

浙 江 大 学

本 科 生 毕 业 设 计 报 告



项目名称 基于 iOS 的投资信息系统的设计与实现

姓 名 谭歆

学 号 3080101204

指导教师 蔡亮

专 业 软件工程

学 院 计算机学院

A Dissertation Submitted to Zhejiang
University for the Degree of Bachelor of
Engineering



TITLE: The Design and Implementation
of an Investment Information System
based on iOS

Author:	<u>Xin Tan</u>
StudentID	<u>3080101204</u>
Mentor	<u>Liang Cai</u>
Major:	<u>Software Engineering</u>
College:	<u>Computer Science</u>
Submitted Date:	<u>2012-05-16</u>

浙江大学本科生毕业论文（设计）诚信承诺书

1. 本人郑重地承诺所呈交的毕业论文（设计），是在指导教师的指导下严格按照学校和学院有关规定完成的。
2. 本人在毕业论文（设计）中引用他人的观点和参考资料均加以注释和说明。
3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。
4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

毕业论文（设计）作者签名：

_____年_____月_____日

摘要

自从苹果公司发布第一代 iPad 以来，平板设备就在越来越多的领域上展现了其卓越的生命力与竞争力，在短短的两年时间内，从无到有，再到全面占领消费者市场，平板电脑以其独特的优势向人们展示了一个神话。诸如 iPad 的平板电脑让人们可以随时随地进行办公、娱乐，其便利性与持久性是普通笔记本电脑无法替代的，再加上像多点触控、加速度感应、GPS 定位及三轴陀螺仪等高端、新颖、人性化的交互体验，更是让平板电脑增色不少。在生活节奏日趋加快的今天，平板电脑的出现可谓是应运而生，极大地增强了办公的移动性等。但同时，美中不足的是，平板电脑体积有限，无法拥有像普通 PC 一样的运算性能，无法扩展存储容量，一切均为低功耗设计。然而，网络是无限的，将有限的平板电脑接入无限的网络，最终创造无限的可能是本文的重点。

云计算（Cloud Computing）是一个新兴的互联网应用领域，云存储是云计算中着重于提供高性能存储空间的一种互联网服务，二者可以弥补平板设备的性能问题。

本文旨在提供一种，将 iPad 应用到投资信息管理上，并借助云存储扩展 iPad 对资源的访问能力的一套解决方案。

关键词 iPad；平板电脑；iOS；云存储；同步；信息系统

Abstract

Since Apple released the 1st generation of iPad, tablet devices is showing its superior vitality and competitiveness in a growing number of fields, in just two years, from not even exists, to the full occupation of the consumer market, Tablet PC show people a myth with its unique strengths. Tablet PC, such as the iPad, allows people to work and do entertainment anywhere anytime, the convenience and persistence can't be replaced by a regular laptop, in addition, the multi-touch, accelerometer, GPS positioning and 3-axis gyroscope which give an advanced, innovative and user-friendly interactive experience, make the Tablet PC even more considerable. As the pace of life is increasingly accelerated today, the appearance of Tablet PC is at a right time, which greatly enhanced the office mobility. But at the same time, the only drawback is, be limited by the volume of tablet PCs, it cannot own a good computing performance as same as a desktop, and the storage capacity cannot be extended; everything is designed to a low battery usage. However, the network is unlimited; to connect the limited Tablet PC to an unlimited network, and ultimately to create endless possibilities is the main purpose of this article.

Cloud Computing is an emerging Internet application domain, cloud storage is a kind of cloud computing which focused on providing high-performance storage space, these two can make up the performance problem of tablet devices.

This article aims to provide an enterprise solution for investment management, which takes the benefit of iPad and Cloud storage.

Keywords iPad, Tablet, iOS, Cloud Storage, Synchronize, Information System

目录

摘要	I
Abstract.....	II
第 1 章 绪论	1
1.1 项目背景.....	1
1.2 平板设备的发展.....	1
1.2.1 平板电脑的优势.....	2
1.2.2 平板电脑的定位.....	2
1.3 云计算.....	3
1.3.1 云计算的优势.....	3
1.3.2 云存储.....	4
1.3.3 GlusterFS.....	4
1.4 本文结构.....	4
第 2 章 项目实施方案	6
2.1 项目目标.....	6
2.1.1 基本目标.....	6
2.1.2 高级目标.....	6
2.2 技术框架简介.....	7
2.2.1 iOS	7
2.2.2 JSON	7
2.2.3 RSS.....	8
2.2.4 RESTFul.....	8
2.2.5 OAuth.....	8
2.3 系统结构.....	9
2.4 iPad 客户端设计	11
2.4.1 预览图生成模块.....	12
2.4.2 RSS 读取模块.....	12
2.4.3 云存储访问与多线程下载模块.....	12

2.4.4 文档操作模块.....	13
2.4.5 技术难点	13
2.5 服务端设计	14
2.5.1 数据库设计.....	14
2.5.2 接口设计	16
2.5.3 Email 处理模块.....	18
2.5.4 设计亮点	19
2.6 本章小结	21
第 3 章 项目实施	22
3.1 客户端实现	22
3.1.1 预览图生成.....	22
3.1.2 云存储访问.....	22
3.1.3 多线程下载及进度条.....	30
3.2 服务端实现	35
3.2.1 登录验证	35
3.2.2 同步 Fund	36
3.2.3 同步 Notes	36
3.2.4 Email 抓取	37
3.3 本章小结	38
第 4 章 项目成果	39
4.1 iPad 应用	39
4.1.1 Home.....	39
4.1.2 Favorite	41
4.1.3 News	42
4.1.4 Library	43
4.1.5 Management.....	45
4.2 服务端代码	45
4.3 本章小结	45
第 5 章 项目展望	46
5.1 项目总结.....	46

5.2 前景展望.....	46
参考文献	47
致谢	48

第1章 绪论

1.1 项目背景

在讯息千变万化的 21 世纪，如何及时、准确地把握投资信息将成为成功的关键。随着生活节奏的加快，信息来源的增加，传统的 PC 上处理事务的方式已经捉襟见肘，略显不足。人们越来越迫切地需要能够随时随地、自由地办公，虽然笔记本电脑可以部分满足要求，然而由于其自身的如电池续航能力不强、体积大，携带笨重等缺点，大部分时间人们还是会在桌面上使用笔记本电脑，它的移动性也就成为了摆设。

平板电脑的出现会让这一切有所改观。

1.2 平板设备的发展

2010 年 1 月 28 号，苹果公司发布了第一代 iPad，全球便掀起了一股平板电脑的浪潮。

平板电脑这一概念并不是苹果首创，而是来自微软。当时微软构想了一款不需要键盘、不用翻盖、方便携带却又功能完整的 PC 产品[1]，却又没能成功地进行推广，但由此衍生出了可用触笔写且屏幕可旋转的平板电脑变体——tablet PC。IBM 的 ThinkPad X 系列中的几款产品便是很好的原型。

苹果公司的 iPad 重新定义了平板这一名词，使其更加切合产品的本意。与以往的 tablet PC 不同，平板电脑（Pad）自身具有诸如多点触控、重力感应、携带方便等特点，很快便占领了消费者的市场。对于商务人士而言，出差或旅行途中携带大量沉重的行李不是件轻松的事，传统的笔记本电脑自然无法胜任。平板电脑的体积则介于笔记本电脑与掌上电脑（PDA）之间，结合了掌上电脑的轻便与笔记本电脑的强大功能，对日常的上网、即时通讯、文档查看、收发邮件等都可以轻松应对。可以说，平板电脑的流行是对上网本的一个致使打击。

基于平板独道的交互体验，各种新颖的应用层出不穷；基于 iOS 的 iPad 和基于开源的 Android 系统及 Windows 8 的各种平板电脑新产品的出现也更是为消费者们提供了极大的选择余地。

1.2.1 平板电脑的优势

当今主流的平板电脑都配备了 10 寸左右的多点触控屏、重力及加速度感应计、磁场感应计、三轴陀螺仪、蓝牙、WIFI、光线感应器等等，多元化的传感器也使得它拥有足以颠覆传统的键盘、鼠标等的交互体验。

总体上讲，平板电脑较笔记本电脑和普通桌面电脑有以下方面的优势：

- 便携性。小且方便移动使用。
- 交互性。多点触控，方向感应等。
- 持久性。更长的电池续航时间。
- 活跃性。应用开发周期短，更新及维护方便。
- 安全性。目前病毒、恶意代码较少。

如图 1-1 所示，除了 1 月份增长数较低外，2 月、3 月都达到了八万多个应用，即每天都有 2800 多新应用上架。而这还仅仅是 App Store 的中国区的统计结果[8]。平板电脑的活跃性可想而知。



图 1-1 2012 年第一季度 App Store 中国区新增应用个数

但同时，另一方面由于体积小，散热差，它又存在存储空间小、性能低下、单个硬件不能更新升级、外设接入不方便等不足。

1.2.2 平板电脑的定位

平板电脑从一开始流行就注重在娱乐上，不难发现，苹果公司 App Store 中游戏应用总是更新最快的（如图 1-2）。其他一些行业应用，如电子商务、医学、教育、物联网等也都在发展，但相对缓慢，这也是本项目诞生的背景和机遇。平板电脑定位于商务是十分合理的，首先商务注重移动性，其次注重信息的实时性和可展示性。移动性平板天生具备，配合 WIFI 和 3G 平板也能轻松做到实时在线，外加比掌上电脑好得多的显示效果，向客户介绍业务和成绩等均不在话下。

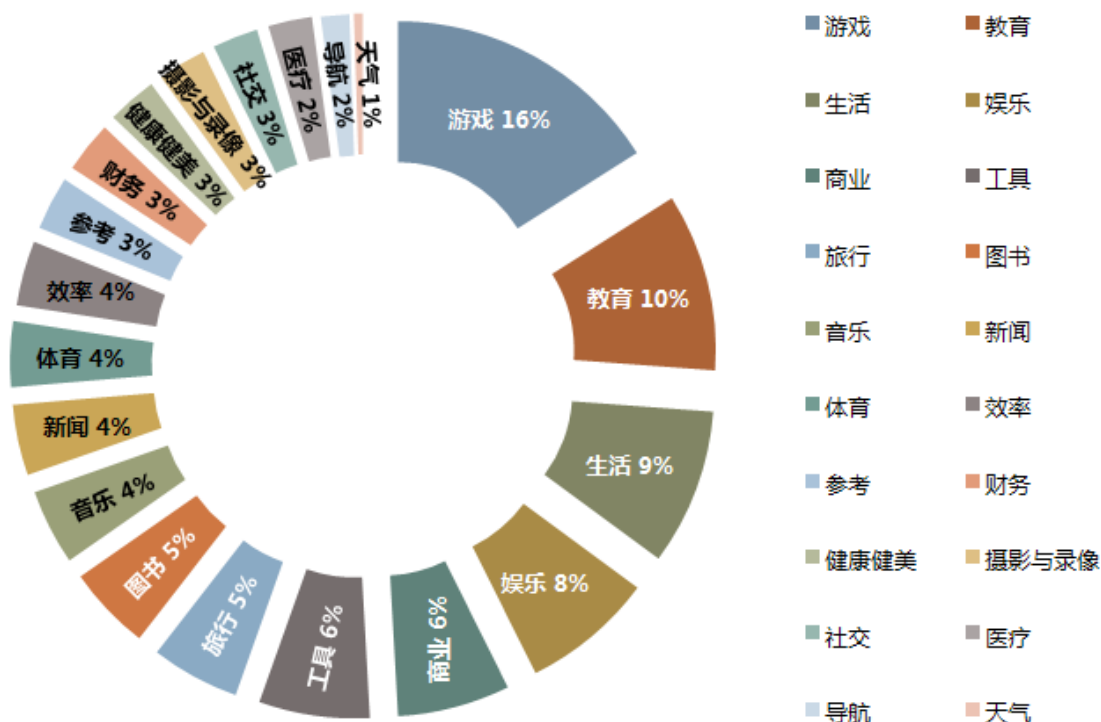


图 1-2 2012 年 3 月新增 iPad 应用类别比例

1.3 云计算

云计算是一种基于互联网的新的计算方式，这种方式允许将集成化的、拥有强大性能的软硬件资源按需共享给其它设备使用，它是 80 年代大型计算机到客户端、服务器的大转变之后的又一种巨变[2]。云计算拥有复杂的后台控制，却可以留给用户简单、明了的应用接口，用户不需要了解云计算背后的技术细节，不需要知道云计算服务的地理位置，无需计算机相关的知识，更不用直接控制云计算资源。对于用户来说，云资源是趋于无限大的，当然这得益于云计算服务采用的虚拟化技术。

