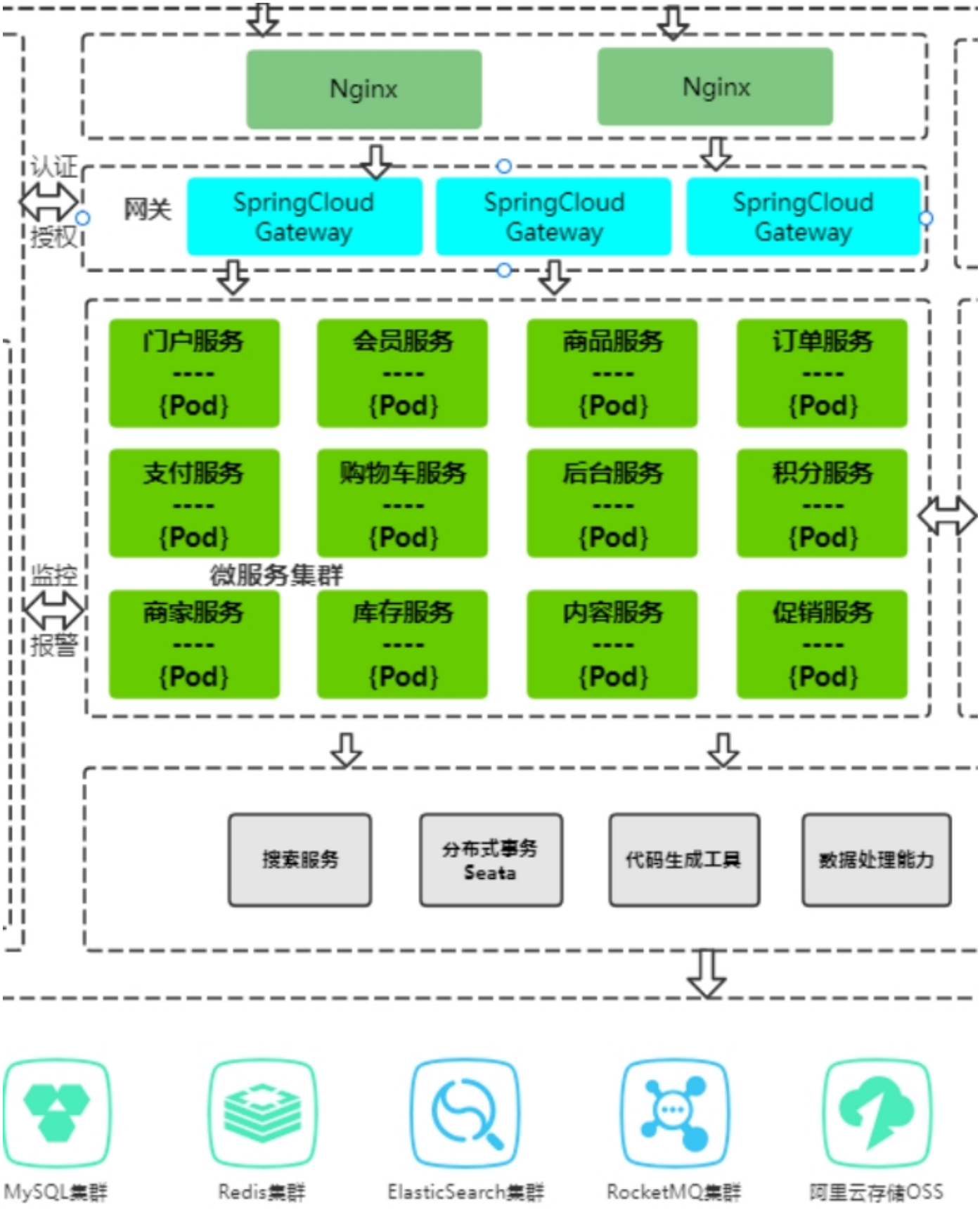
通用秒杀架构梳理

通用秒杀架构

系统的设计是个由巨入细的过程，想去设计好它，那你首先得去了解清楚它。所以这节课我们将重点分析传统架构设计的特点，接着介绍最新的秒杀系统架构，并做好技术选型和环境准备。

一般性系统架构

下面先看一个大家常用的系统功能架构图:



这种功能结构以及系统架构，是我们非常熟悉的。很多时候，Nginx只做反向代理和负载均衡，甚至这层对大部分做业务开发的研发人员来说，都是无感知的，一般运维部门在做生产环境搭建时，都会配好。研发人员更多的是在开发Web 服务和其他RPC服务/微服务，我们把页面以及页面所依赖的静态资源都放到Web 服务中，同时Web 服务还提供业务接口，RPC服务提供一些支撑服务。

