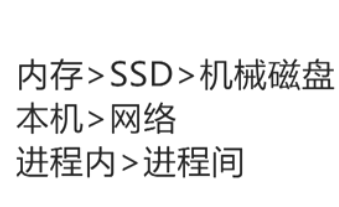
Openresty学习，参考网易云课堂

url: <http://study.163.com/course/courseLearn.htm?courseId=1520005#/learn/video?lessonId=1813061&courseId=1520005>

搭建服务端考虑量大要素：

1. 缓存
2. 语言支持异步非阻塞

缓存：



异步非阻塞

Openresty颠覆了高性能服务端的开发模式

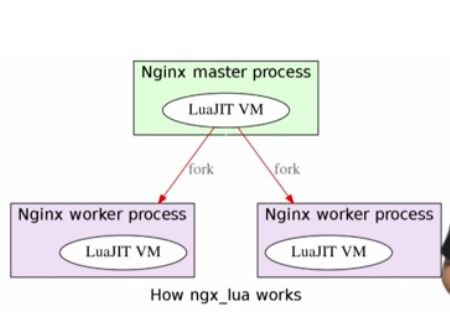
## Nginx：

不止负载均衡和反向代理

Nginx的c module是性能之王，每天承载量是几百亿

Lua是小巧的嵌入式脚本语言，网游等游戏里面用的较多

Openresty用的是luaJIT，结合nginx来做服务端



将luajit的虚拟机嵌入到nginx的worker里面，每次请求来的时候就会被发到nginx里的worker来进行计算



和市面上的解决方案比较：

Node.js:缺点，总是利用回调来处理异步非阻塞

Python：3.4版本加入了异步非阻塞的知识，3.5加入了协程；但是目前流行的还是2.7版本

Golang：没有热调试（不用重启就可发现是哪些问题）

SystemTap：热调试

BaaS:云



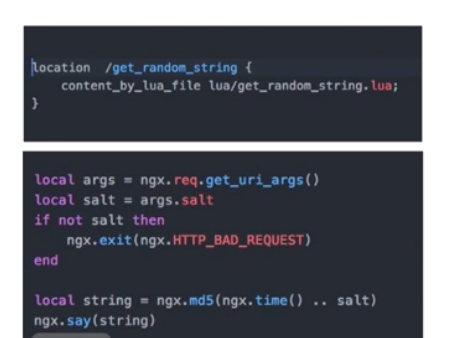
调试阶段可以关闭掉lua\_nginx的cache，但生产环境下最好开启



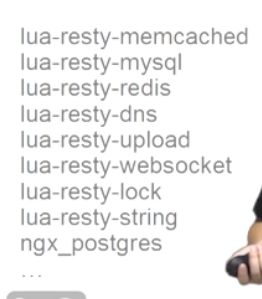
书籍推荐：openresty的最佳实践—自己编写



代码：



当前常见的模块列表（类php里面的扩展）：





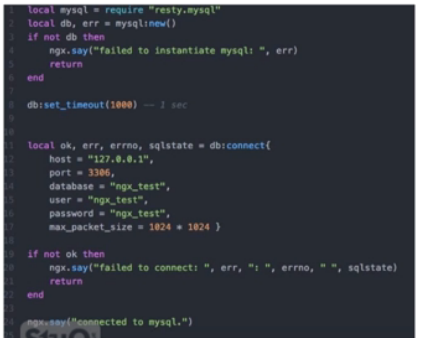
Ngx-lua的api完全是异步非阻塞的，但是lua自身的api可能是阻塞的

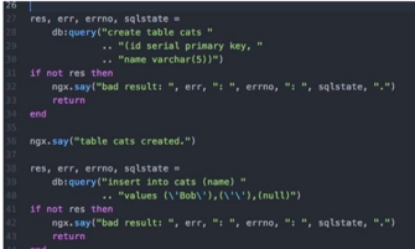
Nginx可以-s reload来重启

连接数据库



首先安装驱动/模块





Co-socker来实现数据库操作的异步非阻塞

操作Redis:



