XR806 noPoll 网络模块

开发指南

版本号：1.0

发布时间：2020-11-12

版本历史

版本	日期	责任人	版本描述
1.0	2020-11-12	AWA 1096	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 平台上 noPoll 模块的使用方法。

1.2 目标读者

XR806 开发及维护人员

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品

1.4 文档约定

1.4.1 标志说明

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

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

2 概述

2.1 背景说明

noPoll 是一个 WebSocket 的开源实现，可用于构建纯 WebSocket 解决方案和为已有的面向 TCP 的应用程序提供 WebSocket 支持。

noPoll 相对于其他 WebSocket 实现（比如 libwebsocket），具有更快的速度和更小内存占用的优点。

2.2 规格特性

noPoll 模块提供了以下功能。

表 2-1 noPoll 模块的功能特性

规格类型	支持规格	规格描述	备注
功能	WebSocket(ws://)	基本的 WebSocket 功能	
	TLS WebSocket(wss://)	加密模式下的 WebSocket 功能	

2.3 文件位置

以 SDK 包为根目录，本网络协议涉及到的主要文件位置如下。

表 2-2 noPoll 网络协议的文件位置

组件名	文件分类	文件位置
Mbed TLS	源码文件	sdk/src/net/nopoll
	头文件	sdk/include/net/nopoll/
	示例工程	sdk/project/example/nopoll

说明

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

3 应用说明

3.1 应用简述

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

3.2 配置说明

表 3-1 XR806 noPoll 模块配置列表

配置项	配置说明
调试打印接口	设置说明： 此项配置使能/关闭调试信息 设置位置： sdk/include/net/nopoll/nopoll_log.h，及应用代码 设置方式： 1. 定义宏 SHOW_DEBUG_LOG 的值，如打开调试信息： #define SHOW_DEBUG_LOG (1) 2. 应用代码需要调用函数 nopoll_log_set_handler 来设置 log 的处理函数，或调用函数 nopoll_log_enable 来使用 noPoll 默认的 log 处理函数。函数说明请参阅 3.3.2 节调试配置接口说明
Pong 包接收	设置说明： 此项配置不是 noPoll 原生代码配置，是 XR806 SDK 实现的配置项。配置应用层是否接收服务器响应的 Pong 包 设置位置： sdk/src/net/nopoll/src/nopoll_conn.c 设置方式： 1. 定义宏 NOPOLL_DELIVER_PONG_FRAME 的值，如启动接收： #define NOPOLL_DELIVER_PONG_FRAME 1

3.3 接口说明

noPoll 接口过多，本文只列举了一些常用的接口，更加详细的接口说明可参考 noPoll 官方资料：
http://www.aspl.es/nopoll/html/index.html

表 3-2 noPoll 模块常用接口简介

接口名	简要介绍
创建接口	本组接口用于创建 noPoll 上下文，属于外部接口。

接口名	简要介绍
连接相关接口	本组接口用于创建 WebSocket 连接，属于外部接口。
发送接口	本组接口用于发送消息，属于外部接口
接收接口	本组接口用于获取消息，属于外部接口
断开连接接口	本组接口用于断开连接，属于外部接口
销毁接口	本组接口用于销毁资源，属于外部接口

3.3.1 创建接口

3.3.1.1 nopoll_ctx_new

表 3-3 nopoll_ctx_new 接口函数说明

信息项	说明
原型	noPollCtx* nopoll_ctx_new(void);
功能	创建 noPollCtx 句柄
参数	无
返回值	noPollCtx 类型指针

3.3.1.2 nopoll_conn_opts_new

表 3-4 nopoll_conn_opts_new 接口函数说明

信息项	说明
原型	noPollConnOpts * nopoll_conn_opts_new (void);
功能	创建 noPollConnOpts 操作句柄
参数	无
返回值	noPollConnOpts 类型指针

3.3.2 调试配置接口

3.3.2.1 nopoll_log_enable

表 3-5 nopoll_log_enable 接口函数说明

信息项	说明
原型	void nopoll_log_enable(noPollCtx *ctx, nopoll_bool value);
功能	使能/失能 log 打印信息
参数	noPollCtx *ctx 含义解释：noPollCtx 句柄。 使用说明：nopoll_ctx_new 创建的会话句柄。 nopoll_bool value 含义解释：使能/失能 log 打印信息。

