RT-THREAD WEBNET User Manual

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Wednesday 7th November, 2018

Versions and Revisions

Date	Version	Author	Note
2013-05-05	v1.0.0	bernard	Initial version
2018-08-06	v2.0.0	chenyong	Version Update

Table of contents

Chapter 1

Software Package Introduction

The WebNet software package is independently developed by RT-Thread and is based on the HTTP protocol Web server implementation. It not only provides

It not only provides the basic functions of HTTP Client communication, but also supports multiple module function extensions to meet the functional requirements of developers for embedded device servers.

1.1 Software Package Directory Structure

The directory structure of the WebClient package is as follows:

name	illustrate
docs	Document Directory
inc	Header file directory
src	Source file directory
module	Function module file directory
samples	Sample file directory
LICENSE	License File
README.md	Software package instructions
SConscript	RT-Thread default build script

1.2 Software Package Features

The WebNet software package provides a wide range of functional support. Most of the functions are provided in the form of modules. Each functional module provides a separate

The control mode and different functional modules do not interfere with each other. The modules and the main program cooperate to form a set of complete functions, high performance, high concurrency and

A highly customizable web server framework.

WebNet software package features:

• Support **HTTP 1.0/1.1**

The WebNet software package is used to run the network server function in the embedded device. Its function is based on the HTTP protocol and supports HTTP 1.0 and HTTP 1.1 protocols to improve web server compatibility.

• Support **CGI** function

The Common Gateway Interface (CGI) function is a runtime specification for external programs that can be used to extend server functionality. CGI facilitates interaction between browsers and servers. The CGI function in the WebNet software package allows users to register custom **CGI** execution functions. The server determines the CGI type requested by the browser, executes the corresponding function, and provides appropriate feedback.

• Support **ASP** variable replacement function

ASP (Active Server Pages) implements variable replacement within web pages. When a browser accesses a page with a **<% %>** tag (supporting page files with the .asp suffix), it is automatically replaced with a custom, registered **ASP** execution function and provided with appropriate feedback. This feature makes it easy to independently modify the implementation of each variable execution function, modifying different parts of the entire page and facilitating collaboration between front-end and back-end development.

• Support **AUTH** basic authentication function

During communication using the HTTP protocol, the HTTP protocol defines the Basic Authentication process, which allows HTTP servers to authenticate web browsers. Basic authentication provides a simple, fast, and accurate method for verifying user identities, making it suitable for systems and devices that require high server security. The Basic Authentication feature in the WebNet software package is designed to be linked to directories in URLs, enabling permissions to be assigned based on those directories.

• Support **INDEX** directory file display function

After the WebNet server is started, enter the corresponding device IP address and directory name in the browser to access the directory and list the The information of all files in the directory is similar to a simple file server and can be used to query and download server files.

• Support **ALIAS** alias access function

ALIAS alias access function can set multiple aliases for folders, which can be used to simplify operations on long paths and facilitate user access to resources.

• Support **SSI** file embedding function

The SSI (Server Side Include) feature is typically used to execute server-side programs or insert text content into web pages. WebNet's SSI feature currently only supports inserting text content into web pages. When a file or tag is present in the page code (supporting page files with .shtml and .shtm suffixes), it will automatically parse the information into the corresponding file. By embedding SSI files, the entire page processing can be modularized, allowing modifications to the page to be completed by modifying a specific file.

• Support file upload function

Supports uploading files from browser to WebNet server, and can customize file processing function to store uploaded files in file system or directly write to the target memory. This function can be used to achieve:

1. Upload resource files to the file system, such as images and web pages;

2. Upload the memory image to burn the memory in the system, such as SPI FLASH;
3. Upload the configuration file to the system to update the system settings;
4. Upload the firmware and write it directly to FLASH to implement the firmware update function.

• Support pre-compression function

The pre-compression function is used to compress web resource files in advance (supports compressed files with the .gz suffix), reducing the demand for storage space and compressing Shorten the file system reading time, reduce network traffic, and increase the speed of web page loading. Generally speaking, after using the pre-compression function, the WebNet server The storage space occupied is only 30%~40% of the original.

• Support cache function

The WebNet cache mechanism can determine whether the browser requested file has been modified, and thus decide whether to send the complete file content to the browser. The WebNet package buffering mechanism is divided into three levels as follows:

1. Level 0: Disables the cache function, meaning the browser reads the entire file content from WebNet each time;
2. Level 1: Enable the cache function. WebNet reads the last modified time of the requested file and returns the last modified time if it is the same as the local file. Return 304 to inform the browser that the file has not been updated and will not send the file;
3. Level 2: Enable the cache function. Based on the original judgment of modification time, add cache file validity time support and operation validity time The browser can access the file again.

• Support breakpoint resume function

WebNet server supports breakpoint resuming function, that is, if the client encounters an error in downloading the file, it only needs to provide The offset of the previous file download on the WebNet server. The server will send the file content to the client from the specified file offset, and resume the download. This function can ensure that files are uploaded quickly and accurately, and improve server operation efficiency.

1.3 Software Package Performance Testing

Test environment: AM1808, main frequency 400M, DDR2 memory

Test environment	Page file is on SD card, DM9000AEP external network card
Static page request	142 pages/s
CGI page request	429 pages/s

Test environment	Page file on SD card, EMAC external network card
Static page request	190 pages/s
CGI page request	567 pages/s

1.4 Software Package Resource Usage

ROMÿ16 Kbytes

RAM: 1.5 Kbytes + 1.1 Kbytes x number of connections

NOTE: The above data is for reference only. The example platform is Cortex-M3. Due to the different instruction lengths and frameworks, the software package resources are occupied.

When WebNet processes pages, some resources are consumed in the file system and modules.

Chapter 2

Sample Program

The WebNet package provides a comprehensive sample page to demonstrate the various features of the package, including: AUTH, CGI, ASP,

This chapter mainly introduces how to use the examples of each functional module in the WebNet software package.

Sample File

Sample program path	illustrate
samples/wn_sample.c	Comprehensive sample code
samples/wn_sample_upload.c	Upload file sample code
samples/index.html	Comprehensive example page
samples/index.shtml	SSI functionality example page
samples/version.asp	ASP Function Sample Page

2.1 Preparation

2.1.1 Software Package Acquisition

• 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 WebNet package and configure the Enable **webnet** samples configuration as shown below:

```
RT-Thread online packages
    IoT - internet of things --->
        [*] WebNet: A HTTP Server for RT-Thread
            (80) Server listen port                         ## Server listening socket port number
            (16) Maximum number of server connections ## The maximum number of connections supported by the server
            (/webnet) Server root directory ## Server root directory
                  Select supported modules --->              ## Functional modules enabled by default
                     [  ] LOG: Enanle output log support
                     -*- AUTH: Enanle basic HTTP authentication support
```

Machine Translated by Google

```
                    -*- CGI: Enanle Common Gateway Interface support
                    -*- ASP: Enanle Active Server Pages support
                    -*- SSI: Enanle Server Side Includes support
                    -*- INDEX: Enanle list all the file in the directory support
                    -*- ALIAS: Enanle alias support [
                      ] DAV: Enanle Web-based Distributed Authoring and Versioning
                         support
                    -*- UPLOAD: Enanle upload file support [
                      ] GZIP: Enable compressed file support by GZIP
                    (0) CACHE: Configure cache level
          [*] Enable webnet samples                              ## Start the test routine
               Version (latest) --->
```

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

• Compile and download

2.1.2 Page file preparation

In the WebNet package example, you need to get local static pages, which requires the support of the file system (FAT file system, ROMFS file system, etc.).

system, etc., only the device virtual file system that supports RT-Thread is needed).

Static pages need to be uploaded to the server root directory in the file system (the root directory used in the example is /webnet). After the device successfully mounts the file system, you need to perform the following operations in sequence:

1. Use the mkdir webnet command to create the WebNet package root directory **/webnet and use** the cd webnet command to enter the directory.

2. Use the mkdir admin and mkdir upload commands to create /**webnet/admin** and /**webnet/upload** for AUTH Functional and Upload functional testing;

3. Upload the three files index.html, index.shtml, and version.asp in the /sample **directory of** the WebNet software package to the /**webnet** directory (WebNet root directory) of the device in sequence. (You can use the TFTP tool to upload files. For specific operation methods, refer to the TFTP user manual.)

After successfully creating the directory and uploading the file, you can start the routine to test the WebNet software function.

2.2 Startup routine

The parameters and environment configuration of this routine are as follows:

• Listening port number: 80

• Root directory address: /webnet

• File system: FAT file system

After the device is started and connected to the network successfully, enter the webnet_test command in the Shell command line to start the WebNet server.

