Веденин Вячеслав @vedenin1980
Java developer
48,7 карма 31,0 рейтинг

3 сентября 2015 в 10:54

Разработка → Шпаргалка Java программиста 1: JPA и Hibernate в вопросах и ответах tutorial

Разработка веб-сайтов*, Программирование*, Open source*, Java*, API*

Знаете ли вы JPA? А Hibernate? А если проверить?

▾ В чем смысл серии статей 'Шпаргалки Java программиста'

За время работы Java программистом я заметил, что как правило программисты постоянно и планомерно используют от силы 10-20% от возможностей той или иной технологии, при этом остальные возможности быстро забываются и при появлении новых требований, переходе на новую работу или подготовке к техническому интервью приходится перечитывать все документации и спецификации с нуля. Зато наличие краткого конспекта особенностей тех или иных технологий (шпаргалок) позволяет быстро освежить в памяти особенности той или иной технологии.

▾ Общее оглавление 'Шпаргалок'

Данная статья будет полезна и для тех кто только собирается изучать JPA и Hibernate (В этом случае рекомендую сразу открывать ответы), и для тех кто уже хорошо знает JPA и Hibernate (В этом случае статья позволит проверить свои знания и освежить особенности технологий). Особенно статья будет полезна тем кто собирается пройти техническое интервью, где возможно будут задавать вопросы по JPA и Hibernate (или сам собирается провести техническое интервью).

Рекомендую так считать правильные ответы: если вы ответили на вопрос по вашему мнению правильно и полностью — поставьте себе 1 балл, если ответили только частично — 0.5 балл. Везде где только возможно я старался добавлять цитаты из оригинальной документации (но из-за ограничений лицензии Oracle не могу давать слишком большие цитаты из документации).

Общие вопросы

- Вопрос 1. Что такое JPA?
  Ответ

  JPA (Java Persistence API) это спецификация Java EE и Java SE, описывающая систему управления сохранением java объектов в таблицы реляционных баз данных в удобном виде. Сама Java не содержит реализации JPA, однако есть существует много реализаций данной спецификации от разных компаний (открытых и нет). Это не единственный способ сохранения java объектов в базы данных (ORM систем), но один из самых популярных в Java мире.

  Оригинал

- Вопрос 2. В чем её отличие JPA от Hibernate?
  Ответ

  Hibernate одна из самых популярных открытых реализаций последней версии спецификации (JPA 2.1). Даже скорее самая популярная, почти стандарт де-факто. То есть JPA только описывает правила и API, а Hibernate реализует эти описания, впрочем у Hibernate (как и у многих других реализаций JPA) есть дополнительные возможности, не описанные в JPA (и не переносимые на другие реализации JPA).

- Вопрос 3. Можно ли использовать JPA с noSQl базами?
  Ответ

  Вообще, спецификация JPA говорит только о отображении java объектов в таблицы реляционных баз данных, но при этом существует ряд реализаций данного стандарта для noSql баз данных: Kundera, DataNucleus, ObjectDB и ряд других. Естественно, при это не все специфичные для реляционных баз данных особенности спецификации переносятся при этом на nosql базы полностью.

- Вопрос 4. В чем её отличие JPA от JDO?
  Ответ

  JPA (Java Persistence API) и Java Data Objects (JDO) две спецификации сохранения java объектов в базах данных. Если JPA сконцентрирована только на реляционных базах, то JDO более общая спецификация которая описывает ORM для любых возможных баз и хранилищ. В принципе можно рассматривать JPA как специализированную на релятивистских баз часть спецификации JDO, даже при том что API этих двух спецификаций не полностью совпадает. Также отличаются «разработчики» спецификаций — если JPA разрабатывается как JSR, то JDO сначала разрабатывался как JSR, теперь разрабатывается как проект Apache JDO. Подробнее про сравнения функционала и API спецификаций можно посмотреть здесь.

- Вопрос 5. Что такое Entity?
  Ответ

  Entity это легковесный хранимый объект бизнес логики (persistent domain object). Основная программная сущность это entity класс, который так же может использовать дополнительные классы, который могут использоваться как вспомогательные классы или для сохранения состояния entity.

  Java Persistence 2.1. Chapter 2

  Оригинал

- Вопрос 6. Может ли Entity класс наследоваться от не Entity классов (non-entity classes)?
  Ответ

  Может

  Java Persistence 2.1. Chapter 2.1

  Оригинал

- Вопрос 7. Может ли Entity класс наследоваться от других Entity классов?
  Ответ

  Тоже может

  Java Persistence 2.1. Chapter 2.1

  Оригинал

- Вопрос 8. Может ли не Entity класс наследоваться от Entity класса?
  Ответ

  И это тоже допустимо

  Java Persistence 2.1. Chapter 2.1

  Оригинал

- Вопрос 9. Может ли Entity быть абстрактным классом?
  Ответ

  Может, при этом он сохраняет все свойства Entity, за исключением того что его нельзя непосредственно инициализировать.

