顶点云 开发者文档

发布

Forec

2017 年 01 月 02 日

Contents

ZENITH

欢迎阅读顶点云服务器的开发者文档。本文档分为两部分：《顶点云应用程序服务器》和《顶点云 *Web* 服务器》。其中，Web 服务器使用 Flask 框架，应用程序服务器使用 Golang，这二者不在本文档的范围之内，如有兴趣请移步：

- Flask 文档
- Go 语言文档

《顶点云应用程序服务器》和《顶点云 *Web* 服务器》均包含对应的部署说明和快速上手指南，我建议您按文档推荐的顺序阅读。

下面简要介绍二者的关系：在提出设计顶点云时就已考虑到设计两个服务器的需求，因此二者相互兼容，可同时运行在一台主机，同时为应用程序客户端和 Web 用户提供服务。顶点云的应用程序服务器早于 Web 服务器开发，开发历时约一个月，Web 服务器根据应用程序服务器复现以提供浏览器支持，耗时约两周。如果您想了解更多顶点云的背景，欢迎阅读 前言 。

CHAPTER 1

用户指南

此部分先介绍了顶点云的背景，之后划分为应用程序服务器和 Web 服务器，二者均有各自对应的配置、上手指南。

1.1 前言

在了解顶点云前请阅读本文。希望本文可以回答您有关顶点云的用途和目的，以及是否应当使用顶点云等问题。

1.1.1 顶点云的来源

顶点云是 我（ Forec ）和 non1996 在 2016 年程序设计实践的课程设计，目的是为北邮学生提供一个基于校内网的云存储平台。

北邮有着强大的校园网和教育网资源，但却缺少一个可供校内学生存储/分享文件的平台，即便是在连接到校园网的计算机上打印一份很小的文件，也需要移动存储介质的帮助。在校园网的基础上提供一个简易、小规模的云存储应用，对校园生活的方方面面都能带来较大的改善。

顶点（ZENITH）云的名称随机选自我一位室友准备的 GRE 单词。

1.1.2 顶点云的功能

顶点云最初的设定就是提供基本的云存储功能。顶点云最终提供功能的设计参考了目前主流云盘业务的特点，增加了分享、Fork、网页通讯等。注册用户可上传资源、拷贝/移动自己的资源、Fork他人的共享资源（需资源提取码）等。

顶点云提供了两种访问方式，使用 Windows 系统的用户可下载客户端获取顶点云应用程序服务器的支持，其他系统用户可通过任意浏览器访问顶点云的 Web 服务器。

顶点云的 Web 服务器允许用户可上传不超过 100 MB 的文件。顶点云的 Web 端同时允许用户上传整个目录压缩包，用户只要将要上传的目录打包并修改后缀名，服务器即可正确还原整个目录结构，这与当前百度云盘的单个文件上传相比有所改进。

顶点云的 Windows 客户端提供了更好的用户体验，在测试中，使用有线网络可达到 42MB/s 的下载速度。此外，Windows 客户端对用户更友好，操作更方便。

顶点云的 Windows 客户端由 non1996 编写，感谢他对我的理解和支持，在我屡次修改需求和协议的情况下还能坚持下去。

1.1.3 顶点云的配置

顶点云的 Web 服务器和应用程序服务器均提供了一些默认的配置文件，Web 服务器的配置文件为 `config.py`，应用服务器的配置文件为 `config.go`。你可以在 应用程序服务器全局设置 查看应用程序服务器的详细配置说明，在 *Web* 服务器全局配置 查看 Web 服务器的详细配置说明。

1.1.4 顶点云的可持续发展

顶点云作为一个非常简单的云存储系统，自身存在非常多的不足。在设计过程中，我使用了很多非常幼稚的方法来投机取巧，我会在空闲时间逐步修复其中的不足之处、添加新的功能。如果你有兴趣，也欢迎你为顶点云添砖加瓦。你可以向我 发送邮件 或直接在顶点云的 GitHub 仓库中提交 ISSUE 或 PR 。感谢您的支持！

接下来请阅读 部署顶点云应用程序服务器 或 部署顶点云 *Web* 服务器 。

1.2 顶点云应用程序服务器

此部分文档将主要介绍顶点云应用程序服务器的框架、模型和用户代理，文档分成几个部分，我推荐你按如下顺序阅读：

- 部署顶点云应用程序服务器
- 应用程序服务器全局设置
- 快速上手
- 模型介绍
- 协议设计
- 框架分析
- 测试
- 用户代理

顶点云应用程序服务器使用 Golang 编写，Go 语言的语法和特性不在本文档的范围内，如果你想了解 Go 语言请移步：

- Go 语言文档

1.2.1 开发者指南

这部分文档比较松散，首先简单介绍部署和配置，之后将从框架、协议设计到具体细节说明顶点云应用程序服务器如何处理请求、转发请求、文件共享等。

部署顶点云应用程序服务器

顶点云的应用程序服务器使用 Go 语言编写，部署前请确保您已安装 Golang，并向环境变量中正确添加了 GOROOT、GOPATH 等记录。我们推荐的 Go 语言版本为 Go 1.7.3。有关 Go 语言的环境配置不在本文档范围内，如果您尚不了解，请参阅以下相关链接：

- 安装 Golang
- Go 语言文档

获取源码

顶点云的应用程序服务器源码托管在 GitHub 上，您可以使用 Git 克隆仓库或直接通过 GitHub 下载源码的压缩包。假设您熟悉 Git，请通过以下命令获取源码。

请注意：顶点云的 GitHub 仓库中包含了应用程序服务器和 Web 服务器两部分，因为 Go 语言要求编译的包需要在 *GOPATH* 目录下，因此你需要将 *app* 目录移动到 *GOPATH* 下，并将其重命名为 *cloud-storage* 。

```
git clone https://github.com/Forec/zenith-cloud.git
    cd zenith-cloud
    mv app $GOPATH/cloud-storage
cd $GOPATH/cloud-storage
```

此时您应当已经进入顶点云的应用程序服务器源码目录。

安装第三方库

顶点云默认使用的数据库为 SQLITE3，使用了第三方的 go-sqlite3 库，请确保您已正确安装 Golang，并执行以下命令安装第三方库。

```
go get github.com/mattn/go-sqlite3
```

运行测试

顶点云的应用程序服务器提供了一部分单元测试，您可以运行单元测试以确保环境配置正常。

进入 *app/test* 目录，运行 *runtest.sh* （Linux 系统）或 *runtest.bat* （Windows系统）。若测试结果显示通过则顶点云的应用程序服务器部署成功。

至此，顶点云应用程序已部署完成。顶点云可运行在任何主流体系结构计算机以及任何操作系统上。接下来请您阅读 应用程序服务器全局设置 以根据您的系统配置顶点云。

应用程序服务器全局设置

在阅读以下内容前，请确保您已经按照 部署顶点云应用程序服务器 正确部署了应用程序服务器，并位于源码目录下（/path-to/zenith-cloud/app/）。

您可以根据您的环境修改源码目录下的 *config/config.go* 文件以配置服务器。

样例配置文件

您从 GitHub 仓库获取的源码中已经包含了一份默认的配置文件，忽略注释和函数部分后内容如下：

```
const STORE_PATH = "G:\\Cloud\\"
const DOWNLOAD_PATH = "G:\\Cloud\\"
const CLIENT_VERSION = "Windows"
const AUTHEN_BUFSIZE = 1024
const BUFLEN = 4096 * 1024
const MAXTRANSMITTER = 20
const DATABASE_TYPE = "sqlite3"
const DATABASE_PATH = "work.db"
const START_USER_LIST = 10
const TEST_USERNAME = "forec@bupt.edu.cn"
const TEST_PASSWORD = "TESTTHISPASSWORD"
```

```
const TEST_SAFELEVEL = 3
const TEST_IP = "127.0.0.1"
const TEST_PORT = 10087
const SEPERATER = "+"
const CHECK_MESSAGE_SEPERATE = 5
```

样例配置文件每项均有对应注释，您也可以查看下面的 样例文件详细解释 了解每一项的具体功能。

样例文件详细解释

配置文件中的每一项对应配置如下：

1. STORE_PATH： 服务器中存储所有用户上传文件的路径。
2. DOWNLOAD_PATH： 在 app/client 目录中提供了一个测试客户端，此记录用于设置测试客户端从服务器下载文件的存储路径。
3. CLIENT_VERSION： 测试用客户端使用的操作系统。
4. AUTHEN_BUFSIZE： 服务器和客户端之间命令交互连接使用的缓冲区大小，如果你不了解此字段的含义，可以保留默认值。此字段详细的介绍请参考 交互式命令缓冲区 。
5. BUFLEN： 服务器和客户端之间传输长数据流时使用的缓冲区大小，在网络环境较差的情况下应降低该项的大小，在网络丢包率较低的情况下应增加该项的大小。如果你不了解此字段的含义，可以保留默认值。此字段的详细介绍请参考 长数据流传输缓冲区 。
6. MAXTRANSMITTER： 服务器为每个客户端允许分配的最大同时活动连接数量，增加此数值可以在用户尝试下载多个文件时提供并行能力。
7. DATABASE_TYPE： 服务器使用的数据库类型。
8. DATABASE_PATH： 服务器使用的数据库所在的路径，此路径可为绝对路径，也可为相对服务器可执行文件所在目录的相对路径。
9. START_USER_LIST： 服务器在启动后会维护一个活动的用户池，此项指定了服务器启动时分配的用户池大小，在加入新用户后无需分配新内存，除非用户池已满。
10. TEST_USERNAME： 客户端测试使用的用户名，你可以手动此用户加入数据库，或在服务器第一次启动时将此用户模型加入数据库。默认配置文件中的用户名、密码已经加入到仓库中的 work.db 数据库。
11. TEST_PASSWORD： 客户端测试使用的登陆密码。
12. TEST_SAFELEVEL： 测试客户端和服务器约定使用的安全等级，可设置为任意整数，小于等于 1 时被规整为 1，大于等于 3 时被规整为 3。1、2、3 等级分别对应在加密时使用 16、24、32字节的密钥。
13. TEST_IP： 测试服务器时开放的 IP 地址。
14. TEST_PORT： 测试服务器时开放的端口。
15. SEPERATER： 此项指定了服务器和客户端约定的命令格式使用的分隔符，默认情况下为 + 。如果你不了解命令格式，可以查看 命令格式 。
16. CHECK_MESSAGE_SEPERATE： 此项指定了服务器转发消息执行的间隔，默认为 5。服务器会每隔 CHECK_MESSAGE_SEPERATE 秒向每个用户转发其他用户发送给他的消息。如果你不明白此项的意义，请查看 用户通讯 。

例如，在使用 WiFi 连接的 Ubuntu 16.04 下部署服务器以供 Windows 客户端使用时，可参考的配置文件如下：

```
const STORE_PATH = "/usr/local/cloud-store/"
const DOWNLOAD_PATH = "/usr/local/cloud-download"
const CLIENT_VERSION = "Windows"
const AUTHEN_BUFSIZE = 1024
const BUFLEN = 1024 * 1024  // 网络情况较差，减小缓冲区长度
const MAXTRANSMITTER = 10
const DATABASE_TYPE = "sqlite3"
const DATABASE_PATH = "/usr/local/cloud/cloud.db"
const START_USER_LIST = 10
const TEST_USERNAME = "test@test.com"
const TEST_PASSWORD = "TEST"
const TEST_SAFELEVEL = 1
const TEST_IP = "you ip address on Internet" // 开放在公网地址
const TEST_PORT = 10087
const SEPERATER = "+"
const CHECK_LIVE_SEPERATE = 10
const CHECK_LIVE_TAG = "[check]"
const CHECK_MESSAGE_SEPERATE = 10
```

接下来请您阅读 快速上手 。

快速上手

此部分文档将带领您在一个假想的纯净环境中部署、配置、扩展自定义功能并启动顶点云服务器。

假想环境

有一天我意外获得了一台免费的 CVM，而我恰巧获得了一次得到顶点云源码的机会。这台云主机使用的操作系统为 CentOS 7.2，公网 IP 地址为 `123.123.123.123` ，已安装好 Golang 并配置了环境变量。下面的指令均通过 SSH 远程操作。

```
cd $GOPATH
git clone https://github.com/Forec/zenith-cloud.git
    cd zenith-cloud
    mv app $GOPATH/cloud-storage
cd $GOPATH/cloud-storage
```

我已经获得了顶点云应用程序服务器的源码，接下来安装第三方库。

```
go get github.com/mattn/go-sqlite3
```

很高兴 Golang 顺利地配置了第三方库。接下来测试一下代码是否能够在本地机器通过测试。

```
cd test
./runtest.sh
```

非常顺利！测试脚本告诉我所有测试均已完成，顶点云应用程序各个基础模块能够在这台服务器上运转正常。

针对假想环境修改配置文件

鉴于顶点云提供的默认配置仅适用于 Forec 的史诗级笔记本，下面根据这台服务器的情况修改配置文件。编辑 *config.go* :

```
cd ../config
nano config.go
```

我根据如下考虑对配置文件做了一定修改：

- 我想将用户上传的文件存储到 /usr/local/cloud ，因此我修改 STORE_PATH = "/usr/local/cloud"
- 我想让这台云主机上运行的服务器为 Linux 用户提供服务，因此修改 CLIENT_VERSION = "Linux"
- 我的云主机网络情况良好并且带宽很高，因此我觉得可以适当提高缓冲区的大小，修改 AUTHEN_BUFSIZE = 4096，修改 BUFSIZE = 3 * 4096 * 1024
- 我觉得顶点云默认使用的 SQLITE 数据库很方便，因此我决定保留默认配置
- 我的服务器这么强，因此我将每个用户允许的最大同时连接线程数设置为 50
- 我觉得我的服务器可能会受到大家的热烈欢迎，因此我决定增加初始用户池大小，修改 START_USER_LIST = 100，我想应该足够了。
- 我想让我的服务器更安全，因此我保留了默认配置中的最高安全等级
- 我想让世界上任何一个角落均能访问我的顶点云服务器，因此我将测试 IP 地址修改为 123.123.123.123
- 我觉得剩下的配置没什么需要变动的了，不过每 5 秒转发一次消息似乎有些过慢了，因此我将 CHECK_MESSAGE_SEPERATE 修改为 2

看起来配置文件没什么值得修改的了，我决定按下 CTRL+X 保存配置文件，顺便通过 git diff 检查了一下修改过的部分：

```
const STORE_PATH = "/usr/local/cloud"
const CLIENT_VERSION = "Linux"
const AUTHEN_BUFSIZE = 4096
const BUFLEN = 3 * 4096 * 1024
const MAXTRANSMITTER = 50
const START_USER_LIST = 100
const TEST_SAFELEVEL = 3
const TEST_IP = "123.123.123.123"
const CHECK_MESSAGE_SEPERATE = 2
```

启动顶点云应用程序服务器

看起来万事大吉了，我决定启动服务器看看究竟是否如此。

```
cd ../cloud
go build cloud.go
./cloud
```

似乎运行成功了？我决定配置一下测试客户端，看看是否能够正常使用。

运行测试客户端

启动一个新的 SSH 连接，进入配置文件所在的目录，编辑配置文件。

配置文件中的默认用户测试密码实在是太长了，但是很无奈，为了方便不得不使用默认的数据库和测试用户。经过检查，我还需要设置测试客户端下载文件保存路径，因此我修改 DOWNLOAD_PATH = "/usr/local/cloud/download"。

