JDBC 规范

主讲： 郭鑫

北京动力节点教育科技有限公司
动力节点课程讲义
DONGLIJIEDIANKECHENGJIANGYI
www.bjpowernode.com

第1章JDBC 概述

1.1接口作用

1.1.1 作用

A、使程序“可插拔”，易于扩展；

B、接口是一套规范，程序员应该面向接口去调用，而不需要关心接口底层的具体实现

1.1.2 举例：餐厅菜单点餐

（1） 环境变量配置：

A、在 path 值后面添加 C:\Program Files\Java\jdk1.7.0_75\bin

（2） 定义菜单：饭馆中菜单为抽象接口

```
public interface FoodMenu
{
    void food1();
    void food2();
    void food3();

}
```

A、 菜单接口 Interface： FoodMenu.java

a、 饭馆负责制定的规范、标准

b、 厨师面向菜单实现

c、 顾客面向菜单调用

d、 任何一个接口都有调用方、实现方

e、 面向接口编程的目的是：调用方和实现方的真正解耦合

（3） 定义实现方：厨师是菜单的实现方

A、 厨师 1 面向菜单实现：Cooker1.java

```
public class Cooker1 implements FoodMenu
{
    public void food1(){
        System.out.println("Cooker1 food1");
    }

    public void food2(){
        System.out.println("Cooker1 food2");
    }

    public void food3(){
        System.out.println("Cooker1 food3");
    }
}
```

B、 厨师 2 面向菜单实现：Cooker2.java

```
public class Cooker2 implements FoodMenu
{
    public void food1(){
        System.out.println("Cooker2 food1");
    }

    public void food2(){
        System.out.println("Cooker2 food2");
    }

    public void food3(){
        System.out.println("Cooker2 food3");
    }
}
```

C、 厨师 3 面向菜单实现：Cooker3.java

```
public class Cooker3 implements FoodMenu
{
    public void food1(){
        System.out.println("Cooker3 food1");
    };

    public void food2(){
        System.out.println("Cooker3 food2");
    };

    public void food3(){
        System.out.println("Cooker3 food3");
    };

}
```

（4） 定义调用方：顾客是菜单的调用方

```
public class Customer
{
    //顾客点菜的方案
    //实例变量
    //关联关系
    private FoodMenu foodMenu;

    public void setFoodMenu(FoodMenu foodMenu){
        this.foodMenu = foodMenu;
    }

    //顾客手拿菜单
    public void order(){
        foodMenu.food1();
        foodMenu.food2();
        foodMenu.food3();
    }
}
```

A、 顾客面向的不是具体的厨师

B、 顾客面向抽象的菜单点菜

（5） 测试调用：Test.java

```
public class Test
{
    public static void main(String[] args){
        //定义一个顾客
        Customer zhangsan = new Customer();

        //餐厅所有菜单
        FoodMenu foodMenu1 = new Cooker1();
        FoodMenu foodMenu2 = new Cooker2();

        //顾客手持菜单
        zhangsan.setFoodMenu(foodMenu1);

        //顾客开始点菜
        zhangsan.order();
    }
}
```

1.1.3 接口作用总结：

A、面向抽象编程，面向接口编程，尽量使用多态机制

B、可以提高程序的扩展力，降低程序的耦合度

C、让程序变得具有很强的可接插特性

1.2 JDBC 是什么

1.2.1 Java Data Base Connectivity，java 语言连接数据库系统；

1.2.2 JDBC 接口是一套 class 文件，谁制定？

（1） SUN 公司负责制定 JDBC 规范

1.2.3 JDBC 接口调用方是谁？

（1） java 程序员：连接数据库并且操作数据库

1.2.4 JDBC 接口的实现类谁写，实现方是谁？

（1） 各大数据库厂商的 java 程序员负责编写 SUN 公司制定的 JDBC 接口的实现类

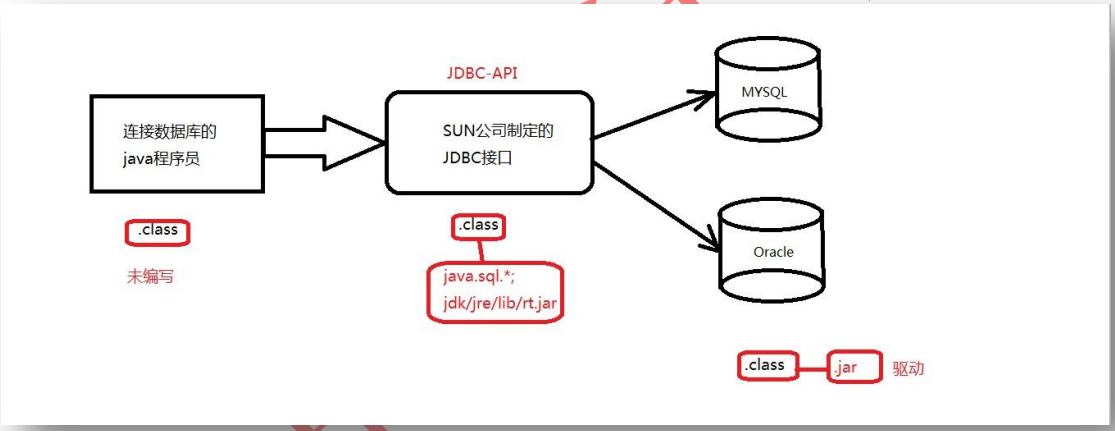
1.2.5 连接数据库驱动

（1） 各大数据库厂商编写的 JDBC 接口的实现类，编译之后将这些实现类打成 jar 包并且发布，

（2） 所有连接数据库的 java 程序员需要从官网上下载这些 jar 再使用，这些 jar 包通常被我们称为连接数据库的驱动；

（3） 那么我们 java 程序员要想连接 MySQL 数据库，需要先从官网下载连接 MySQL 数据库的专用驱动 jar 包，并且需要将这些 jar 包配置到环境变量 classpath 中

