

# JDBC课程

## 1 JUnit

JUnit是一个用于Java编程语言的开源测试框架，用于编写和运行单元测试。



JUnit 起源于 1997年，当时Java测试过程中缺乏成熟的工具，两位编程大师 Kent Beck 和 Erich Gamma 在一次旅途的飞机上完成了JUnit 雏形的设计和实现。

### JUnit特点：

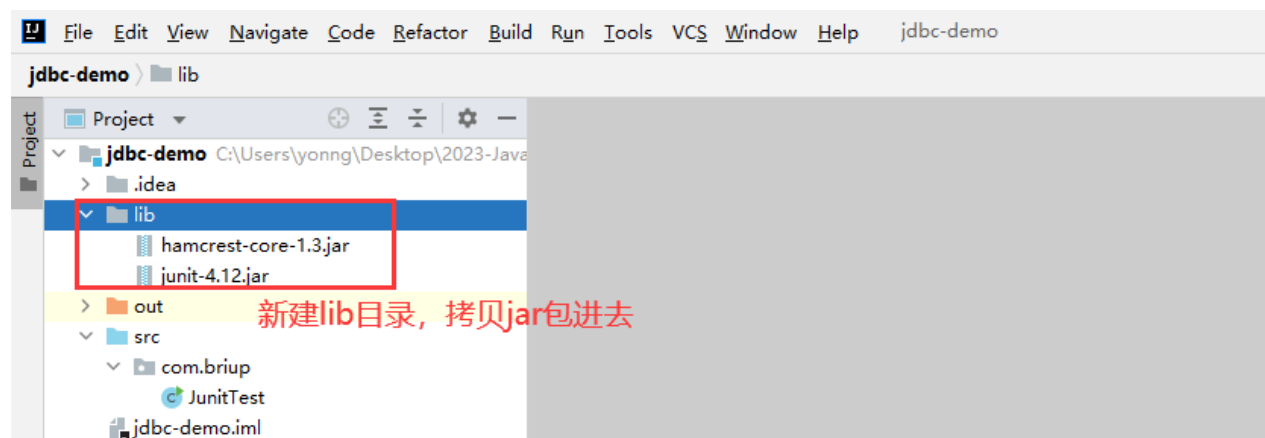
- JUnit是一个开放源代码的测试工具
- 提供注解来识别测试方法
- JUnit测试可以让你编写代码更快，并能提高质量
- JUnit优雅简洁。没那么复杂，花费时间较少
- JUnit在一个条中显示进度：如果运行良好则是绿色；如果运行失败，则变成红色

### 1.1 jar包导入

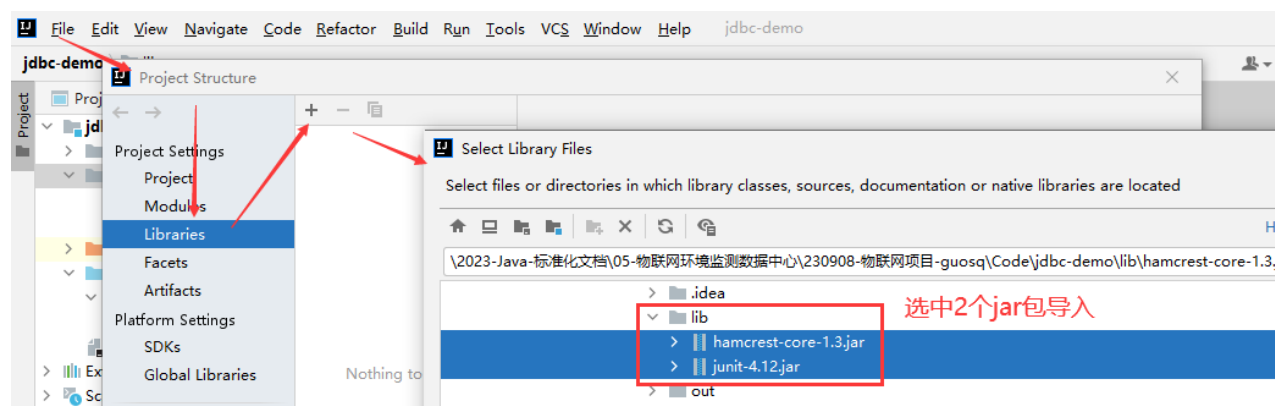
因为JUnit单元测试框架，不是JDK自带的，所以我们需要额外导入JUnit的jar包

 hamcrest-core-1.3.jar	2016/11/23 15:59	Executable Jar File	44 KB
 junit-4.12.jar	2016/11/23 15:59	Executable Jar File	308 KB

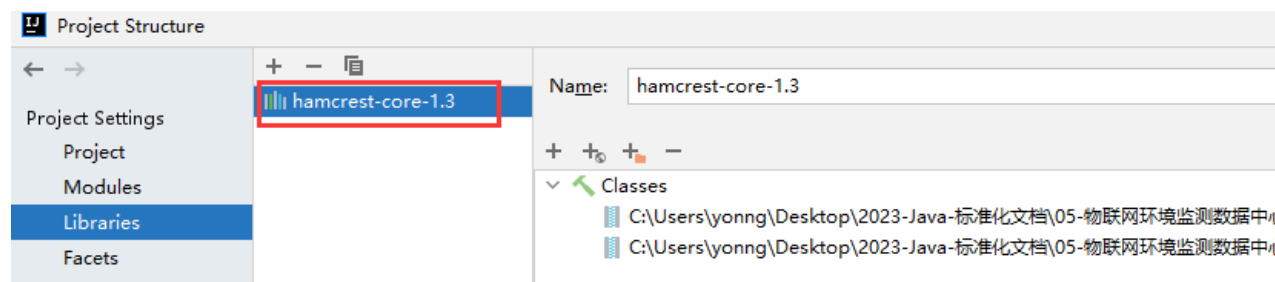
1) 创建项目jdbc-demo, 新增lib目录, 拷贝jar包进去



2) 添加jar到Libraries中



成功效果:



## 1.2 基础使用

例如, 通过一个示例, 认识JUnit的基础功能

```

1  package com.briup;
2
3  public class JunitTest {
4      public static void main(String[] args) {
5          System.out.println("hello junit");
6      }
7
8      @Test
9      public void test() {
10         System.out.println("junit test");
11     }
12 }

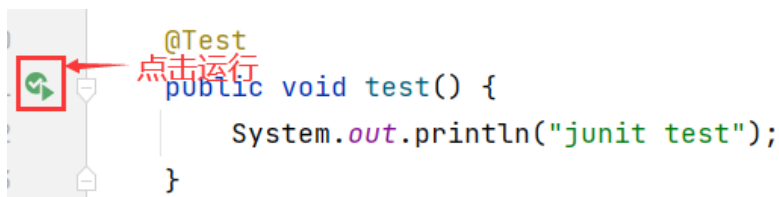
```

### 注意事项:

- 测试方法必须是公共的无参数无返回值的非静态方法
- 在测试方法上使用 `@Test` 注解标注该方法是一个测试方法

### 运行:

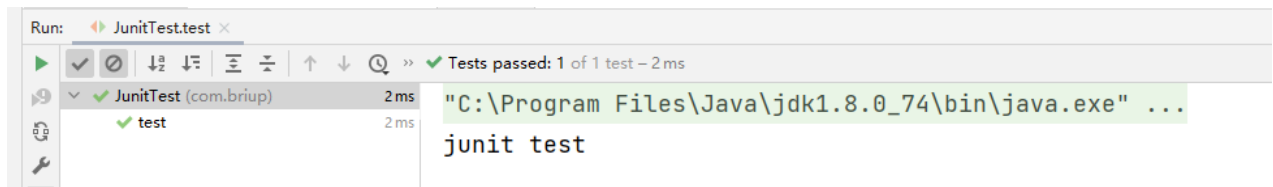
#### 运行方式1:



#### 运行方式2: 选中测试方法右键通过junit运行方法



### 结果:



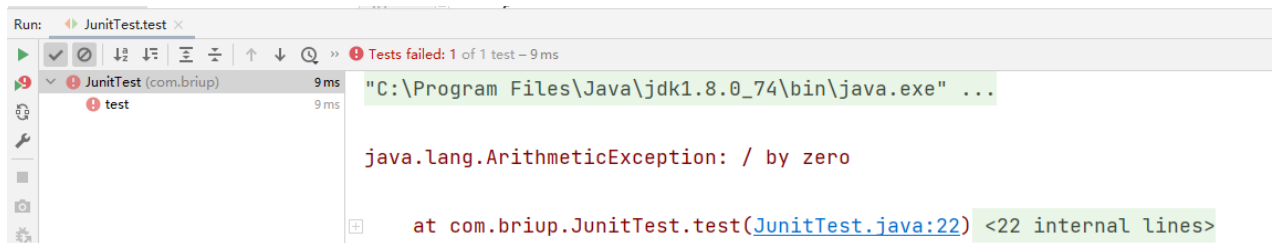
**注意：全都是绿色，则运行成功！**

**运行错误演示：**

修改代码如下：

```
1  @Test
2  public void test() {
3      int n = 1 / 0;
4      System.out.println("junit test");
5  }
```

