Apache网站服务

访问网站的基本流程

我们每天都会使用web客户端上网浏览网页，最常见的web客户端就是web浏览器，如通用的微软IE，以及技术人员偏爱的火狐浏览器、谷歌浏览器等。当我们在web浏览器输入网站地址时（例如`www.baidu.com`），很快就会看到网站的内容。这一切似乎看起来很神奇，那么在其背后到底时怎样的实现流程呢？也许普通的上网者无需关注，但作为一个IT技术人员，特别时合格的Linux运维人员，就需要清晰的掌握了。

第一步：客户端用户在浏览器输入`www.baidu.com`网址地址，回车时，系统会显示`www.baidu.com`的界面

1. 客户端，浏览器输入网址信息点击回车
2. 客户端，完成域名解析过程DNS
3. 浏览器缓存
4. 系统缓存
5. 路由器缓存————————以上均为DNS客户端的缓存！！
6. ISP DNS缓存
7. 根域名服务器
8. 顶级域名服务器
9. 主机名服务器
10. 保存结果至缓存
11. 客户端，直接访问响应网址服务器，建立tcp三次握手过程
12. 客户端，访问网址服务器，发送HTTP请求报文（多次）
13. 服务端，响应客户端请求，回复HTTP响应报文（多次）
14. 客户端，浏览器查看到网址页面，浏览器完成html解析
15. 客户端，结束网站访问

HTTP协议

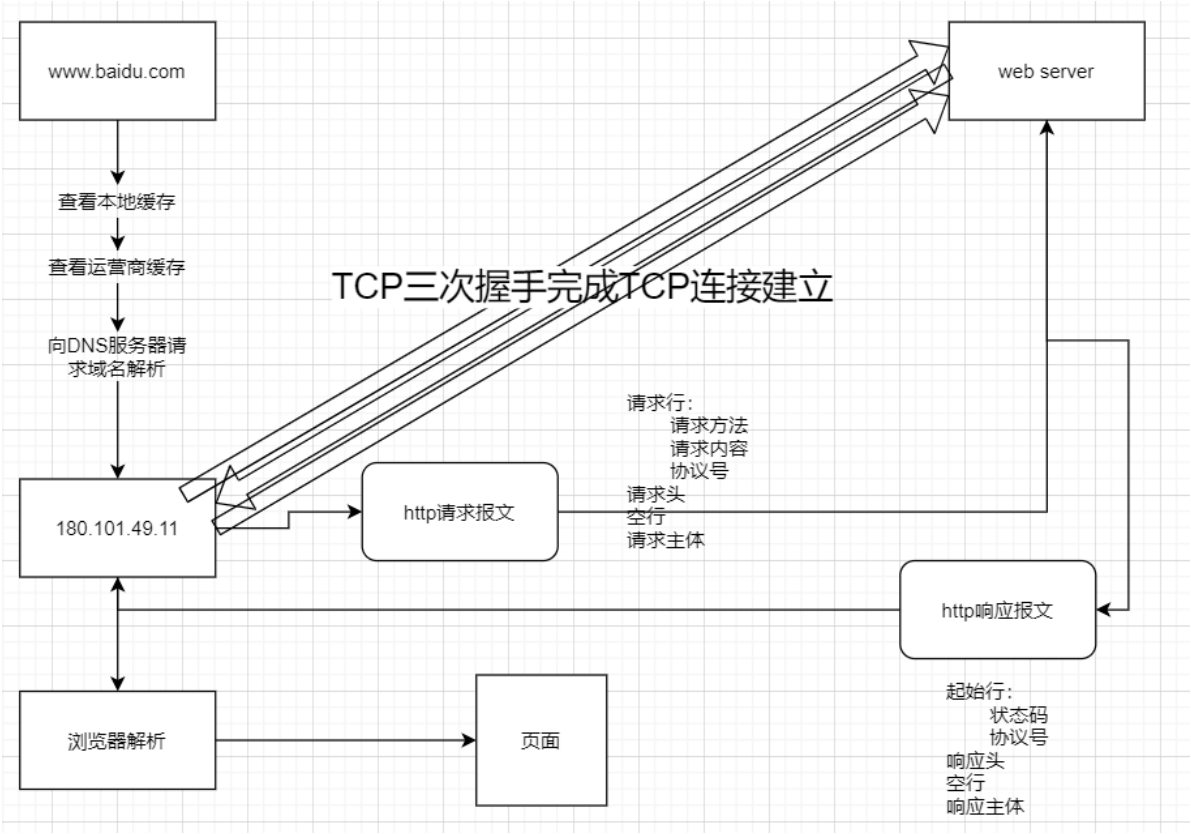
简介

- HTTP协议，全称HyperText Transfer Protocol，中文名为超文本传输协议，是互联网中最常用的一种网络协议。HTTP重要应用之一是www服务。涉及HTTP协议最初的目的就是提供一种发布和接受HTML页面的方法。HTTP协议是互联网上最常用的通信协议之一，它有很多的应用，但最流行的就是用于web浏览器和web服务器之间的通信，即www应用或称web应用
- HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件,图片文件, 查询结果等）的应用层协议
- HTTP协议工作于C/S或B/S架构。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。 Web服务器根据接收到的请求后，向客户端发送响应信息。

