

Tarena Teaching System

JAVA 数据库编程

学员用书



Java企业应用及互联网
高级工程师培训课程

目 录

Unit01	1
1. JDBC 原理	3
1.1. JDBC 标准	3
1.1.1. JDBC 是什么	3
1.1.2. JDBC 接口及数据库厂商实现	3
1.1.3. JDBC 工作原理	3
1.1.4. Driver 接口及驱动类加载	4
1.1.5. Connection 接口	4
1.1.6. Statement 接口	5
1.1.7. ResultSet 接口	5
1.2. 数据库厂商实现	6
1.2.1. Oracle 实现	6
1.2.2. MySQL 实现	6
2. JDBC 基础编程	6
2.1. 连接管理	6
2.1.1. 通过连接工具类获取连接	6
2.1.2. 通过属性文件维护连接属性	7
2.1.3. 从类路径中加载属性文件	7
2.1.4. 连接的关闭	7
2.2. 连接池技术	8
2.2.1. 为什么要使用连接池	8
2.2.2. 使用 Apache DBCP 连接池	9
2.2.3. 通过 DataSource 获取连接	10
2.2.4. 连接池参数	10
2.3. 异常处理	10
2.3.1. SQLException 简介	10
2.3.2. 处理 SQLException	12
经典案例	13
1. JDBC 实现对 Emp 数据的简单查询 (Oracle)	13
2. JDBC 实现对 Emp 数据的简单查询 (MySQL)	21
3. 实现 DBUtility , 提供连接的获取 , 关闭功能	25
4. 使用 Apache DBCP 连接池重构 DBUtility	31
课后作业	39
Unit02	41
1. JDBC 核心 API	42

1.1. Statement	42
1.1.1. Statement 执行查询	42
1.1.2. Statement 执行插入	42
1.1.3. Statement 执行更改	43
1.2. PreparedStatement	43
1.2.1. PreparedStatement 原理	43
1.2.2. 通过 PS 提升性能	44
1.2.3. SQL Injection 简介	45
1.2.4. 通过 PS 防止 SQL Injection	46
1.3. ResultSet	46
1.3.1. 结果集遍历	46
1.3.2. ResultSetMetaData	46
1.3.3. 可滚动结果集	47
经典案例	48
1. 更新和插入 Emp 数据	48
2. 用户名密码验证功能	61
课后作业	73
Unit03	74
1. JDBC 高级编程	76
1.1. 事务处理	76
1.1.1. 事务简介	76
1.1.2. JDBC 事务 API	77
1.1.3. JDBC 标准事务编程模式	77
1.2. 批量更新	77
1.2.1. 批量更新的优势	77
1.2.2. 批量更新 API	78
1.2.3. 防止 OutOfMemory	78
1.3. 返回自动主键	78
1.3.1. 关联数据插入	78
1.3.2. 通过序列产生主键 (Oracle)	79
1.3.3. JDBC 返回自动主键 API	79
1.4. 分页查询	80
1.4.1. JDBC 实现 Oracle 分页查询	80
1.4.2. JDBC 实现 MySQL 分页查询	81
2. DAO	81
2.1. 什么是 DAO	81
2.1.1. DAO 封装对数据的访问	81
2.1.2. 实体对象	82
2.2. 编写 DAO	82
2.2.1. 查询方法	82

2.2.2. 更新方法	83
2.2.3. 异常处理机制	83
经典案例	84
1. 实现账户转账操作	84
2. 批量插入 Emp 数据	90
3. 插入 Dept 及其关联的 Emp 信息	96
4. 实现 Emp 数据的分页查询 (Oracle 和 MySQL)	108
5. 完成 NetCTOSS 项目中，账务账号的 DAO 设计及实现	131
课后作业	158

Java 数据库编程

Unit01

知识体系.....**Page 3**

JDBC 原理	JDBC 标准	JDBC 是什么
		JDBC 接口及数据库厂商实现
		JDBC 工作原理
		Driver 接口及驱动类加载
		Connection 接口
		Statement 接口
		ResultSet 接口
	数据库厂商实现	Oracle 实现
		MySQL 实现
JDBC 基础编程	连接管理	通过连接工具类获取连接
		通过属性文件维护连接属性
		从类路径中加载属性文件
		连接的关闭
	连接池技术	为什么要使用连接池
		使用 Apache DBCP 连接池
		通过 DataSource 获取连接
	异常处理	SQLException 简介
		处理 SQLException

经典案例.....**Page 13**

JDBC 实现对 Emp 数据的简单查询 (Oracle)	Driver 接口及驱动类加载
	Connection 接口
	Statement 接口
	ResultSet 接口
JDBC 实现对 Emp 数据的简单查询 (MySQL)	MySQL 实现
实现 DBUtility , 提供连接的获取 , 关闭功能	通过连接工具类获取连接
	通过属性文件维护连接属性
	从类路径中加载属性文件

	连接的关闭
使用 Apache DBCP 连接池重构 DBUtility	为什么要使用连接池
	使用 Apache DBCP 连接池
	通过 DataSource 获取连接
	连接池参数

课后作业.....**Page 39**

1. JDBC 原理

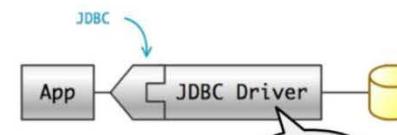
1.1. JDBC 标准

1.1.1. 【JDBC 标准】 JDBC 是什么

知识讲解

JDBC是什么

- Java Database Connectivity : Java访问数据库的解决方案
- 希望用相同的方式访问不同的数据库，以实现与具体数据库无关的Java操作界面
- JDBC 定义一套标准接口，即访问数据库的通用API，不同的数据库厂商根据各自数据库的特点去实现这些接口



+ 知识讲解

1.1.2. 【JDBC 标准】 JDBC 接口及数据库厂商实现

知识讲解

JDBC接口及数据库厂商实现

- DriverManager → 驱动管理
- Connection → 连接接口
- DatabaseMetaData
- Statement → 语句对象接口
- PreparedStatement
- CallableStatement
- ResultSet → 结果集接口
- ResultSetMetaData

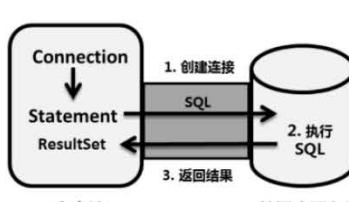
+ 知识讲解

1.1.3. 【JDBC 标准】 JDBC 工作原理

知识讲解

JDBC工作原理

- JDBC定义接口
- 数据库厂商实现接口
- 程序员调用接口，实际调用的是底层数据库厂商的实现部分



+ 知识讲解

Technology
Tarena
达内科技

JDBC工作原理（续1）

- JDBC工作过程：
 - 加载驱动，建立连接
 - 创建语句对象
 - 执行SQL语句
 - 处理结果集
 - 关闭连接

知识
讲解

1.1.4. 【JDBC 标准】Driver 接口及驱动类加载

Driver接口及驱动类加载

• 驱动类加载方式 (Oracle) :

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

装载驱动类，驱动类通过static块实现在
DriverManager 中的
“自动注册”

1.1.5. 【JDBC 标准】Connection 接口

Connection接口

Class.forName("oracle.jdbc.OracleDriver")

知识点讲解

```
Connection conn = DriverManager.getConnection(  
    "jdbc:oracle:thin:@192.168.0.26:1521:tarena",  
    "openlab", "open123");
```

根据url连接参数
找到与之匹配的
Driver对象，调用其
方法获取连接

NOTE:Connection只是接口！真正的实现是由数据库厂商
提供的驱动包完成的

1.1.6. 【JDBC 标准】Statement 接口

Technology
Tarena
达内科技

Statement接口

- Statement stmt=conn.createStatement();

知识讲解

The diagram illustrates the `Statement` interface. A central box labeled `stmt` has three arrows pointing to three separate boxes, each representing a different method:

- `boolean flag = stmt.execute(sql);`
- `ResultSet rs = stmt.executeQuery(sql);`
- `int flag = stmt.executeUpdate(sql);`

1.1.7. 【JDBC 标准】 ResultSet 接口

Tarena
达内科技

ResultSet接口

- 执行查询SQL语句后返回的结果集，由ResultSet接口接收
- 常用处理方式：遍历 / 判断是否有结果（登录）

知识讲解

```
String sql = "select * from emp";
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
    System.out.println(rs.getInt("empno") + ", "
        + rs.getString("ename"));
}
```

Tarena
达内科技

ResultSet接口(续1)

- 查询的结果存放在ResultSet对象的一系列行中
- ResultSet对象的最初位置在行首
- ResultSet.next()方法用来在行间移动
- ResultSet.getXXX()方法用来取得字段的内容

知识讲解

1.2. 数据库厂商实现

1.2.1. 【数据库厂商实现】Oracle 实现

知识讲解

Oracle实现

• 下载对应数据库的驱动
ojdbc6.jar / ojdbc14.jar

• 将驱动类加载到项目中
MyEclipse : Build Path

• 加载驱动类
Class.forName("oracle.jdbc.OracleDriver")

+

1.2.2. 【数据库厂商实现】MySQL 实现

知识讲解

MySQL实现

• 下载对应数据库的驱动
mysql-connector-java-5.0.4-bin.jar

• 将驱动类加载到项目中
MyEclipse : Build Path

• 加载驱动类
Class.forName("com.mysql.jdbc.Driver");

+

2. JDBC 基础编程

2.1. 连接管理

2.1.1. 【连接管理】通过连接工具类获取连接

知识讲解

通过连接工具类获取连接

• 在工程中，编写一个访问数据库的工具类，此后所有访问数据库的操作，都从工具类中获取连接

• 两种方式：

- 直接把数据配置写在工具类中
- 把数据库配置写在一个properties属性文件里，工具类读取属性文件，逐行获取数据库参数

建议第二种

+

2.1.2. 【连接管理】通过属性文件维护连接属性

Tarena
达内科技

通过属性文件维护连接属性

#驱动类名

jdbc.driver=oracle.jdbc.driver.OracleDriver

#连接字符串

jdbc.url=jdbc:oracle:thin:@192.168.0.26:1521:tarena

#访问数据库的用户名

jdbc.user=openlab

#访问数据库的密码

jdbc.password=open123

NOTE : 在properties文件中，#符号表示注释

2.1.3. 【连接管理】从类路径中加载属性文件

从类路径中加载属性文件

String path =
"com/tarena/dms/daodemo/v2/db.properties";

属性文件所在的
位置

properties.load(DBUtility.class.getClassLoader()
.getResourceAsStream(path));

获得当前类的路
径，加载指定属
性文件

知识
讲解

+

2.1.4. 【连接管理】连接的关闭

Tarena
达内科技

连接的关闭

- 在工具类中定义公共的关闭连接的方法
- 所有访问数据库的应用，共享此方法

知识讲解

```
public static void closeConnection(Connection con) {  
    if (con != null) {  
        try {  
            con.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

2.2. 连接池技术

2.2.1. 【连接池技术】为什么要使用连接池

Tarena
达内科技

为什么要使用连接池

- 数据库连接的建立及关闭资源消耗巨大
- 传统数据库访问方式：一次数据库访问对应一个物理连接，每次操作数据库都要打开、关闭该物理连接，系统性能严重受损

知识讲解

- 解决方案：数据库连接池（Connection Pool）

系统初始运行时，主动建立足够的连接，组成一个池。每次应用程序请求数据库连接时，无需重新打开连接，而是从池中取出已有的连接，使用完后，不再关闭，而是归还

The diagram illustrates the concept of a connection pool. A large blue circle represents the pool, containing several wavy lines that represent individual database connections. Two yellow cubes outside the circle are connected by lines to two specific points on the wavy lines inside the circle, representing how a client application connects to a pool of available connections.

Tarena
达内科技

为什么要使用连接池（续2）

- 连接池中连接的释放与使用原则
 - 应用启动时，创建初始化数目的连接
 - 当申请时无连接可用或者达到指定的最小连接数，按增量参数值创建新的连接
 - 为确保连接池中最小的连接数的策略：
 - 动态检查：定时检查连接池，一旦发现数量小于最小连接数，则补充相应的新连接，保证连接池正常运转
 - 静态检查：空闲连接不足时，系统才检测是否达到最小连接数
 - 按需分配，用过归还，空闲超时释放，获取超时报错
- 连接池也只是接口，具体实现由厂商来完成

2.2.2. 【连接池技术】使用 Apache DBCP 连接池

知识讲解

使用Apache DBCP连接池

• DBCP(DataBase connection pool)：数据库连接池
• Apache 的一个 Java 连接池开源项目，同时也是 Tomcat 使用的连接池组件
• 连接池是创建和管理连接的缓冲池技术，将连接准备好被任何需要它们的应用使用

+

知识讲解

使用Apache DBCP连接池 (续1)

```
graph LR; Client[客户端] -- "Connection" --> Statement[Statement]; Statement -- "1. 获得连接" --> Pool[连接池<br/>C1<br/>C2<br/>C3<br/>.....]; Pool -- "2. 传递SQL" --> Database[数据库服务器]; Database -- "3. 执行SQL" --> Result[Result]; Result -- "4. 返回结果" --> Pool; Pool -- "5. 释放连接" --> Client;
```

客户端 → Connection → Statement → 1. 获得连接 → 连接池 (C1, C2, C3, ...) → 2. 传递SQL → 数据库服务器 → 3. 执行SQL → 4. 返回结果 → 5. 释放连接 → 客户端

+

知识讲解

使用Apache DBCP连接池 (续2)

- 需要两个jar包文件
 - commons-dbc-1.4.jar 连接池的实现
 - commons-pool-1.5.jar 连接池实现的依赖库
- MyEclipse : Build Path

+

2.2.3. 【连接池技术】通过 DataSource 获取连接

Technology
Tarena
达内科技

通过DataSource获取连接

- 通过属性文件获取连接池参数
- 加载这些参数，获得连接

```
private static BasicDataSource dataSource  
= new BasicDataSource();
```

```
dataSource.setDriverClassName(driveClassName);  
dataSource.setUrl(url);  
dataSource.setUsername(username);  
dataSource.setPassword(password);
```

```
Connection conn = dataSource.getConnection();
```

2.2.4. 【连接池技术】连接池参数

连接池参数

- 常用参数有：
 - 初始连接数
 - 最大连接数
 - 最小连接数
 - 每次增加的连接数
 - 超时时间
 - 最大空闲连接
 - 最小空闲连接
- 根据应用需要，设置合适的值

知识讲解

+

2.3. 异常处理

2.3.1. 【异常处理】SQLException 简介

知识讲解

SQLException简介（续1）

• 一般的SQLException都是因为操作数据库时出错，比如Sql语句写错，或者数据库中的表或数据出错

• 常见异常：

- 登录被拒绝
可能原因：程序里取键值对信息时的大小写和属性文件中不匹配
- 列名无效
可能原因：查找的表和查找的列不匹配
- 无效字符
可能原因：sql语句语法有错，比如语句结尾时不能有分号
- 无法转换为内部表示
可能原因：结果集取数据时注意数据类型

+

知识讲解

SQLException简介（续2）

- 表或者视图不存在
检查SQL中的表名是否正确
- 不能将空值插入
检查执行insert操作时，是否表有NOT NULL约束，而没有给出数据
- 缺少表达式
检查SQL语句的语法
- SQL命令未正确结束
检查SQL语句的语法
- 无效数值
企图将字符串类型的值填入数字型而造成，检查SQL语句

+

知识讲解

SQLException简介（续3）

- 其他可能出现的异常
 - 文件找不到
可能原因：db.properties文件路径不正确

+

2.3.2. 【异常处理】处理 SQLException

处理SQLException

- SQLException属于Checked Exception，必须使用try...catch或throws明确处理

```
public static synchronized Connection getConnection() throws  
    SQLException {  
    //语句  
}
```

知识讲解

```
try {  
    //语句  
} catch (SQLException e) {  
    e.printStackTrace(); //追踪处理  
    //throw new RuntimeException(e); //或者抛出  
}
```

经典案例

1. JDBC 实现对 Emp 数据的简单查询 (Oracle)

- 问题

Oracle 数据库中职员 emp 表的表结构如表 - 1 所示：

表 - 1 职员表 emp 信息

字段名	类型	描述
empno	NUMBER(4,0)	员工 ID
ename	VARCHAR2(10)	员工姓名
job	VARCHAR2(9)	职位
mgr	NUMBER(4,0)	员工管理者的 ID
hiredate	DATE	入职日期
sal	NUMBER(7,2)	薪资
comm	NUMBER(7,2)	绩效
deptno	NUMBER(2,0)	员工所在的部门 ID

职员表 emp 中的示例数据，如图 - 1 所示：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980/12/17	800.00		20
7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
7839	KING	PRESIDENT		1981/11/17	5000.00		10
7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
7900	JAMES	CLERK	7698	1981/12/3	950.00		30
7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
7934	MILLER	CLERK	7782	1982/1/23	1300.00		10

图 - 1

本案例要求使用 JDBC 连接 Oracle 数据库，查询 emp 表的所有员工的 ID、姓名、薪资以及入职时间。

- 方案

使用 JDBC 连接 Oracle 数据库的基本步骤如下：

- 装载驱动程序。

代码如下：

```
Class.forName("oracle.jdbc.OracleDriver");
```

2. 建立连接。

通过调用 `DriverManager` 的 `getConnection` 方法，获取 `Connection` 类的对象，建立连接。代码如下：

```
con = DriverManager.getConnection(  
    "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
```

其中 `getConnection` 方法有三个参数 第一个参数表示连接数据库的字符串格式 URL，它包含了要连接数据库信息，例如本案例中连接 Oracle 数据库的连接字符串格式为：

```
jdbc:oracle:thin:@<主机名>:<端口号(默认 1521)>:<实例名>
```

第二个参数表示连接数据库的用户名，第三个参数表示连接数据库的密码，可以根据连接数据库的不同，用户名和密码的不同，相应的修改以上所述的三个参数。

3. 发送和执行 SQL 语句。

首先 通过 `Connection` 的 `createStatement()` 方法获取数据库操作对象 `Statement`。代码如下：

```
stmt = con.createStatement();
```

`Statement` 主要用于执行 SQL 语句并返回它所生成结果。通过调用它的 `executeQuery` 方法来执行 SQL 语句。

```
rs = stmt.executeQuery("select empno, ename, sal, hiredate from emp");
```

其中，`executeQuery` 方法有一个参数，表示要执行的 SQL 语句，该方法的返回值为 `ResultSet` 对象。`ResultSet` 表示数据库查询操作的结果集。它具有指向其当前数据行的光标。最初，光标被置于第一行之前，调用其 `next` 方法将光标移动到下一行，该方法在 `ResultSet` 对象没有下一行时返回 `false`，因此可以在 `while` 循环中使用它来迭代结果集。代码如下所示：

```
while (rs.next()) {  
    System.out.println(rs.getInt("empno") + ","  
        + rs.getString("ename") + ","  
        + rs.getDouble("sal") + "," + rs.getDate("hiredate"));  
}
```

从上述代码中看出 `ResultSet` 提供了 `getXXX(String column)` 方法，例如：`getInt (String column)` 等，获取当前 `ResultSet` 对象的当前行中指定列名的值，其中参数 `column` 表示数据库表中的列名字。

另外，`ResultSet` 提供了 `getXXX(int columnIndex)` 方法，例如：`getInt (int`

columnIndex) 等 , 获取当前 ResultSet 对象的当前行中指定列索引的值 , 其中参数 columnIndex 表示要获取的列的数据在表中是第几列 , 注意 : 列数从 1 开始。建议使用根据列名去获取列的数据信息。

4. 释放资源

数据库的连接资源是有限的 , 使用完毕后需要及时释放。否则会因资源耗尽使应用程序瘫痪 , 无法正常工作。调用 JDBC 对象的 close 方法即可释放与该对象关联的系统资源。代码如下 :

```
if (rs != null) {  
    rs.close();  
}  
if (stmt != null) {  
    stmt.close();  
}  
if (con != null) {  
    con.close();  
}
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：导入连接 Oracle 数据库所需的 jar 包

创建工程 , 在当前工程下导入连接 Oracle 数据库对应的驱动程序 jar 包。

步骤二：新建类 EmpDAO 及方法 findAll

代码如下所示 :

```
public class EmpDAO {  
    public static void main(String[] args) {  
    }  
  
    public void findAll() {  
    }  
}
```

步骤三：构建连接数据所需的对象以及相应的异常处理

在 findAll 方法中 , 构建连接数据所需的对象以及相应的异常处理 , 代码如下所示 :

```
public class EmpDAO {  
    public static void main(String[] args) {  
    }  
  
    public void findAll() {  
  
        Connection con = null;  
        Statement stmt = null;  
        ResultSet rs = null;  
  
        try {  
            Class.forName("oracle.jdbc.OracleDriver");  
        }
```

```

        } catch (ClassNotFoundException e) {
            System.out.println("驱动类无法找到!");
            throw new RuntimeException(e);
        } catch (SQLException e) {
            System.out.println("数据库访问异常!");
            throw new RuntimeException(e);
        }

    }
}

```

步骤四：装载驱动程序

代码如下所示：

```

public class EmpDAO {
public static void main(String[] args) {

}

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {

        Class.forName("oracle.jdbc.OracleDriver");

    } catch (ClassNotFoundException e) {
        System.out.println("驱动类无法找到!");
        throw new RuntimeException(e);
    } catch (SQLException e) {
        System.out.println("数据库访问异常!");
        throw new RuntimeException(e);
    }
}
}

```

步骤五：建立连接。

通过调用 DriverManager 的 getConnection 方法，获取 Connection 类的对象，建立连接。代码如下所示：

```

public class EmpDAO {
public static void main(String[] args) {

}

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        Class.forName("oracle.jdbc.OracleDriver");

        con = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
    }
}

```

```

        } catch (ClassNotFoundException e) {
            System.out.println("驱动类无法找到！");
            throw new RuntimeException(e);
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        }
    }
}

```

步骤六：发送和执行 SQL 语句

首先 通过 Connection 的 createStatement()方法获取数据库操作对象 Statement。通过调用 Statement 对象的 executeQuery 方法来执行 SQL 语句。代码如下所示：

```

public class EmpDAO {
public static void main(String[] args) {

}

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        Class.forName("oracle.jdbc.OracleDriver");
        con = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");

        stmt = con.createStatement();
        rs = stmt
            .executeQuery("select empno, ename, sal, hiredate from emp");

    } catch (ClassNotFoundException e) {
        System.out.println("驱动类无法找到！");
        throw new RuntimeException(e);
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    }
}
}

```

步骤七：处理查询结果

Statement 的 executeQuery 方法的返回值为 ResultSet 对象。ResultSet 表示数据库查询操作的结果集。它具有指向其当前数据行的光标。最初，光标被置于第一行之前，调用其 next 方法将光标移动到下一行。该方法在 ResultSet 对象没有下一行时返回 false，因此可以在 while 循环中使用它来迭代结果集。代码如下所示：

```

public class EmpDAO {
public static void main(String[] args) {

}

public void findAll() {
    Connection con = null;
    Statement stmt = null;

```

```

ResultSet rs = null;

try {
    Class.forName("oracle.jdbc.OracleDriver");
    con = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
    stmt = con.createStatement();
    rs = stmt
        .executeQuery("select empno, ename, sal, hiredate from emp");

    while (rs.next()) {
        System.out.println(rs.getInt("empno") + ","
            + rs.getString("ename") + ","
            + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
    }
} catch (ClassNotFoundException e) {
    System.out.println("驱动类无法找到！");
    throw new RuntimeException(e);
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
}
}
}
    
```

从上述代码中看出 `ResultSet` 提供了 `getXXX(String column)` 方法，例如：`getInt(String column)` 等，获取当前 `ResultSet` 对象的当前行中指定列名的值，其中参数 `column` 表示数据库表中的列名字。

步骤八：释放资源

在 `finally` 块中 依次关闭 `ResultSet` 对象、`Statement` 对象以及 `Connection` 对象。

代码如下：

```

public class EmpDAO {
    public static void main(String[] args) {

    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + ","
                    + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
            }
        } catch (ClassNotFoundException e) {
            System.out.println("驱动类无法找到！");
            throw new RuntimeException(e);
        } catch (SQLException e) {
    }
}
    
```

```
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);

    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
```

步骤九：测试

在 main 方法中，调用 findAll 方法，代码如下所示：

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

        EmpDAO dao = new EmpDAO();
        dao.findAll();

    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + ","
                    + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
            }
        } catch (ClassNotFoundException e) {
            System.out.println("驱动类无法找到！");
            throw new RuntimeException(e);
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
            }
```

