题目：基于 **nginx** 搭建 **C/S** 架构系统

背景介绍

现如今，各种 C/S 架构的系统都已经在线上运行，并且有千千万万的人进行使用，随着用户量的提升，C/S 的问题就显现出来。例如淘宝的购物网站，其 Server 为 Web Server，但是每逢双 11 等特殊日子，其用户量剧增，这会给服务器造成很大的压力，很可能出现无法访问等问题，因此，各种技术上的解决方案也随着 C/S 架构的系统遇到的问题而相继出现，通过服务器实现负载均衡、缓冲等就是其中的一种解决方案。为了了解服务器端的解决方案，我通过在 nginx 服务器上部署自己用 flask 框架编写的 C/S 类型的项目进行实践，本文则就该实践过程而撰写的实践报告。

nginx 介绍

nginx 是俄罗斯人开发的一个 HTTP 和反向代理服务器，也可以作为邮件代理服务器，目前已经在很多流量很大的网站上使用了很长时间了，如俄罗斯的 Yandex、Mail.Ru 等，国内的淘宝等，nginx 都在这些网站取得了很大的成功。

其基本的 HTTP 服务器特性包括：处理静态文件，索引文件以及自动索引；打开文件描述符缓存；使用缓存加速反向代理；简单负载均衡以及容错；远程 FastCGI，uwsgi，SCGI，和 memcached 服务的缓存加速支持；简单的负载均衡以及容错；模块化的架构。过滤器包括 gzip 压缩、ranges 支持、chunked 响应、XSLT，SSI 以及图像缩放。在 SSI 过滤器中，一个包含多个 SSI 的页面，如果经由 FastCGI 或反向代理处理，可被并行处理；支持 SSL，TLS SNI。

其他的 HTTP 服务器特性包括：基于名字和 IP 的虚拟主机；Keep-alive 和 pipelined 连接支持；灵活的配置；重新加载配置以及在线升级时，不需要中断正在处理的请求；自定义访问日志格式，带缓存的日志写操作以及快速日志轮转；3xx-5xx 错误代码重定向；重写（rewrite）模块：使用正则表达式改变 URI；根据客户端地址执行不同的功能；基于客户端 IP 地址和 HTTP 基本认证机制的访问控制；支持验证 HTTP referer；支持 PUT、DELETE、MKCOL、COPY 以及 MOVE 方法；支持 FLV 流和 MP4 流；速度限制；来自同一地址的同时连接数或请求数限制；嵌入 Perl 语言。

C/S 架构的项目介绍

为了研究 nginx 服务器的相关功能，研究其负载均衡、缓冲区、会话等相关的共享问题，我将之前用 Python Flask 写的一个小网站进行部署，使用 gunicorn 作为内部服务器，并通过 nginx 服务器从外界接收请求并进行转发。

my_website

在该报告中，我的 web 项目名字叫做”my_website”，该项目的前端页面主要采用 bootstrap+jquery+less 进行编写，数据库使用 mysql，项目结构为：

```
--my_website
   --admin
      --略。
   --site
      --confs
      --my_website
         --my_website
            --biz
            --data
            --models
            --static
            --templates
            --utils
            --views
```

```
            __init__.py
        --docs
        --resources
          config.py
          run.py
      --logs
```

gunicorn.sh

在完成 Flask 项目的编写之后，可以开始着手部署 my_website。首先，安装好 gunicorn，并编写 gunicorn.sh 脚本，如下：

```
#!/bin/bash

PAGE_INSTANCE=~/my_website/site/my_website

LOG_FILE=~/my_website/site/logs/gunicorn.log
PID_FILE=~/my_website/site/logs/gunicorn.pid
CALL_INFO="$0 $*"


if [ $(whoami) == 'root' ]; then
    info "Error: Don't run this script as 'root' !"
    exit 1
fi

function tail_log {
    sleep 0.5
    (
        set -x
        tail ${LOG_FILE}
    )
}

function do_start {
    logger start gunicorn by ${CALL_INFO}
    (
        cd ${PAGE_INSTANCE}
        gunicorn --workers=4 --bind 127.0.0.1:6666 --log-file=${LOG_FILE}
--pid=${PID_FILE} --daemon my_website:app
    )
    echo "gunicorn start..."
    tail_log
}


function do_stop {
    logger stop gunicorn by ${CALL_INFO}
    if [ -f ${PID_FILE} ]; then
        PID=`cat ${PID_FILE}`
        rm ${PID_FILE}
        (
            set -x
```

```
            kill -15 $PID
        )
        sleep 1
    else
        echo "PID file '${PID_FILE}' not exists."
    fi
    (
        set -x
        killall -v gunicorn
    )
    echo "gunicorn stopped..."
    tail_log
}


function do_reload {
    logger reload gunicorn by ${CALL_INFO}
    if [ -f ${PID_FILE} ]; then
        PID=`cat ${PID_FILE}`
        (
            set -x
            kill -HUP $PID
        )
    else
        echo "Error: PID file '${PID_FILE}' not exists!"
        (
            set -x
            killall --signal HUP gunicorn -v
        )
    fi
    echo "gunicorn reload..."
    tail_log
}

function do_status {
    echo call info: ${CALL_INFO}
    (
        set -x
        netstat --protocol=unix -nlp | grep python
        tail ${LOG_FILE}
    )
}

case "$1" in
    start)
        do_start
        ;;
    stop)
        do_stop
        ;;
    status)
        do_status
```

```
    ;;
  restart)
    do_stop
    do_start
    ;;
  reload)
    do_reload
    ;;
  *)
    echo $"Usage: $0 {start|stop|restart|reload|status}"
    exit 1
esac
```

