jdbc

第1章：jdbc概述

1.1 数据的持久化

- 持久化(persistence)：**把数据保存到可掉电式存储设备中以供之后使用。**大多数情况下，特别是企业级应用，**数据持久化意味着将内存中的数据保存到硬盘**上加以"固化"，**而持久化的实现过程大多通过各种关系数据库来完成。**
- 持久化的主要应用是将内存中的数据存储在关系型数据库中，当然也可以存储在磁盘文件、XML数据文件中。

第2章：获取数据库连接

2.1 要素一：Driver接口实现类

2.1.1 Driver接口介绍

- java.sql.Driver 接口是所有 DBUtils 驱动程序需要实现的接口。这个接口是提供给数据库厂商使用的，不同数据库厂商提供不同的实现。
- 在程序中不需要直接去访问实现了 Driver 接口的类，而是由驱动程序管理器类(java.sql.DriverManager)去调用这些Driver实现。

```
- Oracle的驱动：oracle.jdbc.driver.OracleDriver
- mySql 的驱动：com.mysql.jdbc.Driver
```

- 将上述jar包拷贝到Java工程的一个目录中，习惯上新建一个lib文件夹，不同的idea有不同的操作。

2.1.2 加载与注册DBUtils驱动

- 加载驱动：加载 DBUtils 驱动需调用 Class 类的静态方法 forName()，向其传递要加载的 DBUtils 驱动的类名
  - **Class.forName("com.mysql.jdbc.Driver");**
- 注册驱动：DriverManager 类是驱动程序管理器类，负责管理驱动程序
  - **使用DriverManager.registerDriver(com.mysql.jdbc.Driver)来注册驱动**
  - 通常不用显式调用 DriverManager 类的 registerDriver() 方法来注册驱动程序类的实例，因为 Driver 接口的驱动程序类**都**包含了静态代码块，在这个静态代码块中，会调用 DriverManager.registerDriver() 方法来注册自身的一个实例。

2.2 要素二：URL

- DBUtils URL 用于标识一个被注册的驱动程序，驱动程序管理器通过这个 URL 选择正确的驱动程序，从而建立到数据库的连接。
- DBUtils URL的标准由三部分组成，各部分间用冒号分隔。
  - **jdbc:子协议:子名称**
  - **协议**：DBUtils URL中的协议总是jdbc
  - **子协议**：子协议用于标识一个数据库驱动程序
  - **子名称**：一种标识数据库的方法。子名称可以依不同的子协议而变化，用子名称的目的是为了**定位数据库**提供足够的信息。包含**主机名**(对应服务端的ip地址)，**端口号，数据库名**

- **几种常用数据库的 DBUtils URL**
  - MySQL的连接URL编写方式：
    - jdbc:mysql://主机名称:mysql服务端口号/数据库名称?参数=值&参数=值
    - jdbc:mysql://localhost:3306/xinzhi
    - jdbc:mysql://localhost:3306/xinzhi?useUnicode=true&characterEncoding=utf8（如果程序与服务器端的字符集不一致，会导致乱码，那么可以通过参数指定服务器端的字符集）
    - 8.0后需要加上&useSSL=false&serverTimezone=UTC"，MySQL在高版本需要指明是否进行SSL连（认证和加密），serverTimezone=Asia/Shanghai　使用UTC（世界统一时间）和中国的时间差八小时

  https://blog.csdn.net/wfanking/article/details/95504879

  - Oracle 的连接URL编写方式：
  - jdbc:oracle:thin:@主机名称:oracle服务端口号:数据库名称
  - jdbc:oracle:thin:@localhost:1521:xinzhi
- SQLServer的连接URL编写方式：
  - jdbc:sqlserver://主机名称:sqlserver服务端口号:DatabaseName=数据库名称
  - jdbc:sqlserver://localhost:1433:DatabaseName=xinzhi

2.3 要素三：用户名和密码

- user,password可以用“属性名=属性值”方式告诉数据库
- 可以调用 DriverManager 类的 getConnection() 方法建立到数据库的连接