  Java Persistence 2.1. Chapter 2.11.1

  Оригинал

- Вопрос 10. Какие требования JPA к Entity классам вы можете перечислить (не менее шести требований)?
  Ответ

  1) Entity класс должен быть отмечен аннотацией Entity или описан в XML файле конфигурации JPA,
  2) Entity класс должен содержать public или protected конструктор без аргументов (он также может иметь конструкторы с аргументами),
  3) Entity класс должен быть классом верхнего уровня (top-level class),
  4) Entity класс не может быть enum или интерфейсом,
  5) Entity класс не может быть финальным классом (final class),
  6) Entity класс не может содержать финальные поля или методы, если они участвуют в маппинге (persistent final methods or persistent final instance variables),
  7) Если объект Entity класса будет передаваться по значению как отдельный объект (detached object), например через удаленный интерфейс (through a remote interface), он так же должен реализовывать Serializable интерфейс,
  8) Поля Entity класс должны быть напрямую доступны только методам самого Entity класса и не должны быть напрямую доступны другим классам, использующим этот Entity. Такие классы должны обращаться только к методам (getter/setter методам или другим методам бизнес-логики в Entity классе),
  9) Enity класс должен содержать первичный ключ, то есть атрибут или группу атрибутов которые уникально определяют запись этого Enity класса в базе данных,

  Java Persistence 2.1. Chapter 2.1 and 2.4

  Оригинал

- Вопрос 11. Какие два типа элементов есть у Entity классов. Или другими словами перечислите два типа доступа (access) к элементам Entity классов.
  Ответ

  JPA указывает что она может работать как с свойствами классов (property), оформленные в стиле JavaBeans, либо с полями (field), то есть переменными класса (instance variables). Соответственно, при этом тип доступа будет либо property access или field access.

Java Persistence 2.1. Chapter 2.2

Оригинал

- Вопрос 12. Что такое атрибут Entity класса в терминологии JPA?
  Ответ

  JPA указывает что она может работать как с свойствами классов (property), оформленные в стиле JavaBeans, либо с полями (field), то есть переменными класса (instance variables). Оба типа элементов Entity класса называются атрибутами Entity класса.

  Java Persistence 2.1. Chapter 2.2

  Оригинал

- Вопрос 13. Какие типы данных допустимы в атрибутах Entity класса (полях или свойствах)?
  Ответ

  Допустимые типы атрибутов у Entity классов:
  1. примитивные типы и их обертки Java,
  2. строки,
  3. любые сериализуемые типы Java (реализующие Serializable интерфейс),
  4. enums;
  5. entity types;
  6. embeddable классы
  7. и коллекции типов 1-6

  Java Persistence 2.1. Chapter 2.2

  Оригинал

- Вопрос 14. Какие типы данных можно использовать в атрибутах, входящих в первичный ключ Entity класса (составной или простой), чтобы полученный первичный ключ мог использоваться для любой базы данных? А в случае автогенерируемого первичного ключа (generated primary keys)?
  Ответ

  Допустимые типы атрибутов, входящих в первичный ключ:
  1. примитивные типы и их обертки Java,
  2. строки,
  3. BigDecimal и BigInteger,
  4. java.util.Date и java.sql.Date,

  В случае автогенерируемого первичного ключа (generated primary keys) допустимы
  1. только числовые типы,

  В случае использования других типов данных в первичном ключе, он может работать только для некоторых баз данных, т.е. становится не переносимым (not portable),

  Java Persistence 2.1. Chapter 2.4

  Оригинал

Сложные структуры JPA

- Вопрос 15. Что такое встраиваемый (Embeddable) класс?
  Ответ

  Встраиваемый (Embeddable) класс это класс который не используется сам по себе, только как часть одного или нескольких Entity классов. Entity класс могут содержать как одиночные встраиваемые классы, так и коллекции таких классов. Также такие классы могут быть использованы как ключи или значения map. Во время выполнения каждый встраиваемый класс принадлежит только одному объекту Entity класса и не может быть использован для передачи

данных между объектами Entity классов (то есть такой класс не является общей структурой данных для разных объектов). В целом, такой класс служит для того чтобы выносить определение общих атрибутов для нескольких Entity, можно считать что JPA просто встраивает в Entity вместо объекта такого класса те атрибуты, которые он содержит.

Java Persistence 2.1. Chapter 2.5

▸ Оригинал

- Вопрос 16. Может ли встраиваемый (Embeddable) класс содержать другой встраиваемый (Embeddable) класс?
  ▾ Ответ

Да, может

Java Persistence 2.1. Chapter 2.5

▸ Оригинал

- Вопрос 17. Может ли встраиваемый (Embeddable) класс содержать связи (relationship) с другими Entity или коллекциями Entity? Если может, то существуют ли какие-то ограничение на такие связи (relationship)?
  ▾ Ответ

Может, но только в случае если такой класс не используется как первичный ключ или ключ map'ы.

