WEBCLIENT 用户手册

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019

WWW.RT-THREAD.ORG

Friday 28th September, 2018

版本和修订

Date	Version	Author	Note
2013-05-05	v1.0.0	bernard	初始版本
2018-08-06	v2.0.0	chenyong	版本更新

目录

第 1 章

软件包介绍

WebClient 软件包是 RT-Thread 自主研发的，基于 HTTP 协议的客户端的实现，它提供设备与 HTTP Server 的通讯的基本功能。

1.1 软件包目录结构

WebClient 软件包目录结构如下所示：

```
webclient
+---docs
|   +---figures                     // 文档使用图片
|   |   api.md                      // API 使用说明
|   |   introduction.md             // 介绍文档
|   |   principle.md                // 实现原理
|   |   README.md                   // 文档结构说明
|   |   samples.md                  // 软件包示例
|   |   user-guide.md               // 使用说明
|   +---version.md                  // 版本
+---inc                             // 头文件
+---src                             // 源文件
+---samples                         // 示例代码
|   |   webclient_get_sample        // GET 请求示例代码
|   +---webclient_post_sample       // POST 请求示例代码
|   LICENSE                         // 软件包许可证
|   README.md                       // 软件包使用说明
+---SConscript                      // RT-Thread 默认的构建脚本
```

1.2 软件包功能特点

WebClient 软件包功能特点：

- 支持 IPV4/IPV6 地址

  WebClient 软件包会自动根据传入的 URI 地址的格式判断是 IPV4 地址或 IPV6 地址，并且从其中解析出连接服务器需要的信息，提高代码兼容性。

- 支持 GET/POST 请求方法

  HTTP 有多种请求方法（GET、POST、PUT、DELETE 等），目前 WebClient 软件包支持 GET 和 POST 请求方法，这也是嵌入式设备最常用到的两个命令类型，满足设备开发需求。

- 支持文件的上传和下载功能

  WebClient 软件包提供文件上传和下载的接口函数，方便用户直接通过 GET/POST 请求方法上传本地文件到服务器或者下载服务器文件到本地，文件操作需要文件系统支持，使用前需开启并完成文件系统的移植。

- 支持 HTTPS 加密传输

  HTTPS 协议（HyperText Transfer Protocol over Secure Socket Layer）和 HTTP 协议一样是基于 TCP 实现的，实际上是在原有的 HTTP 数据外部添加一层 TLS 加密的封装，从而达到加密传输数据的目的。HTTPS 协议地址区别于 HTTP 地址，是以 `https` 开头的。WebClient 软件包中的 TLS 加密方式依赖 mbedtls 软件包 实现。

- 完善的头部数据添加和处理方式

  HTTP 头部信息用于确定当前请求或响应的数据和状态信息，在发送 GET/POST 请求时头部的拼接成为用户操作的一大难题，正常的做法是手动逐行输入或使用字符串拼接方式，WebClient 软件包中提供简单的添加发送请求头部信息的方式，方便用户使用。对于请求返回的头部信息，往往用户需要获取头部字段数据，WebClient 软件包同样提供了 通过字段名获取字段数据的方式，方便获取需要的数据。

1.3 HTTP 协议介绍

1.3.1 HTTP 协议简述

HTTP （Hypertext Transfer Protocol）协议，即超文本传输协议，是互联网上应用最为广泛的一种网络协议，由于其简捷、快速的方式，适用于分布式和合作式超媒体信息系统。HTTP 协议是基于 TCP/IP 协议的网络应用层协议。默认端口为 80 端口。协议最新版本是 HTTP 2.0，目前是用最广泛的是 HTTP 1.1。

HTTP 协议是一种请求/响应式的协议。一个客户端与服务器建立连接之后，发送一个请求给服务器。服务器接收到请求之后，通过接收到的信息判断响应方式，并且给予客户端相应的响应，完成整个 HTTP 数据交互流程。

