mysql connect

mysql的基础，我们之前已经学过，后面我们只关心使用

要使用C语言连接mysql，需要使用mysql官网提供的库，大家可以去官网下载

我们使用C接口库来进行连接

要正确使用，我们需要做一些准备工作：

> - 保证mysql服务有效
> - 在官网上下载合适自己平台的mysql connect库，以备后用

Connector/C 使用

我们下下来的库格式如下：

```
[hb@MiWiFi-R1CL-srv lib]$ tree .
.
├── include
│   ├── big_endian.h
│   ├── byte_order_generic.h
│   ├── byte_order_generic_x86.h
│   ├── decimal.h
│   ├── errmsg.h
│   ├── keycache.h
│   ├── little_endian.h
│   ├── m_ctype.h
│   ├── m_string.h
│   ├── my_alloc.h
│   ├── my_byteorder.h
│   ├── my_compiler.h
│   ├── my_config.h
│   ├── my_dbug.h
│   ├── my_dir.h
│   ├── my_getopt.h
│   ├── my_global.h
│   ├── my_list.h
│   ├── my_pthread.h
│   ├── mysql
│   │   ├── client_authentication.h
│   │   ├── client_plugin.h
│   │   ├── client_plugin.h.pp
│   │   ├── get_password.h
│   │   ├── plugin_auth_common.h
│   │   ├── plugin_trace.h
│   │   ├── psi
│   │   │   ├── mysql_file.h
│   │   │   ├── mysql_idle.h
│   │   │   ├── mysql_mdl.h
│   │   │   ├── mysql_memory.h
│   │   │   ├── mysql_ps.h
│   │   │   ├── mysql_socket.h
│   │   │   ├── mysql_sp.h
│   │   │   ├── mysql_stage.h
```

```
|   |   |   ├── mysql_statement.h
|   |   |   ├── mysql_table.h
|   |   |   ├── mysql_thread.h
|   |   |   ├── mysql_transaction.h
|   |   |   ├── psi_base.h
|   |   |   ├── psi.h
|   |   |   └── psi_memory.h
|   |   ├── service_my_snprintf.h
|   |   └── service_mysql_alloc.h
|   ├── mysql_com.h
|   ├── mysql_com_server.h
|   ├── mysqld_ername.h
|   ├── mysqld_error.h
|   ├── mysql_embed.h
|   ├── mysql.h
|   ├── mysql_time.h
|   ├── mysql_version.h
|   ├── my_sys.h
|   ├── my_xml.h
|   ├── sql_common.h
|   ├── sql_state.h
|   ├── sslopt-case.h
|   ├── sslopt-longopts.h
|   ├── sslopt-vars.h
|   └── typelib.h
└── lib
    ├── libmysqlclient.a
    ├── libmysqlclient_r.a -> libmysqlclient.a
    ├── libmysqlclient_r.so -> libmysqlclient.so
    ├── libmysqlclient_r.so.18 -> libmysqlclient.so.18
    ├── libmysqlclient_r.so.18.3.0 -> libmysqlclient.so.18.3.0
    ├── libmysqlclient.so -> libmysqlclient.so.18
    ├── libmysqlclient.so.18 -> libmysqlclient.so.18.3.0
    └── libmysqlclient.so.18.3.0
```

其中`include`包含所有的方法声明，`lib`包含所有的方法实现（打包成库）

尝试链接mysql client

通过`mysql_get_client_info()`函数，来验证我们的引入是否成功

```
#include <stdio.h>
#include <mysql.h>

int main()
{
    printf("mysql client Version: %s\n", mysql_get_client_info());
    return 0;
}

[hb@MiWiFi-R1CL-srv lib]$ gcc -o test test.c -I./include -L./lib -lmysqlclient
[hb@MiWiFi-R1CL-srv lib]$ ls
include  lib  test  test.c

[hb@MiWiFi-R1CL-srv lib]$ ./test
./test: error while loading shared libraries: libmysqlclient.so.18: cannot open
shared object file: No such file or directory
```

```
[hb@MiWiFi-R1CL-srv lib]$ export LD_LIBRARY_PATH=./lib #动态库查找路径,讲解ldd命令
[hb@MiWiFi-R1CL-srv lib]$ ./test
mysql client Version: 6.1.6
```

至此引入库的工作已经做完，接下来就是熟悉接口

mysql接口介绍

> - 初始化mysql_init()

要使用库，必须先进行初始化!

