FABIO FALZOI
fabio@develer.com

Design Patterns for production grade Go services

develer

24 hours ago...

Gaetano to the rescue!

What will we do?

- A little bit of talk
- Review of my code
- Try the exercises
- Review of the solution

Hey, a nice slidedeck should have an image here!

A simple HTTP server

```go
package main

import (
    ...
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Production environment

Expect the unexpected

Failures are inevitable

Production environment

Do not leak your resources!

Timeouts & Graceful Shutdown

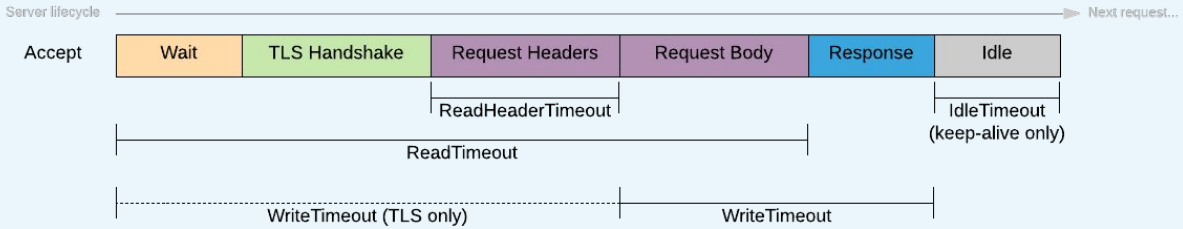
Client Timeouts

```
client := &http.Client{
    // Timeout specifies a time limit for requests made by this
    // Client. The timeout includes connection time, any
    // redirects, and reading the response body. The timer remains
    // running after Get, Head, Post, or Do return and will
    // interrupt reading of the Response.Body.
    Timeout: time.Second,
}

resp, err := client.Get(ts.URL)
if err != nil {
    ...
}
defer resp.Body.Close()
```

Client Timeouts - context.Context

```
ctx, cancel := context.WithTimeout(context.Background(), time.Second)
defer cancel()

req, err := http.NewRequestWithContext(ctx, http.MethodGet, ts.URL, nil)
if err != nil {
    panic(err)
}
resp, err := http.DefaultClient.Do(req)
if err != nil {
    panic(err)
}
defer resp.Body.Close()
```

Server Timeouts




```
server := &http.Server{
    ReadTimeout:       1 * time.Second,
    WriteTimeout:      1 * time.Second,
    IdleTimeout:       30 * time.Second,
    ReadHeaderTimeout: 2 * time.Second,
}
```

*Image taken from here

Server Timeouts - per handler read timeout

```go
type timeoutHandler struct{}

func (h timeoutHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
   defer r.Body.Close()

   timer := time.AfterFunc(5*time.Second, func() {
       r.Body.Close()
   })

   var buf bytes.Buffer
   for {
       timer.Reset(1 * time.Second)

       _, err := io.CopyN(&buf, r.Body, 512)
       if err == io.EOF {
           break
       }
       if err != nil {
           // error handling
       }
   }
}

func main() {
   s := &http.Server{
       ReadHeaderTimeout:20 * time.Second,
       Handler:           timeoutHandler{},
   }
}
```

Graceful shutdown

```
server := &http.Server{
    Addr: ":8080",
}

go func() {
    if err := server.ListenAndServe(); err != nil && !errors.Is(err, http.ErrServerClosed) {
        panic(err)
    }
}()

signalChan := make(chan os.Signal, 1)
signal.Notify(signalChan, syscall.SIGINT, syscall.SIGTERM)

<-signalChan

ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
defer cancel()

if err := server.Shutdown(ctx); err != nil {
    panic(err)
}
```

Graceful shutdown and worker goroutines

```go
func (s *Service) Run(
    ctx context.Context, wg *sync.WaitGroup,
) (<-chan Message, <-chan error) {
    wg.Add(1)
    go func() {
        defer func() {
            // clean up
        }()
        for {
            var (
                msg Message
                err error
            )
            // ...
            select {
            case <-ctx.Done():
                return
            case s.events <- msg:
            case s.errors <- err:
            }
        }
    }()
    return s.events, s.errors
}
```

