Chapter_02

> **Content:** Mysql、JDBC与DBUtils
>
> **Author:** DancingHorse
>
> **Duration:** 2天

课程目录	本节属性	本节目标
1. 小项目的数据设计	概念	了解 熟悉小项目的数据库设计
2. MySQL基础练习	练习	编写 能够完成业务需求的SQL语句
3. JDBC概述	概念	了解 JDBC的相关概念
4. 获取数据库链接	实操	通过 JDBC技术链接数据库
5. 基本的数据操作	实操	实现并掌握Java代码的CRUD
6. 面向对象的方式实现	实操	了解面向对象的编程思想
7. 模拟用户登录操作	实操	通过JDBC模拟登陆验证
8. SQL注入与预处理	概念	掌握JDBC的预处理操作
9. 模拟添加试题的业务操作	实操	模拟管理员的添加操作
10. JDBC的事务管理	概念	了解事务的相关概念
11. DBUtils工具概述	概念	了解DBUtils的相关概念
12. 简单使用DBUtils	实操	了解DBUtils的基本使用

课程重点难点[难点用高亮标识]
1. 编写能够完成业务需求的SQL语句
2. 了解 JDBC的相关概念
3. 了解面向对象的编程思想
4. 了解事务的相关概念

1. 小项目的数据设计

前言

> 本节简单介绍了项目讲解阶段小项目的数据库结构及设计
>
> 本节学习目标:
>
> 1. 了解 熟悉小项目的数据库设计

本节目录
1.1 表设计原则规范
1.2 小项目的数据结构

1.1 表设计原则规范

1.1.1 命名规范

- 库名、表名、字段名：使用小写字母，禁止数字开头，下划线风格，禁止复数名词；
- 命名中不允许出现MYSQL数据库中的保留字；

1.1.2 表设计规范

- 单表列数目建议小于30，列名不宜超过12个字符；
- 不建议使用外键，如果有外键完整性约束，交由应用程序控制；

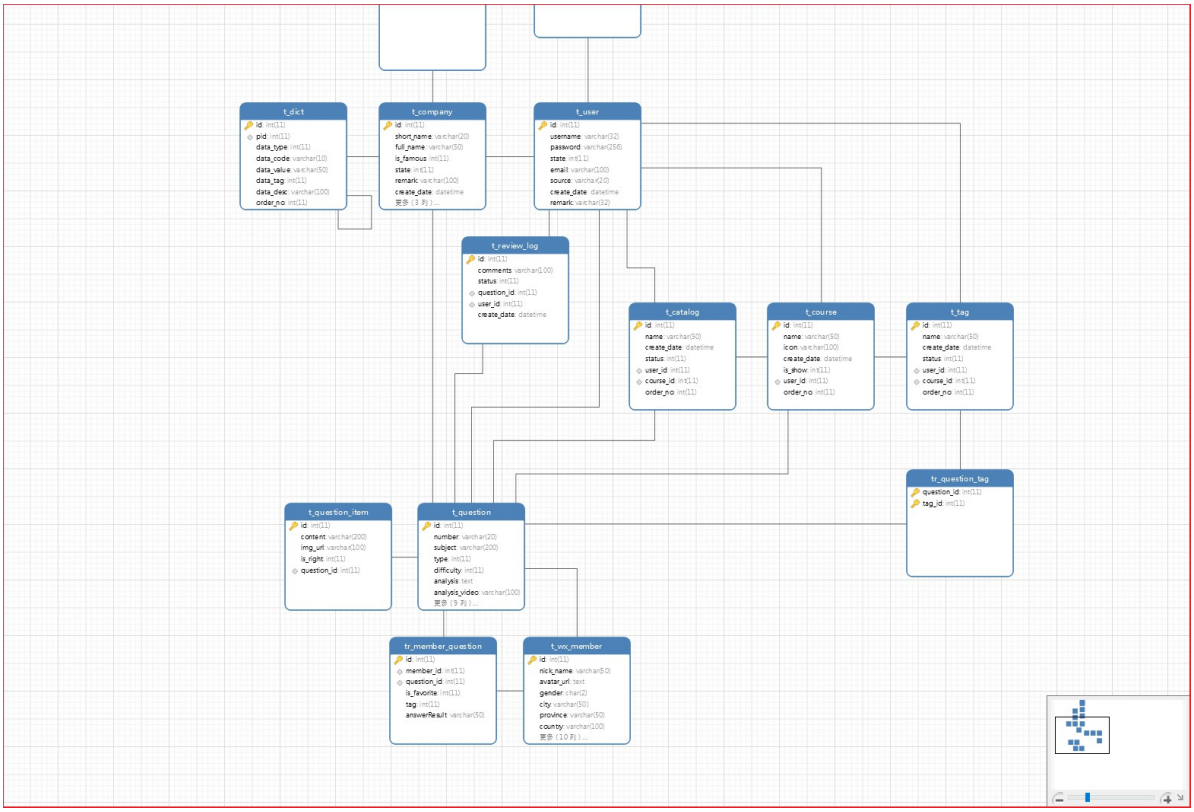
1.1.3 字段设计规范

- 把字段定义为NOT NULL并且提供默认值
- 小数类型用decimal，禁止使用float和double
- 建议使用varchar(20)存储手机号

1.1.4 SQL使用规范

- 遵循单SQL最少功能原则，尽可能交由应用程序控制；
- 建议按需查询字段，不建议查询全部字段（SELECT * ）
- 避免嵌套子查询
- 降低数据库交互次数
- 谨慎使用负面查询
- IN 语句一次查询集合不宜太大

1.3 小项目的数据结构



小结

> 1. 了解 熟悉小项目的数据库设计

2. MySQL基础练习

前言

> 本节目标：编写 能够完成业务需求的SQL语句

2.1 业务需求

学科 - 新增一条学科信息
学科 - 修改新增的学科信息
学科 - 删除该条学科信息
学科- 获取学科的分页列表
学科 - 获取指定学科的信息【学科名称，创建时间，创建者，下属的学科目录，下属的学科标签】
学科 - 获取下属学科试题数量最多的学科名称及数量。
学科 - 获取指定学科的统计信息【学科名称，下属学科目录数量，下属学科标签数量，下属学科试题数量【分别获取不同试题类型的数量】】
试题 - 获取指定试题的展示信息【试题编号，创建时间，创建人，题目来源，题目类型，题目难度，题干，答案与选项，所属学科，所属学科的目录，试题的标签合集】
试题 - 随机获取十条单选题的信息【试题编号，题干，选项，正确答案】
地址 - 获取所有地址【编号，省份【编号，城市】】 层级关系

