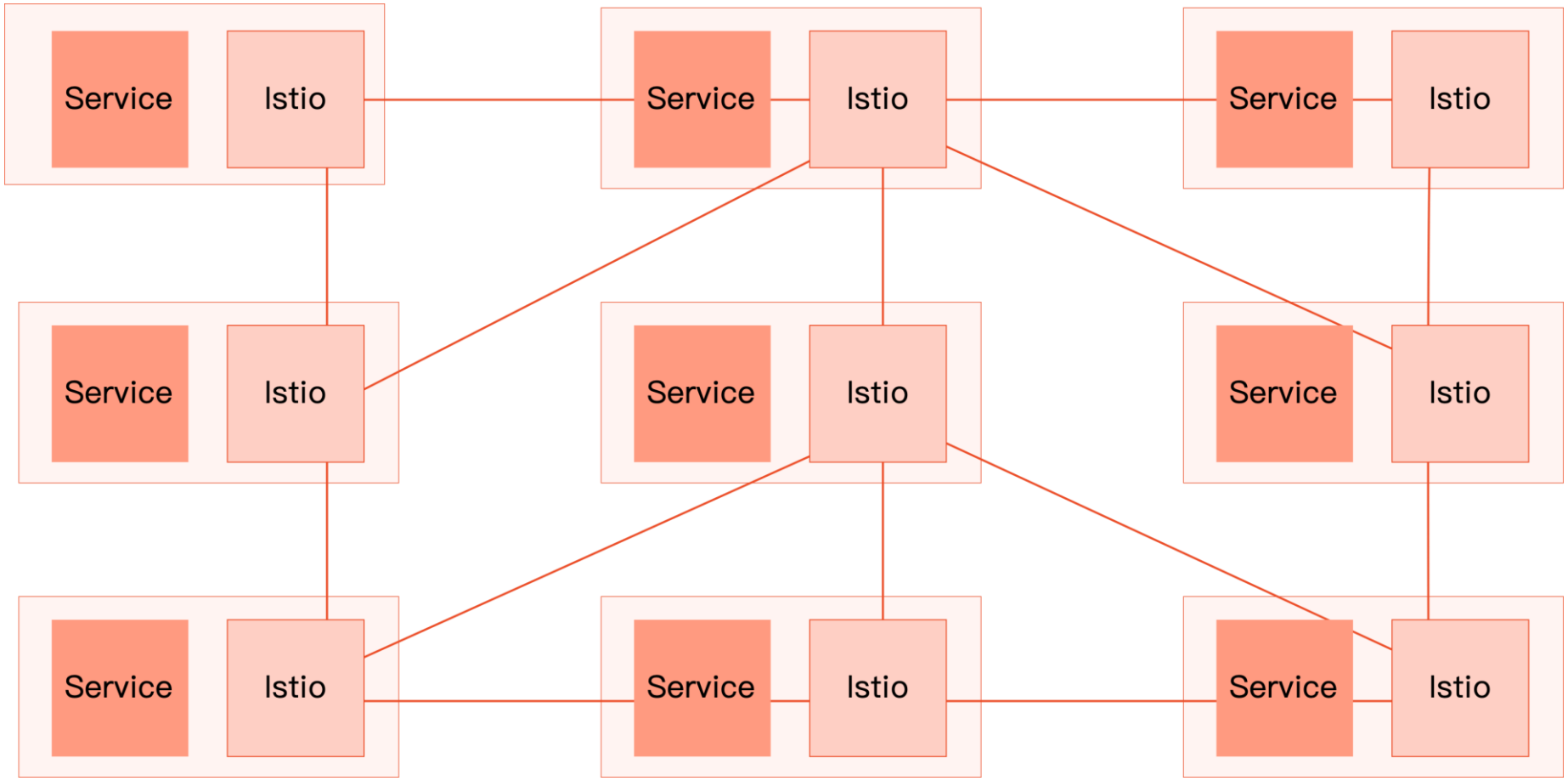
第九季: 探索NGINX UNIT

探索NGINX UNIT

- 用Unit实现应用的动态配置
- Unit的负载均衡配置
- Unit架构设计
- Unit源代码解读

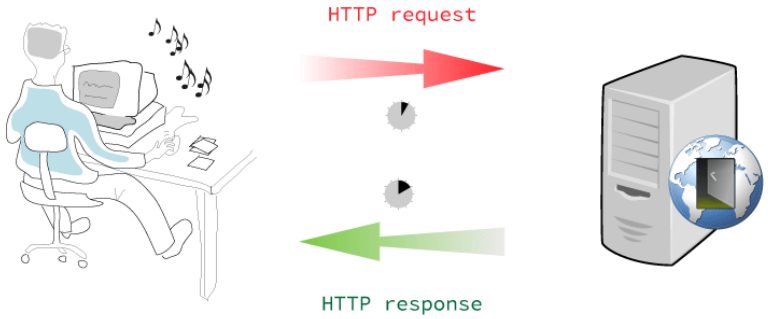
微服务架构的挑战


Service
Istio
Service
Istio
Service
Istio
Service
Istio
Service
Istio
Service
Istio
Service
Istio
Service
Istio
Service
Istio

