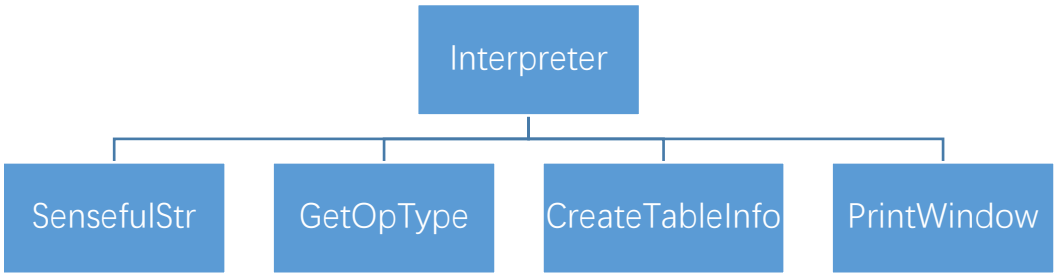
Interpreter and API　3180103481 孙家阳

Interpreter Module

基本框架:



基本功能描述:

将用户输入的语句切割成字符串数组 SensefulStr 作为指令，并剔除无关信息（如，；等），根据字符串数组用 GetOpType 得到该指令类型，创建对应指令的信息存储结构 CreateTableInfo，然后调用 API 执行指令 Interpreter，最后反馈执行结果 PrintWindow

具体接口:

（I）字符串数组 SensefulStr

```
class SensefulStr
{
public:
    SensefulStr(std::string srcstr = "");
    void SetSrcStr(std::string _srcstr);

    std::vector<std::string> GetSensefulStr()const;
private:
```

```
    void Parse();
          // 解析命令为有意字串

    std::string src_str;
          // 原始命令字符串
    std::vector<std::string> sen_str;
          // 解析后的又一字串
    std::string key_char = ";,()=<>\012\015\040";
    bool IsKeyChar(char c);
};
```

（i）成员变量：

src_str：包含一个命令原始命令字符串，调用 getline（），若一条命令包含多行，则仍将其整合为一个字符串

sen_str：经过切割处理后的字符串数组，包含一组命令

key_char：命令中的无关字符，如“,;”等都会被过滤，注意只支持英文符号

（ii）构造函数

调用 parse（）函数对原命令字符串进行切割，过滤无关信息，并转化为数组

（iii）其他成员函数

GetSensefulStr：返回 sen_str

Parse：去除原字符串中的";,()=<>"及单/双引号和空白符，将其转化为字符串数组并保存在 sen_str 中

SetSrcStr：对 src_str 赋值

（II）GetOpType 判断指令类型函数

根据 sen_str 字符串数组，并结合所要求支持的 SQL 指令，判断指令类型，目前支持以下类型

```
enum class CmdType
{
    TABLE_CREATE, TABLE_DROP, TABLE_SELECT, TABLE_INSERT, TABLE_DELETE,
    DB_CREATE,DB_USE,
    INDEX,EXECFILE, QUIT, HELP
};
```

表的创建/删除，记录的选择/插入/删除，数据库的创建/使用，索引（含创建与使用），退出，提供帮助信息。若用户输入指令不属于上述类型，则抛出异常

（III）创建指令的信息存储结构 CreateTableInfo 模块

字符串数组 SensefulStr 是初步加工的指令，接下来要对其进一步修饰加工，提供 API 能接受执行的指令信息，即根据不同类型的指令创建合适的结构存储。主要包含以下函数:

```
//数据库操作相关信息,返回数据库名称
// 创建数据库
std::string CreateDbInfo(std::vector<std::string> sen_str);
// 使用数据库
std::string UseDbInfo(std::vector<std::string> sen_str);

// 表操作相关信息，返回存储有效信息的 struct
//创建表
TB_Create_Info CreateTableInfo(std::vector<std::string> sen_str);
//删除表
std::string DropTableInfo(std::vector<std::string> sen_str);

//记录操作相关信息，返回存储有效信息的 struct
//插入记录
TB_Insert_Info CreateInsertInfo(std::vector<std::string> sen_str);
//选择记录
```

```
TB_Select_Info TableSelectInfo(std::vector<std::string> sen_st
r);
//删除记录
TB_Delete_Info TableDeleteInfo(std::vector<std::string> sen_st
r);

//索引操作相关信息，生成所需创建/删除的索引信息
Index_Info IndexInfo(std::vector<std::string> sen_str);
```

在创建信息存储结构时主要考虑以下两方面：

1）指令格式是否完整且正确，若不是，抛出自定义的 SQLError::CMD_FORMAT_ERROR 异常

2）初步检查该指令的操作是否满足某些字段的属性约束，如 primary key 约束，unique key 约束等，若不是，也会抛出上述异常

（IV）反馈执行结果 PrintWindow

调用 API 模块执行对应类型的指令后，API 相关接口会返回本次运行状态成功与否，由 PrintWindow 接受，并根据运行状态向用户反馈，呈现在 console 中

```
#define PRINTLENGTH 63
class PrintWindow
{
public:
    void CreateTable(bool is_created);
    void DropTable(bool is_dropped);
    void SelectTable(SelectPrintInfo select_table_print_info);
    void InsertRecord(bool is_inserted);
    void CreateDB(bool is_created);
    void UseDB(bool isUsed);
    void DeleteTable(bool isDeleted);
    void doIndex(bool isDone);
private:
    void Print(int len, std::string s); // 打
印 |xxxx         | 其中竖线内长度为 len
    int GetColumnLength(std::string name,std::vector<std::stri
ng> col_name, std::vector<int> col_len);
```

```
};
```

相关接口：

CreateTable：判断表是否创建成功，若成功

```
std::cout << "table create succeed!" << std::endl;
```

