Key to Efficiency

Insights into scaffold code generation and project engineering practices with CWGO

Content

Part 01

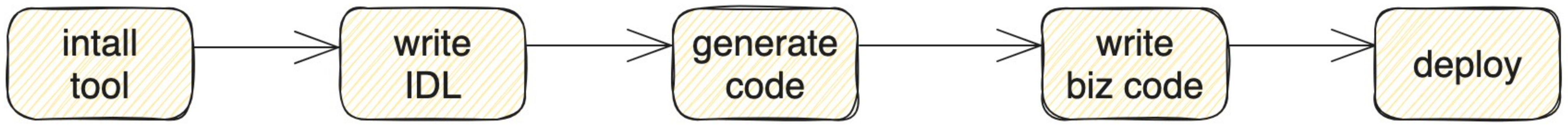
Overview

CloudWeGo Microservice Frameworks

CloudWeGo provides two microservice frameworks for the Golang.

- Hertz
  - Golang HTTP Microservice Framework
  - High Performance
  - High Extensibility
- Kitex
  - Golang RPC Microservice Framework
  - High Performance
  - Multi-Message Protocol: Thrift、Kitex-Protobuf、gRPC

Develop Workflow

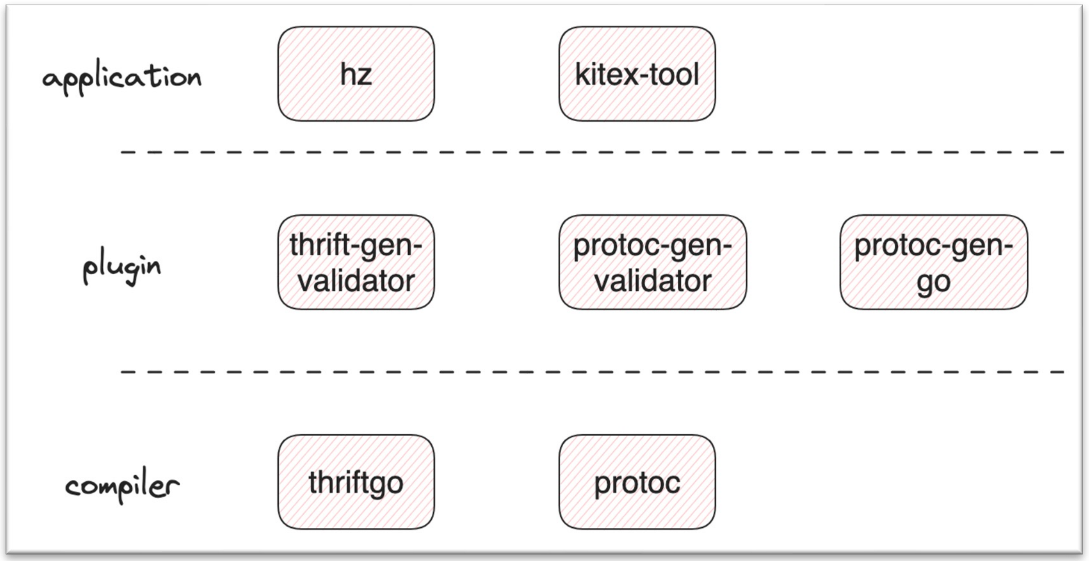


- Install Tool: install framework scaffold tool
- Write IDL: write protobuf/thrift IDL, prescribe interface specification
- Generate Code: based IDL and scaffold tool to generate framework basic code
- Write Biz Code: complete business code
- Deploy: deploy and launch

Develop Workflow

CloudWeGo Tool:

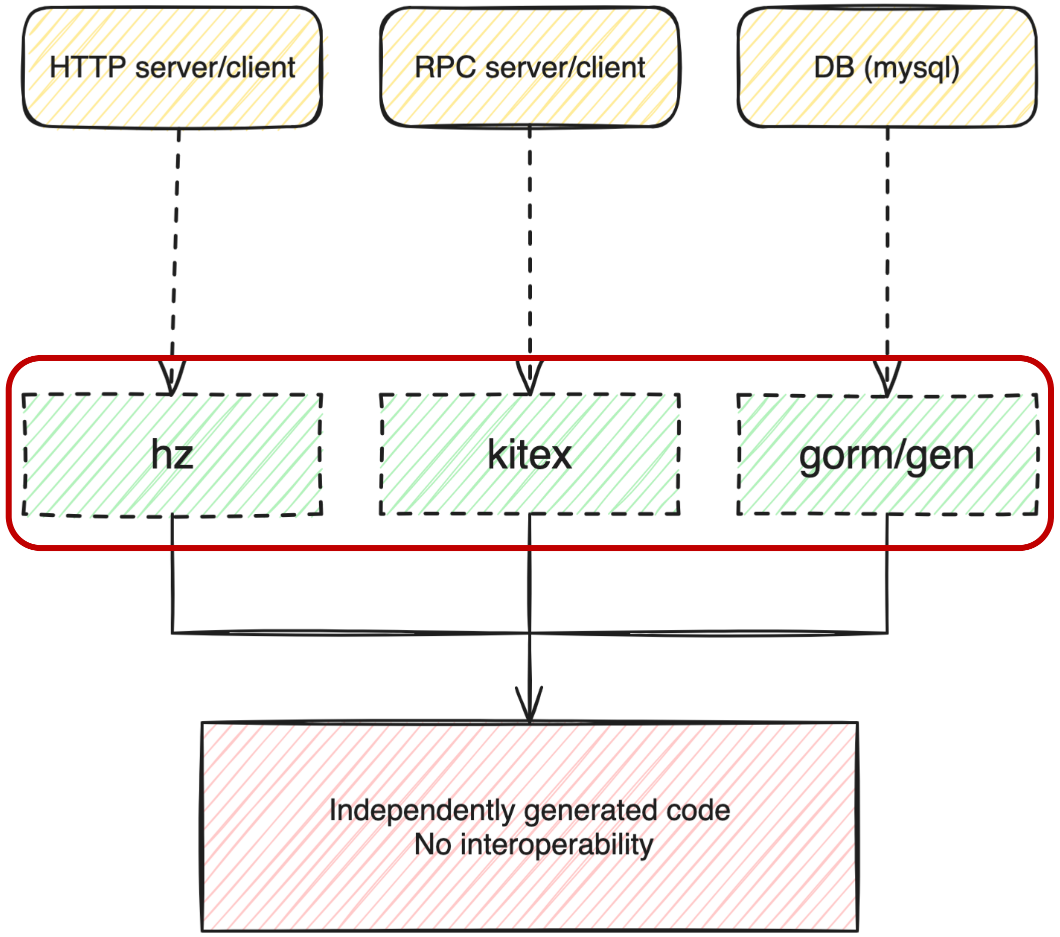
- Hz: code scaffold tool for the Hertz framework
- Kitex-tool: code scaffold tool for the Kitex framework
- Validator: request parameter validating tool
- Compiler: thrift/protobuf compiler



