Ответы на вопросы на собеседование Object relational mapping (ORM), Hibernate (часть 1).

Vasyl K 19:00:00 1 Комментарий

- **ЧТО ТАКОЕ HIBERNATE?**

Это фреймворк для объектно-реляционного отображения сущностей в традиционные реляционные базу данных.

Основные возможности фреймворка:

- Автоматическая генерация и обновление таблиц в базах данных;
- Поскольку основные запросы к базе данных (сохранение, обновление, удаление и поиск) представлены как методы фрейморка, то значительно сокращается код, который пишется разработчиком;
- Обеспечивает использование SQL подобного языка (HQL - hibernate query language). Запросы HQL могут быть записаны рядом объектами данных (POJO классы подготовленные для работы с базой данных).

- **ЧТО ТАКОЕ ORM?**

ORM (англ. Object-relational mapping, рус. Объектно-реляционное отображение) - технология программирования,

которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая "виртуальную объектную базу данных".

- **КАКИЕ ПРЕИМУЩСТВА ОТ ИСПОЛЬЗОВАНИЯ HIBERNATE?**

Некоторые из них:

- Устраняет множество повторяющегося кода, который постоянно преследует разработчика при работе с JDBC. Скрывает от разработчика множество кода, необходимого для управления ресурсами и позволяет сосредоточиться на бизнес логике.
- Поддерживает XML так же как и JPA аннотации, что позволяет сделать реализацию кода независимой.
- Предоставляет собственный мощный язык запросов (HQL), который похож на SQL. Стоит отметить, что HQL полностью объектно-ориентирован и понимает такие принципы, как наследование, полиморфизм и ассоциации (связи).

- Hibernate легко интегрируется с другими Java EE фреймворками, например, Spring Framework поддерживает встроенную интеграцию с Hibernate.
- Поддерживает ленивую инициализацию используя proxy объекты и выполняет запросы к базе данных только по необходимости.
- Поддерживает разные уровни cache, а следовательно может повысить производительность.
- Важно, что Hibernate может использовать чистый SQL, а значит поддерживает возможность оптимизации запросов и работы с любым сторонним вендором БД.
- Hibernate – open source проект. Благодаря этому доступны тысячи открытых статей, примеров, а так же документации по использованию фреймворка.

- **КАК HIBERNATE ПОМОГАЕТ В ПРОГРАММИРОВАНИИ?**

Hibernate реализует ряд фичь которые значительно упрощают работу разработчика.

- Одной из таких фичь является то, что hibernate позволяет разработчику избежать написания большинства SQL запросов (они уже реализованы , вам надо просто использовать методы которые предоставляет фреймворк).
- Под бортом у Hibernate есть куча полезных инструментов которые значительно ускоряют работу приложения, самыми примечательными из них являются двухуровневое кэширования и тонкие настройки lazy и fetch изъятия.
- Сам генерирует таблицы в базу данных

- **КАКИЕ ПРЕИМУЩЕСТВА HIBERNATE НАД JDBC?**

Hibernate имеет ряд преимуществ перед JDBC API:

- Hibernate удаляет множество повторяющегося кода из JDBC API, а следовательно его легче читать, писать и поддерживать.
- Hibernate поддерживает наследование, ассоциации и коллекции, что не доступно в JDBC API.
- Hibernate неявно использует управление транзакциями. Большинство запросов нельзя выполнить вне транзакции. При использовании JDBC API для управления транзакциями нужно явно использовать commit и rollback.
- JDBC API throws SQLException, которое относится к проверяемым исключениям, а значит необходимо постоянно писать множество блоков try-catch. В большинстве случаев это не нужно для каждого вызова JDBC и используется для управления транзакциями. Hibernate оборачивает исключения JDBC через непроверяемые JDBCException или HibernateException, а значит нет необходимости проверять их в коде каждый раз. Встроенная поддержка управления транзакциями в Hibernate убирает блоки try-catch.
- Hibernate Query Language (HQL) более объектно ориентированный и близкий к Java язык программирования, чем SQL в JDBC.
- Hibernate поддерживает кэширование, а запросы JDBC – нет, что может понизить производительность.
- Hibernate предоставляет возможность управления БД (например создания таблиц), а в JDBC можно работать только с существующими таблицами в базе данных.
- Конфигурация Hibernate позволяет использовать JDBC вроде соединения по типу JNDI DataSource для пула соединений. Это важная фича для энтерпрайз приложений, которая полностью отсутствует в JDBC API.
- Hibernate поддерживает аннотации JPA, а значит код является переносимым на другие ORM фреймворки, реализующие стандарт, в то время как код JDBC сильно привязан к приложению.

