Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Spring Data

Grzegorz Rucki

Co to jest Spring Data?

- **Założeniem biblioteki jest dostarczenie przejrzystego spójnego modelu programistycznego zapewniającego dostęp do danych.**
- **Posiada moduły obsługujące różne bazy danych, zarówno relacyjne jak i nieracyjne.**
- **Jest warstwą nabudowaną nad ORM (lub OM w przypadku baz NoSQL)**

```
Application -> Spring Data JPA -> JPA <- Hibernate -> JDBC -> DataSource

-> – uses
<- – implements
```

The same for MongoDB:

```
Application -> Spring Data MongoDB -> MongoDB Java driver -> MongoDB
```

Moduły Spring Data

Main modules

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

Moduły Spring Data

Community modules

- Spring Data Aerospike - Spring Data module for Aerospike.
- 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 Hazelcast - Provides Spring Data repository support for Hazelcast.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.

Najprostszy przypadek

Aby obsługiwać operacje na tabeli potrzebne są trzy klasy:

- Klasę mapującą tabelę
- Interface rozszerzający *CrudRepository*
- Klasę do której wstrzykujemy stworzony interface za pomocą adnotacji *@Autowired*

  (w naszym przypadku będzie to *Application* ze String Boota)

Klasa mapująca tabelę

```java
package run;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;


@Entity
public class Customer {

    private String name ;
    private String city ;
    private String email ;
    private String tel ;
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    public Customer(String name, String city, String email, String tel) {
        this.name = name;
        this.city = city;
        this.email = email;
        this.tel = tel;
    }

    public Customer() {
    }



    public String getName() {

    public void setName(String name) {
```

Interface rozszerzający *CrudRepository*

```java
package run;

import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

}
```

AGH

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

  @Autowired
  private CustomerRepository repository;

  public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
  }

  public void run(String... args) throws Exception {

        repository.save(new Customer("Nowak", "Krakow", "asd@mail","6666666"));
        repository.save(new Customer("Smith", "Londyn", "asd2@mail","123123"));

        repository.count();

        repository.findOne(2l);
        repository.findAll();

        repository.deleteAll();
  }
}
```

Metody CrudRepository

Modifier and Type	Method and Description
long	**count()** Returns the number of entities available.
void	**delete(ID id)** Deletes the entity with the given id.
void	**delete(Iterable<? extends T> entities)** Deletes the given entities.
void	**delete(T entity)** Deletes a given entity.
void	**deleteAll()** Deletes all entities managed by the repository.
boolean	**exists(ID id)** Returns whether an entity with the given id exists.
Iterable<T>	**findAll()** Returns all instances of the type.
Iterable<T>	**findAll(Iterable<ID> ids)** Returns all instances of the type with the given IDs.
T	**findOne(ID id)** Retrieves an entity by its id.
<S extends T> **Iterable<S>**	**save(Iterable<S> entities)** Saves all given entities.
<S extends T> S	**save(S entity)** Saves a given entity.

Tworzenie query na podstwie nazwy metody

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

     List<Customer> findByName(String name);
     List<Customer> findByNameOrderByTel(String name);
     List<Customer> findByNameAndCity(String name, String city);
     List<Customer> findByNameOrCity(String name, String city);
     
}
```

Tworzenie query na podstwie nazwy metody

Keyword	Sample	JPQL snippet
`And`	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
`Or`	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
`Is,Equals`	`findByFirstname`,`findByFirstnameIs`,`findByFirstnameEquals`	`… where x.firstname = ?1`
`Between`	`findByStartDateBetween`	`… where x.startDate between ?1 and ?2`
`LessThan`	`findByAgeLessThan`	`… where x.age < ?1`
`LessThanEqual`	`findByAgeLessThanEqual`	`… where x.age <= ?1`
`GreaterThan`	`findByAgeGreaterThan`	`… where x.age > ?1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
`After`	`findByStartDateAfter`	`… where x.startDate > ?1`
`Before`	`findByStartDateBefore`	`… where x.startDate < ?1`
`IsNull`	`findByAgeIsNull`	`… where x.age is null`
`IsNotNull,NotNull`	`findByAge(Is)NotNull`	`… where x.age not null`
`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended `%`)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended `%`)

Query w adnotacjach

```java
package run;

import java.util.List;

public interface CustomerRepository extends CrudRepository<Customer, Long> {


    @Query("SELECT c FROM customer c WHERE c.name = :searchName")
    public List<Customer> findByNameAnnotation(@Param("searchName") String searchName);

    @Query("SELECT c FROM customer c WHERE c.name = :searchName ORDER BY tel")
    public List<Customer> findByOrderByTelAnnotation(@Param("searchName") String searchName);

    @Query("SELECT c FROM customer c WHERE c.name = :searchName AND c.city = :searchCity")
    public List<Customer> findByNameAndCityAnnotation(@Param("searchName") String searchName,
                                                      @Param("searchCity") String searchCity);
}
```

Kontroler REST

- Spring Data umożliwia szybkie przerobienie stworzonego CrudRepository na RESTowy kontroler.
- Aby to zrobić należy użyć adnotacji @RepositoryRestResource

```
@RepositoryRestResource(collectionResourceRel = "customer", path = "/customer")
public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByName(@Param("name") String name);
    List<Customer> findByNameOrderByTel(@Param("name") String name);
    List<Customer> findByNameAndCity(@Param("name") String name, @Param("city") String city);
    List<Customer> findByNameOrCity(@Param("name") String name, @Param("city") String city);

}
```

Kontroler REST

- Poza dodawaniem, usuwaniem i edycją rekordów umożliwa korzystania z metod zdefiniowanych w stworzonym rozszerzeniu CrudRepository.

- Sposób użycia:

  GET

  localhost:9080/zti_1/customer/search/nazwaFunkcji?arg1=val1&arg2=val2

Koniec