

**课 程 设 计 报 告**

**题目： 基于图的系统日志事件关联分析**

**课程名称： 数据结构**

**专业班级： 计算机1609班**

**学 号： XXXXXXXX**

**姓 名： 王 通**

**指导教师： 袁 凌**

**报告日期： 2018年2月27日**

**计算机科学与技术学院**

任务书

## 一、设计目的

掌握图等结构的存储结构与基本算法，通过解决较复杂的基于图模型的实际问题，提高学生对数据结构知识综合运用的技能与实践能力。

## 二、 设计内容

设计有效的逻辑数据结构与存储结构表示WINDOWS或LINUX或IOS系统日志事件信息及其相关关系，例如事件的级别、日期时间、来源、事件id、任务类别、用户及其它事件属性等；分析与表示事件的各种相关关系，如时间相关，数据相关，因果关系等，并用图结构表示它们；进一步在相关图上进行分析与推理，预测某些可能发生的事件，追溯已发生故障或事件的根源。

## 三、主要数据对象

系统日志中的事件集；分析所产生的关联关系集或者规则集。

## 四、主要数据关系

（1）根据事件的发生时间先后，事件之间可能是线性关系；从单纯事件查询的角度，事件集可能构成普通查找表。

（2）另外，事件之间可能还有复杂的关系，如两个事件的主体可能是同一任务/进程；一个或多个事件可能直接导致另一事件的发生；多个事件中的不同主体可能对同一系统对象（文件，或设备等）进行了操作等。

## 五、设计要求

（1）交互式操作界面(并非一定指图形式界面)；

（2）事件插入、删除、查找与检索等：支持按时间查找、事件主体与客体等查找；

（3）基于日志事件集进行事件关联图的建立与存储 (如主体相关、客体相关、相伴事件、因果关系等)并提供查询功能；

（4）应用关联关系或规则进行系统行为推理、预测；有两种解决途径，依据意愿自行选择：

（4.1）简单方案：根据新发生的事件，查找过往相似事件，并在（3）中的原始相关图上查询相似事件的邻接点事件或以相似事件为起点的路径，作为简单预测的依据。追踪索源则可以查询逆向路径。

（4.2）高级方案：基于原始事件关联图，进行统计分析获取频繁子图模式或关联规则，并表示为关联模式或规则图；并对模式或规则图进行管理：插入、删除、修改、查询、文件存储等。基于模式或规则图进行预测与追踪索源。

（5）事件文件的存储管理，读与写。

## 六、设计提示

从WINDOWS/LINUX/IOS系统日志或事件查看器获取系统事件原始数据，抽取事件有导航关数据与信息并以文件形式存储，用图模型建模事件相关关系并以文件保存这种原始关系。预测与溯源可在图模型上进行简单的路径查找与统计分析而给出直接结果。如果有兴趣，可采用概率图模型及其概率推理方法或者关联规则分析方法等高级技术进行分析，但这类方法涉及一些需要自己学习的背景知识，不作强制设计要求，然而，图模型的应用则是强制性的。不允许使用数据库系统管理数据。在界面设计与其他功能上可自由发挥。

## 七、参考文献

[1] 严蔚敏, 吴伟民. 数据结构（C语言版）. 北京: 清华大学出版社,1997

[2] 王晓东. 计算机算法设计与分析. 北京: 电子工业出版社, 2007

[3] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[4] 傅成兵. 系统日志技术研究与编程实现.计算机与信息技术，2009,( Z1):94~97

目 录

**任务书** I

**1□□□引言** 1

1.1□□×××××× 1

1.2□□×××××× 3

1.2.1□□×××××× 7

1.3□□×××××× 10

**……**

**……**

**……**

3**□□□××××××** 20

3.1□□×××××× 20

3.2□□×××××× 23

3.2.1□□×××××× 25

3.3□□×××××× 30

**……**

**……**

**4□□□** 40

**参考文献** 44

**附录□××××××** 45

（章为宋体小4号加粗，其余宋体小4号，字母、阿拉伯数字为Time New Roman小4号）

1 引言

## 1.1 课题背景与意义

在比较大规模的网络应用或者对安全有一定要求的应用中，通常需要对系统的日志（Syslog）进行记录分类并审核，默认情况下，每个系统会在本地硬盘上记录自己的日志，这样的做法存在很多缺点：首先是管理不便，当主机数量比较 多的时候，逐个查看每台主机的日志会十分麻烦和繁琐，其次是安全问题，一旦有入侵者登录到系统，便可以轻松的清除所有日志，系统的日志信息将不复存在，安全分析人员就获得不了任何入侵信息。因此，在网络中安排一台专用的日志服务器来记录系统日志是一个比较理想的方案。

## 1.2 课程设计的主要研究工作

本课程设计主要从三个方面对系统日志进行管理分析：

1. 基本日志的增删改查及磁盘介质的存储和读取。
2. 基于日志的关联图的建立，包括主体相关、客体相关、相伴事件、因果关系等，其中支持关联图的存储和读取。
3. 基于关联图的分析和预测。

2 系统需求分析与总体设计

## 2.1 系统需求分析

完成一个系统日志管理功能，即对日志进行增加、删除、修改、查询等功能的软件，进一步在日志数据文件的基础上进行关联图的建立和存储，并提供关联图的查询功能，关联图的类型大致分为主体相关，客体相关，相伴事件，因果相关四类。在此基础上，分析当一个新的日志事件产生时，推测产生的原因，并预测其后会发生的结果。

## 2.2 系统总体设计

系统大致分成四个个模块，第一个模块是日志管理，处理日常的日志文件；第二个模块是关联图部分，将系统中的日志文件进行分析；第三个模块是行为分析与预测，通过大量过去的日志文件对新的日志进行分析和预测；第四个是界面布局及管理，设计程序的GUI界面，以及设计一个管理器将前三者的功能汇总到一起。

### 2.2.1 日志管理

日志管理用于存放系统日志，并且提供日志的增加、删除、修改、查询四个基本功能，其中查询和删除两功能提供多种方式进行，比如按照日志ID，按照进程等等。另外，日志模块里面提供日志的文件保存和文件读取功能。日志功能详见下图所示：

### 2.2.2 关联图

关联图部分功能分别是：根据现有日志数据文件建立关联图，通过日志中主客体；伴随以及因果这四个维度的关联，建立一个关联图表示他们的关联关系；关联图的存储和读取，通过C++的流文件对关联图进行保存以便下次程序运行时直接读取；关联图的查询，通过给出的日志ID得到与该ID所代表的日志相关联的日志；关联图的绘制，读取关联图建立这一步得到的图后，将该图绘制成图形表示出来。

### 2.2.3 行为分析与预测

行为分析与预测分成三个模块，第一个模块是行为分析，分析该日志出现的原因；第二个模块是行为预测，预测该事件发生后引发的事件；第三个模块是将这个新事件添加到日志记录中。

### 2.2.4 界面布局及管理

界面布局及管理分成两个模块，第一个模块是界面布局，将系统的功能通过GUI图形界面展示出来；第二个功能是管理，将前面三个功能汇总到一起，进行统一的管理，提高程序的聚合性降低程序的耦合性。

综上四个方面，可以得出以下的系统模块结构图：

这部分可根据用户需求，设计和规划一个系统，说明清楚系统应该有哪些功能模块，每个模块做什么。最后给出完整的系统模块结构图。

以下是文中有图标的形式，要求对图标引用说明。

××××××××××××××××××××××××××××××××××××××××××××××××，其×××××可表示如下：

 (3-1）

 (3-2）

×××××××××××××××××××××××××××× (如表3-1所示)

表3-1□××××××××××

|  |  |  |  |
| --- | --- | --- | --- |
| ××××× | ××× | ××× | ××× |
| ××××× | ×× | ×× | ×× |
| ××××× | ×× | ×× | ×× |
| ××××× | ×× | ×× | ×× |
| ××××× | ×× | ×× | ×× |

（表标题：位于表格上方，黑体小4号，字母、阿拉伯数字为Time New Roman小4号，表内容：宋体5号，字母、阿拉伯数字为Time New Roman 5号）

××××××××××××××××××××××××××× (如图3-1所示)



图3-1□××××××××××

（图标题：位于图下方，黑体小4号，字母、阿拉伯数字为Time New Roman小4号）

.......

--------章与章之间插入分页符----------

3 系统详细设计

## 3.1 有关数据结构的定义

### 3.1.1数据结构的定义

（1）日志文件的类型

系统需要处理两种数据类型，第一种数据类型是日志文件，第二种数据类型是关联图文件。日志文件包括：日志ID，日志名称，日志来源，发生时间，任务类别，级别，用户，进程ID，关键字，描述十个数据项。如图3-1（a）所示：



图3-1（a） 日志文件数据结构

（2）关联图类型

关联图文件包括顶点集，顶点个数，邻接矩阵，邻接顶点映射表这四个数据项，具体表示如图3-1（b）所示：



图3-1（b）关联图文件数据结构

### 3.1.2数据关联

分别为日志文件和关联图文件制作两个管理器，命名为日志文件管理器和图文件管理器。日志文件管理器存放所有日志的地址、日志的数量、哈希函数等等，并提供相应的方法进行操作。图文件管理器存放日志文件管理器的地址，所有图地址，图的数量，并提供相应的操作方法。整体结构图如图3-2所示：



图3-2 数据关联方式

## 3.2 主要算法设计

### 3.2.1 日志管理部分

日志管理器的属性：

初始大小、增量、日志数、空间大小、哈希表最大素数、日志地址集的基地址

日志管理器的方法：

（1）增加日志

算法思想：首先判断日志数量和日志地址集大小，如果日志数量大于等于日志地址集，那么重新分配空间，并对哈希表进行重排，重排的方法如下：首先产生一个接近于空间大小的素数，然后采用开放定址法排列元素，排列方式是线性探测再散列，哈希函数选择除留余数法。算法流程图如图3-3所示：



图3-3 增加日志算法流程图

（2）根据id获得日志地址

算法思想：首先根据哈希函数计算出日志的数组序号，如果序号所在的日志id和待查找的id一致，则查找成功，否则进行线性探测继续查找，当遇见了一个数组元素为NULL时，则代表查找失败，返回相应的状态码。程序流程类似于增加日志的后半部分，故不给出。

（3.1）根据id删除日志

算法思想：该部分类似于查找，查找相应的id所在的地址，然后释放空间并将原来的数组元素赋值为NULL。

（3.2）根据日志名称删除id

算法思想：因为logName不能够实现快速定址，所以只能使用顺序查找方法。遍历整个日志地址集，来查找待删除日志。流程图如图3-4所示：



图3-4 根据名称删除日志流程图

（3.3）根据来源删除日志

算法思想：将算法（3.2）的日志名称相等条件改为日志来源相等即可。

（3.4）根据进程删除日志

算法思想：将算法（3.2）的日志名称相等条件改为进程ID相等即可。

（4）查找日志

算法思想：类似于删除算法，当查找到日志后，直接返回日志地址即可。

（5）修改日志

算法思想：首先查找出相应的日志，然后对日志的各个属性进行赋值即可。

（6）文件读取和保存

算法思想：首先利用C++的运算符重载，将EventLog类型的运算符“>>”和“<<”重载为读取一个EventLog和输出一个EveneLog类型，然后读取日志的个数，循环相应的次数，每次循环中都生成一个日志并读取数据，然后调用AddLog函数保存日志文件。程序的流程图如图3-5所示：



图3-5 文件读取和保存

### 3.2.2 关联图管理部分

关联图管理器属性：

图的个数、空间大小、日志管理器、关联图地址集。

关联图管理器方法：

（1）增加图

算法思想：首先确定图指针数组是否满，如果是的话则重新分配空间。然后产生一个新图，在第graphCount的位置上插入。

（2）产生一个新图

算法思想：图的类型有四种，分别是主体相关，客体相关，伴随事件，因果相关，所以在这个函数下还要嵌套四个小函数。图的内部使用邻接矩阵储存相应的关系，并且使用一个map映射，使Log的ID和数组下表一一对应。生成图的具体步骤如下：首先传入一个Log的ID，判断出其类型的具体值（主体、客体等等），然后在数据集中搜索出类型一致的图，并将该图的顶点加入到图中，完毕后根据与传入的Log的关系进行加边。具体由如下流程图3-6所示：



图3-6 产生新图

（3）删除所有图

算法思想：从0-graphCount，以此释放相应的空间，然后置指针为NULL。

（4）查找图

算法思想：查找图需要知道两个参数，第一个是待查找图中包含的某个Log的ID，第二个是图的类型。从0-graphCount-1以此查找，然后判断该图是否包含ID的Log并且类型是传入的参数，如果是则返回图的地址，如果不是则返回NULL。

（5）作图

算法思想：作图需要传入图、画布、Log管理器、关联图的中心Log的ID。首先在（0,0）点位置做出中心点ID，然后在分成两种，一种是主体、客体、伴随事件，这三种事件都可以表示为一个星型图，另外一种因果相关可以表示为有向无环图。通过这两种特征，可以考虑分开作图。当中心做出后，前一种只需要围绕其做出其他相关联的图即可，后一种根据邻接矩阵，在中心点前面做出“因”，在中心点后面做出“果”即可。两种图的类型如图3-7、3-8所示：

图3-7 星型图 图3-8 有向无环图

（6）保存图为文件

算法思想：首先将EventGraph类实现<<和>>的运算符重载，然后从0-graphCount循环一遍，依次将其写入到输出流，成为文件即可。

（7）从存储介质中读取图

算法思想：利用重载好的运算符，直接一条条读取，每读取一条文件都分配空间，并调用管理图类的addGraph方法，直到遇见文件尾eof。

### 3.2.3 行为分析部分

（1）判断相似Log

算法思想：判断相似Log，采用当日志名称，任务类型，级别，关键字，来源，使用者这几个属性中至少有5个属性相同，那么就视为相似的方法。该函数出入一个Log管理器和待比较的Log，返回一个vector存放所有相似的Log。

（2）行为分析

算法思想：首先根据输入的信息产生一个新的Log，然后查找出相似Log的因果关联图，获得其前驱节点和后记节点。前驱节点作为行为推理，后继结点作为行为预测。

（3）生成新日志

算法思想：当用户想要将新发生的事件加入到日志集中，只需要额外传送Log的ID即可，系统调用Log管理器的addLog方法就将其加入到日志集中。

4 系统实现与测试

## 4.1 系统实现

### 4.1.1 软硬件环境

* WINDOWS 10 操作系统
* Qt 5.3版本
* Qt Creator编译器
* MinGW g++ 编译器

### 4.1.2 数据类型定义

（1）日志

class EventLog

{

//类的属性

private:

string \*logName; //日志名称

int sourceID;//日志来源

time\_t occurTime;//发生时间

int eventID;//事件ID

int taskType;//任务类别（客体）

int classType;//级别

string \*User;//发生事件的用户（主体）

int eventRecordID;//出现问题的进程ID

int keyWord;

string \*description;//事件的描述

public:

int tag=0;//标志域

}

1.日志的构造函数

EventLog(int ID)

EventLog(string logName,int sourceID,time\_t occurTime,

int eventID,int taskType,int classType,string User,

int eventRecordID,int keyWord,string description);

作用：分配内存空间，并将产生的事件进行赋值，其中ID不能为空。

2.日志的析构函数

~EventLog();

作用：释放由构造函数分配的空间

3.各个属性的getter和setter方法

int getID()、string getLogName()、bool setLogName(string name)、int getSourceID()、

bool setSourceID(int id)、time\_t getTime()、bool setTime(time\_t time)、int getTaskType()、bool setTaskType(int type)、int getClassType()、bool setClassType(int type)、string getUser()、bool setUser(string name)、int getEventRecordID()、bool setEventRecordID(int id)、int getKeyWord()、bool setKeyWord(int key)、string getDescription()、bool setDescription(string des)

作用：获得/设置日志的各个属性。

4.运算符<<和>>重载

friend std::ostream &operator<<(std::ostream & , EventLog &);

friend std::istream &operator>>(std::istream & , EventLog &);

作用：方便文件以流的方式读写。

（2）日志管理器

class manageLog

{

public:

static const int INIT\_SIZE = 1000; //初始大小

static const int INCREMENT = 0; //增量

int logNumber;//事件数

int size;//空间大小

int prime;//哈希表的最大素数

EventLog \*\*logs;//基地址

}

1.日志管理器的构造器

manageLog();

作用：分配初始空间。

2.日志管理器的析构函数

~manageLog();

作用：释放占用的空间

3.增加Log

bool AddLog(EventLog \*e);

作用：将新的Log指针添加到日志管理器的日志地址集，以哈希表的形式储存。

4.获得日志地址

EventLog \* getLog(int id);

