Lua 语言在轻量级 Web 服务器设计中的应用①

林　巧[1], 杨坚坚[2]

[1](浙江师范大学 数理与信息工程学院, 金华 321004)
[2](浙江中塑在线股份有限公司, 余姚 315490)

摘　要: Lua 是一种小巧的脚本语言,它的易扩展性与整合性,使得它可以与别的语言很好的融合在一起,实现各种各样的需求,因此它的应用非常的广泛. 本文利用 Lua 脚本语言与标准 C 语言相结合, 在 Linux 操作系统上设计并实现了一个轻量级的 Web 服务器程序. Lua 脚本语言的引入, 使 Web 服务器不仅易于配置和安装, 而且可以在那些无法负担 IIS 的主机上顺畅地运行; 还进一步提高了服务器的运行速度, 并增强了服务器的灵活性和扩展性.

关键词: Lua; 脚本语言; Linux 操作系统; 轻量级 Web 服务器; HTTP 协议

Application of Lua Language in Lightweight Web Server Design

LIN Qiao[1], YANG Jian-Jian[2]

[1](College of Mathmatics Physics and Information Engineering, Zhejiang Normal University, Jinhua 321004, China)
[2](Plastic in Zhejiang Online Co., Ltd, Yuyao 315490, China)

Abstract: Lua is a tiny script language. Its good expansibility and integration make it easy to integrate to other languages and then implements all kinds of applications. So it is widely used. This paper designs and realizes a lightweight web server procedure in linux operating system by using lua scripting language and standard c language. Using the Lua scripting language, not only makes a Web server easy to configure and install, but also can run smoothly in those who cannot afford the IIS hosts; it further improves the running speed of the server, and enhance the flexibility and extensibility of the server.

Key words: Lua; scripting language; Linux operating system; lightweight Web server; HTTP protocol

1　引言

随着计算机网络技术的飞速发展, Internet 已经成为目前世界上最大的计算机互联网络, 而 Web 服务是 Internet 上使用得最为广泛的服务之一. 通过它可以为用户提供包括图象、声音、视频等多媒体信息的 HTML 页面服务. Web 服务的基础就是提供 Web 服务的服务器程序, 当前存在很多不同的 Web 服务器, 据调查, 使用最为频繁的服务器是 Apache[1]和 IIS(Internet Information Server). 因为这两种 Web 服务器都经过高度的锤炼, 而且有着广泛的技术支持和配套资源, 但这并没有阻碍其它类型的 Web 服务器的发展. 为了适应不同类型的应用, 且具有相对比较灵活的服务, 所以就出现了较小型的即轻量级的 Web 服务器. 这些轻量级的 Web 服务器的速度反而更快, 具有简单、易于安装、流线化、要求低和健壮等特点, 它们有着一些大型服务器不可比拟的特性, 而且轻量级 Web 服务器还可作为那些主要服务器的有效补充[2].

虽然轻量级 Web 服务器有很多共同之处, 但是各有各的不同. 大多数轻量级 Web 服务器是用 C 语言编写的, 但是实践证明, 其他语言也可以成功地用于实现 Web 服务器[2], 这些语言包括 Erlang、Java、Lisp、Lua、Perl、Python 和 Tcl. 本文利用 Lua 脚本语言与标准 C 语言相结合, 在 Linux 操作系统上设计并实现了一个轻量级的 Web 服务器程序. 由于 Lua 语言能够很好地与 C 语言融合在一起, 实现各种各样的需求, 所以 Web 服务器的实现采用 Linux 下 Lua 和 C 语言混合

① 基金项目:浙江省教育厅项目(Y201328256)
收稿时间:2015-11-09;收到修改稿时间:2015-12-12　[doi:10.15888/j.cnki.csa.005223]

编程, 使 Web 服务器不仅易于配置和安装, 而且可以在那些无法负担 IIS 的主机上顺畅地运行; 还进一步提高了服务器的运行速度, 并增强了服务器的灵活性和扩展性.

2 Lua语言介绍

Lua 语言是一个简洁、轻量、可扩展的动态脚本语言. Lua 语言最初的设计目的是在嵌入软件开发中, 为应用程序提供灵活的扩展和定制功能. Lua 脚本可以很容易的被 C/C++代码调用, 也可以反过来调用 C/C++的函数, 这使得Lua在C/C++应用程序中能被广泛应用. Lua 由标准 C 编写而成, 几乎在所有操作系统和平台上都可以编译、运行[3].

