为BDB添加SQL支持

SQL语言解析器的实现

黄坚

2011-07

大纲

1

背景知识

- BerkeleyDB简介
- Lex和Yacc简介

2

为什么做本项目

- BerkeleyDB没有SQL支持
- 学习自己实现SQL解析器

3

规划

- 功能
- 实现方案
- 模块规划

4

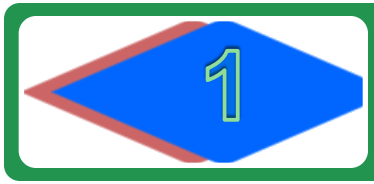
实现

- Lex文件编写
- Yacc文件编写
- 实现的代码编写

5

其他话题

- PostgreSQL的SQL解析学习
- Lex和Yacc的其他用途



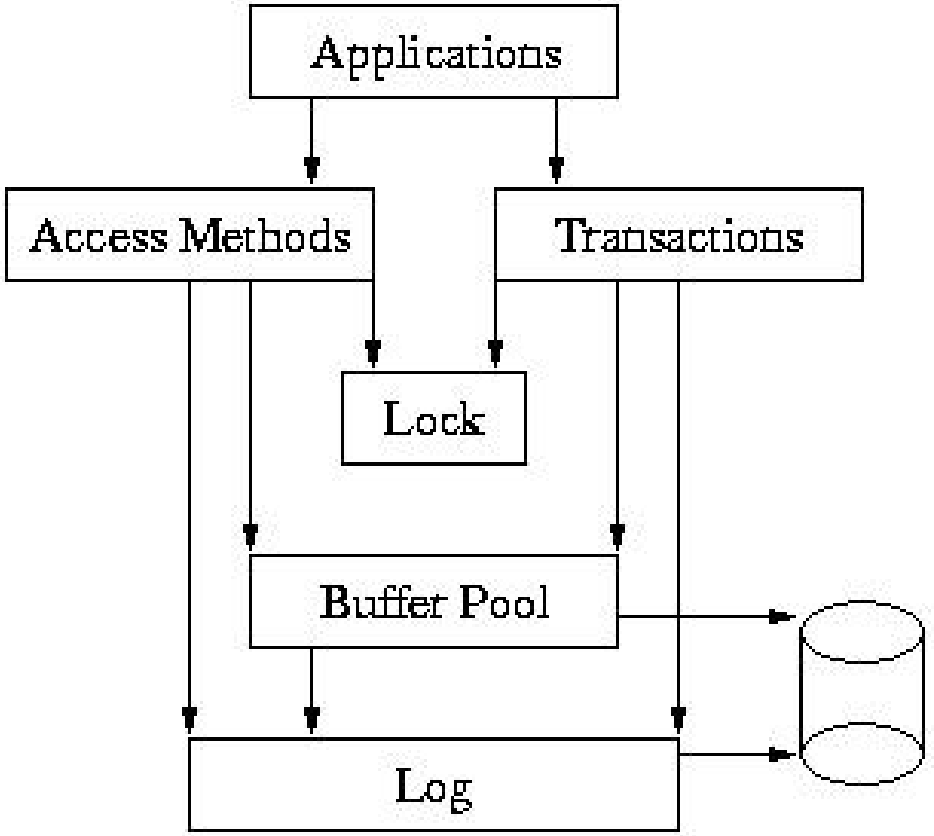
背景知识

- BerkeleyDB简介
- Lex和Yacc简介

BerkeleyDB简介

Berkeley DB最初开发的目的是以新的HASH访问算法来代替旧的hsearch函数和大量的dbm实现（如AT&T的dbm，Berkeley的 ndbm，GNU项目的gdbm），Berkeley DB的第一个发行版在1991年出现，当时还包含了B+树数据访问算法。在1992年，BSD UNIX第4.4发行版中包含了Berkeley DB1.85版。基本上认为这是Berkeley DB的第一个正式版。在1996年中期，Sleepycat软件公司成立，提供对Berkeley DB的商业支持。在这以后，Berkeley DB得到了广泛的应用，成为一款独树一帜的嵌入式数据库系统。2006年Sleepycat公司被Oracle 公司收购，Berkeley DB成为Oracle数据库家族的一员，Sleepycat原有开发者继续在Oracle开发Berkeley DB，Oracle继续原来的授权方式并且加大了对Berkeley DB的开发力度，继续提升了Berkeley DB在软件行业的声誉。Berkeley DB的当前最新发行版本是5.2（ Oracle Berkeley DB 11gR2 (11.2.5.2.28)）。

