Spring Data Projesi

harezmi bilişim çözümleri

Spring Data Projesi Nedir?

JAVA Eğitimleri

- **Farklı veri erişim teknolojileri** ile Spring tabanlı programlama modeli üzerinden çalışmayı sağlayan bir projedir
- Arka taraftaki **veri erişim teknolojisine has özellikler**i de kaybetmeden çalışmayı hedefler
- **İlişkisel ve ilişkisel olmayan veri tabanları**, map-reduce frameworkleri, bulut tabanlı veri servislerini destekler

Spring Data Projesinin Özellikleri

- **Repository** ve **custom-object mapping** kabiliyeti sunar
- Repository metot isimlerinden **dinamik sorgu** üretebilir
- Temel bazı özelliklere sahip **domain sınıflarının hazır implementasyonları** içerir
- Custom **repository kodunu entegre edebilme** imkanı sunar
- Spring ile yakından entegredir

Repository

- Repository sınıflarının implement ettiği **marker arayüz**dür
- Temel amacı repository tarafından yönetilecek **domain sınıfının ve PK değerinin tipi**ni belirlemektir

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

}
```

CrudRepository

- Yönetilen domain sınıfı üzerinde **CRUD işlemleri** yapmayı sağlayacak bir arayüz tanımlar

```
public interface CrudRepository<T, ID extends Serializable>
                        extends Repository<T, ID> {
    <S extends T> S save(S entity);
    <S extends T> Iterable<S> save(Iterable<S> entities);
    T findOne(ID id);
    boolean exists(ID id);
    Iterable<T> findAll();
    Iterable<T> findAll(Iterable<ID> ids);
    long count();
    void delete(ID id);
    void delete(T entity);
    void delete(Iterable<? extends T> entities);
    void deleteAll();
}
```

PagingAndSorting Repository

- CrudRepository arayüzünden türeyen ve entity'lere erişimi **paging** ile kolaylaştırmayı hedefleyen bir arayüzdür

```
public interface PagingAndSortingRepository<T, ID extends
Serializable>
                    extends CrudRepository<T, ID> {

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

Sort nesnesi içerisinde hangi property'lerin ASC veya DESC sıralanarak getirileceği tanımlanır

Sorgu sonucu belirtilen sayfadaki entity'leri içerir

Pageable nesnesi paging hakkında pageNumber, pageSize, sort, next, previous page gibi bilgiler içerir

Veri Erişim Teknolojisine Özel Repository'ler

- **JpaRepository**, **MongoRepository** gibi alt arayüzler mevcuttur
- Bu arayüzler bahsedilen **temel repository kabiliyetleri**ni sunarlar
- Ayrıca **veri erişim teknolojisine has kabiliyetler** içerirler

Sorgu Metotları

- Repository nesnelerinin sunacağı sorguların metotları **her bir entity için ayrı bir Repository arayüzü**nde tanımlanır
- Spring Data bu arayüzleri implement eden proxy nesneler üreterek **sorgu metotlarını** veri erişim teknolojisine göre **dinamik olarak oluşturur**

```
public interface VetRepository extends JpaRepository<Vet, Long> {
    List<Vet> findByLastname(String lastname);
}
```

Sorgu Metotları

- Repository proxy metotlarından sorguyu üretmek için iki farklı yöntem vardır
  - Sorguyu doğrudan **metot isminden üretmek**
  - Manuel olarak **tanımlanmış sorguyu kullanmak**
- Bu yöntemler ilgili veri erişim teknolojisine göre mevcut olabilir veya olmayabilir

Metot İsminden Sorgu Üretmek

- Mekanizma sorgu metodunun isminden **find...By**, **read...By**, **query...By**, **get...By** gibi önekleri çıkarır
- İlk “**By**” ifadesinden sonraki kısım “**where clause**”unu oluşturur
- “**And**” ve “**Or**” ifadeleri ile ayrılmış bölümler de “**condition**”ları oluşturur

