C++课程设计任务指导书

（2021/2022 学年第一学期）

指导教师： 庄巧莉、霍旭文

任务说明

本次课程设计每人独立完成任务 1 或任务 2 中的一项任务即可，自由选择。

课程设计综合评分由 3 部分组成，平时成绩+答辩成绩+报告成绩。其中，平时成绩由到课率和中期检查两部分成绩组成。中期检查主要检查课程设计报告中的业务流程图、功能结构图和类的设计三部分的完成情况。答辩成绩由功能的完成度、界面的交互性、代码的规范度和类设计的合理性四部分组成。功能的完成度评分标准主要根据任务书中给出的阶梯式任务点的完成情况来评定。需要特别注意的是，除了第 1 个任务点，每个任务点的任务都包含了它前一个任务点的任务。例如第 3 个任务点为 80 分以上的任务，则必须同时完成第 1、第 2 和第 3 个任务点的任务，才能拿到功能完成度上 80 分以上的成绩。

课程设计报告成绩主要由报告的规范性决定，课程设计报告的评分标准如下所示：

优秀（>90）	良好(80~89)	中等(70~79)	一般(60~69)	不合格(<60)
内容完整，设计合理，制图规范，文字清晰流畅，无错别字，格式完全符合要求，有详细的分析和总结	内容较完整，设计较合理，制图规范，文字清晰流畅，无错别字，格式符合要求，有分析和总结	内容较完整，设计基本合理，制图基本规范，文字基本流畅，少错别字，格式基本符合要求，有分析和总结	内容基本完整，设计欠合理，制图基本规范，文字基本流畅，少错别字，格式基本符合要求，无分析和总结	内容不完整，设计不合理，制图不规范，文字欠流畅，错别字较多，格式不符合要求，无分析和总结

任务书 1

【题目】自助点餐系统

【目的】

通过设计一个小型的自助点餐系统，训练综合运用所学知识处理实际问题的能力，强化面向对象的程序设计理念，使自己的程序设计与调试水平有一个明显的提高。

【要求】

1、每个学生必须独立完成；

2、课程设计时间为 1 周分散进行；

3、设计语言采用 C++，程序设计方法必须采用面向对象的程序设计方法；

4、课程设计期间，无故缺席按旷课处理；缺席时间达四分之一以上者，未按规定上交实验报告的学生，其成绩按不及格处理。

【内容简介】

有一个小型餐厅，现在这个餐厅打算使用自助点餐系统，方便顾客自己点餐，并提供对餐厅销售情况的统计和管理功能。

【考核标准】

该系统为两种角色的用户提供服务，一种是餐厅管理员，一种是顾客。餐厅管理员根据账号、密码登录系统。顾客无需登录即可使用系统。

1、 顾客通过该餐厅在系统中提供的菜单为自己点餐。系统能够根据顾客的要求正确打出订单，订单内容包括订单编号、菜品名称、每个菜品的价格、份数、折扣等；订单分两种，一种是在店消费，在店消费要求包括餐桌号，是否有包厢费，另一种是外卖，外卖要求包括送餐时间，送餐地点，客户手机号，外卖服务费，成绩≥60；

2、 订单、用户信息保存在数据库中，其中，连接数据库所需信息（数据库服务器地址、用户名、密码、数据库名）存放在文件中，程序通过从文件中读取这些信息获得与数据库的连接。餐厅管理员可以根据订单编号或手机号查找、删除或修改某个订单，查询到的订单按照下单时间先后显示，成绩≥70；

3、 菜单信息保存在数据库中，能够实现对餐厅菜式和价格的管理，包括对菜品和对应价格的增加、修改、删除、查找，折扣的设置，设置后，顾客在点餐时看到的是新设置后的菜单，成绩≥80；

4、 系统可根据数据库中保存的订单记录对销售情况进行统计，根据餐厅管理员的输入日期统计某天的销售情况（包括一共接了多少单，销售额是多少，各个菜品的销售情况，外卖和在店销售的占比）并显示订单详情，成绩≥90；

要求：

数据库管理系统自由选择，推荐使用 MySQL 数据库。对图形化 UI 接口有兴趣的同学推荐选用 Qt 进行开发，使用方法可参考 https://blog.csdn.net/qq_46018418/article/details/107919933，不做强制性要求。

用面向对象的程序设计方法设计该系统。本系统涉及的基本对象有订单对象（包括外卖订单和在店消费订单，可以考虑采用继承机制，提高代码的重用性）、订单管理对象、菜单对象、菜品对象、菜品管理对象、界面对象等。实现对这些对象的合理抽象和封装，正确定义类之间的关系。界面合理，代码文件组织清晰，命名符合规范，代码注释清楚，课设报告书质量高。