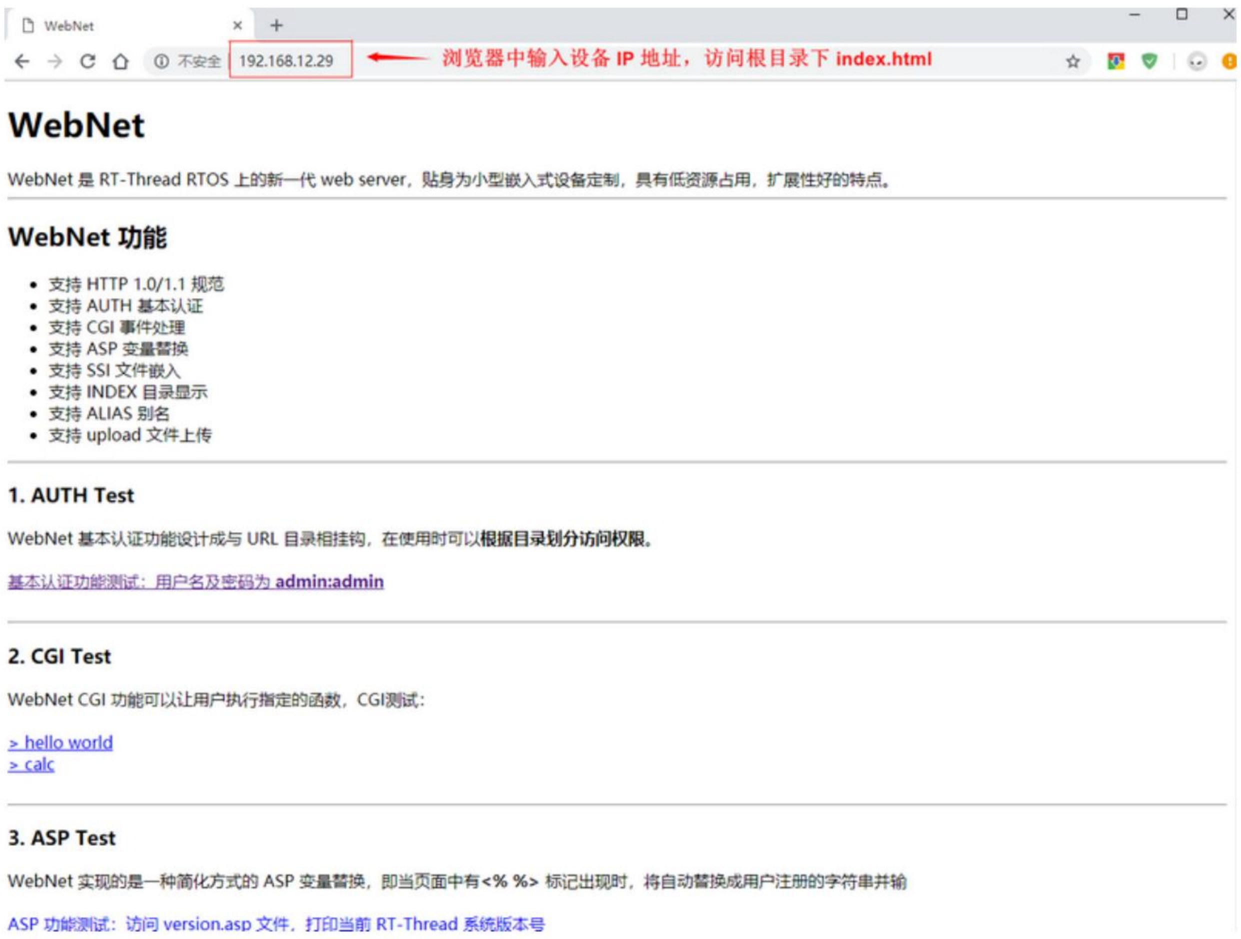
The Shell command line displays the following log information, indicating that the WebNet server has been successfully initialized:

```
msh />webnet_test [I/wn]
 RT-Thread webnet package (V2.0.0) initialize success.
```

Then use the ifconfig command in the Shell command line to obtain the IP address of this device as **192.168.12.29.**

```
msh />ifconfig
network interface: w0 (Default)
PERSON: 1500
MAC: 44 32 c4 75 e0 59
FLAGS: UP LINK_UP ETHARP BROADCAST IGMP
ip address: 192.168.12.29 gw address:
192.168.10.1
netmask : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 223.5.5.5
```

Then enter the device IP address in the browser (here we use Google Chrome), and the **/index.html** file in the root directory of the device will be accessed by default. As shown in the figure below, the page file is displayed normally:



WebNet

WebNet 是 RT-Thread RTOS 上的新一代 web server，贴身为小型嵌入式设备定制，具有低资源占用，扩展性好的特点。

WebNet 功能

- 支持 HTTP 1.0/1.1 规范
- 支持 AUTH 基本认证
- 支持 CGI 事件处理
- 支持 ASP 变量替换
- 支持 SSI 文件嵌入
- 支持 INDEX 目录显示
- 支持 ALIAS 别名
- 支持 upload 文件上传

1. AUTH Test

WebNet 基本认证功能设计成与 URL 目录相挂钩，在使用时可以**根据目录划分访问权限。**

基本认证功能测试：用户名及密码为 **admin:admin**

2. CGI Test

WebNet CGI 功能可以让用户执行指定的函数，CGI测试：

> hello world
> calc

3. ASP Test

WebNet 实现的是一种简化方式的 ASP 变量替换，即当页面中有<% %> 标记出现时，将自动替换成用户注册的字符串并输

ASP 功能测试：访问 version.asp 文件，打印当前 RT-Thread 系统版本号

Figure **2.1:** Routine main page

This page shows the basic functions of the WebNet software package and provides corresponding demonstration examples according to different functions.

The following routines are introduced in sequence:

Machine Translated by Google

• AUTH basic authentication routine •

CGI event handling routine • ASP

variable substitution routine • SSI

file nesting routine • INDEX

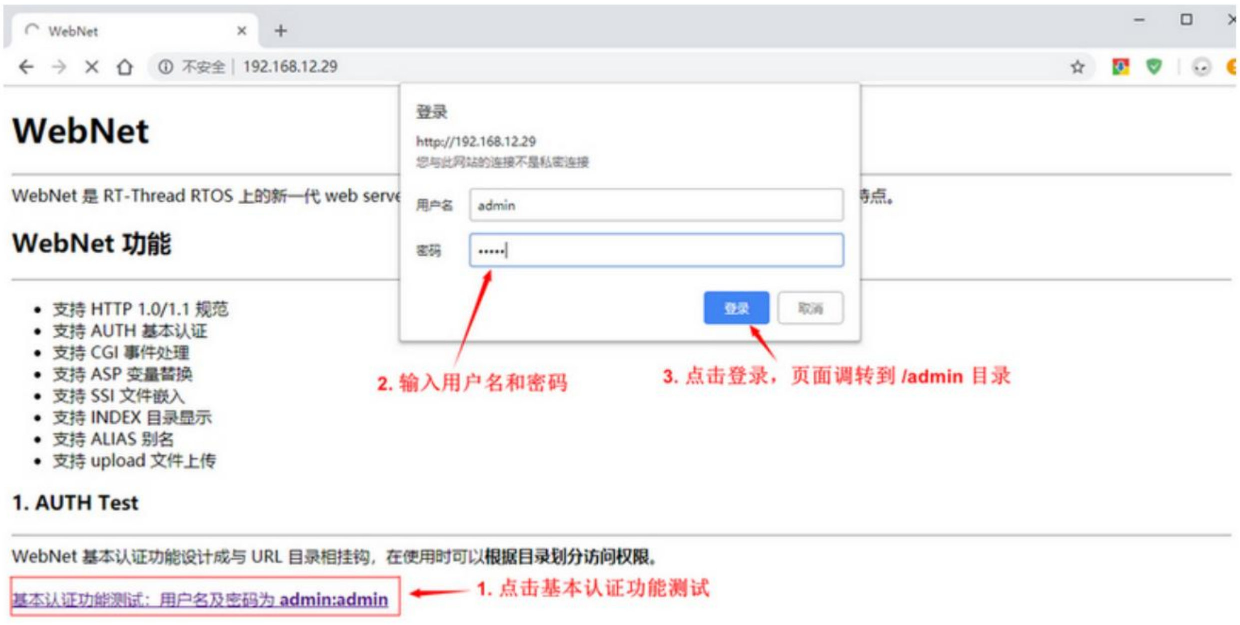
directory display routine • ALIAS alias

access routine • Upload file upload

routine

2.2.1 AUTH Basic Authentication Routine

On the homepage (/index.html) of the example program, under the AUTH Test module, click the Basic Authentication Function Test: Username and password are admin:admin. A Basic Authentication dialog box pops up. Enter the username **admin and password** admin. After successfully entering the username and password, you will be directed to the /admin directory under the root directory, as shown in the following figure:



WebNet

不安全 | 192.168.12.29

WebNet

WebNet 是 RT-Thread RTOS 上的新一代 web serve 特点。

WebNet 功能

- 支持 HTTP 1.0/1.1 规范
- 支持 AUTH 基本认证
- 支持 CGI 事件处理
- 支持 ASP 变量替换
- 支持 SSI 文件嵌入
- 支持 INDEX 目录显示
- 支持 ALIAS 别名
- 支持 upload 文件上传

