



noteless

JDBC与ORM发展与联系 JDBC简介（九）

原文地址: [JDBC与ORM发展与联系 JDBC简介（九）](#)

概念回顾

回顾下JDBC的概念:

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。

JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

JDBC是Java数据库连接技术，所以，他必然根植于Java语言，使用JDBC离不开Java开发环境，是Java语言对于数据库连接的技术实现。

JDBC作为一种协议的体现，在Java代码中就是一系列的接口与实现的约定。

数据库驱动厂商以及应用程序开发者基于这一协议进行对接，从而解耦，从而可以相互分离的独立发展。

既然最终体现形式为一组API这组API到底做了什么？

想要了解JDBC到底做了什么，在windows平台的话，可以直接打开命令窗口

根据用户名密码进行连接，然后发送语句，然后查看结果

```
C:\WINDOWS\system32\cmd.exe - mysql -h 192.168.0.176 -u root -p123456
(c) 2017 Microsoft Corporation. 保留所有权利。

C:\Users\noteless>mysql -h 192.168.0.176 -u root -p123456
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 275
Server version: 5.6.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show database;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corre
sponds to your MySQL server version for the right syntax to use near 'database' at li
ne 1
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| kttxdtdt |
| kttxdtdt2 |
| kttxdtdtx |
| mysql |
| performance_schema |
| sampleddb |
| test |
+-----+
8 rows in set (0.00 sec)

mysql> use sampleddb;
Database changed
mysql> select * from student;
+----+-----+-----+-----+
| id | name | age | sex |
+----+-----+-----+-----+
| 2  | 李四 | 13  | 男  |
| 3  | 王二 | 14  | 女  |
| 4  | 赵六 | 15  | 男  |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

JDBC是Java数据库连接，仍旧是在做”连接数据库“这件事情本身，哪怕变出花来，他的根本仍旧是连接数据库

数据库就是那个数据库，他一直在那里，数据库有他们固有的操作流程步骤以及SQL执行规范

当你用命令窗口连接数据库，发送SQL语句时，你是在操作数据库