浏览器网页是 HTTP 的主要应用方式，但这不代表 HTTP 只能应用于网页，实际上只要通信的双方遵循 HTTP 协议数据传输合适就可以进行数据交互，比如嵌入式领域设备通过 HTTP 协议与服务器连接。

1.3.2 HTTP 协议特点

- 无状态协议

  HTTP 协议是无状态协议。无状态是指协议对于事件的处理没有记忆能力。这意味着如果后续处理需要前面的信息，则必须重传，这可能导致每次连接传送的数据量增大。这样的好处在于，在服务器不需要先前信息时它的应答就较快。

- 灵活的数据传输

  HTTP 允许传输任意类型的数据对象，传输的数据类型由 Content-Type 加以标记区分。

- 简单快速

  客户端向服务器发送请求时，只需要传送请求方式和路径。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

- 支持 B/S 和 C/S 模式

  C/S 结构，即 Client/Server (客户机/服务器) 结构。B/S 结构，即 Browser/Server (浏览器/服务器) 结构。

1.3.3 HTTP 协议请求信息 Request

客户端发送一个 HTTP 请求到服务器的请求消息包括：**请求行、请求头部、空行**和**请求数据**四个部分组成。

- 请求行：用于说明请求类型，用来说明请求类型, 要访问的资源以及所使用的 HTTP 版本；
- 请求头部：紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息；
- 空行：请求头部后面的空行是必须的，区分头部和请求数据；
- 请求数据：请求数据也叫主体，可以添加任意的其他数据。

如下图是一个 POST 请求的信息：

图 1.1: *HTTP 协议请求信息*

1.3.4 HTTP 协议响应信息 Response

一般情况下，服务器接收并处理客户端发过来的请求后会返回一个 HTTP 的响应消息。HTTP 响应也由四个部分组成，分别是：**状态行、消息报头、空行**和**响应数据**。

- 状态行：由 HTTP 协议号（HTTP 1.1），响应状态码（200），状态信息（OK）三部分组成；
- 消息报头：用来说明客户端要使用的一些附加信息（日期，数据长度等）；
- 空行：消息报头后面的空行是必须的，用于区分消息报头和响应数据；
- 响应数据：服务器返回给客户端的文本信息。

如下图是一个 POST 请求的响应的信息：

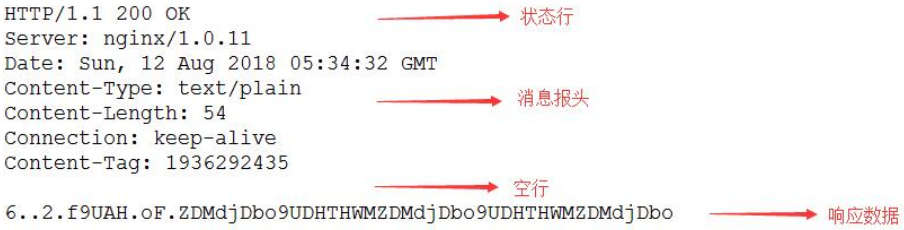

图 1.2: *HTTP 协议响应信息*

1.3.5 HTTP 协议状态码

HTTP 协议中通过返回的状态码信息，判断当前响应状态，WebClient 软件包中也有对状态的获取和判断方式，这里主要介绍常见的状态码的意义。

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

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

常见的状态码有以下几种：

```
200 OK                          //客户端请求成功
206 Partial Content             //服务器已经成功处理了部分 GET 请求
400 Bad Request                 //客户端请求有语法错误，不能被服务器所理解
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
```

第 2 章

示例程序

WebClient 软件包提供两个 HTTP Client 示例程序, 分别用于演示软件包支持的 GET 和 POST 功能，完成数据的上传与下载。

示例文件

示例程序路径	说明
samples/webclient_get_sample.c	GET 请求测试例程
samples/webclient_post_sample.c	POST 请求测试例程

