OpenResty

1.反向代理

Web 代理服务器是网络的中间实体，代理位于 Web 客户端和 Web 服务器之间，属于“中间人”的角色，Http Proxy 服务器既是 Web 服务器又是 Web 客户端：

反向代理（Reverse Proxy）方式是指以代理服务器来接受 internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 internet 上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

2.nginx

Nginx ("engine x") 是一个高性能的 HTTP 和 反向代理服务器。nginx 于 2004 年发布，聚焦于高性能，高并发和低内存消耗问题。并且具有多种 web 服务器功能特性：负载均衡，缓存，访问控制，带宽控制，以及高效整合各种应用的能力，这些特性使 nginx 很适合于现代网站架构。目前，nginx 已经是互联网上第二流行的开源 web 服务器软件。

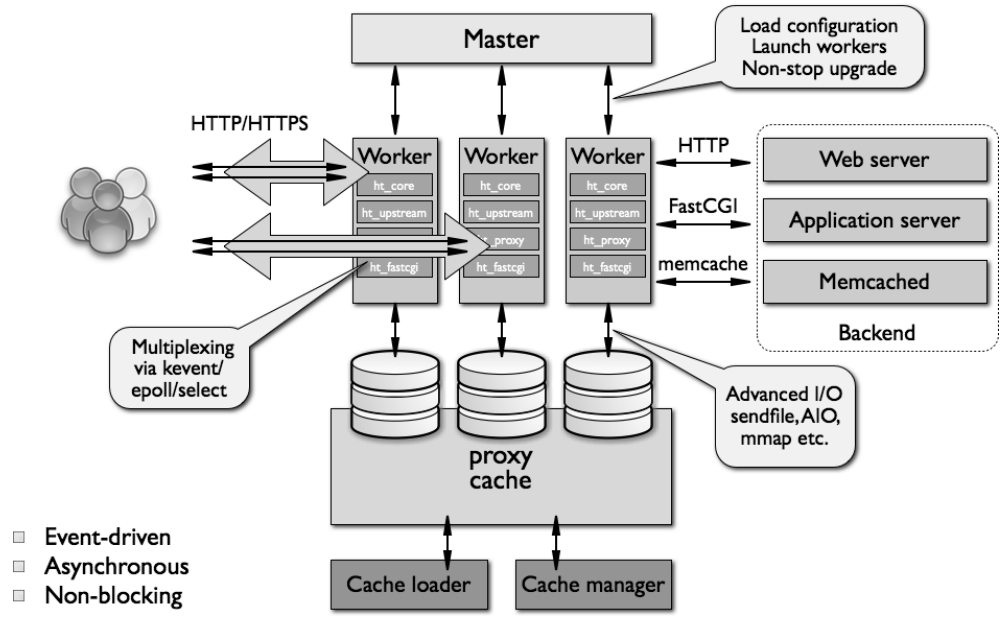
Nginx 的使用如下图所示：

2.1 Nginx worker 代码结构

Nginx worker 的代码包括核心和功能模块。核心维护一个紧凑的事件处理循环，并且在请求处理的每个节点执行对应的模块代码段。模块完成了大部分展现和应用层功能，包括从网络和存储设备读取、写入，转换内容，进行输出过来，SSL（Server-side include）处理，或者如果启动代理则转发请求给后端服务器。

nginx 模块化的架构运行开发者扩展 web 服务器的功能，而不需要修改 nginx 核心。Nginx 可分为核心、事件模块、阶段处理器、协议、变量处理器、过滤器、上游和负载均衡器等。目前，Nginx 不支持动态加载模块，即模块代码是和 nginx 核心代码一起编译的。

Nginx 在 Linux/Solaris 和 BSD 系统上使用 kqueue、epoll 和 event ports 等技术，通过事件通知机制来处理网络连接和内容获取、包括接受、处理和管理连接，并且大大增强了磁盘 IO 性能。目的在于尽可能的提供操作系统建议的手段，用于从网络进程流量，磁盘操作，套接字读取和写入，超时等事件中及时异步地获取反馈。Nginx 为每个基于 Unix 操作系统大量优化了这些多路复用和高级 I/O 操作方法。下图是 Nginx 架构的高层设计：



