author：码农 Mark

本项目出自

《Linuxc/c/++高级全栈开发》职场就业提升课程内容

《 LinuxC/ C++高级全栈开发》 付费课程全新大纲学习内容：
https://www.0voice.com/uiwebsite/html/courses/v16.5.html

✔技术原理+✔源码分析+✔案例分析+✔项目实战

✔学习计划+✔技术答疑+✔简历指导+✔面试指导

本课程掌握80%的内容， 保证拿到就业offer!
●面试常问技术点， 工作常用技术点
●13大项目实战， 可写入简历， 运用到工作中去

除了课程技能学习， Mark全程陪跑就业

服务包含：

①： 面试必备技术栈的提升
②： 工作常用技能深入学习
③： 企业级项目实战训练， 可写入简历
④： 老师一对一学习计划
⑤： 老师一对一技术答疑， 远程指导
⑥： 老师一对一简历优化修改指导
⑦： 老师一对一面试复盘， 职业规划

项目地址

项目代码： https://github.com/mark-0voice/zvnet

项目核心代码： https://github.com/mark-0voice/zvnet-core

> 项目核心代码说明：原项目实现功能较多，要想掌握需要花费大量时间，为此我将项目中最精髓的450 行代码剥离出来，单独作为一个项目，这样可以快速掌握项目。

项目简介

> 本项目实现了一个协程与事件驱动相融合的基础框架。抽象了一个业务解决模型即一条连接对应着一个协程，一种业务也可抽象出一个协程。
>
> 项目适合**方向**：游戏开发（90%），后端开发（40%），网关开发（70%）。
>
> 项目适合**人群**：应届生（60%），社招生游戏方向（90%）。
>
> 项目依赖**技术栈**：*reactor/proactor* 网络编程、*lua/luajit* 以及 *lua/c* 接口编程、定时器设计、内存池、线程池、连接池、*redis*、*MySQL*、*http/https*、*makefile* 等。

安装说明

依赖

ubuntu

```
apt-get install autoconf
```

centos

```
yum install -y autoconf
```

macos

```
brew install autoconf
```

安装编译

centos、ubuntu

```
make linux
```

macos

```
make macosx
```

使用说明

```
# 开启服务端
 ./zvnet example/echo-srv.lua
# 另起一个终端：开启客户端
 ./zvnet example/echo-cli.lua
```

项目依赖《 LinuxC/ C++高级全栈开发》课程技术栈

> *reactor/proactor* 网络编程、 *lua/luajit* 以及 *lua/c* 接口编程、定时器设计、 内存池、线程池、 连接池、 *redis*、 *MySQL*、*http/https*、 *makefile* 等。

makefile

reactor/proactor 网络编程

池式结构


3.1 池式组件
3.1.1 手写线程池与性能分析（项目）
线程池的异步处理使用场景
线程池的组成 任务队列 执行队列
任务回调与条件等待
线程池的动态防缩
扩展：nginx线程池实现对比分析
3.1.2 内存池的实现与场景分析（项目）
内存池的应用场景与性能分析
内存小块分配与管理
内存大块分配与管理
手写内存池，结构体封装与API实现
避免内存泄漏的两种万能方法
定位内存泄漏的3种工具
扩展：nginx内存池实现
3.1.3 mysql连接池的实现（项目）
连接池性能的影响的2个因素，tcp连接和mysql认证
连接请求归还策略
连接超时未归还策略
链接断开重连策略
连接数量最优策略