当然，像我们商城进行动静分离后，VUE前端部分也会放在Nginx上，这就变成了页面以及页面所依赖的静态资源也在Nginx上，Web 服务提供业务接口，这种模式相比上面的有所改进。

对于秒杀来说瞬时流量非常大的情况，就会有很多问题，例如，以我们最为常见的商品详情页为例。

页面访问

商城进行了动静分离，商详页实现在product.vue中。可以看到，每个商品都会去后端获得商品的详细信息并展示。

```
mounted(){
  window.scroll( x: 0, y: 0);
  this.getProductInfo();
},
methods:{
  getProductInfo(){
    let id = this.$route.params.id;

    this.axios.get( url: `/pms/productInfo/${id}`).then((res)=>{
      this.product = res;
    });
```

可以想到，这种实现的商详页在秒杀高并发的情况下，不做任何措施，会对后端服务，特别是产品服务和数据库造成非常大的访问压力，即使产品信息全部缓存，依然会消耗大量的后端资源和带宽。

Web服务器性能问题

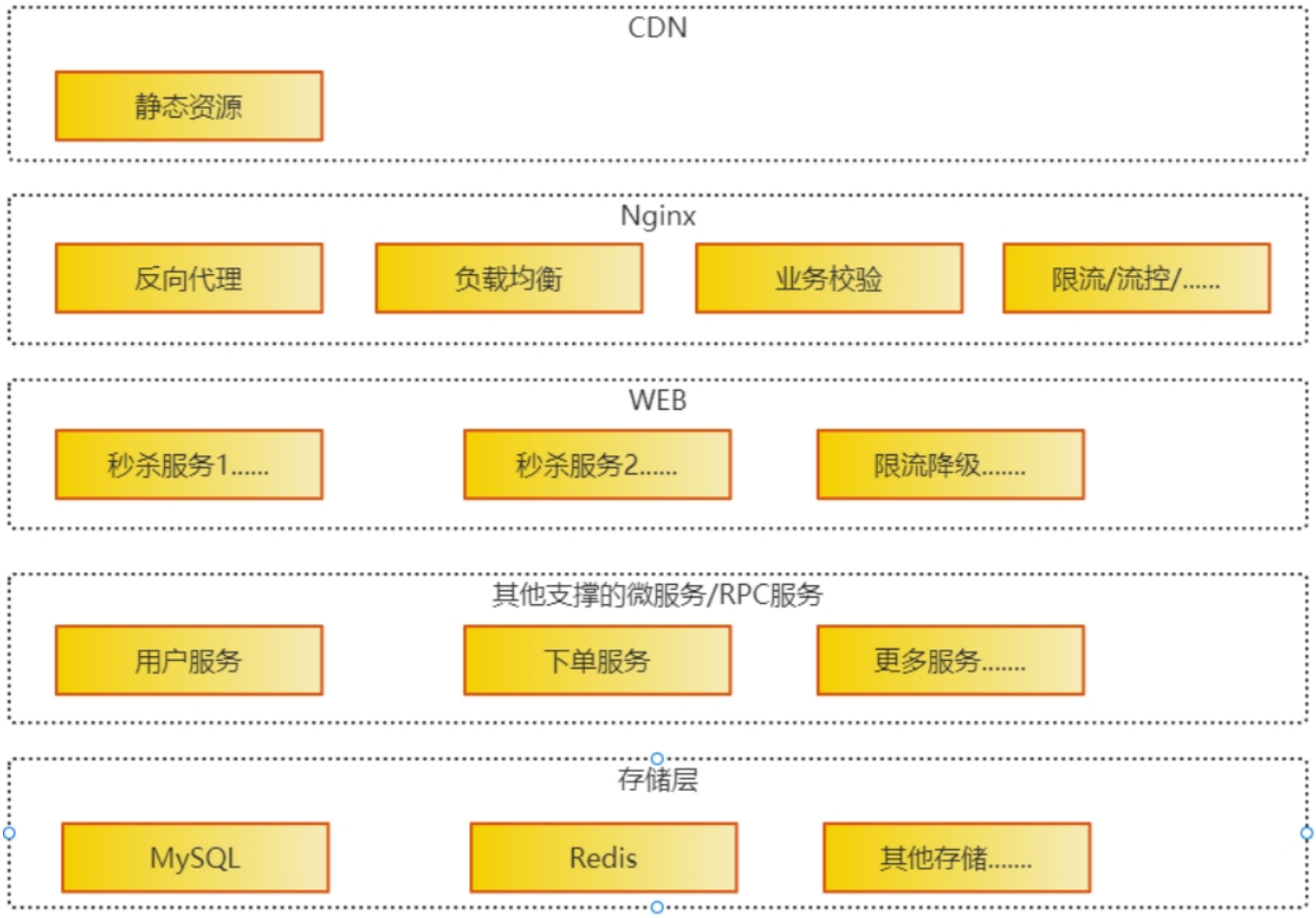
我们一般部署Web 服务，都是使用Tomcat 来部署的， Tomcat在处理请求的时候，是通过线程去处理的。

