Nginx

原理

Nginx的进程模型

> 在工作方式上，Nginx分为单工作进程和多工作进程两种模式。
>
> - 在单工作进程模式下，除主进程（master）外，还有一个工作进程（worker），工作进程是单线程的；
>   - 解读：就是总共2个进程，但是每个进程只在一个线程上运行；
> - 在多工作进程模式下，每个工作进程包含多个线程。Nginx默认为单工作进程模式。

理解：默认一个主进程、一个工作进程（单线程的），如果设置多工作进程模式，那么会有多个工作进程（多线程的）

- > 在一些磁盘使用和CPU负载模式，应调整nginx工作(`worker`)的数量。 在这里说一点基础规则：系统管理员应该为其工作负载尝试几个配置。 一般建议可能如下：如果负载模式是CPU密集型的，例如，处理大量TCP/IP，执行SSL或压缩，则nginx工作(`worker`)的数量应与CPU内核数量相匹配; 如果负载大多是磁盘I/O绑定，例如，从存储或重代理服务不同的内容集合 - 工作(`worker`)的数量可能是核心数量的一到两倍。有些工程师会根据个人存储单元的数量选择工作(`worker`)的数量，但这种方法的效率取决于磁盘存储的类型和配置。

- 所有进程主要使用`共享内存机制`进行进程间通信
- 一些特殊模式下还会有一些特殊进程，比如`缓存加载器`和`缓存管理器`

并发请求竞争

当我们提供80端口的http服务时，一个连接请求过来，每个进程都有可能处理这个连接，怎么做到的呢？

> 首先，每个worker进程都是从master进程fork过来，在master进程里面，先建立好需要listen的socket（listenfd）之后，然后再fork出多个worker进程。所有worker进程的listenfd会在新连接到来时变得可读，为保证只有一个进程处理该连接，所有worker进

> 程在注册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。

- `mutex` - Mutual exclusion 互斥锁

> 首先，Nginx 的处理得先打开 accept_mutex 选项，此时，只有获得了 accept_mutex 的进程才会去添加accept事件，也就是说，Nginx会控制进程是否添加 accept 事件。Nginx 使用一个叫 ngx_accept_disabled 的变量来控制是否去竞争 accept_mutex 锁。在第一段代码中，计算 ngx_accept_disabled 的值，这个值是 Nginx 单进程的所有连接总数的八分之一，减去剩下的空闲连接数量，得到的这个 ngx_accept_disabled 有一个规律，当剩余连接数小于总连接数的八分之一时，其值才大于 0，而且剩余的连接数越小，这个值越大。再看第二段代码，当 ngx_accept_disabled 大于 0 时，不会去尝试获取 accept_mutex 锁，并且将 ngx_accept_disabled 减 1，于是，每次执行到此处时，都会去减 1，直到小于 0。不去获取 accept_mutex 锁，就是等于让出获取连接的机会，很显然可以看出，当空余连接越少时，ngx_accept_disable 越大，于是让出的机会就越多，这样其它进程获取锁的机会也就越大。不去 accept，自己的连接就控制下来了，其它进程的连接池就会得到利用，这样，Nginx 就控制了多进程间连接的平衡了。

优缺点

Nginx VS Apache

命令

- kill -HUP pid

  从容地重启nginx，一般用来重启nginx，或重新加载配置

  - 执行过程：发送kill命令道master进程，master加载新配置，master通知老worker进程停止接收请求（处理完正在处理的请求后，就退出），新worker接管请求；

- nginx在0.8版本之后，引入了一系列命令行参数，来方便我们管理。

  > 比如，./nginx -s reload，就是来重启nginx，./nginx -s stop，就是来停止nginx的运行。如何做到的呢？我们还是拿reload来说，我们看到，执行命令时，我们是启动一个新的nginx进程，而新的nginx进程在解析到reload参数后，就知道我们的目的是控制nginx来重新加载配置文件了，它会向master进程发送信号，然后接下来的动作，就和我们直接向master进程发送信号一样了。

