天津大学计算机科学与技术学院

Mysql++基本用法报告及使用规范

Mysql++ 2.2.3 在 Doboo II 服务端中使用规范标准

常兴龙

2007-6-1

本文档为 Mysql ++ Lib 在 Doboo II 中的使用文档草案，本版本为使用规范基本版，将在开发过程日臻完善，作为工程开发标准统一及后续开发人员的参考规范

第一部分 Mysql++基础篇

Mysql++是把 Mysql 提供的 C 库的一个 C++封装库，它用 STL（Standard Template Language）开发并编写，并为 C++开发程序员提供象操作 STL 容器一样方便的操作数据库的一套机制。

本部分将以Mysql ++本身机制及用法为主导，讨论Mysql++的用法。有关Mysql++ 的开发库有很多可用版本，关于Mysql 的CppApi 可以参看http://sourceforge.net/project/showfiles.php?group_id=7869 ，但Mysql++于 2005 推出面向新的开发包，且能较好的与Mysql配合，所以建议使用Mysql++去代替CppApi，对应网上资源地址为http://tangentsoft.net/mysql++ ，本文以后者为准进行介绍。

第一章 Mysql++基本用法

Mysql++为 Mysql 的 C-Api 的再次封装，所用封装技术标准为 STL，因此可以在各支持 C++平台下编译并重用。其中的 SSQL 标准提供了与 Hibernate 相同的封装思想，使 Table 与 Class 可以做一一映射，以实现用 STL 方便地操作数据表。

1.1 Mysql++各版本比较

在http://tangentsoft.net/mysql++/ 资源列表中，有重多版本可供下载。

主要版本介绍如下：

1. Mysql++ 2.2.3 版:此版本为 2007.04.17 生成版，为截止到目前为止的最新版，如果工作是从头开始，没有代码需要兼容，下载此版本。
2. Mysql++ 2.2.1 版：此版本为 2.1.X 版中的最终版本，由于 2.2.X 版本兼容此版，故对于 2.2.1 版，没有必要下载。
3. Mysql++ 1.71 版：此版本是可以用 VC 6.0 编译通过的唯一官方版本，如果已有工程为 VC6.0 建立的，且没有升级的打算，用该版的 Mysql++。

除了 Mysql++ 1.7.1 版外，其余版本的编译条件至少为 Visual C++.NET (a.k.a. Visual Studio 2002, a.k.a. Visual C++ 7.0)，如果是 Linux 或 MAC 系统则至少为 GCC3.x。

1.2 Mysql++ 2.2.3 在Windows环境下安装说明

从http://tangentsoft.net/mysql++/releases/mysql++-2.2.3.tar.gz下载安装包，该包展开后的结构如下：

```
MYSQL++-2.2.3
├──config
├──Debug
├──doc
├──examples
├──lib
└──Release
```

其中，Doc 中文档为 Mysql++库的使用帮助文档，包含 Html 与 PDF 两个版本。Examples 中分别为 MFC 与 wForms 两个简单的示例工程。但建议把 Mysql++编译成 Dll 或 Lib，而非直接应用在工程中。

在 Mysql++2.2.3 根目录中，有面向 Windows 的安装文件 install.bat,用 UltraEdit 打开，转换为 Dos 格式.运行以下命令以安装 Mysql++：

```
C:\Mysql++2.2.3\Install lib
```

此时，Mysql++2.2.3 应能顺利安装在 C:\Mysql++目录中,在 Visual Studio Include 搜索路径中加入 C:\mysql++\include,在 Visual Studio Lib 搜索路径中加入 C:\mysql++\lib,同时，把安装在 Mysql 数据库下的 libmysql.lib 等库文件（有 Debug 或 Release 版，根据您的需要考贝）考入 c:\mysql++\lib 下，安装至此完成。

第二章 Mysql++在Visual studio环境中使用指南

2.1 Mysql++常用类概览

Transaction 类，Transaction 类为数据库操作提供了事务机制，保证一系列操作的原子性。

Connection 类，Connection 类是对 Mysql 数据库操作的基础。

Query 类，该类从 std::stringstream 继承，因此，程序员可以象操作 Stream 一样操作 Query 类，以生成正确的 Sql 语句，如:

```
Query query;
query<<"drop table test";
query.execute();
```

同样，程序员也可以使用模版，以及 SSQL 来生成正确的 Query 以便对 Mysql 进行操作，这些技巧将在稍后章节进行介绍。在 Mysql++中，与 C#，Java 语言提供的机制不同，Query 类包含了如 Update，Delete 等操作。

