

# 尚硅谷之 JDBC

官网: [www.atguigu.com](http://www.atguigu.com)

版本: V1.1

## 第 1 章概述

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。

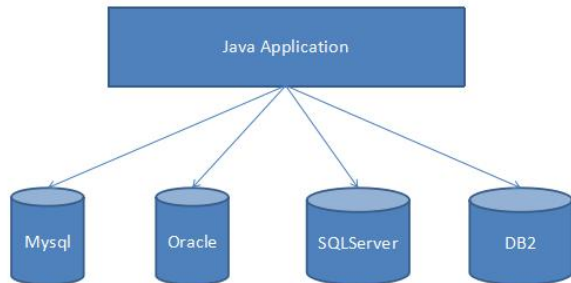
### 1、什么是 JDBC

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

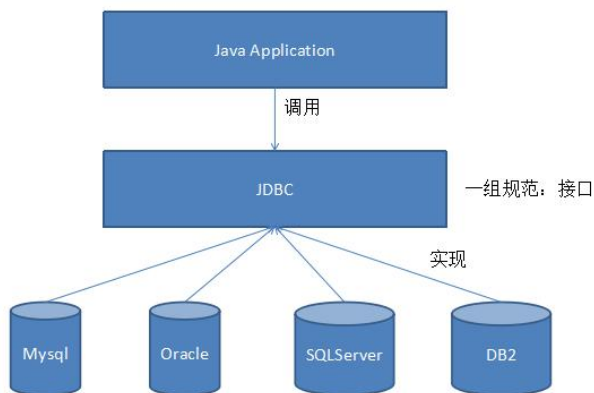
JDBC 为访问不同的数据库提供了一种**统一的途径**, 为开发者屏蔽了一些细节问题。

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。

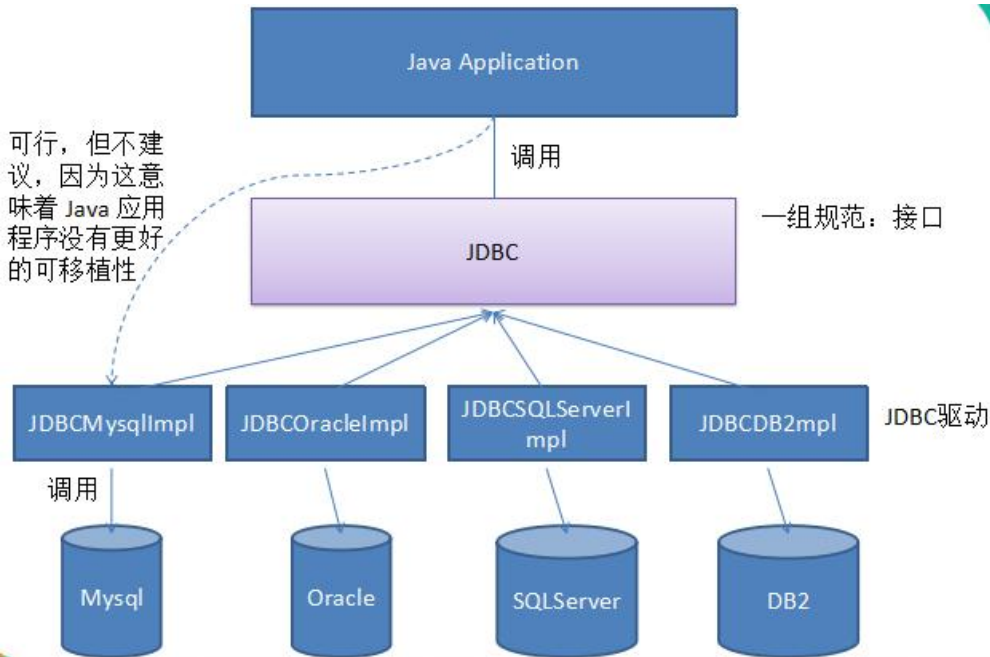
如果没有 JDBC, 那么 Java 程序访问数据库时是这样的:



改装:



实际上:

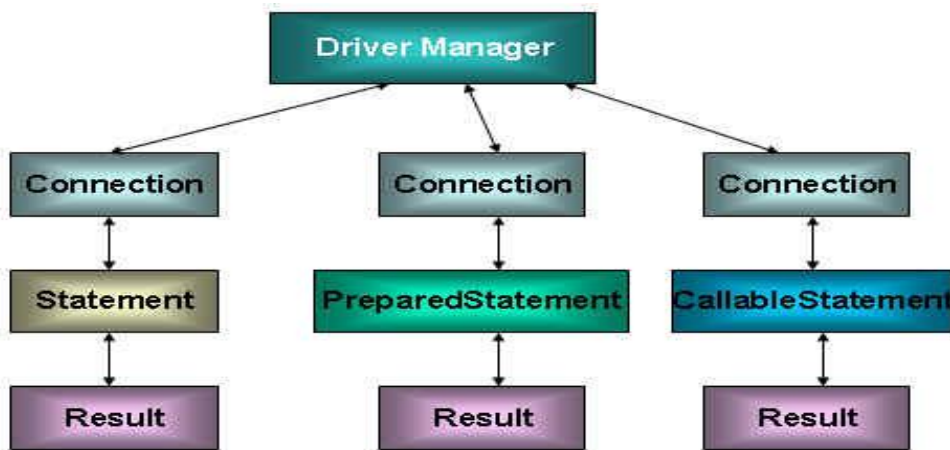


**结论：**

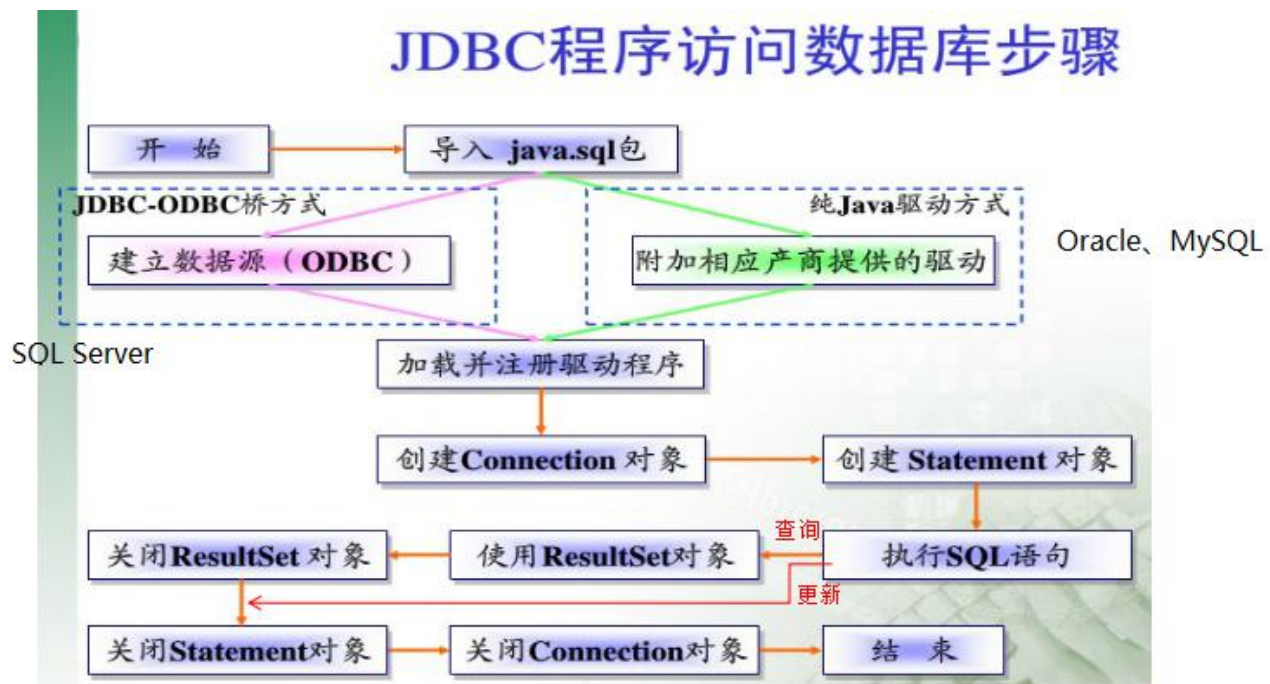
JDBC 是 SUN 公司提供一套用于数据库操作的接口 API，Java 程序员只需要面向这套接口编程即可。不同的数据库厂商，需要针对这套接口，提供不同实现。不同的实现的集合，即为不同数据库的驱动。

## 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



## 第 2 章 使用 JDBC API

### 1、引入 JDBC 驱动程序

#### 1.1 如何获取 JDBC 驱动程序

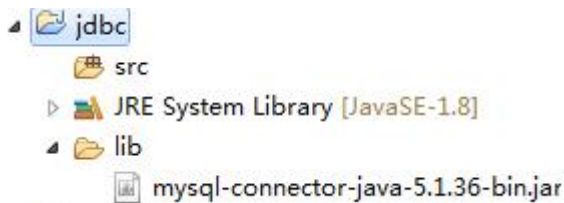
驱动程序由数据库提供商提供下载。

MySQL 的驱动下载地址：<http://dev.mysql.com/downloads/>

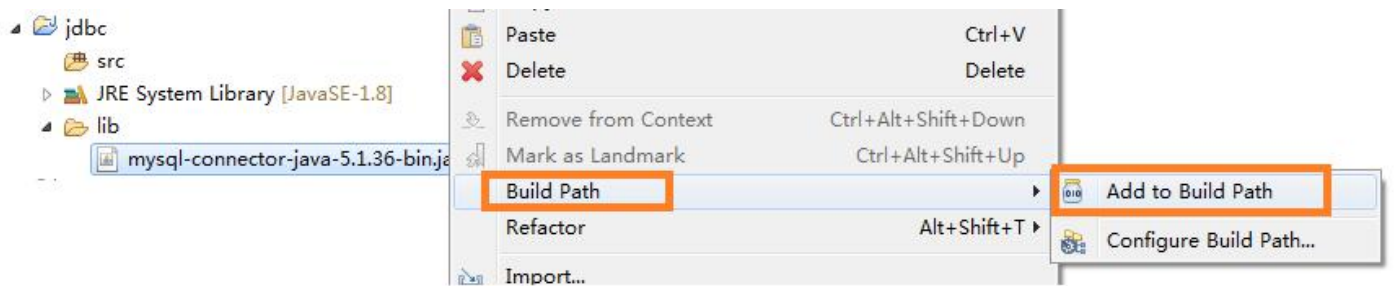
#### 1.2 如何在 Java Project 项目应用中添加数据库驱动 jar



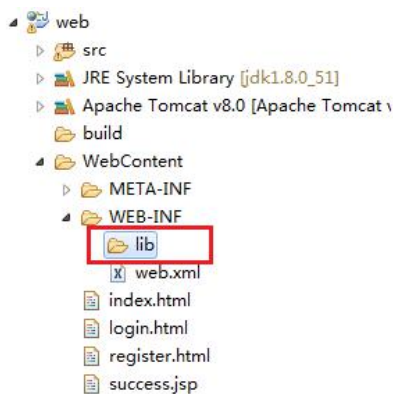
- (1) 把 `mysql-connector-java-5.1.36-bin.jar` 拷贝到项目中一个目录中



- (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）的话，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



## 2、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。

### 2.1 Driver 接口

Java.sql.Driver 接口是所有 JDBC 驱动程序需要实现的接口。这个接口是提供给数据库厂商使用的，不同数据库厂商提供不同的实现。

- MySQL: com.mysql.jdbc.Driver
- SQLServer: com.microsoft.sqlserver.jdbc.SQLServerDriver
- Oracle: oracle.jdbc.driver.OracleDriver

### 2.2 编写加载与注册驱动的代码

#### 1) 直接创建 Driver 接口的实现类对象

例如：

```
Driver driver = new com.mysql.jdbc.Driver();  
//因为与 mysql 驱动包中的类发生直接依赖，所以这样可移植性不够好
```

#### 2) DriverManager 类的 registerDriver()

在实际开发中，程序中不直接去访问实现了 Driver 接口的类，而是由驱动程序管理器类 (java.sql.DriverManager) 去调用这些 Driver 实现。

DriverManager 类是驱动程序管理器类，负责管理驱动程序。

通常不用显式调用 DriverManager 类的 registerDriver() 方法来注册驱动程序类的实例。

```
//DriverManager.registerDriver() 方式注册驱动，还是依赖  
DriverManager.registerDriver(new com.mysql.jdbc.Driver());
```

### 3) Class.forName()或 ClassLoader 对象.loadClass()

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 驱动的类型名

//1、通过反射，加载与注册驱动类，**解耦合（不直接依赖）**

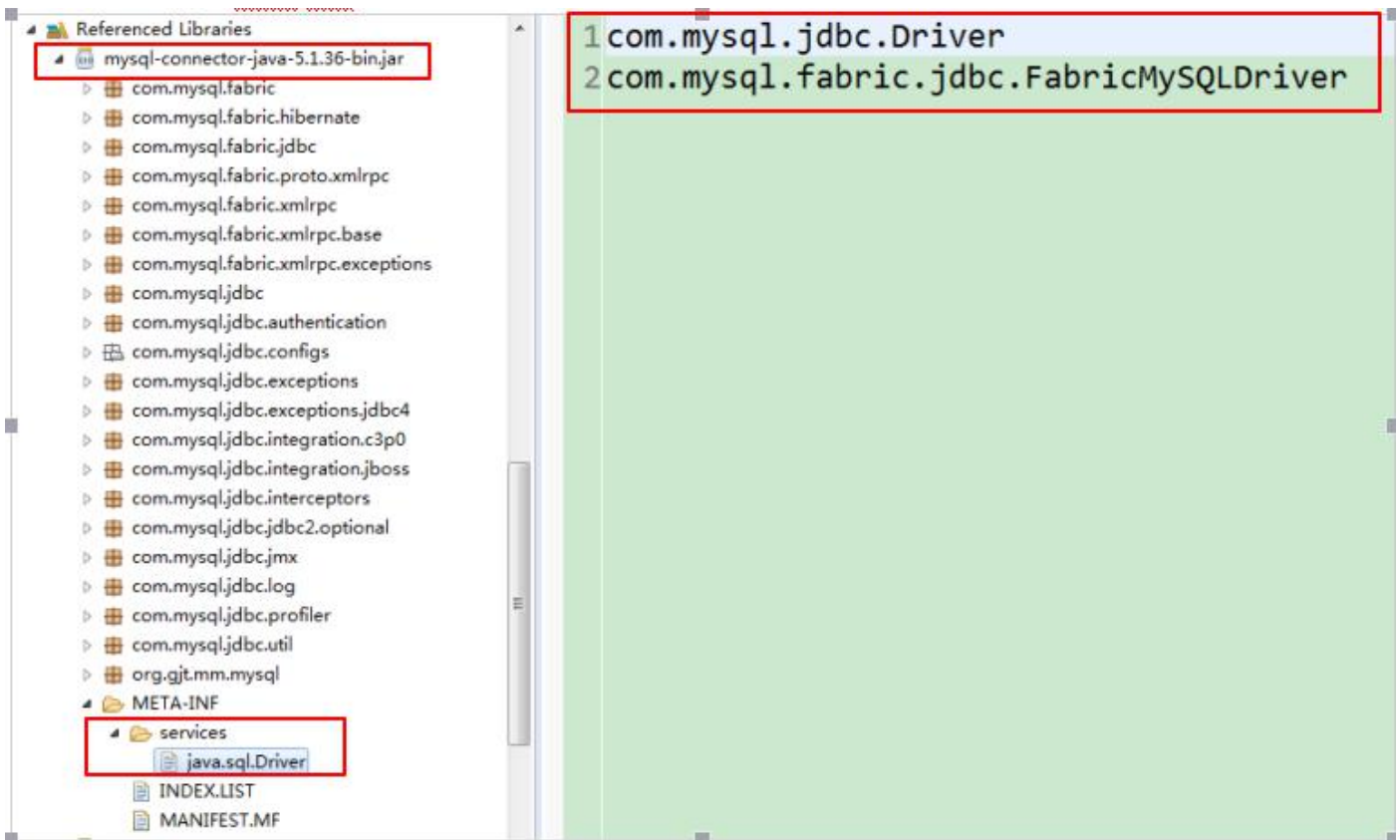
```
Class.forName("com.mysql.jdbc.Driver");
```

