ODB 安装与使用

版权说明

对比特项目感兴趣，可以联系这个微信。

代码 & 板书链接

https://gitee.com/bitedu-tech/cpp-chatsystem

ODB2.5 版本安装：

> 整个安装过程在 3 - 5 小时左右， 视网络情况而定

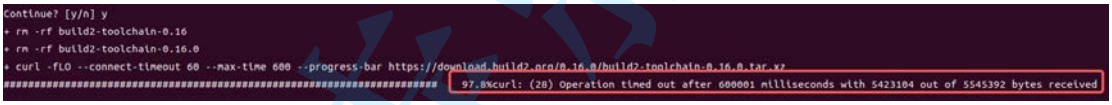
安装 build2：

因为 build2 安装时，有可能会版本更新，从 16 变成 17，或从 17 变 18，因此注意，先从 build2 官网查看安装步骤...

安装步骤：https://build2.org/install.xhtml#unix

```
PowerShell
dev@dev-host:~/workspace$ curl -sSfO
https://download.build2.org/0.17.0/build2-install-0.17.0.sh
dev@dev-host:~/workspace$ sh build2-install-0.17.0.sh
```

安装中因为网络问题，超时失败，解决：将超时时间设置的更长一些




```
Shell
dev@dev-host:~/workspace$ sh build2-install-0.17.0.sh --timeout
1800
```

安装 odb-compiler

```
PowerShell
dev@dev-host:~/workspace$ #注意这里的 gcc-11 需要根据你自己版本而定
dev@dev-host:~/workspace$ sudo apt-get install gcc-11-plugin-dev
dev@dev-host:~/workspace$ mkdir odb-build && cd odb-build
dev@dev-host:~/workspace/odb-build$ bpkg create -d odb-gcc-N cc
```

```
\
  config.cxx=g++                  \
  config.cc.coptions=-O3          \
  config.bin.rpath=/usr/lib \
  config.install.root=/usr/  \
  config.install.sudo=sudo
dev@dev-host:~/workspace/odb-build$ cd odb-gcc-N
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ bpkg build
odb@https://pkg.cppget.org/1/beta
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ bpkg test odb
test odb-2.5.0-b.25+1/tests/testscript{testscript}
tested odb/2.5.0-b.25+1
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ bpkg install odb
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ odb --version
bash: /usr/bin/odb: No such file or directory
#如果报错了，找不到odb，那就在执行下边的命令
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ sudo echo 'export
PATH=${PATH}:/usr/local/bin' >> ~/.bashrc
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ export
PATH=${PATH}:/usr/local/bin
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ odb --version
ODB object-relational mapping (ORM) compiler for C++ 2.5.0-b.25
Copyright (c) 2009-2023 Code Synthesis Tools CC.
This is free software; see the source for copying conditions.
There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

安装 ODB 运行时库：

```
PowerShell
dev@dev-host:~/workspace/odb-build/odb-gcc-N$ cd ..
dev@dev-host:~/workspace/odb-build$ bpkg create -d libodb-gcc-N cc
\
  config.cxx=g++                  \
  config.cc.coptions=-O3          \
  config.install.root=/usr/  \
  config.install.sudo=sudo
dev@dev-host:~/workspace/odb-build$ cd libodb-gcc-N
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg add
https://pkg.cppget.org/1/beta
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg fetch
```

```
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg build libodb
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg build
libodb-mysql
```

安装 mysql 和客户端开发包

```
C++
sudo apt install mysql-server
sudo apt install -y libmysqlclient-dev
```

配置 mysql

```
C++
sudo vim /etc/my.cnf  或者  /etc/mysql/my.cnf 有哪个修改哪个就行
#添加以下内容
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
[mysqld]
character-set-server=utf8
bind-address = 0.0.0.0
```

修改 root 用户密码

```
C++
dev@bite:~$ sudo cat /etc/mysql/debian.cnf
# Automatically generated for Debian scripts. DO NOT TOUCH!
[client]
host     = localhost
user     = debian-sys-maint
password = UWcn9vY0NkrbJMRC
socket   = /var/run/mysqld/mysqld.sock
[mysql_upgrade]
host     = localhost
user     = debian-sys-maint
password = UWcn9vY0NkrbJMRC
socket   = /var/run/mysqld/mysqld.sock
dev@bite:~$ sudo mysql -u debian-sys-maint -p
Enter password: #这里输入上边第 6 行看到的密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH
```

