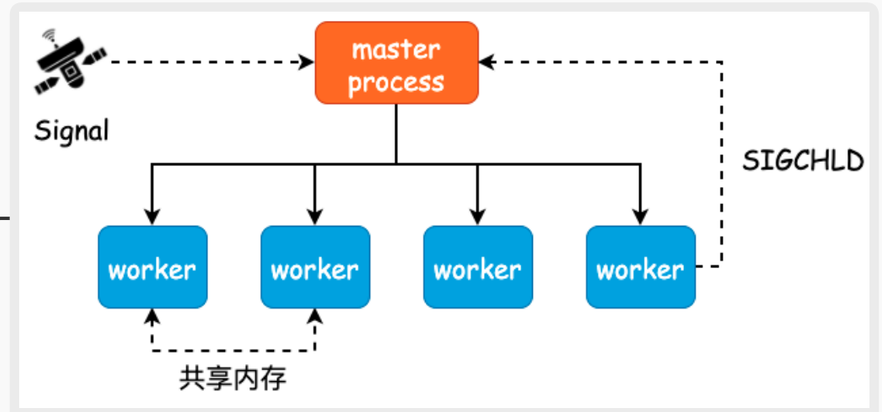
Nginx

基础模型

进程模型



关键点

1. Nginx 采用 master-worker 这种多进程的方式运行，master 进程并不负责实际的业务处理，而是作为管理者的角色运行
2. master 进程和 worker 进程之间采用信号的方式进行通信，也就是当子进程退出时，父进程将收到 SIGCHLD 信号。此时 master 进程为了维持 worker 的声明数量，将会再 fork 一个子进程出来。从这一点上来看，Nginx 也是一种"声明式API"
3. worker 和 worker 之间采用共享内存的方式进行通信

为什么 Nginx 使用多进程模型?

主要原因就在于 Nginx 需要保证提供可靠的服务。如果在引入第三方模块产生错误时，多线程由于位于某一个进程的共享空间内，将会导致整个 Nginx 进程异常退出，即内核向该进程发送 SIGSEGV 信号，Nginx 进程都没了，何况是工作的线程。而在使用多进程模型时，即使某一个 worker 子进程产生了段错误，也只是当前子进程被终止，父进程，也就是 master 进程将都不受影响，Nginx 仍然可对外提供服务（因为还有其它的 worker 进程存在）

```c
#include <pthread.h>
#include <unistd.h>


void *func(void *param) {
    // 给一个野指针赋值
    sleep(1);
    int *ptr;
    *ptr = 1024;
    return (void *)ptr;
}

int main() {
    pthread_t threads[4];
    for (int i = 0; i < 4; i++) {
        pthread_create(&threads[i], NULL, func, NULL);
    }

    for (int i = 0; i < 4; i++) {
        pthread_join(threads[i], NULL);
    }

    sleep(20);
    return 0;
}
```

使用左侧一段非常简短的代码即可说明问题，当其中一个线程访问了不应该访问的内存时，内核会产生一个 Segmentation fault 错误，直接干掉运行的进程，而非线程

Nginx 命令运行原理

我们可以使用 nginx -s reload 来对 Nginx 进行热部署，或者是使用 nginx -s stop 来优雅地停止其运行。其背后的工作原理其实就是信号（signal），也就是发送特定的信号给 master 进程，master 进程接收到信号以后，根据通过 sigaction() 所注册的信号处理函数来进行相应的处理

nginx -s reload 其实就相当于 kill -SIGHUP processID，nginx -s stop 其实就相当于 kill -SIGQUIT processID

使用 nginx -h 其实就可以看到 -s 到底做了什么 —— -s signal　: send signal to a master process: stop, quit, reopen, reload