任务书 2

【题目】校园二手书交易系统

【目的】

通过设计一个小型的校园二手书交易系统，训练综合运用所学知识处理实际问题的能力，强化面向对象的程序设计理念，使自己的程序设计与调试水平有一个明显的提高。

【要求】

1、每个学生必须独立完成；

2、课程设计时间为 1 周；

3、设计语言采用 C++；

4、学生有事离校必须请假。课程设计期间，无故缺席按旷课处理；缺席时间达四分之一以上者，未按规定上交实验报告的学生，其成绩按不及格处理。

【内容简介】

为了使同学们手中闲置的书籍能再次发挥其价值，现在需要你针对在校学生开发一款小型的二手书交易系统。每个同学既可以是卖方，也可以是买方。该系统能帮助卖方同学发布需要售卖的二手书信息，帮助买方同学购买到所需书籍，实现校园内部的便捷交易。

【考核标准】

该系统为两种角色的用户提供服务，一种是卖方，一种是买方。卖方和买方均根据账号、密码登录系统。

1、卖方可将需要售卖的书籍信息录入系统，书籍信息包括 ISBN 编号、书籍名称、作者、出版社、价格、新旧程度、取书方式等。买方可查看正在售卖的所有书籍信息，并能够选择购买某一书籍，完成交易，成绩≥60；（注：买方或者卖方必须能够通过订单信息获得对方的收获地址或者联系方式才能完成

交易）

2、 买方、卖方信息均保存在数据库中，其中，连接数据库所需信息（数据库服务器地址、用户名、密码、数据库名）存放在文件中，程序通过从文件中读取这些信息获得与数据库的连接。实现用户的注册功能，注册信息包括用户名、密码、角色（买方或卖方）、联系地址、联系电话等，成绩≥70；

3、 书籍信息和订单信息保存在数据库中，卖方能够对要售卖的书籍进行管理，包括书籍的增加、删除、修改和查询，成绩≥80；

4、 系统可对数据库中存储的交易订单进行查询和统计，买方同学能够查看到所有购买的订单数量和订单信息，卖方同学能够查看到所有售出的订单数量和订单信息，成绩≥90；

要求：

数据库管理系统自由选择，推荐使用 MySQL 数据库。对图形化 UI 接口有兴趣的同学推荐选用 Qt 进行开发，使用方法可参考 https://blog.csdn.net/qq_46018418/article/details/107919933，不做强制性要求。

用面向对象的程序设计方法设计该系统。本系统涉及的基本对象有买方对象、卖方对象、书籍对象、书籍管理对象、订单对象、界面对象等。实现对这些对象的合理抽象和封装，正确定义类之间的关系。界面合理，代码文件组织清晰，命名符合规范，代码注释清楚，课设报告书质量高。

