武汉大学计算机学院
本科生课程实验报告

Fishield 基于 C++的网络云盘程序

专 业 名 称　：软件工程

课 程 名 称　：网络及分布式计算

指 导 教 师 一：胡继承 教授

学 生 学 号　：2016302580055

学 生 姓 名　：了然

二〇一九年五月

郑 重 声 明

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名：了然　　日期：2019.05.31

摘　要

本次试验的主要目的是利用本学期课堂上所学的关于 Web 网络编程相关知识，在实践中运用。实现一个完整的网络云盘程序，包含完整的服务器端编程以及相应的客户端编程。

程序将包含一个常见网络云盘所应具备的所有基本功能，包含上传文件、下载文件等。同时对用户设置优先级，实现更细粒度的权限管理。并且，为了给予用户更好的体验，我实现了更加细节的功能，包括删除文件、重命名文件、暂停上传/下载任务、继续上传/下载任务等、取消上传/下载任务等。允许用户将暂停任务并在之后继续任务极大的扩展了该软件的适用范围，使之可以应用于较大的文件，允许通过断点续传的方式将一个较大的文件分块上传。

实现方面，我利用由 Google 开发的开源工具 Protobuf。Protobuf 是一款跨平台、跨语言、可扩展的，数据结构序列化工具。我定义了本软件的服务器端与客户端信息交互的协议，本质上这是一个我自己定义的应用层网络协议。随后，我通过 Protobuf 将之简洁快速的序列化，并用 TCP 协议在服务器端与客户端间传递信息。在服务器端，有一个线程作为守护线程不断地运行我自己设计的调度器算法，等待客户端发来请求信息并作出相应反馈。

关键词：C++；网络云盘；服务器端编程；客户端编程；Socket 编程；Protobuf；网络应用层协议;Boost C++ Library; QML; Material Design; 异步 IO

目　录

1　实验目的和意义

1.1　实验目的

本次试验的主要目的是利用本学期课堂上所学的关于 Web 网络编程相关知识，在实践中运用 Socket 编程。实现一个完整的网络云盘程序，包含完整的服务器端程序以及相应的客户端程序。

1.2　实验意义

学生通过亲自设计专属的应用层网络协议，深入理解了网络分层结构的服务模型。同时通过切实的 Socket 编程联系，深入的掌握了 TCP/UDP 的常见特性以及常见使用方式。与此同时，通过服务器端的调度器算法以及数据库的相关代码，学生较好的讲所学网络方向知识与操作系统、数据框方向相关知识有机的结合在了一起，并通过编写代码极大地加深了理解。Protobuf 的使用则使学生与世界领先的开源社区产生了联系，了解到了 State-of-the-art、cutting-edges 的现状。

2 需求分析

2.1 上传文件

用户可以通过客户端程序将本地文件上传、存储到云端服务器。

2.2 下载文件

用户可以通过客户端程序预览到已经保存在服务器端的所有文件，并选择相应文件，将其从云端服务器下载到本地。

2.3 暂停上传/下载任务

当一个上传/下载任务被创建之后，用户可以选择暂停该任务并安全的退出程序。

2.4 取消上传/下载任务

当一个上传/下载任务被创建之后，在该任务完成之前，用户可以选择取消该任务，永久性的终止该上传/下载的过程，并清除一切产生的中间文件。

2.5 删除文件

用户可以通过客户端程序预览到已经保存在服务器端的所有文件，并选择删除其中的部分文件。

2.6 重命名文件

用户可以通过客户端程序预览到已经保存在服务器端的所有文件，并选择修改其中部分文件的名字。修改文件存储路径也被看做是重命名文件的一种特例情况。

2.7 创建文件夹

用户可以通过客户端程序预览到已经保存在服务器端的所有文件，并可以选择在适当位置创建新的文件夹。

2.8 权限控制

我将上传与下载的权限分离，允许用户只拥有上传文件的权限，允许用户只拥有下载文件的权限，允许用户拥有上传/下载的完整权限。

2.9 客户端 IP 限制

更加细致的安全控制是可行的，服务器端可以检查客户端的 IP 地址，可以通过适当配置，只允许部分合法 IP 地址访问（白名单机制），或者禁止部分非法 IP 地址访问（黑名单机制）。

3 应用层协议设计

3.1 概述

在 Fishield 项目中，客户端生成恰当的 Fishield_Request，序列化之后发送给服务器端，服务器端在接收到请求后首先进行反序列化操作，还原成原本的 Fishield_Request 数据结构。然后进行相应处理，并将处理结果写入一个新创建的 Fishield_Response 数据结构中，经序列化后发送给客户端作为相应。

这里提到的 Fishield_Request 和 Fishield_Response 就是 Fishield 项目的应用层协议，它们是我定义好的两种数据结构，其中包含了大量的控制信息。它们的内在结构通过 Protobuf 的 proto2 语言可以及其简单的定义出来。然后 Protobuf 可以通过 proto2 语言所定义的结构生成相应的 C++类，客户端/服务器端代码调用该 C++类。Protobuf 在其中起到了一个间接层的作用，其优势是简洁。用 proto2 语言定义结构比用 C++语言直接定义要简单得多，并且 Protobuf 会自动帮我们生成相应的序列化/反序列化方法，可以极大地简化随后的 C++代码。

