EC2x&AG35-QuecOpen
网络开发应用指导

LTE Standard/Automotive Module 系列

版本：EC2x&AG35-QuecOpen_网络开发_应用指导_V1.0

日期：2017-11-15

状态：临时文件

上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市徐汇区虹梅路 1801 号宏业大厦 7 楼　　邮编：200233
电话：+86 21 51086236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：
http://quectel.com/cn/support/sales.htm

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：
http://quectel.com/cn/support/technical.htm
或发送邮件至：support@quectel.com

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。由于客户操作不当而造成的人身伤害或财产损失，本公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

版权申明

本文档版权属于上海移远通信技术股份有限公司，任何人未经我司允许而复制转载该文档将承担法律责任。

版权所有 ©上海移远通信技术股份有限公司 2019，保留一切权利。
Copyright © Quectel Wireless Solutions Co., Ltd. 2019.

Quectel Confidential

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2017-11-15	钱润生	初始版本

Quectel Confidential

目录

Quectel Confidential

图片索引

Quectel Confidential

1 介绍

本文档针对不太熟悉 Linux 网络应用开发的人员，提供一些入门指导。涉及的内容包括 TCP/UDP Socket 套件字开发，OpenSSL 的应用开发，http(s)应用开发等几个部分。

文档中所有的 API 和库都是来自于 GNU 开源库，源码和文档都可以在网络上搜索到，不属于 Quectel 特有。

Quectel Confidential

2 TCP/UDP 套件字开发

TCP/UDP 协议都是 Linux 内核自己实现，对应协议的应用都是基本 Socket 系统调用接口去实现。

2.1. Socket 介绍

2.1.1. 套接字类型

套接字通常实现的就是传输层协议的应用接口，所以套接字 Socket 有三种类型：

流式套接字（SOCK_STREAM）

流式的套接字可以提供可靠的、面向连接的通讯流，它使用了 TCP 协议。TCP 保证了数据传输的正确性和顺序性。

数据报套接字（SOCK_DGRAM）

数据报套接字定义了一种无连接的服务，数据通过相互独立的报文进行传输，是无序的，并且不保证可靠、无差错，它使用数据报协议 UDP。

原始套接字

原始套接字允许对低层协议如 IP 或 ICMP 直接访问，主要用于新的网络协议的测试等。

2.1.2. 地址结构

网络编程有个重要参数是地址，对应 Socket 通信传入的协议地址类型，对应协议地址。

1. Socket 地址结构

所有 Socket 通信接口传入的地址参数都是：struct sockaddr 或者 struct sockaddr_storage。

```
//-------------------------SOCK ADDR(old)-----------------------------//
struct sockaddr {
    sa_family_t sa_family;
    char sa_data[14];                //protocol-specific address
};

//-------------------------SOCK ADDR(new)-----------------------------//
struct sockaddr_storage
{
     sa_family_t ss_family;
     __ss_aligntype __ss_align;   //Force desired alignment.
     char __ss_padding[_SS_PADSIZE];
```

Quectel Confidential

```
};
```

2. Socket IP 地址结构

IPV4 Socket addr 地址结构

```
//--------------------------IPV4-----------------------------//
struct sockaddr_in {
        sa_family_t sin_family;                 // AF_INET
        in_port_t sin_port;                     // Port number
        struct in_addr sin_addr;                // Internet address.

        // Pad to size of `struct sockaddr'.
        unsigned char sin_zero[sizeof (struct sockaddr) -
                               sizeof (sa_family_t) -
                               sizeof (in_port_t) -
                               sizeof (struct in_addr)];
};

typedef uint32_t in_addr_t;
struct in_addr   {
      in_addr_t s_addr;                         // IPv4 address
};

IPV6 Socket addr 地址结构
//--------------------------IPV6-----------------------------//
struct sockaddr_in6 {
      sa_family_t sin6_family;      // AF_INET6
      in_port_t sin6_port;          // Transport layer port #
      uint32_t sin6_flowinfo;       // IPv6 flow information
      struct in6_addr sin6_addr;    // IPv6 address
      uint32_t sin6_scope_id;       // IPv6 scope-id
};
struct in6_addr {
      union {
            uint8_t u6_addr8[16];
            uint16_t u6_addr16[8];
            uint32_t u6_addr32[4];
      } in6_u;

      #define s6_addr                 in6_u.u6_addr8
      #define s6_addr16               in6_u.u6_addr16
      #define s6_addr32               in6_u.u6_addr32
};
```

通常在进行网络地址编程的时候，首先**直接对 ip 套接字地址**结构 struct sockaddr_in/struct sockaddr_in6 操作，然后**强制转换为套接字地址** struct sockaddr 或者 struct sockaddr_storage

