ingress 简介

ingress

编译

- ingress-nginx 项目 https://github.com/kubernetes/ingress-nginx 包含多种工具
- images/ 目录内是基础和工具镜像编译
  - nginx：包含 nginx 及其patch (对 nginx 进行)，并下载常用模块，包括lua，brotli，modSecurity，mimalloc 等，注意这里默认用的 musl libc 编译
  - cfssl：编译 cloudflare 使用golang编写的证书管理工具
  - custom-error-pages：使用golang编写的重定义错误的容器
  - e2e-test-echo
  - ext-auth-example
  - fastcgi-server
  - grpc-server
  - httpbun: 带有端点的 HTTP 服务，用于测试任何 HTTP 客户端
  - kube-webhook-certgen：生成过期时间（100年）的CA和证书(secret)，然后进行对k8s webhook 资源进行patch以更新 ca，保证webhook请求有效
- rootfs/ 目录内是 ingress 镜像编译，提供基础 lua 库

国密

- 支持国密时，请使用 https://www.gmssl.cn/gmssl/index.jsp 并参考
- 通过配置 image/nginx 下的build.sh 加入国密的静态库并编译时，提示以下错误，原因是使用 musl 库，缺少对应的符号，因此最终修改为使用glibc，具体步骤见 https://github.com/yylt/build/blob/master/ingress/build-0.49.sh

```
// 错误日志
/usr/lib/gcc/x86_64-alpine-linux-musl/10.3.1/../../../../x86_64-alpine-linux-musl/bin/ld: /usr/local/gmssl/lib/libcrypto.a(async.o): in function `async_fibre_swapcontext':
/root/gm/openssl-1.1.1d.gm/crypto/async/arch/async_posix.h:45: undefined reference to `setcontext'
/usr/lib/gcc/x86_64-alpine-linux-musl/10.3.1/../../../../x86_64-alpine-linux-musl/bin/ld: /usr/local/gmssl/lib/libcrypto.a(async_posix.o): in function `ASYNC_is_capable':
/root/gm/openssl-1.1.1d.gm/crypto/async/arch/async_posix.c:28: undefined reference to `getcontext'
...
```

使用

- 在ingress-controller启动时，会指定一个configmap，其内是 nginx 配置参数，大部分参数可以被 ingress 注解覆盖
- 介绍部分控制器的启动参数
  - default-ssl-certificate=[ns]/[name] 默认自签名，如设置则则使用此 secret

- controller-class：默认为 kubernets.io/ingress-nginx
- default-server-port 默认8181 必须有个监听端口
- enable-metrics 默认 true
- valid webhook 默认 9443
- profile-port 默认 10245
- healthz-port 默认10254，/healthz
- status-port 默认10246，/nginx_status
- nginx_status 默认 10247，用于lua tcp配置
- configmap 指定 nginx 配置，格式为 {ns}/{name}， 更多配置见这里
  - ssl-redirect 默认ture，当http连接时，返回https重定向
  - use-gzip：默认false
  - client-body-buffer-size：设置缓存 downstream 的 body 大小
  - load-balance：支持 round_robin(默认) | ewma 。如某些服务需要一致性hash，则可以在 ingress 注解中配置
    - nginx.ingress.kubernetes.io/affinity 通过 session/cookie
    - nginx.ingress.kubernetes.io/upstream-hash-by 通过 ip
- validating-webhook string 用于启动一个准入控制器以验证传入的Ingress的地，采用"[host]:port"的形式。如果未提供，则不会启动准入控制器。
- validating-webhook-certificate string 验证Webhook证书PEM文件的路径。
- validating-webhook-key string 验证Webhook密钥PEM文件的路径

问题

- tls证书创建方式
  - kubectl create -n ingress secret tls --key {x} --cert {x}
- 当 backend 错误时，跳转到下一个upstream
  - proxy_next_upstream: "error timeout http_502"
- 当 backend 需要大数据传输，需配置annotations
  - nginx.ingress.kubernetes.io/proxy-body-size: 8m
- 当然大部分时候也需要配置 client-buffer-size 或者 proxy-buffer-size，其默认大小分别是 16k 和 4K
  - nginx.ingress.kubernetes.io/client-body-buffer-size
  - nginx.ingress.kubernetes.io/proxy-buffer-size
- 当使用80端口时，不希望重定向到443时，添加注解
  - nginx.ingress.kubernetes.io/ssl-redirect: "false"
- 配置四层负载，需要修改启动参数

    - --tcp-services-configmap=[ns]/[name]
    - --udp-services-configmap=[ns]/[name]
- 当 backend 是 tls 时，并且希望透传, 需要有启动参数 ssl-enable-passthrough
    - nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    - nginx.ingress.kubernetes.io/backend-protocol: HTTPS

```
# $(POD_NAMESPACE)/tcp-services
apiVersion: v1
kind: ConfigMap
metadata:
  name: tcp-services
  namespace: kube-system
data:
  #监听端口: 负载端口
  "29418": default/review:29418
  "29420": default/repo:80
```

