WEBCLIENT User Manual

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2013-05-05	v1.0.0	bernard	initial version
2018-08-06	v2.0.0	chenyong	new version update

Table of contents

Chapter 1

Software Package Introduction

The WebClient software package is independently developed by RT-Thread and is based on the implementation of the HTTP protocol client.

Provides basic functions for communication between the device and HTTP Server.

1.1 Software package directory structure

The directory structure of the WebClient package is as follows:

```
webclient
+---docs
 ÿ +---figures ÿ ÿ api.md                          // Document using images
 ÿ ÿ introduction.md ÿ                             // API usage instructions
 ÿ principle.md ÿ ÿ README.md ÿ ÿ                  // Introduction document
 samples.md ÿ ÿ user-guide.md                      // Implementation principle
 ÿ +---version.md                                  // Document structure description
                                                   // Package Example
                                                   // Instructions for use
                                                   // Version
+---inc                                            // head File
+---src                                            // Source File
+---samples ÿ ÿ                                    // Example code
 webclient_get_sample ÿ +---                       // GET request example code
 webclient_post_sample ÿ ÿ README.md +---          // POST request example code
           LICENSE                                 // Package License
 SConscript                                        // Software package instructions
                                                   //RT-Thread default build script
```

1.2 Software Package Features

WebClient software package features:

• Support IPV4/IPV6 addresses

The WebClient software package will automatically determine whether the incoming URI address is an IPV4 address or an IPV6 address based on the format of the address, and parse out the information needed to connect to the server, thereby improving code compatibility.

• Support GET/POST request methods

HTTP has multiple request methods (GET, POST, PUT, DELETE, etc.). Currently, the WebClient software package supports GET and POST request methods, which are also the two most commonly used command types for embedded devices, meeting the needs of device development.

• Support file upload and download functions

The WebClient package provides file upload and download interface functions, which allows users to directly upload local files to the server or download server files to the local through the GET/POST request method. File operations require file system support, so the file system needs to be enabled and transplanted before use.

• Support HTTPS encrypted transmission

HTTPS protocol (HyperText Transfer Protocol over Secure Socket Layer) is implemented based on TCP like HTTP protocol. It actually adds a layer of TLS encryption encapsulation outside the original HTTP data to achieve the purpose of encrypted transmission data. HTTPS protocol address is different from HTTP address and starts with https . The TLS encryption method in WebClient software package depends on mbedtls software package accomplish.

• Complete header data adding and processing methods

HTTP header information is used to determine the data and status information of the current request or response. When sending GET/ POST requests, header splicing becomes a major problem for users. The normal practice is to manually enter line by line or use string splicing. The WebClient software package provides a simple way to add and send request header information for user convenience. For the header information returned by the request, users often need to obtain the header field data. The WebClient software package also provides a way to obtain field data by field name, which is convenient for obtaining the required data.

1.3 Introduction to HTTP Protocol

1.3.1 HTTP protocol overview

HTTP (Hypertext Transfer Protocol) is the most widely used network protocol on the Internet. Due to its simple and fast method, it is suitable for distributed and collaborative hypermedia information systems.

HTTP protocol is a network application layer protocol based on TCP/IP protocol. The default port is port 80. The latest version of the protocol is HTTP 2.0, and the most widely used one is HTTP 1.1.

The HTTP protocol is a request/response protocol. After a client establishes a connection with a server, it sends a request to the server. After receiving the request, the server determines the response method based on the received information and gives the client a corresponding response, completing the entire HTTP data interaction process.

Browser web pages are the main application of HTTP, but this does not mean that HTTP can only be applied to web pages. In fact, as long as the two communicating parties follow the HTTP protocol and the data transmission is appropriate, data interaction can be carried out. For example, embedded devices connect to the server through the HTTP protocol.

1.3.2 HTTP Protocol Features

• Stateless protocol

HTTP is a stateless protocol. Stateless means that the protocol has no memory for event processing. This means that if the subsequent processing requires previous information, it must be retransmitted, which may increase the amount of data transmitted per connection. The advantage of this is that the server responds faster when it does not need previous information.