在众多的脚本语言中, Lua 不仅与它们有着共同的特点, 还具备以下独有的特点[4]:

1)可扩展性. Lua 可以通过新类型和函数来扩展其功能, 它的扩展性非常卓越, 以至于常被用作搭建领域语言的工具. Lua 很容易与 C/ C++、C#、Java、Ada、Smalltalk、Fortran 以及其他语言集成.

2)简单易学. Lua 本身小巧、简单, 内容少但功能非常强大, 易于学习, 很容易实现一些小的应用.

3)高效率. Lua 具有很高的执行效率, 有统计表明, Lua 是目前平均效率最高的脚本语言.

4)与平台无关. Lua 几乎可以运行在所有已知的系统上. Lua 不是通过条件编译实现平台无关的, 而是完全使用标准C, 这意味着只要有 ANSI C 编译器你就可以编译并使用 Lua.

Lua 语言不仅是一种易扩展语言, 也是一种易整合语言. Lua 支持基于组件的软件开发方法, 该方法可以通过整合现有的高级组件来创建新的应用程序. 这些组件可以是已编译好的, 也可以是使用静态类语言(如C/C++)编写的, 而 Lua 则是整合各个组件的粘合剂. 除了作为整合语言外, Lua 自身也是一个功能强大的语言, 它可以编辑组件甚至可以完全使用 Lua 来创建组件[5].

3 轻量级Web服务器的设计

Web 服务器所需要完成的主要任务是: ①接收客户端请求; ②解析客户端请求; ③响应客户端请求; ④向客户端回送请求的结果. 当客户端需要请求特定的 URL 时, 它将与服务器建立 TCP/IP 连接, 并通过 HTTP 协议发送请求至 Web 服务器[6]. Web 服务器接收到客户端的请求之后将生成响应内容, 同时将该响应内容通过连接返回给客户端.

3.1 Web 服务器的工作流程

Web 服务器的工作流程如图 1 所示. Web 服务器在接受、处理客户端的请求之前, 首先要进行初始化即创建 socket 端口(默认为 80 端口), 用来侦听是否有客户发出连接请求. 配置、绑定端口之后, 就开始侦听, 若没有客户发出连接请求, 则继续侦听; 若有客户发出连接请求, 则创建子进程, 用函数 fork( )创建子进程. 子进程用来处理客户向服务器发出的请求, 主进程则继续侦听是否有客户发出连接请求. 这样服务器就进入了请求——响应的循环阶段, 当服务器接收到 GET 请求命令后会对该请求进行检查, 将请求的 URL 映射为特定的文件, 如果该文件存在, 则将该文件发送返回给客户端, 如果不存在, 则返回一个错误消息. 当服务器子程序处理完客户请求后, 关闭子程序. 如果客户请求关闭连接, 则服务器关闭与之连接的 socket 端口, 如果没有客户请求关闭, 则继续侦听.

