

Java语言程序设计

配套教材由清华大学出版社出版发行

第10章 数据库编程

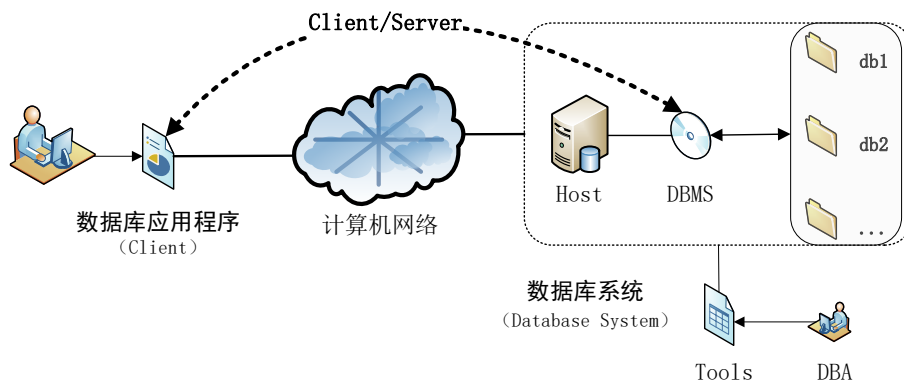


中國農業大學

阚道宏

第10章 数据库编程

- **数据库应用系统**是应用软件开发过程中最为常见的一种系统



— 数据库系统

- 文件、数据库系统 (DataBase System, 简称DBS)
- 数据模型、数据库 (database)
- 关系模型 (relational model)、关系型数据库 (relational database)

— 数据库应用程序

- 增查改删 (Create、Read、Update或Delete, 简称CRUD)
- 数据库编程



第10章 数据库编程

- 本章内容
 - [10.1 数据库系统的基本原理](#)
 - [10.2 JDBC数据库编程代码框架](#)
 - [10.3 JDBC数据库编程实验](#)
 - [10.4 开启自己的Java探索之旅](#)



10.1 数据库系统的基本原理

- 数据库系统的基本组成

- 主机Host

- 数据库管理员DBA

- 用户
- 程序员
- 数据库设计人员

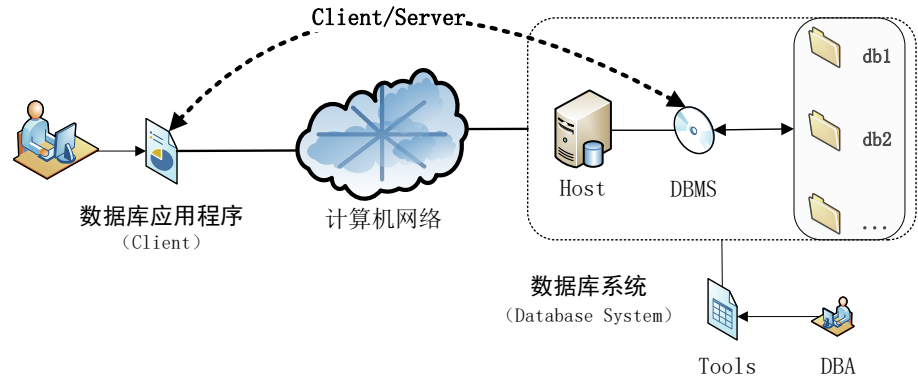
- 数据库管理系统DBMS

- 管理工具软件

- 数据库应用程序

- 数据库db

- 管理工具Tools



中國農業大學

閻道宏

10.1 数据库系统的基本原理

- 数据库系统的基本组成

- 数据库系统的特点

- 数据**冗余**少。数据库系统通过集中管理、网络共享、范式设计和重复消除等手段，尽可能地减少数据冗余
 - 数据**正确可靠**。数据库系统通过制定完整性约束条件对数据进行检查。只有满足约束条件的数据才能存入数据库，这样可以保证所有存放在数据库中的数据都是正确、可靠的
 - 具有授权**访问控制**。数据库系统具有完善的访问控制机制，这样可以保证数据安全，防止信息泄露。访问数据库系统，只有被授权的用户才能访问。通过权限设置，可以控制用户只能访问数据库中的部分数据，即只能访问应当访问的数据
 - 提供**数据库访问服务**。数据库系统以网络服务的形式对外提供数据库访问服务。其客户端程序有两类，一类是DBMS自身提供的管理工具软件，另一类是由程序员开发的数据库应用程序



10.1 数据库系统的基本原理

- **关系型**数据库

- 关系型数据库是按照关系模型来描述和存储客观世界中**实体**（entity）以及**实体间关系**（relationship）的数据库系统

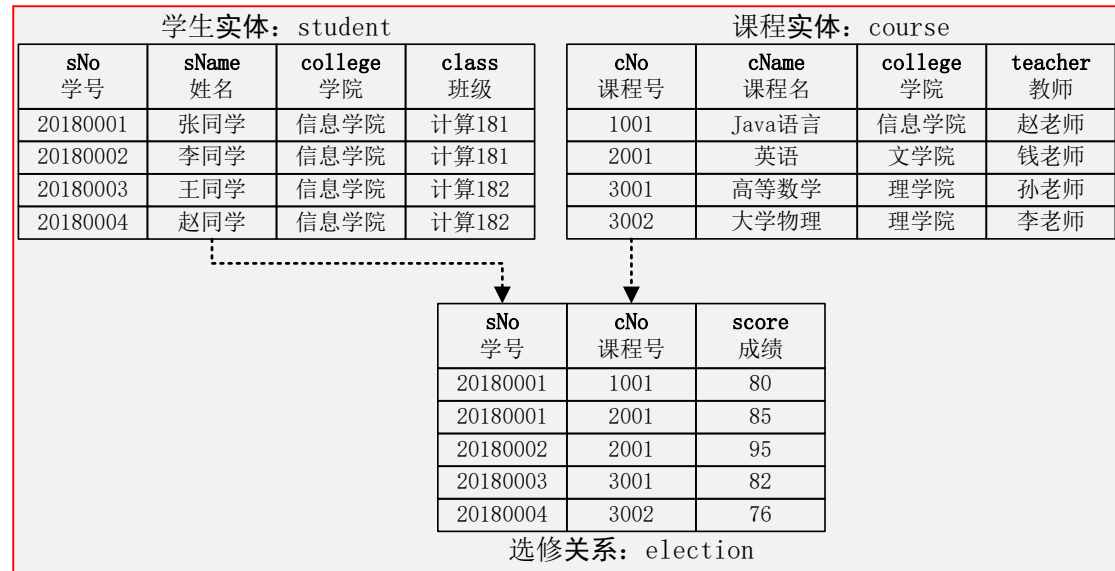
- 关系模型

- 由**行**（row）和**列**（column）组成的二维表格
 - 表格的每一行被称为一条**记录**（record）
 - 一条记录包含多个数据项，每个数据项被称为一个**字段**（field）
 - 同一表格中的记录都具有相同的字段结构。表格中有一个标明字段名称（即列的名称）的表头，它被称作是二维表格的**表结构**（table structure）



10.1 数据库系统的基本原理

- 关系型数据库
 - 关系模型



- 描述和存储**实体**的表格
 - 表 (table)，或数据表 (data table)
 - 主键 (Primary Key, 简称PK)
- 描述和存储**实体间关系**的表格
 - 外键 (Foreign Key, 简称FK)
- 关系模型的优点
 - 易于理解，易于实现
 - 具有坚实的理论基础，E. F. Codd于1970年提出



