moCloud设计文档

# 历史

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Description** |
| V1.0.0 | WuJinlei | 20180211 | Create |
| V1.0.1 | WuJinlei | 20180212 | 增加应用场景--服务器端的相关内容； |
| V1.0.2 | WuJinlei | 20180224 | 增加一些通信数据的格式，包括：signUp logIn logOut时的用户名密码的通信格式；getFilelist时server返回的文件列表的通信格式；  增加client处一个线程的描述，该线程用来异步处理获取文件列表这个动作，提高用户体验； |
| V1.0.3 | WuJinlei | 20180224 | 增加遗留问题描述；  创建“测试”这个章节； |
|  |  |  |  |

# 目录

[moCloud设计文档 1](#_Toc25006)

[历史 1](#_Toc6437)

[目录 2](#_Toc20641)

[1.概述 4](#_Toc23669)

[2.应用场景 4](#_Toc32334)

[2.1.客户端 4](#_Toc29750)

[2.1.1.启动 4](#_Toc15826)

[2.1.2.注册/登录 5](#_Toc26096)

[2.1.3.初始化 6](#_Toc2546)

[2.1.4.获取文件列表 7](#_Toc1862)

[2.1.5.下载文件 7](#_Toc22510)

[2.1.6.在线播放文件 7](#_Toc15867)

[2.1.7.上传文件 7](#_Toc11765)

[2.1.8.删除文件 7](#_Toc14613)

[2.2.服务器 7](#_Toc579)

[2.2.1.初始化 8](#_Toc13308)

[2.2.2.文件管理相关场景 8](#_Toc20532)

[2.2.3.客户端管理相关场景 10](#_Toc7125)

[3.序列图 12](#_Toc31430)

[4.上下文图 12](#_Toc28773)

[5.模块图 12](#_Toc4199)

[6.内存和数据 12](#_Toc17716)

[6.1.客户端 12](#_Toc25092)

[6.1.1.线程 12](#_Toc11482)

[6.1.2.内存 13](#_Toc12209)

[6.2.服务器 13](#_Toc26518)

[7.类图 13](#_Toc32604)

[8.流程图 14](#_Toc14230)

[9.头文件 14](#_Toc5749)

[9.1.通信数据的格式 14](#_Toc4258)

[9.1.1.注册、登录、退出登录 14](#_Toc5484)

[9.1.2.获取文件信息列表 14](#_Toc17931)

[9.2.头文件定义 14](#_Toc27756)

[10.开发阶段 14](#_Toc12545)

[10.1.M1--创建模型、支持基本功能 14](#_Toc23241)

[10.2.M2--上传、下载、续传文件 15](#_Toc23240)

[10.3.M3--删除文件、在线播放文件 15](#_Toc30452)

[11. 遗留问题 15](#_Toc20834)

[11.1Client收到response，是否需要寻找mark 15](#_Toc27379)

[11.2Client发现读写server失败后的处理 16](#_Toc17056)

[12.测试 16](#_Toc12715)

[12.1.功能测试 16](#_Toc6338)

[12.2.性能测试 16](#_Toc17288)

[12.3.边界测试 16](#_Toc27913)

# 1.概述

moCloud定位为一个私有云系统，包含了稳定运行的云服务器，和可以正常使用的云客户端；

Server运行在树莓派上，client实现在手机上。两者可以通过局域网互联，是moCloud正常交互的前提，后续可以考虑将服务扩展至公网。

由于树莓派本身的硬件性能和存储size有限，因此设计方案是：

1. Server端存放一套数据文件；
2. 所有的客户端看到的都是这一套数据；
3. 客户端分两种权限：管理员和普通用户；普通客户端只能上传文件、下载文件、播放文件、获取文件列表等；管理员除此之外，还可以删除文件；

本文档，介绍moCloud的设计方式，实现流程，可以为后续参与该项目的人提供参考，也可以为后续维护提供依据。

# 2.应用场景

## 2.1.客户端

客户端的使用流程为：

* 启动客户端；
* 注册/登录客户端；
* 展示文件列表；
* 上传文件；下载文件；播放文件；删除文件；
* 退出登录；

### 2.1.1.启动

客户端程序运行时，首先应完成启动的动作，只有完成了启动这个步骤，我们才认为具备了与server进行交互的能力，才进行后续的注册/登录、文件操作等功能；如果启动失败，客户端只能退出登录，或调整后重新登录；

