|  |  |
| --- | --- |
| **Вопрос** | **Ответ** |
| Статьи наизусть | <https://javarush.ru/groups/posts/1502-voprosih-na-sobesedovanie-hibernate>  <https://habr.com/ru/post/350682/>  <http://j4sq.blogspot.com/2012/01/java-spring-hibernate.html>  <http://java-online.ru/hibernate-faq.xhtml>  <https://habr.com/ru/post/416851/>  <https://proselyte.net/tutorials/spring-tutorial-full-version/transatcion-management/>  <https://habr.com/ru/post/336816/> |
| Hashmap | HashMap содержит массив Node и Node может представлять класс, содержащий следующие объекты:   1. int — хэш 2. K — ключ 3. V — значение 4. Node — следующий элемент   Хэширование  Хэширование -это процесс преобразования объекта в целочисленную форму, выполняется с помощью метода hashCode(). Очень важно правильно реализовать метод hashCode() для обеспечения лучшей производительности класса HashMap.  Метод hashCode()  Метод hashCode() используется для получения хэш кода объекта. Метод hashCode() класса Object возвращает ссылку памяти объекта в целочисленной форме ([идентификационный хеш (identity hash code)](https://habr.com/company/mailru/blog/321306/)). Сигнатура метода public native hashCode(). Это говорит о том, что метод реализован как нативный, поскольку в java нет какого -то метода позволяющего получить ссылку на объект. Допускается определять собственную реализацию метода hashCode(). В классе HashMap метод hashCode() используется для вычисления корзины (bucket) и следовательно вычисления индекса.  Метод equals()  Метод equals используется для проверки двух объектов на равенство. Метод реализованн в классе Object. Вы можете переопределить его в своем собственном классе. В классе HashMap метод equals() используется для проверки равенства ключей. В случае, если ключи равны, метод equals() возвращает true, иначе false.  Корзины (Buckets)  Bucket -это единственный элемент массива HashMap. Он используется для хранения узлов (Nodes). Два или более узла могут иметь один и тот -же bucket. В этом случае для связи узлов используется структура данных [связанный список](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA). Bucket -ы различаются по ёмкости (свойство capacity). Отношение между bucket и capacity выглядит следующим образом:  capacity = number of buckets \* load factor  Один bucket может иметь более, чем один узел, это зависит от реализации метода hashCode(). Чем лучше реализованн ваш метод hashCode(), тем лучше будут использоваться ваши bucket -ы.   * ***вставка пар Ключ — Значение:*** добавить одну пару ключ — значение в конец HashMap   map.put(new Key("vishal"), 20);    Шаги:   1. Вычислить значение ключа {"vishal"}. Оно будет сгенерированно, как 118. 2. Вычислить индекс с помощью метода index, который будет равен 6. 3. Создать объект node. 4. Поместить объект в позицию с индексом 6, если место свободно.   Как мы уже знаем в случае возникновения коллизий объект node сохраняется в структуре данных "связанный список" и метод equals() используется для сравнения ключей. Это сравнения для поиска верного ключа в связанном списке -линейная операция и в худшем случае сложность равнa *O(n)*.    Для исправления этой проблемы в Java 8 после достижения определенного порога вместо связанных списков используются сбалансированные деревья. Это означает, что HashMap в начале сохраняет объекты в связанном списке, но после того, как колличество элементов в хэше достигает определенного порога происходит переход к сбалансированным деревьям. Что улучшает производительность в худшем случае с O(n) до O(log n).  Важный момент   1. Сложность операций get() и put() практически константна до тех пор, пока не будет проведенно повторное хэширование. 2. В случае коллизий, если индексы двух и более объектов node одинаковые, объекты node соединяются с помощью связанного списка, т.е. ссылка на второй объект node хранится в первом, на третий во втором и т.д. 3. Если данный ключ уже существует в HashMap, значение перезаписывается. 4. Хэш код null равен 0. 5. Когда объект получается по ключу происходят переходы по связанному списку до тех пор, пока объект не будет найден или ссылка на следующий объект не будет равна null.   почему в Java 8 эта функция изменилась следующим образом?  static final int hash(Object key) {  int h;  return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);  }  Тут и выползает интересная особенность. **На то, в какой бакет попадёт новая запись, влияют только младшие биты хеша**. При реальных условиях количество бакетов обычно невысокое (к примеру 8192 вмещается в 14 битов) . А хеш имеет тип int, т.е. аж **32 бита!!!**.  **Поэтому и придумали различными манипуляциями подмешивать старшие биты хэша в младшие, чтобы улучшить распределение по бакетам и, как следствие, производительность.** |
