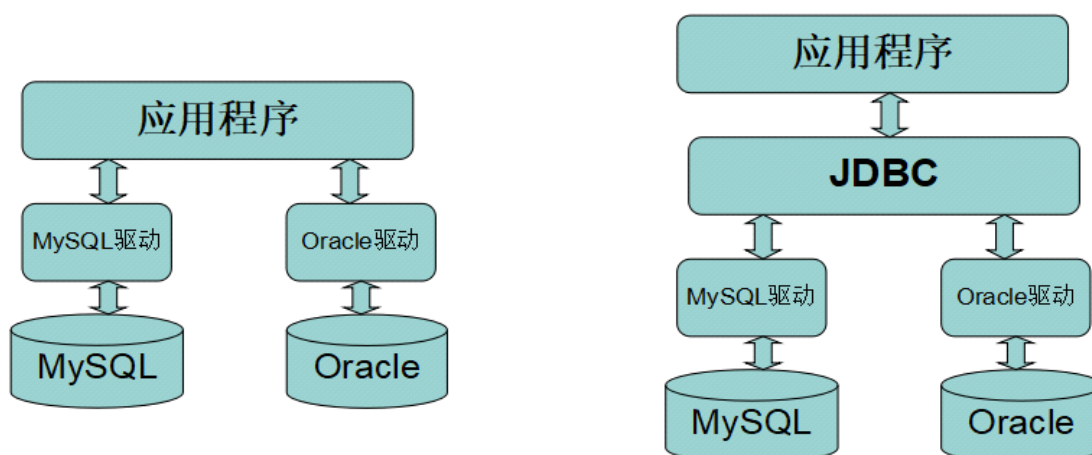


1. 知识回顾



2. JDBC概述



a. 驱动

- 为了能让程序员利用java程序操作数据库，数据库厂商提供了一套jar包，通过导入这个jar包就可以直接调用其中的方法操作数据库，这个jar包称之为驱动。
 - 两个数据库驱动互不兼容。
- 行业中有多种的数据库，要使用这么多数据库需要学习很多数据库驱动，对于程序员来说学习成本非常高。
- 想要让java程序兼容数据库，所有的数据库驱动都实现了jdbc这套接口。

b. JDBC简介：

- JDBC全称为：Java Data Base Connectivity (java数据库连接)，它主要由接口组成。
 - 组成JDBC的2个包：
 - java.sql包
 - javax.sql包
 - 开发JDBC应用需要以上2个包的支持外，还需要导入相应JDBC的数据库实现(即数据库驱动)。



- 不仅需要jdbc接口，还需要驱动这个实现，驱动中就是对jdbc接口的一些实现。

3. 6步实现jdbc

- 注册数据库驱动
- 获取数据库连接
- 创建传输器
- 传输sql并返回结果
- 遍历结果
- 关闭资源

4. 代码实现

```
package cn.tedu.jdbc;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * 6步实现JDBC
 * @author 16
 */
public class Demo1 {
    public static void main(String[] args) throws SQLException, ClassNotFoundException {
        //1.注册数据库驱动
        //在注册数据库驱动过程中，利用构造方法注册驱动，在Driver类中的static块中也会注册一次，我们手动又注册一次
        //代码与我们导入的Driver类所在的包被绑定在了一起，如果修改数据库，需要回到代码当中，对导入的包进行修改。
        DriverManager.registerDriver(new Driver());
        Class.forName("com.mysql.jdbc.Driver");
        //2.获取数据库连接
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2", "root", "root");
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=root");
        //jdbc:mysql://mydb2
        //jdbc:mysql://localhost:3306/mydb2?user=root&password=root

        //3.创建一个传输器
        Statement stat = conn.createStatement();
        //4.利用传输器传输数据，并返回结果集。
        ResultSet rs = stat.executeQuery("select * from user");
        //5.遍历结果
        while(rs.next()){
            int id = rs.getInt("id");
            String name = rs.getString(2);
            Date date = rs.getDate("birthday");
            System.out.println("id:"+id+"name:"+name+"date:"+date);
        }
        // rs.beforeFirst();//表示回到第一行之前的一行
        // rs.previous();//表示向前一行
        // rs.absolute(2);//表示移动到第二行
        //6.关闭资源
        //后创建的先关闭
        rs.close();
        stat.close();
        conn.close();
    }
}
```

```
}
```

5.

