局域网设备控制管理系统

摘　要

演讲的时候用手机遥控PPT，WiFi传文件，摄像头直播等功能。对局域网设备的控制和管理，可以为人们日常生活能提供较大的便利。

关键词 局域网；控制；管理

**Device Controlling and Management System**

**in Local Area Network**

#### Abstract

Control your PowerPoint with your smart phone , transfer file through WiFi , live your camera on Local Area Network .etc . The controlling and management of devices on Local Area Network , can make your daily life more convenient .

**Key words:** Local Area Network , Controlling , Management

1　绪论

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。

2 课题背景

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？

1. .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。
2. 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。
3. 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。3 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WiFi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WiFi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。

文件传输和剪切板共享

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WiFi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。

4 本文主要研究内容

本文主要研究的内容有：HTTP的文件传输协议、局域网广播发现的原理和模型设计、通用网络库的实现和部署、Web摄像头直播的实现、桌面操作系统的鼠标和键盘控制原理。

4 原理

整个系统主要是基于一个HTTP服务器上实现的，因为要尽量贴近浏览器的使用体验，我们把摄像头直播也做到了Web浏览器上面，这样一来又可以实现电脑和手机多设备进行摄像头直播了。

4.1 文件传输

文件传输的原理是使用了HTTP协议的POST方式传输的。

当然了，POST方法的表单格式（也就是enctype类型）有很多种，比如application/x-www-form-urlencoded格式，是专门用于上传文字表单数据的；text/plain是专门用来上传普通文字的，和前者相比之下，他不需要进行url编码，但是我们依然不能选择这种方式来上传文件，因为用户上传的文件的内容不一定时文本格式，也有可能是二进制文件，二进制文件就不能使用这个编码格式了。

最后我们选择的是multipart/form-data表单格式，这种格式编码方式相对前面两种来说会比较复杂，但是他的容错率比较高，他会随机生成一个bound字符串，用来对多个文件内容之间进行边界区分。有了这样的一种编码方式，我们就可以轻松的实现多文件上传了。

另外，在上传的时候，因为需要获取上传的进度，所以我们必须使用AJAX来进行上传控制。

至于文件的下载，很简单。因为我们在手机上启动了一个HTTP服务器，当用户要下载文件的时候，只需要实现HTTP协议的文件解析函数http.ServeFile()就可以了，这个函数已经被写进了官方标准库里面，所以我们不需要二次实现了，直接调用就行了。而且因为下载者是浏览器，所以我们不需要自己去实现文件下载的进度和速度显示了，因为浏览器都有这个功能。

4.2 网络发现

因为我们需要实现当两台设备同时在局域网下启动的时候，两个客户端要能够自动相互发现对方，而且设备的上线、下线和状态的改变都需要向局域网下的其他设备进行广播，所以我们需要一个网络发现模块。

这个网络发现的模块的原理其实是来自UDP协议，因为UDP是不基于连接的，就好像写信一样，我发送出去之后，是不知道对方有没有收到的。所以UDP有这个优势，来实现网络广播。这也给我们的功能实现奠定了基础。

在UDP协议之中，255.255.255.255这个地址是一个保留地址，当操作系统向这个地址发送任何UDP数据包的时候，网关会将这个数据包分发给所有局域网下的设备。这样就实现了UDP广播的功能。

这就是网络发现的原理。

4.3 Web摄像头直播

摄像头直播功能最初是起源于我哉github上面看到的一个项目，该项目使用了Go语言作为后端，然后前端使用了websocket技术和canvas等HTML5技术来实现，当时觉得很神奇，于是我就仔细地研究了一下他的代码，然后安装自己的理解，在自己的系统之上实现了一遍。

他的关键点不是在于如何获取浏览器的摄像头权限，因为获取摄像头权限很简单，只需要一个函数就可以搞定。关键在于他获取到摄像头的视频数据之后，如何对数据进行打包，编码，从而达到视频流的广播效果。我觉得这一点才是难点。

我以前有接触过一点H.264编码格式的视频流编码，很复杂。但是我看玩他的代码之后发现，他使用了一种非常简单易懂的实现方式来完成视频流的编码。在视频权限获取之后，将返回的视频数据导出到canvas画布上，然后转换为图片二进制数据。然后将图片二进制数据进行Base64编码，通过WebSocket进行局域网广播。在其他客户端收到这个Base64编码的图片数据之后，可以直接显示在img标签里面，因为在HTML5标准之中，img标签本身就是支持Base64编码的图片数据显示的，这样就大大降低了数据编码的复杂度。

4.4 键盘鼠标控制

鼠标和键盘的控制其实并不难，因为本身所有操作系统都直接向C语言提供了鼠标和键盘控制的API，在C语言里面直接调用就可以了。但是，我们不可能用C语言来写HTTP服务器啊！这就成为了本系统实现鼠标和键盘控制的最大难点。但是，多亏了Go语言里面的Cgo功能，可以实现Go语言直接调用C语言代码，这样就方便的实现了HTTP服务器的同时，还能直接调用操作系统的API。

