XRADIO noPoll Developer Guide

Revision 1.0

Oct 22, 2019

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

Revision History

Version	Date	Summary of Changes
1.0	2019-10-22	Initial Version

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

Contents

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

1 模块概要

1.1 功能介绍

noPoll 是一个 WebSocket 的开源实现，使用 ANSI C 编写，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 支持 WebSocket(ws://)和 TLS WebSocket(wss://)，从而允许基于消息的（通知处理程序）编程或面向流的访问。

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。

1.2 代码位置

模块	文件类型	位置
noPoll	source	sdk/src/net/nopoll
	header	sdk/include/net/nopoll/
	demo	sdk/project/common/cmd/cmd_nopoll.c
		sdk/project/example/websocket

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

2 noPoll 接口

noPoll 即可以作为客户端模式，又客户作为服务器模式，大多数情况下都是作为客户端模式，服务器模式几乎不用，故本文只描述客户端模式下的接口。

2.1 客户端接口

由于 noPoll 接口太多，很多接口都是用不到的，故本文列举了一些常用的接口，具体的接口可参考 noPoll 官方资料：http://www.aspl.es/nopoll/html/index.html

2.1.1 context 操作接口

noPollCtx* nopoll_ctx_new(void)	
功能	创建一个空的 noPoll 上下文。
参数	void
返回值	noPoll 上下文指针

nopoll_bool nopoll_ctx_ref (noPollCtx * ctx)	
功能	获取对所提供上下文的引用。
参数	ctx：要获取引用的上下文。
返回值	0：失败，1：成功

void nopoll_ctx_ref (noPollCtx * ctx)	
功能	释放对提供上下文的引用。 该函数和 nopoll_ctx_ref 对应使用，该函数还起到 free 资源的功能，当一个资源的引用为 0 时，noPoll 会自动释放该资源。noPoll 不存在 nopoll_ctx_free 接口，故需要用该接口释放。
参数	ctx：要释放引用的上下文。
返回值	void

int nopoll_ctx_ref_count (noPollCtx *ctx)	
功能	获取所提供上下文的当前引用计数。

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

参数	ctx：要获取引用计数的上下文。
返回值	引用计数

2.1.2 打印控制接口

void nopoll_log_enable(noPollCtx *ctx, nopoll_bool value)	
功能	使能打印。
参数	ctx：要使能打印的上下文。 value：1：使能，0：失能
返回值	void

2.1.3 连接操作接口

noPollConn* nopoll_conn_new(noPollCtx *ctx, 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	ctx：要创建连接的 noPoll 上下文 host_ip：服务器地址 host_port：服务器端口号，默认为 80 host_name：将要发送的 host 标头值，即 http 头部信息中的 Host，如果为 NULL，则使用 host_ip 值 get_url：url 的路径，作为握手的一部分，该 url 会被传递到服务器中，如果未设置，则默认使用“/” protocols：此连接的可选协议（用空格分隔的字符串列表） origin：发送到服务器的 origin 标头值，即 http 头部信息中的 Origin，如果为空，则使用（"http://%s", host_name）
返回值	连接对象的指针，NULL 表示创建失败
备注	该函数返回后，并不代表已经连接到了服务器。

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)

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

功能	按照传入的 noPollConnOpts 对象创建一个新的连接
参数	ctx：要创建连接的 noPoll 上下文 opts：连接期间要使用的连接选项 host_ip：服务器地址 host_port：服务器端口号，默认为 80 host_name：将要发送的 host 标头值，即 http 头部信息中的 Host，如果为 NULL，则使用 host_ip 值 get_url：url 的路径，作为握手的一部分，该 url 会被传递到服务器中，如果未设置，则默认使用“/” protocols：此连接的可选协议（用空格分隔的字符串列表） origin：发送到服务器的 origin 标头值，即 http 头部信息中的 Origin，如果为空，则使用（"http://%s", host_name）
返回值	连接对象的指针，NULL 表示创建失败
备注	该函数和 nopoll_conn_new 功能是相同的，只是该函数多了个连接选项，其他的都一样，

