

# **Skynet 框架之菜鸟手册**

一个轻量级的网络游戏服务器

**Ver 0.1.1**

**wangdali <wangdali@qq.com>**

**2014-04**

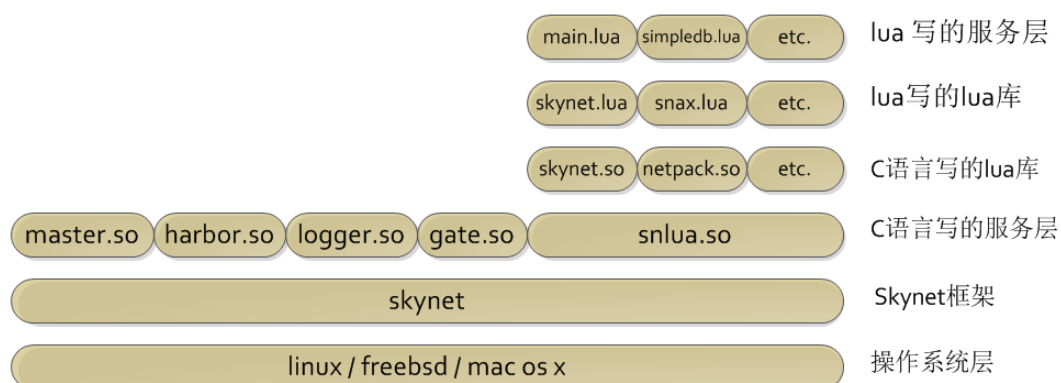


## 目录

一、简介.....	4
二、环境.....	4
1、VMware 下载与安装 .....	4
2、Ubuntu 下载与安装 .....	5
3、Ubuntu 环境配置 .....	6
三、下载.....	9
1、git 使用 .....	9
2、github.com .....	12
四、编译.....	13
1、Makefile 介绍.....	13
2、核心程序介绍.....	17
五、例子.....	20
1、运行例子.....	20
2、代码分析.....	22
六、原理.....	26
1、启动流程.....	26
2、调用服务.....	27
七、服务.....	27
1、用 C 语言写一个服务.....	27
2、用 Lua 语言写一个服务 .....	28

## 一、简介

Skynet 是一个轻量级的网络服务器架构，并不是一个完整的游戏服务端。是服务端的最底层框架，和游戏有关的各种服务都是基于架构之上开发的。所以别想着拿 Skynet 改改就能作为网络游戏（包括私服），跑起来，甚至怎么监听客户端连接的服务都要你自己来写。Skynet 的功能只有管理好服务（加载与调度）和服务之间的调用（请求与响应）。Skynet 今后大部分都要在 lua 脚本下开发，只有需要考虑性能模块才用 C 语言开发成 lua 库，提供给 lua 调用。现在 Skynet 提供给 lua 使用的库还不多，期望以后这些库多了，可以方便、简单的完成一个网络游戏服务端的开发。



## 二、环境

### 1、VMware 下载与安装

我们既然是菜鸟，就在虚拟中安装 Linux 好了。首先我们得先选择一款虚拟机软件，有微软的 Virtual PC 这个是免费的；甲骨文的

VirtualBox 也是免费的；我们选择最常用的 VMware 好了，这有两个版本，一个叫 VMware Player，这个是免费的，可以在官方网站 <http://www.vmware.com> 中下载，另一个叫 VMware Workstation，这个是收费的。以上的几个随便选一个都可以用，不过我们菜鸟还是不差钱，就选收费的 VMware Workstation 好了。

安装 VMware Workstation 要写好多页纸，这里咱们就环保了吧，不懂安装的请找谷歌或者百度帮忙。现在 VMware Workstation 最高版本是 10 版。我们就选择这个了。

## 2、Ubuntu 下载与安装

Skynet 目前可以在以下操作系统上编译：1、Linux； 2、FreeBSD； 3、Mac OS X 下编译使用。需要提醒的是不能在咱们菜鸟最喜欢的 Windows 下编译使用，虽然有老鸟在干这个事，但暂时没有可用的版本。

咱菜鸟还是选择 Linux 下使用好了，这个网上能查到的资料多，不懂得都能搜索到。既然决定用 Linux 我们就选择一个发行版简单点，常用的发行版有：1、RedHat 红帽这个很有名，咱就不用了，你非要用我也不管，后面的内容可能会不适用；2、CentOS 这个就是 RedHat，不介绍了；3、Ubuntu 这个可是十大受欢迎发行版之首。咱就选它好了，简单易用。

Ubuntu 很友好的给咱们国内定制了一个版本，叫 Kylin，中文叫麒麟。下载地址：<http://www.ubuntu.com/desktop/ubuntu-kylin-zh-CN>

根据你的 Windows 版本选择麒麟的版本,有 32 位和 64 位之分。  
32 位的 Windows 下不能装 64 位版本的麒麟。所以请看在 windows  
“计算机”处右键“属性”查看版本。



我的是“64 位版操作系统”，对应下载  
ubuntukylin-14.04-desktop-amd64.iso 。

在 VMware 中安装 Ubuntu 的文章网上很多，而且安装过程很简单，这里就不叙述了。

### 3、Ubuntu 环境配置

Ubuntu 安装好之后，还需要安装一些程序才能编译 Skynet，所以我们先在这里配置一下，第一步我们需要启动一个命令行的“终端”，用命令来安装简单一点。

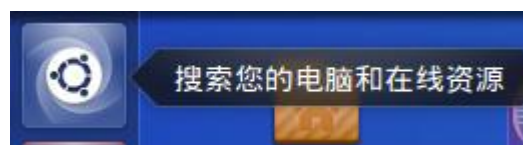
因为 VMware 自动安装的原因，我装完是英文的，你可以把它设

置成中文。 里面找到  用户账户 设置用户的语言为汉语



首先我们得找到“终端”并启动它。我们启动 Ubuntu 在虚拟机

中屏幕的左上角会看到这个图标：




请点击一下，新安装这里面没有曾经启动和使用过的文件列表。我们

需要选择：



第二

个 ，这个时候会看到



这里面就是所有你已经安装的软件列表了，相当于 Window 的“开始”

一〉“程序”。上面提示还有 76 个结果没有显示出来，点一下把他们



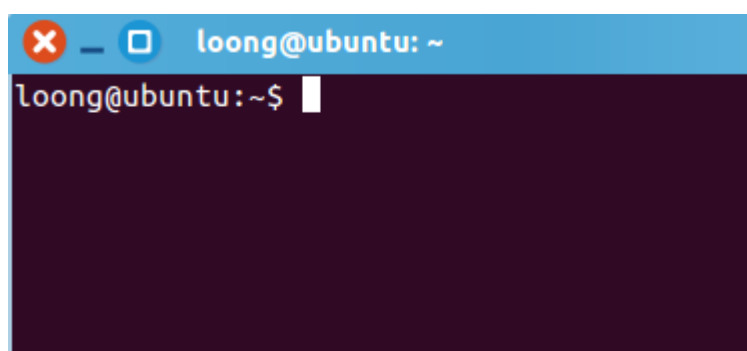
都显示出来。找到

这个就是我们需要的终端了。然



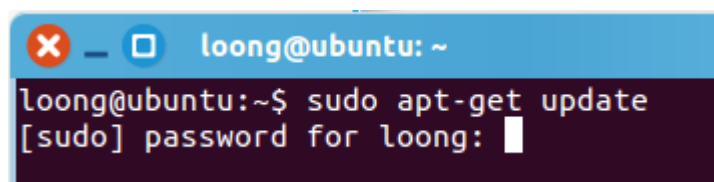
后屏幕左边的菜单可以看到

右键点击一下，把“终端”“锁定到启动器”这样以后就不用到里面找“终端”了，相当 Windows 把图标锁定到任务栏。桌面上还可以看到“终端”的窗口，类似于：



好了，我们可以回到正题了，首先我们把 Ubuntu 更新到最新版本，在“终端”输入：

```
$ sudo apt-get update
```

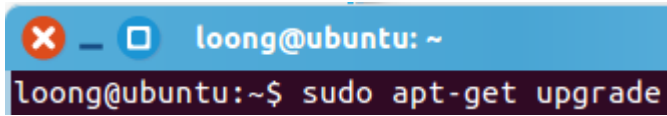


意思是叫你输入 loong 这个用户的密码（我安装的时候起名叫 loong，意为龙，你可以用别的名字，一定要记住密码），用来启动超



级用户的权限，相当于 Windows 的“以管理员身份运行”。输入密码等待自动更新软件列表完成，接着就是更新软件了。在“终端”输入：

```
$ sudo apt-get upgrade
```



