用Spring Data JPA构建你的关系数据库访问

Summary

本文通过一个构建一个Spring Data JPA的startup工程，讨论了在实际开发中如何选取时候自己的Spring Data JPA特性来构建自己的应用， 选取原则及时简单实用。在这一原则指导下， 讨论的基于方法名和@Query注解的查询方法， 分页和排序，用QueryByExample构建动态查询以及自定义数据访问的实现。同时还给出了实战化的测试用例构建方法。力求用这些特性的讨论覆盖95%以上的日常开发场景。

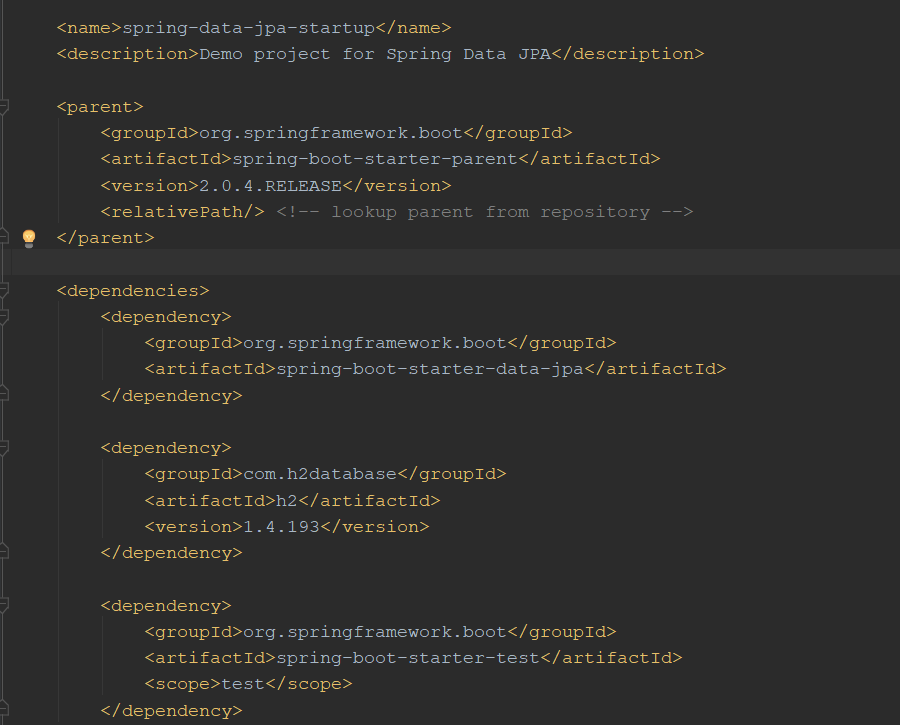
Spring Data是Spring提供的数据访问的一个通用中间层，Spring Data JPA作为Spring Data的成员项目，提供了以JPA方式访问关系数据库的接入方式。Spring Data JPA使得构建基于JPA的DAO访问层变得十分容易。作为一个在项目中使用Spring JDBC Template访问数据库多年的人儿来说，Spring Data JPA给人以耳目一新的感觉，不用写Repository实现类了诶。在与Spring Boot相结合后， 初设设置变得更加精简。

“简单是最大的红利”，通过对Spring Data JPA简化开发方便给我带来的便利性， 本文主要从一下几个方面谈论Spring Data JPA的使用

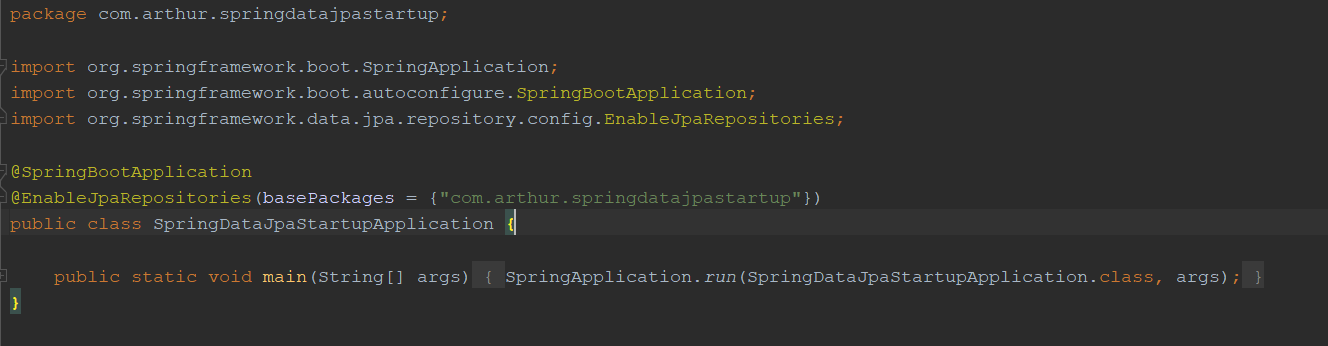
1. 初始构建
2. 查询的构建
3. 对分页和排序的支持
4. 动态查询
5. 用户自定义数据访问实现
6. 集成测试