2.2 工作进程模型

Nginx 不为每个连接派生进程或线程，而是由 worker 进程通过监听共享套接字接收新请求，并且使用高效的循环来处理数千个连接。Nginx 不使用仲裁器或分发器来分发连接，这个工作由操作系统内核机制完成。监听套接字在启动时就完成初始化，worker 进程通过这些套接字接收、读取请求和输出响应。

事件处理循环是 nginx worker 代码中最复杂的部分，它包含复杂的内部调用，并且严重依赖异步任务处理的思想。异步操作通过模块化、时间通知、大量回调函数以及微调定时器等实现。总的来说，基本原则就是尽可能做到非阻塞。Nginx worker 进程唯一会被阻塞的情形是磁盘性能不足。

由于 nginx 不为每个连接派生进程或线程，所以内存使用在大多数情况下是很节约并且高效的。同时由于不用频繁的生成和销毁进程或线程，所以 nginx 也很节省 CPU 时间。Nginx 所做的就是检查网络和存储的状态，初始化新连接并添加到主循环，异步处理直到请求结束

才从主循环中释放并删除。兼具精心设计的系统调用和诸如内存池等支持接口的精确实现，nginx 在极端负载的情况下通常能做到中低 CPU 使用率。

nginx 派生多个 worker 进程处理连接，所以能够很好的利用多核 CPU。通常一个单独的 worker 进程使用一个处理器核，这样能完全利用多核体系结构，并且避免线程抖动和锁。在一个单线程的 worker 进程内部不存在资源匮乏，并且资源控制机制是隔离的。这个模型也允许在物理存储设备之间进行扩展，提高磁盘利用率以避免磁盘 I/O 导致的阻塞。将工作负载分布到多个 worker 进程上最终能使服务器资源被更高效的利用。

针对某些磁盘使用和 CPU 负载的模式，nginx worker 进程数应该进行调整。这里的规则比较基本，系统管理员应根据负载多尝试几种配置。通常推荐：如果负载模式是 CPU 密集型，例如处理大量 TCP/IP 协议，使用 SSL，或者压缩数据等，nginx worker 进程应该和 CPU 核心数相匹配；如果是磁盘密集型，例如从存储中提供多种内容服务，或者是大量的代理服务，worker 的进程数应该是 1.5 到 2 倍的 CPU 核心数。一些工程师基于独立存储单元的数目来决定 worker 进程数，虽然这个方法的有效性取决于磁盘存储配置的类型。

2.3 nginx 进程角色

Nginx 在内存中运行多个进程，一个 master 进程和多个 worker 进程。同时还有一些特殊用途的进程，例如缓存加载和缓存管理进程。在 Nginx 1.x 版本，所有进程都是单线程的，使用共享内存作为进程间通信机制。Master 进程使用 root 用户权限进行，其他进程使用非特权用户权限运行。master 进程负责下列工作：

1. 读取和校验配置文件
2. 创建、绑定、关闭套接字
3. 启动、终止、维护所配置数目的 worker 进程
4. 不中断服务刷新配置文件
5. 不中断服务升级程序（启动新程序或在需要时回滚）
6. 重新打开日志文件
7. 编译嵌入 Perl 脚本

Worker 进程接受、处理来自客户端的连接，提供反向代理和过滤功能以及其他 nginx 所具有的所有功能。由于 worker 进程是 web 服务器每日操作的实际执行者，所有对于 nginx 实例行为，系统管理员应该保持关注 worker 进程。

缓存加载进程负责检查磁盘上的缓存数据并且在内存中维护缓存元数据的数据库。基本上，缓存加载进程使用特定分配好的目录结构来管理已经存储在磁盘上的文件，为 nginx 提供准备，它会遍历目录，检查缓存内容元数据，当所有数据可用时就更新相关的共享内存项。

缓存管理进程主要负责缓存过期和失效。它在 nginx 正常工作时常驻内存中，当有异常则由 master 进程重启。

2.4 Nginx 配置

nginx 配置为简化日常维护而设计，并且提供给了简单的手段用户 web 服务器将来的扩展。配置文件是一些文本文件，通常位于/usr/local/etc/nginx 或/etc/nginx，主要配置文件通常命名为 nginx.conf。为了保持整洁，部分配置可以放到单独的文件中，再自动的包含到主配置文件。