使用JDBC只是换了一种方式，不再是手动了，而是借助于Java代码，然后依赖于底层的数据库驱动，去操作数据库

简言之，你本来是在命令窗口里面，输入一行SQL之后敲回车

现在变成了借助于Java代码，通过几个对象相互配合进而发送SQL

做的所有事情都没有任何变化，查询还是那个查询，更新还是那个更新，变得只是形式

你以为开个法拉利去车站接人和骑电动车去接人有本质区别么？

你要做的还是去车站接人，你要接的人还是那个人，但是形式变了，停车位置变了，时间变了，连旁边的妹子看你得眼光可能也变了，但是，但是，你的事儿还是那个事儿，如果接不到人，一切还不是扯淡？

所以说一个数据库客户端一般可以提供给我们那些服务，JDBC就能够提供给我们那些服务

不过，对于客户端来说，结果直接就可以呈现出来了，但是对于Java代码---方法的调用，需要处理更多的细节，哪些是输出，哪些是输入，参数的传递

所以JDBC没有看起来这么简单

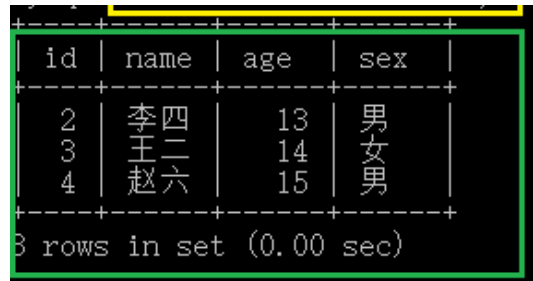
**JDBC作为数据库连接的中间层，将应用程序与数据库连接进行解耦，给开发者提供了极大地方便，从此以后，再也不需要面向数据库驱动进行编程了
只需要面向JDBC进行编程即可，所以JDBC的出现，对于Java连接数据库实现了大一统的局面，解放了生产力**

但是，你既然作为中间层，将两者进行解耦，你就要负责对接，否则就真的彻底断开了，就不叫做解耦了。。。

这其中最重要的一点就是结果集的返回

对于类似命令行或者Navicat的客户端，是直观的呈现，眼睛来识别，而对于接口调用，则是API各个方法中的数据对接，结果集的解析

JDBC是对数据库操作的Java描述，所以对于比如查询来说，结果集的逻辑呈现也是下图类似式样



id	name	age	sex
2	李四	13	男
3	王二	14	女
4	赵六	15	男

3 rows in set (0.00 sec)

JDBC对于结果集的处理核心就是将这些样子的数据返回给应用程序，直观看起来很简单，映射到字段中就涉及到很复杂的转换了

总共有多少行记录？又有多少列？有哪些字段是要处理的？字段顺序是什么？字段类型是什么？SQL类型与Java类型又是如何映射？有些字段的精度又是什么？

某列的值应该跟哪一个实体中的字段进行对照？等等这些都是结果集要处理的，所以说JDBC的确又很复杂

不得不面对的问题

冗余代码

借助于JDBC编程，有很多模块化的代码，在第一个JDBC示例中，所有的步骤都是需要按部就班完成的，而这些步骤很显然，有些是结构化的模式化的，比如连接数据库，关闭连接，异常处理，这些其实对于应用开发者其实并不想处理，但是却不得不处理

简言之，JDBC功能足够，但是便捷性欠缺，结构化本身没错，结构化模式化流程化才能成为标准，但是必然会产生冗余步骤，如何灵活是一个问题

对象映射

目前存储数据最常用最流行的工具是关系型数据库，我们通过JDBC借助于SQL语句操作数据库，但是Java是面向对象的编程语言，所有的操作都是对象
在使用JDBC进行操作时，面向对象的概念却被弱化了

比如下面的这一段代码，对于参数的设置，是按照属性字段的索引，你看不到对象的影子

你可能希望有这么一个学生Student类

这个类有几个属性：id、姓名、年龄、性别

当需要执行下面的插入行为时，可以直接将Student的对象实例直接传递进去即可，而不是这样按照索引去设置。

//3、设置sql语句

