基于 QT的 UDP网络广播程序

摘　要

现在随着计算机网络技术的不断发展， 人类正进入信息化社会， 使用网络进程信息的传送以成为社会的一种基本的方式， 也是未来社会的发展方向。 在企业网中， 公司要求的速度和时间更为重要， 这就要求在信息传送中能够达到快速， 安全的目的。 同时能够保证信息能够发送给所有的公司员工，这就要求这种网络协议能够使用最少的资源，同时给所有人员发送信息。 UDP是一项非常实用可行的网络传输层协议，现在广泛应用于各行各业，并将在今后发挥更大的作用。本文详细阐述了 Qt 开发环境下广播的实现方法。

该系统采用用了基于 UDP协议的实现网络广播。 同时本论文采用了目前流行的图形界面设计 QT技术，在底层用基本的 C/ C++语言实现，在上层用 QT实现对界面的优化，及美化。

最终实现 UDP广播，程序运行起来可向多个用户发送 UDP报文，并能够在界面上显示报文，用户与发送报文的可以相互通信。

关键字 ：广播； QT; UDP

UDP broadcast network program based on QT

Abstract

Now the rapid development of computer network technology, man is entering the information society, the use of network information transmission process to become a basic mode of society, but also the future development direction of the society. In the enterprise network, companies require speed and time is more important, it requires to achieve fast in the information transmission, the purpose of safety. At the same time to ensure that information can be sent to all the employees of the company, which equires the network protocol can use the least resources, at the same time to all staff to send a message. UDP is a very practical network transport layer protocol is feasible, is now widely used in all walks of life, and will in the future play a greater role. This paper describes in detail the implementation method of the development environment of Qt broadcast.

The system adopted by the network broadcast basedon UDP protocol. At the same time, this paper uses a graphical interface design popular QT technologies, using basic C/ C++ language in the bottom, to realize the optimization of the interface with the QT on the upper layer, and landscaping.

The final realization of UDP broadcasting, programsrun up to multiple users send UDP message, and can display the message on the screen, can communicate with users and sending message.

Keywords: broadcast;QT;UDP

目　录

1. 绪论

1.1 研究背景

近来随着计算机的快速发展， 科学技术突飞猛进的发展， 知识经济的初见端倪， 目前基于 UDP协议的信息传送程序给我们的生活带来的很大的方便，现在，企业、机关、学校都建立起了自己的局域网， 在局域网里， 我们可以通过它， 实现在局域网里方便的联络，进行文件传输，消息的发布，自己共享内容的简介等。

在学校里， UDP广播可以起到方便同学之间，教师之间，师生之间的相互联络，这样，不用上 Internet ，可以节省资源，在学校这个大环境里，就可以方便学校与同学之间，教师与同学之间的联络， 学生可以通过它随时收听学校及导员发布的信息。 并能最大度地利用现有的网络资源， 极大地提高工作效率。 为了适应校园网的建设， 实现校园网内进行消息发布，学生交流，师生交流，网上作业等功能。

在公司企业建立起的局域网里， 应用于局域网内企业 （组织） 内部成员之间的交流领域，在没有因特网的支持下也可以进行即时通讯，亦因此使之更安全、更高效；同时因为它界面简洁实用，没有娱乐功能，所以简单易用，是企业（组织）成员更加专注于工作，减少不必要的财力和人力资源浪费。 ，对于企业来说提高企业的办事效率，提高企业的综合素质， 是一个企业不断进步迈向一个新台阶的重要步骤。 外了适应企业的快速发展，提高企业的管理水平，方便企业与内部员工的信息交流，节省办公开销，企业很需要这样一个程序。

为了在给定的主机上能识别多个目的地址， 同时允许多个应用程序在同一台主机上工作并能独立地进行数据报的发送和接收，设计用户数据报协议 UDP。

UDP是 Use Datagram Protocol 的简称，中文名用户数据报协议，是一种非连接式的通信协议，不需要建立有效的通讯连接，是 OSI（Open System Interconnection ，开放式系统互联） 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务 . UDP在 IP 报文的协议号是 17 ，UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点， 也就是说，当报文发送之后， 是无法得知其是否安全完整到达的广播和多播是基于 UDP协议的两种消息发送机制 . 广播数据即从一个工作站发出 ，局域网内的其他所有工作站都能收到它

1.2 系统的可行性分析

1.2.1 技术方面的可行性

本软件通过 QT技术基于 UDP协议实现广播（ Broadcast ），单播（ Unicast ），
（1）UDP协议

UDP是一个无连接协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端， UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

UDP是一个无连接协议，传输数据之前源端和终端建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。UDP信息包的标题很短，只有 8 个字节，相对于 TCP的 20 个字节信息包的额外开销很小。

(2)QT 技术

QT是 Trolltech 公司的标志性产品，是跨平台的 C++图形用户界面（ GUI) 工具包， QT 应用程序接口与工具兼容于所有支持平台，让开发员们掌握一个应用程序接口，便可执行与平台非相关的应用开发与配置。它对不同平台的专门 API 进行专门的封装（文件处理，网络等），QT API 对所有支持平台都是一致的，从而可以进行独立于平台的程序开发和配置。Qt 的良好封装机制使得 Qt 的模块化程度非常高，可重用性较好，对于用户开发来说是非常方便的。

(3)C++编程技术

C++是一种面向对象的通用型程序设计语言，，他是一种更好的 C，支持抽象的数据，支持通用性程序设计，具有更好的可移植性。

（4）TCP/IP 协议技术

在 TCP/IP 协议族中，有两个互不相同的传输协议： TCP（传输控制协议） 和 UDP（用户数据报协议）。TCP为两台主机提供高可靠性的数据通信。它所做的工作包括把应用程序交给它的数据分成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于运输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。而另一方面，UDP则为应用层提供一种非常简单的服务。它只是把称作数据报的分组从一台主机发送到另一台主机，但并不保证该数据报能到达另一端。任何必需的可靠性必须由应用层来提供。

1.2.2 经济方面的可行性

目前基于 UDP和 QT的技术的软件已经很多，但本软件强化了对单播、广播、组播的理解，能够根据他们的特性，在不同的应用场合选择使用不同的功能，并且在一些细节方面做出改进，改善图形化界面的可视效果，以及增删 button ，便于用户操作，在市场上仍有广阔的发展空间。

1.3 展望

UDP是 TCP/IP 协议族为传输层设计的两个协议之一，它在进程与进程的通信过程中，提供了有限的差错校验功能，是一种无连接的，不可靠的协议。UDP在一个较低的水平上完成进程之间的通信，在收到分组的时候没有流量控制机制也没有确认机制，适用于可靠性比较高的局域网。由于 UDP采取了无连接的方式，因此协议简单，在一些特定的应用中协议运行效率高。UDP适合一些实时的应用，如 IP 电话，视频会议，它们要求源主机以恒定的速率发送数据，并且在网络出现拥塞时，可以丢失一些数据，但是延迟不

能太大。基于这些特点，流式多媒体通信、多播等应用在传输层采用的就是 UDP协议。

广播系统具有实用性、经济性、便捷性等特点，广泛应用于各种公共场合。在楼宇智能消防系统中，它实现了报警控制中心向各分控点的语音广播。在灾时疏散人员、调配现场工作人员，它都发挥着不可替代的作用，消防广播是系统中的关键组成部分之一，已经成为其中不可缺少的组成部分。在网络会议中，服务器端负责进行用户管理、信息交互以及表决统计；客户端则实现收听发言，公开发言，私下讨论、投票表决等功能。在学校的考试系统中，老师通过服务端通过广播的形式负责试题的发送，学生在客服端接收试题，并且对试题的异常可以提出申请，老师收到申请后，以单播的形式处理异常，保证每个考生顺利完成考试。

因为 UDP具有 TCP所望尘莫及的速度优势。虽然 TCP协议中植入了各种安全保障功能，但是在实际执行的过程中会占用大量的系统开销，无疑使速度受到严重的影响。反观 UDP由于排除了信息可靠传递机制，将安全和排序等功能移交给上层应用来完成。目前基于 UDP协议的信息传送程序给我们的生活带来的很大的方便，对于企业来说提高企业的办事效率，提高企业的综合素质，是一个企业不断进步迈向一个新台阶的重要步骤。外了适应企业的快速发展，提高企业的管理水平，方便企业与内部员工的信息交流，节省办公开销，企业很需要这样一个程序。

