Agenda 扩展文档

我在基础的 Agenda 上面添加了两个类：log 和 Reload，一共实现了三个功能：Agenda Service 的日志功能；agenda.data 的备份；还有数据的恢复功能。现在大概已经实现了，我想用我认为比较好的方式去说明。

一、Class log：

```
#ifndef LOG_H
#define LOG_H

#include"Date.h"
#include<map>
#include<string>
#include <time.h>
using std::map;
using std::string;

class Log {

    private:
        map<string, string> log;

    public:
        Log();
        ~Log();
        void readFromFile(const char* fpath);
        void write(string str);
        void writeToFile(const char* fpath);
        time_t nowtime;
};
```

我将 log 的文本放在一个跟 Agenda 同目录的“Log.txt”中：

1.每次 Agenda 启动时，Log 会在构建函数中调用 readFromFile()读取储存在文本中的数据，记录在 map 当中；

2.在 AgendaService 中涉及到 添加/删除 操作（如 createUser，createMeeting），登录/登出 操作（UserLogin， UserLogout）还有 启动/退出操作（StartAgenda，QuitAgenda）的函数，都会生成相应的字符串，通过 write()记录到 map 当中；

3.Map 的 first(key)是一个格式为“2014-07-05/01:14:23”的表示时间的字符串，利用 time.h 来实现并获取系统当前时间，用于将该操作与它执行的时间绑定在一起；

4.每当 Agenda 退出后，log 在析构函数里面执行 writeToFile()将内存中的数据储存到“Log.txt”当中，格式如图：

Log.txt ×

```
2014-07-04/12:41:21
Start Agenda
--------------------------------
2014-07-04/12:41:35
administrator log in
--------------------------------
2014-07-04/12:41:37
Quit Agenda
--------------------------------
```

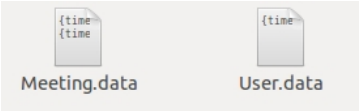
这样就实现了Agenda Service的日志功能。

二、Class Reload：

```
#ifndef Reload_H
#define Reload_H

#include"Storage.h"
#include<list>
#include<map>
#include<string>
#include <time.h>
using std::list;
using std::string;
using std::map;

class Reload {
    public:
        static Reload *getInstance();
        ~Reload();
        bool reloadUser(string date, string username, Storage* storage_);
        bool reloadMeeting(string date, string username, Storage* storage_);
        void write(User U);
        void write(Meeting M);
        time_t nowtime;

    private:
        bool readFromFile(const char *Ufpath, const char *Mfpath);
        bool writeToFile(const char *Ufpath, const char *Mfpath);
        map<string, User> userM;
        map<string, Meeting> meetingM;

        static Reload* instance_;
        Reload();
};
#endif
```

我认为Reload的实现机制跟原来Agenda里面的Storage类大致相同，但是功能有所拓展，主要实现流程是：

1.Reload也是单例模式，在构建函数中readFromFile()，在getInstance()中实例化，在析构函数中writeToFile()，Reload的数据储存在“sync/User.data”和“sync/Meeting.data”中；



（本来我是打算在Reload类中实现Agenda的数据备份功能的，但是后来发现，把“添加”和“删除”的数据不加区分地放在一起，进行数据还原的时候要分辨要很费功夫。比如说：有人创建了一个名为“123”的User，然后误删了“123”，恢复的时候就会有两个“123”。当然，现在这样只存储delete掉的数据也还是会有重复的问题...）

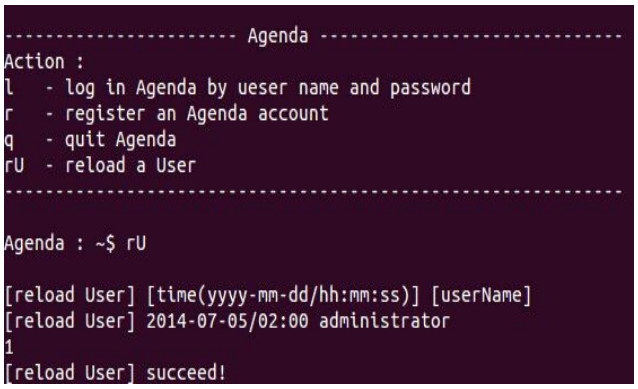
2.我在 AgendaService 中添加了“Reload*”的成员，使得 AgendaService 能够使用 Reload 的接口。当 Agenda 启动后，每当 AgendaService 执行删除 操作（deleteUser，deleteMeeting）时，都会通过 write(User)和 write(Meeting)将该元素记录到两个 map<string, string>当中，同样的关键字是形如“2014-07-05/01:14:23”的记录操作时间的字符串；