5 设计

5.1 整体架构

整个系统是又两个部分组成：网络核心库、UI层。

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守护进程模块

5.2 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。

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(EventHandler)

事件回调接口(EventHandler)，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AlertDialog，在Windows上则是弹出一个AlertWindow。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AppData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SharedPreferences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。

5.3 UDP发现层

守护进程(Deamon)，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)。

5.3.1 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。

5.3.2 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

设备发现模型

【图】

如图，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！“。

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。

5.3.3 总结

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。

5.4 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

5.4.1 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。

5.4.2 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。

5.5 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。

5.5.1 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

5.5.2 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

5.5.3 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。

5.5.4 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。

5.6 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，WebSocket长连接，直播者页面和观看者页面组成。

5.6.1 .WebSocket长连接

WebSocket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WebSocket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。

5.6.2 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WebSocket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。

5.6.3 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WebSocket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。

6 实现

6.1 本地服务器客户端开发现状

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。

6.2 关于跨平台网络库

其实整个的系统核心部分，其实都是通用的，也就是说，虽然我们的软件要跨多个平台，但是我们可以把核心的、公共的代码部分，封装成一个跨平台的网络库，这样一来就可以大大减少后期代码维护成本。

6.3 为什么选择Go语言

Go语言介绍：

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(并发)

2.跨平台

3.网络库丰富

4.语法简单

6.3.1 性能(并发)

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。

6.3.2 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Goroutine运行所有的一切，当然也包括main.main入口函数。

可以说，Goroutine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！

Goroutine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！

6.3.3 良好的语言设计

这也是我个人非常喜欢Go的很大的一个原因。虽然，从学术的角度来讲，Go语言是十分平庸的，不支持许多高级语言的特性。但是，从工程学的角度来讲，Go语言的设计是十分优秀的！规范足够灵活，有其他语言基础的程序员都可以通过快速学习来上手。更加重要的是Go语言自带的完善的工具链，大大提高了团队协作的一致性。比如gofmt自动对Go语言的代码排版，很大程度上杜绝了不同的人写的代码排版风格不一样的问题。吧编辑器配置成在编辑存档的时候自动运行gofmt，这样在编写代码的时候可以随意摆放位置，当你保存的时候自动变成正确的排版代码。是不是很方便！此外还有gofix,govet等非常有用的工具。

6.4 Go语言的现状与前景

其实我在做出这个选择之前已经花了大量时间做过详尽调研。 国外如Google、AWS、Cloudflare、CoreOS等，国内如七牛、阿里等都已经开始大规模使用Golang开发其云计算相关产品。 跟着世界级巨人的脚步应该不至于走错方向，而且在学习Golang的过程中，我也渐渐被其背后的设计哲学所折服。

另外，云风博客中曾说过这样一句话：

我发现我花了四年时间锤炼自己用C 语言构建系统的能力，试图找到一个规范，可以更好的编写软件。结果发现只是对 Go 的模仿。缺乏语言层面的支持，只能是一个拙劣的模仿。

6.5 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github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。

6.6 Android开发现状

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GoMobile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。

6.7 摄像头直播功能实现

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WebSocket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WebSocket发送给服务器，服务器收到数据之后，则对所有的WebSocket连接进行广播。至于观看者这边，同样，也是用WebSocket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。

6.8 Robotgo自动化库

为了实现键盘和鼠标的控制，我们找到了一个第三方库：https://github.com/go-vgo/robotgo

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的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。

7 外文翻译

英文原文：

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 goroutines and channels, explained below). Concurrency, by its design, enables you to efficiently use your CPU horsepower. Even if your processor just has 1 core, concurrency’s design enables you to use that one core efficiently. That is why you can typically have hundreds of thousands of concurrent goroutines (lightweight threads) running on a single machine. Channels and goroutines 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, CDNs, caches etc), distributed systems, microservices etc. So if you’re an engineer or a startup 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 Griesemer, 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 GitHub. 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 https://talks.golang.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 https://talks.golang.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 multi 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), microservices, 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 ‘scp’ 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 Hoare.

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

There are two benefits of this approach:

i) A Goroutine 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 goroutines 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 goroutines.

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 atleast a main thread) for different timeslices 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 microservices. Even if your application resides on a single server (if you’re just starting out), you architect different functionalities required in your application as different microservices, 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) Gofmt: 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 GitHub and copies it to your GoPath so that you can import the library in your project

iv) Godoc: Godoc parses your Go source code — including comments — and produces its documentation in HTML or plain text format. Through godoc’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 versioning 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goroutines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goroutines（轻量级线程）的原因。渠道和goroutines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 Griesemer，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GitHub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https://talks.golang.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https://talks.golang.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goroutine。 Goroutine是一个轻量级的执行线程。 Go运行时将这些goroutine复用到操作系统线程上。这意味着多个goroutine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goroutines执行。

这种方法有两个好处：

i）初始化时的Goroutine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goroutine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goroutine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）Gofmt：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GitHub下载库并将其复制到GoPath，以便您可以将库导入到项目中

iv）Godoc：Godoc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Godoc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 :)

8 结论

9 致谢

10 参考文献