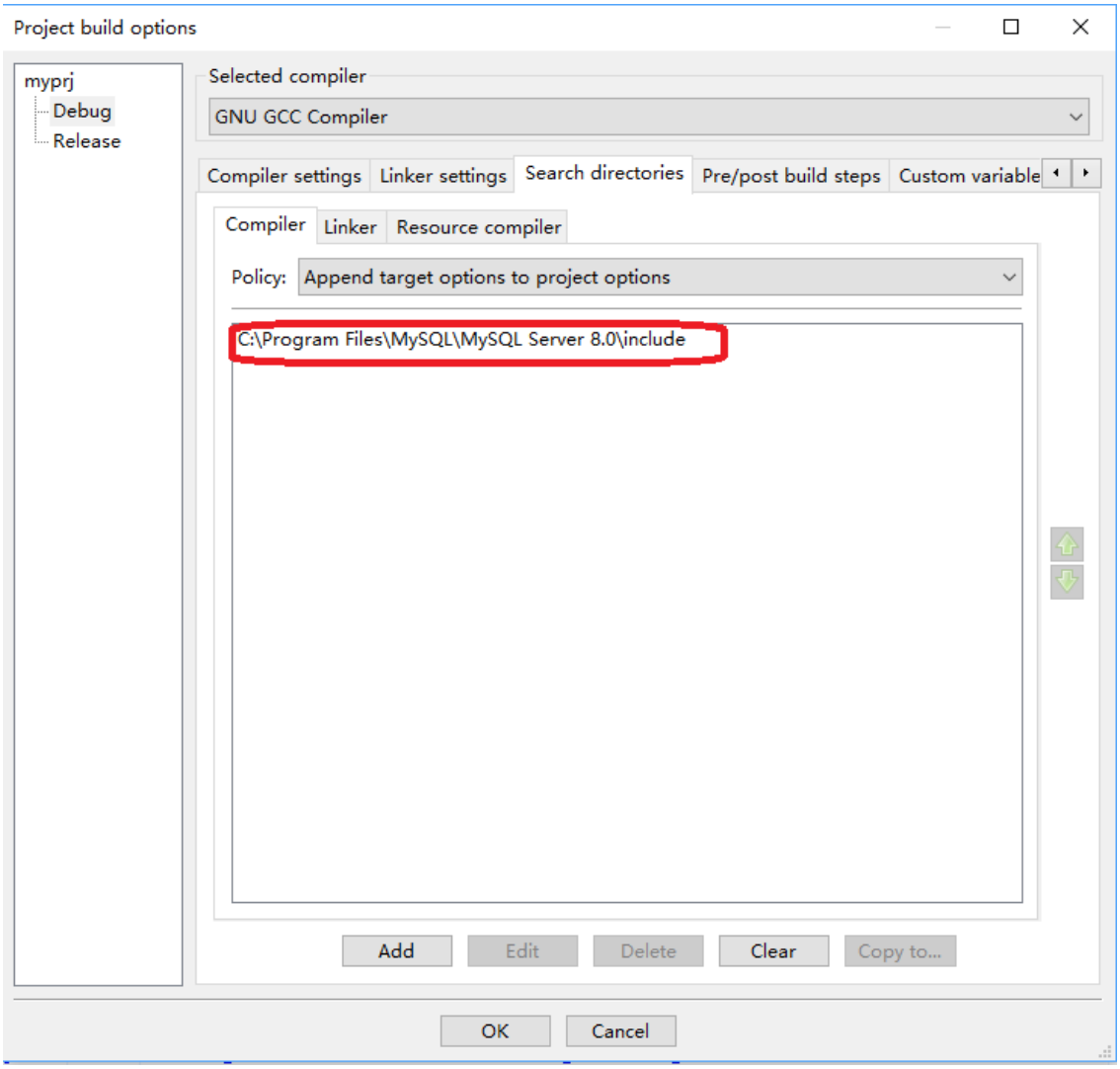
如何编写C++程序实现对MySQL数据库的访问

注意：此文档适用数据库版本为*MySQL 8.0*

1. 使用Codeblocks连接MySQL

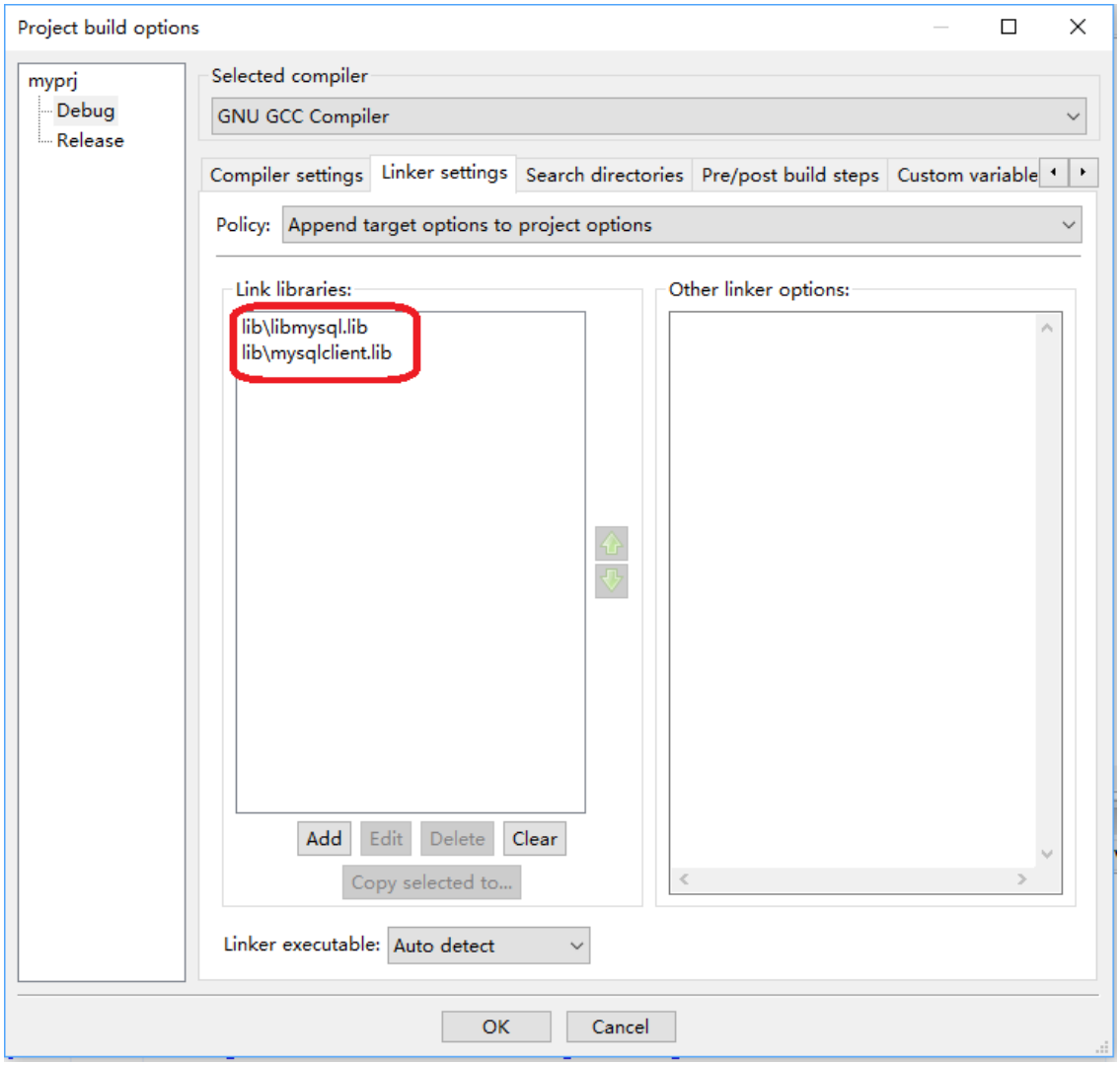
- **引入mysql.h头文件**

  选择Project->Build Options菜单项，在Search directories属性页中设置MySQL的C++接口头文件mysql.h的路径，mysql.h的默认路径在MySQL的安装路径下的include文件夹下。



- **为项目添加MySQL C链接库**

  选择Project->Build Options菜单项，在Linker settings属性页中设置MySQL的C++链接库libmysql.lib和mysqlclient，libmysql.lib和mysqlclient在MySQL的安装路径下的lib文件夹下，可以拷贝出来放到项目目录项然后在设置。



