JDBC

（Java Database Connectivity）

一、概述：

JDBC 从物理结构上说就是 Java 语言访问数据库的一套接口集合。从本质上来说就是调用者（程序员）和实行者（数据库厂商）之间的协议。JDBC 的实现由数据库厂商以驱动程序的形式提供。JDBC API 为 Java 开发者使用数据库提供了统一的编程接口，它由一组 Java 类和接口组成，使得开发人员可以使用纯 Java 的方式来连接数据库，并进行操作。

1．在 JDBC 中包括了两个包：java.sql 和 javax.sql。

① java.sql　基本功能。这个包中的类和接口主要针对基本的数据库编程服务，如生成连接、执行语句以及准备语句和运行批处理查询等。同时也有一些高级的处理，比如批处理更新、事务隔离和可滚动结果集等。

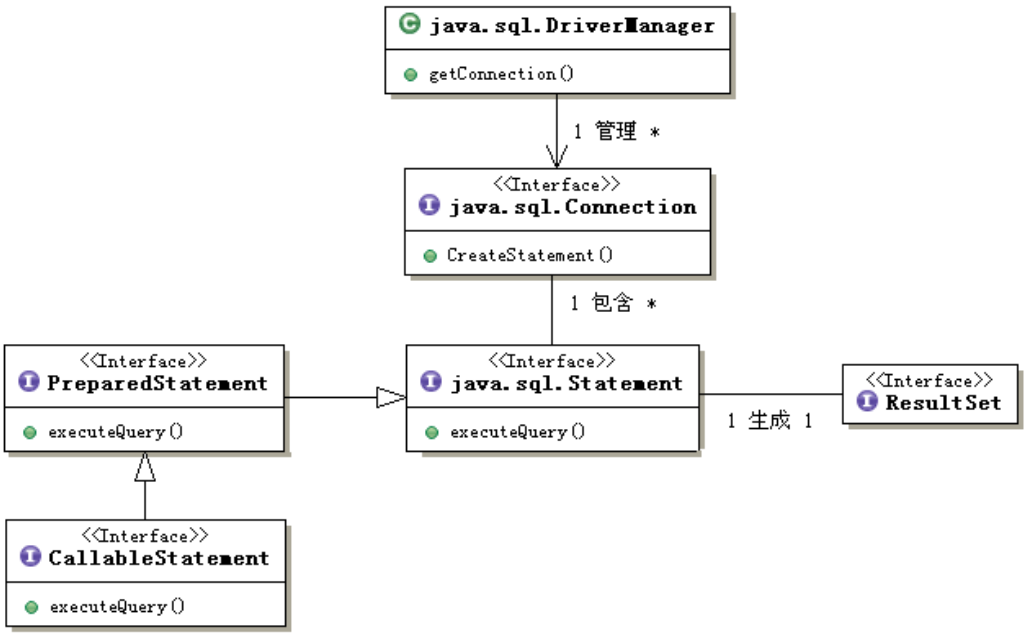
② javax.sql　扩展功能。它主要为数据库方面的高级操作提供了接口和类。如为连接管理、分布式事务和旧有的连接提供了更好的抽象，它引入了容器管理的连接池、分布式事务和行集等。

主要对象和接口：

注：除了标出的 Class,其它均为接口。

API	说明
java.sql.Connection	与特定数据库的连接（会话）。能够通过 getMetaData 方法获得数据库提供的信息、所支持的 SQL 语法、存储过程和此连接的功能等信息。代表了数据库。
java.sql.Driver	每个驱动程序类必需实现的接口，同时，每个数据库驱动程序都应该提供一个实现 Driver 接口的类。
java.sql.DriverManager (Class)	管理一组 JDBC 驱动程序的基本服务。作为初始化的一部分，此接口会尝试加载在”jdbc.drivers”系统属性中引用的驱动程序。只是一个辅助类，是工具。
java.sql.Statement	用于执行静态 SQL 语句并返回其生成结果的对象。
java.sql.PreparedStatement	继承 Statement 接口，表示预编译的 SQL 语句的对象，SQL 语句被预编译并且存储在 PreparedStatement 对象中。然后可以使用此对象高效地多次执行该语句。
java.sql.CallableStatement	用来访问数据库中的存储过程。它提供了一些方法来指定语句所使用的输入/输出参数。
java.sql.ResultSet	指的是查询返回的数据库结果集。
java.sql.ResultSetMetaData	可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。
java.sql.DatabaseMetaData	包含了关于数据库整体元数据信息。

