<https://www.cnblogs.com/qicosmos/p/3342037.html>

先简单说说ORM的优点：

1. 提高开发效率，减少重复劳动，只和业务实体打交道，由业务实体自动生成sql语句，不用手写sql语句。
2. 简单易用, 可维护性好。
3. 隔离数据源，使得我们更换数据源时不用修改代码。

## SmartDB基础库

　　java和c#中有很多ORM框架，如c#中的entity framework、Linq to Sql、NHibernate等，java中有Hibernate、Mybatis等等，其它各种开源的ORM库多如牛毛。这应该得益于托管语言在这方面的优势。然而，c++的世界中，却鲜有ORM框架，c++中比较知名的ORM库应该是ODB了，但是ODB的使用比较麻烦，需要三个库，写的ORM代码还需要ODB专门编译一下（不知道我说的是否确切），总之使用起来比较麻烦，让人望而却步。我的项目中本来只需要用sqlite，但是考虑到后期可能会更换数据库，所以想搞一个ORM库，但时间又比较紧，觉得短时间内搞不定ODB，于是放弃，选择最简单稳妥的作法，先只封装sqlite，目标是封装成简洁，统一的接口，即使到后面更换数据库时，我也不需要改应用层的代码，只要替换底层数据库封装模块就行了。在这个目标的指引下我很快完成了基础库SmartDB的封装，如下是SmartDB的操作接口。

[复制代码](javascript:void(0);)

bool Open(string dbName);//打开数据库

bool Close();//关闭数据库

bool Excecute(const std::string && query); //直接执行sql语句

bool ExcecuteBulk(const Args && ... args); //批量操作的sql语句

bool Prepare(const std::string && query); //批量操作之前的准备

bool Excecute(const std::string && query, const Args && ... args); //执行带占位符的sql语句

R ExecuteScalar(const std::string &query, const Args &... args); //返回函数执行的一个值, 执行简单的汇聚函数，如select count(\*), select max(\*)等

bool GetTable(const string &query, T &table, TableRange\* tableRange, const Args &... args); //返回一个通用的表结构

int GetRowCount(const string &query, const TableRange\* tableRange, const Args &... args);//获取表的总行数

[复制代码](javascript:void(0);)

　　这些接口是直接接受sql语句和参数列表的，所以理论上它可以完成所有的数据库操作。目前内部只支持sqlite，我自测了一下基础库的性能。在我的双核6G的台式机下测试了一下插入速度和读取速度，一个八字段的表（int和doble类型），批量插入速度大概为25W条/s，读取速度大概为11W条/s。这些已经满足了我的需求，但是我觉得可以在这个基础上做得更多，即支持ORM，我期望我的ORM能达到类似于entity framework的调用方式，比ODB更好用（比如智能提示、链式调用等），而性能又不至于降低太多。

[复制代码](javascript:void(0);)

EF中查询方式

DemoDBEntities context = new DemoDBEntities();

ObjectQuery<BlogMaster> query = context.CreateObjectSet<BlogMaster>().Where("it.UserId > @UserId",new ObjectParameter("UserId", 6))

.OrderBy("it.UserId desc");

List<BlogMaster> list = query.ToList();

[复制代码](javascript:void(0);)

## SmartDBEX ORM库

　　c++中ORM库的实现思路: ORM的关键是如何将业务实体转成sql语句以及表的原始数据如何转成业务实体，它们横在数据库和实体之间的两道鸿沟，因此，如何通过一个结构体去自动生成sql脚本是第一个需要解决的问题，如何将原始表转成业务实体是第二个需要解决的问题。我觉得问题的关键在结构体，我需要结构体提供一些信息给我，比如，字段类型，字段名，字段值等信息，这在c#或者java中可以通过反射就可以得到，但c++没有反射，是无法自动就得到这些信息的，不过C++虽然没有托管语言的这些优势，但却有自己的优势--模板元，可以通过模板元来获取这些信息，通过这些信息我就可以自动生成sql脚本了；第二个问题则是通过boost的variant和结构体中的赋值函数来解决。

　　总之，我通过在结构体上做文章，最终达到了我的目标，而且我会保证结构体不会复杂，尽可能少的让用户去写代码，只需要提供几个简单的函数就可以实现ORM了。我是在基础库SmartDB之上扩展了ORM功能，最终的ORM库为SmartDBEX, 它内部是调用了SmartDB实现具体功能。这样做的目的是即可使用操作sql的接口，又可使用ORM接口，用户根据自己选择即可。

　　我在台式机上测了一下SmartDBEX的性能：还是之前的表，批量插入速度为25W条/s，读取并转成实体列表的速度为2.3W条/s。比较满意的是批量插入速度没有降低，差强人意的是读取实体列表的速度降低了不少，性能损耗在两个地方：原始表的字段和实体的字段匹配和原始表字段赋值给实体。没办法，ORM这里必须要损耗一些性能。2.3W条/s的读取速度对一般应用来说已经可以满足要求了。

## 如何使用SmartDB和SmartDBEX

### SmartDB的测试程序

[复制代码](javascript:void(0);)

