nginx

web服务：Apache、nginx、Tomcat、IIS

安装

```
yum -y install gcc gcc-c++ autoconf automake make
yum -y install zlib zlib-devel openssl openssl-devel pcre pcre-devel
# 选最新的文档就行
wget http://nginx.org/download/nginx-1.16.1.tar.gz
tar -xf ...
# --prefix  指定安装目录
# --with-http_stub_status_module  状态统计模块 ，常用于监控nginx的状态  zabbix
# --with-http_ssl_module  用于https的支持。
./configure --prefix=/usr/local/nginx --with-http_stub_status_module --with-http_ssl_module --with-stream --with-file-aio --with-threads --with-http_realip_module
make
make install
```

- 模块：
  - --without： 默认开启的，需要手动关闭
  - --with： 默认是关闭的，需要手动开启
- 脚本：
  - ```
    mkdir -p /nginx2
    cd /nginx2
    #download nginx
    curl -O http://nginx.org/download/nginx-1.17.9.tar.gz

    #解压
    tar xf nginx-1.17.9.tar.gz

    #进入解压目录
    cd nginx-1.17.9

    #解决依赖关系
    yum -y install zlib zlib-devel openssl openssl-devel pcre pcre-devel gcc gcc-c++ autoconf automake make

    #配置
    ./configure  --prefix=/usr/local/nginx9  --user=tzk --group=tzk --with-threads  --with-file-aio  --with-http_ssl_module  --with-http_stub_status_module  --with-stream

    #编译和安装
    make -j 2 ; make install

    #开机启动
    echo "/usr/local/nginx9/sbin/nginx" >>/etc/rc.local
    chmod +x /etc/rc.d/rc.local

    #修改PATH变量
    echo "PATH=/usr/local/nginx9/sbin:$PATH" >>/etc/profile
    PATH=/usr/local/nginx9/sbin:$PATH
    #启动

    nginx
    ```
  - ```
    [root@sc-nginx nginx]# vim /usr/lib/systemd/system/nginx.service
    [Unit]
    Description=The nginx HTTP and reverse proxy server
    ```

```
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/usr/local/nginx9/logs/nginx.pid
# Nginx will fail to start if /run/nginx.pid already exists but has the wrong
# SELinux context. This might happen when running `nginx -t` from the cmdline.
# https://bugzilla.redhat.com/show_bug.cgi?id=1268621
ExecStartPre=/usr/bin/rm -f /usr/local/nginx9/logs/nginx.pid
ExecStartPre=/usr/local/nginx9/sbin/nginx -t
ExecStart=/usr/local/nginx9/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillSignal=SIGQUIT
TimeoutStopSec=5
KillMode=mixed
PrivateTmp=true

[Install]
WantedBy=multi-user.target

[root@sc-nginx nginx]# systemctl daemon-reload
```

一些常见命令

```
# 服务不间断运行---修改了配置文件
nginx -s reload
# 语法检测
nginx -t
nginx -s 信号

其中信号可能是以下之一：
stop —快速关机
quit —正常关机
reload —重新加载配置文件 -->不会停止业务，重新加载新的配置文件
reopen —重新打开日志文件
```

Nginx 服务器，正常运行过程中：

- **多进程：**一个 Master 进程、多个 Worker 进程。
- **Master 进程：**管理 Worker 进程。对外接口：接收外部的操作（信号）；对内转发：根据外部的操作的不同，通过信号管理 Worker；**监控：**监控 Worker 进程的运行状态，Worker 进程异常终止后，自动重启 Worker 进程。
- **Worker 进程：**所有 Worker 进程都是平等的。实际处理：网络请求，由 Worker 进程处理。Worker 进程数量：在 nginx.conf 中配置，一般设置为核心数，充分利用 CPU 资源，同时，避免进程数量过多，避免进程竞争 CPU 资源，增加上下文切换的损耗。

