JDBC 详细介绍

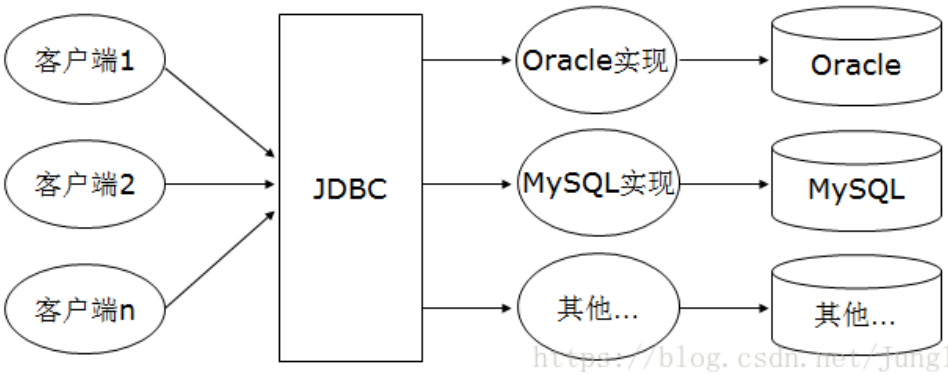
目录

前言

如果你能够仔细阅读完这篇文章，JDBC 的相关知识我想你一定会有所掌握。在阅读的过程中，有任何不理解的地方都欢迎留言讨论。

JDBC 介绍

JDBC（**J**ava **D**ata**B**ase **C**onnectivity）是 Java 和数据库之间的一个桥梁，是一个规范而不是一个实现，能够执行 SQL 语句。它由一组用 Java 语言编写的类和接口组成。各种不同类型的数据库都有相应的实现，**本文中的代码都是针对 MySQL 数据库实现的。**



