Package `context`. HTTP server and routers in Go.

Session 17

Golang course by Exadel

12 Dec 2022

Sergio Kovtunenko
Lead backend developer, Exadel

Agenda

- Package `context`
- HTTP server and routers
- JSON/XML (Un)marshaling
- HTTP server testing
- Various router libraries
- Next time...

Package `context`

Package `context`: general info

▶️ Package `context` (https://pkg.go.dev/context) defines the `Context` type, which carries:

- deadlines
- cancellation signals
- and other request-scoped values across API boundaries and between processes.

▶️ Cancellation aspect:

- Cancellation does not stop execution or trigger panics
- Cancellation informs code that its work is no longer needed
- Code checks for cancellation and decides what to do:
  - shut down? clean up? return errors?

Source: "Cancellation, Context, and Plumbing" by Sameer Ajmani (https://go.dev/talks/2014/gotham-context.slide)

Package `context`: but why do we need it?

▶️ In Go (HTTP) servers, each incoming request is handled in its own goroutine

▶️ HTTP Handler code needs access to request-specific values, like:

- security credentials
- request deadline

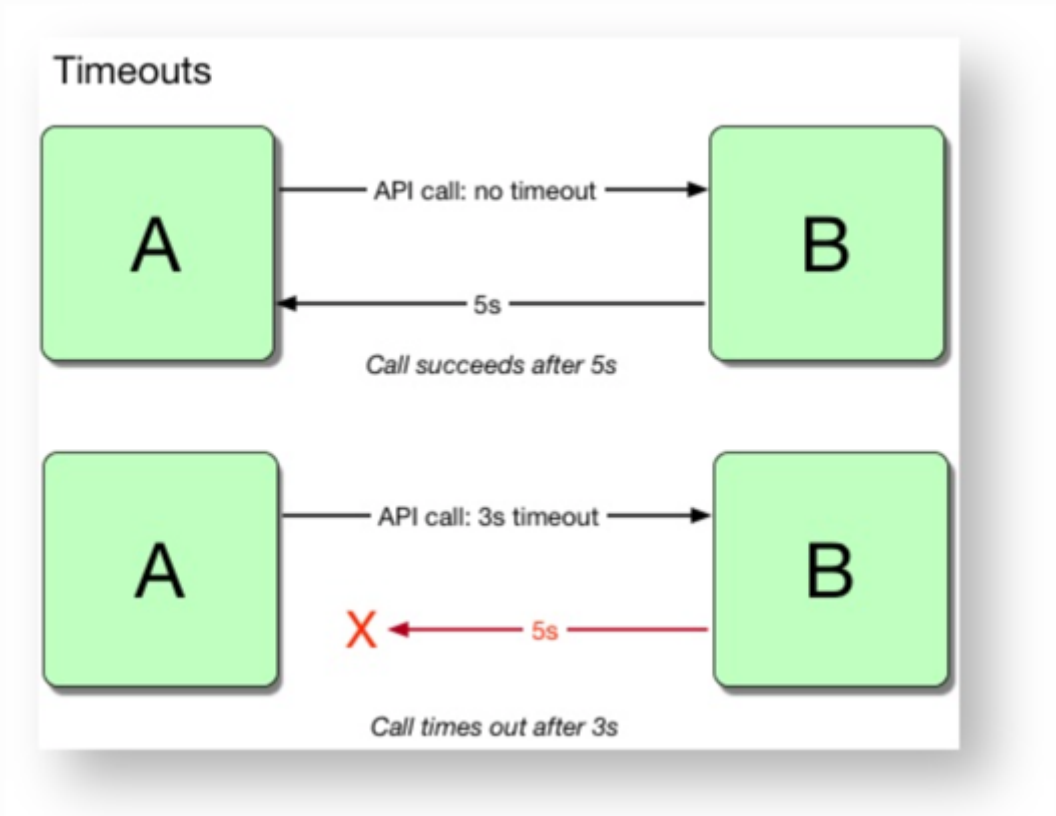
▶️ And when the request completes or times out, its work should be canceled

▶️ This is also important not only for the HTTP servers, but for the generic components as well!

