go 语言实现简易内存缓存

作者：zhengliwei
原文链接：https://ld246.com/article/1605428342231
来源网站：链滴

hello，大家好，欢迎来到银之庭。我是Z，一个普通的程序员。最近工作中遇到了个场景，一个go服调用下游redis的QPS太高，导致redis报错比较多，所以决定在go服务里起个本地缓存，减少对redis访问，上线后效果还不错，对redis的访问一下降了90%左右。今天我们就来看下怎么用go语言实现本地内存缓存吧。

1. 明确目标

首先，我们要明确需要实现哪些特性。本地内存缓存最基本的是个K-V的存储，key一般是string，value为了通用，定义成interface{}。另外，还要有过期删除功能，避免一直读到本地的缓存，数据更新有及时同步，这个过期时间通常由调用方传入。最后，考虑需不需要限制内存使用，在我实际的场景，我是没有限制的，因为我缓存的内容其实很少，而且我设置的过期时间也很短，确定不会占用很大存。总结一下就是：

- 实现一个K-V存储，key是string，value是interface{}
- 支持指定key的过期时间，内部实现过期删除
- 可选实现限制内存使用

2. 代码实现

talk is cheap，下面直接贴完整代码，注释比较详细，相信大家都能看懂：

```
package main

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

// 缓存对象
type CacheItem struct {
    Value     interface{}   // 实际缓存的对象
    LifeTime  time.Duration // 存活时间，上游传入
    CreatedAt time.Time     // 创建时间，和存活时间一起决定是否过期
}

// 缓存是否过期
func (item *CacheItem) Expired() bool {
    return time.Now().Sub(item.CreatedAt) > item.LifeTime
}

// 本地缓存实现类
type LocalCache struct {
    sync.RWMutex                  //继承读写锁，用于并发控制
    Items       map[string]*CacheItem // K-V存储
    GCDuration  int                   // 惰性删除，后台运行时间间隔，单位秒
}

// 新建本地缓存
func NewLocalCache(gcDuration int) *LocalCache {
    localCache := &LocalCache{Items: map[string]*CacheItem{}, GCDuration: gcDuration}
```

```
    // 启动协程，定期扫描过期键，进行删除
    go localCache.GC()

    return localCache
}

// 存入对象
func (cache *LocalCache) Put(key string, value interface{}, lifeTime time.Duration) {
    cache.Lock()
    defer cache.Unlock()

    cache.Items[key] = &CacheItem{
        Value:     value,
        LifeTime:  lifeTime,
        CreatedAt: time.Now(),
    }
}

// 查询对象
func (cache *LocalCache) Get(key string) interface{} {
    cache.RLock()
    defer cache.RUnlock()

    if item, ok := cache.Items[key]; ok {
        if !item.Expired() {
            return item
        } else {
            // 键已过期，直接删除
            // 需要注意的是，这里不能调用cache.Del()方法，因为go的读写锁是不支持锁升级的，会发
死锁
            delete(cache.Items, key)
        }
    }

    return nil
}

// 删除缓存
func (cache *LocalCache) Del(key string) {
    cache.Lock()
    defer cache.Unlock()

    if _, ok := cache.Items[key]; ok {
        delete(cache.Items, key)
    }
}

// 异步执行，扫描过期键并删除
func (cache *LocalCache) GC() {
    for {
        select {
        case <-time.After(time.Duration(cache.GCDuration) * time.Second):
            keysToExpire := []string{}
```

```
            cache.RLock()
            for key, item := range cache.Items {
                if item.Expired() {
                    keysToExpire = append(keysToExpire, key)
                }
            }
            cache.RUnlock()

            for _, keyToExpire := range keysToExpire {
                cache.Del(keyToExpire)
            }
        }
    }
}
```

以上就是go语言里一个简单的本地缓存的实现了，如果大家只是轻度使用，不重度依赖本地缓存的话直接自己手写一遍就行了，如果是重度依赖的话，建议还是找个开源的比较完善的实现，比如下面我推荐的go-cache模块。

3. 开源实现：go-cache

点击查看源码：github地址。

它的实现原理和上面的差不多，只是考虑了更多细节，比如不使用defer，来提升性能，处理gc的协监听了一个关闭管道，使得我们可以从外部停止gc协程，以及注册finalizer函数，保证可以优雅关闭g协程，并提供了更多有用的API，代码结构也更规范，合理，推荐大家使用。