1、课程名称：JDBC

2、知识点

2.1、本次预计讲解的知识点

1、 JDBC 的基本概念及分类
2、 使用 JDBC 进行数据库表的操作
3、 Statement、PreparedStatement、ResultSet 接口的使用
4、 事务处理的概念及应用
7、 DAO

3、具体内容

3.1、JDBC 简介（了解）

JDBC：Java Database Connection，表示数据库连接，是 java 中专门提供的一组用于操作数据库的标准，所有的数据库生产商如果要是想为 java 提供支持，则必须支持此标准，既然是标准的话，所以说 JDBC 实际上是一套类库的接口。

主要的操作类和接口：Connection 接口、Statement 接口、PreapredStatement 接口、ResultSet 接口、DriverManager 类

JDBC 的主要分类：

1、 JDBC-ODBC 桥连接：使用微软的 ODBC 进行数据库的连接，JDBC → ODBC → DB，因为中间加入了一个 ODBC 的过渡，所以这样的操作性能将是非常低的。所以，只要是开发就不会有人使用此方式。但是此种方式是 SUN 本身提供的最原始连接，所以所有的操作类库都是最新的。

2、 JDBC 连接，直接使用各个数据库生产商提供的数据库连接程序，进行数据库操作，JDBC → DB，这样的性能较高，所以在开发中基本上都使用此种形式。

3、 JDBC 网络连接，在正常操作中，不可能每一台电脑上都安装一个数据库。通过网络连接不同主机的数据库，此操作也将由各个数据库生产商提供支持。

3.2、JDBC 操作前的准备（理解）

现在要使用 JDBC 连接 MySQL 数据库，需要经过以下两个步骤。

3.2.1、建立 Person 表

下面建立一张 Person 表。

No.	字段名称	字段类型	描述

1	pid	int(11)	编号，使用序列进行自动增长操作
2	name	varchar(50)	姓名
3	sex	char(2)	性别
4	birthday	date	生日
5	salary	float(7,2)	工资，小数表示

通过以上的表结构，写出数据库的创建脚本：

```
DROP TABLE person ;
CREATE TABLE person(
    pid         INT(11)          PRIMARY KEY NOT NULL ,
    name        VARCHAR(20)  NOT NULL ,
    sex         CHAR(1)   NOT NULL ,
    birthday    DATE      ,
    salary              FLOAT(7,2)
) ;
```

3.3、连接数据库（重点）

如果要想连接数据库，则必须要有以下的几种信息：

1、 数据库的驱动程序地址：jdbc:mysql://localhost:3306
2、 数据库的连接地址：mysql://localhost:3306/test
3、 用户名：root
4、 密码：root

但是，还需要以下的几个类支持：

1、 Class 类，通过 Class 类加载驱动程序
2、 DriverManager 类，通过 DriverManager 管理驱动，并可以取得连接，连接时需要地址、用户名、密码
3、 连接通过 Connection 进行接收

范例：进行数据库的连接操作

```
package org.vince.jdbcdemo01;
import java.sql.Connection;
import java.sql.DriverManager;
public class ConnectionDemo01 {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        System.out.println(conn);
        // 3、关闭
        conn.close();
```

```
    }
}
```

3.4、使用 Statement 接口进行数据库的更新操作（重点）

如果要想进行数据库的操作的话，则必须使用 Statement 接口，那么，如果要执行的是更新操作，则使用 Statement 接口中的以下方法：

No.	方法名称	类型	描述
1	int[] executeBatch() throws SQLException	普通	批处理操作
2	int executeUpdate(String sql) throws SQLException	普通	执行数据库的更新操作，返回更新的行数

如果要想为 Statement 接口实例化，则就需要使用 Connection 接口的如下方法：

No.	方法名称	类型	描述
1	Statement createStatement() throws SQLException	普通	取得 Statement 实例

3.4.1、执行数据插入操作

如果要想执行插入操作，则必须编写插入的 SQL 语法，SQL 语法格式：“INSERT INTO 表名称(列,..) VALUES (值,..)”