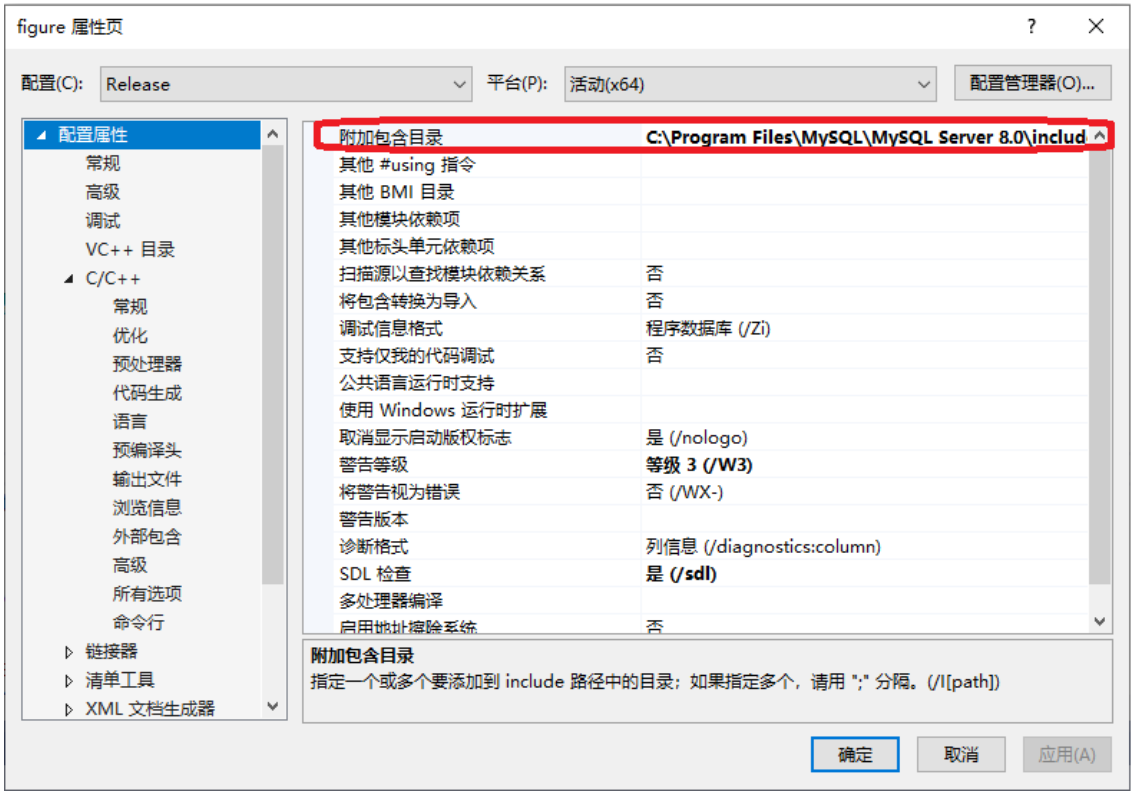
- **导入动态链接**

  将MySQL的动态链接库libmysql.dll放到项目可执行程序的运行目录下，libmysql.dll在MySQL的安装路径下的lib文件夹下

2. 使用Visual Studio连接MySQL

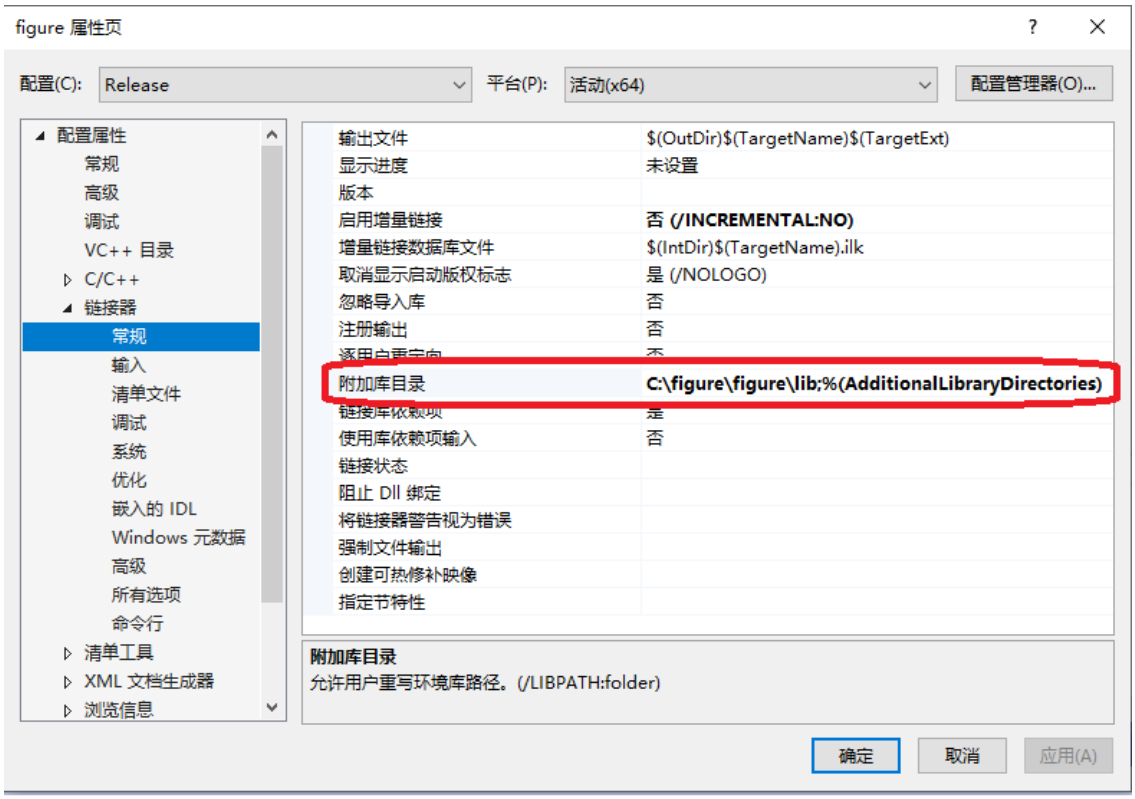
- **引入mysql.h头文件**

  选择项目—> 属性页—> 配置属性 —> C/C++ —> 常规 —>附加包含目录，设置MySQL的C++接口头文件mysql.h的路径，mysql.h的默认路径在MySQL的安装路径下的include文件夹下。

