# 1. DbUtils组件

commons-dbutils 是 Apache 组织的一个开源类库，它对JDBC进行了简单的封装。开发者使用DbUtils能提高开发效率，简化代码。

Apache的DbUtils项目地址：<http://commons.apache.org/proper/commons-dbutils/> 。可自行下载工具包，然后在Java项目中引入“commons-dbutils-1.6.jar”这个JAR包。当然，不能忘了引入数据库驱动包。

使用DbUtils组件的一般步骤：

（1）创建一个QueryRunner对象；

（2）调用QueryRunner对象的方法来操作数据库，如update()方法，方法需要传递数据库连接、预编译的SQL和参数等信息。

（3）调用DBUtils的静态方法close()来关闭数据库连接。

下面进行详细讲解。

## 1.1 用QueryRunner的update()方法执行增删改操作

实例代码：

|  |
| --- |
| **package** com.zhang.test;  **import** org.apache.commons.dbutils.DbUtils; **import** org.apache.commons.dbutils.QueryRunner;  **import** java.sql.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  String url = **"jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8"**;  Class.*forName*(**"com.mysql.jdbc.Driver"**);  *// 同样，先得到数据库连接对象* Connection conn = DriverManager.*getConnection*(url, **"root"**, **"123456"**);  *// 创建QueryRunner对象* QueryRunner qr = **new** QueryRunner();  *// 调用qr的update()方法* **int** rows = qr.update(conn, **"insert into account values(?, ?);"**, **"王五"**, 1000);  System.***out***.println(**"影响行数："** + rows);  *// 关闭连接* DbUtils.*close*(conn);  } } |

其中update()方法可执行增删改操作，常用的重载（也是上述用的）：

public int update(Connection conn, String sql, Object... params)

参数1是数据库连接，参数2是预编译的SQL，最后一个可变参数，就是依次设置预编译SQL中的各个参数，可传递Object类型，DbUtils自行处理。发现使用DBUtils相比之前方便很多。

## 1.2 用batch()方法执行批处理操作

批处理操作也适合于增删改。使用batch()方法，原型为：

public int[] batch(Connection conn, String sql, Object[][] params)；同样参数1是数据库连接，参数2是SQL，参数3这时是一个二维数组，二维数组中有几个一维数组，就表示执行几次SQL，而一维数组中的数据就是设置SQL参数值的。

例子：

|  |
| --- |
| **package** com.zhang.test;  **import** org.apache.commons.dbutils.DbUtils; **import** org.apache.commons.dbutils.QueryRunner;  **import** java.sql.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  String url = **"jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8"**;  Class.*forName*(**"com.mysql.jdbc.Driver"**);  Connection conn = DriverManager.*getConnection*(url, **"root"**, **"123456"**);  QueryRunner qr = **new** QueryRunner();  qr.batch(conn, **"insert into account values(?,?)"**, **new** Object[][]{  {**"张三"**, 1000},  {**"李四"**, 2000},  {**"王五"**, 3000}  });  DbUtils.*close*(conn);  } } |

## 1.3 使用query()方法查询数据库

QueryRunner对象提供了query()方法用于获得SQL查询结果。query()的原型是：

|  |
| --- |
| **public** <T> T query(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params)  参数1：数据库连接  参数2：预编译SQL  参数3：ResultSetHandler对象  参数4：预编译SQL的参数，若SQL无参数，则不写此参数，因为query()有三个参数的重载。  重点讲解的就是参数3。ResultSetHandler<T>是一个接口，因此我们要用它的一个实现类。实现类需要重写下列方法：  T handle(ResultSet rs)，即在此方法中，我们能拿到DbUtils传递过来的ResultSet结果集，我们自行处理即可。  query()和handler()中都使用了泛型T，说明我们实现ResultSetHandler时使用什么类型，那么query()的返回值就是什么类型。我们只要在handler()中操作结果集，返回我们想要的数据类型即可。  比如，查询的SQL是select \* from Account，那么返回的数据类型应该是List<Account>，我们循环遍历ResultSet数据来封装一下实体类即可。  再比如，查询的SQL是select \* from Employee where eid=?，由于是主键查询，返回的只应该是一个实体类Employee类(若无数据就返回null)，此时就不用循环结果集，直接用if(rs.next())...即可。 |