范例：执行插入

```
package org.vince.jdbcdemo02;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class StatementInsertDemo01 {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        Statement stmt = null; // 数据库操作
        String sql = "INSERT INTO person(pid,name,sex,birthday,salary) "
                + "VALUES (1,'张三','男',now(),899)";
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        stmt = conn.createStatement(); // 取得Statement实例
        stmt.executeUpdate(sql); // 执行SQL语句
        // 3、关闭
        stmt.close();
        conn.close();
    }
}
```

3.4.2、执行数据更新操作

使用 UPDATE 语法就可以完成数据库的更新操作：UPDATE 表名称 SET 列=值,… WHERE 更新条件

```
package org.vince.jdbcdemo02;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class StatementUpdateDemo {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        Statement stmt = null; // 数据库操作
        int pid = 3;
        String name = "赵六";
        String sex = "女";
        float salary = 9000.0f;
        String sql = "UPDATE person SET name='" + name + "',sex='" + sex
                + "',salary=" + salary + " WHERE pid=" + pid;
        System.out.println(sql);
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        stmt = conn.createStatement(); // 取得Statement实例
        stmt.executeUpdate(sql); // 执行SQL语句
        // 3、关闭
        stmt.close();
        conn.close();
    }
}
```

3.4.3、执行数据库删除操作

使用 DELETE 执行数据库的删除操作，语法：DELETE FROM 表名称 WHERE 删除条件。

范例：删除编号是 4 的人员

```
package org.vince.jdbcdemo02;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```

```
public class StatementDeleteDemo {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        Statement stmt = null; // 数据库操作
        int pid = 4;
        String sql = "DELETE FROM person WHERE pid=" + pid;
        System.out.println(sql);
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        stmt = conn.createStatement(); // 取得Statement实例
        int len = stmt.executeUpdate(sql); // 执行SQL语句
        System.out.println("更新了" + len + "行数据！ ");
        // 3、关闭
        stmt.close();
        conn.close();
    }
}
```

3.5、使用 ResultSet 接口接收查询结果（重点）

对于数据库操作来讲，每次执行查询之后，实际上都将返回一组查询的结果。如果要想使用 Statement 接口进行查询操作的话，则全部的查询的结果应该返回到程序之中，程序中就要求使用 ResultSet 接口进行接收。

在 Statement 接口中有如下的方法，可以执行查询操作

No.	方法名称	类型	描述
1	ResultSet executeQuery(String sql) throws SQLException	普通	数据库查询操作，所有的查询结果使用 ResultSet 接收

所有的结果使用了 ResultSet 接收之后，实际上下面就需要从 ResultSet 中取回数据。在 ResultSet 中定义了如下方法：

No.	方法名称	类型	描述
1	Date getDate(String columnLabel) throws SQLException	普通	根据指定的列名称，得到日期
2	Date getDate(int columnIndex) throws SQLException	普通	根据列的编号，得到日期
3	String getString(int columnIndex) throws SQLException	普通	根据列的编号，得到字符串
4	String getString(String columnLabel) throws SQLException	普通	根据列名称，得到字符串
5	boolean next() throws SQLException	普通	将指针向下移动，同时判断是否还有内容

实际上在此接口中定义的全部的 get 方法都是有其对应的具体数据类型的。

范例：查询数据库

```
package org.vince.jdbcdemo03;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Date;
public class ResultDemo01 {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        Statement stmt = null; // 数据库操作
        ResultSet rs = null; // 接收查询结果
        // 在查询时，必须明确的写出要查询的具体列，这是一个明确的开发标准
        String sql = "SELECT pid,name,sex,birthday,salary FROM person";
        System.out.println(sql);
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        stmt = conn.createStatement(); // 取得Statement实例
        rs = stmt.executeQuery(sql);// 执行查询操作
        while (rs.next()) {// 指针向下移动
            int pid = rs.getInt("pid");
            String name = rs.getString("name");
            String sex = rs.getString("sex");
            Date date = rs.getDate("birthday");
            float salary = rs.getFloat("salary");
            System.out.println("编号：" + pid + "，姓名：" + name + "，性别：" + sex
                    + "，生日：" + date + "，工资：" + salary);
        }
        // 3、关闭
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

但是，以上的操作毕竟比较麻烦，因为在写的时候要明确的写出列名称，所以在开发中往往都使用编号取出数据：

```
        while (rs.next()) {// 指针向下移动
            int pid = rs.getInt(1);
            String name = rs.getString(2);
            String sex = rs.getString(3);
            Date date = rs.getDate(4);
```

```
            float salary = rs.getFloat(5);
            System.out.println("编号：" + pid + "，姓名：" + name + "，性别：" + sex
                    + "，生日：" + date + "，工资：" + salary);
        }
```

