我的云架构设计

目录

[我的云架构设计 1](#_Toc340841463)

[1 前言 2](#_Toc340841464)

[2 术语 2](#_Toc340841465)

[3 需求描述 3](#_Toc340841466)

[3.1 设备连接管理 3](#_Toc340841467)

[3.2 文件管理 3](#_Toc340841468)

[3.3 互动分享 3](#_Toc340841469)

[4 需求及技术分析 4](#_Toc340841470)

[4.1.1 设备连接管理 4](#_Toc340841471)

[4.1.2 文件管理 5](#_Toc340841472)

[4.1.3 互动分享 7](#_Toc340841473)

[5 总体框架 7](#_Toc340841474)

[5.1 逻辑结构 7](#_Toc340841475)

[5.1.1 应用层 8](#_Toc340841476)

[5.1.2 服务层 8](#_Toc340841477)

[5.1.3 协议层 8](#_Toc340841478)

[5.2 物理结构 9](#_Toc340841479)

[5.2.1 非协议相关组件 9](#_Toc340841480)

[5.2.2 协议相关组件 9](#_Toc340841481)

[6 分模块设计 10](#_Toc340841482)

[6.1 ZFS子系统 10](#_Toc340841483)

[6.1.1 ZFS抽象层 10](#_Toc340841484)

[6.1.2 本地文件系统上的ZFS 11](#_Toc340841485)

[6.1.3 FTP上的ZFS 12](#_Toc340841486)

[6.1.4 DLNA上的ZFS 12](#_Toc340841487)

[6.1.5 公有云上的ZFS 13](#_Toc340841488)

[6.1.6 实现过程中的一些思考 13](#_Toc340841489)

[6.2 设备管理子系统 14](#_Toc340841490)

[6.2.1 私有云设备 14](#_Toc340841491)

[6.2.2 设备扫描及通知 16](#_Toc340841492)

[6.2.3 设备管理器 16](#_Toc340841493)

[6.2.4 组件部署 17](#_Toc340841494)

[6.3 分类浏览及搜索子系统 18](#_Toc340841495)

[6.3.1 搜索 18](#_Toc340841496)

[6.3.2 分类浏览 19](#_Toc340841497)

[6.3.3 索引数据库 19](#_Toc340841498)

[6.3.4 索引交换协议 20](#_Toc340841499)

[6.3.5 组件结构 21](#_Toc340841500)

[6.3.6 对缩略图的支持 21](#_Toc340841501)

[6.4 互动分享子系统 22](#_Toc340841502)

[7 分设备设计 22](#_Toc340841503)

[7.1 WIFI硬盘 22](#_Toc340841504)

[7.2 Android 23](#_Toc340841505)

[7.3 PC 23](#_Toc340841506)

[8 总结 23](#_Toc340841507)

[8.1 整体部署图 24](#_Toc340841508)

[8.2 WIFI硬盘组件图 24](#_Toc340841509)

[8.3 Z5手机组件图 26](#_Toc340841510)

[8.4 PC组件图 27](#_Toc340841511)

# 前言

我的云是公司手机特色应用的主要应用系统之一，目标在于实现同一个注册用户的不同设备之间的文件、信息共享，最大限度提升用户跨设备操作的便捷性，同时又保证一定的安全性。目前规划纳入此系统管理的设备包括：PC（Windows）、Z5手机、WIFI 硬盘、云盘、DLNA设备、WFD设备；用户能够完成的操作包括：文件管理、分享传送、分类检索等。本文档描述该系统的整体结构及主要技术方案。

# 术语

* 我的云：表示纳入管理的所有设备形成的用户逻辑概念，同时也指管理这些设备的软件系统。
* 私有云：同一个用户拥有的只能在局域网内授权访问的设备形成的网络，具体包括Z5手机、安装有我的云应用程序的PC、按需求定制过的WIFI移动硬盘等设备。
* 公有云：指通过internet连接的授权访问的设备，目前主要指云盘。
* 设备发现：一种不需要用户干预，由软件通过某种协议自动完成设备的查找、连接的设备管理方法。
* DLNA：Digital Living Network Alliance（数字生活网络联盟 ），是一种解决方案和标准，用来解决个人PC，消费电器，移动设备在内的无线网络和有线网络的互联互通，使得数字媒体和内容服务的无限制的共享和增长成为可能。
* WIFI Direct：一种两台或多台支持WIFI的设备间不需要路由器直接互连的技术。
* WFD：WIFI Display，一种WIFI Direct的应用标准，用来将一个设备的音视频内容同步在另一设备上显示播出，实现内容共享。
* ZFS：ZTEMT File System,一个抽象的文件操作接口层，用来屏蔽不同存储设备、协议的差异，对上层提供一个一致的编程接口，简化上层应用的工作。