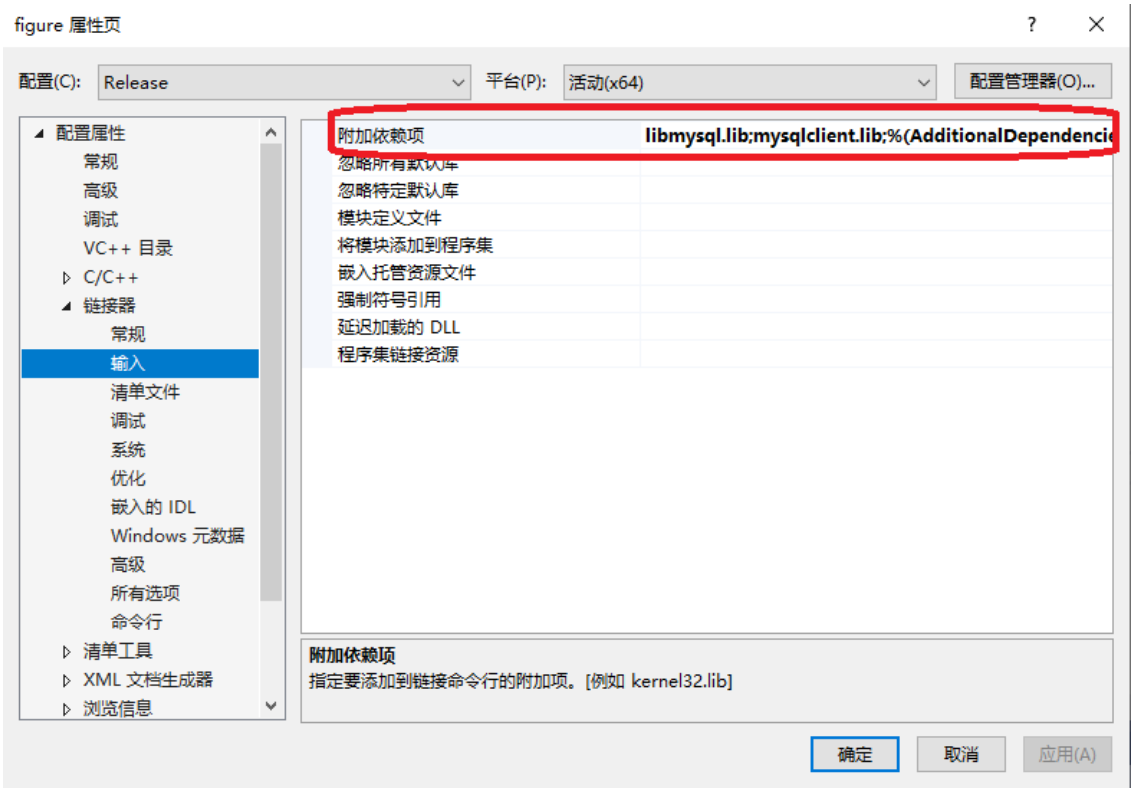


- **为项目添加MySQL C链接库**

选择项目—> 属性页—> 配置属性 —> 链接器 —> 常规 —> 附加库目录，设置MySQL的C++链接库libmysql.lib和mysqlclient所在目录，libmysql.lib和mysqlclient在MySQL的安装路径下的lib文件夹下，可以拷贝出来放到项目目录项然后在设置。



选择项目—> 属性页—> 配置属性 —> 链接器 —> 输入 —> 附加依赖项，设置MySQL的C++链接库libmysql.lib和mysqlclient。



- **导入动态链接**

  将MySQL的动态链接库libmysql.dll放到项目可执行程序的运行目录下，libmysql.dll在MySQL的安装路径下的lib文件夹下。

  注意：如果在上述设置后出现找不到libssl-1_1-x64.dll”的错误信息，请参考博文https://blog.csdn.net/tianya_lu/article/details/115674442

3. 连接达梦数据库

https://www.cnblogs.com/EricShen/p/13638706.html

4. MySQL的C程序接口

- **官方技术文档**

  https://dev.mysql.com/doc/c-api/8.0/en/c-api-function-reference.html

  基础接口文档

  https://dev.mysql.com/doc/c-api/8.0/en/c-api-basic-interface.html

- **基本程序框架**

  1. 通过调用mysql_init（）初始化连接处理程序，并通过调用连接建立函数如mysql_real_connect（）连接到服务器；
  2. 调用mysql_query（）发出SQL语句，调用mysql_fetch_row（）获得结果集后处理其结果；
  3. 通过调用mysql_close（）关闭与MySQL服务器的连接；

- **基本函数用法**

  - 初始化适用于MySQL的数据库连接对象

```
MYSQL *mysql_init(MYSQL *mysql)
```

如果参数mysql是空指针，则函数分配、初始化并返回一个新对象；否则，将初始化对象并返回对象的地址。如果内存不足，无法分配新对象，则返回NULL。

MYSQL结构体用于表示一个数据库连接的处理程序，不要试图复制MYSQL结构。

用法：

```
MYSQL connection;
mysql_init(&my_connection);
```

- 建立MySQL连接

```
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 client_flag)
```

host：数据库服务器地址；user：数据库连接用户名； passwd：数据库连接密码；db：数据库名；port：端口号；unix_socket：指定要使用的套接字或命名管道；client_flag：通常为0，但也可以通过设置启用某些功能，具体见技术文档。

如果连接失败，则返回NULL，否则连接成功，返回数据库连接。

用法：

```
const char* host = "localhost";
const char* username = "root";
const char* password = "123456";
const char* database = "ecommerce";
mysql_real_connect(&connection, host, username, password, database, 3306,
nullptr, 0);
```

- 执行SQL命令

```
int mysql_query(MYSQL *mysql,
                const char *stmt_str)
```

执行以'\0'结尾的字符串stmt_str，通常情况下，stmt_str由单个SQL语句组成。

如果查询成功，mysql_query返回0，否则查询失败，返回错误代码如下：

CR_COMMANDS_OUT_OF_SYNC：命令执行顺序不正确

CR_SERVER_GONE_ERROR：MySQL服务器失效

CR_SERVER_LOST：与服务器的连接在查询过程中丢失

CR_UNKNOWN_ERROR：未知错误

用法：

```
char* sql = "select * from t_user";
mysql_query(&my_connection, sql);
```

- 获取查询结果

```
MYSQL_RES *mysql_store_result(MYSQL *msql)
```

函数返回指向包含结果的MYSQL_RES的指针。如果语句未返回结果集或发生错误，则为NULL。要确定是否发生错误，请检查mysql_error( )是否返回非空字符串，mysql_errno( )是否返回非零，或者mysql_field_count( )是否返回零。