Challenge

Multiple tools, high learning costs

- A lot of tools need to be installed
- Hz and Kitex has difference in **usage**
- Hz and Kitex has difference in **IDL specification**



Challenge

Lack of a complete engineering template

- Templates for generating projects are too simple

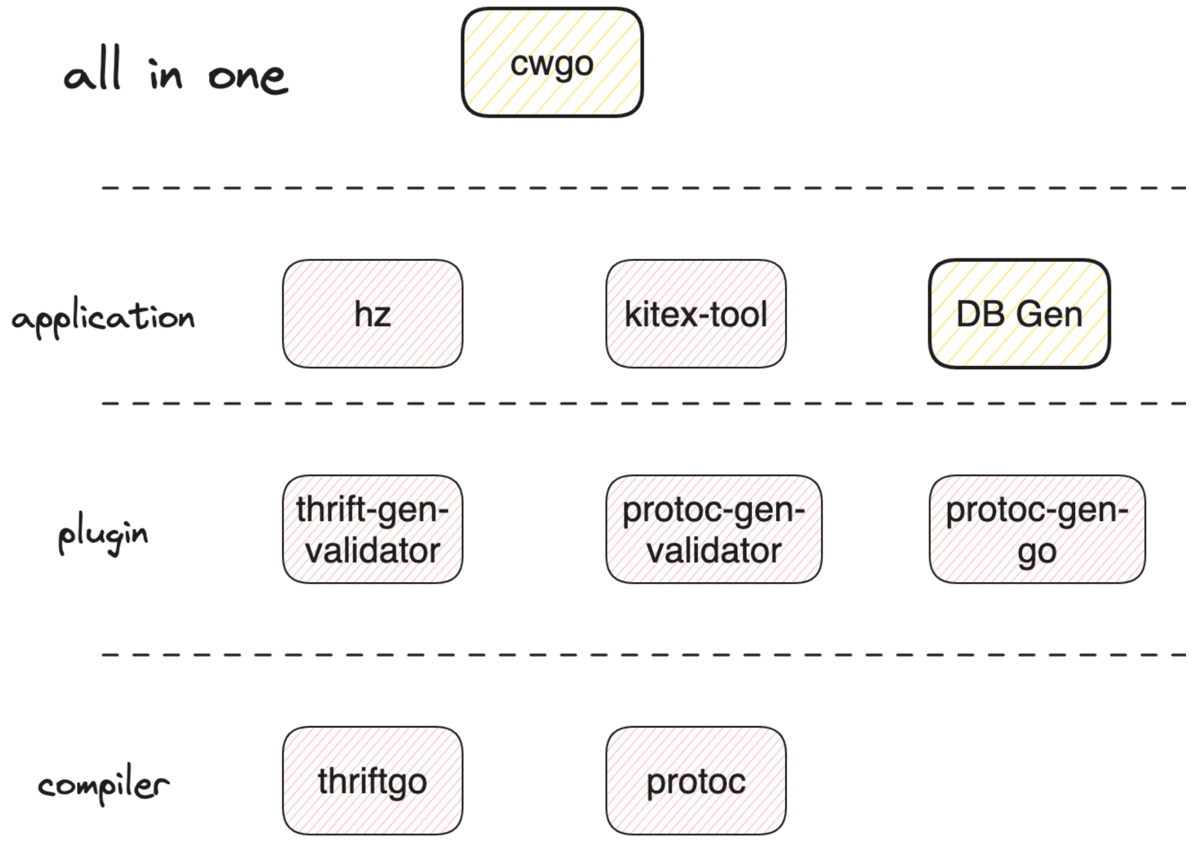
Lack of CRUD code generation

- Unable to generate CRUD code commonly used by users

Solution

Cwgo – CloudWeGo All In One

- Tool Integration
- Capability Abstraction
- Enhance Template
- Feature Enrichment



Part 02

CWGO: Key to Efficiency

Installation & Usage

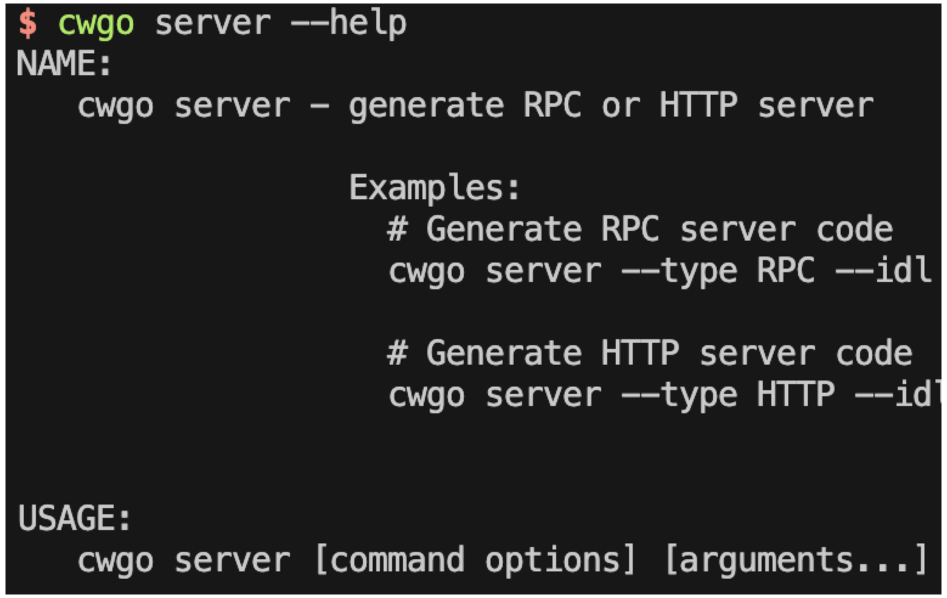
```
go install github.com/cloudwego/cwgo@latest
```