3.2 文件数据结构

文件数据结构包含文件名、文件最后修改时间、文件类型、文件大小、子文件列表五个字段。

与 Unix 系统定义保持一致，文件有普通文件、文件夹、符号链接和其他文件四种类型。文件大小只有在文件类型是普通文件时有意义，子文件列表只有在文件类型是文件夹时有意义。

```
message File
{
        enum FileType
        {
                REGULAR                         = 0;
                DIRECTORY                       = 1;
                SYMLINK                         = 2;
                OTHER                           = 3;
        }
        required string filename        = 1;
        required uint64 mtime           = 2;    // time of last data modification
        required FileType file_type     = 3;
        optional uint64 size            = 4;    // only valid for REGULAR file
        optional FileList filelist      = 5;    // only valid for DIRECTORY file
}
```

图 3.1 文件数据结构定义

3.3 Fishield_Request 数据结构

Request 数据结构的核心是 RequestType，即请求类型。其他字段均为可选的补充参数，都是只有针对某个特殊的请求类型，某几个字段才是有效的。

- 登录请求（LOGIN）

  在该类型下，用户名（username）和密码（password）字段是有效的。客户端程序将用户输入的用户名信息封装到 Fishield_Request 数据结构中，并发送给服务器。服务器查询数据库相关数据，进行校验并返回校验结果。

- 请求文件列表（FILELIST）

  在该类型下，服务器路径（remote_path）字段有效。经校验路径有效后，服务器端程序将服务器路径所代表的位置下所有子文件列表返还给客户端。该请求主要用于向用户展示云端服务器所存储的内容列表。

- 创建文件夹（MKDIR）

  在该类型下，服务器路径（remote_path）字段有效。经校验路径有效后，服务器端程序在服务器路径所指定的位置创建一个空的文件夹。

- 上传文件请求（UPLOAD）

  在该类型下，一个新的上传任务会被创建。此时文件名（filename）和包裹数量（packet_no）字段有效。为了实现有效的断点续传功能，让用户可以选择将任务暂停、在一段时间后重启。我将每个文件分成若干定长的包裹（Packet），并逐一发送。这样通过跟踪发送的包裹编号、已发送的包裹数量以及包裹总数，就可以在应用层实现可暂停的可靠传输。在用户点击暂停某个任务的时候，客户端程序会在发送完当前正在发送的包裹后停止发送。

- 下载文件请求（DOWNLOAD）

  在该类型下，一个新的下载任务会被创建。此时服务器路径（remote_path）和文件名（filename）字段有效。此后下载任务开展的时候，文件同样会被切分成一个个定长的包裹（Packet）逐一从服务器端发送。

- 发送包裹（SEND_PACKET）

  在该类型下，客户端向服务器端发送一个上传任务的包裹（Packet）。此时，任务编号（Task_id）和包裹（Packet）字段有效。任务编号是用于

区分多个不同的任务的，每个上传/下载任务都有一个唯一的任务编号。包裹（Packet）则包含实际发送的数据。

- 接收包裹（RECEIVE_PACKET）

  在该类型下，客户端通知服务器端重启一个已经暂停的下载任务，此时任务编号（Task_id）有效。任务编号用于指定需要重启的任务。

- 删除文件（REMOVE）

  该类型用于删除云端服务器上存储的某个文件。此时服务器路径（remote_path）和文件名（filename）字段有效。

- 重命名文件（RENAME）

  该类型用于将云端服务器上的某个文件重命名，这里的重命名是广泛意义上的重命名，包含文件的位置移动。此时服务器路径（remote_path）、文件名（filename）和新文件路径（new_path）字段有效。新文件路径用于指定修改后的文件路径。

- 取消任务（CANCEL）

  该类型用于取消某个现存的、未完成的上传/下载任务。此时任务编号

```
message Request
{
        enum RequestType
        {
                LOGIN                           = 0;    // try to login
                FILELIST                        = 1;    // request for a filelist
                MKDIR                           = 2;    // create a new directory
                UPLOAD                          = 3;    // initiate a new UPLOAD task
                DOWNLOAD                        = 4;    // initiate a new DOWNLOAD task
                SEND_PACKET                     = 5;    // send a packet
                RECEIVE_PACKET                  = 6;
                DOWNLOAD_CONFIRM                = 7;    // confirm that a download task completes
                REMOVE                          = 8;    // delete a file
                RENAME                          = 9;    // rename or move a existing file
                CANCEL                          = 10;   // cancel a task
                DISKSPACE                       = 11;   // get disk space information
                USERLIST                        = 12;   // get all user information
                ADDUSER                         = 13;   // add a new user
                REMOVEUSER                      = 14;   // remove a user
                IPLIST                          = 15;   // get ip list
                ADDIP                           = 16;   // add a new ip address
                REMOVEIP                        = 17;
        }
        required RequestType req_type   = 1;
        optional string username        = 2;
        optional string password        = 3;
        optional string token           = 4;
        optional string remote_path     = 5;
        optional string filename        = 6;
        optional uint64 packet_no       = 7;    // number of packets that a UPLOAD task will send
        optional Packet packet          = 8;
        optional uint64 task_id         = 9;    // task_id of which CANCEL/PAUSE/RESUME/PACKET
        optional uint64 packet_id       = 10;
        optional string new_path        = 11;   // new path of a RENAMEd file
        optional User user              = 12;
        optional string ipaddress       = 13;

}
```