运行结果：



## 1.3 相关注解

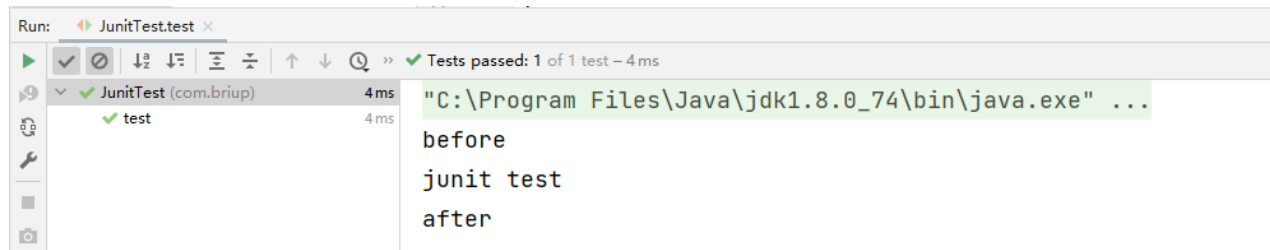
注解说明

注解	含义
@Test	表示测试该方法
@Before	在测试的方法前运行
@After	在测试的方法后运行

代码示例：

```
1  package com.briup;
2
3  public class JunitTest {
4      public static void main(String[] args) {
5          System.out.println("hello junit");
6      }
7
8      @Test
9      public void test() {
10         System.out.println("junit test");
11     }
12
13     // 新增代码
14     @Before
15     public void before() {
16         // 在执行测试代码之前执行，一般用于初始化操作
17         System.out.println("before");
18     }
19
20     // 新增代码
21     @After
22     public void after() {
23         // 在执行测试代码之后执行，一般用于释放资源
24         System.out.println("after");
25     }
26 }
```

再次运行test方法，输出结果如下：



## 2 JDBC

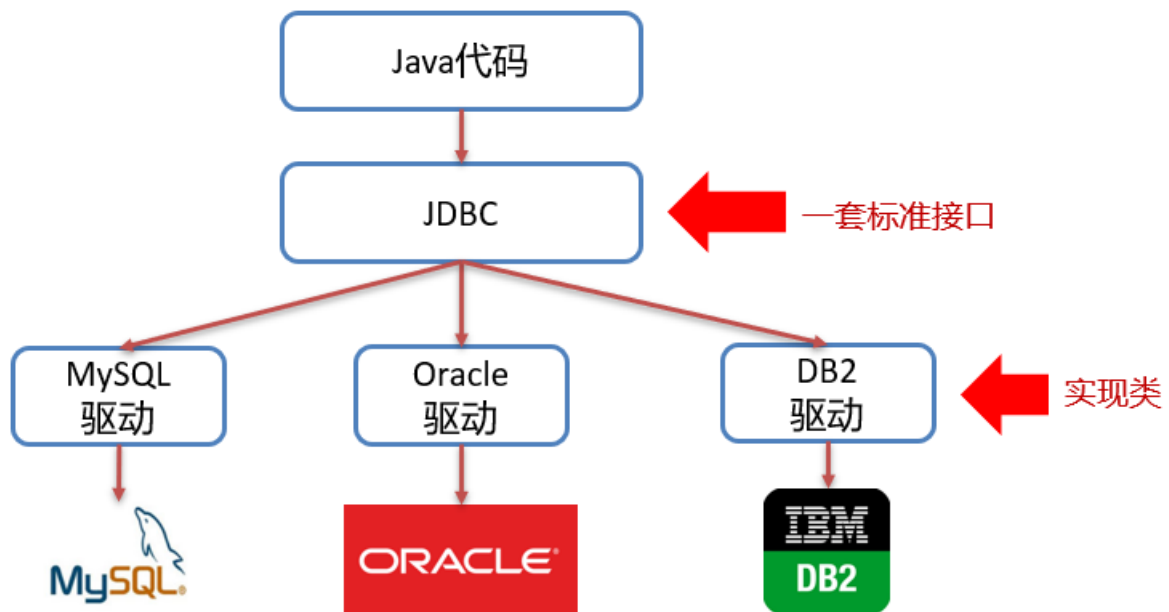
### 2.1 概述

JDBC（Java DataBase Connectivity）Java数据库连接（技术）

也就是通过Java代码，连接到数据库，并可以使用SQL语句，对数据库进行各种操作的技术。

**JDBC**，是一种技术规范，它制定了程序中连接操作不同数据库的标准API，使得程序员可以使用同一套标准的代码，去访问不同数据库。

思考，JDBC只是一套连接和操作数据库的标准（接口），那么是谁对这个标准进行的实现？又是谁使用这套标准来连接和操作数据库的？

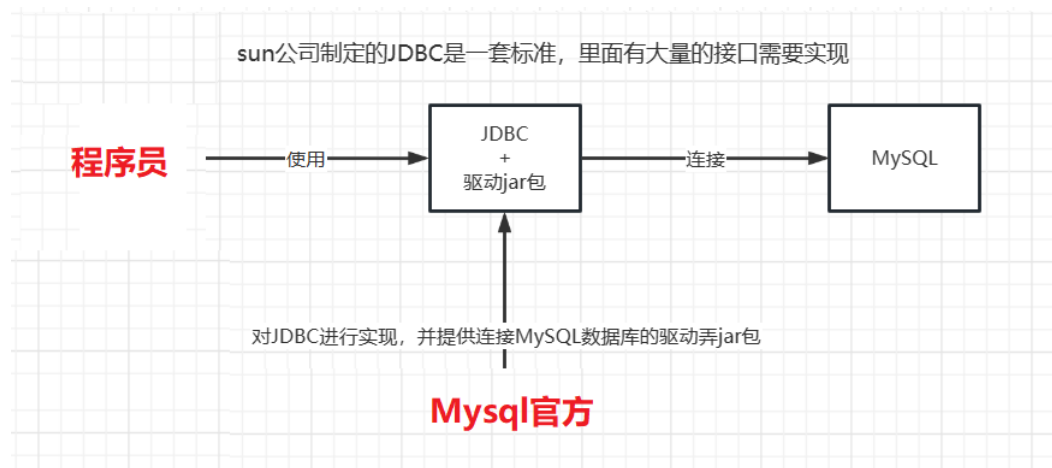


## 2.2 理解


### 理解JDBC:

- 官方（sun公司）定义的一套操作所有关系型数据库的规则，即接口
- 各个数据库厂商去实现这套接口，提供数据库驱动jar包
- 我们可以使用这套接口（JDBC）编程，真正执行的代码是驱动jar包中的实现类

下面以MySQL为例进行说明，其他数据库也一样的情况



具体jar包老师已经提供：

 mysql-connector-java-8.0.28.jar      2021/12/16 0:25      Executable Jar File      2,419 KB

注意，这就是 SUN公司、程序员、数据库、数据库厂商、JDBC、驱动jar包之间的关系

## 2.3 步骤

使用JDBC操作数据库，一般存在6个步骤，具体如下：

### 1. 注册驱动

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

### 2. 获取数据库连接对象

```
Connection conn = DriverManager.getConnection(url, username, password);
```

### 3. 获取数据库操作对象（Statement 类型或者子类型对象）

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

### 4. 执行sql语句

```
String sql = "update ...";  
stmt.executeUpdate(sql);
```

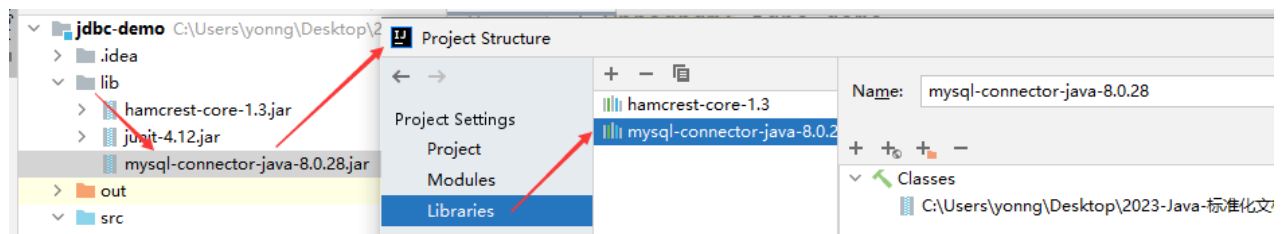


5. 处理结果集（如果需要的话，一般查询语句必须要处理）
6. 释放资源

在使用JDBC操作数据库之前，需要先获取到数据库的连接对象，而获取连接对象之前，需要先注册驱动到驱动管理器中，而注册驱动之前，需要先加载驱动类到内存中。

## 2.4 导入

MySQL驱动jar包的导入参考junit依赖的导入



## 2.5 JDBC-API

上面的章节中，我们知道JDBC操作的前两个步骤是注册驱动和获取数据库连接。

底层实现步骤：

1. 加载驱动类（加载接口具体的实现类到内存中）
2. 将驱动类注册到驱动管理器中（可以自动完成）
3. 通过驱动管理器获取数据库连接对象（当前Java程序和Mysql数据库建立建立）

只有和数据库成功建立连接，才能往下执行sql语句，进而获取结果集并处理。

## 1) DriverManager

DriverManager是驱动管理类，其作用有2个：

- 注册驱动
- 获取数据库连接

### 注册驱动

方法实现：

```
1 //注册驱动
2 public static synchronized void registerDriver(java.sql.Driver
  driver);
```