`MYSQL *mysql_init(MYSQL *mysql);`

如：`MYSQL *mfp = mysql_init(NULL)`

> - 链接数据库mysql_real_connect

初始化完毕之后，必须先链接数据库，在进行后续操作。（mysql网络部分是基于TCP/IP的）

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host,
                          const char *user,
                          const char *passwd,
                          const char *db,
                          unsigned int port,
                          const char *unix_socket,
                          unsigned long clientflag);

//建立好链接之后，获取英文没有问题，如果获取中文是乱码：
//设置链接的默认字符集是utf8，原始默认是latin1
mysql_set_character_set(myfd, "utf8");
```

第一个参数 MYSQL是 C api中一个非常重要的变量（mysql_init的返回值），里面内存非常丰富，有port,dbname,charset等连接基本参数。它也包含了一个叫 st_mysql_methods的结构体变量，该变量里面保存着很多函数指针，这些函数指针将会在数据库连接成功以后的各种数据操作中被调用。mysql_real_connect函数中各参数，基本都是顾名思意。

> - 下发mysql命令mysql_query

```
int mysql_query(MYSQL *mysql, const char *q);
```

第一个参数上面已经介绍过，第二个参数为要执行的sql语句,如“select * from table”。

> - 获取执行结果mysql_store_result

sql执行完以后，如果是查询语句，我们当然还要读取数据，如果update，insert等语句，那么就看下操作成功与否即可。我们来看看如何获取查询结果： 如果mysql_query返回成功，那么我们就通过mysql_store_result这个函数来读取结果。原型如下：

```
MYSQL_RES *mysql_store_result(MYSQL *mysql);
```

该函数会调用MYSQL变量中的st_mysql_methods中的 read_rows 函数指针来获取查询的结果。同时该函数会返回MYSQL_RES 这样一个变量，该变量主要用于保存查询的结果。同时该函数malloc了一片内存空间来存储查询过来的数据，所以我们一定要记的 free(result),不然是肯定会造成内存泄漏的。 执行完mysql_store_result以后，其实数据都已经在MYSQL_RES 变量中了，下面的api基本就是读取MYSQL_RES 中的数据。

- 获取结果行数mysql_num_rows

```
my_ulonglong mysql_num_rows(MYSQL_RES *res);
```

- 获取结果列数mysql_num_fields

```
unsigned int mysql_num_fields(MYSQL_RES *res);
```

- 获取列名mysql_fetch_fields

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *res);
```

如:

```
int fields = mysql_num_fields(res);
MYSQL_FIELD *field = mysql_fetch_fields(res);
int i = 0;
for(; i < fields; i++){
    cout<<field[i].name<<" ";
}
cout<<endl;
```

- 获取结果内容mysql_fetch_row

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result);
```

它会返回一个MYSQL_ROW变量，MYSQL_ROW其实就是char **.就当成一个二维数组来用吧。

```
i = 0;
MYSQL_ROW line;
for(; i < nums; i++){
    line = mysql_fetch_row(res);
    int j = 0;
    for(; j < fields; j++){
        cout<<line[j]<<" ";
    }
    cout<<endl;
}
```

- 关闭mysql链接mysql_close

```
void mysql_close(MYSQL *sock);
```

另外，mysql C api还支持事务等常用操作，大家下来自行了解:

```
my_bool STDCALL mysql_autocommit(MYSQL * mysql, my_bool auto_mode);
my_bool STDCALL mysql_commit(MYSQL * mysql);
my_bool STDCALL mysql_rollback(MYSQL * mysql);
```