1. AUTH Test

WebNet 基本认证功能设计成与 URL 目录相挂钩，在使用时可以根据目录划分访问权限。

基本认证功能测试：用户名及密码为 admin:admin

登录

http://192.168.12.29

您与此网站的连接不是私密连接

用户名 admin

密码 •••••

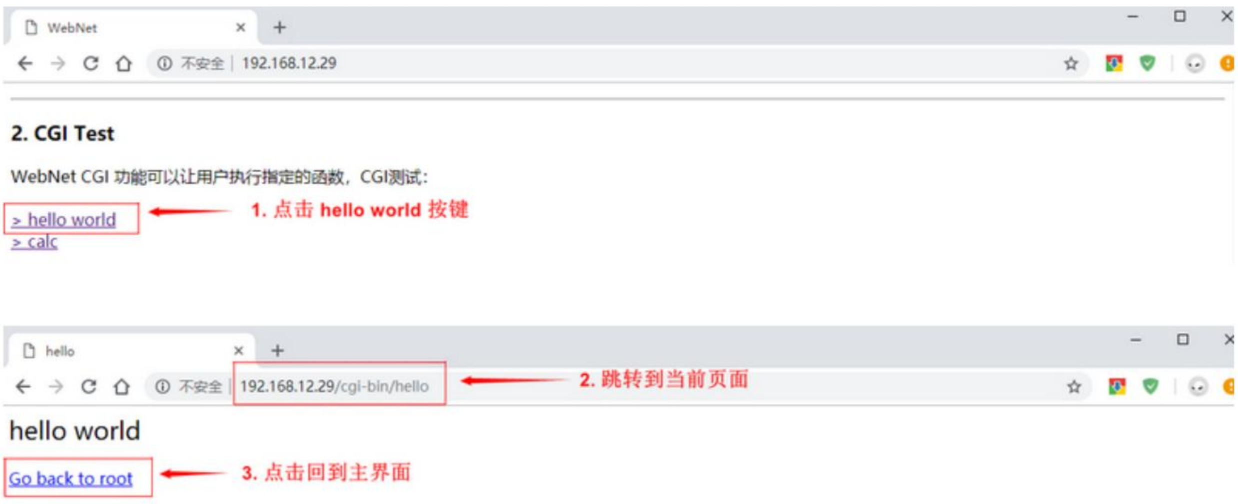
Figure 2.2: Basic authentication routine

2.2.2 CGI event handling routines

This example provides two CGI **examples: the hello world** example and the **calc** example, which are used to demonstrate the basic functions of CGI event processing.

• **Hello world** example

The hello world example demonstrates the ability to display text content on a page. Click the > hello world button under the CGI Test module on the example's homepage to jump to a new page displaying log information. If the new page displays "hello world," it indicates that the CGI event was successfully processed. Then, click the Go back to root button on the new page to return to the example's main page, as shown below:


WebNet
192.168.12.29

2. CGI Test

WebNet CGI 功能可以让用户执行指定的函数，CGI测试：

> hello world
> calc

hello
192.168.12.29/cgi-bin/hello

hello world

Go back to root

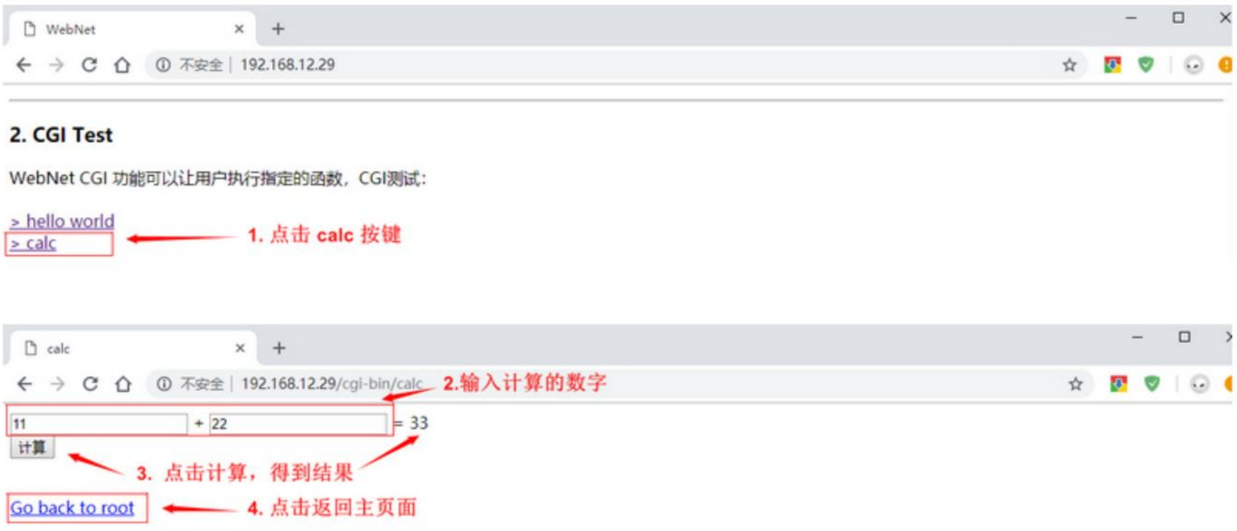
Figure 2.3: *Hello World* Routines

• **calc** routines

The calc routine uses CGI functions to display a simple addition calculator function on the page. Click on the CGI Test module on the routine home page.

> calc button, it will jump to a new page, enter two numbers and click the calculation button, the page will show the result of adding the two numbers.

Click the Go back to root button to return to the main page of the routine, as shown below:


WebNet
192.168.12.29

2. CGI Test

WebNet CGI 功能可以让用户执行指定的函数，CGI测试：

> hello world
> calc

calc
192.168.12.29/cgi-bin/calc

11 + 22 = 33
计算

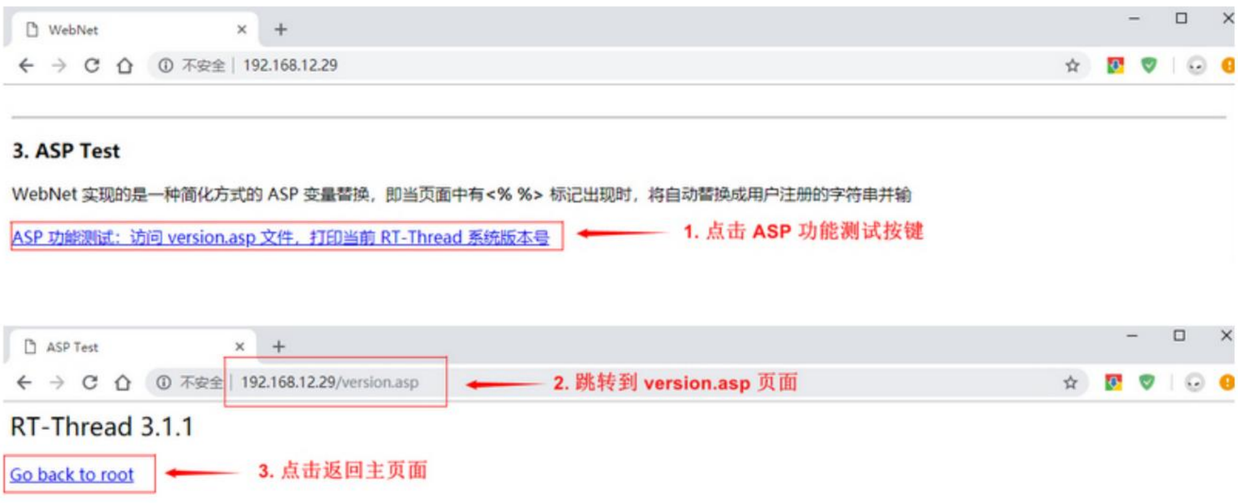
Go back to root

Figure 2.4: *calc* Routines

2.2.3 ASP variable replacement routine

The ASP example demonstrates the page variable replacement function. Click ASP Function Test under the ASP Test module on the example home page: access version .asp file button will jump to the version.asp page in the root directory, which displays the version number of the current RT-Thread system.

Click the Go back to root button on the new page to return to the main page of the routine, as shown below:



3. ASP Test

WebNet 实现的是一种简化方式的 ASP 变量替换，即当页面中有<% %> 标记出现时，将自动替换成用户注册的字符串并输

ASP 功能测试：访问 version.asp 文件，打印当前 RT-Thread 系统版本号

1. 点击 ASP 功能测试按键

2. 跳转到 version.asp 页面

RT-Thread 3.1.1

Go back to root

3. 点击返回主页面

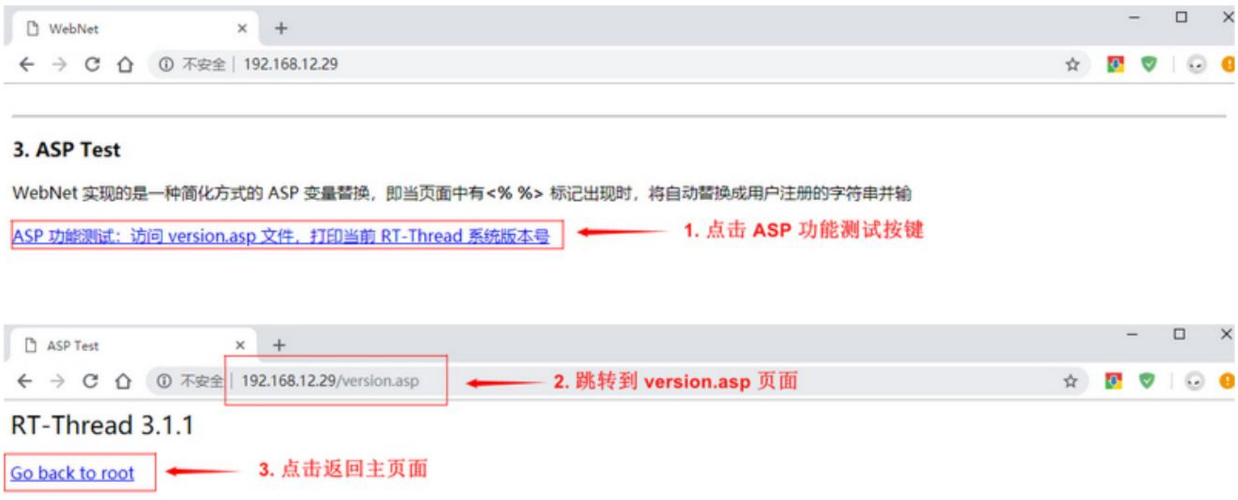
Figure 2.5: *ASP* Routines

2.2.4 SSI file nesting routine

The SSI routine demonstrates the function of nesting a page in another page. Click SSI Function Test under the SSI Test module on the routine homepage.