```
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("关闭连接时发生异常");
    }
}
```

运行 EmpDAO 类，控制台输出结果如下所示：

7369,SMITH,,800.0,1980-12-17
7499,ALLEN,,1600.0,1981-02-20
7521,WARD,,1250.0,1981-02-22
7566,JONES,,2975.0,1981-04-02
7654,MARTIN,,1250.0,1981-09-28
7698,BLAKE,,2850.0,1981-05-01
7782,CLARK,,2450.0,1981-06-09
7788,SCOTT,,3000.0,1987-04-19
7839,KING,,5000.0,1981-11-17
7844,TURNER,,1500.0,1981-09-08
7876,ADAMS,,1100.0,1987-05-23
7900,JAMES,,950.0,1981-12-03
7902,FORD,,3000.0,1981-12-03
7934,MILLER,,1300.0,1982-01-23

从输出结果可以看出，已经查询到 emp 表的所有员工的 ID、姓名、薪资以及入职时间。

• 完整代码

本案例的完整代码如下所示：

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

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:ORCL", "scott", "tiger");
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
        }
    }
}
```

```

        while (rs.next()) {
            System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + ","
                + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
        }
    } catch (ClassNotFoundException e) {
        System.out.println("驱动类无法找到！");
        throw new RuntimeException(e);
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
}

```

2. JDBC 实现对 Emp 数据的简单查询 (MySQL)

- 问题

MySQL 数据库中职员 emp 表的表结构如表 - 2 所示：

表 - 2 职员表 emp 信息

字段名	类型	描述
empno	int(4,0)	员工 ID
ename	varchar(10)	员工姓名
job	varchar (9)	职位
mgr	int(4,0)	员工管理者的 ID
hiredate	date	入职日期
sal	double(7,2)	薪资
comm	double(7,2)	绩效
deptno	double(2,0)	员工所在的部门 ID

职员表 emp 中的示例数据，如图 - 2 所示：

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980-12-17	800.00	<NULL>	20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	<NULL>	20
7654	MARTIN	SALESMAN	7698	1981-09-21	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	<NULL>	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	<NULL>	10
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	<NULL>	20
7839	KING	PRESIDENT	<NULL>	1981-11-17	5000.00	<NULL>	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987-05-27	1100.00	<NULL>	20
7900	JAMES	CLERK	7698	1981-12-01	950.00	<NULL>	30
7902	FORD	ANALYST	7566	1981-12-03	3000.00	<NULL>	20
7934	MILLER	CLERK	7782	1982-01-23	1300.00	<NULL>	10

图 - 2

本案例要求使用 JDBC 连接 MySql 数据库，查询 emp 表的所有员工的 ID、姓名、薪资以及入职时间。

• 方案

使用 JDBC 连接 MySql 数据库的基本步骤和 Oracle 类似，只有前两个步骤不同。

1. 装载驱动程序。

在此，需要使用 MySql 数据库的驱动类，代码如下：

```
Class.forName("com.mysql.jdbc.Driver");
```

2. 建立连接。

建立连接时，连接的字符串格式发生变化，代码如下：

```
con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/tts7", "root", "root");
```

其中 getConnection 方法有三个参数 第一个参数表示连接数据库的字符串格式 URL，它包含了要连接数据库信息，例如本案例中连接 MySql 数据库的连接字符串格式为：

```
jdbc:mysql://<主机名>:<端口号(默认 3306)>/<数据库名>
```

第二个参数表示连接数据库的用户名，第三个参数表示连接数据库的密码，可以根据连接数据库的不同，用户名和密码的不同，相应的修改以上所述的三个参数。

其余，步骤与连接 Oracle 数据类似，这里不再赘述。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：在 MySql 数据库中，创建 emp 表并插入测试数据

SQL 语句如下所示：

```

CREATE TABLE emp(
    empno int(4),
    ename VARCHAR(10),
    job VARCHAR(9),
    mgr int(4),
    hiredate DATE,
    sal double(7,2),
    comm double(7,2),
    deptno double(2,0)
);

INSERT INTO emp VALUES(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);
INSERT INTO emp VALUES(7499,'ALLEN','SALESMAN',7698,'1981-2-20',1600,300,30);
INSERT INTO emp VALUES(7521,'WARD','SALESMAN',7698,'1981-2-22',1250,500,30);
INSERT INTO emp VALUES(7566,'JONES','MANAGER',7839,'1981-4-2',2975,NULL,20);
INSERT INTO emp VALUES(7654,'MARTIN','SALESMAN',7698,'1981-9-21',1250,1400,30);
INSERT INTO emp VALUES(7698,'BLAKE','MANAGER',7839,'1981-5-1',2850,NULL,30);
INSERT INTO emp VALUES(7782,'CLARK','MANAGER',7839,'1981-6-9',2450,NULL,10);
INSERT INTO emp VALUES(7788,'SCOTT','ANALYST',7566,'1987-4-19',3000,NULL,20);
INSERT INTO emp VALUES(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10);
INSERT INTO emp VALUES(7844,'TURNER','SALESMAN',7698,'1981-9-8',1500,0,30);
INSERT INTO emp VALUES(7876,'ADAMS','CLERK',7788,'1987-5-27',1100,NULL,20);
INSERT INTO emp VALUES(7900,'JAMES','CLERK',7698,'1981-12-1',950,NULL,30);
INSERT INTO emp VALUES(7902,'FORD','ANALYST',7566,'1981-12-3',3000,NULL,20);
INSERT INTO emp VALUES(7934,'MILLER','CLERK',7782,'1982-1-23',1300,NULL,10);

```

步骤二：导入连接 MySql 数据库所需的 jar 包

在当前工程下导入连接 MySql 数据库对应的驱动程序 jar 包。

步骤三：重构 EmpDAO 类

重构 EmpDAO 类，在该类中使用 MySql 数据库的驱动程序和连接 MySql 数据库的连接字符串格式，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }
}

```

```

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {

        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/tts7", "root", "root");

        stmt = con.createStatement();
        rs = stmt
            .executeQuery("select empno, ename, sal, hiredate from emp");
        while (rs.next()) {
            System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + "," + rs.getDouble("sal")
                + "," + rs.getDate("hiredate"));
        }
    } catch (ClassNotFoundException e) {
        System.out.println("驱动类无法找到！");
        throw new RuntimeException(e);
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
}
    
```

运行 EmpDAO 类，控制台输出结果如下所示：

```

7369,SMITH,,800.0,1980-12-17
7499,ALLEN,,1600.0,1981-02-20
7521,WARD,,1250.0,1981-02-22
7566,JONES,,2975.0,1981-04-02
7654,MARTIN,,1250.0,1981-09-28
7698,BLAKE,,2850.0,1981-05-01
7782,CLARK,,2450.0,1981-06-09
7788,SCOTT,,3000.0,1987-04-19
7839,KING,,5000.0,1981-11-17
7844,TURNER,,1500.0,1981-09-08
7876,ADAMS,,1100.0,1987-05-23
7900,JAMES,,950.0,1981-12-03
7902,FORD,,3000.0,1981-12-03
7934,MILLER,,1300.0,1982-01-23
    
```

从输出结果可以看出，已经查询到 emp 表的所有员工的 ID、姓名、薪资以及入职时间。

- 完整代码

本案例中，类 EmpDAO 的完整代码如下所示：

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

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/tts7", "root", "root");
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + "," + rs.getDouble("sal")
                    + "," + rs.getDate("hiredate"));
            }
        } catch (ClassNotFoundException e) {
            System.out.println("驱动类无法找到！");
            throw new RuntimeException(e);
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("关闭连接时发生异常");
            }
        }
    }
}
```

3. 实现 DBUtility，提供连接的获取，关闭功能

- 问题

将数据库连接的获取和关闭封装到 DBUtility 类中，并重构案例 “JDBC 实现对 Emp 数据的简单查询（ Oracle ）”，使用 DBUtility 类获取连接和关闭连接。

- 方案

封装数据库连接的获取和关闭的过程如下：

1. 创建属性 db.properties，在该文件中以键值对的形式来存储连接数据库的相关信息，该文件的内容如下：

```
jdbc.driver=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:ORCL
jdbc.user=scott
jdbc.password=tiger
```

这样做的目的是当需要修改连接数据的信息时，只需修改该文件集合，降低了连接数据库的信息与使用类之间的耦合。

2. 创建 DBUtility 类，使用 static 块，初始化连接数据的信息，代码如下：

```
static {
    try {
        // 加载配置文件

        properties.load(DBUtility.class.getClassLoader().getResourceAsStream(
            ));
        driver = properties.getProperty("jdbc.driver");
        url = properties.getProperty("jdbc.url");
        user = properties.getProperty("jdbc.user");
        pwd = properties.getProperty("jdbc.password");
        Class.forName(driver);
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

其中，“day01/v3/db.properties”表示文件 db.properties 所在的包路径，可以按照你自己工程的包路径来写代码。

3. 创建打开和关闭连接数据库的方法，代码如下：

```
public static Connection openConnection() throws SQLException {
    return DriverManager.getConnection(url, user, pwd);
}
public static void closeConnection(Connection con) {
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
```

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建属性 db.properties

创建属性 db.properties ,在该文件中以键值对的形式来存储连接数据库的相关信息 ,该文件的内容如下 :

```
jdbc.driver=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:ORCL
jdbc.user=scott
jdbc.password=tiger
```

步骤二：在类中初始化数据库连接信息

首先 , 创建 DBUtility 类 ; 在该类中 , 使用 static 块 , 加载 db.properties 中的文件内容 , 来初始化数据库连接数据的信息 , 代码如下所示 :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBUtility {
    private static Properties properties = new Properties() ;
    private static String driver = null;
    private static String url = null;
    private static String user = null;
    private static String pwd = null;

    static {
        try {
            // 加载配置文件

            properties.load(DBUtility.class.getClassLoader().getResourceAsStream(
                    "day01/v3/db.properties"));
            driver = properties.getProperty("jdbc.driver");
            url = properties.getProperty("jdbc.url");
            user = properties.getProperty("jdbc.user");
            pwd = properties.getProperty("jdbc.password");
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
}
```

步骤三：创建打开和关闭连接数据库的方法

在 DBUtility 类中 , 添加 openConnection 方法和 closeConnection , 来实现打开数据库连接和关闭数据库连接的功能 , 代码如下所示 :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBUtility {
    private static Properties properties = new Properties() ;
```

```

private static String driver = null;
private static String url = null;
private static String user = null;
private static String pwd = null;

static {
    try {
        // 加载配置文件

        properties.load(DBUtility.class.getClassLoader().getResourceAsStream(
            "day01/v3/db.properties"));
        driver = properties.getProperty("jdbc.driver");
        url = properties.getProperty("jdbc.url");
        user = properties.getProperty("jdbc.user");
        pwd = properties.getProperty("jdbc.password");
        Class.forName(driver);
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}

public static Connection openConnection() throws SQLException {
    return DriverManager.getConnection(url, user, pwd);
}
public static void closeConnection(Connection con) {
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
}

```

步骤四：重构 EmpDAO 类，使用 DBUtility 类获取连接和关闭连接

重构 EmpDAO 类，在该类中使用 DBUtility 类的 openConnection 获取连接、使用 closeConnection 方法关闭连接，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }
}

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {

        con = DBUtility.openConnection();

```

```

stmt = con.createStatement();
rs = stmt
    .executeQuery("select empno, ename, sal, hiredate from emp");
while (rs.next()) {
    System.out.println(rs.getInt("empno") + ","
        + rs.getString("ename") + "," + ","
        + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
}
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}

DBUtility.closeConnection(con);

}
}
}

```

运行类 EmpDAO，输出结果与案例“JDBC 实现对 Emp 数据的简单查询（Oracle）”是相同的。

- **完整代码**

本案例中，db.properties 文件的内容如下所示：

```

jdbc.driver=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:ORCL
jdbc.user=scott
jdbc.password=tiger

```

类 DBUtility 的完整代码如下所示：

```

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

public class DBUtility {
    private static Properties properties =new Properties() ;
    private static String driver = null;
    private static String url = null;
    private static String user = null;
    private static String pwd = null;

    static {
        try {
            // 加载配置文件

```

```

properties.load(DBUtility.class.getClassLoader().getResourceAsStream(
    "day01/v3/db.properties"));
driver = properties.getProperty("jdbc.driver");
url = properties.getProperty("jdbc.url");
user = properties.getProperty("jdbc.user");
pwd = properties.getProperty("jdbc.password");
Class.forName(driver);
} catch (Exception e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}
}

public static Connection openConnection() throws SQLException {
    return DriverManager.getConnection(url, user, pwd);
}

public static void closeConnection(Connection con) {
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println("关闭连接时发生异常");
        }
    }
}
}

```

EmpDAO 类的完整代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            con = DBUtility.openConnection();
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + "," + ","
                    + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
            } catch (SQLException e) {

```

```
        System.out.println("释放资源时发生异常");
    }
    DBUtility.closeConnection(con);
}
}
```

4. 使用 Apache DBCP 连接池重构 DBUtility

- 问题

本案例要求使用 Apache DBCP 连接池重构类 DBUtility 为 ConnectionSource 类，并重构案例“实现 DBUtility，提供连接的获取，关闭功能”中的 EmpDAO 类，在该类中使用 ConnectionSource 来获取连接。

- 方案

直接使用 JDBC 访问数据库时，需要避免以下隐患：

1. 每一次数据操作请求都需要建立数据库连接、打开连接、存取数据和关闭连接等步骤。而建立和打开数据库连接是一件既耗资源又费时的过程，如果频繁发生这种数据库操作，势必会使系统性能下降。

2. 连接对象代表着数据库系统的连接进程，是有限的资源。如果系统的使用用户非常多，有可能超出数据库服务器的承受极限，造成系统的崩溃。

数据库连接池是解决上述问题最常用的方法。所谓连接池，即可以创建并持有数据库连接的组件。连接池可以预先创建并封装一些连接对象并将其缓存起来，当需要使用连接对象时可以向连接池“借”一个连接，用完之后将其“归还”到连接池中。数据库连接池的主要功能如下：

1. 连接池对象的创建和释放。
2. 服务器启动时，创建指定数量的数据库连接。
3. 为用户请求提供可用连接。如果没有空闲连接，且连接数没有超出最大值，创建一个新的数据库连接。
4. 将用户不再使用的连接标识为可用连接，等待其他用户请求。
5. 当空闲的连接数过多时，释放连接对象。

连接池组件一般都需要实现 JDBC 规范中的 javax.sql.DataSource 接口。DataSource 接口定义了获取连接的方法 getConnection 方法。常用的连接池组件有 DBCP、c3p0 和 proxool 等，本案例以 Apache 的 DBCP 组件为例来实现数据库连接池。简单的应用代码如下所示：

```
BasicDataSource ds = new BasicDataSource();
ds.setUrl("jdbc:oracle:thin:@192.168.0.23:1521:tarena");
ds.setDriverClassName("oracle.jdbc.OracleDriver");
ds.setUsername("openlab");
ds.setPassword("open123");

Connection con = ds.getConnection();
```

```

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select count(*) from emp");
if (rs.next()) {
    System.out.println(rs.getInt(1));
}
stmt.close();
con.close();

```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：导入使用 DBCP 组件所需的 jar 包

在当前工程下，导入使用 DBCP 组件所需的 jar 包，包括 commons-dbcp.jar 以及 commons-pool.jar 两个 jar 包，这两个 jar 包的名字可能会因为版本的不同，名字的最后为版本信息，例如：commons-dbcp-1.2.1.jar。

步骤二：重构 db.properties 文件

重构 db.properties 文件，在该文件中添加创建数据库连接池所需的信息，包括初始化连接数、最大空闲连接数、大小空闲连接数、最大连接数量以及超时回收时间。该文件内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000

```

步骤三：创建 ConnectionSource 类，在该类中初始化数据源信息

首先创建 ConnectionSource 类；然后在该类中添加 init 方法，在该方法中数据源信息进行初始，代码如下所示：

```

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;
import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }
}

```

```

public static void init() {

    Properties dbProps = new Properties();
    // 取配置文件可以根据实际的不同修改
    try {

        dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                "day01/v4/db.properties"));
    } catch (IOException e) {
        e.printStackTrace();
    }

    try {
        String                     driveClassName
dbProps.getProperty("jdbc.driverClassName");
        String url = dbProps.getProperty("jdbc.url");
        String username = dbProps.getProperty("jdbc.username");
        String password = dbProps.getProperty("jdbc.password");

        String initialSize = dbProps.getProperty("dataSource.initialSize");
        String minIdle = dbProps.getProperty("dataSource.minIdle");
        String maxIdle = dbProps.getProperty("dataSource.maxIdle");
        String maxWait = dbProps.getProperty("dataSource.maxWait");
        String maxActive = dbProps.getProperty("dataSource.maxActive");

        dataSource = new BasicDataSource();
        dataSource.setDriverClassName(driveClassName);
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);

        // 初始化连接数
        if (initialSize != null)
            dataSource.setInitialSize(Integer.parseInt(initialSize));

        // 最小空闲连接
        if (minIdle != null)
            dataSource.setMinIdle(Integer.parseInt(minIdle));

        // 最大空闲连接
        if (maxIdle != null)
            dataSource.setMaxIdle(Integer.parseInt(maxIdle));

        // 超时回收时间(以毫秒为单位)
        if (maxWait != null)
            dataSource.setMaxWait(Long.parseLong(maxWait));

        // 最大连接数
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!");
    }
}
}

```

步骤四：添加获取连接的方法

在 ConnectionSource 类中添加获取连接的方法 `getConnection`，代码如下所示：

```

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }

    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                    "day01/v4/db.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            String                     driveClassName = dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
            dataSource.setPassword(password);

            // 初始化连接数
            if (initialSize != null)
                dataSource.setInitialSize(Integer.parseInt(initialSize));

            // 最小空闲连接
            if (minIdle != null)
                dataSource.setMinIdle(Integer.parseInt(minIdle));

            // 最大空闲连接
            if (maxIdle != null)
                dataSource.setMaxIdle(Integer.parseInt(maxIdle));

            // 超时回收时间(以毫秒为单位)
            if (maxWait != null)
                dataSource.setMaxWait(Long.parseLong(maxWait));

            // 最大连接数
            if (maxActive != null) {
                if (!maxActive.trim().equals("0"))
                    dataSource.setMaxActive(Integer.parseInt(maxActive));
            }
        } catch (Exception e) {
    }
}

```

```
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}
```

getConnection 方法使用 synchronized 修饰，来保证线程安全。

步骤四：重构 EmpDAO 类

重构 EmpDAO 类，在该类中使用 ConnectionSource 类的 getConnection 方法获取连接，代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {

            con = ConnectionSource.getConnection();

            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + "," + ","
                    + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
            }
```

```
        }
        if (stmt != null) {
            stmt.close();
        }

        if (con != null) {
            con.close();
        }

    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
```

在此，调用 Connection 类的 close 方法关闭连接，会将该连接归还到连接池中。

运行类 EmpDAO，输出结果与案例“JDBC 实现对 Emp 数据的简单查询（Oracle）”是相同的。

• 完整代码

本案例中，db.properties 文件的完整内容如下所示：

```
jdbc.driverClassName=oracle.jdbc.OracleDriver  
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl  
jdbc.username=scott  
jdbc.password=tiger  
  
#<!-- 初始化连接 -->  
dataSource.initialSize=10  
#<!-- 最大空闲连接 -->  
dataSource.maxIdle=20  
#<!-- 最小空闲连接 -->  
dataSource.minIdle=5  
#最大连接数量  
dataSource.maxActive=50  
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->  
dataSource.maxWait=1000
```

`ConnectionSource` 类的完整代码如下所示：

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }

    public static void init() {
        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
    }
}
```

```
try {
    dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
        "day01/v4/db.properties"));
} catch (IOException e) {
    e.printStackTrace();
}

try {
    String                     driveClassName      =
dbProps.getProperty("jdbc.driverClassName");
    String url = dbProps.getProperty("jdbc.url");
    String username = dbProps.getProperty("jdbc.username");
    String password = dbProps.getProperty("jdbc.password");

    String initialSize = dbProps.getProperty("dataSource.initialSize");
    String minIdle = dbProps.getProperty("dataSource.minIdle");
    String maxIdle = dbProps.getProperty("dataSource.maxIdle");
    String maxWait = dbProps.getProperty("dataSource.maxWait");
    String maxActive = dbProps.getProperty("dataSource.maxActive");

    dataSource = new BasicDataSource();
    dataSource.setDriverClassName(driveClassName);
    dataSource.setUrl(url);
    dataSource.setUsername(username);
    dataSource.setPassword(password);

    // 初始化连接数
    if (initialSize != null)
        dataSource.setInitialSize(Integer.parseInt(initialSize));

    // 最小空闲连接
    if (minIdle != null)
        dataSource.setMinIdle(Integer.parseInt(minIdle));

    // 最大空闲连接
    if (maxIdle != null)
        dataSource.setMaxIdle(Integer.parseInt(maxIdle));

    // 超时回收时间(以毫秒为单位)
    if (maxWait != null)
        dataSource.setMaxWait(Long.parseLong(maxWait));

    // 最大连接数
    if (maxActive != null) {
        if (!maxActive.trim().equals("0"))
            dataSource.setMaxActive(Integer.parseInt(maxActive));
    }
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("创建连接池失败!请检查设置!!!");
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
```

类 EmpDAO 的完整代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            rs = stmt
                .executeQuery("select empno, ename, sal, hiredate from emp");
            while (rs.next()) {
                System.out.println(rs.getInt("empno") + ","
                    + rs.getString("ename") + "," + ","
                    + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}
```

课后作业

1. 下列关于 JDBC , 说法正确的是

- A. JDBC 只提供了对 Java 程序员的 API。
- B. JDBC 只提供了对数据库厂商的 API。
- C. JDBC 只提供了第三方中间件厂商的 API。
- D. JDBC 提供了对 Java 程序员 , 数据库厂商及第三方中间件厂商的 API。

2. 简述 JDBC 的原理

3. JDBC 实现对 Dept 数据的简单查询 (Oracle)

Oracle 数据库中部门 dept 表的表结构如表 - 1 所示 :

表 - 1 部门表 dept 信息

字段名	类型	描述
deptno	NUMBER(2,0)	部门 ID
dname	VARCHAR2(14)	部门名称
loc	VARCHAR2(13)	部门所在地

部门表 dept 中的示例数据 , 如图 - 1 所示 :

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

图 - 1

本案例要求使用 JDBC 连接 Oracle 数据库 , 查询 dept 表的所有部门的 ID、部门名称以及部门所在地。

4. 下列 JDBCURL 书写正确的是

- A. JDBC 连接 Oracle 数据库的 URL 为 jdbc:oracle:thin:@<主机名>:<端口号(默认 1521)>:<实例名>
- B . JDBC 连接 Oracle 数据库的 URL 为 oracle:jdbc:thin:@<主机名>:<端口号(默认 1521)>:<实例名>
- C . JDBC 连接 MySql 数据库的 URL 为 mysql:jdbc://<主机名>:<端口号(默认

3306) >/<数据库名>

D . JDBC 连接 MySql 数据库的 URL 为 `jdbc:mysql://<主机名>:<端口号(默认3306)>/<数据库名>`

5. JDBC 实现对 Dept 数据的简单查询 (MySQL)

Oracle 数据库中部门 dept 表的表结构如表 - 2 所示：

表 - 2 部门表 dept 信息

字段名	类型	描述
deptno	int(2,0)	部门 ID
dname	varchar(14)	部门名称
loc	varchar(13)	部门所在地

部门表 dept 中的示例数据 , 如图 - 2 所示 :

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

图 - 2

本案例要求使用 JDBC 连接 MySQL 数据库 , 查询 dept 表的所有部门的 ID、部门名称以及部门所在地。

6. 根据下列 SQLException 信息判断可能出现的错误

请看下列异常 :

1. `java.sql.SQLException:列名无效`
2. `java.sql.SQLException:ORA-00911 : 无效字符`
3. `java.sql.SQLException:无法转换为内部表示`

发生上述异常的原因是什么 ?

Java 数据库编程

Unit02

知识体系.....**Page 42**

JDBC 核心 API	Statement	Statement 执行查询
		Statement 执行插入
		Statement 执行更改
	PreparedStatement	PreparedStatement 原理
		通过 PS 提升性能
		SQL Injection 简介
		通过 PS 防止 SQL Injection
	ResultSet	结果集遍历
		ResultSetMetaData
		可滚动结果集

经典案例.....**Page 48**

更新和插入 Emp 数据	Statement 执行插入
	Statement 执行更改
用户名密码验证功能	SQL Injection 简介
	通过 PS 防止 SQL Injection

课后作业.....**Page 73**

1. JDBC 核心 API

1.1. Statement

1.1.1. 【Statement】 Statement 执行查询

Training
Tarena
达内科技

Statement执行查询

- 创建Statement的方式：
`Connection.createStatement();`
- 执行INSERT, UPDATE和DELETE：
`Statement.executeUpdate()`
- 执行SELECT：
`Statement.executeQuery()`

知识
讲解

Tarena
达内科技

Statement执行查询(续1)

```
String sql = "select empno, ename, sal, hiredate from emp";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

知识讲解

//对rs的处理
stmt.close();

