nginx 与 lighttpd 实现分析比较

- lichuang

讨论限定的版本

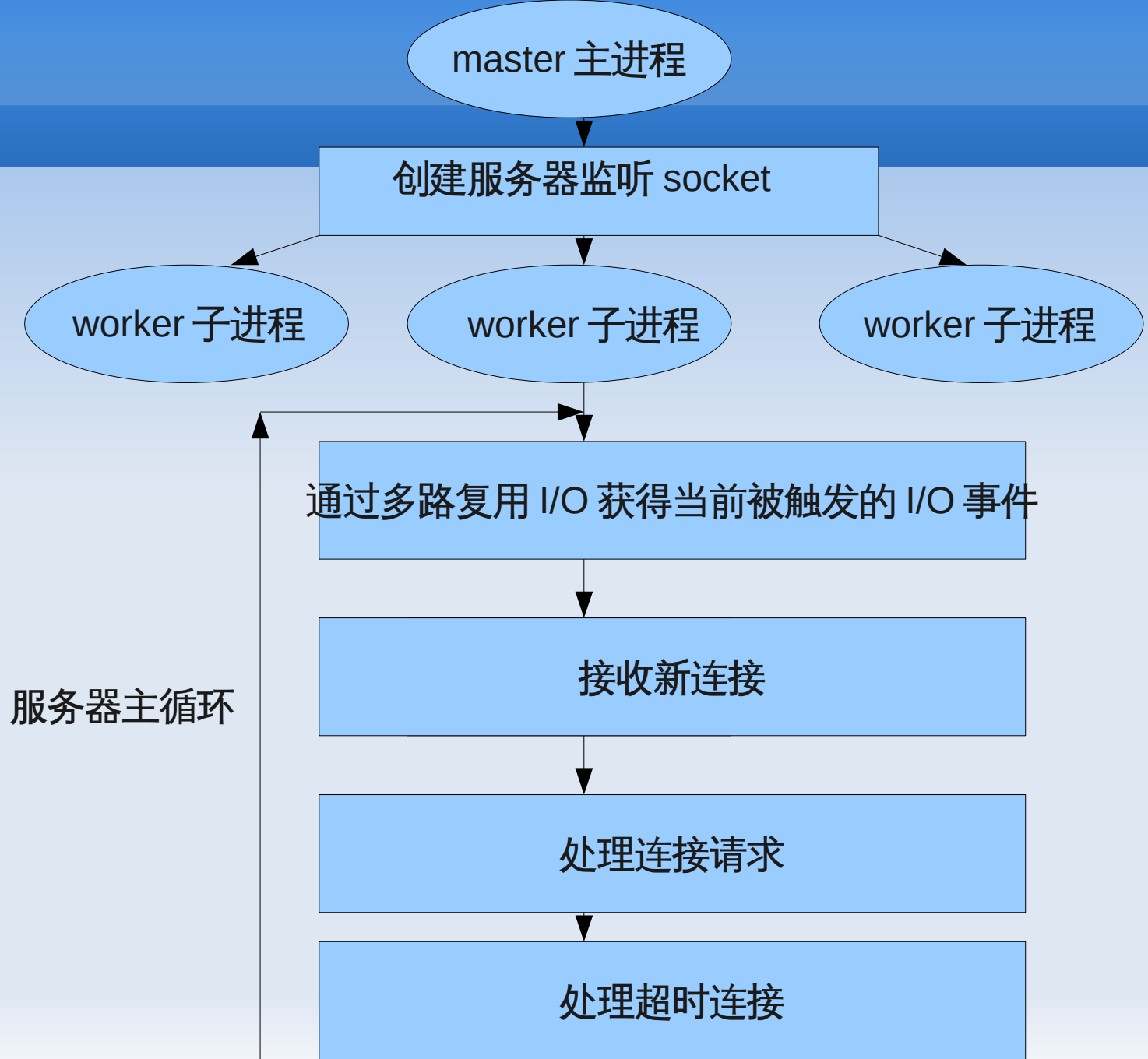
- Lighttpd 1.4.18(2007.9.9 发布），代码量 54417 行
- Nginx 0.7.61(2009.6.22 发布），代码量 109131 行
- 虽然都是轻量级 web 服务器，但是代码量还是有不小差距