图 3.2 Fishield_Request 结构定义

（Task_id）有效，用于指定需要取消的任务。

- 获取磁盘容量信息（DISKSPACE）

  该类型用于获取云端服务器的存储空间信息，主要包括磁盘总容量、已使用容量以及未使用容量大小。

3.4 Fishield_Response 数据结构

Response 数据结构的核心是 ResponseType，即响应类型。其他字段均为可选的补充参数，都是只有针对某个特殊的响应类型，某几个字段才是有效的。

- 成功（SUCCESS）

  该类型代表请求合法且处理成功。是最常见最普通的类型。

- 用户不存在（NOSUCHUSER）

  使用不存在的用户名登录会出发此错误。

- 非法路径（ILLEGALPASSWD）

  下载文件/新建文件夹/重命名文件/删除文件时如果给出的路径非法会出发此错误。

- 未知错误（UNKNOWN）

- 无响应错误（NORESPONSE）

  无法链接到服务器或链接超时会触发此错误。

- 非法请求（ILLEGALREQUEST）

  出现未定义的请求会出发此错误

- 无权限错误（NOPRIVILEGE）

  Fishield 项目可以对用户采取更加细粒度的权限划分，如果某用户没有上传权限并尝试上传文件，或没有下载文件却尝试下载文件会触发此错误。

4　代码架构设计

4.1　概述

整体代码一共可以大题分为数据库交互代码（DB_Manager）、服务器端代码（Server）、调度器代码(Scheduler)、任务传输代码(Task)和客户端代码(Client)五部分。

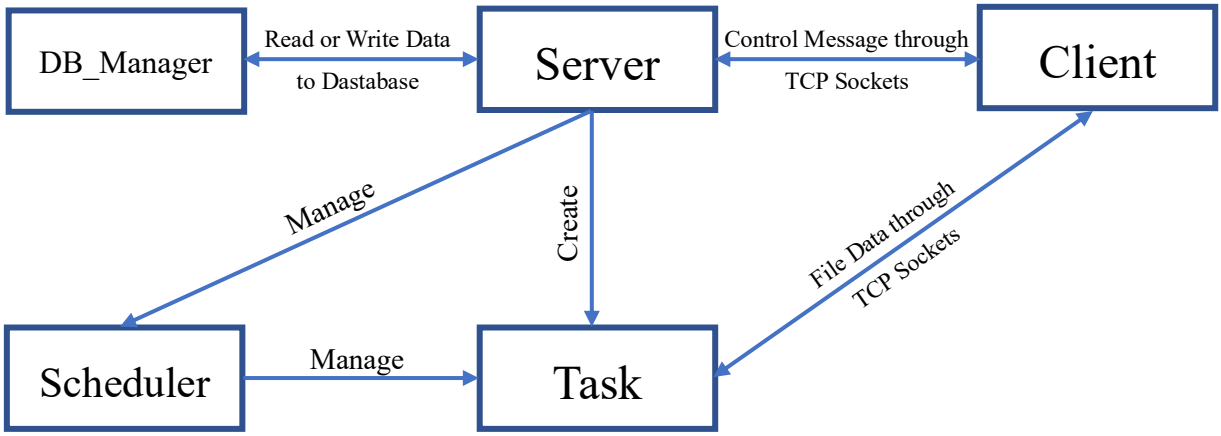


图 4.1 代码结构图

4.2　数据库交互代码模块（DB_Manager）

所有与数据库相关的交互代码全部通过数据库交互代码模块执行。其他模块不直接与数据库产生交互。Fishield 使用的是 mysql 数据库，数据库中主要存储了用户密码信息以及各项上传/下载任务的完成进度。

4.3　服务器代码模块（Server）

服务器代码模块是整个项目的核心，这个模块与其他所有模块都有交互。

服务器模块与客户端模块通过 TCP Socket 传递上一节提及的 Fishield_Request 以及 Fishield_Response 数据。这部分信息主要是控制信息，起到发起/终止/修改任务的作用。

在接到客户端发来的新建上传/下载任务的请求后，服务器端代码模块会创建一个任务对象（Task），并将这个对象交给调度器模块（Scheduler）统一调度管理。并将相应数据通过数据库交互模块（DB_Manager）写入持久化的 Mysql 数据库。

在调度器模块（Scheduler）的调度下，任务（Task）对象会与客户端建立起 TCP Socket 通信渠道，并在该 Socket 中发送/接收所指定的文件。

4.4 调度器模块（Scheduler）

调度器模块拥有一个任务池（Task Pool），其中记录了所有当前处于活跃状态的任务。调度器模块会分离出一个线程通过轮询（Polling）的方式检查每一个任务，并根据每个任务各自不同的状态作出调整、更新调度数据。

4.5 任务模块（Task）

任务模块负责真正完成上传/下载文件的过程。任务模块负责与客户端程序建立 TCP 通信管道，并传输文件。传输文件的时候将文件切分成若干定长的包裹（Packet），以包裹为最小单位依次传递。并在每次传递完一个包裹的时候更新任务状态，检查包裹是否正确传递，实现应用层的可靠数据传输以及断点续传功能。

```
enum TaskStatus{
        UPLOAD_INIT                     = 0;
        UPLOADING                       = 1;
        UPLOADED                        = 2;
        UPLOAD_PAUSED                   = 3;
        UPLOAD_PAUSING                  = 4;
        UPLOAD_RESUME                   = 5;

        DOWNLOAD_INIT                   = 6;
        DOWNLOADING                     = 7;
        DOWNLOADED                      = 8;
        DOWNLOAD_PAUSED                 = 9;
        DOWNLOAD_PAUSING                = 10;
        DOWNLOAD_RESUME                 = 11;

        CANCELING                       = 12;
        CANCELED                        = 13;

        FAILED                          = 14;
        FAILING                         = 15;

}
```