小结

> 1. 编写能够完成业务需求的SQL语句

3. JDBC概述

前言

> 本节简单介绍了JDBC的相关概念
>
> 本节学习目标：了解JDBC的相关概念
>
> 1. 能够说明 JDBC时什么
> 2. 能够说明 为什么使用JDBC
> 3. 了解JDBC的基本组件

本节目录
3.1 什么是JDBC
3.2 为什么用JDBC
3.3 JDBC的架构
3.4 JDBC定义的组件

3.1 什么是JDBC

JDBC(Java Database Connectivity)是一个独立于特定数据库管理系统、通用的SQL数据库存取和操作的公共接口（一组API），定义了用来访问数据库的标准Java类库，使用这个类库可以以一种标准的方法、方便地访问数据库资源。 JDBC 技术是在 Java 语言中被广泛使用 的一种操作数据库的技术。

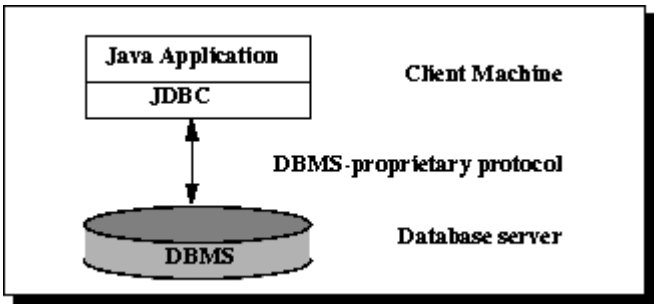
3.2 为什么用JDBC

JDBC 技术是连接数据库与应用程序的纽带，jDBC的目标是使Java程序员使用JDBC可以连接任何提供了JDBC驱动程序的数据库系统。

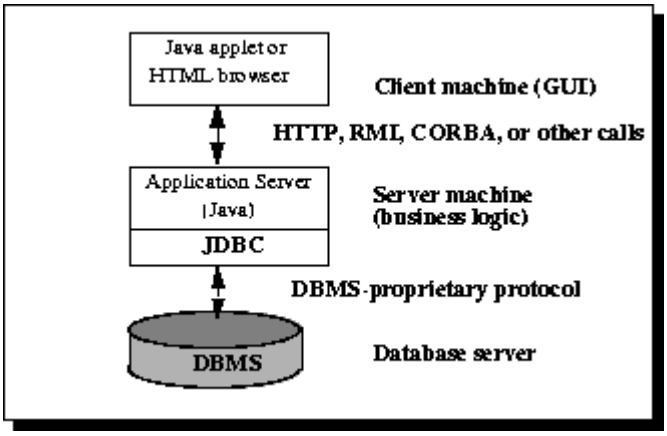
3.3 JDBC的架构

JDBC分为双层架构和三层架构。

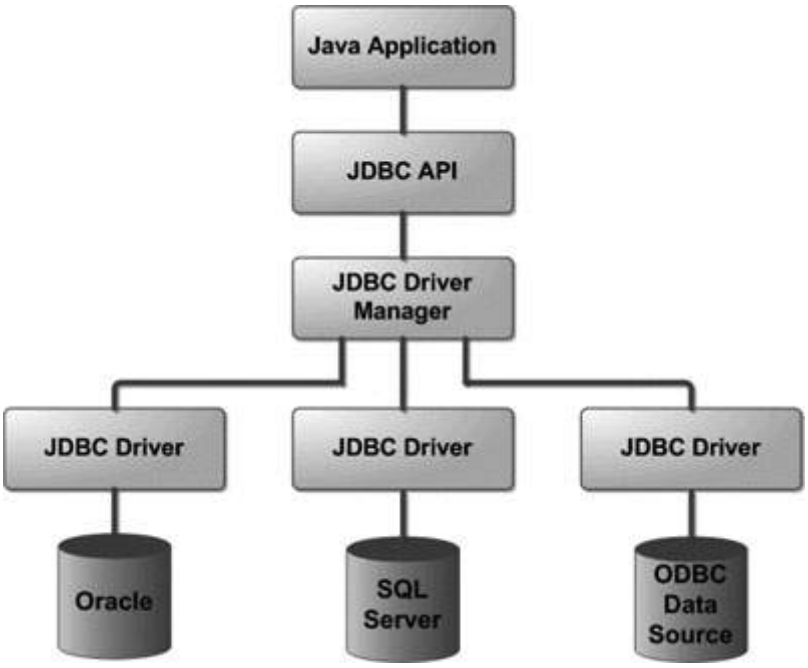
双层架构示例图



三层架构示例图



三层架构示意图2



3.4 JDBC定义的组件

- **DriverManager**: 这个类管理数据库驱动程序的列表。确定内容是否符合从Java应用程序使用的通信子协议正确的数据库驱动程序的连接请求。识别JDBC在一定子协议的第一个驱动器将被用来建立数据库连接。
- **Driver**: 此接口处理与数据库服务器通信。很少直接直接使用驱动程序（Driver）对象，一般使用`DriverManager`中的对象，它用于管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息
- **Connection** : 此接口与接触数据库的所有方法。连接对象表示通信上下文，与数据库中的所有的通信是通过此唯一的连接对象。
- **Statement** : 可以使用这个接口创建的对象的SQL语句提交到数据库。
- **ResultSet**: 这些对象保存从数据库后，执行使用`Statement`对象的SQL查询中检索数据。它作为一个迭代器，可以通过移动它来检索下一个数据。
- **SQLException**: 这个类用于处理发生在数据库应用程序中的任何错误。

