XR806 HTTPC 网络模块

开发指南

版本号：1.0

发布时间：2020-11-09

版本历史

版本	日期	责任人	版本描述
1.0	2020-11-09	AWA 1096	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 平台上 HTTPC 模块的使用方法。

1.2 目标读者

XR806 开发及维护人员

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
 **警告**	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
 **注意**	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
 **说明**	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
 **窍门**	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

2 概述

2.1 背景说明

HTTP(HyperText Transfer Protocol)即超文本传输协议，是一种用于分布式、协作式和超媒体信息系统的应用层协议。HTTPC 模块提供了实现 HTTP 客户端的应用接口。该模块基于 c 实现，可移植性高、支持 1.0./1.1、GET/POST、SSL、AUTH 等。SDK 中移植版本为 1.0，移植过程中裁剪了部分不需要的功能（代理），并对其提供的接口进行了再封装，使用户不用研究底层细节的实现，利用顶层的接口就可以实现客户端应用。

2.2 规格特性

HTTPC 模块提供了以下功能。

表 2-1 HTTPC 模块的规格特性

规格类型	支持规格	规格描述	备注
协议标准	HTTP 1.0	HTTP 1.0 协议标准	
	HTTP 1.1	HTTP 1.1 协议标准	
功能	GET	HTTP GET 功能，请求指定的页面信息，并返回实体主题	
	POST	HTTP POST 功能，向指定资源提交数据进行处理请求	
	SSL	HTTP 加密功能	
	AUTH	HTTP 用户验证功能	

2.3 文件位置

以 SDK 包为根目录，本网络协议涉及到的主要文件位置如下。

表 2-2 HTTPC 网络协议的文件位置

组件名	文件分类	文件位置
HTTPC	源码文件	sdk/src/net/HTTPClient
	头文件	sdk/include/net/HTTPClient
	示例工程	sdk/project/example/httpc

关键文件说明如下。

表 2-3 HTTPC 网络协议的文件说明

文件名	文件说明
HTTPCUsr_api.c	HTTPC 模块的 api 文件

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

3 应用说明

3.1 应用简述

HTTPC 网络协议已经内嵌到 XR806 SDK 中，通过配置以及函数接口调用即可使用。

3.2 配置说明

表 3-1 XR806 HTTPC 模块配置列表

配置项	配置说明
SSL 支持	设置说明： 此项配置 SSL 功能的使用与否 设置位置： /include/net/HTTPClient/API/HTTPClientWrapper.h 设置方式： 定义宏 HTTPC_SSL，如启动 SSL： #define HTTPC_SSL
调试功能启动/关闭	设置说明： 此项配置是否开启调试信息 设置位置： /include/net/HTTPClient/API/debug.h 设置方式： 定义宏_HTTP_DEBUGGING_和 HTTPCLIENT_DEBUG，如启动调试： #define _HTTP_DEBUGGING_ #define HTTPCLIENT_DEBUG

3.3 接口说明

表 3-2 HTTPC 模块接口简介

接口名	简要介绍
HTTPC_open 接口	本接口用于打开一个 HTTPC 连接请求，属于外部接口。
HTTPC_request 接口	本接口用于发起一个 HTTPC 连接请求，属于外部接口。
HTTPC_get_request_info 接口	本接口用于获取一个请求的连接信息，属于外部接口
HTTPC 读写接口	本接口用于往一个 HTTPC 连接写数据，属于外部接口
HTTPC_close 接口	本接口用于关闭一个 HTTPC 连接请求，属于外部接口
HTTPC_reset_session 接口	本接口用于重置一个 HTTPC 连接请求，属于外部接口

接口名	简要介绍
HTTPC 设置接口	本接口用于设置一个 HTTPC 连接请求，属于外部接口

3.3.1 HTTPC_open 接口

3.3.1.1 HTTPC_open

表 3-3 HTTPC_open 接口函数说明

信息项	说明
原型	int HTTPC_open(HTTPParameters *ClientParams);
功能	创建 httpc 会话句柄，并设置会话相关属性，申请相关资源
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明。
返回值	无