nginx

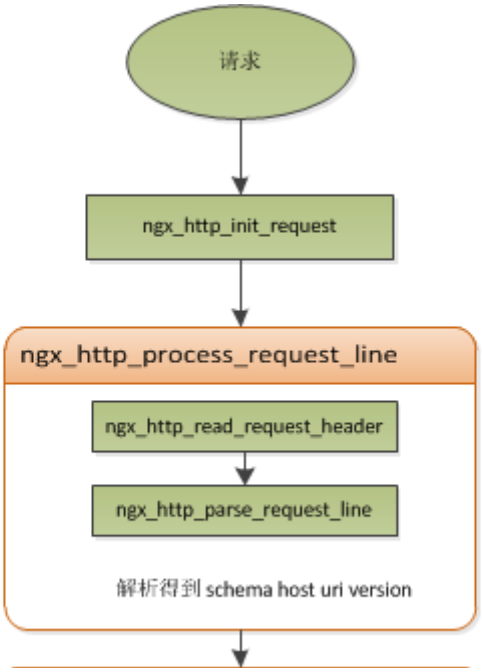
简介

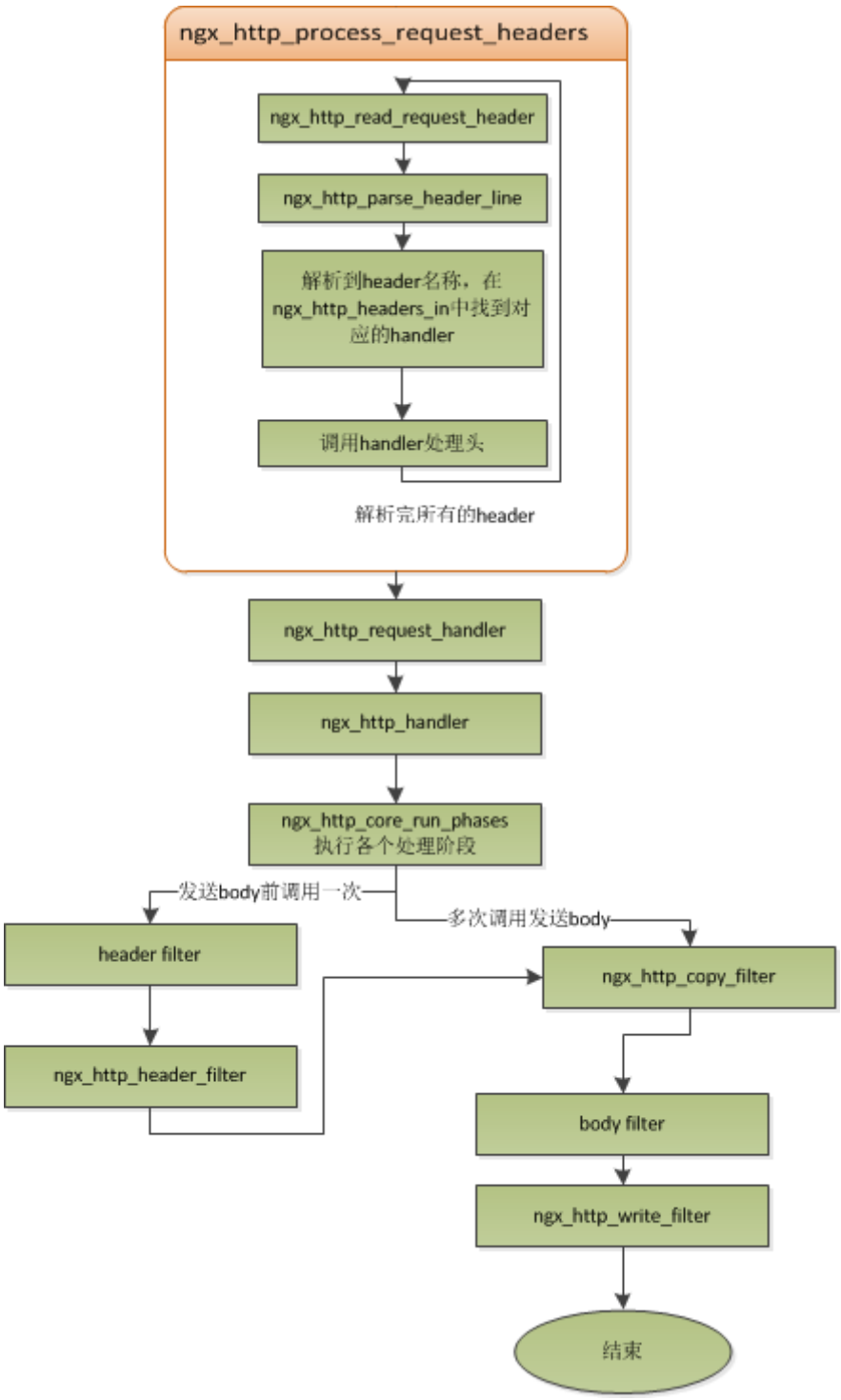
- nginx是以多进程的方式来工作的，当然nginx也是支持多线程的方式的，默认是多进程方式
- master 用来管理worker进程，包含：
    - 接收来自外界的信号，向各worker进程发送信号
    - 监控worker进程的运行状态，当worker进程退出后(异常情况下)，会自动重新启动新的worker进程
- worker进程之间是对等的，竞争来自客户端的请求，互相是独立的，一个请求只能在一个worker进程中处理
- master 新建 listenfd，之后fork worker进程并代入fd，worker注册listenfd读事件前抢accept_mutex，抢到互斥锁的那个进程注册listenfd读事件，在读事件里调用accept接受该连接。当一个worker进程在accept这个连接之后，就开始读取请求，解析请求，处理请求
    - nginx 写的时候，还没有好用的协程，所以使用线程时，会导致只绑定在同一进程，线程切换时会有很多额外的花销，所以采用进程模式
- nginx 为更好的利用多核特性，提供了cpu 绑定选项；
- 基本的web服务器来说，事件通常有三种类型，网络事件、信号、定时器，
    - 网络： nginx使用的是异步非阻塞模式，设置超时时间
    - 信号：若主程收到信号，在信号处理函数处理完后，epoll_wait 会返回错误，然后master再次进入epoll_wait

- 定时器：定时器事件是放在红黑树里面，每次在进入epoll_wait前，先从该红黑树里面拿到所有定时器事件的最小时间

概念

- connection：对tcp连接的封装，其中包括连接的socket，读事件，写事件；http请求的处理是建立在connection之上的，还包括mail 等
  - ngx_connection_t结构体
  - 每个worker有连接数的最大上限值，上限是 ulimit配置，能使用的连接是 work_connections 和 ulimit 取最小值
  - 每个worker都有独立的连接池，连接池的大小是worker_connections
  - 一个nginx能建立的最大连接数是 worker_connections * numberOfWorker，如是反向代理，则除以2(占用一个连接)
- request: ngx_http_request_t是对一个http请求的封装。 我们知道，一个http请求，包含请求行、请求头、请求体、响应行、响应头、响应体
  - 从ngx_http_init_request开始的，在其中，会设置读事件为ngx_http_process_request_line，主要解析 Host 和 Url
  - 当确定 Vhost 或者 Url 后，会进入下一阶段 ngx_http_process_request_headers
  - 之后，header 数据会分成两类，一类特殊值会保存在map中，如host，schema，uri等；另一类普通的会放在链表中；当读到 \r\n 后会进入下一阶段 ngx_http_process_request，设置当前的连接的读写事件处理函数为ngx_http_request_handler
  - 之后使用 ngx_http_handler，会设置读写事件，读事件为ngx_http_block_reading(不读取)；写事件为ngx_http_core_run_phases(执行多阶段请求处理)，在多阶段处理中，会将产生的响应头会放在ngx_http_request_t的headers_out中
- nginx会将整个客户端请求头都放在一个buffer里面，这个buffer的大小通过配置项client_header_buffer_size来设置，可以设置 large_client_header_buffers 配置大的buffer空间，当之前buffer不可用时会转移到大buffer中