2.4 数据库连接方式举例

```
/**
 * @author zn
 * @date 2020/12/15
 */
public class JdbcTest {

    @Test
    public void test1() throws Exception{
        //1.数据库连接的4个基本要素：
        String url = "jdbc:mysql://127.0.0.1:3306/kflb?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true&useSSL=false";
        String user = "root";
        String password = "root";
        //8.0之后名字改了  com.mysql.cj.jdbc.Driver
```

```
        //5.7之后名字改了  com.mysql.jdbc.Driver
        String driverName = "com.mysql.cj.jdbc.Driver";

        //2.实例化Driver
        Class clazz = Class.forName(driverName);
        Driver driver = (Driver) clazz.newInstance();
        //3.注册驱动
        DriverManager.registerDriver(driver);
        //4.获取连接
        Connection conn = DriverManager.getConnection(url, user, password);
        System.out.println(conn);
    }

    @Test
    public void test2() throws Exception{
        //1.数据库连接的4个基本要素：
        String url = "jdbc:mysql://127.0.0.1:3306/kflb?
useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Shanghai&allowPub
licKeyRetrieval=true&useSSL=false";
        String user = "root";
        String password = "root";
        //8.0之后名字改了  com.mysql.cj.jdbc.Driver
        //5.7之后名字改了  com.mysql.jdbc.Driver
        String driverName = "com.mysql.cj.jdbc.Driver";

        //2.实例化Driver
        Class.forName(driverName);
        //3.注册驱动

        //4.获取连接
        Connection conn = DriverManager.getConnection(url, user, password);
        System.out.println(conn);
    }

    @Test
    public void test3() throws Exception{
        //1.数据库连接的4个基本要素：
        String url = "jdbc:mysql://127.0.0.1:3306/kflb?
useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Shanghai&allowPub
licKeyRetrieval=true&useSSL=false";
        String user = "root";
        String password = "root";
        //8.0之后名字改了  com.mysql.cj.jdbc.Driver
        //5.7之后名字改了  com.mysql.jdbc.Driver
        String driverName = "com.mysql.cj.jdbc.Driver";

        //2.实例化Driver
        //3.注册驱动

        //4.获取连接
        Connection conn = DriverManager.getConnection(url, user, password);
        System.out.println(conn);
    }

    @Test
    public void test4() throws Exception{
        Properties properties = new Properties();
```

```
67
properties.load(JdbcTest.class.getClassLoader().getResourceAsStream("jdbc.properties"));
68          String url = properties.getProperty("mysql.url");
69          String user = properties.getProperty("mysql.username");
70          String password = properties.getProperty("mysql.password");
71
72          //2.实例化Driver
73          //3.注册驱动
74
75          //4.获取连接
76          Connection conn = DriverManager.getConnection(url, user, password);
77          System.out.println(conn);
78      }
79
80
81  }
```

MySQL8 提示Public Key Retrieval is not allowed错误解决方法

在使用jdbc连接到mysql时提示错误：

com.mysql.jdbc.exceptions.jdbc4.MySQLNonTransientConnectionException: Public Key Retrieval is not allowed
原因如下(参考官网给出的连接选项)：
　如果用户使用了 sha256_password 认证，密码在传输过程中必须使用 TLS 协议保护，但是如果 RSA 公钥不可用，可以使用服务器提供的公钥；可以在连接中通过 ServerRSAPublicKeyFile 指定服务器的 RSA 公钥，或者AllowPublicKeyRetrieval=True参数以允许客户端从服务器获取公钥；但是需要注意的是 AllowPublicKeyRetrieval=True可能会导致恶意的代理通过中间人攻击(MITM)获取到明文密码，所以默认是关闭的，必须显式开启。

所以可以用mysql_native_password,不要用sha256_password方式，就不会有问题了。

在驱动链dao接的后面添加参数allowPublicKeyRetrieval=true&useSSL=false

```
1  allowPublicKeyRetrieval=true&useSSL=false
```

> 说明：使用配置文件的方式保存配置信息，在代码中加载配置文件
>
> **使用配置文件的好处：**
>
> ①实现了代码和数据的分离，如果需要修改配置信息，直接在配置文件中修改，不需要深入代码
> ②如果修改了配置信息，省去重新编译的过程。

第3章：使用PreparedStatement

3.1 操作和访问数据库

- 数据库连接被用于向数据库服务器发送命令和 SQL 语句，并接受数据库服务器返回的结果。其实一个数据库连接就是一个Socket连接。
- 在 java.sql 包中有 3 个接口分别定义了对数据库的调用的不同方式：
  - **Statement**：用于执行静态 SQL 语句并返回它所生成结果的对象。
  - **PrepatedStatement：SQL** 语句被预编译并存储在此对象中，可以使用此对象多次高效地执行该语句。
  - CallableStatement：用于执行 SQL 存储过程（不学，有兴趣自己研究，但是得先学习存储过程）

3.2 使用Statement操作数据表的弊端

- 通过调用 Connection 对象的 createStatement() 方法创建该对象。该对象用于执行静态的 SQL 语句，并且返回执行结果。
- Statement 接口中定义了下列方法用于执行 SQL 语句：

```
int excuteUpdate(String sql)：执行更新操作INSERT、UPDATE、DELETE
ResultSet executeQuery(String sql)：执行查询操作SELECT
```

- 但是使用Statement操作数据表存在弊端：
  - **问题一：存在拼串操作，繁琐**
  - **问题二：存在SQL注入问题**
- **SQL 注入**是利用某些系统没有对用户输入的数据进行充分的检查，而在用户输入数据中注入非法的 SQL 语句段或命令(如：SELECT user, password FROM user_table WHERE user='a' OR 1 = ' AND password = ' OR '1' = '1') ，从而利用系统的 SQL 引擎完成恶意行为的做法。
- 对于 Java 而言，要防范 SQL 注入，只要用 PreparedStatement(从Statement扩展而来) 取代 Statement 就可以了。

3.2.1 体会增删改代码

```
/**
 * @author zn
 * @date 2020/12/15
 */
public class JdbcUtil {

    public static Connection getConnection(){
        Connection conn = null;
        try {
            Properties properties = new Properties();

 properties.load(JdbcTest.class.getClassLoader().getResourceAsStream("jdbc.properties"));
            String url = properties.getProperty("mysql.url");
            String user = properties.getProperty("mysql.username");
            String password = properties.getProperty("mysql.password");

            //2.实例化Driver
            //3.注册驱动
            //4.获取连接
```

```
            conn = DriverManager.getConnection(url, user, password);
        } catch (IOException | SQLException e){
            e.printStackTrace();
        }
        return conn;
    }
}