Java Persistence 2.1. Chapter 2.5
▸ Оригинал

- Вопрос 18. Какие требования JPA устанавливает к встраиваемым (Embeddable) классам?
  ▾ Ответ

1. Такие классы должны удовлетворять тем же правилам что Entity классы, за исключением того что они не обязаны содержать первичный ключ и быть отмечены аннотацией Entity (см. вопрос 10),
2. Embeddable класс должен быть отмечен аннотацией Embeddable или описан в XML файле конфигурации JPA,

Java Persistence 2.1. Chapter 2.5
▸ Оригинал

- Вопрос 19. Какие типы связей (relationship) между Entity вы знаете (перечислите восемь типов, либо укажите четыре типа связей, каждую из которых можно разделить ещё на два вида)?
  ▾ Ответ

Существуют следующие четыре типа связей
1. OneToOne (связь один к одному, то есть один объект Entity может связан не больше чем с один объектом другого Entity),
2. OneToMany (связь один ко многим, один объект Entity может быть связан с целой коллекцией других Entity),
3. ManyToOne (связь многие к одному, обратная связь для OneToMany),
4. ManyToMany (связь многие ко многим)

Каждую из которых можно разделить ещё на два вида:
1. Bidirectional
2. Unidirectional

Bidirectional — ссылка на связь устанавливается у всех Entity, то есть в случае OneToOne A-B в Entity A есть ссылка на Entity B, в Entity B есть ссылка на Entity A, Entity A считается владельцем этой связи (это важно для случаев каскадного удаления данных, тогда при удалении A также будет удалено B, но не наоборот).

Undirectional- ссылка на связь устанавливается только с одной стороны, то есть в случае OneToOne A-B только у Entity A будет ссылка на Entity B, у Entity B ссылки на A не будет.

Java Persistence 2.1. Chapter 2.9
▸ Оригинал

- Вопрос 20. Что такое Mapped Superclass?
  Ответ

  Mapped Superclass это класс от которого наследуются Entity, он может содержать анотации JPA, однако сам такой класс не является Entity, ему не обязательно выполнять все требования установленные для Entity (например, он может не содержать первичного ключа). Такой класс не может использоваться в операциях EntityManager или Query. Такой класс должен быть отмечен аннотацией MappedSuperclass или соответственно описан в xml файле.

  Java Persistence 2.1. Chapter 2.2
  Примеры

  Оригинал

- Вопрос 21. Какие три типы стратегии наследования мапинга (Inheritance Mapping Strategies) описаны в JPA?
  Ответ

  В JPA описаны три стратегии наследования мапинга (Inheritance Mapping Strategies), то есть как JPA будет работать с классами-наследниками Entity:
  1) одна таблица на всю иерархию наследования (**a single table per class hierarchy**) — все enity, со всеми наследниками записываются в одну таблицу, для идентификации типа entity определяется специальная колонка **"discriminator column"**. Например, если есть entity Animals с классами-потомками Cats и Dogs, при такой стратегии все entity записываются в таблицу Animals, но при это имеют дополнительную колонку animalType в которую соответственно пишется значение «cat» или «dog». **Минусом** является то что в общей таблице, будут созданы все поля уникальные для каждого из классов-потомков, которые будет пусты для всех других классов-потомков. Например, в таблице animals окажется и скорость лазанья по дереву от cats и может ли пес приносить тапки от dogs, которые будут всегда иметь null для dog и cat соотвественно.

  2) объединяющая стратегия (**joined subclass strategy**) — в этой стратегии каждый класс enity сохраняет данные в свою таблицу, но только уникальные колонки (не унаследованные от классов-предков) и первичный ключ, а все унаследованные колонки записываются в таблицы класса-предка, дополнительно устанавливается связь (relationships) между этими таблицами, например в случае классов Animals (см.выше), будут три таблицы animals, cats, dogs, причем в cats будет записана только ключ и скорость лазанья, в dogs — ключ и умеет ли пес приносить палку, а в animals все остальные данные cats и dogs с ссылкой на соответствующие таблицы. **Минусом** тут являются потери производительности от объединения таблиц (join) для любых операций.

  3) одна таблица для каждого класса (**table per concrete class strategy**) — тут все просто каждый отдельный класс-наследник имеет свою таблицу, т.е. для cats и dogs (см.выше) все данные будут записываться просто в таблицы cats и dogs как если бы они вообще не имели общего суперкласса. **Минусом** является плохая поддержка полиморфизма (polymorphic relationships) и то что для выборки всех классов иерархии потребуются большое количество отдельных sql запросов или использование UNION запроса.

  Для задания стратегии наследования используется аннотация **Inheritance** (или соответствующие блоки
  Java Persistence 2.1. Chapter 2.12, J7EE javadoc

  Примеры

  Примеры на github: TABLE_PER_CLASS: 1 2 3, JOINED:1 2 3 ,SINGLE_TABLE: 1 2 3

```
@Entity
@Inheritance(strategy=JOINED)
public class Customer { ... }
@Entity
public class ValuedCustomer extends Customer { ... }
```

  Оригинал

  There are three basic strategies that are used when mapping a class or class hierarchy to a relational database:
  • a single table per class hierarchy
  • a joined subclass strategy, in which fields that are specific to a subclass are mapped to a separate table than the fields that are common to the parent class, and a join is performed to instantiate

the subclass.
• a table per concrete entity class

- Вопрос 22. Какие два типа fetch стратегии в JPA вы знаете?
  Ответ

В JPA описаны два типа fetch стратегии:
1) LAZY — данные поля будут загруженны только во время первого доступа к этому полю,
2) EAGER — данные поля будут загруженны немедленно,