看起来似乎配置好了，我决定运行测试客户端尝试一下。

```
cd client
go build client.go && ./client
```

测试客户端提示我输入命令，看起来似乎运行正常。我决定做如下尝试:

```
请输入命令: ls+0+/
UID   PATH    FILE        CREATED TIME    SIZE    SHARED   MODE
请输入命令: touch+test.txt+/home/+0
xxxxxxxx
请输入命令: put+1+13990+18459158D123788165BBB8C3F3DFDF91
上传传输结束
请输入命令: ls+1+/
UID   PATH    FILE        CREATED TIME    SIZE    SHARED   MODE
  1   /home/  client.go   xxxxxxxxxxxx    13990     0      FILE
请输入命令: get+1+?
文件 test.txt 已被下载
```

我真的很讨厌 Forec 的这一套指令，冗长而且难懂。不过毕竟这里只是一个测试用的客户端，没有图形界面的包装。在尝试了专门设计的 客户端 后，我觉得效果还是可以接受的，不过这都是后话了。

我阅读了 协议设计 ，终于明白了上面指令的意义:

- 第一条 `ls` 指令用于获取目录 `/` 下的资源列表，在开始时数据库中没有任何记录，所以只有返回的表头。两个 `+` 之间的数字 0 代表只查询 `/` 一级目录下的文件
- 第二条 `touch` 指令创建了一个空文件，我创建了一个名为 `test.txt` 的空文件，后面的 `/home/` 代表我想将 `test.txt` 创建在我的云盘的 `/home/` 目录下。很高兴顶点云的服务器还算人性化，虽然我此前并没有创建过 `home` 目录，不过在执行完这条命令后，服务器为我同时创建了 `home` 目录和 `test.txt` 文件。最后一个数字 0 表示我创建的是一个文件而非目录。
- 第三条 `put` 指令向服务器中的一个文件上传数据，我将测试客户端当前路径下的文件 `client.go` 随手上传了。数字 1 代表要上传文件资源的编号，因为我刚刚开始使用顶点云，数据库还是空的，添加的第一条记录必然对应编号 1。它的文件大小是 13990 字节，根据 Forec 的协议计算出的 MD5 值为 18459158D123788165BBB8C3F3DFDF91。
- 第四条 `ls` 指令递归获取目录 `/` 下的资源。很高兴我看到了刚刚创建的文件，并且它的大小已经成了 13990 字节，路径也没有错误。
- 最后一条 `get` 指令，我决定下载刚刚上传的文件，看看是否真的可行。这里的数字 1 仍然是我要下载的文件编号，问号处其实可以填写任意非空值，这个参数只有在我想下载别人的文件时服务器才会检查。很高兴服务器提醒我下载成功了。

经过比对，顶点云服务器似乎基本的功能都执行正常。不过，我有更好的功能想添加。我通过 `CTRL+C` 结束掉了正在运行的服务器和测试客户端。

扩展自定义功能

不得不说 Forec 的设计实在是太简陋了，至少客户端应该能够看到自己的昵称！我想，添加一条指令以获取自己的用户名这个功能应该不那么困难。

在阅读了 框架分析 后，我了解了整个顶点云应用程序服务器的结构，下面我准备添加这个简单的功能。

进入 `cstruct` 目录并编辑 `cuser_operations.go`:

```
cd cstruct
nano cuser_operations.go
```

我在源码的 70 行附近发现了一个 `switch` 代码块，很明显这里将命令映射到了不同的处理函数上。我决定定义一个新的指令 `whatsmyname` 并在最后一个 `case` 和 `default` 之间添加一条映射关系，将它映射到我自定义的受理函数 `lookupname` 上:

```
case len(command) >= 5 && strings.ToUpper(command[:5]) == "CHMOD":
        // 改变资源权限
        u.chmod(db, command)

// 此处添加自定义映射关系
case len(command) >= 11 && strings.ToUpper(command[:11) == "WHATSMYNAME":
        u.lookupname(command)

default:
        // 指令无法识别，返回错误信息
        u.listen.SendBytes([]byte("Invalid Command"))
```

添加了映射关系，我决定实现受理此命令的函数 `lookupname`：

```
func (u *cuser) lookupname(command string) {
        if len(command) < 11 ||
                strings.ToUpper(command[:11]) != "WHATSMYNAME" ||
                u.nickname == ""{
                // 指令不合法或用户不存在昵称
                u.listen.SendBytes("查询失败！")
        } else {
                u.listen.SendBytes(u.nickname)
        }
}
```

看起来已经没有需要改动的地方了。我将服务器重新编译了一次，并启动了测试客户端：

```
cd ../cloud
go build cloud.go && ./cloud &
cd ../client
go build client.go && ./client
请输入命令：whatsnyname
forec
```

虽然我很不喜欢 Forec 这个名字，但是他毕竟只是个测试用户而已，至少这说明了我的功能扩展成功了。

接下来请您阅读 协议设计 。

模型介绍

此部分文档主要介绍顶点云应用程序服务器使用的数据库格式和模型。

数据库

顶点云默认配置使用 SQLITE ，您可以修改配置文件以选择适合您机器的数据库。顶点云的源码目录中提供了一个默认的 `work.db` 数据库文件，如果您选择使用默认配置，该文件足以满足您的需求。

顶点云的数据库包含 5 张表：

- *cuser*： 用户模型表
- *ufile*： 用户资源记录表
- *cfile*： 服务器存储实体文件记录表
- *cmessages*： 用户聊天消息记录表
- *coperations*： 用户操作记录表，此表默认未使用，仅作扩充用

表 cuser 此表用于存储用户信息，表格式如下：

uid	email	password_hash	created	confirmed	nickname	avatar_hash
INTEGER PRIMARY KEY AUTOINCREMENT	VARCHAR(64)	VARCHAR(32)	DATE	BOOLEAN	VARCHAR(64)	VARCHAR(32)
用户编号	用户邮箱	用户密钥 md5	用户创建日期	是否激活	用户昵称	头像链接
about_me	last_seen	member_since	score	role_id	used	maxm
VARCHAR(256)	DATE	DATE	INTEGER	INTEGER	INTEGER	INTEGER
用户介绍	上次登录日期	用户注册时间	用户积分	用户角色	已使用容量	帐号最大容量

表 ufile 此表用于存储用户资源信息，表格式如下：

uid	ownerid	cfileid	path	perlink	created	shared
INTEGER PRIMARY KEY AUTOINCREMENT	INTEGER	INTEGER	VARCHAR(256)	VARCHAR(128)	DATE	INTEGER
资源编号	资源所有者编号	引用实体文件编号	资源路径	资源外链	创建时间	资源分享数
downloaded	filename	private	linkpass	isdir	description	
INTEGER	VARCHAR(128)	BOOLEAN	VARCHAR(4)	BOOLEAN	VARCHAR(256)	
资源下载数量	资源名称	资源是否为私有	文件提取码	是否为目录	资源描述	

表 cfile 此表用于存储服务器上实体文件记录，表格式如下：

uid	md5	size	ref	created
INTEGER PRIMARY KEY AUTOINCREMENT	VARCHAR(32)	INTEGER	INTEGER	DATE
实体文件编号	文件的 md5 值	文件大小	文件被引用数量	创建时间

表 cmessage 此表用于存储用户之间互相发送的消息记录，表格式如下：

mesid	targetid	sendid	message	created
INTEGER PRIMARY KEY AUTOINCREMENT	INTEGER	INTEGER	VARCHAR(512)	DATE
消息编号	消息接收者编号	消息发送者编号	消息内容	创建时间
sended	viewed	send_delete	recv_delete	
BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN	
消息是否已发送	接收者是否已读	发送方是否已删除	接收方是否已删除	

表 coperations 此表用于存储用户删除操作记录。默认的顶点云未使用此表，该表仅在扩展提供找回用户已删除文件的功能时启用。表格式如下：

oprid	deletedU-FileId	deletedUFile-Name	deletedU-FilePath	relatedCFileId	time
INTEGER PRIMARY KEY AUTOINCREMENT	INTEGER	VAR-CHAR(128)	VAR-CHAR(256)	INTEGER	DATE
操作记录编号	删除资源编号	删除资源名称	删除资源路径	引用实体文件编号	操作时间

内置自定义类

此部分文档主要介绍顶点云应用程序服务器封装的几个自定义类（虽然 Go 语言不是 OOP 语言，但其结构提和 OOP 中的类相似，因此此处用 类 代替 结构 ），包括：

- *server*： 服务器类
- *client*： 测试客户端类
- *cuser*： 用户代理类
- *ufile*： 资源类（自定义资源结构，默认使用数据库查询代替此类型，仅为拓展功能设计）
- *cfile*： 文件实体类（自定义资源结构，默认使用数据库查询代替此类型，仅为拓展功能设计）
- *transmitter*： 传输器

下面将主要介绍 *server* 、 *cuser* 以及 *transmitter* 。

server *server* 封装了一个可执行的服务器实例类，结构元素定义如下：

```
type Server struct {
        listener      net.Listener // 请求监听线程
        loginUserList []cs.User    // 已登陆用户列表
        db            *sql.DB      // 数据库句柄
}
```

server 类具有以下几个主要方法，API 的详细介绍请查看 服务器类 ：

- *InitDB*： 初始化数据库函数，在创建服务器实例后调用以修复不存在的表。
- *CheckBroadCast*： 消息转发函数，此函数通常在一个独立的协程中执行，负责用户通讯。
- *Run*： 启动函数，将服务器实例开放在指定 IP 地址和端口。
- *Communicate*： 用户代理函数，每个在线用户均有一个对应的 *Communicate* 函数运行在独立协程中提供服务。
- *Login*： 连接认证函数，负责新连接的认证和转发。

cuser *cuser* 封装了用户代理类，每个 *cuser* 实例负责处理服务器转发的请求并按业务逻辑处理用户各类命令，同时也维护了用户的一部分资料。

cuser 类的结构元素定义如下：

```
type cuser struct {
        id          int64                // 用户编号
        used        int64                // 用户已使用的云盘容量
        maxm        int64                // 用户可使用的最大云盘容量
        listen      trans.Transmitable   // 与客户端交互命令的连接
        infos       trans.Transmitable   // 向客户端推送消息的连接
        username    string               // 用户邮箱
        nickname    string               // 用户昵称
```

```
        token       string                // 用户本次在线使用的 token
        curpath     string                // 用户当前所在的虚拟路径
        avatar_hash string                // 用户头像链接
        pass_hash   string                // 用户密码哈希值
        worklist    []trans.Transmitable // 用户当前活动连接池
}
```

cuser 类只能通过工厂方法构造，其工厂方法为 *NewCUser* ， 其开放的接口为 *User* 接口 。

cuser 类具有如下几个主要方法，API 的详细介绍请查看 *User* 接口 。

- *SetListener* ： 设置用户命令交互线程
- *SetInfos* ： 设置用户被动监听线程
- *AddTransmit* ： 添加一个活动连接到工作池
- *RemoveTransmit* ： 移除当前用户某个活动连接
- *DealWithRequests* ： 处理用户命令交互请求
- *DealWithTransmission* ： 处理用户长数据流传输请求
- *Logout* ： 登出当前用户

传输器 *transmitter* 封装了顶点云的传输过程，每个 *transmitter* 实例负责维护一个活动的 Socket 连接，并以顶点云的协议格式化消息、维持消息边界。

transmitter 类的结构元素定义如下：

```
type transmitter struct {
        conn    net.Conn      // Socket 连接
        block   cipher.Block  // 加密模块
        buf     []byte        // 缓冲区指针
        recvLen int64         // 当前缓冲区内缓存数据长度
        buflen  int64         // 缓冲区总长度
}
```

transmitter 类只能通过工厂方法构造，其工厂方法为 *NewTransmitter* ， 其开放的接口为 *Transmitable* 。

transmitter 类具有如下几个主要方法，API 的详细介绍请查看 传输器 。

- *GetConn* ： 获取传输器封装的 Socket 连接
- *GetBuf* ： 获取传输器内部的缓冲区指针
- *GetBuflen* ： 获取传输器内部的缓冲区长度
- *GetBlock* ： 获取传输器内部的加密模块
- *SetBuflen* ： 设置传输器使用的缓冲区长度
- *SendBytes* ： 使用此传输器按协议格式发送字节流，可维持边界
- *SendFromReader* ： 使用此传输器从可读结构发送字节流
- *RecvUntil* ： 使用此传输器接收数据直到达到设定数量
- *RecvBytes* ： 使用此传输器按协议格式接收字节流，维持边界
- *RecvToWriter* ： 使用此传输器按协议格式接收字节流并写入可写结构
- *Destroy* ： 销毁此传输器

可扩展自定义类 *ufile* 和 *cfile* 类可在扩展功能时使用，这两个类默认只提供了基本的元素设置、获取方法，以及相关的列表操作。如果您需要使用到这两个类，请在 `cstruct/ufile.go` 和 `cstruct/cfile.go` 中添加自定义方法。

接下来请您阅读 协议设计 。

协议设计

顶点云的协议格式非常简单且对带宽的利用率不高，但相比长数据流的传输，其低效还是可以容忍的。

认证流程

此部分主要介绍顶点云客户端在和服务器授权认证时的流程：

1. 客户端申请与服务器开放端口建立 TCP 连接
2. 服务器响应客户端的请求，以明文方式向客户端发送一个约定长度的 `token` ，记为 `T`
3. 若此次认证是客户端初次与服务器建立连接，则客户端将用户名的明文和用户输入的密钥 md5 加盐加密值以字符串形式连接，记为 `B` ，转 ，否则转 4
4. 若此次认证是客户端试图与服务器建立 长数据流连接 或 被动监听连接 ，则将用户名的明文和首次认证身份时接收到的 `token` 以字符串形式连接在一起，记为 `C`
5. 客户端接收 `token` 后按照 认证消息格式 使用 `token` 作为 AES CFB 算法的密钥加密 `B` 或 `C` ，得到结果 `S` 并将 `S` 发送回服务器
6. 服务器接收客户端响应信息，根据 认证消息格式 从认证消息中解析出用户名的明文和一个字符串 `A` ，该字符串或者为用户身份认证使用的密钥 md5 值，或者为客户端初次认证接收的 `token` 值
7. 服务器检查在线用户列表是否存在用户名与请求认证的用户名一致，如存在，则比对服务器存储的该用户模型的 `token` 值，若该 `token` 与 `A` 相同，则转 8，否则转 9 。若在线用户列表不存在用户名和请求认证的用户名一致，则转 10
8. 服务器认为该用户使用的客户端试图建立 长数据流连接 或 被动监听连接 ，因此检查服务器存储的该用户模型中是否已存在被动监听连接，若存在，则认为客户端新申请的连接用于长数据流传输，将其加入该用户的活动连接池中，否则将其设为该用户的被动监听连接。
9. 服务器认为该用户在另一地点重复登录，向当前在线的该用户发送异地登录的警告信息、登出原用户并登入新认证的客户端
10. 服务器在线用户列表不存在该用户，因此此连接必定为初次认证。服务器从数据库检索用户信息，比对该用户的密钥 md5 值与 `A` ，若相同则认为客户端认证信息合法，转 11 ，否则转 13
11. 服务器向在线用户列表添加该用户，并初始化该用户的信息，如为该用户设置 `token` 值为 `T` ，设置昵称、已使用容量、最大容量等
12. 服务器向该用户以 普通字节流消息格式 方式发送 `T` ，以协助客户端确定连接建立。之后服务器将该用户请求转发给用户代理，认证流程结束
13. 服务器认为客户端认证信息不合法，单方面中断 TCP 连接，认证流程结束