在编写 gunicorn.sh 结束后，只需要进入该文件所在目录，执行“./gunicorn.sh start”即可运行 my_website。

my_website.conf

通过 gunicorn 运行 my_website 之后，开始进行 nginx 的部署，首先，我下载 nginx 并解压到”/etc/nginx”目录中，然后开始为 my_website 通过 nginx 进行请求转发编写 nginx 的配置文件 my_website.conf，如下：

```
server {
    listen 80;
    client_max_body_size 4G;
    #my_website 的域名
    server_name my_website.test www.my_website.test;

    keepalive_timeout 5;

    root /home/routh/my_website/site/my_website/my_website/;

  #logs
  access_log /home/routh/my_website/site/logs/my_website.access.log
  error_log /home/routh/my_website/site/logs/my_website.error.log

  #默认请求
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://0.0.0.0:6666
        proxy_redirect off;
  }

    error_page 500 502 503 504 /500.html;
    location = /500.html {
        root /home/routh/my_website/site/my_website/my_website/templates/;
    }
}
```

在编写完该文件，将其放入/etc/nginx/sites-available 中，并在/etc/nginx/sites-enabled 中创建同名的链接指向该文件，如下：

sudo ln -s /etc/nginx/sites-available/my_website.conf /etc/nginx/sites-

enabled/my_website .conf
最后，通过命令行运行“service nginx start”即可启动 nginx 服务器，通过浏览器访问 my_website.test 即可对该 web 项目进行访问。

nginx 的负载均衡

在以上的操作中，只是简单的实现通过 nginx 服务器实现请求转发等基本功能，若随着用户量的增加，请求的连接数量也增加，服务器的相应延时也会随之增加，当达到一定的程度，则只能采用多台服务器进行连接的处理，而通过编写 nginx 配置文件，便能将用户的请求转发到不同的服务器进行相应的处理。因此，我将 my_website 以同样的方式部署到另两台台服务器上，两台服务器的 IP 地址为 192.168.1.121、192.168.1.123 和 192.168.1.124(局域网地址，测试之前，先将域名和对应的 IP 地址写入/etc/hosts 文件中，并通过局域网中其他 PC 机进行测试)，其中数据库存放在第一台服务器上(192.168.1.121)，第二台服务器连接 mysql 的配置命令为：
SQLALCHEMY_DATABASE_URI = 'mysql+mysqldb://my_website:my_website2014@192.168.1.121/my_website?charset=utf8'
在部署好第二台服务器之后，便开始编写 nginx 配置文件，使得用户请求在两台服务器中实现负载均衡。第三台服务器进行同样的处理
修改之前编写的 my_website.conf 文件，其中修改的主要内容如下：

```
#新增加的
upstream my_website.test {
    server 192.168.1.121:6666       weight=2;
    server 192.168.1.123:6666;
    server 192.168.1.124:6666;

    server 192.168.1.125:6666   backup;
    server 192.168.1.126:6666   backup;
}

#默认请求
location / {
   #新增加的
   proxy_pass http://my_website.test
}
```

