数据库连接池

JDBC 开发流程

- 加载驱动（只需要加载一次）
- 建立数据库连接（Connection）
- 执行 SQL 语句（Statement）
- ResultSet 接收结果集（查询）
- 断开连接，释放资源

数据库连接对象是通过 DriverManager 来获取的，每次获取都需要向数据库申请获取连接，验证用户名和密码，

执行完 SQL 语句后断开连接，这样的方式会造成资源的浪费，数据连接资源没有得到很好的重复利用。

可以使用数据库连接池解决这一问题。

数据库连接池的基本思想就是为数据库建立一个缓冲池，预先向缓冲池中放入一定数量的连接对象，当需要获取数据库连接的时候，只需要从缓冲池中取出一个对象，用完之后再放回到缓冲池中，供下一次请求使用，做到了资源的重复利用，允许程序重复使用一个现有的数据库连接对象，而不需要重新创建。

当数据库连接池中没有空闲的连接时，新的请求就会进入等待队列，等待其他线程释放连接。

数据库连接池实现

JDBC 的数据库连接池使用 javax.sql.DataSource 接口来完成的，DataSource 是 Java 官方提供的接口，使用的时候开发者并不需要自己来实现该接口，可以使用第三方的工具，C3P0 是一个常用的第三方实现，实际开发中直接使用 C3P0 即可完成数据库连接池的操作。

1、导入 jar 包。

传统方式拿到的 Connection

```
com.mysql.cj.jdbc.ConnectionImpl@557caf28
```

C3P0 拿到的 Connection

```
com.mchange.v2.c3p0.impl.NewProxyConnection@4988d8b8
```

2、代码实现

```
package com.southwind.test;

import com.mchange.v2.c3p0.ComboPooledDataSource;
```

```
import java.beans.PropertyVetoException;
import java.sql.Connection;
import java.sql.SQLException;

public class DataSourceTest {
    public static void main(String[] args) {
        try {
            //创建C3P0
            ComboPooledDataSource dataSource = new ComboPooledDataSource();
            dataSource.setDriverClass("com.mysql.cj.jdbc.Driver");
            dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8");
            dataSource.setUser("root");
            dataSource.setPassword("root");
            Connection connection = dataSource.getConnection();
            System.out.println(connection);
            //还回到数据库连接池中
            connection.close();
        } catch (PropertyVetoException e) {
            e.printStackTrace();
        } catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

实际开发，将 C3P0 的配置信息定义在 xml 文件中，Java 程序只需要加载配置文件即可完成数据库连接池的初始化操作。

1、配置文件的名字必须是 c3p0-config.xml

2、初始化 ComboPooledDataSource 时，传入的参数必须是 c3p0-config.xml 中 named-config 标签的 name 属性值。

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>

    <named-config name="testc3p0">

        <!-- 指定连接数据源的基本属性 -->
        <property name="user">root</property>
        <property name="password">root</property>
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/library?
useUnicode=true&amp;characterEncoding=UTF-8</property>

        <!-- 若数据库中连接数不足时，一次向数据库服务器申请多少个连接 -->
```

```
        <property name="acquireIncrement">5</property>
        <!-- 初始化数据库连接池时连接的数量 -->
        <property name="initialPoolSize">20</property>
        <!-- 数据库连接池中的最小的数据库连接数 -->
        <property name="minPoolSize">2</property>
        <!-- 数据库连接池中的最大的数据库连接数 -->
        <property name="maxPoolSize">40</property>

    </named-config>

</c3p0-config>
```

```
package com.southwind.test;

import com.mchange.v2.c3p0.ComboPooledDataSource;

import java.beans.PropertyVetoException;
import java.sql.Connection;
import java.sql.SQLException;

public class DataSourceTest {
    public static void main(String[] args) {
        try {
            //创建C3P0
            ComboPooledDataSource dataSource = new
ComboPooledDataSource("testc3p0");
            Connection connection = dataSource.getConnection();
            System.out.println(connection);
            //还回到数据库连接池中
            connection.close();
        } catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

DBUtils

DBUtils 可以帮助开发者完成数据的封装（结果集到 Java 对象的映射）

1、导入 jar 包

ResultHandler 接口是用来处理结果集，可以将查询到的结果集转换成 Java 对象，提供了 4 种实现类。

- BeanHandler 将结果集映射成 Java 对象 Student
- BeanListHandler 将结果集映射成 List 集合 List
- MapHandler 将结果集映射成 Map 对象
- MapListHandler 将结果集映射成 MapList 结合

```
public static Student findByDBUtils(Integer id){
  Connection connection = null;
  Student student = null;
  try {
    connection = dataSource.getConnection();
    String sql = "select * from student";
    QueryRunner queryRunner = new QueryRunner();
    List<Map<String,Object>> list = queryRunner.query(connection,sql,new
MapListHandler());
    for (Map<String,Object> map:list){
      System.out.println(map);
    }
  } catch (SQLException e) {
    e.printStackTrace();
  } finally {
    try {
      connection.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
  return student;
}
```