Master 进程启动时读取和校验这些配置文件。由于 worker 进程是从 master 进程派生的，所以可以使用一份编译好、只读的配置信息。配置信息结构通过通过常见的虚拟内存管理机

制自动共享。

Nginx 配置具有多个不同的上下文，如，main、http、server、upstream、location(以及用于邮件代理的 mail)。这些上下文不重叠，例如一个 Location 指令块不能放入 main 指令块中。

2.5 Nginx Http 请求处理周期

1. 客户端发送 HTTP 请求。
2. nginx 核心从配置文件查找匹配该请求的位置，根据这个位置信息选择适当的阶段处理器。
3. 如果配置为反向代理，负载均衡器挑选一个上游服务器用于转发请求。
4. 阶段处理器完成工作，并且传递每个输出缓冲区给第一个过滤器。
5. 第一个过滤器传递输出给第二个过滤器。
6. 第二个过滤器传递输出给第三个等等。
7. 最终响应发送给客户端。

3.Openrestry

OpenResty 是一个基于 Nginx 的核心 Web 应用程序服务器，包含了大量的第三方 Nginx 模块和大部分系统依赖包。OpenResty 不是 Nginx 的分支，只是一个软件包，OpenResty 允许开发人员使用 Lua 语言构建现有的 Nginx C 模块，快速构建高流量的 Web 应用系统。OpenRestry 的目标是让 Web 服务直接运行在 Nginx 服务内部，充分利用 Nginx 的非阻塞 I/O 模型，不仅对 HTTP 客户端请求，甚至于对远程后端，诸如 MySQL,PostgreSQL,Memcached 以及 Redis 等都进行一致的高性能相应。

OpenResty 最大的特点是扩展 Lua 模块，用来定制非常复杂的业务逻辑。Ngx_lua 模块的原理：

1、 每个 Worker 创建一个 Lua VM，worker 内所有协程共享 VM
2、 将 Nginx I/O 源语封装后注入 Lua VM，允许 Lua 代码直接访问
3、 每个外部请求都有一个 Lua 协程处理，协程之间数据隔离
4、 Lua 代码调用 I/O 操作等异步接口时，会挂起当前协程（并保护上下文数据），而不阻塞 worker
5、 I/O 等异步操作完成时还原相关协程上下文数据，并继续运行。

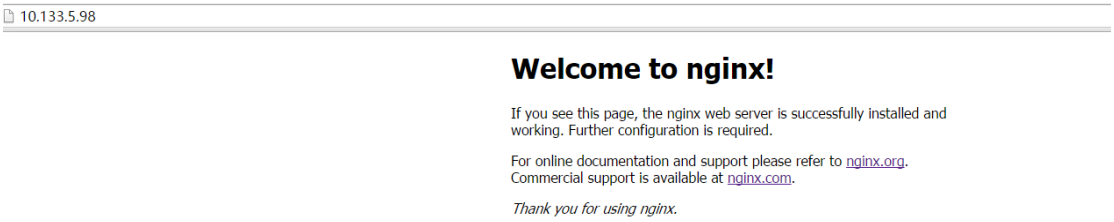
3.1 OpenResty 的使用

OpenResty 的安装不再介绍。OpenResty 核心采用 Nginx，所以 Nginx 的特性 OpenResty 都支持。例如 nginx.conf 的配置如下：

```
server {
    listen          80;
    server_name    localhost;
    location    / {
        root     html;
```

```
        index    index.html index.htm;
    }
    error_page      500 502 503 504    /50x.html;
    location = /50x.html {
        root      html;
    }
}
```

访问 80 端口，显示 nginx 页面：



3.2 OpenResty Lua 使用

使用案例：

```
server {
    listen          80;
    server_name    localhost;
    location / {
        default_type    text/html;
        content_by_lua    'ngx.say("<p>Hello,World</p>")';
    }
    error_page      500 502 503 504    /50x.html;
    location = /50x.html {
        root      html;
    }
}
```