//2、通过类加载器加载驱动类，**解耦合（不直接依赖）**

```
ClassLoader.getSystemClassLoader().loadClass("com.mysql.jdbc.Driver");
```

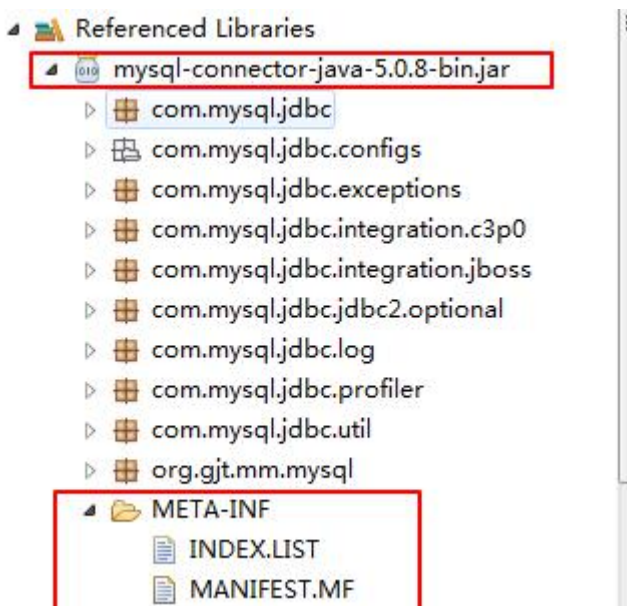
### 4) 服务提供者框架（例如：JDBC 的驱动程序）自动注册（有版本要求）

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



### 3、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)`

### 3.1 JDBC URL

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`
  - `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");
```

//硬编码

### 3.2 jdbc.properties

在 **src** 下建立文件 `jdbc.properties`，当然也可以是其他名字，只不过这里为了见名知意取这个名字通常至少应该包括 `"user"` 和 `"password"` 属性

```
#key=value
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
```

```
user=root
password=root

public static void main(String[] args) throws Exception {
    //预先加载配置文件 jdbc.properties, 把配置信息封装到 Properties 对象中
    Properties pro = new Properties();

    //ClassLoader 只能加载类路径下的资源文件
    //如果不是类路径下, 只能使用 FileInputStream
    pro.load(ClassLoader.getSystemClassLoader().getResourceAsStream("jdbc.properties"));

    //1、加载与注册驱动
    Class.forName(pro.getProperty("driver"));

    //2、获取数据库连接
    // Connection conn = DriverManager.getConnection(pro.getProperty("url"), pro);
    Connection conn = DriverManager.getConnection(pro.getProperty("url"),
    pro.getProperty("user"), pro.getProperty("password"));
    System.out.println(conn);
}
```

//解决了硬编码的问题

## 4、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

### 4.1 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

### 4.2 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 类型与 SQL 类型的对应关系

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

## 5、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

未关闭后果：

```
@Test
public void testConnection4()throws Exception{
    Properties pro = new Properties();
    pro.load(ClassLoader.getSystemClassLoader().getResourceAsStream("jdbc.properties"));
    String url = pro.getProperty("url");

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

## 6、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{
        String sql = "INSERT INTO dept(dname,description) VALUES('财务部','负责发钱工作)";

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

        Statement st = conn.createStatement();

        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{
    String sql = "UPDATE dept SET description = '负责发工资、社保、公积金工作' WHERE dname ='财务部'";

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

    Statement st = conn.createStatement();

    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{
    String sql = "DELETE FROM dept WHERE did =2";

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

    Statement st = conn.createStatement();

    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{
        String sql = "SELECT * FROM dept";

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

        Statement st = conn.createStatement();

        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{
        String sql = "SELECT did,dname FROM dept";

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

        Statement st = conn.createStatement();

        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();
}
}
```

## 第 3 章 JDBC 工具类

### 3.1 使用 JDBC API 操作数据库的基本步骤

分析使用 JDBC API 操作数据库，进行 CRUD 基本步骤都相似。

- 1、加载和注册驱动
  - 2、获取数据库连接
  - 3、准备操作执行 SQL 的 Statement 对象
  - 4、执行 SQL
    - (1) 调用 Statement 对象的 executeUpdate(String sql) 执行 SQL 语句进行插入、修改、删除操作
    - (2) 调用 Statement 对象的 executeQuery(String sql) 执行 SQL 语句进行查询操作
  - 5、处理执行结果
    - (1) CUD 操作，根据返回的 int 值判断结果
    - (2) 查询操作，根据返回 ResultSet 结果集，获取查询数据
  - 6、释放资源
- 总结：
- (1) 加载和注册驱动，整个项目做一次即可
  - (2) 获取数据库连接可以封装到一个方法中
  - (3) 释放资源可以封装到一个方法中

### 3.2 编写工具类 JDBCUtils

```
package com.atguigu.utils;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

/*
 * 工具类：
 */
```

```
* 1、注册驱动：只要运行一次
* 2、获取连接
* 3、关闭资源
*/
```

```
public class JDBCUtils {
    private static String drivername;
    private static String url;
    private static String user;
    private static String password;
    private static Properties pro = new Properties();
    static{
        try {
            //加载，读取 jdbc.properties 配置的信息
            //pro.load 的作用是把 jdbc.properties 文件中配置的信息，一一 put 到 pro 这个 map 中
            pro.load(ClassLoader.getSystemClassLoader().getResourceAsStream("jdbc.properties"));

//            drivername = pro.getProperty("key")
            drivername = pro.getProperty("drivername");
            url = pro.getProperty("url");
            user = pro.getProperty("user");
            password = pro.getProperty("password");

            //注册驱动，加载驱动
            Class.forName(drivername);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection()throws SQLException{
        Connection conn = DriverManager.getConnection(url, user, password);
        return conn;
    }

    public static void closeQuietly(Connection conn){
        try {
            if(conn!=null){
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
public static void closeQuietly(Statement st){
    try {
        if(st!=null){
            st.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void closeQuietly(ResultSet rs){
    try {
        if(rs!=null){
            rs.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void closeQuietly(Statement st,Connection conn){
    closeQuietly(st);
    closeQuietly(conn);
}

public static void closeQuietly(ResultSet rs,Statement st,Connection conn){
    closeQuietly(rs);
    closeQuietly(st);
    closeQuietly(conn);
}
}
```

## 第 4 章 PreparedStatement

### 1、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

## 2、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)	
TinyBlob	最大 255	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
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.jdbc;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class TestBlob {

    public static void main(String[] args) throws Exception{
        //加载 jdbc.properties 资源配置文件
        Properties pro = new Properties();
    }
}
    
```



```
pro.load(ClassLoader.getResourceAsStream("jdbc.properties"));

//1、加载与注册驱动
Class.forName(pro.getProperty("driver"));

//2、获取数据库连接
Connection conn = DriverManager.getConnection(pro.getProperty("url"), pro);

//3、访问数据库
//(1)准备带参数(?)的 SQL
//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
String sql = "insert into user(username,head_picture) value(?,?)";

//(2)通过调用 Connection 对象的 preparedStatement(String sql) 方法获取 PreparedStatement 对象
PreparedStatement pst = conn.prepareStatement(sql);

//(3)调用 PreparedStatement 对象的 setXxx(int parameterIndex,XX value) 方法来设置这些参数
pst.setString(1, "lily");
pst.setBlob(2, new FileInputStream("head/girl.jpg"));

//(4)调用 PreparedStatement 的 executeUpdate()执行 SQL 语句进行插入
//注意此处不能在传 sql，否则?就白设置了
int len = pst.executeUpdate();

//(5)处理结果
if (len > 0) {
    System.out.println("添加成功");
} else {
    System.out.println("添加失败");
}

//4、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。
  - 在 statement 语句中, 即使是相同操作但因为数据内容不一样, 所以整个语句本身不能匹配, 没有缓存语句的意义. 事实是没有数据库会对普通语句编译后的执行代码缓存. 这样每执行一次都要对传入的语句编译一次。
  - (语法检查, 语义检查, 翻译成二进制命令, 缓存)

### 4、示例代码

#### (1) 使用 Statement

```
package com.atguigu.statement;

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

import org.junit.Test;

import com.atguigu.utils.JDBCUtils;

/*
 * 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、获取连接
        Connection conn = JDBCUtils.getConnection();

        //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
        JDBCUtils.closeQuietly(st, conn);
    }

    @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、获取连接
        Connection conn = JDBCUtils.getConnection();

        //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
JDBCUtils.closeQuietly(rs, st, conn);
}

@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) 使用 PreparedStatement

```
package com.atguigu.preparedstatement;

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

import org.junit.Test;

import com.atguigu.utils.JDBCUtils;

/*
 * 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("请输入性别: ");
        String gender = 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();

        String sql = "INSERT INTO emp VALUES(NULL,?,?,?)"; // 参数, 占位符, 通配符, 表示这个地方需要设置值

        // 2、获取连接
        Connection conn = JDBCUtils.getConnection();

        // 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, gender);
        pst.setInt(3, mid);
        pst.setInt(4, did);

        // 5、执行 sql
        int len = pst.executeUpdate();
    }
}
```

```
// 6、释放资源
JDBCUtils.closeQuietly(pst, conn);
}

@Test
public void select() throws Exception {
    // 3、写 sql
    Scanner input = new Scanner(System.in);
    System.out.println("请输入姓名: ");
    String name = input.nextLine();

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

    // 1、注册驱动，注册过了
    // 2、获取连接
    Connection conn = JDBCUtils.getConnection();

    // 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
    JDBCUtils.closeQuietly(rs, pst, conn);
}

@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 password = input.nextLine();

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

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

// 1、注册驱动, 注册过了
// 2、获取连接
Connection conn = JDBCUtils.getConnection();

// 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, password);
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
JDBCUtils.closeQuietly(pst, conn);
}
}
```

## 第 5 章 JDBC 的其他操作

### 1、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.other;

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

import org.junit.Test;

import com.atguigu.utils.JDBCUtils;

public class TestGetGenericKey {
    @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("请输入性别: ");
        String gender = 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();

        String sql = "INSERT INTO emp VALUES(NULL,?,?,?)"; // 参数, 占位符, 通配符, 表示这个地方需要设置值

        // 2、获取连接
        Connection conn = JDBCUtils.getConnection();

        // 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, gender);
pst.setInt(3, mid);
pst.setInt(4, did);

// 5、执行 sql
int len = pst.executeUpdate();

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

// 6、释放资源
JDBCUtils.closeQuietly(pst, conn);
}
}
```

## 2、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）**

### 2.1 Statement

`void addBatch(String sql)`：添加需要批量处理的 SQL 语句

`int[] executeBatch()`：执行批量处理语句；

### 2.2 PreparedStatement

`void addBatch()`将一组参数添加到此 `PreparedStatement` 对象的批处理命令中

`int[] executeBatch()`：执行批量处理语句；

### 2.3 关于效率测试

测试：插入 100000 条记录

- （1）Statement 不使用批处理

- (2) PreparedStatement 不使用批处理
  - (3) Statement 使用批处理
  - (4) PreparedStatement 使用批处理 (效率最高)
- (4) > (3) > (1) > (2)

## 2.4 示例代码

```
package com.atguigu.other;

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

import org.junit.Test;

import com.atguigu.utils.JDBCUtils;

public class TestBatch {
    /*
     * 没有使用批处理
     */
    @Test
    public void testNoBatch() throws Exception {
        long start = System.currentTimeMillis();

        //批处理
        //添加 500 件商品
        String sql = "INSERT INTO t_goods (pname,price) VALUES(?,?)";

        Connection conn = JDBCUtils.getConnection();
        PreparedStatement pst = conn.prepareStatement(sql);

        for (int i = 1; i <= 100; i++) {
            String pname = "商品" + i;
            double price = i;

            pst.setString(1, pname);
            pst.setDouble(2, price);

            int len = pst.executeUpdate();
            System.out.println("第" + i + "条添加: " + (len>0?"成功":"失败"));
        }

        long end = System.currentTimeMillis();
        System.out.println("耗时: " + (end-start));
    }
}
```

```
@Test
public void testBatch()throws Exception {
    long start = System.currentTimeMillis();

    String sql = "INSERT INTO t_goods (pname,price) VALUES(?,?)";

    Connection conn = JDBCUtils.getConnection();
    PreparedStatement pst = conn.prepareStatement(sql);
    for (int i = 1; i <= 100; i++) {
        String pname = "商品" + i;
        double price = i;
        pst.setObject(1, pname);
        pst.setObject(2, price);

        pst.addBatch();//添加到批处理中
    }

    int[] executeBatch = pst.executeBatch();//一批命名同时执行
    for (int i = 0; i < executeBatch.length; i++) {
        System.out.println("第" + i + "条添加: " + (executeBatch[i]>0?"成功":"失败"));
    }

