2021《面向对象程序设计训练》大作业

一言一行一念

Every Word, Every Action and Every Thought

Demo By FanJingtao

说明文件

1. 声明

- 多为零散时间投入，精力有限，设计/编码还有很大可优化空间。
- 不保证注释符合学堂发布的编码规范，仅做了少量帮助理解的必要性说明。
- 不保证命名方式符合学堂发布的编码规范，没有脱离自己平时的风格。
- 不保证没有错别字，请批判性理解。
- 综上，仅做参考，不是打分依据或成绩判定标准。

2. 通用类设计

主要考虑：写类是为了给人用，而不是为了仅用于作业要求，这样才可以使类发挥更大的作用。

2.1. User 类

2.1.1. 涉及用户信息，一般管控非常严格。为了防止非授权新建/复制用户，构造函数设为受保护成员、删除默认构造函数、拷贝构造函数、赋值运算符，从根源上断绝了任意产生 User 对象的各种可能性。

2.1.2. 为了从代码层面杜绝类的使用者获取用户信息的可能性，必须经过用户名和密码验证（verify）函数，才能获得 User 指针。只有在这种情况下，才认为这一操作是必须经过用户授权的。验证过程只能是静态的，可不通过 User 指针进行调用，否则，与必须先验证才能获得 User 指针向矛盾。

2.1.3. 新增用户、删除用户，本质是 User 类的行为（静态），但考虑需要管理员权限才能新增和删除用户，并没有将这些函数设为静态，而是受保护的非静态成员，是为了通过具体的 Administrator 指针调用。

2.1.4. 修改密码是某个用户的行为，只能修改自己的密码，所以是非静态的。

2.1.5. 是否是管理员，在 User 和派生类对象上，行为名字相同，但行为的实现和结果不同，是典型的通过派生实现多态，故设计为虚函数。

2.1.6. 无论是用户名、密码，还是全体 User 对象的指针数组，都是出于分层次功能分解考虑，设置为私有。即使是 Administrator，也只能通过 User 的共有和受保护成员函数受控访问，不能直接操作。

2.1.7. 保存用户到文件、从文件读取并创建用户的行为，满足 ALL 原则，故设计为静态。但并不是直接操作到用户名、密码等私有数据成员，而是通过分而治之方法，调用每个 User 对象的流插入和提取运算符实现。

2.1.8. 流插入和提取运算符，是 User 将自己的数据进行保存和读取的过程，故应设计为非静态。但插入和提取行为的发出者不是 User 而是流，智能重载为友元函数。

2.2. Administrator 类

2.2.1. 通过 Administrator 对象/指针调用新增用户、删除用户，本质上就是调用 User 基类的相关函数。因此用过 using 进行提升访问权限，比自己再次封装函数代码更精炼。