| App Server vs Web Server. | 7. Что такое web server? Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.  Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер (см.: Сервер (аппаратное обеспечение)), на котором это программное обеспечение работает. 8. Что такое web приложение? Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. 9. Что такое application server? Сервер приложений (англ. application server) — это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (Интерфейс прикладного программирования), определённый самой платформой.  Для веб-приложений основная задача компонентов сервера — обеспечивать создание динамических страниц. Однако современные серверы приложений включают в себя и поддержку кластеризации, повышенную отказоустойчивость, балансировку нагрузки, позволяя таким образом разработчикам сфокусироваться только на реализации бизнес-логики. 10. Чем отличаются web server и application server? Сервер приложений (англ. application server) — сервер, исполняющий некоторые прикладные программы. Сервер-приложений — объект, который обрабатывает запросы, связанные с приложениями, точнее для выполнения прикладных процессов (выборка данных, поиск данных, работа с терминалами). По идее эта технология изначально вообще не была связана с Web’om, однако, сейчас чаще говорят сервер web приложений. Практически используется для работы с базами данных.  Веб-сервер — это сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы. Web-сервер — объект, который обрабатывает запросы, в частности http-запросы. Браузер в данном случае клиент, который делает запросы (POST, GET). |
| How client-server works? | Вам нужно серверное приложение, которое будет предоставлять удалённым клиентам доступ к БД, но не напрямую, а через себя путём общения с клиентами по http с помощью REST (проще) или SOAP. Т.е. клиенты отправляют запросы и получают от сервера ответы. Сервер располагается (задеплоен) на удалённой машине в каком-нибудь облаке или куда вы его задеплоите. Там же рядышком будет БД, про которую будет знать только ваше серверное приложение, но не ваши клиенты. Клиенты знают только про приложение, вернее куда стучать, что отправлять и что ожидать в ответ. |
| Request/Session, why do we need it? |  |
| Session object |  |
| REST vs SOAP. | 3. Что такое SOAP? SOAP (от англ. Simple Object Access Protocol — простой протокол доступа к объектам; вплоть до спецификации 1.2) — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур. Официальная спецификация последней версии 1.2 протокола никак не расшифровывает название SOAP. SOAP является расширением протокола XML-RPC.  SOAP может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP, HTTPS и др. Однако его взаимодействие с каждым из этих протоколов имеет свои особенности, которые должны быть определены отдельно. Чаще всего SOAP используется поверх HTTP. 4. Что такое REST? REST (сокр. от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. REST является альтернативой RPC.  В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют REST-запрос), а необходимые данные передаются в качестве параметров запроса. Для веб-сервисов, построенных с учётом REST, то есть не нарушающих накладываемых им ограничений, применяют термин «RESTful». 5. В чем разница между REST и SOAP веб сервисами?  * REST поддерживает различные форматы: text, JSON, XML; SOAP — только XML, * REST работает только по HTTP(S), а SOAP может работать с различными протоколами, * REST может работать с ресурсами. Каждый URL это представление какого-либо ресурса. SOAP работает с операциями, которые реализуют какую-либо бизнес логику с помощью нескольких интерфейсов, * SOAP на основе чтения не может быть помещена в кэш, а REST в этом случае может быть закэширован, * SOAP поддерживает SSL и WS-security, в то время как REST — только SSL, * SOAP поддерживает ACID (Atomicity, Consistency, Isolation, Durability). REST поддерживает транзакции, но ни один из ACID не совместим с двух фазовым коммитом.  6. Как бы вы решили какой из REST или SOAP веб сервисов использовать? REST против SOAP можно перефразировать как «Простота против Стандарта». В случае REST (простота) у вас будет скорость, расширяемость и поддержка многих форматов. В случае с SOAP у вас будет больше возможностей по безопасности (WS-security) и транзакционная безопасность (ACID). |