    JDBCUtils.closeQuietly(pst, conn);

    long end = System.currentTimeMillis();
    System.out.println("耗时: " + (end-start));
}
}
```

### 3、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**

JDBC 还可以通过 Connection 对象的：

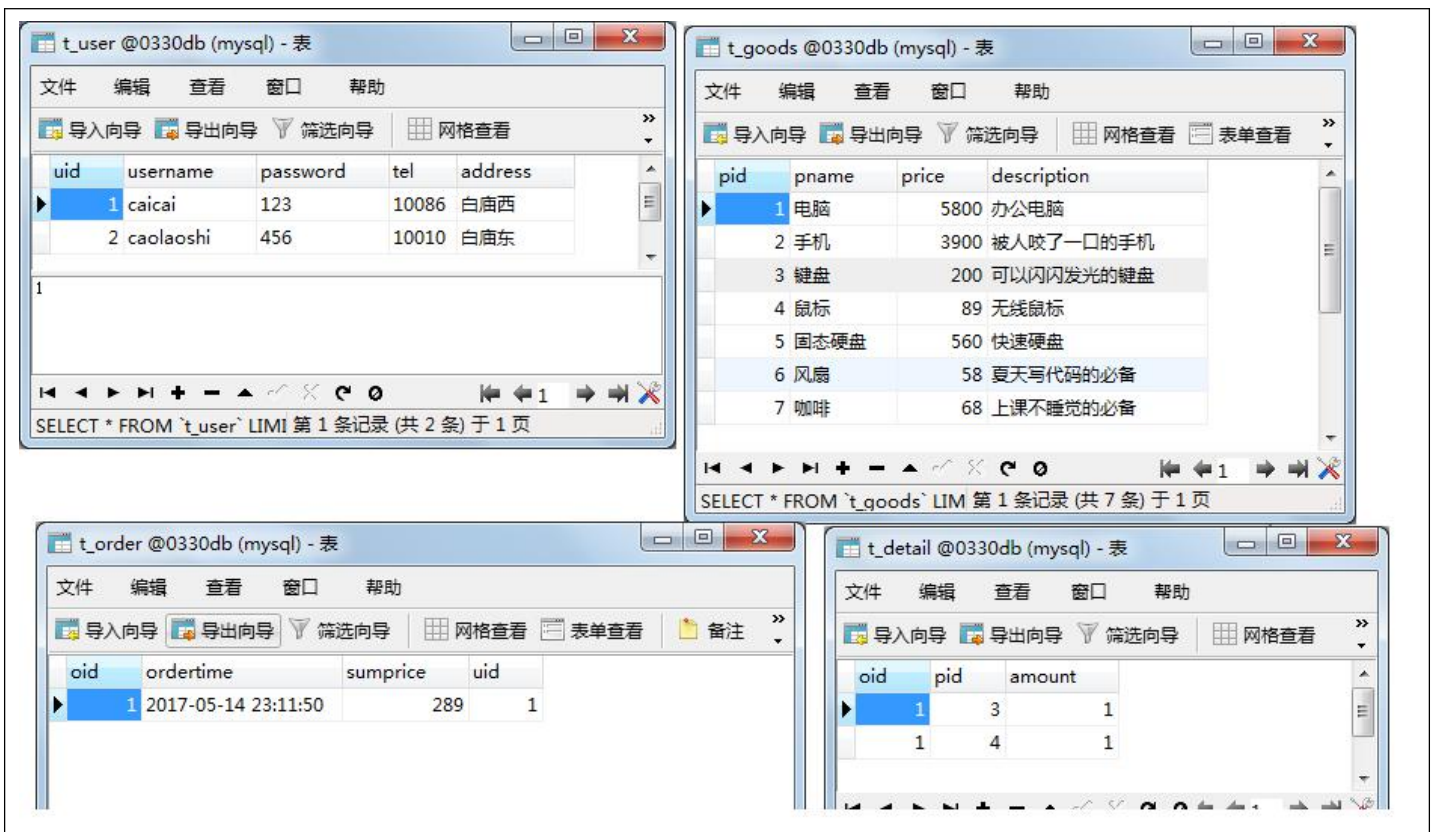
- `int getTransactionIsolation()`：获取此 Connection 对象的当前事务隔离级别
- `void setTransactionIsolation(int level)`：试图将此 Connection 对象的事务隔离级别更改为给定的级别。可能的事

事务隔离级别是 Connection 接口中定义的常量。

level- 以下 Connection 常量之一：

- Connection.TRANSACTION\_READ\_UNCOMMITTED (=1)
- Connection.TRANSACTION\_READ\_COMMITTED (=2)、
- Connection.TRANSACTION\_REPEATABLE\_READ (=4)
- Connection.TRANSACTION\_SERIALIZABLE (=8)。

(注意，不能使用 Connection.TRANSACTION\_NONE (=0)，因为它指定了不受支持的事务。)



```
package com.atguigu.other;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
import org.junit.Test;
```

```
import com.atguigu.utils.DBUtils;
```

```
public class TestTransaction {
```

```
    @Test
```

```
    public void test() {
```

```
int uid = 2;//谁买的
int[] pids = {1,2,3};//购买的商品的编号
int[] amount = {1,1,1};//分别每一个商品的数量
double[] price = {45,34,2.5};//购买时，每一件商品的价格

//如何把这些订单的数据保存到数据库中
//分别在订单表 t_order 添加一条记录：订单编号，订单时间，订单总价，用户编号
//在订单明细表 t_detail 添加几条记录：订单编号，商品编号，数量（多行），这里 3 行

//这些数据要么同时保存成功，要么同时失败（撤销），那么表示他们要组成一个事务

Connection conn = null;
try {

    //手动提交事务
    //1、获取连接对象(注册驱动省略，因为在 DBUtils 注册过了)
    conn = JDBCUtils.getConnection();
    //2、设置手动提交
    conn.setAutoCommit(false);

    String orderInsert = "INSERT INTO t_order (sumprice,uid) VALUES(?,?)";
    double sumprice = 0;
    for (int i = 0; i < price.length; i++) {
        sumprice += price[i]*amount[i];
    }
    //3、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(orderInsert,Statement.RETURN_GENERATED_KEYS);

    pst.setObject(1, sumprice);
    pst.setObject(2, uid);
    pst.executeUpdate();

    ResultSet rs = pst.getGeneratedKeys();
    if(rs.next()){
        Object oid = rs.getObject(1);

        String detailInsert = "INSERT INTO t_details (oid,pid,amount) VALUES(?,?,?)";
        PreparedStatement pst2 = conn.prepareStatement(detailInsert);
        for (int i = 0; i < pids.length; i++) {
            pst2.setObject(1, oid);
            pst2.setObject(2, pids[i]);
            pst2.setObject(3, amount[i]);
```

```
        pst2.addBatch();
    }

    pst2.executeBatch();
}

//4、提交事务
conn.commit();
System.out.println("添加成功");
}catch (Exception e) {
    e.printStackTrace();
    try {
        //如果发生异常，应该回滚
        if(conn!=null){
            System.out.println("添加失败");
            conn.rollback();
        }
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
}finally{
    try {
        //释放资源
        if(conn!=null){
            conn.setAutoCommit(true);
            conn.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}
```

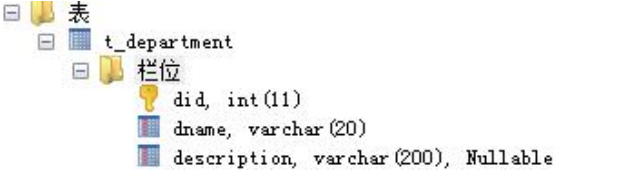
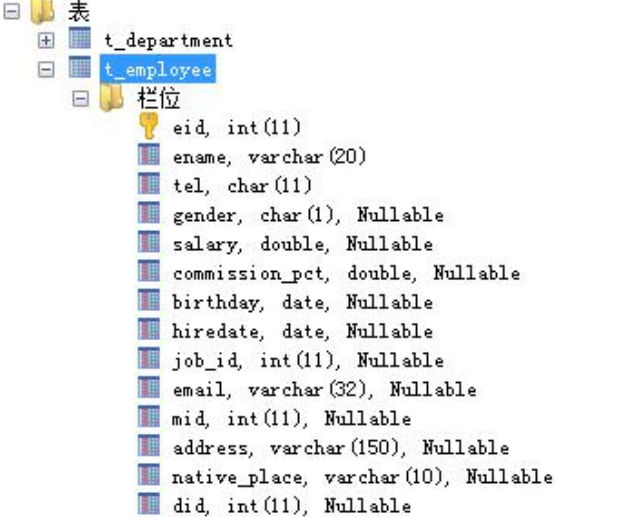
## 第 6 章 DAO 和增删改查通用方法

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。

# 1、练习

## 1.1 表和 JavaBean

 <pre>             表             └─ t_department                 └─ 栏目                     ├── did, int(11)                     ├── dname, varchar(20)                     └── description, varchar(200), Nullable                     </pre>	<pre>             public class Department {                 private Integer id;                 private String name;                 private String description;                 ....             }             </pre>
 <pre>             表             ├── t_department             └─ t_employee                 └─ 栏目                     ├── 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>

## 1.2 DAO 接口

```

package com.atguigu.dao;

import java.util.List;

import com.atguigu.bean.Department;

public interface DepartmentDAO {
    void addDepartment(Department department)throws Exception;
    void updateDepartment(Department department)throws Exception;
    void deleteById(String did)throws Exception;
    Department getById(String did)throws Exception;
    List<Department> getAll()throws Exception;
}
    
```

```

package com.atguigu.dao;
    
```

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

import com.atguigu.bean.Employee;

public interface EmployeeDAO {
    void addEmployee(Employee emp)throws Exception;
    void updateEmployee(Employee emp)throws Exception;
    void deleteById(String eid)throws Exception;
    Employee getById(String eid)throws Exception;
    List<Employee> getAll()throws Exception;
    Long getCount()throws Exception;
    List<Employee> getAll(int page, int pageSize)throws Exception;
    Double getMaxSalary()throws Exception;
    Map<Integer,Double> getAvgSalaryByDid()throws Exception;
}
```

## 1.3 DAO 实现类

### (1) 原生版

```
package com.atguigu.dao.impl.original;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

import com.atguigu.bean.Department;
import com.atguigu.dao.DepartmentDAO;
import com.atguigu.utils.JDBCUtils;

public class DepartmentDAOImpl implements DepartmentDAO{

    @Override
    public void addDepartment(Department department) throws Exception {
        Connection conn = JDBCUtils.getConnection();

        String sql = "INSERT INTO t_department(did,dname,description) VALUES(NULL,?,?)";
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setString(1, department.getName());
        pst.setString(2, department.getDescription());
        pst.executeUpdate();
    }
}
```



```
        JDBCUtils.closeQuietly(pst, conn);
    }

    @Override
    public void updateDepartment(Department department) throws Exception {
        Connection conn = JDBCUtils.getConnection();

        String sql = "UPDATE t_department SET dname = ?,description = ? WHERE did = ?";
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setString(1, department.getName());
        pst.setString(2, department.getDescription());
        pst.setInt(3, department.getId());
        pst.executeUpdate();

        JDBCUtils.closeQuietly(pst, conn);
    }

    @Override
    public void deleteById(String did) throws Exception {
        Connection conn = JDBCUtils.getConnection();

        String sql = "DELETE FROM t_department WHERE did = ?";
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setString(1, did);
        pst.executeUpdate();

        JDBCUtils.closeQuietly(pst, conn);
    }

    @Override
    public Department getById(String did) throws Exception {
        Connection conn = JDBCUtils.getConnection();

        String sql = "SELECT did,dname,description FROM t_department WHERE did = ?";
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setString(1, did);

        ResultSet rs = pst.executeQuery();
        Department dept = null;
        if(rs.next()){
            dept = new Department();
            dept.setId(rs.getInt("did"));
            dept.setName(rs.getString("dname"));
        }
    }
}
```

```
        dept.setDescription(rs.getString("description"));
    }

    JDBCUtils.closeQuietly(rs, pst, conn);

    return dept;
}

@Override
public List<Department> getAll() throws Exception {
    Connection conn = JDBCUtils.getConnection();

    String sql = "SELECT did,dname,description FROM t_department";
    PreparedStatement pst = conn.prepareStatement(sql);

    ResultSet rs = pst.executeQuery();
    ArrayList<Department> list = new ArrayList<Department>();
    while(rs.next()){
        Department dept = new Department();
        dept.setId(rs.getInt("did"));
        dept.setName(rs.getString("dname"));
        dept.setDescription(rs.getString("description"));
        list.add(dept);
    }

    JDBCUtils.closeQuietly(rs, pst, conn);

    return list;
}
}
```

