分类号_____ 密级_____

UDC _____ 学号 2014002061

太原理工大学

毕业设计（论文）

论文题目 局域网设备控制管理系统

Thesis Topic Device Controlling and Management System in Local Area Network

学 生 姓 名	周宇豪
学 号	2014002061
所 在 院 系	信息与计算机学院
专 业 班 级	物联网工程 1402 班
导师姓名职称	李爱萍
完 成 日 期	2018 年 6 月 4 日

2018 年 6 月 20 日

太 原 理 工 大 学

毕业设计（论文）任务书

毕业设计（论文）题目：

局域网设备控制管理系统

毕业设计（论文）要求及原始数据（资料）：

1．综述国内外局域网设备控制研究现状；

2．深入了解 HTTP 服务器的相关技术及应用。

3．熟练掌握 Go,Java,HTML/JavaScript/CSS 的开发语句；

4．设计并实现局域网文件传输，键盘控制，摄像头直播的功能；

5．深入了解计算机程序并发，线程等技术；

6．训练检索文献资料和利用文献资料的能力；

7．训练撰写技术文档与学位论文的能力；

毕业设计（论文）主要内容：

1．综述本地服务器客户端开发现状；

2．分析相关需求，总体设计；

3．局域网文件传输，摄像头直播，键盘控制的应用和实现；

4．了解并发模型，HTTP 服务器架构知识；

5．Android 与桌面 GUI 程序的设计和实现；

6．多个客户端之间相互发现功能的实现；

7.系统测试；

8．实验结果分析和总结。

学生应交出的设计文件（论文）：

1．内容完整、层次清晰、叙述流畅、排版规范的毕业设计论文；

2．包括毕业设计论文、源程序等内容在内的毕业设计电子文档及其它相关材料。

主要参考文献（资料）：

[1]祝翔,董启文,郁可人.基于 Web-socket 的 PK 答题的设计与实现[J].华东师范大学学报(自然科学版),2018(02):89-100.

[2]林路智.Go 语言搭建网站解析[J].电脑知识与技术,2015,11(29):60-61.

[3].Google Go 语言将加速对 Android 平台的支持[J].电脑编程技巧与维护,2010(17):3-4.

[4]何兴鹏,刘钊远,陶琛嵘.Linux程序向Android平台移植的研究[J].计算机测量与控制,2018(05):112-115.

[5]江存.基于 Linux 的 2 种 HTTP 服务器实现与对比分析[J].现代计算机(专业版),2017(24):58-61+76.

[6]王伯槐,张烨.基于 Go 语言的消息推送平台的设计与实现[J].数码设计,2017,6(02):33-36.

[7]Pedro Luis Mateo Navarro,Diego Sevilla Ruiz,Gregorio Martínez Pérez. OHT: Open and cross-platform GUI testing[J]. <journal-title>Journal of Intelligent & Fuzzy Systems,2017,32(5):3231-3243.

[8]Mohamed Ibrahim AK,Lijo George,Kritika Govind,S. Selvakumar. Threshold Based Kernel Level HTTP Filter (TBHF) for DDoS Mitigation[J]. International Journal of Computer Network and Information Security(IJCNIS),2012,4(12):31-39.

[9]Nikhil Tripathi,Neminath Hubballi. Slow rate denial of service attacks against HTTP/2 and detection[J]. Computers & Security,2018,72:255-272.

[10]Edith Cohen,Haim Kaplan,Jeffrey Oldham. Managing TCP connections under persistent HTTP[J]. Computer Networks,1999,31(11):1709-1723.

专业班级 物联网 1402 班 学生 周宇豪

要求设计（论文）工作起止日期 2018 年 3 月 18 日~2018 年 6 月 14 日

指导教师签字 日期 2018 年 6 月 14 日

教研室主任审查签字 日期

系主任批准签字 日期

局域网设备控制管理系统

摘　要

在科技发展迅速的今天，每个人家里都至少有一台以上的智能电子设备，这些智能电子设备一般都是位于同一个家庭局域网下的。本文设计的局域网设备管理和控制系统，是基于 HTTP 协议的一个集文件传输、摄像头直播和输入设备控制为一体的多功能系统。

其功能主要包括：局域网设备之间的文件上传和下载，局域网设备之间的输入设备控制（目前只支持手机端控制电脑端），其中的输入设备包括键盘和鼠标，也就是说，你可以使用手机来控制电脑的键盘和鼠标。

除此之外，本系统还有一个额外的功能，那就是摄像头直播，这里所说的摄像头直播是指：手机将自己的摄像头画面实时直播到局域网上，其他的同一个局域网下的设备都可以查看这台设备的摄像头直播的画面。

本系统从上架到应用市场以来，经过了 3 个月的功能方面的完善和改动，然后有经过了 6 个月时间的不断修补和改善，系统的各项功能和模块都已经非常成熟，已经达到了可以商用的程度。目前所拥有的上万的用户数量，就是该系统的稳定与成熟的证明。

关键词：局域网传文件;Go 语言;本地 HTTP 服务器

Device Controlling and Management on Local Area Network

Abstract

In the rapid development of technology, everyone has at least one or more intelligent electronic devices in their homes. These intelligent electronic devices are generally located in the same home area network. The LAN device management and control system designed in this paper is a multifunctional system based on the HTTP protocol, which integrates file transfer, camera live broadcasting, and input device control.

Its functions mainly include: uploading and downloading files between LAN devices, and input device control between LAN devices (currently only supporting the mobile terminal to control the computer). The input devices include a keyboard and a mouse, that is, you can use The phone controls the computer's keyboard and mouse.

In addition, there is an additional function of the system, that is, the camera broadcast. The live camera mentioned here means that the mobile phone broadcasts its own camera screen to the local area network in real time. Other devices under the same LAN can View the live webcam of this device.

Since the system was put on the shelves and applied to the market, it has undergone three months of functional improvements and changes. After six months of continuous repair and improvement, the system's various functions and modules have been very mature and have reached Commercially available. The number of tens of thousands of users currently possessed is proof of the stability and maturity of the system.

Key words: Local Area Network file transfer; Go Program Language; Local HTTP Server

目录

1. 绪论

本章主要讲解了项目的开发背景，包括我个人在开发这个项目的时候灵感的来源，还有当时开发环境，当时的行业相关领域的背景和国内外行情。同时还介绍了本项目的常用的应用场景。

1.1 项目开发背景

起初做这个系统只是为了解决自己日常生活中的不方便，比如说：手机和电脑传文件的时候还需要数据线，我会觉得实在是太麻烦了！于是自己便随手做了一个 Android APP，这个 App 启动后会开启一个 HTTP 服务器，服务器能用 HTTP 协议来传文件，发布到应用市场上之后，没想到尽然收到了大量的好评。当然，评论之中，也不乏批评我做的不好的地方，但绝大多数的评论都是在给我提供一些建设性的意见，哪些地方做的不好，哪些地方可以这样这样改进，是时受宠若惊。于是我，带着不能辜负大众的一片热情的心情，开始花大量的时间去优化，完善这个系统。

时至今日（2018 年 6 月），此系统在酷安应用市场上已经有 6 万次下载（应用名称为：局域网精灵），6778 个关注者，540 条评论。同时，我已经把这个 App 的 Android 版如今已经上架到各大应用商店了：小米，百度，腾讯。其他应用商店下载量虽然不如酷安平台多，但也算是给使用其他应用平台的用户多一种下载渠道吧，就像我们互联网行业经常对用户说的一样：我可以不用，但是你不能没有，哈哈。虽然这样的一个成绩对于真正的开发者来说并不算多，但是，于我个人而言，是对我自身专业能力的莫大的鼓励。

要知道，在此之前，我尽历了大三下学期的求职阶段。以一个计算机相关专业的本科生的身份来说，大学四年的经历，与全国本科求职者比较，我的简历很普通。难以同其他来自全国各地的本科计算机学生争求同一职位。

我个人的性格是不喜欢与他人竞争，在自己的大学四年里没有参加过任何全国性的竞赛，也没有获得过任何奖学金，成绩也不过及格而已，毫无闪光点。要说我大学四年有何收获，我想应该是想明白了一些人或事，找到了自己生活的理由，给自己的的人生定义了价值取向。我想我以后不会再为“人一辈子活着有什么意义”之类的问题所困扰了。私以为这才是我在大学四年得到的最大收获。毛主席也说过：看待事物要优先抓住主要矛盾。相比漫无目的的奋斗，先找到正确的方向才是我大学阶段亟待解决的主要矛

盾。

当然我在大学四年也不是什么都没干，相比课堂和书本上的学习，我更喜欢自己动手在互联网上学习。从高中开始我就有了这样一个念头，如果没有老师教我了，我该怎么学习？这在高中的同学眼中是一个看起来杞人忧天的想法，现状也证明了我当时的想法的前瞻性。从大一开始逼迫自己去自学，锻炼独立思考问题的能力，独立寻找解决办法的能力。授人以鱼不如授人以渔。我认为一个真正的好老师，应该把学生教到不需要老师的程度才行！

本项目便是在我的自学过程之中诞生的一个点子！待此项目完善至终，也正好到了毕业设计的年份了。选了一个适合自己的题目，便顺势将其作为自己大学本科的毕业设计了。

1.2 项目简介

本文设计的局域网设备管理和控制系统，是基于 HTTP 协议的一个集文件传输、摄像头直播和输入设备控制为一体的多功能系统。

其功能主要包括：局域网设备之间的文件上传和下载，局域网设备之间的输入设备控制（目前只支持手机端控制电脑端），其中的输入设备包括键盘和鼠标，也就是说，你可以使用手机来控制电脑的键盘和鼠标。

除此之外，本系统还有一个额外的功能，那就是摄像头直播，这里所说的摄像头直播是指：手机将自己的摄像头画面实时直播到局域网上，其他的同一个局域网下的设备都可以查看这台设备的摄像头直播的画面。

1.3 国内外现状

处于在这个系统做出来之前，市面上并没有一个特别统一的，大众认可的局域网文件传输软件，也就是说，在这个领域，相对来说还是一个空白。

那个时候大家是怎么在电脑和手机之间传输文件的呢？

普通的用户一般是通过 QQ，来将电脑上的文件发送到手机上。这种方式有很大的缺点，就是传输速度受到带宽的限制，以我自己的网络为例。使用这种方式的传输速度最高也不过 2MB/s，但是如果使用我做的这个系统来传输文件，传输速度可以达到 6MB/s 甚至更高。

还有一种方式就是通过数据线将手机连接电脑，然后传输文件。这种方法虽然速度

快，但是效率低下，当传输一些小文件的时候，大部分时间都花在了找数据线上了。很多用户对此抱怨很深。

高级一点的用户会选择 ftp 等入门难度高的传输协议。这些协议对于普通用户来说很难理解，更加难以使用。最大的缺点是，在传输之前，你还需要给电脑安装一个 ftp 服务器。然而我这个系统在传输的时候，接受者是不需要安装任何客户端的，只要有浏览器就能进行所有的操作。况且浏览器是每一个操作系统都内置的软件。

可以说我这个系统做出来以后，受到大家欢迎的主要原因是解决的一部分网友日常生活中的痛点，抓住了痛点之后，大家自然喜欢。

1.4 应用场景

PPT 遥控器。平时大家在做 PPT 演讲的时候，都是用电脑插上投影仪，然后使用鼠标来切换 PPT 的下一页。这样很麻烦，因为演讲者不能离开电脑，必须在切换下一页的时候，回到电脑旁边，用鼠标点击才能切换。那有了本系统之后，用户就可以实现用手机遥控电脑，只要手机在手里，就可以不需要回到电脑旁边，一直演讲下去，对演讲者的站位会更加自由。

摄像头直播。因为本系统可以实现局域网内的摄像头直播，所以，只要让手机和电脑通过 Wi-Fi 连接在同一个局域网下，然后将手机的摄像头画面直播到局域网，电脑上就可以看到摄像头的画面。因为电脑本身携带不方便，很难移动，这样做可以实现用电脑观看隔壁房间的摄像头画面。因为是 Wi-Fi 连接，所以不需要任何线连接就可以实现手机变成移动摄像头这一梦幻功能。

文件传输。在我们的日常生活中，手机和电脑传文件一直是非常不方便的。有了我们做的这个系统之后，手机和电脑可以轻松的通过 Wi-Fi 来传输文件，而且稳定高效，不用担心数据线中途断开等问题。