```
mysql_native_password BY 'xxxxxx';
Query OK, 0 rows affected (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)

mysql> quit
```

重启 mysql，并设置开机启动

```
C++
sudo systemctl restart mysql
sudo systemctl enable mysql
```

安装 boost profile 库

```
PowerShell
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg build
libodb-boost
```

总体打包安装：

```
PowerShell
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg install --
all --recursive
```

总体卸载：

```
PowerShell
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg uninstall --
all --recursive
```

总体升级：

```
PowerShell
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg fetch
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg status
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg uninstall --
all --recursive
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg build --
```

```
upgrade --recursive
dev@dev-host:~/workspace/odb-build/libodb-gcc-N$ bpkg install --
all --recursive
```

测试样例：

编写数据结构文件： person.hxx

```
C++
#pragma once
#include <string>
#include <cstddef> // std::size_t
#include <boost/date_time/posix_time/posix_time.hpp>

/*
    在 C++ 中，要使用 ODB 将类声明为持久化类，需要包含 ODB 的核心头文
件，并使用 #pragma db object 指令
    #pragma db object 指示 ODB 编译器将 person 类视为一个持久化类。
*/
#include <odb/core.hxx>

typedef boost::posix_time::ptime ptime;

#pragma db object
class Person {
    public:
        Person(const std::string &name, int age, const ptime
&update):
            _name(name), _age(age), _update(update){}
        void age(int val) { _age = val; }
        int age() { return _age; }
        void name(const std::string& val) { _name = val; }
        std::string name() { return _name; }
        void update(const ptime &update) { _update = update; }
        std::string update() { return
boost::posix_time::to_simple_string(_update); }
    private:
        //将 odb:: access 类作为 person 类的朋友。
        //这是使数据库支持代码可访问默认构造函数和数据成员所必需的。
        //如果类具有公共默认构造函数和公共数据成员或数据成员的公共访问器
和修饰符，则不需要友元声明
        friend class odb::access;
        Person () {}
```

```
        //_id 成员前面的 pragma 告诉 ODB 编译器，以下成员是对象的标识符。auto 说明符指示它是数据库分配的 ID。
        #pragma db id auto // 表示 ID 字段将自动生成（通常是数据库中的主键）。
        unsigned long _id;
        unsigned short _age;
        std::string _name;
        #pragma db type("TIMESTAMP") not_null
        boost::posix_time::ptime _update;
};
//将 ODB 编译指示组合在一起，并放在类定义之后。它们也可以移动到一个单独的标头中，使原始类完全保持不变
// #pragma db object(person)
// #pragma db member(person::_name) id

//完成后，需要使用 odb 编译器将当前所写的代码生成数据库支持代码
//odb -d mysql --generate-query --generate-schema person.hxx

//如果用到了 boost 库中的接口，则需要使用选项  : --profile boost/date-time
//odb -d mysql  --generate-query --generate-schema --profile boost/date-time person.hxx
```

生成数据库支持的代码文件：

```
C++
dev@dev-host:~/workspace/odb-test$ odb -d mysql  --std c++11 --generate-query --generate-schema --profile boost/date-time person.hxx
dev@dev-host:~/workspace/odb-test$ ls
person-odb.cxx  person-odb.hxx  person-odb.ixx  person.hxx  person.sql
```

编写主函数代码：test.cc

```
C++
#include <string>
#include <memory>   // std::auto_ptr
#include <cstdlib>  // std::exit
#include <iostream>

#include <odb/database.hxx>
#include <odb/mysql/database.hxx>
```

```
#include "person.hxx"
#include "person-odb.hxx"

int main() {
    std::shared_ptr<odb::core::database> db(
        new odb::mysql::database("root", "Zwc111...",
        "mytest", "127.0.0.1", 0, 0, "utf8"));
    if (!db) { return -1; }
    ptime p = boost::posix_time::second_clock::local_time();
    Person zhang("小张", 18, p);
    Person wang("小王", 19, p);

    typedef odb::query<Person> query;
    typedef odb::result<Person> result;
    {
        odb::core::transaction t(db->begin());
        size_t zid = db->persist(zhang);
        size_t wid = db->persist(wang);
        t.commit();
    }
    {
        ptime p = boost::posix_time::time_from_string("2024-05-22
09:09:39");
        ptime e = boost::posix_time::time_from_string("2024-05-22
09:13:29");
        odb::core::transaction t (db->begin());
        result r (db->query<Person>(query::update < e &&
query::update > p));
        for (result::iterator i(r.begin()); i != r.end(); ++i) {
            std::cout << "Hello, " << i->name() << " ";
            std::cout << i->age() << " " << i->update() <<
std::endl;
        }
        t.commit();
    }
    return 0;
}

