FastCGI接口协议 ¶

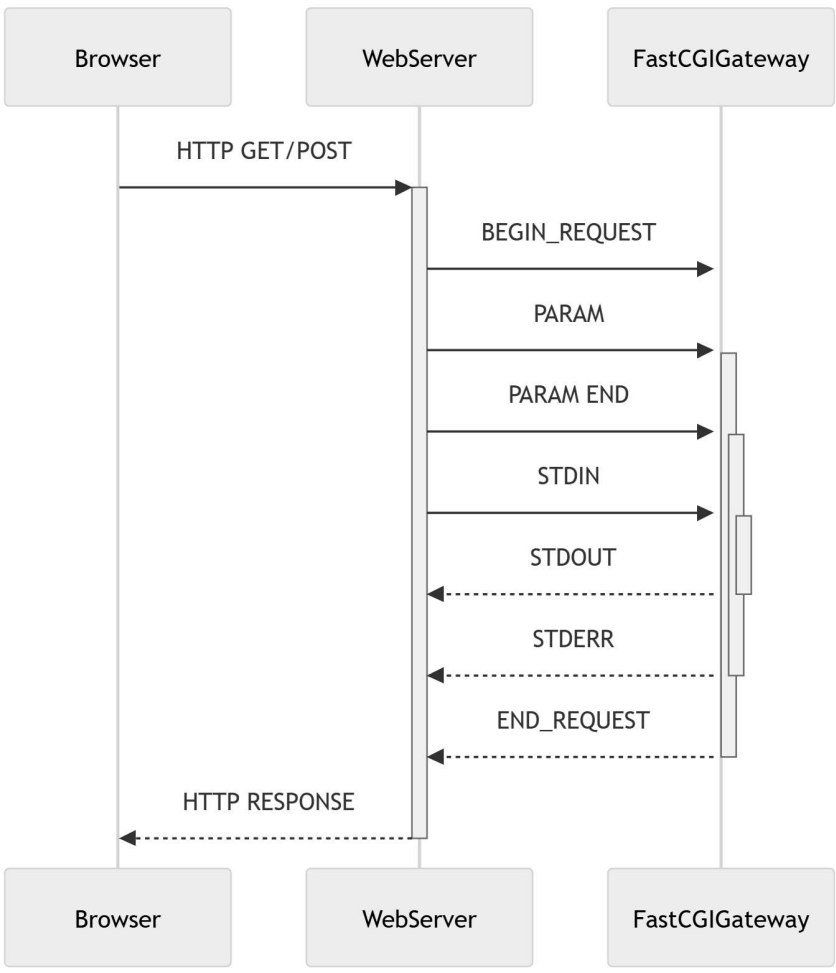
- CGI（公共网关接口）是Web服务器与应用程序之间的标准接口，与具体的编程语言无关
- CGI采用的形式是由Web服务器直接调用应用程序，参数写入环境变量
- FastCGI是在CGI基础上发展而来，也是Web服务器与应用程序之间的接口，不同的是，FastCGI网关是独立运行的服务端，通过Socket与Web服务器通信
- 协议规范参考：
  - https://fastcgi-archives.github.io/FastCGI_Specification.html (https://fastcgi-archives.github.io/FastCGI_Specification.html)
  - https://www.mit.edu/~yandros/doc/specs/fcgi-spec.html (https://www.mit.edu/~yandros/doc/specs/fcgi-spec.html)

网关程序

- Windows 系统安装好最新版的 PHP 后，php-cgi.exe 可作为 FastCGI 网关，运行 php 脚本程序：
  - 从 php.net 网站下载最新版的PHP软件包，解压缩到 c:\php 目录
  - 将 c:\php 目录加入到环境变量PATH中
  - 进入命令行，运行 php-cgi -b 9000 -c c:\php
- Linux 系统安装好最新版的 PHP 后，php-fpm 可作为 FastCGI 网关，运行 php 脚本程序：
  - 创建 php-fpm.conf (从 etc/php-fpm.conf.default 复制），视情况修改日志文件存放路径
  - 创建 www.conf (从 etc/php-fpm.d/www.conf.default 复制)
  - 运行: sudo php-fpm （默认是后台运行）
  - 通过 ps -lef | grep php-fpm 查看进程号，关闭时杀掉进程（kill 进程号）
- 视情况修改php.ini：
  - 错误显示方式：display_errors = stderr
    - 错误将输出到stderr，而不是输出到stdout，否则后台程序的错误将直接显示在网页上（不安全）
  - 修改配置文件后要重新启动网关程序

