A76xx Series Open_SDK HTTP_应用指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_HTTP_应用指导
版本：	V1.00
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

Version History

Version	Date	Owner	What is new
V1.00	2022-11-18		第一版

About this Document

本文档适用于 A1803S open 系列、A1603 open 系列、A1606 open 系列。

目录

缩略语

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hyper Text Transport Protocol
MIME	Multipurpose Internet Mail Extensions
URL	Uniform Resource Locator
WWW	World Wide Web
IP	Internet Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
MQTT	Message Queuing Telemetry Transport

1HTTP 介绍

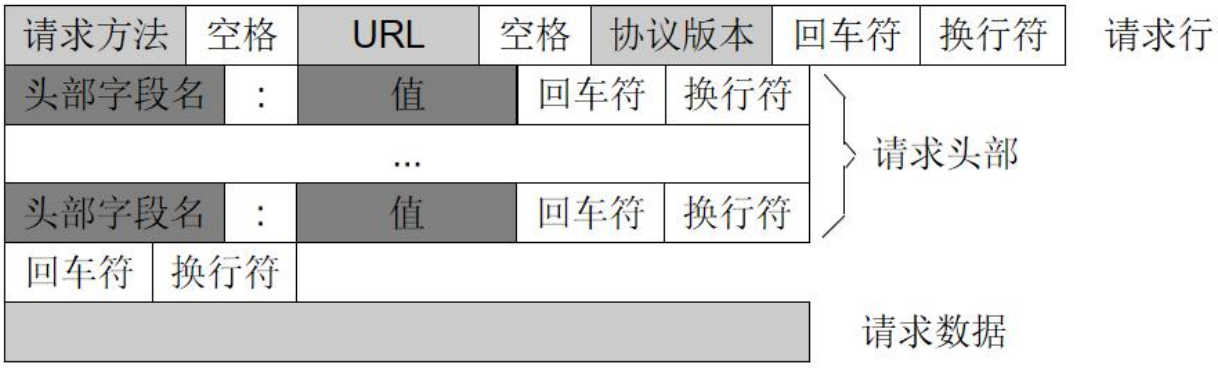
1.1 HTTP 传输过程

一次 HTTP 操作称为一个事务，其工作过程可分为四步：

1. 首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作就开始了。
2. 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可能的内容。
3. 服务器接到请求后，给予响应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务信息、实体信息和可能的内容。
4. 客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

1.2 HTTP 报文格式

1.2.1 Request 报文



1.2.2 Response 报文

1.3 HTTP 请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。
HTTP1.0 定义了三种请求方法：GET，POST 和 HEAD 方法。
HTTP1.1 新增了五种请求方法：OPTIONS，PUT，DELETE，TRACE 和 CONNECT 方法。

GET	请求指定的页面信息，并返回实体主体。
HEAD	类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头。
POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求中。POST 请求可能会导致新的资源的建立或已有资源的修改。
PUT	从客户端向服务器传送的数据取代指定的文档的内容。
DELETE	请求服务器删除指定的页面。
CONNECT	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
OPTIONS	允许客户端查看服务器的性能。
TRACE	回显服务器收到的请求，主要用于测试或诊断。

1.4 HTTP 状态码

状态代码有三位数字组成，第一个数字定义了响应的类别，共分为五种类别：

1××：指示信息—表示请求已接收，继续处理
2××：成功—表示请求已被成功接收、理解、接受
3××：重定向—要完成请求必须进行更进一步的操作
4××：客户端错误—请求有语法错误或请求无法实现
5××：服务器错误—服务器未能实现合法的请求

常见状态码：

200 OK	//客户端请求成功
400 Bad Request	//客户端请求有语法错误，不能被服务器理解
401 Unauthorized	//请求未经授权，这个状态代码必须和 www-authenticate 报头域一起使用
403 Forbidden	//服务器收到请求，但是拒绝提供服务
404 Not Found	//请求资源不存在，eg: 输入了错误的 URL
500 Internal Server Error	//服务器发生不可预期的错误
503 Server Unavailable	//服务器当前不能处理客户端的请求，一段时间后可能恢复正常

