JDBC学习

概述

1. 什么是JDBC

- *Java DataBase Connectivity*
- 在Java语言中编写sql语句，对mysql数据库中的数据进行*CRUD*操作
- 相关类库在：*java.sql.**

2. JDBC的本质

- SUN公司Java程序员写好接口
- 数据库（MySQL、Oracle、DB2、Sybase、...）公司的程序员实现接口

  > 实现类都放在jar包中，这个jar包有个专业术语——**驱动**

- Java程序员**面向接口编程**（解耦合）

> JDBC就是本质上一堆接口

3. JDBC开发前的准备

- 将*jar*包配置到*classpath*中

> 这样类加载器才能找到并加载这些*class*文件

> 当然，你自己写的类也要被加载，所以别忘了在*classpath*中加入当前所在目录

> 但在IDEA工具中就不用配置了（IDEA中有自己的配置）

---

4. JDBC编程六步

1. 注册驱动

> 通知Java程序我们将连接哪个品牌的数据库

2. 获取数据库连接

> 开始Java进程与MySQL进程之间的通道

3. 获取数据库操作对象

> 这个对象是用来执行SQL语句的

4. 执行SQL语句

> CRUD

5. 处理查询结果集

> 如果第四步使用的是select语句，才会有第五步

6. 释放资源

> JDBC是进程之间的通信，占用较多资源的

注册驱动

---

1. 相关类

- *java.sql.DriverManager*
- *java.sql.Driver*
- *java.sql.SQLException*
- *com.mysql.jdbc.Driver*（*jar*包中的）

> 建议将java.sql中的类导入，而写com.mysql.jdbc中类的全名

---

2. 代码示例

```
import java.sql.DriverManager;
import java.sql.Driver;
import java.sql.SQLException;
/*...*/
try {
    Driver driver = new
com.mysql.jdbc.Driver();

 DriverManager.registerDriver(driver);
} catch(SQLException e) {
    e.printStackTrace();
}
/*...*/
```

3. 与Oracle对比

- Oracle中相关类的类名是 *oracle.jdbc.driver.OracleDriver*

4. 注册驱动的第二种方式

> Class.forName()方法导致类加载
>
> 而静态代码块中恰好有注册驱动的代码
>
> 这样可以写入properties

```
try {

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

 //Class.forName("oracle.jdbc.driver.OracleDriver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

获取数据库连接

1. 相关类

- *java.sql.Connection*

2. 代码示例

> 以下代码都要在try...catch中

```
String url =
"jdbc:mysql://localhost:3306/bjpowernode"
;
String user = "root";
String password = "123456";
//这是个多态
Connection conn =
DriverManager.getConnection(url, user,
password);
//这个Connection之后是需要关闭的
```

3. URL概述

- *What*
    - 统一资源定位符
- 组成
    - 协议 + IP地址 + 端口号 + 资源名
- 相关概念
    - 协议

      > 提前规定好的数据传输格式

      > 通俗解释：人与人说话时就会用到“中国普通话协议”

    - IP地址

      > 在网络中定位到某台计算机

    - 端口号

      > 定位计算机上某个服务

    - 资源名

      > 服务下的某个资源

- 举例

URL分解	解释
jdbc:mysql://	java程序与MySQL通信的协议
localhost:	本机IP地址
3306/	MySQL端口号
bjpowernode	数据库名

4. 与Oracle对比

- URL：*oracle:jdbc:thin:@localhost:1521:bjpowernode*

获取数据库操作对象

1. 相关类

- *java.sql.Statement*
- *java.sql.PreparedStatement*

---

2. 代码示例

```
Statement stmt = conn.createStatement();
```

> 一个Connection对象可以创建多个Statement对象

---

3. SQL注入（1）

1. *What*
    - 借助输入拼接，扭曲了原SQL语句的含义
2. *Why*
    - 输入中含有**SQL语句关键字**， 参与了SQL语句的**编译**
    - 是**先拼串后编译**的
3. *How*
    - 用*PreparedStatement*替代*Statement*

类	特点	优点	缺点
Statement	先拼接再编译SQL语句	可以拼接	导致SQ注入
PreparedStatement	先编译再给SQL语句传值	避免SQL注入	不能拼接

- 将第三步改为“获取预编译的数据库操作对象”
    - 先写SQL语句

> 代码示例：使用问号做占位符（**问号两边别加单引号**）