## 1.4 抽取 BasicDAO

```
package com.atguigu.dao.impl;

/*
 * 这个类的作用是：对 DAOImpl 再次抽象，把共同的部分再次抽取
 */
import java.lang.reflect.Field;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.atguigu.utils.JDBCUtils;

//泛型类
public abstract class BasicDAOImpl<T> {
    private Class<T> type;

    @SuppressWarnings("all")
    protected BasicDAOImpl() {
        // 为什么要在构造器中写，因为子类继承 BasicDAOImpl 类一定会调用父类的构造器
        Class clazz = this.getClass();// this 代表的是正在创建的那个对象，即子类的对象

        // 获取 clazz 的带泛型父类信息
        Type superType = clazz.getGenericSuperclass();

        // Father<String>: 参数化的类型
        ParameterizedType p = (ParameterizedType) superType;

        // 获取泛型实参
        Type[] ts = p.getActualTypeArguments();

        // 因为当前类只有一个泛型形参，即子类中只有一个泛型实参
        type = (Class) ts[0];
    }

    protected int update(String sql, Object... params) throws SQLException {
        //1、获取连接
        Connection conn = JDBCUtils.getConnection();
        //2、执行更新数据库语句
        int len = executeUpdate(conn, sql, params);
        //3、关闭连接
        JDBCUtils.closeQuietly(conn);
        return len;
    }

    // 如果有需要在事务中完成更新，可以调用这个带 conn 的方法
    protected int update(Connection conn, String sql, Object... params) throws SQLException {
        //执行更新数据库语句
        int len = executeUpdate(conn, sql, params);
    }
}
```

```
        return len;
    }

    private int executeUpdate(Connection conn, String sql, Object... params) throws SQLException {
        //1、sql 预编译
        PreparedStatement pst = conn.prepareStatement(sql);

        //2、设置 sql 中的?
        if (params != null && params.length > 0) {
            // 数组的下标是从 0 开始，? 的编号是 1 开始
            for (int i = 0; i < params.length; i++) {
                pst.setObject(i + 1, params[i]);
            }
        }

        //3、执行 sql
        int len = pst.executeUpdate();

        //4、释放资源
        JDBCUtils.closeQuietly(pst);
        return len;
    }

    //通用的查询方法之一：查询一行，即一个对象
    /**
     * 执行查询操作的 SQL 语句，SQL 可以带参数(?)
     * @param sql String 执行查询操作的 SQL 语句
     * @param args Object... 对应的每个?设置的值，顺序要与?对应
     * @return T 封装了查询结果的实体
     * @throws Exception
     */
    protected T get(String sql, Object... params) throws Exception {
        //1、获取连接
        Connection conn = JDBCUtils.getConnection();

        //2、执行查询语句
        ResultSet rs = executeQuery(conn, sql, params);

        //3、处理查询结果
        // (1) 获取查询的结果集的元数据信息
        ResultSetMetaData rsmd = rs.getMetaData();

        // (2) 这是查询的结果集中，一共有几列
        int count = rsmd.getColumnCount();
    }
}
```

```
// (3) 创建实例对象
T t = type.newInstance();// 要求这个 Javabean 类型必须有无参构造

// (4) 遍历结果集
while (rs.next()) {
    /*
     * 问题? (1) sql 语句中查询了几列, 每一列是什么属性 (2) 怎么把这个值设置到 Javabean 的属性中
     */
    // (5) 循环每一行有几列
    for (int i = 0; i < count; i++) {
        // (6) 获取第几列的名称
        // String columnName = rsmd.getColumnName(i+1);
        // 如果 sql 中没有取别名, 那么就是列名, 如果有别名, 返回的是别名
        String fieldName = rsmd.getColumnLabel(i + 1);

        // (7) 获取该列的值
        // Object value = rs.getObject(columnName);
        Object value = rs.getObject(fieldName);

        // (8) 设置 obj 对象的某个属性中
        Field field = type.getDeclaredField(fieldName);// JavaBean 的属性名
        field.setAccessible(true);
        field.set(t, value);
    }
}

//4、释放资源
JDBCUtils.closeQuietly(rs);
JDBCUtils.closeQuietly(conn);

return t;
}

private static ResultSet executeQuery(Connection conn, String sql, Object... params) throws SQLException
{
    //1、sql 预编译
    PreparedStatement pst = conn.prepareStatement(sql);

    //2、设置?
    if (params != null && params.length > 0) {
        // 数组的下标是从 0 开始, ? 的编号是 1 开始
        for (int i = 0; i < params.length; i++) {
            pst.setObject(i + 1, params[i]);
        }
    }
}
```

```
//3、查询
ResultSet rs = pst.executeQuery();
return rs;
}

// 通用的查询方法之二：查询多行，即多个对象
// Class<T> clazz: 用来创建实例对象，获取对象的属性，并设置属性值
/**
 * 执行查询操作的 SQL 语句，SQL 可以带参数(?)
 * @param sql String 执行查询操作的 SQL 语句
 * @param args Object... 对应的每个?设置的值，顺序要与?对应
 * @return ArrayList<T> 封装了查询结果的集合
 * @throws Exception
 */
public ArrayList<T> getList(String sql, Object... args) throws Exception {
    // 1、获取连接
    Connection conn = JDBCUtils.getConnection();

    //2、执行查询 sql
    ResultSet rs = executeQuery(conn,sql, args);

    //3、获取结果集的元数据
    ResultSetMetaData metaData = rs.getMetaData();
    // 获取结果中总列数
    int count = metaData.getColumnCount();

    // 创建集合对象
    ArrayList<T> list = new ArrayList<T>();

    while (rs.next()) { // 遍历的行
        // 1、每一行是一个对象
        T obj = type.newInstance();

        // 2、每一行有很多列
        // for 的作用是为 obj 对象的每一个属性设置值
        for (int i = 0; i < count; i++) {
            // (1)每一列的名称
            String fieldName = metaData.getColumnLabel(i + 1); // 获取第几列的名称，如果有别名获取别名，如果没有别名获取列名

            // (2)每一列的值
            Object value = rs.getObject(i + 1); // 获取第几列的值
            // (3)获取属性对象
            Field field = type.getDeclaredField(fieldName);
            // (4)设置可见性
```

```
        field.setAccessible(true);
        // (5)设置属性值
        field.set(obj, value);
    }

    // 3、把 obj 对象放到集合中
    list.add(obj);
}

// 6、释放资源
JDBCUtils.closeQuietly(rs);
JDBCUtils.closeQuietly(conn);

// 7、返回结果
return list;
}

//通用的查询方法之三：查询单个值
//单值：select max(salary) from employee; 一行一列
//select count(*) from t_goods; 一共几件商品
public Object getValue(String sql,Object... args)throws Exception{
    //1、获取连接
    Connection conn = JDBCUtils.getConnection();

    //2、执行查询 sql
    ResultSet rs = executeQuery(conn, sql, args);

    Object value = null;
    if(rs.next()){//一行
        value = rs.getObject(1);//一列
    }

    //3、释放资源
    JDBCUtils.closeQuietly(rs);
    JDBCUtils.closeQuietly(conn);

    return value;
}

//通用的查询方法之四：查询多行多列，但每一行又不是一个 JavaBean
/*
 * SELECT did,AVG(salary),MAX(Salary) FROM t_employee GROUP BY did;
 * did  avg(salary)  max(salary)
 * 1    1990.90      8900
 * 2    4889         6899
```

```
*/
public List<Map<String,Object>> getListMap(String sql,Object... args)throws Exception{
    //1、获取连接
    Connection conn = JDBCUtils.getConnection();

    //2、执行 sql
    ResultSet rs = executeQuery(conn, sql, args);

    //获取结果集的元数据对象
    ResultSetMetaData metaData = rs.getMetaData();
    //一共有几列
    int count = metaData.getColumnCount();
    //创建 List
    ArrayList<Map<String,Object>> list = new ArrayList<Map<String,Object>>();

    while(rs.next()){
        //每一行是一个 Map 的对象
        HashMap<String,Object> map = new HashMap<String,Object>();

        //map 的 key 是列名
        for (int i = 0; i < count; i++) {
            //（1）获取列名或别名
            String columnName = metaData.getColumnLabel(i+1);
            //（2）获取对应的值
            Object value = rs.getObject(i+1);
            //（3）把这对值放到 map 中
            map.put(columnName, value);
        }

        //把 map 放到 List 中
        list.add(map);
    }

    //6、释放资源
    JDBCUtils.closeQuietly(rs);
    JDBCUtils.closeQuietly(conn);

    return list;
}

//通用的查询方法之四：查询一行多列，但一行又不是一个 JavaBean
public Map<String,Object> getMap(String sql,Object... args)throws Exception{
    List<Map<String, Object>> listMap = getListMap(sql,args);
    if(listMap.size(>0){
        return listMap.get(0);
    }
}
```



```
    }  
    return null;  
  }  
}
```

## 1.5 继承 BasicDAO 的后的 DAO 实现类

### DepartmentDAO 实现类

```
package com.atguigu.dao.impl.basic;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import com.atguigu.bean.Department;  
import com.atguigu.dao.DepartmentDAO;  
import com.atguigu.dao.impl.BasicDAOImpl;  
  
public class DepartmentDAOImplBasic extends BasicDAOImpl<Department> implements DepartmentDAO{  
  
    @Override  
    public void addDepartment(Department department) throws Exception {  
        String sql = "INSERT INTO t_department(did,dname,description) VALUES(NULL,?,?)";  
        update(sql, department.getName(),department.getDescription());  
    }  
  
    @Override  
    public void updateDepartment(Department department) throws Exception {  
        String sql = "UPDATE t_department SET dname = ?,description = ? WHERE did = ?";  
        update(sql, department.getName(),department.getDescription(),department.getId());  
    }  
  
    @Override  
    public void deleteById(String did) throws Exception {  
        String sql = "DELETE FROM t_department WHERE did = ?";  
        update(sql, did);  
    }  
  
    @Override  
    public Department getById(String did) throws Exception {  
        String sql = "SELECT did as id,dname as name,description FROM t_department WHERE did = ?";  
        Department department = get(sql, did);  
        return department;  
    }  
}
```

```
@Override
public List<Department> getAll() throws Exception {
    String sql = "SELECT did as id,dname as name,description FROM t_department";
    ArrayList<Department> list = getList(sql);
    return list;
}
}
```

## EmployeeDAO 实现类

```
package com.atguigu.dao.impl.basic;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import com.atguigu.bean.Employee;
import com.atguigu.dao.EmployeeDAO;
import com.atguigu.dao.impl.BasicDAOImpl;

public class EmployeeDAOImpl extends BasicDAOImpl<Employee> implements EmployeeDAO{

    @Override
    public void addEmployee(Employee emp) throws Exception {
        String sql = "INSERT INTO t_employee "
            +
            "(eid,ename,tel,gender,salary,commission_pct,birthday,hiredate,job_id,email,mid,address,native_place,did)
            "
            + "VALUES(NULL,?,?,?,?,?,?,?,?,?,?,?,?,?)";
        Object[] args = new Object[13];
        args[0] = emp.getEname();
        args[1] = emp.getTel();
        args[2] = emp.getGender();
        args[3] = emp.getSalary();
        args[4] = emp.getCommissionPct();
        args[5] = emp.getBirthday();
        args[6] = emp.getHiredate();
        args[7] = emp.getJobId();
        args[8] = emp.getEmail();
        args[9] = emp.getMid();
    }
}
```

```
        args[10] = emp.getAddress();
        args[11] = emp.getNativePlace();
        args[12] = emp.getDid();

        update(sql, args);
    }

    @Override
    public void updateEmployee(Employee emp) throws Exception {
        String sql = "UPDATE t_employee SET "
            +
            "ename=?,tel=?,gender=?,salary=?,commission_pct=?,birthday=?,hiredate=?,job_id=?,email=?,MID=?,address=?,
            native_place=?,did=?"
            + " WHERE eid = ?";
        Object[] args = new Object[14];
        args[0] = emp.getEname();
        args[1] = emp.getTel();
        args[2] = emp.getGender();
        args[3] = emp.getSalary();
        args[4] = emp.getCommissionPct();
        args[5] = emp.getBirthday();
        args[6] = emp.getHiredate();
        args[7] = emp.getJobId();
        args[8] = emp.getEmail();
        args[9] = emp.getMid();
        args[10] = emp.getAddress();
        args[11] = emp.getNativePlace();
        args[12] = emp.getDid();
        args[13] = emp.getEid();
        update(sql, args);
    }

    @Override
    public void deleteById(String eid) throws Exception {
        String sql = "DELETE FROM t_employee WHERE eid = ?";
        update(sql, eid);
    }

    @Override
    public Employee getById(String eid) throws Exception {
        String sql = "SELECT eid,ename,tel,gender,salary,commission_pct as
'commissionPct',birthday,hiredate,"
            + "job_id as 'jobId',email,mid,address,native_place as 'nativePlace' ,did FROM t_employee
WHERE eid = ?";
        Employee employee = get(sql, eid);
    }
}
```

```
        return employee;
    }

    @Override
    public List<Employee> getAll() throws Exception {
        String sql = "SELECT eid,ename,tel,gender,salary,commission_pct as
'commissionPct',birthday,hiredate,"
            + "job_id as 'jobId',email,mid,address,native_place as 'nativePlace' ,did FROM t_employee";

        ArrayList<Employee> list = getList(sql);
        return list;
    }

    @Override
    public Long getCount() throws Exception {
        String sql = "SELECT COUNT(*) FROM t_employee";
        Long value = (Long) getValue(sql);
        return value;
    }

    @Override
    public List<Employee> getAll(int page, int pageSize) throws Exception {
        String sql = "SELECT eid,ename,tel,gender,salary,commission_pct as
'commissionPct',birthday,hiredate,"
            + "job_id as 'jobId',email,mid,address,native_place as 'nativePlace' ,did FROM t_employee
LIMIT ?,?";

        ArrayList<Employee> list = getList( sql, (page-1)*pageSize, pageSize);
        return list;
    }

    @Override
    public Double getMaxSalary() throws Exception {
        String sql = "SELECT MAX(salary) FROM t_employee";
        Double value = (Double) getValue(sql);
        return value;
    }

    @Override
    public Map<Integer, Double> getAvgSalaryByDid() throws Exception {
        String sql = "SELECT did,MAX(salary) FROM t_employee GROUP BY did";
        List<Map<String, Object>> list = getListMap(sql);

        HashMap<Integer, Double> result = new HashMap<>();
        for (Map<String, Object> map : list) {
```

```
        Set<Entry<String, Object>> entrySet = map.entrySet();
        Integer did = null;
        Double salary = null;
        for (Entry<String, Object> entry : entrySet) {
            String key = entry.getKey();
            if("did".equals(key)){
                did = (Integer) entry.getValue();
            }else{
                salary = (Double) entry.getValue();
            }
        }

        result.put(did, salary);
    }
    return result;
}
}
```

