

Harmony OS 智能硬件 入门系列课程 <快速上手>

快速掌握Hi3861开发板基础开发技巧

第 7 讲：Hi3861上网络编程



本讲内容

- 第1节：lwIP概述
- 第2节：lwIP功能特性
- 第3节：lwIP开发指引
- 第4节：lwIP示例代码

本讲目标

- 1、了解lwIP功能特性
- 2、掌握TCP 服务端开发的方法
- 3、掌握TCP 客户端开发的方法
- 4、掌握UDP 服务端开发的方法
- 5、掌握UDP 客户端开发的方法

第1节：lwIP概述

- 知识点1：lwIP介绍
- 知识点2：RFC（Request For Comments）遵从

知识点1 【lwIP介绍】

- LwIP是Light Weight (轻型)IP协议，有无操作系统的支持都可以运行。LwIP实现的重点是在保持TCP协议主要功能的基础上减少对RAM 的占用，它只需十几KB的RAM和40K左右的ROM就可以运行，这使LwIP协议栈适合在低端的嵌入式系统中使用。
- lwIP协议栈主要关注的是怎么样减少内存的使用和代码的大小，这样就可以让lwIP适用于资源有限的小型平台例如嵌入式系统。为了简化处理过程和内存要求，lwIP对API进行了裁减，可以不需要复制一些数据。

知识点2 【RFC (Request For Comments) 遵从】

■ lwIP遵从如下RFC协议标准:

- RFC 791 (IPv4 Standard)
- RFC 2460 (IPv6 Standard)
- RFC 768 (UDP) User Datagram Protocol
- RFC 793 (TCP) Transmission Control Protocol
- RFC 792 (ICMP) Internet Control Message Protocol
- RFC 826 (ARP) Address Resolution Protocol
- RFC 1035 (DNS) DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION
- RFC 2030 (SNTP) Simple Network Timer Protocol (SNTP) Version 4 for
- IPv4, IPv6 and OSI
- RFC 2131 (DHCP) Dynamic Host Configuration Protocol
- RFC 2018 (SACK) TCP Selective Acknowledgment Option

知识点2 【RFC (Request For Comments) 遵从】

- RFC 7323 (Window Scaling)
- RFC 6675 (SACK for TCP) RFC 3927 (Autoip) Dynamic Configuration of IPv4
- Link-Local Addresses
- RFC 2236 (IGMP) Internet Group Management Protocol, Version 2
- RFC4861 (ND for IPv6) Neighbor Discovery for IP version 6 (IPv6)
- RFC4443 Internet Control Message Protocol (ICMPv6) for the Internet
- Protocol Version 6 (IPv6) Specification
- RFC4862 IPv6 Stateless Address Autoconfiguration
- RFC2710 Multicast Listener Discovery (MLD) for IPv6

本节小结

本讲所学知识点有：

- 知识点1：lwIP介绍
- 知识点2：RFC（Request For Comments）遵从

第2节：lwIP功能特性

- 知识点1：支持的特性
- 知识点2：不支持的特性
- 知识点3：小型化的特性

知识点1 【支持的特性】

■ lwIP支持的特性如下：

- 网际协议版本4 (IPv4: Internet Protocol version 4)
- 互联网控制消息协议 (ICMP: Internet Control Message Protocol)
- 用户数据报协议 (UDP: User Datagram Protocol)
- 传输控制协议 (TCP: Transmission Control Protocol)
- 域名解析器 (DNS: Domain Name System) 客户端
- 动态主机配置协议 (DHCP) 客户端和服务端
- RFC 2131偏差标准
- 以太网地址解析协议 (ARP)
- 因特网组管理协议 (IGMP)
- socket接口类型
- TCP选择性确认选项

知识点2 【不支持的特性】

■ lwIP不支持的特性或协议如下：

- PPPoS/PPPOE
- SNMP代理（目前，lwIP仅支持私有MIB）
- 通过多个网络接口实现IP转发
- 路由功能（仅支持终端设备功能）

知识点3 【小型化的特性】

■ lwIP小型化的特性如下：

- 简单网络时间协议（Simple Network Time Protocol）是一种基于包交换、可变延迟数据网络的计算机系统之间的时钟同步网络协议。lwIP支持基于RFC 2030协议的SNTP版本4。S
- lwIP支持AutoIP模块。
- lwIP提供设置Wi-Fi驱动状态的接口4
- lwIP提供设置Wi-Fi驱动状态为lwIP栈的接口。
- lwIP在SOCK_RAW上提供PF_PACKET选项。

