Agenda 大作业实验报告

18329015

教务四班

郝裕玮

一、重要声明

老师好！在这里我想提前补充说明一件事，因为考虑到这件事如果不提前说清楚的话，可能会导致我的代码被判为抄袭，所以我把对此事的说明放在了第一部分。

事情经过是这样的：因为我在写 AgendaService.cpp 时，一开始交了很多次都没有满分通过 Matrix 的机器测试，然后当时的我有点急躁，所以我选择上网查找了一些参考代码进行复制粘贴，想先看看网上的能不能通过机器测试，然后再根据网上的参考代码来对我自己写的文件进行对比找出错误。（因为我自己 debug 一直没有找到问题所在）然后不出所料，网上的代码满分通过了机器测试，之后我将我自己的代码和网上的代码进行了对比修改后（个人认为我修改后的代码是没有大面积抄袭网上的代码的。即使出现了可能重复度较高的代码部分，我也均附上了相关的注释，表明我的确理解了这一部分代码的原理），也顺利通过。但后来我考虑到老师和助教对我的代码进行查重时可能会只查 Matrix 上第一次满分通过的代码，而忽略之后满分通过的代码，所以我在咨询了李佳学长，陈德和学长和林津霞学姐这三位助教之后，得到了统一的答复：我在我的实验报告中详细说明这一情况即可，如果到时候查重出现问题，他们会主动记录我在实验报告中的反馈。

所以我最后发到邮箱的代码文件会和我第一次满分通过 AgendaService 类机器测试（2020-08-21/12：56，commit 注释为 user6）的代码有所不同，如果最后查重真的有问题，麻烦老师和助教以我发的代码文件为准。（或者以 Matrix 上第二次满分通过 AgendaService 类机器测试的代码（2020-08-24/20：05，commit 注释为 Final 15）为准也可以）

下面是我当时参考代码的网站：

https://blog.csdn.net/dcy19991116/article/details/84633885

二、对于三层架构的讲解

虽然三层架构在我们的代码中不会直接体现，但我觉得它解释了为什么我们需要补充老师所给的这些头文件，因为这些头文件已经帮我们实现了三层架构，我们所需要做的就只是补充每个头文件里的函数。这实际上大大减轻了我们的负担，即我们无需构思是否还要补充别的头文件，因为老师已经帮我们确定了此次代码工作的大方向。

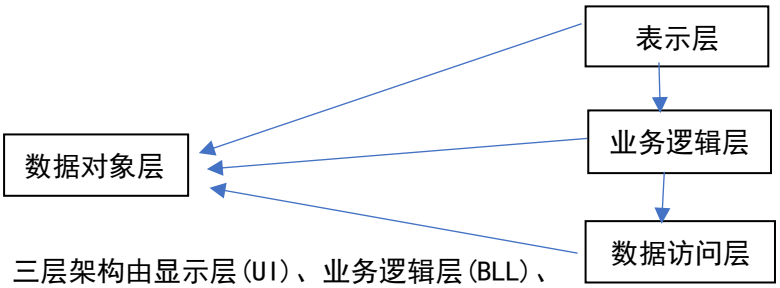
接下来我想对三层架构进行一些分析，也是为了后续报告中保证我能够思路清晰地讲解各部分代码的用处。

我们先分析一下两层架构。两层架构将界面展示、业务逻辑、数据访问等都写到一起，所以如果用户需求变化，我们就需要对整个项目进行大量修改，这对于系统的维护和升级极其不利；而且通过界面层直接访问数据库，还会有安全隐患。两层架构如下图所示：

