I couldn't think of a good title

but this talk is about newrelic, instrumenting go with AST rewrites, and some other stuff too

Hello, I'm Ehden

node.js agent engineer @ Contrast Security
Go enthusiast

@cixel (Gopher slack)
github.com/cixel
ehdens@gmail.com
ehden@contrastsecurity.com

CONTRAST SECURITY®

How to ~~learn~~ use a programming language

1. Use the language
2. Use Google
3. Repeat

How can I do my day job in this language?

— me, some time in 2017

Agents

1. user adds agent to an app
2. agent weaves itself into functions it cares about
    - request start/end
    - db queries
    - file system operations
3. agent changes function behavior (usually preserves semantics)

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

```
app.get('/time', (req, res) => {
  const flightNumber = req.query.flightNumber
  db.query('SELECT Departure,Arrival FROM Flights WHERE FlightNumber = ' + flightNumber, (err, rows) => {
    if (err) { res.send(err); return; }

    const dep = rows[0].Departure
    const arr = rows[0].Arrival

    res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
  })
})

// $ curl www.airliner.com/time?flightNumber=12345
// Flight 12345 will depart at 2:30pm and arrive at 3:30pm%
```

Terminology

source: where user controlled data comes from

```
const flightNumber = req.query.flightNumber
```

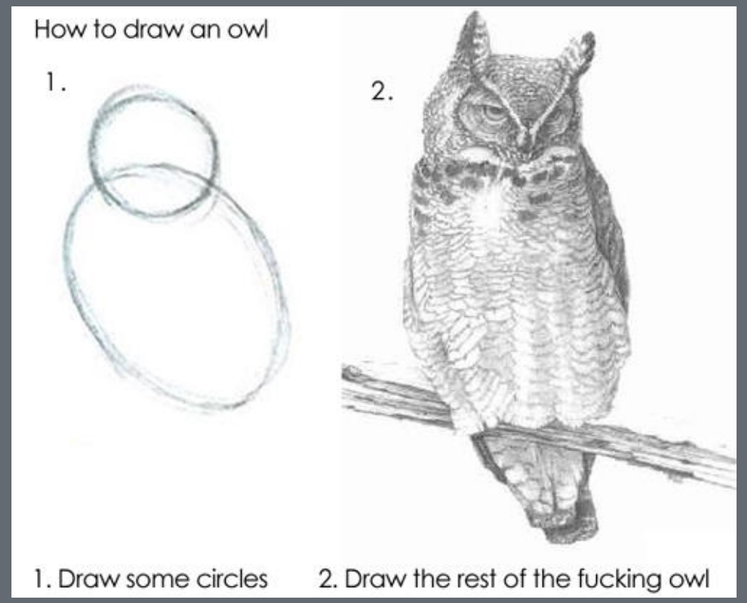
sink: where user controlled data ends up

```
db.query(/*...*/)
res.send(/*...*/)
```

Terminology (cont.)