对象和接口关系图：



驱动程序按照工作方式分为四类：（了解）

1、 JDBC-ODBC bridge + ODBC 驱动

JDBC-ODBC bridge 桥驱动将 JDBC 调用翻译成 ODBC 调用，再由 ODBC 驱动翻译成访问数据库命令。

优点：可以利用现存的 ODBC 数据源来访问数据库。

缺点：从效率和安全性的角度来说比较差。不适合用于实际项目。

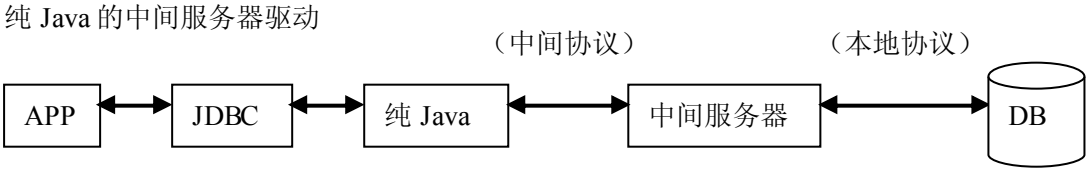
2、 基于本地 API 的部分 Java 驱动

我们应用程序通过本地协议跟数据库打交道。然后将数据库执行的结果通过驱动程序中的 Java 部分返回给客户端程序。

优点：效率较高；

缺点：安全性较差。

3、 纯 Java 的中间服务器驱动

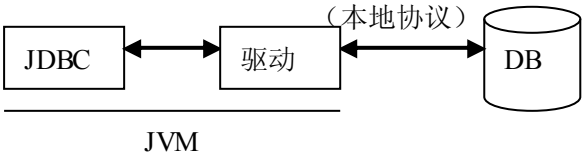


缺点：两段通信，效率比较差

优点：安全性较好

4、 纯 Java 本地协议：通过本地协议用纯 Java 直接访问数据库。

特点：效率高，安全性好。



二、JDBC 编程的步骤

必须掌握！

① 注册一个 driver

注册驱动程序有三种方式：

ojdbc14.jar

方式一： **class.forName("oracle.jdbc.driver.OracleDriver");**

Java 规范中明确规定：所有的驱动程序必须在静态初始化代码块中将驱动注册到驱动程序管理器中。

方式二： **Driver drv = new oracle.jdbc.dirver.OracleDriver();** //针对没有隐式注册时采用

DriverManager.registerDriver(drv);

方式三： 编译时在虚拟机中加载驱动

java –D jdbc.dirvers=驱动全名 类名

例： javac –D jdbc.drivers=oracle.jdbc.driver.OracleDriver xxx.java

使用系统属性名，加载驱动； -D 表示为系统属性赋值。

附： mysql 的 Driver 的全名 com.mysql.jdbc.Driver

SQLServer 的 Driver 的全名 com.microsoft.jdbc.sqlserver.SQLServerDriver

② 建立连接

Oracle 子协议

conn=DriverManager.getConnection("**jdbc:oracle:thin:@192.168.0.23:**1521**:tarena**",

" User"," Pasword");

IP 地址及端口号和数据库实例名

用户名，密码

Connection 连接是通过 **DriverManager** 的静态方法 getConnection(....)来得到的，这个方法的实质是把参数传到实际的 Driver 中的 connect()方法中来获得数据库连接的。