# 需求描述

经过反复讨论及技术调研分析，目前确定的需求包括如下几个方面。

## 设备连接管理

为了极大简化用户的操作，屏蔽不同设备的技术特性，需要有一种机制能够完成用户设备的自动化连接管理。用户只需要输入最少量的必需信息，完成身份认证、鉴权等操作，其他的管理都是自动的或智能导航的。

## 文件管理

* 需要模糊掉设备边界，给用户提供一种透明的大的文件系统逻辑视图，用户用类似本地文件系统操作的方式来操作所有设备上的文件。目前需要能够管理的设备包括：Z5手机、安装有我的云应用软件的PC（目前只需要支持Windows系统）、支持FTP协议的WIFI移动硬盘、云盘、提供文件浏览/下载/上传能力的DLNA设备。
* 弱化原始文件系统的树形目录存储结构，为用户提供一个以内容为中心的分类视图，提供智能搜索。具体分类包括：图片、视频、音乐、文档、应用安装包、压缩包、以及用户自定义的标签分类。
* 为手机的Android平台上的其他应用提供一个扩展的文件系统接口，使得可以透明访问所有设备上的文件，尤其是图片及多媒体资料。

## 互动分享

用户能够将当前查看的内容通过DLNA、WFD等技术，将部分内容和其他设备共享，获得分享乐趣。具体包括：将当前的多媒体文件在其他设备上播放、播放其他设备上的多媒体文件、将当前屏幕内容投影到其他设备上等。

# 需求及技术分析

### 设备连接管理

上图是我们系统中的设备连接拓扑图，连线上的标签显示了在不同节点间的通信协议。我的云应用程序主要部署在Z5手机和PC上，WIFI硬盘上也有一部分组件运行，通过各种协议和其他设备连接。

其中Z5、PC、WIFI硬盘、云盘组成了存储系统，是用来完成文件管理的设备。同时在支持DLNA协议的Z5手机、PC间还可以完成互动分享功能。其他设备则用来完成互动分享功能。

根据设备连接需求，自动化检测是一个重点目标。除BT设备外，其他设备都是通过TPC/IP网络连接的，并且除了公有云外，其他的设备一般都是通过DHCP服务动态获取IP地址接入局域网的，因此地址并不固定，用户无法确定当前设备的地址就无法连接。因此自动发现技术来帮助用户完成连接过程是必要的。

上图中只有DLNA是能够完成自动发现的协议，其他的设备连接需要我们采用其他辅助手段来完成。在存储系统包含的Z5、PC、WIFI硬盘、云盘四种设备中，Z5和PC两种设备上我们会部署我的云应用，因此我们可以通过自定义协议来完成设备发现工作；WIFI硬盘将采用合作伙伴的产品，可以通过定制WIFI硬盘的固件，将我们的定义设备发现协议加入，也可以实现自动发现；公有云由于不能够实现定制，目前不考虑自动发现；但公有云是永久在线并且地址固定，所以通过预先配置来直接加入，也可以达到用户免配置的体验。

### 文件管理

下图是当手机中前两种典型的对本地文件系统的访问架构：

一种是以MediaPlayer为代表的：通过C/C++实现的Framework层来访问Linux文件系统

一种是以FileManager为代表的：通过java语言的Java IO来访问Linux文件系统

通过上面的图形可以发现，这两种之间的公共层在Linux文件系统，如果需要扩展到对远程文件系统的访问，必须将扩展放到Linux文件系统层，即将远程系统mount到Linux文件系统的某个节点上。

#### 理想方案

如下图：

添加进来的远程系统需要实现kernel文件系统，然后加入到Linux的文件系统树中，上层的访问就完全一致了。这个方案需要开发Linux 文件系统驱动程序，基于文件系统的复杂性评估，技术难度上比较大。另外由于该方案在Linux内核层实现，对可靠性有非常高的要求，一旦加入的模块出现故障，将可能导致整个系统都不可用，短期内不太可能达到这样的可靠性要求，风险极高。

#### 可行方案

既然在内核层实现文件系统的方案不可行，我们可以采用在用户态引入一个和VFS类似的抽象层来隔离不同存储系统差异，使得对上层应用呈现一个一致的接口，也能够达到我们的目的。如上图，在Linux本地文件系统之上我们可以构建一个称为ZFS的抽象层来提供所有文件操作，上层应用将原来对Linux本地文件系统的访问依赖切换到对ZFS层的依赖。ZFS层之下，由不同的模块来完成从Linux本地文件系统、FTP、DLNA、公有云等存储传输协议到ZFS的适配。

该方案同样也适用于PC侧的设计，只是PC侧少了其他应用，ZFS的客户端只有我的云管理器一个应用而已。

该方案的优势在于：其实现是在用户态的，有大量现成的资源可供参考，可以快速实现，且对系统的稳定性没有影响，质量要求不会太苛刻。

缺点是：每个需要使用网络文件的应用都需要修改自己的文件访问模块，我们的应用的修改工作量较大，对第三方应用程序无法直接提供支持。目前我们规划的需要使用网络文件的应用不多，因此工作量方面不会有太大的问题；而对第三方应用程序形成障碍未必完全是缺点，并且如果我们对文件访问模块的修改是在Framework层（如Media Service）进行的，则有可能能够直接对某些使用这些Framework组件的第三方应用程序提供支持，上图左上角也展示了这种情形。