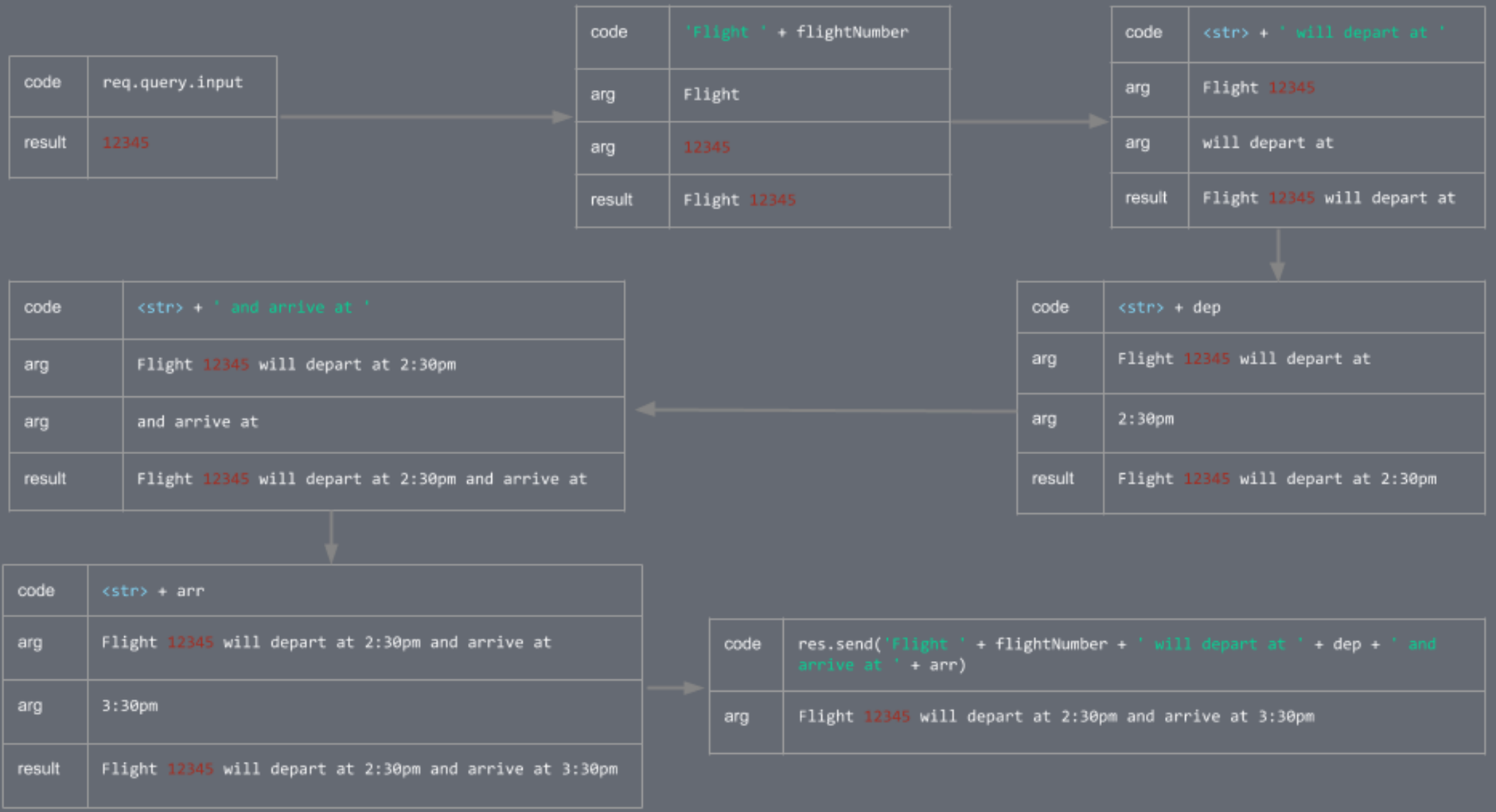
propagator: *everything in between*

```
'Flight ' + flightNumber
```



sql injection

XSS


code
req.query.input
result
12345
code
'Flight ' + flightNumber
arg
Flight
arg
12345
result
Flight 12345
code
<str> + ' will depart at '
arg
Flight 12345
arg
will depart at
result
Flight 12345 will depart at
code
<str> + dep
arg
Flight 12345 will depart at
arg
2:30pm
result
Flight 12345 will depart at 2:30pm
code
<str> + ' and arrive at '
arg
Flight 12345 will depart at 2:30pm
arg
and arrive at
result
Flight 12345 will depart at 2:30pm and arrive at
code
<str> + arr
arg
Flight 12345 will depart at 2:30pm and arrive at
arg
3:30pm
result
Flight 12345 will depart at 2:30pm and arrive at 3:30pm
code
res.send('Flight ' + flightNumber + ' will depart at ' + dep + ' and arrive at ' + arr)
arg
Flight 12345 will depart at 2:30pm and arrive at 3:30pm

How it works

How it works

- the answer is long and ~~boring~~technical
- ask me later if you're curious about javascript instrumentation

New Relic

Install the Go agent

In order to install the New Relic Go agent, you need a New Relic license key. Then, to install the agent:

1. From http://github.com/newrelic/go-agent , use your preferred process; for example:

```
go get github.com/newrelic/go-agent
```

2. Import the `github.com/newrelic/go-agent` package in your application.
3. Initialize the New Relic Go agent by adding the following in the `main` function or in an `init` block:

```
config := newrelic.NewConfig("YOUR_APP_NAME", "_YOUR_NEW_RELIC_LICENSE_
app, err := newrelic.NewApplication(config)
```

4. Instrument web transactions by wrapping standard HTTP requests in your app code. For example:

```
http.HandleFunc(newrelic.WrapHandleFunc(app, "/users", usersHandler))
```

5. Instrument other transactions you want to monitor.
6. Optional: Instrument segments for an extra level of timing detail.
7. Compile and deploy your application.