表 3-4 HTTPParameters 结构体的成员说明

成员项	说明
CHAR Uri[HTTP_CLIENT_MAX_URL_LENGTH];	含义解释：目标 url。 使用说明：赋值即可
HTTP_VERB HttpVerb;	含义解释：http 方式，配置 get 方式还是 post 方式。 使用说明：enum HTTP_VERB 类型，赋值即可
UINT32 Verbose;	含义解释：无，没有作用，赋值 0 即可。 使用说明：无
CHAR UserName[HTTP_CLIENT_MAX_USERNAME_LENGTH];	含义解释：登陆的用户名。 使用说明：赋值即可
CHAR Password[HTTP_CLIENT_MAX_PASSWORD_LENGTH];	含义解释：登陆的密码。 使用说明：赋值即可
HTTP_AUTH_SCHEMA AuthType;	含义解释：登陆的认证方式。 使用说明：enum HTTP_AUTH_SCHEMA 类型，赋值即可
BOOL isTransfer;	含义解释：该次请求数据正在传输中的标志位。 使用说明：HTTPC 内部使用变量，应用代码不需要使用该值

成员项	说明
HTTP_SESSION_HANDLE pHTTP;	含义解释：该次请求的会话指针。 使用说明：TTPC 内部使用变量，应用代码不需要使用该值
UINT32 Flags;	含义解释：发送请求的会话标志位。 使用说明：赋值即可
VOID *pData;	含义解释：post 方式时，post 的数据的指针。 使用说明：赋值即可
UINT32 pLength;	含义解释：数据长度。 使用说明：赋值即可
UINT32 nTimeout;	含义解释：超时时间，单位为 ms。 使用说明：赋值即可

3.3.2 HTTPC_request 接口

3.3.2.1 HTTPC_request

表 3-5 HTTPC_request 接口函数说明

信息项	说明
原型	int HTTPC_request(HTTPParameters *ClientParams, HTTP_CLIENT_GET_HEADER Callback);
功能	发起连接请求
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明 HTTP_CLIENT_GET_HEADER Callback 含义解释：用户自定义头部回调函数。 使用说明：该回调函数需要返回用户自定义的头部及头部值，格式需要满足”key1:valule1&key2:value2”。有时，用户需要发送特定的头部给服务器，此时则可通过设置 Callback 回调函数
返回值	0：成功 其它：失败

3.3.3 HTTPC_get_request_info 接口

3.3.3.1 HTTPC_get_request_info

表 3-6 HTTPC_get_request_info 接口函数说明

信息项	说明
原型	int HTTPC_get_request_info(HTTPParameters *ClientParams, void *HttpClient);
功能	获取连接参数或信息
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明 void *HttpClient 含义解释：用于存储连接参数的结构体指针。 使用说明：该参数为 HTTP_CLIENT 结构体类型指针。HTTP_CLIENT 结构体的成员说明，请参阅表 3-7 HTTP_CLIENT 结构体的成员说明
返回值	0：成功 其它：失败

表 3-7 HTTP_CLIENT 结构体的成员说明

成员项	说明
UINT32 HTTPStatusCode;	含义解释：HTTPC 的状态码。 使用说明：无
UINT32 RequestBodyLengthSent;	含义解释：发送的负载数据的总长度。 使用说明：enum HTTP_VERB 类型，赋值即可
UINT32 ResponseBodyLengthReceived;	含义解释：接收的负载数据的总长度。 使用说明：无
UINT32 TotalResponseBodyLength;	含义解释：头部里 content-length 的值。 使用说明：无
UINT32 HttpState;	含义解释：HTTPC 的内部状态值。 使用说明：无
HTTP_REDIRECT_PARAM *RedirectUrl;	含义解释：redirect url 转换后的新的 url。 使用说明：无
UINT32 HttpFlags;	含义解释：HTTPC 内部的 flags。