Try: Accessing the /version.shtml page button will jump to the index.shtml page in the root directory, which displays the main page nested in the page.

index.html content. Click the Go back to root button on the new page to return to the main page of the example.



3. ASP Test

WebNet 实现的是一种简化方式的 ASP 变量替换，即当页面中有<% %> 标记出现时，将自动替换成用户注册的字符串并输

ASP 功能测试：访问 version.asp 文件，打印当前 RT-Thread 系统版本号

1. 点击 ASP 功能测试按键

2. 跳转到 version.asp 页面

RT-Thread 3.1.1

Go back to root

3. 点击返回主页面

Figure 2.6: *SSI* Routines

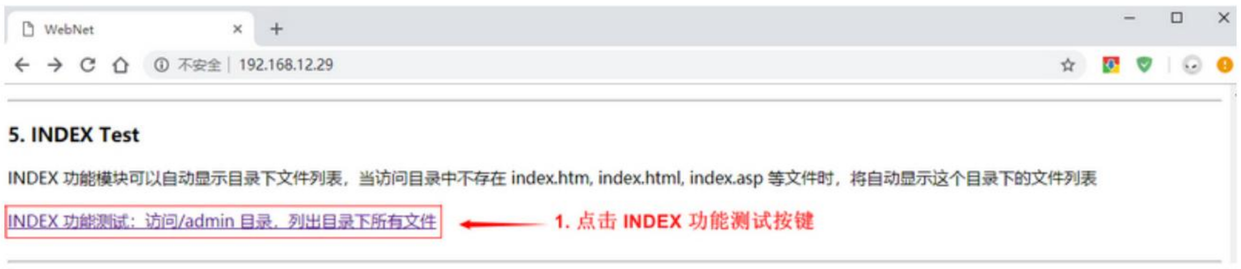
2.2.5 INDEX directory display routine

The INDEX routine demonstrates the page file listing functionality.

First, you need to upload a file to the /admin directory of the root directory. In the example, the admin.txt file has been uploaded to this directory.

Then click INDEX function test under the INDEX Test module on the routine homepage : Access the /admin directory button and you will jump to the root directory

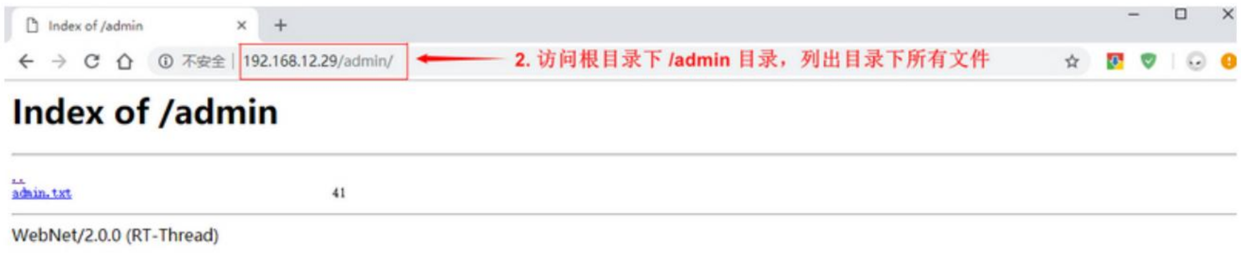
/admin directory and lists all file names and file lengths in the directory, as shown in the following figure:


5. INDEX Test

INDEX 功能模块可以自动显示目录下文件列表，当访问目录中不存在 index.htm, index.html, index.asp 等文件时，将自动显示这个目录下的文件列表

INDEX 功能测试：访问/admin 目录，列出目录下所有文件

1. 点击 INDEX 功能测试按键


192.168.12.29/admin/

2. 访问根目录下 /admin 目录，列出目录下所有文件

Index of /admin

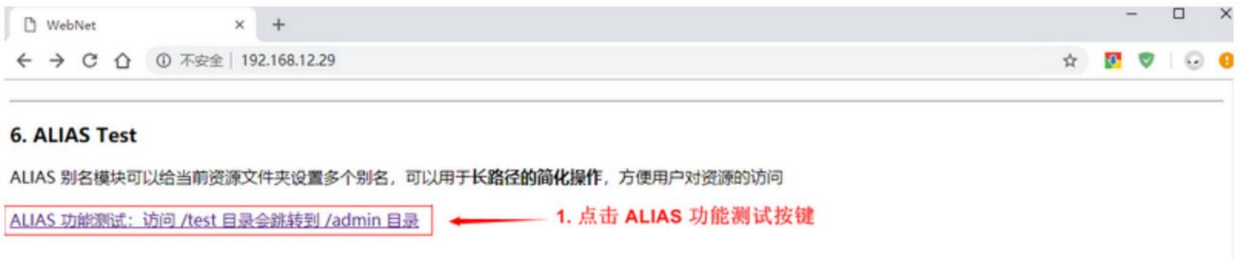
admin.txt 41

WebNet/2.0.0 (RT-Thread)

Figure 2.7: *INDEX* Routines

2.2.6 ALIAS Alias Access Routines

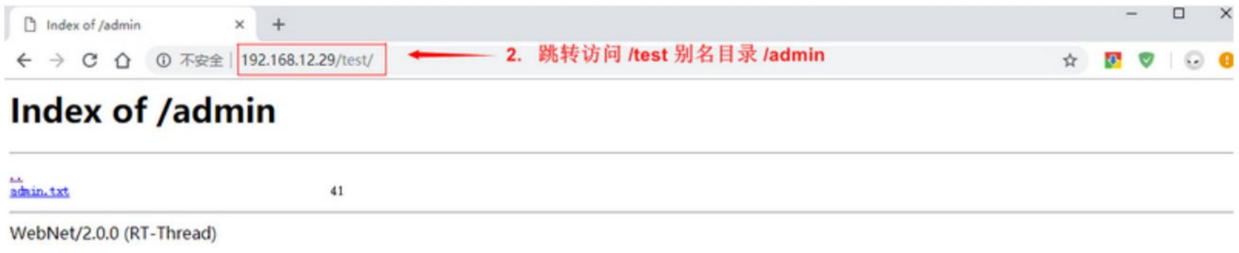
The ALIAS routine demonstrates the ability to use a directory alias to access the directory. In this routine code, the /test directory has been set to an alias of / admin, click ALIAS Function Test under the ALIAS Test module on the routine homepage : Accessing the /test directory will jump to the /admin directory button, We actually access the /test directory, but will jump to the /admin directory and list all the file information in the directory, as shown in the following figure:


6. ALIAS Test

ALIAS 别名模块可以给当前资源文件夹设置多个别名，可以用于长路径的简化操作，方便用户对资源的访问

ALIAS 功能测试：访问 /test 目录会跳转到 /admin 目录

1. 点击 ALIAS 功能测试按键


192.168.12.29/test/

2. 跳转访问 /test 别名目录 /admin

Index of /admin

admin.txt 41

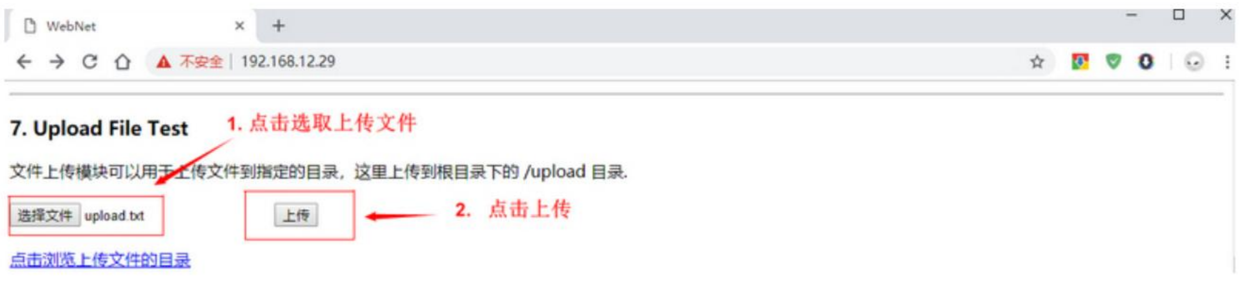
WebNet/2.0.0 (RT-Thread)

Figure 2.8: *ALIAS* Routines

2.2.7 Upload file upload routine

The Upload routine implements the function of uploading files to a fixed directory on the WebNet server. Click on the Upload File Test module on the routine homepage. Click the Select File button, select the file to be uploaded (this example uses the upload.txt file), and click Upload to upload the file to the root directory.