context

- 这里介绍的是上图中的 core_run_phases 内部，以配置文件说明开始
- 配置文件内是存在不同块，每块属于不同上下文，配置是内层优先级最大
- 主要模块
  - main: 运行时，与具体业务功能无关的一些参数，比如工作进程数，运行的身份等。
  - http: 与提供http服务相关的一些配置参数。例如：是否使用keepalive啊，是否使用gzip进行压缩等。
  - server: http服务上支持若干虚拟主机。每个虚拟主机一个对应的server配置项，配置项里面包含该虚拟主机相关的配置

    - location: http服务中，某些特定的URL对应的一系列配置项
- 模块内主要配置
    - main：user，worker_processes，error_log，events，http
    - http: server
    - server:listen, server_name, access_log, location, protocol, proxy, smtp_auth, xclient
    - location: index, root
- 模块分类
    - event: 独立于操作系统的事件处理机制的框架，提供各具体事件的处理，具体有ngx_events_module，ngx_event_core_module和ngx_epoll_module等，具体使用何种事件处理模块，依赖于具体的操作系统和编译选项。
    - phase: 也被称为handler模块,负责处理客户端请求并产生响应内容，如 static, rewrite模块。执行以下几项任务：
        - 获取location
        - 发送response header
        - 发送response body
        - 按顺序经过以下几个阶段：
            - POST_READ_PHASE: 读取请求内容阶段
            - SERVER_REWRITE_PHASE: Server请求地址重写阶段
            - FIND_CONFIG_PHASE: 配置查找阶段:
            - REWRITE_PHASE: Location请求地址重写阶段
            - POST_REWRITE_PHASE: 请求地址重写提交阶段
            - PREACCESS_PHASE: 访问权限检查准备阶段
            - ACCESS_PHASE: 访问权限检查阶段
            - POST_ACCESS_PHASE: 访问权限检查提交阶段
            - TRY_FILES_PHASE: 配置项try_files处理阶段
            - CONTENT_PHASE: 内容产生阶段，三方模块主要在这阶段实现
            - LOG_PHASE: 日志模块处理阶段
    - filter: 也称为filter模块，主要是负责对phase输出的内容进行处理，可以对输出进行修改。可以实现对输出的所有html页面增加预定义的footbar，对输出的图片的URL进行替换等，以下模块执行顺序从后往前
        - write_filter_module,
        - header_filter_module,
        - chunked_filter_module,
        - range_header_filter_module,
        - gzip_filter_module,

- postpone_filter_module,
- ssi_filter_module,
- charset_filter_module,
- userid_filter_module,
- headers_filter_module,
- copy_filter_module,
- range_body_filter_module,
- not_modified_filter_module,

内置变量

- http://nginx.org/en/docs/http/ngx_http_core_module.html#Embedded Variable
- 注意只有 request 信息
- 每个变量取值都是 $ 开头，有两类特殊变量
  - arg_[name]：name是query的的参数名称
  - http_[name]：name是 header的参数名称，如http_cookie, http_user_agent

```
# 四层信息
remote_addr 客户端地址
remote_port 客户端端口
remote_user HTTP基础认证时的信息
server_addr 服务器端地址
server_name 服务器名，virtualHost
server_port 服务器端口
server_protocol 服务器的HTTP版本，如HTTP/1.1

# 七层信息
host: 依次顺序未 path中主机名>HOST请求头>符合请求的服务器名
request_uri: URI信息,且有请求参数，即? 后内容
uri: URI信息，内部跳转或使用 index files 会不同
args: 请求中的参数值，别名是query_string
scheme:  请求使用的Web协议，http 或 https
request: 原始的请求path信息
request_method: HTTP请求方法，通常为GET或POST
request_body: 客户端的请求主体
request_filename: 当前连接请求的文件路径，由root或alias指令与URI请求生成。
request_length: 请求的长度 (包括header和body)
body_bytes_sent: 传输给客户端的字节数，Header不计算在内
bytes_sent: 传输给客户端的字节数
content_length: 来自请求头content_length字段内容
content_type: 来自请求头content_type字段内容
```

```
# 其他
request_time: 处理请求的时间,单位ms
request_completion: 如果请求成功，值为”OK”，其他则为空
status: HTTP响应代码
is_args: 如果请求中有参数，值为“?”，否则为空字符串。
limit_rate: 用于设置响应的速度限制
msec: 当前的Unix时间戳
pid: 工作进程的pid
```