两大部分内容

- 两者的整体架构分析
- 细节的差异

两者在整体架构方面大同小异

- 都是采用 master 进程 + 多个 worker 进程 + 多路复用 I/O 事件处理器 的架构
- master 进程即主进程，负责创建监听 socket ，创建 worker 子进程，并且监控子进程状态，但是自身并不处理连接请求
- worker 进程之间相互独立，各自使用多路复用 I/O 事件处理器等完成各自的工作

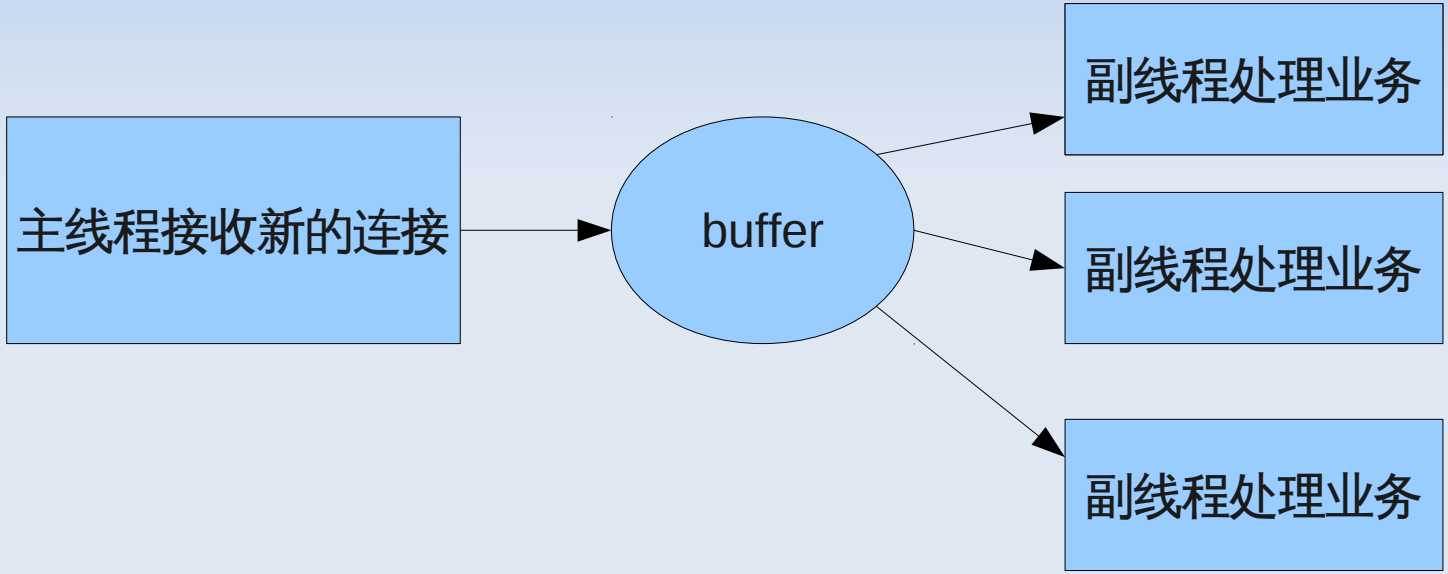

master 主进程
创建服务器监听 socket
worker 子进程
worker 子进程
worker 子进程
通过多路复用 I/O 获得当前被触发的 I/O 事件
接收新连接
处理连接请求
处理超时连接
服务器主循环

模型的思考

- 一个足够“简单”的模型，但是事实证明，也足够的高效。
- 既然说是“简单”，能举个“复杂”的例子吗？
- Why ？

一个相对复杂的模型

- 经典的生产者 - 消费者模型的服务器设计
-



为什么说它“复杂”？

- 线程间切换需要成本，处理一个请求耗费的操作更多了
- 编程的考量更多，要处理可能出现的同步 / 一致性等问题

Why ？

- 为什么 Nginx/Ligty 仅用多进程 + 多路复用 I/O 处理即可以取得高效？
-

Web 服务器的业务逻辑

- Web 服务器的业务处理相对简单：接收新的连接，接收连接请求，分析连接请求，根据请求进行回复。
- 即使有可能出现的耗时操作，如读取数据库等也有办法解决。
-