URI和URL

- 统一资源标志符URI就是在某一规则下能把一个资源独一无二地标识出来。
- URL是一种特殊类型的URI，全称是UniformResourceLocator(统一资源定位符),是互联网上用来标识某一处资源的 地址。
    - 比如，http://www.runoob.com/http/http-tutorial.html

HTTP协议请求和响应过程



1. HTTP请求报文

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

- 请求行
  - 请求方法
    - GET：请求指定的页面信息
    - HEAD：类似get请求，但是返回仅仅是头部信息，一个使用场景是在下载一个大文件前先获取其大小再决定是否要下载, 以此可以节约带宽资源.
    - POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）
    - PUT：从客户端向服务器传送的数据取代指定的文档内容
    - DELETE：请求服务器删除指定的页面
    - CONNECT： 预留给能够将连接改为管道方式的代理服务器
    - OPTIONS：允许客户端查看服务器的性能
    - TRACE： 回显服务器收到的请求，主要用于测试或诊断!
    - GET方法和POST方法的区别：1.GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如`EditPosts.aspx? name=test1&id=123456`. POST方法是把提交的数据放在HTTP包的Body中。 2.GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制。 3.GET方式需要使用`Request.QueryString`来取得变量的值，而POST方式通过`Request.Form`来获取变量的值。 4.GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL 上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码。
  - 请求信息

        - index.html(首页文件)
    - 请求协议
        - HTTP0.9：仅支持GET方法，仅能访问HTML格式的资源
        - HTTP1.0：增加POST和HEAD方法，MIME支持多种数据格式，开始支持Cache，支持tcp短连接
        - HTTP1.1：支持持久连接（长连接），一个TCP连接允许多个请求，新增PUT、PATCH、DELETE等
        - HTTP2.0：性能大幅提升，新的二进制格式，多路复用，header压缩，服务端推送。
- 请求头
    - 客户端有关信息介绍说明
- 空行
    - 和请求主体分隔开
- 请求主体
    - 使用get方法时，没有请求主体
    - 使用post方法时，有请求主体信息

2. 响应报文

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

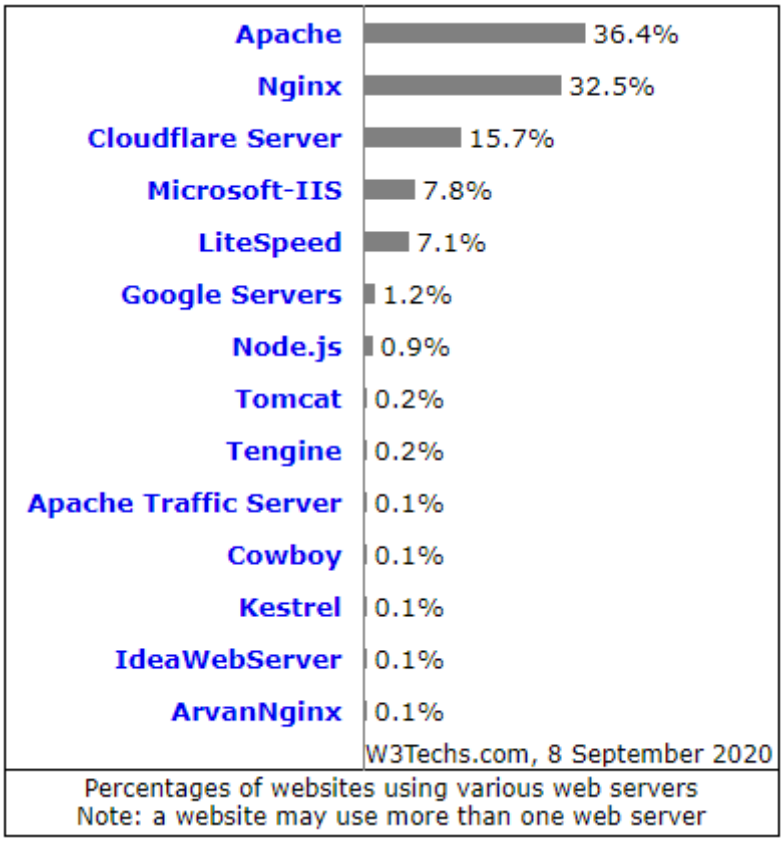
- 起始行
    - 协议版本
    - 状态码
        - 1xx：指示信息——表示请求已经接收，继续处理
        - 2xx：成功——表示请求已经被成功接收、理解、接收
        - 3xx：重定向——要完成请求必须进行更进一步的操作
        - 4xx：客户端错误——请求的语法有错误或请求无法实现
        - 5xx：服务端错误——服务器未能实现合法的请求!
        - 200：OK请求已经正常处理完毕
        - 301：请求永久重定向
        - 302：请求临时重定向
        - 304：请求被重定向到客户端本地缓存
        - 400：客户端请求存在语法错误
        - 401：客户端请求没有经过授权
        - 403：客户端的请求被服务器拒绝，一般为客户端没有访问权限
        - 404：客户端请求的URL在服务端不存在
        - 500：服务端永久错误
        - 503：服务端发生临时错误
- 响应头部
- 空行
- 响应主体