MYSQL_RES定义如下：

用法：

```
MYSQL_RES res_ptr = mysql_store_result(&my_connection);
```

- 取结果列数

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

函数返回结果集中的列数。

用法：

```
int column = mysql_num_fields(res_ptr);
```

- 取结果行数

```
uint64_t mysql_num_rows(MYSQL_RES *result)
```

函数返回结果集中的行数

用法：

```
int row = mysql_num_rows(res_ptr);
```

- 按行返回查询信息

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

函数返回结果集的下一行。MYSQL_ROW是一个字符串数组，每一个元素是对应的属性值，下标从0开始，可以通过遍历每一个元素读取结果集中的每一行的每一列的属性值。

MYSQL_ROW定义如下：

```
typedef char **MYSQL_ROW;
```

用法：

```
MYSQL_ROW row;
unsigned int num_fields = mysql_num_fields(res_ptr);
while ((row = mysql_fetch_row(res_ptr)))
{
   for(unsigned int i = 0; i < num_fields; i++)
   {
       printf("[%s] ", row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

- 获取影响结果的行数

```
uint64_t mysql_affected_rows(MYSQL *mysql)
```

如果是UPDATE、DELETE或INSERT语句，则返回上一条语句更改、删除或插入的行数，对于SELECT语句，mysql_infected_rows（）的工作方式与mysql_ num_rows（）类似。

用法：

```
uint64_t num_rows = mysql_affected_rows(&connection);
```

- 获取表字段名

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

函数返回包含表字段信息的MYSQL_FIELD类型的数组，字段信息包括字段的名称（name）、类型和大小。可通过不断

MYSQL_FIELD定义如下：

```
typedef struct MYSQL_FIELD {
  char *name;                 /* Name of column */
  char *org_name;             /* Original column name, if an alias */
  char *table;                /* Table of column if column was a field */
  char *org_table;            /* Org table name, if table was an alias */
  char *db;                   /* Database for table */
  char *catalog;              /* Catalog for table */
  char *def;                  /* Default value (set by mysql_list_fields) */
  unsigned long length;       /* Width of column (create length) */
  unsigned long max_length; /* Max width for selected set */
  unsigned int name_length;
  unsigned int org_name_length;
  unsigned int table_length;
  unsigned int org_table_length;
  unsigned int db_length;
  unsigned int catalog_length;
  unsigned int def_length;
  unsigned int flags;         /* Div flags */
  unsigned int decimals;      /* Number of decimals in field */
  unsigned int charsetnr;     /* Character set */
  enum enum_field_types type; /* Type of field. See mysql_com.h for types */
  void *extension;
} MYSQL_FIELD;
```

用法：

```
MYSQL_FIELD *fields;
unsigned int num_fields = mysql_num_fields(res_ptr);
fields = mysql_fetch_fields(res_ptr);
for(unsigned int i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

- 获取MySQL API函数调用错误信息

```
const char *mysql_error(MYSQL *mysql)
```

对于mysql指定的连接，mysql_error（）返回一个以null结尾的字符串，其中包含最近调用的失败API函数的错误消息。如果函数没有失败，mysql_error（）的返回值可能是前一个错误，也可能是表示没有错误的空字符串。