proxy

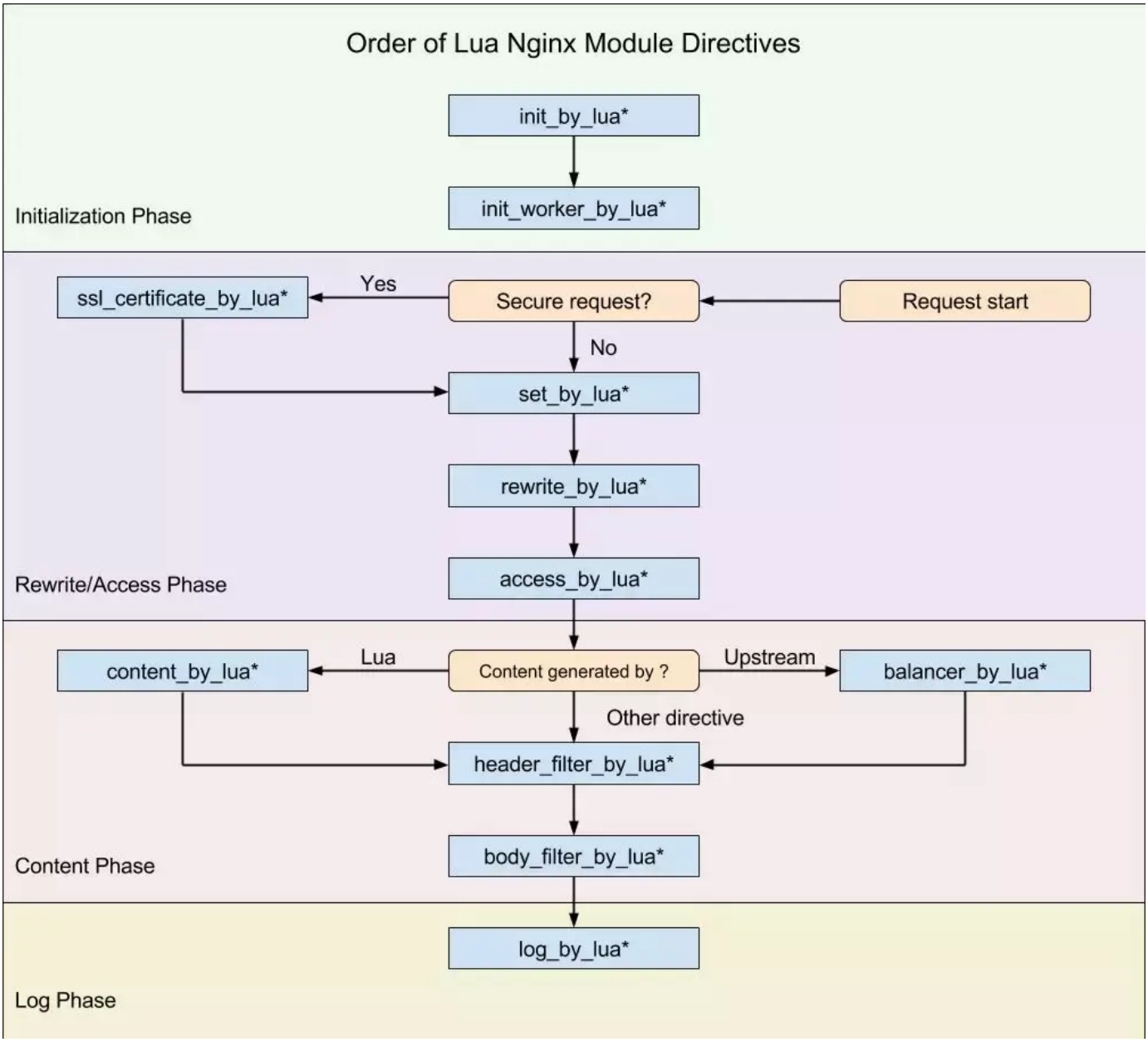
- nginx主要用于反向代理，
  - upstream 是被代理的服务，即 backend
  - downstream 是请求的服务，即 fronent，或者客户端
- client 配置，描述的是 fronent 的配置，nginx 对请求数据校验，甚至缓存
  - client_body_buffer_size 设置用于读取request缓冲区大小。如果请求正文大于缓冲区，则整个正文或仅写入其部分到临时文件，16k
  - client_body_temp_path 默认 client_body_temp
  - client_body_timeout 默认 60s，读取客户端请求正文的超时，针对两个连续读取操作之间的时间段设置，当时间内未传输任何内容，则请求将终止，并 408 错误
  - client_header_buffer_size：请求头的缓存去大小
  - client_header_timeout：默认 60s
  - client_max_body_size：默认1m，设置客户端请求主体的最大允许尺寸，如果请求中的大小超过了配置的值，返回 413 错误，0则禁用检查
- 在 nginx 中还见到有 proxy_xxx 配置，这里是对 backend 的回复进行处理，buffer 和cache 区别
  - buffer 是从 server -> client 数据放在本地再一次性给客户端
  - cache 是充当server，不和upstream交互，直接返回
  - buffering： 默认on，接收来自 upstream 的数据，并将其保存在由设置的缓冲区内，主要是减少和downstream交互时间。当响应不适合内存，则会将其的一部分保存到磁盘上的临时文件中，写入的临时文件由 proxy_max_temp_file_size 和 proxy_temp_file_write_size 控制
  - buffers： 设置数量和大小，默认8 4k，即8个4k大小

优化

- worker_processes表示启动的Nginx工作进程数，可以通过调整这个值来充分利用服务器的多核CPU资源。一般将worker_processes设置为CPU核心数的两倍
- worker_connections表示一个Nginx工作进程可以同时处理的连接数，可以根据服务器硬件资源进行调整
- sendfile和tcp_nopush是Nginx优化性能的两个重要配置项

- aio 和 directio 用于配置大文件传输时，directio 配置当大于该大小时使用 aio
- 正则表达式：会消耗大量的CPU资源，因此应尽量避免过多的正则表达式匹配
- if语句会影响性能，因此应尽可能避免使用if语句，使用location指令来替代
- 避免使用access_log和error_log
- client_body_buffer_size和client_header_buffer_size指定请求头和请求体的缓冲区大小
- client_max_body_size指定请求体的最大大小, 若太小容易断流(413)
- large_client_header_buffers指定缓存大请求头的大小
- malloc 算法，当前 mimalloc 性能优于 jemalloc,pmalloc 等

lua



- init_by_lua 范围：http
  - master 进程在加载配置文件时运行指定的 lua 脚本，通常用来注册 lua 的全局变量或在服务器启动时预加载 lua 模块。例如 lua_shared_dict 共享内存的申请，只有当 nginx 重起后，共享内存数据才清空，常用于统计

- init_worker_by_lua
  - worker进程启动时调用指定的lua代码，通常用来创建每个工作进程的计时器(通过lua的ngx.timer API)，进行后端健康检查或者其它日常工作
- set_by_lua server, server if, location, location if
  - 设置个变量，常与计算逻辑，然后返回结果，
  - 该阶段不能运行Output API、Control API、Subrequest API、Cosocket API
- rewrite_by_lua http, server, location, location if
  - 作为rewrite阶段的处理，为每个请求执行指定的lua代码。注意这个处理是在标准HtpRewriteModule之后进行的
- access_by_lua http, server, location, location if
  - 请求在访问阶段后调用，收集到大部分的变量。这条指令运行于nginx access阶段的末尾，因此总是在 allow 和 deny 这样的指令之后运行，可以修改请求头
- content_by_lua location, location if
  - 为每个请求执行lua代码，为请求者输出响应内容。此阶段是所有请求处理阶段中最为重要的一个，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
- header_filter_by_lua http, server, location, location if
  - 用来设置响应 cookie 和 headers
- body_filter_by_lua http, server, location, location if
  - 会在一次请求中被调用多次, 这是实现基于 HTTP 1.1 chunked 编码
    log_by_lua http, server, location, location if