另外，剪切板在电脑和手机之间的传输也是我们日常生活中的一大难题。如果电脑不安装 QQ 等软件很难传输，所以我们完全实现了在 Web 端的剪切板共享，从此，如果你在手机上看到什么网址，文字之类的，在手机上复制了之后，立马可以在电脑上接收到了，十分方便。

1.5 本章小结

本章主要介绍了本系统的主要的三大功能。以前经常在互联网上阅读一些关于产品

经理的思维方式的文章，颇有感触。就拿这些市面上的面向消费者的应用程序来说吧，我们以前是不叫应用的，我们以前都叫软件，或者程序。但是自从移动互联网到来之后，我们开始把软件和程序称之为应用了，这个词是来源于国外的单词 Application。虽然是直接从字面意思翻译过来的，但是如果我们以我们以往的认知来理解应用这个词还是感觉与真正我们要指代的意思是有失偏颇的，但是我们中国人对于一些新兴的词汇接受的速度是非常非常快的，这也是为什么移动互联网能够把像我们父母那一批以前在桌面互联网时代对于互联网一窍不通的人群能够带入进我们中国大批的网民大军里面来，因为当你开始重新创造一个词汇或者一个旧的词汇，但是我们给他赋予一个全新的意思（相对我们以往的认知来说），这个时候，人们对于该词汇的反应其实是和面对一个英文单词的反应是一模一样的，就好比把他当成了一个全新的单词开始认识。婴儿为什么学语言学的比大人快呢？因为他们每天都是在接触新的东西，所以就算是老人，当把他们放到一个全新的环境里面去，里面全都是他们有生之年没有遇到过的新事物，他们的学习速度也会非常快的。所以说年龄不是我们学习速度慢的接口，只是环境所至。

随着移动互联网的兴起，不但中国网民的数量越来越多了，每个人家里平均的电子设备数量也越来越多了，大家的家里现在大多数都不止一台手机，也许有两台手机，或者一台手机，一台电脑，甚至家里每个人都有一台手机，每个人都有一台电脑。当百姓家里的电子设备多了之后，对于电子设备的管理和控制就成了我们的新问题。如何才能高效的给家里的其他电子设备传输自己刚才下载的一个文件？如果快速的分享自己看到的一段文字到其他的设备上？这些都成为了我们每天都会遇到的小问题。所以，本文所介绍的系统，就是用来解决这些用户看似很小，实际上能够大大提高百姓日常生活中的工作效率的痛点的。

所以我认为本文所介绍的系统对于大部分家里有多台电子设备的人们是非常有意义的。

2. 需求分析

本章对项目的目标用户进行了一次大致的概述，对这个系统主要的功能有哪些进行了大致的描述。然后花了较大的篇幅对本系统在开发过程中使用到的技术进行了详细而又细致的讲解。

2.1 目标分析

随着互联网时代的到来，人们开始花越来越多的时间泡在网上。过去我们其实对于互联网是非常负面的态度。你比如说，过去，我们会把那些整天泡在网络上的人称为“网虫”，因为虫子移动速度慢，就好像他们每天坐在电脑面前，身体瘦弱，每天只有吃饭和上厕所的时候会离开电脑，开始走动，而且走动的时候都是慢吞吞的，迈着疲惫的步伐，就和毛毛虫一样，所以大家会叫这一类人“网虫”。然而今天，处在互联网盛世的我们，如今再回过头来看这一类人，其实会觉得心里很不是滋味。当初我们那么鄙视这一类人，他们每天面对的是家长的指责，还有朋友的嘲笑，如今这些最早接触互联网的人已经赚的盆满钵满了，那些没有赶上互联网潮流的人们到了现在也只能叹息。其实，仔细想一想，你会发现，每一次人类的科技革命都是发生在一些普罗大众之外的边缘群体，这些边缘群体都有一个共同的特征，他们喜欢接触新的事物，不会被世俗的眼光所困惑，不会被世俗的价值观所禁锢，而拥有这些特征的人群，基本上都是年轻人。这也是为什么马云走访各国演讲，每次都会强调要大家重视年轻人，正是因为年轻人没有包袱，再加上一腔热血，想象力比老人丰富，未来永远是属于年轻人的。

在 21 世纪这样一个数字时代，我们每个人都离不开互联网，就好像我们现在每家每户如今都离不开电一样。每个家庭至少有两部电子设备，这两部电子设备，或许是电脑，或许是手机。他们都或多或少地承载了当下人们每天的生活，工作，娱乐，和办公。我们每一个人，似乎都开始变成“网虫”了。当然，一旦家里的电子设备开始多了起来之后，我们对于电子设备的管理就成了新的问题。如何让局域网下的各个设备更加灵活的连接起来，是我们要解决的问题。

2.2 功能分析

2.2.1 功能简述

用户可以自由地分享和接受其他同一局域网下的设备的上传过来的文件。同时，其他同一局域网下的设备可以自由的访问分享者设备当前剪切板的具体内容。

当电脑客户端点击开启了受控端选项之后，同一局域网下的其他移动设备就可以发现，并向开启了受控端选项的电脑设备发送键盘指令，也就是我们通常所说的键盘控制。

用户开启应用之后，访问摄像头直播的 Web 页面，点击直播，即可向所有同一局域网下的其他设备广播自己摄像头所拍摄到的画面。

2.2.2 主要功能

1. 文件传输。用户打开 App 之后，选择用户想要分享的文件，这个时候 App 会在后台启动一个 HTTP 服务器，其他在同一个局域网下的其他设备只需要输入分享者的局域网 IP 地址，就可以访问用户刚才分享出去的文件，可以下载，还可以上传文件到分享者的手机上面去，电脑同理。

图 2-1 轻松传文件到电脑

2. 键盘控制。这一个功能需要电脑客户端先开启受控端选项，因为我们必须在受控者允许的情况下，才能让他接受其他设备发送过来的键盘指令。在受控端开启了受控选项之后，同一局域网下的其他设备就会自动发现有新的受控端设备上线了，然后其他设备就可以直接向该受控端发送键盘指令了。

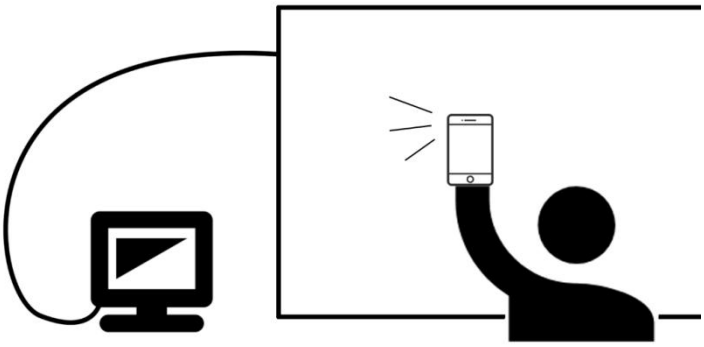

图 2-2 演讲的时候，用手机遥控 PPT

3. 摄像头直播。这一个功能是基于 Web 的，也就是说，只要局域网下有一个客户端开启了 Web 服务，其他的设备通过访问这个 Web 服务器，就同时都拥有了直播者和观看者两个功能了。在其他设备进入摄像头直播的 Web 页面之后，可以选择到底是要成为直播者，还是观看者。

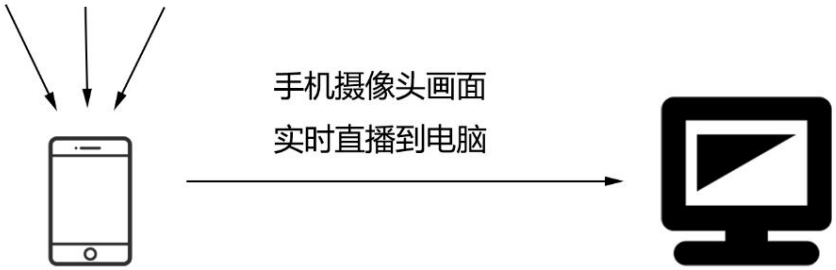

图 2-3 手机摄像头画面实时直播到电脑

2.2.3 主要技术

本文设计到的技术比较多，而且很复杂，下面我们分别详细介绍一下各个技术。

1. HTTP 协议

HTTP 协议其实是一个非常古老的技术了，过分一点的说，这个其实是上个世纪的技术了，但是不能说因为这个技术太过古老了，我们就不应该用它。电还是上上个世纪的产物呢，我们现在 21 世纪不也是照样还在用电呢？

如今 Google 等各大互联网公司一直在向大家建议使用一种新的 HTTPS 协议，其目的是为了让互联网更加安全，因为 HTTPS 是在 HTTP 的基础之上加了一个 SSL 加密层。以我个人的体验来说，HTTPS 虽然安全性提高了，但是有一个致命的缺点，那就是传输速度。由于 HTTPS 使用到了非对称加密方式，众所周知，非对称加密

方式的加密和解密的速度会大大降低，同时，传输速度还随着加密等级成指数上涨。所以说 HTTPS 是不适合我们这个系统的，原因有二：其一，HTTPS 传输速度慢，会影响文件传输模块的传输速度；其二：HTTPS 是需要一个共有证书的，大部分证书都需要花钱买，虽然有免费的证书吧，但是致命缺点是每过几个月这个免费的证书就会失效，这显然是不适合我们这种局域网下的 Web 项目的。既然都在一个局域网了，HTTPS 那种安全等级其实就没有必要了，因为局域网和局域网之间是有一个网关进行隔离的，如果有黑客想要对局域网下的设备进行攻击的话，是需要先把局域网的网关攻破才能有资格去进行下一层设备的黑客攻击。所以说局域网下其实对安全性的要求并不是很高。而且 HTTPS 的证书也很麻烦。

如今很多追求网速的网站依然使用的是 HTTP 协议，不是因为他有多么好，而是因为他兼容性好，简单，快速。作为一种基于 TCP/IP 协议的应用层协议，HTTP 对于数据的封装是经历过了时间的检验的。以往我们如果想在两个设备之间发送数据的话，是直接建立 Socket 连接，也就是普通的 TCP 连接。因为 TCP 协议是基于连接的，所以你在建立连接之后可以间断性的发送数据和接受数据。但是问题来了，如果你要间断性的发送数据，你如何检测两个数据之间的间隔呢？如何区分接收端收到的数据到底是一个数据还是两个数据呢？HTTP 就能解决这种数据包封装的问题，HTTP 协议支持很多种数据编码封装格式。比如说我们最常用的 application/x-WWW-form-encoded 编码格式，这个编码格式主要是用于对表单数据进行编码封装。他会在不同的数据之间加入&符号，用来将多个表单数据分隔开来。那玩意表单的内容就包含&怎么办？这个时候，HTTP 协议会将表单里的具体内容用 URL 编码方式进行编码，这种编码方式可以保证不会出现&等特殊字符，如果数据内容里面已经包含了特殊字符的话，他会将这些特殊字符转换成百分号开头的标识符，这样就实现了表单数据的编码与封装。

2. JavaScript 和 HTML5 标准

JavaScript 做为一门动态技术，一开始其实是符合 Web 行业的要求的。但是随着 Web 技术日新月异的发展速度，大家发现 JavaScript 越来越不适应当前 Web 行业的需求了，一个是因为 JavaScript 的性能问题越来越突出，另外一个是因为 JavaScript 的语法还不够灵活。但是后来 Google 的 Chrome 团队推出的 V8 引擎，拯救了 JavaScript。不仅让他的运行速度大大提升，同时也大大推动了新的 Web 技术的普及，因为这个技术的出现，chrome 浏览器的市场占有率一下子成为了第一名。

Google 作为 HTML5 技术的推动者，我们非常庆幸 chrome 浏览器是由这个公司开发的，因为没有 Google 对 Web 技术的大力推动，也许我们今天依然使用的是 HTML4 标准的浏览器，也许我们今天依然需要在网页上下载一个 Flash 插件才能看网页视频。Google 作为一个全球性的科技巨头公司，对于人类科技的发展的贡献无疑是巨大的。

当年推动 HTML5 标准普及的另外一个重要的原因，是来自全球各地各大技术黑客接二连三地向媒体曝光出来的 Flash 漏洞，这无疑为 Flash 离开历史舞台敲响了警钟。当人们开始抱怨网页 Flash 看视频的时候，CPU 风扇一直转个不停，当人们开始担心自己的电脑可能因为开启了 Flash 插件而遭到木马病毒的入侵的时候，HTML5 标准开始悄然诞生。HTML5 的标准的诞生，不仅仅是为了解决 Flash 领域的各种问题，同样也是为了不让互联网这一 21 世纪最伟大的发明就此堕落。要知道一个技术的发明，可以带动多少产业的更替，可以创造多少就业，虽然也会杀死很多行业，但这是历史的进程，不可改变。