```

```
@Test
    public void testStatement1() {
        Connection connection = null;
        Statement statement = null;

        try {
            String sql1 = "insert into course values (2,'zhangsan',23),
(3,'zhangsan',23)";
            String sql2 = "update course set name = 1 where id > 10";
            connection = JdbcUtil.getConnection();
            statement = connection.createStatement();
            int i = statement.executeUpdate(sql2);
            System.out.println(i);

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }
        }


    }
```

3.2.2 体会查询代码

```
 @Test
    public void testStatement2()  {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

```

```
        try {
            String sql = "select * from course";
            connection = JdbcUtil.getConnection();
            statement = connection.createStatement();
            resultSet = statement.executeQuery(sql);
            List<Course> courses = new ArrayList<>();

            while (resultSet.next()){
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                int teacherId = resultSet.getInt("t_id");
                courses.add(new Course(id,name,teacherId));
            }
            System.out.println(courses);
        } catch (SQLException e){
            e.printStackTrace();
        } finally {
            if (connection != null){
                try {
                    connection.close();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }
            if (statement != null){
                try {
                    statement.close();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }
            if (resultSet != null){
                try {
                    resultSet.close();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
            }
        }


    }
```

3.2.3 代码优化

资源的释放

- 释放ResultSet, Statement,Connection。
- 数据库连接（Connection）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将导致系统宕机。Connection的使用原则是**尽量晚创建，尽量早的释放。**
- 可以在finally中关闭，保证及时其他代码出现异常，资源也一定能被关闭。

1、手动处理异常

2、合理关系资源

3、遍历处理数据

```java
public static void main(String[] args) {

    //1.数据库连接的4个基本要素：
    String url = "jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai";
    String user = "root";
    String password = "root";
    String driverName = "com.mysql.jdbc.Driver";
    String sql = "select id,name,age from user";

    //2.实例化Driver
    //抽离资源，方便合理关闭
    Connection conn = null;
    Statement statement = null;
    ResultSet resultSet = null;
    //手动处理异常
    try {
        Class clazz = Class.forName(driverName);
        Driver driver = (Driver) clazz.newInstance();
        //3.注册驱动
        DriverManager.registerDriver(driver);
        //4.获取连接
        conn = DriverManager.getConnection(url, user, password);

        statement = conn.createStatement();
        resultSet = statement.executeQuery(sql);
        //使用遍历获取数据
        while (resultSet.next()){
            int id = resultSet.getInt("id");
            String name = resultSet.getString("name");
            int age = resultSet.getInt("age");
            System.out.println("id=" + id);
            System.out.println("name=" + name);
            System.out.println("age=" + age);
        }

    } catch (Exception exception) {
        exception.printStackTrace();
    }finally {
        //关闭资源
        if(conn != null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(statement != null){
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(resultSet != null){
            try {
                resultSet.close();
```

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

3.2.4 公共提取

1、不管哪里要操作数据库都要获取连接，所以能不能提取

2、不管哪里都要关闭资源能不能提取

```
package com.xinzhi;

import java.sql.*;

/**
 * @author zn
 * @date 2020/4/1
 **/
public class DBUtil {
    public static Connection getConnection(){
        String url = "jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai";
        String user = "root";
        String password = "root";
        String driverName = "com.mysql.jdbc.Driver";

        Connection conn = null;
        try {
            Class clazz = Class.forName(driverName);
            Driver driver = (Driver) clazz.newInstance();
            //3.注册驱动
            DriverManager.registerDriver(driver);
            //4.获取连接
            conn = DriverManager.getConnection(url, user, password);
        }catch (Exception e){
            e.printStackTrace();
        }
        return conn;
    }

    public static void closeAll(Connection connection, Statement statement, ResultSet rs){
       if(connection != null){
           try {
               connection.close();
           } catch (SQLException e) {
               e.printStackTrace();
           }
       }
       if(statement != null){
           try {
               statement.close();
           } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
        }
        if( rs != null ){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }


}
```

```
public static void main(String[] args) {

    String sql = "select id,name,age from user";

    Connection conn = null;
    Statement statement = null;
    ResultSet resultSet = null;
    //手动处理异常
    try {
        conn = DBUtil.getConnection();
        statement = conn.createStatement();
        resultSet = statement.executeQuery(sql);
        //使用遍历获取数据
        while (resultSet.next()){
            int id = resultSet.getInt("id");
            String name = resultSet.getString("name");
            int age = resultSet.getInt("age");
            System.out.println("id=" + id);
            System.out.println("name=" + name);
            System.out.println("age=" + age);
        }
    } catch (Exception exception) {
        exception.printStackTrace();
    }finally {
        DBUtil.closeAll(conn,statement,resultSet);
    }
}
```

3.2.5 sql注入问题

因为SQL是一个拼接字符串的问题，所以会有攻击者使用一些特殊技巧完成一些操作，从而绕开我们的逻辑。

```
getUserById
```

因为sql语句是预编译的，而且**语句中使用了占位符，规定了sql语句的结构。用户可以设置"?"的值，但是不能改变sql语句的结构**，因此想在sql语句后面加上如“or 1=1”实现sql注入是行不通的。

