(/)

Descripción general de los tipos de cascada JPA / Hibernate

Última modificación: 29 de junio de 2019

> por baeldung (https://www.baeldung.com/author/baeldung/) (https://www.baeldung.com/author/baeldung/)

Persistencia (https://www.baeldung.com/category/persistence/)

JPA (https://www.baeldung.com/tag/jpa/)

1. Introducción

En este tutorial, analizaremos qué cascada está en JPA / Hibernate. Luego, cubriremos los diversos tipos de cascada que están disponibles, junto con su semántica.

2. ¿Qué es la cascada?

Las relaciones entre entidades a menudo dependen de la existencia de otra entidad, por ejemplo, la relación *Persona - Dirección* . Sin la *Persona* , la entidad de *Dirección* no tiene ningún significado propio. Cuando eliminamos la entidad de *persona* , nuestra entidad de *dirección* también debe eliminarse.

La cascada es la manera de lograr esto. **Cuando realizamos alguna acción en la entidad objetivo, la misma acción se aplicará a la entidad asociada.**

2.1. Tipo de cascada JPA

Todas las operaciones en cascada específicas de JPA están representadas por la enumeración *javax.persistence.CascadeType que* contiene entradas:

- *TODOS*
- *PERSISTIR*
- *UNIR*
- *RETIRAR*
- *REFRESCAR*
- *DESPEGAR*

2.2. Tipo de cascada de hibernación

Hibernate admite tres tipos de cascada adicionales junto con los especificados por JPA. Estos tipos de cascada específicos de Hibernate están disponibles en *org.hibernate.annotations.CascadeType* :

- *REPRODUCIR EXACTAMENTE*
- *SAVE_UPDATE*
- *BLOQUEAR*

3. Diferencia entre los tipos de cascada

3.1. *CascadeType* . *TODOS*

Cascade.ALL **propaga todas las operaciones, incluidas las específicas de Hibernate, de una entidad principal a una secundaria.**

Veamos en un ejemplo:

```
@Entity
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;
    @OneToMany(mappedBy = "person", cascade = CascadeType.ALL)
    private List<Address> addresses;
}
```

Tenga en cuenta que en *las* asociaciones *OneToMany* , hemos mencionado el tipo de cascada en la anotación.

Ahora, veamos la *dirección de la* entidad asociada :

```
@Entity
public class Address {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String street;
    private int houseNumber;
    private String city;
    private int zipCode;
    @ManyToOne(fetch = FetchType.LAZY)
    private Person person;
}
```

3.2. *CascadeType* . *PERSISTIR*

La operación de persistencia hace persistente una instancia transitoria. **CascadeType *PERSISTE* propaga la operación persisten de un padre a una entidad secundaria** . Cuando guardamos la entidad de *persona* , la entidad de *dirección* también se guardará.

Veamos el caso de prueba para una operación persistente:

```
@Test
public void whenParentSavedThenChildSaved() {
    Person person = new Person();
    Address address = new Address();
    address.setPerson(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();
    session.clear();
}
```

Cuando ejecutemos el caso de prueba anterior, veremos el siguiente SQL:

```
Hibernate: insert into Person (name, id) values (?, ?)
Hibernate: insert into Address (city, houseNumber, person_id, street,
```

3.3. *CascadeType* . *UNIR*

La operación de fusión copia el estado del objeto dado en el objeto persistente con el mismo identificador. ***CascadeType.MERGE* propaga la operación de combinación de una entidad principal a una secundaria** .

Probemos la operación de fusión:

```
@Test
public void whenParentSavedThenMerged() {
    int addressId;
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();
    addressId = address.getId();
    session.clear();

    Address savedAddressEntity = session.find(Address.class, addressI
    Person savedPersonEntity = savedAddressEntity.getPerson();
    savedPersonEntity.setName("devender kumar");
    savedAddressEntity.setHouseNumber(24);
    session.merge(savedPersonEntity);
    session.flush();
}
```

Cuando ejecutamos el caso de prueba anterior, la operación de combinación genera el siguiente SQL:

```
Hibernate: select address0_.id as id1_0_0_, address0_.city as city2_0_
Hibernate: select person0_.id as id1_1_0_, person0_.name as name2_1_0_
Hibernate: update Address set city=?, houseNumber=?, person_id=?, stre
Hibernate: update Person set name=? where id=?
```

Aquí, podemos ver que la operación de fusión primero carga las entidades de *dirección* y *persona* y luego las actualiza como resultado de *CascadeType MERGE* .

3.4. *CascadeType.REMOVE*

Como sugiere su nombre, la operación de eliminación elimina la fila correspondiente a la entidad de la base de datos y también del contexto persistente.

***CascadeType.REMOVE* propaga la operación de eliminación de la entidad principal a la secundaria. Similar al *CascadeType.REMOVE* de JPA , tenemos *CascadeType.DELETE* , que es específico de Hibernate** . No hay diferencia entre los dos.

Ahora es el momento de probar *CascadeType* . *Eliminar* :

```
@Test
public void whenParentRemovedThenChildRemoved() {
    int personId;
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();
    personId = person.getId();
    session.clear();

    Person savedPersonEntity = session.find(Person.class, personId);
    session.remove(savedPersonEntity);
    session.flush();
}
```

Cuando ejecutemos el caso de prueba anterior, veremos el siguiente SQL:

```
Hibernate: delete from Address where id=?
Hibernate: delete from Person where id=?
```

La *dirección* asociada con la *persona* también se eliminó como resultado de *CascadeType REMOVE* .