conf文件

详解

```
配置文件
主配置文件：/etc/nginx/nginx.conf (yum安装为例)
主要三个模块： 全局块、events块、http块

log_format:
log_format格式变量：
    $remote_addr  #记录访问网站的客户端地址
    $remote_user  #远程客户端用户名
    $time_local  #记录访问时间与时区
    $request  #用户的http请求起始行信息
    $status  #http状态码，记录请求返回的状态码，例如：200、301、404等
    $body_bytes_sent  #服务器发送给客户端的响应body字节数
    $http_referer  #记录此次请求是从哪个连接访问过来的，可以根据该参数进行防盗链设置。
    $http_user_agent  #记录客户端访问信息，例如：浏览器、手机客户端等
    $http_x_forwarded_for  #当前端有代理服务器时，设置web节点记录客户端地址的配置，此参数生效的前提是代理服务器也要进行相关的x_forwarded_for设置

location 优先级匹配：
~  波浪线表示执行一个正则匹配，区分大小写
~*  表示执行一个正则匹配，不区分大小写
^~  表示普通字符匹配，不是正则匹配。如果该选项匹配，只匹配该选项，不匹配别的选项，一般用来匹配目录
=  进行普通字符精确匹配
@   定义一个命名的 location，使用在内部定向时，例如 error_page, try_files

优先级：
1.等号类型（=）的优先级最高。一旦匹配成功，则不再查找其他匹配项。
2.^~类型表达式。一旦匹配成功，则不再查找其他匹配项。
3.正则表达式类型（~ ~*）的优先级次之。如果有多个location的正则能匹配的话，则优先匹配同一个server中，配置在前面的location
  4.常规字符串匹配类型。按前缀匹配

负载均衡：upstream
分配算法： 轮询（默认）、weight 加权、ip_hash、 url_hash、fair
backup  down

X-Forwarded-For头信息可以有多个，中间用逗号分隔，第一项为真实的客户端ip，剩下的就是曾经经过的代理或负载均衡的ip地址，经过几个就会出现几个
proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
$proxy_add_x_forwarded_for变量包含客户端请求头中的"X-Forwarded-For"，与$remote_addr用逗号分开，如果没有"X-Forwarded-For" 请求头，则$proxy_add_x_forwarded_for等于$remote_addr。$remote_addr变量的值是客户端的IP。

X-Forwarded-For头域是为了说明请求经过了哪些服务器。
如果请求中不包含X-Forwarded-For头域，则设置X-Forwarded-For头域值为请求发送者的IP；
如果请求中包含X-Forwarded-For头域，则设置X-Forwarded-For头域值为之前该头域的值后面添加请求发送者的IP，用逗号分隔。

 client-->proxy1-->proxy2--->proxy3--server
 那server拿到的x-forward-for 是这样的：X-Forwarded-For: client, proxy1, proxy2
在proxy1的时候  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for
这个时候会找x-forward-for  没有的话，就会把remote_addr赋給它，到达proxy2的时候又找x-forward-for  有的话就会把remote_addr添加进去。
```

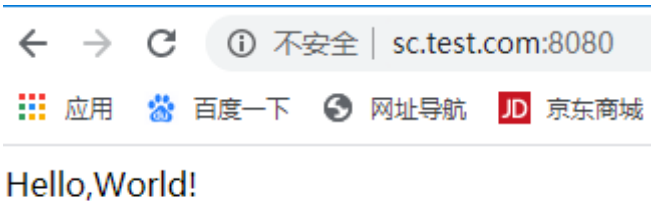
修改端口修改首页

```
server {
        listen       8080 default_server;
        listen       [::]:8080 default_server;
        server_name  sc.test.com;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
                root /usr/share/nginx/html;
                index test.html;
        }