The /upload directory under the directory is as shown below:



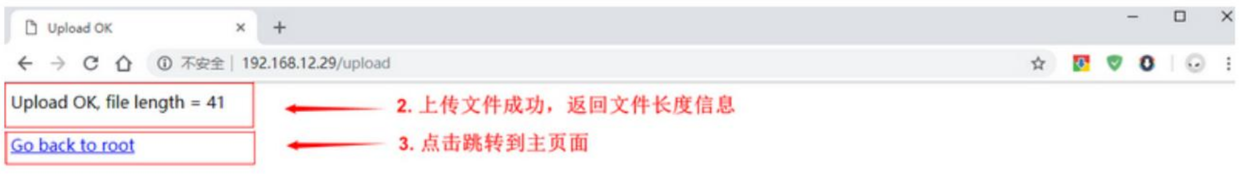
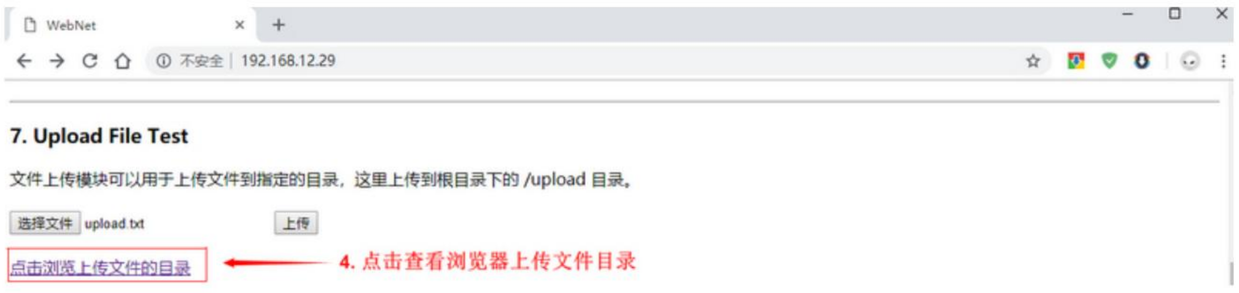


Figure 2.9: Upload files

After the file is uploaded successfully, return to the routine homepage and click the Browse Upload File Directory button to access the /upload file and view the file you just uploaded.

The file information just uploaded.


WebNet
192.168.12.29
7. Upload File Test
文件上传模块可以用于上传文件到指定的目录，这里上传到根目录下的 /upload 目录。
选择文件 upload.txt
上传
点击浏览上传文件的目录
4. 点击查看浏览器上传文件目录

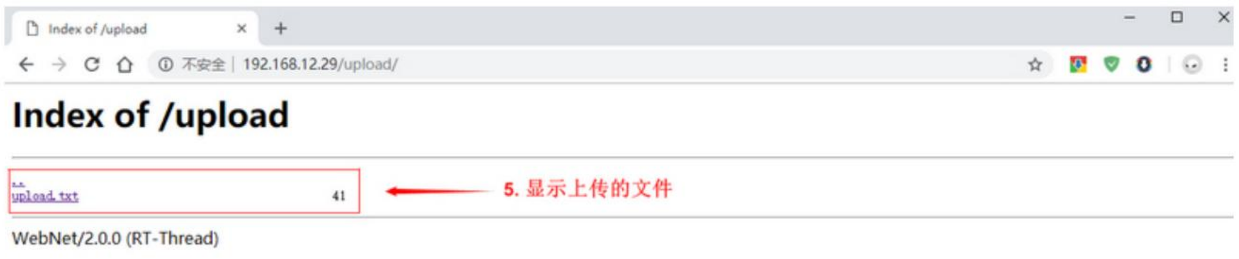


Figure 2.10: View uploaded files

Chapter 3

How it works

The WebNet software package is mainly used to implement HTTP servers on embedded devices. The main working principle of the software package is based on the HTTP protocol.

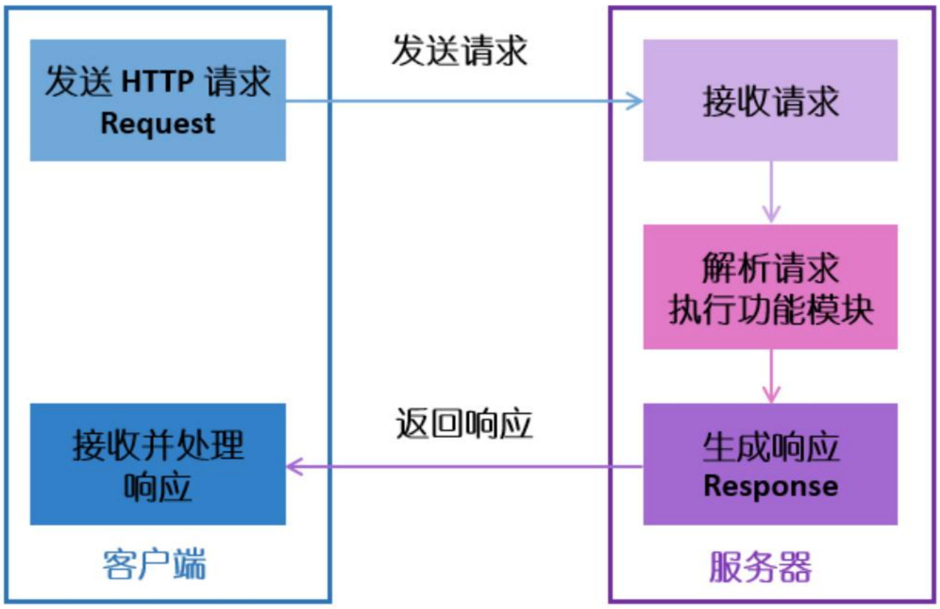


Figure 3.1: *WebNet* How the software package works

The HTTP protocol defines how clients request data from servers and how servers transmit data to clients. The HTTP protocol uses a request/response model. The client sends a request message to the server, which includes the request method, URL, protocol version, request headers, and request data. The server parses the request headers, executes the corresponding function, and sends a response message to the client. The response data includes the protocol version, success or error code, server information, response headers, 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. The default connection port number in WebNet is 80.

2. The client sends an HTTP request

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

3. The server receives the request and parses the data information

After receiving the client's request, the server starts parsing the request information, locates the requested resource file on the server, and then sends the response data to the client, which reads it. A response data 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.

Chapter 4

Usage Guidelines

This section mainly introduces the basic usage process of WebNet software package, and further explains the structures and important APIs that are often involved in the usage process.

A brief description of the line.

4.1 Preparation

4.1.1 ENV Configuration Instructions

First, you need to download the WebNet software package and add it to the project. In the BSP directory, use the menuconfig command to open the ENV configuration interface. Select the WebNet software package in RT-Thread online packages ÿ IoT - internet of things . The specific path is as follows:

```
RT-Thread online packages
    IoT - internet of things --->
        [*] WebNet: A HTTP Server for RT-Thread
            (80) Server listen port
            (16) Maximum number of server connections (/webnet)
            Server root directory
                Select supported modules --->
                    [*] LOG: Enanle output log support
                    [*] AUTH: Enanle basic HTTP authentication support
                    [*] CGI: Enanle Common Gateway Interface support
                    [*] ASP: Enanle Active Server Pages support
                    [*] SSI: Enanle Server Side Includes support
                    [*] INDEX: Enanle list all the file in the directory support
                    [*] ALIAS: Enanle alias support
                    [*] DAV: Enanle Web-based Distributed Authoring and Versioning
                        support
                    [*] UPLOAD: Enanle upload file support
                    [*] GZIP: Enable compressed file support by GZIP
                    (2) CACHE: Configure cache level
                        (1800) Cache-Control time in seconds
            [*] Enable webnet samples
                Version (latest) --->
```

Server listen port: configure the server listening port number;

Maximum number of server connections: Configure the maximum number of connections supported by the server;

Server root directory: Configure the server root directory path;

Select supported modules: Select the function modules supported by the server. By default, all function modules are supported.

• LOG: Configure to enable support for the WebNet software package log function; •
AUTH: Configure to enable support for the Basic Authentication function; • CGI: Configure to enable support for
the CGI event processing function; • ASP: Configure to enable support for the
ASP variable replacement function; • SSI: Configure to enable support for the
file embedding function; • INDEX: Configure to enable support
for displaying the current page file list; • ALIAS: Configure to enable support for the alias access
function; • DAV: Configure to enable support for Web-based distributed
authoring and version control; • Upload: Configure to enable support for the file upload function; • GZIP:
Configure to enable support for server compressed file access; • CHCHE:
Configure to enable support for the cache function (optional levels 0, 1, 2); •
Cache-Control time: Configure the effective time of the cache function, the cache function level
needs to be 2;

Enable webnet samples : configure and add server sample files;

Version: Configure the software package version.

Here we enable all functions of the WebNet package by default and select **the** latest version. Select the appropriate configuration items and

After the version is updated, use the pkgs --update command to download the package and update the user configuration.

4.2 File System Usage Instructions

The use of WebNet software package requires file system support (FAT file system, ROMFS file system, etc., support for RT-Thread

Device virtual file system), used for storing static pages accessed in the WebNet software package, and storing uploaded and downloaded files.

4.3 Workflow

Before using the WebNet software package, you can first understand the basic workflow of the WebNet software package, as shown below:

• Initialize the WebNet software package and start listening for connection requests;

• Receive connection request and create connection session;

• Receive HTTP request data and parse request information;

• Determine the requested functional module and execute the corresponding function;

• Return HTTP response data;

• Close the connection session.

The WebNet software package enables browser-based access to web pages stored on the device, as well as uploading and downloading files. The WebNet software package uses a process based on the HTTP protocol's request and response process. It analyzes the HTTP request type and parameters, determines the response, executes the corresponding functional module, and correctly returns the HTTP response data to complete the entire process.

