Что такое платформа Spring Model-View-Controller (MVC)?

Платформа Spring MVC предоставляет архитектуру контроллера представления модели и готовые компоненты, используемые для разработки слабо связанных веб-приложений. Используя MVC, вы можете разделить различные аспекты программы, такие как бизнес, логика ввода и пользовательский интерфейс, сохраняя при этом слабую связь между ними. Это обеспечивает большую гибкость в ваших веб-приложениях.

Как использовать JavaEE сервлет в Spring Framework?

Web-приложение на Spring MVC технически само по себе работает на сервлетах: всю обработку запросов берет на себя единый DispatcherServlet. С его помощью реализуется паттерн Front Controller.

Если вам нужно определить в программе полностью независимый от Spring-контекста сервлет или фильтр, ничего особенного для этого делать не нужно. Как обычно в Servlet API, нужно объявить класс, добавить его в web.xml как сервлет, добавить для сервлета маппинг.

Сервлет живет вне Spring-контекста, внедрение зависимостей в нём просто так не заработает. Чтобы использовать autowiring, на этапе инициализации сервлета вызывается статический SpringBeanAutowiringSupport.processInjectionBasedOnServletContext, с текущим сервлетом и его контекстом в аргументах. В этом же утилитарном классе есть ряд других средств для работы с контекстом извне.

Если программа построена на Spring Boot, создание бина типа ServletRegistrationBean поможет добавить сервлеты в рантайме. А для декларативного добавления на этапе компиляции, к классу конфигурации применяется @ServletComponentScan. С этой аннотацией стартер приложения просканирует и добавит в контекст все web-компоненты в стиле Servlet 3.0: классы с аннотациями @WebFilter, @WebListener и @WebServlet.

Что такое контроллер в Spring MVC?

Ключевым интерфейсом в Spring MVC является Controller. Контроллер обрабатывает запросы к действиям, осуществляемые пользователями в пользовательском интерфейсе, взаимодействуя с уровнем обслуживания, обновляя модель и направляя пользователей на соответствующие представления в зависимости от результатов выполнения. Controller - управление, связь между моделью и видом.

Основным контроллером в Spring MVC является org.springframework.web.servlet. DispatcherServlet. Задается аннотацией @Controller и часто используется с аннотацией @RequestMapping, которая указывает какие запросы будут обрабатываться этим контроллером.

Какая разница между аннотациями @Component, @Repository и @ Service в Spring?

@Component - используется для указания класса в качестве компонента spring. При использовании поиска аннотаций, такой класс будет сконфигурирован как spring bean.

@Controller - специальный тип класса, применяемый в MVC приложениях. Обрабатывает запросы и часто используется с аннотацией @RequestMapping.

@Repository - указывает, что класс используется для работы с поиском, получением и хранением данных. Аннотация может использоваться для реализации шаблона DAO.

@Service - указывает, что класс является сервисом для реализации бизнес логики (на самом деле не отличается от Component, но просто помогает разработчику указать смысловую нагрузку класса).

Для указания контейнеру на класс-бин можно использовать любую из этих аннотаций. Но различные имена позволяют различать назначение того или иного класса.

Например, бины, получившиеся при помощи @Repository, дополнительно имеют обработку для JDBC Exception

Можем ли мы использовать @ Component вместо @Service для бизнес логики?

Да. конечно.

Если @Component является универсальным стереотипом для любого Spring компонента, то @Service в настоящее время является его псевдонимом. Однако, в официальной документации Spring рекомендуется использовать именно @ Service для бизнес логики. Вполне возможно, что в будущих версиях фреймворка, для данного стереотипа добавится дополнительная семантика, и его бины станут обладать дополнительной логикой.

Расскажите, что вы знаете о DispatcherServlet и ContextLoaderListener.

DispatcherServlet - сервлет диспатчер. Этот сервлет анализирует запросы и направляет их соответствующему контроллеру для обработки. В Spring MVC класс DispatcherServlet является центральным сервлетом, который получает запросы и направляет их соответствующим контроллерам. В приложении Spring MVC может существовать произвольное количество экземпляров DispatcherServlet, предназначенных для разных целей (например, для обработки запросов пользовательского интерфейса, запросов веб-служб REST и т.д.). Каждый экземпляр DispatcherServlet имеет собственную конфигурацию WebApplicationContext, которая определяет характеристики уровня сервлета, такие как контроллеры, поддерживающие сервлет, отображение обработчиков, распознавание представлений, интернационализация, оформление темами, проверка достоверности, преобразование типов и форматирование и т.п.

ContextLoaderListener - слушатель при старте и завершении корневого класса Spring WebApplicationContext. Основным назначением является связывание жизненного цикла ApplicationContext и ServletContext, а так же автоматического создания ApplicationContext. Можно использовать этот класс для доступа к бинам из различных контекстов спринг. Настраивается в web.xml:

Какой жизненный цикл у запроса?

Запрос приходит в DispatcherServlet

DispatcherServlet отправляет запрос на один из контроллеров, основываясь на URL из запроса

Контроллер обрабатывает запрос, делегирует выполнение бизнес-логике бизнес-слою (как правило это классы с аннотацией @Service), и создает модель с данными, которую и отправляет обратно в DispatcherServlet

DispatcherServlet отправляет модель на фронт для вью, основываясь на интерфейсе ViewResolver(подробнее об этом ниже)

DispatcherServlet Создан ли экземпляр в контексте приложения?

Нет, DispatcherServlet экземпляр создается сервлет-контейнерами, такими как Tomcat или Jetty. Вы должны определить DispatcherServlet в файл web.xml, как показано ниже.

Вы можете видеть, что тег загрузки при запуске имеет значение 1, что означает, что DispatcherServlet он создается при развертывании приложения Spring MVC в Tomcat или любом другом контейнере сервлетов. Во время создания он ищет файл servlet-name-context.xml и затем инициализирует bean-компоненты, определенные в этом файле.

Что такое корневой контекст приложения в Spring MVC? Как это загружается?

В Spring MVC контекст, загружаемый с использованием ContextLoaderListener , называется «корневым» контекстом приложения, который принадлежит всему приложению, в то время как тот, который инициализирован с использованием DispatcherServlet , фактически специфи-

чен для этого сервлета.

Технически Spring MVC допускает множественное использование DispatcherServlet в веб-приложении Spring MVC, поэтому каждый контекст является специфическим для соответствующего сервлета. Но, имея тот же корневой контекст, может существовать.

Что такое ContextLoaderListener и для чего это нужно?

Это ContextLoaderListener слушатель, который помогает загрузить Spring MVC. Как следует из названия, он загружается и создает ApplicationContext, так что вам не нужно писать явный код для его создания.

Контекст приложения — это то, куда уходит Spring bean. Для веб-приложения существует подкласс WebAppliationContext.

ContextLoaderListener Также связывает жизненный цикл ApplicationContext для жизненного цикла ServletContext. Вы можете получить ServletContext с WebApplicationContext помощью getServletContext() метода.

Что вы собираетесь делать в web.xml? Где вы это разместите?

Он ContextLoaderListener настроен в web.xml как слушатель, и вы помещаете его в тег, как показано ниже:

<listener>

<listener-class> org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

При развертывании веб-приложения Spring MVC контейнер сервлетов создал экземпляр ContextLoaderListener класса, который загружает Spring WebApplicationContext. Вы также можете увидеть Spring MVC для начинающих, чтобы узнать больше об ContextLoaderListener и WebApplicationContext и их роли в Spring MVC.

Каковы части фреймворка Spring MVC?

Тремя основными частями MVC являются:

• DispatcherServlet: Эта часть MVC управляет всеми HTTP-запросами и ответами, которые взаимодействуют с программой. DispatcherServlet сначала получает соответствующее сопоставление обработчика из файла конфигурации, а затем передает запрос контроллеру. DispatcherServlet является наиболее важной частью платформы Spring Web MVC.

• WebApplicationContext: Это действует как расширение обычного ApplicationContext с дополнительными функциями, необходимыми для веб-приложений. Он может однозначно разрешать темы и автоматически определять, с каким сервлетом он связан.

• Контроллеры: Это компоненты в DispatcherServlet, которые действуют как фильтры между вводом данных пользователем и ответом приложения. Контроллеры принимают ввод пользователя, решают, следует ли преобразовать его в Представление или Модель, и, наконец, возвращают преобразованный ввод в Распознаватель представлений для просмотра.

Как входящий запрос сопоставляется с контроллером и сопоставляется с методом?

Иногда также задают этот вопрос: как DispatcherServlet узнать, какой контроллер должен обработать запрос? Ну, ответ лежит в том, что называется отображением обработчика.

Spring использует сопоставления обработчиков для связи контроллеров с запросами. Два из наиболее часто используемых отображений обработчиков — это BeanNameUrlHandlerMapping и SimpleUrlHandlerMapping.

Если BeanNameUrlHandlerMappingURL-адрес запроса совпадает с именем компонента, класс в определении компонента является контроллером, который будет обрабатывать запрос.

С другой стороны SimpleUrlHandlerMapping, отображение более явное. Вы можете указать количество URL, и каждый URL может быть явно связан с контроллером.

Если вы используете аннотации для настройки Spring MVC, что необходимо, тогда @RequestMapping аннотации используются для сопоставления входящего запроса с контроллером и методом-обработчиком.

Вы также можете настроить @RequestMapping аннотацию по пути URI, параметрам запроса, HTTP-методам запроса и HTTP-заголовкам, присутствующим в запросе.

В чём разница между @Controller и @RestController?

Controller - это один из стереотипов Spring Framework. Компоненты такого типа обычно занимаются обработкой сетевых запросов. Контроллер состоит из набора методов-обработчиков, помеченных аннотацией @RequestMapping.Ответ на запрос можно сформировать разными способами: например просто вернуть из обработчика строку с именем jsp-файла, или же вернуть ResponseBodyEmitter, который будет асинхронно заполняться данными позже. Все возможные варианты перечислены в документации.Большинство современных API реализуется по архитектуре REST. В ней каждая сущность доступна под собственным URI. В методе-обработчике возвращается экземпляр класса этой сущности, который преобразуется в ответ сервера одним из HttpMessageConverter-ов. Например, в JSON его превратит MappingJackson2 HttpMessageConverter. Чтобы использовать этот способ ответа, метод, или весь контроллер, должен иметь аннотацию @ResponseBody.@RestController - это просто сокращенная запись для @Controller + @ResponseBody.

Что такое ViewResolver в Spring?

ViewResolver - распознаватель представлений. Интерфейс ViewResolver в Spring MVC (из пакета org.springframework.web.servlet) поддерживает распознавание представлений на основе логического имени, возвращаемого контроллером. Для поддержки различных механизмов распознавания представлений предусмотрено множество классов реализации. Например, класс UrlBasedViewResolver поддерживает прямое преобразование логических имен в URL. Класс ContentNegotiatingViewResolver поддерживает динамическое распознавание представлений в зависимости от типа медиа, поддерживаемого клиентом (XML, PDF, JSON и т.д.). Существует также несколько реализаций для интеграции с различными технологиями представлений, такими как FreeMarker (FreeMarkerViewResolver), Velocity (VelocityViewResolver) и JasperReports (JasperReportsViewResolver).

InternalResourceViewResolver - реализация ViewResolver, которая позволяет находить представления, которые возвращает контроллер для последующего перехода к нему. Ищет по заданному пути, префиксу, суффиксу и имени.

Что такое MultipartResolver и когда его использовать?

Интерфейс MultipartResolver используется для загрузки файлов. Существуют две реализации: CommonsMultipartResolver и StandardServletMultipartResolver, которые позволяют фреймворку загружать файлы. По умолчанию этот интерфейс не включается в приложении и необходимо указывать его в файле конфигурации. После настройки любой запрос о загрузке будет отправляться этому интерфейсу.

Для чего @RequestParam используется?

Это @RequestParam аннотация Spring MVC, которая используется для извлечения параметра запроса или параметров запроса из URL-адреса в методе обработчика контроллера, как показано ниже:

public String personDetail(@RequestParam(«id») long id)

{

.... return «personDetails»;

}

@RequestParam Аннотаций также поддерживает преобразование типов данных, например , вы можете увидеть здесь строка преобразуется в автоматический вход в систему , но она также может привести к исключению , если параметр запроса нет , или в случае несоответствия типа. Вы также можете сделать параметр необязательным, используя требуемый = false, например @RequestParam (value = «id», required = false)

Каковы различия между @ RequestParam и @PathVariable ?

Несмотря на то, что оба @RequestParam и @PathVariable аннотации используются для извлечения некоторых данных из URL, между ними есть ключевое различие.

@RequestParam Используется для параметров экстракта запросов, например , что — нибудь после «?» в URL, в то время @PathVariable как используется для извлечения части самого URI. Например, если задан URL-адрес http: // localhost: 8080 / SpringMVC / books / 3232233 /? Format = json, то вы можете получить доступ к параметру запроса «format» с помощью @RequestParam аннотации и / books / {id} с помощью @PathVariable, который даст вам 3232233.

Вот еще один пример @PathVariable:

@RequestMapping(«/persons/{id}»)

public String personDetail (@PathVariable («id») long id) {...}

Этот код может извлечь лицо id = 123 из / персон / 123. Он особенно используется в веб-службах RESTful, поскольку их идентификатор обычно является частью пути URI или URL-адреса.

Расскажите про аннотацию @ RequestMapping

Это аннотация в основном используется для указания URI для класс-контроллера. Раньше ее использовали методов класса, чтобы указать URI, http-метод, тип отправляемых данных, и т.п. В более новых версиях Spring ее заменили на аннотации @GetMapping, @PostMapping, и т.п. Теперь она используется только для указания URI до класса-контроллера.

Что за аннотации @GetMapping, @ PostMapping, @DeleteMapping и прочие?

Это более узкие аннотации для маппинга http-методов.

@GetMapping — Обрабатывает get-запросы

@PostMapping — Обрабатывает post-запросы

@DeleteMapping — Обрабатывает delete-запросы

@PutMapping — Обрабатывает put-запросы

@PatchMapping — Обрабатывает patch-запросы

Все написанное ниже характерно также и для других аннотаций.

Аннотация @GetMapping — это просто аннотация которая содержит @RequestMapping(method = RequestMethod.GET).Она также позволяет более глубоко настроить метод-обработчик.Ее параметры(они конвертируются в аналогичные параметры @ RequestMapping):

path — URI

headers — заголовки

name — имя обработчика

params — параметры

produces — тип возвращаемых данных(JSON, XML, текст). Используется в REST

consumes — тип принимаемых данных. Используется в REST

По умолчанию аннотация принимает путь до метода.@GetMapping(«managers») = @GetMapping(path = «managers»)

Что за аннотация @RequestBody?

Она используется для того чтобы указать что метод оперирует не моделями, а данными. То есть отправляет JSON, XML, текст, и т.п. Обычно она неявно используется в REST-сервисах.

Что такое View и какова идея поддержки различных типов View?

A View — это интерфейс в приложении Spring MVC, реализации которого отвечают за отображение контекста и представление модели. Один вид предоставляет несколько атрибутов модели. Представления в Spring MVC могут быть бобами.

Они могут быть созданы как бобы ViewResolver. Поскольку этот интерфейс не имеет состояния, реализации представлений должны быть поточно-ориентированными. При использовании ViewResolverлогическое имя представления может быть преобразовано в различные типы View реализации, например, JstlView для отображения JSP или других реализаций представления для FreeMarker и Velocity.

Как выбирается View в фазе рендеринга? Как отображается View?

DispatcherServlet содержит список специальных «отображателей» для view, которые основываясь на конфигурации сервлета будут содержать бины реализующие интерфейс ViewResolver.

Процесс отображения view:

Контроллер возвращает имя view в DispactherServlet

Имя сопоставляется с именами во ViewResolver

Если находится подходящий ViewResolver, он возвращает View который должен использоваться при рендеринге.

DS передает модель с данными во View и отображает вывод(html-страницу)

Что такое Model?

Model является ссылкой для инкапсуляции данных или вывода для визуализации. Model всегда создается и передается в представление в Spring MVC. Если метод сопоставленного контроллера имеет Model в качестве параметра метода, model Spring Framework автоматически внедряет в этот метод экземпляр.

Все атрибуты, установленные на внедренной модели, сохраняются и передаются в View. Вот пример использования Model в Spring MVC:

public String personDetail(Model model) { ... model.addAttribute(«name», «Joe»); ... }

Почему у вас есть доступ к model вашему View? Откуда это взялось?

У вас должен быть доступ к model вашему представлению, чтобы отобразить вывод. Это тот, model который содержит данные для визуализации. Model Поставляется с контроллером, который обрабатывает их запрос клиента и инкапсулирует выход в Model объект.

Как загрузить файл в Spring MVC?

Внутри спринг предусмотрен интерфейс MultipartResolver для обеспечения загрузки файлов. Фактически нужно настроить файл конфигурации для указания обработчика загрузки файлов, а затем задать необходимый метод в контроллере spring.

Как обрабатывать исключения в Spring MVC Framework?

В Spring MVC интерфейс HandlerExceptionResolver (из пакета org. springframework.web.servlet) предназначен для работы с непредвиденными исключениями, возникающими во время выполнения обработчиков. По умолчанию DispatcherServlet регистрирует класс DefaultHandlerExceptionResolver (из пакета org.springframework.web.servlet. mvc.support). Этот распознаватель обрабатывает определенные стандартные исключения Spring MVC, устанавливая специальный код состояния ответа. Можно также реализовать собственный обработчик исключений, аннотировав метод контроллера с помощью аннотации @ ExceptionHandler и передав ей в качестве атрибута тип исключения. В общем случае обработку исключений можно описать таким образом:

Controller Based - указать методы для обработки исключения в классе контроллере. Для этого нужно пометить такие методы аннотацией @ExceptionHandler.

Global Exception Handler - для обработки глобальных исключений spring предоставляет аннотацию @ControllerAdvice.

HandlerExceptionResolver implementation - общие исключений большая часть времени обслуживают статические страницы. Spring Framework предоставляет интерфейс HandlerExceptionResolver, который позволяет задать глобального обработчика исключений. Реализацию этого интерфейса можно использовать для создания собственных глобальных обработчиков исключений в приложении.

Каковы минимальные настройки, чтобы создать приложение Spring MVC?

Для создания простого Spring MVC приложения необходимо пройти следующие шаги:

Добавить зависимости spring-context и spring-webmvc в проект.

Указать DispatcherServlet в web.xml для обработки запросов внутри приложения.

Задать определение spring bean (аннотацией или в xml).

Добавить определение view resolver для представлений.

Настроить класс контроллер для обработки клиентских запросов.

Как бы вы связали Spring MVC Framework и архитектуру MVC?

Модель (Model) - выступает любой Java bean в Spring. Внутри класса могут быть заданы различные атрибуты и свойства для использования в представлении.

Преставление (View) - JSP страница, HTML файл и т.п. служат для отображения необходимой информации пользователю. Представление передает обработку запросов к диспетчеру сервлетов (контроллеру).

DispatcherServlet (Controller) - это главный контроллер в приложении Spring MVC, который обрабатывает все входящие запросы и передает их для обработки в различные методы в контроллеры.

Как добиться локализации в приложениях Spring MVC?

Spring MVC предоставляет очень простую и удобную возможность локализации приложения. Для этого необходимо сделать следующее:

Создать файл resource bundle, в котором будут заданы различные варианты локализированной информации.

Определить messageSource в конфигурации Spring используя классы ResourceBundleMessageSource или ResourceBundleMessageSource.

Определить localceResolver класса CookieLocaleResolver для включения возможности переключения локали.

С помощью элемента spring:message DispatcherServlet будет определять в каком месте необходимо подставлять локализированное сообщение в ответе.

Как мы можем использовать Spring для создания веб-службы RESTful, возвращающей JSON?

Spring Framework позволяет создавать Resful веб сервисы и возвращать данные в формате JSON. Spring обеспечивает интеграцию с Jackson JSON API для возможности отправки JSON ответов в resful web сервисе. Для отправки ответа в формате JSON из Spring MVC приложения необходимо произвести следующие настройки:

Добавить зависимости Jackson JSON. С помощью maven это делается так:

Настроить бин RequestMappingHandlerAdapter в файле конфигурации Spring и задать свойство messageConverters на использование бина MappingJackson2HttpMessageConverter.

В контроллере указать с помощью аннотации @ResponseBody возвращение Object:

Как проверить (валидировать) данные формы в Spring Web MVC Framework?

Spring поддерживает аннотации валидации из JSR-303, а так же возможность создания своих реализаций классов валидаторов. Пример использования аннотаций:

Что вы знаете Spring MVC Interceptor и как он используется?

Перехватчики в Spring (Spring Interceptor) являются аналогом Servlet Filter и позволяют перехватывать запросы клиента и обрабатывать их. Перехватить запрос клиента можно в трех местах: preHandle, postHandle и afterCompletion.

preHandle - метод используется для обработки запросов, которые еще не были переданы в метода обработчик контроллера. Должен вернуть true для передачи следующему перехватчику или в handler method. False укажет на обработку запроса самим обработчиком и отсутствию необходимости передавать его дальше. Метод имеет возможность выкидывать исключения и пересылать ошибки к представлению.

postHandle - вызывается после handler method, но до обработки DispatcherServlet для передачи представлению. Может использоваться для добавления параметров в объект ModelAndView.

afterCompletion - вызывается после отрисовки представления.

Для создания обработчика необходимо расширить абстрактный класс HandlerInterceptorAdapter или реализовать интерфейс HandlerInterceptor. Так же нужно указать перехватчики в конфигурационном файле Spring.

В чем разница между Filters, Listeners and Interceptors?

Концептуально всё просто, фильтры сервлетов могут перехватывать только HTTPServlets. Listeners могут перехватывать специфические события. Как перехватить события которые относятся ни к тем не другим?

Фильтры и перехватчики делают по сути одно и тоже: они перехватывают какое-то событие, и делают что-то до или после.

Java EE использует термин Filter, Spring называет их Interceptors.

Именно здесь AOP используется в полную силу, благодаря чему возможно перехватывание вызовов любых объектов

В чем разница между ModelMap и ModelAndView?

Model — интерфейс, ModelMap его реализация..

ModelAndView является контейнером для пары, как ModelMap и View.

Обычно я люблю использовать ModelAndView. Однако есть так же способ когда мы задаем необходимые атрибуты в ModelMap, и возвращаем название View обычной строкой из метода контроллера.

В чем разница между model.put() и model.addAttribute()?

Метод addAttribute отделяет нас от работы с базовой структурой hashmap. По сути addAttribute это обертка над put, где делается дополнительная проверка на null. Метод addAttribute в отличии от put возвращает modelmap.model.addAttribute(«attribute1»,»value1»).addAttribute(«attribute2»,»value2»);

Что можете рассказать про Form Binding?

Нам это может понадобиться, если мы, например, захотим взять некоторое значение с HTML страницы и сохранить его в БД. Для этого нам надо это значение переместить в контроллер Спринга.

Если мы будем использовать Spring MVC form tags, Spring автоматически свяжет переменные на HTML странице с Бином Спринга.

Если мне придется с этим работать, я обязательно буду смотреть официальную документацию Spring MVC Form Tags.

Для чего был создан REST?

Чтобы понять концепцию REST, нужно разобрать акроним на его составляющие:

Representational — ресурсы в REST могут быть представлены в любой форме — JSON, XML, текст, или даже HTML — зависит от того, какие данные больше подходят потребителю

State — при работе с REST вы должны быть сконцентрированы на состоянии ресурса, а не на действиях с ресурсом

Transfer — REST включает себя передачу ресурсных данных, в любой представленной форме, от одного приложения другому.

REST это передача состояний ресурса между сервером и клиентом.

Что такое ресурс?

Ресурс в REST — это все, что может быть передано между клиентом и сервером. Вот несколько примеров ресурсов:

Новость

Температура в Санкт-Петербурге в понедельник в 4 утра

Зарплата сотрудника

Выборка из базы данных

Результат поиска

Что обозначает CRUD?

Действия в REST определяются http-методами.Get, Post, Put, Delete, Patch, и другие.

Самые часто-используемые обозначаются аббревиатурой CRUD:

Create — POST

Read — GET

Update — PUT

Delete — DELETE

REST безопасен? Как вы можете защитить его?

По умолчанию REST не защищен.

Вы можете настроить безопасность с помощью Basic Auth, JWT, OAuth2

Что такое save operations?

Это операции, которые не модифицируют ресурсы. Вот их список:

GET

HEAD

OPTIONS

Что такое идемпотентая операция? Почему идемпотентность важна?

Идемпотентые методы — это методы, при каждом вызове которых результат будет одинаковый.

То есть, результат после 1 вызова такого метода будет такой же, как и результат после 10 вызовов этого метода.