HTML5 标准给 Web 行业带来了生机！我们终于可以不安装任何插件，就能在网页上面看视频了。我们终于可以不安装任何插件就可以在网上玩游戏了。HTML5 标准的到来，为互联网的媒体娱乐提供了很好的技术基础。与此同时，HTML5 也给 JavaScript 续了一命，给 JavaScript 带来了很多新的 API 和函数，让 JavaScript 在成为无所不能的编程语言的路上又前进了一步。

HTML5 标准中新增的摄像头等多媒体接口就是我们本文所做的系统实现的必要基础，没有了这个接口，我也不可能做出这样的系统来。

3. Web-Socket 技术

Web-Socket 同样是随着 HTML5 标准一起来到大众的视野中的。但是 Web-Socket 走的更远，他没有局限的 Web 浏览器上面运行，而是成为了一种数据通信的新兴标准。在目前的后端领域，人们在写 Web API 的时候，出来使用基于 HTTP 的 Restful API 之外，人们多了一个选择，那就是 Web-Socket。

初看 Web-Socket，大家可能会觉得这就是一个让 JavaScript 创建 TCP 连接的小玩意，为什么会如此流行呢？对，没错，Web-Socket 底层协议实现确实不复杂，和 Socket 连接相似，但是他却不止如此。Web-Socket 在 Socket 连接的基础之上增加了对数据的封装过程，也就是 Web-Socket 中的 Message 机制。这个机制是必要的，他成功的解决了 Socket 连接中数据的编码封装问题。而且 Web-Socket 相比 HTTP

Restful API 的最大的优势就是简单，速度快。

4. Go 语言

作为一门 Google 开发的编程语言，他不火是没有道理的。自古以来，那些能够火起来的编程语言，都有一个共同的特点，那就是他们有一个比较硬的后台撑腰。比如说 Java 就有 Sun 公司在背后撑腰，Swift 就有苹果公司在他的后面撑腰，C#就有微软在后面撑腰。对比和 Go 语言同一时期发布的全新编程语言 Rust，就能看出有一个强硬的后台是多么重要。同样式新出的编程语言，如今 Go 语言已经稳定在了编程语言排行榜的前 20 名了，然而 Rust 依然在 50 名开外。

当然，一门语言的流行肯定不止上面说的那一个原因，语言本身的特性和风格也是一门语言能否流行起来的因素之一。Go 语言也不例外。当人们谈论 Go 语言的时候，他们在谈论什么？性能？并发？跨平台？静态编译？都不是，其实真正让 Go 语言流行起来的原因是：简单。

简单这个词虽然只有两个字，但是，对一门编程语言来说，他却包含了非常复杂的内容。我以前有参加过一场微软的 Lang.Next 大会，会议的主题是关于各大编程语言的未来，和一些新增的特性。像 JavaScript、C#、Java 等都在大会上有提到过。这些语言在谈论自己新增的特性的时候，都会特别骄傲地说这让我们的编程语言更加地通用，更加强大了。但是我看到的却是，每一门语言都在不断的将其他语言的特性吸收过来，然后加到自己的语言之中，他们称之为语言新版本。这看似进步，然而我却开始担忧了。试想一下，在未来，如果每一个编程语言都这样做，所有的编程语言都开始趋向同质化，是多么恐怖的一件事情啊。当你想要学习一门新的编程语言来拓宽自己的思维，改变自己的思考方式的时候，发现所有的编程语言都是一样的套路，这一定程度上回减少编程界的多样性，也会一定程度的限制计算机编程领域的发展。

在所有的编程语言都在不停地给自己增加特性的时候，Go 语言却在给自己作减法。往一个项目里面加东西很容易，然而，要想作减法很难。你必须知道那些是必要的，那些是不必要的。Go 语言的设计者在谈论自己设计这门语言的设计哲学的时候，无疑都会提到一个词：大道至简。让一门大众使用的编程语言变得越来越简单，易学，同时保持灵活性，我想这也是 Go 语言的新用户越来越多的原因，因为学习门槛低，又能够事半功倍，何乐而不为。

Go 语言是谷歌开发的一款全新的编程语言，可以在不损失应用程序性能的同

事，大大降低代码的复杂度。

他的宗旨是让开发者更加容易地开发出简单、稳定、高效的软件！

为什么选择 Go 语言？（1）性能；（2）并发编程；（3）跨平台；（4）网络库丰富；（5）语法简单。

（1）**性能。**Go 语言针对多处理器系统应用程序专门进行了优化，在使用了 Go 编译之后，程序可以媲美 C 或 C++代码的速度，而且更加安全，而且还支持并行进程。

编译之后的 Go 语言代码的运行速度和 C 语言非常接近，而且编译速度非常快，就好像在使用一个动态的交互式编程语言一样。

目前，现有的各大流行的编程语言都没有对多核处理器进行专门的优化。而 Go 语言就是为了解决这一问题而诞生的。

（2）**并发编程。**时至今日，并发编程已经成为了程序员的基本技能了，在各大技术社区都可以看到诸多与之相关的讨论主题。到底那种方式是最佳的并发编程体验？或许会一直争论下去。但是 Go 语言却一反常态，从底层就将一切都并发化了！运行时使用了 Go routine 运行所有的一切，当然也包括 main.main 入口函数。

可以说，Go routine 已经成为了 Go 语言的标志性特征了。他使用类协程的方式来灵活地处理并发单元，同时却又在运行时层面做了更加深度的优化处理。这样，使得 Go 语言在语法上的并发编程变得极为简单！无需处理回调，无需关注执行时的切换，一切仅仅需要一个 go 关键字，简单而又自然！

搭配 Go 语言的 channel，可以让新手也能轻松实现 CSP 并发模型。将并发单元间的数据耦合拆解开来，各司其职。这对所有纠结于内存共享、锁粒度的开发人员来说都是一个可以期盼的大解脱！如果真的要说有什么不足，那就应该是要有一个更大的计划，将通信从进程内拓展到进程之外，实现真正意义上的分布式！

Go routine 和 Channel 使得编写高并发的服务器软件变得相当容易，很多情况下完全不需要考虑锁的机制和由此带来的一切问题。比如，单个的 Go 应用也能有效的利用多个 CPU 核心，并且执行的性能特别好。这个和 Python 相比有天壤之别！多线程的 Python 程序其实并不能有效地利用多核的优势，只能用多进程的方式来部署。如果使用标准库里面的 multiprocessing 包又会对监控和管理造成很多不必要的挑战。所以，部署一个 python 语言的时候，通常是每个 CPU 核心来部署一个应用，这样就会造成很多不必要的资源浪费。比如说假设某个 Python 应用启动之后

需要占用 100MB 内存，而服务器有 32 个 CPU 核心，那么留下一个核心给系统，而运行 31 个应用副本需要浪费 3GB 的内存资源，太糟糕了！

（3）**跨平台**。Go 语言支持几乎所有的 CPU 架构和操作系统。因为他的编译器由于底层使用了一种叫 Plan9 的汇编语言，相当于有了一个中间层，所以当 Go 语言再转移到另一个 CPU 指令架构的平台上去的时候就变得非常简单了，只需要在那个平台实现 Plan9 的汇编语言实现就可以了。

（4）**网络库丰富**。Go 语言标准库内置了 HTTP 协议实现，还有编写网络库常用到的各种加密算法。在目前特别火的，也是对网络和加密要求极高的区块链编程领域，基本上百分之百都是用 Go 语言开发的，这足以证明 Go 语言在这方面的实力。

（5）**语法简单**。Go 语言一共只有 20 多个关键字，对于编程初学者来说，不需要学习复杂的编程技巧，只需要安装官方的代码例子写，就能很容易写出高性能的代码。这也是为什么目前国内越来越多的公司开始招募 Go 语言开发者，因为入门快。

5. Android Studio 开发

在 Android 刚开始兴起的时候，相关的开发工具还是很少的。没有官方的 IDE，那个时候大家要想开发 Android 程序的话，需要使用 Eclipse 安装插件来开发。在当时看来，Eclipse 的确是当时最好的 Java IDE 了，直到有一个叫 Jet brain 的公司出现了。随着 Android 手机逐渐开始侵占 iPhone 的市场，Google 也开始愈来愈重视自己的这个操作系统了。于是基于 Jet brain 公司的产品，开发了 Android Studio 这样一个 Android 开发官方 IDE。其实一开始的时候，Android Studio 并不稳定，那个时候，所有的 Android 开发教程里面还是推荐使用 Eclipse 来开发。随着 Google 对 Android Studio 的优化技术的成熟，Android Studio 越来越开始有点官方 IDE 的样子了。如今，在网络上已经没有人会推荐使用 Eclipse 来开发 Android 程序了。但是，中国不一样。

在中国的高校里面，大学 Android 相关课程里老师们依然教学生使用 Eclipse 来开发 Android 程序。教育上面的技术落后，无疑拖慢了新老技术更替的速度。让新的技术的普及速度变慢了很多。这其中的原因，一个是因为中国互联网与外国互联网技术上的语言隔阂造成的，另一个是因为中国大学大部分教师年龄太大，接受新事物的速度变得特别慢，在教授年轻的学生知识的时候，依然使用的是老旧的工具，进

一步拖慢了新技术的普及速度。

一开始在实现的时候就是在 Android 平台实现的，等到用户量多了起来的时候，才开始扩展到其他平台。

目前 Android 应用程序开发的现状就是本地化要求比较高的 App 就使用 Native 开发，也就是用 Java 来开发；那些以内容为主的，对本地功能要求不高的，完全可以用网页替代的 App（如：百度贴吧，微博等），就是用一些 Web 技术来开发，也就是用 HTML,CSS 和 JavaScript 来开发。对于本系统而言，JavaScript 的功能自然是不足以满足我们的要求的啦，所以肯定是 Native 方式。但是依然存在一个问题。

因为我们是需要在本地建立一个 HTTP 服务器的，由于 HTTP 服务器本身结构复杂，Java 标准库里面又没有已经实现好的 HTTP 服务器，所以，HTTP 服务器底层实现部分我选择了使用 Go 语言来编写。

Go 语言官方团队构建了一个 Go Mobile 库，这个库是可以帮助你将 Go 语言的代码运行在 Android 上面的。虽然不能用 Go 语言来写 Android 的 UI，但是可以将 Go 语言代码以第三方库的形式嵌入到你的 Android 项目里面，然后使用 Java 来调用。有了这个库，我们就可以把让 Go 语言来写我们的 HTTP 服务器变成现实啦！因为 Go 语言本身就是为了写服务器而生的。所以使用 Go 语言写起来会非常顺手。

6. 桌面 GUI 框架

因为本系统需要做到跨平台，所以在系统实现的时候需要寻找合适的桌面 GUI 库。桌面 GUI 开发有很多选择，主要分为跨平台的 GUI 框架和非跨平台的 GUI 框架。其中我们重点关注跨平台的 GUI 框架。Qt。比较知名的跨平台框架，但是因为导出的安装包体积太大，所以放弃。SCITER。非常轻量级的 GUI 框架，使用 HTML 来写界面，导出的安装包体积也小。UI。来自 git-hub 上的一个第三方 GUI 框架，跨平台，原生支持，Go 语言。唯一的缺点是 GUI 组件太少。

综合考虑之后，最终选择了 SCITER 框架。

2.2 本章小结

本章主要介绍了局域网设备控制和管理系统下所使用到的一些关键性的技术，还有一些关于这些相关技术的一些闲谈。新老技术的更替是需要时间的，但是新技术代替老技术是必然的，因为这是历史的进程，不可改变。至于我们在实际生产的时候，到底是

使用最新的技术，还是使用老旧的技术，我个人认为，选择相对稳定的新技术，才是最好的平衡方法。

3. 系统总体设计

本章详细地介绍了整个系统的总体架构，以及每一个部分在具体设计过程之中的设计哲学，对为什么要这样设计作了详尽的描述。

3.1 系统总体设计

整个系统是又两个部分组成：网络核心库、UI 层。UI 层包括 Android UI 层，Linux UI 层，Windows UI 层，Mac UI 层。应用通过将网络核心库和 UI 层连接起来实现。

