JDBC

第1节　回顾

1.1　表连接

内连接：隐式、显式
隐式：没有 join，使用 where
显式：inner join..on
外连接：左连接和右连接
左连接：left outer join … on
右连接：right outer join … on

1.2　子查询

三种情况：
1)　单行单列：比较运算符：>、<、=
2)　多行单列：使用 in 关键字
3)　多行多列：放在 from 后面，做为一张表再次查询

1.3　备份与还原

备份：mysqldump -u 用户名 -p 密码 数据库 > 文件名
还原：
登录使用数据库
use 数据库
source 文件名

1.4　事务

事务四个特性 ACID：原子性、一致性、隔离性、持久性
在 mysql 中有三条语句：
开启事务：start transaction
提交事务：commit
回滚事务：rollback

设置事务回滚点：savepoint 名字
回到回滚点：rollback to 名字

第2节　学习目标

1)　能够理解 JDBC 的概念
2)　能够使用 DriverManager 类
3)　能够使用 Connection 接口
4)　能够使用 Statement 接口
5)　能够使用 ResultSet 接口
6)　能够说出 SQL 注入原因和解决方案
7)　能够通过 PreparedStatement 完成增、删、改、查
8)　能够完成 PreparedStatement 改造登录案例

第3节　JDBC 入门

3.1　客户端操作 MySQL 数据库的方式：

1) 使用第三方客户端来访问 MySQL：SQLyog、Navicat、SQLWave、MyDB Studio、EMS SQL Manager for MySQL

2) 使用 MySQL 自带的命令行方式

3) 通过 Java 来访问 MySQL 数据库，今天要学习的内容

3.1.1 什么是 JDBC

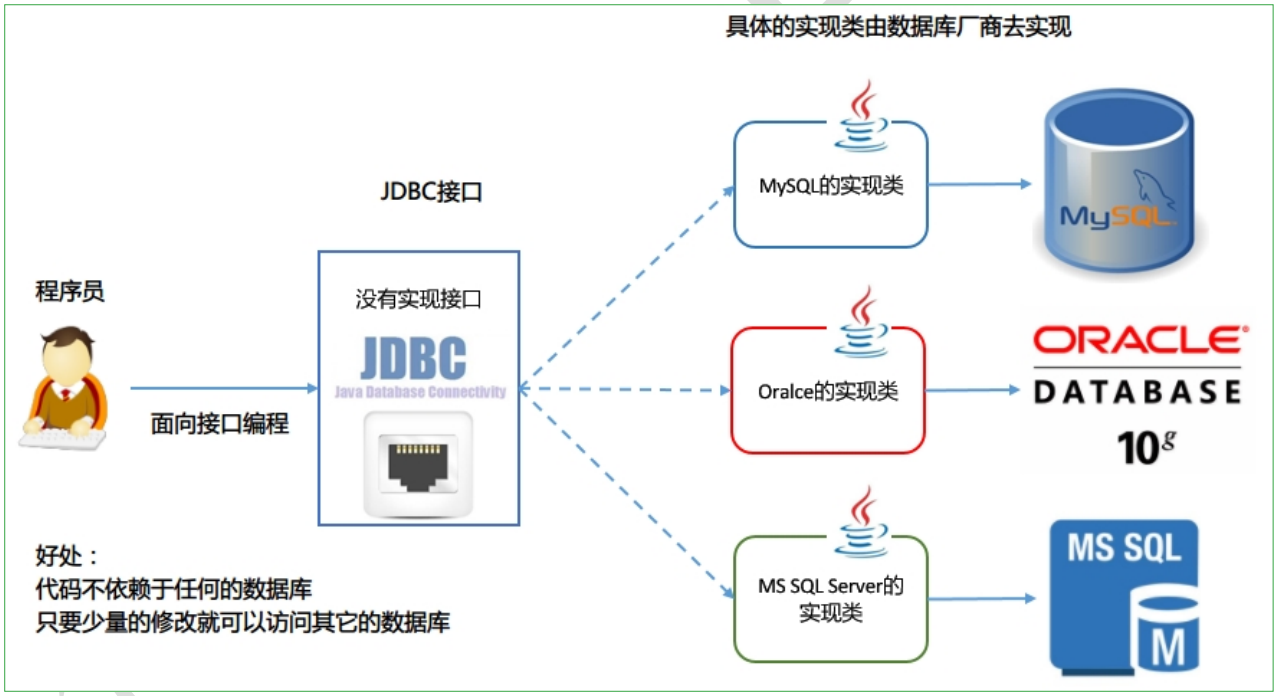
JDBC 规范定义接口，具体的实现由各大数据库厂商来实现。

JDBC 是 Java 访问数据库的标准规范，真正怎么操作数据库还需要具体的实现类，也就是数据库驱动。每个数据库厂商根据自家数据库的通信格式编写好自己数据库的驱动。所以我们只需要会调用 JDBC 接口中的方法即可，数据库驱动由数据库厂商提供。

