。

1 FastCGI

1.1 CGI

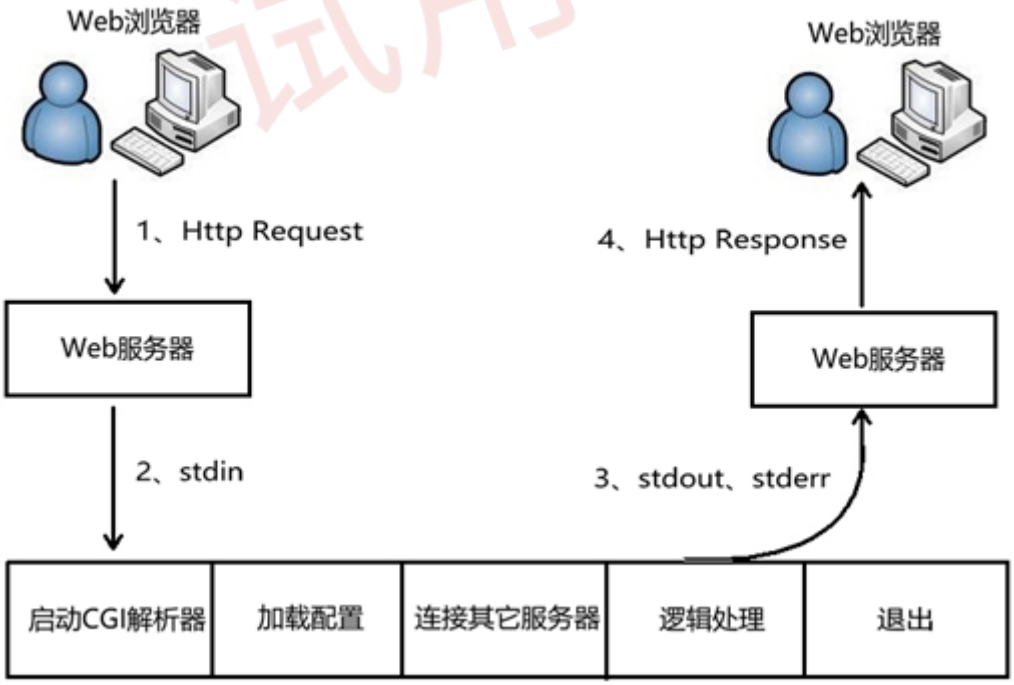
1.1.1 什么是CGI

通用网关接口(Common Gateway Interface、CGI)描述了客户端和服务器程序之间传输数据的一种标准，可以让一个客户端，从网页浏览器向执行在网络服务器上的程序请求数据。

CGI独立于任何语言的，CGI 程序可以用任何脚本语言或者是完全独立编程语言实现，只要这个语言可以在这个系统上运行。Unix shell script、Python、 Ruby、PHP、 perl、Tcl、 C/C++和 Visual Basic 都可以用来编写 CGI 程序。

最初，CGI 是在 1993 年由美国国家超级电脑应用中心(NCSA)为 NCSA HTTPd Web 服务器开发的。这个 Web 服务器使用了 UNIX shell 环境变量来保存从 Web 服务器传递出去的参数，然后生成一个运行 CGI 的独立的进程。

1.1.2 CGI处理流程



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进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

web服务器与CGI通过环境变量、标准输入、标准输出、标准错误互相传递数据。在遇到用户连接请求：

- 先要创建CGI子进程，然后CGI子进程处理请求，处理完事退出这个子进程：fork-and-execute
- CGI方式是客户端有多少个请求，就开辟多少个子进程，每个子进程都需要启动自己的解释器、加载配置，连接其他服务器等初始化工作，这是CGI进程性能低下的主要原因。当用户请求非常多的时候，会占用大量的内存、cpu等资源，造成性能低下。

CGI使外部程序与Web服务器之间交互成为可能。CGI程序运行在独立的进程中，并对每个Web请求建立一个进程，这种方法非常容易实现，但效率很差，难以扩展。面对大量请求，进程的大量建立和消亡使操作系统性能大大下降。此外，由于地址空间无法共享，也限制了资源重用。