动态Server VS 静态Server

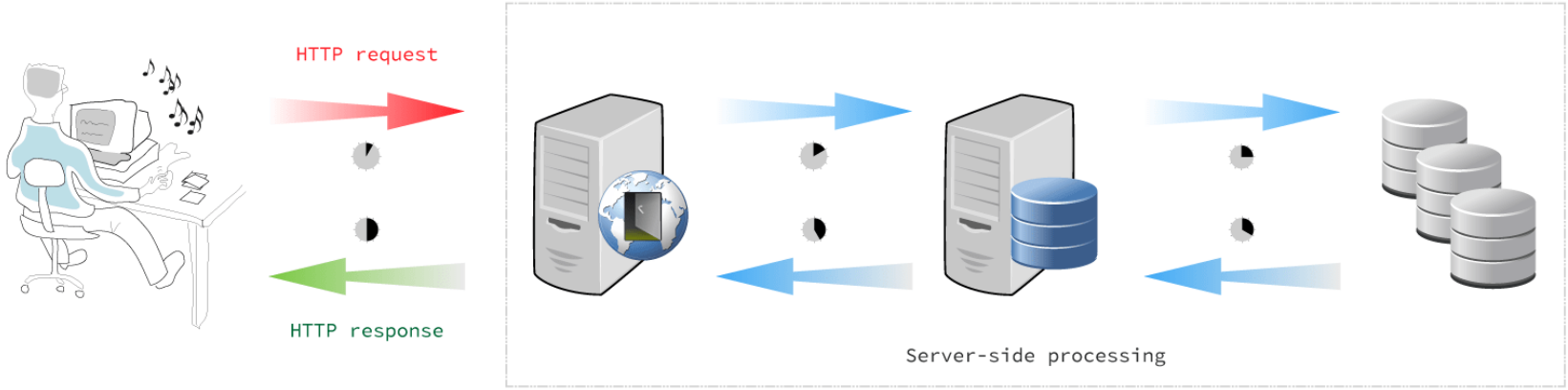
Scheme A

Static Website

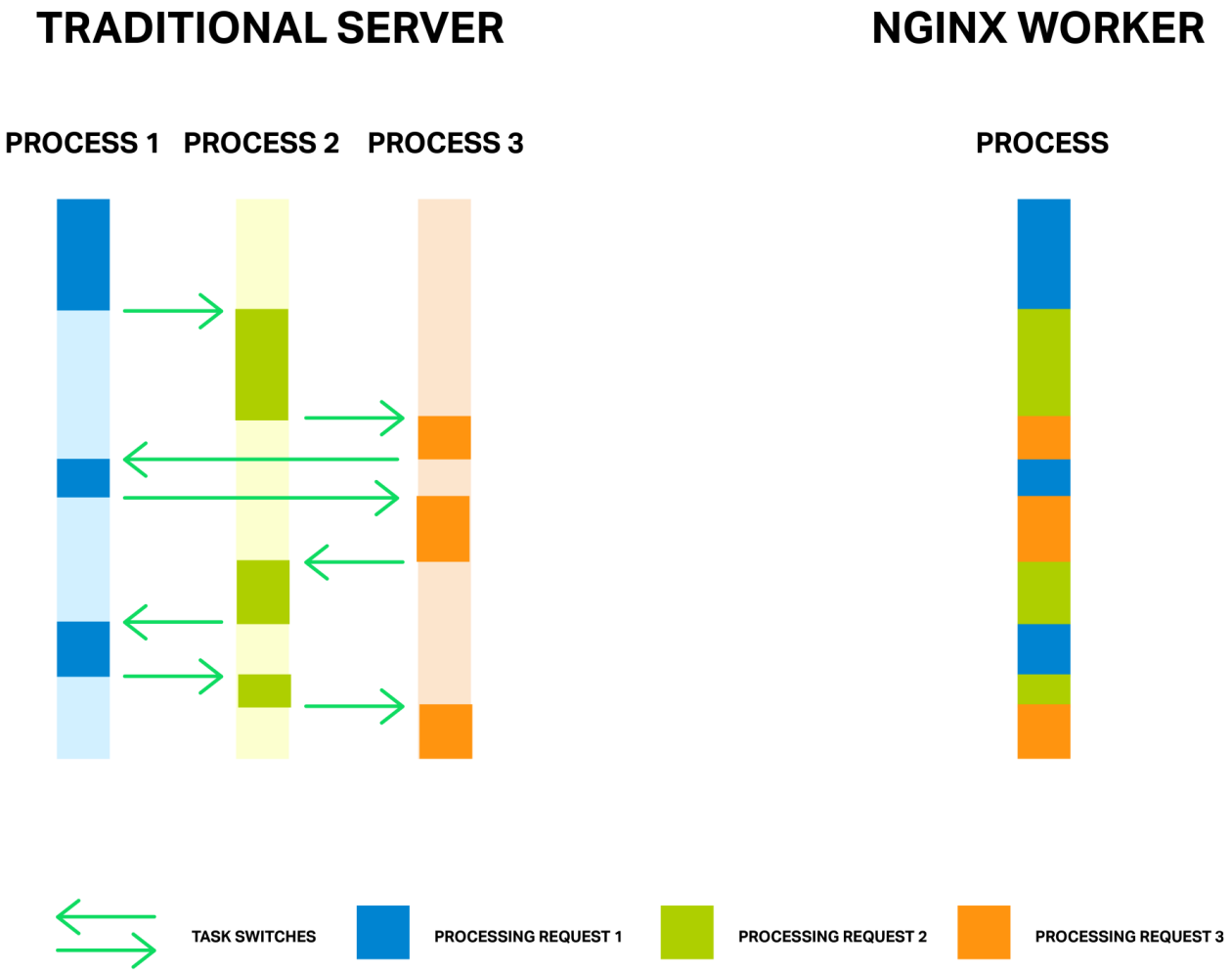


Scheme B

Dynamic Website

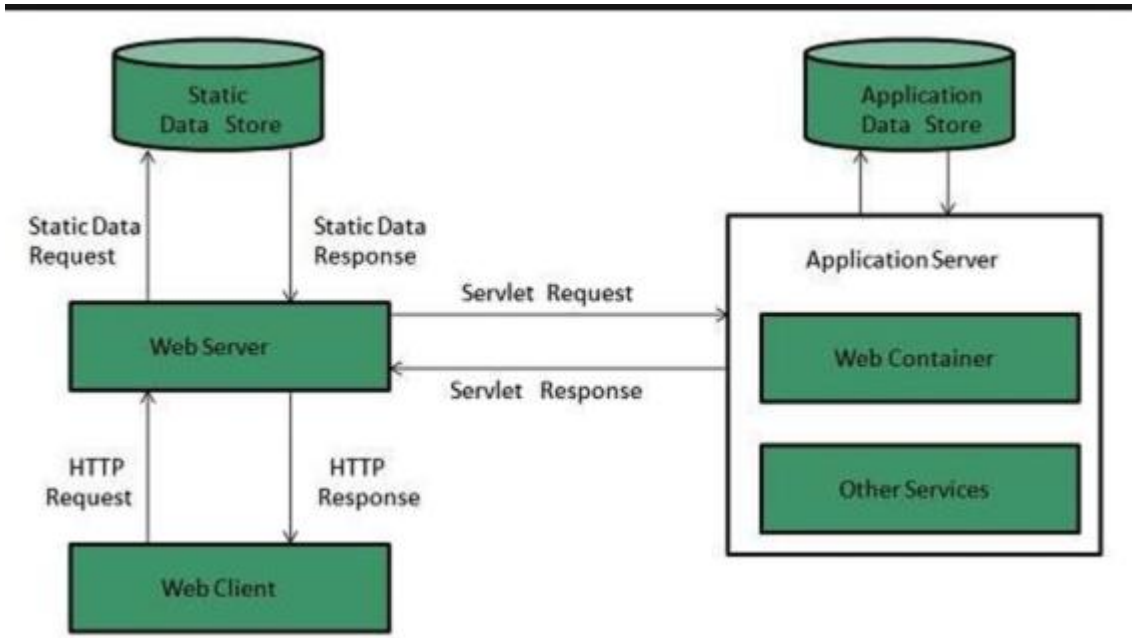


运行效率 VS 开发效率


TRADITIONAL SERVER
NGINX WORKER
PROCESS 1
PROCESS 2
PROCESS 3
PROCESS
TASK SWITCHES
PROCESSING REQUEST 1
PROCESSING REQUEST 2
PROCESSING REQUEST 3

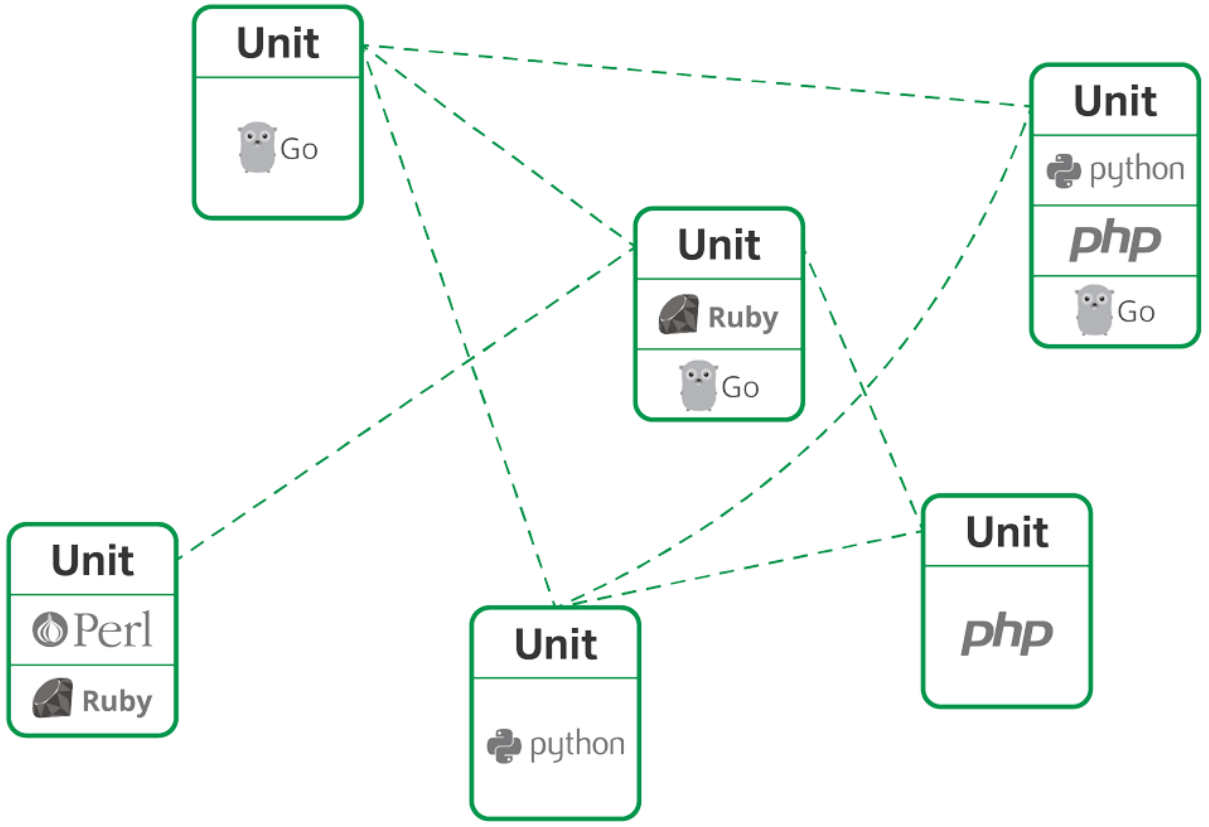
Web容器的抽象

- 进程/线程调度管理
- 网络协议处理
- 通用配置管理
- 环境变量传递参数



NGINX UNIT支持的编程语言

- 代码侵入
  - GoLang
  - NodeJS
- 非代码侵入
  - Ruby
  - Python2/3
  - Java
  - Php
  - Perl
  - WebAssembly



Unit支持的Web框架

- Python
  - WSGI：Django, Flask，Bottle, Pyramid,
  - ASGI：Django Channels, FastAPI, Guillotina, Quart, Responder, Sanic, Starlette,
- PHP
  - CakePHP, CodeIgniter, Laravel, Symfony, Yii
- Node.js
  - Express,
- Ruby
  - Ruby On Rails
- Perl
  - Catalyst,

Nginx Unit的能力

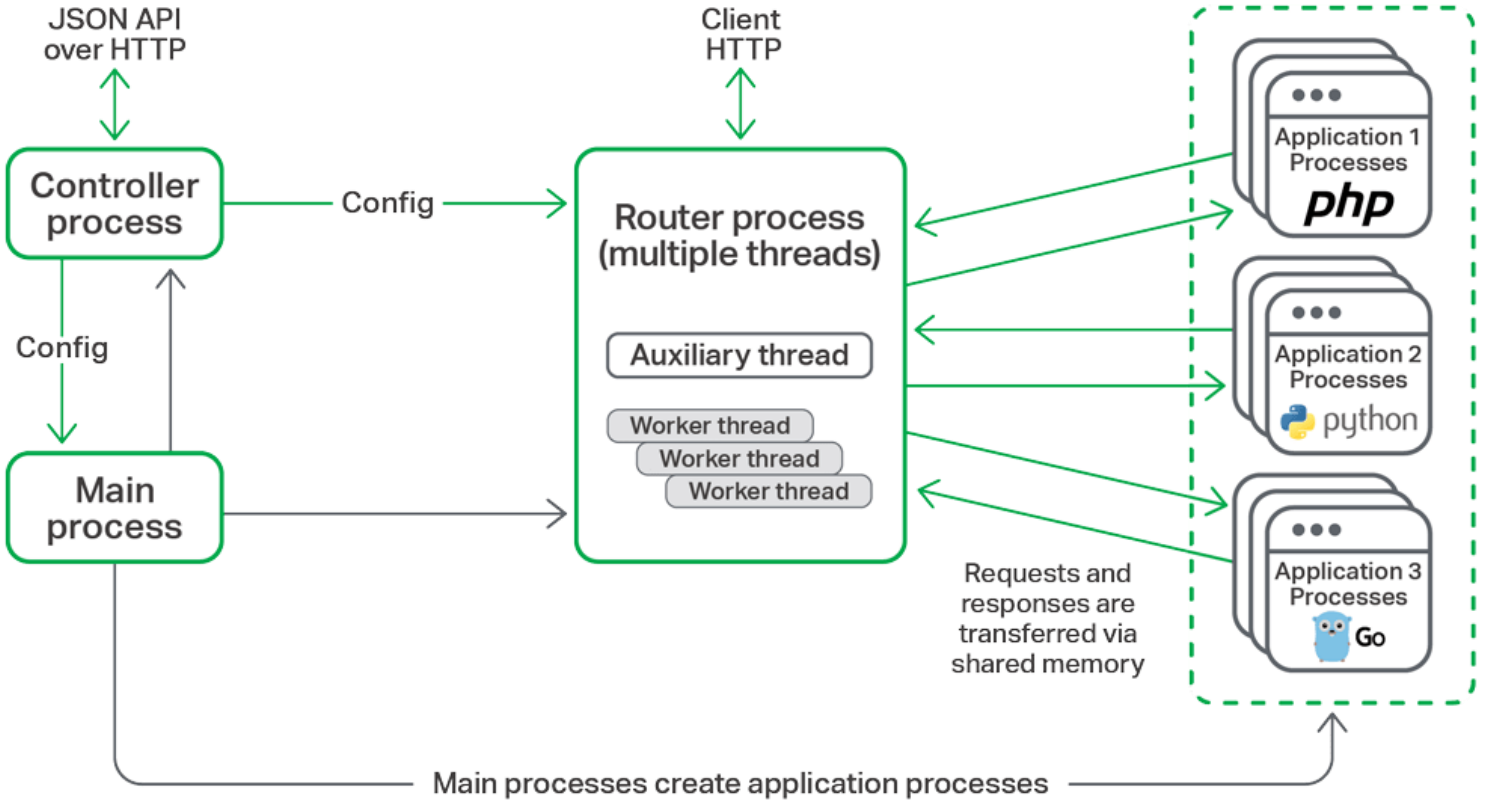
- 作为动态Web容器，调用基于多种编程语言编写的Web接口，构建动态Web Server
- 通过HTTP+ RESTful JSON API动态修改进程的配置文件，且修改过程中不会干扰到正常的请求处理流程
- 同时具备反向代理、负载均衡、静态资源服务的能力
- 对HTTP请求支持复杂的路由匹配，包括正则表达式
- 支持动态调整的负载均衡策略

UNIT支持的操作系统

- Amazon Linux, Amazon Linux 2
- CentOS 6, 7, 8
- Debian 9, 10
- Fedora 29, 30, 31, 32, 33
- RHEL 6, 7, 8
- Ubuntu 16.04, 18.04, 19.10, 20.04, 20.10

UNIT的启动与停止

- 启动参数
  - --no-damon
  - --version
  - --control
  - --group
  - --name
  - --log
  - --module
  - --pid
  - --state
- 停止进程
  - pkill unitd