结论

- Web 服务器是 I/O 密集型的服务器，不是 CPU 操作密集型的服务器
- 每个进程都维持大量的连接，因此需要选择一个足够高效的 I/O 处理机制（epoll/select/poll）

为什么不采用多线程？

- 比如处理一个 http 请求，只需要 1ms ，而线程 / 进程间的切换就需要 0.5ms ，还要在编码时处处小心，付出的这些代价还值得吗？
- 反之，其他业务类型的服务器，处理一个请求需要 10ms ，为了不至于处理一个请求阻塞了对其他请求的处理，就需要根据不同的业务处理分成多个线程 / 进程处理

master 进程

- Master 进程：
- 1）创建监听 socket
- 2）创建 worker 子进程，之后的主要动作就是监控子进程的工作状态

worker 子进程

- Worker 进程的主循环：
- 更新当前时间
- 查看当前监听的事件是否被触发，如有调用相应的处理函数进行处理
- 处理超时连接
-

第二部分 细节

针对第一部分的整体架构，看里面实现的细节差异

master 与 worker 进程的关系

- 前面已经分析过， master 负责创建监听套接字，同时也是 worker 的父进程，这一点上，两者一样，但是 .....

master 与 worker 进程的关系

- Lighttpd 里， master 进程只是简单的负责创建出子进程，之后监控子进程是否退出，如果退出再次重新创建子进程出来干活

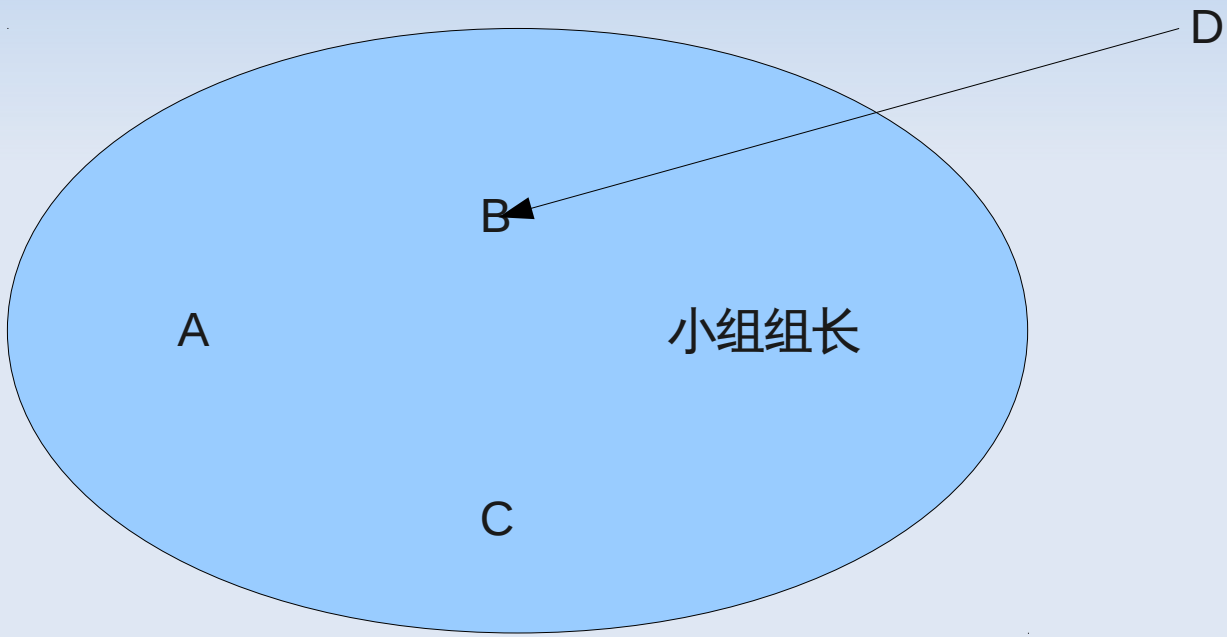
master 与 worker 进程的关系

- Nginx 里， master 进程除了这个功能之外，父子进程之间还可以进行通信，完成热升级配置文件，热更新等功能
- master 进程接管了一切客户发送给进程的信号，客户不能直接控制 worker 进程，只能通过 master 进程

