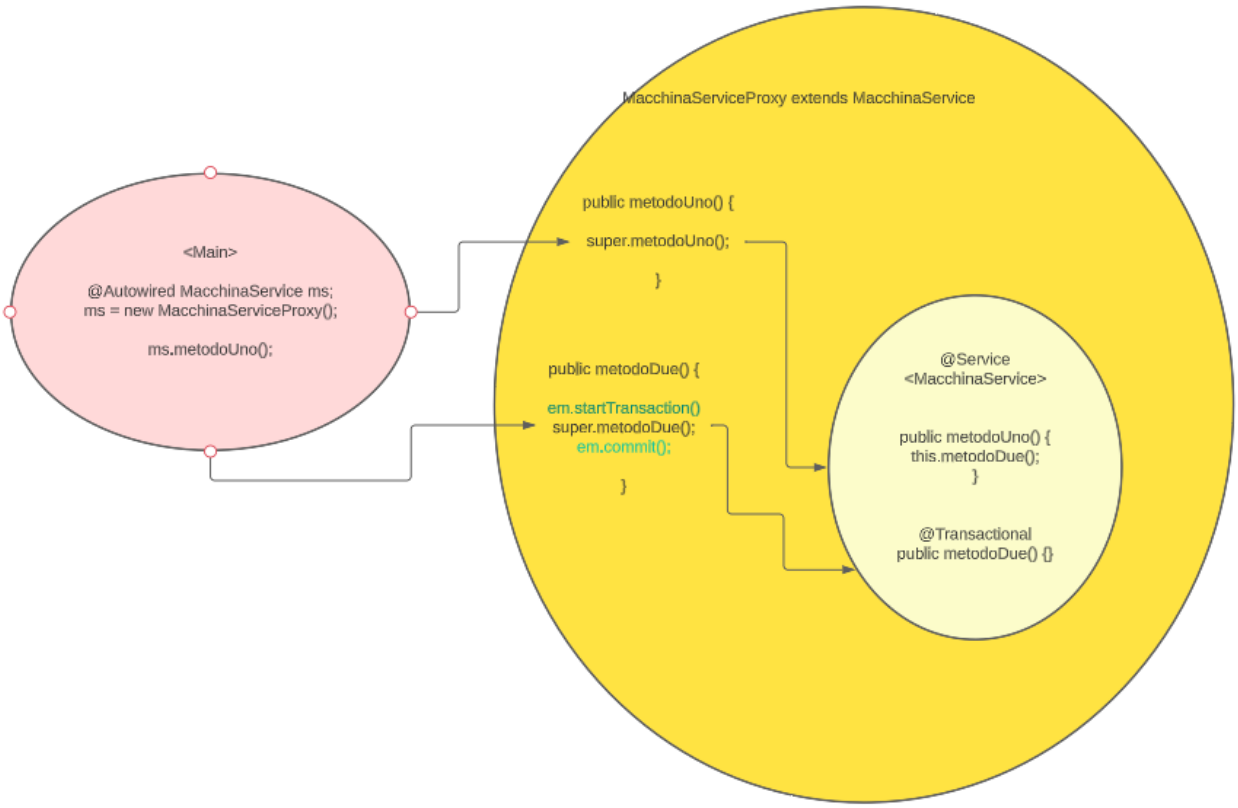
Guida Associazioni tra Entity JPA

1. Proxy dei bean



Chiamare i metodi transazionali dei bean da fuori, altrimenti se dall'interno del bean in un metodo non transazionale, viene chiamato un metodo annotato con @Transactional in realtà non si sta chiamando il metodo avvolto dal proxy.

quindi: dal main che contiene il proxy del bean si deve chiamare metodoUno poi metodoDue. in questo modo si passa ogni volta dal proxy che avvolge il bean ed i metodi.

Chapter 1. OneToMany

1.1. Introduzione

https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

In un sistema di database relazionale, un'associazione uno-a-molti collega due tabelle in base a una colonna di chiave esterna in modo che il record della tabella figlia faccia riferimento alla chiave primaria della riga della tabella padre.

Per quanto semplice possa essere in un database relazionale, quando si tratta di JPA, l' associazione del database uno-a-molti può essere rappresentata tramite un'associazione @ManyToOneo @OneToManyun'associazione poiché l'associazione OOP può essere unidirezionale o bidirezionale.

