



## 摘要

# 目 录

第一章 概述 .....	1
1.1 开发的背景及意义 .....	1
1.2 国内外研究的现状以及存在的问题 .....	1
1.2 设计目标及内容 .....	2
第二章 相关技术研究 .....	2
2.1 Qt5 介绍 .....	2
2.1.1 Qt 的信号和槽(signal and slots) .....	3
2.1.2 Qt 的网络库 .....	3
2.1.3 Qt 的线程库 .....	3
2.1.4 Qt Creator 介绍 .....	4
2.2 TCP/IP 协议栈介绍 .....	4
2.3 winpcap API 介绍 .....	5
2.4 RC4 加密算法介绍 .....	5
2.5 quicklz 压缩算法介绍 .....	6
第三章 软件需求分析 .....	6
2.1 功能性需求分析 .....	6
2.2 非功能性需求分析 .....	6
2.4 开发环境及工具介绍 .....	7
第四章 软件框架设计 .....	7
3.1 系统总体框架设计 .....	7
3.2 功能模块设计 .....	8

3.2.1 主流程 .....	8
3.2.2 HTTP 数据发送与接收.....	8
3.2.3 TCP 数据构造与传输 .....	10
3.2.4 TCP 数据解析 .....	13
3.2.5 加密与压缩 .....	15
3.2.6 数据抓取功能.....	17
第五章 软件详细设计 .....	20
4.1 HTTP 数据发送.....	20
4.2 TCP 数据发送 .....	22
4.3 TCP 数据解析 .....	25
4.4 数据压缩解压、加密解密 .....	27
4.5 数据监听 .....	28
第六章 系统测试 .....	31
5.1 测试环境简介 .....	31
5.2 各功能模块的测试.....	31
5.2.1 HTTP 数据发送测试 .....	32
5.2.2 TCP 数据发送接收测试.....	33
5.2.3 数据加密与解密测试.....	35
5.2.4 数据压缩与解压测试.....	37
第七章 总结 .....	39
参考文献 .....	39



# 第一章 概述

## 1.1 开发的背景及意义

随着信息产业与智能技术的发展和全球信息基础设施的不断完善，各个行业都迈入了互联网+，进入了一个新的时代。与此同时，软件开发作为这些的基础支持行业，也逐渐发展壮大，WEB 开发、APP 开发、平台级服务、各种管理工具……逐渐走入大众的视线。

效率在软件开发中尤其重要，自己做的软件比别人先上线那怕一点时间，都可能取得很大的优势，软件开发者平日里会有许多的琐事需要处理，它们重复、费时、且没有意义，所以对于软件开发而言，减少这样的琐事是一件非常重要的事情，既然是开发联网的应用程序，肯定会涉及到网络中数据的传输；在应用开发的初期阶段，服务端和客户端刚开始对接的时候，这时一旦出现問題非常不好确认问题究竟是出在服务端还是客户端，这个问题一般需要花费一定的时间，如果能够做一个小工具来帮开发者们完成这一繁琐的工作，那么对于开发者们来说绝对是一个福音。

本文将实现了一个网络数据构造软件。这个软件用于构造网络应用层上的数据，帮助开发都者完成功能测试时的各种繁琐问题；通过这个软件，开发者可以构造 HTTP 请求报文（POST 和 GET 均可使用）；也可以构造 TCP 二进制数据流；还可以进行数据抓取，查看发送和收到的 TCP 数据，如果需要还可以对数据进行筛选，决定要抓取哪些类型的数据报文。

## 1.2 国内外研究的现状以及存在的问题

网络数据分析主要作用是网络故障排查，程序 DEBUG 时非常有用，目前使用抓包软件来完成这一功能；现如今的抓包软件主要是 wireshark 和 fiddler，其中 wireshark 用于分析分析网络底层协议，主要针对那些只关心网络底层协议的实现，不关心网络中传输的数据数据内容的用户；Fiddler 是一个强大的 HTTP 数据抓取工具，主要用于截获网络中 HTTP 数据，如果安装有 HTTPS 证书，还能用于抓取 HTTPS 数据包，实现了真正的数据截取，可以由用户决定哪些数据包发送，哪些数据包接收，并且能够对发送/接收的 HTTP 数据进行修改。

但不论是 wireshark 还是 fiddler，都是用于分析网络协议用的，这些都是针对网络管理员设计的功能，对于应用开发者而言，真正关心的是我到底发送了什么数据，又接收了什么数据，而这些底层协议信息毫无用处，所以在应用开发时，应该尽可能的屏蔽掉底层的协议信息，但是目前还没有类似的能够直接显示应用发送的信息软件；本次设计的网络数据构造工具，主要是用于发送应用层的协议，它能够发送/接收 HTTP 数据和 TCP 数据，能够由用户自己来构造 TCP 数据，收到数据后由用户决定如何解析；当然，经也提供了数据包抓取的功能，并可以对数据进行加密解密、压缩解压，大方便了开发者的使用，为开发者减少了一定的工作量；

## 1.2 设计目标及内容

### 1、整个软件全部操作由图形界面来完成

软件所有的功能都通过 GUI 与用户来交互，降低上手难度，方便使用；

### 2、HTTP 数据的发送与接收

用户能够通过该软件发送 HTTP 数据，包括 POST 和 GET 请求两种，并且在接收到数据后，即时地，用正确的方式呈现给用户；

### 3、TCP 数据构造与解析

对于即将发送的数据，用户可以构造任何自己想要的格式，收到数据后，用户可以用自己想要的格式来解析数据；

### 4、TCP 数据的发送与接收

通过该软件，用户能够向目标主机+端口发送已经构造好的 TCP 数据；

### 5、网络数据监听

提供简单的网络数据监听功能，抓取指定网上的 TCP 数据，并对收到的数据按照用户定义的规则进行简单的筛选；

### 6、TCP 数据加密解密

对于即将发送的数据，进行加密操作；对于接收到的数据进行解密操作，解密算法可以配置；

### 7、TCP 数据压缩与解压

对于即将发送的数据，进行压缩操作；对接收到的数据进行解压操作，压缩算法可以由用户配置；

## 第二章 相关技术研究

### 2.1 Qt5 介绍

Qt 是一个由 Qt Company 开发的跨平台 C++ 图形用户界面应用程序的开发框架。它既可以用于开发非 GUI 程序，也可以用于开发 GUI 程序。

标准的 C++ 对象模型可以在运行时非常有效地支持对象范式 (object paradigm)，但是它的静态特性在一些问题领域中不够灵活，图形用户界面编程不仅需要运行时的高效性，还需要高度的灵活性。为此 Qt 在标准 C++ 对象模型的基础上添加了一些特性，形成了自己的对象模型。最主要的特性有：

一个强大的无缝对象通信机制——信号与槽 (signals and slots)；

分层结构的、可查询的对象树 (object trees)，它使用一种很自然的方式来组织对象拥有权 (object ownership)；

可查询、可设计的对象属性系统 (object properties)；

强大的事件过滤器 (events and event filters)；

守卫指针 QPointer，它在引用对象被销毁时自动将其设置为 0；

支持创建自定义类型 (custom type)；

Qt 这些特性都是遵循标准 C++ 规范内实现的，使用这些特性都必须继承自 QObject 类。其中对象通信机制和动态属性系统还需要无对象系统 (Meta-Object System) 的支持。

### 2.1.1 Qt 的信号和槽(signal and slots)

信号和槽用于两个对象之间的通信。信号和槽机制是 Qt 的核心特征，也是 Qt 不同于其它开发框架的最突出特征。在 GUI 编程中，当改变了一个部件时，总希望其它部件也能了解到该变化。更一般来说，我们希望任何对象都可以和其它对象进行通信。当一个特殊事件发生时便可以发射一个信号，比如按钮被单击就发射 clicked() 信号；而槽就是一个函数，它在信号发射后被调用来响应这个信号；比如点击窗口关闭按钮，便会发射一个 clicked() 信号，而与之关联的槽 close() 便会执行；而在实际的使用中，更多的时候还是在使用自定义的 signal、slot，下面将介绍自定义 signal、slot 的方法；

```
/* 自定义槽 */
private slots:
    void MainWindow::myTestSlot(int value)
    {
        qDebug() << "received value = " << value;
    }
/* 自定义信号,函数只需要声明,不需要实现 */
signals:
    void myTestSignal(int value);
/* 关联自定义的信号与槽 */
connect(this, &MainWindow::myTestSignal, this,
&MainWindow::myTestSlot, Qt::AutoConnection);
/* 发射信号 */
emit myTestSignal(123);
```

Qt::AutoConnection 是 connect 第五个参数的默认值，在单线程下会使用 Qt::DirectConnection，在线程下会使用 Qt::QueuedConnection，注意在多线程下 connect 信号与槽需要谨慎使用 Qt::DirectConnection，因为如果加上这个参数后，它便不是线程安全的了；

### 2.1.2 Qt 的网络库

Qt 中的 Qt Network 模块用来编写基于 TCP/IP 的网络程序，其中提供了在网络中较底层的原始套接字使用类，比如 QTcpSocket，QTcpServer 和 QUdpSocket 等；也提供了网络中应用级别的类，比如 QNetworkRequest、QNetworkReply 和 QNetworkAccessManager；更有实现负载管理的类 QNetworkConfiguration、QNetworkConfigurationManger、QNetworkSession…… 如果要使用 Qt Network 模块中的类，则需要在项目文件.pro 中添加 QT += network 一行代码；

### 2.1.3 Qt 的线程库

对于图形界面的编程而言，多线程编程并不是为了解决高并发的的问题，而是在编写有图形界面的应用程序时，会有一些比较耗时的操作会阻塞界面主线程，从而导致界面假死的情况，这时最好的方法就是把这些耗时的操作放到一个子线程中去。

Qt 中的 QThread 类提供了对线程的支持，这包括一组与平台无关的线程、一个线程安全的发送事件的方式以及跨线程的信号-槽的关联。这些使得我们可



以很轻松地开发和移植多线程 Qt 应用程序，可以充分利用多处理器的机器。

### 2.1.4 Qt Creator 介绍

Qt Creator 是一个跨平台的、完整的 Qt 集成开发环境 (IDE)，其中包括了高级的 C++ 代码编辑器、项目和生成管理工具、集成的上下文相关的帮助系统、图形化调试器、代码管理和浏览工具等。我将使用 Qt Creator 5.7 进行软件代码的编写，界面的设计，源代码的管理等工作。

## 2.2 TCP/IP 协议栈介绍

传输控制层协议/因特网互联协议 (Transmission Control Protocol/Internet Protocol) 是 Internet 最基本的协议、Internet 国际互联网的基础，由网络层的 IP 协议和传输层的 TCP 协议组成。TCP/IP 定义了电子设备如何连入 Internet，以及数据在它们之间的传输标准。IP 层接收从以太网发来的数据，并将数据包发送到更高一层的 TCP/UDP 层；相反，IP 层也把从 TCP 或 UDP 层接收来的数据包传送到更低层：

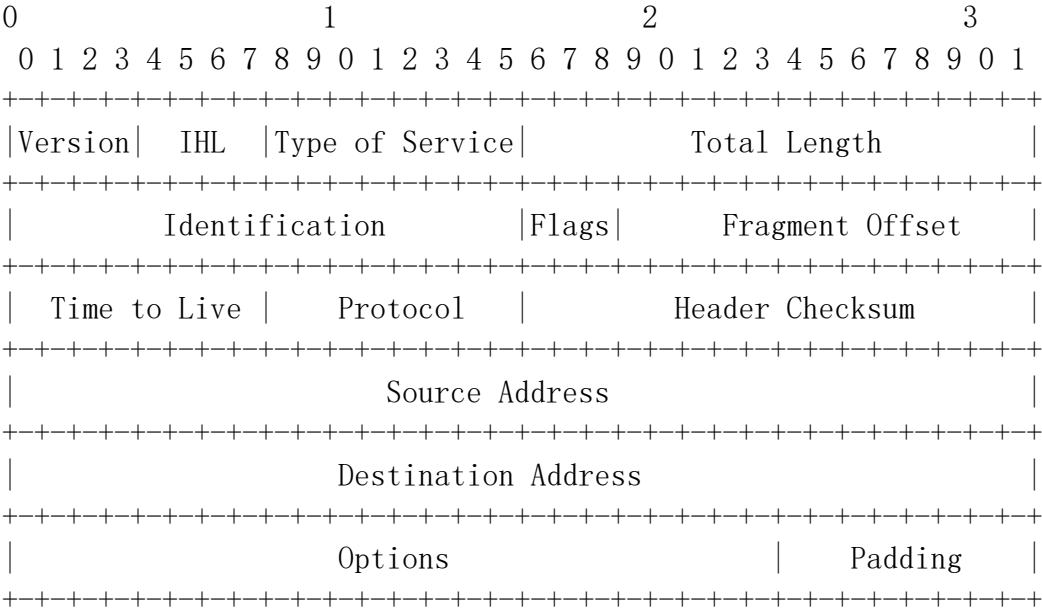
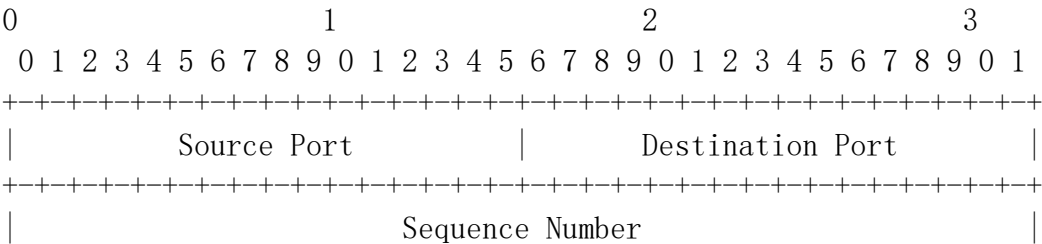