| WSDL | WSDL (англ. Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML. |
| Hibernate, + and -. | **Транзакции**  Как известно hibernate позволяет обновлять сущности только внутри транзакции.   Приведу несколько фактов:   * Любой statement выполняется в БД внутри транзакции. Если даже мы ее явно не открыли. (auto-commit mode). * Как правило мы не ограничиваемся одним запросом к БД. Например: для получения первых 10 записей вы вероятно захотите вернуть количество записей всего. А это уже почти всегда 2 запроса. * Если мы говорим про spring data, то методы репозитория транзакционны [по-умолчанию](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/" \l "transactions), при этом методы чтения — read-only. * Спринговая аннотация @Transactional(readOnly=true) также влияет на FlushMode, точнее Spring переводит его в статус MANUAL, тем самым хибернейт не будет выполнять dirty-checking. * Синтетические тесты с одним-двумя запросами к БД покажут, что auto-commit работает быстрее. Но в боевом режиме это может быть не так.   Если в двух словах: хорошей практикой является любое общение с БД выполнять в транзакции. |
| Transaction. | **Первый вопрос**. Какая связь существует между уровнем изоляции транзакций, параметрами JPA Lock.READ / Lock.WRITE и версиями @Version (Оптимистическая блокировка)? Я более менее понял по отдельности каждую из этих 3 разделов, но не понимаю, как они связаны между собой. Выбранный уровень изоляции транзакций дает некие «гарантии» при проведении операций с таблицами. Параметры Lock.READ/Lock.WRITE позволяют добиться блокировки полей таблицы при проведении операций. Если я правильно понял, для этого как раз и используются поле @Version. Но все таки не до конца улавливаю связь между эти тремя понятиями. Или они дополняют друг друга, или это разные способы решения одних и тех же проблем. В общем, буду очень благодарен за ответ или же ссылку, где можно прочесть информацию именно по вопросы связи между этими понятиями и их совместном использовании.  **Второй вопрос**. Каким способом в Hibernate можно указать требуемый уровень изоляции транзакции для конкретной транзакции? И принято ли так делать?  Попробую ответить по порядку:  **Как соотносятся Lock и уровни изоляции**   1. Уровень изоляции транзакций (*transaction isolation level*) - относится к чтению данных, но не к записи данных; 2. Lock относится к доступу к данным (то есть затрагивает и чтение и запись и модификацию); 3. Можно сказать так, что механизм Lock обеспечивает работоспособность различных уровней изоляции данных, в самом тупом примере: если у вас стоит уровень изоляции READ\_COMMITTED, то вставка произведенная параллельной сессией будет локирована на запись пока сессия не выдаст commit(), но лок на чтение будет открыт сразу при запросе вашего select, а при REPEATABLE\_READ оба лока (на чтение и запись) будут стоять до commit() - соответственно ваш select увидит результаты модификации только после коммита.   **Каким способом в Hibernate можно указать требуемый уровень изоляции транзакции для конкретной транзакции?**  Вообще это не сильно здорово. Уровень изоляции транзакций это принадлежность коннекта к БД, соответственно фактически изменение уровня изоляции равносильно переконнекту к БД, а насколько это дорогая операция вы сами можете оценить исходя из имеющейся у вас ситуации.  Тем не менее, если вы таки решили менять перед каждой транзакцией уровень изоляции транзакций, то самый простой способ это отправить в СУБД *raw SQL query* (любой ORM это поддерживает), типа:  ALTER SESSION SET ISOLATION\_LEVEL SERIALIZABLE  ibernate использует JDBС подключение к базе напрямую без использования какого-либо режима блокировки. Очень важно перед подключением Hibernate ознакомится со спецификой изоляции транзакций Вашей СУБД, потому что режим, установленный уровнем изоляции транзакций, не изменяется во время использования Hibernate. Hibernate также не блокирует объекты в памяти.  Транзакции в БД всегда обязательны, любые контакты с базой данных обязательно должны быть заключены в транзакцию.  