作用：根据输入的id获得获得LOG在日志管理器中的日志，以指针形式返回，如果没有查找到，则返回空指针。

5.删除Log

bool DeleteLog(int eventID);

bool DeleteLog(string logName);

bool DeleteLogbySourceID(int sourceID);

bool DeleteLogbyRecordID(int eventRecordID);

作用：提供多种方式删除Log。

6.查找Log

vector<int> \* searchByID(int index);

vector<int> \* searchByTime(time\_t time);

vector<int> \* searchByTime(time\_t low,time\_t high);

vector<int> \* searchByUser(string User);

vector<int> \* searchByRecordID(int id);

vector<int> \* searchByTaskType(int task);

作用：根据输入的内容，查找出符合条件的Log集，并以vector的形式返回。

7.修改Log

bool modifyLog(int eventId,string logName,int sourceID,time\_t occurTime,int taskType,int classType,string User,int eventRecordID,intkeyWord,string description);

作用：根据传入的eventID查找到相应的日志，然后根据传入的参数进行修改。

8.存盘

bool SaveData ();

作用：将日志以文件的形式保存，因已经实现了>>运算符重载，则直接当做一个简单的变量保存即可，这也是C++的一大优势。

9.读取

bool LoadData ();

作用：从文件中读取日志，因为已经实现了<<运算符重载，则直接将文件当做输入流对待即可。

（3）关联图

class EventGraph

{

private:

int GraphContent;

int size;//邻接表大小

public:

static const int BASENUM = 10;

static const int INCREMENT = 5;

map<int,int> VHmap;//映射表，《index,value》

int \*\*adj;//邻接矩阵

int nodeCount;//顶点数

}

1.关联图的构造函数

EventGraph();

作用：分配内存空间，并将一些变量初始化

2.关联图的析构函数

~EventGraph();

作用：释放内存空间

3.增加节点

bool addNode(int value);

作用：向关联图中额外增加节点，同时将节点加入到映射表中。

4.增加边

bool addAdj(int tailValue,int headValue);

作用：给出弧头结点和弧尾节点，在邻接矩阵中增加一条有向边。

5.获取图的类型

int getContent();

6.设置图的类型

bool setContent(int type);

7.判断是否有顶点

bool hasNode(int value);

作用：根据传入的参数，判断关联图中是否存在ID为value的顶点。

8.获得ID

int getKeyByValue(int value);

作用：根据传入的邻接表编号获得对应的日志ID。

9.运算符重载

friend std::ostream &operator<<(std::ostream & , EventGraph &);

friend std::istream &operator>>(std::istream & , EventGraph &);

作用：重载<<和>>运算符用于文件的读和写。

（4）关联图管理器

class manageGraph

{

private:

manageLog \* ManageLog;//事件集操作

int graphCount;//图的个数

int size;

public:

static const int BASE = 10;

static const int INCRE = 10;

EventGraph \*\* Graph;

}

1.关联图管理器的构造函数

manageGraph(manageLog \* Manage);

2.关联图管理器的析构函数

~manageGraph();

3.增加关联图

bool addGraph(EventGraph \*g);

作用：将图的地址加入到关联图管理器的地址集。

4.生成关联图

EventGraph \* generateGraph(int index,int Type);

作用：根据中心事件的ID和将要生成的关联图类型，生成相应的关联图。该函数同时需要调用四个核心的生成关联图的函数，具体表示如图4-1所示：



图3-9 生成关联图函数调用依赖

5.获得关联图的数量

int GraphCount();

作用：返回该关联图管理器的关联图数量。

6.判断图存在

bool hasGraph(int index, int type);

作用：判断是否存在类型为type且含有ID为index日志的关联图。

7.删除所有关联图

void deleteAllGraph();

8.查找图

EventGraph \* findGraph(int index, int type);

作用：在关联图管理器中查找类型为type且含有ID为index日志的关联图，如果图不存在则返回空指针。

9.绘制关联图

void DrawGraph(QGraphicsScene \* scene, EventGraph \*G, int type);

void DrawGraphCE(QGraphicsScene \* scene, EventGraph \*G,int centralID);

作用：在Qt图形界面上绘制关联图。函数传入一个中心点ID和相应的关联图，然后再屏幕中央使用矩形、文字、直线等图形将关联图表现出来。绘制函数分为两种，一种是因果相关，一种是非因果相关，这两种关联图的类型不一样，具体类型参考之前的描述。调用方式如图4-2所示：



图4-2 绘制函数调用关系

10.文件读取

bool LoadGraph();

作用：从文件中以流的方式读取关联图数据，因实现了关联图的>>运算符重载，则只用即将其视为一个简单变量即可。

11.保存文件

bool SaveGraph();

作用：将关联图保存成文件，因实现了关联图的<<运算符重载，则只用将其视为一个简单变量即可。

（5）工具函数

1.判断素数

bool isPrime(int M);

作用：用于哈希表排列时判断素数。

2.N以内的素数

int MinPrime(int N);

作用：用于哈希表找中求出N以内的素数

3.值转换成Hash

int Value2Hash(int prime,int max,int ID);

作用：给出素数、最大空间、ID值，使用线性探测再散列的方式求解出ID的哈希值。

4.相似的Log

vector<EventLog \*> similarLog(manageLog \* ManageLog,EventLog log);

作用：求解出与所给日志相似的Log集，用vector返回。5.图的前面的节点

vector<EventLog \*> PriorNode(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog log);

作用：给出中心节点后，求出中心节点的前驱，用于行为分析推测原因。

6.图的后面的节点

vector<EventLog \*> NextNode(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog log);

作用：给出中心节点后，求出中心节点的后继，用于行为预测。

7.结果字符串

QString ResultStr(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog \* log);

作用：行为分析预测时用于返回结果的字符串。

**4.2 系统测试**

### 4.2.1 测试说明

测试系统分成三个主模块，分别是日志管理模块、关联图管理模块、行为预测和分析模块。日志管理模块主要测试的功能是日志的增、删、改、查和文件存取；关联图部分测试关联图的建立、绘制、文件存取；行为分析部分测试系统分析日志的能力。系统内预存108条来自于WINDOWS 10系统的系统日志，经过加工处理后成为系统的存储介质。启动程序的时候自动读入这108条数据，测试人员可以直接作为测试数据集。

### 4.2.2 测试数据及测试大纲

系统的测试数据如表4-1所示：

表4-1 测试数据









测试大纲可由图4-2所示：



图4-2 测试大纲

### 4.2.3 运行结果

首先打开系统主界面，系统自动将事先存好的数据，如图4-3所示，图中有三个主界面，分别代表三个主要功能：日志、关联图、行为分析。日志提供五种查询方式，分别按照日志ID、用户、进程ID、任务类型、时间方式查询。当查询出来日志后，双击对应的单元格即可进行修改，如果修改完毕后点击第三个按钮就能将修改的数据提交到系统，如果想要保存到数据文件中，则还需要再点击存盘按钮。

第二个按钮是添加日志，点击后系统会弹出来一个小窗口让你输入相应的数据，输入完毕后点击“OK”则添加成功。同样的，如果你想要把存到数据文件，则还必须点击存盘。



图4-3 主界面

功能一：查找日志

根据ID查找，如图4-4所示：



图4-4 根据ID查找

ID为1的日志是系统开机。

根据日志用户查找，输入“SYSTEM”后，系统查找所有使用者为SYSTEM的日志。同样，系统支持按照同一进程的日志查询，输入进程为32，并勾选按照日志的进程选项，点击查找后，即可出查询结果。两者的查询结果如图4-5、图4-6所示：



图4-5 按照日志的用户查询

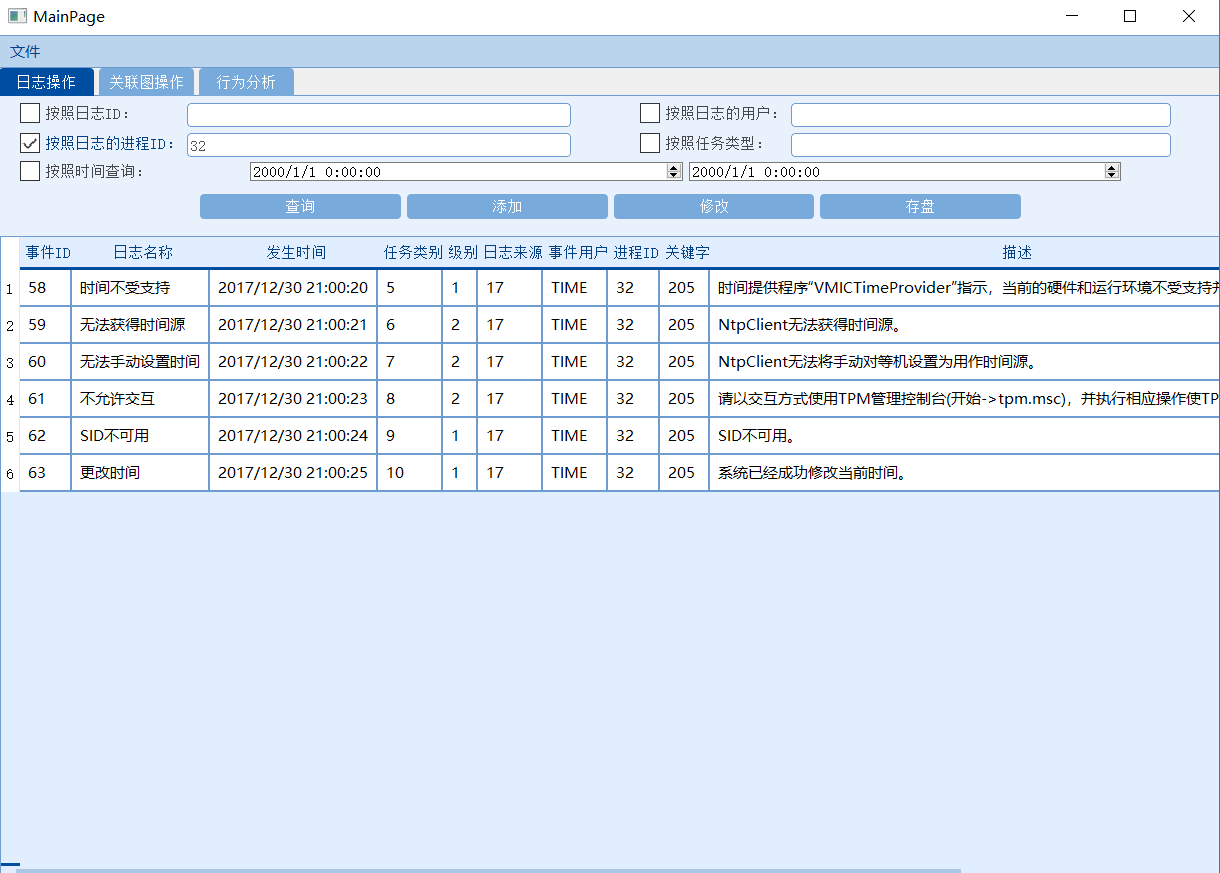


图4-6 按照日志的进程查询

功能二：修改日志

任意查找一个日志，比如ID为32的。双击对应的单元格就能修改里面的数据。比如双击“关键字”所在的那列，将其由255修改为888。操作结果如图4-7所示：



图4-7 修改日志

修改完毕后点击“修改按钮”，如图4-8所示：

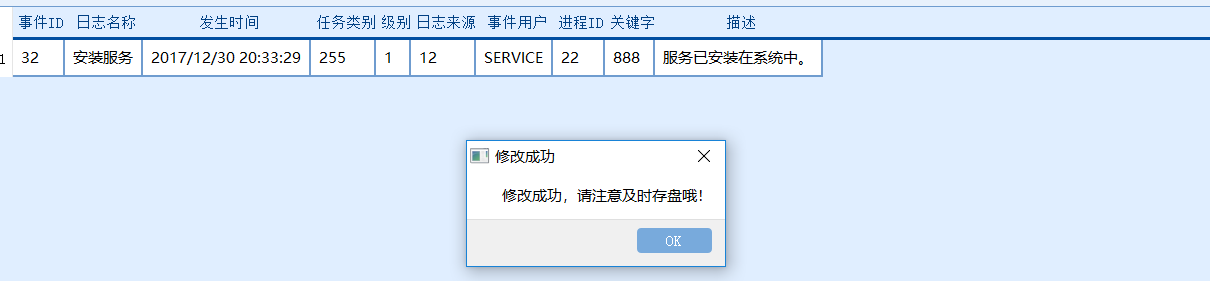


图4-8 修改日志

为了验证修改结果，我们重新点击“查询按钮”，查询出ID为32的日志的数据，结果如图4-9所示：



图4-9 重新查询日志

功能三：添加日志

点击“添加”按钮，弹出图4-10所示对话框，填入基本信息后点击“OK”。

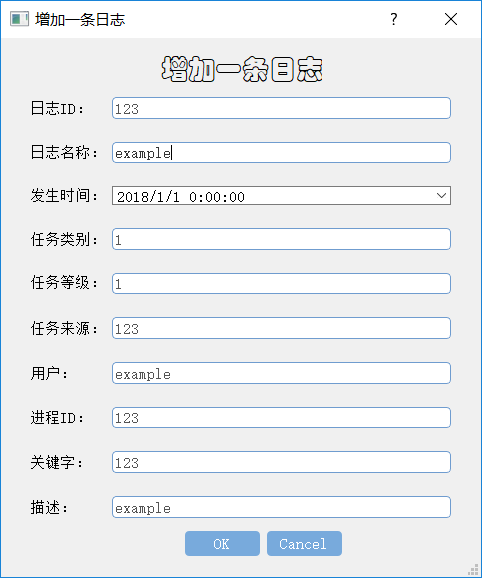


图4-10 增加日志

为验证添加程序的正确性，按照ID的方式查询日志，输入ID为“123”，点击“查询”按钮，结果如图4-11所示：

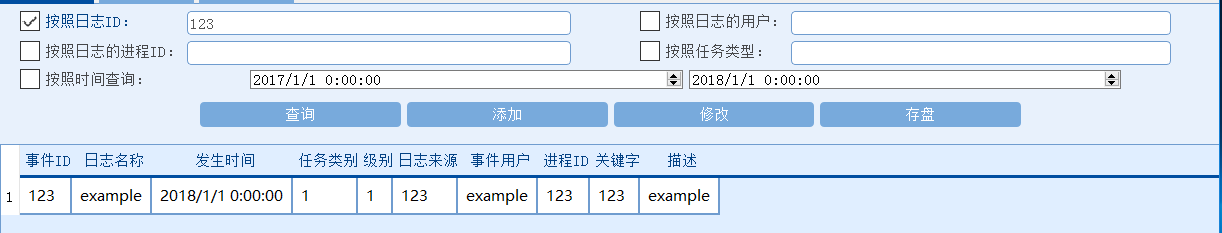


图4-11 验证添加日志功能

功能四：文件存储

此时，为了将该记录保存到存储介质上，还需要点击“存盘”按钮。存盘结果如图4-12所示：

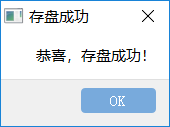


图4-12 日志存盘

功能五：删除日志

删除日志有三种方式：按照日志ID，按照进程ID，按照来源。按照题目的最基本要求，在这里我们只演示按照ID删除日志。点击菜单栏“文件”→“删除日志”→“按照日志ID”，弹出如图4-13所示输入框，在输入框中随便输入一个日志ID，比如输入“8”。

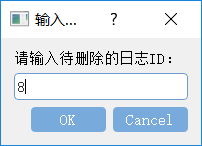


图4-13 删除日志

为了验证删除日志的正确性，在查询部分按照ID方式查询ID为8的日志，预期的结果是空，实际结果如图4-14所示：

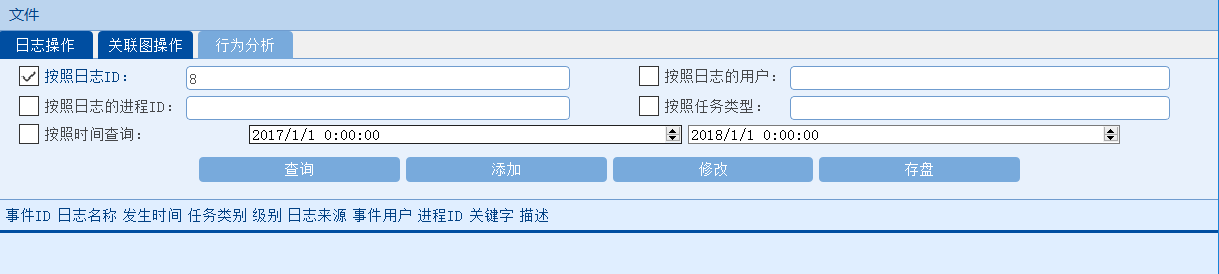


图4-14 验证删除日志

从结果上可以看到，日志ID为8的日志不存在，所以在表格中除了标题头外，什么也没有显示。

功能六：文件读取

首先保存当前文件，点击“存盘”。之前在添加日志的模块中添加了ID为“123”的日志，那么现在退出程序再重新进入。进入后按照ID的方式查询ID为“123”的日志，查询结果如图4-15所示：