成员项	说明
	使用说明：无

3.3.4 HTTPC 读写接口

3.3.4.1 HTTPC_read

表 3-8 HTTPC_read 接口函数说明

信息项	说明
原型	int HTTPC_read(HTTPParameters *ClientParams, VOID *pBuffer, UINT32 toRead, UINT32 *recived);
功能	从服务器读取数据
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明 VOID *pBuffer 含义解释：数据 buffer。 使用说明：略 UINT32 toRead 含义解释：希望读取的数据长度。 使用说明：赋值即可 UINT32 *recived 含义解释：实际读取的数据长度。 使用说明：略
返回值	HTTP 状态码

3.3.4.2 HTTPC_write

表 3-9 HTTPC_write 接口函数说明

信息项	说明
原型	int HTTPC_write(HTTPParameters *ClientParams, VOID *pBuffer, UINT32 toWrite);
功能	向服务器写数据
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明 VOID *pBuffer 含义解释：数据 buffer。 使用说明：略

信息项	说明
	UINT32 toWrite 含义解释：要写的数据长度。 使用说明：赋值即可
返回值	实际写的字节数

3.3.5 HTTPC_close 接口

3.3.5.1 HTTPC_close

表 3-10 HTTPC_close 接口函数说明

信息项	说明
原型	int HTTPC_close(HTTPParameters *ClientParams);
功能	关闭当前连接，并释放相关资源
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明
返回值	0：成功 其它：失败

3.3.6 HTTPC_reset_session 接口

3.3.6.1 HTTPC_reset_session

表 3-11 HTTPC_reset_session 接口函数说明

信息项	说明
原型	int HTTPC_reset_session(HTTPParameters *ClientParams);
功能	重置上次会话的服务信息
参数	HTTPParameters *ClientParams 含义解释：HTTPParameters 结构体指针，此结构体存放 HTTPC 的配置信息。 使用说明：HTTPParameters 结构体的成员说明，请参阅表 3-4 HTTPParameters 结构体的成员说明
返回值	0：成功 其它：失败

3.3.7 HTTPC 设置接口

3.3.7.1 HTTPC_Register_user_certs

表 3-12 HTTPC_Register_user_certs 接口函数说明

信息项	说明
原型	void HTTPC_Register_user_certs(HTTPC_USR_CERTS certs);

信息项	说明
功能	安全访问下，使用该接口设置客户端证书参数
参数	HTTPC_USR_CERTS certs 含义解释：用户证书回调函数。 使用说明：返回证书，证书的格式要按照 tls 要求的客户端(security_client)执行。
返回值	无

3.3.7.2 HTTPC_set_ssl_verify_mode

表 3-13 HTTPC_set_ssl_verify_mode 接口函数说明

信息项	说明
原型	void HTTPC_set_ssl_verify_mode(unsigned char mode);
功能	安全访问下，使用该接口设置客户端验证服务器证书的方式
参数	unsigned char mode 含义解释：验证模式。 使用说明：取值范围为： 0：MBEDTLS_SSL_VERIFY_NONE 1：MBEDTLS_SSL_VERIFY_OPTIONAL 2：MBEDTLS_SSL_VERIFY_REQUIRED 3：MBEDTLS_SSL_VERIFY_UNSET
返回值	无

4 示例说明

HTTPC 接口用于进行一次 HTTP 连接，下面以 HTTP 连接为例，简要说明 HTTPC 接口的使用。

4.1 示例简介

HTTPC 示例的演示的目的是简要介绍 HTTPC 接口的基本使用方法，演示的功能为访问服务器下载文件。

4.1.1 获取方式

HJTTPC 示例有示例工程代码，位于 XR806 SDK 的/project/example/httpc 目录，以下此示例工程简称为 HTTPC 示例工程。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

4.1.2 准备工作

HTTPC 示例工程的硬件准备如下。

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 Uart0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。