图 4.8 IP 数据报头部

TCP 是面向连接的通信协议，通过三次握手来建立连接，通讯完成时要关闭连接，TCP 提供的是一种可靠的数据流服务，采用带重传的肯定确认技术来实现传输的可靠性：



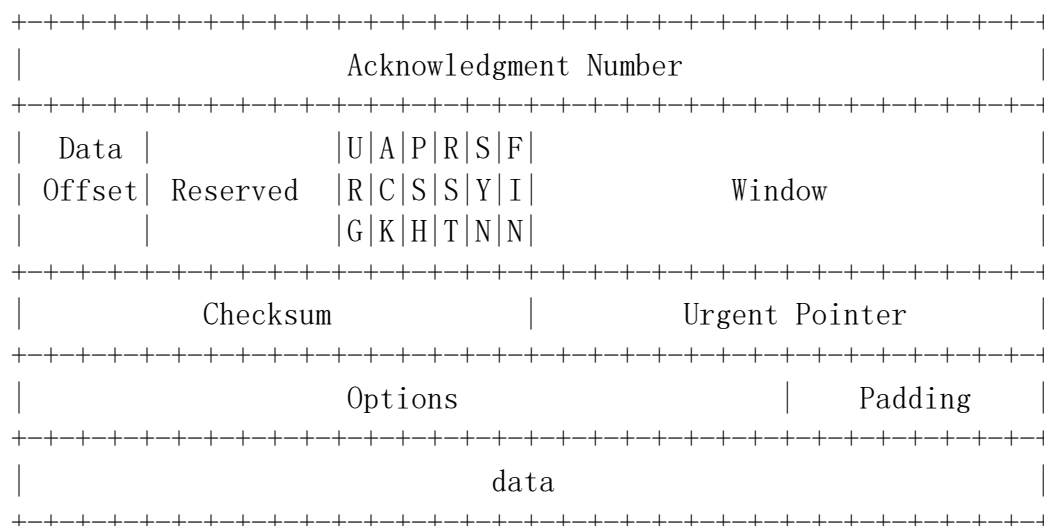


图 4.9 TCP 数据报文头部

## 2.3 winpcap API 介绍

winpcap (windows packet capture) 是 windows 平台下一个免费、的网络访问系统，它提供了一个强大的编程接口，使得开发者可以很轻松地访问网络底层协议的内容，它的主要功能有：

1. 捕获原始数据包，包括在共享网络上各主机发送/接收的以及相互之间交换的数据包
2. 在数据包发往应用程序之前，按照处定义的规则将某些特殊的数据包过滤掉（数据仍然会发向指定应用程序，只是对于 winpcap 应用程序不接收）
3. 在网络上发送原始的数据包
4. 收集网络通信过程中的统计信息；

它只能“嗅探”到物理线路上的数据包，而不能对网络中的数据包进行修改（相于只能读取，不能写入）

## 2.4 RC4 加密算法介绍

RC4 加密算法是大名鼎鼎的 RSA 三人组中的头号人物 Ronald Riverst 在 1987 年设计的密钥长度可变的流加密算法簇。之所以称其为簇，是由于基核心的 S-box 的长度可以为任意值，但是一般为 256 字节。该算法速度可以达到 DES 加密的 10 倍左右，且具有很高级别的非线性。

RC4 算法加密采用的是 XOR，所以一旦子密钥序列出现了重复，密文就有可能被破解。由于存在部分弱密钥，使得密钥序列研不到 100 万字节内就发生了完全的重复，如果是部分重复，则可能在不到 10 万字节内就发生重复，因此推荐在使用 RC4 算法时，必须对加密密钥进行测试，判断其是否为弱密钥。不过好消息是，根据目前的研究来看，没有任何分析对于密钥长度达到 128 位（16 字节）的密钥有效，所以 RC4 依然是目前最安全的加密算法之一，可以在网络加密中放心使用。

## 2.5 quicklz 压缩算法介绍

quicklz 压缩算法号称是全世界速度最快的压缩算法，压缩速度可以达到每秒 308Mbytes，而且压缩算法遵循 GPL 开源协议，源代码完全可以在官方直接下载；quicklz 主要作用是在网络数据的发送之前先经过压缩再发送，数据收到后解压再读取，这样可以大大减少网络中所需要传输的数据和数据传输的时间，特别是对于那些数据大量重复或有大量空字节的数据。

# 第三章 软件需求分析

确立软件的需求是软件开发的第一步。它决定了系统的设计，并确定了系统功能实施的程度。对于完整的系统开发流程，需求分析是我们奠定基础的的第一步。它是软件开发中最为重要的环节，也是软件设计的基础，软件实现的目标。

## 2.1 功能性需求分析

为了达到第一章所提到的目标，那么应该满足以下需求：

- 1、所有的功能都应该在 GUI 的界面下操作完成；
- 2、能够发送 HTTP 的 POST 和 GET 请求，对应接收到的 HTTP 响应，能够以最适当的应用打开；
- 3、能够构造用户想要的二进制流数据，包括编程语言里面的一些基本数据类型：int、char、long、float、double……，以及最常用的字符串 string；
- 4、能够向指定主机发送 TCP 数据，接收到数据后，将收到的数据在新窗口中显示；
- 5、能够解析收到的二进制数据流，比如接收到的数据是一个 double 类型的数据，就要以 double 数据显示出来；
- 6、能够对指定网卡上的数据进行监听，并对数据进行简单的筛选；
- 7、能够对将要发送的 TCP 数据进行压缩，对接收到的 TCP 数据和抓取到的数据进行解压操作；
- 8、对将要发送的 TCP 数据进行加密，对接收到的 TCP 数据和抓取到的数据进行解密；

## 2.2 非功能性需求分析

1、性能需求：

使用 winpcap 进行网络数据监听时，执行监听的函数会一起执行，直到结束数据监听才会终止，也就是说整个线程只会执行这一个函数，该线程的其它函数都不会执行，但是 OS 会定时向 GUI 程序发送一个保活的信号，如果多次没有收到回应信息，那么 OS 就会判定该 GUI 程序已经没有执行，并且会提示用户该进程没有响应；所以对于数据监听的这个函数，应该放到一个单独的线程中去；

另一方面是对于网络中收到并筛选过的数据，软件会将它保存在本地磁盘上面，这又是一个耗时的 IO 操作，有较大的机率阻塞 GUI 主线程，从而导致界

面出现假死的情况；所以针对所有的高频 IO 操作都放到一个 IO 操作子线程中去；

对于网络中收到的数据，我们要及时地在 GUI 界面上显示出来，如果在极短的时间内产生了极大的数据，这时程序可能会不能及时地显示出来或者使主界面出现假死的情况，所以在软件中显示每条抓取的数据时，都应该以最小的代价，显示最有价值的信息，QTableWidget 恰好可以满足这一需求，它里面所有的数据都是用字符串形式来显示的，数据的更新和插入都会非常快，所以对于网络中抓取到的数据都用 QTableWidget 来显示；

## 2、可扩展性需求：

该软件现阶段的加密算法和压缩算法都只有一种，这是由于现阶段时间不足以加入太多的算法，考虑到后期可能会加入一些其它的算法，所以在程序的设计阶段需要考虑到后期的扩展，要让以后能够轻松地加入其它的加密算法，压缩算法；

## 2.4 开发环境及工具介绍

软件采用 Qt 官方提供的 Qt Creator 集成开发环境，编程语言使用 c++，使用了 c++11 的一些新的特性，网络、文件、界面都使用 Qt 提供的类；

网络数据抓取采用 winpcap 开源库，用于抓取和筛选网络上的数据；

软件设计为可以在现在主流的 windows 系统上运行，只需要将程序解压后，便可以直接运行在电脑上运行；

# 第四章 软件框架设计

对数据构造软件系统进行了需求分析之后，下一步主要是针对软件进行框架设计，这是整个软件设计的关键内容，后面的内容都是针对系统框架设计的细化；

## 3.1 系统总体框架设计

### 1、HTTP 数据发送接收模块

该模块主要用于发送和接收 HTTP 数据，输入 URL+参数，选择请求方式后直接发送即可；

### 2、TCP 数据发送接收模块

主要用于发送和接收 TCP 数据，不负责解析数据；

### 3、TCP 数据构造解析模块

将传入该模块的 TCP 数据显示出来，并且可以由用户自定义解析数据；

### 4、数据监听模块

抓取指定网卡上的 TCP 数据，并显示在界面上面

### 5、数据加密与压缩模块

提供数据加密和压缩的类，可以在其它地方调用；

## 3.2 功能模块设计

### 3.2.1 主流程

用户打开程序后，可以在主界面选择使用 HTTP 数据发送、TCP 数据发送、网络数据抓取这三个功能，这三个功能可以同时使用，但是在界面上只能显示一个；

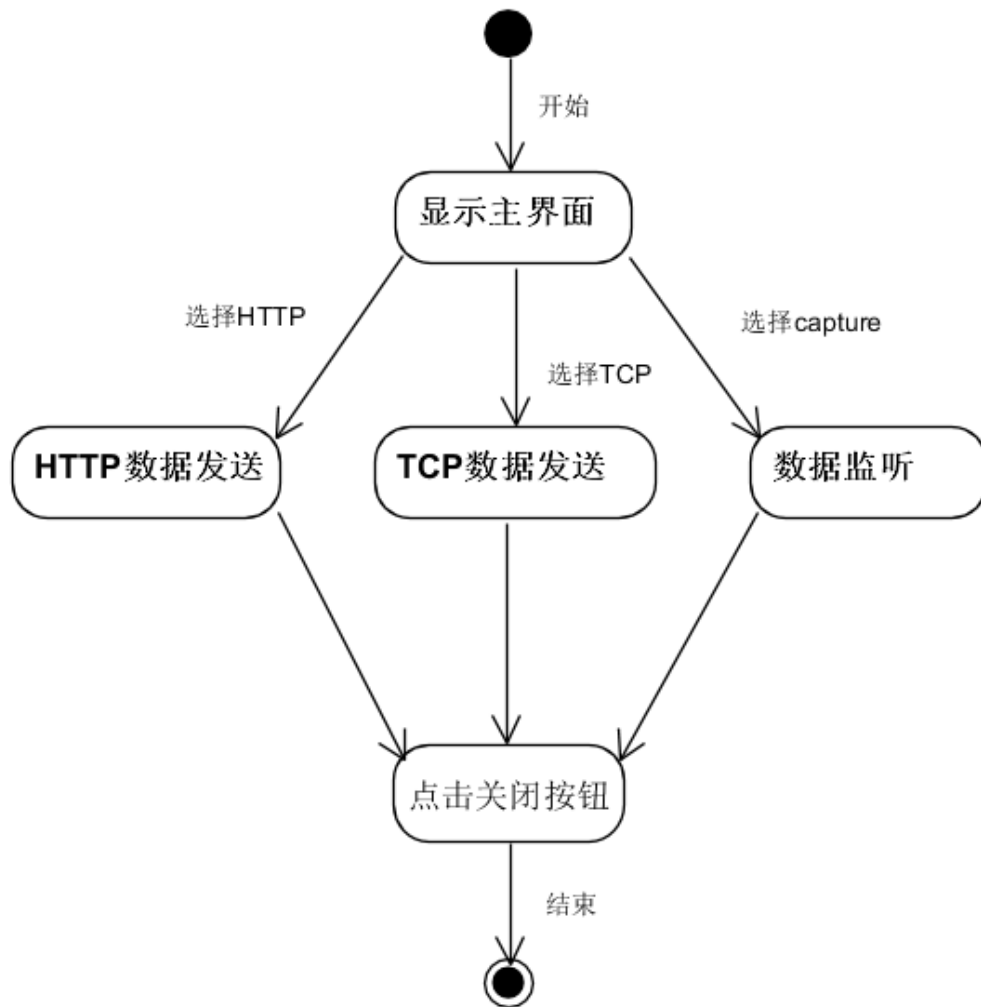
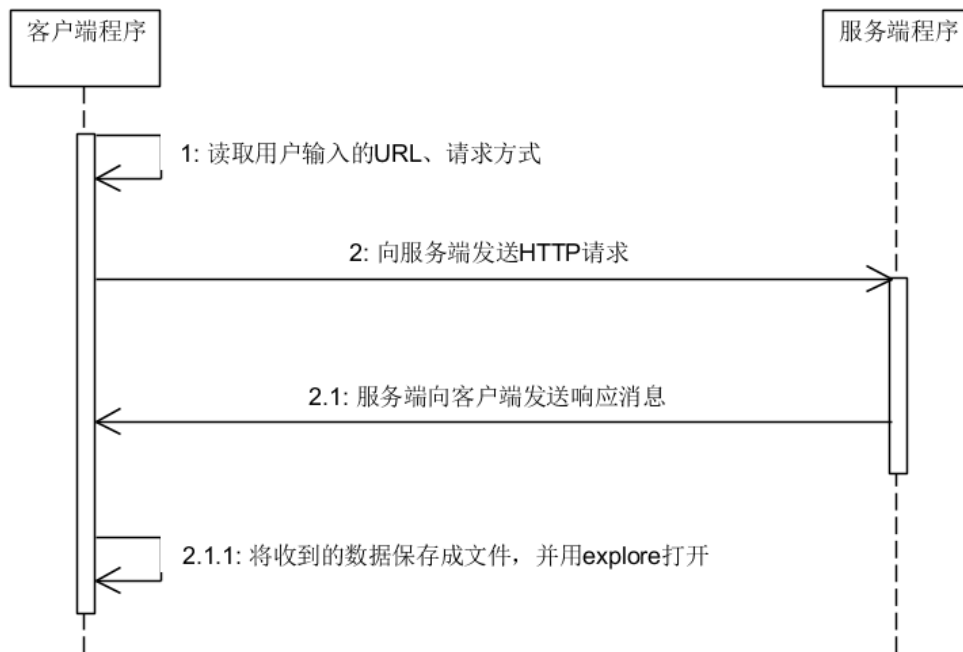