`registerDriver` 方法用于注册驱动，但是我们之前2.3章节不是这样写的，而是 `Class.forName("com.mysql.cj.jdbc.Driver");` 即：通过加载指定类，实现驱动的注册。

我们查询 `com.mysql.cj.jdbc.Driver` 驱动类源码：

```
1 package com.mysql.cj.jdbc;
2
3 public class Driver extends NonRegisteringDriver implements
  java.sql.Driver {
4     public Driver() throws SQLException {
5     }
6
7     static {
8         try {
9             //注册驱动
10             DriverManager.registerDriver(new Driver());
11         } catch (SQLException var1) {
```

```
12         throw new RuntimeException("Can't register  
13         driver!");  
14     }  
15 }
```

在该类中的静态代码块中已经执行了 `DriverManager` 对象的 `registerDriver()` 方法进行驱动的注册了。所以在实际开发中，我们只需要加载 `Driver` 类，该静态代码块就会执行，就可以加载 `Driver` 类。

注意：`java.sql.Driver` 是JDBC中提供的驱动接口，每种数据库驱动类都要实现这个接口。

驱动类已经加载，那么接下来就可以使用驱动管理器获取数据库连接对象了

## 获取数据库连接对象

### 核心方法：

```
1 public static Connection getConnection(String url,  
2     String user, String password) throws SQLException;
```

### 参数说明：

- url：连接路径

语法：`jdbc:mysql://ip地址(域名):端口号/数据库名称?参数键值对1&参数键值对2...`

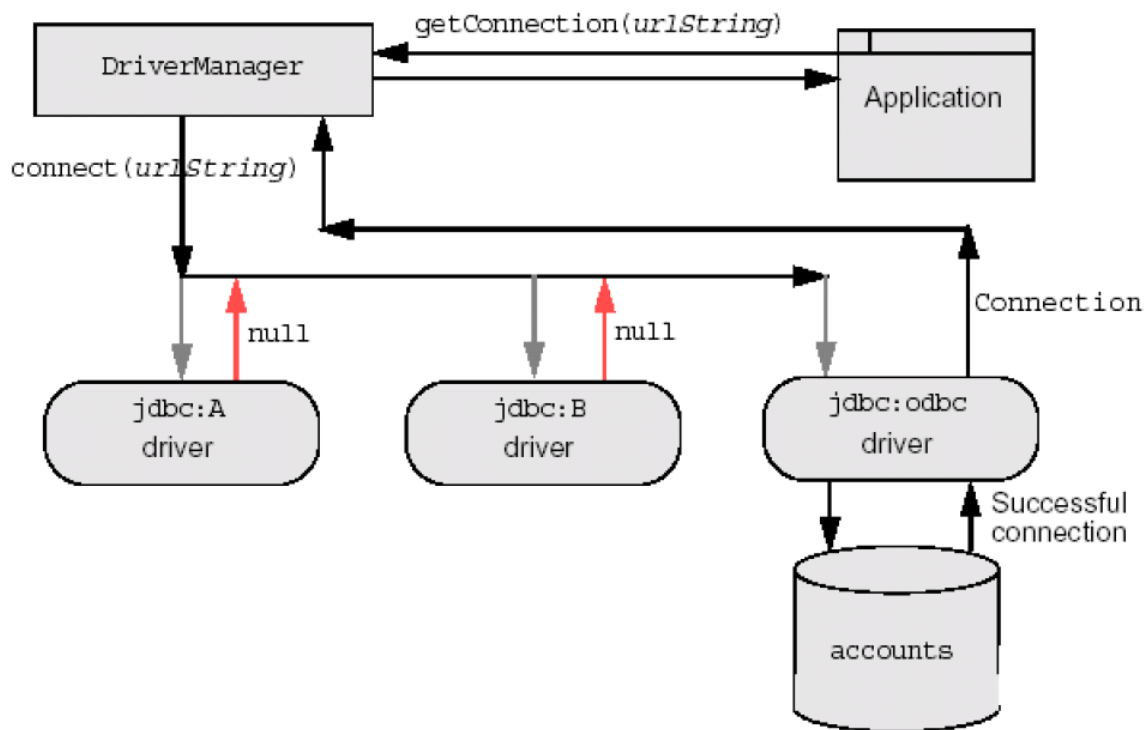
示例：`jdbc:mysql://127.0.0.1:3306/briup`

细节：

- 如果连接的是本机mysql服务器，并且mysql服务默认端口是3306，则url可以简写为：jdbc:mysql:///数据库名称?参数键值对
- 配置 useSSL=false 参数，禁用安全连接方式，解决警告提示

- user：用户名
- password：密码

### 原理分析：



根据上图可以看出，驱动管理器可以从注册的驱动中依次尝试，是否可以通过该驱动获得数据库连接对象，如果可以则返回，如果不可则尝试一下个驱动。

### 案例演示：

获取数据库连接对象并输出其值。

```

1  package com.briup;
2
3  public class ConnectionTest {
4
5      //数据库连接四要素
6      private String driverClass = "com.mysql.cj.jdbc.Driver";
7      //驱动
8      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
9      //数据库地址
10
11     private String user = "root";          //用户名
12     private String password = "briup"; //密码
13
14     //获取数据库连接并输出连接对象
15     @Test
16     public void test_getConnection() {
17         Connection conn = null;
18
19         try {
20             Class.forName(driverClass);
21             conn =
22             DriverManager.getConnection(url,user,password);
23             System.out.println(conn);
24         } catch (Exception e) {
25             e.printStackTrace();
26         }finally {
27             //注意，对象使用完之后，要记得释放资源
28             if(conn != null) {
29                 try {
30                     conn.close();
31                 } catch (SQLException e) {
32                     e.printStackTrace();
33                 }
34             }
35         }
36     }
37 }

```

## 关键点1: Connection接口

`java.sql.Connection` 是JDBC中提供的一个接口，数据库连接对象必须是该接口的实现类对象。获取到数据库连接对象之后，就可以对数据库进行各种操作了。

## 关键点2: 数据库连接四要素

- driverClass: 驱动类的全包名  
mysql8, 其值为"com.mysql.cj.jdbc.Driver"  
mysql5, 其值为"com.mysql.jdbc.Driver"
- user和password分别为用户名和密码
- url: 数据库连接路径，其后面可以追加连接参数，来设置连接的属性，常见参数如下：

补充内容（了解即可）：

参数名称	参数说明	缺省值
connectTimeout	和数据库服务器建立socket连接时的超时，单位：毫秒。0表示永不超时，	0
serverTimezone	设置服务器的时区	
useUnicode	是否使用Unicode字符集，如果参数characterEncoding 设置为utf-8，本参数值必须设置为true	false
characterEncoding	当useUnicode设置为true时，指定字符编码。比如可设置为utf-8	
autoReconnect	当数据库连接异常中断时，是否自动重新连接	
maxReconnects	autoReconnect设置为true时，重试连接的次数	3
initialTimeout	autoReconnect设置为true时，两次重连之间的时间间隔，单位：秒	2
useSSL	是否进行SSL连接 高版本设置useSSL=true,不然会有警告信息	

## 2) Connection

Connection接口主要作用是获取执行 SQL 的对象，并对事务进行管理（后续讨论）。

### 获取执行对象

- 普通执行SQL对象

```
1 Statement createStatement();
```

入门案例中就是通过该方法获取的执行对象。

- 预编译SQL的执行SQL对象：**防止SQL注入**，后续章节专门讨论

```
1 PreparedStatement prepareStatement(sql);
```

### 3) Statement

Statement对象的作用就是用来执行SQL语句。针对不同类型的SQL语句它提供不同的使用方法。

- 执行DDL、DML语句

```
1 //如果执行的是DML语句，则返回修改的行数；如果执行DDL语句，则返回0
2 int executeUpdate(String sql) throws SQLException;
```

- 执行DQL语句

```
1 //返回结果集对象(包含给定查询产生的数据),或者null
2 ResultSet executeQuery(String sql) throws SQLException;
```

该方法涉及到了 `ResultSet` 对象，而这个对象我们还没有学习，一会再重点讲解。

## 3 DDL

DDL (Data Definition Language) 数据定义语言，用来定义数据库对象、操作数据库、表等；



按JDBC操作的6个步骤，我们可以进行DDL操作。

1. 注册驱动
2. 获取数据库连接对象
3. 获取数据库操作对象（`Statement` 类型或者子类型对象）
4. 执行sql语句
5. 处理结果集（如果需要的话，一般查询语句必须要处理）
6. 释放资源

### 案例实现：

定义2个方法，分别去删除t\_user表和创建t\_user表。

```
1  package com.briup;
2
3  public class DDLTest {
4      //数据库连接四要素
5      private String driverClass = "com.mysql.cj.jdbc.Driver";
6      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
7      private String user = "root";
8      private String password = "briup";
9
10     //删除表
11     @Test
12     public void dropTable() {
13         Connection conn = null;
14         Statement stmt = null;
15         try {
16             //1.加载注册驱动
17             Class.forName(driverClass);
18             //2.获取连接对象
```

```

19         conn =
    DriverManager.getConnection(url,user,password);
20         //3.获取Statement对象
21         stmt = conn.createStatement();
22         //4.执行SQL语句
23         String sql = "drop table if exists t_user";
24         int rows = stmt.executeUpdate(sql);
25         //5.处理结果
26         System.out.println("返回结果 rows: " + rows);
27
28     } catch (Exception e) {
29         e.printStackTrace();
30     } finally {
31         //6.释放资源
32         if(stmt != null) {
33             try {
34                 stmt.close();
35             } catch (SQLException e) {
36                 e.printStackTrace();
37             }
38         }
39
40         if(conn != null) {
41             try {
42                 conn.close();
43             } catch (SQLException e) {
44                 e.printStackTrace();
45             }
46         }
47     }
48 }
49
50 //创建表
51 @Test
52 public void createTable() {
53