交互过程

1. Web 服务器向 FastCGI 网关发送一个 8 字节的消息头部（消息类型=开始请求）和消息体（开始请求），标志一个新的请求开始
2. Web 服务器向 FastCGI 网关发送一个 8 字节的消息头部（消息类型=参数）和消息体（参数），传递各类环境参数
3. 如果有多组参数，可以发送多个消息类型=参数的消息
4. Web 服务器向 FastCGI 网关发送一个 8 字节的消息头部（消息类型=参数，内容长度=0），表示参数发送完毕
5. 按照2、3、4步骤发送消息类型=标准输入的消息，消息体为网页提交的 POST 数据，传递页面输入的 POST 数据（可以发送多个消息）
6. FastCGI 网关处理参数和POST数据，生成响应（如果有错误，生成错误信息）
7. FastCGI 网关向 Web 服务器发送一个 8 字节的消息头部（消息类型=标准输出）和消息体（响应内容），传递应用程序的响应内容（可以发送多个消息）
8. 如果有错误信息，FastCGI网关向Web服务器发送一个8字节的消息头部（消息类型=标准错误）和消息体（错误信息），可以发送多个消息
9. FastCGI 网关向 Web 服务器发送一个 8 字节的消息头部（消息类型=结束请求，内容长度=0），标志此次请求结束



数据结构定义

消息头部

- 固定8个字节

In [ ]:
```
typedef struct {
    unsigned char version;              // 版本
    unsigned char type;                 // 消息类型
    unsigned char requestIdB1;          // 请求ID（2字节，高8位）
    unsigned char requestIdB0;          // 请求ID（低8位）
    unsigned char contentLengthB1;      // 内容长度字节数（2字节，高8位）
    unsigned char contentLengthB0;      // 内容长度的低8位
    unsigned char paddingLength;        // 填充长度
    unsigned char reserved;             // 保留
} FCGI_HEADER;
```

消息类型

In [ ]:
```
typedef enum {
    FCGI_BEGIN_REQUEST      =  1, // 开始请求
    FCGI_ABORT_REQUEST      =  2, // 中断请求
    FCGI_END_REQUEST        =  3, // 结束请求
    FCGI_PARAMS             =  4, // 参数（环境变量）
    FCGI_STDIN              =  5, // 标准输入（POST数据）
    FCGI_STDOUT             =  6, // 标准输出（响应）
    FCGI_STDERR             =  7, // 标准错误（错误信息）
    FCGI_DATA               =  8, // 数据
    FCGI_GET_VALUES         =  9, // GET值
    FCGI_GET_VALUES_RESULT  = 10  // GET值的结果
} FCGI_REQ_TYPE;
```

消息体

开始请求

In [ ]:
```
typedef struct {
    unsigned char roleB1;         // 角色（2字节，高8位）
    unsigned char roleB0;         // 角色（低8位）
    unsigned char flags;          // 标志（最低位为1时表示保持连接，其他位暂未使用）
    unsigned char reserved[5];    // 保留
} FCGI_BEGIN_REQ;
```

角色

- 指Web服务器请求时，希望FastCGI网关应承担什么角色

In [ ]:
```
typedef enum {
    FCGI_RESPONDER  = 1,          // 响应者（处理Web服务器的脚本页面）
    FCGI_AUTHORIZER = 2,          // 认证者
    FCGI_FILTER = 3               // 过滤者
} FCGI_ROLE;
```

