06 我快我有理：深扒 Nginx 工作原理

更新时间：2019-12-10 09:48:15

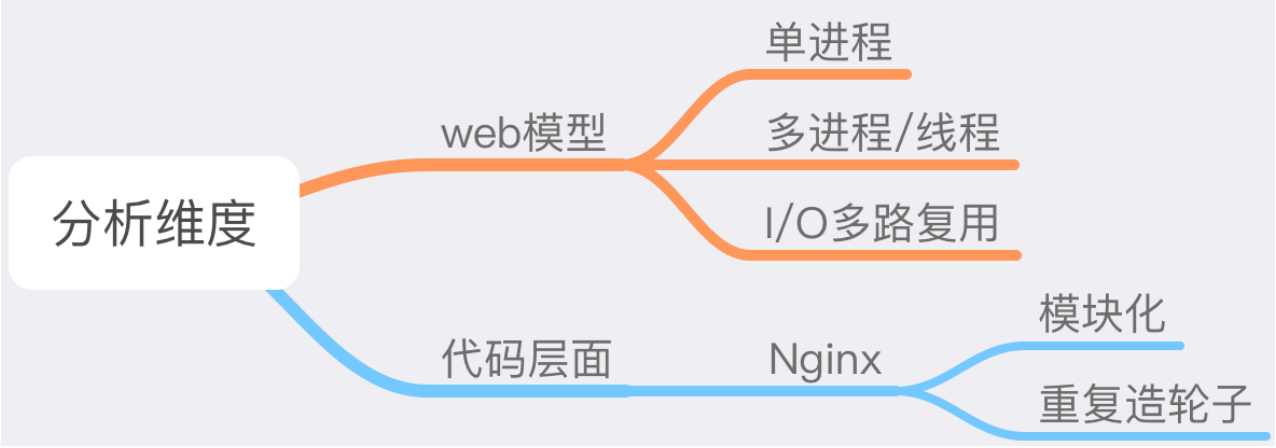
> 我们有力的道德就是通过奋斗取得物质上的成功；这种道德既适用于国家，也适用于个人。——罗素

回顾

我们在上一篇文章中仔细的分析了 Linux 中常见的几种 I/O 模型。理解这部分内容对于理解 Nginx 原理是非常有必要的。本篇内容我们分享一下为什么 Nginx 的效率这么高。

高效原因

在本文中，我会从两个方面来分析 Nginx 高效的原因：



架构层面

一般来说，一个 Web 服务器主要有三种处理连接的方式，分别是：单进程、多进程以及 I/O 多路复用。

单进程

我们在学习网络课程的时候，应该大部分都自己实现过一个简单的 Web 服务器。

下面是用伪代码简单实现了一个 Web 服务器。

```
socket = create_socket();   // 创建一个socket
bind_socket_and_port(socket, port);    // 绑定socket
listen_socket(socket);   // 监听socket

for (;;){
   new_socket = accept_new_connection(new_connection); // 接受请求
   doSomeWork(new_socket); // 业务处理
   close(new_socket); // 关闭请求
}
```

这就是一个标准的单进程服务器。这种服务器的优点是实现起来非常的简单，但是缺点也非常的明显，因为一次只能处理一个请求，显然速度太慢了。

> 实际上，我并没有听说哪个用于生产环境的 Web 服务器是单进程的。但是这个确实是最简单的一种方式～～～

多进程

与单进程模式不同的是，在多进程模式下，每当一个新的请求到来之后，Web 服务器都会创建一个新的进程专门处理这个请求。

```
socket = create_socket();   // 创建一个socket
bind_socket_and_port(socket, port);    // 绑定socket
listen_socket(socket);   // 监听socket

for(;;){
   new_socket = accept_new_connection(new_connection); // 接收一个请求
   fork(); // 创建一个子进程
}
```

在子进程中：

```
void doSomeWork(new_socket){
   // 业务处理
}
```

我们以餐厅的例子来说明：每次来一个客人，就专门的为客人分配一个服务员来为其服务，这个客人的所有需求都有专人来负责。

这种模式的优点就是：非常的稳定，当一个进程因意外退出之后，并不会影响其它的进程（一个服务员罢工，只会影响一位客人）。

很明显的，这种模式最大的一个缺点就是资源浪费。当有成千上万个请求到来的时候，就需要创建成千上万个进程，这对于操作系统的资源来说是一个巨大的挑战，并且进程之间的切换非常的浪费时间（客人太多的话，就需要大量的服务员）。