等待软件更新完成。虽然可以同时开多个“终端”但不能同时用来更新和安装软件，咱菜鸟们还是老老实实的等它们完成吧。如果出现错误和无法下载、无法解析之类的提示，就说明你的虚拟机没有联



网，自己搞定它，可以用 **firefox** 火狐浏览器上网看看。

安装 Skynet 需要的软件：1、autoconf 准确的说这个是 Skynet 下使用的子模块 jemalloc 编译的时候需要的。2、libreadline-dev。输入下面命令。

```
$ sudo apt-get install autoconf
```

```
$ sudo apt-get install libreadline-dev
```

执行命令的过程中可能会提示你需要下载多大的程序，安装需要多少空间，你是否继续，回答 Y 就好了。等待装完吧。

## 三、下载

### 1、git 使用

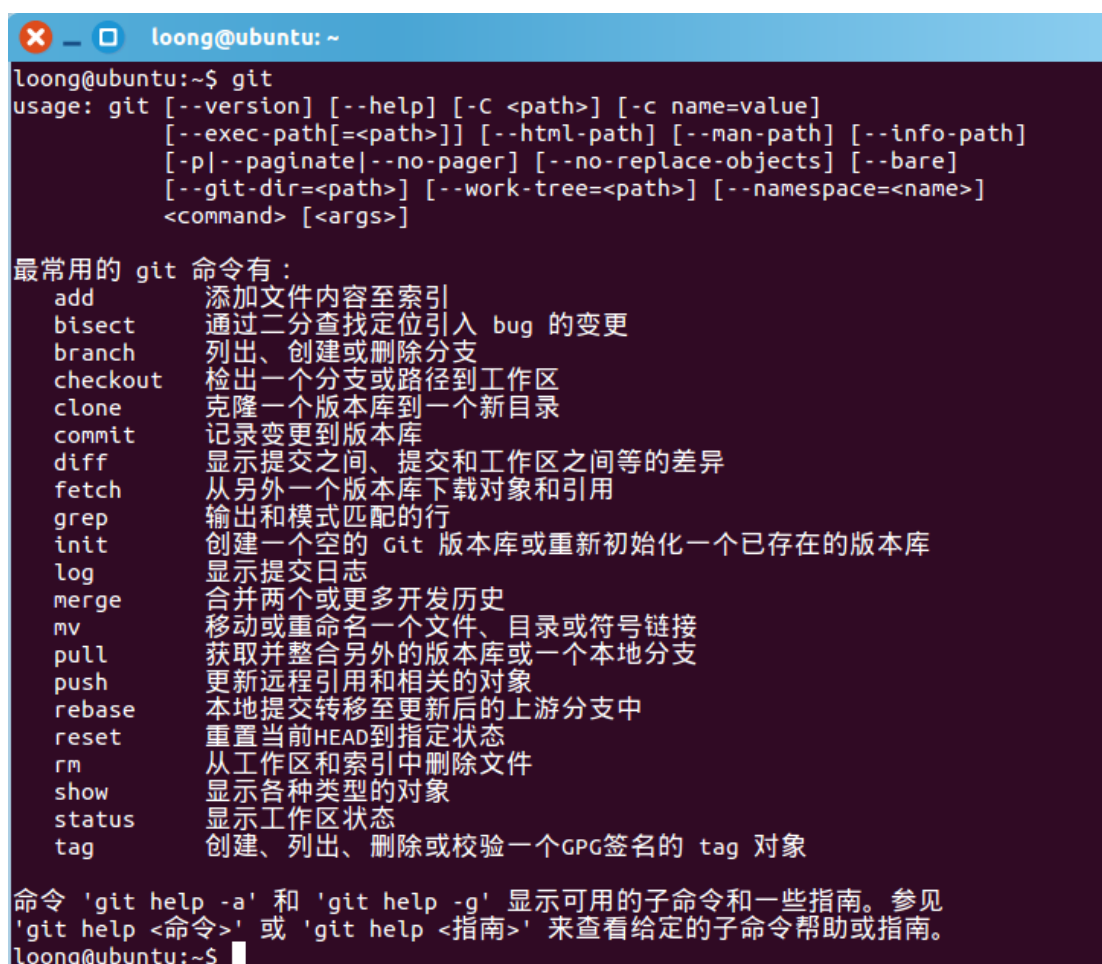
因为 Skynet 使用 git 来管理源代码的版本，所以最好还是简单的学习一下 git。默认的 Ubuntu 没有安装 git，我们需要自己装一下，打

开“终端”输入：

```
$ sudo apt-get install git
```

输入 git 看下使用方法

```
$ git
```

A terminal window titled 'loong@ubuntu: ~' showing the output of the 'git' command. The output includes the usage of git with various options like --version, --help, -C, -c, --exec-path, --html-path, --man-path, --info-path, --paginate, --no-pager, --no-replace-objects, --bare, --git-dir, --work-tree, --namespace, and <command>. Below this, it lists common git commands and their descriptions: add (add file content to index), bisect (find bug via binary search), branch (list/create/delete branch), checkout (check out branch/path to work area), clone (clone repository to new directory), commit (record changes to repository), diff (show differences between commits/work area), fetch (download objects and refs from remote), grep (output lines matching pattern), init (create empty git repository), log (show commit history), merge (merge two or more development histories), mv (move/rename file/directory/symbolic link), pull (fetch and merge remote repository or local branch), push (update remote refs and objects), rebase (rebase local commits onto upstream branch), reset (reset HEAD to specified state), rm (remove files from work area and index), show (show various types of objects), status (show work area status), and tag (create/list/delete/verify GPG signed tag). At the bottom, it mentions that 'git help -a' and 'git help -g' show available subcommands and guides, and 'git help <command>' or 'git help <guide>' show help for specific commands or guides. The prompt 'loong@ubuntu:~\$' is visible at the bottom.

