Урок Спринт 2.11

Старый знакомый

В уроке "5. Мьютексы" и уроке "6. Базовые паттерны concurrency пакета sync" мы уже реализовывали структуры вида

```
type DataSet struct {
   data  map[string]string
   mutex sync.RWMutex
}
```

С Методами

```
func (d *DataSet) Get(key string) string {

}

func (d *DataSet) Set(key string, value string) {

}
```

Это и есть "наивная" реализация кеша в go.

Давайте для закрепления пройденного материала еще раз приведём такую наивную реализацию кеша.

```
package main

import (
    "fmt"
    "sync"
    "time"
)

// создание простого кеша ключ-значение с потокобезопасными операциями чтения и записи
type Cache struct {
    data  map[string]interface{} // мапа для хранения пар ключ-значение
    mutex sync.RWMutex           // мьютекс для синхронизации конкурентного доступа к кешу
}

// создание нового экземпляра кеша с инициализированной мапой данных
func NewCache() *Cache {
```

```
    return &Cache{
        data: make(map[string]interface{}),
    }
}

// извлекает значение, связанное с данным ключом, из кеша
// Get() возвращает значение и признак, указывающий, был ли найден ключ
func (c *Cache) Get(key string) (interface{}, bool) {
    c.mutex.RLock()         // acquire a read lock to allow multiple readers simultaneously
    defer c.mutex.RUnlock() // release the read lock when the function exits
    value, ok := c.data[key]
    return value, ok
}

// установка значения, связанного с данным ключом в кеше
// Set() получает блокировку на запись для обеспечения эксклюзивного доступа во время обновления
func (c *Cache) Set(key string, value interface{}) {
    c.mutex.Lock()         // получение блокировки на запись для эксклюзивного доступа
    defer c.mutex.Unlock() // снятие блокировки записи при завершении работы функции
    c.data[key] = value    // установка значения в кеше по ключу
}

func main() {
    cache := NewCache()

    cache.Set("username", "yandexlyceum")
    cache.Set("year", 2024)

    if value, ok := cache.Get("username"); ok {
        fmt.Println("Value for username:", value)
    } else {
        fmt.Println("username not found in the cache.")
    }

    if value, ok := cache.Get("year"); ok {
        fmt.Println("Value for year:", value)
    } else {
        fmt.Println("year not found in the cache.")
    }

    time.Sleep(2 * time.Second)

    if value, ok := cache.Get("username"); ok {
        fmt.Println("Value for username (after some time):", value)
    } else {
        fmt.Println("username not found in the cache after some time.")
    }

    if value, ok := cache.Get("year"); ok {
        fmt.Println("Value for year (after some time):", value)
    } else {
        fmt.Println("year not found in the cache after some time.")
    }
```

```
}
```

Это прекрасный подход, и часто его может быть достаточно. Но что, если мы будем неограничено добавлять в такой кеш данные? У приложения просто кончится память, потому что размер нашего кеша никак не ограничен. Давайте ответим на вопрос, что можно предпринять, когда место в памяти кончилось, а кешировать продолжать хочется?