Getting rid of uber/dig

Vladimir Fetisov

What is uber/dig?

Where you can find it: go.uber.org/dig

What is it: a Dependency Injection framework

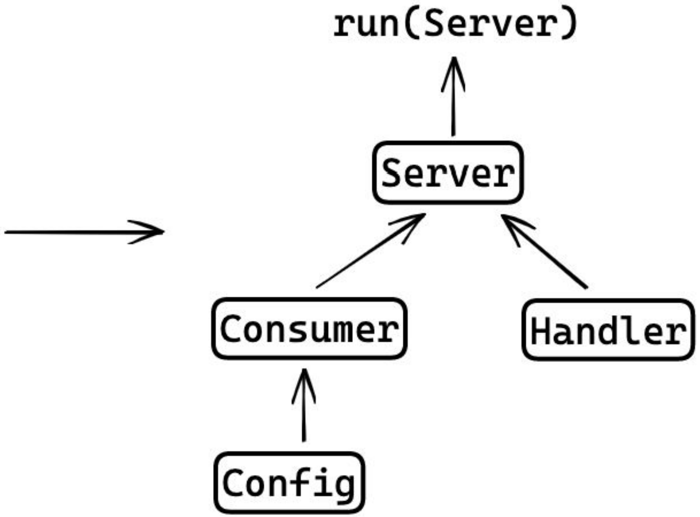
don't mix it up with Dependency Inversion

Why I don't like DI frameworks

- I don't like magic
- I don't like boilerplate

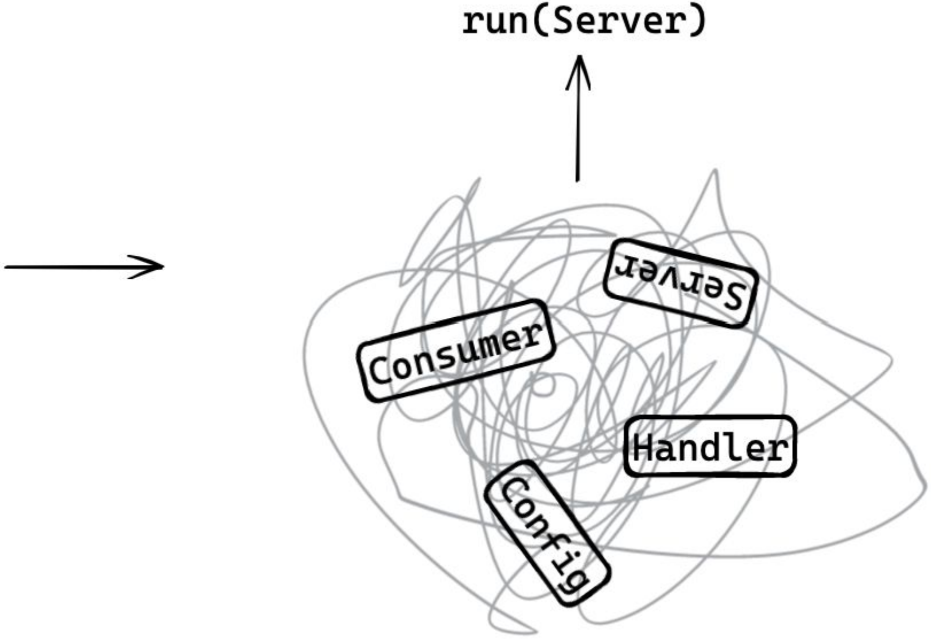
I don't like magic

```
func main() {
    conf := NewConfig()
    consumer := NewConsumer(conf)
    handler := NewHandler()
    server := NewServer(consumer, handler)

    run(server)
}

func run(server Server) {
    server.Run()
     ...
}
```