Server

- Generate server code based on IDL
- Server can be HTTP or RPC

Client

- Generate client code based on IDL
- Client can be HTTP or RPC

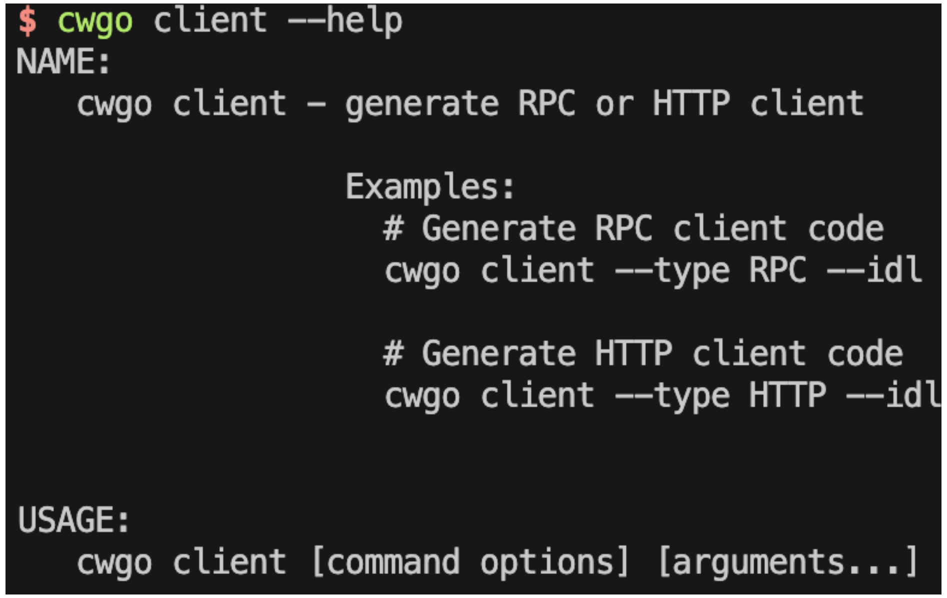


```
$ cwgo server --help
NAME:
   cwgo server - generate RPC or HTTP server

                 Examples:
                   # Generate RPC server code
                   cwgo server --type RPC --idl

                   # Generate HTTP server code
                   cwgo server --type HTTP --id


USAGE:
   cwgo server [command options] [arguments...]
```



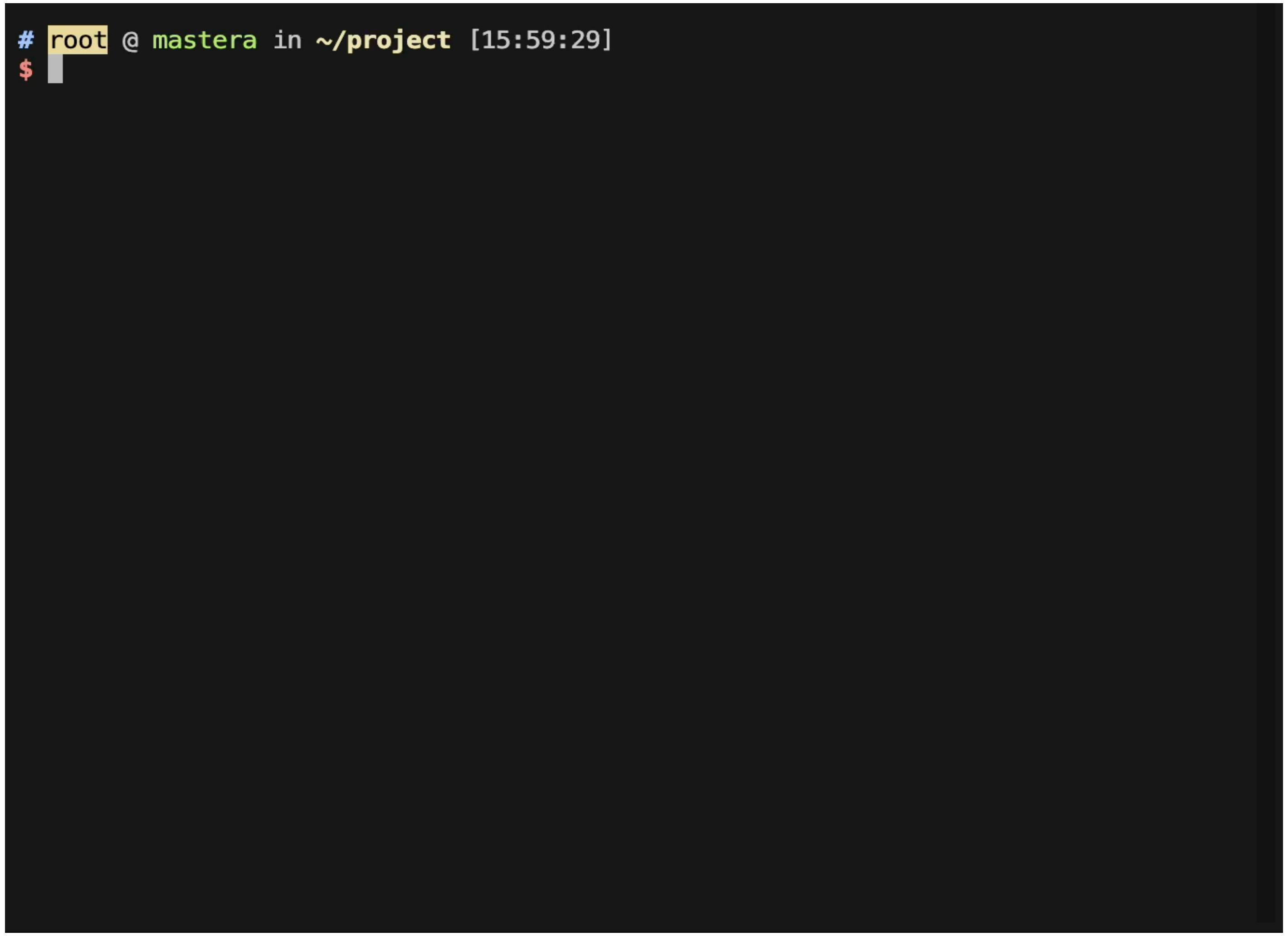
```
$ cwgo client --help
NAME:
   cwgo client - generate RPC or HTTP client

                 Examples:
                   # Generate RPC client code
                   cwgo client --type RPC --idl

                   # Generate HTTP client code
                   cwgo client --type HTTP --idl


USAGE:
   cwgo client [command options] [arguments...]
```

