数据库系统设计-MiniSQL

interpreter模块详细设计报告

黄昭阳 3130000696 - 11/6/15

interpreter模块总体设计

一、实验要求：

Interpreter模块直接与用户交互，主要实现以下功能：

1． 程序流程控制，即“启动并初始化‘接收命令、处理命令、显示命令结果’循环 退出”流程。

2． 接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和语义正确性，对正确的命令调用API层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。

二、实验完成情况：

基本实现实验要求功能，只是实际处理中与实验要求模块稍微不一样，程序的“启动并初始化‘接收命令、处理命令、显示命令结果’循环”我们将认为放在API模块中实现更符合逻辑，interpreter则只是与用户界面进行交互。

1. 为了更合理的对用户的输入进行语法分析，我们使用多态构建了一系列底层的SQLCommand数据结构，使用数据结构进行数据传输，情况如下：

```
class SQLCommand
class SQLCommandCreateTable:public SQLCommand
class SQLCommandDropTable:public SQLCommand
class SQLCommandCreateIndex:public SQLCommand
class SQLCommandDropIndex:public SQLCommand
class SQLCommandSelectFrom:public SQLCommand
class SQLCommandInsertInto:public SQLCommand
class SQLCommandDeleteFrom:public SQLCommand
class SQLCommandQuit:public SQLCommand
```

SQLCommand为所有类的父类，不同的类用来存储不同的命令需要的信息，具体的类内部实现将在下面进行详细介绍。

2. 由于interpreter在对命令进行解析时需要得到catalog中的table和index的有效信息，因此建立interpreter对API的委托，通过委托来获得有效信息，情况如下：

```
class basicDelegate {
public:
    virtual SQLTable* getTableInfo(string* tableName)=0;
    virtual SQLIndex* getIndexInfo(string* indexName)=0;
    virtual string* getRecordFileName(string* tableName)=0;
    virtual string* getIndexFileName(string* indexName)=0;
};

class SQLCommandInterpreterDelegate:public basicDelegate
{
};
```

从函数名可以了解到函数实现的内容，这里不加赘述，basicDelegate还将被其他的委托类所继承，这将在整体设计报告中得到体现，这里不详细解释。

模块整体详细分析

一、interpreter整体结构分析和代码介绍：

```
class SQLCommandInterpreter
{
private:
    //data
    static int index;
       //当前正在对inputString解析时，解析到的位置

    static int tmp;
    static string inputString;
       //用户输入的带解析成命令的字符串

    static string command;
       //使用nextStep函数后，下一条有效关键字将被以字符串形式载入command

    static int intValue;
    static float floatValue;
    static int isString;
        //使用transfer系列函数将command转化成对应的有效数据后，数据存在这组变量中

    static SQLTable* table;
    static SQLAttribute* attribute;

    static int attributeNumber;
    static string* attributeName;
    static SQLDataType attributeType;
    static string* tableName;
    static string* indexName;
    static int attributeProperty;
    static int attributeSize;
    static string* op;
    static SQLCondition* condition;
    static SQLTuple* tuple;
    static SQLIndex* sqlindex;
        //以上很多私有成员变量都是为了方便字符串解析所使用，之后再代码块中进行解释

   static SQLCommandInterpreterDelegate* m_delegate;
    //拥有一个委托作为私有变量，用来获得各种需要的信息
```

```
    static int transferStringToInt();
    static int transferStringToFloat();
        //将command中的字符串内容转化成int或者float后放到intValue或floatValue内

    static void nextStep();
        //从inputString中解析出下一条有效关键字放入command中

    static SQLCondition* transferStringToCondition();
        //将字符串转化成SQLCondition（数据结构）放到condition中

    static SQLCommand* SQLSelectCommandTransfer();
    static SQLCommand* SQLDeleteCommandTransfer();
    static SQLCommand* SQLCreateTableCommandTransfer();
    static SQLCommand* SQLCreateIndexCommandTransfer();
    static SQLCommand* SQLCreateCommandTransfer();
    static SQLCommand* SQLDropTableCommandTransfer();
    static SQLCommand* SQLDropIndexCommandTransfer();
    static SQLCommand* SQLDropCommandTransfer();
    static SQLCommand* SQLInsertCommandTransfer();
    static SQLCommand* SQLQuitCommandTransfer();
        //分别转化成对应的SQLCommand的，并传出指针，当指针值为NULL时说明转化失败

    static SQLCommand* transferError(string error);
        //当发生传输失败时调用此函数，他将输出error中的错误信息，将所有不为NULL的私
有成员指针delete掉（这样才能防止内存泄露），传出NULL指针

    static void resetPointers();
        //将所有私有成员指针置为NULL

public:
    static SQLCommand* transferStringToCommand(string inputString);
        //传入用户输入的字符串，传出该字符串解析成的命令

    static void setDelegate(SQLCommandInterpreterDelegate* delegate)
    {
        m_delegate = delegate;
    }
        //设置委托

};
```

二、interpreter使用方法：

1. 当决定使用interpreter时，应先调用SQLCommandInterpreter::setDelegate()函数来对interpreter设置一个能够获得信息的委托

2. 通过函数transferStringToCommand(getInputString())来获得由字符串解析出来的命令，命令形式为SQLCommand系列的数据结构。