本节小结

本讲所学知识点有：

- 知识点1：支持的特性
- 知识点2：不支持的特性
- 知识点3：小型化的特性

第3节：lwIP开发指引

- 知识点1：前提条件
- 知识点2：依赖关系
- 知识点3：lwIP的使用

知识点1 【前提条件】

■ lwIP需满足下列前提条件：

- lwIP需要优化后方能支持Huawei LiteOS；目前lwIP暂不支持其他操作系统。
- 使用lwIP前，基于lwIP相关配置更新驱动代码。

知识点2 【依赖关系】

■ lwIP的依赖关系如下：

- 依赖于Huawei LiteOS调用。
- 需要通过以太网或Wi-Fi驱动模块完成物理层数据的发送和接收。

知识点3 【lwIP的使用】

- lwIP提供了BSD TCP/IP套接字接口，应用可通过该接口进行连接。lwIP的优势在于：
 - 运行在BSD TCP/IP协议栈上的旧应用代码可直接被移植到lwIP中。
 - lwIP支持DHCP客户端特性，用于配置动态IP地址。
 - lwIP支持DNS客户端特性，应用可通过该特性解析域名。
 - lwIP占用资源少，可满足协议栈功能。

本节小结

本讲所学知识点有：

- 知识点1：前提条件
- 知识点2：依赖关系
- 知识点3：lwIP的使用

第4节：lwIP示例代码

- 示例代码1：TCP服务端
- 示例代码2：TCP客户端
- 示例代码3：UDP服务端
- 示例代码4：UDP客户端

示例代码1 【TCP服务端】

/chapter_07/tcp_server.c

<code/>

```
#include <stdio.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_wifi_api.h"
#include "lwip/ip_addr.h"
#include "lwip/netifapi.h"
#include "lwip/sockets.h"

static char request[128] = "";
void tcp_server(unsigned short port)
{
    ssize_t retval = 0;
    int backlog = 1;
    int sockfd = socket(AF_INET, SOCK_STREAM, 0); // TCP socket
    int connfd = -1;

    struct sockaddr_in clientAddr = {0};
    socklen_t clientAddrLen = sizeof(clientAddr);
    struct sockaddr_in serverAddr = {0};
    serverAddr.sin_family = AF_INET;
```



示例代码1 【TCP服务端】

/chapter_07/tcp_server.c

<code/>

```
serverAddr.sin_port = htons(port);          // 端口号，从主机字节序转为网络字节序
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY); // 允许任意主机接入，0.0.0.0

retval = bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)); // 绑定端口
if (retval < 0) {
    printf("bind failed, %ld!\r\n", retval);
    goto do_cleanup;
}
printf("bind to port %d success!\r\n", port);

retval = listen(sockfd, backlog); // 开始监听
if (retval < 0) {
    printf("listen failed!\r\n");
    goto do_cleanup;
}
printf("listen with %d backlog success!\r\n", backlog);
connfd = accept(sockfd, (struct sockaddr *)&clientAddr, &clientAddrLen);
if (connfd < 0) {
    printf("accept failed, %d, %d!\r\n", connfd, errno);
    goto do_cleanup;
}
```



示例代码1 【TCP服务端】

/chapter_07/tcp_server.c

<code/>

```
printf("accept success, connfd = %d!\r\n", connfd);
printf("client addr info: host = %s, port = %d\r\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
retval = recv(connfd, request, sizeof(request), 0);
if (retval < 0) {
    printf("recv request failed, %d!\r\n", retval);
    goto do_disconnect;
}
printf("recv request{%s} from client done!\r\n", request);
retval = send(connfd, request, strlen(request), 0);
if (retval <= 0) {
    printf("send response failed, %d!\r\n", retval);
    goto do_disconnect;
}
printf("send response{%s} to client done!\r\n", request);

do_disconnect:
    lwip_close(connfd);
do_cleanup:
    printf("do_cleanup...\r\n");
    lwip_close(sockfd);
}
```