3.3 PreparedStatement的使用

3.3.1 PreparedStatement介绍

- 可以通过调用 Connection 对象的 **preparedStatement(String sql)** 方法获取 PreparedStatement 对象
- **PreparedStatement 接口是 Statement 的子接口，它表示一条预编译过的 SQL 语句**
- PreparedStatement 对象所代表的 SQL 语句中的参数用问号(?)来表示，调用 PreparedStatement 对象的 setXxx() 方法来设置这些参数. setXxx() 方法有两个参数，第一个参数是要设置的 SQL 语句中的参数的索引(从 1 开始)，第二个是设置的 SQL 语句中的参数的值

3.3.2 PreparedStatement vs Statement

- 代码的可读性和可维护性。
- **PreparedStatement 能最大可能提高性能：**
  - DBServer会对**预编译**语句提供性能优化。因为预编译语句有可能被重复调用，所以语句在被DBServer的编译器编译后的执行代码被缓存下来，那么下次调用时只要是相同的预编译语句就不需要编译，只要将参数直接传入编译过的语句执行代码中就会得到执行。
  - 在statement语句中,即使是相同操作但因为数据内容不一样,所以整个语句本身不能匹配,没有缓存语句的意义.事实是没有数据库会对普通语句编译后的执行代码缓存。这样每执行一次都要对传入的语句编译一次。
  - (语法检查，语义检查，翻译成二进制命令，缓存)
- PreparedStatement 可以防止 SQL 注入

3.3.3 Java与SQL对应数据类型转换表

Java类型	SQL类型
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
String	CHAR,VARCHAR,LONGVARCHAR
byte array	BINARY , VAR BINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

3.3.4 使用PreparedStatement实现增、删、改操作

```
//通用的增、删、改操作（体现一：增、删、改 ；  体现二：针对于不同的表）
public void update(String sql,Object ... args){
    Connection conn = null;
    PreparedStatement ps = null;
    try {
        //1.获取数据库的连接
```

```
        conn = DBUtilsUtils.getConnection();

        //2.获取PreparedStatement的实例 (或：预编译sql语句)
        ps = conn.prepareStatement(sql);
        //3.填充占位符
        for(int i = 0;i < args.length;i++){
            ps.setObject(i + 1, args[i]);
        }

        //4.执行sql语句
        ps.execute();
    } catch (Exception e) {

        e.printStackTrace();
    }finally{
        //5.关闭资源
        DBUtilsUtils.closeResource(conn, ps);

    }
}
```

3.3.5 使用PreparedStatement实现查询操作

```
statement = conn.prepareStatement(sql);
statement.setInt(1,1);
resultSet = statement.executeQuery();
```

2、反射高级应用

```
    // 通用的针对于不同表的查询:返回一个对象 (version 1.0)
    public <T> T getInstance(Class<T> clazz, String sql, Object... args) {

        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            // 1.获取数据库连接
            conn = DBUtilsUtils.getConnection();

            // 2.预编译sql语句，得到PreparedStatement对象
            ps = conn.prepareStatement(sql);

            // 3.填充占位符
            for (int i = 0; i < args.length; i++) {
                ps.setObject(i + 1, args[i]);
            }

            // 4.执行executeQuery(),得到结果集：ResultSet
            rs = ps.executeQuery();

            // 5.得到结果集的元数据：ResultSetMetaData
            ResultSetMetaData rsmd = rs.getMetaData();

```

```
            // 6.1通过ResultSetMetaData得到columnCount,columnLabel；通过ResultSet得到列值
            int columnCount = rsmd.getColumnCount();
            if (rs.next()) {
                T t = clazz.newInstance();
                for (int i = 0; i < columnCount; i++) {// 遍历每一个列

                    // 获取列值
                    Object columnVal = rs.getObject(i + 1);
                    // 获取列的别名:列的别名，使用类的属性名充当
                    String columnLabel = rsmd.getColumnLabel(i + 1);
                    // 6.2使用反射，给对象的相应属性赋值
                    Field field = clazz.getDeclaredField(columnLabel);
                    field.setAccessible(true);
                    field.set(t, columnVal);

                }

                return t;

            }
        } catch (Exception e) {

            e.printStackTrace();
        } finally {
            // 7.关闭资源
            DBUtilsUtils.closeResource(conn, ps, rs);
        }

        return null;

    }
```

> 说明：使用PreparedStatement实现的查询操作可以替换Statement实现的查询操作，解决Statement拼串和SQL注入问题。

第6章： 数据库事务

6.1 DBUtils事务处理

- 数据一旦提交，就不可回滚。
- 数据什么时候意味着提交?
  - **当一个连接对象被创建时，默认情况下是自动提交事务**：每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。
  - **关闭数据库连接，数据就会自动的提交。**如果多个操作，每个操作使用的是自己单独的连接，则无法保证事务。即同一个事务的多个操作必须在同一个连接下。
- **DBUtils程序中为了让多个 SQL 语句作为一个事务执行:**
  - 调用 Connection 对象的 **setAutoCommit(false);** 以取消自动提交事务
  - 在所有的 SQL 语句都成功执行后，调用 **commit();** 方法提交事务
  - 在出现异常时，调用 **rollback();** 方法回滚事务

> 若此时 Connection 没有被关闭，还可能被重复使用，则需要恢复其自动提交状态 setAutoCommit(true)。尤其是在使用数据库连接池技术时，执行close()方法前，建议恢复自动提交状态。