## 1.6 测试类

```
package com.atguigu.dao.test;

import java.util.List;
import java.util.Scanner;

import org.junit.Test;

import com.atguigu.bean.Department;
import com.atguigu.dao.DepartmentDAO;
import com.atguigu.dao.impl.basic.DepartmentDAOImplBasic;
import com.atguigu.dao.impl.original.DepartmentDAOImpl;
import com.atguigu.utils.CMUtility;

public class TestDepartmentDAO {
    // DepartmentDAO dao = new DepartmentDAOImpl();
    DepartmentDAO dao = new DepartmentDAOImplBasic();

    @Test
    public void addDepartment() {
        Scanner input = new Scanner(System.in);
        System.out.println("请输入部门名称: ");
        String name = input.nextLine();

        System.out.println("请输入部门简介: ");
```

```
String description = input.nextLine();

Department department = new Department(name, description);

try {
    dao.addDepartment(department);
    System.out.println("添加成功");
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("添加失败");
}
}

@Test
public void getAllDepartment() throws Exception {
    List<Department> all = dao.getAll();
    for (Department department : all) {
        System.out.println(department);
    }
}

@Test
public void updateDepartment() {

    try {
        getAllDepartment();

        Scanner input = new Scanner(System.in);
        System.out.println("请选择要修改的部门编号: ");
        String did = input.nextLine();

        Department dept = dao.getById(did);

        System.out.println("请输入部门名称("+dept.getName()+"): ");
        String name = CMUtility.readString(dept.getName());

        System.out.println("请输入部门简介("+dept.getDescription()+"): ");
        String description = CMUtility.readString(dept.getDescription());

        Department department = new Department(dept.getId(),name, description);

        dao.updateDepartment(department);
        System.out.println("修改成功");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
        System.out.println("修改失败");
    }
}

@Test
public void deleteDepartment() {

    try {
        getAllDepartment();

        Scanner input = new Scanner(System.in);
        System.out.println("请选择要删除的部门编号: ");
        String did = input.nextLine();

        dao.deleteById(did);
        System.out.println("删除成功");
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("删除失败");
    }
}
}
```

```
package com.atguigu.dao.test;

import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import org.junit.Test;

import com.atguigu.bean.Employee;
import com.atguigu.dao.EmployeeDAO;
import com.atguigu.dao.impl.basic.EmployeeDAOImpl;

public class TestEmployeeDAO {
    EmployeeDAO ed = new EmployeeDAOImpl();

    @Test
    public void addEmployee()throws Exception{
        //省略键盘输入
        String ename = "张三";
```

```
String tel = "10080";
String gender = "男";
double salary = 10000;
double commissionPct = 0.3;
Date birthday = new Date();
Date hiredate = new Date();
int jobId = 2;
String email="zhangsan@lina.com";
int mid = 1;
String address = "xx";
String nativePlace = "xxx";
int did = 2;

Employee emp = new Employee(ename, tel, gender, salary, commissionPct, birthday, hiredate, jobId,
email, mid, address, nativePlace, did);
try {
    ed.addEmployee(emp);
    System.out.println("添加成功");
} catch (Exception e) {
    e.printStackTrace();
    System.out.println("添加失败");
}
}

@Test
public void updateEmployee()throws Exception{
    //省略键盘输入
    String eid = "1";
    Employee emp = ed.getById(eid);

    //这里只演示修改一下，可以修改除了 eid 以外的所有项目
    emp.setSalary(emp.getSalary() + 1000);

    try {
        ed.updateEmployee(emp);
        System.out.println("修改成功");
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("修改失败");
    }
}

@Test
public void deleteById()throws Exception{
    //省略键盘输入
```



```
String eid = "26";

    try {
        ed.deleteById(eid);
        System.out.println("删除成功");
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("删除失败");
    }
}

@Test
public void getAll()throws Exception{
    List<Employee> all = ed.getAll();
    for (Employee employee : all) {
        System.out.println(employee);
    }
}

@Test
public void getAllPage()throws Exception{
    Long count = ed.getCount();
    System.out.println("总记录数: " + count);
    int pageSize = 5;
    System.out.println("每页显示 5 条");
    int page = 2;
    System.out.println("用户选择第" + page + "页");

    List<Employee> all = ed.getAll(page, pageSize);
    for (Employee employee : all) {
        System.out.println(employee);
    }
}

@Test
public void getMaxSalary()throws Exception{
    Double maxSalary = ed.getMaxSalary();
    System.out.println("公司最高工资是: " + maxSalary);
}

@Test
public void getAvgSalaryByDid()throws Exception{
    Map<Integer, Double> map = ed.getAvgSalaryByDid();
    Set<Entry<Integer, Double>> entrySet = map.entrySet();
}
```

```
    for (Entry<Integer, Double> entry : entrySet) {  
        System.out.println(entry.getKey()+ ":" + entry.getValue());  
    }  
}  
}
```

## 2、编写通用的增删改查方法

### 2.1 通用的增删改

```
//通用的更新数据库的方法：insert,update,delete 语句时  
public static int update(String sql)throws SQLException{  
    //1、获取连接  
    Connection conn = JDBCUtils.getConnection();  
  
    //2、获取 Statement 对象，这个对象是用来给服务器传 sql 并执行 sql  
    Statement st = conn.createStatement();  
  
    //3、执行 sql  
    int len = st.executeUpdate(sql);  
  
    //4、释放资源  
    JDBCUtils.closeQuietly(st, conn);  
  
    return len;  
}  
  
// 通用的更新数据库的方法：insert,update,delete 语句，允许 sql 带?  
public static int update(String sql, Object... args)throws SQLException{  
    Connection conn = JDBCUtils.getConnection();  
  
    int len = update(conn,sql,args);  
  
    JDBCUtils.closeQuietly(conn);  
  
    return len;  
}  
  
// 通用的更新数据库的方法：insert,update,delete 语句，允许 sql 带?  
public static int update(Connection conn, String sql, Object... args)throws SQLException{  
    //2、获取 PreparedStatement 对象，这个对象是用来 sql 进行预编译  
    PreparedStatement pst = conn.prepareStatement(sql);  
  
    //3、设置 sql 中的?  
    if(args!=null && args.length>0){
```

```
//数组的下标是从 0 开始, ? 的编号是 1 开始
for (int i = 0; i < args.length; i++) {
    pst.setObject(i+1, args[i]);
}
}

//4、执行 sql
int len = pst.executeUpdate();

//5、释放资源
JDBCUtils.closeQuietly(pst);

return len;
}
```

## 2.2 尝试编写通用的查询方法

```
/**
 * 执行查询操作的 SQL 语句, SQL 可以带参数(?)
 * @param clazz Class 查询的结果需要封装的实体的 Class 类型, 例如: 学生 Student, 商品 Goods, 订单 Order
 * @param sql String 执行查询操作的 SQL 语句
 * @param args Object... 对应的每个?设置的值, 顺序要与?对应
 * @return T 封装了查询结果的实体
 * @throws Exception
 */
public static <T> T get(Class<T> clazz, String sql, Object... args) throws Exception {
    Connection conn = null;
    PreparedStatement pst = null;
    ResultSet rs = null;
    T entity = null;

    //1、获取连接
    conn = JDBCUtils.getConnection();
    //2、获取 PreparedStatement 对象, 并预编译带参数?的 SQL
    pst = conn.prepareStatement(sql);
    //3、设置参数?的值
    if(args != null && args.length > 0){
        for(int i=0; i < args.length; i++){
            pst.setObject(i+1, args[i]);
        }
    }
    //4、执行 SQL
    rs = pst.executeQuery();
    //5、获取结果, 封装到对象中
    if(rs.next()){
        entity = clazz.newInstance();
    }
}
```

```

        /*
        * 需要解决的问题?
        * (1)查询了哪些列, 即需要为哪些属性赋值, 这些列对应的属性名是什么
        * (2)获取这些列的值, 用反射为属性赋值
        *
        * clazz: 只能得到所有属性, 不知道 sql 中查询了哪些列
        * 只能依赖 sql
        */
    }

    JDBCUtils.free(rs, pst, conn);

    return entity;
}
    
```

### 2.3 ResultSetMetaData 类

可用于获取关于 ResultSet 对象中列的类型和属性信息的对象:

- **int getColumnCount()**返回当前 ResultSet 对象中的列数。
- **String getColumnName(int column)**获取指定列的名称。数据库中的字段名
- **String getColumnLabel(int column)**建议标题通常由 SQL AS 子句来指定。如果未指定 SQL AS, 则返回列名
- **String getColumnType(int column)**: 检索指定列的数据库特定的类型名称。
- **int getColumnDisplaySize(int column)**: 指示指定列的最大标准宽度, 以字符为单位。
- **boolean isNullable(int column)**: 指示指定列中的值是否可以为 null。
- **boolean isAutoIncrement(int column)**: 指示是否自动为指定列进行编号, 这样这些列仍然是只读的。

### 2.4 特殊的 SQL

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;

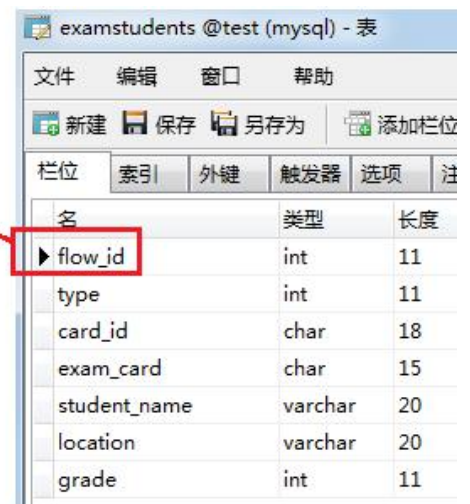
    private String cardID;

    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

## 2.5 工具类通用的查询一个实体的方法

```
//通用的查询方法之一：查询一行，即一个对象
/**
 * 执行查询操作的 SQL 语句，SQL 可以带参数(?)
 * @param clazz Class 查询的结果需要封装的实体的 Class 类型，例如：学生 Student，商品 Goods,订单 Order
 * @param sql String 执行查询操作的 SQL 语句
 * @param args Object... 对应的每个?设置的值，顺序要与?对应
 * @return T 封装了查询结果的实体
 * @throws Exception
 */
public static <T> T get(Class<T> clazz,String sql,Object... args)throws Exception{
    //1、注册驱动
    //2、获取连接
    Connection conn = JDBCUtils.getConnection();

    //3、对 sql 进行预编译
    PreparedStatement pst = conn.prepareStatement(sql);

    //4、设置?
    if(args!=null && args.length>0){
        //数组的下标是从 0 开始，? 的编号是 1 开始
        for (int i = 0; i < args.length; i++) {
            pst.setObject(i+1, args[i]);
        }
    }

    //5、查询
    ResultSet rs = pst.executeQuery();

    //获取查询的结果集的元数据信息
    ResultSetMetaData rsmd = rs.getMetaData();
    //这是查询的结果集中，一共有几列
    int count = rsmd.getColumnCount();

    T t = clazz.newInstance();//要求这个 Javabean 类型必须有无参构造

    while(rs.next()){
        /*
        * 问题?
        * (1) sql 语句中查询了几列，每一列是什么属性
        * (2) 怎么把这个值设置到 Javabean 的属性中
        */
        //循环每一行有几列
        for (int i = 0; i < count; i++) {
```

```
//          //第几列的名称
String columnName = rsmd.getColumnName(i+1);
//          //如果 sql 中没有取别名，那么就是列名，如果有别名，返回的是别名
String fieldName = rsmd.getColumnLabel(i+1);

//          //该列的值
Object value = rs.getObject(columnName);
Object value = rs.getObject(fieldName);

//          //设置 obj 对象的某个属性中
Field field = clazz.getDeclaredField(fieldName);//JavaBean 的属性名
field.setAccessible(true);
field.set(t, value);
    }

}

//5、释放资源
//5、释放资源
JDBCUtils.closeQuietly(rs,pst,conn);

return t;
}
```

## 2.6 工具类通用的查询多个实体的方法

```
//通用的查询方法之二：查询多行，即多个对象
//Class<T> clazz: 用来创建实例对象，获取对象的属性，并设置属性值
/**
 * 执行查询操作的 SQL 语句，SQL 可以带参数(?)
 * @param clazz Class 查询的结果需要封装的实体的 Class 类型，例如：学生 Student，商品 Goods,订单 Order
 * @param sql String 执行查询操作的 SQL 语句
 * @param args Object... 对应的每个?设置的值，顺序要与?对应
 * @return ArrayList<T> 封装了查询结果的集合
 * @throws Exception
 */
public static <T> ArrayList<T> getList(Class<T> clazz,String sql,Object... args)throws Exception{
    //1、注册驱动，不用了
    //2、获取连接
    Connection conn = JDBCUtils.getConnection();

    //3、对 sql 进行预编译
    PreparedStatement pst = conn.prepareStatement(sql);

    //4、对? 进行设置值
```

```
if(args!=null && args.length>0){
    for (int i = 0; i < args.length; i++) {
        pst.setObject(i+1, args[i]);
    }
}

//5、执行 sql
ResultSet rs = pst.executeQuery();
//获取结果集的元数据
ResultSetMetaData metaData = rs.getMetaData();
//获取结果中总列数
int count = metaData.getColumnCount();

//创建集合对象
ArrayList<T> list = new ArrayList<T>();

while(rs.next()){//遍历的行
    //1、每一行是一个对象
    T obj = clazz.newInstance();

    //2、每一行有很多列
    //for 的作用是为 obj 对象的每一个属性设置值
    for (int i = 0; i < count; i++) {
        //(1)每一列的名称
        String fieldName = metaData.getColumnLabel(i+1);//获取第几列的名称，如果有别名获取别名，如果没有别名获取列名
        //(2)每一列的值
        Object value = rs.getObject(i+1);//获取第几列的值
        //(3)获取属性对象
        Field field = clazz.getDeclaredField(fieldName);
        //(4)设置可见性
        field.setAccessible(true);
        //(5)设置属性值
        field.set(obj, value);
    }

    //3、把 obj 对象放到集合中
    list.add(obj);
}

//6、释放资源
JDBCUtils.closeQuietly(rs,pst,conn);

//7、返回结果
return list;
```

```
}
```

## 2.7 工具类通用的查询单值

```
//通用的查询方法之三： 查询单个值
//单值： select max(salary) from employee; 一行一列
//select count(*) from t_goods; 一共几件商品
public static Object getValue(String sql,Object... args)throws Exception{
    //2、获取连接
    Connection conn = JDBCUtils.getConnection();

    //3、对 sql 进行预编译
    PreparedStatement pst = conn.prepareStatement(sql);

    //4、对? 进行设置值
    if(args!=null && args.length>0){
        for (int i = 0; i < args.length; i++) {
            pst.setObject(i+1, args[i]);
        }
    }

    //5、执行 sql
    ResultSet rs = pst.executeQuery();

    Object value = null;
    if(rs.next()){//一行
        value = rs.getObject(1);//一列
    }

    //6、释放资源
    JDBCUtils.closeQuietly(rs,pst,conn);

    return value;
}
```

## 2.8 工具类通用的查询一行多列，非实体

```
//通用的查询方法之四： 查询多行多列，但每一行又不是一个 JavaBean
/*
 * SELECT did,AVG(salary),MAX(Salary) FROM t_employee GROUP BY did;
 * did    avg(salary)    max(salary)
 * 1      1990.90        8900
 * 2      4889           6899
 */
public static List<Map<String,Object>> getListMap(String sql,Object... args)throws Exception{
```



```
//2、获取连接
Connection conn = JDBCUtils.getConnection();

//3、对 sql 进行预编译
PreparedStatement pst = conn.prepareStatement(sql);

//4、对? 进行设置值
if(args!=null && args.length>0){
    for (int i = 0; i < args.length; i++) {
        pst.setObject(i+1, args[i]);
    }
}

//5、执行 sql
ResultSet rs = pst.executeQuery();
//获取结果集的元数据对象
ResultSetMetaData metaData = rs.getMetaData();
//一共有几列
int count = metaData.getColumnCount();
//创建 List
ArrayList<Map<String,Object>> list = new ArrayList<Map<String,Object>>();

while(rs.next()){
    //每一行是一个 Map 的对象
    HashMap<String,Object> map = new HashMap<String,Object>();

    //map 的 key 是列名
    for (int i = 0; i < count; i++) {
        //（1）获取列名或别名
        String columnName = metaData.getColumnLabel(i+1);
        //（2）获取对应的值
        Object value = rs.getObject(i+1);
        //（3）把这对值放到 map 中
        map.put(columnName, value);
    }

    //把 map 放到 List 中
    list.add(map);
}

//6、释放资源
JDBCUtils.closeQuietly(rs,pst,conn);

return list;
}
```