而三层架构符合“高内聚、低耦合”的特点，每个层职责明确。利用分层，降低了层间依赖，使系统的耦合更加松散，从而使系统更加容易维护和复用。

比如：如果需求有变化，只需要更改相应的业务逻辑层；或者要改变数据库的时候，只需要将原来的数据访问层替换掉或者增加新的就可以了，而不需要牵扯到整个项目。

所以三层架构的逻辑图如下图所示：



三层架构由显示层(UI)、业务逻辑层(BLL)、数据访问层(DAL)组成：

1. 显示层（UI）

职责：①向用户展示特定的业务数据

②采集用户的信息和操作

原则：用户至上，兼顾简洁

2. 业务逻辑层（BLL）

职责：① 从 UI 中获取用户指令和数据，执行业务逻辑

②从 UI 中获取用户指令和数据，通过 DAL 写入数据源

③从 DAL 中获取数据，以供 UI 显示用

机制：① UI－>BLL－>UI

② UI－>BLL－>DAL－>BLL－>UI

3. 数据访问层（DAL）

职责：①执行对数据的操作（增删改查）

作用：跟数据源打交道

4. 数据对象层

数据对象层包含了项目需要使用的数据对象，用数据对象来传递数据，它避免了各个层的交叉引用。

三、对于 Date、User、Meeting、Storage 四个类的讲解

1、对于 Date 类

```
int daysOfMonth(int year,int month)
{
    int mon[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
    int ans=0;
    if(month!=2){
        ans=mon[month];
    }
    else{
        if(year%4==0&&year%100!=0||year%400==0){
            ans=29;
        }
        else{
            ans=28;
        }
    }
    return ans;
}
```

上图函数是为了提高代码的复用性，即在 Date 类中如果需要某年某月的天数，直接调用该函数即可（该函数也已经将闰年对 2 月天数的影响纳入到判断依据当中）。

```
std::string Date::dateToString(const Date &t_date)
{
    string temp;
    if(isValid(t_date)==false){
        temp="0000-00-00/00:00";
    }
    else{
        char year[5];
        sprintf(year,"%4d-",t_date.m_year);
        temp.insert(0,year);


        char month[3];
        sprintf(month,"%02d-",t_date.m_month);
        temp.insert(5,month);

        char day[3];
        sprintf(day,"%02d/",t_date.m_day);
        temp.insert(8,day);
        char hour[3];
        sprintf(hour,"%02d:",t_date.m_hour);
        temp.insert(11,hour);

        char minute[2];
        sprintf(minute,"%02d",t_date.m_minute);
        temp.insert(14,minute);
    }
    return temp;
}
```

上图函数是将日期转化为字符串，其中重点在 sprintf 函数和 insert。sprintf 函数可以将字符串（括号中的第 3 个参数）以指定的格式（括号中的第 2 个参数）写入到另一个字符串或字符串数组中当中（括号中的第 1 个参数）。每 sprintf 一次，我们就根据此次写入的字符串长度来判定在 temp 中开始插入的位置，并使用 insert 函数（从第 1 个参数表示的位置开始插入第 2 个参数中的字符串或字符串数组）来逐步完善最终返回的结果。

```
/**
* @brief check whether the CurrentDate is  greater than the t_date
*/
bool Date::operator>(const Date &t_date) const
{
    bool flag=false;
    if((*this).dateToString(*this)>t_date.dateToString(t_date))
    {
        flag=true;
    }
    return flag;
}
```

以重载运算符 > 为例，因为 string 库中已经对 == ，> ，< 进行了重载。所以我们只需在头文件部分#include<string>，即可大大减少代码复杂度。如上图所示直接将需要比较的两个日期都调用 dateToString 函数转化为字符串，然后直接使用运算符进行比较即可。所以 == ， >= ，< ，<= 的代码内容均与上图类似，不再一一赘述。