而且，如果这样的程序还是跨平台的程序的话，那将会给我们的开发节省了大量的时间。QT 正是一个支持多平台的 C++图形用户界面应用程序框架。它提供给应用程序开发者建立图形用户界面所需的所用功能。Qt 是完全面向对象的很容易扩展，并且允许真正地组件编程。所以使用 QT 作为应用程序框架是一个不错的选择。

1.4 研究目标

现在科技越来越发达，手机上的软件，电脑上的软件越来越多，所以开发小的程序也越来越有必要。QT 具备跨平台，易扩展，性能稳定等优点。因此，以嵌入式处理器作为硬件平台，以 QT 作为程序框架，两者的完美结合必将成为未来技术的发展方向。

基于上述背景，开发一个 UDP 网络广播项目，使用 linux 搭配 qt 来构建这个项目。利用 QT4 作为系统应用层的程序框架。开发一个可以运行的网络广播程序。这个程序虽然是在 linux 下开发的，但是它是可移植的，在 windows 下也可以运行。

2. 相关技术理论

2.1 UDP 协议

UDP协议的全称是用户数据报协议，在网络中它与 TCP协议一样用于处理数据包，是一种无连接的协议。在 OSI 模型中，在第四层——传输层，处于 IP 协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。 UDP用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户 / 服务器模式彩已经被一些类似协议所掩盖，但是即使是在今天 UDP仍然不失为一项非常实用和可行的络传输层协议。

与所熟知的 TCP（传输控制协议）协议一样， UDP协议直接位于 IP（网际协议）协议的顶层。根据 OSI（开放系统互连）参考模型， UDP和 TCP都属于传输层协议。

UDP协议的主要作用是将网络数据流量压缩成数据包的形式。 一个典型的数据包就是一个二进制数据的传输单位。 每一个数据包的前 8 个字节用来包含报头信息， 剩余字节则用来包含具体的传输数据。

UDP是 OSI 参考模型中一种无连接的传输层协议， 它主要用于不要求分组顺序到达的传输中，分组传输顺序的检查与排序由应用层完成 [1] ，提供面向事务的简单不可靠信息传送服务。 UDP协议基本上是 IP 协议与上层协议的接口。 UDP协议适用端口分别运行在同一台设备上的多个应用程序。

2.2 UDP 报文

UDP 报头由 4 个域组成，其中每个域各占用 2 个字节，具体如图 2-1 UDP

每个 UDP 报文分 UDP 报头和 UDP 数据区两部分。报头由四个 16 位长（ 2 字节）字段组成，分别说明该报文的源端口、目的端口、报文长度以及校验值。

图 2-1 UDP 报文

分层封装：在 TCP/IP 协议层次模型中， UDP位于 IP 层之上。应用程序访问 UDP层然后使用 IP 层传送数据报。 IP 层的报头指明了源主机和目的主机地址，而 UDP层的报头指明了主机上的源端口和目的端口。

分解操作： UDP的复用、分解与端口

UDP软件应用程序之间的复用与分解都要通过端口机制来实现。 每个应用程序在发送数据报之前必须与操作系统协商以获得协议端口和相应的端口号。

UDP分解操作：从 IP 层接收了数据报之后，根据 UDP的目的端口号进行分解操作。

UDP协议使用端口号为不同的应用保留其各自的数据传输通道。 UDP和 TCP协议正是采用这一机制实现对同一时刻内多项应用同时发送和接收数据的支持。 数据发送一方

（可以是客户端或服务器端） 将 UDP数据包通过源端口发送出去， 而数据接收一方则通过目标端口接收数据。 有的网络应用只能使用预先为其预留或注册的静态端口； 而另外一些网络应用则可以使用未被注册的动态端口。因为 UDP报头使用两个字节存放端口号，所以端口号的有效范围是从 0 到 65535。一般来说，大于 49151 的端口号都代表动态端口。

数据报的长度是指包括报头和数据部分在内的总字节数。因为报头的长度是固定的，所以该域主要被用来计算可变长度的数据部分（又称为数据负载） 。数据报的最大长度根据操作环境的不同而各异。从理论上说，包含报头在内的数据报的最大长度为 65535 字节。不过，一些实际应用往往会限制数据报的大小，有时会降低到 8192 字节。

2.3 套结字

应用层通过传输层进行数据通信时， TCP和 UDP会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP连接或多个应用程序进程可能需要通过同一个 TCP协议端口传输数据。为了区别不同的应用程序进程和连接， 许多计算机操作系统为应用程序与 TCP/ IP 协议交互提供了称为套接字 (Socket) 的接口，区分不同应用程序进程间的网络通信和连接。网络化的应用程序在开始任何 通讯之前都必需要创建套接字。 就像电话的插口一样，没有它就完全没办法通信。生成套接字，主要有 3 个参数：通信的目的 IP 地址、使用的传输层协议 (TCP或 UDP)和使用的端口号。 Socket 原意是 "插座 "。通过将这 3 个参数结合起来，与一个 "插座 "Socket 绑定，应用层就可以和传输层通过套接字接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。Socket 可以看成在两个程序进行 通讯连接中的一个端点，一个程序将一段信息写入 Socket 中，该 Socket 将这段信息发送给另外一个 Socket 中，使这段信息能传送到其他程序中，如图 2-2 详细描述了套接字。

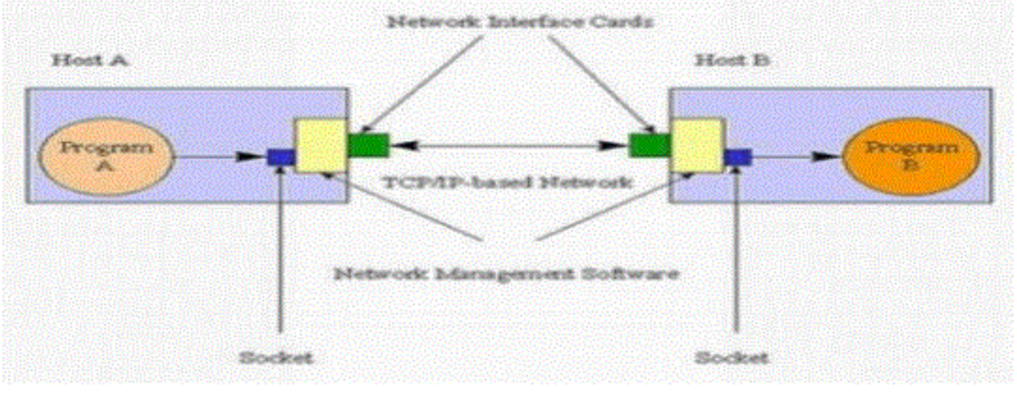


图 2-2 关于套接字

上图，Host A 上的程序 A 将一段信息写入 Socket 中，Socket 的内容被 Host A 的网络管理软件访问，并将这段信息通过 Host A 的网络接口卡发送到 Host B，Host B 的网络接口卡接收到这段信息后，传送给 Host B 的网络管理软件，网络管理软件将这段信息保存在 Host B 的 Socket 中，然后程序 B 才能在 Socket 中阅读这段信息。

假设在图中的网络中添加第三个主机 Host C，那么 Host A 怎么知道信息被正确传

送到 Host B 而不是被传送到 Host C 中了呢？基于 TCP/IP 网络中的每一个主机均被赋予了一个唯一的 IP 地址，IP 地址是一个 32 位的无符号整数，由于没有转变成二进制，因此通常以小数点分隔，如： 198.163.227.6。正如所见 IP 地址均由四个部分组成，每个部分的范围都是 0-255，以表示 8 位地址。

值得注意的是 IP 地址都是 32 位地址，这是 IP 协议版本 4（简称 Ipv4）规定的，目前由于 IPv4 地址已近耗尽，所以 IPv6 地址正逐渐代替 Ipv4 地址，Ipv6 地址则是 128 位无符号整数。