这样的问题就是如果瞬时的大量请求过来，线程池中的线程不够用， Tomcat就会瞬间新建很多线程，直至达到配置的最大线程数，如果线程数设置的过大，这个过程可能会直接将机器的CPU打满，导致机器死掉。即使没有挂掉，在高负载下，当设置的等待队列也满了之后，后面的请求都会被拒绝连接，直到有空出的资源去处理新请求。这时候你可能会想，我加机器分摊流量不就行了？可以是可以，但由此增加的活动成本有可能超出预算。

除此之外，还会伴有类似读写热点、库存超卖等等问题，这些我们会一一处理。

常见的秒杀系统架构

结合秒杀各链路层级，常见的大厂秒杀功能结构与系统架构图如下:



看起来似乎和一般的系统架构没什么区别，但是仔细研究区别还是很大的。一般情况下原先由Web 服务或Nginx服务提供的静态资源放到了CDN (CDN是全国都有的服务器，客户端可以根据所处位置自动就近从CDN上拉取静态资源，速度更快)，来大大减轻抢购瞬时秒杀域名的负担。

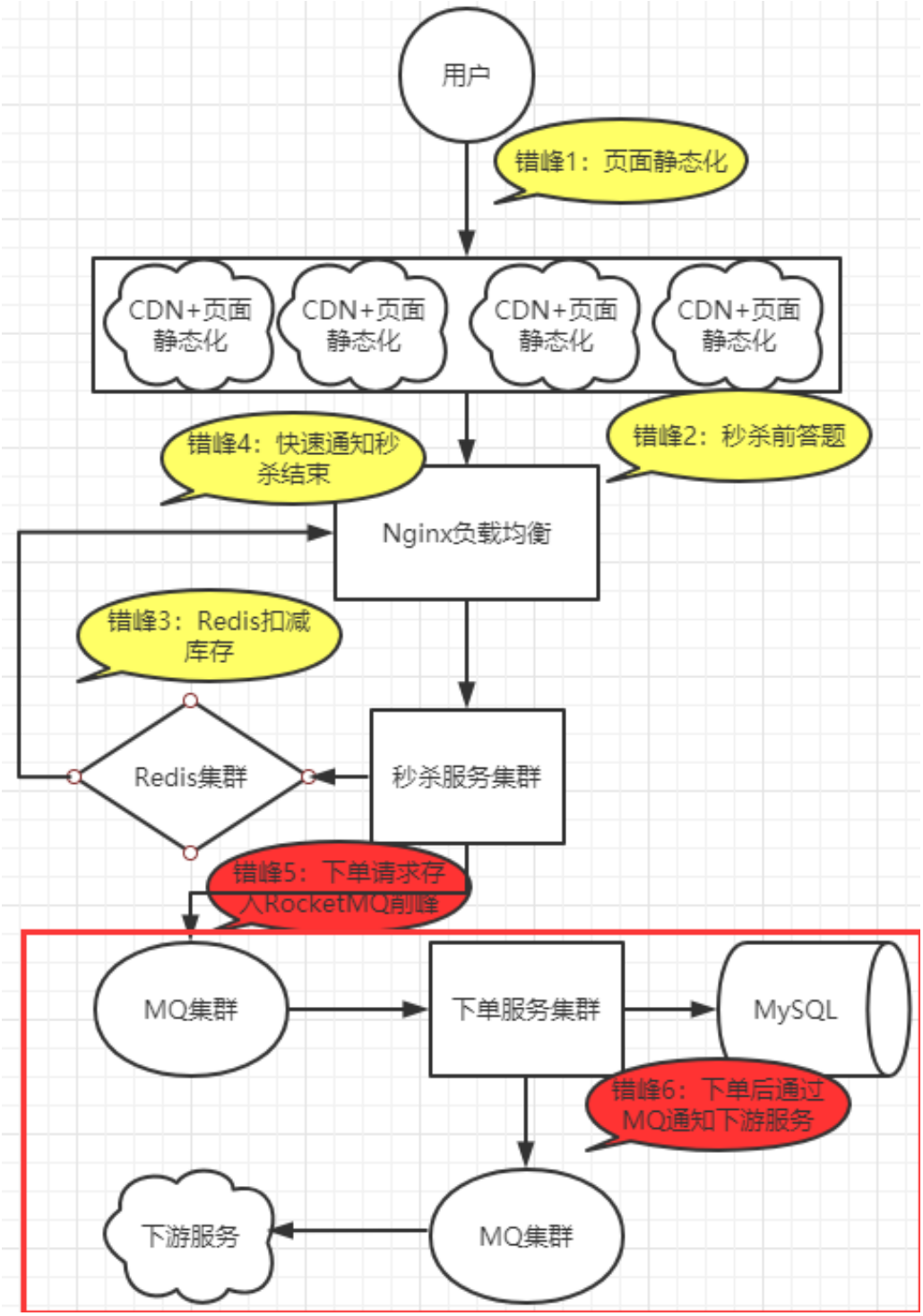
同时所做的最大改变，就是将Nginx的职责放大，前置用来做Web 网关，承担部分业务逻辑校验，并且可能增加黑白名单、限流和流控的功能，这其实也是根据秒杀业务特点所做的调整。这种在Nginx里写业务的做法在很多大公司里都是很常见的，像京东是用来做商详、秒杀的业务网关，美团用来做负载均衡接入层，12306用来做车票查询等等。