2、对于 User 类

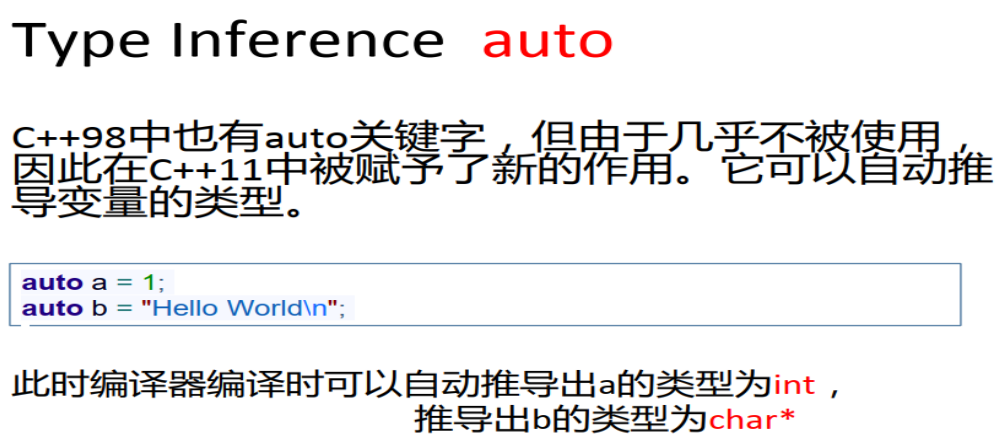
函数重复性很高且内容较为简单，没有需要特别讲解的地方。

3、对于 Meeting 类

```
void Meeting::removeParticipator(const std::string &t_participator)
{
    for(auto i=m_participators.begin();i!=m_participators.end();i++)
    {
        if((*i)==t_participator){
            m_participators.erase(i);
            break;
        }
    }
}
```

上图函数是移除某个会议成员，其中重点是关键字 auto，auto 可以自动推导出变量的类型，这为我们省去很多不必要的麻烦。在循环内部一旦(*i)==t_participator，就立刻调用 erase 函数来删除迭代器 m_participator 中指定的元素 i，此时 i 指向 m_participator 中的 t_participator。

对于关键字 auto，在 Storage 类，AgendaService 类中会经常出现，所以在后面的讲解中不会再重点强调关键字 auto。下图为万海老师的 PPT：Lecture Notes on C++ - 12 C++1x 中对于关键字 auto 的说明：



4、对于 Storage 类

```
string temp1;
while(getline(users,temp1)){
    string name,password,email,phone;
    int mark_position[8];
    int j=0;
    for(i=0;i<temp1.size();i++){
        if(temp1[i]=='"'){
            mark_position[j]=i;
            j++;
        }
    }
    name=temp1.substr(mark_position[0]+1,mark_position[1]-mark_position[0]-1);
    password=temp1.substr(mark_position[2]+1,mark_position[3]-mark_position[2]-1);
    email=temp1.substr(mark_position[4]+1,mark_position[5]-mark_position[4]-1);
    phone=temp1.substr(mark_position[6]+1,mark_position[7]-mark_position[6]-1);
    User user(name,password,email,phone);
    m_userList.push_back(user);
}
users.close();
```

上图为 bool readFromFile(void)函数内的一部分代码，该部分代码是读取用户信息。首先根据用户各项数据的格式："maomao（姓名）","123（密码）","980@qq.com（邮箱）","188194（电话）"来先对其中的双引号进行特殊处理。我们新建一个数组 mark_position 用于存储双引号的位置。由数据的存储格式可知，每两个引号之间就会存有用户的某项信息（每两个指的是 0 和 1，2 和 3……这种，1 和 2，3 和 4 这种中间只有一个逗号，不存在用户信息）。之后我们调用 substr 函数将各项信息依次赋给我们定义的内部变量 name，password，email，phone。（定义在内部可以使得当进行下一次 while 循环时，这些变量会自动清空内容）substr 函数的其中一种用法为：括号内的第 1 个参数为开始提取的位置，括号内的第 2 个参数为从第 1 个参数所对应的位置开始，所需要提取的字符串长度。所以我们将 temp1 和双引号的位置（mark_position）相结合并调用 substr 函数即可得到用户的各项信息。之后创建新用户 user，并使用 push_back 函数将 user 添加到 m_userList 当中即可。

```
    string temp2;
    while(getline(meetings,temp2)){
        string sponsor,participators,startDate,endDate,title;
        int pos=1,len=0;
        pos=temp2.find("\"")+1;
        len=temp2.find("\",\"",pos)-pos;
        sponsor=temp2.substr(pos,len);

        pos+=len+3;
        len=temp2.find("\",\"",pos)-pos;
        participators=temp2.substr(pos,len);
        participators+='&';

        pos+=len+3;
        len=temp2.find("\",\"",pos)-pos;
        startDate=temp2.substr(pos,len);

        pos+=len+3;
        len=temp2.find("\",\"",pos)-pos;
        endDate=temp2.substr(pos,len);

        pos+=len+3;
        len=temp2.find("\"",pos)-pos;
        title=temp2.substr(pos,len);
```