小结

> 1. 能够说明 JDBC时什么
> 2. 能够说明 为什么使用JDBC
> 3. 了解JDBC的基本组件

4. 获取数据库链接

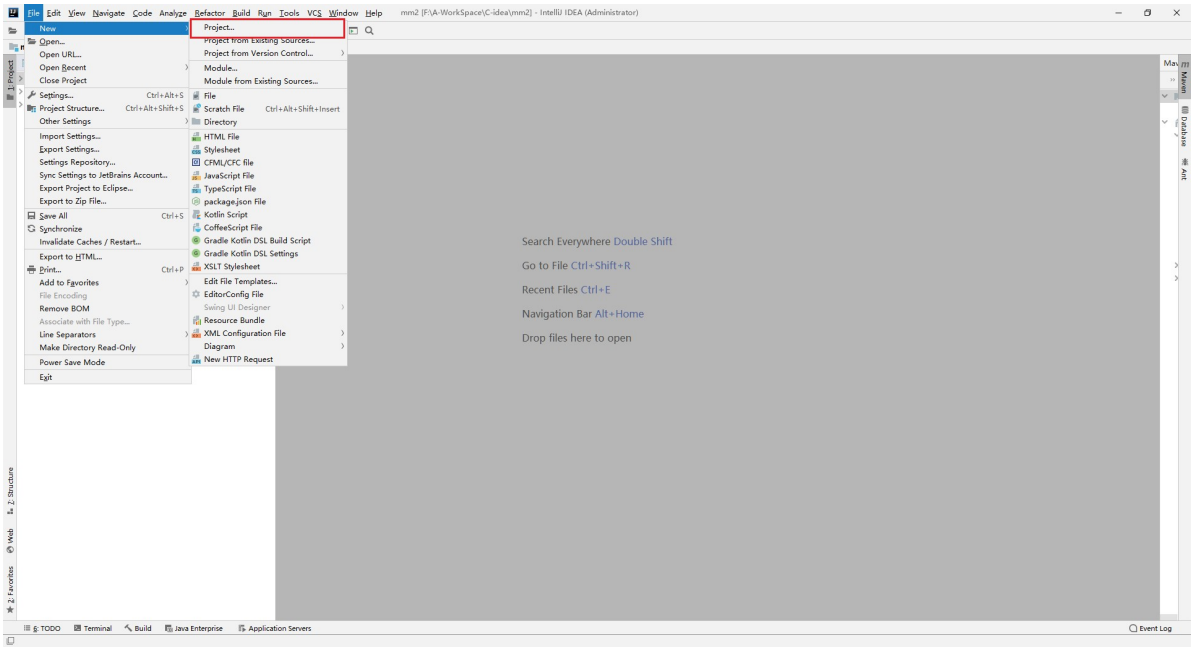
前言

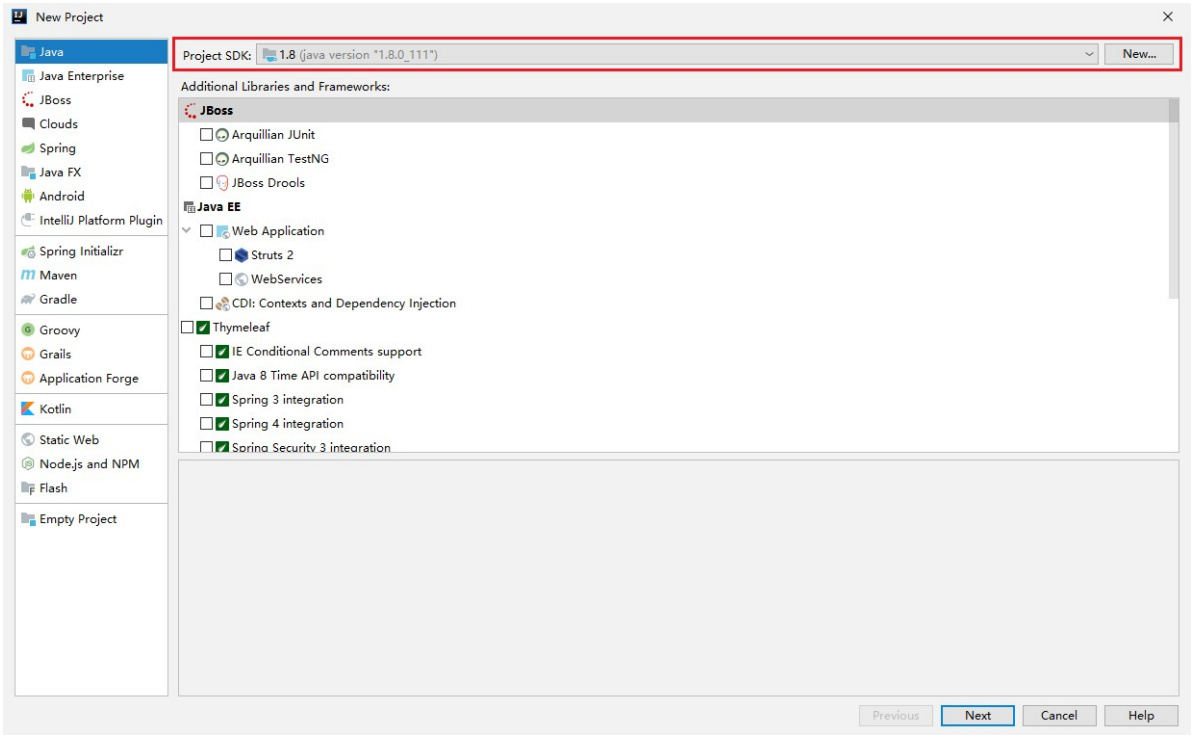
> 本节简单介绍了如何通过JDBC获取数据库的连接
>
> 本节学习目标：通过JDBC技术获取数据库的连接
>
> 1. 完成JDBC连接数据库的操作