Это важно для отказоустойчевого API. Предположим, что клиент хочет обновить ресурс с помощью POST-запроса? Если POST не идемпотентный метод, то при многократном вызове возникнут непредвиденные обновления ресурса. Используя идемпотентные методы, вы ограждаете себя от многих ошибок.

REST хорошо масштабируется?

Да. REST хорошо масштабируется потому что он не хранит состояние.

Это значит что он не хранит информацию о пользовательских сессиях на сервере.

Информация о клиенте не должна хранится на стороне сервера, а должна передаваться каждый раз туда, где она нужна. Вот что значит ST в REST, State Transfer. Вы передаете состояние, а не храните его на сервере.

REST также интероперабельный — это значит, что на нем могут взаимодействовать разные программы написанные на разных языках. Это исходит из 2ух факторов:

Интероперабельные HTTP-клиенты. Разные клиенты должны отправлять одинаковые http-запросы.

Интероперабельность на уровне медиа-типов. Различные клиенты должны корректно отправлять и получать одни и те же ресурсы.

Что такое HttpMessageConverter?

HttpMessageConverter конвертирует запрос в объект и наоборот.

Spring имеет несколько реализаций этого интерфейса, а вы можете создать свою.

В этом случае DispatcherServlet не использует Model и View.

В REST вообще не существует Model и View. Есть только данные, поставляемые контроллером, и представление ресурса, когда сообщение конвертируется из медиа-типа(json, xml...) в объект.

Список конвертеров:

BufferedImageHttpMessageConverter — конвертирует BufferedImage в(из) код изображения.

Jaxb2RootElementHttpMessageConverter — конвертирует xml в(из) объект, помеченный jaxb2 аннотациями. Регистрируется, если jaxb2 находится в classpath.

MappingJackson2HttpMessageConverter — конвертирует JSON в(из) объект. Регистрируется, если Jackson 2 находится в classpath.

StringHttpMessageConverter — конвертирует все медиа-файлы в text/plain.

Зачем нужна @ResponseBody?

Аннотация @ResponseBody ставится на методы, которые работают с данными, а не с моделями. Ее не требуется указывать явно, если используется @RestController.

Обычные методы возвращают Model, а методы аннотированные @ResponseBody возвращают объекты, которые конвертируются в медиа-файлы с помощью HttpMessageConverter.

Зачем нужна аннотация @PathVariable?

Эта аннотация получает определенную часть из URI.

URI: http://localhost:8080/getById/23

Следующий код поместит в переменную id значение 23.

@GetMapping(«getById/{id}») public User getUserById(@PathVariable(«id») String id) { //some logic }

Зачем нужна аннотация @ResponseStatus?

Она позволяет устанавливать код ответа. Обычно Spring сам устанавливает нужный код ответа, но бывают моменты, когда это нужно переопределить.

@PostMapping @ResponseStatus(HttpStatus.CREATED) public void add(...) {...}

Вместо использования аннотации можно возвращать ResponseEntity и вручную устанавливать код ответа.

Не рекомендуется использовать ResponseEntity и @ReponseStatus вместе.

Что такое ResponseEntity?

Это специальный класс, который представляет http-ответ. Он содержит тело ответа, код состояния, заголовки. Мы можем использовать его для более тонкой настройки http-ответа.

Он является универсальным типом, и можно использовать любой объект в качестве тела:

@GetMapping(«/hello») ResponseEntity hello() { return new ResponseEntity(«Hello World!», HttpStatus.OK); }

Расскажите о Spring Framework.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Несмотря на то, что Spring Framework не обеспечивает какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring Framework предоставляет бóльшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании

приложений корпоративного масштаба. Обычно Spring описывают как облегченную платформу для построения Java-приложений, но с этим утверждением связаны два интересных момента. Во-первых, Spring можно использовать для построения любого приложения на языке Java (т.е. автономных, веб приложений, приложений JEE и т.д.), что отличает Spring от многих других платформ, таких как Apache Struts, которая ограничена только веб-приложениями. Во-вторых, характеристика «облегченная» в действительности не имеет никакого отношения к количеству классов или размеру дистрибутива; напротив, она определяет принцип всей философии Spring — минимальное воздействие. Платформа Spring является облегченной в том смысле, что для использования ядра Spring вы должны вносить минимальные (если вообще какие-либо) изменения в код своего приложения, а если в какой-то момент вы решите больше не пользоваться Spring, то и это сделать очень просто. Обратите внимание, что речь идет только о ядре Spring — многие дополнительные компоненты Spring, такие как доступ кданным, требуют более тесной привязки к Spring Framework.

Какие некоторые из важных особенностей и преимуществ Spring Framework?

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring Framework не всецело связан с платформой Java Enterprise, несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

Spring Framework, вероятно, наиболее известен как источник расширений (features), нужных для эффективной разработки сложных бизнес-приложений вне тяжеловесных программных моделей, которые исторически были доминирующими в промышленности. Ещё одно его достоинство в том, что он ввел ранее неиспользуемые функциональные возможности в сегодняшние господствующие методы разработки, даже вне платформы Java. Этот фреймворк предлагает последовательную модель и делает её применимой к большинству типов приложений, которые уже созданы на основе платформы Java. Считается, что Spring Framework реализует модель разработки, основанную на лучших стандартах индустрии, и делает её доступной во многих областях Java. Таким образом к достоинствам Spring можно отнести:

Относительная легкость в изучении и применении фреймворка в разработке и поддержке приложения.

Внедрение зависимостей (DI) и инверсия управления (IoC) позволяют писать независимые друг от друга компоненты, что дает преимущества в командной разработке, переносимости модулей и т.д..

Spring IoC контейнер управляет жизненным циклом Spring Bean и настраивается наподобие JNDI lookup (поиска).

Проект Spring содержит в себе множество подпроектов, которые затрагивают важные части создания софта, такие как вебсервисы, веб программирование, работа с базами данных, загрузка файлов, обработка ошибок и многое другое. Всё это настраивается в едином формате и упрощает поддержку приложения.

Объясните суть паттерна DI или IoC.

Dependency injection (DI) - паттерн проектирования и архитектурная модель, так же известная как Inversion of Control (IoC). DI описывает ситуацию, когда один объект реализует свой функционал через другой объект. Например, соединение с базой данных передается конструктору объекта через аргумент, вместо того чтобы конструктор сам устанавливал соединение. Существуют три формы внедрения (но не типа) зависимостей: сэттер, конструктор и внедрение путем интерфейса.

DI - это способ достижения слабой связанности. IoC предоставляет возможность объекту получать ссылки на свои зависимости. Обычно это реализуется через lookup-метод. Преимущество IoC в том, что эта модель позволяет отделить объекты от реализации механизмов, которые он использует. В результате мы получаем большую гибкость как при разработке приложений, так и при их тестировании.

Какие преимущества применения Dependency Injection (DI)?

К преимуществам DI можно отнести:

Сокращение объема связующего кода. Одним из самых больших плюсов DI является возможность значительного сокращения объема кода, который должен быть написан для связывания вместе различных компонентов приложения. Зачастую этот код очень прост - при создании зависимости должен создаваться новый экземпляр соответствующего объекта.

Упрощенная конфигурация приложения. За счет применения DI процесс конфигурирования приложения значительно упрощается. Для конфигурирования классов, которые могут быть внедрены в другие классы, можно использовать аннотации или XML-файлы.

Возможность управления общими зависимостями в единственном репозитории. При традиционном подходе к управлению зависимостями в общих службах, к которым относятся, например, подключение к источнику данных, транзакция, удаленные службы и т.п., вы создаете экземпляры (или получаете их из определенных фабричных классов) зависимостей там, где они нужны - внутри зависимого класса. Это приводит к распространению зависимостей по множеству классов в приложении, что может затруднить их изменение. В случае использования DI вся информация об общих зависимостях содержится в единственном репозитории (в Spring есть возможность хранить эту информацию в XML-файлах или Java классах).

Улучшенная возможность тестирования. Когда классы проектируются для DI, становится возможной простая замена зависимостей. Это особенно полезно при тестировании приложения.

Стимулирование качественных проектных решений для приложений. Вообще говоря, проектирование для DI означает проектирование с использованием интерфейсов. Используя Spring, вы получаете в свое распоряжение целый ряд средств DI и можете сосредоточиться на построении логики приложения, а не на поддерживающей DI платформе.

Какие IoC контейнеры вы знаете?

Spring является IoC контейнером. Помимо него существуют HiveMind, Avalon, PicoContainer и т.д.

Как реализуется DI в Spring Framework?

Внедрение зависимостей (DI) - это концепция, которая определяет, как должно быть связано несколько классов. Это один из примеров Инверсии контроля. Вам не нужно явно подключать службы и компоненты в коде при использовании внедрения зависимостей. Вместо этого вы описываете службы, необходимые каждому компоненту, в файле конфигурации XML и разрешаете контейнеру IOC автоматически подключать их.

Реализация DI в Spring основана на двух ключевых концепциях Java - компонентах JavaBean и интерфейсах. При использовании Spring в качестве поставщика DI вы получаете гибкость определения конфигурации зависимостей внутри своих приложений разнообразными путями (т.е. внешне в XML-файлах, с помощью конфигурационных Java классов Spring или посредством аннотаций Java в коде). Компоненты JavaBean (также называемые POJO (Plain Old Java Object — простой старый объект Java)) предоставляют стандартный механизм для создания ресурсов Java, которые являются конфигурируемыми множеством способов. За счет применения DI объем кода, который необходим при проектировании приложения на основе интерфейсов, снижается почти до нуля. Кроме того, с помощью интерфейсов можно получить макси-

мальную отдачу от DI, потому что бины могут использовать любую реализацию интерфейса для удовлетворения их зависимости.ff

Какие существуют виды DI? Приведите примеры.

Существует два типа DI: через сэттер и через конструктор.

Через сэттер: обычно во всех java beans используются гэттеры и сэттеры для их свойств:

```
public class NameBean {
String name;
public void setName(String a) {
name = a;
}
public String getName() {
return name;}
}
```

Мы создаем экземпляр бина NameBean (например, bean1) и устанавливаем нужное свойство, например:

```
bean1.setName(«Marfa»);
```

Используя Spring реализация была бы такой:

```
<bean id=»bean1» class=»NameBean»>
<property name=»name»>
<value>Marfa</value>
</property>
</bean>
```

Это и называет DI через сэттер. Пример внедрения зависимости между объектами:

```
<bean id=»bean1» class=»bean1impl»>
<property name=»game»>
<ref bean=»bean2» />
</property>
</bean>
<bean id=»bean2» class=»bean2impl» />
```

Через конструктор: используется конструктор с параметрами. Например:

```
public class NameBean {
String name;
public NameBean(String name) {
this. name = name;
}}
```

Теперь мы внедряем объект на этапе создания экземпляра класса, т.е.

```
bean1 = new NameBean(«Marfa»);
```

Используя Spring это выглядело бы так:

```
<bean id=»bean1» class=»NameBean»>
<constructor-arg>
<value>Marfa</value>
</constructor-arg>
</bean>
```

Что такое Spring? Из каких частей состоит Spring Framework?

Spring - фреймворк с открытым исходным кодом, предназначеный для упрощения разработки enterprise-приложений. Одним из главным преимуществом Spring является его слоистая архитектура, позволяющая вам самим определять какие компоненты будут использованы в вашем приложении. Модули Spring построены на базе основного контейнера, который определяет создание, конфигурация и менеджмент бинов.

Основные модули:

Основной контейнер - предоставляет основной функционал Spring. Главным компонентом контейнера является BeanFactory - реализация паттерна Фабрика. BeanFactory позволяет разделить конфигурацию приложения и информацию о зависимостях от кода.

Spring context - конфигурационный файл, который предоставляет информация об окружающей среде для Spring. Сюда входят такие enterprise-сервисы, как JNDI, EJB, интернационализация, валиадция и т.п.

Spring AOP - отвечает за интеграцию аспектно-ориентированного программирования во фреймворк. Spring AOP обеспечивает сервис управления транзакциями для Spring-приложения.

Spring DAO - абстрактный уровень Spring JDBC DAO предоставляет иерархию исключений и множество сообщений об ошибках для разных БД. Эта иерархия упрощает обработку исключений и значительно уменьшает количество кода, которое вам нужно было бы написать для таких операций, как, например, открытие и закрытие соединения.

Spring ORM - отвечает за интеграцию Spring и таких популярных ORM-фреймворков, как Hibernate, iBatis и JDO.

Spring Web module - классы, которые помогают упростить разработку Web (авторизация, доступ к бинам Spring-а из web).

Spring MVC framework - реализация патерна MVC для построения Web-приложений.

Назовите некоторые из шаблонов проектирования, используемых в Spring Framework?

Spring Framework использует множество шаблонов проектирования, например:

Singleton Pattern: Creating beans with default scope.

Factory Pattern: Bean Factory classes

Prototype Pattern: Bean scopes

Adapter Pattern: Spring Web and Spring MVC

Proxy Pattern: Spring Aspect Oriented Programming support

Template Method Pattern: JdbcTemplate, HibernateTemplate etc

Front Controller: Spring MVC DispatcherServlet

Data Access Object: Spring DAO support

Dependency Injection and Aspect Oriented Programming

Каковы некоторые из важных особенностей и преимуществ Spring Framework?

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring Framework не всецело связан с платформой Java Enterprise, несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

Относительная легкость в изучении и применении фреймворка в разработке и поддержке приложения.

Внедрение зависимостей (DI) и инверсия

управления (IoC) позволяют писать независимые друг от друга компоненты, что дает преимущества в командной разработке, переносимости модулей и т.д..

Spring IoC контейнер управляет жизненным циклом Spring Bean и настраивается наподобие JNDI lookup (поиска).

Проект Spring содержит в себе множество подпроектов, которые затрагивают важные части создания софта, такие как вебсервисы, веб программирование, работа с базами данных, загрузка файлов, обработка ошибок и многое другое. Всё это настраивается в едином формате и упрощает поддержку приложения.

Каковы преимущества использования Spring Tool Suite?

Для упрощения процесса разработки основанных на Spring приложений в Eclipse (наиболее часто используемая IDE-среда для разработки Java-приложений), в рамках Spring создан проект Spring IDE. Проект бесплатный. Он интегрирован в Eclipse IDE, Spring IDE, Mylyn (среда разработки в Eclipse, основанная на задачах), Maven for Eclipse, AspectJ Development Tool.

Что такое AOP? Как это относиться к IoC?

Аспектно-ориентированное программирование (АОП) - парадигма программи-

рования, основанная на идее разделения функциональности для улучшения разбиения программы на модули. AOP и Spring - взаимодополняющие технологии, которые позволяют решать сложные проблемы путем разделения функционала на отдельные модули. АОП предоставляет возможность реализации сквозной логики - т.е. логики, которая применяется к множеству частей приложения - в одном месте и обеспечения автоматического применения этой логики по всему приложению. Подход Spring к АОП заключается в создании «динамических прокси» для целевых объектов и «привязывании» объектов к конфигурированному совету для выполнения сквозной логики.

В чем разница между Сквозной Функциональностью (Cross Cutting Concerns) и АОП (аспектно оринтированное программирование)?

Сквозная Функциональность — функциональность, которая может потребоваться вам на нескольких различных уровнях — логирование, управление производительностью, безопасность и т.д.

АОП — один из подходов к реализации данной проблемы

Почему возвращаемое значение при применении аспекта @Around может потеряться? Назовите причины.

Метод, помеченный аннотацией @ Around, должен возвращать значение, которое он (метод) получил из joinpoint. proceed()

@Around(«trackTimeAnnotation()»)

public Object around(ProceedingJoinPoint joinPoint) throws Throwable{ long startTime = System.currentTimeMillis();

Object retVal = joinPoint.proceed();

long timeTaken = System.currentTimeMillis() - startTime;

logger.info(«Time taken by {} is equal to {}»,joinPoint, timeTaken);

return retVal; }

Что такое Aspect, Advice, Pointcut, JointPoint и Advice Arguments в АОП?

Основные понятия АОП:

Аспект (англ. aspect) - модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.

Совет (англ. advice) - фрагмент кода, который должен выполняться в отдельной точке соединения. Существует несколько типов советов, совет может быть выполнен до, после или вместо точки соединения.

Точка соединения (англ. joinpoint) - это четко определенная точка в выполняемой программе, где следует применить совет. Типовые примеры точек соединения включают обращение к методу, собственно Method Invocation, инициализацию класса и создание экземпляра объекта. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.

Срез (англ. pointcut) - набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в AspectJ применяются Java-сигнатуры) и позволяют их повторное использование с помощью переименования и комбинирования.

Связывание(англ. weaving) представляет собой процесс действительной вставки аспектов в определенную точку кода приложения. Для решений АОП времени компиляции это делается на этапе компиляции, обычно в виде дополнительного шага процесса сборки. Аналогично, для решений АОП времени выполнения связывание происходит динамически во время выполнения. В AspectJ поддерживается еще один механизм связывания под названием связывание во время загрузки (load-time weaving - LTW), который перехватывает лежащий в основе загрузчик классов JVM и обеспечивает связывание с байт-кодом, когда он загружается загрузчиком классов.

Цель(англ. target) - это объект, поток выполнения которого изменяется каким-то процессом АОП. На целевой объект часто ссылаются как на объект, снабженный советом.

Внедрение (англ. introduction, введение) - представляет собой процесс, посредством которого можно изменить структуру объекта за счет введения в него дополнительных методов или полей, изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (англ. metaobject protocol, МОР).

В чем разница между Spring AOP и AspectJ АОП?

AspectJ де-факто является стандартом реализации АОП. Реализация АОП от Spring имеет некоторые отличия:

Spring AOP немного проще, т.к. нет необходимости следить за процессом связывания.

Spring AOP поддерживает аннотации AspectJ, таким образом мы можем работать в спринг проекте похожим образом с AspectJ проектом.

Spring AOP поддерживает только proxy-based АОП и может использовать только один тип точек соединения - Method Invocation. AspectJ поддерживает все виды точек соединения.

Недостатком Spring AOP является работа только со своими бинами, которые существуют в Spring Context.

Что такое Advice в Spring?

Advice - это действие, предпринятое в данной точке соединения. АОП использует Advice в качестве перехватчика до завершения выполнения метода.

Каковы типы рекомендаций для структуры Spring?

До: Это советы, которые выполняются до методов joinpoint. Они помечены знаком @before.

После возврата: они выполняются после того, как метод joinpoint завершит выполнение без проблем. Они помечены знаком аннотации @AfterReturning.

После выполнения: Они выполняются только в том случае, если метод joinnpoint заканчивается созданием исключения. Они помечены с помощью метки аннотации @AfterThrowing.

После: Они выполняются после метода joinpoint, независимо от того, как он завершается. Они помечены знаком @ After.

Вокруг: Они выполняются до и после точки соединения и помечаются с помощью метки @Around аннотации.

Что такое Weaving?

Weaving Spring - это процесс связывания элементов с другими типами приложений или объектами для создания рекомендуемых объектов.

Что такое прокси-объекты и какие типы прокси-объектов может создавать Spring?

Прокси это специальный объект, который имеет такие же публичные методы как и бин, но у которого есть дополнительная функциональность.

Два вида прокси:

JDK-proxy — динамическое прокси. API встроены в JDK. Для него необходим интерфейс

CGLib proxy — не встроен в JDK. Используется когда интерфейс объекта недоступен

Плюсы прокси-объектов:

Позволяют добавлять доп. логику — управление транзакциями, безопасность, логирование

Отделяет некоторый код(логирование и т.п.) от основной логики

Что такое IoC контейнер Spring?

По своей сути IoC, а, следовательно, и DI, направлены на то, чтобы предложить простой механизм для предоставления зависимостей компонента (часто называемых коллаборатором объекта) и управления этими зависимостями на протяжении всего их жизненного цикла. Компонент, который требует определенных зависимостей, зачастую называют зависимым объектом или, в случае IoC, целевым объектом. IoC предоставляет службы, через которые компоненты могут получать доступ к своим зависимостям, и службы для взаимодействия с зависимостями в течение их времени жизни. В общем случае IoC может быть расщеплена на два подтипа: инверсия управления (Dependency Injection) и инверсия поиска (Dependency Lookup). Инверсия управления — это крупная часть того, делает Spring, и ядро реализации Spring основано на инверсии управления, хотя также предоставляются и средства Dependency Lookup. Когда платформа Spring предоставляет коллабораторы зависимому объекту автоматически, она делает это с использованием инверсии управления (Dependency Injection). В приложении, основанном на Spring, всегда предпочтительнее применять Dependency Injection для передачи коллабораторов зависимым объектам вместо того, чтобы заставлять зависимые объекты получать коллабораторы через поиск.

Что такое контейнер и какой у него жизненный цикл?

Основа Spring Framework — контейнер, и наши объекты «живут» в этом контейнере.Контейнер обычно создает множество объектов на основе их конфигураций и управляет их жизненным циклом от создания объекта до уничтожения.

Контейнер — это объект, реализующий интерфейс ApplicationContext.

Жизненный цикл контейнера

Контейнер создается при запуске приложения

Контейнер считывает конфигурационные данные

Из конфигурационных данных создается описание бинов

BeanFactoryPostProcessors обрабатывают описание бина

Контейнер создает бины используя их описание

Бины инициализируются — значения свойств и зависимости внедряются в бин

BeanPostProcessor запускают методы обратного вызова(callback methods)

Приложение запущено и работает

Инициализируется закрытие приложения

Контейнер закрывается

Вызываются callback methods

Что такое Spring бин?

Термин бин (англ. Bean) - в Spring используется для ссылки на любой компонент, управляемый контейнером. Обычно бины на определенном уровне придерживаются спецификации JavaBean, но это не обязательно особенно если для связывания бинов друг с другом планируется применять Constructor Injection. Для получения экземпляра бина используется ApplicationContext. IoC контейнер управляет жизненным циклом спринг бина, областью видимости и внедрением.

Какое значение имеет конфигурационный файл Spring Bean?

Конфигурационный файл спринг определяет все бины, которые будут инициализированы в Spring Context. При создании экземпляра Spring ApplicationContext будет прочитан конфигурационный xml файл и выполнены указанные в нем необходимые инициализации. Отдельно от базовой конфигурации, в файле могут содержаться описание перехватчиков (interceptors), view resolvers, настройки локализации и др...

Каковы различные способы настроить класс как Spring Bean?

Существует несколько способов работы с классами в Spring: XML конфигурация:

Java based конфигурация. Все настройки и указания бинов прописываются в java коде:

Annotation based конфигурация. Можно использовать внутри кода аннотации @Component, @Service, @Repository, @Controller для указания классов в качестве спринг бинов. Для их поиска и управления контейнером прописывается настройка в xml файле:

Какие вы знаете различные scope у Spring Bean?

В Spring предусмотрены различные области времени действия бинов:

singleton - может быть создан только один экземпляр бина. Этот тип используется спрингом по умолчанию, если не указано другое. Следует осторожно использовать публичные свойства класса, т.к. они не будут потокобезопасными.

prototype - создается новый экземпляр при каждом запросе.

request - аналогичен prototype, но название служит пояснением к использованию бина в веб приложении. Создается новый экземпляр при каждом HTTP request.

session - новый бин создается в контейнере при каждой новой HTTP сессии.

Application Область видимости — жизненный цикл ServletContext

WebSocket Область видимости — жизненный цикл WebSocket

Как связаны различные скоупы и многопоточность?

Prototype Scope не потокбезопасный, т.к. он не гарантирует что один и тот же экземпляр будет вызываться только в 1 потоке.

Singleton Scope же наоборот потокобезопасный.

Как создаются бины: сразу или лениво? Как изменить это поведение?

Singleton-бины обычно создаются сразу при сканировании.Prototype-бины обычно создаются только после запроса.

Чтобы указать способ инициализации, можно использовать аннотацию @Lazy. Она ставится на @Bean-методы, на @Configuration-классы, или на @Component-классы.В зависимости от параметра(true или false), который принимает аннотация, инициализация будет или ленивая, или произойдет сразу. По умолчанию(т.е. без указания параметра) используется true.

Что будет если бин с одним скоупом внедрить в бин с другим скоупом?

Singleton bean можно внедрять в любой другой бин.

В сам singleton можно внедрить только prototype или singleton.Если внедрять prototype, то для каждого singleton будет создан уникальный prototype.

Prototype может быть зависимостью для любого бина.Внедрять можно только singleton или prototype.

Как работает инъекция прототипа в синглтон?