中國農業大學

阚道宏

10.1 数据库系统的基本原理

- 关系型数据库
 - 关系型数据库就是按照关系模型建立起来的数据库。一个关系型数据库系统的核心是其中的**关系型数据库管理系统**（Rational DBMS，简称**RDBMS**）软件
 - **RDBMS** 类比 **Execl**
 - 数据库 类似于 工作簿
 - 数据表 类似于 工作表
 - 结构化查询语言**SQL**（Structured Query Language）



10.1 数据库系统的基本原理

- 关系型数据库
 - 常用的关系型数据库管理系统
 - **Oracle**，美国甲骨文公司（Oracle）开发
 - **SQL Server**，美国微软公司（Microsoft）开发
 - **MySQL**，是瑞典MySQL AB公司（由美国甲骨文公司提供支持）开发的一个开源DBMS软件
 - **Access**，是美国微软公司（Microsoft）开发的一种小型数据库管理系统
 - **Java DB**，是Apache Derby软件组织开发的一种小型数据库管理系统，目前也是由美国甲骨文公司提供支持



10.1 数据库系统的基本原理

- 结构化查询语言 **SQL**
 - 常用操作
 - 创建或删除 **数据库**
 - 在数据库中创建或删除 **数据表**
 - 在数据表中插入、修改、删除或查询 **记录**等
 - 创建或删除数据库
 - 创建数据库
CREATE DATABASE database_name
 - 删除数据库
DROP DATABASE database_name



10.1 数据库系统的基本原理

- 结构化查询语言 **SQL**

- 创建数据表

```
CREATE TABLE table_name(  
    field_name1 data_type1(size),  
    field_name2 data_type2(size),  
    field_name3 data_type3(size),  
    .....  
)
```

- 定义各字段的名称、数据类型、长度，以及是否主键等
 - SQL语言中有五大类数据类型：**字符型**（CHAR、VARCHAR），**数值型**（INT、NUMERIC），**逻辑型**（BIT），**日期型**（DATE、TIME）和**变长型**（BLOB、MEMO）

```
CREATE TABLE student(  
    sNo CHAR(10) PRIMARY KEY, sName CHAR(10),  
    college VARCHAR(20), class VARCHAR(20)  
)
```

- 删除数据表

```
DROP TABLE table_name
```



中國農業大學

閻道宏

10.1 数据库系统的基本原理

- 结构化查询语言 **SQL**

- 插入记录

INSERT INTO table_name **VALUES**(value1, value2, value3, ...)

INSERT INTO student **VALUES**('20180001', '张同学', '信电学院', '计算181')

- 查询记录

SELECT * **FROM** table_name

SELECT field_name1, field_name2, ... **FROM** table_name

SELECT field_name1, field_name2, ... **FROM** table_name
WHERE condition

SELECT * **FROM** student
WHERE sNo = '20180001'



中國農業大學

閻道宏

10.1 数据库系统的基本原理

- 结构化查询语言 **SQL**

- 修改记录

UPDATE table_name

SET field_name1 = value1, field_name2 = value2, ...

WHERE condition

UPDATE student

SET class = '计算182'

WHERE sNo = '20180001'

- 删除记录

DELETE FROM table_name

WHERE condition

DELETE FROM student

WHERE sNo = '20180001'



中國農業大學

閻道宏

10.2 JDBC数据库编程代码框架

- 编写数据库应用程序时需要考虑的问题
 - 如何访问**不同RDBMS**管理的数据库，需要为每一种RDBMS编写一个数据库应用程序吗？
 - 一个RDBMS可以管理多个数据库，数据库应用程序如何**指定**访问哪个数据库？
 - 如何**操作**数据库中的数据？数据库应用程序通过SQL语句操作数据库中的数据，该如何将SQL语句**提交**给RDBMS执行，又该如何**取得**RDBMS返回的结果呢？
- Java语言专门设计了一种关系型数据库连接规范，它被称作**JDBC**（Java DataBase Connectivity）规范
- Java API配套提供了一组基于JDBC规范的数据库类和接口，它们被统称为**JDBC API**
- 本节具体讲解**JDBC规范**、**JDBC API**以及数据库应用程序的代码框架



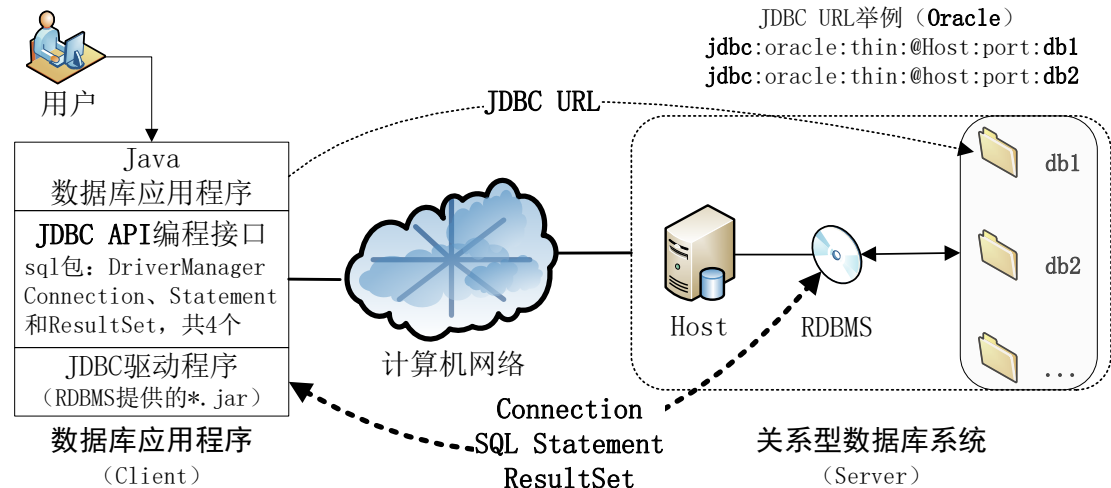
10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 关系型数据库只定义了操作数据库的SQL语言，但并没有为数据库访问服务定义统一的**应用层**协议
 - 对于关系型数据库管理系统RDBMS来说
 - JDBC就是一种对外提供数据库访问服务的“**请求-响应**”规范
 - 虽然不同厂家RDBMS软件之间的差别很大，但它们对外服务的**接口**是一样的
 - 每个RDBMS都需要提供一个专门的**JDBC驱动程序**（driver），用于向数据库应用程序提供JDBC服务接口
 - 对程序员来说
 - JDBC就是一组通用的数据库编程接口**API**，可以访问不同厂家的RDBMS
 - Java语言通过java.sql包中的**DriverManager**类和**Connection**、**Statement**、**ResultSet**等接口实现了这组API，它们被统称为**JDBC API**
 - 使用JDBC API**编写**数据库应用程序，只要**加载**不同的JDBC驱动程序，就可以**访问**不同厂家的数据库系统



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC



- 用户
- Java数据库应用程序: JDBC URL、SQL 语句 (Statement)
- JDBC驱动程序: 数据库连接 (Connection)、执行结果 (例如ResultSet)
- RDBMS



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - JDBC API的编程步骤
 - 加载JDBC驱动程序

java.lang. Class <T>类说明文档			
public final class Class <T> extends Object implements Serializable , GenericDeclaration , Type , AnnotatedElement			
	修饰符	类成员（节选）	功能说明
1	static	Class <?> forName (String className)	加载指定类名的Java类
2	static	Class <?> forName (String name, boolean initialize, ClassLoader loader)	加载指定类名的Java类
3		String getName ()	获取类的类名
4		Constructor<?>[] getConstructors ()	获取类的构造方法数组
5		Field[] getFields ()	获取类的字段数组
6		Method[] getMethods ()	获取类的方法数组
7		Annotation[] getAnnotations ()	获取类的注解数组
8		T newInstance ()	新建一个该类的对象
.....			