假设第二个程序被加入图中的网络的 Host B 中，那么由 Host A 传来的信息如何能被正确的传给程序 B 而不是传给新加入的程序呢？这是因为每一个基于 TCP/IP 网络通讯的程序都被赋予了唯一的端口和端口号，端口是一个信息缓冲区，用于保留 Socket 中的输入 /输出信息，端口号是一个 16 位无符号整数，范围是 0-65535，以区别主机上的每一个程序（端口号就像房屋中的房间号），低于 256 的短口号保留给标准应用程序，比如 pop3 的端口号就是 110，每一个套接字都组合进了 IP 地址、端口、端口号，这样形成的整体就可以区别每一个套接字。要通过互联网进行通信，至少需要一对套接字，一个运行于客户机端，称之为 ClientSocket，另一个运行于服务器端，称之为 serverSocket。根据连接启动的方式以及本地套接字要连接的目标，套接字之间的连接过程可以分为三个步骤：服务器监听，客户端请求，连接确认。

服务器监听：是服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态。

客户端请求：是指由客户端的套接字提出连接请求，要连接的目标是服务器端的套接字为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套字的地址和端口号，然后就向服务器端套接字提出连接请求。

连接确认：是指当服务器端套接字监听到或者说接收到客户端套接字的连接请求，它就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，连接就建立好了。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

套接字起源于 20 世纪 70 年代加州大学伯克利分校版本的 Unix，即人们所说的 BSD Unix。因此，有时人们也把套接字称为“伯克利套接字”或“BSD套接字”。一开始，套接字被设计用在同一台主机上多个应用程序之间的通讯。这也被称作进程间通讯，或 IPC。套接字有两种，分别是基于文件型的和基于网络型的。

Unix 套接字是我们要介绍的第一个套接字家族。其“家族名”为 AF_UNIX（在 POSIX1.g 标准中也叫 AF_LOCAL），表示“地址家族：UNIX”。包括 Python 在内的大多数流行平台上都使用术语“地址家族”及其缩写“AF”。而老一点的系统中，地址家族被称为“域”或“协议家族”，并使用缩写“PF”而不是“AF”。同样的，AF_LOCAL（在 2000-2001 年被列为标准）将会代替 AF_UNIX。不过，为了向后兼容，很多系统上，两者是等价的。Python 自己则仍然使用 AF_UNIX。

由于两个进程都运行在同一台机器上，而且这些套接字是基于文件的。所以，它们的底层结构是由文件系统来支持的。这样做相当有道理，因为，同一台电脑上，文件系统的确是不同的进程都能访问的。

另一种套接字是基于网络的，它有自己的家族名字：AF_INET，或叫“地址家族：Internet。”还有一种地址家族 AF_INET6 被用于网际协议第 6 版（IPv6）寻址上。还有

一些其他的地址家族，不过，它们要么是只用在某个平台上，要么就是已经被废弃，或是很少被使用，或是根本就还没有实现。所有地址家族中， AF_INET 是使用最广泛的一个。Python 2.5 中加入了一种 Linux 套接字的支持：AF_NETLINK（无连接 (稍后讲解)）套接字家族让用户代码与内核代码之间的 IPC 可以使用标准 BSD 套接字接口。而且，相对之前那些往操作系统中加入新的系统调用、 proc 文件系统支持或是 “IOCTL”等复杂的方案来说，这种方法显得更为精巧，更为安全 Python 只支持 AF_UNIX ，AF_NETLINK ，和 AF_INET 家族。由于我们只关心网络编程，所以在本章的大部分时候，我们都只用 AF_INET 。

2.4 套接字地址：主机与端口

如果把套接字比做电话的插口 —即通信的最底层结构，那主机与端口就像区号与电话号码 的一对组合。有了能打电话的硬件还不够， 你还要知道你要打给谁， 往哪打。一个因特网地址由网络通信所必需的主机与端口组成。 而且不用说，另一端一定要有人在听才可以。否则，你就会听到熟悉的声音 “对不起，您拨的是空号，请查询后再拨 ”。你在上网的时候，可能也见过类似的情况，如 “不能连接该服务器。服务器无响应或不可达。

合法的端口号范围为 0～65535。其中，小于 1024 的端口号为系统保留端口。如果你使用的是 Unix 操作系统，那么就可以通过 /etc/services文件获得保留的端口号（及其对应的服务 /协议和套接字类型）。

2.5 端口号

为了区分一台主机接受到的数据包应该交给哪个进程进行处理，使用端口号。 UDP 协议使用端口号为不同的应用保留其各自的数据传输通道。 UDP协议正是采用这一机制实现对同一时刻内多项应用同时发送和接收数据的支持。数据发送一方 (可以是客户端或服务器端)将 UDP数据报通过源端口发送出去，而数据接收一方则通过目标端口接收数据。有的网络应用只能使用预先为其预留或注册的静态端口； 而另外一些网络应用则可以使用未被注册的动态端口。因为 UDP 报头使用两个字节存放端口号，所以端口号的有效范围是从 0 到 65535。一般来说，大于 49151 的端口号都代表动态端口。数据报的长度是指包括报头和数据部分在内的总的字节数。 因为报头的长度是固定的， 所以该域主要被用来计算可变长度的数据部分 (又称为数据负载)。数据报的最大长度根据操作环境的不同而各异。 从理论上说，包含报头在内的数据报的最大长度为 65535 字节。不过，一些实际应用往往会限制数据报的大小，有时会降低到 8192 字节。UDP 协议使用报头中的校验值来保证数据的安全。校验值首先在数据发送方通过特殊的算法计算得出，在传递到接收方之后， 还需要再重新计算。 如果某个数据报在传输过程中被第三方篡改或者由于线路噪音等原因受到损坏， 发送和接收方的校验计算值将不会相符， 由此 UDP 协议可以检测是否出错

2.6 报头的校验值

UDP 协议使用报头中的校验值来保证数据的安全。校验值首先在数据发送方通过特殊的算法计算得出，在传递到接收方之后，还需要再重新计算。如果某个数据报在传输过程中被第三方篡改或者由于线路噪音等原因受到损坏，发送和接收方的校验计算值将不会相符，由此 UDP 协议可以检测是否出错。这与 TCP 协议是不同的，后者要求必须具有校验值。

许多链路层协议都提供错误检查，包括流行的以太网协议，也许你想知道为什么 UDP 也要提供检查和校验。其原因是链路层以下的协议在源端和终端之间的某些通道可能不提供错误检测。虽然 UDP 提供有错误检测，但检测到错误时，UDP 不做错误校正，只是简单地把损坏的消息段扔掉，或者给应用程序提供警告信息。

UDP Helper 是实现对指定 UDP 端口广播报文的中继转发，即将指定 UDP 端口的广播报文转换为单播报文发送给指定的服务器，起到中继的作用 .

2.7 信号和槽机制

在对于事件处理的问题，较早的工具包使用“回调”的方式来实现。回调是指一个函数的指针，如果你希望一个处理函数同志你一些事件，你可以把另一个函数的指针传递给处理函数。处理函数在适当的时候会调用回调函数。

采用回调方式实现对象间的通讯有两个主要缺点，首先回调函数不是类型安全的，我们不能确定处理函数使用了正确的参数来调用回调函数，第二，回调函数和处理函数间的联系非常紧密，因为处理函数必须知道要调用哪个回调函数。 /

信号和槽机制就是：当一个特定的事件发生时，一个或几个被指定的信号就被发射，槽就是一个返回值为 void 的函数，如果存在一个或几个槽和该信号相连接，那在该信号被发射后，这个（些）槽（函数）就会立刻被执行。

QT 的窗口部件已经有很多预定义的信号，也有很多预定义的槽，但我们总是通过继承来加入我们自己的信号和自己的槽，这样我们就可以处理感兴趣的信号了。凡是从 QObject 类或者它的某个子类继承的所有类都可以包含信号和槽。当某个事件发生后，被指定的信号就会被发射，它不知道也没有必要知道是否有槽连接了该信号，这就是信息封装。

