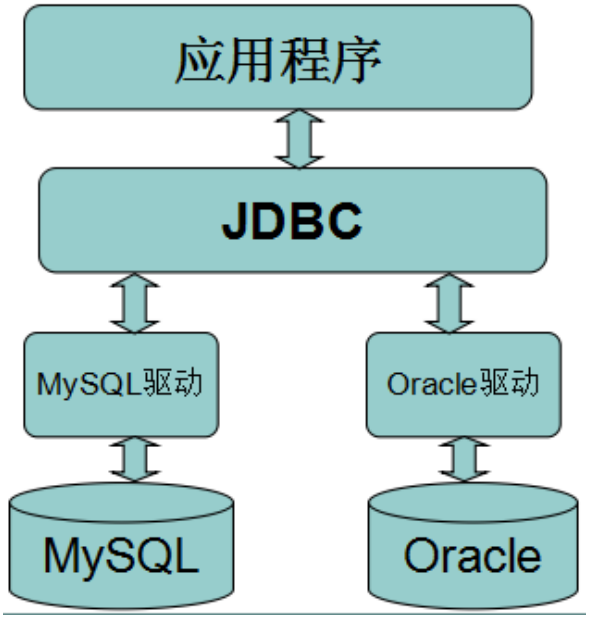
JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。



常用接口

1.**Driver**接口

Driver接口由数据库厂家提供。在编程中要连接数据库，必须先装载特定厂商的数据库驱动程序：

装载MySql驱动：Class.forName("com.mysql.jdbc.Driver");

装载Oracle驱动：

Class.forName("oracle.jdbc.driver.OracleDriver");

2.**Connection**接口

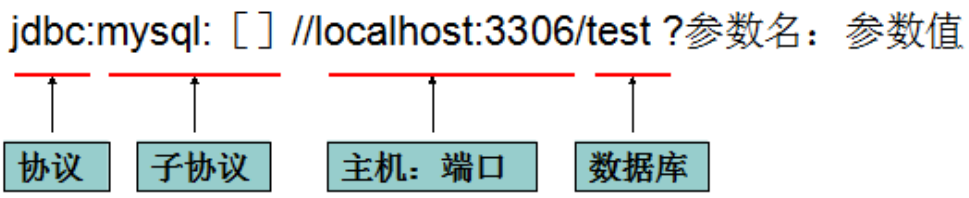
Connectio表示数据库的连接（会话），在连接上下文中执行sql语句并返回结果。`Connection conn = DriverManager.getConnection(url,user,pass);`

常用数据库URL地址的写法：

Oracle：jdbc:oracle:thin:@localhost:1521:shen

SqlServer：jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=shen

MySql：jdbc:mysql://localhost:3306/shen



常用方法：

- createStatement()：创建向数据库发送sql的statement对象。
- prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。
- prepareCall(sql)：创建执行存储过程的callableStatement对象。
- setAutoCommit(boolean autoCommit)：设置事务是否自动提交。
- commit() ：在链接上提交事务。
- rollback() ：在此链接上回滚事务。

3.**Statement**接口

用于执行静态SQL语句并返回它所生成结果的对象。

三种Statement类：

- Statement：由createStatement创建，用于发送简单的SQL语句（不带参数）。
- PreparedStatement ：继承自Statement接口，由preparedStatement创建，用于发送含有一个或多个参数的SQL语句。
- CallableStatement：继承自PreparedStatement接口，由方法prepareCall创建，用于调用存储过程。

常用Statement方法：

- execute(String sql):运行语句，返回是否有结果集

- executeQuery(String sql)：运行select语句，返回ResultSet结果集。
- executeUpdate(String sql)：运行insert/update/delete操作，返回更新的行数。
- addBatch(String sql) ：把多条sql语句放到一个批处理中。
- executeBatch()：向数据库发送一批sql语句执行。
- cleanBatch():清空当前 SQL 命令列表。