上图仍然为 bool readFromFile(void)函数内的一部分代码，该部分代码为读取会议信

息。首先会议各项数据的格式为："maomao（会议主持人）","a&b（参与者）","2017-05-21/16:06（开始时间）","2017-05-22/16:06（结束时间）","maomao（会议标题）"。我在写此部分代码的过程中，发现如果我仍然套用读取用户信息的那部分代码的话，最终结合 UI 运行时就会出现问题，即我每次编译运行后（单次编译运行不会出问题）进行列举会议的操作时（如 la），我的参与者均会减少一人，直至参与者为 0，会议消失。我 debug 了很久也没有找出原因。所以最终决定采用如上图所示的全新的读取会议信息的方法（大体思路仍保持不变，只是换了一些函数）：每次使用 find 函数找出 "," 的所在位置，然后根据符号位置计算出每项会议信息的长度，最后再次使用 substr 函数给各个变量赋值即可。

```
            vector<string> No_participators;
            string No_i;
            for(i=0;i<participators.size();i++){
                if(participators[i]!='&'){
                    No_i.push_back(participators[i]);
                }
                else if(i<participators.size()-1){
                    string temp=No_i;
                    No_participators.push_back(temp);
                    No_i.clear();
                }
                else if(i==participators.size()-1){
                    No_participators.push_back(No_i);
                }
            }
            Date start(startDate);
            Date end(endDate);
            Meeting meeting(sponsor,No_participators,start,end,title);
            m_meetingList.push_back(meeting);
        }
        meetings.close();
    }
    return flag;
```

上图为 bool readFromFile(void)函数内的最后一部分代码，这一部分主要是将 participators 的内容写入到 vector<string>当中，以便最后创建会议变量 meeting 时可以直接使用，避免麻烦。首先我们 for 循环里遍历 participators 字符串数组，在没遇到&前都将当前的字符 participators[i]进行 push_back，复制到 No_i 字符串中（第 i 个参与者）。遇到&后，我们就将当前 No_i 的内容复制给临时变量 temp，再将 temp 进行 push_back，复制到 No_participators 中，并将 No_i 的内容清空（使用 clear( )函数），用于下个参与者的信息复制当中。

最后我们创建 Meeting 类变量 meeting 并进行初始化，并将其 push_back 到 m_meetingList 当中，就完成了新会议的读取。

```
int Storage::deleteUser(std::function<bool(const User &)> filter) {
    int cnt=0;
    m_dirty=true;
    for(auto i=m_userList.begin();i!=m_userList.end();i++){
        if(filter(*i)==true){
            cnt++;
        }
    }
    m_userList.erase(std::remove_if(m_userList.begin(),m_userList.end(),filter),m_userList.end());
    return cnt;
}
```

上图为删除用户的函数，其中的重点为 remove_if 函数，因为如果我们一边遍历容器一边 erase(即在 for 循环内部多次 erase)的话，会导致迭代器失效。所以我们使用 remove_if 函数，它的用法为：括号内前 2 个参数为过滤范围，第 3 个参数为过滤条件，即我们要使用何种清除规则来清除容器中的元素。这里很显然我们就是需要清除使 filter 返回值为 true

的那些元素。对于 remove_if 的使用，我们在 deleteMeeting 函数中也有使用，其使用方法和这里类似，所以不再赘述。

四、对于 AgendaService、AgendaUI 两个类的讲解

1、对于 AgendaService 类

```
bool AgendaService::deleteUser(const std::string &userName, const std::string &password){
    auto filter=[userName, password](const User &user)->bool{
        if(userName==user.getName()&&password==user.getPassword()){
            return true;
        }
        else{
            return false;
        }
    };
    deleteAllMeetings(userName);
    auto meetings_list=listAllParticipateMeetings(userName);
    for(auto i : meetings_list){
        quitMeeting(userName,i.getTitle());
    }
    return m_storage->deleteUser(filter)>0;
}
```