Чтобы избежать блокировки подключения к БД, физические транзакции в базе должны  быть как можно короче. Длинные транзакции делают приложение непригодным для  одновременного использования с высокой нагрузкой. Не держите транзакции  открытыми во время выполнения действий пользователем, вместо этого открывайте  транзакцию только после окончания его работы.  Hibernate сессия работает как своего рода кеш для транзакций, который предоставляет повторные операции чтения для поиска по идентификатору или выполнения запросов, которые приводят к загрузке сущностей.  *Используем сессии правильно*  Не нужно открывать и закрывать сессию в отдельном потоке при каждом обращении к БД. Это же утверждение верно и для транзакций. Группируйте запросы к БД в запланированные последовательности, а их уже заворачивайте в транзакцию. По этой же причине лучше отказаться от использования автокоммита. В режиме автокоммита JDBC драйвер просто выполняет неявный вызов транзакции для каждого запроса. Например, во время чтения данных — много мелких транзакций вряд ли будут лучше, чем одна четко определенная «единица работы». «Единица работы» в нашем случае это ряд операций, которые мы хотим выполнить в БД все вместе.  Наиболее распространенный паттерн использования транзакций в клиент-серверных приложениях это паттерн *session-per-request (сессия для запроса)*. Под запросом в данном случае имеется в виду запрос пользователя на выполнение операции, а не SQL-запрос в БД. В такой модели обработка осуществляется следующим образом:   * запрос от клиента передается на сервер, где работает Hibernate; * открывается новая сессия; * выполняются все операции с БД (в одной «единице работы»); * по завершению работы, или как только ответ для клиента был подготовлен, сессия закрывается.   Используйте одну транзакцию для обслуживания клиентского запроса, запускайте и коммитьте ее при открытии и закрытии сессии соответственно. Отношения между ними получаются один-к-одному и такая модель идеально подходит для многих приложений.  Чтобы упростить эту модель Hibernate предоставляет встроенное управление «текущей сессией». Приложение может получить доступ к «текущей сессии», вызвав метод sessionFactory.getCurrentSession(). Сессия создается только при первом обращении к методу getCurrentSession() и закрывается во время коммита транзакции. Поэтому при использовании getCurrentSession() не нужно беспокоиться о закрытии сессии, она закроется автоматически. Настройка результата работы метода getCurrentSession()производится с помощью интерфейса org.hibernate.context.spi.CurrentSessionContext и параметра конфигурации hibernate.current\_session\_context\_class (он должен принимать значение «thread» для автоматического управления контекстом сессии). |
| MVC | **Модель** Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель.  Модель обладает следующими признаками:   * Модель — это бизнес-логика приложения; * Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях; * Для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл); * Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;    **Представление (View)** В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.  Представление обладает следующими признаками:   * В представлении реализуется отображение данных, которые получаются от модели любым способом; * *В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.*   Примеры представления: HTML-страница, WPF форма, Windows Form. |
| Вопросы по ооп. 10 принципов ооп, солид. | <https://tproger.ru/translations/10-oop-principles/>  <https://javarush.ru/groups/posts/osnovnye-principy-dizajna-klassov-solid-v-java> |
| Как происходит обмен данными сервисы в хибернейт. | Сессия выдаётся в зависимости от натроек TransactionManager. По-умолчанию, Spring ассоциирует сессию с транзакцией и в течении одной транзакции будет выдавать один и тот же экземпляр сессии. Тут важно правильно получать ссессию из Spring, а не создавать каждый раз новую. |
| аррай и лист. сложность и как устроены. | <https://habr.com/ru/post/344288/>  <https://habr.com/ru/post/188010/>  <https://habr.com/ru/post/162017/> |
| как работает хешь на совсем низком уровне на java. | <https://habr.com/ru/post/169733/>  <https://habr.com/ru/company/mailru/blog/321306/> |