密码学

- 阅读 https://www.crypto101.io/
- 一对密钥只做一个用途，要么用作非对称加解密，要么用作签名验证，确保前向安全性(PFS), 反映到密钥协商过程中，就是：
  - 不要使用RSA做密钥协商，一定只用RSA做数字签名。
  - 不要把ECDH的公钥固定内置在客户端做密钥协商
- 常见分类
  - 块加密算法 block cipher: AES, Serpent 等
  - 流加密算法 stream cipher: RC4，ChaCha20 等
  - Hash函数:md5，sha256，sha512, poly1305 等
  - 消息认证码(MAC): message authentication code: HMAC-sha256，AEAD（AES128-GCM） 等
  - 密钥交换 key exchange: DH，ECDH，RSA，前向安全(DHE，ECDHE）
  - 公钥加密 public-key encryption: RSA，rabin-williams 等

    - 数字签名算法 signature algorithm: RSA，DSA，ECDSA (secp256r1 , ed25519) 等
    - 密码衍生函数 key derivation function: SHA-256 , bcrypto，scrypto，pbkdf2 等
    - 随机数生成器 random number generators: /dev/urandom 等

块加密

- 对固定长度的分组进行加密的算法， block ciphers, 公式
    - 加密：C = E(k, P) ; 解密：P = D(k, C)
- block cipher: 常见有 AES, DES, 3DES 等，AES 更安全
- 遗留问题：只能发送非常有限长度的消息，因为i块密码的分组长度。显然，我们希望能够发送更大的消息，或者理想情况下，发送大小不确定的流。我们将使用流密码来解决这个问题
- AES：限制 128 位的块大小和 128、192 和 256 位的密钥大小

流加密

- 加密比特流的对称密钥加密算法，stream ciphers
- ECB模式：电子密码本模式，将流分成不同块，单独加密，相同的输入块将始终映射到相同的输出块，非常严重的安全缺陷
- CBC模式：使用块加密方式，明文加密前使用前一个密文块异或，第一个明文块需要随机数(IV)来异或，通常直接使用密钥做随机数。但CBC 已经不安全，导致 BEAST 攻击
- 本地流密码（CTR模式）：从对称密钥产生伪随机位，该流称为密钥流，然后与明文进行异或生成密文。其实现有 RC4
- ChaCha20：目前很安全，并且其性能可与现代 AES 竞争，尽管后者拥有专门的硬件

密钥交换

- Key exchange protocols
- 上一节中密钥已经可以加密足够多数据，但是如何双方达成同样的密钥呢？该协议称为 Diffie-Hellman（简称 DH），其实现依赖于数学问题，这些问题认为在“错误”方向上解决起来非常复杂，但在“正确”方向上很容易计算
- 基于椭圆曲线离散对数问题的 Diffie-Hellman 算法的实际实现，称为 ECC
- 虽然 DH 协议成功在两个对等点之间产生共享秘密，但仍然缺少构建这些密码系统的一些难题，使用何种工具验证对方的身份，来解决 MITM 攻击
- ECDH
- ECDHE

公钥加密

- Public-key encryption：不涉及单个密钥的密码系统，而是一对密钥：一个公钥（可以自由分发）和一个私钥（独自拥有）
- 可用常见的 RSA， 公钥算法不限于加密，还可以做以下：

    - 密钥交换算法(DH协议)；
    - 块加密算法；
    - 签名算法
- 不对所有事情都使用公钥加密原因在于其性能非常差，和 chacha20 可以达到4数量级以上差距，并且公钥加密一次只能加密小块数据 256B 左右
- 到目前为止，只讨论了加密，没有任何形式的身份验证。虽然可以加密和解密消息，但无法验证该消息是否是实际发送者发送的消息

哈希函数

- hash function： 用不确定长度的输入并生成固定长度值的函数，也称 摘要
- 常见实现有
    - MD5: 输出 128 位摘要，目前，不建议使用 MD5 来生成数字签名
    - sha-1
    - sha-2: 包含一系列哈希函数，如SHA-512/224 和 SHA-512/256。哈希函数基于 Merkle 结构，可用于数字签名、消息认证和随机数生成器。 SHA-2 不仅性能优于 SHA-1，而且由于其抗碰撞性的提高，还提供了更好的安全性。
    - sha-3: 包含SHA3-224、SHA3-256， 这听起来与 SHA-2 属于同一家族，但两者的设计却截然不同。与 SHA-2 相比，SHA3 在硬件方面非常高效，在软件方面相对较慢
- 密码存储：最常见的用例是密码存储
    - 直接存储哈希后的密文，容易受到彩虹表攻击，彩虹表如此有效的原因：相同的密码将在各处产生相同的哈希值。建议加入 salt 值，存储在数据库中的密码哈希旁边。当进行身份验证时，您只需将盐与密码结合起来，对其进行散列，然后将其与存储的散列进行比较
    - 弱密码攻击：不管是否有 salt，直接暴力破解，因此为保护密钥，提出密钥导出函数，将在后章节中更详细地讨论它们
- 哈希树：每个节点是哈希值(由其内容和其祖先的哈希值组成)，通常称为 Merkle 树，像这样的系统被许多系统使用，特别是分布式系统。包括分布式版本控制系统（例如 Git）、数字货币（例如比特币）、分布式点对点网络（例如 Bittorrent）以及分布式数据库（例如 Cassandra）
- 遗留问题：哈希函数本身无法验证消息，因为任何人都可以计算它们。此外哈希函数不能用于保护密码

消息认证码

- Message authentication codes：用于检查消息的真实性和完整性，未经身份验证的加密无法校验，这就是引入 MAC 的原因
    - 和哈希函数不同在于，加入密钥来确保来自
- 算法采用任意长度的消息和固定长度的密钥，并生成标签。 MAC 算法还附带验证算法，该算法采用消息、密钥和标签，并告诉标签是否有效。 仅重新计算并验证标签是否相同并不总是足够，许多安全 MAC 算法是随机的，每次应用时都会产生不同的标签