对于下载源代码，我们只需要知道“clone 克隆一个版本库到一个新目录”这一条怎么使用就好了。

接下来我们开始下载 Skynet 的源代码：

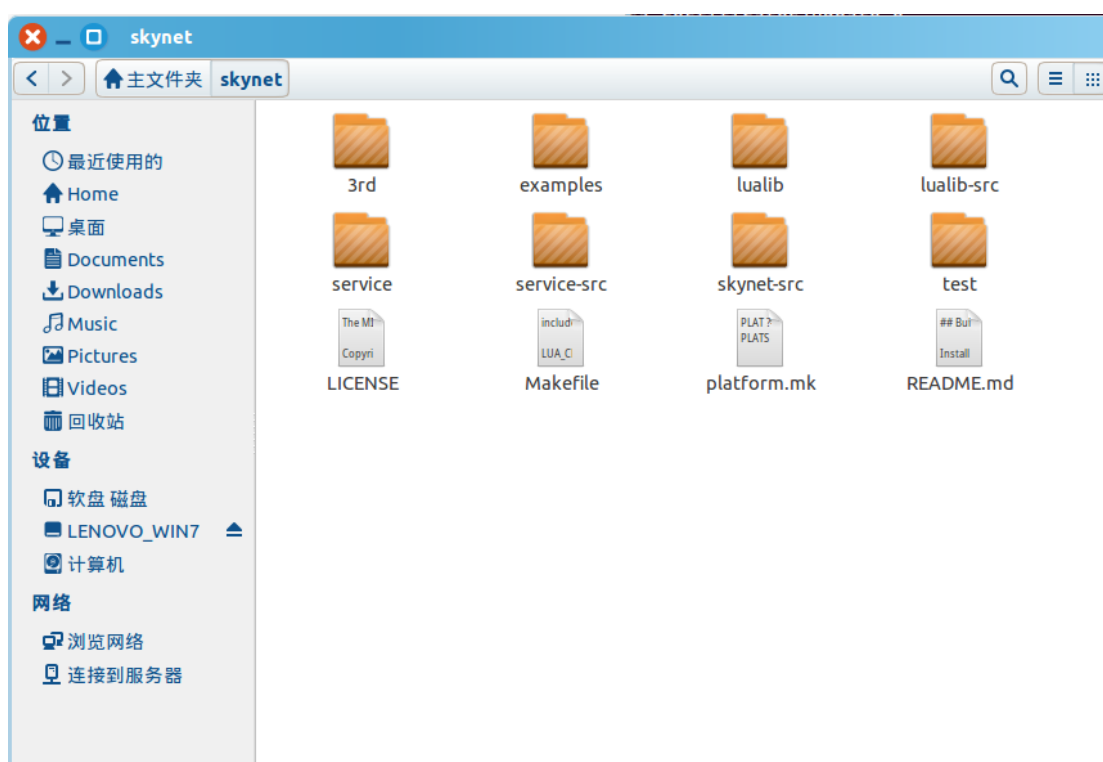
```
$ cd ~
```

```
$ git clone https://github.com/cloudwu/skynet.git
```

```
loong@ubuntu: ~  
loong@ubuntu:~$ git clone https://github.com/cloudwu/skynet.git  
正克隆到 'skynet'...  
remote: Counting objects: 3511, done.  
remote: Compressing objects: 100% (1574/1574), done.  
接收对象中: 100% (3511/3511), 1.22 MiB | 257.00 KiB/s, done.  
remote: Total 3511 (delta 2110), reused 3255 (delta 1918)  
处理 delta 中: 100% (2110/2110), done.  
检查连接... 完成。
```

到目前为止我们已经把 Skynet 的源代码下载到了

/home/loong/skynet 目录下:



上面就是源代码目录的内容,我们就简单的介绍一下目录里面都有什么东西吧:

README.md      简单介绍了怎么编译和测试 Skynet

LICENSE          许可证信息, 采用 MIT, 很宽松的协议。

Makefile          编译规则文件, 用于编译 Skynet

platform.mk	编译与平台相关的设置
3rd	第三方的代码，有 lua 和 jemalloc 等
examples	附带的例子
lualib	使用 lua 写的库
lualib-src	使用 C 写并封装给 lua 使用的库
service	使用 lua 写的 Skynet 的服务模块
service-src	使用 C 写的 Skynet 的服务模块
skynet-src	Skynet 的核心代码
test	使用 lua 写的一些测试代码

以上的核心代码为：skynet-src 目录和 service-src 目录中的代码。剩下的基本都是为了给 lua 脚本使用的。对于我们菜鸟根本不用关心 C 语言写的东西。完全在 lua 上开发就好了。底层的东西留给云风吧。

3rd 中的 jemalloc 是一个内存分配的库，用来提高 malloc 的性能，默认情况下在 linux 下是开启使用的，如果你不想使用可以在 platform.mk 文件中添加下面语句屏蔽掉：

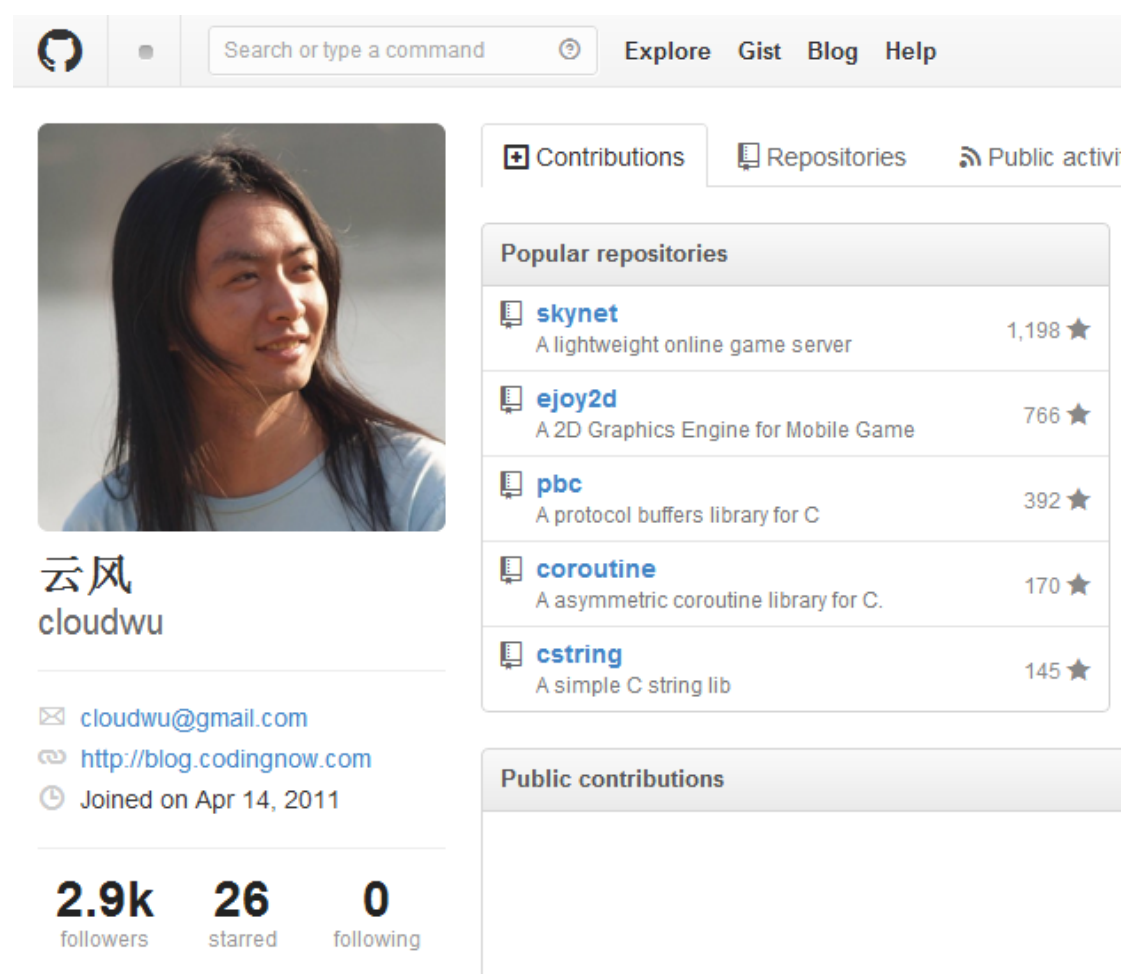
```
linux : SKYNET_DEFINES :=-DNOUSE_JEMALLOC
```

3rd 中的 lua 是云风修改过的，你也可以使用官方版本，修改的内容看这里：<http://lua-users.org/lists/lua-l/2014-03/msg00489.html>






## 2、github.com

Skynet 的源代码托管在 github.com 下，你可以访问：


<https://github.com/cloudwu> 看到云风的其他开源代码，比如 ejoy2d 游戏引擎等。



The screenshot shows the GitHub profile of a user named 'cloudwu' (云风). The profile includes a profile picture of a man with long dark hair, a bio, email address (cloudwu@gmail.com), website (http://blog.codingnow.com), and join date (Apr 14, 2011). It also displays statistics: 2.9k followers, 26 starred repositories, and 0 following. A list of popular repositories is shown, including 'skynet' (1,198 stars), 'ejoy2d' (766 stars), 'pbc' (392 stars), 'coroutine' (170 stars), and 'cstring' (145 stars). The 'Public contributions' section is also visible.

