

# 关于视频课

- 目的：在实际项目中，我们学习的知识是如何被使用的
- 涉及到代码的部分，建议在 PC 上收看

# 13.基于 FFI 实现的 lua-resty-lrucache

# 学习目的

- lua-resty-\*库的一般结构
- FFI 的使用
- 前面几个章节的回顾
- 为测试案例章节做铺垫

# lua-resty-lrucache 是什么

- 基于 LuaJIT FFI 实现的 LRU 缓存
- 单个 worker 内缓存
- 不涉及到 OpenResty 相关 API

# 学习 lua-resty-\*库，先看什么？



- 文档
- 目录结构
- 测试案例

## Name

lua-resty-lrucache - Lua-land LRU Cache based on LuaJIT FFI

## Table of Contents

- [Name](#)
- [Status](#)
- [Synopsis](#) ← 示例代码
- [Description](#) ← 原理讲解
- [Methods](#)
  - [new](#)
  - [set](#)
  - [get](#)
  - [delete](#)
  - [flush\\_all](#)API
- [Prerequisites](#)
- [Installation](#)
- [TODO](#)
- [Community](#)
  - [English Mailing List](#)
  - [Chinese Mailing List](#)
- [Bugs and Patches](#)
- [Author](#)
- [Copyright and License](#)
- [See Also](#)

# 最快上手的代码段



```
-- file myapp.lua: example "myapp" module

local _M = {}

-- alternatively: local lrucache = require "resty.lrucache.pureffi"
local lrucache = require "resty.lrucache"

-- we need to initialize the cache on the lua module level so that
-- it can be shared by all the requests served by each nginx worker process:
local c, err = lrucache.new(200) -- allow up to 200 items in the cache
if not c then
    return error("failed to create the cache: " .. (err or "unknown"))
end

function _M.go()
    c:set("dog", 32)
    c:set("cat", 56)
    ngx.say("dog: ", c:get("dog"))
    ngx.say("cat: ", c:get("cat"))

    c:set("dog", { age = 10 }, 0.1) -- expire in 0.1 sec
    c:delete("dog")

    c:flush_all() -- flush all the cached data
end

return _M
```

注释中包含重要信息

错误处理

覆盖每一个 API

# 更进一步，看文档

- 看完示例代码，可以明白大概的使用方法
- 更详细的参数和返回值，还是需要看文档
- 文档不明白，不要着急看源码，而是要看测试案例



## get

---

syntax: `data, stale_data = cache.get(key)`

Fetches a value with the key. If the key does not exist in the cache or has already expired, a `nil` value will be returned.

Starting from `v0.03`, the stale data is also returned as the second return value if available.










[Back to TOC](#)



```
145 === TEST 4: ttl
146 --- http_config eval: $::HttpConfig
147 --- config
148     location = /t {
149         content_by_lua '
150             local lrucache = require "resty.lrucache"
151             local c = lrucache.new(1)
152
153             c:set("dog", 32, 0.5)
154             ngx.say("dog: ", c:get("dog"))
155
156             ngx.sleep(0.25)
157             ngx.say("dog: ", c:get("dog"))
158
159             ngx.sleep(0.26)
160             ngx.say("dog: ", c:get("dog"))
161         ';
162     }
163 --- request
164     GET /t
165 --- response_body
166     dog: 32
167     dog: 32
168     dog: nil32
169
170 --- no_error_log
171     [error]
172
```

# 目录结构



 lib/resty	Lua 代码	bumped version t
 t	测试案例	feature: added ne
 .gitattributes		added a .gitattrib
 .gitignore		initial checkin.
 .travis.yml	CI	travis: added a st
 Makefile		makefile: install r
 README.markdown	文档	doc: fixed a mino
 dist.ini	OPM	added dist.ini file
 valgrind.suppress	内存泄漏误报	added new