How it works (in Go)

How it works (in Go)

- AST rewriting
- wrap all function calls generically; event-based enter/leave hooks
- redefine function declarations to weave context through async boundaries
- WIP; still a few big hurdles to get over
- not open source, parts may become open source over time
- sorry

github.com/cixel/newrelic-init

Install the Go agent

In order to install the New Relic Go agent, you need a New Relic license key. Then, to install the agent:

1.
```
newrelic-init .
```

5. Instrument other transactions you want to monitor.
6. Optional: Instrument segments for an extra level of timing detail.
7. Compile and deploy your application.

How newrelic-init works

~~magic~~

1. adds `import github.com/newrelic/go-agent` (and other imports)
2. adds an `init` function which sets up monitoring (per the docs)
3. walks the package, wrapping invokations of `http.HandleFunc`

Abstract Syntax Trees

an abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language.

— *Wikipedia*

go/ast

```
a := 1 + 1
```

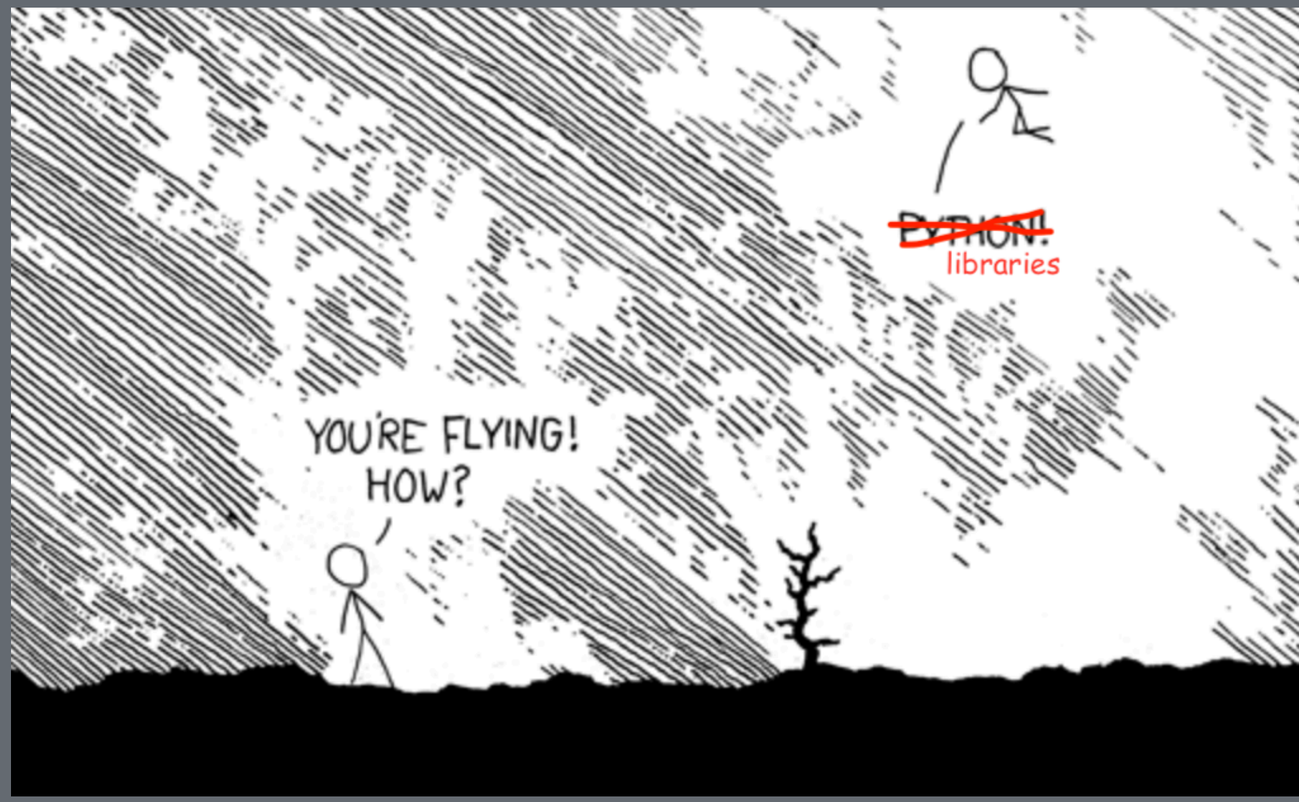
- defines AST types and interfaces
- closely related to:
  - `go/format`
  - `go/parser`
  - `go/printer`
  - `go/scanner`
  - `go/token`
  - `go/types`
  - `...`

```
0: *ast.AssignStmt {
.  Lhs: []ast.Expr (len = 1) {
.  .  0: *ast.Ident {
.  .  .  NamePos: example.go:6:2
.  .  .  Name: "a"
.  .  }
.  }
.  TokPos: example.go:6:4
.  Tok: :=
.  Rhs: []ast.Expr (len = 1) {
.  .  0: *ast.BinaryExpr {
.  .  .  X: *ast.BasicLit {
.  .  .  .  ValuePos: example.go:6:7
.  .  .  .  Kind: INT
.  .  .  .  Value: "1"
.  .  .  }
.  .  .  OpPos: example.go:6:9
.  .  .  Op: +
.  .  .  Y: *ast.BasicLit {
.  .  .  .  ValuePos: example.go:6:11
.  .  .  .  Kind: INT
.  .  .  .  Value: "1"
.  .  .  }
.  .  }
.  }
}
```

