GreatDB
万里数据库

开发规范 2025

万里安全数据库软件

稳定 · 性能 · 易用

北京万里开源软件有限公司
Beijing Great OpenSource Software Co., Ltd.

目录

文档概述

文档适用范围

本文档主要介绍北京万里开源软件有限公司（以下简称“万里开源”）数据库软件（英文“GreatDB”，以下简称“GreatDB”）基于 GreatDB 驱动接口进行应用程序开发，和开发规范的说明。本文档适合初次接触本产品的用户，用于指导用户应用连接 GreatDB 数据库的配置方法、示例以及开发规范的说明性文档。

一、JDBC 开发规范

基于 GreatDB JDBC 驱动接口进行 Java 应用程序开发，相比 MySQL 生态的开源 JDBC 驱动无明显差异，主要差异点是在引用连接串 URL 时需要使用 greatdb 字样。以下是两个 JDBC 连接串的示例：

//非加密连接

jdbc:greatdb://localhost:3306/mysql?useSSL=false

//国密加密连接

jdbc:greatdb://localhost:3306/mysql?useSSL=true&clientCertificateKeyStoreType=GMTLS

jar 包：（请联系万里数据库获取。）

万里数据库推荐如下开发规范指导涉及 JDBC 接口的 Java 应用程序开发工作。

1.1.驱动与连接管理

驱动加载与版本适配:

选择与数据库版本兼容的驱动版本。

避免硬编码驱动类名（如 Class.forName("com.greatdb.cj.jdbc.Driver")），现代驱动（JDBC 4.0+）支持 SPI 自动加载，可省略显式加载步骤。

连接池使用规范:

禁止直接创建 DriverManager 连接，必须使用连接池（如 HikariCP、Druid、C3P0）管理连接，避免频繁创建 / 关闭连接导致的性能损耗。

连接池配置需合理设置参数：

核心池大小（corePoolSize）、最大池大小（maximumPoolSize）根据并发量调整；

连接超时（connectionTimeout）、空闲超时（idleTimeout）避免资源长期占用；

配置连接验证（如 validationQuery="SELECT 1"）确保连接有效性。

连接生命周期管理:

连接(Connection)使用后必须及时关闭，建议通过 try-with-resources 自动释放(JDBC 资源均实现 AutoCloseable）：

避免长时间持有连接（如在事务中执行耗时操作），防止连接池耗尽。

1.2.SQL 执行与参数处理

优先使用 PreparedStatement:

禁止使用 Statement 拼接 SQL（存在 SQL 注入风险），必须用 PreparedStatement 进行参数化查询：PreparedStatement 可预编译 SQL，重复执行时性能更优。

参数绑定规范:

严格匹配参数类型（如 setInt、setString、setTimestamp），避免因类型转换导致的隐含错误（如日期用 setString 可能引发数据库格式不兼容）。

字符串参数无需手动加单引号（PreparedStatement 自动处理），二进制数据（如 BLOB）使用 setBinaryStream 或 setBlob。

结果集（ResultSet）处理:

结果集使用后需关闭，若通过 try-with-resources 管理，可与 PreparedStatement 一同自动释放。

避免在循环中频繁调用 rs.getString(columnName)，可缓存列索引提升性能：

处理大结果集时,设置 fetchSize 控制一次获取的行数(如 pstmt.setFetchSize(100))，避免内存溢出。

1.3.事务管理

显式事务控制:

禁止依赖数据库默认的 “自动提交”（conn.setAutoCommit(true)），关键业务需显式控制事务：

事务隔离级别:

根据业务需求设置合理的隔离级别（默认由数据库决定），避免过度隔离导致性能下降：

长事务禁止:

事务中避免包含耗时操作（如 IO、网络请求），防止锁竞争和连接池阻塞。

1.4.错误处理与日志

异常处理规范:

捕获 SQLException 时，需获取详细信息（错误代码、SQL 状态、消息）：

禁止空捕获（catch (SQLException e) {}），需至少记录日志。

日志记录要求:

记录关键操作日志：连接建立 / 失败、SQL 执行异常、事务提交 / 回滚，包含时间、SQL 语句（脱敏敏感信息）、耗时等。

避免日志中暴露敏感数据（如密码、身份证号），需进行脱敏处理。

1.5.安全性规范

防 SQL 注入:

除参数化查询外，对用户输入的特殊字符（如’、;）需额外校验，但参数化是核心手段。

禁止动态拼接表名、列名（若必须，需通过白名单校验，避免直接使用用户输入）。

敏感信息保护:

数据库账号密码禁止硬编码，需通过配置中心（如 Nacos）、环境变量或加密配置文件（如 Jasypt）获取。

连接字符串中避免明文传输密码，优先使用 SSL 加密连接（如 MySQL 添加 useSSL=true）。

1.6.性能与兼容性

性能优化:

批量操作使用 addBatch()和 executeBatch()减少网络交互。

避免 SELECT *，只查询必要列；大表查询需加索引和分页（LIMIT 或 ROW_NUMBER()）。

关闭不需要的自动提交和游标滚动(如 ResultSet.TYPE_FORWARD_ONLY 提升读取性能)。

跨数据库兼容：

尽量使用标准 SQL（如 ANSI SQL），避免数据库特有语法（如 MySQL 的 LIMIT、Oracle 的 ROWNUM）。

日期时间处理优先使用 java.time(JDBC 4.2 + 支持)，避免数据库特定函数(如 NOW()、SYSDATE）。

1.7.代码规范与可维护性

命名与注释：

资源变量命名清晰：conn（连接）、pstmt（预处理语句）、rs（结果集）。

SQL 语句建议定义为 static final String 常量，复杂 SQL 可单独放在配置文件中。

模块化封装：

封装通用操作(如查询、更新、批量处理)为工具类(如 JdbcTemplate,或直接使用 Spring JDBC），避免重复代码。

业务逻辑与 JDBC 操作分离，通过 DAO（Data Access Object）层统一管理数据库访问。

资源释放保障：

即使使用 try-with-resources，仍需在复杂场景下确保资源释放（如嵌套资源需分别声明）。

避免在 finally 中关闭资源时抛出异常（可能覆盖原异常），建议单独捕获。

二、C API 开发规范

2.1.负载均衡功能配置示例

对于负载均衡功能，C API 通过操作 MYSQL 指针对象创建数据库连接、执行数据库操作，通过 mysql_options 接口设置连接属性，为实现负载均衡功能，mysql_option 中新增枚举值，通过新增的枚举值设置负载均衡相关的连接属性。

负载均衡连接使用步骤：

1. 调用 mysql_options 设置 MYSQL_LOAD_BALANCE_HOSTS 、MYSQL_LOAD_BALANCE_STRATEGY、MYSQL_LOAD_BALANCE_BLOCKLIST_TIMEOUT 三个属性，同一个 MYSQL 对象只需要设置一次 loadbalance 属性；

2. 调用现有 mysql_real_connect 接口以阻塞模式创建连接，调用 现有 mysql_real_connect_nonblocking 以非阻塞模式创建连接；

3. 如需进行 rebalance，调用 mysql_real_connect 或 mysql_real_connect_nonblocking 切换连接，同一个 MYSQL 对象可多次调用 mysql_real_connect / mysql_real_connect_nonblocking 进行 rebalance；

4.连接使用完毕后，调用 mysql_close 关闭 MYSQL 对象。

mysql_real_connect 函数说明：

```
// 创建连接，创建成功返回值非空，后续使用 mysql 进行数据库操作
// 创建失败，返回值为空指针，mysql_errno 获取错误码，mysql_error 获取错误信息
// 如果 mysql 设置了 loadbalance 相关属性，host、port、unix_socket 三个参数不生效
MYSQL *STDCALL mysql_real_connect(MYSQL *mysql, const char *host,
                                  const char *user, const char *passwd,
                                  const char *db, uint port,
                                  const char *unix_socket, ulong client_flag)
```

mysql_real_connect_nonblocking 函数说明：

```
// 创建连接完毕，返回值为 NET_ASYNC_COMPLETE，后续使用 mysql 进行数据库操作
// 连接未完毕，返回值 NET_ASYNC_NOT_READY，继续调用本接口等待连接完毕
// 失败，返回 NET_ASYNC_ERROR，mysql_errno 获取错误码，mysql_error 获取错误信息
// 如果 mysql 设置了 loadbalance 相关属性，host、port、unix_socket 三个参数不生效
net_async_status STDCALL mysql_real_connect_nonblocking(
    MYSQL *mysql, const char *host, const char *user, const char *passwd,
    const char *db, uint port, const char *unix_socket, ulong client_flag)
```

