数据库连接池和DBUtils使用

课前默写

1、写出文件上传的步骤
2、写出上传页面的表单需要设置的属性
3、写出文件下载的步骤

课程回顾

1、文件上传
2、文件下载

今日内容

1、自定义数据库连接池
2、使用DBCP连接池
3、使用C3P0连接池
4、使用Druid连接池
5、DbUtils的使用

教学目标

1、熟悉自定义数据库连接池
2、掌握DBCP连接池
3、掌握C3P0连接池的使用
4、掌握Druid连接池的使用
5、掌握DbUtils的使用

第一章 自定义连接池

使用JDBC操作数据库,需要建立Connection,使用传统的JDBC操作需要每次创建Connection,创建Connection是一个非常性能和消耗时间的过程,我们需要在提高程序性能,那么就需要减少每次创建创建连接带来的负面影响,解决这个问题我们将利用池子概念,预先创建一些链接放入池子中,如果需要操作数据,不用创建新的Connection,只需要在池子中获取即可,使用完毕放入池子!这样就形成了复用!

1.1 自定义连接池

我们可以通过自定义的方式实现连接池!分析连接池类应该包含特定的属性和方法!

属性: 集合 放置Connection

方法: 获取连接方法

回收连接方法

具体实现代码:

```
public class Pool{
```

```
    static LinkedList<Connection> list  = new LinkedList<Connection>();
    static{
        for (int i = 0; i < 10; i++) {
            Connection connection = JDBCUtils.newInstance().getConnection();
            list.add(connection);
        }
    }
    /**
     * 从连接池子中获取连接的方式
     * @return
     */
    public static Connection getConnection(){

        if (list.isEmpty()) {

            Connection connection = JDBCUtils.newInstance().getConnection();

            list.addLast(connection);
        }

        Connection conn = list.removeFirst();

        return conn;
    }
    /**
     * 返回到连接池子中
     */
    public static void addBack(Connection conn){

        if (list.size() >= 10) {
            try {
                conn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }else{
            list.addLast(conn);  //10
        }
    }


    /**
     * 获取连接池子中连接数量的方法
     */

    public static int getSize(){
        return list.size();
    }

}
```

1.2 通过实现java规范实现连接池

Java为连接池实现提供了一个规范(接口),规范的写法,我们需要实现DataSource接口!

但是实现DataSource接口有一个弊端,没有提供回收链接方法!这里我们将使用装饰者模式!

装饰Connection!具体实现代码如下:

1. 创建装饰Connection

```
public class MyConnection   implements Connection{

    //将被装饰者导入

    private Connection conn;
    private LinkedList<Connection> list;


    public MyConnection(Connection conn, LinkedList<Connection> list) {
        super();
        this.conn = conn;
        this.list = list;
    }

    @Override
    public <T> T unwrap(Class<T> iface) throws SQLException {
        // TODO Auto-generated method stub
        return conn.unwrap(iface);
    }

    @Override
    public boolean isWrapperFor(Class<?> iface) throws SQLException {
        // TODO Auto-generated method stub
        return conn.isWrapperFor(iface);
    }

    @Override
    public Statement createStatement() throws SQLException {
        // TODO Auto-generated method stub
        return conn.createStatement();
    }

    @Override
    public PreparedStatement prepareStatement(String sql) throws SQLException {
        // TODO Auto-generated method stub
        return conn.prepareStatement(sql);
    }

    @Override
    public CallableStatement prepareCall(String sql) throws SQLException {
        // TODO Auto-generated method stub
        return null;
    }
```

```
    @Override
    public String nativeSQL(String sql) throws SQLException {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void setAutoCommit(boolean autoCommit) throws SQLException {
        // TODO Auto-generated method stub

    }

    @Override
    public boolean getAutoCommit() throws SQLException {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void commit() throws SQLException {
        // TODO Auto-generated method stub
        conn.commit();
    }

    @Override
    public void rollback() throws SQLException {
        // TODO Auto-generated method stub
        conn.rollback();
    }

    @Override
    public void close() throws SQLException {
        // TODO Auto-generated method stub

        list.addLast(conn);

    }
    ...
}
```