理解：新建一个进程（接手master的工作），解析命令参数，向master发送命令，之后，就是类似上面kill；

- 启动
  - `./nginx`
- 管理命令

  格式：**nginx -s signal**
  - signal可以是：`stop` - 快速关闭、`quit` - 优雅关闭、`reload` - 重载配置、`reopen` - 重新打开日志

配置

配置文件

- nginx 的配置文件，默认的位置包括：
  - `/etc/nginx/nginx.conf`，
  - `/usr/local/etc/nginx/nginx.conf`，或
  - `/usr/local/nginx/conf/nginx.conf`

配置指令

- *如果配置指令包含空格，一定要单引号或双引号括起来*

指令类型

- **普通指令**

  > 每个上下文仅有唯一值。而且，它只能在当前上下文中定义一次。子级上下文可以覆盖父级中的值，并且这个覆盖值只在当前的子级上下文中有效

```
gzip on;
gzip off; # 非法，不能在同一个上下文中指定同一普通指令2次
```

- **数组指令**

  同级上下文中，会追加；子级上下文会覆盖父级上下文配置；
- **行动指令**

行动是改变事情的指令。根据模块的需要，它继承的行为可能会有所不同

具体指令

server

- server_name - 指令的值将检测 Host 头(存储着主机域名)
- Nginx 将会按照下列顺序选择虚拟主机（server上下文）：
  1. 匹配sever_name指令的IP-端口主机
  2. 拥有default_server标记的IP-端口主机

```
server {
  listen      *:80 default_server;
  server_name netguru.co;

  return 200 "Hello from netguru.co";
}
```

  3. 首先定义的IP-端口主机
  4. 如果没有匹配，拒绝连接。
- server_name也可以通过正则表达式进行匹配

  如：在一个server块中配置多个站点：

```
server
   {
     listen       80;
     server_name  ~^(www\.)?(.+)$;
     index index.php index.html;
     root  /data/wwwsite/$2;
   }
```

  站点的主目录应该类似于这样的结构：

```
/data/wwwsite/domain.com
/data/wwwsite/nginx.org
/data/wwwsite/baidu.com
/data/wwwsite/google.com
```

- server_name的正则匹配顺序

> Nginx 会存储 3 个哈希表：确切的名字，以星号开始的通配符，和以星号结尾的通配符。如果结果不在任何表中，则将按顺序进行正则表达式测试。

1. 确切的名字
2. 最长的通配符名称以星号开始，例如“* .example.org”。
3. 最长的通配符名称以星号结尾，例如“mail.**”
4. 首先匹配正则表达式（按照配置文件中的顺序）

- 正则表达式注意
- 命名的正则表达式捕获组在后面可以作为变量使用：

```
server {
    server_name   ~^(www\.)?(?<domain>.+)$;  #命名捕获组

    location / {
        root   /sites/$domain;
    }
}
```

  - PCRE使用下面语法支持命名捕获组：
    - `?<name>` - 从PCRE-7.0开始支持，兼容Perl 5.10语法
    - `?'name'` - 从PCRE-7.0开始支持，兼容Perl 5.10语法
    - `?P<name>` - 从PCRE-4.0开始支持，兼容Python语法
- nginx使用的正则表达式兼容 `PCRE` 。为了使用正则表达式，虚拟主机名必须以波浪线“~”起始
- **.netguru.co**
  - .netguru.co - 本意就是 `*.netguru.co`
- server_name的 `通配符`
  - 通配符名字只可以在名字的 `起始处` 或 `结尾处` 包含一个星号，并且星号与其他字符之间用 `点` 分隔
- 如果一个请求的请求行中包含有host信息，nginx会忽略header中的host，而使用请求行中的host进行server_name匹配