The following uses the example of a browser accessing the main page in the root directory of a WebNet server to introduce the basic workflow of WebNet:

1. Initialize the **WebNet** software package

```
int webnet_init(void);
```

The WebNet package needs to be initialized before use. The initialization function creates a webnet thread. This thread is used to initialize the enabled functional modules, create a server listening socket, and use the listening socket to wait for client connections and data exchange. The following figure shows the main operations of the thread function:

```
/* WebNet server thread processing function*/
static void webnet_thread(void *parameter) {

    ....
    /* Create a listening socket */
    listenfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    ...
    /* Initialize the enabled function
    module*/ webnet_module_handle_event(RT_NULL, WEBNET_EVENT_INIT);

    /* Waiting for connection request, waiting for receiving data*/
    for (;;) {

        ...
        sock_fd = select(maxfdp1, &tempfds, &tempwrtfds, 0, 0); if (sock_fd == 0)

            continue;

        if (FD_ISSET(listenfd, &tempfds)) {

            /* Handle new connection request */
        }

        /* Process received or sent data */
        webnet_sessions_handle_fds(&tempfds, &writeset);
    }
    ...
}
```

2. Receive connection request and create connection session

After the WebNet initialization thread is successfully created, when a new connection request is generated, a connection session structure will be created. The structure definition is as follows:

```
struct webnet_session {

        struct webnet_session *next;                          //Session structure list

        int socket; struct
        sockaddr_in cliaddr; struct webnet_request*
        request;                                              // Session request related information

        rt_uint16_t buffer_length; rt_uint16_t
        buffer_offset; rt_uint8_t
        buffer[WEBNET_SESSION_BUFSZ];                         //Session buffer data, used to receive request data
                according to
        rt_uint32_t session_phase; rt_uint32_t                // Current session status
        session_event_mask; const struct
        webnet_session_ops* session_ops; // Session event execution functions read, write,
                close, etc.
        rt_uint32_t user_data;
};
```

The webnet_session structure stores information about the currently established session and can be used to manage the entire process of connecting to the current session. Before performing HTTP data exchange, this structure must be created and initialized. The creation of a new session is already completed in **the webnet** thread, as shown below:

```
struct webnet_session* accept_session; accept_session =
webnet_session_create(listenfd); if (accept_session == RT_NULL) {


        /* Creation failed, close the connection */
}
```

3. Receive **HTTP** request data, parse the request information, and successfully create a session structure. When the connection session receives an HTTP request, it processes the received HTTP request and sequentially parses the request type, header information, and additional parameters. The general process of parsing request information is as follows:

```
/* This function is used to parse the request mode, header information and parameters of the
current session connection*/ static void _webnet_session_handle_read(struct webnet_session* session) {

        /* Read the current HTTP connection session data*/
        ....

        if (session->buffer_offset) {

                /* Parse HTTP request mode (GET, POST, etc.) */ if (session-
                >session_phase == WEB_PHASE_METHOD) {

                        webnet_request_parse_method(...);
                }
                /* Parse HTTP request header information*/
                if (session->session_phase == WEB_PHASE_HEADER) {

                        webnet_request_parse_header(...);
                }

                /* Parse the request parameters in the HTTP URL*/
                if (session->session_phase == WEB_PHASE_QUERY) {

                        webnet_request_parse_post(...);
                }
        }
}
```

4. Determine the requested function module and execute the corresponding function

By parsing the request mode and header information, the basic information of the current connection session request is obtained. Then, the type of functional module used is determined and the corresponding module is executed. The general process of judgment is as follows:

```
/* This function is used in WebNet to judge and execute the current session request, such as logging function, CGI event processing function
        wait*/
static int _webnet_module_system_uri_physical(struct webnet_session* session, int
        event)
{
        /* If the LOG function module is enabled, use the LOG log output function*/
#ifdef WEBNET_USING_LOG
        webnet_module_log(session, event);
#endif

          /* If the ALIAS function module is enabled, determine whether the current request is an alias request*/
#ifdef WEBNET_USING_ALIAS
        result = webnet_module_alias(session, event); if (result ==
        WEBNET_MODULE_FINISHED) return result;
#endif

          /* If the AUTH function module is enabled, determine whether the current request requires basic authentication*/
#ifdef WEBNET_USING_AUTH
        result = webnet_module_auth(session, event); if (result ==
        WEBNET_MODULE_FINISHED) return result;
#endif

        /* If the CGI function module is enabled, determine whether the current request needs to perform CGI operations*/
#ifdef WEBNET_USING_CGI
        result = webnet_module_cgi(session, event); if (result ==
        WEBNET_MODULE_FINISHED) return result;
#endif
            ...
}
```

5. Return **HTTP** response data

After successfully judging the function module type and correctly executing the corresponding function, the WebNet server will respond to the request of the current session connection. For example, after the CGI function is executed, the webnet_session_printf or webnet_session_write function can be used in the CGI execution function to send the response data to the client.

```
/* This function is the CGI function execution function. When the browser accesses the current CGI event, the function is executed to return the response header information.
        and data*/
static void cgi_hello_handler(struct webnet_session* session) {

        /* Splice the page data that needs to be sent*/
        ....

    /* Send response header information */
    webnet_session_set_header(session, mime_get_type(".html"), 200, "Ok", strlen(
            status));

    /* Send response data */
    webnet_session_write(session, (const rt_uint8_t*)status, rt_strlen(status));
}
```

6. Close the connection session

After the current session connection request is successfully parsed, the functional module is executed, and the response data is sent, the current connection session will be closed, the session structure will be released, and the entire HTTP data interaction process will be completed, enabling access to the web page files provided by the device on the browser, or completing the upload and download operations of files on the server.

4.4 Usage

For the various functional modules of the WebNet server, some functional modules need to set corresponding configuration parameters before use.
The module needs to cooperate with the page code to realize its function. Next, we will introduce how to use different functional modules of the WebNet server.

• **LOG** log display function

After it is turned on, basic information of the session request can be displayed, such as the IP address and port number of the connection setting, HTTP request type, access
Address and other information, it is recommended to turn it on when debugging the code.

• **AUTH** basic authentication function

Basic Authentication allows you to assign access permissions to directories. Before initializing the WebNet server, you need to call the webnet_auth_set function to set the directory's username and password (in the format of username:password). When accessing the directory in a browser, you must enter the correct username and password. The relevant functions are defined as follows:

```
/* Set directory basic authentication information */
void webnet_auth_set(const char* path, const char* username_password);
```

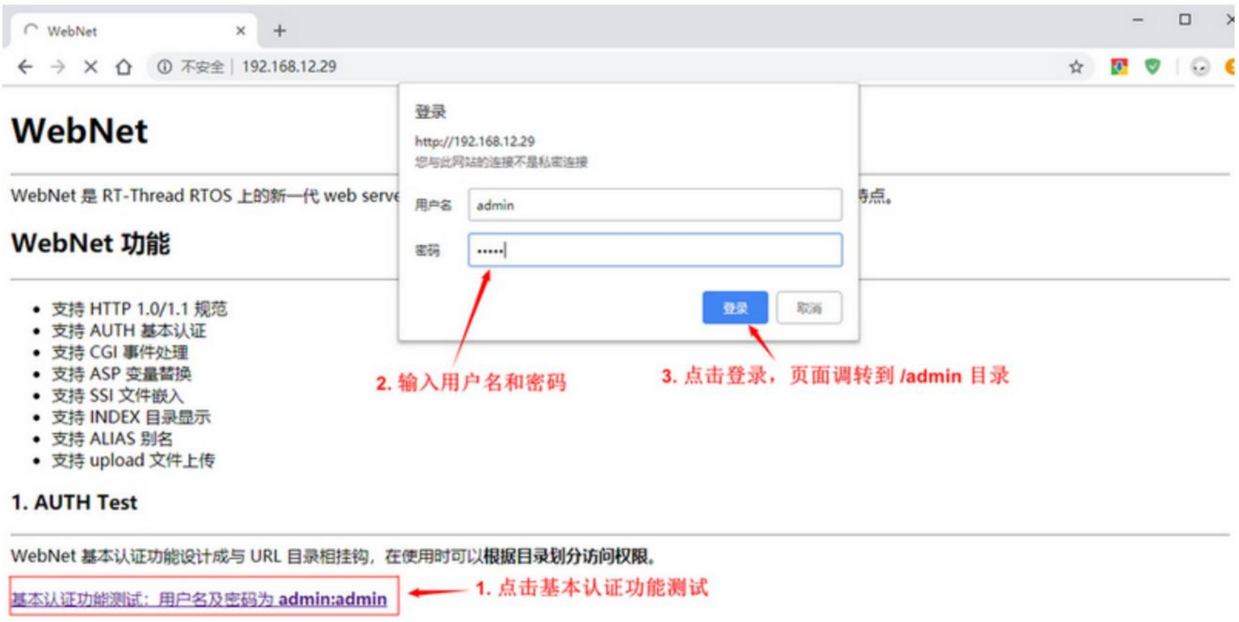
The sample code for the AUTH basic authentication function is as follows:

```
void webnet_test(void) {

        /* Set the username and password of the /admin directory to admin */
          webnet_auth_set("/admin", "admin:admin"); webnet_init();


}
```