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - **加载**JDBC驱动程序
 - 常用**RDBMS**的JDBC驱动程序**类名**
 - Oracle**: oracle.jdbc.driver.OracleDriver
 - MySQL**: com.mysql.jdbc.Driver
 - SQL Server**: com.microsoft.sqlserver.jdbc.SQLServerDriver
 - Access**: sun.jdbc.odbc.JdbcOdbcDriver
 - Java DB**: org.apache.derby.jdbc.EmbeddedDriver
- Class.forName**("oracle.jdbc.driver.OracleDriver");
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 连接数据库
 - 统一资源定位符URL
 - JDBC URL
 - 数据库的JDBC URL
 - 主机名（或IP地址）、TCP端口号、数据库名
 - 数据库文件所在的存储路径及其文件名（或目录名）
 - 常用RDBMS的JDBC URL
 - Oracle:** jdbc:oracle:thin:@host:port:db_name
 - MySQL:** jdbc:mysql://host:port/db_name
 - SQL Server:** jdbc:microsoft:sqlserver://host:port;DatabaseName=db_name
 - Access:** jdbc:odbc:driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ="path/db_filename"
 - Java DB:** jdbc:derby:path/db_foldername; create=true



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 连接数据库
 - 驱动管理器类**DriverManager**
 - 静态方法**getConnection()**: 在应用程序和数据库之间建立连接, 并返回一个接口**Connection**的连接对象

java.sql.DriverManager类说明文档			
public class DriverManager extends Object			
	修饰符	类成员（节选）	功能说明
1	static	Connection getConnection (String url)	建立数据库连接（不需账户）
2	static	Connection getConnection (String url, String user, String password)	建立数据库连接（需要账户）
3	static	void setLoginTimeout (int seconds)	设置等待时间，超时则连接失败
.....			



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 提交SQL语句并接收返回结果
 - 修改（**update**）语句：创建或删除数据表语句，以及对数据表做插入、修改或删除等操作的SQL语句
 - 查询（**query**）语句：SQL中的SELECT语句
 - 数据库应用程序通过JDBC API向RDBMS提交SQL语句。RDBMS接收SQL语句，并按照SQL语句的指令要求执行数据库操作
 - 向RDBMS提交SQL语句需先创建一个接口**Statement**的语句对象。调用连接对象的**createStatement()**方法就可以创建语句对象
Statement s = con.**createStatement**();



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 提交SQL语句并接收返回结果
 - 提交SQL修改语句：调用语句对象的**executeUpdate()**方法
 - s.**executeUpdate**("CREATE TABLE student(...)");
 - s.**executeUpdate**("INSERT INTO student VALUES(...)");
 - 提交SQL查询语句
 - 查询结果：接口**ResultSet**的结果集对象
 - 调用语句对象的**executeQuery()**方法
 - ResultSet** rs = s.**executeQuery**("SELECT * FROM student");



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 提交SQL语句并接收返回结果
 - 接收查询结果，对所接收到的结果集对象进行遍历处理
`ResultSet rs = s.executeQuery("SELECT * FROM student");`
 - 对结果集对象进行遍历处理的代码结构

```
while ( rs.next() ) {  
    .....  
}
```
 - 在语句对象使用结束后，应当关闭语句对象，释放其所占用的资源
`s.close(); // 关闭语句对象s`



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC
 - 使用JDBC API编写数据库应用程序
 - 关闭数据库连接
`con.close();` // 关闭数据库连接con
 - 加载JDBC驱动程序
 - 连接数据库
 - 提交SQL语句并接收返回结果
 - 关闭数据库连接
 - 3个JDBC API接口
 - 连接接口 **Connection**
 - 语句接口 **Statement**
 - 结果集接口 **ResultSet**



10.2 JDBC数据库编程代码框架

- Java数据库连接规范JDBC

java.sql.Connection接口说明文档			
public interface Connection			
extends Wrapper, AutoCloseable			
	修饰符	接口成员（节选）	功能说明
1		Statement createStatement()	创建SQL语句对象
2		PreparedStatement prepareStatement (String sql)	创建SQL预处理语句对象
3		void commit()	提交事务
4		void rollback()	回滚事务
5		boolean isValid (int timeout)	检查连接是否有效
6		boolean isClosed()	检查连接是否已关闭
7		void close()	关闭数据库连接
.....			

java.sql.Statement接口说明文档			
public interface Statement			
extends Wrapper, AutoCloseable			
	修饰符	接口成员（节选）	功能说明
1		boolean execute (String sql)	将SQL语句提交给DBMS执行
2		ResultSet executeQuery (String sql)	将查询语句提交给DBMS执行
3		int executeUpdate (String sql)	将修改语句提交给DBMS执行
4		void close()	关闭语句
.....			



10.2 JDBC数据库编程代码框架

java.sql.ResultSet接口说明文档			
public interface ResultSet			
extends Wrapper, AutoCloseable			
	修饰符	接口成员（节选）	功能说明
1		int getRow()	返回当前记录的行号
2		boolean first()	移到结果集的第一行
3		boolean next()	移到结果集的下一行
4		boolean previous()	移到结果集的上一行
5		boolean last()	移到结果集的最后一行
6		boolean absolute (int row)	移到指定的行
7		int getInt (String columnLabel)	读取字段，返回整数
8		long getLong (String columnLabel)	读取字段，返回长整数
9		float getFloat (String columnLabel)	读取字段，返回单精度实数
10		double getDouble (String columnLabel)	读取字段，返回双精度实数
11		String getString (String columnLabel)	读取字段，返回字符串
12		Date getDate (String columnLabel)	读取字段，返回日期对象
13		Time getTime (String columnLabel)	读取字段，返回时间对象
14		URL getURL (String columnLabel)	读取字段，返回URL对象
15		void insertRow()	将当前行插入数据库
16		void updateInt (String columnLabel, int x)	修改字段（整数类型）
17		void updateDouble (String columnLabel, double x)	修改字段（实数类型）
18		void updateString (String columnLabel, String x)	修改字段（字符串类型）
19		void updateRow()	将当前行的数据写入数据库
20		void deleteRow()	删除当前行
21		boolean rowInserted()	检查当前行是否已被插入
22		boolean rowUpdated()	检查当前行是否已被修改
23		boolean rowDeleted()	检查当前行是否已经删除



10.3 JDBC数据库编程实验

- 编程实验分5个步骤完成
 - 搭建数据库编程**实验环境**，新建一个数据库编程实验用的**Java项目**
 - 为数据库编程实验项目导入所需的**外部JAR包**
 - 创建**数据库**和**数据表**，向数据表插入**记录**
 - **查询**数据表
 - **修改、删除**记录