L' @ManyToOne annotazione ti consente di mappare la colonna Foreign Key nella mappatura dell'entità figlio in modo che il figlio abbia un riferimento all'oggetto entità alla sua entità genitore. Questo è il modo più naturale per mappare un'associazione di database uno-a-molti e, di solito, anche l'alternativa più efficiente .

> Per comodità, per sfruttare le transizioni dello stato dell'entità e il meccanismo di controllo sporco , molti sviluppatori scelgono di mappare le entità figlie come raccolta nell'oggetto genitore e, a questo scopo, JPA offre l' @OneToMany annotazione.

Come ho spiegato nel mio libro , molte volte è meglio sostituire le raccolte con una query, che è molto più flessibile in termini di prestazioni di recupero. Tuttavia, ci sono momenti in cui mappare una raccolta è la cosa giusta da fare e quindi hai due scelte:

@OneToMany un'associazione unidirezionale @OneToMany un'associazione bidirezionale L'associazione bidirezionale richiede che la mappatura dell'entità figlio fornisca un'annotazione @ManyToOne, che è responsabile del controllo dell'associazione .

D'altro canto, l' @OneToMany associazione unidirezionale è più semplice poiché è solo il lato genitore a definire la relazione. In questo articolo spiegherò gli avvertimenti delle @OneToMany associazioni e come superarli.

Esistono molti modi per mappare l' @OneToMany associazione. Possiamo usare una Lista o un Set. Possiamo anche definire @JoinColumnnanche l'annotazione. Quindi, vediamo come funziona tutto questo.

1.2. OneToMany

Associazione a titolo di esempio

Testo della nota

Mappatura corretta

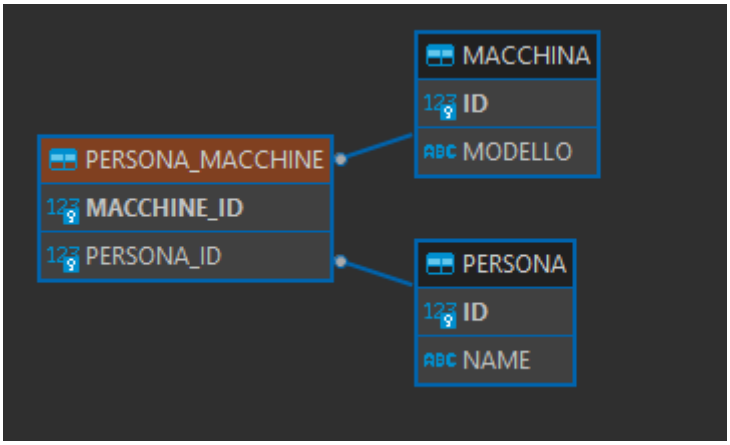
Per una corretta associazione vedere il package "bidirezionale_one_to_many"

1.2.1. Unidirezionale

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class Persona {

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

  private String name;

  @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
   private Set<Macchina> macchine = new HashSet<>();

  /**
   * Evitare di usare le liste, perchè questo genere una quantita di insert nel
momento della rimozione di un elemento dalla lista.
   * Questo perchè essendo le liste ordinate, quando rimuovi un elemento lui si deve
riordinare la lista come in precedenza
   * hibernate vede questo come una modifica e quindi elimina tutti gli elementi e li
reinserisce.
   */
  //  private List<Macchina> macchine = new ArrayList<>();
}
```

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class Macchina {

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

  private String modello;
}
```

- Questo mappatura genera una tabella di mezzo PERSONA_MACCHINE con Persona_id e

Macchina_id 

> Per un DBA, questa assomiglia più a un'associazione di database molti-a-molti che a una relazione uno-a-molti e non è neanche molto efficiente. Invece di due tabelle, ora ne abbiamo tre, quindi utilizziamo più spazio di archiviazione del necessario. Invece di una sola chiave esterna, ora ne abbiamo due. Tuttavia, poiché molto probabilmente indicizzeremo queste chiavi esterne, avremo bisogno del doppio della memoria per memorizzare nella cache l'indice per questa associazione. Non bello!

