# JDBC概述

## 简介

JDBC:java database connectivity SUN公司提供的一套操作数据库的**标准规范**。

JDBC与数据库驱动的关系：接口与实现的关系。

|  |
| --- |
|  |

## JDBC规范

掌握**四个核心对象**：

* DriverManager:用于注册驱动
* Connection: 表示与数据库创建的连接
* Statement: 操作数据库sql语句的对象
* ResultSet: 结果集或一张虚拟表

## JDBC的准备工作：

JDBC规范(接口)在哪里：

JDK中：

java.sql.\*;

javax.sql.\*;

数据库厂商提供的驱动：jar文件



# 案例:JDBC程序

实现查询数据库中的数据显示在java的控制台中

## 第一步：初始化数据库数据

|  |
| --- |
| create database test1;  use test1;  create table user(  id int primary key auto\_increment,  name varchar(40),  password varchar(40),  email varchar(60),  birthday date  )character set utf8 collate utf8\_general\_ci;  insert into user(name,password,email,birthday)  values('zhangsan','123456','zs@sina.com','1980-12-04');  insert into user(name,password,email,birthday)  values('lisi','123456','lisi@sina.com','1981-12-04');  insert into user(name,password,email,birthday)  values('wangwu','123456','wangwu@sina.com','1979-12-04'); |
|  |

## 第二步：创建项目写JDBC代码

### 先导入MySQL数据驱动

|  |
| --- |
|  |
|  |

### 代码

|  |
| --- |
|  |
| 实现JDBC操作  1、注册驱动  2、创建连接  3、得到执行sql语句的Statement对象  4、执行sql语句，并返回结果  5、处理结果  6关闭资源 |
|  |

# JDBC常用的类和接口详解

## DriverMnager

java.sql.Drivermanager类 : 注册驱动和创建连接

### a、注册驱动

DriverManager.registerDriver(new com.mysql.jdbc.Driver());**不建议使用**

原因有2个：

> 导致驱动被注册2次，静态代码块里又注册了一次。

|  |
| --- |
|  |

> 强烈依赖数据库的驱动jar

**解决办法**：

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

### b.与数据库建立连接

查看API

|  |
| --- |
|  |

**第一种方试：**

|  |
| --- |
| static Connection getConnection(String url, String user, String password)  试图建立到给定数据库 URL 的连接。  getConnection("jdbc:mysql://localhost:3306/test1", "root", "123456"); |
| **URL:SUN公司与数据库厂商之间的一种协议。**  jdbc:mysql://localhost:3306/day06  协议 子协议 IP :端口号 数据库名  mysql: jdbc:mysql://localhost:3306/test1 或者 jdbc:mysql:///test1（默认本机连接） |

**第二种方式：**

|  |
| --- |
| Properties info = new Properties();//要参考数据库文档  info.setProperty("user", "root");  info.setProperty("password","root");  getConnection(String url, Properties info) |

**第二种方式**

|  |
| --- |
| getConnection(String url)  DriverManager.getConnection("jdbc:mysql://localhost:3306/test1?user=root&password=123456"); |

## Connection接口

**接口的实现在数据库驱动中。所有与数据库交互都是基于连接对象的。**

## Statement接口

**操作sql语句，并返回相应结果的对象**

**接口的实现在数据库驱动中。用于执行静态 SQL 语句并返回它所生成结果的对象。**

* ResultSet executeQuery(String sql) 根据查询语句返回结果集。**只能执行select语句。**
* int executeUpdate(String sql) 根据执行的DML（**insert update delete**）语句，**返回受影响的行数**。
* boolean execute(String sql) 此方法可以执行**任意sql**语句。返回boolean值，表示是否返回ResultSet结果集。仅当执行select语句，且有返回结果时返回true, 其它语句都返回false;

## ResultSet接口:

结果集（客户端存表数据的对象）

### 封装结果集

提供一个游标，默认游标指向结果集第一行之前。

调用一次next()，游标向下移动一行。

提供一些get方法。

### 封装数据的方法

Object getObject(int columnIndex); 根据序号取值，索引从1开始

Object getObject(String ColomnName); 根据列名取值。

|  |
| --- |
|  |

### java的数据类型与数据库中的类型的关系

byte tityint

short smallint

int int

long bigint

float float

double double

String char varchar

Date date

### 接口的方法

boolean next() 将光标从当前位置向下移动一行

int getInt(int colIndex) 以int形式获取ResultSet结果集当前行指定列号值

int getInt(String colLabel) 以int形式获取ResultSet结果集当前行指定列名值

float getFloat(int colIndex) 以float形式获取ResultSet结果集当前行指定列号值

float getFloat(String colLabel) 以float形式获取ResultSet结果集当前行指定列名值

String getString(int colIndex) 以String 形式获取ResultSet结果集当前行指定列号值

String getString(String colLabel) 以String形式获取ResultSet结果集当前行指定列名值

Date getDate(int columnIndex);