启动这个动作，主要负责：与server建立连接；与server进行密钥协商，得到本轮的密钥。过程描述如下：

1. 读取配置文件，获取客户端的ip地址：
   1. 读取配置失败，启动失败，提示用户后退出启动流程；
2. 使用随机分配的port，创建socket，连接server：
   1. 创建socket失败，启动失败，提示用户后退出启动流程；
   2. 连接server失败，启动失败，提示用户后退出启动流程；
3. 密钥协商，获取本次通信使用的密钥：
   1. 密钥协商失败，启动失败，提示用户后退出启动流程；
4. 启动心跳包流程；

**密钥协商**，负责从server端得到本次通信所使用的加解密算法，和密钥，其流程如下：

1. 构造密钥协商的request；
2. 使用本地存储的公钥，对request加密，得到request\_cipher;
3. 发送request\_cipher到server，并等待返回值：
   1. 存在一个超时时间，在该超时时间内如果没有能够得到返回值，属于协商失败的一种情况，需要提示用户后退出启动流程；
4. 得到返回值，使用公钥对返回值进行解密；
5. 将得到的密钥信息(加解密算法、密钥长度、密钥的值等)，保存在本地内存，供本次通信使用；

**心跳包流程**，指的是客户端需要间隔N秒，向服务器发送一个心跳包，告知服务器它还在进行连接，如果超过M秒没有发送，服务器会把这个客户端的连接信息删除，客户端需要重新连接才可以与server继续通信：

1. 生成一条心跳包请求request；
2. 使用密钥信息，加密得到cipherRequest；
3. 发送心跳包到server，并等待返回值；
   1. 同样有一个超时时间，如果超过了超时时间没有得到返回值，认为网络出现了异常，提示用户后，清空本轮的密钥信息等相关通信数据，尝试反复启动，直至启动成功；
4. 得到返回值cipherResponse后，执行解密；
5. 分析response，查看是否成功：
   1. 返回值可以反馈给客户端，文件列表是否发生了变化：
      1. 如果客户端已经成功登陆了，主动更新文件列表；
      2. 如果客户端没有登陆，忽略该返回值；
6. 间隔N秒后，再次执行一条心跳包请求；

### 2.1.2.注册/登录

注册和登录严格来说是两个动作，但由于体现在一起，而且注册后会有自动的登录流程，因此放在一起。这里只描述注册流程，登录流程可据此预见到：

1. 捕获注册所需要的用户名、密码信息，并据此生成一条request；
2. 使用本轮的密钥，对request加密，得到cipherRequest；
3. 发送cipherRequest，并等待返回值：
   1. 超时时间内得不到返回值，认为注册失败，提示用户后，退出注册流程，但不退出客户端，用户可以使用本次密钥协商的结果，继续注册；
4. 对返回值cipherResponse，使用密钥信息进行解密；
5. 分析解密后的response：
   1. 用户名已经被使用：必须更换一个用户名；
6. 注册成功后，自动将用户名、密码组合成一个request，用来登陆；
7. 对request加密后，发送到server，并等待返回值；
   1. 同样有超时时间，如果超过了超时时间没有返回，认为登录失败，提示登录错误，但是一定要告知注册已经成功了；
8. 获取返回值cipherResponse，解密，判断登录是否成功：
   1. 要考虑登录失败的情况，比如服务器现在接入的用户数多了，不再接入了，等；
9. 登录成功，直接进入初始化流程；