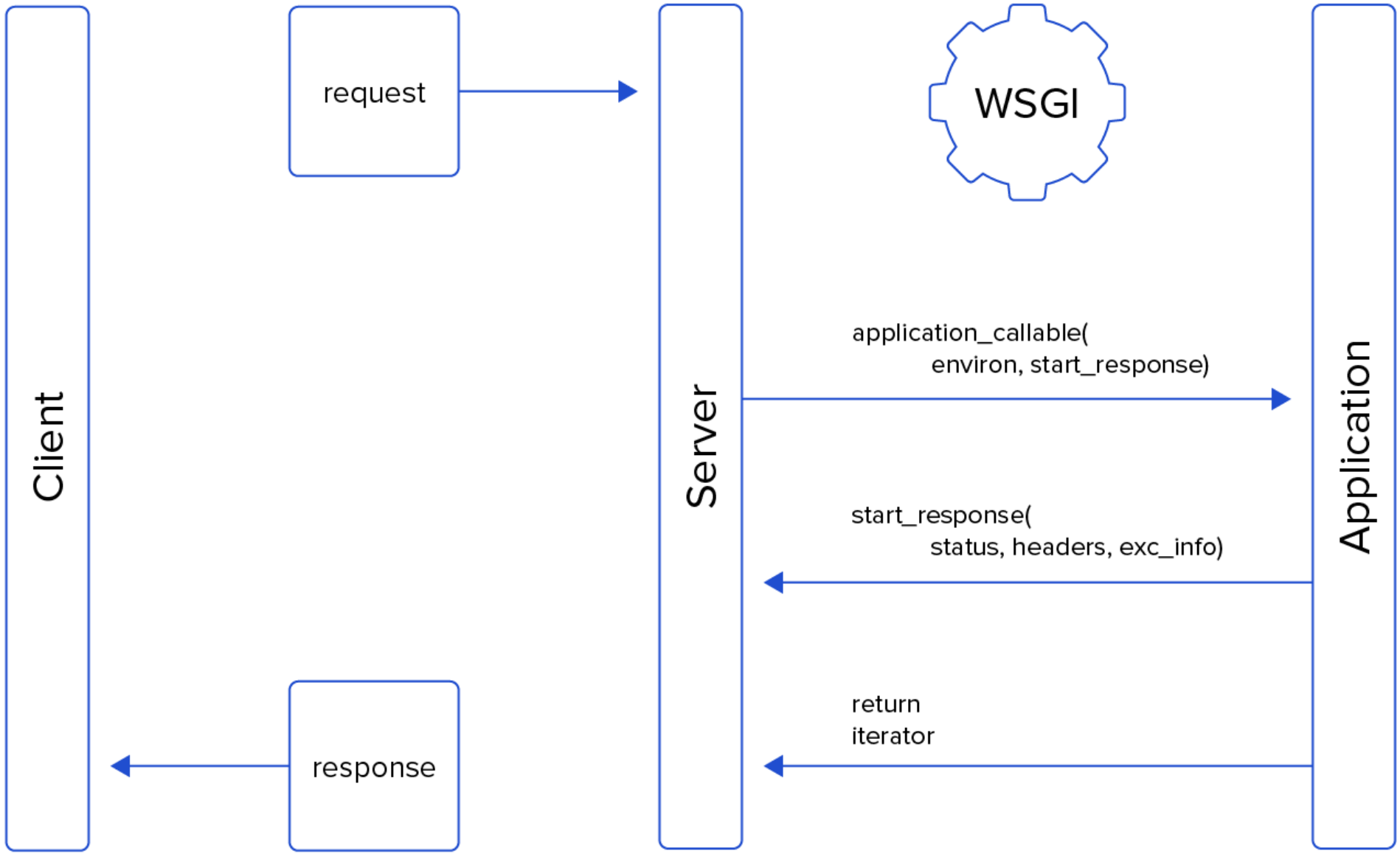
Unit编译&运行

- git clone https://github.com/nginx/unit
- 编译前的依赖软件
  - yum install gcc gcc-c++ make -y
  - yum install openssl-devel –y
  - yum install pcre2-devel –y
- configure参数
  - 编译选项：
    - --cc --cc-opt --ld-opt
    - --openssl
  - 运行选项：
    - --user --group
    - --debug
    - --no-ipv6 --no-unix-sockets
  - 目录选项
    - --prefix --binddir –sbindir --control –incdir --libdir --log --module --pid --state --tmp

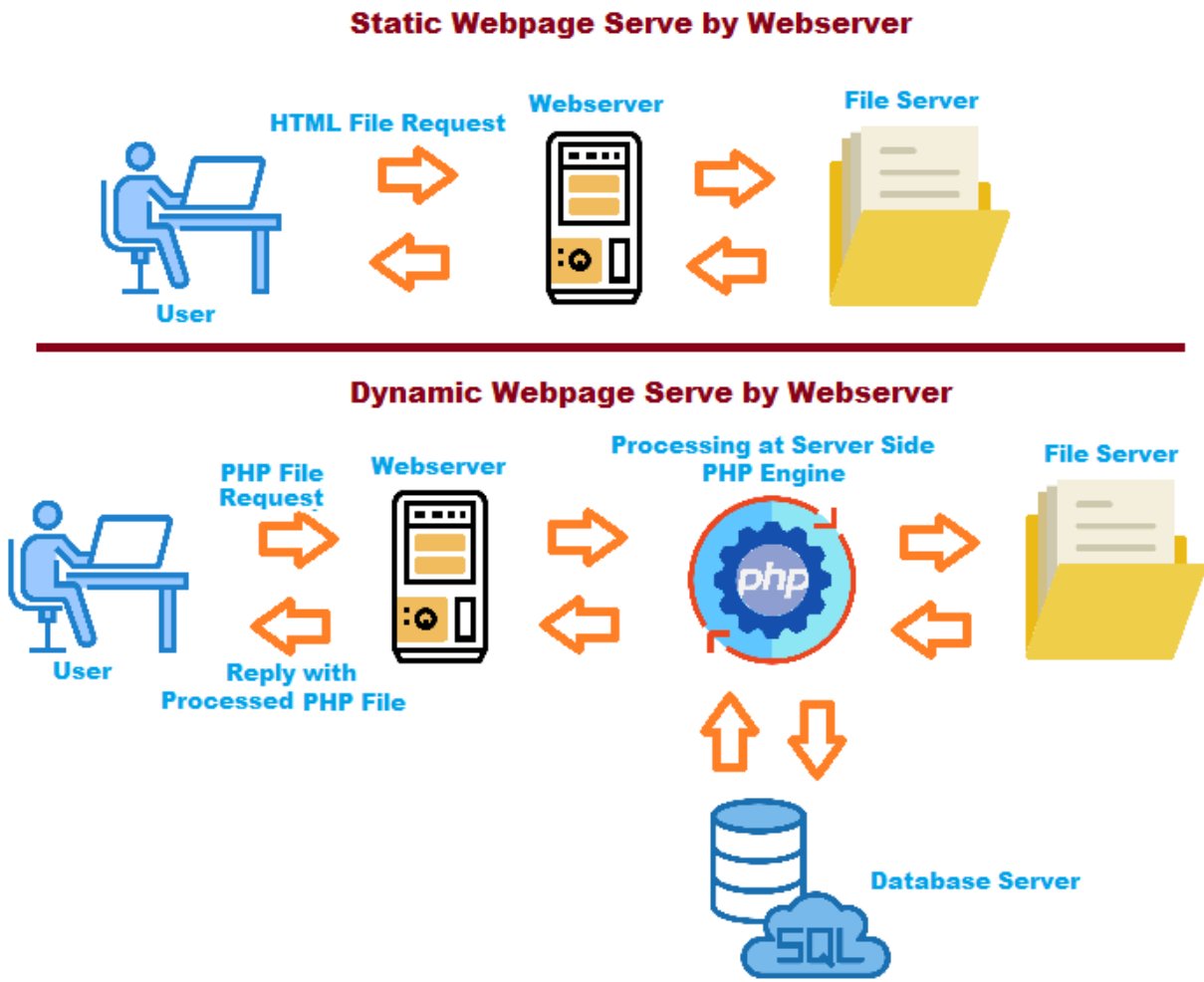
Unit 各语言的编译

- python： yum install python-devel –y
  - configure python
- php： yum install php-devel php-embedded –y
  - configure php
- go：yum install golang –y
  - configure go
- perl：yum install perl-devel perl-libs –y
  - configure perl
- ruby：yum install ruby-devel –y
  - configure ruby
- java：yum install java-1.8.0-openjdk-devel –y
  - configure java
- nodejs：yum install nodejs –y
  - npm install -g node-gyp
- WebAssembly：内置在unit进程中

Python WSGI


Client
request
Server
WSGI
Application
application_callable(
environ, start_response)
start_response(
status, headers, exc_info)
return
iterator
response