AST rewriting 101:

1. parse source code into AST
2. traverse tree, changing whatever you want
3. print the tree back out as source

```
import (
    "golang.org/x/tools/go/ast/astutil"
    "golang.org/x/tools/go/packages"
)
```



```
cfg := &packages.Config{
    Mode: packages.LoadAllSyntax,
}

pkgs, err := packages.Load(cfg, ".")
if err != nil {
    log.Fatal(err)
}

packages.Visit(pkgs, func(*packages.Package) bool {
    return true
}, func(p *packages.Package) {
    for _, file := range p.Syntax {
        astutil.Apply(file, nil, func(c *astutil.Cursor) bool {
            // ...
            return true
        })
    }
})
```

```go
func injectInit(pkg *packages.Package) {
    if pkg.Name != "main" {
        return
    }

    f := pkg.Syntax[0]
    // not sure if I should go through apply/replace here, or just modify the file directly
    astutil.Apply(f, func(c *astutil.Cursor) bool {
        file, ok := c.Node().(*ast.File)
        if !ok {
            return true
        }

        d := buildInitFunc()

        // add our package-wide 'app' variable to assign to
        // for use by wrappers
        file.Decls = append(file.Decls, appVar())

        // cannot c.Replace File nodes (see Cursor doc for reasoning) so we modify the File directly
        file.Decls = append(file.Decls, d)

        astutil.AddNamedImport(pkg.Fset, f, "newrelic", "github.com/newrelic/go-agent")
        astutil.AddImport(pkg.Fset, f, "os")

        return false

    }, nil)
}
```

```go
func injectInit(pkg *packages.Package) {
    if pkg.Name != "main" {
        return
    }

    f := pkg.Syntax[0]
    // not sure if I should go through apply/replace here, or just modify the file directly
    astutil.Apply(f, func(c *astutil.Cursor) bool {
        file, ok := c.Node().(*ast.File)
        if !ok {
            return true
        }

        d := buildInitFunc()

        // add our package-wide 'app' variable to assign to
        // for use by wrappers
        file.Decls = append(file.Decls, appVar())

        // cannot c.Replace File nodes (see Cursor doc for reasoning) so we modify the File directly
        file.Decls = append(file.Decls, d)

        astutil.AddNamedImport(pkg.Fset, f, "newrelic", "github.com/newrelic/go-agent")
        astutil.AddImport(pkg.Fset, f, "os")

        return false

    }, nil)
}
```

```go
func injectInit(pkg *packages.Package) {
    if pkg.Name != "main" {
        return
    }

    f := pkg.Syntax[0]
    // not sure if I should go through apply/replace here, or just modify the file directly
    astutil.Apply(f, func(c *astutil.Cursor) bool {
        file, ok := c.Node().(*ast.File)
        if !ok {
            return true
        }

        d := buildInitFunc()

        // add our package-wide 'app' variable to assign to
        // for use by wrappers
        file.Decls = append(file.Decls, appVar())

        // cannot c.Replace File nodes (see Cursor doc for reasoning) so we modify the File directly
        file.Decls = append(file.Decls, d)

        astutil.AddNamedImport(pkg.Fset, f, "newrelic", "github.com/newrelic/go-agent")
        astutil.AddImport(pkg.Fset, f, "os")

        return false

    }, nil)
}
```

```go
func injectInit(pkg *packages.Package) {
    if pkg.Name != "main" {
        return
    }

    f := pkg.Syntax[0]