2.1.3. IP 地址转换

1. 把 ip 地址转化为用于网络传输的二进制数值

*int inet_aton(const char*cp, struct in_addr*inp);*

inet_aton() 转换网络主机地 ip(如 192.168.1.10)为二进制数值，并存储在 struct in_addr 结构中，即第二个参数*inp，函数返回非 0 表示 cp 主机有地有效，返回 0 表示主机地址无效。（这个转换完后不能用于网络传输，还需要调用 htons 或 htonl 函数才能将主机字节顺序转化为网络字节顺序）。

*in_addr_t inet_addr(const char*cp);*

inet_addr 函数转换网络主机地址（如 192.168.1.10)为网络字节序二进制值，如果参数 char*cp 无效，函数返回-1(INADDR_NONE)，这个函数在处理地址为 255.255.255.255 时也返回－1,255.255.255.255 是一个有效的地址，不过 inet_addr 无法处理。

2. 将网络传输的二进制数值转化为成点分十进制的 ip 地址

*char*inet_ntoa(struct in_addr in);*

inet_ntoa 函数转换网络字节排序的地址为标准的 ASCII 以点分开的地址,该函数返回指向点分开的字符串地址（如 192.168.1.10)的指针，该字符串的空间为静态分配的，这意味着在第二次调用该函数时，上一次调用将会被重写，所以如果需要保存该串最后复制出来自己管理。

3. 新型网路地址转化函数 inet_pton 和 inet_ntop

这两个函数是随 IPv6 出现的函数，对于 IPv4 地址和 IPv6 地址都适用，函数中 p 和 n 分别代表表达（presentation)和数值（numeric)。地址的表达格式通常是 ASCII 字符串，数值格式则是存放到套接字地址结构的二进制值。

2.1.4. 常用函数

进行 Socket 编程的常用函数有：

1. Socket

创建一个 Socket。

2. Bind

用于绑定 IP 地址和端口号到 socket。

3. Connect

该函数用于绑定之后的 client 端，与服务器建立连接

4. Listen

设置能处理的最大连接要求，*listen()* 并未开始接收连线，只是设置 socket 为 listen 模式。

5. Accept

用来接受 socket 连接。

6. send/write

发送数据

7. recv/read

接收

Quectel Confidential

2.2. TCP Socket 开发

2.2.1. TCP 服务器

1. 创建一个 socket，用函数 *socket()*
2. 绑定 IP 地址、端口等信息到 socket 上，用函数 *bind()*
3. 设置允许的最大连接数，用函数 *listen()*
4. 接收客户端上来的连 接，用函数 *accept()*
5. 收发数据，用函数 *send()*和 *recv()*，或者 *read()*和 *write()*
6. 关闭网络连

2.2.2. TCP 客户端

1. 创建一个 socket，用函数 *socket()*
2. 设置要连接的对方的 IP 地址和端口等属性
3. 连接服务器，用函数 *connect()*
4. 收发数据，用函数 *send()*和 *recv()*，或者 *read()*和 *write()*
5. 关闭网络连接

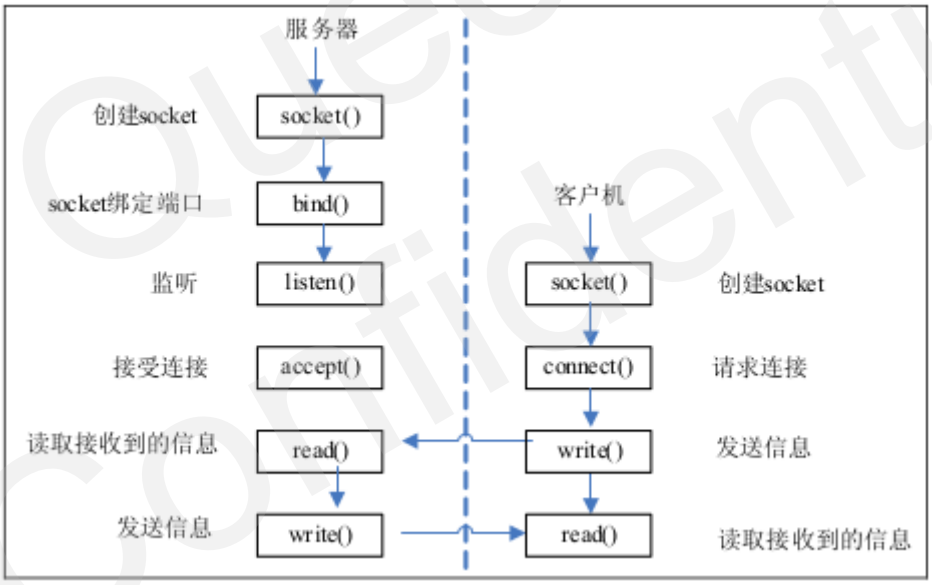
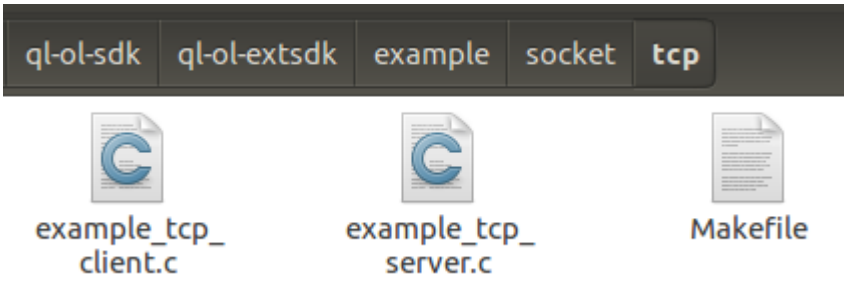