```go
func main() {
    ctx, cancel := context.WithCancel(context.Background())
    var wg sync.WaitGroup

    service, err := NewService()
    if err != nil {
        panic(err)
    }
    events, errors := service.Run(ctx, &wg)
    for {
        select {
        case <-stop:
            // signal cancellation
            cancel()
            // wait for all goroutines to complete shutdown
            wg.Wait()
            return
        case ev := <-events:
            fmt.Printf("received event: %v\n", ev)
        case err := <-errors:
            fmt.Printf("received error: %v\n", err)
        }
    }
}
```

“Talk is cheap
Show me the code”

Retry &
Circuit Breaker

Thundering Herd problem

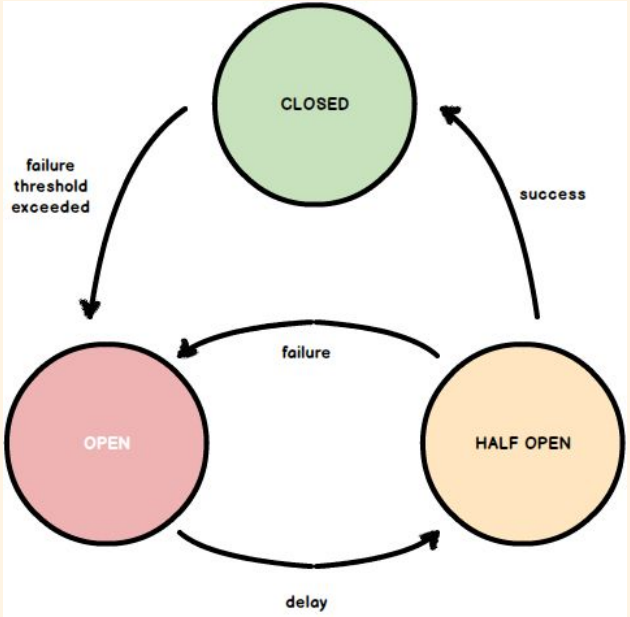
“In computer science, the thundering herd problem occurs when a large number of processes or threads waiting for an event are awoken when that event occurs, but only one process is able to handle the event. When the processes wake up, they will each try to handle the event, but only one will win. All processes will compete for resources, possibly freezing the computer, until the herd is calmed down again.”

Retry

- Useful for *transient failures*
- Backoff strategy
- Randomization of backoff time

Circuit breaker

- Useful for *long lasting faults*
- Fail fast and stay responsive
- Backoff strategy
- Half open strategy



Retry vs Circuit Breaker

Transient failure vs *long lasting faults*

“Talk is cheap
Show me the code”

“Again? Really?”

Scatter Gather & Hedged Requests

Latency of a request - percentile

"A percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations falls."

Latency of a request - percentile

- Consider a list of samples (latencies)
- Sort them in increasing order
- p50 ⇒ 50% of the samples are below that value and 50% of the are above
- p95 ⇒ 95% of the samples are below that value and 5% of the are above
- p99 ⇒ 99% of the samples are below that value and 1% of the are above
- ...

Long Tail Latency - What and Why

- **What** ⇒ higher percentiles (such as 98th, 99th) of latency in comparison to the average latency time.
- **Why** ⇒ shared resources (memory and network bandwidth, CPU cores) queuing, Stop the world garbage collection, etc.

Scatter Gather

- also known as *fan-out*
- Use multiple goroutines to make the same request concurrently

Hedged Requests

- Same principle as fan-out
- Optimize resource usage

“Talk is cheap …”

“Shut up, I’m opening my editor!”

References & further readings

- *Complete guide to timeouts*
- *The tail at scale*

Thank you!

CONTACTS

Design Patterns for production grade Go services

fabio@develer.com
github.com/pippolo84/
@Pippolo84

www.develer.com