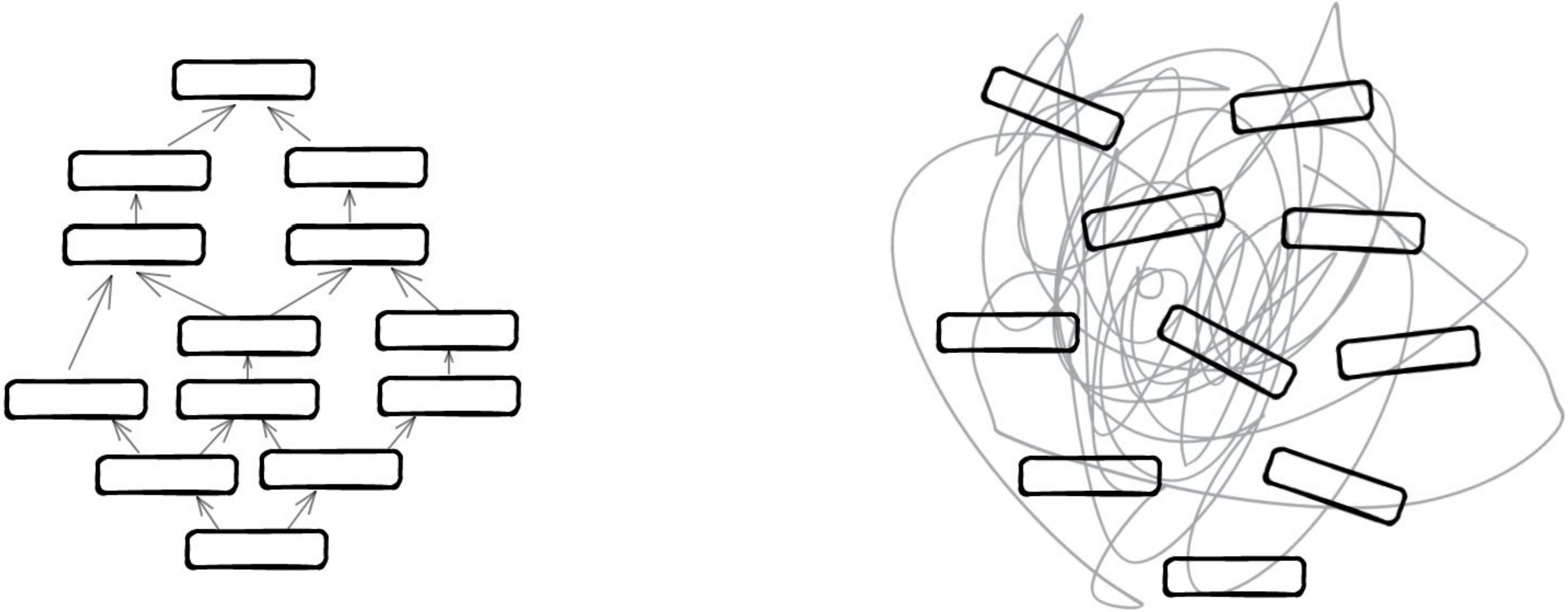


I don't like magic

```
func main() {
    c := dig.New()

    c.Provide(NewHandler)
    c.Provide(NewConfig)
    c.Provide(NewServer)
    c.Provide(NewConsumer)

    c.Invoke(run)
}

func run(server Server) {
    server.Run()
     ...
}
```



I don't like magic



The bigger the project, the less clarity

I don't like boilerplate

dig.Provide wrappers

```
func init() {
  addServiceProvider(
    sqsClientProvider,
    S3ClientProvider,
    snsClientProvider,
    emailClientProvider,
     ...
  )
}
```

declaration of dig.In / dig.Out wrappers

```
type EndpointParams struct {
  dig.In
  Endpoints []Endpoint `group:"endpoint"`
}

type EndpointResult struct {
  dig.Out
  Endpoint Endpoint `group:"endpoint"`
}
```

return of dig.Out structs

```
return EndpointResult{
  Endpoint: Endpoint{
    Handler: handler,
     ...
}
```

How I tried to remove the dig and make sure I hadn't forgotten anything to provide

Plan A:

1. Run the program with dig and build a callgraph
2. Manually remove dig from the code, run the program again, build the second callgraph
3. Compare callgraphs and make sure that calls are the same

golang.org/x/tools/go/callgraph

github.com/ofabry/go-callvis

How I tried to remove the dig and make sure I hadn't forgotten anything to provide

didn't work

Plan A:

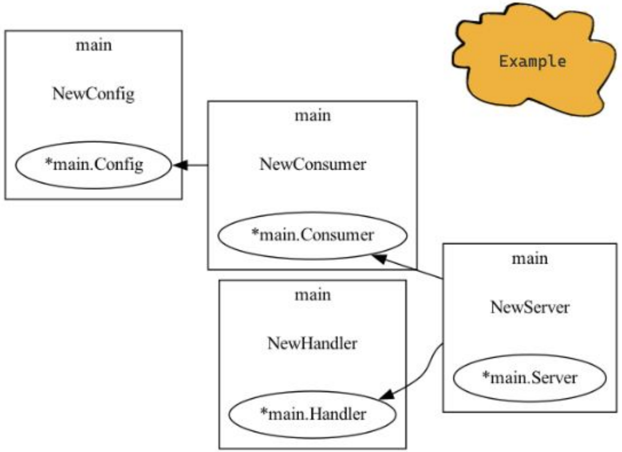
1. ~~Run the program with dig and build a callgraph~~
2. ~~Manually remove dig from the code, run the program again, build the second callgraph~~
3. ~~Compare callgraphs and make sure that calls are the same~~

1. It is very time-consuming to rewrite the main() function using just constructors
2. Comparing graphs is almost impossible. They're too big

How I tried to remove the dig and make sure I hadn't forgotten anything to provide

Plan B:

1. Use dig's visualisation package to build dependencies graph in Graphviz format
2. Parse the Graphviz file and generate / manually rewrite code



How I tried to remove the dig and make sure I hadn't forgotten anything to provide

didn't work

Plan B:

1. ~~Use dig's visualisation package to build dependencies graph in Graphviz format~~
2. ~~Parse the Graphviz file and generate / manually rewrite code~~

