

JDBC

一、概述

在 Java 中，数据库存取技术可分为如下几类：

- JDBC 直接访问数据库
- JDO 技术（Java Data Object）
- 第三方 O/R 工具，如 Hibernate, Mybatis 等

JDBC 是 java 访问数据库的基石，JDO, Hibernate 等只是更好的封装了 JDBC。

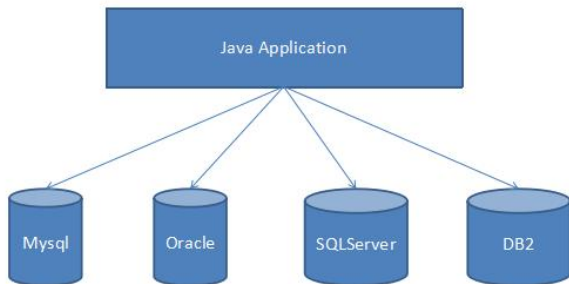
1、什么是 JDBC

JDBC (Java Database Connectivity) 是一个[独立于特定数据库管理系统（DBMS）、通用的 SQL 数据库存取和操作的公共接口](#)（一组 API），定义了用来访问数据库的标准 Java 类库，使用这个类库可以以一种标准的方法、方便地访问数据库资源

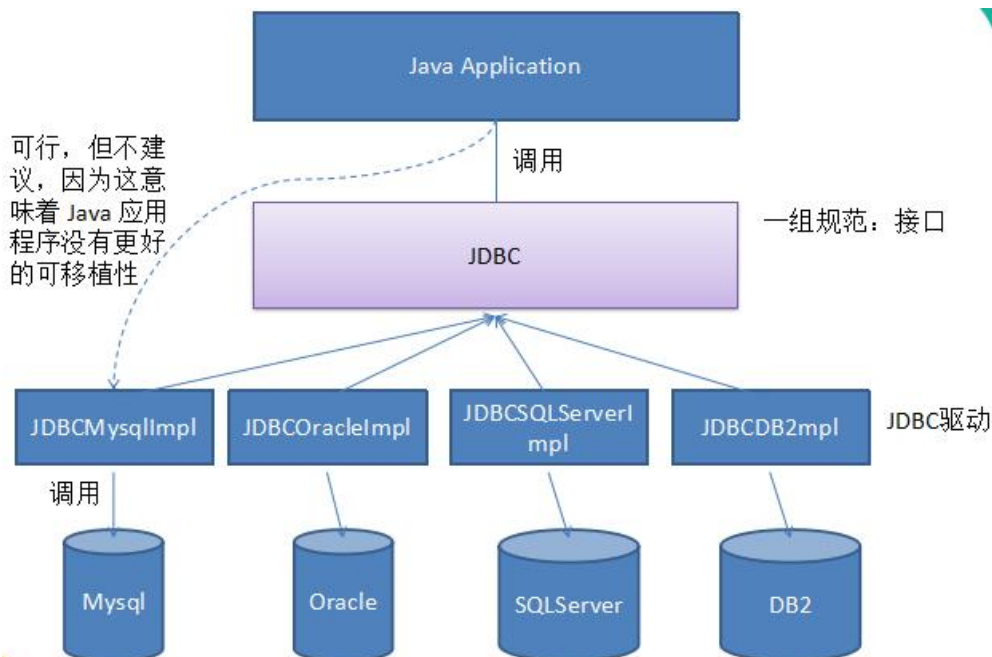
JDBC 为访问不同的数据库提供了一种[统一的途径](#)，为开发者屏蔽了一些细节问题。

JDBC 的目标是使 Java 程序员使用 JDBC 可以连接任何[提供了 JDBC 驱动程序](#)的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。

如果没有 JDBC，那么 Java 程序访问数据库时是这样的：



现在：



结论：

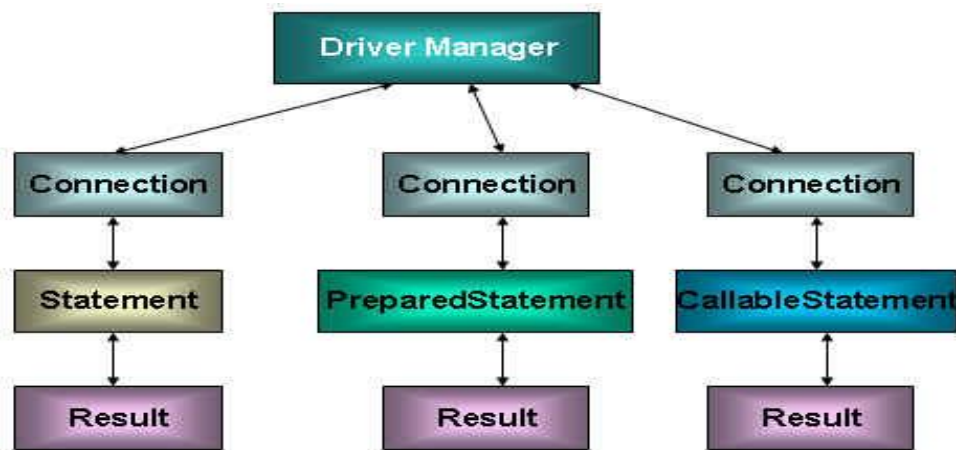
JDBC 是 SUN 公司（Oracle 公司甲骨文）提供一套用于数据库操作的接口 API，Java 程序员只需要面向这套接口编程

即可。

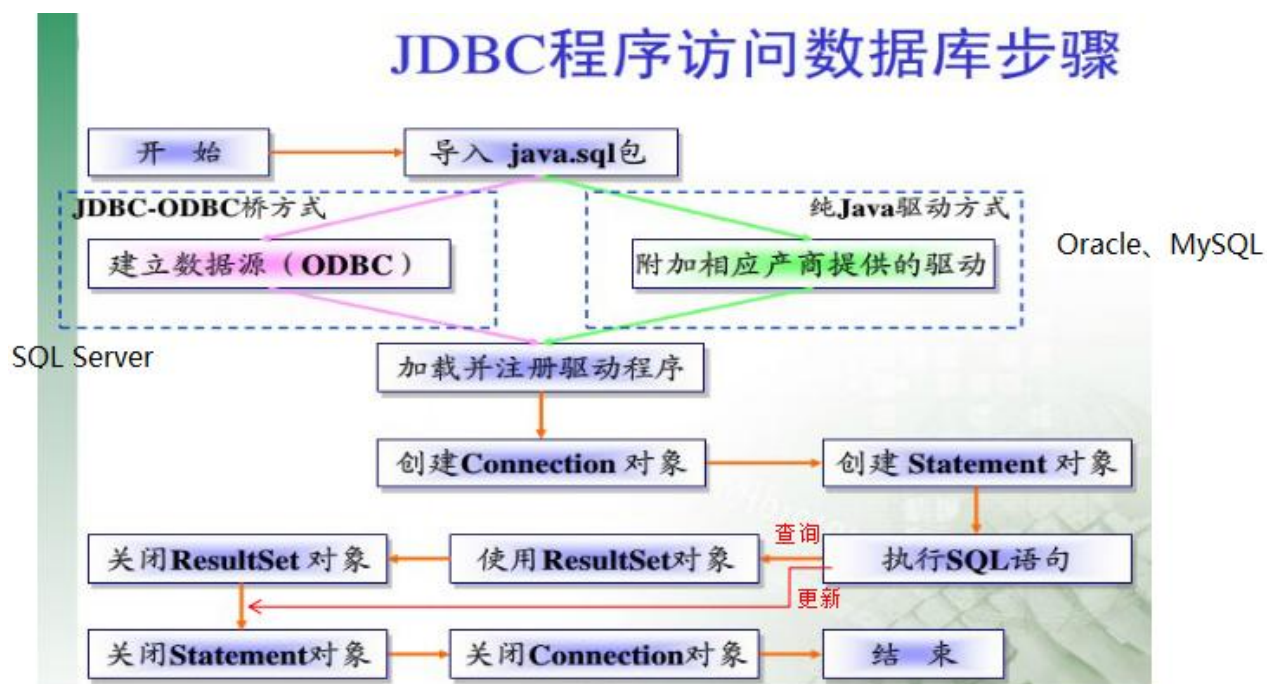
不同的数据库厂商，需要针对这套接口，提供不同实现。不同的实现的集合，即为不同数据库的驱动。

2、JDBC API

JDBC API 是一系列的接口，它统一和规范了应用程序与数据库的连接、执行 SQL 语句，并得到返回结果等各类操作。声明在 java.sql 与 javax.sql 包中。



3、JDBC 程序编写步骤



二、获取数据库连接

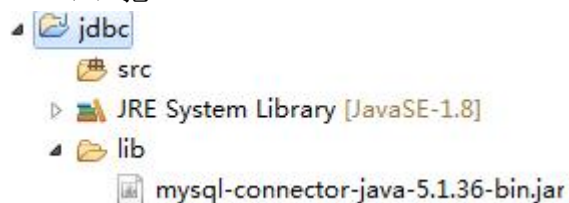
（一）引入 JDBC 驱动程序

驱动程序由数据库提供商提供下载。MySQL 的驱动下载地址：<http://dev.mysql.com/downloads/>

如何在 Java Project 项目应用中添加数据库驱动 jar:

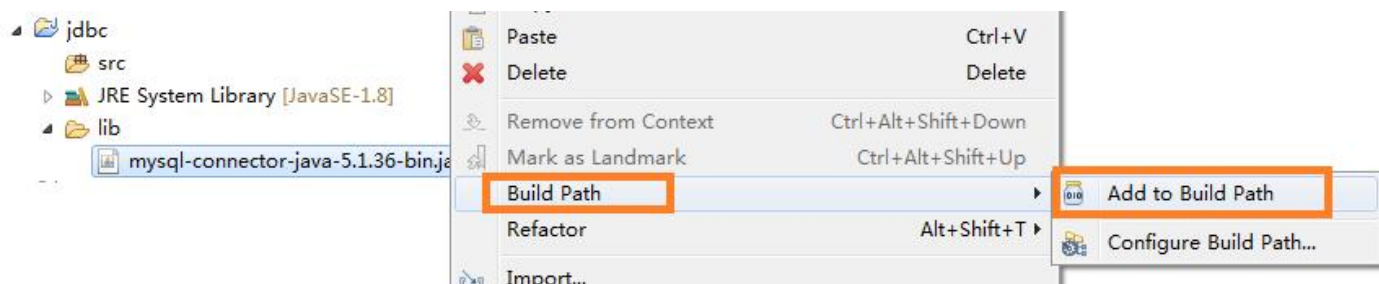


- (1) 把 mysql-connector-java-5.1.36-bin.jar 拷贝到项目中一个目录中

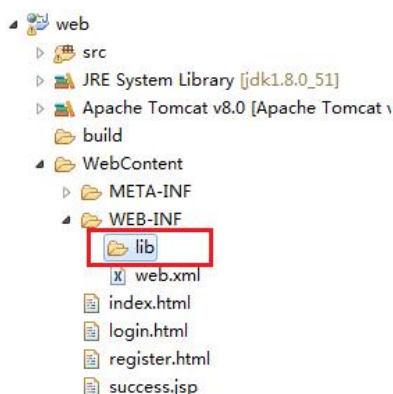


- (2) 添加到项目的类路径下

在驱动 jar 上右键-->Build Path-->Add to Build Path



注意: 如果是 Dynamic Web Project (动态的 web 项目) 话, 则是把驱动 jar 放到 WebContent (有的开发工具叫 WebRoot) 目录中的 WEB-INF 目录中的 lib 目录下即可



(二) 加载并注册驱动

加载并注册驱动:

加载驱动, 把驱动类加载到内存

注册驱动, 把驱动类的对象交给 DriverManager 管理, 用于后面创建连接等使用。

1、Class.forName()

因为 Driver 接口的驱动程序类都包含了静态代码块, 在这个静态代码块中, 会调用 DriverManager.registerDriver() 方法来注册自身的一个实例, 所以可以换一种方式来加载驱动。(即只要想办法让驱动类的这段静态代码块执行即可注册驱动类, 而要让这段静态代码块执行, 只要让该类被类加载器加载即可)

```

public class Driver extends NonRegisteringDriver implements java.sql.Driver {
    //
    // Register ourselves with the DriverManager
    //
    static {
        try {
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
}

```

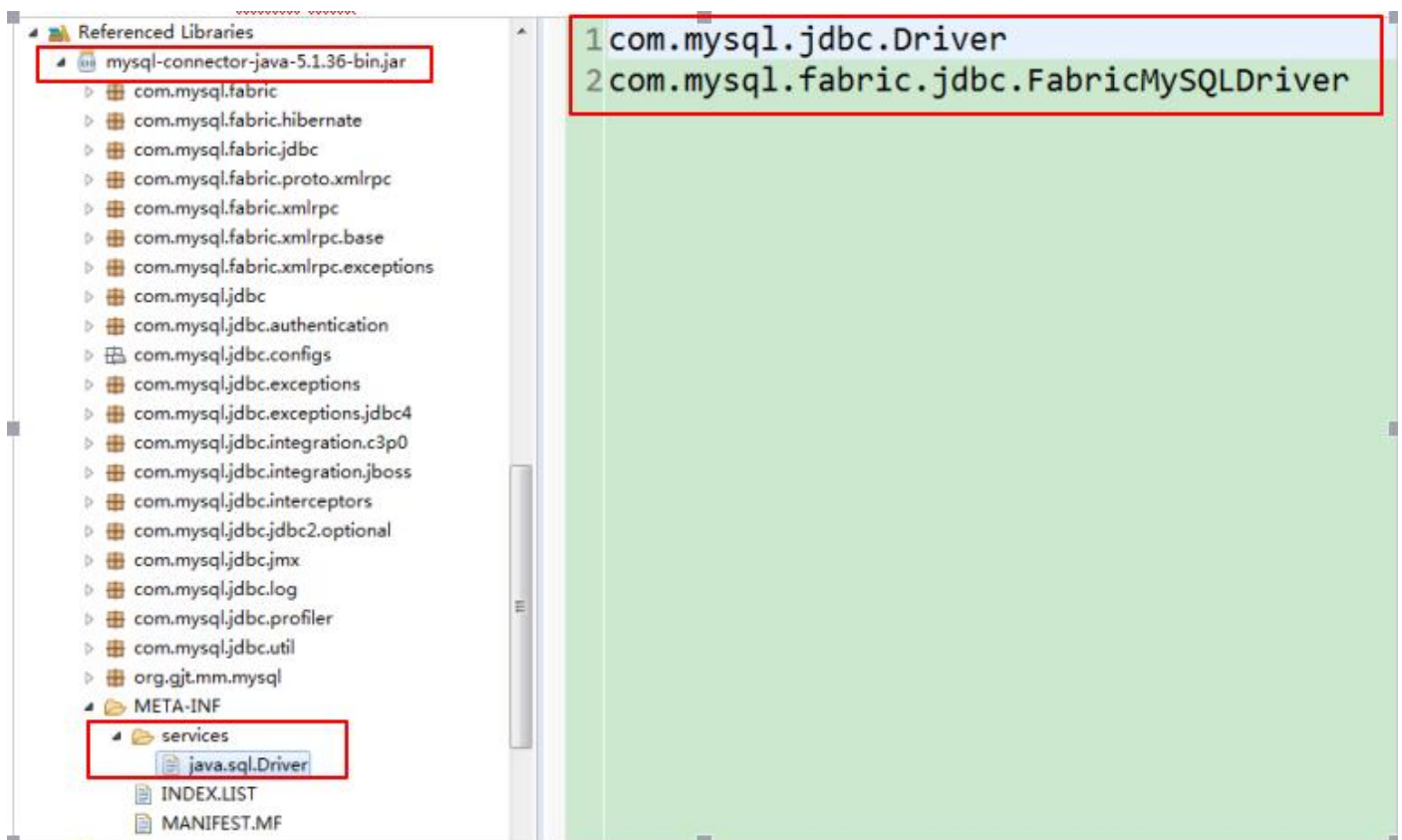
调用 Class 类的静态方法 `forName()`，向其传递要加载的 JDBC 驱动类的类名

//通过反射，加载与注册驱动类，**解耦合（不直接依赖）**

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

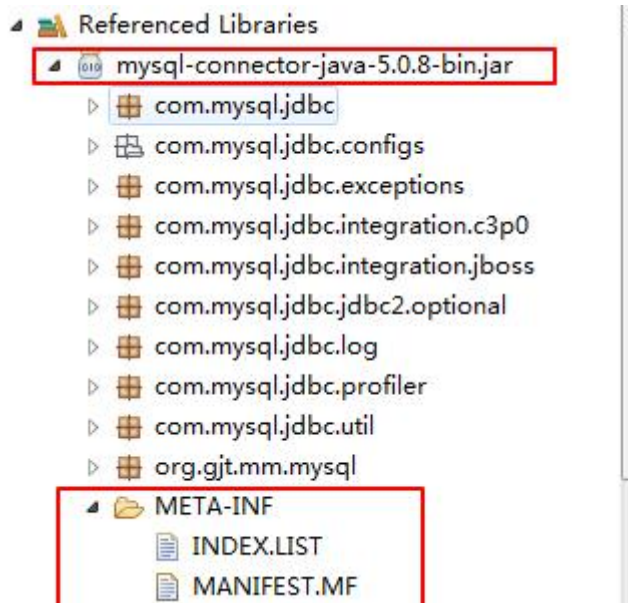
2、服务提供者框架（例如：JDBC 的驱动程序）自动注册（有版本要求）

符合 JDBC 4.0 规范的驱动程序包含了一个文件 `META-INF/services/java.sql.Driver`，在这个文件中提供了 JDBC 驱动实现的类名。例如：`mysql-connector-java-5.1.40-bin.jar` 文件中就可以找到 `java.sql.Driver` 文件，用文本编辑器打开文件就可以看到：`com.mysql.jdbc.Driver` 类。



JVM 的服务提供者框架在启动应用时就会注册服务，例如：MySQL 的 JDBC 驱动就会被注册，而原代码中的 `Class.forName("com.mysql.jdbc.Driver")` 仍然可以存在，但是不会起作用。

但是注意 `mysql-connector-java-5.0.8-bin.jar` 版本的 jar 中没有，如下



（三）获取数据库链接

可以通过 DriverManager 类建立到数据库的连接 Connection:

DriverManager 试图从已注册的 JDBC 驱动程序集中选择一个适当的驱动程序。

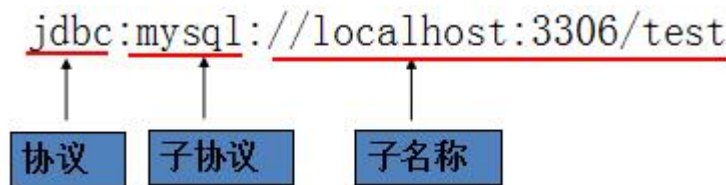
- public static Connection getConnection(String url)
- public static Connection getConnection(String url, String user, String password)
- public static Connection getConnection(String url, Properties info) 其中 Properties info 通常至少应该包括 "user" 和 "password" 属性

