contextify

tool to wire context

Siddharth Karandikar (clypd)

background

- At clypd (now xandr) we have been using ***go*** from 2013
- We use ***go*** for all backend work
    - Web apis
    - Business logic
    - Scheduler code
    - Batch jobs, ETL pipelines
    - Tools, utils
- 500K lines of code (including tests)
- ~1500 methods in DS layer (and many long running queries)
- ~320 API endpoints

expected behaviour from good backend

- When the request is canceled (or times out) - all goroutines started by the handler should exit as quickly as possible, free resources and make room for next request

context pkg

- Package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes.
- It became part of stdlib in v1.7 (Aug 2016)

demo - context cancellation and its use

```
package main
import (...)
func bigBackendProcessing(ctx context.Context) time.Time {
        select {
        case t := <-time.After(time.Second * 10):
                log.Println("time calculation done!")
                return t
        case <-ctx.Done():
                log.Println("context done, time calculation aborted")
                return time.Date(2000, 1, 1, 0, 0, 0, 0, time.UTC)
        }
}

func main() {
        http.HandleFunc("/time", func(w http.ResponseWriter, r *http.Request) {
                ctx := r.Context()
                t := bigBackendProcessing(ctx)
                fmt.Fprintln(w, "time:", t.String())
        })
        log.Fatal(http.ListenAndServe(":3333", nil))
}
```

context support in packages

- net/http - Aug 2016
- database/sql - Feb 2017
- lib/pq - Feb 2017 (postgresql db driver)
- gorp - Oct 2019 (library to simplify interaction with db)

After Oct 2019, it made sense to make sure that context is wired through all layers.

how to go about wiring DS layer

- Bottom-up or top-down? - recursive, can go anywhere in code
- We need some control too
- Simple regular expressions, gofmt, guru - not a solution

- We decided to write separate tool to parse, process and update out sourcecode using ***go*** AST and other packages

“Inside Google, we ended up not needing to build all this: Context was introduced early enough that Go users could plumb it manually where needed. I think a context plumbing tool could still be interesting and useful to other Go users. I'd love to see someone build it!”

- Sameer Ajmani (2017)

reading and writing go source code

- Working with source code is also readily available
  - go/ast, go/scanner, go/printer, go/parser
  - cmd/gofmt - available from day 1, uses these packages
- astutil

- Since 1.5 (Aug 2015), ***go*** is completely written in ***go*** (and some assembly)
  - cmd/compile, cmd/link, cmd/vet etc.

lets look at small AST code

```
package main
import (...)
var code = `package main
func main() {
	fmt.Println("Hello, World!")
}`
func main() {
	fset := token.NewFileSet()
	ast, err := parser.ParseFile(fset, "", code, parser.ParseComments)
	if err != nil {
		log.Fatalln(err)
	}
	res := astutil.Apply(ast, pre, post)
	buf := &bytes.Buffer{}
	if err = format.Node(buf, token.NewFileSet(), res); err != nil {
		log.Fatalf("error formatting new code: %v", err)
	}
	fmt.Println("===Original code===")
	fmt.Println(code)
	fmt.Println("===Modified code===")
	fmt.Println(buf.String())
}
```

```
func pre(c *astutil.Cursor) bool {
	switch v := c.Node().(type) {
	case *ast.BasicLit:
		v.Value = "pre::" + v.Value
	}
	return true
}
func post(c *astutil.Cursor) bool {
	switch v := c.Node().(type) {
	case *ast.BasicLit:
		v.Value = v.Value + "::post"
	}
	return true
}
```

phase 1

- Find out all the methods that don't accept context. Then for each method m
    - Text search whole source code (excluding vendor pkg) to look for references of this method - use name, argument count, return val count etc.
- For each file in the result, process it for method m
    - If we find method m as a function definition or declaration in interface; add context argument
    - If we find method m as a call (i.e. m is getting called at that point)
        - If call is from test, pass context.Background()
        - If call is from function/method that doesn't have access to context or has access to context, pass context.TODO()

Demo

phase 2

Once Phase 1 is done, we would have wired context - but still there will be some cases that use TODO. Here is how we can replace TODOs with real context. We can use AST writer for this.

For all the files in source code, for each file f

- Open the file and process every function one by one
  - If function doesn't have context argument, continue
  - If function has context argument, look for functions getting called from within this function body
    - Replace context.TODO with real context

Demo

Takeaways

- Don’t ignore Context, its important
- Programmatically reading/writing ***go*** code is not that difficult

Thanks!

references

- Discussion on go-nuts - Sameer Ajmani and others
- Online AST viewer - http://goast.yuroyoro.net/
- Samples
  - https://zupzup.org/ast-manipulation-go/
  - https://github.com/xdg-go/go-rewrap-errors