方法将返回SQL语句执行后的结果集

+

1.1.2. 【Statement】 Statement 执行插入

Tarena
达内科技

Statement执行插入

```
String sql = "insert into emp(empno, ename, job, sal)  
values(1001, '张三丰', 'Manager' 9500);  
int flag = -1;  
try {  
    con = ConnectionSource.getConnection();  
    stmt = con.createStatement();  
    flag = stmt.executeUpdate(sql);  
    //处理结果  
}catch(SQLException e){  
    //处理异常  
}
```

方法将返回SQL语句执行后影响的记录数

1.1.3. 【Statement】 Statement 执行更改

Statement执行更改

和INSERT操作完全相同，只是SQL语句不同

```
String sql = "update emp set sal = 9900 where empno =
1001";
int flag = -1 ;
try {
    con = ConnectionSource.getConnection();
    stmt = con.createStatement();
    flag = stmt.executeUpdate(sql);
    //处理结果
} catch(SQLException e){
    //处理异常
}
+}
```



1.2. PreparedStatement

1.2.1. 【PreparedStatement】 PreparedStatement 原理

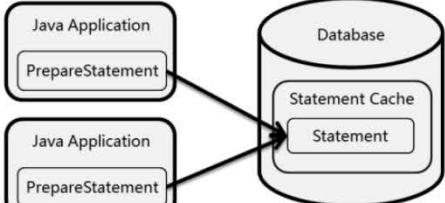
PreparedStatement原理

- Statement主要用于执行静态SQL语句，即内容固定不变的SQL语句
- Statement每执行一次都要对传入的SQL语句编译一次，效率较差
- 某些情况下，SQL语句只是其中的参数有所不同，其余子句完全相同，适用于PreparedStatement
- 预防sql注入攻击



PreparedStatement原理（续1）

- PreparedStatement是接口，继承自Statement
- SQL语句提前编译，三种常用方法 execute、executeQuery 和 executeUpdate 已被更改，不再需要参数



PreparedStatement原理(续2)

Tarena
达内科技

- `PreparedStatement` 实例包含已事先编译的 SQL 语句
 - SQL 语句可有一个或多个 IN 参数
 - IN参数的值在 SQL 语句创建时未被指定。该语句为每个 IN 参数保留一个问号（“？”）作为**占位符**
 - 每个问号的值必须在该语句执行之前，通过适当的`setInt`或者`setString`方法提供。
 - 由于 `PreparedStatement` 对象已预编译过，所以其执行速度要快于 `Statement` 对象。因此，多次执行的 SQL 语句经常创建为 `PreparedStatement` 对象，以提高效率。
 - 批量处理



PreparedStatement原理(续3)

Tarena 达内科技

```
PreparedStatement pstmt =  
    con.prepareStatement(  
        "UPDATE emp SET job= ? WHERE empno = ?");  
pstmt.setLong(1, "Manager");  
pstmt.setInt(2,1001);  
pstmt.executeUpdate();
```

SQL语句已发送
给数据库，并编
译好为执行作好
准备。

执行SQL语句

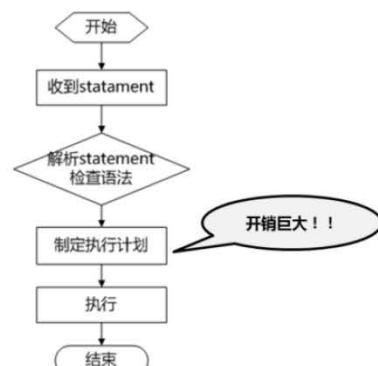
准备



1.2.2. 【PreparedStatement】通过 PS 提升性能

通过PS提升性能

Tarena 技术领先



通过PS提升性能（续1）

知识讲解

- 数据库具备缓存功能，可以对statement的执行计划进行缓存，以免重复分析
- 缓存原理：
 - 使用statement本身作为key并将执行计划存入与statement对应的缓存中
 - 对曾经执行过的statements，再运行时执行计划将重用
- 举例：
 - SELECT a, b FROM t WHERE c = 1 ;
 - 再次发送相同的statement时，数据库会对先前使用过的执行计划进行重用，降低开销

+

通过PS提升性能（续2）

知识讲解

- 悲剧：
 - SELECT a, b FROM t WHERE c = 1 ;
 - SELECT a, b FROM t WHERE c = 2 ;

```
String sql = "select a,b from t where c = ?";  
PreparedStatement ps = conn.prepareStatement(sql);  
for (int i = 0; i < 1000; i++) {  
    ps.setInt(1, i);  
    ResultSet rs = ps.executeQuery();  
    rs.close();  
    ps.close();  
}
```

被视作不同的SQL语句，执行计划不可重用

执行计划可重用

+

1.2.3. 【PreparedStatement】 SQL Injection 简介

SQL Injection简介

知识讲解

- 场景：

```
String sql = "select * from t where username = " +  
    name + " and password = " + passwd + "!";
```

- 输入参数后，数据库接受到的完整sql语句将是：

```
select * from t where username = 'scott' and  
password = 'tiger';
```

- 如果用户输入的passwd参数是：a' or 'b' = 'b，则数据库收到的SQL语句将是：

```
select * from t where username = 'scott' and  
password = 'a' or 'b' = 'b';
```

此SQL语句的where条件将永远为true

这种现象被称作SQL注入

+

1.2.4. 【PreparedStatement】通过 PS 防止 SQL Injection

通过PS防止SQL Injection

- 对JDBC而言，SQL注入攻击只对Statement有效，对PreparedStatement无效，因为PreparedStatement不允许在插入参数时改变SQL语句的逻辑结构
- 使用预编译的语句对象，用户传入的任何数据不会和原SQL语句发生匹配关系，无需对输入的数据做过滤
- 如果用户将“ or 1 = 1” 传入赋值给占位符，SQL语句将无法执行

```
select * from t  
where username = ? and password = ?;
```

+

1.3. ResultSet

1.3.1. 【ResultSet】结果集遍历

Technology
Tarena
达内科技

结果集遍历

- 常用的遍历方式：

```
String sql = "select empno, ename, sal, hiredate  
from emp";
```

```
rs = stmt.executeQuery(sql);
```

```
while ((rs.next())) {
```

```
    int empno = rs.getInt("empno");
```

```
    String ename = rs.getString("ename");
```

```
    double sal = rs.getDouble("sal");
```

```
    Date hiredate = rs.getDate("hiredate");
```

```
}
```

```
rs.close();
```

1.3.2. 【ResultSet】 ResultSetMetaData

Technology
Tarena
达内科技

ResultSetMetaData

- ResultSetMetaData: 数据结果集的元数据
- 和查询出来的结果集相关，从结果集(ResultSet)中获取

知识讲解

```
ResultSetMetaData rsm = rs.getMetaData();
int columnCount = rsm.getColumnCount();
String columnName = null;
for (int i = 1; i <=columnCount; i++) {
    columnName = rsm.getColumnName(i);
}
```

1.3.3. 【ResultSet】可滚动结果集

知识讲解

可滚动结果集

• 常用的ResultSet，初始指针在第一行之前（Before First）
• 只能使用next()方法将指针向后移动，不能反向
• 一次移动一行，不能跳行
• 可滚动的结果集：指针可以在结果集中任意移动

+

知识讲解

可滚动结果集（续1）

• 获得可滚动的ResultSet，Statement的创建有所不同：
Statement stmt = conn.createStatement(type, concurrency);
PreparedStatement stmt = conn.prepareStatement(sql, type, concurrency)
• type取值：
TYPE_FORWARD_ONLY: 只能向前移动，默认参数
TYPE_SCROLL_INSENSITIVE: 可滚动，不感知数据变化
TYPE_SCROLL_SENSITIVE: 可滚动，感知数据变化
• concurrency取值：
CONCUR_READ_ONLY: 只读
CONCUR_UPDATABLE: 可更新

+

知识讲解

可滚动结果集（续2）

• 可滚动结果集的常用方法：
– first
– last
– beforeFirst
– afterLast
– isFirst
– isLast
– isBeforeFirst
– isAfterLast
– relative
– next
– previous
– absolute

+

经典案例

1. 更新和插入 Emp 数据

- 问题

本案例要求使用 JDBC 向 Emp 表中插入和更新数据，详细要求如下：

1. 向 Emp 表中插入一条记录。其中为列 empno、ename、job、mgr、hiredate、sal、comm、deptno 的数据分别为 1001、"rose"、"Analyst"、7901、"2014-05-01"，3000.00、500.00、10。
2. 更新职员 ID 为 1001 的薪资为 4500。

- 方案

Statement 对象提供了 executeUpdate 方法，该方法可以执行指定的 sql 语句，该语句可以是 insert、update、delete。应用代码如下：

```
int result = stmt.executeUpdate(sql);
```

另外，我们在设计 add 方法时，该方法的参数是 Emp 类型，方法的声明如下：

```
public void add(Emp emp) {}
```

之所以把 Emp 类作为 add 方法的参数，是因为我们要保存的职员数据在 Emp 表的有 8 个字段，也就是说有 9 项内容需要存入数据中。如果不把 Emp 类型作为 add 方法的参数类型，那么 add 方法将有 8 个参数，造成参数过多。对于数据库中的表来说 8 个字段不算多，在企业中做项目的时候可能会有几十个字段的情况。所以使用对象封装方法参数是十分有必要的。另外，update 方法的设计与 add 方法的设计类似。

Emp 类是数据库表 Emp 和 Java 实体类之间的映射，创建该类遵守以下规则：

1. 如果类的成员变量的名字是 xxx，那么为了更改或获取成员变量的值，即更改或获取属性的值，在类中可以使用 getter 和 setter 方法，方法的命名如下：

- getXxx()，用来获取属性 xxx
- setXxx()，用来修改属性 xxx

2. 对于 boolean 类型的成员变量，即布尔逻辑类型的属性，允许使用"is"代替上述的"get"和"set"。

3. getter 和 setter 方法须为 public 的。

4. 类中如果有构造方法，那么这个构造方法为 public 的并且是无参数的。

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建 Emp 类

创建 Emp 类，该类为数据库表 Emp 与实体类之间的映射，代码如下所示：

```
public class Emp {  
    private int empNo;  
    private String ename;  
    private String job;  
    private int mgr;  
    private String hiredate;  
    private double sal;  
    private double comm;  
    private int deptno;  
    public Emp() {  
        super();  
    }  
    public Emp(int empNo, String ename, String job, int mgr, String hiredate,  
              double sal, double comm, int deptno) {  
        super();  
        this.empNo = empNo;  
        this.ename = ename;  
        this.job = job;  
        this.mgr = mgr;  
        this.hiredate = hiredate;  
        this.sal = sal;  
        this.comm = comm;  
        this.deptno = deptno;  
    }  
    public int getEmpNo() {  
        return empNo;  
    }  
    public void setEmpNo(int empNo) {  
        this.empNo = empNo;  
    }  
    public String getEname() {  
        return ename;  
    }  
    public void setEname(String ename) {  
        this.ename = ename;  
    }  
    public String getJob() {  
        return job;  
    }  
    public void setJob(String job) {  
        this.job = job;  
    }  
    public int getMgr() {  
        return mgr;  
    }  
    public void setMgr(int mgr) {  
        this.mgr = mgr;  
    }  
    public String getHiredate() {  
        return hiredate;  
    }  
    public void setHiredate(String hiredate) {  
        this.hiredate = hiredate;  
    }  
    public double getSal() {  
        return sal;  
    }  
    public void setSal(double sal) {  
        this.sal = sal;  
    }  
}
```

```
public double getComm() {
    return comm;
}
public void setComm(double comm) {
    this.comm = comm;
}
public int getDeptno() {
    return deptno;
}
public void setDeptno(int deptno) {
    this.deptno = deptno;
}
```

步骤二：在 EmpDAO 类中添加 add 方法

在 EmpDAO 类中添加 add 方法，用于实现向数据库的 Emp 表中添加数据，代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        ...
    }

    public void add(Emp emp) {
    }
}
```

步骤三：拼写 insert 语句

在 add 方法中定义 insert 语句，代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        ...
    }

    public void add(Emp emp) {
```

```

String sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
values("
    + emp.getEmpNo()
    + ", "
    + " "
    + emp.getEname()
    + ", "
    + " "
    + emp.getJob()
    + ", "
    + emp.getMgr()
    + ","
    + "to_date('"
    + emp.getHiredate()
    + "','yyyy-mm-dd'), "
    + emp.getSal()
    + ","
    + emp.getComm() + ", " + emp.getDeptno() + ")";
}

}

```

步骤四：执行插入语句

首先创建数据库连接；然后通过连接创建 Statement 对象；最后使用 Statement 对象的 updateExecute 方法，执行插入语句并处理异常，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        dao.findAll();
    }

    public void findAll() {
        ...
    }

    public void add(Emp emp) {

        Connection con = null;
        Statement stmt = null;
        int flag = -1;

        String sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,
deptno) values("
            + emp.getEmpNo()
            + ", "
            + " "
            + emp.getEname()
            + ", "
            + " "
            + emp.getJob()
            + ", "
            + emp.getMgr()
            + ","
            + "to_date('"

```

```

        + emp.getHiredate()
        + "' , 'yyyy-mm-dd' ), "
        + emp.getSal()
        + ", "
        + emp.getComm() + ", " + emp.getDeptno() + ")";

    try {
        con = ConnectionSource.getConnection();
        stmt = con.createStatement();

        flag = stmt.executeUpdate(sql);
        if (flag > 0) {
            System.out.println("新增记录成功!");
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常!");
        throw new RuntimeException(e);
    } finally {
        try {
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
}

```

步骤五：测试插入数据是否成功

在 EmpDAO 类的 main 方法中，调用 add 方法，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        //dao.findAll();

        // 2.insert

        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                         500.00, 10);
        dao.add(emp);

    }

    public void findAll() {
        ...
    }
}

```

```
public void add(Emp emp) {  
    Connection con = null;  
    Statement stmt = null;  
    int flag = -1;  
    String sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,  
deptno) values ("  
        + emp.getEmpNo()  
        + ", "  
        + " '"  
        + emp.getEname()  
        + "' , "  
        + " '"  
        + emp.getJob()  
        + "' , "  
        + emp.getMgr()  
        + ", "  
        + "to_date('"  
        + emp.getHiredate()  
        + "','yyyy-mm-dd'), "  
        + emp.getSal()  
        + ", "  
        + emp.getComm() + ", " + emp.getDeptno() + ")";  
  
    try {  
        con = ConnectionSource.getConnection();  
        stmt = con.createStatement();  
  
        flag = stmt.executeUpdate(sql);  
        if (flag > 0) {  
            System.out.println("新增记录成功!");  
        }  
    } catch (SQLException e) {  
        System.out.println("数据库访问异常!");  
        throw new RuntimeException(e);  
    } finally {  
        try {  
            if (stmt != null) {  
                stmt.close();  
            }  
            if (con != null) {  
                con.close();  
            }  
        } catch (SQLException e) {  
            System.out.println("释放资源发生异常");  
        }  
    }  
}
```

运行上述代码，数据库 Emp 表中记录如图-1 所示：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980/12/17	800.00		20
7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
7839	KING	PRESIDENT		1981/11/17	5000.00		10
7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
7900	JAMES	CLERK	7698	1981/12/3	950.00		30
7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
7934	MILLER	CLERK	7782	1982/1/23	1300.00		10
1001	rose	Analyst	7901	2014/5/1	3000.00	500.00	10

图-1

通过上述运行结果，会发现在数据库 Emp 表中添加了一条职员 ID 为 1001 的记录。

步骤六：对 Emp 表中的数据执行更新

在 EmpDAO 类中 添加 update 方法，在方法实现将员工 ID 为 1001 的薪资更新为 4500，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        dao.add(emp);

    }

    public void findAll() {
        ...
    }

    public void add(Emp emp) {
        ...
    }

    public void update(Emp emp) {
        Connection con = null;
        Statement stmt = null;
        int flag = -1;
        String sql = "update emp set sal = " + emp.getSal() + "," + " comm = "
                    + emp.getComm() + " where empno = " + emp.getEmpNo();

        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();

```

```
        flag = stmt.executeUpdate(sql);
        if (flag > 0) {
            System.out.println("更新记录成功!");
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常!");
        throw new RuntimeException(e);
    } finally {
        try {
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}

}
```

步骤七：测试 update 方法更新数据是否成功

在 EmpDAO 的 main 方法中添加代码，调用 update 方法，代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        //dao.findAll();

        // 2.insert

        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 4500.00,
                        500.00, 10);

        // dao.add(emp);

        // 3.update

        emp.setSal(4500.00);
        dao.update(emp);

    }

    public void findAll() {
        ...
    }

    public void add(Emp emp) {
        ...
    }
}
```

```
public void update(Emp emp) {
    ...
}
```

运行上述代码，数据库 Emp 表中记录如图-2 所示：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980/12/17	800.00		20
7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
7839	KING	PRESIDENT		1981/11/17	5000.00		10
7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
7900	JAMES	CLERK	7698	1981/12/3	950.00		30
7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
7934	MILLER	CLERK	7782	1982/1/23	1300.00		10
1001	rose	Analyst	7901	2014/5/1	4500.00	500.00	10

图-2

从运行结果可以看出，职员 ID 为 1001 的薪资被更新为 4500。

• 完整代码

本案例中，类 Emp 完整代码如下所示：

```
public class Emp {
    private int empNo;
    private String ename;
    private String job;
    private int mgr;
    private String hiredate;
    private double sal;
    private double comm;
    private int deptno;
    public Emp() {
        super();
    }
    public Emp(int empNo, String ename, String job, int mgr, String hiredate,
              double sal, double comm, int deptno) {
        super();
        this.empNo = empNo;
        this.ename = ename;
        this.job = job;
        this.mgr = mgr;
        this.hiredate = hiredate;
        this.sal = sal;
        this.comm = comm;
        this.deptno = deptno;
    }
    public int getEmpNo() {
        return empNo;
    }
    public void setEmpNo(int empNo) {
        this.empNo = empNo;
    }
    public String getEname() {

```

```
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    public int getMgr() {
        return mgr;
    }
    public void setMgr(int mgr) {
        this.mgr = mgr;
    }
    public String getHiredate() {
        return hiredate;
    }
    public void setHiredate(String hiredate) {
        this.hiredate = hiredate;
    }
    public double getSal() {
        return sal;
    }
    public void setSal(double sal) {
        this.sal = sal;
    }
    public double getComm() {
        return comm;
    }
    public void setComm(double comm) {
        this.comm = comm;
    }
    public int getDeptno() {
        return deptno;
    }
    public void setDeptno(int deptno) {
        this.deptno = deptno;
    }
}
```

EmpDAO 类的完整代码如下所示：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 4500.00,
                        500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        dao.update(emp);
    }
}
```

```

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        con = ConnectionSource.getConnection();
        stmt = con.createStatement();
        rs = stmt
            .executeQuery("select empno, ename, sal, hiredate from emp");
        while (rs.next()) {
            System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + "," + ","
                + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常!");
        throw new RuntimeException(e);
    } finally {
        try {
            if(rs!=null){
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}

public void add(Emp emp) {
    Connection con = null;
    Statement stmt = null;
    int flag = -1;
    String sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,
deptno) values("
        + emp.getEmpNo()
        + ", "
        + " '"
        + emp.getEname()
        + "', "
        + " '"
        + emp.getJob()
        + "', "
        + emp.getMgr()
        + ", "
        + "to date('"
        + emp.getHiredate()
        + "','yyyy-mm-dd'), "
        + emp.getSal()
        + ", "
        + emp.getComm() + ", " + emp.getDeptno() + ")";
    try {
        con = ConnectionSource.getConnection();
        stmt = con.createStatement();

        flag = stmt.executeUpdate(sql);
        if (flag > 0) {
            System.out.println("新增记录成功!");
        }
    }
}

```

```

        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源发生异常");
            }
        }
    }

    public void update(Emp emp) {
        Connection con = null;
        Statement stmt = null;
        int flag = -1;
        String sql = "update emp set sal = " + emp.getSal() + "," + " comm = "
            + emp.getComm() + " where empno = " + emp.getEmpNo();

        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();

            flag = stmt.executeUpdate(sql);
            if (flag > 0) {
                System.out.println("更新记录成功！");
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源发生异常");
            }
        }
    }
}

```

db.properties 文件的内容与之前案例一样没有变化，该文件完整内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量

```

```
dataSource.setMaxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.setMaxWait=1000
```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }

    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                    "day01/v4/db.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            String driveClassName =
                dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize=dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
            dataSource.setPassword(password);

            // 初始化连接数
            if (initialSize != null)
                dataSource.setInitialSize(Integer.parseInt(initialSize));

            // 最小空闲连接
            if (minIdle != null)
                dataSource.setMinIdle(Integer.parseInt(minIdle));

            // 最大空闲连接
            if (maxIdle != null)
                dataSource.setMaxIdle(Integer.parseInt(maxIdle));

            // 超时回收时间(以毫秒为单位)
            if (maxWait != null)
                dataSource.setMaxWait(Long.parseLong(maxWait));
        }
    }
}
```

```

// 最大连接数
if (maxActive != null) {
    if (!maxActive.trim().equals("0"))
        dataSource.setMaxActive(Integer.parseInt(maxActive));
}
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("创建连接池失败!请检查设置!!!!");
}
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}

```

2. 用户名密码验证功能

- 问题

Oracle 数据库中用户表 users 的表结构如表 - 1 所示：

表 - 1 用户表 users 信息

字段名	类型	描述
id	NUMBER(4,0)	用户 ID
username	VARCHAR2(30)	用户登录名
password	VARCHAR2(30)	用户密码

用户表 users 中的示例数据，如图 - 3 所示：

ID	USERNAME	PASSWORD
1	tarena	tarena123
2	syl	syl001

图 - 3

本案例详细要求如下：

- 使用 Statement 实现用户名和密码的验证功能，并测试用户名为 “tarena”、密码为 “tarena123” 以及用户名为 “goodman”、密码为 “a' OR 'b'='b” 是否能登录成功。
- 使用 PreparedStatement 实现用户名和密码的验证功能，并测试用户名为 “goodman”、密码为 “a' OR 'b'='b” 是否能登录成功。

- 方案

实现本案例的方案如下所示：

1. 使用 Statement 实现用户名和密码的验证功能。示例代码如下：

```
sql = "select * from users where username = '" + username
      + "' and password = '" + password+"'";
System.out.println(sql);
con = ConnectionSource.getConnection();
stmt = con.createStatement();
rs = stmt.executeQuery(sql);
```

上述代码存在 SQL 注入的问题，可以使用 PreparedStatement 来解决 SQL 注入的问题。

2. 使用 PreparedStatement 实现用户名和密码的验证功能。PreparedStatement 是 Statement 的子类，表示预编译的 SQL 语句的对象。在使用 PreparedStatement 对象执行 SQL 命令时，命令被数据库编译和解析，并放入命令缓冲区。缓冲区中的预编译 SQL 命令可以重复使用。示例代码如下所示：

```
sql = "select * from users where username = ? and password = ?";
con = ConnectionSource.getConnection();
stmt = con.prepareStatement(sql);
stmt.setString(1, username);
stmt.setString(2, password);
rs = stmt.executeQuery();
```

使用 PreparedStatement 来执行 SQL 语句。在 SQL 语句中有 2 个问号，在代码中要给它们分别设置值，规则是：从左到右，对应 1, 2, ...。

3. 关于 SQL 注入。对于 JDBC 而言，SQL 注入攻击只对 Statement 有效，对 PreparedStatement 是无效的，这是因为 PreparedStatement 不允许在插入时改变查询的逻辑结构。如果有一条 SQL 语句为：

```
"select * from 表 where 用户名 = '用户名'"
```

Statement 的 SQL 语句是这样写的：

```
"select * from 表 where 用户名 = '" + 变量值 + "'"
```

而 PreparedStatement 的 SQL 语句是这样写的：

```
"select * from 表 where 用户名 = ?"
```

这样输入 "aa' or '1' = '1"，使用 Statement 执行的 SQL 语句为：

```
"select * from 表 where 用户名 = 'aa' or '1' = '1'"
```

而使用 PreparedStatement 是将 "aa' or '1' = '1" 作为一个字符串赋值给问号 "?"，使其作为"用户名"字段的对应值，这样来防止 SQL 注入。

实现机制不同，注入只对 SQL 语句的准备（编译）过程有破坏作用，而 PreparedStatement 已经准备好了，执行阶段只是把输入串作为数据处理，不再需要对 SQL 语句进行解析、准备，因此也就避免了 SQL 注入问题。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：创建 users 表，并插入示例数据

在 Oracle 数据库中，创建 users 表，并插入如图-3 所示的示例数据，SQL 语句如下所示：