以上图函数为例，AgendaService 函数中最重要的一个部分就是 Lambda 函数的运用。Lambda 函数的范式为：[捕捉列表](参数)mutable—>返回值类型{函数体}。所以如上图所示，我们将捕捉列表设为 userName 和 password，参数为 const User& user,返回值类型设为 bool。并将返回值赋值给 filter（使用 auto 关键字）。最终我们通过调用 Storage 类的某个函数并将 filter 作为该函数参数的方式，成功实现了将 AgendaService 类和 Storage 类联系在一起的目的。（即将 AgendaService 类的函数指令发送到 Storage 类的对应函数，并最后实现对.csv 文件的数据修改）

因为 Lambda 函数几乎在每个函数中都有出现，所以在接下来的讲解中将不再单独解释 Lambda 函数。

因为除了 Lambda 函数为大多数函数的重点外，其他的重点基本上集中在各个函数内部逻辑思维的具体实现,所以接下来我将对我觉得重要的函数的代码实现部分直接截图并在图中附上注释（中文），且不再单独采取文字叙述（因为我认为在代码处直接附上备注会比额外的文字解释更加直观，但为了防止老师打开文件时我的中文注释变成乱码，我会将发给老师的文件中的所有注释均改成英文，如果给老师造成不便还请谅解）（为了使注释更鲜明我把整体背景换成了白色）

（1）用户注册

```
bool AgendaService::userRegister(const std::string &userName, const std::string &password,const std::string &email, const std::string &phone){
    std::list<User> temp=listAllUsers();
    for(auto i=temp.begin();i!=temp.end();i++){//遍历用户列表
        if(i->getName()==userName){//判断该用户名是否已注册
            return false;
        }
    }
    User user(userName,password,email,phone);//若用户名尚未注册则将其初始化为User类变量user
    m_storage->createUser(user);//调用create函数创建新用户
    return true;
}
```

（2）删除用户

```
bool AgendaService::deleteUser(const std::string &userName, const std::string &password){
    auto filter=[userName, password](const User &user)->bool{
        if(userName==user.getName()&&password==user.getPassword()){//判断当前登录用户与待删除用户是否一致
            return true;
        }
        else{
            return false;
        }
    };
    deleteAllMeetings(userName);//删除该用户所有主持的会议
    auto meetings_list=listAllParticipateMeetings(userName);
    for(auto i : meetings_list){//遍历会议列表
        quitMeeting(userName,i.getTitle());//若该用户所有参加的会议
    }
    return m_storage->deleteUser(filter)>0;
}
```

（3）创建会议

```
bool AgendaService::createMeeting(const std::string &userName, const std::string &title,const std::string &startDate,
                                  const std::string &endDate,const std::vector<std::string> &participator){
    Date start(startDate);
    Date end(endDate);
    Meeting meeting(userName,participator,start,end,title);
    if(start>=end||start.isValid(start)==false||end.isValid(end)==false){//日期不合法则返回false
        return false;
    }
    bool flag=false;
    for(auto i : listAllUsers()){//遍历所有用户
        if(i.getName()==userName){//判断主持人姓名是否已注册
            flag=true;
            break;
        }
    }
    if(flag==false){//若主持人姓名不存在则返回false
        return flag;
    }
    if(participator.size()==0){//判断该会议的参与人员数量是否为0
        return false;
    }
```

```
    auto filter=[title](const Meeting &meeting)->bool{
        bool flag1;
        if(meeting.getTitle()==title){//判断会议标题是否已经存在
            flag1=true;
        }
        else{
            flag1=false;
        }
        return flag1;
    };
    if((m_storage->queryMeeting(filter)).size()>0){
        return false;
    }
    auto filter1=[userName](const Meeting &meeting)->bool{
        bool flag2;
        if(userName==meeting.getSponsor()){
            flag2=true;
        }
        else{
            flag2=false;
        }
        return flag2;
    };
```

```
    std::list<Meeting> meeting1=listAllMeetings(userName);
    for(auto i : meeting1){//遍历该用户的所有会议（participator and sponsor）
        if(!(i.getEndDate()<=start||i.getStartDate()>=end)){//判断会议时间与其他会议时间是否存在重叠
            return false;
        }
    }
    std::vector<std::string> participators;
    Meeting temp(userName,participators,start,end,title);
    m_storage->createMeeting(temp);
    for(auto i=participator.begin();i!=participator.end();i++){//遍历参会人员
        bool flag3;
        flag3=addMeetingParticipator(userName,title,*i);
        if(flag3==true){
            participators.push_back(*i);//将新成员加入该会议成员列表
        }
        else{
            m_storage->deleteMeeting(filter);//将该用户从该会议中删除
            return false;
        }
    }
    return true;
}
```