3.5. *CascadeType.DETACH*

La operación de separación elimina la entidad del contexto persistente. **Cuando usamos *CascaseType.DETACH*, la entidad secundaria también se eliminará del contexto persistente** .

Vamos a verlo en acción:

```
@Test
public void whenParentDetachedThenChildDetached() {
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();

    assertThat(session.contains(person)).isTrue();
    assertThat(session.contains(address)).isTrue();

    session.detach(person);
    assertThat(session.contains(person)).isFalse();
    assertThat(session.contains(address)).isFalse();
}
```

Aquí, podemos ver que después de separar *persona* , ni *persona* ni *dirección* existen en el contexto persistente.

3.6. *CascadeType* . *BLOQUEAR*

Inintuitivamente, *CascadeType.LOCK* vuelve a *adjuntar* la entidad y su entidad secundaria asociada con el contexto persistente nuevamente.

Veamos el caso de prueba para entender *CascadeType.LOCK* :

```
@Test
public void whenDetachedAndLockedThenBothReattached() {
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();

    assertThat(session.contains(person)).isTrue();
    assertThat(session.contains(address)).isTrue();

    session.detach(person);
    assertThat(session.contains(person)).isFalse();
    assertThat(session.contains(address)).isFalse();
    session.unwrap(Session.class)
      .buildLockRequest(new LockOptions(LockMode.NONE))
      .lock(person);

    assertThat(session.contains(person)).isTrue();
    assertThat(session.contains(address)).isTrue();
}
```

Como podemos ver, cuando usamos *CascadeType.LOCK* , *adjuntamos* la *persona de* la entidad y su *dirección* asociada al contexto persistente.

3.7. *CascadeType* . *REFRESCAR*

Las operaciones de actualización **releen el valor de una instancia dada de la base de datos** . En algunos casos, podemos cambiar una instancia después de persistir en la base de datos, pero luego necesitamos deshacer esos cambios.

En ese tipo de escenario, esto puede ser útil. **Cuando usamos esta operación con CascadeType *REFRESH* , la entidad secundaria también se vuelve a cargar desde la base de datos cada vez que se actualiza la entidad principal.**

Para una mejor comprensión, veamos un caso de prueba para *CascadeType.REFRESH* :

```
@Test
public void whenParentRefreshedThenChildRefreshed() {
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.persist(person);
    session.flush();
    person.setName("Devender Kumar");
    address.setHouseNumber(24);
    session.refresh(person);

    assertThat(person.getName()).isEqualTo("devender");
    assertThat(address.getHouseNumber()).isEqualTo(23);
}
```

Aquí, hicimos algunos cambios en la *persona* y la *dirección de las* entidades guardadas . Cuando actualizamos la entidad de *persona* , la *dirección* también se actualiza.

3.8. *CascadeType.REPLICATE*

La operación de replicación se usa cuando tenemos más de una fuente de datos, y queremos que los datos estén sincronizados. Con *CascadeType.REPLICATE* , una operación de sincronización también se

propaga a entidades secundarias siempre que se realice en la entidad principal.

Ahora, vamos a probar *CascadeType. REPLICAR* :

```
@Test
public void whenParentReplicatedThenChildReplicated() {
    Person person = buildPerson("devender");
    person.setId(2);
    Address address = buildAddress(person);
    address.setId(2);
    person.setAddresses(Arrays.asList(address));
    session.unwrap(Session.class).replicate(person, ReplicationMode.(
    session.flush();

    assertThat(person.getId()).isEqualTo(2);
    assertThat(address.getId()).isEqualTo(2);
}
```

Debido a *CascadeType REPLICATE* , cuando replicar la *persona* entidad, entonces su asociado *de direcciones* también se replica con el identificador nos propusimos.

3.9. *CascadeType.SAVE_UPDATE*

CascadeType.SAVE_UPDATE propaga la misma operación a la entidad secundaria asociada. Es útil cuando usamos **operaciones Hibernate-específicos como *guardar, actualizar* y *saveOrUpdate*** .

Veamos *CascadeType. SAVE_UPDATE* en acción:

```
@Test
public void whenParentSavedThenChildSaved() {
    Person person = buildPerson("devender");
    Address address = buildAddress(person);
    person.setAddresses(Arrays.asList(address));
    session.saveOrUpdate(person);
    session.flush();
}
```

Debido a *CascadeType.SAVE_UPDATE* , cuando ejecutamos el caso de prueba anterior, podemos ver que tanto la *persona* como la *dirección* se guardaron. Aquí está el SQL resultante:

```
Hibernate: insert into Person (name, id) values (?, ?)
Hibernate: insert into Address (city, houseNumber, person_id, street,
```

4. Conclusión

En este artículo, discutimos la cascada y las diferentes opciones de tipo de cascada disponibles en JPA e Hibernate.

El código fuente del artículo está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/persistence-modules/jpa-hibernate-cascade-type) .

Una introducción de datos de PRIMAVERA, JPA y Detalles de Semántica de Transacción con JPA

Consigue la persistencia con la primavera

Deja una respuesta

LAS CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
CLIENTE HTTP (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "BACK TO BASICS" (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
TUTORIAL HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON TUTORIAL DE PRIMAVERA (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
PUBLICIDAD EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)