```
create table users(
    id number(4),
    username varchar2(30),
    password varchar2(30)
);

insert into users(id,username,password)values(1,'tarena','tarena123');
insert into users(id,username,password)values(2,'syl','syl001');
```

步骤二：使用 Statement 实现验证用户名密码是否存在

新建 UserDAO 类，在该类中添加 login1 方法，在该方法中使用 Statement 实现验证用户名密码是否存在的方法，代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UserDAO {
    public static void main(String[] args) {

        public void login1(String username, String password) {
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;
            String sql = null;

            try {
                sql = "select * from users where username = '" + username
                    + "' and password = '" + password + "'";
                System.out.println(sql);
                con = ConnectionSource.getConnection();
                stmt = con.createStatement();
                rs = stmt.executeQuery(sql);

                if (rs.next()) {
                    System.out.println("登录成功！");
                } else {
                    System.out.println("登录失败！");
                }
            } catch (SQLException e) {
                System.out.println("数据库访问异常！");
            }
        }
    }
}
```

```
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
```

步骤三：使用用户名为“tarena”、密码为“tarena123”测试是否能成功登录

在 UserDAO 的 main 方法中，使用用户名为“tarena”、密码为“tarena123”作为 login1 方法的参数，测试是否能成功登录，代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UserDAO {
    public static void main(String[] args) {
        // login

        UserDAO dao = new UserDAO();
        dao.login1("tarena", "tarena123");

    }

    public void login1(String username, String password) {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            sql = "select * from users where username = '" + username
                  + "' and password = '" + password + "'";
            System.out.println(sql);
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);

            if (rs.next()) {
                System.out.println("登录成功！");
            } else {
                System.out.println("登录失败！");
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {

```

```
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
```

运行 UserDao 类，控制台输出结果如下：

```
select * from users where username = 'tarena' and password = 'tarena123'
登录成功！
```

从输出结果可以看出，使用用户名为“tarena”、密码为“tarena123”可以登录成功。由于用户名为“tarena”、密码为“tarena123”在数据库中是存在的数据，登录成功符合预期结果。另外，从输出的 SQL 语句可以看出，最终将变量信息替换为实际传入的参数信息了。

步骤四：测试 login1 方法

在 UserDao 的 main 方法中，使用用户名为“goodman”、密码为“a' OR 'b'='b”作为 login1 方法的参数，测试是否能成功登录，代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UserDao {
    public static void main(String[] args) {
        UserDao dao = new UserDao();
        //dao.login1("tarena", "tarena123");

        dao.login1("tarena", "a' OR 'b'='b");

    }
    public void login1(String username, String password) {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            sql = "select * from users where username = '" + username
                  + "' and password = '" + password + "'";
            System.out.println(sql);
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
        }
```

```

        rs = stmt.executeQuery(sql);

        if (rs.next()) {
            System.out.println("登录成功！");
        } else {
            System.out.println("登录失败！");
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
}

```

运行 UserDao 类，控制台输出结果如下：

```

select * from users where username = 'tarena' and password = 'a' OR 'b'='b'
登录成功！

```

从输出结果可以看出，使用用户名为 “goodman”、密码为 “a’ OR ‘b’='b” 可以登录成功。但是用户名为 “goodman”、密码为 “a’ OR ‘b’='b” 在数据库中是不存在的数据，登录成功不符合预期结果。问题出在哪里？

问题出在 SQL 语句上。查看上述控制台输出结果，可以看出 SQL 语句的 where 子句中使用 or 关键字连接两个表达式，即 or 关键字后边的表达式如果返回 true，那么整个 where 条件的结果将是 true。而 or 关键字后边的表达式为 “‘b’='b’”，该表达式的返回结果永远为 true，因此，where 条件的返回结果一定为 true。这种现象在编程中称为 SQL 注入，是应该避免的编程方式。可以使用 PreparedStatement 来防止 SQL 注入。

步骤五： 使用 PreparedStatement 实现验证用户名密码功能

在 UserDao 类中添加 login 方法，在该方法中使用 PreparedStatement 执行 SQL 语句，来实现验证用户名密码功能，代码如下所示：

```

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

public class UserDao {
    public static void main(String[] args) {

```

```

        // login
        UserDAO dao = new UserDAO();
        //dao.login1("tarena", "tarena123");
        dao.login1("tarena", "a' OR 'b'='b");
    }

    public void login1(String username, String password) {
        ...
    }

    public void login(String username, String password) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            sql = "select * from users where username = ? and password = ?";
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql);
            stmt.setString(1, username);
            stmt.setString(2, password);
            rs = stmt.executeQuery();

            if (rs.next()) {
                System.out.println("登录成功！");
            } else {
                System.out.println("登录失败！");
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源发生异常");
            }
        }
    }
}

```

步骤六： 测试 login 方法

在 UserDAO 的 main 方法中，使用用户名为 “goodman”、密码为 “a' OR 'b'='b” 作为 login 方法的参数，测试是否能成功登录，代码如下所示：

```

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

```

```

import java.sql.SQLException;

public class UserDAO {
    public static void main(String[] args) {
        // login
        UserDAO dao = new UserDAO();
        //dao.login1("tarena", "tarena123");
        //dao.login1("tarena", "a' OR 'b'='b");
        dao.login("tarena", "a' OR 'b'='b");
    }

    public void login1(String username, String password) {
        ...
    }

    public void login(String username, String password) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            sql = "select * from users where username = ? and password = ?";
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql);
            stmt.setString(1, username);
            stmt.setString(2, password);
            rs = stmt.executeQuery();

            if (rs.next()) {
                System.out.println("登录成功！");
            } else {
                System.out.println("登录失败！");
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源发生异常");
            }
        }
    }
}

```

运行 EmpDAO 类，控制台输出结果如下所示：

登录失败！

从输出结果可以看出，登录失败。用户名为 “goodman”、密码为 “a' OR 'b'='b” 的数据在数据库 users 表中是不存在的数据，登录失败符合测试的预期，有效的防止了 SQL

注入的问题。

- **完整代码**

本案例中，创建 users 表，并插入示例数据，SQL 语句如下所示：

```
create table users(
    id number(4),
    username varchar2(30),
    password varchar2(30)
);

insert into users(id,username,password)values(1,'tarena','tarena123');
insert into users(id,username,password)values(2,'syl','syl001');
```

UserDAO 类的完整代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UserDAO {
    public static void main(String[] args) {
        // login
        UserDAO dao = new UserDAO();
        //dao.login1("tarena", "tarena123");
        //dao.login1("tarena", "a' OR 'b'='b");
        dao.login("tarena", "a' OR 'b'='b");
    }

    public void login1(String username, String password) {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            sql = "select * from users where username = '" + username
                  + "' and password = '" + password + "'";
            System.out.println(sql);
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);

            if (rs.next()) {
                System.out.println("登录成功！");
            } else {
                System.out.println("登录失败！");
            }
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
            }
        }
    }
}
```

```

        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源发生异常");
    }
}

public void login(String username, String password) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    String sql = null;

    try {
        sql = "select * from users where username = ? and password = ?";
        con = ConnectionSource.getConnection();
        stmt = con.prepareStatement(sql);
        stmt.setString(1, username);
        stmt.setString(2, password);
        rs = stmt.executeQuery();

        if (rs.next()) {
            System.out.println("登录成功！");
        } else {
            System.out.println("登录失败！");
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
}

```

db.properties 文件的内容与之前案例一样没有变化，该文件完整内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50

```

```
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.setMaxWait=1000
```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }
    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                    "day01/v4/db.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            String                     driveClassName      =
dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
            dataSource.setPassword(password);

            // 初始化连接数
            if (initialSize != null)
                dataSource.setInitialSize(Integer.parseInt(initialSize));

            // 最小空闲连接
            if (minIdle != null)
                dataSource.setMinIdle(Integer.parseInt(minIdle));

            // 最大空闲连接
            if (maxIdle != null)
                dataSource.setMaxIdle(Integer.parseInt(maxIdle));

            // 超时回收时间(以毫秒为单位)
            if (maxWait != null)
                dataSource.setMaxWait(Long.parseLong(maxWait));

            // 最大连接数
        }
    }
}
```

```
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}
```

课后作业

1. 更新和插入 Dept 数据

本案例要求使用 JDBC 向 Dept 表中插入和更新数据，详细要求如下：

1. 向 Dept 表中插入一条记录。其中为列 deptno、dname、loc 插入的数据分别为 50、"developer"、 "Beijing"。
2. 更新部门 ID 为 50 的部门所在地改为 "ShangHai"。

2. 查询指定 Emp 的工资

查询指定姓名的员工的工资。详细要求如下：

1. 使用 Statement 实现查询指定姓名的员工的工资，并测试姓名为 "CLARK" 的员工工资，以及姓名为 "a' OR 'b'='b" 的员工工资。
2. 使用 PreparedStatement 实现查询指定姓名的员工的工资，并测试姓名为 "CLARK" 的员工工资，以及姓名为 "a' OR 'b'='b" 的员工工资。

Java 数据库编程

Unit03

知识体系.....**Page 76**

JDBC 高级编程	事务处理	事务简介
		JDBC 事务 API
		JDBC 标准事务编程模式
	批量更新	批量更新的优势
		批量更新 API
		防止 OutOfMemory
	返回自动主键	关联数据插入
		通过序列产生主键 (Oracle)
		JDBC 返回自动主键 API
	分页查询	JDBC 实现 Oracle 分页查询
		JDBC 实现 MySQL 分页查询
DAO	什么是 DAO	DAO 封装对数据的访问
		实体对象
	编写 DAO	查询方法
		更新方法
		异常处理机制

经典案例.....**Page 84**

实现账户转账操作	JDBC 事务 API
	JDBC 标准事务编程模式
批量插入 Emp 数据	批量更新 API
	防止 OutOfMemory
插入 Dept 及其关联的 Emp 信息	关联数据插入
	通过序列产生主键 (Oracle)
	JDBC 返回自动主键 API
实现 Emp 数据的分页查询 (Oracle 和 MySQL)	JDBC 实现 Oracle 分页查询
	JDBC 实现 MySQL 分页查询
完成 NetCTOSS 项目中，账务账号的 DAO 设计及实现	DAO 封装对数据的访问
	实体对象
	查询方法

	更新方法
	异常处理机制

课后作业.....Page 158

达内IT培训集团

1. JDBC 高级编程

1.1. 事务处理

1.1.1. 【事务处理】事务简介

事务简介

- 事务（ Transaction ）：数据库中保证交易可靠的机制
- JDBC支持数据库中的事务概念
- 在JDBC中，事务默认是自动提交的

知识讲解

+

The diagram illustrates a database transaction flow:

- The process starts with a **start** node.
- An annotation box indicates: **A, B账户均存在** (Accounts A and B exist) and **A要转账1000给B** (Account A transfers 1000 to Account B).
- The transaction begins with a **A账户扣除1000元** (Account A deducts 1000 yuan) operation.
- This is followed by a **B账户增加1000元** (Account B adds 1000 yuan) operation.
- The transaction concludes with an **end** node.

A callout bubble on the right states: **必须在一个事务中完成** (Must be completed in a single transaction).

知识讲解

+

Tarena
达内科技

事务简介(续2)

• 事务特性ACID：

- 原子性（Atomicity）：事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行
- 一致性（Consistency）：事务在完成时，必须使所有的数据都保持一致状态
- 隔离性（Isolation）：由并发事务所作的修改必须与任何其它并发事务所作的修改隔离
- 持久性（Durability）：事务完成之后，它对于系统的影响是永久性的

• 事务是数据库的概念，JDBC支持事务，本质还是在数据库中实现的

1.1.2. 【事务处理】JDBC 事务 API

Technology
Tarena
达内科技

JDBC事务API

- 相关API :

- Connection.getAutoCommit():获得当前事务的提交方式，
默认为true
- Connection.setAutoCommit():设置事务的提交属性，参
数是
 - true : 自动提交；false : 不自动提交**
- Connection.commit():提交事务
- Connection.rollback():回滚事务

知识讲解

1.1.3. 【事务处理】 JDBC 标准事务编程模式

Technology
Tarena
达内科技

JDBC标准事务编程模式

try{

// 1.定义用于在事务中执行的SQL语句

```
String sql1 = "update account set amount = amount - " + amount + "  
where id = " + from + "";
```

```
String sql2 = "update account set amount = amount + " + amount + "  
where id = " + to + "";
```

autoCommit = con.getAutoCommit(); // 2.获得自动提交状态

con.setAutoCommit(false); // 3.关闭自动提交

stmt.executeUpdate(sql1); // 4.执行SQL语句

stmt.executeUpdate(sql2);

con.commit(); // 5.提交

con.setAutoCommit(autoCommit); // 6.将自动提交功能恢复到原来的状态

//其他语句;

}catch(SQLException e){

conn.rollback();//异常时回滚 }

知识讲解

1.2. 批量更新

1.2.1. 【批量更新】批量更新的优势

Tarena
达内科技

批量更新的优势

- 批处理：发送到数据库作为一个单元执行的一组更新语句
- 批处理降低了应用程序和数据库之间的网络调用
- 相比单个SQL语句的处理，批处理更为有效

知识讲解

1.2.2. 【批量更新】批量更新 API

 Tarena
达内科技

批量更新API

- addBatch(String sql)
 - Statement类的方法, 可以将多条sql语句添加Statement对象的SQL语句列表中
- addBatch()
 - PreparedStatement类的方法, 可以将多条预编译的sql语句添加到PreparedStatement对象的SQL语句列表中
- executeBatch()
 - 把Statement对象或PreparedStatement对象语句列表中的所有SQL语句发送给数据库进行处理
- clearBatch()
 - 清空当前SQL语句列表

知识讲解

1.2.3. 【批量更新】防止 OutOfMemory

Tarena
达内科技

防止OutOfMemory

- 如果PreparedStatement对象中的SQL列表包含过多的待处理SQL语句，可能会产生OutOfMemory错误

- 及时处理SQL语句列表

```
for (int i = 0; i < 1000; i++) {
```

```
    sql = "insert into emp(empno, ename)
```

```
        values(emp_seq.nextval, 'name" + i + "");
```

```
    stmt.addBatch(sql); //将SQL语句加入到Batch中
```

```
    if (i % 500 == 0) {
```

```
        stmt.executeBatch(); //及时处理
```

```
        stmt.clearBatch(); //清空列表
```

```
}
```

```
}
```

```
//最后一次列表不足500条，处理
```

```
stmt.executeBatch();
```

1.3. 返回自动主键

1.3.1. 【返回自动主键】关联数据插入

知识讲解

```
graph TD; A[事务启动] --> B[插入主表记录]; B --> C[获得主表主键值]; C --> D["插入从表记录，上一步获得的主表主键值，作为从表外键值"]; D --> E[事务提交]
```

The diagram illustrates the process of inserting associated data. It consists of five rectangular boxes connected by downward arrows. The first box is labeled '事务启动' (Transaction Start). The second box is labeled '插入主表记录' (Insert into Main Table Record). The third box is labeled '获得主表主键值' (Obtain Main Table Primary Key Value). The fourth box contains the instruction '插入从表记录，上一步获得的主表主键值，作为从表外键值' (Insert into Sub-table Record, use the primary key value obtained from the previous step as the foreign key value). The fifth box is labeled '事务提交' (Transaction Commit).

1.3.2. 【返回自动主键】通过序列产生主键（Oracle）

Tarena
达内科技

通过序列产生主键 (Oracle)

- 在oracle中，建议主键通过序列获得：

```
String sql = "insert into dept (deptno, dname, loc)  
values(dept_seq.nextval,?,?)";
```

- 如果单表操作，不需要返回刚刚插入的主键值

- 如果关联操作，需要将刚刚插入的主键值返回

知识讲解

1.3.3. 【返回自动主键】JDBC 返回自动主键 API

Tarena
达内科技

JDBC返回自动主键API (续1)

• 方法二

利用PreparedStatement的getGeneratedKeys方法获取

自增类型的数据，性能良好，只要一次SQL交互

```
sql = "insert into dept (deptno, dname, loc)
```

```
values(dept_seq.nextval,?,?)";
```

```
stmt = con.prepareStatement(sql, new String[] { "deptno" });
```

```
stmt.setString(1, "Research");
```

```
stmt.setString(2, "beijing");
```

```
stmt.executeUpdate();
```

```
rs = stmt.getGeneratedKeys();
```

```
rs.next();
```

```
int deptno = rs.getInt(1);
```

stmt第二个参数是
GeneratedKeys
的字段名列表，字
符串数组

获得主键值
所在的rs

从rs中取出主键值，
从表使用

1.4. 分页查询

1.4.1. 【分页查询】 JDBC 实现 Oracle 分页查询

JDBC实现Oracle分页查询

知识讲解

+

Tarena
达内科技

- 利用Oracle的rownum，传入SQL语句，以获得分页结果集：
`String sql = "select * from (select rownum rn, empno, ename, job,mgr, hiredate, sal, comm, deptno from (select * from emp order by empno)) where rn between ? and ?";`
- 计算分页结果集的起点和终点，替代SQL中的占位符：
`int begin = (page - 1) * pageSize + 1;`
`int end = begin + pageSize - 1;`

返回第几页

每页多少条

JDBC实现Oracle分页查询（续1）

知识讲解

+

Tarena
达内科技

```
stmt = con.prepareStatement(sql);
stmt.setInt(1, begin);
stmt.setInt(2, end);
rs = stmt.executeQuery();
```

- 每次只向数据库请求一页的数据量
- 内存压力小适合大数据量数据表

JDBC实现Oracle分页查询（续2）

知识讲解

+

Tarena
达内科技

- 另一种分页策略介绍：基于缓存的分页技术（假分页）
 - 一次性把数据全部取出来放在缓存中，根据用户要看的页数(page)和每页记录数(pageSize)，计算把哪些数据输出显示，将可滚动结果集的指针移动到指定位置
 - 只访问数据库一次，第一次取数比较慢，以后每页都从缓存中取，比较快
 - 比较适合小数据量，如果数据量大，对内存压力较大

1.4.2. 【分页查询】JDBC 实现 MySQL 分页查询

Technology
Tarena
达内科技

JDBC实现MySQL分页查询

- 不同数据库的SQL区别

- MySQL的实现方式：

```
select * from t limit begin , pageSize;
```

知识讲解

- 在Java程序中，MySQL和Oracle分页的方式，仅限于SQL语句不同

2. DAO

2.1. 什么是 DAO

2.1.1. 【什么是 DAO】DAO 封装对数据的访问

Tarena
达内科技

DAO封装对数据的访问

- DAO (Data Access Object) 数据访问对象
- 建立在数据库和业务层之间，封装所有对数据库的访问
- 目的：数据访问逻辑和业务逻辑分开

知识讲解

The diagram illustrates the DAO pattern architecture. It features three main components arranged horizontally: a rectangular box labeled '业务层' (Business Layer), a rectangular box labeled 'DAO', and a vertical cylinder labeled '数据库' (Database). A horizontal arrow points from the '业务层' box to the 'DAO' box, and another horizontal arrow points from the 'DAO' box to the '数据库' cylinder, representing the flow of data access requests.

Tarena
达内科技

DAO封装对数据的访问（续1）

- 为了建立一个健壮的Java应用，需将所有对数据源的访问操作抽象封装在一个公共API中，包括：
 - 建立一个接口，接口中定义了应用程序中将会用到的所有事务方法
 - 建立接口的实现类，实现接口对应的所有方法，和数据库直接交互
- 在应用程序中，当需要和数据源交互时则使用DAO接口，不涉及任何数据库的具体操作

知识点讲解

DAO封装对数据的访问（续2）

- DAO通常包括：
 1. 一个DAO工厂类；
 2. 一个DAO接口；
 3. 一个实现DAO接口的具体类；
 4. 数据传递对象（实体对象或值对象）.

知识讲解

+

2.1.2. 【什么是 DAO】实体对象

实体对象

- DAO层需要定义对数据库中表的访问
- 对象关系映射(ORM : Object/Relation Mapping)描述对象和数据表之间的映射，将Java程序中的对象对应到关系数据库的表中
 - 表和类对应
 - 表中的字段和类的属性对应
 - 记录和对象对应

知识讲解

+

2.2. 编写 DAO

2.2.1. 【编写 DAO】查询方法

查询方法

```
public Account findById(Integer id) throws SQLException {  
    Connection conn = getConnection();  
    String sql = SELECT_BY_ID; //预先定义好的SQL查询语句  
    PreparedStatement ps = conn.prepareStatement(sql);  
    ps.setInt(1, id); //传入参数  
    ResultSet rs = ps.executeQuery();  
    Account account = null;  
    while(rs.next()) { //处理结果集  
        account = new Account();  
        account.setId(rs.getInt("ACCOUNT_ID"));  
        //设置account对象所有的属性，略  
    }  
    return account;  
}
```

2.2.2. 【编写 DAO】更新方法

Technology
Tarena
达内科技

更新方法

```
public boolean update(Account account) throws  
SQLException {  
    Connection conn = getConnection();  
    String sql = UPDATE_STATUS;//预先定义好的SQL语句  
    PreparedStatement ps = conn.prepareStatement(sql);  
    ps.setInt(1,account.getId());//传入参数  
    ps.setString(2, account.getStatus());  
    int flag= ps.executeUpdate();  
    return (flag>0 ? true : false);  
}
```

知识讲解

2.2.3. 【编写 DAO】异常处理机制

Technology
Tarena
达内科技

异常处理机制

• 多层系统的异常处理原则：

- 谁抛出的异常，谁捕捉处理，因为只有异常抛出者，知道怎样捕捉处理异常；
- 尽量在当前层中捕捉处理抛出的异常，尽量不要抛出到上层接口；
- 尽量在每层中封装每层的异常类，这样可准确定位异常抛出的层；
- 如异常无法捕捉处理，则向上层接口抛出，直至抛给JVM
- 应尽量避免将异常抛给 JVM

知识讲解

经典案例

1. 实现账户转账操作

- 问题

使用 JDBC 连接数据库，实现账号转账业务。从 A 账户转账给 B 账户 500 元钱。

- 方案

本案例中，要实现转账业务，需要执行两次更新操作，一是更新 A 账户的金额为在原有金额基础上减去 500 元；二是更新 B 账户的金额为在原有金额的基础上增加 500 元。这两次更新操作要么全部成功、要么全部失败，来表示转账的成功或失败。如果 A 账户的金额更新为在原有金额基础上减去 500 元，而 B 账户却没有更新为在原有金额的基础上增加 500 元，这样就造成了数据的不一致。我们可以使用事务来控制两次更新操作要么全部成功、要么全部失败。

本案例中，可以使用事务将两次更新操作封装成一个逻辑单元，要么完全执行，要么完全不执行，保证了数据的完整性。

下列方法可实现 JDBC 事务的基本操作：

- Connection.setAutoCommit(boolean) 当参数为 false 时，关闭自动提交
- Connection.commit() 提交事务
- Connection.rollback() 回滚事务

使用 JDBC 控制事务的核心代码如下：

```
try {
    con = ConnectionSource.getConnection();
    stmt = con.createStatement();

    // 更新数据的 SQL 语句
    String sql1 = "update account set amount = amount - " + amount
        + " where id = '" + from + "'";
    String sql2 = "update account set amount = amount + " + amount
        + " where id = '" + to + "'";

    // 关闭自动提交
    con.setAutoCommit(false);
    // 执行 SQL 语句
    stmt.executeUpdate(sql1);
    stmt.executeUpdate(sql2);
    // 提交
    con.commit();
} catch (SQLException e) {
    try {
        con.rollback();
    } catch (SQLException e1) {
        System.out.println("回滚事务异常！");
        throw new RuntimeException(e1);
    }
}
```

```
        }
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    }
}
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：创建 Account 表，并插入测试数据

在 Oracle 数据库中，创建表并插入测试数据，SQL 语句如下所示：

```
create table account (
    id char(1),
    amount number(10,2)
);
insert into account values('A', 1000);
insert into account values('B', 2000);
commit;
```

步骤二：准备 JDBC 操作数据库的基本代码

首先，新建类 Trans，在该类中新建 transfer 方法，方法的声明如下所示：

```
public void transfer(String from, String to, double amount) {}
```

该方法表示从账户 from 转账给账户 to，转账金额为 amount。

然后，准备数据库连接 Connection 对象、操作 SQL 语句的 Statement 对象并进行异常的处理，代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class Trans {
    public void transfer(String from, String to, double amount) {
        Connection con = null;
        Statement stmt = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}
```

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

步骤三：实现转账功能

使用 Connection 的 setAutoCommit 方法、commit 方法以及 rollback 方法来控制事务，以确保转账功能正确实现，代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class Trans {
    public void transfer(String from, String to, double amount) {
        Connection con = null;
        Statement stmt = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();

            // 更新数据的 SQL 语句
            String sql1 = "update account set amount = amount - " + amount
                + " where id = '" + from + "'";
            String sql2 = "update account set amount = amount + " + amount
                + " where id = '" + to + "'";

            // 关闭自动提交
            con.setAutoCommit(false);
            // 执行 SQL 语句
            stmt.executeUpdate(sql1);
            stmt.executeUpdate(sql2);
            // 提交
            con.commit();

        } catch (SQLException e) {

            try {
                con.rollback();
            } catch (SQLException e1) {
                System.out.println("回滚事务异常！");
                throw new RuntimeException(e1);
            }

            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}
```