2.2.2. 判断是否为管理员功能，通过虚函数重载实现不同行为。

2.3. MD5 类

2.3.1. 仅为加密过程封装，一般来说只提供静态函数就好。但希望 MD5 保存加密后的密文，所以也提供了通过对象进行加密、比较等行为。

2.3.2. 从功能内聚的考虑，通过运算符重载提供了等于、不等于，仅仅在密文之间进行比较的功能。

2.4. CipherText 类

2.4.1. 此类存在的意义仅在于增量式功能扩扩展，提供了明文通过加密后与密文对象的比较。

2.4.2. 出于功能分层实现的考虑，CipherText 虽提供了关系运算符、流插入与提取运算符的重载，但均是对 MD5 相关函数的组合调用实现的。

2.5. Date 类

2.5.1. 构造、拷贝、赋值均是应有操作，故没有删除。

2.5.2. 是否是闰年、日期是否合法，既有静态也有非静态的方法，方便使用。

2.5.3. 通过派生得到 DateTime 类是可能存在的类代码重用行为，所以，析构函数设置为虚函数。虽然没有用，但基类析构函数设为虚函数是良好习惯。

2.5.4. 大、小、等、不等之类的日期关系运算时常规操作，但重载为运算符比自己写函数更为合适，更符合大多数人的编程习惯和理解方式。

2.5.5. 私有的 Set 函数，用来调用日期合理性验证、星期几计算，并被构造函数、拷贝、赋值等行为调用。

2.5.6. 自增自减运算符用于日期增减，用简单的方式封装了跨年、夸月、闰年、日期是否超出最大最小范围等一些列复杂运算。

2.5.7. 日期不能相加，但可以相减。通过减法运算符重载提供了求日期差的功能。

2.5.8. InDaysFromAD 函数用于计算从公元 1 年 1 月 1 日到指定日期的天数，是两个日期大小比较、日期差、星期几计算的通用核心算法。可以设为私有，也可以设为公有，两者相权衡，设为公有，提供更多的使用可能性。

2.5.9. Date 类算法实现部分也很有趣，考虑了 1582 年前后的历法变化，是万年历算法，因此并没有显式设定/改变日期许可范围的

功能。

2.5.10. 允许的最大、最小日期、获取当前系统日期，均不需要通过对象调用，不依赖与任何 Date 对象的存在，符合 ZERO 远侧，因此是公有静态的。

2.6. Time 类

2.6.1. 时间类与日期类设计思路和考虑是相通的，不在赘述

2.6.2. 不同点在，时间的自增自减运算是循环的。

2.7. DateTime 类

2.7.1. DateTime 类型存在，是为了通用性考虑，很多环境是需要联合使用 Date 和 Time 两个数据的。

2.7.2. 如何得到 DateTime 类，有多重可能性：1）从 Time 派生得到 DateTime，可以看做是对时间的更泛化描述，但需要重写 Date 相关代码，亏了；2）从 Date 派生得到 DateTime，可以看做对日期的细分，但需要重写 Time 代码，还是亏了；3）组合关系；4）多继承。后两种方式实现方式不同，但从对象内存布局上，基本是一致的。我采用了多继承方式。

2.7.3. 关系运算符、自增自减运算符、减法运算符重载，仅仅考虑时间和日期比较、变化的联动，日期和时间各自的变化是通过 Date 和 Time 自身的运算符实现的。这也是功能分配和分层的思路。

2.7.4. 日期时间的范围 DateTimeRange，可以看做是两个时间日期组成的 pair，stl 中有这种数据类型，故用 using 进行了类型定义和命名

2.7.5. 获得日期范围，是日期相关操作，是类的行为（静态）。进一步考虑开闭区间，提供 Between（闭区间）、From（开区间）、Until（开区间），All（开区间）共 4 中静态函数，直接生成特定 DateTimeRange 实例。

2.8. Message 类

2.8.1. 因为要保存和读取全部 Message，所以与 User 类似，禁用默认构造、拷贝、赋值等操作，避免消息重复。

2.8.2. 日期时间、类型、标题、内容，是通过组合实现的。

2.8.3. 消息类型的枚举、排序类型的枚举，均定义为 Message 的公有内嵌类型，是为了强化其余 Message 的关联关系，强化使用者印象。使用的时候也是 Message::Type::……，Message::SortType::……，更方便记忆。

2.8.4. 新增、删除、保存、读取消息的行为，不能由 Message 对象/指针发出，ZERO 原则，应设为静态。

2.8.5. 消息搜索，也是静态的。进一步考虑，无论何种搜索，返回值都是数组，日期范围、关键词列表是否作为搜索依据，是可选的，有三种组合，故只实现一种最全面的搜索，其余两种通过接口重载实现，仅为第一种行为的特化封装。