PHP Webserver

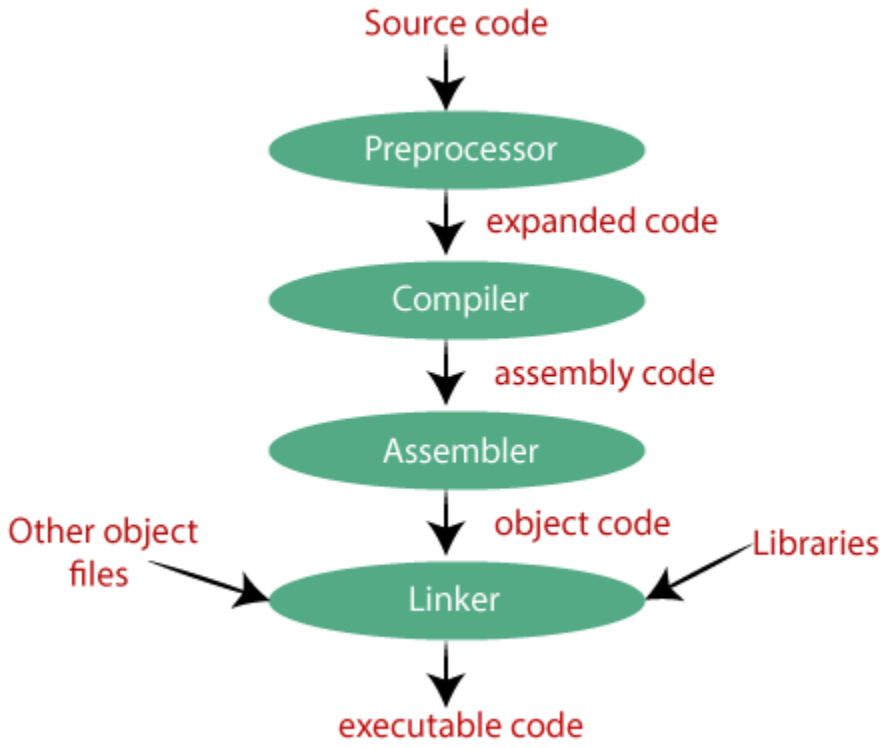


探索NGINX UNIT

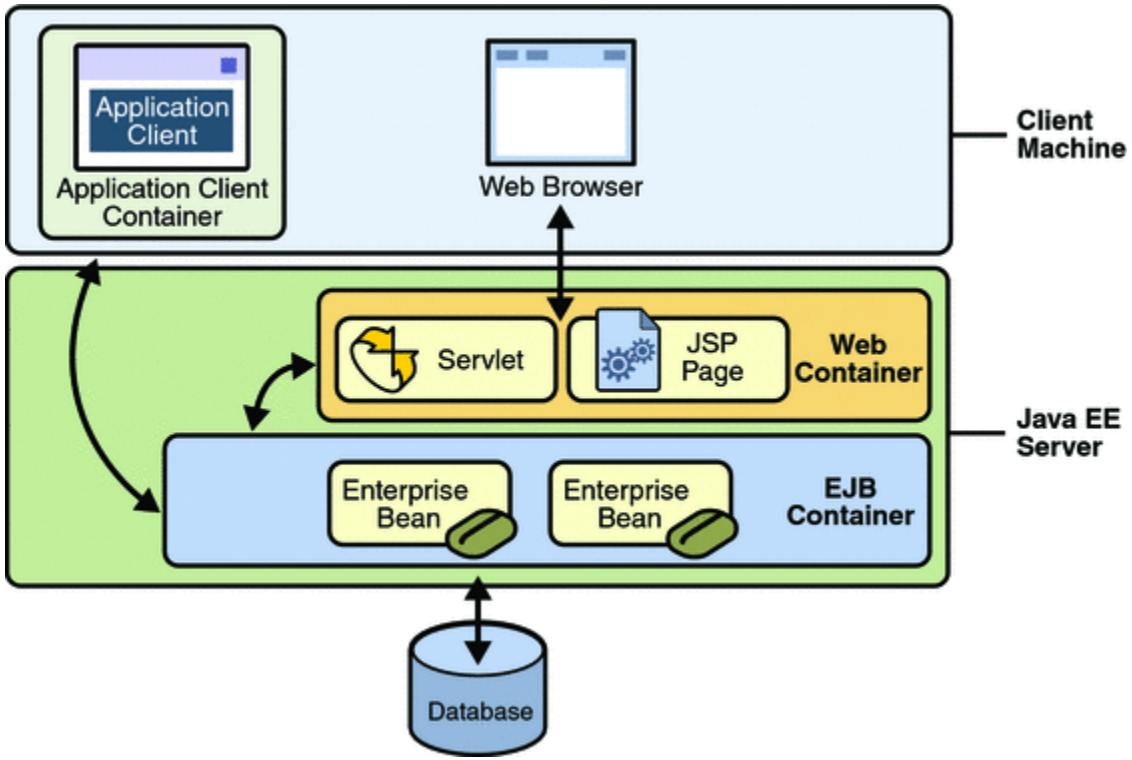
- 用Unit实现应用的动态配置
- **Unit的负载均衡配置**
- Unit架构设计
- Unit源代码解读

WebAssembly与汇编语言

- 预编译
- 编译
- 汇编
- 链接



Java


Application Client
Application Client Container
Web Browser
Client Machine
Servlet
JSP Page
Web Container
Java EE Server
Enterprise Bean
Enterprise Bean
EJB Container
Database

applications的通用选项

选项	描述
type (必选)	应用的编程语言，包括Go 、Node.js、java、perl、php、python、ruby。可以携带版本，例如 "type": "python 3", "type": "python 3.4"，"type": "python 3.4.9rc1".
limits	限制用户请求。如timeout可以限制每个HTTP请求的处理超时时间，一旦超过timeout秒数，就会自动返回HTTP错误。而requests参数限制unit进程处理请求的最大个数，一旦超出了requests值，进程就会重启，以解决可能的内存泄露等问题。
processes	应用进程的数量，设置为整数时表示静态进程数量（默认为1个）；也可以设置为动态进程数量，包括max指定最大进程数量，spare指定最小进程数量，当进程数大于spare时，若1个进程在idle_timeout秒没有收到请求，就会被unit进程关闭
working_directory	应用进程的工作目录，默认为unitd进程的工作目录
user	执行应用进程的系统用户。如果未设置，会使用编译时--user指定的用户（默认为nobody），或者unitd启动时--user指定的用户
group	与user类似，指定应用进程的所属用户组
environment	传递给应用的环境变量

UNIT Python配置

- **type: python**
  - module：wsgi或者asgi所在文件
  - callable：被调用的wsgi/asgi函数
  - home：python的virtual env虚拟环境地址
  - path：python库的查询路径，它会添加到sys.path中
  - protocol：wsgi或者asgi
  - threads：线程数
  - thread_stack_size：线程栈大小

UNIT Java配置

- **type: java**
  - webapp：war打包文件或者Web目录文件的放置路径
  - classpath：jar包等依赖库的路径
  - options：JVM启动参数
  - threads：线程数，默认值1
  - thread_stack_size：线程栈大小

UNIT PHP配置

- **type: php**
  - root：.php文件的根目录
  - index：当script未指定时，如果URL是/，则返回index.html
  - options：指定php.ini配置文件的位置与选项
    - file：php.ini的位置，例如/etc/php.ini
    - admin：设置PHP_INI_SYSTEM模式中的指令，例如memory_limit、variables_order、 expose_php等
    - user：设置PHP_INI_USER模式中的指令，例如display_errors
  - targets：如果php中有多种处理规则，可通过targets配置并由pass路由
  - script：指定处理所有HTTP请求的PHP文件

UNIT外部进程

- type: external
  - 编程语言
    - golang
    - nodejs
    - webassembly
  - 配置项
    - executable：可执行文件，包含路径
    - arguments：可执行文件的运行参数

UNIT ruby与perl配置

- **type: ruby**
  - script：Rack脚本路径
  - threads：线程数，默认值1
- **type: perl**
  - script：PSGI脚本路径
  - threads：线程数，默认值1
  - thread_stack_size：线程栈大小

hide progress	verbose	extra info	output	timeout
-s	**-v** **--trace-ascii <file>**	**-w "format"**	**-O** **-o <file>**	**-m <seconds>**
POST	**POST encoded**	**multipart formpost**	**PUT**	**HEAD (ers too)**
-d "string" **-d @file**	**--data-urlencode "[name]=val"**	**-F name=value** **-F name=@file**	**-T <file>**	**-I** **-i**
custom method	**read cookiejar**	**write cookiejar**	**send cookies**	**user-agent**
-X "METHOD"	**-b <file>**	**-c <file>**	**-b "c=1; d=2"**	**-A "string"**
proxy	**add/remove headers**	**custom address**	**smaller data**	**insecure HTTPS**
-x <host:port>	**-H "name: value"** **-H "name:"**	**--resolve <host:port:addr>**	**--compressed**	**-k**
Basic auth	**follow redirects**	**parallel**	**generate code**	**list options**
-u user:passwd	**-L**	**-Z**	**--libcurl <file>**	**--help**

curl的常用命令选项

- -X：指定HTTP方法
- -d：传递HTTP请求包体
- --data-binary：以二进制文件作为HTTP请求包体
- --unix-socket：使用本机上的unix socket
- --limit-rate：限速下载速度，字节/秒（可使用k/m等单位）
- -H：增加HTTP请求头部
- -I：查看HTTP响应头部
- -l：指定TLS版本
- -G：给GET请求携带-d中的包体

修改配置的4个HTTP RESTFul API

- **操作对象：URI**
  - /config
    - /listeners
    - /applications
    - /routes
    - /upstreams
    - /settings
    - /access_log
  - /certificates
- **Body格式：JSON**
- **Method方法**
  - GET：获取配置
  - PUT：更新配置中的JSON对象
  - POST：更新配置中的JSON数组
  - DELETE：删除配置中的JSON对象

listeners配置

- **pass**
  - applications/xxx
  - applications/PHPxxx/target
  - routes
  - upstreams
- **tls**
  - certificates

探索NGINX UNIT

- 用Unit实现应用的动态配置
- Unit的负载均衡配置
- Unit架构设计
- Unit源代码解读

Route路由配置中的动作

- **action**
  - pass：类似listerners中的pass
    - 可以指向applications、其他routes或者upstreams
  - share：静态文件映射，访问目录时自动获取index.html文件
    - 返回ETag指纹，格式为`%MTIME_HEX%-%FILESIZE_HEX%`
    - MIME Types映射由/config/settings/http/static/mime_types控制
    - fallback：share文件找不到时，可由fallback处理。可以嵌套。
  - proxy：HTTP协议的反向代理
    - 目前不能指向upstreams，不支持localhost
  - return：返回不含包体的HTTP响应
    - location：当响应码为重定向请求时，指定重定向URL
- **match**

Route路由配置中的匹配规则

- action
- match：通配符或者正则匹配，支持与、或、非关系
  - IP地址：
    - destination、 source
  - 名值对
    - arguments、 cookies、 headers
  - host
  - uri
  - method
  - scheme：仅http或者https两个值

请求属性与UNIT变量

- arguments
- cookies
- destination
- headers
- scheme
- source
- host
  - 请求头部中的HOST头部，会转换为小写字母，且不包含端口号
- method
  - 请求行中的方法，例如GET或者POST
- uri
  - 不包括?后参数部分的URI，且已经执行过URI百分号解码

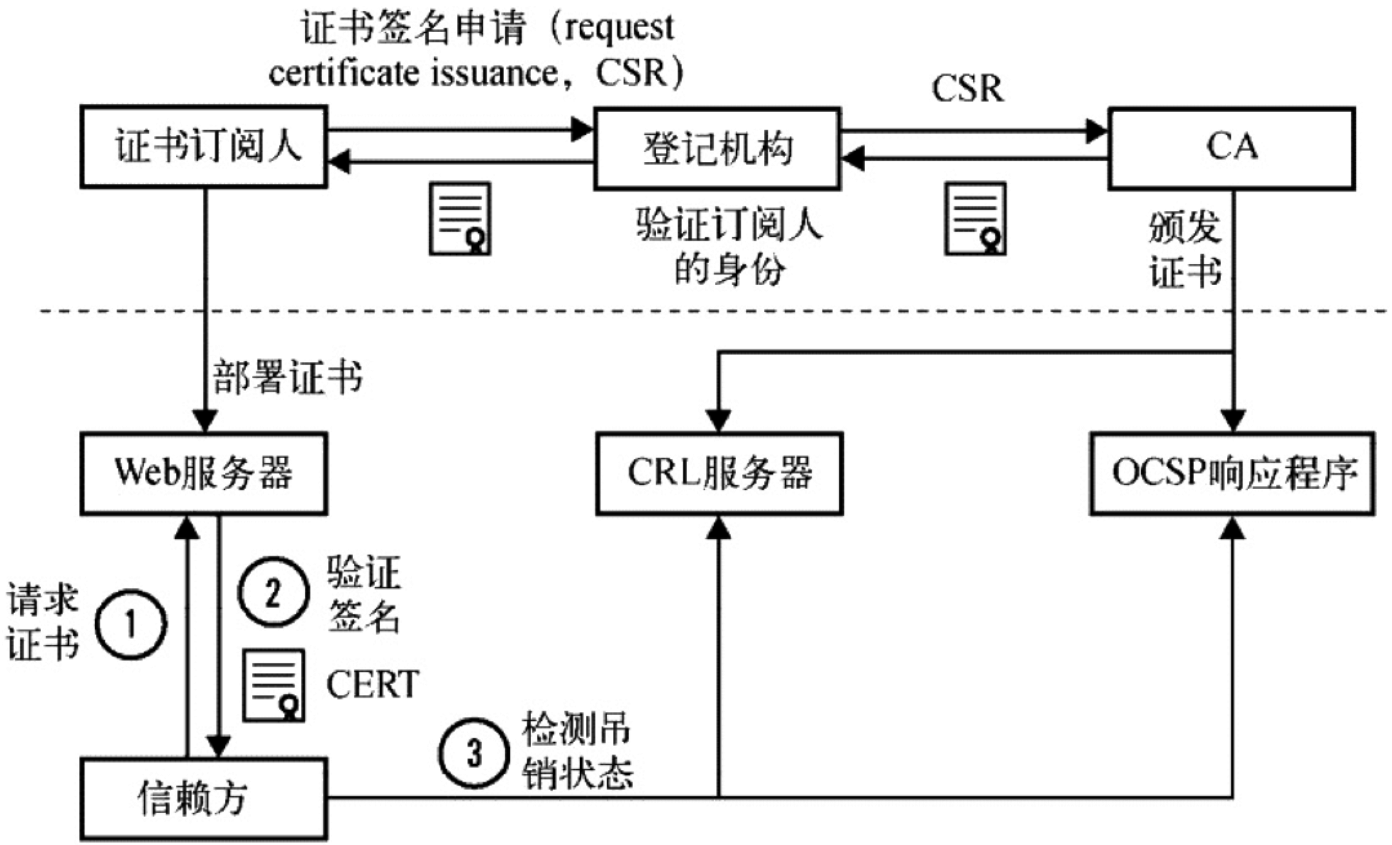
UNIT配置文件的操作

- **操作对象**
  - URL：只能是JSON对象的名字，或者是JSON数组的序号
- **操作方法**
  - GET：查询JSON配置
  - POST：向JSON数组中添加新的JSON对象
  - PUT：覆盖（幂等方法）JSON对象的值
  - DELETE：删除JSON对象
- **数据格式**
  - JSON