图 3.1 主流程图

### 3.2.2 HTTP 数据发送与接收

用户输入 URL、选择请求方式，并点击发送按钮，开始发送 HTTP 数据，接收到 HTTP 数据后，再显示数据内容；



3.2 HTTP 数据发送时序图

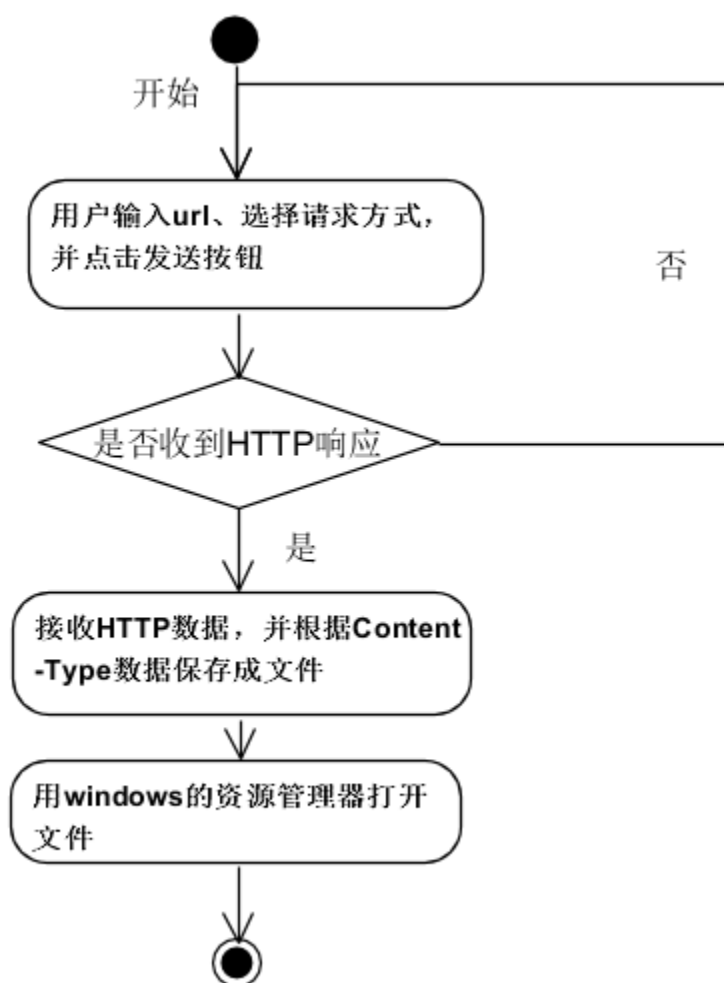


图 3.3 HTTP 数据发送活动图

<b>SendDataHttp</b>
-map _contentType : map<string, string>
-request_by_get(string url) : void
-request_by_post(string url) : void
-received_Data(QNetworkReply reply) : void
-getSuffix(string contentType) : string
+SendDataHttp(QWidget parent)

图 3.4 HTTP 数据发送类图

### 3.2.3 TCP 数据构造与传输

用户输入目标 IP 地址、目标端口，再通过程序中的数据构造功能，创建自己想要发送的数据，（如果需要，可以加密压缩数据）之后点击发送按钮即开始发送数据，如果自己与目标主机网络是畅通且目标主机开启了监听，那么数据的连接将会开启，并向目标主机发送数据；

接收到数据后，将数据传给 TCP 数据解析模块，解析数据（如果需要，对接收到的数据进行解密、解压）；

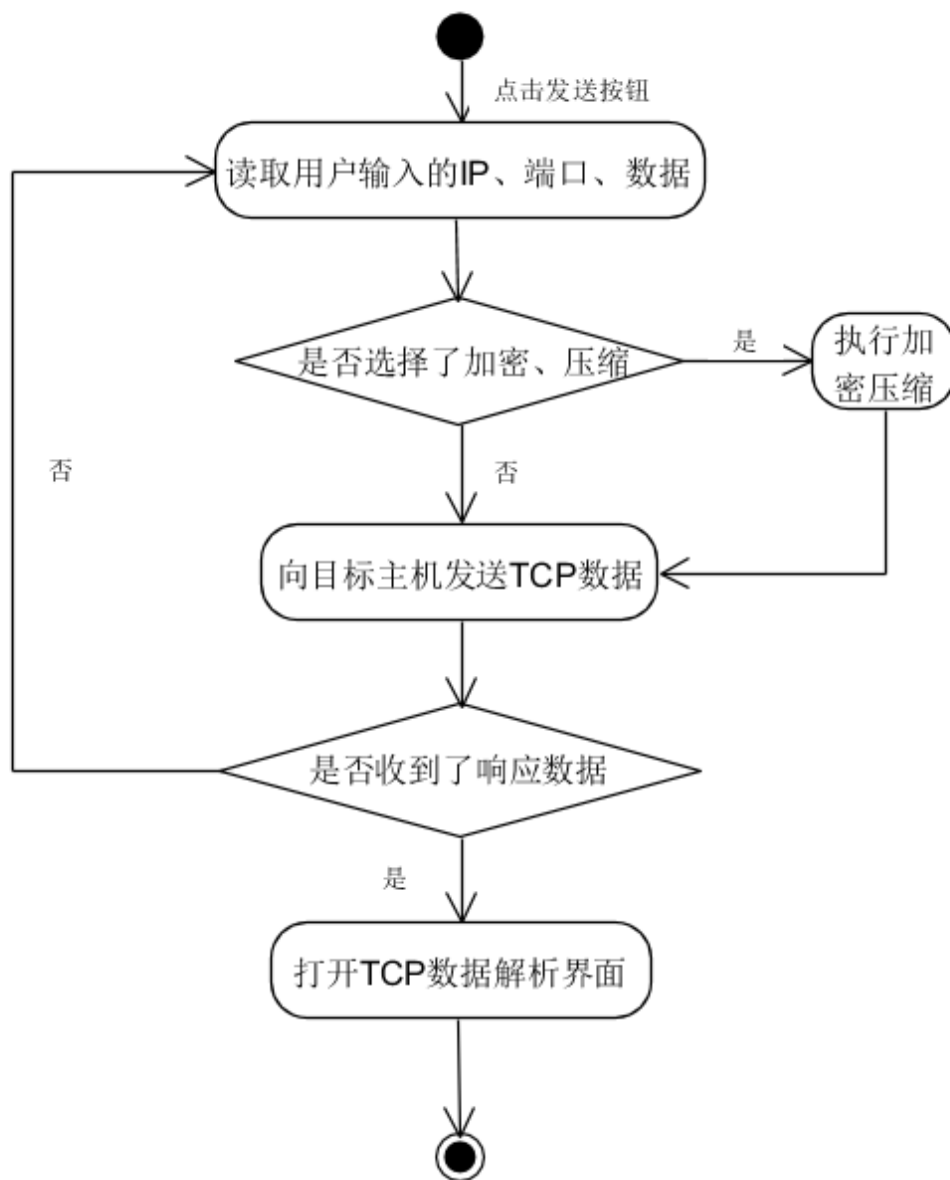


图 3.5 TCP 数据发送活动图



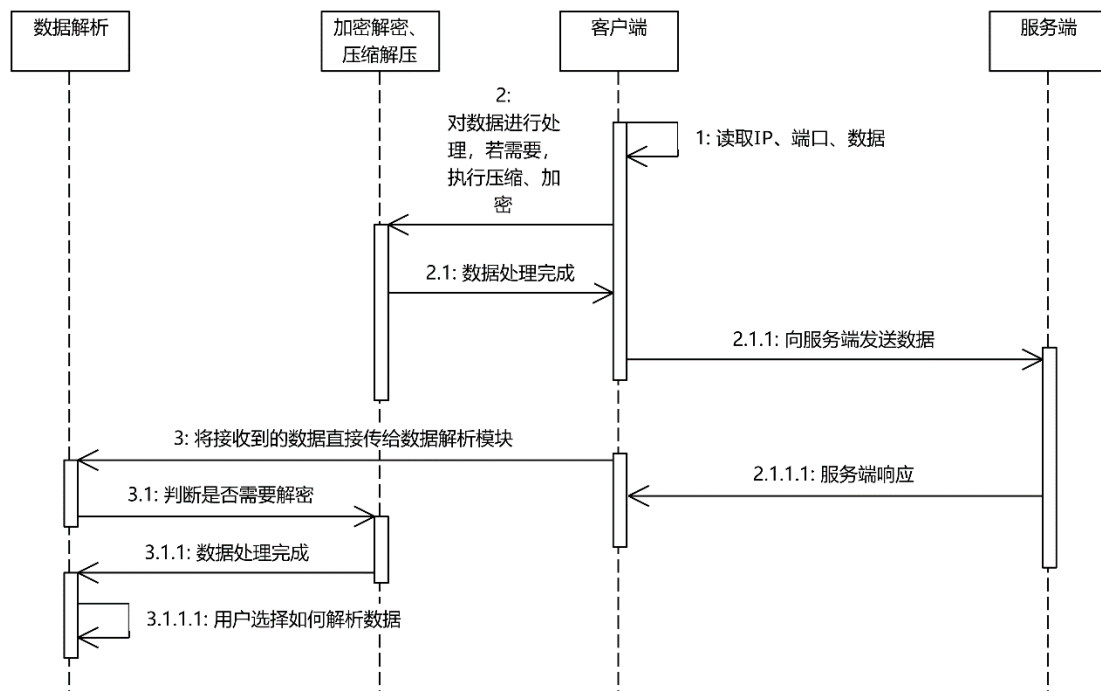


图 3.6 TCP 数据发送流程图

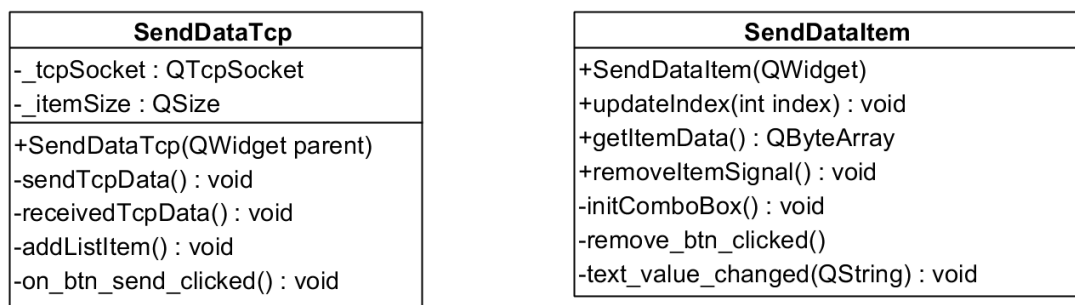


图 3.7 TCP 数据发送类图

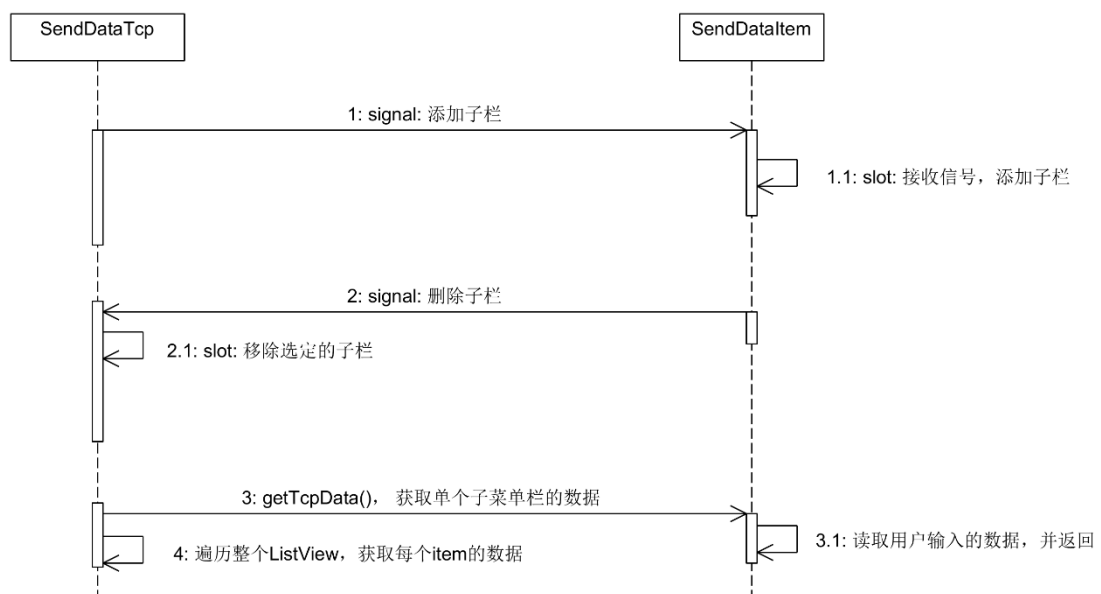


图 3.8 TCP 数据构建流程图

### 3.2.4 TCP 数据解析

在用户发送一条 TCP 消息后，收到回应会自动弹出 TCP 数据解析的界面，或者在后面的数据抓取功能中，双击单条数据，查看详细信息，也会弹出 TCP 数据解析的界面在解析 TCP 数据之前，如果发现用户已经选择或配置了压缩加密的算法，则先执行解密和解压的操作，且顺序不能乱，因为在加密和压缩一块使用的时候，是先进行压缩，再进行加密的，如果顺序弄反了，将不会得到正确的结果；解析之后的数据，默认会以字符串的形式出现'\0'字符默认以 ASCII 字符 '.' 来表示，如果用户需要更加查看数据中单个或多个字节，那么可以使用 TCP 数据查看选项来查看；