1.3 画图描述 JDBC 原理



1.4什么是 API

1.4.1 定义：Application Programming Interface 应用程序编程接口，就是一套类库；【这里的接口不是专指 java 中的 interface】

1.4.2 java 中的 API 包括三个元素：

（1） API 字节码

（2） API 源码

（3） API 帮助文档

以上三个元素的版本要保持一致；

A、例如：JDK1.7 源码 —> JDK1.7 字节码 —> JDK1.7 帮助文档

1.5JDBC 在开发中的作用

1.5.1 在实际开发中，数据库的产品特别多的，例如：MySQL、Oracle、DB2、Sybase、SQLServer 等等

1.5.2 每一个数据库的底层实现原理都是不同的，那么我们 java 程序员去连接数据库的时候，需要关心数据库底层具体是怎样实现的吗？我们需要关心数据库的细节吗？

1.5.3 我们若要关心数据库的底层实现原理，那么 java 程序员就需要学习多套连接数据库的程序。MySQL 专门一套、Oracle 专门一套等等，这是不可能的

1.5.4 java 程序员连接数据库的时候，不需要关心底层数据库的实现原理，应该给 java 程序员提供一套标准的接口

1.5.5 java 程序员应该面向 JDBC 接口去调用，完成数据库的操作。

1.5.6 也就是说我们 java 程序员不需要关心底层是什么数据库，只要编写一套通用的 java 程序即可连接各种数据库。

1.6 编写程序模拟 JDBC 本质

1.6.1 第一步：JDBC.java -SUN 负责制定 JDBC 接口

```
/*
    SUN公司负责制定的JDBC接口规范。
*/
public interface JDBC
{
    /*
        SUN制定的连接数据库的方法
    */
    void getConnection();
}
```

（1） SUN 负责制定 JDBC 接口规范、标准

（2） SUN 制定了连接数据库方法 getConnecton()

1.6.2 第二步：MySQL.java-数据库厂商对 JDBC 接口进行实现

MySQL 数据库厂商对 JDBC 接口进行实现

```
/*
    MySQL公司实现的SUN公司的接口规范
*/
public class MySQL implements JDBC
{
    public void getConnection(){
        System.out.println("连接MySQL DB ...");
    }
}
```

Oracle 数据库厂商对 JDBC 接口进行实现

```
public class Oracle implements JDBC
{
    public void getConnection(){
        System.out.println("连接Oracle  DB  . . . . ");
    }
}
```

（1） 数据库厂商对 JDBC 接口进行了实现

（2） 实现之后被打成 jar 包，称为驱动

1.6.3 第三步：JdbcProgrammer1.java -Java 程序员调用 JDBC 接口

（1） 第一种方式：创建 JDBC jdbc = new MySQL()对象进行连接

```
public class JdbcProgrammer1
{
    public static void main(String[] args){

        //连接数据库的时候不需要关心底层具体的数据
        JDBC jdbc = new MySQL();
        //JDBC jdbc = new Oracle();

        //面向接口调用
        jdbc.getConnection();
    }
}
```

A、 Java 程序员不需要关心底层具体的数据库

B、 只需要关心 JDBC 接口提供了哪些方法可以连接数据库以及操作数据库

（2） 问题：以上操作还是面向的具体对象，我们通过反射机制将

JDBC jdbc = new Oracle()进行封装；

A、 第一步：编写静态代码块 static{ ....}，类加载时自动运行

a、 创建一个 User.java 类在 com.bjpowernode.reflect 包下

```
package com.bjpowernode.reflect;

public class User {
    static{
        System.out.println("User 类加载了....");
    }
}
```

类的包名

B、 第二步：通过 Class.forName(“com.bjpowernode.reflect.User”)创建 User 对象

a、 创建 ReflectTest.java 来测试

```
package com.bjpowernode.reflect;

public class ReflectTest {

    public static void main(String[] args) throws Exception {
        Class.forName("com.bjpowernode.reflect.User");
    }

}
```

（3） 创建 IOTest.java 读取配置文件，通过反射机制创建对象

A、创建 db.properties 文件

```
a、   driver = com.mysql.jdbc.Driver

b、   #driver = oracle.jdbc.driver.OracleDriver

c、   url = jdbc:mysql://localhost:3366/bjpowernode

d、   user = root

e、password =123
```

B、 通过 FileReader reader = new FileReader(“db.properties”)读取配置文件

C、 创建属性对象 Properties pro = new Properties();

D、 调用 pro.load(reader)方法将配置文件中数据加载到 Map 集合中

E、 关闭流 reader.close();

