2018-12-11 - golang-tools session - meeting notes

Previous session notes

2018-11-27 - golang-tools session - meeting notes

Recording

https://youtu.be/5isg5Xv3Yr0

Attendees

- Rebecca Stambler
- Michael Matloob
- Jay Conrod
- Ian Cottrell
- bcmills
- Roger Peppe
- Daniel Marti
- Marwan Sulaiman
- Ayan George
- Paul Jolly
- ...

Agenda

- Guest speakers:
  - Language Server (LSP) update/overview - Rebecca Stambler
- Updates on editor module-mode "betas" from VSCode, Atom and Vim
- Any updates on conversion of tools that fall under the tools umbrella issue (https://github.com/golang/go/issues/24661)
- go/packages updates
- go/analysis updates
- cmd/go updates
- LSP updates
- AOB

Notes

- *Editor's note: despite best intentions of a 40 min meeting, we raced through to an hour again… will aim to keep it more brief in the New Year*
- One guest speaker for this week, a very exciting topic to round out the year: Rebecca Stambler - Language Server (LSP) update/overview/explanation
  - Overview from the perspective of someone looking to contribute to the LSP work
  - Starting point: https://github.com/golang/tools/tree/master/internal/lsp
    - Will be kept internal for now until the it becomes clear which APIs it makes sense to expose more widely
  - The implementation is still rough but has support for:
    - Completion
    - Diagnostics
    - Formatting
    - Definitions/type defs/hover support
  - source package
    - Where most of the logic is (e.g. completion)
    - Like a library version of guru if you will - could become a library in the future
    - If you wanted to add a feature to, say, completion, this is where you would do that

        - cache package
            - Doesn't actually do anything yet… but this is where things will be "made faster"
        - protocol package
            - Bare-bones of LSP spec implementation; an almost exact translation of the LSP spec docs, not particularly Go-style/idiomatic therefore
        - testdata directory
            - Contains .gof file-based test cases for the existing features, e.g. completion, hover support
            - Expectations made in special comments, e.g. https://github.com/golang/tools/blob/master/internal/lsp/testdata/foo/foo.go
            - For anyone considering contributing, tests would be an incredibly valuable
        - Integration testing
            - Start point: https://github.com/golang/tools/tree/master/cmd/golsp/integration/vscode
            - This is a minimal version of the VSCode plugin
        - Current status
            - LSP in very active development
            - Rebecca and team are very happy to accept PRs/CLs
            - Best thing is to raise an issue first (e.g. https://github.com/golang/go/issues/29202), that way people can clearly "claim" things they would like to work on, coordinate, discuss questions etc
        - From the perspective of folks looking to integrate the LSP into other editors
            - Rebecca and team happy to contribute to discussion, thinking etc
            - Best to start/coordinate such conversations in golang-tools and then go from there
        - Questions
            - Roger - how much does the LSP rely on a shared file-system?
                - Currently local file-system based, works using go/packages + overlays
                - Future ambitions to make it work with remote LSP implementations
            - Marwan - what's the status of files in packages that add new imports?
                - Overlays now support (with the exception of _test.go files) new imports
                - Related issue: https://github.com/golang/go/issues/28809
                - Michael to update the go/packages docs to reflect the current state of overlays: https://go-review.googlesource.com/c/tools/+/153678 (merged)
- VSCode update - Ramya
    - No update this time around
- Atom update - Joe/Zac
    - No update this time around
- vim-go update - Billie
    - No update this time around
- gocode, godef, goimports and other tools under the tracking umbrella issue
    - goimports
        - Overview from Ian on the great work that Heschi has done to improve the modules support within the current implementation of golang.org/x/tools/imports
        - Works based on passes - more details in the recording. (*related Go 2 proposal: https://github.com/golang/go/issues/29036*)
        - This work (https://go-review.googlesource.com/c/tools/+/152317) is now merged, so the mainline goimports now supports modules. Hence people can "upgrade" via:
            - GO111MODULE=off go get -u golang.org/x/tools/cmd/goimports
    - godef
        - All Ian's work is now merged into Roger's original repo, hence people can "upgrade" for modules support via:
            - GO111MODULE=off go get -u github.com/rogpeppe/godef
    - gocode
        - Still being "maintained" as a fork:
            - GO111MODULE=off go get -u github.com/stamblerre/gocode
    - gorename
        - No updates per se; still expensive because it effectively requires a go list -export all in order to work