2.8.6. Search 通过两种内嵌的函数对象类 Filter 和 Sorter 实现。因此，Filter 和 Sort 是针对 Message 类特化设定的，故设置为 Message 的私有类型。

2.8.7. Filter 和 Sorter，共性是需要日期范围、关键词列表、排序方式控制，只有控制参数取值不同，但算法并无不同。所以使用重载 operator()的类实现，而不是用 Lambda 表达式。

2.8.8. 构造函数私有化、全局对象指针列表等行为的存在于 User 是类似的，不再赘述。

3. 业务流程/控制器类设计

3.1. ControlerAbstractBase 类

3.1.1. 控制器类是没有界面的软件，但其与界面显示还是相关的。此类型设计为所有控制器类的基类，也是无法实例化的抽象类。

3.1.2. 控制器的共同行为是检测通用类的异常、返回表示执行结果的特定枚举值。这些行为涵盖了 :读取/存储、登录、创建新用户、搜索、提交等。

3.1.3. 其中，最特殊的行为是搜索，因为控制器是与界面耦合的，搜索结果在控制器中如何被组织和转换，与界面显示形式相关。搜索过程的行为和算法是通用的，因此被定义为受保护的，可以被派生的控制器类调用。

3.1.4. 同时，不同的地方在于：一条消息被如何转化为交给界面显示的字符串（MessageToString 行为）、界面显示的 “言/行/念”、

“Word/Action/Thought” 等有界面决定的字符串如何被转化为枚举值（StrToMessageType 行为）。

3.1.5. 在所控制器上都共通的行为，在基类中实现。在各个控制器（比如与 QT 界面结合的控制器、与控制台黑框结合的控制器）上行为实现不同，但都必须都有的行为，在基类中声明为纯虚函数，强制要求各个派生类要给出自己的具体实现。

3.1.6. 基于这种设计，控制器被界面调用公有函数，返回特定类型的值，不同环境下（编译器、开发环境、操作系统）的控制器仅需对 ControlerAbstractBase 做继承和两个纯虚函数的编程，就可以进行扩充和修改。

3.2. Controler_QT 类和 ControlerCommandLine 类

3.2.1. DEMO 中，提供了为 QT 图形化界面和控制台黑框界面设计的两种派生控制器类，大家可以对照比较。

3.2.2. 两者在对外提供功能上是完全一致的，不同之处在于两个纯虚函数重载的行为不同，一个是为了在 GUI 界面上以列表形式显示，另一个是为了在控制台黑屏幕上输出。

4. 界面类设计

4.1. 重点考虑

界面创建的实际、谁创建谁，谁显示谁，怎么构造，怎没销毁。

4.2. 构造函数修改

QT 工程中，修改了所有窗口类的构造函数，增加了控制器的引用作为参数，并用私有成员进行保存。这样可以只创建 1 个控制器实体，所有窗口共用。

4.3. 窗口类型

在窗口关联的创建、销毁、显示中，除了登录界面外，其余窗口都是从 QDialog 继承的，就有 exec 函数，进行模态显示（请百度），并且创建窗口用的是 shared_ptr，自动销毁，可以多次创建，不需要考虑初始化时序等。

4.4. 与控制器联动

在主函数中声明了控制器类，传递给登录界面做参数，登录界面在创建管理员和用户界面的时候继续传递。

4.5. 去业务逻辑

界面类，不需要包含任何功能逻辑：日期是否为空、标题是否为空、内容是否为空，创建用户是否有管理员权限……，这些都是在控制器中实现的，界面仅仅是输入和显示。

5. DEMO 目录结构与依存关系

5.1. Models 目录

包含了 User、Administrator、MD5、CipherText、Date、Time、DateTime、Message 类的头文件与源文件。

5.2. Controlers 目录

包含了 ControlerAbstractBase 的头文件与源文件。

5.3. Controlers 目录下的 Controler_QT 目录

包含了派生自 ControlerAbstractBase 类的 Controler_QT 类的头文件和源文件。

5.4. Projects 目录下的 MAC/WINDOWS_QT_5_14_2 目录

包括了可在 Windows 和 macOS 下以 QT 编译的工程文件、main.cpp、窗口类的 ui 文件、头文件、源文件。

5.5. Projects 目录下的 MAC_XCODE_12_5_1 目录

包括了可在 macOS 下以 xcode 编译的工程文件、main.cpp。

5.6. 在其他环境中复用

xcode 中的 main.cpp 文件，同样可以适用于 vsCode、VS，在 windows 系统中生成控制台程序。

5.7. Data 目录

用与测试的 User.txt 和 Messages.txt 文件