比较

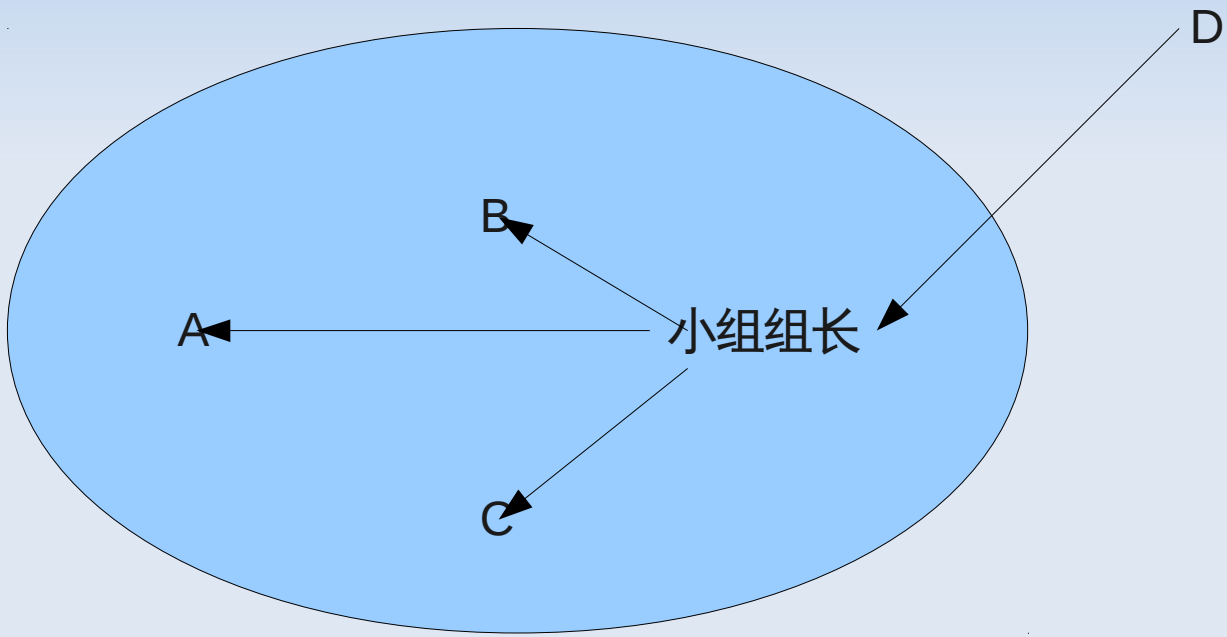
- Nginx 中 master 进程的责任更多，除了监控 worker 子进程之外，还处理客户的控制指令，统一由它分发给 worker 子进程

Why ？



不通过组长直接控制组内人员的结构图

Why ？



由组长统一分配组内人员的结构图

优点

- Master 与 worker 进程之间各司其职， master 是对外的接口， worker 是真正干活的
- 用户的控制指令可以由 master 详细记录

worker 进程间的负载均衡

- 由于各个 worker 进程之间相互独立，由内核协议栈统一将接收的新连接分发给各个 worker 子进程进行处理，很可能出现有些子进程处理的连接多，有些处理的少 ---- 即负载不均衡

worker 进程间的负载均衡

- Nginx 中，子进程每次接收一个新连接，会根据当前该进程的连接数量，更新一个阈值，由这个阈值决定是否该暂缓接收新的连接
- 这样，各个子进程之间有一个相对的均衡
- Nginx 中还对 accept 操作做了加锁，任意时刻只可能有一个子进程可以接收新的连接，避免惊群现象的出现