10.3 JDBC数据库编程实验

- 搭建数据库编程实验环境
 - Java开发包（**JDK**）和Java集成开发环境（**IDE**）
 - 数据库系统
 - 连接网络上某个已经搭建好的数据库系统服务器
 - 在自己的计算机上搭建一个新的数据库系统
 - 使用**Java DB**在自己的计算机上搭建一个全新的数据库系统



10.3 JDBC数据库编程实验

- 搭建数据库编程实验环境
 - 按照以下**5项实验要求**搭建好自己的JDBC数据库编程实验环境
 - Java开发包。下载并安装**JDK 1.8**或更新版本
 - Java集成开发环境。下载并安装**Eclipse 4.7**或更新版本
 - 数据库系统
 - 使用JDK 1.8或更新版本自带的**Java DB**关系型数据库管理系统
 - JDK 1.8安装目录下的“**db**”子目录中
 - 新建数据库编程实验项目：**Chapter10**
 - 为数据库编程实验项目导入**外部JAR包**
 - Java DB的**JDBC**驱动程序是以JAR包文件形式提供的
 - JDK 1.8安装目录的“**db\lib**”子目录下的“**derby.jar**”



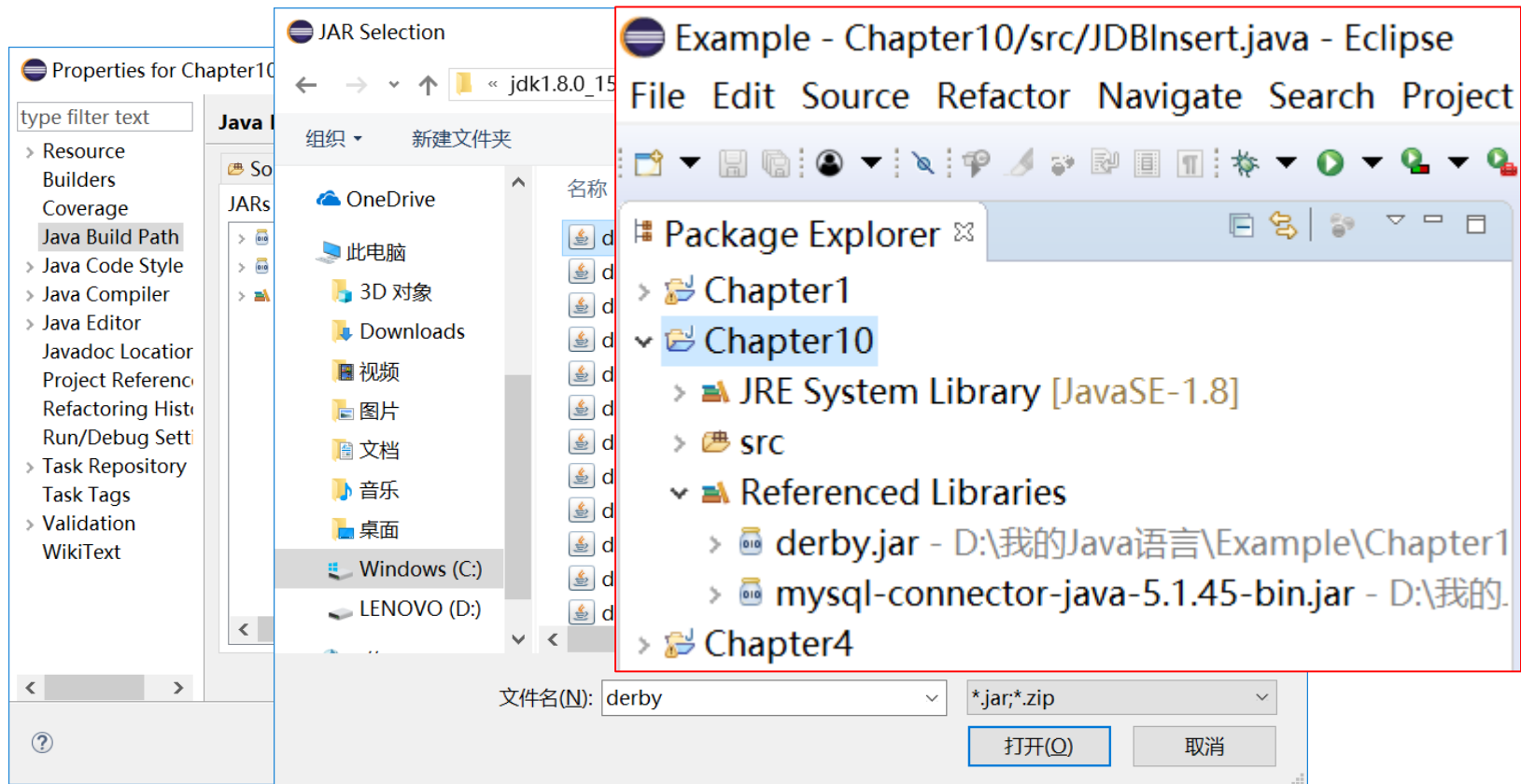
10.3 JDBC数据库编程实验

- 为Java项目导入**外部JAR包**
 - 因为外部JAR包可能安装在硬盘的任何目录下，Eclipse集成开发环境无法确定这些外部JAR包文件的**存放位置**
 - 如果一个Java项目需要用到外部JAR包，则必须为该项目做一次导入操作，将外部JAR包的安装目录添加到Java项目的组建路径（**Java Build Path**）中
 - 进入Java项目的**属性页**
菜单“**Project**” → “**Properties**”



10.3 JDBC数据库编程实验

- 为Java项目导入外部JAR包



10.3 JDBC数据库编程实验

- 创建**数据库**和**数据表**
 - 模拟数据库应用系统中的数据库**初始创建**环节
 - 创建**数据库**。在本地JavaDB数据库系统中创建一个名为“**cau**”的教务信息数据库
 - 创建**数据表**。在教务信息数据库cau中创建一个保存学生信息的数据表**student**。学生表student的表结构包含学号sNo、姓名sName、学院college和班级class等4个字段

学生实体: student

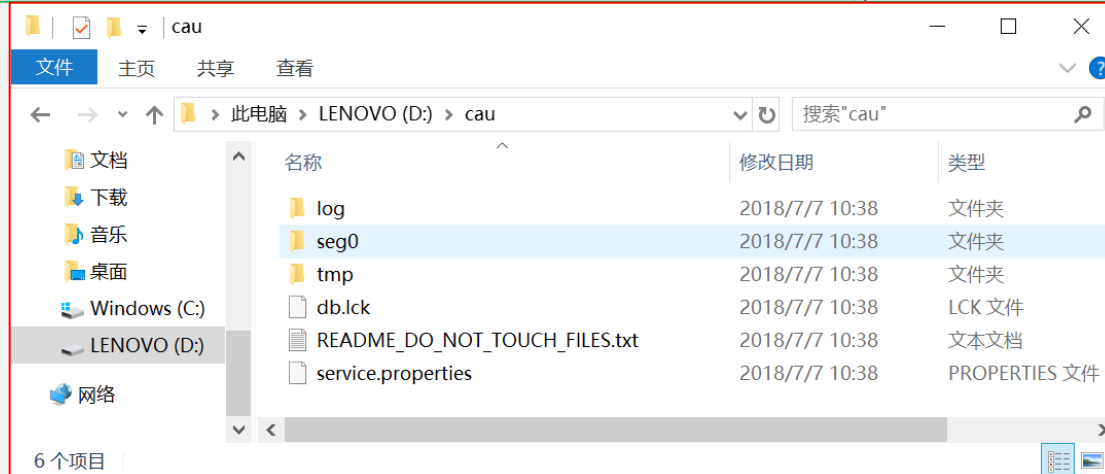
sNo 学号	sName 姓名	college 学院	class 班级
20180001	张同学	信息学院	计算181
20180002	李同学	信息学院	计算181
20180003	王同学	信息学院	计算182
20180004	赵同学	信息学院	计算182