下面就实现上面的案例。

案例1：实现select \* from Account。

（1）Account实体类

|  |
| --- |
| **package** com.zhang.bean;  **public class** Account {  **private** String **name**;  **private int money**;   **public** Account(String name, **int** money) {  **this**.**name** = name;  **this**.**money** = money;  }  **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getMoney() {  **return money**;  }  **public void** setMoney(**int** money) {  **this**.**money** = money;  }  @Override  **public** String toString() {  **return "Account{"** +  **"name='"** + **name** + **'\''** +  **", money="** + **money** +  **'}'**;  } } |

（2）AccountHandler实现ResultSetHandler接口

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.bean.Account; **import** org.apache.commons.dbutils.ResultSetHandler;  **import** java.sql.ResultSet; **import** java.sql.SQLException; **import** java.util.ArrayList; **import** java.util.List;  *// 由于query()的返回值要是List<Account>，因此这里的泛型参数就用List<Account>* **public class** AccountHandler **implements** ResultSetHandler<List<Account>> {  @Override  **public** List<Account> handle(ResultSet rs) **throws** SQLException {  List<Account> accList = **new** ArrayList<>();  **while**(rs.next()) {  accList.add(**new** Account(rs.getString(1), rs.getInt(2)));  }  **return** accList;  } } |

（3）主类

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.bean.Account; **import** org.apache.commons.dbutils.DbUtils; **import** org.apache.commons.dbutils.QueryRunner;  **import** java.sql.\*; **import** java.util.List;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  String url = **"jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8"**;  Class.*forName*(**"com.mysql.jdbc.Driver"**);  Connection conn = DriverManager.*getConnection*(url, **"root"**, **"123456"**);  QueryRunner qr = **new** QueryRunner();  *// query()的参数3就用AccountHandler对象，并且返回值是List<Account>。* List<Account> accountList = qr.query(conn, **"select \* from account"**, **new** AccountHandler());  System.***out***.println(accountList);  DbUtils.*close*(conn);  } } |

案例2：实现select \* from Employee where eid=?

（1）Employee实体类

|  |
| --- |
| **package** com.zhang.bean;  **public class** Employee {  **private int eid**; *// 编号* **private** String **ename**; *//姓名* **public** Employee(**int** eid, String ename) {  **this**.**eid** = eid;  **this**.**ename** = ename;  }  **public int** getEid() {  **return eid**;  }  **public void** setEid(**int** eid) {  **this**.**eid** = eid;  }  **public** String getEname() {  **return ename**;  }  **public void** setEname(String ename) {  **this**.**ename** = ename;  }  @Override  **public** String toString() {  **return "Employee{"** +  **"eid="** + **eid** +  **", ename='"** + **ename** + **'\''** +  **'}'**;  } } |

（2）SingleEmployeeHandler实现ResultSetHandler接口

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.bean.Employee; **import** org.apache.commons.dbutils.ResultSetHandler;  **import** java.sql.ResultSet; **import** java.sql.SQLException;  *// 返回的就是一个Employee实体，因此直接用<Employee>* **public class** SingleEmployeeHandler **implements** ResultSetHandler<Employee> {  @Override  **public** Employee handle(ResultSet rs) **throws** SQLException {  Employee emp = **null**;  **if**(rs.next()) {  emp = **new** Employee(rs.getInt(1), rs.getString(2));  }  **return** emp;  } } |

（3）主类

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.bean.Employee; **import** org.apache.commons.dbutils.DbUtils; **import** org.apache.commons.dbutils.QueryRunner;  **import** java.sql.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  String url = **"jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8"**;  Class.*forName*(**"com.mysql.jdbc.Driver"**);  Connection conn = DriverManager.*getConnection*(url, **"root"**, **"123456"**);  QueryRunner qr = **new** QueryRunner();  *// query()的参数3就用SingleEmployeeHandler* Employee emp = qr.query(conn, **"select \* from employee where eid = ?"**, **new** SingleEmployeeHandler(), 12);  System.***out***.println(emp);  DbUtils.*close*(conn);  } } |

