**社交应用的数据存储与实现**

**说明文档**

**熊伟伦 5120379076 F1203703**

**简介：**

该程序属于《程序设计2》课程大作业，按照《社交应用的数据存储与实现》的要求实现。

语言标准：C++11

开发环境：Qt Creator 2.8.1

编译器：MinGW 4.8.0（g++ 4.8.0 for Windows）

其他工具：GitBash、WinHex、Notepad++、Windows计算器、Windows画板、Office

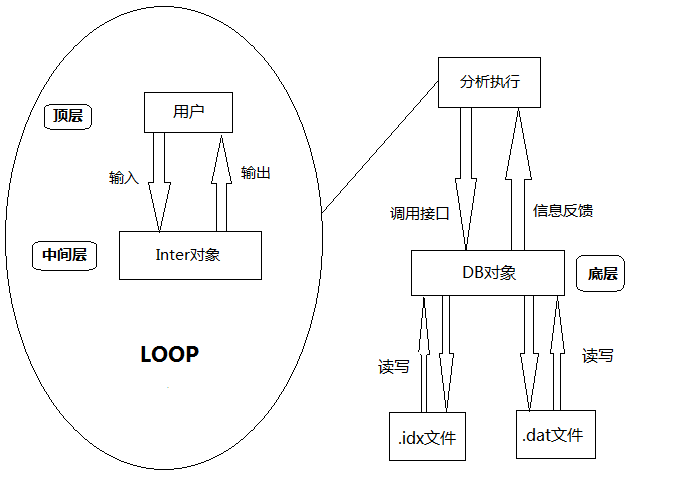
实现《社交应用的数据存储与实现》所要求的全部“基本功能”。

实现《社交应用的数据存储与实现》提到的**全部**“扩展功能”。

配合PPT观看风味更佳。

**程序实现：**

**一、总体架构**：

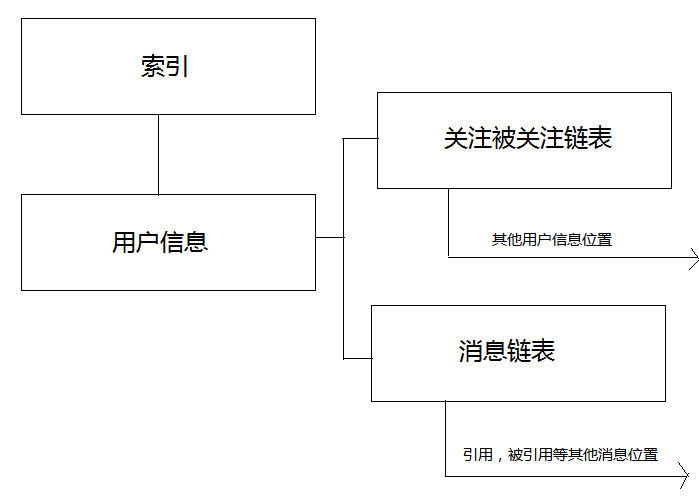


**二、后台：**

**1.后台简介：**

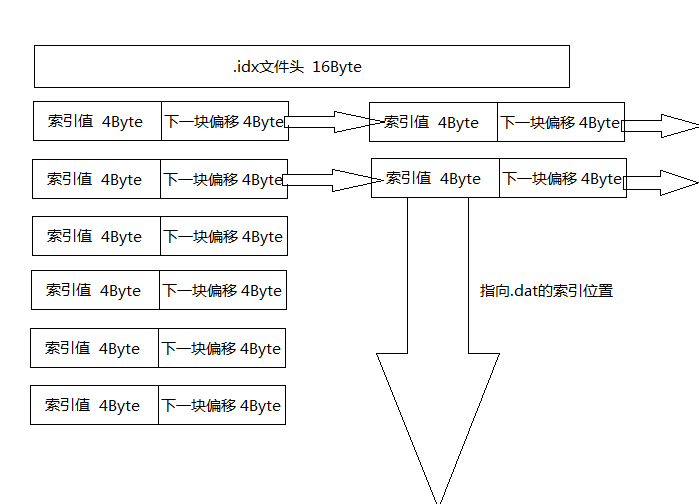
后台底层代码主要保存在db.h和db.cpp文件中，使用DB类在程序中构成一个数据库对象供前台后台交互的代码进行调用。包装底层数据库用于与前台代码调用的函数主要保存在cmd\_inter.h和cmd\_inter.cpp的Cmd\_Inter类中，以get\_xxx，write\_xxx为函数名。