【案例：用户AA向用户BB转账100】

```java
public void testDBUtilsTransaction() {
    Connection conn = null;
    try {
        // 1.获取数据库连接
        conn = DBUtilsUtils.getConnection();
        // 2.开启事务
        conn.setAutoCommit(false);
        // 3.进行数据库操作
        String sql1 = "update user_table set balance = balance - 100 where user = ?";
        update(conn, sql1, "AA");

        // 模拟网络异常
        //System.out.println(10 / 0);

        String sql2 = "update user_table set balance = balance + 100 where user = ?";
        update(conn, sql2, "BB");
        // 4.若没有异常，则提交事务
        conn.commit();
    } catch (Exception e) {
        e.printStackTrace();
        // 5.若有异常，则回滚事务
        try {
            conn.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    } finally {
        try {
            //6.恢复每次DML操作的自动提交功能
            conn.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        //7.关闭连接
        DBUtilsUtils.closeResource(conn, null, null);
    }
}

```


第七章、对资源的讨论

connection是一种稀有资源，一个连接建立就是创造了一个资源，我们思考一个问题，一个QQ连上了服务器对服务器而言就是建立了一个连接，这是有代价的。我们常常听说，同时在线人数太多，会导致服务崩溃，就是这么个道理。

那通常我们有什么解决方案呢?

第一种方案：就一个人玩就行了，我就是全服第一。

第二种方案：将服务器的人数限定一下，最多不能超过多少，超过了就排队，这当然不错，对吧。

第一种方案：单例

单例模式解决连接问题：

```
private static Connection connection = null;

public static Connection getConnection(){
    //调用方法时，如果发现没有初始化，就创建一个
    if(DBUtil.connection == null){
        String url = "jdbc:mysql://localhost:3306/xinzhishop?useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai";
        String user = "root";
        String password = "root";
        String driverName = "com.mysql.jdbc.Driver";

        try {
            DBUtil.connection = DriverManager.getConnection(url, user, password);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return DBUtil.connection;
}
```

```
private static Connection connection = null;

//只要类一家在咱们就搞一个连接
static {
    String url = "jdbc:mysql://localhost:3306/xinzhishop?useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai";
    String user = "root";
    String password = "root";
    String driverName = "com.mysql.jdbc.Driver";

    try {
        DBUtil.connection = DriverManager.getConnection(url, user, password);
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
}

public static Connection getConnection(){
    return DBUtil.connection;
}
```

单例只是一种思想；

1、connection只从一个地方获取

2、实例化的时间定义他是懒汉式还是恶汉式

3、懒汉式：就是不到关键时候就不动，用的时候才创建对象

4、饿汉式：就是二话不说类一加载就创建对象

通常的单例类的写法

```
public class Dog {
    //初始化一个空对象
    private static Dog dog = null;
    //私有化构造方法，让别的地方不能new
    private Dog(){}
    //获取实例的方法
    public Dog getInstance(){
        //看看是不是null，是null，再new一个
        if( dog == null ){
            Dog.dog = new Dog();
        }
        return dog;
    }

}
```

```
/**
 * @author zn
 * @date 2020/4/3
 **/
public class Dog {
    //静态的变量一加载就初始化
    private static Dog dog = new Dog();
    //私有化构造方法，让别的地方不能new
    private Dog(){}
    //只能通过这个静态方法获取实例
    public static Dog getInstance(){
        return dog;
    }

}
```

第二种方案：数据库连接池

1、 JDBC数据库连接池的必要性

- 在使用开发基于数据库的web程序时，传统的模式基本是按以下步骤：
  - **在主程序中建立数据库连接**
  - **进行sql操作**
  - **断开数据库连接**
- 这种模式开发，存在的问题:
  - 普通的JDBC数据库连接使用 DriverManager 来获取，每次向数据库建立连接的时候都要将 Connection 加载到内存中，再验证用户名和密码(得花费0.05s～1s的时间)。需要数据库连接的时候，就向数据库要求一个，执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。**数据库的连接资源并没有得到很好的重复利用。**若同时有几百人甚至几千人在线，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃。
  - **对于每一次数据库连接，使用完后都得断开。**否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库。（回忆：何为Java的内存泄漏？）
  - **这种开发不能控制被创建的连接对象数**，系统资源会被毫无顾及的分配出去，如连接过多，也可能导致内存泄漏，服务器崩溃。

2、 数据库连接池技术

- 为解决传统开发中的数据库连接问题，可以采用数据库连接池技术。
- **数据库连接池的基本思想**：就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。
- **数据库连接池**负责分配、管理和释放数据库连接，它**允许应用程序重复使用一个现有的数据库连接，而不是重新建立一个。**
- 数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由**最小数据库连接数来设定**的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的**最大数据库连接数量**限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。

- **数据库连接池技术的优点**

  1. 资源重用

  由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。

  2. 更快的系统反应速度

  数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间

  3. 新的资源分配手段

  对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置，实现某一应用最大可用数据库连接数的限制，避免某一应用独占所有的数据库资源