### 互动分享

互动分享主要完成的是DLNA共享、WFD屏幕投影等功能。在Android上DLNA、WFD都已经支持或部分支持，我的云需要做的工作就是为这些功能提供便捷入口。

PC上则需要做更多工作。DLNA和WFD都是在现有的TCP/IP技术上做的扩展，目前是开放标准，PC上需要实现这些标准协议。手机上已经有相关的实现，可以作为PC实现的参考

播放，可直接使用Windows Media Player，这一部分需要做一些预研。

# 总体框架

## 逻辑结构

通过上面的分析，可以看到我的云是一个分布在不同节点上的软件系统，这些节点是对等的。单个节点上，如下图所示，该软件大致包含三个层次:



### 应用层

这里是和用户直接交互的功能展示层，主要表现文件管理及互动分享操作，另外还有分类浏览和搜索两个辅助操作来提升用户体验，以及用于让用户输入必要的身份验证等信息的注册/登录界面。

Android平台上包括文件管理器、图库、互动分享三个主要应用，搜索、分类浏览功能集成到这几个应用当中了，不是单独的应用。

Windows平台上考虑到操作方式的差别，文件管理器和互动分享操作集成到一个应用中，图库暂时不考虑。

### 服务层

该层位于应用层之下，对上层提供抽象文件系统ZFS及索引数据库支撑服务，同时还有实现设备连接管理的设备管理服务，以及建立索引数据库的媒体扫描服务。

这一层在Android平台和Windows上都存在，实现的功能也完全一致。

### 协议层

这一层是各个具体的设备连接访问协议的实现层。目前包括：

#### 标准协议

这些协议是标准的或公开的协议/API集合

* WFD：用来实现WIFI Display屏幕投影分享模式；PC上可能没有WIFI设备，暂不实现；
* BT：用来实现单个文件的传输分享；PC上可能没有蓝牙设备，暂不实现；
* DLNA：用来实现DLNA媒体分享，同时也提供虚拟目录文件浏览、下载功能；
* FTP：用来在私有云存储节点间进行文件系统共享操作，是实现私有云文件管理功能的主要协议；
* 公有云：实现网盘文件系统的通信协议，各个不同网盘提供商的协议存在差异，目前调研过的百度网盘采用的是HTTP协议。
* LocalFS：各私有云存储节点上的本地文件系统，android上是Linux的VFS，Windows上是NTFS等。

#### 扩展协议

这些协议是我们为实现软件功能自定义的私有协议；

* Discovery：设备发现协议，完成私有云设备间的自动发现、连接的协议；
* IndexDBExchange：索引数据库交换协议，完成不同存储节点上的索引数据库的交换同步的协议。

## 物理结构

### 非协议相关组件

* Android平台：需要完成文件管理器应用、互动分享应用、图库应用改造、Media Provider改造；需要完成一个设备发现服务；需要完成一个注册登录应用。
* PC平台：需要完成我的云应用，其中包括文件管理器、互动分享应用、媒体扫描，设备发现等模块；不同模块可考虑使用DLL运行在同一个进程内，也可采用某种IPC手段分布到不同进程中。同一个进程中的结构访问简单，分布到多个进程中的结构稳定性好，不同的方案各有优缺点，模块详细设计时可自由决策。

### 协议相关组件

具体实现时，每个协议都需要有服务器和客户端两个组件。具体到每个协议：

* WFD：Android已完成全部功能，PC暂不支持；
* BT：Android已完成全部功能，PC暂不支持；
* DLNA：Android平台已有DLNA分享功能，需要完成ZFS的客户端封装；PC上则需要完成分享功能及ZFS的客户端封装；
* FTP：Android及PC平台都需要完成服务器和客户端，以及ZFS的客户端封装；
* 公有云：Android及PC平台需要完成ZFS的客户端封装；
* LocalFS：Android及PC平台需要完成ZFS的客户端封装；
* Discover：Android、PC平台、WIFI硬盘上需要完成客户端和服务器；
* IndexDBExchange：Android、PC平台、WIFI硬盘上需要完成客户端和服务器；

# 分模块设计

## ZFS子系统

ZFS子系统是我的云中的存储管理子系统的内部命名，其中包括一个抽象的ZFS文件操作接口层及其下的各种不同文件传输机制的封装实现层。



上图展示了ZFS子系统的静态结构，左边是ZFS子系统的客户端，中间及右边部分是ZFS子系统相关模块。

### ZFS抽象层

ZFS抽象层是一组文件系统访问接口，是实现我的云文件管理功能的重点。不同的文件访问方式通过这一层实现了统一，应用层只需要访问这个抽象层来实现文件系统的管理，而不用关心底层的文件操作的实现机制的差异。