Source: "Cancellation, Context, and Plumbing" by Sameer Ajmani (https://go.dev/talks/2014/gotham-context.slide)

Problem statement: timeouts visualization

▶️ Timeout visualization:



Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

Problem statement: timeouts handling

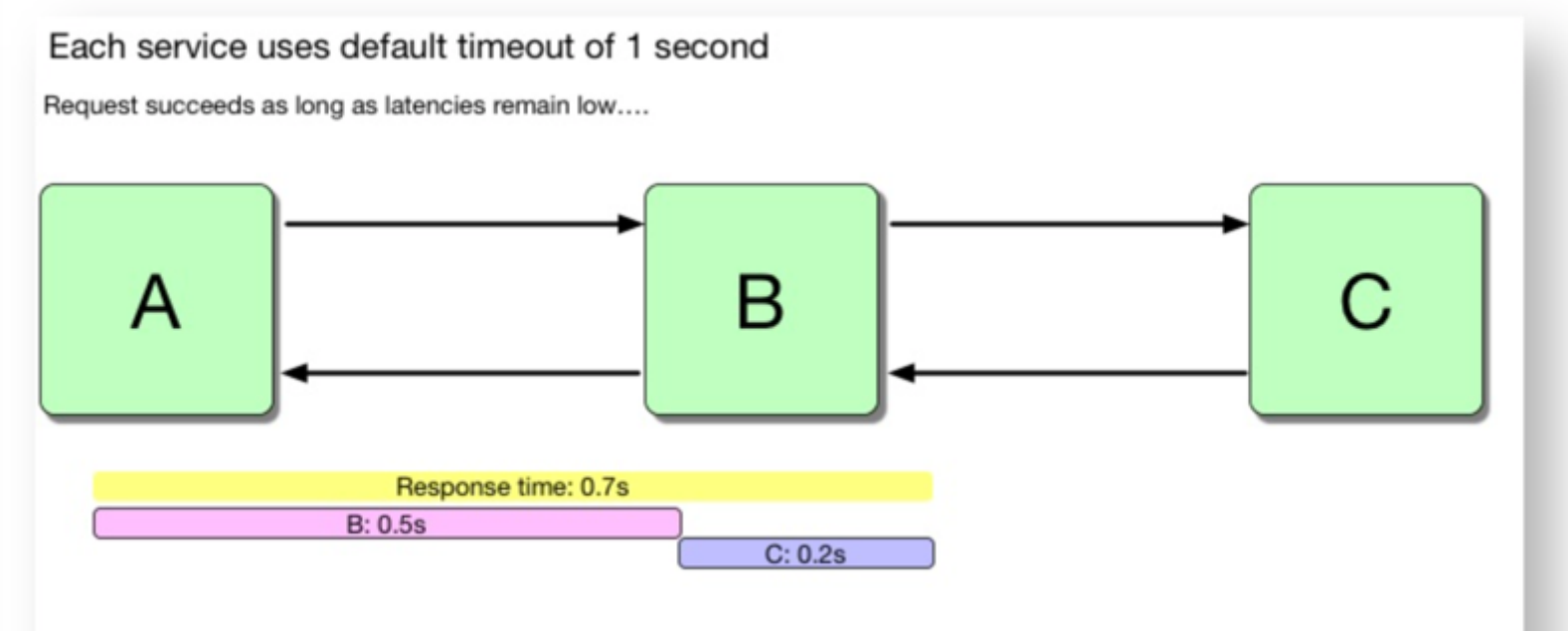
▶️ Why we shall care about timeouts? Graceful handling such situations, like:

- returning an error
- returning a fallback value
- retrying

Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

Timeout visualization between the services/components (1/3)