一般来说云计算包括以下三个层次的服务：

- [1] 基础设施即服务（IaaS）
- [2] 平台即服务（PaaS）
- [3] 软件即服务（SaaS）

1.3.1 云计算的优势

正如上一节所提到的，云计算具有常规计算方式所不具备的优势。总体来

讲，云计算具有地理位置无关性、设备无关性、时效性、可靠性等特点，这使得它迅速建立来了一个全新的市场领域，新的服务模式。

位置无关性主要表现在：只要有网络的地方就可以计算。云计算本身是一种基于互联网的服务，在有网络访问的地放就可以访问相应的服务。设备无关性在于所有的设备上的应用都是基于同一种标准开发，无论客户端设备是什么，均能得到同等的服务。时效性一方面在于所有的服务都已经现有，而且随时可用；另一方面也在于云计算本身强大的计算能力，其性能瓶颈基本上在网络传输上而不是计算本身。云计算一般都会在一定范围内集群大量计算机，任何一部分的计算机故障都不会导致整个系统的崩溃，可靠性比小型计算机系统好的多。

1.3.2 云存储

云存储可以理解为云计算的一种特殊模式。当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置大量的存储设备，那么云计算系统就转变成为一个云存储系统[3]，所以云存储是一个以数据存储和管理为核心的云计算系统。现在有许多成熟的商业云存储服务，如 Box, Dropbox, LiveOffice 等，当然也有不少优秀的开源云存储平台，如 GlusterFS。

1.3.3 GlusterFS

GlusterFS 是一个优秀的开源分布式文件系统（官方网站：<http://www.gluster.org/>），基于它可以构建出一个高性能、高扩展性、高可用性的云存储系统。它采用弹性的哈希算法，去除了元数据服务器的依赖，使得它可以线性地扩展存储容量；同时它支持多用方式的冗余，极大限度上保证了数据的安全；可以在线添加与删除存储设备而不影响其它操作，等。基于以上的特点，部分商用云存储系统是在它的基础上做的二次开发，如 Box。

1.4 本文结构

绪论部分到此结束，本文后面将分章节具体介绍项目实施和实现的技术细节，其中：

第二章介绍项目实施方案；

第三章讲项目实现的一些细节；

第四章为项目成果展示；
第五章是项目应用前景展望；
最后为参考文献与致谢。

第2章 项目实施方案

2.1 项目目标

随着信息技术的发展，信息的格式多种多样，来源也是五花八门，同时更新频率高，销售与市场分析人员对信息的追踪越来越困难。如何将信息有效地整合以提供资讯监视的便利？本项目由此诞生。本项目为外包项目，需求方为美国一家公司（以后简称公司）。该项目力求构建一个能将各类资讯（主要是文档）整合在一起的、方便销售人员查看与分享的信息平台。

2.1.1 基本目标

项目的基本需求是：用户能够访问云存储服务 Box 以提升存储 iPad 的存储能力；用户可以在本地建立文件夹以放置相关度较高的文档，并搜索文件名；用户可以将最常用的一个文档放到一个特别的地方以供快速查看；用户可以对这些文档作一些笔记、心得等，并提供搜索；用户可能从 RSS 中获取最新资讯，并可以查看详情；用户可以预览云存储文档，并下载云存储中的文档到本地；用户可以对下载好的本地文档做删除和通过 Email 分享操作；等。

2.1.1.1 名词解释

基于以上需求，项目中产生如下名词：

- Fund。一个本地的文件夹，用于集中放置一些相关的文档。
- Notes。与 Fund 相关的笔记。
- News。主要是一些投资、销售方面的 RSS 订阅。

2.1.2 高级目标

在基本目标的基础上，需要实现如下高级版本的功能：

1. 应用可以多用户使用。使用前用户需要到公司官方网站上注册一个帐号，在应用启动时提供一个登录页面，使用注册好的信息登录。
2. 用户登录后，只能看到在这个 iPad 上自己的 Fund 及 Notes。
3. 用户可以同步自己的 Notes 到服务器，在另一台设备上登录后依然可以看到。

4. 用户可以绑定自己的社交网络帐号以分享 Notes。
5. 用户可以访问多个云存储服务, 并对其上传、下载、重命名、删除等多种操作。
6. 用户可以向公司邮箱以特定的格式绑定到某个 Fund 发送邮件, 然后登录 iPad 应用后可以在对应的 Fund 下看到邮件内容及附件, 并可以像云存储一样下载附件。
7. 用户可以定制自己的 RSS 链接。

2.2 技术框架简介

2.2.1 iOS

iOS 是苹果公司开发的一款移动设备的操作系统, 原名为 iPhone OS, 且只用于 iPhone 上, 不过现在随着苹果产品的革新, iOS 已经用于苹果的多种设备上, 甚至包括 Apple TV。iOS 基于 Darwin 开发, 其系统核心架构分为四层[4] :

- 核心操作系统层 (the Core OS layer)
- 核心服务层 (the Core Services layer)
- 媒体层 (the Media layer)
- 可轻触层 (the Cocoa Touch layer)

得益 iOS 精巧的设计, 使得它可以高效率地运行在硬件配置并不高的设备上。它对触屏拥有原生的良好支持, 让用户可以简单地通过手指来进行各种操作, 且精度很高。它流畅的动画效果更是为 iPhone 等产品增色不少。

和 Mac OS 一样, iOS 的软件开发也必须在 Mac 系统中。苹果的 Xcode 是进行 iOS 开发的得力工具, 用户只要安装完无需任何配置便可以马上进行开发与调试。目前 iOS 的最新版本为 5.1, Xcode 最新版本为 4.3。

2.2.2 JSON

JSON 是一种数据交换格式, 它较 XML 具有数据量小得多的优点。简单来说, JSON 用于对象——文本——对象的转换。JSON 的对象是一组无序的名称/值 (key/value) 对, 这是许多编程语言中都有的概念, 相当于字典的概念。数组在 JSON 是一种特殊的对象, 它包含一系列的有序的对象。JSON 用于数据交换主要优势有: 数据量小, 解析生成操作方便, 便于人和机器的理解。

JSON 也是一种语言及客户端无关的技术, 且其本身传输为文本, 可以压缩

以提高传输效率。所以在数据的传输上考虑使用 JSON。

2.2.3 RSS

RSS (Really Simple Syndication) 英文原意是"聚合真的很简单", 它是一种消息格式, 主要用在需要经常更新内容的网站上, 它提供一种标准让网站可以将标题、时间、摘要、消息来源链接等信息自动发布, 并让用户可以根据自己的喜好轻松地订阅这些消息。

RSS 消息格式基于 XML 标准, 任何能够解析 XML 的程序可以对它进行解析, 只不过它使用定制化后标签。

2.2.4 RESTFul

REST (Representational State Transfer) 是一种软件架构风格, 它从资源的角度来观察整个网络, 分布在各处的资源由 URI 确定, 而客户端的应用通过 URI 来获取资源的表征。RESTFul 则是遵循 REST 并基于 HTTP 的一种 WEB 服务。较于如 C#之类的 Web Service, RESTFul 具有代码无关性、客户端无关性等特点。因为它是基于 HTTP 的, 任何能够发送 HTTP 请求的设备均可调用 RESTFul 服务。另外, 采用 REST 的服务器端可以利用自身的缓存机制来加快响应速度、提高性能, 也能提高可扩展性及兼容性等。

在本项目服务器的 API 设计中, 可以考虑采用 REST 风格。

2.2.5 OAuth

随着互联网的发展, 信息之间的相互依赖越来越强, 各类互联网服务的整合已经成为一种趋势。与此同时, 服务商也越来越注重用户信息的安全, 如何既不让 A 服务知道用户在 B 服务上的帐号密码, 又能使 A 安全地访问 B 服务的资源成为一个难题。OAuth (开放授权) 的发展解决了服务整合过程中的验证与授权问题。

使用 OAuth 进行认证和授权的过程如下所示[5] :

1. 客户端 (应用程序或网站) 想操作用户存放在服务提供方的资源。
2. 客户端向服务提供方请求一个临时令牌 (Request Token)。
3. 服务提供方验证客户端的身份后, 授予一个临时令牌。
4. 客户端获得临时令牌后, 将用户引导至服务提供方的授权页面请求用户授权。在这个过程中将临时令牌和客户端的回调连接发送给服务提供方。

5. 用户在服务提供方的网页上输入用户名和密码，然后授权该客户端访问所请求的资源。
 6. 授权成功后，服务提供方引导用户返回客户端的网页。
 7. 客户端根据临时令牌从服务提供方那里获取访问令牌 (Access Token)。
 8. 服务提供方根据临时令牌和用户的授权情况授予客户端访问令牌。
 9. 客户端使用获取的访问令牌访问存放在服务提供方上的受保护的资源。
- 如图 2-1 描述了整个流程。