图 4.2 任务状态

在实现中，每个任务从开始到完成，一共被分为了 16 个可能出现的状态，如图 4.2 所示。其中包括开始上传（UPLOAD_INIT）、上传中（UPLOADING）、已上传

（UPLOADED）、上传已暂停（UPLOAD_PAUSED）、上传暂停中（UPLOAD_PAUSING）、上传继续等状态(UPLOAD_RESUME)、开始下载(DOWNLOAD_INIT)、下载中(DOWNLOADING)、已下载（DOWNLOADED）、下载已暂停（DOWNLOAD_PAUSED）、下载暂停中（DOWNLOAD_PAUSING)、下载继续等状态(DOWNLOAD_RESUME)、取消中(CANCELING)、已取消（CANCELED)、失败中（FAILING)、已失败（FAILED）等。

4.6 客户端模块（Client）

客户端模块一般负责向服务器端发起任务，并协助任务模块代码实现应用层的可靠数据传输。

5 客户端 UI 界面设计

5.1 概述

客户端 UI 界面采用 QML 语言进行设计，并应用了 Google 提出的 Material Design，实现了简约大气的界面风格。

QML 是一种及其简单的标记语言，与 HTML/XML 等标记语言类似，QML 主要运用层次化的语句表示结构，并通过 QML 引擎渲染出最终结果。QML 的后端响应逻辑主要通过 QMake 的 Signal/Slot 语言特性实现。

Signal/Slot 特性并不是 C++标准特性。这是由 QMake 独立定义的特性，主要实现了不同组件间的异步信息传递功能。在编译的时候，QMake 会首先进行一次预编译，将带有 Signal/Slot 特性的非标准 C++代码预编译成标准 C++代码。随后调用标准 C++编译器，如 MinGW，最终编译成可执行程序。

5.2 Material Design

Material design 的核心思想，就是把物理世界的体验带进屏幕。去掉现实中的杂质和随机性，保留其最原始纯净的形态、空间关系、变化与过渡，配合虚拟世界的灵活特性，还原最贴近真实的体验，达到简洁与直观的效果。

Material Design 不能简单地归纳为平面化设计（Flat Design）。实际上，Android 4.0 的设计风格，也不是纯粹的平面化设计，在经过仔细观察之下，我们可以看到 Android 4.0 在细节上并没有反对高光、阴影、纹理，换言之它并不反对立体感。不过，它也不能归类为拟物化设计，毕竟它所使用的图案、形状并非是对现实实体的模拟，而是按照自己对数字世界的理解，以色彩、图案、形状进行视觉信息上的划分。

根据 Hi-iD 之前在《花样,形式,风格,氛围,主义……设计》中总结设计当中不同的层次，以 Google 过去各个产品而言，都充满了不同的花样和形式，但无法统一为一种风格。而 Material Design 则结合卡片式设计，又结合现实世界里纸张的隐喻，统一了 Google 在设计上的表达，从而展示出一种强烈的风格。这种风格不会因为我们使用 Android Wear 或 Android TV 等不同的设备，而感到不同。Hi-iD 说得好，“风格既是自我表达也是一种记名和品牌”。

Material Design 在设计上并没有完全抛弃 Google 过去在设计上取得的成果。Material Design 和 Google 的标志一样，崇尚多彩，但它并不使用那种很艳丽的颜色，反而似乎是为了让图案变得沉稳，而有意令原本很晃眼的色彩混入一点点的灰色，让图案变得活泼之余，又不会因为过于艳丽而让人感觉到俗套和嬉皮。它也没有抛弃阴影，仔细观察 Android 4.0 的下拉菜单，我们可以看到底部和右侧有着淡淡的阴影。

为了统一跨设备间的界面和交互，让用户得到连贯的体验。Material Design 不再让像素处于同一个平面，而是让它们按照规则处于空间当中，具备不同的维度。按照 Wired 的话来说，那就是让像素具备海拔高度，这样子的话，系统的不同层面的元素，都是有原则、可预测的，不让用户感到无所适从，也避免开发者担心因为不同的视觉风格而产生冲突。

Material Design 还规范了 Android 的运动元素，让按钮的弹入弹出，卡片的滑入滑出以及从一个界面变化成另一个界面的方法(比如从介绍一首歌的界面到控制播放的界面)，都是秩序的、深思熟虑过的。Wired 总结，Material Design 中只有在高亮动作以及改变交互状态时，才会使用运动元素来表示。

对于现实世界中的隐喻，Material Design 更加倾向于用色彩来提示。我们按下屏幕当中的按钮时，可以看到按钮颜色迅速发生变化，向石头投入湖面一样，产生了一波涟漪。杜瓦迪这样设计是因为 Material Design 中的按钮都处于一个平面，不再突起，因此它必须采用和以往不同的表示方法，以表明自己已经被按下。