而这么做的目的，就是要充分利用Nginx的高并发、高吞吐能力，并且非常契合秒杀业务的特点，即入口流量大。但流量组成却非常的混杂，这些请求中，一部分是刷子请求，一部分是无效请求（传参等异常)，剩下的才是正常请求，一般情况下这个的比例可能是6:1: 3，所以需要在网关层尽可能多地接收流量进来，并做精确地筛选，将真正有效的3成请求分发到下游，剩余的7成拦截在网关层。不然把这些流量都打到Web服务层，Web服务再新起线程来处理刷子和无效请求，这是种资源的浪费。

所以网关层对秒杀系统而言，至关重要，而Nginx刚好可以胜任此项任务。所以Nginx在主要的秒杀系统设计中，扮演着非常重要的角色。

电商项目技术选型

在做具体技术选型时，我们会尽量选择大家比较熟悉的技术路线。电商项目整体的秒杀系统技术选型如下：



核心设计是对巨大的瞬间流量进行层层错峰。

错峰1：页面静态化。动静分离的本质是将包含浏览者信息的动态数据和不包含浏览者信息的静态资源区分开。例如在商品单品页，商品信息是不包含浏览者信息的，这部分就可以抽象出静态资源。而用户登录状态、cookie等这些动态数据也尽可能缓存起来，并且使缓存能够离用户更近。具体实现时，电商项目采用Freemarker模板引擎实现。

错峰2：秒杀前答题。秒杀答题的形式可以是多种多样的，目的是防止机器刷单，以及错开用户的下单时长。在秒杀场景下，答题速度靠后的请求自然就没有库存了，也可以减少系统的请求量。具体实现时，电商项目通过采用HappyCaptcha添加动态验证码来实现。

错峰3：Redis扣减库存。Redis缓存的作用主要有两个，一是快速扣减库存，保护数据库流量，并且库存扣减完成后，快速通知Nginx，屏蔽后续请求；二是提前识别热点数据，并且针对热点数据提供优化处理。处理的方案主要是三个，一是优化，二是限制，三是隔离，包括业务隔离、系统隔离、数据隔离。

错峰4：Nginx快速通知秒杀结束。当秒杀的商品抢购完成后，在网关层进行流量管控，直接拒绝后续的下单请求，从而保护后端服务。具体实现时，电商项目通过引入OpenResty提供对nginx的服务增强。

错峰 5：引入 MQ 进行流量削峰。通过 MQ 对前端并发请求进行削峰处理，从而减少瞬间流量对后端服务器造成的压力。

错峰 6：引入 MQ 进行下单服务异步化。后端服务完成下单业务后，只需要将下单消息发到 MQ ，而不用再关心其他下游业务。这样也能减轻后端下单服务的压力。

这个技术路线中，大部分技术我们都在 VIP 课程中进行过分享，唯一有些朋友比较陌生的是，在Nginx网关层，我们采用了OpenResty来进行构建。关于电商项目的具体实现，我们会在后面章节做具体介绍。这里先给大家简单介绍一下OpenResty。

OpenResty

简介

Nginx最早被发明出来，就是来应对互联网高速发展下，出现的并发几十方、上百万的网络请求连接场景的，传统Apache 服务器无法有效地解决这种问题，而Nginx却具有并发能力强、资源消耗低的特性。