Oracle URL 的格式：

jdbc:oracle:thin:（协议）@XXX.XXX.X.XXX:XXXX（IP 地址及端口号）:XXXXXXX（所使用的库名）

MySql URL 的写法 例： jdbc:mysql://192.168.8.21:3306/test

SQLServer URL 的写法 例：jdbc:microsoft:sqlserver://192.168.8.21:1433

③ 获得一个 Statement 对象 **stm = conn.createStatement();**

④ 通过 Statement 执行 SQL 语句

stm.excuteQuery(Sring sql); //返回一个查询结果集

stm.excuteUpdate(String sql); //返回值为 int 型，表示影响记录的条数。

stm.excute(String sql); //返回 true，表示查询；返回 false，表示其它操作。

将 sql 语句通过连接发送到数据库中执行，以实现对数据库的操作。

⑤ 处理结果集 ResultSet

使用 Connection 对象获得一个 Statement，Statement 中的 executeQuery(String sql) 方法可以使用 select 语句查询，并且返回一个结果集 ResultSet，通过遍历这个结果集，可以获得 select 语句的查寻结果，ResultSet 的 next()方法会操作一个游标从第一条记录的前面开始读取，直到最后一条记录。executeUpdate(String sql) 方法用于执行 DDL 和 DML 语句，主要还是 DML，包括 insert，delete，update 操作。

只有执行 select 语句才有结果集返回。

例：
```
Statement str=con.createStatement();  //创建 Statement
String sql="insert into test(id,name) values(1,"+"'"+"test"+"'"+")";
str. executeUpdate(sql);//执行 Sql 语句
String sql="select * from test";
ResultSet rs=str. executeQuery(String sql);//执行 Sql 语句，执行select 语句后有结果集
//遍历处理结果集信息
    while(rs.next()){
      System.out.println(rs.getInt("id"));
      System.out.println(rs.getString("name"))
    }
```

next()如果有下一条记录返回 true,否则为 false;有，则游标向下一条记录.

⑥ 关闭数据库连接（释放资源） 调用.close()

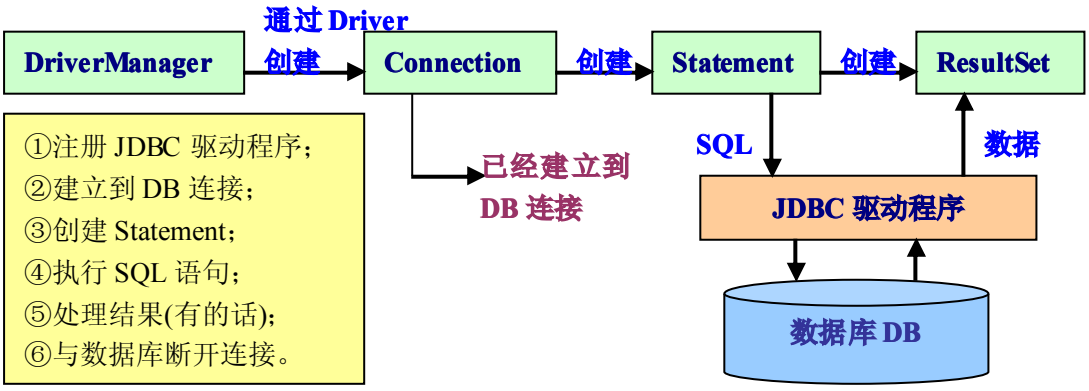
用完及时关！

```
rs.close();        stm.close();        conn.close();
```

ResultSet Statement Connection 是依次依赖的。

注意：要按先 ResultSet 结果集，后 Statement，最后 Connection 的顺序关闭资源，因为 Statement 和 ResultSet 是需要连接时才可以使用的，所以在使用结束之后有可能其它的 Statement 还需要连接，所以不能先关闭 Connection。

图形演绎编写 JDBC 程序的一般过程：



三、几个重要接口：

(1) Statement —— SQL 语句执行接口

Statement 接口代表了一个数据库的状态，在向数据库发送相应的 SQL 语句时，都需要创建 Statement 接口或者 PreparedStatement 接口。在具体应用中，Statement 主要用于操作不带参数（可以直接运行）的 SQL 语句，比如删除语句、添加或更新。

Statement a. 某些情况下，效率不高；

b. 语法结构不够清晰，造成对类型的支持不太好；

(2) PreparedStatement —— 预编译的 Statement (重点)

Step1: 通过连接获得 PreparedStatement 对象，用带占位符(?)的 sql 语句构造。

PreparedStatement pstm = con.prepareStatement("select * from test where id=?");

Step2: 设置参数

Pstm.setString(1, "08868");

Step3: 执行 sql 语句

rs = pstm.excuteQuery();