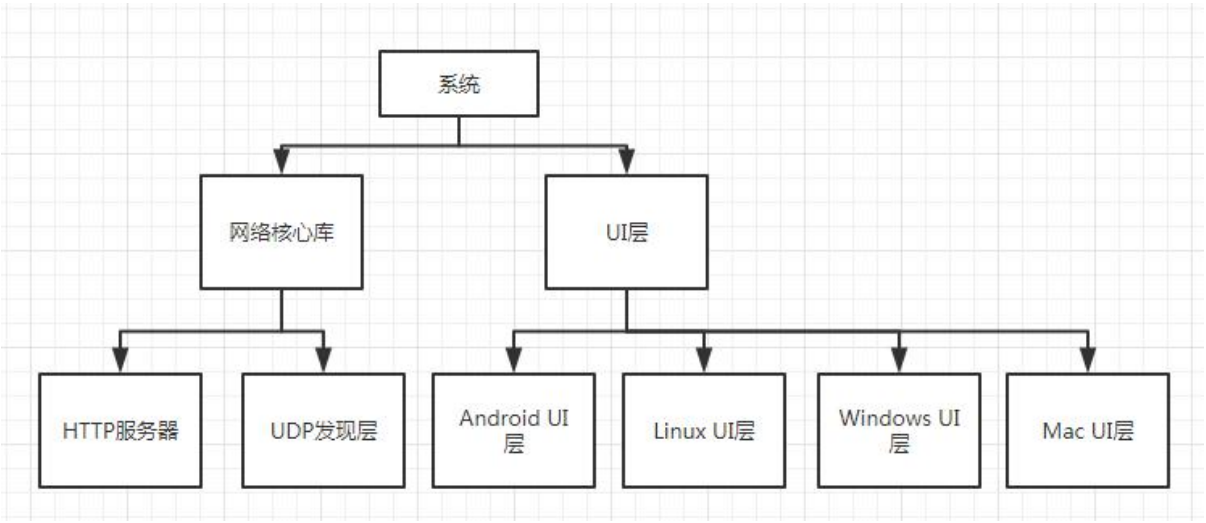

图 3-1 系统整体架构图

为什么要这样设计呢？由于本系统的目标是要做到跨多个平台(Windows,Linux,Android,Mac 等)，为了减轻开发和维护的负担，整个系统必须最大化地将通用的部分剥离出来，尽量做到一次编写，到处运行。

因为移动端(Android)和桌面端(Windows,Linux,Mac)的 UI 差异太大，所以无法实现 UI 层的共享。但是，网络核心库在各个操作系统中是通用的，可以共享。因为本系统对于 UI 界面没有太大的要求，所以网络核心库是整个系统编写过程中的重中之重！在完成网络核心库之后，再将网络核心库集成到各大操作系统的 UI 程序里面，通过一个事件回调接口(后面会在系统初始化里面详细讲解)来实现网络核心库与 UI 层的通信。这样就实现了系统的跨平台。

其中，网络核心库主要分为两大模块：HTTP 服务器模块、UDP 发现模块

3.2 HTTP 服务层

整个系统在启动的时候，第一步就是启动一个 HTTP 服务器。

这个 HTTP 服务器承载着文件传输、剪切板复制、摄像头直播和控制指令发送等功能。也就是说，基本上所有的模块都是基于这个 HTTP 服务器来传输信息和数据的。

3.2.1 HTTP 服务器初始化

HTTP 服务器启动之后第一件事情就是进行系统初始化操作。系统在初始化的时候，需要传入几个参数：事件回调接口、HTTP 服务器所要监听的端口号、临时文件夹的路径还有文件接收路径。如果所传入的参数为空的话，则使用默认的值。

(1) 事件回调接口(Event Handler)

事件回调接口(Event Handler)，是核心网络库向 UI 层通信的媒介。

为什么要设计这样一个接口？在 GUI 编程中，不管是桌面端，还是移动端。整个 UI 的操作、显示和修改，都是在同一个线程里面的，我们把这个线程叫做 UI 线程。在 UI 线程之中，是不允许进行一些耗时的操作的，比如说：Socket 监听，网络请求等。否则会出现程序的图形界面卡死的情况。

所以，在本系统中，因为需要启动一个 HTTP 服务器，这个 HTTP 服务器是绝对不能在 UI 线程中启动的，必须要另起一个新的线程。那么问题就来了，如果这个 HTTP 服务器有什么状态变化的情况（比如：HTTP 服务器收到一个上传过来的文件，收到控制指令等），如何去通知 UI 线程，并在图形界面之中显示出来呢？

我们只能设计这样一个接口，在 UI 层实现接口中定义好的方法，然后当 HTTP 服务器出现状态变化的时候，只需要调用该接口对应的方法即可。

事件回调接口是一个接口(Interface)，定义在网络核心库之中。里面定义了一些事件的方法，但是没有实现他们。这些方法是需要由 UI 层来实现的。因为 UI 层各有不同，比如收到的消息如何显示？在 Android 上是显示一个 Alert Dialog，在 Windows 上则是弹出一个 Alert Window。所以这些事件回调的方法，必须是由不同的 UI 层自己去实现，无法共享。

事件回调接口中包含一下事件方法：

- 当收到剪切板的时候
- 当收到 Web 上传的文件的时候

- 当收到其他客户端上传来的文件夹的时候
- 当新的设备上线或者下线的时候
- 当收到远程控制指令的时候
- 存储键值对
- 获取键值对

为什么事件回调接口中还需要包含存储键值对、获取键值对两个方法？系统在初始化的时候需要读取配置文件，这些配置文件是以键值对的形式存储在设备之中的。然而，在不同的操作系统之中，存储配置信息的方式各有不同。比如，Windows 是存储在 App Data 目录下的，Linux 是存储在用户目录下的隐藏文件夹里面的，而在 Android 系统中，配置信息是存储在 Shared Preferences 之中的。所以，我们需要抽象出一个存储键值对方法，和一个获取键值对的方法，来让各大平台各自实现自己的存储过程即可。

(2) HTTP 服务器监听的端口号

HTTP 服务器要监听的端口号(Port)，是在启动 HTTP 服务器时的必要参数。他有自己的默认值，用户可以在设置界面里面更改端口号。但是，因为服务器一旦启动了，端口号就不能修改了，所以当用户在设置界面里面修改了端口号之后，必须重启整个系统才能使修改生效。

(3) 临时文件夹路径

临时文件夹路径(Temp Path)，该参数是为了防止 Temp 目录空间不足的情况出现(Linux 下默认 Temp 目录存储空间大小为 4G)，于是让用户可以自定义 Temp 目录位置。

(4) 文件接收路径

文件接收路径，是该参数是在启动 HTTP 服务器时的必要参数，他有自己的默认值，用户也可以对其进行修改。在整个系统之中，该参数是以一个共有变量的形式存储在内存之中的。也就是说，如果用户在设置界面里面修改了该参数，则不需要重启系统，即可使之生效。

3.3 UDP 发现层

守护进程(Daemon)，是整个网络核心库的两大主要模块之一。主要的作用是进行同一局域网下的在线设备相互发现(后面会在“设备发现模型”部分详细讲解)。

3.3.1 守护进程初始化

守护进程初始化，是在 HTTP 服务器初始化之后进行的。初始化的时候，只需要一个参数即可：操作系统的标识。

操作系统的标识，主要作用不过是让用户能够快速识别其他在线设备。

3.3.2 设备发现模型

为什么需要“在线设备互相发现”这一功能？在同一局域网之中，当有多台设备同时开启了本系统时。用户可以进行一些更加高级的操作，比如：远程控制，向指定设备发送文件夹等。如果没有在线设备相互发现这一功能的话，用户需要自行查询每台机器的局域网 IP 地址，同时还要自己一个字一个字填写 IP 地址和端口号，况且 IP 地址和端口号的输入过程十分麻烦且耗时。

为了解决这一问题，为了让用户能够无需输入在线设备的 IP 地址，用户唯一需要做的事情就是在所有在线设备列表里面选择自己想要的设备即可。所以我们增加了 UDP 守护进程，主要用于同一局域网下的在线设备互相发现。

在线设备发现模型的要求

- 当设备上线的时候，要广播通知所有局域网下的已有设备
- 当设备下线的时候，所有局域网下的其他设备也要实时更新在线设备列表
- 设备需要让所以同一局域网下的其他设备知道自己的状态(如：是否开启了接收控制指令的选项)
- 当设备的状态改变时，要实时地在其他设备的在线设备列表里面更新

本系统的设备发现模型包含三个部分：广播发现、心跳连接和一个状态获取 HTTP 接口。

(1).广播发现

广播发现部分，是整个设备发现模型的基础，也是实现整个模型的过程的第一步。当设备启动本系统的时候，UDP 守护进程初始化，同时会向所在的局域网广播一条上线提醒消息。但是，消息之中并不会包含”上线“等复杂表示，该消息仅仅只包含一个信息：那就是本机的 HTTP 服务器端口号。也就是说，所谓的广播发现，不过是一个向局域网下的全部设备广播一下自己的 HTTP 服务器端口号的简单操作罢了。

为什么只包含一个 HTTP 服务器端口号？

广播发现这一行为是整个设备模型的基础，但也仅仅只是基础而已，他不会包含太多的功能或者作用。在系统启动的时候，向局域网的所有设备广播自己的 HTTP 服务器端口号，当其他在线设备收到这一个 UDP 广播消息的时候，仅仅意味着：”当前局域网下的在线设备列表产生变化了，需要更新一下在线设备列表了！“。

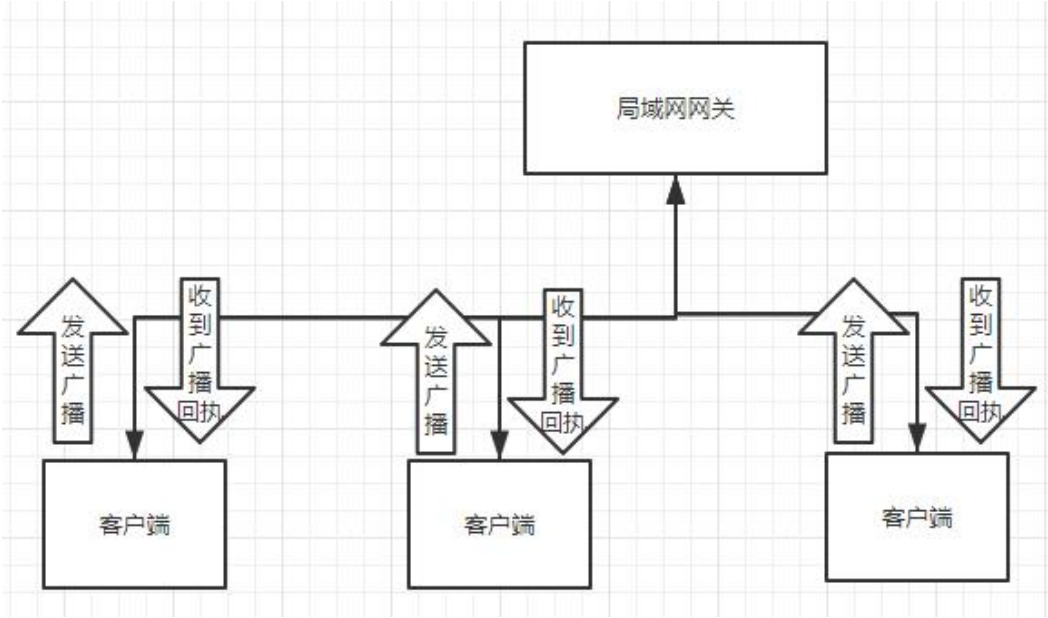

图 3-2 UDP 发现过程

虽然仅仅只包含一个简单的 HTTP 服务器端口号，但它却像一个跳板，告诉其他设备：我的状态改变了(这里的状态不仅仅包括上线、下线，还有在用户开启或关闭了”接收控制指令“选项的时候，也属于状态变化的范畴，这个时候也会向局域网下的所有设备广播自己的 HTTP 服务器端口号。)，快利用你们收到的 HTTP 服务器端口号，在我的 HTTP 服务器上获取我的具体状态信息。

(2).心跳连接

在设备发现模型之中，为了满足“当设备下线的时候，所有局域网下的其他设备要实时更新自己的在线设备列表”这一要求，我们必须要让每一台设备直接都建立一个心跳连接，这个心跳连接不会进行任何数据的传输，他的唯一作用就是让其他的设备知道自己是在线的。如果这个心跳连接断开的话，就说明此设备已经离线了。通过这样一种方式，就实现了“当设备下线的时候，所有局域网下的其他设备要实时更新自己的在线设备列表”。