worker 进程间的负载均衡

- Ligty 在子进程当前接收的连接数量达到可用连接数量的 0.9 倍时，禁止接收新的连接
- ligty 没有对惊群现象做处理

惊群

- 惊群是什么？
- 在新版内核中是否还存在惊群？
- 惊群有多大的影响？

处理超时连接

- Lighttpd 采用的是注册一个回调函数，每一秒被触发一次，在每次触发时调用函数轮询所有的连接查看是否超时。
- 这个实现是一个非常耗时的操作。

ligty 处理超时连接的伪代码

- 信号处理函数
- ｛置标志位｝
- 服务器初始化的时候注册一个信号处理函数，每一秒被触发一次。每次服务器主循环中判断标志位是否被置位，如果被置位，轮询连接查找超时连接

犯过的一个愚蠢的错误

- 伪代码：
- 信号处理函数
- ｛轮询连接查找超时连接｝
- 服务器初始化的时候注册一个信号处理函数，每一秒被触发一次。

犯过的一个愚蠢的错误

- 假设当前进程在执行一些不可重入函数的操作时，被中断打断，而这次中断又会去执行同样的不可重入函数，将会造成进程死锁

犯过的一个愚蠢的错误

- 附件中的 server_gdb.txt 是出现死锁情况时的 gdb 跟踪函数栈桢信息
- 这份信息显示，进程在调用 free （不可重入函数）时被中断打断，而中断中由于要释放超时连接，再次调用 free 函数，造成死锁

教训

- 信号是不可预测的，不知道在什么时候就被触发了
- 在信号处理函数中不要调用不可重入的函数
- 尽量少的在信号处理函数中处理事情，就目前见过的几个服务器项目（ligty,nginx,libevent）等都是在信号触发时保存一个标志位表示信号被触发，服务器主循环中再处理

Nginx 处理超时连接

- 使用红黑树作为存放定时器的数据结构
- 每次从红黑树中取出根节点，从而得到距离目前最快发生的时间差，使用这个时间差作为调用多路复用 I/O 操作的参数，当函数返回，只可能是 I/O 事件被触发，或者超时
- 处理完 I/O 事件之后，得到处理前后的时间差，根据这个时间差依次查看红黑树中哪些定时器可以被处理

比较

- Ligty 的超时处理太耗时
- Nginx 采用巧妙的数据结构和策略设计，既可以不使用信号去触发定时器，由可以不必轮询所有的定时器查看是否超时

处理连接请求

- 处理一个 http 请求的大致流程：接收请求报文包头，根据包头接收包体，分析请求报文，回复请求结果
- 这些流程可以看作是不同的处理“状态”

