**说明：如何生成线程式客户端**

客户端库总是线程安全的。最大的问题在于从套接字读取的net.c中的子程序并不是中断安全的。或许你可能希望用自己的告警中断对服务器的长时间读取，以此来解决问题。如果为SIGPIPE中断安装了中断处理程序，套接字处理功能应是线程安全的。

为了避免连接中断时放弃程序，[**MySQL**](http://lib.csdn.net/base/14)将在首次调用mysql\_server\_init()、mysql\_init()或mysql\_connect()时屏蔽SIGPIPE。如果你打算使用自己的SIGPIPE处理程序，首先应调用mysql\_server\_init()，然后安装你的处理程序，

在较旧的发布在我方网站上（http://www.[**MySQL**](http://lib.csdn.net/base/mysql).com/）的二进制版本中，未用线程安全的选项对客户端库进行正常编译（默认情况下，编译的Windows二进制版本是线程安全的）。较新的二进制分发版应是正常的和线程安全的客户端库。

为了获得能从其他线程中断客户端的线程式客户端，并在与MySQL服务器通信时设置超时，应使用“-lmysys”、“-lmystrings”和“-ldbug”库，以及服务器使用的net\_serv.o代码。

如果你不需要使用中断或超时，可编译线程安全客户端库（mysqlclient\_r）并使用它。。请参见[25.2节，“MySQL C API”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c)。在该情况下，不必担心net\_serv.o对象文件或其他MySQL库。

使用线程式客户端并打算使用超时或中断时，可更好地利用thr\_alarm.c文件中的子程序。如果你正在使用来自mysys库的子程序，唯一需要记住的事是首先调用my\_init()！请参见[25.2.11节，“C API线程函数介绍”](http://dev.mysql.com/doc/refman/5.1/zh/apis.html#c-thread-functions)。

对于除mysql\_real\_connect()外的所有函数，在默认情况下它们均是线程安全的。在下面的说明中，介绍了编译线程安全客户端库的方法，以及以线程安全方式使用它的方法。（下面关于mysql\_real\_connect()的说明实际上也适用于mysql\_connect()，但由于mysql\_connect()已不再被重视，总应尽量使用mysql\_real\_connect()）。

要想使mysql\_real\_connect()成为线程安全的，必须用下述命令再次编译客户端库：

shell> **./configure --enable-thread-safe-client**

它创建了线程安全客户端库libmysqlclient\_r。（假定你的[**操作系统**](http://lib.csdn.net/base/operatingsystem)有线程安全的gethostbyname\_r()函数）。按照连接，该库是线程安全的。可遵循下述警告，使两个线程共享相同的连接：

·        在相同的连接上，两个线程不能同时将查询发送到MySQL服务器。尤其是，必须确保在mysql\_query()和mysql\_store\_result()之间，没有使用相同连接的其他线程。

·        很多线程均能访问由mysql\_store\_result()检索的不同结果集。

·        如果使用了mysql\_use\_result，务必确保无其他线程正在使用相同的连接，直至关闭了结果集为止。然而，对于线程式客户端，最好是共享相同的连接以使用mysql\_store\_result()。

·        如果打算在相同的连接上使用多个线程，必须在mysql\_query()和mysql\_store\_result()调用组合上拥有互斥锁。一旦mysql\_store\_result()准备就绪，可释放锁定，其他线程可在相同的连接上执行查询。

·        如果使用POSIX线程进行编程，可使用pthread\_mutex\_lock()和pthread\_mutex\_unlock()来建立并释放互斥锁。

如果你有1个调用MySQL函数的线程，而该函数未创建与MySQL[**数据库**](http://lib.csdn.net/base/14)的连接，就需了解下述事宜：

调用mysql\_init()或mysql\_connect()时，MySQL会为调试库使用的线程创建与线程相关的变量（尤其）。

在线程调用mysql\_init()或mysql\_connect()之前，如果调用了MySQL函数，该线程将没有所需的线程类变量，而且你很可能或早或晚结束于内核转储。

