
GOPHER CHINA 2020
中国 上海 / 2020-11.21-22
Functional options and config for APIs
毛剑@bilibili

Redis client example

```
// DialTimeout acts like Dial for establishing the
// connection to the server, writing a command and reading a reply.
func Dial(network, address string) (Conn, error)
```

“我要自定义超时时间！”
“我要设定 Database！”
“我要控制连接池的策略！”
“我要安全使用Redis，让我填一下 Password！”
“可以提供一下慢查询请求记录，并且可以设置 slowlog 时间？”

Features

```
// DialTimeout acts like Dial for establishing the
// connection to the server, writing a command and reading a reply.
func Dial(network, address string) (Conn, error)


// DialTimeout acts like Dial but takes timeouts for establishing the
// connection to the server, writing a command and reading a reply.
func DialTimeout(network, address string, connectTimeout, readTimeout, writeTimeout
time.Duration) (Conn, error)


// DialDatabase acts like Dial but takes database for establishing the
// connection to the server, writing a command and reading a reply.
func DialDatabase(network, address string, database int) (Conn, error)

// DialPool
func DialPool...

// ...
```

net/http

```go
package main

import (
  "log"
  "net/http"
  "time"
)

func main() {
  s := &http.Server{
  Addr: ":8080",
  Handler: nil,
  ReadTimeout: 10 * time.Second,
  WriteTimeout: 10 * time.Second,
  MaxHeaderBytes: 1 << 20,
}
  log.Fatal(s.ListenAndServe())
}
```

type Server

```go
type Server struct {
    // Addr optionally specifies the TCP address for the server to listen on,
    // in the form "host:port". If empty, ":http" (port 80) is used.
    // The service names are defined in RFC 6335 and assigned by IANA.
    // See net.Dial for details of the address format.
    Addr string

    Handler Handler // handler to invoke, http.DefaultServeMux if nil

    // TLSConfig optionally provides a TLS configuration for use
    // by ServeTLS and ListenAndServeTLS. Note that this value is
    // cloned by ServeTLS and ListenAndServeTLS, so it's not
    // possible to modify the configuration with methods like
    // tls.Config.SetSessionTicketKeys. To use
    // SetSessionTicketKeys, use Server.Serve with a TLS Listener
    // instead.
    TLSConfig *tls.Config
```

Configuration struct API

```go
// Config redis settings.
type Config struct {
  *pool.Config
  Addr string
  Auth string
  DialTimeout time.Duration
  ReadTimeout time.Duration
  WriteTimeout time.Duration
}

// NewConn new a redis conn.
func NewConn(c *Config) (cn Conn, err error)

func main() {
  c := &redis.Config{
    Addr: "tcp://127.0.0.1:3389",
  }
  r, _ := redis.NewConn(c)
  c.Addr = "tcp://127.0.0.1:3390" // 副作用是什么？
}
```

Configuration struct API

```go
// NewConn new a redis conn.
func NewConn(c Config) (cn Conn, err error)

// NewConn new a redis conn.
func NewConn(c *Config) (cn Conn, err error)

// NewConn new a redis conn.
func NewConn(c ...*Config) (cn Conn, err error)

import (
  "github.com/go-kratos/kratos/pkg/log"
)

func main() {
  log.Init(nil) // 这样使用默认配置
  // config.fix() // 修正默认配置
}
```

“I believe that we, as Go programmers, should work hard to ensure that nil is never a parameter that needs to be passed to any public function.” — Dave Cheney

Functional options

Self-referential functions and the design of options -- Rob Pike
Functional options for friendly APIs -- Dave Cheney

```go
// DialOption specifies an option for dialing a Redis server.
type DialOption struct {
  f func(*dialOptions)
}
```

```go
// Dial connects to the Redis server at the given network and
// address using the specified options.
func Dial(network, address string, options ...DialOption) (Conn, error) {
  do := dialOptions{
    dial: net.Dial,
  }
  for _, option := range options {
    option.f(&do)
  }
  // ...
}
```

