

1. shark 简介

shark 是一个高性能的协程网络组建，可以使用同步的方式编写高性能异步的网络程序。shark 适合需要使用 epoll 处理大量并发 tcp 连接的场景，性能与裸用 epoll 相当，代码复杂度大大降低。shark 在不降低性能的基础上，大大降低使用 C++ 编写高性能网络程序的难度，使用简单的几个接口就可以构建高性能的网络程序。

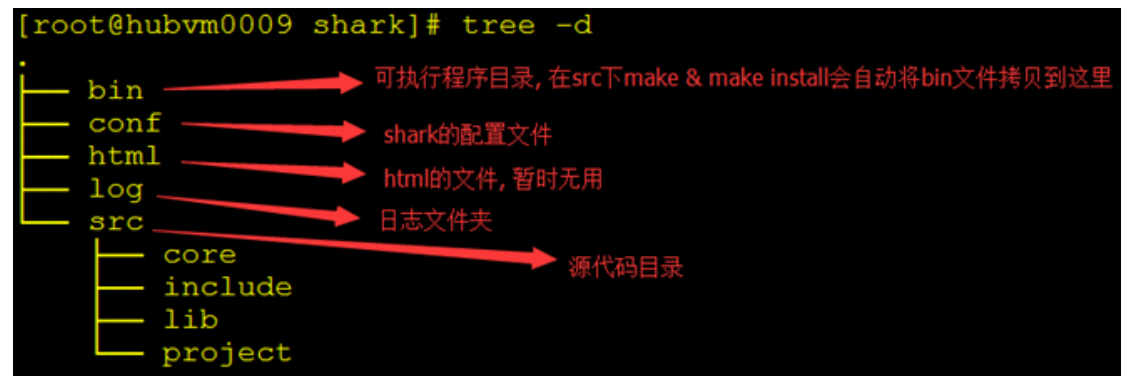
2. shark 特性

- 1) 同步无锁方式编写高性能异步代码
- 2) 支持多核，支持核心绑定
- 3) 支持多进程并行计算，支持进程间负载均衡
- 4) Hook 并优化阻塞的网络系统调用，让第三方组件同步变异步
- 5) 支持动态协程池，汇编实现的高性能协程上下文切换，每秒达 1.7 亿次
- 6) 高效的超时机制，支持千万级 timer
- 7) 高性能的缓存机制
- 8) 支持连接池，mysql

3. 目录说明

3.1 shark 目录

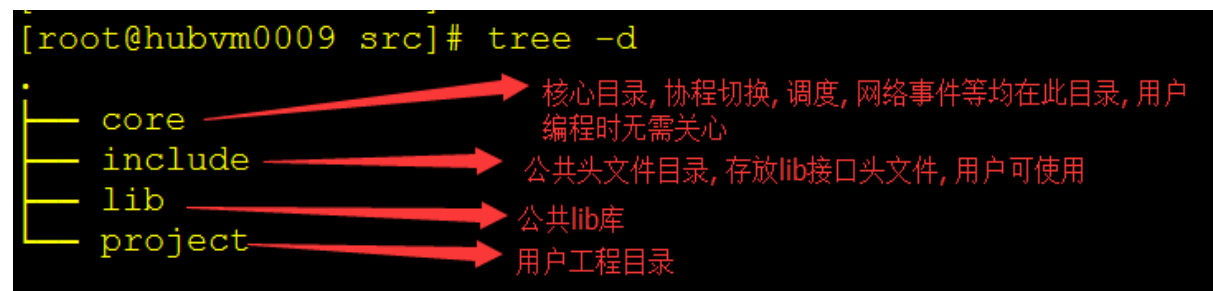
```
[root@hubvm0009 shark]# tree -d
```



- bin → 可执行程序目录, 在src下make & make install会自动将bin文件拷贝到这里
- conf → shark的配置文件
- html → html的文件, 暂时无用
- log → 日志文件夹
- src → 源代码目录
 - core
 - include
 - lib
 - project