通过访问 http://my_website.test，**nginx** 服务器对用户的请求实现了负载均衡，若有 **4** 个请求，其中 **2** 个请求转向 **192.168.1.121:6666** 进行处理，另外两个请求分别转向 **192.168.1.123:6666** 和 **192.168.1.124:6666** 进行处理；若主服务器出现故障，则将请求交给 **192.168.1.125:6666** 和 **192.168.1.126:6666** 两台备用服务器处理。

nginx 的缓冲区

通过配置了 nginx 的负载均衡以后，可以将用户的请求有效地分发到不同的服务器上进行处理，不仅缓解了单个服务器的压力，也有效的利用了服务器的空闲资源，达到了更好的效果。但是，用户的每次请求都必须转到 gunicorn 服务器上进行处理，并返回用户所需要的页面，若在 nginx 的配置文件中增加 cache 的功能，则在处理用户的大部分请求时，可以直接从 nginx 的 cache 中返回用户所需要的页面，减少的请求转发和页面生成所消耗的时间，提高请求处理的效率，也能让用户获得更好的体验。
因此，为了配置 nginx 缓存，首先编辑/etc/nginx/nginx.conf 文件，如下：

```
user www-data;

worker_processes 4;

pid /var/run/nginx.pid;
```

```
events {
	worker_connections 768;
	# multi_accept on;
}

http {

	##
	# Basic Settings
	##

	sendfile on;
	tcp_nopush on;
	tcp_nodelay on;
	keepalive_timeout 65;
	types_hash_max_size 2048;

	include /etc/nginx/mime.types;
	default_type application/octet-stream;

	##
	# Logging Settings
	##

	access_log /var/log/nginx/access.log;
```

```
    error_log /var/log/nginx/error.log;



    ##

    # Gzip Settings

    ##



    gzip on;

    gzip_disable "msie6";


    ##
    # cache
    ##
    proxy_connect_timeout 5;
    proxy_read_timeout 60;
    proxy_send_timeout 5;
    proxy_buffer_size 16k;
    proxy_buffers 4 64k;
    proxy_busy_buffers_size 128k;
    proxy_temp_file_write_size 128k;
    proxy_temp_path /home/routh/my_website/temp_dir;
    proxy_cache_path /home/routh/my_website/cache levels=1:2
keys_zone=cache_one:200m inactive=1d max_size=30g;
    ##
    # cache end
    ##


    include /etc/nginx/conf.d/*.conf;

    include /etc/nginx/sites-enabled/*;

}
```

然后修改 my_website.conf 文件，其中修改的主要内容如下：

```
#新增加的
location ~ .*\.(gif|jpg|png|htm|html|css|js|flv|ico|swf)(.*) {
    proxy_pass http://my_website.test;
    proxy_redirect off;
    proxy_set_header Host $host;
    proxy_cache cache_one;
    proxy_cache_valid 200 302 1h;
    proxy_cache_valid 301 1d;
    proxy_cache_valid any 1m;
```

```
    expires 30d;
}
```

通过配置 http 段，设置了临时目录为/home/routh/my_website/temp_dir；设置缓存目录为/home/routh/my_website/cache，该缓存目录为二级目录，大小为 200M，非活动时间为 1 天，最大容量为 30G。

通过配置 server 段，设置了缓存共享内存区块，即 keys_zone 的名称；设置了 http 状态码为 200，302，其缓存时间为 1 小时；设置失效时间为 30 天；该增加的 location 内容主要针对请求静态文件，若请求动态文件，则不产生缓存，直接进行请求转发。

nginx 中 cache 和 session 的共享问题

因为现在具有多台服务器同时运行着我的 web 项目 my_website，因此，会造成 cache 和 session 的共享问题，即多台服务器之间的缓存文件和 session Id 不同步。为此，我通过在配置文件中增加 ip_hash 来解决该问题，如下红色部分：

```
upstream my_website.test {

    ip_hash;

    server 192.168.1.121:6666       weight=2;
    server 192.168.1.123:6666;
    server 192.168.1.124:6666;

    server 192.168.1.125:6666   backup;
    server 192.168.1.126:6666   backup;
}

#默认请求
location / {
    #新增加的
    proxy_pass http://my_website.test
}
```

ip_hash 是一种算法，即每个请求按访问 IP 的 hash 结果进行分配，这样来自同一个 IP 的用户请求将会固定地访问一个服务器，这有效地解决动态网页存在的 session 共享问题，也可以保证用户可以直接访问其缓存文件，提高命中率。而在我部署的环境中，nginx 服务器作为最外端的服务器，其接受到的 IP 是真实用户的 IP，这就可以避免同一个 IP 用户访问出现的是其他服务器的 IP 地址，也保证了 ip_hash 的有效性。