After setting up the basic authentication function for the /admin directory, when accessing the /admin directory in a browser, you need to enter the set username and password to access it, as shown in the figure:



WebNet

WebNet 是 RT-Thread RTOS 上的新一代 web serve … 特点。

WebNet 功能

- 支持 HTTP 1.0/1.1 规范
- 支持 AUTH 基本认证
- 支持 CGI 事件处理
- 支持 ASP 变量替换
- 支持 SSI 文件嵌入
- 支持 INDEX 目录显示
- 支持 ALIAS 别名
- 支持 upload 文件上传

1. AUTH Test

WebNet 基本认证功能设计成与 URL 目录相挂钩，在使用时可以根据目录划分访问权限。

基本认证功能测试：用户名及密码为 admin:admin

登录
http://192.168.12.29
您与此网站的连接不是私密连接
用户名 admin
密码 •••••

Figure 4.1: Basic authentication function

- **CGI** functionality

The CGI function can customize the execution function of an event. When the browser sends a request corresponding to the event, the WebNet server can perform the corresponding operation. It is necessary to call the webnet_cgi_register function to register the CGI execution function before initializing the WebNet server. The relevant function definitions are as follows:

```
/* Set the CGI event root directory*/
void webnet_cgi_set_root(const char* root); /* Set CGI event
execution function*/ void
webnet_cgi_register(const char* name, void (*handler)(struct webnet_session*
        session));

/* Send header information to the client of WebNet connection, used in CGI event execution
function*/ void webnet_session_set_header(struct webnet_session *session, const char* mimetype,
          int code, const char* status, int length);
/* Send fixed format data to the client of WebNet connection, used in CGI event execution function*/
void webnet_session_printf(struct webnet_session *session, const char* fmt, ...); /* Send data to the client of the WebNet
connection, used in the CGI event execution function*/
```

```
int webnet_session_write(struct webnet_session *session, const rt_uint8_t* data,
        rt_size_t size);
```

The sample code for using the CGI function is as follows:

```
static void cgi_hello_handler(struct webnet_session* session) {

          const char* hello = "Hello World\n";
          webnet_session_set_header(session, mime_get_type(".html"), 200, "Ok", strlen(
                  status));
          webnet_session_write(session, hello, rt_strlen(hello)); webnet_session_printf(session,
          "%s", hello);

} void webnet_test(void) {

        /* Set CGI event execution function*/
      webnet_cgi_register("hello", cgi_hello_handler);
        webnet_init();
}
```

The corresponding page code is as follows. Clicking the **hello world** button on the browser will send the corresponding CGI request to the server.

```
<html>
    <body>
        <hr>
        <h3> CGI Test</h3>
        WebNet CGI function allows users to execute specified functions, CGI test:
        <br/><br/>

        <br/>
    </body>
</html>
```

- **ASP** variable replacement function

The ASP variable replacement function can match the variables contained in the **<% %>** tags in the web page code and replace them with the execution functions registered in the code.

Therefore, before initializing WebNet, you need to call webnet_asp_add_var to set the ASP variable replacement function. The relevant function definition is as follows:

```
/* Set ASP variable execution function*/
void webnet_asp_add_var(const char* name, void (*handler)(struct webnet_session*
        session));
```

The ASP function sample code is as follows:

```
static void asp_var_version(struct webnet_session* session) {

        webnet_session_printf(session, "RT-Thread: %d.%d.%d\n", RT_VERSION,
                RT_SUBVERSION, RT_REVISION);
}

void webnet_test(void) {

        /* Set ASP variable to execute function*/
        webnet_asp_add_var("version", asp_var_version); webnet_init();

}
```

The corresponding page code is as follows (the file name is version.asp). Accessing this page code will replace ASP to display the latest version of RT-Thread:

```
<html>
    <head>
        <title> ASP Test </title>
    </head>
    <body>
        <% version %>                              /* ASP variable is replaced with RT_Thread version number display*/
    </body>
</html>
```

• **SSI** file nesting function

WebNet supports embedding text files into web pages. If there is a markup in the page, it will be replaced with the corresponding file content. SSI Functional pages usually end with .shtml **or .stm .** The following is the sample page code (the file name is index.shtml):

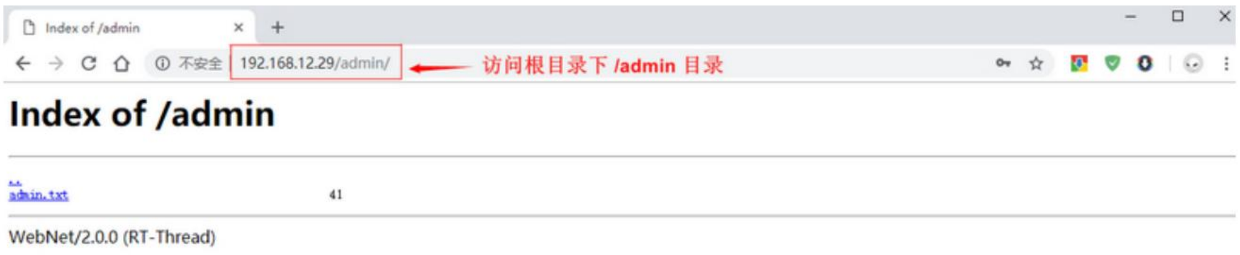
```
<html>
    <head>
        <title> SSI Test </title>
    </head>
    <body>
        <h1> SSI Test</h1>
        <font size=\"+2\">The index.html page embedded in the following page</font>
        <hr>
        <!--#include virtual="/index.html" --> /* This page embeds the content of the index.html file*/
    </body> </
html>
```

• **INDEX** directory file display function

After the WebNet server is initialized successfully, enter the set IP address and the directory to be accessed directly in the browser to list the current directory.

Record all the files, as shown below, access the server/admin directory, and list all the files in the directory:

Machine Translated by Google



Index of /admin

admin.txt 41

WebNet/2.0.0 (RT-Thread)

Figure 4.2: Directory display function

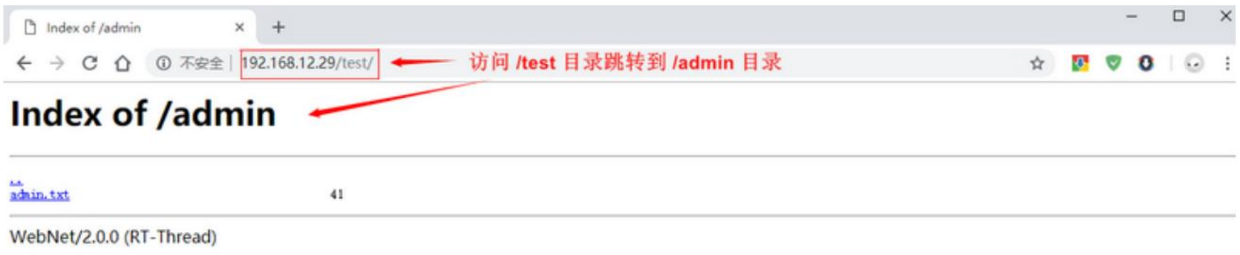
• **ALIAS** alias access function

ALIAS Alias access function can set up alias access for a folder. You need to set up the folder before initializing the WebNet server.

Alias, as shown in the following code, calls the webnet_alias_set function to set the alias of the /test directory to /admin. When the browser accesses /test

Will jump to the /admin directory:

```
void webnet_test(void)
{
        /* Set the alias of the /test directory to /admin */
        webnet_alias_set("/test", "/admin");
        webnet_init();
}
```



Index of /admin

admin.txt 41

WebNet/2.0.0 (RT-Thread)

Figure 4.3: Alias access function

• **Upload** file upload function

Upload file upload function is used to upload local files to the specified directory of WebNet server. Before uploading files, you need to create and implement

Now upload the file structure as shown below:

```
struct webnet_module_upload_entry
{
        const char* url;                                              /* Directory name for file uploads*/

        int (*upload_open) (struct webnet_session* session); int (*upload_close)(struct      /* Open the file */
        webnet_session* session); int (*upload_write)(struct webnet_session* session, const  /* Close the file */
        void* data, rt_size_t
                length);          /* Write data to file */
        int (*upload_done) (struct webnet_session* session);           /* Download complete */
};
```

This structure defines the directory file for uploading files and the event callback functions that need to be used, such as: opening files, closing files, writing data to Documents, etc.

Users need to implement the callback function according to the actual situation. The general operations in each callback function are as follows:

• upload_open: Create and open a file in the specified directory by parsing the file name; • upload_close: Close the file; • upload_write: Write data to the opened file; • upload_done: After the file is uploaded successfully, the response data is returned to the browser.

In the process of implementing the callback function, the function definition that may be used to obtain the current upload file session information is as follows:

```
/* Get the name of the currently uploaded file*/
const char* webnet_upload_get_filename(struct webnet_session* session); /* Get the type of the currently
uploaded file*/ const char*
webnet_upload_get_content_type(struct webnet_session* session); /* Get the file descriptor of the currently
uploaded file after it is opened*/
const void* webnet_upload_get_userdata(struct webnet_session* session);
```

For specific implementation methods, please refer to the implementation methods of each function in the software package /samples/wn_sample_upload.c.