高性能组件

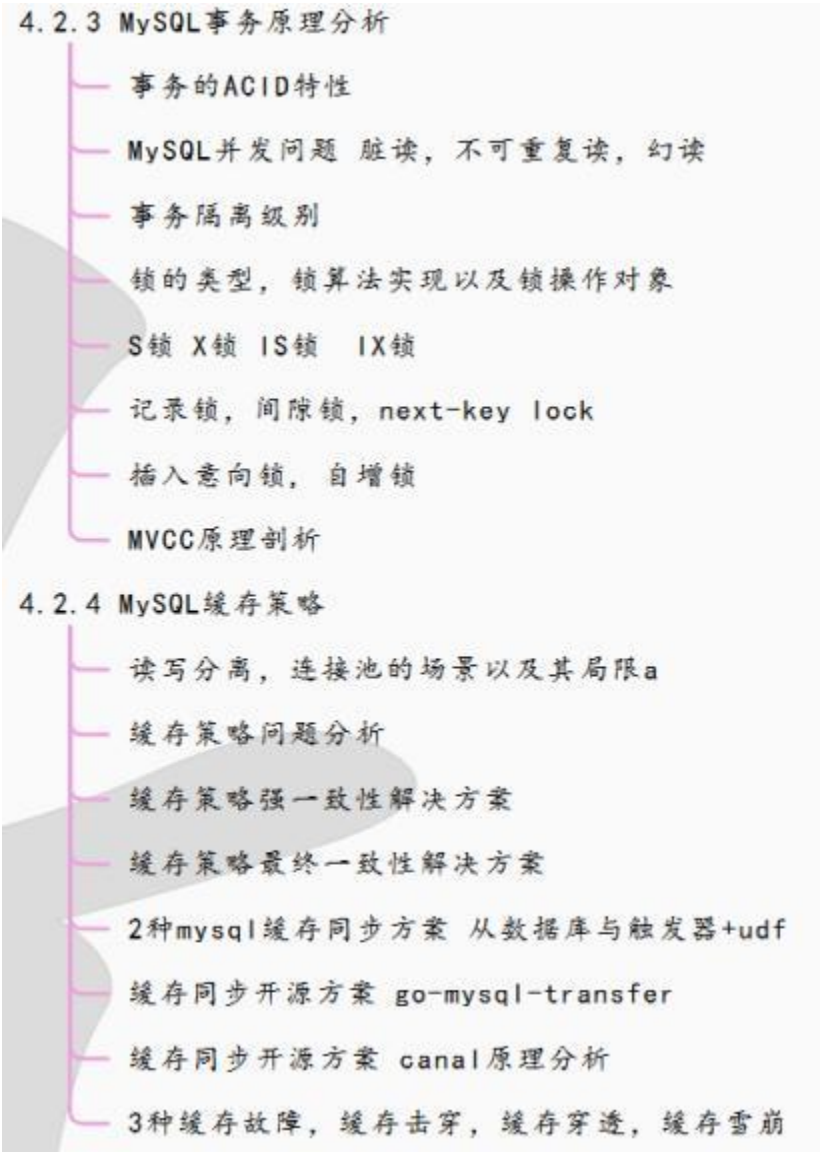

3.2.3 网络缓冲区设计
1. RingBuffer设计
2. 定长消息包
3. ChainBuffer设计
4. 双缓冲区设计
3.2.4 定时器方案红黑树，时间轮，最小堆（项目）
定时器的使用场景
定时器的红黑树存储
时间轮的实现
最小堆的实现
分布式定时器的实现