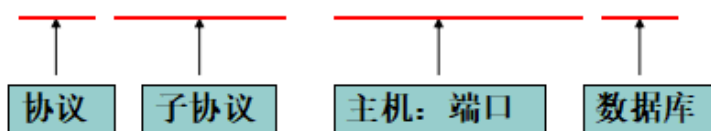
1. 程序详解—DriverManager

- Jdbc程序中的DriverManager用于加载驱动，并创建与数据库的链接，这个API的常用方法：
DriverManager.registerDriver(new Driver())
DriverManager.getConnection(url, user, password),
- 注意：在实际开发中并不推荐采用registerDriver方法注册驱动。原因有二：
 - 查看Driver的源代码可以看到，如果采用此种方式，会导致驱动程序注册两次，也就是在内存中会有两个Driver对象。
 - 程序依赖mysql的api，脱离mysql的jar包，程序将无法编译，将来程序切换底层数据库将会非常麻烦。
- 推荐方式：Class.forName("com.mysql.jdbc.Driver");
 - 采用此种方式不会导致驱动对象在内存中重复出现，并且采用此种方式，程序仅仅只需要一个字符串，不需要依赖具体的驱动，使程序的灵活性更高。
 - 同样，在开发中也不建议采用具体的驱动类型指向getConnection方法返回的connection对象。

2. 数据库URL

URL用于标识数据库的位置，程序员通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：

jdbc:mysql://localhost:3306/test ?参数名=参数值



3. 常用数据库URL地址的写法：

Oracle写法：jdbc:oracle:thin:@localhost:1521:sid

SqlServer—jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=sid

MySql—jdbc:mysql://localhost:3306/sid

Mysql的url地址的简写形式：jdbc:mysql:///sid

常用属性：useUnicode=true&characterEncoding=UTF-8

4. 程序详解—Connection

- Jdbc程序中的Connection，它用于代表数据库的链接，Connection是数据库编程中最重要对象，客户端与数据库所有交互都是通过connection对象完成的，这个对象的常用方法：

createStatement()：创建向数据库发送sql的statement对象。

prepareStatement(sql)：创建向数据库发送预编译sql的PrepareStatement对象。

prepareCall(sql)：创建执行存储过程的callableStatement对象。

setAutoCommit(boolean autoCommit)：设置事务是否自动提交。

commit()：在链接上提交事务。

rollback()：在此链接上回滚事务。

5. 程序详解—Statement

- Jdbc程序中的Statement对象用于向数据库发送SQL语句，Statement对象常用方法：

executeQuery(String sql)：用于向数据库发送查询语句。

executeUpdate(String sql)：用于向数据库发送insert、update或删除语句

execute(String sql)：用于向数据库发送任意sql语句

addBatch(String sql)：把多条sql语句放到一个批处理中。

executeBatch()：向数据库发送一批sql语句执行。

6. 程序详解—ResultSet

- Jdbc程序中的ResultSet用于代表Sql语句的执行结果。ResultSet封装执行结果时，采用的类似于表格的方式。ResultSet对象维护了一个指向表格数据行的游标，初始的时候，游标在第一行之前，调用ResultSet.next()方法，可以使游标指向具体的数据行，进行调用方法获取该行的数据。
- ResultSet既然用于封装执行结果的，所以该对象提供的都是用于获取数据的get方法：
- 获取任意类型的数据
 - getObject(int index)
 - getObject(string columnName)
- 获取指定类型的数据，例如：
 - getString(int index)
 - getString(String columnName)
- 提问：数据库中列的类型是varchar，获取该列的数据调用什么方法？Int类型呢？bigInt类型呢？Boolean类型？