PreperedStatement接口

Statement的子类，PreperedStatement可以避免SQL注 入。当运行时动态地把参数传给PreprareStatement时，即使参数里有敏感字符如 or '1=1'也数据库会作为一个参数一个字段的属性值来处理而不会作为一个SQL指令。Statement会使数据库频繁编译SQL，可能造成数据库缓冲区溢出。PreparedStatement 可对SQL进行预编译，从而提高数据库的执行效率。并且PreperedStatement对于sql中的参数，允许使用占位符的形式进行替换，简化sql语句的编写。

```
PreperedStatement st =    conn.preparedStatement()
eg.    PreparedStatement ps = conn.preparedStatement(sql);
       ps.setString(1, “col_value”);
```

4.ResultSet接口

代表封装Sql语句的执行结果。维护了一个指向表格数据行的游标。

ResultSet提供检索不同类型字段的方法，常用的有：

- getString(int index)、getString(String columnName)：获得在数据库里是varchar、char等类型的指定数据对象。
- getObject(int index)、getObject(String columnName)：获取在数据库里任意类型的数据。

ResultSet还提供了对结果集进行滚动的方法：

- next()：移动到下一行
- Previous()：移动到前一行

- absolute(int row)：移动到指定行
- beforeFirst()：移动resultSet的最前面。
- afterLast() ：移动到resultSet的最后面。

依次关闭对象及连接释放资源：ResultSet → Statement → Connection

连接步骤

加载JDBC驱动程序 → 建立数据库连接Connection → 创建执行SQL的语句Statement → 处理执行结果ResultSet → 释放资源

注册驱动

方式一：Class.forName("com.MySQL.jdbc.Driver");

推荐，不会对具体的驱动类产生依赖。要求JVM查找并加载指定的类，也就是说JVM会执行该类的静态代码段。

方式二：DriverManager.registerDriver(com.mysql.jdbc.Driver);

会造成DriverManager中产生两个一样的驱动，并会对具体的驱动类产生依赖。

Properties类

用于读取Java的.properties配置文件，格式为文本文件，文件的内容的格式是“键=值”的格式。

1. getProperty (String key)，用指定的键在此属性列表中搜索属性。
2. load (InputStream inStream)，从输入流中读取属性列表（键和元素对）。通过对指定的文件（.properties 文件）进行装载来获取该文件中的所有键 - 值对。以供 getProperty (String key) 来搜索。
3. setProperty (String key, String value) ，调用 Hashtable 的方法 put 。他通过调用基类的put方法来设置 键 - 值对。
4. store (OutputStream out, String comments)，以适合使用 load 方法加载到 Properties 表中的格式，将此 Properties 表中的属性列表（键和元素对）写入输出流。与 load 方法相反，该方法将键 - 值对写入到指定的文件中去。

5. clear ()，清除所有装载的 键 - 值对。该方法在基类中提供。

eg:

资源文件：db.properties

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/shen
username=shen
password=Anxin062039
```

读取：

```
InputStream in =
JdbcUtils.class.getClassLoader().getResourceAsStream("db.properties");
/*      获取对象所属的class类，再获取类加载对象，由类加载器来获取资源。当传入的参数没有"/"时，获取的是当前类所在包下（src）的对应文件。而当参数带有"/"，则是从ClassPath根目录下获取文件。
特点：程序不用明确制定配置文件的具体所在目录。程序可以自动的在//src目录下搜索该文件，并加载，但文件不能太大。*/
/*   //先得到资源
     URL url = Demo2.class.getClassLoader().getResource("db.properties");
     //得到资源路径
     String path = url.getPath();
     //得到与该资源相关的流
     FileInputStream in = new FileInputStream(path);
好处：当有人更改配置文件信息时，我再次访问的时候，得到的是更改后的信息。  */
                        Properties prop = new Properties();
                        prop.load(in);