BerkeleyDB框架



BerkeleyDB简介

- 编程接口：
  - C
  - C++
  - C#
  - Java
  - TCL

BerkeleyDB C++编程接口简介

- C/C++编程接口是本示例的基础
- 接口分类
  - 打开/关闭环境
  - 打开/关闭数据库（表）
  - 关联索引
  - 对表的操作（插入、删除、查询数据、根据主键修改数据）

BDB代码示例（上）

```
#include <iostream>
#include <db_cxx.h>

using namespace std;

#define FILE_NAME     "1.data"
#define ENV_HOME     "/cdr2/home/alex/_EnvHome"
#define DB_NAME       "my_db.db"
#define SHM_ID        44444
#define ENV_FLAGS    DB_CREATE|DB_INIT_MPOOL|DB_SYSTEM_MEM|/*DB_INIT_LOCK|*/DB_INIT_LOG|DB_RECOVER|DB_INIT_TXN
#define ENV_RM      DB_USE_ENVIRON
#define DB_FLAGS    DB_CREATE

int main()
{
   int ret;
   try
   {
      // 创建Envirment
      cout << "Create envirment ... " << endl;
      DbEnv env (0);
      env.open (ENV_HOME, ENV_FLAGS, 0);
      // 创建Database
      cout << "Create database ... " << endl;
      Db db (&env, 0);
      db.open (NULL, DB_NAME, NULL, DB_BTREE, DB_FLAGS, 0);
      // 加载数据
      char value[1024+1];
      char key[1024+1];
      char * p;
      Dbt dbKey, dbVal;
      int iCount = 0;
```