```
String sql = "INSERT into student(name,age,sex) VALUES(?,?,?)";
```

//4、获得sql语句执行对象

```
PreparedStatement ps = conn.prepareStatement(sql,Statement.NO_GENERATED_KEYS);
```

```
ps.setString( parameterIndex: 1, x: "王大猫9698");
```

```
ps.setInt( parameterIndex: 2, x: 68);
```

```
ps.setString( parameterIndex: 3, x: "男");
```

```
ps.executeUpdate();
```

155	王大猫969	68	男	(Null)
-----	--------	----	---	--------

结果集的取回也是类似的

当你想要查询一个列表时，你不得不如下这般处理

你是不是会想，我有一个Student类了，为什么不能直接给我返回一个List<Student>？那样不是很方便么？

//3、设置sql语句

```
String sql = "select * from student";
```

//4、获得sql语句执行对象

```
Statement stmt = conn.createStatement();
```

//5、执行并保存结果集

```
ResultSet rs = stmt.executeQuery(sql);
```

//6、处理结果集

```
while (rs.next()) {
```

```
    System.out.print("id:" + rs.getInt( columnIndex: 1));
```

```
    System.out.print(",姓名:" + rs.getString( columnIndex: 2));
```

```
    System.out.print(",年龄:" + rs.getInt( columnIndex: 3));
```

```
    System.out.println(",性别:" + rs.getString( columnIndex: 4));
```

```
}
```

所以看得出来，Java作为纯粹的面向对象编程语言，一切皆是对象，但是目前常用的数据库却是关系型数据库

关系模型就像一张二维表格，因而一个关系型数据库就是由二维表及其之间的联系组成的一个数据组织，这并不是对象型的

JDBC的操作方式是也不是面向对象的，整个过程面向对象编程的思维观念很大程度上被遏制了

所以，尽管JDBC将应用程序与数据库驱动进行解耦，应用程序开发者面向JDBC进行编程，而不需要面向数据库进行编程

但是谁也没办法否认Java是纯粹的面向对象，所以在对象与关系型数据库的字段之间，又缺少了一层，这层用于将字段与对象进行映射对照

没有这层功能，只能是应用程序开发者借助于JDBC自己手动的将字段组装成对象，很繁琐，而且，不成规范，就如同没有JDBC之前开发数据库操作的程序那样，需要自己实现。

简言之，关系型数据库不是面向对象的，而JAVA却是纯粹的面向对象的语言，这势必不能很流畅的合作，JDBC对象的映射全靠自己

ORM

鉴于以上提出来的问题，在使用Java开发时，我们希望真正的建立一个对象型数据库，或者说至少使用起来看起来像一个对象型数据库

但是，目前常用的数据库又的确是关系型数据库，这一点短期内又无法改变

所以出现了ORM，对象关系映射（Object Relational Mapping，简称ORM，或O/RM，或O/R mapping）

面向对象是从软件工程基本原则（如耦合、聚合、封装）的基础上发展起来的，而关系数据库则是从数学理论发展而来的，两套理论存在显著的区别。

而面向对象的编程思想是软件开发的一大趋势，而关系数据库也是目前的必然存在，两种理论的差别的不匹配，造就了ORM，乱世出英雄。

ORM到底做什么？

JDBC将应用程序开发者与底层数据库驱动程序进行解耦，作为中间层承上启下

而ORM是插入在应用程序与JDBC API之间的一个中间层，JDBC并不能很好地支持面向对象的程序设计