ZFS抽象层为了实现通用性，其中定义的接口是所有需要管理的存储系统文件访问接口的并集。并非所有的设备都要实现所有的接口操作，有些操作不能被某种设备支持，例如DLNA FS不支持文件改名。在ZFS接口中可以设计一个特性声明操作来查询该设备能够支持的操作或不能支持的特性；客户应用程序可以通过此操作的结果来采取策略来禁用或打开相应的文件操作UI选项，比如对不支持改名操作的存储系统在弹出菜单中隐藏改名菜单项。

分析需求可见，ZFS的客户应用包括Java形式的Android文件管理器、C++形式的Android Media Service、以及C++形式的Windows我的云应用UI，因此ZFS抽象层是一个和开发语言无关的层，设计时其中只定义文件系统相关操作的集合，在实际应用时再映射到具体的开发语言形式。

ZFS接口应该包含的操作包括：

* 目录管理：创建、删除、重命名、列举等；
* 文件管理：新建、删除、重命名、读取、写入、查询属性等；
* 存储器信息：查询容量

具体实现时，为了屏蔽不同ZFS实现的创建过程，需要在此层提供一个对象工厂来为上层应用创建需要的ZFS对象。该工厂创建不同ZFS实例所需信息从设备管理器获取。

ZFS设计上是一组接口和多种实现，是典型的桥接设计模式结构。

### 本地文件系统上的ZFS



如上图，Local FS是ZFS的本地文件系统实现。这是ZFS的最简单实现形式，基本是直接对本地操作系统文件操作API的转调用即可完成。如果ZFS的实现映射语言为Java，还可以直接使用Java IO来实现。

### FTP上的ZFS



如上图，FTP FS是ZFS在FTP协议上的实现。每个云存储节点上都包含一个FTP Client通过FTP协议连接到远程的FTP 服务器上（部署在Z5手机、PC、WIFI硬盘上）。

工作流程：ZFS发起的文件操作会被FTP Client解释成FTP协议命令和数据操作，传送到FTP服务器，FTP服务器完成对当地文件系统的对应操作。

### DLNA上的ZFS



DLNA FS是ZFS在DLNA上的实现。和FTP上的ZFS非常类似， 部署在Z5手机/PC上的DLNA PROXY模块能够通过DLNA协议连接到支持文件浏览、下载的DLNA设备，ZFS发起的文件操作会被解释成DLNA协议的SOAP命令发送给DLNA服务器，DLNA服务器再和PROXY通信完成列举或文件传送。

### 公有云上的ZFS



Cloud FS是ZFS在云盘上的实现。不同的云盘提供的API形式也不一样，有的提供的是SDK，有的提供的是http协议，有的可能两者都提供。上图的Cloud Proxy模块对于提供有SDK的云盘系统实际就是SDK本身。

工作流程：ZFS发起的文件操作会转换为对Cloud Proxy的调用，Cloud proxy将这些调用解释成云盘内部通信协议（对我们调研过的百度云盘来说，其内部通信协议是HTTS）和云盘服务器通信，云盘服务器再和Cloud proxy通信完成云盘上的文件系统操作。

顺带说一下，百度网盘同时提供了HTTPS通信协议接口、PHP和Java形式的SDK，但SDK中没有提供文件随机读写，只有整体上传下载。要想完整实现ZFS的文件读写接口，需要使用HTTPS协议接口中的断点续传接口。

### 实现过程中的一些思考

#### 性能

为了实现跨设备的透明操作，文件复制时ZFS涉及到文件传输。

在同一个存储设备上的文件复制：如果本地文件系统支持，可以由本地文件系统完成，ZFS只需要发送一个指令即可。如果通信协议不支持复制（FTP协议就不支持复制命令），需要考虑扩展一个单独的命令。无法通过扩展协议完成本地复制的，处理方案同下述跨设备复制操作。

跨设备复制：不同设备之间无法获知对方的存在，所以拷贝操作需要由文件管理器来中转。文件管理器从源存储设备读取数据，写入到目的存储设备中。由于需要涉及大量的字节拷贝，如果ZFS实现模块是在和文件管理器不同的进程中，则存在巨大的跨进程通信开销。因此要求ZFS的所有实现模块的读写操作必须在是Host客户应用程序（如文件管理器）进程中的。这就要求ZFS的实现模块应该都是动态库（jar、SO、DLL）形式。非读写操作由于数据量较小，实现时可以自由选择独立进程或是动态库形式。

#### 多重连接

Z5手机和PC这两种设备的任意节点之间都存在FTP和DLNA两种连接，由于DLNA是标准协议，且无鉴权，无法区分DLNA是来自我的云还是其他公共设备。因此在文件管理器中会出现同一个设备的两个连接点。经过讨论，我们认为这个是无害的，就像在Windows的资源管理器中登录到一个自己机器上的FTP服务器，该服务器访问的目录在资源管理器中就会有两个节点，这实际是同一个目录的不同视图。