    // not sure if I should go through apply/replace here, or just modify the file directly
    astutil.Apply(f, func(c *astutil.Cursor) bool {
        file, ok := c.Node().(*ast.File)
        if !ok {
            return true
        }

        d := buildInitFunc()

        // add our package-wide 'app' variable to assign to
        // for use by wrappers
        file.Decls = append(file.Decls, appVar())

        // cannot c.Replace File nodes (see Cursor doc for reasoning) so we modify the File directly
        file.Decls = append(file.Decls, d)

        astutil.AddNamedImport(pkg.Fset, f, "newrelic", "github.com/newrelic/go-agent")
        astutil.AddImport(pkg.Fset, f, "os")

        return false

    }, nil)
}
```

```go
func injectInit(pkg *packages.Package) {
    if pkg.Name != "main" {
        return
    }

    f := pkg.Syntax[0]
    // not sure if I should go through apply/replace here, or just modify the file directly
    astutil.Apply(f, func(c *astutil.Cursor) bool {
        file, ok := c.Node().(*ast.File)
        if !ok {
            return true
        }

        d := buildInitFunc()

        // add our package-wide 'app' variable to assign to
        // for use by wrappers
        file.Decls = append(file.Decls, appVar())

        // cannot c.Replace File nodes (see Cursor doc for reasoning) so we modify the File directly
        file.Decls = append(file.Decls, d)

        astutil.AddNamedImport(pkg.Fset, f, "newrelic", "github.com/newrelic/go-agent")
        astutil.AddImport(pkg.Fset, f, "os")

        return false

    }, nil)
}
```

```
// creates the node: `var newrelicApp newrelic.Application`
func appVar() *ast.GenDecl {
    d := &ast.GenDecl{
        Tok: token.VAR,
        Specs: []ast.Spec{
            &ast.ValueSpec{
                Names: []*ast.Ident{ast.NewIdent("newrelicApp")},
                Type: &ast.SelectorExpr{
                    X:   ast.NewIdent("newrelic"),
                    Sel: ast.NewIdent("Application"),
                },
            },
        },
    }

    return d
}
```

Wrapping `http.HandleFunc`

```
if doesFuncTypeMatch(call, httpHandleFuncType, pkg.TypesInfo) {
    wrapped := &ast.CallExpr{
        Fun: c.Fun,
        Args: []ast.Expr{
            &ast.CallExpr{
                Fun: &ast.SelectorExpr{
                    X:   ast.NewIdent("newrelic"),
                    Sel: ast.NewIdent("WrapHandleFunc"),
                },
                Args: []ast.Expr{
                    ast.NewIdent("newrelicApp"),
                    c.Args[0],
                    c.Args[1],
                },
            },
        },
    }
    cursor.Replace(wrapped)
}
```

Why not search and replace?

```
var h = http.HandleFunc
h("/hi", handler)
```

```
func doesFuncTypeMatch(call *ast.CallExpr, sig string, info *types.Info) bool {
    typ, ok := info.TypeOf(call.Fun).(*types.Signature)
    if !ok {
        return false
    }

    if typ.String() != sig {
        return false
    }

    return true
}
```

```
func(pattern string, handler func(net/
http.ResponseWriter, *net/http.Request))
```

Heck

go/ast: Free-floating comments are single-biggest issue when manipulating the AST #20744

Open **griesemer** opened this issue on Jun 21, 2017 · 16 comments

griesemer commented on Jun 21, 2017 — Contributor

Reminder issue.