- 使用 JDBC 的好处：

1) 程序员如果要开发访问数据库的程序，只需要会调用 JDBC 接口中的方法即可，不用关注类是如何实现的。

2) 使用同一套 Java 代码，进行少量的修改就可以访问其他 JDBC 支持的数据库



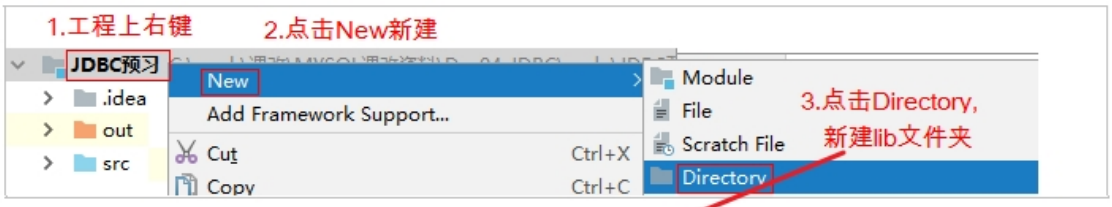
3.1.2 使用 JDBC 开发使用到的包：

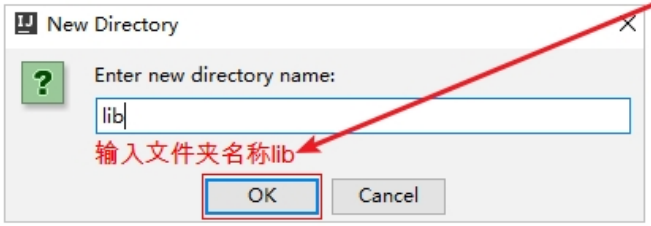
会使用到的包	说明
java.sql	所有与 JDBC 访问数据库相关的接口和类
javax.sql	数据库扩展包，提供数据库额外的功能。如：连接池
数据库的驱动	由各大数据库厂商提供，需要额外去下载，是对 JDBC 接口实现的类

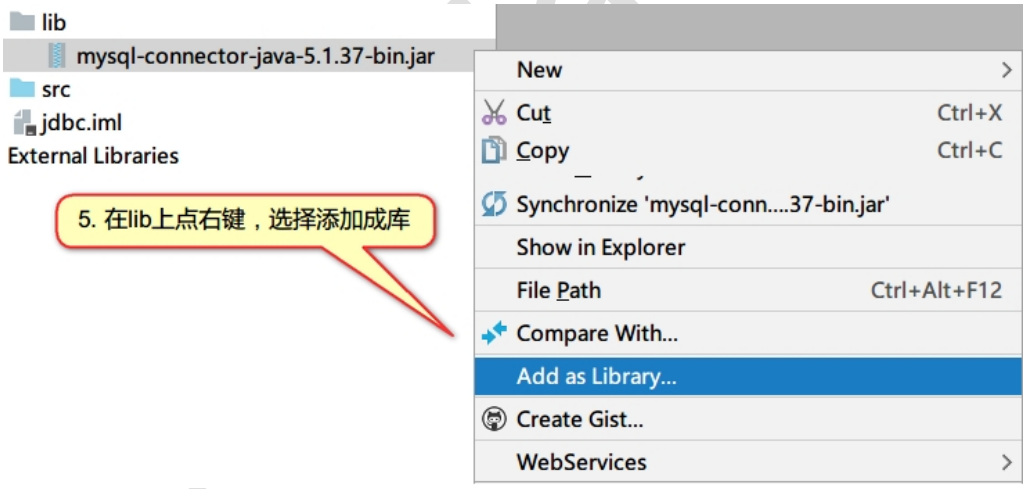
3.2 JDBC 的核心 API

接口或类	作用
DriverManager 类	1) 管理和注册数据库驱动 2) 得到数据库连接对象
Connection 接口	一个连接对象，可用于创建 Statement 和 PreparedStatement 对象
Statement 接口	一个 SQL 语句对象，用于将 SQL 语句发送给数据库服务器。
PreparedStatemen 接口	一个 SQL 语句对象，是 Statement 的子接口
ResultSet 接口	用于封装数据库查询的结果集，返回给客户端 Java 程序

3.3 导入驱动 Jar 包







3.4 加载和注册驱动

加载和注册驱动的方法	描述
Class.forName(数据库驱动实现类)	加载和注册数据库驱动，数据库驱动由 mysql 厂商 "com.mysql.jdbc.Driver"

- 疑问：为什么这样可以注册驱动？

```
public class Demo1 {
    public static void main(String[] args) throws ClassNotFoundException {
```

```
        //抛出类找不到的异常，注册数据库驱动
        Class.forName("com.mysql.jdbc.Driver");
    }
}
```

- com.mysql.jdbc.Driver 源代码：

// Driver 接口，所有数据库厂商必须实现的接口，表示这是一个驱动类。

```
public class Driver implements java.sql.Driver {
    public Driver() throws SQLException {
    }

    static {
        try {
            DriverManager.registerDriver(new Driver());  //注册数据库驱动
        } catch (SQLException var1) {
            throw new RuntimeException("Can't register driver!");
        }
    }
}
```

✧ 注: 从 JDBC3 开始，目前已经普遍使用的版本。可以不用注册驱动而直接使用。Class.forName 这句话可以省略。

第4节 DriverManager 类

4.1 DriverManager 作用：

1) 管理和注册驱动
2) 创建数据库的连接

4.2 类中的方法：

DriverManager 类中的静态方法	描述
Connection getConnection (String url, String user, String password)	通过连接字符串，用户名，密码来得到数据库的连接对象
Connection getConnection (String url, Properties info)	通过连接字符串，属性对象来得到连接对象

4.3 使用 JDBC 连接数据库的四个参数：

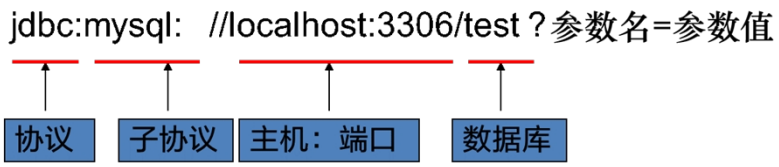
JDBC 连接数据库的四个参数	说明
用户名	登录的用户名
密码	登录的密码
连接字符串 URL	不同的数据库 URL 是不同的，mysql 的写法 jdbc:mysql://localhost:3306/数据库[?参数名=参数值]
驱动类的字符串名	com.mysql.jdbc.Driver