Java Persistence 2.1. Chapter 11.1.6 J7EE javadoc

Оригинал

Основные операции с Entity

- Вопрос 23. Что такое EntityManager и какие основные его функции вы можете перечислить?
  Ответ

EntityManager это интерфейс, который описывает API для всех основных операций над Enitity, получение данных и других сущностей JPA. По сути главный API для работы с JPA. Основные операции:
1) Для операций над Entity: persist (добавление Entity под управление JPA), merge (обновление), remove (удаления), refresh (обновление данных), detach (удаление из управление JPA), lock (блокирование Enity от изменений в других thread),
2) Получение данных: find (поиск и получение Entity), createQuery, createNamedQuery, createNativeQuery, contains, createNamedStoredProcedureQuery, createStoredProcedureQuery
3) Получение других сущностей JPA: getTransaction, getEntityManagerFactory, getCriteriaBuilder, getMetamodel, getDelegate
4) Работа с EntityGraph: createEntityGraph, getEntityGraph
4) Общие операции над EntityManager или всеми Entities: close, isOpen, getProperties, setProperty, clear

Java Persistence 2.1. Chapter 3.1.1 J7EE javadoc
Примеры

Оригинал

- Вопрос 24. Какие четыре статуса жизненного цикла Entity объекта (Entity Instance’s Life Cycle) вы можите перечислить?
  Ответ

У Entity объекта существует четыре статуса жизненного цикла: new, managed, detached, или removed. Их описание
1) new — объект создан, но при этом ещё не имеет сгенерированных первичных ключей и пока ещё не сохранен в базе данных,
2) managed — объект создан, управляется JPA, имеет сгенерированные первичные ключи,
3) detached — объект был создан, но не управляется (или больше не управляется) JPA,
4) removed — объект создан, управляется JPA, но будет удален после commit'а транзакции.

Java Persistence 2.1. Chapter 3.2

Оригинал

- Вопрос 25. Как влияет операция persist на Entity объекты каждого из четырех статусов?
  Ответ

1) Если статус Entity new, то он меняется на managed и объект будет сохранен в базу при commit'е транзакции или в результате flush операций,
2) Если статус уже managed, операция игнорируется, однако зависимые Entity могут поменять статус на managed, если у них есть аннотации каскадных изменений,
3) Если статус removed, то он меняется на managed,
4) Если статус detached, будет выкинут exception сразу или на этапе commit'а транзакции,

Java Persistence 2.1. Chapter 3.2.2

Оригинал

- Вопрос 26. Как влияет операция remove на Entity объекты каждого из четырех статусов?
  Ответ

  1) Если статус Entity new, операция игнорируется, однако зависимые Entity могут поменять статус на removed, если у них есть аннотации каскадных изменений и они имели статус managed,
  2) Если статус managed, то статус меняется на removed и запись объект в базе данных будет удалена при commit'e транзакции (так же произойдут операции remove для всех каскадно зависимых объектов),
  3) Если статус removed, то операция игнорируется,
  4) Если статус detached, будет выкинут exception сразу или на этапе commit'a транзакции,

  Java Persistence 2.1. Chapter 3.2.3
  Оригинал

- Вопрос 27. Как влияет операция merge на Entity объекты каждого из четырех статусов?
  Ответ

  1) Если статус detached, то либо данные будет скопированы в существующей managed entity с тем же первичным ключом, либо создан новый managed в который скопируются данные,
  1) Если статус Entity new, то будет создана новый managed entity, в который будут скопированы данные прошлого объекта,
  2) Если статус managed, операция игнорируется, однако операция merge сработает на каскадно зависимые Entity, если их статус не managed,
  3) Если статус removed, будет выкинут exception сразу или на этапе commit'a транзакции,

  Java Persistence 2.1. Chapter 3.2.7.1
  Оригинал

- Вопрос 28. Как влияет операция refresh на Entity объекты каждого из четырех статусов?
  Ответ

  1) Если статус Entity managed, то в результате операции будут востановлены все изменения из базы данных данного Entity, так же произойдет refresh всех каскадно зависимых объектов,
  2) Если статус new, removed или detached, будет выкинут exception,

  Java Persistence 2.1. Chapter 3.2.5
  Оригинал

- Вопрос 29. Как влияет операция detach на Entity объекты каждого из четырех статусов?
  Ответ

  1) Если статус Entity managed или removed, то в результате операции статус Entity (и всех каскадно-зависимых объектов) станет detached.
  2) Если статус new или detached, то операция игнорируется,

  Java Persistence 2.1. Chapter 3.2.6
  Оригинал

Аннотации JPA

- Вопрос 30. Для чего нужна аннотация Basic?
  Ответ

  Basic — указывает на простейший тип маппинга данных на колонку таблицы базы данных. Также в параметрах аннотации можно указать fetch стратегию доступа к полю и является ли это поле обязательным или нет.
  Примеры
  Оригинал

- Вопрос 31. Для чего нужна аннотация Access?
  Ответ

  Она определяет тип доступа (access type) для класса entity, суперкласса, embeddable или отдельных атрибутов, то есть как JPA будет обращаться к атрибутам entity, как к полям класса (FIELD) или как к свойствам класса (PROPERTY), имеющие гетеры (getter) и сетеры (setter).

