LaTeX

1. Компоненты в Angular: взаимодействие с представлениями и сервисами
2. Инициализация Spring Beans
1. В модулях определяются компоненты — строительные блоки интерфейса, инкапсулирующие верстку, стили, и логику приложения, которые можно переиспользовать внутри других компонентов. Любой компонент состоит из TS класса, помеченного декоратором @Component, HTML шаблона, CSS стилей. Компонент вместе с шаблоном образуют представление. Представления образуют иерархию. Существует двухсторонняя связь между классом компонента и представлением — при изменении данных в компоненте обновляется представление и наоборот. Задачи приложения, которые не касаются представления, выносятся в сервисы. Для того чтобы созданный сервис мог быть использован компонентом или другим сервисом, нужно пометить @Injectable() сервис. Так как они создаются именно для стороннего использования, то рекомендуется всегда использовать декоратор. Все сервисы регистрируются Injector-ом, который является частью механизма DI в Angular.
2. Подходы: Метод бина с аннотацией @PostConstruct, не позволяет вводить параметры; Метод afterPropertiesSet() бина реализующего интерфейс InitializingBean; @Bean(initmethod="somInitMethod"). Этот подход не допускает входных параметров. В xml-конфигурации можно установить для всех бинов сразу, с помощью default-init-method. Могут быть использованы в JSF путём конфигурации в faces-config.xml. Подход Spring Boot: Есть интерфейсы CommandLineRunner и ApplicationRunner, оба из которых будут работать после создания ApplicationContext, оба из них позволяют вводить бины в качестве входных параметров. ApplicationListener позволяет прослушивать стандартные события, связанные с жизненным циклом контекста, а также кастомные события.

1. Реализация Ajax в JSF
2. CDI beans: контекст (Bean Scope)
1. 1 способ: JavaScript API - jsf.ajax.request(); event - событие, по которому отправляется AJAX-запрос ; execute - компоненты, обрабатываемые в цикле обработки запроса ; render - перерисовываемые компоненты. <h:commandButton id="submit"value="submit"onclick="jsf.ajax.request(this, event, {execute: 'myinput', render: 'outtext'});/>. 2 способ: <h:commandButton id='submit', value='submit'> <f:ajax execute="@form"render="msg"/></h:commandButton>

2. Определяет жизненный цикл бинов и их видимость друг для друга. @RequestScoped - создается для каждого HTTP-запроса, существует во время обработки одного HTTP-запроса и уничтожается после его завершения; @ViewScoped - контекст-страница(компонент создается один раз при обращении к странице, и используется ровно столько, сколько пользователь находится на странице); из JSF, но тоже работает; @SessionScoped - связан с HTTP-сессией пользователя, существует, пока существует HTTP-сессия; @ApplicationScoped - создается один раз для всего приложения, существует в течение жизни приложения; @ConversationScoped - Связан с определенной беседой, которая состоит из последовательности запросов и ответов. Областью жизни управляет программист. Управление осуществляется через инъекцию объекта javax.enterprise.context.Conversation; @Dependent - Используется по умолчанию. ЖЦ определяется тем где он был использован; Экземпляры бина зависят от контекста, в котором они были внедрены.

1. Spring Web MVC: View Resolvers
2. Angular: ключевые особенности, отличия от AngularJS
1. View Resolver — интерфейс, реализуемый объектами, которые способны находить представление по его имени. С помощью него Dispatcher Servlet находит нужный View. Представление в Spring Web MVC может быть построено на разных технологиях. С каждым представлением сопоставляется его символическое имя. Преобразованием символических имён в ссылки на конкретные представления занимается специальный класс, реализующий интерфейс org.springframework.web.servlet.ViewResolver. В одном приложении можно использовать несколько ViewResolver'ов <bean id="viewResolver"class="o.s.w.s.v.UrlBasedViewReso - lver» <property name="viewClass"value="o.s.w.s.v.Jstl - View"/> <property name="prefix"value="/WEB-INF/jsp/"/> <property name="suffix«/bean>
2. Angular - написан на TypeScript, развитие AngularJS особенности: кроссплатформенное; для разработки надо настроить сборочное окружение; приложение состоит из модулей (NgModules); модули обеспечивают контекст для компонентов; из компонентов строятся представления; компоненты взаимодействуют с сервисами через DI; Angular как и AngularJS реализуют модель MVVM. В AngularJS жесткие рамки для компонентов; есть иерархия компонентов; гораздо менее безопасен и управляем; задействует JavaScript, Angular же использует TypeScript. Angular адаптирован под слабые мобильные устройства.