（4）增加会议成员

```
bool AgendaService::addMeetingParticipator(const std::string &userName,
                                           const std::string &title,
                                           const std::string &participator){
    bool flag=false;
    for(auto i : listAllUsers()){//遍历用户列表
        if(i.getName()==participator){//判断参会人员是否已注册
            flag=true;
            break;
        }
    }
    if(flag==false){
        return false;
    }
    auto meeting=m_storage->queryMeeting([&](const Meeting &meeting1)->bool{
        if(meeting1.getSponsor()==userName&&meeting1.getTitle()==title){//判断待加入会议是否存在
            return true;
        }
        else{
            return false;
        }
    });
    if(meeting.size()==0){
        return false;
    }
```

```
    auto it=meeting.begin();
    auto temp=*it;
    auto participator_meeting=listAllMeetings(participator);
    for(auto i : participator_meeting){//遍历participator所有主持或参与的会议
        if(!(temp.getEndDate()<=i.getStartDate()||temp.getStartDate()>=i.getEndDate())){//判断participator能否参加此次会议，避免会议时间冲突
            return false;
        }
    }
    auto filter=[userName,title](const Meeting& meeting)->bool{
        if(meeting.getSponsor()==userName&&meeting.getTitle()==title){//判断待加入会议是否存在
            return true;
        }
        else{
            return false;
        }
    };
    auto switcher=[participator](Meeting& meeting)->bool{
        meeting.addParticipator(participator);//将participator加入此次会议
    };
    m_storage->updateMeeting(filter,switcher);//更新会议状态
    return true;
}
```

（5）删除会议成员

```
bool AgendaService::removeMeetingParticipator(const std::string &userName,
                                              const std::string &title,
                                              const std::string &participator){
    auto filter=[userName,title,participator](const Meeting& meeting)->bool{
        if(meeting.getSponsor()!=userName||meeting.getTitle()!=title){//判断待加入会议是否存在
            return false;
        }
        bool flag=false;
        for(auto i : meeting.getParticipator()){//遍历参会成员
            if(i==participator){//判断该用户是否参与了此次会议
                flag=true;
                break;
            }
        }
        return flag;
    };
    auto switcher=[participator](Meeting& meeting)->bool{
        meeting.removeParticipator(participator);//删除会议用户
    };
    int cnt=m_storage->updateMeeting(filter,switcher);//更新会议状态
```

```
    auto filter1=[](const Meeting& temp)->bool{
        if(temp.getParticipator().size()==0){
            return true;
        }
        else{
            return false;
        }
    };
    m_storage->deleteMeeting(filter1);//若会议成员数量为0则删除会议
    return cnt>0;
}
```

（6）退出会议

```
bool AgendaService::quitMeeting(const std::string &userName, const std::string &title){
    auto filter=[userName,title](const Meeting& meeting)->bool{
        if(meeting.isParticipator(userName)&&meeting.getTitle()==title){//判断该用户是否参加了此会议
            return true;
        }
        else{
            return false;
        }
    };
    auto switcher=[userName](Meeting& meeting)->bool{
        meeting.removeParticipator(userName);//移除该会议成员
    };
    int cnt=m_storage->updateMeeting(filter,switcher);
    auto filter1=[](const Meeting& meeting)->bool{
        if(meeting.getParticipator().size()==0){//判断会议成员是否为0
            return true;
        }
        else{
            return false;
        }
    };
    m_storage->deleteMeeting(filter1);
    return cnt>0;
}
```