而且，在使用 ResultSet 接口进行输出的时候，不能混乱改变取得数据的顺序。此 bug 已经修复了。

JDBC 程序访问数据库的步骤

步骤一：加载 JDBC 驱动程序
步骤二：提供连接 URL
步骤三：建立一个数据库的连接
步骤四：创建一个 statement
步骤五：执行 SQL 语句
步骤六：处理结果
步骤七：关闭 JDBC 对象

3.6、PreparedStatement（绝对重点）

从实际的开发来看，所有的数据库的 CRUD 操作，都将使用 PreparedStatement 完成，因为此种方式性能较高，而且安全性较高。

因为 PreparedStatement 使用预处理的操作完成，而 Statement 是直接拼凑 SQL 语句完成。

在预处理操作中使用“?”进行占位的操作，占位之后，可以向里面设置各个内容进去。

3.6.1、增加操作

```
package org.vince.jdbcdemo04;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.text.SimpleDateFormat;
public class PreparedStatementInsertDemo {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        PreparedStatement pstmt = null; // 数据库操作
        String name = "Mr'Smith";
        String sex = "中";
        String birthday = "1990-02-14";
        float salary = 90.0f;
        String sql = "INSERT INTO person(pid,name,sex,birthday,salary) "
                + "VALUES (1,?,?,?,?)";
        System.out.println(sql);
```

```
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        pstmt = conn.prepareStatement(sql); // 实例化PreparedStatement对象
        pstmt.setString(1, name);
        pstmt.setString(2, sex);
        pstmt.setDate(3, new java.sql.Date(new SimpleDateFormat("yyyy-MM-dd")
                .parse(birthday).getTime()));
        pstmt.setFloat(4, salary);
        pstmt.executeUpdate() ;
        // 3、关闭
        pstmt.close();
        conn.close();
    }
}
```

3.6.2、查询操作

```
package org.vince.jdbcdemo04;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Date;
public class PreparedStatementQueryDemo {
    public static final String DBDRIVER = "com.mysql.jdbc.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/test";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static void main(String[] args) throws Exception {
        Connection conn = null; // 用于接收数据库连接
        PreparedStatement pstmt = null; // 数据库操作
        ResultSet rs = null; // 接收查询结果
        // 在查询时，必须明确的写出要查询的具体列，这是一个明确的开发标准
        String sql = "SELECT pid,name,sex,birthday,salary FROM person";
        System.out.println(sql);
        // 1、加载数据库驱动程序
        Class.forName(DBDRIVER);
        // 2、数据库连接
        conn = DriverManager.getConnection(DBURL, DBUSER, DBPASS); // 数据库连接
        pstmt = conn.prepareStatement(sql) ;
        rs = pstmt.executeQuery(sql);// 执行查询操作
        while (rs.next()) {// 指针向下移动
```

```
            int pid = rs.getInt(1);
            String name = rs.getString(2);
            String sex = rs.getString(3);
            Date date = rs.getDate(4);
            float salary = rs.getFloat(5);
            System.out.println("编号：" + pid + "，姓名：" + name + "，性别：" + sex
                    + "，生日：" + date + "，工资：" + salary);
        }
        // 3、关闭
        rs.close();
        pstmt.close();
        conn.close();
    }
}
```