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

步骤四：测试

在 Trans 类的 main 方法中，调用 transfer 方法，代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class Trans {
    public void transfer(String from, String to, double amount) {
        Connection con = null;
        Statement stmt = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();

            // 插入数据的 SQL 语句
            String sql1 = "update account set amount = amount - " + amount
                + " where id = '" + from + "'";
            String sql2 = "update account set amount = amount + " + amount
                + " where id = '" + to + "'";

            // 关闭自动提交
            con.setAutoCommit(false);
            // 执行 SQL 语句
            stmt.executeUpdate(sql1);
            stmt.executeUpdate(sql2);
            // 提交
            con.commit();
        } catch (SQLException e) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                System.out.println("回滚事务异常！");
                throw new RuntimeException(e1);
            }
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}

Trans trans = new Trans();
trans.transfer("A", "B", 500);
```

```
}
```

运行 Trans 类，然后查看 Oracle 数据库中的 account 表，会发现 A 账户的金额减少了 500 元，B 账户的金额增加了 500 元。

- 完整代码

本案例中，类 Trans 的完整代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class Trans {
    public void transfer(String from, String to, double amount) {
        Connection con = null;
        Statement stmt = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();

            // 更新数据的 SQL 语句
            String sql1 = "update account set amount = amount - " + amount
                + " where id = '" + from + "'";
            String sql2 = "update account set amount = amount + " + amount
                + " where id = '" + to + "'";

            // 关闭自动提交
            con.setAutoCommit(false);
            // 执行 SQL 语句
            stmt.executeUpdate(sql1);
            stmt.executeUpdate(sql2);
            // 提交
            con.commit();
        } catch (SQLException e) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                System.out.println("回滚事务异常！");
                throw new RuntimeException(e1);
            }
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }

    public static void main(String args[]) {
        Trans trans = new Trans();
        trans.transfer("A", "B", 500);
    }
}
```

}

db.properties 文件的内容与之前案例一样没有变化，该文件完整内容如下所示：

```
jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 (6000 毫秒/1000 等于 60 秒 )-->
dataSource.maxWait=1000
```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }

    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                    "day01/v4/db.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            String driveClassName = dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
```

```

        dataSource.setPassword(password);

        // 初始化连接数
        if (initialSize != null)
            dataSource.setInitialSize(Integer.parseInt(initialSize));

        // 最小空闲连接
        if (minIdle != null)
            dataSource.setMinIdle(Integer.parseInt(minIdle));

        // 最大空闲连接
        if (maxIdle != null)
            dataSource.setMaxIdle(Integer.parseInt(maxIdle));

        // 超时回收时间(以毫秒为单位)
        if (maxWait != null)
            dataSource.setMaxWait(Long.parseLong(maxWait));

        // 最大连接数
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}
    
```

2. 批量插入 Emp 数据

- 问题

向 Emp 表中批量插入 100 条数据，需要插入数据的列为 empno、ename 以及 sal。这三个字段对应的数据分别为 empno 列的数据通过序列 emp_seq 自动生成、ename 列的数据为字符串 “name” + 循环次数 i 组成、sal 的数据由随机生成的 10000 以内的整数构成。

- 方案

每循环一次，向数据库插入一条数据，频繁的访问数据库，效率很低。

在 Java 中专门提供的批处理的 API。在对数据库频繁操作时，可以使用 JDBC 批处理方式提高程序的效率。批处理的主要特点如下：

- 使用同一 Connection 资源，一次发送多条 SQL 语句执行。
- 提高应用程序与 DB 之间的吞吐量，缩短 DB 的响应时间。

- 与逐条执行 SQL 的方式相比，需要处理的数据量越大，批处理的优势越明显。

Statement 和 PreparedStatement 都提供了 addBatch、executeBatch 方法，用于实现缓存 SQL 语句和批量执行。使用 Statement 实现批处理的核心代码如下：

```
for (int i = 0; i < 100; i++) {  
    // 插入数据的 SQL 语句  
    sql = "insert into emp(empno, ename, sal) values(" + "emp_seq.nextval, 'name" + i + "', "  
    + new Random().nextInt(10000) + ")";  
    // 将 SQL 语句加入到 Batch 中  
    stmt.addBatch(sql);  
}  
// 执行批处理  
stmt.executeBatch();
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：在 Oracle 数据库中创建序列 emp_seq

在 Oracle 数据库中创建序列名为 emp_seq，该序列的起始值为 1、步进为 1，SQL 语句如下所示：

```
create sequence emp_seq start with 1 increment by 1;
```

步骤二：准备 JDBC 操作数据库的基本代码

首先 新建类 Batch，在该类中新建 batchAdd 方法，然后 准备数据库连接 Connection 对象、操作 SQL 语句的 Statement 对象以及设置事务管理；最后进行异常的处理，代码如下所示：

```
import java.sql.Connection;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Random;  
  
public class Batch {  
  
    public void batchAdd() {  
        Connection con = null;  
        Statement stmt = null;  
        String sql = null;  
        try {  
            con = ConnectionSource.getConnection();  
            stmt = con.createStatement();  
            // 关闭自动提交  
            con.setAutoCommit(false);  
  
            // 提交  
            con.commit();  
        } catch (SQLException e) {  
            System.out.println("数据库访问异常！");  
            throw new RuntimeException(e);  
        }  
    }  
}
```

```

        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
    public static void main(String args[]) {
}
}

```

步骤三：批量向 Emp 表中插入数据

使用 Statement 的 addBatch 方法和 executeBatch 方法，批量向 Emp 表中插入数据，代码如下所示：

```

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;

public class Batch {

    public void batchAdd() {
        Connection con = null;
        Statement stmt = null;
        String sql = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            // 关闭自动提交
            con.setAutoCommit(false);

            for (int i = 0; i < 100; i++) {
                // 插入数据的 SQL 语句
                sql = "insert into emp(empno, ename, sal) values("
                    + "emp_seq.nextval, 'name" + i + "', "
                    + new Random().nextInt(10000) + ")";
                // 将 SQL 语句加入到 Batch 中
                stmt.addBatch(sql);
            }
            // 执行批处理
            stmt.executeBatch();

            // 提交
            con.commit();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {

```

```
        stmt.close();
    }
    if (con != null) {
        con.close();
    }
} catch (SQLException e) {
    System.out.println("释放资源时发生异常");
}
}
public static void main(String args[]) {
}
```

步骤四：测试是否批量插入数据成功

在 Batch 类的 main 方法中，调用 batchAdd 方法，代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;

public class Batch {

    public void batchAdd() {
        Connection con = null;
        Statement stmt = null;
        String sql = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            // 关闭自动提交
            con.setAutoCommit(false);

            for (int i = 0; i < 100; i++) {
                // 插入数据的 SQL 语句
                sql = "insert into emp(empno, ename, sal) values(" +
                    "emp seq.nextval, 'name" + i + "", " +
                    new Random().nextInt(10000) + ")";
                // 将 SQL 语句加入到 Batch 中
                stmt.addBatch(sql);
            }
            // 执行批处理
            stmt.executeBatch();
            // 提交
            con.commit();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}
```

```
    }
    public static void main(String args[]) {
        Batch batch = new Batch();
        batch.batchAdd();
    }
}
```

运行 Batch 类，然后去查看 oracle 数据库中的 Emp 表，会发现批量向该表中插入了 100 条记录。

- **完整代码**

本案例中，Batch 类的完整代码如下所示：

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;

public class Batch {

    public void batchAdd() {
        Connection con = null;
        Statement stmt = null;
        String sql = null;
        try {
            con = ConnectionSource.getConnection();
            stmt = con.createStatement();
            // 关闭自动提交
            con.setAutoCommit(false);

            for (int i = 0; i < 100; i++) {
                // 插入数据的 SQL 语句
                sql = "insert into emp(empno, ename, sal) values(" +
                    "emp_seq.nextval, 'name" + i + "", " +
                    + new Random().nextInt(10000) + ")";
                // 将 SQL 语句加入到 Batch 中
                stmt.addBatch(sql);
            }
            // 执行批处理
            stmt.executeBatch();
            // 提交
            con.commit();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }

    public static void main(String args[]) {
        Batch batch = new Batch();
        batch.batchAdd();
    }
}
```

```
}
```

db.properties 文件的内容与之前案例一样没有变化，该文件完整内容如下所示：

```
jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000
```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }

    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                    "day01/v4/db.properties"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            String driveClassName = dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
```

```

        dataSource.setUsername(username);
        dataSource.setPassword(password);

        // 初始化连接数
        if (initialSize != null)
            dataSource.setInitialSize(Integer.parseInt(initialSize));

        // 最小空闲连接
        if (minIdle != null)
            dataSource.setMinIdle(Integer.parseInt(minIdle));

        // 最大空闲连接
        if (maxIdle != null)
            dataSource.setMaxIdle(Integer.parseInt(maxIdle));

        // 超时回收时间(以毫秒为单位)
        if (maxWait != null)
            dataSource.setMaxWait(Long.parseLong(maxWait));

        // 最大连接数
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}

```

3. 插入 Dept 及其关联的 Emp 信息

- 问题

向 Dept 表中插入如图-1 所示的数据：

DEPTNO	DNAME	LOC
1	开发部	beijing

图-1

向 Emp 表中插入如图-2 所示的数据：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	donna	clerk	0	2014/3/12	5000.00	400.00	1

图-2

其中，Dept 表的主键为 deptno，Emp 表的主键为 empno。Emp 表的 deptno 列为外键，

该列的数据引用 Dept 表的主键列 deptno 的数据。

要求向 Dept 表插入数据的同时，向 Emp 表插入数据。Emp 表的 deptno 列的数据为之前插入 Dept 表的主键列 deptno 的数据。另外，Dept 表的主键列 deptno 的数据通过序列 dept_seq 获得，Emp 表的主键列 empno 的数据通过序列 emp_seq 获得。

• 方案

本案例中，向 Dept 表插入数据的同时，向 Emp 表插入数据。Emp 表的 deptno 列的数据为之前插入 Dept 表的主键列 deptno 的数据。首先，将两次插入操作放在同一个事务中。另外，要获取使用序列 dept_seq 自动生成主键列 deptno 的数据，为向 Emp 表中插入数据时 deptno 列的数据。

数据库表的主键一般情况下与业务无关，而且通常采用自动生成的方式。Oracle 数据库采用 sequence 的方式产生主键；而其他的一些数据库（如 SQLServer、MySQL）具有自动增长主键功能。

在实际开发中，经常需要在插入一条记录后获得刚刚生成记录的主键。

PreparedStatement 对象在执行 insert 操作后会保存数据库产生的自动主键，可调用 PreparedStatement 的 getGeneratedKeys 方法获得。getGeneratedKeys 的返回值为结果集对象，可以通过结果集获取刚刚生成的主键。使用 PreparedStatement 获取插入操作时数据库自动生成的主键列数据的核心代码如下：

```
sql = "insert into dept (deptno, dname, loc) values(dept_seq.nextval,?,?)";
con = ConnectionSource.getConnection();
stmt = con.prepareStatement(sql, new String[] { "deptno" });
stmt.setString(1, dept.getDname());
stmt.setString(2, dept.getLoc());
// 执行 SQL 语句
stmt.executeUpdate();
rs = stmt.getGeneratedKeys();
int deptno = 0;
if (rs.next()) {
    deptno = rs.getInt(1);
}
System.out.print("id:" + deptno);
```

其中 prepareStatement(sql, new String[] { "deptno" })方法的第一个参数表示要执行的 SQL 语句，第二参数表示 SQL 语句操作的表的主键列的列名字，并且 SQL 语句操作的表包含应该返回的自动生成键。如果 SQL 语句不是 insert 语句，则驱动程序将忽略该数组。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建序列 dept_seq

在 Oracle 数据库中创建序列名为 dept_seq，该序列的起始值为 1、步进为 1，SQL 语句如下所示：

```
create sequence dept_seq start with 1 increment by 1;
```

另外，序列 emp_seq 可以使用案例“批量插入 Emp 数据”中创建的该名字的序列。

步骤二：创建 Dept 类

Dept 类为实体类和数据表 dept 之间的映射，该类的代码如下所示：

```
public class Dept {  
  
    private int deptno;  
    private String dname;  
    private String loc;  
  
    public Dept() {  
        super();  
    }  
    public Dept(int deptno, String dname, String loc) {  
        super();  
        this.deptno = deptno;  
        this.dname = dname;  
        this.loc = loc;  
    }  
    public int getDeptno() {  
        return deptno;  
    }  
    public void setDeptno(int deptno) {  
        this.deptno = deptno;  
    }  
    public String getDname() {  
        return dname;  
    }  
    public void setDname(String dname) {  
        this.dname = dname;  
    }  
    public String getLoc() {  
        return loc;  
    }  
    public void setLoc(String loc) {  
        this.loc = loc;  
    }  
}
```

步骤三：准备 JDBC 操作数据库的基本代码

首先，新建类 DeptJoinEmp，在该类中新建 addEmp 方法；然后，准备数据库连接 Connection 对象、操作 SQL 语句的 Statement 对象以及设置事务管理；最后进行异常的处理，代码如下所示：

```
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class DeptJoinEmp {  
  
    public void addEmp(Emp emp, Dept dept) {  
        Connection con = null;  
        PreparedStatement stmt = null;  
        ResultSet rs = null;  
        String sql = null;
```

```

try {
    con = ConnectionSource.getConnection();
    // 关闭自动提交
    con.setAutoCommit(false);
    // 提交
    con.commit();

} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
}

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

```

步骤四：向 Dept 表中插入数据并获取自动生成的主键

向 Dept 表中插入数据，并使用 PreparedStatement 的 getGeneratedKeys 方法获得刚刚生成的主键，代码如下所示：

```

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

public class DeptJoinEmp {

    public void addEmp(Emp emp, Dept dept) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            con = ConnectionSource.getConnection();
            // 关闭自动提交
            con.setAutoCommit(false);

            // 插入主表
            sql = "insert into dept (deptno, dname, loc) values(dept_seq.nextval,?,?)";
            stmt = con.prepareStatement(sql, new String[] { "deptno" });
            stmt.setString(1, dept.getDname());
            stmt.setString(2, dept.getLoc());

```

```

        // 执行 SQL 语句
        stmt.executeUpdate();
        rs = stmt.getGeneratedKeys();
        int deptno = 0;
        if (rs.next()) {
            deptno = rs.getInt(1);
        }
        System.out.print("id:" + deptno);

        // 提交
        con.commit();

    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }
}

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

步骤五：向 Emp 表中插入数据

将上一步骤中获取到的 Dept 表的主键列数据，作为 Emp 表的外键列 deptno 的数据，代码如下所示：

```

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

public class DeptJoinEmp {

    public void addEmp(Emp emp, Dept dept) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            con = ConnectionSource.getConnection();
            // 关闭自动提交
            con.setAutoCommit(false);
            // 插入主表
            sql = "insert into dept (deptno, dname, loc)
values(dept_seq.nextval,?,?)";
    
```

```

stmt = con.prepareStatement(sql, new String[] { "deptno" });
stmt.setString(1, dept.getDname());
stmt.setString(2, dept.getLoc());

// 执行 SQL 语句
stmt.executeUpdate();
rs = stmt.getGeneratedKeys();
int deptno = 0;
if (rs.next()) {
    deptno = rs.getInt(1);
}
System.out.print("id:" + deptno);

// 插入从表

sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)"
+ " values(emp_seq.nextval,?, ?, ?, to_date(?,'yyyy-mm-dd'), ?, ?, ?)";
stmt = con.prepareStatement(sql);
stmt.setString(1, emp.getEname());
stmt.setString(2, emp.getJob());
stmt.setInt(3, emp.getMgr());
stmt.setString(4, emp.getHiredate());
stmt.setDouble(5, emp.getSal());
stmt.setDouble(6, emp.getComm());
stmt.setInt(7, deptno);
stmt.executeUpdate();

// 提交
con.commit();
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
}

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

步骤六：测试

按照图-1、图-2 中数据构造 Dept 对象和 Emp 对象，作为参数传递给 addEmp 方法，代码如下所示：

```

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

```

import java.sql.SQLException;

public class DeptJoinEmp {

    public void addEmp(Emp emp, Dept dept) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            // 插入主表
            sql = "insert into dept (deptno, dname, loc)
values(dept seq.nextval,?,?)";

            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql, new String[] { "deptno" });
            stmt.setString(1, dept.getDname());
            stmt.setString(2, dept.getLoc());

            // 关闭自动提交
            con.setAutoCommit(false);
            // 执行SQL语句
            stmt.executeUpdate();
            rs = stmt.getGeneratedKeys();
            int deptno = 0;
            if (rs.next()) {
                deptno = rs.getInt(1);
            }
            System.out.print("id:" + deptno);

            // 插入从表
            sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,
deptno)" + "values(emp_seq.nextval,?, ?, ?, to_date(?,'yyyy-mm-dd'), ?, ?, ?)";
            stmt = con.prepareStatement(sql);
            stmt.setString(1, emp.getEname());
            stmt.setString(2, emp.getJob());
            stmt.setInt(3, emp.getMgr());
            stmt.setString(4, emp.getHiredate());
            stmt.setDouble(5, emp.getSal());
            stmt.setDouble(6, emp.getComm());
            stmt.setInt(7, deptno);
            stmt.executeUpdate();

            // 提交
            con.commit();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (stmt != null) {
                    stmt.close();
                }
                if (con != null) {
                    con.close();
                }
            } catch (SQLException e) {
                System.out.println("释放资源时发生异常");
            }
        }
    }
}

```

```
public static void main(String args[]) {  
  
    Dept dept = new Dept();  
    dept.setDname("开发部");  
    dept.setLoc("beijing");  
  
    Emp emp = new Emp();  
    emp.setEname("donna");  
    emp.setJob("clerk");  
    emp.setHiredate("2014-03-12");  
    emp.setSal(5000.00);  
    emp.setComm(400.00);  
  
    DeptJoinEmp operation = new DeptJoinEmp();  
    operation.addEmp(emp, dept);  
  
}  
}
```

运行上述代码，向 Dept 表插入数据的同时，也向 Emp 表插入了数据。Emp 表的 deptno 列的数据为刚刚插入 Dept 表的主键列 deptno 的数据。

• 完整代码

本案例中，Dept 类的完整代码如下所示：

```
public class Dept {  
  
    private int deptno;  
    private String dname;  
    private String loc;  
  
    public Dept() {  
        super();  
    }  
    public Dept(int deptno, String dname, String loc) {  
        super();  
        this.deptno = deptno;  
        this.dname = dname;  
        this.loc = loc;  
    }  
    public int getDeptno() {  
        return deptno;  
    }  
    public void setDeptno(int deptno) {  
        this.deptno = deptno;  
    }  
    public String getDname() {  
        return dname;  
    }  
    public void setDname(String dname) {  
        this.dname = dname;  
    }  
    public String getLoc() {  
        return loc;  
    }  
    public void setLoc(String loc) {  
        this.loc = loc;  
    }  
}
```

Emp 类的完整代码如下：

```
public class Emp {  
    private int empNo;  
    private String ename;  
    private String job;  
    private int mgr;  
    private String hiredate;  
    private double sal;  
    private double comm;  
    private int deptno;  
  
    public Emp() {  
        super();  
    }  
    public Emp(int empNo, String ename, String job, int mgr, String hiredate,  
              double sal, double comm, int deptno) {  
        super();  
        this.empNo = empNo;  
        this.ename = ename;  
        this.job = job;  
        this.mgr = mgr;  
        this.hiredate = hiredate;  
        this.sal = sal;  
        this.comm = comm;  
        this.deptno = deptno;  
    }  
    public int getEmpNo() {  
        return empNo;  
    }  
    public void setEmpNo(int empNo) {  
        this.empNo = empNo;  
    }  
    public String getEname() {  
        return ename;  
    }  
    public void setEname(String ename) {  
        this.ename = ename;  
    }  
    public String getJob() {  
        return job;  
    }  
    public void setJob(String job) {  
        this.job = job;  
    }  
    public int getMgr() {  
        return mgr;  
    }  
    public void setMgr(int mgr) {  
        this.mgr = mgr;  
    }  
    public String getHiredate() {  
        return hiredate;  
    }  
    public void setHiredate(String hiredate) {  
        this.hiredate = hiredate;  
    }  
    public double getSal() {  
        return sal;  
    }  
    public void setSal(double sal) {  
        this.sal = sal;  
    }  
    public double getComm() {  
        return comm;  
    }  
}
```

```

public void setComm(double comm) {
    this.comm = comm;
}
public int getDeptno() {
    return deptno;
}
public void setDeptno(int deptno) {
    this.deptno = deptno;
}
}
  
```

DeptJoinEmp 类的完整代码如下所示：

```

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

public class DeptJoinEmp {

    public void addEmp(Emp emp, Dept dept) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        String sql = null;

        try {
            con = ConnectionSource.getConnection();
            // 关闭自动提交
            con.setAutoCommit(false);

            // 插入主表
            sql = "insert into dept (deptno, dname, loc)
values(dept_seq.nextval,?,?)";
            stmt = con.prepareStatement(sql, new String[] { "deptno" });
            stmt.setString(1, dept.getDname());
            stmt.setString(2, dept.getLoc());

            // 执行 SQL 语句
            stmt.executeUpdate();
            rs = stmt.getGeneratedKeys();
            int deptno = 0;
            if (rs.next()) {
                deptno = rs.getInt(1);
            }
            System.out.print("id:" + deptno);

            // 插入从表
            sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,
deptno" + " values(emp_seq.nextval,?, ?, ?, to_date(?,'YYYY-mm-dd'), ?, ?, ?)";
            stmt = con.prepareStatement(sql);
            stmt.setString(1, emp.getEname());
            stmt.setString(2, emp.getJob());
            stmt.setInt(3, emp.getMgr());
            stmt.setString(4, emp.getHiredate());
            stmt.setDouble(5, emp.getSal());
            stmt.setDouble(6, emp.getComm());
            stmt.setInt(7, deptno);
            stmt.executeUpdate();
            // 提交
            con.commit();
        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
  
```

```

        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }

    public static void main(String args[]) {
        Dept dept = new Dept();
        dept.setDname("开发部");
        dept.setLoc("beijing");

        Emp emp = new Emp();
        emp.setEname("donna");
        emp.setJob("clerk");
        emp.setHiredate("2014-03-12");
        emp.setSal(5000.00);
        emp.setComm(400.00);

        DeptJoinEmp operation = new DeptJoinEmp();
        operation.addEmp(emp, dept);
    }
}

```

db.properties 文件的内容与之前案例一样没有变化，该文件完整内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000

```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;
}

```

```
public ConnectionSource() {
}
public static void init() {

    Properties dbProps = new Properties();
    // 取配置文件可以根据实际的不同修改
    try {

        dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
                "day01/v4/db.properties"));
    } catch (IOException e) {
        e.printStackTrace();
    }

    try {
        String                     driveClassName
        dbProps.getProperty("jdbc.driverClassName");
        String url = dbProps.getProperty("jdbc.url");
        String username = dbProps.getProperty("jdbc.username");
        String password = dbProps.getProperty("jdbc.password");

        String initialSize = dbProps.getProperty("dataSource.initialSize");
        String minIdle = dbProps.getProperty("dataSource.minIdle");
        String maxIdle = dbProps.getProperty("dataSource.maxIdle");
        String maxWait = dbProps.getProperty("dataSource.maxWait");
        String maxActive = dbProps.getProperty("dataSource.maxActive");

        dataSource = new BasicDataSource();
        dataSource.setDriverClassName(driveClassName);
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);

        // 初始化连接数
        if (initialSize != null)
            dataSource.setInitialSize(Integer.parseInt(initialSize));

        // 最小空闲连接
        if (minIdle != null)
            dataSource.setMinIdle(Integer.parseInt(minIdle));

        // 最大空闲连接
        if (maxIdle != null)
            dataSource.setMaxIdle(Integer.parseInt(maxIdle));

        // 超时回收时间(以毫秒为单位)
        if (maxWait != null)
            dataSource.setMaxWait(Long.parseLong(maxWait));

        // 最大连接数
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
}
```

```
        }
        return conn;
    }
}
```

4. 实现 Emp 数据的分页查询 (Oracle 和 MySQL)

- 问题

使用 JDBC 分别连接 Oracle 数据库和 MySQL 数据库 , 实现对 Emp 表数据的分页查询功能。

- 方案

对于较大的数据量 , 通常采用分页查询的方式。不同的数据库产品有不同的数据库级的分页查询策略。例如 : Oracle 通常使用 rownum 的方式 ; 而 MySQL 使用 limit 的方式。

Oracle 采用 rownum 和子查询实现分页查询 , SQL 语句如下 :

```
select * from
(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm, deptno from
(select * from emp order by empno))
where rn between 6 and 10;
```