信息项	说明
	使用说明：1：使能，0：失能。
返回值	无

3.3.2.2 nopoll_log_set_handler

表 3-6 nopoll_log_set_handler 接口函数说明

信息项	说明
原型	void nopoll_log_set_handler(noPollCtx *ctx, noPollLogHandler handler, noPollPtr user_data);
功能	设置 log 处理函数
参数	noPollCtx *ctx 含义解释：noPollCtx 句柄。 使用说明：nopoll_ctx_new 创建的会话句柄。 noPollLogHandler handler 含义解释：设置的回调函数，用于 log 处理。 使用说明：赋值即可。 noPollPtr user_data 含义解释：回调函数的用户数据。 使用说明：赋值即可。
返回值	无

3.3.3 连接相关接口

3.3.3.1 nopoll_conn_new_opts

表 3-7 nopoll_conn_new_opts 接口函数说明

信息项	说明
原型	noPollConn* nopoll_conn_new_opts(noPollCtx *ctx, noPollConnOpts *opts, const char *host_ip, const char *host_port, const char *host_name, const char *get_url, const char *protocols, const char *origin);
功能	创建一个新的连接
参数	noPollCtx *ctx 含义解释：noPollCtx 句柄。 使用说明：nopoll_ctx_new 创建的会话句柄。 noPollConnOpts *opts 含义解释：noPollConnOpts 操作句柄。 使用说明：nopoll_conn_opts_new 创建的会话句柄。 const char *host_ip 含义解释：服务器 ip 地址。 使用说明：赋值即可。

信息项	说明
	const char *host_port 含义解释：服务器端口号。 使用说明：赋值即可。 const char *host_name 含义解释：将要发送的 host 标头值，即 HTTP 头部信息中的 host。 使用说明：赋值即可，如为 NULL，则使用 host_ip 值。 const char *get_url 含义解释：url 的路径。 使用说明：赋值即可，如为 NULL，则默认使用”/”。 const char *protocols 含义解释：此连接的可选协议。 使用说明：赋值即可。 const char *origin 含义解释：发送到服务器的 origin 标头值，即 HTTP 头部信息中的 origin。 使用说明：赋值即可如果为 NULL，则使用（"http://%s", host_name）
返回值	noPollConn 类型指针

3.3.3.2 nopoll_conn_opts_set_ssl_certs

表 3-8 nopoll_conn_opts_set_ssl_certs 接口函数说明

信息项	说明
原型	nopoll_bool nopoll_conn_opts_set_ssl_certs(noPollConnOpts *opts, const char *certificate, int certificate_size, const char *private_key, int private_key_size, const char *chain_certificate, int chain_certificate_size, const char *ca_certificate, int ca_certificate_size);
功能	设置 Mbed TLS 证书，用于 TLS 连接
参数	noPollConnOpts *opts 含义解释：noPollConnOpts 操作句柄。 使用说明：nopoll_conn_opts_new 创建的会话句柄。 const char *certificate 含义解释：自己的证书。 使用说明：赋值即可。 int certificate_size 含义解释：证书长度。 使用说明：赋值即可。 const char *private_key 含义解释：自己的私钥。 使用说明：赋值即可。

信息项	说明
	int private_key_size 含义解释：私钥长度。 使用说明：赋值即可。 const char *chain_certificate 含义解释：自己的证书链。 使用说明：赋值即可。 int chain_certificate_size 含义解释：证书链的长度。 使用说明：赋值即可。 const char *ca_certificate 含义解释：CA 证书。 使用说明：赋值即可。 int ca_certificate_size 含义解释：CA 证书的长度。 使用说明：赋值即可。
返回值	1：成功 0：失败

3.3.3.3 nopoll_conn_opts_ssl_peer_verify

表 3-9 nopoll_conn_opts_ssl_peer_verify 接口函数说明

信息项	说明
原型	void nopoll_conn_opts_ssl_peer_verify(noPollConnOpts *opts, nopoll_bool verify);
功能	使能/失能对端证书验证，用于 TLS 连接
参数	noPollConnOpts *opts 含义解释：noPollConnOpts 操作句柄。 使用说明：nopoll_conn_opts_new 创建的会话句柄。 nopoll_bool verify 含义解释：使能/失能。 使用说明：0：失能验证，即 TLS 握手时不验证对端证书；1：使能验证，即验证对端证书。
返回值	无

3.3.3.4 nopoll_conn_tls_new

表 3-10 nopoll_conn_tls_new 接口函数说明

信息项	说明
原型	noPollConn* nopoll_conn_tls_new(noPollCtx *ctx, noPollConnOpts *options, const char *host_ip, const char *host_port, const char *host_name, const char *get_url,