Start Up

在Spring Boot 中开启一个特性都很简单， 加入相应的Maven依赖， 再使用相应的Enable标记即可。我在做见过的StartUp项目时使用了Spring Boot 2.0.4.RELEASE ， 同时依赖的spring-boot-starter-data-jpa和H2 database

`

在String Boot的Application类上添加@EnableJpaRepositories注解即可开启Spring Data JPA， Spring Boot自动扫描相应的组件， 并未Repository添加代理， 这既是我们不需要提供Repository实现类的奥妙所在。@EnableJpaRepositories注解中有一个basePackages属性， 其值会限定JPA先关扫描的范围。这对多数据提供方访问的应用特别有意义。例如一个应用的同范畴底层数据既有存在于关系数据库中，又有在文件系统中的， 对应的范文代码分别位于com.abc.app.repositore.jap 和com.abc.app.repositore.xml， 此时指定basePackages为com.abc.app.repositore.jpa就显得有意义。

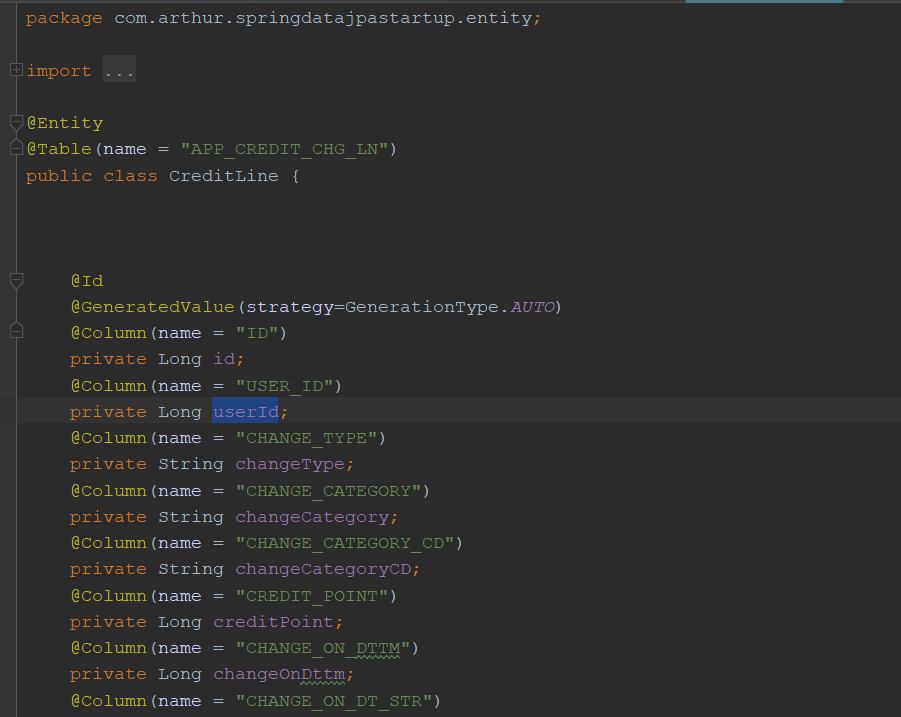


接下来配置相应的数据库连接信息和JPA属性，这里我们使用了H2的Server模式连接一个H2的文件数据库test，所以在测试时需要先让这个数据库运行起来， 具体可以参考上一篇推送。 同时设置让JPA展示SQL， 对数据库的表做结构的变化做update。