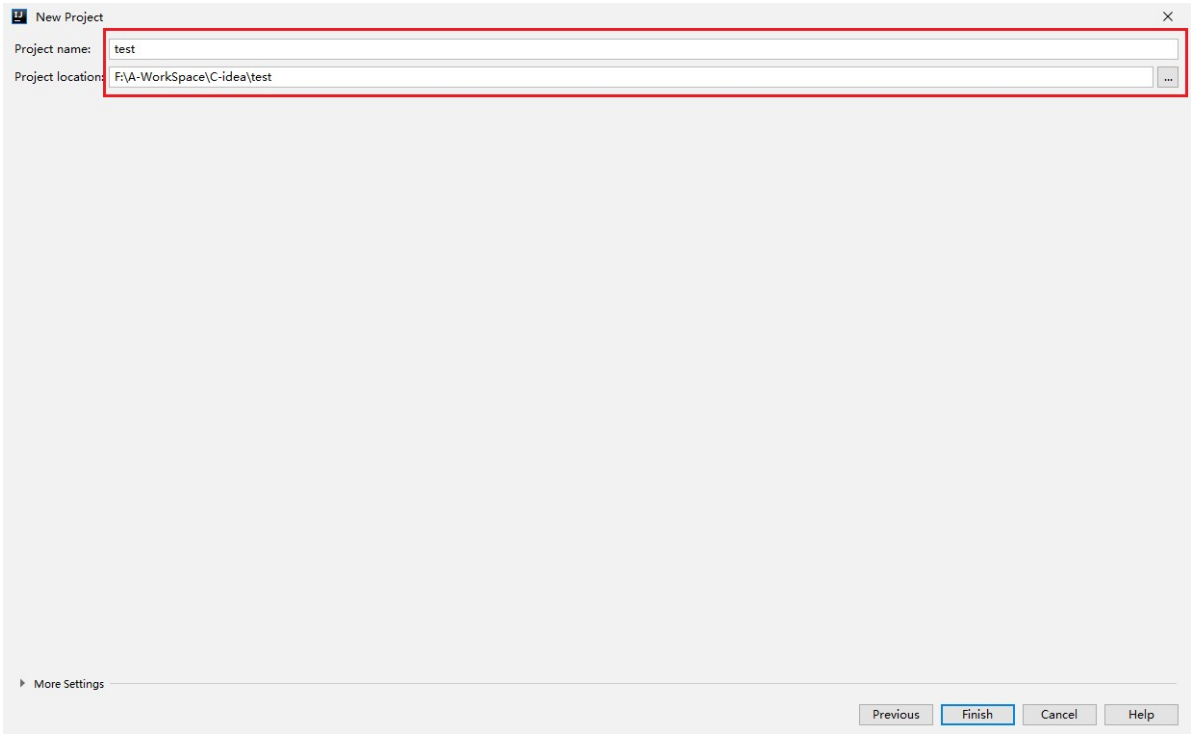
本节目录
4.1 前期准备阶段
4.2 代码实现阶段

4.1 前期准备阶段

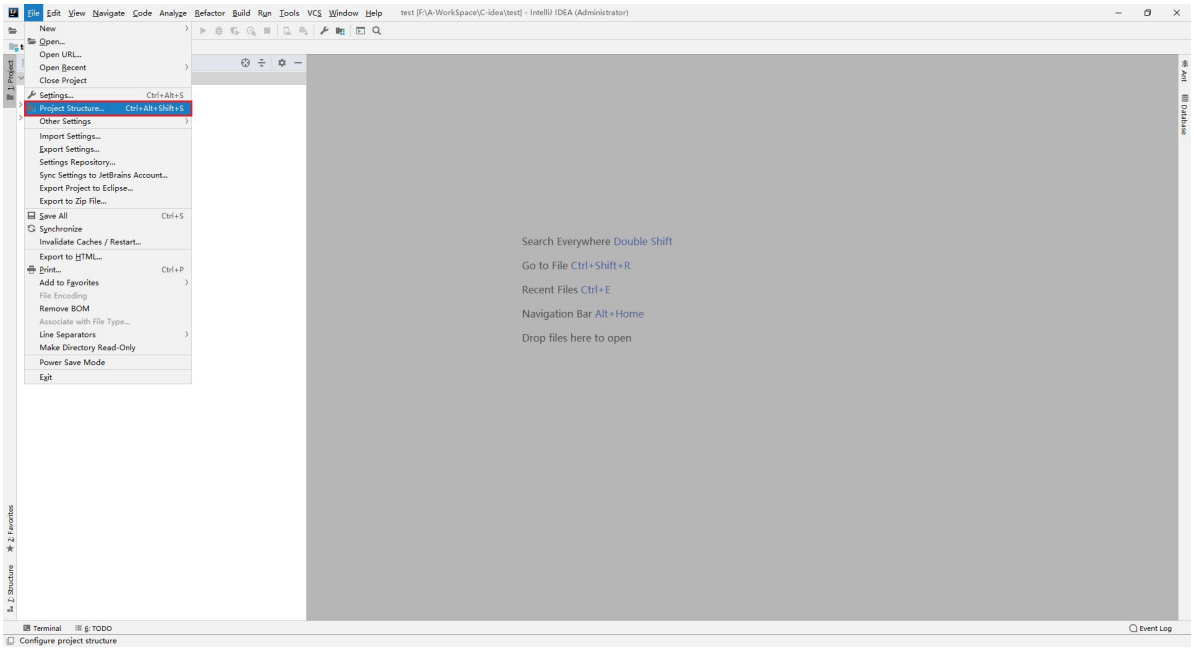
- 新建Java项目

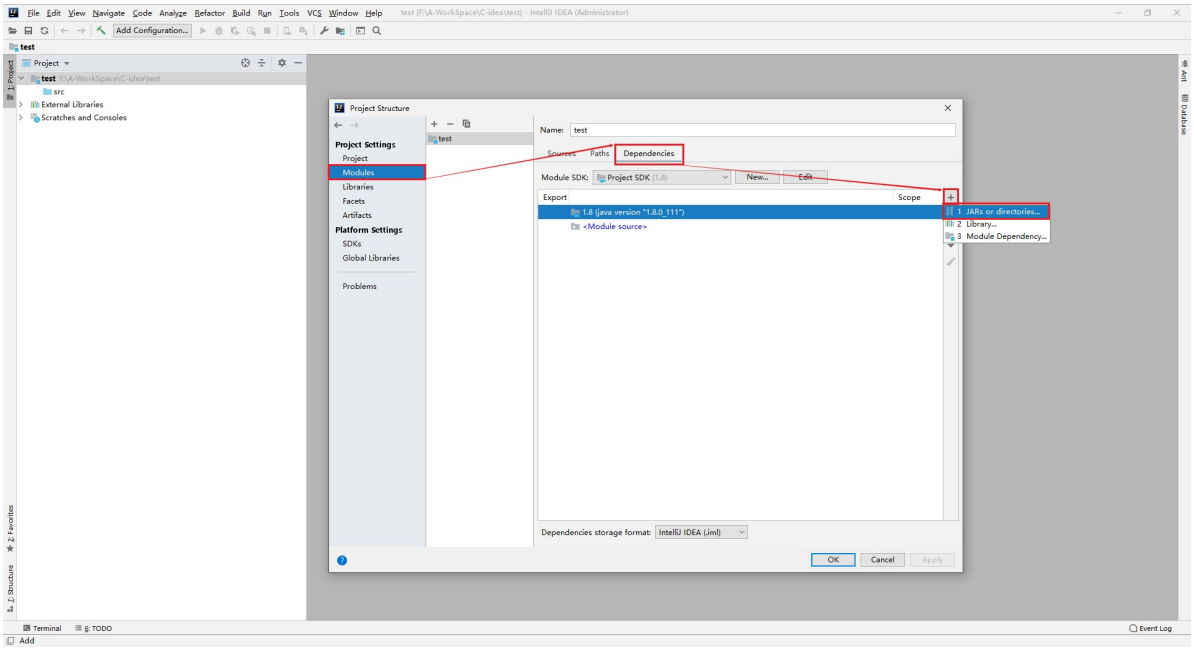

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
New
Project...
Open...
Open URL...
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
Other Settings
Import Settings...
Export Settings...
Settings Repository...
Sync Settings to JetBrains Account...
Export Project to Eclipse...
Export to Zip File...
Save All Ctrl+S
Synchronize
Invalidate Caches / Restart...
Export to HTML...
Print... Ctrl+P
Add to Favorites
File Encoding
Remove BOM
Associate with File Type...
Line Separators
Make Directory Read-Only
Power Save Mode
Exit
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
HTML File
Stylesheet
CFML/CFC file
JavaScript File
TypeScript File
package.json File
Kotlin Script
CoffeeScript File
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