协议消息格式

此部分主要介绍顶点云 C/S 交互时使用的消息格式。消息格式主要分为 认证消息格式 和 普通字节流消息格式 。

认证消息格式 认证消息格式指的是客户端和服务器建立连接之初，客户端响应服务器认证身份请求时使用的协议格式。此格式的报文在每个连接开始时发送且仅发送一次，如果对此报文验证失败，服务器会主动断开和客户端的连接，客户端需要重新申请接入。

认证消息格式分为四个部分，大致如下表所示，所有长度的单位均为字节。如果你不了解表头各项含义，请阅读 认证流程 。

B 或 C 的明文长度	S 的长度	用户名明文的长度	S
8 字节	8 字节	8 字节	长度不定

上表中，前三个字段均为由一个 `Int64` 类型数据按大端序转化得到 8 个字节。

普通字节流消息格式 顶点云在认证过程以外的任何传输过程中均使用此格式，每个包可分为三个部分，大致如下表所示，表中所有长度的单位均为字节。应当注意，如果一个字节流过长， *SendBytes* 和 *SendFromReader* 会将其自动拆分为多段，组成多个包发送。

该包携带的明文长度	该包总长度	携带的加密数据
8 字节	8 字节	长度不定

加密

顶点云的应用程序服务器采用 AES CFB 对称密钥加密算法，理论上会受到中间人攻击威胁。我计划在下一版本中修改加密方式，用 RSA 取代当前的对称加密。

代码中使用了 Go 语言内置的 `cipher` 加密模块，具体加密、解密函数请查阅 数字处理模块 。

MD5 计算

顶点云的应用程序服务器采用如下方式计算文件/字节流的 MD5 值:

- 对于一串字节流，直接计算其 MD5 值，转化为 16 进制大写字符串（共 32 个字符）作为结果。
- 对于一个文件，将其每 4MB 划分为一块，最后一块不足 4MB 也算作一块。分别计算每块的 MD5 值并转化为 16 进制大写字符串，将其按块顺序拼接成一个新的字符串。计算这个新字符串的 MD5 值并转化为 16 进制大写字符串作为结果。

顶点云默认提供了公有函数计算这两类数据的 MD5 值，你可以查看 *MD5* 和 *CalcMD5ForReader* 以了解更多。

命令格式

顶点云采用简单文本格式的字节流传输用户指令，客户端 GUI 将用户输入格式化后按命令格式发送给服务器以获取支持。顶点云的内置命令可分为 交互式命令 和 文件传输命令 。不同命令的参数数目可能不同，参数间用 `SEPERATER` （如果您尚不了解，请查阅 应用程序服务器全局设置 ）分隔，因此用户参数中不应包含任何 `SEPERATER` 字符。

交互式命令

交互式命令指用户向服务器请求查询、文件软操作等服务时使用的指令，此类命令的发送和响应均通过客户端和服务器建立的交互式 传输器 传输，响应格式也为纯文本字节流。默认的顶点云包含如下交互式指令:

Command	Param1	Param2	Param3	Others
ls	Recurssive	Quering Path	Quering Keywords	
touch	Filename	Path	Type Identifier	Nothing
cp	File Id	New Path	Nothing	
mv	File Id	New Filename	New Path	Nothing
rm	File Id	Nothing		
fork	File Id	Password	New Path	Nothing
chmod	File Id	Is Private	Password	Nothing
send	Nickname	Message	Nothing	

命令参数解释 下面详细说明各指令参数含义，在阅读前请确保您已经了解 模型介绍 中的基本内容:

- **ls**
- `Recurrsive`：可为 0 或 1。为 1 代表递归查询，服务器会返回当前用户云盘空间中 `Quering Path` 下任意深度的文件和目录；为 0 时仅返回一级目录的资源。
- `Quering Path`：用户要查询的云盘空间路径，路径不存在时等同于查询空目录。
- `Quering Keywords`：用户查询使用的关键词，数量不限。例如查询名称中包含 *zenith* 和 *cloud* 的资源，可附加这两个单词作为参数。
- 样例：查询 `/home/` 任意深度目录下的名称包含 *test* 的资源，使用命令 `ls<SEP>1<SEP>/home/<SEP>test`，`<SEP>` 为用户配置文件中指定的 `SEPERATER` 字符。
- **touch**
- `Filename`：要创建的新文件的名称。
- `Path`：创建的新文件所在的路径，若该路径不存在，服务器会自动创建该路径以及该路径中所有层次的目录。
- `Type Identifier`：可以为 1 或 0，为 1 代表创建文件夹，否则为文件。
- **cp**
- `File Id`：要拷贝的资源编号。
- `New Path`：要拷贝到的目标路径。
- **mv**
- `File Id`：要移动的资源编号。
- `New Filename`：移动后为资源重新命名的名称。
- `New Path`：移动到的目标路径。
- **rm**
- `File Id`：要删除的资源编号。
- **fork**
- `File Id`：要 Fork 的资源编号。
- `Password`：要 Fork 的资源的提取码。
- `New Path`：要 Fork 到的目标路径。
- **chmod**
- `File Id`：要修改权限的资源编号。

- `Is Private`：可为 1 或 0。为 1 则将资源设置为私有，若资源为目录则该目录下所有子目录/文件均被设置为私有；为 0 则将资源设置为共享，并使用 `Password` 设置资源的提取码，若资源为目录则该目录下所有子目录/文件均被设置为共享且提取码均为 `Password`。
- `Password`：将资源设置为共享时指定的提取码。若 `Is Private` 为 1，则此项可为任意非空字符。
- **send**
- `Nickname`：消息接收方的用户昵称。
- `Message`：消息实体，若消息中包含 `SEPERATER` 字符，则服务器会将该字符修正为空格。

交互式命令缓冲区 交互式命令缓冲区指服务器和客户端之间传输、响应交互式命令时，传输器 使用的缓冲区大小。此值由 `config/config.go` 中的 `AUTHEN_BUFSIZE` 项指定。

文件传输命令

文件传输命令指用户向服务器申请文件上传/下载/更新时使用的指令，此类命令的发送和响应均通过客户端和服务器建立的临时长数据流 传输器 传输。默认的顶点云包含如下文件传输指令：

Command	Param1	Param2	Param3
put	File Id	File Size	File MD5
get	File Id	Password	

命令参数解释 下面详细说明各指令参数含义，在阅读前请确保您已经了解 模型介绍 中的基本内容：

- **put**
- `File Id`：要写入数据的资源的编号，该资源编号对应的资源类型必需为文件，且必需属于当前用户，否则服务器将请求视作不合法。 *put* 操作会将数据写入资源编号对应的文件，该文件原本指向的实体文件引用会被替换为新的实体文件编号。换句话说，此操作等于向一个已存在的文件重新写入数据，该文件的内容会被替换为写入的数据，大小会被设定为新数据的长度。
- `File Size`：待写入数据的长度，单位为字节。
- `File MD5`：根据待写入数据计算出的 MD5 值，计算方法请参考 *MD5 计算* 。
- **get**
  - `File Id`：要下载的资源编号，该资源编号对应的资源类型可以为文件或目录，如果为目录，客户端会自动在本地构建云盘中的虚拟目录并将整个目录下载到本地磁盘中。
  - `Password`：要下载的资源的提取码。
  - 该资源属于当前用户：提取码可填写任意非空字符，服务器会自动忽视该字段；
  - 该资源属于其他用户且为共享资源：若该资源的提取码为空，则服务器不会检查用户填写的提取码，否则将比对提取码是否一致，一致则允许下载，否则认为请求不合法
  - 该资源属于其他用户且为私有资源：请求不合法

长数据流连接 长数据流连接指的是用户发出文件传输请求时，服务器和客户端建立的临时连接。这类连接的缓冲区更大，传输速度更快，在传输结束后连接会断开，因此此类连接的寿命较短。

长数据流传输缓冲区 长数据流传输缓冲区指服务器和客户端之间传输文件时，传输器 使用的缓冲区大小。此值由 `config/config.go` 中的 `BUFSIZE` 项指定。

上传命令协议 用户上传文件指令 `put` 的执行流程如下：

1. 假定服务器已经验证了客户端合法身份（如指令是否合法、资源是否存在、是否属于当前用户），若验证不通过则返回错误码 `300` （指令格式不合法）、`301` （资源不存在）等。
2. 服务器检查是否已存在文件实体的 MD5 值与用户要上传的 MD5 值相同，若存在则认为文件相同，返回状态码 `200` 并结束流程，否则发送状态码 `201` 表示需要传输并转 3。
3. 服务器使用用户提供的 MD5 值作为文件名新建临时文件，启动传输并将数据写入此临时文件，若传输过程出现错误则返回错误码 `203` 。
4. 数据传输完成后计算临时文件的 MD5 值，若与用户提供的相同则向 *cfile* 表中添加新记录，并将此文件重命名为新记录的编号，否则删除此文件并返回错误码 `403` 。
5. 服务器更新用户指定资源的实体文件引用编号，并返回状态码 `200` ，结束传输。
6. 以上任何一步出现因服务器原因导致的错误，将返回错误码 `500` 。

下载命令协议 用户下载资源指令 `get` 的执行流程如下：

1. 假定服务器已经验证了客户端的合法身份（如指令是否合法、资源是否存在、用户提供的提取码是否正确），若验证不通过则返回字符串 `NOTPERMITTED` 并中断连接，否则返回字符串 `VALID` 启动传输。
2. 服务器计算用户要下载的文件/目录总数。若用户下载的资源为单个文件，则总数为 1，否则为用户试图下载的目录下的全部文件和目录数量之和加 1。例如用户试图下载如下目录结构中的目录 `a` ，则需要发送的总数为 5，包括：目录 `a` ，目录 `a/b` ，目录 `a/c` 以及文件 `a/b/d.db` 和文件 `a/b/e/db` 。

```
- parent-folder
  - a
        - b
          - d.db
          - e.db
        - c
```

3. 服务器按顺序调用 *SendBytes* ，向客户端发送用户要下载的根目录名，以及类型（即数字 1，使用 *Int64 与字节转化* 转化为 8 个字节）。
4. 服务器将剩余的待发送 **目录** 按路径长度排序，之后新建一个列表 A，对于待发送的每个目录，将其在云盘中相对待下载根目录的路径和目录名连接起来，生成新的字符串加入到列表中。
5. 服务器按排序后的顺序调用 *SendBytes* ，向客户端发送列表 A 中的字符串、类型（即数字 1，使用 *Int64 与字节转化* 转化为 8 个字节）。
6. 服务器将剩余所有文件按任意顺序发送，对于每个文件，按如下顺序调用 *SendBytes* ，依次发送文件相对待下载根目录的路径+文件名、文件类型（即数字 0,转化为 8 个字节），并启动连接发送此文件。此文件传输结束后进入下一个文件的循环。对于上面的目录结构，下载流程如下：

- 服务器发送 8 个字节（使用 *Int64 与字节转化* 转化）表示的数字 5；
- 服务器使用 *SendBytes* 发送用户试图下载的根目录名称 `a` ；
- 服务器使用 *SendBytes* 发送 8 个字节表示的数字 1；
- 服务器将剩余目录 `a/b` 和 `a/c` 按路径长度排序，顺序维持不变；之后对其目录名作处理，将其相对目录 `a` 的父目录的路径保存到一个新列表中，即：`["a/b", "a/c"]` ；
- 服务器按列表的顺序，使用 *SendBytes* 分别发送：字符串 `a/b` 、8 字节数字 1、字符串 `a/c` 以及 8 字节数字 1；
- 服务器按任意顺序发送文件。假设先发送 `a/b/e.db` 。使用 *SendBytes* 发送 `e.db` 的路径+文件名，即 `a/b/e.db` ，之后发送 8 字节表示的数字 0，最后启动 *SendFromReader* 传输该资源引用的实体文件内容。

- 重复上一步的流程发送 `d/db` 。发送完成后断开连接，完成整个流程。

传输器设计

传输器模块是顶点云设计中最重要的一部分。顶点云使用传输器将协议与 Socket 连接封装在一起，简化了代码编写的复杂度。

因为 Socket 可能将多个包合并发送，接收方必需按协议设定的包格式接收、并将剩余数据缓存以待下次使用。因此每个传输器必须维护一定缓存空间，并且不能遗漏尚未使用但已接收的数据。传输器的设计目标即提供以下几个方法，使服务器的传输器和客户端对应的传输器之间每次通讯只需调用对应 API 即可。

- *SendBytes*：任意一方调用此方法即可将一串字节流发送到远端传输器，若对方未调用 *RecvBytes* 响应则阻塞，超时后断开；
- *RecvBytes*：任意一方调用此方法即可接收到远端传输器使用 *SendBytes* 发来的一份报文，若对方未调用 *SendBytes* 发送数据则阻塞，超时后断开。即使远端多次调用 *SendBytes*，传输器也可将接收到的、暂不需使用的剩余部分缓存以供后续使用。即 *SendBytes* 和 *RecvBytes* 是相对应的方法，二者通过传输器内部的缓存虚拟了两个缓冲池实现全双工通信，双方均可以将自己要发送的内容投放到虚拟缓冲池（对方传输器的缓冲区）中，也可以从虚拟缓冲池（我方传输器缓冲区）中获取对方发送的内容；
- *SendFromReader*：任意一方调用此方法即可将一个可读结构中的指定长度数据发送到远端服务器，若对方未调用任何接收函数等待则阻塞，超时后断开。类似 *SendBytes*，但若指定长度超过可读结构的内容长度，则认为执行失败；
- *RecvToWriter*：任意一方调用此方法即可响应对方调用的 *SendFromReader* 并将接收到的数据还原后写入作为参数的可写结构。即 *SendFromReader* 和 *RecvToWriter* 是相对应的方法，这两者的调用应当对称。

以上四个函数中， *SendBytes* 和 *SendFromReader* 二者发送流程相同， *RecvBytes* 和 *RecvToWriter* 二者接收流程相同。

SendBytes 发送流程如下：

1. 发送方以明文方式发送 8 个字节（大端序，使用 *Int64 与字节转化* 转化）表示的明文长度
2. 发送方从待发送字节数组中读取不超过缓冲区长度 1/3 的数据，统计待发送的明文长度，记为 A，若 A 为 0 则发送结束
3. 将待发送明文使用传输器内置的加密模块编码，得到待发送数据，统计其长度，记为 B，则 B+16 为此次要发送的包的长度
4. 拼接一个待发送数据包，该包的格式遵守 普通字节流消息格式 。发送完成后转 2