2API 介绍

2.1 sAPI_Httpinit

此接口通过激活 PDP 上下文启动 HTTP 服务

原型	SC_HTTP_RETURNCODE sAPI_HttpInit(*int channel*, sMsgQRef *magQ_urc*);
参数	[in] **channel**: 数据传输通道 0 - serial port 1 - usb [in] **magQ_urc**: 消息队列的标识，在整个 http 服务期间有效。
返回值	SC_HTTPS_SUCCESS:启动 http 服务成功 SC_HTTPS_FAIL:启动 http 服务失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"
NOTE	在本文当中所有 API 数据的接收方式： SIM_MSG_T msgQ_data_recv = {0, 0, -1, NULL}; sAPI_MsgQRecv(httpsUIResp_msgq, &msgQ_data_recv, SC_SUSPEND); SChttpApiTrans *sub_data = (SChttpApiTrans *)(msgQ_data_recv.arg3); char *buf= sub_data->data //具体的数据内容 结构体及变量定义见第五章 详细的 api 用法见 demo_https.c

2.2 sAPI_HttpTerm

此接口用于停止 HTTP 服务

原型	SC_HTTP_RETURNCODE sAPI_HttpTerm(void);
参数	无
返回值	SC_HTTPS_SUCCESS:停止 HTTP 服务成功 SC_HTTPS_FAIL:停止 HTTP 服务失败 参考第五章 SC_HTTP_RETURNCODE

头文件	#include "simcom_https.h"

2.3 sAPI_HttpPara

此接口用于设置 HTTP 参数

原型	SC_HTTP_RETURNCODE sAPI_HttpPara(char* command_type,char *parameters);
参数	[in] command_type:指定第二个参数的类型，该参数只能是："URL"、"CONNECTTO"、"RECVTO"、"CONTENT"、"ACCEPT"、"SSLCFG"、"USERDATA"、"READMODE" [in]与其命令参数匹配的值
返回值	SC_HTTPS_SUCCESS:设置 HTTP 参数成功 SC_HTTPS_FAIL:设置 HTTP 参数失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

第一个参数与第二个参数对应关系

"URL"	网络资源 URL：.字符串类型，必须以 "http://" 或者"https://"开头 a;)http://'server'/'path':'tcpPort'. b);https://'server'/'path':'tcpPort' "server": DNS 域名或者 IP 地址 "path":服务器上文件或目录的路径 "tcpPort": http 连接端口默认是 80 ，https 连接端口号默认是 443
"CONNECTTO"	访问服务器的超时时间，整数类型，超时时间范围是 20-120s，默认 120s.
"RECVTO"	从服务器接收数据的超时时间，整数类型，超时时间范围是 2s-120s, 默认是 20s.
"CONTENT"	这是用于 HTTP"Content-Type"标签，字符串类型，最大长度是 is 256, 默认值 "text/plain".
"ACCEPT"	这是用于"Accept-type" 标签，字符串类型，最大长度 256，默认值 "*/*".
"SSLCFG"	这是设置 SSL 上下文 id，整数类型，范围是 0-9. 默认值 0.
"USERDATA"	定制的 HTTP 报头信息，字符串类型，最大长度是 256.
"READMODE"	整数类型, i 可以被设置未 0 or 1. 如果设置未 1，可以从同一位置重复读取响应内容数据. 限制是 HTTP 服务器响应内容的大小应该小于 1M. 默认值是 0.

2.4 sAPI_HttpAction

此接口用于执行 HTTP 方法，你可以使用这个接口向 HTTP/HTTPS 服务器发送 get/post 请求

原型	SC_HTTP_RETURNCODE sAPI_HttpAction(int methods);
参数	[in] **methods**: 0: 执行 GET 方法 1: 执行 POST 方法 2: 执行 HEAD 方法 3: 执行 DELETE 方法
返回值	SC_HTTPS_SUCCESS:执行 HTTP 方法成功 SC_HTTPS_FAIL:执行 HTTP 方法失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

2.5 sAPI_HttpData

此应用程序接口用于保存 post 数据。

原型	SC_HTTP_RETURNCODE **sAPI_HttpData**(char *buffer, int len);
参数	[in] **buffer**: HTTP post 的数据 [in] **len**: 数据的长度
返回值	SC_HTTPS_SUCCESS:执行 HTTP 方法成功 SC_HTTPS_FAIL:执行 HTTP 方法失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

2.6 sAPI_HttpHead

此接口用于：当模块收到服务器的响应数据时，读取服务器响应的 HTTP 报头信息

原型	SC_HTTP_RETURNCODE **sAPI_HttpHead**(void);
参数	无
返回值	SC_HTTPS_SUCCESS:获取响应成功 SC_HTTPS_FAIL:获取响应头失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

2.7 sAPI_HttpRead

此接口用于从服务器读取数据。

原型	SC_HTTP_RETURNCODE sAPI_HttpRead(int commd_type, int start_offset, int byte_size);
参数	[in] **commd_type:** 0: 获取缓冲区保存数据的总大小，通过参数: byte_size 获取指定大小的数据 1: 设置读取数据的偏移量和大小 [in] **start_offset:**读取数据的开始位置 [in] **byte_size:** 读取数据的
返回值	SC_HTTPS_SUCCESS:获取响应头成功 SC_HTTPS_FAIL:获取响应头失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

