equals hashCode и toString написание для разных случаев

1. Лучший способ реализовать equals, hashCode и toString с JPA и Hibernate

https://vladmihalcea.com/the-best-way-to-implement-equals-hashcode-and-tostring-with-jpa-and-hibernate/

Последнее изменение: 19 ноября 2020 г.

Представьте, что у вас есть инструмент, который может автоматически обнаруживать проблемы с производительностью JPA и Hibernate. [Оптимизатор гиперсистентности](https://vladmihalcea.com/hypersistence-optimizer/) - вот такой инструмент!

Улучшение байт-кода и toString

На прошлой неделе Марк Струберг, член Apache Software Foundation и участник OpenJPA, сделал следующее заявление:

По сути, он говорит, что реализация toString- это плохо с точки зрения производительности. Что ж, это может быть так в OpenJPA, но в Hibernate все немного иначе. По умолчанию Hibernate не использует [расширение байт-кода](https://vladmihalcea.com/how-to-enable-bytecode-enhancement-dirty-checking-in-hibernate/) .

Следовательно, метод toString может использовать любые базовые атрибуты сущности (которые необходимы для идентификации определенной сущности в журналах) до тех пор, пока базовые атрибуты выбираются при загрузке сущности из базы данных.

Тем не менее, Hibernate позволяет [лениво загружать атрибуты](https://vladmihalcea.com/the-best-way-to-lazy-load-entity-attributes-using-jpa-and-hibernate/) , но даже в этом случае улучшение байт-кода не обязательно является лучшим подходом. Использование подэлементов может быть лучшей альтернативой, и это даже не требует улучшения байт-кода.

Equals и hashCode

К сожалению, Марк продолжает это обсуждение с этим очень вводящие в заблуждение заявления о equalsи hashCode:

Это утверждение неверно, что подробно продемонстрирует этот пост.

Договор о равенстве

Согласно [спецификации Java](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-) , хорошая equalsреализация должна обладать следующими свойствами:

1. рефлексивный
2. симметричный
3. переходный
4. последовательный

Первые три довольно интуитивно понятны, но обеспечение *согласованности* в контексте сущностей JPA и Hibernate обычно является самой большой проблемой для разработчиков.

Как [уже было объяснено](https://vladmihalcea.com/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/) , equalsи hashCodeдолжны вести себя последовательно через все [лицо переходов между состояниями](https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/) .

Типы идентификаторов

С точки зрения равного контракта идентификаторы можно разделить на две категории:

* Присвоенные идентификаторы
* Идентификаторы, генерируемые базой данных

Присвоенные идентификаторы

Назначенные идентификаторы выделяются до очистки контекста постоянства, и мы можем разделить их на две подкатегории:

* Естественные идентификаторы
* UUID, не зависящие от базы данных

Естественные идентификаторы назначаются сторонним органом, например ISBN книги.

Номера UUID, не зависящие от базы данных, генерируются вне базы данных, как при вызове [java.util.UUID#randomUUID](https://docs.oracle.com/javase/7/docs/api/java/util/UUID.html" \l "randomUUID())метода.

И естественные идентификаторы, и UUID, не зависящие от базы данных, могут быть известны, когда сущность сохраняется. По этой причине, это безопасно использовать их в equalsи hashCodeреализации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24  25  26 год  27  28 год  29 | @Entity(name = "Book")  @Table(name = "book")  public class Book      implements Identifiable<Long> {        @Id      @GeneratedValue      private Long id;        private String title;        @NaturalId      private String isbn;        @Override      public boolean equals(Object o) {          if (this == o) return true;          if (!(o instanceof Book)) return false;          Book book = (Book) o;          return Objects.equals(getIsbn(), book.getIsbn());      }        @Override      public int hashCode() {          return Objects.hash(getIsbn());      }        //Getters and setters omitted for brevity  } |

Подробнее об @NaturalIdаннотации читайте в [этой статье](https://vladmihalcea.com/the-best-way-to-map-a-naturalid-business-key-with-jpa-and-hibernate/) .