Metot İsminden Sorgu Üretmek

JAVA Eğitimleri

```
public interface VetRepository extends JpaRepository<Vet, Long> {

    List<Vet> findByLastName(String lastName);

    List<Vet> findByFirstNameAndLastName(
                            String firstName, String lastName);

    List<Vet> findByLastNameOrderByFirstNameAsc(String lastName);

    List<Vet> findByLastNameOrderByFirstNameDesc(String lastName);

    List<Vet> findByFirstNameAndLastNameIgnoreCase(
                            String firstName, String lastName);

    List<Vet> findByFirstNameAndLastNameAllIgnoreCase(
                            String firstName, String lastName);

    List<Vet> findByGraduationYearGreaterThan(
                            Integer graduationYear);
```

Metot İsminden Sorgu Üretmek

```
    List<Vet> findByGraduationYearLessThan(Integer graduationYear);

    List<Vet> findByGraduationYearBetween(Integer graduationYearFrom,
Integer graduationYearTo);

    List<Vet> findFirstByOrderByLastNameAsc();

    List<Vet> findFirst5ByLastName(String lastName);

    List<Vet> findTopByOrderByLastNameAsc();

    List<Vet> findTop5ByLastName(String lastName);

    List<Vet> findFirst5ByLastName(String lastName, Pageable pageable);

    List<Vet> findFirst5ByLastName(String lastName, Sort sort);

}
```

Property İfadesinin Çözümlenmesi

```
List<Owner> findByAddressZipCode(ZipCode zipCode);
```

Property çözümleme işi domain sınıfından başlar.
1.İlk olarak By ifadesinden sonraki bütün bölüm (AddressZipCode) domain sınıfında aranır.
2.Böyle bir property Owner sınıfı içerisinde mevcut değil ise, sağ taraftan başlayarak property ismi “camel case” bölümlerden tokenize edilerek çözümlenmeye çalışılır: addressZip.code, address.zipCode

```
List<Owner> findByAddress_ZipCode(ZipCode zipCode);
```

“_” karakteri ile property ifadesinin hangi noktalardan tokenize edileceği belirlenebilir

Manuel Sorgu Tanımlamak

- **@Query** anotasyonu ile tanımlanır

```
@Query(
name = "findVetsByFirstNameAndSpecialty",
value = "select distinct v from Vet v left join fetch v.specialties s
where v.firstName = $1 and s.name = $2")
List<Vet> findVetsByFirstNameAndSpecialtyName(
                        String firstName, String specialtyName);
```

Asenkron Sorgular

- Repository sorguları Spring'in **Async** metot çalıştırma kabiliyeti ile asenkron çalıştırılabilir
- Bu durumda sorgu metodu hemen return eder, ancak sorgu arka planda **ayrı bir Task olarak** çalıştırılır
- Sorgu sonucuna da sonrasında **Future nesnesi** üzerinden ulaşılabilir

Asenkron Sorgular

```
@Async
Future<Vet> findVetsAsyncByLastName(String lastName);
```

Java concurrent API'nin Future tipidir

```
@Async
CompletableFuture<Vet> findVetsAsyncByFirstName(
                                        String firstName);
```

Java 8 ile birlikte gelen Future tipidir

Spring'e ait Future tipidir

```
@Async
ListenableFuture<Vet>
findVetsAsyncByGraduationYearBetween(Integer from, Integer to);
```

Repository Nesnelerinin Yaratılması

- Proxy repository bean'leri **XML tabanlı** veya **Java tabanlı** Spring konfigürasyonları ile yaratılabilirler
- Her Spring Data modülünün **kendine ait repositories elemanı** mevcuttur
- Belirli **paketlerin altı taranarak** Repository arayüzlerinden bean instance'ları yaratılabilir

Repository Nesnelerinin Yaratılması (XML)

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jpa="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-
1.8.xsd">

    <jpa:repositories
        base-package="com.javaegitimleri.petclinic.repository">
    </jpa:repositories>