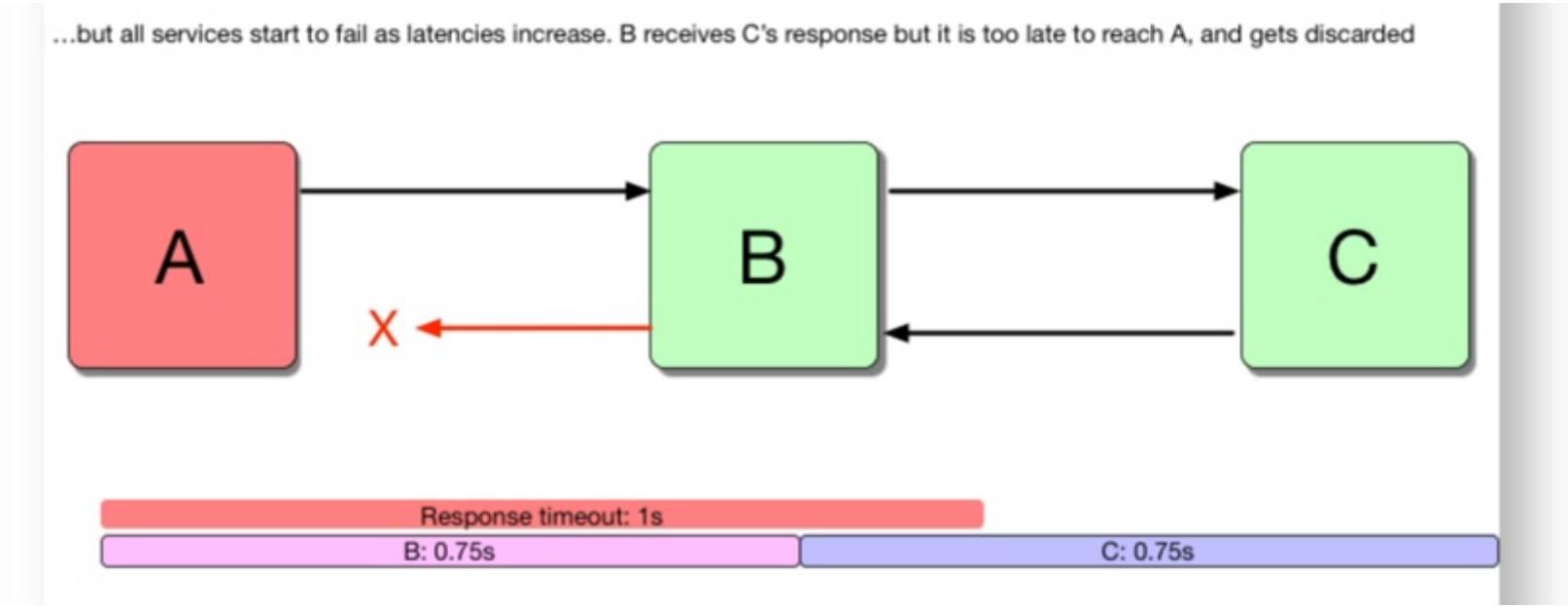
▶️ Sample case: 3-service/components architecture. Each service naively uses a default timeout of 1 second.



Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

Timeout visualization between the services/components (2/3)

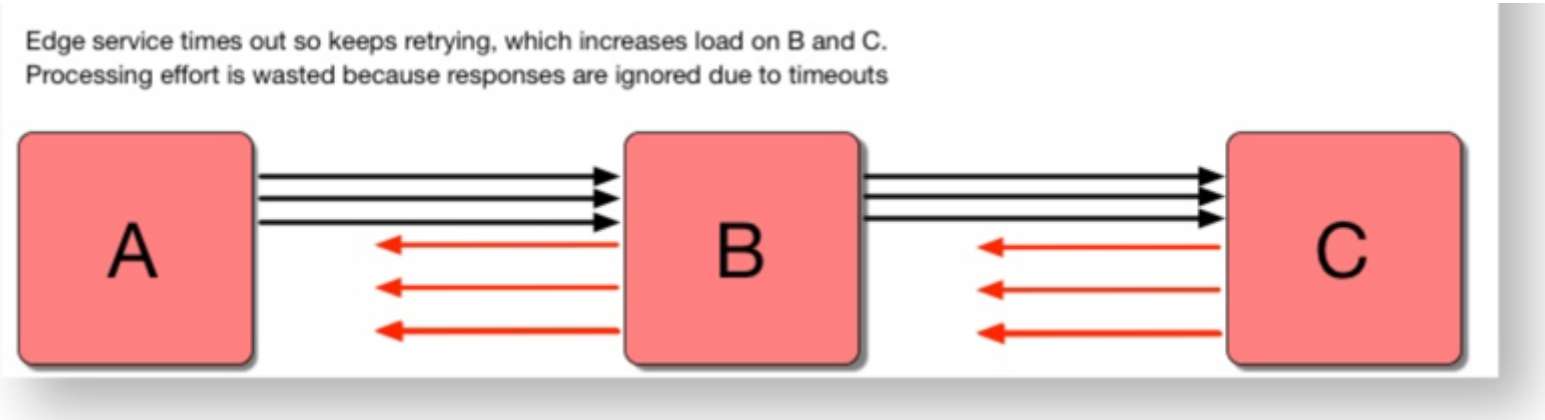
▶️ Sample case: 3-service/components architecture. Each service naively uses a default timeout of 1 second.



Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

Timeout visualization between the services/components (3/3)

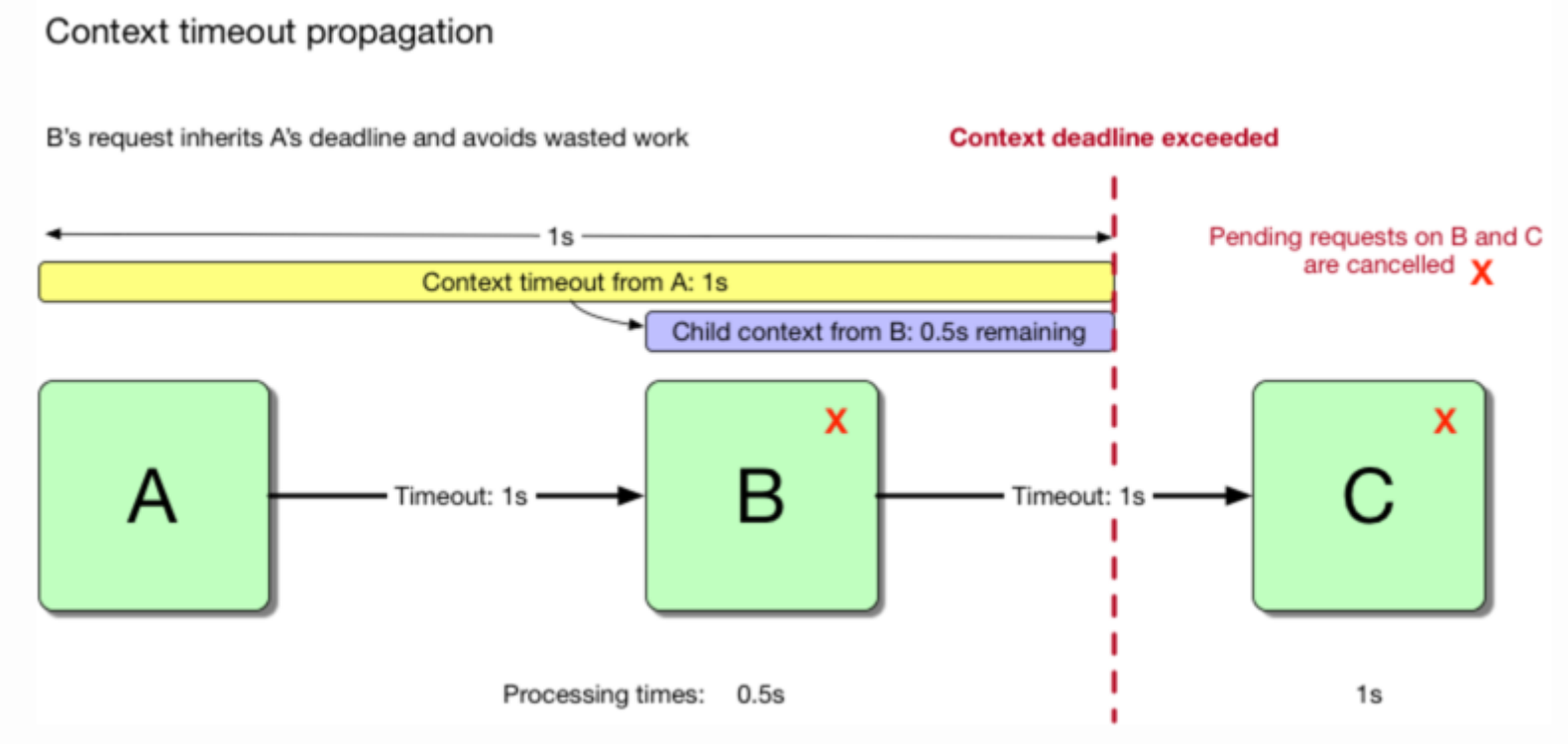
▶️ Sample case: 3-service/components architecture. Each service naively uses a default timeout of 1 second.



Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

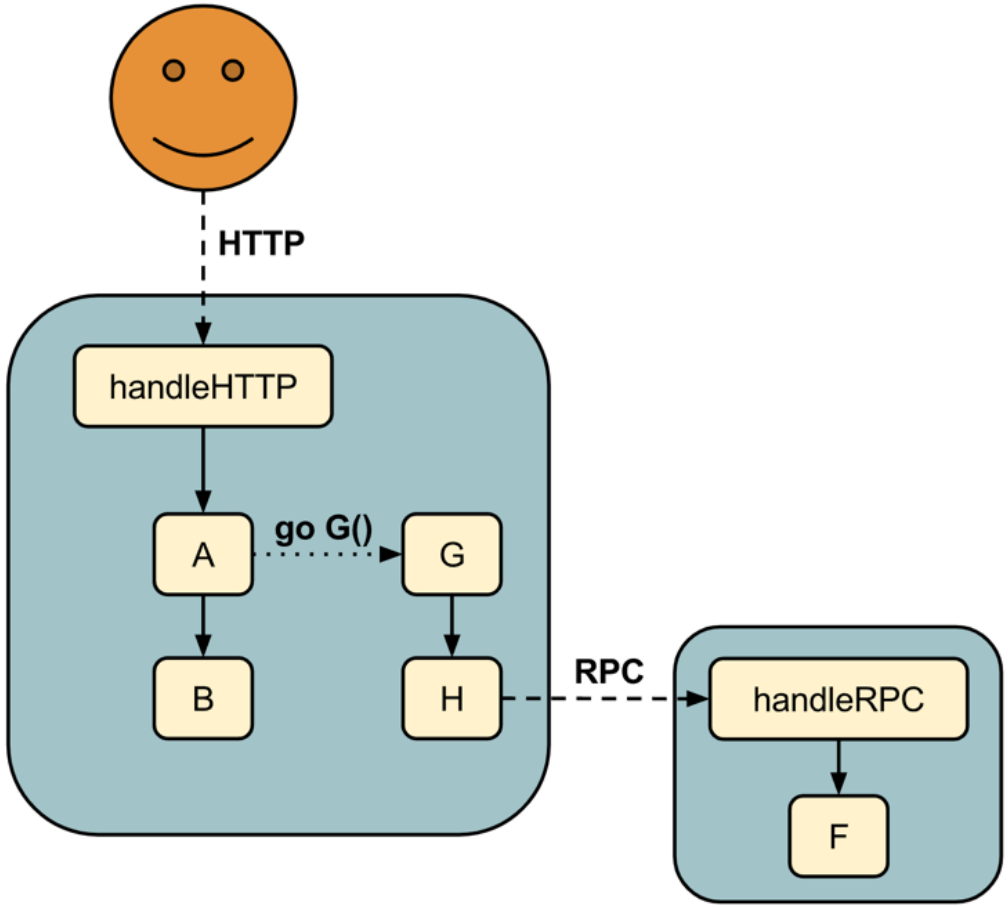
Context Propagation

▶️ Context aims to solve this problem by propagating the timeout and context information across API boundaries



Source: "Context Deadlines and How to Set Them" by Michael Cartmell (https://engineering.grab.com/context-deadlines-and-how-to-set-them)

