**Query DSL**

Query DSL is an Java framework, which helps with creating and running type-safe queries in a domain specific language that is similar to SQL.

Querydsl defines a general statically typed syntax for querying on top of persisted domain model data. JDO and JPA are the primary integration technologies for Querydsl.

Query DSL can be used with JPA, Hibernate, SQL, MongoDB.

As Nimbus Framework is using MongoDB, so Usage of MongoDB with QueryDSL

The below dependency jar is required to use query DSL:

<dependency>

  <groupId>com.mysema.querydsl</groupId>

  <artifactId>querydsl-mongodb</artifactId>

  <version>${querydsl.version}</version>

</dependency>

Then next configure Maven APT plugin which generates the query types used by Querydsl.

MorphiaAnnotationProcessor finds domain types annotated with the com.google.code.morphia.annotations.Entity and com.google.code.morphia.annotations.Embedded annotations and generates query types for them.

Run clean install and you will get your Query types generated into target/generated-sources/java.

Now we are able to construct Querydsl query instances and instances of the query domain model.

Queryies:

For the MongoDB-module MorphiaQuery is the main Query implementation. It is instantiated like this :

// test database

String dbname = "testdb"; // Morphia configuration

Morphia morphia = new Morphia().map(Customer.class); // datastore

Datastore ds = morphia.createDatastore(dbname);

QCustomer customer = QCustomer.customer;

MorphiaQuery<Customer> query = new MorphiaQuery<Customer>(morphia, ds, customer);

Ex1 : To retrieve the customer with the first name Bob you would construct a query like this :

QCustomer customer = QCustomer.customer;

MorphiaQuery<Customer> query = new MorphiaQuery<Customer>(morphia, ds, customer);

Customer bob = query.where(customer.firstName.eq("Bob")).uniqueResult();

cascading methods of the MorphiaQuery class are:

where : Define the query filters, either in varargs form separated via commas or cascaded via the and-operator.

orderBy : Define the ordering of the result as an varargs array of order expressions. Use asc() and desc() on numeric, string and other comparable expression to access the OrderSpecifier instances.

limit, offset, restrict : Define the paging of the result. Limit for max results, offset for skipping rows and restrict for defining both in one call.