网站测评指标

- IP：根据用户访问的源IP信息进行统计
  - 统计一天内访问网站最对的前十个地址
- PV：根据用户页面访问量进行统计
  - 统计一个用户访问页面数量最对的前十个页面
- UV：根据用户访问的cookie信息，统计用户访问数量
- 网站并发：单位时间内同时处理的请求数
- 对网站进行压力测试
  - `yum install -y httpd-tools`
  - `ab -n 100 -c 10 https://ip/`

常用的网站服务软件

- https://w3techs.com/technologies/overview/web_server



Apache服务

Apache介绍

- Apache是什么
  - Apache HTTP Server简称为Apache，是Apache软件基金会的一个高性能、功能强大、见状可靠、又灵活的开放源代码的web服务软件，它可以运行在广泛的计算机平台上如Linux、Windows。因其平台型和很好的安全性而被广泛使用，是互联网最流行的web服务软件之一
- 特点
  - 功能强大
  - 高度模块化
  - 采用MPM多路处理模块
  - 配置简单
  - 速度快

    - 应用广泛
    - 性能稳定可靠
    - 可做代理服务器或负载均衡来使用
    - 双向认证
    - 支持第三方模块
- 应用场合
    - 使用Apache运行静态HTML网页、图片
    - 使用Apache结合PHP、Linux、MySQL可以组成LAMP经典架构
    - 使用Apache作代理、负载均衡等
- MPM工作模式
    - prefork：多进程I/O模型，一个主进程，管理多个子进程，一个子进程处理一个请求。
    - worker：复用的多进程I/O模型，多进程多线程，一个主进程，管理多个子进程，一个子进程管理多个线程，每个 线程处理一个请求。
    - event：事件驱动模型，一个主进程，管理多个子进程，一个进程处理多个请求。

httpd 命令

httpd 为apache http server服务提供的工具

- -c：在读取配置文件前，先执行选项中的指令。
- -C：在读取配置文件后，再执行选项中的指令。
- -d<服务器根目录>：指定服务器的根目录。
- -D<设定文件参数>：指定要传入配置文件的参数。
- -f<设定文件>：指定配置文件。
- -h：显示帮助。
- -l：显示服务器编译时所包含的模块。
- -L：显示httpd指令的说明。
- -S：显示配置文件中的设定。
- -t：测试配置文件的语法是否正确。
- -v：显示版本信息。
- -V：显示版本信息以及建立环境。
- -X：以单一程序的方式来启动服务器。

安装并设置第一个站点

```
[root@localhost ~]#yum install -y httpd
[root@localhost ~]#echo '<h1>It works!</h1>' > /var/www/html/index.html
[root@localhost ~]#systemctl start httpd
```

1. 检查防火墙和selinux是否关闭

```
[root@localhost ~]#systemctl stop firewalld
[root@localhost ~]#systemctl status firewalld
[root@localhost ~]#setenforce 0
[root@localhost ~]#getenforce
```

2. 检查端口是否存在

```
[root@localhost ~]#ss -tanl | grep 80
```

3. 查看进程是否存在

```
[root@localhost ~]#ps -ef | grep http
```

4. 在服务器本地进行测试

```
[root@localhost ~]#wget <http://IP地址>
[root@localhost ~]#curl <IP地址>
```

- 文件说明

```
/etc/httpd/：主配置文件目录
/etc/httpd/conf/httpd.conf：服务配置文件
/etc/httpd/conf.d/：服务配置目录（模块化）
/etc/httpd/conf.modules.d/：模块配置目录
/etc/sysconfig/httpd：守护进程配置文件
/usr/lib64/httpd/modules/：可用模块
/usr/sbin/：相关命令目录
/var/log/httpd/：日志目录
/var/www/：站点目录
```

- 主配置文件

```
##主配置说明##
[root@node3 ~]# grep "^[^ #]" /etc/httpd/conf/httpd.conf
ServerRoot "/etc/httpd" # 服务器的根
Listen 80 # 监听的端口
Include conf.modules.d/*.conf # 包含模块
User apache # 用户
Group apache # 属组
ServerAdmin root@localhost # 服务器管理员
DocumentRoot "/var/www/html"
ErrorLog "logs/error_log" # 错误日志
LogLevel warn # 日志等级
EnableSendfile on # 开启
IncludeOptional conf.d/*.conf # 虚拟服务器配置文件
说明：<></>此类称之为容器，针对某个容器做配置
```

持久连接

持久连接，每个资源获取完成后不会断开连接，而是继续等待其它的请求完成

- 默认参数

```
KeepAlive默认是on，默认的超时时间是5秒。
KeepAliveTimeout # 连接超时
MaxKeepAliveRequests # 最大保持连接请求
```

- 测试持久连接

```
[root@server ~]# yum install telnet -y
[root@server ~]# echo 'this is test!' > /var/www/html/index.html
[root@server ~]# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