HTTPC 的软件准备，包括烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》

4.1.3 操作步骤

完成烧写，复位后，通过串口命令连接网络即可。

表 4-1 HTTPC 示例工程的命令列表

命令	说明
快速配置网络	命令格式： net sta config <ssid> [psk] 命令解释： 快速配置被连接 AP 的 SSID 和密码 参数说明： ssid：被连接 AP 的 ssid psk：连连接 AP 的密码，如果为开放网络，则不需要此项。
启用网络	命令格式： net sta enable 命令解释：

命令	说明
	开始连接 AP，需要先完成网络配置

4.2 效果展示

在评估板中运行 HTTPC 示例程序后，在控制台中会打印出下载的文件，并最后打印下载完成，如下所示。

```
...
download complete, download_len 8192, time 5s
...
download complete, download_len 10980, time 11s
```

4.3 实现流程

HTTPC 示例工程展示了客户端 GET 功能。本节会介绍客户端 HTTP GET 和 HTTP POST 的代码实现流程。

4.3.1 客户端GET实现流程

第一步：安装客户端证书

```
//该步不是必须的，安全传输条件下，需要在发起请求之前设置证书
#ifdef HTTPC_CMD_REGISTER_USER_CALLBACK
    HTTPC_Register_user_certs(get_certs);
#endif
```

第二部：初始化并创建连接句柄

```
HTTPParameters *clientParams;
clientParams = malloc(sizeof(*clientParams));
clientParams->Uri = //赋值 uri
clientParams->HttpVerb = VerbGet;   //赋值方法

//下面三行为认证相关的设置
//clientParams->UserName
//clientParams->Password
//clientParams->AuthType = AuthSchemaDigest

if (HTTPC_open(clientParams) != 0) {
    CMD_ERR("http open err..\n");
}
```

第三步：发起连接请求

```
if ((ret = HTTPC_request(clientParams, NULL)) != 0) {
```

```
    CMD_ERR("http request err..\n");
    goto relese;
}
```

第四步：获取请求反馈信息

```
if (HTTPC_get_request_info(clientParams, &httpClient) != 0){
    CMD_ERR("http get request info err..\n");
}
```

第五步：读写数据

```
if ((ret = HTTPC_read(clientParams, buf, toReadLength, (void *)&Received)) != 0) {
    //CMD_DBG("get data,Received:%d\n",Received);
    if (ret == 1000) {
        ret = 0;
        CMD_DBG("The end..\n");
    } else
        CMD_ERR("Transfer err...ret:%d\n",ret);
    break;
} else {
    //CMD_DBG("get data,Received:%d\n",Received);
}
```

第六步：释放资源

```
HTTPC_close(clientParams);
```

4.3.2 客户端POST实现流程

第一步：初始化参数并创建句柄

```
HTTPParameters *clientParams;
clientParams = malloc(sizeof(*clientParams));
clientParams->Uri = uri //赋值 uri
clientParams->HttpVerb = VerbPost;

memcpy(buf, credentials, strlen(credentials));
clientParams->pData = buf; //post 的数据
clientParams->pLength = strlen(credentials); //post  数据长度

if ((ret = HTTPC_open(clientParams)) != 0) {
    CMD_ERR("http open err..\n");
    goto relese;
}
```

第二步：发起连接请求

```
if ((ret = HTTPC_request(clientParams, NULL)) != 0) {
        CMD_ERR("http request err..\n");
        goto relese;
}
```

第三步：获取连接请求反馈信息

```
if ((ret = HTTPC_get_request_info(clientParams, &httpClient)) != 0) {
        CMD_ERR("http get request info err..\n");
        goto relese;
}
```

第四步：读取数据

```
if ((ret = HTTPC_read(clientParams, buf, toReadLength, (void *)&Received)) != 0) {
        //CMD_DBG("get data,Received:%d\n",Received);
        if (ret == 1000) {
                ret = 0;
                CMD_DBG("The end..\n");
        } else
                CMD_ERR("Transfer err...ret:%d\n",ret);
        break;
} else {
        //CMD_DBG("get data,Received:%d\n",Received);
}
```

第五步：释放资源并关闭连接

```
HTTPC_close(clientParams);
```

著作权声明

商标声明

免责声明