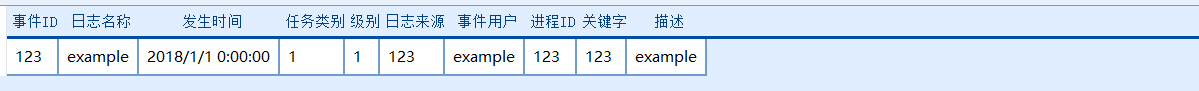


图4-15 验证文件保存和读取功能

日志操作模块展示完毕。

下面是关联图模块，在页面切换为“关联图操作”，如图4-16所示：

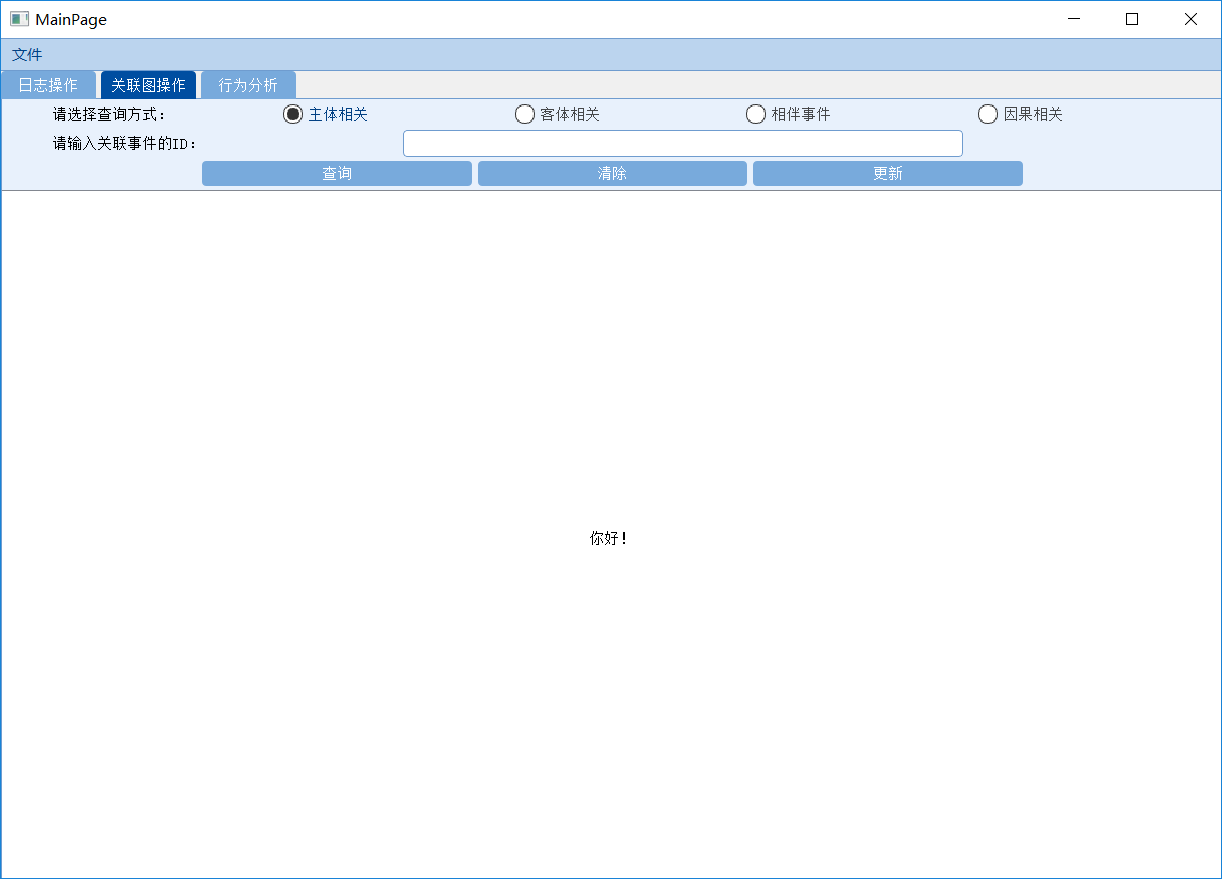


图4-16 关联图模块

在使用关联图模块时应当先点击“更新”按钮，因为随着日志部分的增加和删除，关联图也会改变，而关联图更新时，会重新搜索日志集并建立不同关联类型的日志关联图。输入某个日志的ID，系统会自动的调出已经搜索好的关联图并在下面的画板中绘制出来，根据前面的分析，前三种关联图是星型图，最后一种是有向无环图。点击“清除”按钮，画板会被置空。

功能七：关联图的建立

点击“更新”按钮即可建立。

功能八：关联图的展示

关联图总共有四种，分别是主体相关、客体相关、相伴事件、因果相关。主体相关意味着日志的使用者相同，客体相关意味着在一个时间领域内，日志的来源相同，相伴事件意味着首先在一个时间邻域内进程ID相同，因果相关则是在一个时间邻域内，在同一个进程中，时间早的会导致时间晚的发生。

展示主体相关，在日志ID输入框输入“1”，结果展示如图4-17所示：

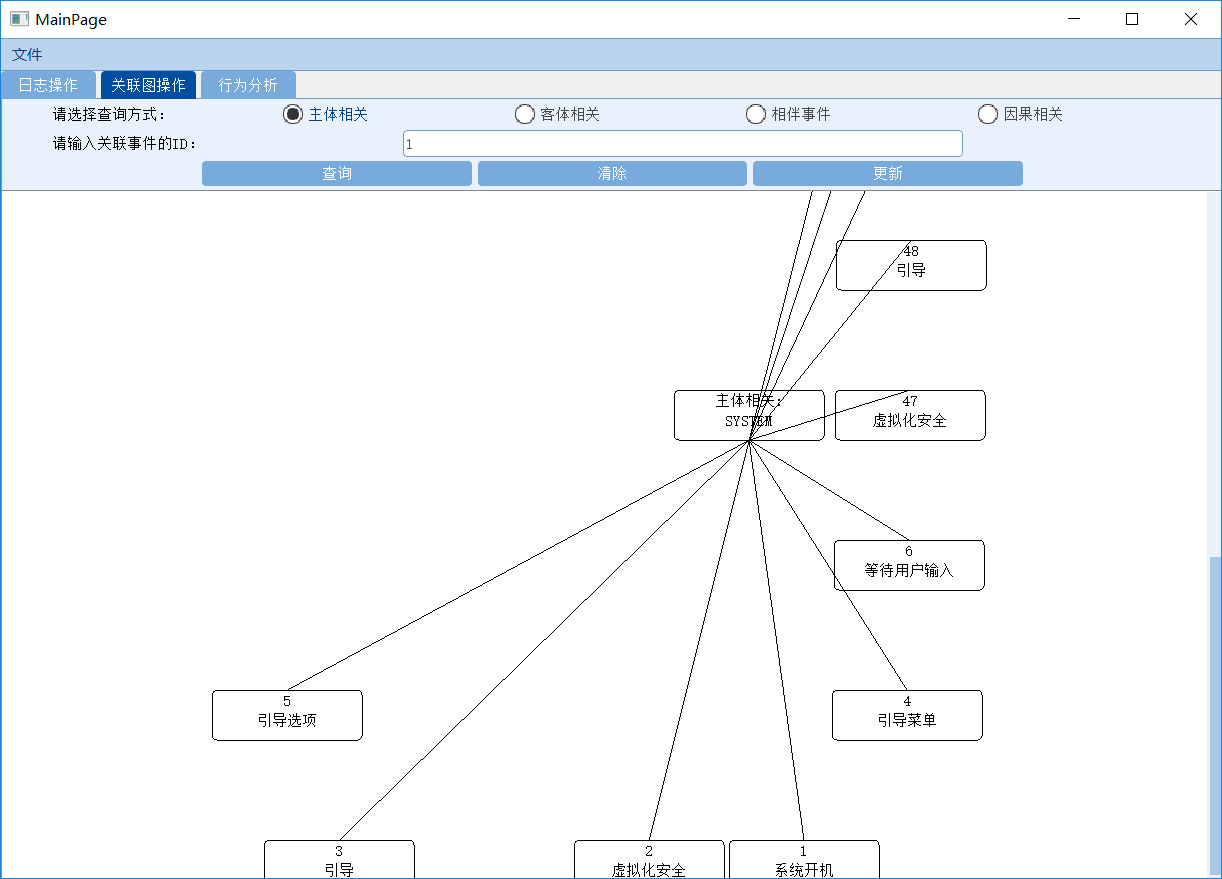


图4-17 主体相关关联图

可以看到，关联图的关联关系是以“SYSTEM”为主体的。

点击“客体相关”单选框，并输入ID“2”，点击“查询”，系统调出与日志ID为“2”的日志客体相关的关联图，结果如图4-18所示：

点击“相伴事件”单选框，并输入ID为“2”，点击查询，系统调出与日志ID为“2”的日志伴随事件的关联图，结果如图4-19所示：

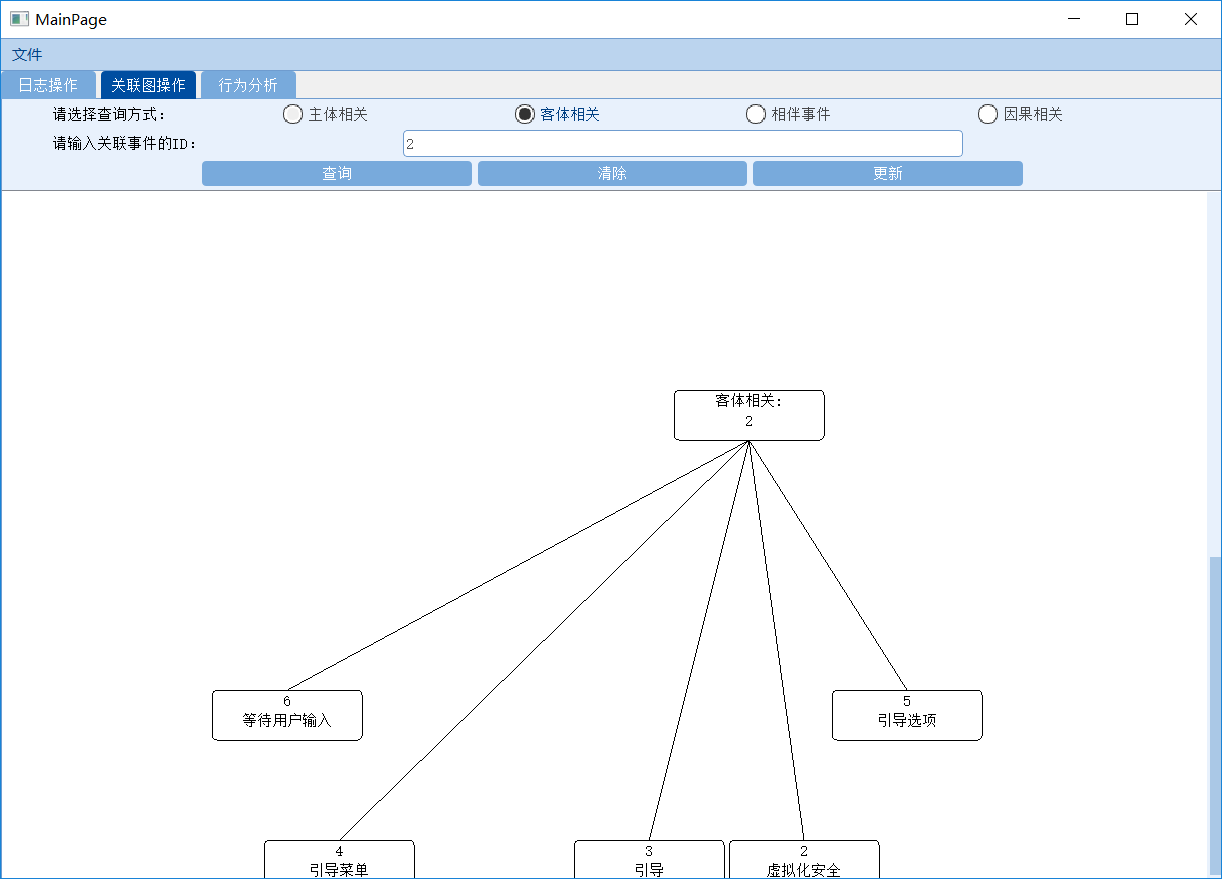


图4-18 客体相关关联图

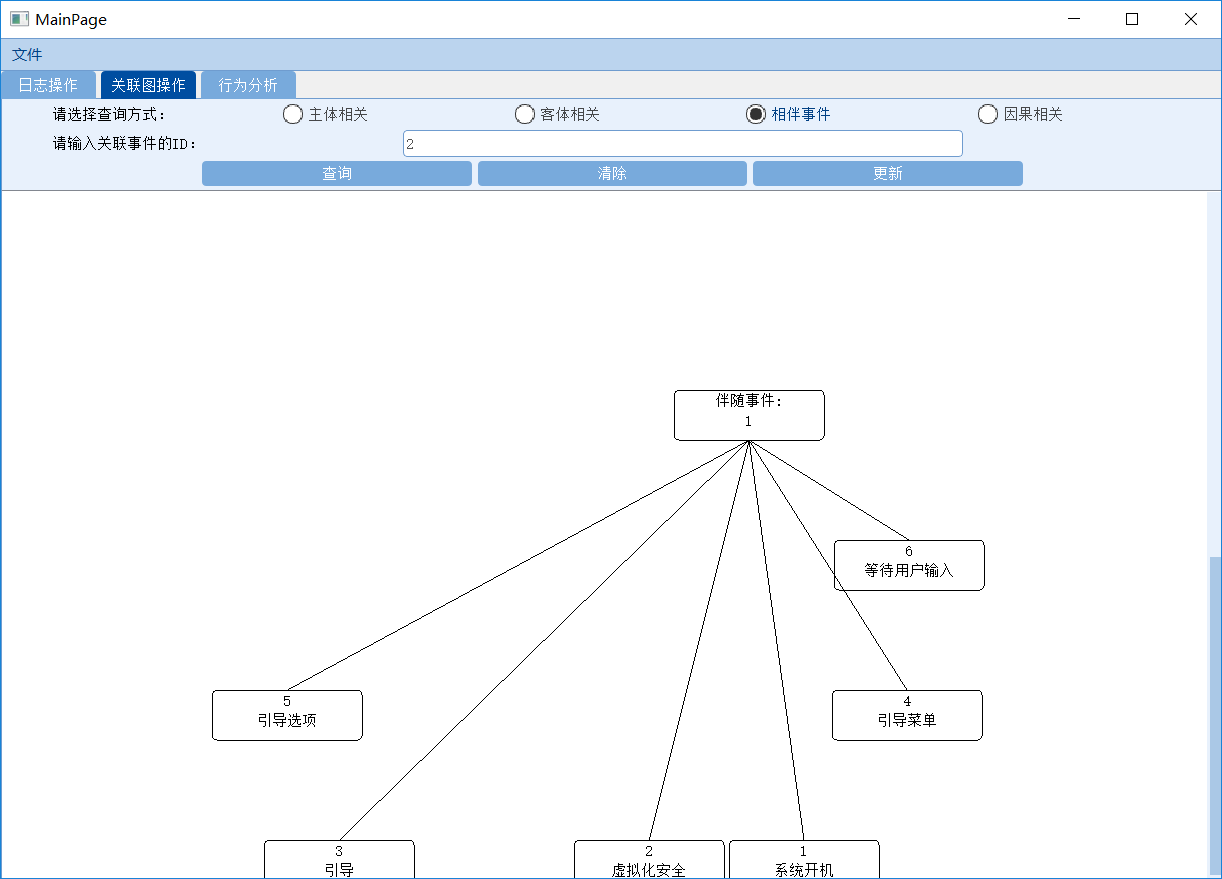


图4-19 伴随事件

因果相关的表示方法有些不同，同一个关联图，输入不同的中心日志ID形状可能不相同，系统会指明出本次事件的当前位置，左边代表因，右边代表果。关联图的操作结果如图4-20所示：