图 1 TCP 套接字工作流程

2.2.3. 例子



2.3. UDP Socket 开发

2.3.1. UDP 服务器

1. 创建一个 Socket，用函数 *socket()*
2. 绑定 IP 地址、端口等信息到 socket 上，用函数 *bind()*
3. 循环接收数据，用函数 *recvfrom()*
4. 关闭网络连接

2.3.2. UDP 客户端

1. 创建一个 Socket，用函数 *socket()*
2. 绑定 IP 地址、端口等信息到 socket 上，用函数 *bind()*
3. 设置对方的 IP 地址和端口等属性
4. 发送数据，用函数 *sendto()*
5. 关闭网络连接

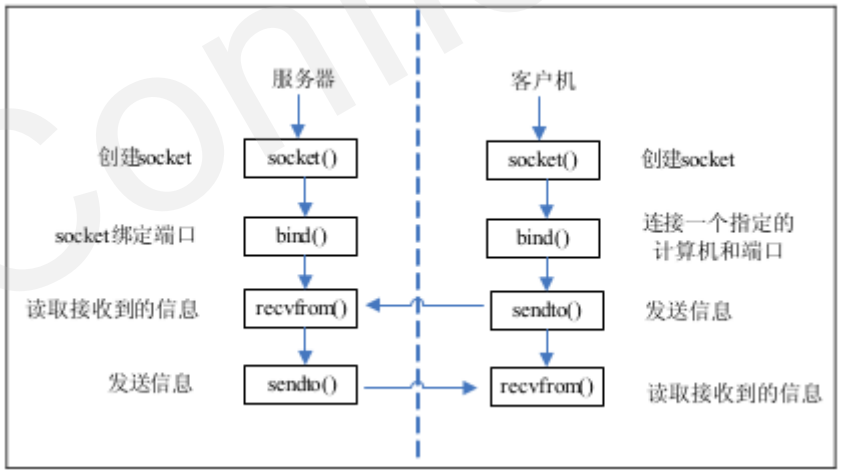
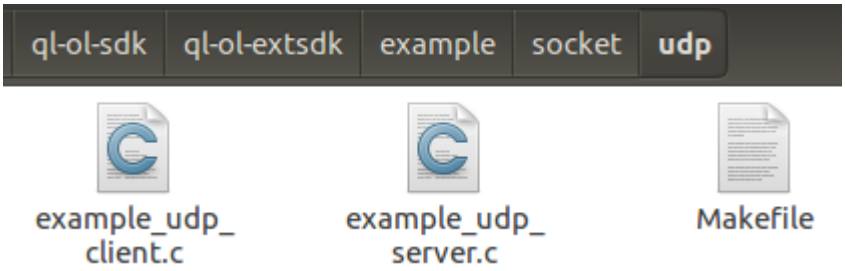


图 2 UDP 套接字工作流程

Quectel Confidential

2.3.3. 例子



Quectel Confidential

3 SSL 应用开发

SSL 应用开发可以基于很多应用库，常用的有 OpenSSL、Guntls 库。本部分的应用开发基于 OpenSSL 库,版本 1.0.1。

3.1. SSL 介绍

SSL(Secure Socket Layer)是 NetScape 公司提出的主要用于 web 的安全通信标准,分为 2.0 版和 3.0 版.TLS(Transport Layer Security)是 IETF 的 TLS 工作组在 SSL3.0 基础之上提出的安全通信标准，目前版本是 1.0，即 RFC2246.SSL/TLS 提供的安全机制可以保证应用层数据在互联网络传输不被监听，伪造和窜改。它的简要历史如下：

1994 年，NetScape 公司设计了 SSL 协议（Secure Sockets Layer）的 1.0 版，但是未发布。
1995 年，NetScape 公司发布 SSL 2.0 版，很快发现有严重漏洞。
1996 年，SSL 3.0 版问世，得到大规模应用。
1999 年，IETF 组织 ISOC 接替 NetScape 公司，发布了 SSL 的升级版 TLS 1.0 版。
2006 年和 2008 年，TLS 进行了两次升级，分别为 TLS 1.1 版和 TLS 1.2 版。最新的变动是 2011 年 TLS 1.2 的修订版。