RecvBytes 接收流程如下：

1. 接收方调用 *RecvUntil* 接收满 8 字节，提取待接收明文长度
2. 接收方调用 *RecvUntil* 接收满 16 字节，提取当前包头
3. 接收方根据包头提取本包长度、明文长度，并调用 *RecvUntil* 接收整个包
4. 接收方解包，提取数据并附加到返回值中
5. 接收方将待接收明文长度减去本包包含的明文长度，若待接收长度为 0 则接收流程结束
6. 转 2，接收下一个包

用户通讯

为了降低模块之间的耦合度，顶点云的用户代理模块不具有回调服务器方法的权限。用户使用 `send` 命令向特定用户发送消息时，用户代理仅仅会将此消息写入数据库，而发送则交由服务器的守护线程处理。

被动监听连接

为了向客户端推送系统或其他用户发送的消息，每个客户端与服务器除了基本的交互式 传输器 外，还有一个被动监听连接用于被动接受服务器发送的消息并将消息展示给用户。这个连接在用户登录认证成功后立刻启动，以长数据流的方式通过认证。因为它是第一个长数据流连接，因此服务器将此连接视作被动监听线程并分配给已登录的对应用户。

接下来请您阅读 框架分析 。

框架分析

此部分文档主要介绍顶点云应用程序服务器的框架结构。

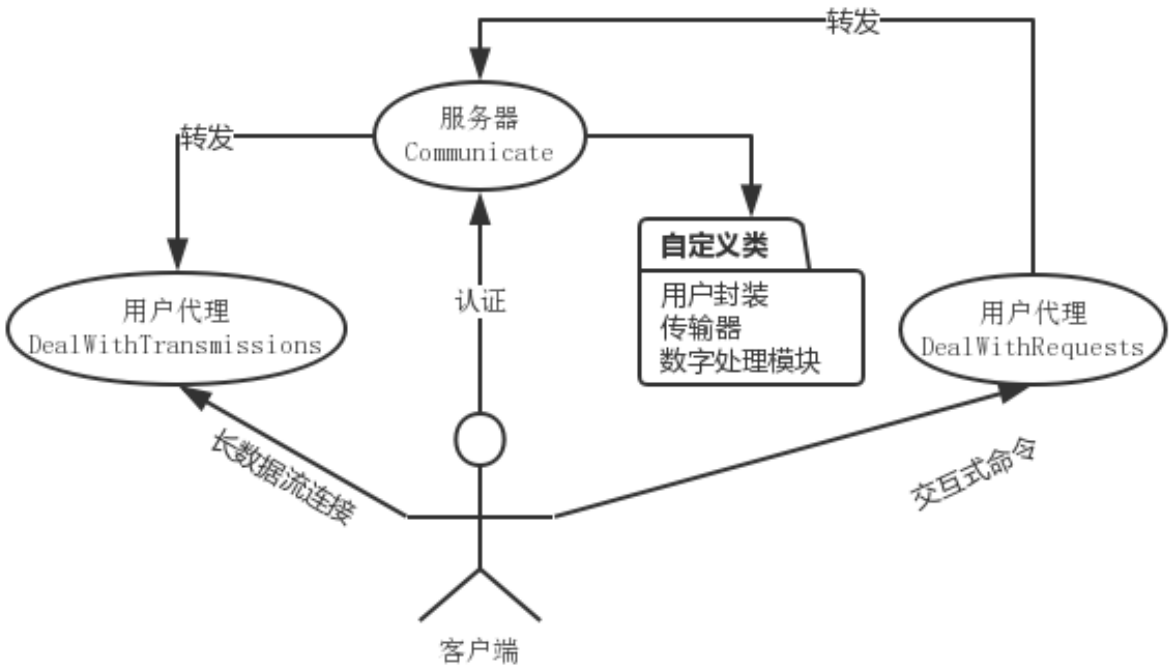
顶点云服务器源码文件结构如下：

```
- app
   - authenticate
         - authenticate.go
             - authenticate_test.go
       - client
         - client.go
       - cloud
         - cloud.go
             - work.db
       - config
         - config.go
       - cstruct
         - cuser.go
             - ulist.go
             - cuser_operations.go
             - cuser_transmissions.go
             - ...
       - server
         - server.go
       - transmit
         - transmit.go
             - transmit_test.go
```

以上几个目录对应的模型结构如下：

- *authenticate*： 对应 认证流程 中的流程以及 API 文档中的 数字处理模块 。
- *client*： 对应 测试 中的测试客户端。
- *cloud*： 用于创建和启动 *server* 实例。
- *config*： 对应 应用程序服务器全局设置 。
- *cstruct*： 对应 内置自定义类 。
- *server*： 对应 内置自定义类 中的 *server* 类。
- *transmit*： 对应 传输器设计 、 传输器 。

服务器各模块之间关系如下图所示：



接下来请您阅读 测试 。

测试

此部分文档主要介绍顶点云的单元测试和测试客户端。顶点云提供了对部分内置模块的单元测试，覆盖了 认证流程 、 传输器设计 和 内置自定义类 中的绝大部分。 *server* 可通过顶点云配备的 测试客户端 手动测试。

单元测试

此部分主要涉及几个内置包的单元测试。你可以直接运行 `test/runtest.sh` 或 `test/runtest.bat` 来激活全部单元测试。

`authenticate` 和 `cstruct` 两个包的单元测试比较简单，均采用简单的输入-输出样例比对测试。

`transmit` 包由一个微型测试服务器和一个微型测试客户端并发进行单元测试，其中涉及到 `authenticate` 包的部分功能，因此也可视作认证模块。

下面主要介绍 `transmit` 包的单元测试。

传输器测试 传输器的测试由一个微型服务器和配套的微型客户端实现。大致逻辑如下：

- 微型服务器在指定测试 IP 和端口启动
- 微型客户端在新线程中启动并尝试连接服务器以获取服务
- 服务器向客户端发送指定测试文件
- 服务器向客户端发送指定测试字符串
- 客户端接收完毕后退出

- 服务器发送结束后等待一段时间，认为客户端已经退出后检查客户端接收到的文件/字符串是否正确，正确则测试通过

传输器的测试代码位于 `authenticate/authenticate_test.go` 。测试代码主要包含两个函数，声明如下：

```
func TestTransmission(t *testing.T)
func client_test(t *testing.T)
```

函数 `TestTransmission` 用于启动微型服务器、为微型客户端提供支持以及检查传输是否正确。函数 `client_test` 用于延时启动微型客户端并接收服务器的传输请求。

默认测试使用的传输文件名为 `test_in.txt` ，顶点云提供的默认测试文件大小为 2.34MB。你可以在测试代码中替换 `test_in_filename` 为你自己的测试传输文件路径。

测试客户端

服务器逻辑功能较复杂，很难使用单元测试逐一覆盖，并且用户在使用中通常具有随机性。因此顶点云没有为服务器的每一个接口、用户代理的每一项处理函数添加单元测试，而是编写了测试客户端以手动测试。

测试用客户端代码位于 `client/client.go` ，共约 500 行， *client* 类主要的方法有：

- *Run* ： 启动测试客户端并提供与用户的交互
- *Connect* ： 负责客户端与服务器建立连接
- *Authenticate* ： 在 *Connect* 成功之后负责与服务器建立认证
- *ThreadConnect* ： 用于与服务器之间建立一个新传输器，并交付数据传输处理函数使用
- *getFile* ： 用于处理用户的下载请求
- *putFile* ： 用于处理用户的上传请求

测试客户端的界面相对专门设计的 GUI 客户端不是那么友好，但它确实是非常简单有效的测试工具。用户使用测试客户端时使用的指令需要严格按照 命令格式 的要求，参数之间的分隔符要与 `config/config.go` 中 *SEPERATER* 项指定的字符（串）相同。

例如，你可以按照 运行测试客户端 中的样例使用测试客户端。

修改测试客户端

这里主要介绍如何修改测试客户端以适应新的需求。

在测试客户端扩展功能与向用户代理添加功能类似。如果要添加的功能与数据传输相关，那么你需要在 *Run* 函数中添加转发逻辑，之后实现你的处理函数。

例如，在 快速上手 中我们为用户代理添加了新的功能。如果添加的功能不是简单的获取用户名，而是某个需要特殊处理流程的功能（如命令云主机处理一幅上传的图片并返回处理后的结果），则需要在 `client/client.go` 的第 230 行后插入转发逻辑：

```
case len(input) > 3 && strings.ToUpper(input[:3]) == "PUT":
    // 上传文件
        go c.putFile(input)
case len(input) > 8 && strings.ToUpper(input[:3]) == "GENERATE":
    // 插入新的转发逻辑，命令云存储服务器处理一幅上传的图片并返回处理后的内容
        go c.generateImage(input)
default:
    // 简单命令交互
```

之后我们需要实现自定义处理函数 generateImage ，该函数应当被添加到 client.go 的任意位置。其大致格式如下所示：

```
func (c *Client) generateImage(input string){
    // 验证指令合法
        // ...
        newThread := c.ThreadConnect(c.ip, c.port)
        // ...
    // 剩余自定义业务逻辑
}
```

接下来请您阅读 用户代理 。

用户代理

此部分文档主要介绍用户代理的一些逻辑处理函数。这部分函数是顶点云功能实现的主体，但无法将具体的实现逻辑一一梳理。这里只会简单介绍各个逻辑处理函数的功能。

交互式命令处理

此部分相关代码均位于 cstruct/cuser_operations.go 中，该代码文件中的函数结构如下所示：

```
DealWithRequests
     - ls
     - touch
     - rm
     - cp
     - mv
     - fork
     - send
     - chmod
```

只有 DealWithRequests 为公有方法，用于命令-逻辑转发，其他函数均为逻辑处理函数，不对其他包开放。

你可以参考这些逻辑处理函数实现的流程来模仿实现更多的自定义处理函数。顶点云应用程序服务器目前提供的交互式命令有限，仍有很多功能尚未提供，欢迎你为顶点云扩展更多自定义功能。新拓展的功能可以以独立文件形式提供，只要保证同属于 cstruct 包即可。

数据传输命令处理

此部分相关代码均位于 cstruct/cuser_transmissions.go 中，该代码文件中的函数结构如下图所示：

```
DealWithTransmissions:
     - get
     - put
```

只有 DealWithTransmissions 为公有方法，用于命令-逻辑转发，其他函数均为逻辑处理函数，不对其他包开放。

与数据传输命令相关的连接均会加入到当前用户的活动连接池中。当前用户的活动连接池需要由 DealWithTransmissions 方法维护，在处理完当前转发的逻辑处理函数后，DealWithTransmissions 在销毁前会移除当前结束的活动连接。

添加代理

添加代理的方式可以参考 扩展自定义功能 以及 修改测试客户端 。

接下来您可以阅读 教程 。

教程

我在个人博客开辟了一块专栏用于介绍顶点云开发从无到有的过程，如果您有兴趣了解我从设计到实现的细节，欢迎阅读专栏 顶点云设计与实现 。

顺便需要一提的是，专栏中有一部分已经随着顶点云的变更过时了，可能会出现教程和文档不一致的情况，此时以本文档为准。

到这里顶点云应用程序服务器的开发者文档部分就结束了，您可以查阅 应用程序 *API* 文档 了解更多 API 使用方法，或者阅读 应用设计要点 来查看我们对设计者提出的几点建议。您也可以阅读 顶点云 *Web* 服务器 以了解顶点云 Web 服务器的相关信息。

1.2.2 API 参考

这部分文档详细说明某个函数、结构或方法。

应用程序 API 文档

数字处理模块

此部分详细介绍数字处理模块中几个 API 的使用方式。此模块主要分为 *AES* 、 *base64* 、 *int2bytes* 、 *md5* 以及 *token* 几部分。

AES CFB 模块 此部分涉及 AES CFB 对称加密的三个公有函数：

- *NewAesBlock* ： AES 块的工厂方法，根据密钥生成一个新的 AES 加/解密模块。
- *AesEncode* ： 使用作为参数的 AES 块加密一段明文。
- *AesDecode* ： 使用作为参数的 AES 块和明文长度解密一段密文。

NewAesBlock *NewAesBlock* 用于构造新的 AES 模块，它位于文件 `authenticate/authenticate.go` 中，其函数声明如下：

```
func NewAesBlock(key []byte) cipher.Block
```

作为参数的 `key` 为一串字节，作为生成模块加/解密的密钥。此函数会返回一个满足 `cipher.Block` 接口的新 AES 模块。

AesEncode *AesEncode* 用于对一串明文加密，它位于文件 `authenticate/authenticate.go` 中，其函数声明如下：

```
AesEncode(plaintext []byte, block cipher.Block) []byte
```

该函数接受两个参数，第一个是待加密的明文字节流，第二个是一个 AES 模块。该函数会以字节切片的形式返回加密的结果。

AesDecode *AesDecode* 用于对一串加密的字节解密，它位于文件 `authenticate/authenticate.go` 中，其函数声明如下：

```
func AesDecode(cipherText []byte, plainLen int64, block cipher.Block) ([]byte, error)
```

该函数接受 3 个参数，第一个参数为待解密的字节流，第二个为解密后的明文长度，第三个为一个 AES 模块。如果该 AES 模块成功解密，则返回结果的第一部分为明文，第二部分为 `nil`，否则任何错误都将通过返回结果的第二部分传回高层函数。高层函数应当判断函数调用结果中是否包含错误以便及时恢复。

Base64 模块 此部分涉及 Base64 编码/解码的两个公有函数：

- *Base64Encode*： 使用 base64 对字节流编码
- *Base64Decode*： 使用 base64 对字节流解码

Base64Encode *Base64Encode* 可对一段明文使用 base64 编码，它位于 `authenticate/authenticate.go` 中，其函数声明如下：

```
func Base64Encode(plaintext []byte) []byte
```

该函数接受 1 个参数，该参数为待编码的字节流。该函数将参数编码后的结果以字节切片的形式返回。

Base64Decode *Base64Decode* 可对一段使用 base64 编码的字节解码。如果传入的待解码字节流不符合 base64 编码格式，则函数返回值中会携带错误。它位于文件 `authenticate/authenticate.go` 中，其函数声明如下：

```
func Base64Decode(ciphertext []byte) ([]byte, error)
```

该函数接受 1 个参数，该参数为待解码的字节流。该函数将参数解码后的结果以字节切片的形式返回，如果解码失败则会携带错误。高层函数应当检查此函数的返回值中是否包含错误信息。

Int64 与字节转化 此部分涉及 `Int64` 类型和字节数组的转换，包含两个公有函数：

- *Int64ToBytes*： 将一个 `Int64` 类型的数据按大端序转化为 8 个字节。
- *BytesToInt64*： 将 8 个字节按大端序转化为一个 `Int64` 类型的数据。

Int64ToBytes *Int64ToBytes* 位于文件 `authenticate/authenticate.go` 中，其函数声明为：

该函数接受 1 个参数，该参数为待转换的 `Int64` 格式数据，函数将转换后的 8 个字节以字节切片的形式返回。

BytesToInt64 *BytesToInt64* 位于文件 `authenticate/authenticate.go` 中，其函数声明为：

该函数接受 1 个参数，该参数为待转换的字节切片，函数会截取该切片的前 8 个字节，将转换后的 `Int64` 格式返回。

MD5 和 token 此部分涉及 MD5 值和 token 的生成。包含 5 个公有函数，均位于文件 `authenticate/authenticate.go` 中：