1. Фаза получения значений компонентов (Apply Request Values Phase)
2. Реализация IoC и DI в Spring
1. Данная фаза идёт после фазы формирования представления и до фазы валидации значений компонентов. На стороне клиента все значения хранятся в строковом формате, поэтому нужна проверка их корректности: Вызывается конвертер в соответствии с типом данных значения. Если конвертация - успешно, значение сохраняется в локальной переменной компонента. Если конвертация заканчивается неудачно, создаётся сообщение об ошибке, которое помещается в FacesContext.
2. Инверсия управления — принцип для уменьшения связности кода. Самописным кодом управляет фреймворк, занимающийся жизненным циклом компонентов и коммуникацией между ними. Реализуется посредством внедрения зависимостей. Внедрение зависимостей - паттерн проектирования, который позволяет создавать объект, использующий другие объекты. При этом поля объекта настраиваются внешней сущностью. Позволяет убрать зависимость компонента от контейнера. ApplicationContext контейнер, который тоже управляет жизненным циклом компонентов и коммуникацией между ними и также реализуется посредством внедрения зависимостей, которые реализуется через аннотации. @Autowired - внедрение. Spring IoC контейнеру требуются метаданные для конфигурации. Для этого классы помечаются аннотацией @Component, а также её наследниками @Repository, @Service и @Controller. @Configuration, @Bean - позволяют определить бины и их конфигурацию программно. В старых версиях Spring DI можно было настроить с использованием XML-файлов, определяя бины и их зависимости в контексте приложения. Без XML с помощью кода с использованием классов, помеченных @Configuration, и методов @Bean, которые возвращают экземпляры бинов.

1. Фазы jsf: Invoke Application и Render Response
2. Способы задания конфигурации в Spring
1. При поступлении запроса необходимо выполнить определенную цепочку действий, чтобы проанализировать запрос и подготовить ответ. За программиста это делает фреймворк(JSF) Invoke Application Phase: Управление передаётся слушателям событий, JSF обрабатывает события также решается вопрос навигации; Формируются новые значения компонентов. (Вызывается метод UIViewRoo#processApplication() для обработки событий.) Render Response Phase: JSF Runtime обновляет представление в соответствии с результатами обработки запроса; Если это первый запрос к странице, то компоненты помещаются в иерархию представления; Формируется ответ сервера на запрос; На стороне клиента происходит обновление страницы.
2. Два способа конфигурации: xml — устаревший вариант; annotations — при помощи «сканирования»; XML-файл, в котором прописываются все бины, путь до класса, свойства, конструкторы. Загружается этот файл из classpath. Создаем класс с аннотацией Configuration, внутри помечаем методы аннотацией Bean; (annotation based config) добавляя к классам аннотации Component. При сканировании выполняется поиск бинов, помеченных @Component и @Configuration. @ComponentScan — для указания пакетов, в которых нужно выполнить сканирование. Внедрение зависимостей происходит через аннотацию Autowired; Groovy config; Property files; 2 и 3 - java config;

1. Профили и платформы Java EE
2. Типы DI в Spring
1. Web Profile — содержит в себе только те компоненты, которые нужны для работы веб приложения, это Servlet, JSP, JSF, JPA, CDI, EJB. Full Profile — полный сборник джавы ее, в нем есть еще JAX-RS, JAX-WS, JAXB, JNDI, JAVA MAIL. Платформы: JME- представляет из себя API и минимально требовательную VM для разработки и старта приложения на смартфоне/планшете. JSE- занимается обеспечением основными стандартными функциями самой Java, и она определяет: базовые типы и объекты языка, классы более высокого уровня, производительность приложения в сети и обеспечение защищенности. JEE- для разработки Enterprise приложений. Она строится на основе платформы JSE, а еще дает возможность разработки более крупно масштабируемых, сложно уровневых и безопасных программ. Содержит: WebSocket, JSF, Unified EL, API для вебслужб RESTful, DI, EJB, JPA, и Java Transaction API.
2. Constructor-based DI - контейнер вызовет конструктор с аргументами бинов, которые потом заинджектятся в класс; Setter-based DI - сначала контейнер вызовет конструктор бина без аргументов, после вызовет помеченные аннотациями @Autowired сеттеры и впихнет туда нужные зависимости. До сих пор остается местом споров. Это связано с другой особенностью Spring Boot — Ioc. Программист сам решает когда вызывать ту или иную процедуру, делать DI и т.п. В Spring Boot — это делает IoC — инициализация и вызовы процедур в Runtime. Получается, что используя DI с помощью «сеттера» вы не можете знать в какой именно момент вы зависимость будет внедрена; Field-based DI - контейнер через рефлексию будет в поля класса пропихивать зависимости. Используется редко по причине нарушения инкапсуляции, ведь внедряемое поле должно быть помечено как public.