- 插入新**记录**。在学生表student中插入4条新记录



例 10-1 在本地 JavaDB 数据库系统中创建教务信息数据库的 Java 示例程序 (JDBCCreate.java)

```
1  import java.sql.*;
2  public class JDBCCreate {
3      public static void main(String[] args) {
4          // 创建数据库连接
5          String url = "jdbc:derby:C:/temp/edu;create=true";
6          String user = "root";
7          String password = "root";
8          Connection con = null;
9          Statement s = null;
10         // 插入保存第 3 名同学信息的记录
11         String sqlInsert = "INSERT INTO student VALUES( " +
12             "'20180003', '王同学', '信电学院', '计算 182' )";
13         s.executeUpdate(sqlInsert);
14         // 插入保存第 4 名同学信息的记录
15         sqlInsert = "INSERT INTO student VALUES( " +
16             "'20180004', '赵同学', '信电学院', '计算 182' )";
17         s.executeUpdate(sqlInsert);
18         System.out.println("4 student records inserted.");
19         // 数据库访问结束，关闭 SQL 语句对象和数据库连接对象
20         s.close(); con.close();
21     }
22     catch(ClassNotFoundException e) { e.printStackTrace(); } // 驱动程序加载异常
23     catch(SQLException e) { e.printStackTrace(); } // SQL 语句执行异常
24 } }
```



例 10-2 查询并

1 impo
2 publ
3
4
5
6

```
15 // 查询语句会返回查询到的结果集 ResultSet，遍历并显示其中的记录
16 System.out.println("学生表 student 中的全部记录如下：");
17 while ( rs.next() ) { // 移到下一记录，如到末尾则返回 false，循环结束
18     System.out.print( rs.getString("sNo") + "\t" ); // 字段：学号 sNO
19     System.out.print( rs.getString("sName") + "\t" ); // 字段：姓名 sName
20     System.out.print( rs.getString("college") + "\t" ); // 字段：学院 college
21     System.out.println( rs.getString("class") ); // 字段：班级 class
22 }
23 // 查询并显示出学生表 student 中“计算 181”班同学的学号和姓名
24 sqlSelect = "SELECT sNo, sName FROM student " + // SQL：查询学号和姓名
25             "WHERE class = '计算 181'"; // 查询条件：班级='计算 181'
26 rs = s.executeQuery(sqlSelect); // 向 Java DB 提交 SQL 查询语句
```

Problems @ Javadoc Declaration Console

<terminated> JDBCSelect [Java Application] C:\Java\jre1.8.
org.apache.derby.jdbc.EmbeddedDriver loaded.
jdbc:derby:D:\cau; create=true connected.

学生表student中的全部记录如下：

20180001	张同学	信电学院	计算181
20180002	李同学	信电学院	计算181
20180003	王同学	信电学院	计算182
20180004	赵同学	信电学院	计算182

学生表student中“计算181”班同学的学号和姓名：

20180001	张同学
20180002	李同学

表 student 中“计算 181”班同学的学号和姓名：");

遍历并显示结果集 rs

```
rs.getString("sNo") + "\t" ); // 字段：学号 sNO
rs.getString("sName") ); // 字段：姓名 sName
```

闭 SQL 语句对象和数据库连接对象

```
e) { e.printStackTrace(); } // 驱动程序加载异常
ntStackTrace(); } // SQL 语句执行异常
```

10.3 JDBC数据库编程实验

- **修改或删除**记录

- 模拟数据库应用系统中用户**修改或删除**数据库数据的环节

- **修改**记录。将学生表student中学号为“20180001”同学的姓名由“张同学”改为“章同学”
- **删除**记录。删除学生表student中学号为“20180002”同学的记录



例 10-3 修改和删除学生表中学生记录的 Java 示例程序 (JDBUpdate.java)

Problems @ Javadoc Declaration Console

<terminated> JDBCSelect [Java Application] C:\Java\jre1.8.0

org.apache.derby.jdbc.EmbeddedDriver loaded.

jdbc:derby:D:\cau; create=true connected.

学生表student中的全部记录如下:

20180001 章同学 信电学院 计算181

20180003 王同学 信电学院 计算182

20180004 赵同学 信电学院 计算182

学生表student中“计算181”班同学的学号和姓名:

20180001 章同学

导入 java.sql 包中数据库相关的类和接口

主类: 修改或删除学生表中的学生记录

ing[] args) { // 主方法

库访问过程中可能出现的勾选异常

r = "org.apache.derby.jdbc.EmbeddedDriver"; // Java DB 驱动

(dbDriver); // 加载驱动

ntln(dbDriver + " loaded.");

"jdbc:derby:D:\\cau; create=true"; // 数据库 cau 的 JDBC URL

n = DriverManager.getConnection(dbURL); // 连接数据库

ntln(dbURL + " connected.");

学号为“20180001”同学的姓名由“张同学”改为“章同学”

con.createStatement(); // 创建语句对象

```

13      String sqlUpdate = "UPDATE student " + // SQL 修改语句: 修改记录
14          "SET sName='章同学' WHERE sNo='20180001'";
15      s.executeUpdate(sqlUpdate); // 向 Java DB 提交 SQL 修改语句
16      System.out.println("A student record updated.");
17      // 删除记录: 删除学生表中学号为“20180002”同学的记录
18      String sqlDelete = "DELETE FROM student " + // SQL 修改语句: 删除记录
19          "WHERE sNo='20180002'";
20      s.executeUpdate(sqlDelete); // 向 Java DB 提交 SQL 修改语句
21      System.out.println("A student record deleted.");
22      // 数据库访问结束, 关闭 SQL 语句对象和数据库连接对象
23      s.close(); con.close();
24  }
25  catch(ClassNotFoundException e) { e.printStackTrace(); } // 驱动程序加载异常
26  catch(SQLException e) { e.printStackTrace(); } // SQL 语句执行异常
27  } }
    
```

10.4 开启自己的Java探索之旅

- 关于Java语言，我们已经学习的内容有
 - Java语言的**基本语法**，以及如何搭建Java编程环境
 - **面向对象程序设计**方法，其中包括**类与对象编程**、**类的组合与继承**、**对象的替换与多态**，以及**接口编程**、**泛型编程**、**异常处理**等内容
 - 不同的应用**编程场景**，以及如何使用**Java API**类库进行应用编程。已学习的应用编程场景包括**数值计算**、**数据集合处理**、**图形用户界面**、**数据的输入输出**、**文字处理**、**图像和音频处理**、**多线程并发编程**、**网络编程**和**数据库编程**等



10.4 开启自己的Java探索之旅

- 实际应用中，计算机程序还有很多其他的应用场景
 - **程序测试**。测试是软件开发过程中的一个重要环节。每次修改程序后都需要对程序进行重新测试，是否可以编写一个测试程序来帮助程序员降低测试工作量呢？
 - **媒体播放器**。如果需要播放多媒体文件，如何在自己的Java程序中编写播放代码，内嵌一个媒体播放器呢？
 - 如何编写一个安卓（Android）系统的**App**？
 - 如何开发一个**Web**网络应用系统？
 -
- 通过**程序测试**和**媒体播放器**这两个应用场景，体验一下如何根据应用需求去**独立探索**新的Java技术