总的来说，Nginx_有5大优点，即模块化、事件驱动、异步、非阻塞、多进程单线程。

前面我们说过，Nginx在主要的秒杀系统设计中，扮演着非常重要的角色，意味着Nginx上要承载很多的业务逻辑。Nginx的底层模块一般都是用C语言写的，如果我们想在Nginx的基础之上写业务逻辑会很不方便，所以这个时候我们还得借助OpenResty，它是Nginx的一个社区分支。OpenResty是中国人章亦春发起，最早是雅虎中国的一个公司项目，基于Perl和Haskell实现，2007年开始开源，后来章亦春大佬加入淘宝后进行了彻底的设计和重写。

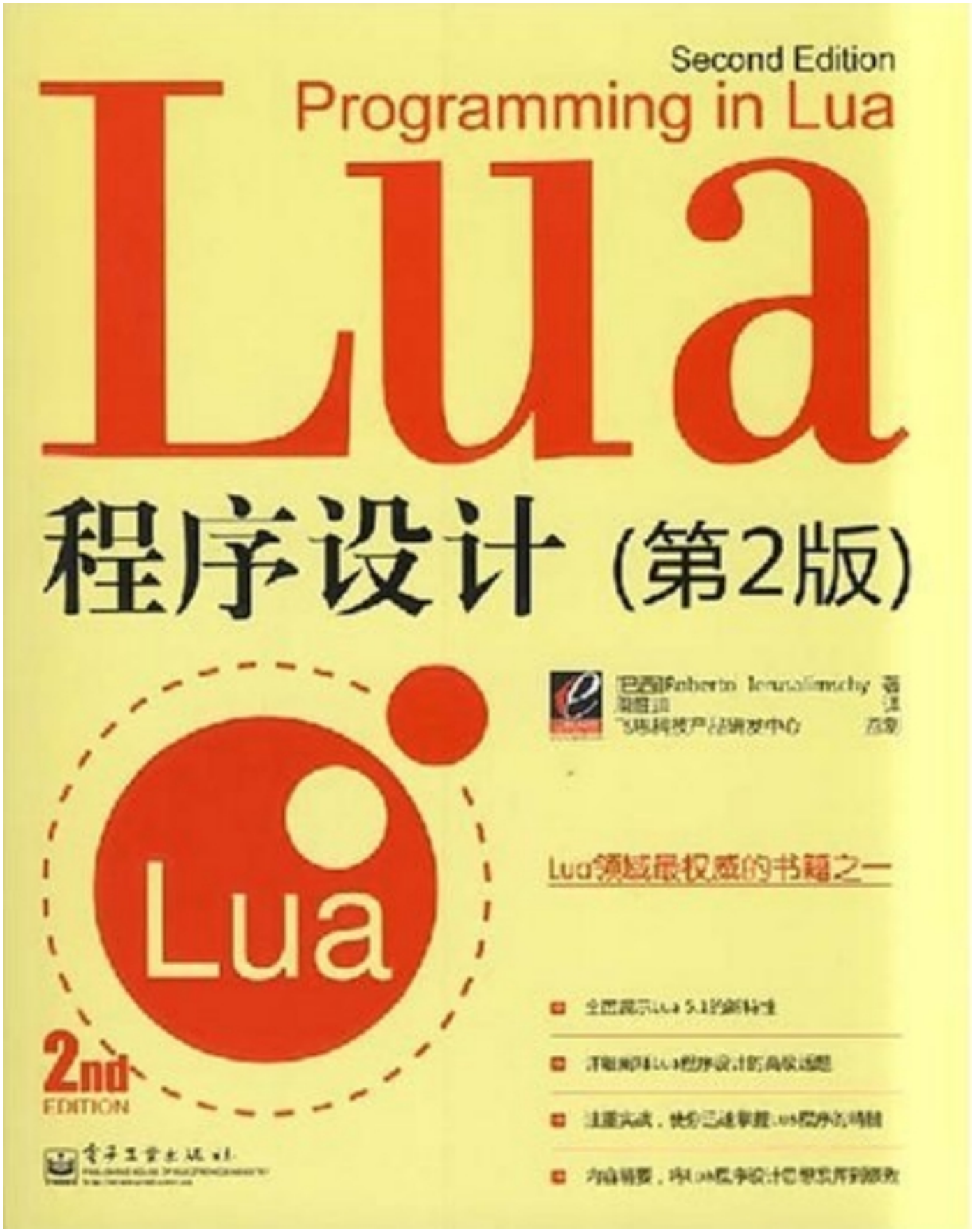
按照官网的说法，OpenResty 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