Redis

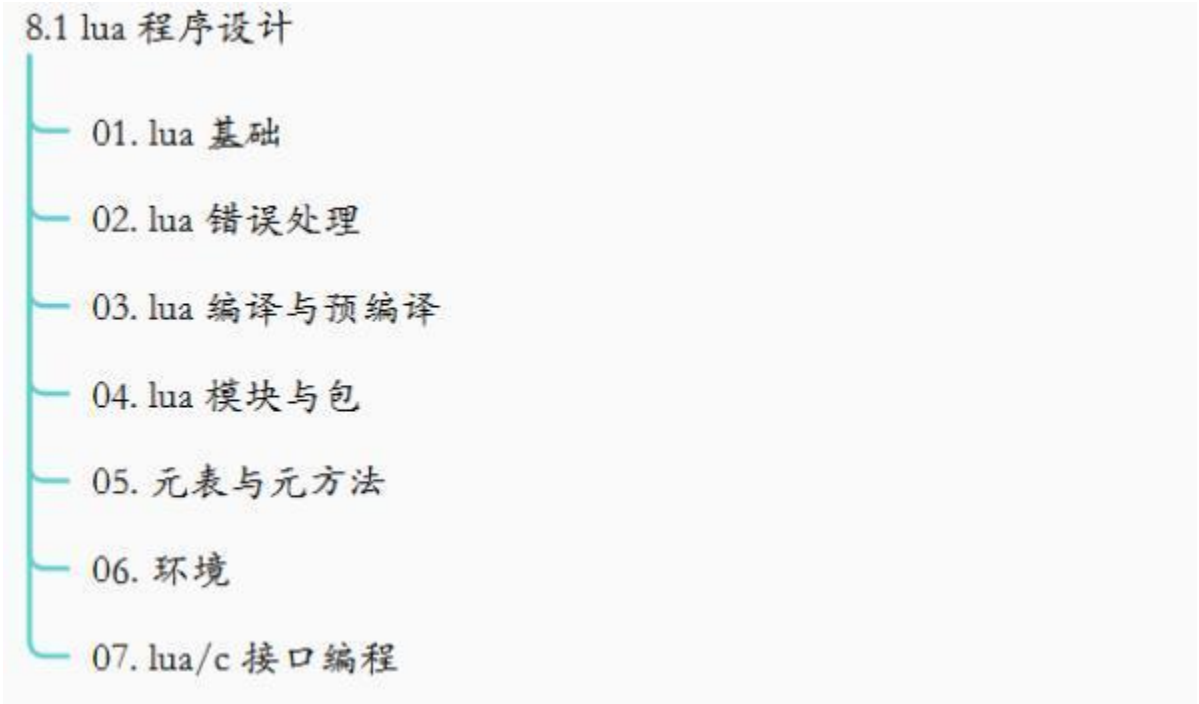

4.1 Redis
4.1.1 Redis相关命令详解及其原理
string, set, zset, list, hash
分布式锁的实现
lua脚本解决ACID原子性
Redis事务的ACID性质分析
4.1.2 Redis协议与异步方式
Redis协议解析
特殊协议操作 订阅发布
手撕异步redis协议
4.1.3 存储原理与数据模型
string的三种编码方式 int,raw,embstr
双向链表的list实现
字典的实现, hash函数
解决键冲突与rehash
跳表的实现与数据论证
整数集合实现
压缩列表原理证明