  4. 统一的连接管理，避免数据库连接泄漏

  在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露

3、 多种开源的数据库连接池

- DataSource 通常被称为数据源，它包含连接池和连接池管理两个部分，习惯上也经常把 DataSource 称为连接池
- **DataSource用来取代DriverManager来获取Connection，获取速度快，同时可以大幅度提高数据库访问速度。**
- 特别注意：
  - 数据源和数据库连接不同，数据源无需创建多个，它是产生数据库连接的工厂，因此**整个应用只需要一个数据源即可。**
  - 当数据库访问结束后，程序还是像以前一样关闭数据库连接：conn.close(); 但conn.close()并没有关闭数据库的物理连接，它仅仅把数据库连接释放，归还给了数据库连接池。

（1） C3P0数据库连接池（太老了，不学）

（2） DBCP数据库连接池（太老了，不学）

（3） Druid（德鲁伊）数据库连接池

引入一个jar包

```
druid-1.1.17.jar
```

Druid是阿里巴巴开源平台上一个数据库连接池实现，它结合了C3P0、DBCP、Proxool等DB池的优点，同时加入了日志监控，可以很好的监控DB池连接和SQL的执行情况，可以说是针对监控而生的DB连接池，**可以说是目前最好的连接池之一。**

```
package com.atguigu.druid;

import java.sql.Connection;
import java.util.Properties;

import javax.sql.DataSource;

import com.alibaba.druid.pool.DruidDataSourceFactory;

public class TestDruid {
    public static void main(String[] args) throws Exception {
        Properties pro = new Properties();
pro.load(TestDruid.class.getClassLoader().getResourceAsStream("druid.properties"));
        DataSource ds = DruidDataSourceFactory.createDataSource(pro);
        Connection conn = ds.getConnection();
        System.out.println(conn);
    }
}

```

配置文件为：【druid.properties】

```
url=jdbc:mysql://localhost:3306/xinzhishop?rewriteBatchedStatements=true
username=root
password=root
driverClassName=com.mysql.jdbc.Driver

```

```
initialSize=10
maxActive=20
maxWait=1000
filters=wall


1、初始化时建立物理连接的个数 默认0
2、最大连接池数量  默认8
3、获取连接时最大等待时间，单位毫秒。
4、防御sql注入的filter:wall
```

（4） Hikari（光）数据库连接池

引入四个jarbao：

```
HikariCP-3.4.2.jar
slf4j-api-1.7.29.jar
slf4j-log4j12-1.7.21.jar
log4j-1.2.17.jar
```

> 号称历史上最快的数据库连接池

```
public static Connection getConnection(){
    //1.数据库连接的4个基本要素：
    Connection connection = null;
    InputStream in =
UserDaoImpl.class.getClassLoader().getResourceAsStream("config/jdbc.config")
;
    Properties properties = new Properties();
    try {
        properties.load(in);
    } catch (IOException e) {
        e.printStackTrace();
    }

    HikariConfig hikariConfig = new HikariConfig(properties);
    DataSource source = new HikariDataSource(hikariConfig);
    try {
        connection = source.getConnection();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return connection;
}
```

```
jdbcUrl=jdbc:mysql://localhost:3306/xinzhishop
username=root
password=root
driverClassName=com.mysql.jdbc.Driver
```

```
idleTimeout=600000
connectionTimeout=30000
minimumIdle=10
maximumPoolSize=60

1、保持连接的最大时长，比如连接多了，最小连接数不够用，就会继续创建，比如又创建了10个，如果这时没有了业务，超过该设置的时间，新创建的就会被关闭
2、连接的超时时间，比如网络不好，一个连接超过折磨常时间没有创建好就不创建了
3、连接池最少的连接数
4、连接池最大的连接数
```

看见红色的日志难受

需要在src下建立一个log4j.properties文件内容，这个文件告诉我们启动连接池的一些日志信息

```
log4j.rootLogger=debug, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=firestorm.log

log4j.appender.R.MaxFileSize=100KB
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n

log4j.logger.com.codefutures=DEBUG
```

第八章、搞一个通用DAO

- DAO：Data Access Object访问数据信息的类和接口，包括了对数据的CRUD（Create、Retrival、Update、Delete），而不包含任何业务相关的信息。
- 在所有的dao中会有很多重复性的工作，我们可以封装一个父类来完成此类重复工作，我们称之为BaseDAO。

1、入门级basedao

```
/**
 * @author zn
 * @date 2020/4/4
 **/
public class BaseDaoImpl implements IBaseDao {

    private static DataSource DATA_SOURCE = null;

    static {
```

```
        InputStream in = UserDaoImpl.class.getClassLoader().getResourceAsStream("config/jdbc.config");
        Properties properties = new Properties();
        try {
            properties.load(in);
        } catch (IOException e) {
            e.printStackTrace();
        }

        HikariConfig hikariConfig = new HikariConfig(properties);
        BaseDaoImpl.DATA_SOURCE = new HikariDataSource(hikariConfig);
    }

    public Connection getConnection() {
        if (BaseDaoImpl.DATA_SOURCE != null) {
            try {
                return BaseDaoImpl.DATA_SOURCE.getConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    public void closeAll(Connection conn,Statement statement, ResultSet resultSet) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void save(String sql, Object... params) {
        Connection conn = null;
        PreparedStatement statement = null;
        try {
            conn = getConnection();
            statement = conn.prepareStatement(sql);
            for (int i = 1; i <= params.length; i++) {
                statement.setObject(i, params[i]);
```

```
            }
            statement.execute();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, null);
        }
    }
}
```