Popular repositories		
	<b>skynet</b> A lightweight online game server	1,198 ★
	<b>ejoy2d</b> A 2D Graphics Engine for Mobile Game	766 ★
	<b>pbc</b> A protocol buffers library for C	392 ★
	<b>coroutine</b> A asymmetric coroutine library for C.	170 ★
	<b>cstring</b> A simple C string lib	145 ★

八卦下，云风现在创业忙，瘦了，还是以前这张帅。要多注意身体呀。

点击  Repositories 有更多的开源项目，在此谢谢云风。

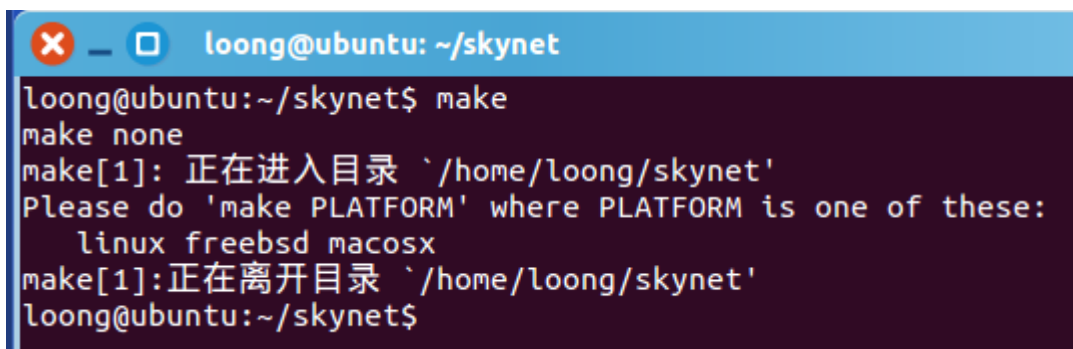
## 四、编译

### 1、Makefile 介绍

唠叨了半天终于到这一步了。本手册是面对菜鸟的，没办法说了一大堆没用的东西。

在 linux 一般都是使用 Makefile 来管理项目的，相当于我们在 windows 下使用 Visual Studio 的 .sln 。linux 编译很简单，一般就是：

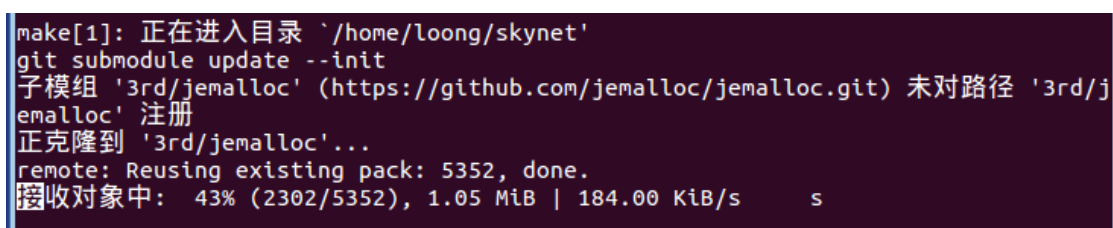
\$ make

A terminal window titled 'loong@ubuntu: ~/skynet' showing the output of the 'make' command. The output indicates that 'make none' was executed, and it prompts the user to specify a platform (linux, freebsd, or macosx) using 'make PLATFORM'. The user has not yet specified a platform, so the terminal shows the prompt again.

```
loong@ubuntu:~/skynet$ make
make none
make[1]: 正在进入目录 `/home/loong/skynet'
Please do 'make PLATFORM' where PLATFORM is one of these:
    linux freebsd macosx
make[1]:正在离开目录 `/home/loong/skynet'
loong@ubuntu:~/skynet$
```

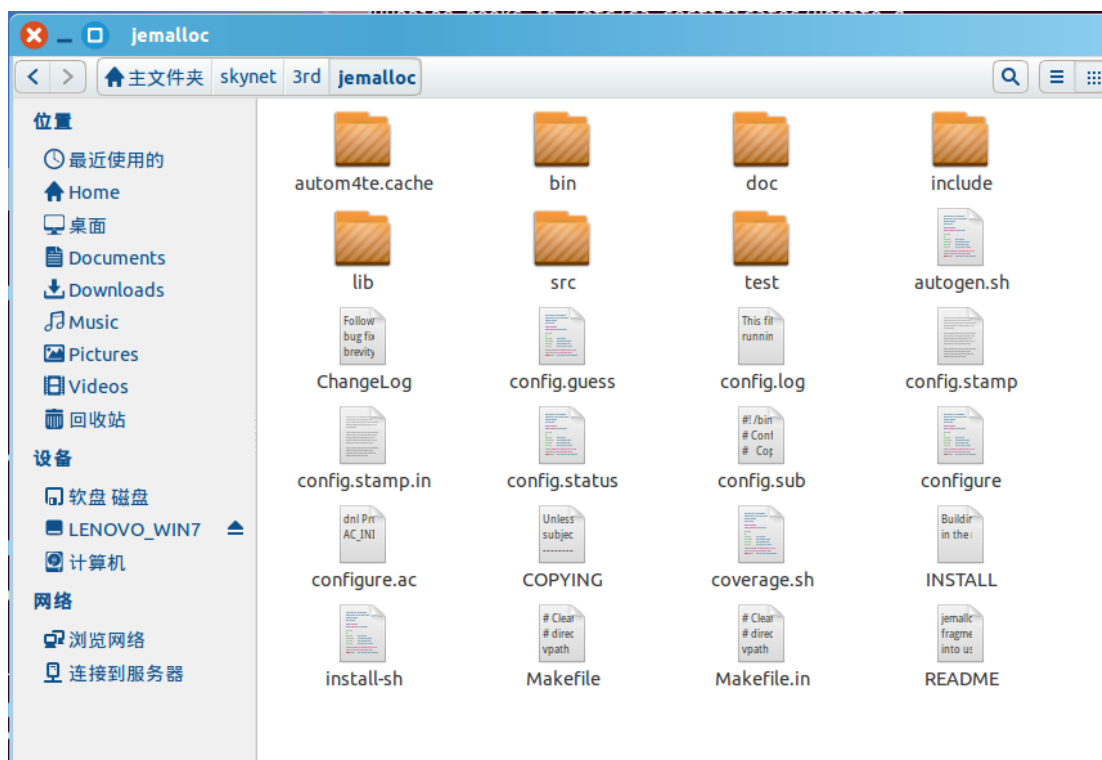
意思是你 make 的时候要带有平台名，因为我们是在 linux 下：

\$make linux

A terminal window showing the output of the 'make linux' command. It shows 'git submodule update --init' being executed, which clones the '3rd/jemalloc' submodule. The output indicates that the submodule was successfully cloned and updated.

```
make[1]: 正在进入目录 `/home/loong/skynet'
git submodule update --init
子模组 '3rd/jemalloc' (https://github.com/jemalloc/jemalloc.git) 未对路径 '3rd/jemalloc' 注册
正克隆到 '3rd/jemalloc'...
remote: Reusing existing pack: 5352, done.
接收对象中: 43% (2302/5352), 1.05 MiB | 184.00 KiB/s    s
```