Result 类，该类主要存储数据库查询存储结果,该类一般不直接构建，而是做为由 Query 类返回集的容器。

Row 类，该类对应数据表中的一行。

其它比较常用的类，如：UseRes , ResNSel 类将在使用其示例时介绍。

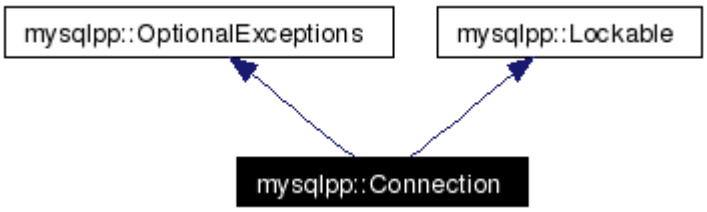
2.2 Mysql++用法详述

Mysql++默认的工作空间名称为 mysqlpp,如果本章不做特别说明，默认已加入以下语句:

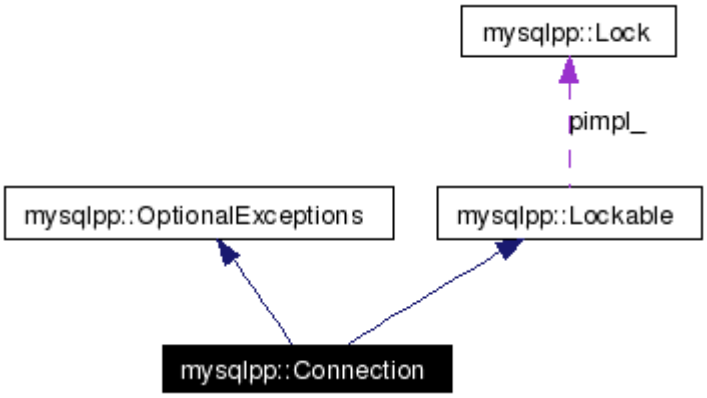
```
#include <string>
#include <iostream>
#include “mysql++.h”
using namespace mysqlpp;
using namespace std;
```

2.2.1 建立 Mysql 连接

Connection 类的继承关系图如下图所示：



Connection 类的简要协作关系图如下图所示:



用法：

(1)创建一个 Connection 对象，但不建立连接至服务器的连接：

```
mysqlpp::Connection::Connection ( bool te = true )
```

其中，te 为真时，当 onnection 出错时抛出异常,为假时，不抛出异常。

(2)创建连接对象，并建立连接至服务器的连接：

```
Connection(const char *db, const char *host="", const char *user="", const char *passwd="", uint port=0, my_bool compress=0, unsigned int connect_timeout=60, cchar *socket_name=0, unsigned int client_flag=0)
```

(3)以某个已有连接为模版，建立一个新连接

Connection (const Connection &other)

(4)用 Mysql 结构创建新连接

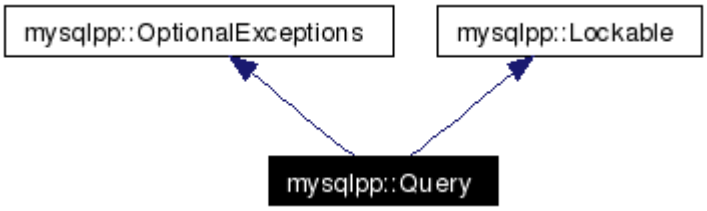
connect (const MYSQL &mysql)

示例：

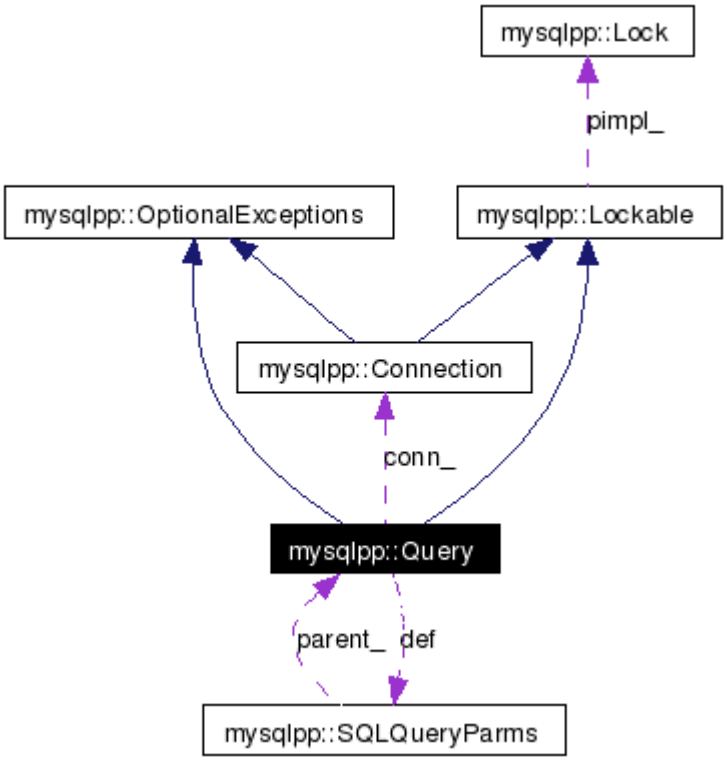
```
mysqlpp::Connection con(false);
con.connect("test","","root","123456");
```