4.4 连接数据库的 URL 地址格式：

协议名:子协议://服务器名或 IP 地址:端口号/数据库名?参数=参数值

4.4.1 MySQL 写法：

- URL用于标识数据库的位置，程序员通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：



4.4.2 MySQL 中可以简写：

前提：必须是本地服务器，端口号是 3306
jdbc:mysql:///数据库名

4.4.3 乱码的处理

如果数据库出现乱码，可以指定参数: ?characterEncoding=utf8，表示让数据库以 UTF-8 编码来处理数据。

```
jdbc:mysql://localhost:3306/数据库?characterEncoding=utf8
```

4.5 案例：得到 MySQL 的数据库连接对象

1) 使用用户名、密码、URL 得到连接对象

```
package com.itheima;

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

/**
 * 得到连接对象
 */
public class Demo2 {
    public static void main(String[] args) throws SQLException {
        String url = "jdbc:mysql://localhost:3306/day24";
        //1)   使用用户名、密码、URL 得到连接对象
        Connection connection = DriverManager.getConnection(url, "root", "root");
        //com.mysql.jdbc.JDBC4Connection@68de145
        System.out.println(connection);
    }
}
```

2) 使用属性文件和 url 得到连接对象

```
package com.itheima;

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

public class Demo3 {
    public static void main(String[] args) throws SQLException {
        //url 连接字符串
        String url = "jdbc:mysql://localhost:3306/day24";
        //属性对象
        Properties info = new Properties();
        //把用户名和密码放在 info 对象中
```

```
        info.setProperty("user","root");
        info.setProperty("password","root");
        Connection connection = DriverManager.getConnection(url, info);
        //com.mysql.jdbc.JDBC4Connection@68de145
        System.out.println(connection);
    }
}
```

第5节 Connection 接口：

5.1 Connection 作用：

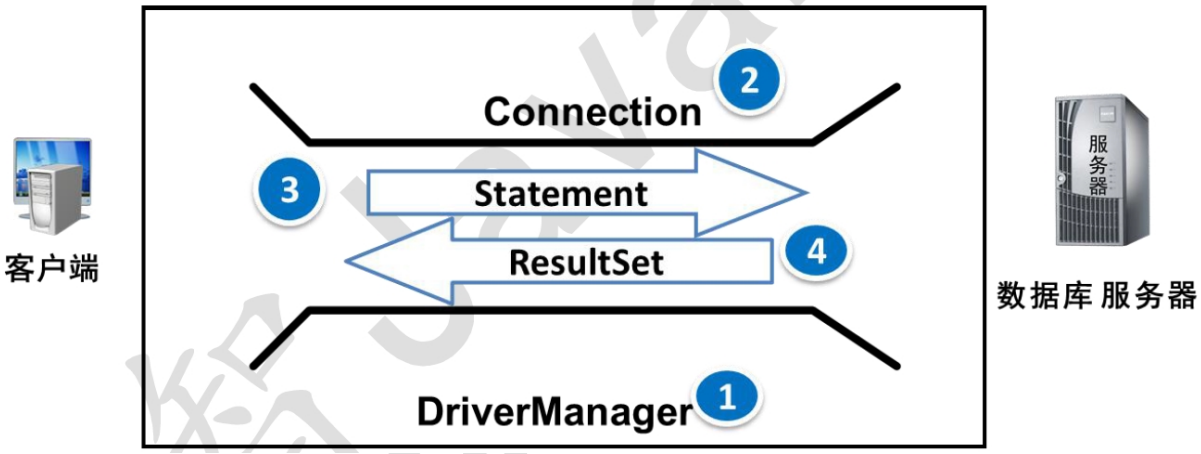
Connection 接口，具体的实现类由数据库的厂商实现，代表一个连接对象。

5.2 Connection 方法：

Connection 接口中的方法	描述
Statement createStatement()	创建一条 SQL 语句对象

第6节 Statement 接口

6.1 JDBC 访问数据库的步骤



1) 注册和加载驱动(可以省略)
2) 获取连接
3) Connection 获取 Statement 对象
4) 使用 Statement 对象执行 SQL 语句
5) 返回结果集
6) 释放资源

6.2 Statement 作用：

代表一条语句对象，用于发送 SQL 语句给服务器，用于执行静态 SQL 语句并返回它所生成结果的对象。

6.3 Statement 中的方法：

Statement 接口中的方法	描述
int executeUpdate(String sql)	用于发送 DML 语句，增删改的操作，insert、update、delete 参数：SQL 语句 返回值：返回对数据库影响的行数
ResultSet executeQuery(String sql)	用于发送 DQL 语句，执行查询的操作。select 参数：SQL 语句 返回值：查询的结果集

6.4 释放资源

1) 需要释放的对象：ResultSet 结果集，Statement 语句，Connection 连接
2) 释放原则：先开的后关，后开的先关。ResultSet → Statement → Connection
3) 放在哪个代码块中：finally 块

6.5 执行 DDL 操作

6.5.1 需求：使用 JDBC 在 MySQL 的数据库中创建一张学生表

Field	Type		Null	Key	Default		Extra
id	int(11)	7B	NO	PRI	*(NULL)*	0K	auto_increment
name	varchar(20)	11B	NO		*(NULL)*	0K	
gender	tinyint(1)	10B	YES		1	1B	
birthday	date	4B	YES		*(NULL)*	0K	