3.2 源代码目录

```
[root@hubvm0009 src]# tree -d
```



- core → 核心目录, 协程切换, 调度, 网络事件等均在此目录, 用户编程时无需关心
- include → 公共头文件目录, 存放lib接口头文件, 用户可使用
- lib → 公共lib库
- project → 用户工程目录

3.3 core 目录

```
[root@hubvm0009 core]# tree
.
├── coro_sched.c      ──────────> 协程调度模块
├── coro_sched.h
├── coro_switch.c     ──────────> 协程切换模块
├── coro_switch.h
├── env.c             ──────────> 系统环境模块
├── env.h
├── netevent.c        ──────────> 网络事件模块
├── netevent.h
├── process.c         ──────────> 进程管理模块
├── process.h
├── shark.c           ──────────> shark初始化模块
├── shark.h
├── sys_hook.c        ──────────> HOOK系统调用模块
├── sys_hook.h
├── sys_signal.c      ──────────> shark信号处理
└── sys_signal.h
```

3.4 lib 目录

```
[root@hubvm0009 lib]# tree
.
├── cmysql.c          ──────────> mysql支持模块
├── conf.c            ──────────> 配置模块
├── conn_pool.c       ──────────> 连接池模块
├── cqueue.c          ──────────> 环形队列模块
├── kref.c            ──────────> 引用计数
├── log.c             ──────────> 日志模块
├── memcache.c        ──────────> 内存缓存模块
├── net.c             ──────────> 网络基本操作
├── rbtree.c          ──────────> 红黑树
├── shm.c             ──────────> 进程间共享内存模块
├── spinlock.c        ──────────> 自旋锁
├── str.c             ──────────> 字符串操作
└── util.c           ──────────> 系统的小功能组件
```

4. 编程手册

4.1 API 接口

1) `int project_init()`

该接口会在 shark 启动时调用，如果你的程序需要一些初始化，比如申请内存，初始化环境等等，请在该接口中实现

2) `void handle_request(int fd)`

该接口用于处理一个已经建立好的新连接，尔后你需要调用 `recv` 收包，之后处理.

需要注意的是，在处理完后，你并不需要手动去 `close` 连接，直接返回即可，shark 已经统一打包好了

以上 2 个 API 在 `httpsvr.c` 中有样例，可参考

4.2 配置

`shark.conf` 配置文件位于 `shark/conf` 目录下，分为 2 类配置，系统类和用户类，目前只有系统类.

没有复杂的语法，shark 的配置使用极为简洁，语法类似 `c(c++)` 语言赋值，一行就是一个配置项，配置项左边为变量，右边为变量的值. 配置使用方式详见 `shark.conf`

下面详细介绍系统配置的意义，用于指导我们该如何配置，更好的发挥 shark 性能

4.2.1 系统配置

配置项	说明
<code>worker_processes</code>	worker 进程数，默认为 <code>auto</code> ，即 worker 个数和系统 <code>cpu</code> 核心数保持一致 配置建议：

	如果业务主要是网络 IO 型，建议为 default，如果是磁盘 IO 型，建议多配置一点，比如双倍核心数
worker_connections	每个 worker 最多能同时处理的连接，超过这个阈值，新的连接请求将会丢弃 配置建议： 如果是网络 IO 型，估摸下业务每秒能处理的连接，比如 QPS 为 3K，那就填写 3000。如果是磁盘 IO 型，这意味着每个连接处理的时间会比较长，系统的支撑能力会受限，建议少填写点。 最终系统能同时处理的连接为 <code>worker_connections * worker_processes</code>
coroutine_stacksize	协程栈大小，单位为 KB，系统会自动转为操作系统页对齐大小(X86 下系统页为 4KB)，建议默认为 4KB，mysql 情况下 8KB，如此“节省”的原因是因为 shark 的栈开销最终为 <code>worker_processes * worker_connections * coroutine_stacksize</code> 。在编写业务逻辑时，只要不要在函数里整 <code>char array[1024]</code> 这样的大局部变量，基本没问题，另外也不用担心溢出时，数据会被破坏掉，栈是有保护的，溢出时，worker 会 <code>segment fault</code> 而退出。当然开辟大空间的栈也是没问题的，权衡系统资源即可
log_path	日志路径，默认在 <code>shark/log</code> 下，建议使用默认的
log_level	日志级别。shark 支持 CRIT, ERR, WARN, INFO, DBG 五种级别日志，优先级依次降低，即如果配置为 ERR 级别，那么只有 CRIT, ERR 级别的日志会被打印
log_reserve_days	日志保留天数，shark 每天生成一个日志文件，文件名以日期为后缀，对于超过保留天数的日志，shark 会自动删除。如果你想保留最近 7 天的日志，那就填写 7
server_ip	TCP 服务器绑定的 ip，默认为 null，你也可以指定绑定地址，但大部分建议是默认
listen	TCP 服务器绑定端口