1. Spring MVC: handler mapping
2. JSX. Применение в реакте. Пример синтаксиса
1. Механизм, позволяющий распределять запросы по различным обработчикам. Помимо «основного» Handler'а, в обработке запроса могут участвовать один или несколько «перехватчиков» (реализаций интерфейса HandlerInterceptor). Когда DispatcherServlet получает запрос, он на основании конфигурации HandlerMapping выбрать на какой контроллер пойдет запрос. Этот mapping - механизм, позволяющий распределять запросы по различным обработчикам. Механизм в общем похож на сервлеты и фильтры. Из коробки программисту доступно несколько реализаций Handler Mapping.
2. React представляет собой дерево из компонентов. Точкой входа (корнем) являет index.js который определяет компоненты. Каждый компонент включает в себя другие компоненты. В React приложениях разметка пишется в JSX файлах. JSX — надстройка над JS, которая позволяет вкраплять HTML-синтаксис в код. Можно использовать стандартные HTML элементы (такие как div, span, h1, input) так и кастомные React компоненты. function warningButton(){return <CustomButton color="кровиВТшников"/>}; JSX код: text компилируется в вызов функции React.createElement("div", className: "foo", "text")

1. MVC в JSF
2. Главные аннотации в CDI beans java EE
1. Model — бины, в которых содержится бизнес логика, данные и методы работы с данными. public class User {private String username;private String email;} View — xhtml шаблон в котором формируется дерево компонентов. Компоненты могут взаимодействовать с бинами, вызывать их методы или получать из них данные, тем самым передавая их пользователю. <h:form><h:inputText value="#{user.username}"/> <h:commandButton value="Сохранить"action="#{userController.saveUser}" </h:form> Controller — реализуется самим фреймворком. Это класс FacesServlet, который занимается диспетчеризацией и управлением жизненным циклом. @ManagedBean@RequestScoped public class UserController {private User user;public UserController() {user = new User();} public String saveUser() {}}
2. @RequestScoped - контекст - запрос; @ViewScoped - контекст-страница; @SessionScoped - контекст - сессия; @ApplicationScoped - контекст - приложение; @ConversationScoped - Областью жизни управляет программист. Управление осуществляется через инъекцию объекта context.Conversation; @Dependent. Используется по умолчанию. ЖЦ определяется тем где он был использован; @Produces/@Disposes, бины - фабрики которые управляют экземплярами других бинов; @Informal — бин-наследник; @Inject - используется для указания точки внедрения зависимости. Инъекции могут происходить в поле, в метод и в конструкторе; @Named - используется, для того, чтобы выдать имя бину, тогда его можно будет использовать на jsf странице; @Qualifier - аннотация, которая используется для создания аннотаций-спецификаторов, которые четко указывают, какой бин надо инжектить. Над классом ставится аннотация для указания квалификатора бина. Над точкой внедрения ставится такая же аннотация; @Default/Alternative — управляет выбором бина при наличии нескольких
1. Структура JSF приложения
2. Spring MVC: особенности, интеграция в Spring
1. JSP или XHTML - страницы содержащие компоненты GUI. JSP или XHTML представляют из себя обычный HTML, но со своими тэгами и префиксами. Для этого в стандартной структуре jsf есть: Библиотека тэгов - они описывают эти дополнительные тэги jsf; Управляемые бины - бины управляемые рантаймом jsf (контейнером), чем является faces Servlet; Дополнительные объекты (компоненты, конверторы и валидаторы); Дополнительные тэги; web.xml - Файл развёртывания, где определяются параметры и настройки контейнера сервлетов, а также связи сервлетов JSF; faces-config.xml - Конфигурационный файл JSF, где определяются бины, навигация, управляемые бины, ресурсы и другие параметры приложения JSF.
2. Spring Web MVC – фреймворк в составе Spring для разработки веб-приложений. Основан на паттерне, который делит предложение компоненты на модель (Model), представление (View) и контроллер (Controller). Model инкапсулирует данные приложения, в целом они будут состоять из POJO. View отвечает за отображение данных фрэймворк не специфицирует жестко технологию на которой будет построено представление. По умолчанию JSP. Controller обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения на View. Back-end; универсальный, удобен для разработки REST API. На клиентской стороне интегрируется с популярными JS-фреймворками. Удобно интегрируется с Thymeleaf. Можно настраивать Spring MVC как через XML-конфигурации, так и через Java-конфигурацию с использованием классов конфигурации @Configuration. Spring MVC использует множество аннотаций, таких как @Controller, @RequestMapping, @RequestParam, @ResponseBody и другие, для обозначения компонентов и определения маппингов URL.