2、大神级basedao

终极目标，少部分人能看懂，会用即可

```
package com.xinzhi.dao;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.List;

/**
 * @author zn
 * @date 2020/4/4
 **/
public interface IBaseDao<T> {

    /**
     * 获取连接的方法
     * @return
     */
    Connection getConnection();

    /**
     * 关闭资源的方法
     * @param statement
     * @param resultSet
     */
    void closeAll(Statement statement, ResultSet resultSet);

    /**
     * 通用的保存方法
     * @param sql
     * @param params
     */
    void save(String sql,Object... params);

    /**
     * 高级部分，大神写的
     * 有要求 数据库的名字和类名必须一样
     *         每个字段和属性的名字也要一样
     * 有规矩好办事
     */


```

```
    /**
     * 通用的查询所有的结果的方法
     * @param clazz
     * @return
     */
    List<T> findAll(Class clazz);

    /**
     * 通用保存的方法
     * @param obj
     */
    void save(Object obj);

    /**
     * 通用的更新方法
     * @param obj
     * @param fieldName
     * @param fieldValue
     */
    void update(Object obj,String fieldName,Object fieldValue);

    /**
     * 通用的删除方法
     * @param clazz
     * @param fieldName
     * @param fieldValue
     */
    void delete(Class clazz,String fieldName,Object fieldValue);

    /**
     * 通用的查找一个方法
     * @param clazz
     * @param fieldName
     * @param fieldValue
     * @return
     */
    T findOne(Class clazz,String fieldName,Object fieldValue);
}

```

```
package com.xinzhi.dao.impl;

import com.xinzhi.dao.IBaseDao;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Field;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

```

```java
/**
 * @author zn
 * @date 2020/4/4
 **/
public class BaseDaoImpl<T> implements IBaseDao<T> {

    private static DataSource DATA_SOURCE = null;

    static {
        InputStream in = UserDaoImpl.class.getClassLoader().getResourceAsStream("config/jdbc.config");
        Properties properties = new Properties();
        try {
            properties.load(in);
        } catch (IOException e) {
            e.printStackTrace();
        }

        HikariConfig hikariConfig = new HikariConfig(properties);
        BaseDaoImpl.DATA_SOURCE = new HikariDataSource(hikariConfig);
    }

    public Connection getConnection() {
        if (BaseDaoImpl.DATA_SOURCE != null) {
            try {
                return BaseDaoImpl.DATA_SOURCE.getConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    public void closeAll(Connection conn,Statement statement, ResultSet resultSet) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

```
71
72      //简单的通用保存，通过可变参数赋值
73      public void save(String sql, Object... params) {
74          PreparedStatement statement = null;
75          try {
76              Connection conn = getConnection();
77              statement = conn.prepareStatement(sql);
78              for (int i = 1; i <= params.length; i++) {
79                  statement.setObject(i, params[i]);
80              }
81              statement.execute();
82          } catch (Exception e) {
83              e.printStackTrace();
84          } finally {
85              closeAll(statement, null);
86          }
87      }
88
89      /**
90       * 高级部分
91       * 有要求 数据库的名字和类名必须一样
92       * 每个字段和属性的名字也要一样
93       * 有规矩好办事,重在体会思想
94       * 搞明白还能这么干就行了
95       * 思路:
96       * 因为规定了数据库名称和类名形同，字段也相同
97       * 所有可以通过反射获取类名和字段名拼接一个字符串
98       *
99       * @return
100      */
101
102
103     public List<T> findAll(Class clazz) {
104         //拼一个sql   select id,username,password from user
105         //其中id,username,password可变但是他是类的字段啊
106         //user可变但是他是类名啊，反射登场了
107         List<T> list = new ArrayList<>();
108         PreparedStatement statement = null;
109         ResultSet resultSet = null;
110         try {
111             //利用反射拼出一个select语句
112             Field[] fields = clazz.getDeclaredFields();
113             StringBuilder fieldStr = new StringBuilder();
114             fieldStr.append("select ");
115             for (Field field : fields) {
116                 fieldStr.append(field.getName().toLowerCase()).append(",");
117             }
118             fieldStr.deleteCharAt(fieldStr.length() - 1);
119             fieldStr.append(" from ");
120
 fieldStr.append(clazz.getName().toLowerCase().substring(clazz.getName().la
stIndexOf(".") + 1));
121
122             Connection conn = getConnection();
123             statement = conn.prepareStatement(fieldStr.toString());
124             resultSet = statement.executeQuery();
125
126             while (resultSet.next()) {
```

```
                Object obj = clazz.newInstance();
                for (Field field : fields) {
                    Object value = resultSet.getObject(field.getName());
                    field.setAccessible(true);
                    field.set(obj, value);
                }
                list.add((T) obj);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, resultSet);
        }
        return list;
    }


    public void save(Object obj) {
        Class clazz = obj.getClass();
        Field[] fields = clazz.getDeclaredFields();

        //拼接出一个insert语句
        StringBuilder sql = new StringBuilder("insert into ");

 sql.append(clazz.getName().toLowerCase().substring(clazz.getName().lastInd
exOf(".") + 1))
                .append(" (");
        for (Field field : fields) {
            sql.append(field.getName().toLowerCase()).append(",");
        }
        sql.deleteCharAt(sql.length() - 1);
        sql.append(") values (");
        for (Field field : fields) {
            sql.append("?,");
        }
        sql.deleteCharAt(sql.length() - 1);
        sql.append(")");
        System.out.println(sql);