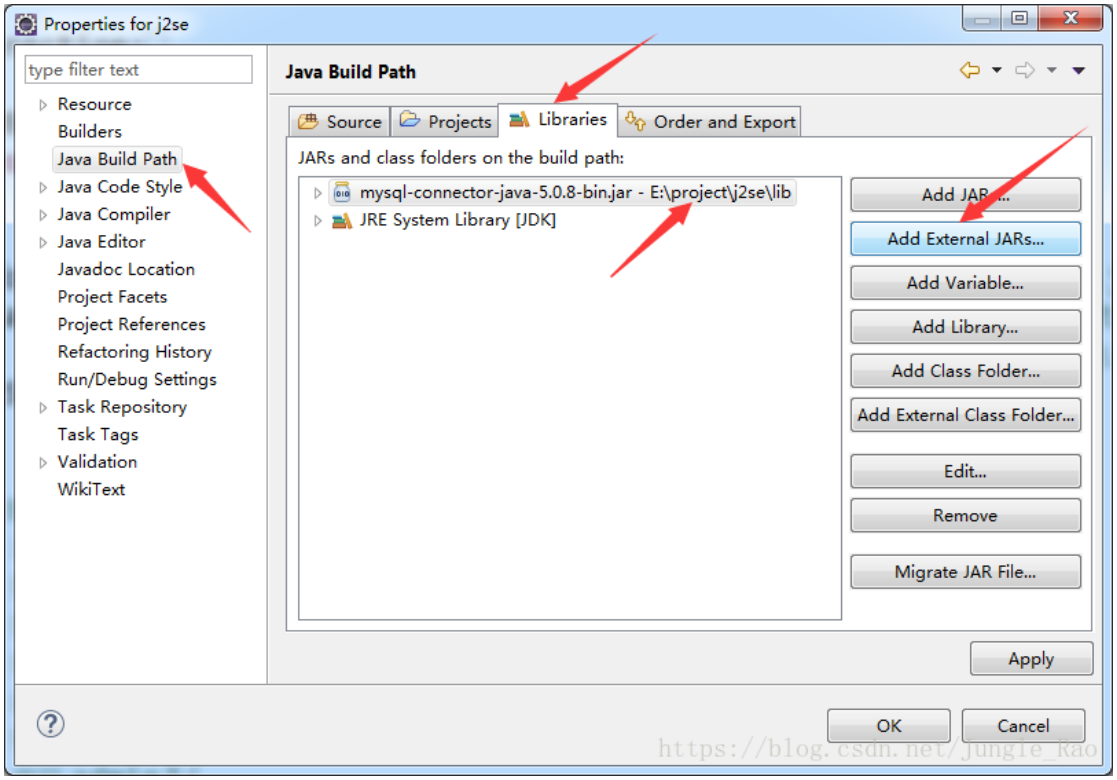
JDBC 编程步骤

1.装载相应数据库的 JDBC 驱动并进行初始化

- **导入专用的 jar 包（不同的数据库需要的 jar 包不同）**

访问 MySQL 数据库需要用到第三方的类，这些第三方的类，都被压缩在一个.Jar 的文件里。mysql-connector-java-5.0.8-bin.jar 包可以在网上下载，或者在 MySQL 的安装目录下找到。通常下载到该 jar 包之后将其放到在项目的 lib 目录下，在本例就会放在 E:\project\j2se\lib 这个位置，然后在 eclipse 中导入这个 jar 包。

导包步骤： 右键 project->property->java build path->libaries->add external jars



如果没有完成上述步骤的导包操作，后面会抛出 ClassNotFoundException

- **初始化驱动**

通过初始化驱动类 com.mysql.jdbc.Driver，该类就在 mysql-connector-java-5.0.8-bin.jar 中。如果你使用的是 oracle 数据库那么该驱动类将不同。

注意：Class.forName 需要捕获 ClassNotFoundException.

```
try {
        Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
```

Class.forName 是把这个类加载到 JVM 中，加载的时候，就会执行其中的静态初始化块，完成驱动的初始化的相关工作。

2.建立 JDBC 和数据库之间的 Connection 连接

这里需要提供：数据库服务端的 IP 地址:127.0.0.1 (这是本机，如果连接其他电脑上的数据库，需填写相应的 IP 地址)

数据库的端口号： 3306 （mysql 专用端口号）

数据库名称 exam（根据你自己数据库中的名称填写）

编码方式 UTF-8

账号 root

密码 admin（如果你在创建数据库的时候没有使用默认的账号和密码，请填写自己设置的账号和密码）

```
Connection c = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/exam?characterEncoding=UTF-8", "root", "admin");
```

Connection 是与特定数据库连接回话的接口，使用的时候需要导包，而且必须在程序结束的时候将其关闭。getConnection 方法也需要捕获 SQLException 异常。

因为在进行数据库的增删改查的时候都需要与数据库建立连接，所以可以在项目中将建立连接写成一个工具方法，用的时候直接调用即可：

```
1.      /**
2.       * 取得数据库的连接
3.       * @return 一个数据库的连接
4.       */
5.  public static Connection getConnection(){
6.              Connection conn = null;
7.              try {
8.                              //初始化驱动类 com.mysql.jdbc.Driver
9.                  Class.forName("com.mysql.jdbc.Driver");
```

```
                conn =
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/exam?characterEncoding=UT
F-8","root", "admin");
                //该类就在 mysql-connector-java-5.0.8-bin.jar 中,如果忘记了第一个步
骤的导包，就会抛出 ClassNotFoundException
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }catch (SQLException e) {

                e.printStackTrace();
            }
            return conn;
        }
```

3.创建 Statement 或者 PreparedStatement 接口，执行 SQL 语句

- 使用 Statement 接口

Statement 接口创建之后，可以执行 SQL 语句，完成对数据库的增删改查。其中 ，增删改只需要改变 SQL 语句的内容就能完成，然而查询略显复杂。在 Statement 中使用字符串拼接的方式，该方式存在句法复杂，容易犯错等缺点，具体在下文中的对比中介绍。所以 Statement 在实际过程中使用的非常的少，所以具体的放到 PreparedStatement 那里给出详细代码。

字符串拼接方式的 SQL 语句是非常繁琐的，中间有很多的单引号和双引号的混用，极易出错。

```
Statement s = conn.createStatement();
// 准备 sql 语句
// 注意： 字符串要用单引号'
String sql = "insert into t_courses values(null,"+"'数学')";
//在 statement 中使用字符串拼接的方式，这种方式存在诸多问题
```

```
s.execute(sql);
System.out.println("执行插入语句成功");
```

- 使用 PreparedStatement 接口

与 Statement 一样，PreparedStatement 也是用来执行 sql 语句的与创建 Statement 不同的是，需要根据 sql 语句创建 PreparedStatement。除此之外，还能够通过设置参数，指定相应的值，而不是 Statement 那样使用字符串拼接。