基于规范实现的连接池

```
/**
 * 创建一个规范的连接池子
 */
public class DataSourcePool implements DataSource{
    static LinkedList<Connection> list  = new LinkedList<Connection>();
    static{

        for (int i = 0; i < 10; i++) {
            Connection connection = JDBCUtils.newInstance().getConnection();
            list.add(connection);
        }
```

```
    }


    public static int getSize(){
        return list.size();
    }

    @Override
    public Connection getConnection() throws SQLException {
        // TODO Auto-generated method stub

        Connection conn = list.removeFirst();

        MyConnection conn1 = new  MyConnection(conn, list);

        return conn1;
    }


    @Override
    public PrintWriter getLogWriter() throws SQLException {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void setLogWriter(PrintWriter out) throws SQLException {
        // TODO Auto-generated method stub

    }

    @Override
    public void setLoginTimeout(int seconds) throws SQLException {
        // TODO Auto-generated method stub

    }

    @Override
    public int getLoginTimeout() throws SQLException {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Logger getParentLogger() throws SQLFeatureNotSupportedException {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public <T> T unwrap(Class<T> iface) throws SQLException {
        // TODO Auto-generated method stub

        return null;
```

```
    }

    @Override
    public boolean isWrapperFor(Class<?> iface) throws SQLException {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Connection getConnection(String username, String password) throws SQLException {
        // TODO Auto-generated method stub
        return null;
    }
}
```

第二章 使用DBCP连接池

> DBCP(DataBase connection pool),数据库连接池。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。

2.1 DBCP连接池的使用

2.1.1 创建项目

创建JavaWeb项目

2.1.2 导入相应jar包

mysql-jdbc.jar

commons-dbcp.jar

commons-pool.jar

2.1.3 硬编码方式使用DBCP连接池

> 所谓的硬编码方式就是在代码中添加配置

```
@Test
public void testHard() throws SQLException{
        //TODO 硬编码 使用DBCP连接池子
        BasicDataSource source = new BasicDataSource();
        //设置连接的信息
        source.setDriverClassName("com.mysql.jdbc.Driver");
        source.setUrl("jdbc:mysql://localhost:3306/day2");
        source.setUsername("root");
        source.setPassword("111");

        Connection connection = source.getConnection();

        String sql = "select * from student";

        Statement createStatement = connection.createStatement();
```

```
        ResultSet executeQuery = createStatement.executeQuery(sql);

        while (executeQuery.next()) {

            System.out.println(executeQuery.getString(2));
        }
        connection.close(); //回收
    }
```

2.1.4 软编码方式使用DBCP连接池

> 所谓的软编码,就是在项目中添加配置文件,这样就不需要每次代码中添加配合!

1. 项目中添加配置

文件名称: info.properties

文件位置: src下

```
#连接设置
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/day2
username=root
password=111
#<!-- 初始化连接 -->
initialSize=10
#最大连接数量
maxActive=50
#<!-- 最大空闲连接 -->
maxIdle=20
#<!-- 最小空闲连接 -->
minIdle=5
#<!-- 超时等待时间以毫秒为单位 6000毫秒/1000等于60秒 -->
maxWait=6000
```

1. 代码实现

```
    @Test
    public void testSoft() throws Exception{
        //TODO DBCP软编码连接池子使用
        BasicDataSourceFactory factory = new BasicDataSourceFactory();
        Properties properties = new  Properties();
        //配置文件添加到properties对象中  javase
        properties.load(new FileInputStream("src/info.properties"));
        //生成连接池子  需要配置文件
        DataSource source = factory.createDataSource(properties);
       Connection connection = source.getConnection();
        String sql = "select * from student";
        Statement createStatement = connection.createStatement();
        ResultSet executeQuery = createStatement.executeQuery(sql);
        while (executeQuery.next()) {

            System.out.println(executeQuery.getString(2));
```

```
        }
        connection.close(); //回收
    }
```

第三章 使用C3P0连接池

> C3P0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。目前使用它的开源项目有Hibernate，Spring等。

```
c3p0与dbcp区别
    1.
        dbcp没有自动回收空闲连接的功能
        c3p0有自动回收空闲连接功能
    2.
        dbcp需要手动设置配置文件
        c3p0不需要手动设置
```