JDBC URL 用于标识一个被注册的驱动程序，驱动程序管理器通过这个 URL 选择正确的驱动程序，从而建立到数据库的连接。JDBC URL 的标准由三部分组成，各部分间用冒号分隔。

jdbc:<子协议>:<子名称>

- 协议：JDBC URL 中的协议总是 jdbc
- 子协议：子协议用于标识一个数据库驱动程序
- 子名称：一种标识数据库的方法。子名称可以依不同的子协议而变化，用子名称的目的是为了定位数据库提供足够的信息

例如：



- MySQL 的连接 URL 编写方式：
 - jdbc:mysql://主机名称:mysql 服务端口号/数据库名称?参数=值&参数=值
 - jdbc:mysql://localhost:3306/testdb
 - jdbc:mysql://localhost:3306/testdb?useUnicode=true&characterEncoding=utf8 (如果 JDBC 程序与服务端字符集不一致，会导致乱码，那么可以通过参数指定服务器端的字符集)
 - jdbc:mysql://localhost:3306/testdb?user=root&password=123456
- Oracle9i:
 - jdbc:oracle:thin:@主机名称:oracle 服务端口号:数据库名称
 - jdbc:oracle:thin:@localhost:1521:testdb
- SQLServer

- jdbc:sqlserver://主机名称:sqlserver 服务端口号:DatabaseName=数据库名称
- jdbc:sqlserver://localhost:1433:DatabaseName=testdb

//1、加载与注册驱动

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

//2、获取数据库连接

```
String url = "jdbc:mysql://localhost:3306/test";
```

```
Connection conn = DriverManager.getConnection(url, "root", "root");
```

（四）操作或访问数据库

数据库连接被用于向数据库服务器发送命令和 SQL 语句，并接受数据库服务器返回的结果。

其实一个数据库连接就是一个 Socket 连接。

在 java.sql 包中有 3 个接口分别定义了对数据库的调用的不同方式：

- Statement：用于执行静态 SQL 语句并返回它所生成结果的对象。
 - PreparedStatement：SQL 语句被预编译并存储在此对象中，然后可以使用此对象多次高效地执行该语句。
 - ◆ CallableStatement：用于执行 SQL 存储过程

Statement

通过调用 Connection 对象的 createStatement() 方法创建该对象

该对象用于执行静态的 SQL 语句，并且返回执行结果

Statement 接口中定义了下列方法用于执行 SQL 语句：

int executeUpdate(String sql)：执行更新操作 INSERT、UPDATE、DELETE

ResultSet executeQuery(String sql)：执行查询操作 SELECT

ResultSet

通过调用 Statement 对象的 executeQuery() 方法创建该对象

ResultSet 对象以逻辑表格的形式封装了执行数据库操作的结果集，ResultSet 接口由数据库厂商实现

ResultSet 对象维护了一个指向当前数据行的游标，初始的时候，游标在第一行之前，可以通过 ResultSet 对象的 next() 方法移动到下一行

ResultSet 接口的常用方法：

- boolean next()
- getXxx(String columnLabel)：columnLabel 使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称
- getXxx(int index)：索引从 1 开始
- ...

java类型	SQL类型
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
String	CHAR, VARCHAR, LONGVARCHAR
byte array	BINARY , VAR BINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

（五）释放资源

Connection、Statement、ResultSet 都是应用程序和数据库服务器的连接资源，使用后一定要关闭，可以在 finally 中关闭

演示未关闭后果：

```
package com.atguigu.conn;

import java.sql.Connection;
import java.sql.DriverManager;

public class TestConnectionClose {
    public static void main(String[] args) throws Exception{
        //1、加载与注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2、获取数据库连接
        String url = "jdbc:mysql://localhost:3306/test";

        //my.ini 中 max_connections=10
        for (int i = 0; i < 15; i++) {
            Connection conn = DriverManager.getConnection(url, "root", "123456");
            System.out.println(conn);
            //没有关闭，资源一直没有释放
        }
    }
}
```

（六）增、删、改、查示例代码

```
package com.atguigu.statement;
```

```

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;


import org.junit.Test;


/*
 * 网络编程: tcp
 *
 * 服务器端:
 * 1、ServerSocket server = new ServerSocket(3306);
 * 2、Socket socket = server.accept();
 * 3、InputStream input = socket.getInputStream();//接收 sql, 客户端传过来的
 * 4、在服务器执行 sql
 * 5、把结果给客户端
 *
 * 客户端:
 * 1、Socket socket = new Socket(服务器的 IP 地址, 3306);
 * 2、传 sql
 * 3、OutputStream out = socket.getOutputStream();
 * 4、out.write(sql);
 * 5、接收结果
 * 6、断开连接 out.close();socket.close();
 */

public class TestStatement {
    @Test

    public void testAdd()throws Exception{
        Class.forName("com.mysql.jdbc.Driver");

        String url = "jdbc:mysql://localhost:3306/test";
        String user = "root";
        String password = "123456";
        Connection conn = DriverManager.getConnection(url, user, password);

        Statement st = conn.createStatement();

        String sql = "INSERT INTO t_department(dname,description) VALUES('财务部','负责发钱工作')";
        int len = st.executeUpdate(sql);//把 insert,update,delete 都用这个方法

        if(len>0){
            System.out.println("添加成功");
        }else{
            System.out.println("添加失败");
        }

        st.close();
        conn.close();
    }
}

```


@Test

```
public void testUpdate()throws Exception{
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    Statement st = conn.createStatement();

    String sql = "UPDATE t_department SET description = '负责发工资、社保、公积金工作' WHERE dname ='财务部'";
    int len = st.executeUpdate(sql);//把 insert,update,delete 都用这个方法

    if(len>0){
        System.out.println("修改成功");
    }else{
        System.out.println("修改失败");
    }

    st.close();
    conn.close();
}
```

@Test

```
public void testDelete()throws Exception{
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    Statement st = conn.createStatement();

    String sql = "DELETE FROM t_department WHERE did =6";
    int len = st.executeUpdate(sql);//把 insert,update,delete 都用这个方法

    if(len>0){
        System.out.println("删除成功");
    }else{
        System.out.println("删除失败");
    }

    st.close();
    conn.close();
}
```

@Test

```
public void testSelect()throws Exception{
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    Statement st = conn.createStatement();

    String sql = "SELECT * FROM t_department";
    ResultSet rs = st.executeQuery(sql);//select 语句用 query 方法
    while(rs.next()){//是否有下一行
        //取这一行的单元格
        int id = rs.getInt(1);
        String name = rs.getString(2);
        String desc = rs.getString(3);

        System.out.println(id+"\t" + name + "\t" + desc);
    }

    rs.close();
    st.close();
    conn.close();
}
```

@Test

```
public void testSelect2()throws Exception{
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    Statement st = conn.createStatement();

    String sql = "SELECT did,dname FROM t_department";
    ResultSet rs = st.executeQuery(sql);//select 语句用 query 方法
    while(rs.next()){//是否有下一行
        //取这一行的单元格
        int id = rs.getInt("did");
        String name = rs.getString("dname");
        System.out.println(id+"\t" + name);
    }

    rs.close();
}
```

```

        st.close();

        conn.close();
    }
}

```

四、PreparedStatement

1、Statement 的不足

(1) SQL 拼接

(2) SQL 注入

SQL 注入是利用某些系统没有对用户输入的数据进行充分的检查，而在用户输入数据中注入非法的 SQL 语句段或命令，从而利用系统的 SQL 引擎完成恶意行为的做法。对于 Java 而言，要防范 SQL 注入，只要用 PreparedStatement 取代 Statement 就可以了。

(3) 处理 Blob 类型的数据

BLOB (binary large object)，二进制大对象，BLOB 常常是数据库中用来存储二进制文件的字段类型。插入 BLOB 类型的数据必须使用 PreparedStatement，因为 BLOB 类型的数据无法使用字符串拼接写的。MySQL 的四种 BLOB 类型(除了在存储的最大信息量上不同外，他们是等同的)

类型	大小(单位：字节)	如果还是报错：xxx too large，那么在 mysql 的安装目录下，找 my.ini 文件加上如下的配置参数： max_allowed_packet=16M 注意：修改了 my.ini 文件，一定要重新启动服务
TinyBlob	最大 255	
Blob	最大 65K	
MediumBlob	最大 16M	
LongBlob	最大 4G	

实际使用中根据需要存入的数据大小定义不同的 BLOB 类型。需要注意的是：如果存储的文件过大，数据库的性能会下降。

```

CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `head_picture` mediumblob,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

package com.atguigu.statement;

import java.sql.Connection;
import java.sql.DriverManager;