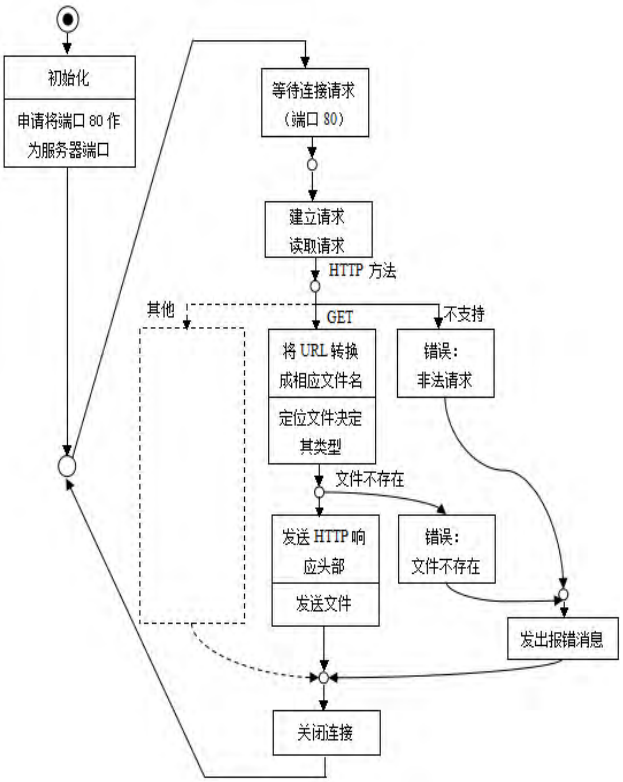

图 1 Web 服务器的工作流程图

3.2 轻量级 Web 服务器的设计思想

(1)用 Lua 来实现配置文件功能. 因为 Lua 脚本非

常简洁清晰, 一看就懂, 是非常适合做配置文件的, Lua 代替 XML、ini 等文件格式, 因而使 Web 服务器更加容易理解和维护, 更易于配置和安装.

(2)采用一个主进程、多个工作进程的进程模型. 主进程在 Web 服务器中做一些初始化工作, 如读取配置文件、绑定端口、屏蔽信号等, 当创建出多个工作进程后, 进入休眠状态. 由于工作进程是继承主进程的, 所以工作进程进入监听事件状态.

(3)采用国外著名 Web 服务器 Nginx 的内存池思路. 对于 Web 服务器来说, 频繁地进行内存方面的操作会导致性能极大的下降, 所以要尽量减少内存方面的操作. 使用内存池的目的很明确, 为了避免频繁地申请小块内存, 提前申请好一大块内存, 供小块内存在这里面申请, 从而降低内存碎片等问题.

(4)使用请求池来减少频繁地内存分配. 每一个工作进程都有一个请求池, 当一个新的请求进入后, 就直接从请求池分配给这个请求. 请求池用到了数组链表, 从而使请求能够循环使用, 这种做法也可以减少了内存方面的操作, 从而提高 Web 服务器的性能.

(5)使用 epoll 做 I/O 复用模型, 即用 epoll 来监听各类事件. epoll 是 Linux 内核为处理大批量文件描述符而作了改进的 poll, 是 Linux 下多路复用 I/O 接口 select/poll 的增强版[7], 它能显著提高程序在大量并发连接中只有少量活跃的情况下的系统 CPU 利用率.

(6)利用 Lua 脚本语言能比较方便的实现字符串处理的特性来生成 HTTP 响应头. 服务器对客户端发来的请求进行处理并作出响应, 是个非常重要的过程.

(7)服务器端缓存的实现是借助 Lua 脚本实现的, 所有缓存都会放到 Lua 的一个 table 变量中. Lua 的功能非常强大, 它可以执行系统的指令[8], 比如: os.clock ()、os.date ([format [, time]])、os.difftime(t2, t1)、os.execute ([command])、os.remove (filename)、os.time ([table])等命令, 能方便的清理对应的缓存.

4 Lua在Web服务器设计中的应用

本 Web 服务器的设计是采用 Linux 操作系统下 Lua 语言和 C 语言混合编程, 下面着重介绍 Lua 在设计中的应用主要体现在以下几个方面:

(1)用 Lua 来实现配置文件功能

(2)用 Lua 生成 HTTP 响应头

(3)用 Lua 实现 Web 服务器缓存

4.1 用 Lua 来实现配置文件功能

4.1.1 配置文件的生成

目前, 我们常用的配置文件格式有 XML 和 ini. XML 层次分明, 但其缺点就是写起来太繁杂, 对一些关键字如< >之类的处理比较特殊. 而 ini 呢, 配置不够灵活, 只有简单的段-键-值模式, 对于一些多层结构的配置, 或者一些列表类型的配置, 就显得力不从心了, 往往需要编码人员自己分割字符串. Lua 脚本非常简洁清晰, 是非常适合做配置文件的, 无论是映射表型配置, 还是列表型配置, 还是简单的键值配置, Lua 都可以完全胜任. 而且, Lua 格式清新简洁, 一看就懂.

