RSQL

elastyczne przeszukiwanie baz danych

Marcin Baranowski

Wyzwanie

stworzyć generyczny i elastyczny mechanizm filtrowania rekordów wysyłanych na frontend

Generyczny mechanizm wyszukiwania

raz stworzony powinien działać bez względu na modyfikacje i/lub dodanie nowych encji

Elastyczny mechanizm wyszukiwania

- Pozwala na wyszukiwanie wartości:
    - Konkretnych (**equals**)
    - Podobnych (**like**)
    - Większych/mniejszych (lub równych) (też daty),
- Umożliwia łączenie powyższych poprzez operatory logiczne (and, or)
-Jest rozszerzalny o nowe operatory

Metody JPA

```
findByNameAndSurname(String name, String surname);

findDistinctPeopleByLastnameOrFirstname(String lastname, String
firstname);

findByLastnameIgnoreCase(String lastname);

findByHeightGreaterThanEqualAndBirthDateAfter(int height, Date
birthDate);

findByLastnameOrderByFirstnameAsc(String lastname);
```

Metody JPA

+ Proste

+ Nie wymagają dodatkowego kodu, poza nazwą metody w repozytorium

- Filtrowane wartości muszą być znane w czasie pisania kodu (niewystarczająco elastyczne rozwiązanie)

- Niedostatecznie generyczne – przy tworzeniu nowego pola lub encji trzeba tworzyć nowe metody

Criteria builder

```
public List<Employee> findByCriteria(String employeeName,String fieldName){
    return employeeRepository.findAll(new Specification<Employee>() {
        @Override
        public Predicate toPredicate(Root<Employee> root, CriteriaQuery<?> query,
                CriteriaBuilder criteriaBuilder) {

            List<Predicate> predicates = new ArrayList<>();
            if(employeeName!=null) {
                        predicates.add(criteriaBuilder
                        .and(criteriaBuilder.equal(root.get("employeeName"), employeeName)));
            }

            query.orderBy(criteriaBuilder.desc(root.get(fieldName)));

            return criteriaBuilder.and(predicates.toArray(new Predicate[predicates.size()]));
        }
    });
}
```

Criteria builder

+ Pozwala na bardziej zaawansowane zapytania niż metody JPA
+ Zachowuje informacje o typach
- Wymaga pisania wiele kodu
- Wciąż za mało elastyczny

Querydsl

```
List<Tuple> userTitleCounts = queryFactory
        .select(blogPost.title,
                blogPost.id.count().as(count))
        .from(blogPost)
        .groupBy(blogPost.title)
        .orderBy(count.desc())
        .fetch();
```

QueryDSL

+ Pozwala na bardziej złożone zapytania niż metody JPA

+ Mniej kodu, niż Criteria Builder

- Wymaga generowania dodatkowych klas (Q-klas)

-Za mało elastyczny

RSQL

Język zapytań dla restowych api oparty o FIQL

Może być użyty do przeszukiwania bazy danych

RSQL – operatory porównania

Operator	Opis
==	równy (equal/like)
!=	nie równy
=lt= lub <	mniejszy
=le= lub <=	mniejszy lub równy
=gt= lub >	większy
=ge= lub >=	większy lub równy
=in=	zawiera się w
=out=	nie zawiera się w

RSQL – pozostałe operatory

Operator	Opis
; lub and	logiczny łącznik AND
, lub or	logiczny łącznik OR
()	nawias – określa kolejność działań

RSQL - przykłady

1. `name=="Kill Bill”;year>2003`

2. `genres=in=(sci-fi,action);(director=='Christopher Nolan',actor==*Bale);year=ge=2000`

3. `director.lastName==Nolan;year>=2000;year<2010`

4. `genres=in=(scifi,action); genres=out=(romance,animated,horror),director==Que*Tarantino`

RSQL

+ Elastyczny
+ Rozszerzalny
+ Działa na poziomie zapytań rest
- Wymaga osobnej biblioteki do komunikacji z bazą danych

Implementacja

- Dodanie bibliotek w pom.xml
- Implementacja kontrolera
- Implementacja budowniczego specyfikacji

demo

Materiały

https://github.com/SilverSheep/Rsql-frontend

https://github.com/SilverSheep/Rsql-backend

Zasoby

https://github.com/jirutka/rsql-parser

https://www.baeldung.com/rest-api-search-language-rsql-fiql

https://docs.spring.io/spring-data/jpa/docs/1.6.0.RELEASE/reference/

https://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/htm

http://www.querydsl.com/