```

```
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

import org.junit.Test;

/*
 * Statement:
 * 1、SQL 拼接
 * 2、SQL 注入
 * 3、处理不了 Blob 类型的数据
 */
public class TestStatementProblem {

    @Test
    public void add() throws Exception{
        Scanner input = new Scanner(System.in);
        System.out.println("请输入姓名: ");
        String name = input.nextLine();

        System.out.println("请输入领导编号: ");
        int mid = input.nextInt();

        System.out.println("请输入部门编号: ");
        int did = input.nextInt();

        //1、连接数据库
        Class.forName("com.mysql.jdbc.Driver");

        String url = "jdbc:mysql://localhost:3306/1221db";
        String user = "root";
        String password = "123456";
        Connection conn = DriverManager.getConnection(url, user, password);

        //2、创建 Statement 对象
        Statement st = conn.createStatement();

        //3、编写 sql
        String sql = "INSERT INTO emp (ename,`mid`,did) VALUES('" + name+"'," + mid + "," + did + ")";

        //4、执行 sql
        int update = st.executeUpdate(sql);
        System.out.println(update>0?"添加成功":"添加失败");

        //5、释放资源
        st.close();
        conn.close();
    }
}
```

```

@Test
public void select() throws Exception{
    Scanner input = new Scanner(System.in);
    System.out.println("请输入姓名: ");
    String name = input.nextLine();

    //1、连接数据库
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/1221db";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    //2、写 sql
    //孙红雷 ' or '1' = '1
    String sql = "SELECT eid,ename,tel,gender,salary FROM t_employee WHERE ename = '" + name + "'";
    System.out.println(sql);
    // SELECT eid,ename,tel,gender,salary FROM t_employee WHERE ename = '孙红雷 ' or '1' = '1'

    //3、用 Statement 执行
    Statement st = conn.createStatement();

    //4、执行查询 sql
    ResultSet rs = st.executeQuery(sql);
    while(rs.next()){
        int id = rs.getInt(1);
        String ename = rs.getString(2);
        String tel = rs.getString(3);
        String gender = rs.getString(4);
        double salary = rs.getDouble(5);

        System.out.println(id+"\t" + ename + "\t" + tel + "\t" + gender + "\t" + salary);
    }

    //5、释放资源
    st.close();
    conn.close();
}

@Test
public void testAddBlob(){
    String sql = "INSERT INTO `user` (username,`password`,photo)VALUES('chai','123',没法在 String 中处理 Blob 类型的数据)";
}
}

```

2、PreparedStatement 概述

可以通过调用 Connection 对象的 `prepareStatement(String sql)` 方法获取 PreparedStatement 对象

PreparedStatement 接口是 Statement 的子接口，它表示一条预编译过的 SQL 语句

- PreparedStatement 对象所代表的 SQL 语句中的参数用问号(?)来表示，调用 PreparedStatement 对象的 `setXxx()` 方法来设置这些参数。`setXxx()` 方法有两个参数，第一个参数是要设置的 SQL 语句中的参数的索引(从 1 开始)，第二个是设置的 SQL 语句中的参数的值
- `ResultSet executeQuery()` 执行查询，并返回该查询生成的 `ResultSet` 对象。
- `int executeUpdate()`：执行更新，包括增、删、该

```
package com.atguigu.preparedstatement;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

import org.junit.Test;

/*
 * PreparedStatement: 是 Statement 子接口
 * 1、SQL 不需要拼接
 * 2、SQL 不会出现注入
 * 3、可以处理 Blob 类型的数据
 * tinyblob: 255 字节以内
 * blob: 65K 以内
 * mediumblob: 16M 以内
 * longblob: 4G 以内
 *
 * 如果还是报错: xxx too large, 那么在 mysql 的安装目录下, 找 my.ini 文件加上如下的配置参数:
 * max_allowed_packet=16M
 * 注意: 修改了 my.ini 文件, 一定要重新启动服务
 *
 */
public class TestPreparedStatement {
    @Test
    public void add() throws Exception {
        Scanner input = new Scanner(System.in);
        System.out.println("请输入姓名: ");
        String name = input.nextLine();

        System.out.println("请输入手机号码: ");
        String tel = input.nextLine();

        System.out.println("请输入性别: ");
        String gender = input.nextLine();
    }
}
```



```

System.out.println("请输入薪资: ");
double salary = input.nextDouble();

System.out.println("请输入部门编号: ");
int did = input.nextInt();

//1、连接数据库
Class.forName("com.mysql.jdbc.Driver");

String url = "jdbc:mysql://localhost:3306/test";
String user = "root";
String password = "123456";
Connection conn = DriverManager.getConnection(url, user, password);

//2、编写带? 的 SQL
String sql = "INSERT INTO t_employee (ename,tel,gender,salary,did) VALUES(?,?,?, ?,?)";

// 3、准备一个 PreparedStatement: 预编译 sql
PreparedStatement pst = conn.prepareStatement(sql); // 对带? 的 sql 进行预编译

// 4、把?用具体的值进行代替
pst.setString(1, name);
pst.setString(2, tel);
pst.setString(3, gender);
pst.setDouble(4, salary);
pst.setInt(5, did);

// 5、执行 sql
int len = pst.executeUpdate();
System.out.println(len>0?"添加成功":"添加失败");

// 6、释放资源
pst.close();
conn.close();
}

@Test
public void select() throws Exception {
    Scanner input = new Scanner(System.in);
    System.out.println("请输入姓名: ");
    String name = input.nextLine();

    //1、连接数据库
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

```

```

//2、编写带?的 sql
//孙红雷 ' or '1' = '1
String sql = "SELECT eid,ename,tel,gender,salary FROM t_employee WHERE ename = ?";

// 3、把带? 的 sql 语句进行预编译
PreparedStatement pst = conn.prepareStatement(sql);

// 4、把? 用具体的变量的赋值
pst.setString(1, name);

// 5、执行 sql
ResultSet rs = pst.executeQuery();
while (rs.next()) {
    int id = rs.getInt("eid");
    String ename = rs.getString("ename");
    String tel = rs.getString("tel");
    String gender = rs.getString("gender");
    double salary = rs.getDouble("salary");

    System.out.println(id + "\t" + ename + "\t" + tel + "\t" + gender + "\t" + salary);
}

// 6、释放资源
rs.close();
pst.close();
conn.close();
}

@Test
public void addBlob() throws Exception {
    Scanner input = new Scanner(System.in);
    System.out.println("请输入用户名: ");
    String username = input.nextLine();

    System.out.println("请输入密码: ");
    String pwd = input.nextLine();

    System.out.println("请指定照片的路径: ");
    String photoPath = input.nextLine();

    //1、连接数据库
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

```

```

//2、 INSERT INTO `user` VALUES(NULL,用户名,密码,照片)
String sql = "INSERT INTO `user` (username,`password`,head_picture)VALUES(?,?,?)";

// 3、准备一个 PreparedStatement: 预编译 sql
PreparedStatement pst = conn.prepareStatement(sql);// 对带? 的 sql 进行预编译

// 4、对? 进行设置
pst.setString(1, username);
pst.setString(2, pwd);
pst.setBlob(3, new FileInputStream(photoPath));

// 5、执行 sql
int len = pst.executeUpdate();
System.out.println(len > 0 ? "添加成功" : "添加失败");

// 6、释放资源
pst.close();
conn.close();
}
}

```

3、PreparedStatement vs Statement

- 代码的可读性和可维护性. Statement 的 **sql 拼接**是个难题。
- PreparedStatement 可以防止 SQL 注入
- PreparedStatement 可以处理 Blob 类型的数据
- PreparedStatement 能最大可能提高性能: (**Oracle 和 PostgreSQL8 是这样,但是对于 MySQL 不一定比 Statement 高**)
 - DBServer 会对预编译语句提供性能优化。因为预编译语句有可能被重复调用,所以语句在被 DBServer 的编译器编译后的执行代码被缓存下来,那么下次调用时只要是相同的预编译语句就不需要编译,只要将参数直接传入编译过的语句执行代码中就会得到执行。

4、JDBC 取得数据库自动生成的主键

获取自增长的键值:

(1) 在创建 PreparedStatement 对象时

原来:

```
PreparedStatement pst = conn.prepareStatement(sql);
```

现在:

```
PreparedStatement pst = conn.prepareStatement(orderInsert,Statement.RETURN_GENERATED_KEYS);
```

(2) 原来执行更新

原来:

```
int len = pst.executeUpdate();
```

现在:

```
int len = pst.executeUpdate();
```

```
ResultSet rs = pst.getGeneratedKeys();
```

```
if(rs.next()){
```

Object key = rs.getObject(第几列); //获取自增长的键值

}

```
package com.atguigu.preparedstatement;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class TestGenerateKey {

    public static void main(String[] args) throws Exception{
        Scanner input = new Scanner(System.in);
        System.out.println("请输入姓名: ");
        String name = input.nextLine();

        System.out.println("请输入手机号码: ");
        String tel = input.nextLine();

        System.out.println("请输入性别: ");
        String gender = input.nextLine();

        System.out.println("请输入薪资: ");
        double salary = input.nextDouble();

        System.out.println("请输入部门编号: ");
        int did = input.nextInt();

        //1、连接数据库
        Class.forName("com.mysql.jdbc.Driver");

        String url = "jdbc:mysql://localhost:3306/test";
        String user = "root";
        String password = "123456";
        Connection conn = DriverManager.getConnection(url, user, password);

        //2、编写带? 的 SQL
        String sql = "INSERT INTO t_employee (ename,tel,gender,salary,did) VALUES(?,?,?, ?,?)";

        // 3、准备一个 PreparedStatement: 预编译 sql
        // 执行添加语句, 如果需要获取自增长的键值, 那么在此处要告知 mysql 服务器, 在创建 PreparedStatement 对象时, 增加一个参数
        // autoGeneratedKeys - 指示是否应该返回自动生成的键的标志, 它是 Statement.RETURN_GENERATED_KEYS 或 Statement.NO_GENERATED_KEYS 之一
        PreparedStatement pst = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);

        // 4、把?用具体的值进行代替
```

```

        pst.setString(1, name);
        pst.setString(2, tel);
        pst.setString(3, gender);
        pst.setDouble(4, salary);
        pst.setInt(5, did);

        // 5、执行 sql
        int len = pst.executeUpdate();
        System.out.println(len>0?"添加成功":"添加失败");

        ResultSet rs = pst.getGeneratedKeys();
        if(rs.next()){
            System.out.println("新员工编号是: " + rs.getObject(1));
        }

        // 6、释放资源
        pst.close();
        conn.close();
    }
}