EditorConfig File
Swing UI Designer
Resource Bundle
XML Configuration File
Diagram
New HTTP Request
Search Everywhere Double Shift
Go to File Ctrl+Shift+R
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open
TODO
Terminal
Build
Java Enterprise
Application Servers
Event Log

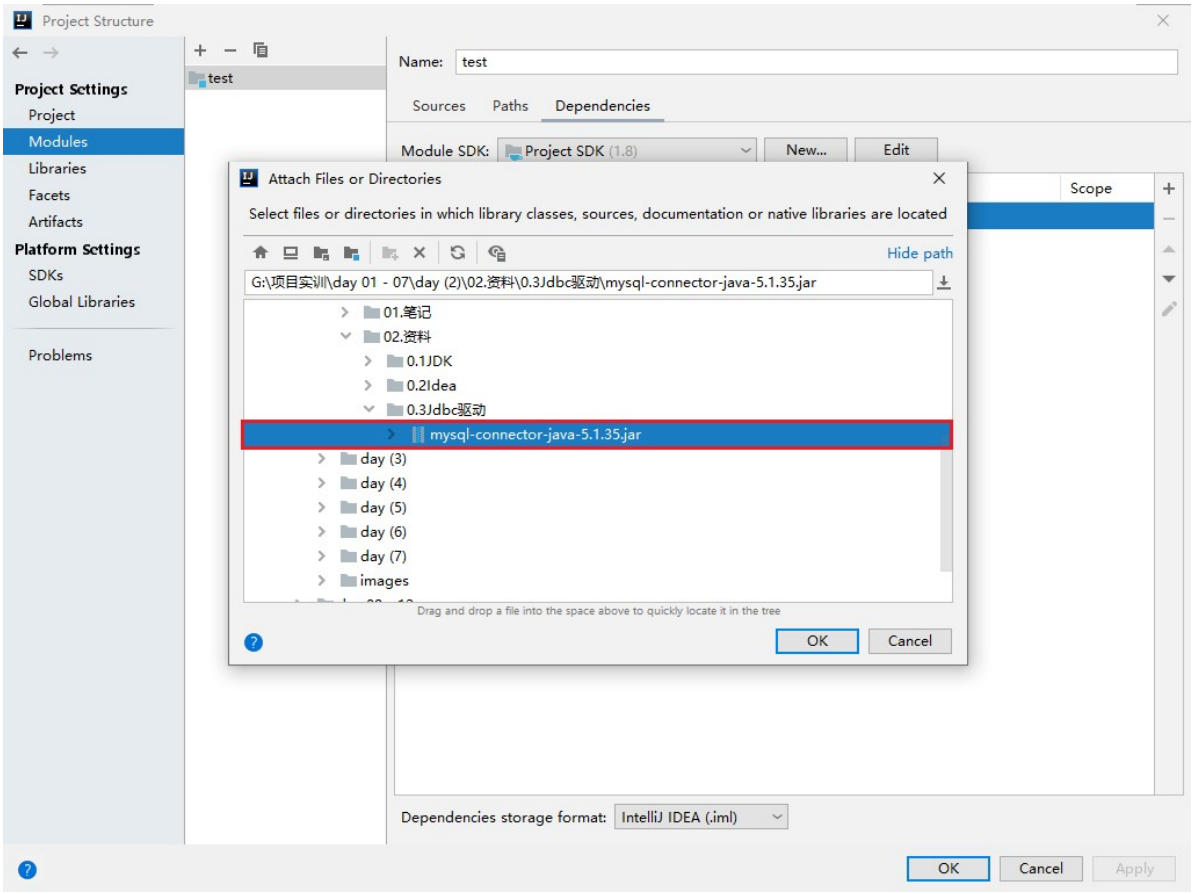

New Project
Java
Java Enterprise
JBoss
Clouds
Spring
Java FX
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle
Groovy
Grails
Application Forge
Kotlin
Static Web
Node.js and NPM
Flash
Empty Project
Project SDK: 1.8 (java version "1.8.0_111")
New...
Additional Libraries and Frameworks:
JBoss
Arquillian JUnit
Arquillian TestNG
JBoss Drools
Java EE
Web Application
Struts 2
WebServices
CDI: Contexts and Dependency Injection
Thymeleaf
IE Conditional Comments support
Java 8 Time API compatibility
Spring 3 integration
Spring 4 integration
Spring Security 3 integration
Previous
Next
Cancel
Help

