Tools session 1 GopherCon 2019

Notes from the first of two sessions at GopherCon 2019. Link to session 2 notes:

https://docs.google.com/document/d/1ZI_WqpLCB8DO6teJ3aBuXTeYD2iZZZlkDptmcY6Ja60/edit#

- People in the room
    - Ian Cottrell
    - Paul Jolly
    - Daniel Martí
    - Caleb Spare
    - Mathieu Lonjaret
    - Roger Peppe
    - Jay Conrod
    - David Chase
    - Emmanuel Odeke
    - Bryan Mills
    - Artem Vysotsky
    - Suzy Mueller
    - Muir Manders
    - Peter Sandford
    - Rob Findley
    - Michael Matloob
    - Chris Wendt
    - Denis Isaev
    - Rebecca Stambler
    - Heschi Kreinick
    - Michael Stapelberg
    - Tristan Willy
    - Elena Morozova
    - Alex McKinney
    - Syrie Bianco
    - Hana Kim
- Brainstorm of topics for discussion
    - gopls - what is needed to reach v1.0?
    - gopls - features as extensions
    - go/packages performance
    - telemetry
    - go/packages support of bazel
    - debuggers
    - refactoring
    - code generators

gopls

- Should Go tools team be putting all eggs in one basket like this, supporting and owning gopls?
- What is the minimum requirement for gopls v1.0.0
    - Stability (inc testing)
    - gopls features
        - Cross package references/renaming?
            - Things have been constrained for now so that the reverse dependency analysis problem is scope limited
            - What about extending this scope to the current module?

            - Action item: potentially send out a strawman email to golang-tools that says "we want to ship v1.0.0 of gopls with references/rename support limited to the current package" and see who shouts
    - Editor issues
        - Emacs LSP plugin doesn't automatically find the Go module root; can be fixed by adding extra plugins, but that's not ideal
        - VSCode is in a good place….
        - … not totally clear on the state of the nation with respect to other editors
        - Question therefore is what does a release of gopls v1.0.0 mean? Because from the end users perspective, releasing v1.0.0 of gopls means nothing if the experience of getting things working through their editor is horrendous
        - Perhaps release v1.0.0 is actually a release to the authors of the editor plugins therefore?
        - Many editors have multiple LSP plugins, which is confusing
        - We should "bless" a number (ideally one) of LSP/Go plugins for each editor, which will require a specific version of gopls
        - Maybe this requires a "certified" status that requires the plugin support good error reporting process (see below), etc?
        - Need an "owner" for each editor plugin, all details linked from the gopls wiki (golang.org)
        - Action item: start the process of establishing this list of editors, plugins, owners, "certified' plugins etc. As part of this, make clear what v1.0.0 of gopls actually means (i.e. it only really makes sense to the editor plugin authors, end users shouldn't really care)
    - v1.0.0 roadmap
        - Mainly stability
        - Performance is also important; memory use has been an issue in the past. Many of the tools downstream require source information in memory
        - End user experience with the editors
        - Conclusion: performance and memory issues are not currently a blocker for v1.0.0
    - Error reporting diagnostics
        - How do people report issues? Is there a good process for doing this, from the end users' experience
        - Editor specific instructions on how to raise issues/bugs
        - Having a good error reporting process for users of all editors, regardless of the mechanics/details, is a blocker for v1.0.0
        - Action item: linked to the item above for identifying "certified" editor plugins; part of the certification requires a clear, well-documented error reporting process
    - Extra features
        - What tools are missing in gopls now? Is extensions a 1.0 blocker?
        - Implementations - what types implement this interface
        - Save hooks
            - gofmt -s
            - goreturns
            - …
    - Extensions
        - There have been a number of iterations looking at how to integrate external tools with gopls; none of the ideas we've had so far work properly
        - gopls is never going to have everything a user needs; e.g. internal tools, code generators
        - gopls is in a good position to be able to cache and reuse information
        - none of the data structures are meant to be shared between tools
        - how do we add extensions without recompiling? Go plugins?
        - inter-process communication is never going to be efficient
        - In-process you run the risk of some tools/plugins going rogue and mutating data when they shouldn't
        - if people compile gopls, how can useful bug reports exist?
        - Taking Uber as an example
            - Would they be able to support building their own version of gopls with extra extensions

        - What would this mean as far as the Go team supporting this version of gopls?
        - If gopls were invoking separate programs, at least you have the process isolation and gopls itself could report the bits its responsible for
    - The only really viable alternative is shipping gopls as a module that people write
    - Go language interpreter based approach?
    - Starlark based extension mechanism?
    - Evaluate Yaegi (talk on Fri)?
    - Fixing plugin?
    - What non-analysis tools would we want gopls to support?
        - Struct tag editing
        - Tools that stub out tests for you
        - Code generators...
    - The Uber folks think 1.0 should be cut without extension support; many of their tools are to be run sporadically via a binary, and not as part of an IDE
    - Conclusion: doesn't sound like extensions are a blocker for v1.0.0, but knowing they are on the roadmap (details notwithstanding) is important
    - Action item: communicate the strawman argument that extensions will NOT in for v1.0.0 and see if people shout
    - Action item: tell tool authors what to tell users when they ask how to run their tool as part of gopls

Debuggers

- State of delve is pretty good
- State of editor integration with delve is less good
- Debugger Adapter Protocol (DAP) - Delve team have made noises in the direction of accepting contributions to support this. Heschi can help someone make it happen.
- Delve is probably going to be blessed as the recommended Go debugger
- Once it starts being blessed, users are going to wonder how to use it with their editors
- Will likely have to go through a similar process to gopls of blessing certain plugins, having owners etc...
- What is the state of editor integrations with delve/DAP?
    - e.g. VSCode has native support (confirm)?
    - Vim appears to have https://github.com/puremourning/vimspector ...
- Action item: figure out the state of debugger integration in all popular editors to kick off process

go/packages performance

- go/packages performance breaks down into two phases
    - go list phase
    - go/packages phase
- There are a number of open issues from Yandex
    - Action item: Michael to follow up with Denis
- Action item: what types of problems are people having? Start a golang-tools email thread to gather feedback, e.g.
    - The current load mode bits are quite simple, do we want more knobs and granular control?
    - Do we want to version go/packages with tags?