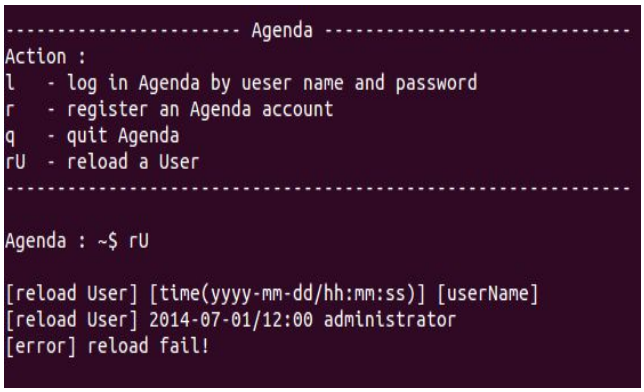
3.当用户需要恢复数据时，通过更改后的 UI，控制 AgendaService 调用 Reload 的接口 reloadUser 和 reloadMeeting 进行数据恢复：

该操作需要提供恢复的起始时间点（只会恢复该时间点之后执行的操作），用户名还有 AgendaService 中的 Storage*成员，该部分的 UI 界面如下：

（1）用户只能在未登录状态恢复一个用户；

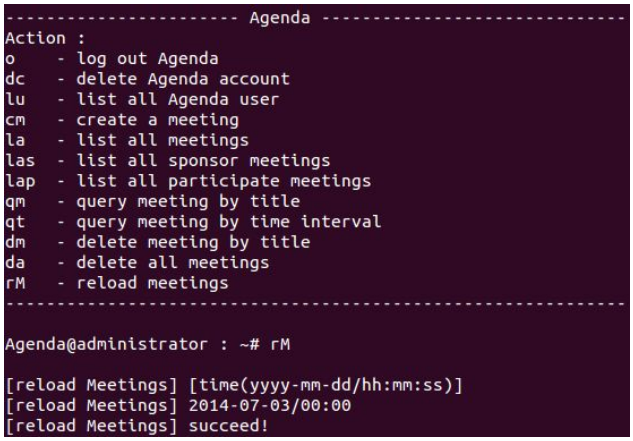
（2）当 User.data 里面没有符合条件的用户，或者该用户名已经被另外一个用户注册使用，无法恢复；

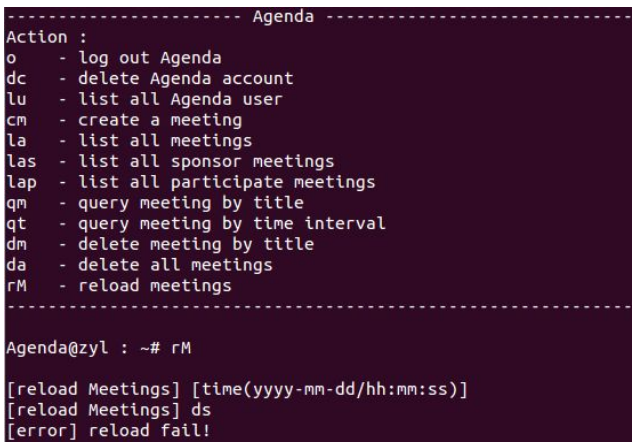




（3）恢复 Meeting 的操作要在用户的登录状态之下才可以执行，恢复改时间点之后的属于该用户的 Meeting；

（4）若该用户不存在符合条件的会议，也会有“reload fail！”的提示，会议的 Title 与现有的会议的 Title 有冲突的，还有就是要恢复的会议与已经存在的回忆存在时间冲突的也无法恢复；





（5）本来想要实现在 ReadFromFile 当中剔除掉距离系统当前时间 24 小时之前的数据，即只能恢复当前时间 24 小时之前的数据，但是想到要判断时间差

好像等于再写一个日期加减的功能...所以最后还是没加上去了。

三、agenda.data 的备份

我实现备份的方法很简单，只是修改了 Storage 类的构建和析构函数：

```
18 Storage::Storage() {
19     if (readFromFile("agenda.data")) {
20     } else {
21         readFromFile("sync/back-up.data");
22     }
23 }
24 Storage::~Storage() {
25     writeToFile("agenda.data");
26     writeToFile("sync/back-up.data");
27     instance_ = NULL;
28 }
```

当 Aganda 启动的时候，如果 Storage 无法打开“agenda.data”那么就可以从“src/back-up.data”那里读取数据；

而每次 quitAgenda，要将内存的数据写入“agenda.data”的时候，也同时要写一分数据放进“sync/back-up.data”里面。

当我写完 Reload 类之后，我感觉自己已经将原来的三层结构改得面目全非了。在想着实现数据恢复这个功能的时候，其实最大的难度还是在于如何架构，和组织新添加的类与原来的 Storage，AgendaService 之间的关系。所以刚开始写的时候思路比较混乱，也一直的混乱到了结束。这份文档大概的将我的思路，还有新添加拓展功能的用法描述了出来，对于每个功能，我都直接在 UI 里面用一两组数据测试了一下，应该没什么问题了吧...