```
GET / HTTP/1.1
Host:127.0.0.1

HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 14:17:29 GMT
Server: Apache/2.4.6 (CentOS)
Last-Modified: Wed, 14 Jul 2021 14:16:40 GMT
ETag: "e-5c71600ca9dad"
Accept-Ranges: bytes
Content-Length: 14
Content-Type: text/html; charset=UTF-8

this is test!
Connection closed by foreign host.

```

- 修改持久连接的参数，重启httpd，再次测试

```
[root@server ~]# cat /etc/httpd/conf.d/keeplive.conf
KeepAlive on
KeepAliveTimeout 30
MaxKeepAliveRequests 100
[root@server ~]# systemctl restart httpd.service
[root@server ~]# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET / HTTP/1.1
Host:127.0.0.1

HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 14:20:58 GMT
Server: Apache/2.4.6 (CentOS)
Last-Modified: Wed, 14 Jul 2021 14:16:40 GMT
ETag: "e-5c71600ca9dad"
Accept-Ranges: bytes
Content-Length: 14
Content-Type: text/html; charset=UTF-8

this is test!
Connection closed by foreign host.
```

多路处理模块

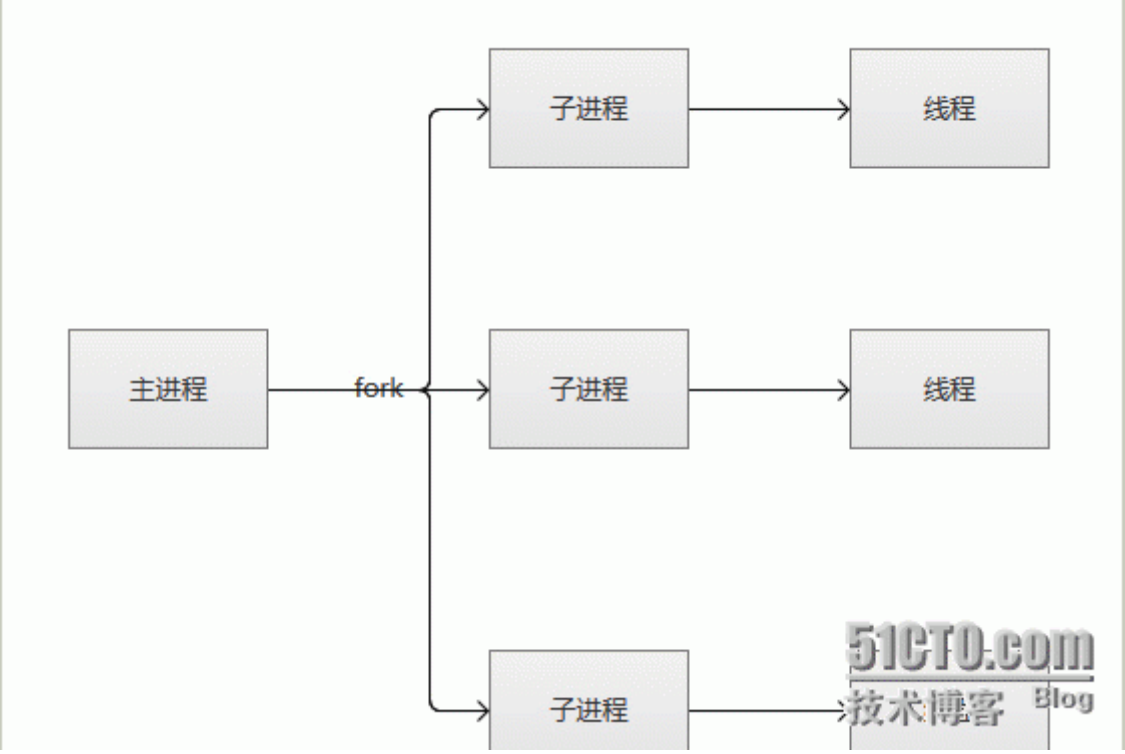
MPM工作模式

- prefork：多进程I/O模型，一个主进程，管理多个子进程，一个子进程处理一个请求。
- worker：复用的多进程I/O模型，多进程多线程，一个主进程，管理多个子进程，一个子进程管理多个线程，每个 线程处理一个请求。
- event：事件驱动模型，一个主进程，管理多个子进程，一个进程处理多个请求。

prefork模式

优点：适合于没有线程安全库，需要避免线程兼容性问题的系统。它是要求将每个请求相互独立的情况下最好的mpm，这样若一个请求出现问题就不会影响到其他请求。

缺点：一个进程相对占用更多的系统资源，消耗更多的内存。而且，它并不擅长处理高并发请求，在这种场景下，它会将请求放进队列中，一直等到有可用进程，请求才会被处理。



- 查看默认选择处理模块为prefork

```
[root@server ~]# httpd -V
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::eaf3:dc40:2bf:6da2. Set the 'ServerName' directive globally to suppress this message
Server version: Apache/2.4.6 (CentOS)
Server built:   Nov 16 2020 16:18:20
Server's Module Magic Number: 20120211:24
Server loaded:  APR 1.4.8, APR-UTIL 1.5.2
Compiled using: APR 1.4.8, APR-UTIL 1.5.2
Architecture:   64-bit
Server MPM:     prefork
  threaded:     no
    forked:     yes (variable process count)
```

- 切换apache的mpm工作模式