以阻塞模式创建负载均衡连接示例：

```
MYSQL *mysql_connection = mysql_init(nullptr);
// 指定 load balance 的连接串
const char * hosts = "127.0.0.1:3306,127.0.0.1:3306"
mysql_options(mysql_connection, MYSQL_LOAD_BALANCE_HOSTS,
        (void *)const_cast<char *>(hosts));
// 指定 load balance 策略
mysql_load_balance_strategy load_balance_strategy = LOAD_BALANCE_RANDOM;
mysql_options(mysql_connection,  MYSQL_LOAD_BALANCE_STRATEGY,
        (void *)&load_balance_strategy);
// 指定 blocklist 超时时间
```

```
    unsigned long long block_timeout = 3000;
    mysql_options(mysql_connection,  MYSQL_LOAD_BALANCE_BLOCKLIST_TIMEOUT,
            (void *)&block_timeout);
    // 创建 load balance 的连接，API 内部忽略 host、port、unix_socket 三个参数
    if (!mysql_real_connect(mysql_connection, nullptr/*host*/, opt_user/*user*/,
                    password,  ""/*db*/,  0/*port*/,  nullptr/*unix_socket*/,
0/*client_flag*/)) {
    // 连接失败
    return;
    }
    mysql_query(mysql_connection, “select id from t”);
    // rebalance，mysql_connection 断开当前连接，重新连接到其他数据库进程
    mysql_real_connect(mysql_connection, nullptr/*host*/, opt_user/*user*/,
                    password,  ""/*db*/,  0/*port*/,  nullptr/*unix_socket*/,
0/*client_flag*/);
    // mysql_connection 使用完毕，关闭连接
    mysql_close(mysql_connection);
```

以非阻塞模式创建负载均衡连接举例：

```
    MYSQL *mysql_connection = mysql_init(nullptr);
    // 指定 load balance 的连接串
    const char * hosts = "127.0.0.1:3306,127.0.0.1:3306"
    mysql_options(mysql_connection, MYSQL_LOAD_BALANCE_HOSTS,
            (void *)const_cast<char *>(hosts));
    // 指定 load balance 策略
    mysql_load_balance_strategy load_balance_strategy = LOAD_BALANCE_RANDOM;
    mysql_options(mysql_connection,  MYSQL_LOAD_BALANCE_STRATEGY,
            (void *)&load_balance_strategy);
    // 创建 load balance 的连接
    net_async_status mysql_conn_status = mysql_real_connect_nonblocking(
                mysql_connection, nullptr/*host*/, opt_user/*user*/,
                    password,  ""/*db*/,  0/*port*/,  nullptr/*unix_socket*/,
0/*client_flag*/);
    while (NET_ASYNC_NOT_READY == mysql_conn_status) {
        mysql_conn_status = mysql_real_connect_nonblocking(
                mysql_connection, nullptr/*host*/, opt_user/*user*/,
                    password,  ""/*db*/,  0/*port*/,  nullptr/*unix_socket*/,
0/*client_flag*/);
    // mysql_connection 使用完毕，关闭连接
    mysql_close(mysql_connection);
```

提供接口返回负载均衡连接的状态，状态信息以字符串形式返回，结果格式类似于 show

engine innodb status，目前支持的统计指标有：

1. load balance 连接点列表
2. 创建过的 connection 总数，各个连接点创建过的 connection 数量
3. 当前不可用的连接点地址列表，即 blocklist 信息，blocklist 超时时间

获取统计信息接口：

```
// status 返回统计信息结果，API 内部管理内存，应用程序不需要处理内存的申请与释放
// 获取状态信息成功，返回 0；失败，返回错误码
int mysql_get_loadbalance_status(MYSQL *mysql, char **status)
```

注意，mysql 的生命周期大于 status 的生命周期，通过 mysql_get_loadbalance_status 获取连接状态信息后，关闭 mysql 之后，status 也随之释放，不可再使用 status。