7. 常用数据类型转换表

SQL类型	Jdbc对应方法	返回类型
BIT(1) bit(n)	getBoolean getBytes()	Boolean byte[]
TINYINT	getByte()	Byte
SMALLINT	getShort()	Short
Int	getInt()	Int
BIGINT	getLong()	Long
CHAR, VARCHAR, LONGVARCHAR	getString()	String
Text(clob) Blob	getClob getBlob()	Clob Blob
DATE	getDate()	java.sql.Date
TIME	getTime()	java.sql.Time
TIMESTAMP	getTimestamp()	java.sql.Timestamp

8. ResultSet中的api

- ResultSet还提供了对结果集进行滚动的方法：

next()：移动到下一行

Previous()：移动到前一行

`absolute(int row)`：移动到指定行
`beforeFirst()`：移动resultSet的最前面。
`afterLast()`：移动到resultSet的最后面。

9. 程序详解—释放资源

○ 为什么要关闭资源?

在安装数据库的时候，设置过最大连接数量，如果用了不还连接，别人就无法使用了。

rs对象中可能包含很大的一个数据，对象保存在内存中，这样就十分占用内存。需要将他关闭。

最晚创建的对象，最先关闭。

- Jdbc程序运行完后，切记要释放程序在运行过程中，创建的那些与数据库进行交互的对象，这些对象通常是ResultSet, Statement和Connection对象。
- 特别是Connection对象，它是非常稀有的资源，用完后必须马上释放，如果Connection不能及时、正确的关闭，极易导致系统宕机。Connection的使用原则是尽量晚创建，尽量早的释放。
- 为确保资源释放代码能运行，资源释放代码也一定要放在finally语句中。

10. 释放资源

- 在关闭过程中可能会出现异常，为了能够关闭资源，需要将资源在finally中关闭。
- 如果在finally中关闭资源则需要将conn,stat,rs三个对象定义成全局的变量。
- 在conn,stat,rs三个变量出现异常的时候可能会关闭不成功，我们需要将他们在finally中置为null。conn,stat,rs这三个对象是引用，将引用置为null，它引用的对象就会被JVM回收，也能保证资源的释放。

登录功能

星期二, 2019年7月9日 15:50

```
package cn.tedu.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

import cn.tedu.utils.JDBCUtils;

//模拟登录功能
public class Login {
    public static void main(String[] args) {
        System.out.println("请输入用户名: ");
        Scanner sc = new Scanner(System.in);
        String username = sc.nextLine();
        System.out.println("请输入密码: ");
        String password = sc.nextLine();
        //      testLogin(username,password);
        testPreLogin(username,password);
    }

    //preparedstatement访问数据库 防止sql注入攻击
    private static void testPreLogin(String username, String password) {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            conn = JDBCUtils.getConnection();
            ps = conn.prepareStatement("select * from user where username=? and password = ?");
            ps.setString(1, username);
            ps.setString(2, password);
            rs = ps.executeQuery();
            if(rs.next()){
                System.out.println("登录成功");
            }else{
                System.out.println("登录失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            JDBCUtils.close(conn, ps, rs);
        }
    }

    private static void testLogin(String username, String password) {
```

```
Connection conn = null;
Statement stat = null;
ResultSet rs = null;
try {
    conn = JDBCUtils.getConnection();
    stat = conn.createStatement();
    rs = stat.executeQuery(
        "select * from user where username = '"+username+"' and password = '"+password+"'");
    if(rs.next()){
        System.out.println("登录成功");
    }else{
        System.out.println("登录失败");
    }
} catch (Exception e) {
    e.printStackTrace();
}finally{
    JDBCUtils.close(conn, stat, rs);
}

}

1.
```


1. SQL注入攻击

由于SQL语句的参数是由前台页面发送到后台，sql语句是在后台拼接而来，在拼接的过程中，用户输入了数据库的关键字符，导致sql语句的语义发生的改变，这种攻击方式称之为sql注入攻击。

2. PreparedStatement

有预编译功能的传输器。

preparedStatement的执行原理：

第一步：将sql语句的主干部分发送到数据库。主干部分的参数使用问号预留。sql语句到达数据库后会变成一段机器码。(一段二进制数据。)