| почему сингелтон антипаттерн. | 1. **Синглтон нарушает SRP (Single Responsibility Principle)** — класс синглтона, помимо того чтобы выполнять свои непосредственные обязанности, занимается еще и контролированием количества своих экземпляров.   2. **Глобальное состояние**. Про вред глобальных переменных вроде бы уже все знают, но тут та же самая проблема. Когда мы получаем доступ к экземпляру класса, мы не знаем текущее состояние этого класса, и кто и когда его менял, и это состояние может быть вовсе не таким, как ожидается. Иными словами, корректность работы с синглтоном зависит от порядка обращений к нему, что вызывает неявную зависимость подсистем друг от друга и, как следствие, серьезно усложняет разработку.  3. **Зависимость обычного класса от синглтона не видна в публичном контракте класса**. Так как обычно экземпляр синглтона не передается в параметрах метода, а получается напрямую, через GetInstance(), то для выявления зависимости класса от синглтона надо залезть в тело каждого метода — просто просмотреть публичный контракт объекта недостаточно.   4.**Наличие синглтона понижает тестируемость приложения в целом и классов**, которые используют синглтон, в частности. Во-первых, вместо синглтона нельзя подпихнуть Mock-объект, а во-вторых, если синглтон имеет интерфейс для изменения своего состояния, то тесты начинают зависеть друг от друга. Говоря же проще — синглтон повышает связность, и все вышеперечисленное, в том или ином виде, есть следствие повышения связности.  Естественно, можно акккуратненько пройти по граблям и использовать синглетон, но (цитата из доки к пикоконтейнеру) "Overuse makes for bad solutions. At the enterprise level, it makes for very very bad solutions"...  Тем более, что при тщательном рассмотрении вопроса, использования синглтона, как правило, можно легко избежать. А если можно легко избежать, значит это нужно сделать, чтобы удержать себя от излишнего соблазна "оверюза"... Например, для контроля количества экземпляров объекта вполне можно (и нужно) использовать различного рода фабрики.  Наибольшая же опасность, как было сказано, подстерегает при попытке построить на основе сиглтонов всю архитектуру приложения, такому подходу существует масса замечательных альтернатив. Например, IoC контейнеры |
| Изолирование транзакции. | <https://easyjava.ru/spring/spring-data-access/izolyaciya-i-rasprostranenie-tranzakcij-v-spring/> |
| Транзакции в спринге. | <https://habr.com/ru/company/otus/blog/431508/> |
| Скоупы бина. | <https://easyjava.ru/spring/spring-framework/spring-bean-scopes/> |
| Как бины инжектятся. | <https://sysout.ru/sposoby-vnedreniya-zavisimostej-dependency-injection-v-spring/> |
| Proxy mode. Mode default | <https://habr.com/ru/post/347752/> |
| Слушатели в спринге. | События в спринге реализуются очень просто. Любой бин может прослушивать события если он реализует интерфейс **ApplicationListener**. Само событие это класс производный от **ApplicationEvent** а сам этот класс не обязан быть бином. Публикуются события с помощью метода **publishEvent()**класса реализующего интерфейс **ApplicationEventPublisher** который наследует интерфейс **ApplicationContext**.  **MessageEventListener** — класс слушатель. Классом слушателем может стать любой класс который **А)** является бином, **Б)** реализует интерфейс **ApplicationListener**. Когда выполняются два этих условия, **ApplicationContext** автоматически регистрирует его как бин-слушатель. В нашем примере, класс **MessageEventListener** заинтересован только в событиях **MessageEvent** так как он реализует типизированный интерфейс **ApplicationListener**:  Публикация события. Для того чтобы опубликовать событие, нужно создать экземпляр класса события и передать его методу **publishEvent** класса реализующего интерфейс **ApplicationEventPublisher** которым является класс **ApplicationContext** |
| 2 человека вносят изменения в одну и то же сущность. Без перетирания. <http://javastudy.ru/hibernate/hibernate-envers-revisions/> |  |
|  | Есть сущность который отмечена как bean и в ней есть два public метода. Оба метода помещены аннотацией @transactional.  @Component  public class Main{  @transactional  public void one(){  //вызывает метод two()  }  @transactional  public void two(){}  }  мне интересно будет создано 2 транзакции или все таки одна? Ведь вызов происходит внутри одного метода и общего бина.  Если мы говорим про аннотацию javax.transaction.Transactional, то в состоянии "по-умолчанию", будет создана одна транзакция. Аннотация принимает в качестве аргумента так называемый TxType. Это такой enum, которым внутри контекста разруливается поведение транзакции. По-умолчанию выставлено значение TxType.REQUIRED, которое означает, что при выполнении метода требуется сессия, для выполнения транзакции. Если такой сессии внутри контекста нет -она будет создана (в вашем случае в момент вызова метода one), если же сессия уже есть в контексте (вызов метода two), то текущая транзакция будет продолжена. |