示例代码1 【TCP服务端】

操作演示



示例代码2 【TCP客户端】

/chapter_07/tcp_client.c

<code/>

```
#include <stdio.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_wifi_api.h"
#include "lwip/ip_addr.h"
#include "lwip/netifapi.h"
#include "lwip/sockets.h"

static char request[] = "Hello";
static char response[128] = "";
void conent_tcp_server(const char* host, unsigned short port)
{
    ssize_t retval = 0;
    int sockfd = socket(AF_INET, SOCK_STREAM, 0); // TCP socket
    struct sockaddr_in serverAddr = {0};
    serverAddr.sin_family = AF_INET; // AF_INET表示IPv4协议
    serverAddr.sin_port = htons(port); // 端口号，从主机字节序转为网络字节序
```



示例代码2 【TCP客户端】

/chapter_07/tcp_client.c

<code/>

```
if (inet_pton(AF_INET, host, &serverAddr.sin_addr) <= 0) { // 将主机IP地址从“点分十进制”字符串 转化为 标准格式 (32位
整数)
    printf("inet_pton failed!\r\n");
    goto do_cleanup;
}
// 尝试和目标主机建立连接，连接成功会返回0，失败返回 -1
if (connect(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
    printf("connect failed!\r\n");
    goto do_cleanup;
}
printf("connect to server %s success!\r\n", host);
// 建立连接成功之后，这个TCP socket描述符 —— sockfd 就具有了“连接状态”，发送、接收 对端都是 connect 参数指
定的目标主机和端口
retval = send(sockfd, request, sizeof(request), 0);
if (retval < 0) {
    printf("send request failed!\r\n");
    goto do_cleanup;
}
```

示例代码2 【TCP客户端】

/chapter_07/tcp_client.c

<code/>

```
printf("send request{%s} %ld to server done!\r\n", request, retval);

retval = recv(sockfd, &response, sizeof(response), 0);
if (retval <= 0) {
    printf("send response from server failed or done, %ld!\r\n", retval);
    goto do_cleanup;
}

response[retval] = '\0';
printf("recv response{%s} %ld from server done!\r\n", response, retval);

do_cleanup:
printf("do_cleanup...\r\n");
lwip_close(sockfd);
}
```



示例代码2 【TCP客户端】

操作演示



示例代码3 【UDP服务端】

/chapter_07/udp_server.c

<code/>

```
#include <stdio.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_wifi_api.h"
#include "lwip/ip_addr.h"
#include "lwip/netifapi.h"
#include "lwip/sockets.h"

static char message[128] = "";
void udp_server(unsigned short port)
{
    ssize_t retval = 0;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0); // UDP socket

    struct sockaddr_in clientAddr = {0};
    socklen_t clientAddrLen = sizeof(clientAddr);
    struct sockaddr_in serverAddr = {0};
```



示例代码3 【UDP服务端】

/chapter_07/udp_server.c

<code/>

```
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);

retval = bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr));
if (retval < 0) {
    printf("bind failed, %ld!\r\n", retval);
    goto do_cleanup;
}
printf("bind to port %d success!\r\n", port);

retval = recvfrom(sockfd, message, sizeof(message), 0, (struct sockaddr *)&clientAddr, &clientAddrLen);
if (retval < 0) {
    printf("recvfrom failed, %ld!\r\n", retval);
    goto do_cleanup;
}
```



示例代码3 【UDP服务端】

/chapter_07/tcp_client.c

<code/>

```
printf("recv message {%s} %ld done!\r\n", message, retval);
printf("peer info: ipaddr = %s, port = %d\r\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));

retval = sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&clientAddr, sizeof(clientAddr));
if (retval <= 0) {
    printf("send failed, %ld!\r\n", retval);
    goto do_cleanup;
}
printf("send message {%s} %ld done!\r\n", message, retval);

do_cleanup:
printf("do_cleanup...\r\n");

lwip_close(sockfd);
}
```



示例代码4 【UDP客户端】

/chapter_07/udp_client.c

<code/>

```
#include <stdio.h>
#include <unistd.h>
#include "ohos_init.h"
#include "cmsis_os2.h"
#include "hi_wifi_api.h"
#include "lwip/ip_addr.h"
#include "lwip/netifapi.h"
#include "lwip/sockets.h"

static char request[] = "Hello. I am from chinasoft";
static char response[128] = "";

void send_to_udp_server(const char* host, unsigned short port)
{
    ssize_t retval = 0;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0); // UDP socket
    struct sockaddr_in toAddr = {0};
```