  Java Persistence 2.1. Chapter 11.1.1

  Оригинал

- Вопрос 32. Какой аннотациями можно перекрыть связи (override entity relationship) или атрибуты, унаследованные от суперкласса, или заданные в embeddable классе при использовании этого embeddable класса в одном из entity классов и не перекрывать в остальных?
  Ответ

  Для такого перекрывания существует четыре аннотации:
  1. AttributeOverride чтобы перекрыть поля, свойства и первичные ключи,
  2. AttributeOverrides аналогично можно перекрыть поля, свойства и первичные ключи со множественными значениями,
  3. AssociationOverride чтобы перекрывать связи (override entity relationship),
  4. AssociationOverrides чтобы перекрывать множественные связи (multiple relationship),

  Java Persistence 2.1. Chapter 11.1.2-11.1.5

  Оригинал

  Примеры (AssociationOverride)

- Вопрос 33. Какой аннотацией можно управлять кешированием JPA для данного Entity?
  Ответ

  Cacheable — позволяет включить или выключить использование кеша второго уровня (second-level cache) для данного Entity (если провайдер JPA поддерживает работу с кешированием и настройки кеша (second-level cache) стоят как ENABLE_SELECTIVE или DISABLE_SELECTIVE, см вопрос 41). Обратите внимание свойство наследуется и если не будет перекрыто у наследников, то кеширование измениться и для них тоже.
  Примеры

  Оригинал

- Вопрос 34. Какие аннотации служит для задания класса преобразования basic аттрибута Entity в другой тип при сохранении/получении данных их базы (например, работать с аттрибутом Entity boolean типа, но в базу сохранять его как число)?
  Ответ

  Convert и **Converts** — позволяют указать класс для конвертации Basic аттрибута Entity в другой тип (Converts — позволяют указать несколько классов конвертации). Классы для конвертации должны реализовать интерфейс AttributeConverter и могут быть отмечены (но это не обязательно) аннотацией **Converter**.

  Подробнее, см Javadoc 7ee.
  Примеры

- Вопрос 35. Какой аннотацией можно задать класс, методы которого должен выполнится при определенных JPA операциях над данным Enitity или Mapped Superclass (такие как удаление, изменение данных и т.п.)?
  Ответ

  Аннотация EntityListeners позволяет задать класс Listener, который будет содержать методы обработки событий (callback methods) определенных Entity или Mapped Superclass.

  Подробнее, см Javadoc 7ee.

Примеры

- Вопрос 36. Для чего нужны callback методы в JPA? К каким сущностям применяются аннотации callback методов? Перечислите семь callback методов (или что тоже самое аннотаций callback методов)
  Ответ

  Callback методы служат для вызова при определенных событиях Entity (то есть добавить обработку например удаления Entity методами JPA), могут быть добавлены к entity классу, к mapped superclass, или к callback listener классу, заданному аннотацией EntityListeners (см предыдущий вопрос). Существует семь callback методов (и аннотаций с теми же именами):

  1) PrePersist
  2) PostPersist
  3) PreRemove
  4) PostRemove
  5) PreUpdate
  6) PostUpdate
  7) PostLoad

  Подробнее, см Javadoc 7ee или спецификация JPA2.1 глава 3.5.2

  Примеры

- Вопрос 37. Какие аннотации служать для установки порядка выдачи элементов коллекций Entity?
  Ответ

  Для этого служит аннотация OrderBy и OrderColumn

  Подробнее, см Javadoc 7ee или спецификация JPA2.1 глава 11.1.42

  Примеры

- Вопрос 38. Какой аннотацей можно исключить поли и свойства Entity из маппинга (property or field is not persistent)?
  Ответ

  Для этого служит аннотация Transient

  Подробнее, см Javadoc 7ee или спецификация JPA2.1 глава 11.1.52

  Пример

Сложные вопросы JPA

- Вопрос 39. Какие шесть видов блокировок (lock) описаны в спецификации JPA (или какие есть значения у enum LockModeType в JPA)?
  Ответ

  У JPA есть шесть видов блокировок, перечислим их в порядке увеличения надежности (от самого ненадежного и быстрого, до самого надежного и медленного):

  1) NONE — без блокировки
  2) OPTIMISTIC (или синоним READ, оставшийся от JPA 1) — оптимистическая блокировка,
  3) OPTIMISTIC_FORCE_INCREMENT (или синоним WRITE, оставшийся от JPA 1) — оптимистическая блокировка с принудительным увеличением поля версионности,
  4) PESSIMISTIC_READ — пессимистичная блокировка на чтение,
  5) PESSIMISTIC_WRITE — пессимистичная блокировка на запись (и чтение),
  6) PESSIMISTIC_FORCE_INCREMENT — пессимистичная блокировка на запись (и чтение) с принудительным увеличением поля версионности,

  Подробнее, см Javadoc 7ee и описание оптимистичных и пессимистичных блокировок баз данных.