数据库使用.idx索引文件和.dat数据文件，每个文件都有特定的文件头供程序进行判定是否是合法的该类型文件，如果不是则会进行文件初始化。DB类中包含私有变量fstream idx，fstream dat与.idx文件和.dat文件进行交互。

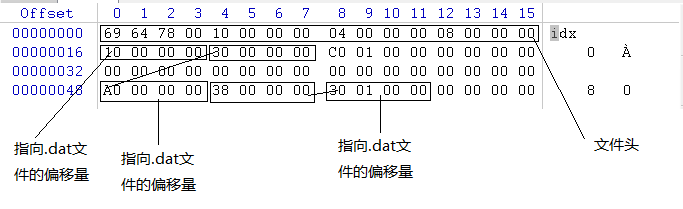


**2.后台索引**：

索引的基本数据结构使用散列表，解决冲突使用拉链法。



以4个桶为例，16到47 Byte为初始的桶，16~19指向.dat文件的某个用户信息的开头，20~23根据开链原则指向下一个链节48(0x30)，48~51又指向某个用户信息的开头，同理52~55指向该桶索引链的下一节56(0x38)。



桶的个数保存在 #define BUCKET N

供交互程序调用的搜索接口 string db\_dat\_read\_string(**int** pos, **int** length);

每个散列表的链的一节由2个byte组成，前一个byte保存对应用户名在.dat文件中的偏移量，第二个byte保存.idx文件中该链下一节的偏移量。

对索引文件的操作，另提供辅助函数

**int** db\_idx\_get\_1st\_4byte(**int** pos);

**int** db\_idx\_get\_2nd\_4byte(**int** pos);

**void** db\_idx\_write\_1st\_4byte(**int** pos, **int** value);

**void** db\_idx\_write\_2nd\_4byte(**int** pos, **int** value);

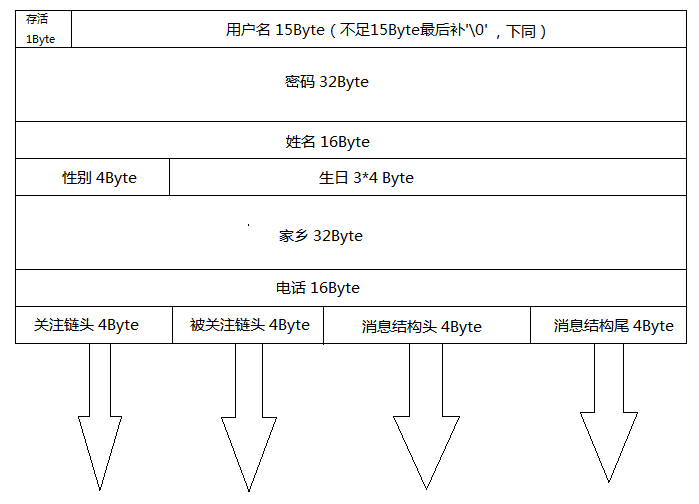
**int** db\_idx\_alloc(**int** size);

**bool** db\_link\_idx\_key(**int** key\_pos, string key\_name);

帮助前台与后台交互的代码进行操作。

**3.后台数据结构：**

a) 用户信息结构



注：“存活”表示该用户是否被删除。

最后一行的16个Byte用于索引用户的关注列表，被关注列表，消息链表。

在Cmd\_Inter类中有下列函数通过调用底层DB类的基本读写函数进行操作，简化了前台代码的实现。

**void** write\_id(**int** begin\_pos, string id);

**void** write\_exist(**int** begin\_pos, string exist);

**void** write\_password(**int** begin\_pos, string password);

**void** write\_name(**int** begin\_pos, string name);

**void** write\_sex(**int** begin\_pos, **int** sex);

**void** write\_birth\_year(**int** begin\_pos, **int** year);

**void** write\_birth\_month(**int** begin\_pos, **int** month);

**void** write\_birth\_day(**int** begin\_pos, **int** day);

**void** write\_hometown(**int** begin\_pos, string hometown);

**void** write\_phone(**int** begin\_pos, string phone);

**void** write\_follow(**int** begin\_pos, string id);

**void** write\_be\_follow(**int** begin\_pos, string id);

**void** write\_speak\_head(**int** begin\_pos, **int** val);

**void** write\_speak\_tail(**int** begin\_pos, **int** val);

string get\_id(**int** begin\_pos);

string get\_exist(**int** begin\_pos);

string get\_password(**int** begin\_pos);

string get\_name(**int** begin\_pos);

