1. nginx 架构 1

简介 1

1.1 配置文件 1

1.1.1 启动配置 1

1.1.2 http服务器配置 1

1.1.2.1 directive---location 1

1.1.2.2 directive—try\_files 1

1.1.2.3 directive—use 1

1.1.2.4 directive --- ssl\_certificate file 1

1.1.2.5 directive ---proxy\_pass url 2

1.1.2.6 directive ---auth\_basic string|off 2

1.1.3 nginx与fastcgi通信配置 3

1.1.3.1 tcp 方式 3

1.1.3.2 unix socket(unix domain socket)方式 3

2. 命令行 3

2.1 启动命令 3

2.2 测试nginx配置正确 3

3. 架构 3

3.1 进程架构 3

3.2 Process-based vs Event-driven 3

3.2.1 differences 3

# 1. nginx 架构

## 简介

nginx提供module、directives、variables三种。

## 1.1 配置文件

### 1.1.1 启动配置

/etc/init/nginx.conf

/etc/nginx/nginx.conf:

nginx服务器的启动配置参数，如：进程数控制，http服务控制、邮件服务器控制。

对于http服务，指定了安全协议、日志设定、压缩设置等。

### 1.1.2 http服务器配置

/etc/nginx/nginx.conf中指定了nginx的http的基本规则，同时在文件/etc/nginx/sites-enabled目录中具体设置了http服务，使用include命令进行引入。

Nginx.conf配置文件中设定了user角色, 当nginx服务器启动时， master进程则一般由root来启动，worker进程则根据user指令指定的用户身份进行创建。

一般user指定的用户角色的权限较低，只有read和execute的权限

Nginx服务器配置文件中存在目录sites-available和目录sites-enabled, 对于sites-available目录中的所有配置为虚拟服务器所有可用的，但是并未被nginx正式使用的配置。Sites-enabled的目录中的配置则是sites-available中的部分的syslink，被nginx.conf文件引入，被实际运用的规则

目录conf.d/中的文件则定义了一些全局server的配置，如配置全局server接收客户端的请求体／上传文件的大小。

对于nginx.conf配置文件，存在directives  
pid： 用于制定pid的存储路径文件（pid /run/nginx/pid.log）

#### 1.1.2.1 directive---location

参考： <https://www.digitalocean.com/community/tutorials/understanding-nginx-server-and-location-block-selection-algorithms>

<http://seanlook.com/2015/05/17/nginx-location-rewrite/>

server\_name \_： 表示所有的domain都成功match改server block.

location表明了请求nginx服务器的路径的匹配。

----符号～： 表明实行正则匹配。匹配查找最佳的路径。区分大小写。

----符号^～： 表明实行正则匹配。 匹配成功，停止往下查找

----符号～\*：表明以什么结尾。不区分大小写。

----无符号： 表示路径为前缀

----符号 =： 表示相同

----符号@：表示定义一个命令的location

当无符号前缀匹配和符号～正则匹配的规则相同，则选取正则匹配的路径

每个location和其内部继承的directive共同决定了请求会如何进行处理。

Location的internal redirect:

---index

---try\_files

---rewrite

---error\_page

对于内部的重导向，多数作用于 = match；

try\_files表示处理请求的location将重新被评估。 其最后可以提供url，用于nginx做内部转发。

Location 内部的root指令，会覆盖外层定义的root内容。

#### 1.1.2.2 directive—try\_files

表明查看请求的文件是否存在。其后可以跟多个文件或者带有目录的文件。 匹配查询时按照顺序进行一次查找。同时注意，请求文件的路径是根据root或者alias指令组装的path。 0.7.51 版本以后，try\_files其后可以添加命令的location块或者是http相应码。

Try\_files $uri $uri/index.html @budget

Location @budget {

Proxy\_pass http://unix:/home/test/dockers/svr-place/master/web.sock

}

#### 1.1.2.3 directive—use

处理链接请求。使用默认的即可。

#### 1.1.2.4 directive --- ssl\_certificate file

其后添加ssl的文件。

#### 1.1.2.5 directive ---proxy\_pass url

作用域： location、 if in location、limit——except

设置代理服务器的地址。如何<http://localhost:8000/uri>。注意的是，

对于unix风格的domain，追加unix:符号。如： http://unix:/tmp/backend.socket :/uri/;

使用代理传递：

---：使用<http://domain/path>风格，表明是将请求代理到新的服务，

---：使用<http://unix/directory_of_machine>风格，表明是将请求代理到本服务器的某个sock服务程序。

注意：

当location块同时出现rewrite和proxy\_pass，那么最终请求会代理到新的域名服务器上，同时rewrite操作导致的path的改变会附加到proxy\_pass提供的域名服务器上。

相类似的directive：

---proxy\_pass\_request\_headers on|off： 表示是否将original url的headers传递给代理服务器。

#### 1.1.2.6 directive ---auth\_basic string|off

表示需要进行用户名和密码的验证，其后的字符串为提示信息。

---auth\_basic\_user\_file file-path: 表示密码的路径

1.1.2.7 rewrite、proxy\_pass、location

rewrite只对除去域名和url的参数外的路径进行重写和重定向功能。只存在与server、location、if块中。Rewrite是在同一域名中实现更改资源路径的命令。Location可以做控制访问或者反向代理，proxy\_pass到其他机器。

1. 执行server块的rewrite指令
2. 执行location匹配
3. 执行选定的location中的rewrite指令

flag的标志位

last一般在server和if块中，表示完成rewrite， 改变的url进一步进行匹配

break停止后续rewrite的执行，对于改变的url，新的url不会在进行匹配

redirect 临时重定向，

permanent 永久重定向

### 1.1.3 全局变量

#### 1.1.3.1 $uri vs $request\_uri

uri表明的是统一资源定位符，在http请求中，指代的是资源的path。

$request\_uri指的是从客户端发送到服务器端的请求的路径。$uri表明的是服务器经过处理后的路径。

### 1.1.6 nginx与fastcgi通信配置

#### 1.1.6.1 tcp 方式

#### 1.1.6.2 unix socket(unix domain socket)方式

unix中使用us方式进行系统进程间的通信（IPC）, 一般以.sock作为socket的唯一标示，两个进程同时饮用同一个socket描述符，即可建立通道进行通信

# 2. 命令行

## 2.1 启动命令

systemctl restart nginx 或者service nginx restart, 对于老版的ubuntu，使用/etc/init.d/nginx restart

## 2.2 测试nginx配置正确

Nginx –t –c /etc/nginx/nginx.conf

# 3. 架构

# 3.1 进程架构

---master process：读取配置文件、bind port、

---cache loader：负责加载disk\_based cached into memory.

---cache manager: 从disk caches中清除entries

---worker processes: 处理链接请求，读写请求和其他业务逻辑

每个worker process处理多个connections， 减少context numbers的上下文交换次数。

每个woker process是单线程。进程之间通过使用shared memory进行交流。Shared memory中包含了shared cache data、session persistence、、、

## 3.2 Process-based vs Event-driven

reference: <http://www.aosabook.org/en/nginx.html>

### 3.2.1 differences

根据链接请求，使用为每个链接请求分配一个进程或者线程的方式。对于massive web request, 过于消耗资源。