```

```

54         Connection conn = null;
55         Statement stmt = null;
56
57         try {
58             //1.加载注册驱动
59             Class.forName(driverClass);
60             //2.获取连接对象
61             conn = DriverManager.getConnection(url, user,
password);
62             //3.获取Statement对象
63             stmt = conn.createStatement();
64             //4.执行SQL语句
65             String sql = "create table t_user("
66                 + "id int primary key,"
67                 + "name varchar(50) not null,"
68                 + "age int"
69                 + ")";
70             int rows = stmt.executeUpdate(sql);
71             //5.处理结果
72             System.out.println("运行结果 rows: " + rows);
73
74         } catch (Exception e) {
75             e.printStackTrace();
76         } finally {
77             //6.释放资源
78             if (stmt != null) {
79                 try {
80                     stmt.close();
81                 } catch (SQLException e) {
82                     e.printStackTrace();
83                 }
84             }
85             if (conn != null) {
86                 try {
87                     conn.close();
88                 } catch (SQLException e) {

```

```
89             e.printStackTrace();
90         }
91     }
92 }
93 }
94 }
```

注意:

- 释放资源时，先创建的后释放，后创建的先释放!
- executeUpdate(DDL语句), 执行后，返回值为0

## 4 DML

---

DML(Data Manipulation Language)数据操作语言，用来对表中的数据进行insert, update, delete操作,

executeUpdate(DML语句), 执行后，返回值为修改的行数。

### 案例实现:

对上述案例中创建的t\_user表，进行insert、update、delete操作。

insert功能实现:

```
1  package com.briup;
2
3  public class DMLTest {
4      private String driverClass = "com.mysql.cj.jdbc.Driver";
5      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6      private String user = "root";
7      private String password = "briup";
8  }
```

```

9      //插入数据
10     @Test
11     public void insert() {
12         Connection conn = null;
13         Statement stmt = null;
14
15         try {
16             //1.加载注册驱动
17             Class.forName(driverClass);
18             //2.获取连接对象
19             conn =
20             DriverManager.getConnection(url,user,password);
21             //3.获取Statement对象
22             stmt = conn.createStatement();
23             //4.执行SQL语句
24             String sql = "insert into t_user(id,name,age)
25             values(1,'tom',20)";
26             int rows = stmt.executeUpdate(sql);
27             //5.处理结果
28             System.out.println("运行结果 rows: " + rows);
29
30         } catch (Exception e) {
31             e.printStackTrace();
32         } finally {
33             //6.释放资源
34             if(stmt != null) {
35                 try {
36                     stmt.close();
37                 } catch (SQLException e) {
38                     e.printStackTrace();
39                 }
40             }
41             if(conn != null) {
42                 try {
43                     conn.close();
44                 } catch (SQLException e) {

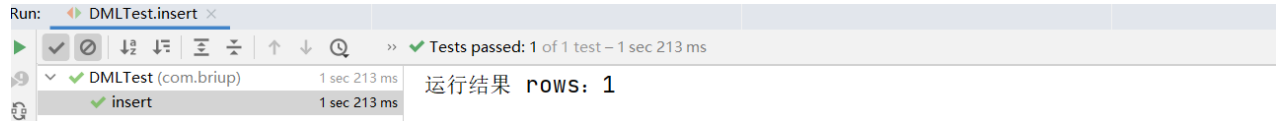
```

```

43             e.printStackTrace();
44         }
45     }
46 }
47 }
48 }

```

运行结果显示插入1条数据：



**注意，JDBC代码执行的sql语句不需要加分号；，加上分号会执行报错。但是在命令行里面执行sql语句，是要加分号的！**

update功能实现：

```

1  //更新数据
2  @Test
3  public void update() {
4
5      Connection conn = null;
6      Statement stmt = null;
7
8      try {
9          //1.加载注册驱动
10         Class.forName(driverClass);
11         //2.获取连接对象
12         conn = DriverManager.getConnection(url,user,password);
13         //3.获取Statement对象
14         stmt = conn.createStatement();
15         //4.执行SQL语句
16         String sql = "update t_user set name='mary' where id =
17         1";
18         int rows = stmt.executeUpdate(sql);

```

```

18         //5.处理结果
19         System.out.println("运行结果 rows: " + rows);
20
21     } catch (Exception e) {
22         e.printStackTrace();
23     } finally {
24         //6.释放资源
25         if(stmt != null) {
26             try {
27                 stmt.close();
28             } catch (SQLException e) {
29                 e.printStackTrace();
30             }
31         }
32         if(conn != null) {
33             try {
34                 conn.close();
35             } catch (SQLException e) {
36                 e.printStackTrace();
37             }
38         }
39     }
40 }

```

自行测试输出。

delete功能实现：

```

1     //删除数据
2     @Test
3     public void delete() {
4
5         Connection conn = null;
6         Statement stmt = null;
7

```

```

8      try {
9          //1.加载注册驱动
10         Class.forName(driverClass);
11         //2.获取连接对象
12         conn = DriverManager.getConnection(url,user,password);
13         //3.获取Statement对象
14         stmt = conn.createStatement();
15         //4.执行SQL语句
16         String sql = "delete from t_user where id = 1";
17         int rows = stmt.executeUpdate(sql);
18         //5.处理结果
19         System.out.println("运行结果 rows: " + rows);
20
21     } catch (Exception e) {
22         e.printStackTrace();
23     } finally {
24         //6.释放资源
25         if(stmt != null) {
26             try {
27                 stmt.close();
28             } catch (SQLException e) {
29                 e.printStackTrace();
30             }
31         }
32         if(conn != null) {
33             try {
34                 conn.close();
35             } catch (SQLException e) {
36                 e.printStackTrace();
37             }
38         }
39     }
40 }

```

自行测试输出。



思考：通过上述JDBC代码可以成功操作Mysql数据库，现在如果需要更换数据库，如oracle、SQLServer、DB2等，如何修改上述代码，能够使程序成功执行。

## 5 DQL

DQL(Data Query language)数据查询语言，对表中的数据进行查询select操作；

执行DQL语句：

```
1 //返回结果集对象(包含给定查询产生的数据),或者null
2 ResultSet executeQuery(String sql) throws SQLException;
```

该方法涉及到了 `ResultSet` 对象，具体讲解如下。

### 1) ResultSet

结果集ResultSet对象作用：封装了SQL查询语句的结果。

从 `ResultSet` 对象中我们可以获取查询的数据，具体方法如下：

```
1 // 将光标从当前位置向前移动一行
2 // 判断当前行是否为有效行
3 boolean next() throws SQLException;
4
5 // 返回true则表示当前行存在数据，为有效行
6 // 返回false则表示当前行没有数据，为无效行
```

```

1 // 获取指定列数据
2 public xxx getXxx(参数); //如: int getInt(参数); String
   getString(参数);
3
4 // 参数类型有2种
5 // int类型, 表示列的编号, 从1开始
6 // String类型, 表示列的名称

```

## JDBC处理ResultSet原理分析:

对象	t_user @briup (230719-mys...	
开始事务	文本	筛选
排序	导入	导出
id	name	age
1	mary	20
2	tom	19
3	jack	21

光标指向第一行有效数据之前

一开始光标指定于第一行前, 如图所示红色箭头指向于表头行。

当我们调用了 `next()` 方法后, 光标就下移到第一行数据, 同时方法返回 `true`。

此时就可以通过 `getInt("id")` 获取当前行id字段的值, 也可以通过 `getString("name")` 获取当前行name字段的值。

如果想获取下一行的数据, 继续调用 `next()` 方法, 以此类推。

## 2) 案例实现

执行select查询语句, 并解析结果集对象, 获取查询的数据并输出。

```

1 package com.briup;
2
3 public class DQLTest {
4     private String driverClass = "com.mysql.cj.jdbc.Driver";

