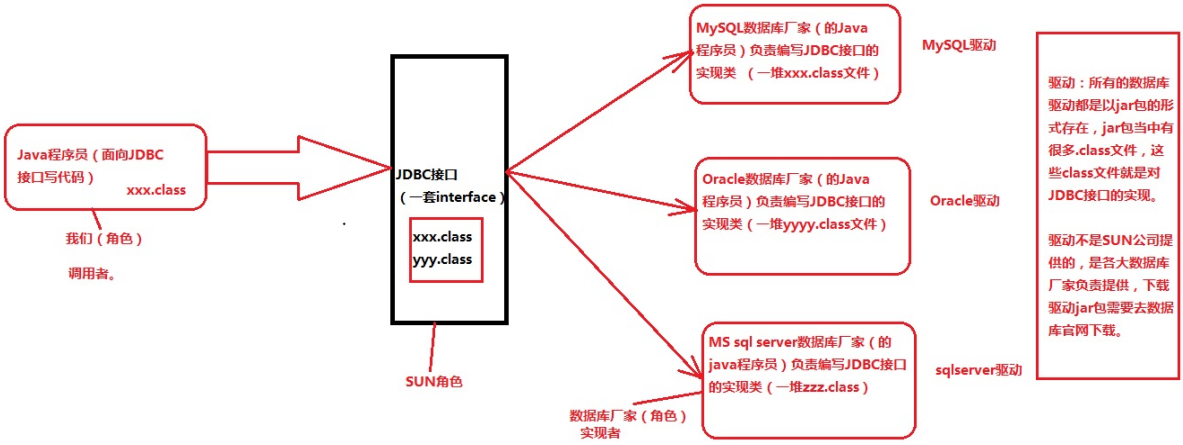
JDBC

一、基本步骤

- JDBC----Java DataBase Connectivity（Java语言连接数据库）
- JDBC的本质是什么？-----JDBC是SUN公司制定的一套接口（interface）
  java.sql.*; (这个软件包下有很多接口。)
  接口都有调用者和实现者。面向接口调用、面向接口写实现类，这都属于面向接口编程。
- 解耦合



- JDBC开发前的准备工作，先从官网下载对应的驱动jar包，然后将其配置到环境变量classpath当中。
  classpath= . ; D:\course\06-JDBC\resources\MySql Connector Java 5.1.23\mysql-connector-java-5.1.23-bin.jar
- 以上的配置是针对于文本编辑器的方式开发，使用IDEA工具的时候，不需要配置以上的环境变量。IDEA有自己的配置方式。
- JDBC编程六步（需要背会）

  第一步：注册驱动（作用：告诉Java程序，即将要连接的是哪个品牌的数据库）

  第二步：获取连接（表示JVM的进程和数据库进程之间的通道打开了，这属于进程之间的通信，重量级的，使用 完之后一定要关闭通道。）

  第三步：获取数据库操作对象（专门执行sql语句的对象）

  第四步：执行SQL语句（DQL DML....）

  第五步：处理查询结果集（只有当第四步执行的是select语句的时候，才有这第五步处理查询结果集。）

  第六步：释放资源（使用完资源之后一定要关闭资源。Java和数据库属于进程间的通信，开启之后一定要关闭。）

- 释放资源，一般写在finally语句块中，先释放查询结果集，再释放数据库操作对象，最后释放连接
- 最好把连接数据库的信息写入配置文件中，通过ResourceBundle读取

```
1
```

```
import java.sql.*;

public class JDBCTest02{
    public static void main(String[] args){
        Connection conn = null;
        Statement stmt = null;
        try{
            //1、注册驱动
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
            //2、获取连接
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","333");
            //3、获取数据库操作对象
            stmt = conn.createStatement();
            //4、执行SQL语句
            String sql = "delete from dept where deptno = 40";
            int count = stmt.executeUpdate(sql);
            System.out.println(count == 1 ? "删除成功" : "删除失败");
        }catch(SQLException e){
            e.printStackTrace();
        }finally{
            e.printStackTrace();
        }finally{
            //6、释放资源
            if(stmt != null){
                try{
                    stmt.close();
                }catch(SQLException e){
                    e.printStackTrace();
                }
            }
            if(conn != null){
                try{
                    conn.close();
                }catch(SQLException e){
                    e.printStackTrace();
                }
            }
        }

    }
}
```