获取连接状态信息举例：

```
char *status = nullptr;
if (0 == mysql_get_loadbalance_status(mysql, &status)) {
fprintf(stdout, "status:\n%s\n", status);
}
```

status 信息格式如下：

```
load balance status
hosts: 127.0.0.1:2002,127.0.0.1:2003,127.0.0.1:2004
load balance strategy: ROUND_ROBIN
current connection ip: 127.0.0.1, port: 2003
external blocking connect count: 11, external nonblocking connect count: 0
avliable host info
avliable host 127.0.0.1:2002 internal connect count: 5, connect success count: 5, connect fail count: 0, reconnect count: 0
avliable host 127.0.0.1:2003 internal connect count: 6, connect success count: 6, connect fail count: 0, reconnect count: 0
block host info
block host timeout: 1000 milliseconds, current time epoch: 1735887502612
block host 127.0.0.1:2004 internal connect count: 1, connect success count: 0, connect fail count: 1, reconnect count: 0, block time epoch: 1735887502556
mark hosts available consume: 1 microseconds, mark hosts unavliable consume: 1 microseconds.
```

获取连接 ip 和端口号：

```
// 返回 mysql 当前连接到的数据库的 ip 和 port
// 应用程序不需要处理 ip 内存的申请与释放，port 的内存需要应用程序来管理
// 获取状态信息成功，返回 0；失败，返回错误码
int mysql_get_current_connection_host(MYSQL *mysql, char **ip, unsigned int *port)
```

注意，mysql 的生命周期大于 ip 的生命周期，通过 mysql_get_current_connection_host 获取 ip 后，关闭 mysql 之后，ip 也随之释放，不可再使用 ip。

获取连接 ip 和端口号举例：

```
char *ip = nullptr;
unsigned int port;
if (0 == mysql_get_current_connection_host(mysql, &ip, &port)) {
fprintf(stdout, "ip: %s, port: %u\n", ip, port);
}
```

注意事项：

1. 使用负载均衡方式开启连接后，事务执行过程中如果发生通信异常，应用程序会接收到异常信息，应用程序需要再次调用 mysql_real_connect/mysql_real_connect_nonblocking 进行 rebalance，连接到可用的数据库节点。

2. 负载均衡连接使用完毕后，应用程序需要调用 mysql_close 关闭连接。

3. 负载均衡功能与 MYSQL_OPT_RECONNECT 重连功能不冲突，如果同时开启负载均衡和重连功能，在 SQL 语句执行过程中发生网络异常，会对当前的连接尝试重连，如果重连失败，可以调用 mysql_real_connect/mysql_real_connect_nonblocking 连接到其他数据库节点。

对于国密算法加密连接通道的设置，需要通过 mysql_options 设置新的候选值，示例如下：

```
    if (mysql_options(&mysql, MYSQL_OPT_GM_SSL, "tassl_gm") != 0) {
      fprintf(stderr, "设置国密失败%s\n", mysql_error(&mysql));
      mysql_close(&mysql);
      return EXIT_FAILURE;
    }
```

除上述开发规范外，万里数据库还推荐如下开发规范指导涉及 C API 接口的应用程序开发工作。

2.2.环境与依赖规范

版本兼容性：

明确指定目标数据库版本，确保使用的 C API 版本与数据库服务版本兼容。

编译时通过 -lmysqlclient 链接正确的库文件，避免使用过时的头文件（如 mysql.h 需对应当前库版本）。

2.3.连接管理规范

连接创建与释放：

使用 mysql_init() 初始化连接句柄，mysql_real_connect() 建立连接，确保传入正确的主机、端口、用户名、密码及数据库名。

连接失败时，必须通过 mysql_error() 获取错误信息并记录日志，避免直接忽略错误。

连接使用完毕后，必须调用 mysql_close() 释放资源，防止句柄泄露。

连接参数设置：

显式设置字符集（如 mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "utf8mb4")），避免中文等多字节字符乱码。

按需设置连接超时（MYSQL_OPT_CONNECT_TIMEOUT）、读写超时（MYSQL_OPT_READ_TIMEOUT/MYSQL_OPT_WRITE_TIMEOUT），防止长时间阻塞。