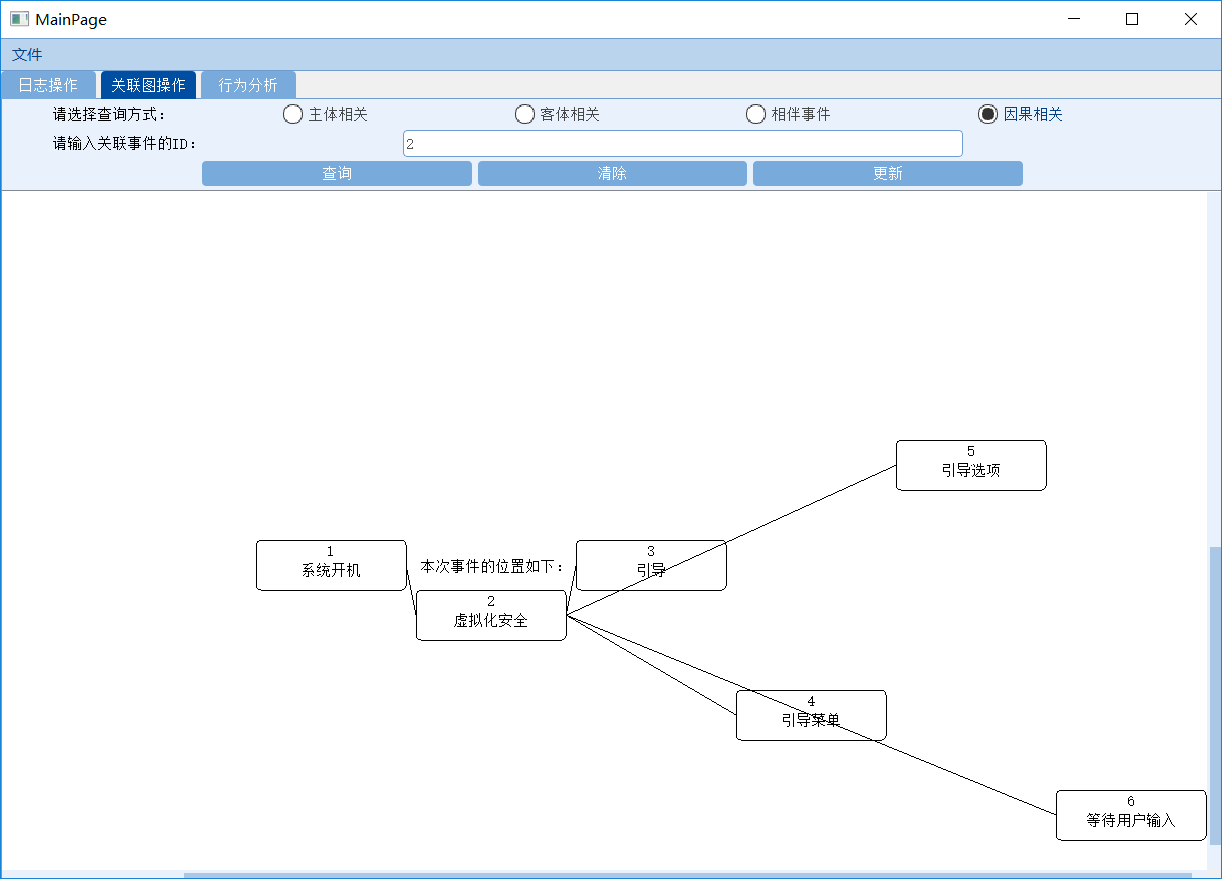


图4-20 因果相关关联图

当输入ID为4时，关联图还是同一个，但是表示方式则不同，操作结果如图4-21所示：

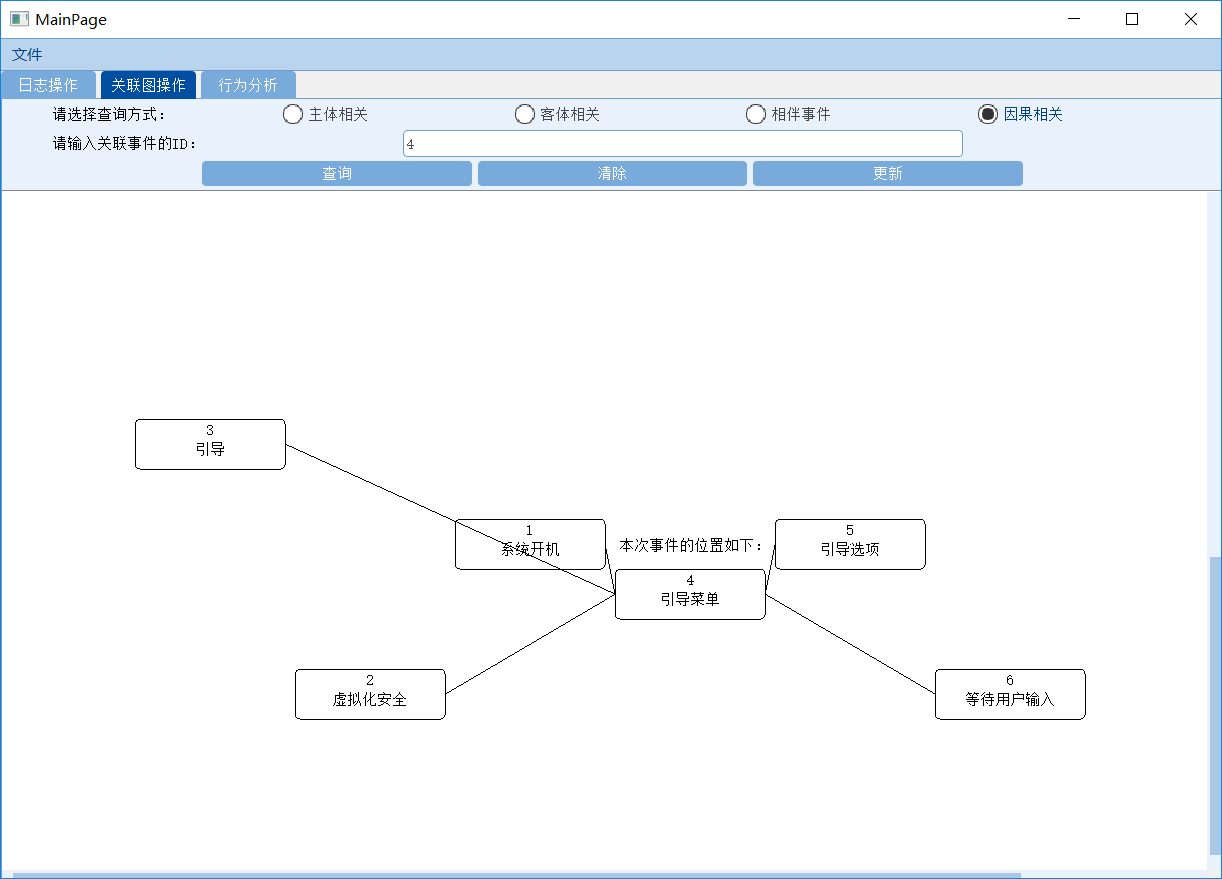


图4-21 以ID是4的日志作为中心事件

从上面看到，两者的关联图的形状是不相同的，但是事实上系统只存放了这一张图，只是跟据中心事件将关联图的形状进行调整。当将ID为“4”的日志作为中心事件时，ID为2的日志则成为了导致ID为4的日志产生的原因。系统指出ID为“4”的日志的位置，并将在其之前发生的日志绘制到左边，将其之后的位置会知道右边。

功能九：关联图保存与读取

当点击“更新”后，之前建立的ID为“123”的日志也会建立相应的关联图，但因为其所有属性都是新的，故其关联图的形状是孤立的一个节点。其关联图展示如图4-22所示：

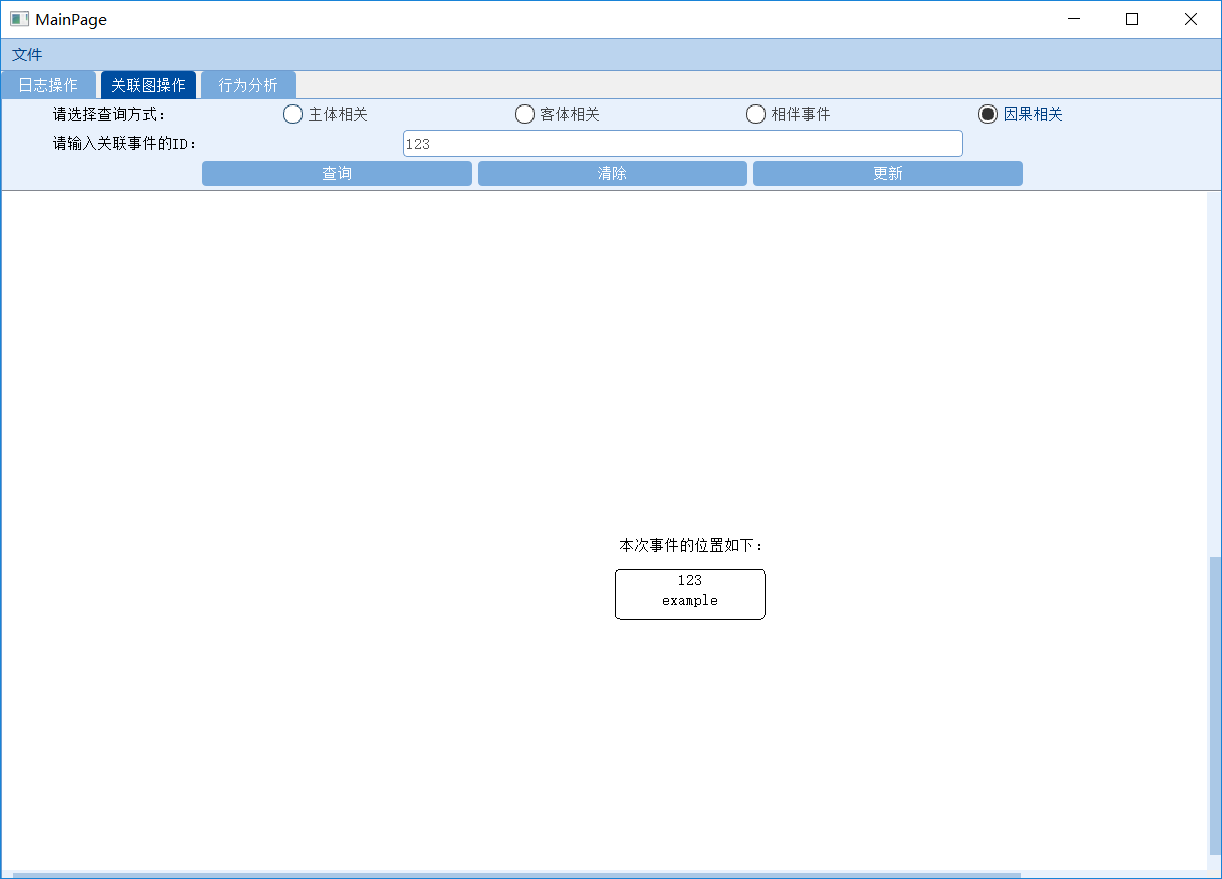


图4-22 建立的关联图

为了验证实现关联图能够不点击就可展示，我们首先点击“文件”→“关联图保存”，然后关闭系统重新进入。进入系统后，将页面切换到关联图操作，然后不点击“更新”按钮，直接输入ID“123”，展示结果如图4-23所示：

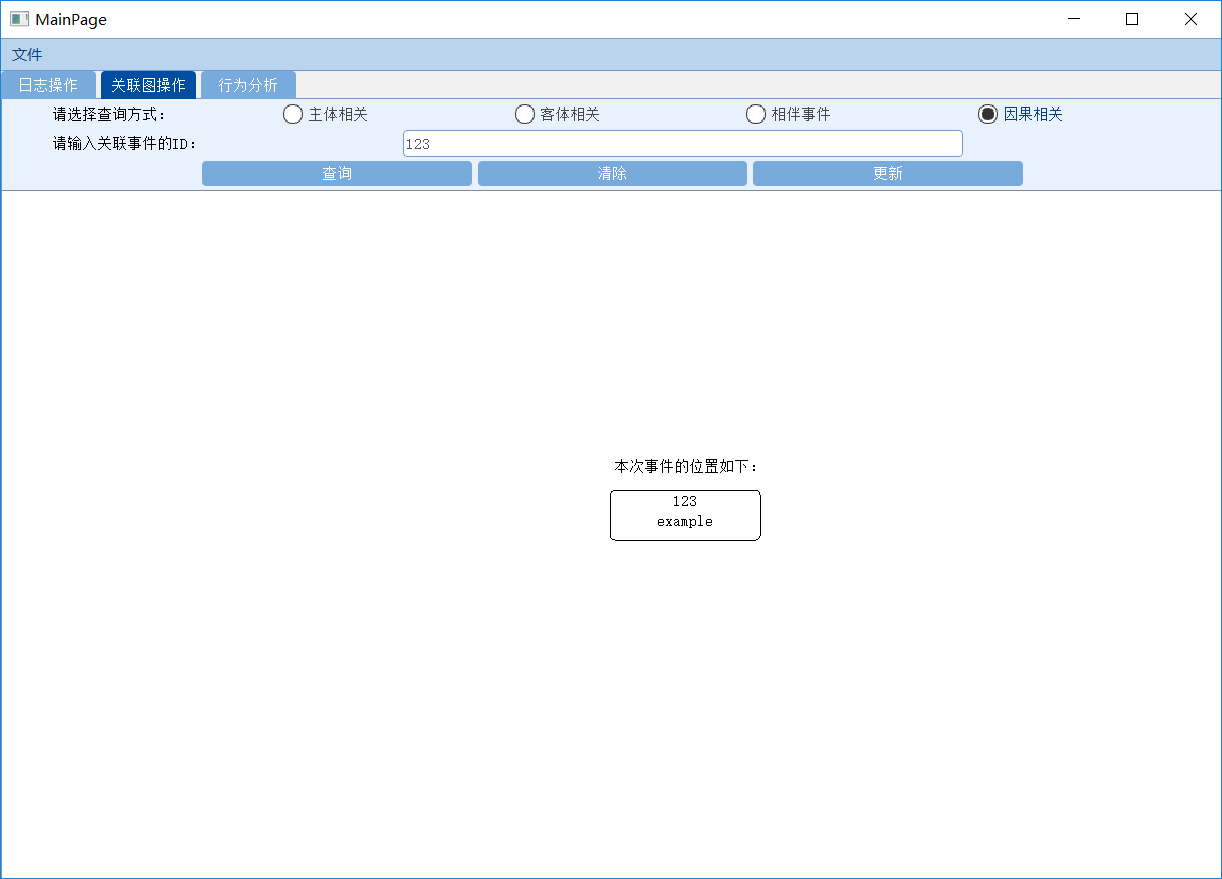


图4-23 验证关联图保存和读取

关联图模块展示完毕。

下面是行为分析模块。

功能十：行为分析

将页面切换到“行为分析”。根据标签上的说明，将一条新的日志信息填写到下面的表格中，系统将根据过往的产生的日志分析出该日志的行为。系统的行为分析是基于关联图模块的因果相关关联图，首先系统根据输入的信息匹配出与之相似的关联图，相似的关联图可以有一个、多个或者没有。当相似关联图没有时，系统返回出没有的提示，否则系统会在下面的空白处一一列出关联图的搜索结果，给出日志的产生原因和预测结果。

“清除”按钮则将输入的信息全部清空。“保存该日志”按钮可以让用户将这条新产生的日志保存到系统的日志集中。但是注意，保存日志后必须在日志操作模块处进行存盘，否则下次启动该日志还是没有的。

假设输入如图4-24所示的新日志的信息，并点击“分析”按钮。



图4-24 行为分析

功能十一：保存成日志

当向将该日志保存到系统中，则点击“保存该日志”按钮，弹出输入新ID的对话框，如图4-25所示：

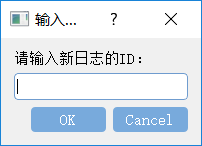


图4-25 输入新ID

输入“124”后，点击“OK”后，该条日志即保存在系统中，为了验证这一过程，我们在查询日志中查询这条日志，如图4-26所示：

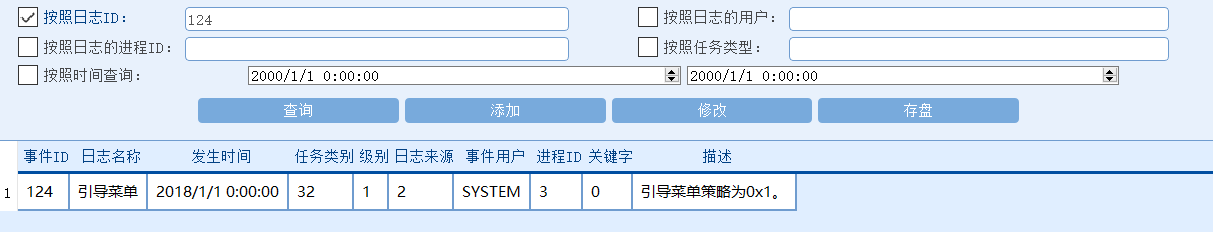


图4-26 验证保存新日志功能

从结果看，添加成功。

至此，系统所有功能全部展示完毕！

5 总结与展望

## 5.1 全文总结

本课程设计的主要工作如下：

（1）分析程序主要功能

拿到选题后，我先仔细阅读了选题的要求，但是对系统日志不是很了解，然后去网上查阅了相关的资料，对系统日志有个大概的了解。分别根据所给的要求列出系统的主要模块，此时系统的大致模块已经逐渐清晰，我只需要再读一遍题目，检查是否有功能上的缺失。