第二部：将sql语句的参数发送到数据库。参数在发送到数据库后会看做是纯文本的内容来使用。

在sql语句的主干和参数发送完成之后，需要通知数据库服务器执行sql。通知方式：

```
preparedstatement.executeQuery();
preparedstatement.executeUpdate();
```

```
package cn.tedu.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import cn.tedu.utils.JDBCUtils;

//预编译功能的传输器
public class PreparedDemo1 {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            conn = JDBCUtils.getConnection();
            //传输sql语句的主干部分
            ps = conn.prepareStatement("select * from exam where id = ?");
            //传输sql语句的参数部分
            ps.setInt(1, 3);
            //通知数据库服务器执行sql语句
            rs = ps.executeQuery();
            while(rs.next()){
                String name = rs.getString("name");
                System.out.println("name:"+name);
            }
        } catch (Exception e) {
```

```
        // TODO Auto-generated catch block
        e.printStackTrace();
    }finally{
        JDBCUtils.close(conn, ps, rs);
    }
}
```

1. 使用业务场景：当需要向数据库发送一批SQL语句执行时，应避免向数据库一条条的发送执行，而应采用JDBC的批处理机制，以提升执行效率。

- 实现批处理有两种方式，第一种方式：

Statement.addBatch(sql)

- 执行批处理SQL语句

executeBatch()方法：执行批处理命令

clearBatch()方法：清除批处理命令

2. statement批处理：

```
Connection conn = null;
Statement st = null;
ResultSet rs = null;
try {
    conn = JdbcUtil.getConnection();
    String sql1 = "insert into person(name,password,email,birthday)
        values('kkk','123','abc@sina.com','1978-08-08')";
    String sql2 = "update user set password='123456' where id=3";
    st = conn.createStatement();
    st.addBatch(sql1); //把SQL语句加入到批命令中
    st.addBatch(sql2); //把SQL语句加入到批命令中
    st.executeBatch();
} finally{
    JdbcUtil.close(conn, st, rs);
}
```

- 采用Statement.addBatch(sql)方式实现批处理：

- 优点：可以执行多条不同结构的sql语句
- 缺点：没有使用预编译机制，效率低下，如果要执行多条结构相同仅仅参数不同的语句时，仍然需要多次写sql语句的主干。

3. PreparedStatement批处理：

```
Connection conn = null;
PreparedStatement ps = null;
try{
    conn = JDBCUtils.getConn();
    ps = conn.prepareStatement("insert into psbatch values(null,?)");

    for(int i=1;i<=100000;i++){
        ps.setString(1, "name"+i);
        ps.addBatch();
        if(i%1000==0){
            ps.executeBatch();
            ps.clearBatch();
        }
    }
    ps.executeBatch();

}catch (Exception e) {
    e.printStackTrace();
}finally{
    JDBCUtils.close(null, ps, conn);
}
```