- Questo metodo viene utilizzato per creare una persona e per associargli una macchinaBi

```
public void inizializzaPersonaConUnaMacchinaAssociata() {

    log.info(() -> "Inizializzo il db con la creazione di una persona con una
macchinaBi associata");

   Persona stefano = new Persona();
    stefano.setName("Stefano");

    Macchina ford = new Macchina();
    ford.setModello("ford");

    Macchina bmw = new Macchina();
    bmw.setModello("bmw");

    stefano.getMacchine().add(ford);
    stefano.getMacchine().add(bmw);

    personaRepository.save(stefano);

    log.info(()-> "Fine inizializzazione");
  }
```

- Con questa mappatura vengono effettuate queste query di inserimento:

```
insert into persona (name,id) values (?,default)
insert into macchina (modello,id) values (?,default)
```

```
insert into macchina (modello,id) values (?,default)

insert into persona_macchine (persona_id,macchine_id) values (?,?)
insert into persona_macchine (persona_id,macchine_id) values (?,?)
```

- Con questa mappatura vengono effettuate queste query per rimuovere una macchina da un proprietario:

```
select m1_0.persona_id,m1_1.id,m1_1.modello from persona_macchine m1_0 join macchina
m1_1 on m1_1.id=m1_0.macchine_id where m1_0.persona_id=?
delete from persona_macchine where persona_id=?
insert into persona_macchine (persona_id,macchine_id) values (?,?)
```

Mappatura con Persone che contiene una Lista di macchine:

1. Viene fatta una delete sulla tabella PERSONA_MACCHINE per tutti gli elementi che contengono l'id della persona a cui vogliamo rimuovere una singola macchina. questo significa che togli in prima battuta tutte le associazini tra quell'utente e le auto.
2. Poi fa la insert per riassociare le auto rimaste a quella persona.

Questo avviene perche Persona ha al suo interno una lista di Macchina, la lista deve mantenere l'ordine e per questo vengono rimosse e reinserite nello stesso ordine.

QUINDI --→ utilizzare i SET

Set<Macchina>

Utilizzando i Set ecco le query che vengono generate:

```
select p1_0.id,p1_0.name from persona p1_0
select m1_0.persona_id,m1_1.id,m1_1.modello from persona_macchine m1_0 join macchina
m1_1 on m1_1.id=m1_0.macchine_id where m1_0.persona_id=?
delete from persona_macchine where persona_id=? and macchine_id=?
```

Viene effettuta la delete puntuale dell'elemento.

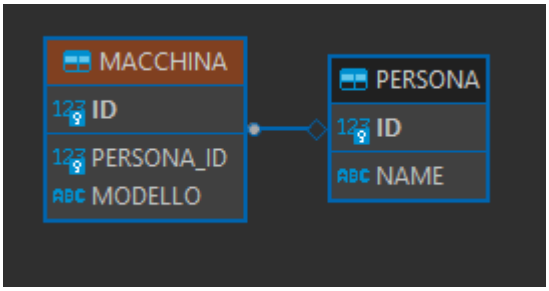
1.2.2. Con JoinColumn

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class Persona {

  @Id
```

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

  private String name;

  @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)

  // Aggiunta questa annotazione
  @JoinColumn(name = "persona_id")
  private List<Macchina> macchine = new ArrayList<>();
}
```

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class Macchina {

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

  private String modello;
}
```

- L' @JoinColumn annotazione aiuta Hibernate a capire che PERSONA_ID nella tabella MACCHINE è presente una colonna Foreign Key che definisce questa associazione.



- Con questa mappatura vengono effettuate queste query: questa volta inseriamo due macchine.

```
insert into persona (name,id) values (?,default)
insert into macchinaBi (modello,id) values (?,default)
insert into macchinaBi (modello,id) values (?,default)
update macchinaBi set persona_id=? where id=?
update macchinaBi set persona_id=? where id=?
```

Come possiamo vedere vengono effettuate 3 insert, 1 per la persona e 2 per le macchine, poi ogni macchinaBi viene fatto l'update per assegnargli la persona.

1.3. OneToMany BIDIREZIONALE

Il modo migliore per mappare un'associazione @OneToMany è fare affidamento sul @ManyToOne lato per propagare tutte le modifiche allo stato dell'entità:

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class PersonaBi {

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

  private String name;

  @OneToMany(mappedBy = "proprietario", cascade = CascadeType.ALL, orphanRemoval =
true)
  private List<MacchinaBi> macchine = new ArrayList<>();

  /**
   * presenta due metodi di utilità (ad esempio aggiungiMacchina e rimuoviMacchina)
che vengono utilizzati
   * per sincronizzare entrambi i lati dell'associazione bidirezionale. Dovresti
sempre fornire
   * questi metodi ogni volta che lavori con un'associazione bidirezionale poiché,
altrimenti,
   * rischi di problemi di propagazione dello stato molto sottili
   */
  public void aggiungiMacchina(MacchinaBi macchinaBi) {
    macchine.add(macchinaBi);
    macchinaBi.setProprietario(this);
  }

  public void rimuoviMacchina(MacchinaBi macchinaBi) {
    macchine.remove(macchinaBi);
    macchinaBi.setProprietario(null);
  }
}
```