- *GetRandomString*： 根据给定长度随机生成字符串。
- *MD5*： 计算传入参数的 MD5 值。
- *CalcMD5ForReader*： 计算传入可读结构的 MD5 值。

- *IsMD5*：判断某个字符串是否为十六进制的 MD5 格式。
- *GenerateToken*：根据给定的安全等及生成 token。

GetRandomString *GetRandomString* 的函数声明如下：

```
func GetRandomString(length int) string
```

此函数接受一个整型数据作为生成随机字符串的长度，并返回生成的随机字符串。随机字符串包含的字符可为数字或字母（区分大小写）。若传入参数小于 0 则返回的字符串长度为 0。

MD5 *MD5* 的函数声明如下：

```
func MD5(text string) []byte
```

此函数接受一个字符串并计算其 MD5 值，以 MD5 值的 16 进制形式返回，16 进制中的字母为大写。可以直接使用 `string` 强制转换此函数的返回值，此时即可得到字符串表示的 16 进制 MD5 值。

CalcMD5ForReader *CalcMD5ForReader* 的函数声明如下：

```
func CalcMD5ForReader(reader *bufio.Reader) []byte
```

此函数接受一个可读结构作为参数，对其包含的全部数据分块计算 MD5 值。分块方法可参考 *MD5 计算* 。函数返回值与 *MD5* 返回值类型相同。

IsMD5 *IsMD5* 的函数声明如下：

```
func IsMD5(text string) bool
```

此函数接受一个字符串作为参数，检查其是否符合大写的 16 进制 32 位 MD5 格式，若符合则返回 `True`，否则返回 `False`。

GenerateToken *GenerateToken* 的函数声明如下：

```
func GenerateToken(level uint8) []byte
```

此函数接受一个 `uint8` 类型作为参数，按 样例文件详细解释 中介绍的 `TEST_SAFELEVEL` 项生成对应长度的 token。大于 3 和小于 1 的参数均会被规整到 1 ～ 3。参数为 1 时返回 16 字节长度的 token，参数为 2 时返回 24 字节长度的 token，参数为 3 时返回 32 字节长度的 token。

User 接口

User 是针对 *cuser* 类设计的接口，其定义如下：

NewCUser *NewCUser* 是 *cuser* 类的工厂方法，其声明如下：

```
func NewCUser(username string, uid int64, curpath string) *cuser
```

你需要传入新建用户的用户名、用户编号以及用户当前路径（此参数传入 / 即可，在顶点云的默认配置中，此参数未启用）。例如：

```
u := NewCUser("Forec", 1, "/")
```

公有方法 这里只介绍除元素获取和设置的其他方法。

SetListener *SetListener* 用于为用户设置交互式传输器，通常在服务器认证首次登录用户时使用，返回值为 `True` 。该函数位于文件 `cstruct/cuser.go` 中，声明如下：

```
SetListener(trans.Transmitable) bool
```

SetInfos *SetInfos* 用于为用户设置被动监听传输器，在用户第一个长数据流连接到来时调用，返回值为 `True` 。该函数位于文件 `cstruct/cuser.go` 中，声明如下：

```
SetInfos(trans.Transmitable) bool
```

AddTransmit *AddTransmit* 用于向用户活动工作池中添加一个新的传输器，添加成功则返回 `True` ，若工作池中的活动连接数目已经达到了 `MAXTRANSMITTER` 则返回 `False` 。该函数位于文件 `cstruct/cuser.go` 中，声明如下：

```
AddTransmit(trans.Transmitable) bool
```

RemoveTransmit *RemoveTransmit* 用于从用户活动工作池中移除一个指定的传输器，通常在用户活动连接工作执行完成后由用户代理调用，若未找到指定的传输器返回 `False` ，否则返回 `True` 。该函数位于文件 `cstruct/cuser.go` 中，声明如下：

```
RemoveTransmit(trans.Transmitable) bool
```

Logout *Logout* 用于登出当前用户，销毁当前用户在内存中的记录，销毁交互式传输器以及其他任何属于该用户的活动/非活动的传输器。该函数位于文件 `cstruct/cuser.go` 中，声明如下：

```
Logout()
```

DealWithRequests *DealWithRequests* 函数负责索引并转交远程用户发送的交互式命令，将交互式命令转发给各对应执行函数。在用户在线期间，该函数始终存活，在 *DealWithRequests* 函数结束后，服务器将自动执行用户登出操作。传入参数为服务器维护的数据库句柄。该函数位于文件 `cstruct/cuser_operations.go` 中，声明如下：

```
DealWithRequests(*sql.DB)
```

此方法将在 用户代理 中详细介绍。它是服务器处理用户请求的核心，能够方便的扩展功能。

DealWithTransmission *DealWithTransmission* 函数负责索引并转交远程用户发送的文件传输命令，将文件传输命令转发给各对应执行函数。在某个传输执行期间，该函数始终存活。在 *DealWithTransmission* 函数结束后，用户代理会自动移除作为函数参数的传输器。传入参数为服务器维护的数据库句柄和本次传输使用的 传输器 。该函数位于文件 `cstruct/cuser_transmissions.go` 中，声明如下：

```
DealWithTransmission(*sql.DB, trans.Transmitable)
```

此方法将在 用户代理 中详细介绍。它是服务器处理用户请求的核心，能够方便的扩展功能。

传输器

此部分详细介绍 传输器 提供方法的使用方式。传输器无法直接使用结构生成，只可以通过工厂方法 *NewTransmitter* 生成。传输器提供了一个对外的公有接口 *Transmitable* 。

Transmitable *Transmitable* 是 传输器 的公共接口，其定义如下：

```
type Transmitable interface {
        GetConn() net.Conn                          // 获取 Socket 连接
        GetBuf() []byte                             // 获取缓冲区指针
        GetBuflen() int64                           // 获取缓冲区长度
        GetBlock() cipher.Block                     // 获取加密模块
        SetBuflen(int64) bool                       // 设置缓冲区长度
        SendBytes([]byte) bool                      // 按协议格式发送字节流，可维持边界
        SendFromReader(*bufio.Reader, int64) bool // 从可读结构发送字节流
        RecvUntil(int64, int64, <-chan time.Time) (int64, error)
        // 接收数据直到达到设定数量
        RecvBytes() ([]byte, error)       // 按协议格式接收字节流，维持边界
        RecvToWriter(*bufio.Writer) bool // 按协议格式接收字节流并写入可写结构
        Destroy()                         // 销毁此传输接口
}
```

NewTransmitter *NewTransmitter* 是 传输器 的工厂方法，其函数声明如下：

```
func NewTransmitter(tconn net.Conn, tbuflen int64, token []byte) *transmitter
```

你需要传入一个 Socket 连接、传输器使用的缓冲区大小以及此传输器加密模块使用的密钥来生成一个新的传输器。例如，当前用户新加入一个连接 `conn`，需要使用 1024 字节缓冲区，密钥使用 ``1234567890123456789012345678901`2`，则通过以下代码创建传输器：

```
t := NewTransmitter(conn, 1024, "12345678901234567890123456789012")
```

公有方法 传输器模块是整个顶点云最核心的模块之一，它提供了如下几个非常重要的公有方法：

- *GetConn*： 获取传输器封装的 Socket 连接
- *GetBuf*： 获取传输器内部的缓冲区指针
- *GetBuflen*： 获取传输器内部的缓冲区长度
- *GetBlock*： 获取传输器内部的加密模块
- *SetBuflen*： 设置传输器使用的缓冲区长度
- *SendBytes*： 使用此传输器按协议格式发送字节流，可维持边界
- *SendFromReader*： 使用此传输器从可读结构发送字节流
- *RecvUntil*： 使用此传输器接收数据直到达到设定数量
- *RecvBytes*： 使用此传输器按协议格式接收字节流，维持边界
- *RecvToWriter*： 使用此传输器按协议格式接收字节流并写入可写结构
- *Destroy*： 销毁此传输器

下面主要介绍 *SendBytes* 、 *SendFromReader* 、 *RecvUntil* 、 *RecvBytes* 、 *RecvToWriter* 以及 *Destroy* 方法。

在阅读之前，请确保您了解传输器的基本原理，否则请先阅读 传输器设计 。

SendBytes *SendBytes* 函数发送一串字节流交与远程传输器，其声明如下：

```
func (t *transmitter) SendBytes(toSend []byte) bool
```

发送成功时此函数返回 `True`，在发送过程中出现任何异常均会返回 `False`。因为 AES CFB 算法加密后生成的密文长度较明文长度更长，但通常在明文长度的 2 倍以下。因此 *SendBytes* 函数在发送过程中会拆分待发送的字节数组，保证发出的每个包内的明文长度不超过传输器缓冲区的 1/3，因此加密后的长度不会超过缓冲区长度。

SendFromReader *SendFromReader* 函数从一个可读结构中读取指定长度的数据交与远程传输器，其声明如下：

```
func (t *transmitter) SendFromReader(reader *bufio.Reader, totalLength int64) bool
```

此函数接受两个参数，第一个为可读结构，传输器从此参数中读取数据；第二个为待发送数据的长度。发送成功时此函数返回 `True`，在发送过程中出现任何异常均会返回 `False`。此函数发送方式与 *SendBytes* 类似，但若可读结构中的数据长度少于指定的数据长度（传入的第二个参数），则此函数将返回 `False`，否则只读到指定长度位置便停止发送并返回。

RecvUntil *RecvBytes* 函数从远程传输器接收满指定长度的数据，其声明如下：

```
func (t *transmitter) RecvUntil(until int64, init int64, chR <-chan time.Time) (int64, error)
```

此函数传入参数较多，下面详细介绍各个参数含义：

- *until*：缓冲区中接收到的数据长度超过（含） *until* 时退出此函数
- *init*：缓冲区当前已经接收到的数据长度
- *chR*：接收速率

此函数返回值包含一个 `Int64` 类型数据和可能出现的错误。若接收成功则返回值的 `Int64` 数据表示当前传输器缓冲区中存储的数据长度，否则返回值中包含错误。如果远程传输器迟迟没有发送数据，此函数会阻塞。

此函数通常在 *RecvBytes* 和 *RecvToWriter* 中调用，用于分别接收包头以及每个包体。如果你对此函数的意义不是很了解，请查阅 传输器设计 。

RecvBytes *RecvBytes* 函数从远程传输器接收符合一个消息边界的字节流，其声明如下：

```
func (t *transmitter) RecvBytes() ([]byte, error)
```

接收成功时此函数返回接收到的字节数组和 `nil`，若在发送过程中出现任何异常，则返回值中将包含错误。

RecvToWriter *RecvToWriter* 函数从远程传输器接收一定长度的数据并写入一个可读结构，其函数声明如下：

```
func (t *transmitter) RecvToWriter(writer *bufio.Writer) bool
```

此函数接受一个可写结构，传输器从远程传输器读取数据，并且写入到可写结构中。接收成功时此函数返回 `True`，在接收过程中出现任何异常均会返回 `False`。此函数接收方式与 *RecvBytes* 类似。如果你尚不了解接收原理，请查阅 传输器设计 。

服务器类

此部分详细介绍 *server* 类方法的使用方式。

server 类包含如下几个公有方法：

- *InitDB*： 初始化数据库函数，在创建服务器实例后调用以修复不存在的表。
- *CheckBroadCast*： 消息转发函数，此函数通常在一个独立的协程中执行，负责用户通讯。
- *Run*： 启动函数，将服务器实例开放在指定 IP 地址和端口。
- *Communicate*： 用户代理函数，每个在线用户均有一个对应的 *Communicate* 函数运行在独立协程中提供服务。
- *Login*： 连接认证函数，负责新连接的认证和转发。

公有方法

InitDB *InitDB* 函数位于 `server/server.go` 中，用于修复数据库中缺失的表。你可以在创建 *server* 类实例后 *s* 后调用 `s.InitDB()` 。

CheckBroadCast *CheckBroadCast* 函数位于 `server/server.go` 中，用于转发用户通讯消息。你可以在初始化 *server* 类实例 *s* 后通过 `go s.CheckBroadCast()` 启动一个守护线程来执行转发逻辑 。

Run *Run* 函数位于 `server/server.go` 中，用于启动服务器。你可以在初始化 *server* 类实例 *s* ，并确保所有运行前逻辑均已调用完成的情况下调用 `s.Run()` 来启动服务器。此函数内部由 `for` 循环阻塞，如果此函数退出则整个服务器程序都将结束。

Communicate *Communicate* 函数位于 `server/server.go` 中，用于为每个用户提供服务。此函数在服务器的 *Run* 方法中调用，每当一个新的连接请求到来，服务器将启动一个新的协程执行 *Communicate* 函数，并处理用户请求。

Communicate 会根据用户请求的类型决定将请求转交给 用户代理 ，或者创建一个新的用户代理。

Login *Login* 函数位于 `server/server.go` 中，用于认证用户请求的合法性。此函数在服务器的 *Communicate* 方法中调用，*Communicate* 根据 *Login* 的返回值决定如何处理用户请求。

1.2.3 其他材料

这部分文档包括：设计要点和变动记录

应用设计要点

顶点云服务器设计过程主要经过需求分析、协议设计、模块划分以及在代码实现过程中微调。

我们建议，如果您想使用顶点云的代码做一些自定义开发工作，最好能够维护如下几条准则，这是我们根据设计过程中遇到过的问题总结出的经验：

1. 先设计基本的模型结构，之后逐步实现业务逻辑，最后根据业务逻辑扩展模型结构。我们在设计顶点云的基础功能时，经常遇到需要调整结构的情况，因为先前考虑的不足，很多临时加入的判断条件需要模型修改才能支持。因此我们建议，除非您已经有非常完善的设计流程、框架，或者您要开发规模更大的功能，否则最好在业务逻辑实现前不要过早确定最终的模型。
2. 维持服务器和用户代理的独立。您应当理解我们将 *server* 和 用户代理 划分开的目的，降低模块间的耦合度是维持顶点云逐步推进的重要基础。
3. 维持逻辑转发和功能处理函数的独立。您应当理解我们将 用户代理 中各个函数使用包含 `switch` 块的公有方法转发的目的和意义。

4. 可以模仿但请不要学习。我们的功能处理函数大量使用了 `goto` 语句。虽然它能够让程序很快跳出错误，但我们更建议使用 `panic` 、 `recover` 机制。我们在设计功能处理函数时最终选择了使用 `goto` ，并不代表我们提倡这种方式。请您根据自己的喜好来决定是模仿还是使用更多更优的方式。

感谢您的阅读！到这里顶点云的应用程序服务器文档就结束了，您可以阅读 顶点云应用服务器变动记录 以了解顶点云应用程序服务器开发过程的变动历史，或者阅读 顶点云 *Web* 服务器 以了解顶点云 Web 服务器的相关信息。