## 2.9 工具类通用的查询多行多列，非实体

```
//通用的查询方法之四：查询一行多列，但一行又不是一个 JavaBean
public static Map<String,Object> getMap(String sql,Object... args)throws Exception{
    List<Map<String, Object>> listMap = getListMap(sql,args);
    if(listMap.size(>0){
        return listMap.get(0);
    }
    return null;
}
```

# 第 7 章 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：

在使用开发基于数据库的 web 程序时，**传统的模式**基本是按以下步骤：

- 在主程序（如 servlet、beans、DAO）中建立数据库连接。
- 进行 sql 操作
- 断开数据库连接。

这种模式开发，存在的问题：

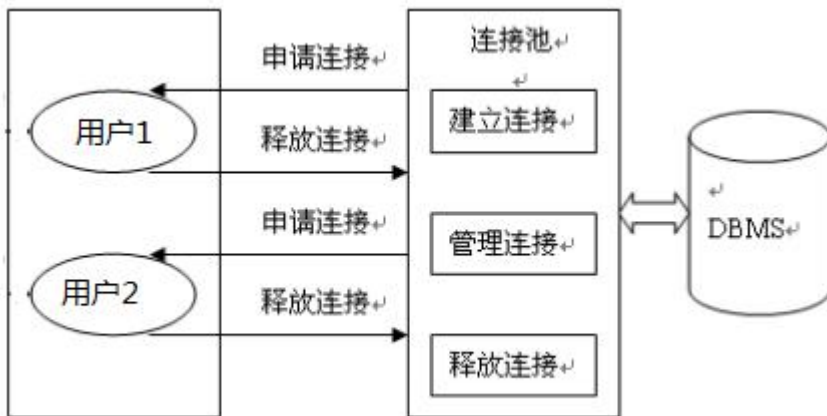
- 普通的 JDBC 数据库连接使用 DriverManager 来获取，每次向数据库建立连接的时候都要将 Connection 加载到内存中，再验证 IP 地址，用户名和密码(得花费 0.05s~1s 的时间)。需要数据库连接的时候，就向数据库要求一个，执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。**数据库的连接资源并没有得到很好的重复利用**。若同时有几百人甚至几千人在线，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃。
- 对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而**未能关闭**，将会导致**数据库系统中的内存泄漏**，最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

数据库连接池的基本思想：

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**。

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由**最小数据库连接数**来设定的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。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、两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 数据库连接池
- C3P0 数据库连接池

`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

### 2.1 DBCP 数据源

DBCP 是 Apache 软件基金组织下的开源连接池实现，该连接池依赖该组织下的另一个开源系统：Common-pool。如需使用该连接池实现，应在系统中增加如下两个 jar 文件：

- Commons-logging.jar：连接池的实现
- Commons-pool.jar：连接池实现的依赖库

Tomcat 的连接池正是采用该连接池来实现的。该数据库连接池既可以与应用服务器整合使用，也可由应用程序独立使用。

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。

## 方式一：

### 示例代码：

#### 步骤：

1、加入两个 jar

DBCP 数据库连接池的的 jar: Commons-dbcp.jar

连接池实现的依赖库: Commons-pool.jar, 如果不加这个, 运行报如下异常

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons/pool/KeyedObjectPoolFactory
    at com.jdbc.datasource.TestDBCP.main(TestDBCP.java:14)
Caused by: java.lang.ClassNotFoundException: org.apache.commons.pool.KeyedObjectPoolFactory
    at java.net.URLClassLoader.findClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    ... 1 more
```

#### 2、编写代码

```
package com.jdbc.datasource;

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

import org.apache.commons.dbcp.BasicDataSource;

public class TestDBCP {

    public static void main(String[] args) throws SQLException {
        //1.创建 DBCP 数据源（即连接池）
        BasicDataSource ds = new BasicDataSource();

        //2.设置数据源的必须属性
        ds.setDriverClassName("com.mysql.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/test");
        ds.setUsername("root");
        ds.setPassword("root");

        //3.设置数据源的可选属性
        //(1)指定数据库连接池中初始化连接数的个数
        ds.setInitialSize(10);

        //(2)指定最大的连接数：同一时刻可以同时向数据库申请的连接数
        ds.setMaxActive(50);

        //(3)在数据库连接池中保存的最少的空闲连接的数量
        ds.setMinIdle(2);
    }
}
```

//(4)等待数据库连接池分配连接的最长时间. 单位为毫秒. 超出该时间将抛出异常.

```
ds.setMaxWait(1000*5);
```

//4.从数据源中获取数据库连接

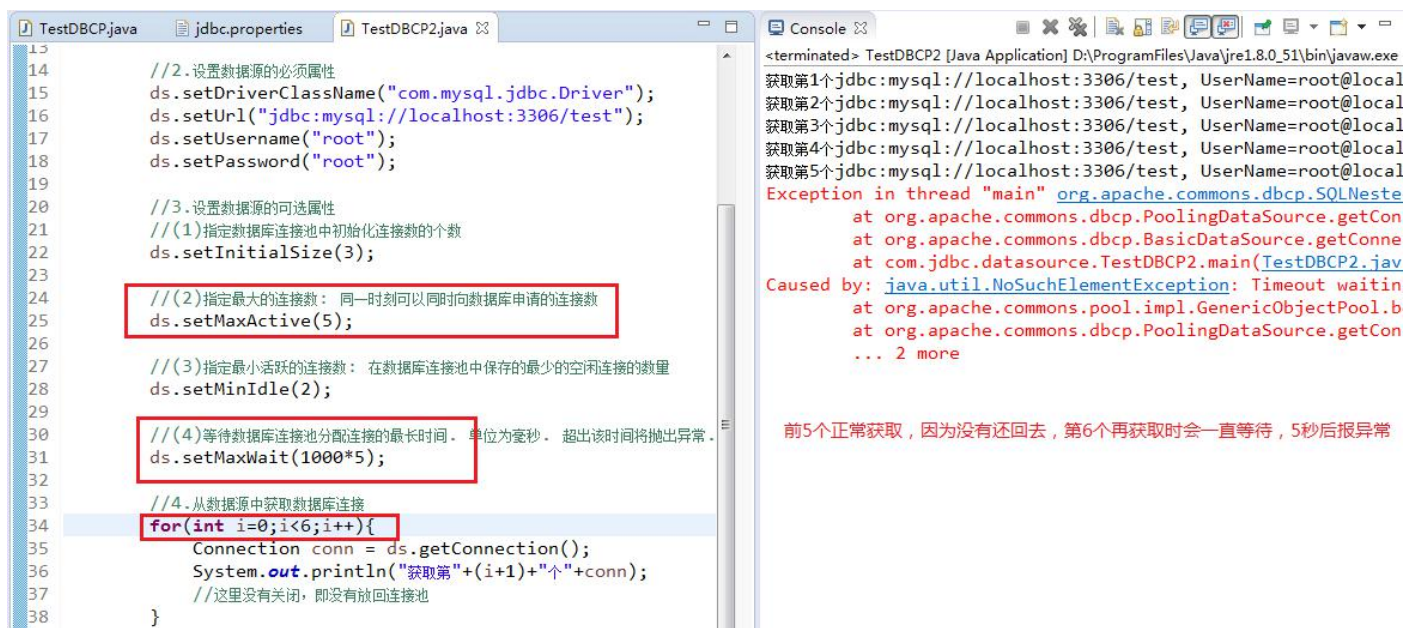
```
Connection conn = ds.getConnection();
```

```
System.out.println(conn);
```

```
}
```

```
}
```

## 测试超过连接数



```

13
14 //2. 设置数据源的必须属性
15 ds.setDriverClassName("com.mysql.jdbc.Driver");
16 ds.setUrl("jdbc:mysql://localhost:3306/test");
17 ds.setUsername("root");
18 ds.setPassword("root");
19
20 //3. 设置数据源的可选属性
21 //(1)指定数据库连接池中初始化连接数的个数
22 ds.setInitialSize(3);
23
24 //(2)指定最大的连接数： 同一时刻可以同时向数据库申请的连接数
25 ds.setMaxActive(5);
26
27 //(3)指定最小活跃的连接数： 在数据库连接池中保存的最少的空闲连接的数量
28 ds.setMinIdle(2);
29
30 //(4)等待数据库连接池分配连接的最长时间. 单位为毫秒. 超出该时间将抛出异常.
31 ds.setMaxWait(1000*5);
32
33 //4. 从数据源中获取数据库连接
34 for(int i=0;i<6;i++){
35     Connection conn = ds.getConnection();
36     System.out.println("获取第"+(i+1)+"个"+conn);
37     //这里没有关闭, 即没有放回连接池
38 }
    
```

Console Output:

```

<terminated> TestDBCP2 [Java Application] D:\ProgramFiles\Java\jre1.8.0_51\bin\javaw.exe
获取第1个jdbc:mysql://localhost:3306/test, UserName=root@local
获取第2个jdbc:mysql://localhost:3306/test, UserName=root@local
获取第3个jdbc:mysql://localhost:3306/test, UserName=root@local
获取第4个jdbc:mysql://localhost:3306/test, UserName=root@local
获取第5个jdbc:mysql://localhost:3306/test, UserName=root@local
Exception in thread "main" org.apache.commons.dbcp.SQLNeste
    at org.apache.commons.dbcp.PoolingDataSource.getConne
    at org.apache.commons.dbcp.BasicDataSource.getConne
    at com.jdbc.datasource.TestDBCP2.main(TestDBCP2.jav
    Caused by: java.util.NoSuchElementException: Timeout waitin
    at org.apache.commons.pool.impl.GenericObjectPool.b
    at org.apache.commons.dbcp.PoolingDataSource.getCon
    ... 2 more
    
```

前5个正常获取, 因为没有还回去, 第6个再获取时会一直等待, 5秒后报异常

```
package com.jdbc.datasource;
```

```
import java.sql.Connection;
```

```
import java.sql.SQLException;
```

```
import org.apache.commons.dbcp.BasicDataSource;
```

```
public class TestDBCP2 {
```

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

```
        //1.创建 DBCP 数据源 (即连接池)
```

```
        BasicDataSource ds = new BasicDataSource();
```

```
        //2.设置数据源的必须属性
```

```
        ds.setDriverClassName("com.mysql.jdbc.Driver");
```

```
        ds.setUrl("jdbc:mysql://localhost:3306/test");
```

```
        ds.setUsername("root");
```

```
        ds.setPassword("root");
```

```

//3.设置数据源的可选属性
//(1)指定数据库连接池中初始化连接数的个数
ds.setInitialSize(3);

//(2)指定最大的连接数: 同一时刻可以同时向数据库申请的连接数
ds.setMaxActive(5);

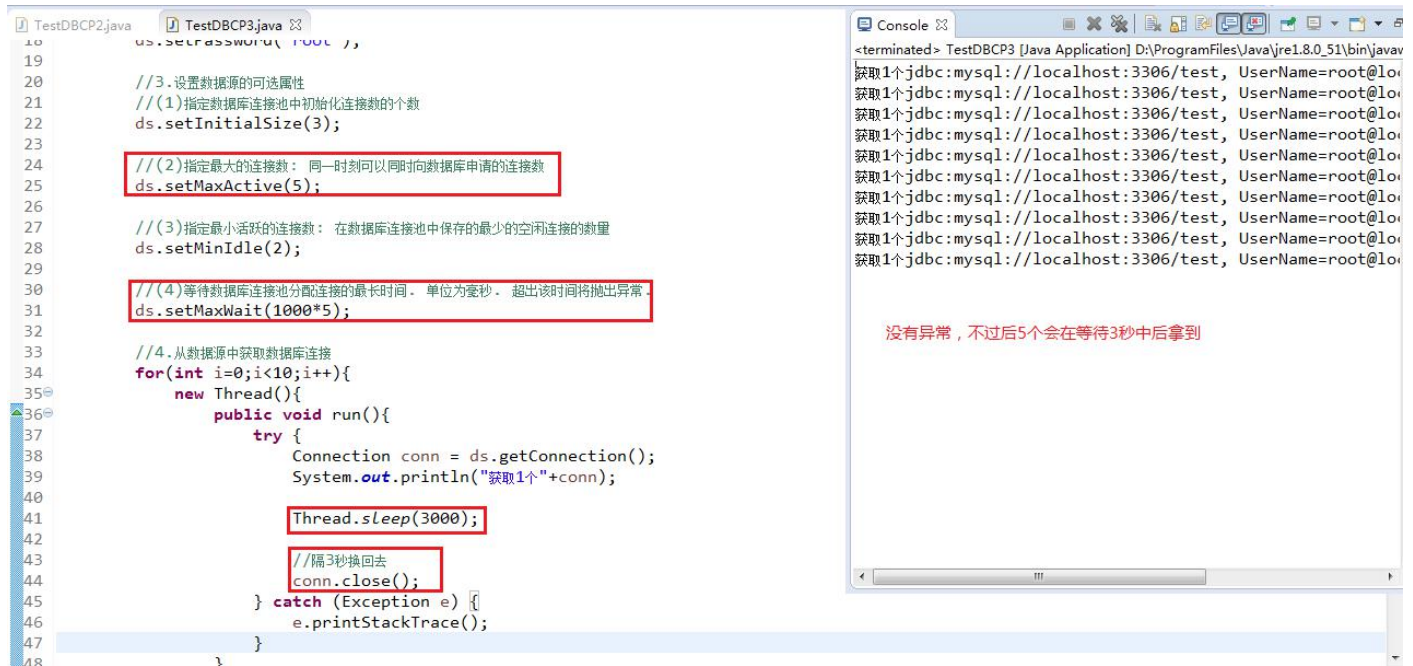
//(3)指定最小活跃的连接数: 在数据库连接池中保存的最少的空闲连接的数量
ds.setMinIdle(2);

//(4)等待数据库连接池分配连接的最长时间. 单位为毫秒. 超出该时间将抛出异常.
ds.setMaxWait(1000*5);

//4.从数据源中获取数据库连接
for(int i=0;i<6;i++){
    Connection conn = ds.getConnection();
    System.out.println("获取第"+(i+1)+"个"+conn);
    //这里没有关闭, 即没有放回连接池
}
}
}

```

## 测试如果连接重复使用