- 考虑下tarball文件，和其 sha256 存放在一个目录，如果要替换tar文件，那替换sha文件也可以，所以 MAC 目的是验证 sha 文件是上传者的
- 将密文与 MAC 组合的常见方法有 3 种，K 是密钥，P是消息，t是标签，E是加密
  - 验证和加密。您进行身份验证并启用单独加密明文。这就是 SSH 的工作原理，在符号中：C = E(K, P), t = MAC(K, P)，并且您发送密文 C 和标签 t
  - 验证然后加密。对明文进行身份验证，然后对明文和身份验证标记的组合进行加密。这就是 TLS 通常的做法，在符号中：t = MAC(K, P), C = E(K, P ∥ t)，并且您只发送 C
  - 加密然后验证。加密明文，计算该密文的 MAC。这就是 IPSec 的工作原理。在符号中：C = E(K, P), t = MAC(K, C)，并且您发送 C 和 t。有资深人员认为这种方式是一个缺陷
- 认证加密模式(AEAD)：前提是许多消息实际上由两部分，实际内容 + 元数据。AEAD 模式通过提供指定方式将元数据添加到加密内容来解决此问题，以便对整个加密内容和元数据进行身份验证
  - 主流的AEAD模式是 aes-gcm-128/aes-gcm-256/chacha20-poly1305

签名算法

- Signature algorithms， 相当于消息认证码的公钥，由三部分组成
  - 密钥生成算法: 得到密钥信息
  - 签名生成算法: 使用密钥和消息，生成签名
  - 签名验证算法
- RSA：签名算法可以使用公钥加密构建。签名使用私钥，公钥来解释消息
- DSA(数字签名算法): 密钥生成分两步进行。第一步是选择参数，这些参数可以在用户之间共享。第二步是为单个用户生成公钥和私钥
- ECDSA: 椭圆曲线数字签名算法，实现有secp256r1 , ed25519)

密钥派生函数

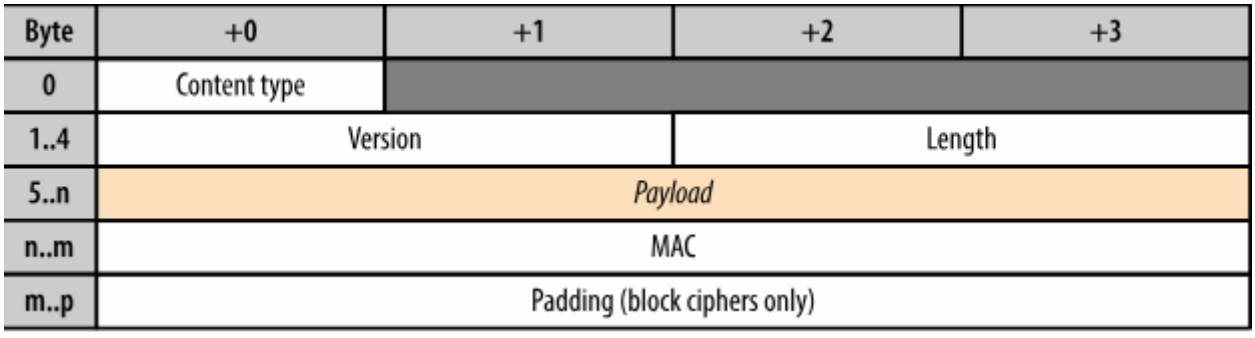
- Key derivation functions，简称 PRF
- 作用：从一个秘密值导出一个或多个秘密值（密钥）的函数，通常因为秘密值太短或者不够用
- 目标
  - message privacy (通过加密encryption实现，所有信息都加密传输)
  - 完整性， message integrity（ 通过MAC校验机制，被篡改会立刻发现 ）
  - 认证， mutual authentication （双方都可以配备证书，防止身份被冒充 ）
  - 通用性 （ 公开 rfc，不受限于任何专利技术）
  - 可扩展性 (通过扩展机制添加新功能）
  - 高效率 （通过session cache，恰当部署cache之后，tls的效率很高）