1. The callgraph is better, but doesn't have enough information to generate the code

How I tried to remove the dig and make sure I hadn't forgotten anything to provide

Perfect

Plan C:

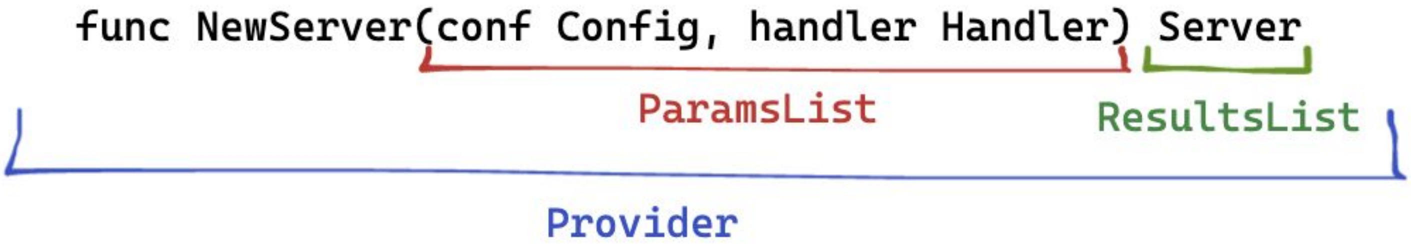
Use dig's runtime. Print the initialisation of constructors along the dependency tree traversal path

How does uber/dig work?

Provider - a functions that creates/provides a value of a certain type

ParamsList - a list of constructor's arguments

ResultsList - a list of values returned by constructor



```go
func NewServer(conf Config, handler Handler) Server
```

How does uber/dig work?

```go
type provider interface {
    // ParamList returns information about the direct dependencies of this
    // constructor.
    ParamList() paramList

    // ResultList returns information about the values produced by this
    // constructor.
    ResultList() resultList

    // Calls the underlying constructor, reading values from the
    // containerStore as needed.
    Call(containerStore) error
}
```

provider interface has the only implementation:

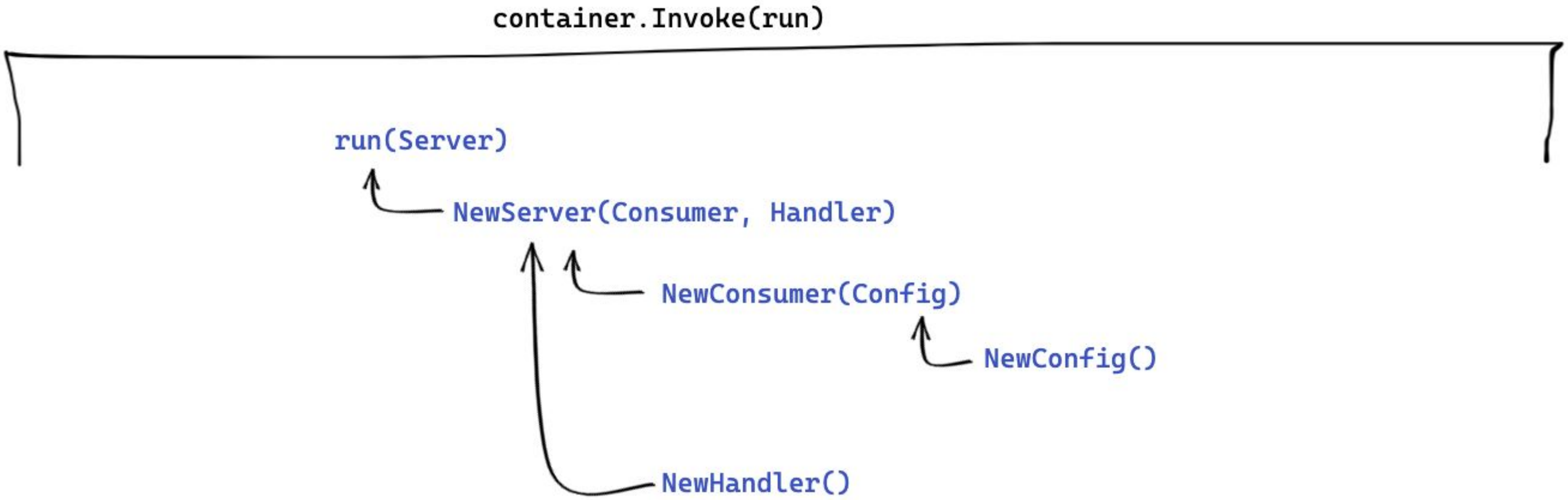
* it's basically a wrapper around functions you pass on to container.Provider

How does uber/dig work?

```go
type param interface {
    // Build this dependency and any of its dependencies from the provided
    // Container
    Build(store containerStore) (reflect.Value, error)

    // other fields
}
```

param interface has 3 implementations:
* paramSingle - for any single value
* paramObject - for types embedding dig.Out
* paramGroupedSlice - for fields with the `group` tag