(3).状态获取 HTTP 接口

每一台设备都需要设置这样的一个状态获取 HTTP 接口。因为 HTTP 协议是无状态的，这个接口的作用很简单，不管是谁来访问这个接口，都会向访问者返回本设备的实

时状态信息，返回之后则立即断开连接。

每当收到在线设备列表变更的消息的时候，每台设备都会访问所有设备的状态获取HTTP 接口。如果无法访问，则说明该设备已经离线，如果能访问，则将返回来的设备实时状态信息覆盖原有的设备列表中的信息。

3.4 文件传输模块

整个文件传输模块由两部分组成：文件部分和剪切板部分

3.4.1 文件部分

在文件部分的数据结构设计上，每一个文件个体包含三个信息：文件名，文件的绝对路径和文件的 ID。

为什么要设计文件 ID 这一信息？

由于文件传输模块是建立在 HTTP 服务器之上的，所以，为了防止不同的文件在的URL 下载地址出现冲突，URL 不能以文件名作为标识，又因为文件路径包含的“/”符号与 URL 的分隔符冲突，所以也不能以文件路径作为文件 URL 的唯一表示。只能在每一个文件添加了之后，用一个随机生成的数字作为每一个分享出去的文件的唯一标识，这就是文件 ID 的由来。

文件部分包含四个 HTTP 页面或接口，分别是：

- 文件列表页面
- 文件下载接口
- 文件预览接口
- 文件上传接口

文件部分是文件传输模块的重要组成部分，其未来的发展方向还有很多，改进的方法也有很多，比如使用并行下载来提升文件传输的速度等。

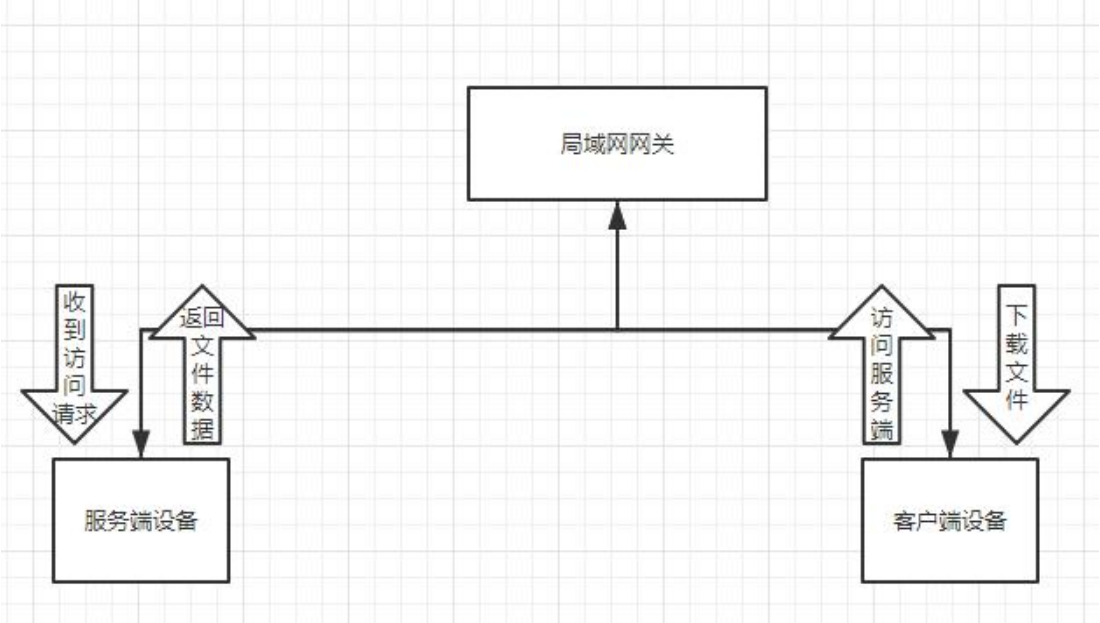

图 3-3 文件传输过程

3.4.2 剪切板部分

剪切板部分比较简单，同样是给予 HTTP 服务器的。主要包含两个共有变量，一个用于存储剪切板的内容，另一个用于存储剪切板开启的状态。

为什么需要剪切板部分？

剪切板是人们日常使用手机的过程中最常用到的功能之一。比如平时大家在手机上复制一段文字，复制一个网址，想要传给电脑怎么办呢？以前的办法是通过 QQ 或者微信发送给电脑端，然后电脑端再复制一下。这样有两个缺点：一个是电脑必须联网，另一个是电脑还必须安装 QQ 或者微信的客户端，如果说还有什么缺点的话，我认为应该是二次操作。所谓的二次操作是指手机复制了一次剪切板，发送到电脑上之后，电脑还得再复制一次。很不方便。所以我决定在本系统之中加入剪切板功能，就是为了方便大家在家具智能设备之间共享剪切板。

3.5 远程控制模块

远程控制模块包括控制者和被控制者。目前的版本，控制者只能是手机，被控制者只能是电脑。

3.5.1 控制者

控制者通过发送将控制指令以 HTTP 请求的方式发送给被控制者

3.5.2 被控制者

被控制者通过在 HTTP 服务器上新增一个接口，专门用于接收 HTTP 控制指令。当收到 HTTP 控制指令的时候，再又 UI 层去解析并执行控制指令

3.5.3 权限问题

被控制者默认是不接受任何控制指令的，这个时候如果有控制者想发送控制指令过来，会返回无效信息。只有当被控制者开启受控端选项的时候，才会接收控制指令。在“受控端”开启的时候，有两种选项：第一种是接收任何人的控制指令，第二种是接收指定设备的控制指令，第三种是接收除了指定设备以外的所有控制指令。

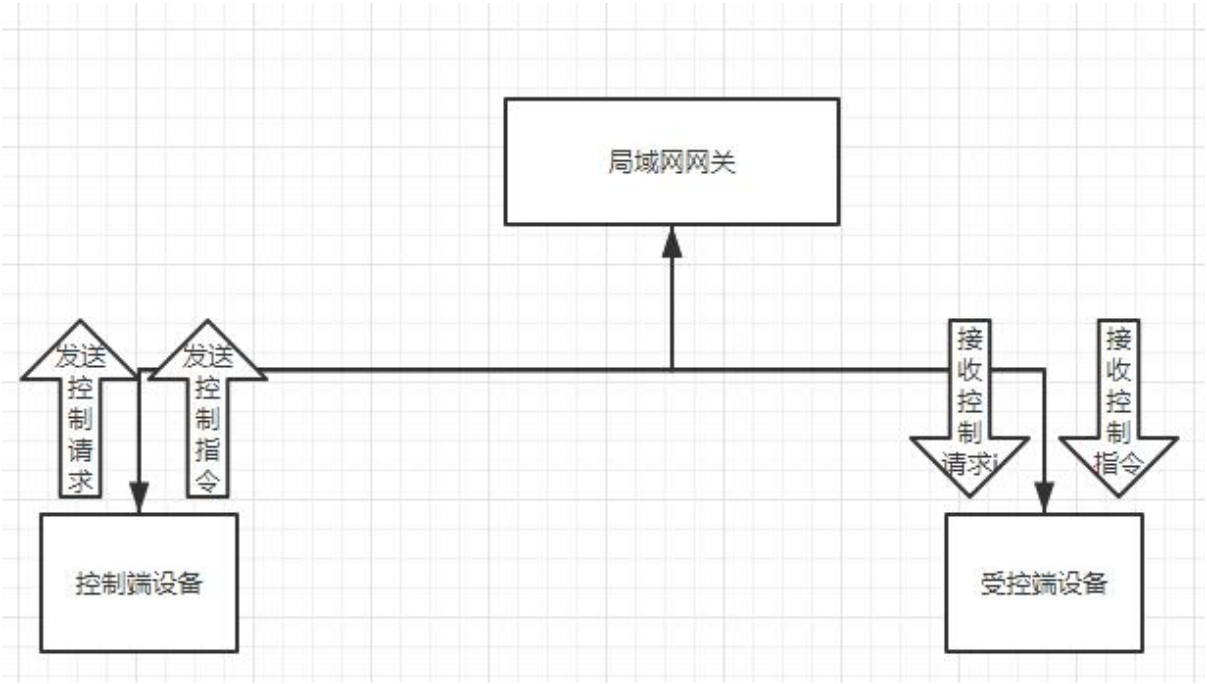

图 3-4 远程控制过程

3.5.4 申请控制权限的过程

如果控制者是在受控端的可控设备列表之外的话，需要先向受控端申请控制权限，受控端同意了之后，才能进行远程控制。

3.6 摄像头直播模块

摄像头直播模块是基于 HTTP 服务器的，主要由广播，Web-Socket 长连接，直播者页面和观看者页面组成。

3.6.1 .Web-Socket 长连接

Web-Socket 长连接是将所有观看者和直播者通过 HTTP 服务器连接起来的媒介。之所以选择 Web-Socket 是因为这是目前最新的 Web 技术————HTML5 推荐的保持长连接的方法。摄像头直播时的广播就是依靠这一长连接来实现的。

3.6.2 广播

广播是在 HTTP 服务器收到直播者在 Web 浏览器上发过来的摄像头数据的时候，将这一数据由服务器向所有在线的观看者发送过去的行为。

没当有新的用户连接到摄像头直播页面，就会与服务器建立一个 Web-Socket 连接，同时，将该连接加入所有已连接的数组里面。这样一来，当直播者需要广播数据的时候，只需要遍历一下这个数组，然后逐一发送即可。

3.6.3 直播者页面和观看者页面

直播者页面和观看者页面是由 HTML5 技术实现的，通过调用 HTML5 浏览器接口，获取摄像头数据，然后通过 Web-Socket 发送给服务器，在观看者收到摄像头数据之后，利用 Canvas 画布显示出来即可。

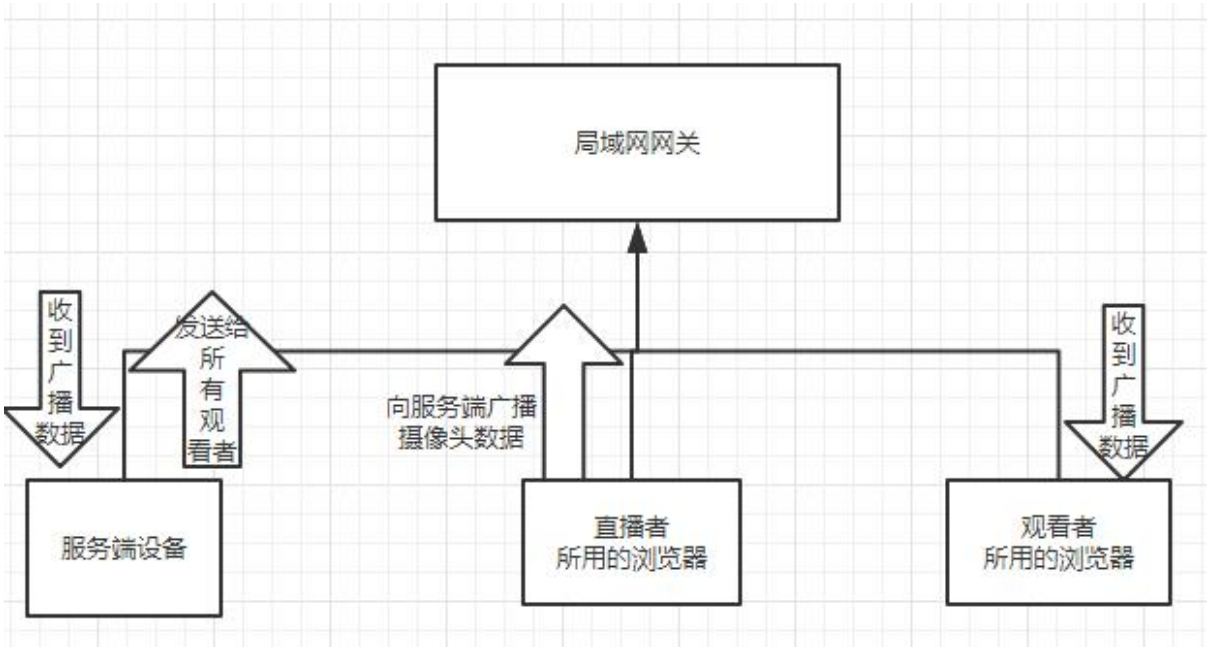

图 3-5 摄像头直播过程

3.7 本章小结

在本系统中设计的这个设备发现模型，在设计模式上，完美的实现了分而治之的思想。将设备实时状态更新这一功能按照类型分给了三个部分：广播发现，心跳连接和状态获取 HTTP 接口。他们分别代表了三个角色：通知者，监视者和展示者。每一个角色自身的功能极其简单，但是三个角色合作起来，却又能正好完成一项复杂的工作。这也是我个人在 Go 语言的学习过程之中领悟到的：解耦。这样一来，就不会像其他的设备发现模型那样，出现无限的广播反弹的现象，而且又简单。正所谓：把一件简单的事情做复杂很容易，但是能把一件复杂的事情简化的人才是大师。