• Flexible data transfer

HTTP allows the transmission of any type of data object, and the type of data transmitted is marked and distinguished by Content-Type.

• Simple and fast

When the client sends a request to the server, it only needs to transmit the request method and path. Since the HTTP protocol is simple, the program size of the HTTP server is small, so the communication speed is very fast.

• Support B/S and C/S mode

C/S structure, namely Client/Server structure. B/S structure, namely Browser/Server structure.

1.3.3 HTTP protocol request information Request

The request message sent by the client to the server via HTTP consists of four parts: request line, request header, blank line and request data.

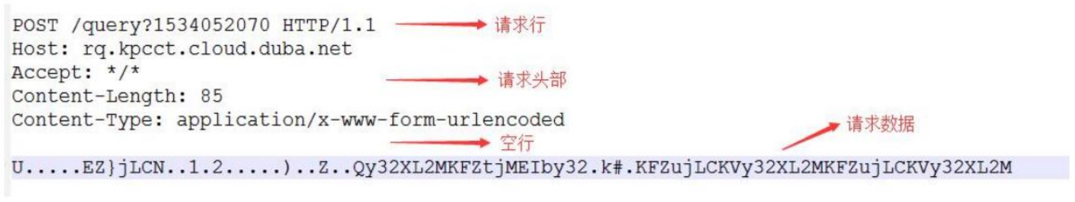
• Request line: used to describe the request type, the resource to be accessed, and the HTTP Version;

• Request header: The part immediately following the request line (i.e. the first line) that specifies additional information to be used by the server. interest;

• Blank line: The blank line after the request header is required to distinguish the header from the request data;

• Request data: Request data is also called body, and any other data can be added.

The following figure shows the information of a POST request:



```
POST /query?1534052070 HTTP/1.1
Host: rq.kpcct.cloud.duba.net
Accept: */*
Content-Length: 85
Content-Type: application/x-www-form-urlencoded

U.....EZ}jLCN..1.2.....)..Z..Qy32XL2MKFZtjMEIby32.k#.KFZujLCKVy32XL2MKFZujLCKVy32XL2M
```

Figure 1.1: *HTTP* Protocol Request Information

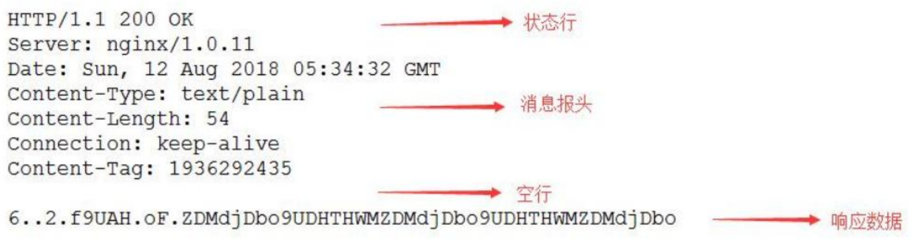
1.3.4 HTTP protocol response information Response

Generally speaking, after receiving and processing the request sent by the client, the server will return an HTTP response message.

HTTP response also consists of four parts: status line, message header, blank line and response data.

- Status line: consists of HTTP protocol number (HTTP 1.1), response status code (200), and status information (OK) composition;
- Message header: used to indicate some additional information to be used by the client (date, data length, etc.);
- Blank line: The blank line after the message header is required and is used to distinguish the message header from the response data;
- Response data: text information returned by the server to the client.

The following figure shows the response information of a POST request:



```
HTTP/1.1 200 OK
Server: nginx/1.0.11
Date: Sun, 12 Aug 2018 05:34:32 GMT
Content-Type: text/plain
Content-Length: 54
Connection: keep-alive
Content-Tag: 1936292435

6..2.f9UAH.oF.ZDMdjDbo9UDHTHWMZDMdjDbo9UDHTHWMZDMdjDbo
```

Figure 1.2: *HTTP* Protocol response information

1.3.5 HTTP protocol status code

The HTTP protocol uses the returned status code information to determine the current response status. The WebClient software package also has How to obtain and judge the status. Here we mainly introduce the meaning of common status codes.