How does uber/dig work?



container.Invoke(run)

run(Server)

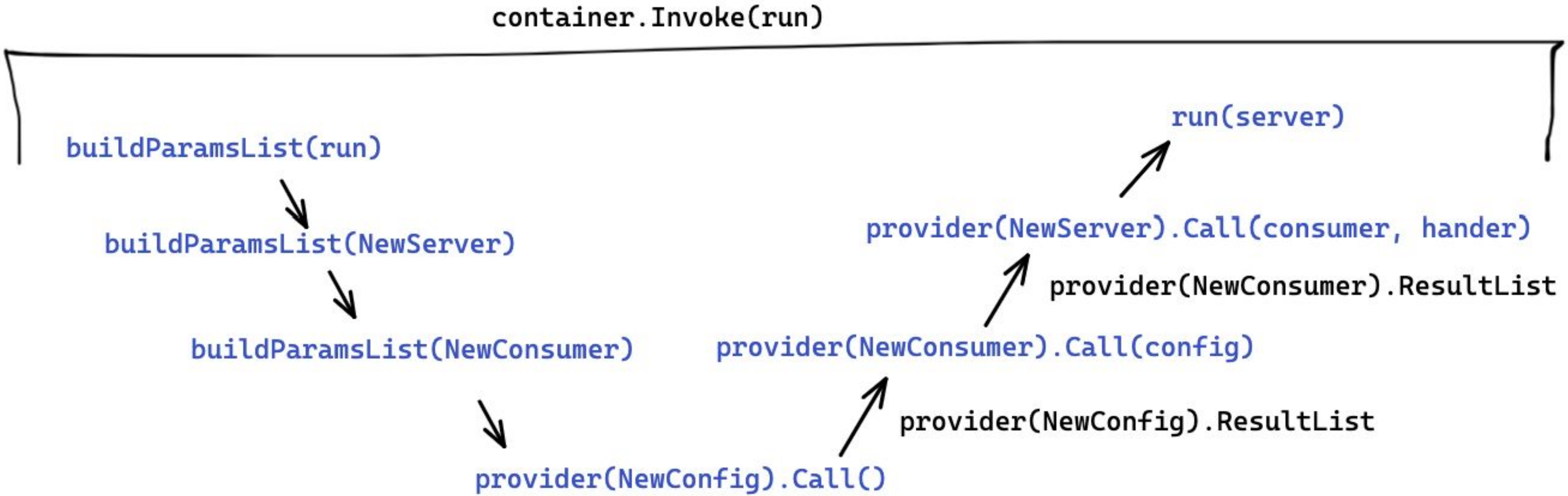
NewServer(Consumer, Handler)

NewConsumer(Config)

NewConfig()

NewHandler()

How does uber/dig work?



container.Invoke(run)

buildParamsList(run)

buildParamsList(NewServer)

buildParamsList(NewConsumer)

provider(NewConfig).Call()

provider(NewConfig).ResultList

provider(NewConsumer).Call(config)

provider(NewConsumer).ResultList

provider(NewServer).Call(consumer, hander)

run(server)

What have I done?

I put "prints" in the following places:

- When dig calls param.Build()
    - To print statements when we create dig.In, dig.Out etc.
- When dig calls provider.Call()
    - To print functions calls

Therefore, every time dig calls a provider or filling out a dig.Out it also prints a statement

anti-dig

And created a library which is a drop-in replacement for go.uber.org/dig and named it anti-dig

🌐https://github.com/3timeslazy/anti-dig

How to use it?

1. Replace "go.uber.org/dig" ⇒ dig "github.com/3timeslazy/anti-dig"
2. Run your program
3. Instead of your code running you'll see a generated file with your code working without dig. Just plain constructors

The result

```
import dig "github.com/3timeslazy/anti-dig

func main() {
    c := dig.New()

    c.Provide(NewHandler)
    c.Provide(NewConfig)
    c.Provide(NewServer)
    c.Provide(NewConsumer)

    c.Invoke(run)
}

func run(server Server) {
    server.Run()
     ...
}
```

→

```
package main

func Provide() (*Server) {
    var2 := NewHandler()
    var4 := NewConfig()
    var3, err := NewConsumer(var4)
    if err != nil {
        return nil
    }
    var1 := NewServer(var2, var3)
    return var1
}
```

Let's make out example more interesting

```go
func main() {
    c := dig.New()

    c.Provide(NewHandler, dig.Name("handler"))
    c.Provide(NewServer)
    c.Provide(NewConsumer, dig.Name("consumer"))

    c.Invoke(run)
}

type ServerParams struct {
    dig.In
    Handler  *Handler  `name:"handler"`
    Consumer *Consumer `name:"consumer"`
}
```