```
[root@server ~]# cat /etc/httpd/conf.modules.d/00-mpm.conf | grep -Ev "^#|^$"
LoadModule mpm_prefork_module modules/mod_mpm_prefork.so

[root@server ~]# ps aux | grep httpd
root 12886 0.4 0.2 221928 4956 ? Ss 11:03 0:00 /usr/sbin/httpd -DFOREGROUND
apache 12887 0.0 0.1 221928 2992 ? S 11:03 0:00 /usr/sbin/httpd -DFOREGROUND
```

```
apache 12888 0.0 0.1 221928 2992 ? S 11:03 0:00 /usr/sbin/httpd -
DFOREGROUND
apache 12889 0.0 0.1 221928 2992 ? S 11:03 0:00 /usr/sbin/httpd -
DFOREGROUND
apache 12890 0.0 0.1 221928 2992 ? S 11:03 0:00 /usr/sbin/httpd -
DFOREGROUND
apache 12891 0.0 0.1 221928 2992 ? S 11:03 0:00 /usr/sbin/httpd -
DFOREGROUND
若要使用worker和event工作模型，只需要在/etc/httpd/conf.modules.d/00-mpm.conf中取消对应注释即可
```

- 修改prefork参数

```
默认参数：
StartServers 5 # 服务启动时的进程数
MaxSpareServers 10 # 最大空闲服务进程数
MinSpareServers 5 # 最小空闲进程数
MaxRequestWorkers 256 # 单个进程最多接受的进程数
[root@server ~]# vim /etc/httpd/conf.d/mpm.conf
StartServers 10
MaxSpareServers 15
MinSpareServers 10
MaxRequestWorkers 256
MaxRequestsPerChild 4000
[root@localhost ~]# systemctl restart httpd
[root@server ~]# ps -ef | grep httpd
```

- 压测工具

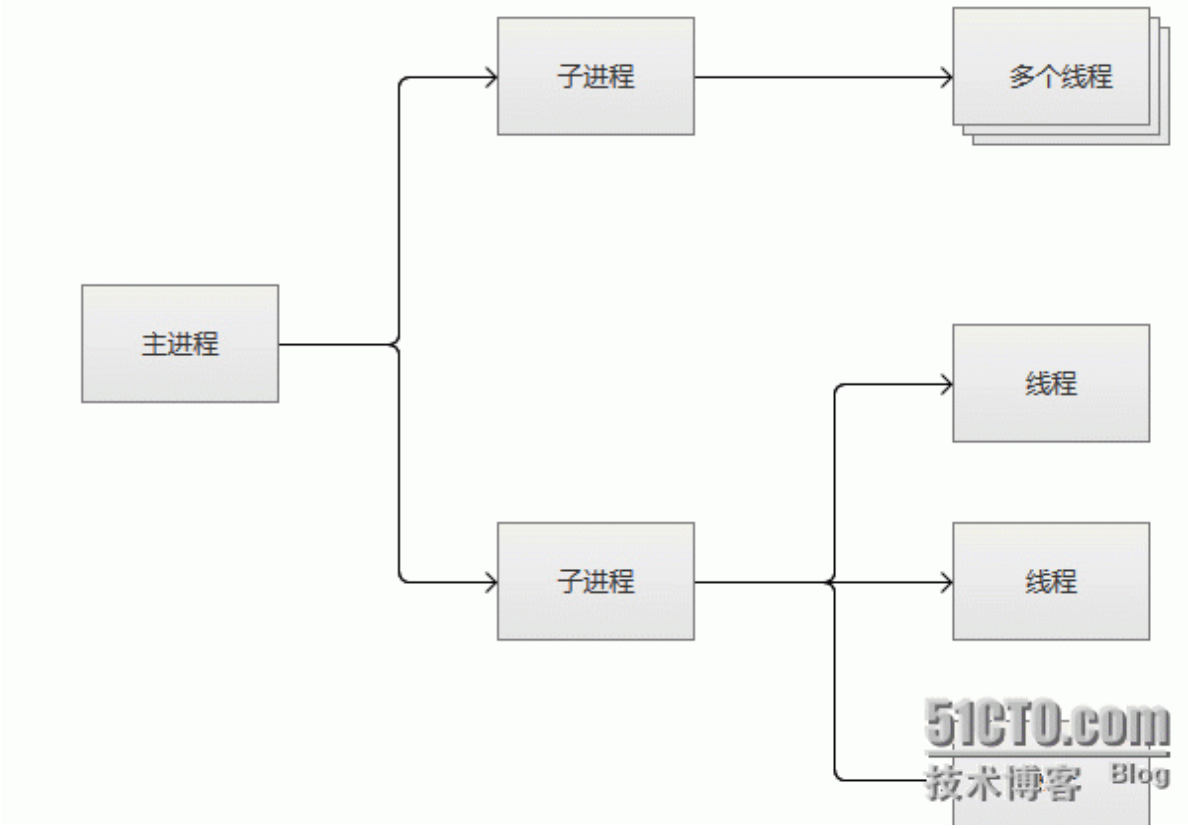
```
[root@localhost ~]# ab -n 1000000 -c 1000  http://127.0.0.1/
# -n 即requests，用于指定压力测试总共的执行次数
# -c 即concurrency，用于指定的并发数
```

work模式

work使用了多进程和多线程的混合模式，worker模式也同样会先派生一下子进程，然后每个子进程创建一些线程，同时包括一个监听线程，每个请求过来会被分配到一个线程来服务。

优点：线程比进程会更轻量，因为线程是通过共享父进程的内存空间，因此，内存的占用会减少一些，在高并发高流量的场景下会比prefork有更多可用的线程，表现会更优秀一些。

缺点：如果一个线程出现了问题也会导致同一进程下的线程出现问题，如果是多个线程出现问题，也只是影响apache的一部分，而不是全部。由于用到多进程多线程，需要考虑到线程的安全。



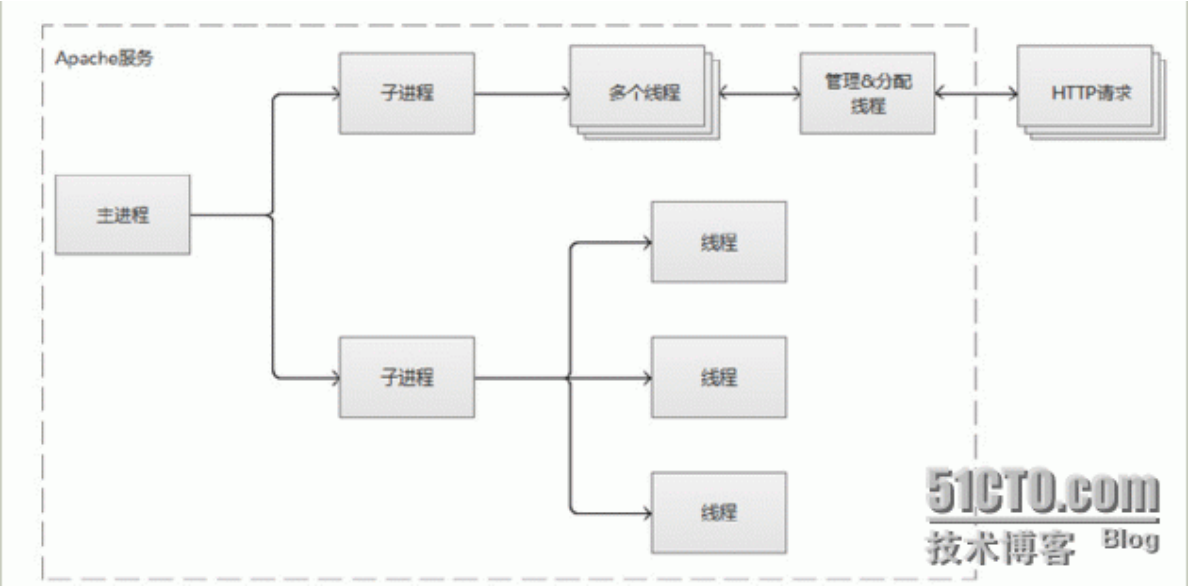
- 相关参数