用法：

```
char *error = mysql_error(&connection));
```

- 关闭连接

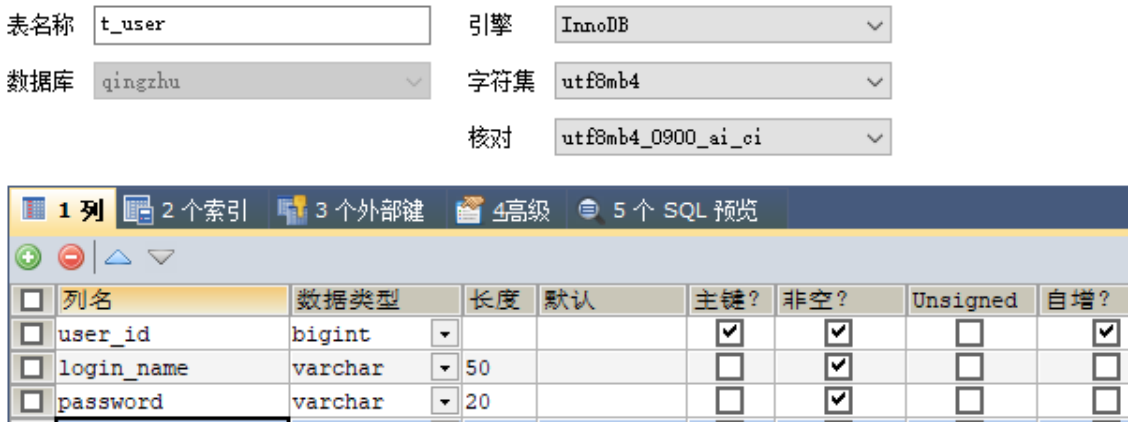
```
void mysql_close(MYSQL *mysql)
```

关闭打开的数据库连接。关闭后，mysql将不可用。

用法：

```
mysql_close(&connection);
```

- **操作实例**
  - **表结构**



  - **数据库连接类**

```
#include "mysql.h"
class DbConnection{
public:
    DbConnection(const char*, const char*, const char*, const char*);
```

```
    ~DbConnection();
    MYSQL_RES executeQuery(const char* sql);//查询成功，返回结果集，失败，抛出异常
    int executeSQL(const char* sql);
private:
    MYSQL connection;
};
DbConnection::DbConnection(const char* host, const char* username, const
char* password, const char* database)
{
     //初始化MySQL连接对象
     mysql_init(&connection);
     //建立MySQL连接
     //连接成功
     if(NULL == mysql_real_connect(&connection, host, username, password,
database, 3306, nullptr, 0))
        throw DbExcepiton(mysql_error(&connection));
    //设置查询编码为gbk，以支持中文
    mysql_query(&connection, "set names gbk");
}
DbConnection::~DbConnection(){
    mysql_close(&connection);
}
MYSQL_RES  DbConnection::executeQuery(const char* sql)
{
    //执行SQL命令
    int res = mysql_query(&connection, sql);

    if(res) //执行失败
        throw DbExcepiton(mysql_error(&connection));
    else
    {
        //获取结果集
        MYSQL_RES* res_ptr = mysql_store_result(&connection);
        //如果获取成功
        if(res_ptr)
            return *res_ptr;
        else throw DbExcepiton(mysql_error(&connection));
    }
}
 int DbConnection::executeSQL(const char* sql)
 {
    //执行SQL命令
    int res = mysql_query(&connection, sql);

    if(res) //执行失败
        throw DbExcepiton(mysql_error(&connection));
    else
    {
        return mysql_affected_rows(&connection);
    }
 }
```

- **数据库异常类**

```
#include<exception>
using namespace std;
class DbExcepiton: public exception
{
public:
    DbExcepiton(const char* msg):err_msg(msg){}
    const char* what() const throw(){return err_msg;}
private:
    const char* err_msg;
};
```

- **select操作**

```
#include<iostream>
#include"mysql.h"
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
     const char* host = "localhost";
     const char* username = "root";
     const char* password = "123456";
     const char* database = "encommerce";
     try{
         DbConnection db(host, username, password, database);
         //执行select命令，获取结果集
         char *sql;
         sql = "select * from t_user";
         MYSQL_RES res = db.executeQuery(sql);

         //输出字段名
         unsigned int num_fields = mysql_num_fields(&res);
         MYSQL_FIELD* fields = mysql_fetch_fields(&res);
         for(unsigned int i = 0; i < num_fields; i++){
            cout.width(20); cout.setf(ios::left);
            cout << fields[i].name;
         };
         cout << endl;

         //输出查询出的结果集中的每条记录中的数据
         MYSQL_ROW row;
         while ((row = mysql_fetch_row(&res)))
         {

           for(unsigned int i = 0; i < num_fields; i++)
           {
               cout.width(20); cout.setf(ios::left);
               cout << row[i] ? row[i] : "NULL";
           }
           cout << endl;
         }
     }catch(DbExcepiton exp){
        cout << exp.what() << endl;
```

```
    }
    return 0;
}
```

- **insert操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);try{
        string sql;
        string loginName, password;
        loginName = "\'zstu002\'";
        password = "\'111111\'";
        //构建SQL语句
        sql = "insert into t_user (login_name, password) values (";
        sql = sql + loginName + "," + password + ")";
        //执行insert命令，返回成功插入的记录个数
        int num_add = db.executeSQL(sql.c_str());
        cout << num_add << " users is inserted!" << endl;
    }catch(DbExcepiton exp){
       cout << exp.what() << endl;
    }
    return 0;
}
```

- **update操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);
        string sql;
        int userId = 5;
        //构建SQL语句
        sql = "delete from t_user where ";
        sql = sql + "user_id = " + to_string(userId);
        //执行update命令，返回成功更新的记录个数
```

```
        int num_add = db.executeSQL(sql.c_str());
        if(num_add != 0)
           cout << num_add << " users is deleted!" << endl;
        else cout << "The deleted user does not exist" << endl;
    }catch(DbExcepiton exp){
       cout << exp.what() << endl;
    }
    return 0;
}
```

- **delete操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);
        string sql;
        string password = "\'123456\'";
        int userId = 6;
        //构建SQL语句
        sql = "update t_user set ";
        sql = sql + "password = " + password;
        sql = sql + "where user_id = " + to_string(userId);
        cout << sql << endl;
        //执行delete命令，返回成功删除的记录个数
        int num_add = db.executeSQL(sql.c_str());
        if(num_add != 0)
           cout << num_add << " users is updated!" << endl;
        else cout << "The updated user does not exist" << endl;
    }catch(DbExcepiton exp){
       cout << exp.what() << endl;
    }
    return 0;
}
```