信息项	说明
	const char *protocols, const char *origin);
功能	创建一个新的 TLS 连接
参数	noPollCtx *ctx 含义解释：noPollCtx 句柄。 使用说明：nopoll_ctx_new 创建的会话句柄。 noPollConnOpts *options 含义解释：noPollConnOpts 操作句柄。 使用说明：nopoll_conn_opts_new 创建的会话句柄。 const char *host_ip 含义解释：服务器 ip 地址。 使用说明：赋值即可。 const char *host_port 含义解释：服务器端口号。 使用说明：赋值即可。 const char *host_name 含义解释：将要发送的 host 标头值，即 HTTP 头部信息中的 host。 使用说明：赋值即可，如为 NULL，则使用 host_ip 值。 const char *get_url 含义解释：url 的路径。 使用说明：赋值即可，如为 NULL，则默认使用”/”。 const char *protocols 含义解释：此连接的可选协议。 使用说明：赋值即可。 const char *origin 含义解释：发送到服务器的 origin 标头值，即 HTTP 头部信息中的 origin。 使用说明：赋值即可如果为 NULL，则使用（"http://%s", host_name）
返回值	noPollConn 类型指针

3.3.3.5 nopoll_conn_wait_until_connection_ready

表 3-11 nopoll_conn_wait_until_connection_ready 接口函数说明

信息项	说明
原型	nopoll_bool nopoll_conn_wait_until_connection_ready(noPollConn *conn, int timeout);
功能	等待完成连接
参数	noPollConn *conn 含义解释：noPollConn 句柄。

信息项	说明
	使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。 int timeout 含义解释：等待超时时间。 使用说明：赋值即可，单位为秒。
返回值	1：成功 0：连接失败或超时

3.3.4 发送接口

3.3.4.1 nopoll_conn_send_ping

表 3-12 nopoll_conn_send_ping 接口函数说明

信息项	说明
原型	nopoll_bool nopoll_conn_send_ping(noPollConn *conn);
功能	发送一次 Ping 包到服务器
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	0：失败 1：成功

3.3.4.2 nopoll_conn_send_text

表 3-13 nopoll_conn_send_text 接口函数说明

信息项	说明
原型	int nopoll_conn_send_text(noPollConn *conn, const char *content, long length);
功能	发送一段完整的字符串数据，同时表明这是该段字符串数据的最后一笔数据
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。 const char *content 含义解释：字符串数据。 使用说明：赋值即可 long length 含义解释：数据的长度。 使用说明：赋值即可
返回值	发送的字节数

3.3.4.3 nopoll_conn_send_text_fragment

表 3-14 nopoll_conn_send_text_fragment 接口函数说明

信息项	说明
原型	int nopoll_conn_send_text_fragment(noPollConn *conn, const char *content, long length);
功能	发送字符串数据的一部分。该接口表示发送的数据不是最后一笔数据。如果有一笔很大的数据，需要分片发送，则循环调用该接口，最后一笔数据则调用 nopoll_conn_send_text
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。 const char *content 含义解释：字符串数据。 使用说明：赋值即可 long length 含义解释：数据的长度。 使用说明：赋值即可
返回值	发送的字节数

3.3.4.4 nopoll_conn_send_binary

表 3-15 nopoll_conn_send_binary 接口函数说明

信息项	说明
原型	int nopoll_conn_send_binary(noPollConn *conn, const char *content, long length);
功能	发送一段完整的二进制类型数据，同时表明这是该段数据的最后一笔数据
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。 const char *content 含义解释：二进制数据。 使用说明：赋值即可 long length 含义解释：数据的长度。 使用说明：赋值即可
返回值	发送的字节数

3.3.4.5 nopoll_conn_send_binary_fragment

表 3-16 nopoll_conn_send_binary_fragment 接口函数说明

信息项	说明
原型	int nopoll_conn_send_binary_fragment(noPollConn *conn, const char *content, long length);
功能	发送二进制数据的一部分。该接口表示发送的数据不是最后一笔数据。如果有一笔很大的数据，需要分片发送，则循环调用该接口，最后一笔数据则调用 nopoll_conn_send_binary
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。 const char *content 含义解释：二进制数据。 使用说明：赋值即可 long length 含义解释：数据的长度。 使用说明：赋值即可
返回值	发送的字节数

3.3.4.6 nopoll_conn_complete_pending_write

表 3-17 nopoll_conn_complete_pending_write 接口函数说明

信息项	说明
原型	int nopoll_conn_complete_pending_write(noPollConn *conn);
功能	完成上一次未完成的发送操作。当发送一笔数据时，可能一次性发送不完，所以在下一次发送操作时，需要调用该接口，完成上一次未发送完成的数据
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	发送的字节数