```



HTTP 连接建立和请求处理过程如下：

- Nginx 启动时，Master 进程，加载配置文件。
- Master 进程，初始化监听的 Socket。
- Master 进程，Fork 出多个 Worker 进程。
- Worker 进程，竞争新的连接，获胜方通过三次握手，建立 Socket 连接，并处理请求。

http功能配置

rewrite配置

1、对URL进行跳转

- 修改nginx配置：(输入www.tanzikun.com --> www.sanchuang.cn)

```
server {
        listen       80;
        server_name  www.tanzikun.com;

        #charset koi8-r;

        access_log  logs/tzk.access.log  main;

        location / {
            root   html/tanzikun;
            rewrite ^/(.*?) http://www.sanchuangedu.cn/ permanent;
            #index  tanzikun.html index.html index.htm;
        }
```

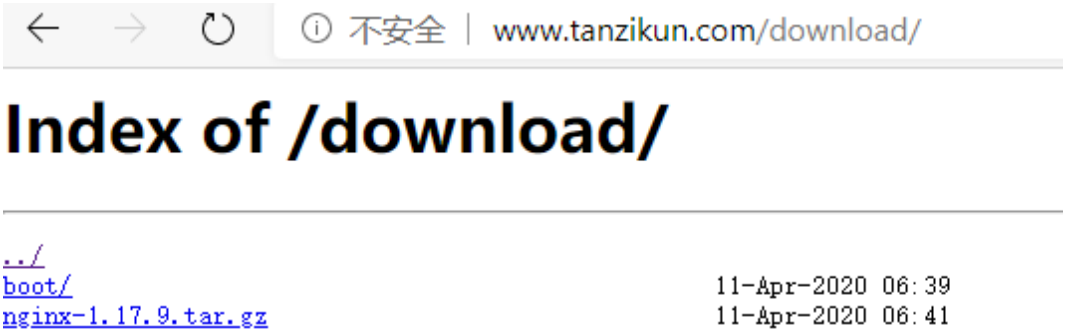
- rewrite的限制功能

```
#对浏览器进行限制
            location / {
            root   html/tanzikun;
            # 如果使用的不是Chrome内核的浏览器，那就跳转到www.baidu.com
            if ($http_user_agent !~* Chrome) {
                rewrite ^/lihu http://www.baidu.com redirect;
            }
            index  index.html index.htm;
        }
#对ip地址进行限制
            location / {
            root   html/tanzikun;
            # 如果是192.168.0.1[0-9]，那就跳转到www.qq.com
            if ($remote_addr ~* 192.168.0.1[0-9]) {
              rewrite ^/tanzk http://www.qq.com redirect;
             }
            index  index.html index.htm;
        }
```

2、做网站提供下载功能

- 在location中开启autoindex：

```
location /download {
            root   html/tanzikun;
            autoindex on;
        }
```



3、限速功能

- 在http中和location中都要修改：

```
http {
  limit_conn_zone $binary_remote_addr zone=sanchuang:10m;  #定义
#说明：首先用limit_conn_zone定义了一个内存区块索引sanchuang，大小为10m，它以$binary_remote_addr作为key 根据ip地址来进行连接限制。该配置只能在http里面配置，不支持在server里配置。
   server {
        listen       80;
        server_name  www.tanzikun.com;

        charset koi8-r;
        limit_conn sanchuang 10;
        #limit_conn 定义针对sanchuang这个zone，并发连接为10个。在这需要注意一下，这个10指的是单个IP的并发最多为10个。
#下载目录的配置
        location /download {
            root   html/tanzikun;
            autoindex on;
        limit_rate_after 100k;   #当一个文件下载到指定大小（本例中为100k）之后开始限速。
        limit_rate 10k; #定义下载速度为10k/s。
        }
}
}
```

- ***效果显示如下：***

```
[root@nginx-bianyi ~]# wget http://www.tanzikun.com/download/test.dd
--2020-04-11 15:12:27--  http://www.tanzikun.com/download/test.dd
正在解析主机 www.tanzikun.com (www.tanzikun.com)... 192.168.200.152
正在连接 www.tanzikun.com (www.tanzikun.com)|192.168.200.152|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：104857600 (100M) [application/octet-stream]
正在保存至：“test.dd”

test.dd                     0%[                                    ] 348.00K  9.98KB/s  剩余 1h 56m
```

