.NET开发大作业（2014）

minsql

|  |  |  |
| --- | --- | --- |
| 1 | 程序运行结果（10分） |  |
| 2 | 代码书写（10分） |  |
| 3 | 程序功能（10分） |  |
| 4 | 文档完整度（10分） |  |
| 5 | 回答问题（10分） |  |
|  | 大作业总成绩 |  |

**本组成员：**

|  |  |  |
| --- | --- | --- |
|  | 姓名和学号、班级 | 成绩 |
| 1 | 邓万程(201292461) 软1201 |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

基于正则引擎行读取的minsql

背景:

本人熟悉sql语法并做过大量基于数据库的web应用，希望有所突破，深入底层，探究sql解释集及数据存储的实现.

背景知识:

通过调查研究，发现数据库的实现大多是基于C/C++汇编这类底层语言实现的，而C#的优势在于OOP，抽象，所以选择了一种简易实现sql的方式，

基于正则引擎和行读取的实现.mysql

索引是由B+数实现，且直接操纵磁盘地址，并带有缓存功能，散列基于hash表，实现更为复杂，不再累述.

优点在于:

mysql等一般sql语言是基于上下文无关文法实现的，故报错机制不是细节，正则引擎的实现则会带来更多的细节报错. 统计目前实现的代码，可报错50处.

可为初学sql语句的初学者带来好的学习体验.

实现了数据库sql基本语句, 从文件恢复数据库, 由于时间限制, 表的连接, 通信API, 事务机制未实现.

系统架构

主要有三个框架

Main loop

Tools funtions [ string op, file op ]

Sql engine

Error report

流程如下：

Main loop : true{

Sql engine <- 用户输入

Call string op;

if (err)

call error\_report

else

File op;

}

实现:

Main loop: class sql\_main

(35 lines)

功能:

等待用户输入送至sql\_engine

命令行时可接受文件处理

Sql\_engine:

Class sql\_engine:

(480 lines)

功能:

正则引擎分析用户输入

,获取表信息

,获取满足条件的文件行

,包含报错机制

Class sql\_engine\_optional:

(60 lines)

功能:

扩展sql语句

Tools functions:

Class Str\_op

(220 lines)

功能: 处理字符串, 生成指定字符串

如以 “!=” 分割字符串去除串两旁空格, 解析生成自定字符串 ...

Class File\_op

(120 lines)

功能: 对文件的随机读写

Class File\_sql\_deal:

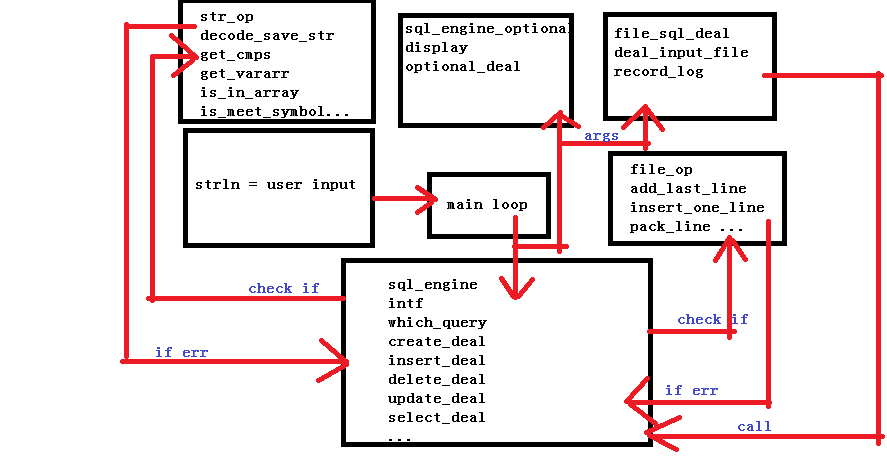
(20 lines)

功能: 对用户输入的记录(正确错误皆记录)