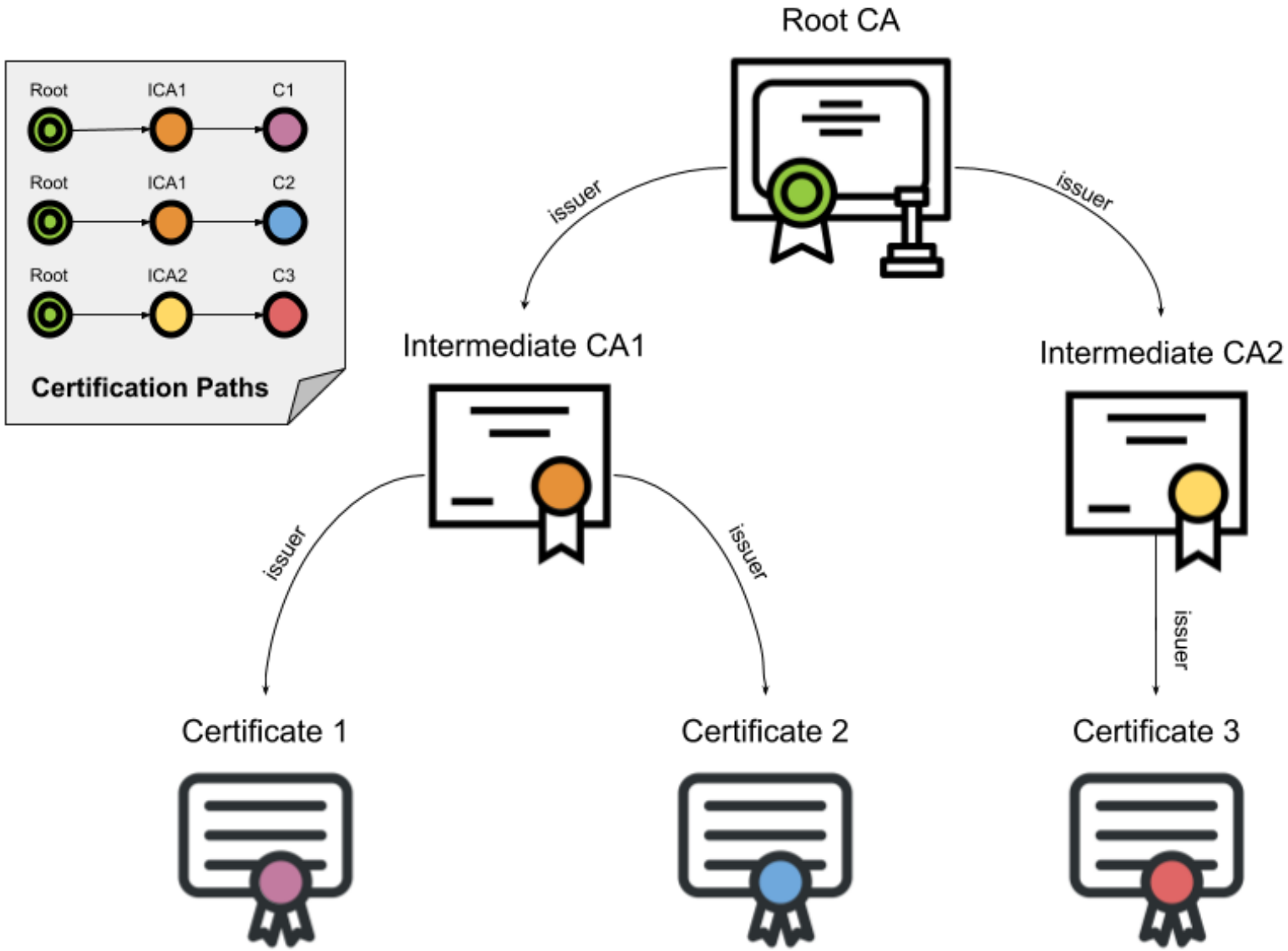
settings中的HTTP通用配置

- **header_read_timeout：默认值30秒**
  - 读取HTTP请求头部的最大时间，超时后返回408错误码
- **body_read_timeout：默认值30秒**
  - 与客户端之间连续2次TCP读操作间的最大间隔，超时后返回408错误码
- **send_timeout：默认值30秒**
  - 与客户端间连续2次TCP写操作的最大间隔秒数，超时后直接关闭TCP连接
- **idle_timeout：默认值180秒**
  - KeepAlive长连接的最大空闲秒数，超时后返回408错误码并关闭TCP连接
- **max_body_size：默认值8MB**
  - 客户端HTTP请求包体的最大长度，超出后返回413错误码
- **static：目前仅含有mine_types成员**
  - 作为静态资源服务时，通过mine_types依据文件名或者后缀名，设置content-type头部
- **discard_unsafe_fields：默认值true**
  - true表示HTTP头部仅支持字母、数字和-号，而false则支持RFC7230中的所有合法头部

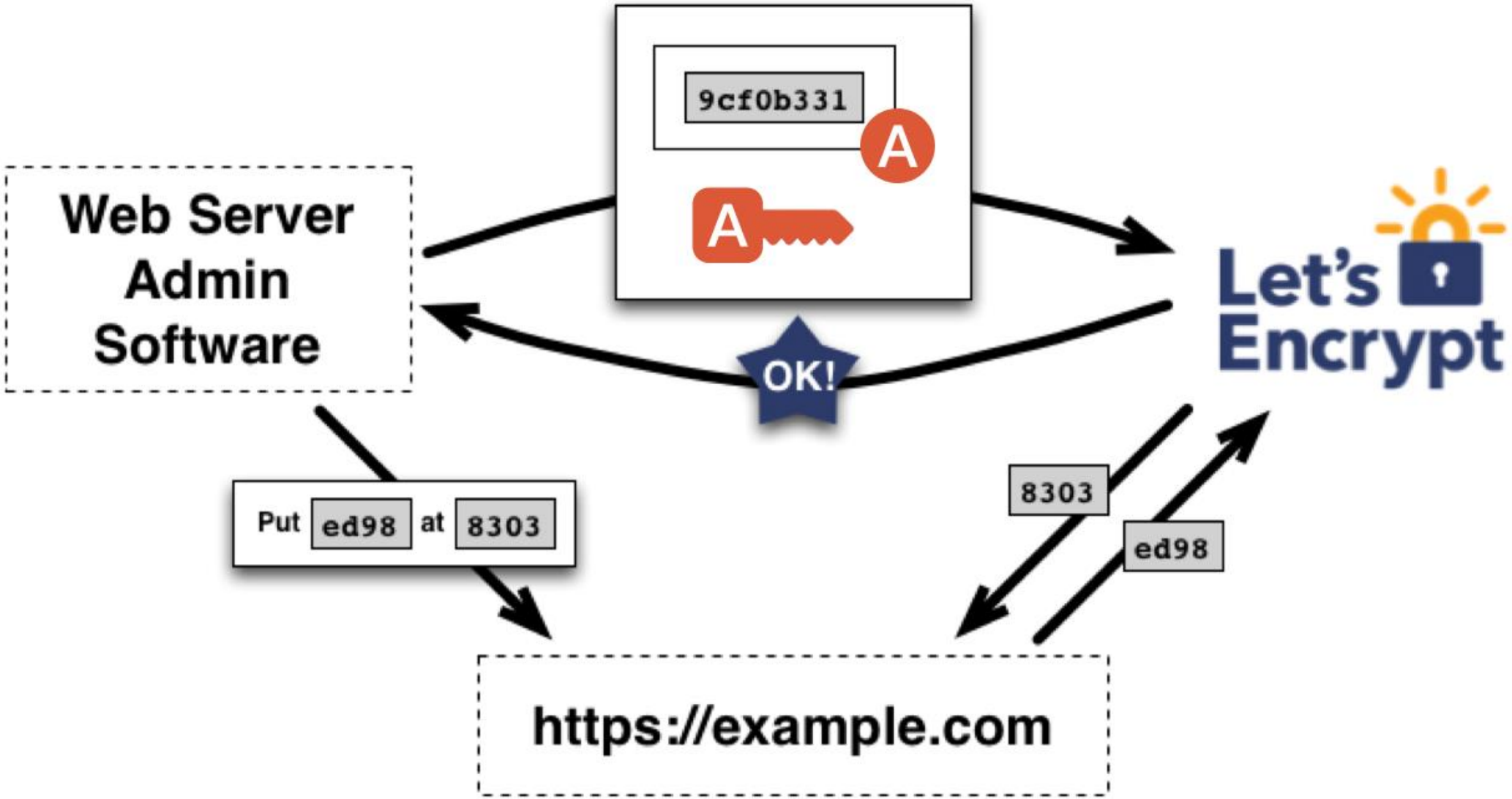
PKI公钥基础设施


证书签名申请（request certificate issuance，CSR）
证书订阅人
登记机构
CSR
CA
验证订阅人的身份
颁发证书
部署证书
Web服务器
CRL服务器
OCSP响应程序
请求证书
1
2
验证签名
CERT
3
检测吊销状态
信赖方

三级证书体系


Root
ICA1
C1
Root
ICA1
C2
Root
ICA2
C3
Certification Paths
Root CA
issuer
issuer
Intermediate CA1
Intermediate CA2
issuer
issuer
issuer
Certificate 1
Certificate 2
Certificate 3

Lets Encrypt



UNIT TLS/SSL配置

- certificates
  - bundle
    - key
    - chain
      - subject/issuer
        - common_name
        - organization
        - country
        - alt_names
        - state_or_province
      - validity
        - since
        - until

