



SPRING BOOT MAPPING



Mappatura one to one3

Mappatura one to many5

MAPPATURA ONE TO ONE

La mappatura "one to one" in informatica è una relazione tra due entità in cui ogni istanza dell'entità A è associata a una sola istanza dell'entità B, e viceversa. Questo tipo di relazione può essere utile in molte applicazioni, ad esempio quando si ha bisogno di associare un singolo cliente a una singola fattura.

Vedremo come implementare una mappatura "one to one" in Spring Boot, utilizzando un esempio di relazione tra un'entità "Customer" e un'entità "Invoice".

1. Definire le entità Per prima cosa, definiamo le due entità "Customer" e "Invoice". In questo esempio, un cliente può avere una sola fattura, quindi abbiamo una relazione "one to one".
2. Definire le due repository
3. Definire i relativi service
4. Esporre i service con Api

```
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToOne(mappedBy = "customer", cascade = CascadeType.ALL)
    private Invoice invoice;
    // getter e setter
}

@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String number;
    @OneToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;
    // getter e setter
}
```

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {}
```

```
public interface InvoiceRepository extends JpaRepository<Invoice, Long> {}
```

Differenza tra CrudRepository e JpaRepository

CrudRepository e JpaRepository sono entrambe interfacce fornite da Spring Data JPA per semplificare l'interazione con il database e l'implementazione delle operazioni CRUD (Create, Read, Update, Delete) su di esso. Tuttavia, esistono alcune differenze tra queste due interfacce.

CrudRepository è l'interfaccia di base fornita da Spring Data JPA e fornisce metodi per le operazioni CRUD di base, come ad esempio save, delete, findById e findAll. JpaRepository estende CrudRepository e fornisce metodi aggiuntivi per le operazioni di interrogazione e ricerca come ad esempio findByXXX e deleteByXXX. JpaRepository supporta anche funzionalità di paging e ordinamento dei risultati.

Inoltre, JpaRepository è specifico per JPA (Java Persistence API) e quindi offre metodi specifici per l'interazione con l'EntityManager di JPA, come ad esempio flush e detach. JpaRepository supporta anche la definizione di query personalizzate tramite l'utilizzo di annotazioni come @Query.

In generale, se il tuo caso d'uso richiede solo le operazioni CRUD di base, puoi utilizzare CrudRepository. Se invece hai bisogno di operazioni di interrogazione più avanzate o di funzionalità di paging e ordinamento, o se stai utilizzando JPA come provider di persistenza, allora è consigliabile utilizzare JpaRepository.

MAPPATURA ONE TO MANY

La mappatura One-to-Many è un tipo di relazione tra due tabelle di un database dove una riga della tabella principale (o "padre") può essere associata a molte righe della tabella secondaria (o "figlio"). In questo tipo di relazione, la tabella figlio contiene una colonna che fa riferimento alla chiave primaria della tabella padre. In questo modo, ogni riga nella tabella figlio è associata a una sola riga nella tabella padre.

Mostreremo come implementare la mappatura One-to-Many in un'applicazione Spring Boot, utilizzando l'ORM Hibernate per interagire con il database. Utilizzeremo anche il linguaggio di programmazione Java per il codice dell'applicazione.

Per cominciare, supponiamo di avere due tabelle nel nostro database: "studente" e "corsi". Un singolo studente può iscriversi a molti corsi, mentre ogni corso è associato a un solo studente.

1. Definizione delle entità Per definire le nostre entità, creiamo due classi Java chiamate "Studente" e "Corso". La classe Studente avrà una lista di corsi a cui lo studente è iscritto, mentre la classe Corso avrà una variabile di istanza per rappresentare lo studente che ha creato il corso.

Ecco un esempio di come potrebbe apparire la classe Studente:

```
@Entity
@Table(name = "studente")
public class Studente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;

    @OneToMany(mappedBy = "studente")
    private List<Corso> corsi = new ArrayList<>();

    // costruttori, getter e setter
}
```

```
@Entity
public class Corso {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @ManyToOne
    @JoinColumn(name = "studente_id")
    private Studente studente;

    // costruttori, getter e setter
}
```

Nell'esempio sopra, la classe Studente ha un attributo "corsi" che è una lista di oggetti Corso. Questa relazione è mappata con l'annotazione @OneToMany sulla classe Studente.

La classe Corso ha un attributo "studente" che rappresenta lo studente che ha iscritto al corso. Questa relazione è mappata con l'annotazione @ManyToOne sulla classe Corso, che indica che molti oggetti Corso possono essere associati a un unico oggetto Studente.

Inoltre, l'annotazione @JoinColumn viene utilizzata per specificare il nome della colonna nella tabella dei corsi che contiene l'id dello studente.

Infine, l'attributo "cascade" sull'annotazione @OneToMany viene utilizzato per indicare che tutte le operazioni di persistenza sui corsi devono essere propagate allo studente associato.

Con questa mappatura, un oggetto Corso può essere associato a molti oggetti Studente, ma ogni oggetto Studente può essere associato solo a un oggetto Corso.