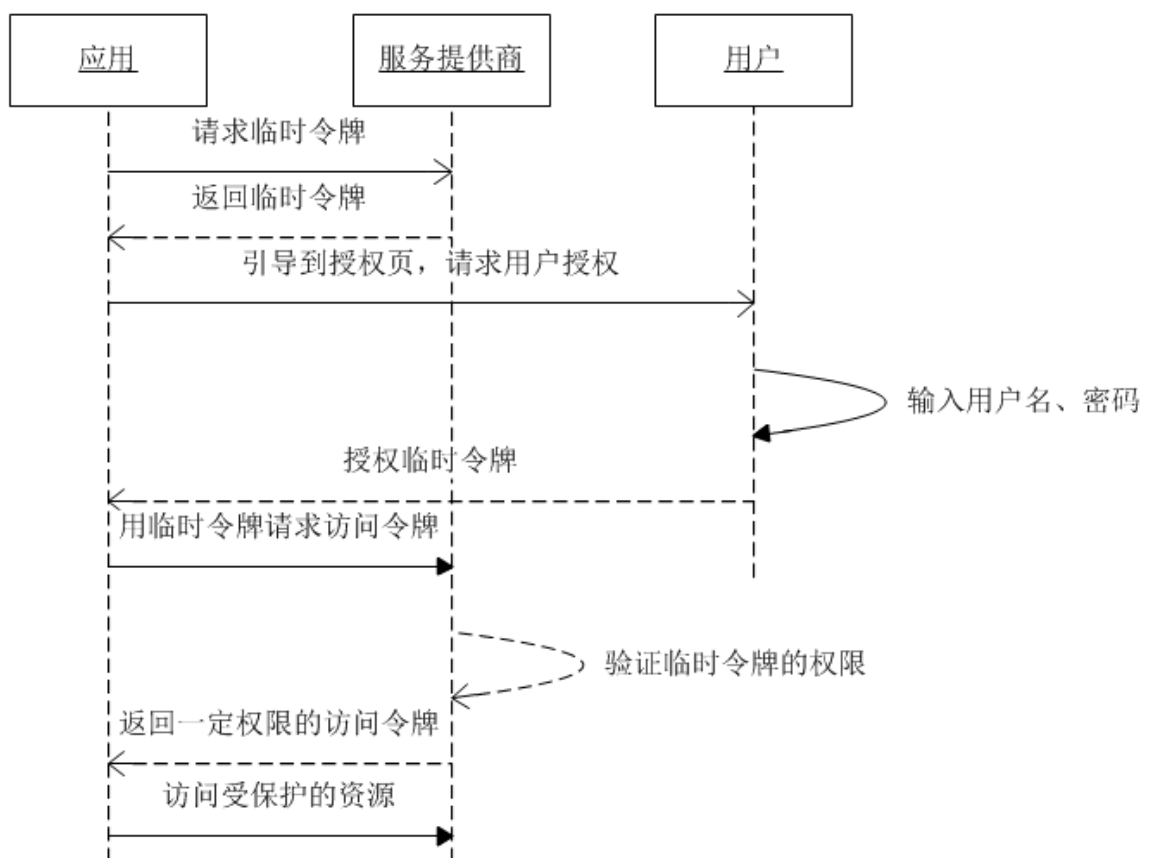


图 2-1 OAuth 认证流程

2.3 系统结构

项目基本目标不需要服务器支持，结构较为简单，大致可以描述如图 2-2：

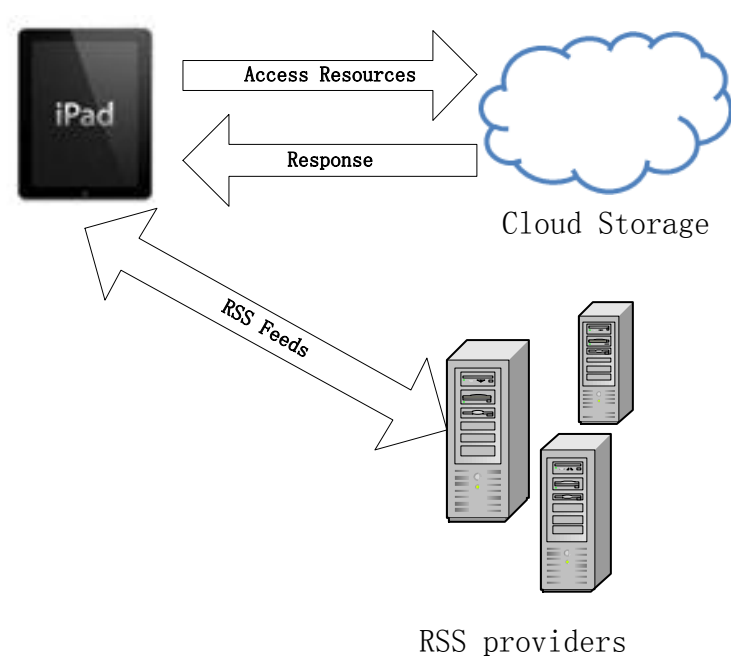


图 2-2 基本目标下的系统结构

高级目标加入了服务端的需求，所以在整体结构上多了一层，如图 2-3：

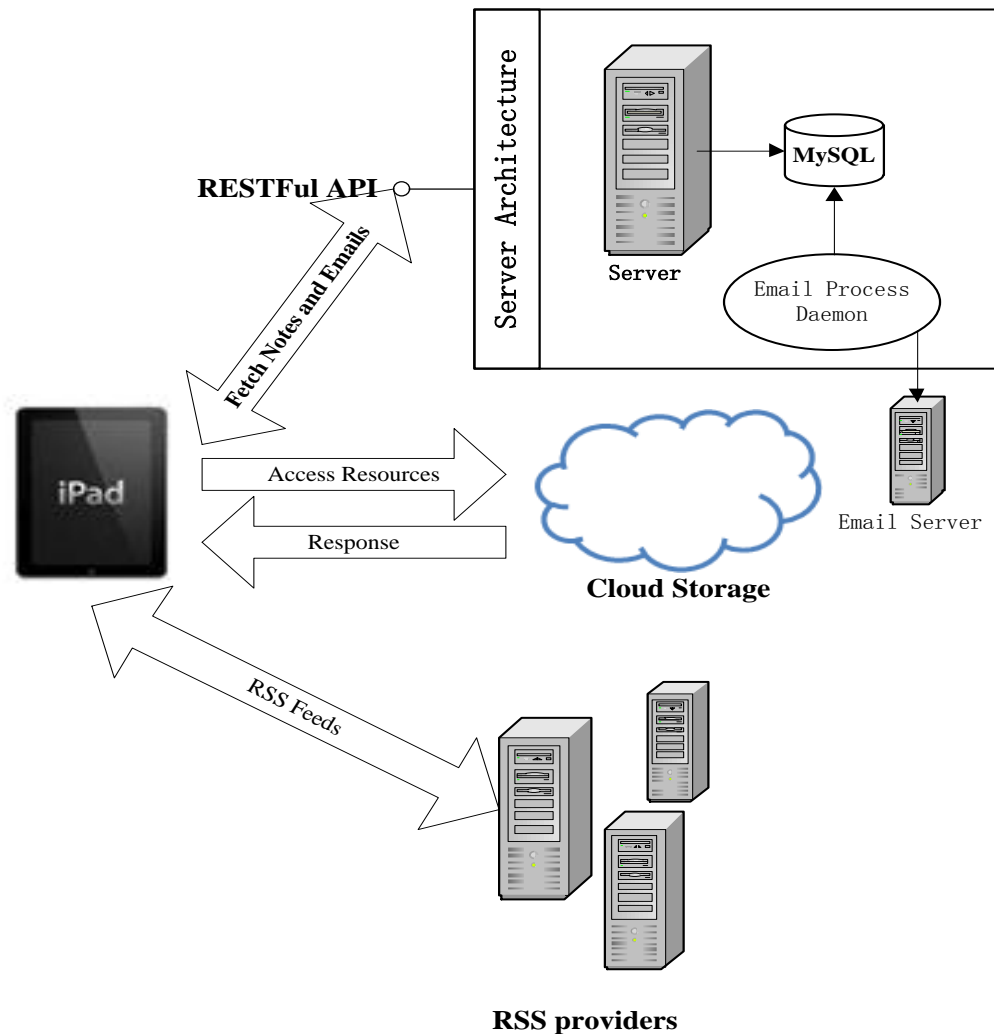


图 2-3 高级目标下的系统结构图

2.4 iPad 客户端设计

iPad 应用程序在用户交互界面上主要分为五块内容(如表 2-1),其中 Home, Favorite, News, Library 集成为一个主界面的四个子标签页,方便用户点击切换, Management 页面在用户点击按钮时才会出现。

表 2-1 主要交互界面说明

界面名称	界面说明
Home	这里是用户自己创建的 Fund 及与 Fund 相关的 Notes, Fund 中的文件分可读文档和多媒体文件分别放置, 并显示文件的预览图, 方便用户查找。用户在这个界面中查看下载到本地的文件, 并可以在这个界面中删除文件, 以及拖拽一个常用文件到 Favorite 中。
Favorite	这里是用户收藏的一个常用文件, 可以是可读文档, 也可以是多媒体文件。
News	这里是 RSS 订阅的内容, 用户可以选择不同的 RSS 源, 并可以查看详细内容。
Library	这里是云存储访问入口, 可以浏览云服务中的文件, 并可以点击预览。用户可以登录以授权应用访问云服务, 并可以登出。
Management	这是用户创建和删除 Fund 的地方, 并可以通过拖拽方式下载云存储中的文件到 Fund,

功能上, iPad 客户端应用主要分为以下模块。

2.4.1 预览图生成模块

在 Home 界面中需要显示所有文档预览图列表。应用需要支持的可阅读文档格式包括: PPT, DOC, XLS, PDF, 多媒体文档包括: MP4, MOV, MP3。iOS 对 PDF 文档格式有良好的底层支持, 可以方便简单地获取到预览图, 而对 MS Office 系统文档支持有限, 需要自己实现。另外, 由于文件量可能很大, 生成的过程应该是异步的, 生成好后回调显示预览的方法, 并可以并发操作。

2.4.2 RSS 读取模块

除了文档之外, 另一个重要的信息来源就是 RSS 了。一方面 RSS 来源广, 另一方面更新及时。

因为 RSS 消息格式是基于 XML 的, 对它的解析本身不是问题, 但于 RSS 格式其内部又分为好几种, 每种都使用了不同的标签, 所以处理起来还是稍微有点难度的。考虑到 RSS 已经是一种成熟的标准, 必定有不少开源解析器, 所以这里使用基于 Objective-C 的 MWFeedParser[9] 来读取解析 RSS 链接。

2.4.3 云存储访问与多线程下载模块

在本项目中, iPad 客户端需要对云存储服务平台 Box 进行一些操作。由于

Box 平台的文件访问是通过文件 ID（文件夹也是文件）来进行的，而文件 ID 并不能直观地反映出文件的层级关系，并且高级目标中还要支持其他云平台，而它们的 ID 等又可能与 Box 不同。所以在设计上需要有所讲究，既要能方便访问，又要能后台无关。

考虑到后台存储也可以仅仅是一个简单的 FTP 服务器，而 FTP 的访问是同本地文件访问一样用路径的方式，路径本身已经带了文件的层级关系。于是设计让所有的云存储访问都采用路径的方式。

Box 的根文件夹 ID 为 0，于是可以通过 Box 的 API 获得 0 号文件夹下的子元素的名字和 ID 等信息，这里可以设计一个字典来保存这些信息，实际上它可以看成是一个多叉树，当要进入子文件夹时，通过名称找到 ID，再获取子元素，并保存到树的对应节点上。

2.4.4 文档操作模块

文档操作主要是文档的显示、查看和删除。文档显示常规上都是按网格(Grid)对齐显示，如 3X3 的网格，就能放 9 个文档。当文档超过 9 个时，会出现分页提示，显示总页数以及当前页。然而 iOS 中并没有 Grid 控件，需要自己实现，这里采用开源的 Grid 控件 GMGridView[10] 加快开发过程。同时，如果是正在下载的文档，这里会显示下载进度条。用户通过点击以全屏查看文档，并可以通过发送邮件将文档共享出去。正在下载的文档不可查看。在文档显示视图上，用户拖拽文档时，会出现一个垃圾箱，当用户继续拖拽文档到垃圾箱中去时，文档被删除。正在下载的文档不可删除。