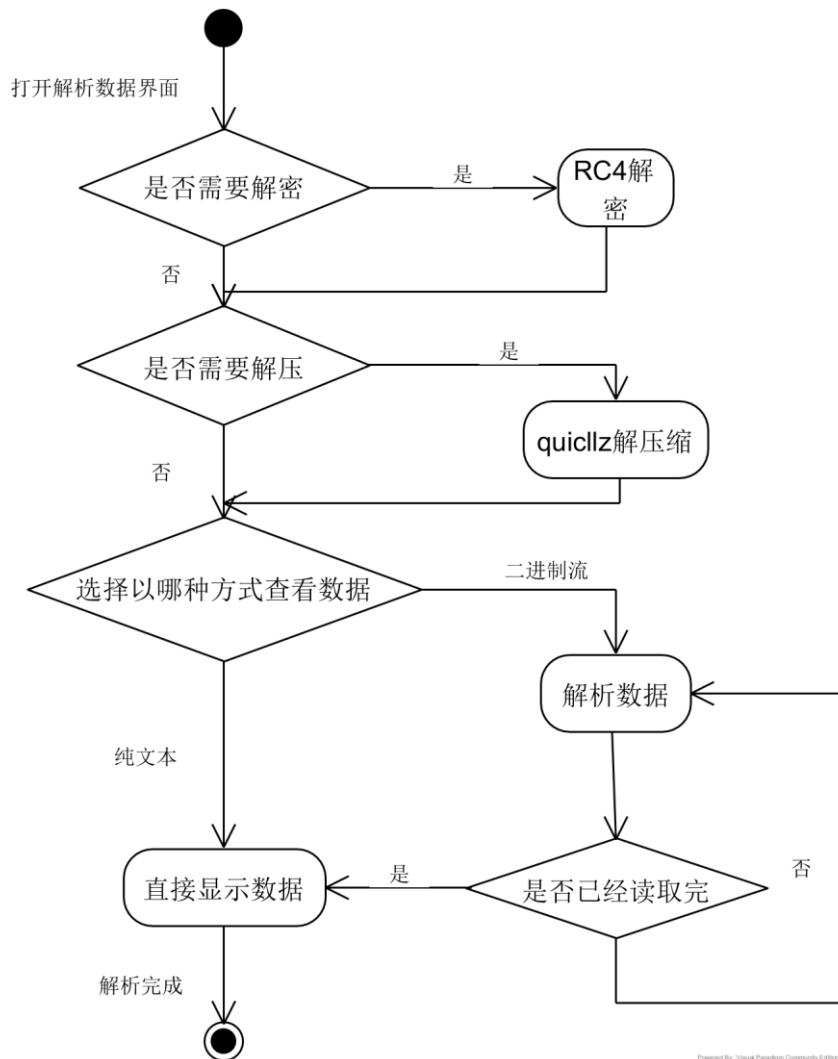


图 3.9 数据解析活动图

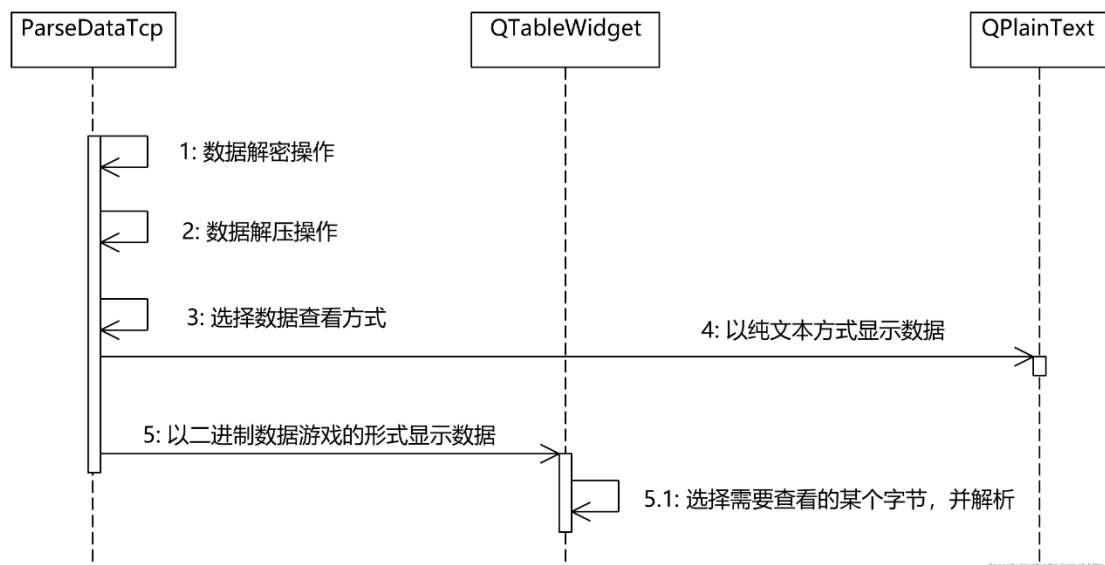


图 3.10 数据解析流程图

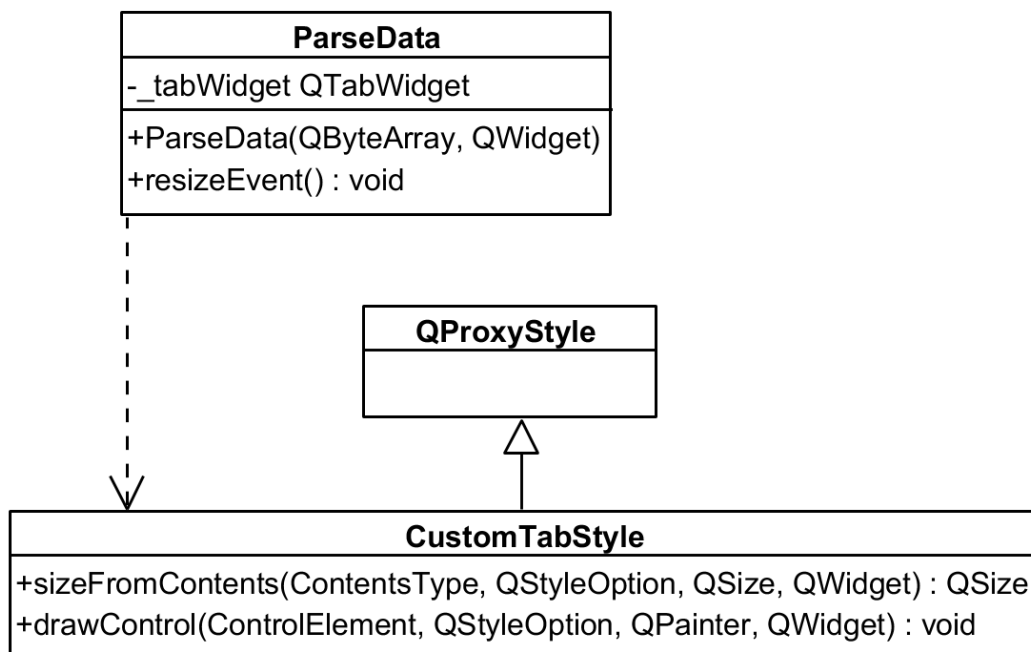


图 3.11 数据解析主界面类图

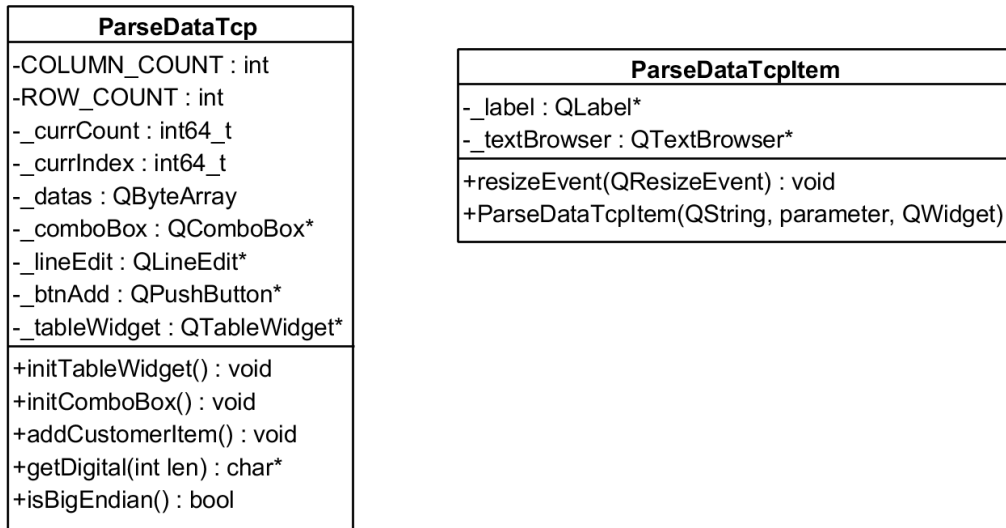


图 3.12 TCP 数据以二进制数据解析类图

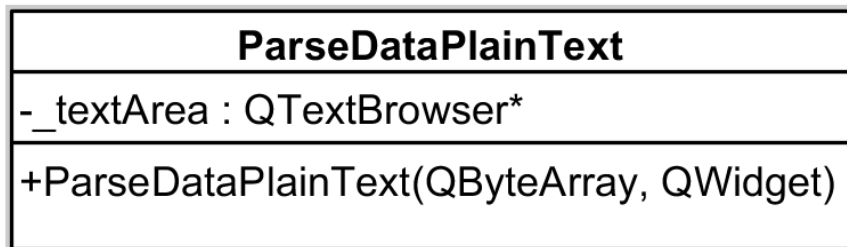


图 3.13 以纯文本数据解析类型

### 3.2.5 加密与压缩

要进行加密解密、压缩解压的操作，首先要确定是否需要进行此类型的操作，如果在不需要进行加密或压缩的情况下强行进行压缩操作，那么原本正常的的数据就会变得不可预测，所以加密压缩这些操作要在需要的时候才会执行；

这里的选择单例模式，可以在整个程序中共享一个全局的唯一对象，在这个对象中保存加密解密、压缩解压信息，在发送和解析数据的时候，通过这个全局唯一对象中存储的数据判断是否需要进行相应的操作，默认情况下是没有压缩与加密的操作，需要用户通过给出的图形接口来进行配置相应的加密与压缩算法；

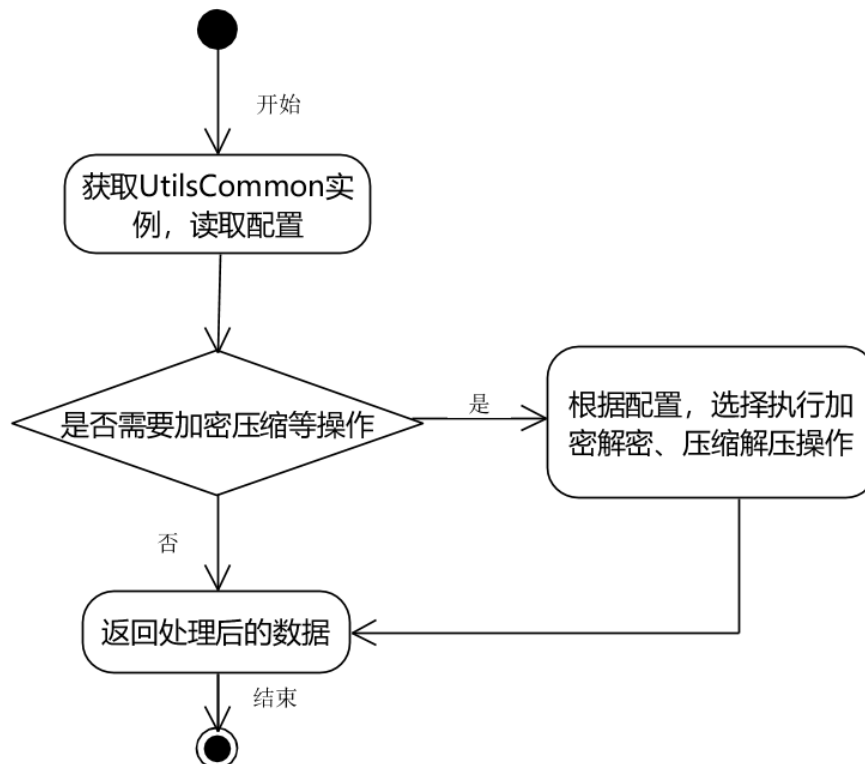


图 3.14 加密压缩主流程活动图

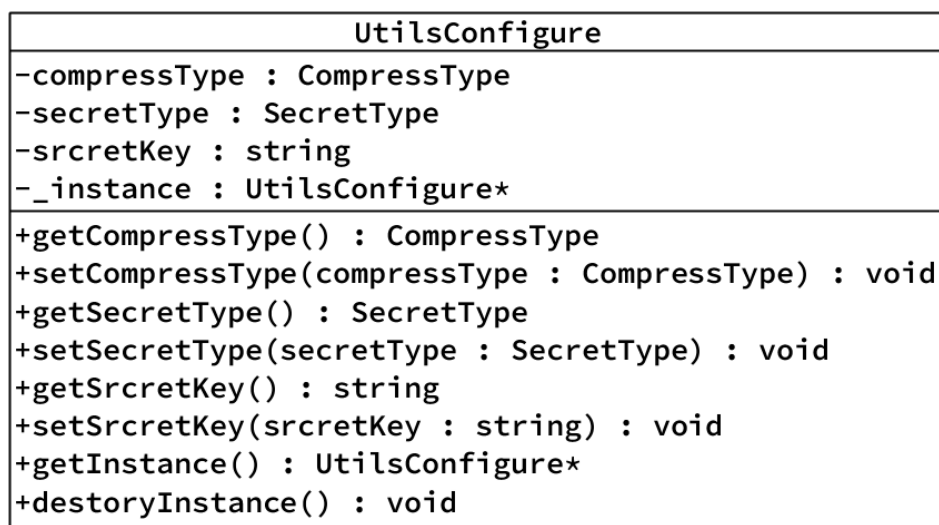


图 3.15 UtilsConfigure 单例类图

Quicklz 算法直接调用函数库，不需要进行多余的设计

RC4 没有专程的函数库，所以需要自己实现并封装好，方便使用；因为 RC4 加密与解密的操作完全一致，所以类型的设计中虽然有 encrypt 和 decrypt 两个方法，但是这两方法都是调用同一个方法来实现其它对应的功能的；