### 2.1.3.初始化

登录之后，自动执行初始化的动作，初始化主要包括两部分功能：获取文件列表并展示；继续上次未完成的上传、下载认为。

获取文件列表并展示，具体流程可以参见2.1.4详述的流程；

继续上次未完成的任务，例如上传、下载文件等，以上传文件为例，流程如下：

1. 查看是否存在unCompletedTask.log；
2. 解析文件，查看是否存在未完成的上传文件的任务；
3. 判断这些文件，是否依然存在于本地目录中，将其分类后，提示用户：有多少文件上次未完成，上次的时间是何时，现在有多少文件还存在、可以继续上传，上次上传了多少进度；
4. 请用户主动选择，是否继续上传，用户可以选择其中的一个或多个，可以选择上传了一部分的，或者完全没有上传过的；但是用户选择的，一定是当前本地存在的文件；
5. 如果选择了没有上传过的文件，直接执行2.1.7.的上传文件流程即可；
6. 如果选择了上传了一部分的文件，
   1. 首先向服务器发送请求，询问上次发送的文件是否还存在，是否还是该用户在该时间上传的：
   2. 等待返回值，支持超时时间，超时时间内没有返回，认为失败，请用户重新选择；
   3. 解密返回值，分析返回值：
      1. 如果依然是该用户上传的，并且size、offset等基本信息都正确，那么直接执行2.1.7的流程，继续上传文件；
      2. 如果不再是该用户上传的文件，或者文件在server端不存在了，或者offset等信息匹配失败，都需要告知用户该情况，并提示上传失败；

上传文件可以考虑有两个阶段开发，断点续传的功能，放到第二个阶段；

### 2.1.4.获取文件列表

文件列表展示，是moCloudClient的主界面；

获取文件列表，操作步骤如下：

1. 生成request；
2. 加密，得到cipherRequest；
3. 发送请求到server，并等待返回值：
   1. 超时时间内获取不到，提示用户，等待主动发起下一轮的获取；
4. 解密response，分析response，得到文件列表信息；
5. 将文件列表信息，展示在界面上；

### 2.1.5.下载文件

TODO

### 2.1.6.在线播放文件

TODO

### 2.1.7.上传文件

TODO

### 2.1.8.删除文件

TODO

## 2.2.服务器

服务器端的主要工作，可以概述为如下几个方面：

* 客户端的管理：包括接收客户端的请求，密钥协商，响应客户端的其他控制指令，管理客户端的心跳包机制，等等；
* 本地文件的管理：包括同步更新所有的本地文件，文件发生变化时广播到所有客户端，支持文件的上传、下载、播放，等；

### 2.2.1.初始化

服务器启动后，通过“初始化”的操作，开始服务：

1. 读取配置文件，得到“云文件目录”的路径；
   1. 配置文件读取失败，或不符合规则，返回错误，退出运行；
2. 执行“文件管理模块”的初始化，详见2.2.2中相关描述；
   1. 如果初始化失败，退出运行；
3. 执行“客户端管理模块”的初始化，详见2.2.3中相关描述；
   1. 如果初始化失败，退出运行；
4. 读取配置文件，创建socket，监听端口，等待客户端连接；

### 2.2.2.文件管理相关场景

#### 初始化

1. 传入的“云文件目录”是否存在：
   1. 目录不存在，创建；
   2. 路径有效，但路径并不是指向一个目录，而是一个文件或其他格式文件，返回错误，初始化失败；
2. 目录下的子目录是否符合规范：
   1. 约定的子目录不全，补全；
   2. 有其他的子目录，不处理；
   3. 子目录的路径指向的不是目录，而是文件或其他格式，返回错误，初始化失败；
3. 依次获取各个目录下的各个文件，判断文件是否在本地存储中已经存在：
   1. 不存在，新建一条记录，并更新到本地存储中；
   2. 存在，判断基本信息是否一致，不一致的，将本地文件的信息更新到本地存储中；
4. 再查看本地存储中是否存在这样的记录：在本地存储中有，但是本地文件中已经没有了的，
   1. 有这样的记录，将其在本地存储中删除；

