# 1.JDBC概念

JDBC技术规定了四种Java数据库操作的形式

1. JDBC-ODBC桥接技术(100%不用)，在Windows中有ODBC技术,指的是开放数据库连接,由微软提供的数据库连接应用,而JAVA简介利用JDBC技术间接操作ODBC技术,从而实现数据库的连接,流程:程序 -> JDBC -> ODBC ->数据库,性能不好,支持的版本最新的
2. JDBC直接连接:有不同的数据库生产商提供指定的数据库连接驱动程序,由于是JDBC直接操作数据库,所以是性能是最好的,但是支持的JDBC版本不是最新的
3. JDBC网络连接:使用专门的数据库的网络连接指定进行指定主机的数据库操作,此种方式最多
4. 模拟指定数据库的通讯协议自己编写数据库操作

Java连接任何数据库性能都是很高的,但是只有一个数据库性能是最差的,是SQLserver

# 2.数据库操作了介绍

所有类简介:

一个类:DriverManager类

四个接口:Connection Statement ResultSet PreparedStatement

操作流程:

1. 加载数据库的驱动程序(向容器加载)
2. 进行数据库连接(通过DriverManager类完成,Connection表示连接)
3. 进行数据的CRUD(Statement,PreparedStatement,ResultSet)
4. 关闭数据库操作以及连接(直接关闭连接就够了)

**DriverManager类中**:

取得数据库连接的对象: public static Connection getConnection(String url, String user,String password) throws SQLException

**Connection接口中**

数据库关闭: public void close() throws SQLException

创建Statement接口对象:public Statement createStatement() throws SQLException

创建PreparedStatement 接口对象:PreparedStatement prepareStatement(String sql)

创建CallableStatement 接口对象:CallableStatement prepareCall(String sql)

**Statement接口中:**

**这个接口用于执行静态SQL语句**

**数据更新:** public int executeUpdate(String sql) throws SQLException //返回更新的行数

**数据查询:**ResultSet executeQuery(String sql) throws SQLException //返回的类型和Set无关

**ResultSet接口中:(注意基本不可能使用这个接口,用preparedStatement接口)**

**向下移动指针,并判断是否有数据行:** public boolean next() throws SQLException

注意:移动完指针后就可以取出数据了

**取出数据列的内容**:getInt(字段名),getDouble(字段名),getString(字段名),getDate(字段名), getObject(int columnmIndex)

注意:getXXX方法被重载了,有两种参数,第一:字段名,不区分大小写第二: 索引(从1开始),不过建议使用字段名

范例:基本数据库连接和关闭

|  |
| --- |
| package com.company;  import java.sql.\*;  public class test {  private static final String DBDRIVER="com.mysql.jdbc.Driver";  private static final String DBURL = "jdbc:mysql://118.89.54.105:3306/test";  private static final String USER = "lw";  private static final String PASSWORD = "linwei";  public static void main(String args[])throws Exception{  //第一步:加载数据库驱动程序,此时不需要实例化,因为会有容器自己管理  Class.forName(DBDRIVER);  //第二步:连接数据库  Connection conn = DriverManager.getConnection(DBURL, USER,PASSWORD);  System.out.print(conn);  //第四部:关闭数据库  conn.close();  }  } |

范例:数据库更新

|  |
| --- |
| public class test {  private static final String DBDRIVER="com.mysql.jdbc.Driver";  private static final String DBURL = "jdbc:mysql://118.89.54.105:3306/test";  private static final String USER = "lw";  private static final String PASSWORD = "linwei";  public static void main(String args[])throws Exception{  //第一步:加载数据库驱动程序,此时不需要实例化,因为会有容器自己管理  Class.forName(DBDRIVER);  //第二步:连接数据库  Connection conn = DriverManager.getConnection(DBURL, USER,PASSWORD);  //第三步:数据库操作  //插入操作  Statement stmt = conn.createStatement();  String sql = "insert into student(name,score)values('小明',34.5)";  int len = stmt.executeUpdate(sql);  System.out.print("更新了:"+len + "行");  //修改操作  sql = "update student set name='xiaoming' where name='小明'";  stmt.executeUpdate(sql);  //数据查询  sql = "select name,score from student";  ResultSet rs = stmt.executeQuery(sql);  while(rs.next()){  String name = rs.getString("name");  double score = rs.getDouble("score");  System.out.println("姓名:" + name + ",成绩:" + score);  }  //第四部:关闭资源,后打开的先关闭  stmt.close();//  conn.close();  }  } |

Connection

客户端

getConnection

MySQL连接实现

Oracle连接实现

DriverManager

JJDBC,在驱动数据库连接对象时,采用的是工厂设计模式,而DriverManager是一个工厂类,客户端调用时会完全隐藏具体的实现子类

# 3.PreparedStatement接口

虽然JDBC里面有Statement接口,但是从实际来讲,Statement接口存在有严重的操作缺陷，不会在工作中使用.