| [Inject beans с разными Scope](https://ru.stackoverflow.com/questions/1026913/inject-beans-%d1%81-%d1%80%d0%b0%d0%b7%d0%bd%d1%8b%d0%bc%d0%b8-scope) | Есть ли какие нибудь ограничения или правила использования бинами друг-друга с разными scope.   * singleton, prototype, request, session, globalSession.   Например у меня есть бин singleton, могу ли я заинжектить в него бин с session и как это будет работать?   * Если proxy-классы бинов создаются с помощью cglib, то можете. – [Sergey Gornostaev](https://ru.stackoverflow.com/users/204271/sergey-gornostaev" \o "57,158 баллов репутации) [22 сен в 14:02](https://ru.stackoverflow.com/questions/1026913/inject-beans-%d1%81-%d1%80%d0%b0%d0%b7%d0%bd%d1%8b%d0%bc%d0%b8-scope#comment1747204_1026913)   Динамическое генерирование прокси-классов в Java |
| [Нарушает ли hibernate принцип инкапсуляции](https://ru.stackoverflow.com/questions/1025883/%d0%9d%d0%b0%d1%80%d1%83%d1%88%d0%b0%d0%b5%d1%82-%d0%bb%d0%b8-hibernate-%d0%bf%d1%80%d0%b8%d0%bd%d1%86%d0%b8%d0%bf-%d0%b8%d0%bd%d0%ba%d0%b0%d0%bf%d1%81%d1%83%d0%bb%d1%8f%d1%86%d0%b8%d0%b8) | Вы абсолютно правы, hibernate нарушает принцип инкапсуляции. И даже более того, любая [ORM](https://ru.wikipedia.org/wiki/ORM) заставляет нас использовать [POJO классы, которые не имеют ничего общего с инкапсуляцией](https://nvoulgaris.com/how-getters-and-setters-harm-encapsulation/).  Тем не менее, есть способ работать с БД, не нарушая принцип инкапсуляции - [sql-speaking-objects](https://www.yegor256.com/2014/12/01/orm-offensive-anti-pattern.html" \l "sql-speaking-objects).  Я никого ни к чему не призываю, просто есть разные подходы работы с БД. Однако некоторые программисты могут заметить, что на самом деле POJO классы - это не объекты, это данные, а, значит, там не может нарушаться принципы объектного проектирования. Такое суждение имеет право на жизнь, при таком подходе в программах перемешиваются данные и объекты, работающие с данными, но это уже не имеет ничего общего с ООП, потому что в ОО архитектуре есть только объекты, а данные хранятся всегда внутри объектов и никогда не выглядывают наружу. |
| Фабрика и строитель | <https://javarush.ru/groups/posts/584-patternih-proektirovanija> |
| Методы класса object | <https://javarush.ru/quests/lectures/questmultithreading.level01.lecture03> |
| Пара задачек на лямбду |  |
| Пара задачек на sql | <https://habr.com/ru/post/181033/> |
| Задача на поиск первого вхождения элемента и его индекса |  |
| concurrent | <https://habr.com/ru/company/luxoft/blog/157273/> |
| Synchronized and volatile | <https://habr.com/ru/post/108016/> |
| deadlock | public class TestThread {  public static Object Lock1 = new Object();  public static Object Lock2 = new Object();  public static void main(String args[]) {  ThreadDemo1 T1 = new ThreadDemo1();  ThreadDemo2 T2 = new ThreadDemo2();  T1.start();  T2.start();  }  private static class ThreadDemo1 extends Thread {  public void run() {  synchronized (Lock1) {  System.out.println("Thread 1: Holding lock 1...");  try { Thread.sleep(10); }  catch (InterruptedException e) {}  System.out.println("Thread 1: Waiting for lock 2...");  synchronized (Lock2) {  System.out.println("Thread 1: Holding lock 1 & 2...");  }  }  }  }  private static class ThreadDemo2 extends Thread {  public void run() {  synchronized (Lock2) {  System.out.println("Thread 2: Holding lock 2...");  try { Thread.sleep(10); }  catch (InterruptedException e) {}  System.out.println("Thread 2: Waiting for lock 1...");  synchronized (Lock1) {  System.out.println("Thread 2: Holding lock 1 & 2...");  }  }  }  }  } |
|  |  |
|  | @RestController  public class UserController {  @Autowired  private UserService userService;  public User getUser() {  return userService.updateUserById(1L);  }  }  @Component  public class UserService {  @Transactional  public int updateUserById(Long userId) {  User user = updateUserByIdWithBusinessLogic (userId);  List<Role> roles = user.getRoles();  int i = roles.size();  return i;  }  public User updateUserByIdWithBusinessLogic(Long userId) {  <бизнес логика по работе с базой данных>  }  }  @Entity  User  List<Roles> fetchType = lazy |
|  |  |