- **ЧТО ТАКОЕ КОНФИГУРАЦИОННЫЙ ФАЙЛ HIBERNATE?**

Файл конфигурации Hibernate содержит в себе данные о базе данных и необходим для инициализации SessionFactory. В .xml файле необходимо указать вендора базы данных или JNDI ресурсы, а так же информацию об используемом диалекте, что поможет hibernate выбрать режим работы с конкретной базой данных.

- **СПОСОБЫ КОНФИГУРАЦИИ РАБОТЫ С HIBERNATE.**

Существует четыре способа конфигурации работы с Hibernate :
- используя аннотации;
- hibernate.cfg.xml;
- hibernate.properties;
- persistence.xml.

Самый частый способ конфигурации: через аннотации и файл persistence.xml, что касается файлов hibernate.properties и hibernate.cfg.xml, то hibernate.cfg.xml главнее (если в приложение есть оба файла, то принимаются настройки из файла hibernate.cfg.xml). Конфигурация аннотациями, хоть и удобна, но не всегда возможна, к примеру, если для разных баз данных или для разных ситуаций вы хотите иметь разные конфигурацию сущностей, то следует использовать xml файлы конфигураций.

- **ЧТО ТАКОЕ HIBERNATE MAPPING FILE?**

Файл отображения (mapping file) используется для связи entity бинов и колонок в таблице базы данных. В случаях, когда не используются аннотации JPA, файл отображения .xml может быть полезен (например при использовании сторонних библиотек).

- **ЧТО ТАКОЕ ПЕРЕХОДНЫЕ ОБЪЕКТЫ (TRANSIENT OBJECTS)?**

Экземпляры долгоживущих классов, которые в настоящее время не связаны с Сессией. Они, возможно, были инициализированы в приложении и еще не сохранены, или же они были инициализированы закрытой Сессией.

- **ЧТО ТАКОЕ ПОСТОЯННЫЕ ОБЪЕКТЫ (PERSISTENT OBJECTS)?**

Короткоживущие, однопоточные объекты, содержащие постоянное состояние и бизнес-функции. Это могут быть простые Java Beans/POJOs (Plain Old Java Object). Они связаны только с одной Сессией. После того, как Сессия закрыта, они будут отделены и свободны для использования в любом протоколе прикладного уровня (например, в качестве объектов передачи данных в и из представления).

- **ЧТО ТАКОЕ TRANSACTIONFACTORY?**

Фабрика для экземпляров Transaction. Интерфейс не открыт для приложения, но может быть расширен или реализован разработчиком.

- **ЧТО ТАКОЕ CONNECTIONPROVIDER?**

Фабрика и пул JDBC соединений. Интерфейс абстрагирует приложение от основного источника данных или диспетчера драйверов. Он не открыт для приложения, но может быть расширен или реализован разработчиком.

- **ЧТО ТАКОЕ ТРАНСАКЦИЯ (TRANSACTION)?**

Однопоточный, короткоживущий объект, используемый приложением для указания atomic переменных работы. Он абстрагирует приложение от основных JDBC, JTA или CORBA трансакций. Сессия может охватывать несколько Трансакций в некоторых случаях. Тем не менее, разграничение транзакций, также используемое в основах API или Transaction, всегда обязательно.

