ВИКОРИСТАННЯ JPA В SPRING

ІСТОРІЯ JPA

У попередніх версіях EJB визначався рівень персистентності поєднувався з рівнем бізнес-логіки за допомогою інтерфейсу javax.ejb.EntityBean.

Під час введення EJB 3.0 рівень персистентності був відокремлений і вказаний як JPA 1.0 (Java Persistence API). Специфікації цього API були опубліковані разом зі специфікаціями JAVA EE5 11 травня 2006 р. Із використанням JSR 220.

JPA 2.0 була випущена зі специфікаціями JAVA EE6 10 грудня 2009 року як частина Java Community Process JSR 317.

JPA 2.1 був випущений із специфікацією JAVA EE7 22 квітня 2013 р. З використанням JSR 338.

SPRING DATA JPA

- Spring Data JPA, частина сімейства Spring Data, дозволяє легко впровадити сховища на основі JPA. Цей модуль стосується розширеної підтримки рівнів доступу до даних на основі JPA. Це спрощує створення Spring-програм, що використовують технології доступу до даних.
- Для виконання простих запитів, а також виконання пагінації та аудиту потрібно написати занадто багато типового коду. Spring Data JPA має на меті значно покращити реалізацію рівнів доступу до даних, зменшивши зусилля до необхідної кількості. Розробник пише власні інтерфейси сховища, включаючи користувацькі методи пошуку, а Spring забезпечує реалізацію автоматично.

РОЗМІТКА СУТНОСТІ

@Entity	вказує, що клас оголошується як сутність.
@Table	Ця анотація вказує на оголошення імені таблиці.
@Basic	У цій анотації явно вказуються поля без обмежень.
@Embeded	вказує властивості вбудованого класу або об'єкта
@Id	вказує первинний ключ таблиці класу.
@GeneratedValue	вказує, як атрибут ідентичності може бути ініціалізований, наприклад, автоматичний, ручний або значення взято з таблиці послідовності.
@Transient	Ця анотація вказує властивість, яка не є постійною, тобто значення ніколи не зберігається в базі даних.
@ Column	використовується, щоб вказати стовпець або атрибут для властивості.

РОЗМІТКА СУТНОСТІ

@SequenceGenerator	використовується для визначення значення властивості, яке вказано в анотації @GeneratedValue, створює послідовність.
@TableGenerator	використовується для визначення генератора значень для властивості, зазначеної в анотації @GeneratedValue, створює таблицю.
@AccessType	використовується для встановлення типу доступу. Якщо ви встановите @AccessType (FIELD), то буде мати місце поле для доступу. Якщо ви встановите @AccessType (PROPERTY), тоді буде відбуватися оцінка властивості.
@JoinColumn	використовується для визначення асоціації об'єктної сутності або сукупності об'єктів, використовується в асоціаціях багато до одного і один до багатьох.
@UniqueConstraint	використовується для визначення поля, унікального обмеження для первинної або вторинної таблиці.
@ColumnResult	Ця анотація посилається на назву стовпця в запиті SQL за допомогою пункту select.

РОЗМІТКА СУТНОСТІ

@ManyToMany	використовується для визначення зв’язку багато-до-багатьох між таблицями приєднання.
@ManyToOne	використовується для визначення зв’язку багато-до-одного між таблицями приєднання.
@OneToMany	використовується для визначення зв’язку один-до-багатьох між таблицями приєднання.
@OneToOne	використовується для визначення відношення один-до-одного між таблицями приєднання.
@NamedQueries	використовується для визначення списку іменних запитів.
@NamedQuery	використовується для визначення запиту з використанням статичного імені.

РОЗМІТКА СУТНОСТІ

<entity-mappings>: тег визначає визначення схемисутності у файлі xml.

<description>: тег визначає опис програми.

<entity>: тег визначає клас сутності, який ви хочете перетворити в таблицю в базі даних. Клас атрибутів визначає назву класу сутності POJO.

<table>: тег визначає назву таблиці, якщо не вказати, то переноситься ім'я класу.

<attributes>: тег визначає атрибути (поля в таблиці).

<id>: тег визначає первинний ключ таблиці, <generated-value> визначає, як присвоїти значення первинного ключа, наприклад Automatic, Manual, Sequence.

<basic>: тег використовується для визначення решти атрибутів таблиці.

<column-name>: тег використовується для визначення користувацького імені поля таблиці

ПАКЕТИ JPA

- *Repository<T, ID extends Serializable>* interface is a marker interface that has two purposes:
  - It captures the type of the managed entity and the type of the entity's id.
  - It helps the Spring container to discover the "concrete" repository interfaces during classpath scanning.
- *CrudRepository<T, ID extends Serializable>* interface provides CRUD operations for the managed entity.
- *PagingAndSortingRepository<T, ID extends Serializable>* interface declares the methods that are used to sort and paginate entities that are retrieved from the database.
- *QueryDslPredicateExecutor<T>* interface is not a "repository interface". It declares the methods that are used to retrieve entities from the database by using QueryDsl *Predicate* objects.

ПАКЕТИ JPA

- The *JpaRepository<T, ID extends Serializable>* interface is a JPA specific repository interface that combines the methods declared by the common repository interfaces behind a single interface.
- The *JpaSpecificationExecutor<T>* interface is not a “repository interface”. It declares the methods that are used to retrieve entities from the database by using *Specification<T>* objects that use the JPA criteria API.
- Швидкий старт приклад зі spring.io tutorials (https://spring.io/guides/gs/accessing-data-jpa/)

ПРИКЛАД

Звязок JPA-репозиторію з MVC-контроллером:

Оголошуємо інтерфейс для JPA-репозиторію

```
public interface CountryRepository extends CrudRepository<Country , Integer> {
}
```

Створюємо сервіс для роботи з ним

```
@Service("countryService")
public class CountryService {
@Autowired
CountryRepository countryRepository;
```

ПРИКЛАД

```
@Transactional
public List getAllCountries() {
List countries=new ArrayList();
  Iterable countriesIterable=countryRepository.findAll();
  Iterator countriesIterator=countriesIterable.iterator();
  while(countriesIterator.hasNext())
  {   countries.add(countriesIterator.next());  }
  return countries;
        }
}
```

ПРИКЛАД

Додаємо екземпляр в контроллер

```
@Controller
public class CountryController {
@Autowired
CountryService countryService;
@RequestMapping(value = "/getAllCountries", method = RequestMethod.GET,
headers = "Accept=application/json")
public String getCountries(Model model) {
    List listOfCountries = countryService.getAllCountries();
  model.addAttribute("country", new Country());
  model.addAttribute("listOfCountries", listOfCountries);
  return "countryDetails";
}
```