```

10 ds.setCredentials("root",
19
20 //3.设置数据源的可选属性
21 //(1)指定数据库连接池中初始化连接数的个数
22 ds.setInitialSize(3);
23
24 // (2)指定最大的连接数: 同一时刻可以同时向数据库申请的连接数
25 ds.setMaxActive(5);
26
27 // (3)指定最小活跃的连接数: 在数据库连接池中保存的最少的空闲连接的数量
28 ds.setMinIdle(2);
29
30 // (4)等待数据库连接池分配连接的最长时间. 单位为毫秒. 超出该时间将抛出异常.
31 ds.setMaxWait(1000*5);
32
33 //4.从数据源中获取数据库连接
34 for(int i=0;i<10;i++){
35     new Thread(){
36         public void run(){
37             try {
38                 Connection conn = ds.getConnection();
39                 System.out.println("获取1个"+conn);
40
41                 Thread.sleep(3000);
42
43                 //隔3秒换回去
44                 conn.close();
45             } catch (Exception e) {
46                 e.printStackTrace();
47             }
48         }
49     }

```

Console Output:

```

<terminated> TestDBCP3 [Java Application] D:\ProgramFiles\Java\jre1.8.0_51\bin\javav
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost
获取1个jdbc:mysql://localhost:3306/test, Username=root@localhost

```

没有异常, 不过后5个会在等待3秒中后拿到

```

package com.jdbc.datasource;

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

import org.apache.commons.dbcp.BasicDataSource;

```

```
public class TestDBCP3 {

    public static void main(String[] args) throws SQLException {
        //1.创建 DBCP 数据源（即连接池）
        BasicDataSource ds = new BasicDataSource();

        //2.设置数据源的必须属性
        ds.setDriverClassName("com.mysql.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/test");
        ds.setUsername("root");
        ds.setPassword("root");

        //3.设置数据源的可选属性
        //(1)指定数据库连接池中初始化连接数的个数
        ds.setInitialSize(3);

        //(2)指定最大的连接数：同一时刻可以同时向数据库申请的连接数
        ds.setMaxActive(5);

        //(3)指定最小活跃的连接数：在数据库连接池中保存的最少的空闲连接的数量
        ds.setMinIdle(2);

        //(4)等待数据库连接池分配连接的最长时间。单位为毫秒。超出该时间将抛出异常。
        ds.setMaxWait(1000*5);

        //4.从数据源中获取数据库连接
        for(int i=0;i<10;i++){
            new Thread(){
                public void run(){
                    try {
                        Connection conn = ds.getConnection();
                        System.out.println("获取 1 个"+conn);

                        Thread.sleep(3000);

                        //隔 3 秒换回去
                        conn.close();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }.start();
        }
    }
}
```

```
}  
}
```

## 方式二：

### 优化代码：

直接使用 BasicDataSource，耦合，而且在代码中需要很多 setXxx () 设置属性值

使用 dbcp.properties 属性配置文件和 BasicDataSourceFactory 更灵活

步骤：

1. 加载 dbcp 的 properties 配置文件: **配置文件中的键需要来自 BasicDataSource 的属性.**
2. 调用 BasicDataSourceFactory 的 createDataSource 方法创建 DataSource 实例
3. 从 DataSource 实例中获取数据库连接.

```
driverClassName=com.mysql.jdbc.Driver  
url=jdbc:mysql://localhost:3306/test  
username=root  
password=root  
initialSize=3  
maxActive=5  
minIdle=2  
maxWait=5000
```

```
package com.jdbc.datasource;
```

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

```
import javax.sql.DataSource;
```

```
import org.apache.commons.dbcp.BasicDataSourceFactory;
```

```
public class TestDBCP3 {
```

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

```
        //1.获取配置文件信息
```

```
        //注意：配置文件中的 key 来自 BasicDataSourceFactory 的属性（set 方法后面单词首字母改小写即可）
```

```
        Properties pro = new Properties();
```

```
        pro.load(ClassLoader.getResourceAsStream("dbcp.properties"));
```

```
        //2.创建 DBCP 数据源（即连接池）
```

```
        DataSource ds = BasicDataSourceFactory.createDataSource(pro);
```

```
        //3.从数据源中获取数据库连接
```

```
        Connection conn = ds.getConnection();
```

```
        System.out.println("获取 1 个"+conn);
```

```
    }
```

```
}
```



## JDBCUtils 修改 DBCP 版:

```
package com.atguigu.utils;

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

import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSourceFactory;

public class DBCPUtils {
    private static Properties pro = new Properties();
    private static DataSource ds ;

    static{
        try {
            //加载, 读取 jdbc.properties 配置的信息
            pro.load(ClassLoader.getSystemClassLoader().getResourceAsStream("dbcp.properties"));

            //创建池子
            ds = BasicDataSourceFactory.createDataSource(pro);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection()throws Exception{
        return ds.getConnection();
    }
}
```

## 2.2 C3P0 数据源

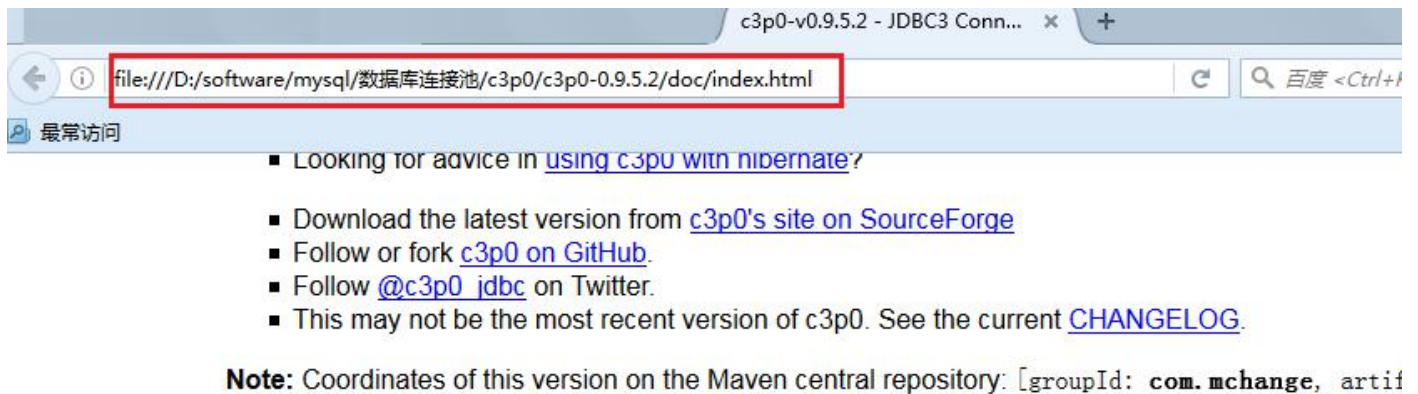
C3P0 是一个开源的 JDBC 连接池, 它实现了数据源和 JNDI 绑定, 支持 JDBC3 规范和 JDBC2 的标准扩展。目前使用它的开源项目有 Hibernate, Spring 等。

c3p0 与 dbcp 区别

dbcp 没有自动回收空闲连接的功能

c3p0 有自动回收空闲连接功能

## 方式一:



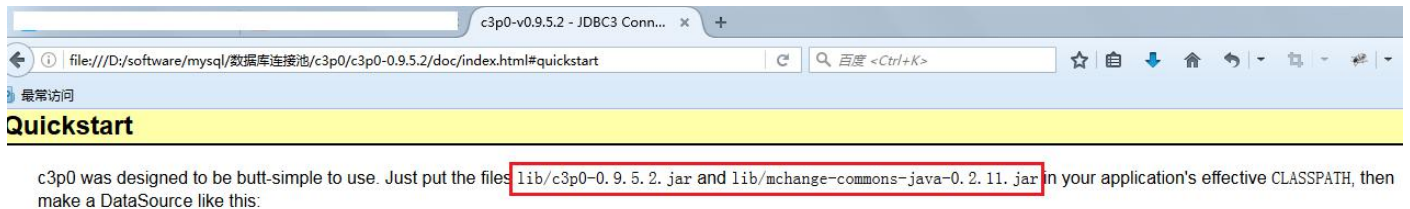
Looking for advice in [using c3p0 with hibernate?](#)

- Download the latest version from [c3p0's site on SourceForge](#)
- Follow or fork [c3p0 on GitHub](#).
- Follow [@c3p0\\_jdbc](#) on Twitter.
- This may not be the most recent version of c3p0. See the current [CHANGELOG](#).

**Note:** Coordinates of this version on the Maven central repository: [groupId: **com.mchange**, artif

## Contents

- [Contents](#)
- [Quickstart](#)
- [What is c3p0?](#)
- [Prerequisites](#)
- [Installation](#)
- [Using c3p0](#)



c3p0 was designed to be butt-simple to use. Just put the files `lib/c3p0-0.9.5.2.jar` and `lib/mchange-commons-java-0.2.11.jar` in your application's effective CLASSPATH, then make a DataSource like this:

```
import com.mchange.v2.c3p0.*;

...

ComboPooledDataSource cpds = new ComboPooledDataSource();
cpds.setDriverClass( "org.postgresql.Driver" ); //loads the jdbc driver
cpds.setJdbcUrl( "jdbc:postgresql://localhost/testdb" );
cpds.setUser( "dbuser" );
cpds.setPassword( "dbpassword" );
```

**[Optional]** If you want to turn on PreparedStatement pooling, you must also set `maxStatements` and/or `maxStatementsPerConnection` (both default to 0):

```
cpds.setMaxStatements( 180 );
```

### 步骤:

#### 1、加入 jar

如果是 c3p0-0.9.1.2 版本，加入一个 jar 即可 c3p0-0.9.1.2.jar

如果是 c3p0-0.9.2 之后的版本，需要加入两个 jar: c3p0-0.9.X.jar 和 mchange-commons-java-XX.jar

#### 2、编写代码

```
package com.jdbc.datasource;
```

```
import com.mchange.v2.c3p0.ComboPooledDataSource;
```

```
public class TestC3P0 {
    public static void main(String[] args) throws Exception {
        //1、创建 c3p0 数据源对象
        ComboPooledDataSource ds = new ComboPooledDataSource();
        //2、设置必须属性
        ds.setDriverClass( "com.mysql.jdbc.Driver" );
        ds.setJdbcUrl( "jdbc:mysql://localhost:3306/test" );
        ds.setUser("root");
        ds.setPassword("root");

        //3、获取连接
        System.out.println(ds.getConnection());
    }
}
```

## 方式二：



Browser tabs: c3p0-v0.9.5.2 - JDBC3 Conn... x Overview

Address bar: file:///D:/software/mysql/数据库连接池/c3p0/c3p0-0.9.5.2/doc/index.html

最常访问

- xvi. [User extensions to configuration](#)
- xvii. [Mixing named, per-user, and user-defined configuration extensions](#)
- 8. [Performance](#)
- 9. [Known shortcomings](#)
- 10. [Feedback and support](#)
- 11. [Appendix A: Configuration Properties](#)
- 12. [Appendix B: Configuration Files, etc.](#)**
  - i. [Overriding c3p0 defaults with a c3p0.properties file](#)
  - ii. [Overriding c3p0 defaults with "HOCON" \(typesafe-config\) configuration files](#)
  - iii. [Overriding c3p0 defaults with System properties](#)
  - iv. [Named and Per-User configuration: Overriding c3p0 defaults via c3p0-config.xml](#)
  - v. [Precedence of Configuration Settings](#)
- 13. [Appendix C: Hibernate-specific notes](#)
- 14. [Appendix D: Configuring c3p0 pooled DataSources for Apache Tomcat](#)
- 15. [Appendix E: JBoss-specific notes](#)
- 16. [Appendix F: Oracle-specific API: createTemporaryBLOB\(\) and createTemporaryCLOB\(\)](#)

(See also the API Documentation [here](#))

Here is an example c3p0-config.xml file:

```
<c3p0-config>
  <default-config>
    <property name="automaticTestTable">con_test</property>
    <property name="checkoutTimeout">30000</property>
    <property name="idleConnectionTestPeriod">30</property>
    <property name="initialPoolSize">10</property>
    <property name="maxIdleTime">30</property>
    <property name="maxPoolSize">100</property>
    <property name="minPoolSize">10</property>
    <property name="maxStatements">200</property>

    <user-overrides user="test-user">
      <property name="maxPoolSize">10</property>
      <property name="minPoolSize">1</property>
      <property name="maxStatements">0</property>
    </user-overrides>
  </default-config>

  <!-- This app is massive! -->
  <named-config name="intergalactoApp">
    <property name="acquireIncrement">50</property>
    <property name="initialPoolSize">100</property>
    <property name="minPoolSize">50</property>
    <property name="maxPoolSize">1000</property>