- 注JDBC的sql语句不需要加分号结尾
- 常用的注册驱动方式：

```
/*
    注册驱动的另一种方式（这种方式常用）
*/
import java.sql.*;

public class JDBCTest03{
    public static void main(String[] args){
        try{
            //1、注册驱动
            // 这是注册驱动的第一种写法。
            // DriverManager.registerDriver(new com.mysql.jdbc.Driver());
            // 注册驱动的第二种方式：常用的。
            // 为什么这种方式常用？因为参数是一个字符串，字符串可以写到xxx.properties文件中。
            // 以下方法不需要接收返回值，因为我们只想用它的类加载动作。
            Class.forName("com.mysql.jdbc.Driver");
            //2、获取连接
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","333");
            // com.mysql.jdbc.JDBC4Connection@41cf53f9
            System.out.println(conn);

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

- 遍历结果集

```java
/*
    处理查询结果集（遍历结果集。）
*/
import java.sql.*;

public class JDBCTest05{
    public static void main(String[] args){
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try{
            //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","333");
            //3、获取数据库操作对象
            stmt = conn.createStatement();
            //4、执行sql
            String sql = "select empno as a,ename,sal from emp";
            // int executeUpdate(insert/delete/update)
            // ResultSet executeQuery(select)
            rs = stmt.executeQuery(sql); // 专门执行DQL语句的方法。
            //5、处理查询结果集

            while(rs.next()){
                /*
                String empno = rs.getString(1);
                String ename = rs.getString(2);
                String sal = rs.getString(3);
                System.out.println(empno + "," + ename + "," + sal);
                */

                /*
                // 这个不是以列的下标获取，以列的名字获取
                //String empno = rs.getString("empno");
                String empno = rs.getString("a"); // 重点注意：列名称不是表中的列名称，是查询结果集的列名称。
                String ename = rs.getString("ename");
                String sal = rs.getString("sal");
                System.out.println(empno + "," + ename + "," + sal);
                */

                // 除了可以以String类型取出之外，还可以以特定的类型取出。
                /*
                int empno = rs.getInt(1);
                String ename = rs.getString(2);
                double sal = rs.getDouble(3);
                System.out.println(empno + "," + ename + "," + (sal + 100));
                */

                int empno = rs.getInt("a");
                String ename = rs.getString("ename");
                double sal = rs.getDouble("sal");
                System.out.println(empno + "," + ename + "," + (sal + 200));
            }

        }catch(Exception e){
            e.printStackTrace();
        }finally{
            //6、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();
                }
            }
        }
    }
```

二、sql注入

- 导致SQL注入的根本原因是：
  用户输入的信息中含有sql语句的关键字，并且这些关键字参与sql语句的编译

过程，导致sql语句的愿意被扭曲，进而达到SQL注入

- 解决SQL注入问题：
  ------只要用户提供的信息不参与SQL语句的编译过程，就可解决。采用java.sql.PreparedStatement。PreparedStatement接口继承了java.sql.Statement,属于预编译的数据库操作对象，会预先对SQL语句的框架进行编译，然后再给SQL语句传值。
  -----在接收用户输入的位置先用？占位，编译结束后再将用户输入传递进SQL语句中，这样用户提供的信息就不参与SQL语句的编译过程
- PreparedStatement进行查询操作

```
boolean loginSuccess = false;
// 单独定义变量
String loginName = userLoginInfo.get("loginName");
String loginPwd = userLoginInfo.get("loginPwd");

// JDBC代码
Connection conn = null;
PreparedStatement ps = null; // 这里使用PreparedStatement（预编译的数据库操作对象）
ResultSet rs = null;

try {
    // 1、注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 2、获取连接
    conn = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/bjpowernode", user: "root", password: "333");
    // 3、获取预编译的数据库操作对象
    // SQL语句的框子。其中一个?，表示一个占位符，一个?将来接收一个"值"，注意：占位符不能使用单引号括起来。
    String sql = "select * from t_user where loginName = ? and loginPwd = ?";
    // 程序执行到此处，会发送sql语句框子给DBMS，然后DBMS进行sql语句的预先编译。
    ps = conn.prepareStatement(sql);
    // 给占位符?传值（第1个问号下标是1，第2个问号下标是2，JDBC中所有下标从1开始。）
    ps.setString( parameterIndex: 1, loginName);
    ps.setString( parameterIndex: 2, loginPwd);
    // 4、执行sql
    rs = ps.executeQuery();
    // 5、处理结果集
    if(rs.next()){
        // 登录成功
```

- Statement和PreparedStatement对比：
  ---Statement存在sql注入问题，PreparedStatement解决了sql注入问题
  ---Statement是编译一次执行一次，PreparedStatement是编译一次可以执行N次，效率较高
  ----PreparedStatement会在编译阶段做类型的安全检查
- 什么情况下必须使用Statement?
  ----业务要求必须支持SQL注入，需要进行SQL语句拼接的话必须使用Statement。例如需要用户输入esc或desc进行升序降序操作，使用占位符的话会转换成字符串，不符合sql语法，所以不能用PreparedStatement。
- PreparedStatement进行增删改操作