1.1.3 环境变量

GET请求，它将数据打包放置在环境变量QUERY_STRING中，CGI从环境变量QUERY_STRING中获取数据。
常见的环境变量如下表所示：

环境变数	含义
AUTH_TYPE	存取认证类型
CONTENT_LENGTH	由标准输入传递给CGI程序的数据长度，以bytes或字元数来计算
CONTENT_TYPE	请求的MIME类型
GATEWAY_INTERFACE	服务器的CGI版本编号
HTTP_ACCEPT	浏览器能直接接收的Content-types, 可以有HTTP Accept
header定义	
HTTP_USER_AGENT	递交表单的浏览器的名称、版本和其他平台性的附加信息
HTTP_REFERER	递交表单的文本的URL，不是所有的浏览器都发出这个信息，不要依赖它
PATH_INFO	传递给CGI程序的路径信息
QUERY_STRING	传递给CGI程序的请求参数，也就是用"?"隔开，添加在URL
后面的字串	
REMOTE_ADDR	client端的host名称
REMOTE_HOST	client端的IP位址
REMOTE_USER	client端送出来的使用者名称
REMOTE_METHOD	client端发出请求的方法(如get、post)
SCRIPT_NAME	CGI程序所在的虚拟路径，如/cgi-bin/echo
SERVER_NAME	server的host名称或IP地址
SERVER_PORT	收到request的server端口
SERVER_PROTOCOL	所使用的通讯协定和版本编号
SERVER_SOFTWARE	server程序的名称和版本

1.1.4 标准输入

环境变量的大小是有一定的限制的，当需要传送的数据量大时，储存环境变量的空间可能会不足，造成数据接收不完全，甚至无法执行CGI程序。
因此后来又发展出另外一种方法：POST，也就是利用I/O重新导向的技巧，让CGI程序可以由stdin和stdout直接跟浏览器沟通。
当我们指定用这种方法传递请求的数据时，web服务器收到数据后会先放在一块输入缓冲区中，并且将数据的大小记录在CONTENT_LENGTH这个环境变量，然后调用CGI程序并将CGI程序的stdin指向这块缓冲区，于是我们就可以很顺利的通过stdin和环境变数CONTENT_LENGTH得到所有的信息，再没有信息大小的限制了。

2.2 FastCGI

2.2.1 什么是FastCGI

快速通用网关接口(Fast Common Gateway Interface / FastCGI)是通用网关接口(CGI)的改进，描述了客户端和服务器程序之间传输数据的一种标准。

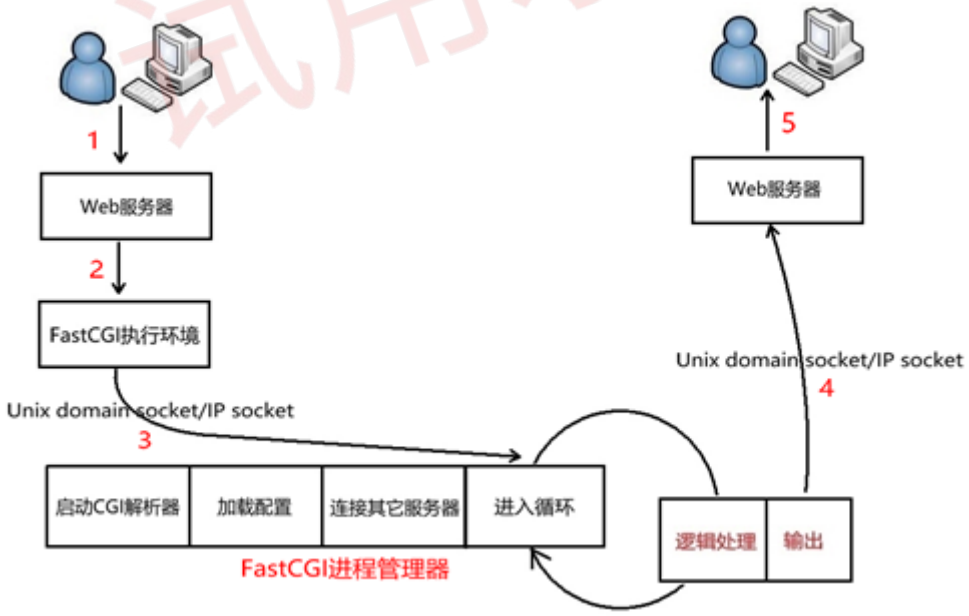
FastCGI致力于减少Web服务器与CGI程式之间互动的开销，从而使服务器可以同时处理更多的Web请求。与为每个请求创建一个新的进程不同，FastCGI使用持续的进程来处理一连串的请求。这些进程由FastCGI进程管理器管理，而不是web服务器。