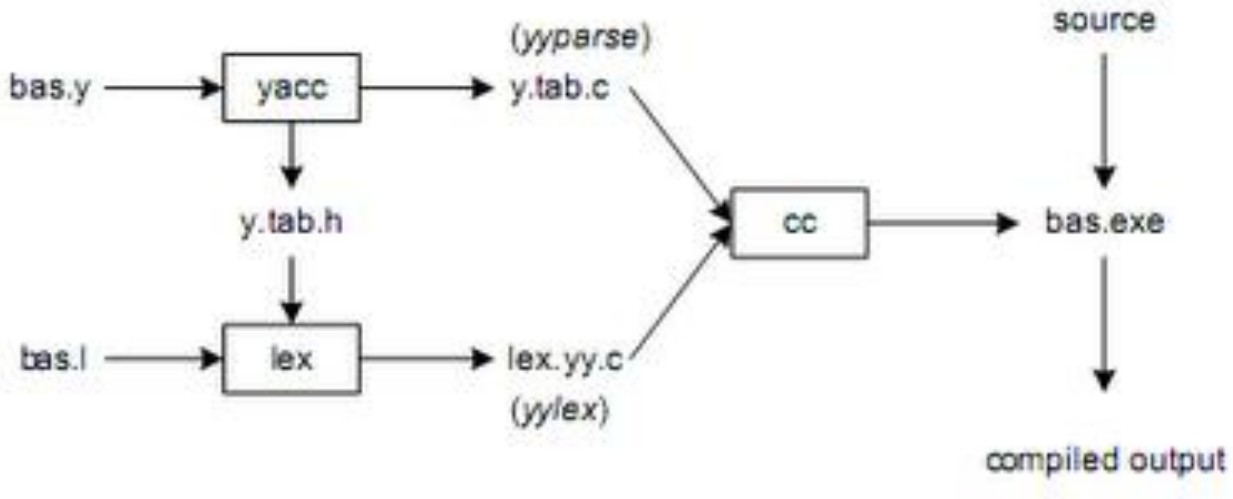
BDB代码示例（下）

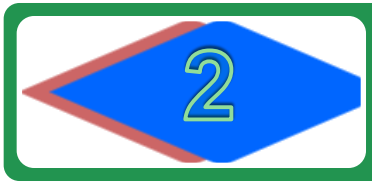
```
        cout << "Load data ... " << endl;
        FILE * file = fopen(FILE_NAME, "r");
        while (fgets(value, 1024, file) != NULL)
        {
            p = strchr(value, '|');
            strncpy(key, value, p-value);
            key[p-value] = 0;
            dbKey.set_data(key);
            dbKey.set_size(strlen(key)+1);
            dbVal.set_data(value);
            dbVal.set_size(strlen(value));
            db.put(NULL, &dbKey, &dbVal, DB_NOOVERWRITE);
            if (iCount++ % 10000 == 0)
            {
                cout << "Insert " << iCount << " records!" << endl;
            }
        }
        fclose(file);
        // 关闭Database
        cout << "Close Berkeley DB ... " << endl;
        db.close(0);
        // 关闭Envirment
        env.close(0);
        // env.remove (ENV_HOME, 0);
    }
    catch (DbException &e)
    {
        cout << "DBException #" << e.get_errno ()
            << " : " << e.what ()
            << endl;
    }
    catch(std::exception &e)
    {}
    return 0;
}
```

Lex和Yacc简介

- Lex 代表 Lexical Analyzar。Yacc 代表 Yet Another Compiler Compiler。
  - Lex 是一种生成扫描器的工具。扫描器是一种识别文本中的词汇模式的程序。
  - Lex 和 C 是强耦合的。一个 *.lex* 文件（Lex 文件具有 *.lex* 的扩展名）通过 lex 公用程序来传递，并生成 C 的输出文件。这些文件被编译为词法分析器的可执行版本。
  - 它是一种工具，将任何一种编程语言的所有语法翻译成针对此种语言的 Yacc 语 法解析器。
  - GNU提供了工具完成以上工作，分别是flex和bison

Lex和Yacc合作模式



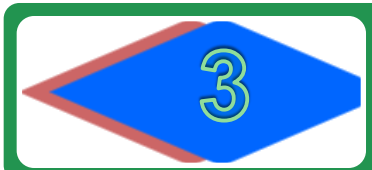


为什么做本项目

- BerkeleyDB没有SQL支持
- 学习自己实现SQL解析器

为什么做本项目

- BDB没有SQL支持
  - 4.5.x以前没有SQL支持，查询数据不方便
  - 现在依旧有大量的应用跑在较老的db上
- 学习做语法解析器
  - 学习Lex和Yacc
- 挑战自我
  - 一周内完成从零开始到基本可用



规划

- 功能
- 实现方案
- 模块规划

我们可以在一周内做到什么

- 提供方法存储系统信息
  - 存储数据库所在路径
  - 存储表结构
  - 存储索引等信息
- 支持的数据类型
  - INT
  - LONG
  - CHAR()
- 实现一个命令行的交互界面
  - 提示符
  - 描述数据库基本信息
  - 查询数据库中有哪些表
  - 描述表信息
  - 对单表进行增删改查

实现方案

- 命令行界面进行交互
  - 列出有哪些表
  - 描述表结构
  - 单表的操作（增加、删除、更改、查询）
  - 选择适当的索引
- 记录表结构等信息
  - 使用改造版本的SQL语句
  - 需要记录数据库所在目录信息
  - 需要记录表结构
  - 需要记录索引信息

数据库信息记录文件

内容如下：

```
CREATE DATABASE AT ./EnvTest DEFAULT DATAFILE bfs.db;

CREATE TABLE table1
{
    col1 LONG ,
    col2 INT  ,
    col3 CHAR(100),
    col4 LONG,
    PRIMARY KEY UNIQUE( col1, col2 )
} DATAFILE abc.db;

CREATE INDEX idx_col2 ON table1( col2 ) DATAFILE abc_idx.db;
```