上述 SQL 语句的功能为按照员工编号升序排列员工信息 , 获取排序后第 6 至 10 位之间的 5 条员工信息。

实现上述功能的 MySQL 数据库的 SQL 语句如下 :

```
select * from emp order by empno limit 5,5;
```

MySQL 中使用 limit 关键字实现分页查询。其中 , limit 后第一个参数为开始获取数据的行号 (从 0 开始) , 第二个参数为获取记录的行数。第二个参数可省略 , 表示从第一个参数开始 , 获取后续所有记录。

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：添加方法 findByPageOracle 方法，并构建该方法的骨架代码

首先 , 在 EmpDAO 类中添加方法 findByPageOracle , 该方法的声明如下所示 :

```
public void findByPageOracle(int page, int pageSize) {}
```

其中 , 参数 page 表示要查询的页数、参数 pageSize 表示每页显示的记录数。

然后 , 构建 findByPageOracle 方法的骨架代码 , 代码如下所示 :

```
import java.sql.Connection;
```

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import day02.ConnectionSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        // dao.update(emp);

    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {

            } catch (SQLException e) {
                System.out.println("数据库访问异常！");
                throw new RuntimeException(e);
            } finally {
                try {
                    if (rs != null) {
                        rs.close();
                    }
                    if (stmt != null) {
                        stmt.close();
                    }
                    if (con != null) {
                        con.close();
                    }
                } catch (SQLException e) {
                    System.out.println("释放资源时发生异常");
                }
            }
        }
    public void findAll() {
        ...
    }

    public void add(Emp emp) {
        ...
    }
```

```
public void update(Emp emp) {  
    ...  
}
```

步骤二：定义 SQL 语句

在 `findByPageOracle` 方法中，定义变量 `sql_total` 以及 `sql` 来表示两条 SQL 语句，一条用于查询 `Emp` 表的总记录数，另一条作为分页的 SQL 语句，代码如下所示：

```
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
import day02.ConnectionSource;  
import day02.Emp;  
  
public class EmpDAO {  
    public static void main(String[] args) {  
        // 1.select  
        EmpDAO dao = new EmpDAO();  
        // dao.findAll();  
  
        // 2.insert  
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,  
                         500.00, 10);  
        // dao.add(emp);  
  
        // 3.update  
        emp.setSal(4500.00);  
        // dao.update(emp);  
    }  
  
    /**  
     * @param page  
     *          要查看第几页  
     * @param pageSize  
     *          每页记录数  
     */  
    public void findByPageOracle(int page, int pageSize) {  
        Connection con = null;  
        PreparedStatement stmt = null;  
        ResultSet rs = null;  
  
        String sql_total = "select count(*) from emp";  
        String sql = "select * from "  
            + "(select rownum rn, empno, ename, job,mgr, hiredate, sal, comm,"  
            + "deptno from "  
            + "(select * from emp order by empno) )"  
            + " where rn between ? and ?";  
  
        try {  
  
            } catch (SQLException e) {  
                System.out.println("数据库访问异常!");  
                throw new RuntimeException(e);  
            } finally {  
        }  
    }  
}
```

```
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }

}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
```

步骤三：查询 Emp 表的总记录数

获取数据库连接，使用 PreparedStatement 执行 SQL 语句（sql_total 变量定义的 SQL 语句），获取数据库中 Emp 表的总记录数，代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import day02.DataSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        // dao.update(emp);
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     */
```

```

*
每页记录数
*/
public void findByPageOracle(int page, int pageSize) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    int total = -1;// 总记录数

    String sql_total = "select count(*) from emp";
    String sql = "select * from "
        +"(select rownum rn, empno, ename, job,mgr, hiredate, sal, comm,
deptno from "
        + "(select * from emp order by empno) )"
        + " where rn between ? and ?";

    try {

        con = ConnectionSource.getConnection();
        stmt = con.prepareStatement(sql_total);

        // 获得总的记录数
        rs = stmt.executeQuery();
        if (rs.next()) {
            total = rs.getInt(1);
        }

    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }
}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}

```

步骤四：计算总页数

将总记录数与每页记录数取余数，如果余数为 0，则总页数等于总记录数除以每页记录数的商；如果余数不为 0，则总页数等于总记录数除以每页记录数的商的基础上加 1，代码如下所示：

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import day02.ConnectionSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        // dao.update(emp);
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        int total = -1; // 总记录数
        int pages = -1; // 总页数

        String sql_total = "select count(*) from emp";
        String sql = "select * from "
                    + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
                    + "(select * from emp order by empno) )"
                    + " where rn between ? and ?";

        try {
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql_total);

            // 获得总的记录数
            rs = stmt.executeQuery();
            if (rs.next()) {
                total = rs.getInt(1);
            }
        }
```

```

        // 计算总共需要多少页

        int mod = total % pageSize;
        if (mod == 0)
            pages = total / pageSize;
        else
            pages = total / pageSize + 1;

    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }
}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
}

```

步骤五：边界判断

如果要查看的页码大于总页数，则要查看的页码等于总页数；如果要查看的页码小于1，则要查看的页码等于1，代码如下所示：

```

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

import day02.DataSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
    }
}

```

```

        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                           500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        // dao.update(emp);

        // 4.findByPage
        dao.findByPageOracle(2, 3); // 查看第二页，每页 3 条
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        int total = -1; // 总记录数
        int pages = -1; // 总页数

        String sql_total = "select count(*) from emp";
        String sql = "select * from "
                    + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
                    + "(select * from emp order by empno) )"
                    + " where rn between ? and ?";

        try {
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql_total);

            // 获得总的记录数
            rs = stmt.executeQuery();
            if (rs.next()) {
                total = rs.getInt(1);
            }

            // 计算总共需要多少页
            int mod = total % pageSize;
            if (mod == 0)
                pages = total / pageSize;
            else
                pages = total / pageSize + 1;

            // 如果要查看的页数大于最大页，或者小于 1，则取最后一页或第一页

            if (page > pages){
                page = pages;
            }else if (page < 1) {
                page = 1;
            }

        } catch (SQLException e) {
            System.out.println("数据库访问异常！");
            throw new RuntimeException(e);
        } finally {
    }
}

```

```

        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源时发生异常");
        }
    }

}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
}
    
```

步骤六：计算取记录的起始位置和结束位置

起始位置 (begin) 的计算公式如下：

```
int begin = (page - 1) * pageSize + 1;
```

结束位置 (end) 的计算公式如下：

```
int end = begin + pageSize - 1;
```

在 findByPageOracle 方法中的实现代码如下所示：

```

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

import day02.DataSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        // dao.add(emp);
    }
}
    
```

```

    // 3.update
    emp.setSal(4500.00);
    // dao.update(emp);
}

/**
 *
 * @param page
 *          要查看第几页
 * @param pageSize
 *          每页记录数
 */
public void findByPageOracle(int page, int pageSize) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    int total = -1;// 总记录数
    int pages = -1;// 总页数

    String sql_total = "select count(*) from emp";
    String sql = "select * from "
        + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
        + "(select * from emp order by empno) )"
        + " where rn between ? and ?";

    try {
        con = ConnectionSource.getConnection();
        stmt = con.prepareStatement(sql_total);

        // 获得总的记录数
        rs = stmt.executeQuery();
        if (rs.next()) {
            total = rs.getInt(1);
        }

        // 计算总共需要多少页
        int mod = total % pageSize;
        if (mod == 0)
            pages = total / pageSize;
        else
            pages = total / pageSize + 1;

        // 如果要查看的页数大于最大页，或者小于1，则取最后一页或第一页
        if (page > pages)
            page = pages;
        else if (page < 1) {
            page = 1;
        }

        int begin = (page - 1) * pageSize + 1;
        int end = begin + pageSize - 1;

    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
        }
    }
}

```

```

        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
}

```

步骤七：执行分页查询 SQL 语句

在 `findByIdPageOracle` 方法中执行分页查询的代码如下所示：

```

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

import day02.ConnectionSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
                          500.00, 10);
        // dao.add(emp);

        // 3.update
        emp.setSal(4500.00);
        // dao.update(emp);
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByIdPageOracle(int page, int pageSize) {
        Connection con = null;

```

```

PreparedStatement stmt = null;
ResultSet rs = null;
int total = -1;// 总记录数
int pages = -1;// 总页数

String sql_total = "select count(*) from emp";
String sql = "select * from "
        + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
        + "(select * from emp order by empno) )"
        + " where rn between ? and ?";

try {
    con = ConnectionSource.getConnection();
    stmt = con.prepareStatement(sql_total);

    // 获得总的记录数
    rs = stmt.executeQuery();
    if (rs.next()) {
        total = rs.getInt(1);
    }

    // 计算总共需要多少页
    int mod = total / pageSize;
    if (mod == 0)
        pages = total / pageSize;
    else
        pages = total / pageSize + 1;

    // 如果要查看的页数大于最大页，或者小于1，则取最后一页或第一页
    if (page > pages)
        page = pages;
    else if (page < 1) {
        page = 1;
    }

    int begin = (page - 1) * pageSize + 1;
    int end = begin + pageSize - 1;

    stmt = con.prepareStatement(sql);
    stmt.setInt(1, begin);
    stmt.setInt(2, end);
    rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + ","
                + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
    }
}

} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    }
}

```

```

        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}

}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
}

```

步骤八：测试

在 EmpDAO 类的 main 方法中调用 findByPageOracle 方法，代码如下所示：

```

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

import day02.ConnectionSource;
import day02.Emp;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        //Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
        //      500.00, 10);
        // dao.add(emp);

        // 3.update
        //emp.setSal(4500.00);
        // dao.update(emp);
        // 4.findByPageOracle

        dao.findByPageOracle(2, 3); // 查看第二页，每页 3 条

    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        Connection con = null;

```

```

PreparedStatement stmt = null;
ResultSet rs = null;
int total = -1;// 总记录数
int pages = -1;// 总页数

String sql_total = "select count(*) from emp";
String sql = "select * from "
        + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
        + "(select * from emp order by empno) )"
        + " where rn between ? and ?";

try {
    con = ConnectionSource.getConnection();
    stmt = con.prepareStatement(sql_total);

    // 获得总的记录数
    rs = stmt.executeQuery();
    if (rs.next()) {
        total = rs.getInt(1);
    }

    // 计算总共需要多少页
    int mod = total / pageSize;
    if (mod == 0)
        pages = total / pageSize;
    else
        pages = total / pageSize + 1;

    // 如果要查看的页数大于最大页，或者小于1，则取最后一页或第一页
    if (page > pages)
        page = pages;
    else if (page < 1) {
        page = 1;
    }

    int begin = (page - 1) * pageSize + 1;
    int end = begin + pageSize - 1;

    stmt = con.prepareStatement(sql);
    stmt.setInt(1, begin);
    stmt.setInt(2, end);
    rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + ","
                + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
    }
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}

```

```

        }
    }

}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
}
}

```

运行 EmpDAO 类，在控制台会输出第二页的三条数据。

步骤九：连接 MySQL 数据库，实现对 Emp 表中数据的分页查询

连接 MySQL 数据库，实现对 Emp 表中数据的分页查询，与连接 Oracle 是类似的。需要注意的是将 db.properties 文件中连接数据库的信息改为与 MySQL 数据库相关的，该文件内容如下：

```

#jdbc.driverClassName=oracle.jdbc.OracleDriver
#jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
#jdbc.username=scott
#jdbc.password=tiger

jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/tts7
jdbc.username=root
jdbc.password=root

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000

```

在 EmpDAO 类中添加 findByPageMySQL 方法，实现连接 MySQL 数据库，实现对 Emp 表中数据的分页查询，代码如下所示：

```

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

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
    }
}

```

```

        // dao.findAll();

        // 2.insert
        //Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
        //      500.00, 10);
        // dao.add(emp);

        // 3.update
        //emp.setSal(4500.00);
        // dao.update(emp);

        // 4.findByPageOracle
        //dao.findByPageOracle(2, 3); // 查看第二页，每页3条

        // 5.findByPageMySQL
        dao.findByPageMySQL(2, 3); // 查看第二页，每页3条

    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        ...
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageMySQL(int page, int pageSize) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        int total = -1; // 总记录数
        int pages = -1; // 总页数

        String sql_total = "select count(*) from emp";
        String sql = "select * from emp order by empno limit ?,?";

        try {
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql_total);

            // 获得总的记录数
            rs = stmt.executeQuery();
            if (rs.next()) {
                total = rs.getInt(1);
            }
            System.out.println(total);
            // 计算总共多少页
            int mod = total % pageSize;
            if (mod == 0)

```

```

        pages = total / pageSize;
    else
        pages = total / pageSize + 1;

        // 如果要查看的页数大于最大页，或者小于1，则取最后一页或第一页
    if (page > pages){
        page = pages;
    }else if (page < 1) {
        page = 1;
    }
    System.out.println(sql);
    int start = (page - 1) * pageSize;
    stmt = con.prepareStatement(sql);
    stmt.setInt(1, start);
    stmt.setInt(2, pageSize);
    rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getInt("empno") + ","
                        + rs.getString("ename") + ","
                        + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
    }
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
}

public void findAll() {
    ...
}

public void add(Emp emp) {
    ...
}

public void update(Emp emp) {
    ...
}
    
```

• 完整代码

本案例中，EmpDAO 类的完整代码如下所示：

```

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

```
import java.sql.SQLException;
import java.sql.Statement;

public class EmpDAO {
    public static void main(String[] args) {
        // 1.select
        EmpDAO dao = new EmpDAO();
        // dao.findAll();

        // 2.insert
        //Emp emp = new Emp(1001, "rose", "Analyst", 7901, "2014-05-01", 3000.00,
        //      500.00, 10);
        // dao.add(emp);

        // 3.update
        //emp.setSal(4500.00);
        // dao.update(emp);

        // 4.findByPageOracle
        //dao.findByPageOracle(2, 3); // 查看第二页，每页 3 条

        // 5.findByPageMySQL
        dao.findByPageMySQL(2, 3); // 查看第二页，每页 3 条
    }

    /**
     *
     * @param page
     *          要查看第几页
     * @param pageSize
     *          每页记录数
     */
    public void findByPageOracle(int page, int pageSize) {
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        int total = -1; // 总记录数
        int pages = -1; // 总页数

        String sql_total = "select count(*) from emp";
        String sql = "select * from "
            + "(select rownum rn, empno, ename, job, mgr, hiredate, sal, comm,
deptno from "
            + "(select * from emp order by empno) )"
            + " where rn between ? and ?";

        try {
            con = ConnectionSource.getConnection();
            stmt = con.prepareStatement(sql_total);

            // 获得总的记录数
            rs = stmt.executeQuery();
            if (rs.next()) {
                total = rs.getInt(1);
            }

            // 计算总共需要多少页
            int mod = total % pageSize;
            if (mod == 0)
                pages = total / pageSize;
            else
                pages = total / pageSize + 1;

            // 如果要查看的页数大于最大页，或者小于 1，则取最后一页或第一页
            if (page > pages)
```

```

        page = pages;
    else if (page < 1) {
        page = 1;
    }

    int begin = (page - 1) * pageSize + 1;
    int end = begin + pageSize - 1;

    stmt = con.prepareStatement(sql);
    stmt.setInt(1, begin);
    stmt.setInt(2, end);
    rs = stmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getInt("empno") + ","
            + rs.getString("ename") + "," + ","
            + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
    }
} catch (SQLException e) {
    System.out.println("数据库访问异常!");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
}

/**
*
* @param page
*          要查看第几页
* @param pageSize
*          每页记录数
*/
public void findByPageMySQL(int page, int pageSize) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    int total = -1;// 总记录数
    int pages = -1;// 总页数

    String sql_total = "select count(*) from emp";
    String sql = "select * from emp order by empno limit ?,?";

    try {
        con = ConnectionSource.getConnection();
        stmt = con.prepareStatement(sql_total);

        // 获得总的记录数
        rs = stmt.executeQuery();
        if (rs.next()) {
            total = rs.getInt(1);
        }
        System.out.println(total);
        // 计算总共需要多少页
        int mod = total % pageSize;
    }
}

```

```
if (mod == 0)
    pages = total / pageSize;
else
    pages = total / pageSize + 1;

// 如果要查看的页数大于最大页，或者小于1，则取最后一页或第一页
if (page > pages)
    page = pages;
else if (page < 1) {
    page = 1;
}
System.out.println(sql);
int start = (page - 1) * pageSize;
stmt = con.prepareStatement(sql);
stmt.setInt(1, start);
stmt.setInt(2, pageSize);
rs = stmt.executeQuery();
while (rs.next()) {
    System.out.println(rs.getInt("empno") + ","
        + rs.getString("ename") + ","
        + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
}
} catch (SQLException e) {
    System.out.println("数据库访问异常！");
    throw new RuntimeException(e);
} finally {
    try {
        if (rs != null) {
            rs.close();
        }
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源时发生异常");
    }
}
}

public void findAll() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        con = ConnectionSource.getConnection();
        stmt = con.createStatement();
        rs = stmt
            .executeQuery("select empno, ename, sal, hiredate from emp");
        while (rs.next()) {
            System.out.println(rs.getInt("empno") + ","
                + rs.getString("ename") + ","
                + rs.getDouble("sal") + "," + rs.getDate("hiredate"));
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常！");
        throw new RuntimeException(e);
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (stmt != null) {
                stmt.close();
            }
        }
    }
}
```

```

        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源发生异常");
    }
}

public void add(Emp emp) {
    Connection con = null;
    Statement stmt = null;
    int flag = -1;
    String sql = "insert into emp(empno, ename, job, mgr, hiredate, sal, comm,
deptno) values("
        + emp.getEmpNo()
        + ", "
        + " '"
        + emp.getEname()
        + ", "
        + " '"
        + emp.getJob()
        + ", "
        + emp.getMgr()
        + ", "
        + "to date('"
        + emp.getHiredate()
        + "'", 'yyyy-mm-dd'), "
        + emp.getSal()
        + ", "
        + emp.getComm() + ", " + emp.getDeptno() + ")";
}

try {
    con = ConnectionSource.getConnection();
    stmt = con.createStatement();

    flag = stmt.executeUpdate(sql);
    if (flag > 0) {
        System.out.println("新增记录成功!");
    }
} catch (SQLException e) {
    System.out.println("数据库访问异常!");
    throw new RuntimeException(e);
} finally {
    try {
        if (stmt != null) {
            stmt.close();
        }
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("释放资源发生异常");
    }
}
}

public void update(Emp emp) {
    Connection con = null;
    Statement stmt = null;
    int flag = -1;
    String sql = "update emp set sal = " + emp.getSal() + ", " + " comm = "
        + emp.getComm() + " where empno = " + emp.getEmpNo();

    try {

```

```

        con = ConnectionSource.getConnection();
        stmt = con.createStatement();

        flag = stmt.executeUpdate(sql);
        if (flag > 0) {
            System.out.println("更新记录成功!");
        }
    } catch (SQLException e) {
        System.out.println("数据库访问异常!");
        throw new RuntimeException(e);
    } finally {
        try {
            if (stmt != null) {
                stmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (SQLException e) {
            System.out.println("释放资源发生异常");
        }
    }
}
}

```

db.properties 文件的完整内容如下所示：

```

#jdbc.driverClassName=oracle.jdbc.OracleDriver
#jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
#jdbc.username=scott
#jdbc.password=tiger

jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/tts7
jdbc.username=root
jdbc.password=root

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=50
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000

```

ConnectionSource 类与之前案例一样没有变化，该类完整内容如下所示：

```

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionSource {
    private static BasicDataSource dataSource = null;

    public ConnectionSource() {
    }
    public static void init() {

```

```

Properties dbProps = new Properties();
// 取配置文件可以根据实际的不同修改
try {
    dbProps.load(ConnectionSource.class.getClassLoader().getResourceAsStream(
        "day01/v4/db.properties"));
} catch (IOException e) {
    e.printStackTrace();
}

try {
    String                     driveClassName      =
dbProps.getProperty("jdbc.driverClassName");
    String url = dbProps.getProperty("jdbc.url");
    String username = dbProps.getProperty("jdbc.username");
    String password = dbProps.getProperty("jdbc.password");

    String initialSize= dbProps.getProperty("dataSource.initialSize");
    String minIdle = dbProps.getProperty("dataSource.minIdle");
    String maxIdle = dbProps.getProperty("dataSource.maxIdle");
    String maxWait = dbProps.getProperty("dataSource.maxWait");
    String maxActive = dbProps.getProperty("dataSource.maxActive");

    dataSource = new BasicDataSource();
    dataSource.setDriverClassName(driveClassName);
    dataSource.setUrl(url);
    dataSource.setUsername(username);
    dataSource.setPassword(password);

    // 初始化连接数
    if (initialSize != null)
        dataSource.setInitialSize(Integer.parseInt(initialSize));

    // 最小空闲连接
    if (minIdle != null)
        dataSource.setMinIdle(Integer.parseInt(minIdle));

    // 最大空闲连接
    if (maxIdle != null)
        dataSource.setMaxIdle(Integer.parseInt(maxIdle));

    // 超时回收时间(以毫秒为单位)
    if (maxWait != null)
        dataSource.setMaxWait(Long.parseLong(maxWait));

    // 最大连接数
    if (maxActive != null) {
        if (!maxActive.trim().equals("0"))
            dataSource.setMaxActive(Integer.parseInt(maxActive));
    }
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("创建连接池失败!请检查设置!!!");
}
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}
}

```

5. 完成 NetCTOSS 项目中，账务账号的 DAO 设计及实现

• 问题

完成 NetCTOSS 项目中，账务账号的 DAO 设计及实现，详细要求如下：

1. 新增账务账号。
2. 修改某个账务账号的信息。
3. 修改某个账务账号的状态。
4. 查询所有的账务账号。
5. 根据某个账务账号 ID 查询该账务账号的全部信息。

• 方案

在企业开发时，通常采用分层模式，常用的层次划分为表现层+控制层+业务层+持久层+数据源。持久层的功能是通过某些技术或框架将数据库的内容映射成对象，通过操作这些对象实现对数据库的操作。其主要目的是弥补业务对象和数据源关系表之间的差异，便于对数据库操作。持久层可采用 JDBC、Hibernate、Mybatis 等技术实现。

本案例采用 DAO 模式通过 JDBC 来实现持久层。DAO 模式就是 Data Access Object，即数据访问对象。它存在于数据源和业务层之间，封装了对数据库的访问细节，例如数据库连接、发送执行 SQL 语句和连接资源的关闭等。DAO 的主要目的是将底层数据访问操作与高层业务逻辑操作完全分开，为业务层提供透明的数据访问服务，增强程序的灵活性。DAO 组件封装了对数据表的操作，为业务组件提供数据访问服务。在业务组件中，有些简单的业务处理，仅需要使用某一个 DAO 组件的一个方法就可以完成；但有些业务处理比较复杂，需要使用若干个 DAO 组件的方法完成。为了保障业务的完整性和 DAO 组件的重用性，因此不能将事务控制写在 DAO 组件的每个方法内，而应该是将事务控制独立封装，然后在业务方法中应用。本案例的工程结构如图-3 所示。