Идентификаторы, генерируемые базой данных

Иная история с идентификаторами, сгенерированными базой данных. Поскольку идентификатор назначается базой данных во время очистки, гарантия согласованности нарушается, если мы реализовали equals и hashCode на основе идентификатора, как и для назначенных идентификаторов.

Эта проблема подробно описываестя в 2-м пункте.

Следовательно, всякий раз, когда у вас есть идентификатор, созданный базой данных, синтетический ключ (будь то числовой идентификатор или [тип UUID базы данных](https://www.postgresql.org/docs/current/static/datatype-uuid.html) ), вы должны использовать следующую реализацию equals и hashCode:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32** | **@Entity(name = "Post")**  **@Table(name = "post")**  **public class Post implements Identifiable<Long> {**    **@Id**  **@GeneratedValue**  **private Long id;**    **private String title;**    **public Post() {}**    **@Override**  **public boolean equals(Object o) {**  **if (this == o) return true;**    **if (!(o instanceof Post))**  **return false;**    **Post other = (Post) o;**    **return id != null &&**  **id.equals(other.getId());**  **}**    **@Override**  **public int hashCode() {**  **return getClass().hashCode();**  **}**    **//Getters and setters omitted for brevity**  **}** |

Таким образом, результат hashCodeдает одно и то же значение для всех переходов состояний сущностей, и equalsметод будет использовать проверку идентификатора только для непереходных сущностей.

Это оно!

Единственный случай, когда вы увидите узкое место в производительности из-за одного хеш-ведра, - это если у вас большая коллекция из десятков тысяч записей.

Но тогда это означает, что вы получили эту большую коллекцию из базы данных. Ухудшение производительности при выборке такой коллекции из базы данных на несколько порядков выше, чем накладные расходы на одну корзину.

Вот почему вы никогда не сопоставляете большие коллекции с Hibernate. Вместо этого вы используете запросы. Но тогда для небольших коллекций.

Кроме того, в большинстве случаев вам даже не нужно использовать a Setили Map. Для двунаправленных ассоциаций List(s)все равно лучше работать.

Больше заблуждений

Марк написал [сообщение](https://struberg.wordpress.com/2016/10/15/tostring-equals-and-hashcode-in-jpa-entities/) в [блоге,](https://struberg.wordpress.com/2016/10/15/tostring-equals-and-hashcode-in-jpa-entities/) чтобы оправдать свои убеждения.

В своей статье Маркс говорит, что реализация равенства идентификаторов, сгенерированных базой данных, не работает для mergeили getReference().

Даже в продвинутой версии Влада есть дыры. Например, если вы используете em.getReference () или em.merge ().

[Как реализовать equals и hashCode с использованием идентификатора объекта JPA (первичного ключа),](https://vladmihalcea.com/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/) статья демонстрирует, что эта реализация equals работает для отсоединенных объектов. В этом был весь смысл такой реализации. Мы хотим, чтобы он работал при всех переходах между состояниями сущностей.

Что касается этого getReference(), то там тоже есть проверка. Это все на [GitHub](https://github.com/vladmihalcea/high-performance-java-persistence/blob/master/core/src/test/java/com/vladmihalcea/book/hpjp/hibernate/equality/AbstractEqualityCheckTest.java" \l "L55) .

Есть один аргумент, с которым я согласен, и он касается того, чтобы проверка на равенство использовала только неизменяемые атрибуты сущности. Вот почему порядковый номер идентификатора объекта очень привлекателен. И с методом реализации равенства, который я вам предлагаю, вы можете безопасно использовать его.

К сожалению, Марк продолжает заблуждаться, например:

Зачем вообще нужны equals () и hashCode ()?

Это хороший вопрос. И мой ответ: «Вы этого не сделаете!»

Ну ты вообще!