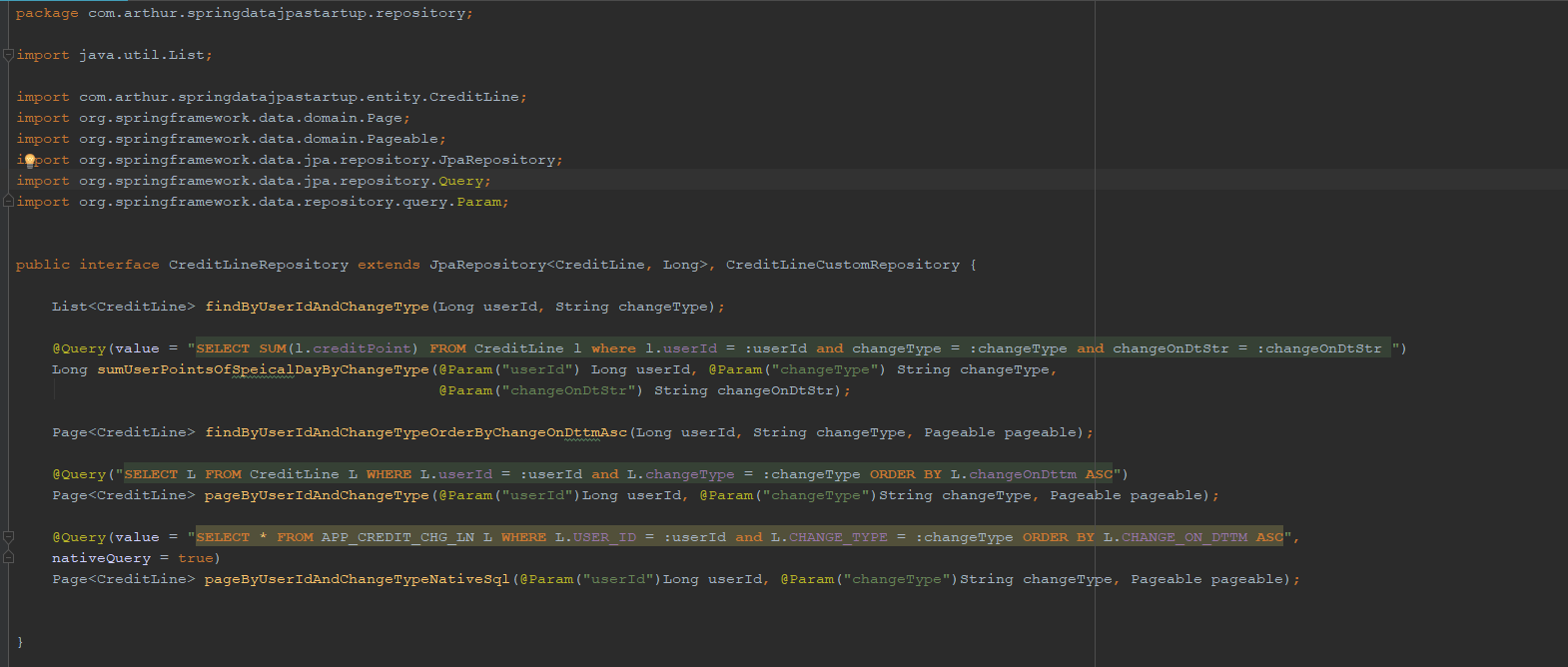
Entity

Spring Data JPA默认的Provider是hibernate， 所以ORM方面就遵循hibernate的概念。这里使用了javax.persistence的注解来实现ORM元数据的配置。这个实体的业务逻辑是记录系统用户积分的变化详情，例如签到获取积分或兑换消费积分等等。



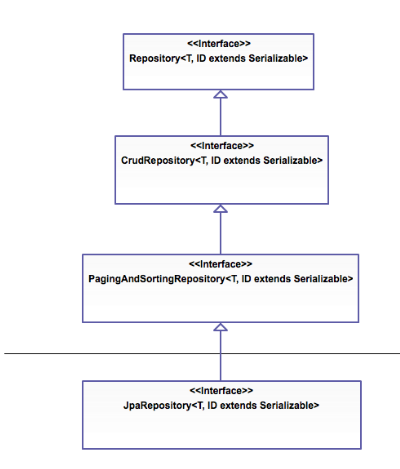
Repository

Repository是Spring对数据访问组件的抽象，在Repository中实现对数据的访问， 这里我们通过JPA访问关系数据库。由于Spring Data JPA在启动扫描时自动创建了代理类作为repository的实现， 应用开发者只需要与自己业务实体先关的数据访问接口既可以了。所以针对上一节的实体，声明一个接口，让他扩展自JpaRepository接口， 并指明泛型类型为实体类和实体类ID的类型即可。