Showcase



```
# root @ mastera in ~/project [15:59:29]
$
```

RPC Server Layout

- MVC Layout
- DB Dal Init
- Unit Test
- Handler/IDL-Model

```
├── biz // business logic directory
|  ├── dal // data access layer
|  |  ├── init.go
|  |  ├── mysql
|  |  |  └── init.go
|  |  └── redis
|  |  └── init.go
|  └── service // service layer, where business logic is stored. When updating, the new
|  ├── HelloMethod.go
|  └── HelloMethod_test.go
├── build.sh
├── conf // Store configuration files in different environments
|  └── ...
├── docker-compose.yaml
├── go.mod // go.mod file, if not specified on the command line, the relative path relc
├── handler.go // Business logic entry, will be fully covered when updated
├── idl
|  └── hello. thrift
├──kitex.yaml
├── kitex_gen // Generate code related to IDL content, do not touch
|  └── ...
├── main.go // program entry
├── readme.md
└── script // startup script
     └── bootstrap.sh
```

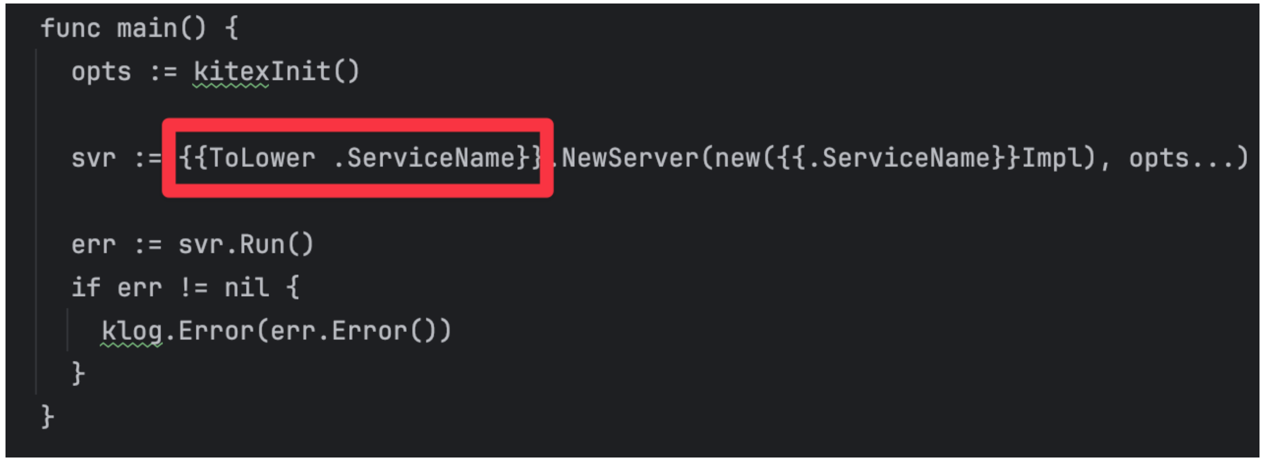
HTTP Server Layout

- MVC Layout
  - Shielding Framework Details
- DB Dal Init
- Unit Test
- Handler/Router/IDL-Model

```
├── biz // business logic directory
| ├── dal // data access layer
| | ├── init.go
| | ├── mysql
| | | └── init.go
| | └── redis
| | └── init.go
| ├── handler // view layer
| | └── hello
| | └── example
|
| | └── hello_service_test.go // single test file
| ├── router // generated code related to routes defined in idl
| | ├── hello
| | | └── example // hello/example corresponds to the namespace defined in thrift idl; for protobuf idl, it
| | | ├── hello.go // The route registration code generated by cwgo for the route defined in hello.thrift; e
| | | └── middleware.go // Default middleware function, hz adds a middleware to each generated routing group
| | └── register.go // call to register the routing definition in each idl file; when a new idl is added, it
| ├── service // service layer, where business logic is stored. When updating, the new method appends the fi
| | ├── hello_method.go // specific business logic
| | └── hello_method_test.go
| └── utils // tool directory
| └── resp.go
├── build.sh // compile script
├── conf // Store configuration files in different environments
| └── ...
├── docker-compose.yaml
├── go.mod // go.mod file, if not specified on the command line, the relative path relative to GOPATH will b
├── hertz_gen // Generate code related to IDL content
| └── ...
├── idl
| └── hello.thrift
├── main.go // program entry
├── readme.md
└── script // startup script
    └── bootstrap.sh
```