3. 使用示例：

```
    SQLCommandInterpreter::setDelegate(new A());
    SQLCommand* command;
    while(1)
    {
        command =
SQLCommandInterpreter::transferStringToCommand(getInputString());
        if(command == NULL)
            cout << endl << "Failed!" << endl;
    }
        //意义很明显，这里不加赘述，在API中command将被用作判断下一步业务逻辑的关键，
而不是只判断是否为NULL
```

模块关键函数详细介绍与分析

1. 用来解析下一条有效关键字的函数nextStep

```
void SQLCommandInterpreter::nextStep()
{
    cout << "SQLCommandInterpreter::nextStep" << endl;
    int charFlag = 0;
    isString = 0;
    while(isSeperator(inputString[index]))
    {
       index++;
    } //过滤无意义分隔符
    if( inputString[index] == '(' )
    {
        index++;
        command = "(";
    } else if( inputString[index] == ')' ) //对于括号特殊判断
    {
        index++;
        command = ")";
    } else {
        tmp = index;
        while(!isSeperator(inputString[tmp])||charFlag) //将不是分隔符的加入
        {
            if(inputString[tmp] == '\0') break;
            if(inputString[tmp] == '(') break;
            if(inputString[tmp] == ')') break;
            if(inputString[tmp] == '\'')
            {
                charFlag = !charFlag;
                if(charFlag) index++;
                isString = 1;
            }
            tmp++;
        }
        command = inputString.substr(index,tmp-index-isString);
              //取出关键字到command内
        index = tmp;
    }
}
```

分析： 根据分隔符取出关键字，将关键字放到command中，同时index更新位置以便下一次取下一个关键字

2. 接受输入字符串的接口函数transferStringToCommand()

```
SQLCommand* SQLCommandInterpreter::transferStringToCommand(string
inputString)
{
    cout << "SQLCommandInterpreter::transferStringToCommand" << endl;
    cout << "inputString = " << inputString << endl;
    index = 0;
    SQLCommandInterpreter::inputString = inputString;
    nextStep();
    if(command.compare("select") == 0) return SQLSelectCommandTransfer();
    if(command.compare("delete") == 0) return SQLDeleteCommandTransfer();
    if(command.compare("create") == 0) return SQLCreateCommandTransfer();
    if(command.compare("drop") == 0) return SQLDropCommandTransfer();
    if(command.compare("insert") == 0) return SQLInsertCommandTransfer();
    if(command.compare("quit") == 0) return SQLQuitCommandTransfer();
    return NULL;
}
```

分析：　　获得第一个关键字，根据第一个关键字来判断应该构建哪一个种命令，其中create和drop命令内根据index和table的不同还有再次的分流，其内部全部是字符串的处理，这里不再详细描述。

interpreter模块测试

一、测试代码

```
#include <iostream>
#include "interpreter.hpp"

class A:public SQLCommandInterpreterDelegate
{
    SQLTable* getTableInfo(string* tableName);
};

SQLTable* A::getTableInfo(string* tableName)
{
    cout << "getTableInfo: " << *tableName << endl;
    if(tableName->compare("student") != 0) return NULL;
    SQLTable* table = new SQLTable(tableName);
    SQLAttribute* attribute;
    string* name;
    SQLDataType type;
    int property,size;

    name = new string("SID");
    type = CHAR;
    property = PRIMARY;
    size = 5;
    attribute = new SQLAttribute(name,type,property,size);
    table->appendAttribute(attribute);

    name = new string("Name");
    type = CHAR;
    property = 0;
    size = 20;
    attribute = new SQLAttribute(name,type,property,size);
    table->appendAttribute(attribute);

    name = new string("Age");
    type = INT;
    property = 0;
    size = sizeof(int);
    attribute = new SQLAttribute(name,type,property,size);
    table->appendAttribute(attribute);

    return table;
}
```

```
string getInputString()
{
    string cmd;
    char chrcmd[100];
    int index;
    index = 0;
    cin.get(chrcmd[index]);
    while (chrcmd[index]!=';') {
        index++;
        cin.get(chrcmd[index]);
    }
    cmd = string(chrcmd);
    return cmd;
}

int main(int argc, const char * argv[]) {

    SQLCommandInterpreter::setDelegate(new A());
    SQLCommand* command;
    while(1)
    {
        command =
SQLCommandInterpreter::transferStringToCommand(getInputString());
        if(command == NULL)
            cout << endl << "Failed!" << endl;
    }

    return 0;
}
```

二、测试命令与结果：

```
select * from student where ad ='asd';

        Transfer Faild:
                attributeName not exist in such table

        Failed!

        select * from xzc where as = 'xx;
        Transfer Faild:
                no table has this name

        Failed!

select * from student where SID = '31232';
insert into student values('31309','huangzhaoyang',5);
delete from student where SID = '31324';
create table student(
        name char(3),
        age int,
        sid char(5),
        primary key(sid));
 //运行成功
```

interpreter模块总结

1. 通过对buffer模块的精心设计和细致实现，使我完全理解了数据库管理软件中与用户命令的交互是如何实现的

2. interpreter模块架构了一层底层的数据结构，并使用了多态和委托机制，使我对于面向对象的变成思想有了更好的学习和更深刻的理解。

3. 为了与API模块能够良好结合，interpreter模块的接口设计完全独立，与外部其他模块的耦合性十分低，能够很好地结合和使用。