2.1 准备工作

2.1.1 获取软件包

- menuconfig 配置获取软件包和示例代码

  打开 RT-Thread 提供的 ENV 工具，使用 **menuconfig** 配置软件包。

  启用 WebClient 软件包，并配置使能测试例程（Enable webclient GET/POST samples），如下所示：

```
RT-Thread online packages
    IoT - internet of things  --->
        [*] WebClient: A HTTP/HTTPS Client for RT-Thread
        [ ]   Enable support tls protocol
        [*]   Enable webclient GET/POST samples # 开启 WebClient 测试例程
              Version (latest)  --->            # 开启使用最新版本软件包
```

- 使用 `pkgs --update` 命令下载软件包
- 编译下载

2.2 启动例程

本例程使用的测试网站是 RT-Thread 系统的官方网站。GET 请求示例可以从网站中获取并打印显示文件内容；POST 请求示例可以上传数据到测试网站，测试网站会响应相同的数据。

> HTTP 收发数据包括头部数据和正文数据两部分，以下称头部数据为 `header 数据`，正文数据为 `body 数据`。

2.2.1 GET 请求示例

GET 请求示例流程：

- 创建 client 会话结构体
- client 发送 GET 请求 header 数据（使用默认 header 数据）
- server 响应 header 数据和 body 数据
- 打印 server 响应 body 数据
- GET 请求测试完成/失败

GET 请求示例使用方式有如下两种：

- 在 MSH 中使用命令 `web_get_test` 执行 GET 请求示例程序，可以获取并打印显示默认网址下载的文件信息，如下图 LOG 显示：

```
msh />web_get_test
webclient GET request response data :
RT-Thread is an open source IoT operating system from China, which has
    strong scalability: from a tiny kernel running on a tiny core, for
    example ARM Cortex-M0, or Cortex-M3/4/7, to a rich feature system
    running on MIPS32, ARM Cortex-A8, ARM Cortex-A9 DualCore etc.

msh />
```

- 在 MSH 中使用命令 `web_get_test [URI]` 格式命令执行 GET 请求示例程序，其中 URI 为用户自定义的支持 GET 请求的地址。

2.2.2 POST 请求示例

POST 请求示例流程如下：

- 创建 client 会话结构体
- 拼接 POST 请求需要的 header 数据
- client 发送拼接的 header 数据和 body 数据
- server 响应 header 数据和 body 数据
- 打印 server 响应 body 数据
- POST 请求测试完成/失败

POST 请求示例使用方式有如下两种：

- 在 MSH 中使用命令 `web_post_test` 执行 POST 请求示例程序，可以获取并打印显示响应数据（默认 POST 请求的地址是类似于回显的地址，会返回上传的数据），如下图 LOG 显示：

```
msh />web_post_test
webclient POST request response data :
RT-Thread is an open source IoT operating system from China!
msh />
```

- 在 MSH 中使用命令 `web_post_test [URI]` 格式命令执行 POST 请求示例程序，其中 URI 为用户自定义的支持 POST 请求的地址。