这里操作对象是统一的，不同的文档类型及网格大小不同。所以对于统一的操作可以定义一个基类，不同的文档类型在子类中区分开来。

2.4.5 技术难点

客户端实现的技术难点主要是多种格式的文件预览图的生成、文件的多线程下载及进度条显示、后台（云存储）无关的代码设计。

2.4.5.1 预览图生成

如 2.4.1 中所提到的，iOS 对于 MS Office 系列文档支持并不太好，不能在底层代码上绘制出预览图。然而 MS Office 文档可以在快速查看控制器 (QLViewController) 及网页视图 (UIWebView) 中打开，这是唯一可行的两个

突破口。所以可以创建一个屏幕上不可见的网页视图，并在其中加载要打开的文档，打开之后通过获取视图的绘制句柄（Context），将它的画面绘制在自己的句柄上以创建一个图像，这样一来预览图就生成好了。

对于 PDF 文档可以用 iOS 的 CG 函数直接绘制，且速度很快。

iOS 对 MP4 和 MOV 等 MPEG-4 编码的视频也有较好的支持，可以获取视频中的一帧作为预览图，这里不多阐述。

2.4.5.2 多线程下载及进度条显示

进度条显示本身不是难题，问题就在 iOS 系统中内存有限，经常会有视图的重用，即滑出可见区域的视图将被移除并作为正要滑入可见区域视图，以节约内存。如果简单地将一个指针指向一个正在下载的视图的进度条，那么当这个视图被重用时就会出现进度条错误，因为指针是不可能知道被指的对象是不是被重用了。本项目中可以使用消息机制解决这个问题。

2.4.5.3 后台无关的设计

这个的难点在于，对多种不同的云存储能实现统一形式的访问，且都能支持进度条的更新。项目中采用了多种设计模式以适应这种需求。

2.5 服务端设计

2.5.1 数据库设计

在高级目标下，考虑到用户可以登录、同步 Notes、下载 Email 等，服务器上应有以下数据表。

2.5.1.1 Users

如表 2-2 所示，是用户的一些基本信息。但实际上，公司的网站已经有注册、登录功能，所以可以复用那张表。

表 2-2 Users 表结构

字段	类型	说明
Id	auto_increment	这是表的主键
Name	varchar(255)	用户别名，用于显示

字段	类型	说明
Email	varchar(255)	用户注册邮箱，用于登录
Password	char(32)	MD5 加密后的密码
Status	enum{ACTIVE,INACTIVE}	用户状态
RegisterIP	varchar(15)	用户注册 IP
RegisterDate	date	用户注册时间
LastLoginIP	varchar(15)	上次登录 IP
LastLoginDate	date	上次登录时间

2.5.1.2 Funds

表 2-3 Funds 表结构

字段	类型	说明
Id	auto_increment	表主键
Name	varchar(255)	文件夹名
CreateDate	date	创建时间
OwnerID	int	这是用户 Id 的外键

2.5.1.3 Notes

表 2-4 Notes 表结构

字段	类型	说明
Id	auto_increment	表主键
Title	varchar(512)	标题，限制为 512 个字符
Content	text	内容，长度无限制
Date	date	最后修改过的日期
FundID	int	外键，Notes 所属的 Fund 的 Id

2.5.1.4 Emails

项目的 Email 功能要求用户可以用自己的注册邮箱向公司一个特定的邮箱以特定的规则发邮件，然后服务器上的守护进程会监视并处理这些邮件，将其

对应到用户指定的 Fund 下，并保存其附件。所以这里需要两张表，一张保存邮件本身，一张保存附件信息。

表 2-5 Email 表结构

字段	类型	说明
Id	auto_increment	表主键
Subject	varchar(1024)	邮件标题
Body	text	邮件正文
MIME	enum{PLAIN,HTML}	邮件类型，纯文本或富文本 HTML
From	varchar(255)	发送者邮箱地址
Date	date	发送时间
FundID	int	外键，邮件所属的 Fund 的 Id
Attachment	int	邮件附件数量

表 2-6 Attachment 表结构

字段	类型	说明
Id	auto_increment	表主键
Name	varchar(1024)	附件名
Size	int	附件大小，单位字节
Type	varchar(10)	附件类型，扩展名
Path	text	附件保存完整路径
EmailID	int	外键，附件所属的邮件 Id

2.5.2 接口设计

服务器需要完成以下工作：邮件的抓取及存储，附件的抓取及存储，用户登录验证，Notes 的同步与更新等。其中用户验证是比较难的一部分，现有的基于 cookie 的登录机制仅适用于浏览器，而 OAuth 对于这个小系统又太复杂。为了简化操作，基于 OAuth 的原理及 RESTful 的思想，要设计自己的一套验证机制。

术语定义：

- **Token**。服务端返回的一段随机字符串，用于客户端验证。
- **Expire**。Token 的过期时间。
- **Sign**。客户端对服务端的操作及操作参数，和 **Token** 值一起通过一定的运算得到的一个 **HASH** 值，作为服务端的权限验证。

2.5.2.1 登录接口

如表 2-7 客户端将用户填写的用户名、密码以 **POST** 方式发送到服务器，服务器验证通过后会返回一个 **Token**，并设置好它的 **Expire**。**Token** 会保存在服务器端的数据库中，并与当前用户关联，直到过期。客户端接收 **Token** 后可以将其保存到磁盘上，或内存中，以供后面的操作使用。

为了保证用户名、密码，及 **Token** 不会被人恶意抓取，整个过程以 **HTTPS** 方式连接。

表 2-7 登录接口设计

登入接口	https://hostname/login
方式	POST
参数说明	username 用户名; password 密码
返回值	输入正确: { "status": "OK", "token": "...", "expire": 1337001234, "user_id": 1234 } 输入错误: { "status": "Error", "description": "invalid username or password" }

2.5.2.2 同步

用户登录到系统后，首先客户端会同步本地 **Fund** 信息，然后当用户点开一个 **Fund** 时，开始同步当前 **Fund** 的 **Notes** 及邮件列表。同步操作采用 **RESTful** 规范，同时会带上一个 **Sign**，作用服务端对此操作的权限认证。登录之后，所有的操作中都不会出现 **Token**，它是作为一种保密信息，只有客户端和服务端知道。**Sign** 其实隐含到 **Token** 的信息，但却不能还原出来，达到加强安全的目的。

表 2-8 同步接口设计

Fund 接口	https://hostname/funds/<user_id>/<sign>
----------------	---

方式	GET
参数说明	user_id 用户 ID; sign 链接签名
返回值	输入正确: <pre>{ "status":"OK", "funds":[{"id":1,"name":"fund1","date":1337001234}, {"id":2,"name":"fund2","date":1337003333}] }</pre> 输入错误: <pre>{"status":"Error","description":"invalid sign"}</pre>
Notes 接口	https://hostname/notes/<fund_id>/<user_id>/<sign>
方式	GET
参数说明	fund_id 文件夹 ID; user_id 用户 ID; sign 链接签名
返回值	输入正确: <pre>{ "status":"OK", "notes":[{"id":1,"title":"note1","content":"test","date":1337001234}, {"id":2,"title":"note2","content":"test","date":1337003333}] }</pre> 输入错误: <pre>{"status":"Error","description":"invalid sign"}</pre>

当然另外还有一些添加、删除的接口，这里不进行详细列表。

2.5.3 Email 处理模块

在高级目标中，用户可以在邮件主题前用“[]”将邮件要放置的文件夹名指定好，然后像发送普通邮件一样发送到公司的特定邮箱。服务端需要运行一个守护进程，它监视公司邮箱的变化，一旦收到新的邮件，它通过发件人找到用

户 ID，通过邮件主题找到文件夹名，结合用户 ID 就可以找到唯一的一个文件夹 ID，然后在 Email 及 Attachment 表中插入数据。

2.5.4 设计亮点

本项目中接口的授权设计基于 OAuth 的思想，同时也参照了 Box 云存储服务的授权过程，并对其进行了一定程度的优化，使用接口更加安全。这种安全性是设计上的一大亮点。

2.5.4.1 Box 的实现方式

下面就来看看 Box 的实现方式，并分析为什么这种实现不安全。

术语定义：

- **API_KEY**。所有需要访问 Box 资源的终端都被看作是一个应用，它可以是一个网站，一个移动设备，或是普通 PC 软件。首先开发者需要有自己的帐号下创建一个应用，同时获取 Box 随机生成的 API_KEY，这样 Box 就可以知道是谁在访问。
- **Ticket**。请求授权前，应用需要将自己的 API_KEY 作为参数传给 Box，然后 Box 会返回一个票据（Ticket），用于后面的授权。
- **AUTH_TOKEN**。真正用于访问资源的授权码。

首先在开发者页面（<https://www.box.com/developers/services>）上创建一个应用，生成一个 API_KEY：b4ejygdxfmxyz5s9c09ivc4lkv9zmuk。这是准备工作，然后就可以跟踪 Box 的 SDK 看它如何进行授权的。

表 2-9 Box 授权过程

步骤	说明	Box API
1	获取 Ticket	<code>https://www.box.net/api/1.0/rest?action=get_ticket&api_key=<api_key></code>
2	获取 AUTH_TOKEN 并保存	<code>https://www.box.net/api/1.0/rest?action=get_auth_token&api_key=<api_key>&ticket=<ticket></code>

步骤	说明	Box API
3	引导到 Box 页面，让用户授权	<a href="https://www.box.net/api/1.0/auth/<ticket>">https://www.box.net/api/1.0/auth/<ticket>
4	使用授权过的 AUTH_TOKEN 进行资源访问	<ul style="list-style-type: none"> ● 列出文档树： <a href="https://www.box.net/api/1.0/rest?action=get_account_tree&api_key=<api_key>&auth_token=<auth_token>&folder_id=0&params[]=nozip">https://www.box.net/api/1.0/rest?action=get_account_tree&api_key=<api_key>&auth_token=<auth_token>&folder_id=0&params[]=nozip ● 下载文档： <a href="https://www.box.net/api/1.0/download/<auth_token>/<file_id>">https://www.box.net/api/1.0/download/<auth_token>/<file_id> ● 上传文档： <a href="https://upload.box.net/api/upload/<auth_token>/<folder_id>">https://upload.box.net/api/upload/<auth_token>/<folder_id>

从上面的授权过程可以很清楚地看到，作为可以访问用户资源的机密信息 AUTH_TOKEN 直接暴露在链接中，这就让别有用心的人有机可趁。因为可以通过抓包工具抓取用户所在网关的所有访问请求，虽然 Box 的授权过程是安全连接，但链接中的字符是不会加密的，所以可以很轻松地获取到用户帐户的访问权限，并可以下载用户的私密文件。

2.5.4.2 项目中的解决方案

上一节中指出，不安全主要来自机密信息 AUTH_TOKEN 的暴露，但是不作为参数放在请求中就无法进行授权，这是个矛盾，但如果将这种信息加密后再做为参数就可以解决这个问题。