Date getDate(String columnName);

void close() 关闭ResultSet 对象

### 可移动游标的方法

boolean next() 将光标从当前位置向前移一行。

boolean previous() 将光标移动到此 ResultSet 对象的上一行。

boolean absolute(int row) 参数是当前行的索引，从1开始 根据行的索引定位移动的指定索引行。

void afterLast() 将光标移动到末尾，正好位于最后一行之后。

void beforeFirst() 将光标移动到开头，正好位于第一行之前。

### 释放资源

资源有限，要正确关闭

|  |
| --- |
| **public** **class** Demo04 {  **public** **static** **void** main(String[] args) {  Connection conn = **null**;  Statement statement = **null**;  ResultSet resultSet = **null**;  **try** {  //1、注册驱动  Class.*forName*("com.mysql.jdbc.Driver");//使用了这个类就加载了  //2、创建连接  conn = DriverManager.*getConnection*("jdbc:mysql://localhost/test1?user=root&password=123456");    //3、得到执行sql语句的Statement对象  statement = conn.createStatement();  String sql = "select \* from user;";    //4、执行sql语句，并返回结果  resultSet = statement.executeQuery(sql);  //5、处理结果  **while**(resultSet.next()){  System.***out***.println(resultSet.getObject(1));  System.***out***.println(resultSet.getObject(2));  System.***out***.println(resultSet.getObject(3));  System.***out***.println(resultSet.getObject(4));  System.***out***.println(resultSet.getObject(5));  System.***out***.println("======================");  }  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally** {  //6关闭资源  **if**(resultSet != **null**){  **try** {  resultSet.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }    **if**(statement != **null**){  **try** {  statement.close();  } **catch** (SQLException e) {  e.printStackTrace();  }    **if**(conn != **null**){  **try** {  conn.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  }  } |

# JDBC的CURD操作

## CURD 增删改查

C:Create

U:Update

R:Read

D:Delete

## 抽取工具类

* 封装的思想

|  |
| --- |
|  |

## 插入数据

|  |
| --- |
| @Test  **public** **void** test1(){  //增  Connection conn = **null**;//创建连接  PreparedStatement stmt = **null**;//预处理  **try** {  conn = DBUtils.*getConnection*();  stmt = conn.prepareStatement("INSERT INTO user VALUES(?,?,?,?,?)");  //设置参数  stmt.setInt(1, 6);  stmt.setString(2, "tony");  stmt.setString(3, "333");  stmt.setString(4, "tony@163.com");  //stmt.setString(5, "2015-09-11");  stmt.setDate(5, **new** java.sql.Date(System.*currentTimeMillis*()));  **int** row = stmt.executeUpdate();  System.***out***.println("影响的行数：" + row);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally** {  DBUtils.*closeAll*(conn, stmt, **null**);  }  } |

## 更改数据

|  |
| --- |
| @Test  **public** **void** test2(){  //增  Connection conn = **null**;//创建连接  PreparedStatement stmt = **null**;//预处理  **try** {  conn = DBUtils.*getConnection*();  stmt = conn.prepareStatement("UPDATE user SET password = ? , email = ? WHERE id = ?");  //设置参数  stmt.setString(1, "123456");  stmt.setString(2, "lily@qq.com");  stmt.setInt(3, 5);  **int** row = stmt.executeUpdate();  System.***out***.println("影响的行数：" + row);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally** {  DBUtils.*closeAll*(conn, stmt, **null**);  }  } |

## 删除数据

|  |
| --- |
| @Test  **public** **void** test3(){  //增  Connection conn = **null**;//创建连接  PreparedStatement stmt = **null**;//预处理  **try** {  conn = DBUtils.*getConnection*();  stmt = conn.prepareStatement("delete from user WHERE id = ?");  //设置参数  stmt.setInt(1, 5);    **int** row = stmt.executeUpdate();  System.***out***.println("影响的行数：" + row);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally** {  DBUtils.*closeAll*(conn, stmt, **null**);  }  } |

## 查询数据

|  |
| --- |
| @Test  **public** **void** test4(){  Connection conn = **null**;//创建连接  Statement stmt = **null**;//预处理  ResultSet resultSet = **null**;//结果集  **try** {  conn = DBUtils.*getConnection*();  stmt = conn.createStatement();  resultSet = stmt.executeQuery("select \* from user;");  List<User> list = **new** ArrayList<User>();  //5、处理结果  **while**(resultSet.next()){  User user = **new** User();  user.setId(resultSet.getInt(1));  user.setName(resultSet.getString(2));  user.setPassword(resultSet.getString(3));  user.setEmail(resultSet.getString(4));  //java.sql.Date继承java.util.Date  user.setDate(resultSet.getDate(5));  list.add(user);  }  //打印集合  System.***out***.println(list);    } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }**finally** {  DBUtils.*closeAll*(conn, stmt, **null**);  }  } |

# 用户登录案例

|  |
| --- |
|  |
|  |

# SQL注入问题

|  |
| --- |
|  |
|  |

**preparedStatement：预编译对象**， 是Statement对象的子类。

特点：

性能要高

会把sql语句先编译，解决SQL注入问题