Statemnet 发送完整的 SQL 语句到数据库不是直接执行而是由数据库先编译，再运行。而 PreparedStatement 是先发送带参数的 SQL 语句，再发送一组参数值。如果是同构的 SQL 语句，PreparedStatement 的效率要比 Statement 高。而对于异构的 SQL 则两者效率差不多。

同构：两个 SQL 语句可编译部分是相同的，只有参数值不同。

异构：整个 SQL 语句的格式是不同的。

注意点： ① 使用预编译的 Statement 编译多条 SQL 语句一次执行

② 可以跨数据库使用，编写通用程序

③ 能用预编译时尽量用预编译

PreparedStatement 也执行相应的 SQL 语句。它继承于 Statement 接口，除了具备 Statement 所有功能，还可以对 SQL 语句进行预处理。

PreparedStatement 较 Statement 而言:

- 存储预编译的 Statement，多次执行时有较好的执行效率。
- 对于程序员而言，代码比较容易维护。
- 有较好的安全性，如：可以在一定程度上防止 SQL 注入（SQL Injection）。

主要方法：

① ResultSet **executeQuery**(sql) throws SQLException

在此 PreparedStatement 对象中执行 SQL 查询，并返回该查询生成的 ResultSet 对象。从不返回 null；如果发生数据库访问错误或者 SQL 语句没有返回 ResultSet 对象则抛出 SQLException 异常。

② int **executeUpdate**(sql) throws SQLException

在此 PreparedStatement 对象中执行 SQL 语句，该语句必须是一个 SQL INSERT、UPDATE 或 DELETE 语句；或者是一个什么都不返回的 SQL 语句，比如 DDL 语句。返回值 int 表示影响的记录条数，一条都没有则返回 0；

③ boolean **execute**(sql)throws SQLException

在此 PreparedStatement 对象中执行 SQL 语句，该语句可以是各种类型 SQL 语句。有结果集则返回 true, 没有结果集则返回 false；

④各种 set 方法

将指定位置的参数设置为指定的类型。比如 ps.setString(3, “tarena”);

(3) ResultSet —— 结果集操作接口

ResultSet 接口是查询结果集接口，它对返回的结果集进行处理。ResultSet 是程序员进行 JDBC 操作的必需接口。

(4) ResultSetMetaData —— 元数据操作接口

ResultSetMetaData 是对元数据进行操作的接口，可以实现很多高级功能。Hibernate 运行数据库的操作，大部分都是通过此接口。可以认为，此接口是 SQL 查询语言的一种反射机制。ResultSetMetaData 接口可以通过数组的形式，遍历数据库的各个字段的属性，对于我们开发者来说，此机制的意义重大。

JDBC 通过元数据(MetaData)来获得具体的表的相关信息，例如，可以查询数据库中有哪些表，表有哪些字段，以及字段的属性等。MetaData 中通过一系列 getXXX 将这些信息返回给我们。

MetaData 包括：
- 数据库元数据 Database MetaData　　使用 connection.getMetaData()获得
  包含了关于数据库整体元数据信息。
- 结果集元数据 Result Set MetaData　　使用 resultSet.getMetaData()获得
  比较重要的是获得表的列名、列数等信息。

结果集元数据对象：ResultSetMetaData meta = rs.getMetaData();

- ✓ 字段个数：meta.getColomnCount();
- ✓ 字段名字：meta.getColumnName();
- ✓ 字段 JDBC 类型：meta.getColumnType();
- ✓ 字段数据库类型：meta.getColumnTypeName();

数据库元数据对象：DatabaseMetaData dbmd = con.getMetaData();

- ✓ 数据库名：dbmd.getDatabaseProductName();
- ✓ 数据库版本号：dbmd.getDatabaseProductVersion();
- ✓ 数据库驱动名：dbmd.getDriverName();
- ✓ 数据库驱动版本号：dbmd.getDriverVersion();
- ✓ 数据库 URL：dbmd.getURL();
- ✓ 该连接的登录名：dbmd.getUserName();