2.4.SQL 执行规范

SQL 语句处理：

禁止直接拼接用户输入到 SQL 语句中（防止 SQL 注入），必须使用 mysql_real_escape_string() 对字符串参数进行转义，或优先使用 预处理语句（Prepared Statements）。

预处理语句推荐流程：mysql_stmt_init() → mysql_stmt_prepare() → mysql_stmt_bind_param() → mysql_stmt_execute() → mysql_stmt_close()，适用于重复执行的 SQL 或含参数的场景。

执行结果检查：

执行 SQL 后（如 mysql_query() 或 mysql_stmt_execute()），必须检查返回值（0 为成功，非 0 为失败），失败时通过 mysql_error() 或 mysql_stmt_error() 获取详细错误信息。

对于查询语句（SELECT 等），需通过 mysql_store_result() 或 mysql_use_result() 获取结果集，处理完毕后必须调用 mysql_free_result() 释放结果集资源。

2.5.结果集处理规范

结果集遍历：

使用 mysql_fetch_row() 遍历结果行，通过 mysql_num_fields() 获取列数，mysql_fetch_field() 获取列元信息（如字段名、类型）。

处理大结果集时，优先使用 mysql_use_result()（流式获取，内存占用低），但需注意必须尽快处理并释放，避免阻塞服务器。

数据类型转换：

从结果集中获取数据时，需根据字段类型（如 MYSQL_TYPE_INT、MYSQL_TYPE_VARCHAR）进行正确的类型转换，避免因类型不匹配导致数据错误。

对于 NULL 值，通过 mysql_fetch_lengths() 检查字段长度是否为 NULL 来判断，避免直接使用空指针。

2.6.错误处理与日志规范

错误捕获：

所有可能失败的 API 调用(如连接、执行、获取结果)必须添加错误检查,不能依赖“默认成功”。

错误信息需包含上下文（如当前执行的 SQL、参数值、调用的 API 函数名），便于问题定位。

日志记录：

关键操作（连接成功 / 失败、SQL 执行异常、资源释放失败）必须记录日志，日志级别区分（INFO/WARN/ERROR）。

避免在日志中明文记录密码等敏感信息。

2.7.资源管理规范

句柄与内存管理：

连接句柄（MYSQL*）、预处理句柄（MYSQL_STMT*）、结果集（MYSQL_RES*）必须一一对应释放，遵循 “谁创建谁释放” 原则。

避免在循环或分支中创建资源后未释放（如提前 return 导致 mysql_close() 或 mysql_free_result() 未执行）。

内存泄漏检查：

使用工具（如 valgrind）定期检测内存泄漏，重点检查结果集、预处理参数绑定的内存是否完全释放。

2.8.性能与安全规范

性能优化：

批量操作优先使用预处理语句（减少编译次数），或通过 INSERT ... VALUES (...), (...), ... 减少网络交互。

避免频繁创建 / 关闭连接，可考虑连接池（需自行实现或依赖第三方库）。

安全加固：

最小权限原则：数据库账号仅授予必要权限（如仅 SELECT/INSERT，避免 DROP/ALTER）。

敏感数据（如密码）在代码中避免硬编码，优先通过配置文件或环境变量读取，并确保配置文件权限安全（如仅所有者可读写）。

三、ODBC 开发规范

万里数据库兼容 MySQL 生态 ODBC 驱动，推荐如下开发规范用于指导涉及 ODBC 接口的应用程序开发工作。

3.1.环境与连接管理

环境句柄初始化：

通过 SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv) 初始化环境句柄，且需设置 ODBC 版本（如 SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0)），避免版本兼容问题。

环境句柄使用后必须通过 SQLFreeHandle(SQL_HANDLE_ENV, henv) 释放，防止资源泄漏。

连接句柄管理：

连接句柄需基于环境句柄分配（SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc)），连接数据库时使用 SQLConnect 或 SQLDriverConnect（支持连接字符串）。