客户端在安全链接中，将用户信息以 POST 方式发送到服务器，服务器验证过后再在安全链接中返回 Token，这个过程中 Token 是安全的，且只有客户端和服务器知道。在以后的接口调中，Token 不会作为参数传入链接，取而代之的是 Sign:

$$\text{Sign} = \text{MD5}(\text{parameters} + \text{Token})$$

服务端在接收请求时，通过同样的算法验证 Sign，再做对应的操作，返回

数据。即使有人恶意抓取链接，也无法还原 Token，所以也只能做被抓的链接对应的操作。是访问的整个过程。

2.6 本章小结

本章介绍了项目的目标及项目实现前期的一些技术框架和方案，并分别描述了客户端及服务端的一些设计细节，为增强了项目成功的信心，为后面项目的具体实现打下了基础。

第3章 项目实施

3.1 客户端实现

客户端功能上的细节较多，有些逻辑判断也比较复杂，下面就几个主要功能简单说明实现过程。

3.1.1 预览图生成

生成 PDF 文件的预览图可以用 `CGPDFDocumentCreateWithURL()` 函数来生成，操作简单。而对于 MS Office 系列文件，可以将文件在一个 `UIWebView` 中打开，成功打开后，让 `UIWebView` 绘制在自己创建的 `Context` 并然后生成图像。

实现片断如下：

```
- (void)loadWebView
{
    UIWebView *webView = ...;
    webView.delegate = self;
    [webView loadRequest:...]; // 加载文件
}

- (void)webViewDidFinishLoad:(UIWebView *)webView // 成功加载
{
    .....

    UIGraphicsBeginImageContext(预览图大小); // 创建自己的 Context
    CGContextRef context = UIGraphicsGetCurrentContext();
    [webView.layer renderInContext:context]; // 绘制到这个 Context
    预览图 = UIGraphicsGetImageFromCurrentImageContext(); // 生成预览图
    UIGraphicsEndImageContext();

    .....
}
```

3.1.2 云存储访问

云存储访问的实现是项目的关键，因为透过云存储以扩展 iPad 无限的存储能力是本项目的要点。考虑到需要对多个云存储服务提供支持，在代码设计上就需要一些巧妙的模式。

在代码实现中，与云资源访问相关最重要的类是 `HTFileDownloader`，它是一个单例，以及它的委托 `HTFileDownloaderDelegate`，它是一个协议，主要提供以下接口：

```
- (NSString*)home;
- (NSString*)home;
- (BOOL)cdTo:(NSString*)path;
- (void)cdUp;
- (void)logout;
- (void)loginWithCompletionHandle:(void(^)(void))handle;
- (NSArray*)itemsInPath:(NSString*)path;
- (void)itemsInPath:(NSString*)path
  asyncWithBlock:(void(^)(NSArray *items))block;
- (NSArray*)itemsInPathForceReload:(NSString*)path;
- (BOOL)isLogin;
- (void)login;
- (void)downloadFile:(NSString*)path
                  toPath:(NSString*)localPath
  withProgressHandler:(id<HTDownloadProgressDelegate>)handler;
```

这里定义了项目可能用到的所有文件操作接口，当然实际上用不到全部，具体的云存储只要实现部分接口即可工作。

这种设计的优点是，即提供一个全局访问点，又可以切换具体实现。`HTFileDownloader` 的文件操作都回转发给 `delegate` 执行，只要设置一个新的 `delegate` 便可以使用新的云服务。

所有的云存储在这里都看作是一个 `Service`，可以调用 `HTFileServiceFactory` 的类方法创建一个 `Service`。下面就 `Box`，`Dropbox`，`SkyDrive` 三种云服务来看看具体实现过程。

3.1.2.1 Box

`Box` 的 SDK 中文件类 `BoxFile` 与文件夹类 `BoxFolder` 都继承自 `BoxObject`，用一个整型 `objectId` 来区分不同的文件与文件夹，根文件夹的 `objectId` 为 0。`Box` 对文件的所用操作都基于文件 ID。

定义 HTFileServiceBoxnet 类，服从 HTFileDownloaderDelegate 协议，它内部维护一个变长的字典 NSMutableDictinory *_fileTree，用以支持统一的路径形式的访问，即，把路径形式的访问转换为 ID。这个字典主要结构如下：

```
{
    "id":0;
    "type":"folder"; // 根目录
    "items":
    {
        "folder1":
        {
            "id":12494341;
            "type":"folder";
            "items":
            {
                "file1":
                {
                    "id":2222222;
                    "type":"file";
                    "url":"https://www.box.com/download/....doc";
                }
                "file2":
                {
                    "id":1111111;
                    "type":"file";
                    "url":"https://...";
                }
            }
        }
        "file3":
        {
            "id":2232323;
            "type":"file";
            "url":"https://....";
        }
    }
}
```

id 是个 NSNumber，type 是 NSString，而 items 又是变长字典 NSMutableDictionary，当要访问 /folder1/file1 时，首先将访问的路径分为三个节点：“/”，“folder1”，“file1”。“/”节点就是_fileTree 本身，从它的 items 中找到“folder1”键对应的值，这样就进入“folder1”节点，再从当前节点的 items 中找到“file1”键对应的值，这样就可以找到 file1 文件对应的 ID，然后再进行其他操作。

HTFileServiceBoxnet 对象初始化时，只有根目录是已知的，其它节点将在用户的不断访问过程中构造并保存。初始化代码片断如下：

```
- (id)init
{
    self = [super init];
    if (self){
        _fileTree = [[NSMutableDictionary alloc]
initWithObjectsAndKeys:
                        [NSNumber numberWithInt:0], @"id",
                        @"folder", @"type", nil];
    }
    return self;
}
```

文档树的访问主要代码实现如下：

```
// 给定完整路径，返回路径对应的节点
- (NSDictionary*)getNodeinPath:(NSString *)path
{
    if (!path || [path isEqualToString:@""])
        return nil;

    NSArray *paths = [path pathComponents]; // 拆分路径为部件
    NSDictionary *current = _fileTree; // 从根目录开始
    for (NSString *folder in paths) { // 遍历路径部件
        if ([folder isEqualToString:@" /"]) {
            continue; // 跳过根目录
        }
        NSMutableDictionary *items = [current valueForKey:@"items"];
        if (!items) { // 如果当前还有没缓存子目录，构造子目录
```

```

        items = [self buildFileMapTree:[current valueForKey:@"id"]];
        if (items)
            /*
             * 构造好后，保存到_fileTree，方便下次访问
             */
            [current setValue:items forKey:@"items"];
        else {
            current = nil;
            break;
        }
    }
    current = [items valueForKey:folder]; // 进入下一级节点
    if (!current) {
        break; // 没有这个目录，退出
    }
}
return current;
}

// 给定完整路径，返回文件的 ID
- (NSNumber*)getFileIDinPath:(NSString *)path
{
    NSDictionary *current = [self getNodeinPath:path]; // 获取节点
    return [current valueForKey:@"id"];
}

```

在上面 getNodeinPath:方法中，如何 items 为 nil，则需要构造目录的子元素，将文件夹的 ID 作参数传入 buildFileMapTree:方法即返回文件夹中的所有子元素。具体实现如下：

```

- (NSMutableDictionary*)buildFileMapTree:(NSNumber *)folderID
{
    .....

    BoxFolder * folderModel = 同步获取文件夹信息(folderID);
    if (成功执行) {
        int count = [folderModel.objectsInFolder count];
        NSMutableDictionary *items = [NSMutableDictionary

```

```

dictionaryWithCapacity:count]; // 生成空节点

    for (BoxObject *boxobj in folderModel.objectsInFolder) {
        BOOL isFile = [boxobj isKindOfClass:[BoxFile class]];
        NSDictionary *dic = nil;
        if (isFile) { // 如果是文件
            NSString *fileURL = 生成文件完整链接;
            dic = [NSMutableDictionary dictionaryWithObjectsAndKeys:
                boxobj.objectId,@"id",
                @"file",@"type",
                [NSURL URLWithString:fileURL],@"url", nil];
        }
        else { // 如果是文件夹
            dic = [NSMutableDictionary dictionaryWithObjectsAndKeys:
                boxobj.objectId,@"id",
                @"folder",@"type",
                ((BoxFolder*)boxobj).fileCount,@"count", nil];
        }
        [items setValue:dic forKey:boxobj.objectName]; // 保存子节点
    }
    return items; // 返回构造好的节点
}
else
    return nil;
}

```

其中 SDK 中关键类为 BoxNetworkOperation，它是所有操作的基类，也是 NSOperation 的子类，所以可以加到 NSOperationQueue 中以实现异步操作，但它本身是同步操作。BoxDownloadOperation 是 BoxNetworkOperation 的子类，所以 Box 实现的-(void) downloadFile:toPath:withProgressHandler:方法为同步的，这会影响到后续云存储访问代码的实现。下载实现的代码片断如下：

```

- (void)downloadFile:(NSString *)path
    toPath:(NSString *)localPath
withProgressHandler:(id<HTDownloadProgressDelegate>)handler
{
    NSString *fileID = [self getFileIDinPath:path];
    NSString *fileName = [[path pathComponents] lastObject];

```

```

HTDownloaderBoxnet *db = [[HTDownloaderBoxnet alloc] init];
db.delegate = handler;

NSFileManager *fm = [NSFileManager defaultManager];
NSString *to = [localPath stringByAppendingPathComponent:fileName];
NSString *or = to;
int count = 1;
// 这里主要用来判断重名文件是否存在, 如果存在就给它加上编号, 即
// file.doc 重命名为 file(1).doc, 若还存在, 则为 file(2).doc, 依此类推
while ([fm fileExistsAtPath:or]) {          // 如果文件存在
    NSString *ext = [to pathExtension];      // 保存后缀名
    // 去掉后缀名
    NSString *tmp = [to stringByDeletingPathExtension];
    // 加上编号及后缀名
    or = [tmp stringByAppendingFormat:@"%d).%@", count++, ext];
}
to = or;    // 一个不会重名的完整路径

BoxDownloadOperation *download =
    [BoxDownloadOperation operationForFileID:fileID
                                     toPath:to];

download.delegate = db;
[download start];    // 开始同步下载, 其中 delegate 用于通知外面下载进度
[db release];
}

```

3.1.2.2 Dropbox

Dropbox 文件访问的后台实现是跟本地文件系统一样, 使用的是路径。所以它的代码实现要简单得多。

跟 Box 类似, 定义 HTFileServiceDropbox 类服从 HTFileDownloaderDelegate, 然后实现接口方法。Dropbox 的 SDK 中关键类是 DBSession 和 DBRestClient, 前者用于授权及保存授权信息, 后者用于实际 API 调用。

需要注意的是, Box 中的下载操作是同步, 而 Dropbox 是异步的, 为尽量减少原有代码改动, 将 Dropbox 也实现为同步方式。实现方法为将 HTFileServiceDropbox 继承自 NSOperation, 并在异步操作后进入消息循环, 直