本设计中配置文件的文件名为: zs.config, 其详细内容如图 2 所示, 虽然其文件后缀是.config, 但是它本质上是一个 Lua 脚本文件, 之所以以.config 作为后缀, 主要是为了更好的辨别. 因为 zs.config 这个文件是一个 Lua 脚本文件, 所以它必须符合 Lua 脚本语言的语法. 否则, Web 服务器初始化时, 将只能获取系统默认配置参数.

```
workers = 1
worker_connections = 10240

pid = "./log/znuserv.pid"
is_deamon = 0

use_event_timeout = 1
event_timeout = 400

listen_port = 9867
php_listen_port = 81
server_name = "localhost"

index_files = {"index.html", "index.php"}
root_dir = "./html"
page_404 = "./html/404.html"

use_cache = 0
cache = 100
```

图 2　配置文件 zs.config

4.1.2 配置文件的读取

首先 Lua 要在 C 代码中创建一个状态变量 L, L 是一个指向结构体类型为 lua_State 的指针变量, 该结构体将负责对 Lua 的运行状态进行维护[8]. 代码如下:

```
lua_State *L;
L=luaL_newstate( );
if ( luaL_dofile (L, config_file) )
{   zs_err ("Couldn't load file: %s\n",
```

```
        lua_tostring(L,-1) );
        lua_close(L);
        return ZS_NO;
    }
```

通过这个状态变量, C 就可以访问 Lua 中的数据了. 在主程序 znuserv.c 中, 获取配置文件的函数为 zs_get_config. 它根据事先设定好的配置参数一次一次地遍历, 每次遍历, 就通过 Lua CAPI 获取在配置文件 zs.config 中对应的配置参数名, 如图 3 所示.

```
enum {LISTEN_PORT, SERVER_NAME, INDEX_FILES,
      ROOT_DIR, WORKERS, WORKER_CONNETIONS,
      EVENT_TIMEOUT, PHP_LISTEN_PORT,CACHE,
      IS_DEAMON, USE_CACHE, PAGE_404, PID,
      USE_EVENT_TIMEOUT, TIMER_STRUCTURE};
const char *config_option[] = {
    "listen_port",
    "server_name",
    "index_files",
    "root_dir",
    "workers",
    "worker_connections",
    "event_timeout",
    "php_listen_port",
    "cache",
    "is_deamon",
    "use_cache",
    "page_404",
    "pid",
    "use_event_timeout",
    "timer_structure",
    "NULL"
};
```

图 3 配置参数

获取配置参数的整个过程是遍历完字符串数组 config_option 中的所有元素, 直到遇到 NULL. 当获取某个参数时, 就会通过 Lua 的 CAPI 函数去读取 zs.config 文件中对应的参数值. 从 zs.config 读取符合 Lua 语法的变量, 然后存放到 Lua 的栈中, 再在 C 代码中进行对 Lua 栈的操作, 取出需要的变量值.

在 zs_get_config 函数中, 有一个 switch 语句, 其参数是从 0 开始的, 对应于枚举类型中的索引号. 这样根据 switch 中的参数, 可以获取出索引对应的参数值. 以获取 workers 这个配置参数为例, 在 switch 语句中的 case 值为 WORKERS, 那么, 就会执行获取 workers 配置参数值的 Lua 函数. 代码如下:

```
case WORKERS:
    lua_getglobal ( L, "workers" );
    if ( (tmp=lua_tonumber(L,-1) ) < 0
        || tmp>ZS_MAX_PROCESSES )
    {   tmp=DF_WORKERS;
        zs_err ( "ERROR.The argument *workers*
        is error. It has been set default value.\n" ); }
        ctx->conf->workers = tmp;     break;
```

函数 lua_getglobal 是从状态变量 L 中获取变量名为 workers 的值, 获取 workers 的值后, Lua 会把这个值放到 Lua 的栈中. 所以, C 代码想要获取其值, 就要对栈进行操作, 又因为 Lua 是弱类型的, C 语言是强类型的, 所以在取栈顶时, 要对其进行类型转换, 这里的 workers 变量是整型, 所以对栈操作时是使用 lua_tonumber 函数.