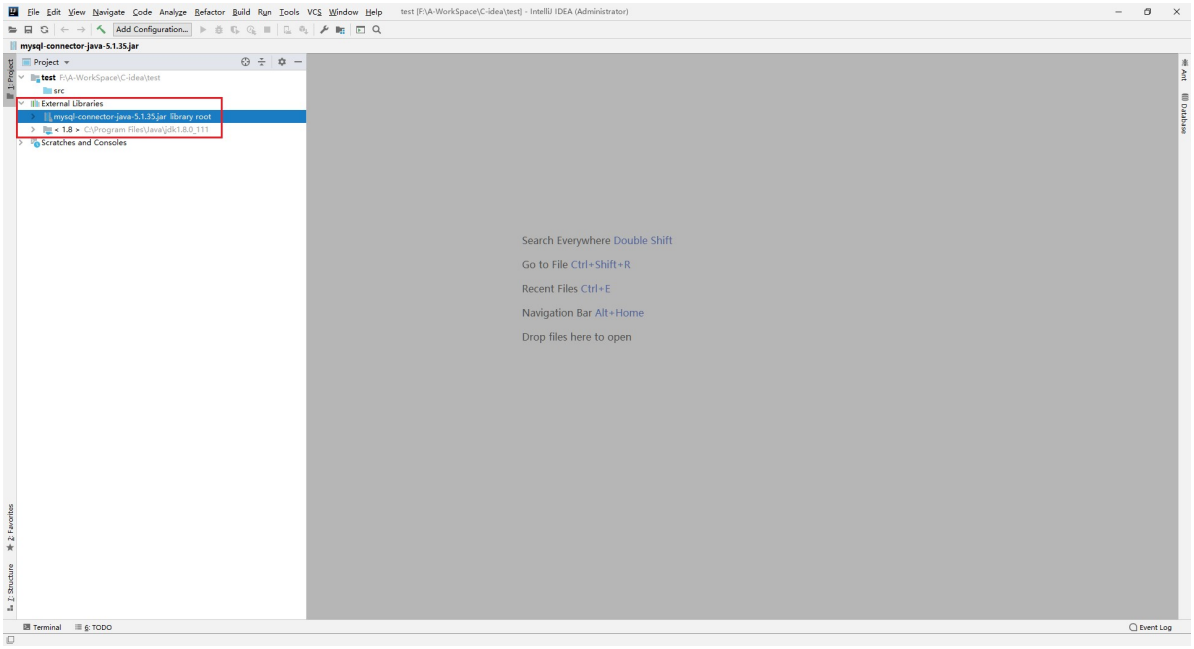


- 加入Jdbc驱动包 mysql-connector-java.jar




Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
Name: test
Sources
Paths
Dependencies
Module SDK: Project SDK (1.8)
New...
Edit
Export
Scope
1.8 (java version "1.8.0_111")
<Module source>
1 JARs or directories...
2 Library...
3 Module Dependency...
Dependencies storage format: IntelliJ IDEA (.iml)
OK
Cancel
Apply


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
test
Name: test
Sources
Paths
Dependencies
Module SDK: Project SDK (1.8)
New...
Edit
Scope
Attach Files or Directories
Select files or directories in which library classes, sources, documentation or native libraries are located
Hide path
G:\项目实训\day 01 - 07\day (2)\02.资料\0.3Jdbc驱动\mysql-connector-java-5.1.35.jar
01.笔记
02.资料
0.1JDK
0.2Idea
0.3Jdbc驱动
mysql-connector-java-5.1.35.jar
day (3)
day (4)
day (5)
day (6)
day (7)
images
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel
Dependencies storage format: IntelliJ IDEA (.iml)
OK
Cancel
Apply



4.2 代码实现阶段

```
public class JDBCTest {
    /* 数据库的链接地址 */
    public static final String URL = "jdbc:mysql://localhost:3306/mm_test";
    /* 登录用户名称*/
    public static final String USER = "root";
    /* 登录用户密码*/
    public static final String PASSWORD = "123456";
    /* 基本链接步骤 */
    public static void main(String[] args) throws Exception {
        /* 1.加载驱动程序 */
        Class.forName("com.mysql.jdbc.Driver");
        /* 2.获得数据库连接*/
        Connection connection = DriverManager.getConnection(URL, USER,
PASSWORD);
        /* 3.将链接对象输出在控制台 检查链接是否成功*/
        System.out.println("连接:"+connection);
        /* 4.关闭相关连接*/
        connection.close();
    }
}
```