Если вы не реализуете equalsи hashCodeзатем тест слияния не получится , поэтому нарушение гарантии целостности. Все это объясняется в моей статье « [Как реализовать equals и hashCode с использованием идентификатора объекта (первичного ключа)»](https://vladmihalcea.com/2016/06/06/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/) , кстати.

И еще одно заблуждение с точки зрения гибернации.

Почему не следует хранить управляемые и отсоединенные объекты в одной коллекции

Мало того, что вам НЕ следует избегать смешивания отдельных и управляемых сущностей, но это на самом деле отличная функция, которая позволяет вам удерживать отсоединенные объекты и, следовательно, предотвращать [потерю обновлений в долгих разговорах](https://vladmihalcea.com/preventing-lost-updates-in-long-conversations/) .

И еще одно заблуждение с точки зрения реализации Hibernate:

Итак, наличие кеша - действительно отличная идея, но \* пожалуйста \* не храните объекты JPA в кеше. По крайней мере, пока ими управляют.

Hibernate стремится обеспечить стабильную согласованность. Вот почему стратегии параллелизма кэша [READ\_WRITE](https://vladmihalcea.com/how-does-hibernate-read_write-cacheconcurrencystrategy-work/) и [TRANSACTIONAL](https://vladmihalcea.com/how-does-hibernate-transactional-cacheconcurrencystrategy-work/) позволяют не беспокоиться о таких несоответствиях. Этот уровень изоляции гарантирует поставщик кеша второго уровня. Прямо как система реляционных баз данных.

Только [NONSTRICT\_READ\_WRITE](https://vladmihalcea.com/how-does-hibernate-nonstrict_read_write-cacheconcurrencystrategy-work/) предлагает более слабый уровень изоляции, но выбор *нестрогого* наименования в конце концов [информативен](https://vladmihalcea.com/how-does-hibernate-nonstrict_read_write-cacheconcurrencystrategy-work/) .

2. Как реализовать equals и hashCode с использованием идентификатора объекта JPA (первичный ключ)

https://vladmihalcea.com/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/

Последнее изменение: 19 ноября 2020 г.

Представьте, что у вас есть инструмент, который может автоматически обнаруживать проблемы с производительностью JPA и Hibernate. [Оптимизатор гиперсистентности](https://vladmihalcea.com/hypersistence-optimizer/) - вот такой инструмент!

Вступление

Как [объяснялось ранее](https://vladmihalcea.com/hibernate-facts-equals-and-hashcode/) , использование бизнес-ключа сущности JPA для equalsи hashCodeвсегда является лучшим выбором. Однако не все сущности имеют уникальный бизнес-ключ, поэтому нам нужно использовать другой столбец базы данных, который также является уникальным, в качестве первичного ключа.

Но использовать идентификатор объекта для равенства очень сложно, и этот пост покажет вам, как его можно использовать без проблем.

Испытательная привязь

Когда дело доходит до реализации equalsи hashCode, вы должны иметь в виду одно и только одно правило:

Equals и hashCode должны вести себя согласованно при всех [переходах состояния объекта](https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/) .