4.2 用 Lua 生成 HTTP 响应头

4.2.1 HTTP 响应头的生成

因为 Lua 脚本语言能比较方便的实现字符串处理, 所以本设计中的 HTTP 响应头采用 Lua 来生成. 客户端向服务器端发送一个请求, 服务器端以一个状态行作为响应, 响应的内容包括: 消息协议的版本、成功或者错误编码、服务器信息、实体元信息以及必要的实体内容. 当生成完响应头后, 就要把响应体加上去, 形成完整的一个响应消息. 详见图 4 所示.

src/zs_header.lua

```
local header_str = "HTTP/1.1 "
header_str = header_str .. c
if c == 200 then
    header_str = header_str .. " OK\r\n"
elseif c == 304 then
    header_str = header_str .. " Not Modified\r\n"
elseif c == 400 then
    header_str = header_str .. " Bad Request\r\n"
elseif c == 404 then
    header_str = header_str .. " Not Found\r\n"
else
    header_str = header_str
end
header_str = header_str .. "Server: znuserv\r\n"
header_str = header_str .. "Content_Type: text/html\r\n"
header_str = header_str .. "Connection: close\r\n"
header_str = header_str .. "Last-Modified: " .. ld .. "\r\n\r\n"
```

图 4 Lua 脚本生成 HTTP 响应头

其中的变量 c 为 C 代码中传递进去的 HTTP 响应头的状态码, 变量 ld 为文件的最后一次修改的时间. 这里, HTTP 的响应头的状态码只支持 200、304、404. 对于 404, 需要对请求中的 URL 在服务器的文档路径上查找才能得出; 对于 304, 需要根据变量 if_modified_since 来判断, 如果请求的 URL 的文件修改时间与变量 if_modified_since 一致, 那么状态码就为 304; 如果一切都正常那么就发送 200.

4.2.2　HTTP 响应头的发送

响应头的发送与响应内容的发送不在一起, 但它们发送的方式很类似, 都是非阻塞写操作, 代码如图 5 所示. 其中变量 has_written 是记录目前已经写了多少字节, 循环写到套接字中, 直到返回错误为 EAGAIN 为止.

src/zs_http.c

```
while (len != req->has_written) {
    n = write(req->sockfd, lua_tostring(ctx->Hdr, -1) +
     req->has_written, len - req->has_written);
    if (n == -1) {
        if (errno == EAGAIN) {
            break;
        } else {
            return;
        }
    } else if (n == 0) {
        zs_err("sending header. Client close.\n");
        return ;
    } else if (n > 0) {
        req->has_written += n;
    }
}
```

图 5　非阻塞写 HTTP 响应头

4.3 用 Lua 实现 Web 服务器缓存

4.3.1 存缓存

本设计中 Web 服务器的缓存是存放在 Lua 的一个 table 变量中的. 在缓存 table 中, 需要为每个页面保存其文件的修改时间, 通过页面文件的修改时间与缓存中的修改时间进行比较, 若相同, 则页面没有被修改过, 若不同, 则要对缓存中的页面进行更新, 再发送给客户端. 存缓存的 C 语言部分如图 6 所示.

Lua 语言部分代码如下:

```
Function store_cache ( key, content, md )
local page = { }
page.p = 1
page.score =1
page.time = os.time ( )
page.content = content
page.modified_time = md
cache[key] = page
index =index + 1
cache[index] =cache[key]
end
```

src/zs_cache.c

```
if (req->is_static_file == 0) {
    zs_read_res_cnt(ctx, req);
    lua_getglobal(ctx->L, "store_cache");
    lua_pushstring(ctx->L, req->pf);
    lua_pushstring(ctx->L, req->res_cnt);
    lua_pushnumber(ctx->L, req->modified_time);
    if (lua_pcall(ctx->L, 3, 0, 0) != 0) {
        zs_err("store cache lua_pcall error.\n");
        return ZS_ERR;
    }
} else {
    lua_getglobal(ctx->L, "store_b_cache");
    lua_pushstring(ctx->L, req->pf);
    lua_pushnumber(ctx->L, req->modified_time);
    if (lua_pcall(ctx->L, 2, 0, 0) != 0) {
        zs_err("store b cache error\n");
        return ZS_ERR;
    }
}
```

图 6　存缓存 C 部分

从 C 代码传到 Lua 函数 store_cache 中的参数为请求页面的路径+页面名、页面内容和文件的修改时间. 路径+页面名为索引, 其它的内容包括点击量 p、得分 score、第一次点击的时间 time、页面内容 content 和文件修改时间 modified_time.

4.3.2 取缓存

取缓存时, 首先需要比较当前文件的修改时间与缓存中的修改时间是否一致, 如果不一致, 就要先进行缓存的更新操作. 取缓存的代码如图 7 所示. 其中, 变量 cache_buf 用于保存从缓存中取出来的数据, 这里的数据并不是保存在 res_cnt 中的, 目的主要是区别内容的来源.