4. 开发与实现

本章主要讲系统具体是如何实现的，其中附带了较多的系统各个部分的关键代码，和运行测试的截图。

4.1 开发环境

表 4-1 开发环境列表

编号	名称	开发或运行环境
1	客户端	Android,Windows,Linux,Mac
2	开发工具	Android Studio,Sublime Text 3,Go
3	硬件设备	小米 5A 一台，小米笔记本 Air 一台

4.2 HTTP 服务器

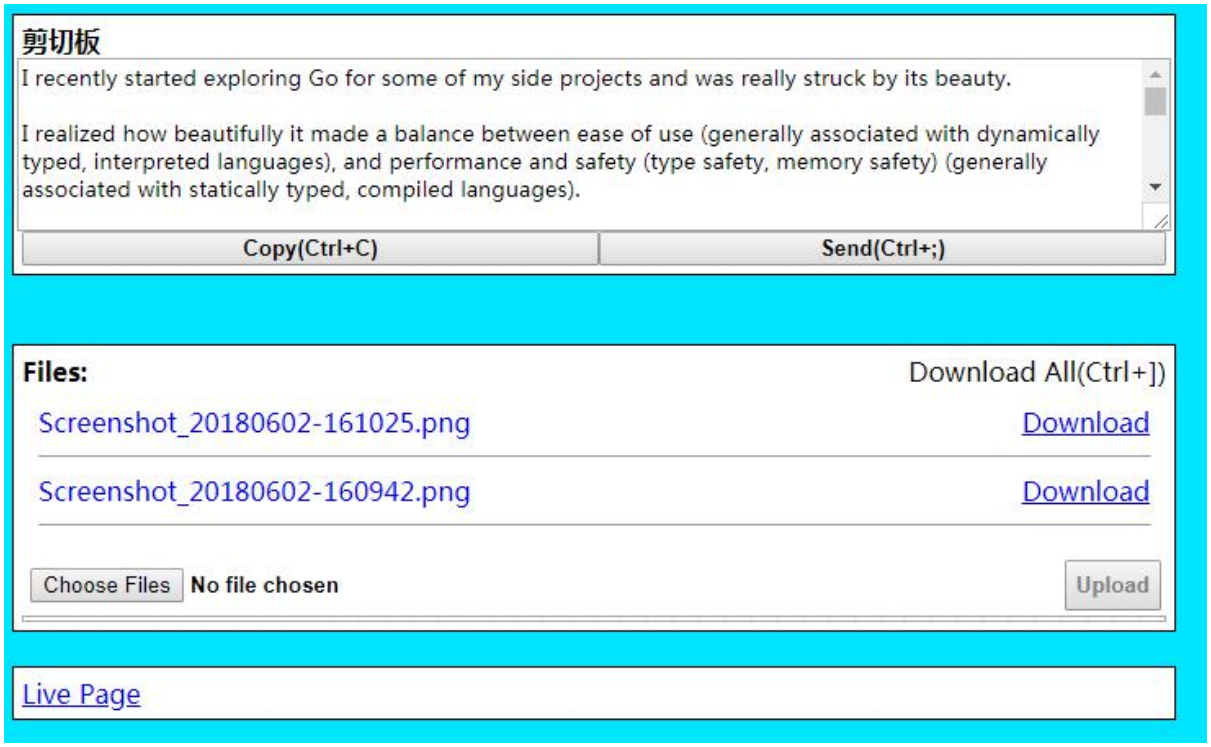

图 4-1 Web 文件传输页面

理论上来说，任何语言都可以用来写 HTTP 服务器，但是不同的语言都有自己的优劣之分。选择什么样的语言来写 HTTP 服务器，咱们先看一看目前各大互联网企业的服

务器都是用什么语言写的。

Java 是目前服务端使用最多的语言，对于一些大型的网站，云计算，服务器都是用 Java 写的。但是，缺点是太过庞大，因为我们其实做的是一个面向普通用户的软件，软件的体积必须要尽可能的小，如果你写一个服务器，要运行的话还要用户再去安装一个 JRE，那用户体验肯定就太差了。再者，Java 本身没有将 HTTP 协议写进标准库里面去，也就是说，你需要使用第三方的 HTTP 框架来完成，第三方框架最出名的也就是 Spring 框架了，这个框架也是太过庞大了，之适用于大型网站，不够轻巧。

PHP、Python、C#也都可以写服务器，但是，因为我们需要全平台支持，如果不能再手机端运行的话，就不能使用这个语言。

Go 语言是我们最合适的选择，他支持全平台，简单，标准库内置 HTTP 协议实现，不需要第三方 HTTP 框架。同时运行速度还非常快。

用户在 Web 文件传输界面可以点击 Download 下载别人分享出来的文件，也可以直接点击文件名预览图片等文件，同时下面也可以上传本机文件到手机上。

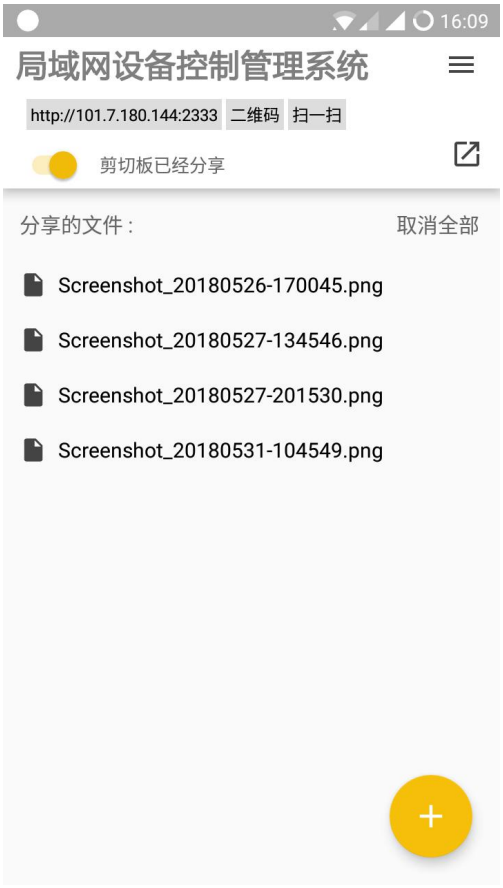

图 4-2 Android 文件分享界面

核心代码为

```
mf,e:=os.OpenFile(MyStoragePath+path+v.Filename,
os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
defer mf.Close()
io.Copy(mf, file)
```

4.3 键盘控制

为了实现键盘和鼠标的控制，我们找到了一个第三方库：

Robot-Go

这个第三方库实现了跨平台的键盘鼠标输入控制，也就是说，可以在 Go 语言代码里面控制键盘和鼠标的输入。比如执行指令"enter"，就相当于按了键盘上的回车按键。另外一个，关于组合按键，他是这样解决的。例如：输入"control alt t"这样的命令，以空格来隔开，就可以实现组合按键的功能。

本系统在实现的时候也是原封不动地使用了该第三方库的指令系统，在控制端输入指令之后，通过 HTTP 协议发送给受控端，然后受控端执行指令即可。

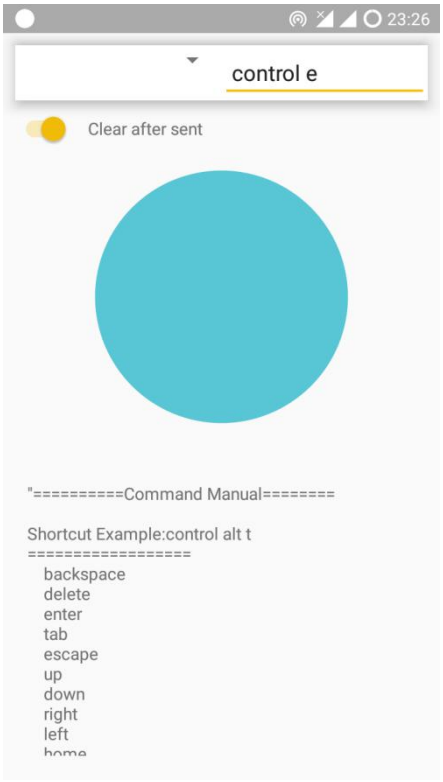

图 4-3 Android 控制端界面

键盘控制指令的核心代码为

```
b, e := ioutil.ReadAll(r.Body)
if e != nil {
    fmt.Println("readAll() failed:", e)
    return
}
MyEventHandler.OnRemoteControlCmdReceived(string(b))
```

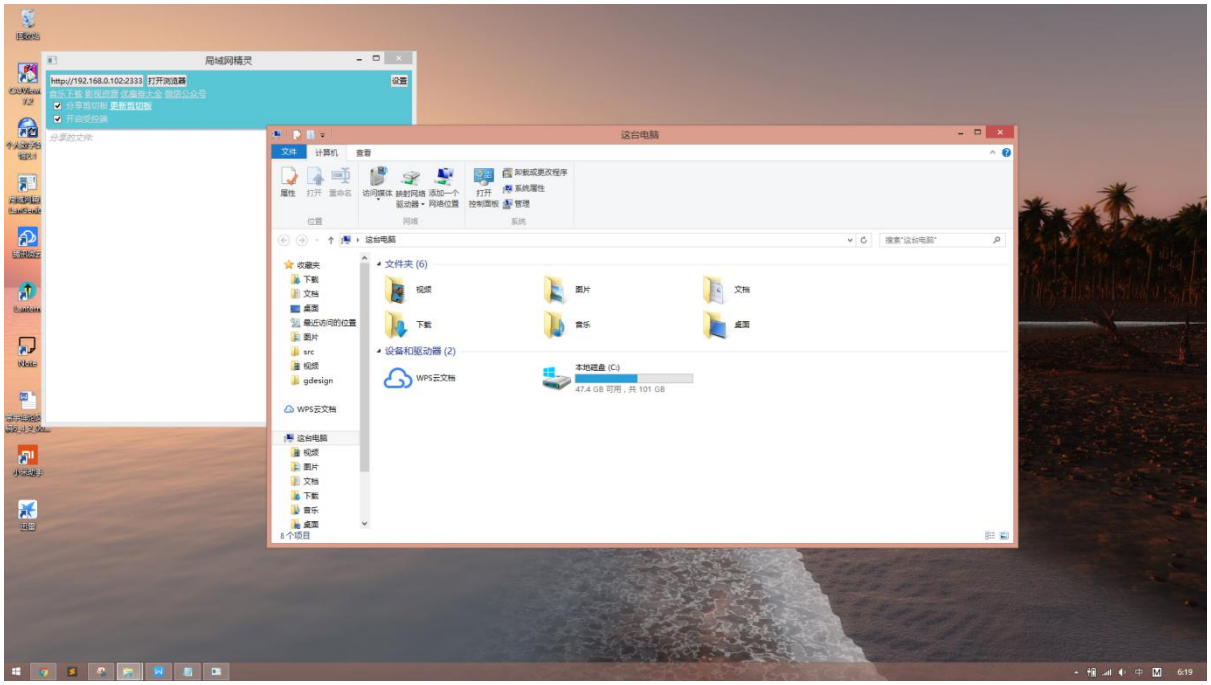

图 4-3 Windows 受控端收到 command e 指令后的显示结果

4.4 摄像头直播

摄像头直播功能主要是参考了网上的一个第三方摄像头直播示例代码，然后有了灵感。

为什么在 Web 端实现？

整个摄像头直播功能主要是在 Web 端实现的，也就是说需要借助浏览器来调用系统的摄像头接口。这和大家以往想象的摄像头直播软件不同。之所以这么实现，一个是因为 Go 语言本身就有很多这种类型的例子；其二是因为在 Web 端实现的话，另一台设备不需要安装客户端了，只要有一个浏览器即可实现直播者或者观看者的角色。我觉得

这样会方便很多。

首先直播者在 Web 端点击一个按钮，JavaScript 便通过 Web-Socket 连接到 HTTP 服务器，然后用 JavaScript 通过 HTML5 的摄像头 API ——navigator.GETUSERMEDIA()函数，来申请摄像头画面数据。拿到摄像头画面数据之后呢，把这些数据通过刚才连接上的 Web-Socket 发送给服务器，服务器收到数据之后，则对所有的 Web-Socket 连接进行广播。至于观看者这边，同样，也是用 Web-Socket 与服务器进行连接，然后监听数据。如果服务器发送数据过来了，观看者这边再用 JavaScript 把画面显示在 HTML 中的 IMG 标签里面即可。