3.1 使用步骤

3.1.1 创建项目

3.1.2 导入jar包

c3p0-0.9.1.2.jar

mysql-connector-java-5.0.8.jar

3.1.3.添加配置文件

> c3p0是在外部添加配置文件,工具直接进行应用,因为直接引用,所以要求固定的命名和文件位置

文件位置: src

文件命名:c3p0-config.xml/c3p0-config.properties

```
<c3p0-config>
    <!-- 默认配置，如果没有指定则使用这个配置 -->
    <default-config>
        <!-- 基本配置 -->
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/day2</property>
        <property name="user">root</property>
        <property name="password">111</property>
        <!--扩展配置-->
        <!-- 连接超过30秒报错-->
        <property name="checkoutTimeout">30000</property>
        <!--30秒检查空闲连接 -->
        <property name="idleConnectionTestPeriod">30</property>
        <property name="initialPoolSize">10</property>
         <!-- 30秒不适用丢弃-->
        <property name="maxIdleTime">30</property>
        <property name="maxPoolSize">100</property>
        <property name="minPoolSize">10</property>

        <property name="maxStatements">200</property>
```

```
    </default-config>
    <!-- 命名的配置 -->
    <named-config name="zhaowf">
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/day2</property>
        <property name="user">root</property>
        <property name="password">111</property>
        <!-- 如果池中数据连接不够时一次增长多少个 -->
        <property name="acquireIncrement">5</property>
        <property name="initialPoolSize">20</property>
        <property name="minPoolSize">10</property>
        <property name="maxPoolSize">40</property>
        <property name="maxStatements">20</property>
        <property name="maxStatementsPerConnection">5</property>
    </named-config>
</c3p0-config>
```

注意: c3p0的配置文件内部可以包含命名配置文件和默认配置文件!默认是选择默认配置!如果需要切换命名配置可以在创建c3p0连接池的时候填入命名即可!

3.2 使用c3p0进行数据库操作

```
public class TestC3p0 {

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

        //1.创建C3P0连接池子
        Connection connection = DataSourceUtils.getConnection();

        Statement createStatement = connection.createStatement();

        String sql = "select * from student;";

        ResultSet resultSet = createStatement.executeQuery(sql);

        while (resultSet.next()) {

            System.out.println(resultSet.getString(1));
        }
        DataSourceUtils.close(connection, createStatement, resultSet);
    }
}
```

3.4 使用c3p0连接池编写工具类

基于c3p0连接池编写数据库操作工具类!

```
/**
 * 从连接池子中获取连接!
 *
 *  C3P0的连接池子
 *  0.获取连接池子对象  DBUtils
 *  1.获取连接
```

```
 *  2.关闭资源
 */
public class DataSourceUtils {

    private static  ComboPooledDataSource  dataSource = new ComboPooledDataSource();

    /**
     * 返回连接池对象方法
     * @return c3p0连接池子
     */
    public static  ComboPooledDataSource getDataSource(){
        return dataSource;
    }

    /**
     * 连接池中获取连接的方法
     * @return 连接
     */
    public static Connection getConnection(){

        Connection conn  = null;
        try {
            conn = dataSource.getConnection();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return conn;
    }

    //关闭资源
    public static void close(Connection conn){
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    public static  void close(Statement st){

        if (st != null) {
            try {
                st.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }
```

```
    public  static void close(ResultSet  set){

        if (set != null) {
            try {
                set.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    public static void close(Connection conn,Statement st){
        close(conn);
        close(st);
    }

    public  static void close(Connection conn,Statement st,ResultSet rt){
        close(conn);
        close(st);
        close(rt);
    }
}
```

第四章 使用Druid数据库连接池

Druid 是目前比较流行的高性能的，分布式列存储的OLAP框架(具体来说是MOLAP)。它有如下几个特点:
一. 亚秒级查询
    druid提供了快速的聚合能力以及亚秒级的OLAP查询能力，多租户的设计，是面向用户分析应用的理想方式。
二.实时数据注入
    druid支持流数据的注入，并提供了数据的事件驱动，保证在实时和离线环境下事件的实效性和统一性
三.可扩展的PB级存储
    druid集群可以很方便的扩容到PB的数据量，每秒百万级别的数据注入。即便在加大数据规模的情况下，也能保证时其效性
四.多环境部署
    druid既可以运行在商业的硬件上，也可以运行在云上。它可以从多种数据系统中注入数据，包括hadoop, spark, kafka, storm和samza等
五.丰富的社区
    druid拥有丰富的社区，供大家学习