支持的SQL语句

- DATABASE; 显示数据库信息
- TABLES; 列出所有的表
- DESC table_name; 显示表的结构信息
- SELECT *,col1 FROM table_name WHERE col2 = ‘abc’;
- INSERT INTO table_name (col1, col2) VALUES(1, ‘def’);
- UPDATE table_name SET col2 = ‘ccc’ WHERE col1 = 2
- DELETE FROM table_name WHERE col1 = 3;

实现

- Lex文件编写
- Yacc文件编写
- 实现的代码编写

Lex文件编写

- 第一部分第一段：定义段，填写一些需要引入的c的头文件、定义一些需要引入的类型，一些需要引用的函数定义。

```
%{
#include <stdlib.h>
#include <string.h>
#include "SqlElement.h"
#include "y.SQL.hpp"

void yyerror(char *);

#define yylval  SQLlval

%}
```

Lex文件编写

- 第一部分第二段：一组正则表达式定义和状态定义，定义拆词的规则等（本例中没有使用状态定义）。

```
%option prefix="SQL"            #这里给输出文件添加前缀

/* Macros */
letter           [a-zA-Z_.]
digit            [0-9]
hex              [0-9A-Fa-f]
letter_or_digit  [a-zA-Z_0-9.]
sign             [\+\-]
string_char      [\]\[\\\+\"=\)\(\{\}\*&\^\%\$#@!:;,\|\~\`\.\<\>\?\/a-zA-Z_0-9 \t\n\-]
white_space      [ \t\n]
blank            [ \t]
quota            \'\'
non_double_quota [^\"\n]
other            .

%%
```

Lex文件编写

- 第二部分：Lex 的模式匹配规则，需要匹配的正则表达式以及对应的处理函数。

```
SELECT                          { return SELECT;}
INSERT                          { return INSERT;}
UPDATE                          { return UPDATE;}
//注：这里不区分大小写，由命令行的-i参数觉得

\'({quota}|{string_char})*\'    {
    if(*yytext != '\'')
         printf("ERROR: wanted a string, got: %s\n",yytext);
    yytext[yyleng-1]='\0';
    yylval.strVal = strdup((const char*)yytext + 1);
    return (STRING);
}
```

Lex文件编写

- 第三部分：辅助函数断，这一段必须包括 yywrap() 函数。由于我们要配合Yacc使用，我们的代码仅仅是实现yywrap()函数，如下：

```
%%

int yywrap(void)
{
//  printf("Reach end of string.\n");
    return 1;
}
```

- 如果仅仅做词法分析，可以如下：

```
void main()
{
    yylex(); /* start the analysis*/
    // ... 参考IBM developerWorks上的统计词数的例子
}
int yywrap()
{ return 1; }
```

Lex文件编译成C程序

- 编译命令

- 看看config_common.h
- 看看生成的文件lex.config.cpp

Yacc文件编写

- 第一部分第一段：定义段，与Lex类似，将编译入C程序的定义段

```
%{
#include <stdio.h>
#include <vector>
#include "SQL_common.h"
#include "Util.h"
#include "ParseContext.h"

#define YYPARSE_PARAM SQLctx

extern char *SQLtext;

void yyerror(const char *s);
int yylex(void);

#define NOT_SUPPORT( tag )       \
    {                                                                    \
           printf("Error: %s not supported now!!!\n", tag );\
           YYABORT;   \
    }
%}
```

Yacc文件编写

❖ 第一部分第二段：定义段，声明各种类型

```
%token <intVal> SELECT INSERT UPDATE DELETE
%token TABLE

%left '-' '+'
%left '*' '/' '%'
%right NOT

%type <strVal> IDENT
%type <strVal> table_name field_name alias_name

%type <pValue> data_value direct_data_int direct_data_float direct_data_str
%type <longVal> LONG
%type <fltVal> FLOAT
%type <strVal> STRING
%type <Expression> expression select_item
%type <Condition>  condition where_clause

%type <intVal> AND OR
%type <intVal> comp_op EQ GE GT LE LT NE

%type <UpdateItem> update_set_item
%type <UpdateItemList> update_set_list
```