F、 通过 pro.getProperty(key)获取属性 value

1.7 JDBC 开发前的准备

1.7.1 配置驱动到 classpath 中：

（1） 目的：在 EditPlus 中开发的 JDBC 连接可以使用 Mysql 驱动

（2） CLASSPATH =

.;E:\course\02-JDBC\JDBC_Connection\mysql-connector-java-5.1.7-bin.jar

1.7.2 URL 是什么？

（1） 定义：统一资源定位符

（2） URL 代表网络中的某个资源的绝对路径

（3） 通过 URL 可以定位网络中的资源

（4） URL 由：协议、IP、端口号、资源名称、……

A、 协议是：提前制定好的通信数据格式、规范，按照这种特定格式发送数据包，对方收到数据包之后，也按照这种规范解析这个数据包，获取有价值数据；

B、 IP 作用：定位计算机

C、 端口号作用：定位服务

（5） 数据库 URL：

A、 MySQL 数据库：jdbc:mysql://localhost:3366/bjpowernode

a、 jdbc:mysql:// 协议

b、 localhost IP

c、 3366 端口号

d、 bjpowernode 数据库名称

B、 Oracle 数据库：jdbc:oracle:thin:@localhost:1521:bjpowernode

a、 jdbc:oracle:thin:@ 协议

b、 locahost IP

c、 1521 端口号

d、 bjpowernode 数据库名称

第2章 JDBC 编程六步曲

2.1 第一步：注册驱动[ed]

```
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCTest01
{
    public static void main(String[] args){

        try{
            //1、注册驱动

            //1.1 创建驱动对象
            //java.sql.Driver driver = new com.mysql.jdbc.Driver();

            Driver driver = new com.mysql.jdbc.Driver();

            //1.2 注册驱动
            DriverManager.registerDriver(driver);

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


    }
}
```

2.1.1 创建驱动对象：告知 JDBC 我们即将连接哪个数据库

2.1.2 通过 DriverManager.registerDriver(driver)注册驱动

2.2 第二步：获取数据库连接 URL

```
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;

public class JDBCTest02
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            //1.1创建驱动对象
            Driver driver = new com.mysql.jdbc.Driver();
            //1.2注册驱动
            DriverManager.registerDriver(driver);

            //2、获取数据库连接
            /*
                URL统一资源定位
            */
            String url = "jdbc:mysql://localhost:3366/bjpowernode";
            String user = "root";
            String password = "123";
            Connection conn = DriverManager.getConnection(url,user,password);

            //MySQL数据库对java.sql.Connecton接口的实现类的完整类名com.mysql.jdbc.JDBC4Connection
            System.out.println(conn);//com.mysql.jdbc.JDBC4Connection@1a477b7

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

2.2.1 通过 DriverManager.getConnection(url,user,pwd)获取连接

2.2.2 重点：Connection 连接对象不能随意创建，最后使用完要手动关闭

2.2.3 MySQL 数据库支持最多的连接数量为 100

2.3 第三步：获取数据库操作对象

```
//2.获取数据库连接
Connection conn = DriverManager.getConnection(url,user,password);
```

2.3.1 一个数据库连接对象可以创建多个数据库操作对象

2.3.2 通过 conn.createStatement();

```
/*
    JDBC编程六步曲：
        1、注册驱动
        2、获取连接
        3、获取数据库操作对象
        4、执行SQL语句
        5、处理查询结果集
        6、释放资源

    JDBC编程第三步：获取数据库操作对象
    拿到这个对象就可以通过这个对象执行SQL语句

*/

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

public class JDBCTest03
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            //1.1创建驱动对象
            Driver driver = new com.mysql.jdbc.Driver();
            //1.2注册驱动
            DriverManager.registerDriver(driver);

            //2、获取数据库连接
            /*
                URL统一资源定位
            */
            String url = "jdbc:mysql://localhost:3366/bjpowernode";
            String user = "root";
            String password = "123";
            Connection conn = DriverManager.getConnection(url,user,password);


            //3、获取数据库操作对象
            //一个连接对象可以创建多个数据库操作对象
            //通过内存地址可以看出MySQL驱动对Statement接口的实现完整类名
            Statement stmt = conn.createStatement();
            System.out.println(stmt);

            Statement stmt2 = conn.createStatement();
            System.out.println(stmt2);

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

2.4 第四步：执行 SQL 语句之 DML 语句

2.4.1 通过数据库操作对象 stmt.executeUpdate(sql);编译执行 SQL

2.4.2 JDBC 编写 SQL 语句不需要以分号结尾

2.4.3 数据库管理系统会将编写好的 SQL 语句编译并执行

```
/*
    JDBC编程六步曲：
        1、注册驱动
        2、获取连接
        3、获取数据库操作对象
        4、执行SQL语句
        5、处理查询结果集
        6、释放资源

    JDBC编程第四步：执行DML语句
    以下程序主要执行DML语句

*/
import java.sql.Driver;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.SQLException;
public class JDBCTest04
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
            //2、获取连接
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");
            //3、获取数据库操作对象
            Statement stmt = conn.createStatement();
            //4、执行SQL语句：DML:insert、update、delete
            //注意：JDBC编写SQL语句不需要以分号结尾
            //String sql = "insert into dept(deptno,dname,loc) values(50,'销售部门','北京')";
            //String sql = "insert into dept(deptno,dname,loc) values(60,'市场部','天津')";
            //String sql = "delete from dept where deptno = 60";
            String sql = "update dept set dname = '市场部',loc = '天津' where deptno = 50";
            //发送SQL语句给数据库管理系统
            //数据库管理系统会将此SQL语句编译并执行
            int count = stmt.executeUpdate(sql);
            System.out.println(count);

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

    }
}
```