6.5.2 代码：

```
package com.itheima;

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

/**
 * 创建一张学生表
 */
public class Demo4DDL {

    public static void main(String[] args) {
        //1. 创建连接
        Connection conn = null;
        Statement statement = null;
        try {
            conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");
            //2. 通过连接对象得到语句对象
            statement = conn.createStatement();
            //3. 通过语句对象发送 SQL 语句给服务器
            //4. 执行 SQL
            statement.executeUpdate("create table student (id int PRIMARY key auto_increment, " +
                    "name varchar(20) not null, gender boolean, birthday date)");
            //5. 返回影响行数(DDL 没有返回值)
            System.out.println("创建表成功");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        //6. 释放资源
        finally {
            //关闭之前要先判断
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
```

```
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

6.6 执行 DML 操作

- 需求：向学生表中添加 4 条记录，主键是自动增长

id	name	gender	birthday
1	孙悟空	1	1993-03-24
2	白骨精	0	1995-03-24
3	猪八戒	1	1903-03-24
4	嫦娥	0	1993-03-11

- **步骤：**

1) 创建连接对象
2) 创建 Statement 语句对象
3) 执行 SQL 语句：executeUpdate(sql)
4) 返回影响的行数
5) 释放资源

- 代码：

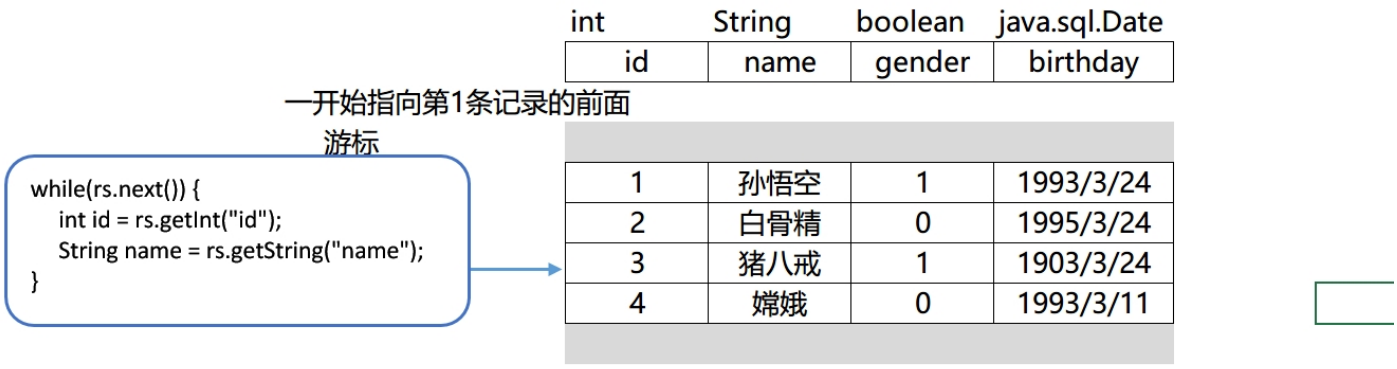
```
package com.itheima;

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

/**
 * 向学生表中添加 4 条记录，主键是自动增长
 */
public class Demo5DML {

    public static void main(String[] args) throws SQLException {
//        1) 创建连接对象
        Connection connection = DriverManager.getConnection("jdbc:mysql:///day24", "root",
"root");
//        2) 创建 Statement 语句对象
        Statement statement = connection.createStatement();
//        3) 执行 SQL 语句：executeUpdate(sql)
        int count = 0;
//        4) 返回影响的行数
        count += statement.executeUpdate("insert into student values(null, '孙悟空', 1, '1993-03-
24')");
        count += statement.executeUpdate("insert into student values(null, '白骨精', 0, '1995-03-
24')");
        count += statement.executeUpdate("insert into student values(null, '猪八戒', 1, '1903-03-
```

```
24')");
        count += statement.executeUpdate("insert into student values(null, '嫦娥', 0, '1993-03-
11')");
        System.out.println("插入了" + count + "条记录");
//        5) 释放资源
        statement.close();
        connection.close();
    }