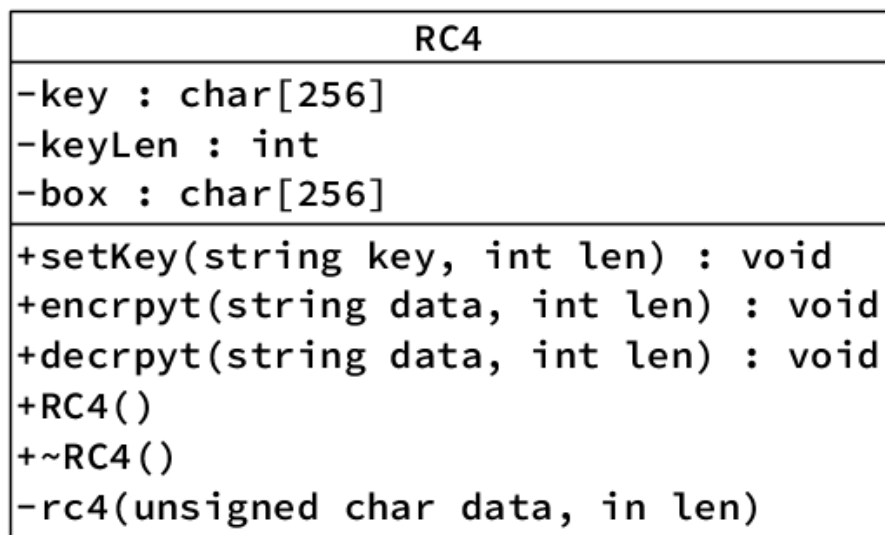


图 3.16 RC4 加密类的类图

### 3.2.6 数据抓取功能

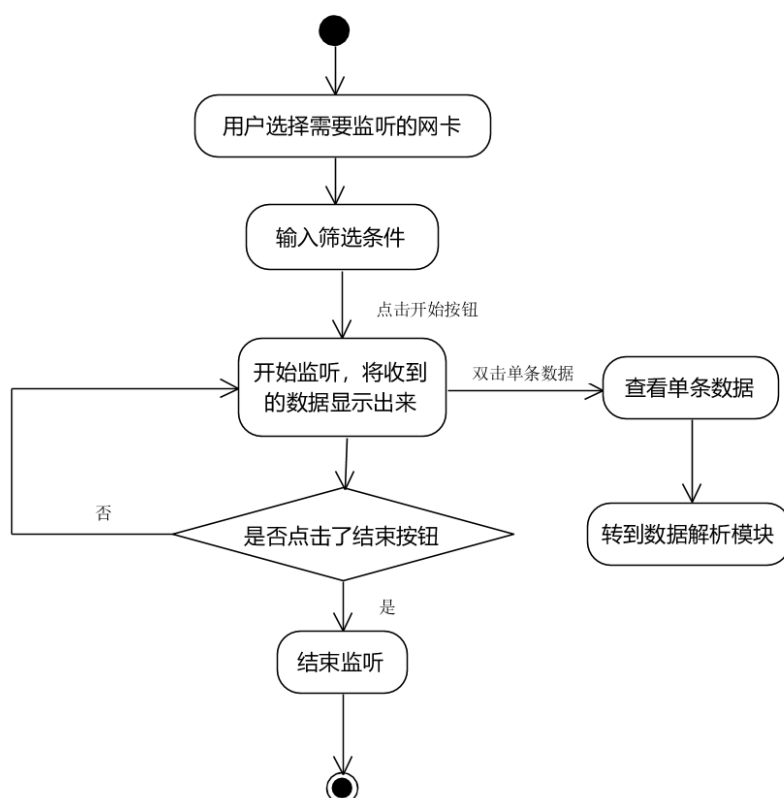


图 3.17 数据监听主流程图

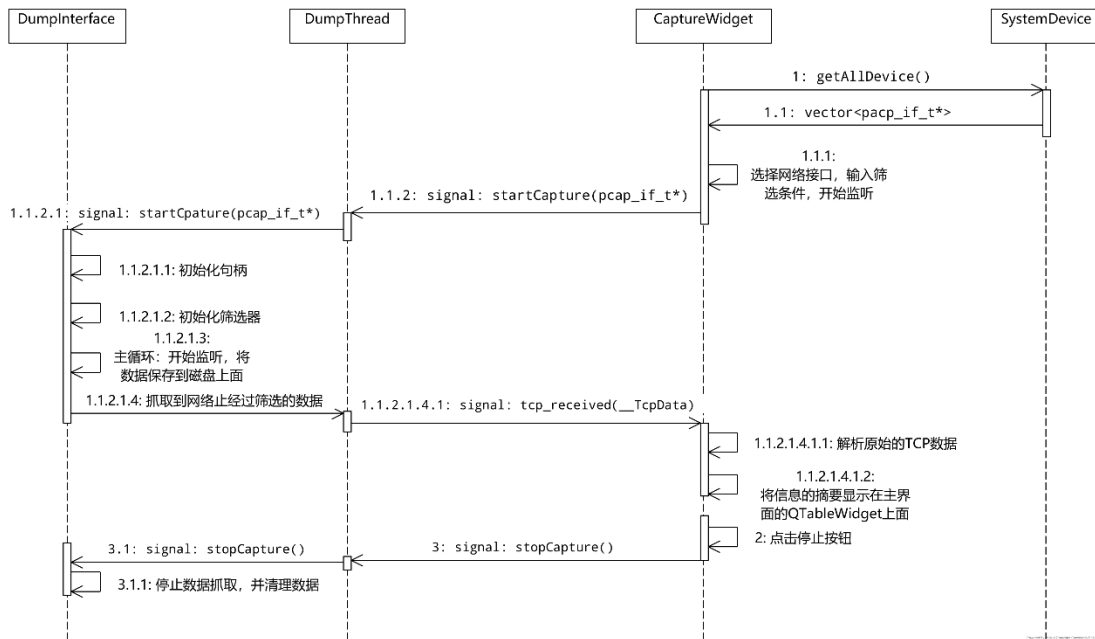


图 3.18 网络数据监听时序图

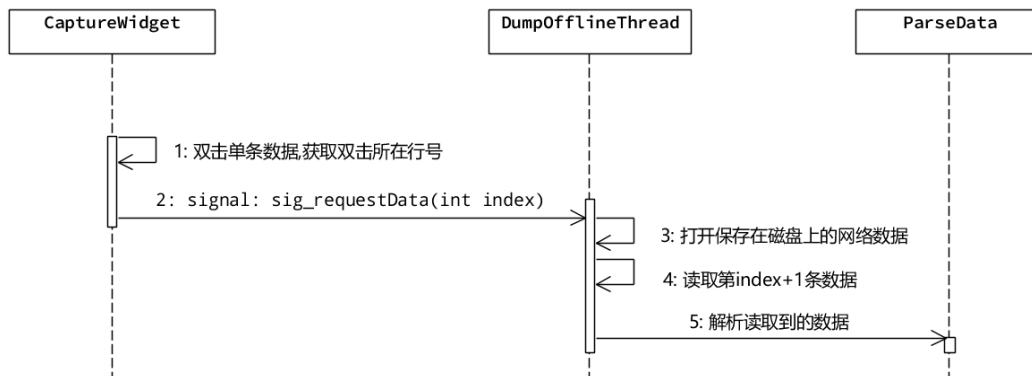


图 3.19 查看单条数据时序图

DumpInterface
-adhandle : pcap_t* -dump_file : pcap_dump_t* -isContinue : bool
+DumpInterface() +~DumpInterface() +capturePacket(pcap_if_t dev, string filter) : void +stopCapture() : void #on_tcp_received(__UdpData) : void #on_udp_received(__TcpData) : void -on_packet_received(pcap_pkthdr head, uchar data) : void -openDumpFile() : void

图 3.20 数据监听抽象接口 DumpInterface  
数据监听线程类，继承自 DumpInterface 抽象类和 QObject 类

DumpThread
+DumpThread(QObject parent) -on_tcp_received(__TcpData) : void -on_udp_received(__UdpData) : void +slot_startCapture(pcap_if_t device, string filter) : void +slot_stopCapture() : void +sig_tcpReceived(__TcpData) : void +sig_udpReceived(__UdpData) : void

图 3.21 数据监听线程类 DumpThread

新开一个线程类，读取保存在磁盘上的网络数据，继承自 QObject 类

DumpOfflineThread
-adhandle : pcap_t
+sig_readComplete(char data) : void +slot_request(int index) : void +~DumpOfflineThread() +DumpOfflineThread(QObject parent)

图 3.22 读取保存在磁盘上的网络数据类图

CaptureWidget
-ui : CaptureWidget -captureThread : DumpThread* -readDumpThread : DumpOfflineThread* -rbuff : char[64] -buff : char[64] -itemCounts : int
+CaptureWidget(QWidget parent) +~CaptureWidget() -on_btnStop_clicked() : void -on_btnStart_clicked() : void -on_btnAdvance_clicked() : void -deal_Tcp(__TcpData data) : void -deal_Udp(__UdpData data) : void -slot_showDetials(char data[]) : void -slot_requestDetials(int index) : void +sig_startCapture(pcap_if_t device, string filter) : void +sig_stopCapture() : void -integerToIp(int a) : string -getBrief(int tot_len, char[]) : char[] -initTableWidget() : void

图 3.23 网络数据监听主界面类图



## 第五章 软件详细设计

### 4.1 HTTP 数据发送

在 QT 的网络模块中，要实现 HTTP 数据的发送并不是通过网络连接来直接发送的，而是通过专门的管理类 `QNetworkAccessManager` 来实现网络数据的发送与接收，要请求的数据放在 `QNetworkRequest` 对象中，这样的设计可以让用户更加专注与应用的开发而无需过多的关心底层协议的实现；

首先完成如图 4.1 界面设计

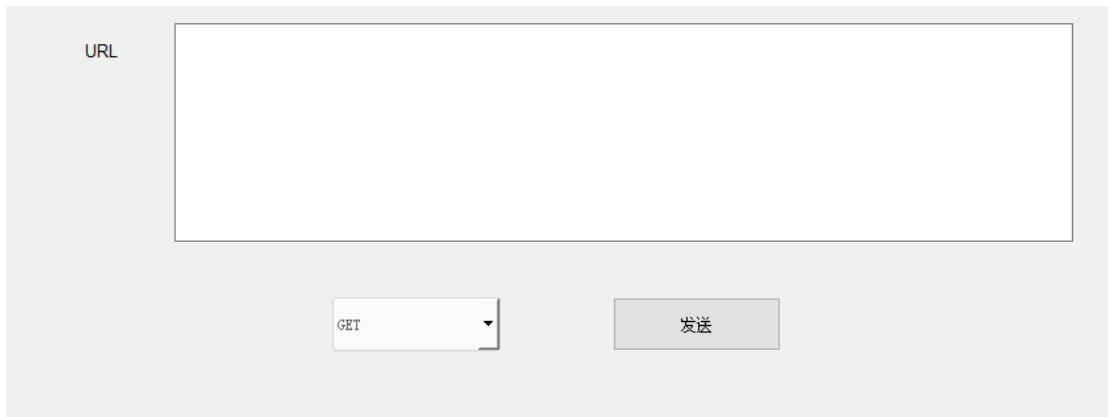


图 4.1 HTTP 数据发送界面

最大的那个白色的框是一个 `QPlainTextEdit` 对象，主要用于处理文本输入，在这个功能中的作用是读取用户输入的 URL 请求附带的参数；左下方的小下拉列表是一个 `QComboBox` 对象，一共有两个选项:GET 和 POST，用于让用户选择使用哪一种请求方式，右下角的发送按钮，是一个 `QPushButton` 对象，在用户输入好 URL 和参数，选择好请求方式，后点击发送按钮；这里程序会读取 URL 输入框的内容，以及 `QComboBox` 中的选择，以确定要使用哪一种请求方式；

发送 POST 请求：

```
void SendDataHttp::request_by_post(const QString &text)
{
    std::string str = text.toStdString();
    int pos = str.find_first_of('?');

    std::string url = std::string(str, 0, pos);
    std::string params = std::string(str, pos+1, str.size()-
                                    url.size()-1);
    QNetworkAccessManager *manager = new QNetworkAccessManager(this);
    QNetworkRequest request;
    request.setUrl(QUrl(QString(tr(url.c_str()))));

    manager->post(request, QByteArray(params.c_str(),
                                     params.size()));
}
```

```

        connect(manager, &QNetworkAccessManager::finished, this,
                &SendDataHttp::received_Data);
    }

```

发送 GET 请求,

```

void SendDataHttp::request_by_get(const QString &url)
{
    QNetworkAccessManager *manager =
        new QNetworkAccessManager(this);
    QNetworkRequest request;
    request.setUrl(QUrl(url));
    manager->get(request);
    connect(manager, &QNetworkAccessManager::finished, this,
            &SendDataHttp::received_Data);
}

```

收到 HTTP 响应后的处理, 收到 HTTP 数据响应扣, 根据 HTTP 头中所携带的 `HttpContent-Type` 信息来选择以哪一种文件格式来保存所收到的数据, 当保存完数据后, 新建一个进程, 使用 Windows 系统的资源管理器来打开保存在磁盘上的文件, 至于到底以哪一种方式来打开该文件, 完全由用户所使用的操作系统确定, 若没有安装打开此类型文件的应用, Windows 系统会提示用户选择以哪一种应用来打开此文件, 若实在找不到, 也可以选择联网查找; 比如说收到的数据是 html 文件, 那么 Windows 系统就会使用默认浏览器来打开该文件;

```

void SendDataHttp::received_Data(QNetworkReply *reply)
{
    if(!reply) {
        qDebug() << "QNetworkReply is nullptr";
    } else if(reply->error() == QNetworkReply::NoError){
        QString filename = "text";
        std::string all_header =
            reply->header(QNetworkRequest::ContentTypeHeader).toString(
                ).toStdString();
        /* http contentType 也可能分为多段, 用分号隔开 */
        std::string header = all_header.substr(0,
            all_header.find_first_of(';'));
        /* getSuffix 函数可以根据 HTTP 响应头中的 Content-Type 来确定文件以哪
            一种后缀的文件名来保存 */
        filename.append(getSuffix(header).c_str());

        QByteArray data = reply->readAll();
        QFile recv(filename);
        if(recv.exists()){
            recv.remove();
        }
        if(!recv.open(QFile::WriteOnly)){
            qDebug() << "open file error";
        }
    }
}

```

```

    }
    recv.write(data);
    recv.close();
    QProcess::execute("explorer.exe", QStringList() << filename);
} else {
    qDebug() << reply->errorString();
}
}
}