2.5第五步：处理查询结果集

2.5.1 通过字段下标获取数据

（1） 例如：**rs.getString(下标)**

```java
/*
    JDBC编程六步曲
        1、注册驱动
        2、获取连接
        3、获取数据库操作对象
        4、执行SQL语句
        5、处理查询结果集
        6、释放资源

        JDBC编程第五部：若以上执行SQL语句为DQL语句，我们就要处理查询结果集
        如何处理结果集：
        目前将结果集中的数据打印遍历输出
*/

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

public class JDBCTest05
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2、获取连接
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");

            //3、获取数据库操作对象
            Statement stmt = conn.createStatement();

            //4、执行SQL语句
            //String sql = "select empno,ename,sal from emp";
            String sql = "select empno a,ename b,sal c from emp";
            //发送DQL语句给数据库系统，将查询结果放到ResultSet结果集对象中
            ResultSet rs = stmt.executeQuery(sql);
            /*
                结果集中封装的了什么样的数据？
                +-------+--------+---------+
                | empno | ename  | sal     |
                +-------+--------+---------+
                |  7369 | SMITH  |  800.00 |
                |  7499 | ALLEN  | 1600.00 |
                |  7521 | WARD   | 1250.00 |
                |  7566 | JONES  | 2975.00 |
                |  7654 | MARTIN | 1250.00 |
                |  7698 | BLAKE  | 2850.00 |
                |  7782 | CLARK  | 2450.00 |
                |  7788 | SCOTT  | 3000.00 |
                |  7839 | KING   | 5000.00 |
                |  7844 | TURNER | 1500.00 |
                |  7876 | ADAMS  | 1100.00 |
                |  7900 | JAMES  |  950.00 |
                |  7902 | FORD   | 3000.00 |
                |  7934 | MILLER | 1300.00 |
                +-------+--------+---------+

            */
            //5、处理结果集
            /*
                rs.next()方法的作用
                将光标向前移动一行，
                若指向当前行有记录，返回true
                若指向当前行无记录，返回false
            */
            while(rs.next()){
                //取当前光标指向行中的数据
                //无法数据表中字段是什么类型，都是字符串形式取出


                //以下是根据字段下标获取
                String empno = rs.getString(1);//JDBC所有下标都是以1开始，返回员工编号
                String ename = rs.getString(2);
                String sal = rs.getString(3);

                System.out.println(empno + ":" + ename + ":" + sal);

            }

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


    }
}
```

2.5.2 通过结果集中字段名称获取数据

（1） 例如：rs.getString(“ename”)

（2） 该方式程序健壮，建议使用

```java
/*
    JDBC编程六步曲
        1、注册驱动
        2、获取连接
        3、获取数据库操作对象
        4、执行SQL语句
        5、处理查询结果集
        6、释放资源

        JDBC编程第五部：若以上执行SQL语句为DQL语句，我们就要处理查询结果集
        如何处理结果集：
        目前将结果集中的数据打印遍历输出
*/

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

public class JDBCTest05
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2、获取连接
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");

            //3、获取数据库操作对象
            Statement stmt = conn.createStatement();

            //4、执行SQL语句
            //String sql = "select empno,ename,sal from emp";
            String sql = "select empno a,ename b,sal c from emp";
            //发送DQL语句给数据库系统，将查询结果放到ResultSet结果集对象中
            ResultSet rs = stmt.executeQuery(sql);
            /*
                结果集中封装的了什么样的数据？
                +-------+--------+---------+
                | empno | ename  | sal     |
                +-------+--------+---------+
                |  7369 | SMITH  |  800.00 |
                |  7499 | ALLEN  | 1600.00 |
                |  7521 | WARD   | 1250.00 |
                |  7566 | JONES  | 2975.00 |
                |  7654 | MARTIN | 1250.00 |
                |  7698 | BLAKE  | 2850.00 |
                |  7782 | CLARK  | 2450.00 |
                |  7788 | SCOTT  | 3000.00 |
                |  7839 | KING   | 5000.00 |
                |  7844 | TURNER | 1500.00 |
                |  7876 | ADAMS  | 1100.00 |
                |  7900 | JAMES  |  950.00 |
                |  7902 | FORD   | 3000.00 |
                |  7934 | MILLER | 1300.00 |
                +-------+--------+---------+

            */
            //5、处理结果集
            /*
                rs.next()方法的作用
                将光标向前移动一行，
                若指向当前行有记录，返回true
                若指向当前行无记录，返回false
            */
            while(rs.next()){
                //取当前光标指向行中的数据
                //无法数据表中字段是什么类型，都是字符串形式取出

                //以下根据字段名称获取字段值【该方式程序健壮，建议使用】
                String empno = rs.getString("empno");
                String ename = rs.getString("ename");
                String sal = rs.getString("sal");

                System.out.println(empno + ":" + ename + ":" + sal);


                /*
                //字符串参数为查询结果集中字段名称
                String empno = rs.getString("a");
                String ename = rs.getString("b");
                String sal = rs.getString("c");

                System.out.println(empno + "--" + ename + "--" + sal);
                */

            }

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


    }
}
```

