## 创建C/C++访问mysql数据库项目

1. mingw64工具、mysql数据库安装配置完成
2. Vscode/cursor**创建独立的项目**（确保.vscode配置文件不会冲突）
3. 配置tasks.json文件：

{

"version": "2.0.0",

"tasks":

[

{

"label": "C/C++: g++.exe build active file",

"type": "process",

"command": "D:/mingw64/bin/g++.exe", # 改成你自己的路径

"args":

[

"-fdiagnostics-color=always",

"-g",

"${file}",

**"-L",**

**"C:\\Program Files\\MySQL\\MySQL Server 5.7\\lib", // mysql 库文件**

**"-I",**

**"C:\\Program Files\\MySQL\\MySQL Server 5.7\\include", // mysql头文件**

**"-llibmysql", // 导入那个库**

"-o",

"${fileDirname}\\${fileBasenameNoExtension}.exe"

],

"options":

{

"cwd": "D:/mingw64/bin/" # 改成你自己的路径

},

"group": "build",

"detail": "compiler: D:/mingw64/bin/g++.exe", # 改成你自己的路径

"problemMatcher": "$g++"

}

]

}

1. 复制“C:\Program Files\MySQL\MySQL Server 5.7\lib”下的libmysql.dll文件，黏贴到项目根路径下。
2. 包含头文件mysql.h

#include<mysql.h>

1. C、C++代码使用MySQL C API函数

MYSQL\* m\_Mysql = new MYSQL(); // Mysql连接句柄

MYSQL\_RES\* m\_SelectResult; // 用于存储Mysql查询结果集合

* mysql\_init()

用于初始化一个MYSQL对象，返回一个指向该对象的指针，即Mysql句柄。这个句柄用于后续的数据库连接和操作

mysql\_init(m\_Mysql);// Mysql初始化函数，返回Mysql句柄

* mysql\_real\_connect

用于连接到MySQL数据库。

它接受多个参数，包括MySQL句柄、主机地址、用户名、密码、数据库名称、端口号、Unix套接字和客户端连接标志。

函数返回一个连接句柄，如果连接成功则返回该句柄，否则返回NULL。

* mysql\_query函数

用于执行SQL查询，并接受两参数：第一个参数是一个指向已初始化的MYSQL结构体的指针，第二个参数是一个指向包含SQL查询语句的C字符串的指针。函数返回0表示查询成功，非0值表示出现错误。

* mysql\_store\_result函数

获得结果集,用于将查询结果存储在客户端，以便后续处理。

它接受一个指向已初始化的MYSQL结构体的指针作为参数，然后返回一个指向结果集的指针。这个结果集包含了查询返回的数据，可以通过该指针来访问和处理查询结果。

m\_SelectResult = mysql\_store\_result(m\_Mysql);

* mysql\_num\_fields(m\_SelectResult) //获取列数
* MYSQL\_FIELD\*field = mysql\_fetch\_field(m\_SelectResult)//取字段
* MYSQL\_ROW row = mysql\_fetch\_row(m\_SelectResult) //取出一行
* mysql\_free\_result(m\_SelectResult) //释放存储的查询结果集合
* mysql\_close(m\_Mysql) //释放Mysql连接资源

// SQL预处理语句(预处理语句是 SQL 语句的模板，它可以在执行之前被多次设置参数，然后被执行。这可以提高性能，因为数据库可以只解析一次 SQL 语句，并且还可以帮助防止 SQL 注入攻击。)

* mysql\_stmt\_init()
* mysql\_stmt\_prepare()
* mysql\_stmt\_bind\_param()
* mysql\_stmt\_execute()
* mysql\_stmt\_error()
* mysql\_stmt\_close()
* mysql\_stmt\_init()

初始化一个 MYSQL\_STMT 结构体，用于后续的预处理语句操作。

返回一个指向新初始化的 MYSQL\_STMT 的指针，或者如果失败则返回 NULL。

* mysql\_stmt\_prepare()

准备一个 SQL 语句以进行后续执行。

需要一个已初始化的 MYSQL\_STMT 指针和 SQL 语句字符串。

如果成功，该函数将 SQL 语句解析为预处理语句，并准备参数绑定和执行。

如果失败，可以通过 mysql\_stmt\_error() 获取错误信息。

* mysql\_stmt\_bind\_param()

绑定参数到预处理语句。

需要一个 MYSQL\_BIND 结构体数组，其中每个结构体描述一个要绑定的参数。

这允许你在执行预处理语句之前为其设置参数值。

* mysql\_stmt\_execute()

执行预处理语句。

如果预处理语句中有参数，它们应该在执行之前被绑定。

如果执行成功，该函数将返回 0。否则，你可以使用 mysql\_stmt\_error() 来获取错误信息。

* mysql\_stmt\_error()

返回与最近一次 MYSQL\_STMT 操作相关的错误消息。

这在检查 mysql\_stmt\_prepare(), mysql\_stmt\_execute() 等函数的错误时非常有用。

* mysql\_stmt\_close()

关闭一个 MYSQL\_STMT 结构体，释放与之关联的所有资源。

在完成对预处理语句的所有操作后，应该调用此函数来避免资源泄漏。

* 使用这些函数通常的流程：预处理流程

1. 使用 mysql\_stmt\_init() 初始化一个预处理语句。
2. 使用 mysql\_stmt\_prepare() 准备 SQL 语句。
3. （可选）使用 mysql\_stmt\_bind\_param() 绑定参数。
4. 使用 mysql\_stmt\_execute() 执行预处理语句。
5. 检查错误并处理结果。
6. 使用 mysql\_stmt\_close() 关闭预处理语句。