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, const char *protocols, const char *origin)	
功能	创建一个 TLS 连接
参数	ctx：要创建连接的 noPoll 上下文 opts：连接期间要使用的连接选项 host_ip：服务器地址 host_port：服务器端口号，默认为 80 host_name：将要发送的 host 标头值，即 http 头部信息中的 Host，如果为 NULL，则使用 host_ip 值 get_url：url 的路径，作为握手的一部分，该 url 会被传递到服务器中，如果未设置，则默认使用“/” protocols：此连接的可选协议（用空格分隔的字符串列表） origin：发送到服务器的 origin 标头值，即 http 头部信息中的 Origin，如果为空，则使用（"http://%s", host_name）
返回值	连接对象的指针，NULL 表示创建失败
备注	该接口和 nopoll_conn_new_opts 接口类似，只是该接口是创建 TLS 连接，该函数返回后，并不

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

	代表已经连接到了服务器。

void nopoll_conn_close(noPollConn *conn)	
功能	关闭一个连接，无论是客户端模式还是服务器模式，都用该接口关闭。
参数	conn：需要关闭的连接。
返回值	void

int nopoll_conn_complete_pending_write(noPollConn *conn)	
功能	完成上一个未完成的发送操作。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	conn：需要发送的连接。如果 conn 为 NULL，则函数会返回 0
返回值	发送的数据，-1 表示发送失败

int nopoll_conn_flush_writes(noPollConn *conn, long timeout, int previous_result)	
功能	用于检查挂起的写操作，并刷新它们，将挂起的写操作数据发送出去，直到数据发送完成或超时为止。该函数使用 nopoll_conn_pending_write_bytes 和 nopoll_conn_complete_pending_write 来检查并完成挂起的写操作，该接口是将 nopoll_conn_pending_write_bytes 和 nopoll_conn_complete_pending_write 进行封装了一下。 由于写入挂起的数据是一种常见操作，因此该函数可在写操作后调用，将剩余的数据发送出去
参数	conn：要操作的连接 timeout：超时时间，微秒 previous_result：可选参数，由于该接口可能需要多次调用，每调用一次，都会返回实际发送出去的数据个数，将上一次发送出去的数据个数传递到该参数中，则上一次发送的数据个数将被添加到本次结果中，所以只要按照以上的操作方式，该接口返回的是发送出去的总数据个数。这种逻辑设计的很不合理，故一般情况下设置该参数设置为 0，发送出去的总数据个数需要用户自己在外部累加。
返回值	发送出去的数据个数

void nopoll_conn_connect_timeout(noPollCtx *ctx, long microseconds_to_wait)	
功能	配置 noPoll 连接超时时间。此接口允许设置 nopoll_conn_new 使用的 TCP 连接超时时间。
参数	ctx：将要执行操作的上下文。

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

	microseconds_to_wait：超时时间，微秒为单位，默认值为 20000000 微秒（20s）
返回值	void

noPollCtx *nopoll_conn_ctx(noPollConn *conn)	
功能	获取某个连接的 noPoll 上下文
参数	conn：返回其上下文的连接
返回值	连接的上下文

const char* nopoll_conn_get_accepted_protocol(noPollConn *conn)	
功能	获取连接所支持的协议
参数	conn：要操作的连接
返回值	支持的协议，如果在握手期间或者握手未完成，则返回 NULL

const char* nopoll_conn_get_close_reason(noPollConn *conn)	
功能	获取连接关闭的原因
参数	conn：要操作的连接
返回值	远程服务器报告的原因，关闭原因；如果未报告，则为 NULL。

int nopoll_conn_get_close_status(noPollConn *conn)	
功能	获取连接关闭的状态码
参数	conn：要操作的连接
返回值	远程服务器报告的状态码，关闭原因；0：conn 指针为 NULL，1005：关闭帧，1006：无关闭帧

const char* nopoll_conn_get_cookie(noPollConn *conn)	
功能	获取握手期间接收到的 Cookie 标头内容（如果已收到）。
参数	conn：要操作的连接
返回值	cookie 指针，如果未收到则为 NULL