2.5.3 通过特定类型获取数据

（1） 例如：rs.getInt(“empno”);

```java
/*
    JDBC编程六步曲
        1、注册驱动
        2、获取连接
        3、获取数据库操作对象
        4、执行SQL语句
        5、处理查询结果集
        6、释放资源

        JDBC编程第五部：若以上执行SQL语句为DQL语句，我们就要处理查询结果集
        如何处理结果集：
        目前将结果集中的数据打印遍历输出
*/

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

public class JDBCTest05
{
    public static void main(String[] args){

        try{

            //1、注册驱动
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2、获取连接
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");

            //3、获取数据库操作对象
            Statement stmt = conn.createStatement();

            //4、执行SQL语句
            //String sql = "select empno,ename,sal from emp";
            String sql = "select empno a,ename b,sal c from emp";
            //发送DQL语句给数据库系统，将查询结果放到ResultSet结果集对象中
            ResultSet rs = stmt.executeQuery(sql);
            /*
                结果集中封装的了什么样的数据？
                +-------+--------+---------+
                | empno | ename  | sal     |
                +-------+--------+---------+
                |  7369 | SMITH  |  800.00 |
                |  7499 | ALLEN  | 1600.00 |
                |  7521 | WARD   | 1250.00 |
                |  7566 | JONES  | 2975.00 |
                |  7654 | MARTIN | 1250.00 |
                |  7698 | BLAKE  | 2850.00 |
                |  7782 | CLARK  | 2450.00 |
                |  7788 | SCOTT  | 3000.00 |
                |  7839 | KING   | 5000.00 |
                |  7844 | TURNER | 1500.00 |
                |  7876 | ADAMS  | 1100.00 |
                |  7900 | JAMES  |  950.00 |
                |  7902 | FORD   | 3000.00 |
                |  7934 | MILLER | 1300.00 |
                +-------+--------+---------+

            */
            //5、处理结果集
            /*
                rs.next()方法的作用
                将光标向前移动一行，
                若指向当前行有记录，返回true
                若指向当前行无记录，返回false
            */
            while(rs.next()){
                //取当前光标指向行中的数据
                //无法数据表中字段是什么类型，都是字符串形式取出

                //以特定数据类型取出数据
                //empno是int类型
                //ename是varchar类型
                //sal是double类型
                int empno = rs.getInt("a");
                String ename = rs.getString("b");
                Double sal = rs.getDouble("c");

                System.out.println((empno+1) + "-->" + ename + "-->" + sal);
            }

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


    }
}
```

2.6第六步：释放资源

（1） 为了保障能够释放资源，将释放代码编写到 finally 语句中

（2） 需要关闭 ResultSet、Statement、Connection

（3） 一个 Connection 可以创建多个 Statement,一个 Statement 可以得出多个 ResultSet，所以先关闭 ResultSet 再关闭 Statement 最后关闭 Connection

（4） 分别进行 try catch 关闭资源

```
}finally{
    //6、释放资源,分别进行try catch
    try{
        if(rs != null){
            rs.close();
        }
    }catch(SQLException e){
        e.printStackTrace();
    }

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

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

2.7 MyEclipse 开发 JDBC 第一个例子 JDBCTest01.java

2.7.1 查询员工姓名及所对应的部门名称

```
package com.bjpowernode.jdbc;

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

public class JDBCTest01 {

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try{
            //1、注册驱动
            Driver driver = new com.mysql.jdbc.Driver();
            DriverManager.registerDriver(driver);
            //2、获取连接
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode", "root", "123");
            //3、获取数据库操作对象
            stmt = conn.createStatement();
            //4、执行SQL语句
            String sql = "select e.ename,d.dname from emp e join dept d on e.deptno = d.deptno";
            rs = stmt.executeQuery(sql);
            //5、处理查询结果集
            while(rs.next()){
                String ename = rs.getString("ename");
                String dname = rs.getString("dname");

                System.out.println(ename + "---"+ dname);

            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            //6、释放资源
            if(rs != null){
                try {
                    rs.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(stmt != null){
                try {
                    stmt.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }
    }

}
```

2.8注册驱动的另一种方式 JDBCTest02.java

2.8.1 第一种：先获取驱动对象，然后完成驱动注册

```
//1、注册驱动
//第一种方式
Driver driver = new com.mysql.jdbc.Driver();
DriverManager.registerDriver(driver);
```

（1） 包含以下两个语句：

A、 Driver driver = new com.mysql.jdbc.Driver();

B、 DriverManager.registerDriver(driver);

2.8.2 第二种：通过自定义一个 RegisterDriver 类，通过

Class.forName("com.bjpowernode.jdbc.RegisterDriver")

加载 RegisterDriver 类中静态代码块

（1） 静态代码块，包含以下语句：

A、 Driver driver = new com.mysql.jdbc.Driver();

B、 DriverManager.registerDriver(driver);

（2） 优点：**Class.forName("")**双引号中内容可以写到配置文件里

以上写法比较繁琐，但是数据库厂商已实现

2.8.3 数据库厂商已写 com.jdbc.mysql.Driver

在 **com.jdbc.mysql.Driver** 这个类里有一个 **static** 语句块

创建 **Driver** 对象时自动加载注册驱动语句

2.9连接数据库的信息可配置

2.9.1 prj-io 举例：IO 流读取配置文件[在 myeclipse 环境下，默认当前路径为项目的根]

```
package com.bjpowernode.io;

import java.io.FileReader;
import java.util.Properties;


public class IOTest01 {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception{
        //读取db.properties配置文件
        //1.绝对路径
//      FileReader reader = new FileReader("E:\\course\\02-JDBC\\prj-io\\db.properties");

        //1.2相对路径：只要在myeclipse环境下，默认当前路径为项目的根
        FileReader reader = new FileReader("db.properties");

        //1.3 db.properties文件在src文件中时
//      FileReader reader = new FileReader("src/db.properties");
        //1.4 db.properties文件在com.bjpowernode.io包里面、
//      FileReader reader = new FileReader("src/com/bjpowernode/io/db.properties");

        //创建属性对象
        Properties pro = new Properties();

        //通过属性对象的load方法将配置文件中的数据加载到内存中
        pro.load(reader);