Finally, before WebNet is initialized, you need to call the webnet_upload_add function to set the uploaded file information, as shown in the following code:

```
static int upload_open (struct webnet_session* session) {

    /* Open or create a new file */

} static int upload_close (struct webnet_session* session) {

    /* Close the file */

} static int upload_write (struct webnet_session* session) {

    /* Write data to file */

} static int upload_done (struct webnet_session* session) {

    /* Download completed, return response data*/
}
const struct webnet_module_upload_entry upload_entry_upload = {

    "/upload",
    upload_open,
    upload_close,
    upload_write,
    upload_done
};

void webnet_test(void) {
    /* Register file upload execution function*/
      webnet_upload_add(&upload_entry_upload); webnet_init();

}
```

The code for uploading files on the corresponding page is as follows:

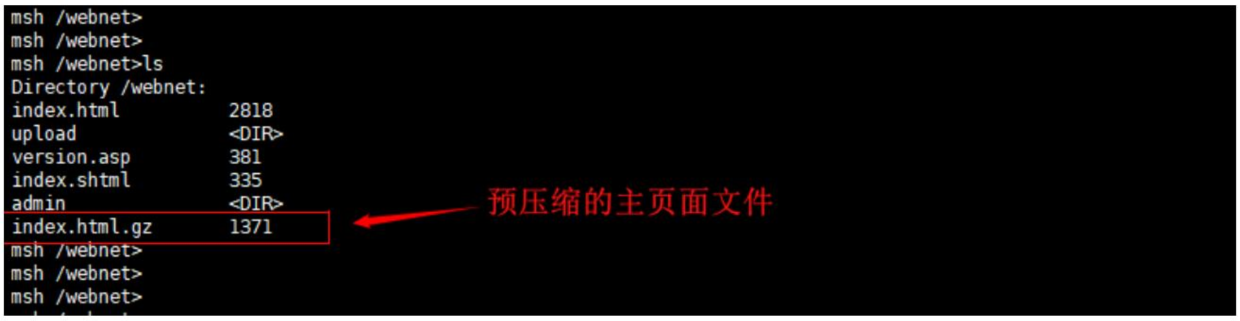
```
<html>
  <body>
      <h3>Upload File Test</h3> The file upload
      module can be used to upload files to a specified directory. Here, it is uploaded to the /upload directory under the root directory .
      <br/><br/>
      <form name="upload" method="POST" enctype="multipart/form-data" action="/upload">
          <input type="file" name="file1" > <input type="submit"
          name="submit" value="ÿ ÿ">
      </form>
      <br/>

      <br/><br/>
  </body>
</html>
```

• Pre-compression function

The WebNet server pre-compression function compresses page resource files on the server in advance, generating a compressed file with a .gz suffix. For example, when a browser accesses the index.html page in the root directory, if a compressed file named **index.html.gz** exists , the contents of the compressed file are sent to the browser, which automatically parses and displays the original page file.

When using the pre-compression function, you need to first upload the compressed **index.html.gz** file to the file system, as shown in the following figure:



```
msh /webnet>
msh /webnet>
msh /webnet>ls
Directory /webnet:
index.html          2818
upload              <DIR>
version.asp         381
index.shtml         335
admin               <DIR>
index.html.gz       1371
msh /webnet>
msh /webnet>
msh /webnet>
```

Figure 4.4: Pre-compression function

4.5 Frequently Asked Questions

1. The browser does not display page information when accessing the device **IP** address

• Cause: The root directory address is set incorrectly;

• Solution: Make sure that the root directory address you set is consistent with the directory address created on the device file system, and that there is a page file under the root directory.

Piece.

2. The device has an **out of pbuf** error

• Cause: Insufficient device memory;

• Solution: WebNet software package functions such as uploading files require additional resource space. It is recommended to run it on a device with sufficient resource space.

line, or use it on qemu.

Chapter 5

API Description

For the convenience of users, commonly used APIs are listed here and related usage instructions are given.

5.1 Initialization Function

```
int webnet_init(void);
```

Used to initialize the WebNet server, including creating threads to monitor client connection events, initializing enabled function modules, etc.

parameter	describe
none	none
return	describe
= 0	Initialization successful
< 0	Initialization failed

5.2 Set the listening socket port

```
void webnet_set_port(int port);
```

It is used to set the current WebNet server listening port number. The default listening port number of the WebNet server is 80, which is also the HTTP protocol. When using the default port number to access the URL address, you can directly access it without entering the port number. When using a non-default port number, you need to To specify the port number in the URL address, such as: http://host:8080/index.html . This function can only be used for the initialization of the **WebNet** server. Before the beginning.

parameter	describe
port	Set the listening socket port
return	describe

parameter	describe
none	none

5.3 Get the listening socket port

```
int webnet_get_port(void);
```

Used to obtain the current WebNet server listening socket port number.

parameter	describe
none	none
return	describe
>=0	Listening socket port number

5.4 Set the server root directory

```
void webnet_set_root(const char* webroot_path);
```

Used to set the current WebNet server root directory path. The default root directory of the WebNet server is /webnet. The browser and WebNet The paths used or accessed in the function are based on the root directory path. When the browser accesses http://host/index.html , the file system /webnet/index.html in the browser is returned to the browser.

parameter	describe
webroot_path	Set the root directory address
return	describe
none	none

5.5 Get the server root directory

```
const char* webnet_get_root(void);
```

Used to obtain the root directory address of the current WebNet server.

parameter	describe
none	none
return	describe

parameter	describe
!= NULL	Root directory address

5.6 Get the type of request link

```
const char* mime_get_type(const char* url);
```

Used to obtain the type of the currently requested URL link, such as web page, image, text, etc.

parameter	describe
url	Request link address
return	describe
!= NULL	Type of requested link

5.7 Add ASP variable processing method

void webnet_asp_add_var(const char* name, void *(handler)(struct webnet_session* session));

This function is used to add an ASP variable processing method. When the added name variable name appears in the ASP file, the corresponding handle operation.

parameter	describe
name	ASP variable names
void *(handler)(struct webnet_session* session)	ASP variable processing
return	describe
none	none

5.8 Add CGI event handling method

void webnet_cgi_register(const char* name, void *(handler)(struct webnet_session* session));

This function is used to register a CGI event handler. When the browser requests a URL with name , the corresponding handle operation.

parameter	describe
name	CGI event name

parameter	describe
void *(handler)(struct webnet_session* session)	CGI event handling
return	describe
none	none

5.9 Setting the CGI Event Root Directory

```
void webnet_cgi_set_root(const char* root);
```

The default CGI event root directory of the WebNet server is /cgi-bin. When the browser requests the address http://host/cgi-bin/test When the CGI event processing function corresponding to the test name is executed.

This function is used to set a new CGI event root directory. The previous CGI root directory will no longer take effect.

parameter	describe
root	CGI event root directory
return	describe
none	none

5.10 Set basic authentication information

```
void webnet_auth_set(const char* path, const char* username_password);
```

Used to set basic authentication information for directory access, including user name and password.

parameter	describe
path	Directory where basic authentication information needs to be set
username_password	The username and password to be set are in the format username:password
return	describe
none	none

5.11 Setting Directory Aliases

```
void webnet_alias_set(char* old_path, char* new_path);
```

Used to set the alias of a directory. After the setting is successful, you can use the directory alias to access the directory.

parameter	describe
old_path	Directory that needs to be aliased
new_path	The directory alias set is generally a directory that exists on the server
return	describe
none	none

5.12 Sending HTTP request header

```
void webnet_session_set_header(struct webnet_session* session, const char* mimetype, int
code, const char* title, int length);
```

Used to splice and send header information to the connected client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session The current server connection session	
mimetype The response file type (Content-Type) to be sent, which can be obtained using the mime_get_type function	
code	The response status code sent is normally 200
title	The response status type sent, normally it is OK
length	The length of the response file to be sent (Content-Length)
return	describe
none	none

5.13 Sending HTTP response data

```
int webnet_session_write(struct webnet_session* session, const rt_uint8_t* data, rt_size_t
size);
```

Used to send response data to the client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session	The current server-connected session
data	Pointer to the data being sent
size	Length of data sent
return	describe
none	none

5.14 Sending HTTP fixed format response data

```
void webnet_session_printf(struct webnet_session* session, const char* fmt, ...);
```

Used to send fixed-format response data to the client, generally used in ASP variable processing functions and CGI event processing functions.

parameter	describe
session	The current server-connected session
fmt	Customized input data expressions
...	Input parameters
return	describe
none	none

5.15 Get the name of the uploaded file

```
const char* webnet_upload_get_filename(struct webnet_session* session);
```

Get the name of the currently uploaded file, used to open or create the file.

parameter	describe
session	The current server-connected session
return	describe
!= NULL	The name of the currently uploaded file

5.16 Get the type of uploaded file

```
const char* webnet_upload_get_content_type(struct webnet_session* session);
```

Get the type of the currently uploaded file.

parameter	describe
session	The current server-connected session
return	describe
!= NULL	The type of the currently uploaded file

5.17 Get uploaded file parameters

```
const char* webnet_upload_get_nameentry(struct webnet_session* session, const char*
name);
```

Gets the registered upload file delimiter (HTTP request boundary parameter).

parameter	describe
session	The current server-connected session
name	Directory path for uploaded files
return	describe
!= NULL	The type of the currently uploaded file

5.18 Get the file descriptor of the uploaded file

```
const void* webnet_upload_get_userdata(struct webnet_session* session);
```

Get the file descriptor generated after the current uploaded file is opened, which is used to read and write data to the file.

parameter	describe
session	The current server-connected session
return	describe
!= NULL	File descriptor for the uploaded file