Material Design 所展示的模板当中，最显眼的是它的小圆点。它的作用好像 iPhone 上的 Home 键，是快捷功能入口，又是视觉上有趣的点缀。

然而，正如 Material Design 名字所暗示的，为了适应多尺寸的屏幕，杜瓦迪以及他的团队，寻求一种更加抽象的表达，一种存在屏幕里的显示“材料”。根据 The Verge 报道，杜瓦迪团队在面对 Google 产品里大量采用的卡片式设计时，灵感火花一闪：何不如这些“卡片”，想像成现实当中存在的，四处滑动的物体。

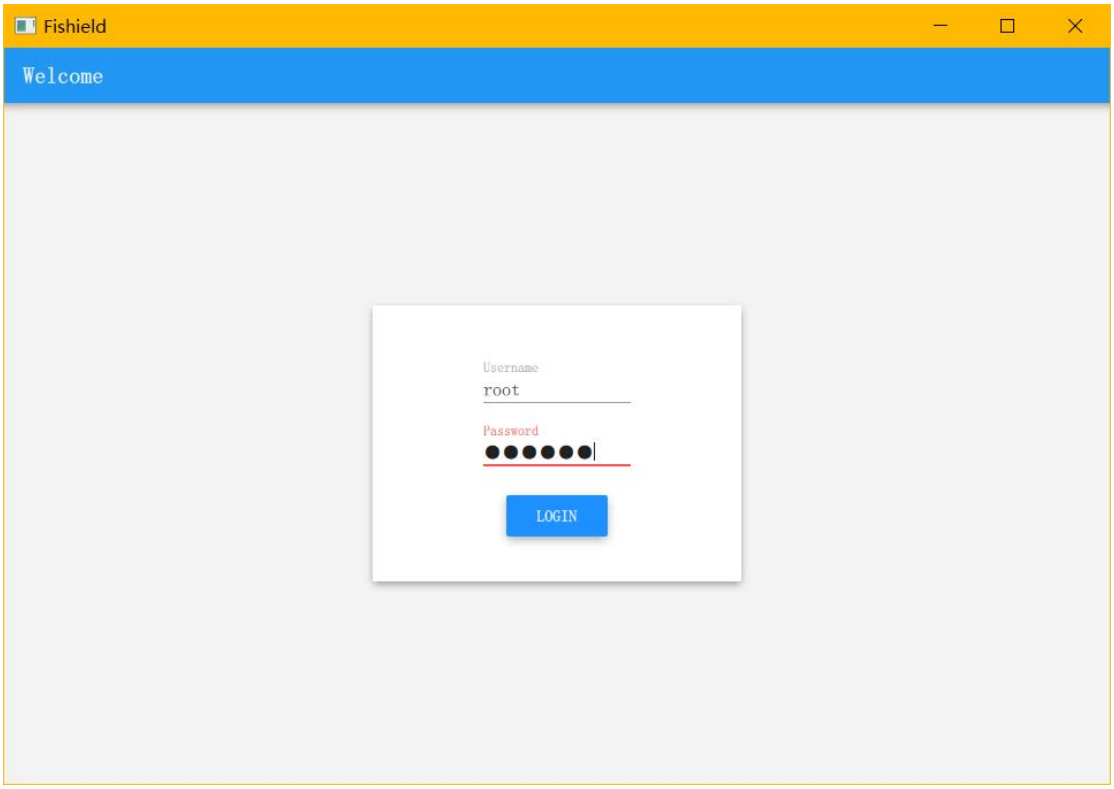
如果这些卡片遵循物理世界里的法则，那么它就有自己的规矩，不见得每个人都能够任意使用，对于设计师来说“限制”是有必要的。在 Google 设计师的