- **КАКИЕ СУЩЕСТВУЮТ СТРАТЕГИИ ЗАГРУЗКИ ОБЪЕКТОВ В HIBERNATE?**

Существуют следующие типа fetch'a:

- Join fetching: hibernate получает ассоциированные объекты и коллекции одним SELECT используя OUTER JOIN
- Select fetching: использует уточняющий SELECT чтобы получить ассоциированные объекты и коллекции. Если вы не установите lazy fetching определив lazy="false", уточняющий SELECT будет выполнен только когда вы запрашиваете доступ к ассоциированным объектам
- Subselect fetching: поведение такое же, как у предыдущего типа, за тем исключением, что будут загружены ассоциации для все других коллекций, "родительским" для которых является сущность, которую вы загрузили первым SELECT'ом.
- Batch fetching: оптимизированная стратегия вида select fetching. Получает группу сущностей или коллекций в одном SELECT'е.

- **КАКИЕ БЫВАЮТ ID GENERATOR КЛАССЫ В HIBERNATE?**

increment – генерирует идентификатор типа long, short или int, которые будет уникальным только в том случае, если другой процесс не добавляет запись в эту же таблицу в это же время.
identity – генерирует идентификатор типа long, short или int. Поддерживается в DB2, MySQL, MS SQL Server, Sybase и HypersonicSQL.
sequence – использует последовательности в DB2, PostgreSQL, Oracle, SAP DB, McKoi или генератор Interbase. Возвращает идентификатор типа long, short или int.
hilo – использует алгоритм hi/lo для генерации идентификаторов типа long, short или int. Алгоритм гарантирует генерацию идентификаторов, которые уникальны только в данной базе данных.
seqhilo – использует алгоритм hi/lo для генерации идентификаторов типа long, short или int учитывая последовательность базы данных.

uuid – использует для генерации идентификатора алгоритм 128-bit UUID. Идентификатор будет уникальным в пределах сети. UUID представляется строкой из 32 чисел.
guid – использует сгенерированую БД строку GUID в MS SQL Server и MySQL.
native – использует identity, sequence или hilo в завимисимости от типа БД, с которой работает приложение
assigned – позволяет приложению устанавливать идентификатор объекту, до вызова метода save(). Используется по умолчанию, если тег <generator> не указан.
select – получает первичный ключ, присвоенный триггером БД
foreign – использует идентификатор другого, связанного с данным объекта. Используется в <one-to-one> ассоциации первичных ключей.
sequence-identity – специализированный генератор идентификатора.

- **КАКИЕ КЛЮЧЕВЫЕ ИНТЕРФЕЙСЫ ИСПОЛЬЗУЕТ HIBERNATE?**

Существует пять ключевых интерфейсов которые используются в каждом приложении связанном с Hibernate:

- Session interface;
- SessionFactory interface;
- Configuration interface;
- Transaction interface;
- Query and Criteria interfaces.

- **НАЗОВИТЕ НЕКОТОРЫЕ ВАЖНЫЕ АННОТАЦИИ, ИСПОЛЬЗУЕМЫЕ ДЛЯ ОТОБРАЖЕНИЯ В HIBERNATE.**

Hibernate поддерживает как аннотации из JPA, так и свои собственные, которые находятся в пакете org.hibernate.annotations. Наиболее важные аннотации JPA и Hibernate:

- javax.persistence.Entity: используется для указания класса как entity bean.
- javax.persistence.Table: используется для определения имени таблицы из БД, которая будет отображаться на entity bean.
- javax.persistence.Access: определяет тип доступа, поле или свойство. Поле – является значением по умолчанию и если нужно, чтобы hibernate использовать методы getter/setter, то их необходимо задать для нужного свойства.
- javax.persistence.Id: определяет primary key в entity bean.
- javax.persistence.EmbeddedId: используется для определения составного ключа в бине.
- javax.persistence.Column: определяет имя колонки из таблицы в базе данных.
- javax.persistence.GeneratedValue: задает стратегию создания основных ключей. Используется в сочетании с javax.persistence.GenerationType enum.
- javax.persistence.OneToOne: задает связь один-к-одному между двумя сущностными бинами. Соответственно есть другие аннотации OneToMany, ManyToOne и ManyToMany.
- org.hibernate.annotations.Cascade: определяет каскадную связь между двумя entity бинами. Используется в связке с org.hibernate.annotations.CascadeType.
- javax.persistence.PrimaryKeyJoinColumn: определяет внешний ключ для свойства. Используется вместе с org.hibernate.annotations.GenericGenerator и

org.hibernate.annotations.Parameter.