图片防盗链功能

- 防止别的网站盗链自己的网站，使得自己网站的带宽等资源被浪费。

```
location ~* \.(gif|jpg|png|swf|flv)$ {
        valid_referers none blocked www.; #定义有效的引用
        if ($invalid_referer) {
                return 404;
          }
        }
```

统计功能:

- ```
  location /basic_status {
                  stub_status;
          }
  # 参数:
  Active connections:活跃的连接数
  accepts：接收请求数
  handled：处理的请求数
  ```

认证功能:

- ```
  location /basic_status {
                  auth_basic "tzk";
                  auth_basic_user_file conf/htpasswd;
                  stub_status;
          }

  # htpasswd文件在conf目录下
  # 生成用户名密码命令：htpasswd -c /usr/local/nginx/conf/htpasswd tzk
  # deny 192.168.0.11    禁止这个ip    deny all   禁止所有ip（一般放在最后面）
  # allow 192.168.0.11    允许这个ip
  ```

HTTPS:

https://zhuanlan.zhihu.com/p/27395037

- 加密算法:
  - 对称
  - 非对称： ecdsa
  - hash： MD5
- 在nginx服务器上使用证书：***一个证书是和一个域名绑定的***
  - ```
    #    server {
    #        listen       443 ssl http2 default_server;
    #        listen       [::]:443 ssl http2 default_server;
    #        server_name  _;
    #        root         /usr/share/nginx/html;
    #
    #        ssl_certificate "/etc/pki/nginx/server.crt"; 证书的公钥
    #        ssl_certificate_key "/etc/pki/nginx/private/server.key";
    证书的私钥
    #        ssl_session_cache shared:SSL:1m;
    #        ssl_session_timeout  10m;
    #        ssl_ciphers HIGH:!aNULL:!MD5;
    #        ssl_prefer_server_ciphers on;
    #
    #        # Load configuration files for the default server block.
    #        include /etc/nginx/default.d/*.conf;
    ```
    ```

负载均衡器：

- 1、定义在http语句块中：

```
# 定义一个负载均衡器名字叫daishuyun调度器
upstream daishuyun {
    server 192.168.0.10;
    server 192.168.0.11;
}
```

- 2、将访问请求转发到负载均衡器：

```
server {
    listen 80;
    server_name www.daishuyun.com;
    location / {
        proxy_pass http://daishuyun;
        #health_check
    }
}
```

--- **后台的real-server不知道客户机的IP地址，LB在做数据转发时，将ip包中的源ip和目的ip都做了改变**，但是如何让后台real-server知道客户机的ip：在real-server的日志格式中添加一个参数传入源ip的值。***我这里加的参数是：$HTTP_X_REAL_IP***

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                     '"$http_user_agent" "$http_x_forwarded_for"';
```

然后在LB的http语句块中加入这个变量http_x_forwarded_for：

```
location / {
    proxy_pass http://daishuyun;
    proxy_set_header X-Real-IP $remote_addr;
    # 将$remote_addr的值传给HTTP_X_REAL_IP这个参数
}
```

real-server中添加：$HTTP_X_REAL_IP

```
log_format  main  '$HTTP_X_REAL_IP - $remote_addr - $remote_user
[$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                     '"$http_user_agent" "$http_x_forwarded_for"';
```

对上游real-server的配置

fail_timeout	**设置必须多次尝试失败才能将服务器标记为不可用的时间，以及将服务器标记为不可用的时间（默认为10秒）**
max_fails	**设置在fail_timeout服务器标记为不可用的时间内必须发生的失败尝试次数（默认为1次尝试）**
backup	**当其他的服务器都不提供服务的时候，在启用这台服务器提供服务--》**备胎
slow_start	**慢启动**
down	**将上游的服务器标识为不可用，不会再发送任何的请求给这台服务器**

eg:

```
upstream backend {
    server backend1.example.com slow_start=30s;  # 晚30秒给上游的服务器转发请求
    server backend2.example.com;
    server 192.0.0.1 backup;
}
```