四、JDBC 异常处理：

JDBC 中，和异常相关的两个类是 SQLException 和 SQLWarning。

1. SQLException 类：用来处理较为严重的异常情况。

比如：① 传输的 SQL 语句语法的错误；

② JDBC 程序连接断开；

③ SQL 语句中使用了错误的函数。

SQLException 提供以下方法：

getNextException() —— 用来返回异常栈中的下一个相关异常；

getErrorCode() —— 用来返回代表异常的整数代码 (error code)；

getMessage() —— 用来返回异常的描述信息 (error message)。

2. SQLWarning 类：用来处理不太严重的异常情况，也就是一些警告性的异常。其提供的方法和使用与 SQLException 基本相似。

结合异常的两种处理方式，明确何时采用哪种。

A. throws 处理不了，以及要让调用者知道，就 throws;

B. try … catch 能自行处理，就进行异常处理。

五、JDBC 中使用 Transaction 编程（事务编程）

1. 事务是具备以下特征(ACID)的工作单元：

- ➢ 原子性（Atomicity）—— 如果因故障而中断，则所有结果均被撤消；
- ➢ 一致性（Consistency）—— 事务的结果保留不变；
- ➢ 孤立性（Isolation）—— 中间状态对其它事务是不可见的；
- ➢ 持久性（Durability）—— 已完成的事务结果上持久的。

原子操作，也就是不可分割的操作，必须**一起成功一起失败**。

2. 事务处理三步曲：（事务是一个边界）

① connection.setAutoCommit(false); //把自动提交关闭

② 正常的 DB 操作 //若有一条 SQL 语句失败了，自动回滚

③ connection.commit() //主动提交

或 connection.rollback() //主动回滚

```
完整的代码片段：
try{
    con.setAutoCommit(false);    //step① 把自动提交关闭
    Statement stm = con.createStatement();
    stm.executeUpdate("insert into person(id, name, age) values(520, 'X-Man', 18)");
    stm.executeUpdate("insert into Person(id, name, age) values(521, 'Super', 19)");
    //step② 正常的 DB 操作
    con.commit();                          //step③ 成功主动提交
} catch(SQLException e){
    try{
        con.rollback();
    } catch(Exception e){ e.printStackTrace(); }      //step③′ 失败则主动回滚
}
```

3. JDBC 事务并发产生的问题和事务隔离级别

JDBC 事务并发产生的问题：

① 脏读（Dirty Reads） 一个事务读取了另一个并行事务还未提交的数据。

② 不可重复读（UnPrpeatable Read） 一个事务再次读取之前的数据时，得到的数据不一致，被另一个已提交的事务修改。

③ 幻读（Phantom Read） 一个事务重新执行一个查询，返回的记录中包含了因为其它最近提交的事务而产生的新记录。

为了避免以上三种情况的出现，则采用

事务隔离级别：

TRANSACTION_NONE	不使用事务
TRANSACTION_READ_UNCOMMITTED	可以读取未提交数据
TRANSACTION_READ_COMMITTED	可以避免脏读，不能够读取没提交的数据，最常用的隔离级别 大部分数据库的默认隔离级别
TRANSACTION_REPEATABLE_READ	可以避免脏读，重复读取
TRANSACTION_SERIALIZABLE	可以避免脏读，重复读取和幻读，（事务串行化）会降低数据库效率

以上的五个事务隔离级别都是在 **Connection** 类中定义的静态常量，使用 **setTransactionIsolation(int level)** 方法可以设置事务隔离级别。

比如：con.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED**);**

六、JavaBean 定义

1、是一个普通的 Java 类；

2、在结构上没有预先的规定，不需要容器；

3、要求放在包中，要求实现 java.io.Serializalbe 接口

4、要求有一个无参的构造方法；

5、属性的类型必须保持唯一，返回值必须和 set 方法参数类型一致

6、对每个属性要有对应的 get 和 set 方法。注：隐藏属性可以没有。

另外，POJO 与 JavaBean 的区别：

POJO——Pure Old Java Object or Plain Ordinary Java Object （简单 Java 类对象）

POJO 原则上不鼓励在 JavaBean 里面写业务逻辑方法。简单说 POJO 除了能赋值，也就是提供 get/set 方法，别的什么也不能做，它就好比一个水杯, 不是个能烧水的水壶，因为它没有烧水()这个方法，因此只能盛水。POJO 主要用来和数据库里的表进行对应。