```

5、批处理

当需要成批插入或者更新记录时。可以采用 Java 的批量更新机制，这一机制允许多条语句一次性提交给数据库批量处理。通常情况下比单独提交处理更有效率。

JDBC 的批量处理语句包括下面两个方法：

- `addBatch()`：添加需要批量处理的 SQL 语句或参数
- `executeBatch()`：执行批量处理语句；

通常我们会遇到两种批量执行 SQL 语句的情况：

- 多条 SQL 语句的批量处理；
- 一个 SQL 语句的批量传参；

注意：

JDBC 连接 MySQL 时，如果要使用批处理功能，请在 url 中加参数 `?rewriteBatchedStatements=true` `PreparedStatement` 作批处理插入时使用 `values`（使用 `value` 没有效果）

```

package com.atguigu.preparedstatement;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

import org.junit.Test;

public class TestBatch {
    @Test
    public void noBatch() throws Exception{

```

```

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

String url = "jdbc:mysql://localhost:3306/test";
String user = "root";
String password = "123456";
Connection conn = DriverManager.getConnection(url, user, password);

String sql = "INSERT INTO t_department(dname,description) VALUES(?,?)";
PreparedStatement st = conn.prepareStatement(sql);

for(int i=0; i<1000; i++){
    st.setString(1, "测试部门" + i);
    st.setString(2, "测试部门描述" + i);

    st.executeUpdate();
}

st.close();
conn.close();
}

@Test
public void useBatch()throws Exception{
    Class.forName("com.mysql.jdbc.Driver");

    String url = "jdbc:mysql://localhost:3306/test?rewriteBatchedStatements=true";
    String user = "root";
    String password = "123456";
    Connection conn = DriverManager.getConnection(url, user, password);

    String sql = "INSERT INTO t_department(dname,description) VALUES(?,?)";
    PreparedStatement st = conn.prepareStatement(sql);

    for(int i=0; i<1000; i++){
        st.setString(1, "测试部门" + i);
        st.setString(2, "测试部门描述" + i);

        st.addBatch();
    }

    st.executeBatch();

    st.close();
    conn.close();
}
}

```


6、事务

JDBC 程序中当**一个连接对象**被创建时，默认情况下是自动提交事务：每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。

JDBC 程序中为了让多个 SQL 语句作为一个事务执行：**（重点）**

- 调用 Connection 对象的 `setAutoCommit(false)`；以取消自动提交事务
- 在所有的 SQL 语句都成功执行后，调用 `commit()`；方法提交事务
- 在其中某个操作失败或出现异常时，调用 `rollback()`；方法回滚事务
- 若此时 Connection 没有被关闭，则需要恢复其自动提交状态 `setAutoCommit(true)`；

注意：

如果多个操作，每个操作使用的是自己单独的连接，则无法保证事务。即同一个事务的多个操作必须在同一个连接下

```
package com.atguigu.transaction;

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

public class TestTransaction {
    public static void main(String[] args){
        Connection conn = null;
        try {
            //1、连接数据库
            Class.forName("com.mysql.jdbc.Driver");

            String url = "jdbc:mysql://localhost:3306/test";
            String user = "root";
            String password = "123456";
            conn = DriverManager.getConnection(url, user, password);
            //设置手动提交
            conn.setAutoCommit(false);

            String sql1 = "update t_department set description = ? where did = ?";
            PreparedStatement pst1 = conn.prepareStatement(sql1);
            pst1.setObject(1, "挣大钱的");
            pst1.setObject(2, 4);
            int len1 = pst1.executeUpdate();
            System.out.println(len1>0?"更新部门信息成功":"更新部门信息失败");
            pst1.close();

            String sql2 = "update t_employee set salary = salary + ? where did = ?";
            PreparedStatement pst2 = conn.prepareStatement(sql2);
            pst2.setObject(1, 20000);
            pst2.setObject(2, 4);
            int len2 = pst2.executeUpdate();
            System.out.println(len2>0?"更新部门信息成功":"更新部门信息失败");
            pst2.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```

        conn.commit();
    } catch (Exception e) {
        try {
            if(conn!=null){
                conn.rollback();
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    } finally{
        try {
            if(conn!=null){
                //恢复自动提交
                conn.setAutoCommit(true);
                //释放连接
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}
}

```

五、数据库连接池

1、数据库连接池

(1) 数据库连接池的必要性

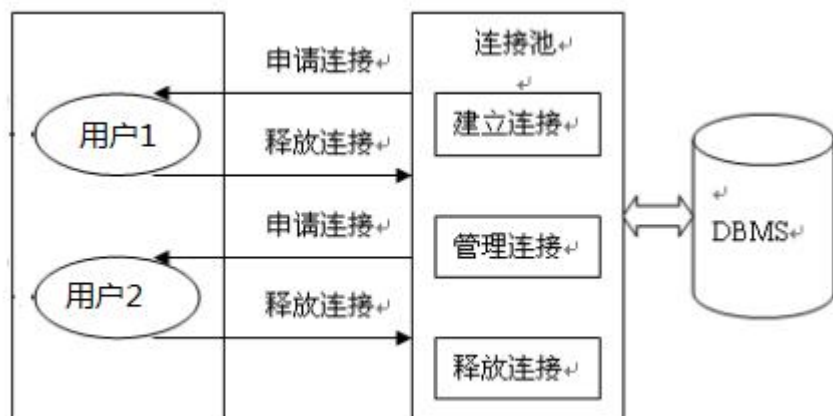
不使用数据库连接池存在的问题:

- 普通的 JDBC 数据库连接使用 `DriverManager` 来获取,每次向数据库建立连接的时候都要将 `Connection` 加载到内存中,再验证 IP 地址,用户名和密码(得花费 0.05s~1s 的时间)。需要数据库连接的时候,就向数据库要求一个,执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。**数据库的连接资源并没有得到很好的重复利用**。若同时有几百人甚至几千人在线,频繁的进行数据库连接操作将占用很多的系统资源,严重的甚至会造成服务器的崩溃。
- 对于每一次数据库连接,使用完后都得断开。否则,如果程序出现异常而**未能关闭**,将会导致**数据库系统中的内存泄漏**,最终将导致重启数据库。
- 这种开发不能控制被创建的**连接对象数**,系统资源会被毫无顾及的分配出去,如连接**过多**,也可能导致**内存泄漏,服务器崩溃**。

为解决传统开发中的数据库连接问题,可以采用**数据库连接池技术 (connection pool)**。

数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接,当需

要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是重新建立一个。连接池的**最大数据库连接数量**限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。



数据库连接池技术的优点：

- **资源重用：**
 - 由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。
- **更快的系统反应速度**
 - 数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间
- **新的资源分配手段**
 - 对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置，实现某一应用最大可用数据库连接数的限制，避免某一应用独占所有的数据库资源
- **统一的连接管理，避免数据库连接泄露**
 - 在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露

（2）多种开源的数据库连接池

JDBC 的数据库连接池使用 `javax.sql.DataSource` 来表示，`DataSource` 只是一个接口，该接口通常由服务器(Weblogic, WebSphere, Tomcat)提供实现，也有一些开源组织提供实现：

- DBCP 是 Apache 提供的数据库连接池，速度相对 c3p0 较快，但因自身存在 BUG，Hibernate3 已不再提供支持
- C3P0 是一个开源组织提供的一个数据库连接池，速度相对较慢，稳定性还可以
- Proxool 是 sourceforge 下的一个开源项目数据库连接池，有监控连接池状态的功能，稳定性较 c3p0 差一点
- BoneCP 是一个开源组织提供的数据库连接池，速度快
- Druid 是阿里提供的数据库连接池，据说是集 DBCP、C3P0、Proxool 优点于一身的数据库连接池，但是速度不知道是否有 BoneCP 快

`DataSource` 通常被称为数据源，它包含连接池和连接池管理两个部分，习惯上也经常把 `DataSource` 称为连接池

注意：

- 数据源和数据库连接不同，数据源无需创建多个，它是产生数据库连接的工厂，因此整个应用只需要一个数据源即可。
- 当数据库访问结束后，程序还是像以前一样关闭数据库连接：`conn.close()`；但 `conn.close()` 并没有关闭数据库的物理连接，它仅仅把数据库连接释放，归还给了数据库连接池。

(3) Druid（德鲁伊）数据源

Druid 是阿里巴巴开源平台上一个数据库连接池实现，它结合了 C3P0、DBCP、Proxool 等 DB 池的优点，同时加入了日志监控，可以很好的监控 DB 池连接和 SQL 的执行情况，可以说是针对监控而生的 DB 连接池，据说是目前最好的连接池。



druid-1.1.10.jar

```
package com.atguigu.druid;

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

import javax.sql.DataSource;

import com.alibaba.druid.pool.DruidDataSourceFactory;

public class TestDruid {
    public static void main(String[] args) throws Exception {
        Properties pro = new Properties();
        pro.load(TestDruid.class.getClassLoader().getResourceAsStream("druid.properties"));
        DataSource ds = DruidDataSourceFactory.createDataSource(pro);
        Connection conn = ds.getConnection();
        System.out.println(conn);
    }
}
```

```
url=jdbc:mysql://localhost:3306/test?rewriteBatchedStatements=true
username=root
password=123456
driverClassName=com.mysql.jdbc.Driver
initialSize=10
maxActive=20
maxWait=1000
filters=wall
```

详细配置参数：

配置	缺省	说明
name		配置这个属性的意义在于，如果存在多个数据源，监控的时候可以通过名字来区分开来。 如果没有配置，将会生成一个名字，格式是： "DataSource-" + System.identityHashCode(this)
jdbcUrl		连接数据库的 url，不同数据库不一样。例如：mysql： jdbc:mysql://10.20.153.104:3306/druid2 oracle： jdbc:oracle:thin:@10.20.149.85:1521:ocnauto
username		连接数据库的用户名