}
```

6.7 执行 DQL 操作

6.7.1 ResultSet 接口：

- 作用：封装数据库查询的结果集，对结果集进行遍历，取出每一条记录。



boolean next() 游标向下移动1行

判断当前指向的记录是否还有下一条记录，如果返回true，表示还有下一条，否则返回false

取记录有2种方式：

1.通过列名：getInt("id"),getString("name"),getBoolean("gender"),getDate("birthday")

2. 通过列号：getInt(1), getString(2),getBoolean(3),getDate(4)

- 接口中的方法：

ResultSet 接口中的方法	描述
boolean next()	1) 游标向下移动 1 行 2) 返回 boolean 类型，如果还有下一条记录，返回 true，否则返回 false
数据类型 getXxx()	1) 通过字段名，参数是 String 类型。返回不同的类型 2) 通过列号，参数是整数，从 1 开始。返回不同的类型

boolean	**getBoolean**(String columnLabel) 以 Java 编程语言中 boolean 的形式获取此 ResultSet 对象的当前行中指定列的值。
byte	**getByte**(String columnLabel) 以 Java 编程语言中 byte 的形式获取此 ResultSet 对象的当前行中指定列的值。
short	**getShort**(String columnLabel) 以 Java 编程语言中 short 的形式获取此 ResultSet 对象的当前行中指定列的值。
long	**getLong**(String columnLabel) 以 Java 编程语言中 long 的形式获取此 ResultSet 对象的当前行中指定列的值。
float	**getFloat**(String columnLabel) 以 Java 编程语言中 float 的形式获取此 ResultSet 对象的当前行中指定列的值。
double	**getDouble**(String columnLabel) 以 Java 编程语言中 double 的形式获取此 ResultSet 对象的当前行中指定列的值。
String	**getString**(String columnLabel) 以 Java 编程语言中 String 的形式获取此 ResultSet 对象的当前行中指定列的值。

6.7.2 常用数据类型转换表

SQL 类型	Jdbc 对应方法	返回类型
BIT(1) bit(n)	getBoolean()	boolean
TINYINT	getByte()	byte
SMALLINT	getShort()	short
INT	getInt()	int
BIGINT	getLong()	long
CHAR,VARCHAR	getString()	String
Text(Clob)　Blob	getClob getBlob()	Clob　Blob
DATE	getDate()	java.sql.Date 只代表日期
TIME	getTime()	java.sql.Time　只表示时间
TIMESTAMP	getTimestamp()	java.sql.Timestamp　同时有日期和时间

✧ java.sql.Date、Time、Timestamp(时间戳)，三个共同父类是：java.util.Date

6.7.3 需求：确保数据库中有 3 条以上的记录，查询所有的学员信息

- 步骤：

1) 得到连接对象
2) 得到语句对象
3) 执行 SQL 语句得到结果集 ResultSet 对象
4) 循环遍历取出每一条记录
5) 输出的控制台上
6) 释放资源

结果：

```
编号：1, 姓名：孙悟空, 性别：true, 生日：1993-03-24
编号：2, 姓名：白骨精, 性别：false, 生日：1995-03-24
编号：3, 姓名：猪八戒, 性别：true, 生日：1903-03-24
编号：4, 姓名：嫦娥, 性别：false, 生日：1993-03-11
```

```
package com.itheima;

import java.sql.*;
```

```
/**
 * 查询所有的学生信息
 */
public class Demo6DQL {

    public static void main(String[] args) throws SQLException {
        //1) 得到连接对象
        Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/day24","root","root");
        //2) 得到语句对象
        Statement statement = connection.createStatement();
        //3) 执行 SQL 语句得到结果集 ResultSet 对象
        ResultSet rs = statement.executeQuery("select * from student");
        //4) 循环遍历取出每一条记录
        while(rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            boolean gender = rs.getBoolean("gender");
            Date birthday = rs.getDate("birthday");
        //5) 输出的控制台上
            System.out.println("编号：" + id + ", 姓名：" + name + ", 性别：" + gender + ", 生日：" +
birthday);
        }
        //6) 释放资源
        rs.close();
        statement.close();
        connection.close();
    }
}
```

6.7.4 关于 ResultSet 接口中的注意事项：

1) 如果光标在第一行之前，使用 rs.getXX()获取列值，报错：Before start of result set
2) 如果光标在最后一行之后，使用 rs.getXX()获取列值，报错：After end of result set
3) 使用完毕以后要关闭结果集 ResultSet，再关闭 Statement，再关闭 Connection

第7节 数据库工具类 JdbcUtils

- 什么时候自己创建工具类？

如果一个功能经常要用到，我们建议把这个功能做成一个工具类，可以在不同的地方重用。

7.1 需求：

上面写的代码中出现了很多重复的代码，可以把这些公共代码抽取出来。

7.2 创建类 JdbcUtil 包含 3 个方法：

1) 可以把几个字符串定义成常量：用户名，密码，URL，驱动类
2) 得到数据库的连接：getConnection()
3) 关闭所有打开的资源：
   close(Connection conn, Statement stmt)，close(Connection conn, Statement stmt, ResultSet rs)

```
package com.itheima.utils;

import java.sql.*;

/**
```

```
 * 访问数据库的工具类
 */
public class JdbcUtils {

    //可以把几个字符串定义成常量：用户名，密码，URL，驱动类
    private static final String USER = "root";
    private static final String PWD = "root";
    private static final String URL = "jdbc:mysql://localhost:3306/day24";
    private static final String DRIVER= "com.mysql.jdbc.Driver";

    /**
     * 注册驱动
     */
    static {
        try {
            Class.forName(DRIVER);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**
     * 得到数据库的连接
     */
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL,USER,PWD);
    }

    /**
     * 关闭所有打开的资源
     */
    public static void  close(Connection conn, Statement stmt) {
        if (stmt!=null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn!=null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * 关闭所有打开的资源
     */
    public static void close(Connection conn, Statement stmt, ResultSet rs) {
        if (rs!=null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
```

```
            }
        }
        close(conn, stmt);
    }

}
```

7.3 案例：用户登陆

7.3.1 需求：

1) 有一张用户表
2) 添加几条用户记录

```
create table user (
  id int primary key auto_increment,
  name varchar(20),
  password varchar(20)
)

insert into user values (null,'jack','123'),(null,'rose','456');

-- 登录, SQL 中大小写不敏感
select * from user where name='JACK' and password='123';

-- 登录失败
select * from user where name='JACK' and password='333';
```

3) 使用 Statement 字符串拼接的方式实现用户的登录，用户在控制台上输入用户名和密码。

7.3.2 步骤：

1) 得到用户从控制台上输入的用户名和密码来查询数据库
2) 写一个登录的方法
   a) 通过工具类得到连接
   b) 创建语句对象，使用拼接字符串的方式生成 SQL 语句
   c) 查询数据库，如果有记录则表示登录成功，否则登录失败
   d) 释放资源