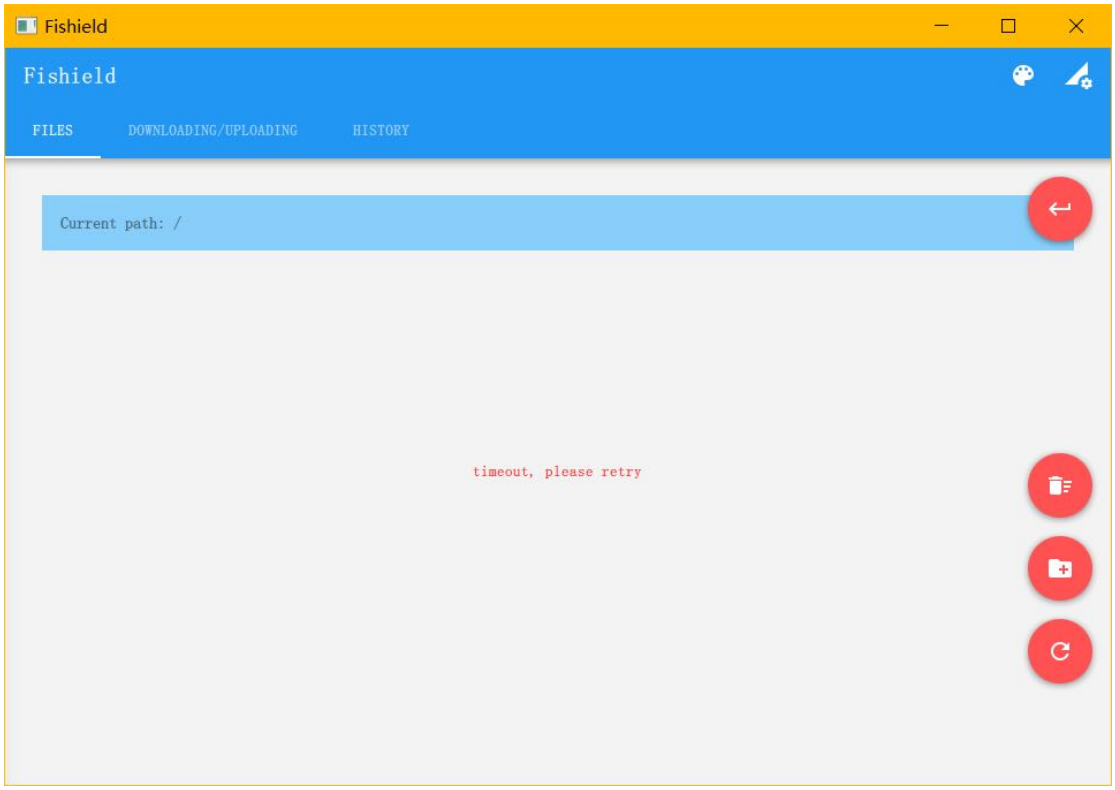
想象中，这种抽象的“材料”特性很像纸张，但它又做到现实当中纸张做不到的事情，比如变大变小。这赋予这种“材料”极大的灵活性，让它足以适应不同尺寸的屏幕。然而，对于 Google 来说，Material Design 还将扩展到 Google 其它产品当中去，让所有产品都烙印上浓浓的 Google 风格。

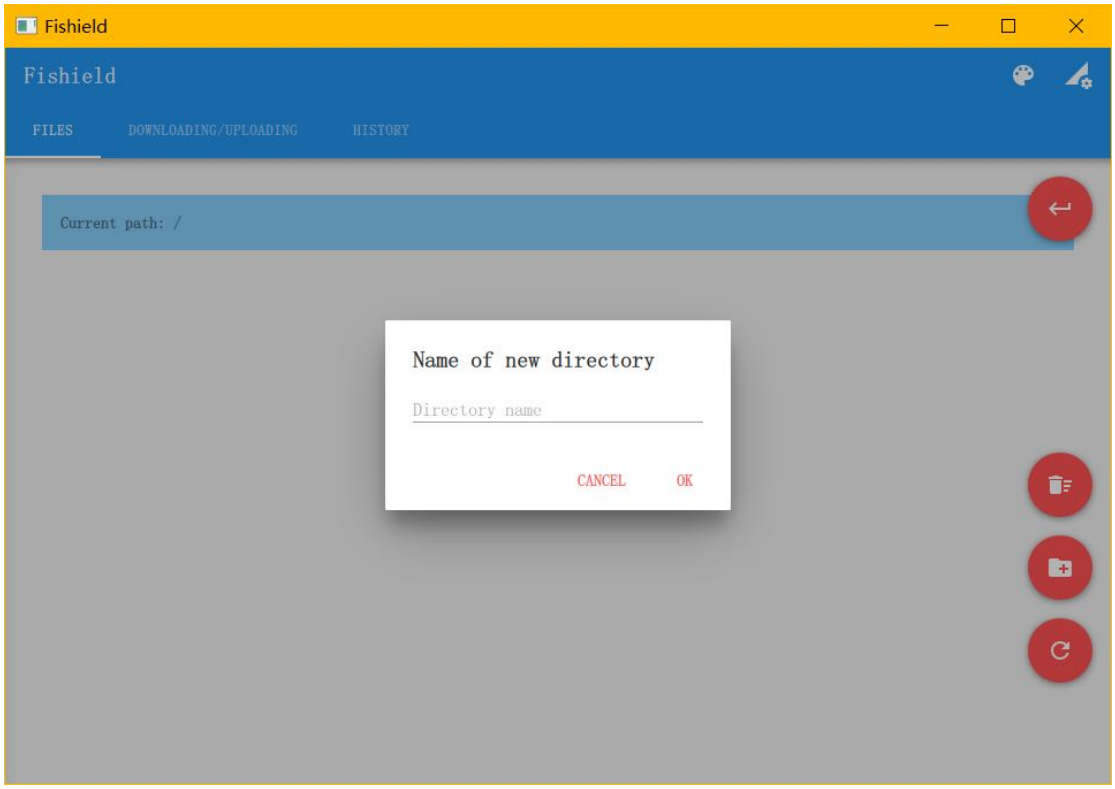
对于杜瓦迪来说，从 Android 4.0 到 Material Design，体现了他身为设计师的素质和能力，而在这不同的经历当中，他的感受是，“打造 Ice Cream 时，我好似军舰上的一名上校，被叫去轮机舱，而这艘船即将在 30 分钟后转弯。创造 Material Design 就像指挥太平洋舰队的海军上将。我们要在一无所知的情况下作出决定，而船在 48 小时后启航。”