```

## 4.2 TCP 数据发送

与传统的 TCP 数据发送方法一样，QT 的 TCP 数据发送功能也是通过新建一个套接字来实现的，并用该套接字来发送网络数据，但是与之不同的是 QT 的套接字实现了自己的封装，且在入了信号与槽，更加容易使用；

如图 4.2 为 TCP 数据发送的界面图，主机输入框是一个 QLineEdit 对象，用于输入目标 IP 地址，端口输入框也是一个 QLineEdit 对象，右边的发送按钮和高级按钮都是 QPushButton 对象，点击高级按钮后，会弹出让用户选择加密和压缩算法的一个对话框，点击确定后，就会使用当前的加密和压缩生效；点击发送按钮后就会读取当前的配置，用户输入的目标主机、端口、以及构建的数据（若需要再进行压缩、加密操作），之后向目标主机发送数据，收到响应的数据后，会自动打开数据解析界面（数据解析界面将会在后面介绍）；

界面的再下面一点，标有数字 1、2 和一个添加按钮的地方是一个 QListWidget 对象，在 QListWidget 面板中，放入了一个 item，item 中有一个大的添加按钮，一起在 QListWidget 的最下方，点击添加按钮则在添加按钮的上面一格插入一个 item 其格式与标号 1、2 的一样，点击 item 右边的删除按钮，则会删除对应的 item，并更新 item 最左方的序号；



图 4.2 TCP 数据发送界面

底部添加按钮实现

```

_btnAdd = new QPushButton(tr("添加"));
QListWidgetItem *item_button = new QListWidgetItem(_listWidget);
_listWidget->setItemWidget(item_button, _btnAdd);

```

```

_listWidget->addItem(item_button);
connect(_btnAdd, &QPushButton::clicked, this,
&SendDataTcp::addListItem);
添加 item 函数（只有核心部分的代码，细节见源码）
void SendDataTcp::addListItem()
{
    QListWidgetItem *item = new QListWidgetItem();

    SendDataItem *widget =
        new SendDataItem(_listWidget->count(), _itemSize);
    _listWidget->insertItem(_listWidget->count()-1, item);
    _listWidget->setItemWidget(item, widget);
    _listWidget->scrollToBottom();
    connect(widget, &SendDataItem::removeItemSignal, this, [=]() {
        _listWidget->removeItemWidget(item);
        if(nullptr != item) delete item;
        if(nullptr != widget) delete widget;
    });
}

```

网络数据传输部分

```

/* socket 初始化 */
_tcpSocket = new QTcpSocket(this);
connect(_tcpSocket, &QTcpSocket::connected,
        this, &SendDataTcp::sendTcpData);
connect(_tcpSocket, &QTcpSocket::readyRead,
        this, &SendDataTcp::receivedTcpData);

.....
/* 连接到指定主机：端口 */
_tcpSocket->connectToHost(QHostAddress(host) , port);

.....
/* 发送用户构造的数据 */
QByteArray bytes;
for(int idx = 0; idx < _listWidget->count(); idx++) {
    SendDataItem *item = dynamic_cast<SendDataItem*>(
        _listWidget->itemWidget(_listWidget->item(idx)));
    if(nullptr == item) continue;
    bytes.append(item->getItemData());
}
/* 如果需要，这里执行压缩加密的操作 */
_tcpSocket->write(bytes);

.....
接收数据，并调用 ParseData 模块，解析收到的数据，并关闭连接
QByteArray bytes = _tcpSocket->readAll();
ParseData *parseData = new ParseData(bytes);

```

```
parseData->show();
_tcpSocket->close();
```

单条数据构造部分：在本次设计中，为单条数据的构造专门创建了一个类，用来 构造数据，在需要的时候，返回构造完成的单条数据，



图 4.3 单条数据界面

如图 4.3 最左边为序号，方便用户查看当前一共输入了多少条数据，具体第几条的数据是多少，这一点非常重要，而且界面做过了特殊处理，在用户删除单条 item 时，会更新所有的序号，使其变得连续且有序，紧跟序号之后的是一个下拉菜单，用户可以通过这个下拉菜单来选择需要发送的数据类型，其中包括了现在会用到的所有基本数据类型 char、short、int、long、float、double，当然也加入了字符串的支持；在中间的输入框中，需要输入用户相应的数据，记住，创建了一个 item 是必须要输入数据的，不然发送的数据可能会与预期的不一样，而软件不会做出提示；当用户点击发送按钮后，程序会遍历所有的 item，通过 item 的对象获取存放在 item 中的用户输入的数据，并将所有的数据存起来，再处理，发送；

但是这里有一个很大的问题，那就是 float 的数据类型，与其它数据会有所不同，比如说 0x12345678 和 0x11223344 这两个数据在 windows 上是这样存储的（数据高位在内存低位，数据低位在内存高位，也就是小端字节序）



图 4.4 小端字节充在内存中的存放

但是 float 数据的存取与这个存法完全相反，它是数据高位在内存地址高位，数据低位在内存地址低位，完全是按照大端序的存法来的，这里我不知道是 QByteArray 和 QDataStream 的问题还是 float 本身是这样的（或者是编译器实现的问题），所以我对 float 的数据类型做了特殊处理，在返回字节流的时候，我把它的数据反过来存：

```
char ptr[4];  memset(ptr, 0x00, sizeof(ptr));
float f = value.toFloat();
memcpy(ptr, &f, sizeof(float));
for(int i=3; i>=0; i--) bytes.append(ptr[i]);
```

关于字节序的大端与小端，在 TCP 数据解析的时候会专门说明；

4.3 TCP 数据解析

TCP 数据解析这个部分，相对而言要简单一些，它的功能就是给它一个 QByteArray，然后按照指定的方式显示出来就可以了，不过显示的方式分两种：第一种是直接以纯文本字符串来显示，不可见字符或 ‘\0 ‘的结束符用 ‘.’ 来代替：

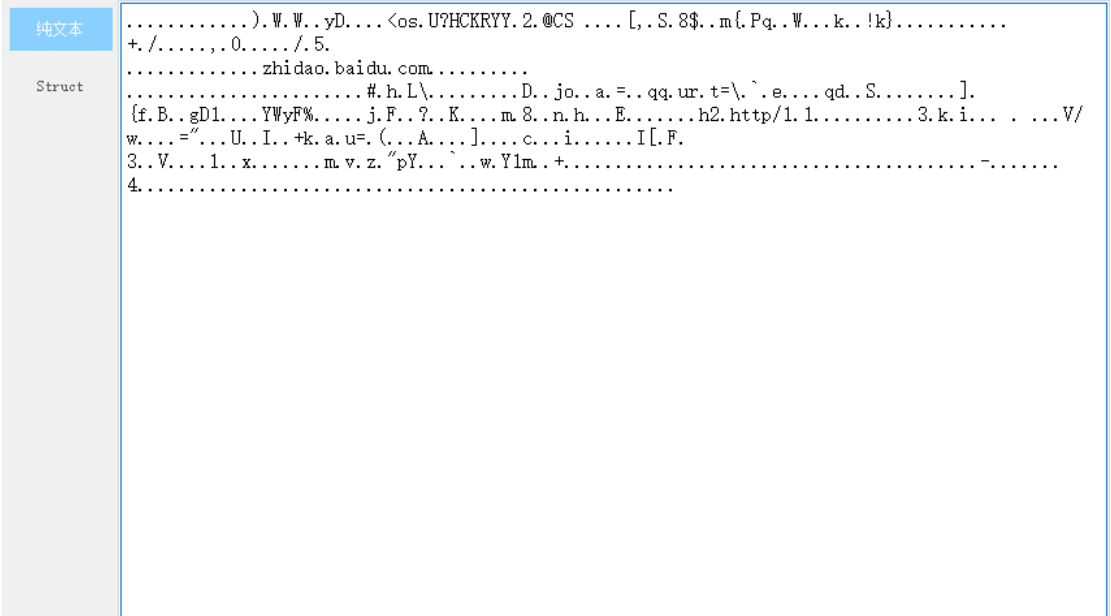


图 4.5 以纯文本的方式解析数据

另一种方式是接收到的数据如果是原始的非字符串的二进制数据流，那么就需要用专用的显示方法来显示了，点击界面左侧栏上的 struct，就进入了二进制数据流解析的过程中了：

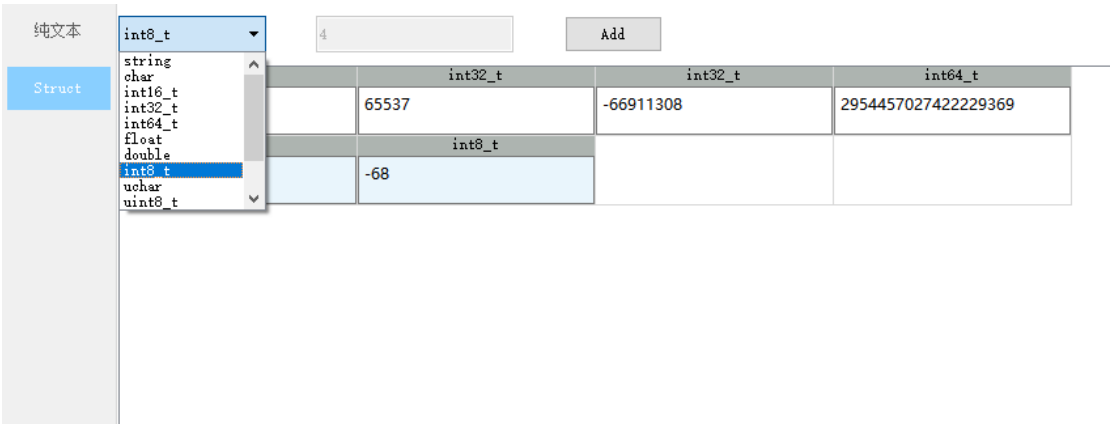


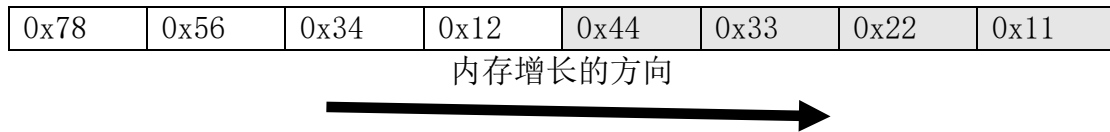
图 4.6 解析非字符串的二进制数据流

数据解析的界面如图 4.6，与 TCP 数据的构造类似，用户也可以选择将要解析的数据类型，选择之后点击 add 添加，下方的 QTableWidgetItem 就会多出一个 Item 用来显示解析也来的对应数据，如果要查看指定位上的数据，可以先读取一个 string，设置好长度，点击添加，这个时候，数据就偏移了刚刚设定的位数，再读取自己想要的数据就可以了；

但是在实现的时候会有大问题，因为数据在网络中的传输作用的是大端字节序(big endian)，而在 windows 等主流的操作系统上，数据在内存中是以

小端字节序(little endian)来存放的，直接举个例子：

两个四字节的数据 0x12345678 和 0x11223344 以小端的方式来存储，在内存中是这样的：



而以大端方式来存是这样的：

0x12	0x34	0x56	0x78	0x11	0x22	0x33	0x44
------	------	------	------	------	------	------	------

矛盾显而易见，网络中传输用大端，计算机内传输用小端，这两个读出来根本不是同一个东西，所以我们需要对机器的大小端进行判断，之后再选择以大端还是小端来处理数据，isLittle 函数，如果机器是小端字节序，那么返回 true，如果机器是大端字节序，返回 false；

```
inline bool isLittleEndian ()
{
    int32_t a = 1;
    char *p = (char *)&a;
    return (1 == *p);
}
```

要解析成指定的数据，首先要从原来的数据中读取指定长度位的数据，才能进行解析，同时在读取指定长度时，还要考虑大端字节序和小端字节序的问题，所以这里封装了一个函数，专门用来处理这个问题，该函数首先判断数据是否已经解析完了，如果已经解析完，直接返回，没有解析完成，才进行接下来的操作，从网络数据中读取指定位数，然后完成转化，返回一个 char\* 的指针，里面存放了解析完成的原始数据：

```
char* ParseDataTcp::getDigital(int length)
{
    if(length + _currIndex > _datas.size()) return NULL;
    int r_idx;
    char *ret = new char[16];
    memset(ret, 0x00, 16);
    if(!isLittleEndian()) { /* big endian */
        for(r_idx=0; r_idx<length; r_idx++) {
            ret[r_idx] = _datas.at(_currIndex + r_idx);
        }
    } else { /* little endian */
        for(r_idx = length-1; r_idx>=0; r_idx--) {
            ret[r_idx] = _datas.at(_currIndex + length - r_idx - 1);
        }
    }
    _currIndex += length;
    return ret;
}
```

如果要解析 int 类型的数据，直接使用字符串转 int 函数 atoi 即可完成，最后再将数据和数据类型在 QTableWidgetItem 上面显示出来；

## 4.4 数据压缩解压、加密解密

RC4 算法的特点是算法简单，运行速度快，而且密钥长度可变(1-256 字节)，在如今的技术下，当密钥的长度达到 16 字节时，用暴力搜索算法已经不太可行，所以 RC4 在现阶段而言是相对比较安全的对称加密算法；

在实现 RC4 算法之前有必要先介绍一下算法中涉及的基本东西；

密钥流：RC4 算法的关键是根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是一样的，加密生成的密文长度也是一样的，因为密文的第  $i$  字节 = 明文第  $i$  字节 ^ 密钥流的第  $i$  字节；

状态变量 S：长度为 256，第个单元都是一个字节，算法运行在任何时候，S 都包括 0~255 的 9 比特数的排列组合，只不过数值的位置发生了变化；

密钥 K：度为 1~256 字节，也就是所说的密码，通常取 16 位；

加密步骤：设置密钥和密钥流

```
void RC4::setKey(const char *key, int keyLen)
{
    _keyLen = keyLen;
    memcpy(_key, key, keyLen);

    unsigned char k[MAX_LEN] = { 0 };
    for (int i = 0; i < MAX_LEN; i++) {
        _box[i] = i;
        k[i] = _key[i % _keyLen];
    }
    for (int i = 0, j = 0; i < MAX_LEN; i++) {
        j = (j + _box[i] + k[i]) % MAX_LEN;
        std::swap(_box[i], _box[j]);
    }
}
```