Customized Template

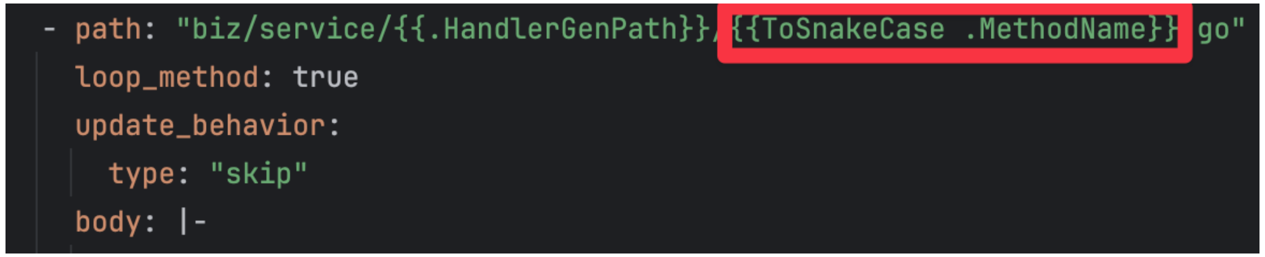
- All default generated code can be modified
- Can add your customized templates
- Using the "Go Template" writing template
- You can get all parsed IDL information
- Add useful template functions
- more: Template Extension



```
func main() {
  opts := kitexInit()

  svr := {{ToLower .ServiceName}}.NewServer(new({{.ServiceName}}Impl), opts...)

  err := svr.Run()
  if err != nil {
    klog.Error(err.Error())
  }
}
```



```
- path: "biz/service/{{.HandlerGenPath}}/{{ToSnakeCase .MethodName}}.go"
  loop_method: true
  update_behavior:
    type: "skip"
  body: |-
```

Template Usage

Local

```
cwgo server -type RPC -service {service name} -idl {idl path}  -template {local tpl path}
```

git https

```
-service {service name} -idl {idl path}  -template https://github.com/***/cwgo_template.git -branch {branch path}
```

git ssh

```
RPC -service {service name} -idl {idl path}  -template git@github.com:***/cwgo_template.git -branch {branch path}
```

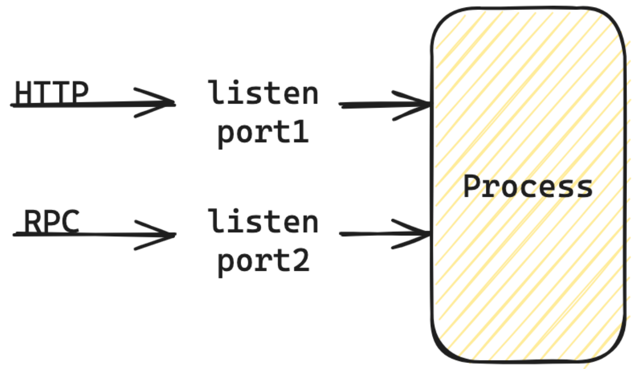
Cwgo-Hex (Hertz+Kitex)

Question: How to listen HTTP and RPC requests within a single process?