OpenResty 通过汇聚各种设计精良的 Nginx 模块（主要由 OpenResty 团队自主开发），从而将 Nginx 有效地变成一个强大的通用 Web 应用平台。这样，Web 开发人员和系统工程师可以使用 Lua 脚本语言调动 Nginx 支持的各种 C 以及 Lua 模块，快速构造出足以胜任 10K 乃至 1000K 以上单机并发连接的高性能 Web 应用系统。

为什么要用Lua语言来做Nginx开发呢?这就要说到Lua语言的特点了， Lua的线程模型是单线程多协程的模式，而Nginx刚好是单进程单线程，天生的完美搭档。同时Lua是一种小巧的脚本语言，语法非常的简单。所以在Redis中也是用Lua作为脚本语言的。Lua语言请自行学习，推荐两本豆瓣评分8分以上书籍


Broadview
Lua.org
Lua程序设计
Programming in Lua (fourth edition)
（第4版）
[巴西] Roberto Ierusalimschy 著
梅瑜魁 译
Lua
中国工信出版集团
电子工业出版社

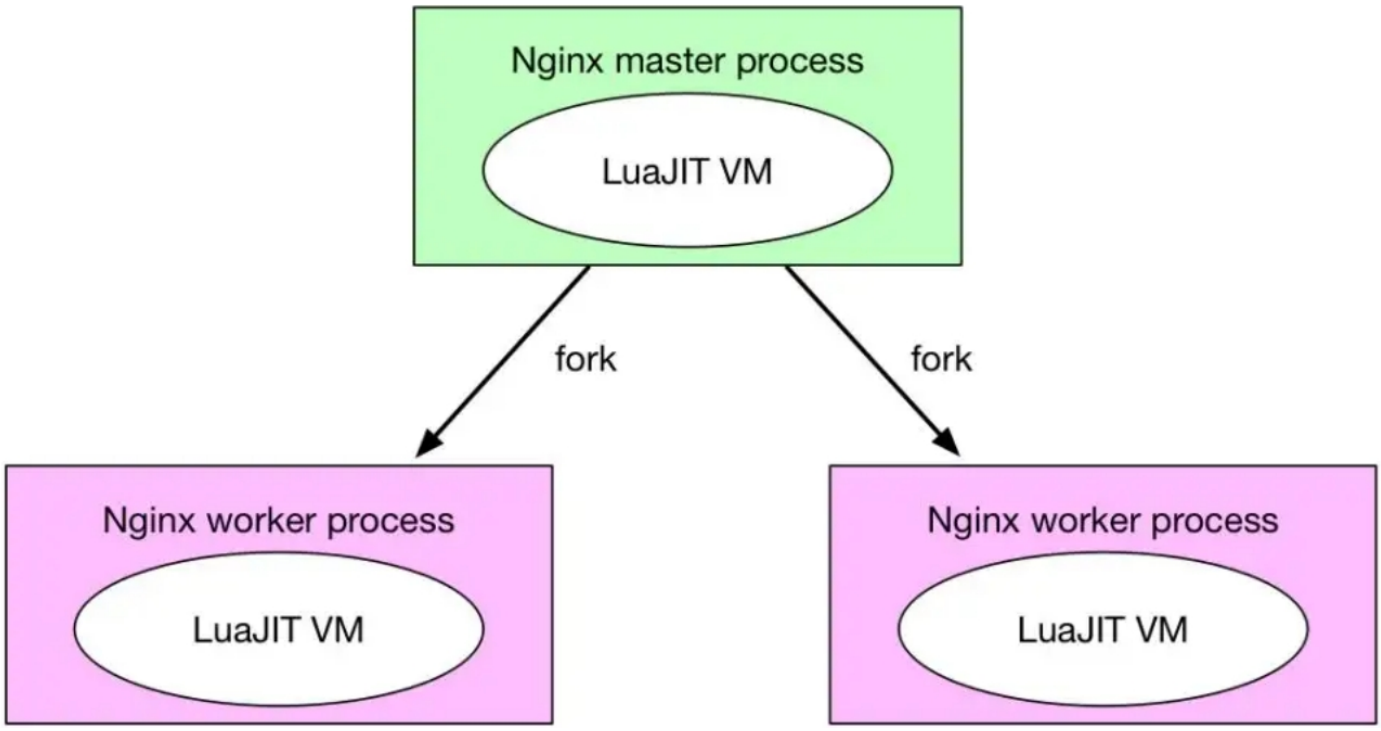
或



至于OpenResty的安装等知识请参考OpenResty中文官网：https://openresty.org/cn/

既然要使用OpenResty ， 我们还是要大概了解下Nginx和OpenResty 中的一些基本原理。

原理



Nginx 服务器启动后，产生一个 Master 进程（Master Process），Master 进程执行一系列工作后产生一个或者多个 Worker 进程（Worker Processes)。其中，Master 进程用于接收来自外界的信号，并向各 Worker 进程发送信号，同时监控 Worker 进程的工作状态。当 Worker 进程退出后(异常情况下)，Master 进程也会自动重新启动新的 Worker 进程。Worker 进程则是外部请求真正的处理者。