2.8 sAPI_HttpPostfile

此接口用于将文件从客户机发送到服务器。

原型	SC_HTTP_RETURNCODE **sAPI_HttpPostfile**(char * filename, int dir);
参数	[in] **filename**: 需要发送的文件名 [in] **dir**: 文件所在的路径 1: C:/ 2: D:/
返回值	SC_HTTPS_SUCCESS:获取响应头成功 SC_HTTPS_FAIL:获取响应头失败 参考第五章 SC_HTTP_RETURNCODE
头文件	#include "simcom_https.h"

3 结果代码

3.1 HTTP 响应码的描述

状态码	描述
100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

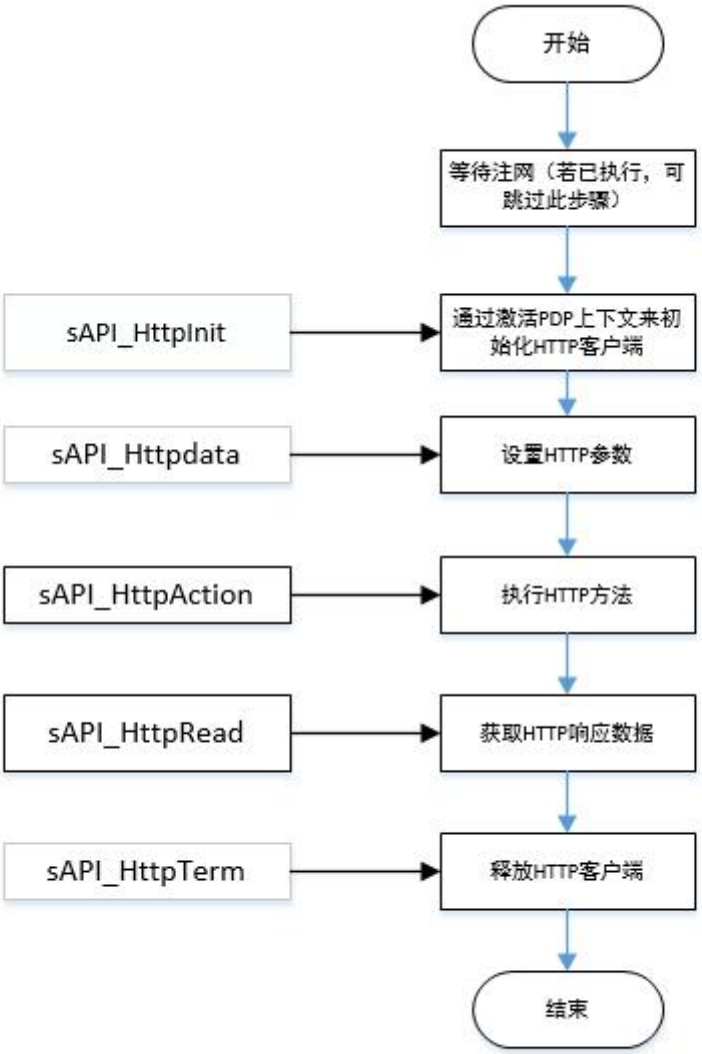
409	Conflict
410	Gone
411	Lenth Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
416	Requested range not satisfiable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway timeout
505	HTTP Version not supported
600	Not HTTP PDU
601	Network Error
602	No memory
603	DNS Error
604	Stack Busy

3.2 错误码的描述

<errcode>	Description
0	Success
701	Alert state
702	Unknown error
703	Busy
704	Connection closed error
705	Timeout
706	Receive/send socket data failed
707	File not exists or other memory error
708	Invalid parameter
709	Network error

710	start a new ssl session failed
711	Wrong state
712	Failed to create socket
713	Get DNS failed
714	Connect socket failed
715	Handshake failed
716	Close socket failed
717	No network error
718	Send data timeout
719	CA missed

4 使用流程


开始
等待注网（若已执行，可跳过此步骤）
sAPI_HttpInit
通过激活PDP上下文来初始化HTTP客户端
sAPI_Httpdata
设置HTTP参数
sAPI_HttpAction
执行HTTP方法
sAPI_HttpRead
获取HTTP响应数据
sAPI_HttpTerm
释放HTTP客户端
结束

