第 30 天 DBUtils 和连接池

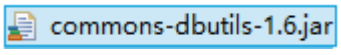
今日内容介绍

- DBUtils
- 连接池

第1章 DBUtils

如果只使用 JDBC 进行开发，我们会发现冗余代码过多，为了简化 JDBC 开发，本案例我们讲采用 apache commons 组件一个成员：DBUtils。

DBUtils 就是 JDBC 的简化开发工具包。需要项目导入 commons-dbutils-1.6.jar 才能够正常使用 DBUtils 工具。



1.1 概述

DBUtils 是 java 编程中的数据库操作实用工具，小巧简单实用。
DBUtils 封装了对 JDBC 的操作，简化了 JDBC 操作，可以少写代码。
Dbutils 三个核心功能介绍

- QueryRunner 中提供对 sql 语句操作的 API.
- ResultSetHandler 接口，用于定义 select 操作后，怎样封装结果集.
- DbUtils 类，它就是一个工具类,定义了关闭资源与事务处理的方法

1.2 QueryRunner 核心类

- update(Connection conn, String sql, Object... params) ，用来完成表数据的增加、删除、更新操作
- query(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params) ，用来完成表数据的查询操作

1.3 QueryRunner 实现添加、更新、删除操作

- update(Connection conn, String sql, Object... params) ，用来完成表数据的增加、删除、更新操作

1.3.1 添加

```
public void insert(){
    try {
        //获取一个用来执行 SQL 语句的对象  QueryRunner
        QueryRunner qr = new QueryRunner();

        String sql = "INSERT INTO zhangwu(name,money,parent) VALUES(?,?,?)";
        Object[] params = {"股票收入", 5500, "收入"};
        Connection conn = JDBCUtils.getConnection();
        int line = qr.update(conn,sql,params);// 用来完成表数据的增加、删除、更新操作
        //结果集处理
        System.out.println("line = " + line);

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

1.3.2 更新

```
public void update(){
    try {
        //创建一个 QueryRunner 对象，用来完成 SQL 语句的执行
        QueryRunner qr = new QueryRunner();
        //执行 SQL 语句
        String sql = "UPDATE zhangwu SET money = money+1000 WHERE name=?";
        Object[] params = {"股票收入"};
        Connection conn = JDBCUtils.getConnection();
        int line = qr.update(conn, sql, params);
        //结果集的处理
        System.out.println("line="+line);

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

1.3.3 删除

```
public void delete(){
    try {
```

```
            //创建一个 QueryRunner 对象，用来完成 SQL 语句的执行
            QueryRunner qr = new QueryRunner();
            //执行 SQL 语句
            String sql = "DELETE FROM zhangwu WHERE name = ?";
            Object[] params = {"股票收入"};
            Connection conn = JDBCUtils.getConnection();
            int line = qr.update(conn, sql, params);
            //结果集的处理
            System.out.println("line="+line);

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
```

1.4QueryRunner 实现查询操作

- query(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params) ，用来完成表数据的查询操作

1.4.1 ResultSetHandler 结果集处理类

ArrayHandler	将结果集中的第一条记录封装到一个 Object[]数组中，数组中的每一个元素就是这条记录中的每一个字段的值
ArrayListHandler	将结果集中的每一条记录都封装到一个 Object[]数组中，将这些数组在封装到 List 集合中。
BeanHandler	将结果集中第一条记录封装到一个指定的 javaBean 中。
BeanListHandler	将结果集中每一条记录封装到指定的 javaBean 中，将这些 javaBean 在封装到 List 集合中
ColumnListHandler	将结果集中指定的列的字段值，封装到一个 List 集合中
ScalarHandler	它是用于单数据。例如 select count(*) from 表操作。
MapHandler	将结果集第一行封装到 Map 集合中,Key 列名, Value 该列数据
MapListHandler	将结果集第一行封装到 Map 集合中,Key 列名, Value 该列数据,Map 集合存储到 List 集合