第 3 章

工作原理

WebClient 软件包主要用于在嵌入式设备上实现 HTTP 协议，软件包的主要工作原理基于 HTTP 协议实现，如下图所示：

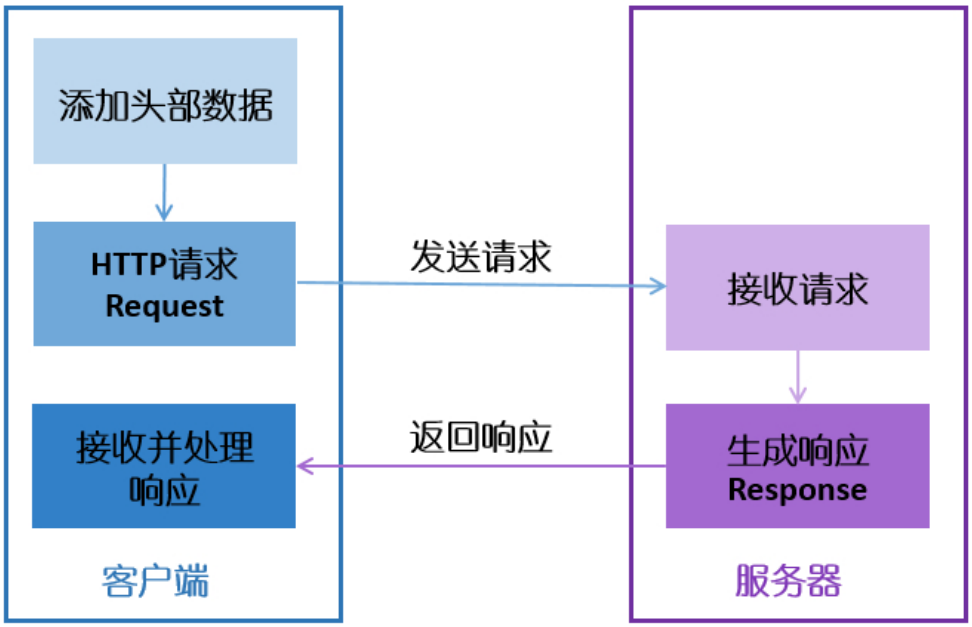

图 3.1: *WebClient 软件包工作原理*

HTTP 协议定义了客户端如何从服务器请求数据，以及服务器如何把数据传送给客户端的方式。HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

在 HTTP 协议的实际使用过程中，一般遵循以下流程：

1. 客户端连接到服务器

通常是通过 TCP 三次握手建立 TCP 连接，默认 HTTP 端口号为 80。

2. 客户端发送 HTTP 请求（GET/POST）

   通过 TCP 套接字，客户端向 Web 服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据四部分组成

3. 服务器接受请求并返回 HTTP 响应

   服务器解析请求，定位请求资源。服务器将需要发送的资源写到 TCP 套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据四部分组成。

4. 客户端和服务器断开连接

   若客户端和服务器之间连接模式为普通模式，则服务器主动关闭 TCP 连接，客户端被动关闭连接，释放 TCP 连接。若连接模式为 keepalive 模式，则该连接保持一段时间，在该时间内可以继续接收数据。

5. 客户端解析响应的数据内容

   客户端获取数据后应该先解析响应状态码，判断请求是否成功，然后逐行解析响应报头，获取响应数据信息，最后读取响应数据，完成整个 HTTP 数据收发流程。

第 4 章

使用指南

本节主要介绍 WebClient 软包的基本使用流程，并针对使用过程中经常涉及到的结构体和重要 API 进行简要说明。

4.1 准备工作

首先需要下载 WebClient 软件包，并将软件包加入到项目中。在 BSP 目录下使用 menuconfig 命令打开 env 配置界面，在 `RT-Thread online packages → IoT - internet of things` 中选择 WebClient 软件包，操作界面如下图所示：