5　基准性能测试

对于 Web 服务器来说, 性能是一个非常重要的因素. 下面是使用 http_load 这个基准测试软件分别对本轻量级 Web 服务器 znuserv 和国外著名 Web 服务器 Nginx 进行对比测试, 测试结果如下:

src/zs_cache.c

```
lua_getglobal(ctx->L, "get_cache");
lua_pushstring(ctx->L, req->pf);
n = lua_pcall(ctx->L, 1, 2, 0);
if (n == LUA_OK) {
    req->cache_buf = zs_palloc(req->pool, (1 << 19));
    req->res_length = lua_tonumber(ctx->L, -1);
    req->has_read = 0;
    for (i = 0; i < req->res_length; i++) {
        req->cache_buf[i] = lua_tostring(ctx->L, -2)[i];
    }
} else {
    return ZS_ERR;
}
```

图 7 取缓存

```
10000 fetches, 52 max parallel, 1.33e+06 bytes, in 0.795509 seconds
133 mean bytes/connection
12570.6 fetches/sec, 1.67189e+06 bytes/sec
msecs/connect: 0.275106 mean, 1.943 max, 0.095 min
msecs/first-response: 0.899719 mean, 3.003 max, 0.493 min
HTTP response codes:
  code 200 -- 10000
```

图 8 znuserv 的测试结果

```
10000 fetches, 217 max parallel, 2.01e+06 bytes, in 1.61064 seconds
201 mean bytes/connection
6208.7 fetches/sec, 1.24795e+06 bytes/sec
msecs/connect: 7.94573 mean, 1000.72 max, 0.111 min
msecs/first-response: 7.77103 mean, 225.292 max, 0.435 min
HTTP response codes:
  code 200 -- 10000
```

图 9 Nginx 的测试结果

从图 8 和图 9 中可以得到, znuserv 每秒可获取 12570.6 个页面(12570.6 fetches/sec), 而 Nginx 每秒可获取 6208.7 个页面(6208.7 fetches/sec). 可见在 http_load 的测试结果中, znuserv 的性能要比 Nginx 要好一些. 由于基准测试软件也可能存在误差, 所以上面的测试结果仅仅反应着在同一台机器上的某一个时间段的性能, 并不能将此数据断定为各自 Web 服务器的性能, 但至少能表明本轻量级 Web 服务器 znuserv 的性能可以接近甚至超越成熟的 Web 服务器 Nginx.

6 结语

文中对 Lua 的特点和应用进行了详细分析, 设计并实现了一个在 Linux 操作系统下的轻量级 Web 服务器程序. Lua 在设计中实现了配置文件的功能, 代替 XML、ini 等文件格式, 因而更加容易理解和维护. 此外, Lua 语言所具有的良好的安全保证、自动内存管理、简便的字符串处理等功能, 在设计中都有具体体现. 总而言之, 用 Lua 脚本语言实现的轻量级 Web 服务器, 易于配置和安装, 进一步提高了系统的灵活性和扩展性; 轻量级 Web 服务器可以适用于市场领头产品和其他“重量级”服务器无法胜任的情况, 可以在小功率的主机上良好地运行.

参考文献

1 Brittenham P. Web Services Development Concepts (WSDC1.0), IBM Software Group, 2001.5.

2 马毅.轻量级 Web 服务器的实现与应用[学位论文].西安:西北大学,2008.

3 黄海云,季刚.Lua 语言在 Telephony 辅助开发模拟器中的应用.科技信息,2008,(34):579-580.

4 苗新亮,刘栋,雷波,杨远辉.Lua 脚本语言在安全设备管理系统中的应用.信息安全与通信保密,2014,(2):92-94.

5 石琦.Lua 在自动测试中的应用.自动化技术与应用,2009,28(12):31-34.

6 封相远.基于 Linux 操作系统的 Web 服务器的设计与实现[学位论文].天津:天津大学,2007.

7 Stevens WR, Rago SA. 尤晋元,张亚英,戚正伟,译.UNIX 环境高级编程.北京:人民邮电出版社,2013.

8 Ierusalimschy R. 周惟迪,译.Lua 程序设计.北京:电子工业出版社,2008.