Раньше мы уже рассматривали различия скоупов singleton и prototype в Spring Framework. Допустим ситуацию, когда в singleton-компонент внедряется зависимость со скоупом prototype - когда будет создан её объект?Если просто добавить к определению бина аннотацию @Scope(SCOPE_PROTOTYPE), и использовать этот бин в синглтоне через аннотацию @Autowired - будет создан только один объект. Потому что синглтон создается только однажды, и обращение к прототипу случится тоже однажды при его создании (при внедрении зависимости).Примитивный способ получать новый объект при каждом обращении - отказаться от @Autowired, и доставать его из контекста вручную. Для этого нужно вызывать context.getBean(MyPrototype.class).Воспользоваться автоматическим внедрением зависимостей можно через внедрение метода (паттерн «Команда»). Автовайрится не сам объект, а производящий его метод.Более красивый декларативный способ - правильно настро-

ить определение бина. В аннотации @Scope кроме самого scopeName доступен второй параметр - proxyMode. По умолчанию его значение NO - прокси не создается. Но если указать INTERFACES или TARGET_CLASS, то под @Autowired будет внедряться не сам объект, а сгенерированный фреймворком прокси. И когда проксируемый бин имеет скоуп prototype, то объект внутри прокси будет пересоздаваться при каждом обращении.

Что такое жизненный цикл Spring Bean?

Beans - центральный объект заботы Spring Framework. За кулисами фреймворка с ними происходит множество процессов. Во многие из них можно вмешаться, добавив собственную логику в разные этапы жизненного цикла. Через следующие этапы проходит каждый отдельно взятый бин:

1. Инстанцирование объекта. Техническое начало жизни бина, работа конструктора его класса;

2. Установка свойств из конфигурации бина, внедрение зависимостей;

3. Нотификация aware-интерфейсов. BeanNameAware, BeanFactoryAware и другие. Мы уже писали о таких интерфейсах ранее. Технически, выполняется системными подтипами BeanPostProcessor, и совпадает с шагом 4;

4. Пре-инициализация - метод postProcessBeforeInitialization() интерфейса BeanPostProcessor;

5. Инициализация. Разные способы применяются в таком порядке:

• Метод бина с аннотацией @PostConstruct из стандарта JSR-250 (рекомендуемый способ);

• Метод afterPropertiesSet() бина под интерфейсом InitializingBean;

• Init-метод. Для отдельного бина его имя устанавливается в параметре определения initMethod. В xml-конфигурации можно установить для всех бинов сразу, с помощью default-init-method;

6. Пост-инициализация - метод postProcessAfterInitialization() интерфейса BeanPostProcessor.

Когда IoC-контейнер завершает свою работу, мы можем кастомизировать этап штатного уничтожения бина. Как со всеми способами финализации в Java, при жестком выключении (kill -9) гарантии вызова этого этапа нет. Три альтернативных способа «деинициализации» вызываются в том же порядке, что симметричные им методы инициализации:

1. Метод с аннотацией @PreDestroy;

2. Метод с именем, которое указано в свойстве destroyMethod определния бина (или в глобальном default-destroy-method);

3. Метод destroy() интерфейса DisposableBean.Не следует путать жизненный цикл отдельного бина с жизненным циклом контекста и этапами подготовки фабрик бинов. О них мы поговорим в будущих публикациях.

Объясните работу BeanFactory в Spring.

BeanFactory - это реализация паттерна Фабрика, его функицональность покрывает создание бинов. Так как эта фабрика знает многие об объектах приложения, то она может создавать связи между объектами на этапе создания экземпляра. Существует несколько реализаций BeanFactory, самая используемся - «org.springframework.beans.factory.xml.XmlBeanFactory». Она загружает бины на основе конфигурационного XML-файла. Чтобы создать XmlBeanFactory передайте конструктору InputStream, например:

BeanFactory factory = new XmlBeanFactory(new FileInputStream(«myBean.xml»));

После этой строки фабрика знает о бинах, но их экземпляры еще не созданы. Чтобы инстанцировать бин нужно вызвать метод getBean(). Например:

myBean bean1 = (myBean)factory.getBean(«myBean»);

В чем разница между BeanFactory и ApplicationContext?

BeanFactory - это базовый, компактный контейнер с ограниченной функциональностью. Его лучше всего использовать для простых задач или при использовании машин с низким ресурсом.

ApplicationContext - это расширенный, более интенсивный контейнер с расширенным интерфейсом и дополнительными возможностями, такими как AOP. Этот контейнер лучше всего использовать, когда вам требуется больше функциональности, чем на заводе Bean, и у вас достаточно ресурсов, доступных на машине.

Как получить объекты ServletContext и ServletConfig внутри Spring Bean?

Доступны два способа для получения основных объектов контейнера внутри бина:

Реализовать один из Spring*Aware (ApplicationContextAware, ServletContextAware, ServletConfigAware и др.) интерфейсов.

Использовать автоматическое связывание @Autowired в спринг. Способ работает внутри контейнера спринг.

В чем роль ApplicationContext в Spring?

В то время, как BeanFactory используется в простых приложениях, Application Context - это более сложный контейнер. Как и BeanFactory он может быть использован для загрузки и связывания бинов, но еще он предоставляет:

возможность получения текстовых сообщений, в том числе поддержку интернационализации;

общий механизм работы с ресурсами;

события для бинов, которые зарегестрированы как слушатели.

Из-за большей функциональности рекомендуется использование Application Context вместо BeanFactory. Последний используется только в случаях нехватки ресурсов, например при разработке для мобильных устройств

Как получить ApplicationContext в интеграционном тесте?

Если вы используете JUnit 5, то вам нужно указать 2 аннотации:

@ExtendWith(TestClass.class) — используется для указания тестового класса

@ContextConfoguration(classes = JavaConfig.class) — загружает java/xml конфигурацию для создания контекста в тесте

Можно использовать аннотацию @SpringJUnitConfig, которая сочетает обе эти аннотации.Для теста веб-слоя можно использовать аннотацию @SpringJUnitWebConfig.

Как завершить работу контекста в приложении?

Если это не веб-приложение, то есть 2 способа:

Регистрация shutdown-hook с помощью вызова метода registerShutdownHook(), он также реализован в классе AbstractApplicationContext. Это предпочтительный способ.

Можно вызвать метод close() из класса AbstractApplicationContext.

В Spring Boot приложении:

Spring Boot самостоятельно зарегистрирует shutdown-hook за вас.

Для чего нужен Component Scan?

Если вы понимаете как работает Component Scan, то вы понимаете Spring

Первый шаг для описания Spring Beans это добавление аннотации — @Component, или @Service, или @Repository.

Однако, Spring ничего не знает об этих бинах, если он не знает где искать их. То, что скажет Spring где искать эти бины и называется Component Scan. В @ComponentScan вы указываете пакеты, которые должны сканироваться.

Spring будет искать бины не только в па-

кетах для сканирования, но и в их подпакетах.

Как вы добавите Component Scan в Spring Boot?

@SpringBootApplication public class Application { public static void main(String[] args) { SpringApplication.run(Application.class, args); } }

@SpringBootApplication определяет автоматическое сканирование пакета, где находится класс Application

Всё будет в порядке, ваш код целиком находится в указанном пакете или его подпакетах.

Однако, если необходимый вам компонент находится в другом пакете, вы должны использовать дополнительно аннотацию @ComponentScan, где перечислите все дополнительные пакеты для сканирования

В чём отличие между @Component и @ComponentScan?

@Component и @ComponentScan предназначены для разных целей

@Component помечает класс в качестве кандидата для создания Spring бина.@ ComponentScan указывает где Spring искать классы, помеченные аннотацией @ Component или его производной

Для чего используется аннотация @ Bean?

В классах конфигурации Spring, @Bean используется для определения компонентов с кастомной логикой.

Почему для создания Spring beans рекомендуются интерфейсы?

Улучшенное тестирование. В тестах бин может быть заменен специальным объектом(mock или stub), который реализует интерфейс бина.

Позволяет использовать механизм динамических прокси из JDK(например, при создании репозитория через Spring Data)

Позволяет скрывать реализацию

Как внедряется singleton-бин?

Если в контейнере нет экземпляра бина, то вызывается @Bean-метод. Если экземпляр бина есть, то возвращается уже созданный бин.

В чём разница между @Bean и @ Component?

@Bean используется в конфигурационных классах Spring. Он используется для непосредственного создания бина.

@Component используется со всеми классами, которыми должен управлять Spring. Когда Spring видит класс с @ Component, Spring определяет этот класс как кандидата для создания bean.

Как выглядит типичная реализция метода используя Spring?

Для типичного Spring-приложения нам необходимы следующие файлы:

Интерфейс, описывающий функционал приложения

Реализация интерфейса, содержащая свойства, сэттеры-гэттеры, функции и т.п.

Конфигурационный XML-файл Spring'а.

Клиентское приложение, которое использует функцию.

Можем ли мы применить @Autowired с не сеттерами и не конструкторами методами?

Да, конечно.

@Autowired может использоваться вместе с конструкторами, сеттерами или любым другими методами. Когда Spring находит @Autowired на методе, Spring автоматически вызовет этот метод, после создания экземпляра бина. В качестве аргументов, будут подобраны подходящие объекты из контекста Spring.

Как внедрить простые значения в свойства в Spring?

Для этого можно использовать аннотацию @Value.Такие значения можно получать из property файлов, из бинов, и т.п.

@Value(«$some.key») public String stringWithDefaultValue;

В эту переменную будет внедрена строка, например из property или из view.

Как вы решаете какой бин инжектить, если у вас несколько подходящих бинов. Расскажите о @Primary и @Qualifier?

Если есть бин, который вы предпочитаете большую часть времени по сравнению с другими, то используйте @Primary, и используйте @Qualifier для нестандартных сценариев.

Если все бины имеют одинаковый приоритет, мы всегда будем использовать @ Qualifier

Если бин надо выбрать во время исполнения программы, то эти аннотации вам не подойдут. Вам надо в конфигурационном классе создать метод, пометить его аннотацией @Bean, и вернуть им требуемый бин.

Как произвести DI в private поле?

Вы можете использовать разные типы внедрения:

Конструктор

Сеттер

Field-injection

Value

Что такое связывание в Spring и расскажите об аннотации @Autowired?

Процесс внедрения зависимостей в бины при инициализации называется Spring Bean Wiring. Считается хорошей практикой задавать явные связи между зависимостями, но в Spring предусмотрен дополнительный механизм связывания @Autowired. Аннотация может использоваться над полем или методом для связывания по типу. Чтобы аннотация заработала, необходимо указать небольшие настройки в конфигурационном файле спринг с помощью элемента context:annotation-config.

Опишите поведение аннотации @ Autowired

1) Контейнер определяет тип объекта для внедрения

2) Контейнер ищет бины в контексте(он же контейнер), которые соответствуют нужному типу

3) Если есть несколько кандидатов, и один из них помечен как @Primary, то внедряется он

4) Если используется аннотации @ Autowire + Qualifier, то контейнер будет использовать информацию из @Qualifier, чтобы понять, какой компонент внедрять

5) В противном случае контейнер попытается внедрить компонент, основываясь на его имени или ID

6) Если ни один из способов не сработал, то будет выброшено исключение

Контейнер обрабатывает DI с помощью AutowiredAnnotationBeanPostProcessor. В связи с этим, аннотация не может быть использована ни в одном BeanFactoryPP или BeanPP.

Если внедряемый объект массив, коллекция, или map с дженериком, то Spring внедрит все бины подходящие по типу в этот массив(или другую структуру данных). В случае с map ключом будет имя бина.

//параметр указывает, требуется ли DI @ Authowired(required = true/false)

Как можно использовать аннотацию @Autowire и в чем отличие между способами?

Ниже перечислены типы DI, которые могут быть использованы в вашем приложении:

Constructor DI

Setter DI

Field DI

DI через конструктор считается самым лучшим способом, т.к. для него не надо использовать рефлексию, а также он не имеет недостатков DI через сеттер.DI

через поле не рекомендуется использовать, т.к. для этого применяется рефлексия, снижающая производительность. DI через конструктор может приводить к циклическим зависимостям. Чтобы этого избежать, можно использовать ленивую инициализацию бинов или DI через сеттер.

Аннотация Qualifier

Данная аннотация позволяет несколько специфицировать бин, который необходим для @Autowired. @Qualifier принимает один входной параметр — имя бина.

@Autowired@Qualifier(«specialTestBean»)

private TestBean bean;

* This source code was highlighted with Source Code Highlighter.

Эта конструкция будет искать в контексте бин с именем specialTestBean и в нашем примере мы соответственно получим исключение, так как TestBean объявлен с именем 'testBean' (@Service(«testBean»)). На основе @Qualifier можно создавать свои признаки бинов, об этом достаточно хорошо написано (и, что немаловажно, с огромным количеством примеров) в Spring Reference Manual.

Каковы различные типы автоматического связывания в Spring?

Существует четыре вида связывания в спринг:

autowire byName,

autowire byType,

autowire by constructor,

autowiring by @Autowired and @Qualifier annotations

Приведите пример часто используемых аннотаций Spring.

@Controller - класс фронт контроллера в проекте Spring MVC.

@RequestMapping - позволяет задать шаблон маппинга URI в методе обработчике контроллера.

@ResponseBody - позволяет отправлять Object в ответе. Обычно используется для отправки данных формата XML или JSON.

@PathVariable - задает динамический маппинг значений из URI внутри аргументов метода обработчика.

@Autowired - используется для автоматического связывания зависимостей в spring beans.

@Qualifier - используется совместно с @Autowired для уточнения данных связывания, когда возможны коллизии (например одинаковых имен\типов).

@Service - указывает что класс осуществляет сервисные функции.

@Scope - указывает scope у spring bean.

@Configuration, @ComponentScan и @Bean - для java based configurations.

AspectJ аннотации для настройки aspects и advices, @Aspect, @Before, @After,@Around, @Pointcut и др.

Что такое профили? Какие у них причины использования?

При использовании Java-конфигурации вы можете использовать аннотацию @Profile.Она позволяет использовать разные настройки для Spring в зависимости от указанного профиля.Ее можно ставить на @Configuration и Component классы, а также на Bean методы.

Profile(«!test») //загружать со всеми прифилями, кроме теста

@Bean(«dataSource») @Profile(«production») public DataSource jndiDataSource() {...} @Bean(«dataSource») @Profile(«development») public DataSource standaloneDataSource() {...}

Можем ли мы послать объект как ответ метода обработчика контроллера?

Да, это возможно. Для этого используется аннотация @ResponseBody. Так можно отправлять ответы в виде JSON, XML в restful веб сервисах.

Является ли Spring bean потокобезопасным?

По умолчанию бин задается как синглтон в Spring. Таким образом все публичные переменные класса могут быть изменены одновременно из разных мест. Так что - нет, не является. Однако поменяв область действия бина на request, prototype, session он станет потокобезопасным, но это скажется на производительности.

Почему иногда мы используем @ResponseBody, а иногда ResponseEntity?

ResponseEntity необходим, только если мы хотим кастомизировать ответ, добавив к нему статус ответа. Во всех остальных случаях будем использовать @ResponseBody.

@GetMapping(value=»/resource») @ResponseBody public Resource sayHello() { return resource; }

@PostMapping(value=»/resource») public ResponseEntity createResource() { .... return ResponseEntity.created(resource).build(); }

Стандартные HTTP коды статусов ответов, которые можно использовать.

200 — SUCCESS

201 — CREATED

404 — RESOURCE NOT FOUND

400 — BAD REQUEST

401 — UNAUTHORIZED5

00 — SERVER ERROR

Для @ResponseBody единственные состояния статуса это SUCCESS(200), если всё ок и SERVER ERROR(500), если произошла какая-либо ошибка.

Допустим мы что-то создали и хотим отправить статус CREATED(201). В этом случае мы используем ResponseEntity.

Как создать ApplicationContext в программе Java?

В независимой Java программе ApplicationContext можно создать следующим образом:

AnnotationConfigApplicationContext - при использовании Spring в качестве автономного приложения можно создать инициализировать контейнер с помощью аннотаций.

ClassPathXmlApplicationContext - получает информацию из xml-файла, находящегося в classpath.

FileSystemXmlApplicationContext - получает информацию из xml-файла, но с возможностью загрузки файла конфигурации из любого места файловой системы.

XmlWebApplicationContext - получает информацию из xml-файла за пределами web-приложения.

Можем ли мы иметь несколько файлов конфигурации Spring?

Спомощьюуказанияcontext ConfigLocation можно задать несколько файлов конфигурации Spring. Параметры указываются через запятую или пробел:

Поддерживается возможность указания нескольких корневых файлов конфигурации Spring:

Файл конфигурации можно импортировать:

Как внедрить java.util.Properties в Spring Bean?

Для возможности использования Spring EL для внедрения свойств (properties) в различные бины необходимо определить propertyConfigure bean, который будет загружать файл свойств.

Или через аннотации @Value

Как настраивается соединение с БД в Spring?

Используя datasource «org.springframework.jdbc.datasource.DriverManagerDataSource».

Как сконфигурировать JNDI не через datasource в applicationContext.xml?

Используя «org.springframework.jndi.

JndiObjectFactoryBean»

Каким образом можно управлять транзакциями в Spring?

Транзакциями в Spring управляют с помощью Declarative Transaction Management (программное управление). Используется аннотация @Transactional для описания необходимости управления транзакцией. В файле конфигурации нужно добавить настройку transactionManager для DataSource.

Каким образом Spring поддерживает DAO?

Spring DAO предоставляет возможность работы с доступом к данным с помощью технологий вроде JDBC, Hibernate в удобном виде. Существуют специальные классы: JdbcDaoSupport, HibernateDaoSupport, JdoDaoSupport, JpaDaoSupport.

Класс HibernateDaoSupport является подходящим суперклассом для Hibernate DAO. Он содержит методы для получения сессии или фабрики сессий. Самый популярный метод - getHibernateTemplate(), который возвращает HibernateTemplate. Этот темплейт оборачивает checked-исключения Hibernate в runtime-исключения, позволяя вашим DAO оставаться независимыми от исключений Hibernate.

Как интегрировать Spring и Hibernate?

Для интеграции Hibernate в Spring необходимо подключить зависимости, а так же настроить файл конфигурации Spring. Т.к. настройки несколько отличаются между проектами и версиями, то смотрите официальную документацию Spring и Hibernate для уточнения настроек для конкретных технологий.

Как задаются файлы маппинга Hibernate в Spring?

Через applicationContext.xml в web/WEB-INF

Как добавить поддержку Spring в web-приложение

Достаточно просто указать ContextLoaderListener в web.xml файле

В чем различие между web.xml и the Spring Context - servlet.xml?

web.xml — Метаданные и конфигурация любого веб-приложения, совместимого с Java EE. Java EE стандарт для веб-приложений.servlet.xml — файл конфигурации, специфичный для Spring Framework.

Что предпочитаете использовать для конфигурации Spring - xml или аннотирование?

Предпочитаю аннотации, если кодовая база хорошо описывается такими элементами, как @Service, @Component, @Autowired

Однако когда дело доходит до конфигурации, у меня нет каких-либо предпочтений. Я бы оставил этот вопрос команде.

Можно ли использовать xyz.xml вместо applicationContext.xml?

ContextLoaderListener - это ServletContextListener, который инициализируется когда ваше web-приложение стартует. По-умолчанию оно загружает файл WEB-INF/applicationContext.xml. Вы можете изменить значение по-умолчанию, указав параметр contextConfigLocation.

Расскажите о Spring Security.

Проект Spring Security предоставляет широкие возможности для защиты приложения. Кроме стандартных настроек для аутентификации, авторизации и распределения ролей и маппинга доступных страниц, ссылок и т.п., предоставляет защиту от различных вариантов атак (например CSRF). Имеет множество различных настроек, но остается легким в использовании.

Этапы инициализации контекста Spring

1. Парсирование конфигурации и создание BeanDefinition

2. Настройка созданных BeanDefinition

3. Создание кастомных FactoryBean

4. Создание экземпляров бинов

5. Настройка созданных бинов

Этап Парсирование конфигурации и создание BeanDefinition

После выхода четвертой версии спринга, у нас появилось четыре способа конфигурирования контекста:

Xml конфигурация — ClassPathXmlApplicationContext(«context.xml»)

Конфигурация через аннотации с указанием пакета для сканирования — AnnotationConfigApplicationContext(«package.name»)

Конфигурация через аннотации с указанием класса (или массива классов) помеченного аннотацией @Configuration -AnnotationConfigApplicationContext(JavaConfig.class). Этот способ конфигурации называется — JavaConfig.

Groovy конфигурация — GenericGroovyApplicationContext(«context.groovy»)

Цель первого этапа — это создание всех BeanDefinition. BeanDefinition — это специальный интерфейс, через который можно получить доступ к метаданным будущего бина. В зависимости от того, какая у вас конфигурация, будет использоваться тот или иной механизм парсирования конфигурации.

Xml конфигурация

Для Xml конфигурации используется класс — XmlBeanDefinitionReader, который реализует интерфейс BeanDefinitionReader. Тут все достаточно прозрачно. XmlBeanDefinitionReader получает InputStream и загружает Document через DefaultDocumentLoader. Далее обрабатывается каждый элемент документа и если он является бином, то создается BeanDefinition на основе заполненных данных (id, name, class, alias, init-method, destroy-method и др.). Каждый BeanDefinition помещается в Map. Map хранится в классе DefaultListableBeanFactory. В коде Map выглядит вот так.

private final Map<String, BeanDefinition> beanDefinitionMap = new ConcurrentHashMap<String, BeanDefinition>(64);

Конфигурация через аннотации с указанием пакета для сканирования или JavaConfig

Конфигурация через аннотации с указанием пакета для сканирования или JavaConfig в корне отличается от конфигурации через xml. В обоих случаях используется класс AnnotationConfigApplicationContext.

new AnnotationConfigApplicationContext(JavaConfig.class);

или

new AnnotationConfigApplicationContext(«package.name»);

Если заглянуть во внутрь AnnotationConfigApplicationContext, то можно увидеть два поля.

private final AnnotatedBeanDefinitionReader reader; private final ClassPathBeanDefinitionScanner scanner;

ClassPathBeanDefinitionScanner сканирует указанный пакет на наличие классов помеченных аннотацией @Component (или любой другой аннотацией которая включает в себя @Component). Найденные классы парсируются и для них создаются BeanDefinition.Чтобы сканирование было запущено, в конфигурации должен быть указан пакет для сканирования.

@ComponentScan({«package.name»})

или

<context:component-scan base-package=»package.name»/>

AnnotatedBeanDefinitionReader работает в несколько этапов.

Первый этап — это регистрация всех @Configuration для дальнейшего парсирования. Если в конфигурации используются Conditional, то будут зарегистрированы только те конфигурации, для которых Condition вернет true. Аннотация

Conditional появилась в четвертой версии спринга. Она используется в случае, когда на момент поднятия контекста нужно решить, создавать бин/конфигурацию или нет. Причем решение принимает специальный класс, который обязан реализовать интерфейс Condition.

Второй этап — это регистрация специального BeanFactoryPostProcessor, а именно BeanDefinitionRegistryPostProcessor, который при помощи класса ConfigurationClassParser парсирует JavaConfig и создает BeanDefinition.

Groovy конфигурация

Данная конфигурация очень похожа на конфигурацию через Xml, за исключением того, что в файле не XML, а Groovy. Чтением и парсированием groovy конфигурации занимается класс GroovyBeanDefinitionReader.

Этап Настройка созданных BeanDefinition

После первого этапа у нас имеется Map, в котором хранятся BeanDefinition. Архитектура спринга построена таким образом, что у нас есть возможность повлиять на то, какими будут наши бины еще до их фактического создания, иначе говоря мы имеем доступ к метаданным класса. Для этого существует специальный интерфейс BeanFactoryPostProcessor, реализовав который, мы получаем доступ к созданным BeanDefinition и можем их изменять. В этом интерфейсе всего один метод.

public interface BeanFactoryPostProcessor { void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException; }

Метод postProcessBeanFactory принимает параметром ConfigurableListableBeanFactory. Данная фабрика содержит много полезных методов, в том числе getBeanDefinitionNames, через который мы можем получить все BeanDefinitionNames, а уже потом по конкретному имени получить BeanDefinition для дальнейшей обработки метаданных.

Давайте разберем одну из родных реализаций интерфейса BeanFactoryPostProcessor. Обычно, настройки подключения к базе данных выносятся в отдельный property файл, потом при помощи PropertySourcesPlaceholderConfigurer они загружаются и делается inject этих значений в нужное поле. Так как inject делается по ключу, то до создания экземпляра бина нужно заменить этот ключ на само значение из property файла. Эта замена происходит в классе, который реализует интерфейс BeanFactoryPostProcessor. Название этого класса — PropertySourcesPlaceholderConfigurer. Весь этот процесс можно увидеть на рисунке ниже.Давайте еще раз разберем что же у нас тут происходит. У нас имеется BeanDefinition для класса ClassName. Код класса приведен ниже.