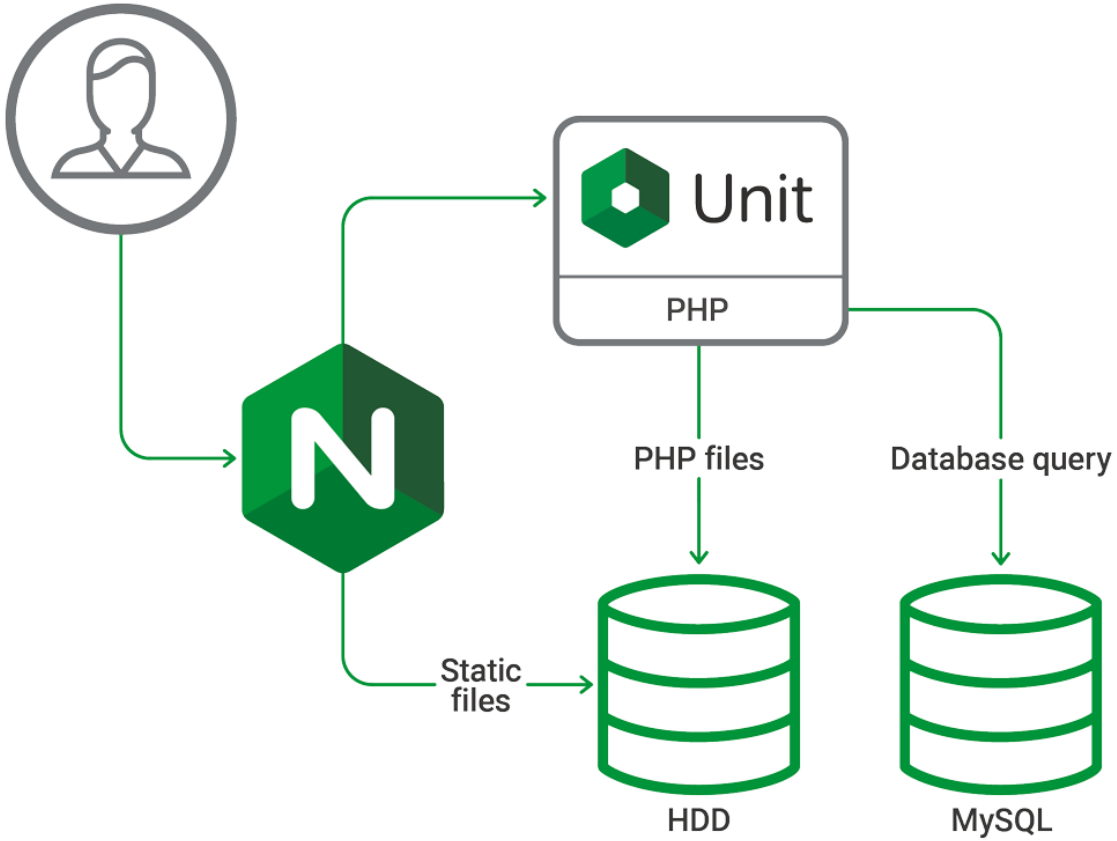
certbot + UNIT

```
# ./configure --prefix=/home/unit/ --debug --openssl

# certbot certonly --standalone  -d unit.taohui.tech
# openssl  asn1parse  -i  -in  /etc/letsencrypt/live/unit.taohui.tech/fullchain.pem

# cat /etc/letsencrypt/live/unit.taohui.tech/fullchain.pem \ >
/etc/letsencrypt/live/unit.taohui.tech/privkey.pem > bundle.pem

# curl -X PUT --data-binary @bundle.pem --unix-socket ../control.unit.sock
http://localhost/certificates/certbot1

# curl -X PUT --data-binary  \ '{"pass": "applications/wsgi", "tls": {"certificate": "certbot1"}}' --
unix-socket ../control.unit.sock 'http://localhost/config/listeners/*:9443'
```

探索NGINX UNIT

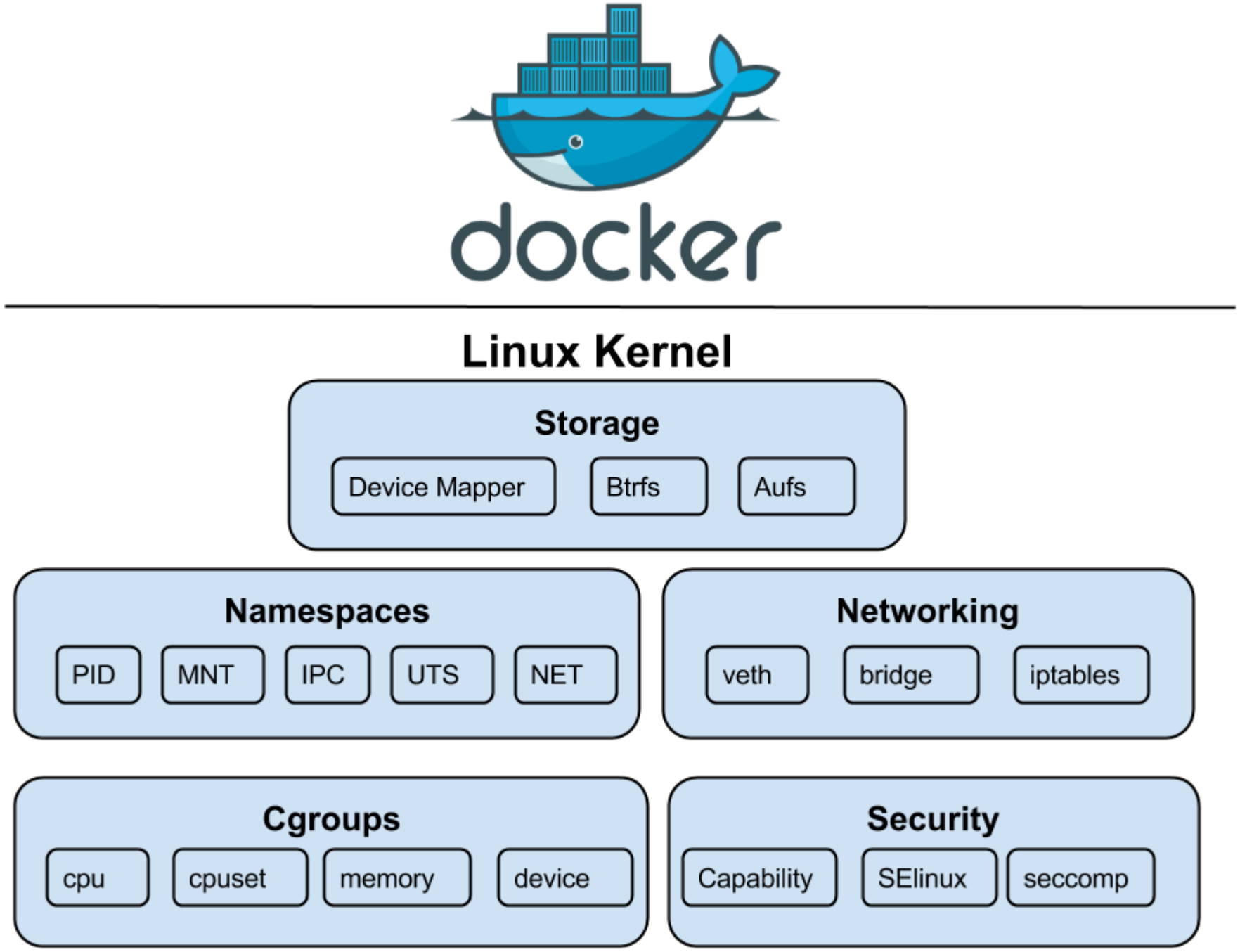
- 用Unit实现应用的动态配置
- Unit的负载均衡配置
- Unit架构设计
- **Unit源代码解读**

Nginx与Unit的协作



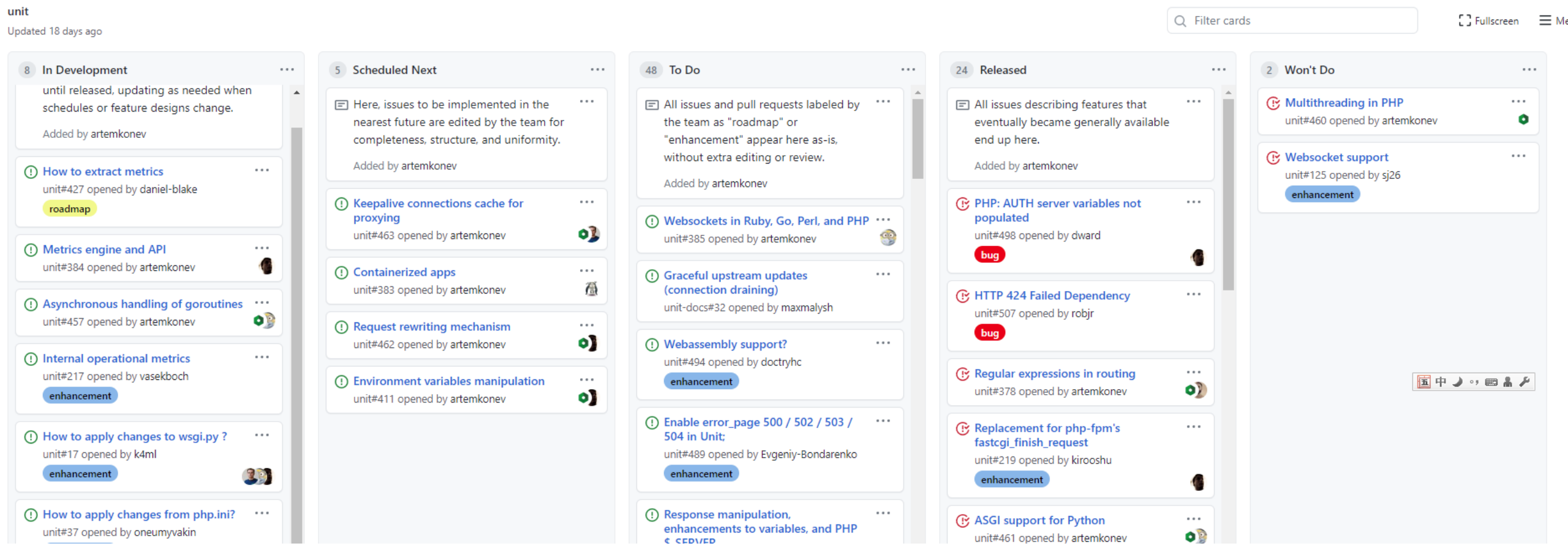
资源隔离

- namespaces
  - cgroup
  - credential
  - mount
  - network
  - pid
  - uname
- uidmap/gidmap
  - container
  - host
  - size
- rootfs
- automount