若创建未成功（运行时抛出异常），则

```
std::cout << "table create failed!" << std::endl;
```

DropTable：判断表是否删除成功，输出方式类似上，下同

SelectTable：判断记录是否筛选成功

InsertRecord：判断记录是否插入成功

CreateDB：判断创建当前数据库是否成功

UseDB：判断使用当前数据库是否成功

DeleteTable：判断记录是否删除成功

doIndex：判断索引的建立/删除操作是否成功

Print：格式化打印，使指令的返回结果分栏并列对齐，每栏长度为 len，同时打印栏分隔符“|”

GetColumnLength：获得当前记录该字段的值的长度，若大于该字段所允许的最大长度，则返回后者

（V）Interpreter 接口

提供给 main 函数，用于解释执行语句的接口，接受字符串数组 SensefulStr，指令类型 CmdType，执行反馈结果信息 PrintWindow，对于每种类型的指令，先创建指令信息，再执行操作，最后向用户反馈结果。无法识别的指令会抛出异常。

API Module

基本功能描述:

向下：根据 Interpreter 层解释生成的命令内部形式，调用 Catalog Manager 提供的信息进行进一步的验证及确定执行规则，然后调用 Record Manager、Index Manager、Catalog Manager 及 B+树模块提供的相应接口执行各 SQL 语句及命令语句。

向上：API 模块进一步进行语法与逻辑检查，抛出可能的异常，提供执行 SQL 语句的接口供 Interpreter 层调用。

具体接口:

主要供 Interpreter 调用的接口有以下八个:

```
// 创建数据库
bool CreateDatabase(std::string database_name, CatalogPosition
 &cp);


// 选择数据库
bool UseDatabase(std::string db_name, CatalogPosition &cp);


// 创建表
bool CreateTable(TB_Create_Info tb_create_info, std::string pa
th = std::string("./"));

// 删除表
bool DropTable(std::string table_name, std::string path = std:
:string("./"));


// 插入记录
```

```
bool InsertRecord(TB_Insert_Info tb_insert_info, std::string p
ath = std::string("./"));

// 选择记录
SelectPrintInfo SelectTable(TB_Select_Info tb_select_info, std
::string path = std::string("./"));


// 删除记录
bool DeleteTable(TB_Delete_Info tb_delete_info, std::string pa
th = std::string("./"));

//插入/删除索引
bool doIndex(Index_Info index_info, std::string path = std::st
ring("./"));
```

其余接口为 API 内部调用，会在介绍主要接口时附带介绍

（I）CreateDatabase 创建数据库

判断当前路径下数据库是否存在，若存在，则返回不成功，并输出错误信息；若不存在则创建子目录作为数据库

（II）UseDatabase 使用数据库

先判断数据库是否存在，若存在，将数据库根目录路径定位到此数据库下，若不存在，返回不成功并输出错误信息

（III）CreateTable 创建表

先检查信息，调用 Record 模块提供的 Check_TB_Create_Info，检查检查创建信息（如字段数量是否溢出，primary key 是否唯一，字段类型是否支持等），以及当前目录是否在某个数据库中。

然后调用 B+树模块创建索引文件.idx（primary key 会默认创建索引，该索引不

能被用户创建或删除，若表没有指定 primary key，则指定第一个字段为 primary key），并调用 buffer 模块创建数据文件.dbf

（IV）DropTable 删除表

先判断所需删除的表是否存在，然后判断该表对应的数据文件是否已经被打开使用（若已打开，则需要先丢弃在内存中的文件页，避免程序结束再次写回）。

找到所有本表数据文件.dbf 及本表上的索引文件.idx，删除所有文件

（V）InsertRecord 插入记录

先检查信息，调用 Record 模块提供的 Check_TB_Insert_Info，检查检查创建信息以及所操作的表是否在当前目录中。

将记录信息封装成记录数据对象，按字段类型分类，找到对应的字段名称并在插入记录里寻找该字段的值，最后将记录插入到数据文件及所有索引中。

在插入前我们要先判断该条记录中 primary/unique 字段的值是否已经存在，这里要用到下列三个函数

```
// 范围查找单个字段 返回查找的值在数据文件中的地址
std::vector<std::pair<KeyAttr, FileAddr>> Search(CompareCell compare_cell, std::string table_name, std::string path = std::string("./"));
std::vector<std::pair<KeyAttr, FileAddr>> KeySearch(CompareCell compare_cell, std::string table_name, std::string path = std::string("./"));  //主键查找
std::vector<std::pair<KeyAttr, FileAddr>> RangeSearch(CompareCell compare_cell, std::string table_name, std::string path = std::string("./"));  // 遍历查找
```

InsertRecord 中直接调用 Search 函数，由 Search 决定采用索引查找

（KeySearch）或遍历查找（RangeSearch）

其中 RangeSearch 函数需要调用 GetDbfRecord 函数来获得指定地址的记录

```
// 取出指定地址的数据
RecordHead GetDbfRecord(std::string table_name, FileAddr fd, s
td::string path = std::string("./"));
```

（VI）SelectTable 选择记录

先检查信息，调用 Record 模块提供的 Check_TB_Select_Info，检查检查创建信息以及所操作的表是否在当前目录中。

判断是否有 where 条件，若无则遍历，若有则调用 Search 函数。

最后函数返回 select 后得到的所有结果。

（VII）DeleteTable 删除记录

操作类似 SelectTable，删除所有指定结果，在查找时调用 Search 函数，同样区分索引查找与遍历查找。

（V）doIndex 创建/删除索引

首先进行异常处理，试图删除不存在的索引和创建已存在的索引都会抛出异常并返回。然后判断该指令为创建还是删除，在对应目录下创建/删除对应表的索引。