2、对于 AgendaUI 类

基本上所有函数的内部都是相关语句的打印以及对 AgendaService 类函数的调用，不需要过多的逻辑思维，所以这里不再细致讲解。

五、对于关键功能的测试

1、注册功能（r）

```
Agenda :~$ r

[register] [username] [password] [email] [phone]
[register] SYSU 123456 935087919@qq,com 13705642299
[register] succeed!
```

2、登陆功能（l）

```
Agenda :~$ l

[log in] [username] [password]
[log in] SYSU 123456
[log in] succeed!
```

3、列举所有用户（lu）

```
Agenda@SYSU :~# lu

[list all users]

name            email                   phone
cwt             123                     123
hyw             123                     123
yc              123                     123
wlz             123                     123
bbw             123                     123
SYSU            935087919@qq.com        13705642299
```

4、删除用户（dc）

```
Agenda@SYSU :~# dc

[delete agenda account] succeed!
```

此时登陆的账号为 SYSU

```
Agenda@hyw :~# lu

[list all users]

name            email                   phone
cwt             123                     123
hyw             123                     123
yc              123                     123
wlz             123                     123
bbw             123                     123
```

随机登录其他用户，再进行 lu 操作，发现成功删除用户 SYSU

5、创建会议（cm）

（1）会议时间有效，成员均存在，且不和其他会议冲突

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 2
[create meeting] [please enter the participator 1 ]
hyw
[create meeting] [please enter the participator 2 ]
wlz
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test1 2020-08-27/11:38 2020-08-27/11:43
[create meeting] succeed!
```

（2）会议时间无效（开始时间晚于结束时间或日期不合法）

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 2
[create meeting] [please enter the participator 1 ]
hyw
[create meeting] [please enter the participator 2 ]
wlz
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test3 2020-08-26/09:15 2020-08-26/09:10
[create meeting] error!
```

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 1
[create meeting] [please enter the participator 1 ]
hyw
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test3 10000-13-40/25:61 10001-13-40/25:62
[create meeting] error!
```

（3）会议标题重复（已经存在 test1 和 test4）

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 1
[create meeting] [please enter the participator 1 ]
bbw
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test1 2020-08-29/12:00 2020-08-29/12:05
[create meeting] error!
```

（4）部分会议人员不存在（未注册，这里指用户 wsa）

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 2
[create meeting] [please enter the participator 1 ]
hyw
[create meeting] [please enter the participator 2 ]
wsa
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test3 2020-08-26/10:00 2020-08-26/10:05
[create meeting] error!
```

（5）会议时间与主持人（sponsor）或者某位参会人员的时间冲突

```
Agenda@cwt :~# cm

[create meeting] [the number of participators]
[create meeting] 2
[create meeting] [please enter the participator 1 ]
yc
[create meeting] [please enter the participator 2 ]
bbw
[create meeting] [title] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[create meeting] test3 2020-08-27/11:35 2020-08-27/11:40
[create meeting] error!
```

test1 的会议时间为 2020-08-27/11：38~2020-08-27/11：43

6、添加会议成员（amp）

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,yc,bbw,wlz
test1       cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,wlz
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

```
Agenda@cwt :~# amp

[add participator] [meeting title] [participator username]
[add participator] test1 bbw
[add participator] succeed!
```

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,yc,bbw,wlz
test1       cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,wlz,bbw
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

成功在 test1 会议中添加成员 bbw

7、删除会议成员（rmp）

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,yc,bbw,wlz
test1       cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,wlz,bbw
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

```
Agenda@cwt :~# rmp

[remove participator] [meeting title] [participator username]
[remove participator] test1 wlz
[remove participator] succeed!
```

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,yc,bbw,wlz
test1       cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,bbw
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

成功删除 test1 会议中的成员 wlz

8、请求退出会议（rqm）

目前登录账号为 yc，从图中可知 yc 参加了 cwt 主持的 test4

```
Agenda@yc :~# la

[list all meetings]

title      sponsor     start time          end time            participators
test4      cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,yc,bbw,wlz
SYSU1      cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

```
Agenda@yc :~# rqm

[quit meeting] [meeting title]
[quit meeting] test4
[quit meeting] succeed!
```

```
Agenda@yc :~# la

[list all meetings]

title      sponsor     start time          end time            participators
SYSU1      cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

发现 yc 的会议列表中已不存在 test3

9、列举会议（以账号 cwt 为例）

（1）列举所有会议（la）