图 4-4 Android 直播者画面

摄像头直播的关键代码是：

```
func sendToAll(str string) {
    for k, v := range members {
        if e := websocket.Message.Send(v, str); e != nil {
            members = append(members[:k], members[k+1:]...)
```

```
                break
            }
        }
    }
```

Web 端的实现代码为：

```
<video id="sourcevid" controls></video>
<canvas id="output" style="display:none"></canvas>
<script>
    var data,v=
    document.getElementById("sourcevid"),
    mcavas=document.getElementById("output"),
    mcavasContext=mcavas.getContext("2d")
</script>
```

5. 总结

5.1 遇到的问题

5.1.1Android 临时文件夹问题

在用户上传文件到手机上的时候，因为大文件在接收的时候肯定是不能存在内存里的，我们是需要做一个本地缓存文件，然后对接收到的数据进行分段合并。这个本地缓存文件在 Linux 上本来是存储在临时文件夹的。但是在最近的几个 Android 版本出来了之后，Android 在操作系统级别上去掉了原来的临时文件夹功能。如果有程序试图使用临时文件夹来存储分段的缓存文件，就会出现 permission denied 的错误。

经过我的仔细调查之后，找到了解决办法。那就是修改 Go 语言运行时的临时文件夹环境变量，这样，Go 语言运行时的临时文件夹就会修改成当前程序有权限的文件夹。

5.2 感想与展望

做软件的开发是一件非常有意思的事情。因为你在编程的时候，其实是一种创造的过程，把一个东西从无到有，这个过程中每个人都是造物主。在我第一次接触编程之后，我发现自己的眼界变得开阔了，不仅仅局限在编程过程中有这种感受。在生活中，我也会习惯性地开始以创造者的角度来思考问题，在这个思考的过程中，我学到了很多很多。

关于本文所讲的系统，我个人认为以后还是会有很多提升的空间的。比如在系统上面加入 P2P 功能，让用户的文件传输不仅仅局限在局域网里面，让所有的用户可以在公网上进行文件传输。当然这实现起来可能会有很多困难，但是这不妨碍这个功能成为吸引人的特性。

参考文献

[1]祝翔,董启文,郁可人.基于WebSocket的PK答题的设计与实现[J].华东师范大学学报(自然科学版),2018(02):89-100.

[2]林路智.Go 语言搭建网站解析[J].电脑知识与技术,2015,11(29):60-61.

[3].Google Go 语言将加速对 Android 平台的支持[J].电脑编程技巧与维护,2010(17):3-4.

[4]何兴鹏,刘钊远,陶琛嵘.Linux 程序向 Android 平台移植的研究[J].计算机测量与控制,2018(05):112-115.

[5]江存.基于 Linux 的 2 种 HTTP 服务器实现与对比分析[J].现代计算机(专业版),2017(24):58-61+76.

[6]王伯槐,张烨.基于 Go 语言的消息推送平台的设计与实现[J].数码设计,2017,6(02):33-36.

[7]Pedro Luis Mateo Navarro,Diego Sevilla Ruiz,Gregorio Martínez Pérez. OHT: Open and cross-platform GUI testing[J]. <journal-title>Journal of Intelligent & Fuzzy Systems,2017,32(5):3231-3243.

[8]Mohamed Ibrahim AK,Lijo George,Kritika Govind,S. Selvakumar. Threshold Based Kernel Level HTTP Filter (TBHF) for DDoS Mitigation[J]. International Journal of Computer Network and Information Security(IJCNIS),2012,4(12):31-39.

[9]Nikhil Tripathi,Neminath Hubballi. Slow rate denial of service attacks against HTTP/2 and detection[J]. Computers & Security,2018,72:255-272.

[10]Edith Cohen,Haim Kaplan,Jeffrey Oldham. Managing TCP connections under persistent HTTP[J]. Computer Networks,1999,31(11):1709-1723.

致谢

本次毕业设计，首先要感谢我的指导老师李爱萍，我的毕业设计是在李老师的指导下完成的，老师给予了我很大的帮助。并为具体开发以及论文的书写提出了许多宝贵意见，他严谨仔细的治学态度深深的影响了我，在此对李老师的帮助表示衷心的感谢！

在这里还要特别感谢和我一起工作学习的同学，通过跟他们的讨论也解决了我在设计过程中的一些疑惑，增长了我的见识，拓宽了我的视野，同时也让我发现了自身的不足，谢谢你们给予我的帮助。

最后我要感谢我的母校太原理工大学，在这里我度过了令我终身难忘的四年，四年中，我收获了很多，不仅收获了知识，还收获了一帮志同道合的师长和朋友，感谢他们一直以来的支持和鼓励。

外文原文

I recently started exploring Go for some of my side projects and was really struck by its beauty.

I realized how beautifully it made a balance between ease of use (generally associated with dynamically typed, interpreted languages), and performance and safety (type safety, memory safety) (generally associated with statically typed, compiled languages).

Apart from these, two more features make it really the perfect language for modern systems development. Both these features are explained in more detail in the Strengths section below.

One of them is first class support for concurrency in the language (through go routines and channels, explained below). Concurrency, by its design, enables you to efficiently use your CPU horsepower. Even if your processor just has 1 core, concurrences design enables you to use that one core efficiently. That is why you can typically have hundreds of thousands of concurrent go routines (lightweight threads) running on a single machine. Channels and go routines are central to distributed systems since they abstract the producer-consumer messaging paradigm.

The other feature I really like about Go is interfaces. Interfaces enable loosely coupled or decoupled components for your systems. Meaning that a part of your code can just rely on an interface type and doesn’ t really care about who implements the interface or how the interface is actually implemented. Your controller can then supply a dependency which satisfies the interface (implements all the functions in the interface) to that code. This also enables a really clean architecture for unit testing (through dependency injection). Now, your controller can just inject a mock implementation of the interface required by the code to be able to test if it’s doing its job correctly or not.

Keeping all these features in mind, I think Go is really a great language. Especially for use cases like cloud systems development (web servers, CDN, caches etc), distributed systems, micro services etc. So if you’ re an engineer or a start up trying to decide what language you want to explore or try out, do give Go a serious thought.

Introduction

Go is an open source language, created at Google by Robert Grimes, Rob Pike, and Ken Thompson. Open source here means that everybody can contribute to the language by opening proposals for new features, fix bugs etc. The language’s code is available on Git Hub. Documentation on how you can contribute to the language is provided here.

Why was Go needed

The authors mention that the primary motive for designing a new language was to solve software engineering issues at Google. They also mention that Go was actually developed as an alternative to C++.

Rob Pike mentions the purpose for the Go programming language:

“Go’s purpose is therefore not to do research into programming language design; it is to improve the working environment for its designers and their coworkers. Go is more about software engineering than programming language research. Or to rephrase, it is about language design in the service of software engineering.”

Issues that were plaguing the software engineering horizon at Google were (taken from HTTP://talks.clangor.org/2012/splash.article):

a) slow builds—builds would sometime take as long as an hour to complete

b) uncontrolled dependencies

c) each programmer using a different subset of the language

d) poor program understanding (code hard to read, poorly documented, and so on)

e) duplication of effort

f) cost of updates

g) version skew

h) difficulty of writing automatic tools

i) cross-language builds

For Go to succeed, Go must solve these problems (taken from HTTP://talks.clangor.org/2012/splash.article):

a) Go must work at scale, for large teams of programmers working on them, for programs with large numbers of dependencies.

b) Go must be familiar, roughly C-like. Google needs to get programmers productive

quickly in Go, means that the language cannot be too radical.

c) Go must be modern. It should have features like concurrency so that programs can make efficient use of muti core machines. It should have built-in networking and web server libraries so that it aids modern development.

Target Audience

Go is a systems programming language. Go really shines for stuff such as cloud systems (web servers, caches), micro services, distributed systems (due to concurrency support).

Strengths

a) Statically typed: Go is statically typed. This means that you need to declare types for all your variables and your function arguments (and return variables) at compile time. Although this may sound inconvenient, this is a great advantage since a lot of errors will be found at compile time itself. This factor plays a very big role when your team size increases, since declared types make functions and libraries more readable and more easier to understand.

b) Compilation Speed: Go code compiles really fast, so you don’t need to keep waiting for your code to compile. :) In fact, the ‘go run’ command fires up your Go program so quickly so that you don’t even get a feeling that your code got compiled first. It feels like an interpreted language.

c) Execution Speed: Go code gets directly compiled to machine code, depending upon the OS (Linux/Windows/Mac) and the CPU instruction set architecture (x86, x86‐64, arm etc) of the machine the code is being compiled upon. So, it runs really fast.

d) Portable: Since the code gets directly compiled to machine code, therefore, the binaries become portable. Portability here means that you can pick up the binary from your machine (let’s say Linux, x86‐64) and directly run that on your server (if your server is also running Linux on a x86‐64 architecture).

This becomes possible since Go binaries are statically linked, meaning that any shared operating system libraries your program needs are included in the binary at the time of the compilation. They are not dynamically linked at the time of running the program.

This has a huge benefit for deployment of your programs on multiple machines in a data center. If you have 100 machines in your data center, you can simply ‘Sc your program

binary to all of them, as long as the binary is compiled for the same OS and instruction set architecture your machines run on. You don't need to care about which version of Linux they are running. There is no need for checking/managing dependencies. The binaries simply run and all your services are up :)

e) Concurrency: Go has first class support for concurrency. Concurrency is one of the major selling points of Go. The language designers have designed the concurrency model around the 'Communicating Sequential Processes' paper by Tony Hare.

The Go run time allows you to run hundreds of thousands of concurrent go routines on a machine. A Go routine is a lightweight thread of execution. The Go run time multiplexes those go routines over operating system threads. That means that multiple go routines can run concurrently on a single OS thread. The Go run time has a scheduler whose job is to schedule these go routines for execution.

There are two benefits of this approach:

i) A Go-routine when initialized has a stack of 4 KB. This is really tiny as compared to a stack of an OS thread, which is generally 1 MB. This number matters when you need to have hundreds of thousands of different go routines running concurrently. If you would run more than thousands of OS threads in parallel, the RAM obviously will become a bottleneck.

ii) Go could have followed the same model as other languages like Java, which support the same concept of threads as OS threads. But in that case, the cost of a context switch between OS threads is much larger than the cost of a context switch between different go routines.

Since I'm referring to "concurrency" multiple times in this article, I would advise you to check out Rob Pike's talk on 'Concurrency is not parallelism". In programming, concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations. Unless you have a processor with multiple cores or have multiple processors, you can't really have parallelism since a CPU core can only execute one thing at a time. On a single core machine, it's just concurrency that's doing its job behind the scenes. The OS scheduler schedules different processes (threads actually. every process has at least a main thread) for different time slices on the processor. Therefore, at one moment in time, you can only have one thread(process) running

on the processor. Due to the high speed of execution of the instructions, we get the feeling that multiple things are running. But it’s actually just one thing at a time.

Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

f) Interfaces: Interfaces enable loosely coupled systems. An interface type in Go can be defined as a set of functions. That’s it. Any type which implements those functions implicitly implements the interface, i.e. you don’t need to specify that a type implements the interface. This is checked by the compiler automatically at compile time.

This means that a part of your code can just rely on an interface type and doesn’t really care about who implements the interface or how the interface is actually implemented. Your main/controller function can then supply a dependency which satisfies the interface (implements all the functions in the interface) to that code. This also enables a really clean architecture for unit testing (through dependency injection). Now, your test code can just inject a mock implementation of the interface required by the code to be able to test if it’s doing its job correctly or not.