#### 获取文件信息

支持：获取所有文件的信息；获取一种类型的所有文件信息；获取单个文件的信息；

以获取单个文件的信息为例：

1. 查看文件是否存在于本地存储中：
   1. 不存在的话，返回错误；
2. 读取文件信息，并组织成需要的格式，并返回；

#### 读文件--下载、播放、读异常处理

读文件涉及到的业务，包括了下载文件，播放文件；

下载文件和播放文件，由客户端管理模块的每个客户端对象，记录自己正在下载那个文件、下载的进度，下一次要下载的offset和size等；文件管理模块，只负责根据传入的offset和size，从文件中读取一部分内容，返回；

需要处理一种异常情况：假如server运行过程中，出现某一个文件被删除了，或者被人更换了文件体导致size变小了等，读取会失败，一旦读取失败了，我们要refresh本地存储的同时，也要广播给所有客户端，让他们重新获取文件列表，停止对这个文件的下载、播放等任务；这个过程称之为“读异常处理”；

1. 要读取的文件是否存在：
   1. 如果文件不存在，但是在本地存储中有该文件的信息，说明是在本次服务器启动后，运行期间，文件被删除了，1.清空本地存储中的本条文件记录；2.返回错误之后；3.触发“读异常处理”机制，告知所有client重新获取一次文件信息列表；
2. 状态是否是“deleting”：
   1. 如果状态是deleting，说明管理员马上要删除该文件了，不再支持读文件了，返回错误值；
3. 是否是第一次读文件：
   1. 是：打开文件，保存文件句柄，读的用户数量更新为1；
   2. 否：读的用户数量加1；
4. 根据传入的offset和size，读文件：
   1. 读文件失败，判断本地存储中文件信息和当前的文件信息，是否一致，如果不一致，说明服务器运行期间文件被修改过，需要：1.更新本地存储中的本条文件记录；2.返回错误；3.触发“读异常处理”机制，告知所有client重新获取一次文件列表；
5. 将读取到的内容，返回，一次读取完成；
6. 如果收到了“读结束”的标记：
   1. 如果已经是最后一个读用户了，直接关闭读句柄；
   2. 如果不是最后一个读用户，读的用户数量减1即可；

#### 写文件--上传文件

写文件针对的是上传文件的场景，上传文件，要上传不同名字的文件，而且上传过程中将文件后缀命名为“.uploading”，上传文成后，由客户端管理者将该文件，重命名为正确的名字；具体流程参见客户端管理模块针对上传文件的处理；

1. 文件在本地存储中，是否存在：
   1. 存在，说明文件已经被创建，判断写文件的用户是否和本次写文件的用户相同，
      1. 相同，读取文件句柄，执行步骤5；
      2. 不同，是另一个client试图上传同名文件，返回错误，告知client已经有用户上传该文件了，上传完成前不能重复上传；
   2. 不存在，说明是第一次写该文件，执行步骤2；
2. 文件在本地存储是否存在：
   1. 如果已经存在了，不能够再次写，必须更换其他名字后再写；因此返回失败；
3. 以写模式打开文件：
   1. 打开文件失败，返回失败；
4. 更新文件信息到本地存储；
5. 将传入数据，按照传入的offset、size，写入文件：
   1. 如果出现了跳block的情况，中间的block先写入全0值占位；

如果收到写完成的指令，将文件重命名，取消“.uploading”后缀，并更新到本地存储中；

#### 删除文件

删除文件的操作，只有管理员权限才可以做，用户不可以做；

1. 查看发起者的权限：
   1. 普通用户权限，不允许删除文件，返回错误值；
2. 查看该文件在本地存储中是否存在；
   1. 不存在，返回错误；
3. 更新该文件的状态到deleting；
4. 删除文件；
5. 更新本地存储；
6. 触发“读文件异常”机制，告知所有client更新文件列表；