Для того, чтобы проверить эффективность equalsи hashCodeреализации, следующий тест может быть использован:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | protected void assertEqualityConsistency(          Class<T> clazz,          T entity) {        Set<T> tuples = new HashSet<>();        assertFalse(tuples.contains(entity));      tuples.add(entity);      assertTrue(tuples.contains(entity));        doInJPA(entityManager -> {          entityManager.persist(entity);          entityManager.flush();          assertTrue(              "The entity is not found in the Set after it's persisted.",              tuples.contains(entity)          );      });        assertTrue(tuples.contains(entity));        doInJPA(entityManager -> {          T entityProxy = entityManager.getReference(              clazz,              entity.getId()          );          assertTrue(              "The entity proxy is not equal with the entity.",              entityProxy.equals(entity)          );      });        doInJPA(entityManager -> {          T entityProxy = entityManager.getReference(              clazz,              entity.getId()          );          assertTrue(              "The entity is not equal with the entity proxy.",              entity.equals(entityProxy));      });        doInJPA(entityManager -> {          T \_entity = entityManager.merge(entity);          assertTrue(              "The entity is not found in the Set after it's merged.",              tuples.contains(\_entity)          );      });        doInJPA(entityManager -> {          entityManager.unwrap(Session.class).update(entity);          assertTrue(              "The entity is not found in the Set after it's reattached.",              tuples.contains(entity)          );      });        doInJPA(entityManager -> {          T \_entity = entityManager.find(clazz, entity.getId());          assertTrue(              "The entity is not found in the Set after it's loaded in a different Persistence Context.",              tuples.contains(\_entity)          );      });        doInJPA(entityManager -> {          T \_entity = entityManager.getReference(clazz, entity.getId());          assertTrue(              "The entity is not found in the Set after it's loaded as a proxy in a different Persistence Context.",              tuples.contains(\_entity)          );      });        T deletedEntity = doInJPA(entityManager -> {          T \_entity = entityManager.getReference(              clazz,              entity.getId()          );          entityManager.remove(\_entity);          return \_entity;      });        assertTrue(          "The entity is not found in the Set even after it's deleted.",          tuples.contains(deletedEntity)      );  } |

Естественный идентификатор

Первый вариант использования, который нужно проверить, - это *естественное* сопоставление *идентификаторов* . Учитывая следующую сущность:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24  25  26 год  27 | @Entity  public class Book implements Identifiable<Long> {        @Id      @GeneratedValue      private Long id;        private String title;        @NaturalId      private String isbn;        @Override      public boolean equals(Object o) {          if (this == o) return true;          if (!(o instanceof Book)) return false;          Book book = (Book) o;          return Objects.equals(getIsbn(), book.getIsbn());      }        @Override      public int hashCode() {          return Objects.hash(getIsbn());      }        //Getters and setters omitted for brevity  } |

Свойство isbn также @NaturalId, следовательно, он должен быть уникальным и не обнуляемым. Оба equals и hashCode используют свойство isbn в своих реализациях.

Подробнее об @NaturalId аннотации читайте в [этой статье](https://vladmihalcea.com/the-best-way-to-map-a-naturalid-business-key-with-jpa-and-hibernate/) .

При запуске следующего тестового примера:

|  |  |
| --- | --- |
| 1  2  3  4  5 | Book book = new Book();  book.setTitle("High-PerformanceJava Persistence");  book.setIsbn("123-456-7890");    assertEqualityConstraints(Book.class, book); |

Все работает нормально, как и ожидалось.

По умолчанию java.lang.Object равно и hashCode

Что, если в нашей сущности нет столбца, который можно было бы использовать в качестве @NaturalId? Первое побуждение - не определять свои собственные реализации equals и hashCode, как в следующем примере:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Entity(name = "Book")  public class Book implements Identifiable<Long> {        @Id      @GeneratedValue      private Long id;        private String title;        //Getters and setters omitted for brevity  } |

Однако при тестировании этой реализации:

|  |  |
| --- | --- |
| 1  2  3  4 | Book book = new Book();  book.setTitle("High-PerformanceJava Persistence");    assertEqualityConstraints(Book.class, book); |

Hibernate выдает следующее исключение:

|  |  |
| --- | --- |
| 1 | java.lang.AssertionError: The entity is not found after it's merged |

Исходная сущность не совпадает с сущностью, возвращаемой методом слияния, потому что два разных объекта (ов) не используют одну и ту же ссылку.