槽是可以用来接收信号的正常的对象的成员函数，一个槽不知道它是否被其它信号连接。可以把一个信号和一个槽进行单独连接，这时槽会因为该信号被发射而被执行；也可以把几个信号连接在同一个槽上，这样任何一个信号被发射都会使得该槽被执行；也可以把一个信号和多个槽连接在一起，这样该信号一旦被发射，与之相连接的槽都会被马上执行，但执行的顺序不确定，也不可以指定；也可以把一个信号和另一个信号进行连接，这样，只要第一个信号被发射，第二个信号立刻就被发射。

2.8 绑定

由管理机构指定端口和动态绑定的方式， UDP/TCP 中采用端口（ port）来标识 传输端口代表 TCP/UDP 的传输服务访问点，在进程通信中标识相互通信的进程通信的对端进程地址可表示为： (IP address , port) 传输端口的绑定（ binding）进程在某个传输端口进行数据传输前，必须首先通过系统调用与该端口建立绑定关系 UDP/TCP 的传输端口号（ port number） 端口号用于标识 UDP/TCP 的传输端口 UDP/TCP 协议各分别可以提供最多 64K 个传输端口进程通信时，必须了解对端进程的地址 (IP + port)动态绑定方式（本地分配） TCP/IP 系统种端口分配方法应用进程通信采用“客户-服务器”（client-server）模式将传输端口划分为两类：保留端口和自由端口。保留端口(well-known port)：为服务进程全局分配的端口自由端口是在进程需要进行通信时，由本地进行动态分配的客户进程首先动态申请一个本地自由端口号， 再通过服务进程所公布的保留端口与服务器进程建立联系， 并进行相应协商； 上述过程成功后中， 就可开始进程间的通信。

3. 系统总体的描述

3.1 系统基本简介和概要

UDP协议，即拥护数据报协议 (Use Datagram Protocol) ，是一个简单的面向数据报的传输层协议。他不提供可靠性，即只把应用程序传给 IP 层的数据发送出去，但是并不能保证他们能到达目的。 广播和组播是基于 UDP协议的两种消息发送机制， 广播数据即从一个工作站发出，局域网内的其他所有工作站都能收到它。 IP 协议下，组播是广播的一种变形， IP 组播要求将对收发数据感兴趣的所有主机加入到一个特定的组。

Qt 是诺基亚开发的一个跨平台的 C++图形用户界面应用程序框架。 它提供给应用程序开发者建立艺术级的图形用户界面所需的所用功能。 Qt 是完全面向对象的，很容易扩展，并且允许真正地组件编程。

通过 QT技术和 UDP实现整个软件的功能包括：广播，单播

3.2 系统能够完成的功能概要

对于广播报文，能够实现一对一的消息发送机制，只有特定的一个 IP 才能够接收到信息对于广播，从一个工作站发出，局域网内的其他所有工作站都能收到它。

3.3 软件的特点

3.3.1 单播的特点

在客户端与服务器端建立一个单独的数据通道，从一台服务器送出的每个数据包只能传送到特定的客户端， 但是由于其能够针对每个客户的及时响应， 所以现在的网页浏览全部采用的是单播协议。

他的优点在于服务器及时响应客户的请求， 而且能够针对每个客户不同的请求发送不同的数据，容易实现个性化服务。不足之处在于，如果 10 个客户机需要相同的数据时，服务器要逐一传送，重复 10 次相同的工作，增加了服务器的负载。

3.3.2 广播的特点

广播是多点传递的最普遍的形式 ,它向每一个目的节点传送一个分组的拷贝 。网络设备简单，维护简单，服务器流量负载很低，主机之间“一对所有”的通讯模式，网络对每一台主机发出的信号都就行无条件的复制并转发， 所有的主机都可以接收到所有信息（不管你是否需要），由于其不用路径选择，所以其网络成本很低廉。

需要强调的是：广播无法针对每个客户的要求和时间及时提供个性化服务，对此信息不感兴趣的主机也必须收到，相反，有可能由于网络拥塞，想得到的却得不到，会耗费大量的主机资源和网络资源，而且不能穿透子网，因为路由器会封锁广播通信，广播传输会增加非接收者的开销。

3.3.3 系统创新点

现今社会处于读图时代，人们习惯于对事物进行视觉上的接受，如何适应在这种环境下设计出来便于人们接受的产品，成为人们研发的重要方向，而用QT设计出来的网络广播程序，不仅克服了传统的广播中声音稍纵即逝的局限性。也改善图形化界面的可视效果，便于用户操作。使网络广播在市场上实现效益的最大化。具体来说，用户可根据不同的选项选择消息发送的样式，自主选择在本机网络发送还是在局域网上发送，以模块化方式组织程序各个模块之间的联系，在需要时可手动加入新的模块。并且强化了对单播、广播的理解，能够根据他们的特性，在不同的应用场合选择使用不同的功能。

通常 UDP 与服务器是分开的，我实现的程序将客户端于服务端封装在一起，节约了资源，提高了软件的重用。

4. 系统分析与总体设计

4.1 系统需求分析

广播系统具有实用性、经济性、便捷性等特点，广泛应用于各种公共场合，如智能楼宇消防系统，网络教学。而 UDP 能够排除信息可靠传递机制，将安全和排序等功能移交给上层应用来完成。 能够根据不同的应用场合以最快， 最完整，最节省资源的方式将信息传递给用户，使用户的利益得到了最大的保障。所以说用 UDP 协议来传递信息给我们的生活带来的很大的方便， 对于企业来说提高企业的办事效率， 适应企业的快速发展，提高企业的管理水平，方便企业与内部员工的信息交流，节省办公开销，企业很需要这样一个程序。

4.2 系统开发及运行环境

硬件平台：

惠普 6450B

软件平台：

操作系统： LINUX

开发工具包： QT 4.6

分辨率：最佳效果 1024×768 像素。

4.3 系统主要功能要求

系统设计目标如下：主要功能包括： TCP 协议中包含了专门的传递保证机制，当数据接收方收到发送方传来的信息时， 会自动向发送方发出确认消息； 发送方只有在接收到该确认消息之后才继续传送其它信息， 否则将一直等待直到收到确认信息为止。 与 TCP 不同， UDP 协议并不提供数据传送的保证机制。如果在从发送方到接收方的传递过程中出现数据包的丢失，协议本身并不能作出任何检测或提示。因此，通常人们把 UDP 协议称为不可靠的传输协议。 相对于 TCP 协议，UDP 协议的另外一个不同之处在于如何接收突发性的多个数据包。不同于 TCP，UDP 并不能确保数据的发送和接收顺序。

鉴于以上程序要求，我采用的是基于 UDP 协议的 Socket编程方式。对应套接字的类型选择面向连接的流式套接字 （SOCK_STREAM ）,QUdpSocket 类提供一个 udp 套接字,QUdpscoket 是 QabstractSocket类非常方便的一个子类 QUDPServer 类用来与远端服务器连线，如果开发人员要接受客户端连线，则使用 QUDPServer。QUDPServer 使用 listen()方法开始倾开所指定的连接埠，开发人员可以使用 Listening()方法测试是否正在倾听连线。

4.4 系统总体设计

模块化是指解决一个复杂问题时自顶向下逐层把系统划分成若干模块的过程，有多种属性，分别反映其内部特性。模块化是一种处理复杂系统分解为更好的可管理模块的方式。

模块化用来分割，组织和打包软件。每个模块完成一个特定的子功能，所有的模块按某种方法组装起来，成为一个整体，完成整个系统所要求的功能。

模块具有以下几种基本属性：接口、功能、逻辑、状态，功能、状态与接口反映模块的外部特性，逻辑反映它的内部特性。

在系统的结构中，模块是可组合、分解和更换的单元。模块化是一种处理复杂系统分解成为更好的可管理模块的方式。它可以通过在不同组件设定不同的功能，把一个问题分解成多个小的独立、互相作用的组件，来处理复杂、大型的软件。