### 2.2.3.客户端管理相关场景

客户端管理模块负责管理：

* 接收新的客户端连接请求；
* 管理所有的客户端连接，例如心跳机制；
* 为每个客户端提供单独的对象，处理其业务请求；
* 其他相关业务；

#### 初始化

初始化操作，准备相应的数据结构，分配需要的内存，启动对应的线程等；

1. 清空存放所有客户端信息的map；
2. 启动“客户端状态管理”线程，该线程负责定时检查客户端的状态，如果状态更新为了invalid，说明心跳超时或者客户端主动发起了byebye的动作，需要清除该客户端的信息；
3. 启动接收连接的主线程，该线程负责accept新的客户端请求；

#### 收到新的连接

1. 判断该IP是否已经存在于map中：
   1. 存在，步骤2；
   2. 不存在，说明是新的client的控制端口发来的连接请求，步骤3；
2. 查看该IP在map中，控制端口、数据端口的状态：
   1. 控制端口有效，数据端口也有效：该IP不能再发起connect请求，一个client只能启动一个客户端，返回错误；
   2. 控制端口有效，数据端口无效：本请求是该IP的数据端口发起的，将本次的请求加入map中，返回正确值；
   3. 与控制端口重复：不能重复使用同一个ip和端口发起connect命令，返回错误；
3. 将该请求加入到map中；
4. 创建一个clientMgr对象，该对象负责接收该client的所有控制指令，并处理之；

#### 处理客户端的业务请求

TODO

#### 主动清除客户端

清除客户端的动作，触发源分为两种：客户端异常了导致心跳包没有定时发送，服务器要清除它的信息；客户端不再登录server了，发送了byebye指令，server要清除它的信息；

1. 遍历map，找到状态为invalid的clientMgr对象；
2. 删除该对象；

#### 广播消息

“读异常机制”需要能够广播消息给所有的client，告知他们重新获取文件列表；这里的广播就是这个意思。

1. 遍历map中各个客户端；
2. 状态为running的，说明已经做过密钥协商、登录了的，且有效的client，执行步骤3；否则，跳过该client，不予发送信息；
3. 组织消息体，发送到client；

# 3.序列图

# 4.上下文图

# 5.模块图

# 6.内存和数据

线程，内存

## 6.1.客户端

### 6.1.1.线程

客户端需要支持：心跳包发送线程、文件信息列表刷新线程、文件数据接收线程；

心跳包发送线程，概述如下：

1. 密钥协商后启动该线程，程序退出时停止；
2. 通过返回值判断服务器是否运行正常：
   1. 能正常获取返回值：服务器正常运行；
   2. 获取不到返回值：与服务器通信中断，停止当前所有任务，等待连接；
   3. 返回值有其他意义：比如服务器上的文件列表发生变化，通过该返回值体现，告知client重新获取一次服务器文件列表，等等；

文件信息列表刷新线程，主要目的是将getFilelist的操作异步实现，原理阐述如下：

1. Client发送心跳包给server后，server的返回值告知client，文件列表(可以知道是那个类型的文件列表)变化了；
2. 心跳包线程，发送信号量给文件信息列表刷新线程；
3. 文件信息列表刷新线程捕获到信号量后，查看client是否已经登录：
   1. 如果已经登录，继续step4；
   2. 如果未登录，不做处理，直接返回；
4. 发送getFilelist的request给server，获取给定类型的文件列表信息；
5. 将这次获取到的列表信息，设置到一个全局变量中；

至此，该线程的工作就完成了。

一旦用户需要查询文件列表的时候，并不是直接向服务器要数据，而是直接拿这个文件信息列表缓冲区中的值，这样可以减少网络通信造成的延时，提高用户体验。

文件接收，需要2个线程，一个负责接收来自于server的回复数据，称为接收线程；一个用来将数据写入到指定文件中，称为写入线程；

主要用来在下载文件时使用：

1. 程序启动后，启动写入线程，由于该线程以信号量作为触发机制，因此不占用系统资源；有确切的下载请求时，再启动接收线程；
2. 程序退出时停止这两个线程；
3. 接收线程接收到数据后，将数据暂存到本地内存；写入线程被唤醒后，读取本地内存的数据后，写入特定文件；

### 6.1.2.内存

上传文件时，一次限定最多传送多少个字节的block，只需要初始化的时候，分配特定size的内存，存放这个block的size即可，由于数据部分不加密，因此这个内存的size也更好确定；

