本文简单介绍nginx conf文件的结构，已经如何进行配置：

* 如何配置nginx提供静态内容，
* 如何配置nginx作为代理服务器，
* 如何配置转发请求到FastCGI服务

Nginx进程模型：1个主进程，n个工作进程，主进程负责配置和工作进程的管理，实际的请求由工作进程进行处理。Nginx是基于事件驱动和多路复用的工作模型。

**1. nginx启停**

nginx的启动可以直接执行nginx的bin文件， 当nginx启动后，可以通过-s 参数来控制nginx

nginx -s reload #重新加载配置文件

nginx -s reopen #重新打开log文件

nginx -s stop #快速关闭nginx服务

nginx -s quit #优雅的关闭nginx服务，等待工作进程处理完所有的请求

* 1
* 2
* 3
* 4

Nginx重新加载配置文件的过程：主进程接受到加载信号后：

* 首先会校验配置的语法，然后生效新的配置，
* 如果成功，则主进程会启动新的工作进程，同时发送终止信号给旧的工作进程。
* 否则主进程回退配置，继续工作。

在第二步，旧的工作进程收到终止信号后，会停止接收新的连接请求，知道所有现有的请求处理完，然后退出。

**2. nginx.conf文件的结构**

nginx的配置由特定的标识符(指令符)分为多个不同的模块。   
指令符分为**简单指令**和**块指令**。

* 简单指令格式：[name parameters;]
* 块指令格式：和简单指令格式有一样的结构，但其结束标识符不是分号，而是大括号{},块指令内部可以包含simple directives 和block directives, 可以称块指令为上下文(e.g. events, http, server, location)

conf文件中，所有不属于块指令的简单指令都属于main上下文的，http块指令属于main上下文，server块指令http上下文。

**2.1 配置静态访问**

Web server很重要一部分工作就是提供静态页面的访问，例如images, html page。nginx可以通过不同的配置，根据request请求，从本地的目录提供不同的文件返回给客户端。   
打开安装目录下的nginx.conf文件，默认配置文件已经在http指令块中创建了一个空的server块，在nginx-1.8.0中的http块中已经创建了一个默认的server块。内容如下：

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

通常情况下，conf文件存在多个server块，通过listen的端口(默认是80端口)和server\_name进行区分, 对不同的请求提供不同的服务，如下：

server {

listen 80;

server\_name a.example.org;

...

}

* 1
* 2
* 3
* 4
* 5

listen指令的格式如下：

| **Note** | **Description** |
| --- | --- |
| Syntax: | listen address[:port] [default\_server] [ssl] [http2 | spdy] [proxy\_protocol] [setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept\_filter=filter] [deferred] [bind] [ipv6only=on|off] [reuseport] [so\_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];  listen port [default\_server] [ssl] [http2 | spdy] [proxy\_protocol] [setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept\_filter=filter] [deferred] [bind] [ipv6only=on|off] [reuseport] [so\_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];  listen unix:path [default\_server] [ssl] [http2 | spdy] [proxy\_protocol] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept\_filter=filter] [deferred] [bind] [so\_keepalive=on|off |
| Default: | listen \*:80 | \*:8000; |
| Context: | server |

listen指令的参数：可以是ip, hostname, ip/hostname:port, port, UNIX-domain socket.例如：

listen 127.0.0.1:8000;listen 127.0.0.1;listen 8000;listen \*:8000;listen localhost:8000;

listen unix:/var/run/nginx.sock;

* 1
* 2

server块内部的listen和server\_name不能和其他server块的完全相同，否则启动加载配置的时候会出现如下错误：

server {

listen 80;

server\_name a.example.org;

...

}

server {

listen 80;

server\_name a.example.org;

...

}

#nginx -s reload

nginx: [warn] conflicting server name "a.example.org" on 0.0.0.0:80, ignored

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

当nginx决定了哪一个server处理客户端请求后，nginx会解析request header中URI(**这里以及后面提到的大部分都是指相对URI**)，然后匹配server块中的location指令的参数，匹配规则下一节会介绍。例如下例：

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

location块指令会用其参数与客户端请求的URI进行匹配，匹配的URI请求会被定向到root指令定义的特殊本地文件系统目录中，重定向规则为：将URI添加到root参数后面，生成一个本地文件路径，即：**root参数 + URI请求**。这里示例参数”/”会匹配所有的请求，一般都会默认存在。示例定位后的目录为html/，默认是定位到安装目录的路径下的html/。这里location块指令内部的两个简单指令的含义是：