顶点云应用服务器变动记录

版本	日期	变动
v0.01	*2016-10-17*	创建项目
v0.02	*2016-10-21*	完成基础模块划分与设计
v0.10	*2016-11-09*	完成服务器类设计，添加用户代理逻辑转发
v0.20	*2016-11-13*	完成大部分交互式命令执行函数
v0.40	*2016-11-23*	完成不具备文件传输能力的测试客户端和服务器
v0.60	*2016-12-03*	完成上传/下载功能
v0.61	*2016-12-04*	修正 MD5 值计算、RM 执行函数 bug
v0.80	*2016-12-07*	扩展数据库模型以适应 Web 服务器
v1.0	*2016-12-13*	和客户端交互测试超过 3 天，修正多处 bug，增加异地 登录的提醒、顶出操作
v1.0	*2016-12-20*	为所有代码添加注释
v1.2	*2016-12-24*	修复大量已知 bug

1.3 顶点云 Web 服务器

此部分文档将主要介绍顶点云 Web 服务器的框架和视图函数，文档分成几个部分，我推荐你按如下顺序阅读:

- 部署顶点云 *Web* 服务器
- *Web* 服务器全局配置
- 快速上手
- 框架分析
- 蓝本介绍
- 视图函数说明
- *Web* 服务器测试

顶点云的 Web 服务器使用 Flask 框架，Flask 的介绍不在本文档的范围内，如果你想了解 Flask 请移步:

- Flask 文档

1.3.1 开发者指南

这部分文档比较松散，首先简单介绍顶点云 Web 服务器部署和配置，之后将从框架、视图函数处理到部分实现细节说明顶点云 Web 服务器如何处理用户请求、实现文件共享、目录上传等。

部署顶点云 Web 服务器

顶点云的 Web 服务器使用 Python3 编写，基于 Flask 框架，在部署前请确保您已经安装 Python3、Pip、Pyvenv 等，并已正确设置环境变量。我们推荐的 Python3 版本为 Python 3.5.0。有关 Python3 和 Flask 的介绍不在本文档范围内，如果您尚不了解，请参阅以下相关链接：

- Python3 快速上手指南
- Pip 安装指南
- Pyvenv 使用指南
- Flask 文档

获取源码

顶点云的 Web 服务器源码托管在 GitHub 上，您可以使用 Git 克隆仓库或直接通过 GitHub 下载源码的压缩包。假设您熟悉 Git，请通过以下命令获取源码。

```
git clone https://github.com/Forec/zenith-cloud.git
cd zenith-cloud/web/
```

此时您应当已经进入顶点云 Web 服务器源码目录。

安装第三方支持

顶点云 Web 服务器使用到的所有第三方库均包含在需求文件 `web/requirements.txt` 中，您有两种方式部署。

一键部署脚本 顶点云 Web 服务器为 Linux 提供了一键部署脚本，它位于 `web/settings` 下。您可以运行以下命令。

```
cd settings
./setup.sh
```

如果您使用 Windows 系统，请参考下方的手动配置，或者如果您使用 Git Command，可以在 Git 的 Bash 命令行中运行 *setup.sh* 。

如果您的 Python 环境工作正常并且网络畅通，您应该可以看到终端中没有提示任何信息并且显示 部署完成字样。

手动部署 您可以选择手动部署顶点云 Web 服务器，流程如下：

```
mkdir venv
python3 -m venv venv/
source venv/bin/activate                // Windows 系统此步骤为 venv/Scripts/activate.bat
pip3 install -r requirements.txt --index-url https://pypi.douban.com/simple
pip3 install gunicorn --index-url https://pypi.douban.com/simple
python3 manager.py simple_init
deactivate
```

如果您的 Python 环境工作正常并且网络畅通，此时顶点云的 Web 服务器应当已经部署完毕。

运行测试

顶点云的 Web 服务器提供了一部分单元测试，您可以运行单元测试以确保环境配置正常。

进入 *web* 目录，运行 *python3 manage.py test* 。若测试结果显示通过则顶点云 Web 服务器部署成功。

顶点云的 Web 服务器可运行在任何主流体系结构计算机以及任何操作系统上。接下来请您阅读 *Web* 服务器全局配置 以根据您的系统配置顶点云 Web 服务器。

Web 服务器全局配置

在阅读以下内容前，请确保您已经按照 部署顶点云 *Web* 服务器 正确部署了顶点云 Web 服务器，并位于源码目录下（/path-to/zenith-cloud/web/）。

您可以根据您的环境修改源码目录下的 *config.py* 文件以配置服务器。

样例配置文件

您从 GitHub 仓库获取的源码中已经包含了一份默认的配置文件，这份配置文件中提供了一个基类 *Config* ，去掉注释后它的内容如下：

```
class Config:
 SECRET_KEY = os.environ.get('SECRET_KEY') or \
              '9d0e91f3372224b3ec7afec2' \
              '4313e745efcf00ba4a5b767b' \
              '35b17834d5f26efac197fd69' \
              'd881dd92e629dbfdc2f1fbf6'
 SQLALCHEMY_COMMIT_ON_TEARDOWN = True
 SQLALCHEMY_TRACK_MODIFICATIONS = True
 ZENITH_MAIL_SUBJECT_PREFIX = '[顶点云]'
 ZENITH_MAIL_SENDER = os.environ.get('ZENITH_MAIL_SENDER') or \
                      'cloud-storage@forec.cn'
 ZENITH_FILES_PER_PAGE = 10
 ZENITH_FOLLOWERS_PER_PAGE = 10
 ZENITH_COMMENTS_PER_PAGE = 10
 PROFILE_ZENITH_FILES_PER_PAGE = 6
 ZENITH_MESSAGES_PER_PAGE = 10
 ZENITH_TEMPFOLDER_LENGTH = 12
 ZENITH_PATH_SEPERATOR = '\\'
 ZENITH_FILE_STORE_PATH = 'G:\\Cloud\\'
 ZENITH_TEMPFILE_STORE_PATH = ZENITH_FILE_STORE_PATH + \
                          'TEMP' + ZENITH_PATH_SEPERATOR
 ZENITH_FOLDER_ZIP_SUFFIX = 'zenith'
 ZENITH_INVALID_INFFIX = ['//', '\\', '/', '..', '%', '^', '&',
                          '*', '$', '!', '+', '#']
 EMAIL_ADMIN ='forec@bupt.edu.cn'
 ZENITH_RANDOM_PATH_ELEMENTS = ['a', 'b', 'c', 'd', 'e', 'f', 'g',
                                'h', 'i', 'j', 'k', 'l', 'm', 'n',
                                'o', 'p', 'q', 'r', 's', 't', 'u',
                                'v', 'w', 'x', 'y', 'z', '1', '2',
                                '3', '4', '5', '6', '7', '8', '9',
                                '0', 'A', 'B', 'C', 'D', 'E', 'F',
                                'G', 'H', 'I', 'J', 'K', 'L', 'M',
                                'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                                'U', 'V', 'W', 'X', 'Y', 'Z']
 ZENITH_VALID_THUMBNAIL = ['.jpg', '.png', '.ico', '.jpeg']
 ZENITH_VALID_THUMBNAIL_SIZE = 512 * 1024
```

样例配置文件中，每项均有对应注释，您也可以查看下面的 样例文件详细解释 了解每一项的具体功能。

样例文件详细解释

配置文件中 *Config* 类的每一项对应配置如下：

1. `SECRET_KEY`： 用于服务器生成 token 使用的密钥，不可泄露，应当设置在服务器部署系统的环境变量中
2. `SQLALCHEMY_COMMIT_ON_TEARDOWN`： 为 `True` 时服务器终止运行时向数据库提交变动
3. `SQLALCHEMY_TRACK_MODIFICATIONS`： 为 `True` 时数据库将追踪服务器对数据的改动
4. `ZENITH_MAIL_SUBJECT_PREFIX`： 服务器向用户发送的验证邮件的主题前缀
5. `ZENITH_MAIL_SENDER`： 服务器向用户发送验证邮件使用的邮箱
6. `ZENITH_FILES_PER_PAGE`： Index 页面每页显示的文件数量
7. `ZENITH_FOLLOWERS_PER_PAGE`： 关注着界面每页显示的用户数量
8. `ZENITH_COMMENTS_PER_PAGE`： 文件详细页面每页显示的评论数量
9. `PROFILE_ZENITH_FILES_PER_PAGE`： 用户资料页每页显示的文件数量
10. `ZENITH_MESSAGES_PER_PAGE`： 用户消息页面每页显示的消息数量
11. `ZENITH_TEMPFOLDER_LENGTH`： 服务器在临时存储文件时生成的随机目录名长度
12. `ZENITH_PATH_SEPERATOR`： 服务器所属文件系统的目录分隔符，Windows为 *，*Linux* 为 */*
13. `ZENITH_FILE_STORE_PATH`： 服务器用于存储用户文件的路径
14. `ZENITH_TEMPFILE_STORE_PATH`： 服务器生成的随机目录的根路径，默认为文件存储路径下的 TMEP 文件夹
15. `ZENITH_FOLDER_ZIP_SUFFIX`： 用户上传目录压缩包时使用的后缀，使用此后缀的文件会被视作一个目录
16. `ZENITH_INVALID_INFFIX`： 此列表中的字符不能出现在用户文件名中，否则视为不合法
17. `EMAIL_ADMIN`： 管理员账户使用的邮箱
18. `ZENITH_RANDOM_PATH_ELEMENTS`： 服务器生成的随机路径包含的元素
19. `ZENITH_VALID_THUMBNAIL`： 服务器允许用户上传的头像后缀名
20. `ZENITH_VALID_THUMBNAIL_SIZE`： 服务器允许用户上传的最大头像大小

Config 类仅仅是配置类的基类，你需要扩展此类才可完成配置。在 *config.py* 文件中，存在一些默认的 *Config* 子类，如 *LinuxConfig* 、 *WindowsConfig* 。

下面简单介绍 *LinuxConfig* 并以一个环境为例讲解如何配置。

LinuxConfig 类的内容如下，它是我（Forec）在我的云主机部署 Web 服务器时使用的配置类：

```
class LinuxConfig(Config):
 ZENITH_PATH_SEPERATOR = '/'     # 服务器所属文件系统的目录分隔符
 ZENITH_FILE_STORE_PATH = '/root/work/cloud/Cloud/'   # 服务器存储用户文件的路径
 ZENITH_TEMPFILE_STORE_PATH = ZENITH_FILE_STORE_PATH + \
                              'TEMP' + ZENITH_PATH_SEPERATOR
 ZENITH_SERVER_ADDRESS = 'cloud.forec.cn' # 服务器部署的域名/IP地址
 SERVER_NAME = ZENITH_SERVER_ADDRESS
 SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(basedir, 'work.db')
 MAIL_SERVER = 'smtp.exmail.qq.com'
```

```
 MAIL_PORT = 25 # SSL is 465
 MAIL_USE_TLS = True
 MAIL_USERNAME = "cloud-storage@forec.cn"
 MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
```

可以看出， *LinuxConfig* 类重写了 *Config* 类的几项，同时添加了几个新的选项。新选项介绍如下：

1. `ZENITH_SERVER_ADDRESS`： 服务器部署使用的域名/IP地址
2. `SERVER_NAME`： Flask 中的 *url_for* 函数使用的服务器名，通常保持和 *SERVER_NAME* 一致
3. `SQLALCHEMY_DATABASE_URI`： 服务器使用的数据库所在的路径
4. `MAIL_SERVER`： 服务器发送邮件使用的邮箱服务器
5. `MAIL_PORT`： 服务器使用 stmp 协议的端口号，通常为 25。使用 SSL 时设置为 465 但这取决于 *MAIL_SERVER* 是否支持 SSL
6. `MAIL_USE_TLS`： 是否启用安全连接发送邮件，通常设置为 `True`
7. `MAIL_USERNAME`： 服务器发送邮件使用的邮箱帐号，通常和 *ZENITH_MAIL_SENDER* 保持一致
8. `MAIL_PASSWORD`： 服务器发送邮件使用的邮箱帐号的密码，通常保存在环境变量中

下面通过一个实例环境解释如何配置。

例如，在安装 Ubuntu 16.04 的主机上部署 Web 服务器，可参考的配置文件如下（使用扩展类）：

```
class MyConfig(Config):
 ZENITH_PATH_SEPERATOR = '/'
 ZENITH_FILE_STORE_PATH = '/home/cloud/'  # 将用户上传文件存储在 /home/cloud 中
 ZENITH_TEMPFILE_STORE_PATH = ZENITH_FILE_STORE_PATH + \
                              'TEMP' + ZENITH_PATH_SEPERATOR
 ZENITH_SERVER_ADDRESS = 'myaddress.my.io' # 自定义的域名，你需要先购买此域名并且映射到部署主机上
 SERVER_NAME = ZENITH_SERVER_ADDRESS
 SQLALCHEMY_DATABASE_URI = '/usr/local/cloud/mydb.db'
     # 设置数据库为 /usr/local/cloud/mydb.db
 MAIL_SERVER = 'smtp.163.com'         # 使用 163 邮箱
 MAIL_PORT = 25
 MAIL_USE_TLS = True
 MAIL_USERNAME = "mycloud@163.com"
 MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD') or '123456'
     # 在环境变量中添加密码，若环境变量未找到对应值则使用 123456
```

添加自定义配置类到表

在已经定义了自定义配置类后，你需要将自定义配置类添加到表驱动中以使 工厂方法 能根据我的配置类生成服务器实例。

在 *config.py* 中，有一个名为 *config* 的字典如下：

```
config = {
 'development' : DevelopmentConfig,       # 开发环境
 'linux': LinuxConfig,                    # 提供的 Linux 模板环境
 'windows': WindowsConfig,                # 提供的 Windows模板环境
 'testing' : TestingConfig,               # 测试环境
 'default' : DevelopmentConfig            # 默认为开发环境
 }
```

你需要添加自己的自定义配置类到此表中，如添加 `'myconfig': MyConfig` 。之后，修改 *manage.py* 中的第 21 行 `app = create_app('default')` 为 `app = create_app('myconfig')` 即可。

接下来请您阅读 快速上手 。

快速上手

此部分文档将带领您在一个假想的纯净环境中部署、配置、扩展自定义功能并启动顶点云 Web 服务器。

假想环境

有一天我意外获得了一台免费的 CVM，而我恰巧获得了一次得到顶点云 Web 服务器源码的机会。这台云主机使用的操作系统为 CentOS 7.2，公网 IP 地址为 `123.123.123.123`，已安装好 Python3、Pip以及 Pyvenv 并配置了环境变量。下面的指令均通过 SSH 远程操作。

```
git clone https://github.com/Forec/zenith-cloud.git
cd zenith-cloud/web/
```

我已经获得了顶点云应用程序服务器的源码，接下来使用一键配置脚本部署环境。

```
cd settings
./setup.sh
```

很高兴看到配置脚本通知我部署完成。接下来测试一下代码是否能够在本地机器通过测试。

```
cd ..
source venv/bin/activate  # Windows 下请执行 venv/Scripts/activate.bat
python manage.py test
```