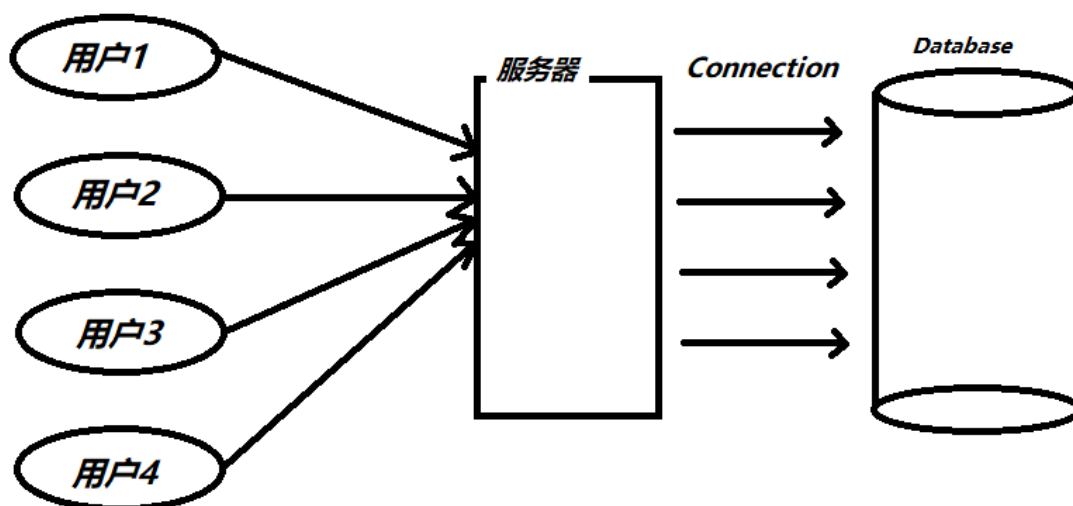
- 采用Statement.addBatch(sql)方式实现批处理：

- 优点:有预编译机制,效率比较高.执行多条结构相同,参数不同的sql时,不需要重复写sql的主干

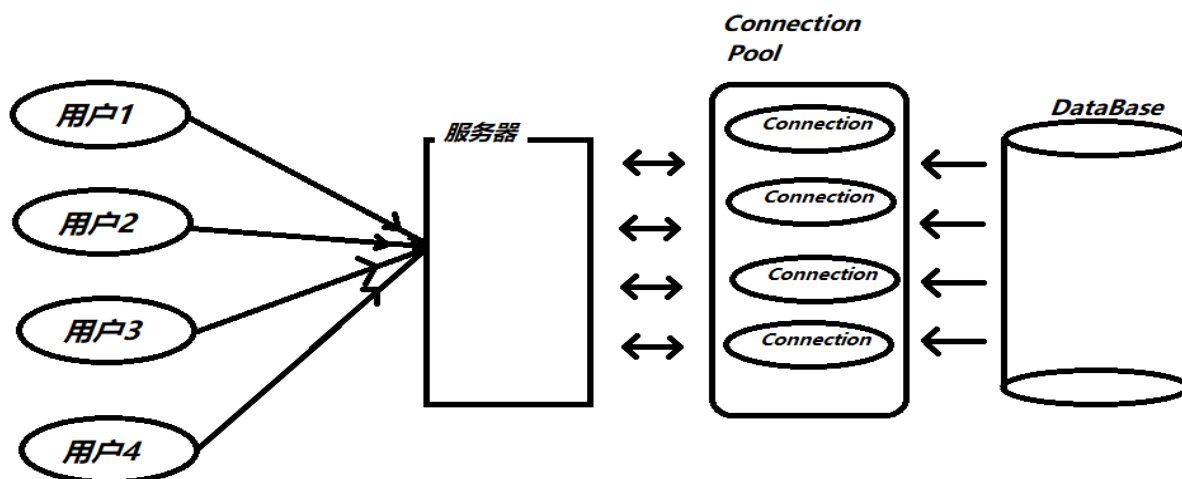
- 缺点:只能执行主干相同参数不同的sql,没有办法在一个批中加入结构不同的sql

1. C3P0连接池

a. 为什么使用连接池？



- 缺点：用户每次请求都需要向数据库获得链接，而数据库创建连接通常需要消耗相对较大的资源，创建时间也较长。假设网站一天10万访问量，数据库服务器就需要创建10万次连接，极大的浪费数据库的资源，并且极易造成数据库服务器内存溢出、宕机。



- 它是将那些已连接的数据库连接存放在一个容器里（连接池），这样以后别人要连接数据库的时候，将不会重新建立数据库连接，会直接从连接池里取出可用的连接，用户使用完毕后，连接又重新还回到连接池中。
- 注意：连接池里的连接将会一直保存在内存里，即使你没用也是一样。所以这个时候你得权衡一下连接池的连接数量了。

2. 实现：

- 编写连接池需实现`javax.sql.DataSource`接口。`DataSource`接口中定义了两个重载的`getConnection`方法：
 - `Connection getConnection()`
 - `Connection getConnection(String username, String password)`
- 实现`DataSource`接口，并实现连接池功能的步骤：
 - 在`DataSource`构造函数中批量创建与数据库的连接，并把创建的连接保存到一个集合对象中。
 - 实现`getConnection`方法，让`getConnection`方法每次调用时，从集合对象中取一个`Connection`返回给用户。
 - 当用户使用完`Connection`，调用`Connection.close()`方法时，`Collection`对象应保证将自己返回到连接池的集合对象中,而不要把`conn`还给数据库。
 - 扩展`Connection`的`close`方法。
 - 在关闭数据库连接时，将`connection`存回连接池中，而并非真正的关闭。