* root 指定重定向后uri的资源查找路径，这里html为相对路径，相对于nginx的安装目录。
* index指定首页index文件的名称，可以配置多个，参数以空格分开，按配置顺序查找。

默认的在nginx安装目录下都会存在一个html目标，在我电脑中为：/usr/local/nginx-1.8.0/html， 然后里面存在默认的nignx的欢迎界面，例如我安装nginx后直接启动nginx bin文件，然后访问对应的域名就得到了如下的页面：   
   
如果有多个location指令块匹配到，nginx的选择策略是the longest prefix最长前缀匹配原则。   
例如，当上述的server中再增加一个location块，匹配参数为”/htdocs/”，重定向的资源路径为如下配置，

server {

listen 80;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

location /htdocs {

root /home/anonymalias;

index index.html;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

当访问[http://anonymalias.oicp.net:8008/htdocs/](http://blog.csdn.net/anonymalias/article/details/%5bhttp:/anonymalias.oicp.net:8008/htdocs/)，就会匹配到/home/anonymalias/htdocs/index.html

**2.2 location指令**

nginx的location指令是配置的核心，用于匹配client请求uri的path部分，然后对不同的请求提供不同的静态内容，或者通过反向代理重定向到内部的server。

对于client的request， nginx会进行预处理，nginx首先对采用 ’%XX’(uri采用%+十六进制格式用于在浏览器和插件中显示非标准的字母和字符) 格式文本编码的uri进行解码。然后处理path中的相对路径符号’.’和‘..’，然后对于含有两个及以上的’/’压缩成一个’/’。 这样处理完后会得到一个干净的path，然后会用这个path会在server的location指令的参数进行匹配。

location 语法：

location [ = | ~ | ~\* | ^~ ] uri { ... }

location @name { ... }

* 1
* 2
* 3

**location指令的参数**：

* **可以是一个前缀字符串**，参数修饰符分为：   
  + 无任何修饰符；
  + “=” 修饰符：定义一个精确匹配，匹配是精确匹配优先级最高；
  + “^~” 修饰符：对应的参数作为最长字符串匹配到后，不会继续去搜索参数为正则表达式的location
* **也可以是一个正则表达式**，参数修饰符分为：   
  + ”~” 修饰符： location参数部分大小写敏感；
  + “~\*” 修饰符：location参数部分大小写不敏感；

location ~\* \.(gif|jpg|jpeg)$ {

[ configuration A ]

}

location ~ \.(gif|jpg|jpeg)$ {

[ configuration B ]

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

若请求URI为/images/a.JPG，只能匹配A。

**location匹配的过程**：   
1. 首先nginx会把request的uri在location正常字符串参数中匹配出符合的最长字符串，并保存这个结果；   
2. 如果最长前缀匹配结果前面有 ”^~”修饰符，那么停止继续搜索；   
3. 如果最长匹配结果前有”=”修饰符，也会停止继续搜索；   
4. 接下来，去匹配参数为正则表达式的所有location，根据location的配置顺序，在匹配到第一个正则表达式时，即停止搜索其他的正则表达式；   
所以如官网上的下例：

location = / {

[ configuration A ]

}

location / {

[ configuration B ]

}

location /documents/ {

[ configuration C ]

}

location ^~ /images/ {

[ configuration D ]

}

location ~\* \.(gif|jpg|jpeg)$ {

[ configuration E ]

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 当请求”/”时，匹配到A
* 当请求”/index.html”，会匹配到B；
* 当请求”documens/document.html”，会匹配到C；
* 当请求”/images/1.gif”，会匹配到D；**匹配流程**：首先会匹配到D，由于D的location的参数含有修饰符”^~”，当匹配到D后，不会再搜索参数为正则表达式的location；
* 当请求”/documents/1.jpg”，会匹配到E；**匹配流程**：首先会匹配到C，此时会保存C的匹配结果，然后继续搜索参数为正则的location，结果发现E匹配上了，那么会丢弃之前匹配到的C

**2.3 配置proxy server**

nginx使用很频繁的另一个服务就是作为代理服务器，提供将外部client的请求反向代理给内部被代理的 server， 然后接收proxied server 的响应，并回包给client。

proxy server的配置和上面配置提供静态web服务的结构都是一致的。proxy server的作用就是将一个request 转发到一个internal server，然后将internal server的response回传给client。关于如何将internal server的回包转发给client是nignx底层做的事情，不需要要进行相关的配置，在配置层面上，我们要做的事情就是如何将一个client请求转发给internal server。   
下面是nginx官方最简单的一个proxy server的配置示例：

server {

location / {

proxy\_pass http://localhost:8080/;