事实上我们自己的接口可以扩展自JpaRepository，PagingAndSortingRepository和CrudRepository， (甚至是Repository)的任意一个 ，启动扫描搜索的是is-like-a Repository? 这几个接口的继承关系如下，功能上Repository是一个命名接口，标识了子类进行数据访问的功能分类，CrudRepository提供了数据增删差改的方法接口，PagingAndSortingRepository则提供了分页与排序支持，JpaRepository则在分页排序基础上增加了同JPA Provider的交互，如flush。途中的横线代表了一个依赖分隔， 横线上方是Spring Data的基础功能， 下方是Spring Data JPA提供的功能。 也就是说如果选择继承自PagingAndSortingRepository或是CrudRepository， 完全不需要使用JPA，那么使用Spring Data JPA的好处在哪里？ 答案是动态SQL的支持， 这个我们后面再谈。 业务实体数据访问接口中继承的CreditLineCustomRepository我们也在后面来谈。

结论是我们声明一个接口继承自JpaRepository来提供我们业务实体数据访问的功能。



查询构建

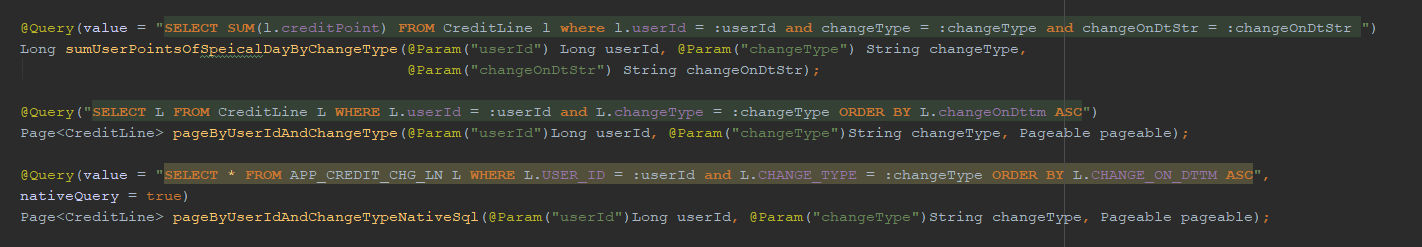
Spring Data JPA 支持基于方法名称, 命名查询(Named Query) 和 @Query注解的查询构建

Spring Data JPA中如果查询方法名遵循一定命名规范， 那么这个方法可以被直接解析为查询语句。例如下面的findByUserIdAndChangeType方法将被解析为根据userId和changeType属性的值进行等值查询。 原因是方法名以findBy开头， 后面跟着限定的属性名。这种方法还可以限定非等值约束，还可以限定排序字段及方向， 例如findByUserIdAndChangeTypeOrderByChangeOnDttmAsc会按照明细生成时间做升序排列。



基于方法名称的查询优点很明显—— 直接，方便。但面对复杂业务， 方法名变得冗长似乎不可避免，这时命名查询就显得很合理。命名查询的要义就是给查询一个Key， 查询语句本身则是这个Key附带的值， 通过在文件， 或实体类的注解中声明这一key-value的映射， 在repository中将Key作为方法名使用来获取使用，这样就无需冗长的方法名了。那问题是查询的声明和使用分离， 这样真的好吗？你真的要考验IDE搜索内容的能力，以及自己大脑记忆搜索关键字的能力吗？

基于此查询的声明和使用在一起的方式是否更有利于日常开发？Spring Data JPA通过@Query注解实现这一目的，value属性提供查询语句， 默认是JPQL (HQL的等价物，因为Hibernate是JPA的默认官方实现)， 如果提供了SQL语句， 需要将nativeQuery置为true。



分页和排序

JpaRepository继承了PagingAndSortingRepository接口， 所以可以直接使用相关功能。

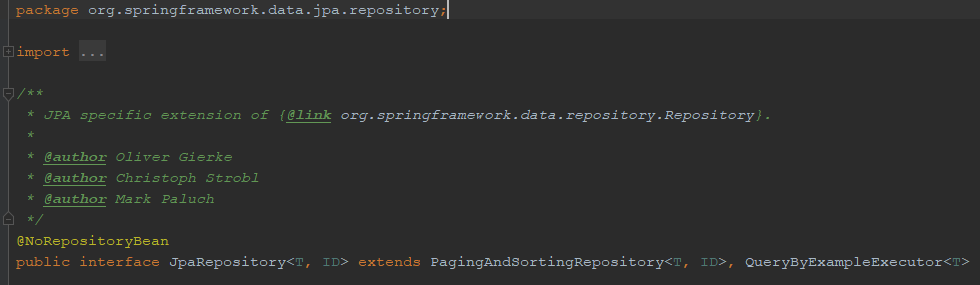
排序可以通过Sort对象指明，也可以通过查询语句直接显示声明。用Sort更OO， 用SQL更简洁， 业务简单的情况下， 代码还更少。