                        driver = prop.getProperty("driver");
                        url = prop.getProperty("url");
                        username = prop.getProperty("username");
                        password = prop.getProperty("password");
```

事务

- **事务基本概念**
  - **一组要么同时执行成功，要么同时执行失败的SQL语句。是数据库操作的一个执行单元！**
  - **事务开始于：**
    - 连接到数据库上，并执行一条DML语句(INSERT、UPDATE或DELETE)。
    - 前一个事务结束后，又输入了另外一条DML语句。
  - **事务结束于：**
    - 执行COMMIT或ROLLBACK语句。
    - 执行一条DDL语句，例如CREATE TABLE语句；在这种情况下，会自动执行COMMIT语句。
    - 执行一条DCL语句，例如GRANT语句；在这种情况下，会自动执行COMMIT语句。
    - 断开与数据库的连接。
    - 执行了一条DML语句，该语句却失败了；在这种情况中，会为这个无效的DML语句执行ROLLBACK语句。

- 事务的四大特点（ACID）
  - atomicity（原子性）
    - 表示一个事务内的所有操作是一个整体，要 么全部成功，要么全失败；
  - consistency（一致性）
    - 表示一个事务内有一个操作失败时，所有的更改过的数据都必须回滚到修改前的状态；
  - isolation（隔离性）
    - 事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。
  - durability（持久性）
    - 持久性事务完成之后，它对于系统的影响是永久性的。
- 事务隔离级别从低到高：
  - **读取未提交（Read Uncommitted)**
  - **读取已提交(Read Committed)**
  - 可重复读（Repeatable Read)
  - 序列化（serializable)

事务操作：

```
conn.setAutoCommit(false); //设为手动提交，用于处理事务
stmt.addBatch("insert into t_user (userName,pwd,regTime) values ('hao" + i + "',666666,now())");
stmt.executeBatch();
conn.commit();  //提交事务
```

事务回滚：

```
Savepoint sp = null;
...
```

```
sp = conn.setSavepoint();//在这里设置事务回滚点
...
catch (Exception e) {
                        try {
                                conn.rollback(sp);//回滚到该事务点，即该点之前的会正常执行（sql1）
                                conn.commit();  //回滚了要记得提交,如果没有提交sql1将会自动回滚
                        }
```

设置事务隔离级别：

```
conn.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);//避免脏读
conn.setAutoCommit(false);
/*
Serializable：可避免脏读、不可重复读、虚读情况的发生。（串行化）
Repeatable read：可避免脏读、不可重复读情况的发生。（可重复读）
Read committed：可避免脏读情况发生（读已提交）。
Read uncommitted：最低级别，以上情况均无法保证。(读未提交)
*/
```

批处理Batch：

- 灵活指定SQL语句中的变量
  - PreparedStatement
- 对存储过程进行调用
  - CallableStatement
- 运用事务处理
  - Transaction
- 批处理
  - Batch
  - 对于大量的批处理，建议使用Statement，因为PreparedStatement的预编译空间有限，当数据量特别大时，会发生异常。

```
st.addBatch(sql1);      //添加sql操作
        st.addBatch(sql2);
        st.addBatch(sql3);
        st.executeBatch();          //提交操作
        st.clearBatch();            //清除当前SQL命令列表
```

调用存储过程：

{暂空}

CLOB文本大对象操作

大文本存储：

```
File file = new File(path);
st.setCharacterStream(1, new FileReader(file), (int) file.length());//将该文件添加，数据库字段类型为Text。
```

大文本读取：

```
Reader reader = rs.getCharacterStream("resume");//数据库字段类型为Text的名称为resume
char buffer[] = new char[1024];
int len = 0;
FileWriter out = new FileWriter("c:\\1.txt");
while((len=reader.read(buffer))>0){
        out.write(buffer, 0, len);
}
```

BLOB二进制大对象的使用

- BLOB（Binary Large Object）
  - 用于存储大量的二进制数据
  - 大字段有些特殊，不同数据库处理的方式不一样，大字段的操作常常是以流的方式来处理的。而非一般的字段，一次即可读出数据。
- Mysql中相关类型：
  - TINYBLOB最大长度为255(2^[8]–1)字节的BLOB列。
  - BLOB[(M)]最大长度为65,535(2^[16]–1)字节的BLOB列。
  - MEDIUMBLOB最大长度为16,777,215(2^[24]–1)字节的BLOB列。
  - LONGBLOB最大长度为4,294,967,295或4GB(2^[32]–1)字节的BLOB列。