Использование идентификатора объекта для equals и hashCode

Итак, если значение по умолчанию equals и hashCode это тоже не подходит, тогда давайте использовать идентификатор объекта для нашей пользовательской реализации. Давайте просто использовать наш IDE для генерации equals и hashCode увидеть , как это работает:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24 | @Entity  public class Book implements Identifiable<Long> {        @Id      @GeneratedValue      private Long id;        private String title;        @Override      public boolean equals(Object o) {          if (this == o) return true;          if (!(o instanceof Book)) return false;          Book book = (Book) o;          return Objects.equals(getId(), book.getId());      }        @Override      public int hashCode() {          return Objects.hash(getId());      }        //Getters and setters omitted for brevity  } |

При запуске предыдущего тестового примера Hibernate выдает следующее исключение:

|  |  |
| --- | --- |
| 1 | java.lang.AssertionError: The entity is not found after it's persisted |

Когда объект был впервые сохранен в наборе, идентификатор был нулевым. После того, как объект был сохранен, идентификатор был присвоен автоматически сгенерированному значению, поэтому хэш-код отличается. По этой причине объект не может быть найден в наборе после того, как он был сохранен.

Исправление идентификатора объекта, равного и hashCode

Для решения предыдущей проблемы есть только одно решение: hashCode всегда должен возвращать одно и то же значение:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26** | **@Entity**  **public class Book implements Identifiable<Long> {**    **@Id**  **@GeneratedValue**  **private Long id;**    **private String title;**    **@Override**  **public boolean equals(Object o) {**  **if (this == o) return true;**    **if (!(o instanceof Book))**  **return false;**    **Book other = (Book) o;**    **return id != null &&**  **id.equals(other.getId());**  **}**    **@Override**  **public int hashCode() {**  **return getClass().hashCode();**  **}**    **//Getters and setters omitted for brevity**  **}** |

Кроме того, когда идентификатор объекта равен null, мы можем гарантировать равенство только для одинаковых ссылок на объекты. В противном случае ни один временный объект не может быть равен любому другому временному или постоянному объекту. Поэтому проверка равенства идентификатора выполняется только в том случае, если текущий Objectидентификатор не равен нулю.

С этой реализацией, equalsи hashCodeтест работает нормально для всех сущности переходов состояний. Причина, по которой это работает, заключается в том, что значение hashCode не меняется, следовательно, мы можем полагаться на java.lang.Objectссылочное равенство, пока есть идентификатор null.

Заключение

Идентификатор объекта можно использовать для equalsи hashCode, но только если он все время hashCodeвозвращает одно и то же значение. Это может показаться ужасным поступком, поскольку это лишает смысла использование нескольких сегментов в файле HashSetor HashMap.

Однако из соображений производительности всегда следует ограничивать количество сущностей, хранящихся в коллекции. Никогда не следует извлекать тысячи сущностей в a, @OneToMany Setпотому что снижение производительности на стороне базы данных на несколько порядков выше, чем при использовании одного хешированного ведра.

3. Как реализовать Equals и HashCode для сущностей JPA

https://vladmihalcea.com/hibernate-facts-equals-and-hashcode/

Последнее изменение: 14 августа 2020 г.

Представьте, что у вас есть инструмент, который может автоматически обнаруживать проблемы с производительностью JPA и Hibernate. [Оптимизатор гиперсистентности](https://vladmihalcea.com/hypersistence-optimizer/) - вот такой инструмент!

Вступление

Каждый объект Java наследует методы equals и hashCode, но они полезны только для объектов Value и бесполезны для объектов, ориентированных на поведение без сохранения состояния.

Хотя сравнение ссылок с использованием оператора «==» несложно, для равенства объектов все немного сложнее.

Требования

Поскольку вы отвечаете за то, что означает равенство для определенного типа объекта, обязательно, чтобы ваши реализации equals и hashCode следовали всем правилам, указанным в java.lang.Object JavaDoc ( [equals](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \l "equals%28java.lang.Object%29) и [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \l "hashCode%28%29) ).

Также важно знать, как ваше приложение (и используемые в нем фреймворки) используют эти два метода.

К счастью, Hibernate не требует, чтобы они проверяли, изменились ли объекты, имея для этой цели специальный [механизм грязной проверки](https://vladmihalcea.com/the-anatomy-of-hibernate-dirty-checking/) .