会自动的去下载 git submoudle 的子项目，即 3rd/jemalloc 目录中的文件，刚开始这个目录是空的，现在有内容了。



编译成功后会看到:

```
loong@ubuntu: ~/skynet
ice/harbor.so -Iskynet-src
mkdir luaclib
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-skynet.c lualib-src/lu
a-seri.c -o luaclib/skynet.so -Iskynet-src -Iservice-src -Ilualib-src
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-socket.c -o luaclib/so
cketdriver.so -Iskynet-src -Iservice-src
cc -g -O2 -Wall -I3rd/lua -fPIC --shared 3rd/lua-int64/int64.c -o luaclib/int6
4.so
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-bson.c -o luaclib/bson
.so -Iskynet-src
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-mongo.c -o luaclib/mon
go.so -Iskynet-src
cc -g -O2 -Wall -I3rd/lua -fPIC --shared -I3rd/lua-md5 3rd/lua-md5/md5.c 3rd/l
ua-md5/md5lib.c 3rd/lua-md5/compat-5.2.c -o luaclib/md5.so
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-netpack.c -Iskynet-src
-o luaclib/netpack.so
cd 3rd/lua-cjson && make LUA_INCLUDE_DIR=../../3rd/lua CC=cc CJSON_LDFLAGS="-fPI
C --shared" && cp cjson.so ../../luaclib/cjson.so
make[2]: 正在进入目录 `/home/loong/skynet/3rd/lua-cjson'
cc -c -O3 -Wall -pedantic -DNDEBUG -I../../3rd/lua -fpic -o lua_cjson.o lua_cjs
on.c
cc -c -O3 -Wall -pedantic -DNDEBUG -I../../3rd/lua -fpic -o strbuf.o strbuf.c
cc -c -O3 -Wall -pedantic -DNDEBUG -I../../3rd/lua -fpic -o fpconv.o fpconv.c
cc -fPIC --shared -o cjson.so lua_cjson.o strbuf.o fpconv.o
make[2]: 正在离开目录 `/home/loong/skynet/3rd/lua-cjson'
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-clientsocket.c -o luac
lib/clientsocket.so -lpthread
cc -g -O2 -Wall -I3rd/lua -fPIC --shared -Iskynet-src lualib-src/lua-memory.c
-o luaclib/memory.so
cc -g -O2 -Wall -I3rd/lua -fPIC --shared lualib-src/lua-profile.c -o luaclib/p
rofile.so
make[1]: 正在离开目录 `/home/loong/skynet'
loong@ubuntu:~/skynet$
```

看看编译完后 skynet 目录下都多出了那些程序：



skynet

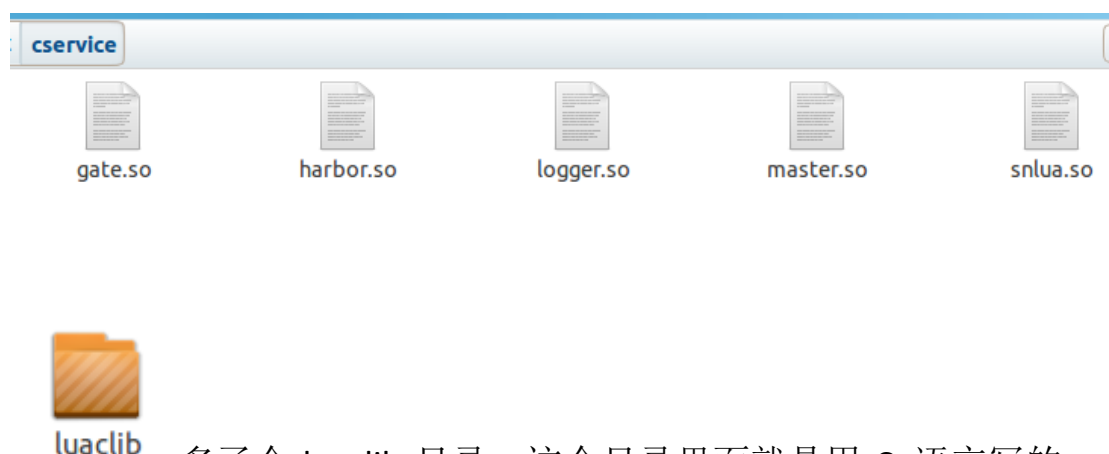
这个是主程序，linux 没有后缀名，相当 windows 下的.exe 文件。



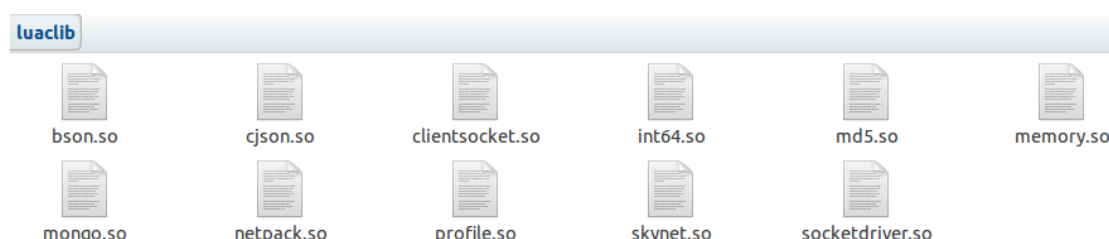
cservice

多了个 cservice 目录，这个目录里面就是用 C 语言写的 Skynet 的服务模块，即 service-src 编译后的动态链接库。我们看看都有哪些。





多了个 luaclib 目录，这个目录里面就是用 C 语言写的提供给 lua 脚本使用的库，即 lualib-src 编译后的动态链接库。



后面我们再介绍它的作用。

编译的其他东西我们就不关心了，在 3rd 目录里面。这里的 md5.so、int64.so、cjson.so 就是 3rd 里面编译出来的。

## 2、核心程序介绍



① **skynet** 主程序，架构从这里启动，可在“终端”中输入：

```
$ ./skynet examples/config
```

./说明是当前目录，skynet 是程序名，examples/config 为配置文件的路径和名字，这里的配置文件是用 lua 写的一个脚本。内容为：



```
config x
root = "./"
thread = 8
logger = nil
harbor = 1
address = "127.0.0.1:2526"
master = "127.0.0.1:2013"
start = "main"
standalone = "0.0.0.0:2013"
luaservice = root.."service/*.lua; "..root.."test/*.lua; "..root.."examples/*.lua"
snax = root.."examples/*.lua; "..root.."test/*.lua"
cpath = root.."cservice/*.so"
```

这里最重要的一条就是：`start="main"` 意思就是设置第一个启动的服务。这里指向 `examples/main.lua` 脚本服务模块。因为下面配置的 `luaservice` 的路径，所以不需要我们再输入 `examples/`了，`skynet` 会在配置的目录下找到 `main.lua` 脚本。`cpath` 是用 C 语言写的服务模块路径。`root="./"` 表示根目录为 `skynet` 启动的目录。`thread` 为启动的工作线程数。`logger=nil` 表示不记录日志，你可以指定一个路径和文件名，这样 “终端” 输出的内容就写到日志中文件了。

其中 `standalone = "0.0.0.0:2013"` 这个是 `master` 服务监听的地址和端口；`address` 和 `master` 是节点使用的，`address` 是节点的监听地址和端口，`master` 是节点要连接的 `master` 服务的地址。