在开发中一定要明确的是，绝对必须严格的使用 PreparedStatement，这是最重要的。

3.7、事务处理（重点）

事务：　所有的操作要么同时成功，要么同时失败。

在 Oracle 中提供了 Commit、Rollback 命令进行事务的提交与回滚。

实际上在 JDBC 中也存在事务处理，如果要想进行事务处理的话，则必须按照以下的步骤完成：

1、 要取消掉 JDBC 的自动提交：void setAutoCommit(boolean autoCommit) throws SQLException

2、 执行各个 SQL 语句，加入到批处理之中。

3、 执行操作，如果执行成功，则手工提交：void commit() throws SQLException

- 如果出现了错误，则回滚：void rollback() throws SQLException

3.8、封装 JDBC 操作数据库的工具类（重点）

为了提高代码的重用性，回忆 JDBC 操作数据库的代码，封装一个 JDBC 操作数据库的工具类：DbUtils.java，把一些非常公用的代码抽取成方法以便使用。

```
public class DBUtil {
    private final static String USERNAME = "root";
    private final static String PASSWORD = "root";
    private final static String URL = "jdbc:mysql://localhost:3306/test";
    static {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
            }
        }
        public static Connection getConnection() {
            Connection conn = null;
            try {
                conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return conn;
        }
        public static void close(ResultSet rs, Statement state, Connection conn) {
            try {
                if (rs != null)
                    rs.close();
                if (state != null)
                    state.close();
                if (conn != null)
                    conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
}
```

```
public class UseDBUtilTest {
        public void delete(Integer id){
            Connection conn = null;
            PreparedStatement ps = null;
            String sql = "delete from student where id=?";
            try{
                conn = DBUtil.getConnection();
                ps = conn.prepareStatement(sql);
                ps.setInt(1, id);
                int  i = ps.executeUpdate();
            }catch(SQLException e){
                e.printStackTrace();
            }finally{
                DBUtil.close(null, ps, conn);
            }
        }
        public static void main(String[] args) {
            // TODO Auto-generated method stub
            UseDBUtilTest test = new UseDBUtilTest();
```

```
            test.delete(3);
        }
}
```

实际开发中为了提高 JDBC 程序连接不同数据库的能力，经常把连接数据库的 URL、用户名，密码等信息编写在一个属性文件(jdbc.properties)中，请照这个思想对 DbUtils.java 进行再一次的封装。

```
public class DBUtil {
        private    static String USERNAME = "";
        private    static String PASSWORD = "";
        private    static String URL = "";
        private DBUtil(){}
        static {
                getProperties();
                try {
                        Class.forName("com.mysql.jdbc.Driver").newInstance();
                } catch (InstantiationException e) {
                        e.printStackTrace();
                } catch (IllegalAccessException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
        }
        /**
         * 获取属性文件中的参数值
         */
        private static void getProperties() {
                //通过指定文件名从当前线程上下文中获取资源输入流
                InputStream input = Thread.currentThread().getContextClassLoader()
                                .getResourceAsStream("test/jdbc/jdbc.properties");
                //实例化属性文件操作类
                Properties config = new Properties();
                try {
                        //加载输入流
                        config.load(input);
                        //获取属性文件中的属性值
                        USERNAME = config.getProperty("username");
                        PASSWORD = config.getProperty("password");
                        URL = config.getProperty("url");
                        //关闭输入流
                        input.close();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
```

```
    }

    /**
     * 获取数据库连接
     * @return
     */
    public static Connection getConnection() {
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return conn;
    }
    /**
     * 关闭数据库连接
     * @param rs
     * @param state
     * @param conn
     */
    public static void close(ResultSet rs, Statement state, Connection conn) {
        try {
            if (rs != null)
                rs.close();
            if (state != null)
                state.close();
            if (conn != null)
                conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        DBUtil.getConnection();
        System.out.println("连接成功");
    }
}
```

test.jdbc.jdbc.properties 文件

```
username=root
password=root
url=jdbc:mysql://localhost:3306/test
```

3.9、DAO 设计模式（重点）

什么是 DAO?

DAO 是 Data Access Object 数据访问接口，数据访问：故名思义就是与数据库打交道。夹在业务逻辑与数据库资源中间。

DAO 模式实际上是两个模式的组合即 Data Accessor (数据访问者)模式和 Active Domain Object(领域对象)模式。

Data Accessor 模式实现了数据访问和业务逻辑的分离

Active Domain Object 模式实现了业务数据的对象化封装

DAO 模式通过对业务层提供数据抽象层接口，实现了以下目标：

- **1. 数据存储逻辑的分离**

通过对数据访问逻辑进行抽象，为上层机构提供抽象化的数据访问接口。业务层无需关心具体的 select,insert,update 操作，这样，一方面避免了业务代码中混杂 JDBC 调用语句，使得业务落实实现更加清晰，另一方面，由于数据访问接口与数据访问实现分离，也使得开发人员的专业划分成为可能。某些精通数据库操作技术的开发人员可以根据接口提供数据库访问的最优化实现，而精通业务的开发人员则可以抛开数据曾德繁琐细节，专注于业务逻辑编码。

- **2. 数据访问底层实现的分离**

DAO 模式通过将数据访问计划分为抽象层和实现层，从而分离了数据使用和数据访问的底层实现细节。这意味着业务层与数据访问的底层细节无关，也就是说，我们可以在保持上层机构不变得情况下，通过切换底层实现来修改数据访问的具体机制，常见的一个例子就是，我们可以通过仅仅替换数据访问层实现，将我们的系统部署在不同的数据库平台之上。