Classe Persona

presenta due metodi di utilità (ad esempio aggiungiMacchina e rimuoviMacchina) che vengono utilizzati per sincronizzare entrambi i lati dell'associazione bidirezionale. Dovresti sempre fornire questi metodi ogni volta che lavori con un'associazione bidirezionale poiché, altrimenti, rischi di problemi di propagazione dello stato molto sottili

```
@Entity
@NoArgsConstructor
@Getter
@Setter
public class MacchinaBi {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

  private String modello;

  /**
   * L' @ManyToOneassociazione utilizza FetchType.LAZYperché, altrimenti, ricadremmo
sul recupero
   * EAGER, il che è negativo per le prestazioni .
   */
  @ManyToOne(fetch = FetchType.LAZY)
  private PersonaBi proprietario;

  /**
   * L'entità figlio, Macchina, implementa i metodi equalse hashCode. Poiché non
possiamo fare
   * affidamento su un identificatore naturale per i controlli di uguaglianza ,
dobbiamo utilizzare
   * invece l'identificatore di entità per il equalsmetodo. Tuttavia, è necessario
farlo
   * correttamente in modo che l'uguaglianza sia coerente in tutte le transizioni
dello stato
   * dell'entità , che è anche il motivo per cui hashCodedeve essere un valore
costante. Poiché
   * facciamo affidamento sull'uguaglianza per removeComment, è buona norma
sovrascrivere equalse
   * hashCodeper l'entità figlio in un'associazione bidirezionale.
   */
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    MacchinaBi macchina = (MacchinaBi) o;
    return Objects.equals(id, macchina.id);
  }

  @Override
  public int hashCode() {
    return Objects.hash(id);
  }
}
```

° inizializzazione dei dati con una persona che possiede due macchine

```
insert into persona_bi (name,id) values (?,default)
insert into macchina_bi (modello,proprietario_id,id) values (?,?,default)
insert into macchina_bi (modello,proprietario_id,id) values (?,?,default)
```

° Rimozione della macchina da lato Persona:

```
select p1_0.id,p1_0.name from persona_bi p1_0
select m1_0.proprietario_id,m1_0.id,m1_0.modello from macchina_bi m1_0 where m1_0
.proprietario_id=?
update macchina_bi set modello=?,proprietario_id=? where id=?
```

° Rimozione Del propietaro dalla macchina:

```
select m1_0.id,m1_0.modello,m1_0.proprietario_id from macchina_bi m1_0 where m1_0
.proprietario_id is not null
update macchina_bi set modello=?,proprietario_id=? where id=?
```

° Rimozione Del propietaro dalla macchina, utilizzando anche il metodo del proprietario di rimuovere la macchina:

```
public void rimuoviProprietario(){
    // this = è la macchina.
    //TODO verificare se è necessario rimuovere dal proprietario la macchina???
    this.proprietario.rimuoviMacchina(this);
    this.setProprietario(null);
  }
```

a. Devo ancora capire se sia necessario per tenere sincronizzati i dati lato java (forse dipende dal caso d'uso).

```
select m1_0.id,m1_0.modello,m1_0.proprietario_id from macchina_bi m1_0 where m1_0
.proprietario_id is not null
select p1_0.id,p1_0.name from persona_bi p1_0 where p1_0.id=?
select m1_0.proprietario_id,m1_0.id,m1_0.modello from macchina_bi m1_0 where m1_0
.proprietario_id=?
update macchina_bi set modello=?,proprietario_id=? where id=?
```

Chapter 2. ManyToMany

Da fare