结束请求

In [ ]:
```
typedef struct {
    unsigned char appStatusB3;      // 应用级别状态码（相当于应用程序exit的退出码，4字节，高8位）
    unsigned char appStatusB2;      // 次高8位
    unsigned char appStatusB1;      // 次低8位
    unsigned char appStatusB0;      // 低8位
    unsigned char protocolStatus;   // 协议级别状态码 参考：FCGI_STATUS
    unsigned char reserved[3];      // 保留
} FCGI_END_REQ;
```

协议级别状态码

In [ ]:
```
typedef enum {
    FCGI_REQUEST_COMPLETE   = 0,        // 请求的正常结束
    FCGI_CANT_MPX_CONN      = 1,        // 拒绝新请求。超出连接个数限制
    FCGI_OVERLOADED         = 2,        // 拒绝新请求。超出负载
    FCGI_UNKNOWN_ROLE       = 3         // 拒绝新请求。不能识别的角色
} FCGI_STATUS;
```

参数格式

- 总体上是多组 name 和 value 构成的对
- 每组先是 name 长度和 value 长度，然后是 name 内容和value 内容
- name 和 value 长度字段为 1 个或 4 个字节，具体为：
  - 若第1个字节 < 128，则长度字段为 1 个字节
  - 否则，长度字段为 4 个字节
- 内容字段均为字符串形式

用于Web Server的示例代码

- fastcgi.c 实现了 Web 服务器端向 FastCGI 网关发送请求，处理响应的基本功能
- 通过 fastcgi.h 声明了可供 Web 服务器调用的函数
- 具体为：
  - 协议字段定义详见 fcgidef.h
  - 函数原型声明详见 fastcgi.h
  - 具体协议的实现代码详见 fastcgi.c
  - 演示测试代码详见 fastCGI_Test.c

调用顺序：

```
// 1-2只调一次
1. s = fcgi_init_socket();                  // socket初始化
2. fcgi_connect(s, gatewayIP, gatewayPort);     // 建立与FastCGI网关的TCP连接

// 下面的3-7可调用多次
3. fcgi_begin_request(s, reqID);            // 发送开始请求消息
4. fcgi_params(s, reqID, params, paramNum);     // 发送参数消息（事先将参数数组组织好）
5. fcgi_param_end(s, reqID);                // 发送参数结束消息
6. fcgi_stdin(s, reqID, postData, dataLen);     // 发送POST数据
7. fcgi_getResp(s, &std_out, &out_len, &std_err, &err_len, &app_stat, &proto_stat);  // 接收响应并解析出标准输出和标准错误

// 处理完毕后关闭socket
8. close(s);                            // 关闭socket
```

说明

- 标准输出内容由头部和体部两部分构成，之间用一个空行隔开（连续2个回车换行）
  - 头部包含多个格式为“名称：值”的字段，每行一个，用回车换行结束
    - 头部字段包括：Status，Content-type（注意与标准HTTP不同，后面type是小写）
  - 返回给浏览器的只需要体部内容
  - 可以根据头部字段的内容决定是否原样返回体部内容，还是返回自定义状态码
- 标准错误的内容一般不返回给浏览器，只是在Web服务器输出错误日志（屏幕输出或写到文件）
  - 标准错误主要是当脚本程序存在语法错误或运行错误时出现
  - 当出现标准错误时，返回浏览器的HTTP状态码可以设为502

fastcgi.h

In [ ]:
```
typedef unsigned char u_char;
typedef unsigned short u_short;

// 名值对(Name Value Pair)
typedef struct {
    u_char name[255];
    u_char value[255];
} NV_PAIR;

// 异常退出
void fcgi_exitWithErr(const char err[]);

// 初始化
int fcgi_init_socket();

// 连接网关
int fcgi_connect(int s, char ip[], int port);

// 发送开始请求消息
int fcgi_begin_request(int sock, u_short reqID);

// 发送参数消息
int fcgi_params(int sock, u_short reqID, NV_PAIR params[], int num);

// 发送参数结束消息
int fcgi_param_end(int sock, u_short reqID);

// 发送标准输入消息
int fcgi_stdin(int sock, u_short reqID, u_char *data, size_t dataLen);

// 读取并解析响应消息
// 返回标准输出内容和字节数，标准错误内容和字节数，应用的状态，协议的状态
// app_stat=0，表示应用没有发生错误，否则发生了错误
// proto_stat=0，表示FCGI请求成功，否则表示拒绝
// 正常返回0，socket失败返回-1
int fcgi_getResp(int sock, u_char **std_out, size_t *out_len,
                           u_char **std_err, size_t *err_len,
                           int *app_stat, int *proto_stat);
```