配置4层负载均衡器：

https://docs.nginx.com/nginx/admin-guide/load-balancer/tcp-udp-load-balancer/

eg:

```
worker_processes  2;
error_log  logs/error.log  info;
pid        logs/nginx.pid;
events {
    worker_connections  1024;
}
stream {
    upstream daishuyun {
        server 192.168.0.19:80;
        server 192.168.0.20:80;
             }
    upstream dns_servers {
        server 192.168.0.19:53;
        server 192.168.0.20:53;
             }
    server {
            listen     8080;
            proxy_pass sanchuang;
          }
    server {
        listen     53 udp;
               proxy_pass daishuyun;
                    }
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request"
'
```

```
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  logs/access.log  main;
    sendfile        on;
    tcp_nopush     on;
    keepalive_timeout  65;
    gzip  on;
}

#注意下http语句块里没有server语句块了，server语句块全部放到stream语句块里了 -->需要验证
```

nginx 高性能 高并发

Nginx 为什么拥有高性能并且能够支撑高并发?

- Nginx 采用多进程+异步非阻塞方式（IO 多路复用 Epoll）。
- 请求的完整过程：建立连接→读取请求→解析请求→处理请求→响应请求。
- 请求的完整过程对应到底层就是：读写 Socket 事件。

nginx-负载均衡

4种负载均衡算法

- 轮询
  - weight权重--->默认值是1
- ip_hash
  - 这个方法确保了相同的客户端的请求一直发送到相同的服务器，以保证session会话。这样每个访客都固定访问一个后端服务器，可以解决session不能跨服务器的问题。
  - ```
    在nginx版本1.3.1之前，不能在ip_hash中使用权重（weight）。
    ip_hash不能与backup同时使用。
    此策略适合有状态服务，比如session。
    当有服务器需要剔除，必须手动down掉。

    #动态服务器组
        upstream dynamic_zuoyu {
            ip_hash;    #保证每个访客固定访问一个后端服务器
            server localhost:8080   weight=2;  #tomcat 7.0
            server localhost:8081;  #tomcat 8.0
            server localhost:8082;  #tomcat 8.5
            server localhost:8083   max_fails=3 fail_timeout=20s;
    #tomcat 9.0
        }
    ```

- least_conn
  - ```
    把请求转发给连接数较少的后端服务器。轮询算法是把请求平均的转发给各个后端，使它们的负载大致相同；但是，有些请求占用的时间很长，会导致其所在的后端负载较高。这种情况下，least_conn这种方式就可以达到更好的负载均衡效果。

    #动态服务器组
        upstream dynamic_zuoyu {
            least_conn;    #把请求转发给连接数较少的后端服务器
            server localhost:8080   weight=2;  #tomcat 7.0
            server localhost:8081;  #tomcat 8.0
            server localhost:8082 backup;  #tomcat 8.5
            server localhost:8083   max_fails=3 fail_timeout=20s;
    #tomcat 9.0
        }
    ```

- url_hash

- ```
  按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，要配合缓存命中率来使用。同一个资源多次请求，可能会到达不同的服务器上，导致不必要的多次下载，缓存命中率不高，以及一些资源时间的浪费。而使用url_hash，可以使得同一个url（也就是同一个资源请求）会到达同一台服务器，一旦缓存住了资源，再此收到请求，就可以从缓存中读取。

  #动态服务器组
      upstream dynamic_zuoyu {
          hash $request_uri;    #实现每个url定向到同一个后端服务器
          server localhost:8080;  #tomcat 7.0
          server localhost:8081;  #tomcat 8.0
          server localhost:8082;  #tomcat 8.5
          server localhost:8083;  #tomcat 9.0
      }
  ```

- fair
  - ```
    按照服务器端的响应时间来分配请求，响应时间短的优先分配。

    #动态服务器组
        upstream dynamic_zuoyu {
            server localhost:8080;  #tomcat 7.0
            server localhost:8081;  #tomcat 8.0
            server localhost:8082;  #tomcat 8.5
            server localhost:8083;  #tomcat 9.0
            fair;    #实现响应时间短的优先分配
        }
    ```

  - **能者多劳**

nginx vs Apache