```
String sql = "select * from t_user where login_name = ? and login_pwd = ?";
```

- 获取预编译的数据库操作对象（**编译**）

> 代码示例：改用prepareStatement方法

```
stmt = conn.prepareStatement(sql);
//这里就会发送SQL语句给DBMS进行编译
```

- 给占位符传值

> 代码示例：是什么类型就"set"什么

```
stmt.setString(1, loginName);
stmt.setString(2, loginPwd);
//下标从1开始，1代表第1个问号
//setString会把loginName整体直接加单引号
```

- 见第四步“执行SQL语句”中的“3. SQL注入（2）”

---

4. Statement使用场景

- PreparedStatement用得较多，适合**传值**、**不参与编译**的情况
- Statement适合字符串拼接，要拼接的字符串是不得不**参与编译**的

> 例如：提供选项，让用户只能选择待拼接的字符串，而不是随意输入

---

5. PreparedStatement执行DML

1. 写SQL语句
2. 获取预编译的数据库操作对象

3. 赋值

4. 执行

> 代码示例

```
String sql = "update dept set dname = ?, loc = ? where deptno = ?";
ps = conn.prepareStatement(sql);
ps.setString(1, "软件研发部");
ps.setString(2, "北京");
ps.setInt(3, 50);
int count = ps.executeUpdate();
```

---

6. 模糊查询

> 代码示例：
>
> **不要在单引号中用问号**；

```
String sql = "select ename from emp where ename like ?";
ps = conn.prepareStatement(sql);
ps.setString(1, "%o%");
rs = ps.executeQuery();
```

执行SQL语句

1. 相关方法

- *int executeUpdate(String sql)*

  > 执行给定的SQL语句（DML），返回影响到的记录条数

- *ResultSet executeQuery(String sql)*

  > 执行给定的SQL语句(DQL)，返回查询结果集

---

2. 代码示例

```
String insertSql = "insert into dept(deptno, dname, loc) values(50, '销售部', '北京')";//JDBC中SQL语句不需要以分号结尾
int count = stmt.executeUpdate(insertSql);
```

```
String updateSql = "update dept set dname = '人事部', loc = '天津' where deptno = 50";
int count = stmt.executeUpdate(updateSql);
```

```
String deleteSql = "delete from dept where deptno = 50";
int count = stmt.executeUpdate(deleteSql);
```

```
String querySql = "select empno, ename, sal from emp order by sal desc";
ResultSet rs = stmt.executeQuery(querySql);
```

---

3. SQL注入（2）

- 上接“获取数据库操作对象”中的“3. SQL注入（1）”

> 代码示例：不必再传入SQL语句了

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

释放资源

---

1. 相关操作

- 在finally语句块中
- 先关闭ResultSet
- 再关闭Statement
- 最后关闭Connection
- 记得要**分开**处理异常

---

2. 代码示例

```

```

```
/*...*/
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
    /*...*/
} catch(SQLException e) {
    e.printStackTrace();
} finally {
    if(rs != null) {
        try {
            rs.close();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
    if(stmt != null) {
        try {
            stmt.close();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
    if(conn != null) {
        try {
            conn.close();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

处理查询结果集

1. 相关类

- java.sql.ResultSet

2. 代码示例

> 搭架子的代码就不写了

- 执行查询语句以及循环取元素

```
String querySql = "select empno, ename, sal from emp order by sal desc";
rs = stmt.executeQuery(querySql);
while(rs.next()) {
    /*...*/
}
```

- 无论底层元素是什么类型，都以String类型取出

```
String empno = rs.getString(1);//注意下标从1开始
String ename = rs.getString(2);
String sal = rs.getString(3);
```

- 以特定类型取出

```
int empno = rs.getInt(1);
String ename = rs.getString(2);
double sal = rs.getDouble(3);
```