及从文件恢复数据库

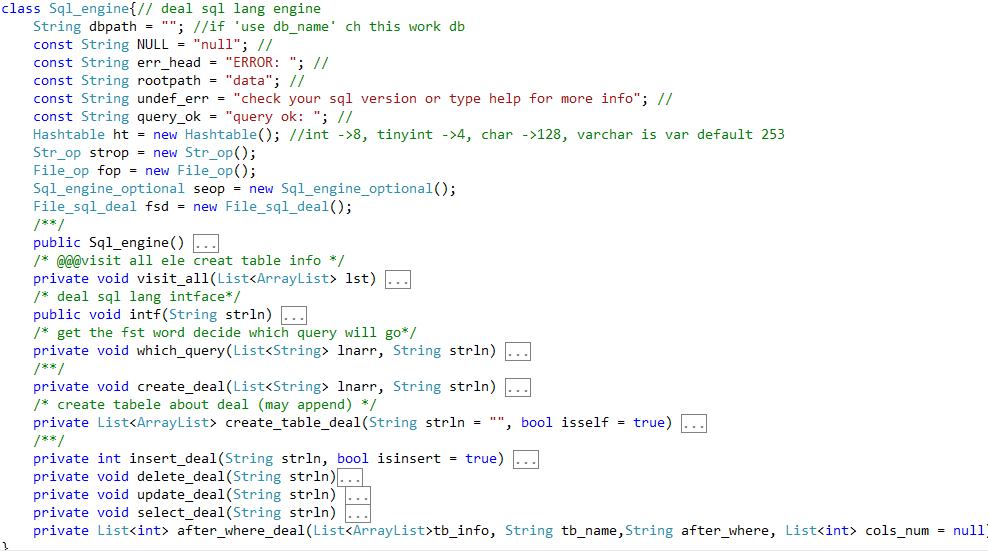
Total (935 lines)

系统设计图



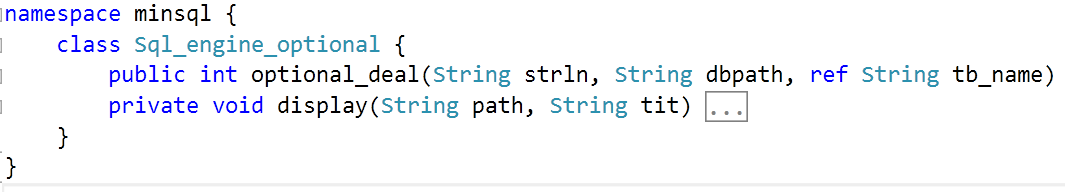
类视图:

Sql\_ engine



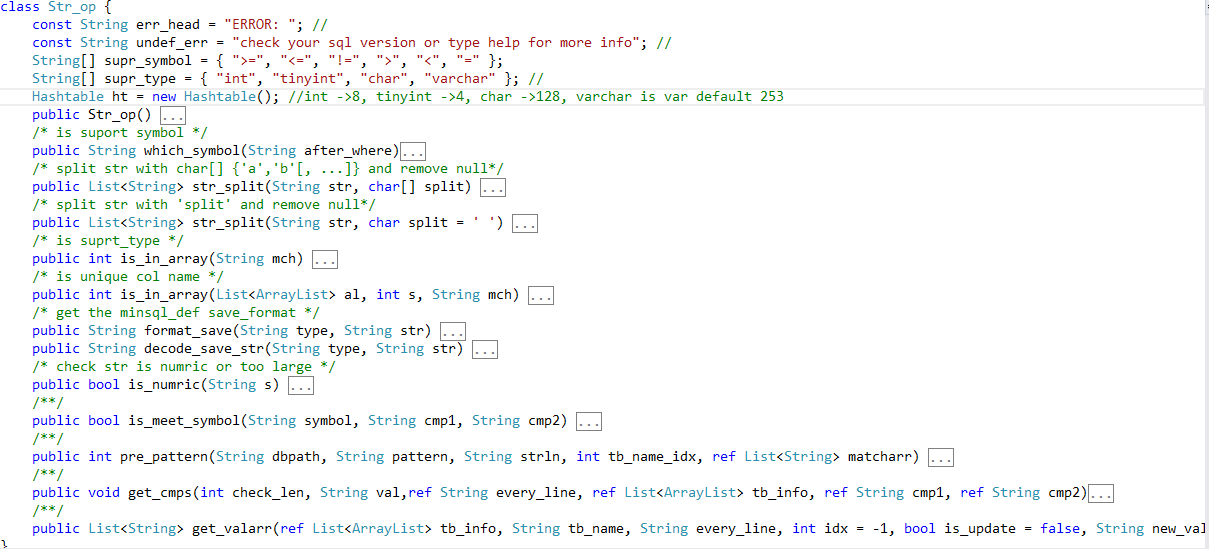
这是sql引擎每一个函数都很重要

Sql\_engine\_optional



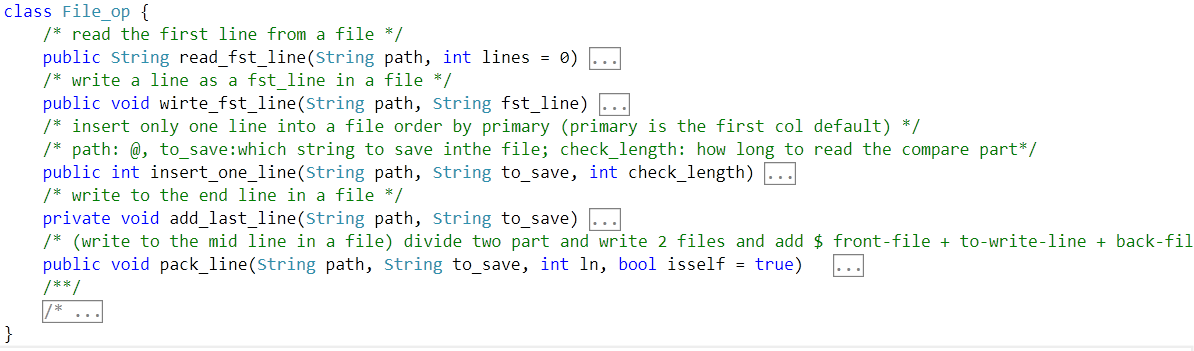
扩展sql, 最好带上

Str\_op



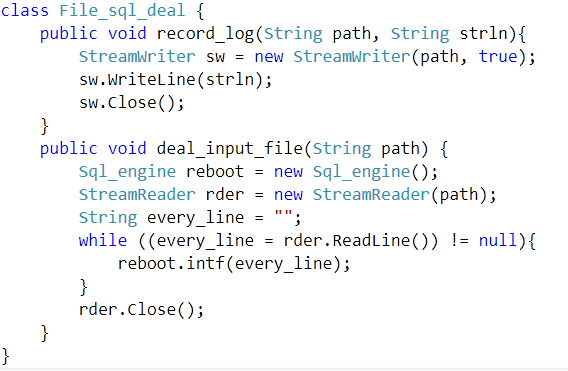
Sql语句解析, 及文件行解析依赖于str\_op, 重要

File\_op



数据存储及获取依赖于file\_op，重要

File\_sql\_deal



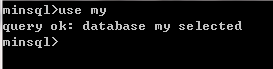
数据库记录与恢复, 可选

以上为类视图代码，秉承代码规则为

最大化复用，一个模块只关心一个功能，低耦合

运行界面

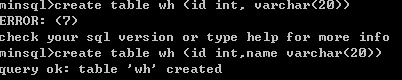
Use db:



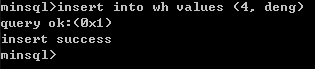
Create database dbnm



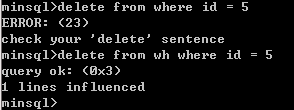
Create table tbnm (col1, type[, ...])



Insert into tbnm [(...)] values (val1[, ...])



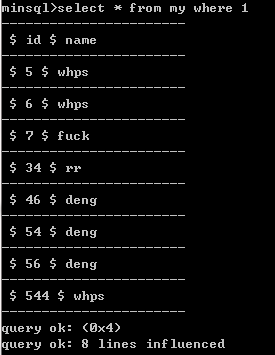
Delete from tbnm where (...)



Update tbnm set (...) where (...)

...

Select (...) from tbnm where (...)



还有一些语句测试即可

系统总结

/\*

\*\*只支持英文字符集

\*\*不支持null, 意味着每列必须要指定值

\*\*只支持行输入

\*\*只支持关键字小写

\*\*where 支持单条件

\*\*update支持单列修改

\*\*默认主键为第一列且顺序排列(ASC)

\*\*update 可能打乱顺序(已改)

\*\*/

$

系统与原先设计并不是很吻合, 写代码过程发现, 处理简单情况更容易

$

条件分支太多 性能肯定慢

$

before uptade 之前的句子肯定有正则匹配模式的错误大概是应该"+"的模式为"\*"(已改)

$

越到后期变量名越含糊不清

$

create\_table\_deal 写的效率低下

@

用到的文件句柄,要记得关

@

写单元测试会大大加大开发速度

@

双层的循环容易出现把 i 当成 j 来用

@

把搞不定的未知 or 内存引用错误，统统抛给 try-catch 简单易用