Functional options

```go
package main

import (
  "time"

  "github.com/go-kratos/kratos/pkg/cache/redis"
)

func main() {
  c, _ := redis.Dial("tcp", "127.0.0.1:3389",
  redis.DialDatabase(0),
  redis.DialPassword("hello"),
  redis.DialReadTimeout(10*time.Second))
}
```

Functional options

```
// DialOption specifies an option for dialing a Redis server.
type DialOption func(*dialOptions)


// Dial connects to the Redis server at the given network and
// address using the specified options.
func Dial(network, address string, options ...DialOption) (Conn, error) {
  do := dialOptions{
    dial: net.Dial,
  }
  for _, option := range options {
    option(&do)
  }
  // ...
}
```

Functional options

```
type option func(f *Foo) option

// Verbosity sets Foo's verbosity level to v.
func Verbosity(v int) option {
  return func(f *Foo) option {
    prev := f.verbosity
    f.verbosity = v
    return Verbosity(prev)
  }
}

func DoSomethingVerbosely(foo *Foo, verbosity int) {
  // Could combine the next two lines,
  // with some loss of readability.
  prev := foo.Option(pkg.Verbosity(verbosity))
  defer foo.Option(prev)
  // ... do some stuff with foo under high verbosity.
}
```

Functional options

```
type GreeterClient interface {
SayHello(ctx context.Context, in *HelloRequest, opts ...grpc.CallOption)
(*HelloReply, error)
}

type CallOption interface {
  before(*callInfo) error
  after(*callInfo)
}

// EmptyCallOption does not alter the Call configuration.
type EmptyCallOption struct{}

// TimeoutCallOption timeout option.
type TimeoutCallOption struct {
  grpc.EmptyCallOption
  Timeout time.Duration
}
```

Hybrid APIs

```go
// Dial connects to the Redis server at the given network and
// address using the specified options.
func Dial(network, address string, options ...DialOption) (Conn, error)

// NewConn new a redis conn.
func NewConn(c *Config) (cn Conn, err error)
```

“JSON/YAML 配置怎么加载，无法映射 DialOption 啊！”

“嗯，不依赖配置的走 options，配置加载走config”

Configuration & APIs

"For example, both your infrastructure and interface might use plain JSON. However, avoid tight coupling between the data format you use as the interface and the data format you use internally. For example, you may use a data structure internally that contains the data structure consumed from configuration. The internal data structure might also contain completely implementation-specific data that never needs to be surfaced outside of the system."

-- the-site-reliability-workbook 2

Configuration & APIs

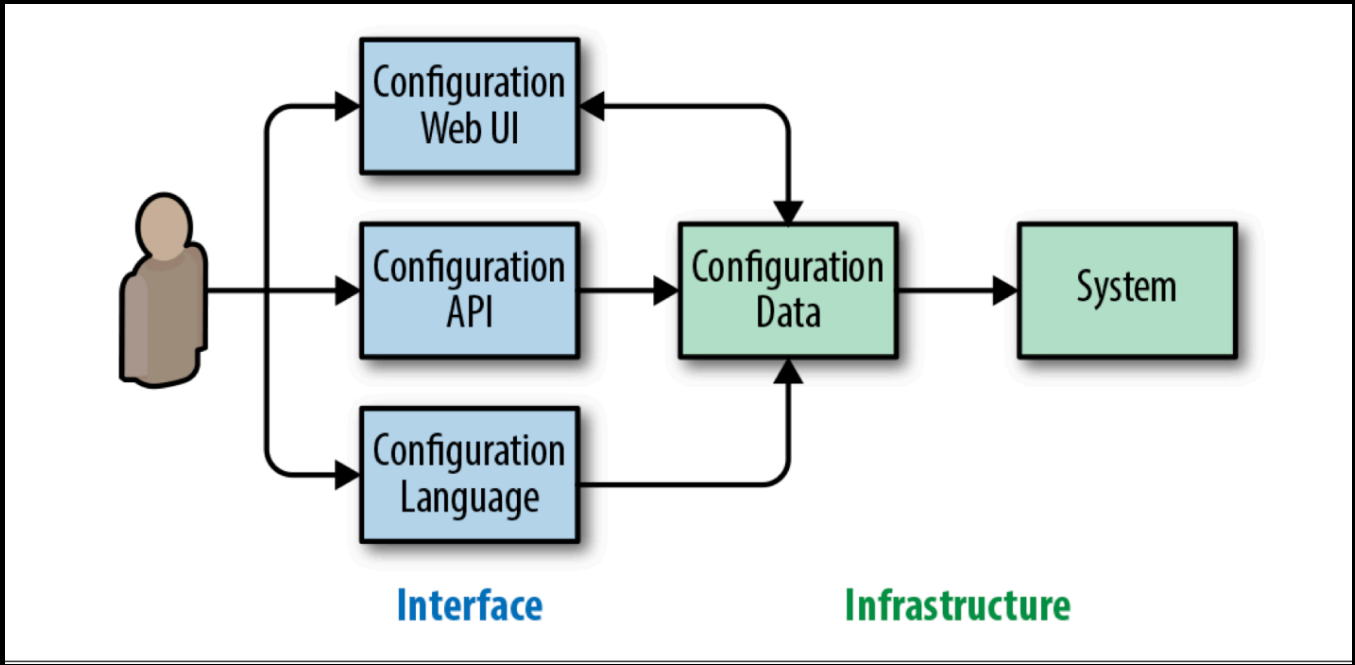
```
// Dial connects to the Redis server at the given network and
// address using the specified options.
func Dial(network, address string, options ...DialOption) (Conn, error)
```