In general, cancellation is transitive



Source: "Cancellation, Context, and Plumbing" by Sameer Ajmani (https://go.dev/talks/2014/gotham-context.slide)

Package `context`: historical overview about origins of the package (1/2)

▶️ Origins of the built-in package `context` was golang.org/x/net/context (https://pkg.go.dev/golang.org/x/net/context)

▶️ context package has been introduced in go1.7 (https://go.dev/doc/go1.7#context)

- proposal: context: new package for standard library #14660 (https://github.com/golang/go/issues/14660)

▶️ Before the `context` package, the typical done `channel` was used to signal work cancellation

▶️ 3rd-party drivers may respect the given `context` and cancel interaction inside driver

Package `context`: historical overview about origins of the package (2/2)

▶️ More context support:

- Since Go 1.7:
  - net
  - net/http
  - os/exec
- Since Go 1.8:
  - http.Server.Shutdown
  - database/sql
  - net.Resolver

Source: "The state of Go as of 2017" presentation (https://go.dev/talks/2017/state-of-go.slide)

Package `context`: implementation details

▶️ Public API:

```
type Context interface {
    Deadline() (dealine time.Time, ok bool)
    Done() <-chan struct{}
    Err() error
    Value(key interface{}) interface{}
}
```

▶️ Possible errors:

```
var Canceled = errors.New("context canceled")

type deadlineExceededError struct{}
func (deadlineExceededError) Error() string   { return "context deadline exceeded" }
func (deadlineExceededError) Timeout() bool   { return true }
func (deadlineExceededError) Temporary() bool { return true }
```

Package `context`: examples

- ▶️ Examples: `code/01_contextexample/01_context_test.go`
- ▶️ WARN: always plan to cancel context to avoid context leak

Package `context`: how to check if context is canceled ?

- using select-case:

```
select {
case <-ctx.Done():
    return ctx.Err()
default:
}
```

- using `ctx.Err()` method:

```
if err := ctx.Err(); err != nil {
    return err
}
```

Package `context`: best practices

- Do not store variable of `context.Context` type inside the structs, pass them explicitly in functions, preferably, as the first argument.
- Don't pass `nil` Context to functions that take a `context.Context`
- Add deadlines to your Contexts
- When we need to attach a value to context, use custom private types for keys in `context.Context` variables
- Pass **only** request scoped values via `context.Context` (also, try to minimize the usage of this technique)
- `context.TODO` should be used where not sure what to use or if the current function will be updated to use context in future
- Warning: starting long-running task from the HTTP handler and propagate same HTTP request context to the task!

HTTP server and routers

HTTP server and routers: to serve static files

▶️ Basic static server:

```
package main

import "net/http"

func main() {
  port := ":9999"
  handler := http.FileServer(http.Dir("files"))
  http.ListenAndServe(port, handler)
}
```

HTTP server and routers: to serve simple requests

▶️ Basic example:

```
package main
import (
  "fmt"
  "net/http"
)

func hello(w http.ResponseWriter, req *http.Request) {
  fmt.Fprintf(w, "hello\n")
}

func main() {
  http.HandleFunc("/hello", hello)

  http.ListenAndServe(":8090", nil)
}
```

▶️ Example: code/02_httpserverexample/01_simple/main.go

HTTP server and routers: core types

- http.HandlerFunc (https://pkg.go.dev/net/http#HandlerFunc): The HandlerFunc type is an adapter to allow the use of ordinary functions as HTTP handlers.

```
type HandlerFunc func(ResponseWriter, *Request)
```

- http.Handler (https://pkg.go.dev/net/http#Handler): A Handler responds to an HTTP request.

