


Home

PUBLIC


 Stack Overflow

Tags

Users

Jobs

TEAMS

 Create Team

Owning Side vs Non-Owning Side In hibernate and its usage in reference of mapped by element?

[Ask Question](#)

I came across the terms i.e owning side and non-owning side while studying hibernate.For example :- here is the statement regarding usage of mappedby element in terms of its usage in one to one mapping

If the relationship is bidirectional, the non-owning side must use the mappedBy element of the OneToOne annotation to specify the relationship field or property of the owning side.

But i did not get what actually owning side and non-owning side is?

[java](#) [hibernate](#)

asked Jan 5 '14 at 13:31



[M Sach](#)

15.2k 57 170 263

You mean what does it do or the reason behind the name mappedBy? I put bellow an explanation for why it's necessary and why it's called like that, let me know if you have further questions. – [Angular University](#) Jan 5 '14 at 18:06

3 Answers

ADDRESS_ID **int** NULL FOREIGN KEY REFERENCES **Address** (ID)

answered Jan 5 '14 at 13:36



[SJuan76](#)

21.7k 6 31 72

I was also under same impression. Please see my answer also – [M Sach](#) Jan 5 '14 at 13:53

The idea of a owning side of a bidirectional relation comes from the fact that in relational databases there are no bidirectional relations like in the case of objects.

In databases we only have foreign keys, where only one table can have a foreign key to another. Let's take an example that would not work as expected and see why mappedBy is necessary:

```
@Entity
@Table(name="PERSONS")
public class Person {
    @OneToMany
    private List<IdDocument> idDocuments;
}

@Entity
@Table(name="IDDOCUMENT")
public class IdDocument {
    @ManyToOne
    private Person person;
}
```

This would create not only tables PERSONS and IDDOCUMENTS. but would also create a third table

```

    iddocuments_id bigint NOT NULL,
    CONSTRAINT fk_persons FOREIGN KEY (persons_id) REFERENCES persons (id),
    CONSTRAINT fk_docs FOREIGN KEY (iddocuments_id) REFERENCES iddocument (id),
    CONSTRAINT pk UNIQUE (iddocuments_id)
)

```

Notice the primary key on documents only. In this case Hibernate tracks both sides of the relation independently: If you add a document to relation `Person.idDocuments`, it inserts a record in `PERSON_IDDOCUMENTS`.

If we change the `Person` of a `IdDocument`, it changes the foreign key `person_id` on table `IDDOCUMENTS`.

Hibernate is creating two unidirectional (foreign key) relations on the database, to implement one bidirectional object relation, because databases do not support bidirectional relations.

But what we want is for the object relation to be only mapped by the foreign key on table `IDDOCUMENTS` towards `PERSON`: one document belongs to only one person.

There is no need for the extra table, that would force us to modify both `Person.idDocuments` and `IdDocument.person` inside the same database transaction to keep the relation consistent.

To solve this we need to configure Hibernate to stop tracking the modifications on relation `Person.idDocuments`. Hibernate should only track the other side of the relation `IdDocument.person`, and to do so we add `mappedBy`:

```

@OneToMany(mappedBy="person")
private List<IdDocument> idDocuments;

```

This means "modifications on this side of the relation are already **Mapped By** by the other side of the relation `IdDocument.person`, so no need to track it here separately in an extra table".

This gives us the mapping we want, but has one major consequence:

- Modifications on the `Person.idDocuments` collection are no longer tracked by Hibernate, and it is the responsibility of the developer to modify `IdDocument.person` instead in order to modify the association.

[edited Jan 5 '14 at 15:49](#)

[answered Jan 5 '14 at 14:56](#)

Here is my understanding with simple example where College has many students(One to many relationship)

College(Value Object) -----> **College (Database Table)** corresponds to

Student(Value Object) -----> **Student (Database Table)** having column
which is foreign key to **College** table .
So This is owning side and
College is **Non-owning side**)

In terms of Object representation, Student object is owning side becoz it will be having reference pointing to college column. So Student is owning side and College in non-owning side.

Usage of Owner side and Non-Ownning side in hibernate in relation of mapped by element

Non-Ownning Side Object

```
@Entity
public class College {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int collegeId;
    @OneToMany(mappedBy="college") // here non-owning side using mapped by element
    to specify // the relationship field of owning side
    private List<Student> students;
}
```

Owning Side Object

```
@Entity
```

In conclusion the owning side is the entity that has the reference to the other. In terms of DB, it translates to table entity having column which is foreign key to column in other table like in case of College has many students.

Note:- owning-side table must contain a join column that refers to the other table's id. It means owning side entity should contain @JoinColumn annotation otherwise it will consider the name as primary key column name of other table. See [@OneToOne unidirectional and bidirectional](#)

edited May 23 '17 at 12:33



Community ♦

1 1

answered Jan 5 '14 at 13:40



M Sach

15.2k 57 170 263