```
StartServers
#服务器启动时建立的子进程数量,在workers模式下默认是3.

ServerLimit
#系统配置的最大进程数量

MinSpareThreads
#空闲子进程的最小数量，默认75

MaxSpareThreads
#空闲子进程的最大数量，默认250

ThreadsPerChild
#每个子进程产生的线程数量，默认是64

MaxRequestWorkers /MaxClients
#限定服务器同一时间内客户端最大接入的请求数量.

MaxConnectionsPerChild
#每个子进程在其生命周期内允许最大的请求数量，如果请求总数已经达到这个数值，子进程将会结束，如果设置为0，子进程将永远不会结束。在Apache2.3.9之前称之为MaxRequestsPerChild。
```

event模式

这个是apache中最新的模式，在现在的版本里已经是稳定可用的模式，它和worker模式很像，最大的区别在于，它解决了keep-alive场景下，长期被占用的线程的资源浪费问题（某些线程因为被keep-alive，挂载哪里等待，中间基于没有请求过来，一直等到超时）

event中会有一个专门的线程来管理这些keep-alive类型的线程，当有真实请求u过来的时候，将请求传递给服务线程，执行完毕后，又允许它释放。这样，一个线程就能处理几个请求了，实现了异步非阻塞。



参数可以参考work模式中的参数

访问控制机制

基于IP地址访问控制

更改站点根目录案例

- 重新定义根目录

```
# 定义服务器的文档的页面路径：
[root@server1 conf]# vim httpd.conf
......
DocumentRoot "/data/www/html"
......
# 准备页面
[root@server1 ~]# echo "this path /data/wwww/html" >
/data/www/html/index.html
# 重启服务
[root@server1 ~]# systemctl restart httpd
```

- 测试访问，发现状态码为403没有权限

```
[root@server1 conf]# curl 192.168.80.151 -I
HTTP/1.1 403 Forbidden
Date: Mon, 22 Feb 2021 06:19:54 GMT
Server: Apache/2.4.6 (CentOS)
Last-Modified: Thu, 16 Oct 2014 13:20:58 GMT
ETag: "1321-5058a1e728280"
Accept-Ranges: bytes
Content-Length: 4897
Content-Type: text/html; charset=UTF-8
```

- 访问控制机制中开放相应目录权限