}

location ~ \.(gif|jpg|png)$ {

root /data/images;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

配置的含义：所有URI中已.gif, .jpg, .png结尾的请求都会被映射到/data/images本地磁盘目录，把所有其他的URI请求透传给配置的被代理的server: <http://localhost:8080/>。

nginx反向代理模块有以下几个很重要的参数：

* [proxy\_pass](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass)：   
  该指令是反向代理的基本指令，用于设置代理服务器的协议和地址；对于一个client的请求，proxy\_pass指令通过以下方式进行uri的转发：   
  + 如果proxy\_pass指令的参数没有URI，那么请求的URI会被原样的传递给internal server。
  + 如果proxy\_pass指令的参数含有URI，client请求的URI匹配该location的部分将会被proxy\_pass的path参数替换。   
    例如：请求为127.0.0.1/name/index.html 会被转发为：127.0.0.1/remote/index.html

location /name/ {

proxy\_pass http://127.0.0.1/remote/;

}

* 1
* 2
* 3

说明：1.1.12之前，如果proxy\_pass指令不含有URI，那么原始请求中的URI在某些情况下会被处理后再传递给proxy server。

在某些情况下,请求URI部分是否被替换是不能确定:   
1. 在location指令的参数是一个正则表达式的时候；这个情况下，proxy\_pass的参数应该不喊URI部分。   
2. 当URI在location指令模块中被rewrite指令改变后，？？？

* [proxy\_pass\_header](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass_header)：   
  语法：proxy\_pass\_header field   
  field参数是http所有的header名字，具体可以参考： [HTTP/1.1协议][1] P100 chapter14关于HTTP协议所有header field的定义。

该指令用于特定的http header从proxy server传回给client。因为默认情况下有些header是不会回传给client的。默认情况下，nginx不会把proxy server传回的参数中的”Data”, “Server”, “X-Pad”, “X-Accel-…”回传给client。其中proxy\_hide\_header指令，用于限定哪些header不回传给client。

* [proxy\_set\_header](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_set_header)：   
  语法：proxy\_set\_header field value;   
  该指令用于将client传递给proxy server的request header重新定义或者添加字段。Value可以是文本，变量和两者结合。   
  如果set 的header field的值为空，那么这个header是不会传递给proxy server的，例如：

proxy\_set\_header Accept-Encoding "";

* 1

默认情况下，只有以下两个字段会被重新定义：

proxy\_set\_header Host $proxy\_host;

proxy\_set\_header Connection close;

* 1
* 2

其实默认定义往往不是我们期望要的，例如：Host header被置为了proxy\_pass指令设置的代理server的url和port。但在实际应用中，我们往往是Host保留client的信息：

proxy\_set\_header Host $host;

* 1

一般情况下proxy server的location会进行以下基本的设置：

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

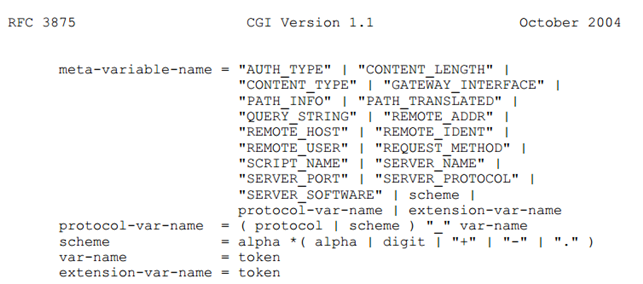
* 1
* 2
* 3
* 4

**2.5 配置FastCGI proxying**

**什么是CGI**：   
CGI全称：通用网关接口(Common Gateway Interface)，是Web服务器与外部应用程序(CGI程序)之间的接口，说的那么晦涩其实CGI就是：**Web服务器与CGI程序通信的协议标准**。很多人以及网上的很多文章都没有分清CGI和CGI程序这两个不同的概念。   
CGI1.1 RFC3875文档有详细的CGI协议的标准，[下载链接](http://www.rfc-editor.org/info/rfc3875).

Web Server在CGI程序调用过程中，扮演着应用程序网关的角色。Web服务器在收到客户端请求后，会将client的http请求根据CGI标准转换成一个CGI请求，然后将这个CGI请求交给CGI程序，CGI程序处理完后，Web服务器会将CGI的回包(符合CGI标准)转换成一个client的http回包，下发给client。

关于**CGI请求，有2种传递数据**的途径：**the meta-variables** and **message-body**.