到异步操作完成才退出。现实代码片断如下：

```
- (void)downloadFile:(NSString*)path
    toPath:(NSString*)localPath
withProgressHandler:(id<HTDownloadProgressDelegate>)handler
{
    NSString *fileName = path.lastPathComponent;

    // 检查文件重名，并加上编号，同 Box

    [downloadDelegateQueue setObject:handler
                                forKey:to]; // 保存 handler，以便更新下载进度
    [handler downloadFileStart:to]; // 通知下载开始

    [self.restClient loadFile:path
                    intoPath:to]; // 开始异步下载
    CFRunLoopRun(); // 当前线程进入消息循环，阻塞以实现同步
}
```

然后在 DBRestClient 的委托方法中退出消息循环，如在成功下载后：

```
- (void)restClient:(DBRestClient*)client
    loadedFile:(NSString*)destPath
{
    id<HTDownloadProgressDelegate> handler = [downloadDelegateQueue
objectForKey:destPath];
    [handler downloadFile:destPath
                    complete:YES]; // 通知下载完成
    [downloadDelegateQueue removeObjectForKey:destPath]; // 移除 handler

    CFRunLoopStop(CFRunLoopGetCurrent()); // 退出消息循环
}
```

这样一来，异步操作就变成同步操作了。

3.1.2.3 SkyDrive

同 Box 一样，SkyDrive 的文件操作也是基于 ID 的，不同的是 SkyDrive 的文件 ID 是字符串，而且更为复杂。它的根目录为/me/skydrive，代码实现上有着与 Box 差不多的思路，同样需要维护一个_fileTree，也需要在第一次时构造节点并保存。同时 SkyDrive 的下载也是异步的，需要像 Dropbox 一样将其转换为同

步操作。所以可以认为 SkyDrive 的实现是 Box 与 Dropbox 实现的结合。

3.1.3 多线程下载及进度条

在 iPad 这种小型移动设备上，内存是种珍贵的资源，iOS 在底层上对内存管理进行了优化，极在限度上减少了内存占用率，同时它要求开发者们养成良好习惯，不用的视图和缓存应当在内存不足时马上释放掉。在 2.4.4 中提到的优秀的开源控件 GMGridView 即采用了内存节约型的设计。当文档过多时，视图会横向形成滚动，而当滚动页较多时（超过 3 页），那么已经滚出可见区域较远的那部分 Grid 可以重用到即将滚入的 Grid 上。所以基本上只要 3 页的内存就可以显示任意多页。

为了显示进度条，Grid 中添加了 UIProgressView 子视图，当然将这个视图的指针传给下载的 delegate，当进度发生改变时更新进度条位置是可以的，问题就在于分页过多出现 Grid 重用就麻烦了。前面是在下载，但后面可能是已经下好的文件，当用户滚动到后面时，前面的 Grid 被重用，那么前面文件的进度条就显示到后面已经下好的文件上了。

这里 iOS 内部的消息机制就派上用场了。消息机制的优点是：消息发送者不需要知道谁在接收消息；消息接收者也不需要知道谁在发送消息。从而达到一定程度上的解耦合。无论是什么云存储，在下载文件过程中进度改变时发送以文件名命名的消息，同时每个 Grid 接收自己显示的文件名对应的消息，这样即使页数多，用户滚动时，被重用的 Grid 会因为监听了新的消息而不会被影响到。

定义 HTDownloadProgressDelegate 协议，它提供三个方法（见表 3-1），在不同的云存储访问实现中需要添加自己的类服从这个协议。主要目的是为了提供一个标准化的进度通知接口，因为不同的云存储可能有自己不同的实现方式。这样才能在下载时传入统一的参数，即 HTFileDownloadDelegate 中

```
- (void)downloadFile:(NSString *)path
                toPath:(NSString *)localPath
withProgressHandler:(id<HTDownloadProgressDelegate>)handler
```

方法的 handler 参数。

表 3-1 HTDownloadProgressDelegate 协议

方法名	说明
-----	----

方法名	說明
-(void)downloadFileStart:(NSString*)path;	文件開始下載通知，參數為文件本地路徑
-(void)downloadFile:(NSString*)path progress:(float)ratio;	文件下載進度更新
-(void)downloadFile:(NSString*)path complete:(BOOL)success;	文件下載中止，參數 success 說明是否完成下載

定義 HTDownloadOperation 繼承自 NSOperation 類，並服從 HTDownloadProgressDelegate 協議，使得它既可以添加為多線程任務，又可以管理自己任務的下載進度。

```
...
// 下載云存儲文件 /folder/remotefile1 到本地文件 /localfolder/localfile1
NSString *remoteFile = @"/folder/remote_file1";
NSString *localFile = @"/localfolder/localfile1";

HTDownloadOperation *download = [HTDownloadOperation
                                operationWithRemotePath:remoteFile
                                andLocalPath:localFile];

[[HTFileDownloader sharedDownloader] addDownloadOperation:download];
...
```

HTDownloadOperation 覆寫 NSOperation 的 main 等方法，調用 HTFileDownloader 的方法下載文件。

```
- (void)main
{
    if (self.isCancelled)
        return;

    HTFileDownloader *fd = [HTFileDownloader sharedDownloader];
    [fd downloadFile:self.remotePath          // 同步方法
     toFolder:self.localPath
     withProgressHandler:self];
}
```

並在協議方法中發送消息。

```
#pragma mark - HTDownload Delegate

- (void)downloadFileStart:(NSString*)path
{
    .....

    NSNotificationCenter *center = [NSNotificationCenter
defaultCenter];
    // 在主线程中发送消息名为本地文件所在目录路径的消息
    dispatch_async(dispatch_get_main_queue(), ^{
        [center postNotificationName:self.localPath
                        object:[NSDictionary
dictionaryWithObjectsAndKeys:
                                path.lastPathComponent,@"file",
                                [NSNumber
numberWithFloat:0.0],@"ratio",nil]];
    });
}

- (void)downloadFile:(NSString*)path progress:(float)ratio
{
    .....

    dispatch_async(dispatch_get_main_queue(), ^{
        NSNotificationCenter *center = [NSNotificationCenter
defaultCenter];
        [center postNotificationName:self.localPath
                        object:[NSDictionary
dictionaryWithObjectsAndKeys:
                                path.lastPathComponent,@"file",
                                [NSNumber
numberWithFloat:ratio],@"ratio",nil]];
    });
}

- (void)downloadFile:(NSString*)path complete:(BOOL)success
{
```

```
if (success) {
    .....

    dispatch_async(dispatch_get_main_queue(), ^{
        NotificationCenter *center = [NSNotificationCenter
defaultCenter];
        [center postNotificationName:self.localPath
                                object:[NSDictionary
dictionaryWithObjectsAndKeys:
                                path.lastPathComponent,@"file",
                                [NSNumber
numberWithFloat:1.1],@"ratio",nil]];
        [center postNotificationName:@"LibTableViewNeedReload"
                                object:nil];
    });
    // a more than 1.0 value to diff with progress 1.0
}
else {
    dispatch_async(dispatch_get_main_queue(), ^{
        NotificationCenter *center = [NSNotificationCenter
defaultCenter];
        [center postNotificationName:self.localPath
                                object:[NSDictionary
dictionaryWithObjectsAndKeys:
                                path.lastPathComponent,@"file",
                                [NSNumber
numberWithFloat:-0.1],@"ratio",nil]];
    });
    // a less than 0.0 value to indicate a failure
}
[self finish];
}
```

注意上面的代码中使用了 `dispatch_async` 函数。因为消息接收者（Grid）需要更新自己的视图，所以必须在主线程上，`dispatch_get_main_queue` 返回主线程操作队列。不在主线程上发送这些消息会产生不可预料的后果！

定义 `HTFileBrowserViewCell` 类作为显示文件预览图及文件名的 Grid，以及

HTFileBrowserController 作为控制器。HTFileBrowserController 一次只显示一个本地文件夹，它接收以自己显示的文件夹路径为名称的所有消息。在控制器中维护一个变长字典 docCells，它保存文件名与实际显示的 Grid 的对应关系，控制器接收消息，并通过这个字典找到相应的 Grid 更新进度条。

```
.....
// 当控制器的显示的路径改变时，监听新的消息
- (void)setLocalPath:(NSString *)localPath
{
    .....
    NotificationCenter *center = [NSNotificationCenter
defaultCenter];
    [center removeObserver:self];    // 不再监听之前的消息
    [center addObserver:self        // 监听新路径的消息
        selector:@selector(monitorFileDownload:)
        name:localPath
        object:nil];

    .....
}

.....
- (void)monitorFileDownload:(NSNotification*)notification
{
    // 获取消息信息
    NSDictionary *info = (NSDictionary*)notification.object;
    // 找出进度更新的文件名
    NSString *file = [info objectForKey:@"file"];

    if (找到 Grid) {
        根据以下情况更新视图：
        进度为 0~1 时，更新进度条；
        进度小于 0 时，此文件下载失败；
        进度大于 1 时，下载已经完成；
    }
    else {
        创建一个新的 Grid，并保存到 docCells 字典中；
    }
}
```

```

}
}
    
```

整个流程可以用图 3-1 来表示：

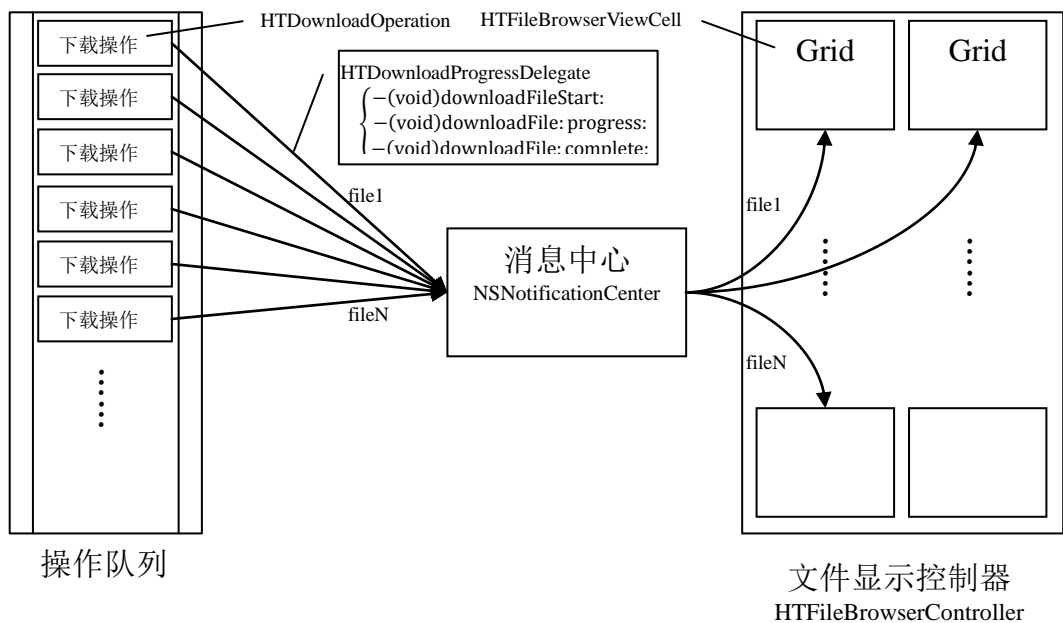


图 3-1 多线程下载及进度条显示实现机理

3.2 服务端实现

3.2.1 登录验证

服务器对用户的登录包括用户名密码的校验其 Token 的生成过程。主要实现如下：

```

function login($username,$password) {
    $password = md5($password);
    $id = 查找用户 ID;
    if (匹配成功) {
        $token = 随机生成 Token;
        $expire = 当前时间加一个小时;
        保存$token, $expire 及对应的用户$id 以供后面验证;
        返回信息到客户端
    } else {
        返回错误信息到客户端;
    }
}
    
```

```
}
```