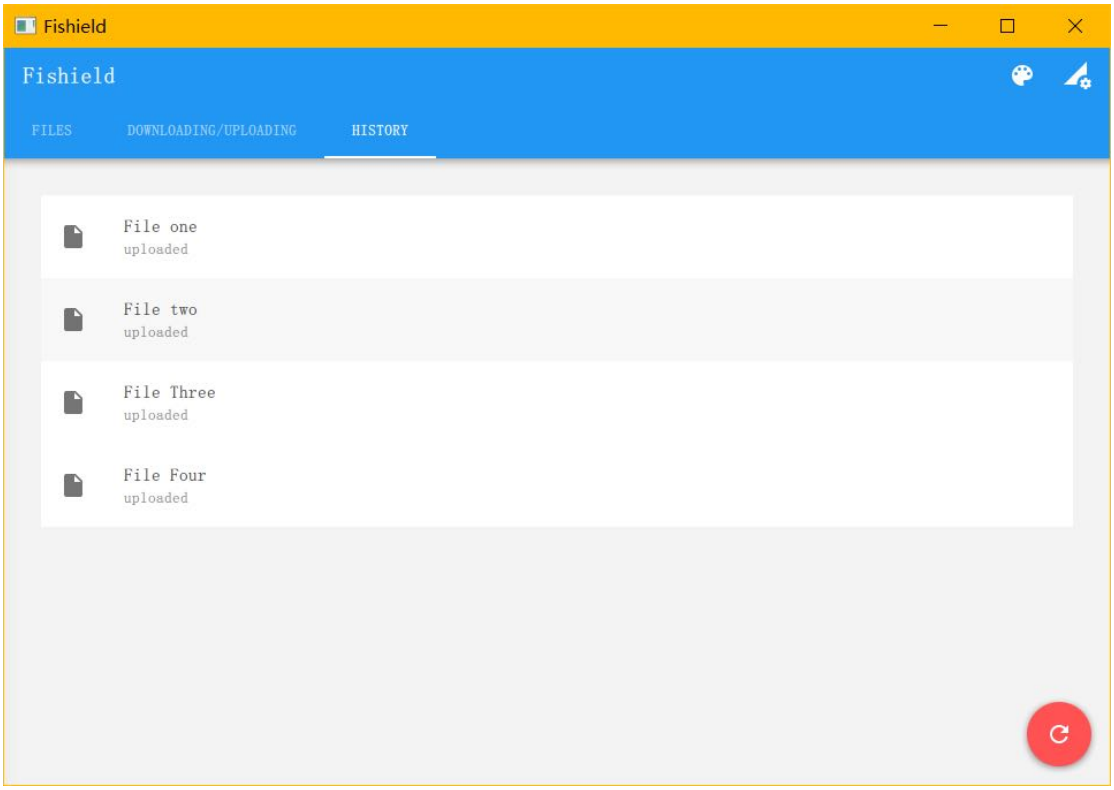
至于 Material Design 本身，它是一个足以媲美苹果设计的一套设计框架。