Nginx Unit Road Map

- https://github.com/orgs/nginx/projects/1

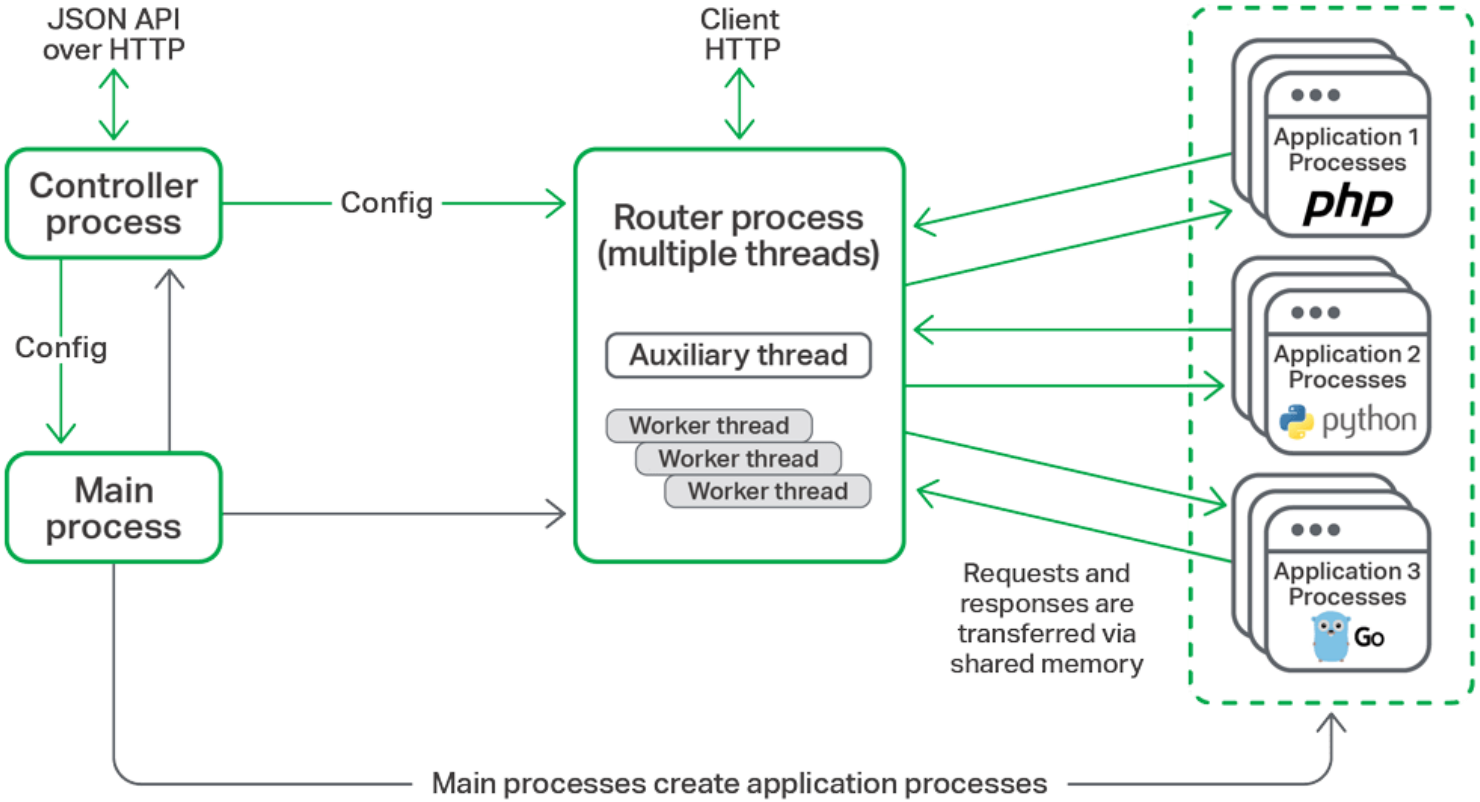


UNIT生态

- unitd进程
- 多语言应用
  - 动态语言网络库
    - golang：unit.nginx.org/go库
    - nodejs：unit-http库
  - C语言与动态语言的交互调用
    - WebAssembly：libunit.a
    - java：java.unit.so
      - tomcat-servlet-api-9.0.39.jar tomcat-el-api-9.0.39.jar tomcat-jsp-api-9.0.39.jar tomcat-jasper-9.0.39.jar tomcat-jasper-el-9.0.39.jar tomcat-juli-9.0.39.jar tomcat-api-9.0.39.jar tomcat-util-scan-9.0.39.jar tomcat-util-9.0.39.jar ecj-3.23.0.jar jetty-util-9.4.33.v20201020.jar jetty-server-9.4.33.v20201020.jar jetty-http-9.4.33.v20201020.jar classgraph-4.8.90.jar
    - python：python.unit.so python2.7
    - php：php.unit.so php5
    - perl：perl.unit.so
    - ruby：ruby.unit.so

```
typedef enum {
    NXT_APP_EXTERNAL,
    NXT_APP_PYTHON,
    NXT_APP_PHP,
    NXT_APP_PERL,
    NXT_APP_RUBY,
    NXT_APP_JAVA,

    NXT_APP_UNKNOWN,
} nxt_app_type_t;
```

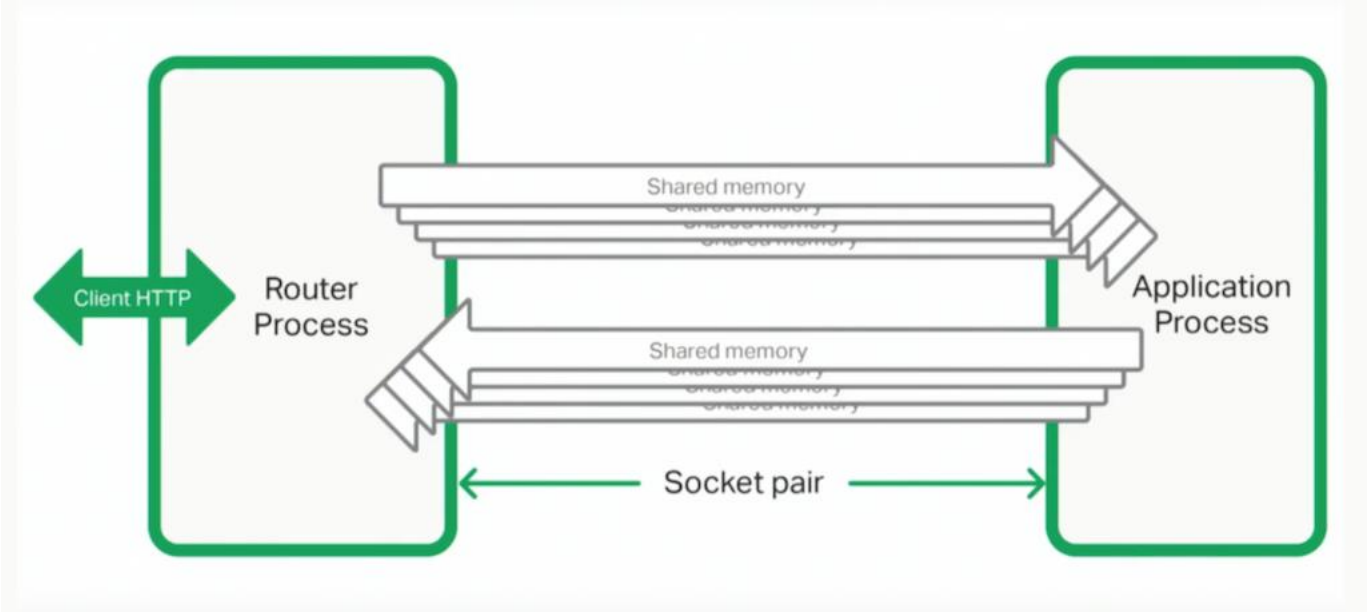
Unit进程架构

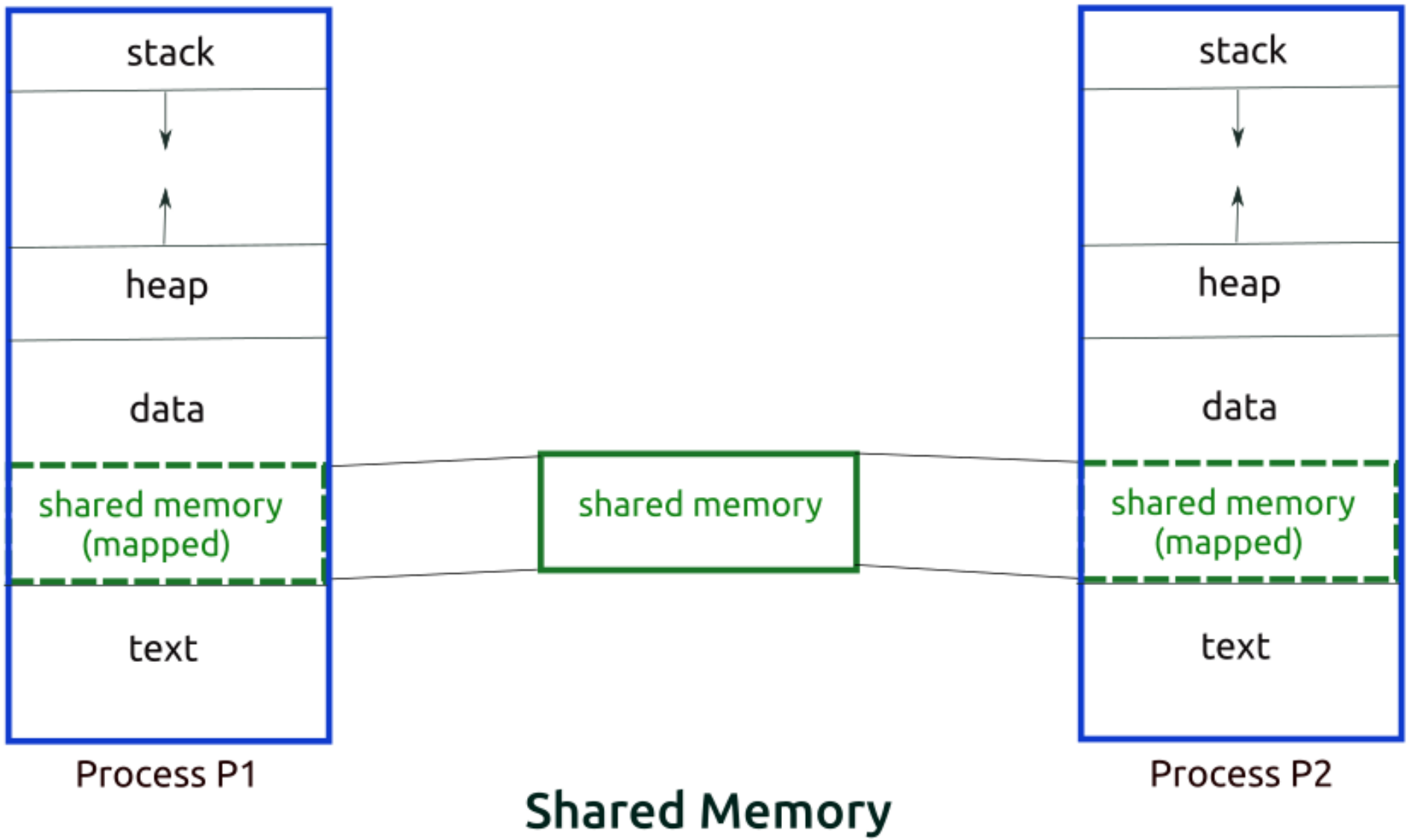

JSON API
over HTTP
Client
HTTP
Controller
process
Config
Router process
(multiple threads)
Auxiliary thread
Worker thread
Worker thread
Worker thread
Config
Main
process
Application 1
Processes
php
Application 2
Processes
python
Application 3
Processes
Go
Requests and
responses are
transferred via
shared memory
Main processes create application processes

UNIT进程的核心流程

- **启动次序**
  - main：管理子进程的父进程
    - controller：监听--control socket指定的端口
    - router：监听config/listerners里指定的端口
    - 应用进程：监听socker pair端口
      - UNIT进程/外部进程
- **处理请求流程**
  - controller：REST json请求
  - router：HTTP请求
  - 应用进程：router转发的HTTP请求