目前，应用最广泛的是 TLS 1.0，接下来是 SSL 3.0。但是，主流浏览器都已经实现了 TLS 1.2 的支持。TLS 1.0 通常被标示为 SSL 3.1，TLS 1.1 为 SSL 3.2，TLS 1.2 为 SSL 3.3。

3.2. OpenSSL 介绍

OpenSSL 不仅仅是 SSL。它可以实现消息摘要、文件的加密和解密、数字证书、数字签名和随机数字。关于 OpenSSL 库的内容非常多，远不是一篇文章可以容纳的。OpenSSL 不只是 API，它还是一个命令行工具。命令行工具可以完成与 API 同样的工作，而且更进一步，可以测试 SSL 服务器和客户机。

对程序来说，OpenSSL 将整个握手过程用一对函数体现，即客户端的 SSL_Connect 和服务端的 SSL_Accept.而后的应用层数据交换则用 SSL_Read 和 SSL_Write 来完成。

Quectel Confidential

3.3. 基于 OpenSSL 开发

3.3.1. 常用 API

1. int SSL_CTX_set_cipher_list(SSL_CTX *,const char *str);

根据 SSL/TLS 规范，在 ClientHello 中，客户端会提交一份自己能够支持的加密方法的列表，由服务端选择一种方法后在 ServerHello 中通知服务端, 从而完成加密算法的协商。

可用的算法为:
EDH-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
DES-CBC3-SHA
DHE-DSS-RC4-SHA
IDEA-CBC-SHA
RC4-SHA
RC4-MD5
EXP1024-DHE-DSS-RC4-SHA
EXP1024-RC4-SHA
EXP1024-DHE-DSS-DES-CBC-SHA
EXP1024-DES-CBC-SHA
EXP1024-RC2-CBC-MD5
EXP1024-RC4-MD5
EDH-RSA-DES-CBC-SHA
EDH-DSS-DES-CBC-SHA
DES-CBC-SHA
EXP-EDH-RSA-DES-CBC-SHA
EXP-EDH-DSS-DES-CBC-SHA
EXP-DES-CBC-SHA
EXP-RC2-CBC-MD5
EXP-RC4-MD5

这些算法按一定优先级排列，如果不作任何指定，将选用 DES-CBC3-SHA.用 SSL_CTX_set_cipher_list 可以指定自己希望用的算法(实际上只是提高其优先级，是否能使用还要看对方是否支持)。

我们在程序中选用了 RC4 做加密，MD5 做消息摘要(先进行 MD5 运算，后进行 RC4 加密)，即 *SSL_CTX_set_cipher_list(ctx,"RC4-MD5");*

在消息传输过程中采用对称加密(比公钥加密在速度上有极大的提高)，其所用秘钥(shared secret)在握手过程中中协商(每次对话过程均不同，在一次对话中都有可能有几次改变)，并通过公钥加密的手段由客户端提交服务端。

2. *void SSL_CTX_set_verify(SSL_CTX ctx,int mode,int (*callback)(int, X509_STORE_CTX));*
缺省 mode 是 SSL_VERIFY_NONE，如果想要验证对方的话，便要将此项变成 SSL_VERIFY_PEER.

Quectel Confidential

SSL/TLS 中缺省只验证 server，如果没有设置 SSL_VERIFY_PEER 的话，客户端连证书都不会发过来。

3. *int SSL_CTX_load_verify_locations(SSL_CTX*ctx, const char*CAfile,const char*CApath);*
要验证对方的话，当然装要有 CA 的证书了，此函数用来便是加载 CA 的证书文件。

4. *int SSL_CTX_use_certificate_file(SSL_CTX*ctx, const char*file, int type);*
加载自己的证书文件

5. *int SSL_CTX_use_PrivateKey_file(SSL_CTX*ctx, const char*file, int type);*
加载自己的私钥，以用于签名。

6. *int SSL_CTX_check_private_key(SSL_CTX*ctx);*
调用了以上两个函数后，自己检验一下证书与私钥是否配对。

7. *void RAND_seed(const void*buf,int num);*

8. *OpenSSL_add_ssl_algorithms()或SSLeay_add_ssl_algorithms();*

其实都是调用 *int SSL_library_init(void);*
进行一些必要的初始化工作，用 openssl 编写 SSL/TLS 程序的话第一句便应是它。

9. *void SSL_load_error_strings(void);*
如果想打印出一些方便阅读的调试信息，便要在一开始调用此函数。

10. *void ERR_print_errors_fp(FILE*fp);*
如果调用了 *SSL_load_error_strings()*后，便可以随时用 *ERR_print_errors_fp()*来打印错误信息了。