3.2.2 同步 Fund

同步过程不仅要验证签名的正确性，还要验证操作的合法性，即 Fund 的 ID 是不是当前请求的用户的 Fund。基本过程如下：

```
// 同步 Fund
function sync_fund($userid,$sign) {
    $token = 查找用户的 Token($uesrid);
    $thesign = md5($token+参数);
    if ($thesign === $sign) {    // 验证通过
        $funds = 查找用户所有的 Fund($userid);
        返回数据到客户端;
    } else {
        返回未授权错误信息;
    }
}

// 删除 Fund
function del_fund($fundid,$userid,$sign) {
    验证$sign;
    $theuser = 查找 Fund 所属的用户($fundid);
    if ($theuser === $userid) {
        删除 Fund 记录;
        返回成功信息;
    } else {
        返回非法操作信息;
    }
}

function add_fund($fundname,$userid,$sign) {
    .....
}
```

3.2.3 同步 Notes

同步 Notes 与同步 Fund 的操作基本一样，同样要验证签名及所属关系，比 Fund 多了一层判断，避免误操作他人的数据。


```
// 同步 Notes
function sync_notes($fundid,$userid,$sign) {
    验证$sign;
    if ($fundid 属于 $userid) {
        返回此 Fund 下的 Notes;
    } else {
        返回非法操作信息;
    }
}

// 删除 Notes
function del_notes($noteid,$fundid,$userid,$sign) {
    验证$sign;
    if ($fundid 属于 $userid) {
        if ($noteid 属于 $fundid) {
            返回成功信息;
        }
    }
    返回非法操作;
}

.....
```

3.2.4 Email 抓取

服务器端运行着的守护进程可以通过 POP3 查询邮件服务器的邮件变化,当收到新的邮件时便触发一个操作。如 2.5.3 节中提到的,用户需要按规定的格式设置邮件主题,即将邮件要放置的 Fund 名用“[]”括起来,如表 3-2 所示。

表 3-2 邮件发送格式

主题:	[Fund1] This is Email Subject
附件:	1.pdf; 2.doc;
正文:	This is Email Body, Test content.

大致处理过程如下:

```
.....
```

```
Email email = 收到新邮件;
String send = email.getFrom();
String subject = email.getSubject();
String fundname = 正则匹配找出 Fund(subject);

int userid = 通过注册邮箱查询用户 ID(send);
if (找到) {
    ArrayList<Fund> funds = 通过用户 ID 找到所有的 Fund(userid);
    Fund tofund = null;
    for (Fund fund:funds) {
        if (fund.name && fund.name.equals(fundname)) {
            tofund = fund;
            break;
        }
    }
    if (fund) {          // 找到对应的 Fund
        保存 email 及所属的 fund.id;
    }
}
else
    不做处理，退出;
```

3.3 本章小结

本章以代码或伪代码的方式分别讲述了客户端与服务端一些主要功能的实现细节，如何攻克难关并最终成功实现的方法。

在客户端的设计上用到了委托、单例、桥接、观察者等多种设计模式，降低了代码的维护成本，并且更加可读、易于理解。

服务端的代码实现较为常规，但考虑到其安全性，验证判断逻辑也有一定难度。

第4章 项目成果

项目成果分为两块内容，一个 iPad 应用程序，另一个是服务器代码。其中基本目标下的 iPad 应用已经上线 Apple Store，所有用户均可免费下载；服务器端还未完全实现。

4.1 iPad 应用

目前拥有基本目标下的功能的 iPad 应用已经开始在 Apple Store 上销售。这里分别看看它的五个主要交互界面并加以说明。

4.1.1 Home

如图 4-1，在 Home 标签下用户可以看到本地所有的 Fund。点击进入一个 Fund，左边以列表形式列出与当前 Fund 相关的 Notes，右边是已经下载的文件。

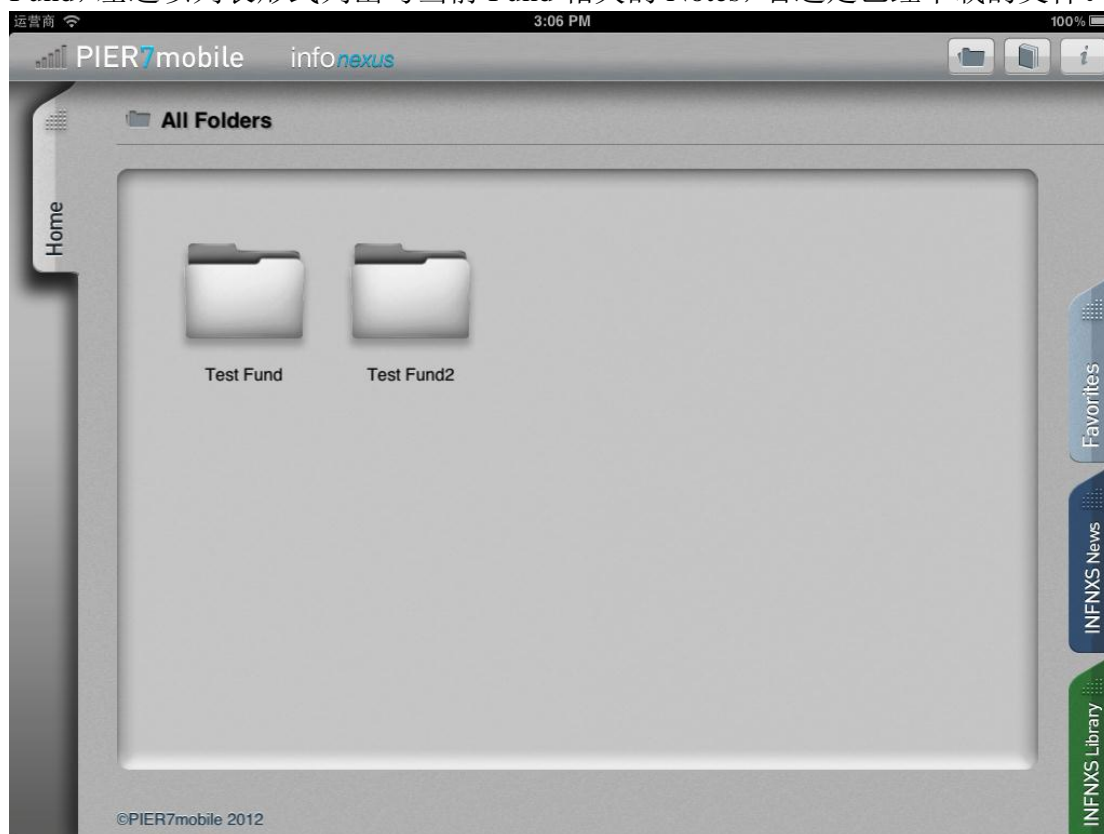


图 4-1 Home 界面的所有 Fund 视图

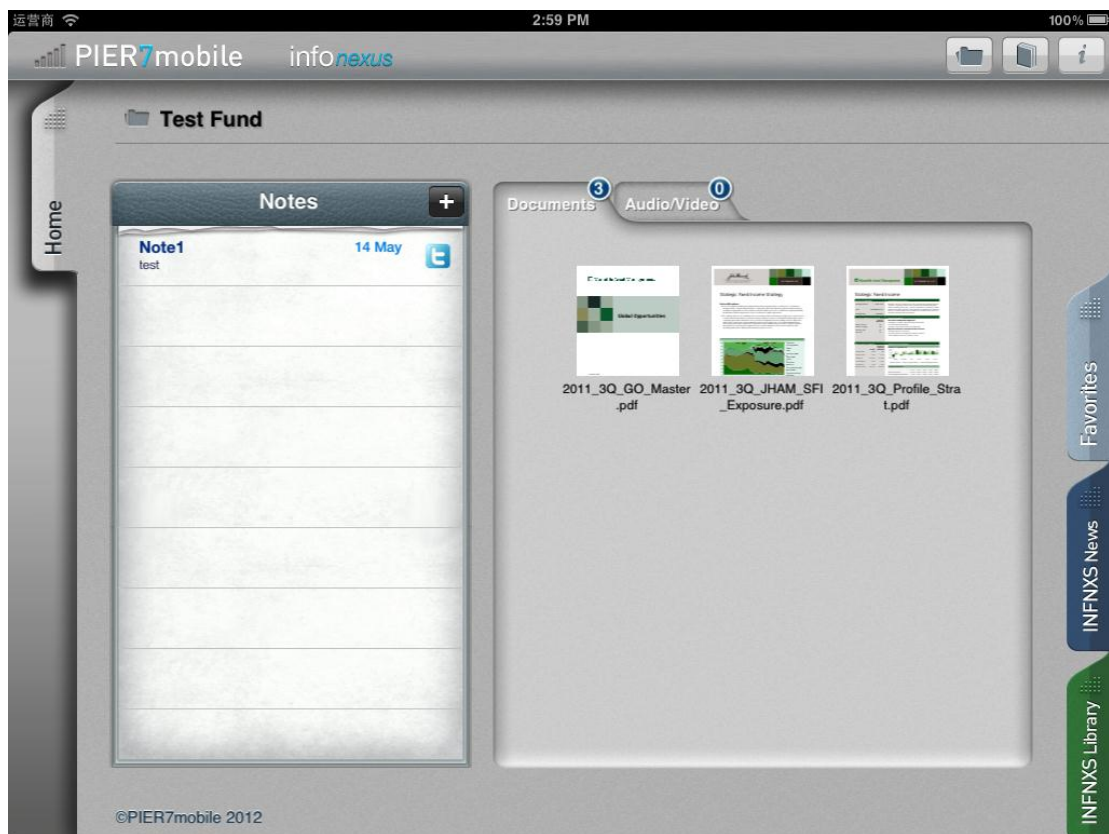


图 4-2 Home 界面的单个 Fund 视图

在左边用户可以点击“+”新建 Notes，点击某个 Notes 可以查看详情，并可以编辑。右边点击文档可以全屏查看（如图 4-3），长按拖拽可以进行删除和放入 Favorite。