连接成功后，建议设置连接属性（如超时 SQL_ATTR_CONNECTION_TIMEOUT、事务隔离级别 SQL_ATTR_TXN_ISOLATION）。

断开连接需调用 SQLDisconnect(hdbc)，并释放连接句柄 SQLFreeHandle(SQL_HANDLE_DBC, hdbc)。

3.2.SQL 语句执行规范

语句句柄操作：

语句句柄通过 SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt) 分配，每个语句句柄对应一个 SQL 操作，使用后需通过 SQLFreeHandle(SQL_HANDLE_STMT, hstmt) 释放。

执行 SQL 前，建议通过 SQLPrepare 预处理语句（尤其重复执行时），提高效率；单次执行可直接使用 SQLExecDirect。

长连接场景下，定期检测连接有效性（如执行 SELECT 1），避免使用失效连接。

参数绑定与类型安全：

使用 SQLBindParameter 绑定输入 / 输出参数，明确参数类型（如 SQL_INTEGER、SQL_VARCHAR）、长度和精度，避免类型不匹配导致的错误。

字符串参数需指定长度（含终止符），二进制数据需指定字节数。

结果集处理：

对于查询语句（SELECT），需通过 SQLFetch 或 SQLFetchScroll 提取结果，使用 SQLBindCol 绑定结果集列到变量，明确数据类型和缓冲区大小。

处理大字段（如 TEXT、BLOB）时，需通过 SQLGetData 分块读取，避免内存溢出。

结果集使用完毕后，需调用 SQLCloseCursor(hstmt) 关闭游标。

3.3.错误处理与日志

错误信息获取：

所有 ODBC 函数调用后必须检查返回值（非 SQL_SUCCESS 和 SQL_SUCCESS_WITH_INFO 需处理）。

通过 SQLGetDiagRec 或 SQLGetDiagField 获取详细错误信息（包括 SQLSTATE、错误代码、描述），便于调试。

日志记录：

关键操作（如连接失败、SQL 执行错误）需记录日志，包含时间、操作类型、错误详情，便于问题追踪。

3.4.安全性规范

防止 SQL 注入：

禁止直接拼接用户输入到 SQL 语句，必须使用参数化查询（SQLBindParameter）。

示例：错误方式 SELECT * FROM users WHERE name = '${userInput}'；正确方式 SELECT * FROM users WHERE name = ? 并绑定参数。

敏感信息保护：

数据库账号、密码避免硬编码，建议通过配置文件（加密存储）或环境变量读取，且内存中使用后及时清零。

传输过程中（如连接字符串）建议使用加密协议（如 SSL），尤其远程数据库连接。

权限最小化：

数据库账号仅授予必要权限（如查询、插入，避免 DROP、ALTER 等高危操作），降低风险。

3.5.兼容性

尽量使用标准 SQL 语法，减少数据库特定扩展（如 MySQL 的 LIMIT、SQL Server 的 TOP），如需使用需通过条件编译适配。

对不同数据库的类型映射（如日期时间类型）做兼容处理，避免因类型差异导致的错误。

3.6.代码风格与可维护性

命名规范：

句柄变量建议前缀明确（如 henv 表示环境句柄，hdbc 表示连接句柄，hstmt 表示语句句柄）。

函数和变量名使用有意义的名称（如 ConnectToDatabase、ExecuteQuery），避免缩写。

模块化设计：

将连接管理、SQL 执行、错误处理等封装为独立函数（如 ODBC_Connect、ODBC_Execute），提高复用性。

避免在业务逻辑中直接嵌入大量 ODBC 操作代码，降低耦合。

四、Python 开发规范

万里数据库产品使用基于开源 Python 生态的数据库 API 接口进行 Python 应用程序开发工作，例如 mysql-connector-python、PyMySQL 等开源方案。