1.4.2 JavaBean

JavaBean 就是一个类，在开发中常用封装数据。具有如下特性

1. 需要实现接口：java.io.Serializable ，通常实现接口这步骤省略了，不会影响程序。
2. 提供私有字段：private 类型 字段名;
3. 提供 getter/setter 方法：
4. 提供无参构造

```
    /*
     * 账务类
     */
    public class ZhangWu {
        private int id;
        private String name;
        private double money;
        private String parent;

        public ZhangWu() {
            super();
        }
        public int getId() {
            return id;
        }
        public void setId(int id) {
            this.id = id;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public double getMoney() {
            return money;
        }
        public void setMoney(double money) {
            this.money = money;
        }
        public String getParent() {
            return parent;
        }
        public void setParent(String parent) {
            this.parent = parent;
        }

        @Override
        public String toString() { //该方法可以省略
            return "ZhangWu [id=" + id + ", name=" + name + ", money=" + money + ", parent="
+ parent + "]";
        }
    }
```

1.4.3 ArrayHandler 与 ArrayListHandler 查询

- ArrayHandler：将结果集中的第一条记录封装到一个 Object[]数组中，数组中的每一个元素就是这条记录中的每一个字段的值

```
public class ArrayHandlerDemo {


    @Test
    public void method(){
        try {
            //获取 QueryRunner 对象
            QueryRunner qr = new QueryRunner();
            //执行 SQL 语句
            String sql = "SELECT * FROM zhangwu";
            Object[] params = {};
            Connection conn = JDBCUtils.getConnection();
            Object[] objArray = qr.query(conn, sql, new ArrayHandler(), params);
            //结果集的处理
            System.out.println( Arrays.toString(objArray) );

            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- ArrayListHandler：将结果集中的每一条记录都封装到一个 Object[]数组中，将这些数组在封装到 List 集合中。

```
public class ArrayListHandlerDemo {
    @Test
    public void method(){
        try {
            //获取 QueryRunner 对象
            QueryRunner qr = new QueryRunner();
            //执行 SQL 语句
            String sql = "SELECT * FROM zhangwu WHERE money>?";
            Object[] params = {2000};
            Connection conn = JDBCUtils.getConnection();
             List<Object[]> list = qr.query(conn, sql, new ArrayListHandler(),
params);
            //结果集的处理
             for (Object[] objArray : list) {
                System.out.println(  Arrays.toString(objArray) );
            }
```

```
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

1.4.4 BeanHandler 与 BeanListHandler 查询

- BeanHandler ：将结果集中第一条记录封装到一个指定的 javaBean 中。

```
    public class BeanHandlerDemo {
        @Test
        public void method(){
            try{
                //获取 QueryRunner
                QueryRunner qr = new QueryRunner();
                //执行 SQL 语句
                String sql = "SELECT * FROM zhangwu WHERE id=?";
                Object[] params = {1};
                Connection conn = JDBCUtils.getConnection();
                ZhangWu zw = qr.query(conn, sql, new
BeanHandler<ZhangWu>(ZhangWu.class), params);
                //结果集处理
                System.out.println(zw);

                conn.close();
            } catch(SQLException e){
                throw new RuntimeException(e);
            }
        }
    }
```

- BeanListHandler ：将结果集中每一条记录封装到指定的 javaBean 中，将这些 javaBean 在封装到 List 集合中

```
    public class BeanListHandlerDemo {
        @Test
        public void method(){
            try{
                //获取 QueryRunner
                QueryRunner qr = new QueryRunner();
                //执行 SQL 语句
                String sql = "SELECT * FROM zhangwu WHERE money>?";
                Object[] params = {2000};
```

```
                Connection conn = JDBCUtils.getConnection();
                List<ZhangWu>      list      =      qr.query(conn,      sql,      new
BeanListHandler<ZhangWu>(ZhangWu.class), params);
                //结果集处理
                for (ZhangWu zw : list) {
                    System.out.println(zw);
                }

                conn.close();
            } catch(SQLException e){
                throw new RuntimeException(e);
            }
        }
    }
```

1.4.5 ColumnListHandler 与 ScalarHandler 查询

- ColumnListHandler：将结果集中指定的列的字段值，封装到一个 List 集合中

```
    public class ColumnListHandlerDemo {
        @Test
        public void method(){
            try {
                //获取 QueryRunner 对象
                QueryRunner qr = new QueryRunner();
                //执行 SQL 语句
                String sql = "SELECT name FROM zhangwu WHERE money>?";
                Object[] params = {2000};
                Connection conn = JDBCUtils.getConnection();
               List<String> list = qr.query(conn, sql, new ColumnListHandler<String>(),
params);
                //结果集的处理
                for (String str : list) {
                    System.out.println(str);
                }

                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

- ScalarHandler：它是用于单数据。例如 select count(*) from 表操作。