RC 加密与解密，因为 RC4 加密算法的原因，其实它的加密解密完全是同一个步骤：

```
void RC4::rc4(unsigned char * data, int len)
{
    unsigned char tBox[MAX_LEN] = { 0 };
    memcpy(tBox, _box, MAX_LEN);

    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    for (k = 0; k < len; k++) {
        i = (i + 1) % MAX_LEN;
        j = (j + tBox[i]) % MAX_LEN;
        std::swap(tBox[i], tBox[j]);
        t = (tBox[i] + tBox[j]) % MAX_LEN;
        data[k] ^= tBox[t];
    }
}
```



```
}
```

至于数据压缩算法，直接从网上下载相应的加密库在程序中调用即可，下面列出压缩算法所会用到的方法：

数据压缩算法：destination 必需是已经分配内存的指针

```
size_t qlz_compress(const void *source, char *destination, size_t
size, qlz_state_compress *state);
```

传入压缩后的数据，即可获得解压后数据的大小，可以决定分配多大的内存：

```
size_t qlz_size_decompressed(const char *source);
```

数据解压函数：destination 必须是已经分配内存的指针，并用分配空间的大小必须要大于或等于 qlz\_size\_decompressed 的返回值：

```
size_t qlz_decompress(const char *source, void *destination,
qlz_state_decompress *state);
```

## 4.5 数据监听

数据监听这个功能是这个软件实现中最为复杂的一个部分，数据监听函数一旦开始执行，那么它的主循环便会一直执行，并不会退出，听起来好像没有什么问题，如果把数据监听的线程与主界面放在同一个线程中的话，那就意味着主界面中的代码会一直处于阻塞状态，在图形界面中这个会弹出程序未响应的提示框，用户点什么操作它都不会响应；唯一的解决办法就是把这个会阻塞主线程的函数放到一个单独的子线程中，有什么状态需要改变的时候，再发送信号通知子线程进行相应的操作，或者子线程发送消息给主线程；另一个问题是网络中的数据可能会很大，如果要把这些数据全部存放在内存中，那么数据太多的时候，内存会完全不够用，从而导致系统崩溃，所以这里的解决方法是每收到一条数据，全将数据保存到磁盘中，需要的时候再去读取；最后一个问题是这个文件是两个线程共用的，所以可能会出现等待的问题，也有可能使主界面假死的情况，所以又需要新加一个线程专门用来读取在磁盘中的文件；

综合上面的分析，除主线程外，还需要单独创建两个线程，用与数据监听和磁盘文件读取；

现在程序的框架已经出来了，那就是在主界面创建完成之后，再新开两个线程，一个用于数据监听，另一个用于读取保存在本地的网络数据文件；数据监听该怎么来实现呢？直接从硬件获取？还好已经有现成的 API 可以帮我们完成这一件事情——winpcap，作为一个开源的程序库，winpcap 提供了几个非常强大的功能：捕获原始的数据包，包括在共享网络中各主机发送/接收的以及相互之间交换的数据包；对于网络中发来的数据，可以对数据进行自定义的筛选；也可以发送原始的数据包；对网络中的数据进行分析与统计；

在这次实现的软件中，主要使用的是 winpcap 的数据捕获能力和网络数据筛选的功能；

网络接口		Intel(R) 82574L CFQ		高级选项	
主机		端口		最小长度限制	
开始					
	IP+端口	长度	协议	简要信息(双击单条信息查看详情)	
49	192.168.80.129:49844->13.250.162.133:443	1			
50	13.250.162.133:443->192.168.80.129:49844	0			
51	192.168.80.129:49859->14.17.41.155:8080	0			
52	14.17.41.155:8080->192.168.80.129:49859	0			
53	192.168.80.129:49859->14.17.41.155:8080	0			
54	192.168.80.129:49859->14.17.41.155:8080	324		POST / HTTP/1.1..Host: 14.17.41.155:8080..Accept: */*..Content	
55	14.17.41.155:8080->192.168.80.129:49859	0			
56	14.17.41.155:8080->192.168.80.129:49859	137		HTTP/1.1 200 ok.cmd: 0..Content-Length: 55..encrypt: false.j	
57	192.168.80.129:49859->14.17.41.155:8080	0			
58	14.17.41.155:8080->192.168.80.129:49859	0			

图 4.7 网络数据监听界面

界面中显示了数据包的数量，数据包是第几个收到的，数据报文的源地址和目的地址，TCP 上层应用层协议的数据长度，以及数据的简要信息；所有的数据都放在一个 QTableWidget 中间，双击单条数据即可查看数据的详细信息，该功能屏蔽了网络底层实现的细节，让用户更加专注于应用层传输的内容；

wincap 是一个第三方的库，使用前一定要 include 头文件，在工程文件.pro 中加入.lib 库，这样才能正常编译：

```
# wincap
INCLUDEPATH += $$PWD/wincap/Include
LIBS += $$PWD/wincap/Lib/wpcap.lib
LIBS += $$PWD/wincap/Lib/Packet.lib
```

另外，由于 wincap 还引用了 winsock 的内容，所以需要链接 winsock 的 lib 库，在.pro 文件中加入以下内容（有#号的行代表注释）：

```
# ws2_32.lib Windows Socket
LIBS += -lws2_32
```

要抓取网络数据，首先通过 wincap 提供的 API 获取所有的网络设备：

```
vector<pcap_if_t*> SystemDevice::init()
{
    pcap_if_t *devs;
    char errbuff[PCAP_ERRBUF_SIZE + 1];
    if(-1 == pcap_findalldevs(&devs, errbuff)) {
        return false;
    }
    vector<pcap_if_t*> _allDevs;
    for(pcap_if_t *dev = devs; dev; dev = dev->next) {
        _allDevs.push_back(dev);
    }
    return _allDevs;
}
```

当网络接口获取完成后，所有的网络接口信息将会在界面上的下拉菜单中显示，这时用户可以通过下拉菜单来选取抓取哪个接口上的数据；选择数据后

在下面的筛选选项中可以对网络中的数据进行筛选，筛选完成后，就可以点击开始按钮进行数据抓取了，下面给出了数据抓取的核心代码：

```
void DumpInterface::capturePacket(const pcap_if_t *dev)
{
    const char *devName = dev->name;
    char errbuf[PCAP_ERRBUF_SIZE + 1];
    _adhandle = pcap_open_live(devName, 65536, 1, 1000, errbuf);
    /* build filter */
    struct bpf_program fcode;
    pcap_compile(_adhandle, &fcode, "tcp and ip", 1, 0x00ffffff);
    pcap_setfilter(_adhandle, &fcode);
    /* start capture */
    int res = 0;
    pcap_pkthdr *pkt_header;
    const u_char *pkt_data;
    while(!_isContinue && 0 <= (res = pcap_next_ex(_adhandle,
                                                &pkt_header, &pkt_data))) {
        if(0 == res) continue;
        on_packet_received(pkt_header, pkt_data);
    }
}
```

从网络上抓取到数据之后，就会调用 `on_packet_received(pcap_pkthdr *, u_char*)` 这个函数，处理发来的数据，根据网络中发来的不同类型的协议，作不同的处理，它会自动忽略掉除 TCP 和 UDP 以外的数据报文，当接收到数据后，它会先根据 14 位的 MAC 长度计算出 IP 数据头的起始位置，再根据 IP 头的头部长度的信息计算出 TCP 数据报的头，再根据 TCP 的数据头部的长度计算出真实的数据起始位置，TCP 上层协议的数据长度可以用 IP 数据报的长度减去 IP 头部长度+TCP 头部长度；

跟之前一样，因为在 X86 体系的机器上，都是小端字节序，所以对网络中发回来的数据，需要将大端转为小端，以下分别是大端 16 位转小端和 32 位转小端的高效方法（核心代码，具体细节见源码）：

```
inline uint16_t __ntohs(uint16_t x) {
    return ((x >> 8) & 0x00ff) | ((x & 0x00ff) << 8);
}
inline uint32_t __ntohl(uint32_t x) {
    return (((x) & 0xff000000) >> 24) | (((x) & 0x00ff0000) >> 8) |
           (((x) & 0x0000ff00) << 8) | (((x) & 0x000000ff) << 24);
}
```

还有就是在此功能模块中用到的多线程技术，QT 中多线程有多种使用方法，这里采用 QT 官方较为推崇的写法：

- 将要执行的耗时操作放到一个继承自 `QThread` 类的子类中；
- 关联必要的信号与槽，用于和主线程之间通信；
- 创建一个新的 `QThread` 类对象和我们的线程类对象；
- 将要执行的线程移交到 `QThread` 对象中去；

调用 `QThread::start()` 方法，开始执行子线程；

```
_captureThread = new DumpThread(); // DumpThread 继承自 QThread
connect(_captureThread, &DumpThread::sig_tcpReceived, this,
&CaptureWidget::deal_Tcp, Qt::QueuedConnection); //主线程与子线程通信
QThread *mainLoopThread = new QThread(); // 创建一个子线程
_captureThread->moveToThread(mainLoopThread); // 将子线程移交到新的线程
mainLoopThread->start(); // 开始新的线程
```

剩下的就都一些界面方面的代码，太大的技术难点要看实现细节的参照源码；

## 第六章 系统测试

测试的目的是确保系统可以在任何环境下执行正常操作，并检测系统操作期间可能出现的任何问题。软件开发者在开发过程中不可避免地会产生错误。要充分测试系统可以有效保证软件的质量。本章这次设计的软件进行功能测试，并分析测试结果；

### 5.1 测试环境简介

本机：软件运行在我自己的电脑上，Intel i5 处理器，华硕主板，运行了 Windows10 最新版本，系统未经过任何修改，未安装任务电脑管家和杀毒软件；

远程主机（IP 为 120.79.224.255）：为了提供一个真实的网络测试环境，这次测试使用的远程主机采用阿里的云的轻量级云服务器，服务器上运行着正版的 Windows Server R2 的操作系统，为了本次的测试，特意开放了 8080 端口和 9999 端口（由于服务器的学生认证即将过期，所以并不保证服务器在今后还能正常访问）；

8080 端口上运行了 tomcat 服务器，所有发送的 HTTP 请求，都将会重定向到一个 JSP 页面，这个 JSP 页面会将所有的请求参数格式化为参数名+参数附带的值的形式呈现出来，并显示请求的方式；另一个 9999 端口运行着一个用 java 写的服务端程序，它将所有的请求数据原样发回，不作任何修改；

### 5.2 各功能模块的测试

这个软件在设计的时候，所有的功能都是互不干扰的，也就是说所有的功能都可以同时使用而互不干扰，所以除了对各个功能的单独测试外，还需要测试多个功能同时使用时的表现情况；该软件测试主要分为 HTTP 数据发送测试、TCP 数据发送测试、数据加密测试、数据压缩测试、数据抓取测试、TCP 加密压缩数据发送测试、抓取的数据解压解密测试几个部分，因为 TCP 数据发送加密压缩都是在一个不可拆分的功能所在就放在 TCP 数据发送测试中一起测试；而数据抓取和解压解密也是不可拆分的，所以也放在一起测试；综上软件测试主要分为以下三个部分的测试，下面将给出详细的测试过程；

### 5.2.1 HTTP 数据发送测试

点击上方三个选项中的 HTTP 选项，进入 HTTP 数据发送的界面，在 URL 框内输入要请求的 URL，格式为网页页面链接+要请求的参数，GET 和 POST 请求的格式完全一样，例如 <http://example.com:port/index.html?param=2&param2=2>：