要想使这些操作平稳工作，需要采取下述措施：

1.   如果程序在调用mysql\_real\_connect()之前需要调用任何其他MySQL函数，请在启动程序时调用my\_init()。

2.   调用任何MySQL函数之前，在线程处理程序中调用mysql\_thread\_init()。

3.   在线程中，调用pthread\_exit()之前请调用mysql\_thread\_end()。这样，就能释放MySQL线程类变量使用的内存。

将客户端链接到libmysqlclient\_r时，如果存在未定义的符号，可能会出错。在大多数情况下，其原因在于，未将线程库包含在link/compile行上。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
基于C API的MySQL数据库多线程访问方法

2009-04-28 9:40

摘要：目前，数据库在各行各业中广泛应用。在众多商业数据库软件中，SQL SERVER 和ORACLE被较多的使用，因此这两个数据库软件的价格也较昂贵。本文主要介绍MySQL数据库在单线程或多线程程序环境下使用C API访问MySQL数据库方法，并给出了相应代码和分析。该数据库属于开源数据库，具有较高的成熟度，并且对于社区版本可以免费使用，因此对于需要使用C API访问数据库的项目开发，可降低开发成本。  
　　关键词：MySQL；C API；多线程   
　　中图分类号：TP311文献标识码：A文章编号：1009-3044(2007)16-30904-02   
　　Based on C API MySQL Database Multi-threaded Access Methods   
　　YU Cheng-gong   
　　(Zhejiang Pharmaceutical College,Ningbo 315100,China)   
　　Abstract:Currently,the database in all walks of life were widely used.In many commercial database software,SQL Server and Oracle are used more and therefore their prices are more expensive. This paper describes the MySQL database in single-threaded or multi-threaded programming environment using C API access to the MySQL database, and gives the corresponding code and analysis. The database is open source database, with a higher maturity level, and the community version can be used for free. It satisfies the need to use the C API access to the database of project development, and reduces development costs.  
　　Key words:MySQL;C API;multi-threaded   
　　   
　　1 引言   
　　   
　　随着社会信息化的深入，数据库在社会各个领域被广泛应用。在这些数据库应用项目开发过程中，需要做两方面的决策：1. 使用何种数据库软件；2. 采用何种方式访问数据库。数据库软件的选择面比较宽，在目前众多商业数据库软件中，SQL SERVER 和ORACLE被较多的使用，当然这两个数据库软件的价格也较昂贵，本文选择可免费使用的MySQL数据库社区版本，MySQL库属于开源数据库，具有较高的成熟度和可靠性。数据库的访问方式有很多，可以使用ODBC、DAO、ADO等方法，这些方法简单直接但是效率不高，不适合大型复杂的系统使用，例如网络游戏的数据库系统开发需要考虑同时大量的数据库访问，因此访问的效率非常重要。基于C程序语言的高效率，使用C API访问数据库可以提高数据库的访问效率。基于以上两点，本文将介绍基于C API的MySQL数据库访问方法，给出在单线程和多线程程序环境下的具体代码和分析。   
　　   
　　2 建表   
　　   
　　为了方便说明数据库的访问，先建立一个数据库表TestTable，可以使用SQL语句创建该表，也可以使用MySQL提供的图形界面来创建。数据库表中字段如下：   
　　该数据库表使用最常见的用户名和密码作为字段，本文将通过该表来实现不同程序环境下基于C API的数据库的访问方法。   
　　   
　　3 单线程程序的数据库访问   
　　   
　　单线程应用程序访问MySQL数据库相对简单，其过程包含以下几步：   
　　(1)初始化MySQL库   
　　(2)初始化数据库连接句柄   
　　(3)连接数据库   
　　(4)通过SQL语句操作数据库并处理相应数据   
　　(5)关闭数据库连接   
　　(6)结束MySQL库   
　　通过这五个步骤即可实现数据库的访问，具体代码和分析如下：   
　　//在main主函数中添加代码   
　　//1.定义访问数据库所需变量   