- 如果不允许请求中缺少“Host”头，可以定义如下主机，丢弃这些请求：

```
server {
    listen       80;
    #如果server块中没有定义server_name，nginx使用空名字作为虚拟主机名；
nginx 0.8.48版本以下（含）在同样的情况下会使用机器名作为虚拟主机名
    server_name  "";
    return       444;
}
```

  在这里，我们设置主机名为空字符串以匹配未定义“Host”头的请求，而且返回了一个nginx特有的，非http标准的返回码444，它可以用来关闭连接。

  > 从0.8.48版本开始，这已成为主机名的默认设置，所以可以省略`server_name ""`。而之前的版本使用机器的*hostname*作为主机名的默认值。

- 特殊的server_name - *
  - 特殊的名字
- `确切名字`和`通配符名字`存储在哈希表中。哈希表和监听端口关联。哈希表的尺寸在配置阶段进行了优化，可以以最小的CPU缓存命中失败来找到名字。设置哈希表的细节参见这篇文档

location

try_files

- 应该避免在 server 上下文中出现 try_files - why

client_header_buffer_size

> Nginx 会将整个请求头都放在一个 buffer 里面，这个 buffer 的大小通过配置项 client_header_buffer_size 来设置，如果用户的请求头太大，这个 buffer 装不下，那 Nginx 就会重新分配一个新的更大的 buffer 来装请求头，这个大 buffer 可以通过 large_client_header_buffers 来设置，这个 large_buffer 这一组 buffer，比如配置 48k，就是表示有四个 8k 大小的 buffer 可以用。注意，为了保存请求行或请求头的完整性，一个完整的请求行或请求头，需要放在一个连续的内存里面，所以，一个完整的请求行或请求头，只会保存在一个 buffer 里面。这样，如果请求行大于一个 buffer 的大小，就会返回 414 错误，如果一个请求头大小大于一个 buffer 大小，就

> 会返回 400 错误。

> 客户端请求头部的缓冲区大小(client_header_buffer_size)，这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。

- 传送门

lingering_close

- 传送门

proxy_pass

- 通俗示例

日志配置

- 传送门

安装设置 ✈

- 例：在nginx目录下执行，./configure --prefix=../nginx --sbin-path=../nginx/bin/nginx

编译时特殊依赖

zlib

- `--without-http_gzip_module` — 不编译http_gzip_module模块。该模块可以压缩HTTP服务器的响应，该模块需要 `zlib` 库才能编译和运行。
- `--with-zlib=path` — 设置zlib库源文件的路径地址

PCRE

- `--without-http_rewrite_module` — 不编译http_rewrite_module模块。该模块允许HTTP服务器重定向请求，改变请求的URI地址 。创建并运行该模块需要 `PCRE` 库支持。
- `--with-pcre=path` — 设置PCRE库源文件的路径地址。PCRE库的发行版(version 4.4 — 8.30)需要先从PCRE站点下载并解压缩
- `--with-pcre-jit` — 编译PCRE库时增加"实时编译（pcre_jit）"支持。
- 配置中使用到的正则匹配主要使用到的就是PCRE库

OpenSSL

- `--with-http_ssl_module` — 为HTTP服务器编译HTTPS协议支持的模块。该模块默认是不编译的。它需要OpenSSL库才能编译和运行。

优化

- worker进程数最好和CPU物理核数相同
- 可以绑定worker进程到特定核心上

server_name

> nginx首先搜索确切名字的哈希表，如果没有找到，搜索以星号起始的通配符名字的哈希表，如果还是没有找到，继续搜索以星号结束的通配符名字的哈希表。
>
> 因为名字是按照域名的节来搜索的，所以搜索通配符名字的哈希表比搜索确切名字的哈希表慢。注意特殊的通配符名字"`.example.org`"存储在通配符名字的哈希表中，而不在确切名字的哈希表中。
>
> 正则表达式是一个一个串行的测试，所以是最慢的，而且不可扩展。
>
> 鉴于以上原因，请尽可能使用确切的名字。

博文

nginx 并发数问题思考：worker_connections,worker_processes与 max clients