图 5.1 HTTP-GET 请求发送

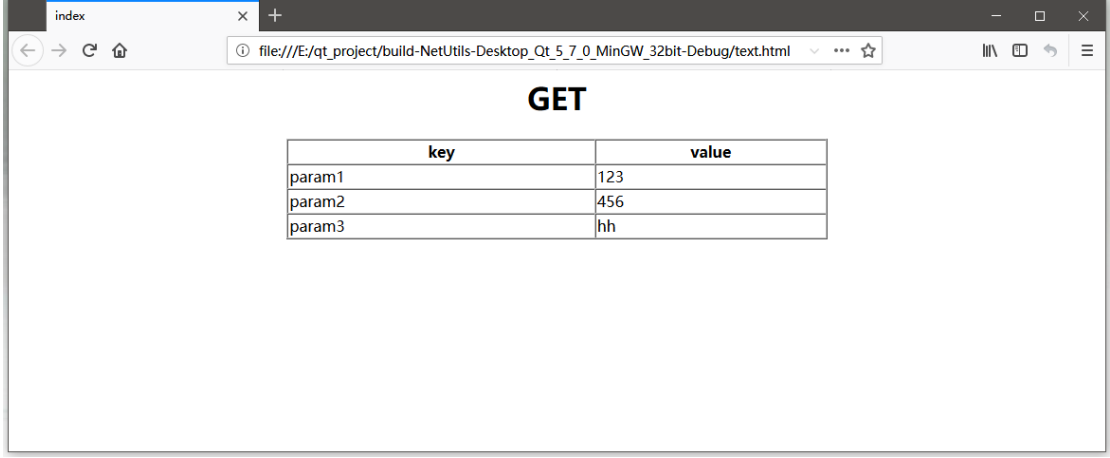


图 5.2 HTTP-GET 请求响应

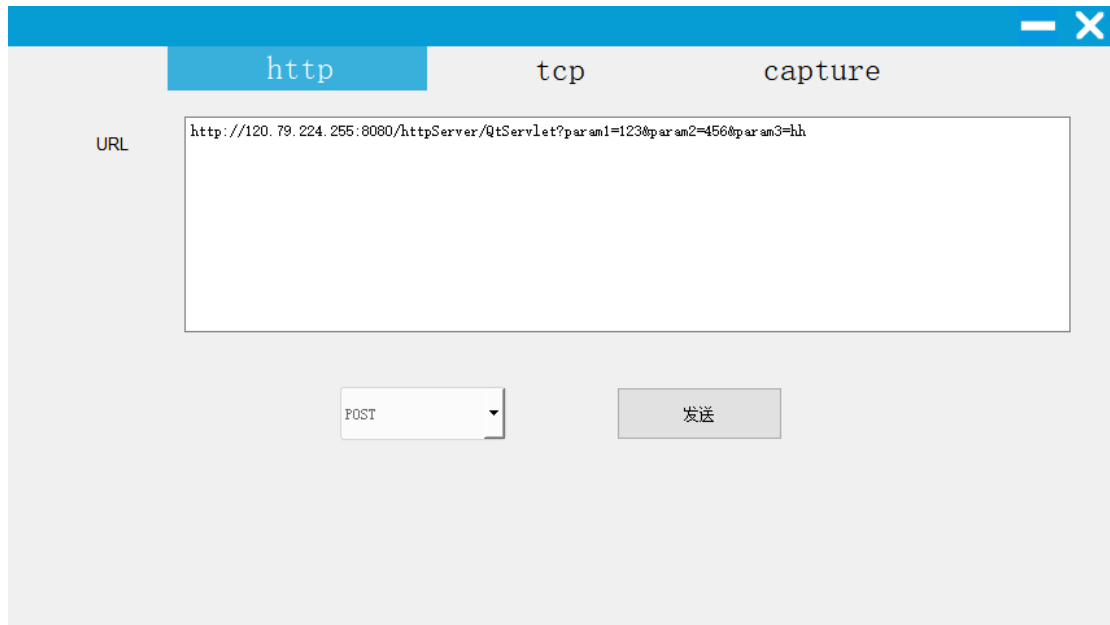


图 5.3 HTTP-POST 请求发送

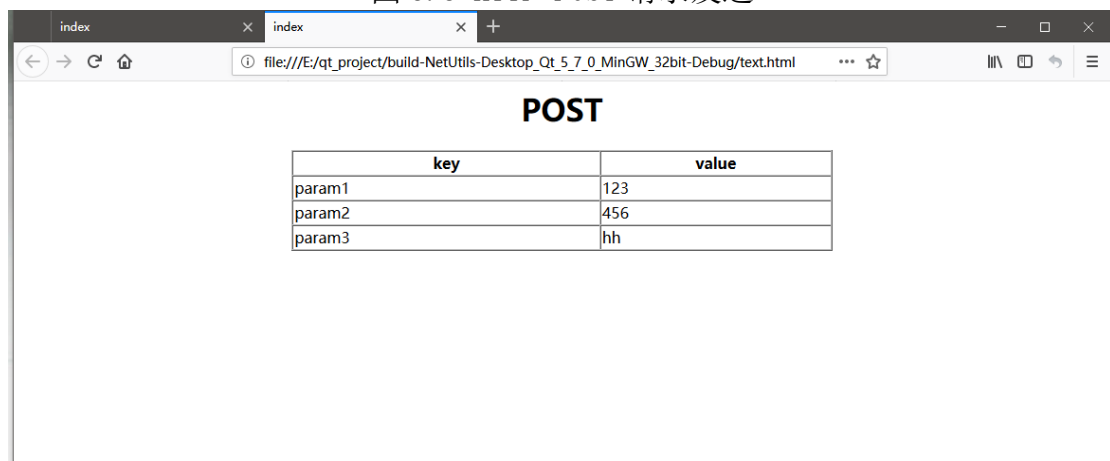


图 5.4 HTTP-POST 请求响应

### 5.2.2 TCP 数据发送接收测试

因为 TCP 数据的发送与接收本为一个整体，不可再拆分，所以放在一起测试：

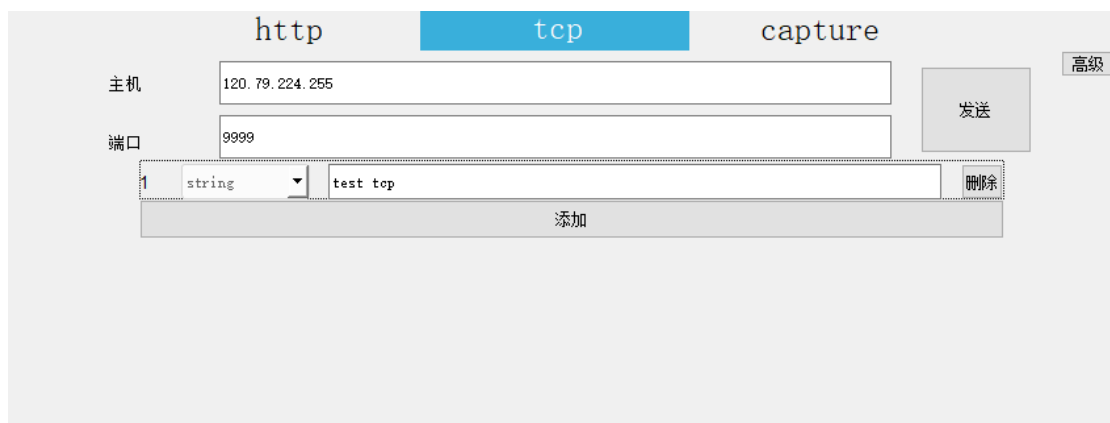


图 5.5 发送 TCP 数据

纯文本

test tcp

Struct

图 5.6 接收 TCP 数据

5.2.3 TCP 数据构造与解析测试

TCP 数据的构造与解析也是两个对应的功能，所以放在一起测试，所有的数据类型都使用一次，再发送：

1	string	test tcp	删除
2	char	c	删除
3	int8_t	-8	删除
4	int16_t	-16	删除
5	int32_t	-32	删除
6	int64_t	-64	删除
7	double	123213.342573894	删除
8	u_char	u	删除
9	uint8_t	8	删除
10	uint16_t	16	删除
11	uint32_t	32	删除

图 5.7 TCP 数据构造-1

12	uint64_t	64	删除
13	float	21334.222	删除
添加			

图 5.8 TCP 数据构造-2

纯文本

Struot

test tcp.....@...{.su.....@F..r

图 5.9 以字符串方式解析

纯文本

Struot

float

8

Add

string	char	int8_t	int16_t
test tcp	c	-8	-16
int32_t	int64_t	double	uchar
-32	-64	123213.342574	u
uint8_t	uint16_t	uint32_t	uint64_t
8	16	32	64
float			
21334.222656			

图 5.10 以发送的数据格式解析

可以看出，构造的数据和解析完成的数据完全一样，数据构造与解析功能无误

### 5.2.3 数据加密与解密测试

因为测试用的服务端只是一个数据跳转站，并不修改实际的数据，所以如果发出去的是加密数据，那么接收回来的数据也是加密的数据；





同的，此时可以证明数据是经过加密处理的，且解密是成功了；

5.2.4 数据压缩与解压测试

压缩算法

quicklz

加密算法

none

取消

确定

图 5.15 设置数据压缩算法

http

tcp

capture

主机

120.79.224.255

端口

9999

发送

高级

1	int64_t	1234567890	删除
2	string	aaaaaaaaaaaaaaaaaaaaaaaaaaaaa5555555555	删除
3	int64_t	0	删除
4	int64_t	0	删除
5	int64_t	0	删除
6	int64_t	0	删除

添加

图 5.16 构造要发送的数据

纯文本

int64\_t

41

Add

Struct

int64_t	string	int64_t	int64_t
1234567890	aaaaaaaaaaaaaaaaaaaaa aaaaaaaaa5555555555	0	0
int64_t	int64_t		
0	0		

图 5.17 收到测试服务端传回的消息

5365	464.831922	192.168.42.140	120.79.224.255	TCP	89 55280 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=66048 Len=35
5366	464.857111	120.79.224.255	192.168.42.140	TCP	89 9999 → 55280 [PSH, ACK] Seq=1 Ack=36 Win=131072 Len=35

> Frame 5365: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0

> Ethernet II, Src: b2:75:89:30:cc:88 (b2:75:89:30:cc:88), Dst: a6:04:1d:8f:ea:e4 (a6:04:1d:8f:ea:e4)

> Internet Protocol Version 4, Src: 192.168.42.140, Dst: 120.79.224.255

> Transmission Control Protocol, Src Port: 55280, Dst Port: 9999, Seq: 1, Ack: 1, Len: 35

▼ Data (35 bytes)

Data: 4523510088038000000000499602d26161670771b737373...

[Length: 35]

0000	a6 04 1d 8f ea e4 b2 75 89 30 cc 88 08 00 45 00	.....u .0....E.
0010	00 4b 2c fe 40 00 80 06 89 2b c0 a8 2a 8c 78 4f	.K,.,@... .+.*.x0
0020	e0 ff d7 f0 27 0f 5d 2d 5b 46 42 bc 86 ff 50 18	....'.]- [FB...P.
0030	01 02 21 1c 00 00 45 23 51 00 88 03 80 00 00 00	..l...E# 0.....
0040	00 49 96 02 d2 61 61 61 70 77 1b 73 73 46 44	.I...aaa pw.sssFD
0050	01 00 00 00 19 00 00 00 00	.....

图 5.18 wireshark 抓取的数据

数据原本的大小为8\*5+41=81 字节，通过 wireshark 抓包后发送在网络中数据大小仅为 35 字节大小，且接收的数据能够解析出原来的格式，可以说明数据压缩与解压的功能也是可以正常使用的；

5. 5. 3 数据抓取测试

选择现在联网用的网卡，并设置监听端口为 8080 端口，开始监听：

http		tcp		capture	
网络接口		\Device\NPF_{619CF879-355C-4DFE-8COD-E14339FB878D}		高级选项	
主机		端口	8080	最小长度限制	

IP+端口	长度	协议	简要信息(双击单条信息查看详情)
39 192.168.42.140:55645->120.79.224.255:8...	0		
40 192.168.42.140:55647->120.79.224.255:8...	0		
41 120.79.224.255:8080->192.168.42.140:55...	0		
42 192.168.42.140:55647->120.79.224.255:8...	0		
43 192.168.42.140:55647->120.79.224.255:8...	245		POST /httpServer/QtServlet HTTP/1.1..Content-Type: application
44 120.79.224.255:8080->192.168.42.140:55...	0		
45 192.168.42.140:55647->120.79.224.255:8...	11		a=1&b=2&c=3
46 120.79.224.255:8080->192.168.42.140:55...	819		HTTP/1.1 200 OK..Server: Apache-Coyote/1.1..Set-Cookie: JSESSI
47 192.168.42.140:55647->120.79.224.255:8...	0		
48 192.168.42.140:55645->120.79.224.255:8...	0		

图 5.19 数据监听界面

纯文本	POST /httpServer/QtServlet HTTP/1.1. Content-Type: application/x-www-form-urlencoded. Content-Length: 11. Connection: Keep-Alive. Accept-Encoding: gzip, deflate. Accept-Language: zh-CN,en,*. User-Agent: Mozilla/5.0. Host: 120.79.224.255:8080. .
Struct	

图 5.20 双击查看单条信息的详情



图 5.21 监听 9999 端口

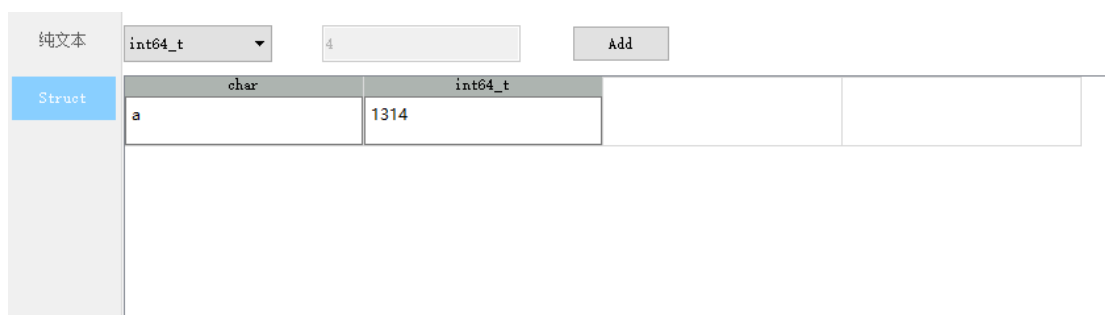


图 5.22 双击查看并解析数据

## 第七章 总结

本文深入分析了目前所存在的各种问题，运用 Qt、C++ 实现了网络应用层数据构造工具，用 WinPcap 实现了数据监听、筛选的功能，再加上 RC4 加密算法和 quicklz 压缩算法进一步完善了程序，增加了工具的适用性，最后采用多线程技术，增加了程序的可靠性，提高了程序的运行效率。本次主要完成以下工作：

- 1、通过查阅大量文献，参考了一些实际工程项目，完成了系统的选题以开发技术的选择；
- 2、运用自己之前所学的知识，参考一些实战项目，加入了网上的一些比较好的 DEMO，对照论文的需求分析，逐步完成了该软件；
- 3、在完成软件需求的基础上，采用多线程技术提高了程序的运行效率；在系统设计及实现的过程中，充分考虑了软件的可扩展性；
- 4、在系统的实现过程中，一步一步地对实现的功能进行测试，渐渐发现了一些程序 BUG，并一一解决；
- 5、在软件实现后，采用了具体的测试用例，对所有的系统模块、整个系统功能进行了测试，也对现阶段仍存在的问题进行了说明，完成了论文预期的工作；

## 参考文献

- [1] 霍亚飞. Qt Creator 快速入门. 北京航空航天大学出版社. 2017
- [2] Stanley B. Lippman Barbara E. Moo José José LaJoie. C++ Primer. 电

子工业出版社. 2013

[3]The WinPcap Team. WinPcap 中文技术文档.

<http://www.ferrisxu.com/WinPcap/html/index.html>. 2007

[4]IEEE Computer Society. TRANSMISSION CONTROL PROTOCOL.

<https://tools.ietf.org/html/rfc793>. 1981

[5] IEEE Computer Society. INTERNET PROTOCOL.

<https://tools.ietf.org/html/rfc791>. 1981

[6]linux 4.4.16 内核源码

[7]yafeilinux. Qt 快速入门系列教程.

<http://www.qter.org/portal.php?mod=view&aid=26>. 2012

[9]qt. QTableWidget Class | Qt Widgets 5.10. <http://doc.qt.io/qt-5/qtablewidget.html>. 2018