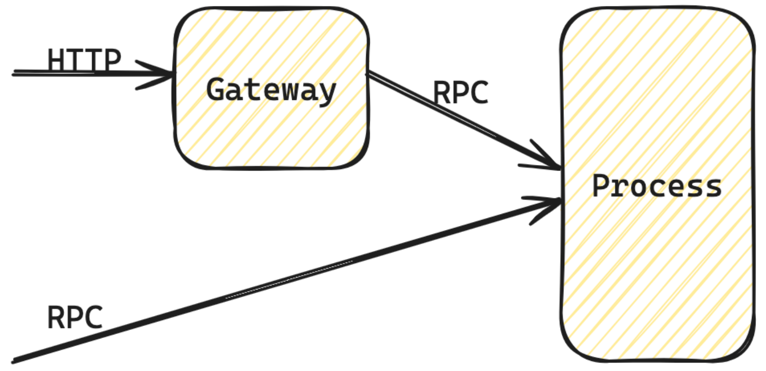
- Listen 2 port for HTTP and RPC

- Using gateway to convert protocol

1

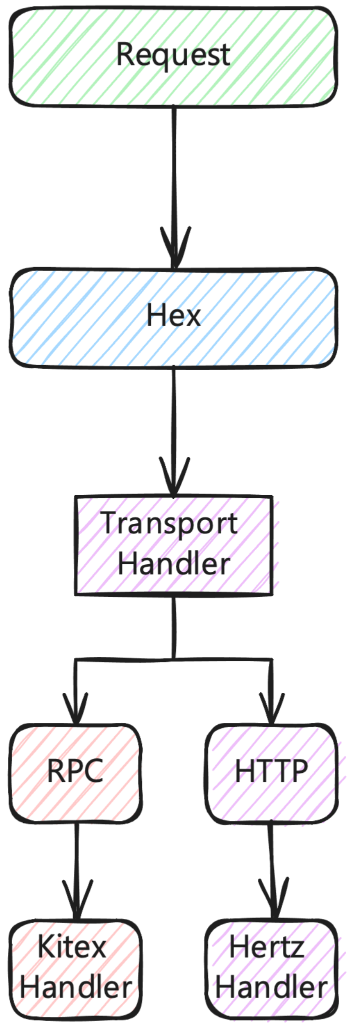


2



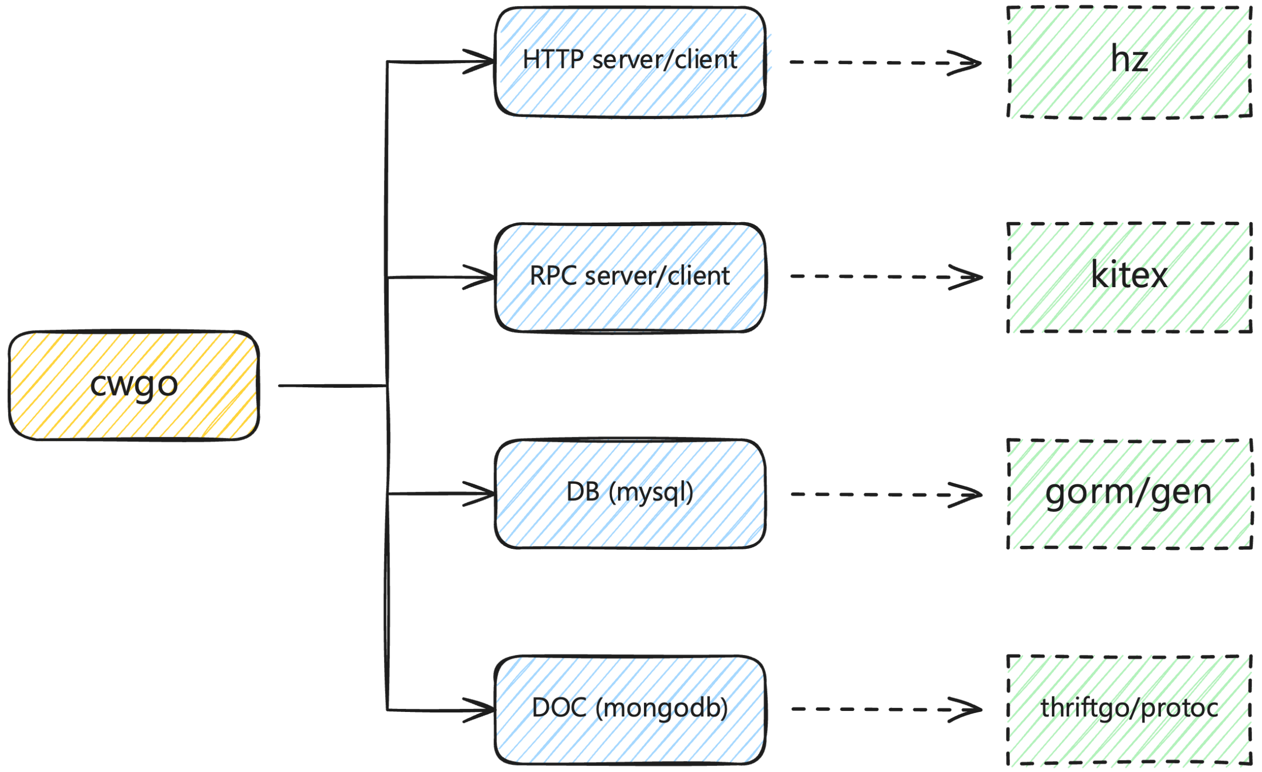
Cwgo-Hex (Hertz+Kitex)

- Listen HTTP and RPC on the same port
- Protocol Sniffing and Request Dispatching
- Reuse IDL and Improve Efficiency
- Avoid Integrating Other Components and Reduce Complexity



Cwgo-Mysql

- Based **gorm/gen** to generate orm
- Naturally Integrated into CloudWeGo
- Simple and Easy to Use



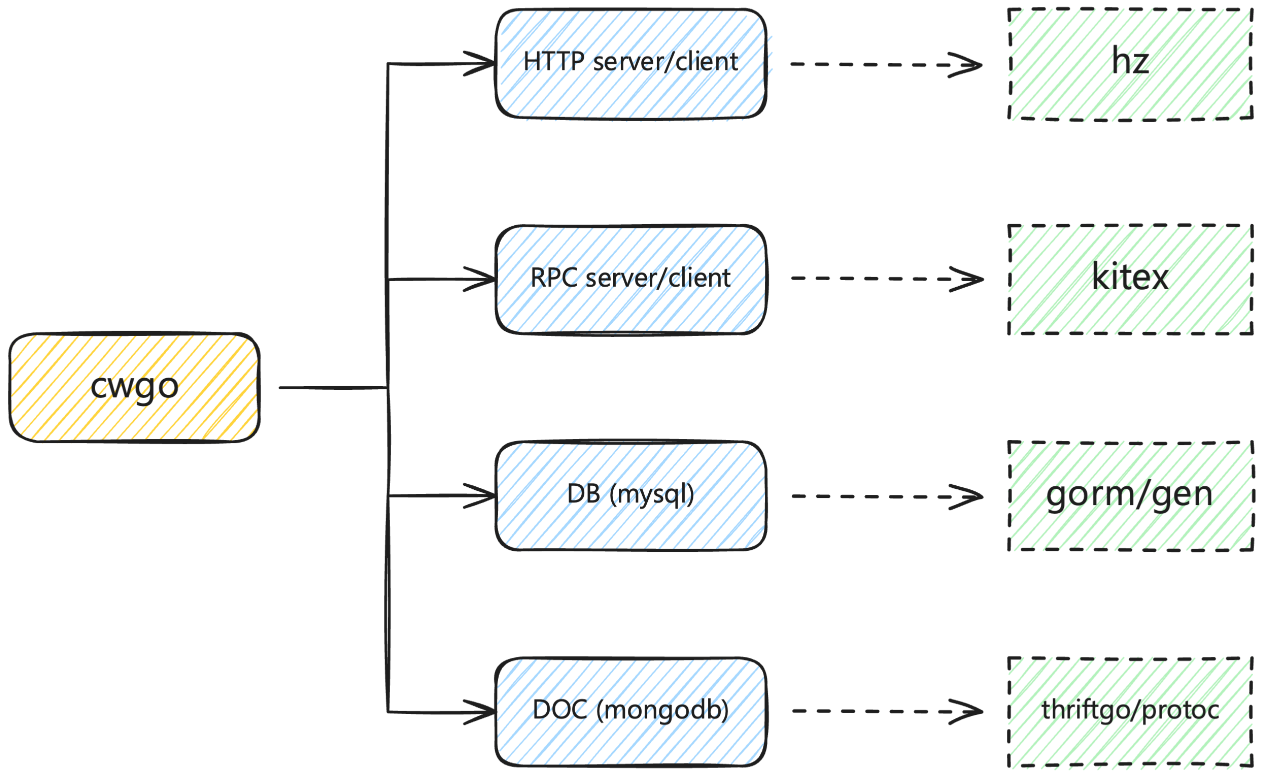
Usage

```
cwgo  model --db_type mysql --dsn "gorm:gorm@tcp(localhost:9910)/gorm?charset=utf8&parseTime=True&loc=Loca
```