```
    public class ScalarHandlerDemo {
```

```
    @Test
    public void method(){
        try {
            //获取 QueryRunner 对象
            QueryRunner qr = new QueryRunner();

            //执行 SQL 语句
            String sql = "SELECT MAX(money) FROM zhangwu";
            Object[] params = {};
            Connection conn = JDBCUtils.getConnection();
            Double max = qr.query(conn, sql, new ScalarHandler<Double>(), params);
            //结果集的处理
            System.out.println("max=" + max);

            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

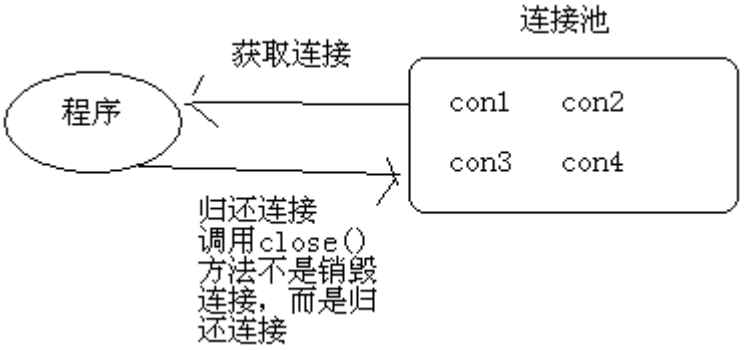
第2章 连接池

实际开发中“获得连接”或“释放资源”是非常消耗系统资源的两个过程，为了解决此类性能问题，通常情况我们采用连接池技术，来共享连接 Connection。这样我们就不需要每次都创建连接、释放连接了，这些操作都交给了连接池

2.1 连接池概述

- 概念

用池来管理 Connection，这样可以重复使用 Connection。有了池，所以我们就不用自己来创建 Connection，而是通过池来获取 Connection 对象。当使用完 Connection 后，调用 Connection 的 close() 方法也不会真的关闭 Connection，而是把 Connection “归还”给池。池就可以再利用这个 Connection 对象了。



- 规范

Java 为数据库连接池提供了公共的接口：**javax.sql.DataSource**，各个厂商需要让自己的连接池实现这个接口。这样应用程序可以方便的切换不同厂商的连接池！

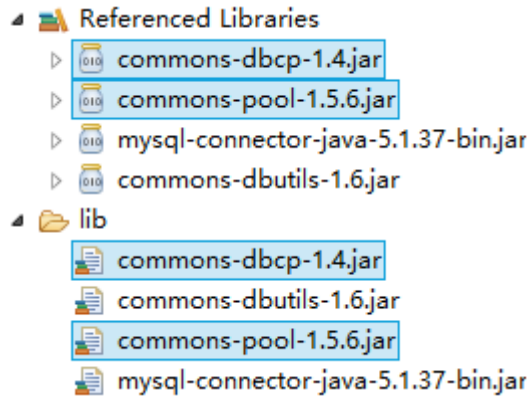
常见的连接池：DBCP、C3P0。

接下来，我们就详细的学习一下 DBCP 连接池。C3P0 连接池我们在就业班学习。

2.2DBCP 连接池

DBCP 也是一个开源的连接池，是 Apache Common 成员之一，在企业开发中也比较常见，tomcat 内置的连接池。

2.2.1 导入 jar 包



2.2.2 编写工具类

连接数据库表的工具类，采用 DBCP 连接池的方式来完成，Java 中提供了一个连接池的规则接口 ：

DataSource：它是 java 中提供的连接池，作为 DriverManager 工具的替代项。在 DBCP 包中提供了 DataSource 接口的实现类，我们要用的具体的连接池 BasicDataSource 类

```
public class JDBCUtils {
```

```
    public static final String DRIVER = "com.mysql.jdbc.Driver";
    public static final String URL = "jdbc:mysql://localhost:3306/daydb";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "root";
    /*
     * 创建连接池 BasicDataSource
     */
    public static BasicDataSource dataSource = new BasicDataSource();
    //静态代码块
    static {
        //对连接池对象 进行基本的配置
        dataSource.setDriverClassName(DRIVER); // 这是要连接的数据库的驱动
        dataSource.setUrl(URL); //指定要连接的数据库地址
        dataSource.setUsername(USERNAME); //指定要连接数据的用户名
        dataSource.setPassword(PASSWORD); //指定要连接数据的密码
    }
    /*
     * 返回连接池对象
     */
    public static DataSource getDataSource(){
        return dataSource;
    }
}
```