4.1 使用步骤

4.1.1 导入jar包

4.1.2 编写工具类

```
/**
 * 阿里的数据库连接池
 * 性能最好的
 * Druid
 * */
public class DruidUtils {
```

```
    //声明连接池对象
    private static DruidDataSource ds;
    static{
        ///实例化数据库连接池对象
        ds=new DruidDataSource();
        //实例化配置对象
        Properties properties=new Properties();
        try {
            //加载配置文件内容
        properties.load(DruidUtils.class.getResourceAsStream("dbcpconfig.properties"));
            //设置驱动类全称
            ds.setDriverClassName(properties.getProperty("driverClassName"));
            //设置连接的数据库
            ds.setUrl(properties.getProperty("url"));
            //设置用户名
            ds.setUsername(properties.getProperty("username"));
            //设置密码
            ds.setPassword(properties.getProperty("password"));
            //设置最大连接数量
            ds.setMaxActive(Integer.parseInt(properties.getProperty("maxActive")));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    //获取连接对象
    public static Connection getConnection() {
        try {
            return ds.getConnection();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }
}
```

4.1.3 测试

```
public class Druid_Test{
    @Test
    public void test1(){
        findAll();
        find("小千千","47983");
    }
    public List<Master> findAll() {
        // TODO Auto-generated method stub
        Connection conn=null;
        PreparedStatement pstat=null;
        ResultSet rs=null;
        List<Master> masters=new ArrayList<Master>();
        try{

            conn=DruidUtils.getConnection();
```

```
            pstat=conn.prepareStatement("select * from master");
            rs=pstat.executeQuery();
            while(rs.next()){
                int id=rs.getInt("id");
                String name=rs.getString("name");
                String password=rs.getString("password");
                int money=rs.getInt("money");
                masters.add(new Master(id, name, password, money));
            }
            return masters;
        }catch(Exception e){
            throw new RuntimeException(e);
        }finally {
            DbUtils.release(rs, pstat, conn);
        }
    }
    public Master find(String name, String password) {
        // TODO Auto-generated method stub
        Connection conn=null;
        PreparedStatement pstat=null;
        ResultSet rs=null;
        Master master=null;
        try{
            conn=DbUtils.getConnection();
            pstat=conn.prepareStatement("select * from master where name=? and password=?");
            pstat.setString(1, name);
            pstat.setString(2, password);
            rs=pstat.executeQuery();
            if(rs.next()){
                int id=rs.getInt("id");
                int money=rs.getInt("money");
                master=new Master(id, name, password, money);
            }
            return master;
        }catch(Exception e){
            throw new RuntimeException(e);
        }finally {
            DbUtils.release(rs, pstat, conn);
        }
    }
```

第五章 DBUtils工具类使用

> Commons DbUtils是Apache组织提供的一个对JDBC进行简单封装的开源工具类库，使用它能够简化JDBC应用程序的开发，同时也不会影响程序的性能。

5.1 DBUtils简介

DBUtils是java编程中的数据库操作实用工具，小巧简单实用，

1.对于数据表的读操作，他可以把结果转换成List，Array，Set等java集合，便于程序员操作；

2.对于数据表的写操作，也变得很简单（只需写sql语句）

3.可以使用数据源，使用INDI，数据库连接池等技术来优化性能--重用已经构建好的数据库连接对象，而不像php，asp那样，费时费力的不断重复的构建和析构这样的对象。

DBUtils包括3个包：

org.apache.commons.dbutils

org.apache.commons.dbutils.handlers

org.apache.commons.dbutils.wrappers

DBUtils封装了对JDBC的操作，简化了JDBC操作，可以少写代码。

org.apache.commons.dbutils

DbUtils 关闭链接等操作

QueryRunner 进行查询的操作

org.apache.commons.dbutils.handlers

ArrayHandler ： 将ResultSet中第一行的数据转化成对象数组

ArrayListHandler将ResultSet中所有的数据转化成List，List中存放的是Object[]