Cwgo-Mongo

challenge

- Usage is Troublesome, Low Development Efficiency
- Do Not Support Transactions、Bulk Operations、 Aggregation Operations
- Incompatible MongoDB version, unable to use advanced features



Usage

Your IDL

```
struct User {
    1: string Id (go.tag="bson:\"id,omitempty\"")
    2: string Username (go.tag="bson:\"username\"")
    3: i32 Age (go.tag="bson:\"age\"")
    4: string City (go.tag="bson:\"city\"")
}
(
mongo.InsertUser = "InsertOne(ctx context.Context, user *user.User) (interface{}, error)"
mongo.FindUsernameOrderbyIdSkipLimitAll = "FindUsernames(ctx context.Context, skip, limit int64) ([]*user.User, error)"
mongo.UpdateUsernameByIdEqual = "UpdateContact(ctx context.Context, username string, id string) (bool, error)"
mongo.DeleteByUsernameEqual = "DeleteById(ctx context.Context, username string) (int, error)"
)
```

Command

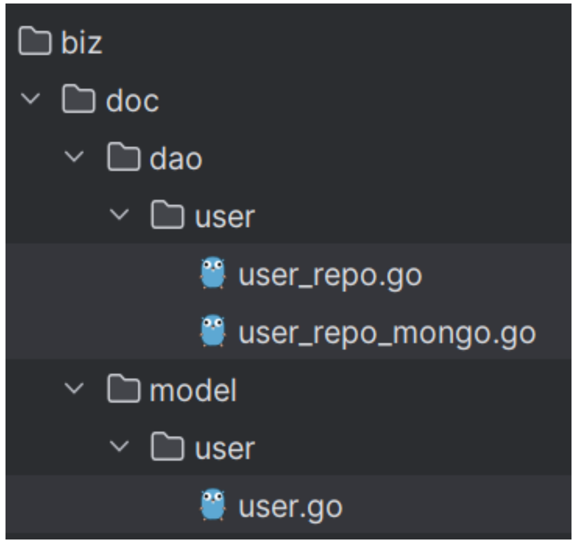
```
cwgo doc --idl user.thrift --module {your module name}
```

CloudWeGo

Usage

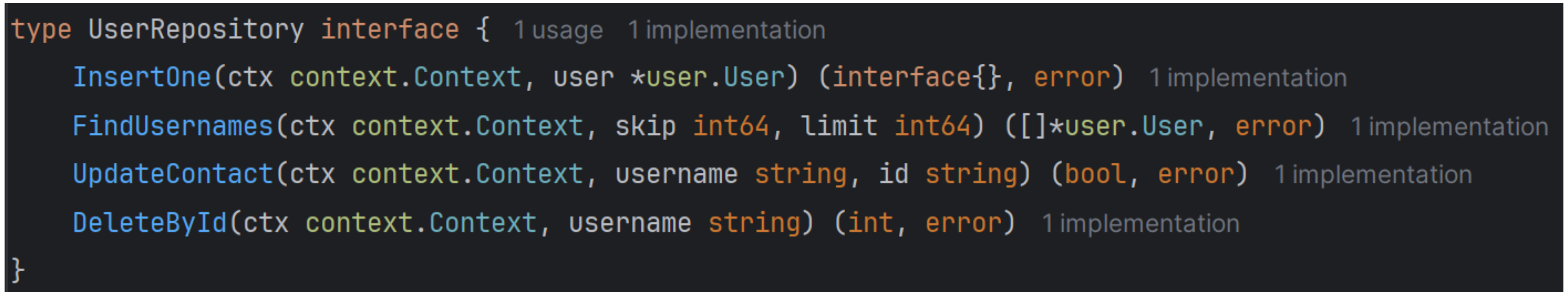
user_repo.go： interface definition
user_repo_mongo.go： interface implementation and CRUD code
user.go： generated code by thriftgo

Structure

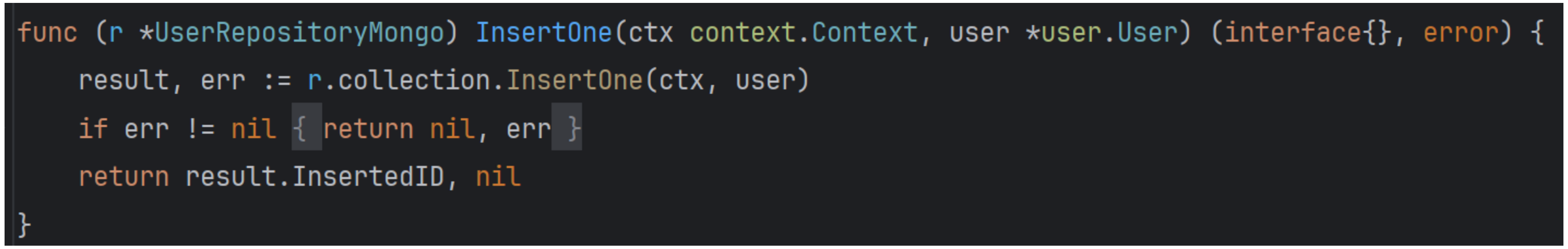


```
biz
  doc
    dao
      user
        user_repo.go
        user_repo_mongo.go
    model
      user
        user.go
```

Code



```go
type UserRepository interface {  1 usage  1 implementation
    InsertOne(ctx context.Context, user *user.User) (interface{}, error)  1 implementation
    FindUsernames(ctx context.Context, skip int64, limit int64) ([]*user.User, error)  1 implementation
    UpdateContact(ctx context.Context, username string, id string) (bool, error)  1 implementation
    DeleteById(ctx context.Context, username string) (int, error)  1 implementation
}
```