特点:各个模块可独立工作 ,即便单组模块出现故障也不影响整个系统工作，界面设计：用 UI 设计器进行界面设计

界面分成三大块，显示消息窗体界面，发送广播和接受广播的发送者和接收者用户列表窗体界面，以及发送消息窗体界面。

逻辑设计：对界面上的窗口和需要响应的各种事件编写对应的信号和槽函数，和相应的程序逻辑，

(1)总体功能模块如图 4-1 所示。

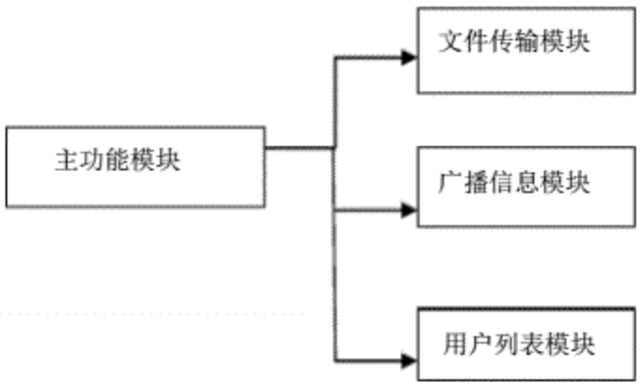


图 4-1 系统主功能模块图

(2)UDP　服务器端模块如图　4-2 所示：

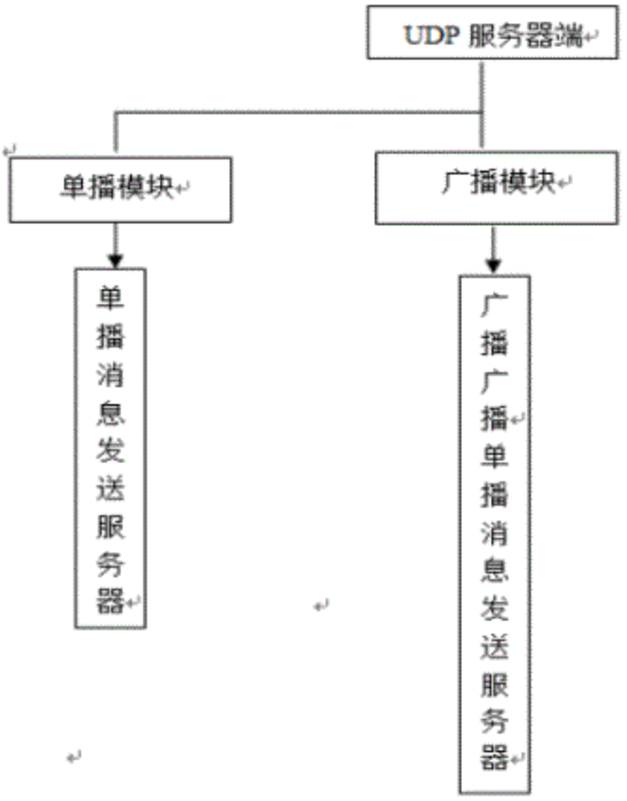


图 4-2 UDP 服务器模块图

UDP 服务器端模块包括单播消息发送模块、广播消息发送模块和组播消息发送模块，他们都能对需要广播的的消息进行编辑，编辑完以后，按一下发送按键，就可以使消息发送出去，然后可以编辑下一条消息，在需要发送时，再次重复上面的动作。

(3)UDP 客户端模块如图　4-3 所示

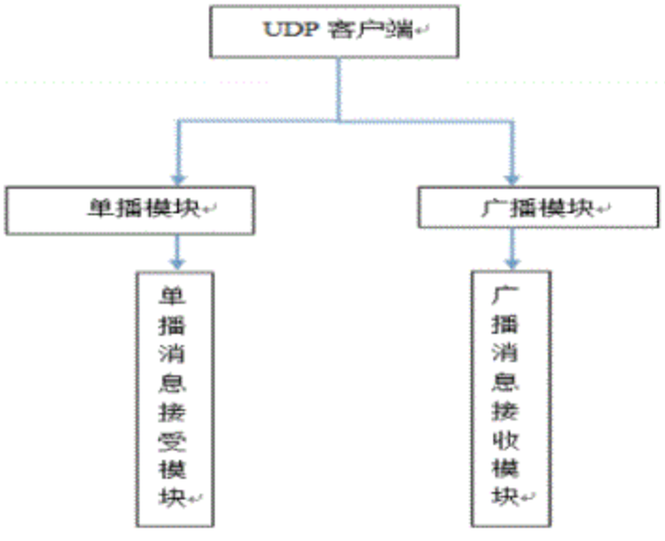


图 4-3 UDP 客户端模块

UDP 客服端模块包括单播消息接收模块、广播消息接收模块和，他们都能对需要广播的的消息进行接收。当需要接收消息时，只要连上服务器的都能接受到消息。

一般情况下客户端都是保持畅通的，这样就能够在第一时间收到服务器发来的消息，例如，智能小区的消防系统，当总控制室收到火灾报警时，立即向小区的个个广播点发送广播，也正是因为广播点的网络保持畅通，才使得人们能够及时获得火灾消息，做出相应预防措施，极大的控制了灾情，使灾害最小化。

4.5 各个模块的设计和功能

4.5.1 单播模块

(1)服务端模块如图 4-4 所示：

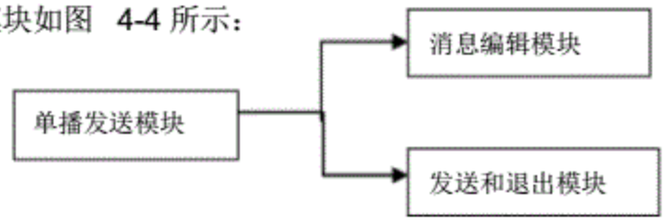


图 4-4 单播发送模块

此模块包括消息的编辑模块、发送与退出模块。消息的编辑模块可以随时在“message”下编辑要发送的消息。发送与退出模块可以选择进行消息的发送与停止发送。按一下 START 按键，就可以使消息每隔一千毫秒钟发送一次，当需要停止消息发送的时候， 按一下 STOP 按键即可停止。值得一提的是，单播消息发送出去后只有指定的 IP 和端口才能接收到服务器发来的消息。

(2) 客户端模块如图 4-5 所示：

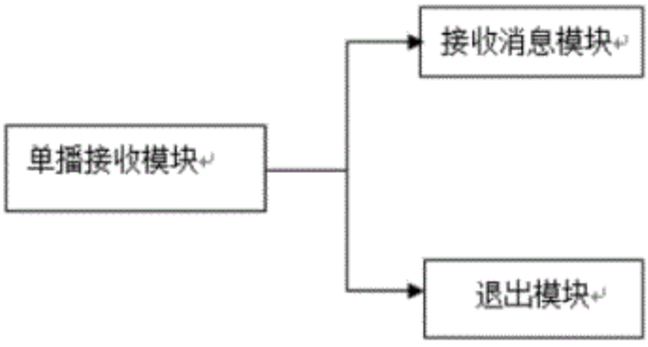


图 4-5 客户端接收模块

此模块包括接收消息模块 ,连接上服务端的客户端模块都能自动接受到服务器端发过来的广播

4.5.2 广播模块

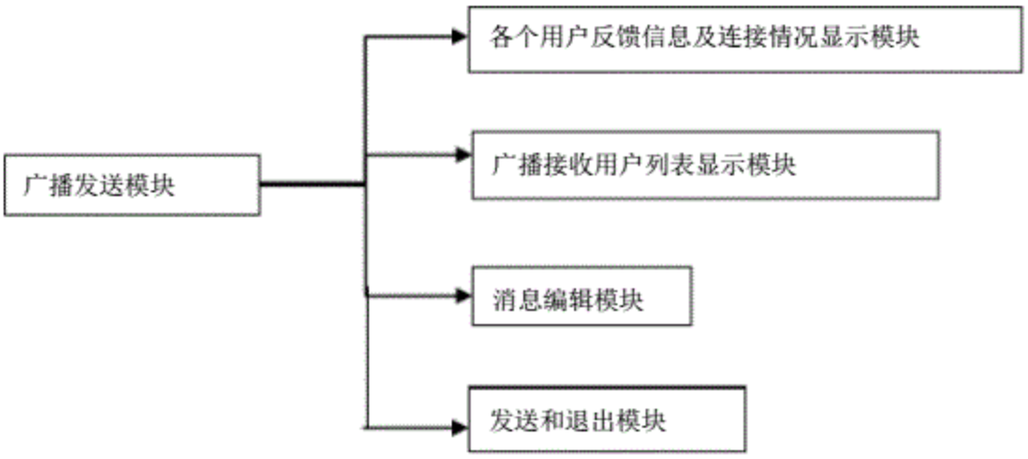


图 4-6 广播发送模块

此模块包括消息的编辑模块、发送与退出模块（如图 4-6 所示）。功能和单播消息的发送机制类似， 只不过广播消息发送出去后， 位于同一局域网内的用户都可以收到服务器发来的消息。