3.3.4.7 nopoll_conn_pending_write_bytes

表 3-18 nopoll_conn_pending_write_bytes 接口函数说明

信息项	说明
原型	int nopoll_conn_pending_write_bytes(noPollConn *conn);
功能	检查挂起的字节数，在发送数据前，可以调用该接口检查上一次发送是否发送完全，如果没有发送完全，则调用 nopoll_conn_complete_pending_write 接口发送
参数	noPollConn *conn 含义解释：noPollConn 句柄。

信息项	说明
	使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	挂起的字节数

3.3.5 接收接口

3.3.5.1 nopoll_conn_is_ok

表 3-19 nopoll_conn_is_ok 接口函数说明

信息项	说明
原型	nopoll_bool nopoll_conn_is_ok(noPollConn *conn);
功能	检查提供的连接是否处于连接状态。一般接收数据前，都会判断当前是否还处于连接状态
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	0：已断开连接 1：已连接

3.3.5.2 nopoll_conn_get_msg

表 3-20 nopoll_conn_get_msg 接口函数说明

信息项	说明
原型	noPollMsg* nopoll_conn_get_msg(noPollConn *conn);
功能	获取消息，该接口是非阻塞的，所以有消息时，会返回消息，没有消息时会返回 NULL
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	消息指针

3.3.5.3 nopoll_msg_get_payload_size

表 3-21 nopoll_msg_get_payload_size 接口函数说明

信息项	说明
原型	int nopoll_msg_get_payload_size(noPollMsg *msg);
功能	获取消息里面 payload 的长度
参数	noPollMsg *msg 含义解释：消息指针。 使用说明：nopoll_conn_get_msg 获取到的消息。
返回值	payload 的长度

3.3.5.4 nopoll_msg_opcode

表 3-22 nopoll_msg_opcode 接口函数说明

信息项	说明
原型	noPollOpCode nopoll_msg_opcode(noPollMsg *msg);
功能	获取消息类型
参数	noPollMsg *msg 含义解释：消息指针。 使用说明：nopoll_conn_get_msg 获取到的消息。
返回值	消息类型

3.3.5.5 nopoll_msg_is_final

表 3-23 nopoll_msg_is_final 接口函数说明

信息项	说明
原型	nopoll_bool nopoll_msg_is_final(noPollMsg *msg);
功能	获取该消息是否是最后一笔数据
参数	noPollMsg *msg 含义解释：消息指针。 使用说明：nopoll_conn_get_msg 获取到的消息。
返回值	0：不是最后一笔数据 1：最后一笔数据

3.3.5.6 nopoll_msg_get_payload

表 3-24 nopoll_msg_get_payload 接口函数说明

信息项	说明
原型	const unsigned char *nopoll_msg_get_payload(noPollMsg *msg);
功能	获取消息里面的 payload
参数	noPollMsg *msg 含义解释：消息指针。 使用说明：nopoll_conn_get_msg 获取到的消息。
返回值	payload 的指针

3.3.6 断开连接接口

3.3.6.1 nopoll_conn_close

表 3-25 nopoll_conn_close 接口函数说明

信息项	说明
原型	void nopoll_conn_close(noPollConn *conn);
功能	断开连接

信息项	说明
参数	noPollConn *conn 含义解释：noPollConn 句柄。 使用说明：nopoll_conn_new_opts 或 nopoll_conn_tls_new 创建的会话句柄。
返回值	无

3.3.7 销毁接口

3.3.7.1 nopoll_conn_opts_free

表 3-26 nopoll_conn_opts_free 接口函数说明

信息项	说明
原型	void nopoll_conn_opts_free(noPollConnOpts *opts);
功能	释放 noPollConnOpts 操作句柄
参数	noPollConnOpts *opts 含义解释：noPollConnOpts 操作句柄。 使用说明：nopoll_conn_opts_new 创建的会话句柄。
返回值	无

3.3.7.2 nopoll_ctx_unref

表 3-27 nopoll_ctx_unref 接口函数说明

信息项	说明
原型	void nopoll_ctx_unref(noPollCtx *ctx);
功能	释放 noPollCtx 句柄
参数	noPollCtx *ctx 含义解释：noPollCtx 句柄。 使用说明：nopoll_ctx_new 创建的会话句柄。
返回值	无