**int** get\_sex(**int** begin\_pos);

**int** get\_year(**int** begin\_pos);

**int** get\_month(**int** begin\_pos);

**int** get\_day(**int** begin\_pos);

string get\_hometown(**int** begin\_pos);

string get\_phone(**int** begin\_pos);

**int** get\_follow\_pointer(**int** begin\_pos);

**int** get\_be\_follow\_pointer(**int** begin\_pos);

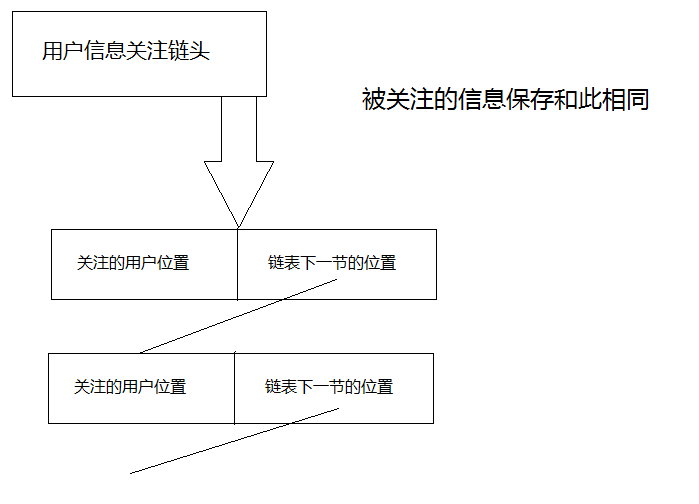
**int** get\_speak\_head\_pointer(**int** begin\_pos);

**int** get\_speak\_tail\_pointer(**int** begin\_pos);

b) 用户关注与被关注列表结构

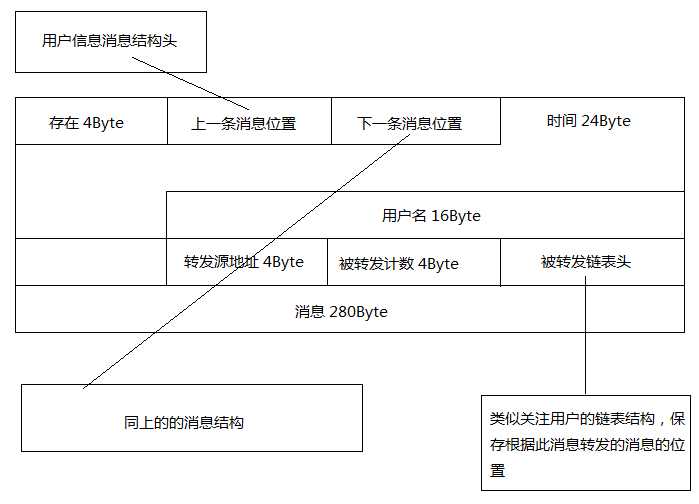
保存关注与被关注的用户列表，由每个用户信息的关注的链头指向该链表结构。

在前台使用flist和bflist即会遍历登陆用户的该链表结构，并显示关注，被关注的用户名。



注：删除关注时关注的用户位置设置为 \0\0\0\0，之后添加关注时会使用该位置，不造成硬盘空间浪费，被关注的结构同理。

c) 用户消息结构



注：“存在”表示该消息是否被删除，如果被删除，之后添加进的消息会占用此位置，不造成硬盘空间的浪费。

转发源地址保存了该消息所转发的消息的位置，如果是原创消息，则为\0\0\0\0。

转发链表头保存了以此消息为源所转发的消息，保存形式类似用户关注链表。

**4.后台与前台交互的部分算法概述**

由于这部分实现内容较多，固简述部分算法。

1. 添加新的关注用户

添加新的关注用户时，会根据登陆用户的信息位置，找到用户关注链头，并且从链头开始进行搜索。如果搜索到关注用户的位置为\0\0\0\0（删除关注用户造成的），则会将新关注的用户位置写在此处，否则继续遍历链表，直到指向下一节的位置为\0\0\0\0，则会在文件后alloc一块新空间保存这一节。