        //关闭流
        reader.close();

        //通过属性对象的getProperty()方法获取属性内的属性值
        String driver = pro.getProperty("driver");
        String url = pro.getProperty("url");
        String user = pro.getProperty("user");
        String password = pro.getProperty("password");

        //注意：属性配置文件中key一旦确定，不可修改；配置文件中只可修改value值

        System.out.println(driver+"\n"+url+"\n"+user+"\n"+password);

    }

}
```

（1） 创建 db.properties 配置文件

（2） 通过 FileReader reader = new FileReader(“db.properties”);读取配置文件

（3） 创建属性对象 Properties pro = new Properties();

（4） 通过属性对象 pro.load(reader)方法将配置信息读取到 Map 集合对象中

（5） 关闭流 reader.close();

（6） 通过 pro.getProperty(“key”)获取 value

2.9.2 JDBCTest03.java：连接数据库信息编写到配置文件 db.properties 中

通过以上方式读取配置文件中的 key：value 值

```java
package com.bjpowernode.jdbc;

import java.io.FileReader;


public class JDBCTest03 {

    public static void main(String[] args) throws Exception{

        //读取属性配置文件，获取配置信息
        //1、读取db.properties文件
        FileReader reader = new FileReader("db.properties");
        //创建属性对象
        Properties pro = new Properties();
        //通过属性对象pro的load方法将配置文件中的数据读取到Map集合中
        pro.load(reader);
        //关闭流
        reader.close();

        //通过pro.getProperty(key)获取key的value值
        String driver = pro.getProperty("driver");
        String url = pro.getProperty("url");
        String user = pro.getProperty("user");
        String password = pro.getProperty("password");

        //-----------------连接数据库----------------------------------

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try{
            //1、注册驱动
            Class.forName(driver);

            //2、获取连接
            conn = DriverManager.getConnection(url, user, password);

            //3、获取数据库操作对象
            stmt = conn.createStatement();

            //4、执行SQL语句
            String sql = "select e.ename,d.dname from emp e join dept d on e.deptno = d.deptno";
            rs = stmt.executeQuery(sql);

            //5、处理查询结果集
            while(rs.next()){
                String ename = rs.getString("ename");
                String dname = rs.getString("dname");

                System.out.println(ename +"---"+dname);
            }

        }catch(Exception e){
            e.printStackTrace();
        }finally{
            //6、释放资源
            if(rs != null){
                try {
                    rs.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(stmt != null){
                try {
                    stmt.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }
    }

}
```

（1） 注册驱动:Class.forName(driver);

（2） 获取连接 DriverManager.getConnection(url,user,pwd);

（3） 获取数据操作对象 stmt = conn.createStatement();

（4） 执行 SQL 语句 stmt.executeUpdate(sql)

或 stmt.executeQuery(sql)

（5） 处理查询结果集

A、 rs.next()方法：将光标从当前位置向前移一行

（6） 关闭资源

A、 立即释放此 ResultSet、Statement、Connection 对象的数据库和 JDBC 资源，而不是等待该对象自动关闭时发生此操作

第3章 Statement 和 PreparedStatement

3.1模拟实现用户注册 BufferedReader，JDBCTest04.java

3.1.1 用户输入信息

（1） 接收用户输入：BufferedReader bf =

```
new BufferedReader(new InputStreamReader(System.in));
System.out.println("请输入用户名密码完成信息注册");
System.out.println("请输入用户名：");
String username = br.readLine();
    System.out.println("请输入密码：");
    String password = br.readLine();
    br.close();
    register(username,password);//调用
```

3.1.2 用户注册方法:public static void register(String username, String password)：

（1） 完成数据库连接并且将用户输入插入到数据库

```java
package com.bjpowernode.jdbc;

import java.io.BufferedReader;

public class JDBCTest04_Register {

    /**
     * 使用DOS窗口实现用户注册
     * 1、接收用户的输入：用户名和密码
     * 2、连接数据库，执行insert 语句，完成信息保存
     */
    public static void main(String[] args) throws Exception{
        //接收用户输入
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("请输入用户名和密码完成信息注册");
        System.out.println("请输入用户名：");
        String username = br.readLine();
        System.out.println("请输入密码：");
        String password = br.readLine();
        br.close();
//      System.out.println(username + ":"+password);

        //注册用户信息
        register(username,password);

    }

    /**
     * 用户注册信息
     * @param username  用户名
     * @param password  密码
     */
    public static void register(String username, String password) {
        Connection conn = null;
        Statement stmt = null;
        int count = 0;
        try {
            //注册驱动
            Class.forName("com.mysql.jdbc.Driver");
            //获取连接
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode", "root", "123");
            //获取数据库操作对象
            stmt = conn.createStatement();
            //执行SQL语句
            String sql = "insert into tb1_user(username,userpwd) values('"+username+"','"+password+"')";
            count = stmt.executeUpdate(sql);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally{
            if(stmt != null){
                try {
                    stmt.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

            if(count == 1){
                System.out.println("欢迎"+username+"注册成功");
            }
        }
    }

}
```

3.2模拟实现用户登录，JDBCTest05.java

3.2.1 用户输入信息：BufferedReader bf =

new BufferedRader(new InputStreamReader(System.in));

```
System.out.println("欢迎来到动力节点，输入用户名和密码登录!");
System.out.println("请输入用户名：");
    String username = br.readLine();
    System.out.println("请输入密码：");
    String userpwd = br.readLine();
    br.close();
login(username,userpwd);//调用登录方法
```

3.2.2 用户登录方法：public static void login(String username,String userpwd);

（1） 完成数据库连接并验证用户名和密码