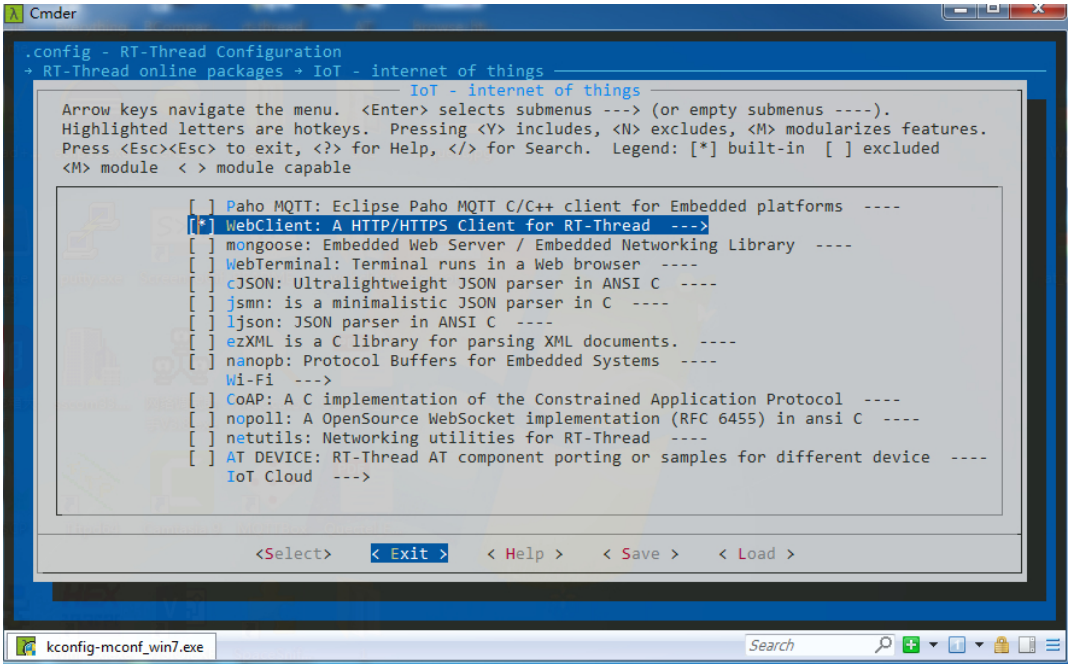

图 4.1: *WebClient 软件包配置*

详细配置介绍如下所示：

```
RT-Thread online packages
    IoT - internet of things  --->
        [*] WebClient: A HTTP/HTTPS Client for RT-Thread
        [ ]   Enable support tls protocol
        [ ]   Enable webclient GET/POST samples
              Version (latest)  --->
```

Enable support tls protocol：开启对 HTTPS 支持；

Enable webclient GET/POST samples：添加示例代码；

Version：配置软件包版本号。

选择合适的配置项后，使用 `pkgs --update` 命令下载软件包并更新用户配置。

4.2 使用流程

使用 WebClient 软件包发送 GET/POST 请求一般需要完成如下基本流程：

1. **创建客户端会话结构体**

```
struct  webclient_header
{
    char *buffer;                          //添加或者获取的头部数据
    size_t length;                         //存放当前头部数据长度

    size_t size;                           //存放最大支持的头部数据长度
};

struct webclient_session
{
    struct webclient_header *header;       //保存头部信息结构体
    int socket;                            //当前连接套接字
    int resp_status;                       //响应状态码

    char *host;                            //连接服务器地址
    char *req_url;                         //连接的请求地址

    int chunk_sz;                          //chunk 模式下一块数据大小
    int chunk_offset;                      //chunk 模式剩余数据大小

    int content_length;                    //当前接收数据长度（非 chunk 模
        式）
```

```
    size_t content_remainder;           //当前剩余接收数据长度

#ifdef WEBCLIENT_USING_TLS
    MbedTLSSession *tls_session;        // HTTPS 协议相关会话结构体
#endif
};
```

webclient_session 结构体用于存放当前建立的 HTTP 连接的部分信息，可用与 HTTP 数据交互整个流程。建立 HTTP 连接前需要创建并初始化该结构体，创建的方式示例如下：

```
struct webclient_session *session = RT_NULL;

/* create webclient session and set header response size */
session = webclient_session_create(1024);
if (session == RT_NULL)
{
    ret = -RT_ENOMEM;
    goto __exit;
}
```

2. **拼接头部数据**

WebClient 软件包提供两种请求头部发送方式：

- 默认头部数据

  如果要使用默认的头部信息，则不需要拼接任何头部数据，可直接调用 GET 发送命令。默认头部数据一般只用于 GET 请求。

- 自定义头部数据

自定义头部数据使用 webclient_header_fields_add 函数添加头部信息，添加的头部信息位于客户端会话结构体中，在发送 GET/POST 请求时发送。