> 早期由于没有 I/O 多路复用机制，所以很多 Web 服务器都是使用这种方式来实现的，比如大名鼎鼎的 Apache。

I/O 多路复用

上一篇文章中，我们已经分享了 I/O 多路复用的机制。简单来说，就是一次监控很多个 I/O 操作，当某个 I/O 准备好之后，就进行相应的操作，比如读写文件等。

```
socket = create_socket();   // 创建一个socket
bind_socket_and_port(socket, port);    // 绑定socket
listen_socket(socket);   // 监听socket

for(;;){
    new_socket = accept_new_connection(new_connection); // 接收一个请求
    add_new_socket_to_epoll(new_socket); // 将新的socket加入到epoll中
    ready_socket_array = epoll_wait(); // 等待I/O操作
    for (socket in ready_socket_array){
        // 循环处理所有的socket
        // XXX
    }
}
```

为了支持高并发的请求，现在越来越多的 Web 服务都正在采用这种 I/O 多路复用的结构，优秀的 Nginx 同样采用了这种模式，所以能够完美的解决 C10K 问题。

其实，说到多路复用，有一个绕不开的话题，那就是 select 和 epoll。这两个函数都是 Linux 提供的系统调用，用于完成 I/O 多路复用。

如果大家去网上搜一搜，可以找到大量的分析二者区别的文章。在这里呢，我简单分享一下我的理解：
select 是早期的 Linux 系统用于 I/O 多路复用的一个函数。默认情况下，它可以监控 1024 个 I/O。
epoll 是新版 Linux 提供的 I/O 多路复用的系统函数，解决 select 的一些缺点。

我们下面通过两段伪代码来了解一下 select 和 epoll 的区别:

```
// 下面是一段伪代码
ret = select(all_fds)

// 遍历所有的I/O
for (fd in all_fds){
    // 进行业务处理
}
ret = select(all_fds);
// 遍历准备好的I/O
for (fd in ready_fds){
    // 进行业务处理
}
```

上面两段伪代码写得很清楚了，select 函数之后，程序要遍历所有的 I/O，而 epoll 只需要遍历准备好的 I/O。比如我们监控了 1000 个 I/O，但是同一时间只有两个 I/O 准备好了。这个时候如果使用 select 函数的话，我们要遍历这 1000 个 I/O，找到准备好的 I/O，然后进行相应的操作。而对于 epoll 的话，直接就会返回准备好的 I/O，我们只需要处理这两个 I/O 就行了。如果同时监控大量 I/O 的话，效率差别就特别明显了。