```

```

5     private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6     private String user = "root";
7     private String password = "briup";
8
9     @Test
10    public void select() {
11        Connection conn = null;
12        Statement stmt = null;
13        ResultSet rs = null;
14
15        try {
16            //1.加载注册驱动
17            Class.forName(driverClass);
18            //2.获取连接对象
19            conn =
20            DriverManager.getConnection(url,user,password);
21            //3.获取Statement对象
22            stmt = conn.createStatement();
23            //4.执行SQL语句
24            String sql = "select id,name,age from t_user";
25            rs = stmt.executeQuery(sql);
26            //5.处理结果集
27            while(rs.next()) {
28
29                //5.1 使用列的序号获取值，从1开始
30                //long id = rs.getInt(1);
31                //String name = rs.getString(2);
32                //int age = rs.getInt(3);
33
34                //5.2 使用列的名字获取值
35                long id = rs.getInt("id");
36                String name = rs.getString("name");
37                int age = rs.getInt("age");
38
39                System.out.println("id: " + id + ", name: " +
40                name + ", age: " + age);

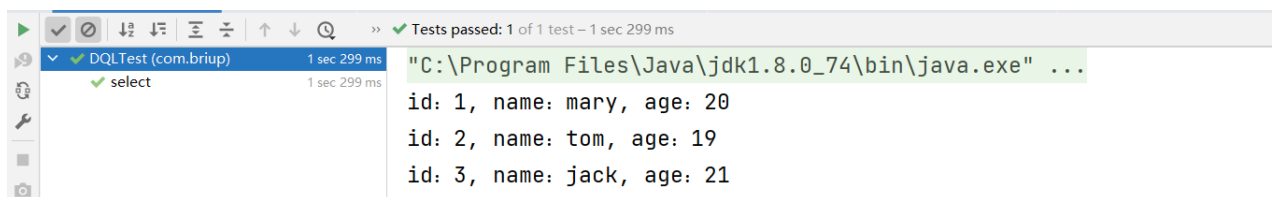
```

```

39         }
40     } catch (Exception e) {
41         e.printStackTrace();
42     } finally {
43         //6.释放资源
44         if(rs != null) {
45             try {
46                 rs.close();
47             } catch (SQLException e) {
48                 e.printStackTrace();
49             }
50         }
51         if(stmt != null) {
52             try {
53                 stmt.close();
54             } catch (SQLException e) {
55                 e.printStackTrace();
56             }
57         }
58         if(conn != null) {
59             try {
60                 conn.close();
61             } catch (SQLException e) {
62                 e.printStackTrace();
63             }
64         }
65     }
66 }
67 }

```

运行输出：



## 6 登录案例

### 准备工作:

修改t\_user表，增加password列，并重新添加数据。

```
1  -- 清空表中数据
2  delete from t_user;
3
4  -- 删除指定列
5  -- alter table t_user drop column password;
6
7  -- 往name后面新增password列
8  alter table t_user add column password varchar(20) not null
   after name;
9
10 -- 新增数据
11 insert into t_user values(1,'admin','admin',21),
   (2,'briup','briup',20);
```

**思考1:** 如果用户admin要登录，后端要执行什么sql语句？

```
select count(*) from t_user where name='admin' and
password='admin';
```

上述sql语句执行过程中，如果结果为0，则登录校验失败，如果结果为非0值，则登录校验成功。

### 案例实现:

提前准备好用户名和密码（实际开发中需要在前端页面上录入），然后拼接得到sql语句，最终执行，判断用户是否登录！

```
1  package com.briup;
2
3  public class LoginTest {
4      private String driverClass = "com.mysql.cj.jdbc.Driver";
5      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6      private String user = "root";
7      private String password = "briup";
8
9      @Test
10     public void test_login() {
11         Connection conn = null;
12         Statement stmt = null;
13         ResultSet rs = null;
14
15         try {
16             //提前准备好用户名和密码（实际开发中需要在前端页面上录入）
17             String username = "admin";
18             String passwd = "admin";
19
20             //1.加载注册驱动
21             Class.forName(driverClass);
22             //2.获取连接对象
23             conn =
24             DriverManager.getConnection(url,user,password);
25             //3.获取Statement对象
26             stmt = conn.createStatement();
27             //4.执行SQL语句
28             String sql = "select count(*) from t_user where
29             name = '"+username+"' and password = '"+passwd + "'";
30
31             rs = stmt.executeQuery(sql);
32
33             //5.处理结果集
```

```

32         while(rs.next()) {
33             int count = rs.getInt(1);
34             System.out.println("count: " + count);
35             if(count == 0)
36                 System.out.println("登录失败!");
37             else
38                 System.out.println("登录成功!");
39         }
40
41     } catch (Exception e) {
42         e.printStackTrace();
43     } finally {
44         //6.释放资源
45         if(rs != null) {
46             try {
47                 rs.close();
48             } catch (SQLException e) {
49                 e.printStackTrace();
50             }
51         }
52         if(stmt != null) {
53             try {
54                 stmt.close();
55             } catch (SQLException e) {
56                 e.printStackTrace();
57             }
58         }
59         if(conn != null) {
60             try {
61                 conn.close();
62             } catch (SQLException e) {
63                 e.printStackTrace();
64             }
65         }
66     }
67 }

```

分别使用有效的用户名和密码进行测试。

**思考2：**如果用户名传入的是 `admin` 而密码传入其他值，能不能登录成功？

答案是可以的，借助**SQL注入**可以实现。

## 7 SQL注入

SQL注入是一种常见的**安全漏洞**，它发生在应用程序没有正确验证和处理用户输入的情况下。

**SQL注入攻击原理：**

**利用用户输入的数据作为SQL查询语句的一部分，从而改变原始查询的意图。**

攻击者可通过注入恶意SQL代码来执行绕过身份验证、窃取数据、修改数据、执行任意命令等操作。

由于没有对用户输入内容进行充分检查，而SQL又是拼接而成，在用户输入参数时，在参数中添加一些SQL关键字，达到改变SQL运行结果的目的，也可以完成恶意攻击。

如果要利用SQL注入攻击，则执行下面2条SQL语句即可：

```
select * from t_user where name='admin' and password='' or  
'1'='1'
```



```
select * from t_user where name='' OR '1'='1'; -- and
password='test'
```

请自行分析，为什么上面2条sql语句执行都能登录成功！

### 案例展示：

使用上述SQL注入模拟攻击。LoginTest新增方法：

```
1  //SQL注入模拟
2  @Test
3  public void test_sqlInjection() {
4      Connection conn = null;
5      Statement stmt = null;
6      ResultSet rs = null;
7
8      try {
9          //SQL注入1
10         String username = "admin";
11         String passwd = "' or '1'='1'";
12
13         //SQL注入2
14         //      String username = "' OR '1'='1'; -- ";
15         //      String passwd = "随意";
16
17         //1.加载注册驱动
18         Class.forName(driverClass);
19         //2.获取连接对象
20         conn = DriverManager.getConnection(url,user,password);
21         //3.获取Statement对象
22         stmt = conn.createStatement();
23         //4.执行SQL语句
24         String sql = "select count(*) from t_user where name =
25         '"+username+"' and password = '"+passwd + "'";
26         System.out.println("sql: " + sql);
```

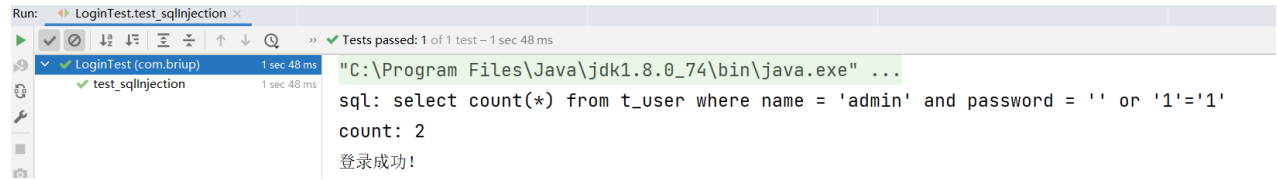
```

26         rs = stmt.executeQuery(sql);
27
28         //5.处理结果集
29         while(rs.next()) {
30             int count = rs.getInt(1);
31             System.out.println("count: " + count);
32             if(count == 0)
33                 System.out.println("登录失败! ");
34             else
35                 System.out.println("登录成功! ");
36         }
37
38     } catch (Exception e) {
39         e.printStackTrace();
40     } finally {
41         //6.释放资源
42         if(rs != null) {
43             try {
44                 rs.close();
45             } catch (SQLException e) {
46                 e.printStackTrace();
47             }
48         }
49         if(stmt != null) {
50             try {
51                 stmt.close();
52             } catch (SQLException e) {
53                 e.printStackTrace();
54             }
55         }
56         if(conn != null) {
57             try {
58                 conn.close();
59             } catch (SQLException e) {
60                 e.printStackTrace();
61             }

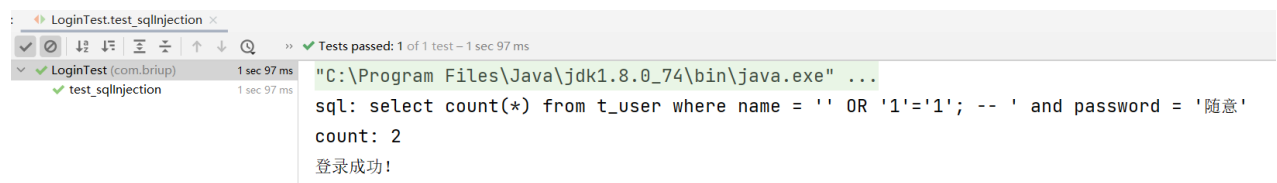
```

```
62     }
63 }
64 }
```

情形1:



情形2:



**思考：如何解决SQL注入？**

可以借助PreparedStatement对象解决。

## 8 PreparedStatement

**PreparedStatement** 是 **Statement** 接口的子接口，提供了一种更高效、更安全的方式执行SQL语句。其适用于需要多次执行相同或相似SQL语句的场景。

与 **Statement** 不同，**PreparedStatement** 在执行之前会先将SQL语句发送给数据库进行预编译，然后可以多次使用相同的预编译语句执行不同的参数值。