图 4-3 文件查看视图

4.1.2 Favorite

此界面功能较为单一，只显示一个用户拖拽到这里的常用文件，双击可以打开查看。如图 4-4。

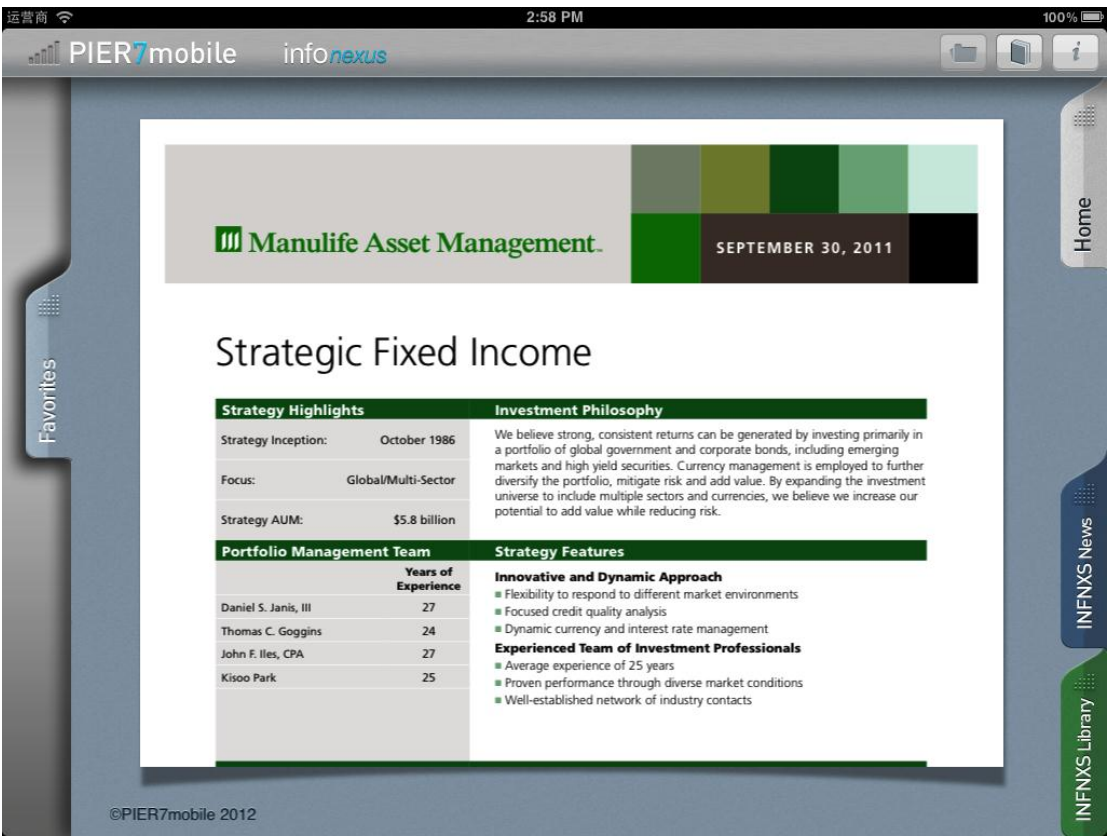



图 4-4 Favorite 界面

4.1.3 News

News 内容全部来自定制化的 RSS Feeds，用户可以选择不同的 RSS 源以查看感兴趣的信息。



图 4-5 New 界面

此界面分为两列，右边为 RSS 内容摘要，点击一个后，左边显示内容的来源网页。点击按钮可以在新的界面中全屏查看该网页。

4.1.4 Library

这里是云存储中的文件列表，用户首先需要登录才能看到。在这个版本中只支持 Box 云服务。

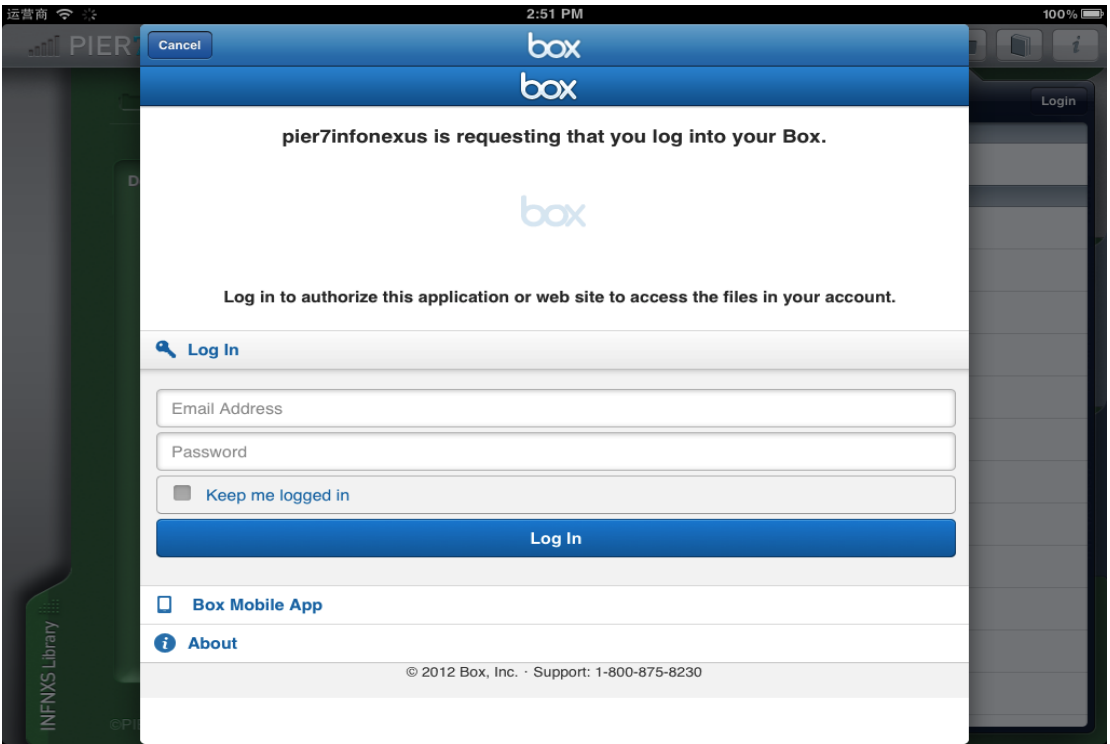


图 4-6 Box 的登录界面

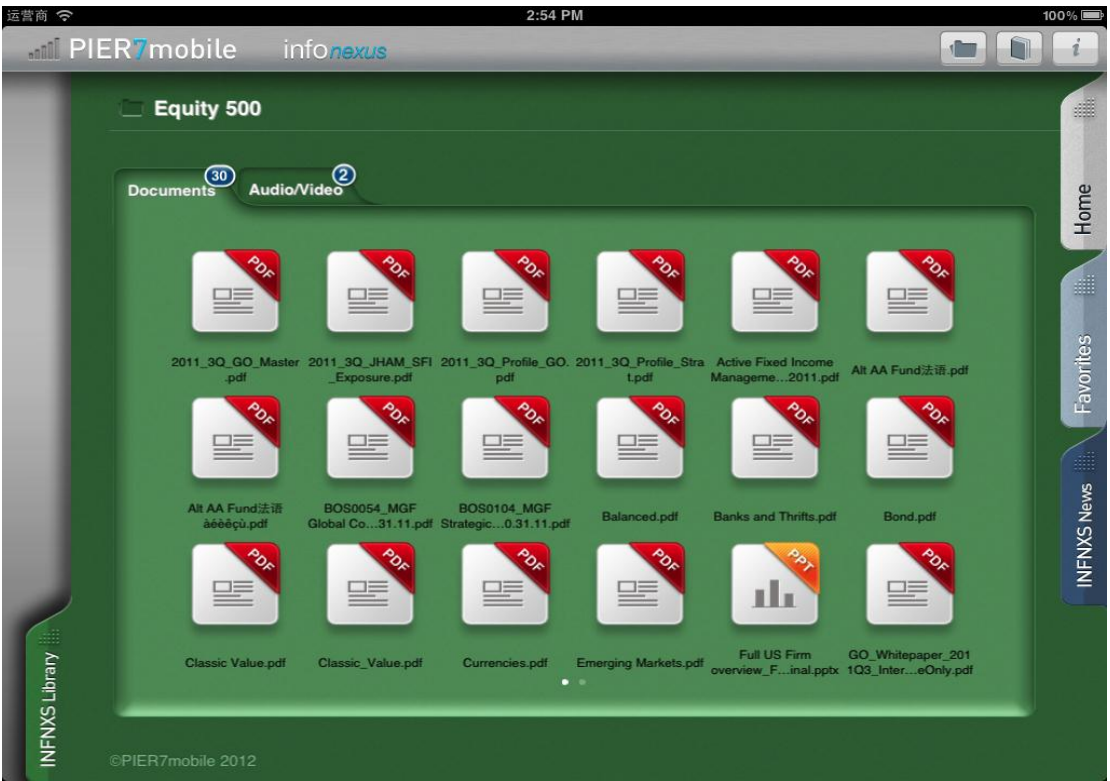


图 4-7 登录后选择 Equity 500 文件夹的视图

4.1.5 Management

在这个界面中，用户可以以简单的拖拽方式下载云存储的文件到本地，前提是在 Library 中登录。



图 4-8 文件下载

4.2 服务端代码

由于项目进度原因，在没有支持服务端的情况下发布了 iPad 应用的基本目标版。目前服务端只实现了 Email 的抓取实现，其他接口还在进一步的优化设计和调试中。

4.3 本章小结

本章以应用截图的方式讲解了 iPad 应用的几个主要功能界面，限于篇幅没有作详尽的功能说明，更多内容请参考应用的使用说明书。由于服务端实现没有完全实现，目前没有可展示的成果。

第5章 项目展望

5.1 项目总结

[单击此处输入论文正文]

5.2 前景展望

参考文献

- [1] 纵观平板电脑发展史. 中国电脑教育报. 2010-11-15, 第 A03 版.
- [2] 云计算. 维基百科.
<http://zh.wikipedia.org/wiki/%E9%9B%B2%E7%AB%AF%E9%81%8B%E7%AE%97>.
- [3] 云存储. 百度百科. <http://baike.baidu.com/view/2044736.htm>.
- [4] iOS. 维基百科. <http://zh.wikipedia.org/wiki/IOS>.
- [5] OAuth. 维基百科. <http://zh.wikipedia.org/wiki/OAuth>.
- [6] Carlo Chung. Objective-C 编程之道: iOS 设计模式解析. 北京: 人民邮电出版社, 2011.
- [7] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. 设计模式——可复用面向对象软件的基础. 北京: 机械工业出版社. 2000.
- [8] 2012 年 3 月中国区 APP Store 数据监测报告. <http://www.ruanlie.com>.
- [9] 开源 RSS 解析库 MWFeedParser. <https://github.com/mwaterfall/MWFeedParser>
- [10] 开源 iOS 网格视图 GMGridView.
<https://github.com/gmoledina/GMGridView>
- [11] Dave Mark, Jeff LaMarche. iPhone3 开发基础教程. 北京: 人民邮电出版社, 2009.11.

致谢

本科生毕业论文（设计）任务书

一、题目：_____

二、指导教师对毕业论文（设计）的进度安排及任务要求：

起讫日期 200 年 月 日至 200 年 月 日

指导教师（签名）_____ 职称_____

三、系或研究所审核意见：

负责人（签名）_____

年 月 日

毕 业 论 文（设计） 考 核

一、指导教师对毕业论文（设计）的评语：

指导教师(签名) _____
年 月 日

二、答辩小组对毕业论文（设计）的答辩评语及总评成绩：

成绩比例	文献综述 占（10%）	开题报告 占（20%）	外文翻译 占（10%）	毕业论文（设计） 质量及答辩 占（60%）	总 评 成绩
分值					

答辩小组负责人（签名） _____
年 月 日