```
Agenda@cwt :~# la

[list all meetings]

title      sponsor     start time          end time            participators
test4      cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,bbw,wlz
test1      cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,bbw
SYSU1      cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
test2      hyw         2020-08-29/10:12    2020-08-29/10:17    cwt,yc
```

（2）列举所有主持的会议（las）

```
Agenda@cwt :~# las

[list all sponsor meetings]

title      sponsor     start time          end time            participators
test4      cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,bbw,wlz
test1      cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,bbw
SYSU1      cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

（3）列举所有参与的会议（lap）（不含自己主持的）

```
Agenda@cwt :~# lap

[list all participator meetings]

title      sponsor     start time          end time            participators
test2      hyw         2020-08-29/10:12    2020-08-29/10:17    cwt,yc
```

10、查询会议（以账号 cwt 为例）

（1）以会议标题为查询索引（qm）

```
Agenda@cwt :~# qm

[query meetings] [title]
[query meetings] test2

title       sponsor     start time          end time            participators
test2       hyw         2020-08-29/10:12    2020-08-29/10:17    cwt,yc
```

（2）以会议时间范围为查询索引（qt）

```
Agenda@cwt :~# qt

[query meetings] [start time(yyyy-mm-dd/hh:mm)] [end time(yyyy-mm-dd/hh:mm)]
[query meetings] 2020-08-27/10:00 2020-08-29/10:00

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,bbw,wlz
test1       cwt         2020-08-27/11:38    2020-08-27/11:43    hyw,bbw
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
```

11、删除会议（以账号 cwt 为例）

（1）以会议标题为删除索引（dm）

```
Agenda@cwt :~# dm

[delete meeting] [title]
[delete meeting] test1
[delete meeting] succeed!
```

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test4       cwt         2020-08-27/10:45    2020-08-27/10:50    hyw,bbw,wlz
SYSU1       cwt         2020-08-29/09:20    2020-08-29/09:25    hyw,yc
test2       hyw         2020-08-29/10:12    2020-08-29/10:17    cwt,yc
```

（2）删除个人创建的所有会议（da）

```
Agenda@cwt :~# da
[delete all meeting] succeed!
```

```
Agenda@cwt :~# la

[list all meetings]

title       sponsor     start time          end time            participators
test2       hyw         2020-08-29/10:12    2020-08-29/10:17    cwt,yc
```

六、实现过程中遇到的问题

此次大作业过程中，我遇到了不少的问题，但主要集中在两个方面：Storage 类和 AgendaService 类。

在写 Storage 类的函数时，我一开始对文件读写的概念和用法并不清晰，之后花了一段时间去自学才明白了如何用代码来创建.csv 文件并对其中的数据进行读写。

之后我对 filter 和 switcher 函数也有一点畏难情绪，看完 FAQ 也不太理解。不过在参考了很多网站的讲解之后，也对这个概念有了一定程度的理解，明白了 filter 和 switcher 实际上是用于将 AgendaService 类和 Storage 类联系起来，使得 AgendaService 类的函数

能够传达命令给 Storage 类的对应函数，最终达到修改.csv 文件的数据的目的。

最后在写 AgendaService 类的函数时，我经常因为考虑不到一些边界情况所以导致代码经常过不了机器测试。最后在参考了一部分网上的代码后，我通过自己的努力重新修改了我的 AgendaService 类代码，并成功通过了机器测试。

七、实验总结

在完成了此次 Agenda 项目之后，我学习到了很多新知识并且对原有知识的应用更加熟练。在此次大作业中，我学会了 git 的各种基本操作（push, reset, check-out 等等）。了解了三层架构的基本概念，并明白了三层架构相较于两层架构的优势所在。最后对于文件的读写也有了一定的基础。总体来说，这次大作业让我对于 C++的封装特性和面向对象编程有了更深层次的理解，也明白了为什么我们会推广面向对象编程。