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

- Do not forget to set timeouts: explanation (https://blog.cloudflare.com/the-complete-guide-to-golang-net-http-timeouts/)

Middleware pattern

▶️ Logging middleware example:

```
func Logging(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, req *http.Request) {
    start := time.Now()
    next.ServeHTTP(w, req)
    log.Printf("%s %s %s", req.Method, req.RequestURI, time.Since(start))
  })
}
```

▶️ How to use logging middleware:

```
func main() {
  mux := http.NewServeMux()
  server := NewTaskServer()
  mux.HandleFunc("/task/", server.taskHandler)

  handler := middleware.Logging(mux)

  log.Fatal(http.ListenAndServe("localhost:"+os.Getenv("SERVERPORT"), handler))
}
```

HTTP clients

- Using standard library we can use HTTP clients (https://pkg.go.dev/net/http#example-Get)
- More advanced libraries are available, for example: go-resty/resty (https://github.com/go-resty/resty)
- Example: `code/02_httpserverexample/02_httpclient/01_httpclient_test.go`

Typical JSON/XML request/response marshaling

▶️ If the HTTP request/response body is of type `JSON/XML`, then we need a library to marshal/unmarshal them

▶️ JSON standard library: "encoding/json" (https://pkg.go.dev/encoding/json)

▶️ XML standard library: "encoding/xml" (https://pkg.go.dev/encoding/xml)

▶️ `JSON/XML` marshalling: do you know the format beforehand?

- strict format, known beforehand
  - using 3rd-party libraries we can get performance boost
- unknown or "dynamic" JSON payload format

▶️ Examples: `code/03_jsonexample/01_jsonexample_test.go`

HTTP server testing

HTTP server testing

▶️ Special standard package with utilities for HTTP testing: httptest (https://pkg.go.dev/net/http/httptest)

▶️ Provides ability to:

- test HTTP interactions: perform HTTP request and investigate response details,
- start HTTP server for testing

▶️ Examples: `code/04_httptestexample/01_httptesting_test.go`

Various router libraries

Various router libraries

- HTTP router benchmarking (https://github.com/julienschmidt/go-http-routing-benchmark)
- Gorilla Mux (https://github.com/gorilla#gorilla-toolkit)
- go-chi/chi (https://github.com/go-chi/chi)
- valyala/fasthttp (https://github.com/valyala/fasthttp)
- There are different approaches to HTTP routing in Go (https://benhoyt.com/writings/go-routing/)
- Routers might be **compatible or not** with the standard go HTTP server API.

Which one to use?

- ▶️ Which Go router should I use? (with flowchart) (https://www.alexedwards.net/blog/which-go-router-should-i-use)
- ▶️ Decide to go with a small library or a framework?
- ▶️ Frameworks are:
  - Echo (https://echo.labstack.com/)
  - aah (https://aahframework.org/)
  - beego (https://github.com/beego/beego)
  - gin (https://github.com/gin-gonic/gin)
  - go-kit (https://github.com/go-kit/kit)
  - fiber (https://github.com/gofiber/fiber)
  - revel (https://github.com/revel/revel)

Homework:

- ▶️ Package `context` guides:
  - How to correctly use context.Context in Go 1.7 (https://medium.com/@cep21/how-to-correctly-use-context-context-in-go-1-7-8f2c0fafdf39)
  - Under the hood of Go's context (https://vishnubharathi.codes/blog/go-contexts/)
  - Context Deadlines and How to Set Them (https://engineering.grab.com/context-deadlines-and-how-to-set-them)
- ▶️ The complete guide to Go net/http timeouts (https://blog.cloudflare.com/the-complete-guide-to-golang-net-http-timeouts/)
- ▶️ Go JSON Cookbook (https://eli.thegreenplace.net/2019/go-json-cookbook/)
- ▶️ Which Go router should I use? (with flowchart) (https://www.alexedwards.net/blog/which-go-router-should-i-use)
- ▶️ Investigate http client library: go-resty/resty (https://github.com/go-resty/resty)

Next session...

▶️ Session18:

Generics in Go

- General vs. Basic interfaces
- Type constraints
- Type parameter and instantiation
- Generic interfaces

Thank you

Golang course by Exadel

12 Dec 2022

Sergio Kovtunenko
Lead backend developer, Exadel
skovtunenko@exadel.com (mailto:skovtunenko@exadel.com)
https://github.com/skovtunenko (https://github.com/skovtunenko)
@realSKovtunenko (http://twitter.com/realSKovtunenko)