小结

> 1. 完成JDBC连接数据库的操作

5. 基本的数据操作

前言

> 本节简单介绍了
>
> 本节学习目标：实现并掌握Java代码的CRUD
>
> 1. 能够查询 数据库中的结果
> 2. 能够新增 数据内容

> 3. 能够修改 数据内容
> 4. 能够删除 数据库中的数据

本节目录

5.1 插入删除及修改操作

```
public class JDBCTest {
    /* 数据库的链接地址 */
    public static final String URL = "jdbc:mysql://localhost:3306/mm_test";
    /* 登录用户名称*/
    public static final String USER = "root";
    /* 登录用户密码*/
    public static final String PASSWORD = "123456";
    /* 基本链接步骤 */
    public static void main(String[] args) throws Exception {
        /* 1.加载驱动程序 */
        Class.forName("com.mysql.jdbc.Driver");
        /* 2.获得数据库连接*/
        Connection connection = DriverManager.getConnection(URL, USER,
PASSWORD);
        /* 3.将链接对象输出在控制台 检查链接是否成功*/
        System.out.println("连接:"+connection);
        /* 4.获取数据库操作对象*/
        Statement statement = connection.createStatement();
        /* 5.编写SQL语句*/
        /*添加*/String sql1 = "insert into t_user(id,username,password)
values(10,'ceshi','ceshi');";
        /*修改*/String sql2 = "update t_user set username ='UpCeshi' where
id=10;";
        /*删除*/String sql3 = "delete from t_user where id=10;";
        /* 6.执行sql语句*/
        statement.execute(sql1);
        /* 7.关闭相关链接*/
        statement.close();
        connection.close();
    }
}
```

5.2 查询数据操作

```
public class JDBCTest {
    /* 数据库的链接地址 */
    public static final String URL = "jdbc:mysql://localhost:3306/mm_test";
    /* 登录用户名称*/
    public static final String USER = "root";
    /* 登录用户密码*/
    public static final String PASSWORD = "123456";
    /* 基本链接步骤 */
    public static void main(String[] args) throws Exception {
        /* 1.加载驱动程序 */
```

```
        Class.forName("com.mysql.jdbc.Driver");
        /* 2.获得数据库连接*/
        Connection connection = DriverManager.getConnection(URL, USER,
PASSWORD);
        /* 3.将链接对象输出在控制台 检查链接是否成功*/
        System.out.println("连接:"+connection);
        /* 4.获取数据库操作对象*/
        Statement statement = connection.createStatement();
        /* 5.编写SQL语句*/
        /*查询*/String sql = "select * from t_user";
        /* 6.执行查询方法,并获取返回结果*/
        ResultSet resultSet = statement.executeQuery(sql);
        /* 7.循环取出每一条数据*/
        while(resultSet.next()){/* 判断吓一跳数据是否存在，不存在则停止*/
            /* 8.通过列名取出值*/
            String name = resultSet.getString("username");
            /* 9.将值输出在控制台显示*/
            System.out.println(name);
        }
        /* 7.关闭相关链接*/
        resultSet.close();
        statement.close();
        connection.close();
    }
}
```

小结

> 1. 能够查询 数据库中的结果
> 2. 能够新增 数据内容
> 3. 能够修改 数据内容
> 4. 能够删除 数据库中的数据

6. 面向对象的方式实现

前言

> 本节简单用面向对象的思想来实现对数据库的操作
>
> 本节学习目标：了解面向对象的编程思想
>
> 1. 能够清楚 面向对象的编程思想
> 2. 能过够创建接口并完成功能

本节目录
6.1 面向对象概述
6.2 创建entity实体类
6.3 创建JDBC Connection工具类
6.4 创建dao层并定义接口
6.5 实现接口并完成功能

6.1 面向对象概述

面向对象(Object Oriented)是软件开发方法。面向对象是**相对于**面向过程来讲的，面向对象方法，把相关的数据和方法组织为一个**整体**来看待，从更高的层次来进行系统**建模**，更贴近事物的自然运行模式。

对象的含义是指具体的某一个事物，即在现实生活中能够看得见摸得着的事物。在面向对象程序设计中，对象所指的是计算机系统中的某一个成分。在面向对象程序设计中，对象包含两个含义，其中一个是数据，另外一个是动作。**对象则是数据和动作的结合体。**对象不仅能够进行操作，同时还能够及时记录下操作结果。

6.2 创建entity实体类

实体类一般对应一个数据表，其中的属性对应数据表中的字段。

- 对对象实体的封装，体现OO思想。
- 属性可以对字段定义和状态进行判断和过滤。
- 把相关信息用一个实体类封装后，可以作为参数传递，更加方便。

```
import java.util.Date;

public class User {
    private int id;
    private String username;
    private String password;
    private int state;
    private String email;
    private String source;
    private Date create_date;
    private String remark;

    public User() {
    }

    public User(int id, String username, String password, int state, String
email, String source, Date create_date, String remark) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.state = state;
        this.email = email;
        this.source = source;
        this.create_date = create_date;
        this.remark = remark;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }
```

```
    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getState() {
        return state;
    }

    public void setState(int state) {
        this.state = state;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getSource() {
        return source;
    }

    public void setSource(String source) {
        this.source = source;
    }

    public Date getCreate_date() {
        return create_date;
    }

    public void setCreate_date(Date create_date) {
        this.create_date = create_date;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
```

```
                ", password='" + password + '\'' +
                ", state=" + state +
                ", email='" + email + '\'' +
                ", source='" + source + '\'' +
                ", create_date=" + create_date +
                ", remark='" + remark + '\'' +
                '}';
    }
}
```

6.3 创建JDBCConnection工具类

```
//例:DU工具类
public class JDBCConnection {

    /* 1.数据库的链接地址 */
    private static final String URL = "jdbc:mysql://localhost:3306/mm_all";
    /* 登录用户名称*/
    private static final String USERNAME = "root";
    /* 登录用户密码*/
    private static final String PASSWORD = "123456";
    /* 连接对象*/
    private static Connection connection;

    /* 1.获取连接*/
    public static Connection getConnection(){
        try {
            //1加载驱动
            Class.forName("com.mysql.jdbc.Driver");
            return connection = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
        } catch (Exception e) {//捕捉所有的异常
            e.printStackTrace();
        }
        return null;
    }
    /*2.关闭，释放资源的方法close 栈式关闭（最先连接，最后关闭连接） */
    public static void close(ResultSet rs, Statement stmt, Connection
connection){
        try{//关闭结果集
            if(rs!=null) rs.close();
        }catch (SQLException e){
            e.printStackTrace();
        }
        try{//关闭sql语句
            if(stmt!=null) stmt.close();
        }catch (SQLException e){
            e.printStackTrace();
        }
        try{//关闭连接
            if(connection!=null) connection.close();
        }catch (SQLException e){
            e.printStackTrace();
        }
    }

}
```