分页通过Pageable对象传递分页参数，Pageable也可以包含Sort对象， 通过Page对象返回分页结果。 需要说明的是有比较老的文章指出分页不支持nativeQuery， 但我的实验结果表明这种结论已经不成立了，pageByUserIdAndChangeTypeNativeSql方法说了这一点，在测试部分有相关测试用例。

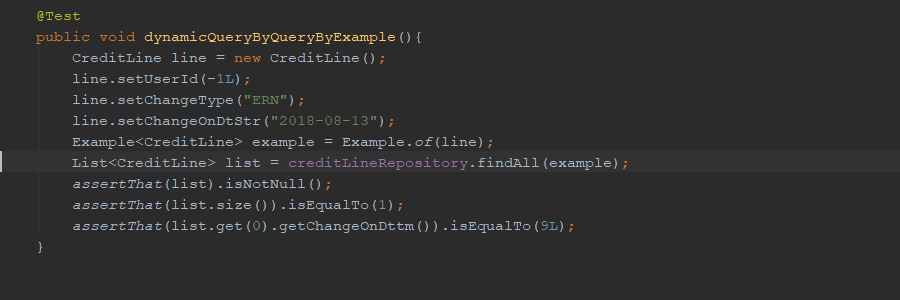
动态查询

Spring Data JPA 甚至Spring Data在一开始就支持动态查询， Spring Data用到了Querydsl API，Spring Data JPA早期则提供了JPA Criteria API。抛开JPA Criteria API可读性差， 不易构建负责查询的缺点。 这二者都需要通过工具生成meta-model的方式来构建查询， 大家都喜欢简单，都喜欢直接， 可是当你看到很多很多叫做QXXX (Querydsl 元模型), 或XXX\_ （Criteria元模型）的类时你一定不会觉得清爽的。

好消息是目前的Spring Data JPA利用QueryByExample来实现动态查询， 没有中间元模型搞事情。JpaRepository还继承了QueryByExampleExecutor，来通过Example对象构建动态查询。



这是一个使用Example构建查询的例子， 更多例子参见[这里](https://github.com/spring-projects/spring-data-examples/tree/master/jpa/query-by-example)

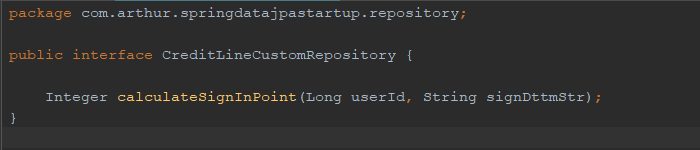


用户自定义数据访问

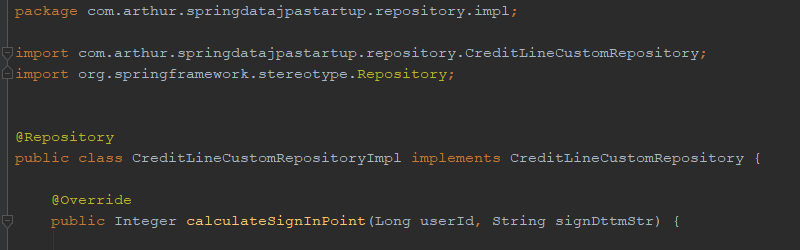
我们用查询方面解析构建简单的查询， @Query构建稍复杂的查询， 用QueryByExample构建非常复杂的查询， 我们还需要自定义的操作吗？Spring Data JPA通过自动代理对Repository进行了实现， 如何再加入自定义数据访问？

第一个问题一定是需要的， 应为业务需求总是会被应对的，总会有前面的工具解决不了或不是那么又要简单的方案。那么怎么做？三步走

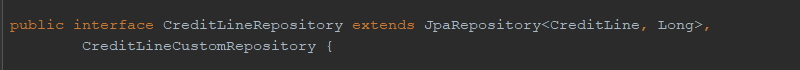
1. 声明自定义操作接口接口， 比如CreditLineCustomRepository



1. 提供实现，并标注为@Repository



1. 在初始接口中继承改自定义接口



测试

测试最关键的问题是解决测试数据的在测试前的初始化和测试后的清理工作。为此可以准备初始化脚本和清理脚本至于 test/resources/目录下，同时在@Sql注解scripts属性中通过/指明为classpath跟路径。