（2）设计界面UI

为了能够让程序的功能更完美的展示出来，我选择使用Qt图形库编写软件，经过短暂的Qt学习后，我便开始上手设计UI了，系统的UI很简单，就是将所要表达的数据展示出来即可，算是一种比较常规的方法了。其中比较新颖的是将关联图表示出来。即使不将关联图表示出来也可以完成题目的要求，但是表示出来会更加直接和美观。

（3）数据结构和类的设计

首先设计了日志和关联图的结构，用于表示最基本的数据类型，同时设计了日志管理器结构和图管理器，这样可以将这两者结合起来操作。为了方便操作，专门编写一个tool.cpp文件存放需要使用的工具函数。因为采用C++语言编写程序，那么可以利用语言中的面向对象特性，将结构直接以类的方式实现。设计之初只有四个类，但是后来绘制关联图的时候发现必须要继承原来的QGraphicItem类，因为只有这样才能够重写我需要实现的功能。

（4）编写函数

编写函数部分在第三步的时候已经有了函数声明，我只需要向其中填入具体的实现即可。部分函数只是一个外套，具体实现被封装起来了，有些函数比较简单，直接在声明的时候就实现了。

（5）部分优化

为了让系统运行流畅一些，我选择使用哈希表的方式存放日志，并让日志实现<<和>>运算符重载，降低程序的变成难度等等。

## 5.2工作展望

在今后的研究中，我会围绕着如下几个方面开展工作

（1）增加更多的日志文件的数据项，使之包含更多的信息。

（2）增加新的关联图类型，让程序不仅仅只表示这四种类型的关联图。

（3）改变部分数据的存储方式，将存储类型为“string”的数据项更改为存储int 类型，然后在内部建立一个映射表用于查找，这样可以方便的存储文件。

（4）优化绘制关联图部分。实际上现在的关联图部分有些简陋，还有许多的不足，所以以后应当重点优化这一部分。

（5）优化UI部分，使界面更美观。

6 体 会

经过这次课程设计，我觉得自己的代码能力更进一步。课程设计是写一个相对完整的软件，所以软件内部的各个细节都需要我们亲自完成，这提高了我的大局观，让我明白了自顶向下的程序设计方案。在UI设计环节浪费时间有点多，虽然界面都是非常常规的，但是对于Qt的IDE不是很熟悉，所以在这上面的学习成本有点多。日志类型和图类型的设计都是比较常规，日志管理器和图管理器的设计则给程序带来了方便，这不仅减少了程序的耦合性，还提高了UI界面和程序内部的交互性，使得这一切实现得自然。

日志存放我采用了哈希表的方式，本来是想提高程序的读取速度，但是却给我带来了很多麻烦，比如总是引用到空地址，函数条件处理不方便等等。不过经过这样的错误磨炼，日后再采用哈希表存放的时候，我会注意这样的问题。图管理器之所以采用空间复杂度较高的邻接矩阵，是因为实现方便，错误出现的少。事前我大概估算了一下关联图的节点个数最多20个，因为系统事件出现异常后会中断，记录错误的日志也不会很多，所以关联图算是小规模的，所以采用邻接矩阵非常方便，大大简化了编程的难度。

在函数实现方面，我采用了函数封装内部实现的方式。比如生成关联图的时候，因为图的类型总共有四种，所以在一个统一的函数外壳内部还包含了四个不同的生成函数。将函数封装起来体现出了程序的聚合性，降低了程序的耦合性，而且还可以分模块排查错误，是一种较好的技巧。

巧妙地利用编程语言的优势可以大大简化写程序的复杂度，这里我感触颇深的是两点，第一点是面向对象特性。对于工程实际来讲，大多数都采用了面向对象。面向对象特性能够使各个功能模块化，从整体局部，使程序的设计思路逐渐清晰起来，方便程序的编写与阅读。第二点是C++的运算符重载功能，在课程设计中我重载了<<和>>运算符，这使得文件读写更加简单，操作自定义的数据类型就像操作整数、字符等方式一样，直接将其视为一个变量即可。

程序美中不足的是数据量相对有点小。数据量小可能会导致程序隐藏的bug难以发现，或者程序中不合理的地方不能优化，这一点在以后的软件设计中应当要加强。

经过这次课程设计的锻炼，我在编程方面的技巧有所提高，俗话说，温故而知新可以为师矣，学习数据结构的知识后要多练，才能对其更加掌握。

参考文献

[1] 严蔚敏, 吴伟民. 数据结构（C语言版）. 北京: 清华大学出版社,1997

[2] 王晓东. 计算机算法设计与分析. 北京: 电子工业出版社, 2007

[3] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[4] 傅成兵. 系统日志技术研究与编程实现.计算机与信息技术，2009,( Z1):94~97

附录

## 一：程序源代码

### 1. eventlog.h:

#ifndef EVENTLOG\_H

#define EVENTLOG\_H

#include <string>

#include<ctime>

#include<ostream>

#include<istream>

using std::string;

namespace LOG {

class EventLog;

}

class EventLog

{

private:

string \*logName; //日志名称

int sourceID;//日志来源

time\_t occurTime;//发生时间

int eventID;//事件ID

int taskType;//任务类别（客体）

int classType;//级别

string \*User;//发生事件的用户（主体）

int eventRecordID;//出现问题的进程ID

int keyWord;

string \*description;//事件的描述

public:

int tag=0;//标志域

//事件构造器

EventLog(int ID);

EventLog(string logName,int sourceID,time\_t occurTime,

int eventID,int taskType,int classType,string User,

int eventRecordID,int keyWord,string description);

~EventLog();

/\*\*

\* @brief getID 获得事件ID

\* @return

\*/

int getID(){

if(this==NULL) return -1;

return this->eventID;

}

/\*\*

\* @brief getLogName 获得事件名称

\* @return

\*/

string getLogName(){

if(this->logName==NULL) return "";

return \*this->logName;

}

/\*\*

\* @brief setLogName 设置事件名称

\* @param name 事件名称字符串

\* @return

\*/

bool setLogName(string name){

if(this->logName!=NULL) delete this->logName;

this->logName = new string(name.data());

return true;

}

int getSourceID(){return this->sourceID;}

bool setSourceID(int id){

this->sourceID = id;

return true;

}

time\_t getTime(){return occurTime;}

bool setTime(time\_t time){

if(time >0){this->occurTime = time; return true;}

else return false;

}

int getTaskType(){return this->taskType;}

bool setTaskType(int type){

this->taskType = type;

return true;

}

int getClassType(){return this->classType;}

bool setClassType(int type){

this->classType = type;

return true;

}

string getUser(){

if(this->User==NULL) return "";

return \*this->User;

}

bool setUser(string name){

if(this->User!=NULL) delete this->User;

this->User = new string(name.data());

return true;

}

int getEventRecordID(){return this->eventRecordID;}//获得进程ID

bool setEventRecordID(int id){this->eventRecordID = id;return true;}

//关键字的设置

int getKeyWord(){return this->keyWord;}

bool setKeyWord(int key){this->keyWord = key;return true;}

//描述设置

string getDescription(){

if(this->description==NULL) return "";

return \*this->description;

}

bool setDescription(string des){

if(this->description!=NULL) delete this->description;

this->description = new string(des.data());

return true;

}

friend std::ostream &operator<<(std::ostream & , EventLog &);

friend std::istream &operator>>(std::istream & , EventLog &);

};

### 2. eventlog.cpp:

#include "eventlog.h"

#include <string>

#include<ostream>

#include<istream>

/\*\*

\* @brief EventLog::EventLog 构造函数

\* @param ID事件ID

\*/

EventLog::EventLog(int ID)

{

this->eventID = ID;

this->classType = 0;

this->eventRecordID = 0;

this->keyWord = 0;

this->logName = NULL;

this->sourceID = 0;

this->occurTime = 0;

this->taskType = 0;

this->User = NULL;

this->description = NULL;

this->tag = 0;

}

/\*\*

\* @brief 构造函数

\* @param logName 日志名称

\* @param sourceID 发生进程的ID

\* @param occurTime 发生时间

\* @param eventID 事件ID

\* @param taskType任务类型

\* @param classType任务等级

\* @param User 使用者

\* @param eventRecordID 事件记录ID

\* @param keyWord 关键字

\* @param description 描述

\*/

EventLog::EventLog(string logName,int sourceID,time\_t occurTime,

int eventID,int taskType,int classType,string User,

int eventRecordID,int keyWord,string description){

this->eventID = eventID;

this->classType = classType;

this->eventRecordID = eventRecordID;

this->keyWord = keyWord;

this->logName = new string(logName.data());

this->sourceID = sourceID;

this->occurTime = occurTime;

this->taskType = taskType;

this->User = new string(User.data());

this->description = new string(description.data());

this->tag = 0;

}

/\*\*

\* @brief EventLog::~EventLog 析构函数

\* 将所分配的空间都释放

\*/

EventLog::~EventLog(){

if(this->logName!=NULL) delete this->logName;

if(this->logName!=NULL) delete this->User;

if(this->logName!=NULL) delete this->description;

}

std::ostream & operator<<(std::ostream & out , EventLog & log ){

out << \*log.logName <<' '<< log.sourceID<<' '<<log.occurTime<<' '<<log.eventID<<' '<<

' '<<log.taskType<<' '<<log.classType<<' '<<\*log.User<<' '

<<log.eventRecordID<<' '<<log.keyWord<<' '<<\*log.description<<' ';

return out;

}

std::istream &operator>>(std::istream & input , EventLog &log){

string \* logname = new string();

string \* User = new string();

string \* Description = new string();

input >> \*logname>>log.sourceID>>log.occurTime>>log.eventID>>log.taskType

>>log.classType>>\*User>>log.eventRecordID>>

log.keyWord>>\*Description;

log.logName = logname;

log.User = User;

log.description = Description;

return input;

}

### 3. eventgraph.h:

#ifndef EVENTGRAPH\_H

#define EVENTGRAPH\_H

#include<map>

using std::map;

namespace LOG {

class EventGraph;

}

class EventGraph

{

private:

int GraphContent;

int size;//邻接表大小

public:

static const int BASENUM = 10;

static const int INCREMENT = 5;

map<int,int> Vhmap;//映射表，《index,value》

int \*\*adj;//邻接矩阵

int nodeCount;//顶点数

EventGraph();

~EventGraph();

bool addNode(int value); //增加节点

bool addAdj(int tailValue,int headValue);//增加边

int getContent();//获取关联图的类型

bool setContent(int type);//设置关联图类型

bool hasNode(int value);//是否存在点

int getKeyByValue(int value);//通过key获得Value

friend std::ostream &operator<<(std::ostream & , EventGraph &);

friend std::istream &operator>>(std::istream & , EventGraph &);

};

### 4. eventgraph.cpp:

#include “eventgraph.h”

#include<ostream>

#include<istream>

EventGraph::EventGraph()

{

this->nodeCount = 0;

this->adj = new int\*[this->BASENUM];//行

for(int i = 0;i<BASENUM;i++){

this->adj[i] = new int[BASENUM];//列

}

//邻接矩阵初始化为0

for(int i = 0;i<BASENUM;i++){

for(int j = 0;j<BASENUM;j++){

this->adj[i][j] = 0;

}

}

this->size = BASENUM;

}

EventGraph::~EventGraph(){

//释放空间

for(int i = 0;i<this->size;i++){

delete this->adj[i];

}

delete this->adj;

}

/\*\*

\* @brief EventGraph::addNode 增加图节点

\* @param value 值

\* @return

\*/

bool EventGraph::addNode(int value){

if(this->Vhmap.count(value)) return false;//节点已经存在

if(this->nodeCount>=this->size){

//产生新内存

int \*\* newadj = new int\*[this->size+this->INCREMENT];

for(int i = 0;i<this->size+this->INCREMENT;i++){

newadj[i] = new int[this->size+this->INCREMENT];

}

//新内存初始化

for(int i = 0;i<this->size+this->INCREMENT;i++){

for(int j = 0;j<this->size+this->INCREMENT;j++){

newadj[i][j] = 0;

}

}

//将原来的数据移到新内存中

for(int i = 0;i<this->size;i++){

for(int j = 0;j<this->size;j++){

newadj[i][j] = this->adj[i][j];

}

}

//释放原来的内存

for(int i = 0;i<this->size;i++){

delete this->adj[i];

}

delete this->adj;

//修改属性

this->adj = newadj;

this->size = this->size+this->INCREMENT;

}

this->Vhmap[value] = this->nodeCount++;

return true;

}

/\*\*

\* @brief EventGraph::addAdj 增加图的边

\* @param tailValue 弧尾节点的值

\* @param headValue 弧头结点的值

\* @return

\*/

bool EventGraph::addAdj(int tailValue,int headValue){

//判断传值是否合法

if(!this->Vhmap.count(tailValue)) return false;

if(!this->Vhmap.count(headValue)) return false;

int tail = Vhmap.at(tailValue);

int head = Vhmap.at(headValue);

//修改邻接矩阵的值

this->adj[tail][head] = 1;

return true;

}

/\*\*

\* @brief EventGraph::hasNode 图是否有某节点

\* @param value 节点的值

\* @return

\*/

bool EventGraph::hasNode(int value){

return Vhmap.count(value)==1;

}

/\*\*

\* @brief EventGraph::getKeyByValue 通过值获得顶点序号

\* @param value 顶点的值

\* @return

\*/

int EventGraph::getKeyByValue(int value){

for(map<int,int>::iterator iter = this->Vhmap.begin();iter!=this->Vhmap.end();iter++){

if(iter->second == value) return iter->first;

}

return -1;

}

/\*\*

\* @brief EventGraph::getContent 获得图的类型

\* @return

\*/

int EventGraph::getContent(){

return this->GraphContent;

}

/\*\*

\* @brief EventGraph::setContent 设置图的类型

\* @param type 类型

\* @return

\*/

bool EventGraph::setContent(int type){

this->GraphContent = type;

return true;

}

std::ostream &operator<<(std::ostream & out , EventGraph & G){

out<<G.size<<’ ‘<<G.nodeCount<<’ ‘<<G.GraphContent<<’ ‘;

for(map<int,int>::iterator iter = G.Vhmap.begin();iter!=G.Vhmap.end();iter++){

out<<iter->first<<’ ‘<<iter->second<<’ ‘;

}

for(int i = 0;i<G.nodeCount;i++){

for(int j = 0;j<G.nodeCount;j++){

out<<G.adj[i][j]<<’ ‘;

}

}

return out;

}

std::istream &operator>>(std::istream & input , EventGraph & G){

input >>G.size>>G.nodeCount>>G.GraphContent;

G.adj = new int\*[G.size];

for(int i = 0;i<G.size;i++){

G.adj[i] = new int[G.size];

for(int j = 0;j<G.size;j++){

G.adj[i][j] = 0;

}

}

for(int i = 0;i<G.nodeCount;i++){

int key;int value;

input >> key>>value;

G.Vhmap[key] = value;

}

for(int i = 0;i<G.nodeCount;i++){

for(int j = 0;j<G.nodeCount;j++){

input >> G.adj[i][j];

}

}

return input;

}

### 5. manageLog.h:

#ifndef MANAGELOG\_H

#define MANAGELOG\_H

#include”eventlog.h”

#include”tools/linkedlist.h”

#include<vector>

using std::vector;

namespace LOG {

class manageLog;

}

class manageLog

{

public:

static const int INIT\_SIZE = 1000; //初始大小

static const int INCREMENT = 0; //增量

int logNumber;//事件数

int size;//空间大小

int prime;//哈希表的最大素数

EventLog \*\*logs;//基地址

manageLog();

~manageLog();

//增加LOG

bool AddLog(EventLog \*e);//增加LOG

EventLog \* getLog(int id);//获得LOG

//删除Log

bool DeleteLog(int eventID);

bool DeleteLog(string logName);

bool DeleteLogbySourceID(int sourceID);

bool DeleteLogbyRecordID(int eventRecordID);

//查找Log

vector<int> \* searchByID(int index);

vector<int> \* searchByTime(time\_t time);

vector<int> \* searchByTime(time\_t low,time\_t high);

vector<int> \* searchByUser(string User);

vector<int> \* searchByRecordID(int id);

vector<int> \* searchByTaskType(int task);

//修改Log

bool modifyLog(int eventId,string logName,int sourceID,time\_t occurTime,

int taskType,int classType,string User,

int eventRecordID,int keyWord,string description);