示例代码4 【UDP客户端】

/chapter_07/udp_client.c

<code/>

```
toAddr.sin_family = AF_INET;
toAddr.sin_port = htons(port); // 端口号, 从主机字节序转为网络字节序
if (inet_pton(AF_INET, host, &toAddr.sin_addr) <= 0) { // 将主机IP地址从“点分十进制”字符串 转化为 标准格式 (32位整数)
    printf("inet_pton failed!\r\n");
    goto do_cleanup;
}

// UDP socket 是“无连接的”, 因此每次发送都必须先指定目标主机和端口, 主机可以是多播地址
retval = sendto(sockfd, request, sizeof(request), 0, (struct sockaddr *)&toAddr, sizeof(toAddr));
if (retval < 0) {
    printf("sendto failed!\r\n");
    goto do_cleanup;
}
printf("send UDP message {%s} %ld done!\r\n", request, retval);

struct sockaddr_in fromAddr = {0};
socklen_t fromLen = sizeof(fromAddr);
```


示例代码4 【UDP客户端】

/chapter_07/udp_client.c

<code/>

```
// UDP socket 是“无连接的”，因此每次接收时前并不知道消息来自何处，通过 fromAddr 参数可以得到发送方的信息  
(主机、端口号)  
retval = recvfrom(sockfd, &response, sizeof(response), 0, (struct sockaddr *)&fromAddr, &fromLen);  
if (retval <= 0) {  
    printf("recvfrom failed or abort, %ld, %d!\r\n", retval, errno);  
    goto do_cleanup;  
}  
response[retval] = '\0';  
printf("recv UDP message {%s} %ld done!\r\n", response, retval);  
printf("peer info: ipaddr = %s, port = %d\r\n", inet_ntoa(fromAddr.sin_addr), ntohs(fromAddr.sin_port));  
  
do_cleanup:  
    printf("do_cleanup...\r\n");  
    lwip_close(sockfd);  
}
```

本节小结

本讲所学知识点有：

- 示例代码1：TCP服务端
- 示例代码2：TCP客户端
- 示例代码3：UDP服务端
- 示例代码4：UDP客户端

本章总结

本章所学内容有：

- 第1节：lwIP概述
- 第2节：lwIP功能特性
- 第3节：lwIP开发指引
- 第4节：lwIP示例代码

任务挑战

挑战任务1：参考第4节内容，完成并测试示例3、4

```
Serial-COM4 x
netifapi_dhcp_start: 0

server :

    server_id : 192.168.0.1
    mask : 255.255.255.0, 1
    gw : 192.168.0.1
    T0 : 7200
    T1 : 3600
    T2 : 6300

clients <1> :

    mac_idx mac          addr          state  lease  tries  rto
    0      b4c9b9e2e2e6  192.168.0.107  10     0      1      2

netifapi_netif_common: 0

begin demo
send UDP message {Hello. I am from chinasoft} 27 done!
recv UDP message {Hello. I am from chinasoft} 27 done!

peer info: ipaddr = 192.168.0.104, port = 50002

do_cleanup...
```

IPPOP V4.1

IP绑定 路由 MAC信息 网络统计 端口信息 端口映射 网卡流量 报文捕获 终端工具 服务 报文发送

服务端

监听端口: 50002 Stop ☒ 回送 ☒ AutoStart

Remote IP	Remote Port	time	Receive Bytes	
<input type="checkbox"/> 192.168.0.107	61979	2020/11/23 16:2...	27	Hello. I am from chinasoft

2020/11/23 16:24:18->接收客户端:192.168.0.107:61979发送的数据。

客户端

服务器地址: 192.168.0.104 发送字符串: test 发送数: 10 Start

服务器端口: 50002 起始源端口: 0 发速率: 10 次/秒 ☐ 指定长度: 10

Local IP	Local Port	time	Send Bytes	
<input type="checkbox"/> localhost	0	2020/11/23 16:2...	4	test
<input type="checkbox"/> localhost	0	2020/11/23 16:2...	4	test
<input type="checkbox"/> localhost	0	2020/11/23 16:2...	4	test
<input type="checkbox"/> localhost	0	2020/11/23 16:2...	4	test

2020/11/23 16:20:53->接收服务端:192.168.0.104:50002发送的数据。

2020/11/23 16:20:53->接收服务端:192.168.0.104:50002发送的数据。

2020/11/23 16:20:54->接收服务端:192.168.0.104:50002发送的数据。

2020/11/23 16:20:54->接收服务端:192.168.0.104:50002发送的数据。

ready Serial: COM4, 115200 41, 1 41 Rows, 86 Cols Xterm

THANKS

更多学习视频，关注宅客学院.....