* **Meta-variables元变量**：通过定义系统环境变量的方式，来进行数据传递；如下是RFC定义的基本变量的名称，在文档里有详细的说明   
  
* **Message-body**：通过系统定义的方法：标准输入流或管道。Meta-variables传递受到数据大小的限制，

对应于上面**两种CGI数据传输途径**，有两种对应request方法，由Meta-variable中的系统变量REQUEST\_METHOD的值来决定：   
\* GET 方法对应Meta-variable方式的数据传输；   
\* POST方法对应Message-body方式的数据传输；

当然REQUEST\_METHOD还有其他方法，这里不在阐述，在CGI1.1 RFC3875文档的P10 Chapter4:The CGI Request中有详细讲解。

**CGI程序执行的流程**如下，fork-and-execute模式 ：   
1．web 服务器收到客户端（浏览器）的请求Http Request，启动CGI程序，并通过环境变量、标准输入传递数据   
2．cgi进程启动解析器、加载配置（如业务相关配置）、连接其它服务器（如数据库服务器）、逻辑处理等   
3．cgi程将处理结果通过标准输出、标准错误，传递给web 服务器   
4．web 服务器收到cgi返回的结果，构建Http Response返回给客户端，并杀死cgi进程

前面已经介绍过web服务器与cgi程序之间：通过环境变量、标准输入、标准输出、标准错误互相传递数据。

说了那么多CGI是什么以及CGI程序工作方式，那么现在该说**什么是FastCGI了**，   
FastCGI：快速通用网关接口(Fast Common Gateway Interface)是通用网关接口（CGI）的改进FastCGI很简单，因为**它实际上是在CGI基础上增加了一些的扩展**，所以它同样是Web服务器与CGI程序通信的协议标准，只不过，它是一个建议标准。FastCGI致力于减少Web服务器与CGI程式之间互动的开销，从而使服务器可以同时处理更多的Web请求。与使用CGI时，为每个请求创建一个新的进程不同，FastCGI使用持续的进程来处理一连串的请求。这些进程由FastCGI进程管理器管理，而不是web服务器。

FastCGI的开发商都致力于传播的FastCGI作为一个开放的标准。为此，对于当下流行且免费的web server，FastCGI开发者提供了免费FastCGI应用程序库（C / C ++，Java，Perl，TCL）和升级模块（Apache，ISS，lighttpd）。

**FastCGI的执行流程**：   
1．Web 服务器启动时载入初始化FastCGI执行环境 。 例如IIS ISAPI、apache mod\_fastcgi、nginx ngx\_http\_fastcgi\_module、lighttpd mod\_fastcgi   
2．FastCGI进程管理器自身初始化，启动多个CGI解释器进程并等待来自Web 服务器的连接。启动FastCGI进程时，可以配置以ip和UNIX 域socket两种方式启动。   
3．当客户端请求到达Web 服务器时， Web 服务器将请求采用socket方式转发到 FastCGI主进程，FastCGI主进程选择并连接到一个CGI解释器。Web 服务器将CGI环境变量和标准输入发送到FastCGI子进程。   
4．FastCGI子进程完成处理后将标准输出和错误信息从同一socket连接返回Web 服务器。当FastCGI子进程关闭连接时，请求便处理完成。   
5．FastCGI子进程接着等待并处理来自Web 服务器的下一个连接。

由于 FastCGI 程序并不需要不断的产生新进程，可以大大降低服务器的压力并且产生较高的应用效率。它的速度效率最少要比CGI 技术提高 5 倍以上。它还支持分布式的部署， 即 FastCGI 程序可以在web 服务器以外的主机上执行。

总结：CGI 就是所谓的短生存期应用程序，FastCGI 就是所谓的长生存期应用程序。FastCGI像是一个常驻(long-live)型的CGI，它可以一直执行着，不会每次都要花费时间去fork一次(这是CGI最为人诟病的fork-and-execute 模式)。

由于nginx不能像apache那样直接执行外部可执行程序，所以nginx不直接支持CGI程序，但nginx可以支持FastCGI代理，因为nignx支持反向代理， nginx FastCGI 代理的简单配置和反向代理类似，只是指令不一样，如下：

server {

location / {

fastcgi\_pass localhost:9000;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

fastcgi\_param QUERY\_STRING $query\_string;

}

location ~ \.(gif|jpg|png)$ {

root /data/images;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* fastcgi\_pass指令：替代proxy\_pass，用于将cgi请求转发到对应的fastcgi进程管理器所在的端口
* fastcgi\_param指令：用于set参数，用于传递给FastCGI程序，参数名字遵循CGI协议Meta-variable，因为FastCGI的基础是CGI。