② **master.so** 主服务，就做两件事，回应名字的查询和在更新名字后，同步到其他节点。其他节点在启动时会把自己的 `harbor` 注册到 `master` 服务中。这样透过 `master` 服务就能把各个节点都连在一起，相互沟通了。每个节点之间都相互建立一条双向的通讯通道。



③ **harbor.so** 节点服务，每个 Skynet 运行都是一个节点，在 `examples/main.lua` 中使用 `harbor` 来配置节点的编号，Skynet 限制只有 1 到 255 个节点（保留 0 给系统内部使用），并把 `harbor` 的 8bit 值放在服务 Id 的最高 8bit，因为服务 Id 是 32bit 的，所以 Skynet 的本地服务只能用低 24 位来表示，一个 Skynet 最多可以启动 16M 个服务模块。



④ **logger.so** 日志服务，一个简单的日志系统，可以用来记录服务的相关信息。



⑤ **gate.so** 网关服务，管理 Socket。



⑥ **snlua.so** lua 服务，用于加载 lua 写的 skynet 服务模块，而不仅仅限制于 C 语言才能写服务模块。十分核心和重要的模块。当然如果你不喜欢 lua，你也可以换成 python 等其他脚本引擎。

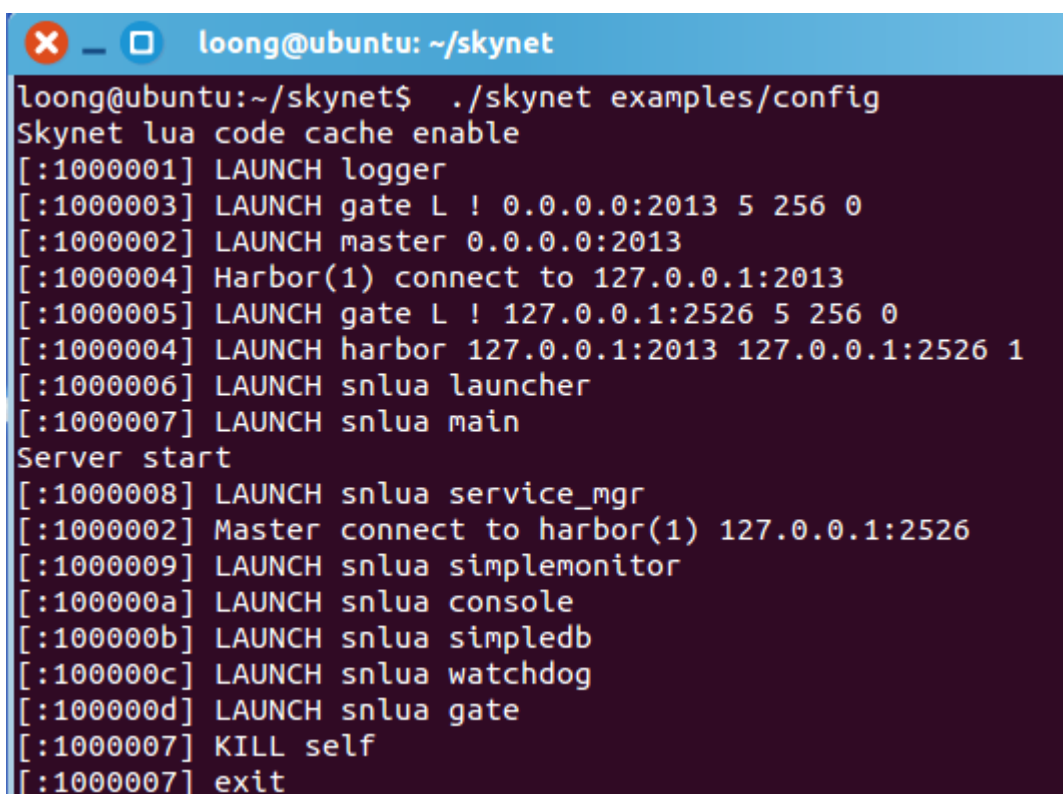
## 五、例子

### 1、运行例子

前面也说了，Skynet 带有一个例子，放在 examples 目录下，这个例子就是实现一个简单的功能，类似于 memcached。一个内存 Key-Value 系统。你可以通过指令 `set key value` 来设置一个 key 和 value，他们是相互对应的关系，然后可以通过 `get key` 来获得上面设置的 value 的这个值。

我们来启动这个例子看看：

`$ ./skynet examples/config`



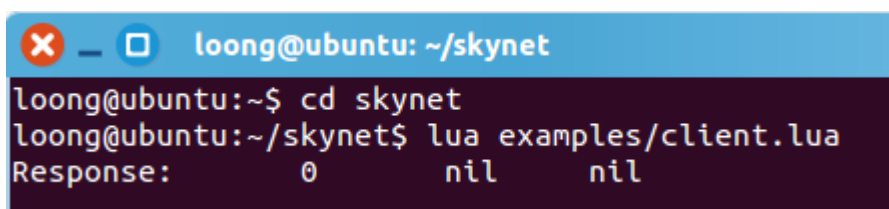
```
loong@ubuntu: ~/skynet
loong@ubuntu:~/skynet$ ./skynet examples/config
Skynet lua code cache enable
[:1000001] LAUNCH logger
[:1000003] LAUNCH gate L ! 0.0.0.0:2013 5 256 0
[:1000002] LAUNCH master 0.0.0.0:2013
[:1000004] Harbor(1) connect to 127.0.0.1:2013
[:1000005] LAUNCH gate L ! 127.0.0.1:2526 5 256 0
[:1000004] LAUNCH harbor 127.0.0.1:2013 127.0.0.1:2526 1
[:1000006] LAUNCH snlua launcher
[:1000007] LAUNCH snlua main
Server start
[:1000008] LAUNCH snlua service_mgr
[:1000002] Master connect to harbor(1) 127.0.0.1:2526
[:1000009] LAUNCH snlua simplemonitor
[:100000a] LAUNCH snlua console
[:100000b] LAUNCH snlua simpledb
[:100000c] LAUNCH snlua watchdog
[:100000d] LAUNCH snlua gate
[:1000007] KILL self
[:1000007] exit
```

启动成功，至于 `[:1000007]KILL self` 自杀是因为 `[:1000007]LAUNCH snlua main` 即 main.lua 服务已经完成了它的任务，

它自己在脚本中设置退出了。

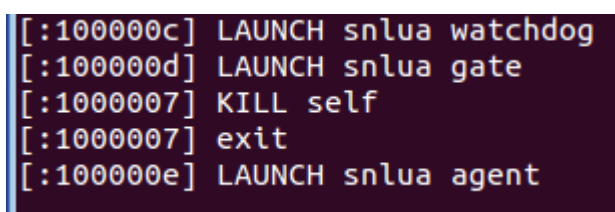
客户端，我们使用 `examples/client.lua` 即可。启动它

`$ lua examples/client.lua`



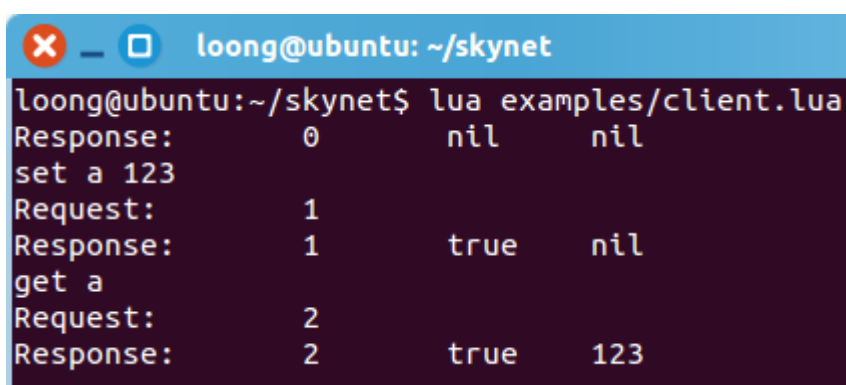
```
loong@ubuntu: ~/skynet
loong@ubuntu:~$ cd skynet
loong@ubuntu:~/skynet$ lua examples/client.lua
Response:      0      nil      nil
```