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

const char* nopoll_conn_get_host_header(noPollConn *conn)	
功能	获取握手期间接收到的 host 标头内容（如果已收到）。
参数	conn：要操作的连接
返回值	host 指针，如果未收到则为 NULL

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	conn：要操作的连接
返回值	消息指针

const char* nopoll_conn_get_origin(noPollConn *conn)	
功能	获取接收到的连接 Origin 头内容
参数	conn：要操作的连接
返回值	Origin 指针

const char* nopoll_conn_host(noPollConn *conn)	
功能	获取连接的主机
参数	conn：要操作的连接
返回值	主机指针

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
参数	conn：要操作的连接
返回值	0：断开连接，1：已连接

nopoll_bool nopoll_conn_is_ready(noPollConn *conn)	
功能	检查提供的连接是否处于 ready 状态，即握手完成
参数	conn：要操作的连接

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

返回值	0：未握手，1：已握手

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	conn：要操作的连接
返回值	0：未握手，1：已握手

int nopoll_conn_send_binary(noPollConn *conn, const char *content, long length)	
功能	发送二进制数据（op 码为 2），同时也表示是最后一笔数据
参数	conn：要操作的连接 content：二进制数据 length：数据长度，该接口长度不能为-1
返回值	发送的字节数，如果小于 0，则表示发送失败

int nopoll_conn_send_binary_fragment(noPollConn *conn, const char *content, long length)	
功能	发送二进制数据（op 码为 2），但是该接口表示发送的数据不是最后一笔数据，即 FIN = 0。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	conn：要操作的连接 content：二进制数据 length：数据长度，该接口长度不能为-1
返回值	发送的字节数，如果小于 0，则表示发送失败

nopoll_bool nopoll_conn_send_ping(noPollConn *conn)	
功能	发送一次 ping 包到服务器
参数	conn：要操作的连接
返回值	0：失败，1：成功

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

int nopoll_conn_send_text(noPollConn *conn, const char *content, long length)	
功能	发送字符串数据（op 码为 1），同时也表示是最后一笔数据
参数	conn：要操作的连接 content：字符串数据 length：数据长度
返回值	发送的字节数，如果小于 0，则表示发送失败

int nopoll_conn_send_text_fragment(noPollConn *conn, const char *content, long length)	
功能	发送字符串数据（op 码为 1），但是该接口表示发送的数据不是最后一笔数据，即 FIN = 0。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	conn：要操作的连接 content：字符串数据 length：数据长度
返回值	发送的字节数，如果小于 0，则表示发送失败

void nopoll_conn_set_on_close(noPollConn *conn, noPollOnCloseHandler on_close, noPollPtr user_data)	
功能	配置关闭连接时调用的通知回调函数
参数	conn：要操作的连接 on_close：回调函数 user_data：用户数据，会传递到回调函数中
返回值	void

void nopoll_conn_set_on_msg(noPollConn *conn, noPollOnMessageHandler on_msg, noPollPtr user_data)	
功能	配置接收到数据时调用的通知回调函数
参数	conn：要操作的连接 on_msg：回调函数 user_data：用户数据，会传递到回调函数中
返回值	void

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

void nopoll_conn_set_on_ready(noPollConn *conn, noPollActionHandler on_ready, noPollPtr user_data)	
功能	配置连接 ready 或者握手完成后调用的通知回调函数
参数	conn：要操作的连接 on_ready：回调函数 user_data：用户数据，会传递到回调函数中
返回值	void

nopoll_bool nopoll_conn_set_sock_block(NOPOLL_SOCKET socket, nopoll_bool enable)	
功能	启用/禁用 非阻止/阻止 socket
参数	socket：socket 参数，用 nopoll_conn_socket 接口获取 enable：1：阻塞，0：非阻塞
返回值	1：成功，0：失败