```
[root@server1 conf]# vim httpd.conf
<Directory "/data/www/html">
Require all granted
</Directory>
[root@server1 ~]# systemctl restart httpd
```

- 再次测试访问发现可以成功访问

```
[root@server1 conf]# curl 192.168.80.151 -I
HTTP/1.1 200 OK
Date: Mon, 22 Feb 2021 06:21:15 GMT
Server: Apache/2.4.6 (CentOS)
Last-Modified: Mon, 22 Feb 2021 06:18:57 GMT
ETag: "1a-5bbe6c6eb43fa"
Accept-Ranges: bytes
Content-Length: 26
Content-Type: text/html; charset=UTF-8

[root@server1 conf]# curl 192.168.80.151
this path /data/wwww/html
```

- 更加详细的访问控制配置的参数

```
Require常见配置参数：
Require all granted             # 全部放行
Require all denied              # 全部拒绝
Require ip IPAd                 # 放行某ip地址
Require not ip IP               # 拒绝某ip地址
Require user user1              # 放行某用户
Require group group1            # 放行某组
PS：34参数需要在…中才可以。
<RequireAll>
    Require all granted
    Require not ip 10.252.46.165
</RequireAll>
```

- 黑名单方式

```
<RequireAll>
Require all granted
Require not ip 172.16.1.1 #拒绝特定IP
</RequireAll>
```

- 白名单方式

```
<RequireAny>
Require all denied
require ip 172.16.1.1 #允许特定IP
</RequireAny>
```

- 只允许特定网段访问

```
<requireany>
require all denied
Require ip 192.168.39.0/24
</requireany>
```

- 只允许特定主机访问

```
<Requireany>
Require all denied
Require ip 192.168.32.7 #只允许特定的主机访问
</Requireany>
```

用户访问控制

认证方式有basic和digest两种

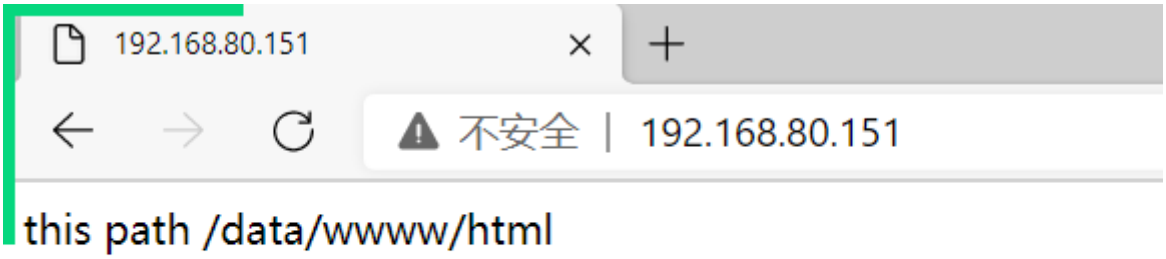
- 创建用户认证文件，为用户认证做准备

```
[root@server1 ~]# htpasswd -c -m /etc/httpd/conf.d/.htpassword lisi
New password:
Re-type new password:
Adding password for user lisi
[root@server1 ~]# htpasswd -b -m /etc/httpd/conf.d/.htpassword zhangsan
zhangsan
Adding password for user zhangsan
```

- 修改配置文件，启用用户认证

```
[root@server1 ~]# vim /etc/httpd/conf/httpd.conf
<Directory "/data/www/html">
AuthType Basic
AuthName "Restricted Resource"
AuthBasicProvider file
AuthUserFile /etc/httpd/conf.d/.htpassword
Require user lisi
</Directory>
[root@server1 ~]# systemctl restart httpd.service
```

- 测试访问，发现lisi可以成功访问，zhangsan不能访问



- 扩展，认证组文件

```
通过认证组文件：
<Directory "/data/www/html">
AuthType Basic
AuthName "Restricted Resource"
AuthBasicProvider file
AuthUserFile /etc/httpd/conf.d/.htpassword
AuthGroupFile /etc/httpd/conf.d/.htgroup
Require group group1
</Directory>
使用浏览器访问测试即可！
```

日志设定

```
ErrorLog "logs/error_log"
LogLevel warn
<IfModule log_config_module>
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
<IfModule logio_module>
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I
%O"
combinedio
</IfModule>
CustomLog "logs/access_log" combined
</IfModule>
```

- 日志参数

```
参数:
%h Remote hostname
%l Remote logname
%u Remote user
%t Time the request was received, in the format [18/Sep/2011:19:18:28 -0400]
%r First line of request
%s Status
%b Size of response in bytes, excluding HTTP headers
Referer 有利于分析用户是通过哪个网站转发的如通过baidu转发的，也可以监控网站盗链的发生。
User-Agent 记录浏览器的类型。防止爬虫一定程度上，爬虫可以伪造浏览器类型。curl -A "evan"
http://I(伪造名字叫evan的浏览器)
```

虚拟主机

基于IP地址虚拟主机