@Component public class ClassName { @Value(«${host}») private String host; @Value(«${user}») private String user; @Value(«${password}») private String password; @Value(«${port}») private Integer port; }

Если PropertySourcesPlaceholderConfigurer не обработает этот BeanDefinition, то после создания экземпляра ClassName, в поле host проинжектится значение — «${host}» (в остальные поля проинжектятся соответсвующие значения). Если PropertySourcesPlaceholderConfigurer все таки обработает этот BeanDefinition, то после обработки, метаданные этого класса будут выглядеть следующим образом.

@Component public class ClassName { @Value(«127.0.0.1») private String host; @Value(«root») private String user; @Value(«root») private String password; @Value(«27017») private Integer port; }

Соответственно в эти поля проинжектятся правильные значения.Для того что бы PropertySourcesPlaceholderConfigurer был добавлен в цикл настройки созданных BeanDefinition, нужно сделать одно из следующих действий.Для XML конфигурации.

<context:property-placeholder location=»property.properties» />

Для JavaConfig.

@Configuration @PropertySource(«classpath:property.properties») public class DevConfig { @Bean public static PropertySourcesPlaceholderConfigurer configurer() { return new PropertySourcesPlaceholderConfigurer(); } }

PropertySourcesPlaceholderConfigurer обязательно должен быть объявлен как static. Без static у вас все будет работать до тех пор, пока вы не попробуете использовать @ Value внутри класса @ Configuration.

Этап Создание кастомных FactoryBean

FactoryBean — это generic интерфейс, которому можно делегировать процесс создания бинов типа . В те времена, когда конфигурация была исключительно в xml, разработчикам был необходим механизм с помощью которого они бы могли управлять процессом создания бинов. Именно для этого и был сделан этот интерфейс. Для того что бы лучше понять проблему, приведу пример xml конфигурации.

<?xml version=»1.0» encoding=»UTF-8»?> <beans xmlns=»http://www.springframework.org/schema/beans» xmlns:xsi=»http://www.w3.org/2001/XMLSchema-instance» xmlns:context=»http://www.springframework.org/schema/context» xsi:schemaLocation=»http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd»> <bean id=»redColor» scope=»prototype» class=»java.awt.Color»> <constructor-arg name=»r» value=»255» /> <constructor-arg name=»g» value=»0» /> <constructor-arg name=»b» value=»0» /> </bean> </beans>

На первый взгляд, тут все нормально и нет никаких проблем. А что делать если нужен другой цвет? Создать еще один бин? Не вопрос.

<?xml version=»1.0» encoding=»UTF-8»?> <beans xmlns=»http://www.springframework.org/schema/beans» xmlns:xsi=»http://www.w3.org/2001/XMLSchema-instance» xmlns:context=»http://www.springframework.org/schema/context» xsi:schemaLocation=»http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd»> <bean id=»redColor» scope=»prototype» class=»java.awt.Color»> <constructor-arg name=»r» value=»255» /> <constructor-arg name=»g» value=»0» /> <constructor-arg name=»b» value=»0» /> </bean> <bean id=»green» scope=»prototype» class=»java.awt.Color»> <constructor-arg name=»r» value=»0» /> <constructor-arg name=»g» value=»255» /> <constructor-arg name=»b» value=»0» /> </bean> </beans>

А что делать если я хочу каждый раз случайный цвет? Вот тут то и приходит на помощь интерфейс FactoryBean.Создадим фабрику которая будет отвечать за создание всех бинов типа — Color.

public class ColorFactory implements FactoryBean<Color> {

@Override public Color getObject() throws Exception {

Random random = new Random();

Color color = new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255));

return color; }

@Override public Class<?> getObjectType() {

return Color.class; }

@Override public boolean isSingleton() {

return false; }

}

Добавим ее в xml и удалим объявленные до этого бины типа — Color.

<?xml version=»1.0» encoding=»UTF-8»?> <beans xmlns=»http://www.springframework.org/schema/beans» xmlns:xsi=»http://www.

w3.org/2001/XMLSchema-instance» xmlns:context=»http://www.springframework.org/schema/context» xsi:schemaLocation=»http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd»> <bean id=»colorFactory» class=»com.malahov.temp.ColorFactory»></bean> </beans>

Теперь создание бина типа Color.class будет делегироваться ColorFactory, у которого при каждом создании нового бина будет вызываться метод getObject.Для тех кто пользуется JavaConfig, этот интерфейс будет абсолютно бесполезен.

Этап Создание экземпляров бинов

Созданием экземпляров бинов занимается BeanFactory при этом, если нужно, делегирует это кастомным FactoryBean. Экземпляры бинов создаются на основе ранее созданных BeanDefinition.

Этап Настройка созданных бинов

Интерфейс BeanPostProcessor позволяет вклиниться в процесс настройки ваших бинов до того, как они попадут в контейнер. Интерфейс несет в себе несколько методов.

```
public interface BeanPostProcessor {

Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException;

Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException;

}
```

Оба метода вызываются для каждого бина. У обоих методов параметры абсолютно одинаковые. Разница только в порядке их вызова. Первый вызывается до init-метода, второй, после. Важно понимать, что на данном этапе экземпляр бина уже создан и идет его донастройка. Тут есть два важных момента:

1) Оба метода в итоге должны вернуть бин. Если в методе вы вернете null, то при получении этого бина из контекста вы получите null, а поскольку через бинпостпроцессор проходят все бины, после поднятия контекста, при запросе любого бина вы будете получать фиг, в смысле null.

2) Если вы хотите сделать прокси над вашим объектом, то имейте ввиду, что это принято делать после вызова init метода, иначе говоря это нужно делать в методе postProcessAfterInitialization.

Процесс донастройки показан на рисунке ниже. Порядок в котором будут вызваны BeanPostProcessor не известен, но мы точно знаем что выполнены они будут последовательно.Для того, что бы лучше понять для чего это нужно, давайте разберемся на каком-нибудь примере.При разработке больших проектов, как правило, команда делится на несколько групп. Например первая группа разработчиков занимается написанием инфраструктуры проекта, а вторая группа, используя наработки первой группы, занимается написанием бизнес логики. Допустим второй группе понадобился функционал, который позволит в их бины инжектить некоторые значения, например случайные числа.На первом этапе будет создана аннотация, которой будут помечаться поля класса, в которые нужно проинжектить значение.

```
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.FIELD)

public @interface InjectRandomInt {

int min() default 0;

int max() default 10;

}
```

По умолчанию, диапазон случайных числе будет от 0 до 10.Затем, нужно создать обработчик этой аннотации, а именно реализацию BeanPostProcessor для обработки аннотации InjectRandomInt.

```
@Component

public class InjectRandomIntBeanPostProcessor implements BeanPostProcessor {

private static final Logger LOGGER = LoggerFactory.getLogger(InjectRandomIntBeanPostProcessor.class);

@Override

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException { LOGGER.info(« postProcessBeforeInitialization::beanName = {}, beanClass = {}», beanName, bean.getClass().getSimpleName());

Field[] fields = bean.getClass().getDeclaredFields(); for (Field field : fields) {

if (field.isAnnotationPresent(InjectRandomInt.class)) {

field.setAccessible(true);

InjectRandomInt annotation = field.getAnnotation(InjectRandomInt.class);

ReflectionUtils.setField(field, bean, getRandomIntInRange(annotation.min(), annotation.max())); } }

return bean; }

@Override

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException { return bean; }

private int getRandomIntInRange(int min, int max) { return min + (int)(Math.random() * ((max - min) + 1)); }

}
```

Код данного BeanPostProcessor достаточно прозрачен, поэтому мы не будем на нем останавливаться, но тут есть один важный момент.BeanPostProcessor обязательно должен быть бином, поэтому мы его либо помечаем аннотацией @Component, либо регестрируем его в xml конфигурации как обычный бин.Первая группа разработчиков свою задачу выполнила. Теперь вторая группа может использовать эти наработки.

```
@Component @Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)

public class MyBean {

@InjectRandomInt

private int value1;

@InjectRandomInt(min = 100, max = 200)

private int value2;

private int value3;

@Override

public String toString() {

return «MyBean{« + «value1=» + value1 + «, value2=» + value2 + «, value3=» + value3 + ‘}’;

 }

}
```

В итоге, все бины типа MyBean, получаемые из контекста, будут создаваться с уже проинициализированными полями value1 и value2. Также тут стоить отметить, этап на котором будет происходить инжект значений в эти поля будет зависеть от того какой @ Scope у вашего бина. SCOPE_SINGLETON — инициализация произойдет один раз на этапе поднятия контекста. SCOPE_PROTOTYPE — инициализация будет выполняться каждый раз по запросу. Причем во втором случае ваш бин будет проходить через все BeanPostProcessor-ы что может значительно ударить по производительности.

Что такое Spring Boot?

pring Boot - это подпроект в рамках организации с открытым исходным кодом Spring. Это универсальное решение для компонентов Spring. Оно в основном упрощает использование Spring, сохраняет тяжелую конфигурацию и предоставляет различные стартовые средства, чтобы разработчики могли быстро приступить к работе.

В чем преимущества Spring Boot?

Spring Boot имеет следующие преимущества:

Простота использования, повышение эффективности разработки и обеспечение более быстрого и обширного начального опыта разработки Spring.

Из коробки, вдали от громоздкой конфигурации.

Предоставляет ряд некоммерческих функций, общих для крупномасштабных

проектов, таких как встроенный сервер, управление безопасностью, мониторинг рабочих данных, проверка рабочего состояния, внешняя настройка и т. Д.

Нет генерации кода и конфигурации XML.

Избегайте большого количества операций импорта Maven и различных конфликтов версий.

Какова основная аннотация Spring Boot? Из каких аннотаций в основном состоят?

Аннотации к классу запуска - @ SpringBootApplication, которая также является основной аннотацией Spring Boot. Основная комбинация включает следующие три аннотации:

Аннотации к классу запуска - @ SpringBootApplication, которая также является основной аннотацией Spring Boot. Основная комбинация включает следующие три аннотации:

@SpringBootConfiguration: объединяет аннотации @Configuration для реализации функции файлов конфигурации.

@EnableAutoConfiguration: включите функцию автоматической настройки или вы можете отключить параметр автоматической настройки, например отключение функции автоматической настройки источника данных: @SpringBootApplication (исключить {DataSourceAutoConfiguration.class})

@ComponentScan: сканирование компонентов Spring.

Вот другие основные аннотации:

@EnableConfigurationProperties — позволяет использовать бины с аннотацией @ ConfigurationProperties

@ConfigurationProperties — позволяет связывать проперти из файлов с бинами

@WebMvcTest — используется для тестов Spring MVC

@SpringBootTest — используется, когда нужны функции Spring Boot в тестах

@DataJpaTest — используется для теста компонентов JPA

Как он работает? Как он понимает что и как нужно сконфигурировать?

Он предполагает что вам надо, основываясь на зависимостях в classpath. «Соглашение над конфигурацией» — он автоконфигурирует Spring специально подобранными настройками, которые потом можно переопределить.

Для этого есть несколько аннотаций, @ ConditionalOnClass, @ConditionalOnBean, @ConditionalOnMissingBean и @ ConditionalOnMissingClass, которые позволяют применять условия к @ Configuration-классам и @Bean-методам в этих классах.

Например:

Бин будет создан только если определенная зависимость есть в classpath.

Используйте @ConditionalOnClass, установив туда имя класса из classpath.

Бин будет создан только если в контейнере нет бина такого типа или с таким именем.

Используйте @ConditionalOnMissingBean, установив туда имя или тип бина для проверки.

Что влияет на настройку Spring Boot?

Существует список аннотаций-условий, каждая из которых используется для управления созданием бинов. Вот список таких аннотаций(на самом деле их больше):

@ConditionalOnClass Пристутствие класса в classpath

@ConditionalOnMissingClass Отсутствие класса в classpath

@ConditionalOnBean Присутствие бина или его типа(класс бина)

@ConditionalOnMissingBean Отсутствие бина или его типа

@ConditionalOnProperty Присутствие Spring-свойства

@ConditionalOnResource Присутствие файла-ресурса

@ConditionalOnWebApplication Если это веб-приложение, будет использоваться WebApplicationContext @ ConditionalOnNotWebApplicationЕсли это не веб-приложение

Что такое JavaConfig?

Spring JavaConfig является продуктом сообщества Spring и предоставляет чистый Java-метод для настройки контейнера Spring IoC. Так что это помогает избежать использования конфигурации XML. Преимущества использования JavaConfig:

(1) Объектно-ориентированная конфигурация. Поскольку конфигурация определяется как класс в JavaConfig, пользователи могут в полной мере использовать объектно-ориентированные функции Java. Один класс конфигурации может наследовать другой, переопределять его метод @Bean и т. Д.

(2) Уменьшите или исключите конфигурацию XML. Доказаны преимущества внешней конфигурации, основанной на принципе внедрения зависимостей. Однако многие разработчики не хотят переключаться между XML и Java. JavaConfig предоставляет разработчикам чистый метод Java для настройки контейнера Spring, аналогичный по концепции конфигурации XML. С технической точки зрения можно использовать только класс конфигурации JavaConfig для настройки контейнера, но на самом деле многие люди думают, что смешивание и сопоставление JavaConfig и XML является идеальным.

(3) Безопасность типов и удобство рефакторинга. JavaConfig предоставляет безопасный способ настройки контейнера Spring. Благодаря поддержке универсальных шаблонов в Java 5.0 компоненты теперь можно получать по типу, а не по имени, без приведения типов или поиска на основе строк.

Делает ли Spring Boot сканирование компонентов? Где он их ищет по умолчанию?

Да, делает, если стоит аннотация @ SpringBootApplication, которая содержит аннотацию @ComponentScanning.

По умолчанию Spring ищет бины в том же package, что и класс с аннотацией. Это можно переопределить, указав классы(или package) в параметрах scanBasePackageClasses или scanBasePackage.

Что такое Spring Boot Starter POM? Чем он может быть полезен?

Стартеры предоставляют набор удобных дескрипторов зависимостей, которые можно включить в ваше приложение. Вы получаете один источник для spring и связанных с ним технологий без необходимости искать и копипастить дескрипторы развертывания.Например, если вы хотите начать работать с Spring JPA, всего лишь добавьте зависимость spring-boot-starter-data-jpa в ваш проект.

Стартеры содержат большинство зависимостей, нужных вам для запуска проекта, работают быстро и согласованно, и поддерживают наборы транзитивных зависимостей.

Каков принцип автоматической настройки Spring Boot?

Аннотации @EnableAutoConfiguration, @ Configuration, @ConditionalOnClass - это ядро автоматической настройки,

@EnableAutoConfiguration импортирует класс автоматической конфигурации, определенный в META-INF / spring. factories, в контейнер.

Отфильтруйте допустимые классы автоконфигурации.

Каждый класс автоматической конфигурации объединяет соответствующий xxxProperties.java для чтения файла конфигурации для функции автоматической настройки.

Как определяются property? Где находится дефолтный PropertySource в Spring Boot?

Spring Boot использует особый порядок PropertySource’ов для того чтобы позволить переопределять значения свойств.

Вот порядок источников для получения свойств приложения:

Общие настройки из директории ~/ spring-boot devtools.properties

Настройки из аннотации @ TestPropertySource из тестов

Атрибуты @SpringBootTest#properties

Аргументы командной строки

Свойства из SPRING_APPLICATION_JSON

Параметры для инициализации ServletConfig

Параметры для инициализации ServletContext

JNDI-атрибуты из java:comp/env

Java System Properties(System. getProperties())

Переменные ОС

RandomValueProperySource

Проперти для профилей, например YAML или application-{profile}.properties

application.properties и YAML не из вашего jar

application.properties и YAML из вашего jar

Аннотация @PropertySource на @ Configuration-классе

Проперти по умолчанию(устанавливаются через SpringApplication. setDefaultProperties())

Также добавлю, что property.yml всегда переопределяют property.properties.

Как вы понимаете последовательность загрузки конфигурации Spring Boot?

В Spring Boot есть несколько способов загрузить конфигурацию.

1) Файл свойств;

2) файл YAML;

3) переменные системного окружения;

4) Параметры командной строки;

и многое другое .....

Что такое ЯМЛ?

YAML - это удобочитаемый язык сериализации данных. Обычно используется для файлов конфигурации. По сравнению с файлами свойств, если мы хотим добавить сложные свойства в файл конфигурации, файл YAML будет более структурированным и менее запутанным. Видно, что YAML имеет иерархические данные конфигурации.

В чем преимущества конфигурации YAML?

YAML теперь можно рассматривать как очень популярный формат файла конфигурации, независимо от того, является ли он внешним или внутренним, вы можете увидеть конфигурацию YAML. Итак, каковы преимущества конфигурации YAML по сравнению с традиционной конфигурацией свойств?

Упорядоченная конфигурация, в некоторых особых случаях упорядоченная конфигурация очень важна

Поддержка массивов, элементы в массиве могут быть базовыми типами данных или объектами.

краткий

По сравнению с файлом конфигурации свойств, YAML имеет недостаток, заключающийся в том, что он не поддерживает аннотацию @PropertySource для импорта пользовательской конфигурации YAML.

Может ли Spring Boot использовать конфигурацию XML?

Spring Boot рекомендует использовать конфигурацию Java вместо конфигурации XML, но конфигурацию XML также можно использовать в Spring Boot.Конфигурация XML может быть представлена с помощью аннотации @ImportResource.

Что такое файл конфигурации ядра загрузки Spring? В чем разница между bootstrap.properties и application.properties?

При простой разработке Spring Boot может быть нелегко встретить файл конфигурации bootstrap.properties, но в сочетании с Spring Cloud эта конфигурация будет часто встречаться, особенно когда вам нужно загрузить некоторые файлы удаленной конфигурации.

Два файла конфигурации ядра загрузки Spring:

bootstrap (. yml или. properties): boostrap загружается родительским ApplicationContext, который загружается до приложения. Конфигурация вступает в силу на этапе загрузки контекста приложения. Вообще говоря, мы будем использовать его в Spring Cloud Config или Nacos. И атрибуты в Boostrap не могут быть перезаписаны;

application (.yml или. properties): загружается ApplicatonContext для автоматической настройки проектов весенней загрузки.

Что такое Spring Profiles?

Spring Profiles позволяет пользователям регистрировать bean-компоненты на основе файлов конфигурации (dev, test, prod и т. Д.). Следовательно, когда приложение работает в стадии разработки, могут быть загружены только определенные bean-компоненты, в то время как в PRODUCTION могут быть загружены некоторые другие bean-компоненты. Предположим, наше требование состоит в том, что документ Swagger применим только к среде контроля качества, а все остальные документы отключены. Это можно сделать с помощью файлов конфигурации. Spring Boot позволяет очень легко использовать файлы конфигурации.

Как запускать приложения Spring Boot на пользовательских портах?

Чтобы запустить приложение Spring Boot на настраиваемом порту, вы можете указать порт в application.properties. server. port = 8090

Как реализовать безопасность приложения Spring Boot?

Чтобы обеспечить безопасность Spring Boot, мы используем зависимость spring-boot-starter-security и должны добавить конфигурацию безопасности. Для этого требуется очень мало кода. Класс конфигурации должен будет расширить WebSecurityConfigurerAdapter и переопределить его методы.

Сравните преимущества и недостатки Spring Security и Shiro?

Поскольку Spring Boot официально предоставляет большое количество очень удобных готовых стартеров, включая Starter Spring Security, он упрощает использование Spring Security в Spring Boot, и даже нужно только добавить зависимость для защиты всех интерфейсов. Поэтому, если это проект Spring Boot, обычно выбирается Spring Security. Конечно, это всего лишь предложенная комбинация, с чисто технической точки зрения, какой бы ни была комбинация, проблем нет. По сравнению с Spring Security, Shiro имеет следующие характеристики:

Spring Security - это тяжелая структура управления безопасностью; Shiro - легкая структура управления безопасностью.

Spring Security имеет сложные концепции и громоздкую конфигурацию; у Широ простые концепции и простая конфигурация.

Spring Security мощен; Широ прост

Как решить междоменные проблемы в Spring Boot?

Междоменный доступ может быть решен через JSONP в интерфейсе пользователя, но JSONP может отправлять только запросы GET и не может отправлять запросы других типов. В приложениях в стиле RESTful это очень безвкусно, поэтому мы рекомендуем передать (CORS, Cross-origin совместное использование ресурсов) для решения междоменных проблем. Это решение не является уникальным для Spring Boot. В традиционной среде SSM CORS можно использовать для решения междоменных проблем, но до того, как мы настроили CORS в XML-файле, теперь мы можем реализовать интерфейс WebMvcConfigurer, а затем пе-

реопределить метод addCorsMappings Решайте междоменные проблемы.

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
@Override
public void addCorsMappings(CorsRegistry registry) {
 registry.addMapping(«/**»)
 .allowedOrigins(«*»)
 .allowCredentials(true)
 .allowedMethods(«GET», «POST», «PUT», «DELETE», «OPTIONS»)
 .maxAge(3600);
```

Передняя и задняя части проекта развертываются отдельно, поэтому необходимо решить междоменные проблемы.

Мы используем файлы cookie для хранения информации для входа в систему и выполняем контроль разрешений в перехватчике spring. Если разрешения не совпадают, мы напрямую возвращаем пользователю фиксированный результат json.

После входа пользователя в систему он используется нормально; когда пользователь выходит из системы или срок действия токена истекает из-за проблемы перехватчика и междоменной последовательности, возникает междоменное явление.

Мы знаем, что http-запрос должен сначала пройти через фильтр, а затем быть обработан перехватчиком после того, как он достигнет сервлета.Если мы поместим cors в фильтр, он может быть выполнен до перехватчика разрешений.