10.4 开启自己的Java探索之旅

- 单元测试JUnit
 - 软件测试（software testing）
 - 从搜索引擎开始
 - 发现并追踪“热词”
 - 单元测试（unit testing）
 - JUnit



10.4 开启自己的Java探索之旅

- 单元测试JUnit
 - 关于软件测试
 - **单元测试**：对单个程序单元进行测试
 - 集成测试：将各程序单元组装起来进行测试
 - 系统测试：将开发好的程序部署到实际运行环境中进行测试
 -
 - 关于Java单元测试与JUnit
 - 在Java语言中，一个类包含一组方法，每个方法实现一种算法
 - 在给定**输入**的情况下，算法应当能够按照设计要求得到对应的输出结果，这个输出结果被称作**预期输出**
 - 一个**输入**，再加上该输入对应的**预期输出**，这就构成算法的一个**测试用例**（test case）
 - 软件测试就是选择一组具有代表性的测试用例，然后检查算法的**实际输出**与**预期输出**是否一致。如果一致，则算法正确，否则算法存在错误
 - **JUnit**将每个**Java类**都作为一个独立的测试单元
 - 使用JUnit可以很方便地编写出**Java类**的**单元测试程序**



10.4 开启自己的Java探索之旅

- 单元测试JUnit
 - JUnit的使用
 - JUnit提倡一种“测试与编码同步进行”的编程理念
 - 基于JUnit来编写和测试Java类代码的过程
 - **编码工程师**编写Java类的定义代码
 - **测试工程师**为Java类设计测试用例，然后按照**JUnit代码框架**将测试用例编写成测试代码。测试工程师可以与编码工程师同步开展工作。JUnit以**断言**（assertion）的形式来比较算法的**实际输出与预期输出**是否一致
 - 测试工程师**运行测试代码**，测试编码工程师所编写的Java类。如果出现错误，则提请编码工程师修改。修改后再重新测试，直到没有错误为止。使用JUnit编写的测试代码是“一次编写，重复使用”
 - 在初步了解了单元测试、测试用例和JUnit编程理念之后，接下来最重要的事情是**试用**JUnit



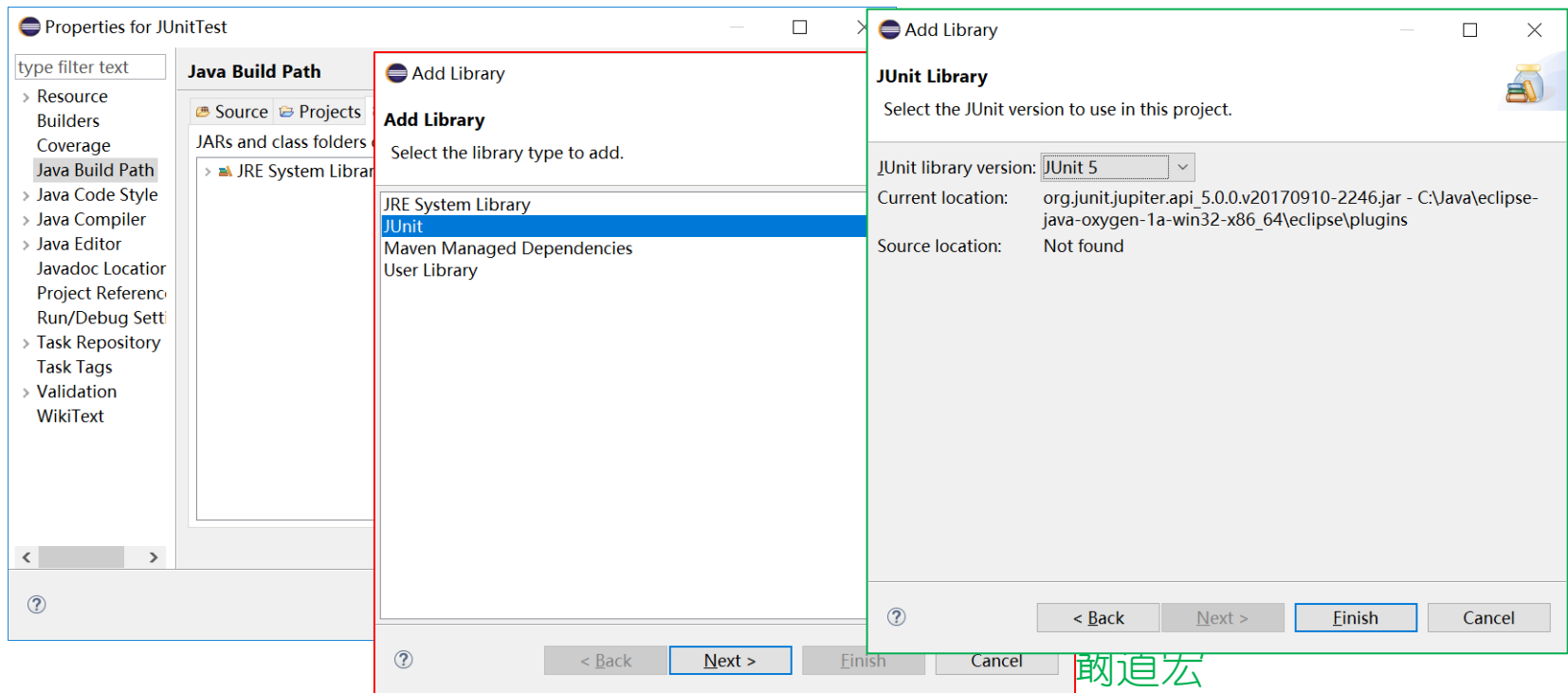
10.4 开启自己的Java探索之旅

- 单元测试JUnit
 - 试用JUnit
 - 入门教程、博客文章
 - 搭建JUnit编程环境
 - Eclipse 4.7版已经集成了JUnit单元测试类库的JAR包，因此不需要再单独下载安装JUnit
 - 新建JUnit试用项目：JUnitTest
 - 导入JUnit单元测试的JAR包
 - 菜单“Project” → “Properties”，进入项目的属性页对话框



10.4 开启自己的Java探索之旅

- 单元测试JUnit
 - 试用JUnit
 - 导入JUnit单元测试的**JAR包**



10.4 开启自己的Java探索之旅

- 单元测试JUnit

- 试用JUnit

- 在项目JUnitTest中新建待测试类

```
public class A { // A是需要被测试的类
    private int a; // 数据成员

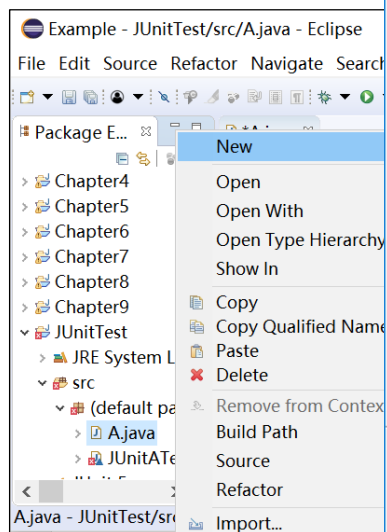
    public A(int x) { a = x; } // 构造方法
    public void set(int x) { a = x; } // 设置数据
    public int get() { return a; } // 读取数据
    public int getSquare() { return a + a; } // 计算平方值
}
```



10.4 开启自

• 单元测试JUnit – 试用JUnit

- 在项目JUnitTest
- 在项目JUnitTest



New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

☐ New JUnit 3 test ☐ New JUnit 4 test ☒ New JUnit Jupiter test

Source folder: JUnitTest/src Browse...