加入的客户端数量可由手动控制， 每运行一次加入一个新的客户端。 广播接收模块有多个客户端接收模块组成，每个客户端接收模块的功能和单播消息接收模块的功能相同，这里不再赘述。当服务器发来消息时， 位于同一子网内的用户都能接收到服务器的消息。该功能在网络教学，消防系统，网络会议等场合得到了广泛的应用。

4.6 系统的流程

(1)服务器端的系统流程图如图 4-7 所示，

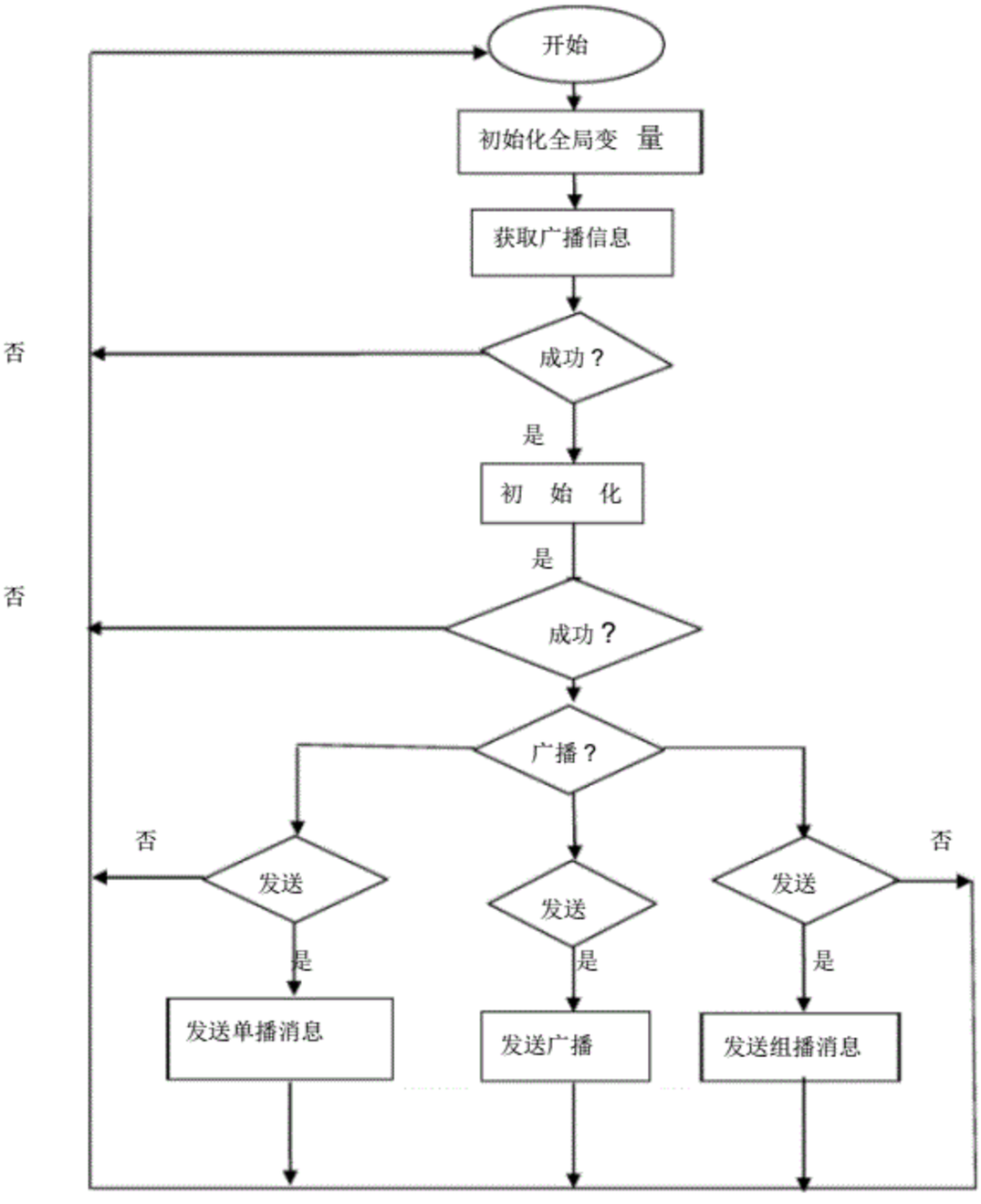


图 4-7 消息的发送流程图

(2) 客户端流程图，客户端流程要经过下列流程，从创建 UDP套结字开始

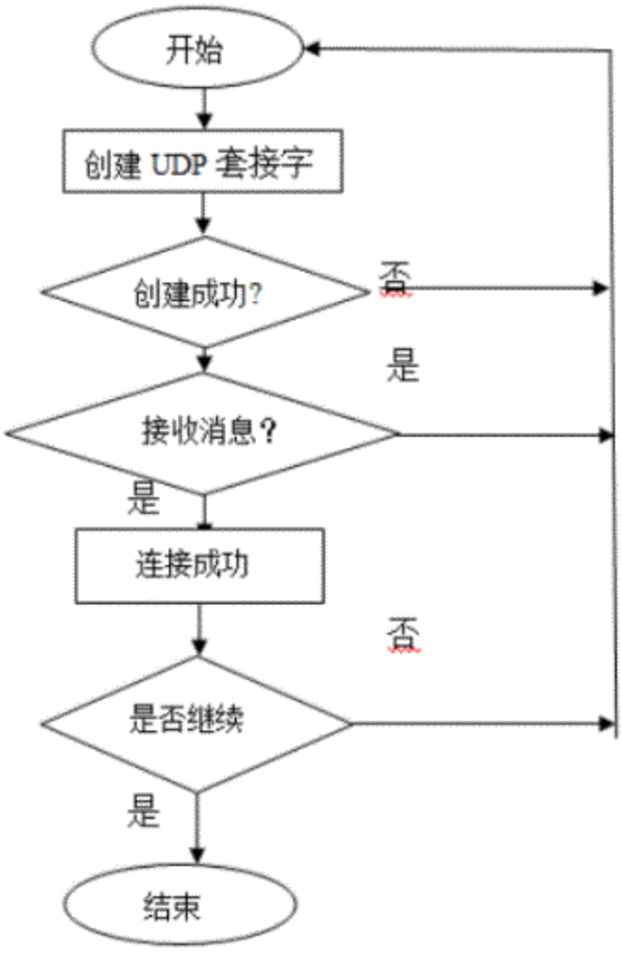
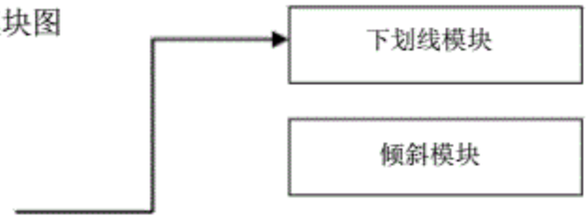


图 4-8 客户端流程

对广播文本的编辑可以实现如下几功部分功能， 对广播文本添加下划线， 对广播文行倾斜，对广播文本设置字号，对广播文本设置字体类型，对广播文本加粗，将要发送的广播文本的颜色改变，如下图所示：