```
package com.itheima;

import com.itheima.utils.JdbcUtils;

import javax.xml.transform.Result;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class Demo7Login {

    //从控制台上输入的用户名和密码
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入用户名：");
        String name = sc.nextLine();
        System.out.println("请输入密码：");
        String password = sc.nextLine();
        login(name, password);
```

```
    }

    /**
     * 登录的方法
     */
    public static void login(String name, String password) {
        //a) 通过工具类得到连接
        Connection connection = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            connection = JdbcUtils.getConnection();
            //b) 创建语句对象，使用拼接字符串的方式生成 SQL 语句
            statement = connection.createStatement();
            //c) 查询数据库，如果有记录则表示登录成功，否则登录失败
            String sql = "select * from user where name='" + name + "' and password='" + password
+ "'";
            System.out.println(sql);
            rs = statement.executeQuery(sql);
            if (rs.next()) {
                System.out.println("登录成功，欢迎您：" + name);
            } else {
                System.out.println("登录失败");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            //d) 释放资源
            JdbcUtils.close(connection, statement, rs);
        }

    }
}
```

7.3.3 SQL 注入问题

- 当我们输入以下密码，我们发现我们账号和密码都不对竟然登录成功了

```
请输入用户名：
newboy
请输入密码：
a' or '1'='1
select * from user where name='newboy' and password='a' or '1'='1'
登录成功，欢迎您：newboy
```

- 问题分析：

```
select * from user where name='newboy' and password='a' or '1'='1'
name='newboy' and password='a' 为假
'1'='1' 真
相当于
select * from user where true; 查询了所有记录
```

我们让用户输入的密码和 SQL 语句进行字符串拼接。用户输入的内容作为了 SQL 语句语法的一部分，改变了原有 SQL 真正的意义，以上问题称为 SQL 注入。要解决 SQL 注入就不能让用户输入的密码和我们的 SQL 语句进行简单的字符串拼接。

第8节 PreparedStatement 接口

8.1 继承结构与作用：

java.sql
接口 PreparedStatement

所有超级接口：
Statement, Wrapper

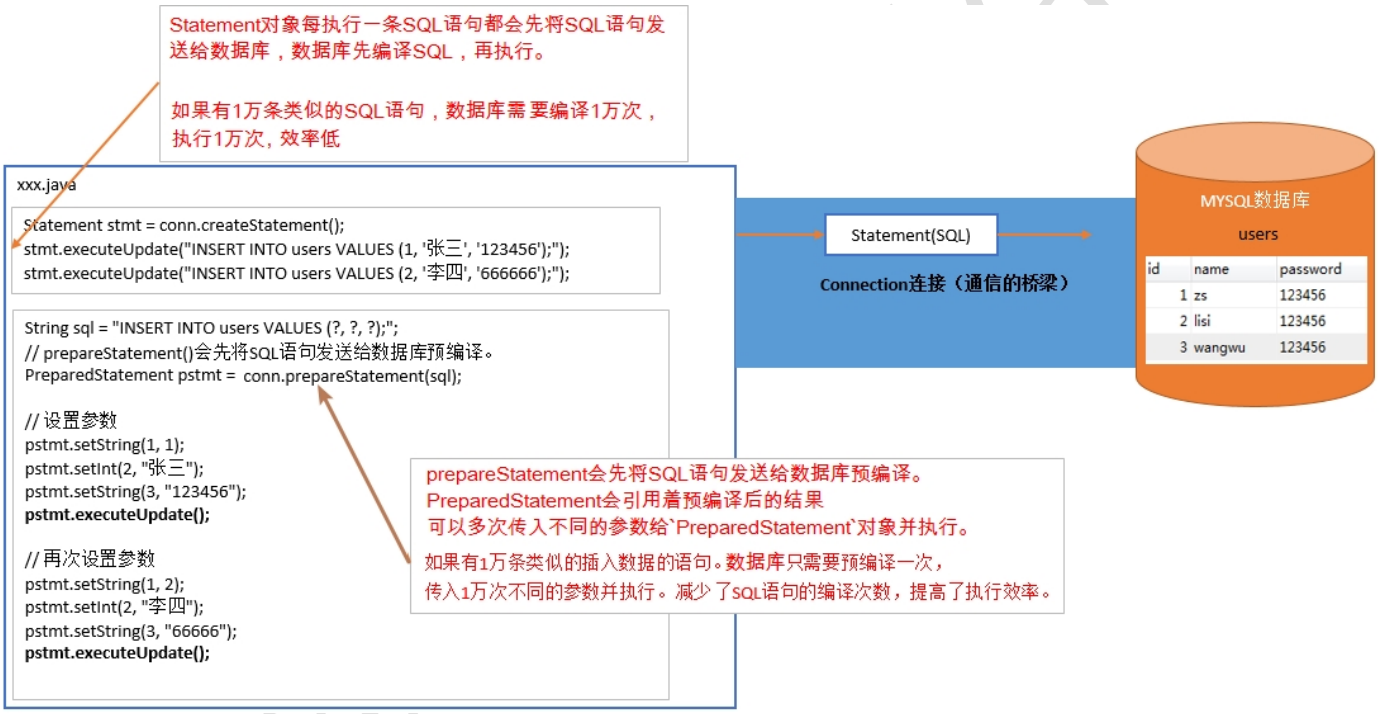
所有已知子接口：
CallableStatement