MySQL

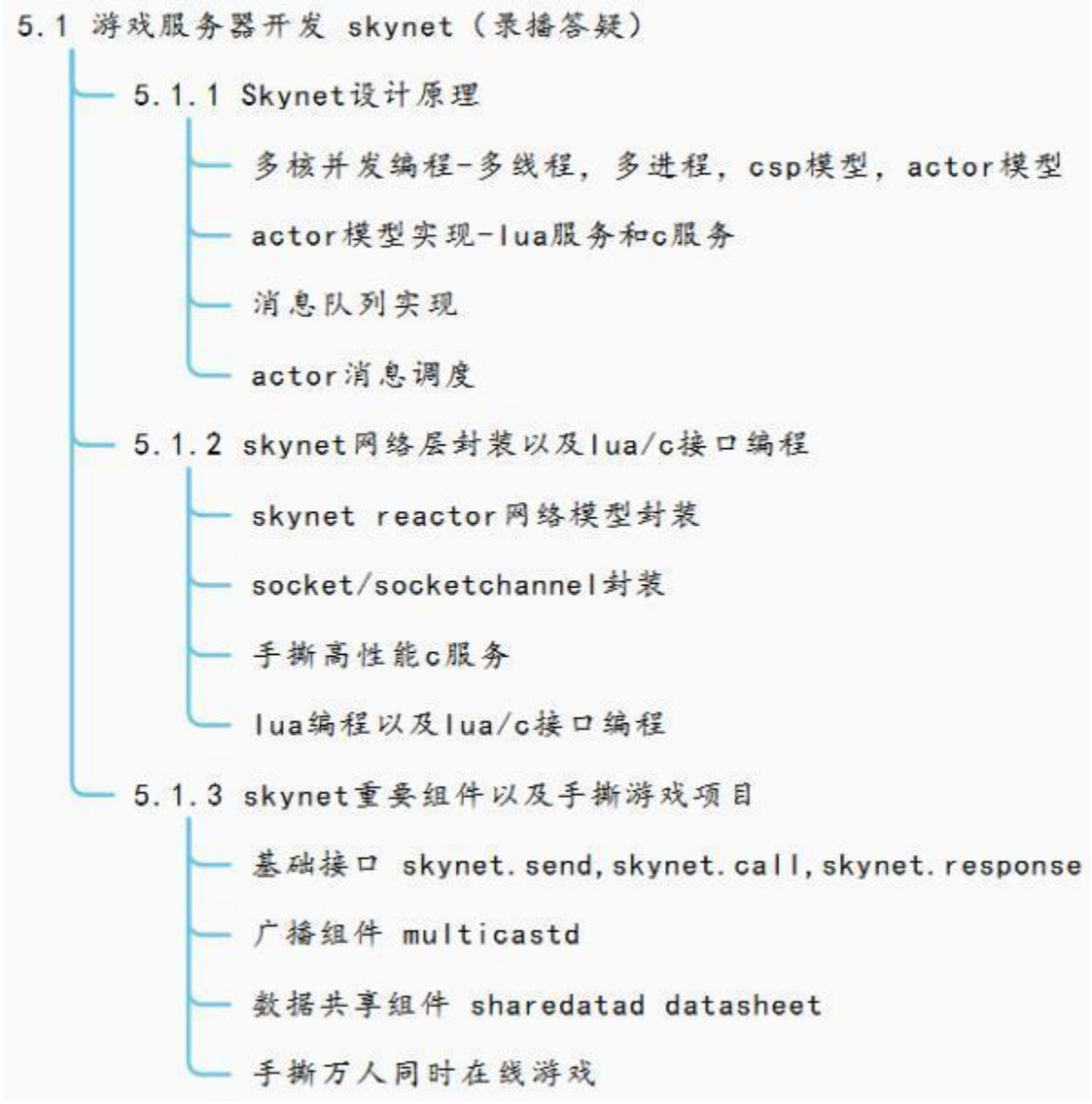

4.2.1 SQL语句，索引，视图，存储过程，触发器
MySQL体系结构，SQL执行流程
SQL CURD与高级查询
视图，触发器，存储过程
MySQL权限管理
4.2.2 MySQL索引原理以及SQL优化
索引，约束以及之间的区别
B+树，聚集索引和辅助索引
最左匹配原则以及覆盖索引
索引失效以及索引优化原则
EXPLAIN执行计划以及优化选择过程分析


4.2.3 MySQL事务原理分析
事务的ACID特性
MySQL并发问题 脏读，不可重复读，幻读
事务隔离级别
锁的类型，锁算法实现以及锁操作对象
S锁 X锁 IS锁 IX锁
记录锁，间隙锁，next-key lock
插入意向锁，自增锁
MVCC原理剖析
4.2.4 MySQL缓存策略
读写分离，连接池的场景以及其局限a
缓存策略问题分析
缓存策略强一致性解决方案
缓存策略最终一致性解决方案
2种mysql缓存同步方案 从数据库与触发器+udf
缓存同步开源方案 go-mysql-transfer
缓存同步开源方案 canal原理分析
3种缓存故障，缓存击穿，缓存穿透，缓存雪崩

lua

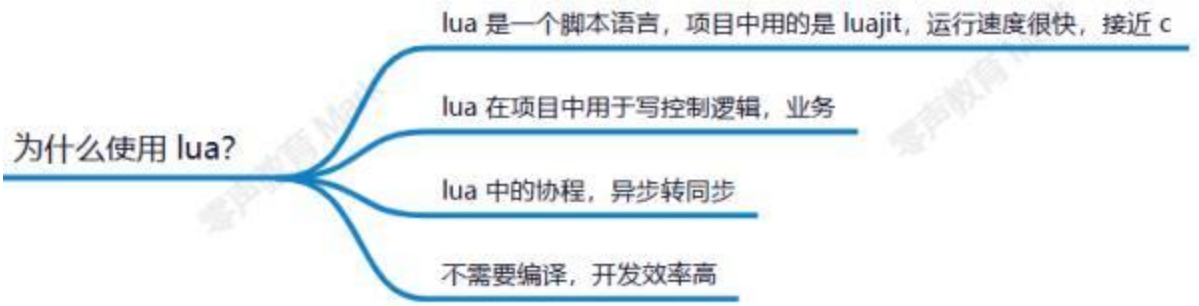

8.1 lua 程序设计
01. lua 基础
02. lua 错误处理
03. lua 编译与预编译
04. lua 模块与包
05. 元表与元方法
06. 环境
07. lua/c 接口编程