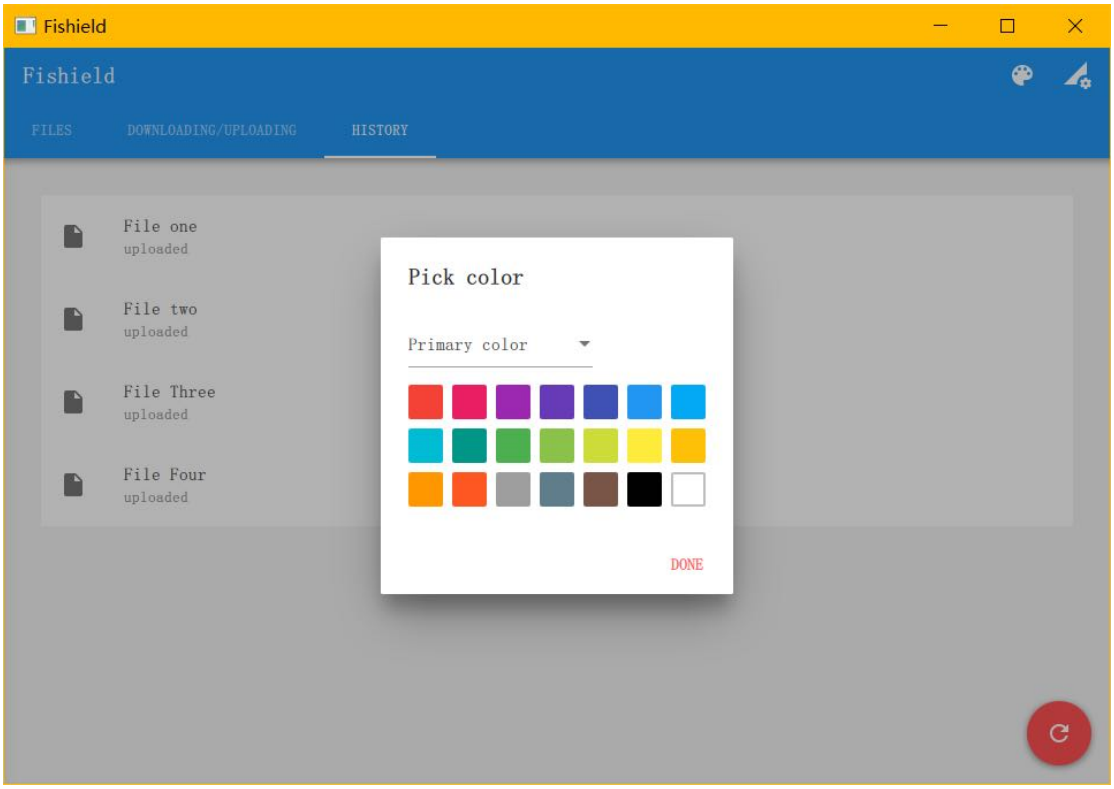
5.3 效果展示

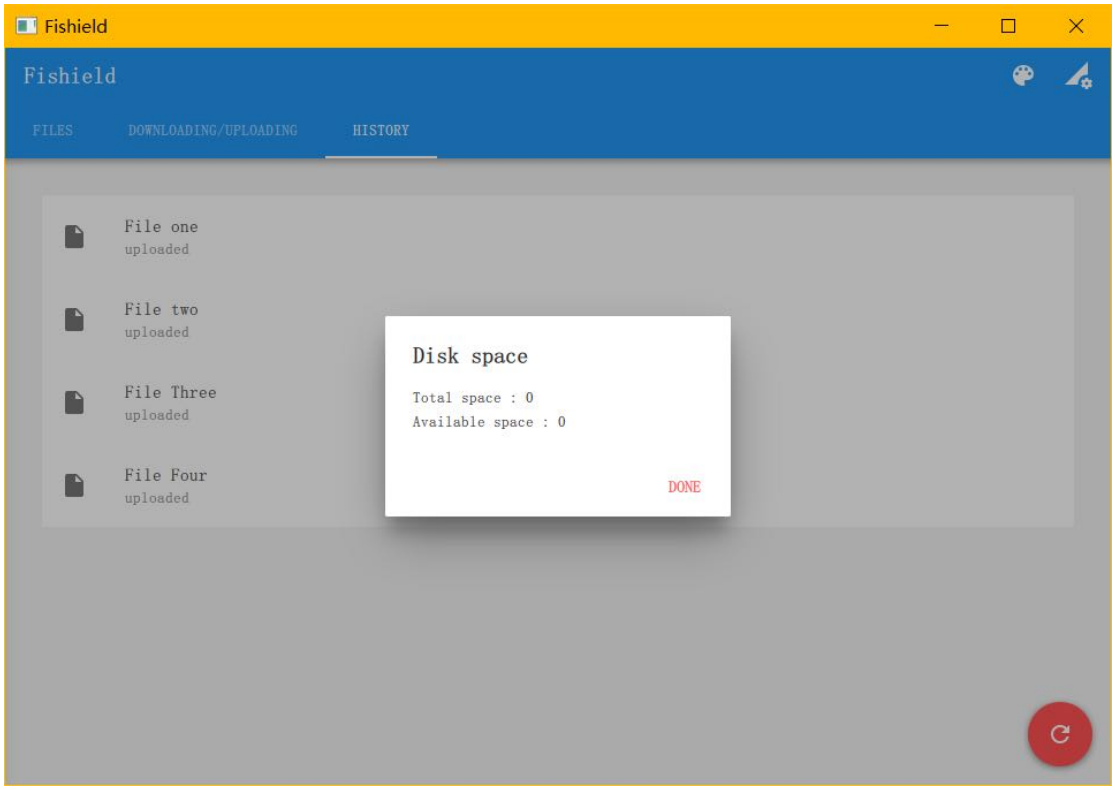



Fishield
Fishield
FILES
DOWNLOADING/UPLOADING
HISTORY
Current path: /
timeout, please retry


Fishield
Fishield
FILES
DOWNLOADING/UPLOADING
HISTORY
Current path: /
Name of new directory
Directory name
CANCEL
OK


Fishield
Fishield
FILES
DOWNLOADING/UPLOADING
HISTORY
File one
uploaded
File two
uploaded
File Three
uploaded
File Four
uploaded


Fishield
Fishield
FILES
DOWNLOADING/UPLOADING
HISTORY
File one
uploaded
File two
uploaded
File Three
uploaded
File Four
uploaded
Pick color
Primary color
DONE


Fishield
Fishield
FILES
DOWNLOADING/UPLOADING
HISTORY
File one
uploaded
File two
uploaded
File Three
uploaded
File Four
uploaded
Disk space
Total space : 0
Available space : 0
DONE

参考文献

[1] Material Design

[2] Material Design Library for QML

[3] Protobuf

[4] Boost C++ Library

【结论】:

通过合理设计应用层网络协议，开发者可以设计并实现出拥有较高性能的定制产品。通过应用层的高层面可靠数据传输协议，我们可以实现更加细致的传输控制，并对程序拥有更多的控制，实现断线续传等非常使用的功能。

同时 Material Design 能与 QML 开发通过 github 开源库良好的结合在一起。Protobuf 的使用可以极大地简化应用层协议的编程实现。

【小结】:

学生通过亲自设计专属的应用层网络协议，深入理解了网络分层结构的服务模型。同时通过切实的 Socket 编程联系，深入的掌握了 TCP/UDP 的常见特性以及常见使用方式。与此同时，通过服务器端的调度器算法以及数据库的相关代码，学生较好的讲所学网络方向知识与操作系统、数据框方向相关知识有机的结合在了一起，并通过编写代码极大地加深了理解。Protobuf 的使用则使学生与世界领先的开源社区产生了联系，了解到了 State-of-the-art、cutting-edges 的现状。

指导老师评语及成绩

【评语】:

成　　绩：　　指导老师签名：

批阅日期：