The status code consists of three digits. The first digit defines the category of the response. There are five categories:

- 1xx: Informational – indicates the request has been received and is being processed
- 2xx: Success – indicates the request was successfully received, understood, accepted
- 3xx: Redirect – Further action is required to complete the request
- 4xx: Client Error – The request had a syntax error or could not be fulfilled
- 5xx: Server Error – The server failed to fulfill a valid request

The following are common status codes:

```
200 OK                              //Client request successful
206 Partial Content                 //The server has successfully processed some GET requests
400 Bad Request                     //The client request has a syntax error and cannot be understood by the server
403 Forbidden                       //The server received the request, but refused to provide service
404 Not Found                       //The requested resource does not exist, e.g., an incorrect URL was entered
500 Internal Server Error           //An unexpected error occurred on the server
503 Server Unavailable              //The server cannot currently process the client's request
```

chapter 2

Sample Program

The WebClient package provides two HTTP Client sample programs, which are used to demonstrate the GET and

And POST functions to complete the upload and download of data.

Sample Files

Sample program path	illustrate
samples/webclient_get_sample.c	GET request test routine
samples/webclient_post_sample.c	POST request test routine

2.1 Preparation

2.1.1 Obtaining the Software Package

- menuconfig configuration to obtain software packages and sample code

Open the ENV tool provided by RT-Thread and use **menuconfig** to configure the software package.

Enable the WebClient package and configure the Enable webclient GET/POST samples as shown below:

```
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 # Enable WebClient test routines # Enable the use of the
              Version (latest) --->                                latest version of the software package
```

• Use the pkgs --update command to download the software package •

Compile and download

2.2 Startup routine

The test website used in this example is the official website of the RT-Thread system. The GET request example can obtain and print the file content from the website; the POST request example can upload data to the test website, and the test website will respond with the same data.

HTTP sent and received data consists of two parts: header data and body data. Hereinafter, header data is referred to as header data and body data is referred to as body data.

2.2.1 GET request example

GET request example flow:

• Create a client session structure •
Client sends GET request header data (using default header data) • Server responds with
header data and body data • Print server response body data
• GET request test completed/failed

There are two ways to use the GET request example:

• Use the web_get_test command in MSH to execute the GET request sample program to obtain and print the default
The file information downloaded from the confirmed URL is shown in the LOG below:

```
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 />
```

• Use the web_get_test [URI] format command in MSH to execute the GET request sample program, where
URI is a user-defined address that supports GET requests.

2.2.2 POST request example

The example POST request process is as follows:

• Create a client session structure •

Concatenate the header data required for the POST

request • The client sends the concatenated header data and body

data • The server responds with the header data and body

data • Print the server response body data •

POST request test completed/failed

There are two ways to use the POST request example:

• Use the command web_post_test in MSH to execute the POST request sample program, and you can get and print the response data (the default POST request address is similar to the echo address, which will return the uploaded data), as shown in the LOG display below:

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

• Use the web_post_test [URI] format command in MSH to execute the POST request sample program, where The URI is a user-defined address that supports POST requests.

Chapter 3

working principle

The WebClient package is mainly used to implement the HTTP protocol on embedded devices. The main working principle of the package is

It is implemented based on HTTP protocol, as shown in the following figure:

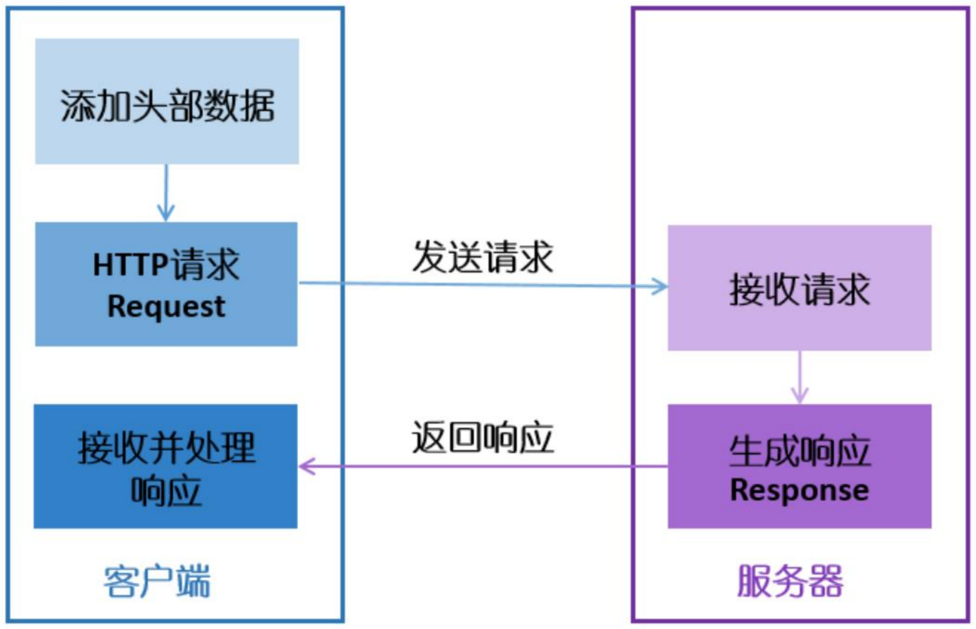


Figure **3.1:** *WebClient* How the software package works

The HTTP protocol defines how the client requests data from the server and how the server transmits data to the client. The HTTP protocol uses a request/response model. The client sends a request message to the server, which contains the request method, URL, protocol version, request header, and request data. The server responds with a status line, which includes the protocol version, success or error code, server information, response header, and response data.

In the actual use of the HTTP protocol, the following process is generally followed:

1. Client connects to server

A TCP connection is usually established through a TCP three-way handshake, and the default HTTP port number is 80.

2. The client sends an HTTP request (GET/POST)

Through the TCP socket, the client sends a text request message to the Web server. A request message consists of four parts: request line, request header, blank line and request data.

3. The server accepts the request and returns an HTTP response

The server parses the request and locates the requested resource. The server writes the resource to be sent to the TCP socket, which is read by the client. A response consists of four parts: status line, response header, blank line, and response data.

4. The client and server are disconnected

If the connection mode between the client and the server is normal mode, the server actively closes the TCP connection, and the client passively closes the connection and releases the TCP connection. If the connection mode is keepalive mode, the connection is maintained for a period of time, during which data can continue to be received.

5. The client parses the response data content

After obtaining the data, the client should first parse the response status code to determine whether the request is successful, then parse the response header line by line, obtain the response data information, and finally read the response data to complete the entire HTTP data sending and receiving process.