Package: (default) Browse...

Name: JUnitATester

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test: A Browse...

< Back Next > Finish Cancel



中國農業大學

閻道宏

10.4 开启自己的Java探索之旅

- 单元测试JUnit

- 试用JUnit

- 在项目JUnitTest中新建待测试类
 - 在项目JUnitTest中新建测试用例类

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class JUnitATester {
    @Test
    void test() {
        fail("Not yet implemented");
    }
}
```



10.4 开启自己的Java探索之旅

例10-4 一个完整的测试用例类JUnitATester的示例代码（JDBUpdate.java）

```
1  import static org.junit.jupiter.api.Assertions.*; // 导入JUnit相关的类库
2  import org.junit.jupiter.api.Test;
3
4  class JUnitATester { // 测试用例类
5      /* 测试目的：检查待测试类A中各方法成员的算法是否正确
6       * 测试方法：使用断言assertEquals()来检查实际输出与预期输出是否一致
7       */
8      @Test // 注解：指定其后面的方法testA()是一个在测试时需要被执行的方法
9      void testA() { // 测试构造方法
10         A obj = new A(5); // 新建一个对象obj并初始化为5。初值5是测试用例的输入
11         assertEquals(obj.get(), 5); // 调用get()方法，使用断言检查实际输出是否为5
12     }
13     @Test
14     void testSetGet() { // 测试设置数据方法set()和读取数据方法get()
15         A obj = new A(0); // 新建一个对象obj
16         obj.set(6); // 调用set()方法，然后再调用get()方法，检查结果
17         assertEquals(obj.get(), 6); // 预期输出为6，使用断言检查实际输出是否一致
18     }
19     @Test
20     void testGetSquare() { // 测试计算平方值方法getSquare()
21         A obj = new A(7); // 新建一个对象obj，并给一个初值7
22         assertEquals(obj.getSquare(), 49); // 预期输出49，使用断言检查实际输出是否一致
23     } }
```



10.4 开启自己的Java探索之旅

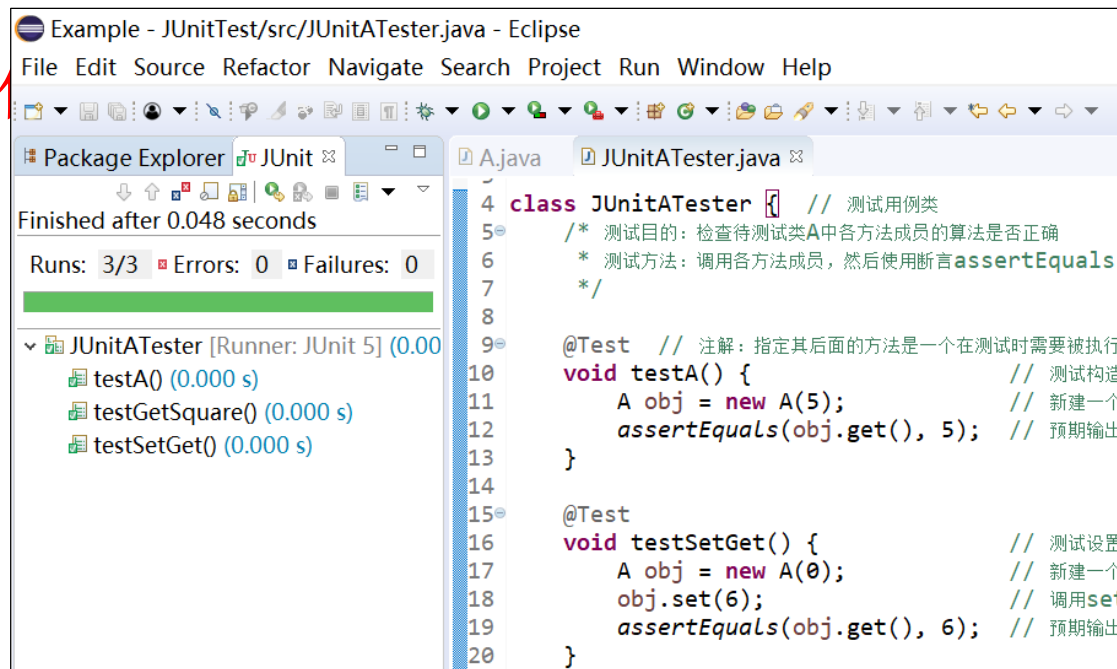
- 单元测试JUnit

- 试用JUnit

- 在项目JUnitTest中新建待测试类
 - 在项目JUnitTest中新建测试用例类
 - 运行测试用例JUnitATester: 右键 “Run As” → “JUnit Test”

- 评价

- 程序阶段确定
阶段具体实现的
设计变动而经常
使用”（只要
量
目中使用



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the JUnit test results for the JUnitATester class. The results show three tests passed: testA() (0.000 s), testGetSquare() (0.000 s), and testSetGet() (0.000 s). The total execution time is 0.048 seconds. On the right, the JUnitATester.java source code is displayed. The code defines a class JUnitATester with two test methods: testA() and testSetGet(). Both methods use assertEquals to verify the results of the methods being tested.

```
Example - JUnitTest/src/JUnitATester.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit
Finished after 0.048 seconds
Runs: 3/3 Errors: 0 Failures: 0

JUnitATester [Runner: JUnit 5] (0.00
testA() (0.000 s)
testGetSquare() (0.000 s)
testSetGet() (0.000 s)

4 class JUnitATester // 测试用例类
5 /* 测试目的: 检查待测试类A中各方法成员的算法是否正确
6 * 测试方法: 调用各方法成员, 然后使用断言assertEquals
7 */
8
9 @Test // 注解: 指定其后面的方法是一个在测试时需要被执行
10 void testA() { // 测试构造
11     A obj = new A(5); // 新建一个
12     assertEquals(obj.get(), 5); // 预期输出
13 }
14
15 @Test
16 void testSetGet() { // 测试设置
17     A obj = new A(0); // 新建一个
18     obj.set(6); // 调用set
19     assertEquals(obj.get(), 6); // 预期输出
20 }
```


10.4 开启自己的Java探索之旅

- 多媒体框架JMF
 - 如果需要**播放**多媒体文件，是否可以在自己的Java程序中编写播放代码，**内**嵌一个媒体播放器呢？
 - 换句话说，能否找到一个多媒体**类库**帮助自己快速编写**媒体播放器**程序呢？
 - 搜索引擎：**JMF**
 - Java多媒体框架（Java Media Framework）
 - 管理器**Manager**
 - 播放器**Player**
 - 控制监听器**ControllerListener**



10.4 开启自己的Java探索之旅

- 多媒体框架JMF

- 下载安装JMF

- Windows操作系统安装包

- jmf-2_1_1e-windows-i586.exe

- 导入安装目录下lib子目录中的JAR包文件jmf.jar

- 模仿编写媒体播放器程序



中國農業大學

阚道宏

例 10-5 一个基于 JMF 的媒体

