Implementing a cache

Benoît Masson
benoit@open-agora.com

Golang Rennes,
Wednesday, March 14th, 2018

Contract

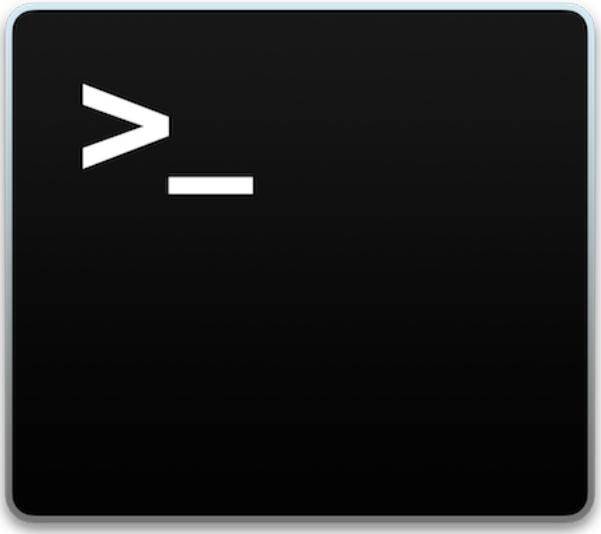
```
package cache

type T interface {
    Get(key string) ([]byte, bool)
    Add(key string, content []byte)
    Invalidate(key string)
}
```

Let's start !

- Implementation `cache.None`
  - dummy cache, does nothing

- Basic tests



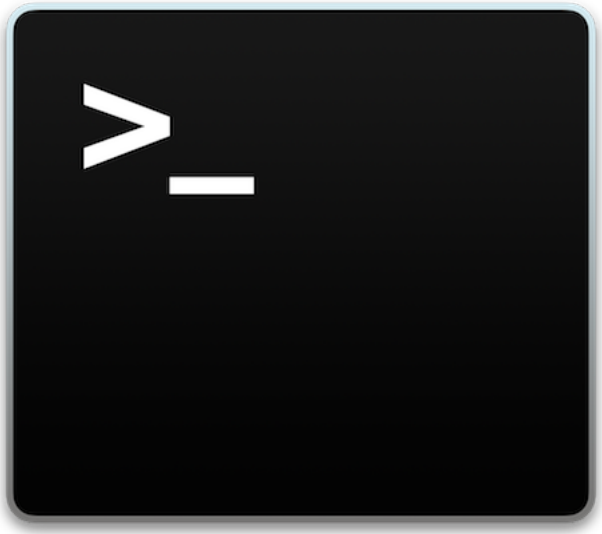
open agora

1.

cache.Memory

cache.Memory

- Stores content in RAM, in a `map`



open agora

2.

cache.SyncMemory

cache.SyncMemory

- 2 options
  - add a sync.RWMutex
  - use a sync.Map

> The Map type is optimized for two common use cases:
> (1) when the entry for a given key is only ever written once but read many times, as in **caches that only grow**, or
> (2) when multiple goroutines read, write, and overwrite entries for **disjoint sets of keys**.
> In these two cases, use of a Map may significantly reduce lock contention compared to a Go map paired with a separate Mutex or RWMutex.

[https://golang.org/pkg/sync/#Map]

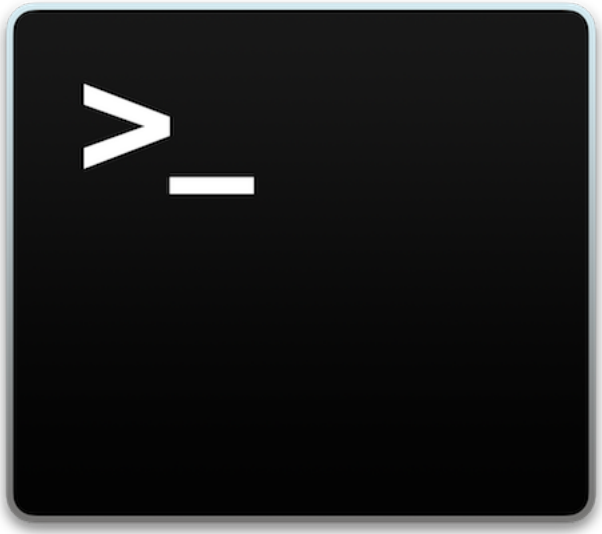
open agora

3.

cache.File

cache.File

- Persistent storage
- Optimize HTTP handler



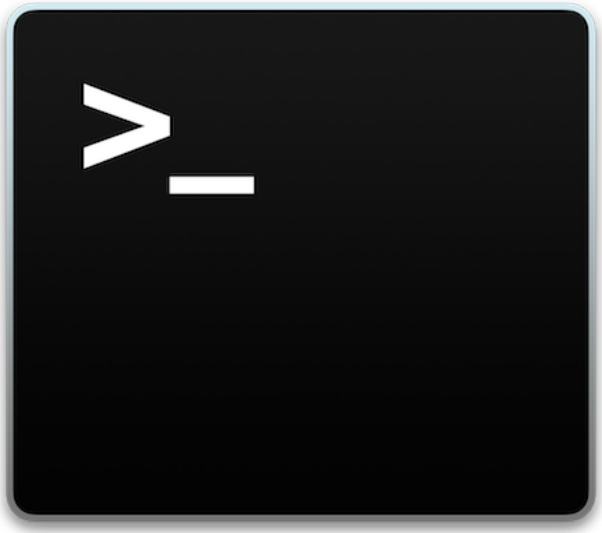
open agora

4.

cache.Expirable

cache.Expirable

- Limit cache size by self-destructing elements after a while
  - timer may be reset after each use



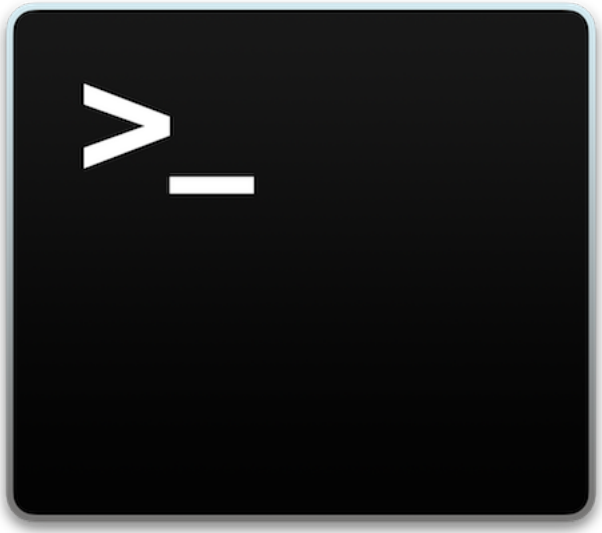
open agora

5.

cache.Bounded

cache.Bounded

- Limit cache size by allowing a maximum number of elements
  - removal policy?



Thanks!

Benoît Masson
benoit@open-agora.com