5 变量定义

```
typedef struct
{
    INT32 status_code;   //请求消息的结果码
    INT32 method;        //请求方法
    INT32 action_content_len;     /*调用  sAPI_HttpAction and sAPI_HttpPostfile 后，接收的数据长度*/
    UINT8 *data;       //传输的数据
    INT32 dataLen;    //数据长度
} SChttpApiTrans;

typedef enum
{
    SC_HTTPS_SUCCESS,                    //成功
    SC_HTTPS_FAIL,                       //失败
    SC_HTTPS_SERVICE_NOT_AVAILABLE,      //HTTP 服务不可用
    SC_HTTPS_INVALID_PARAMETER,          //参数不正确
    SC_HTTPS_FILE_NOT_EXIST,             //文件不存在
    SC_HTTPS_WRITE_FILE_FAIL,            //写文件失败
    SC_HTTPS_READ_FILE_FAIL,             //读文件失败
    SC_HTTPS_DNS_PARSE_FAIL,             //域名解析失败
    SC_HTTPS_CONNECT_FAIL,               //连接失败
    SC_HTTPS_HANDSHAKE_FAILED,           //握手失败
    SC_HTTPS_TRANSFER_ERROR,        //发送或接收数据失败
    SC_HTTPS_ERROR_END
} SC_HTTP_RETURNCODE;


typedef struct sim_msg_cell
{
    UINT32 msg_id;          //消息 id
    int arg1;               //网络是否激活
    int arg2;               //网络是否可用
    void *arg3;             //接收数据指针
} SIM_MSG_T;
```

6 参考 demo

```
#include”simcom_https.h”
void HttpDemo(void)
{
    static sMsgQRef ghttps_msgq_ret;
    SC_HTTP_RETURNCODE ret;
    ret = sAPI_HttpInit(1,httpsUIResp_msgq);
    if(ret==SC_HTTP_FAIL)
    {
        PrintfResp("\r\nhttps init fail\r\n");
        return;
    }
    /*
     设置 HTTP 参数
     可根据需求设置相应的参数
      可设置的参数类型见 2.3 章节
    */

  char command_type[10]="URL";
  char parameters[20]="https://www.baidu.com";
  ret = sAPI_HttpPara(command_type, parameters);
  if(ret==SC_HTTP_FAIL)
  {
        PrintfResp("\r\nhttps set parameters fail\r\n");
        return;
  }

  char *buffer =”http post request”;
  ret = sAPI_HttpData(buffer, strlen(buffer);
  if(ret == SC_HTTPS_SUCCESS)
  {
        PrintfResp("\r\nhttps set data success\r\n");
  }
  else
  {
        PrintfResp("\r\nhttps set data fail\r\n");
  }

  /* action method
  0: GET
```

```
1: POST
2: HEAD
3: DELETE
*/
int method=0;
ret = sAPI_HttpAction(method);
if(ret==SC_HTTP_FAIL)
{
        PrintfResp("\r\nhttps action fail\r\n");
        return;
}

ret = sAPI_HttpHead();
SChttpApiTrans * httptransdata;
if(ret == SC_HTTPS_SUCCESS)
{
        sAPI_MsgQRecv(httpsUIResp_msgq, &msgQ_data_recv, SC_SUSPEND);
        httptransdata=(SChttpApiTrans *)(msgQ_data_recv.arg3);
}
else
{
         PrintfResp("\r\nhttps read head fail\r\n");
         return;
}

int commd_type = 1;
int start_offset = 0;
int byte_size = 100;
ret = sAPI_HttpRead(commd_type, start_offset, byte_size);
if(ret == SC_HTTP_SUCCESS);
{
      sAPI_MsgQRecv(httpsUIResp_msgq, &msgQ_data_recv, OS_SUSPEND);
      httptransdata    =(SChttpApiTrans *);msgQ_data_recv.arg3;
}
else (ret == SC_HTTP_FAIL);
{
      PrintfResp("\r\nhttps read fail\r\n");
}

 char filename = {0};
 int dir = 1;    //1 is C:/ ,2 is D:/
 strcpy(filename, "baidu_get.txt");
 ret = sAPI_HttpPostfile(filename, dir);
 if(ret == SC_HTTPS_SUCCESS)
 {
        sAPI_MsgQRecv(httpsUIResp_msgq, &msgQ_data_recv, OS_SUSPEND);
```

```
            httptransdata    =(SChttpApiTrans *);msgQ_data_recv.arg3;
        }
        else
        {
            PrintfResp("\r\nhttps post file fail\r\n");
        }

        ret = sAPI_HttpTerm();
        if(ret == SC_HTTPS_SUCCESS)
        {
            PrintfResp("\r\nhttp term success\r\n");
        }
        else
        {
            PrintfResp("\r\nhttp term fail\r\n");
        }
}
```