```java
package com.bjpowernode.jdbc;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JDBCTest05_Login {

    /**
     * 使用DOS窗口实现用户登录
     * 1、接收用户名和密码
     * 2、连接数据库，验证用户名和密码是否正确
     */
    public static void main(String[] args) throws Exception{
        //从DOS窗口接收用户名和密码
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("欢迎来到动力节点，输入用户名和密码登录!");
        System.out.println("请输入用户名：");
        String username = br.readLine();
        System.out.println("请输入密码：");
        String userpwd = br.readLine();
        br.close();

        //调用登录方法完成登录
        login(username,userpwd);

    }

    /**
     * 用户登录
     * @param username  用户名称
     * @param userpwd   用户密码
     */
    public static void login(String username, String userpwd) {
                Connection conn = null;
                Statement stmt = null;
                ResultSet rs = null;
                boolean loginSuccess = false;
                try{

                    //注册驱动
                    Class.forName("com.mysql.jdbc.Driver");
                    //获取连接
                    conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");
                    //获取数据库操作对象
                    stmt = conn.createStatement();
                    //执行SQL语句
                    String sql = "select * from tb1_user where username = '"+username +"' and userpwd = '" + userpwd+"'";
                    rs = stmt.executeQuery(sql);
                    if(rs.next()){
                        loginSuccess = true;
                    }
                }catch(Exception e){
                    e.printStackTrace();
                }finally{
                    //释放资源
                    if(rs != null){
                        try{
                            rs.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }
                    if(stmt != null){
                        try{
                            stmt.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }
                    if(conn != null){
                        try{
                            conn.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }

                    System.out.println(loginSuccess ?"登录成功! " : "登录失败! ");
                }

    }
}
```

3.3 SQL 注入

（1） 用户输入的信息中有 SQL 关键字并且参与的 SQL 语句的编译，导致 SQL 语句含义扭曲，这种现象被为 SQL 注入

```
select * from tb1_user where username = 'admin' and userpwd = 'asfkkd' or 'as'='as'
```

（2） 示例：

A、 用户在密码处输入：`asd' or 'as'='as`

B、分析以上原因：导致以上 SQL 语句条件永远为真

因为：该条件被分为两部分，如下：(usename ='admin' and userpwd = 'asfkkd') or 'as'='as'

3.4防止 SQL 注入：PreparedStatement，JDBCTest06.java

3.4.1 使用：ps = conn.prepareStatement(sql);

（1） 先定义 SQL 语句构架，对 SQL 语句进行预先编译，只编译一次

（2） 然后再接收用户提供的信息，即使用户提供的信息中包含 SQL 关键字，这些关键字也不参与编译，是不起作用的。

```java
package com.bjpowernode.jdbc;

import java.io.BufferedReader;

/**
 * 当前这个程序避免SQL注入
 * 如何避免：
 *       先定义SQL语句构架，对SQL语句进行预先编译，只编译一次，然后再接收用户提供的信息
 *       即使用户提供的信息中包含关键字，这些关键字也不参与编译，是不起作用的
 */
public class JDBCTest06_prepare {

    public static void main(String[] args) throws Exception{
    //从DOS窗口接收用户名和密码
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("欢迎来到动力节点，输入用户名和密码登录!");
            System.out.println("请输入用户名：");
            String username = br.readLine();
            System.out.println("请输入密码：");
            String userpwd = br.readLine();
            br.close();

            //调用登录方法完成登录
            login(username,userpwd);

}

    /**
     * 用户登录
     * @param username  用户名称
     * @param userpwd   用户密码
     */
    public static void login(String username, String userpwd) {
                Connection conn = null;
                PreparedStatement ps = null;
                ResultSet rs = null;
                boolean loginSuccess = false;
                try{

                    //1.注册驱动
                    Class.forName("com.mysql.jdbc.Driver");
                    //2.获取连接
                    conn = DriverManager.getConnection("jdbc:mysql://localhost:3366/bjpowernode","root","123");
                    //3.获取数据库操作对象
                    String sql = "select * from tb1_user where username = ? and userpwd = ?";//先定义SQL语句框架
                    ps = conn.prepareStatement(sql);//执行到此，先将SQL语句框架进行编译
                    //给？赋值
                    ps.setString(1, username);
                    ps.setString(2, userpwd);

                    //4.执行SQL语句
                    rs = ps.executeQuery();

                    //5.处理查询结果集
                    if(rs.next()){
                        loginSuccess = true;
                    }
                }catch(Exception e){
                    e.printStackTrace();
                }finally{
                    //6.释放资源
                    if(rs != null){
                        try{
                            rs.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }
                    if(ps != null){
                        try{
                            ps.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }
                    if(conn != null){
                        try{
                            conn.close();
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                    }

                    System.out.println(loginSuccess ?"登录成功！" : "登录失败！");
                }

    }
}
```

3.5 Statement 和 PreparedStatement 对比

（1） 防止 SQL 注入，执行效率高

（2） SQL 语句对于 Statement 来说是：编译一次执行一次

（3） SQL 语句对于 PreparedStatement 是编译一次执行 N 次

**原因：DBMS 厂商实现 JDBC 接口，DBMS 将编译后的 SQL 语句保存在 DBMS 中，由于 DBMS 中有很多编译好的 SQL 语句，这时通过同一个 PreparedStatement 对象进行赋值，便会找到其对应的 PreparedStatement 对象实现其赋值，即：一次编译多次执行

（4） PreparedStatement 是类型安全的，编译期检查传入参数类型

3.6 PreparedStatement 完成模糊查询,JDBCTest07.java

（1） 正确写法：

A、 String sql = "select ename from emp where ename like ?";

B、 ps = conn.prepareStatement(sql);

C、 ps.setString(1,”%K%”);