2.2.3 工具类的使用

- 测试类

```
/*
 * 演示使用 DBUtils 工具  完成数据库表的增加操作
 */
public class Demo {
    // 插入功能
    @Test
    public void insert(){
        try {
            //获取一个用来执行 SQL 语句的对象  QueryRunner
            QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());
            String sql = "INSERT INTO zhangwu(name,money,parent) VALUES(?,?,?)";
            Object[] params = {"股票收入", 5500, "收入"};
            int line = qr.update(sql,params);
            //结果集处理
            System.out.println("line = " + line);

        } catch (SQLException e) {
```

```
            throw new RuntimeException(e);
        }
    }

    //删除功能
    @Test
    public void delete(){
        try {
            //创建一个 QueryRunner 对象，用来完成 SQL 语句的执行
            QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());
            //执行 SQL 语句
            String sql = "DELETE FROM zhangwu WHERE name = ?";
            Object[] params = {"股票收入"};
            int line = qr.update(sql, params);
            //结果集的处理
            System.out.println("line="+line);

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    //更新功能
    @Test
    public void update(){
        try {
            //创建一个 QueryRunner 对象，用来完成 SQL 语句的执行
            QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());
            //执行 SQL 语句
            String sql = "UPDATE zhangwu SET money = money+1000 WHERE name=?";
            Object[] params = {"股票收入"};
            int line = qr.update(sql, params);
            //结果集的处理
            System.out.println("line="+line);

        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    //查询功能,将结果集中第一条记录封装到一个指定的 javaBean 中。
    @Test
    public void search(){
        try{
```

```
                //获取 QueryRunner
                QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());
                //执行 SQL 语句
                String sql = "SELECT * FROM zhangwu";
                Object[] params = {};
                Product p = qr.query(sql, new BeanHandler<Product>(Product.class),
params);
                //结果集处理
                System.out.println(p);

            } catch(SQLException e){
                throw new RuntimeException(e);
            }
        }
    }
```

2.2.4 常见配置项

分类	属性	描述
必须项	driverClassName	数据库驱动名称
	url	数据库的地址
	username	用户名
	password	密码
基本项（扩展）	maxActive	最大连接数量
	minIdle	最小空闲连接
	maxIdle	最大空闲连接
	initialSize	初始化连接

参考文档：http://commons.apache.org/proper/commons-dbcp/configuration.html

第3章 总结

- DBUtils 工具
- 作用：简化 JDBC 的操作
- 常用类与方法
  - QueryRunner 用来执行 SQL 语句对象
    - update(Connection conn, String sql, Object… params) 插入表记录、更新表记录、删除表记录
    - query(Connection conn, String sql, ResultSetHandler handler, Object… params) 查询表记录
  - ResultSetHandler 处理结果集的对象

ArrayHandler	将结果集中的第一条记录封装到一个 Object[]数组中，数组中的每一个元素就是这条记录中的每一个字段的值
ArrayListHandler	将结果集中的每一条记录都封装到一个 Object[]数组中，将这些数组在封装到 List 集合中。
BeanHandler	将结果集中第一条记录封装到一个指定的 javaBean 中。
BeanListHandler	将结果集中每一条记录封装到指定的 javaBean 中，将这些 javaBean 在封装到 List 集合中
ColumnListHandler	将结果集中指定的列的字段值，封装到一个 List 集合中
ScalarHandler	它是用于单数据。例如 select count(*) from 表操作。
MapHandler	将结果集第一行封装到 Map 集合中,Key 列名, Value 该列数据
MapListHandler	将结果集第一行封装到 Map 集合中,Key 列名, Value 该列数据,Map 集合存储到 List 集合

- DBCP 连接池
  - 作用：自身维护了多个 Connection 连接对象维护
  - BasicDataSource 类 是 DataSource 接口的实现类
- DataSource 接口，它是 java 与每种数据库连接池 连接的规范标准
- DBCP 连接池常见的配置

必须项	driverClassName	数据库驱动名称
	url	数据库的地址
	username	用户名
	password	密码
基本项	maxActive	最大连接数量
	initialSize	连接池中初始化多少个 Connection 连接对象
扩展项	maxWait	超时等待时间以毫秒为单位 1000 等于 1 秒