```
public class JDBCTest09 {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement ps = null;
        try {
            // 1、注册驱动
            Class.forName("com.mysql.jdbc.Driver");
            // 2、获取连接
            conn = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/bjpowernode", user: "root",password: "333");
            // 3、获取预编译的数据库操作对象
            String sql = "insert into dept(deptno,dname,loc) values(?,?,?)";
            ps = conn.prepareStatement(sql);
            ps.setInt( parameterIndex: 1, x: 60);
            ps.setString( parameterIndex: 2, x: "销售部");
            ps.setString( parameterIndex: 3, x: "上海");
            // 4、执行SQL
            int count = ps.executeUpdate();
            System.out.println(count);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 6、释放资源
            if (ps != null) {
                try {
                    ps.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
```

三、事务机制

- JDBC中只要执行任意一条DML语句就提交一次，为保证数据安全，需要禁用自动提交
- conn.setAutoCommit(false); -----获取连接之后，需要禁用自动提交机制
- conn.commit(); ------代码执行结束，需要提交，写在捕捉异常上一行
- conn.rollback();-----写在catch语句块中。出现异常，需要回滚，将数据恢复到最初

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

四、行级锁

- MyISAM和MEMORY存储引擎采用的是表级锁（table-level locking）
- InnoDB存储引擎既支持行级锁（ row-level locking），也支持表级锁，但默认情况下是采用行级锁。
- **表级锁**： 每次操作锁住整张表。开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低；
- **行级锁**： 每次操作锁住一行数据。开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高；
- **悲观锁**是就是悲观思想，即认为写多，遇到并发写的可能性高，每次去拿数据的时候都认为别人会修改，所以每次在读写数据的时候都会上锁，这样别人想读写这个数据就会block直到拿到锁。
  -----事务必须排队，不允许并发
- 悲观锁的不同实现：
  共享锁(lock in share mode)-----共享锁又称读锁,一个线程给数据加上共享锁之后,其他线程只能读数据不能修改数据

  排他锁(for update)-----排他锁又称写锁,跟共享锁的区别是,其他线程既不能读也不能写数据(select后面添加 for update)
- **乐观锁**是一种乐观思想，即认为读多写少，遇到并发写的可能性低，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，采取在写时先读出当前版本号，然后加锁操作(比较跟上一次的版本号，如果一样则更新)，如果失败则要重复读-比较-写的操作（回滚）。
  -----支持并发，事务不需要排队，需要用版本号

```
//JDBCd
```

```java
import java.sql.*;

public class JDBCtest {
    public static void main(String[] args) {
        Connection conn=null;
        PreparedStatement ps=null;
        ResultSet rs=null;
        try {
            //1、注册驱动
            Class.forName("com.mysql.cj.jdbc.Driver");
            //2、建立连接
            conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","123456");
            //关闭自动提交
            conn.setAutoCommit(false);
            //3、创建预编译数据库对象
           String sql="insert into t_userlogin (name,password) values(?,?)";
            //String sql="select *from t_userlogin";
            ps = conn.prepareStatement(sql);
            ps.setString(1,"Jack");
            ps.setString(2,"111");
            //4、执行sql语句
            int count=ps.executeUpdate();
            System.out.println(count==1?"插入成功":"插入失败");
            /*
            若是sql="select *from t_userlogin"语句
            4、执行sql语句
            rs=ps.executeQuery();
            5、处理查询结果集
            while(rs.next()){
                String name=rs.getString("name");
                String psw=rs.getString("password");
                System.out.println(name+" "+psw);
            }
             */
            //提交
            conn.commit();
        } catch (Exception e) {
            if (conn == null) {
                try {
                    //回滚
                    conn.rollback();
                } catch (SQLException ex) {
                    ex.printStackTrace();
                }
            }
            e.printStackTrace();
```

```
48          }finally {
49              //6、释放资源
50              if (rs != null) {
51                  try {
52                      rs.close();
53                  } catch (SQLException e) {
54                      e.printStackTrace();
55                  }
56              }
57              if (ps != null) {
58                  try {
59                      ps.close();
60                  } catch (SQLException e) {
61                      e.printStackTrace();
62                  }
63              }
64              if (conn != null) {
65                  try {
66                      conn.close();
67                  } catch (SQLException e) {
68                      e.printStackTrace();
69                  }
70              }
71          }
72      }
73  }
```