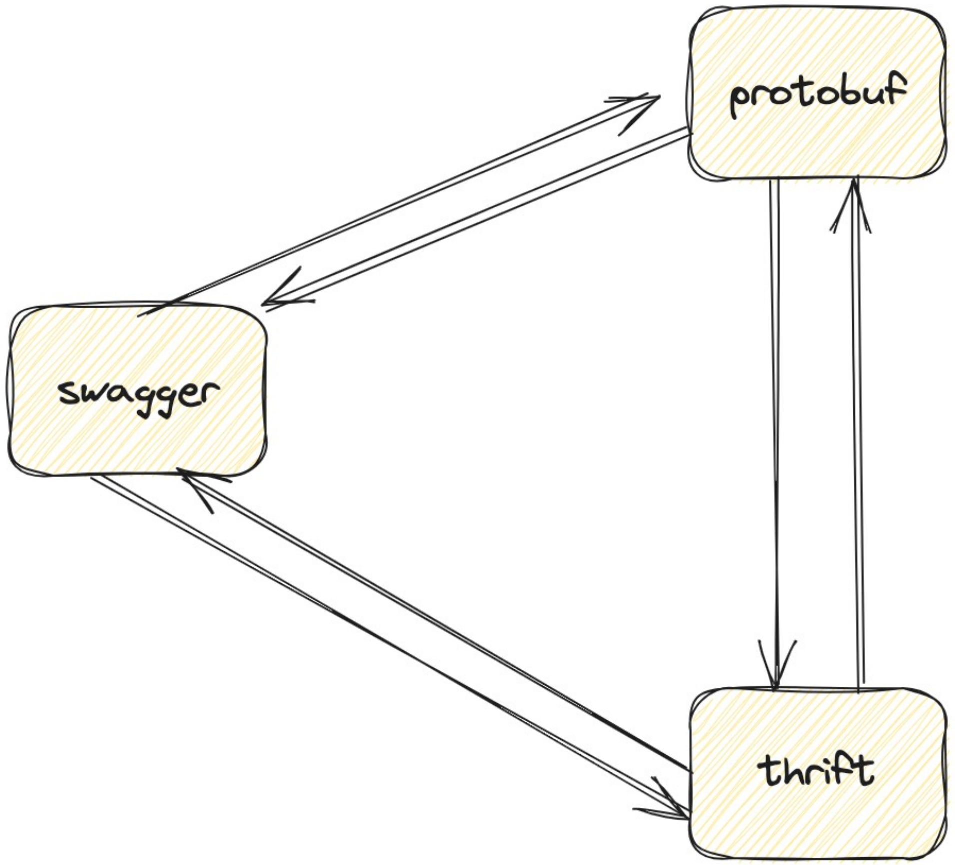
```go
func (r *UserRepositoryMongo) InsertOne(ctx context.Context, user *user.User) (interface{}, error) {
    result, err := r.collection.InsertOne(ctx, user)
    if err != nil { return nil, err }
    return result.InsertedID, nil
}
```

Part 03

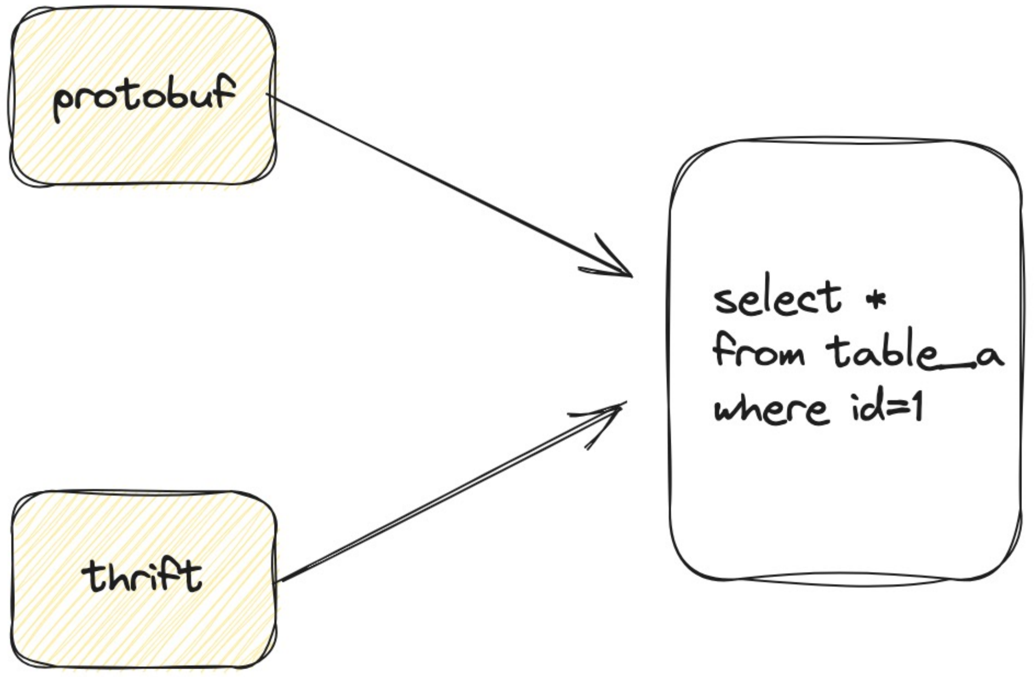
Future

1. IDL <-> Swagger

- Convert between IDL and Swagger
- Assisting services without IDL to access IDL



2. Generate raw SQL based on IDL



3. More Flexible IDE Plugin For CloudWeGo

- Developing VsCode/Goland plugins to generate CloudWeGo services with one-click
- Support generating cronjob、mq consumer code
- Support simpler RPC call solution

What’ s more ?

CloudWeGo Website: https://www.cloudwego.io/

Cwgo Document: https://www.cloudwego.io/docs/cwgo/

Thanks.