　　MYSQL \* myData;   
　　MYSQL\_RES \* res;   
　　MYSQL\_ROW row;   
　　//2. 初始化MySQL库和数据库连接句柄   
　　myData = mysql\_init((MYSQL\*) 0);   
　　//3. 连接数据库，MYSQL\_IP和MYSQL\_PORT表示数据库的IP和端口   
　　// MYSQL\_ACCOUNT, MYSQL\_PASSWORD表示数据库连接的帐号和密码   
　　//MYSQL\_DBNAME表示所要访问的数据库名   
　　mysql\_real\_connect( myData, MYSQL\_IP, MYSQL\_ACCOUNT, MYSQL\_PASSWORD, MYSQL\_DBNAME, MYSQL\_PORT,NULL, 0 )  
　　//4. 通过SQL语句操作数据库并处理相应数据   
　　//4.1新建用户名为abcdef，密码为123456的记录   
　　mysql\_query(myData, "insert into TestTable value(‘abcdef’,’ 123456’)");   
//4.2显示所有记录   
　　//查询所有记录   
　　mysql\_query(myData, "select \* from TestTable");   
　　//将查询结果保存到res中   
　　res = mysql\_store\_result( myData ) ;   
　　//逐条显示记录   
　　Int j = 0;   
　　while ( row = mysql\_fetch\_row( res ) ) {//获取一条记录   
　　j = mysql\_num\_fields( res ) ;//获取每条记录的字段数   
　　for ( k = 0 ; k < j ; k++ )   
　　printf( “%s”, row[k] ) ;   
　　printf( “\n”) ;   
　　}   
　　//释放res   
　　mysql\_free\_result( res ) ;   
　　//5. 关闭数据库连接   
　　mysql\_close( myData );   
　　//6. 结束MySQL库   
　　mysql\_library\_end();   
　　关于代码的几点说明：   
　　(1)定义变量中的三个数据结构为访问MySQL所需，MYSQL结构表示一个数据库连接的句柄，其中包含了数据库连接所需的参数，MYSQL\_RES结构表示数据库访问中一个查询的返回结果，MYSQL\_ROW结构表示返回结构中的一条记录；  
　　(2)获取查询结果res并处理完毕，必须释放res，否则会造成内存泄露   
　　(3)在单线程时，步骤初始化MySQL库和数据库连接句柄可合并， 由mysql\_init()来处理。该函数会自动调用函数mysql\_library\_init()来初始化MySQL库，同时初始化连接句柄。  
　　   
　　4 多线程环境下的数据库访问   
　　   
　　多线程环境下的数据库访问需要保证线程安全。Windows版本的MySQL C API函数都是线程安全的，除了mysql\_library\_init()，而我们刚才的代码中使用的mysql\_init()函数会自动调用函数mysql\_library\_init()来初始化MySQL库，因此在多线程环境下，需要不同的初始化代码和清理代码。具体过程如下：  
　　(1)在主函数中调用mysql\_library\_init（）来初始化MySQL库；   
　　(2)启动各数据库访问线程   
　　(3)主函数等待各个线程的结束   
　　(4)调用mysql\_library\_end ()清理MySQL库。   
　　其中数据库访问线程的代码和单线程数据库访问代码类似，但是需要如下变化：   
　　(1)单线程中的第2步初始化MySQL库和数据库连接句柄，不能再使用mysql\_init()，代码应作如下修改：   
　　//初始化线程   
　　my\_init();   
　　mysql\_thread\_init();   
　　//初始化myData   
　　myData = malloc(sizeof(MYSQL));   
　　memset(&myData, 0, sizeof(MYSQL))   
　　(2)上述初始化myData，只是将myData所有成员设为0，如果有需要可以根据具体情况设置该结构成员的值，例myData->reconnect= 1，其作用是设置数据库连接属性为重连接，即当数据库连接断开时，自动重新连接；  
　　(3)单线程中的第6步不在需要，改为结束线程的清理工作，即调用mysql\_thread\_end()函数。   
　　   
　　5 总结   
　　   
　　综上所述，使用C API访问MySQL数据库在不同线程环境下的区别主要在于初始化和访问结束后清理代码，也就是说除了初始化和清理代码，MySQL提供给我们的C API函数都是线程安全的。最后需要有一点说明，使用C API访问数据库可以提高数据库的访问效率，但并不是所有的数据库项目都适合这种方式，该方式适合需要大量实时并发处理的数据库项目，例如网络游戏的数据库项目，对于有此需求的数据

顶

0

踩