添加示例代码如下：

```
/* 拼接头部信息 */
webclient_header_fields_add(session, "Content-Length: %d\r\n", strlen(
    post_data));

webclient_header_fields_add(session, "Content-Type: application/octet-
    stream\r\n");
```

3. **发送 GET/POST 请求**

头部信息添加完成之后，就可以调用 `webclient_get` 函数或者 `webclient_post` 函数发送 GET/POST 请求命令了，函数中主要操作如下：

- 通过传入的 URI 获取信息，建立 TCP 连接；
- 发送默认或者拼接的头部信息；
- 接收并解析响应数据的头部信息；
- 返回错误或者响应状态码。

发送 GET 请求示例代码如下：

```
int resp_status = 0;

/* send GET request by default header */
if ((resp_status = webclient_get(session, URI)) != 200)
{
    LOG_E("webclient GET request failed, response(%d) error.",
        resp_status);
    ret = -RT_ERROR;
    goto __exit;
}
```

4. **接收响应的数据**

发送 GET/POST 请求之后，可以使用 `webclient_read` 函数接收响应的实际数据。因为响应的实际数据可能比较长，所以往往我们需要循环接收响应数据，指导数据接收完毕。

如下所示为循环接收并打印响应数据方式：

```
int content_pos = 0;
/* 获取接收的响应数据长度 */
int content_length = webclient_content_length_get(session);

/* 循环接收响应数据直到数据接收完毕 */
do
{
    bytes_read = webclient_read(session, buffer, 1024);
    if (bytes_read <= 0)
    {
        break;
```

```
    }

    /* 打印响应数据 */
    for (index = 0; index < bytes_read; index++)
    {
        rt_kprintf("%c", buffer[index]);
    }

    content_pos += bytes_read;
} while (content_pos < content_length);
```

5. 关闭并释放客户端会话结构体

请求发送并接收完成之后，需要使用 webclient_close 函数关闭并释放客户端会话结构体，完成整个 HTTP 数据交互流程。

使用方式如下：

```
if (session)
{
    webclient_close(session);
}
```

4.3 使用方式

WenClient 软件包对于 GET/POST 请求，分别提供了几种不同的使用方式，用于不同的情况。

4.3.1 GET 请求方式

- 使用默认头部发送 GET 请求

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

if(webclient_get(session, URI) != 200)
{
    LOG_E("error!");
}
```

```
while(1)
{
    webclient_read(session, buffer, bfsz);
    ...
}

webclient_close(session);
```

- 使用自定义头部发送 GET 请求

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

webclient_header_fields_add(session, "User-Agent: RT-Thread HTTP Agent\r\
    n");

if(webclient_get(session, URI) != 200)
{
    LOG_E("error!");
}

while(1)
{
    webclient_read(session, buffer, bfsz);
    ...
}

webclient_close(session);
```

- 发送获取部分数据的 GET 请求（多用于断点续传）

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

if(webclient_get_position(URI, 100) != 206)
{
    LOG_E("error!");
}
```

```
while(1)
{
    webclient_read(session, buffer, bfsz);
    ...
}

webclient_close(session);
```

- 使用 webclient_response 接收 GET 数据

  多用于接收数据长度较小的 GET 请求。

```
struct webclient_session *session = NULL;
char *result;

session = webclient_create(1024);

if(webclient_get(session, URI) != 200)
{
    LOG_E("error!");
}

webclient_response(session, &result);

web_free(result);
webclient_close(session);
```

- 使用 webclient_request 函数发送并接收 GET 请求

  多用于接收数据长度较小，且头部信息已经拼接给出的 GET 请求。