```
@Configuration
public class CorsConfig {
```

Что такое CSRF-атака?

CSRF означает подделку межсайтовых запросов. Это атака, которая заставляет конечного пользователя выполнять нежелательные действия в веб-приложении, прошедшем проверку подлинности. Атаки CSRF нацелены на запросы на изменение состояния, а не на кражу данных, поскольку злоумышленники не могут просматривать ответы на поддельные запросы.

Что такое монитор в Spring Boot?

Привод пружинного чехла - одна из важных функций в каркасе пружинного чехла. Монитор загрузки Spring помогает вам получить доступ к текущему состоянию работающего приложения в производственной среде. Некоторые индикаторы необходимо проверять и контролировать в производственной среде. Даже некоторые внешние приложения могут использовать эти службы для отправки предупреждающих сообщений соответствующему персоналу. Модуль монитора предоставляет набор конечных точек REST, к которым можно получить прямой доступ как URL-адреса HTTP для проверки статуса.

Как отключить безопасность конечных точек Actuator в Spring Boot?

По умолчанию все конфиденциальные конечные точки HTTP защищены, и только пользователи с ролью ACTUATOR могут получить к ним доступ. Безопасность реализуется с помощью стандартного метода HttpServletRequest.isUserInRole. Мы можем использовать для отключения безопасности. Отключать безопасность рекомендуется только в том случае, если доступ к конечной точке привода осуществляется через брандмауэр.

Как мы отслеживаем все микроконтроллеры Spring Bootслужба?

Spring Boot предоставляет конечные точки монитора для отслеживания показателей каждой микросервиса. Эти конечные точки полезны для получения информации о приложениях (например, о том, запущены ли они) и о том, нормально ли работают их компоненты (например, базы данных и т. Д.). Однако одним из основных недостатков или трудностей использования монитора является то, что мы должны открывать точки знаний приложения отдельно, чтобы понять его статус или работоспособность. Представьте себе микросервис, в котором задействовано 50 приложений, и администратору придется задействовать терминалы выполнения всех 50 приложений. Чтобы помочь нам разобраться в этой ситуации, мы будем использовать проект с открытым исходным кодом, расположенный по адресу. Он построен на Spring Boot Actuator, который предоставляет веб-интерфейс, который позволяет нам визуализировать показатели нескольких приложений.

Что такое WebSockets?

WebSocket - это протокол связи с компьютером, который обеспечивает полнодуплексный канал связи через одно TCP-соединение.

1. WebSocket - это двунаправленный клиент или сервер WebSocket для инициирования отправки сообщений.

2. WebSocket является полнодуплексным, взаимодействие клиента и сервера не зависит друг от друга.

3. Одиночное TCP-соединение - начальное соединение использует HTTP, а затем это соединение обновляется до соединения на основе сокетов. Затем это единственное соединение используется для всех будущих коммуникаций.

4. Легкость - по сравнению с http, обмен данными сообщениями через WebSocket намного легче.

Что такое данные Spring?

Spring Data - это подпроект Spring. Используется для упрощения доступа к базе данных, поддерживает NoSQL и реляционное хранилище данных. Основная цель - сделать доступ к базе данных удобным и быстрым. Spring Data имеет следующие характеристики:

Проект SpringData поддерживает хранилище NoSQL:

1 MongoDB (база данных документов)

2 Neo4j (база данных графиков)

3 Redis (хранилище ключей / значений)

4 Hbase (база данных семейства столбцов)

Технологии реляционного хранения данных, поддерживаемые проектом SpringData:

1 JDBC

2 JPA

Spring Data Jpa стремится сократить объем разработки уровня доступа к данным (DAO). Единственное, что нужно сделать разработчикам, - это объявить интерфейс уровня сохраняемости, а Spring Data JPA сделает все остальное за вас! Spring Data JPA определяет, какую логику должен реализовать метод, в соответствии с именем стандартного метода в соответствии с именем стандарта.

Что такое Spring Batch?

Spring Boot Batch предоставляет функции многократного использования, которые очень важны при обработке большого количества записей, включая журнал / трассировку, управление транзакциями, статистику обработки заданий, перезапуск задания, пропуск и управление ресурсами. Он также предоставляет более продвинутые технические услуги и функции.С помощью технологии оптимизации и разделения можно реализовать чрезвычайно высокие пакетные и высокопроизводительные задания пакетной обработки. Простые и сложные задания пакетной обработки могут использовать платформу для обработки важных и больших объемов информации с высокой степенью масштабируемости.

Что такое шаблон FreeMarker?

FreeMarker - это шаблонизатор на основе Java, изначально ориентированный на использование программной архитектуры MVC для создания динамических веб-страниц. Основным преимуществом использования Freemarker является полное разделение уровня представления и бизнес-уровня. Программисты могут обрабатывать код приложения, а дизайнеры могут заниматься дизайном HTML-страниц. Наконец, используйте freemarker, чтобы объединить их, чтобы получить окончательную страницу выво-

да.

Как интегрировать Spring Boot и ActiveMQ?

Для интеграции Spring Boot и ActiveMQ мы используем зависимости. Это требует очень небольшой настройки и никакого шаблонного кода.

Что такое Apache Kafka?

Apache Kafka - это распределенная система обмена сообщениями «публикация-подписка». Это масштабируемая, отказоустойчивая система обмена сообщениями «публикация-подписка», которая позволяет нам создавать распределенные приложения. Это проект верхнего уровня Apache. Kafka подходит для использования сообщений офлайн и онлайн.

Что такое Swagger? Вы реализовали это с помощью Spring Boot?

Swagger широко используется для визуализации API-интерфейсов с использованием пользовательского интерфейса Swagger для предоставления онлайн-песочниц для интерфейсных разработчиков. Swagger - это инструмент для создания визуальных представлений RESTful Web-сервисов, спецификации и полной реализации инфраструктуры. Он позволяет обновлять документы с той же скоростью, что и сервер. При правильном определении с помощью Swagger потребители могут использовать минимум логики реализации для понимания удаленных служб и взаимодействия с ними. Поэтому Swagger исключает догадки при вызове сервисов.

Передняя и задняя части разделены, как поддерживать документы интерфейса?

Front-end и back-end разработка становится все более популярной. В большинстве случаев мы используем Spring Boot для выполнения front-end и back-end разработки. Должны быть документы интерфейса для разделения front-end и back-end. Глупый способ - использовать word или md для поддержки документов интерфейса, но эффективность слишком мала.Когда интерфейс меняется, документы в руках каждого должны измениться. В Spring Boot обычным решением этой проблемы является Swagger. Используя Swagger, мы можем быстро создать веб-сайт документации интерфейса. После изменения интерфейса документация будет автоматически обновляться. Все инженеры-разработчики могут получить доступ к этому онлайн-веб-сайту, чтобы получить последнюю версию Документация по интерфейсу очень удобна.

Как перезагрузить изменения в Spring Boot без перезапуска сервера? Как выполнить горячее развертывание проекта Spring Boot?

Этого можно добиться с помощью инструмента DEV. С помощью этой зависимости вы можете сохранить любые изменения, и встроенный tomcat перезапустится. Spring Boot имеет модуль инструментов разработки (DevTools), который помогает повысить продуктивность разработчиков. Одна из основных проблем, с которыми сталкиваются разработчики Java, - это автоматическое развертывание изменений файлов на сервере и автоматический перезапуск сервера. Разработчики могут перезагрузить изменения в Spring Boot без перезапуска сервера. Это избавит от необходимости каждый раз вручную развертывать изменения. Spring Boot не имел этой функции, когда выпустил свою первую версию. Это функция, которая больше всего нужна разработчикам. Модуль DevTools полностью удовлетворяет потребности разработчиков. Этот модуль будет отключен в производственной среде. Он также предоставляет консоль базы данных H2 для лучшего тестирования приложений.

Какие начальные зависимости maven вы использовали?

Используются некоторые из следующих зависимостей

spring-boot-starter-activemq

spring-boot-starter-security

Это помогает уменьшить количество зависимостей и уменьшить конфликты версий.

Что такое стартер в Spring Boot?Это?

Прежде всего, этот Starter не является новой технической точкой и в основном реализован на основе существующих функций Spring. Прежде всего, он предоставляет автоматизированный класс конфигурации, обычно называемый XXXAutoConfiguration. В этом классе конфигурации условные аннотации используются для определения того, вступает ли конфигурация в силу (условные аннотации присущи Spring), а затем он также предоставляет серию конфигураций по умолчанию. , Разработчикам также разрешено настраивать соответствующую конфигурацию в соответствии с реальной ситуацией, а затем вводить эти атрибуты конфигурации с помощью безопасного внедрения атрибутов. Вновь введенные атрибуты заменят атрибуты по умолчанию. Из-за этого многие сторонние фреймворки можно использовать напрямую, вводя зависимости. Конечно, разработчики также могут настроить Starter

Какая польза от spring-boot-starter-parent?

Все мы знаем, что недавно созданный проект Spring Boot по умолчанию имеет родителя. Этот родительский элемент - spring-boot-starter-parent. Spring-boot-starter-parent имеет следующие функции:

1 Скомпилированная версия Java определена как 1.8.

2 Используйте кодировку формата UTF-8.

3 Унаследованный от spring-boot-dependencies, он определяет версию зависимости. Именно потому, что эта зависимость наследуется, нам не нужно записывать номер версии при написании зависимости.

4 Конфигурация для выполнения упаковочных операций.

5 Автоматическая фильтрация ресурсов.

6 Автоматическая настройка плагинов.

7 Фильтрация ресурсов для application.properties и application.yml включает файлы конфигурации различных сред, определенных профилем, таких как application-dev.properties и application-dev.yml.

В чем разница между банкой Spring Boot и обычной банкой?

В конечном итоге jar-файл, упакованный в проект Spring Boot, является исполняемым jar-файлом. Этот jar-файл можно запустить напрямую с помощью команды java -jar xxx.jar. Этот jar-файл нельзя использовать в качестве обычного jar-файла в других проектах. Даже если это зависит, его нельзя использовать. тип.

На Spring Boot jar нельзя полагаться в других проектах, главным образом потому, что его структура отличается от обычных jar-файлов. Обычный пакет jar, имя пакета находится сразу после распаковки, и наш код находится в пакете. После распаковки исполняемый jar, упакованный Spring Boot, является нашим кодом в каталоге \ BOOT-INF \ classes, поэтому он не может быть напрямую Справка. Если вам нужно обратиться к нему, вы можете добавить конфигурацию в файл pom.xml и упаковать проект Spring Boot в два jar-файла, один исполняемый файл и один ссылочный.

Как реализовать задачи синхронизации в Spring Boot?

Задачи по времени также являются обычным требованием.Поддержка задач по времени в Spring Boot в основном происходит из среды Spring.

В Spring Boot есть два разных способа использования синхронизированных задач: один - использовать аннотацию @Scheduled в Spring, а другой - использовать стороннюю структуру Quartz.

Способ использования @Scheduled в Spring в основном реализуется с помощью аннотации @Scheduled

Верно или ложно следующее утверждение: «Каждое приложение Spring Boot - это веб-приложение,

работающее во встроенном Apache Tomcat». Обоснуйте свой ответ

Утверждение ложное.

Когда дело доходит до веб-приложений, Spring Boot работает с множеством контейнеров сервлетов. По умолчанию используется Apache Tomcat , но вы также можете использовать веб-приложение с Jetty, Undertow или вообще без встроенного контейнера сервлетов.

Более того, Spring Boot не привязан только к веб-приложениям, хотя такое впечатление можно получить, используя зависимость spring-boot-starter-web и, следовательно, автоконфигурацию Spring Boot для веб-сайтов. С помощью Spring Boot вы можете писать все виды сервисов, от пакетных заданий и утилит командной строки до серверных модулей обмена сообщениями и реактивных веб-приложений.

В чем разница между Spring Boot и Spring MVC? Или между Spring Boot и Spring Framework? Можете ли вы использовать их вместе в одном проекте?

Spring Boot построен поверх Spring Framework.

Пример: Spring Framework предлагает вам возможность читать файлы свойств .properties из различных мест, например, с помощью аннотаций @PropertySource. Он также предлагает вам возможность писать JSON REST контроллеры с помощью инфраструктуры Web MVC.

Проблема в том, что вы должны указать Spring откуда читать свойства приложения и правильно настроить веб-фреймворк, например, для поддержки JSON. Spring Boot, с другой стороны, берет эти отдельные части и предварительно настраивает их для вас.

Например:

Он всегда автоматически ищет файлы application.properties в различных заранее определенных местах и считывает их.

Он всегда загружает встроенный Tomcat, поэтому вы можете сразу увидеть результаты написания ваших @RestControllers и начать писать веб-приложения.

Он автоматически настраивает все для отправки / получения JSON, не беспокоясь о конкретных зависимостях Maven / Gradle. Все, путем запуска основного метода в классе Java, который аннотируется аннотацией @SpringBootApplication.

Если это объяснение по-прежнему оставляет у вас вопросы, ознакомьтесь с этим обширным руководством по Spring Framework, которое более подробно освещает эту тему.

Назовите два способа создать новый проект Spring Boot с нуля? Кроме того, как узнать, какие Spring Boot стартеры нужны вашему проекту?

Вы можете создавать новые проекты Spring Boot с помощью веб-приложения Spring Initializr или Spring Boot CLI. Интересно, что Spring Initializr - это не просто веб-сайт, на котором вы можете создавать .zip файлы скелета проекта. Это также API, который можно вызывать программно. Все основные IDE (Spring Tool Suite, IntelliJ IDEA Ultimate, Netbeans и VSCode) напрямую интегрируются с ним, так что вы можете создавать новые проекты Spring Boot прямо из вашей IDE.

Что касается стартеров, вам нужно прочитать документацию и иметь немного опыта. Если вы работаете с веб-приложением, вы начнете с spring-boot-starter-web, а затем добавите соответствующий стартер из документации, как только вы захотите включить определенную технологию. Через некоторое время вы получите хорошее представление о том, какие стартеры вам нужны для вашей технологии.

Почему вам не нужно указывать версии зависимостей в файле pom.xml при включении сторонних библиотек? Верно ли это для всех сторонних библиотек или только для некоторых? Как узнать, какие библиотеки поддерживает Spring Boot?

Это потому, что Spring Boot выполняет за вас некоторое управление зависимостями.

На верхнем уровне стартеры Spring Boot закачивают родительский файл pom.xml (или файл build.gradle), в котором определены все зависимости и соответствующие версии, которые поддерживает конкретная версия Spring Boot - так называемый BOM (Bill Of Materials). Затем вы можете просто использовать эти предопределенные версии или переопределить номера версий в своих собственных сценариях сборки.

Вы можете найти список всех поддерживаемых в настоящее время сторонних библиотек и версий в проекте spring-boot-dependencies.

Вы хотите сделать свое приложение настраиваемым, скажем, указать разное соединение с базой данных для среды разработки и рабочей среды. Какие у вас есть варианты?

По умолчанию Spring Boot извлекает свойства почти из 20 мест, от переменных среды и аргументов командной строки до файлов конфигурации (application.properties).Эти местоположения также упорядочены, так что местоположения, расположенные ниже в списке, имеют приоритет над более ранними. Вы можете, например, поместить файл application.properties по умолчанию в свой развертываемый файл .jar, тогда как ваши коллеги по эксплуатации переопределят только некоторые из них, предоставив свой собственный файл application.properties на машине развертывания.

Важно понимать эти местоположения и поведение по умолчанию, чтобы повторно не реализовывать функциональность свойств Spring Boot с настраиваемым PropertySourceProvider или использовать более тяжелые параметры, такие как сервер конфигурации Spring-Cloud.

Вам также необходимо убедиться, что вы понимаете концепцию нечёткой привязки свойств Spring (Relaxed properties binding), поскольку вы можете связывать свойства из этих мест со свойствами компонента конфигурации без явного совпадения имен.

Верно или неверно следующее утверждение: «Каждый проект Spring Boot должен использовать Thymeleaf в качестве механизма создания шаблонов HTML». Какие у вас есть возможности для рендеринга HTML?

Утверждение ложное.

Spring Boot работает с различными механизмами шаблонов HTML, и хотя Thymeleaf является популярным выбором и полностью интегрирован со Spring, вы также можете использовать многие другие, такие как Freemarker, Velocity или даже JSP (хотя это и не строго шаблонный движок).

Обычно рекомендуется выбрать вариант, который вам наиболее удобен / который используется в вашей компании по умолчанию.

Как можно реализовать доступ к реляционной базе данных с помощью Spring Boot? Какие у вас есть варианты?

Spring Boot интегрируется с множеством библиотек доступа к базам данных Java (см. Полный список здесь). Большинство пользователей, вероятно, будут использовать spring-boot-starter-jpa, spring-boot-starter-jdbc или один из соответствующих проектов spring-data.

Существуют также более легкие альтернативы, такие как jOOQ или myBatis. Наконец, вы всегда можете использовать опции NoSQL, такие как MongoDB и т. д.

Вам необходимо настроить ведение журнала в своем приложении, но вы хотите различать уровни журнала на вашем компьютере и уровни журнала в разных средах (qa, test, prod). Какие у вас есть варианты?

Во-первых, всегда полезно правильно понять экосистему ведения журналов

Java, а затем прочитать соответствующую главу Spring Boot документации о ведении журнала.

Когда дело доходит до настройки вывода, существуют различные варианты:

Непосредственно в файле application.properties (который, конечно, может и должен отличаться в средах DEV и PRD).

В зависимости от используемой структуры ведения журнала, указание настраиваемого файла конфигурации (например, logback-spring.xml).

Или даже через JMX во время выполнения.

Две популярные современные библиотеки ведения журнала, Logback и Log4j2, также поддерживают горячую перезагрузку конфигурации ведения журнала без необходимости перезагрузки вашего приложения.

Как проще всего развернуть приложение Spring Boot в рабочей среде? Какие еще есть варианты?

Самый простой способ развернуть приложение Spring Boot — это поместить .jar файл со встроенным контейнером сервлетов на любой сервер или платформу, на которой установлена JRE.По организационным и историческим причинам вы также можете развернуть приложение Spring Boot как файл .war в существующем контейнере сервлетов или сервере приложений.

И последнее, но не менее важное: вы, конечно, также можете поместить свой .jar файл в образ Docker и даже развернуть его с помощью Kubernetes.

Вам сказали включить «Spring Security» в вашем приложении. Что происходит, когда вы добавляете стартер Spring Security в свое приложение?

Это вопрос с подвохом. При добавлении Spring Security Starter в ваше приложение вы будете неожиданно запрашивать логин каждый раз, когда вы пытаетесь получить доступ к вашему приложению. Кроме того, отправка форм / конечные точки REST будут работать иначе или полностью заблокированы.

Суть в том, что вы «не просто включаете» безопасность в приложении Spring Boot, вам нужно четкое понимание того, что вы делаете.

Автор написал исчерпывающее руководство по Spring Security , которое объясняет все основные аспекты безопасности самым простым способом.

Как узнать, какие автоконфигурации Spring Boot применяются при запуске и какие условия оцениваются?

Spring Boot Actuator может предоставить эту информацию через конечные точки HTTP или JMX. Кроме того, вы можете запустить приложение Spring Boot с флагом «--debug».

Обратите внимание, что информация об оцениваемых условиях немного «сырая» и ее трудно переварить. Для этого прочтите это руководство, чтобы убедиться, что вы понимаете, как работают автоконфигурации Spring Boot.

В чем различия между встроенным контейнером и WAR?

Встроенный контейнер представляет собой сервер, который поставляется с конечным приложением, тогда как WAR является архивом, который может быть развернут на внешнем сервере.

Контейнеры сервлетов хороши для управления несколькими приложениями на одном хосте, но они не очень полезны для управления только одним приложением.

С облачным окружение используется одно приложение на виртуальную машину является предпочтительным и более распространенным способом. Современные фреимворки хотят быть более совместимыми с облаками, поэтому переходят на встраиваемые контейнеры.

Какие встроенные контейнеры поддерживает Spring?

Spring Boot поддерживает Tomcat, Jetty, и Undertow.По умолчанию используется TomCat. Для того чтобы изменить контейенер, просто добавьте нужную зависимость в pom.xml.

Что делает @EnableAutoConfiguration?

Она позволяет использовать автоконфигурацию. Автоконфигурация в Spring Boot пытается создать и настроить бины основываясь на зависимостях в classpath, для того чтобы позволить разработчику быстро начать работать с различными технологиями и убрать шаблонный код.

Что такое сервлет?

Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. Хотя сервлеты могут обслуживать любые запросы, они обычно используются для расширения веб-серверов. Для таких приложений технология Java Servlet определяет HTTP-специфичные сервлет классы. Пакеты javax.servlet и javax.servlet.http обеспечивают интерфейсы и классы для создания сервлетов.

Что такое контейнер сервлетов?

Контейнер сервлетов — программа, представляющая собой сервер, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл в соответствии с правилами, определёнными в спецификациях.

Известные реализации:Apache Tomcat, Jetty, JBoss, GlassFish, IBM WebSphere, Oracle Weblogic.

Какие задачи, функциональность контейнера сервлетов?

Контейнер сервлетов может работать как полноценный самостоятельный веб-сервер, быть поставщиком страниц для другого веб-сервера, например Apache, или интегрироваться в Java EE сервер приложений. Обеспечивает обмен данными между сервлетом и клиентами, берёт на себя выполнение таких функций, как создание программной среды для функционирующего сервлета, идентификацию и авторизацию клиентов, организацию сессии для каждого из них.

Что вы знаете о сервлет фильтрах?

Сервлетный фильтр, в соответствии со спецификацией, это Java-код, пригодный для повторного использования и позволяющий преобразовать содержание HTTP-запросов, HTTP-ответов и информацию, содержащуюся в заголовках HTTP. Сервлетный фильтр занимается предварительной обработкой запроса, прежде чем тот попадает в сервлет, и/или последующей обработкой ответа, исходящего из сервлета.Сервлетные фильтры могут:

— перехватывать инициализацию сервлета прежде, чем сервлет будет инициирован;

— определить содержание запроса прежде, чем сервлет будет инициирован;

— модифицировать заголовки и данные запроса, в которые упаковывается поступающий запрос;

— модифицировать заголовки и данные ответа, в которые упаковывается получаемый ответ;

— перехватывать инициализацию сервлета после обращения к сервлету.

Сервлетный фильтр может быть сконфигурирован так, что он будет работать с одним сервлетом или группой сервлетов. Основой для формирования фильтров служит интерфейс javax.servlet.Filter, который реализует три метода:void init (FilterConfig config) throws ServletException;void destroy();void doFilter (ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException;

Метод init() вызывается прежде, чем фильтр начинает работать,и настраивает конфигурационный объект фильтра. Метод doFilter выполняет непосредственно работу фильтра. Таким образом, сервер

вызывает init() один раз, чтобы запустить фильтр в работу, а затем вызывает doFilter() столько раз, сколько запросов будет сделано непосредственно к данному фильтру. После того, как фильтр заканчивает свою работу, вызывается метод destroy().

Зачем нужны слушатели в сервлетах?

Слушатели контекста и сессий — это классы, которые могут следить за тем, когда контекст или сессия были инициализированы, или отслеживать время, когда они должны быть уничтожены, и когда атрибуты были добавлены или удалены из контекста или сессии. Servlet 2.4 расширяет модель слушателей запроса, позволяя отслеживать, как запрос создается и уничтожается, и, как атрибуты добавляются и удаляются из сервлета. В Servlet 2.4 добавлены следующие классы:

ServletRequestListener

ServletRequestEvent

ServletRequestAttributeListener

ServletRequestAttributeEvent

Как обработать исключения, выброшенные другим сервлетом в приложении?

Т.к. браузер понимает только HTTP, то когда приложение выбросит исключение контейнер сервлетов обработает исключение и создаст HTTP response. Это аналогично тому что происходит при кодах ошибок вроде 404, 403 и т.д. Servlet API предоставляет поддержку собственных сервлетов для обработки исключений и ошибок, которые мы можем задать в дескрипторе развертывания. Главная задача таких сервлетов — обработать ошибку или исключение и отправить понятный HTTP ответ пользователю. Например, можно предоставить ссылку на главную страницу, а так же описание некоторых деталей об ошибке.

Что такое дескриптор развертывания?

Дескриптор развертывания — это конфигурационный файл артефакта, который будет развернут в контейнере сервлетов. В спецификации Java Platform, Enterprise Edition дескриптор развертывания описывает то, как компонент, модуль или приложение (такое, как веб-приложение или приложение предприятия) должно быть развернуто.

Этот конфигурационный файл указывает параметры развертывания для модуля или приложения с определенными настройками, параметры безопасности и описывает конкретные требования к конфигурации. Для синтаксиса файлов дескриптора развертывания используется язык XML.

Как реализовать запуск сервлета с запуском приложения?

Контейнер сервлетов обычно загружает сервлет при первом запросе клиента, но иногда необходимо загрузить сервлет прямо на старте приложения (например если сервлет объемный и будет долго грузиться). Для этого необходимо использовать элемент load-on-startup в дескрипторе (или аннотацию loadOnStartup), который укажет необходимость загрузки сервлета при запуске.

<servlet>

<servlet-name>foo</servlet-name>

<servlet-class>com.foo.servlets.Foo</servlet-class>

<load-on-startup>5</load-on-startup>

</servlet>

Значение должно быть int. Если значение отрицательное, то сервлет будет загружен при запросе клиента, а если 0 и далее, то загрузится на старте приложения. Чем меньше число, тем раньше в очереди на загрузку будет сервлет.

Что представляет собой объект ServletConfig?

Интерфейс javax.servlet.ServletConfig используется для передачи конфигурационной информации сервлету. Каждый сервлет имеет свой собственный объект ServletConfig, за создание экземпляра которого ответственен контейнер сервлетов. Для установки параметров конфигурации используются init параметры в web.xml (или аннотации WebInitParam). Для получения объекта ServletConfig данного сервлета используется метод getServletConfig().

Что представляет собой объект ServletContext?

Интерфейс javax.servlet.ServletContext предоставляет доступ к параметрам веб приложения сервлету. Объект ServletContext является уникальным и доступен всем сервлетам веб приложения. Мы можем использовать объект ServletContext, когда нам необходимо предоставить доступ одному или нескольким сервлетам к инициализированным параметрам веб приложения. Для этого используется элемент <context-param> в web.xml. Объект ServletContext можно получить с помощью метода getServletContext() у интерфейса ServletConfig.Контейнеры сервлетов так же могут предоставлять context объекты, уникальные для группы сервлетов. Каждая из групп будет связана со своим набором URL путей хоста.ServletContext был расширен в спецификации Servlet 3 и предоставляет программное добавление слушателей и фильтров в приложение. Так же у этого интерфейса имеются множество полезных методов вроде getMimeType(), getResourceAsStream() и т.д..

В чем отличия ServletContext и ServletConfig?

ServletConfig является уникальным объектом для каждого сервлета, в то время как ServletContext уникальный для всего приложения.

ServletConfig используется для предоставления параметров инициализации сервлету, а ServletContext для предоставления параметров инициализации приложения для всех сервлетов.

У нас нет возможности устанавливать атрибуты в объекте ServletConfig, в то время как можно установить атрибуты в объекте ServletContext, которые будут доступны другим сервлетам.

Что такое Request Dispatcher?