**PreparedStatement** 预编译优点：

- 提高执行效率

- 能够**防止SQL注入攻击**
- 支持各种数据类型和批处理操作

在具体学习 `PreparedStatement` 前，我们先认识下同构和异构。

## 8.1 同构异构

### 异构SQL语句：

结构不相同的SQL语句，例如：

```
1  delete from t_user where id=3;
2
3  select * from t_user where name='tom'
```

### 同构SQL语句：

同构即结构相同。如果有多条sql语句，它们的格式相同，只是要操作的数据不同，那么可以称它们为同构sql语句。

例如：

```
1  insert into t_user(id,name,password,age)
   values(1,'tom1','tom',21);
2  insert into t_user(id,name,password,age)
   values(2,'tom2','tom',22);
3  insert into t_user(id,name,password,age)
   values(3,'tom3','tom',23);
```

同构SQL，在后续的操作中，可以按照下面形式展示：

```
1  -- 问号表示占位符
2  insert into t_user(id,name,password,age) values(?,?,?,?);
3
4  -- 第一次4个问号对应值:  1 tom1 tom 21
5  -- 第二次4个问号对应值:  2 tom2 tom 22
6  -- 第三次4个问号对应值:  3 tom3 tom 23
7
8  // 使用pstmt对象给?赋不同数据值
9  pstmt.setInt(id);
10 pstmt.setString(name);
11 pstmt.setString(password);
12 pstmt.setInt(age);
```

**注意：**执行大量同构SQL语句的情况下，使用PreparedStatement会大大提高效率。

## 8.2 原理分析

了解即可！

**原理图：**

### 同构SQL语句:

```
insert into t_user(id,name,password,age)
values(?,?,?,?);
```

### 使用pstmt对象给? 赋不同数据值

```
pstmt.setInt(id);
pstmt.setString(name);
pstmt.setString(password);
pstmt.setInt(age);
```



### JDBC使用Statement对象执行SQL语句流程（非预编译）：

1. 将SQL语句（含结构+数据）发送到MySQL服务器端
2. MySQL服务端检查SQL语句  
检查语法是否正确
3. MySQL服务端编译SQL语句  
将SQL语句编译成可执行的函数
4. 执行SQL语句

**注意：检查和编译SQL需要花费大量时间，甚至比执行SQL时间还要长。**

### JDBC使用PreparedStatement对象执行SQL语句流程（预编译）：

1. 将SQL语句（含结构）发送到MySQL服务器端
2. MySQL服务端检查SQL语句
3. MySQL服务端编译SQL语句

## 编译成功，以后可重复使用

4. PreparedStatement对象传递参数值到服务器
5. 执行SQL语句
6. 如果再次执行SQL语句，则到第4步，再次传递参数值到服务器，然后直接执行即可。

**结论：预编译省去了后续执行SQL的检查和编译过程，大大提高了性能。**

## 8.3 相关API

### 1) 获取 PreparedStatement 对象

```
1 // java.sql.Connection接口提供方法，获取对象
2 // 注意，因为要预编译，所以一定要提前传递SQL语句
3 PreparedStatement preparedStatement(String sql) throws
  SQLException;
```

案例：

```
1 // SQL语句中的参数值，使用? 占位符替代
2 String sql = "select count(*) from t_user where name = ? and
  password = ?";
3 // 通过Connection对象获取，并传入对应的sql语句
4 PreparedStatement pstmt = conn.prepareStatement(sql);
```

### 2) 设置参数值并执行SQL语句

上面的sql语句中参数使用? 进行占位，在之前之前肯定要设置这些? 的值。

```
1 public interface PreparedStatement extends Statement {
2     //参数1: ?的位置编号,从1开始
3     //参数2: ?的值
```

```

4      void setString(int parameterIndex, String x) throws
      SQLException;
5      void setInt(int parameterIndex, int x) throws
      SQLException;
6      //其他setXxx方法，与上面类似，不做赘述
7
8      // 执行DDL语句和DML语句
9      ResultSet executeQuery() throws SQLException;
10
11     // 执行DQL语句
12     int executeUpdate() throws SQLException;
13
14     // 省略...
15 }

```

**注意：**调用上述2个执行方法时不需要传递SQL语句，因为获取SQL语句执行对象时已经对SQL语句进行预编译了。

## 8.4 具体应用

### 案例展示：

使用PreparedStatement解决上述案例中SQL注入问题。

```

1  package com.briup;
2
3  public class PreparedStatementTest {
4      private String driverClass = "com.mysql.cj.jdbc.Driver";
5      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6      private String user = "root";
7      private String password = "briup";
8
9      @Test
10     public void test_login() {
11         Connection conn = null;

```



```

12         PreparedStatement pstmt = null;
13         ResultSet rs = null;
14
15         try {
16             //正确参数
17             String username = "admin";
18             String passwd = "admin";
19
20             //SQL注入1
21             //         String username = "admin";
22             //         String passwd = "' or '1'='1'";
23
24             //SQL注入2
25             //         String username = "' OR '1'='1'; -- ";
26             //         String passwd = "随意";
27
28             //1.加载注册驱动
29             Class.forName(driverClass);
30             //2.获取连接对象
31             conn =
DriverManager.getConnection(url,user,password);
32             //3.获取PreparedStatement对象
33             String sql = "select count(*) from t_user where
name = ? and password = ?";
34             pstmt = conn.prepareStatement(sql);
35             //4.执行SQL语句
36             // 4.1 设置参数值
37             pstmt.setString(1,username);
38             pstmt.setString(2,passwd);
39
40             // 4.2 执行SQL语句
41             rs = pstmt.executeQuery();
42
43             //5.处理结果集
44             while(rs.next()) {
45                 int count = rs.getInt(1);

```

```
46         System.out.println("count: " + count);
47         if(count == 0)
48             System.out.println("登录失败!");
49         else
50             System.out.println("登录成功!");
51     }
52
53     } catch (Exception e) {
54         e.printStackTrace();
55     } finally {
56         //6.释放资源
57         if(rs != null) {
58             try {
59                 rs.close();
60             } catch (SQLException e) {
61                 e.printStackTrace();
62             }
63         }
64         if(pstmt != null) {
65             try {
66                 pstmt.close();
67             } catch (SQLException e) {
68                 e.printStackTrace();
69             }
70         }
71         if(conn != null) {
72             try {
73                 conn.close();
74             } catch (SQLException e) {
75                 e.printStackTrace();
76             }
77         }
78     }
79 }
80 }
```

使用3种参数分别运行上述案例，观察结果，发现SQL注入问题已经解决。

预编译大大提升效率的案例，我们学完批处理和事务后再给大家演示。

## 9 Batch

---

**批处理 (Batch)：**一次操作中执行多条SQL语句，相比于一次执行一条SQL语句，效率会提高很多。

**相关方法：**

```
1  package java.sql;
2
3  public interface Statement extends Wrapper, AutoCloseable {
4      // 向Batch中加入SQL语句
5      void addBatch( String sql ) throws SQLException;
6
7      // 执行Batch中的SQL语句,返回数组，元素对应每条sql语句的执行结果
8      int[] executeBatch() throws SQLException;
9
10     // 省略...
11 }
```

**案例展示：**

```
1  package com.briup;
2
3  public class BatchTest {
4      private String driverClass = "com.mysql.cj.jdbc.Driver";
```

```

5     private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6     private String user = "root";
7     private String password = "briup";
8
9     @Test
10    public void test_batch() {
11
12        Connection conn = null;
13        Statement stmt = null;
14
15        try {
16            //1.加载注册驱动
17            Class.forName(driverClass);
18            //2.获取连接对象
19            conn =
20                DriverManager.getConnection(url,user,password);
21            //3.获取Statement对象
22            stmt = conn.createStatement();
23            //4.执行SQL语句
24
25            String sql1 = "drop table if exists t_user";
26            String sql2 = "create table t_user("
27                + "id int primary key,"
28                + "name varchar(50) not null,"
29                + "age int"
30                + ")";
31            String sql3 = "insert into t_user(id,name,age)
32                values(1,'tom1',21)";
33            String sql4 = "insert into t_user(id,name,age)
34                values(2,'tom2',22)";
35            String sql5 = "insert into t_user(id,name,age)
36                values(3,'tom3',23)";
37            String sql6 = "update t_user set name='mary' where
38                id>1";

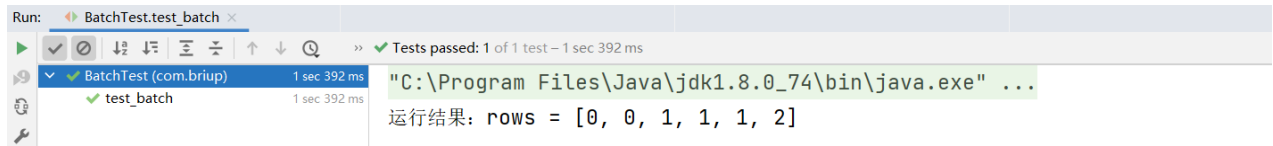
```

```

36         //把要执行的sql语句，添加到批处理中
37         stmt.addBatch(sql1);
38         stmt.addBatch(sql2);
39         stmt.addBatch(sql3);
40         stmt.addBatch(sql4);
41         stmt.addBatch(sql5);
42         stmt.addBatch(sql6);
43
44         //执行批处理中的sql语句，返回每一个语句的结果
45         int[] rows = stmt.executeBatch();
46
47         //5.处理结果
48         System.out.println("运行结果: rows = "+
Arrays.toString(rows));
49
50     } catch (Exception e) {
51         e.printStackTrace();
52     } finally {
53         //6.释放资源
54         if(stmt != null) {
55             try {
56                 stmt.close();
57             } catch (SQLException e) {
58                 e.printStackTrace();
59             }
60         }
61         if(conn != null) {
62             try {
63                 conn.close();
64             } catch (SQLException e) {
65                 e.printStackTrace();
66             }
67         }
68     }
69 }
70 }