多个 Worker 进程之间是对等的，他们同等竞争来自客户端的请求，各进程互相之间是独立的。一个请求，只可能在一个 Worker 进程中处理，一个 Worker 进程不可能处理其它进程的请求。Worker 进程的个数是可以设置的，一般我们会设置与机器 CPU 核数一致。同时，Nginx 为了更好的利用多核特性，具有 CPU 绑定选项，我们可以将某一个进程绑定在某一个核上，这样就不会因为进程的切换带来cache的失效（CPU affinity）。所有的进程的都是单线程（即只有一个主线程）的，进程之间通信主要是通过共享内存机制实现的。

OpenResty本质上是将 LuaJIT 的虚拟机嵌入到 Nginx 的管理进程和工作进程中，同一个进程内的所有协程都会共享这个虚拟机，并在虚拟机中执行Lua代码。在性能上，OpenResty接近或超过 Nginx 的C模块，而且开发效率更高。

Nginx 将HTTP请求的处理过程划分为多个阶段。这样可以使一个HTTP请求的处理过程由很多模块参与处理，每个模块只专注于一个独立而简单的功能处理，可以使性能更好、更稳定，同时拥有更好的扩展性。

1）ngx_http_post_read_phase：

接收到完整的http头部后处理的阶段，它位于uri重写之前。

2）ngx_http_server_rewrite_phase：

uri与location匹配前，修改uri的阶段，用于重定向。

3）ngx_http_find_config_phase：

根据uri寻找匹配的location块配置项阶段，该阶段使用重写之后的uri来查找对应的location，值得注意的是该阶段可能会被执行多次，因为也可能有location级别的重写指令。

4）ngx_http_rewrite_phase：

上一阶段找到location块后再修改uri，location级别的uri重写阶段，该阶段执行location基本的重写指令，也可能会被执行多次。

5）ngx_http_post_rewrite_phase：

防止重写url后导致的死循环，location级别重写的后一阶段，用来检查上阶段是否有uri重写，并根据结果跳转到合适的阶段。

6）ngx_http_preaccess_phase：

下一阶段之前的准备，访问权限控制的前一阶段，该阶段在权限控制阶段之前，一般也用于访问控制，比如限制访问频率，链接数等。

7）ngx_http_access_phase：

让http模块判断是否允许这个请求进入nginx服务器，访问权限控制阶段，比如基于ip黑白名单的权限控制，基于用户名密码的权限控制等。

标准模块 ngx_access、第三方模块 ngx_auth_request 以及第三方模块 ngx_lua 的 access_by_lua 指令就运行在这个阶段。

8）ngx_http_post_access_phase：

访问权限控制的后一阶段，该阶段根据权限控制阶段的执行结果进行相应处理。

9）ngx_http_try_files_phase：

为访问静态文件资源而设置，try_files指令的处理阶段，如果没有配置try_files指令，则该阶段被跳过。

10）ngx_http_content_phase：

处理http请求内容的阶段，大部分http模块介入这个阶段，内容生成阶段，该阶段产生响应，并发送到客户端。

Nginx 的 content 阶段是所有请求处理阶段中最为重要的一个，因为运行在这个阶段的配置指令一般都肩负着生成“内容”（content）并输出 HTTP 响应的使命。