//如果用到了 boost 库中的接口，需要链接库： -lodb-boost

//c++ -o mysql_test mysql_test.cpp person-odb.cxx -lodb-mysql -
lodb -lodb-boost
```

代码编译:

```C++
dev@dev-host:~/workspace/odb-test$ c++ -o test test.cpp person-odb.cxx -lodb-mysql -lodb -lodb-boost
```

运行时报错：这有可能是因为库文件安装在/usr/local/lib 下了

如果前边默认设置将库安装在/usr 下，应该不会有这个问题

```C++
dev@dev-host:~/workspace/odb-test$ ./test
./test: error while loading shared libraries: libodb-2.5.0-b.25.so: cannot open shared object file: No such file or directory
```

```C++
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

ODB 常见操作介绍：

ODB 类型映射：

C++ Type	MySQL Type	Default NULL Semantics
bool	TINYINT(1)	NOT NULL
char	CHAR(1)	NOT NULL
signed char	TINYINT	NOT NULL
unsigned char	TINYINT UNSIGNED	NOT NULL
short	SMALLINT	NOT NULL
unsigned short	SMALLINT UNSIGNED	NOT NULL
int	INT	NOT NULL
unsigned int	INT UNSIGNED	NOT NULL
long	BIGINT	NOT NULL
unsigned long	BIGINT UNSIGNED	NOT NULL
long long	BIGINT	NOT NULL
unsigned long long	BIGINT UNSIGNED	NOT NULL
float	FLOAT	NOT NULL
double	DOUBLE	NOT NULL
std::string	TEXT/VARCHAR(255)	NOT NULL
char[N]	VARCHAR(N-1)	NOT NULL

Boost date_time Type	MySQL Type	Default NULL Semantics
gregorian::date	DATE	NULL
posix_time::ptime	DATETIME	NULL
posix_time::time_duration	TIME	NULL

ODB 编程：

ODB（Open Database）在数据元结构定义时，使用预处理器指令（`#pragma`）来提供元数据，这些元数据指示如何将 C++类型映射到数据库模式。这些`#pragma` 指令是在 C++代码中使用的，它们不是 C++语言的一部分，而是特定于 ODB 编译器的扩展。以下是 ODB 中常用的一些`#pragma` 指令：

1. **#pragma db object**:
    - 用于声明一个类是数据库对象，即这个类将映射到数据库中的一个表。

2. **#pragma db table("table_name")**:
    - 指定类映射到数据库中的表名。如果不指定，则默认使用类名。
3. **#pragma db id**:
    - 标记类中的一个成员变量作为数据库表的主键。
4. **#pragma db column("column_name")**:
    - 指定类成员映射到数据库表中的列名。如果不指定，则默认使用成员变量的名字。
5. **#pragma db view**:
    - 用于声明一个类是一个数据库视图，而不是一个表。
6. **#pragma db session**:
    - 用于声明一个全局或成员变量是数据库会话。
7. **#pragma db query("query")**:
    - 用于定义自定义的查询函数。
8. **#pragma db index("index_name")**:
    - 指定成员变量应该被索引。
9. **#pragma db default("default_value")**:
    - 指定成员变量的默认值。
10. **#pragma db unique**:
    - 指定成员变量或一组变量应该具有唯一性约束。
11. **#pragma db not_null**:
    - 指定成员变量不允许为空。
12. **#pragma db auto**:
    - 指定成员变量的值在插入时自动生成（例如，自动递增的主键）。
13. **#pragma db transient**:
    - 指定成员变量不应该被持久化到数据库中。
14. **#pragma db type("type_name")**:
    - 指定成员变量的数据库类型。
15. **#pragma db convert("converter")**:
    - 指定用于成员变量的自定义类型转换器。
16. **#pragma db pool("pool_name")**:

- 指定用于数据库连接的连接池。

17. **#pragma db trigger("trigger_name")**:
    - 指定在插入、更新或删除操作时触发的触发器。

```C++
namespace odb{
  namespace mysql {
    //mysql 连接池对象类
    class LIBODB_MYSQL_EXPORT new_connection_factory:
    public connection_pool_factory{
        connection_pool_factory (
            std::size_t max_connections = 0,
            std::size_t min_connections = 0,
            bool ping = true)
    };
  }
  //操作句柄类
  class LIBODB_EXPORT database {
      //新增数据
      persist (T& object);
      //更新数据
      void update (T& object);
      void erase (T& object);
      unsigned long long erase_query (const std::string&);
      unsigned long long erase_query (const odb::query<T>&);
      result<T> query (const std::string&);
      result<T> query (const odb::query<T>&, bool cache = true);
      typename result<T>::pointer_type query_one(const
odb::query<T>&);

      virtual transaction_impl* begin () = 0;
  };
  //事务操作类
  class LIBODB_EXPORT transaction {
      transaction (transaction_impl*, bool make_current = true);
      void commit ();
      void rollback ();
  };
  //针对可能为空的字段封装的类似于智能指针的类型
  template <typename T>
```

```
  class nullable {
      typedef T value_type;
      T&       get ();
      const T& get () const;

      T*       operator-> ();
      const T* operator-> () const;

      T&       operator* ();
      const T& operator* () const;
  }
  //针对查询结果所封装的容器类
  template <typename T>
  class result: result_base<T, class_traits<T>::kind> {
      result ()
      result (const result& r)
      iterator begin ()
      iterator end ()
      size_type size ()
      bool empty ()
  };
  //针对查询封装的条件类
  class LIBODB_EXPORT query_base {
      explicit query_base (const std::string& native)
      const clause_type& clause ()
  };
  namespace mysql
  {
    template <typename T>
    class query: public query_base,
                 public query_selector<T, id_mysql>::columns_type
{
        query (const std::string& q)
        query (const query_base& q)
        query (bool v)
    }
  }
  template <typename T>
  class query<T, mysql::query_base>: public mysql::query<T>  {
      query (bool v)
      query (const std::string& q)
      query (const mysql::query_base& q)
  }
}
```