The original design of the go/ast makes it very difficult to update the AST and retain correct comment placement for printing. The culprit is that the comments are "free-floating" and printed based on their positions relative to the AST nodes. (The go/ast package is one of the earliest Go packages in existence, and this is clearly a major design flaw.)

Instead, a newer/updated ast should associate all comments (not just Doc and Comment comments) with nodes. That would make it much easier to manipulate a syntax tree and have comments correctly attached. The main complexity with that approach is the heuristic that decides which comments are attached to which nodes.

👍 11 🎉 3 ❤️ 2

The Comments Problem

```
//go:noescape
func f() {
    // prints "i like cats"
    fmt.Println(/*FIXME maybe this should be cats?*/"i like dogs")

    // some people don't like dogs

    fmt.Println("sorry if you don't like dogs")
}
```

- not all comments can be associated with a node
- comment position is very important
- when AST is changed, it is unclear what to do with 'lossy' comments

The Comments Workaround: `dst`

- github.com/dave/dst
- 'decorated syntax tree'
- adds 'decorations' to **every** node
- `%s/ast/dst/g`

Testing

- golden files are super duper useful for testing AST rewrites
- spend time setting up helpers
- use real packages as test data
- use test names to manage location of test input data/goldens

Demo

Beyond new-relic - untargetted rewrites

```
http.HandleFunc("/edit/", makeHandler(editHandler))
// -->
http.HandleFunc(newrelic.WrapHandleFunc(newrelicApp, "/edit/", makeHandler(editHandler)))
```

```
func() {
    var _arg0 string = "/edit/"
    var _arg1 http.HandlerFunc = func() http.HandlerFunc {
        var _arg0 func(http.ResponseWriter, *http.Request, string) = editHandler
        notContrast.Emit(
            notContrast.PRE,
            "bitbucket.org/contrastsecurity/go-test-apps/wiki",
            "main",
            "makeHandler",
            []interface {}{&_arg0},
            nil,
            nil,
        )
        _res0 := makeHandler(_arg0)
        notContrast.Emit(notContrast.POST, "bitbucket.org/contrastsecurity/go-test-apps/wiki", "main", "makeHandler", []interface {}{&_arg0}, []interface {}{&_res0}, nil)
        return _res0
    }()
    notContrast.Emit(notContrast.PRE, "net/http", "main", "HandleFunc", []interface {}{&_arg0, &_arg1}, nil, nil)
    http.HandleFunc(_arg0, _arg1)
    notContrast.Emit(notContrast.POST, "net/http", "main", "HandleFunc", []interface {}{&_arg0, &_arg1}, nil, nil)
    return
}()
```

Other Fun Stuff We Maybe™ Do

Request Context

```
func someFunc(s string) {
    go func() {
        database.query(s)
    }()
}

func injected_someFunc(ctxt *notContrast.Context, s string) {
    // do something with ctxt
    notContrast.Emit(/*...*/, ctxt)

    // direct copy of someFunc
    go func() {
        database.query(s)
    }()
    //
}
```

Package Wrapping

```
// main
import (
    "notContrast/http"
)

// notContrast/http
import "http"
func HandleFunc(pattern string, handler func(ResponseWriter, *Request)) {
    // do whatever I want
    return http.HandleFunc(patter, handler)
}
```

Contact:

- @cixel (Gopher slack)
- github.com/cixel
- ehdens@gmail.com
- ehden@contrastsecurity.com

Image Credits:

- Contrast Security
- buttersafe - The Detour
- virus caught red-handed
- new relic go docs
- tree (Slate article)
- xkcd - Python
- gopher