显示页面如下：

页面内容： Hello,World ，由 lua module 中的 ngx.say 来提供。

3.3 OpenResty Mysql 的使用

Nginx 使用 lua 脚本可以直接访问 MySQL，取出数据，返回给浏览器。该功能使用 ngx_lua 模块和 lua 库 lua-resty-mysql（MySQL Client Driver）。下面是 nginx MySQL 使用的 nginx.conf:

```
lua_package_path "/home/root/lua-restry-mysql/lib/?.lua;;";
server {
    listen          80;
```

```
server_name    localhost;
location /test {
    default_type text/html;
    content_by_lua '
            ngx.say("<p>Hello,Test</p>")
            local mysql = require "resty.mysql"
            local db,err = mysql:new()
            if not db then
               ngx.say("failed to instantiate mysql:",err)
               return
            end
            db:set_timeout(1000)
            //MySQL 数据库连接，指定 Ip,port,user 及数据库名
            local ok, err, errno, sqlstate = db:connect{
                host = "127.0.0.1",
                port = 3306,
                database = "ngx_test",
                user = "root",
                max_packet_size = 1024*1024
            }
            //db:query 进行 MySQL 的操作，表的删除、创建、插入及查询
          local res,err,errno,sqlstate =
                db:query("drop table if exists cats")
            local res,err,errno,sqlstate =
                db:query("create table cats (id serial primary key,name varchar(5))")
          local res,err,errno,sqlstate =
                db:query("insert into cats values(1,\'Bob\'),(2,\'\'),(3,null)")
            local res,err,errno,sqlstate =
                db:query("select * from cats order by id asc",10)
            local cjson = require "cjson"
            ngx.say("result",cjson.encode(res)';}
```

3.4 OpenResty Memcached 的使用

Memcached 的安装启动后，配置 nginx.conf

```
lua_package_path "/home/root/lua-restry-memcached/lib/?.lua;;";
server {
    listen          80;
    server_name    localhost;

    location /test {
        default_type text/html;
        content_by_lua '
            local memcached = require "resty.memcached"
```

```
            local memc,err = memcached:new()      //创建 memc
            if not memc then
               ngx.say("failed to instantiate memc:",err)
               return
            end
            memc:set_timeout(1000)
            local ok, err = memc:connect("127.0.0.1",11211) //连接到 Memcached Server
            local ok,err = memc:flush_all()
            local ok,err = memc:set("dog",32)    //memcached 数据写入
            local res,flags,err = memc:get("dog")    //memcached  数据获取

            ngx.say("dog",res)
            local ok,err = memc:set_keepalive(10000,100)
        ';
    }
```

3.5 OpenResty Redis 的使用

redis 启动后，配置如下：

```
location /test {
        default_type text/html;
        content_by_lua '
            local redis = require "resty.redis"
            local cache = redis.new()
            local ok,err = cache:connect("127.0.0.1",6379)      //连接到 redis server
            if not ok then
                ngx.say(ngx.ERR,err)
                ngx.exit(ngx.HTTP_SERVICE_UNAVAILABLE);
            end
            local res = cache:set("hello","test redis")      //写入值
            local res = cache:get("hello")                   //查询
            if res==ngx.null then
                ngx.say("This is Null")
                return
            end
            ngx.say("redis get foo ",res)
        ';
    }
```

注意：MySQL,Memched,Redis 连接 lua 脚本在目录/usr/local/openresty/lualib/下，在上面配置中 require "resty.xxx"都和该脚本相关，可以在该目录下添加其他服务的执行脚本

3.6 OpenRestry 连接其他服务

从 Mysql,Memcached,Redis 服务的连接中，可以分析出，服务启用是通过下面来启用：

```
local mysql = require "resty.mysql"
```

从../lualib/resty 查找 mysql.lua(mysql 的服务脚本)，提供的方法如下所示：

方法名	描述
new	创建一个 server 的实例
set_timeout	设置 socket 的 time out 时间
connect	实例连接到 server，通过 ip 及 port
query	Mysql 的操作方法
send_query	命令的实际操作执行
close	关闭连接

从../lublib/resty 中查找到 memcached.lua，提供的方法如下所示：

方法名	描述
new	创建一个 server 的实例
set_timeout	设置 socket 的 time out 时间
connect	实例连接到 server，通过 ip 及 port
get/set/add/replace/append/prepend	Memcache 操作
_store	命令的实际执行
close	关闭连接

其他服务的开发，要实现基本的方法，包括 new,connect,close 及命令的执行

参考文献：

https://github.com/openresty/lua-resty-mysql

https://github.com/openresty/lua-resty-memcached

http://redis.io/download

http://blog.csdn.net/wonderisland/article/details/37923831