```
1 import java.awt.*;
2 import javax.swing.
3 import javax.media.
4
5 public class JMFPlay
6     public static void
7         JFrame w
8         w.setSize
9         w.setDefa
10        w.validate
11        // 播放
12        String fil
13        //String
14        MMPlay
15    } }
16
```

```
17 // 定义一个自己的媒体播放器类，需实现 JMF 的控制监听器接口 ControllerListener
18 class MMPlayer implements ControllerListener { // 媒体播放器类
19     Player p; // Player 是 JMF 的播放器接口
20     JFrame win; // 程序主窗口：播放器将被作为组件添加到主窗口中
21     String mmFile; // 等待播放的音频或视频文件 URL
22     MMPlayer(JFrame w, String s) { // 构造媒体播放器对象
23         win = w; mmFile = s;
24         try { // 处理可能出现的勾选异常
25             MediaLocator ml = new MediaLocator(mmFile); // JMF 的类 MediaLocator
26             p = Manager.createPlayer(ml); // 通过 JMF 的类 Manager 创建播放器对象
27             p.addControllerListener(this); // 添加控制监听器，监听播放器的事件
28             p.prefetch(); // 先加载多媒体文件中的数据
29         } catch (Exception e) { e.printStackTrace(); }
30     }
31     // 实现控制监听器接口 ControllerListener 规定的抽象方法 controllerUpdate()
32     public void controllerUpdate(ControllerEvent e) { // 监听并处理播放器相关的事件
33         if (e instanceof RealizeCompleteEvent) { // 如果是播放器对象创建完成事件
34             Component mmArea = p.getVisualComponent(); // 获取播放区域组件
35             Component cPanel = p.getVisualComponent(); // 获取控制面板组件
36             // 将播放器对象的播放区域组件和控制面板组件添加到程序主窗口中
37             if (mmArea != null) win.add(mmArea, BorderLayout.CENTER);
38             if (cPanel != null) win.add(cPanel, BorderLayout.SOUTH);
39             win.validate();
40         }
41         else if (e instanceof PrefetchCompleteEvent) { // 如果是多媒体数据加载完成事件
42             p.start(); // 开始播放
43         }
44     } }

```

10.4 开启自己的Java探索之旅

- 多媒体框架JMF
 - 下载安装JMF
 - 模仿编写媒体播放器程序
 - 运行多媒体播放器程序
 - 只能播放WAV、AVI等格式的多媒体文件
 - 不支持目前流行的MP3或MP4多媒体文件格式
 - 评估结论：JMF不能在实际项目中使用，试用失败



10.4 开启自己的Java探索之旅

- 安卓App和Web网络应用程序
 - 如何编写一个安卓系统的**App**?
 - 安卓App在本质上是一种运行于**安卓智能手机**上的图形用户界面程序
 - 安卓操作系统由**谷歌**（google）主导，开发安卓系统App主要使用**Java**语言
 - 在自己的计算机上搭建安卓系统App开发环境
 - 下载安装**JDK**和**Eclipse**（或Android Studio）
 - 下载安装**Android SDK**（Software Development Kit）和**ADT**（Android Development Tools） for Eclipse。



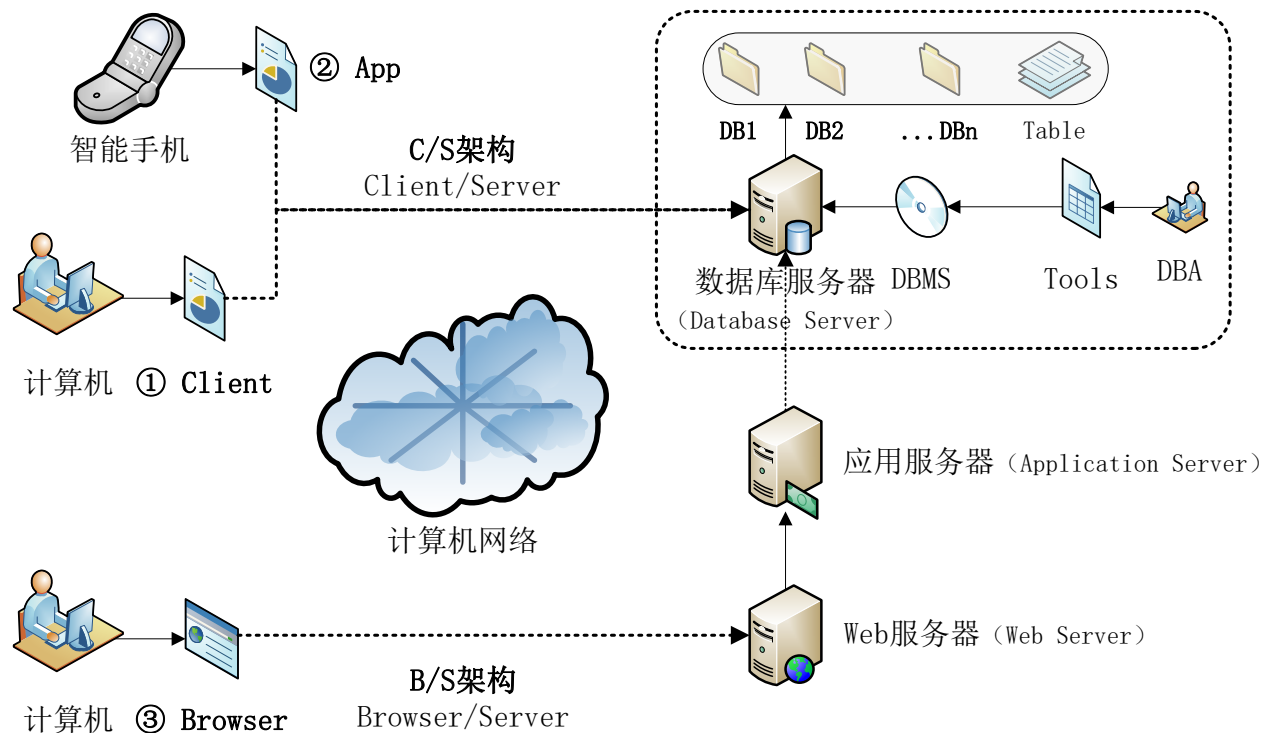
10.4 开启自己的Java探索之旅

- 安卓App和Web网络应用程序
 - 如何开发一个**Web**网络应用系统？
 - **C/S架构**的网络应用程序
 - Web网络应用系统是另一种架构的网络应用程序，其客户端统一使用浏览器，因此这种网络应用程序架构被称作Browser/Server程序架构，简称**B/S架构**
 - HTTP、HTML、CSS、JavaScript、jQuery、Ajax、JSP
 - JavaBean、Servlet、Strut+Spring+Hibernate
 - Java实训课程、网上相关的MOOC课程



10.4 开启自己的Java探索之旅

- Java **网络应用程序** 架构示意图



中國農業大學

閻道宏

第10章 数据库编程

- 本章学习要点

- 了解数据库的基本原理，学习SQL语言和JDBC编程框架
- 熟练运用JDBC API编写数据库应用程序
- 本章所学习的数据库知识已基本能够满足数据库编程的需要。如果希望深入学习数据库，读者可以选择专门的数据库课程，系统学习数据库相关的基础理论和设计方法
- 继续自己的Java之旅。不忘初心，砥砺前行！