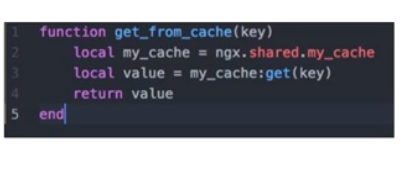
缓存：



方法一：

Shared\_dict即字典缓存，首先在nginx的conf文件中加入一条指令：lua\_shared\_dict my\_cache 128m;预设缓存大小

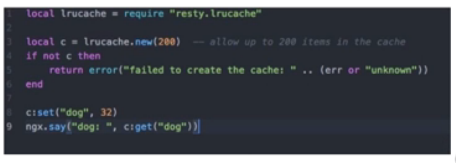
利用在lua代码中利用这个内存mycache



该缓存是纯内存的操作，是多个worker共享的；因此需要有锁以保证功能的原子性

方法二：

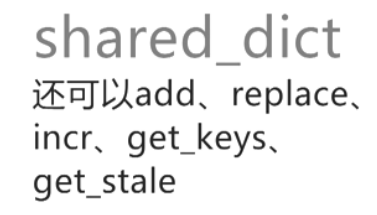






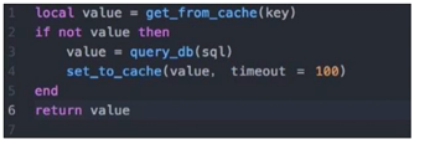
Lua-lrucache设置了cache的限制数量，没有设置限制大小，但是每个worker独占自己的内存数据，因此也就耗费了大量的数据





Ab测试redis





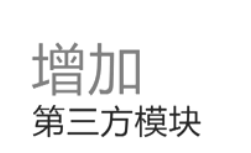


Ffi和第三方模块的使用



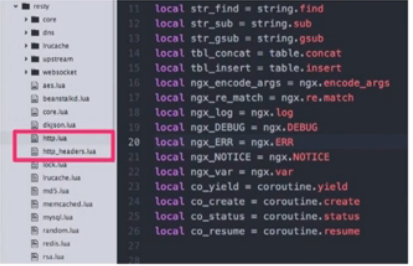
Ffi使得lua可以在luajit里面调用外部的c函数以及在lua文件中使用c数据结构

增加第三方模块:



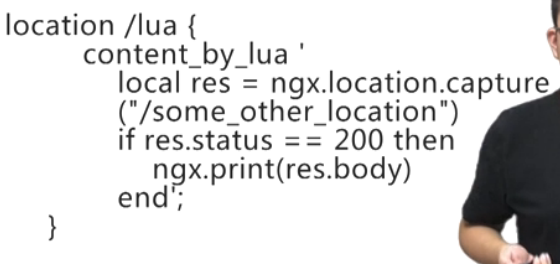
选择第三方：star多，更新频繁，贡献者多

下载第三方库,放在resty目录下:



在lua代码中直接require即可





从A location到B loacation，然后将结果返回，内部是c级别函数的调用，不是http请求，性能很高

如电商网站详情页面的推荐、其他用户还看、相同的商品等：

传统的做法是对每一个功能开发一个api，最后请求详情页面的时候在请求其他的api,这会在某些时候产生卡顿等问题，如访问量大时，因为要访问多个api



而且无法实现降级服务-即后天某个api down掉后前端请求会一直卡在请求阶段从而导致用户体验性不好

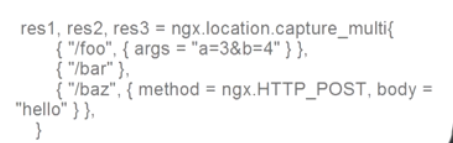


如京东商城的详情页：



然后使用子查询将location串联起来

Openresty可以同时开启多个子查询：



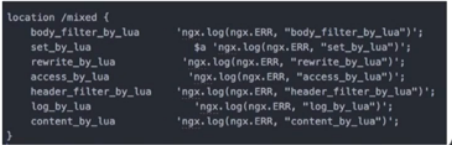
按照顺序来返回，耗费的总时间是最慢的处理所耗费的时间



Nginx官方请求流程：



Openresty的执行阶段和顺序：





主要做流程分支判断和变量初始化



主要做转发、重定向和缓存的功能



主要做一些ip准入、接口权限、合法性以及结合系统的iptables



内容生成阶段



http头信息处理



响应内容的一些处理，如变换大小写



日志功能，都是异步的操作

虽然我们可以在一个阶段完成所有的操作，但是推荐按照不同的阶段来执行对应的操作，这样有利于维护