```
[root@node3 data]# cat /etc/httpd/conf.d/site.conf
<Directory "/data/">
Require all granted
</Directory>
<VirtualHost 192.168.0.140:80>
Servername www.site1.com
DocumentRoot "/data/site1/"
</VirtualHost>
<VirtualHost 192.168.0.145:80>
Servername www.site2.com
DocumentRoot "/data/site2/"
</VirtualHost>

[root@node1 ~]# curl 192.168.0.142
<h1>This is site1</h1>
[root@node1 ~]# curl 192.168.0.145
<h1>This is site2</h1>
```

基于端口虚拟主机

```
[root@server1 ~]# cat /etc/httpd/conf.d/site.conf
Listen 8080
Listen 9090

<Directory "/data/">
Require all granted
</Directory>

<VirtualHost *:8080>
DocumentRoot "/data/site3/"
</VirtualHost>

<VirtualHost *:9090>
DocumentRoot "/data/site4/"
</VirtualHost>

[root@server1 ~]# curl 192.168.80.100:8080
<h1>This is site3</h1>
[root@server1 ~]# curl 192.168.80.100:9090
<h1>This is site4</h1>
```

基于FQDN虚拟主机

```
[root@server1 ~]# cat /etc/httpd/conf.d/site.conf
Listen 10101

<Directory "/data/">
Require all granted
</Directory>

<VirtualHost 192.168.80.100:10101>
Servername www.site5.com
DocumentRoot "/data/site5/"
</VirtualHost>

<VirtualHost 192.168.80.100:10101>
Servername www.site6.com
DocumentRoot "/data/site6/"
</VirtualHost>
~

[root@server1 ~]# cat /etc/hosts
192.168.0.142 www.site5.com
192.168.0.142 www.site6.com
[root@server1 ~]# curl www.site5.com:10101
<h1>This is site5</h1>
[root@server1 ~]# curl www.site6.com:10101
<h1>This is site6</h1>

```

- 三种不同方式的虚拟主机验证汇总

```
[root@server1 ~]# curl 192.168.80.100
<h1>This is site1</h1>
[root@server1 ~]# curl 192.168.80.200
<h1>This is site2</h1>
[root@server1 ~]# curl 192.168.80.100:8080
<h1>This is site3</h1>
[root@server1 ~]# curl 192.168.80.100:9090
<h1>This is site4</h1>
[root@server1 ~]# curl www.site5.com:10101
<h1>This is site5</h1>
[root@server1 ~]# curl www.site6.com:10101
<h1>This is site6</h1>
```

SSL配置

- 安装mod_ssl和openssl

```
[root@node1 ~]# yum install mod_ssl openssl -y
```

- 生成2048位的加密私钥server.key

```
[root@node1 ~]# openssl genrsa -out server.key 2048
```

- 生成证书签名请求server.csr

```
[root@node1 ~]# openssl req -new -key server.key -out server.csr
```

- 生成类型为X509的自签名证书。有效期设置3650天，即有效期为10年server.crt

```
[root@node1 ~]# openssl x509 -req -days 3650 -in server.csr -signkey
server.key -out
server.crt
```

- 复制文件到相应位置

```
[root@node1 ~]# cp server.crt /etc/pki/tls/certs/
[root@node1 ~]# cp server.key /etc/pki/tls/private/
[root@node1 ~]# cp server.csr /etc/pki/tls/private/
```

- 修改配置文件

```
Servername 192.168.0.140:443
SSLCertificateFile /etc/pki/tls/certs/server.crt
SSLCertificateKeyFile /etc/pki/tls/private/server.key
```

- 防火墙放行

```
[root@node1 ~]# firewall-cmd --add-port=443/tcp --per
最后访问测试即可！
```

LAMP架构

- LAMP动态网站部署架构是由一套 Linux+Apache+MySQL+PHP 组成的动态网站系统解决方案。
- LNMP动态网站部署架构是由一套 Linux+Nginx+MySQL+PHP 组成的动态网站系统解决方案。

```
1.安装相关软件包
[root@node1 ~]# yum install httpd php php-mysql mariadb-server -y
[root@node1 ~]# cat /etc/httpd/conf.d/php.conf | grep -Ev "^#|^$"
<FilesMatch \.php$>
SetHandler application/x-httpd-php # 定义了以.php结尾的文件触发x-httpd-php
AddType text/html .php
DirectoryIndex index.php # 定义默认主页面
php_value session.save_handler "files" # 定义保持handler为文件
php_value session.save_path "/var/lib/php/session" # 定义会话保持路径
<FilesMatch>
2.启动httpd服务
[root@node1 ~]# systemctl start httpd
[root@node1 ~]# firewall-cmd --add-port=80/tcp --per
3.编写测试页面
[root@node1 ~]# cat /var/www/html/index.php
<html>
<head>
<title>PHP 测试</title>
</head>
<body>
<?php echo '<p>Hello World</p>'; ?>
</body>
</html>
打开浏览器访问：http://192.168.0.140/index.php
4.启动数据库，并编写测试页面
[root@node1 ~]# cat /var/www/html/test.php
<?php
$link=mysql_connect("127.0.0.1","root","");
if(!$link)
echo "FAILD!连接错误，用户名密码不对";
else
echo "OK!可以连接";
?>
打开浏览器访问：http://192.168.0.140/test.php
```