范例:以数据增加为例来发现Statement的问题

|  |
| --- |
| public class test {  private static final String DBDRIVER="com.mysql.jdbc.Driver";  private static final String DBURL = "jdbc:mysql://118.89.54.105:3306/test";  private static final String USER = "lw";  private static final String PASSWORD = "linwei";  public static void main(String args[])throws Exception{  String name = "Mr'SMITH";  double score = 34.5;  Class.forName(DBDRIVER);  Connection conn = DriverManager.getConnection(DBURL, USER,PASSWORD);  Statement stmt = conn.createStatement();  // String sql = "insert into student(name,score)values('小明',34.5)";  String sql = "insert into student(name,score)values('"+name+"',score)";  int len = stmt.executeUpdate(sql);  //第四部:关闭数据库  conn.close();  }  } |

**问题:**

1)用变量的时候要拼接字符串,很麻烦

2)注意将报错,由于SQL语句中单引号的问题，

**PreparedStatement是执行预编译的SQL语句**

**注意:**

1. **能防止SQL注入**
2. **PreparedStatement可以利用数据库的缓冲,而Statement不行,理论前者效率高(mysql没有sql缓冲功能,oracle有,sqlserver有)**

PreparedStatement是Statement的子接口,获取这个接口的实例,是用

Connection接口里的: PreparedStatement prepareStatement(String sql) throws SQLException

参数sql是一个具有特殊标记的完整SQL,但是此时没有内容,通过PreparedStatement里的setXXX方法来进行设置SQL内容,并有两个方法

更新操作: int executeUpdate() throws SQLException

查询操作: ResultSet executeQuery() throws SQLException

注意这两个方法没有参数

范例:改进数据增加

|  |
| --- |
| public class test {  private static final String DBDRIVER="com.mysql.jdbc.Driver";  private static final String DBURL = "jdbc:mysql://118.89.54.105:3306/test";  private static final String USER = "lw";  private static final String PASSWORD = "linwei";  public static void main(String args[])throws Exception{  String name = "Mr'SMITH";  double score = 34.5;  Class.forName(DBDRIVER);  Connection conn = DriverManager.getConnection(DBURL, USER,PASSWORD);  String sql = "insert into student(name,score)values(?,?)";  PreparedStatement stmt = conn.prepareStatement(sql);  stmt.setString(1,name);  stmt.setDouble(2,45.6);  int len = stmt.executeUpdate();  //第四部:关闭数据库  System.out.print(len);  conn.close();  }  } |

注意:java.util.Date类有三个子类都是在java.sql包中的:

Java.sql.Date:描述的是日期

Java.sql.Time:描述的是时间

Java.sql.Timestamp:描述的是时间戳(日期时间)

如果要将java.util.Date变为java.sql.Date(Time,Timestamp),只能依靠long完成,通过java.util.Date:public long getTime():将Date变为long

Java.sql.Date:public Date(long date):将long变为sql.Date

范例:查询全部数据:

|  |
| --- |
| public class test {  private static final String DBDRIVER="com.mysql.jdbc.Driver";  private static final String DBURL = "jdbc:mysql://118.89.54.105:3306/test";  private static final String USER = "lw";  private static final String PASSWORD = "linwei";  public static void main(String args[])throws Exception{  String name = "Mr'SMITH";  double score = 34.5;  Class.forName(DBDRIVER);  Connection conn = DriverManager.getConnection(DBURL, USER,PASSWORD);  String sql = "select name,score from student where name=?";  PreparedStatement stmt = conn.prepareStatement(sql);  stmt.setString(1,name);  ResultSet rs = stmt.executeQuery();  while(rs.next()){  String x = rs.getString("name");  double y = rs.getDouble("score");  System.out.print("姓名:" + x + ",价格:"+y);  }  conn.close();  }  } |