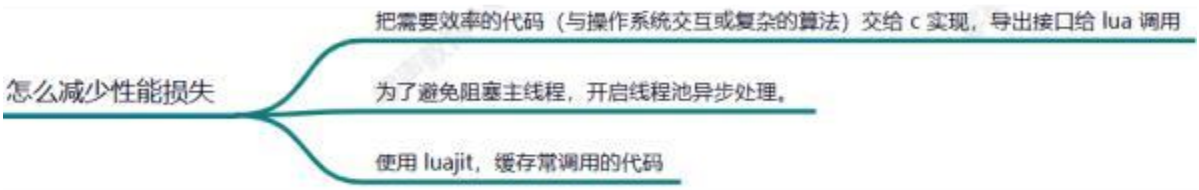
skynet



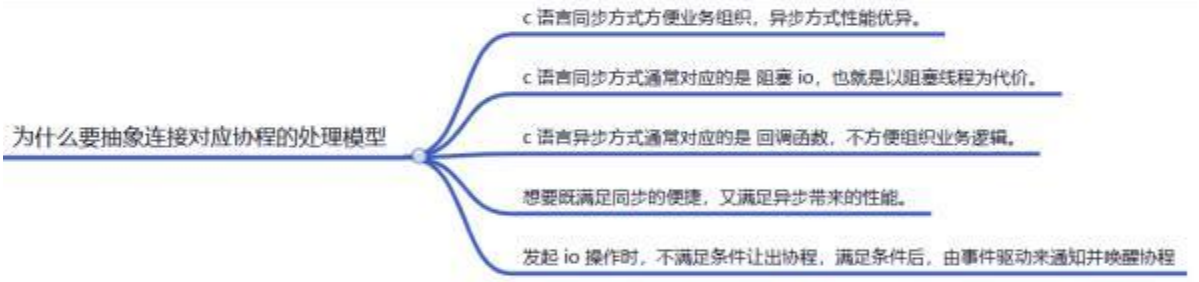
openresty

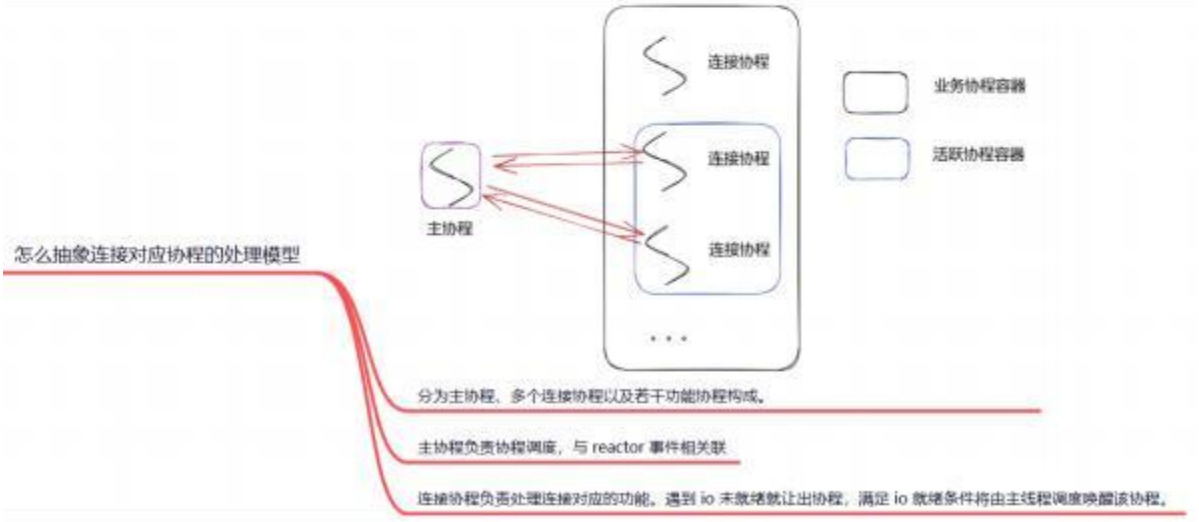
可能的面试问题


为什么使用 lua?
lua 是一个脚本语言，项目中用的是 luajit，运行速度很快，接近 c
lua 在项目中用于写控制逻辑，业务
lua 中的协程，异步转同步
不需要编译，开发效率高