fastCGI_Test.c

In [ ]:
```
#ifdef _WIN32
#include <winsock2.h>
#else
#include <sys/socket.h>
#endif
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fastcgi.h"

NV_PAIR params[100];     // 参数组
int num;                 // 参数个数

void addParam(const char *name, const char *value)
{
    strcpy(params[num].name, name);
    strcpy(params[num].value, value);
    num ++;
}
```

In [ ]:
```
int main(int argc, char *argv[]) {
    int sock = fcgi_init_socket();

    // 只能建一个连接，多次请求通过同一个连接发送
    if (fcgi_connect(sock, "127.0.0.1", 9000) < 0)
        fcgi_exitWithErr("Cannot connect to FastCGI Gateway");
    printf("FastCGI Gateway Connected\n");

    char post_data[2][100] = {"login=abcd&pass=1234", "login=1234&pass=abcd"};
    char query[2][100] = {"action=login", "action=logout"};
    u_short reqID = 0;

    int i;
    for (i=0; i<2; i++)  {
        char dataLen[10];
        printf("\n--- No.%d request ---\n", i);

        num = 0;
        addParam("REQUEST_METHOD","POST");                              // 提交方式
        addParam("SCRIPT_FILENAME", "c:/web/server/do.php");            // 执行的脚本文件路径（绝对路径）
        addParam("CONTENT_TYPE","application/x-www-form-urlencoded");  // 内容类型

        sprintf(dataLen, "%lu", strlen(post_data[i]));
        addParam("CONTENT_LENGTH", dataLen);        // 内容长度（POST数据的长度，字符串形式）

        addParam("QUERY_STRING", query[i]);   // 客户端以GET方式提交的参数串
        addParam("REMOTE_ADDR","127.0.0.1");        // 客户端地址（Web服务器根据socket连接信息获取）
        addParam("REMOTE_PORT", "1111");            // 客户端端口（Web服务器根据socket连接信息获取）

        // 以下为可选参数
        addParam("REQUEST_URI","/do.php");          // 请求URI
        addParam("SCRIPT_NAME","/do.php");          // 执行的脚本程序
        addParam("SERVER_PORT", "80");              // 服务器监听端口
        addParam("SERVER_ADDR", "127.0.0.1");       // 服务器地址
        addParam("DOCUMENT_ROOT","c:/web/server");  // 网站的根目录
        addParam("HTTP_CONTENT_LENGTH", dataLen);   // HTTP请求消息的体部长度（等于CONTENT_LENGTH）

        reqID ++;
        if (fcgi_begin_request(sock, reqID) < 0)
            fcgi_exitWithErr("Send begin Request failed");

        if (fcgi_params(sock, reqID, params, num) < 0)
            fcgi_exitWithErr("Send params failed");

        if (fcgi_param_end(sock, reqID) < 0)
            fcgi_exitWithErr("Send param end failed");

        if (fcgi_stdin(sock, reqID, (u_char *)post_data[i], strlen(post_data[i])) < 0)
            fcgi_exitWithErr("Send stdin failed");

        u_char *std_out, *std_err;
        size_t out_len, err_len;
        int app_stat, proto_stat;
        fcgi_getResp(sock, &std_out, &out_len, &std_err, &err_len, &app_stat, &proto_stat);

        if (proto_stat != 0) printf("Request deny by Gateway, stat=%d\n", proto_stat);
        if (app_stat != 0) printf("App not return zero: stat=%d\n", app_stat);

        if (out_len > 0) {
            printf("\n***std_out(len=%lu):\n%s\n", out_len, std_out);
        }
        if (err_len > 0) {
            printf("\n\n***std_err(len=%lu):\n%s\n", err_len, std_err);
        }
    }

    close(sock);
    return 0;
}
```