- **КАКАЯ РОЛЬ ИНТЕРФЕЙСА SESSION В HIBERNATE?**

Session – это основной интерфейс, который отвечает за связь с базой данных. Так же, он помогает создавать объекты запросов для получение персистентных объектов. (персистентный объект – объект который уже находится в базе данных; объект запроса – объект который получается когда мы получаем результат запроса в базу данных, именно с ним работает приложение). Обьект Session можно получить из SessionFactory :

```
1 | Session session = sessionFactory.openSession();
```

Роль интерфейса Session:

- является оберткой для jdbc подключения к базе данных;
- является фабрикой для транзакций (согласно официальной документации transaction – allows the application to define units of work, что , по сути, означает что транзакция определяет границы операций связанных с базой данных).
- является хранителем обязательного кэша первого уровня.

- **КАКАЯ РОЛЬ ИНТЕРФЕЙСА SESSIONFACTORY В HIBERNATE?**

Именно из объекта SessionFactory мы получаем объекты типа Session. На все приложение существует только одна SessionFactory и она инициализируеться вместе со стартом приложения. SessionFactory кэширует мета-дату и SQL запросы которые часто используются приложением во время работы. Так же оно кэширует информацию которая была получена в одной из транзакций и может быть использована и в других транзакциях.
Обьект SessionFactory можно получить следующим обращением:

```
1 | SessionFactory sessionFactory = configuration.buildSessionFactory();
```

- **ЯВЛЯЕТСЯ ЛИ HIBERNATE SESSIONFACTORY ПОТОКОБЕЗОСПАНСЫМ?**

Т.к. объект SessionFactory immutable (неизменяемый), то да, он потокобезопасный. Множество потоков может обращаться к одному объекту одновременно.

- **В ЧЕМ РАЗНИЦА МЕЖДУ OPENSESSION И GETCURRENTSESSION?**

Hibernate SessionFactory getCurrentSession() возвращает сессию, связанную с контекстом. Но для того, чтобы это работало, нам нужно настроить его в конфигурационном файле hibernate. Так как этот объект session связан с контекстом hibernate, то отпадает необходимость к его закрытию. Объект session закрывается вместе с закрытием SessionFactory.

```
1 | <property name="hibernate.current_session_context_class">thread</property>
```

Метод Hibernate SessionFactory openSession() всегда создает новую сессию. Мы должны обязательно контролировать закрытие объекта сеанса по завершению всех

операций с базой данных. Для многопоточной среды необходимо создавать новый объект session для каждого запроса.
Существует еще один метод openStatelessSession(), который возвращает session без поддержки состояния. Такой объект не реализует первый уровень кэширования и не взаимодействует с вторым уровнем. Сюда же можно отнести игнорирование коллекций и некоторых обработчиков событий. Такие объекты могут быть полезны при загрузке больших объемов данных без удержания большого кол-ва информации в кэше.

- **КАКИЕ ТИПЫ КОЛЛЕКЦИЙ ПРЕДСТАВЛЕНЫ В HIBERNATE?**

Bag, Set, List, Map, Array.

- **КАКИЕ ТИПЫ МЕНЕДЖМЕНТА ТРАНЗАКЦИЙ ПОДДЕРЖИВАЮТСЯ В HIBERNATE?**

Hibernate взаимодействует с БД через JDBC-соединение. Таким образом он поддерживает управляемые и не управляемые транзакции.
Неуправляемые транзакции в web-контейнере:

```
<bean id="transactionManager" class="org.springframework.orm.hibernate.HibernateTransactionManager">
    <property name="sessionFactory">
        <ref local="sessionFactory"/>
    </property>
</bean>
```