配置	缺省	说明
password		连接数据库的密码。如果你不希望密码直接写在配置文件中，可以使用 <code>ConfigFilter</code> 。详细看这里： https://github.com/alibaba/druid/wiki/%E4%BD%BF%E7%94%A8ConfigFilter
driverClassName		根据 url 自动识别 这一项可配可不配,如果不配置 druid 会根据 url 自动识别 dbType, 然后选择相应的 driverClassName(建议配置下)
initialSize	0	初始化时建立物理连接的个数。初始化发生在显示调用 init 方法，或者第一次 getConnection 时
maxActive	8	最大连接池数量
maxIdle	8	已经不再使用，配置了也没效果
minIdle		最小连接池数量
maxWait		获取连接时最大等待时间，单位毫秒。配置了 maxWait 之后，缺省启用公平锁，并发效率会有所下降,如果需要可以通过配置 useUnfairLock 属性为 true 使用非公平锁。
poolPreparedStatements	false	是否缓存 preparedStatement，也就是 PSCache。PSCache 对支持游标的数据库性能提升巨大，比如说 oracle。在 mysql 下建议关闭。
maxOpenPreparedStatements	-1	要启用 PSCache，必须配置大于 0，当大于 0 时，poolPreparedStatements 自动触发修改为 true。在 Druid 中，不会存在 Oracle 下 PSCache 占用内存过多的问题，可以把这个数值配置大一些，比如说 100
validationQuery		用来检测连接是否有效的 sql，要求是一个查询语句。如果 validationQuery 为 null，testOnBorrow、testOnReturn、testWhileIdle 都不会起作用。
testOnBorrow	true	申请连接时执行 validationQuery 检测连接是否有效，做了这个配置会降低性能。
testOnReturn	false	归还连接时执行 validationQuery 检测连接是否有效，做了这个配置会降低性能
testWhileIdle	false	建议配置为 true，不影响性能，并且保证安全性。申请连接的时候检测，如果空闲时间大于 timeBetweenEvictionRunsMillis，执行 validationQuery 检测连接是否有效。
timeBetweenEvictionRunsMillis		有两个含义： 1)Destroy 线程会检测连接的间隔时间 2)testWhileIdle 的判断依据，详细看 testWhileIdle 属性的说明
numTestsPerEvictionRun		不再使用，一个 DruidDataSource 只支持一个 EvictionRun
minEvictableIdleTimeMillis		
connectionInitSqls		物理连接初始化的时候执行的 sql

配置	缺省	说明
exceptionSorter		根据 dbType 自动识别 当数据库抛出一些不可恢复的异常时，抛弃连接
filters		属性类型是字符串，通过别名的方式配置扩展插件，常用的插件有： 监控统计用的 filter:stat 日志用的 filter:log4j 防御 sql 注入的 filter:wall
proxyFilters		类型是 List，如果同时配置了 filters 和 proxyFilters，是组合关系，并非替换关系

2、ThreadLocal

JDK 1.2 的版本中就提供 java.lang.ThreadLocal，ThreadLocal 为解决多线程程序的并发问题提供了一种新的思路。使用这个工具类可以很简洁地编写出优美的多线程程序。

ThreadLocal 用于保存某个线程共享变量，原因是在 Java 中，每一个线程中都有一个 ThreadLocalMap<ThreadLocal, Object>，其 key 就是一个 ThreadLocal，而 Object 即为该线程的共享变量。而这个 map 是通过 ThreadLocal 的 set 和 get 方法操作的。对于同一个 static ThreadLocal，不同线程只能从中 get，set，remove 自己的变量，而不会影响其他线程的变量。

- 1、ThreadLocal.get: 获取 ThreadLocal 中当前线程共享变量的值。
- 2、ThreadLocal.set: 设置 ThreadLocal 中当前线程共享变量的值。
- 3、ThreadLocal.remove: 移除 ThreadLocal 中当前线程共享变量的值。
- 4、ThreadLocal.initialValue: ThreadLocal 没有被当前线程赋值时或当前线程刚调用 remove 方法后调用 get 方法，返回此方法值。

3、封装 JDBCTools

```
package com.atguigu.util;

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

import javax.sql.DataSource;

import com.alibaba.druid.pool.DruidDataSourceFactory;

/*
 * 获取连接或释放连接的工具类
 */
public class JDBCTools {
    // 1、数据源,即连接池
    private static DataSource dataSource;

    // 2、ThreadLocal 对象
    private static ThreadLocal<Connection> threadLocal;

    static {
```



```

    try {
        //1、读取 druid.properties 文件
        Properties pro = new Properties();
        pro.load(JDBCTools.class.getClassLoader().getResourceAsStream("druid.properties"));

        //2、连接连接池
        dataSource = DruidDataSourceFactory.createDataSource(pro);

        //3、创建线程池
        threadLocal = new ThreadLocal<>();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * 获取连接的方法
 *
 * @return
 * @throws SQLException
 */
public static Connection getConnection() {
    // 从当前线程中获取连接
    Connection connection = threadLocal.get();
    if (connection == null) {
        // 从连接池中获取一个连接
        try {
            connection = dataSource.getConnection();
            // 将连接与当前线程绑定
            threadLocal.set(connection);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return connection;
}

/**
 * 释放连接的方法
 *
 * @param connection
 */
public static void releaseConnection() {
    // 获取当前线程中的连接
    Connection connection = threadLocal.get();
    if (connection != null) {
        try {
            connection.close();
            // 将已经关闭的连接从当前线程中移除

```

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

六、Apache—DBUtils 简介

commons-dbutils 是 Apache 组织提供的一个开源 JDBC 工具类库，它是对 JDBC 的简单封装，学习成本极低，并且使用 dbutils 能极大简化 jdbc 编码的工作量，同时也不会影响程序的性能。



commons-dbutils-1.6.jar

1、DbUtils 类

DbUtils：提供如关闭连接、装载 JDBC 驱动程序等常规工作的工具类，里面的所有方法都是静态的。主要方法如下：

- `public static void close(…) throws java.sql.SQLException`： DbUtils 类提供了三个重载的关闭方法。这些方法检查所提供的参数是不是 NULL，如果不是的话，它们就关闭 Connection、Statement 和 ResultSet。
- `public static void closeQuietly(…)`：这一类方法不仅能在 Connection、Statement 和 ResultSet 为 NULL 情况下避免关闭，还能隐藏一些在程序中抛出的 SQLException。
- `public static void commitAndClose(Connection conn) throws SQLException` 用来提交连接的事务，然后关闭连接
- `public static void commitAndCloseQuietly(Connection conn)`： 用来提交连接的事务，然后关闭连接，并且在关闭连接时不抛出 SQL 异常。
- `public static void rollback(Connection conn) throws SQLException` 允许 conn 为 null，因为方法内部做了判断
- `public static void rollbackAndClose(Connection conn) throws SQLException`
- `rollbackAndCloseQuietly(Connection)`
- `public static boolean loadDriver(java.lang.String driverClassName)`：这一方装载并注册 JDBC 驱动程序，如果成功就返回 true。使用该方法，你不需要捕捉这个异常 ClassNotFoundException。

2、QueryRunner 类

该类封装了 SQL 的执行，是线程安全的。

- (1) 可以实现增、删、改、查、批处理、
- (2) 考虑了事务处理需要共用 Connection。
- (3) 该类最主要的就是简单化了 SQL 查询，它与 ResultSetHandler 组合在一起使用可以完成大部分的数据库操作，能够大大减少编码量。

QueryRunner 类提供了两个构造方法：

- `QueryRunner()`：默认的构造方法
- `QueryRunner(DataSource ds)`：需要一个 javax.sql.DataSource 来作参数的构造方法。

(1) 更新

- `public int update(Connection conn, String sql, Object... params) throws SQLException:`用来执行一个更新（插入、更新或删除）操作。
-

(2) 插入

- `public <T> T insert(Connection conn,String sql,ResultSetHandler<T> rsh, Object... params) throws SQLException:` 只支持 INSERT 语句，其中 rsh - The handler used to create the result object from the ResultSet of **auto-generated keys**. 返回值: An object generated by the handler.即自动生成的键值
-

(3) 批处理

- `public int[] batch(Connection conn,String sql,Object[][] params)throws SQLException:` INSERT, UPDATE, or DELETE 语句
- `public <T> T insertBatch(Connection conn,String sql,ResultSetHandler<T> rsh,Object[][] params)throws SQLException:` 只支持 INSERT 语句
-

(4) 使用 QueryRunner 类实现查询

- `public Object query(Connection conn, String sql, ResultSetHandler rsh,Object... params) throws SQLException:` 执行一个查询操作，在这个查询中，对象数组中的每个元素值被用来作为查询语句的置换参数。该方法会自行处理 PreparedStatement 和 ResultSet 的创建和关闭。
-

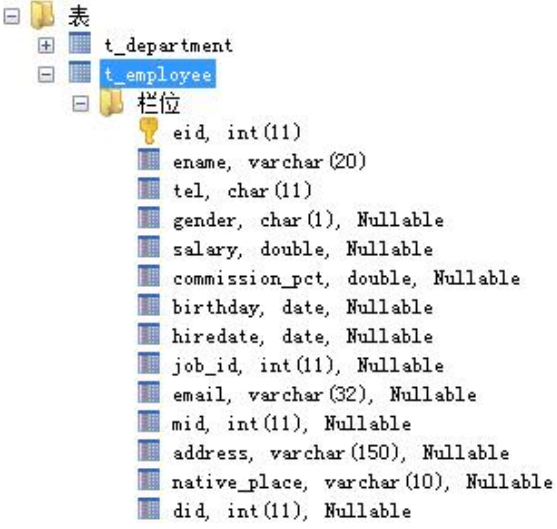
3、ResultSetHandler 接口

该接口用于处理 `java.sql.ResultSet`，将数据按要求转换为另一种形式。`ResultSetHandler` 接口提供了一个单独的方法：`Object handle (java.sql.ResultSet rs)`该方法的返回值将作为 `QueryRunner` 类的 `query()`方法的返回值。