NOPOLL_SOCKET nopoll_conn_socket(noPollConn *conn)	
功能	获取连接对象的 socket
参数	conn：要操作的连接
返回值	连接对象的 socket，-1 为无效

nopoll_bool nopoll_conn_wait_until_connection_ready(noPollConn *conn, int timeout)	
功能	等待完成连接
参数	conn：要操作的连接 timeout：超时时间，单位为秒
返回值	1：连接成功，0：连接失败或者连接超时

2.2 连接选项接口

noPollConnOpts* nopoll_conn_opts_new(void)	
功能	创建一个选项

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

参数	void
返回值	选项对象的指针

void nopoll_conn_opts_free(noPollConnOpts *opts)	
功能	释放连接选项
参数	opts：要操作的选项
返回值	void
备注	调用该接口，需要确定 opts 设置为可重用，因为 noPoll 的一些 api 在运行结果和预期结果不匹配时，会释放 opts。不过实际操作中，可不理会这个要求，因为代码里面有保护，不会造成重复 free 问题。

void nopoll_conn_opts_set_cookie(noPollConnOpts *opts, const char *cookie_content)	
功能	设置连接选项的 cookie，在握手过程中使用
参数	opts：要操作的选项 cookie_content：cookie 数据
返回值	void

void nopoll_conn_opts_set_extra_headers(noPollConnOpts *opts, const char *header_string)	
功能	设置连接选项的额外头部信息，设置的 http 头部选项，如果要设置额外的 http 头部选项，则调用该函数
参数	opts：要操作的选项 header_string：头部数据
返回值	void

void nopoll_conn_opts_set_reuse(noPollConnOpts *opts, nopoll_bool reuse)	
功能	设置连接选项的可重用标志。如果设置为可重用，则 noPoll 中的 api 在操作完后，比如连接完成后，就不会释放 opts。
参数	opts：要操作的选项 reuse：1：可重用，0：不可重用

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

返回值	void

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 TLS 连接的证书、私钥、证书链、CA 证书
参数	opts：要操作的选项 certificate：自己的证书，在双向验证中，发给服务器的证书 private_key：自己的私钥 chain_certificate：自己的证书链 ca_certificate：CA 证书，用来验证服务器发过来的证书
返回值	1：成功，0：失败

void nopoll_conn_opts_ssl_peer_verify (noPollConnOpts * opts, nopoll_bool verify)	
功能	失能 TLS 验证对端证书
参数	opts：要操作的选项 verify：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	void

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

3 模块接口例程

详细参考例程参考 sdk/project/example/websocket 或 sdk/project/common/cmd/cmd_nopoll.c。

3.1 初始化 noPoll

创建 noPoll 客户端 context：

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

创建连接选项：

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

3.2 连接服务器

创建连接：

```
非 TLS 连接：
nopoll_conn    =    nopoll_conn_new_opts(nopoll_client_ctx,    nopoll_conn_opts,
host_ip, host_port, host_name, get_url, NULL, NULL);
TLS 连接：
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);
```

等待连接完成：

```
/* 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;
```

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

```
}
```

3.3 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);
}
```

3.4 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;
    }
}
```

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

3.5 关闭连接释放资源

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

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved

4 参考资料

noPoll 的接口较多，本文只是列举了部接口，具体详细接口可参考以下资料：

官网	http://www.aspl.es/nopoll/index.html
文档中心	http://www.aspl.es/nopoll/html/index.html
核心库手册	http://www.aspl.es/nopoll/html/nopoll_core_library_manual.html
模块接口	http://www.aspl.es/nopoll/html/modules.html
GitHub	https://github.com/asples/nopoll

Copyright @2019 Xradio Technology Co., Ltd. All Rights Reserved