# travis.yml

- GitHub 集成的插件
- 通过 travis CI 后，PR 才会被合并
- 熟悉项目依赖和安装步骤的最佳方式

openresty / lua-resty-lrucache

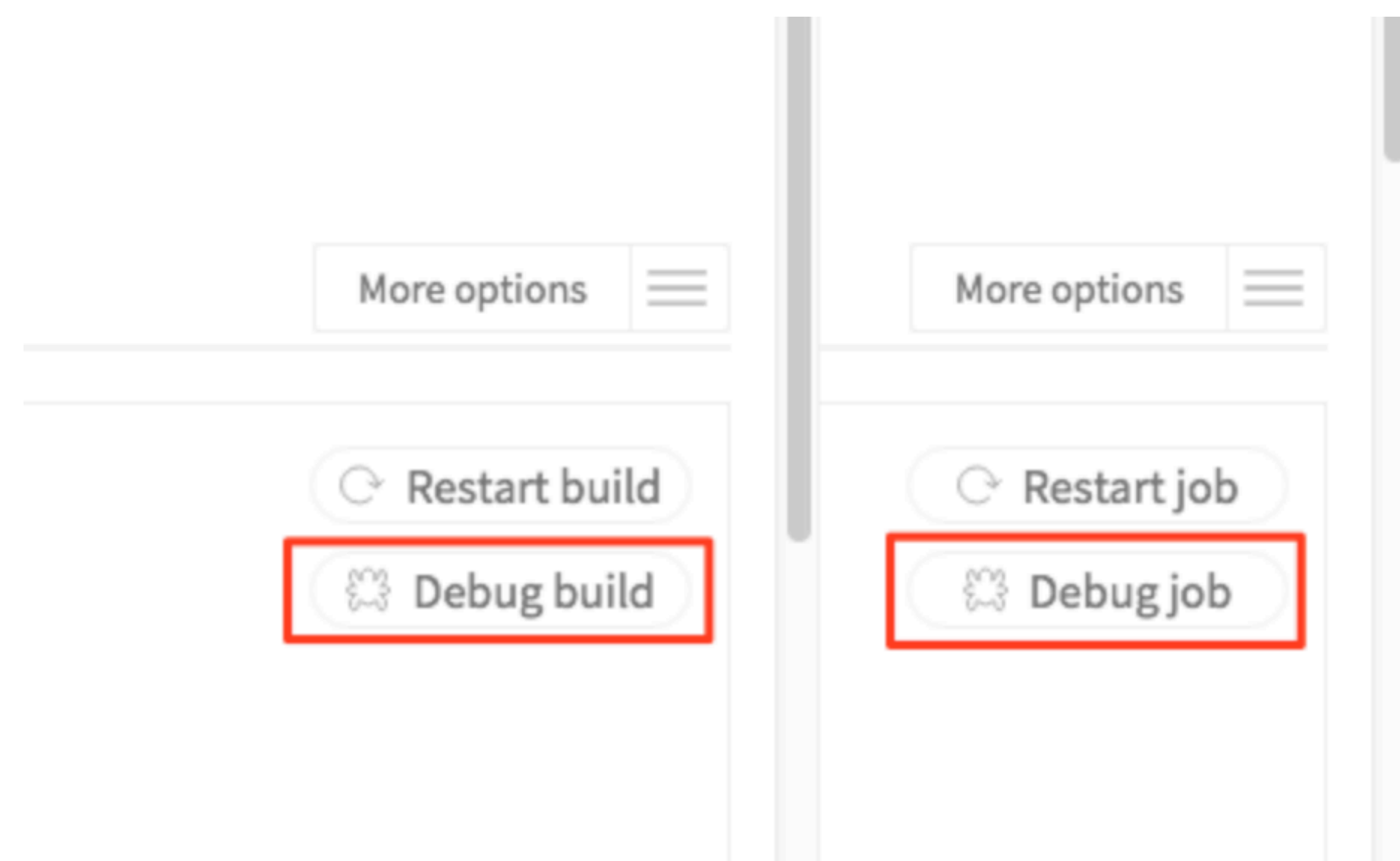
<> Code 1 Issues 6 Pull requests Proje

Branch: master lua-resty-lrucache / .travis.yml

```
49 script:
50   - cd luajit2/
51   - make -j$JOBS CCDEBUG=-g Q= PREFIX=$LUAJIT_PREFIX CC=$CC XCFLAGS='-DLI
52   - sudo make install PREFIX=$LUAJIT_PREFIX > build.log 2>&1 || (cat bui
53   - cd ..
54   - tar zxf download-cache/openssl-$OPENSSL_VER.tar.gz
55   - cd openssl-$OPENSSL_VER/
56   - ./config shared --prefix=$OPENSSL_PREFIX -DPURIFY > build.log 2>&1 |
57   - make -j$JOBS > build.log 2>&1 || (cat build.log && exit 1)
58   - sudo make PATH=$PATH install_sw > build.log 2>&1 || (cat build.log &
59   - cd ../mockeagain/ && make CC=$CC -j$JOBS && cd ..
60   - export PATH=$PWD/work/nginx/sbin:$PWD/nginx-devel-utils:$PATH
61   - export LD_PRELOAD=$PWD/mockeagain/mockeagain.so
62   - export LD_LIBRARY_PATH=$PWD/mockeagain:$LD_LIBRARY_PATH
63   - export TEST_NGINX_RESOLVER=8.8.4.4
64   - export NGX_BUILD_CC=$CC
65   - ngx-build $NGINX_VERSION --with-ipv6 --with-http_realip_module --witl
66   - nginx -V
67   - ldd `which nginx` |grep -E 'luajit|ssl|pcre'
68   - prove -r t
```