该接口有如下实现类可以使用：

- `ArrayHandler`：把结果集中的第一行数据转成对象数组。
- `ArrayListHandler`：把结果集中的每一行数据都转成一个数组，再存放到 List 中。
- `BeanHandler`：将结果集中的第一行数据封装到一个对应的 `JavaBean` 实例中。
- `BeanListHandler`：将结果集中的每一行数据都封装到一个对应的 `JavaBean` 实例中，存放到 List 里。
- `ColumnListHandler`：将结果集中某一列的数据存放到 List 中。
- `KeyedHandler(name)`：将结果集中的每一行数据都封装到一个 Map 里，再把这些 map 再存到一个 map 里，其 key 为指定的 key。
- `MapHandler`：将结果集中的第一行数据封装到一个 Map 里，key 是列名，value 就是对应的值。
- `MapListHandler`：将结果集中的每一行数据都封装到一个 Map 里，然后再存放到 List

4、表与 JavaBean

 <pre>eid, int(11) ename, varchar(20) tel, char(11) gender, char(1), Nullable salary, double, Nullable commission_pct, double, Nullable birthday, date, Nullable hiredate, date, Nullable job_id, int(11), Nullable email, varchar(32), Nullable mid, int(11), Nullable address, varchar(150), Nullable native_place, varchar(10), Nullable did, int(11), Nullable</pre> <p>int,double 等在 Java 中都用包装类, 因为 mysql 中的所有类型都可能是 NULL, 而 Java 只有引用数据类型才有 NULL 值</p>	<pre>public class Employee { private Integer eid; private String ename; private String tel; private String gender;//mysql 中用 char,在 Java 中也要用 String private Double salary; private Double commissionPct; private Date birthday;//此处用 String 或 Date private Date hiredate; private Integer jobId; private String email; private Integer mid; private String address; private String nativePlace; private Integer did; ... }</pre>
--	--

通过给列取别名的方式, 来告知数据库的列名与其对应实体的属性名

```
select flow_id AS id,
       type AS type,
       card_id AS cardID,
       exam_card AS examID,
       student_name AS name,
       location AS location,
       grade AS grade
from examstudents;
```

```
public class Student {

    private int id;

    private int type;


    private String cardID;

    private String examID;

    private String name;

    private String location;

    private int grade;
```



名	类型	长度
flow_id	int	11
type	int	11
card_id	char	18
exam_card	char	15
student_name	varchar	20
location	varchar	20
grade	int	11

5、示例代码

```
package com.atguigu.apache.dbutils;

import java.util.List;
import java.util.Map;

import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;
import org.apache.commons.dbutils.handlers.BeanListHandler;
```

```

import org.apache.commons.dbutils.handlers.MapListHandler;
import org.apache.commons.dbutils.handlers.ScalarHandler;
import org.junit.Test;

import com.atguigu.bean.Book;
import com.atguigu.util.JDBCTools;

public class TestDBUtils {
    QueryRunner qr = new QueryRunner();

    @Test
    public void testUpdate() throws Exception {
        Book book = new Book(0, "红楼梦", "曹雪芹", 88.8, 0, 200, null);
        // 写 sql 语句
        String sql = "insert into books(title,author,price,sales,stock,img_path) values(?,?,?,?,?,?)";
        qr.update(JDBCTools.getConnection(), sql, book.getTitle(), book.getAuthor(), book.getPrice(),
book.getSales(),
            book.getStock(), book.getImgPath());
    }

    @Test
    public void testInsert() throws Exception {
        Book book = new Book(0, "红楼梦", "曹雪芹", 88.8, 0, 200, null);
        // 写 sql 语句
        String sql = "insert into books(title,author,price,sales,stock,img_path) values(?,?,?,?,?,?)";
        Long id = qr.insert(JDBCTools.getConnection(), sql, new ScalarHandler<Long>(), book.getTitle(),
            book.getAuthor(), book.getPrice(), book.getSales(), book.getStock(), book.getImgPath());
        System.out.println("新书编号: " + id);
    }

    @Test
    public void testBatch() throws Exception {
        Object[][] params = new Object[2][3];
        params[0][0] = 1;
        params[0][1] = 1;
        params[0][2] = 1;

        params[1][0] = 1;
        params[1][1] = 1;
        params[1][2] = 2;

        String sql = "update books set sales = sales + ? , stock = stock - ? where id = ?";
        qr.batch(JDBCTools.getConnection(), sql, params);
    }

    @Test
    public void testGetBean() throws Exception {
        // 写 sql 语句
        // 当 JavaBean 的属性名与字段名不一致时, 可以通过指定别名告知属性名
    }
}

```

```

        String sql = "select id,title,author,price,sales,stock,img_path imgPath from books where id = ?";
        Book b = qr.query(JDBCTools.getConnection(), sql, new BeanHandler<Book>(Book.class), 2);
        System.out.println(b);
    }

    @Test
    public void testGetBeanList() throws Exception {
        // 写 sql 语句
        // 当 JavaBean 的属性名与字段名不一致时, 可以通过指定别名告知属性名
        String sql = "select id,title,author,price,sales,stock,img_path imgPath from books where price
between ? and ? ";
        List<Book> list = qr.query(JDBCTools.getConnection(), sql, new BeanListHandler<Book>(Book.class), 10,
20);

        for (Book b : list) {
            System.out.println(b);
        }
    }

    @Test
    public void testGetSingleValue() throws Exception {
        // 获取数据库中图书的总记录数
        String sql = "select count(*) from books";
        Long count = qr.query(JDBCTools.getConnection(), sql, new ScalarHandler<Long>());
        System.out.println(count);
    }

    @Test
    public void testGetMap() throws Exception {
        String sql = "SELECT user_id userId,COUNT(*) FROM `orders` GROUP BY user_id";
        List<Map<String, Object>> list = qr.query(JDBCTools.getConnection(), sql, new MapListHandler());
        for (Map<String, Object> map : list) {
            System.out.println(map);
        }
    }
}

```

```

package com.atguigu.bean;

```

```

public class Book {
    private Integer id;
    private String title;
    private String author;
    private Double price;
    private Integer sales;
    private Integer stock;
    private String imgPath ="static/img/default.jpg";
    public Book(Integer id, String title, String author, Double price, Integer sales, Integer stock, String
imgPath) {
        super();
        this.id = id;
    }
}

```



```
        this.title = title;
        this.author = author;
        this.price = price;
        this.sales = sales;
        this.stock = stock;
        if(imgPath != null){
            this.imgPath = imgPath;
        }
    }
    public Book() {
        super();
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public Double getPrice() {
        return price;
    }
    public void setPrice(Double price) {
        this.price = price;
    }
    public Integer getSales() {
        return sales;
    }
    public void setSales(Integer sales) {
        this.sales = sales;
    }
    public Integer getStock() {
        return stock;
    }
    public void setStock(Integer stock) {
        this.stock = stock;
    }
    public String getImgPath() {
```

```

        return imgPath;
    }

    public void setImgPath(String imgPath) {
        this.imgPath = imgPath;
    }

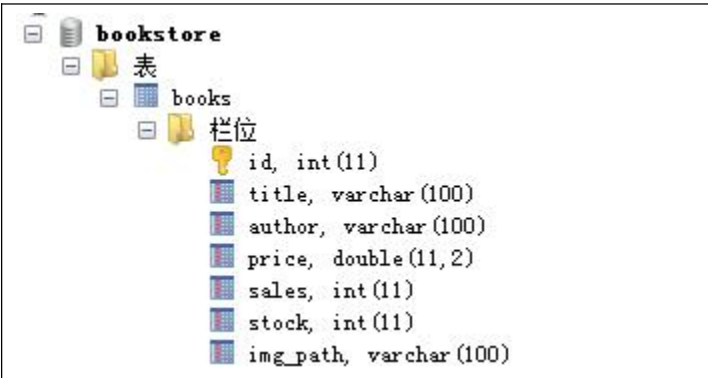
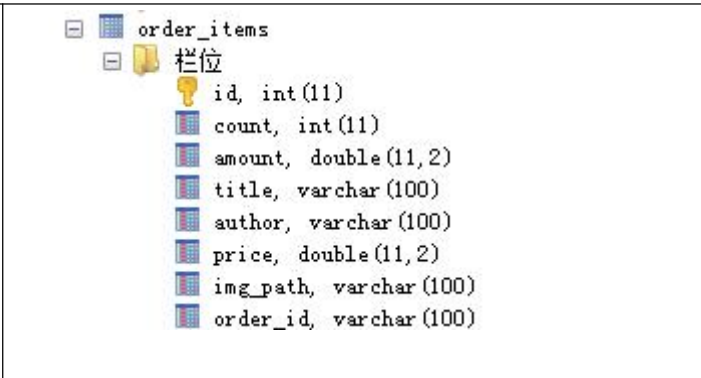
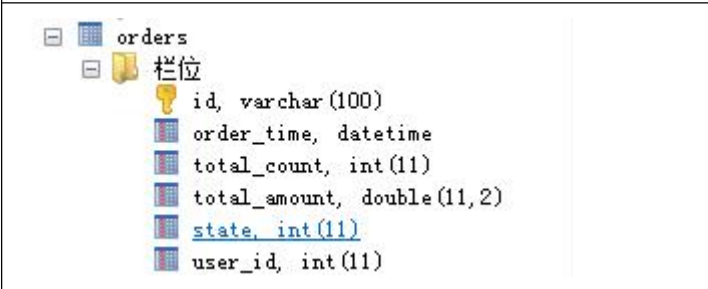
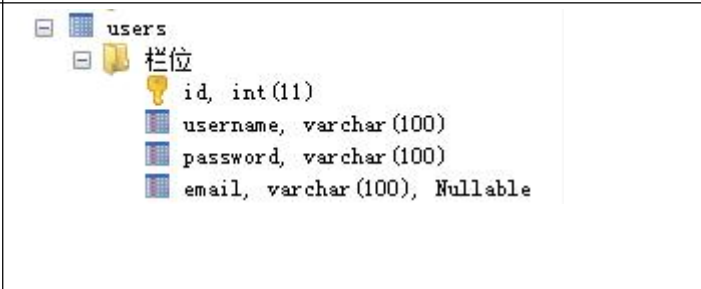
    @Override
    public String toString() {
        return "Book [id=" + id + ", title=" + title + ", author=" + author + ", price=" + price + ", sales="
+ sales
        + ", stock=" + stock + ", imgPath=" + imgPath + "]";
    }
}