# 4.批处理

**如果要执行2000条的插入语句,如果不用批处理,则要向服务器发送2000条的sql语句,效率不高,使用批处理就能解决这个问题**

批处理指的是一次性向数据库之中发出多条操作命令一起执行，要想操作批处理需要使用

Statement和PreparedStatement上的方法

**Statement接口定义的方法:**

增加批处理语句: public void addBatch(String sql) throws SQLException 把sql语句添加到缓冲区中

执行批处理: int[] executeBatch() throws SQLException//返回的数组是包含了所有批处理语句的执行结果，第一个批处理语句对应数组下标0的结果，排序排序

清空sql缓存区语句:void clearBatch()

**PreparedStatement接口定义的方法:**

**增加批处理:** public void addBatch() throws SQLException 把参数添加到缓冲区中

执行批处理:和Statement一样

清空sql缓存区:和Statement一样

范例:Statement

|  |
| --- |
| public class Main {  private static String URL = "jdbc:mysql://47.96.27.61:3306/test";  private static String USER = "lw";  private static String PASSWD = "linwei";  public static void main(String args[]) throws SQLException, ClassNotFoundException {  Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  Statement stmt = conn.createStatement();  //每次发送20条sql  for(int i=1;i<=2000;i++) {  stmt.addBatch("INSERT INTO student VALUES ("+i+",'张三',2,'男')");  if(i%20==0){  //执行批处理  stmt.executeBatch();  //清空缓存区  stmt.clearBatch();  }  }  stmt.close();  conn.close();  }  } |

由于JDBC具备自动提交的功能,如果某条sql语句错误,这条错误的语句不会执行,不是错误的语句都能执行,这显然是有问题的

范例:PreparedStatement

|  |
| --- |
| public class Main {  private static String URL = "jdbc:mysql://47.96.27.61:3306/test";  private static String USER = "lw";  private static String PASSWD = "linwei";  public static void main(String args[]) throws SQLException, ClassNotFoundException {  Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  PreparedStatement stmt = conn.prepareStatement("INSERT INTO student VALUES (?,?,?,?)");  //每次发送20条sql  for(int i=1;i<=2000;i++) {  stmt.setInt(1,i);  stmt.setString(2,"张三");  stmt.setInt(3,20);  stmt.setString(4,"男");  //把参数添加到缓冲区  stmt.addBatch();  //每20次发送1次参数  if(i%20==0){  //执行批处理  stmt.executeBatch();  //清空缓冲区  stmt.clearBatch();  }  }  stmt.close();  conn.close();  }  } |

JDBC提供了事务处理操作来进行手工的事务控制,所有的方法都在Connection接口中：

1. 事务提交: public void commit() throws SQLException
2. 事务回滚: public void rollback() throws SQLException

# CallableStatement接口

用于执行存储过程的sql语句

含有的方法:

ResultSet executeQuery()存储过程只能执行查询sql

|  |
| --- |
| DELIMITER $  CREATE PROCEDURE pro\_testByStu(IN sid INT)  BEGIN  SELECT \* FROM student WHERE id=sid;  END $; |

|  |
| --- |
| public static void main(String args[]) throws SQLException, ClassNotFoundException {  Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  String sql = "call pro\_testByStu(?)";  CallableStatement stmt = conn.prepareCall(sql);  stmt.setInt(1,1);  ResultSet rs = stmt.executeQuery();//执行存储过程只能使用这个方法  while(rs.next()){  int id = rs.getInt("id");  String name = rs.getString("name");  int age = rs.getInt("age");  System.out.println("id=" + id + ",name=" + name+",age="+age);  }  stmt.close();  conn.close();  }  } |

范例:带输出参数的存储过程:

|  |
| --- |
| DELIMITER $;  CREATE PROCEDURE pro\_testByStu2(IN sid INT,OUT sname VARCHAR(20))  BEGIN  SELECT NAME INTO sname FROM student WHERE id=sid;  END $; |

|  |
| --- |
| public static void main(String args[]) throws SQLException, ClassNotFoundException {  Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  String sql = "call pro\_testByStu2(?,?)";  CallableStatement stmt = conn.prepareCall(sql);  stmt.setInt(1,1);  //设置输出参数  //sqlType就是存储过程中输出参数的数据类型varchar  stmt.registerOutParameter(2,java.sql.Types.VARCHAR);  stmt.executeQuery();//执行存储过程只能使用这个方法  //这里的参数和设置输出参数的位置保持一致  //注意:  //ResultSet的getXXX方法是为了获取列的值  //CallableStatement的getXXX是为了获取输出参数的值  String name = stmt.getString(2);  System.out.println(name);  stmt.close();  conn.close();  }  } |