万里数据库推荐如下开发规范用于指导涉及 Python 开发接口的应用程序开发工作。

4.1.环境与依赖管理

版本控制：

明确指定接口版本（如 PyMySQL>=1.0.2,<2.0），避免因版本兼容问题导致异常。

连接配置：

数据库配置（主机、端口、用户名、密码等）避免硬编码，应通过环境变量（如 os.environ.get）或配置文件（如 .env、config.yaml）加载。

敏感信息（密码、密钥）需加密存储或使用 secrets 管理工具，禁止提交到代码仓库。

4.2.连接与会话管理

连接池使用：

生产环境必须使用连接池（如 DBUtils.PooledDB），避免频繁创建 / 关闭连接导致性能损耗。

连接池参数需根据业务压力调整（如 maxconnections 避免连接耗尽）。

使用 with 语句自动管理连接和游标，确保资源释放（即使发生异常）。

避免长期持有连接（如在循环或长任务中保持连接），用完立即释放。

4.3.SQL 操作规范

防 SQL 注入：

禁止直接拼接 SQL 字符串（如 f"SELECT * FROM users WHERE id={user_id}"），必须使用参数化查询：

复杂查询如需动态拼接表名 / 字段名，需先校验合法性（如白名单过滤），避免直接传入用户输入。

字符集与编码：

连接时指定 charset='utf8mb4'（支持 emoji 和全 Unicode 字符），避免中文乱码。

数据库表和字段的字符集也需同步设置为 utf8mb4。

事务管理：

显式控制事务（默认自动提交可能导致数据不一致）：

事务中避免包含耗时操作（如网络请求），减少锁表时间。

游标与结果处理：

根据需求选择游标类型。默认游标（返回元组）适合简单查询。

字典游标（cursorclass=pymysql.cursors.DictCursor）：返回字典（键为字段名），可读性更好。

大结果集使用 cursor.fetchmany(size) 分批获取，避免内存溢出。

4.4.错误处理与日志

异常捕获：

捕获 PyMySQL 特定异常（如 pymysql.MySQLError、pymysql.OperationalError），而非泛泛的 Exception，便于定位问题：

日志记录：

记录关键操作（如执行的 SQL、影响行数、错误信息），但禁止日志中包含敏感数据（密码、手机号等）。

调试环境可输出 SQL 语句，生产环境需关闭（避免泄露信息）。

4.5.性能与安全

索引与查询优化：

避免 SELECT *，只查询需要的字段。

复杂查询需先在数据库中测试执行计划（EXPLAIN），确保使用索引。

权限最小化：

数据库账号仅授予必要权限（如业务账号只给 SELECT/INSERT/UPDATE，禁止 DROP 等高危操作）。

连接超时设置：

配置合理的超时参数（connect_timeout、read_timeout、write_timeout），避免连接长期挂起：

4.6.测试与部署

单元测试：

使用测试数据库（而非生产库）进行测试，可借助 pytest 结合 unittest.mock 模拟数据库操作。

部署检查

上线前确认：

已关闭调试模式（如日志中不输出敏感信息）。

连接池参数适配生产环境压力。

所有 SQL 已通过参数化处理，无注入风险。

五、版权声明

5.1.法律声明

若接收北京万里开源软件有限公司（以下称为“万里数据库”）的此份文档，即表示您已同意以下条款。若不同意以下条款，请停止使用本文档。

本文档所载内容受著作权法的保护，著作权为北京万里开源软件有限公司所有，但注明引用其他方的内容除外。北京万里开源软件有限公司保留任何未在本文档中明示授予的权利。文档中涉及万里数据库的专有信息。未经万里数据库事先书面许可，任何单位和个人不得复制、传递、分发、使用和泄漏该文档以及该文档包含的任何图片、表格、数据及其他信息或者其他任何商业目的的使用。

5.2.商标声明

GreatDB 和GreatDB Cluster是万里数据库的注册商标。万里数据库产品的名称和标志是万里数据库的商标或注册商标。在本文档中提及的其他产品或公司名称可能是其各自所有者的商标或注册商标。在未经万里数据库或第三方权利人事先书面同意的情况下，阅读本文档并不表示以默示、不可反言或其他方式授予阅读者任何使用本文档中出现的任何标记的权利。

5.3.服务声明

本产品符合有关环境保护和人身安全方面的设计要求，产品的存放、使用和弃置应遵照产品手册、相关合同或相关国家法律法规的要求进行。

本文档按“现状”和“仅此状态”提供，文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。本文档中的信息随着万里数据库产品和技术的进步将不断更新，万里数据库不再通知此类信息的更新。

GreatDB
万里数据库

联系我们 Contact Us

地址：北京市朝阳区CBD国际大厦7层701B
电话：400-032-7868
邮箱：sales@greatdb.com
网站：https://www.greatdb.com

北京万里开源软件有限公司

稳定 · 性能 · 易用