1. NewHandler now returns an error

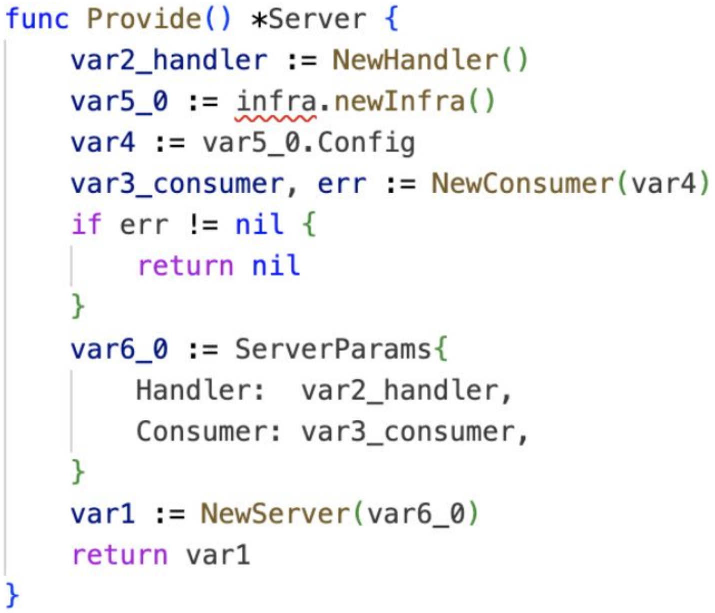
2. NewServer accepts ServerParams

3. newInfra return a value of type Infra

```go
type Infra struct {
    dig.In
    Config Config
}
```

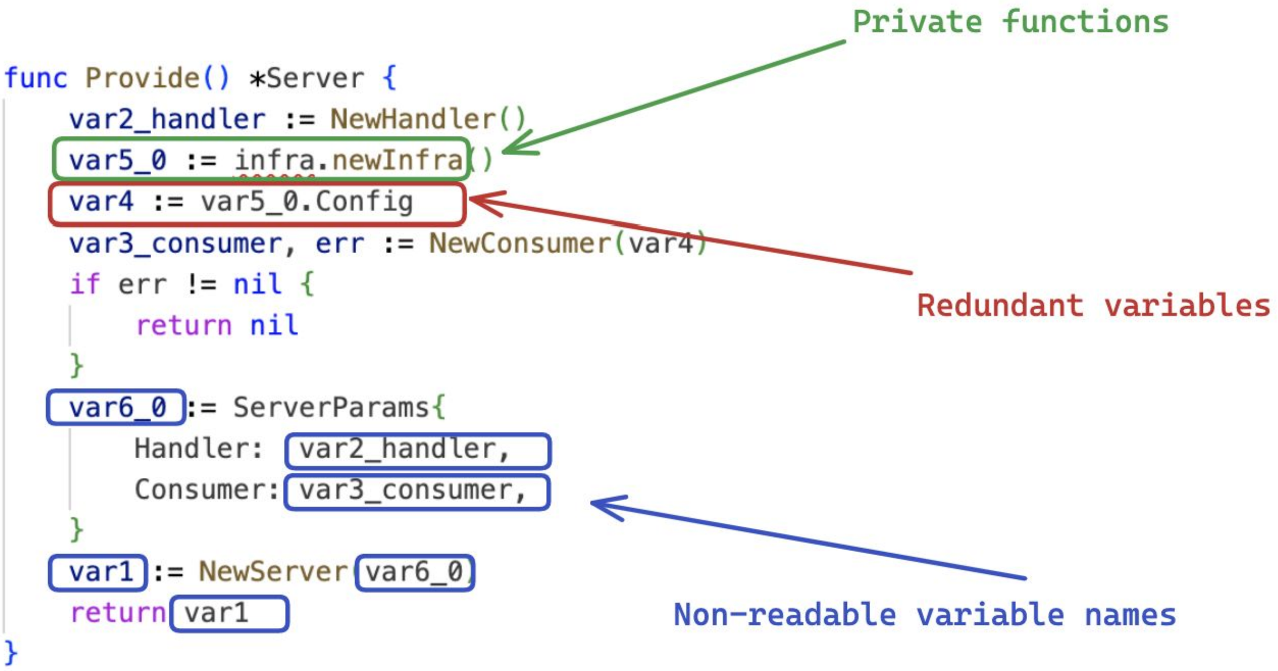
4. newInfra is provided outside the main package

Generated code



```go
func Provide() *Server {
    var2_handler := NewHandler()
    var5_0 := infra.newInfra()
    var4 := var5_0.Config
    var3_consumer, err := NewConsumer(var4)
    if err != nil {
        return nil
    }
    var6_0 := ServerParams{
        Handler:  var2_handler,
        Consumer: var3_consumer,
    }
    var1 := NewServer(var6_0)
    return var1
}
```

Generated code



```go
func Provide() *Server {
    var2_handler := NewHandler()
    var5_0 := infra.newInfra()
    var4 := var5_0.Config
    var3_consumer, err := NewConsumer(var4)
    if err != nil {
        return nil
    }
    var6_0 := ServerParams{
        Handler:  var2_handler,
        Consumer: var3_consumer,
    }
    var1 := NewServer(var6_0)
    return var1
}
```