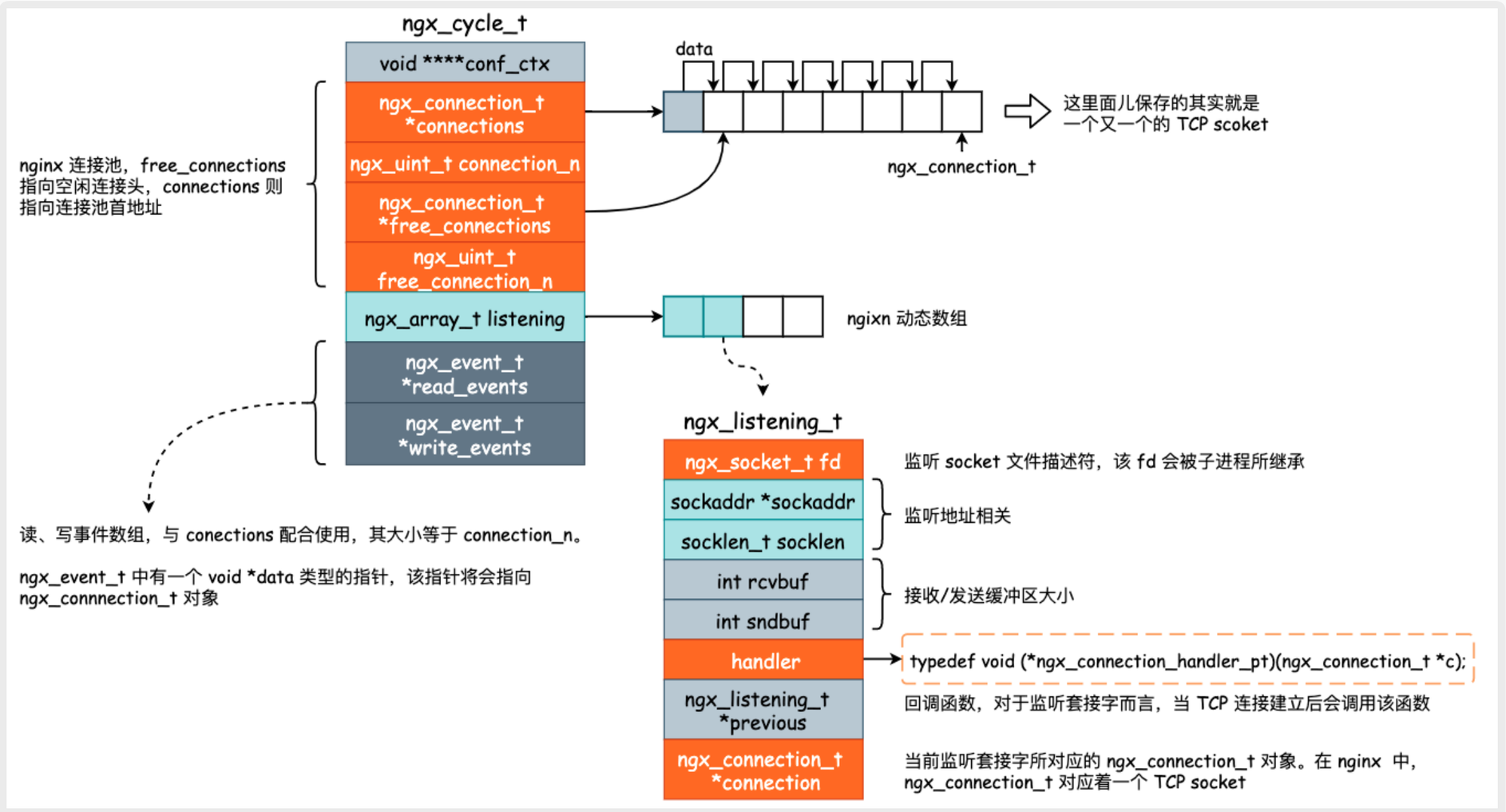
Nginx

reload 过程

流程

1. 向 master 进程发送 SIGHUP 信号（nginx -s reload），master 进程进入信号处理函数，并开始校验配置语法是否正确
2. 如果配置文件中新增了监听端口，例如 443，那么 master 进程将打开该端口
3. master 使用新的配置项创建新 worker 子进程，并向旧 worker 进程发送 SIGQUIT 信号
4. 旧 worker 进程关闭监听 socket，不再接收新的连接，在原有连接全部处理完毕后，则结束自身的运行

nginx 核心结构



ngx_cycle_t

不管是 master 进程，还是 worker 进程，都会持有一个 ngx_cycle_t 对象，用于把控监听对象、事件以及连接对象

ngx_cycle_t 中最为重要的成员就是 connections 和 free_connections 所组成的 socket 连接池，这部分的内存区域在 nginx 启动时就已经分配完毕了，在 ngx_event_process_init() 函数中执行

connections 指向连接池的首地址，而 free_connections 则指向下一个空闲连接。在获取连接时，取出 free-_connections 所指向的连接，并让链表头结点指向下一个节点。在归还连接时，直接插入到链表的首部位置即可