（3）对广播文本编辑模块图



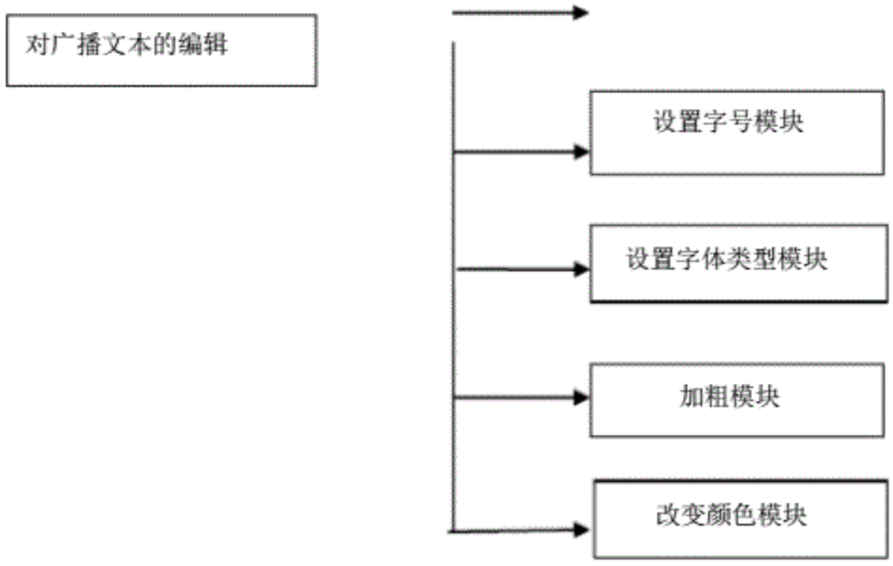


图 4-9 文本编辑模块图

对广播信息的处理包括两部分，保存广播信息模块和清空广播信息模块。

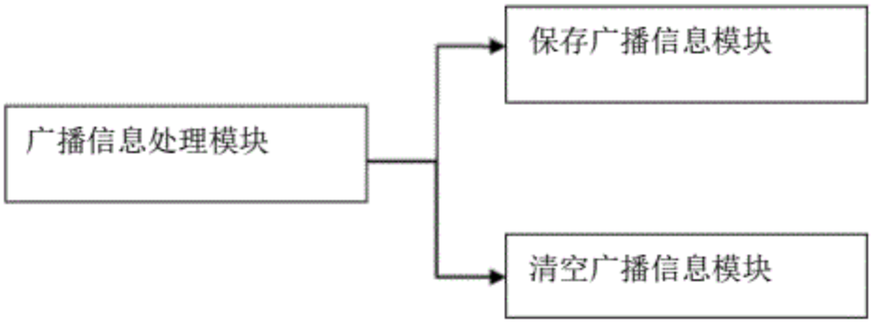


图 4-10 信息处理模块图

5. 网络广播程序的详细设计

界面详细设计 ,首先对广播模块的界面进行详细设计，界面分成三部分，用户及广播消息显示模块，列表模块，发送消息模块，工具栏模块。

5.1 界面设计

程序界面设计本程序采用 QT 自带的 UI 设计器对界面进行了布局和设计，在代码中具体是通过继承的方式，利用 setup:UI 方法将设计器上所画的窗口用控件实例化出来。主界面效果图如图 所示

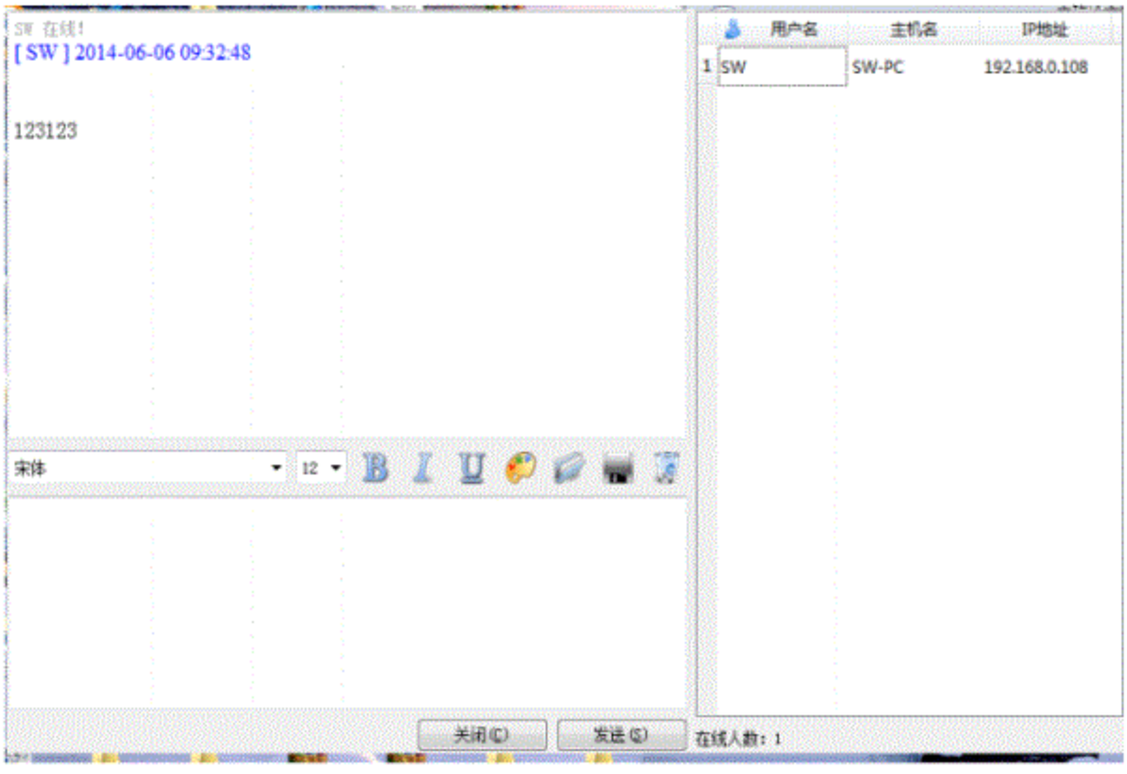


图 5-1 网络广播界面

界面设计及格式设计如下

```
this->setWindowFlags(Qt::FramelessWindowHint);
ui->setupUi(this);
this->resize(850,550);
ui->textEdit->setFocusPolicy(Qt::StrongFocus);
ui->textBrowser->setFocusPolicy(Qt::NoFocus);
ui->textEdit->setFocus();
ui->textEdit->installEventFilter(this)
```

5.2 模块功能设计

（1）广播模块主功能设计： UDP 搭建广播模块步骤： 按照搭建网络服务器的流程进行，分为以下几个步骤进行。

1、创建套接字—— socket()

2、指定本地地址—— bind()

3、建立套接字连接—— connect()和 accept()

4、监听连接—— listen()
5、数据传输—— send()与 recv()
6、关闭套接字—— closesocket()
关键代码如下，

```
udpSocket = new QUdpSocket(this);
    port = 45454;
    udpSocket->bind(port,QUdpSocket::ShareAddress
                                        | QUdpSocket::ReuseAddressHint);
    connect(udpSocket,SIGNAL(readyRead()),this,SLOT(processPendingDatagrams()));
    sendMessage(NewParticipant);
```

(2)信号和槽的连接，在这段代码中绑定了端口，并建立了 UDP 套接字，在 QT 中是将很多流程封装到 QT 自带的 socket 类中，槽函数的连接 ,将广播端信号个服务器端的槽连接在一起

```
    connect(lineEdit, SIGNAL(returnPressed()), this, SLOT(returnPressed()));
    connect(&client, SIGNAL(newMessage(QString,QString)),
            this, SLOT(appendMessage(QString,QString)));
    connect(&client, SIGNAL(newParticipant(QString)),
            this, SLOT(newParticipant(QString)));
    connect(&client, SIGNAL(participantLeft(QString)),
            this, SLOT(participantLeft(QString)));
    myNickName = client.nickName();
    newParticipant(myNickName);
    tableFormat.setBorder(0);
    QTimer::singleShot(10 * 1000, this, SLOT(showInformation()));
}
```