给数据库中添加课程：（以下代码中最后关闭资源的两个方法 DbUtil.close(pstmt); DbUtil.close(conn); 和上面的建立连接的方法是一样的，是在工具类中定义了的关闭方法，下文会给出其代码）

```
    /**
     * 添加课程
     * @param courseName 课程名称
     */
    public void addCourse(String courseName){
        String sql = "insert into t_course(course_name) values(?)";
//该语句为每个 IN 参数保留一个问号（“？”）作为占位符
        Connection conn = null;                          //和数据库取得连接
        PreparedStatement pstmt = null;          //创建 statement
        try{
            conn = DbUtil.getConnection();
            pstmt = (PreparedStatement) conn.prepareStatement(sql);
            pstmt.setString(1, courseName); //给占位符赋值
            pstmt.executeUpdate();                  //执行
        }catch(SQLException e){
            e.printStackTrace();
        }
        finally{
            DbUtil.close(pstmt);
```

```
                    DbUtil.close(conn);             //必须关闭
            }
        }
```

对数据库中的课程进行删除：

```
    /**
     * 删除课程
     * @param courseId
     */
    public void delCourse(int courseId){
            String sql = "delete from t_course where course_id = ?";
            Connection conn = null;
            PreparedStatement pstmt = null;
            try {
                    conn = DbUtil.getConnection();
                    pstmt = (PreparedStatement) conn.prepareStatement(sql);
                    pstmt.setInt(1, courseId);
                    pstmt.executeUpdate();
            } catch (SQLException e) {
                    // TODO: handle exception
                    e.printStackTrace();
            }finally{
                    DbUtil.close(pstmt);
                    DbUtil.close(conn);             //必须关闭
            }
    }
```

对数据库中的课程进行修改：

```
    /**
     * 修改课程
     * @param courseId
     * @param courseName
     */
    public void modifyCourse(int courseId,String courseName){
            String sql = "update t_course set course_name =? where
course_id=?";
            Connection conn = null;
            PreparedStatement pstmt = null;
            try {
```

```
                    conn = DbUtil.getConnection();
                    pstmt = (PreparedStatement) conn.prepareStatement(sql);
                    pstmt.setString(1, courseName);  //利用 Preparedstatement 的
set 方法给占位符赋值
                    pstmt.setInt(2, courseId);
                    pstmt.executeUpdate();
                } catch (SQLException e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }finally{
                    DbUtil.close(pstmt);
                    DbUtil.close(conn);             //必须关闭
                }
        }
```

由上面的增删改程序可以看出，他们的代码都是大同小异的，主要是 SQL 语句存在差异，其他的地方几乎是完全相同的。其中有几个地方需要注意：

1. 使用 PreparedStatement 时，他的 SQL 语句不再采用字符串拼接的方式，而是采用占位符的方式。“？”在这里就起到占位符的作用。这种方式除了避免了 statement 拼接字符串的繁琐之外，还能够提高性能。每次 SQL 语句都是一样的，java 类就不会再次编译，这样能够显著提高性能。

   ```
   String sql = "update t_course set course_name =? where course_id=?";
   ```

   后面需要用到 PreparedStatement 接口创建的 pstmt 的 set 方法给占位符进行赋值。**注意一点，这里的参数索引是从 1 开始的。**

   ```
   pstmt = (PreparedStatement) conn.prepareStatement(sql);
   pstmt.setString(1, courseName);  //利用 Preparedstatement 的 set 方法给占位符赋值
   pstmt.setInt(2, courseId);
   pstmt.executeUpdate();
   ```

   增删改都使用 pstmt.executeUpdate();语句进行 SQL 语句的提交 ，下文的查询会有所不同，请注意。

2. 在添加的过程的，如果添加的数据量比较大的话，可以用批量添加。 PreparedStatement 接口提供了相应的批量操作的方法。

```
for(int i=1;i<100;i++){
    pstmt.setInt(1,8000+i);
    pstmt.setString(2,"赵_"+i);
    pstmt.addBatch();
//批量更新
    if(i%10==0){
    pstmt.executeBatch();
    }
}
```