服务端会看到有客户端链接进来：



```
[ :100000c] LAUNCH snlua watchdog
[ :100000d] LAUNCH snlua gate
[ :1000007] KILL self
[ :1000007] exit
[ :100000e] LAUNCH snlua agent
```

每一个客户端都会启动一个 `agent` 处理它的请求。我们简单使用一下：



```
loong@ubuntu:~/skynet$ lua examples/client.lua
Response:      0      nil      nil
set a 123
Request:       1
Response:      1      true     nil
get a
Request:       2
Response:      2      true     123
```

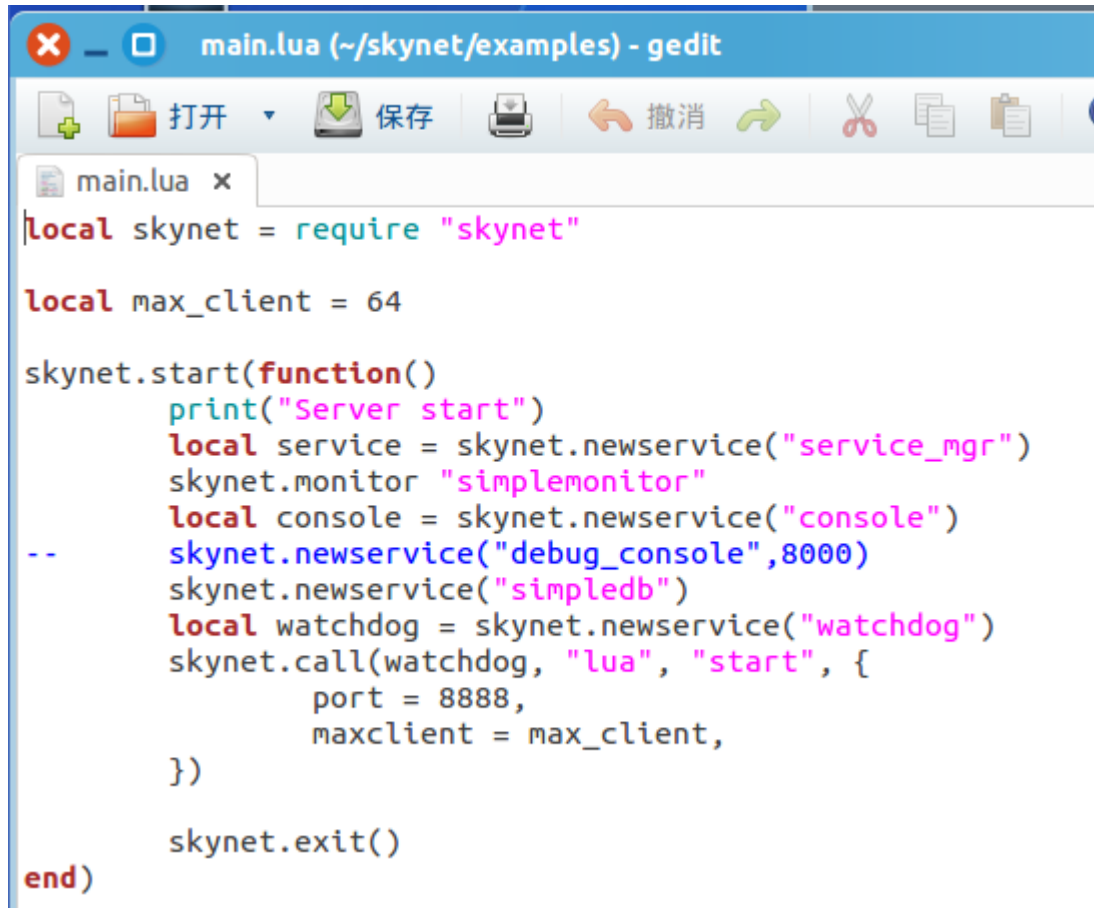
输入： `set a 123`

然后： `get a`

即，先设置 `a=123`，然后试下 `get a` 是不是获得 `a` 的值为 `123`。

## 2、代码分析

Skynet 启动 examples/config 脚本，通过 start="main" 开启了第一个 Skynet 的服务 main.lua，这个文件也在 examples 中，我们看看这个文件写的什么：



```
local skynet = require "skynet"

local max_client = 64

skynet.start(function()
    print("Server start")
    local service = skynet.newservice("service_mgr")
    skynet.monitor "simplemonitor"
    local console = skynet.newservice("console")
    -- skynet.newservice("debug_console",8000)
    skynet.newservice("simpledb")
    local watchdog = skynet.newservice("watchdog")
    skynet.call(watchdog, "lua", "start", {
        port = 8888,
        maxclient = max_client,
    })

    skynet.exit()
end)
```

第一行先引用 skynet 这个库，这个是用 lua 写的，即 lualib/skynet.lua。里面定义了这些接口：

skynet.start()用于服务的入口，加载 lua 服务时先运行这里的代码，它调用了 luacolib-src/lua\_skynet.c 里面的 callback()，最终调用 Skynet 的框架 skynet\_callback()来设置回调函数。

skynet.newservice() 用于启动一个 lua 写的服务，省略掉.lua 后缀

名。它调用了 `skynet.call()` 然后 `skynet.call()`调用 `luaclib-src/lua_skynet.c` 里面的 `send()`，最终调用 Skynet 的框架 `skynet_send()`压入队列。

`skynet.call()`用于发送一条消息给 Skynet 的框架。消息会压入队列，等待 Skynet 框架的调度。

`skynet.exit()`移除服务，通过 `skynet.send()`发送一条消息给 Skynet 框架来移除 lua 的这个服务。

`skynet.monitor()` 用于监视服务，看它是否关闭。

`main.lua` 一共打开了四个服务：

- 1、 `service_mgr` 这个是系统的模块，用于管理服务。
- 2、 `console` 这个是系统的模块，用于输出。
- 3、 `simplifiedb` 这个是例子的模块，用于管理 Key – Value 数据。
- 4、 `watchdog` 这个是例子的模块，用于监视 socket 端口，等待数据。

`main.lua` 没有调用其它函数，加载完服务，它也就完成了任务，所以它最后调用了 `skynet.exit()`把自己杀掉了。

现在 Skynet 已经启动了 `watchdog` 服务，监听着 8888 端口，等待客户端的连接。

下面是 `watchdog` 服务的 `skynet_start()`开始函数：

```
skynet.start(function()
    skynet.dispatch("lua", function(session, source, cmd, subcmd, ...)
        if cmd == "socket" then
            local f = SOCKET[subcmd]
            f(...)
            -- socket api don't need return
        else
            local f = assert(CMD[cmd])
            skynet.ret(skynet.pack(f(subcmd, ...)))
        end
    end)

    gate = skynet.newservice("gate")
end)
```

skynet.dispatch()这个服务的回调函数，通过 SOCKET[]来调用函数，这些函数有：

SOCKET.open() 打开 agent 服务并启动，使用 gate 来管理 socket。

SOCKET.close() 关闭 agent 服务。

SOCKET.error() 打印错误信息。

SOCKET.data() 有数据到来。

下面就来看看 agent 服务的代码：

```
function CMD.start(gate , fd)
    client_fd = fd
    skynet.call(gate, "lua", "forward", fd)
    send_client "Welcome to skynet"
end

skynet.start(function()
    skynet.dispatch("lua", function(_,_, command, ...)
        local f = CMD[command]
        skynet.ret(skynet.pack(f(...)))
    end)
end)
```