```
public interface PreparedStatement
extends Statement
```

表示预编译的 SQL 语句的对象。

PreparedStatement 是 Statement 接口的子接口，继承于父接口中所有的方法。它是一个预编译的 SQL 语句

8.2 PreparedSatement 的执行原理



1) 因为有预先编译的功能，提高 SQL 的执行效率。
2) 可以有效的防止 SQL 注入的问题，安全性更高。

8.3 Connection 创建 PreparedStatement 对象

Connection 接口中的方法	描述
PreparedStatement prepareStatement(String sql)	指定预编译的 SQL 语句，SQL 语句中使用占位符? 创建一个语句对象

8.4 PreparedStatement 接口中的方法：

PreparedStatement 接口中的方法	描述
int executeUpdate()	执行 DML，增删改的操作，返回影响的行数。
ResultSet executeQuery()	执行 DQL，查询的操作，返回结果集

8.5 PreparedSatement 的好处

1. prepareStatement()会先将 SQL 语句发送给数据库预编译。PreparedStatement 会引用着预编译后的结果。可以多次传入不同的参数给 PreparedStatement 对象并执行。减少 SQL 编译次数，提高效率。
2. 安全性更高，没有 SQL 注入的隐患。
3. 提高了程序的可读性

8.6 使用 PreparedStatement 的步骤：

1) 编写 SQL 语句，未知内容使用?占位："SELECT * FROM user WHERE name=? AND password=?";
2) 获得 PreparedStatement 对象
3) 设置实际参数：setXxx(占位符的位置，真实的值)
4) 执行参数化 SQL 语句
5) 关闭资源

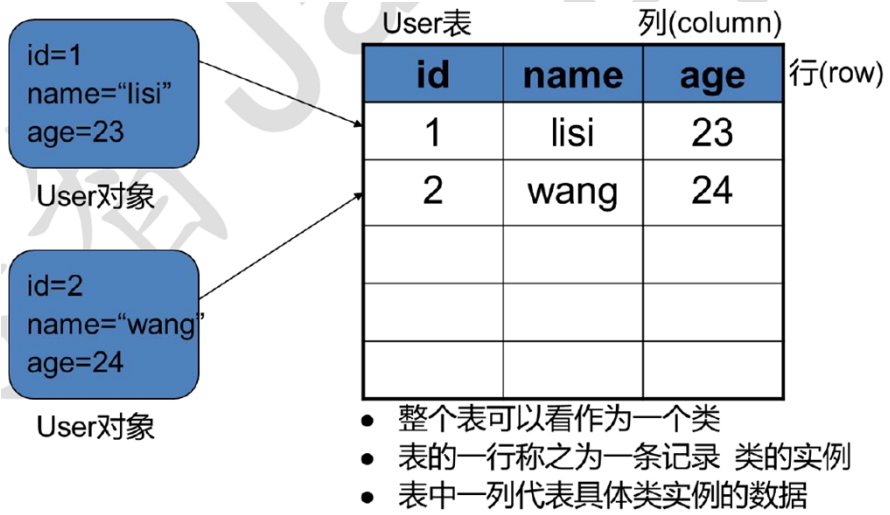
PreparedStatement 中设置参数的方法	描述
void setDouble(int parameterIndex, double x)	将指定参数设置为给定 Java double 值。
void setFloat(int parameterIndex, float x)	将指定参数设置为给定 Java REAL 值。
void setInt(int parameterIndex, int x)	将指定参数设置为给定 Java int 值。
void setLong(int parameterIndex, long x)	将指定参数设置为给定 Java long 值。
void setObject(int parameterIndex, Object x)	使用给定对象设置指定参数的值。
void setString(int parameterIndex, String x)	将指定参数设置为给定 Java String 值。

- 使用 PreparedStatement 改写上面的登录程序，看有没有 SQL 注入的情况

```
package com.itheima;

import com.itheima.utils.JdbcUtils;

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

/**
 * 使用 PreparedStatement
 */
public class Demo8Login {
    //从控制台上输入的用户名和密码
    public static void main(String[] args) throws SQLException {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入用户名：");
        String name = sc.nextLine();
        System.out.println("请输入密码：");
        String password = sc.nextLine();
        login(name, password);
    }

    /**
     * 登录的方法
     * @param name
```

```
     * @param password
     */
    private static void login(String name, String password) throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        //写成登录 SQL 语句，没有单引号
        String sql = "select * from user where name=? and password=?";
        //得到语句对象
        PreparedStatement ps = connection.prepareStatement(sql);
        //设置参数
        ps.setString(1, name);
        ps.setString(2,password);
        ResultSet resultSet = ps.executeQuery();
        if (resultSet.next()) {
            System.out.println("登录成功: " + name);
        }
        else {
            System.out.println("登录失败");
        }
        //释放资源,子接口直接给父接口
        JdbcUtils.close(connection,ps,resultSet);
    }
}
```

8.7 表与类的关系



8.7.1 案例：使用 PreparedStatement 查询一条数据，封装成一个学生 Student 对象

```
 package com.itheima;

import com.itheima.entity.Student;
import com.itheima.utils.JdbcUtils;

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

public class Demo9Student {

    public static void main(String[] args) throws SQLException {
        //创建学生对象
        Student student = new Student();
```

```
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("select * from student where id=?");
        //设置参数
        ps.setInt(1,2);
        ResultSet resultSet = ps.executeQuery();
        if (resultSet.next()) {
            //封装成一个学生对象
            student.setId(resultSet.getInt("id"));
            student.setName(resultSet.getString("name"));
            student.setGender(resultSet.getBoolean("gender"));
            student.setBirthday(resultSet.getDate("birthday"));
        }
        //释放资源
        JdbcUtils.close(connection,ps,resultSet);

