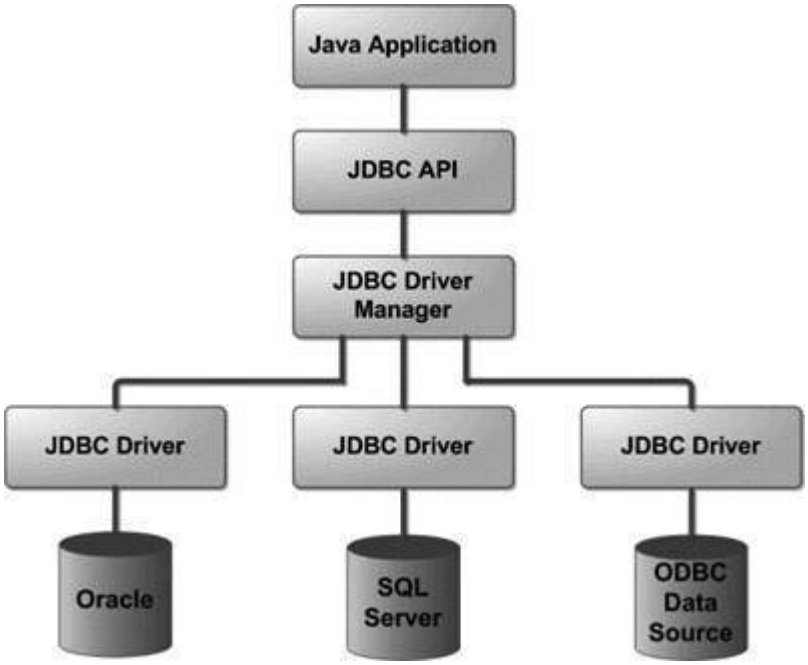
JDBC 架构

JDBC是一个规范，它提供了一整套接口，允许以一种可移植的访问底层数据库API。

JDBC 驱动程序实现了 JDBC API 中定义的接口。JDK附带的`Java.sql`包包含各种类，其类的行为被定义，实现在第三方驱动程序中完成。



创建 JDBC 应用程序

构建JDBC应用程序涉及以下六个步骤：

- **导入包**：需要包含包含数据库编程所需的JDBC类的包。 大多数情况下，使用`import java.sql.*`就足够了。
- **注册JDBC驱动程序**：需要初始化驱动程序，以便可以打开与数据库的通信通道。

  注册驱动程序是将Oracle驱动程序的类文件加载到内存中的过程

  ```
  // 自动注册
  Class.forName("oracle.jdbc.driver.OracleDriver");
  ```

  ```
  //DirverManager 静态方法
  Driver myDriver = new oracle.jdbc.driver.OracleDriver();
  DriverManager.registerDriver( myDriver );
  ```

- **打开一个连接**：需要使用`DriverManager.getConnection()`方法创建一个`Connection`对象，它表示与数据库的物理连接。

  ```
  getConnection(String url)
  getConnection(String url, Properties prop)
  getConnection(String url, String user, String password)
  ```

- **执行查询**：需要使用类型为`Statement`的对象来构建和提交SQL语句到数据库。

接口	推荐使用
Statement	用于对数据库进行通用访问，在运行时使用静态SQL语句时很有用。`Statement`接口不能接受参数。
PreparedStatement	当计划要多次使用SQL语句时使用。`PreparedStatement`接口在运行时接受输入参数。
CallableStatement	当想要访问数据库存储过程时使用。`CallableStatement`接口也可以接受运行时输入参数。

- **从结果集中提取数据**：需要使用相应的`ResultSet.getXXX()`方法从结果集中检索数据。

  ResultSet 有几种类型，规定光标是否可前后滚动，结果集是否可更改和并发性等。

  ResultSet 接口方法可分为下面三类：

  - **浏览方法**：用于移动光标（移动指向的 row）。
  - **获取方法**：用于查看光标指向的当前行的列中的数据。
  - **更新方法**：用于更新当前行的列中的数据。 然后在基础数据库中更新数据。
- **清理环境**：需要明确地关闭所有数据库资源，而不依赖于JVM的垃圾收集。

程序示例：见 ConnectionDemo.java

JDBC数据类型

日期和时间数据类型：

SQL 类型	**JDBC/Java** 类型	**set XXX**	**get XXX**
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
DATE	java.sql.Date	setDate	updateDate

JDBC 事务

关闭 **sql** 自动提交

```
conn.setAutoCommit(false);
```

提交和回滚

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees VALUES (107, 22, 'Sita', 'Singh')";
   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

使用保存点

可以在事务中定义逻辑回滚点。 如果通过保存点(`Savepoint`)发生错误时，则可以使用回滚方法来撤消所有更改或仅保存保存点之后所做的更改。

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   //set a Savepoint
   Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
   String SQL = "INSERT INTO Employees VALUES (106, 24, 'Curry', 'Stephen')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees VALUES (107, 32, 'Kobe', 'Bryant')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```

JDBC 批量处理

批量处理允许将相关的SQL语句分组到批处理中，并通过对数据库的一次调用来提交它们，一次执行完成与数据库之间的交互。一次向数据库发送多个SQL语句时，可以减少通信开销，从而提高性能。

`Statement`， `PreparedStatement`和`CallableStatement`都可以进行批处理操作。

使用 PrepareStatement 对象进行批处理

以下是使用`PrepareStatement`对象进行批处理的典型步骤顺序

- 使用占位符创建SQL语句。
- 使用`prepareStatement()`方法创建`PrepareStatement`对象。
- 使用`setAutoCommit()`将自动提交设置为`false`。
- 使用`addBatch()`方法在创建的`Statement`对象上添加SQL语句到批处理中。
- 在创建的`Statement`对象上使用`executeBatch()`方法执行所有SQL语句。
- 最后，使用`commit()`方法提交所有更改。

```
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) VALUES(?, ?, ?, ?)";
// Create PrepareStatement object
PreparedStatemen pstmt = conn.prepareStatement(SQL);
//Set auto-commit to false
conn.setAutoCommit(false);
// Set the variables
pstmt.setInt( 1, 400 );
pstmt.setString( 2, "JDBC" );
pstmt.setString( 3, "Li" );
pstmt.setInt( 4, 33 );
// Add it to the batch
pstmt.addBatch();
// Set the variables
pstmt.setInt( 1, 401 );
pstmt.setString( 2, "CSharp" );
pstmt.setString( 3, "Liang" );
pstmt.setInt( 4, 31 );
// Add it to the batch
pstmt.addBatch();

//add more batches

//Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

JDBC 流

`PreparedStatement`对象可以使用输入和输出流来提供参数数据。例如，读取文件中的数据，将数据存入数据库；读取数据库中的值，将这些值存入文件。文件中数据内容的格式为 xml。