非常顺利！测试脚本告诉我所有测试均已完成，顶点云 Web 服务器各个基础模块能够在这台服务器上运转正常。

针对假想环境修改配置文件

鉴于顶点云提供的默认配置仅适用于 Forec 的史诗级笔记本，下面根据这台服务器的情况修改配置文件。编辑 *config.py* :

```
nano config.py
```

我拷贝了一份 `LinuxConfig` 并重命名该子类为 `MyConfig`，然后根据如下考虑对 `MyConfig` 做了一定修改:

- 我想将用户上传的文件存储到 `/usr/local/cloud`，因此我修改 `ZENITH_FILE_STORE_PATH = "/usr/local/cloud"`
- 我觉得顶点云默认使用的 SQLITE 数据库很方便，并且放置在源码根目录下也没什么问题，因此我决定保留默认配置中的 `SQLALCHEMY_DATABASE_URI`
- 我想让世界上任何一个角落均能访问我的顶点云服务器，因此我修改 `ZENITH_SERVER_ADDRESS = '123.123.123.123'`
- 我要让顶点云 Web 服务器用我的邮箱发送认证邮件。我的邮箱是 mymail@gmail.com，因此我修改如下部分:
- `MAIL_SERVER = 'smtp.gmail.com'`
- `MAIL_PORT = 25`
- `MAIL_USE_TLS = True`
- `MAIL_USERNAME = "mymail@gmail.com"`

- 我觉得邮箱的密码还是不要放在代码中比较好，因此我向环境变量添加了 MAIL_PASSWORD 值并保留了 MAIL_PASSWORD 的设置

看起来配置文件没什么值得修改的了，我决定按下 CTRL+X 保存配置文件，顺便检查一下新定义的配置类：

```
class MyConfig(Config):
 ZENITH_PATH_SEPERATOR = '/'
 ZENITH_FILE_STORE_PATH = '/usr/local/cloud'
 ZENITH_TEMPFILE_STORE_PATH = ZENITH_FILE_STORE_PATH + \
                              'TEMP' + ZENITH_PATH_SEPERATOR
 ZENITH_SERVER_ADDRESS = '123.123.123.123' # 服务器部署的域名/IP地址
 SERVER_NAME = ZENITH_SERVER_ADDRESS
 SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(basedir, 'work.db')
 MAIL_SERVER = 'smtp.gmail.com'
 MAIL_PORT = 25 # SSL is 465
 MAIL_USE_TLS = True
 MAIL_USERNAME = "mymail@gmail.com"
 MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
```

添加自定义类到表驱动

我决定按照 添加自定义配置类到表 中的说法将我的自定义配置类添加到表驱动中。

向 *config.py* 的 *config* 字典中添加 'myconfig': MyConfig 后如下：

```
config = {
 'development' : DevelopmentConfig,       # 开发环境
 'linux': LinuxConfig,                    # 提供的 Linux 模板环境
 'windows': WindowsConfig,                # 提供的 Windows模板环境
 'testing' : TestingConfig,               # 测试环境
 'default' : DevelopmentConfig,           # 默认为开发环境
 'myconfig' : MyConfig                                  # 自定义添加的配置类
 }
```

之后修改 *manage.py* 的第 21 行为：

```
app = create_app('myconfig')
```

启动服务器

顶点云 Web 服务器可通过两种方式启动。我们推荐使用 *settings* 目录下的启动脚本，启动脚本使用 *gunicorn* 能够提高服务器的并发能力。

一键启动 *web/settings* 目录提供了顶点云 Web 服务器的启动脚本，您可以运行 *run.sh* （Linux 系统）或 *run.bat* （Windows 系统）来启动服务器。默认会开启在本机（127.0.0.1）的 5001 端口。您可以修改启动脚本中的 IP 地址和端口号。

手动启动 您也可以选择手动控制服务器的启动。通常在 Debug 情况下使用此方式，因为 Flask 对并发请求的原生支持并不很令人满意。

```
source venv/bin/activate          # Windows 下请执行 venv/Scripts/activate.bat
python manage.py runserver        # 您可以指定 -h 和 -p 参数，分别代表开放服务器的IP 地址和端口号
```

现在您可以从本机的浏览器访问您的服务器了。

扩展自定义功能

不得不说 Forec 的设计实在是太简陋了，为什么用户无法注册！幸好我学习过 Flask 框架，也许我应该自己添加这个功能?

在阅读了 框架分析 后，我了解了整个顶点云 Web 服务器的结构，下面我准备添加这个简单的功能。

进入 *web/app/auth* 目录并编辑 *views.py* :

```
cd web/app/auth
nano views.py
```

我在源码的 56 行发现了一句注释，原来默认的顶点云提供了注册接口，但将注册部分屏蔽掉了，反馈给用户的仅仅是展示界面。注册的视图函数如下所示。

```
@auth.route('/register', methods = ['GET', 'POST'])
def register():
 # 展示状态，禁止注册
 return render_template('auth/testing.html', _external=True)

 # form = RegistrationForm()
 # if current_user.is_authenticated:
 #     flash('您已经登陆，登陆状态下无法注册')
 #     return redirect(url_for('main.index', _external=True))
 # if form.validate_on_submit():
 #     user = User(email = form.email.data,
 #                 nickname = form.nickname.data,
 #                 password = form.password.data)
 #     db.session.add(user)
 #     db.session.commit()
 #     token = user.generate_confirmation_token()
 #     send_email(user.email,
 #                '确认您的帐户',
 #                'auth/email/confirm',
 #                user=user,
 #                token=token)
 #     flash('一封确认邮件已经发送到您填写的邮箱，'
 #           '请查看以激活您的帐号')
 #     login_user(user)
 #     return redirect('http://mail.'+user.email.split('@')[-1])
 # return render_template('auth/register.html', form=form)
```

我决定开放注册接口，因此我将被注释的部分取消注释，将视图函数中的第一句 *return* 删除。

```
@auth.route('/register', methods = ['GET', 'POST'])
def register():
 # 展示状态，禁止注册
 # return render_template('auth/testing.html', _external=True)

 form = RegistrationForm()
 if current_user.is_authenticated:
     flash('您已经登陆，登陆状态下无法注册')
     return redirect(url_for('main.index', _external=True))
 if form.validate_on_submit():
     user = User(email = form.email.data,
                 nickname = form.nickname.data,
                 password = form.password.data)
     db.session.add(user)
     db.session.commit()
```

```
    token = user.generate_confirmation_token()
    send_email(user.email,
               '确认您的帐户',
               'auth/email/confirm',
               user=user,
               token=token)
    flash('一封确认邮件已经发送到您填写的邮箱，'
          '请查看以激活您的帐号')
    login_user(user)
    return redirect('http://mail.'+user.email.split('@')[-1])
return render_template('auth/register.html', form=form)
```

我重新启动了服务器，现在注册接口已经打开。

接下来请您阅读 框架分析 。

框架分析

此部分文档主要介绍顶点云 Web 服务器的框架结构。

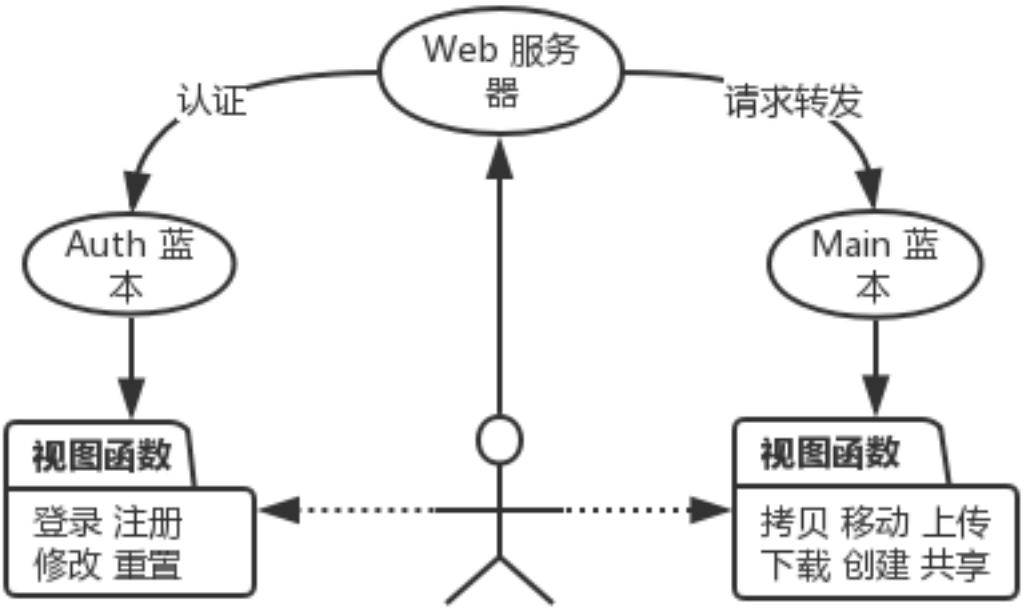
顶点云服务器源码文件结构如下：

```
- web
  - app
    - auth
    - main
    - static
      - thumbnail
    - templates
    - ...
    - models.py
  - settings
  - config.py
  - manage.py
  - work.db
```

以上几个目录对应的功能如下：

- *auth*： 对应 *Auth* 蓝本 ，用于处理用户注册、登录等需要特权的请求。
- *main*： 对应 *Main* 蓝本 ，用于处理用户大部分不需要特权的请求，以及主要的功能实现。
- *static*： 对应 Flask 框架中存储静态文件的目录，此目录存储 Web 服务器使用的图片、js、css等文件。
- *static/thumbnail*： 此目录存储用户自定义的头像文件。
- *template*： 此目录存储 Web 服务器渲染网页使用的模板。
- *models.py*： 对应 模型介绍 中介绍的各类模型。
- *config.py*： 对应 *Web* 服务器全局配置 中的配置文件。
- *manage.py*： 管理服务器的配置文件，包含了一系列自定义命令，你可以查看 管理器 以了解更多。

各模块之间关系如下图所示：



管理器

manage.py 管理着顶点云 Web 服务器。它提供了很多操作服务器的基础命令，如启动、初始化、测试等，你可以通过 *python manage.py <command>* 来运行不同命令。默认的顶点云 Web 服务器管理器提供的命令有：

- *shell*： 启动交互式 Python 命令行并自动导入顶点云 Web 服务器的各类模块
- *db*： 数据库迁移类命令，如 *db migrate* 、 *db init* 等
- *test*： 启动 *web/tests* 目录下的单元测试
- *init*： 初始化数据库，随机生成用户和数据
- *simple_init*： 简单初始化数据库，只加入五个特定的用户

工厂方法

web/app/__init__.py 中包含了 Web 服务器的工厂方法，当你通过 *manage.py* 执行命令时， *manage.py* 会调用工厂方法生成一个应用实例。

工厂方法名为 *create_app* ，正如我们在 扩展自定义功能 中介绍的那样，修改 *manage.py* 的第 13 行实际是修改了工厂方法的参数。这个工厂方法根据传入的参数从 *Web* 服务器全局配置 中寻找对应表项（配置类），并使用该配置类生成服务器实例。

接下来请您阅读 蓝本介绍 。

蓝本介绍

此部分文档主要介绍顶点云 Web 服务器中的两个蓝本： *main* 以及 *auth* ，以及如何创建新的自定义蓝本。

Auth 蓝本

此蓝本主要用于授权用户，如注册、登录、重置密码等。蓝本位于 *web/app/auth* 目录下，结构为：

```
- auth
  - __init__.py
  - forms.py
  - views.py
```

上面三个文件中， *__init__.py* 用于初始化蓝本 Auth，*forms.py* 定义了 Auth 蓝本视图函数需要使用的表单类，*views.py* 具体处理转发给 Auth 蓝本的请求。

下面简单介绍 *forms.py* 中提供的表单，对于 *views.py* 中的视图函数，请查阅 *Auth* 蓝本视图函数 。

forms.py 提供了如下表单：

- *LoginForm*： 用户登录界面填写的表单
- *RegistrationForm*： 用户注册界面填写的表单，顶点云默认关闭了注册接口，你需要参考 扩展自定义功能 来开启此接口
- *ChangePasswordForm*： 用户修改密码时填写的表单
- *ChangeEmailForm*： 用户修改邮箱时填写的表单
- *PasswordResetForm*： 用户重置密码时填写的表单
- *PasswordResetRequestForm*： 用户申请重置密码时填写的表单

Main 蓝本

此蓝本主要用于处理用户服务请求。蓝本位于 *web/app/main* 目录下，结构为：

```
- main
  - __init__.py
  - forms.py
  - views.py
  - errors.py
```

上面三个文件中， *__init__.py* 用于初始化蓝本 Main，*forms.py* 定义了 Main 蓝本视图函数需要使用的表单类，*views.py* 具体处理转发给 Main 蓝本的请求，*errors.py* 处理用户遇到的错误并使用服务器模板渲染错误界面。

下面简单介绍 *forms.py* 中提供的表单，对于 *views.py* 中的视图函数，请查阅 *Main* 蓝本视图函数 。

forms.py 提供了如下表单：

- *EditProfileForm*： 用户编辑个人资料界面填写的表单
- *EditProfileAdminForm*： 管理员编辑其他用户个人资料界面填写的表单
- *UploadForm*： 用户上传文件时填写的表单
- *CommentForm*： 用户评论文件时填写的表单
- *SearchForm*： 用户搜索文件时填写的表单
- *FileDeleteConfirmForm*： 用户确认删除文件时填写的表单
- *ChatForm*： 用户聊天界面的对话框
- *SetShareForm*： 用户设置文件共享和提取码时填写的表单
- *ConfirmShareForm*： 用户对其他用户的共享文件执行 Fork、下载等操作前填写提取码的表单

- *NewFolderForm*： 用户创建新文件夹时填写的表单

自定义蓝本

关于蓝本的概念不属于本文档讨论范围，如果您尚不了解，可以参考 Flask 文档中关于蓝本的部分 。

对于顶点云，如果您需要添加自定义蓝本，请在目录 *web/app* 下创建新的蓝本目录并添加您的自定义路由。

接下来请您阅读 模型介绍 。

模型介绍

此部分文档主要介绍顶点云 Web 服务器使用的数据库格式和模型。

数据库

顶点云 Web 服务器使用的数据库与 顶点云应用程序服务器 使用的数据库兼容。您可以直接阅读 数据库 来查看数据库格式。

内置自定义类

此部分文档主要介绍顶点云 Web 服务器定义的几个模型。用户模型的定义均位于 *web/app/models.py* 中，包括:

- *User*： 用户类
- *Message*： 用户聊天消息类，与 表 *cmessage* 保持一致
- *Follow*： 第三方表，用于存储用户之间的多对多关注关系
- *Permission*： 用户权限模型
- *Role*： 用户身份模型
- *CFILE*： 实体文件模型，与 表 *cfile* 保持一致
- *File*： 用户资源记录模型，与 表 *ufile* 保持一致
- *Comment*： 用户评论类
- *Pagination*： 自定义文件分页模型，用于在列表过长时分页显示，该模型允许从列表构造分页器，可替代 SQLAlchemy 提供的 *Pagination* 类
- *AnonymousUser*： 匿名用户模型，提供给 *flask_login* 作为未登录用户实例

