1.0.23 jOOQ - The easiest way to write SQL in Java

一、背景

Mybatis(plus)在国内是使用最多的主流框架，而jOOQ、Ebean等小众框架则知道的人不多，但也有很多独特的优点；JPA则是一组Java持久层Api的规范，Spring Data JPA是JPA Repository的实现，本来和Hibernate、Mybatis、jOOQ之类的框架不在同一个层次上，但引入Spring Data JPA之类的框架之后，我们会直接使用JPA的API查询更新数据库，就像我们使用Mybatis一样，所以这里也把JPA和其他框架放在一起进行比较。同样，JDBC和其他框架也在同一层次，位于所有持久框架的底层，而Spring JDBC Template部分消除了使用JDBC的繁琐细节，降低了使用成本，快手的项目中也在大量使用。

- SQL封装和性能

在使用Hibernate的时候，我们查询的是POJO实体类，而不再是数据库的表，例如hql语句 select count(*) from User，里面的User是一个Java类，而不是数据库表User。这符合ORM最初的理想，ORM认为我们使用OO的思维方式，和关系数据库的思维方式差距巨大，为了填补对象和关系思维方式的鸿沟，必须做一个对象到关系的映射，然后在Java的对象世界中，我们可以使用纯的对象的思维方式，查询POJO对象，查询条件是对象属性，不再需要有任何表、字段等关系的概念，这样就更容易做持久层的操作。

JPA可以视为Hibernate的儿子，也继承了这个思路，把SQL彻底封装起来，让我们看不到关系的概念，用纯的面向对象思想，重新创造一个新的查询语言代替sql，比如hql，还有JPQL等。支持JPA的框架，例如Ebean都属于这种类型的框架。

但封装SQL，使用另一种纯的面向对象查询语言代替sql，真的能够让我们更容易实现持久层操作吗？MyBatis的流行证明了事实并非如此，至少在大多数情况下，使用hql并不比使用sql简单。首先，从很多角度上看，hql/JPQL等语言更加复杂和难以理解；其次就是性能上明显降低，速度更慢，内存占用巨大，而且还不好优化。最为恼火的是，当关系的概念被替换为对象的概念之后，查询语言的灵活性变得很差，表达能力也比sql弱很多。写查询语句的时候受到各种各样的限制，一个典型的例子就是多表关联查询。

不管是hibernate还是jpa，表之间的连接查询，被映射为实体类之间的关联关系，这样，如果两个实体类之间没有（实现）关联关系，你就不能把两个实体（或者表）join起来查询。这是很恼火的事情，因为我们很多时候并不需要显式定义两个实体类之间的关联关系就可以实现业务逻辑，如果使用hql，只是为了join我们就必须在两个实体类之间添加代码，而且还不能逆向工程，如果表里面没有定义外键约束的话，逆向工程会把我们添加的关联代码抹掉。

MyBatis则是另外一种类型的持久化框架，它没有封装SQL也没有创建一种新的面相对象的查询语言，而是直接使用SQL作为查询语言，只是把结果填入POJO对象而已。使用sql并不比hql和JPQL困难，查询速度快，可以灵活使用任意复杂的查询只要数据库支持。从SQL封装角度上看，MyBatis比Hibernate和JPA成功，SQL本不该被封装和隐藏，让Java我们使用SQL既不麻烦也更容易学习和上手，这应该是MyBatis流行起来的重要原因。

轻量级持久层框架jOOQ也和MyBatis一样，直接使用SQL作为查询语言，比起MyBatis，jOOQ虽然知名度要低得多，但jOOQ不但和MyBatis一样可以利用SQL的灵活性和高效率，通过逆向工程，jOOQ还可以用Java代码来编写SQL语句（DSL），利用IDE的代码自动补全功能，自动提示表名和字段名，减少我们记忆负担，还可以在元数据发生变化时发生编译错误，提示我们修改相应的SQL语句。

Ebean作为一种基于JPA的框架，它也使用JPQL语言进行查询，多数情况下会让人很恼火。但据说Ebean不排斥SQL，可以直接用SQL查询，也可以用类似jOOQ的DSL方式在代码中构造SQL语句（还是JPQL语句？），但没用过Ebean，所以具体细节不清楚。

JDBC Template就不用说了，它根本没做ORM，当然是纯SQL查询。利用Spring框架，可以把JDBC Template和JPA结合起来使用，在JPA不好查询的地方，或者效率低不好优化的地方使用JDBC，缓解了Hibernate/JPA封装SQL造成的麻烦，但结合大家在项目中使用它的真实感受，它真的不太好用。

- DSL和变化适应性

为了实现复杂的业务逻辑，不论是用SQL还是hql或者JPQL，我们都不得不写很多简单的或者复杂的查询语句，ORM无法减少这部分工作，最多是用另一种面向对象风格的语言去表达查询需求，如前所述，用面向对象风格的语言不见得比SQL更容易。通常业务系统中会有很多表，每个表都有很多字段，即便是编写最简单的查询语句也不是一件容易的事情，需要记住数据库中有哪些表，有哪些字段，记住有哪些函数等。写查询语句很多时候成为一件头疼的事情。