Chapter 4

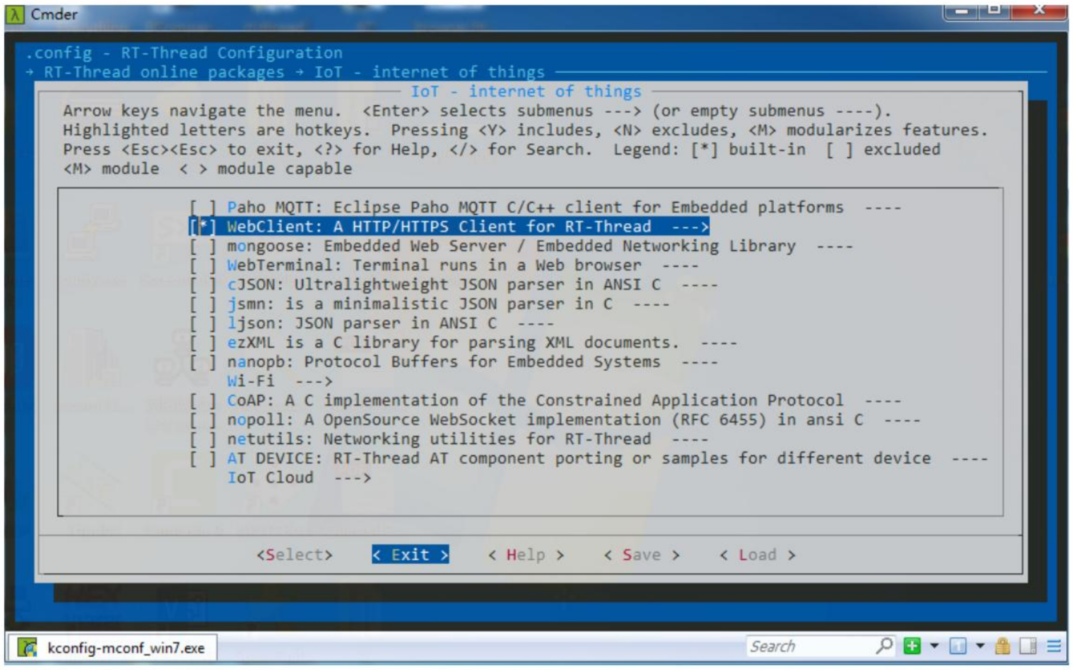
user's guidance

This section mainly introduces the basic usage process of the WebClient software package, and focuses on the structures that are often involved in the use process.

A brief description of the main body and important APIs.

4.1 Preparation

First, you need to download the WebClient software package and add it to the project. In the BSP directory, use the menuconfig command to open the env configuration interface, and select the WebClient software package in RT-Thread online packages ÿ IoT - internet of things . The operation interface is shown in the figure below:



```
.config - RT-Thread Configuration
→ RT-Thread online packages → IoT - internet of things ─────────────────────
                    IoT - internet of things
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

        [ ] Paho MQTT: Eclipse Paho MQTT C/C++ client for Embedded platforms  ----
        [*] WebClient: A HTTP/HTTPS Client for RT-Thread  --->
        [ ] mongoose: Embedded Web Server / Embedded Networking Library  ----
        [ ] WebTerminal: Terminal runs in a Web browser  ----
        [ ] cJSON: Ultralightweight JSON parser in ANSI C  ----
        [ ] jsmn: is a minimalistic JSON parser in C  ----
        [ ] ljson: JSON parser in ANSI C  ----
        [ ] ezXML is a C library for parsing XML documents.  ----
        [ ] nanopb: Protocol Buffers for Embedded Systems  ----
            Wi-Fi  --->
        [ ] CoAP: A C implementation of the Constrained Application Protocol  ----
        [ ] nopoll: A OpenSource WebSocket implementation (RFC 6455) in ansi C  ----
        [ ] netutils: Networking utilities for RT-Thread  ----
        [ ] AT DEVICE: RT-Thread AT component porting or samples for different device  ----
            IoT Cloud  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 4.1: *WebClient* Package Configuration

The detailed configuration is as follows:

```
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 : Enable support for HTTPS;

Enable webclient GET/POST samples : add sample code;

Version : Configure the package version number.

After selecting the appropriate configuration item, use the pkgs --update command to download the software package and update the user configuration.

4.2 Usage Process

Using the WebClient software package to send a GET/POST request generally requires the following basic process:

1. Create a client session structure

```
struct webclient_header
{
    char *buffer;                          //Add or get header data
    size_t length;                         //Store the current header data length

    size_t size;                           //Store the maximum supported header data length
};

struct webclient_session
{
    struct webclient_header *header; //Save header information structure
    int socket; int                        //Current connection socket
    resp_status;                           //Response status code

    char *host;                            //Connect to server address
    char *req_url;                         //Connection request address

    int chunk_sz; int                      //chunk mode, the size of a block of data
    chunk_offset;                          //chunk mode remaining data size

    int content_length ;                   //Current received data length (non-chunk mode)
```

```
        size_t content_remainder;                    //The current remaining received data length

#ifdef WEBCLIENT_USING_TLS
        MbedTLSSession *tls_session;                 // HTTPS protocol related session structure
#endif
};
```

The webclient_session structure is used to store some information of the currently established HTTP connection, and can be used to interact with the entire HTTP data process. Before establishing an HTTP connection, you need to create and initialize this structure. The creation method is as follows:

```
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. Splice head data

The WebClient package provides two ways to send request headers:

• Default header data

If you want to use the default header information, you do not need to concatenate any header data, and can directly call the GET command to send it. The default header data is generally only used for GET requests.

• Custom header data

Custom header data uses the webclient_header_fields_add function to add header information. The added header information is located in the client session structure and is sent when sending a GET/POST request.

Add the following sample code:

```
/* Splicing head information */
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. Send **GET/POST** request

After the header information is added, you can call the webclient_get function or the webclient_post function to send a GET/POST request command. The main operations in the function are as follows:

- Get information through the incoming URI and establish a TCP connection;
- Sending default or concatenated header information;
- Receive and parse the header information of the response data;
- Return an error or response status code.