# 读取文件到数据库中

|  |
| --- |
| public class Main {  private static String URL = "jdbc:mysql://47.96.27.61:3306/test";  private static String USER = "lw";  private static String PASSWD = "linwei";  public static void main(String args[]) throws SQLException, ClassNotFoundException, FileNotFoundException {  Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  String sql = "insert into news values(?,?)";  PreparedStatement stmt = conn.prepareStatement(sql);  stmt.setString(1,"故事1");  Reader in = new FileReader("./src/a.txt");  stmt.setCharacterStream(2,in);  stmt.executeUpdate();  stmt.close();  conn.close();  }  } |

# 获取自增长的值

Connection接口里的:

PreparedStatement prepareStatement(String sql, int autoGeneratedKeys)

autoGenerateKeys指示是否应该返回自动生成键的标志,自增长生成的值放在了执行完sql语句后的PreparedStatement对象

Statement.RETURN\_GENERATED\_KEYS:可以返回自增长生成的值

Statement.NO\_GENERATED\_KEYS(默认):不返回自增长生成的值

|  |
| --- |
| Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  String deptSql = "insert into dept(name) VALUES(?)";  String employeeSql = "insert into employee(name,deptId) VALUES(?,?)";  PreparedStatement stmt = conn.prepareStatement(deptSql,Statement.RETURN\_GENERATED\_KEYS);  //插入部门的sql  stmt.setString(1,"软件的维护部门");  stmt.executeUpdate();  //获取自增长的值,ResultSet只包含了自增长的字段  ResultSet rs = stmt.getGeneratedKeys();  int deptId = 0;  if(rs.next()){  deptId = rs.getInt(1);  }  //插入员工的sql  stmt = conn.prepareStatement(employeeSql);  stmt.setString(1,"李四");  stmt.setInt(2,deptId);  stmt.executeUpdate();  stmt.close();  conn.close(); |

# 8.事务

innoDB数据库类型:支持事务

MyISAM:不支持事务,但是执行效率高

mysql数据库操作事务的命令:

set autocommit=1/0;

0:表示关闭自动提交事务,切换到手动提交

1:表示自动提交事务,每次执行完sql后都会提交

commit;提交事务,事务一旦提交就永远的保存到数据库中,不能再回滚了

rollback:回滚事务,当事务中的任何一个操作出现问题时,就使用回滚命令,就回滚到开启事务的点

**事务的4大特性:**

原子性(Atomicity):事务是一个不可分割的工作单位,事务中的操作要么都发生,要么都不发生

一致性(Consistency):事务必须使数据库从一个一致状态变换到另一个一致性状态

隔离性(Isolation):事务的隔离性是多个用户并发访问数据库时,数据库为每一个用户开启的事务,不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离

持久性(Durability):持久性是指一个事务一旦被提交,它对数据库中数据的改变就是永久性的,接下来的即使数据库发生故障也不应该对其有任何影响

事务并发问题:

脏读:一个事务读到另一个事务未提交的更新数据

不可重复读:一个事务读到另一个事务已经提交的更新数据

幻读:一个事务读到另一个事务新插入的数据

**事务隔离级别**:

隔离级别 脏读 不可重复读 幻读

Serializable 否 否 否

Repeatable Read 否 否 是 默认级别

Read Committed 否 是 是

Read Uncommitted 是 是 是

|  |
| --- |
| Class.forName("com.mysql.jdbc.Driver");  Connection conn = DriverManager.getConnection(URL,USER,PASSWD);  PreparedStatement stmt = null;  /\*\*  \* 需求:eric转账2000元给jacky  \*/  //关闭自动提交功能  try {  conn.setAutoCommit(false);//向数据库发送set autocommit=0;  //从eric账户上扣除2000元  String delSql = "update account set balance=balance-2000 where name='eric'";  //向jacky账户打入2000元  String addSql = "update account set balance=balance+2000 where name='jack'";  stmt = conn.prepareStatement(delSql);  stmt.executeUpdate();  stmt = conn.prepareStatement(addSql);  stmt.executeUpdate();  //当所有的sql执行完成,没有问题,则提交事务  conn.commit();//向数据库发送commit;命令  System.out.println("转账成功");  }catch (Exception e){  e.printStackTrace();  try{  conn.rollback();//向数据库发送rollback;命令  }catch (SQLException el){  el.printStackTrace();  }  }finally {  stmt.close();  conn.close();  } |