2.2.2 使用 Query 类进行查询

Query 类的继承关系图如下图所示：



Query 类的协作关系图如下图所示：



用法：

Query 类也有几种构造函数，

（1）创建一个依赖于特定 Connection 连接的 Query

Query (Connection *c, bool te=true),如果 te 为 TRUE 时，如果数据库出错，则抛出异常。

（2）从已有的 Query 对象复制构造

Query (const Query &q)
但在实际操作中，很少用 new 对此类进行操作，一般是由以下语句返回:
Query query=con.query();
此类中，相比 Connection，更重要的并不是如何构建它，而是如何使用它.

(1)可以象操作 Stream 数据一样的操作 Query 对象，如：

query << "select * from test"

(2)对于无返回值，或只有一个自增返回值的，可以使用 Query 的 execute 方法.

- ✧ bool mysqlpp::Query::exec (const std::string & str)
  如:exec(new string("delete from test where id='1'");返回布尔值来表示成功还是失败
- ✧ ResNSel mysqlpp::Query::execute (const char * str, size_t len)
  如果在所执行的语句中，有 null 字符，则需用此种格式，其中的 len 为整个语句的长度,
  返回的类型为 ResNSel，ResNSel 包含以下几个主要字段,
  Info,执行结果的附加信息。
  rows,本次执行所影响的行数。
  Insert_id,取得新插入行的自增号的 id
  success,本次执行是否成功
- ✧ ResNSel mysqlpp::Query::execute (const char * str)
  执行 C-Style 的语句，一般以 0 结尾
- ✧ ResNSel mysqlpp::Query::execute (const SQLString & str)
  把 str 提交到模版中，或执行一个 C++ string.
- ✧ ResNSel mysqlpp::Query::execute ( ) [inline]
  批量执行已嵌入的语句。

上述这些用法，主要是针对没有任何返回值的 SQL 语句进行操作。如，插入，删除等。利用 ResNSel 足以得到足够的信息。如果只需要知道操作是否成功，使用 exec()操作。

(3) 对于有返值的查询操作，需要分情况使用。

store()操作返回 Result 集,如果需要使用自定义的容器结构，如 vector , set 等，则可使用 storein()操作，对于大数据集则需要使用 use()进行查询，此操作返回 ResUse 类的对象，用 fectch_row()取得每一行。

使用小技巧：如何为查询信息添加引号和处理特殊字符？

在 Sql 语句的实际操作中，有很多时候需要为 Sql 语句中的特定数据加单引号，如:

```
Select * from test where objName='Vc Object'
```

因为“Vc Object”’中含有空格，所以它必须用引号括起来。在这种情况下，一般可以做以下处理:

```
string s="Vc Object";
Query query = con.query();
query << "select * from test where objName = " << quote_only << s;
```

如果把上条语句中 *quote_only* 改为 *quote* ， 则可以对含有特殊字符的语句进行处理，如:

```
select * from test where objName = 'Lily's Object'
```

此外，还有 *mysqlpp:escape*，只是处理特殊字符，一般在处理 *Blog* 字段（如对图象存取）时使用.

2.2.3 SSQLS 在 Mysql ++中的使用

SSQLS(Specialized SQL Structures) 是 Mysql ++中一种强大的功能，使用 SSQLS，程序员可以方便地对结构进行存取，如:

```
Query query = con.query();
query << "select * from user ";
vector <User> res;
query.storein( res );
```

要使用此功能，必须对 User 结构进行定义，对 User 结构定义的是通过一系列宏指定的，宏的格式如下所示：

```
sql_create_#(NAME, COMPCOUNT, SETCOUNT, TYPE1, ITEM1, ... TYPE#, ITEM#)
```

第一个#是指当前结构中的变量个数，NAME 是指结构的名字，COMPCOUNT 从 1 开始此参数指定参与比较的字段，SETCOUNT 是指需要初始化的参数个数，它可以是 0 表示无需参数初始化，后续的每对 TYPE 和 ITEM 是类型和变量的对应对。
如：

```
sql_create_5(User,
1, 5,                  //1 表示 name 将被做为比较字段， 5 个字段都要初始化
                       //User user(...) 中的 5 个参数都要指定，由于宏的实现及
                       //C++的限制，COMPCOUNT 与 SETCOUNT 不能相同
mysqlpp::sql_char, name,
mysqlpp::sql_bigint,userObjectId ,
```

```
mysqlpp::sql_double, weight,
mysqlpp::sql_double, classId,
mysqlpp::sql_date, sdate)
```