</beans>
```

Repository Nesnelerinin Yaratılması (Java)

```
@Configuration
@EnableJpaRepositories(
basePackages={"com.javaegitimleri.petclinic.repository"})
public class ApplicationConfiguration {
    ...
}
```

Repository Nesnelerine Erişim

- Proxy repository nesneleri diğer normal Spring bean'leri gibi **enjekte** edilebilirler
- Ya da ApplicationContext'den **lookup** yapılabilirler

Repository Nesnelerine Erişim

```
@Service
@Transactional
public class PetClinicServiceImpl implements PetClinicService {

    private VetRepository vetRepository;

    @Autowired
    public void setVetRepository(VetRepository vetRepository) {
        this.vetRepository = vetRepository;
    }

    @Override
    public Collection<Vet> findVets() {
        return vetRepository.findAll();
    }
}
```

Custom Repository Metot Yazılması

- Zaman zaman Repository bean'lerine **custom metot implemantasyonları** eklenmesi gerekmektedir
- Spring Data buna imkan tanımaktadır
- Custom metodu içeren Repository arayüzünden **farklı bir arayüz** tanımlanır
- Arayüzü implement eden **bir sınıf oluşturulur**
- Bu sınıf içerisinde de **custom metot implement edilir**

Custom Repository Metot Yazılması

```
public interface VetRepositoryCustom {
    List<Vet> findVetsWithSpecialties();
}

public interface VetRepository extends
                    JpaRepository<Vet, Long>, VetRepositoryCustom {
    ...
}

public class VetRepositoryImpl implements VetRepositoryCustom {

    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Override
    public List<Vet> findVetsWithSpecialties() {
        return sessionFactory.getCurrentSession()
            .createQuery("select distinct v from Vet v left join fetch
v.specialties").list();
    }
}
```

Belirtilen suffixlerle bitmeleri önemlidir!

Custom Repository Metot Yazılması

- Eğer custom repository implemantasyonuna herhangi bir bağımlık enjekte edilmeyecek ise bean tanımlamaya gerek yoktur
- Spring custom arayüzü implement eden “**Impl” uzantılı sınıfı** tespit eder ve bundan bir instance yaratır
- Bu sınıfın **Repository arayüzü ile aynı paket altında** olması gerekir

Custom Repository Metot Yazılması

- Eğer bağımlılık enjekte edilecek ise bu durumda **Impl uzantılı isimde bir bean tanımlanmalı**dır

```
<beans...>
    <bean id="vetRepositoryImpl"
        class="com.javaegitimleri.petclinic.repository.VetRepositoryImpl">
            <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

</beans>
```

Custom Davranışın Bütün Repository'lere Eklenmesi

- Bütün Repository bean'lerinin sahip olacağı **ortak metotlar yazmak da mümkün**dür
- Bu durumda custom metotları içerecek arayüz **Spring Data'nın Repository arayüzünden extend etmeli**dir
- Projenin **Repository arayüzleri** ise bu durumda **custom arayüzden extend edecek**lerdir

Custom Davranışın Bütün Repository'lere Eklenmesi

```
@NoRepositoryBean
public interface BaseRepository<T, ID extends Serializable>
            extends JpaRepository<T, ID>{
    Session getSession();
}

public abstract class BaseRepositoryImpl<T, ID extends Serializable>
                extends SimpleJpaRepository<T, ID>
                    implements BaseRepository<T, ID> {

    private EntityManager entityManager;

    public BaseRepositoryImpl(Class<T> domainClass, EntityManager em) {
        super(domainClass, em);
        entityManager = em;
    }

    @Override
    public Session getSession() {
        return (Session) entityManager.getDelegate();
    }
}
```

Custom Davranışın Bütün Repository'lere Eklenmesi

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jpa="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-1.8.xsd">

    <jpa:repositories
            base-package="com.example.repository"
            base-class="com.example.repository.BaseRepositoryImpl">
    </jpa:repositories>

</beans>
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com