下载文件时，为了减少写入操作的缓慢导致的可能性阻塞，

读取文件列表时，由于文件的个数无法预估，因此采取动态内存的方式，将得到的所有文件信息，以链表的形式存储；在使用完成后，将链表空间释放；

## 6.2.服务器

# 7.类图

# 8.流程图

# 9.头文件

## 9.1.通信数据的格式

客户端和服务器通信时，可以通过body来进行额外数据的发送，本章节介绍各个场合下的通信数据的组成格式；

### 9.1.1.注册、登录、退出登录

注册、登录、退出登录，都是首先发送一个标准的request header(密文形式)，之后尾随一个如下格式的body(明文格式)：username=%s, password=%s；

解析规则是：第一个username和最后一个“,password=”之间的所有内容，都是用户的username；最后一个“,password=”之后的所有内容，都是用户的password；

### 9.1.2.获取文件信息列表

获取文件信息列表，是由client主动发起，server返回要查询的类型的所有文件的基本信息的操作；

Server的返回值，除response header(加过密的密文)外，还会附带一个response body(明文)；该response body的组成，就是MOCLOUD\_BASIC\_FILEINFO这个二进制数据的依次排列；

Client解析时，按照顺序，通过sizeof(MOCLOUD\_BASIC\_FILEINFO)可以获知每个文件的信息；

## 9.2.头文件定义

# 10.开发阶段

## 10.1.M1--创建模型、支持基本功能

第一阶段，主要创建客户端和服务器模型，保证正常的通信机制，和基本的业务模型，具体来说，主要是如下几个方面：

* 加解密模块：支持正常的加解密；
* 信息校验模块：支持需要的校验算法；
* 通信模块：client和server可以正常通信；
* 心跳包管理模块：客户端可以正常发送心跳包，服务器可以管理心跳包，两者都可以根据心跳包的正常与否，做出正确的反应；
* 基本功能之密钥协商：可以正常进行密钥协商；
* 基本功能之获取文件列表：client可以正确获取到server当前的所有文件的列表信息，并展示给用户；

1. MoCloudUtilsCrypt，moCloudUtilsCheck，打桩； OK
2. moCloudUtilsType，moCloudUtils，实现；OK
3. moCloudClient：
   1. Init: connectToServer; doKeyAgree; OK
   2. startHeartBeat : sayHiToServer; OK
   3. signUp : 注册; OK
   4. signIn : 登录；OK
   5. getFileList; OK
   6. unInit: sayByeByeToServer; freeAllResources;OK
4. moCloudServer：启动，接收client的connect请求，每一个client新建一个thread处理；
5. moCloudServer：文件管理模块；
6. moCloudServer：client管理模块，心跳包管理最重要；
7. moCloudServer：支持基本功能之密钥协商和获取文件列表；
8. 正式实现moCloudUtilsCrypt；
9. 正式实现moCloudUtilsCheck；

## 10.2.M2--上传、下载、续传文件

## 10.3.M3--删除文件、在线播放文件

# 遗留问题

## 11.1Client收到response，是否需要寻找mark

Client收到response后，是否有必要，寻找第一个可用的mark，才认为找到了response header；

现在的处理逻辑，是发送了request之后，直接等待返回值，返回值到了，直接按照response header的格式进行解析，这样是基于对tcp的信赖，我认为已经足够了，但后续如果测试发现有时候有些数据的response并不是这样满足条件的，就需要更新这部分代码：

Recv到server发来的数据后，解密，之后查找mark，找到了才认为找到了一个response header；

## 11.2Client发现读写server失败后的处理

Client发现server读写出现问题之后，应该怎么处理呢？

最好的体现，肯定是马上告知用户，然后轮询，一旦server重新上线，自动进行init操作，用户重新logIn，就可以又正常使用了。

这部分业务需要好好想想再实现。

# 12.测试

## 12.1.功能测试

## 12.2.性能测试

## 12.3.边界测试