## 设备管理子系统

设备管理子系统是为管理各种设备的发现和连接而设计的软件模块，其中包括各种设备扫描服务及设备连接状态通知。大致结构如下：



这里DLNA发现及蓝牙设备扫描发现是标准过程，私有云连接协议是FTP，无法支持自动发现，需补充自定义的私有云发现协议。

### 私有云设备

#### 自动发现

因为私有云设备之间是FTP协议，在连接时需要指定服务器的IP地址、端口、用户名及密码这4项信息才能连接成功。私有云设备大多都是通过DHCP服务动态获得IP地址，每次都可能不同，普通用户不太容易获知当前的IP地址，即使能够获知，每次都去手工输入这些信息也会非常麻烦。FTP服务器的用户名、密码如果都是固定的，一旦泄漏，任何人都可以连接私有云设备，安全风险很大；如果经常强制要求定期更换用户名、密码，安全性会大大提高，但用户手工操作就很麻烦。为解决这个问题，我们设计了私有云设备的自动发现协议。

回顾前言，云的范围是指同一个注册用户的不同设备，不是同一个用户的设备不在我的云管理范围之内，同一个云内设备间可以共享资料，不同云之间或云外对云内不具有访问权限。这里显然要求一个设备能够加入到云中，必须是声明为属于某个注册用户，这就需要登录到我们的用户中心服务器。Z5手机在设计时已考虑为每个购机用户提供一个ZTEMT 用户账号，用户在启动手机后就会有启动向导程序引导用户注册，只有注册用户才能使用我的云功能。要加入某个注册用户的云，需要在安装有我的云应用的设备上使用注册用户的账号信息至少登录一次我们的用户中心服务器，以获取身份凭证；云设备间使用此凭证相互认证，以确认是属于同一个用户。为简单起见，我们将身份凭证直接设计为一个加密密钥。从服务器获取到的身份证密钥在本地安全存储，以备后面使用。

因为私有云设备是在局域网内互联的，因此我们在某个私有云节点设备上启动设备扫描时，可以发送一个组播消息，此组播消息内容为随机产生的，用身份证密钥加密。此组播消息能够被局域网内的所有加入了该组播的设备接收到，但由于是加密消息，只有拥有同一身份证密钥的设备能够解密此消息。成功解密消息的设备对消息内容按既定规则变形，重新加密发送回发送方作为广播消息的确认，由于消息内容已变形，回复的加密消息字节和广播的消息会不一致，这就避免了恶意程序监听网络包，将广播消息原样发回的欺骗手段。广播发送方接收到回应后解密，只有正确回应变形消息的节点才会被确认为合法节点。在回应的加密消息中，接收方还会带上本地FTP的服务器地址、端口、用户名、密码等信息，发起方后续就可以用这些信息自动建立FTP连接而无需用户参与。

整个发现过程如下：

1. 发起方A产生一个随机消息（如ABCD），用身份证密钥KeyA加密，组播播到局域网中预设的组播地址
2. 另一个拥有身份证密钥KeyA的节点B接收到此加密消息，使用KeyA解密，获取消息原文（ABCD）
3. B采用预先约定的变形规则（如翻转字符串）对消息原文变形得到回应消息（DCBA）,附加上本地FTP服务器的IP地址、端口、用户名、密码（这些都是随机产生且具有足够的强度），形成响应包，再使用密钥KeyA加密，发回给A
4. A使用KeyA解密响应包，检查回应消息是否是刚才发送的消息（ABCD）的合法变形（DCBA）,确认回应者的合法身份
5. 确认合法身份后，A利用响应包中附带的B的FTP服务器的IP地址、端口、用户名、密码信息连接B的FTP服务器，自动连接成功，自动发现完成。

这里只列出了设备A扫描获取连接信息的过程，如果第1步的消息中带上了自己的FTP服务器的IP地址、端口、用户名、密码等信息，也可以广播通知B自己接入，B可以直接执行第5步连接A。

由于此发现过程需要带上双方的FTP服务信息，要求FTP服务已经启动，因此该发现过程最好由FTP服务触发启动。FTP服务又需要在网络已连接时才能确定自己的IP、端口信息，所以FTP服务的启动应由LAN连接状态事件来启动。

当某个节点的FTP服务关闭时，应该发送广播通知其他节点自己退出。如果是因为网络故障或关闭，第一个发现访问故障的节点广播通知其他节点此故障节点退出。这些消息都采用加密形式。

这个发现过程不仅适用于使用FTP作为文件协议的情形，其他的如CIFS的类似情形也适用，只是传送的连接认证信息不同。

#### 登录注册

上述过程完全依赖身份证密钥KeyA，该密钥是通过登录用户中心服务器获取的。这个要求KeyA具有唯一性，每个用户在注册时服务器都需要为其分配一个唯一的身份证密钥，客户端登录时采用HTTPS安全连接来获取该密钥。如果用户后续修改自己的注册账号密码，则表示原来的身份认证信息不再有效，服务器需要为其创建新的密钥，所有利用原密码登录过的设备也不再拥有合法身份，需要重新登录服务器来获取新的密钥。为避免用户的困惑，可在修改密码的UI上醒目提示用户其私有云的所有节点都需要重新登录，否则不可用。