tls 简介

套件

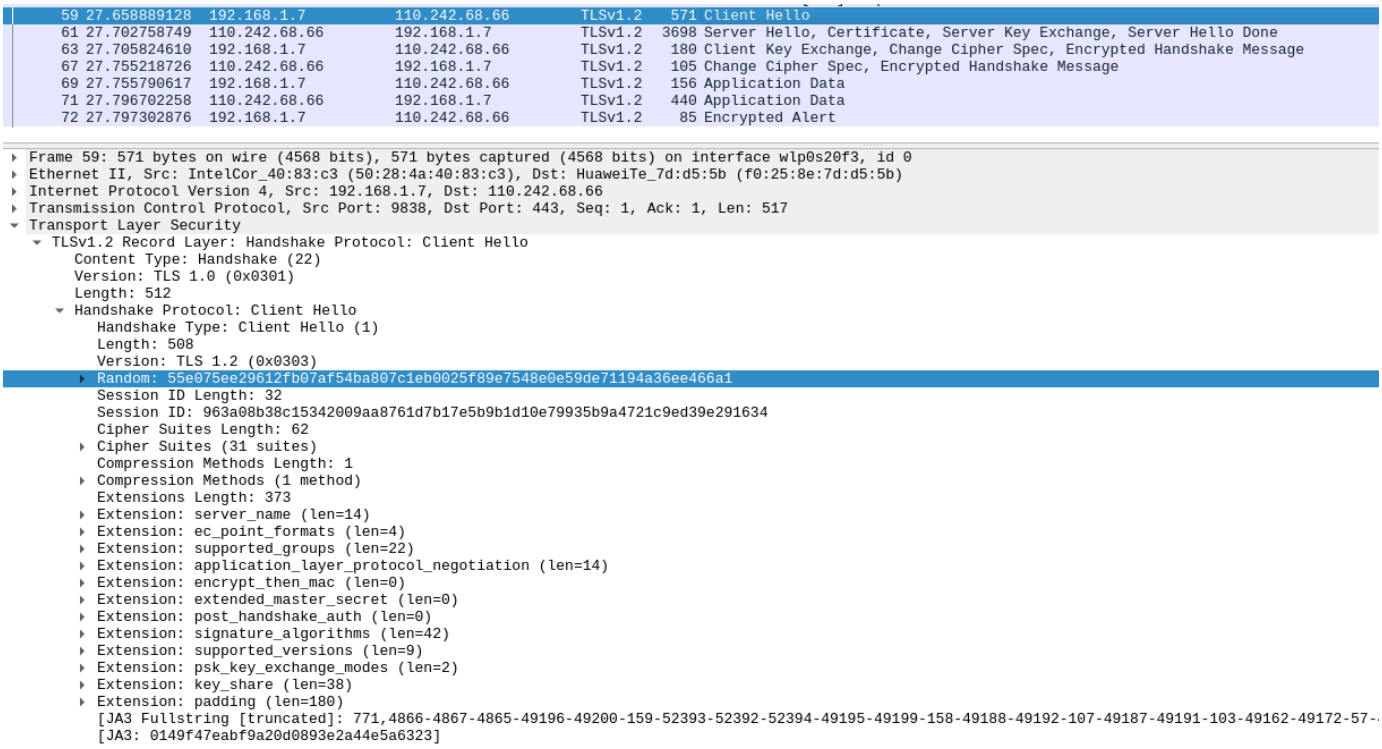
- CipherSuite是4个算法的组合:
  - key exchange(密钥交换)算法: RSA, ECDHE 等
  - authentication (认证)算法 ： 非对称签名算法，RSA, DHE, ECDSA 等
  - encryption(加密)算法：对称加密
  - message authentication code (消息认证码 简称MAC)算法 : SHA256
- 查看支持的密钥列表，介绍每列含义
  - openssl ciphers -V | column -t
- 套件 ECDHE-RSA-AES128-GCM-SHA256 为例
  - 密钥交换使用 ECDHE
  - 认证使用 RSA
  - 对称加密使用 AES128-GCM
  - 消息认证码使用 SHA256
- 套件 AES256-SHA256 为例：Kx=RSA, Au=RSA, Enc=AES(256), Mac=SHA256
- 套件 rsa_aes_128_gcm_sha256：交换/认证 RSA, 加密 AESGCM(128), 消息认证码 SHA256，前面已经提到，这不是前向安全

```
// nginx 配置
ssl_certificate      /opt/certs/certificates/_.yylt.space.crt;
ssl_certificate_key  /opt/certs/certificates/_.yylt.space.key;
# 或者只 TLSv1.3 ，不设置 ciphers
ssl_protocols   TLSv1.2 TLSv1.3;
ssl_ciphers     ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-
SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-
AES256-GCM-SHA384;
```



- TLS协议使用 record 数据包，内部有以下协议，使用 contentType 区分
  - handshake 协议，做认证密钥协商
  - changecipher spec 协议, 用来通知对端从handshake切换到record协议, 有点冗余，在TLS1.3里面已经被删掉了
  - alert协议，the alert protocol, 用来通知各种返回码，
  - application data协议，把http，smtp等的数据流传入record层传输

握手



```
    59 27.658889128  192.168.1.7          110.242.68.66        TLSv1.2   571 Client Hello
    61 27.702758749  110.242.68.66        192.168.1.7          TLSv1.2  3698 Server Hello, Certificate, Server Key Exchange, Server Hello Done
    63 27.705824610  192.168.1.7          110.242.68.66        TLSv1.2   180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
    67 27.755218726  110.242.68.66        192.168.1.7          TLSv1.2   105 Change Cipher Spec, Encrypted Handshake Message
    69 27.755790617  192.168.1.7          110.242.68.66        TLSv1.2   156 Application Data
    71 27.796702258  110.242.68.66        192.168.1.7          TLSv1.2   440 Application Data
    72 27.797302876  192.168.1.7          110.242.68.66        TLSv1.2    85 Encrypted Alert
```

```
▸ Frame 59: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_40:83:c3 (50:28:4a:40:83:c3), Dst: HuaweiTe_7d:d5:5b (f0:25:8e:7d:d5:5b)
▸ Internet Protocol Version 4, Src: 192.168.1.7, Dst: 110.242.68.66
▸ Transmission Control Protocol, Src Port: 9838, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 508
        Version: TLS 1.2 (0x0303)
      ▸ Random: 55e075ee29612fb07af54ba807c1eb0025f89e7548e0e59de71194a36ee466a1
        Session ID Length: 32
        Session ID: 963a08b38c15342009aa8761d7b17e5b9b1d10e79935b9a4721c9ed39e291634
        Cipher Suites Length: 62
      ▸ Cipher Suites (31 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 373
      ▸ Extension: server_name (len=14)
      ▸ Extension: ec_point_formats (len=4)
      ▸ Extension: supported_groups (len=22)
      ▸ Extension: application_layer_protocol_negotiation (len=14)
      ▸ Extension: encrypt_then_mac (len=0)
      ▸ Extension: extended_master_secret (len=0)
      ▸ Extension: post_handshake_auth (len=0)
      ▸ Extension: signature_algorithms (len=42)
      ▸ Extension: supported_versions (len=9)
      ▸ Extension: psk_key_exchange_modes (len=2)
      ▸ Extension: key_share (len=38)
      ▸ Extension: padding (len=180)
        [JA3 Fullstring [truncated]: 771,4866-4867-4865-49196-49200-159-52393-52392-52394-49195-49199-158-49188-49192-107-49187-49191-103-49162-49172-57-
        [JA3: 0149f47eabf9a20d0893e2a44e5a6323]
```