/\*创建表\*/

void TestCreateTable(SmartDB& db)

{

const string sqlcreat = "CREATE TABLE if not exists TestInfoTable(ID INTEGER NOT NULL, KPIID INTEGER, CODE INTEGER, V1 INTEGER, V2 INTEGER, V3

REAL);";

if (!db.Excecute(sqlcreat))

return;

}

/\*单条插入\*/

void TestInsert(SmartDB& db)

{

const string sqlinsert = "INSERT INTO TestInfoTable(ID, KPIID, CODE, V1, V2, V3) VALUES(1, 2, 3, 4, 5, 6.032);";

if (!db.Excecute(sqlinsert))

return;

const string sqlinsert1 = "INSERT INTO TestInfoTable(ID, KPIID, CODE, V1, V2, V3) VALUES(?, ?, ?, ?, ?, ?);";

int n = 2;

string str = "3.1423";

if (!db.Excecute(sqlinsert1, n, n, n, n, n, str))

return;

}

/\*批量插入, 启用事务\*/

void TestBulkInsert(SmartDB& db)

{

const string sqlinsert = "INSERT INTO TestInfoTable(ID, KPIID, CODE, V1, V2, V3) VALUES(?, ?, ?, ?, ?, ?);";

timer t;

Transaction transaction(&db); //开始事务

db.Prepare(sqlinsert); //准备sql，含占位符

bool ret = false;

for (size\_t i = 0; i < 1000000; i++)

{

ret = db.ExcecuteBulk(i, i, i, i, i, i);

if (!ret)

break;

}

if (ret)

transaction.Commit(); //提交事务

else

transaction.RollBack(); //回滚

cout << t.elapsed() << endl;

}

[复制代码](javascript:void(0);)

### SmartDBEX的测试程序

[复制代码](javascript:void(0);)

void TestORM(SmartDB& db)

{

SmartDBEx dbex(db);

//创建表

auto r = dbex.Create<TestInfo>();

//查询

auto& query = dbex.Query<TestInfo>().Where(Var(TestInfo::CODE < 10 and TestInfo::ID < 10));

vector<TestInfo> v;

bool b = dbex.Get<TestInfo>(v);

//获取总行数

int count = dbex.Count<TestInfo>().Scalar();

cout << count << endl;

//单条插入

r = dbex.Insert(t);

//批量插入

TestInfo t = {1,2,3,4,5,6};

Transaction tans(&db);

r = dbex.Prepare<TestInfo>();

for (size\_t i = 0; i < 1000000; i++)

{

r = dbex.BulkInsert(t);

}

r = tans.Commit();

//删除

r = dbex.Delete<TestInfo>()();

}

[复制代码](javascript:void(0);)

　　这里我们不妨对比一下ODB的调用方式，ODB的查询方式：

[复制代码](javascript:void(0);)

typedef odb::query<person> query;

typedef odb::result<person> result;

{

transaction t (db->begin ());

result r (db->query<person> (query::age > 30));

for (result::iterator i (r.begin ()); i != r.end (); ++i)

{

cout << "Hello, " << i->first () << "!" << endl;

}

t.commit ();

}

[复制代码](javascript:void(0);)

SmartDBEX的查询方式:

auto& query = dbex.Query<TestInfo>().Where(Var(TestInfo::CODE < 10 and TestInfo::ID < 10));

vector<TestInfo> v;

bool b = dbex.Get<TestInfo>(v);

呵呵，有点像吧，但哪个更直观简洁呢？

## 有待完善的地方

　　牺牲了好几个周末的时间做这个ORM，主要是ORM的关键技术的思路和实现花了不少时间，今天终于做完了第一个版本，算是对前段时间的思考和研究做个总结和交代吧。不过由于是业余时间开发，时间和精力有限，ORM接口目前只有一些基本功能，还不太完善，而且目前内部只支持sqlite，至于其它数据库的支持，等到后面再继续完善了。另外，库内部没有采用缓存和延迟加载等技术，不是不能加，是暂时没有时间和精力去继续完善了，而且性能已经满足我的要求了，优化完善的工作来日方长吧；后期还可以考虑支持结构体通过文件配置的方式生成，进一步增强灵活性。

## 欢迎使用

　　欢迎大家下载测试工程和源码并试用，发现问题请报告给我，我好完善。也期待有人愿意在此基础上和我一起完善它，使它成为一个优秀的c++ORM库，让c++开发人员的日子变得更美好，让c++的世界更加绚丽多彩。

[源码下载地址](http://files.cnblogs.com/qicosmos/SmartDB_V1.00.rar)

c++11 boost技术交流群：296561497，欢迎大家来交流技术。

一点梦想：尽自己一份力，让c++的世界变得更美好！

分类: [C++](http://www.cnblogs.com/qicosmos/category/466797.html),[c++11 使用c++11改进我们的程序系列](http://www.cnblogs.com/qicosmos/category/490693.html" \t "_blank),[C++11工程实践](http://www.cnblogs.com/qicosmos/category/520451.html)