- 仅保留 options API;
- config file 和 options struct 解耦;

配置工具的实践：

- 语义验证
- 高亮
- Lint
- 格式化

YAML + Protobuf



Configuration & APIs

```go
// Options apply config to options.
func (c *Config) Options() []redis.Options {
  return []redis.Options{
    redis.DialDatabase(c.Database),
    redis.DialPassword(c.Password),
    redis.DialReadTimeout(c.ReadTimeout),
  }
}

func main() {
  // instead use load yaml file.
  c := &Config{
    Network: "tcp",
    Addr: "127.0.0.1:3389",
    Database: 1,
    Password: "Hello",
    ReadTimeout: 1 * time.Second,
  }
  r, _ := redis.Dial(c.Network, c.Addr, c.Options()...)
}
```

```go
package redis

// Option configures how we set
up the connection.
type Option interface {
  apply(*options)
}
```

Configuration & APIs

```go
func ApplyYAML(s *redis.Config, yml string)
error {
  js, err := yaml.YAMLToJSON([]byte(yml))
  if err != nil {
    return err
  }
  return ApplyJSON(s, string(js))
}

// Options apply config to options.
func Options(c *redis.Config) []redis.Options {
  return []redis.Options{
    redis.DialDatabase(c.Database),
    redis.DialPassword(c.Password),
    redis.DialReadTimeout(c.ReadTimeout),
  }
}
```

```proto
syntax = "proto3";

import "google/protobuf/duration.proto";

package config.redis.v1;

// redis config.
message redis {
  string network = 1;
  string address = 2;
  int32 database = 3;
  string password = 4;
  google.protobuf.Duration read_timeout
= 5;
}
```

Configuration & APIs

```
func main() {
  // load config file from yaml.
  c := new(redis.Config)
  _ = ApplyYAML(c, loadConfig())
  r, _ := redis.Dial(c.Network, c.Address, Options(c)...)
}
```

Configuration Best Pratice

代码更改系统功能是一个冗长且复杂的过程，往往还涉及Review、测试等流程，但更改单个配置选项可能会对功能产生重大影响，通常配置还未经测试。

配置的目标：

- 避免复杂
- 多样的配置
- 简单化努力
- 以基础设施 -> 面向用户进行转变
- 配置的必选项和可选项
- 配置的防御编程
- 权限和变更跟踪
- 配置的版本和应用对齐
- 安全的配置变更：逐步部署、回滚更改、自动回滚

Figure 14-1. Control panel in the NASA spacecraft control center, illustrating possibly very complex configuration


GOPHER CHINA 2020
中国 上海 / 2020-11.21-22
Thanks