Управляемые транзакции на сервере приложений, использующий JTA:

```
<bean id="transactionManager" class="org.springframework.transaction.jtaTransactionManager.">
    <property name="sessionFactory">
        <ref local="sessionFactory" />
    </property>
</bean>
```

- **ЧТО СОБОЙ ЯВЛЯЕТ КОЛЛЕКЦИЯ ТИПА BAG И ЗАЧЕМ ОНА ИСПОЛЬЗУЕТСЯ?**

Своей реализации тип коллекции Bag очень напоминает Set, разница состоит в том, что Bag может хранить повторяющиеся значения. Bag хранит непроиндексированный список элементов. Большинство таблиц в базе данных имеют индексы отображающие положение элемента данных один относительно другого, данные индексы имеют представление в таблице в виде отдельной колонки. При объектно-реляционном маппинге, значения колонки индексов мапится на индекс в Array, на индекс в List или на key в Map. Если вам надо получить коллекцию объектов не содержащих данные индексы, то вы можете воспользоваться коллекциями типа Bag или Set (коллекции содержат данные в неотсортированном виде, но могут быть отсортированы согласно запросу).

- **КАКИЕ ТИПЫ КЭША ИСПОЛЬЗУЮТСЯ В HIBERNATE?**

Hibernate использует 2 типа кэша: кэш первого уровня и кэш второго уровня.
Кэш первого уровня ассоциирован с объектом сесии, в то время, как кэш второго уровня ассоциирован с объектом фабрики сессий. По-умолчанию Hibernate

использует кэш первого уровня для каждой операции в транзакции. В первую очередь кэш используется чтобы уменьшить количество SQL-запросов. Например если объект модифицировался несколько раз в одной и той же транзакции, то Hibernate сгенерирует только один UPDATE.
Чтобы уменьшить трафик с БД, Hibernate использует кэш второго уровня, который является общим для всего приложения, а не только для данного конкретного пользователя. Таким образом если результат запроса находится в кэше, мы потенциально уменьшаем количество транзакций к БД.
EHCache – это быстрый и простой кэш. Он поддерживает read-only и read/write кэширование, а так же кэширование в память и на диск. Но не поддерживает кластеризацию.
OSCache – это другая opensource реализация кэша. Помимо всего, что поддерживает EHCache, эта реализация так же поддерживает кластеризацию через JavaGroups или JMS.
SwarmCache – это просто cluster-based решение, базирующееся на JavaGroups. Поддерживает read-only и нестрогое read/write кэширование. Этот тип кэширование полезен, когда количество операций чтения из БД превышает количество операций записи.
JBoss TreeCache – предоставляет полноценный кэш транзакции.

- **КАКИЕ СУЩЕСТВУЮТ ТИПЫ СТРАТЕГИЙ КЭША?**

Read-only: эта стратегия используется когда данные вычитываются, но никогда не обновляется. Самая простая и производительная стратегия
Read/write: может быть использована, когда данные должны обновляться.
Нестрогий read/write: эта стратегия не гарантирует, что две транзакции не модифицируют одни и те же данные синхронно.
Transactional: полноценное кэширование транзакций. Доступно только в JTA окружении.

- **ЧТО ВЫ ЗНАЕТЕ О КЭШИРОВАНИЕ В HIBERNATE? ОБЪЯСНИТЕ ПОНЯТИЕ КЭШ ПЕРВОГО УРОВНЯ В HIBERNATE?**

Hibernate использует кэширование, чтобы сделать наше приложение быстрее. Кэш Hibernate может быть очень полезным в получении высокой производительности приложения при правильном использовании. Идея кэширования заключается в сокращении количества запросов к базе данных.
Кэш первого уровня Hibernate связан с объектом Session. Кэш первого уровня у Hibernate включен по умолчанию и не существует никакого способа, чтобы его отключить. Однако Hibernate предоставляет методы, с помощью которых мы можем удалить выбранные объекты из кэша или полностью очистить кэш.
Любой объект закэшированный в session не будет виден другим объектам session. После закрытия объекта сессии все кэшированные объекты будут потеряны.