下面将简单介绍每个模型提供的方法。

用户类 *User* 模型同时继承了 *UserMixin* 和 SQLAlchemy 数据库模型，它具有如下元素:

```
# 用户 id
uid = db.Column(db.Integer, primary_key=True)
email = db.Column(db.String(64),unique=True)
# 用户密码的加盐哈希值
password_hash = db.Column(db.String(32))
# 用户创建时间
created = db.Column(db.DateTime, default = datetime.utcnow)
# 用户是否已激活邮箱
```

```
confirmed = db.Column(db.Boolean, default= False)
# 用户昵称
nickname = db.Column(db.String(64))
# 用户头像链接
avatar_hash = db.Column(db.String(32))
# 用户个人介绍
about_me = db.Column(db.Text)
# 与 created 相同，adapter
member_since = db.Column(db.DateTime,
                         default = datetime.utcnow)
# 上次登录时间
last_seen = db.Column(db.DateTime,
                      default = datetime.utcnow)
# 用户积分
score = db.Column(db.Integer, default = 20)
# 用户角色 id（管理员/审核员/普通用户等）
role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
# 用户拥有的文件，外链 File 表
files = db.relationship('File', backref='owner', lazy = 'dynamic')
# 用户发布的评论，外链 Comment 表
comments = db.relationship('Comment',backref='author', lazy='dynamic')
# 此用户关注的人，外链 Follow 表
followed = db.relationship('Follow',
                           # 指定外键
                           foreign_keys= [Follow.follower_id],
                           # 回调变量
                           backref = db.backref('follower', lazy='joined'),
                           lazy = 'dynamic',
                           # 当用户删除时连带删除全部记录
                           cascade='all, delete-orphan')
# 此用户的关注者，外链 Follow 表
followers = db.relationship('Follow',
                           foreign_keys= [Follow.followed_id],
                           backref = db.backref('followed', lazy='joined'),
                           lazy = 'dynamic',
                           cascade='all, delete-orphan')
# 用户发送过的消息
sendMessages = db.relationship('Message',
                               backref='sender',
                               lazy='dynamic',
                               foreign_keys = [Message.sendid])
# 用户接收到的消息
recvMessages = db.relationship('Message',
                               backref='receiver',
                               lazy='dynamic',
                               foreign_keys = [Message.targetid])
# 用户已使用的网盘空间，单位为字节
used = db.Column(db.Integer, default=0)
# 用户最大网盘空间，单位为字节，默认 256 MB
maxm = db.Column(db.Integer, default=256*1024*1024)
```

User 类具有如下方法：

- *get_id*： 获取用户 id
- *verify_password*： 验证密码是否正确
- *generate_confirmation_token*： 生成用户邮箱验证 token
- *generate_email_change_token*： 生成修改邮箱 token

- *generate_reset_token*： 生成重置密码 token
- *generate_delete_token*： 生成删除文件 token
- *reset_password*： 用户验证重置密码的 token
- *confirm*： 用户验证邮箱激活的 token
- *change_email*： 用户验证修改邮箱的 token
- *delete_file*： 用户验证删除文件的 token
- *generate_copy_token*： 生成文件复制操作的 token
- *copy_token_verify*： 验证用户文件复制的 token
- *generate_move_token*： 生成用户文件移动的 token
- *move_token_verify*： 验证用户文件移动的 token
- *generate_fork_token*： 生成用户 Fork 文件 token
- *fork_token_verify*： 校验用户 Fork 文件 token 的合法性
- *generate_download_token*： 生成用户下载的 token
- *download_token_verify*： 验证用户下载 token 的合法性
- *generate_view_token*： 生成共享文件查看 token
- *view_token_verify*： 验证查看其他用户文件的 token 合法性
- *gravatar*： 获取用户头像链接，若存在自定义头像则返回自定义头像链接，否则从 gravatar 获取
- *can*： 用户是否具有某项权限
- *is_administrator*： 用户是否为管理员
- *ping*： 更新用户最近登录时间
- *follow*： 关注某个用户
- *unfollow*： 取消关注某个用户
- *is_following*： 是否已关注某个用户
- *is_followed_by*： 是否被某用户关注
- *followed_files*： 用户关注的人发布的共享文件
- *generate_fake*： 生成随机用户

可以看出，*User* 类的多数方法都用于处理需要鉴别用户身份的请求，包括生成 token 、验证 token 以及在验证通过后执行相应的处理。以删除资源为例， *Main* 蓝本 中的 *delete_do* 视图函数将请求转发给当前用户，由当前用户验证 token 并执行删除操作，删除操作定义在方法 *delete_file* 中。

详细方法的参数请查看 *web/app/models.py* ，代码中给出了详细的注释。

实体文件 CFILE 类 *CFILE* 类和 表 *cfile* 保持一致，同时提供了如下 3 个方法:

- *md5FromFile(filepath)*： 计算指定路径的文件的 MD5 值，计算方法与 *MD5* 计算 相同。
- *makeFile(filepath, size)*： 在指定路径创建一个指定大小的随机内容的文件。
- *generate_fake(count)*： 在 *config.py* 中指定的 *ZENITH_FILE_STORE_PATH* 下生成指定数量个随机文件。

权限 *Permission* 类指定了如下几种权限：

```
FOLLOW = 0x01   # 关注其他用户
COMMENT = 0x02   # 评论文件
PUBLIC_FILES = 0x04   # 发布文件
MODERATE_COMMENTS = 0x08   # 管理评论
MODERATE_FILES = 0x10   # 管理文件
ADMINISTER = 0x80   # 管理员
```

身份 中的 *permissions* 元素代表该身份具有的权限，将 *permissions* 与指定权限做与操作，若结果为 1 则代表该身份具有指定权限。

身份 *Role* 类定义了用户的不同身份，顶点云默认提供的身份和对应权限有：

```
roles = {
 'Uncheck_user': (0x00, True),
 'User': (Permission.FOLLOW |
          Permission.COMMENT |
          Permission.WRITE_ARTICLES, False),
 'Moderator_comments': (Permission.FOLLOW |
          Permission.COMMENT |
          Permission.WRITE_ARTICLES |
          Permission.MODERATE_COMMENTS, False),
 'Moderator_tasks':(
          Permission.COMMENT |
          Permission.WRITE_ARTICLES |
          Permission.MODERATE_FILES, False),
 'Administrator': (0xff, False)
}
```

Moderator_tasks 作为可扩展身份，默认的顶点云暂时没有启用。默认顶点云支持 *Uncheck_user* （邮箱未认证的用户）、*User* （普通用户）、*Moderator_comments* （评论管理员）以及 *Administrator* （超级管理员）。

接下来请您阅读 视图函数说明 。

视图函数说明

此部分文档简要介绍顶点云 Web 服务器提供的视图函数，将按 *Auth* 蓝本 和 *Main* 蓝本 分成两个部分。

Auth 蓝本视图函数

此部分视图函数来自 *Auth* 蓝本 ，位于文件 *web/app/auth/views.py* 中。共包含如下视图函数：

视图函数名	功能
rules	提供了 “注册须知” 界面的入口
login	提供了登录界面入口
logout	提供了登出操作，登出后默认重定向到登陆入口
register	提供了注册入口
confirm	提供了用户注册邮箱激活入口，根据激活链接尾部的 *token* 校验用户是否合法
resend_confirmation	用于在用户未收到激活邮件时重发
change_password	为用户修改密码的视图函数
change_email_request	为用户重置邮箱请求入口
change_email	用于验证用户重置邮箱后的激活链接
password_reset_request	为用户重置密码入口
password_reset	用于验证用户重置密码请求 token 的合法性
secure_center	返回安全中心界面
before_request	注册了用户未激活邮箱时的跳转接口
unconfirmed	提供了未验证的界面

Main 蓝本视图函数

此部分视图函数来自 *Main* 蓝本，位于文件 *web/app/main/views.py* 中。共包含如下视图函数：

视图函数名	功能
moderate	提供了 “管理” 界面的入口
home	提供了顶点云介绍界面的入口
index	服务器主页入口点，将展示用户共享的资源描述
show_all	将用户 cookie 中的 show_followed 选项复位
show_followed	将用户 cookie 中的 show_followed 选项置位
user	服务器用户资料界面入口
edit_profile	为当前已登陆用户提供编辑用户资料入口
edit_profile_admin	为具有管理员权限的用户提供编辑任意用户资料的入口
file	显示具体的资源信息
follow	为用户关注其它用户提供了跳板，若关注成功则跳转到被关注用户的资料界面
unfollow	为用户提供了 follow 的逆操作
followers	显示某用户关注者的界面入口
followed_by	显示某用户关注的人的入口
delete_file	提供了删除文件界面的入口
delete_file_confirm	对用户的删除操作进行确认并执行
edit_file	为用户编辑文件信息（重命名、修改描述）界面提供了入口
moderate_comments	为评论管理员提供了审核、屏蔽评论的界面入口
moderate_comments_disable	为评论管理员提供了将某条评论屏蔽的入口
moderate_comments _disable_own	为文件所有者提供了管理自己文件下评论的权限
moderate_files	为管理员提供了修改、删除、设置任意文件状态的入口
moderate_files_delete	为管理员用户提供了删除文件的入口，管理员也可以通过用户的 *delete_file* 来 实现此功能
messages	用户消息界面的入口
cloud	提供了“我的云盘”界面入口
view_share_folder_entry	提供了用户访问其他用户共享目录的认证入口
view_do	为用户提供了访问其他用户共享目录的界面
download	为设有分享密码的文件/目录提供了验证界面，否则直接跳转到 *download_do* 入口
download_do	验证用户的下载请求是否合法，传入的 token 将由当前用户进行解析，若失败则返 回 40
upload	为用户上传文件界面提供入口
copy	为用户复制文件/目录界面提供了入口

Tabl

视图函数名	功能
copy_check	验证用户复制操作的入口，当前用户根据传入的 token 验证合法性并向数据库执行 写入操
move	为用户移动文件/目录界面提供了入口
move_check	验证用户移动操作的入口，当前用户根据传入的 token 验证合法性并向数据库执行 写入操
fork	提供了一个简单的验证界面，用户需要输入要 fork 的文件/目录的提取码
fork_do	为用户 Fork 文件/目录界面提供了入口
fork_check	fork_check 是验证用户 Fork 操作的入口，当前用户根据传入的 token 验证合法 性并向数据
download_do	验证用户的下载请求是否合法，传入的 token 将由当前用户进行解析，若失败则返 回 40
newfolder	为用户创建目录提供了入口，用户可在云盘界面、复制、移动、Fork 的同时创建新 文件
delete_message	为用户提供了删除聊天消息的入口，一条聊天消息的接收/发送方均可删除消息，但 删除
recall_message	为用户提供了发送消息撤回功能，功能与 QQ 的消息撤回相同，超过 2 分钟的消息 无法
chat	提供了用户聊天入口
close_chat	为用户提供了一次性忽略某个用户所有未读消息的功能
delete_chat	为用户提供了一次性删除某个用户所有聊天记录的功能
set_share	为用户提供了设置文件共享属性的功能，提供了简单的设置密码界面
set_private	set_share 的逆操作，将某个文件/目录及关联的目录/文件全部设为私有

接下来请您阅读 *Web* 服务器测试 。

Web 服务器测试

此部分文档主要介绍顶点云 Web 服务器的单元测试。你可以在 *web* 目录下运行 *python manage.py test* 运行全部单元测试。

单元测试主要覆盖用户登录、注册、样例文件修改、拷贝、移动、Fork、下载等，各单元测试文件覆盖视图函数如下表所示。

单元测试文件名	测试覆盖视图函数
test_basic.py	测试环境是否正常，服务器是否启动
test_client.py	*copy*、*copy_check*、 *move*、*move_check*、 *fork*、*fork_do*、*fork_check*、*fork_verify*、*newfolder*、 *set_share*、*set_private*、 *messages*、*chat*、*delete_message*、*recall_message*、*reset_password_request*、*reset_password*

到这里顶点云 Web 服务器的开发者文档部分就结束了，您可以阅读 *Web* 服务器设计要点 来查看我们对设计者提出的几点建议，也可以阅读 顶点云应用程序服务器 以了解顶点云 应用程序服务器的相关信息。

1.3.2 其他材料

这部分文档包括：设计要点和变动记录

Web 服务器设计要点

顶点云 Web 服务器的设计主要复用了应用程序服务器的设计。

我们建议，如果您想使用顶点云 Web 服务器的代码做一些自定义开发工作，最好能够维护如下几条准则，这是我们根据编写过程中遇到过的问题总结出的经验：

1. 先设计蓝本，以及蓝本的职责，再决定蓝本提供的视图函数。

2. 视图函数应尽量避免重复代码，应尽量将共同的代码部分抽象，如使用一个表驱动函数池。在设计顶点云 Web 服务器时，我没有很好的组织代码，因此无论是应用程序服务器还是 Web 服务器，均出现了一处 bug 多处修改的困境。
3. 将需要根据环境改变的变量写入到配置文件中，如果是扩展功能的配置项，应创建新的子类并包含此项。用户可以选择多重继承来实现扩展功能的组合。

感谢您的阅读！到这里顶点云的 Web 服务器文档就结束了，您可以阅读 顶点云 *Web* 服务器变动记录 以了解顶点云 Web 服务器开发过程的变动历史，或者阅读 顶点云应用程序服务器 以了解顶点云应用程序服务器的相关信息。

顶点云 Web 服务器变动记录

版本	日期	变动
v0.01	*2016-12-02*	创建项目
v0.10	*2016-12-04*	完成多数视图函数，剩余上传、共享、下载以及云盘界面
v0.20	*2016-12-05*	添加图标、文件类型、邮件推送等
v0.22	*2016-12-06*	完成消息相关功能
v0.40	*2016-12-07*	为部分视图函数增加 token 验证，增加移动、复制、Fork 等视图函数处理中的重名检测。更新用户空间使用量、更 新随机路径生成规则
v0.41	*2016-12-08*	修正文件操作列表、用户资料页显示数目有误等 bug
v0.60	*2016-12-09*	增加单个文件下载
v0.80	*2016-12-10*	增加其他用户共享目录查看，增加文件关键词搜索
v0.90	*2016-12-11*	添加新建文件夹
v0.91	*2016-12-12*	增加单个文件上传，修正英文界面为中文
v0.97	*2016-12-13*	替换部分渲染模板
v0.98	*2016-12-14*	替换全部渲染模板
v1.0	*2016-12-17*	增加配置、运行脚本
v1.0	*2016-12-20*	增加全部注释
v1.0	*2016-12-21*	添加开发者文档
v1.2	*2016-12-26*	完善测试，修正一些前端错误

CHAPTER 2

许可证信息

此部分标明顶点云使用的开源许可证以及授权声明。

2.1 许可证信息

顶点云使用的许可证可在 此处 查看，内容如下：

```
Copyright (c) 2015-2017, Forec <forec@bupt.edu.cn>

Permission to use, copy, modify, and/or distribute this code for
any purpose with or without fee is hereby granted, provided that
the above copyright notice and this permission notice appear in
all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL
THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN     ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

无论是否用于商业目的，你均可以根据自己的需要使用、修改、拷贝顶点云的代码，但必须保证以上版权声明和许可证信息出现在所有顶点云代码的副本中。