2021

Helpful Cluster API commands for Devs 30 Sep 2021

Track active file in Goland 28 Apr 2021

Comments

ALSO ON **JAMES STURTEVANT**

Running Kubernetes Minikube on ...

5 years ago · 11 comments

Running Kubernetes Minikube on Windows 10 with WSL Sometimes ...

Getting Started with Project Oxford ...

7 years ago · 1 comment

Getting Started with Project Oxford Machine Learning APIs and Deployment on ...

Dependency Injection with Service Fabric · ...

7 years ago · 1 comment

Dependency Injection with Service Fabric In my previous post I talked ...

6 years

ASP.NE
Authen
you are

1 Comment

G

Join the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

5 Share

B

BigBear

3 years ago

How does delve figure out the correct type of a variable, when it calls the "p ${variable}" comr

0 0 • Reply • Share ›

Subscribe Privacy Do Not Sell My Data

© 2021. All rights reserved. Disclaimer: I currently work for Microsoft. The opinions expressed here are my own and do not necessarily reflect those of my employer.