The sample code for sending a GET request is as follows:

```
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. Receive response data

After sending a GET/POST request, you can use the webclient_read function to receive the actual response data. Because the actual response data may be long, we often need to loop to receive the response data until the data is received.

The following is a method for looping to receive and print response data:

```
int content_pos = 0; /* Get
the length of the received response data*/
int content_length = webclient_content_length_get(session);

/* Loop to receive response data until all data is received*/
do
{
        bytes_read = webclient_read(session, buffer, 1024); if (bytes_read <= 0) {


                break;
```

```
    }

    /* Print response data */
    for (index = 0; index < bytes_read; index++) {

        rt_kprintf("%c", buffer[index]);
    }

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

5. Close and release the client session structure

After the request is sent and received, you need to use the webclient_close function to close and release the client session structure to complete the entire HTTP data interaction process.

Here's how to use it:

```
if (session) {

    webclient_close(session);
}
```

4.3 Usage

The WenClient package provides several different ways to use GET/POST requests for different situations.

4.3.1 GET request method

• Send a GET request using default headers

```
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);
```

• Send GET request with custom headers

```
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);
```

• Send a GET request to obtain partial data (mostly used for breakpoint resuming)

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

if(webclient_get_position(URI, 100) != 206) {

        LOG_E("error!");
}
```

Machine Translated by Google

```
while(1) {

        webclient_read(session, buffer, bfsz);
        ...
}

webclient_close(session);
```

• Using webclient_response to receive GET data is mostly used to receive

GET requests with a small data length.

```
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);
```

• Use the webclient_request function to send and receive GET requests

It is mostly used to receive GET requests with small data length and concatenated header information.

```
char *result;

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

web_free(result);
```

4.3.2 POST request method

• Multipart data POST request

It is mostly used for POST requests that upload large amounts of data, such as uploading files to a server.

```
struct webclient_session *session = NULL;

session = webclient_create(1024);

/* Splice necessary header information */
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");

/* Upload data in segments webclient_post. The third transmission upload data is NULL. Change to the following loop
        Transfer data*/
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);
```

• POST request for the entire data segment

It is mostly used for POST requests that upload small amounts of data.

```
char *post_data = "abcdefg";

session = webclient_create(1024);

/* Splice necessary header information */
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);
```

• Use the webclient_request function to send a POST request

It is mostly used for POST requests that upload small files and have header information concatenated.

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

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

4.4 Frequently Asked Questions

4.4.1 HTTPS address is not supported

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

• Cause: Using an HTTPS address but not enabling HTTPS support.

• Solution: Enable the Enable support tls protocol option in the menuconfig configuration options of the WebClient package.

4.4.2 Header data length exceeds

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

• Cause: The length of the added header data exceeds the maximum supported header data length.

• Solution: When creating the client session structure, increase the maximum supported header data length passed in.

Chapter 5

API Description

5.1 Create a session

```
struct webclient_session *webclient_session_create(size_t header_sz);
```

Create a client session structure.

parameter	describe
header_sz	Maximum supported head length
return	describe
!= NULL	webclient session structure pointer
= NULL	Creation failed

5.2 Close the session connection