- 使用列名取出元素（常用）

> 列取别名之后要用别名

> 不用加“表名.”

```
int empno = rs.getInt("empno");
String ename =
rs.getString("ename");
double sal = rs.getDouble("sal");
```

其他内容

1.在IDEA中使用JDBC

- 配置lib
    - 在工程目录下创建*lib*文件夹*(Directory)*
    - 将*jar*包放到*lib*中
    - 将*jar*包*"Add as Library..."*

---

2. 使用配置文件

- 用注册驱动第二种方法
- 新建resources包，新建db.properties配置文件

```
### mysql connectivity configuration ###
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost":3306/bjpowernode
user=root
password=123456
```

```
### oracle connectivity configuration ###
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:bjpowernode
user=scott
password=tiger
```

- 结合Bundle

```
/*...*/
ResourceBundle bundle = ResourceBundle.getBundle("resources/db");
String driver = bundle.getString("driver");
String url = bundle.getString("url");
String user = bundle.getString("user");
String password = bundle.getString("password");
/*...*/
Class.forName(driver);
conn = DriverManager.getConnection(url, user, password);
/*...*/
```

3. 模拟登录功能

1. 步骤概述
    1. 提供一个**输入**的界面，可以让用户输入用户名和密码
    2. 底层数据库当中需要有一张**用户表**，用户表中存储了**用户信息**

3. 当java程序接收到用户名和密码的时候，**连接数据库验证**用户名和密码
    - 验证通过，表示登录成功
    - 验证失败，表示登录失败

2. 具体操作
    1. 在工程目录下配置sql脚本
    2. 封装方法
        1. 初始化界面，并接收用户的输入

            > 代码示例

            ```
            private static Map<String, String> initUI() {/*...*/}
            ```

        2. 验证登录名和密码

            > 代码示例

            ```
            private static boolean checkNameAndPwd(String loginName, String loginPwd) {/*...*/}
            ```

---

4. JDBC事务

1. JDBC默认自动提交
    - 在实际开发中，必须将JDBC的自动提交机制关闭
2. 操作方法
    1. 在获取连接后，立刻关闭自动提交机制，并开启事务
    2. 在DML语句全部成功结束后，提交事务

        > 代码示例

        ```
        conn = DriverManager.getConnection("...");
        conn.setAutoCommit(false);
        /*...*/
        conn.commit();
        ```

    3. 保险起见，出现异常后，在catch子句中处理，将事务回滚

        > 代码示例

```
catch (Exception e) {
    try {
        if (conn != null) {
            conn.rollback();
        }
    } catch (SQLException ex) {

     ex.printStackTrace();
    }
    /*...*/
}
```

5. JDBC工具类

1. 概述

> 自定义工具类，将构造方法私有化

```
public class DBUtil{
    // Suppresses default constructor, ensuring non-instantiability.
    private DBUtil(){}
}
```

2. 注册驱动

> 代码示例：在静态代码块中执行

```
static {
    try {

Class.forName(bundle.getString("dri
ver"));
    } catch (ClassNotFoundException
e) {
        e.printStackTrace();
    }
}
```

3. 获取连接

> 代码示例

```
public static Connection
getConnection() throws SQLException
{
    String url =
bundle.getString("url");
    String user =
bundle.getString("user");
    String password =
bundle.getString("password");
    Connection conn =
DriverManager.getConnection("passwo
rd");
    return conn;
}
```

4. 释放资源

> 代码示例

```
public static void
close(Connection conn, Statement
stmt, ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

5. 投入使用

> 代码示例

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;
try {
    conn = DBUtil.getConnection();
    String sql = "select ename, sal from emp where ename like ?";
    ps = conn.prepareStatement(sql);
    ps.setString(1, "A%");
    rs = ps.executeQuery();
    while (rs.next()) {

 System.out.println(rs.getString("ename") + ", " + rs.getDouble("sal"));
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    DBUtil.close(conn, ps, rs);
    //如果没有rs就传入null
}
```

5. 行级锁

> 代码示例

```
select ename , sal from emp where empno = ? for update;
```

> 解释：在本次事务中，任何事务都不能对本事务查询到的记录进行修改，直到本次事务结束