Private functions

Redundant variables

Non-readable variable names

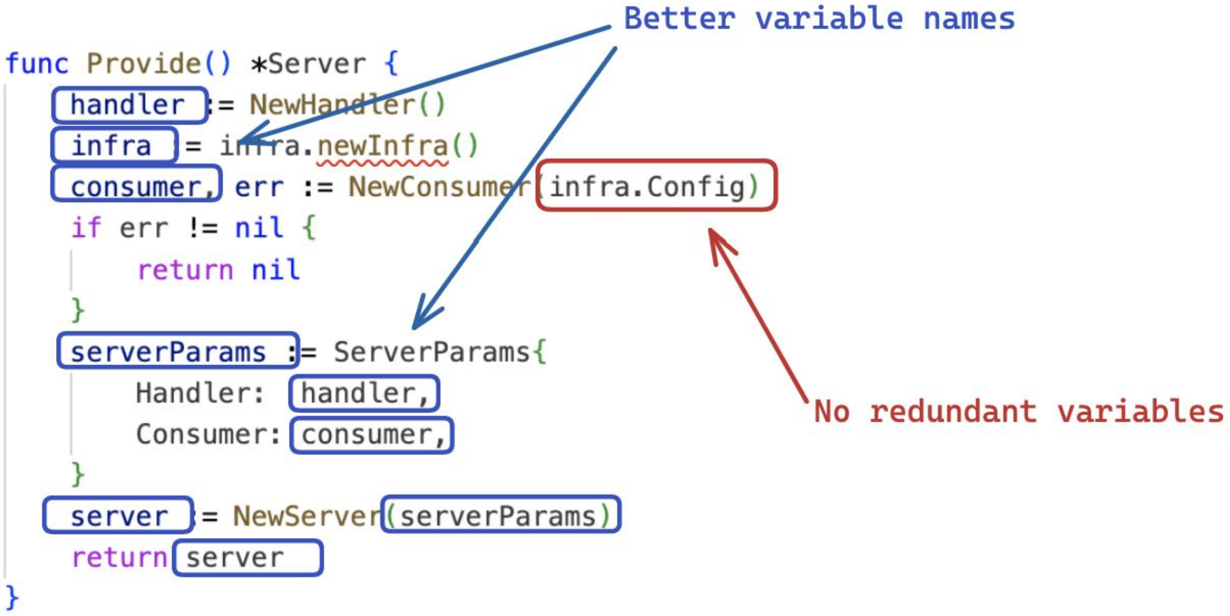
How to make the generated code better?

`Use "golang.org/x/tools/go/ast/astutil" !`

Generate the code → Parse the AST → Better generated code

- Remove redundant variables
- Make readable variable names
- And some more ...

How to make the generated code better?

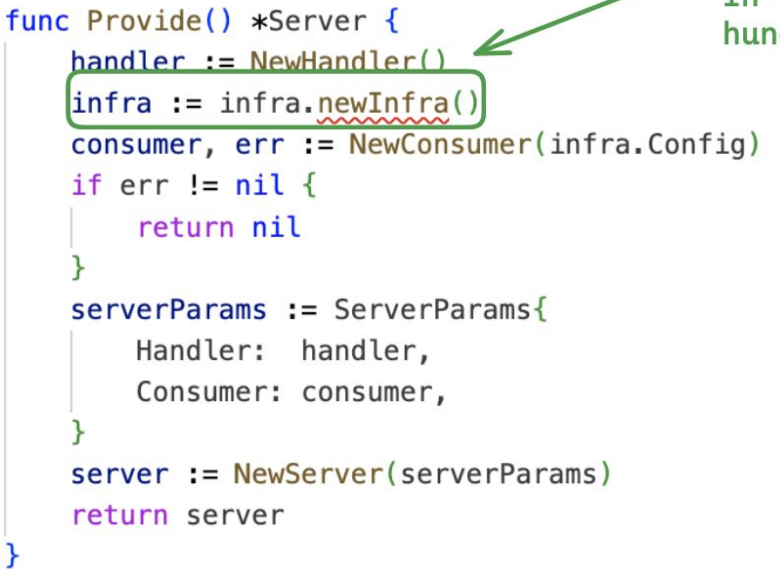


```go
func Provide() *Server {
    handler := NewHandler()
    infra := infra.newInfra()
    consumer, err := NewConsumer(infra.Config)
    if err != nil {
        return nil
    }
    serverParams := ServerParams{
        Handler:  handler,
        Consumer: consumer,
    }
    server := NewServer(serverParams)
    return server
}
```

Better variable names

No redundant variables

How to make the generated code better?



```go
func Provide() *Server {
	handler := NewHandler()
	infra := infra.newInfra()
	consumer, err := NewConsumer(infra.Config)
	if err != nil {
		return nil
	}
	serverParams := ServerParams{
		Handler:  handler,
		Consumer: consumer,
	}
	server := NewServer(serverParams)
	return server
}
```