- Вопрос 40. Какие два вида кэшей (cache) вы знаете в JPA и для чего они нужны?
  Ответ

  JPA говорит о двух видов кэшей (cache):
  1) first-level cache (кэш первого уровня) — кэширует данные одной транзакции,
  2) second-level cache (кэш второго уровня) — кэширует данные дольше чем одна транзакция. Провайдер JPA может, но не обязан реализовывать работу с кэшем второго уровня. Такой вид кэша позволяет сэкономить время доступа и улучшить производительность, однако оборотной стороной является возможность получить устаревшие данные.

  Подробнее, см JPA 2.1 specification, 3.9 Caching
  Оригинал

- Вопрос 41. Какие есть варианты настройки second-level cache (кэша второго уровня) в JPA или что аналогично опишите какие значения может принимать элемент shared-cache-mode из persistence.xml?
  Ответ

  JPA говорит о пяти значениях shared-cache-mode из persistence.xml, который определяет как будет использоваться second-level cache:
  1) ALL — все Entity могут кэшироваться в кеше второго уровня,
  2) NONE — кеширование отключено для всех Entity,
  3) ENABLE_SELECTIVE — кэширование работает только для тех Entity, у которых установлена аннотация Cacheable(true) или её xml эквивалент, для всех остальных кэширование отключено,
  4) DISABLE_SELECTIVE — кэширование работает для всех Entity, за исключением тех у которых установлена аннотация Cacheable(false) или её xml эквивалент
  5) UNSPECIFIED — кеширование не определенно, каждый провайдер JPA использует свою значение по умолчанию для кэширования,

  Подробнее, см JPA 2.1 specification, 3.9 Caching
  Оригинал

  > The shared-cache-mode element has five possible values: ALL, NONE, ENABLE_SELECTIVE, DISABLE_SELECTIVE, UNSPECIFIED.

- Вопрос 42. Как можно изменить настройки fetch стратегии любых атрибутов Entity для отдельных запросов (query) или методов поиска (find), то если у Enity есть атрибут с fetchType = LAZY, но для конкретного запроса его требуется сделать EAGER или наоборот?
  Ответ

  Для этого существует EntityGraph API, используется он так: с помощью аннотации NamedEntityGraph для Entity, создаются именованные EntityGraph объекты, которые содержат список атрибутов у которых нужно поменять fetchType на EAGER, а потом данное имя указывается в hits запросов или метода find. В результате fetchType атрибутов Entity меняется, но только для этого запроса. Существует две стандартных property для указания EntityGraph в hit:
  1) javax.persistence.fetchgraph — все атрибуты перечисленные в EntityGraph меняют fetchType на EAGER, все остальные на LAZY
  2) javax.persistence.loadgraph — все атрибуты перечисленные в EntityGraph меняют fetchType на EAGER, все остальные сохраняют свой fetchType (то есть если у атрибута, не указанного в EntityGraph, fetchType был EAGER, то он и останется EAGER)

  С помощью NamedSubgraph можно также изменить fetchType вложенных объектов Entity.

  Примеры

```
// Определяем Entity и EntityGraph
@Entity
@Table(name = "order")
@Named(name = "graphOrderItems",
               attributeNodes = @NamedAttributeNode(attributeNodes = "items")
)
public class Order implements Serializable {

    ...
```

```
    @OneToMany(mappedBy = "order", fetch = FetchType.LAZY)
    private Set<Item> items = new HashSet<Item>();

    @OneToMany(mappedBy = "order", fetch = FetchType.EAGER)
    private Set<Features> features = new HashSet<Features>();

    @OneToMany(mappedBy = "order", fetch = FetchType.LAZY)
    private Set<Comment> comments = new HashSet<Comment>();
    ...

// Вызываем метод поиска с javax.persistence.fetchgraph
..
EntityGraph graph = this.em.getEntityGraph("graphOrderItems");

Map hints = new HashMap();
hints.put("javax.persistence.fetchgraph", graph);

return this.em.find(Order.class, orderId, hints); // items во время запроса будет иметь fetchType = EAGER, a features и c
omments имеют fetchType = LAZY

// Вызываем метод поиска с javax.persistence.loadgraph
..
EntityGraph graph = this.em.getEntityGraph("graphOrderItems");

Map hints = new HashMap();
hints.put("javax.persistence.loadgraph", graph);

return this.em.find(Order.class, orderId, hints); // items и features во время запроса будет иметь fetchType = EAGER, a
 comments все также имеет fetchType = LAZY
```

Подробнее, см JPA 2.1 specification, 3.7 EntityGraph

Оригинал

> An entity graph can be used with the find method or as a query hint to override or augment FetchType semantics. The standard properties javax.persistence.fetchgraph and javax.persistence.loadgraph are used to specify such graphs to queries and find operations

- Вопрос 43. Каким способом можно в коде работать с кэшем второго уровня (удалять все или определенные Entity из кеша, узнать закэшировался ли данное Entity и т.п.)?

  Ответ

  Для работы с кэшем второго уровня (second level cache) в JPA описан Cache интерфейс, содержащий большое количество методов по управлению кэшем второго уровня (second level cache), если он поддерживается провайдером JPA, конечно. Объект данного интерфейса можно получить с помощью метода getCache у EntityManagerFactory.

  Подробнее, см JPA 2.1 specification, 7.10 Cache Interface

  Оригинал

- Вопрос 44. Каким способом можно получить метаданные JPA (сведения о Entity типах, Embeddable и Managed классах и т.п.)?