使用 SSQLS 插入一条记录，代码如下:

```
User user( ‘xl’ , ’23444’ , ‘70’ , ’40’ , ‘2007-6-16’ );
query.insert( user );
query.execute();
```

使用 SSQLS 修改一条记录，代码如下：

```
query << “select * from user where name = ‘xl’ “;
Result res = query.store();

If ( res.empty() )
{
   throw sqlpp::BadQuery(  “ no found! “ );

}

User user = res.at(0);

User orgin_user = user ; //建立一个Copy，使得Mysql ++ 知道更改处所在
user.name = ‘cxl’;
query.update ( orgin_user , user);
query.execute();
```

使用小技巧: 数据库中的字段类型在 Mysql++中如何对应？

在 Mysql++中，对数据库的字段类型做了一一对应，如数据库的 Datetime 类型对应 Mysql++中的 mysqlpp::sql_datetime 类型。所有的对应情况在 sql_types.h 中定义，有必要时查询该头文件。

应该特别指出的是，对于数据库中的 null，Mysql++使用模版进行处理（参看 **lib/type_info.cpp**），如:

```
mysqlpp::Null<mysqlpp::sql_tinyint_unsigned> myfield;
myfield = mysqlpp::null;
```

应当指出，Null 类型不能转换为 C++中的常规类型，如果试图转换将得到 **BadNullConversion** 的异常,把 Null 值插入流中，将得到 (NULL),如果不希望这样，可以在声明时加入第二个参数，如下:

```
mysqlpp::Null<unsigned char, mysqlpp::NullisZero> myfield;
```

```
myfield = mysqlpp::null;
```

2.2.4 在 Mysql++中使用事务

所谓事务，是指把一系列相关操作做为一个原子操作来进行，其中任一个子操作的失败会导致事务的回滚(Roll Back),所有的操作都成功后，提交(Commit)操作把该系列操作结果存储到数据库中。

在 Mysql++中，Transaction 类专门处理事务操作，如下：

```
mysqlpp::Connection con(mysqlpp::use_exceptions);//打开Exception机制
mysqlpp::Transaction trans(con);
try
{
  //一系列操作
 con.commit();
}
catch(Exception & er)
{
 con.rollback();
}
```

特别需要注意的是，如果在出 trans 的作用域时，commit 行为没有发生，则会执行 rollback 操作。

2.2.5 取得数据库表的 Meta 信息

数据库表的 Meta 信息，在 Result 中有相应的存储，使用方法如下:

```
Query query = con.query();
query << "select * from test";
cout << query.preview() << endl; //预览Sql语句
Result res = query.store();
cout << "The Result size is :" << res.size() <<endl;

cout << "Query info :" <<endl;
for (unsigned int i = 0 ; i < res.names.size() ;  i++)
{
    cout  << setw(2) << i << setw(15)  <<  res.names.c_str()
        << setw(15) << res.types(i).sql_name
     << setw(20) << res.types(i).name << endl;
}
```

2.2.6 使用 Query 模板进行操作

使用 Mysql++特定的占位符可以方便地建立 Query 的操作模板，如下：

```
query << "select (%2:clsid, %3:wgt) from test where %1:nme = %0q:what";
query.parse();
```

其中，占位符的格式如下所示:

```
%###(modifier)(:name)(:)
```

###，是从 0 开始的三位数，Modifier 的详细描述见下表:

Modifier	作用
%	打印%
""	不进行转义处理或引号处理
q	对于字符串，字母或 Mysql 指定要做引号处理的字符，使用 C Api 中的 mysql-escapestring 进行处理同时进行转义处理
Q	对上述规则中需做处理的字符做引号处理，但不做转义处理
r	即使是数字，也做引号处理,同时转义
R	即使是数字，也做引号片时，但不转义

表 2 -2-6 Modifer 的可能取值及作用

：name 是为了引用占用符方便从而指定的一个名称。
在取回记录时，使用以下语句:

```
Result res = query.store("cxl", "name", "classid", "weight")
```

这将产生以下的 SQL 语句：

```
select (classid , weight) from test where name = “cxl”
```

使用默认值，设定默认值采用以下的方式：

```
query.def[0] = “cxl”;
query.def[1] = “name”;
query.def[2] = “classid”;
//query.def[3] = “weight”;
```

或者

```
query.def[“what”] = “cxl”;
query.def[“nme”] = “name”;
query.def[“clsid”] = “classid”;
//query.def[“wgt”] = “weight”;
```

然后，在调用 Query 的 store 方法时，传入未设定的参数。

```
Result res = query.store();
//Result res = query.store(“weight”);
```