```
//3.获取数据库操作对象
String sql = "select ename from emp where ename like ?";//先定义SQL语句框架
ps = conn.prepareStatement(sql);
ps.setString(1, "%O%");
```

（2） 错误写法：

```
//3.获取数据库操作对象
String sql = "select ename from emp where ename like '%?%';//先定义SQL语句框架
ps = conn.prepareStatement(sql);
ps.setString(1, "O");
```

错误写法

3.7 PreparedStatement 完成 DML 操作，JDBCTest08.java

（1） 增删改查又称为：CRUD 操作

A、 C：create 增加

B、 R：retrive 检索

C、 U：update 更新

D、 D：delete 删除

（2） 增加操作

A、 String sql = “insert into dept(deptno,dname,loc) values(?,?,?)”;

B、 ps = conn.prepareStatement(sql);

C、 ps.setInt(1,60);

D、 ps.setString(2,”销售部”);

E、 ps.setString(3,”北京”);

F、 int count = ps.executeUpdate();

```
//3.1 增加数据
String sql = "insert into dept(deptno,dname,loc) values(?,?,?)";
//预编译
ps = conn.prepareStatement(sql);
//赋值
ps.setInt(1, 60);
ps.setString(2,"销售部");
ps.setString(3, "北京");
```

（3） 修改操作

A、 String sql = “update dept set dname = ?,loc = ? where deptno = ?”;

B、 ps = conn.prepareStatement(sql);

C、 ps.setString(1,”人事部”);

D、 ps.setString(2,”南京”);

E、 ps.setInt(3,60);

F、 Int count = ps.executeUpdate();

```
//3.2 修改数据
String sql = "update dept set dname=?,loc=? where deptno=?";
//预编译
ps = conn.prepareStatement(sql);
//赋值
ps.setString(1, "人事部");
ps.setString(2, "南京");
ps.setInt(3, 60);
```

（4） 删除操作

A、 String sql = “delete from dept where deptno =?”;

B、 ps = conn.prepareStatement(sql);

C、 ps.setInt(1,60);

D、 Int count = ps.executeUpdate();

```
//3.3 删除60部门
String sql = "delete from dept where deptno = ?";
//预编译
ps = conn.prepareStatement(sql);
//赋值
ps.setInt(1, 60);
```

第4章 JDBC 事务

4.1 JDBC 事务

（1） JDBC 默认情况下，事务是自动提交的：即在 JDBC 中执行一条 DML 语句就执行一条

（2） 将事务的自动提交修改为手动提交即可避免自动提交

（3） 在事务执行的过程中任何一步出现异常都要进行回滚

（4） JDBC 事务只有三行代码：JDBCTest09.java

A、 设置自动提交事务：conn.setAutoCommit(false);

B、 事务提交：conn.commit();

C、 事务回滚：conn.rollback();

（5） 作业：使用 JDBC 完成银行账户转账，要求使用事务，使用 DOS 窗口模拟

4.2 JDBC 行级锁又称为悲观锁

4.2.1 如：多线程访问同一张表中相同记录的时候，其它线程排队

（1） 解决方案有 3 种（java、数据库、SQL 语句）

A、 java：使用 synchronized

B、 数据库：设置隔离级别为：串行化 serializable

C、 SQL 语句：使用行级锁或悲观锁机制 for update

（2） 例子：

A、演示步骤：

a、 第一步：Debug 模式运行 JDBCTest10.java

b、 第二步：正常运行 JavaTest11.java

c、 第三步：看控件台显示内容

B、 JDBCTest10.java 查询表 emp_bak 中 job 为 MANAGER 的员工

a、 在获取数据库连接后面开启事务

b、 在 catch 语句块中进行事务回滚

```
//1.注册驱动
Class.forName("com.mysql.jdbc.Driver");
//2.获取连接
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","111");
//开启事务
conn.setAutoCommit(false);
//3.获取数据库操作对象
String sql = "select ename,job from emp_bak where job=? for update";//先定义SQL语句框架
ps = conn.prepareStatement(sql);
ps.setString(1, "MANAGER");
//4.执行SQL语句
rs = ps.executeQuery();
//5.处理结果集
while(rs.next()){
    String ename = rs.getString("ename");
    String job = rs.getString("job");
    System.out.println("ename:"+ename+"---- job:"+job);
}
conn.commit();
```

C、 JDBCTest11.java 更新 emp_bak 中 job 为 MANAGER 的 sal 为 3000

a、 在获取数据库连接后面开启事务

b、 在 catch 语句块中进行事务回滚

```
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");
        //2.获取连接
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","111");
        //开启事务
        conn.setAutoCommit(false);
        //3.获取预编译数据库操作对象
        String sql = "update emp_bak set sal = ? where job=?";
        ps = conn.prepareStatement(sql);
        ps.setDouble(1, 3000);
        ps.setString(2, "MANAGER");
        //4.执行SQL语句
         int count = ps.executeUpdate();
         System.out.println(count);
         //提交事务
         conn.commit();

    }catch(Exception e){
        //5.出现异常，事务进行回滚
        if(conn != null){
            try {
                conn.rollback();
            } catch (SQLException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }finally{
```

出现异常，事务回滚