//存盘和读取

bool LoadData();

bool SaveData();

};

### 6. manageLog.cpp:

#include "managelog.h"

#include "tools/tool.h"

#include <vector>

#include <string>

#include<fstream>

/\*\*

\* @brief manageLog::manageLog 管理图类的构造器

\*/

manageLog::manageLog(){

this->logNumber = 0;

this->size = INIT\_SIZE;

this->prime = MinPrime(this->size);

this->logs = (EventLog \*\* )malloc(sizeof(EventLog\*)\*INIT\_SIZE);

for(int i = 0;i<this->size;i++) logs[i] = NULL;

LoadData();

}

manageLog::~manageLog(){

for(int i = 0;i<this->size;i++){

if(logs[i]!=NULL){

delete logs[i];

}

}

free(this->logs);

}

/\*\*

\* @brief manageLog::AddLog 增加事件

\* @param e 事件指针e

\* @return 状态码

\*/

bool manageLog::AddLog(EventLog \*e){

//如果空间还有地址，那么对哈希表进行重排

//哈希表采用线性探测再散列方式

if(logNumber+1<size){

int index = e->getID();

index = index % prime;

while(logs[index]!=NULL){

if(logs[index]->getID()==e->getID()) return false;

index++;

index %= prime;

}

logs[index] = e;

logNumber++;

}

else{

//否则先重新分配内存，然后重排哈希表，再加入元素

EventLog \*\* oldlogs = this->logs;

EventLog \*\* newlogs;

//重新分配内存

newlogs = (EventLog \*\*)malloc((this->size+this->INCREMENT)\*sizeof(EventLog\*));

if(!newlogs) return false;

for(int i = 0;i<this->size+this->INCREMENT;i++) newlogs[i] = NULL;

this->logs = newlogs;

//产生最接近空间大小的素数

this->prime = MinPrime(this->size+this->INCREMENT);

//把Log从旧内存迁移到新内存

for(int i = 0;i<size;i++){

if(oldlogs[i]==NULL) continue;

int id = oldlogs[i]->getID();

id %= prime;

while(newlogs[id]){

if(newlogs[id]->getID()==e->getID()) return false;

id++;

id %= prime;

}

newlogs[id] = oldlogs[i];

}

free(oldlogs);//释放原有的空间

this->size = this->size+this->INCREMENT;//空间大小增加

//进行插入元素

int id = e->getID();

id %= prime;

while(newlogs[id]){

id++;

id = id % prime;

}

newlogs[id] = e;

logNumber++;

}

return true;

}

/\*\*

\* @brief manageLog::getLog 根据序号获得事件

\* @param index

\* @return

\*/

EventLog \* manageLog::getLog(int index){

//获得HASH值

int id = index%prime;

//定位位置

do{

if(this->logs[id]==NULL){

return NULL;

}

if(this->logs[id]->getID()==index){

return this->logs[id];

}

id++;

if(id >=this->size) id %= prime;

}while(true);

return NULL;

}

/\*\*

\* @brief manageLog::DeleteLog删除LOG

\* @param eventID 事件ID

\* @return 状态码

\*/

bool manageLog::DeleteLog(int eventID){

int id = eventID%prime;

vector<int> \*V;

V = searchByID(id);//查找id

//查找失败

if(V==NULL) return false;

id = V->back();

delete V;//释放内存

delete this->logs[id];

this->logs[id] = NULL;

logNumber--;

return true;

}

bool manageLog::DeleteLog(string logName){

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&

this->logs[i]->getLogName()==logName){

if(!DeleteLog(this->logs[i]->getID())){

return false;

}

else{

logNumber--;

}

}

}

return true;

}

/\*\*

\* @brief manageLog::DeleteLogbySourceID 根据源删除LOG

\* @param sourceID

\* @return

\*/

bool manageLog::DeleteLogbySourceID(int sourceID){

//循环查找

for(int i = 0;i<this->size;i++){

if(this->logs[i]==NULL) continue;

if(this->logs[i]->getSourceID()==sourceID){

if(!DeleteLog(logs[i]->getID())){

return false;

}

else{

logNumber--;

}

}

}

return true;

}

bool manageLog::DeleteLogbyRecordID(int eventRecordID){

//循环查找并删除

for(int i = 0;i<this->size;i++){

if(this->logs[i]==NULL) continue;

if(this->logs[i]->getEventRecordID()==eventRecordID){

if(!DeleteLog(logs[i]->getID())){

return false;

}

else{

logNumber--;

}

}

}

return true;

}

//根据ID查找LOG

vector<int> \* manageLog::searchByID(int index){

vector<int> \* V = new vector<int>();

int id = index%prime;

do{

if(this->logs[id]==NULL){

delete V;

return NULL;

}

if(this->logs[id]->getID()==index){

V->push\_back(id);

return V;

}

id++;

id %= prime;

}while(true);

return NULL;

}

/\*\*

\* @brief 通过时间查询

\* @param 输入时间点

\* @return 返回一个log的队列

\*/

vector<int> \* manageLog::searchByTime(time\_t time){

vector<int> \* V = new vector<int>();

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&this->logs[i]->getTime()==time){//满足时间

V->push\_back(i);

}

}

return V;

}

/\*\*

\* @brief 通过时间查询

\* @param low 时间下界

\* @param high上界

\* @return 消息队列

\*/

vector<int> \* manageLog::searchByTime(time\_t low,time\_t high){

vector<int> \* Q = new vector<int>();

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&this->logs[i]->getTime()>=low

&&this->logs[i]->getTime()<=high){

Q->push\_back(i);

}

}

return Q;

}

/\*\*

\* @brief 通过使用者查询

\* @param User 使用者字符串

\* @return 消息队列

\*/

vector<int> \* manageLog::searchByUser(string User){

vector<int> \* Q = new vector<int>();

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&this->logs[i]->getUser()==User){

Q->push\_back(i);

}

}

return Q;

}

vector<int> \* manageLog::searchByRecordID(int id){

vector<int> \* Q = new vector<int>();

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&this->logs[i]->getEventRecordID()==id){

Q->push\_back(i);

}

}

return Q;

}

vector<int> \* manageLog::searchByTaskType(int task){

vector<int> \* Q = new vector<int>();

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL&&this->logs[i]->getTaskType()==task){

Q->push\_back(i);

}

}

return Q;

}

/\*\*

\* @brief manageLog::modifyLog 修改LOG

\* @param eventId

\* @param logName

\* @param sourceID

\* @param occurTime

\* @param taskType

\* @param classType

\* @param User

\* @param eventRecordID

\* @param keyWord

\* @param description

\* @return

\*/

bool manageLog::modifyLog(int eventId,string logName,int sourceID,time\_t occurTime,

int taskType,int classType,string User,

int eventRecordID,int keyWord,string description){

//调用相应的setter方法就可以

vector<int> \* result;

result = this->searchByID(eventId);

int id = result->back();delete result;

this->logs[id]->setLogName(logName);

this->logs[id]->setSourceID(sourceID);

this->logs[id]->setTime(occurTime);

this->logs[id]->setTaskType(taskType);

this->logs[id]->setClassType(classType);

this->logs[id]->setUser(User);

this->logs[id]->setEventRecordID(eventRecordID);

this->logs[id]->setKeyWord(keyWord);

this->logs[id]->setDescription(description);

return true;

}

/\*\*

\* @brief manageLog::LoadData 加载日志

\* @return

\*/

bool manageLog::LoadData(){

std::ifstream infile("EventLog");//打开文件

if(!infile.is\_open()) return false;

EventLog \* newlog;

do{

newlog = new EventLog(1);//产生日志

infile >> \*newlog; //输入（>>运算符已经重载）

//遇见文件尾则退出循环

if(infile.eof()) {

delete newlog;

break;

}

else{

this->AddLog(newlog);//加入

}

}while(true);

infile.close();

return true;

}

/\*\*

\* @brief manageLog::SaveData 保存日志

\* @return

\*/

bool manageLog::SaveData(){

std::ofstream outfile("EventLog");//打开文件

//循环输入

for(int i = 0;i<this->size;i++){

if(this->logs[i]!=NULL){

outfile << \*logs[i];//（<<运算符已重载）

}

}

outfile.close();//关闭文件

return true;

}

### 7. managegraph.h:

#include”src/managelog.h”

#include”src/eventgraph.h”

#include”tools/gnode.h”

namespace LOG {

class manageGraph;

enum GraphType{Subject,Object,AccomEvent,CauseAndEffect};

}

class manageGraph

{

private:

manageLog \* ManageLog;//事件集操作

int graphCount;//图的个数

int size;

public:

static const int BASE = 10;

static const int INCRE = 10;

EventGraph \*\* Graph;//主体相关图

// vector<Gnode \*> v;//用作关联图的删除

manageGraph(manageLog \* Manage);

~manageGraph();

bool addGraph(EventGraph \*g);//增加图

EventGraph \* generateGraph(int index,int Type);//从事件记录中生成图

int GraphCount();

bool hasGraph(int index, int type);//判断某张图是否存在

void deleteAllGraph();//删除所有图

EventGraph \* findGraph(int index, int type);//查找图

void DrawGraph(QgraphicsScene \* scene, EventGraph \*G, int type);

void DrawGraphCE(QgraphicsScene \* scene, EventGraph \*G,int centralID);//画因果关系图

bool LoadGraph();

bool SaveGraph();

};

### 8. managegraph.cpp:

#include “managegraph.h”

#include<QgraphicsScene>

#include<fstream>

#include<QmessageBox>

/\*\*

\* @brief manageGraph::manageGraph 管理图类的构造器

\* @param Manage 管理LOG的地址

\*/

manageGraph::manageGraph(manageLog \* Manage)

{

this->size = this->BASE;

this->Graph = new EventGraph\*[this->size];//分配空间

for(int i = 0;i<size;i++){

this->Graph[i] = NULL;

}

this->graphCount = 0;

this->ManageLog = Manage;

this->LoadGraph();

}

manageGraph::~manageGraph(){

for(int i = 0;i<this->graphCount;i++){

delete this->Graph[i];

}

delete Graph;

}

/\*\*

\* @brief manageGraph::addGraph 增加图

\* @param g 图指针

\* @return 正确

\*/

bool manageGraph::addGraph(EventGraph \*g){

if((this->graphCount+2)==this->size){//空间是否满，重新分配空间

EventGraph \*\* newgra = new EventGraph\*[this->size+this->INCRE];

for(int i = 0;i<graphCount;i++){

newgra[i] = this->Graph[i];

}

delete this->Graph; //释放原有空间

this->Graph = newgra;

this->size = this->size+this->INCRE;

}

//插入图

this->Graph[this->graphCount] = g;

this->graphCount++;//图的数量增加

return true;

}

/\*\*

\* @brief manageGraph::GraphCount

\* 获得图的数量

\* @return

\*/

int manageGraph::GraphCount(){

return this->graphCount;

}

/\*\*

\* @brief manageGraph::hasGraph 是否存在图

\* @param index 节点的值

\* @param type 图的类型

\* @return

\*/

bool manageGraph::hasGraph(int index, int type){

//遍历图，寻找符合要求的图

for(int i = 0;i<this->graphCount;i++){

if(this->Graph[i]==NULL) continue;

if(this->Graph[i]->getContent()==type&&

this->Graph[i]->hasNode(index)) return true;

}

return false;

}

void SubCreate(manageLog \* ManageLog, EventGraph & Graph,int index);//使用者相同

void ObjCreate(manageLog \* ManageLog, EventGraph & Graph,int index);//来源相同相同

void AccumCreate(manageLog \* ManageLog, EventGraph & Graph,int index);//任务类型相同

void CauseAndEffectCreate(manageLog \* ManageLog, EventGraph & Graph,int index);//同一进程下的事件

/\*\*

\* @brief manageGraph::generateGraph 根据序号和类型生成相应的图

\* @param index 序号

\* @param Type 类型

\* @return

\*/

EventGraph \* manageGraph::generateGraph(int index, int Type){

EventGraph \* gra = new EventGraph();//生成一个空图

//根据要求生成相应的图

switch(Type){

case LOG::GraphType::Subject:

SubCreate(this->ManageLog,\*gra,index);

gra->setContent(LOG::GraphType::Subject);

break;

case LOG::GraphType::Object:

ObjCreate(this->ManageLog,\*gra,index);

gra->setContent(LOG::GraphType::Object);

break;

case LOG::GraphType::AccomEvent:

AccumCreate(this->ManageLog,\*gra,index);

gra->setContent(LOG::GraphType::AccomEvent);

break;

case LOG::GraphType::CauseAndEffect:

CauseAndEffectCreate(this->ManageLog, \*gra,index);

gra->setContent(LOG::GraphType::CauseAndEffect);

break;

default:

delete gra;

return NULL;

}

return gra;

}

/\*\*

\* @brief SubCreate 生成主体相关图

\* @param ManageLog LOG管理器

\* @param Graph 图引用

\* @param index 序号值

\*/

void SubCreate(manageLog \* ManageLog, EventGraph & Graph,int index){

Graph.addNode(index);//增加顶点

for(int i = 0;i<ManageLog->size;i++){

if(ManageLog->logs[i]==NULL) continue;

if(ManageLog->logs[i]->getID()==index) continue;

//若两个事件的使用者相同，并且名称不同，则该接待加入到图中

if(ManageLog->logs[i]->getUser().compare(ManageLog->getLog(index)->getUser())==0

&&ManageLog->logs[i]->getLogName().compare(ManageLog->getLog(index)->getLogName())!=0){

Graph.addNode(ManageLog->logs[i]->getID());

}

}

}

/\*\*

\* @brief ObjCreate 生成客体相关图

\* @param ManageLog LOG管理器

\* @param Graph 图引用

\* @param index 序号值

\*/

void ObjCreate(manageLog \* ManageLog, EventGraph & Graph,int index){

Graph.addNode(index);

for(int i = 0;i<ManageLog->size;i++){

if(ManageLog->logs[i]==NULL) continue;

if(ManageLog->logs[i]->getID()==index) continue;

int time1 = ManageLog->logs[i]->getTime();int time2 = ManageLog->getLog(index)->getTime();

if(ManageLog->logs[i]->getSourceID()==ManageLog->getLog(index)->getSourceID()

&&(time1-time2)\*(time1-time2)<100&&ManageLog->logs[i]->getLogName().compare(ManageLog->getLog(index)->getLogName())!=0){

Graph.addNode(ManageLog->logs[i]->getID());

}

}

}

/\*\*

\* @brief AccumCreate 生成伴随事件图

\* @param ManageLog LOG管理器

\* @param Graph 图引用

\* @param index 序号值

\*/

void AccumCreate(manageLog \* ManageLog, EventGraph & Graph,int index){

Graph.addNode(index);

for(int i = 0;i<ManageLog->size;i++){

if(ManageLog->logs[i]==NULL) continue;

if(ManageLog->logs[i]->getID()==index||ManageLog->getLog(i)->getLogName().compare(ManageLog->getLog(index)->getLogName())==0) continue;

int time1 = ManageLog->logs[i]->getTime();int time2 = ManageLog->getLog(index)->getTime();

//进程ID相同，且时间相差在10s以内，则增加该顶点

if(ManageLog->logs[i]->getEventRecordID()==ManageLog->getLog(index)->getEventRecordID()

&&(time1-time2)\*(time1-time2)<100 ){

Graph.addNode(ManageLog->logs[i]->getID());

}

}

}

/\*\*

\* @brief CauseAndEffectCreate 生成因果相关图

\* @param ManageLog LOG管理器

\* @param Graph 图引用

\* @param index 序号值

\*/

void CauseAndEffectCreate(manageLog \* ManageLog, EventGraph & Graph,int index){

Graph.addNode(index);

vector<int> nodes;

nodes.push\_back(index);

for(int i = 0;i<ManageLog->size;i++){

if(ManageLog->logs[i]==NULL) continue;

if(ManageLog->logs[i]->getID()==index) continue;

//事件进程相同，且时间相差10s则增加定点

int time1 = ManageLog->logs[i]->getTime();int time2 = ManageLog->getLog(index)->getTime();

if(ManageLog->logs[i]->getEventRecordID()==ManageLog->getLog(index)->getEventRecordID()&&

(time1-time2)\*(time1-time2)<500){

Graph.addNode(ManageLog->logs[i]->getID());

nodes.push\_back(ManageLog->logs[i]->getID());

/\*

if(time1>time2){

Graph.addAdj(ManageLog->logs[index]->getTime(),ManageLog->logs[i]->getID());

}

else{

Graph.addAdj(ManageLog->logs[i]->getID(),ManageLog->logs[index]->getTime());