- **3. 资源管理和调度的分离**

在数据库操作中，资源的管理和调度是一个非常值得关注的主题。大多数系统的性能瓶颈往往并非集中于业务逻辑处理本身。在系统涉及的各种资源调度过程中，往往存在着最大的性能黑洞，而数据库作为业务系统中最重要的系统资源，自然也成为关注的焦点。DAO 模式将数据访问逻辑从业务逻辑中脱离开来，使得在数据访问层实现统一的资源调度成为可能，通过数据库连接池以及各种缓存机制(Statement Cache,Data Cache 等，缓存的使用是高性能系统实现的一个关键所在)的配合使用，往往可以保持上层系统不变的情况下，大幅度提升系统性能。

- **4. 数据抽象**

在直接基于 JDBC 调用的代码中，程序员面对的数据往往是原始的 RecordSet 数据集，诚然这样的数据集可以提供足够的信息，但对于业务逻辑开发过程而言，如此琐碎和缺乏寓意的字段型数据实在令人厌倦。

DAO 模式通过对底层数据的封装，为业务曾提供一个面向对象的接口，使得业务逻辑开发员可以面向业务中的实体进行编码。通过引入 DAO 模式，业务逻辑更加清晰，且富于形象性和描述性，这将为日后的维护带来极大的便利。试想，在业务曾通过 Customer.getName 方法获得客户姓名，相对于直接通过 SQL 语句访问数据库表并从 ResultSet 中获得某个字符型字段而言，哪种方式更加易于业务逻辑的形象化和简洁化?

下面让我们来看看代码：

```
//首先，我们这个计算打折后金额的业务过程中，涉及了两个业务对象，即客户对象 Customer，和促销规则对象 Promotion。自然，这两个对象也就成为了此业务领域(Business Domain)中的 Domain Object，所谓 Domain Object，简单来讲就是对领域内(Domain)涉及的各个数据对象，反映到代码，就是一个拥有相关属性的 getter,setter 方法的 JavaClass(Java Bean)

Public BigDecimal calcAmount(String customerID,BigDecimal amount){
        //根据客户 ID 获得客户记录
        Customer customer    = CustomerDAO.getCustomer(customerID);
```

```
        //根据客户登记获得打折规则
        Promotion promotion = PromotionDAO.getPromotion(customer.getLevel());

        //累积客户总消费额，并保存累计结果
        Customer.setSumAmount(customer.getSumAmount().add(amount));
        CustomerDAO.save(customer);

        //返回打折后金额
        Return    amount.multiply(promotion.getRatio());
}

//这样的代码相信已经足够明晰，即使对于缺乏数据库技术基础的读者也可以轻松阅读
```

从上面这段代码中，我们可以看到，通过 DAO 模式对各个数据库对象进行封装，我们对业务层屏蔽了数据库访问的底层实现，业务曾仅包含与本领域相关的逻辑对象和算法，这样对于业务逻辑开发人员(以及日后专注于业务逻辑的代码阅读者)而言，面对的是一个简洁明快的逻辑实现结构。业务层的开发和维护将变得更加简单。
DAO 模式中, 数据库访问层实现被隐藏到 Data Accessor 中, 前面说过, DAO 模式实际上是两个模式的组合, 即 Data Accessor 和 Domain Object 模式。
何谓 Data Accessor?即将数据访问的实现机制加以封装，与数据的使用代码相分离，从外部来看，Data Accessor 提供了黑盒式的数据存取接口。

DAO 模式是标准的 J2EE 设计模式之一。开发人员使用这个模式把底层的数据访问操作和上层的商务逻辑分开。
一个典型的 DAO 实现有下列几个组件:

- 一个 DAO 工厂类;
- 一个 DAO 接口;
- 一个实现 DAO 接口的具体类;
- 数据传递对象(DTO)，有些时候叫做值对象(VO).

问：如何使用 DAO 工厂模式?

4、总结

1、 理解 JDBC 的基本概念及分类
2、 使用 JDBC 进行数据库表的操作
3、 熟练掌握 Statement、PreparedStatement、ResultSet 接口的使用
4、 事务处理的概念及应用
5、 熟练掌握 DAO

5、作业

1、 熟练掌握本节所有内容，反复敲代码理解
2、 使用 MySQL 数据库创建表 Product（产品）包含字段：商品编号，商品名称，商品单价，商品类型，描述。

使用 JDBC 实现对该表的操作：

1>添加记录（主键自动添加）

2>修改记录（可以修改除主键以外的字段）

3>删除记录（根据商品编号删除）

4>查询记录（可根据商品编号查询，也可根据商品名称模糊查询）

（以上功能实现要求使用 DAO 模式封装，连接工具类从配置文件读取数据库连接信息。并理解 DAO 模式的概念）