1. 删除关注用户

类似于添加关注用户，删除关注用户同样找到用户的关注链头，遍历判断是否为所删除的用户，如果是，则链表中关注的用户的位置设为\0\0\0\0，并退出该操作。

1. 发布新消息

用户登陆后输入speak会发布新消息，或者ls查看关注用户后使用trans操作可以转发并评论其他消息。两种操作都属于发布新消息。发布新消息，程序会根据用户信息位置找到消息连表头并遍历，直到找到一个被删除的消息的位置或者遍历到最后一条消息，程序会填补或者alloc新的空间写入该消息，并且重新维护链表结构以及用户信息指向链表的头和尾的索引。

对于转发的消息，新的消息会将转发源消息的被转发次数加上1，并且转发源消息保存了其转发源，因此也可以做同样的事，一系列转发而来的消息的被转发次数都会增加。除此之外，转发源的被转发链表也会添加新消息的位置，用于删除操作。

1. 删除消息

由于在消息的数据结构中维护了一个被转发链表，因此，删除消息除了设置本身消息的存在为否之外，还会遍历被转发链表对转发该消息的消息做同样的删除操作，这些被间接删除的消息也会做同样的遍历转发链表删除的事情，因此，所有与该消息有关系的消息全部被删除。

1. 折叠消息

在显示消息的时候，使用一个向量保存所有的消息并对发布时间进行排序，然后从最新发布的消息开始遍历判断其他消息是否为该消息的转发源，如果是，则进行消息的折叠，省略公共的转发部分。

**三、前台：**

**1.前台简介：**

前台做成了类似于“命令行”模式的交互界面，这样用起来感觉很cool。

**2.前台命令介绍：**

help 显示帮助界面

quit 关闭程序

register 注册用户

cls 清屏

cd 查看某用户

infor 在查看状态以外，显示自己的信息，查看状态，显示被查看者的信息

change 改变自己的信息

del 删除登陆的用户

spfind 根据条件搜索用户，- 和 -1 表示忽略条件

recommend 好友推荐，推荐老乡

follow 关注某用户

unfollow 解除关注某用户

flist 显示关注列表

bflist 显示被关注列表

speak 发布新消息

trans 在ls状态下转发消息

ls 显示新鲜事

ls –i 显示个人发布的消息

login 登陆

logout 登出

在某些操作内部还有其他命令，例如-q 等，在执行时会有相应提示。

**3.前台结构：**

Cmd\_Inter.run 进行主循环，循环调用 Cmd\_Inter.getInput 读取用户指令，保存到 cmd\_type 进行分词，判断并调用Cmd\_Inter.exe\_xxx函数执行相应操作，这些函数包装并且调用与底层交互的函数。

