SPRING DATA

Arkadiusz Borek
Senior Developer / Team Leader
@arekborek

Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

Main modules

- Spring Data Commons - Core Spring concepts underpinning every Spring Data project.
- Spring Data JPA - Makes it easy to implement JPA-based repositories.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Provides easy configuration and access to Redis from Spring applications.
- Spring Data Solr - Spring Data module for Apache Solr.
- Spring Data Gemfire - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data KeyValue - Map-based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

Community modules

- Spring Data Aerospike - Spring Data module for Aerospike.
- Spring Data Cassandra - Spring Data module for Apache Cassandra.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.

Spring Data JPA

Spring ORM + Spring JDBC + Hibernate

Spring Boot + DataSource configuration

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=pass
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.database=POSTGRESQL
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create-drop

logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

CRUD

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);

    // … more functionality omitted.
}
```

Pagind and sorting

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

JpaRepository

GO TO CODE

Query methods

Interface:

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);

}
```

Query:

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

Keywords:
And, Or, Between, LessThan, GreaterThan, After, Before, IsNull, IsNotNull, NotNull, Like, NotLike, StartingWith, EndingWith, Containing, OrderBy, Not, In, NotIn, True, False

Using @Query

```
public interface UserRepository extends JpaRepository<User, Long> {

  //@Query("select u from User u where u.emailAddress = ?1")
  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?0", nativeQuery = true)
  User findByEmailAddress(String emailAddress);

}

public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")
  User findByLastnameOrFirstname(@Param("lastname") String lastname,
                                 @Param("firstname") String firstname);

}
```

Applying query hints

```
public interface UserRepository extends Repository<User, Long> {

  @QueryHints(value = { @QueryHint(name = "name", value = "value")},
              forCounting = false)
  Page<User> findByLastname(String lastname, Pageable pageable);

}
```

The just shown declaration would apply the configured QueryHint for that actually query but omit applying it to the count query triggered to calculate the total number of pages.

Entity

GO TO CODE

Auditing (1)

```
class Customer {

  @CreatedBy
  private User user;

  @CreatedDate
  private DateTime createdDate;

  // … further properties omitted

}
```

@CreatedBy, @LastModifiedBy, @CreatedDate, @LastModifiedDate

Auditing (2)

```
class SpringSecurityAuditorAware implements AuditorAware<User> {

  public User getCurrentAuditor() {

    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

    if (authentication == null || !authentication.isAuthenticated()) {
      return null;
    }

    return ((MyUserDetails) authentication.getPrincipal()).getUser();
  }
}



@Configuration
@EnableJpaAuditing
class Config {

  @Bean
  public AuditorAware<AuditableUser> auditorProvider() {
    return new AuditorAwareImpl();
  }
}
```

Transactionality

```
public interface UserRepository extends SimpleJpaRepository<User, Long> {

  @Override
  @Transactional(timeout = 10)
  public List<User> findAll();

  // Further query method declarations

}
```

@Transactional

The annotation supports further configuration as well:

- the **Propagation** Type of the transaction
- the **Isolation Level** of the transaction
- a **Timeout** for the operation wrapped by the transaction
- a **readOnly** flag – a hint for the persistence provider that the transaction should be read only
- the **Rollback** rules for the transaction

Note that – by default, rollback happens for runtime, unchecked exceptions only. Checked exception do not trigger a rollback of the transaction; the behavior can of course be configured with the rollbackFor and noRollbackFor annotation parameters.

Locking

```
interface UserRepository extends Repository<User, Long> {

  // Plain query method
  @Lock(LockModeType.READ)
  List<User> findByLastname(String lastname);

}
```