Spring Data JPA:
Easy Way to Query Database without SQL

Audrius Mičiulis

Spring Data

- Spring Data JPA
- Spring Data Mongo DB
- Spring Data Redis
- Spring Data Solr
- Spring Data Hadoop
- Spring Data REST
- Spring Data NEO4J
- Spring Data Cassandra
- etc.

Spring Data Dependencies

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.2.16.Final</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.7.2.RELEASE</version>
  </dependency>
</dependencies>
```

Spring Data Configuration

- Configure dependencies
- Create the properties file
- Configure the DataSource bean
- Configure the Entity Manager factory bean
- Configure the Transaction Manager bean
- Enable annotation-driven transaction management
- Configure Dependency Injection
- Configure Spring Data JPA
- Create database

CrudRepository<T, ID>

- `T findOne(ID primaryKey);`
- `Iterable<T> findAll();`
- `Long count();`
- `boolean exists(ID primaryKey);`
- `T save(T entity);`
- `void delete(T entity);`
- …

Examples

- `userRepository.findAll();`
- `userRepository.findOne(id);`
- `userRepository.count();`
- `userRepository.save(user);`
- `userRepository.exists(userId);`

PagingAndSortingRepository<T, ID>

- `Iterable<T> findAll(Sort sort);`
- `Page<T> findAll(Pageable pageable);`
- …

Examples

- `taskRepository.findAll(pageable);`

CustomRepository<T, ID>

```
@NoRepositoryBean
public interface BaseRepository<T, ID extends
Serializable> extends Repository<T, ID> {

    T findOne(ID id);

    T save(T entity);

}
```

Defining query methods

- **find…By…, count…By…, delete…By…**
- **…Distinct…**
- **…And…, …Or…**
- **…Equals…, (…Is…), …Not…, …In…, …NotIn…, …Like…, …NotLike…, …StartingWith…, …EndingWith…, …Containing…**
- **…Between…, …LessThan…, …LessThenEqual…, …GreaterThan…, …GreaterThanEqual…**
- **…After…, …Before…**
- **…IsNull…, …isNotNull, (…notNull…), True, False**
- **…IgnoreCase…, …AllIgnoreCase…**
- **…OrderBy…Asc, …OrderBy…Desc**

Examples

- `Long countByFirstName(String firstName);`
- `List<User> findByUserTypeOrderByFirstNameDesc(UserType userType);`
- `User findByFirstNameAndLastName(String firstName, String lastName);`
- `User findByFirstNameAndLastNameAllIgnoreCase(String firstName, String lastName);`
- `List<Task> findByAssignedToUserFirstNameAndAssignedToUserLastName(String firstName, String lastName);`
- `List<Task> findByAssignedToUser_FirstNameAndAssignedToUser_LastName(String firstName, String lastName);`

Limiting the results

- …**`first10`**…
- **`Pageable`**

Examples

- `List<Task> findFirst3ByCreatedByUser(User user, Sort sort);`

Custom queries

- @Query, @Param
- JPQL (Java Persistence Query Language)
- Validation at boot time

Examples

```
@Query("select t from Task t where t.number IN (:numbers)")
List<Task> findByNumberIn(@Param("numbers") Collection<String>
numbers);
```

Named queries

- @NamedQuery, @NamedNativeQuery
- <named-query name="…">
  <query>…</query>
  </named-query>

Examples

```
@Entity
@Table(schema = "core", name = "task")
@NamedQuery(name = "Task.findAssignedTasks", query = "select t
from Task t where t.assignedToUser is not null")
public class Task {
…
}
```

Custom interfaces

- Create interface
- Create implementation
- Extend created interface from repository

Examples

```
public interface CustomTaskRepository {
  String findTaskNumberByTitleFragment(String fragment);
}


@Repository
public class TaskRepositoryImpl implements CustomTaskRepository {

  @PersistenceContext
  private EntityManager entityManager;

  @Override
  public String findTaskNumberByTitleFragment(String fragment) {
    Query query = entityManager.createNativeQuery(…);
    query.setParameter("fragment", "%" + fragment + "%");
    return (String) query.getSingleResult();
  }
}
```

Spring Data JPA

- Sophisticated support to build repositories based on Spring and JPA
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of @Query annotated queries at bootstrap time
- Support for QueryDSL predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Java based repository configuration by introducing @EnableJpaRepositories

Disadvantages

- Method names are very long on complicated structures
- Native SQL can be more efficient
- No support for aggregating queries

Questions