下面我们来看稍显麻烦一点的查询操作：

```
        /**
         * 查询课程
         * @return
         */
        public List<Course> findCourseList(){
                String sql = "select * from t_course order by course_id";
                Connection conn = null;
                PreparedStatement pstmt = null;
                ResultSet rs = null;
                //创建一个集合对象用来存放查询到的数据
                List<Course> courseList = new ArrayList<>();
                try {
                        conn = DbUtil.getConnection();
                        pstmt = (PreparedStatement) conn.prepareStatement(sql);
                        rs = (ResultSet) pstmt.executeQuery();
                        while (rs.next()){
                                int courseId = rs.getInt("course_id");
                                String courseName = rs.getString("course_name");
                                //每个记录对应一个对象
                                Course course = new Course();
                                course.setCourseId(courseId);
                                course.setCourseName(courseName);
                                //将对象放到集合中
```

```
                courseList.add(course);
            }
        } catch (SQLException e) {
            // TODO: handle exception
            e.printStackTrace();
        }finally{
            DbUtil.close(pstmt);
            DbUtil.close(conn);             //必须关闭
        }
        return courseList;
    }
```

查询操作使用 executeQuery()进行更新。其他相关的问题放在第四步（处理和显示结果）中解释。

4.处理和显示结果

执行查询语句，并把结果集返回给集合 ResultSet

```
ResultSet rs = s.executeQuery(sql);
```

利用 While(ResultSet.next()){...}循环将集合 ResultSet 中的结果遍历出来。

ResultSet.getXX(); 这里的 get 方法的括号里面可以填属性值，如下图代码中的 course_id,还可以填该属性在数据表中的列号，**从 1 开始编码**，例如：course_id 在我的 t-courses 数据表中位于第一列，所以执行 get 方法的时候，我除了代码段中写法外，还可以这样写 int courseId = rs.getInt(1);但是不推荐使用列号的这种方式，因为一段数据表中个属性值得顺序发生变化，就会导致这里出错，而使用属性名则不会出现这样的问题。

```
while (rs.next()){
    int courseId = rs.getInt("course_id");
    String courseName = rs.getString("course_name");
    //每个记录对应一个对象
    Course course = new Course();
```

```
//在我的项目中创建了 course 类，其中定义了 set 方法，所以这里将查询到的值传给了 course，也可以直接打印到控制台
                course.setCourseId(courseId);
                course.setCourseName(courseName);
                //将对象放到集合中
                courseList.add(course);
            }
```

还有一点需要说明的是：

因为在我的项目中创建了 course 类，其中定义了 set 方法，所以这里将查询到的值传给了 course，你也可以直接用打印语句将 CourseId 和 CourseName 打印到控制台。

```
course.setCourseId(courseId);
course.setCourseName(courseName);
```

5.释放资源

在 JDBC 编码的过程中我们创建了 Connection、ResultSet 等资源，这些资源在使用完毕之后是一定要进行关闭的。关闭的过程中遵循从里到外的原则。因为在增删改查的操作中都要用到这样的关闭操作，为了使代码简单，增加其复用性，这里我将这些关闭的操作写成一个方法和建立连接的方法一起放到一份工具类中。

```
/**
     * 封装三个关闭方法
     * @param pstmt
     */
    public static void close(PreparedStatement pstmt){
            if(pstmt != null){                                    //避免出现空指针异常
```

```
                try{
                    pstmt.close();
                }catch(SQLException e){
                    e.printStackTrace();
                }
            }
        }
        public static void close(Connection conn){
            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }
            }
        }
        public static void close(ResultSet rs){
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }
            }
        }
```

JDBC 编程的内容就这些了，如果你已经全部掌握，还有事务、获取自增、获取元数据、ORM、DAO、数据连接池等内容可以自行了解一下。

另外，Statement 和 PreparedStatement 的异同，execute 和 executeUpdate 的区别等内容，这里做一些介绍。

Statement 和 PreparedStatement 的异同及优缺点

同：两者都是用来执 SQL 语句的

异：PreparedStatement 需要根据 SQL 语句来创建，它能够通过设置参数，指定相应的值，不是像 Statement 那样使用字符串拼接的方式。

PreparedStatement 的优点：

1、其使用参数设置，可读性好，不易记错。在 statement 中使用字符串拼接，可读性和维护性比较差。

2、其具有预编译机制，性能比 statement 更快。

3、其能够有效防止 SQL 注入攻击。

execute 和 executeUpdate 的区别

相同点：二者都能够执行增加、删除、修改等操作。

不同点：

1、execute 可以执行查询语句，然后通过 getResult 把结果取出来。executeUpdate 不能执行查询语句。

2、execute 返回 Boolean 类型，true 表示执行的是查询语句，false 表示执行的 insert、delete、update 等。executeUpdate 的返回值是 int，表示有多少条数据受到了影响。