        //可以数据
        System.out.println(student);
    }
}
```

8.7.2 案例：将多条记录封装成集合 List<Student>，集合中每个元素是一个 JavaBean 实体类

- 需求： 查询所有的学生类，封装成 List<Student>返回
- 代码：

```
Student{id=1, name='孙悟空', gender=true, birthday=1993-03-24}
Student{id=2, name='白骨精', gender=false, birthday=1995-03-24}
Student{id=3, name='猪八戒', gender=true, birthday=1903-03-24}
Student{id=4, name='嫦娥', gender=false, birthday=1993-03-11}
```

```
package com.itheima;

import com.itheima.entity.Student;
import com.itheima.utils.JdbcUtils;

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

public class Demo10List {
    public static void main(String[] args) throws SQLException {
        //创建一个集合
        List<Student> students = new ArrayList<>();
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("select * from student");
        //没有参数替换
        ResultSet resultSet = ps.executeQuery();
        while(resultSet.next()) {
            //每次循环是一个学生对象
            Student student = new Student();
            //封装成一个学生对象
            student.setId(resultSet.getInt("id"));
```

```
            student.setName(resultSet.getString("name"));
            student.setGender(resultSet.getBoolean("gender"));
            student.setBirthday(resultSet.getDate("birthday"));
            //把数据放到集合中
            students.add(student);
        }
        //关闭连接
        JdbcUtils.close(connection,ps,resultSet);
        //使用数据
        for (Student stu: students) {
            System.out.println(stu);
        }
    }
}
```

8.8 PreparedStatement 执行 DML 操作

```
  package com.itheima;

import com.itheima.utils.JdbcUtils;

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

public class Demo11DML {
    public static void main(String[] args) throws SQLException {
        //insert();
        //update();
        delete();
    }

    //插入记录
    private static void insert() throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("insert into student
values(null,?,?,?)");
        ps.setString(1,"小白龙");
        ps.setBoolean(2, true);
        ps.setDate(3,java.sql.Date.valueOf("1999-11-11"));
        int row = ps.executeUpdate();
        System.out.println("插入了" + row + "条记录");
        JdbcUtils.close(connection,ps);
    }

    //更新记录：换名字和生日
    private static void update() throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("update student set name=?, birthday=?
where id=?");
        ps.setString(1,"黑熊怪");
        ps.setDate(2,java.sql.Date.valueOf("1999-03-23"));
        ps.setInt(3,5);
        int row = ps.executeUpdate();
        System.out.println("更新" + row + "条记录");
        JdbcUtils.close(connection,ps);
    }
```

```
    //删除记录：删除第 5 条记录
    private static void delete() throws SQLException {
        Connection connection = JdbcUtils.getConnection();
        PreparedStatement ps = connection.prepareStatement("delete from student where id=?");
        ps.setInt(1,5);
        int row = ps.executeUpdate();
        System.out.println("删除了" + row + "条记录");
        JdbcUtils.close(connection,ps);
    }
}
```

第9节　JDBC 事务的处理

之前我们是使用 MySQL 的命令来操作事务。接下来我们使用 JDBC 来操作银行转账的事务。

9.1　准备数据

```
CREATE TABLE account (
    id INT PRIMARY KEY AUTO_INCREMENT,
    NAME VARCHAR(10),
    balance DOUBLE
);
-- 添加数据
INSERT INTO account (NAME, balance) VALUES ('Jack', 1000), ('Rose', 1000);
```

9.2　API 介绍

Connection 接口中与事务有关的方法	说明
void setAutoCommit(boolean autoCommit)	参数是 true 或 false 如果设置为 false，表示关闭自动提交，相当于开启事务
void commit()	提交事务
void rollback()	回滚事务

9.3　开发步骤

1) 获取连接
2) 开启事务
3) 获取到 PreparedStatement
4) 使用 PreparedStatement 执行两次更新操作
5) 正常情况下提交事务
6) 出现异常回滚事务
7) 最后关闭资源

- 案例代码

```
package com.itheima;

import com.itheima.utils.JdbcUtils;

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

public class Demo12Transaction {

    //没有异常，提交事务，出现异常回滚事务
    public static void main(String[] args) {
```

```
        //1) 注册驱动
        Connection connection = null;
        PreparedStatement ps = null;
        try {
        //2) 获取连接
            connection = JdbcUtils.getConnection();
        //3) 开启事务
            connection.setAutoCommit(false);
        //4) 获取到 PreparedStatement
            //从 jack 扣钱
            ps = connection.prepareStatement("update account set balance = balance - ? where
name=?");
            ps.setInt(1, 500);
            ps.setString(2,"Jack");
            ps.executeUpdate();
            //出现异常
            System.out.println(100 / 0);
            //给 rose 加钱
            ps = connection.prepareStatement("update account set balance = balance + ? where
name=?");
            ps.setInt(1, 500);
            ps.setString(2,"Rose");
            ps.executeUpdate();
            //提交事务
            connection.commit();
            System.out.println("转账成功");
        } catch (Exception e) {
            e.printStackTrace();
            try {
                //事务的回滚
                connection.rollback();
            } catch (SQLException e1) {
                    e1.printStackTrace();
            }
            System.out.println("转账失败");
        }
        finally {
        //7) 关闭资源
            JdbcUtils.close(connection,ps);
        }

    }
}
```