```
Nginx：
IO 多路复用，Epoll（freebsd 上是 kqueue）
高性能
高并发
占用系统资源少

Apache：
阻塞+多进程/多线程
更稳定，Bug 少
模块更丰富
```

Nginx 最大连接数

基础背景：

- Nginx 是多进程模型，Worker 进程用于处理请求。
- 单个进程的连接数（文件描述符 fd），有上限（nofile）：ulimit -n。
- Nginx 上配置单个 Worker 进程的最大连接数：worker_connections 上限为 nofile。
- Nginx 上配置 Worker 进程的数量：worker_processes。

因此，Nginx 的最大连接数：

- Nginx 的最大连接数：Worker 进程数量 x 单个 Worker 进程的最大连接数。
- 上面是 Nginx 作为通用服务器时，最大的连接数。
- Nginx 作为反向代理服务器时，能够服务的最大连接数：（Worker 进程数量 x 单个 Worker 进程的最大连接数）/ 2。
- Nginx 反向代理时，会建立 Client 的连接和后端 Web Server 的连接，占用 2 个连接。

思考：

- 每打开一个 Socket 占用一个 fd?
- 为什么，一个进程能够打开的 fd 数量有限制?

HTTP 请求和响应

HTTP 请求：

- **请求行：method、uri、http version**
- **请求头**
- **请求体**

HTTP 响应：

- **响应行：http version、status code**
- **响应头**
- **响应体**

IO 模型

处理多个请求时，可以采用：IO 多路复用或者阻塞 IO+多线程：

- **IO 多路复用**：一个线程，跟踪多个 Socket 状态，哪个就绪，就读写哪个。
- **阻塞 IO+多线程**：每一个请求，新建一个服务线程。

IO 多路复用和多线程的适用场景?

- IO 多路复用：单个连接的请求处理速度没有优势。
- 大并发量：只使用一个线程，处理大量的并发请求，降低上下文环境切换损耗，也不需要考虑并发问题，相对可以处理更多的请求。
- 消耗更少的系统资源（不需要线程调度开销）。

- 适用于长连接的情况（多线程模式长连接容易造成线程过多，造成频繁调度）。
- 阻塞 IO +多线程：实现简单，可以不依赖系统调用。
- 每个线程，都需要时间和空间。
- 线程数量增长时，线程调度开销指数增长。

select/poll 和 epoll 比较如下（3个函数属于内核）：

详细内容，参考：

> https://www.cnblogs.com/wiessharling/p/4106295.html

select/poll 系统调用：

select：

- 查询 fd_set 中，是否有就绪的 fd，可以设定一个超时时间，当有 fd (File descripter) 就绪或超时返回。
- fd_set 是一个位集合，大小是在编译内核时的常量，默认大小为 1024。
- 特点：连接数限制，fd_set 可表示的 fd 数量太小了；线性扫描：判断 fd 是否就绪，需要遍历一边 fd_set；数据复制：用户空间和内核空间，复制连接就绪状态信息。

poll：

- **解决了连接数限制：** poll 中将 select 中的 fd_set 替换成了一个 pollfd 数组，解决 fd 数量过小的问题。
- **数据复制：** 用户空间和内核空间，复制连接就绪状态信息。

epoll，event 事件驱动：

- **事件机制：** 避免线性扫描，为每个 fd，注册一个监听事件，fd 变更为就绪时，将 fd 添加到就绪链表。
- **fd 数量：** 无限制（OS 级别的限制，单个进程能打开多少个 fd）。

select，poll，epoll：

- I/O 多路复用的机制。
- I/O 多路复用就通过一种机制，可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作；监视多个文件描述符。
- 但 select，poll，epoll 本质上都是同步 I/O：用户进程负责读写（从内核空间拷贝到用户空间），读写过程中，用户进程是阻塞的；异步 IO，无需用户进程负责读写，异步 IO，会负责从内核空间拷贝到用户空间。