```
char *result;

webclient_request(URI, header, NULL, &result);

web_free(result);
```

4.3.2 POST 请求方式

- 分段数据 POST 请求

  多用于上传数据量较大的 POST 请求，如：上传文件到服务器。

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

/*  拼接必要的头部信息 */
webclient_header_fields_add(session, "Content-Length: %d\r\n",
    post_data_sz);
webclient_header_fields_add(session, "Content-Type: application/octet-
    stream\r\n");

/* 分段数据上传 webclient_post 第三个传输上传数据为 NULL，改为下面循环上
    传数据*/
if( webclient_post(session, URI, NULL) != 200)
{
    LOG_E("error!");
}

while(1)
{
    webclient_write(session, post_data, 1024);
    ...
}

if( webclient_handle_response(session) != 200)
{
    LOG_E("error!");
}

webclient_close(session);
```

- 整段数据 POST 请求

  多用于上传数据量较小的 POST 请求。

```
char *post_data = "abcdefg";

session = webclient_create(1024);

/*  拼接必要的头部信息 */
webclient_header_fields_add(session, "Content-Length: %d\r\n", strlen(
    post_data));
webclient_header_fields_add(session, "Content-Type: application/octet-
    stream\r\n");
```

```
if(webclient_post(session, URI, post_data) != 200);
{
    LOG_E("error!");
}
webclient_close(session);
```

- 使用 webclient_request 函数发送 POST 请求

  多用于上传文件较小且头头部信息已经拼接给出的 POST 请求。

```
char *post_data = "abcdefg";
char *header = "xxx";

webclient_request(URI, header, post_data, NULL);
```

4.4 常见问题

4.4.1 HTTPS 地址不支持

```
[E/WEB]not support https connect, please enable webclient https configure
    !
```

- 原因：使用 HTTPS 地址但是没有开启 HTTPS 支持。
- 解决方法：在 WebClient 软件包 menuconfig 配置选项中开启 Enable support tls protocol 选项支持。

4.4.2 头部数据长度超出

```
[E/WEB]not enough header buffer size(xxx)!
```

- 原因：添加的头部数据长度超过了最大支持的头部数据长度。
- 解决方法：在创建客户端会话结构体的时候，增大传入的最大支持的头部数据长度。

第 5 章

API 说明

5.1 创建会话

struct webclient_session *webclient_session_create(size_t header_sz);

创建客户端会话结构体。

参数	描述
header_sz	最大支持的头部长度
返回	**描述**
!= NULL	webclient 会话结构体指针
= NULL	创建失败

5.2 关闭会话连接

int webclient_close(struct webclient_session *session);

关闭传入的客户端会话连接，并释放内存。

参数	描述
session	当前连接会话结构体指针
返回	**描述**
=0	成功

5.3 发送 GET 请求

int webclient_get(struct webclient_session *session, const char* URI);

发送 HTTP GET 请求命令。

参数	描述
session	当前连接会话结构体指针
URI	连接的 HTTP 服务器地址
返回	**描述**
>0	HTTP 响应状态码
<0	发送请求失败

5.4 发送获取部分数据的 GET 请求

int webclient_get_position(struct webclient_session *session, const char* URI, int position);

发送带有 Range 头信息的 HTTP GET 请求命令，多用于完成断点续传功能。

参数	描述
session	当前连接会话结构体指针
URI	连接的 HTTP 服务器地址
position	数据偏移量
返回	**描述**
>0	HTTP 响应状态码
<0	发送请求失败

5.5 发送 POST 请求

int webclient_post(struct webclient_session *session, const char* URI, const char *post_data);

发送 HTTP POST 请求命令，上传数据到 HTTP 服务器。

参数	描述
session	当前连接会话结构体指针
URI	连接的 HTTP 服务器地址
post_data	需要上传的数据地址
返回	**描述**
>0	HTTP 响应状态码
<0	发送请求失败

5.6 发送数据

int webclient_write(struct webclient_session *session, const unsigned char* buffer, size_t size);

发送数据到连接的服务器。

参数	描述
session	当前连接会话结构体指针
buffer	发送数据的地址
size	发送数据的长度
返回	**描述**
>0	成功发送数据的长度
=0	连接关闭
<0	发送数据失败