Интерфейс RequestDispatcher используется для передачи запроса другому ресурсу (это может быть HTML, JSP или другой сервлет в том же приложении). Мы можем использовать это для добавления контента другого ресурса к ответу. Этот интерфейс используется для внутренней коммуникации между сервлетами в одном контексте. В интерфейсе реализовано два метода:void forward(ServletRequest var1, ServletResponse var2) — передает запрос из сервлета к другому ресурсу (сервлету, JSP или HTML файлу) на сервере.void include(ServletRequest var1, ServletResponse var2) — включает контент ресурса (сервлет, JSP или HTML страница) в ответ.Доступ к интерфейсу можно получить с помощью метода ServletContext getRequestDispatcher(String s). Путь должен начинаться с / , который будет интерпретироваться относительным текущего корневого пути контекста.

Как можно создать блокировку (deadlock) в сервлете?

Дедлок можно получить реализовав зацикленный вызов метода, например вызвав метод doPost() в методе doGet() и вызвать doGet() в методе doPost().

Как получить адрес сервлета на сервере?

Для получения актуального пути сервлета на сервере можно использовать эту конструкцию: getServletContext(). getRealPath(request.getServletPath())

Как получить информацию о сервере из сервлета?

Информацию о сервере можно получить с использованием объекта ServletContext с помощью метода getServerInfo(). Т.е. getServletContext().getServerInfo().

Как получить ip адрес клиента на сервере?

Использовать request.getRemoteAddr() для получения ip клиента в сервлете.

Что вы знаете о классах обертках (wrapper) для сервлетов?

В Servlet HTTP API предоставляются два класса обертки — HttpServletRequestWrapper и HttpServletResponseWrapper. Они помогают разработчикам реализовывать собственные реализации типов request и response сервлета. Мы можем расширить эти классы и переопределить только необходимые методы для реализации собственных типов объектов ответов и запросов. Эти классы не используются в стандартном программировании сервлетов.

Каков жизненный цикл сервлета и когда какие методы вызываются?

Контейнер сервлетов управляет четырьмя фазами жизненного цикла сервлета:

Загрузка класса сервлета — когда контейнер получает запрос для сервлета, то происходит загрузка класса сервлета в память и вызов конструктора без параметров.

Инициализация класса сервлета — после того как класс загружен контейнер инициализирует объект ServletConfig для этого сервлета и внедряет его через init() метод. Это и есть место где сервлет класс преобразуется из обычного класса в сервлет.

Обработка запросов — после инициализации сервлет готов к обработке запросов. Для каждого запроса клиента сервлет контейнер порождает новую нить (поток) и вызывает метод service() путем передачи ссылки на объект ответа и запроса.

Удаление из Service — когда контейнер останавливается или останавливается приложение, то контейнер сервлетов уничтожает классы сервлетов путем вызова destroy() метода.

Можно описать как последовательность вызова методов: init(), service(), destroy().

public void init(ServletConfig config) - используется контейнером для инициализации сервлета. Вызывается один раз за время жизни сервлета.

public void service(ServletRequest request, ServletResponse response) - вызывается для каждого запроса. Метод не может быть вызван раньше выполнения init() метода.

public void destroy() - вызывается для уничтожения сервлета (один раз за время жизни сервлета).

В каком случае вы будете переопределять метод service()?

Метод service() переопределяется, когда мы хотим, чтобы сервлет обрабатывал как GET так и POST запросы в одном методе. Когда контейнер сервлетов получает запрос клиента, то происходит вызов метода service(), который в свою очередь вызывает doGet(), doPost() методы, основанные на HTTP методе запроса. Есть мнение, что метод service() переопределять особого смысла нет, кроме указанного вначале случая использования одного метода на два типа запросов.

Есть ли смысл определить конструктор для сервлета, как лучше инициализировать данные?

Такая возможность есть, но считается бессмысленной. Инициализировать данные лучше переопределив метод init(), в котором получить доступ к параметрам инициализации сервлета через использование объекта ServletConfig.

В чем отличия GenericServlet и HttpServlet?

Абстрактный класс GenericServlet — независимая от используемого протокола реализация интерфейса Servlet. HttpServlet, как понятно из название, реализация интерфейса сервлета для протокола HTTP. Следует отметить, что HttpServlet extends GenericServlet.

Как вызвать из сервлета другой сервлет этого же и другого приложения?

Если необходимо вызывать сервлет из того же приложения, то необходимо использовать механизм внутренней коммуникации сервлетов (inter-servlet communication mechanisms). Мы можем вызвать другой сервлет с помощью RequestDispatcher forward() и include() методов для доступа к дополнительным атрибутам в запросе для использования в другом сервлете. Метод forward() используется для передачи обработки запроса в другой сервлет. Метод include() используется, если мы хотим вложить результат работы другого сервлета в возвращаемый ответ.

Если необходимо вызывать сервлет из другого приложения, то использовать RequestDispatcher уже не получится (определен для приложения). Поэтому можно использовать ServletResponse sendRedirect() метод и предоставить полный URL из другого сервлета. Для передачи данных можно использовать cookies как часть ответа сервлета, а потом использовать их в нашем сервлете.

Стоит ли волноваться о «многопоточной безопасности» работая с сервлетами?

Методы класса HTTPServlet init() и destroy() вызываются один раз за жизненный цикл сервлета — поэтому по поводу них беспокоиться не стоит. Методы doGet(), doPost() вызываются на каждый запрос клиента и т.к. сервлеты используют многопоточность, то здесь нужно задумываться о потокобезопасной работе.

В случае наличия локальных переменных в этих методах нет необходимости думать о многопоточной безопасности, т.к. они будут созданы отдельно для каждой нити. Но если используются глобальные ресурсы, то необходимо использовать синхронизацию как и в любом многопоточном приложении Java.

В чем отличие между веб сервером и сервером приложений?

Веб сервер необходим для обработки HTTP request от браузера клиента и ответа клиенту с помощью HTTP response. Веб сервер понимает язык HTTP и запускается по HTTP протоколу. Примером веб сервера может служить реализация от Apache — Tomcat.

Сервер приложений предоставляет дополнительные возможности, такие как поддержка JavaBeans, JMS Messaging, Transaction Management и др. Можно сказать, что сервер приложений — это веб сервер с дополнительными возможностями, которые помогают разрабатывать корпоративные приложения.

Какой метод HTTP не является неизменяемым?

HTTP метод называется неизменяемым, если он всегда возвращает одинаковый результат. HTTP методы GET, PUT, DELETE, HEAD, OPTIONS являются неизменяемыми. Необходимо реализовывать приложение так, чтобы эти методы возвращали одинаковый результат. К изменяемым методам относится HTTP метод POST. Post метод используется для реализации чего-либо, что изменяется при каждом запросе.

К примеру, для доступа к HTML странице или изображению необходимо использовать метод GET, т.к. он возвращает одинаковый результат. Но если нам необходимо сохранить информацию о заказе в базе данных, то нужно использовать POST метод. Неизменяемые методы так же известны как безопасные методы и нет необходимости заботиться о повторяющихся запросах от клиента для этих методов.

В чем разница между методами GET и POST?

GET метод является неизменяемым, тогда как POST — изменяемый.

С помощью метода GET можно посылать ограниченное кол-во данных, которые будут пересланы в заголовке URL. В случае POST метода мы можем пересылать большие объемы данных, т.к. они будут находится в теле метода.

Данные GET метода передаются в открытом виде, что может использоваться в

зловредных целях. POST данные передаются в теле запроса и скрыты от пользователя.

GET метод является HTTP методом по умолчанию, а POST метод необходимо указывать явно, чтобы отправить запрос.

GET метод используется гиперссылками на странице.

Что такое MIME-тип?

MIME (произн. «майм», англ. Multipurpose Internet Mail Extensions — многоцелевые расширения интернет-почты) — стандарт, описывающий передачу различных типов данных по электронной почте, а также, в общем случае, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету. Content-Type response header это и есть MIME тип. Сервер посылает MIME тип клиенту для того, чтобы он понял какой тип данных пересылается. Это помогает верно отобразить полученные данные на клиенте. Наиболее часто используемые MIME типы: text/html, text/xml, application/xml и многие др.

В ServletContext существует метод getMimeType() для получения корректного MIME типа файла и дальнейшего использования этой информации для указания типа контента в ответе.

Какие различные методы управления сессией в сервлетах вы знаете?

Сессия является обычным состоянием взаимодействия сервера и клиента и может содержать в себе множество запросов и ответов клиент-сервер. Т.к. HTTP и веб сервер не запоминают состояния (stateless), то единственным способом поддерживать сессию является пересылка уникальной информации (session id) в каждом запросе и ответе между клиентом и сервером.

Существуют несколько распространенных способов управления сессией в сервлетах:

Аутентификация пользователя

HTML hidden field (скрытое поле)

Cookies

URL Rewriting

Session Management API

Как применяются Cookies в сервлетах?

Cookies (куки) используются в клиент-серверном взаимодействии и они не являются чем-то конкретным к Java. Servlet API предоставляет поддержку cookies через класс javax.servlet.http. Cookie implements Serializable, Cloneable. Для получения массива cookies из запроса необходимо воспользоваться методом HttpServletRequest getCookies(). Для добавления cookies в запрос методов не предусмотрено.

Аналогично HttpServletResponse addCookie(Cookie c) — может добавить cookie в response header, но не существует геттера для этого типа передачи данных.

HTTP это?

Протокол передачи гипертекста HTTP (Hypertext Transfer Protocol) - это протокол для распределённых информационных систем. Он был создан для обмена данными по сети Интернет.

HTTP базируется на протоколе TCP/IP, который используется для передачи данных (HTML страниц, результатов запросов, изображений и т.д.) по сети Интернет. По умолчанию, TCP использует 80-й порт, другие порты могут быть настроены дополнительно. TCP предоставляет стандартизированный способ взаимосвязи компьютеров между собой. Спецификация HTTP определяет, как именно запросы клиента должны быть построены и отправлен на сервер и то, как сервер должен отвечать на эти запросы.

Основные свойства HTTP

HTTP является простым, но в то же время сильным протоколом благодаря трем свойствам:

HTTP не зависит от соединения Клиент HTTP (чаще всего, браузер), отправляет HTTP запрос и, после отправки запроса, отсоединяется от сервера и ждёт ответа. Сервер обрабатывает запрос и создаёт новое соединение с клиентом для отправки ответа.

HTTP не привязан к конкретному типу данныхЭто означает, что с помощью HTTP мы можем передавать любой тип данных, при условии, что и клиент и сервер «умеют» работать с данным типом данных. Сервер и клиент должны определить тип контента с помощью определённого типа MIME.

HTTP взаимодействует только через соединениеКлиент и сервер могут взаимодействовать друг с другом только с помощью запроса. После этого они «забывают» друг о друге. Из-за этой особенности протокола ни клиент, ни сервер не могут получить информацию «за пределами» запроса.

HTTP/1.0 использует соединение для каждого цикла «запрос/ответ».

HTTP/1.1 может использовать один или несколько циклов «запрос-ответ» внутри одного соединения.

Идемпотентный метод

Метод HTTP является идемпотентным, если повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера. Другими словами, идемпотентный метод не должен иметь никаких побочных эффектов (side-effects), кроме сбора статистики или подобных операций. Корректно реализованные методы GET, HEAD, PUT и DELETE идемпотентны, но не метод POST. Также все безопасные методы являются идемпотентными.

Для идемпотентности нужно рассматривать только изменение фактического внутреннего состояния сервера, а возвращаемые запросами коды статуса могут отличаться: первый вызов DELETE вернёт код 200, в то время как последующие вызовы вернут код 404. Из идемпотентности DELETE неявно следует, что разработчики не должны использовать метод DELETE при реализации RESTful API с функциональностью удалить последнюю запись.

Базовая архитектура HTTP

В крайне упрощённой форме, архитектуру HTTP можно представить следующим образом:

Протокол HTTP основан клиент-серверной архитектуре, в которой браузер, «поисковик» и т.д. действует как «клиент», а веб-сервер - как «сервер».

Клиент

Клиент HTTP отправляет запрос на сервер в виде метода запроса, URL и версии протокола, после которых идёт MIME сообщение, которое и содержит модификаторы запроса, информацию о клиенте и, возможно контент соединения TCP/IP.

Сервер

Сервер HTTP отвечает на запрос строкой статуса, которая включает в себя версию протокола и код успешного выполнения, либо ошибки, после которого идёт сообщение MIME, содержащее информацию о сервере, мета-информацию о сущности и, возможно, контент самой сущности.

Стартовая строка HTTP

Стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать.

Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так:

Метод URI HTTP/Версия протокола

Пример запроса:

GET /web-programming/index.html HTTP/1.1

Стартовая строка ответа сервера имеет следующий формат:

HTTP/Версия КодСостояния [Пояснение]

Например, на предыдущий наш запрос клиентом данной страницы сервер ответил строкой:

HTTP/1.1 200 Ok

Заголовки HTTP

HTTP header обеспечивает необходимую информацию о запросе, ответе или об отправленном объекте в теле сообщения. Существует четыре типа HTTP сообщений header'a:

General-header Применимы как для запроса, так и для ответа.

Request-header Применимы только для запроса.

Response-header Применимы только для ответа.

Entity-header Определяют метаинформацию об объекте, переданном в теле, либо, если сообщение не содержит тела, о ресурсе, определённом запросом.

Тело сообщения HTTP

Это опциональный (не обязательный) элемент HTTP сообщения, который содержит объект, связанный с запросом, либо с ответом. Если объект тела связан с обычным Content-Type и Content-Length, то строки элемента header определяют тип конкретного объекта.

Тело сообщения содержит данные HTTP запроса (тип данных и т.д.), а HTTP ответ содержит данные, полученные от сервера (файлы, изображения и т.д.).

Метод запроса http

Данный элемент указывает метод, который должен быть вызван на стороне сервера по указанному идентификатору URI.

В HTTP существует восемь методов:

HEAD Используется для получения строки статуса и заголовка от сервера по URI. Не изменяет данные.

GET Используется для получения данных от сервера по указанному URI. Не изменяет данные.

POST Используется для отправки данных на сервер (например информации о разработчике и т.д.) с помощью форм HTML.

PUT Замещает все предыдущие данные на ресурсе новыми загруженными данными.

DELETE Удаляет все текущие данные на ресурсе, определённом URI.

CONNECT Устанавливает туннельное соединение с сервером по указанному URI.

OPTIONS Описывает свойства соединения для указанного ресурса.

TRACE Предоставляет сообщение, содержащее обратный трейс расположения указанного в URI ресурса.

URI (Uniform Resource Identifier) - это идентификатор ресурса на который отправляется запрос.

Код статуса HTTP

В ответ на запрос от клиента, сервер отправляет ответ, который содержит, в том числе, и код состояния. Данный код несёт в себе особый смысл для того, чтобы клиент мог отчётливей понять, как интерпретировать ответ:

1xx: Информационные сообщения

Набор этих кодов был введён в HTTP/1.1. Сервер может отправить запрос вида: Expect: 100-continue, что означает, что клиент ещё отправляет оставшуюся часть запроса. Клиенты, работающие с HTTP/1.0 игнорируют данные заголовки.

2xx: Сообщения об успехе

Если клиент получил код из серии 2xx, то запрос ушёл успешно. Самый распространённый вариант - это 200 OK. При GET запросе, сервер отправляет ответ в теле сообщения. Также существуют и другие возможные ответы:

202 Accepted: запрос принят, но может не содержать ресурс в ответе. Это полезно для асинхронных запросов на стороне сервера. Сервер определяет, отправить ресурс или нет.

204 No Content: в теле ответа нет сообщения.

205 Reset Content: указание серверу о сбросе представления документа.

206 Partial Content: ответ содержит только часть контента. В дополнительных заголовках определяется общая длина контента и другая инфа.

3xx: Перенаправление

Своеобразное сообщение клиенту о необходимости совершить ещё одно действие. Самый распространённый вариант применения: перенаправить клиент на другой адрес.

301 Moved Permanently: ресурс теперь можно найти по другому URL адресу.

303 See Other: ресурс временно можно найти по другому URL адресу. Заголовок Location содержит временный URL.

304 Not Modified: сервер определяет, что ресурс не был изменён и клиенту нужно задействовать закэшированную версию ответа. Для проверки идентичности информации используется ETag (хэш Сущности - Enttity Tag);

4xx: Клиентские ошибки

Данный класс сообщений используется сервером, если он решил, что запрос был отправлен с ошибкой. Наиболее распространённый код: 404 Not Found. Это означает, что ресурс не найден на сервере. Другие возможные коды:

400 Bad Request: вопрос был сформирован неверно.

401 Unauthorized: для совершения запроса нужна аутентификация. Информация передаётся через заголовок Authorization.

403 Forbidden: сервер не открыл доступ к ресурсу.

405 Method Not Allowed: неверный HTTP метод был задействован для того, чтобы получить доступ к ресурсу.

409 Conflict: сервер не может до конца обработать запрос, т.к. пытается изменить более новую версию ресурса. Это часто происходит при PUT запросах.

5xx: Ошибки сервера

Ряд кодов, которые используются для определения ошибки сервера при обработке запроса. Самый распространённый: 500 Internal Server Error. Другие варианты:

501 Not Implemented: сервер не поддерживает запрашиваемую функциональность.

503 Service Unavailable: это может случиться, если на сервере произошла ошибка или он перегружен. Обычно в этом случае, сервер не отвечает, а время, данное на ответ, истекает.

Чем отличаются HTTP и HTTPS?

HTTP (HyperText Transfer Protocol - «протокол передачи гипертекста») - протокол прикладного уровня передачи данных (изначально - в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTPS (HyperText Transfer Protocol Secure) - расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443.

Чем отличаются методы get и post?

GET передает данные серверу используя URL, а POST передает данные, используя тело HTTP запроса.

Длина URL'a ограничена 1024 символами, что и будет верхним пределом для данных, которые можно отослать GET'ом. POST может отправлять гораздо большие объемы данных. Кроме того, передача данных методом POST более безопасна, чем методом GET, так как секретные данные (например пароль) не отображаются напрямую в web-клиенте пользователя (в отличии от URL, который виден почти всегда).

Что такое www?

Всемирная паутина (сокращенно World

Wide Web или WWW) - это единство информационных ресурсов, которые связаны между собой средствами телекоммуникаций и основаны на гипертекстовом представлении данных, разбросанных по всему миру.

Что такое w3c?

W3C - абревиатура, которая обозначает Консорциум мировой сети (World Wide Web Consortium), организацию, цель которой - разработка и внедрение единых стандартов работы Интернета. Главная задача - это постоянное внедрение принципов работы мировой сети, главным из которых является полная совместимость всех материалов, размещенных в сети, с программным и аппаратным обеспечением пользователей.

Что такое URI?

URI, Uniform Resource Identifier (унифицированный идентификатор ресурса) - последовательность символов, идентифицирующая физический или абстрактный ресурс, который не обязательно должен быть доступен через сеть Интернет, причем, тип ресурса, к которому будет получен доступ, определяется контекстом и/или механизмом. Как правило делится на URL и URN, поэтому URL и URN это составляющие URI.

Что такое URL?

URL, Uniform Resource Locator (единообразный локатор (определитель местонахождения) ресурса). Также можно встретить более раннюю расшифровку аббревиатуры URL - Universal Resource Locator (универсальный локатор ресурсов) - другими словами это указатель размещения сайта в интернете, помимо идентификации ресурса, определяет местонахождение ресурса и способ обращения к нему. URL служит стандартизированным способом записи адреса ресурса в сети Интернет, URL-адрес содержит доменное имя и указание пути к странице, включая название файла этой страницы.

Что такое URN?

URN, Unifrorm Resource Name (унифицированное имя ресурса) - является уникальным именем объекта. URN включает в себя название пространства имен и идентификатора в этом пространстве. URN является частью концепции URI. Имена URN, в отличие от URL, не включают в себя указания на местонахождение и способ обращения к ресурсу. Смысл URN в том, что он определяет только название конкретного предмета, который может находится во множестве конкретных мест.

Что такое интернет протокол IP?

Интернет протокол (Internet Protocol, IP) - протокол сетевого уровня сетевой модели OSI (Open Systems Interconnection) и относится к протоколам, которые организуют соединения на основе коммутации каналов. Это один из самых распространенных протоколов является низкоуровневым маршрутизирующим сетевым протоколом, разбивающим данные на небольшие пакеты и посылающим их через сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу.

Что такое протокол управления TCP?

TCP, Transmission Control Protocol (Протокол Управления Передачей) - является сетевым протоколом более высокого уровня, обеспечивающим связывание, сортировку и повторную передачу пакетов, чтобы обеспечить надежную доставку данных.

Что такое TCP/IP?

Стек протоколов TCP/IP - набор сетевых протоколов передачи данных, используемых в сетях, включая сеть Интернет. Название TCP/IP происходит из двух наиважнейших протоколов семейства - Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были разработаны и описаны первыми в данном стандарте.

Что такое FTP?

FTP (File Transfer Protocol - протокол передачи файлов) - стандартный протокол, предназначенный для передачи файлов по TCP-сетям (например, Интернет). Использует 21й порт. FTP часто используется для загрузки сетевых страниц и других документов с частного устройства разработки на открытые сервера хостинга.

Что такое UDP?

UDP (User Datagram Protocol - протокол пользовательских датаграмм) - один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

Что такое протокол передачи данных, какие вы знаете?

Протокол передачи данных - набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.

HTTP (Hyper Text Transfer Protocol) - это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц с одного компьютера на другой.

FTP (File Transfer Protocol) - это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файлсо своего компьютера на удаленный.

POP (Post Office Protocol) - это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

SMTP (Simple Mail Transfer Protocol) - протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

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.

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. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными сервисами.

Что такое application server?

Сервер приложений (application server) - это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (Интерфейс прикладного программирования), который определен самой платформой.

Чем отличаются web server и application server?

Сервер приложений - сервер, исполняющий некоторые прикладные программы.

Веб-сервер - это сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы.

Что такое MIME type?

MIME (Multipurpose Internet Mail Extension, Многоцелевые расширения почты Интернета) - спецификация для передачи по сети файлов различного типа: изображений, музыки, текстов, видео, архивов и др. Указание MIME-типа используется в HTML обычно при передаче данных форм и вставки на страницу различных объектов.

Дайте определение понятиям «авторизация» и «аутентификация», в чем их различия?

Аутентификация - это проверка соответствия субъекта и того, за кого он пытается себя выдать, с помощью некой уникальной информации (отпечатки пальцев, цвет радужки, голос и тд.), в простейшем случае - с помощью имени входа и пароля.

Авторизация - это проверка и определение полномочий на выполнение некоторых действий (например, чтение файла /var/mail/eltsin) в соответствии с ранее выполненной аутентификацией.

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 преимущества от использования Hibernate?

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, ко-

торые 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 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?

Объект Hibernate Session является связью между кодом java приложения и hibernate. Это основной интерфейс для выполнения операций с базой данных. Жизненный цикл объекта session связан с началом и окончанием транзакции. Этот объект предоставляет методы для CRUD (create, read, update, delete) операций для объекта персистентности. С помощью этого экземпляра можно выполнять HQL, SQL запросы и задавать критерии выборки.

(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 :

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?

SessionFactory является фабрикой классов и используется для получения объектов session. SessionFactory отвечает за считывание параметров конфигурации Hibernate и подключение к базе данных. Обычно в приложении имеется только один экземпляр SessionFactory и потоки, обслуживающие клиентские запросы, получают экземпляры session с помощью объекта SessionFactory. Внутреннее состояние SessionFactory неизменно (immutable). Internal state (внутреннее состояние) включает в себя все метаданные об Object/ Relational Mapping и задается при создании SessionFactory.

SessionFactory также предоставляет методы для получения метаданных класса и статистики, вроде данных о втором уровне кэша, выполняемых запросах и т.д.

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:

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.

<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:

Управляемые транзакции на сервере приложений, использующий JTA:

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.

Если сессия обычно (у нас точно) привязана к транзакции и закрывается каждый раз по ее окончании, то SessionFactory создается один раз на все приложение. Этот кэш и считается кэшем второго уровня, но по умолчанию он не работает - его надо включать.

Кэш второго уровня - это прослойка, общая для всех сессий. То есть одна сессия извлекла сущность, а другая может получить к этой сущности потом доступ.

Очевидно, что с такой прослойкой есть проблема - ее данные могут устареть. В базе данные одни, а в кэше второго уровня - другие. Особенно, если помимо нашего приложения базу обновляет еще какой-то процесс. Но иногда все-таки от кэша второго уровня есть польза. Например, если наша сущность в принципе не редактируема (такой вот задан функционал), а доступна только для чтения. В этом случае почему бы не дать возможность всем сессиям брать готовую сущность, а не извлекать ее из базы каждый раз заново. Все равно сущность не устареет (так как она неизменяемая).

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 окру-

жении.

Что вы знаете о кэшировании 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 - это кэш Сессии (Session), который является обязательным. Через него проходят все запросы. Перед тем, как отправить объект в БД, сессия хранит объект за счёт своих ресурсов. А именно: Hibernate хранит отслеживаемые сущности в Map, ключами которой являются id сущностей, а значениями - сами объекты-сущности. Если мы извлекаем из базы сущность по id с помощью EntityManager.find(), то сущность помещается в этот Map и хранится в нем до закрытия сессии. И при повторном find() SQL-команда select в базе данных выполнена не будет. Hibernate возьмет эту сущность из Map - карты отслеживаемых сущностей.

В том случае, если мы выполняем несколько обновлений объекта, Hibernate старается отсрочить (насколько это возможно) обновление для того, чтобы сократить количество выполненных запросов. Если мы закроем сессию, то все объекты, находящиеся в кэше теряются, а далее - либо сохраняются, либо обновляются.

Как настраивается кэш второго уровня в Hibernate?

Чтобы указать кэш второго уровня нужно определить hibernate.cache.provider_class в hibernate.cfg.xml:

<hibernate-configuration>

<session-factory>

<property name=»hibernate.cache.provider_class»>org. hibernate, cache. EHCacheProvider</property>

</session-factory>

</hibernate-configuration>

По-умолчанию используется EHCache.

Чтобы использвать кэш запросов нужно его включить установив свойство hibernate.cache.use_query_cache в true в hibernate.properties.

Использовать аннотацию @Cache и указание настройки стратегии кэширование над entity bean.