While this is great for decoupling, the other benefit is that you then start thinking about your architecture as different micro services. Even if your application resides on a single server (if you’re just starting out), you architect different functionaries required in your application as different micro services, each implementing an interface it promises. So other services/controllers just call the methods in your interface not actually caring about how they are implemented behind the scenes.

g) Garbage collection: Unlike C, you don’t need to remember to free up pointers or worry about dangling pointers in Go. The garbage collector automatically does this job.

h) No exceptions, handle errors yourself: I love the fact that Go doesn’t have the standard exception logic that other languages have. Go forces developers to handle basic errors like ‘couldn’t open file’ etc rather than letting them wrap up all of their code in a try catch block. This also puts pressure on developers to actually think about what needs to be done to handle these failure scenarios.

i) Amazing tooling: One of the best aspects about Go is its tooling. It has tools like:

i) Go FMT: It automatically formats and indents your code so that your code looks like the same as every Go developer on the planet. This has a huge effect on code readability.

ii) Go run: This compiles your code and runs it, both :). So even though Go needs to be compiled, this tool makes you feel like it’ s an interpreted language since it just compiles your code so fast that you don’ t even feel when the code got compiled.

iii) Go get: This downloads the library from Git Hub and copies it to your Go Path so that you can import the library in your project

iv) Go doc: Go doc parses your Go source code — including comments — and produces its documentation in HTML or plain text format. Through Go doc's web interface, you can then see documentation tightly coupled with the code it documents. You can navigate from a function’ s documentation to its implementation with one click.

There is a lot of development going on in the Go horizon. You can find all Go libraries and frameworks for all sorts of tools and use cases here.

Weaknesses

1. Lack of generics — Generics let us design algorithms around types-to-be-specified-later. Let’ s say you need to write a function to sort a list of integers. Later on, you need to write another function for sorting a list of strings. At that moment, you realize that the code would pretty much look the same but you can’ t use the original function since the function can either take a list of type integer or a list of type string as an argument. This would require code duplication. Therefore, generics let you design algorithms around types which can be specified later. You can design an algorithm for sorting a list of type T. Then, you can call the same function with integers/strings/any other type given that there exists an ordering function for that type. Meaning that the compiler can check if one value of that type is bigger than another value of that type or not (since this is needed for sorting)

There should ideally be some way for dependency version so that you can simply include the version number of a 3rd party library in your dependency file. Even if their API changes, you don’ t need to worry about it since the newer API will come with a newer version. You can later go back to check what changes were made and then take a decision on whether or not to upgrade the version in your dependency file and change your client code according to the changes in the API interface.

Go’s official experiment dep should ideally become the solution to this problem soon. Probably in Go 2 :)

中文翻译

我最近开始探索 Go 语言的一些项目，并且被它的美丽所震撼。

我意识到它在易用性（通常与动态类型化，解释性语言相关）和性能和安全性（类型安全性，内存安全性）（通常与静态类型，编译语言相关）之间取得了平衡。

除此之外，还有两个功能使其成为现代系统开发的完美语言。这两个功能在下面的优势部分中有更详细的解释。

其中之一是对语言并发性的一流支持（通过 go routines 和渠道，下面解释）。并发，通过其设计，使您能够有效地使用您的 CPU 马力。即使您的处理器只有 1 个内核，并发的设计也能让您高效地使用该内核。这就是为什么您通常可以在单台机器上运行数十万个并发 go routines（轻量级线程）的原因。渠道和 go routines 是分布式系统的核心，因为它们抽象了生产者 - 消费者的消息范例。

我非常喜欢 Go 的另一个特性是接口。接口为您的系统提供松耦合或分离组件。这意味着你的代码的一部分可以只依赖于接口类型，并不关心谁实现了接口或接口是如何实现的。然后，您的控制器可以提供一个满足接口（实现接口中的所有功能）的代码的依赖关系。这也为单元测试提供了一个非常干净的架构（通过依赖注入）。现在，您的控制器可以注入代码所需的接口的模拟实现，以便能够测试它是否正确地执行其工作。

记住所有这些功能，我认为 Go 是一门很棒的语言。特别是对于像云系统开发（Web 服务器，CDN，缓存等），分布式系统，微服务等用例。因此，如果你是一名工程师或初创企业试图决定你想要探索或尝试什么语言，那么给 Go 一个认真的想法。

介绍

Go 是一款开源语言，由 Robert Grimes，Rob Pike 和 Ken Thompson 在 Google 创建。这里的开放源代码意味着每个人都可以通过为新功能提供建议，修复错误等来为语言做出贡献。该语言的代码在 Git Hub 上提供。这里提供了有关如何为语言做出贡献的文档。

为什么需要 Go

作者提到，设计新语言的主要动机是解决 Google 的软件工程问题。他们还提到 Go 实际上是作为 C ++的替代品而开发的。

Rob Pike 提到 Go 编程语言的目的：

“因此，Go 的目的不是研究编程语言设计;它是为了改善设计师和同事的工作环境。

Go 比编程语言研究更关注软件工程。或者换句话说，就是关于软件工程服务中的语言设计。“

困扰 Google 软件工程视野的问题（摘自 HTTP://talks.clangor.org/2012/splash.article）：

a）缓慢的构建 - 构建有时需要一个小时才能完成

b）不受控制的依赖

c）每个程序员使用该语言的不同子集

d）程序理解不佳（代码难以阅读，记录不当等等）

e）重复努力

f）更新的成本

g）版本歪斜

h）编写自动工具的难度

i）跨语言构建

为了成功，Go 必须解决这些问题（摘自 HTTP://talks.clangor.org/2012/splash.article）：

a）围绕大量程序员开展工作，围绕大量依赖的程序必须大规模开展工作。

b）Go 必须是熟悉的，大致类 C。谷歌需要在 Go 中快速提高程序员的效率，这意味着语言不能太激进。

c）Go 必须是现代的。它应该具有像并发这样的功能，以便程序可以高效地使用多核心机器。它应该有内置的网络和 Web 服务器库，以便它有助　　于现代化的发展。

目标听众

Go 是一种系统编程语言。对于诸如云系统（网络服务器，缓存），微服务，分布式系统（由于并发支持）而言，Go 确实非常出色。

优势

a）静态类型：Go 是静态类型的。这意味着您需要在编译时为所有变量和函数参数（以及返回变量）声明类型。虽然这听起来不方便，但这是一个很大的优势，因为在编译时本身会发现很多错误。当你的团队规模增加时，这个因素起着非常重要的作用，因为声明的类型使得函数和库更易读，更容易理解。

b）编译速度：Go 代码编译速度非常快，因此您无需继续等待代码编译。 :)事实上，'go run'命令会很快启动你的 Go 程序，所以你甚至不会感觉到你的代码是先编译好的。这感觉就像一种解释性语言。

c）执行速度：根据操作系统（Linux / Windows / Mac）和代码正在编译的机器的CPU 指令集体系结构（x86，x86-64，arm 等），Go 代码直接编译为机器代码。所以，它运行速度非常快。

d）便携式：由于代码直接编译为机器码，因此，二进制文件变得便携。这里的可移植性意味着你可以从你的机器（比如 Linux，x86-64）获取二进制文件，并直接在你的服务器上运行（如果你的服务器也在 x86-64 架构上运行 Linux）。

由于 Go 二进制文件是静态链接的，这意味着您的程序需要的任何共享操作系统库都将在编译时包含在二进制文件中。它们在运行程序时不会动态链接。

这对于在数据中心的多台机器上部署程序具有巨大的好处。如果您的数据中心中有100 台机器，只要将二进制文件编译为您的机器所运行的相同操作系统和指令集体系结构，就可以简单地将您的程序二进制文件“SCP”到所有这些机器。你不需要关心他们正在运行的 Linux 版本。不需要检查/管理依赖关系。二进制文件只是运行，你的所有服务都在运行:)

e）并发性：Go 对并发有一流的支持。并发是 Go 的主要卖点之一。语言设计师围绕托尼霍尔的“沟通顺序过程”论文设计了并发模型。

Go 运行时允许您在机器上运行数十万个并发 Go-routine。Go-routine 是一个轻量级的执行线程。 Go 运行时将这些 Go-routine 复用到操作系统线程上。这意味着多个Go-routine 可以在单个操作系统线程上同时运行。 Go 运行时有一个调度程序，其任务是调度这些 go routines 执行。

这种方法有两个好处：

i）初始化时的 Go-routine 具有 4 KB 的堆栈。与一个一般为 1 MB 的 OS 线程堆栈相比，这非常小巧。当你需要同时运行几十万个不同的 Go-routine 时，这个数字很重要。如果你要并行运行数千个 OS 线程，RAM 显然将成为瓶颈。

ii）Go 可以遵循与 Java 等其他语言相同的模型，它支持与 OS 线程相同的线程概念。但是在这种情况下，OS 线程之间的上下文切换成本比不同的 Go-routine 之间的上下文切换成本要大得多。

由于我在本文中多次提及“并发性”，因此我建议您查看 Rob Pike 关于“并发性不是并行性”的讨论。在编程中，并发是独立执行的进程的组成，而并行则是（可能相关的）计算的同时执行。除非你有一个拥有多个内核的处理器或者拥有多个处理器，否则你不能真正拥有并行性，因为 CPU 内核一次只能执行一件事。在单个核心机器上，只

有并发才是幕后工作。 OS 调度程序针对处理器上的不同时间片调度不同的进程（实际上线程，每个进程至少有一个主线程）。因此，在某个时刻，您只能在处理器上运行一个线程（进程）。由于指令的执行速度很快，我们感觉到有很多事情正在运行。但实际上这只是一件事。

并发是一次处理很多事情。并行是一次做很多事情。

f）接口：接口使松散耦合的系统成为可能。 Go 中的接口类型可以被定义为一组函数。而已。任何实现这些函数的类型都会隐式地实现接口，即不需要指定类型实现接口。这由编译器在编译时自动检查。

这意味着你的代码的一部分可以只依赖于一个接口类型，并不关心谁实现了接口或接口是如何实现的。然后你的主/控制器函数可以提供一个满足接口（实现接口中所有函数）的依赖关系。这也为单元测试提供了一个非常干净的架构（通过依赖注入）。现在，您的测试代码可以注入代码所需的接口的模拟实现，以便能够测试它是否正确地执行其工作。

虽然这对于解耦是非常好的，但另一个好处是您可以开始将您的体系结构视为不同的微服务。即使您的应用程序驻留在单个服务器上（如果您刚刚开始），也可以将应用程序中所需的不同功能设计为不同的微服务，每个微服务都实现它承诺的接口。所以其他服务/控制器只是调用界面中的方法，而不是实际关心它们是如何在幕后实现的。

g）垃圾收集：与 C 不同，你不需要记住释放指针或担心 Go 中悬挂指针。垃圾收集器自动完成这项工作。

h）没有例外，自己处理错误：我喜欢 Go 没有其他语言具有的标准异常逻辑的事实。去强迫开发人员处理“无法打开文件”等基本错误，而不是让他们将所有代码包装在 try catch 块中。这也迫使开发人员实际考虑需要采取什么措施来处理这些故障情况。

i）惊人的工具：关于 Go 的最好方面之一是它的工具。它有如下工具：

i）Go FMT：它会自动格式化和缩进你的代码，这样你的代码看起来就像这个星球上的每个 Go 开发者一样。这对代码可读性有巨大的影响。

ii）运行：编译你的代码并运行它们，都是:)。因此，即使 Go 需要编译，这个工具也让你觉得它是一种解释型语言，因为它只是编译你的代码的速度非常快，以致于当代码编译完成时你甚至不会感觉到它。

iii）转到：从 Git Hub 下载库并将其复制到 Go Path，以便您可以将库导入到项目中

iv）Go doc：Go doc 解析您的 Go 源代码 - 包括注释 - 并以 HTML 或纯文本格式生成其文档。通过 Go doc 的网络界面，您可以看到与其所记录代码紧密结合的文档。只需点击一下，您就可以从函数的文档导航到其实现。

Go 的发展有很多发展。你可以在这里找到所有的 Go 库和框架，用于各种工具和用例。

弱点

1.泛型的缺乏 - 泛型让我们在稍后指定待指定的类型时设计算法。假设您需要编写一个函数来对整数列表进行排序。稍后，您需要编写另一个函数来排序字符串列表。在那一刻，你意识到代码几乎看起来一样，但你不能使用原始函数，因为函数可以将一个整数类型列表或一个字符串类型列表作为参数。这将需要代码重复。因此，泛型允许您围绕稍后可以指定的类型设计算法。您可以设计一个算法来排序 T 类型的列表。然后，您可以使用整数/字符串/任何其他类型调用相同的函数，因为存在该类型的排序函数。这意味着编译器可以检查该类型的一个值是否大于该类型的另一个值（因为这是排序所需的）

理想情况下应该有一些依赖版本的方法，这样你就可以简单地在你的依赖文件中包含第三方库的版本号。即使他们的 API 改变了，你也不需要担心，因为新的 API 将带有更新的版本。您稍后可以回头查看所做的更改，然后决定是否升级您的依赖文件中的版本并根据 API 接口中的更改更改您的客户端代码。

Go 的官方实验 dep 应该很快成为这个问题的解决方案。可能在 Go 2 :)