5.7 接收数据

int webclient_read(struct webclient_session *session, unsigned char* buffer, size_t size);

从连接的服务器接收数据。

参数	描述
session	当前连接会话结构体指针
buffer	接收数据的存放地址

参数	描述
size	最大接收数据的长度
返回	**描述**
>0	成功接收数据的长度
=0	连接关闭
<0	接收数据失败

5.8 设置接收和发送数据超时时间

int webclient_set_timeout(struct webclient_session *session, int millisecond);

设置连接的接收和发送数据超时时间。

参数	描述
session	当前连接会话结构体指针
millisecond	设置的超时时间，单位毫秒
返回	**描述**
=0	设置超时成功

5.9 在请求头中添加字段数据

int webclient_header_fields_add(struct webclient_session *session, const char* fmt, ...);

该函数用于创建会话之后和发送 GET 或 POST 请求之前，用于添加请求头字段数据。

参数	描述
session	当前连接会话结构体指针
fmt	添加字段数据的表达式
...	添加的字段数据，为可变参数
返回	**描述**
>0	成功添加的字段数据的长度
<=0	添加失败或者头部数据长度超出

5.10 通过字段名获取字段值数据

const char *webclient_header_fields_get(struct webclient_session* session, const char *fields);

该函数用于发送 GET 或 POST 请求之后，可以通过传入的字段名称获取对应的字段数据。

参数	描述
session	当前连接会话结构体指针
fields	HTTP 字段名称
返回	**描述**
= NULL	获取数据失败
!= NULL	成功获取的字段数据

5.11 接收响应数据到指定地址

int webclient_response(struct webclient_session *session, unsigned char **response);

该函数用于发送 GET 或 POST 请求之后，可以接收响应数据到指定地址。

参数	描述
session	当前连接会话结构体指针
response	存放接收数据的字符串地址
返回	**描述**
>0	成功接收数据的长度
<=0	接收数据失败

5.12 发送 GET/POST 请求并接收响应数据

int webclient_request(const char *URI, const char* header, const char *post_data, unsigned char **response);

参数	描述
URI	连接的 HTTP 服务器地址
header	需要发送的头部数据
	= NULL，发送默认头数据信息，只用于发送 GET 请求
	!= NULL，发送指定头数据信息，可用于发送 GET/POST 请求
post_data	发送到服务器的数据
	= NULL，该发送请求为 GET 请求
	!= NULL，该发送请求为 POST 请求
response	存放接收数据的字符串地址
返回	**描述**
>0	成功接收数据的长度
<=0	接收数据失败

5.13 获取 HTTP 响应状态码

int webclient_resp_status_get(struct webclient_session *session);

该函数用于发送 GET 或 POST 请求之后，用于获取返回的响应状态码。

参数	描述
session	当前连接会话结构体指针
返回	**描述**
>0	HTTP 响应状态码

5.14 获取 Content-Length 字段数据

int webclient_content_length_get(struct webclient_session *session);

该函数用于发送 GET 或 POST 请求之后，用于获取返回的 Content-Length 字段数据。

参数	描述
session	当前连接会话结构体指针

参数	描述
返回	**描述**
>0	Content-Length 字段数据
<0	获取失败

5.15 下载文件到本地

int webclient_get_file(const char *URI, const char* filename);

从 HTTP 服务器下载文件并存放到本地。

参数	描述
URI	连接的 HTTP 服务器地址
filename	存放文件位置、名称
返回	**描述**
=0	下载文件成功
<0	下载文件失败

5.16 上传文件到服务器

int webclient_post_file(const char *URI, const char* filename, const char *form_data);

从 HTTP 服务器下载文件并存放到本地。

参数	描述
URI	连接的 HTTP 服务器地址
filename	需要上传的文件位置、名称
form_data	附加选项
返回	**描述**
=0	上传文件成功
<0	上传文件失败