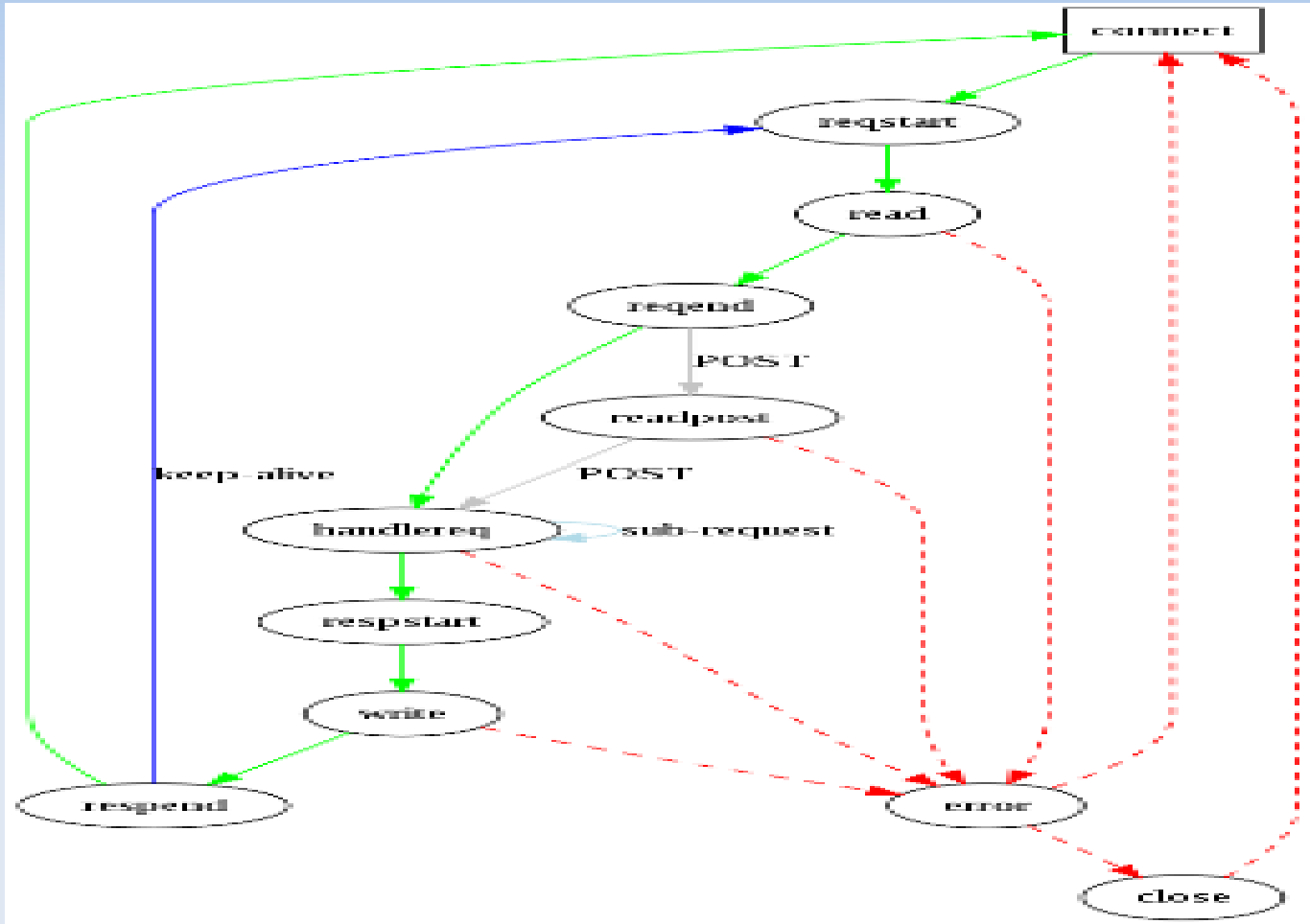
处理连接请求

- Nginx 里针对不同的状态，调用不同的处理函数，每一个状态的处理函数中，在处理成功之后，将处理函数指针赋值为下一个处理函数