В документации Hiberante перечислены ситуации, когда требуются эти два метода:

* *при добавлении сущностей в набор коллекций*
* *при повторном присоединении сущностей к новому контексту сохранения*

Эти требования возникают из Object.equals« *непротиворечивого* » ограничения, которое приводит нас к следующему принципу:

**Сущность должна быть равна себе во всех**[**состояниях объекта JPA**](https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/) :

* преходящий
* прикрепил
* обособленный
* удалено (пока объект отмечен для удаления и все еще находится в куче)

Таким образом, можно сделать вывод, что:

* Мы не можем использовать автоматически увеличивающийся идентификатор базы данных в hashCodeметоде, поскольку временная и присоединенная версии объекта больше не будут находиться в одном хешированном сегменте.
* Мы не можем полагаться на значения по умолчанию Object equalsи hashCodeреализации, поскольку две сущности, загруженные в двух разных контекстах персистентности, в конечном итоге станут двумя разными объектами Java, что нарушит правило равенства всех состояний.
* Итак, если Hibernate использует равенство для однозначной идентификации Objectобъекта в течение всего времени его существования, нам нужно найти правильную комбинацию свойств, удовлетворяющую этому требованию.

Деловое равенство

Те поля сущности, которые обладают свойством быть уникальными во всем пространстве объектов сущностей, обычно называются бизнес-ключом.

Бизнес-ключ также не зависит от какой-либо технологии сохранения, используемой в нашей архитектуре проекта, в отличие от автоматически увеличиваемого идентификатора синтетической базы данных.

Итак, бизнес-ключ должен быть установлен с того момента, как мы создаем Entity, и никогда не менять его.

Давайте возьмем несколько примеров сущностей относительно их зависимостей и выберем соответствующий бизнес-ключ.

Вариант использования корневого объекта (объект без родительской зависимости)

Вот как реализованы equals / hashCode:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24  25  26 год  27  28 год  29  30 | @Entity  public class Company {      @Id      @GeneratedValue(strategy = GenerationType.AUTO)      private Long id;        @Column(unique = true, updatable = false)      private String name;        @Override      public int hashCode() {          HashCodeBuilder hcb = new HashCodeBuilder();          hcb.append(name);          return hcb.toHashCode();      }        @Override      public boolean equals(Object obj) {          if (this == obj) {              return true;          }          if (!(obj instanceof Company)) {              return false;          }          Company that = (Company) obj;          EqualsBuilder eb = new EqualsBuilder();          eb.append(name, that.name);          return eb.isEquals();      }  } |

Поле имени представляет бизнес-ключ компании и поэтому объявлено уникальным и не обновляемым. Таким образом, два объекта Company равны, если у них одинаковое имя, игнорируя любые другие поля, которые он может содержать.

Дочерние сущности с выбранным родителем EAGER

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24  25  26 год  27  28 год  29  30  31 год  32  33  34  35 год  36  37  38  39  40  41 год  42  43 год  44 год | @Entity  public class Product {      @Id      @GeneratedValue(strategy = GenerationType.AUTO)      private Long id;        @Column(updatable = false)      private String code;        @ManyToOne(fetch = FetchType.EAGER)      @JoinColumn(name = "company\_id",                  nullable = false, updatable = false)      private Company company;        @OneToMany(fetch = FetchType.LAZY,                 cascade = CascadeType.ALL,                 mappedBy = "product",                 orphanRemoval = true)      @OrderBy("index")      private Set images = new LinkedHashSet();        @Override      public int hashCode() {          HashCodeBuilder hcb = new HashCodeBuilder();          hcb.append(code);          hcb.append(company);          return hcb.toHashCode();      }        @Override      public boolean equals(Object obj) {          if (this == obj) {              return true;          }          if (!(obj instanceof Product)) {              return false;          }          Product that = (Product) obj;          EqualsBuilder eb = new EqualsBuilder();          eb.append(code, that.code);          eb.append(company, that.company);          return eb.isEquals();      }  } |