        - More caching possible in an LSP setting
- go/packages
    - Overlay support docs changed merged, per above
    - Some initial, internal, hacky support for propagating type sizes; more details in https://go-review.googlesource.com/c/tools/+/153199
- go/analysis
    - No specific updates
    - go/analysis based work will be incorporated in the LSP at a later date
    - Alan has (temporarily) switched to work on another project, and so has handed over the reins to others in the team for now. Wishing him all the best… and look forward to welcoming him back to golang-tools one day :)
- cmd/go
    - Jay has been looking into go list speed/bugs
    - We have now have release notes for 1.12
    - Modules status for 1.12: the behaviour will be the same as 1.11, we will hopefully lose the experimental label in 1.13
        - Some specific tweaks to vendoring/replacements required before we can move out of experimental
    - Module mode outside of a module continues to improve with changes landing that fix, amongst other issues:
        - https://github.com/golang/go/issues/24250
        - https://github.com/golang/go/issues/26241
        - https://github.com/golang/go/issues/26794
    - From this point onwards, small fixes will make the cut, larger fixes may not. Milestones have been moved to 1.13 where appropriate
    - go list
        - go list is surprising user-space CPU intensive; seems like there's room for improvement/caching here
        - Jay is also working on a file state-keyed metadata cache that should speed things up generally for go list operations (link to follow). Hopefully this will land in 1.12
        - Jay has also done some investigation into https://github.com/golang/go/issues/28739 (his response at https://github.com/golang/go/issues/28739#issuecomment-446265506). Looks like this boils down to basic file operations being ~4x faster in Docker (on MacOS host) vs on MacOS natively.
    - Upcoming Go releases
        - Security releases: 1.11.3 and 1.10.6 will happen first with only security-related fixes
        - Patch release for 1.11 series will follow that includes https://go-review.googlesource.com/c/go/+/151358/ amongst other fixes/backports
        - 1.12 release candidate will then follow
- LSP updates - covered earlier
- AOB - submodules
    - Feedback on the following welcomed
    - Conundrum 1: you open your editor (e.g. VSCode), open the directory that is the root of a repository, then open a file in a submodule in that repo, how are tools supposed to do anything with that file? Reason being, if per previous calls we follow the approach of setting the current workspace (main module) to the directory/module that is the repo root (i.e. the directory that you "opened"), and that parent module depends on the submodule that contains the file you opened, then the parent module (more than likely) depends on a version of the submodule (and hence the file you opened) in the module cache, not the file you see on disk. If a tool now attempts anything with a file= query on that file, then the answer (according to the parent module's go.mod) will be that there is no package in any module, reachable from the main module, that contains that file. How are we supposed to understand the intent of the user under these circumstances?
    - Options:
        - Let the user know they are likely missing a replace directive they probably intended to have? But when would you give them this suggestion?
        - gohack takes a simple but naive approach to this

        - One of the key nuances here is that the scenario above is described in terms of a text editor/IDE where you have a tree-view (or equivalent) of the repo directory you have opened. Hence you can see, on disk, the submodule… even though that directory may not be reachable from the main module (because the main module refers to a specific version in the module cache). So in effect this comes down to a problem of there being a discrepancy between a directory structure (as seen on disk) and a workspace view (i.e. the modules/files reachable from the main module). One option would be to have the IDE/editor hide sub modules that are in the directory structure we opened but that are not reachable from the main module.
    - Conundrum 2: if the users happens to open a file that is reachable from the current workspace (i.e. main module), should we behave different than if it is not reachable?
    - Conundrum 3: the above points all relate back to the question of context in editors
    - The questions that really come out of this?
        - What would people expect to see here?
        - How would they expect it to work?
            - Coming from GOPATH/dep/a-another world
            - With an understanding of how modules works..
        - What do people want to see here?
        - Do we have consensus on the "right" solution here?
        - Does this answer vary based on editor?
    - Worth starting a thread in golang-tools to flush out the various options here
    - We could automatically generate replace statements that get added to go.local.mod (without any fear of these replacements being committed)
- AOB - module/repo/package/file preferences
    - Picking up on the conversation in https://groups.google.com/d/msg/golang-tools/ZtCU2SteN1w/L6FoA6YgBQAJ and in Slack
    - The idea is that it be possible to set GOOS, GOARCH and build tag preferences/defaults. This might be need to be at various levels:
        - Repo
        - Module
        - Package
        - ...
    - Per Ian, this feels like something above the go tool and go.mod, potentially something in a separate file that could be understood by various editors
    - Ayan to experiment with what a straw man solution might look like
- AOB - there was an interesting discussion about pre/post conditions in code:
    - https://gophers.slack.com/archives/C4MTR76PQ/p1542729633080900
- AOB - this call's Interesting tools
    - https://github.com/Quasilyte/go-consistent
    - https://github.com/Quasilyte/go-namecheck

Thanks very much again to Rebecca for walking through the LSP overview.

Next call

TBC - some time in Jan 2019

Call message log

roger peppe

16:38

wow, old-school comments in lsp/protocol :)

Marwan Sulaiman

16:38

is there a place to see what you guys are working on and what's "help needed"?

Marwan Sulaiman

16:48

@Michael : https://github.com/golang/go/issues/28809

roger peppe

16:52

those changes sound great!

roger peppe

16:55

https://github.com/golang/go/issues/29036

Ayan George

17:00

:)

conor hennessy

17:00

:)

Daniel Martí

17:00

alan left us over text!

Marwan Sulaiman

17:02

is there a release estimate for when modules are out of the experimental flag?

Michael Matloob

17:02

Here's the link for the cl surfacing sizes https://go-review.googlesource.com/c/tools/+/153199

Daniel Martí

17:05

is there a particular reason why file syscalls are slower on mac compared to linux?

Ayan George

17:06

can you elaborate on what this feature is?

roger peppe

17:07

https://github.com/golang/go/issues/28974

Ayan George

17:07

thank you.

a-yan

Daniel Martí

17:09

maybe you could add a restriction, like "sub-modules cannot depend on parent modules"

Ayan George

17:18

yes -- that's what i came here to discuss.

roger peppe

17:18

@Daniel i sometimes wonder if it might be worth just disallowing submodules (as opposed to several modules with a shared root in the same repo)

roger peppe

17:21

tag combinations are a tricky issue in general

Daniel Martí

17:23

I do have some use cases for multiple modules within a single git repo, but I agree that sub-modules seems like a very niche thing

I'd probably be fine with submodules being unsupported

roger peppe

17:24

i use multiple modules within a single git repo already, but i haven't found a use for submodules yet either.

Daniel Martí

17:25

when you say multiple modules within a single git repo, can they depend on each other?

Bryan Mills

17:25

Yes, they can depend on each other but testing is tricky.

roger peppe

17:25

yeah; and of course that does lead to some of the same issues

i've just done it by having a PR per module

s/done it/approached testing/

Daniel Martí

17:27

yeah fair enough