```

运行结果：



## 10 Transaction

默认情况下，在JDBC中执行的DML语句，所产生的事务，都是自动提交的。也就是说，每执行一次DML语句，所产生的事务，就会自动提交。

案例展示：

```
1  package com.briup;
2
3  public class TransactionTest {
4      private String driverClass = "com.mysql.cj.jdbc.Driver";
5      private String url = "jdbc:mysql://127.0.0.1:3306/briup";
6      private String user = "root";
7      private String password = "briup";
8
9      @Test
10     public void transaction() {
11
12         Connection conn = null;
13         PreparedStatement pstmt = null;
14
15         try {
16             //1.加载注册驱动
17             Class.forName(driverClass);
18             //2.获取连接对象
```

```

19         conn =
    DriverManager.getConnection(url,user,password);
20
21         //设置事务自动提交为false，也就是需要手动提交
22         conn.setAutoCommit(false);
23
24         //3.获取Statement对象
25         String sql = "update t_user set name=? where id =
    ?";
26
27         pstmt = conn.prepareStatement(sql);
28         //4.执行SQL语句
29         pstmt.setString(1,"lucy");
30         pstmt.setInt(2,1);
31
32         int rows = pstmt.executeUpdate(sql);
33         //手动提交事务
34         conn.commit();
35
36         //5.处理结果
37         System.out.println("运行结果: rows = " + rows);
38     } catch (Exception e) {
39         //出现异常，回滚事务
40         try {
41             conn.rollback();
42         } catch (SQLException e1) {
43             e1.printStackTrace();
44         }
45         e.printStackTrace();
46     } finally {
47         //6.释放资源
48         if(pstmt != null) {
49             try {
50                 pstmt.close();
51             } catch (SQLException e) {
52                 e.printStackTrace();
53             }
54         }
55     }

```

```

53         }
54         if(conn != null) {
55             try {
56                 conn.close();
57             } catch (SQLException e) {
58                 e.printStackTrace();
59             }
60         }
61     }
62 }
63 }

```

注意1，代码中当前连接中的事务设置为了手动提交

注意2，代码中操作完成提交事务，出现异常则回滚事务

注意3，finally语句中的 `conn.close()` 代码，也可以提交事务

另外，如果有需要，也可以在程序中设置回滚点： **(了解即可)**

```

1  //DML
2  Savepoint p1 = conn.setSavepoint("p1");
3  //DML
4  Savepoint p2 = conn.setSavepoint("p2");
5  //DML
6  Savepoint p3 = conn.setSavepoint("p3");
7  //DML
8  Savepoint p4 = conn.setSavepoint("p4");
9
10 conn.rollback(p2);

```

注意，一般情况下，用不到Savepoint相关功能



## PreparedStatement (预处理+批处理+事务) 综合案例

注意：Mysql中要额外开启预编译功能

```
1 //在代码中编写url时需要加上以下参数
2 useServerPrepStmts=true
```

### 案例展示:

使用stmt常规方式，往t\_user表中插入10000条数据，并输出执行时长；

再使用pstmt+批处理+手动提交事务方式，往t\_user表中插入10000条数据，并输出执行时长。

```
1 package com.briup;
2
3 public class PrecompileTest {
4     private String driverClass = "com.mysql.cj.jdbc.Driver";
5     //useServerPrepStmts=true
6     private String url = "jdbc:mysql://127.0.0.1:3306/briup?
    useServerPrepStmts=true";
7     private String user = "root";
8     private String password = "briup";
9
10    private Long startTime;
11    private Long endTime;
12
13    @Before
14    public void before() {
15        startTime = System.currentTimeMillis();
16        System.out.println("程序开始执行, startTime: " +
    startTime);
17    }
18
19    @After
```

```

20     public void after() {
21         endTime = System.currentTimeMillis();
22         System.out.println("程序执行完成, endTime: " +
endTime);
23         System.out.println("耗时: " + (endTime - startTime));
24     }
25
26     @Test
27     public void test_pstmt() {
28
29         Connection conn = null;
30         PreparedStatement ps = null;
31
32         try {
33             //1.加载注册驱动
34             Class.forName(driverClass);
35             //2.获取连接对象
36             conn = DriverManager.getConnection(url, user,
password);
37             conn.setAutoCommit(false);
38
39             //3.获取PS对象
40             //具体每次不同的数据, 可以先用占位符表示, ps会提前把
sql发给数据库预处理, 后面只发送数据
41             String sql = "insert into
t_user(id,name,password,age) values(?,?, 'tom',20)";
42             ps = conn.prepareStatement(sql);
43
44             //4.执行SQL语句
45             for (int i = 1; i <= 10000; i++) {
46                 ps.setInt(1, i);
47                 ps.setString(2, "tom"+i);
48
49                 //添加到批处理
50                 ps.addBatch();
51

```

```

52         //执行批处理
53         if(i % 200 == 0) {
54             ps.executeBatch();
55             conn.commit();
56         }
57     }
58
59     conn.commit();
60
61     //5.处理结果
62     System.out.println("ps执行结束: ");
63 } catch (Exception e) {
64     e.printStackTrace();
65 } finally {
66     //6.释放资源
67     if (ps != null) {
68         try {
69             ps.close();
70         } catch (SQLException e) {
71             e.printStackTrace();
72         }
73     }
74     if (conn != null) {
75         try {
76             conn.close();
77         } catch (SQLException e) {
78             e.printStackTrace();
79         }
80     }
81 }
82
83
84 @Test
85 public void test_stmt() {
86     Connection conn = null;
87     Statement stmt = null;

```

```

88
89         try {
90             //1.加载注册驱动
91             Class.forName(driverClass);
92             //2.获取连接对象
93             conn = DriverManager.getConnection(url, user,
password);
94
95             //3.获取Statement对象
96             stmt = conn.createStatement();
97             // 4. 执行SQL语句
98             String sql = "";
99             for (int i = 1; i <= 10000; i++) {
100                 sql = "insert into
t_user(id,name,password,age) values("+i+", 'tom" + i +
"', 'tom', 20)";
101
102                 stmt.executeUpdate(sql);
103             }
104
105             //5.处理结果
106             System.out.println("stmt执行结束!");
107         } catch (Exception e) {
108             e.printStackTrace();
109         } finally {
110             //6.释放资源
111             if (stmt != null) {
112                 try {
113                     stmt.close();
114                 } catch (SQLException e) {
115                     e.printStackTrace();
116                 }
117             }
118             if (conn != null) {
119                 try {
120                     conn.close();

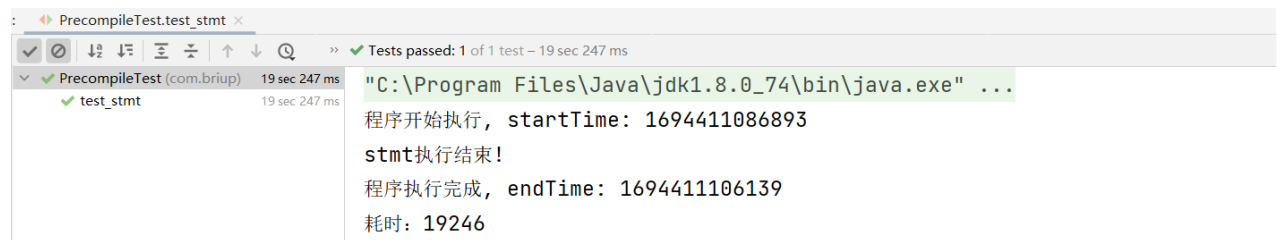
```

```

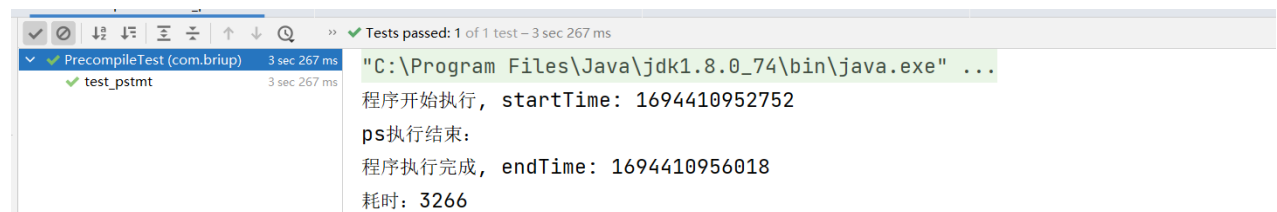
121             } catch (SQLException e) {
122                 e.printStackTrace();
123             }
124         }
125     }
126 }
127 }

```

stmt效果:



预处理效果:



**结论: 同构sql语句的执行, 推荐使用PreparedStatement方式。**

补充内容:

按照JDBC规范的想法, 实际上preparedstatement的出现主要的目的是为了提高SQL语句的性能, 而防止SQL注入和类型转换实际上并不是规范原本的目标。但是实践中这两个功能却受到更多的欢迎。

