Tomasz Nurkiewicz

Spring Data JPA – jeszcze prostszy dostęp do bazy danych

Specyfikacja JPA miała za zadanie ujednolicenie różnych bibliotek realizujących mapowanie relacyjno-obiektowe w Javie, takich jak Hibernate czy EclipseLink. Spring Data JPA to niewielka biblioteka upraszczająca pracę z JPA poprzez automatyczne tworzenie kodu repozytoriów (DAO) oraz wygodną obsługę stronicowania i sortowania. Nie tylko znacząco skraca kod dostępu do danych, ale promuje też spójną strukturę tej warstwy.

W niniejszym artykule przyjrzymy się bibliotece Spring Data JPA. Pochodzi ona z szerszego katalogu projektów o nazwie Spring Data i, jak nie trudno się domyśleć, najlepiej sprawdza się w aplikacjach opartych o framework Spring. Zakładam, że posiadacie już działającą aplikację springową używającą JPA. Zastosowana implementacja zasadniczo nie ma znaczenia, przykłady w tym artykule bazują na Hibernate i bazie H2.

DODANIE BIBLIOTEKI DO PROJEKTU

Jeśli nasza aplikacja już korzysta z Java Persistence API (JPA), wprowadzenie Spring Data JPA jest bardzo proste. Warto też zauważyć, że możemy wykorzystywać i migrować istniejący kod na tę bibliotekę stopniowo. Przede wszystkim potrzebujemy następującej zależności:

Listing 1. Zależność do Spring Data JPA (maven)

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
  <version>1.5.1.RELEASE</version>
</dependency>
```

Następnie wystarczy włączyć bibliotekę prostą dyrektywą w Javie:

Listing 2. Włączenie biblioteki w Javie

```
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.
EnableJpaRepositories;

@Configuration
@EnableJpaRepositories(basePackages = "com.nurkiewicz.jpa")
public class MyConfiguration {
  //...
}
```

albo w XML-owej konfiguracji:

Listing 3. Włączenie biblioteki przy użyciu XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/data/jpa
      http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

  <repositories base-package="com.nurkiewicz.jpa" />

</beans:beans>
```

Od tej chwili możemy się cieszyć wsparciem tej biblioteki. Przyjrzyjmy się zatem, dlaczego Spring Data JPA powinna stanowić niemal nieodzowny element każdej aplikacji, korzystając z JPA.

REPOZYTORIA

Centralnym interfejsem dostarczanym przez bibliotekę jest `JpaRepository<T>`, którego instancję generuje biblioteka dla każdego obiektu domenowego (`@Entity`) typu `T`. W rzeczywistości interfejs ten jest nieco bardziej skomplikowany, tutaj uproszczony na potrzeby artykułu:

Listing 4. Metody interfejsu `JpaRepository`

```
interface JpaRepository<T, ID> {
  boolean exists(ID id);
  long count();
  T findOne(ID id);
  T getOne(ID id);
  T save(T entity);
  List<T> save(Iterable<T> entities);
  T saveAndFlush(T entity);
  List<T> findAll();
  List<T> findAll(Sort sort);
  Page<T> findAll(Pageable pageable);
  List<T> findAll(Iterable<ID> ids);
  void delete(ID id);
  void delete(T entity);
  void delete(Iterable<T> entities);
  void deleteAll();
  void deleteInBatch(Iterable<T> entities);
  void deleteAllInBatch();
  void flush();
}
```

Typ `T` reprezentuje typ encji (z reguły odpowiadającej jednej tabeli w bazie danych), a typ `ID` odpowiada kluczowi głównemu. Omówmy pokrótce, do czego służą poszczególne metody oraz dlaczego są one wygodniejsze niż stosowanie bezpośrednio interfejsu `EntityManager`:

- `exists()` sprawdza, czy rekord o podanym kluczu głównym istnieje
- `count()` zwraca ilość rekordów w tabeli
- `findOne()` wyszukuje obiekt o podanym kluczu głównym, zwracając `null` jeśli nie istnieje
- `getOne()` leniwie wyszukuje obiekt, pod jego nieobecność rzuca (także leniwie) `EntityNotFoundException`
- `save()` zapisuje encję – w zależności od jej stanu skutkuje albo aktualizacją (`UPDATE`), albo dodaniem nowego wiersza (`INSERT`). Przeciążona wersja tej metody zapisuje większy zbiór encji za jednym razem, potencjalnie wydajniej. Z kolei `saveAndFlush()` wymusza wysłanie zapytania `UPDATE/INSERT` do bazy, oczywiście sam zapis czeka do zatwierdzenia transakcji
- `findAll()` zwraca wszystkie rekordy z danej tabeli. Warto przyjrzeć się parametrom oraz wartościom zwracanym. O ile pierwsza wersja zwraca wszystkie rekordy bez ich stronicowania i sortowania (jest zatem mało użyteczna, a czasem nawet niebezpieczna), to pozostałe wersje `findAll()` są znacznie ciekawsze. Możliwe jest bowiem nie tylko deklaratywne posortowanie wszystkich rekordów (parametr `Sort`), ale nawet zażądanie konkretnej strony (`Pageable`). W tym drugim przypadku jako rezultat otrzymujemy nie `List<T>`, ale znacznie bogatszy obiekt `Page<T>`. Za chwilę przekona-