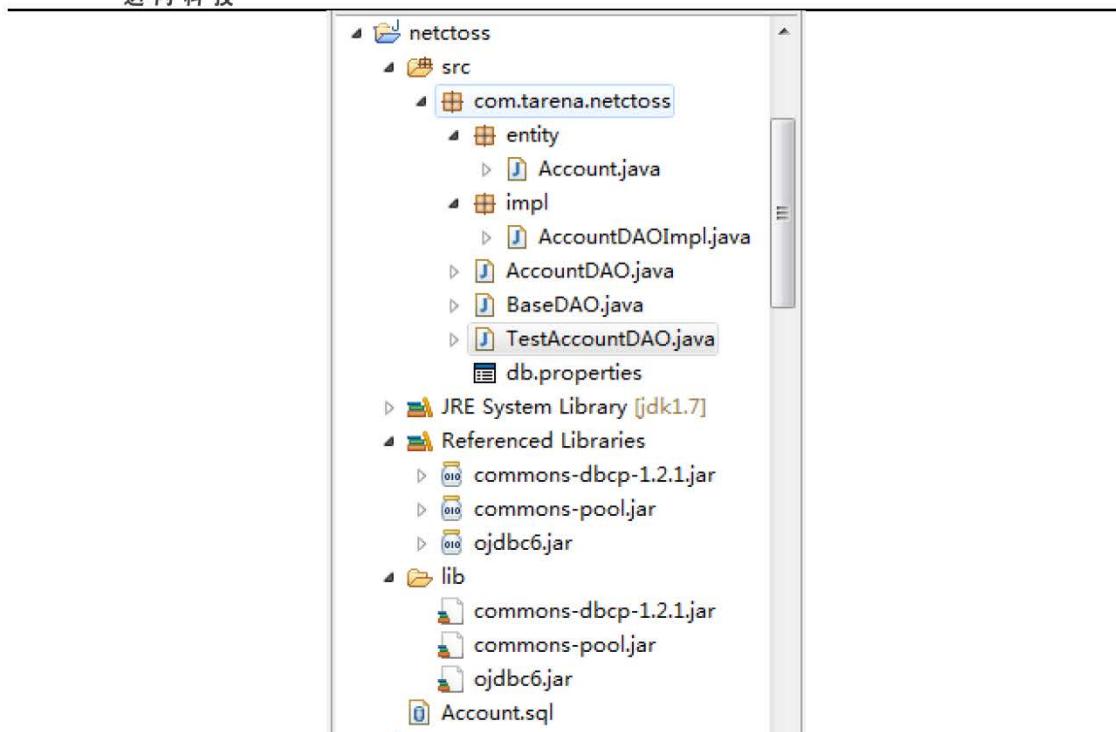


图-3

对图-3 中的工程文件的介绍如下：

1. Account.java 为数据库中的 Account 表和 Java 对象的映射；
2. db.properties 封装了连接数据的信息；
3. BaseBAO.java 该类中封装了创建数据库连接方法以及事务处理；
4. AccountDAO.java 为一个接口，该接口中定义了五个方法，该五个方法的声明如下所示：

```
package com.tarena.netctoss;

import java.sql.SQLException;
import java.util.List;

import com.tarena.netctoss.entity.Account;

public interface AccountDAO {

    /**
     * 根据某个账务账号 ID 查询该账务账号的全部信息
     * @param id 账务账号 ID
     * @return 某个账务账号的全部信息，为一个 Account 对象
     * @throws SQLException
     */
    Account findById(Integer id) throws SQLException;
    /**
     * 查询所有的账务账号
     * @return 所有账务账号 返回 List 集合
     * @throws SQLException
     */
    List<Account> findAll() throws SQLException;
}
```

```

* 新增账务账号
* @param account 要添加的账务账号
* @return 添加后账务账号，包含账号账号 ID
* @throws SQLException
*/
Account save(Account account) throws SQLException;
/** 
 * 修改某个账务账号的信息
* @param account 要修改的账务账号
* @return 返回修改后的账务账号
* @throws SQLException
*/
Account modify(Account account) throws SQLException;
/** 
 * 修改某个账务账号的状态
* @param account 要修改状态的账务账号
* @return 返回修改状态后的账务账号
* @throws SQLException
*/
Account modifyStatus(Account account) throws SQLException;
}

```

以上五个方法的作用，请参考注释部分。

5. AccountDAOImpl.java 该类继承自 BaseDAO ,来直接获取创建连接的方法 ;另外 ,该类实现了 AccountDAO 接口 ,将该接口中的方法做出实现。

6. TestAccountDAO.java 该类用于测试 AccountDAOImpl 所实现的方法的正确性。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：准备环境

首先 , 创建名为 netctoss 的工程 ;然后 , 在该工程下创建 lib 目录 , 存储实现该工程的功能所需的 jar 包 ;最后 , 将所需的 jar 导入到该工程环境中。

步骤二：创建序列、表以及向表中插入数据

首先 , 创建名为 account_seq 的序列 ;然后 , 创建名为 Account 的表并向该表中插入测试数据 , SQL 语句如下所示 :

```

create sequence account_seq;

create table account(
    account_id    number(9) constraint account_id_pk primary key,
    recommender_id number(9) constraint account_recommender_id_fk
        references account(account_id),
    login_name    varchar2(30) not null
        constraint account_login_name_uk unique,
    login_passwd   varchar2(30) not null,
    status         char(1) constraint account_status_ck
        check (status in (0,1,2)) not null,
    create_date    date default sysdate,
    pause_date     date,
    close_date     date,
)

```

```

real name      varchar2(20)  not null,
idcard no      char(18)      not null
                constraint account_incard_no unique,
birthdate      date,
gender         char(1)       constraint account_gender_ck
                check (gender in (0,1)) not null,
occupation     varchar2(50),
telephone      varchar2(15)  not null,
email          varchar2(50),
mailaddress    varchar2(200),
zipcode        char(6),
qq             varchar2(15),
last login time   date,
last login ip    varchar2(15)
);

INSERT INTO ACCOUNT(ACCOUNT_ID, RECOMMENDER_ID, LOGIN_NAME, LOGIN_PASSWD,
STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO, BIRTHDATE,
GENDER,OCCUPATION,TELEPHONE,EMAIL,MAILADDRESS,ZIPCODE,QQ,LAST LOGIN TIME,
LAST LOGIN IP)
VALUES  (ACCOUNT SEQ.NEXTVAL,null,  'a',  '123',  '0',  SYSDATE,  NULL,
NULL,'zhangsan', '1234',
      to_date('1981-01-01','yyyy-mm-dd'), '0', 'job','123', 'a@b.c', 'test avenue',
'1223', '1234', SYSDATE, '192')

```

步骤三：创建数据库中的 Account 表和 Java 对象的映射类 Account

代码如下所示：

```

package com.tarena.netctoss.entity;

import java.util.Date;

public class Account {
    private int id;
    private int recommenderId;
    private String loginName;
    private String loginPasswd;
    private String status;
    private Date createDate;
    private Date pauseDate;
    private Date closeDate;
    private String realName;
    private String idcardNo;
    private Date birthdate;
    private String gender;
    private String occupation;
    private String telephone;
    private String email;
    private String mailaddress;
    private String zipcode;
    private String qq;
    private Date lastLoginTime;
    private String lastLoginIp;

    public Account() {
        super();
    }

    public Account(int id, int recommenderId, String loginName,
                  String loginPasswd, String status, Date createDate, Date pauseDate,
                  Date closeDate, String realName, String idcardNo, Date birthdate,
                  String gender, String occupation, String telephone, String email,
                  String mailaddress, String zipcode, String qq, Date lastLoginTime,
                  String lastLoginIp) {
        super();

```

```
    this.id = id;
    this.recommenderId = recommenderId;
    this.loginName = loginName;
    this.loginPasswd = loginPasswd;
    this.status = status;
    this.createDate = createDate;
    this.pauseDate = pauseDate;
    this.closeDate = closeDate;
    this.realName = realName;
    this.idcardNo = idcardNo;
    this.birthdate = birthdate;
    this.gender = gender;
    this.occupation = occupation;
    this.telephone = telephone;
    this.email = email;
    this.mailaddress = mailaddress;
    this.zipcode = zipcode;
    this.qq = qq;
    this.lastLoginTime = lastLoginTime;
    this.lastLoginIp = lastLoginIp;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public int getRecommenderId() {
    return recommenderId;
}
public void setRecommenderId(int recommenderId) {
    this.recommenderId = recommenderId;
}
public String getLoginName() {
    return loginName;
}
public void setLoginName(String loginName) {
    this.loginName = loginName;
}
public String getLoginPasswd() {
    return loginPasswd;
}
public void setLoginPasswd(String loginPasswd) {
    this.loginPasswd = loginPasswd;
}
public String getStatus() {
    return status;
}
public void setStatus(String status) {
    this.status = status;
}
public Date getCreateDate() {
    return createDate;
}
public void setCreateDate(Date createDate) {
    this.createDate = createDate;
}
public Date getPauseDate() {
    return pauseDate;
}
public void setPauseDate(Date pauseDate) {
    this.pauseDate = pauseDate;
}
public Date getCloseDate() {
    return closeDate;
}
public void setCloseDate(Date closeDate) {
    this.closeDate = closeDate;
}
```

```
    }
    public String getRealName() {
        return realName;
    }
    public void setRealName(String realName) {
        this.realName = realName;
    }
    public String getIdcardNo() {
        return idcardNo;
    }
    public void setIdcardNo(String idcardNo) {
        this.idcardNo = idcardNo;
    }
    public Date getBirthdate() {
        return birthdate;
    }
    public void setBirthdate(Date birthdate) {
        this.birthdate = birthdate;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public String getOccupation() {
        return occupation;
    }
    public void setOccupation(String occupation) {
        this.occupation = occupation;
    }
    public String getTelephone() {
        return telephone;
    }
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getMailaddress() {
        return mailaddress;
    }
    public void setMailaddress(String mailaddress) {
        this.mailaddress = mailaddress;
    }
    public String getZipcode() {
        return zipcode;
    }
    public void setZipcode(String zipcode) {
        this.zipcode = zipcode;
    }
    public String getQq() {
        return qq;
    }
    public void setQq(String qq) {
        this.qq = qq;
    }
    public Date getLastLoginTime() {
        return lastLoginTime;
    }
    public void setLastLoginTime(Date lastLoginTime) {
        this.lastLoginTime = lastLoginTime;
    }
    public String getLastLoginIp() {
```

```

        return lastLoginIp;
    }
    public void setLastLoginIp(String lastLoginIp) {
        this.lastLoginIp = lastLoginIp;
    }
    public String toString(){
        System.out.println("id=" + id);
        System.out.println("recommenderId=" + recommenderId);
        System.out.println("loginName=" + loginName);
        System.out.println("loginPasswd=" + loginPasswd);
        System.out.println("status=" + status);
        System.out.println("createDate=" + createDate);
        System.out.println("pauseDate=" + pauseDate);
        System.out.println("closeDate=" + closeDate);
        System.out.println("realName=" + realName);
        System.out.println("idcardNo=" + idcardNo);
        System.out.println("birthdate=" + birthdate);
        System.out.println("gender=" + gender);
        System.out.println("occupation=" + occupation);
        System.out.println("telephone=" + telephone);
        System.out.println("email=" + email);
        System.out.println("mailaddress=" + mailaddress);
        System.out.println("zipcode=" + zipcode);
        System.out.println("qq=" + qq);
        System.out.println("lastLoginTime=" + lastLoginTime);
        System.out.println("lastLoginIp=" + lastLoginIp);
        return null;
    }
}

```

步骤四：创建 db.properties 文件，该文件封装了连接数据的信息

该文件内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#jdbc.driverClassName=com.mysql.jdbc.Driver
#jdbc.url=jdbc:mysql://localhost:3306/tts7
#jdbc.username=root
#jdbc.password=root

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=500
#<!-- 超时等待时间以毫秒为单位 6000 毫秒/1000 等于 60 秒 -->
dataSource.maxWait=1000

```

步骤五：创建 BaseDAO 类，在该类中封装创建数据库连接方法以及事务处理

代码如下所示：

```
package com.tarena.netctoss;
```

```
import org.apache.commons.dbcp.BasicDataSource;

import java.io.IOException;
import java.sql.SQLException;
import java.sql.Connection;
import java.util.Properties;

public class BaseDAO {
    private static BasicDataSource dataSource = null;
    public static Properties properties = new Properties();

    public BaseDAO() {
    }

    // 用于装载 db.properties 获得数据库参数
    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(BaseDAO.class.getClassLoader().getResourceAsStream(
                "com/tarena/netctoss/db.properties"));

        } catch (IOException e1) {
            e1.printStackTrace();
        }

        try {
            String                     driveClassName
dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
            dataSource.setPassword(password);

            // 初始化连接数
            if (initialSize != null)
                dataSource.setInitialSize(Integer.parseInt(initialSize));

            // 最小空闲连接
            if (minIdle != null)
                dataSource.setMinIdle(Integer.parseInt(minIdle));

            // 最大空闲连接
            if (maxIdle != null)
                dataSource.setMaxIdle(Integer.parseInt(maxIdle));

            // 超时回收时间(以毫秒为单位)
            if (maxWait != null)
                dataSource.setMaxWait(Long.parseLong(maxWait));

            // 最大连接数
        }
    }
}
```

```

        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}

public static void begin() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.setAutoCommit(false);
    } catch (Exception e) {
        System.out.println("开启事务失败!");
        e.printStackTrace();
    }
}

public static void commit() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.commit();
    } catch (Exception e) {
        System.out.println("提交事务失败!");
        e.printStackTrace();
    }
}

public static void rollback() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.rollback();
    } catch (Exception e) {
        System.out.println("回滚事务失败!");
        e.printStackTrace();
    }
}
}

```

步骤六：创建 AccountDAO 接口，该接口中定义了对数据的增删改查的功能

代码如下所示：

```

package com.tarena.netctoss;

import java.sql.SQLException;
import java.util.List;
import com.tarena.netctoss.entity.Account;

```

```

public interface AccountDAO {

    /**
     * 根据某个账务账号 ID 查询该账务账号的全部信息
     * @param id 账务账号 ID
     * @return 某个账务账号的全部信息，为一个 Account 对象
     * @throws SQLException
     */
    Account findById(Integer id) throws SQLException;
    /**
     * 查询所有的账务账号
     * @return 所有账务账号 返回 List 集合
     * @throws SQLException
     */
    List<Account> findAll() throws SQLException;
    /**
     * 新增账务账号
     * @param account 要添加的账务账号
     * @return 添加后账务账号，包含账号账号 ID
     * @throws SQLException
     */
    Account save(Account account) throws SQLException;
    /**
     * 修改某个账务账号的信息
     * @param account 要修改的账务账号
     * @return 返回修改后的账务账号
     * @throws SQLException
     */
    Account modify(Account account) throws SQLException;
    /**
     * 修改某个账务账号的状态
     * @param account 要修改状态的账务账号
     * @return 返回修改状态后的账务账号
     * @throws SQLException
     */
    Account modifyStatus(Account account) throws SQLException;
}

```

步骤七：创建 AccountDAOImpl 类

创建 AccountDAOImpl 类，该类继承自 BaseDAO，来直接获取创建连接的方法；另外，该类实现了 AccountDAO 接口，将该接口中的方法做出实现，代码如下所示：

```

package com.tarena.netctoss.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import com.tarena.netctoss.entity.Account;
import com.tarena.netctoss.AccountDAO;
import com.tarena.netctoss.BaseDAO;

public class AccountDAOImpl extends BaseDAO implements AccountDAO {

```

```

        private static final String SELECT_BY_ID = "SELECT ACCOUNT_ID, RECOMMENDER_ID,
LOGIN_NAME, LOGIN_PASSWD,"
        + " STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO,
BIRTHDATE, GENDER, OCCUPATION,"
        +
"TELEPHONE, EMAIL, MAILADDRESS, ZIPCODE, QQ, LAST_LOGIN_TIME, LAST_LOGIN_IP "
        + "FROM ACCOUNT WHERE ACCOUNT_ID=?";

        private static final String FIND_ALL = "SELECT ACCOUNT_ID, RECOMMENDER_ID,
LOGIN_NAME, LOGIN_PASSWD,"
        + " STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO,
BIRTHDATE, GENDER, OCCUPATION,"
        +
"TELEPHONE, EMAIL, MAILADDRESS, ZIPCODE, QQ, LAST_LOGIN_TIME, LAST_LOGIN_IP "
        + "FROM ACCOUNT";

        private static final String MODIFY = "UPDATE ACCOUNT SET LOGIN_PASSWD = ?,
REAL_NAME = ?, GENDER = ?, OCCUPATION = ?,"
        + "TELEPHONE = ?, EMAIL = ?, MAILADDRESS = ?, ZIPCODE = ?, QQ = ?"
        + "WHERE ACCOUNT_ID=?";

        private static final String UPDATE_STATUS = "UPDATE ACCOUNT SET STATUS = ?
WHERE ACCOUNT_ID=?";

        private static final String INSERT = "INSERT INTO ACCOUNT(ACCOUNT_ID,
RECOMMENDER_ID, LOGIN_NAME, LOGIN_PASSWD,"
        + " STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO,
BIRTHDATE, GENDER, OCCUPATION,"
        +
"TELEPHONE, EMAIL, MAILADDRESS, ZIPCODE, QQ, LAST_LOGIN_TIME, LAST_LOGIN_IP) "
        + "VALUES (ACCOUNT_SEQ.NEXTVAL, ?, ?, ?, '0', SYSDATE, NULL,
NULL, ?, ?, to_date(?,'YYYY-mm-dd'), ?, ?, ?, ?, ?, ?, SYSDATE, ?)";

    @Override
    public Account findById(Integer id) throws SQLException {
        Connection conn = getConnection();
        String sql = "SELECT * FROM ACCOUNT WHERE ACCOUNT_ID = ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        Account account = null;
        while (rs.next()) {
            account = new Account();
            account.setId(rs.getInt("ACCOUNT_ID"));
            account.setRecommenderId(rs.getInt("RECOMMENDER_ID"));
            account.setLoginName(rs.getString("LOGIN_NAME"));
            account.setLoginPasswd(rs.getString("LOGIN_PASSWD"));
            account.setStatus(rs.getString("STATUS"));
            account.setCreateDate(rs.getDate("CREATE_DATE"));
            account.setPauseDate(rs.getDate("PAUSE_DATE"));
            account.setCloseDate(rs.getDate("CLOSE_DATE"));
            account.setRealName(rs.getString("REAL_NAME"));
            account.setIdcardNo(rs.getString("IDCARD_NO"));
            account.setBirthdate(rs.getDate("BIRTHDATE"));
            account.setGender(rs.getString("GENDER"));
            account.setOccupation(rs.getString("OCCUPATION"));

            account.setTelephone(rs.getString("TELEPHONE"));
            account.setEmail(rs.getString("EMAIL"));
            account.setMailaddress(rs.getString("MAILADDRESS"));
            account.setZipcode(rs.getString("ZIPCODE"));
            account.setQq(rs.getString("QQ"));
            account.setLastLoginTime(rs.getDate("LAST_LOGIN_TIME"));
            account.setLastLoginIp(rs.getString("LAST_LOGIN_IP"));
        }
        return account;
    }
}

```

```

@Override
public List<Account> findAll() throws SQLException {
    // TODO Auto-generated method stub
    Connection conn = getConnection();
    String sql = FIND_ALL;
    PreparedStatement ps = conn.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
    Account account = null;
    List<Account> list = new ArrayList<Account>();
    while (rs.next()) {
        account = new Account();
        account.setId(rs.getInt("ACCOUNT ID"));
        account.setRecommenderId(rs.getInt("RECOMMENDER ID"));
        account.setLoginName(rs.getString("LOGIN NAME"));
        account.setLoginPasswd(rs.getString("LOGIN PASSWD"));
        account.setStatus(rs.getString("STATUS"));
        account.setCreateDate(rs.getDate("CREATE_DATE"));
        account.setPauseDate(rs.getDate("PAUSE_DATE"));
        account.setCloseDate(rs.getDate("CLOSE_DATE"));
        account.setRealName(rs.getString("REAL NAME"));
        account.setIdcardNo(rs.getString("IDCARD NO"));
        account.setBirthdate(rs.getDate("BIRTHDATE"));
        account.setGender(rs.getString("GENDER"));
        account.setOccupation(rs.getString("OCCUPATION"));

        account.setTelephone(rs.getString("TELEPHONE"));
        account.setEmail(rs.getString("EMAIL"));
        account.setMailaddress(rs.getString("MAILADDRESS"));
        account.setZipcode(rs.getString("ZIPCODE"));
        account.setQq(rs.getString("QQ"));
        account.setLastLoginTime(rs.getDate("LAST LOGIN TIME"));
        account.setLastLoginIp(rs.getString("LAST LOGIN IP"));

        list.add(account);
    }
    return list;
}

@Override
public Account save(Account account) throws SQLException {
    Connection conn = getConnection();
    String sql = INSERT;
    PreparedStatement ps = conn.prepareStatement(sql, new String[]
    { "account id" });

    Calendar c = Calendar.getInstance();
    c.setTime(account.getBirthdate());
    String birth = c.get(Calendar.YEAR) + "-" + c.get(Calendar.MONTH) + "-"
+ c.get(Calendar.DATE);

    ps.setInt(1, account.getRecommenderId());
    ps.setString(2, account.getLoginName());
    ps.setString(3, account.getLoginPasswd());
    ps.setString(4, account.getRealName());
    ps.setString(5, account.getIdcardNo());
    ps.setString(6, birth);
    ps.setString(7, account.getGender());
    ps.setString(8, account.getOccupation());
    ps.setString(9, account.getTelephone());
    ps.setString(10, account.getEmail());
    ps.setString(11, account.getMailaddress());
    ps.setString(12, account.getZipcode());
    ps.setString(13, account.getQq());
    ps.setString(14, account.getLastLoginIp());

    ps.executeUpdate();
    ResultSet rs = ps.getGeneratedKeys();
}

```

```

        rs.next();
        int id = rs.getInt(1);
        account.setId(id);

        return account;
    }

    @Override
    public Account modify(Account account) throws SQLException {
        Connection conn = getConnection();
        String sql = MODIFY; // 预先定义好的 SQL 语句

        //如果输入的 id 不存在，直接返回 null
        Account account_fromdb = this.findById(account.getId());
        if (account_fromdb == null){
            return null;
        }

        PreparedStatement ps = conn.prepareStatement(sql);

        ps.setString(1, account.getLoginPasswd());// 传入参数
        ps.setString(2, account.getRealName());
        ps.setString(3, account.getGender());
        ps.setString(4, account.getOccupation());
        ps.setString(5, account.getTelephone());
        ps.setString(6, account.getEmail());
        ps.setString(7, account.getMailaddress());
        ps.setString(8, account.getZipcode());
        ps.setString(9, account.getQq());
        ps.setInt(10, account.getId());

        int flag = ps.executeUpdate();

        return (flag > 0) ? account : null;
    }

    /**
     * private static final String UPDATE_STATUS = "UPDATE ACCOUNT SET STUTAS
     * = ? WHERE ACCOUNT_ID=?";
     */

    @Override
    public Account modifyStatus(Account account) throws SQLException {
        // TODO Auto-generated method stub
        Connection conn = getConnection();
        String sql = UPDATE_STATUS; // 预先定义好的 SQL 语句
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, account.getStatus());// 传入参数
        ps.setInt(2, account.getId());

        int flag = ps.executeUpdate();
        return (flag > 0) ? account : null;
    }
}

```

步骤八：创建 TestAccountDAO 类，用于测试功能是否实现

创建 TestAccountDAO 类，该类用于测试 AccountDAOImpl 所实现的方法的正确性，代码如下所示：

```
package com.tarena.netctoss;

import java.sql.SQLException;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

import com.tarena.netctoss.entity.Account;
import com.tarena.netctoss.impl.AccountDAOImpl;

public class TestAccountDAO {
    public static void main(String[] args) {
        TestAccountDAO test = new TestAccountDAO();
        //test.testSave();
        //test.testFindById(3);
        //test.testfindAll();
        //test.testModify();
        test.testModifyStatus();
    }

    public void testModifyStatus() {
        AccountDAO dao = new AccountDAOImpl();
        Account account = new Account();
        account.setId(3);
        account.setStatus("2");//0:开通;1:暂停;2:删除
        try {
            dao.modifyStatus(account);
            //检查更新结果
            Account a = dao.findById(3);
            System.out.println(a.toString());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void testModify() {
        Calendar c = Calendar.getInstance();
        c.set(1988, 6, 17);
        Date birth = c.getTime();

        System.out.println(birth);

        Account account = new Account();
        account.setId(3);
        account.setRecommenderId(3);
        account.setLoginName("hitmo");
        account.setLoginPasswd("123mo");
        account.setRealName("zsmo");
        account.setIdcardNo("007mo");
        account.setBirthdate(birth);
        account.setGender("1");
        account.setOccupation("studentmo");
        account.setTelephone("tel1234mo");
        account.setEmail("a@b.cmo");
        account.setMailaddress("126 avenue mo");
        account.setZipcode("1071mo");
        account.setQq("1111mo");
        account.setLastLoginIp("192.mo");
        AccountDAO dao = new AccountDAOImpl();
        try {
            account = dao.modify(account);
            System.out.println(account.toString());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```

public void testSave() {
    Calendar c = Calendar.getInstance();
    c.set(1980, 3, 12);
    Date birth = c.getTime();

    System.out.println(birth);

    Account account = new Account();
    account.setRecommenderId(1);
    account.setLoginName("hit");
    account.setLoginPasswd("123");
    account.setRealName("zs");
    account.setIdcardNo("007");
    account.setBirthdate(birth);
    account.setGender("0");
    account.setOccupation("student");
    account.setTelephone("tel1234");
    account.setEmail("a@b.c");
    account.setMailaddress("126 avenue");
    account.setZipcode("1071");
    account.setQq("1111");
    account.setLastLoginIp("192");
    AccountDAO dao = new AccountDAOImpl();
    try {
        dao.save(account);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void testFindById(Integer id) {
    AccountDAO dao = new AccountDAOImpl();
    try {
        Account account = dao.findById(id);
        account.toString();

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

public void testfindAll() {
    AccountDAO dao = new AccountDAOImpl();
    try {
        List<Account> list = dao.findAll();
        for(Account account : list){
            account.toString();
            System.out.println("-----");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