  Ответ

  Для получения такой информации в JPA используется интерфейс Metamodel. Объект этого интерфейса можно получить методом getMetamodel у EntityManagerFactory или EntityManager.

  Подробнее, см JPA 2.1 specification, 5 Metamodel API

  Оригинал

- Вопрос 45. Что такое JPQL (Java Persistence query language) и чем он отличается от SQL?

Ответ

JPQL (Java Persistence query language) это язык запросов, практически такой же как SQL, однако вместо имен и колонок таблиц базы данных, он использует имена классов Entity и их атрибуты. В качестве параметров запросов так же используются типы данных атрибутов Entity, а не полей баз данных. В отличии от SQL в JPQL есть автоматический полиморфизм (см. следующий вопрос). Также в JPQL используется функции которых нет в SQL: такие как KEY (ключ Map'ы), VALUE (значение Map'ы), TREAT (для приведение суперкласса к его объекту-наследнику, downcasting), ENTRY и т.п.

Подробнее, см JPA 2.1 specification, Chapter 4 Query Language

Оригинал

- Вопрос 46. Что означает полиморфизм (polymorphism) в запросах JPQL (Java Persistence query language) и как его «выключить»?

  Ответ

  В отличии от SQL в запросах JPQL есть автоматический полиморфизм, то есть каждый запрос к Entity возвращает не только объекты этого Entity, но так же объекты всех его классов-потомков, независимо от стратегии наследования (например, запрос select * from Animal, вернет не только объекты Animal, но и объекты классов Cat и Dog, которые унаследованы от Animal). Чтобы исключить такое поведение используется функция TYPE в where условии (например select * from Animal a where TYPE(a) IN (Animal, Cat) уже не вернет объекты класса Dog).

  Подробнее, см JPA 2.1 specification, Chapter 4 Query Language

  Оригинал

- Вопрос 47. Что такое Criteria API и для чего он используется?

  Ответ

  Criteria API это тоже язык запросов, аналогичным JPQL (Java Persistence query language), однако запросы основаны на методах и объектах, то есть запросы выглядят так:

```
CriteriaBuilder cb = ...
CriteriaQuery<Customer> q = cb.createQuery(Customer.class);
Root<Customer> customer = q.from(Customer.class);
q.select(customer);
```

  Подробнее, см JPA 2.1 specification, Chapter 6 Criteria API

  Оригинал

Отличия Hibernate 5.0 от JPA 2.1 или JPA 2.0 от JPA 2.1

- Вопрос 48. В чем разница в требованиях к Entity в Hibernate, от требований к Entity, указанных в спецификации JPA (см. вопрос 10)?

  Ответ

  1) Конструктор без аргументов не обязан быть public или protected, рекомендуется чтобы он был хотя бы package видимости, однако это только рекомендация, если настройки безопасности Java позволяют доступ к приватным полям, то он может быть приватным,
  2) JPA категорически требует не использовать final классы, Hibernate лишь рекомендует не использовать такие классы чтобы он мог создавать прокси для ленивой загрузки, однако позволяет либо выключить прокси @Proxy(lazy=false), либо использовать в качестве прокси интерфейс, содержащий все методы маппинга для данного класса (аннотацией @Proxy(proxyClass=интерфейс.class))

  Подробнее, см hibernate 5.0 manual

- Вопрос 49. Какая уникальная стратегия наследования есть в Hibernate, но нет в спецификации JPA?

  Ответ

В отличии JPA в Hibernate есть уникальная стратегия наследования, которая называется implicit polymorphism.

Подробнее, см hibernate 5.0 manual
▸ Оригинал

- Вопрос 50. Какие основные новые возможности появились в спецификации JPA 2.1 по сравнению с JPA 2.0 (перечислите хотя бы пять-шесть новых возможностей)?
  ▾ Ответ

В спецификации JPA 2.1 появились:
1) Entity Graphs — механизм динамического изменения fetchType для каждого запроса,
2) Converters — механизм определения конвертеров для задания функций конвертации атрибутов Entity в поля базы данных,
3) DDL генерация — автоматическая генерация таблиц, индексов и схем,
4) Stored Procedures — механизм вызова хранимых процедур из JPA,
5) Criteria Update/Delete — механизм вызова bulk updates или deletes, используя Criteria API,
6) Unsynchronized persistence contexts — появление возможности указать SynchronizationType,
7) Новые возможности в JPQL/Criteria API: арифметические подзапросы, generic database functions, join ON clause, функция TREAT,
8) Динамическое создание именованных запросов (named queries)

Подробнее о изменении интерфейсов и API в JPA 2.1:
1) Интерфейс EntityManager получил новые методы createStoredProcedureQuery, isJoinedToTransaction и createQuery(CriteriaUpdate или CriteriaDelete)
2) Абстрактный класс AbstractQuery стал наследоваться от класса CommonAbstractCriteria, появились новые интерфейсы CriteriaUpdate, CriteriaDelete унаследованные CommonAbstractCriteria,
3) PersistenceProvider получил новые функции generateSchema позволяющие генерить схемы,
4) EntityManagerFactory получил методы addNamedQuery, unwrap, addNamedEntityGraph, createEntityManager (с указанием SynchronizationType)
5) Появился новый enum SynchronizationType, Entity Graphs, StoredProcedureQuery и AttributeConverter интерфейсы,

Подробнее, см Java Persistence 2.1 и Java Persistence 2.0

P.S. Если нашли техническую ошибку, ошибку в переводе или хотите что-то добавить (в том числе новые интересные вопросы), буду благодарен если напишите их в комментарии этой статьи или в личку.