- Client Hello: 提供 sessionid，random 和支持的 套件，以及很多有用扩展，更多见 http://www.iana.org/assignments/tls-extensiontype-values
  - server_name: 即sni，告知对端的访问的host
  - alpn：支持的协议，如 h2，http/1.1
  - support_version: 支持的tls版本
  - signature_algorithms：支持签名算法，要求服务证书必须以此实现

```
▾ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 104
  ▾ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 100
      Version: TLS 1.2 (0x0303)
    ▸ Random: a8d176a9f42a9e01b786aad21241b4b98ffb94ca84626b5fe91ef1a76ff9a0c3
      Session ID Length: 32
      Session ID: fc884de51d31e0b932a02ad7cdcd8da8b11e81caec30548284b2b96a8244621b
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
      Extensions Length: 28
    ▸ Extension: renegotiation_info (len=1)
    ▸ Extension: ec_point_formats (len=4)
    ▸ Extension: application_layer_protocol_negotiation (len=11)
      [JA3S Fullstring: 771,49199,65281-11-16]
      [JA3S: 1089ea6f0461a29006cc96dfe7a11d80]
▾ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3183
  ▾ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3179
      Certificates Length: 3176
    ▾ Certificates (3176 bytes)
        Certificate Length: 1874
      ▸ Certificate: 3082074e30820636a0030201020210058ba3a807bc5a8dc3886f0d05787cd4300d06092a… (id-at-commonName=www.baidu.cn,
        Certificate Length: 1296
      ▸ Certificate: 3082050c308203f4a0030201020210051f0c7eddc88dbaf00c50e285f42265300d06092a… (id-at-commonName=DigiCert Secu
▾ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 333
  ▾ Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 329
    ▾ EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 048c105f45d2cccf37125c0a5f1e2db022222417d001140b8c9ff0a810721ffe28f8af34…
      ▸ Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
        Signature Length: 256
        Signature: 2a8e119128bb5d577040b229f355a0494319d1b69ba00766b04e4fcbada296ad39415a8d…
▾ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
    Content Type: Handshake (22)
```

- 上图中描述 server->client 信息，包括
  - Server Hello：包括支持的tls版本
  - Certificate： 证书
  - ServerKeyExchange（可选）
  - CertificateRequest（可选）
  - ServerHelloDone

```
▾ Transport Layer Security
  ▾ TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
  ▾ TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 53
      Encrypted Application Data: 07d6338a0cf9b40dcb87cb1284a3f1ac6b0493d32c1625dcd3561f522a834fbce1a164fc…
      [Application Data Protocol: http-over-tls]
```

-上图中是 client->server 信息

- Certificate（可选）
- ClientKeyExchange：看上去是成功意思
- CertificateVerify（可选）

- 计算对称密钥计算

    - master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)
    - 客户端和服务端都会使用相同 PRF算法计算 真正公钥，长度48位，
    - 加入随机数的原因主要是为了防止重放攻击，确保每次都不一样
- 当 finish 完成后，会得到数据有
    - client write MAC key
    - server write MAC key
    - client write encryption key
    - server write encryption key
    - client write IV
    - server write IV
- 在密码学中，对称加密算法一般需要encryption key，IV两个参数，MAC算法需要MAC key参数，因此都用于不同的用途
- AEAD是MAC和encrypt的集成，所以输入数据不需要在算MAC
- 不成功问题原因
    - tls 版本太低
    - cipher suite 不能达成一致
    - 服务端要校验客户端证书，但客户端未提供
    - 证书问题
- 证书验证过程
    - 查看哪个ca签名，验证ca是否在信任库中
    - 用证书的公钥解密签名，得到网站的校验码。并用同样算法算出校验码，验证是否相同表明证书没有被篡改
    - 取出 CN 和扩展字段 Alternative name ,若访问服务的地址在这里面，则说明这个证书不正确

证书

- https://www.cnblogs.com/charlieroro/p/10948173.html
- 查看证书内容
    - openssl x509 -in crt.pem -text
- 证书要求格式必须是 x509.v3 格式，并且服务器证书的公钥，必须和选择的密钥交换算法配套，证书信息如下：
    - 版本号 EXPLICIT Version DEFAULT v1
    - 序列号 CertificateSerialNumber
    - 签名 AlgorithmIdentifier
    - 颁发者 Issuer Name

    - 证书有效性 Validity （有效日期）
    - 主题 Subject Name
        - Common Name | 通用名称 | CN
        - Organizational Unit name |机构单元名称 |OU
        - Organization name| 机构名 |O
        - Locality |地理位置 |L
        - State or province name |州/省名 |S
        - Country |国名 |C
    - 主题公钥信息 SubjectPublicKeyInfo
    - 颁发者唯一身份信息 IMPLICIT UniqueIdentifier OPTIONAL
    - 主题唯一身份信息 IMPLICIT UniqueIdentifier OPTIONAL
    - 扩展信息 X509v3 extensions
        - 常用的有 subject alertName，Key Usage
    - 签名算法 Signature Algorithm
- 配套要求，前部分是密钥交换，后部分是数字签名(认证算法)
    - RSA / RSA_PSK：需 rsa 公钥，且keyEnciphermen需置位
    - DHE_RSA / ECDHE_RSA：需 rsa 公钥，digitalSignature需置位
    - DH_DSS / DH_RSA：需DH公钥，要求 keyAgreement 必须置位
    - ECDH_ECDSA / ECDH_RSA：公钥必须使用客户端支持的ec曲线和点格式
    - ECDHE_ECDSA：公钥必须使用客户端支持的ec曲线和点格式
- 证书扩展字段
    - Constraints：是否可以作为CA证书，及其可签发的证书层级
    - Subject Alternative Name：主体备用名称，多域证书
    - Extended Key Usage：具体地指定公钥的使用场景，如服务器身份验证、签名等
        - server auth：必须设置digital Signature，用于服务端校验
        - client auth：必须设置digital Signature，用于客户端校验
    - Key Usage：公钥应如何使用，如仅限签名，具体有以下数值
        - digital Signature：用于数字签名，可以用于终端证书
        - non Repudiation：当公钥校验签名时
        - key encipherment: 公钥用于对private key进行加密时
        - data encipherment：公钥用于直接加密原始的用户数据
        - key agreement：公钥用于密钥协商的时候设置
        - key CertSign：公钥用于校验证书的签名时，需 constrainte 的 CA 设置

国密

- https://www.gmssl.cn/gmssl/index.jsp
  - SM1，sm4，sm7 是分组加密算法
  - SM2，sm9 是非对称加密算法
  - SM3 是密码杂凑算法
- 使用国密替换 openssl 后，不是所有套件都支持，只有以下
  - ECDHE-RSA-AES128-GCM-SHA256
  - AES128-SHA
  - DES-CBC3-SHA
  - ECC-SM4-CBC-SM3
  - ECC-SM4-GCM-SM3
- https://curl.gmssl.cn/ 下载 gmcurl
  - gmcurl --gmssl -k -H 'host: foo.com' --verbose https://127.0.0.1

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:AES128-SHA:DES-CBC3-SHA:ECC-SM4-CBC-SM3:ECC-SM4-GCM-SM3;
ssl_verify_client off;
```