Chardilog.h

```
include "client.h"
class ChatDialog : public QDialog, private Ui::ChatDialog
{
    Q_OBJECT
public:
    ChatDialog(QWidget *parent = 0);
public slots:
    void appendMessage(const QString &from, const QString &message);
private slots:
    void returnPressed();
    void newParticipant(const QString &nick);
    void participantLeft(const QString &nick);
    void showInformation();
```

```
private:
    Client client;
    QString myNickName;
    QTextTableFormat tableFormat;
};
#endif
```

这是会话模块的头文件，其中有对槽函数和私有成员的声明

（3）设置环境变量 :在程序中具体用代码实现了对环境变量的设置 ,关键代码如下所示 ，

```
QStringList environment = QProcess::systemEnvironment();
    foreach (QString string, envVariables)
    {
        int index = environment.indexOf(QRegExp(string));
        if (index != -1)
        {
            QStringList stringList = environment.at(index).split('=');
            if (stringList.size() == 2)
            {
                return stringList.at(1);
                  break;
              }
```

（4）界面左侧列表显示消息详细设计，在代码中如下所示，

```
void ChatDialog::appendMessage(const QString &from, const QString &message)
{
    if (from.isEmpty() || message.isEmpty())
        return;
    QTextCursor cursor(textEdit->textCursor());
    cursor.movePosition(QTextCursor::End);
    QTextTable *table = cursor.insertTable(1, 2, tableFormat);
    table->cellAt(0, 0).firstCursorPosition().insertText('<' + from + "> ");
    table->cellAt(0, 1).firstCursorPosition().insertText(message);
    QScrollBar *bar = textEdit->verticalScrollBar();
    bar->setValue(bar->maximum());
}
void ChatDialog::returnPressed()
{
    QString text = lineEdit->text();
    if (text.isEmpty())
        return;
    if (text.startsWith(QChar('/'))) {
        QColor color = textEdit->textColor();
        textEdit->setTextColor(Qt::red);
        textEdit->append(tr("! Unknown command: %1")
```

```
                                        .arg(text.left(text.indexOf(' '))));
            textEdit->setTextColor(color);
        } else {
            client.sendMessage(text);
            appendMessage(myNickName, text);
        }
        lineEdit->clear();
}
void ChatDialog::newParticipant(const QString &nick)
{
        if (nick.isEmpty())
            return;
        QColor color = textEdit->textColor();
        textEdit->setTextColor(Qt::gray);
        textEdit->append(tr("* %1 has joined").arg(nick));
        textEdit->setTextColor(color);
        listWidget->addItem(nick);
}
void ChatDialog::participantLeft(const QString &nick)
{
        if (nick.isEmpty())
            return;
        QList<QListWidgetItem *> items = listWidget->findItems(nick,
                                                               Qt::MatchExactly);
        if (items.isEmpty())
            return;
        delete items.at(0);
        QColor color = textEdit->textColor();
        textEdit->setTextColor(Qt::gray);
        textEdit->append(tr("* %1 has left").arg(nick));
        textEdit->setTextColor(color);
}
```

设置延时弹出消息模块，如果打开主界面为单一界面且

```
void ChatDialog::showInformation()
{
        if (listWidget->count() == 1) {
            QMessageBox::information(this, tr("  宋玮 UDP  网络广播 "),
                                     tr("Launch several instances of this "
                                        "program on your local network and "
                                        "start broadcasting!"));
```

}

（5）连接模快设计，在 .cpp 文件中具体实现了槽函数，在其中实例化了方法 连接模块的详细设计 CONNECTION_H CONNECTION_CPP 连接模块继承了 QTCPsocket 自己重新定义了连接协议，其中第一用心跳程序 pingTimer .start() 和 pongTime.start() ，来随时用来保持可随时跟踪着服务器和客户端的连接状态， 心跳程序每隔固定的时间间隔来检测服务器和各个客户端是否还保持连接 ，在静态变量中设置了心跳程序的时间间隔 。

该协议中用到了两个枚举类型， 分别定义了连接状态和连接数据类型 ，连接状态有三种 ，等待问候状态，读取问候信息，准备就绪；数据类型有 5 种分别是，文本类型，心跳程序的数据类型，问候信息以及未定义的数据类型。

连接中涉及的函数有 1 读取数据缓冲区， 2 当前数据长度类型 3 读取数据报头 4 拥有足够数据 5 数据进程函数。第四 static const int MaxBufferSize = 1024000socket 套接字实为一个管道，由于怕 socket 管道破裂，要为设置的数据报设置一个上限，该静态常量的解决方法很好的解决了这个问题。服务端详细设计代码如 Peermanager.h 和 Peermanager.cpp所示：

Peermanager.cpp.

```
class PeerManager : public QObject
{
    Q_OBJECT
public:
    PeerManager(Client *client);
    void setServerPort(int port);
    QByteArray userName() const;
    void startBroadcasting();
    bool isLocalHostAddress(const QHostAddress &address);
signals:
    void newConnection(Connection *connection);
private slots:
    void sendBroadcastDatagram();
    void readBroadcastDatagram();
private:
    void updateAddresses();
    Client *client;
    QList<QHostAddress> broadcastAddresses;
    QList<QHostAddress> ipAddresses;
    QUdpSocket broadcastSocket;
    QTimer broadcastTimer;
    QByteArray username;
    int serverPort;
};
#endif
```

Peermanager.cpp

```
#include <QtNetwork>
#include "client.h"
#include "connection.h"
#include "peermanager.h"
static const qint32 BroadcastInterval = 2000;
static const unsigned broadcastPort = 45000;
    broadcastSocket.bind(QHostAddress::Any, broadcastPort, QUdpSocket::ShareAddress
                                    | QUdpSocket::ReuseAddressHint);
    connect(&broadcastSocket, SIGNAL(readyRead()),
            this, SLOT(readBroadcastDatagram()));
    broadcastTimer.setInterval(BroadcastInterval);
    connect(&broadcastTimer, SIGNAL(timeout()),
            this, SLOT(sendBroadcastDatagram()));
}
void PeerManager::setServerPort(int port)
{
    serverPort = port;
}
QByteArray PeerManager::userName() const
{
    return username;
}
void PeerManager::startBroadcasting()
{
    broadcastTimer.start();
}
bool PeerManager::isLocalHostAddress(const QHostAddress &address)
{
    foreach (QHostAddress localAddress, ipAddresses) {
        if (address == localAddress)
            return true;
    }
    return false;
}
void PeerManager::sendBroadcastDatagram()
{
    QByteArray datagram(username);
    datagram.append('@');
    datagram.append(QByteArray::number(serverPort));
    bool validBroadcastAddresses = true;
    foreach (QHostAddress address, broadcastAddresses) {
        if (broadcastSocket.writeDatagram(datagram, address,
```

```
                                                    broadcastPort) == -1)
                validBroadcastAddresses = false;
        }
        if (!validBroadcastAddresses)
                updateAddresses();
}
void PeerManager::readBroadcastDatagram()// 读取广播报文
{
        while (broadcastSocket.hasPendingDatagrams()) {
                QHostAddress senderIp;
                quint16 senderPort;
                QByteArray datagram;
                datagram.resize(broadcastSocket.pendingDatagramSize());
                if (broadcastSocket.readDatagram(datagram.data(), datagram.size(),
                                                 &senderIp, &senderPort) == -1)
                        continue;
                QList<QByteArray> list = datagram.split('@');
                if (list.size() != 2)
                        continue;
                int senderServerPort = list.at(1).toInt();
                if (isLocalHostAddress(senderIp) && senderServerPort == serverPort)
                        continue;
                if (!client->hasConnection(senderIp)) {
                        Connection *connection = new Connection(this);
                        emit newConnection(connection);
                        connection->connectToHost(senderIp, senderServerPort);
                }
        }
}
void PeerManager::updateAddresses()// 更新广播地址
{
        broadcastAddresses.clear();
        ipAddresses.clear();
        foreach (QNetworkInterface interface, QNetworkInterface::allInterfaces()) {
                foreach (QNetworkAddressEntry entry, interface.addressEntries()) {
                        QHostAddress broadcastAddress = entry.broadcast();
                        if (broadcastAddress != QHostAddress::Null && entry.ip() != QHostAddress::LocalHost) {
                                broadcastAddresses << broadcastAddress;
                                ipAddresses << entry.ip();
                        }
                }
```