-

处理连接请求

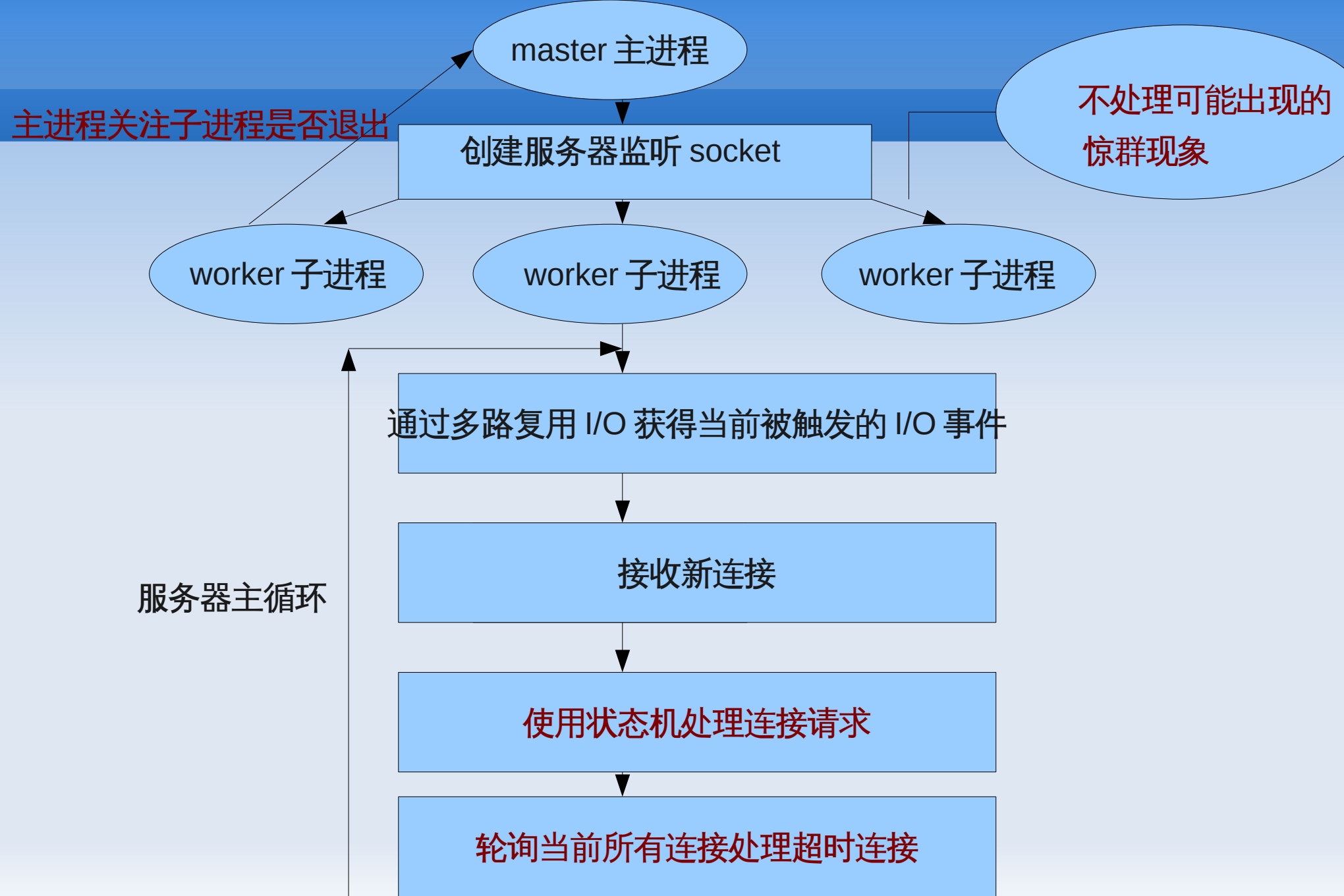
- Ligty 使用状态机，每个状态的处理函数只需要保存此次处理的结果，由状态机根据这些结果决定下一步的状态