4.2.2 用户配置

用户配置和系统配置遵循一样的语法，追加到 `shark.conf` 的尾部即可，比如你配置的 mysql 如下：

```
mysql_svr = 10.10.159.18
mysql_usr = root
mysql_pwd = sandai
```

在你的工程里，使用它们极为简单，shark 只提供了一个 API: `get_conf`。任何时候调用它即可得到你的配置数据

```
char *mqsvr = get_conf("mysql_svr");  
printf("mysql server name:%s\n", mqsvr);
```

配置实现和使用方式可以参考 conf.c(h)

4.3 日志

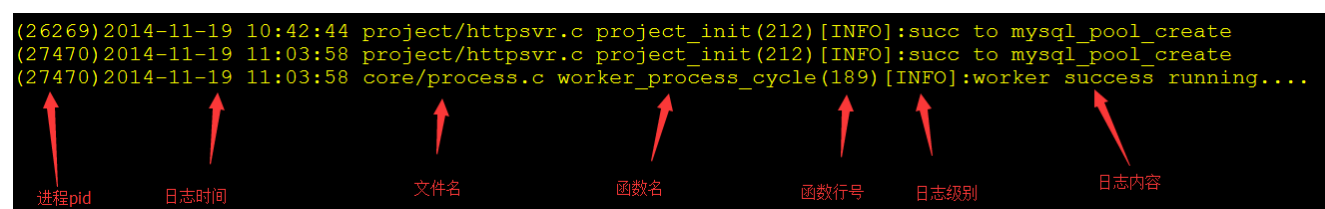
日志使用也非常简洁，没有人喜欢复杂，不要让用户思考和学习，而让用户凭直觉就能作出正确的选择，插个题外话，继续...

日志被设计成无锁的且非阻塞模型，专为多进程以及协程环境设计，测试性能极佳。日志的使用方式和 printf 并无差异，在任何时刻任何位置调用

5 类级别接口即可: CRIT, ERR, WARN, INFO, DBG, 函数名称就是日志级别，此外再无其他使用条件或者参数。比如:

```
INFO("succ to mysql_pool_create");  
DBG("Received connection: %d", fd);
```

打印的日志格式说明如下:



```
(26269)2014-11-19 10:42:44 project/httpsvr.c project_init(212)[INFO]:succ to mysql_pool_create  
(27470)2014-11-19 11:03:58 project/httpsvr.c project_init(212)[INFO]:succ to mysql_pool_create  
(27470)2014-11-19 11:03:58 core/process.c worker_process_cycle(189)[INFO]:worker success running....
```

进程pid 日志时间 文件名 函数名 函数行号 日志级别 日志内容

4.4 如何使用

使用方式和 nginx 类似，详细使用方式见下图，另外 shark 支持热部署，使用 ./shark -s stop 即可

```
[root@hubvm0009 bin]# ./shark ?  
Usage: shark [-?hvt] [-s signal]
```

Options:

```
-?,-h      : this help  
-v         : show version and exit  
-t         : print running shark config and exit  
-s signal  : send signal to a master process: stop, quit, reopen, reload  
              stop: stop accept new connection and wait all conneciont to be handled  
              quit: direct exit, do not wait all connection handled  
              reopen: reopen log file  
              reload: reboot shark with all connection handled
```