BeanHandler ： 将ResultSet中第一行的数据转化成类对象

BeanListHandler ： 将ResultSet中所有的数据转化成List，List中存放的是类对象

ColumnListHandler ： 将ResultSet中某一列的数据存成List，List中存放的是Object对象

KeyedHandler ： 将ResultSet中存成映射，key为某一列对应为Map。Map中存放的是数据

MapHandler ： 将ResultSet中第一行的数据存成Map映射

MapListHandler ： 将ResultSet中所有的数据存成List。List中存放的是Map

ScalarHandler ： 将ResultSet中一条记录的其中某一列的数据存成Object

org.apache.commons.dbutils.wrappers

SqlNullCheckedResultSet ： 对ResultSet进行操作，改版里面的值

StringTrimmedResultSet ： 去除ResultSet中中字段的左右空格。Trim()

主要方法：

DbUtils类：启动类

ResultSetHandler接口：转换类型接口

MapListHandler类：实现类，把记录转化成List

BeanListHandler类：实现类，把记录转化成List，使记录为JavaBean类型的对象

Query Runner类：执行SQL语句的类

5.2 DBUtils+C3P0工具类实现

5.2.1 项目准备

- 创建项目

- 导入jar包 工具类 配置文件

dbutils.jar

c3p0.jar

jdbc.jar

之前项目的info.properties

c3p0工具类

5.2.2 实现代码

```
public class ResultHanlder {

    @Test
    public void testArrayHander() throws SQLException {

        // ArrayHandler:适合取1条记录。把该条记录的每列值封装到一个数组中Object[]
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        Object[] query = runner.query("select * from day13 where id = ?", new ArrayHandler(),
9);

        for (Object object : query) {

            System.out.println(object);
        }

    }

    @Test
    public void testArrayListHander() throws SQLException {

        // ArrayHandler:适合取1条记录。把该条记录的每列值封装到一个数组中Object[]
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        List<Object[]> query = runner.query("select * from day13 ", new ArrayListHandler());

        for (Object[] objects : query) {
            for (Object object : objects) {

                System.out.println(object);
            }
        }

    }

    @Test
    public void testColumnListHander() throws SQLException {

        // ColumnListHandler:取某一列的数据。封装到List中。
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());
```

```
        List<Object> query = runner.query("select * from day13 ", new ColumnListHandler(2));

        for (Object objects : query) {

            System.out.println(objects);
        }

    }

    @Test
    public void testKeyedHandler() throws SQLException {

        // KeyedHandler:取多条记录，每一条记录封装到一个Map中，再把这个Map封装到另外一个Map中，key为指定的字段值
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        Map<Object, Map<String, Object>> query = runner.query("select * from day13 ", new KeyedHandler(2));

        for (Object key : query.keySet()) {
            System.out.println(key);
            for (Entry<String, Object> entry : query.get(key).entrySet()) {
                System.out.println("key:" + entry.getKey() + "  values:" + entry.getValue());
            }
        }

    }

    @Test
    public void testMapHandler() throws SQLException {

        // MapHandler:适合取1条记录。把当前记录的列名和列值放到一个Map中
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        Map<String, Object> query = runner.query("select * from day13 ", new MapHandler());

        for (Entry<String, Object> entry : query.entrySet()) {
            System.out.println("key:" + entry.getKey() + "  values:" + entry.getValue());
        }
    }

    @Test
    public void testMapListHandler() throws SQLException {

        // MapListHandler:适合取多条记录。把每条记录封装到一个Map中，再把Map封装到List中
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        List<Map<String, Object>> query = runner.query("select * from day13 ", new MapListHandler());

        for (Map<String, Object> entry : query) {
```

```
            for (Entry<String, Object> mapEntr : entry.entrySet()) {
                System.out.println("key:" + mapEntr.getKey() + "  values:" +
mapEntr.getValue());
            }
        }
    }

    @Test
    public void testScalarHandler() throws SQLException {

        // ScalarHandler:适合取单行单列数据
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());

        Object query = runner.query("select count(*) from day13 ", new ScalarHandler());
        System.out.println(query);
    }

    @Test
    public void testBeanHandler() throws SQLException {
        // BeanHandler:适合取单行单列数据
        QueryRunner runner = new QueryRunner(C3P0Util.getDataSource());
        Day13 query = runner.query("select name,age from day13 ", new BeanHandler<Day13>
(Day13.class));
        System.out.println(query.toString());
    }
}
```

作业题

1、使用数据库连接池和DbUtils改造之前的项目

面试题

1、描述数据库连接池的优缺点