QueryDSL、jOOQ、Ebean甚至MyBatis和JPA都设计一些特性，帮助开发人员编写查询语句，有人称之为“DSL风格数据库编程”。最早实现这类功能的可能是QueryDSL，把数据库的表结构逆向工程为java的类，然后可以让java我们能够用java的语法构造出一个复杂的查询语句，利用IDE的代码自动补全功能，可以自动提示表名、字段名、查询语句的关键字等，很成功的简化了查询语句的编写，免除了我们记忆各种名字、函数和关键字的负担。

QueryDSL有很多版本，但用得多的是QueryDSL JPA，可以帮助开发人员编写JPQL语句，如前所述，JPQL语句有很多局限不如SQL灵活高效。后来的jOOQ和Ebean，基本上继承了QueryDSL的思路，Ebean基本上还是JPA风格的ORM框架，虽然也支持SQL，但不清楚其DSL特性是否支持SQL语句编写，在官网上看到的例子都是用于构造JPQL语句。

这里面最成功的应该是jOOQ，和QueryDSL不同，jOOQ的DSL编程是帮助开发人员编写SQL语句，抛弃累赘的ORM概念，jOOQ这个功能非常轻小，非常容易学习和使用，同时性能也非常好，不像QueryDSL和Ebean，需要了解复杂的JPA概念和各种奇异的限制，jOOQ编写的就是普通的SQL语句，只是把查询结果填充到实体类中（严格说jOOQ没有实体类，只是自动生成的Record对象），jOOQ甚至不一定要把结果转换为实体类，可以让开发人员按照字段取得结果的值，相对于JDBC，jOOQ会把结果值转换为合适的Java类型，用起来比JDBC更简单。

传统主流的框架对DSL风格支持得很少，Hibernate里面基本上没有看到有这方面的特性。MyBatis提供了"SQL语句构建器"来帮助开发人员构造SQL语句，但和QueryDSL/jOOQ/Ebean差很多，不能提示表名和字段名，语法也显得累赘不像SQL。

JPA给人的印象是复杂难懂，它的MetaModel Api继承了特点，MetaModel API+Criteria API，再配合Hibernate JPA 2 Metamodel Generator，让人有点QueryDSL JPA的感觉，只是绕了一个大大的弯，叠加了好几层技术，最后勉强实现了QueryDSL JPA的简单易懂的功能。很多人不推荐JPA+QueryDSL的用法，而是推荐JPA MetaModel API+Criteria API+Hibernate JPA 2 Metamodel Generator的用法，让人很难理解，也许是因为这个方案是纯的标准的JPA方案。

数据库DSL编程的另一个主要卖点是变化适应性强，数据库表结构在开发过程中通常会频繁发生变化，传统的非DSL编程，字段名只是一个字符串，如果字段名或者类型改变之后，查询语句没有相应修改，编译不会出错，也容易被开发人员忽略，是bug的一个主要来源。DSL编程里面，字段被逆向工程为一个java类的属性，数据库结构改变之后，作为java代码一部分的查询语句会发生编译错误，提示开发人员进行修改，可以减少大量bug，减轻测试的负担，提高软件的可靠性和质量。

- 安全性

一般来说，拼接查询语句都会有安全隐患，容易被sql注入攻击。不论是jdbc，还是hql/JPQL，只要使用拼接的查询语句都是不安全的。对于JDBC来说，使用参数化的sql语句代替拼接，可以解决问题。而JPA则应该使用Criteria API解决这个问题。

对于jOOQ之类的DSL风格框架，最终会被render为参数化的sql，天生免疫sql注入攻击。Ebean也支持DSL方式编程，也同样免疫sql注入攻击。

这是因为DSL风格编程参数化查询比拼接字符串查询更简单，没人会拼接字符串。而jdbc/hql/JPQL拼接字符串有时候比参数化查询更简单，特别是jdbc，很多人会偷懒使用不安全的方式。

- 跨数据库移植（工作中极少遇到）

Hibernate和JPA使用hql和JPQL这类数据库无关的中间语言描述查询，可以在不同数据库中无缝移植，移植到一个SQL有巨大差别的数据库通常不需要修改代码或者只需要修改很少的代码。Ebean如果不使用原生SQL，而是使用JPA的方式开发，也能在不同数据库中平滑的移植。

MyBatis和jOOQ直接使用SQL，跨数据库移植时都难免要修改SQL语句。这方面MyBatis比较差，只有一个动态SQL提供的特性，对于不同的数据库编写不同的sql语句。

jOOQ虽然无法像Hibernate和JPA那样无缝移植，但比MyBatis好很多。jOOQ的DSL很大一部分是通用的，例如分页查询中，Mysql的limit/offset关键字是很方便的描述方式，但Oracle和SQLServer的SQL不支持，如果我们用jOOQ的DSL的limit和offset方法构造SQL语句，不修改移植到不支持limit/offset的Oracle和SQLServer上，我们会发现这些语句还能正常使用，因为jOOQ会把limit/offset转换成等价的目标数据库的SQL语句。jOOQ根据目标数据库转换SQL语句的特性，使得在不同数据库之间移植的时候，只需要修改很少的代码，明显优于MyBatis。

JDBC Template应该最差，只能尽量使用标准sql语句来减少移植工作量。

二、优势

- fluent api、多表联查较方便
- 编译期校验、提高项目质量
- spring背书（spring-boot-starter-jooq）（spring 5和spring boot 2带来（使用）了大量好用的新特性，除了响应式编程、Kotlin支持、Lettuce、Caffeine Cache等之外，也引入了对jOOQ的支持）

三、接入方法

在项目中使用jOOQ作为持久层框架