В этом примере мы всегда выбираем Компанию для Продукта, и поскольку код Продукта не уникален среди Компаний, мы можем включить родительскую сущность в наш бизнес-ключ. Родительская ссылка помечена как не обновляемая, чтобы предотвратить нарушение контракта equals / hashCode (перенос продукта из одной компании в другую в любом случае не имеет смысла). Но эта модель нарушается, если у Parent есть набор дочерних сущностей, и вы вызываете что-то вроде:

|  |  |
| --- | --- |
| 1  2  3  4 | public void removeChild(Child child) {      children.remove(child);      child.setParent(null);  } |

Это нарушит контракт equals / hashCode, поскольку для родительского объекта установлено значение null, и дочерний объект не будет найден в коллекции детей, если это был Set. Поэтому будьте осторожны при использовании двунаправленных ассоциаций с дочерними объектами, использующими этот тип equals / hashCode.

Дочерние сущности с ленивым родительским элементом

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 год  22  23  24  25  26 год  27  28 год  29  30  31 год  32  33  34  35 год  36  37 | @Entity  public class Image {      @Id      @GeneratedValue(strategy = GenerationType.AUTO)      private Long id;        @Column(updatable = false)      private String name;        @ManyToOne(fetch = FetchType.LAZY)      @JoinColumn(name = "product\_id", nullable = false,                  updatable = false)      private Product product;        @Override      public int hashCode() {          HashCodeBuilder hcb = new HashCodeBuilder();          hcb.append(name);          hcb.append(product);          return hcb.toHashCode();      }        @Override      public boolean equals(Object obj) {          if (this == obj) {              return true;          }          if (!(obj instanceof Image)) {              return false;          }          Image that = (Image) obj;          EqualsBuilder eb = new EqualsBuilder();          eb.append(name, that.name);          eb.append(product, that.product);          return eb.isEquals();      }  } |

Если изображения получены без продукта, а контекст сохранения закрыт, и мы загружаем изображения в набор, мы получим исключение LazyInitializationException, как в следующем примере кода:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | List images = transactionTemplate.execute(new TransactionCallback<List>() {      @Override      public List doInTransaction(TransactionStatus transactionStatus) {          return entityManager.createQuery(              "select i from Image i ", Image.class)          .getResultList();      }  });    //Throws LazyInitializationException |

Поэтому я бы не рекомендовал этот вариант использования, поскольку он подвержен ошибкам, и для правильного использования equals и hashCode нам всегда нужно инициализировать LAZY-ассоциации.

Дочерние сущности, игнорирующие родителя

В этом случае мы просто удаляем родительскую ссылку из нашего бизнес-ключа. Пока мы всегда используем Child через коллекцию Parent children, мы в безопасности. Если мы загружаем дочерние элементы от нескольких родителей, и бизнес-ключ не является уникальным среди них, мы не должны добавлять их в коллекцию Set, поскольку Set может отбрасывать дочерние объекты, имеющие один и тот же бизнес-ключ, от разных родителей.

Заключение

Если вы хотите использовать идентификатор объекта при реализации equalsи hashCode, ознакомьтесь с [этим постом,](https://vladmihalcea.com/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/) чтобы узнать, как это сделать правильно.

Выбор правильного бизнес-ключа для сущности - нетривиальная задача, поскольку она отражается на использовании вашей сущности внутри и за пределами области Hibernate. Использование комбинации полей, уникальных для Entities, вероятно, лучший выбор для реализации методов equals и hashCode.

Использование [EqualsBuilder](http://commons.apache.org/proper/commons-lang/javadocs/api-2.5/org/apache/commons/lang/builder/EqualsBuilder.html) и [HashCodeBuilder](http://commons.apache.org/proper/commons-lang/javadocs/api-2.5/org/apache/commons/lang/builder/HashCodeBuilder.html) помогает нам писать краткие реализации equals и hashCode, и, похоже, это также работает с прокси Hibernate.