11. *X509*SSL_get_peer_certificate(SSL*s);*
握手完成后，便可以用此函数从 SSL 结构中提取出对方的证书(此时证书得到且已经验证过了)整理成 X509 结构。

12. *X509_NAME*X509_get_subject_name(X509*a);*
得到证书所有者的名字，参数可通过 *SSL_get_peer_certificate()*得到的 X509 对象。

13. *X509_NAME*X509_get_issuer_name(X509*a)*
得到证书签署者(往往是 CA)的名字，参数可用通过 *SSL_get_peer_certificate()*得到的 X509 对象。

14. *char*X509_NAME_oneline(X509_NAME*a,char*bufint size);*
将以上两个函数得到的对象变成字符型，以便打印出来。

15. SSL_METHOD 的构造函数
包括：

SSL_METHOD TLSv1_server_method(void); / TLSv1.0/*

*SSL_METHOD TLSv1_client_method(void); / TLSv1.0 */*

*SSL_METHOD SSLv2_server_method(void); / SSLv2 */*
*SSL_METHOD SSLv2_client_method(void); / SSLv2 */*

*SSL_METHOD SSLv3_server_method(void); / SSLv3 */*
*SSL_METHOD SSLv3_client_method(void); / SSLv3 */*

*SSL_METHOD SSLv23_server_method(void); / SSLv3 but can rollback to v2 */*
*SSL_METHOD SSLv23_client_method(void); / SSLv3 but can rollback to v2 */*

在程序中究竟采用哪一种协议(TLSv1/SSLv2/SSLv3)，就看调哪一组构造函数了。

3.3.2. 相关头文件

Socket 头文件
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>

SSL 头文件
#include <openssl/ssl.h>
#include <openssl/err.h>

3.3.3. 证书制作

这里介绍如何生成一个自签名的根证书和私钥。
1. 生成一个 2048bit 的 RSA 算法密钥对
OpenSSL genrsa -out server_key.pem 2048

2. 证书生成请求
OpenSSL req-new-key server_key.pem-out server_cert.csr-subj"/C=CN/ST=ANHUI/L=HEFEI/O=QUEC/OU=OPEN/CN=www.quectel.com"-days 7300

3. 生成 X509 格式规范证书
OpenSSL x509-req-days 7300-sha256–signkey server_key.pem-in server_cert.csr-out server_cert.pem

4. 转换证书编码格式 PEM->DER
OpenSSL x509-inform PEM -in server_cert.pem -outform DER -out server_cert.der