6.4 创建dao层并定义接口

- 什么是dao
  - DAO(Data Access Object)是一个数据访问接口，数据访问：顾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。
  - 在核心J2EE模式中是这样介绍DAO模式的：为了建立一个健壮的J2EE应用，应该将所有对数据源的访问操作抽象封装在一个公共API中。用程序设计的语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口在逻辑上对应这个特定的数据存储。
- 为什么要定义接口
  - 定义接口有利于代码的规范：对于一个大型项目而言，架构师往往会对一些主要的接口来进行定义，或者清理一些没有必要的接口。这样做的目的一方面是为了给开发人员一个清晰的指示，告诉他们哪些业务需要实现；同时也能防止由于开发人员随意命名而导致的命名不清晰和代码混乱，影响开发效率。
  - 有利于对代码进行维护：比如你要做一个画板程序，其中里面有一个面板类，主要负责绘画功能，然后你就这样定义了这个类。可是在不久将来，你突然发现现有的类已经不能够满足需要，然后你又要重新设计这个类，更糟糕是你可能要放弃这个类，那么其他地方可能有引用他，这样修改起来很麻烦。如果你一开始定义一个接口，把绘制功能放在接口里，然后定义类时实现这个接口，然后你只要用这个接口去引用实现它的类就行了，以后要换的话只不过是引用另一个类而已，这样就达到维护、拓展的方便性。
  - 保证代码的安全和严密：一个好的程序一定符合高内聚低耦合的特征，那么实现低耦合，定义接口是一个很好的方法，能够让系统的功能较好地实现，而不涉及任何具体的实现细节。这样就比较安全、严密一些，这一思想一般在软件开发中较为常见。

```
/**
 * 基本数据操作
 */
public interface UserDao {
    /*获取所有用户信息*/
    List<User> selectAll() throws SQLException;
    /*获取指定ID的学生信息*/
    public User selectByName(String userName) throws SQLException;
    /*添加一个用户*/
    void insert(User user)throws SQLException;
    /*修改用户信息*/
    void update(User user)throws SQLException;
    /*添加一个用户*/
    void delete(User user)throws SQLException;
}
```

6.5 实现接口并完成功能

```
import java.util.Date;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
```

```java
public class UserDaoImpl implements UserDao {

    private Connection connection;
    private Statement statement;

    @Override
    public List<User> selectAll() throws SQLException {
        List<User> list = new ArrayList<>();
        connection = JDBCConnection.getConnection();
        statement = connection.createStatement();
        String sql = "select * from t_user";
        ResultSet resultSet = statement.executeQuery(sql);
        while(resultSet.next()){
            User user = new User();
            user.setId(resultSet.getInt("id"));
            user.setUsername(resultSet.getString("username"));
            user.setPassword(resultSet.getString("password"));
            user.setState(resultSet.getInt("state"));
            user.setEmail(resultSet.getString("email"));
            user.setSource(resultSet.getString("source"));
            user.setCreateDate(resultSet.getDate("create_date"));
            user.setRemark(resultSet.getString("remark"));
            list.add(user);
        }
        JDBCConnection.close(resultSet, statement, connection);
        return list;
    }

    @Override
    public User selectByName(String userName) throws SQLException {
        User user = new User();
        connection = JDBCConnection.getConnection();
        statement = connection.createStatement();
        String sql = "select * from t_user where username ='"+userName+"'";
        ResultSet resultSet = statement.executeQuery(sql);
        while(resultSet.next()){
            user.setId(resultSet.getInt("id"));
            user.setUsername(resultSet.getString("username"));
            user.setPassword(resultSet.getString("password"));
            user.setState(resultSet.getInt("state"));
            user.setEmail(resultSet.getString("email"));
            user.setSource(resultSet.getString("source"));
            user.setCreateDate(resultSet.getDate("create_date"));
            user.setRemark(resultSet.getString("remark"));
        }
        JDBCConnection.close(resultSet, statement, connection);
        return user;
    }

    @Override
    public void insert(User user) throws SQLException {
        connection = JDBCConnection.getConnection();
        statement = connection.createStatement();
        String sql = "insert into t_user set " +
                "username = '"+user.getUsername()+
                "',password = '"+user.getPassword()+
                "',state = "+user.getState()+
                ",email = '"+user.getEmail()+
```

```
                "',source = '"+user.getSource()+
                "',create_date = '"+ new
java.sql.Date(user.getCreateDate().getTime())+
                "',remark = '"+user.getRemark()+"'";
        statement.execute(sql);
        JDBCConnection.close(null, statement, connection);

    }

    @Override
    public void update(User user) throws SQLException {
        connection = JDBCConnection.getConnection();
        statement = connection.createStatement();
        String sql = "update  t_user set " +
                "username = '"+user.getUsername()+
                "',password = '"+user.getPassword()+
                "',state = "+user.getState()+
                ",email = '"+user.getEmail()+
                "',source = '"+user.getSource()+
                "',create_date = '"+ new
java.sql.Date(user.getCreateDate().getTime())+
                "',remark = '"+user.getRemark()+
                "' where id = "+user.getId();
        statement.execute(sql);
        JDBCConnection.close(null, statement, connection);
    }

    @Override
    public void delete(int id) throws SQLException {
        connection = JDBCConnection.getConnection();
        statement = connection.createStatement();
        String sql = "delete from t_user where  id= "+id;
        statement.execute(sql);
        JDBCConnection.close(null, statement, connection);
    }

    public static void main(String[] args) throws SQLException {
        UserDao userDao = new UserDaoImpl();
        System.out.println(":ALL");
        for (User user : userDao.selectAll()) {
            System.out.println(user);
        }
        System.out.println(":BY NAME");
        System.out.println(userDao.selectByName("admin"));
        User user = new User();
        user.setId(2);
        user.setUsername("lisi");
        user.setPassword("123456");
        user.setState(2);
        user.setEmail("111@qq.com");
        user.setSource("后台");
        user.setCreateDate(new Date());
        user.setRemark("测试数据");
        System.out.println(user.getCreateDate().getTime());
        userDao.delete(3);
    }
}
```

小结

> 1. 掌握