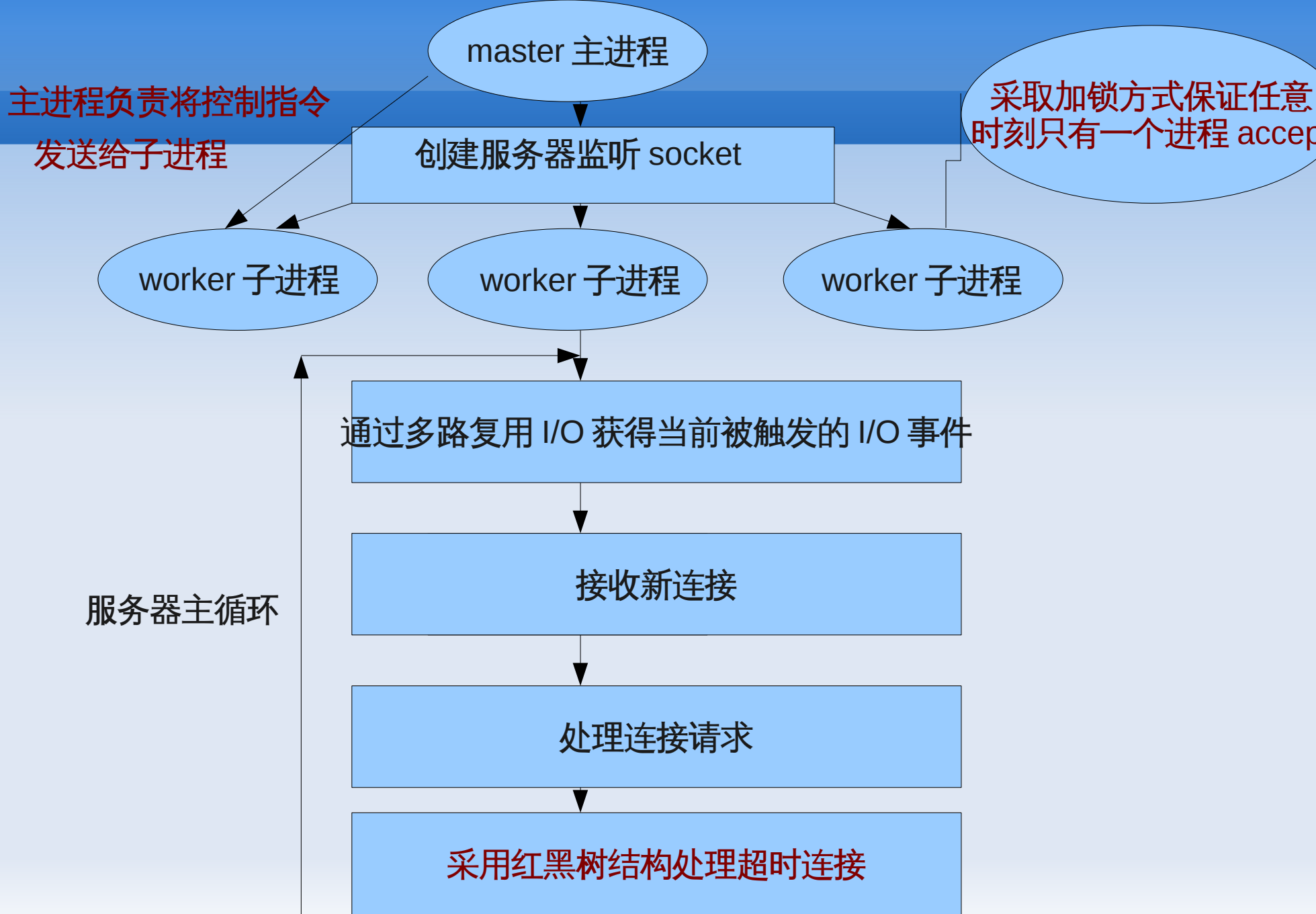
Ligty 状态机


connect
reqstart
read
reqend
POST
readpost
keep-alive
POST
handlereq
sub-request
respstart
write
resqend
error
close

比较

- 在处理连接请求这一点上，两者的方式不尽相同，但是本质一样– -- 都是根据不同的状态调用不同的处理
- nginx 状态处理函数之间的耦合紧密，状态切换时的下一步处理由状态处理函数来决定
- ligty 将状态切换的动作放在状态机里，各个状态处理函数不关心下一步需要做什么，状态之间的耦合小


master 主进程
主进程关注子进程是否退出
创建服务器监听 socket
不处理可能出现的
惊群现象
worker 子进程
worker 子进程
worker 子进程
通过多路复用 I/O 获得当前被触发的 I/O 事件
接收新连接
服务器主循环
使用状态机处理连接请求
轮询当前所有连接处理超时连接


master 主进程
主进程负责将控制指令
发送给子进程
创建服务器监听 socket
采取加锁方式保证任意
时刻只有一个进程 accep
worker 子进程
worker 子进程
worker 子进程
通过多路复用 I/O 获得当前被触发的 I/O 事件
接收新连接
处理连接请求
采用红黑树结构处理超时连接
服务器主循环

总结

特性	ligty	nginx
父子进程关系	父进程仅关注子进程是否退出	父进程既控制子进程，同时也接管所有外部控制指令
子进程间的负载均衡	达到一个阙值之后暂停接收新连接	达到一个阙值之后暂停接收新连接
惊群	Let it be	对 accept 操作加锁，任意时刻只能有一个子进程接收新连接
处理超时连接	每秒触发一次处理，每次都是轮询所有连接	红黑树存放定时器数据，更精确，同时也不会轮询所有连接来查询超时
处理连接请求	由状态机来决定	不同状态调用不同的函数处理