11）ngx_http_log_phase：

log阶段处理，比如记录访问量/统计平均响应时间。log_by_lua

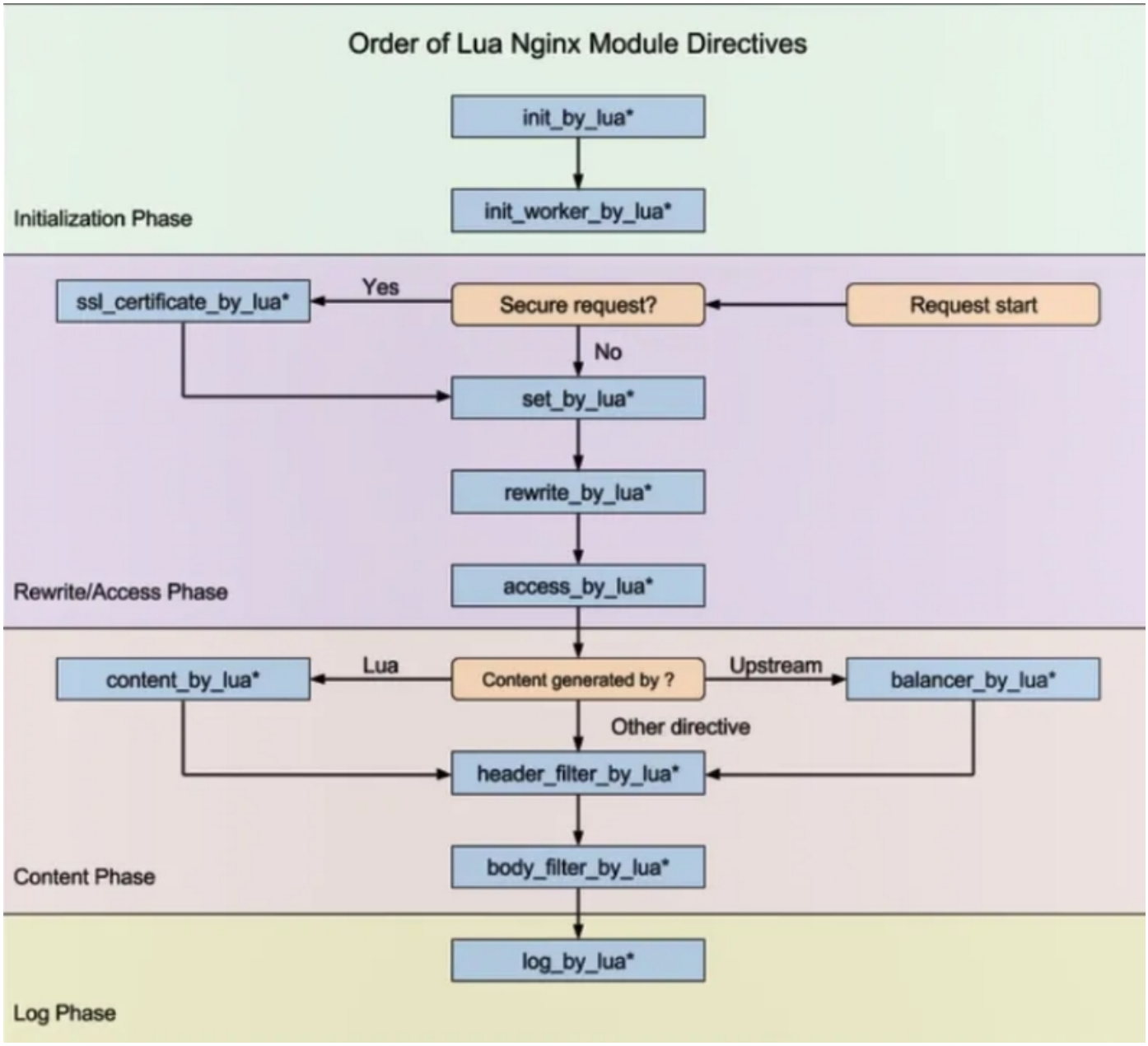
处理完请求后的日志记录阶段，该阶段记录访问日志。

以上11个阶段中，http无法介入的阶段有4个：

3）ngx_http_find_config_phase、5）ngx_http_post_rewrite_phase、8）ngx_http_post_access_phase、9）ngx_http_try_files_phase。

OpenResty在HTTP处理阶段基础上分别在Rewrite/Access阶段、Content阶段、Log阶段注册了自己的handler，加上系统初始阶段master的两个阶段，共11个阶段为Lua脚本提供处理介入的能力。

Nginx的 lua插载点如下



其中

init_by_lua：Master进程加载 Nginx 配置文件时运行，一般用来注册全局变量或者预加载Lua模块。

init_worker_by_lua：每个worker进程启动时执行，通常用于定时拉取配置/数据或者进行后端服务的健康检查。

set_by_lua：变量初始化。

rewrite_by_lua:可以实现复杂的转发、重定向逻辑。

access_by_lua:IP准入、接口权限等情况集中处理。

content_by_lua:内容处理器，接收请求处理并输出响应。

header_filter_by_lua:响应头部或者cookie处理。

body_filter_by_lua:对响应数据进行过滤，如截断或者替换。

log_by_lua:会话完成后，本地异步完成日志记录

> 有道云笔记链接：https://note.youdao.com/s/6Uej1qWF