Still has this private func

In the real service there may be hundreds of private functions

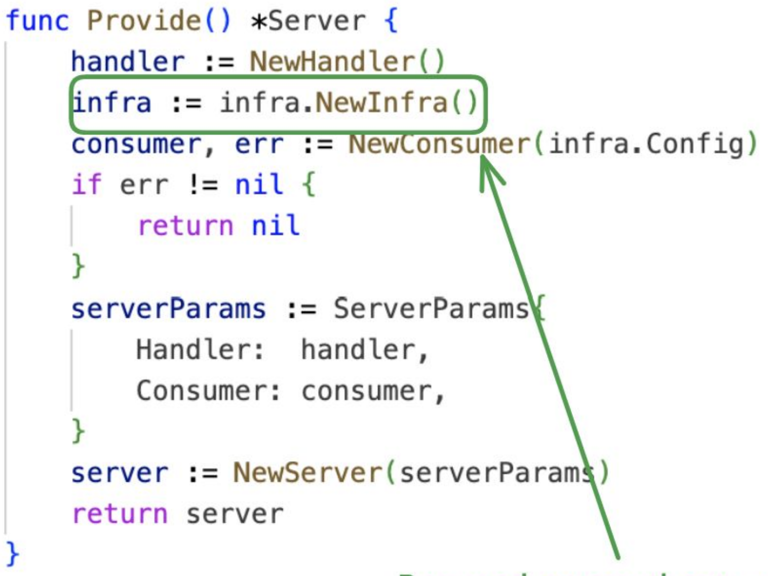
How to make the generated code better?

`Use "github.com/golang/tools/tree/master/refactor/rename" !`

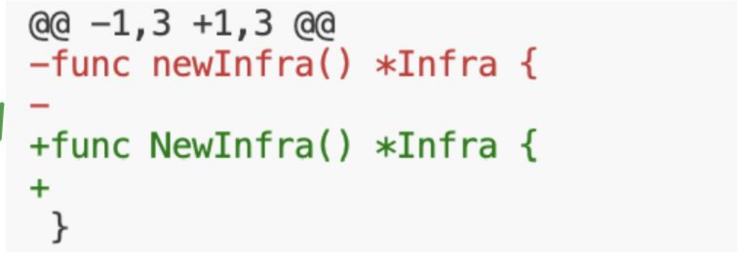
`golang/tools/rename` is a CLI that performs
type-safe renaming of identifiers in Go source code

So, I changed it a bit:
- Got this tool to rename a few identifiers at a time
- Made a couple of optimisations such as reused large variables, parse files only once, etc.

How to make the generated code better?



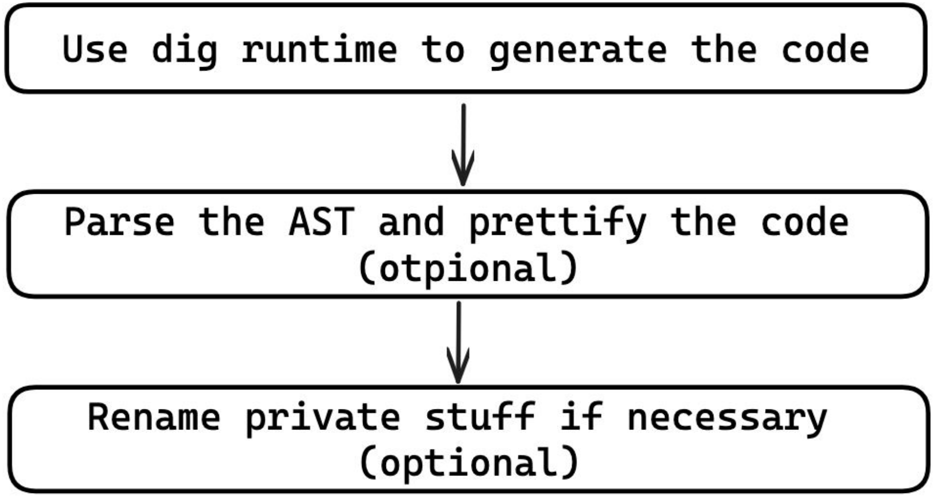
```go
func Provide() *Server {
    handler := NewHandler()
    infra := infra.NewInfra()
    consumer, err := NewConsumer(infra.Config)
    if err != nil {
        return nil
    }
    serverParams := ServerParams{
        Handler:  handler,
        Consumer: consumer,
    }
    server := NewServer(serverParams)
    return server
}
```



```diff
@@ -1,3 +1,3 @@
-func newInfra() *Infra {
-
+func NewInfra() *Infra {
+
 }
```

Renamed everywhere within the project

Overall flow



Use dig runtime to generate the code

Parse the AST and prettify the code (otpional)

Rename private stuff if necessary (optional)

Not everything can be generated

Not yet supported:

- dig.As option
- functions returning many values
- anonymous functions as providers
- Perhaps, something else I'm not aware of

Production use case

What we had?