测试的过程中需要注意的是修改信息、修改状态以及根据 ID 查询，这些方法都需要传入已存在的账务账号 ID。

- **完整代码**

本案例中，创建 account_seq 序列、Account 的表以及插入 SQL 语句如下所示：

```
create sequence account seq;
```

```
create table account(
    account_id    number(9) constraint account_id_pk primary key,
    recommender_id number(9) constraint account_recommender_id_fk
        references account(account id),
    login name    varchar2(30) not null
        constraint account login name uk unique,
    login_passwd  varchar2(30) not null,
    status        char(1) constraint account_status_ck
        check (status in (0,1,2)) not null,
    create date   date default sysdate,
    pause date    date,
    close date    date,
    real name    varchar2(20) not null,
    idcard_no    char(18)    not null
        constraint account_incard_no unique,
    birthdate     date,
    gender        char(1) constraint account gender ck
        check (gender in (0,1)) not null,
    occupation   varchar2(50),
    telephone    varchar2(15) not null,
    email         varchar2(50),
    mailaddress   varchar2(200),
    zipcode       char(6),
    qq            varchar2(15),
    last login time date,
    last_login_ip varchar2(15)
);
INSERT INTO ACCOUNT(ACCOUNT_ID, RECOMMENDER_ID, LOGIN_NAME, LOGIN_PASSWD,
STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO, BIRTHDATE,
GENDER,OCCUPATION,TELEPHONE,EMAIL,MAILADDRESS,ZIPCODE,QQ,LAST_LOGIN_TIME,
LAST_LOGIN_IP)
VALUES  (ACCOUNT_SEQ.NEXTVAL,null,  'a',    '123',    '0',    SYSDATE,    NULL,
NULL,'zhangsan', '1234',
to date('1981-01-01','yyyy-mm-dd'), '0', 'job','123', 'a@b.c', 'test avenue',
'1223', '1234', SYSDATE, '192');
```

类 Account 的完整代码如下所示：

```
package com.tarena.netctoss.entity;

import java.util.Date;

public class Account {
    private int id;
    private int recommenderId;
    private String loginName;
    private String loginPasswd;
    private String status;
    private Date createDate;
    private Date pauseDate;
    private Date closeDate;
    private String realName;
    private String idcardNo;
    private Date birthdate;
    private String gender;
    private String occupation;
    private String telephone;
    private String email;
    private String mailaddress;
    private String zipcode;
    private String qq;
    private Date lastLoginTime;
    private String lastLoginIp;
```

```
public Account() {
    super();
}

public Account(int id, int recommenderId, String loginName,
               String loginPasswd, String status, Date createDate, Date pauseDate,
               Date closeDate, String realName, String idcardNo, Date birthdate,
               String gender, String occupation, String telephone, String email,
               String mailaddress, String zipcode, String qq, Date lastLoginTime,
               String lastLoginIp) {
    super();
    this.id = id;
    this.recommenderId = recommenderId;
    this.loginName = loginName;
    this.loginPasswd = loginPasswd;
    this.status = status;
    this.createDate = createDate;
    this.pauseDate = pauseDate;
    this.closeDate = closeDate;
    this.realName = realName;
    this.idcardNo = idcardNo;
    this.birthdate = birthdate;
    this.gender = gender;
    this.occupation = occupation;
    this.telephone = telephone;
    this.email = email;
    this.mailaddress = mailaddress;
    this.zipcode = zipcode;
    this.qq = qq;
    this.lastLoginTime = lastLoginTime;
    this.lastLoginIp = lastLoginIp;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public int getRecommenderId() {
    return recommenderId;
}
public void setRecommenderId(int recommenderId) {
    this.recommenderId = recommenderId;
}
public String getLoginName() {
    return loginName;
}
public void setLoginName(String loginName) {
    this.loginName = loginName;
}
public String getLoginPasswd() {
    return loginPasswd;
}
public void setLoginPasswd(String loginPasswd) {
    this.loginPasswd = loginPasswd;
}
public String getStatus() {
    return status;
}
public void setStatus(String status) {
    this.status = status;
}
public Date getCreateDate() {
    return createDate;
}
public void setCreateDate(Date createDate) {
    this.createDate = createDate;
}
```

```
public Date getPauseDate() {
    return pauseDate;
}
public void setPauseDate(Date pauseDate) {
    this.pauseDate = pauseDate;
}
public Date getCloseDate() {
    return closeDate;
}
public void setCloseDate(Date closeDate) {
    this.closeDate = closeDate;
}
public String getRealName() {
    return realName;
}
public void setRealName(String realName) {
    this.realName = realName;
}
public String getIdcardNo() {
    return idcardNo;
}
public void setIdcardNo(String idcardNo) {
    this.idcardNo = idcardNo;
}
public Date getBirthdate() {
    return birthdate;
}
public void setBirthdate(Date birthdate) {
    this.birthdate = birthdate;
}
public String getGender() {
    return gender;
}
public void setGender(String gender) {
    this.gender = gender;
}
public String getOccupation() {
    return occupation;
}
public void setOccupation(String occupation) {
    this.occupation = occupation;
}
public String getTelephone() {
    return telephone;
}
public void setTelephone(String telephone) {
    this.telephone = telephone;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public String getMailaddress() {
    return mailaddress;
}
public void setMailaddress(String mailaddress) {
    this.mailaddress = mailaddress;
}
public String getZipcode() {
    return zipcode;
}
public void setZipcode(String zipcode) {
    this.zipcode = zipcode;
}
public String getQq() {
    return qq;
}
```

```

    }
    public void setQq(String qq) {
        this.qq = qq;
    }
    public Date getLastLoginTime() {
        return lastLoginTime;
    }
    public void setLastLoginTime(Date lastLoginTime) {
        this.lastLoginTime = lastLoginTime;
    }
    public String getLastLoginIp() {
        return lastLoginIp;
    }
    public void setLastLoginIp(String lastLoginIp) {
        this.lastLoginIp = lastLoginIp;
    }
    public String toString(){
        System.out.println("id=" + id);
        System.out.println("recommenderId=" + recommenderId);
        System.out.println("loginName=" + loginName);
        System.out.println("loginPasswd=" + loginPasswd);
        System.out.println("status=" + status);
        System.out.println("createDate=" + createDate);
        System.out.println("pauseDate=" + pauseDate);
        System.out.println("closeDate=" + closeDate);
        System.out.println("realName=" + realName);
        System.out.println("idcardNo=" + idcardNo);
        System.out.println("birthdate=" + birthdate);
        System.out.println("gender=" + gender);
        System.out.println("occupation=" + occupation);
        System.out.println("telephone=" + telephone);
        System.out.println("email=" + email);
        System.out.println("mailaddress=" + mailaddress);
        System.out.println("zipcode=" + zipcode);
        System.out.println("qq=" + qq);
        System.out.println("lastLoginTime=" + lastLoginTime);
        System.out.println("lastLoginIp=" + lastLoginIp);
        return null;
    }
}

```

db.properties 文件内容如下所示：

```

jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:orcl
jdbc.username=scott
jdbc.password=tiger

#jdbc.driverClassName=com.mysql.jdbc.Driver
#jdbc.url=jdbc:mysql://localhost:3306/tts7
#jdbc.username=root
#jdbc.password=root

#<!-- 初始化连接 -->
dataSource.initialSize=10
#<!-- 最大空闲连接 -->
dataSource.maxIdle=20
#<!-- 最小空闲连接 -->
dataSource.minIdle=5
#最大连接数量
dataSource.maxActive=500
#<!-- 超时等待时间以毫秒为单位 (6000 毫秒/1000 等于 60 秒) -->
dataSource.maxWait=1000

```

BaseDAO 类的完整代码如下所示：

```
package com.tarena.netctoss;

import org.apache.commons.dbcp.BasicDataSource;

import java.io.IOException;
import java.sql.SQLException;
import java.sql.Connection;
import java.util.Properties;

public class BaseDAO {
    private static BasicDataSource dataSource = null;
    public static Properties properties = new Properties();

    public BaseDAO() {
    }

    // 用于装载 db.properties 获得数据库参数
    public static void init() {

        Properties dbProps = new Properties();
        // 取配置文件可以根据实际的不同修改
        try {

            dbProps.load(BaseDAO.class.getClassLoader().getResourceAsStream(
                "com/tarena/netctoss/db.properties"));

        } catch (IOException e1) {
            e1.printStackTrace();
        }

        try {
            String                     driveClassName
dbProps.getProperty("jdbc.driverClassName");
            String url = dbProps.getProperty("jdbc.url");
            String username = dbProps.getProperty("jdbc.username");
            String password = dbProps.getProperty("jdbc.password");

            String initialSize = dbProps.getProperty("dataSource.initialSize");
            String minIdle = dbProps.getProperty("dataSource.minIdle");
            String maxIdle = dbProps.getProperty("dataSource.maxIdle");
            String maxWait = dbProps.getProperty("dataSource.maxWait");
            String maxActive = dbProps.getProperty("dataSource.maxActive");

            dataSource = new BasicDataSource();
            dataSource.setDriverClassName(driveClassName);
            dataSource.setUrl(url);
            dataSource.setUsername(username);
            dataSource.setPassword(password);

            // 初始化连接数
            if (initialSize != null)
                dataSource.setInitialSize(Integer.parseInt(initialSize));

            // 最小空闲连接
            if (minIdle != null)
                dataSource.setMinIdle(Integer.parseInt(minIdle));

            // 最大空闲连接
            if (maxIdle != null)
                dataSource.setMaxIdle(Integer.parseInt(maxIdle));

            // 超时回收时间(以毫秒为单位)
            if (maxWait != null)
```

```

        dataSource.setMaxWait(Long.parseLong(maxWait));

        // 最大连接数
        if (maxActive != null) {
            if (!maxActive.trim().equals("0"))
                dataSource.setMaxActive(Integer.parseInt(maxActive));
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("创建连接池失败!请检查设置!!!!");
    }
}

public static synchronized Connection getConnection() throws SQLException {
    if (dataSource == null) {
        init();
    }
    Connection conn = null;
    if (dataSource != null) {
        conn = dataSource.getConnection();
    }
    return conn;
}

public static void begin() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.setAutoCommit(false);
    } catch (Exception e) {
        System.out.println("开启事务失败!");
        e.printStackTrace();
    }
}

public static void commit() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.commit();
    } catch (Exception e) {
        System.out.println("提交事务失败!");
        e.printStackTrace();
    }
}

public static void rollback() {
    try {
        Connection conn = getConnection();
        if (conn != null)
            conn.rollback();
    } catch (Exception e) {
        System.out.println("回滚事务失败!");
        e.printStackTrace();
    }
}
}

```

AccountDAO 接口的完整代码如下所示：

```

package com.tarena.netctoss;

import java.sql.SQLException;
import java.util.List;
import com.tarena.netctoss.entity.Account;

```

```

public interface AccountDAO {

    /**
     * 根据某个账务账号 ID 查询该账务账号的全部信息
     * @param id 账务账号 ID
     * @return 某个账务账号的全部信息，为一个 Account 对象
     * @throws SQLException
     */
    Account findById(Integer id) throws SQLException;
    /**
     * 查询所有的账务账号
     * @return 所有账务账号 返回 List 集合
     * @throws SQLException
     */
    List<Account> findAll() throws SQLException;
    /**
     * 新增账务账号
     * @param account 要添加的账务账号
     * @return 添加后账务账号，包含账号账号 ID
     * @throws SQLException
     */
    Account save(Account account) throws SQLException;
    /**
     * 修改某个账务账号的信息
     * @param account 要修改的账务账号
     * @return 返回修改后的账务账号
     * @throws SQLException
     */
    Account modify(Account account) throws SQLException;
    /**
     * 修改某个账务账号的状态
     * @param account 要修改状态的账务账号
     * @return 返回修改状态后的账务账号
     * @throws SQLException
     */
    Account modifyStatus(Account account) throws SQLException;
}

```

AccountDAOImpl 类的完整代码如下所示：

```

package com.tarena.netctoss.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import com.tarena.netctoss.entity.Account;
import com.tarena.netctoss.AccountDAO;
import com.tarena.netctoss.BaseDAO;

public class AccountDAOImpl extends BaseDAO implements AccountDAO {

    private static final String SELECT_BY_ID = "SELECT ACCOUNT_ID, RECOMMENDER_ID," +
        "LOGIN_NAME, LOGIN_PASSWD," +
        "STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO," +
        "BIRTHDATE, GENDER, OCCUPATION," +
        "TELEPHONE, EMAIL, MAILADDRESS, ZIPCODE, QQ, LAST_LOGIN_TIME, LAST_LOGIN_IP " +
        "FROM ACCOUNT WHERE ACCOUNT_ID=?";
}

```

```

        private static final String FIND_ALL = "SELECT ACCOUNT_ID, RECOMMENDER_ID,
LOGIN_NAME, LOGIN_PASSWD,"
        + " STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO,
BIRTHDATE, GENDER, OCCUPATION,"
        +
"TELEPHONE,EMAIL,MAILADDRESS,ZIPCODE,QQ,LAST_LOGIN_TIME,LAST_LOGIN_IP "
        + "FROM ACCOUNT";

        private static final String MODIFY = "UPDATE ACCOUNT SET LOGIN_PASSWD = ?,
REAL_NAME = ?, GENDER = ?, OCCUPATION = ?,"
        + "TELEPHONE = ?,EMAIL = ?,MAILADDRESS = ?,ZIPCODE = ?,QQ = ?"
        + "WHERE ACCOUNT_ID=?";

        private static final String UPDATE_STATUS = "UPDATE ACCOUNT SET STATUS = ?
WHERE ACCOUNT_ID=?";

        private static final String INSERT = "INSERT INTO ACCOUNT(ACCOUNT_ID,
RECOMMENDER_ID, LOGIN_NAME, LOGIN_PASSWD,"
        + " STATUS, CREATE_DATE, PAUSE_DATE, CLOSE_DATE, REAL_NAME, IDCARD_NO,
BIRTHDATE, GENDER, OCCUPATION,"
        +
"TELEPHONE,EMAIL,MAILADDRESS,ZIPCODE,QQ,LAST_LOGIN_TIME,LAST_LOGIN_IP) "
        + "VALUES (ACCOUNT_SEQ.NEXTVAL,?, ?, ?, '0', SYSDATE, NULL,
NULL,?, ?, to_date(?,'yyyy-mm-dd'), ?, ?, ?, ?, ?, ?, SYSDATE, ?)";

@Override
public Account findById(Integer id) throws SQLException {
    Connection conn = getConnection();
    String sql = SELECT_BY_ID;
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setInt(1, id);
    ResultSet rs = ps.executeQuery();
    Account account = null;
    while (rs.next()) {
        account = new Account();
        account.setId(rs.getInt("ACCOUNT_ID"));
        account.setRecommenderId(rs.getInt("RECOMMENDER_ID"));
        account.setLoginName(rs.getString("LOGIN_NAME"));
        account.setLoginPasswd(rs.getString("LOGIN_PASSWD"));
        account.setStatus(rs.getString("STATUS"));
        account.setCreateDate(rs.getDate("CREATE_DATE"));
        account.setPauseDate(rs.getDate("PAUSE_DATE"));
        account.setCloseDate(rs.getDate("CLOSE_DATE"));
        account.setRealName(rs.getString("REAL_NAME"));
        account.setIdcardNo(rs.getString("IDCARD_NO"));
        account.setBirthdate(rs.getDate("BIRTHDATE"));
        account.setGender(rs.getString("GENDER"));
        account.setOccupation(rs.getString("OCCUPATION"));

        account.setTelephone(rs.getString("TELEPHONE"));
        account.setEmail(rs.getString("EMAIL"));
        account.setMailaddress(rs.getString("MAILADDRESS"));
        account.setZipcode(rs.getString("ZIPCODE"));
        account.setQq(rs.getString("QQ"));
        account.setLastLoginTime(rs.getDate("LAST_LOGIN_TIME"));
        account.setLastLoginIp(rs.getString("LAST_LOGIN_IP"));
    }
    return account;
}

@Override
public List<Account> findAll() throws SQLException {
    // TODO Auto-generated method stub
    Connection conn = getConnection();
    String sql = FIND_ALL;
    PreparedStatement ps = conn.prepareStatement(sql);
    ResultSet rs = ps.executeQuery();
}

```

```

    Account account = null;
    List<Account> list = new ArrayList<Account>();
    while (rs.next()) {
        account = new Account();
        account.setId(rs.getInt("ACCOUNT_ID"));
        account.setRecommenderId(rs.getInt("RECOMMENDER_ID"));
        account.setLoginName(rs.getString("LOGIN NAME"));
        account.setLoginPasswd(rs.getString("LOGIN PASSWD"));
        account.setStatus(rs.getString("STATUS"));
        account.setCreateDate(rs.getDate("CREATE_DATE"));
        account.setPauseDate(rs.getDate("PAUSE_DATE"));
        account.setCloseDate(rs.getDate("CLOSE DATE"));
        account.setRealName(rs.getString("REAL NAME"));
        account.setIdcardNo(rs.getString("IDCARD NO"));
        account.setBirthdate(rs.getDate("BIRTHDATE"));
        account.setGender(rs.getString("GENDER"));
        account.setOccupation(rs.getString("OCCUPATION"));

        account.setTelephone(rs.getString("TELEPHONE"));
        account.setEmail(rs.getString("EMAIL"));
        account.setMailaddress(rs.getString("MAILADDRESS"));
        account.setZipcode(rs.getString("ZIPCODE"));
        account.setQq(rs.getString("QQ"));
        account.setLastLoginTime(rs.getDate("LAST_LOGIN_TIME"));
        account.setLastLoginIp(rs.getString("LAST LOGIN IP"));

        list.add(account);
    }
    return list;
}

@Override
public Account save(Account account) throws SQLException {
    Connection conn = getConnection();
    String sql = INSERT;
    PreparedStatement ps = conn.prepareStatement(sql, new String[]
    { "account id" });

    Calendar c = Calendar.getInstance();
    c.setTime(account.getBirthdate());
    String birth = c.get(Calendar.YEAR) + "-" + c.get(Calendar.MONTH) + "-"
    + c.get(Calendar.DATE);

    ps.setInt(1, account.getRecommenderId());
    ps.setString(2, account.getLoginName());
    ps.setString(3, account.getLoginPasswd());
    ps.setString(4, account.getRealName());
    ps.setString(5, account.getIdcardNo());
    ps.setString(6, birth);
    ps.setString(7, account.getGender());
    ps.setString(8, account.getOccupation());
    ps.setString(9, account.getTelephone());
    ps.setString(10, account.getEmail());
    ps.setString(11, account.getMailaddress());
    ps.setString(12, account.getZipcode());
    ps.setString(13, account.getQq());
    ps.setString(14, account.getLastLoginIp());

    ps.executeUpdate();
    ResultSet rs = ps.getGeneratedKeys();
    rs.next();
    int id = rs.getInt(1);
    account.setId(id);

    return account;
}

```

```

@Override
public Account modify(Account account) throws SQLException {
    Connection conn = getConnection();
    String sql = MODIFY; // 预先定义好的 SQL 语句

    //如果输入的 id 不存在, 直接返回 null
    Account account_fromdb = this.findById(account.getId());
    if (account_fromdb == null){
        return null;
    }

    PreparedStatement ps = conn.prepareStatement(sql);

    ps.setString(1, account.getLoginPasswd());// 传入参数
    ps.setString(2, account.getRealName());
    ps.setString(3, account.getGender());
    ps.setString(4, account.getOccupation());
    ps.setString(5, account.getTelephone());
    ps.setString(6, account.getEmail());
    ps.setString(7, account.getMailaddress());
    ps.setString(8, account.getZipcode());
    ps.setString(9, account.getQq());
    ps.setInt(10, account.getId());

    int flag = ps.executeUpdate();

    return (flag > 0) ? account : null;
}

/**
 * private static final String UPDATE_STATUS = "UPDATE ACCOUNT SET STUTAS
= ? WHERE ACCOUNT_ID=?";

*/
@Override
public Account modifyStatus(Account account) throws SQLException {
    // TODO Auto-generated method stub
    Connection conn = getConnection();
    String sql = UPDATE_STATUS; // 预先定义好的 SQL 语句
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setString(1, account.getStatus());// 传入参数
    ps.setInt(2, account.getId());

    int flag = ps.executeUpdate();
    return (flag > 0) ? account : null;
}
}

```

TestAccountDAO 类的完整代码如下所示：

```

package com.tarena.netctoss;

import java.sql.SQLException;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

import com.tarena.netctoss.entity.Account;
import com.tarena.netctoss.impl.AccountDAOImpl;

public class TestAccountDAO {
    public static void main(String[] args) {
        TestAccountDAO test = new TestAccountDAO();
        //test.testSave();
    }
}

```

```

//test.testFindById(3);
//test.testfindAll();
//test.testModify();
test.testModifyStatus();
}

public void testModifyStatus() {
    AccountDAO dao = new AccountDAOImpl();
    Account account = new Account();
    account.setId(3);
    account.setStatus("2");//0: 开通; 1: 暂停; 2: 删除
    try {
        dao.modifyStatus(account);
        //检查更新结果
        Account a = dao.findById(3);
        System.out.println(a.toString());
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
public void testModify() {
    Calendar c = Calendar.getInstance();
    c.set(1988, 6, 17);
    Date birth = c.getTime();

    System.out.println(birth);

    Account account = new Account();
    account.setId(3);
    account.setRecommenderId(3);
    account.setLoginName("hitmo");
    account.setLoginPasswd("123mo");
    account.setRealName("zsмо");
    account.setIdcardNo("007mo");
    account.setBirthdate(birth);
    account.setGender("1");
    account.setOccupation("studentmo");
    account.setTelephone("tel1234mo");
    account.setEmail("a@b.cmo");
    account.setMailaddress("126 avenue mo");
    account.setZipcode("1071mo");
    account.setQq("1111mo");
    account.setLastLoginIp("192.mo");
    AccountDAO dao = new AccountDAOImpl();
    try {
        account = dao.modify(account);
        System.out.println(account.toString());
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void testSave() {
    Calendar c = Calendar.getInstance();
    c.set(1980, 3, 12);
    Date birth = c.getTime();

    System.out.println(birth);

    Account account = new Account();
    account.setRecommenderId(1);
    account.setLoginName("hit");
    account.setLoginPasswd("123");
    account.setRealName("zs");
    account.setIdcardNo("007");
    account.setBirthdate(birth);
    account.setGender("0");
}

```

```
account.setOccupation("student");
account.setTelephone("tel1234");
account.setEmail("a@b.c");
account.setMailaddress("126 avenue");
account.setZipcode("1071");
account.setQq("1111");
account.setLastLoginIp("192");
AccountDAO dao = new AccountDAOImpl();
try {
    dao.save(account);
} catch (SQLException e) {
    e.printStackTrace();
}
}

public void testFindById(Integer id) {
    AccountDAO dao = new AccountDAOImpl();
    try {
        Account account = dao.findById(id);
        account.toString();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void testfindAll() {
    AccountDAO dao = new AccountDAOImpl();
    try {
        List<Account> list = dao.findAll();
        for(Account account : list){
            account.toString();
            System.out.println("-----");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



课后作业

1. 实现将某一个部门 (Dept) 的员工 (Emp) 工资的提升

本案例的详细要求如下：

1. 为职位 (job) 为 “ANALYST” 的员工，工资提升 20%。
2. 为职位 (job) 为 “MANAGER” 的员工，工资提升 30%。
3. 要求某部门下的以上两个职位的员工工资，要么工资全部提升成功，要么工资全部提升失败。

2. 批量插入 Dept 数据

向 Dept 表中批量插入 100 条数据，需要插入数据的列为 deptno、dname，这两列的数据要求如下：

1. deptno 列的数据通过序列 dept_seq 自动生成；
2. dname 列的数据为字符串，格式为：“name” + 循环次数 i。

3. 向 Emp 表中插入一个团队成员

向 Emp 表中插入一个团队成员，该团队的成员信息如图-1 所示。

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	tom	manager	7839	2014/5/1	5000	300	30
2	marry	clerk	1	2014/5/28	3000		30
3	terry	salesman	1	2014/5/29	2500	200	30
4	jim	salesman	1	2014/5/26	2500	200	30

图-1

从图-1 可以看出 tom 为这个团队的管理者，其他三位员工的管理者 ID(mgr) 都为 1，而 1 是管理者 tom 的员工编号 (empno)。

要求向 Emp 表插入以上四个员工的信息。职员 marry、terry、jim 的管理者 ID(mgr) 为刚刚插入 Emp 表的管理者 tom 的员工编号 (empno) 的数据。另外，Emp 表的主键列 empno 的数据通过序列 emp_seq 获得。

4. 实现对 Dept 数据的分页查询 (Oracle 和 MySQL)

使用 JDBC 分别连接 Oracle 数据库和 MySQL 数据库，实现对 Dept 表数据的分页查询功能。

5. 完成 NetCTOSS 项目中，权限管理模块的角色的 DAO 设计及实现

详细要求如下：

1. 查询所有角色信息。
2. 添加某个角色。
3. 修改某个角色的角色名称。

达内IT培训集团