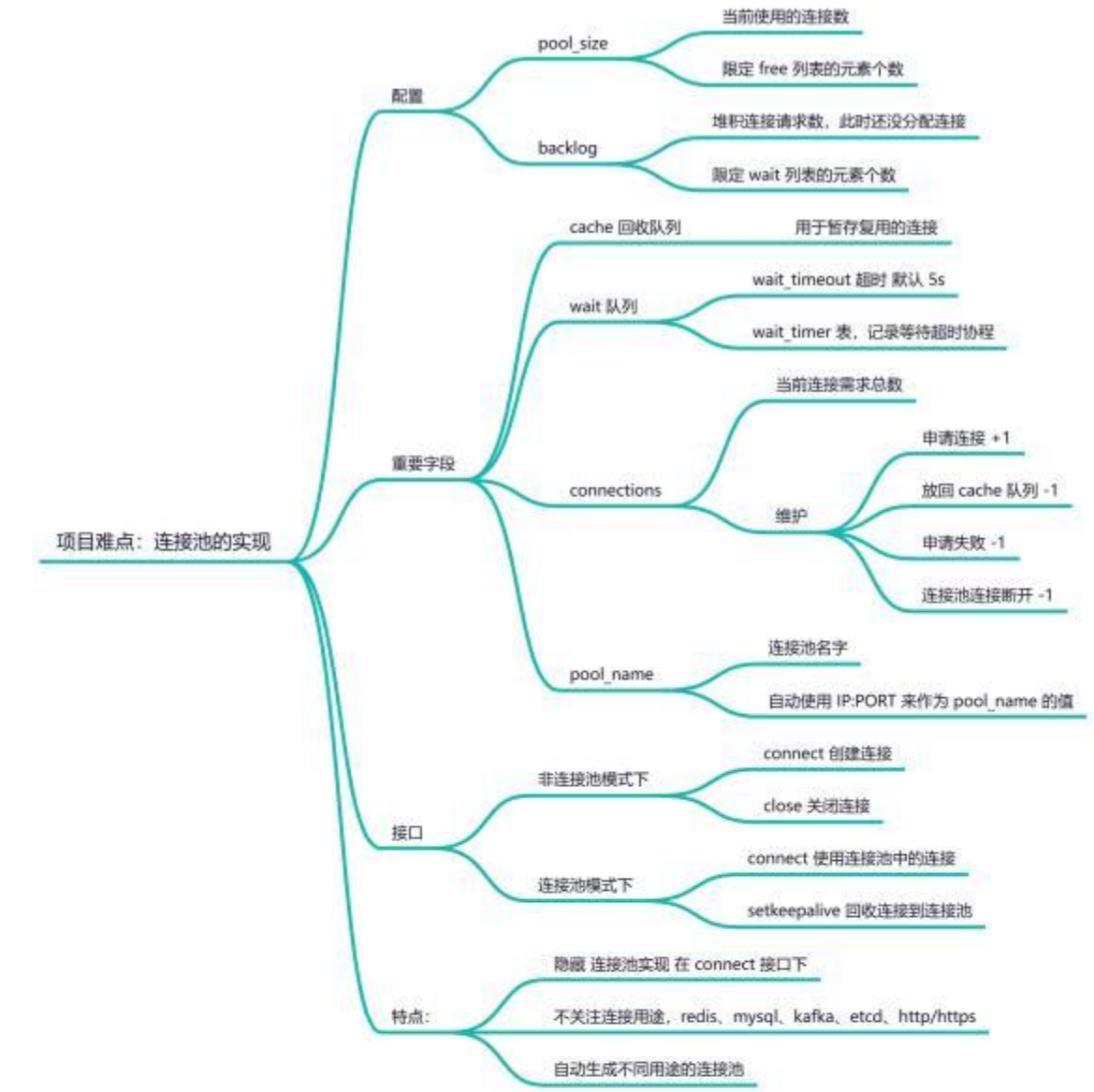

怎么减少性能损失
把需要效率的代码（与操作系统交互或复杂的算法）交给 c 实现，导出接口给 lua 调用
为了避免阻塞主线程，开启线程池异步处理。
使用 luajit，缓存常调用的代码