nginx服务支持FastCGI模式，能够快速高效地处理动态请求。而nginx对应的FastCGI模块为：

- ngx_http_fastcgi_module。

ngx_http_fastcgi_module 模块允许将请求传递给 FastCGI 服务器。

2.2.2 FastCGI处理流程



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。

CGI 是所谓的短生存期应用程序，FastCGI 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)。

2.2.3 进程管理器管理：spawn-fcgi

2.2.3.1 什么是spawn-fcgi

Nginx不能像Apache那样直接执行外部可执行程序，但Nginx可以作为代理服务器，将请求转发给后端服务器，这也是Nginx的主要作用之一。其中Nginx就支持FastCGI代理，接收客户端的请求，然后将请求转发给后端FastCGI进程。

由于FastCGI进程由FastCGI进程管理器管理，而不是Nginx。这样就需要一个FastCGI进程管理器，管理我们编写FastCGI程序。

spawn-fcgi是一个通用的FastCGI进程管理器，简单小巧，原先是属于lighttpd的一部分，后来由于使用比较广泛，所以就迁移出来作为独立项目。

spawn-fcgi使用pre-fork 模型，功能主要是打开监听端口，绑定地址，然后fork-and-exec创建我们编写的FastCGI应用程序进程，退出完成工作。FastCGI应用程序初始化，然后进入死循环侦听socket的连接请求。

2.2.3.2 编译安装spawn-fcgi

spawn-fcgi源码包下载地址：http://redmine.lighttpd.net/projects/spawn-fcgi/wiki
编译和安装spawn-fcgi相关命令：

```
wget http://download.lighttpd.net/spawn-fcgi/releases-1.6.x/spawn-fcgi-
1.6.4.tar.gz
tar -zxf spawn-fcgi-1.6.4.tar.gz
cd spawn-fcgi-1.6.4/
./configure
make
make install
```

如果遇到以下错误：
./autogen.sh: x: autoreconf: not found
因为没有安装automake工具，ubuntu用下面的命令安装即可：
apt-get install autoconf automake libtool
spawn-fcgi的帮助信息可以通过man spawn-fcgi或spawn-fcgi -h获得，下面是部分常用spawn-fcgi参数信息：

参数	含义
f	指定调用FastCGI的进程的执行程序位置
-a	绑定到地址addr
-p	绑定到端口port
-s	绑定到unix domain socket
-C	指定产生的FastCGI的进程数，默认为5(仅用于PHP)
-P	指定产生的进程的PID文件路径
-F	指定产生的FastCGI的进程数(C的CGI用这个)
-u和-g FastCGI	使用什么身份(-u用户、-g用户组)运行，CentOS下可以使用apache用户，其他的根据情况配置，如nobody、www-data等