3. 手写MyPool连接池:

```
package cn.tedu.pool;

import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.LinkedList;
import java.util.List;

import javax.sql.DataSource;
/**
 * 手写连接池
 * @author 16
 *
 */
public class MyPool implements DataSource{
    //定义一个能够存储连接的数据结构，由于经常使用插入和删除操作，所以List较好。
    private static List<Connection> pool = new LinkedList<Connection>();
    static{//在程序之后立刻创建一批连接以备使用
        try{
            Class.forName("com.mysql.jdbc.Driver");
            for(int i=0;i<5;i++){
                //每次都创建一个新的连接对象
                Connection conn =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=root");
                //将创建好的每一个连接对象添加到List中，模拟将连接加入连接池
                pool.add(conn);
            }
        }catch(Exception e){
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }

    //创建连接(从连接池中取出一个连接)
    @Override
    public Connection getConnection() throws SQLException {
        if(pool.size()==0){//取出连接之前首先判断当前连接池中是否还有连接，如果没有则重新创建一批连接
            for(int i=0;i<5;i++){
                Connection conn =
                DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb2?user=root&password=root");
                pool.add(conn);
            }
        }
    }
}
```

```

    }
    //从List中取出一个连接对象
    //此处不能使用get(),get()方法只能读取对应下标的元素,没有将读取到的元素移除,如果是取出连接对
    象,应将对象移除。
    Connection conn = pool.remove(0);
    ConnDecorate connpool = new ConnDecorate(conn, this);
    System.out.println("成功获取一个连接,池中还剩: "+pool.size()+"个连接");
    return connpool;
}
//返还连接
//手写一个返还连接的方法
public void retConn(Connection conn){
    try {
        //归还的连接如果已经关闭或者为空,则不允许放入池中。
        if(conn!=null&&!conn.isClosed()){
            pool.add(conn);
            System.out.println("成功还回一个连接,池中还剩: "+pool.size()+"个连接");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public PrintWriter getLogWriter() throws SQLException {
    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 <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;
}
}

```

开源连接池：

现在很多WEB服务器(Weblogic, WebSphere, Tomcat)都提供了DataSource的实现，即连接池的实现。通常我们把DataSource的实现，按其英文含义称之为数据源，数据源中都包含了数据库连接池的实现。

也有一些开源组织提供了数据源的独立实现：

DBCP 数据库连接池

C3P0 数据库连接池

实际应用时不需要编写连接数据库代码，直接从数据源获得数据库的连接。程序员编程时也应尽量使用这些数据源的实现，以提升程序的数据库访问性能。

DBCP

DBCP 是 Apache 软件基金组织下的开源连接池实现，使用DBCP数据源，应用程序应在系统中增加如下两个 jar 文件：

Commons-dbcp.jar：连接池的实现

Commons-pool.jar：连接池实现的依赖库

DBCP示例代码：

```
static{
    InputStream in = JdbcUtil.class.getClassLoader().
        getResourceAsStream("dbcpconfig.properties");
    Properties prop = new Properties();
    prop.load(in);

    BasicDataSourceFactory factory = new BasicDataSourceFactory();
    dataSource = factory.createDataSource(prop);
}
```

#<!-- 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=60000

C3p0配置：

[c3p0-config.xml](#)


```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>
  <default-config>
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/jdbc</property>
    <property name="user">root</property>
    <property name="password">root</property>

  </default-config>

</c3p0-config>
```

[c3p0.properties](#)

```
c3p0.driverClass=com.mysql.jdbc.Driver
c3p0.jdbcUrl=jdbc:mysql://localhost:3306/jdbc
c3p0.user=root
c3p0.password=root
```