```
import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity

@Table(name = «ADDRESS»)

@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region=»employee»)

public class Address {

}
```

Какая разница в работе методов load(); и get();?

Hibernate session обладает различными методами для загрузки данных из базы данных. Наиболее часто используемые методы для этого - get() и load().

Метод load(); обычно используется когда в не уверен что запрашиваемый объект уже находится в базе данных. Если объект не найден, то метод кидает исключение Если объект найден — метод возвращает прокси объект, который является ссылкой на объект находящийся в базе данных (запрос в базу данных еще не был осуществлен, своего рода lazy изъятие), непосредственный запрос к базе данных когда мы непосредственно обращаемся к необходимому объекту через прокси объект.

Метод get(); используется тогда, вы на 100 процентов не уверены есть ли запрашиваемый объект в базе данных. В случае обращение к несуществующему объекту, метод get(); вернет null. В случае нахождения объект, метод get(); вернет сам объект и запрос в базу данных будет произведен немедленно.

Каковы существуют различные состояния у entity bean?

Transient: состояние, при котором объект никогда не был связан с какой-либо сессией и не является персистентностью. Этот объект находится во временном состоянии. Объект в этом состоянии может стать персистентным при вызове метода save(), persist() или saveOrUpdate(). Объект персистентности может перейти в transient состоянии после вызова метода delete().

Persistent: когда объект связан с уникальной сессией он находится в состоянии persistent (персистентности). Любой экземпляр, возвращаемый методами get() или load() находится в состоянии persistent.

Detached: если объект был персистентным, но сейчас не связан с какой-либо сессией, то он находится в отвязанном (detached) состоянии. Такой объект можно сделать персистентным используя методы update(), saveOrUpdate(), lock() или replicate(). Состояния transient или detached так же могут перейти в состояние persistent как новый объект персистентности после вызова метода merge().

Что произойдет, если будет отсутствовать конструктор без аргументов у Entity Bean?

Hibernate использует рефлексию для создания экземпляров Entity бинов при вызове методов get() или load(). Для этого используется метод Class.newInstance(), который требует наличия конструктора без параметров. Поэтому, в случае его отсутствия, вы получите ошибку HibernateException.

Как используется вызов метода Hibernate Session merge()?

Hibernate merge() может быть использован для обновления существующих значений, однако этот метод создает копию из переданного объекта сущности и возвращает его. Возвращаемый объект является частью контекста персистентности и отслеживает любые изменения, а переданный объект не отслеживается.

В чем разница между Hibernate save(), saveOrUpdate() и persist()?

Hibernate save() используется для сохранения сущности в базу данных. Проблема с использованием метода save() заключается в том, что он может быть вызван без транзакции. А следовательно если у нас имеется отображение нескольких объектов, то только первичный объект будет сохранен и мы получим несогласованные данные. Также save() немедленно возвращает сгенерированный идентификатор. Hibernate persist() аналогичен save() с транзакцией. persist() не возвращает сгенерированный идентификатор сразу. Hibernate saveOrUpdate() использует запрос для вставки или обновления, основываясь на предоставленных данных. Если данные уже присутствуют в базе данных, то будет выполнен запрос обновления. Метод saveOrUpdate() можно применять без транзакции, но это может привести к аналогичным проблемам, как и в случае с методом save().

Что такое Lazy fetching(изъятие) в Hibernate?

Тип изьятия Lazy, в Hibernate, связан с листовыми(дочерними) сущностями и определяют политику совместного изъятия, если идет запрос на изъятие сущности родителя.

Простой пример:

Есть сущность Дом. Он хранит информацию о своем номере, улице, количество квартир и информацию о семьях которые живут в квартирах, эти семьи формируют дочернюю сущность относительно сущности Дом. Когда мы запрашиваем информацию о Доме, нам может быть совершенно ненужным знать информацию семьях которые в нем проживают, тут нам на помощь приходит lazy(ленивое) изъятие(fetching) которая позволяет сконфигурировать сущность Дом, чтобы информацию о семьях подавалась только по востребованию, это значительно облегчает запрос и ускоряет работу приложения.

В чем разница между sorted collection и ordered collection? Какая из них лучше?

При использовании алгоритмов сортировки из Collection API для сортировки коллекции, то он вызывает отсортированный список (sorted list). Для маленьких коллекций это не приводит к излишнему расходу ресурсов, но на больших коллекциях это может привести к потери производительности и ошибкам OutOfMemory. Так же entity бины должны реализовывать интерфейс Comparable или Comparator для работы с сортированными коллекциями. При использовании фреймворка Hibernate для загрузки данных из базы данных мы можем применить Criteria API и команду order by для получения отсортированного списка (ordered list). Ordered list является лучшим выбором к sorted list, т.к. он использует сортировку на уровне базы данных. Она быстрее и не может привести к утечке памяти.

Пример запроса к БД для получения ordered list:

```
List<Employee> empList = session.
createCriteria(Employee.class)
.addOrder(Order.desc(«id»)).list();
```

Как реализованы Join'ы Hibernate?

Существует несколько способов реализовать связи в Hibernate.

Использовать ассоциации, такие как one-to-one, one-to-many, many-to-many.

Использовать в HQL запросе команду JOIN. Существует другая форма «join fetch«, позволяющая загружать данные немедленно (не lazy).

Использовать чистый SQL запрос с командой join.

Почему мы не должны делать Entity class как final?

Хибернейт использует прокси классы для ленивой загрузки данных (т.е. по необходимости, а не сразу). Это достигается с помощью расширения entity bean и, следовательно, если бы он был final, то это было бы невозможно. Ленивая загрузка данных во многих случаях повышает производительность, а следовательно важна.

Что вы знаете о HQL и каковы его преимущества?

Hibernate Framework поставляется с мощным объектно-ориентированным языком запросов - Hibernate Query Language (HQL). Он очень похож на SQL, за исключением, что в нем используются объекты вместо имен таблиц, что делает язык ближе к объектно-ориентированному программированию. HQL является регистронезависимым, кроме использования в запросах имен java переменных и классов, где он подчиняется правилам Java. Например, SelECt то же самое, что и select, но com.blogspot.jsehelper.MyClass отличен от com.blogspot.jsehelper.MyCLASS. Запросы HQL кэшируются (это как плюс так и минус).

Что такое Query Cache в Hibernate?

Hibernate реализует область кэша для запросов resultset, который тесно взаимодействует с кэшем второго уровня Hibernate. Кэш запросов используется для кэширования результатов запроса. Когда кэш запроса включен, результаты запроса сохраняются вместе с комбинацией запросов и параметров вызова. Каждый раз запрос вызовет проверку на наличие результата у кэш менеджера. Если результаты найдены в кэше, они возвращаются, иначе инициализируется транзакция в БД.

Для подключения этой дополнительной функции требуется несколько дополнительных шагов в коде. Query Cache полезны только для часто выполняющихся запросов с повторяющимися параметрами. Для начала необходимо добавить эту запись в файле конфигурации Hibernate:

```
<property name=»hibernate.cache.use_
query_cache»>true</property>
```

Уже внутри кода приложения для запроса применяется метод setCacheable(true), как показано ниже:

```
Query query = session. createQuery(«from
Employee»);
query. setCacheable(true);
query. setCacheRegion(«ALL_EMP»);
```

Можем ли мы выполнить SQL (sql native) запрос в Hibernate?

С помощью использования SQLQuery можно выполнять чистый запрос SQL. В общем случае это не рекомендуется, т.к. вы потеряете все преимущества HQL (ассоциации, кэширование).

```
Transaction tx = session.beginTransaction();
SQLQuery query = session.
createSQLQuery(«select emp_id, emp_
name, emp_salary from Employee»);
List<Object[]> rows = query.list();
for(Object[] row : rows){
Employee emp = new Employee();
emp.setId(Long.parseLong(row[0].
toString()));
emp.setName(row[1].toString());
emp.setSalary(Double.parseDouble(row[2].
toString()));
System.out.println(emp);
}
```

Назовите преимущества поддержки нативного sql в Hibernate.

Использование нативного SQL может быть необходимо при выполнении запросов к некоторым базам данных, которые могут не поддерживаться в Hibernate. Примером может служить некоторые специфичные запросы и «фишки» при работе с БД от Oracle.

Что такое Named SQL Query?

Hibernate поддерживает именованный запрос, который мы можем задать в каком-либо центральном месте и потом использовать его в любом месте в коде. Именованные запросы поддерживают как HQL, так и Native SQL. Создать именованный запрос можно с помощью JPA аннотаций @NamedQuery, @NamedNativeQuery или в конфигурационном файле отображения (mapping files).

Каковы преимущества Named SQL Query?

Именованный запрос Hibernate позволяет собрать множество запросов в одном месте, а затем вызывать их в любом классе.

Синтаксис Named Query проверяется при создании session factory, что позволяет заметить ошибку на раннем этапе, а не при запущенном приложении и выполнении запроса.

Named Query глобальные, т.е. заданные однажды, могут быть использованы в любом месте.

Однако одним из основных недостатков именованного запроса является то, что его очень трудно отлаживать (могут быть сложности с поиском места определения запроса).

Как добавить логирование log4j в Hibernate приложение?

Добавить зависимость log4j в проект. Создать log4j.xml или log4j.properties файл и добавить его в classpath. Для веб приложений используйте ServletContextListener, а для автономных приложений DOMConfigurator или PropertyConfigurator для настройки логирования. Создайте экземпляр org.apache.log4j.Logger и используйте его согласно задачи.

Как логировать созданные Hibernate SQL запросы в лог-файлы?

Для логирования запросов SQL добавьте в файл конфигурации Hibernate строчку:

```
<property name=»hibernate.show_
sql»>true</property>
```

Что такое Hibernate?

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (object-relational mapping — ORM). Она представляет со-

бой свободное программное обеспечение с открытым исходным кодом (open source), распространяемое на условиях GNU Lesser General Public License. Данная библиотека предоставляет легкий в использовании каркас (фреймворк) для отображения объектно-ориентированной модели данных в традиционные реляционные базы данных. Hibernate совместима с JSR-220/317 и предоставляет стандартные средства JPA.

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 фреймморка, 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 вы знаете о Hibernate прокси и как это помогает в ленивой загрузке (lazy load)?

Hibernate использует прокси объект для поддержки отложенной загрузки. Обычно при загрузке данных из таблицы Hibernate не загружает все отображенные (замаппинные) объекты. Как только вы ссылаетесь на дочерний объект или ищите объект с помощью геттера, если связанная сущность не находиться в кэше сессии, то прокси код перейдет к базе данных для загрузки связанной сущности. Для этого используется javassist, чтобы эффективно и динамически создавать реализации подклассов ваших entity объектов.

Как управлять транзакциями с помощью Hibernate?

Hibernate вообще не допускает большинство операций без использования транзакций. Поэтому после получения экземпляра session от SessionFactory необходимо выполнить beginTransaction() для начала транзакции. Метод вернет ссылку, которую мы можем использовать для подтверждения или отката транзакции.

В целом, управление транзакциями в фреймворке выполнено гораздо лучше, чем в JDBC, т.к. мы не должны полагаться на возникновение исключения для отката транзакции. Любое исключение автоматически вызовет rollback.

Что такое каскадные связи (обновления) в Hibernate?

Если у нас имеются зависимости между сущностями (entities), то нам необходимо определить как различные операции будут влиять на другую сущность. Это реализуется с помощью каскадных связей (или обновлений). Вот пример кода с использованием аннотации @Cascade:

```
import org.hibernate.annotations.Cascade

@Entity

@Table(name = «EMPLOYEE»)

public class Employee {

@OneToOne(mappedBy = «employee»)

@Cascade(value = org.hibernate.
annotations.CascadeType.ALL)

private Address address;

}
```

Обратите внимание, что есть некоторые различия между enum CascadeType в Hibernate и в JPA. Поэтому обращайте внимание какой пакет вы импортируете при использовании аннотации и константы типа. Наиболее часто используемые CascadeType перечисления описаны ниже.

None: без Cascading. Формально это не тип, но если мы не указали каскадной связи, то никакая операция для родителя не будет иметь эффекта для ребенка.

ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. В общем - всё.

SAVE_UPDATE: Cascades save и update. Доступно только для hibernate.

DELETE: передает в Hibernate native DELETE действие. Только для hibernate.

DETATCH, MERGE, PERSIST, REFRESH и REMOVE - для простых операций.

LOCK: передает в Hibernate native LOCK действие.

REPLICATE: передает в Hibernate native REPLICATE действие.

Какие каскадные типы есть в Hibernate?

Наиболее часто используемые CascadeType перечисления описаны ниже.

None: без Cascading. Формально это не тип, но если мы не указали каскадной связи, то никакая операция для родителя не будет иметь эффекта для ребенка.

ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. В общем — всё.

SAVE_UPDATE: Cascades save и update. Доступно только для hibernate.

DELETE: передает в Hibernate native DELETE действие. Только для hibernate.

DETATCH, MERGE, PERSIST, REFRESH и REMOVE - для простых операций.

LOCK: передает в Hibernate native LOCK действие.

REPLICATE: передает в Hibernate native REPLICATE действие.

Что такое сесиия и фаблика сессий в Hibernate? Как настроить session factory в конфигурационном файле Spring?

Hibernate сессия - это главный интерфейс взаимодействия Java-приложения и Hibernate. SessionFactory позволяет создавать сессии согласно конфигурации hibernate.cfg.xml. Например:

```
// Initialize the Hibernate environment

Configuration cfg = new Configuration().
configure();

// Create the session factory

SessionFactory factory = cfg.
buildSessionFactory();

// Obtain the new session object

Session session = factory. openSession()
```

При вызове Configuration().configure()

загружается файл hibernate.cfg.xml и происходит настройка среды Hibernate. После того, как конфигурация загружена, вы можете сделать дополнительную модификацию настроек уже на программном уровне. Данные корректировки возможны до создания экземпляра фабрики сессий. Экземпляр SessionFactory как правило создается один раз и используется во всем приложении.

Главная задача сессии - обеспечить механизмы создания, чтения и удаления для экземпляров примапленых к БД классов. Экземпляры могут находиться в трёх состояниях:

transient - никогда не сохранялись, не ассоциированы ни с одной сессией;

persistent - ассоциированы с уникальной сессией;

detached - ранее сохраненные, не ассоциированы с сессией.

Объект Hibernate Session представляет одну операцию с БД. Сессию открывает фабрика сессий. Сессия должна быть закрыта, когда все операции с БД совершены. Пример:

```
Session session = null;

UserInfo user = null;

Transaction tx = null;

try

session = factory. openSession();

tx = session. beginTransaction();

user = (UserInfo) session. load(UserInfo.
class, id);

tx.commit();

catch(Exception e)

if (tx != null)