}

\*/

}

}

//加边，根据时间点，分成本事件之前和本事件之后

for(vector<int>::iterator iter\_out = nodes.begin();iter\_out!=nodes.end();iter\_out++){

for(vector<int>::iterator iter\_in = nodes.begin();iter\_in!=nodes.end();iter\_in++){

if(\*iter\_out==(\*iter\_in)) continue;

//根据时间分类

if(ManageLog->getLog(\*iter\_out)->getTime()>ManageLog->getLog(\*iter\_in)->getTime()){

Graph.addAdj(\*iter\_in,\*iter\_out);

}

else{

Graph.addAdj(\*iter\_out,\*iter\_in);

}

}

}

}

/\*\*

\* @brief manageGraph::deleteAllGraph

\* 删除所有的图

\*/

void manageGraph::deleteAllGraph(){

for(int i = 0;i<this->GraphCount();i++){

delete this->Graph[i];

this->Graph[i] = NULL;

}

this->graphCount = 0;

}

/\*\*

\* @brief manageGraph::findGraph 查找图

\* @param index 图的id

\* @param type 图的类型

\* @return 图的地址

\*/

EventGraph \* manageGraph::findGraph(int index, int type){

EventGraph \* G = NULL;

for(int i = 0;i<this->graphCount;i++){

if(this->Graph[i]==NULL) continue;

//当满足类型相同，并且顶点相同，图找到

if(this->Graph[i]->getContent()==type&&

this->Graph[i]->hasNode(index)){

G = this->Graph[i];

break;

}

}

return G;

}

/\*\*

\* @brief manageGraph::DrawGraph 将图以二维图片表示出来（非核心逻辑代码，注释省略）

\* @param scene 场景

\* @param G 图

\* @param type

\*/

void manageGraph::DrawGraph(QgraphicsScene \* scene, EventGraph \* G,int type){

int count = 0;int num = -100;

int ypos = 250;

map<int,int>::iterator iter = G->Vhmap.begin();

Qstring titleTxt;

Gnode \* title;

switch (type){

case LOG::GraphType::Subject:

titleTxt = Qstring::fromStdString(this->ManageLog->getLog(iter->first)->getUser());

title = new Gnode(“主体相关:”,titleTxt);

title->setPos(-155,-200);

scene->addItem(title);

break;

case LOG::GraphType::Object:

titleTxt = Qstring::number(this->ManageLog->getLog(iter->first)->getSourceID());

title = new Gnode(“客体相关:”,titleTxt);

title->setPos(-155,-200);

scene->addItem(title);

break;

case LOG::GraphType::AccomEvent:

titleTxt = Qstring::number(this->ManageLog->getLog(iter->first)->getTaskType());

title = new Gnode(“伴随事件:”,titleTxt);

title->setPos(-155,-200);

scene->addItem(title);

break;

}

if(type!=LOG::GraphType::CauseAndEffect){

for(map<int,int>::iterator i = G->Vhmap.begin();i!= G->Vhmap.end();i++,count++){

Gnode \*gnode = new Gnode(Qstring::number(i->first),

Qstring::fromStdString(this->ManageLog->getLog(i->first)->getLogName()));

num = num+(-1)\*count\*155;

if(!(num<650&&num>-650)){

ypos -=150;

num = count;

}

gnode->setPos(num,ypos);

scene->addItem(gnode);

scene->addLine(-155,-175,num,ypos-25);

}

scene->update();

}

else

DrawGraphCE(scene,G,iter->first);

}

void propDraw(manageLog \* ManageLog, QgraphicsScene \* scene, EventGraph \*G,vector<int> \*review, int centralID,int X,int Y);

void backDraw(manageLog \* ManageLog, QgraphicsScene \* scene, EventGraph \*G,vector<int> \*review, int centralID,int X,int Y);

/\*\*

\* @brief manageGraph::DrawGraphCE 将因果关系图画成二维图片，非核心逻辑代码，注释省略

\* @param scene

\* @param G

\* @param centralID

\*/

void manageGraph::DrawGraphCE(QgraphicsScene \* scene, EventGraph \*G,int centralID){

QgraphicsTextItem \* txttile = new QgraphicsTextItem(“本次事件的位置如下：”);

txttile->setPos(-75,-60);

scene->addItem(txttile);

Gnode \* gnode = new Gnode(Qstring::number(centralID),Qstring::fromStdString(this->ManageLog->getLog(centralID)->getLogName()));

gnode->setPos(0,0);

scene->addItem(gnode);

vector<int> \*review = new vector<int>();

review->push\_back(G->Vhmap[centralID]);

//向后传播

int count = 0;

for(int i = 0;i<G->nodeCount;i++){

if(G->Vhmap[centralID]==i) continue;

if(G->adj[G->Vhmap[centralID]][i]!=0){

vector<int>::iterator it;

it=find(review->begin(),review->end(),i); //查找是否存在

if (it==review->end())

{

count++;

review->push\_back(G->Vhmap[centralID]);

//画边

scene->addLine(75,0,0+count\*160-75,0+pow((-1),count)\*50\*count);

backDraw(this->ManageLog,scene,G,review,G->getKeyByValue(i), 0+count\*160, 0+pow((-1),count)\*50\*count);

}

}

}

//向前传播

count = 0;

for(int i = 0;i<G->nodeCount;i++){

if(G->Vhmap[centralID]==i) continue;

if(G->adj[i][G->Vhmap[centralID]]!=0){

vector<int>::iterator it;

it=find(review->begin(),review->end(),i); //查找是否存在

if (it==review->end())

{

count++;

review->push\_back(G->Vhmap[centralID]);

//画边

scene->addLine(-75,0,0-count\*160+75,0+pow((-1),count)\*50\*count);

propDraw(this->ManageLog,scene,G,review,G->getKeyByValue(i), 0-count\*160, 0+pow((-1),count)\*50\*count);

}

}

}

scene->update();

}

void backDraw(manageLog \* ManageLog,QgraphicsScene \* scene, EventGraph \*G,vector<int> \*review, int centralID,int X,int Y){

Gnode \* gnode = new Gnode(Qstring::number(centralID),Qstring::fromStdString(ManageLog->getLog(centralID)->getLogName()));

gnode->setPos(X,Y);

// review->push\_back(G->Vhmap[centralID]);

scene->addItem(gnode);

int count = 0;

for(int i = 0;i<G->nodeCount;i++){

if(G->Vhmap[centralID]==i) continue;

if(G->adj[G->Vhmap[centralID]][i]!=0){

scene->addLine(X,Y,X+count\*160, Y+pow((-1),count)\*50\*count);

vector<int>::iterator it;

it=find(review->begin(),review->end(),i); //查找是否存在

if (it!=review->end())

{

count++;

review->push\_back(G->Vhmap[centralID]);

backDraw(ManageLog,scene,G,review,G->getKeyByValue(i), X+count\*80, Y+pow((-1),count)\*50\*count);

}

}

}

}

void propDraw(manageLog \* ManageLog, QgraphicsScene \* scene, EventGraph \*G,vector<int> \*review, int centralID,int X,int Y){

Gnode \* gnode = new Gnode(Qstring::number(centralID),Qstring::fromStdString(ManageLog->getLog(centralID)->getLogName()));

gnode->setPos(X,Y);

// review->push\_back(G->Vhmap[centralID]);

scene->addItem(gnode);

int count = 0;

for(int i = 0;i<G->nodeCount;i++){

if(G->Vhmap[centralID]==i) continue;

if(G->adj[i][G->Vhmap[centralID]]!=0){

// scene->addLine(X,Y,X-count\*160, Y-pow((-1),count)\*50\*count);

vector<int>::iterator it;

it=find(review->begin(),review->end(),i); //查找是否存在

if (it!=review->end())

{

count++;

review->push\_back(G->Vhmap[centralID]);

backDraw(ManageLog,scene,G,review,G->getKeyByValue(i), X-count\*80, Y-pow((-1),count)\*50\*count);

}

}

}

}

/\*\*

\* @brief manageGraph::LoadGraph 加载图

\* @return

\*/

bool manageGraph::LoadGraph(){

//打开文件

std::ifstream infile(“EventGraph”);

if(!infile.is\_open()) return false;

int total;

infile >>total;

EventGraph \* G;

//读取图数量，循环读取图的邻接矩阵信息

for(int i = 0;i<total;i++){

G = new EventGraph();

infile >> \*G; //（>>运算符已重载）

this->addGraph(G);

}

infile.close();

return true;

}

/\*\*

\* @brief manageGraph::SaveGraph 保存图

\* @return

\*/

bool manageGraph::SaveGraph(){

std::ofstream outfile(“EventGraph”);//打开文件

outfile <<this->graphCount<<’ ‘;//输出图的数量

//循环输出图的玲姐矩阵

for(int i = 0;i<this->graphCount;i++){

if(this->Graph[i]!=NULL){

outfile<< \*Graph[i];//（<<运算符已重载）

}

}

return true;

}

### 9. adddialog.h:

#include <Qdialog>

#include<src/managelog.h>

namespace Ui {

class AddDialog;

}

class AddDialog : public Qdialog

{

Q\_OBJECT

public:

manageLog \* ManageLog;//管理LOG类

AddDialog(Qwidget \*parent,manageLog \* manage\_log);

~AddDialog();

private slots:

void on\_buttonBox\_accepted();

void on\_buttonBox\_rejected();

private:

Ui::AddDialog \*ui;

};

### 10. adddialog.cpp:

#include “adddialog.h”

#include “ui\_adddialog.h”

#include <QmessageBox>

#include<src/eventlog.h>

#include<src/managelog.h>

AddDialog::AddDialog(Qwidget \*parent,manageLog \*manage\_log) :

Qdialog(parent),

ui(new Ui::AddDialog)

{

ui->setupUi(this);

this->ManageLog = manage\_log;

}

AddDialog::~AddDialog()

{

delete ui;

}

void AddDialog::on\_buttonBox\_accepted(){

if(ui->EventIDEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志ID！”);

}

else if(ui->LogNameEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志名称！”);

}

else if(ui->ClassTypeEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入级别！”);

}

else if(ui->DescriptionEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志描述！”);

}

else if(ui->EventRecordEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志进程！”);

}

else if(ui->KeyWordEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入关键字！”);

}

else if(ui->SourceIDEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志来源！”);

}

else if(ui->TaskTypeEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入日志类别！”);

}

else if(ui->UserEdit->text()==””){

QmessageBox::about(this, “警告”, “警告：请输入调用者！”);

}

else{

//添加节点，直接调用构造器即可

EventLog \* log = new EventLog(ui->LogNameEdit->text().toStdString(),ui->SourceIDEdit->text().toInt(), ui->OccurTimeEdit->dateTime().toSecsSinceEpoch(),

ui->EventIDEdit->text().toInt(),ui->TaskTypeEdit->text().toInt(),

ui->ClassTypeEdit->text().toInt(),ui->UserEdit->text().toStdString(),

ui->EventRecordEdit->text().toInt(),ui->KeyWordEdit->text().toInt(),

ui->DescriptionEdit->text().toStdString());

if(this->ManageLog->AddLog(log)){

QmessageBox::about(this, “添加成功”, “恭喜！添加成功”);

ui->EventIDEdit->setText(“”);ui->LogNameEdit->setText(“”);ui->SourceIDEdit->setText(“”);

ui->ClassTypeEdit->setText(“”);ui->TaskTypeEdit->setText(“”);ui->DescriptionEdit->setText(“”);

ui->UserEdit->setText(“”);ui->EventRecordEdit->setText(“”);ui->KeyWordEdit->setText(“”);

}

else {

QmessageBox::about(this, “错误”, “该ID已有事件！”);

}

}

return this->show();

}

void AddDialog::on\_buttonBox\_rejected()

{

this->destroy();

}

### 11. mainpage.h:

#include <QmainWindow>

#include”src/managelog.h”

#include”src/managegraph.h”

#include”QstandardItemModel”

#include<QpushButton>

#include<QgraphicsScene>

namespace Ui {

class MainPage;

}

class MainPage : public QmainWindow

{

Q\_OBJECT

public:

manageLog \* ManageLog;//管理LOG

manageGraph \* ManageGraph;//管理图

QstandardItemModel \* Tablemodel;//日志操作的Model

QgraphicsScene \* GraphScene;//作图类

explicit MainPage(Qwidget \*parent = 0);

~MainPage();

private slots:

void on\_AddButton\_clicked();

void on\_QueryButton\_clicked();

void on\_SaveButton\_clicked();

void on\_pushButton\_clicked();

void on\_action\_Delete\_ID\_triggered();

void on\_action\_Dleete\_RecordID\_triggered();

void on\_action\_delete\_Source\_triggered();

void on\_GraphQuery\_clicked();

void on\_GraphUpdate\_clicked();

void on\_GraphClear\_clicked();

void on\_Analysis\_clicked();

void on\_clear\_Button\_clicked();

void on\_Save\_Button\_clicked();

private:

Ui::MainPage \*ui;

};

### 12. mainpage.cpp:

#include “mainpage.h”

#include “ui\_mainpage.h”

#include <QcheckBox>

#include<QlineEdit>

#include<adddialog.h>

#include<vector>

#include<set>

#include<map>

#include<QdateTime>

#include<QmessageBox>

#include<QinputDialog>

#include”src/managegraph.h”

#include”tools/gnode.h”

#include”tools/tool.h”

MainPage::MainPage(Qwidget \*parent) :

QmainWindow(parent),

ui(new Ui::MainPage)

{

ui->setupUi(this);

//设置表格

Tablemodel = new QstandardItemModel();

//设置列号

Tablemodel->setColumnCount(10);

Tablemodel->setHeaderData(0,Qt::Horizontal,Qstring::fromUtf8(“事件ID”));

Tablemodel->setHeaderData(1,Qt::Horizontal,Qstring::fromUtf8(“日志名称”));

Tablemodel->setHeaderData(2,Qt::Horizontal,Qstring::fromUtf8(“发生时间”));

Tablemodel->setHeaderData(3,Qt::Horizontal,Qstring::fromUtf8(“任务类别”));

Tablemodel->setHeaderData(4,Qt::Horizontal,Qstring::fromUtf8(“级别”));

Tablemodel->setHeaderData(5,Qt::Horizontal,Qstring::fromUtf8(“日志来源”));

Tablemodel->setHeaderData(6,Qt::Horizontal,Qstring::fromUtf8(“事件用户”));

Tablemodel->setHeaderData(7,Qt::Horizontal,Qstring::fromUtf8(“进程ID”));

Tablemodel->setHeaderData(8,Qt::Horizontal,Qstring::fromUtf8(“关键字”));

Tablemodel->setHeaderData(9,Qt::Horizontal,Qstring::fromUtf8(“描述”));

ui->tableView->setModel(Tablemodel);

//加载manage文件

this->ManageLog = new manageLog();

this->ManageGraph = new manageGraph(this->ManageLog);

//设置绘图工具

GraphScene = new QgraphicsScene();

ui->graphicsView->setScene(GraphScene);

GraphScene->addText(“你好！”);

}

MainPage::~MainPage()

{

delete Tablemodel;

delete GraphScene;

delete this->ManageLog;

delete this->ManageGraph;

}

void MainPage::on\_AddButton\_clicked()

{

AddDialog \* adddialog;

adddialog = new AddDialog(this,(manageLog \*)(this->ManageLog));

adddialog->exec();

}

void MainPage::on\_QueryButton\_clicked()

{

Tablemodel->removeRows(0,Tablemodel->rowCount());

vector<int> \*SR[5] = {NULL,NULL,NULL,NULL,NULL};

if(ui->EventIDCheck->isChecked()) SR[0] = ManageLog->searchByID(ui->EventIDEdit->text().toInt());

if(ui->EventRecordIDCheck->isChecked()) SR[1] = ManageLog->searchByRecordID(ui->EventREdit->text().toInt());

if(ui->EventTaskCheck->isChecked()) SR[2] = ManageLog->searchByTaskType(ui->EventTaskEdit->text().toInt());

if(ui->EventUserCheck->isChecked()) SR[3] = ManageLog->searchByUser(ui->EventUserEdit->text().toStdString());

if(ui->TimeCheck->isChecked()) SR[4] = ManageLog->searchByTime(ui->TimeEditLow->dateTime().toSecsSinceEpoch(),

ui->TimeEditHigh->dateTime().toSecsSinceEpoch());

std::set <int>SS;

for(int i = 0;i<5;i++){

if(SR[i]==NULL) continue;

for(int e:\*SR[i]){

SS.insertI;

}

delete SR[i];

}

int j = 0;

for(int e:SS){

Tablemodel->setItem(j,0,new QstandardItem(Qstring::number(ManageLog->logs[e]->getID())));

Tablemodel->setItem(j,1,new QstandardItem(Qstring::fromStdString(ManageLog->logs[e]->getLogName())));

QdateTime d;d = QdateTime::fromSecsSinceEpoch(ManageLog->logs[e]->getTime());

d.setTimeSpec(Qt::LocalTime);