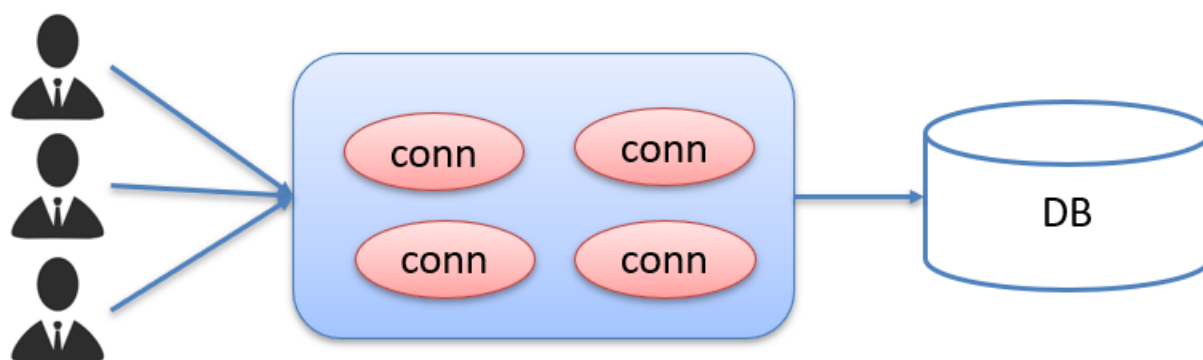
preparedstatment性能是否能提高性能, 实际要具体的看数据库服务器端处理复杂SQL的情况, 对于oracle而言在复杂的SQL上面二者性能差异明显(可以使用报表系统的SQL来测试), 而比较简单的数据库MySQL和PostgreSQL其的性能提升不明显。在这些某些版本的JDBC实现PreparedStatement的实现就是直接使用statment的实现。

## 11 连接池

## 11.1 概述

池(Pool)技术在一定程度上可以明显优化服务器应用程序的性能，提高程序执行效率和降低系统资源开销。

数据库连接是一种关键的、有限的、昂贵的资源，对数据库连接的管理能显著影响到整个应用程序的性能指标，那么如何提高对数据库操作的性能？——数据库连接池正是针对这个问题提出来的。



数据库连接池，在系统初始化时创建一定数量数据库连接对象，需要时直接从池中取出一个空闲对象，用完后并不直接释放掉对象，而是再放到对象池中，以便下一次对象请求可以直接复用。消除了对象创建和销毁所带来的延迟，从而提高系统的性能。

### 优点：

- 资源复用

由于数据库连接得到重用，避免了频繁创建、释放连接引起的大量性能开销；

- 更快的系统响应速度

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，从而缩减了系统整体响应时间。

- 统一的连接管理，避免数据库连接泄漏

根据预先的连接占用超时设定，强制收回被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄漏。

## 常见数据库连接池

- DBCP
- C3P0
- Druid

我们现在使用更多的是Druid，它的性能比其他两个会好一些。

## 11.2 Druid

### Druid（德鲁伊）

- Druid连接池是阿里巴巴开源的数据库连接池项目
- 功能强大，性能优秀，是Java语言最好的数据库连接池之一

### Druid连接池介绍

功能类别	功能	Druid	HikariCP	DBCP	Tomcat-jdbc	C3P0
性能	PSCache	是	否	是	是	是
	LRU	是	否	是	是	是
	SLB负载均衡支持	是	否	否	否	否
稳定性	ExceptionSorter	是	否	否	否	否
扩展	扩展	Filter			JdbcInterceptor	
监控	监控方式	jmx/log/http	jmx/metrics	jmx	jmx	jmx
	支持SQL级监控	是	否	否	否	否
	Spring/Web关联监控	是	否	否	否	否
	诊断支持	LogFilter	否	否	否	否
	连接泄露诊断	logAbandoned	否	否	否	否
安全	SQL防注入	是	无	无	无	无
	支持配置加密	是	否	否	否	否

从上表可以看出，Druid连接池在性能、监控、诊断、安全、扩展性这些方面远远超出其他竞品。

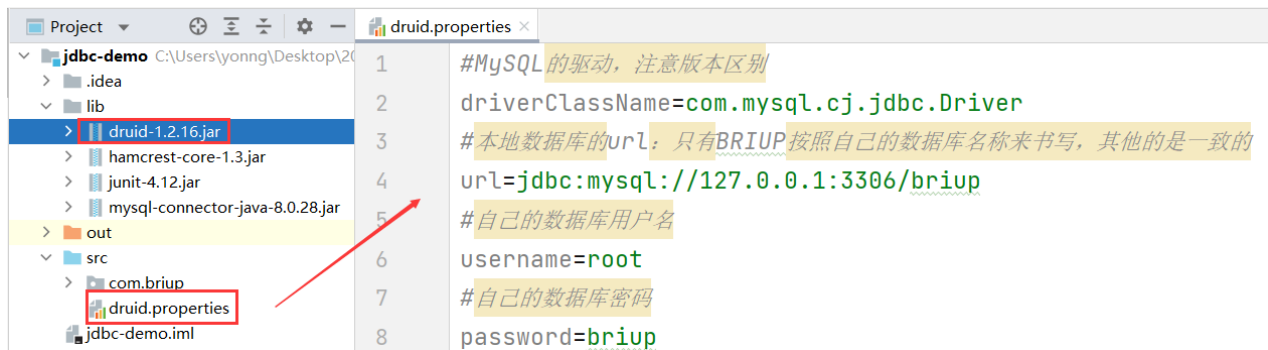
## 11.3 使用

`javax.sql.DataSource` 是Java中定义的一个数据源标准的接口，通过它获取到的数据库连接对象，如果调用 `close()` 方法，不会再关闭连接，而是归还连接到连接池中。

### 第一步：导入jar包

参照之前jar包导入方式。





## 第二步：添加配置文件

在项目的src文件夹下面，创建资源文件：`druid.properties`

配置文件内容如下：

```
1  #MySQL的驱动，注意版本区别
2  driverClassName=com.mysql.cj.jdbc.Driver
3  #本地数据库的url：只有BRIUP按照自己的数据库名称来书写，其他的是一致的
4  url=jdbc:mysql://127.0.0.1:3306/briup
5  #自己的数据库用户名
6  username=root
7  #自己的数据库密码
8  password=briup
9  # 初始化连接数量
10 initialSize=5
11 # 最大连接数
12 maxActive=10
13 # 最大等待时间
14 maxWait=3000
```

注意：位置一定是在src文件夹下面

## 第三步：使用配置文件，创建数据库连接池，并获得连接

```
1  package com.briup;
```

```

2
3 public class DruidTest {
4     //使遍历配置文件druid.properties内容
5     @Test
6     public void test_outProperties() {
7         Properties properties = new Properties();
8         InputStream is =
9         ConnectionTest.class.getClassLoader().getResourceAsStream("druid.properties");
10        try {
11            properties.load(is);
12            properties.forEach((k,v)->System.out.println(k +
13            ": " + v));
14        } catch (IOException e) {
15            e.printStackTrace();
16        }
17
18        //获取druid连接对象
19        @Test
20        public void test_druid() {
21            Connection conn = null;
22            try {
23                //1.加载配置文件
24                Properties properties = new Properties();
25                InputStream is =
26                ConnectionTest.class.getClassLoader().getResourceAsStream("druid.properties");
27                properties.load(is);
28
29                //2.获取连接池对象
30                DataSource dataSource =
31                DruidDataSourceFactory.createDataSource(properties);
32
33                //3.获取连接
34                conn = dataSource.getConnection();

```

```

32         System.out.println("conn: " + conn);
33
34         //后续操作，跟以往相同，省略...
35
36     } catch (Exception e) {
37         e.printStackTrace();
38     } finally {
39         if(conn != null) {
40             try {
41                 conn.close();
42             } catch (SQLException e) {
43                 e.printStackTrace();
44             }
45         }
46     }
47 }
48 }

```

## 12 封装

---

```

1  package com.briup.util;
2
3  public class JdbcUtils {
4
5      private static DataSource dataSource;
6
7      static {
8          try {
9              Properties properties = new Properties();

```

```

10         properties.load(JdbcUtils.class.getClassLoader().getResourceA
sStream("druid.properties"));
11         dataSource =
DruidDataSourceFactory.createDataSource(properties);
12     } catch (Exception e) {
13         e.printStackTrace();
14     }
15 }
16
17     public static Connection getConnection() throws
SQLException {
18         return dataSource.getConnection();
19     }
20
21     public static void close(ResultSet rs, Statement stmt,
Connection conn) {
22         if (rs != null) {
23             try {
24                 rs.close();
25             } catch (SQLException e) {
26                 e.printStackTrace();
27             }
28         }
29
30         if (stmt != null) {
31             try {
32                 stmt.close();
33             } catch (SQLException e) {
34                 e.printStackTrace();
35             }
36         }
37
38         if (conn != null) {
39             try {
40                 conn.close();

```

```
41         } catch (SQLException e) {
42             e.printStackTrace();
43         }
44     }
45 }
46
47 public static void close(Statement stmt, Connection conn)
48 {
49     close(null, stmt, conn);
50 }
```

使用上述工具类，重写实现增删改查功能，体会工具类的好处。