```
int webclient_close(struct webclient_session *session);
```

Closes the incoming client session connection and frees memory.

parameter	describe
session	Pointer to the current connection session structure
return	describe
=0	success

5.3 Sending a GET request

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

Send an HTTP GET request command.

parameter	describe
session	Pointer to the current connection session structure
URI	HTTP server address to connect to
return	describe
>0	HTTP response status codes
<0	Failed to send request

5.4 Send a GET request to obtain partial data

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

Send an HTTP GET request command with Range header information, which is mostly used to complete the breakpoint resume function.

parameter	describe
session	Pointer to the current connection session structure
URI	HTTP server address to connect to
position	Data offset
return	describe
>0	HTTP response status codes
<0	Failed to send request

5.5 Sending a POST request

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

Send HTTP POST request command to upload data to HTTP server.

parameter	describe
session	Pointer to the current connection session structure
URI	HTTP server address to connect to
post_data	The address of the data to be uploaded
return	describe
>0	HTTP response status codes
<0	Failed to send request

5.6 Sending Data

int webclient_write(struct webclient_session *session, const unsigned char* buffer,

size_t size);

Sends data to the connected server.

parameter	describe
session	Pointer to the current connection session structure
buffer	The address to send data to
size	The length of the data sent
return	describe
>0	The length of the successfully sent data
=0	Connection closed
<0	Failed to send data

5.7 Receiving Data

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

size);

Receive data from a connected server.

parameter	describe
session	Pointer to the current connection session structure
buffer	The storage address of received data

parameter	describe
size	Maximum length of received data
return	describe
>0	The length of the successfully received data
=0	Connection closed
<0	Failed to receive data

5.8 Set the timeout for receiving and sending data

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

Set the timeout for receiving and sending data for the connection.

parameter	describe
session	Pointer to the current connection session structure
millisecond	Set the timeout in milliseconds
return	describe
=0	Set timeout successfully

5.9 Add field data to the request header

```
int webclient_header_fields_add(struct webclient_session session, const char

fmt, …);
```

This function is used to add request header field data after creating a session and before sending a GET or POST request.

parameter	describe
session	Pointer to the current connection session structure
fmt	Add an expression to the field data
…	The added field data is a variable parameter
return	describe
>0	The length of the field data successfully added
<=0	Adding failed or the header data length exceeds

5.10 Get field value data by field name

const char *webclient_header_fields_get(struct webclient_session* session, const

char *fields);

This function is used to send a GET or POST request, and can get the corresponding field number through the passed field name.

according to.

parameter	describe
session	Pointer to the current connection session structure
fields	HTTP Field Name
return	describe
= NULL	Failed to obtain data
!= NULL	Successfully obtained field data

5.11 Receive response data to the specified address

int webclient_response(struct webclient_session *session, unsigned char **re-sponse);

This function is used to receive response data to the specified address after sending a GET or POST request.

parameter	describe
session	Pointer to the current connection session structure
response	The string address where the received data is stored
return	describe
>0	The length of the successfully received data
<=0	Failed to receive data

5.12 Send **GET/POST** request and receive response data

int webclient_request(const char *URI, const char* header, const char *post_data,

unsigned char **response);

parameter	describe
URI	HTTP server address to connect to
header	Header data to be sent
	= NULL, send default header data information, only used to send GET requests
	!= NULL, send the specified header data information, which can be used to send GET/ POST request
post_data	Data sent to the server
	= NULL, the request is a GET request
	!= NULL, the request is a POST request
response	The string address where the received data is stored
return	describe
>0	The length of the successfully received data
<=0	Failed to receive data

5.13 Get HTTP response status code

```
int webclient_resp_status_get(struct webclient_session *session);
```

This function is used to get the returned response status code after sending a GET or POST request.

parameter	describe
session	Pointer to the current connection session structure
return	describe
>0	HTTP response status codes

5.14 Get Content-Length field data

```
int webclient_content_length_get(struct webclient_session *session);
```

This function is used to obtain the returned Content-Length field data after sending a GET or POST request.

parameter	describe
session	Pointer to the current connection session structure

parameter	describe
return	describe
>0	Content-Length field data
<0	Failed to obtain

5.15 Download the file to your local computer

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

Download files from the HTTP server and store them locally.

parameter	describe
URI	HTTP server address to connect to
filename	Storage file location and name
return	describe
=0	Download file successfully
<0	Failed to download the file

5.16 Upload files to the server

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

Download files from the HTTP server and store them locally.

parameter	describe
URI	HTTP server address to connect to
filename	The location and name of the file to be uploaded
form_data	Additional options
return	describe
=0	Upload file successfully
<0	Failed to upload file