5. 查看证书

Quectel Confidential

OpenSSL x509 -inform DER-in server_cert.der-text

3.3.4. SSL 服务器

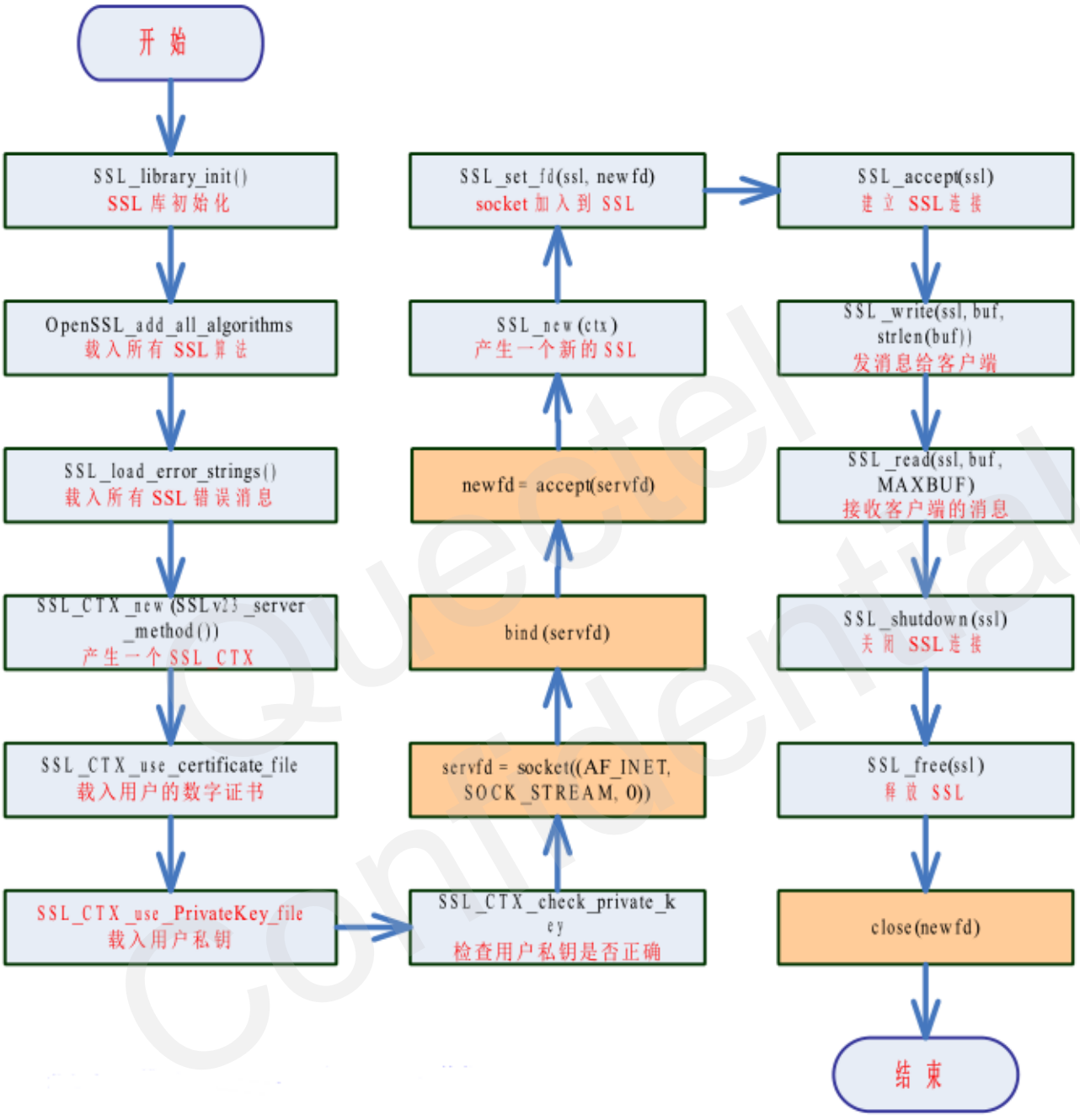


图 3 SSL 服务端开发流程

Quectel Confidential

3.3.5. SSL 客户端

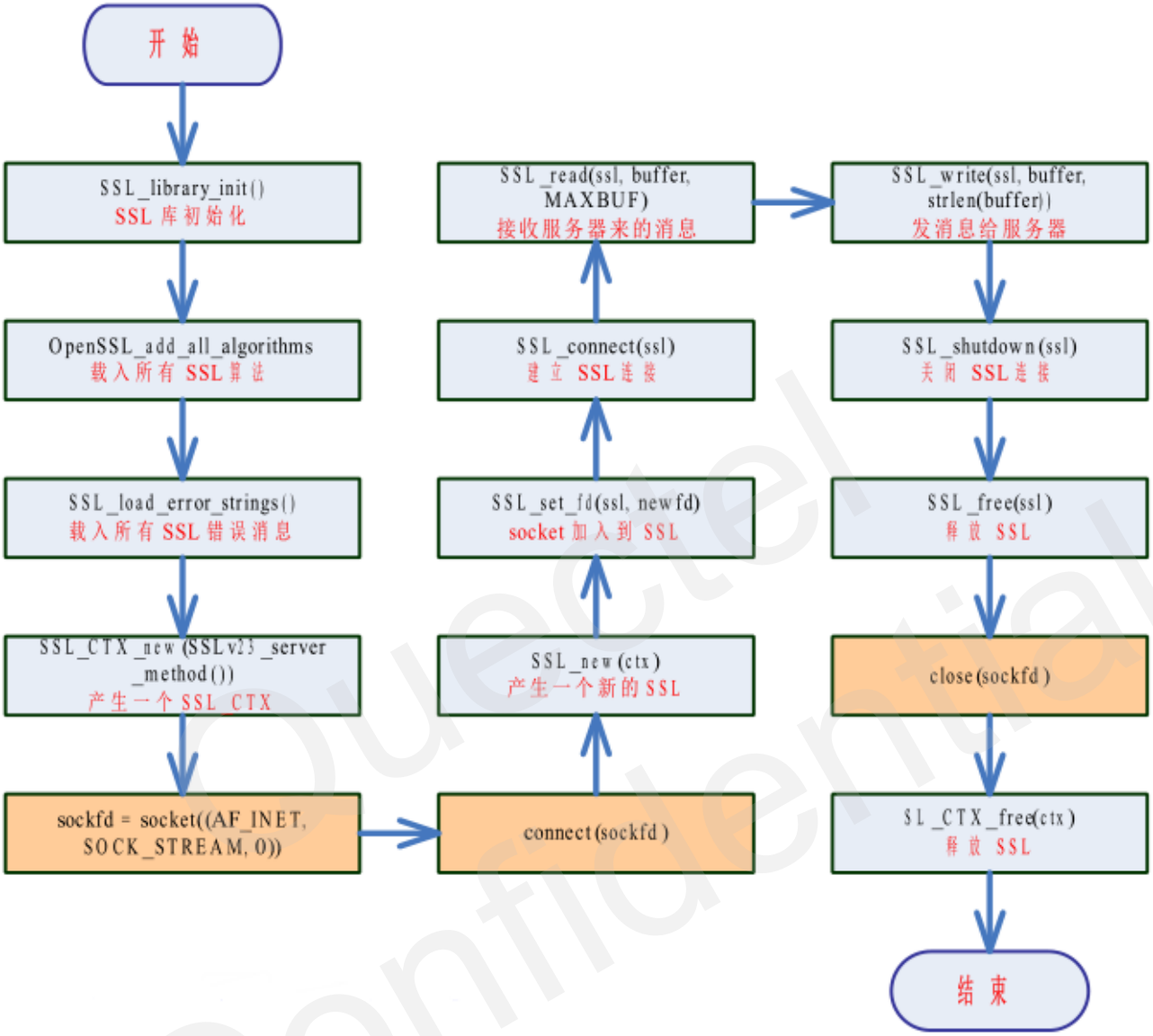
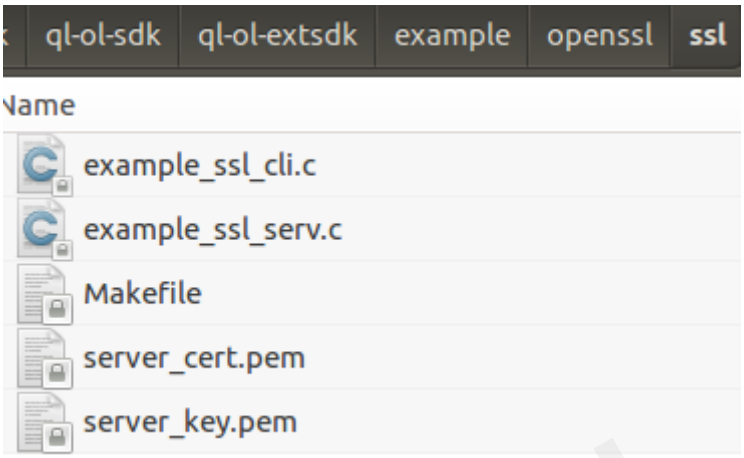