Tablemodel->setItem(j,2,new QstandardItem(d.toString(“yyyy/M/d H:mm:ss”)));

Tablemodel->setItem(j,3,new QstandardItem(Qstring::number(ManageLog->logs[e]->getTaskType())));

Tablemodel->setItem(j,4,new QstandardItem(Qstring::number(ManageLog->logs[e]->getClassType())));

Tablemodel->setItem(j,5,new QstandardItem(Qstring::number(ManageLog->logs[e]->getSourceID())));

Tablemodel->setItem(j,6,new QstandardItem(Qstring::fromStdString(ManageLog->logs[e]->getUser())));

Tablemodel->setItem(j,7,new QstandardItem(Qstring::number(ManageLog->logs[e]->getEventRecordID())));

Tablemodel->setItem(j,8,new QstandardItem(Qstring::number(ManageLog->logs[e]->getKeyWord())));

Tablemodel->setItem(j,9,new QstandardItem(Qstring::fromStdString(ManageLog->logs[e]->getDescription())));

j++;

}

ui->tableView->horizontalHeader()->setDefaultAlignment(Qt::AlignCenter);

ui->tableView->resizeColumnsToContents();

}

void MainPage::on\_SaveButton\_clicked()

{

if(ManageLog->SaveData()){

QmessageBox::about(this, “存盘成功”, “恭喜，存盘成功！”);

}

else{

QmessageBox::about(this, “存盘失败”, “错误，存盘失败！”);

}

}

void MainPage::on\_pushButton\_clicked()

{

for(int i = 0;i<Tablemodel->rowCount();i++){

QstandardItem \* item = Tablemodel->item(i,0);

int id = item->data(0).toInt();

QdateTime time = QdateTime::fromString(Tablemodel->item(i,2)->data(0).toString(), “yyyy/M/d H:mm:ss”);

time.setTimeSpec(Qt::LocalTime);

ManageLog->modifyLog(id,Tablemodel->item(i,1)->data(0).toString().toStdString(),Tablemodel->item(i,5)->data(0).toInt(),

(int)(time.toSecsSinceEpoch()),Tablemodel->item(i,3)->data(0).toInt(),Tablemodel->item(i,4)->data(0).toInt(),

Tablemodel->item(i,6)->data(0).toString().toStdString(),

Tablemodel->item(i,7)->data(0).toInt(),Tablemodel->item(i,8)->data(0).toInt(),

Tablemodel->item(i,9)->data(0).toString().toStdString());

}

QmessageBox::about(this, “修改成功”, “修改成功，请注意及时存盘哦！”);

}

void MainPage::on\_action\_Delete\_ID\_triggered()

{

bool isOK;

Qstring text = QinputDialog::getText(this, “输入待删除的ID”,”请输入待删除的日志ID：”,QlineEdit::Normal,””,&isOK);

// QmessageBox::about(NULL,text,text);

if(isOK) {

if(this->ManageLog->DeleteLog(text.toInt())){

QmessageBox::about(this, “删除成功！”, “删除成功！”);

}

else{

QmessageBox::about(this, “删除失败”, “删除失败！”);

}

}

}

void MainPage::on\_action\_Dleete\_RecordID\_triggered()

{

bool isOK;

Qstring text = QinputDialog::getText(this, “输入待日志的进程ID”,”请输入日志的进程ID：”,QlineEdit::Normal,””,&isOK);

if(isOK) {

if(this->ManageLog->DeleteLogbyRecordID(text.toInt())){

QmessageBox::about(this, “删除成功！”, “删除成功！”);

}

else{

QmessageBox::about(this, “删除失败”, “删除失败！”);

}

}

}

void MainPage::on\_action\_delete\_Source\_triggered()

{

bool isOK;

Qstring text = QinputDialog::getText(this, “输入待删除的日志源ID”,”请输入待删除的日志源ID：”,QlineEdit::Normal,””,&isOK);

if(isOK) {

if(this->ManageLog->DeleteLogbySourceID(text.toInt())){

QmessageBox::about(this, “删除成功！”, “删除成功！”);

}

else{

QmessageBox::about(this, “删除失败”, “删除失败！”);

}

}

}

void MainPage::on\_GraphQuery\_clicked()

{

//初始时先把上一次的作图残留释放

QgraphicsScene \* scene = this->GraphScene;

scene->clear();

manageGraph \* ManageGraph = this->ManageGraph;

if(ManageGraph->GraphCount()==0){

QmessageBox::about(this,”提示”,”数据中没有图文件，请点击右侧“更新”按钮！”);

return ;

}

//如果查询到

int type;EventGraph \* G;

if(ui->SubRadio->isChecked()){

G = ManageGraph->findGraph(ui->IndexEdit->text().toInt(),type = LOG::GraphType::Subject);

}

if(ui->ObjRadio->isChecked()){

G = ManageGraph->findGraph(ui->IndexEdit->text().toInt(),type = LOG::GraphType::Object);

}

if(ui->AccomRadio->isChecked()){

G = ManageGraph->findGraph(ui->IndexEdit->text().toInt(),type = LOG::GraphType::AccomEvent);

}

if(ui->CAERadio->isChecked()){

G = ManageGraph->findGraph(ui->IndexEdit->text().toInt(),type = LOG::GraphType::CauseAndEffect);

}

if(!G){

QmessageBox::about(this,”提示”,”图不存在！”);

return ;

}

if(type!=LOG::GraphType::CauseAndEffect) this->ManageGraph->DrawGraph(scene, G,type);

else{

this->ManageGraph->DrawGraphCE(scene,G,ui->IndexEdit->text().toInt());

}

}

void MainPage::on\_GraphUpdate\_clicked()

{

//首先删除所有的图

ManageGraph->deleteAllGraph();

//其次产生新的图

for(int i = 0;i<ManageLog->size;i++){

//该事件不存在，则进行下一个

if(ManageLog->logs[i]==NULL) continue;

//如果事件存在，且没有形成图，则产生一个这样的图；

if(!ManageGraph->hasGraph(ManageLog->logs[i]->getID(),LOG::GraphType::Subject)){

EventGraph \* g = ManageGraph->generateGraph(ManageLog->logs[i]->getID(),LOG::GraphType::Subject);

ManageGraph->addGraph(g);

}

if(!ManageGraph->hasGraph(ManageLog->logs[i]->getID(),LOG::GraphType::Object)){

EventGraph \* g = ManageGraph->generateGraph(ManageLog->logs[i]->getID(),LOG::GraphType::Object);

ManageGraph->addGraph(g);

}

if(!ManageGraph->hasGraph(ManageLog->logs[i]->getID(),LOG::GraphType::AccomEvent)){

EventGraph \* g = ManageGraph->generateGraph(ManageLog->logs[i]->getID(),LOG::GraphType::AccomEvent);

ManageGraph->addGraph(g);

}

if(!ManageGraph->hasGraph(ManageLog->logs[i]->getID(),LOG::GraphType::CauseAndEffect)){

EventGraph \* g = ManageGraph->generateGraph(ManageLog->logs[i]->getID(),LOG::GraphType::CauseAndEffect);

ManageGraph->addGraph(g);

}

}

this->ManageGraph->SaveGraph();

}

void MainPage::on\_GraphClear\_clicked()

{

this->GraphScene->clear();

}

void MainPage::on\_Analysis\_clicked()

{

if(!(ui->LogNameE->text()!=””&&ui->TaskTypeE->text()!=””&&ui->ClassTypeE->text()!=””&&

ui->KeyWordE->text()!=””&&ui->SourceIDE->text()!=””&&ui->UserE->text()!=””&&

ui->EventRecordE->text()!=””&&ui->DescriptionE->text()!=””)){

QmessageBox::about(this,”提示”,”信息不完整，请补充完整！”);

return;

}

EventLog \* log = new EventLog(999);

log->setLogName(ui->LogNameE->text().toStdString());

log->setTaskType(ui->TaskTypeE->text().toInt());

log->setClassType(ui->ClassTypeE->text().toInt());

log->setKeyWord(ui->KeyWordE->text().toInt());

log->setSourceID(ui->SourceIDE->text().toInt());

log->setUser(ui->UserE->text().toStdString());

log->setTime(ui->OccurTimeE->dateTime().toSecsSinceEpoch());

log->setEventRecordID(ui->EventRecordE->text().toInt());

log->setDescription(ui->DescriptionE->text().toStdString());

Qstring s = ResultStr(this->ManageGraph,this->ManageLog,log);

delete log;

if(s==””){

QmessageBox::about(this,”提示”,”没有相似的事件！”);

ui->ResuleLabel->setPlainText(“分析失败，系统中没有相似的事件！”);

}

else{

ui->ResuleLabel->setPlainText(s);

}

return;

}

void MainPage::on\_clear\_Button\_clicked()

{

ui->LogNameE->setText(“”);

ui->TaskTypeE->setText(“”);

ui->ClassTypeE->setText(“”);

ui->KeyWordE->setText(“”);

ui->SourceIDE->setText(“”);

ui->UserE->setText(“”);

ui->EventRecordE->setText(“”);

ui->DescriptionE->setText(“”);

}

void MainPage::on\_Save\_Button\_clicked()

{

if(!(ui->LogNameE->text()!=””&&ui->TaskTypeE->text()!=””&&ui->ClassTypeE->text()!=””&&

ui->KeyWordE->text()!=””&&ui->SourceIDE->text()!=””&&ui->UserE->text()!=””&&

ui->EventRecordE->text()!=””&&ui->DescriptionE->text()!=””)){

QmessageBox::about(this,”提示”,”信息不完整，请补充完整！”);

return;

}

Qstring str = QinputDialog::getText(this,”输入新日志的ID”,”请输入新日志的ID：”);

int ID = str.toInt();

EventLog \* log = new EventLog(ID);

log->setLogName(ui->LogNameE->text().toStdString());

log->setTaskType(ui->TaskTypeE->text().toInt());

log->setClassType(ui->ClassTypeE->text().toInt());

log->setKeyWord(ui->KeyWordE->text().toInt());

log->setSourceID(ui->SourceIDE->text().toInt());

log->setUser(ui->UserE->text().toStdString());

log->setTime(ui->OccurTimeE->dateTime().toSecsSinceEpoch());

log->setEventRecordID(ui->EventRecordE->text().toInt());

log->setDescription(ui->DescriptionE->text().toStdString());

this->ManageLog->AddLog(log);

QmessageBox::about(this,”添加成功”,”添加成功，请及时存盘！”);

}

### 13. gnode.h:

#include<QGraphicsItem>

#include<QPainter>

#include<string>

/\*\*

\* @brief The GNode class

\* UI作图

\*/

class GNode : public QGraphicsItem

{

public:

QString ID;

QString text;

GNode(QString ID, QString text, QGraphicsItem \*parent = Q\_NULLPTR);

QRectF boundingRect() const{

return QRectF(-75, -25, 150, 50);

}

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option,

QWidget \*widget)

{

QColor Q(0,0,0,255);

painter->setPen(Q);

painter->drawRoundedRect(-75, -25, 150, 50, 5, 5);

painter->drawText(-75, -25, 150, 20, Qt::AlignCenter, this->ID);

painter->drawText(-75,-10,150,30,Qt::AlignCenter,text);

}

};

### 14. gnode.cpp:

#include "gnode.h"

GNode::GNode(QString ID, QString text, QGraphicsItem \*parent)

:QGraphicsItem(parent)

{

this->text = text;

this->ID = ID;

}

### 15. tool.cpp:

#include <cmath>

#include"tools/tool.h"

bool isPrime(int M){

for(int i = 2;i<sqrt(M);i++){

if(M%i==0) return false;

}

return true;

}

int MinPrime(int N){

for(int i = N;i>1;i--){

if(isPrime(i)) return i;

}

return 1;

}

/\*\*

\* @brief Value2Hash 将整数映射到哈希表

\* @param prime 素数

\* @param max 最大值

\* @param value 数值

\* @param array 数组基地址

\* @return 哈希表

\*/

int Value2Hash(int prime,int max,int value,int \* array){

while(value<0) value+=prime;//让值为正数

value = value%prime;//映射到0-prime-1

//寻找空位

while(array[value]!=-1){

value++;

value%=prime;

}

return value;

}

/\*\*

\* @brief similarLog 获得相似的事件，并返回成vector

\* @param ManageLog

\* @param log

\* @return

\*/

vector<EventLog \*> similarLog(manageLog \* ManageLog,EventLog log){

vector<EventLog \*> simlog;

for(int i = 0;i<ManageLog->size;i++){

EventLog \* founded = ManageLog->logs[i];

if(founded==NULL) continue;

int count = 0;//相似因子,当相似因子的数大于5时，视为该LOG相似

if(founded->getLogName().compare(log.getLogName())==0) count++;

if(founded->getTaskType()==log.getTaskType()) count++;

if(founded->getClassType()==log.getClassType()) count++;

if(founded->getKeyWord()==log.getKeyWord()) count++;

if(founded->getSourceID()==log.getSourceID()) count++;

if(founded->getUser().compare(log.getUser())==0) count++;

if(count >= 5) simlog.push\_back(founded);

}

return simlog;

}

/\*\*

\* @brief PriorNode 获得一个节点的所有前驱节点

\* @param ManageGraph 图管理器

\* @param log 节点事件

\* @return

\*/

vector<EventLog \*> PriorNode(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog log){

EventGraph \* G = ManageGraph->findGraph(log.getID(),LOG::GraphType::CauseAndEffect);

vector<EventLog \*> vec;

for(int i = 0;i<G->nodeCount;i++){

if(G->adj[i][G->VHmap[log.getID()]]==1){//邻接矩阵的值为1时，保存相应的节点

int id = G->getKeyByValue(i);//反向获得ID

vec.push\_back(ManageLog->getLog(id));

}

}

return vec;

}

/\*\*

\* @brief NextNode 获得一个节点的所有后继节点

\* @param ManageGraph

\* @param ManageLog

\* @param log

\* @return

\*/

vector<EventLog \*> NextNode(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog log){

EventGraph \* G = ManageGraph->findGraph(log.getID(),LOG::GraphType::CauseAndEffect);

vector<EventLog \*> vec;

for(int i = 0;i<G->nodeCount;i++){

if(G->adj[G->VHmap[log.getID()]][i]==1){//邻接矩阵的值为1时，保存相应的节点

int id = G->getKeyByValue(i);//反向获得ID

vec.push\_back(ManageLog->getLog(id));

}

}

return vec;

}

/\*\*

\* @brief ResultStr 根据查询结果获得最终评判字符串

\* @param ManageGraph

\* @param ManageLog

\* @param log

\* @return

\*/

QString ResultStr(manageGraph \* ManageGraph, manageLog \* ManageLog, EventLog \* log){

//查询相似事件的结果

vector <EventLog\*> simlog = similarLog(ManageLog,\*log);

if(simlog.empty()){

return "";

}

QString labelText;

int num = (int)simlog.size();

QString \* strSet = new QString[num];

labelText = "过往的事件总共有"+QString::number(num)+"种情况，我们做了如下预测：\n";

for(int i = 0;i<num;i++){

strSet[i] = "第"+QString::number(i+1)+"种情况：";

EventLog \* eventlog = simlog.back();

strSet[i] = strSet[i]+"本次事件与ID为"+QString::number(eventlog->getID())+"的事件相似，所以我们推测";

simlog.pop\_back();

//获得前驱和后记事件；然后给出结果

vector<EventLog \*> prior = PriorNode(ManageGraph, ManageLog, \*eventlog);

vector<EventLog \*> nextN = NextNode(ManageGraph, ManageLog, \*eventlog);

QString pr,nx;

for(int j = 0;j<(int)(prior.size());j++){

EventLog \* log2 = prior.back();prior.pop\_back();

pr = pr+" “"+ QString::fromStdString(log2->getDescription()) +"”";

}

for(int j = 0;j<(int)(nextN.size());j++){

EventLog \* log3 = nextN.back();nextN.pop\_back();

nx = nx+" “"+ QString::fromStdString(log3->getDescription()) +"”";

}

if(pr.length()==0){

strSet[i] = strSet[i]+"本次事件可能是源头事件,";

}

else{

strSet[i] = strSet[i] + "本次事件可能是"+pr+"导致的，";

}

if(nx.length()==0){

strSet[i] = strSet[i] +"本次事件是汇点事件。\n";

}

else{

strSet[i] = strSet[i] + "随后可能会发生"+nx+"等事件。\n";

}

}

for(int i = 0;i<num;i++){

labelText = labelText+strSet[i];

}

delete []strSet;

return labelText;

}

### 16. main.cpp:

#include "mainpage.h"

#include <QApplication>

#include<QFile>

#include <QTextCodec>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QFile qss("style.qss");

qss.open(QFile::ReadOnly);

qApp->setStyleSheet(qss.readAll());

qss.close();

MainPage w;

w.show();

return a.exec();

}