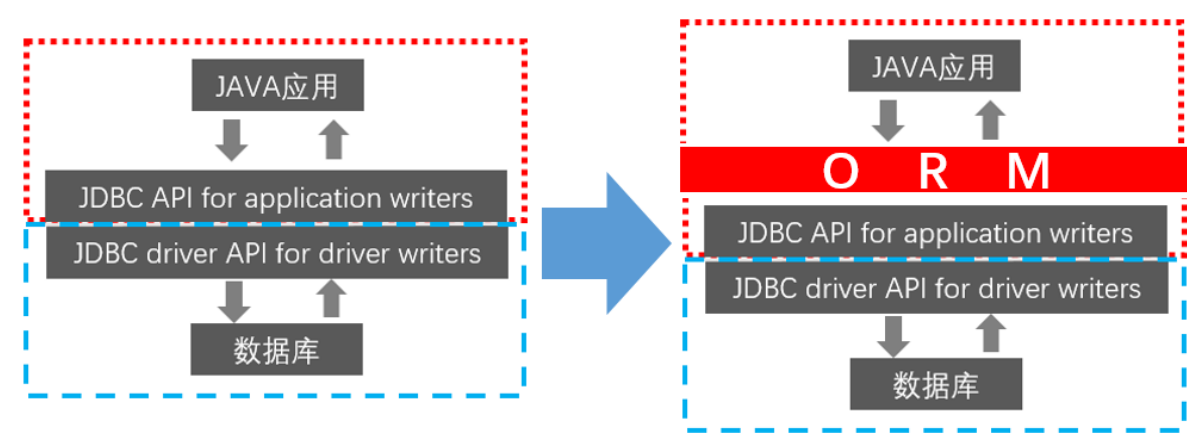
ORM解决了这个问题，通过JDBC将字段高效的与对象进行映射

应用程序开发人员不再需要直接与JDBC API进行打交道了，可以使用更加便利的ORM工具，提高开发效率

所以ORM是干什么的？

ORM用于完成Java对象与关系型数据库的映射，是JDBC的一层封装，提高了易用性。

简言之，ORM工具就是JDBC的封装，简化了JDBC的使用，完成关系型数据库中数据与Java对象的映射。



ORM工具框架最大的核心就是封装了JDBC的交互，你不在需要处理结果集中的字段或者行或者列

借助于ORM可以快速进行开发，而无需关注JDBC交互细节

但是既然是JDBC的封装，多一层封装，就势必会带来性能的开销，这是无法回避的事实，不过现在技术不断发展，性能开销越来越小。

从上面的解释看，好似有些狭义，会认为ORM框架仅仅完成对象的映射，其实并不然，ORM最原始的是一个概念，所有的ORM产品是基于ORM思想的实现实体

他们往往都附加了更多的功能，比如很多ORM框架也叫做持久化ORM框架

什么意思呢？

持久化简单理解就是脱离内存可以独立保存，保存到数据库，保存到文件等等形式，都是持久化

“持久化ORM框架”中的持久化一般是指保存到数据库，所以说如果一个ORM提供了CRUD操作API，应用程序可以借助于ORM完成数据持久化的操作，这就算是一个持久化ORM框架

就如同很多DataSource的实现中添加了很多功能，有些就直接被叫做数据库连接池

所以说具体怎么讲，都是字面的含义，真正需要做的是理解ORM思想的含义：

完成对象与关系型数据库的映射，封装底层与数据库的交互，并且很多都提供了强大的附加功能，比如持久化

现在的ORM基本上都是包括对持久类对象进行CRUD操作的API

对于Java来说，常用的有Hibernate和Mybatis (iBatis) 还有Spring JDBC等，在ORM核心思想的基础上周边又做了很多事情

所以说基本上很少有人直接使用原生的JDBC，可能有的公司中不会使用这些框架，因为毕竟框架的引入会牺牲性能

而且框架是作为JDBC的封装，就好比一个工具类，而且是别人封装的工具类，终归有些地方可能有的人用的不顺手，或者说不适合有些场景，大公司有些会自己研发一套自己需要的类ORM工具，自己使用

ORM框架各有千秋利弊，你可以不用各种已成的框架，但是，没有任何人可以否定ORM背后的思想，

ORM会一定程度上降低性能但是借助于代码生成工具等可以极大地提高开发效率

而且，ORM工具有极强的可维护性，虽然会降低性能，但是更多的时候可能是代码不够完美，算法不够高明，逻辑不够清晰，所以负责任的说ORM在很多场景都是很好的一种选择。

原文地址:[JDBC与ORM发展与联系 JDBC简介 \(九\)](#)

如果本文对您有些许帮助

可以点击下方 [好文要顶](#) 或者 [关注我](#) 支持一下~

予人玫瑰，手留余香~



分类: [05.JDBC简介](#)

好文要顶

关注我

收藏该文



noteless

关注 - 9

粉丝 - 326

+加关注

« 上一篇: [Data Source与数据库连接池简介 JDBC简介 \(八\)](#)

» 下一篇: [第一个Mybatis程序示例 Mybatis简介 \(一\)](#)

posted @ 2019-01-29 08:37 noteless 阅读(17591) 评论(3) 编辑 收藏

评论列表

#1楼 2019-01-29 11:22 暗夜追月

如果只是针对单表操作都还好说。然而SQL语句支持复杂的多表操作，还有一些聚合函数等。这时候还是写SQL比较方便。

支持(0) 反对(0)

#2楼 [楼主] 2019-01-30 08:50 noteless

@ 暗夜追月

目前没有人会直接使用JDBC，最少也会有一个DBUtils的工具类，JdbcTemplate就相当于DBUtils，Mybatis则是更进一步的封装，但是从SQL的视角看仍旧是轻量级的封装，复杂的SQL、多表关联、子查询、聚合函数等在Mybatis都是可以的，而且借助于ORM框架，可以更加高效的进行开发以及维护

支持(1) 反对(0)

#3楼 2019-04-19 09:42 Michael翔

好文，赞！已推荐！

支持(0) 反对(0)