也可以直接给参数3传递一个匿名内部类，实现其中的方法。此方式适用于此ResultSetHandler只需要使用一次的时候。例子：

|  |
| --- |
| **package** com.zhang.test;  **import** com.zhang.bean.Employee; **import** org.apache.commons.dbutils.DbUtils; **import** org.apache.commons.dbutils.QueryRunner; **import** org.apache.commons.dbutils.ResultSetHandler;  **import** java.sql.\*;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  String url = **"jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8"**;  Class.*forName*(**"com.mysql.jdbc.Driver"**);  Connection conn = DriverManager.*getConnection*(url, **"root"**, **"123456"**);  QueryRunner qr = **new** QueryRunner();  *// query()的参数3就用SingleEmployeeHandler* Employee emp = qr.query(conn, **"select \* from employee where eid = ?"**, **new** ResultSetHandler<Employee>() {  @Override  **public** Employee handle(ResultSet rs) **throws** SQLException {  Employee emp = **null**;  **if**(rs.next()) {  emp = **new** Employee(rs.getInt(1), rs.getString(2));  }  **return** emp;  }  }, 12);  System.***out***.println(emp);  DbUtils.*close*(conn);  } } |

上述使用DbUtils实现了简单的对象关系映射（对象关系映射，即ORM，Object Relational Mapping），把数据库中的数据封装到对象中，实现用面向对象的方法操作数据。以后还会学习更强大的ORM框架，比如Hibernate和Mybatis。

# 2. 优化数据库连接：使用连接池

我们之前的案例中，会频繁打开和关闭一个数据库连接对象。而且在实际应用中，同时可能有多个数据库连接在使用，比如有n个用户正在查询自己的信息。

数据库连接资源是十分宝贵的，频繁打开关闭连接会耗费资源，并且实际中我们还需要管理多个连接资源。为了对数据库连接进行有效的管理，需要使用连接池技术：预先创建多个数据库连接（即连接池），当需要用到连接时，就从池中取出一个使用，用完后再放回池中。

可定义数据库连接池的“初始化连接数”、“最大连接数”等，并记录“当前连接数”，这样我们也能自行编写实现连接池的代码：当需要新连接时，若池中有空闲的，则直接从池中取；否则就新建连接，但是总连接数不能超过最大连接数，否则创建不成功，只能等待其他连接使用完毕才能获得连接。

实际上JDK已经规定了连接池的一系列规范，比如：连接池需要实现接口javax.sql.DataSource，这个DataSource是数据源，用于给外部提供数据库连接，这样就更加简化了我们获得数据库连接的方式。

自己实现连接池比较麻烦，一般使用第三方开源的连接池。常用的有DBCP和C3P0连接池。

注意，DBCP和C3P0只负责提供DataSource和管理数据库连接，并不是像DBUtils那样对JDBC操作进行了封装，因此为了简化数据库操作，可以结合连接池使用DBUtils。即连接池给DBUtils提供DataSource数据源。可以使用QueryRunner的构造函数重载QueryRunner(DataSource ds)，来通过数据源获得连接，这样update()、batch()和query()方法都不需要传递数据库连接对象Connection了，DBUtils提供了这样的重载。而且使用DataSource也不需要考虑手动关闭Connection连接，因为这些事是连接池做的。这样结合使用十分方便。

如果不使用DBUtils，单纯的使用DataSource对象的getConnection()方法也能轻松得到Connection连接对象。

## 2.1 DBCP连接池

DBCP是Apache的开源连接池实现，要使用DBCP，除了应该在项目中引入DBCP的jar包外，还应引入DBCP的两个依赖jar包，即commons-pool.jar和commons-logging.jar，这都是Apache的产品，可在Apache官网寻找下载。

DBCP可通过properties配置文件来配置数据库连接的参数，比如在src下新建db.properties文件，内容如下：

|  |
| --- |
| *# 配置数据库驱动类* **driverClassName**=**com.mysql.jdbc.Driver** *# 连接的URL* **url**=**jdbc:mysql://localhost:3306/mytest?useSSL=false&useUnicode=true&characterEncoding=utf8** *# 用户名* **username**=**root** *# 密码* **password**=**123456** *# 设置初始化连接数* **initialSize**=**5** *# 最大连接数* **maxActive**=**10** *# 最大空闲时间。若超过这个时间此连接还是空闲，则回收此连接* **maxIdle**=**6000** |

使用配置的好处就是可根据需要来配置数据库，而不需要重新更改代码并且重新编译代码。而之前的方式都是“硬编码”，即把所有的信息都在代码中写死的。我们也可通过自行通过IO方式来实现自己读取配置文件。以后大部分信息都是在配置文件中配置的，这样使得系统更加灵活。