```

七、DAO 和增删改查通用方法

DAO: Data Access Object 访问数据信息的类和接口，包括了对数据的 CRUD（Create、Retrival、Update、Delete），而不包含任何业务相关的信息

作用：为了实现功能的模块化，更有利于代码的维护和升级。

 <pre> bookstore ├── 表 │ └── books │ ├── 栏位 │ │ ├── id, int(11) │ │ ├── title, varchar(100) │ │ ├── author, varchar(100) │ │ ├── price, double(11,2) │ │ ├── sales, int(11) │ │ ├── stock, int(11) │ │ └── img_path, varchar(100) </pre>	 <pre> order_items ├── 栏位 │ ├── id, int(11) │ ├── count, int(11) │ ├── amount, double(11,2) │ ├── title, varchar(100) │ ├── author, varchar(100) │ ├── price, double(11,2) │ ├── img_path, varchar(100) │ └── order_id, varchar(100) </pre>
 <pre> orders ├── 栏位 │ ├── id, varchar(100) │ ├── order_time, datetime │ ├── total_count, int(11) │ ├── total_amount, double(11,2) │ ├── state, int(11) │ └── user_id, int(11) </pre>	 <pre> users ├── 栏位 │ ├── id, int(11) │ ├── username, varchar(100) │ ├── password, varchar(100) │ └── email, varchar(100), Nullable </pre>

1、DAO 接口

```

package com.atguigu.dao;

import com.atguigu.bean.User;

```

```
public interface UserDao {  
    /**  
     * 根据用户名和密码获取数据库中的记录  
     *  
     * @param user  
     * @return User: 用户名和密码正确 null: 用户名或密码不正确  
     */  
    public User getUser(User user);  
  
    /**  
     * 根据用户名获取数据库中的记录  
     *  
     * @param user  
     * @return true: 用户名已存在, false: 用户名可用  
     */  
    public boolean checkUserName(User user);  
  
    /**  
     * 将用户保存到数据库  
     *  
     * @param user  
     */  
    public void saveUser(User user);  
}
```

```
package com.atguigu.dao;  
  
import java.util.List;  
  
import com.atguigu.bean.Book;  
  
public interface BookDAO {  
    /**  
     * 获取所有图书的方法  
     *  
     * @return  
     */  
    public List<Book> getBooks();  
  
    /**  
     * 添加图书的方法  
     *  
     * @param book  
     */  
    public void addBook(Book book);  
  
    /**  
     * 根据图书的 id 删除图书的方法
```

```

*
* @param bookId
*/
public void deleteBookById(String bookId);

/**
 * 根据图书的 id 获取图书信息
 *
 * @param bookId
 * @return
 */
public Book getBookById(String bookId);

/**
 * 更新图书信息的方法
 *
 * @param book
 */
public void updateBook(Book book);

/**
 * 批量更新图书的库存和销量
 *
 * @param params
 */
public void batchUpdateSalesAndStock(Object[][] params);
}

```

```

package com.atguigu.dao;

import java.util.List;

import com.atguigu.bean.Order;

public interface OrderDAO {
    /**
     * 保存订单的方法
     *
     * @param order
     */
    public void saveOrder(Order order);

    /**
     * 获取所用订单的方法
     *
     * @return
     */
    public List<Order> getOrders();
}

```

```

/**
 * 获取我的订单的方法
 *
 * @param userId
 * @return
 */
public List<Order> getMyOrders(int userId);

/**
 * 更新订单的状态的方法，例如已发货、确认收货等
 *
 * @param orderId
 * @param state
 */
public void updateOrderState(String orderId, int state);
}

```

```

package com.atguigu.dao;

import java.util.List;

import com.atguigu.bean.OrderItem;

public interface OrderItemDAO {
    /**
     * 根据订单号获取对应的订单项
     *
     * @param orderId
     * @return
     */
    public List<OrderItem> getOrderItemsByOrderId(String orderId);

    /**
     * 批量插入订单项的方法
     *
     * @param params
     */
    public void batchInsertOrderItems(Object[][] params);
}

```

2、BasicDAOImpl

```

package com.atguigu.dao.impl;

import java.sql.Connection;
import java.sql.SQLException;

```

```

import java.util.List;

import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;
import org.apache.commons.dbutils.handlers.BeanListHandler;
import org.apache.commons.dbutils.handlers.ScalarHandler;

import com.atguigu.util.JDBCTools;

/**
 * 定义一个用来被继承的对数据库进行基本操作的 Dao
 */
public class BasicDAOImpl {
    private QueryRunner queryRunner = new QueryRunner();

    /**
     * 通用的增删改操作
     *
     * @param sql
     * @param params
     * @return
     */
    public int update(String sql, Object... params) {
        // 获取连接
        Connection connection = JDBCTools.getConnection();
        int count = 0;
        try {
            count = queryRunner.update(connection, sql, params);
        } catch (SQLException e) {
            //将编译时异常转换为运行时异常向上抛
            throw new RuntimeException(e);
        }
        return count;
    }

    /**
     * 获取一个对象
     *
     * @param sql
     * @param params
     * @return
     */
    public <T> T getBean(Class<T> type, String sql, Object... params) {
        // 获取连接
        Connection connection = JDBCTools.getConnection();
        T t = null;
        try {
            t = queryRunner.query(connection, sql, new BeanHandler<T>(type), params);
        } catch (SQLException e) {

```

```

        //将编译时异常转换为运行时异常向上抛
        throw new RuntimeException(e);
    }
    return t;
}

/**
 * 获取所有对象
 *
 * @param sql
 * @param params
 * @return
 */
public <T> List<T> getBeanList(Class<T> type,String sql, Object... params) {
    // 获取连接
    Connection connection = JDBCTools.getConnection();
    List<T> list = null;
    try {
        list = queryRunner.query(connection, sql, new BeanListHandler<T>(type), params);
    } catch (SQLException e) {
        //将编译时异常转换为运行时异常向上抛
        throw new RuntimeException(e);
    }
    return list;
}

/**
 * 获取一个单一值的方法，专门用来执行像 select count(*)... 这样的 sql 语句
 *
 * @param sql
 * @param params
 * @return
 */
public Object getSingleValue(String sql, Object... params) {
    // 获取连接
    Connection connection = JDBCTools.getConnection();
    Object value = null;
    try {
        value = queryRunner.query(connection, sql, new ScalarHandler(), params);
    } catch (SQLException e) {
        //将编译时异常转换为运行时异常向上抛
        throw new RuntimeException(e);
    }
    return value;
}

/**
 * 进行批处理的方法
 * 关于二维数组 Object[][] params
 *      二维数组的第一维是 sql 语句要执行的次数

```

```

*      二维数组的第二维就是每条 sql 语句中要填充的占位符
*
* @param sql
* @param params
*/
public void batchUpdate(String sql , Object[][] params){
    //获取连接
    Connection connection = JDBCTools.getConnection();
    try {
        queryRunner.batch(connection ,sql, params);
    } catch (SQLException e) {
        //将编译时异常转换为运行时异常向上抛
        throw new RuntimeException(e);
    }
}
}

```

3、DAO 实现类

```

package com.atguigu.dao.impl;

import com.atguigu.bean.User;
import com.atguigu.dao.UserDAO;

public class UserDAOImpl extends BasicDAOImpl implements UserDAO{

    @Override
    public User getUser(User user) {
        // 写查询数据库的 sql 语句
        String sql = "select id , username , password , email from users where username = ? and password = ?";
        // 调用 BaseDao 中的 getBean 方法
        User bean = getBean(User.class, sql, user.getUsername(), user.getPassword());
        return bean;
    }

    @Override
    public boolean checkUserName(User user) {
        // 写查询数据库的 sql 语句
        String sql = "select id , username , password , email from users where username = ?";
        // 调用 BaseDao 中的 getBean 方法
        User bean = getBean(User.class, sql, user.getUsername());
        return bean!=null; //不为空, 说明已存在, 返回 true, 如果 bean 是空的, 没找到, bean!=null 返回 false, 说明不存在
    }

    @Override
    public void saveUser(User user) {

```



```
//写添加数据到数据库的 sql 语句
```

```
String sql = "insert into users(username,password,email) values(?,?,?)";
```

```
//调用 BaseDao 中通用的增删改的方法
```

```
update(sql, user.getUsername(),user.getPassword(),user.getEmail());
```

```
}
```

```
}
```