try

tx.rollback();
```

```
catch (HibernateException e1)
throw new DAOException(e1.toString());
throw new DAOException(e.toString());
finally
if (session != null)
try
session. close();
catch (HibernateException e) {
```

Как использовать JNDI DataSource сервера приложений с Hibernate Framework?

В веб приложении лучше всего использовать контейнер сервлетов для управления пулом соединений. Поэтому лучше определить JNDI ресурс для DataSource и использовать его в веб приложении. Для этого в Hibernate нужно удалить все специфичные для базы данных свойства из и использовать указания свойства JNDI DataSource:

<property name=»hibernate. connection. datasource»>java: comp/env/jdbc/ MyLoca1DB</property>

Как интегрировать Hibernate и Spring?

Лучше всего прочитать о настройках на сайтах фреймворков для текущей версии. Оба фреймворка поддерживают интеграцию из коробки и в общем настройка их взаимодействия не составляет труда. Общие шаги выглядят следующим образом.

Добавить зависимости для hibernate-entitymanager, hibernate-core и spring-orm.

Создать классы модели и передать реализации DAO операции над базой данных. Важно, что DAO классы используют SessionFactory, который внедряется в конфигурации бинов Spring.

Настроить конфигурационный файл Spring (смотрите в офф. документации или из примера на этом сайте).

Дополнительно появляется возможность использовать аннотацию @Transactional и перестать беспокоиться об управлением транзакцией Hibernate.

Какие паттерны применяются в Hibernate?

Domain Model Pattern - объектная модель предметной области, включающая в себя как поведение так и данные.

Data Mapper - слой мапперов (Mappers), который передает данные между объектами и базой данных, сохраняя их независимыми друг от друга и себя.

Proxy Pattern - применяется для ленивой загрузки.

Factory pattern - используется в SessionFactory

Расскажите о Hibernate Validator Framework.

Проверка данных является неотъемлемой частью любого приложения. Hibernate Validator обеспечивает эталонную реализацию двух спецификаций JSR-303 и JSR-349 применяемых в Java. Для настройки валидации в Hibernate необходимо сделать следующие шаги.

Добавить hibernate validation зависимости в проект.

Так же требуются зависимости из JSR 341, реализующие Unified Expression Language для обработки динамических выражений и сообщений о нарушении ограничений.

Использовать необходимые аннотации в бинах.

Какие преимущества дает использование плагина Hibernate Tools Eclipse?

Плагин Hibernate Tools упрощает настройку маппинга, конфигурационного файла. Упрощает работы с файлами свойств или xml тегами. Помогает минимизировать ошибки написания кода.

Расскажите о преимуществах использования Hibernate Criteria API.

Hibernate Criteria API является более объектно-ориентированным для запросов, которые получают результат из базы данных. Для операций update, delete или других DDL манипуляций использовать Criteria API нельзя. Критерии используются только для выборки из базы данных в более объектно-ориентированном стиле.

Вот некоторые области применения Criteria API:

Criteria API поддерживает проекцию, которую мы можем использовать для агрегатных функций вроде sum(), min(), max() и т.д.

Criteria API может использовать ProjectionList для извлечения данных только из выбранных колонок.

Criteria API может быть использована для join запросов с помощью соединения нескольких таблиц, используя методы createAlias(), setFetchMode() и setProjection().

Criteria API поддерживает выборку результатов согласно условиям (ограничениям). Для этого используется метод add() с помощью которого добавляются ограничения (Restrictions).

Criteria API позволяет добавлять порядок (сортировку) к результату с помощью метода addOrder().

Что вы знаете о классе HibernateTemplate?

Spring Framework предоставляет различные подходы для интеграции с Hibernate. Тем не менее, мы наиболее часто будем использовать подход, использующий HibernateTemplate. Есть две основные причины:

Класс скрывает детали управления сессиями и транзакциями.

Предоставляет подход основанный на шаблонах

HibernateTemplate класс скрывает трудности управления сессиями и транзакциями при использовании Hibernate для доступа к данным. Нужно только инициализировать HibernateTemplate путем передачи экземпляра SessionFactory. Spring Framework берет на себя беспокойство за детали связанные с сессиями и транзакциями. Это помогает устранить инфраструктурный код, который может вносить суматоху при увеличении сложности. HibernateTemplate, так же как и JdbcTemplate, предоставляет шаблонный подход для доступа к данным. Когда вы используете HibernateTemplate, вы будете работать с callbacks. Обратные вызовы - это единственный механизм в шаблонном подходе, который уведомляет шаблон запускать нужную задачу. Преимущество наличия обратного вызова в том, что там только одна точка входа в слой доступа к данным. И эта точка входа определяется шаблоном, в этом случае HibernateTemplate.

В комментариях дополнили, что использование HibernateTemplate не является рекомендуемым. Вместо использования HibernateTemplate из пакета org. springframework.orm рекомендуется использовать декларативный подход (@ Transactional). Таким образом фреймворк сам позаботится об операциях open, commit, close, flush.

Как интегрировать Hibernate с Servlet или Struts2 веб приложением?

Для интеграции необходимо использовать ServletContextListener.

Best Practices в Hibernate

При использовании фреймворка Hibernate рекомендуется придерживаться некоторых правил.

Всегда проверяйте доступ к primary key. Если он создается базой данных, то вы не должны иметь сеттера.

По умолчанию hibernate устанавливает значения в поля напрямую без использования сеттеров. Если необходимо заставить хибернейт их применять, то проверьте использование аннотации @ Access(value=AccessType.PROPERTY) над свойством.

Если тип доступа - property, то удостоверьтесь, что аннотация используется с геттером.

Избегайте смешивания использования

аннотации над обоими полями и геттером.

Используйте нативный sql запрос только там, где нельзя использовать HQL.

Используйте ordered list вместо сортированного списка из Collection API, если вам необходимо получить отсортированные данные.

Применяйте именованные запросы разумно - держите их в одном месте и используйте только для часто применяющихся запросов. Для специфичных запросов пишите их внутри конкретного бина.

В веб приложениях используйте JNDI DataSource вместо файла конфигурации для соединения с БД.

Избегайте отношений многие-ко-многим, т.к. это можно заменить двунаправленной One-to-Many и Many-to-One связью.

Для collections попробуйте использовать Lists, maps и sets. Избегайте массивов (array), т.к. они не дают преимуществ ленивой загрузки.

Не обрабатывайте исключения, которые могут откатить транзакцию и закрыть сессию. Если это проигнорировать, то Hibernate не сможет гарантировать, что состояние в памяти соответствует состоянию персистентности (могут быть коллизии данных).

Применяйте шаблон DAO для методов, которые могут использоваться в entity бинах.

Предпочитайте ленивую выборку для ассоциаций.

Что такое Hibernate? В чём разница между JPA и Hibernate?

Я думаю, чтобы ответить на данный вопрос, нам сперва нужно понять, что такое JPA. JPA — это спецификация, описывающая объектно-реляционное отображение простых Java объектов и предоставляющая API для сохранения, получения и управления такими объектами. То есть, как мы помним, реляционные базы данных (БД) представлены в виде множества связанных между собой таблиц. И JPA — общепринятый стандарт, который описывает, как объекты могут взаимодействовать с реляционными базами данных. Как видие, JPA — это что-то абстрактное и неосязаемое. Это как бы сама идея, подход.В то же время Hibernate — это конкретная библиотека, реализующая парадигмы JPA. То, есть с помощью этой библиотеки вы можете работать с реляционной базой данных через объекты, которые представляют данные с БД (Entity). Как говорят, данная библиотека очень близка к идеалам JPA и возможно, поэтому она стала популярна. А как вы понимаете, популярность использования — хороший аргумент для дальнешйей разработки и улучшений. К тому же за частым использованием стоит огромное комьюнити, которое разобрало уже все возможные и невозможные вопросы, связанные с данным инструментом.

Что такое каскадность? Как она используется в Hibernate?

Как я и сказал ранее, в Hibernate взаимодействие ведется через объекты данных, называемые entity. Эти entity представляют какие-то конкретные таблицы в базе данных, и как вы помните, в Java классы могут содержать ссылки на другие классы. Эти отношения отражаются и на базе данных. В БД, как правило, это либо внешние ключи (для OneToOne, OneToMany, ManyToOne), либо промежуточные таблицы (для ManyToMany) Подробнее о взаимосвязи между сущностями можно почитать в этой статье. Когда в вашем entity есть ссылки на другие связанные сущности, над этими ссылками ставятся аннотации для указания типа связи: @OneToOne, @OneToMany, @ ManyToOne, @ManyToMane, в чьих параметрах вы можете указать значение свойства — cascade — тип каскаданости для данной связи. У JPA есть специфические методы для взаимодействия с сущностями (persist, save, merge...). Каскадные типы как раз используются для того, чтобы показать, как должны себя вести связанные данные при использовании этих методов на целевую сущность. Итак, какие же существуют стратегии каскаскадности (типы каскадности)? Стандарт JPA подразумевает использование шести видов каскадности:

PERSIST — операции сохранения будут происходить каскадно (для методов save() и persist()). То есть, если мы сохраняем сущность, связанную с другими сущностями, они также сохраняются в БД (если их ещё там нет)

MERGE — операции обновления будут происходить каскадно (для метода merge())

REMOVE — операции удаления происходят каскадно (метод remove())

ALL — содержит сразу три каскадные операции — PERSIST - MERGE - REMOVE

В JPA есть понятие персистентная (persistence) сущность — сущность, связанная с её данными в БД, которая управляется текущей сессией (соединением). Если её изменить, но при этом не сохранить изменения в БД, всё равно её данные в БД будут изменены.

DETACH — связанные сущности не будут управляться сессией (метод detach()). То есть, при их изменении не будет автоматического изменения их данных в БД — они переводятся из состояния persistence в detached (сущность, не управляемая JPA)

REFRESH — при каждом обновлении сущности данными из БД (refresh() — обновляет detached объекты) связанные сущности обновляются так же. Например, вы изменили как-то данные, взятые из БД, и хотите вернуть их изначальные значения. В таком случае вам и пригодится данная операция.

Hibernate поддерживает все эти стандартные каскадные операции, но также привносит три свои:

REPLICATE — используется, когда у нас есть более одного источника данных и мы хотим, чтобы данные синхронизировались (метод Hibernate — replicate). У всех сущностей должны быть идентификаторы (id), чтобы не было проблем с их генерацией (чтобы для разных БД одна и та же сущность не имела разных id)

SAVE_UPDATE — каскадное сохранение/удаление (для метода Hibernate — saveOrUpdate)

LOCK — операция, обратная к DETACHED: переводит detached сущность обратно в состояние persistence, т.е. entity станет снова отслеживаемой текущей сессией

Если не выбран тип каскадирования, никакая операция с сущностью не будет иметь эффекта для связанных с ней других entity.

N+1 SELECT problem и пути ее решения?

Проблема n + 1 может возникнуть в случае, когда одна сущность (таблица) ссылается на другую сущность (таблицу).

В такой ситуации получается, что для получения значения зависимой сущности выполняется n избыточных запросов, в то время как достаточно одного.

Никого не нужно убеждать, что это негативно влияет на производительность системы и создает ненужную нагрузку на базу данных. Особенно то, что количество запросов увеличивается с ростом n.

Сама проблема часто представляется как возникающая только в отношениях «один ко многим» (javax.persistence. OneToMany) или только в случае ленивой загрузки данных (javax.persistence. FetchType.LAZY). Это не так, и следует помнить, что эта проблема также может возникнуть в отношениях один-к-одному и при «жадной» загрузке зависимых сущностей.

Решение

1)Устранение проблемы с помощью Join Fetch

2)Добавив аннотацию @BatchSize над полем , Hibernate получит данные в рамках одного запроса. (Всего их будет два)

@Transactional

Аннотация @Transactional указывает, что метод должен выполняться в транзакции. Менеджер транзакций открывает новую транзакцию и создаёт для неё экземпляр Session, который доступен через

sessionFactory.getCurrentSession(). Все методы, которые вызываются в методе с данной аннотацией, также имеют доступ к этой транзакции, потому что экземпляр Session является переменной потока (ThreadLocal). Вызов sessionFactory.openSession() откроет совсем другую сессию, которая не связана с транзакцией.

Параметр rollbackFor у @Transactional

Параметр rollbackFor указывает исключения, при выбросе которых должен быть произведён откат транзакции. Есть обратный параметр — noRollbackFor, указывающий, что все исключения, кроме перечисленных, приводят к откату транзакции.

Параметр propagation у @Transactional

Параметр propagation он указывает принцип распространения транзакции. Может принимать любое значение из перечисления

Propagation.REQUIRED — выполняться в существующей транзакции, если она есть, иначе создавать новую.

Propagation.MANDATORY — выполняться в существующей транзакции, если она есть, иначе генерировать исключение.

Propagation.SUPPORTS — выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции.

Propagation.NOT_SUPPORTED — всегда выполняться вне транзакции. Если есть существующая, то она будет остановлена.

Propagation.REQUIRES_NEW — всегда выполняться в новой независимой транзакции. Если есть существующая, то она будет остановлена до окончания выполнения новой транзакции.

Propagation.NESTED — если есть текущая транзакция, выполняться в новой, так называемой, вложенной транзакции. Если вложенная транзакция будет отменена, то это не повлияет на внешнюю транзакцию; если будет отменена внешняя транзакция, то будет отменена и вложенная. Если текущей транзакции нет, то просто создаётся новая.

Propagation.NEVER — всегда выполнять вне транзакции, при наличии существующей генерировать исключение.

Что такое веб сервисы?

Веб-служба, веб-сервис (англ. web service) — идентифицируемая веб-адресом программная система со стандартизированными интерфейсами. Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC, REST и т. д.). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения. К характеристикам веб сервисов относят:

Функциональная совместимость

Расширяемость

Возможность машинной обработки описания

В чем разница между SOA и web service?

Сервис-ориентированная архитектура (SOA, service-oriented architecture) — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных (англ. loose coupling) заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам. Программные комплексы, разработанные в соответствии с сервис-ориентированной архитектурой, обычно реализуются как набор веб-служб, взаимодействующих по протоколу SOAP, но существуют и другие реализации (например, на базе jini, CORBA, на основе REST). Веб сервисы реализующие эту концепцию используют XML, JSON и др., а так же интернет протоколы вроде HTTP(S), SMTP и др..

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.

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».

Преимущества, которые дает REST

У приложений, которые соблюдают все вышеперечисленные ограничения, есть такие преимущества: надёжность (не нужно сохранять информацию о состоянии клиента, которая может быть утеряна);

производительность (за счёт использования кэша);

масштабируемость;

прозрачность системы взаимодействия;

простота интерфейсов;

портативность компонентов;

лёгкость внесения изменений;

способность эволюционировать, приспосабливаясь к новым требованиям.

REST принципы

Как было сказано выше, REST определяет, как компоненты распределенной системы должны взаимодействовать друг с другом. В общем случае это происходит посредством запросов-ответов. Компоненту, которая отправляет запрос называют клиентом; компоненту, которая обрабатывает запрос и отправляет клиенту ответ, называют сервером. Запросы и ответы, чаще всего, отправляются по протоколу HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста»). Как правило сервер — это некое веб-приложение. Клиентом же может быть не то чтобы что угодно, но довольно многое. Например, мобильное приложение, которое запрашивает у сервера данные. Либо браузер, который отправляет запросы с веб-страницы на сервер для загрузки данных. Приложение А может запрашивать данные у приложения Б. Тогда А является клиентом по отношению к Б, а Б — сервером по отношению к А. Одновременно с этим, А может обрабатывать запросы от В, Г, Д и т.д. В таком случае, приложение А является одновременно и сервером, и клиентом. Все зависит от контекста. Однозначно одно: компонента которая шлет запрос — это клиент. Компонента, которая принимает, обрабатывает и отвечает на запрос — сервер.

REST ограничения

Однако не каждая система, чьи компоненты обмениваются данными посредством запросов-ответов, является REST (или же RESTful) системой. Чтобы система считалась RESTful, она должна «вписываться» в шесть REST ограничений:

1. Приведение архитектуры к модели клиент-сервер

В основе данного ограничения лежит разграничение потребностей. Необходимо отделять потребности клиентского интерфейса от потребностей сервера, хранящего данные. Данное ограничение повышает переносимость клиентского кода на другие платформы, а упрощение серверной части улучшает масштабируемость системы. Само разграничение на «клиент» и «сервер» позволяет им развиваться независимо друг от друга.

2. Отсутствие состояния

Архитектура REST требует соблюдения следующего условия. В период между запросами серверу не нужно хранить информацию о состоянии клиента и наоборот. Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Таким образом и сервер, и клиент могут «понимать» любое принятое сообщение, не опираясь при этом на предыдущие сообщения.

3. Кэширование

Клиенты могут выполнять кэширование ответов сервера. У тех, в свою очередь, должно быть явное или неявное обозначение как кэшируемых или некэшируемых, чтобы клиенты в ответ на последующие запросы не получали устаревшие или неверные данные. Правильное использование кэширования помогает полностью или частично устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и расширяемость системы.

4. Единообразие интерфейса

К фундаментальным требованиям REST архитектуры относится и унифицированный, единообразный интерфейс. Клиент должен всегда понимать, в каком формате и на какие адреса ему нужно слать запрос, а сервер, в свою очередь, также должен понимать, в каком формате ему следует отвечать на запросы клиента. Этот единый формат клиент-серверного взаимодействия, который описывает, что, куда, в каком виде и как отсылать и является унифицированным интерфейсом

5. Слои

Под слоями подразумевается иерархическая структура сетей. Иногда клиент может общаться напрямую с сервером, а иногда — просто с промежуточным узлом. Применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузки и распределённого кэширования. Приведем пример. Представим себе некоторое мобильное приложение, которое пользуется популярностью во всем мире. Его неотъемлемая часть — загрузка картинок. Так как пользователей — миллионы человек, один сервер не смог бы выдержать такой большой нагрузки. Разграничение системы на слои решит эту проблему. Клиент запросит картинку у промежуточного узла, промежуточный узел запросит картинку у сервера, который наименее загружен в данный момент, и вернет картинку клиенту. Если здесь на каждом уровне иерархии правильно применить кэширование, то можно добиться хорошей масштабируемости системы.

6. Код по требованию (необязательное ограничение)

Данное ограничение подразумевает, что клиент может расширять свою функциональность, за счет загрузки кода с сервера в виде апплетов или сценариев.

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.

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).

Объясните понятие WSDL.

WSDL (англ. Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

Каждый документ WSDL 1.1 можно разбить на следующие логические части:

определение типов данных (types) — определение вида отправляемых и получаемых сервисом XML-сообщений

элементы данных (message) — сообщения, используемые web-сервисом

абстрактные операции (portType) — список операций, которые могут быть выполнены с сообщениями

связывание сервисов (binding) — способ, которым сообщение будет доставлено

Что такое JAX-WS?

Java API for XML Web Services (JAX-WS) — это прикладной программный интерфейс языка Java для создания веб-служб, являющийся частью платформы Java EE. JAX-WS является заменой технологии JAX-RPC, предоставляя более документо-ориентированную модель сообщений и упрощая разработку[1] веб-служб за счёт использования аннотаций, впервые появившихся в Java SE 5. Технология JAX-WS является стандартом и описана в JSR 224.

Некоторые преимущества:

Использование аннотаций устраняет необходимость создания дескрипторов веб-служб.

Декларация конечных точек (endpoints) происходит непосредственно в классах Java.

Прямая интеграция с JAXB 2.0.

Внедрение ресурсов (Resource injection).

Поддержка MTOM.

Возможность выбора между двумя путями разработки: снизу-вверх (программист разрабатывает endpoint-классы сам) и сверху-вниз (Java классы генерируются по WSDL).

Расскажите о JAXB.

Java Architecture for XML Binding (JAXB) позволяет Java разработчикам ставить в соответствие Java классы и XML представления. JAXB предоставляет две основные возможности: сериализация Java объектов в XML и наоборот, то есть десериализация из XML обратно в Java объект. Другими словами, JAXB позволяет хранить и извлекать данные в памяти в любом XML-формате, без необходимости выполнения определенного набора процедур загрузки и сохранения XML. Он похож на xsd.exe и XmlSerializer в .NET Framework.

JAXB особенно полезен, когда спецификация является сложной и меняющейся. В этом случае, постоянные изменения схемы XML определений для синхронизации их с определениями Java могут занять много времени и быть подвержены ошибкам.

Что такое XOP?

XOP (XML-binary Optimized Packaging) — механизм, рекомендованный W3C для встраивания двоичных данных в набор информационных элементов XML (XML Information Set).

Что вы знаете о кодировании в SOAP (encoding)?

Кодирование SOAP представляет собой метод для структурирования запроса, который предлагается в рамках спецификации SOAP, известный как SOAP-сериализация.

Что определяет атрибут encodingStyle в SOAP?

SOAP encodingStyle определяет правила сериализации, используемые в сообщении SOAP. Этот атрибут может появиться на любом элементе, и область видимости этого атрибута будет распространяться на все дочерние элементы, даже на те, которые не имеют явно этого атрибута. Для сообщений SOAP по умолчанию кодирование не определено.

SOAP-ENV:encodingStyle=»http://www.w3.org/2001/12/soap-encoding»

Какие два конечных типа веб сервисов используют JAX-WS?

RPC (remote procedure call) style web service в JAX-WS;

document style web service в JAX-WS.

Какие существуют правила для кодирования записи header?

заголовок должен быть идентифицирован с помощью полного имени, которое содержит пространство имен URI и локальное имя. Все непосредственные дочерние элементы SOAP заголовка должны быть заданы в пространстве имен,

атрибут SOAP encodingStyle должен использоваться для указания стиля кодирования заголовка,

атрибут SOAP mustUnderstand и атрибут SOAP actor должны использоваться для указания того, как обрабатывать запись и кем.

Что вы знаете об инструменте wsimport?

Инструмент wsimport используется для синтаксического анализа существующих Web Services Description Language (WSDL-файл) и генерации необходимых файлов (JAX-WS портируемые артефакты) для клиента веб-сервиса для доступа к опубликованному веб-сервису.

Что вы знаете об инструменте wsgen?

Инструмент wsgen используется для анализа существующего класса реализации веб-службы и создает необходимые файлы (JAX-WS портируемые артефакты) для развертывания веб-служб.

Какие вы можете выделить различия между SOAP и другими техниками удаленного доступа?

SOAP проще в использовании, т.к. он не симметричный вроде DCOM или COBRA,

SOAP является более независимым от платформы и языка в отличие от DCOM или CORBA,

SOAP использует HTTP в качестве транспортного протокола и данные сохраняются в формате XML, который может быть прочтен человеком, тогда как DCOM или CORBA имеют свои собственные бинарные форматы, которые используются для транспортировки данных сложным образом.

SOAP идентифицирует объект, отличный от конечного URL. Объекты SOAP являются независимыми и их сложно поддерживать. В случае других методов удаленного доступа работа в этом случае может быть проще.

Что такое resource в REST?

Это уникальный URL с представлением объекта, который может быть получен с помощью запросов GET и изменен с помощью PUT, POST, DELETE.

Какие HTTP методы поддерживаются в REST?

GET; POST; PUT; DELETE; OPTIONS; HEAD.

Когда можно использовать GET запрос вместо POST для создания ресурса?

Невозможно использовать GET запрос для изменения (создания) ресурса.

Какая разница между GET и POST запросами?

GET передает данные серверу используя URL, когда POST передает данные, используя тело HTTP запроса.

Длина URL'а ограничена 1024 символами, это и будет верхним ограничением для данных, которые можно отослать GET'ом.

POST может отправлять гораздо большие объемы данных. Лимит устанавливается веб-сервером и обычно равен около 2MB.

Передача данных методом POST более безопасна, чем методом GET, так как секретные данные (например пароль) не отображаются напрямую в web-клиенте пользователя (в отличии от URL, который виден почти всегда).

Что означает WADL?

Web Application Description Language (WADL) — машинно-читаемое XML-описание для web-приложений HTTP (как правило, веб-сервисы REST). Аналог WSDL для SOAP.

WADL моделирует ресурсы, предоставляемые сервисом, и взаимосвязи между ними. WADL был предложен как стандарт W3C компанией Sun Microsystems в августе 2009, но консорциум не имеет никаких планов насчёт него и WADL ещё не получил широкого применения.

Какие вы знаете фреймворки, которые реализуют REST веб сервисы?

Их много, вот некоторые из них: Jersey, Restlet, EasyRest.

Какая разница между AJAX и REST?

В AJAX запрос посылается к серверу с помощью объектов XMLHttpRequest. В REST используется структура URL и использование ресурсов вращается вокруг шаблона запрос\ответ.

AJAX асинхронно исключает взаимодействие между клиентом и сервером, в то время как REST требует взаимодействия между клиентом и сервером.

AJAX технология «set». REST - предоставляет методы для пользователей для запроса данных или информации от сервера.

Что такое JPA?

JPA - это технология, обеспечивающая объектно-реляционное отображение простых JAVA объектов и предоставляющая API для сохранения, получения и управления такими объектами.

JPA - это спецификация (документ, утвержденный как стандарт, описывающий все аспекты технологии), часть EJB3 спецификации.

Сам JPA не умеет ни сохранять, ни управлять объектами, JPA только определяет правила игры: как что-то будет действовать. JPA также определяет интерфейсы, которые должны будут быть реализованы провайдерами. Плюс к этому JPA определяет правила о том, как должны описываться метаданные отображения и о том, как должны работать провайдеры. Дальше, каждый провайдер, реализуя JPA определяет получение, сохранение и управление объектами. У каждого провайдера реализация разная.

Реализации JPA:

Hibernate

Oracle TopLink

Apache OpenJPA

Из чего состоит JPA?

JPA состоит из трех основных пунктов:

API - интерфейсы в пакете javax.persistance. Набор интерфейсов, которые позволяют организовать взаимодействие с ORM провайдером.

JPQL - объектный язык запросов. Очень похож на SQL, но запросы выполняются к объектам.

Metadata - аннотации над объектами. Набор аннотаций, которыми мы описываем метаданные отображения. Тогда уже JPA знает какой объект в какую таблицу нужно сохранить. Метаданные можно описывать двумя способами: XML-файлом или через аннотации.

В чем её отличие JPA от Hibernate?

Hibernate одна из самых популярных открытых реализаций последней версии спецификации (JPA 2.1). 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).

В чем её отличие JPA от JDO?

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. Также отличаются «разработчики» спецификаций - если JPA разрабатывается как JSR, то JDO сначала разрабатывался как JSR, теперь разрабатывается как проект Apache JDO.

Можно ли использовать JPA с noSQL базами?

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.

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

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. 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.

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

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).

Что такое Criteria API и для чего он используется?

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

Что такое Entity?

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.

Может ли не Entity класс наследоваться от Entity класса?

Может.

Может ли Entity класс наследоваться от других Entity классов?

Может.

Может ли Entity быть абстрактным классом?

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

Может ли Entity класс наследоваться от не Entity классов (non-entity classes)?

Может.

Какие требования JPA к Entity классам вы можете перечислить (не менее шести требований)?

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) Entity класс должен содержать первичный ключ, то есть атрибут или группу атрибутов которые уникально определяют запись этого Entity класса в базе данных.

Что такое атрибут Entity класса в терминологии JPA?

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.

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

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.

Какие типы данных допустимы в атрибутах Entity класса (полях или свойствах)?

Допустимые типы атрибутов у Entity классов:

примитивные типы и их обертки Java,

строки,

любые сериализуемые типы Java (реализующие Serializable интерфейс),

enums;

entity types;

embeddable классы

и коллекции типов 1-6

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)?

Допустимые типы атрибутов, входящих в первичный ключ:

примитивные типы и их обертки Java,

строки,

BigDecimal и BigInteger,

java.util.Date и java.sql.Date,В случае автогенерируемого первичного ключа (generated primary keys) допустимы 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).

Что такое встраиваемый (Embeddable) класс?

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.

Может ли встраиваемый (Embeddable) класс содержать другой встраиваемый (Embeddable) класс?

Да, может.

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)?

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

Какие требования JPA устанавливает к встраиваемым (Embeddable) классам?

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

2. Embeddable класс должен быть отмечен аннотацией Embeddable или описан в XML файле конфигурации JPA.

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)?

Существуют следующие четыре типа связей

OneToOne (связь один к одному, то есть один объект Entity может связан не больше чем с один объектом другого Entity),

OneToMany (связь один ко многим, один объект Entity может быть связан с целой коллекцией других Entity),

ManyToOne (связь многие к одному, обратная связь для OneToMany),

ManyToMany (связь многие ко многим).

Каждую из которых можно разделить ещё на два вида:

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.

Что такое Mapped Superclass?

Mapped Superclass это класс от которого наследуются Entity, он может содержать аннотации 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.

Какие два типа fetch стратегии в JPA вы знаете?

В JPA описаны два типа fetch стратегии:

LAZY - данные поля будут загружены только во время первого доступа к этому полю,

EAGER - данные поля будут загружены немедленно.

Какие три типы стратегии наследования мапинга (Inheritance Mapping Strategies) описаны в JPA?

В JPA описаны три стратегии наследования мапинга (Inheritance Mapping Strategies), то есть как JPA будет работать с классами-наследниками Entity:

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 соответственно.

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.

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 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.

Что такое EntityManager и какие основные его функции вы можете перечислить?

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:

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),

Получение данных: find (поиск и получение Entity), createQuery, createNamedQuery, createNativeQuery, contains, createNamedStoredProcedureQuery, createStoredProcedureQuery

Получение других сущностей JPA: getTransaction, getEntityManagerFactory, getCriteriaBuilder, getMetamodel, getDelegate

Работа с EntityGraph: createEntityGraph, getEntityGraph

Общие операции над EntityManager или всеми Entities: close, isOpen, getProperties, setProperty, clear.

Какие четыре статуса жизненного цикла Entity объекта (Entity Instance's Life Cycle) вы можете перечислить?

У Entity объекта существует четыре статуса жизненного цикла: new, managed, detached, или removed. Их описание

new - объект создан, но при этом ещё не имеет сгенерированных первичных ключей и пока ещё не сохранен в базе данных,

managed - объект создан, управляется JPA, имеет сгенерированные первичные ключи,

detached - объект был создан, но не управляется (или больше не управляется) JPA,

removed - объект создан, управляется JPA, но будет удален после commit'а транзакции.

Как влияет операция merge на Entity объекты каждого из четырех статусов?

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,

2. Если статус Entity new, то будет создана новый managed entity, в который будут скопированы данные прошлого объекта,

3. Если статус managed, операция игнорируется, однако операция merge сработает на каскадно зависимые Entity, если их статус не managed,

4. Если статус removed, будет выкинут exception сразу или на этапе commit'а транзакции.

Как влияет операция remove на Entity объекты каждого из четырех статусов?

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'е 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'а транзакции.

Как влияет операция persist на Entity объекты каждого из четырех статусов?

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.

Как влияет операция refresh на Entity объекты каждого из четырех статусов?

1. Если статус Entity managed, то в результате операции будут востановленны 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.

Как влияет операция detach на Entity объекты каждого из четырех статусов?

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.

Для чего нужна аннотация Access?

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).

Для чего нужна аннотация Basic?

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.

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?

Для такого перекрывания существует четыре аннотации:

AttributeOverride чтобы перекрыть поля, свойства и первичные ключи,

AttributeOverrides аналогично можно перекрыть поля, свойства и первичные ключи со множественными значениями,

AssociationOverride чтобы перекрывать связи (override entity relationship),

AssociationOverrides чтобы перекрывать множественные связи (multiple relationship).

Какие аннотации служит для задания класса преобразования basic атрибута Entity в другой тип при сохранении/получении данных их базы (например, работать с атрибутом Entity boolean типа, но в базу сохранять его как число)?

Convert и Converts - позволяют указать класс для конвертации Basic атрибута 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.

Какой аннотацией можно управлять кешированием JPA для данного Entity?

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 изменится и для них тоже.

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.)?

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

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).

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): PrePersist PostPersist PreRemove PostRemove PreUpdate PostUpdate PostLoad

Какой аннотацей можно исключить поли и свойства Entity из маппинга (property or field is not persistent)?

Для этого служит аннотация Transient.

Какие аннотации служить для установки порядка выдачи элементов коллекций Entity?

Для этого служит аннотация OrderBy и OrderColumn.

Какие шесть видов блокировок (lock) описаны в спецификации JPA (или какие есть значения у enum LockModeType в JPA)?

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

NONE - без блокировки

OPTIMISTIC (или синоним READ, оставшийся от JPA 1) - оптимистическая блокировка

OPTIMISTIC_FORCE_INCREMENT (или синоним WRITE, оставшийся от JPA 1) - оптимистическая блокировка с принудительным увеличением поля версионности

PESSIMISTIC_READ - пессимистичная блокировка на чтение

PESSIMISTIC_WRITE - пессимистичная блокировка на запись (и чтение)

PESSIMISTIC_FORCE_INCREMENT - пессимистичная блокировка на запись (и чтение) с принудительным увеличением поля версионности.

Какие два вида кэшей (cache) вы знаете в JPA и для чего они нужны?

JPA говорит о двух видов кэшей (cache):

first-level cache (кэш первого уровня) - кэширует данные одной транзакции,

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.

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?

JPA говорит о пяти значениях shared-cache-mode из persistence.xml, который определяет как будет использоваться second-level cache:

ALL - все Entity могут кэшироваться в кеше второго уровня

NONE - кеширование отключено для всех Entity

ENABLE_SELECTIVE - кэширование работает только для тех Entity, у которых установлена аннотация Cacheable(true) или её xml эквивалент, для всех остальных кэширование отключено

DISABLE_SELECTIVE - кэширование работает для всех Entity, за исключением тех у которых установлена аннотация Cacheable(false) или её xml эквивалент

UNSPECIFIED - кеширование не определенно, каждый провайдер JPA использует свою значение по умолчанию для кэширования

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?

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:

javax.persistence.fetchgraph - все атрибуты перечисленные в EntityGraph меняют fetchType на EAGER, все остальные на LAZY

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.

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

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

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.)?

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.

В чем разница в требованиях к Entity в Hibernate, от требований к Entity, указанных в спецификации JPA?

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))

Какая уникальная стратегия наследования есть в Hibernate, но нет в спецификации JPA?

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