2.2.4 软件开发套件：fcgi

2.2.4.1 编译安装fcgi

使用C/C++编写FastCGI应用程序，可以使用FastCGI软件开发套件或者其它开发框架，如fcgi。
fcgi下载地址：wget https://fossies.org/linux/www/old/fcgi-2.4.0.tar.gz
编译和安装fcgi相关命令：

```
wget https://fossies.org/linux/www/old/fcgi-2.4.0.tar.gz --no-check-certificate
tar -zxf fcgi-2.4.0.tar.gz
cd fcgi-2.4.1-SNAP-0910052249/
./configure

编译前在libfcgi/fcgio.cpp 文件上添加#include <stdio.h>
make
make install
```

如果编译出现下面问题：

```
fcgio.cpp: In destructor 'virtual fcgi_streambuf::~fcgi_streambuf()':
fcgio.cpp:50:14: error: 'EOF' was not declared in this scope
     overflow(EOF);
              ^
fcgio.cpp: In member function 'virtual int fcgi_streambuf::overflow(int)':
fcgio.cpp:70:72:** error:** 'EOF' was not declared in this scope
          if (FCGX_PutStr(pbase(), plen, this->fcgx) != plen) return EOF;
                                                                        ^
fcgio.cpp:75:14: error: 'EOF' was not declared in this scope
     if (c != EOF)
              ^
fcgio.cpp: In member function 'virtual int fcgi_streambuf::sync()':
fcgio.cpp:86:18: error: 'EOF' was not declared in this scope
     if (overflow(EOF)) return EOF;
                  ^
fcgio.cpp:87:41: error: 'EOF' was not declared in this scope
     if (FCGX_FFlush(this->fcgx)) return EOF;
                                         ^
```

fcgio.cpp: In member function 'virtual int fcgi_streambuf::underflow()':
fcgio.cpp:113:35: **error:** 'EOF' was not declared in this scope
if (glen <= 0) return EOF;
解决方法：在fcgio.h/fcgio.cpp 文件上添加#include <stdio.h>

2.2.4.2 测试程序

示例代码：vim fcgi.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "fcgi_stdio.h"

int main(int argc, char *argv[])
{
    int count = 0;

    //阻塞等待并监听某个端口，等待Nginx将数据发过来

    while (FCGI_Accept() >= 0)
    {
        //如果想得到数据，需要从stdin去读，实际上从Nginx上去读
        //如果想上传数据，需要往stdout写，实际上是给Nginx写数据

        printf("Content-type: text/html\r\n");
        printf("\r\n");
        printf("<title>Fast CGI Hello!</title>");
        printf("<h1>Fast CGI Hello!</h1>");
        //SERVER_NAME：得到server的host名称
        printf("Request number %d running on host <i>%s</i>\n",
               ++count, getenv("SERVER_NAME"));
    }

    return 0;
}
```

编译代码：

```
gcc fcgi.c -o test -lfcgi
```

test就是其中一个针对client一个http请求的业务服务应用程序。 需要在后台跑起来，并且让spawn负责管理。
记得先：ldconfig，否则spawn-fcgi启动异常

```
root@iZbp1d83xkvoja33dm7ki2Z:~/0voice/cloud-drive/test# ldconfig
root@iZbp1d83xkvoja33dm7ki2Z:~/0voice/cloud-drive/test# spawn-fcgi -a 127.0.0.1 -p 8001 -f ./test
spawn-fcgi: child spawned successfully: PID: 24314
```

查看8001是否有被监听

```
root@iZbp1h2l856zgoegc8rvnhZ:~/tuchuang# lsof -i:8001
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
test    3370 root    0u  IPv4 5624558      0t0  TCP localhost:8001 (LISTEN)
```

2.2.4.3 有关Nginx的fcgi的配置

#监听用户的test请求，通过fastcgi_pass交给本地8001端口处理
#此时spwan-cgi已经将8001端口交给之前我们写好的test进程处理

```
location /test {
          fastcgi_pass 127.0.0.1:8001;
          fastcgi_index test;
          include fastcgi.conf;
}
```

记得nginx先重新加载配置文件

```
/usr/local/nginx/sbin/nginx -s reload
```

当Nginx收到http://ip/test请求时，比如http://114.215.169.66/test，会匹配到location test块，将请求传到后端的FastCGI应用程序处理：