|  |  |
|  |  |

Всегда повторялись

|  |  |
| --- | --- |
| Hashmap |  |
| Interface |  |
| App Server vs Web Server. |  |
| Spring. Difference between @ и xml. |  |
| How client-server works? |  |
| Request/Session |  |
| Session object |  |
| HTTP Services? |  |
| REST vs SOAP. |  |
| WSDL |  |
| JMS, MQ, integration technologies. |  |
| Async and sync integration. |  |
| Hibernate, + and – нарушение икапсуляции |  |
| Transaction. |  |
| Одна transaction вызывает другу, дедлок в хибернейт |  |
| MVC | Знать внутренную солянку наизусть |
| Оптимизация рекурсии с факториалом | Использовать хешь |
|  | Три мапа ее сложность и сложность графа. Используется в БД.  Чтобы передать объект в try() он должен имплементировать клозебл  Амотамрные операции и атомикинтеджер  Блокированные очереди  Конкаррент хешь мапа создает копию при обходе итератором и поэтому не ломается из изменении. Так же баскеты делять на группы для потоков.  Каллебл от раннебл  Алгоритм сортировки и поиска двинчый и бинарный  Паттерны четко знать 3 штуки, фасад и как реализуются |
| Jvm, Jre, jdk |  |
|  | Singleton (Одиночка) Описание:   * Ограничивает создание одного экземпляра класса, обеспечивает доступ к его единственному объекту. Конструктор класса приватный. Метод getInstance() создает только один экземпляр класса.  Factory (Фабрика) Описание:   * Используется, когда у нас есть суперкласс с несколькими подклассами и на основе ввода, нам нужно вернуть один из подкласса. Класс не знает какого типа объект он должен создать. Объекты создаются в зависимости от входящих данных.  Abstract Factory (Абстрактная фабрика) Описание:   * Позволяет выбрать конкретную реализацию фабрики из семейства возможных фабрик. Создает семейство связанных объектов. Легко расширять.  Builder (Строитель) Описание:   * Используется для создания сложного объекта с использованием простых объектов. Постепенно он создает больший объект от малого и простого объекта. Позволяет изменять внутреннее представление конечного продукта.  Prototype (Прототип) Описание:   * Помогает создать дублированный объект с лучшей производительностью, вместо нового создается возвращаемый клон существующего объекта. Клонирует существующий объект.  СтруктурныеAdapter (Адаптер) Описание:   * Используя паттерн, мы можем объединить два несовместимых объекта. Конвертер между двумя несовместимыми объектами. |
| Потокобезопасны ли service в Spring |  |
| Изолированность транзакций в хибернейт |  |
| Thread Lock роль и где используется |  |
| Свойства callable, можно ли остановить поток |  |
| Как изнутри спринг понимает, какой бин когда создавать, как он работает, как начинает загрузка, как определить какой бин когда вызывать.  Так что такое бин как единица информации, какую роль играет и в чем отличие от компонента. |  |
| Rest контроллер = контроллер + возврат сущности.  Подробно. |  |
|  |  |
|  | Разница между PUT и POST - это вопрос семантики. Коль скоро для операций используются разные глаголы, то и смысл у них должен быть разным.  Представьте, что ваш сервис оперирует понятиями блокнот (notebook) и запись (post). Один блокнот может содержать множество записей.  Для **добавления** новой записи в блокнот c идентификатором **id** вы будете использовать метод **POST** с URL mydomain/notebooks/id/. Ваш сервис, ориентируясь на метод POST, сам присвоит нужный идентификатор записи, добавит ее в блокнот и вернет вам URL созданной записи (для доступа к записи по GET или для удаления по DELETE). При этом хорошо бы вернуть клиенту URL созданной записи.  Допустим, запись с идентификатором **post-id** уже создана и доступна по URL mydomain/notebooks/id/posts/post-id. Но клиент (владелец записи) исправил в ней ошибку и хочет **перезаписать** ее. Для этого он использует метод **PUT** с URL mydomain/notebooks/id/posts/**post-id** и передает обновленную запись в теле запроса. Ваш сервис, ориентируясь на метод PUT удаляет старую запись и записывает новую, при этом она доступна по тому же URL.  Конечно, никто не мешает вам всегда использовать метод POST (например HTML 4 позволял использовать только методы GET и POST). Но все же стоит придерживаться рекомендаций в целях единообразной трактовки методов всеми разработчиками.  **UPD:** Вспомнил еще кое-что. Рекомендуется использоваться метод POST для создания подчиненного ресурса (дочернего по отношению к другому ресурсу; пример блокнота и записи как раз очень подходит). |