DBCP数据库连接池与代理模式

1. DBCP：**数据库连接池(DataBase connection pool)** 。DBCP(DataBase connection pool),[数据库连接池](http://baike.baidu.com/view/84055.htm)。是 apache 上的一个 java 连接池项目，也是 tomcat 使用的连接池组件。单独使用dbcp需要2个包：commons-dbcp.jar,commons-pool.jar由于建立数据库连接是一个非常耗时[耗资源](http://baike.baidu.com/view/630544.htm)的行为，所以通过连接池预先同数据库建立一些连接，放在内存中，应用程序需要建立数据库连接时直接到连接池中申请一个就行，用完后再放回去。
2. 代理模式**：Connection的代理模式**：

由于在java中只是提供了**Connection接口**，并没有具体的实现类。具体的**Connection的实现类**是由各个数据库实现的，但是不同的数据库提供的实现类不相同，如果想重写Connection的某些方法，需要实现具体数据库的Connection实现类，这样不能满足各种不同的数据库，为了满足可以实现任何数据库的需求，出现了代理模式。

1. **所谓代理模式**就是创建一个Connection接口的代理实现类MyConnection，这个类里面大多数方法仍然是各大数据库自己的方法，只不过是添加了些新的方法。例如：

public class MyConnection implements Connection {

private Connection realConnection;

public MyConnection(Connection realConnection) {

this.setRealConnection(realConnection);

}

public Connection getRealConnection() {

return this.realConnection;

}

public void setRealConnection(Connection realConnection) {

this.realConnection = realConnection;

}

}

4. 数据源和连接池：



1. 常用的开源实现：



1. 示例:

public class Test {

public static void main(String[] args) {

**ConnectionPool connpool = new ConnectionPool();**

**for (int i = 0; i < 16; i++) {**

**Connection connection = connpool.getConnection();**

**System.out.println(connection);**

**connpool.free(connection);**

}

}

}

public class ConnectionPool **implements DataSource**{

//实现接口DataSource需要实现很多方法，这里省略了。

private LinkedList<Connection> connPool = new LinkedList<Connection>();

public ConnectionPool(){

for (int i = 0; i < 5; i++) {

connPool.addLast(DriverUtil.getConnection());

}

}

public Connection getConnection() {

**return connPool.removeFirst();//取出连接**

}

public void free(Connection conn) {

**connPool.addLast(conn);//把连接从新放进去**

}

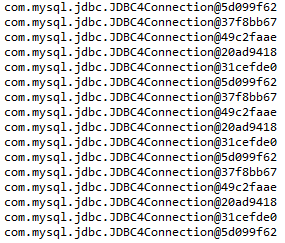
public Connection creatNewConn() {

return DriverUtil.getConnection();

}

}

执行结果：



**可以看出，5个连接可以得到重复利用。**

1. 对上面的案例进行改进：

如果有多个线程同时访问，需要考虑**同步安全问题**。

public static void main(String[] args) throws SQLException {

ConnectionPool connpool = new ConnectionPool();

for (int i = 0; i < 16; i++) {

Connection connection = connpool.getConnection();

System.out.println(connection);

connpool.free(connection);

}

}

public class ConnectionPool implements DataSource {

private static int maxCount = 7;

private static int currentCount = 0;

private LinkedList<Connection> connPool = new LinkedList<Connection>();

public ConnectionPool() {

for (int i = 0; i < 5; i++) {

connPool.addLast(DriverUtil.getConnection());

this.currentCount++;

}

}

public Connection getConnection() throws SQLException {

**synchronized (connPool) {**

if (connPool.size() > 0)

return connPool.removeFirst();

else if (this.currentCount < this.maxCount) {

this.currentCount++;

return this.creatNewConn();

}

else

throw new SQLException("已径达到了最大的限度。");

}

}

public void free(Connection conn) {

connPool.addLast(conn);

}

public Connection creatNewConn() {

return DriverUtil.getConnection();

}

}

**结果**：