在程序中，使用DBCP核心类BasicDataSourceFactory得到DataSource数据源。以下是结合DBUtils使用的案例：

|  |
| --- |
| **package** com.zhang.test;  **import** org.apache.commons.dbcp2.BasicDataSourceFactory; **import** org.apache.commons.dbutils.QueryRunner;  **import** javax.sql.DataSource; **import** java.io.InputStream; **import** java.util.Properties;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  *// 加载db.properties配置文件。为何使用ClassLoader.getSystemResourceAsStream()？参见本章附录。* InputStream in = ClassLoader.*getSystemResourceAsStream*(**"db.properties"**);  *// 创建Properties对象装载配置文件* Properties props = **new** Properties();  *// 装载* props.load(in);  *// 使用BasicDataSourceFactory的方法，根据props对象得到数据源* DataSource ds = BasicDataSourceFactory.*createDataSource*(props);  *// 可使用数据源创建QueryRunner* QueryRunner qr = **new** QueryRunner(ds);  **int** row = qr.update(**"insert into account values(?,?)"**, **"王五"**, 10000);  System.***out***.println(row);  *// 数据库连接无需自己关闭，由连接池管理* } } |

在程序运行中，DBCP会自动维护连接池。当然程序结束之后，维护的连接也会随之关闭。我们可以在QueryRunner那一行设置一个断点，调试程序。当程序运行停在此断点处时，在MySQL中执行“show processlist”查看当前所有的数据库连接，可以看到连接池管理的所有连接的状态。

## 2.2 C3P0连接池

C3P0也是常用的连接池。C3P0的网址: <http://www.mchange.com/projects/c3p0/> 。可在对应的Github上下载最新的C3P0文件：<https://github.com/swaldman/c3p0/releases> 。

使用C3P0需要引入lib目录下的jar包，如果不是连接Oracle数据库，则不用引入“c3p0-oracle-thin-extras-0.9.5.2.jar”文件，只需引入另两个文件。

C3P0使用xml文件来配置数据库连接。C3P0默认自动加载classpath路径下的c3p0-config.xml文件，因此我们在项目src目录下新建一个c3p0-config.xml文件，因为项目编译时，src目录下的配置文件会自动被复制到classpath路径下。配置内容如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"** *?>* <**c3p0-config**>  *<!-- default-config节点中的是默认配置，C3P0默认会加载此配置 -->* <**default-config**>  *<!-- 配置连接URL。注意，在XML中，连接字符串中的&符号需要转义成&amp;否则出错 -->* <**property name="jdbcUrl"**>  jdbc:mysql://localhost:3306/mytest?useSSL=false**&amp;**useUnicode=true**&amp;**characterEncoding=utf8  </**property**>  <**property name="driverClass"**>com.mysql.jdbc.Driver</**property**>  <**property name="user"**>root</**property**>  <**property name="password"**>123456</**property**>  <**property name="initialPoolSize"**>3</**property**>  <**property name="maxPoolSize"**>6</**property**>  <**property name="maxIdleTime"**>6000</**property**>  </**default-config**>  *<!-- 在default-config同级节点下，也能配置其他数据库连接，比如SQL Server的 -->  <!-- 这里只是示例 -->* <**named-config name="sqlServer"**>  <**property name="jdbcUrl"**>jdbc:sqlserver://localhost:1433;databaseName=mytest</**property**>  <**property name="driverClass"**>com.microsoft.sqlserver.jdbc.SQLServerDriver</**property**>  <**property name="user"**>sa</**property**>  <**property name="password"**>123456</**property**>  <**property name="initialPoolSize"**>3</**property**>  <**property name="maxPoolSize"**>6</**property**>  <**property name="maxIdleTime"**>6000</**property**>  </**named-config**> </**c3p0-config**> |

C3P0直接用核心类ComboPooledDataSource创建对象，该对象就能当做DataSource使用，因为此类最终还是实现了DataSource接口。代码：

|  |
| --- |
| **package** com.zhang.test;  **import** com.mchange.v2.c3p0.ComboPooledDataSource; **import** org.apache.commons.dbutils.QueryRunner;  **public class** Demo {  **public static void** main(String[] args) **throws** Exception {  *// 创建ComboPooledDataSource对象，实际上就是数据源* ComboPooledDataSource ds = **new** ComboPooledDataSource();  *//ComboPooledDataSource默认使用XML中default-config节点中的信息。  //如果想使用非默认的配置节点信息，则创建此对象时要传入该配置的名字，即：  //ComboPooledDataSource dataSource = new ComboPooledDataSource("sqlserver");    // 用ds创建QueryRunner* QueryRunner qr = **new** QueryRunner(ds);  qr.update(**"insert into account values(?,?)"**, **"张三"**, 130000);  System.***out***.println(**"OK"**);  } } |