图 4 SSL 客户端开发流程

Quectel Confidential

3.3.6. 例子



Quectel Confidential

4 HTTP(S)应用开发

本部分简单介绍 HTTP 协议，在 Linux 系统环境下，HTTP 应用开发可以基于很多开发库，但是最强大属于 CURL 库。

4.1. HTTP 介绍

HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（www.world wide web）服务器传输超文本到本地浏览器的传送协议。

HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件，图片文件，查询结果等）。

HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 www 中使用的是 HTTP /1.0 和 1.1。

HTTP/1.1 相较于 HTTP/1.0 协议的区别主要体现在：

1. 缓存处理
2. 带宽优化及网络连接的使用
3. 错误通知的管理
4. 消息在网络中的发送
5. 互联网地址的维护
6. 安全性及完整性

4.1.1. HTTP 传输过程

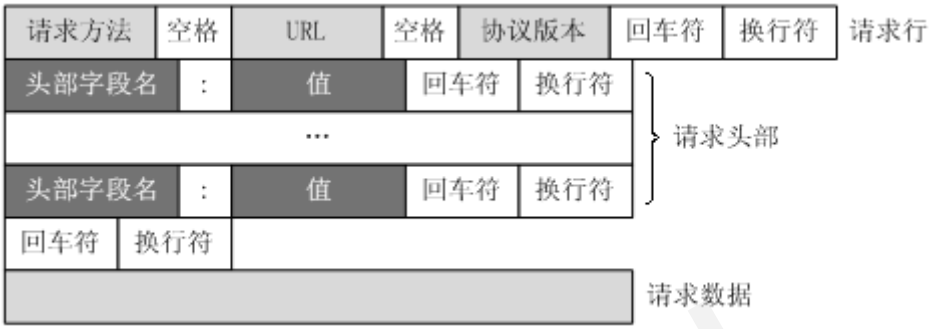
一次 HTTP 操作称为一个事务，其工作过程可分为四步：

1. 首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作就开始了。
2. 建立连接后，客户机发送一个请求给服务器，**请求方式**的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可能的内容。
3. 服务器接到请求后，给予相应的**响应信息**，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。
4. 客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