注册登录的模块需要保存用户的身份密钥，并提供给设备管理器，用来发起自动发现广播。

#### 设备标识

为了避免同一个设备每次连接时展示的信息都不同，需要为每个设备定义一个云内唯一的设备标识，以便应用层能够做一些用户友好的操作，比如文件管理器每次连接同一个设备时都定位到上次浏览的目录。原有的协议也是不支持的，需要加入到我们的发现协议中。

具体方法就是用户操作UI使用注册用户登录成功后，就在本地创建一个GUID作为该节点的设备标识并保存，直至其重新以另一个身份登录。GUID具有全球唯一性，因此能够确保在云内是唯一的，且在不同连接过程中都保持不变。

该标识对软件识别是足够的，但对用户不够友好，因此还应该增加一个用户友好设备名，此名称可由用户自定义。

在上述发现过程的第三步，B可以将自己的设备GUID及友好名也打包进响应包，A就可以直接获取到这个信息了。当然也可以定义一个交互协议，在后续去获取。

### 设备扫描及通知

设备扫描过程可能会比较耗时，因此应该避免扫描过程中让应用程序等待，采用异步通知机制。具体来说，设备扫描器组件应包含一个通知器，当检查到某个设备已连接或已断开，就发出通知（Android上可以是intent广播，PC上可以是消息或callback）。

不同的协议设备发现的机制不同，扫描器内部应该细分，按不同的通信类型划分成不同的模块，分开完成。这对简化结构、提升可维护性/可扩展性等都有较大的帮助。

### 设备管理器

设备管理器是设备管理子系统向外提供的接口组件，其中包含了设备扫描过程的启动关闭、既有设备节点信息的列举等。

由于上面ZFS子系统描述的原因，ZFS的各个具体实现的读写组件需要Host到应用进程中，而设备管理子系统在android上需要为多个进程服务，应该设计成独立的Service进程，因此设备管理子系统中只完成设备查找记录步骤，连接则是在应用进程中进行的。就上面的私有云设备发现过程来说，设备管理只处理1-4步，第5部则是由Host在各个应用中的FTP FS动态库来处理。

私有云的设备发现过程中需要的本机注册用户信息从注册登录模块获取，FTP用户名/密码/IP/端口等信息从FTP服务模块获取。

### 组件部署



上图是设备管理子系统在Android平台上的一个大致的组件结构图，该子系统应该作为一个独立的Service，使用Binder为不同应用提供设备管理服务。其内部包含各个发现协议的实现组件，其中私有云发现协议是需要全新开发的。

PC上的结构和Android上完全类似，但由于不存在多个应用，设备管理子系统可选择以DLL形式作为进程内服务，减少跨进程通信开发工作。

除了Android和PC外，设备发现的服务器部分还需要部署到WIFI硬盘上，为重用组件，上图的PrivateCloudDiscoveryServer应设计成Linux可执行文件，由于WIFI硬盘不支持Java，因此该Server应该使用Linux native API来实现。

## 分类浏览及搜索子系统

传统的OS文件系统都是按目录组织的数据存储结构，而普通用户关心的却是内容分类存储结构，这两个存储结构并不匹配，导致目前流行的趋势是利用软件将数据存储结构的中的内容按分类映射成一个虚拟的内容视图，而弱化了目录结构视图。这给用户带来了使用体验上的改进。但在内容视图中，同一种内容可能非常多，传统的翻页浏览对于用户查找内容效率极低，因此搜索又被引入进来，以提高用户定位内容的效率。我的云也重点强调了按内容分类浏览的内容视图及搜索功能。

### 搜索

Android平台上原生已经自带了搜索工具，其结构设计完整，具有很好的可扩展性，我们的搜索功能可以在其基础上扩展。Android原生搜索的架构大致如下图：



Android系统自身带有QuickSearchBox这个全局搜索工具应用，作为搜索的用户入口。该应用内部会通过SearchManagerService管理的多个SearchableSource对象来完成搜索。SearchManagerService是通过PackageManager扫描各个应用的manifest数据，从那些声明了android.intent.action.SEARCH或android.intent.action.web\_SEARCH的package中提取出SearchableInfo，用这些SearchableInfo可构建出SearchableSource。所以SearchableSource实际是各个支持搜索的package自己提供的，这些信息里包含了一个ContentProvider，SearchableSource就是在这个ContentProvider中进行搜索。搜索结果保存在抽象的Suggestion中，通过intent传递给应用。