Yacc文件编写

- 第二部分：语法规则

```
select_stat:
    SELECT select_val_list FROM result_set_list where_clause
    {
          CSQLContext *Context = (CSQLContext *)SQLctx;
          Context->nOpType = SELECT;
          Context->ResultSet.SetSelectStatement( *$2, *$4, $5 );
          delete $2;
          delete $4;
    }
    ;
where_clause:
    // Empty
    {
          $$ = NULL;
    }
    | WHERE condition
    {
          $$ = $2;
    }
    ;
```

Yacc文件编写

- 第二部分：语法规则（2）

```
condition:
    expression comp_op expression
    {
            $$ = new CCondition();
            $$->SetExpression( *$1 ,*$3, $2 );
            delete $1;
            delete $3;
    }
    | '(' condition ')'
    {
            $$ = new CCondition();
            $$->SetSubCondition( $2 );
    }
    | condition AND condition
    {
            $1->SetAndCond( $3 );
    }
    | condition OR condition
    {
            $1->SetOrCond( $3 );
    }
    | NOT condition
    {
            if ( ! $2->SetHasNot() )
            {
                    SQLerror( "Can't NOT on one condition twice" );
                    YYABORT;
            }
            $$ = $2;
    }
    ;
```

Yacc文件编写

- 第三段：辅助函数断，只实现了一个打印出错信息的函数：

```
void SQLerror(const char *s)
{
    fprintf( stdout, "Error: %s: %s\n", s, SQLtext );
}
```

语法解析之外的主要代码

- SqlElement.h, SqlElement.cpp
  - 处理语法分析树
  - SQL执行的实现
- InputHandler.h,InputHandler.cpp
  - 工具交互界面（结合readline库）

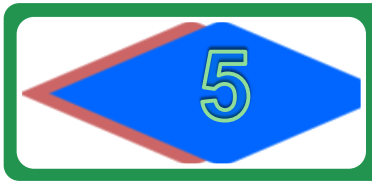
将以上内容组合起来

- Makefile

```
lex.SQL.cpp: SQL.l y.SQL.hpp
    $(LEX) -P SQL -i --header-file=SQL_common.h -o lex.SQL.cpp SQL.l
y.SQL.hpp y.SQL.cpp: SQL.y SQL_common.h
    $(YACC) -y -p SQL -d -o y.SQL.cpp SQL.y
```

运行效果

```
./sql4db.exe -s database.sql

IM*SQL: Release 1.0.0 - Production on Sat, Jul 02, 2011 12:44:22 PM
Copyright (c) 1996, 2009, Sunrise. All Rights Reserved.

Connected to:
        ./EnvTest

SQL> tables;

TABLE1
ACCOUNT

SQL> desc account;

Real table name: account
Data file name: bfs.db

Column(s):
 Name                                    Type                         PK UNIQUE
 --------------------------------------- ---------------------------- -- ------
 ACCT_ID                                 LONG                         *  *
 CUST_ID                                 LONG
 UPDATE_DATE                             LONG
 EXPIRE_DATE                             LONG
 STATUS                                  INT
 ACCT_TYPE                               INT

Index(s):
 Name                    Field(s)
 ----------------------- ----------------------------------
 idx_account_cust_id     CUST_ID

SQL> select * from account where acct_id = 1;

ACCT_ID CUST_ID UPDATE_DATE     EXPIRE_DATE     STATUS  ACCT_TYPE

Primary key access
1       1       20110123        20120123        0       1

1 line(s) get.
SQL>
```



其他话题

- PostgreSQL的SQL解析学习
- Lex和Yacc的其他用途

PostgreSQL的SQL解析

- PostgreSQL的SQL解析位于服务端，psql程序只负责解释以”\”开始的命令
- SQL语言的解析代码位于：

  postgresql-9.0.4\src\backend\parser，

  lex文件：scan.l

  Yacc文件：gram.y
- 其他的语法解析器代码：psql等

其他用途

- 特殊的配置文件