    <!-- intergalactoApp adopts a different approach to configuring statement caching -->
    <property name="maxStatements">0</property>
    <property name="maxStatementsPerConnection">5</property>
  </named-config>
</c3p0-config>
```



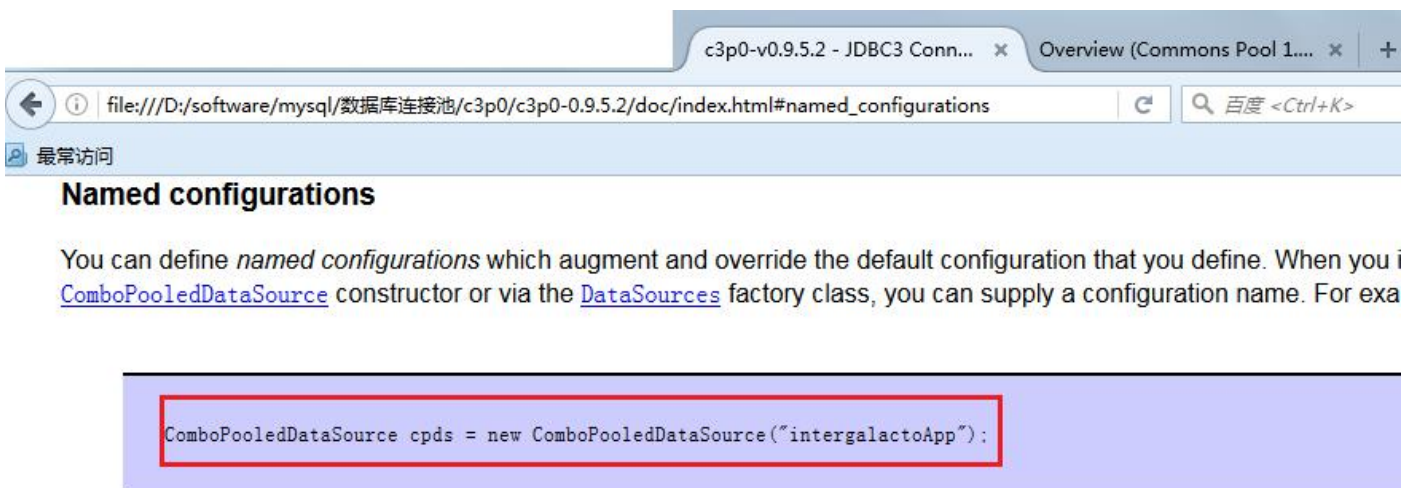
file:///D:/software/mysql/数据库连接池/c3p0/c3p0-0.9.5.2/doc/index.html

最常访问

- ii. [Using the DataSources factory class](#)
  - o [Box: Overriding authentication information \(from non-c3p0 DataSources\)](#)
- iii. [Querying Pool Status](#)
  - o [Box: Using C3P0Registry to find a reference to a DataSource](#)
- iv. [Cleaning Up Pool Resources](#)
- v. [Advanced: Building Your Own PoolBackedDataSource](#)
- vi. [Advanced: Raw Connection and Statement Operations](#)

## 7. Configuration

- i. [Introduction](#)
- ii. [Basic Pool Configuration](#)
- iii. [Managing Pool Size and Connection Age](#)
- iv. [Configuring Connection Testing](#)
  - o [Box: Simple advice on Connection testing](#)
- v. [Configuring Statement Pooling](#)
- vi. [Configuring Recovery From Database Outages](#)
- vii. [Managing Connection Lifecycles with Connection Customizers](#)
- viii. [Configuring Unresolved Transaction Handling](#)
- ix. [Configuring To Debug and Workaround Broken Client Applications](#)
- x. [Configuring To Avoid Memory Leaks On Redeploy](#)
- xi. [Other DataSource Configuration](#)
- xii. [Configuring and Managing c3p0 via JMX](#)
- xiii. [Configuring Logging](#)
- xiv. [Named configurations](#)
- xv. [Per-user configurations](#)



c3p0-v0.9.5.2 - JDBC3 Conn... x Overview (Commons Pool 1... x +

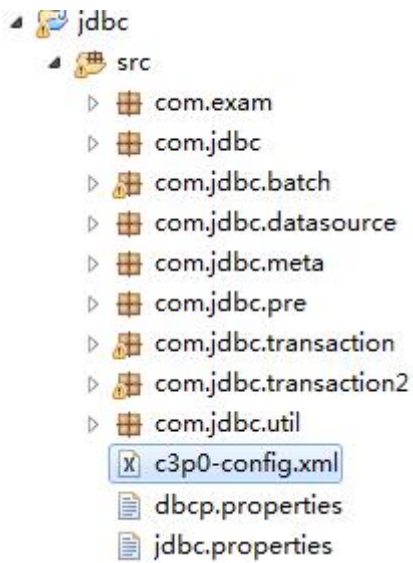
file:///D:/software/mysql/数据库连接池/c3p0/c3p0-0.9.5.2/doc/index.html#named\_configurations

最常访问

## Named configurations

You can define *named configurations* which augment and override the default configuration that you define. When you i [ComboPooledDataSource](#) constructor or via the [DataSources](#) factory class, you can supply a configuration name. For exa

```
ComboPooledDataSource cpds = new ComboPooledDataSource("intergalactoApp");
```



步骤:

1. 在 src 目录创建 c3p0-config.xml 文件, 参考帮助文档中 Appendix B: Configuration Files 的内容
2. 创建 ComboPooledDataSource 实例;  
DataSource dataSource = new ComboPooledDataSource("helloc3p0");
3. 从 DataSource 实例中获取数据库连接.

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>

    <named-config name="helloc3p0">

        <!-- 指定连接数据源的基本属性 -->
        <property name="user">root</property>
        <property name="password">root</property>
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/test</property>

        <!-- 若数据库中连接数不足时, 一次向数据库服务器申请多少个连接 -->
        <property name="acquireIncrement">5</property>
        <!-- 初始化数据库连接池时连接的数量 -->
        <property name="initialPoolSize">5</property>
        <!-- 数据库连接池中的最小的数据库连接数 -->
        <property name="minPoolSize">5</property>
        <!-- 数据库连接池中的最大的数据库连接数 -->
        <property name="maxPoolSize">10</property>
    </named-config>
```

```
</c3p0-config>
```

```
package com.jdbc.datasource;
```

```
import javax.sql.DataSource;
```

```
import com.mchange.v2.c3p0.ComboPooledDataSource;

public class TestC3P02 {
    public static void main(String[] args) throws Exception {
        DataSource dataSource = new ComboPooledDataSource("helloc3p0");
        System.out.println(dataSource.getConnection());
    }
}
```

## JDBCUtils 修改成 c3p0 版

```
package com.atguigu.utils;

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

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class C3p0Utils {
    //创建数据源，用的是 c3p0-config.xml 文件中<default-config>
    private static ComboPooledDataSource dataSource = new ComboPooledDataSource();

    //获取数据源对象
    public static ComboPooledDataSource getDataSource() {
        return dataSource;
    }

    //获取连接
    public static Connection getConnection() throws SQLException{
        return dataSource.getConnection();
    }
}
```

## 第 8 章 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`。/2QueryRunner 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。

## 使用 `QueryRunner` 类实现更新 (增、删、改、批处理)

### (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。

`public int update(Connection conn, String sql) throws SQLException`: 用来执行一个不需要置换参数的更新操作。

### (2) 插入

`public <T> T insert(Connection conn, String sql, ResultSetHandler<T> rsh) throws SQLException`: 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

`public <T> T insert(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params) throws SQLException`: 只支持 `INSERT`

`public <T> T insert(String sql, ResultSetHandler<T> rsh) throws SQLException`: 只支持 `INSERT`

`public <T> T insert(String sql, ResultSetHandler<T> rsh, Object... params) throws SQLException`: 只支持 `INSERT`

### (3) 批处理

`public int[] batch(Connection conn, String sql, Object[][] params) throws SQLException`: `INSERT, UPDATE, or DELETE` 语句

`public int[] batch(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`

`public <T> T insertBatch(String sql, ResultSetHandler<T> rsh, Object[][] params) throws SQLException`: 只支持 `INSERT`

### (4) 是否需要传递 `Connection`?

A: 不需要传递 `Connection` 对象:

前提是**不考虑事务**而且 `QueryRunner` 对象创建时指定数据源, 这样在 `QueryRunner` 的所有增删改查方法中都会从数



据源中自己获取连接

B: 必须传递 Connection 对象

如果有事务，必须传递 Connection 对象，因为同一个事务的多条语句必须在一个 Connection 连接中完成

```
public static void main(String[] args) throws Exception{
    //1、通过数据库连接池来获取连接
    DataSource ds = new ComboPooledDataSource("mypool");

    //2、传 sql，并执行，并接收结果
    String sql = "insert into t_goods(pname,price,description)values(?,?,?)";
    QueryRunner qr = new QueryRunner(ds);
    int len = qr.update(sql, "电源",78,"充电必备");//自己到数据库连接池中拿连接
    System.out.println(len>0?"添加成功":"添加失败");
}
```

```
public static void main(String[] args) throws Exception{
    //1、通过数据库连接池来获取连接
    DataSource ds = new ComboPooledDataSource("mypool");
    QueryRunner qr = new QueryRunner();

    //从 web 页面传过来，有这样的数据
    int uid = 1;
    int[] pids = {5,6,7};
    int[] amount = {1,1,1};
    double[] price = {560,58,68};

    String sql1 = "insert into t_order(ordertime,sumprice,uid)values(now(),?,?)";
    String sql2 = "insert into t_detail(oid,pid,amount)values(?,?,?) ";
    //2、获取连接
    Connection conn = null;
    try {
        conn = ds.getConnection();
        conn.setAutoCommit(false);//手动提交事务

        double sumprice = 0;
        for(int i=0; i<amount.length; i++){
            sumprice += amount[i] * price[i];
        }

        //返回的是自增的键值
        Object object = qr.insert(conn, sql1, new ScalarHandler(), sumprice,uid);

        Object[][] params = new Object[pids.length][3];
        for(int i=0; i<params.length; i++){
            for(int j=0; j<params[i].length; j++){
                params[i][0] = object;//订单编号
                params[i][1] = pids[i];//产品编号
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



PreparedStatement 和 ResultSet 的创建和关闭。

- public Object query(String sql, ResultSetHandler rsh, Object... params) throws SQLException: 几乎与第一种方法一样; 唯一的不同在于它不将数据库连接提供给方法, 并且它是从提供给构造方法的数据源(DataSource) 或使用的 setDataSource 方法中重新获得 Connection。
- public Object query(Connection conn, String sql, ResultSetHandler rsh) throws SQLException: 执行一个不需要置换参数的查询操作。
- public Object query( String sql, ResultSetHandler rsh) throws SQLException: 执行一个不需要置换参数的查询操作。

## 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。

```
@Test
public void testResultSetHandler() {
    // 1.创建 QueryRunner 的实例
    QueryRunner qr = new QueryRunner();

    Connection conn = null;

    try {
        // 2.获取连接
        conn = JDBCTools.getConnection();

        class MyResultSetHandler implements ResultSetHandler{
            @Override
            public Object handle(ResultSet rs) throws SQLException {
                Student stu = new Student();
                if (rs.next()) {
                    stu.setId(rs.getInt(1));
                    stu.setName(rs.getString(2));
                    stu.setSex(rs.getString(3));
                    stu.setMajor(rs.getString(4));
                    stu.setClasses(rs.getString(5));
                }
                return stu;
            }
        }

        String sql = "select sno,sname,sex,major,classes from t_stu where sno =?";
        // 3、使用 query 方法
        // QueryRunner 的 query 方法的返回值取决于其 ResultSetHandler 参数的 handle 方法的返回值
        Object obj = qr.query(conn, sql, new MyResultSetHandler(),1);

        System.out.println(obj);
    } catch (SQLException e) {
```

```
        e.printStackTrace();
    } finally {
        JDBCTools.free(null, null, conn);
    }
}
```

- **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

## BeanHandler 实现类

```
/*
 * BeanHandler: 把结果集的第一条记录转为创建 BeanHandler 对象时传入的 Class 参数对应的对象.
 * 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
 */
public static void main(String[] args) throws SQLException {
    //1、连接池
    DataSource ds = new ComboPooledDataSource("mypool");

    //2、直接使用 QueryRunner
    QueryRunner qr = new QueryRunner(ds);
    String sql = "select pid as id,pname,price,description from t_goods where pid =?";
    Goods goods = qr.query(sql, new BeanHandler<Goods>(Goods.class), 1);
    System.out.println(goods);
}
```

## BeanListHandler 实现类

```
/*
 * BeanListHandler: 把结果集转为一个 List, 该 List 不为 null, 但可能为空集合(size() 方法返回 0) 若
 * SQL 语句的确能够查询到记录, List 中存放创建 BeanListHandler 传入的 Class 对象对应的对象.
 */
public static void main(String[] args) throws SQLException {
    //1、连接池
    DataSource ds = new ComboPooledDataSource("mypool");

    //2、直接使用 QueryRunner
    QueryRunner qr = new QueryRunner(ds);
    String sql = "select pid as id,pname,price,description from t_goods";
    List<Goods> query = qr.query(sql, new BeanListHandler<Goods>(Goods.class));
    for (Goods goods : query) {
```

```
        System.out.println(goods);
    }
}
```

## MapHandler 实现类

```
public static void main(String[] args) throws SQLException {
    // 1、连接池
    DataSource ds = new ComboPooledDataSource("mypool");

    // 2、直接使用 QueryRunner
    QueryRunner qr = new QueryRunner(ds);

    String sql = "select did,count(*) from employee where did = 1";
    Map<String, Object> map = qr.query(sql, new MapHandler());
    Set<Entry<String, Object>> entrySet = map.entrySet();
    for (Entry<String, Object> entry : entrySet) {
        // System.out.println(entry.getKey() + "-->" + entry.getValue());
        if ("did".equals(entry.getKey())) {
            System.out.println("部门编号: " + entry.getValue());
        } else {
            System.out.println("人数: " + entry.getValue());
        }
    }
}
```

## MapListHandler 实现类

```
public static void main(String[] args) throws SQLException {
    //1、连接池
    DataSource ds = new ComboPooledDataSource("mypool");

    //2、直接使用 QueryRunner
    QueryRunner qr = new QueryRunner(ds);

    String sql = "select did,count(*) from employee group by did";
    /*
    * did count(*)
    * 1 7
    * 2 3
    * 3 1
    *
    * List:
    *   map:
    *     key(did) value(1)
    *     key(count(*) value(7)
    * 第二行
    */
}
```

```
        * map:
        *   key(did) value(2)
        *   key(count(*) value(3)
        * 第三行
        * map:
        *   key(did) value(3)
        *   key(count(*) value(1)
    *
    */
    List<Map<String, Object>> query = qr.query(sql, new MapListHandler());
    for (Map<String, Object> map : query) {
        Set<Entry<String, Object>> entrySet = map.entrySet();
        for (Entry<String, Object> entry : entrySet) {
            //System.out.println(entry.getKey() + "-->" + entry.getValue());
            if("did".equals(entry.getKey())){
                System.out.println("部门编号: " + entry.getValue());
            }else{
                System.out.println("人数: " + entry.getValue());
            }
        }
    }
}
```

## ScalarHandler 实现类

```
/*
 * ScalarHandler: 把结果集转为一个数值(可以是任意基本数据类型和字符串, Date 等)返回
 * ScalarHandler () 只取第一行第一列
 * ScalarHandler(int columnIndex): 取第一行的第 columnIndex 列
 * ScalarHandler(String columnName): 取第一行的列名为 columnName 列的值
 */
@Test
public static void main(String[] args) throws SQLException {
    //1、连接池
    DataSource ds = new ComboPooledDataSource("mypool");

    //2、直接使用 QueryRunner
    QueryRunner qr = new QueryRunner(ds);

    // String sql = "select count(*) from t_goods";
    String sql = "select max(price) from t_goods";
    Object query = qr.query(sql, new ScalarHandler());
    System.out.println(query);
}
```

## 2、JDBC 修改成 DbUtils 版

```
package com.atguigu.utils;

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

import org.apache.commons.dbutils.DbUtils;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class DBUtilsTool {
    //创建数据源，用的是 c3p0-config.xml 文件中<default-config>
    private static ComboPooledDataSource dataSource = new ComboPooledDataSource();

    //获取数据源对象
    public static ComboPooledDataSource getDataSource() {
        return dataSource;
    }

    //获取连接
    public static Connection getConnection() throws SQLException{
        return dataSource.getConnection();
    }

    public static void closeQuietly(Connection conn){
        DbUtils.closeQuietly(conn);
    }

    public static void closeQuietly(Statement st){
        DbUtils.closeQuietly(st);
    }

    public static void closeQuietly(ResultSet rs){
        DbUtils.closeQuietly(rs);
    }
}
```

### 应用案例演示：书城项目