可以看出，android搜索框架已经包含有很好的扩展性设计。不同的搜索源通过SearchableInfo中描述的ContentProvider抽象出来了。新的搜索源只需要提供自己的ContentProvider，同时在Manifest中声明对SEARH/WEB\_SEARCH两种intent action的支持，就完成了向SearchManagerService的注册。因此整个框架结构上是完整的，对支持新增云存储搜索的需求能够完全支撑，基本不需要做修改。我们需要做的就是为云存储制作一个搜索源。

### 分类浏览

OS中的文件都是按目录存储的，而不是按文件分类存储，因此要实现分类浏览，必须对所有文件扫描分类。扫描过程受限于存储设备的IO速度，一般都会需要一个较长的时间：实验数据在Android上扫描7800个文件需要约120秒，PC上由于是机械硬盘，速度可能还会慢一个数量级。因此在用户查看时去及时扫描显示是不可行的，必须采用某种策略，将扫描的结果保存到一个索引数据库中，在以后的应用场景中复用此扫描结果。这样只需要完整扫描一次，然后对文件系统的更改做增量扫描，后续的分类查看就会是数据库搜索操作，速度会非常快。

Andoid系统上实际已经包含一个分类浏览视图，那就是图库。它能够将手机上的所有图片类文件整理后集中在图库应用中显示。分析图库应用的软件结构，可以发现它能够实现这个功能主要还是在于它内部建立了一个索引数据库，由Media Scanner组件按一定策略扫描手机SD卡上的文件，将它们按一定规则分类存储在若干个数据表中，图库就是从这个索引数据库中获取相关的图片文件信息，用来快速显示概要信息的。

参照这个方案，我们只需要在数据库中增加我们需要的文档、音乐、视频、压缩包等分类类型标签，我们就可以通过在这个数据库中快速查询来组织分类浏览视图。此数据库无疑可以作为文件搜索的搜索源，来满足搜索云文件的需求。我们需要解决的就是如何建立此索引数据库。

### 索引数据库

Android系统内部的扫描器完成了对手机本地文件系统的扫描，但对于云存储系统上的其他节点，需要我们实现扫描器完成扫描工作。

这里有两种方案

1. 每个云节点独立扫描所有已连接的设备文件系统，在本地建立所有设备的索引数据。
2. 每个云节点只扫描本地文件系统，并和连接的设备交换索引数据库。

方案1存在一个不可克服的困难，当远端云设备离线更改文件系统时，本地扫描器得不到更改通知，因此没法进行增量扫描。

因此方案2基本上是唯一选择。但也存在一些困难或疑问：

* 交换索引数据库的时机。明显的一个比较合适的时机是每次设备连接时。这里可能会存在一个数据量的问题，如果每次连接都交换索引数据库，这个交换传输的数据量是否能够接受？实验的数据是，7800个文件形成的索引数据库大小约为3.5MB，这个数据库用WIFI来传输大概在1秒上下，基本可以接受。而且该数据库内容基本都是文本，如果采用压缩传输，则还能够大幅减少传输时间。另外可以在交换索引数据的协议中加入上次交换的时间记录或数据库版本记录，如果在此之后没有更改，则不用重新传输。这会更进一步优化交换索引数据库的时间。
* 文件系统修改后的其他各个点的同步。最简单的方法是每次更改后向其他所有节点都重新传送新的索引数据库。这个对于频繁修改和索引数据库较大的情况下会出现性能问题。应该只传送修改内容的扫描结果的数据库记录。
* 公有云无法支持交换索引数据库。这个问题基本无法简单解决，前期可以考虑将公有云从分类浏览中排除。另一种可能的方案是公有云中保持一个索引数据库，但该数据库的更新由所有设备共同完成，每个设备对公有云修改后都需要主动更新索引文件。为避免多个节点对该文件的并发访问，可以通过我们的服务器来仲裁加锁，因为如果能够访问公有云，也必定能够访问我们的服务器。

未定制的WIFI硬盘无法支持交换索引数据库。状况基本同公有云类似，但加锁机制不同，需要在局域网内通过选举形成一个锁服务器。由于受经济、人力资源限制，需要并发维护索引数据库的情形我们目前暂不考虑，这些设备上的分类浏览和搜索功能不支持。

为保持性能和方便同步，索引数据库交换后不进行合并，而保持多个独立文件，对其检索就会形成的多个数据集，对这些数据集无法形成统一的排序，因此文件管理器中依赖于数据集排序的按字母序滑动查找的功能对于多个设备合并显示的功能暂不支持。

### 索引交换协议

通过以上分析，为建立本地索引数据库，所有的节点上都需要一个扫描器按一定策略扫描本地文件建立索引。在Android上该扫描器可以直接利用Media Scanner进行改造，PC上和WIFI硬盘上则需要重新开发。

可以看到索引数据库的交换协议是完成索引的关键。索引数据库能够交换的前提是所有节点上的索引数据库的结构是完全一致的，这样才能被其他节点解释使用。交换协议大致包含以下几个阶段