上面的 60 多行都是给 prove 来做  
铺垫；  
prove 是 perl 中跑测试的工具，其  
中的 t 即 /t 目录。

- travis CI 是可以 ssh 登录并调试的：私有库自带，开源库需要申请



# 代码的实现

- lua-resty-lrucache 包含两个版本的实现：resty.lrucache 和 resty.lrucache.pureffi，暴露的 API 是一样的
- 前者的缓存是 Lua table 实现，pureffi 是基于 FFI 的 hash 表
- 如果缓存命中率高，使用resty.lrucache；命中率低，使用 resty.lrucache.pureffi。因为 Lua table 不适合频繁的删除元素
- 我们以 resty.lrucache 为例介绍



openresty / lua-resty-lrucache

<> Code 1 Issues 6 Pull requests 0 Projects

Branch: master lua-resty-lrucache / lib / resty / lrucache.lua

```
~
4  local ffi = require "ffi"
5  local ffi_new = ffi.new
6  local ffi_sizeof = ffi.sizeof
7  local ffi_cast = ffi.cast
8  local ffi_fill = ffi.fill
9  local ngx_now = ngx.now
10 local uintptr_t = ffi.typeof("uintptr_t")
11 local setmetatable = setmetatable
12 local tonumber = tonumber
13
```

变量 local 化，加快查找

```
--
32 -- queue data types
33 --
34 -- this queue is a double-ended queue and the first node
35 -- is reserved for the queue itself.
36 -- the implementation is mostly borrowed from nginx's ngx_queue_t data
37 -- structure.
38
39 ffi.cdef[[
40     typedef struct lrucache_queue_s  lrucache_queue_t;
41     struct lrucache_queue_s {
42         double          expire; /* in seconds */
43         lrucache_queue_t *prev;
44         lrucache_queue_t *next;
45     };
46 ]]
47
48 local queue_arr_type = ffi.typeof("lrucache_queue_t[?]")
49 local queue_type = ffi.typeof("lrucache_queue_t")
50 local NULL = ffi.null
51
```

FFI 定义的数据结构

双向链表实现

可变长度数组

VLA(variable-length array)

- 双端队列(缩写为Deque)： 可以从前或后添加、删除元素
- OpenResty 中有多种空值： Lua 的 Nil, ngx.null, ffi.null, cJSON.null。再加上字符串和布尔值，判断 true、false 不是一件容易的事情（后面专门讲）

# 初始化



```
132 local _M = {  
133     _VERSION = '0.09'  
134 }  
135 local mt = { __index = _M }  
136
```

用 table 和 metatable 模拟类

```
143 function _M.new(size)  
144     if size < 1 then  
145         return nil, "size too small"  
146     end  
147
```

```
148     local self = {  
149         hasht = {},  
150         free_queue = queue_init(size),  
151         cache_queue = queue_init(),  
152         key2node = {},  
153         node2key = {},  
154     }
```

保存 key 和 value, TTL 在队列中保存

空闲队列

缓存队列初始化大小为 0

O(1)时间复杂度查找

```
155     return setmetatable(self, mt)  
156 end  
157
```

# 队列初始化



```
63
64  local function queue_init(size)
65      if not size then
66          size = 0
67      end
68      local q = ffi_new(queue_arr_type, size + 1)
69      ffi_fill(q, ffi_sizeof(queue_type, size + 1), 0)
70
```

VLA 在 `ffi.new` 时确定长度

填充队列

# get 接口的实现



```
158
159 function _M.get(self, key)
160     local hasht = self.hasht ← 没有必要做 local 化, 只用了一次
161     local val = hasht[key]
162     if val == nil then
163         return nil 1 个返回值
164     end
165
166     local node = self.key2node[key]
167
168     -- print(key, ": moving node ", tostring(node), " to cache queue head")
169     local cache_queue = self.cache_queue LRU: 最近使用的放在最前面
170     queue_remove(node)
171     queue_insert_head(cache_queue, node)
172
173     if node.expire >= 0 and node.expire < ngx_now() then
174         -- print("expired: ", node.expire, " > ", ngx_now())
175         return nil, val ← 2 个返回值
176     end
177     return val 返回 stale 数据, 文档和测试案例印证
178 end
179
```

# 回顾

- 从文档、测试案例和源码来逐步深入
- lua-resty-lrucache 的数据结构和具体实现

**Q&A**