> PS1：select 监控的文件描述符数量可以在内核中修改。
> PS2：Nginx 使用的是性能更高的 epoll 机制。

我们上面分析了这三种 Web 服务器的架构的优缺点，而 Nginx 正是使用了性能最高的 I/O 多路复用方式。

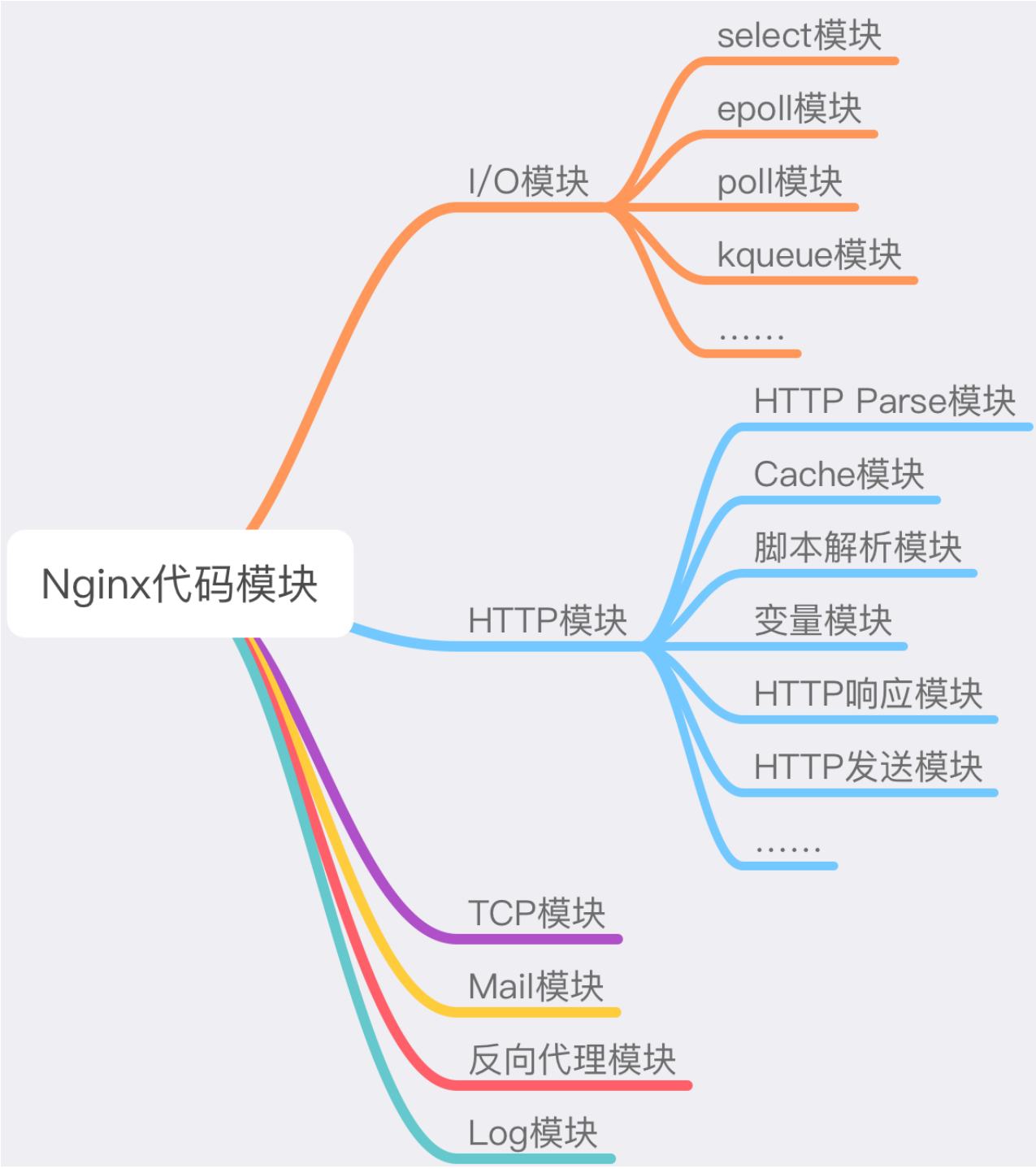
这仅仅是 Nginx 高效的原因之一，下面我们从代码层面分享一下 Nginx 高效的原因。

代码层面

我个人正在学习 Nginx 源码，所以从我的角度来分享一下我在学习过程中的感受，我认为这些东西也是 Nginx 高效的一个原因。

模块化

首先，Nginx 代码模块化的结构。每个功能都是一个基本独立的模块，比如处理 I/O 操作的模块，处理 Http 的模块，处理反向代理的模块。



这样的逻辑拆分，可以有效的保证了代码的健壮性，每个模块之间没有强耦合关系，模块与模块之间互不影响。并且这种形式对于我们编写 Nginx 插件也是非常友好的。

重复造轮子

这里说的重复造轮子并不是贬义词。Nginx 为了高效，节省内存，并没有使用现成的许多东西，而是对常用的一些数据结构重新造轮子，包括如下：

这部分可以说充分的体现了 Nginx 作者登峰造极的编程水平，对内存的利用率，对性能的考虑体现在了每一行代码之间，这也是 Nginx 被人们广为赞誉的一个原因，那就是相当的节省内存。

> 如果有机会，大家一定要看一看 Nginx 的源码～～～

总结

本文简单的分享了一下 Nginx 为什么这么快，既包括了架构方面，也包含了代码层面。希望对大家有所帮助。

}

05 Linux的 IO 原理

07 打扮你的nginx：配置文件初识