4 示例说明

noPoll 接口用于进行 WebSocket 连接通信，下面以 WebSocket 连接通信为例，简要说明 noPoll 接口的使用。

4.1 示例简介

noPoll 示例的演示的目的是简要介绍 noPoll 接口的基本使用方法，演示的功能为连接服务器，并进行数据发送与接收。

4.1.1 获取方式

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

说明

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

4.1.2 准备工作

noPoll 示例工程的硬件准备如下。

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

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

4.1.3 操作步骤

开发板完成烧写，复位，通过串口命令连接网络后，示例代码自动运行。

表 4-1 noPoll 示例工程的命令列表

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

命令	说明
	net sta enable 命令解释： 开始连接 AP，需要先完成网络配置

4.2 效果展示

在评估板中运行 noPoll 示例程序后，在控制台中会打印出连接的各个阶段，并显示数据接收过程，如下所示。

```
host ip:174.129.224.73, host port:80
the connection is ready
send data to echo.websocket.org: nopoll example test
rece data from echo.websocket.org: nopoll example test
host ip:174.129.224.73, host port:443
the connection is ready
send data to echo.websocket.org: nopoll example test
rece data from echo.websocket.org: nopoll example test
```

4.3 实现流程

第一步：初始化 noPoll

```
nopoll_client_ctx = nopoll_ctx_new();
if (nopoll_client_ctx == NULL) {
        printf("nopoll_ctx_new failed\n");
        goto exit;
}
nopoll_conn_opts = nopoll_conn_opts_new();
if (nopoll_conn_opts == NULL) {
        printf("nopoll_conn_opts_new failed\n");
        goto exit;
}
```

第二步：连接服务器

```
nopoll_conn = nopoll_conn_new_opts(nopoll_client_ctx, nopoll_conn_opts, host_ip, host_port, host_name, get_url, NULL, NULL);
/* wait 10s until the connection ready */
if (nopoll_conn_wait_until_connection_ready(nopoll_conn, 10))
        printf("the connection is ready\n");
else {
```

```
        printf("connection timeout\n");
        goto exit;
}
```

如果使用 TLS 连接，则使用下面的函数接口：

```
if (!nopoll_conn_opts_set_ssl_certs(nopoll_conn_opts, NULL, 0, NULL, 0, NULL, 0, websocket_demo_ca, strlen(websocket_demo_ca) + 1)) {
        printf("nopoll_conn_opts_set_ssl_certs failed\n");
        goto exit;
}
/* set ssl verfy */
nopoll_conn_opts_ssl_peer_verify(nopoll_conn_opts, nopoll_true);
nopoll_conn = nopoll_conn_tls_new(nopoll_client_ctx, nopoll_conn_opts, host_ip, host_port, host_name, get_url, NULL, NULL);/* wait 10s until the connection ready    */
if (nopoll_conn_wait_until_connection_ready(nopoll_conn, 10))
        printf("the connection is ready\n");
else {
        printf("connection timeout\n");
        goto exit;
}
```

第三步： 发送数据

```
#define WEBSOCKET_DEMO_TEXT "websocket demo test"

static int nopoll_complete_pending_write(noPollConn *conn)
{
        int tries = 0;
        while (tries < 5 && errno == NOPOLL_EWOULDBLOCK &&
                        nopoll_conn_pending_write_bytes (conn) > 0) {
                nopoll_sleep(50000);
                if (nopoll_conn_complete_pending_write (conn) == 0)
                        return 0;
                tries++;
        }
        return 1;
}

int len = strlen(WEBSOCKET_DEMO_TEXT);
ret = nopoll_conn_send_text(nopoll_conn, WEBSOCKET_DEMO_TEXT, len);
```

```
if (ret != len ) {
        /* if the actual data sent and the data want sent are not equal */
        if (nopoll_complete_pending_write(nopoll_conn))
                printf("size = %u, but nopoll_conn_send_text ret = %d\n", len, ret);
}
```

第四步： 接收数据

```
noPollMsg *msg;
const char *content;
int times = 0;
while (1) {
        if (!nopoll_conn_is_ok(nopoll_conn)) {
                printf("received websocket connection close\n");
                break;
        }
        msg = nopoll_conn_get_msg(nopoll_conn);
        if (msg) {
                content = (const char *)nopoll_msg_get_payload(msg);
                printf("rece data from %s: %s\n", host_name, content);
                nopoll_msg_unref(msg);
                break;
        } else {
                nopoll_sleep(100000); // 1s
                times++;
                if (times > 10) // try 10 times
                        break;
        }
}
```

第五步：断开连接

```
if (nopoll_conn != NULL) {
        nopoll_conn_close(nopoll_conn);
        nopoll_conn = NULL;
}
```

第六步：释放资源

```
if (nopoll_conn_opts != NULL) {
        nopoll_conn_opts_free(nopoll_conn_opts);
        nopoll_conn_opts = NULL;
```

```
}
if (nopoll_client_ctx != NULL) {
        nopoll_ctx_unref(nopoll_client_ctx);
        nopoll_client_ctx = NULL;
}
```

著作权声明

商标声明

免责声明