前面 watchdog 调用 SOCKET.open()的时候就调用了这里的 CMD.start()，在客户端输出了“Welcome to skynet”。

Agent 的核心就是注册了协议，并根据协议把数据发送给 simpledb 服务去处理：



```
skynet.register_protocol {
    name = "client",
    id = skynet.PTYPE_CLIENT,
    unpack = function (msg, sz)
        return jsonpack.unpack(skynet.tostring(msg,sz))
    end,
    dispatch = function (_, _, session, args)
        local ok, result = pcall(skynet.call, "SIMPLEDB", "lua", table.unpack(args))
        if ok then
            response_client(session, { true, result })
        else
            response_client(session, { false, "Invalid command" })
        end
    end
end
}
```

协议的详细部分看 `lualib/skynet.lua`。

最后我们看看 `simpledb` 服务：

```
local skynet = require "skynet"
local db = {}

local command = {}

function command.GET(key)
    return db[key]
end

function command.SET(key, value)
    local last = db[key]
    db[key] = value
    return last
end

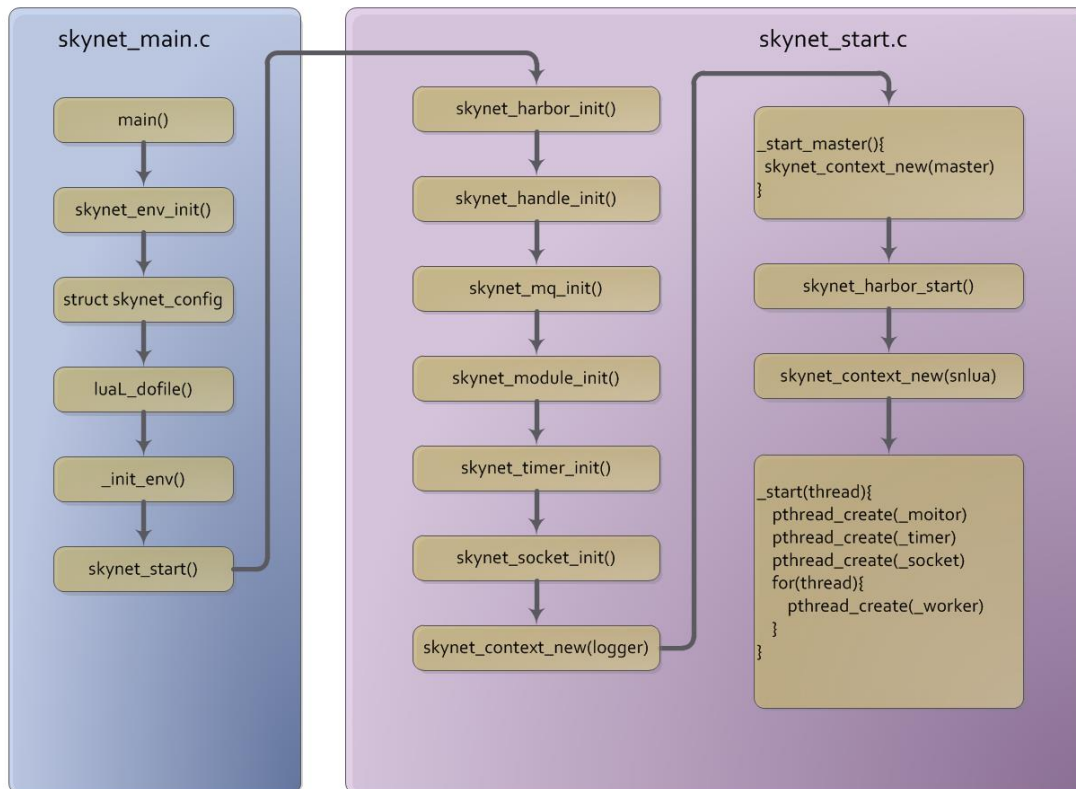
skynet.start(function()
    skynet.dispatch("lua", function(session, address, cmd, ...)
        local f = command[string.upper(cmd)]
        skynet.ret(skynet.pack(f(...)))
    end)
    skynet.register "SIMPLEDB"
end)
```

很简单的处理了 SET 和 GET。

以上只是大概浏览了一遍 Skynet 附带的例子，了解了一些 Skynet 提供给 lua 使用的接口，其他接口可以查看 `skynet.lua` 代码。

## 六、原理

### 1、启动流程



启动流程主要集中在两个文件中：

1 是 `skynet-src/skynet_main.c` 这个是 `main()` 函数所在，主要就是设置一下 lua 的环境、默认的配置、打开 config 配置文件，并修改默认配置。最后调用 `skynet_start()` 函数，这个函数在 `skynet_start.c` 文件中。

2 是 `skynet-src/skynet_start.c` 这个文件主要是初始化 Skynet 的各个模块，包括 harbor 节点、handle 服务 ID、mq 消息队列、module 加载动态链接库、timer 时钟、socket 套接字以及加载一些服务 logger 日志服务、master 主服务、harbor 节点服务、snlua 加载 lua 模块的

服务；以及最后启动几种线程包括 `_moitor`、`_timer`、`_socket` 和根据线程数启动 `n` 个工作线程。

这里需要注意的是 Skynet 将通过 `snlua` 服务加载第一个用户的服务：

```
ctx = skynet_context_new("snlua", "launcher");
if (ctx) {
    skynet_command(ctx, "REG", ".launcher");
    ctx = skynet_context_new("snlua", config->start);
}
```

即：`ctx = skynet_context_new("snlua", config->start);` 这行，意思就是使用 `snlua` 加载 `config->start` 这个服务。而 `config->start` 指向配置文件 `config` 的 `start = "main"` 这行。

## 2、调用服务

这个主要在 C 语言中实现，代码在 `skynet-src/skynet_server.c` 的 `skynet_context_new()` 函数中，这个函数主要就是实例化服务动态链接库中的 `"_create()"` 和 `"_init()"`，以及给服务创建一个私有的消息队列。并填充到 `struct skynet_context` 结构中。这个结构很重要。

`skynet_send()`，发送消息到队列中，等待调用服务的回调函数。

# 七、服务

## 1、用 C 语言写一个服务

用 C 语言编写一个服务，最简单最完整的就是 `service-src/server_logger.c` 这个服务。

里面包含了：

- 1、`logger_create()` 创建服务
- 2、`logger_init()` 初始化服务
- 3、`logger_release()` 释放服务
- 4、`_logger()` 回调函数，这个名字有 `skynet_callback(_logger);` 决定。

一个服务中的这四个函数就是 Skynet 服务动态链接库的 API。这里的 `logger_create()`，其中 `logger` 表示的是 `logger.so` 的名字，Skynet 的 `module` 会提取 `logger.so` 的名字作并加上“`_create`”来识别服务中的函数地址。函数的执行顺序是先执行“`_create()`”再执行“`_init()`”。而“`_release`”由 `skynet_context_release()` 调用来释放。而回调函数这是其他服务调用这个服务时会去调用它进行处理。服务的主要任务实现就在回调函数中处理。

## 2、用 Lua 语言写一个服务

用 Lua 语言写一个服务，最简单的例子在 `test/testsocket.lua` 这个服务，它主要实现了回射(echo)功能，客户端发送什么数据给服务端，服务端就像一面镜子把同样的数据反射回去给客户端。

`skynet.start()` 启动一个服务。