        PreparedStatement statement = null;
        try {
            Connection conn = getConnection();
            statement = conn.prepareStatement(sql.toString());
            for (int i = 0; i < fields.length; i++) {
                fields[i].setAccessible(true);
                statement.setObject(i + 1, fields[i].get(obj));
            }

            statement.execute();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, null);
        }
    }


    @Override
```

```
    public void update(Object obj, String fieldName, Object fieldValue) {
        PreparedStatement statement = null;
        try {
            Class clazz = obj.getClass();

            //拼接出一个update语句
            StringBuilder sql = new StringBuilder("update " +
clazz.getName().toLowerCase().substring(clazz.getName().lastIndexOf(".") +
1)
                    + " set ");
            Field[] fields = clazz.getDeclaredFields();
            for (Field field : fields) {
                field.setAccessible(true);

 sql.append(field.getName()).append("=").append("?").append(",");
            }
            sql.deleteCharAt(sql.length() - 1);
            sql.append(" where ").append(fieldName).append("=?");
            System.out.println(sql);

            Connection conn = getConnection();
            statement = conn.prepareStatement(sql.toString());
            for (int i = 0; i < fields.length; i++) {
                fields[i].setAccessible(true);
                statement.setObject(i + 1, fields[i].get(obj));
            }
            statement.setObject(fields.length + 1, fieldValue);
            statement.execute();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, null);
        }
    }

    @Override
    public void delete(Class clazz, String fieldName, Object fieldValue) {
        //拼接一个delete语句
        String sql = "delete from " +
clazz.getName().toLowerCase().substring(clazz.getName().lastIndexOf(".") +
1)
                + " where " + fieldName + "=?";
        System.out.println(sql);
        PreparedStatement statement = null;
        try {
            Connection conn = getConnection();
            statement = conn.prepareStatement(sql);
            statement.setObject(1, fieldValue);
            statement.execute();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, null);
        }
    }

    @Override
    public T findOne(Class clazz, String fieldName, Object fieldValue) {
```

```
        T t = null;

        PreparedStatement statement = null;
        ResultSet resultSet = null;
        try {
            Field[] fields = clazz.getDeclaredFields();
            //拼接一个语句
            StringBuilder sql = new StringBuilder();
            sql.append("select ");
            for (Field field : fields) {
                sql.append(field.getName().toLowerCase()).append(",");
            }
            sql.deleteCharAt(sql.length() - 1);
            sql.append(" from ");

 sql.append(clazz.getName().toLowerCase().substring(clazz.getName().lastInd
exOf(".") + 1))
                    .append(" where " + fieldName + "=?");

            System.out.println(sql.toString());

            Connection conn = getConnection();
            statement = conn.prepareStatement(sql.toString());
            statement.setObject(1,fieldValue);
            resultSet = statement.executeQuery();

            while (resultSet.next()) {
                Object obj = clazz.newInstance();
                for (Field field : fields) {
                    Object value = resultSet.getObject(field.getName());
                    field.setAccessible(true);
                    field.set(obj, value);
                }
                t = (T) obj;
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            closeAll(conn,statement, resultSet);
        }
        return t;
    }
}
```

第九章、数据库事务

1、JDBC事务处理

- 数据一旦提交，就不可回滚。
- 数据什么时候意味着提交?
  - **当一个连接对象被创建时，默认情况下是自动提交事务**：每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。
  - **关闭数据库连接，数据就会自动的提交。**如果多个操作，每个操作使用的是自己单独的连接，则无法保证事务。即同一个事务的多个操作必须在同一个连接下。

- **JDBC程序中为了让多个 SQL 语句作为一个事务执行：**
  - 调用 Connection 对象的 **setAutoCommit(false);** 以取消自动提交事务
  - 在所有的 SQL 语句都成功执行后，调用 **commit();** 方法提交事务
  - 在出现异常时，调用 **rollback();** 方法回滚事务

> 若此时 Connection 没有被关闭，还可能被重复使用，则需要恢复其自动提交状态setAutoCommit(true)。尤其是在使用数据库连接池技术时，执行close()方法前，建议恢复自动提交状态。

【案例：用户AA向用户BB转账100】

```
public void testJDBCTransaction() {
    Connection conn = null;
    try {
        // 1.获取数据库连接
        conn = JDBCUtils.getConnection();
        // 2.开启事务
        conn.setAutoCommit(false);
        // 3.进行数据库操作
        String sql1 = "update user set balance = balance - 100 where id =
?";
        update(conn, sql1, "AA");

        // 模拟网络异常
        //System.out.println(10 / 0);
                                                ;
        update(conn, sql2, "BB");
        // 4.若没有异常，则提交事务
        conn.commit();
    } catch (Exception e) {
        e.printStackTrace();
        // 5.若有异常，则回滚事务
        try {
            conn.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    } finally {
        try {
            //6.恢复每次DML操作的自动提交功能
            conn.setAutoCommit(true);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        //7.关闭连接
        JDBCUtils.closeResource(conn, null, null);
    }
}

```

2、代码优化

将获取连接放在service层，到层传递，因为事务都在service层。