项目中水平触发和边缘触发
水平触发和边缘触发是 io 多路复用的触发方式
io 多路复用包含 select、poll、epoll
select、poll 只有水平触发
epoll 既有水平触发，又有边缘触发，默认是水平触发
水平触发：只要满足 io 就绪条件就会触发
边缘触发：新的 io 就绪事件到来只会触发一次
水平触发通常用于业务处理模型，取出一个数据包就开展业务；
边缘触发通常用于高性能处理模型，如代理服务，尽量将接收的数据转发给后端；
epoll 为什么水平触发，可以采用阻塞 io，边缘触发不可以采用阻塞 io
如果采用边缘触发，在接收数据处理的时候，必须要把所有数据读取出来
当读取结束后，继续读的时候，会阻塞线程，所以不可以采用阻塞 io

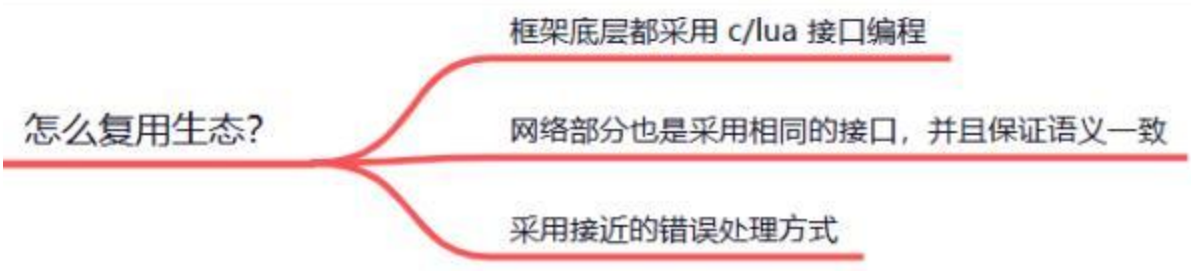

为什么要抽象连接对应协程的处理模型
c 语言同步方式方便业务组织，异步方式性能优异。
c 语言同步方式通常对应的是 阻塞 io，也就是以阻塞线程为代价。
c 语言异步方式通常对应的是 回调函数，不方便组织业务逻辑。
想要既满足同步的便捷，又满足异步带来的性能。
发起 io 操作时，不满足条件让出协程，满足条件后，由事件驱动来通知并唤醒协程


连接协程
连接协程
连接协程
主协程
业务协程容器
活跃协程容器
怎么抽象连接对应协程的处理模型
分为主协程、多个连接协程以及若干功能协程构成。
主协程负责协程调度，与 reactor 事件相关联
连接协程负责处理连接对应的功能。遇到 io 未就绪就让出协程，满足 io 就绪条件将由主线程调度唤醒该协程。


行业内是怎么做的
skynet
openresty


项目难点：连接池的实现
配置
pool_size
当前使用的连接数
限定 free 列表的元素个数
backlog
堆积连接请求数，此时还没分配连接
限定 wait 列表的元素个数
重要字段
cache 回收队列
用于暂存复用的连接
wait 队列
wait_timeout 超时 默认 5s
wait_timer 表，记录等待超时协程
connections
当前连接需求总数
维护
申请连接 +1
放回 cache 队列 -1
申请失败 -1
连接池连接断开 -1
pool_name
连接池名字
自动使用 IP:PORT 来作为 pool_name 的值
接口
非连接池模式下
connect 创建连接
close 关闭连接
连接池模式下
connect 使用连接池中的连接
setkeepalive 回收连接到连接池
特点：
隐藏 连接池实现 在 connect 接口下
不关注连接用途，redis、mysql、kafka、etcd、http/https
自动生成不同用途的连接池


项目中定时器如何实现
定时器是处理大量定时任务的基础模块
采用最小堆来组织所有定时任务
通过 epoll_wait 超时参数来检测最近要触发的任务
为每个触发的任务开启一个新的协程来执行任务
特色
封装 fork 接口用于执行异步任务
封装 sleep 接口阻塞当前协程若干个时间单位


怎么复用生态？
框架底层都采用 c/lua 接口编程
网络部分也是采用相同的接口，并且保证语义一致
采用接近的错误处理方式