A service with:

* ~100 constructors
* ~80 types used to startup the service

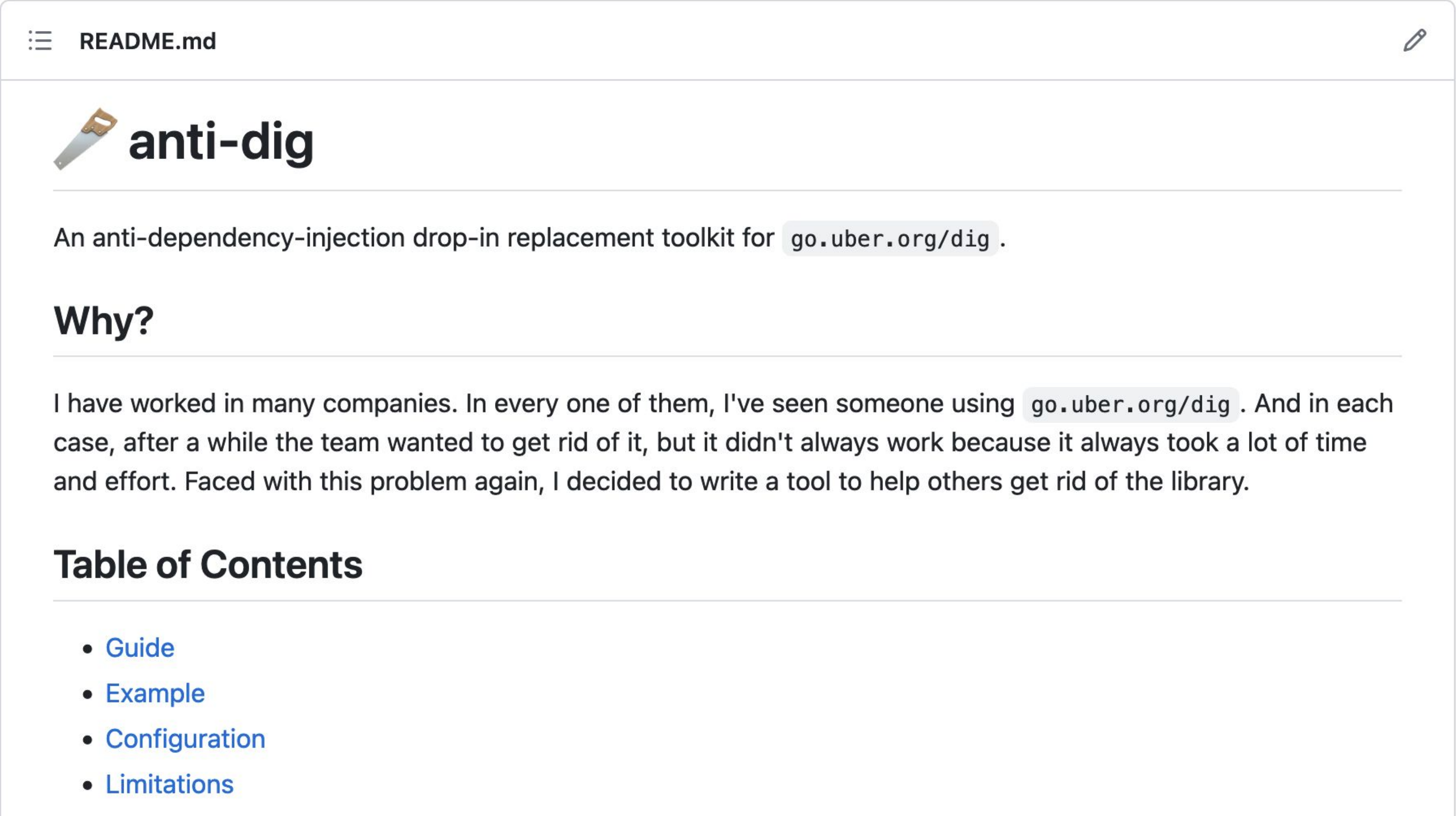
What we did?

* Generated the code
* Tested it on staging
* Deployed on production

As a result

+ No magic in the code
+ Removed a lot of boilerplate
+ Removed unused constructors and types previously hidden behind dig's magic
+ We can navigate the codebase quickly
+ One big "main.go" file much easier to comprehend

− One big "main.go" might look ugly



README.md

🪚 anti-dig

An anti-dependency-injection drop-in replacement toolkit for `go.uber.org/dig`.

Why?

I have worked in many companies. In every one of them, I've seen someone using `go.uber.org/dig`. And in each case, after a while the team wanted to get rid of it, but it didn't always work because it always took a lot of time and effort. Faced with this problem again, I decided to write a tool to help others get rid of the library.

Table of Contents

Thank you!

github.com/3timeslazy/anti-dig

@3timeslazy@techhub.social

vladimir.fetisov@pm.me