#### 交换索引数据库

设备发现后，设备A向远端节点B发送一个索引数据库请求，该请求中包含本地存储的远端节点对应的索引数据库。远端节点B收到该请求后，如果本地索引数据库建立完成，就回复该数据库的文件路径。发起设备A通过FTP下载该索引数据库，替换本地对应的数据库（可简单用设备节点B的设备GUID命名该数据库文件）。

鉴于有些节点的文件系统可能不是频繁更新，我们可以优化此过程。我们为索引数据库建立一个版本标记，设备A向远端节点B发送一个索引数据库请求时带上本地存储的B节点索引数据的版本信息；B收到请求后对比自身存储的索引数据库的版本信息，如果对方版本号和自身的一致，就无需传输索引数据库文件。

另外，如果索引数据库文件尺寸过大（阈值可通过实验确定），其传输时间超过压缩时间时，则B在回应时可以传递一个压缩文件路径。

#### 索引同步

在首次索引文件交换成功后，后续可能还有文件系统更改（新增、删除文件、目录）。扫描器可通过文件系统监控（Linux系统用inotify，Windows系统中用文件系统监控API FindFirstChangeNotification等），获知这些改变，将改动部分增量扫描加入到本地索引数据库。这时一般是小修改，重新传送整个索引数据库文件不经济。可考虑有扫描器在更新数据库时，将更新的记录打包，通知给所有已连接的节点。

对于PC，可能还会让用户自行选择一个共享目录作为云设备根目录，这个根目录如果切换了，索引数据库是需要重新扫描建立的，这时需要发送一个通知，让所有已连接的节点暂停对本节点的分类浏览；扫描完成后再发送一次通知，所有已连接的节点重新获取新的索引数据库。

### 组件结构



分类浏览、搜索的核心是索引数据库，实现索引数据库包括建立索引的扫描器和完成索引数据库交换协议的Server/Client等三个组件。这三个组件需要部署到Android、PC和WIFI硬盘上。Android和WIFI硬盘由于Linux内核的同源性，可以共用相同的实现；PC上需要独立实现。

### 对缩略图的支持

图库、文件管理器分类浏览图片时，除了显示时间、媒体尺寸等信息外，还有一个需求就是能够以缩略图形式来展示部分内容。一般来说缩略图都是由文件系统之外的其他软件来生成的，需要读取整个图片文件。对于远程文件系统显然来不及，即使是异步读取，性能也是很低的。

可以考虑和索引数据库一样，每个节点自己在媒体扫描时建立图片文件的缩略图缓存，这一部分数据独立于索引数据库存储，在交换索引数据库时不进行传输。当远程需要缩略图时，采用请求应答的方式，逐个传送需要的缩略图，UI上再异步更新显示。由于缩略图数据量一般较原图小很多，所以异步显示的效率应该是可以接受的。

## 互动分享子系统

互动分享是使用标准协议完成，除了设备管理，主要工作在应用层UI上，这里不做叙述。

# 分设备设计

## WIFI硬盘

目前我们采用的WIFI硬盘带有FTP服务的，用户接口是WEB方式。我们只要求硬盘厂商提供一个用户配置页面，以输入注册的ZTEMT用户名、密码。其他工作全部由我们自己来完成，这样可最小化定制工作量，同时也能够对我们的软件设计起到保护。



上图是WIFI硬盘中的软件逻辑结构，右边的部分是我们提供的。各模块交互逻辑大概如下：

* WebServer接收用户输入的用户名、密码，通过命令行传递给登录认证程序
* 登录认证程序通过HTTPS与用户中心服务器通信，认证用户并获取用户身份密钥，保存到本地，通过IPC通知Discovery程序
* Discovery程序是私有云发现协议Server，它监听指定的广播端口并使用登录认证程序提供的用户身份密钥来处理从广播端口接收到的设备发现消息，Discovery程序在设备发现消息的响应中带上FTP服务器ftpd的配置的用户名/密码
* Discovery程序同时也作为ftpd的控制器，负责为ftpd产生随机的安全的用户名/密码配置，并重启ftpd使新配置生效

需要注意：

* WIFI硬盘可能有两个IP地址，也就有两个网卡，在实现时要区分监听的网卡
* Discovery程序需要一直运行，应该成为开机启动的守护进程
* 登录认证程序需要检查是否已有用户密钥，如果没有需要尝试连接用户中心服务器。

## Android

略

## PC

略

# 总结

上图是最终的包含了设备自动发现及索引数据库交换协议的设备连接图。每个节点需要完成的工作是实现这些连线上的协议，Z5、PC、WIFI上额外包含扫描器，Z5和PC上还包括用户界面、管理各种设备的设备管理器组件。

## 整体部署图



## WIFI硬盘组件图



WIFI硬盘的私有云发现协议只需要Server，因为它不需要主动连接其他节点，只需要接受其他节点的连接。

## Z5手机组件图



## PC组件图



PC相对于手机少了WFD及BT功能，在物理结构上也会少一些进程。