**四、测试：**

配置简述：

CPU Intel Core i5-3210M 双核 2.5GHz

内存 8G DDR3 1600MHz

硬盘 希捷1TB 5400转/分

**1. test 01**

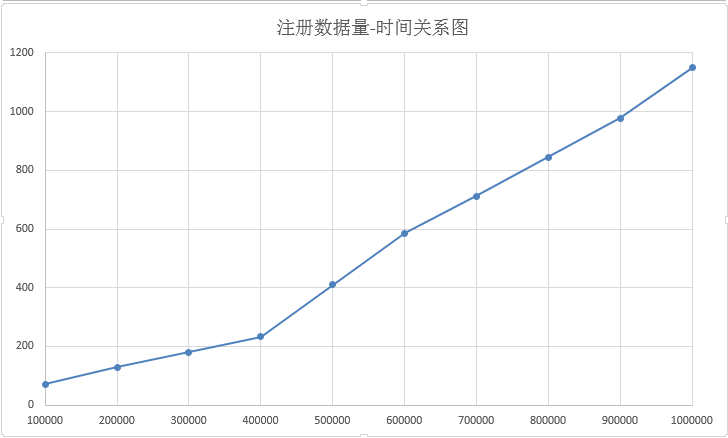
测试注册的正确性，条件查找的正确性，关注，解除关注的正确性。

注册100万个用户并且对其进行条件搜索，判断结果是否和注册一样，并且关注这些用户，结论是正确的。（个人认为正确性测试完全没有必要测试100万级别的数据量）

测试代码在 Cmd\_Inter.cpp::test\_01 函数中，下同。（解除main.cpp中的注释即可测试）

**2. test 02**

测试注册用户的性能



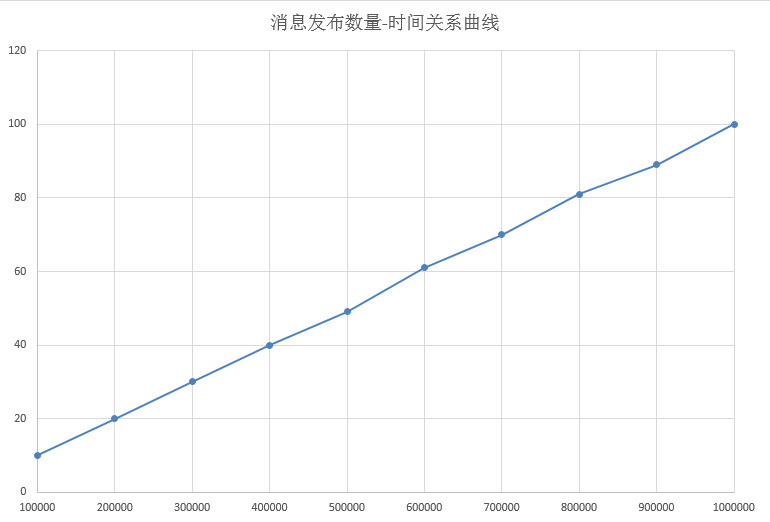
如图所示显示的是注册账户的数量与消耗时间的关系图，数据从10万到100万每隔10万测一组，关系线条基本符合直线增加，有个趋势数据量越多时间增长得越快。这主要是由于拉链法造成的。可以知道，当数据趋近于无限远超过桶的数量并且哈希均匀的话，时间复杂度是O(n)，之后注册一个用户的时间开销越来越大。

由于测试时间很长，最长长达接近20分钟，并且在循环过程中根据代码来看，理论上不会一直申请额外的内存空间，每次循环都会删除掉这一次所用的空间。因此可以直接通过任务管理器观察到是否有内存泄露。在100万数据量的测试过程中，根据任务管理器，程序的内存占用量一直保持972KB，因此认为不存在内存泄露。

特别测试了在100万数据时，初始化散列表桶数为100万个，初始时索引文件大小7813KB，测试结束后索引文件大小12513KB，计算得到大约有60.16%的数据发生了哈希冲突，进行了开链。介于哈希函数是我自己写的，有这样的平均分配的概率基本可以接受。

**3. test 03**

测试用户发布消息的性能



注册一个用户并发布100万数量级的不同的消息，发布完成后test04会检测正确性。

性能测试每隔10万的数据量取一个点，最后得到的曲线十分标准（小伙伴们都惊呆了）近乎直线。

可见虽然消息的存储使用的是链表，但是由于使用的是双向链表并且保存了链尾，插入新消息可以直接搜索到最后一个消息后进行添加，因此任何时候发布消息的时间复杂度都为O(1)。

对于删除消息留下的空隙，程序会定期检查空隙将下一次消息存放在此进行填补，不会影响发布消息的复杂度或者造成大量的空间浪费，因此不再测试。

**4.test 04**

测试用户发布消息的正确性

创建用户，并重复发布100万条不同的消息，随后读取这些消息，判断是否和发布的消息一致。结果是一致的，说明消息的正确性是没有问题的。

不过随着同一个用户发布的消息越来越多，读取同一个用户的某个消息的开销也越来越多，时间复杂度是O(n)。这是用链表存储用户消息的缺陷。

不过程序对存储消息采用了双向链表的方式，这样既能快速读取最新发布的消息，又能快速读取最早发布的消息，具有很大的灵活性。

**五、总结：**

总的来说，整个project包括文档花费的时间挺多，不过应该也还没到100小时。

个人觉得做的比较好的地方有：

1. 使用双向链表保存消息，降低添加新消息的时间复杂度到O(1)。
2. 将转发消息的转发源保存了，从而能够递归地增加转发次数。
3. 保存了转发该消息的消息地址，从而能够递归地删除所有和该消息有关的消息。
4. 消息的时间保存了年、月、日、时、分、秒，能够根据从前到后进行比较来对消息的时间进行排序，速度较快。
5. 完成了类似于命令行的前台交互，感觉很cool。
6. 自己设计了哈希函数，测试得到的分配还是比较平均的。

个人觉得不足之处：

1. 没有使用特别高级的数据结构，例如B树。
2. 部分附加功能仅限于满足“有”这个功能，例如关注推荐（推荐家乡一样的用户），没有做些很cool的功能。
3. 正确性的测试报告没有写的很完整具体详细，但是我在每做一个功能都进行了大量的测试确保正确。