程序运行中，控制台会输出一系列的红色信息，这并不是警告和错误，而是C3P0提供的日志信息。日志中提供了时间和其他的信息，日志也能提供调试、警告和错误等信息，在开发过程中使用能清晰的知道系统运行的状况。

总结：推荐以后的应用都使用连接池，即方便开发又提高效率。

# 附1：JDBC连接SQL Server数据库

步骤如下：

（1）下载适用于SQL Server的JDBC驱动，网址：<https://www.microsoft.com/zh-CN/download/details.aspx?id=11774> 。下载tar.gz压缩包即可。下载前可看看页面上的说明。在“系统要求”中可知道，若使用Java8，则可使用Sqljdbc42.jar包，支持的数据库版本是SQL Server2008——2016。

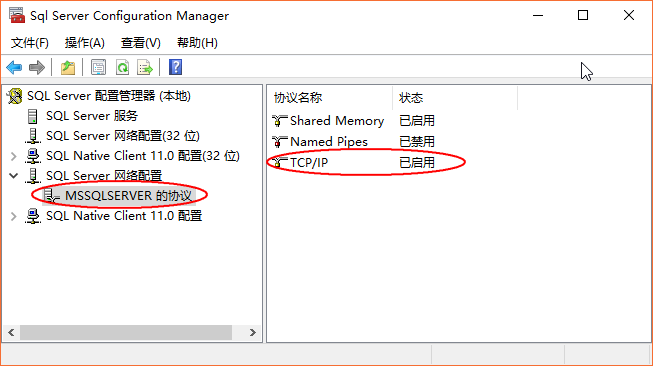
（2）解压压缩包，拿到jre8目录中的jar文件，引用到项目中。

（3）然后即可正常写Java代码。注意的是：数据库连接url格式为：jdbc:sqlserver://IP地址:端口;databaseName=数据库名；需要加载的驱动类为：com.microsoft.sqlserver.jdbc.SQLServerDriver。SQLServer默认端口是1433。

（4）Java代码中使用sa账户和密码验证。这时可成功连接数据库。示例代码：

|  |
| --- |
| Class.*forName*(**"com.microsoft.sqlserver.jdbc.SQLServerDriver"**); Connection conn = DriverManager.*getConnection*(**"jdbc:sqlserver://localhost:1433;databaseName=MyTest"**, **"sa"**, **"123456"**); |

有时不能正常连接数据库，则检查SQL Server服务是否打开了，并且还要保证SQL Server的网络配置正常：打开“Sql Server Configuration Manager”，设置“MSSQLSERVER的协议”中，TCP/IP为“启用”状态，如图：



若使用外界的计算机访问本地数据库，即远程访问，那么本机也需要在防火墙中配置1433端口为“允许入站”，方法和MySQL配置时方法一致。

另外，学习过.Net的人还会使用Windows连接验证方式，这样无需使用密码，但是不推荐这样做，并且这样做较麻烦。步骤如下：

（1）数据库连接url使用Windows验证，即加上“integratedSecurity=true”，比如：“jdbc:sqlserver://localhost:1433;databaseName=MyTest;integratedSecurity=true”。

（2）Java代码中无需写用户名和密码，即：“DriverManager.getConnection(url)”即可。

（3）仅仅上述步骤会出错。还需要向Windows系统注册一个“sqljdbc\_auth.dll”的文件。该文件在压缩包的auth文件夹中已经提供了。根据自己电脑位数选择合适的版本注册。有两种注册的方法：1.把该dll所在文件夹添加为Path环境变量；2.把该dll文件添加到“C:/Windows/System32”文件夹中。

温馨提示：连接SQL Server数据库时，也能使用DBUtils组件和连接池技术（DBCP和C3P0）。只是需要注意下述问题：

使用DBUtils时，SQL Server不能正确识别处理预编译的SQL语句，需要在创建QueryRunner对象时，提供一个knowParaType参数，此参数类型是boolean，设置为true即可。比如：

QueryRunner qr = new QueryRunner(true)；

当使用数据库连接池提供的DataSource数据源时，也别忘了提供此参数：

QueryRunner qr = new QueryRunner(ds, true);