P.P.S. Так же советую посмотреть мой opensource проект [useful-java-links](https://github.com/Vedenin/useful-java-links/tree/master/link-rus) — возможно, наиболее полная коллекция полезных Java библиотек, фреймворков и русскоязычного обучающего видео. Так же есть аналогичная [английская версия](https://github.com/Vedenin/useful-java-links/) этого проекта и начинаю opensource подпроект [Hello world](https://github.com/Vedenin/useful-java-links/tree/master/helloworlds) по подготовке коллекции простых примеров для разных Java библиотек в одном maven проекте (буду благодарен за любую помощь).

▾ Общее оглавление 'Шпаргалок'

Сколько вы набрали баллов (дали правильных ответов)?

- ○ 45-50
- ○ 40-44
- ○ 35-39

- 30-34
- 20-29
- 10-19
- меньше 10
- не считал

Проголосовало 244 человека. Воздержалось 257 человек.

Насколько корректными вам показались вопросы?

- все вопросы правильные и корректные
- есть небольшие незначительные ошибки
- слишком много вопросов на зубрежку
- слишком сложные вопросы
- большое количество технических ошибок
- много ненужных вопросов
- все плохо

Проголосовало 139 человек. Воздержалось 279 человек.

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

java, jpa, jdo, ejb, ejb3, orm, hibernate

+27 123k 844

Веденин Вячеслав @vedenin1980 карма 48,7 рейтинг 31,0
Java developer

Похожие публикации

+25 JPA: Хранение перечислений в базе данных
14,6k 100 23

+19 Java EE 6. Обзор JPA 2.0, часть 2: Коллекции
9,8k 30 2

+25 Java EE 6. Обзор JPA 2.0, часть 1: Введение
11,8k 32 10

Комментарии (13)

Optik 3 сентября 2015 в 14:08 +6

Зачем это в блоге scala?

> **vedenin1980** 3 сентября 2015 в 16:38 *(комментарий был изменён)* –2
>
> В основном потому что scala тоже может использовать JPA библиотеки, такие как hibernete, не буду спорить что это лучший выбор, но иногда (например, когда в одном проекте смешивается код и Java и scala) приходится с ними работать и scala разработчикам.
>
> > **vedenin1980** 3 сентября 2015 в 17:40 *(комментарий был изменён)* +4
> >
> > Ок, раз все так дружено проголосовали против, убрал блог scala.

Revkov 3 сентября 2015 в 16:56 *(комментарий был изменён)* +2

Мне кажется или до меня хочет что-то донести человек, который пишет Hibernet, Hibernete, но никак не Hibernate?

> **vedenin1980** 3 сентября 2015 в 17:38 +1
>
> Кажется, я ни до кого ничего «доносить» не собирался :), будет информация кому-то полезна — хорошо, не будет — я как минимум обновил и структурировал свои знания.

sentyaev 3 сентября 2015 в 22:02 +1

Вопрос 10, пункт 7.
Класс не передается по значению, передаются сериализованные данные.

> **vedenin1980** 3 сентября 2015 в 22:30 0
>
> Да спасибо, поправил. Хотя честно говоря тогда получается масло масленое в этом условии JPA спецификации.
>
> > **sentyaev** 4 сентября 2015 в 02:55 +1
> >
> > Перечитал еще раз.
> > If an entity instance is to be passed by value as a detached object (e.g., through a remote interface), the entity class must implement the Serializable interface.
> >
> > Ваша первая версия была лучше, чем сейчас.
> > Да, действительно объект передается как-бы по значению, но на самом же деле это довольно сильное упрощение.
> > Видимо в данном контексте этого достаточно.

EaE 4 сентября 2015 в 07:47 *(комментарий был изменён)* 0

А вот у меня есть вопрос.
Как заставить хибернейт при включенном auto update не пытаться создавать уже существующие индексы в базе, объявленные через @ Index над полями сущности?

Borz 4 сентября 2015 в 12:23 0

я присваивал своим индексам те же имена, что присваивает им Hibernate

SOLON7 4 сентября 2015 в 09:53 +2

Спасибо очень позновательнно!!! Снова почувствовал себя институтским ботаником))

PSerber 8 октября 2015 в 13:58 +2

Спасибо, а про основы **Spring Spring MVC,** можно будеть ?!!!

vedenin1980 8 октября 2015 в 21:16 +1

Да, сначала планирую дописать статью про разные виды коллекций (не только стандартные), потом планирую перейти к DI и Spring.

Интересные публикации

- Гейзенбаг: Версия 1.0 0
- Компания Google представила набор тестов Wycheproof 0
- Улучшение производительности PHP 7 3
- Защищенный Dell 4
- Преобразование формы представления данных при помощи Excel+PowerQuery 0