使用样例

在 odb 的使用中，我们最关注的其实是三个点，

- 一个是增删改的基础操作
- 一个是基于原生 sql 语句的查询，
- 一个是基于多表连接的复杂查找

能解决这三个问题，就能满足数据库的基本功能操作。

```
C++
#include <odb/database.hxx>
#include <odb/mysql/database.hxx>
#include <gflags/gflags.h>

#include "person.hxx"
#include "person-odb.hxx"

DEFINE_int32(msyql_port, 0, "mysql 服务器端口");
DEFINE_string(msyql_host, "127.0.0.1", "mysql 服务器地址");
DEFINE_string(msyql_db, "bite_im", "mysql 默认连接库名称");
DEFINE_string(msyql_user, "root", "mysql 默认连接用户名");
DEFINE_string(msyql_passwd, "bite_666", "mysql 默认连接密码");
DEFINE_int32(msyql_conn_pool, 3, "mysql 连接池中连接最大数量");

class MysqlClient
{
public:
    static std::shared_ptr<odb::database> create(
        const std::string& user,
        const std::string& passwd,
        const std::string& db_name,
        const std::string& host,
        int port,
        size_t conn_pool_count) {
        //初始化连接池
        std::unique_ptr<odb::mysql::connection_factory> pool(
            new
odb::mysql::connection_pool_factory(conn_pool_count));
        //构造 mysql 操作句柄
        std::shared_ptr<odb::database> db(new
odb::mysql::database(
            user.c_str(), passwd.c_str(), db_name.c_str(),
```

```
host.c_str(), port, 0, "utf8", 0, std::move(pool)));
        return db;
    }
    //开始事务，并返回事务对象
    static std::shared_ptr<odb::transaction> transaction(const
std::shared_ptr<odb::database> &db) {
        return std::make_shared<odb::transaction>(db->begin());
    }
    //通过事务对象提交事务
    static void commit(const std::shared_ptr<odb::transaction> &t)
{
        return t->commit();
    }
    //通过事务对象回滚事务
    static void rollback(const std::shared_ptr<odb::transaction>
&t) {
        return t->rollback();
    }
};

class StudentDao {
    public:
        StudentDao(const std::string& user,
        const std::string& passwd,
        const std::string& db_name,
        const std::string& host,
        int port,
        size_t conn_pool_count):_db(MysqlClient::create(
            user, passwd, db_name, host, port, conn_pool_count))
        {}
        void append(Student& stu) {
            try{
                auto t = MysqlClient::transaction(_db);
                _db->persist(stu);
                MysqlClient::commit(t);
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
        }
        void update(const Student& stu) {
            try{
                auto t = MysqlClient::transaction(_db);
                _db->update(stu);
                MysqlClient::commit(t);
```

```
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
        }
        void remove(const std::string &name) {
            try{
                auto t = MysqlClient::transaction(_db);
                typedef odb::query<Student> query;
                _db->erase_query<Student>(query::name == name);
                MysqlClient::commit(t);
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
        }
        std::vector<std::string> select(int n) {
            std::vector<std::string> res;
            try{
                auto t = MysqlClient::transaction(_db);
                typedef odb::query<all_name> query;
                typedef odb::result<all_name> result;
                std::string cond = "order by age desc limit ";
                cond += std::to_string(n);
                result r(_db->query<all_name>(cond));
                for (auto i(r.begin()); i != r.end(); ++i) {
                    res.push_back(i->name);
                }
                MysqlClient::commit(t);
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
            return res;
        }
    private:
        std::shared_ptr<odb::database> _db;
};

class ClassesDao {
    public:
        ClassesDao(const std::string& user,
        const std::string& passwd,
        const std::string& db_name,
        const std::string& host,
        int port,
        size_t conn_pool_count):_db(MysqlClient::create(
```

```
            user, passwd, db_name, host, port, conn_pool_count))
        {}
        void append(const std::string &id, const std::string
&desc) {
            Classes c(id, desc);
            try{
                auto t = MysqlClient::transaction(_db);
                _db->persist(c);
                MysqlClient::commit(t);
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
        }
        std::vector<class_student> select(const std::string &id) {
            std::vector<class_student> res;
            try{
                auto t = MysqlClient::transaction(_db);
                typedef odb::query<class_student> query;
                typedef odb::result<class_student> result;
                std::string cond = "Classes.classes_id=" + id;
                result r(_db->query<class_student>(cond));
                for (auto i(r.begin()); i != r.end(); ++i) {
                    res.push_back(*i);
                }
                MysqlClient::commit(t);
            } catch(std::exception &e) {
                std::cout << e.what() << std::endl;
            }
            return res;
        }
    private:
        std::shared_ptr<odb::database> _db;
};

int main(int argc, char *argv[])
{
    // ptime p = boost::posix_time::time_from_string("2017-05-22
09:09:39");
    // Student stu1("张三", 14, p, "11111");
    // Student stu2("李四", 15, p, "11111");
    // Student stu3("王五", 12, p, "11111");
    // Student stu4("赵六", 13, p, "22222");
    // Student stu5("刘七", 14, p, "22222");
```

```
    // StudentDao student_tb(FLAGS_msyql_user, FLAGS_msyql_passwd,
    //     FLAGS_msyql_db, FLAGS_msyql_host,
    //     FLAGS_msyql_port, FLAGS_msyql_conn_pool);
    // student_tb.append(stu1);
    // student_tb.append(stu2);
    // student_tb.append(stu3);
    // student_tb.append(stu4);
    // student_tb.append(stu5);

    // stu4.age(16);
    // student_tb.update(stu4);

    // auto res = student_tb.select(3);
    // for (auto r : res) {
    //     std::cout << r << std::endl;
    // }
    ClassesDao classes_tb(FLAGS_msyql_user, FLAGS_msyql_passwd,
        FLAGS_msyql_db, FLAGS_msyql_host,
        FLAGS_msyql_port, FLAGS_msyql_conn_pool);
    classes_tb.append("11111", "一年级一班");
    classes_tb.append("22222", "一年级二班");

    auto res = classes_tb.select("11111");
    for (auto r : res) {
        std::cout << r.stu << "\t" << r.classes << std::endl;
    }
    return 0;
}
```

官网链接

- https://codesynthesis.com/products/odb/doc/manual.xhtml
- https://codesynthesis.com/products/odb/
- https://codesynthesis.com/products/odb/doc/install-build2.xhtml