Quectel Confidential

4.1.2. HTTP 报文格式

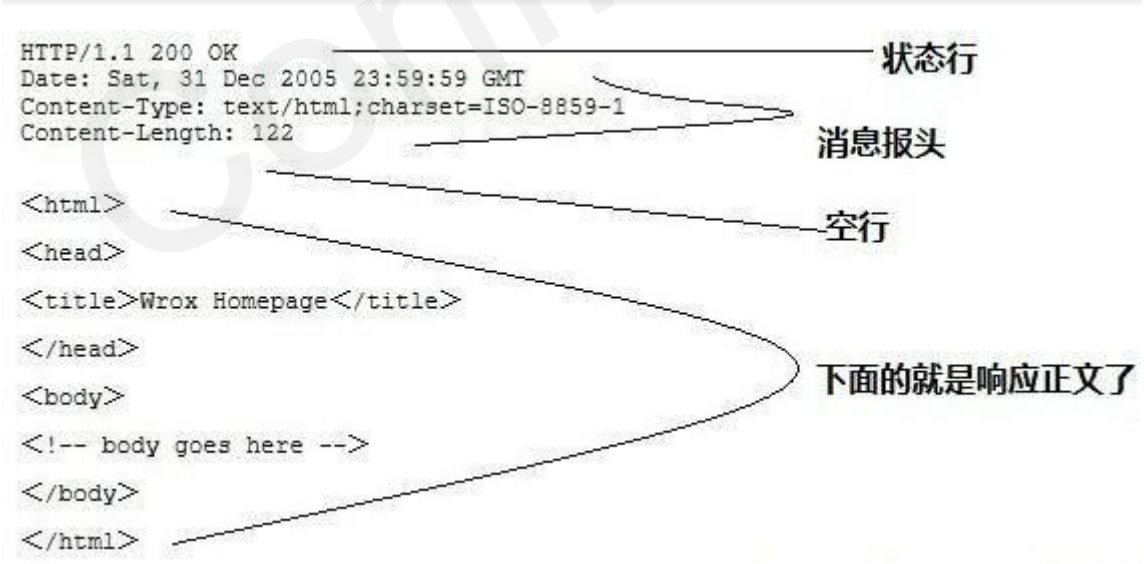
4.1.2.1. Request 报文



Get 请求例子，使用 Charles 抓取的 Request：

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
Host: img.mukewang.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36
Accept: image/webp，image/*,*/*;q=0.8
Referer: http://www.imooc.com/
Accept-Encoding: gzip, deflate，sdch
Accept-Language: zh-CN,zh;q=0.8
```

4.1.2.2. Response 报文



Quectel Confidential

4.1.3. HTTP 请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。
HTTP1.0 定义了三种请求方法：GET, POST 和 HEAD 方法。
HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。
GET 请求指定的页面信息，并返回实体主体。
HEAD 类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头。
POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
PUT 从客户端向服务器传送的数据取代指定的文档的内容。
DELETE 请求服务器删除指定的页面。
CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
OPTIONS 允许客户端查看服务器的性能。
TRACE 回显服务器收到的请求，主要用于测试或诊断。

4.1.4. HTTP 状态码

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

1xx：指示信息—表示请求已接收，继续处理
2xx：成功—表示请求已被成功接收、理解、接受
3xx：重定向—要完成请求必须进行更进一步的操作
4xx：客户端错误—请求有语法错误或请求无法实现
5xx：服务器端错误—服务器未能实现合法的请求

常见状态码：

```
200 OK                      //客户端请求成功
400 Bad Request             //客户端请求有语法错误，不能被服务器所理解
401 Unauthorized            //请求未经授权，这个状态代码必须和 www-authenticate 报头域
                              一起使用
403 Forbidden               //服务器收到请求，但是拒绝提供服务
404 Not Found               //请求资源不存在，eg：输入了错误的 URL
500 Internal Server Error   //器发生不可预期的错误
503 Server Unavailable      //器当前不能处理客户端的请求，一段时间后可能恢复正常
```

4.2. 基于 curl 库开发

4.2.1. 基本步骤

1. 初始化库

```
curl = curl_easy_init()
```

2. 设置域名
curl_easy_setopt(curl, CURLOPT_URL, "http://www.quectel.com/");

3. 资源请求
curl_easy_perform(curl);

4. 本地解析
curl_easy_getinfo(curl, CURLINFO_CONTENT_TYPE, &ct);

5. 清空会话
curl_easy_cleanup(curl);

4.2.2. HTTPS 特殊操作

1. 设置 SSL 协议版本
curl_easy_setopt(curl, CURLOPT_SSLVERSION, CURL_SSLVERSION_TLSv1)

2. 对端证书检查
curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L); //关闭检查

3. 添加根证书
curl_easy_setopt(curl, CURLOPT_CAINFO, "cacert.pem");

4. 设置 SSL 加密算法套件
curl_easy_setopt(curl, CURLOPT_SSL_CIPHER_LIST, "TLSv1");

4.2.3. 例子

Quectel Confidential