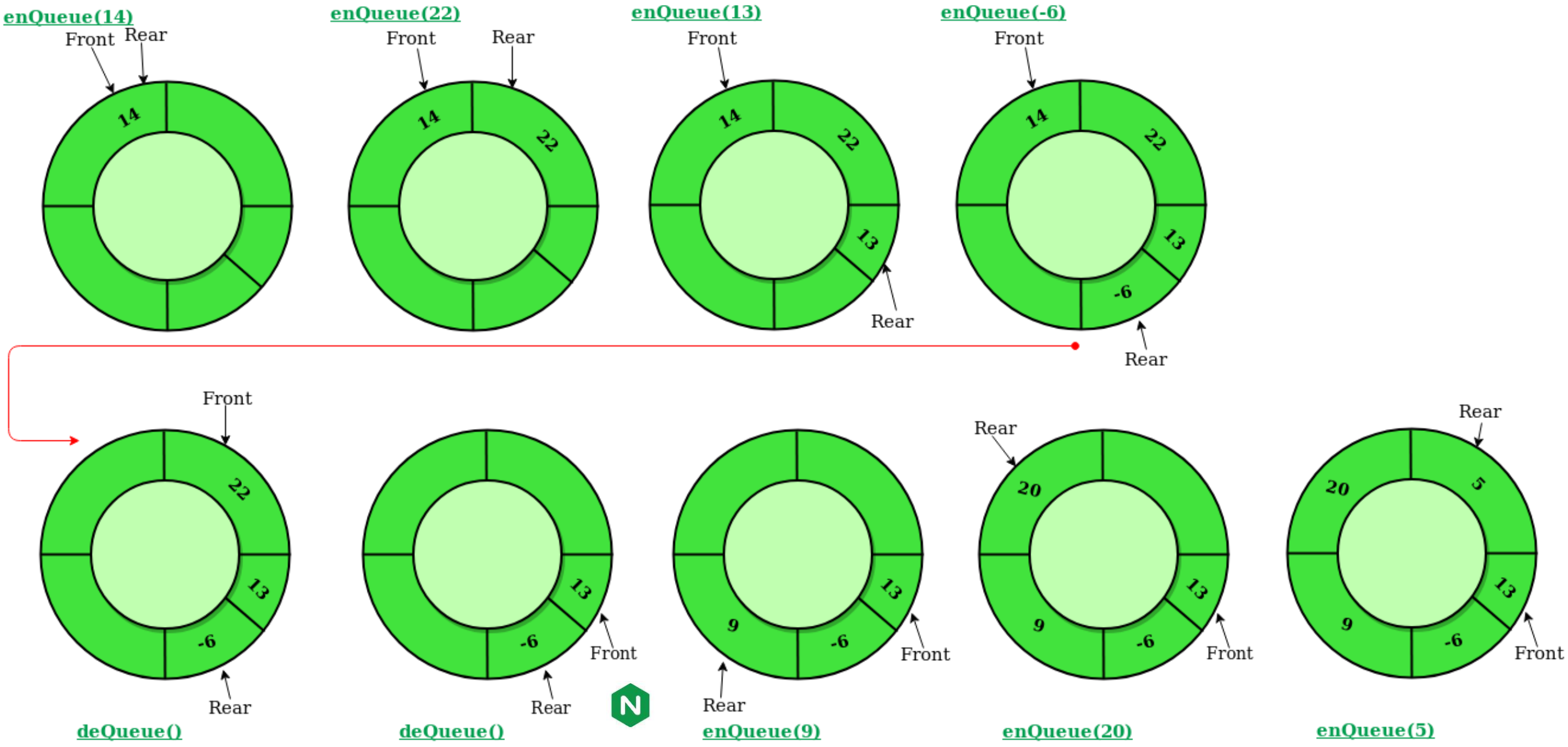
共享内存+socket pair：进程间通讯





Shared Memory

Numeric Naive Circular Queue



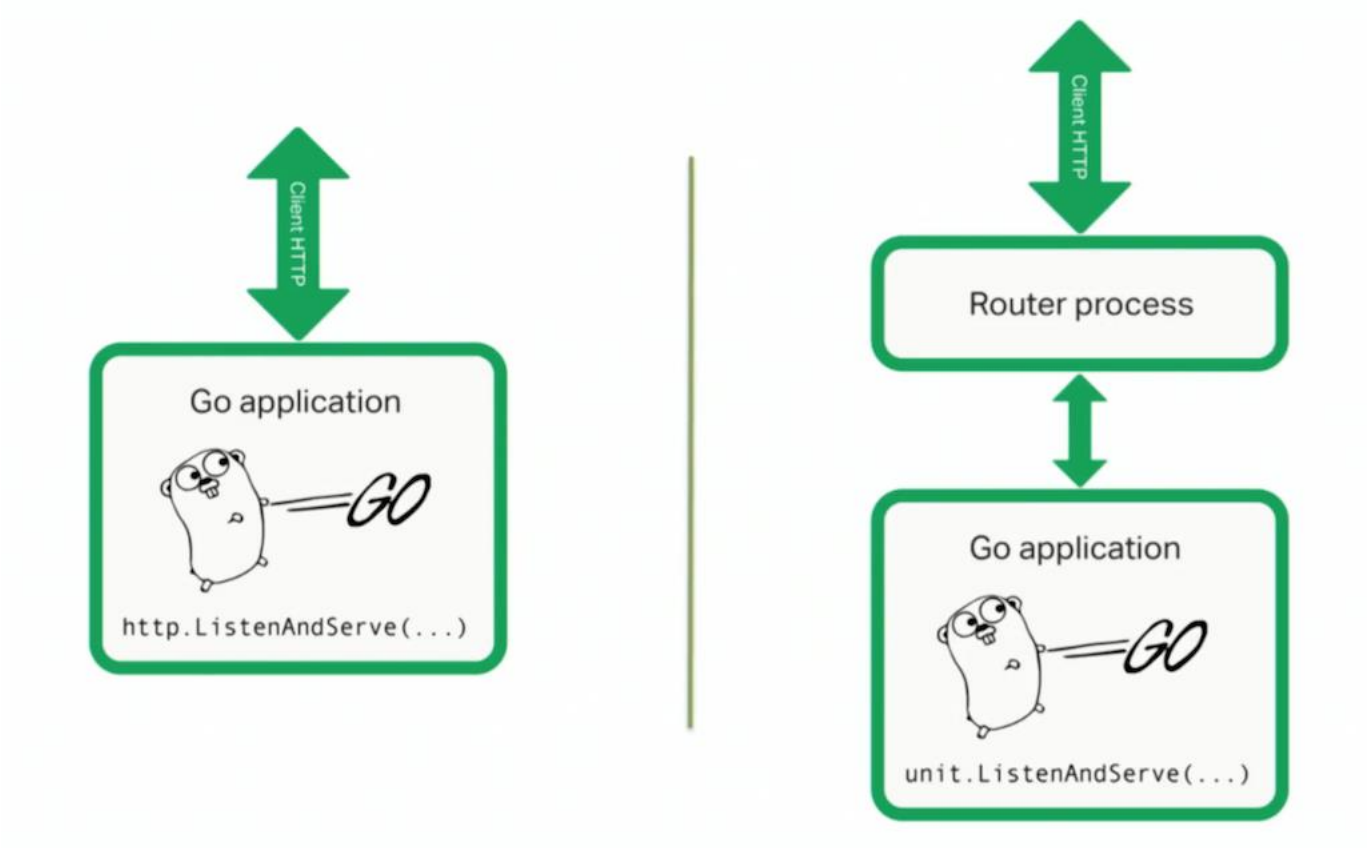
NNCQ循环队列核心数据结构

```
#define NXT_PORT_QUEUE_MSG_SIZE   31
#define NXT_NNCQ_SIZE   16384
#define NXT_PORT_QUEUE_SIZE       NXT_NNCQ_SIZE
typedef uint32_t nxt_nncq_atomic_t;

typedef struct {
    nxt_nncq_atomic_t  head;
    nxt_nncq_atomic_t  entries[NXT_NNCQ_SIZE];
    nxt_nncq_atomic_t  tail;
} nxt_nncq_t;

typedef struct {
    uint8_t   size;
    uint8_t   data[NXT_PORT_QUEUE_MSG_SIZE];
} nxt_port_queue_item_t;

typedef struct {
    nxt_nncq_atomic_t       nitems;         //队列元素计数
    nxt_nncq_t              free_items;     //空闲元素构成的链表
    nxt_nncq_t              queue;          //有效元素构成的链表
    nxt_port_queue_item_t   items[NXT_PORT_QUEUE_SIZE];    //所有分配的队列元素
} nxt_port_queue_t;
```

用户HTTP消息同步流程

- **router进程**
  - 从TCP连接中接收、解析、Match HTTP请求
  - 将HTTP请求序列化至共享内存块中`nxt_router_prepare_msg`
  - 将共享内存中的请求通知到应用进程`nxt_app_queue_send`
    - 构造描述信息，插入NNCQ队列中
  - 循环读取NNCQ队列中的HTTP响应
  - 将HTTP响应序列化后发送到TCP连接上
- **application进程**
  - 从NNCQ队列中读取到描述信息`nxt_unit_read_buf`
    - 可选从`socket pair`中读取描述信息
  - 从共享地址中，解析、处理HTTP请求`nxt_unit_process_msg`
    - 对于`HTTP`请求，需要在`NNCQ`队列中发送`ACK`确认消息
  - 从C中调用具体语言的HTTP回调函数`nxt_unit_process_ready_req`
    - 生成响应后，将HTTP响应发送给router进程`nxt_unit_response_send`
      - 序列化HTTP响应至共享内存块中
      - 将描述信息插入NNCQ队列

外部进程（Go、NodeJS）的通讯框架



C进程与Python代码间的相互调用

- **C进程调用Python代码**
  1. 选择执行线程，设置好sys.path等系统环境
  2. 导入Python代码wsgi.py
  3. 找到application方法
  4. 构造HTTP请求为入参environ
  5. 构造发送HTTP响应的start_response方法作为入参
  6. 调用application方法，并处理其返回值
  7. 发送HTTP响应到route进程
  8. 销毁当前上下文及其使用资源
- **Python代码调用C代码**
  - 有包体：缓存HTTP头部,交由上述第7步发送（PEP 3333WSGI规范）
  - 空包体：调用`nxt_unit_response_send`发给route进程

UNIT WebAssembly启动流程

- UNIT启动流程
  - 分配nxt_unit_init结构体
    - 设置请求的回调函数request_handler
  - 执行nxt_unit_init(init)
    - `nxt_unit_ctx_t *nxt_unit_init(nxt_unit_init_t *);`
  - 执行nxt_unit_run(ctx)
    - `int nxt_unit_run(nxt_unit_ctx_t *);`
  - 执行nxt_unit_done(ctx)
    - `void nxt_unit_done(nxt_unit_ctx_t *);`

```
struct nxt_unit_init_s {
    void                    *data;
    void                    *ctx_data;
    int                     max_pending_requests;

    uint32_t                request_data_size;
    uint32_t                shm_limit;

    nxt_unit_callbacks_t    callbacks;

    nxt_unit_port_t         ready_port;
    uint32_t                ready_stream;
    nxt_unit_port_t         router_port;
    nxt_unit_port_t         read_port;
    int                     log_fd;
};
```

UNIT WebAssembly处理请求流程

- nxt_unit_response_init方法初始化HTTP响应
  - `int nxt_unit_response_init(nxt_unit_request_info_t *req, uint16_t status, uint32_t max_fields_count, uint32_t max_fields_size);`
- nxt_unit_response_add_field添加HTTP头部
  - `int nxt_unit_response_add_field(nxt_unit_request_info_t *req, const char* name, uint8_t name_length, const char* value, uint32_t value_length);`
- nxt_unit_response_add_content添加HTTP包体
  - `int nxt_unit_response_add_content(nxt_unit_request_info_t *req, const void* src, uint32_t size);`
- nxt_unit_response_send发送HTTP响应
  - `int nxt_unit_response_send(nxt_unit_request_info_t *req);`

谢谢