1. Spring MVC: обработка запросов, DispatcherServlet
2. Single Page Application(SPA): преимущества, недостатки
1. Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы и ответы на них. При получении запроса, происходит следующая цепочка событий: DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет какой контроллер должен быть вызван. Контроллер принимает запрос и вызывает соответствующий служебный метод (GET, POST), который возвращает в диспатчер имя View. При помощи ViewResolver диспатчер определяет, какой View надо использовать на основании полученного имени. После того, как View создан, диспатчер отправляет данные модели в виде атрибутов, которые уже в конечном итоге отображаются в браузере.
2. Веб-приложение, использующее единственный HTML-документ как оболочку для всех веб-страниц и динамически подгружает HTML, CSS, JS, обычно посредством AJAX. За навигацию отвечает JS. Клиент и сервер реализуются независимо и взаимодействуют по REST(обычно JSON). Преимущества: легкость создания из-за огромного количества готовых библиотек и фреймворков; Простое кэширование данных; Скорость работы, основная часть ресурсов уже загружена, на страничку подгружаются только необходимые данные. Недостатки: Тяжелые клиентские фреймворки; Без JS невозможно пользоваться полным функционалом приложения; Недоступна SEO оптимизация. Загрузка больших объемов JavaScript-кода может отрицательно сказаться на производительности и использовании ресурсов устройства клиента, особенно на мобильных устройствах.

1. Платформы Java. Сходства и различия.
2. Двухфазные и трехфазные конструкторы в Spring и Java EE
1. Java Micro Edition представляет из себя API и минимально требовательную VM для разработки и старта приложения на смартфоне. Основана на более ранней версии JSE, поэтому некоторые функции не работают Java Standard Edition занимается обеспечением основными стандартными функциями самой Java, и она определяет абсолютно все: базовые типы и объекты языка, классы более высокого уровня, производительность приложения в сети и обеспечение защищенности. Java Enterprise Edition для разработки Enterprise приложений. Она строится на основе платформы JSE, а еще дает возможность разработки более крупно масштабируемых, сложно уровневых и безопасных программ. Содержит: WebSocket, JSF, Unified EL, API для RESTful, DI, EJB, JPA, и Java Transaction API. Все платформы Java поддерживают полный функционал языка Java и отличаются лишь наличием или отсутствием определенных API.
2. Двухфазовые: обычный конструктор + метод с аннотацией @PostConstruct. Сначала вызовется обычный конструктор, а затем помеченный метод. На момент его вызова все зависимости будут обработаны и доступны. Трехфазовый: сначала был нативный конструктор, потом обработались зависимости, уже с обработанными зависимостями вызвалось конструирование объекта, а потом добавились срезы. Аспекты - это “вкрапления”, которые позволяют добавить поведение до/после вызова оригинального метода через Proxy Такой конструктор имеет свой скоуп.

1. Технология RMI. Использование RMI в Java EE
2. Управление состоянием в React. Flux and Redux
1. Система RMI позволяет объекту, запущенному на одной виртуальной машине Java, вызывать методы объекта, запущенного на другой виртуальной машине Java. Работает поверх TCP. В общем случае, объекты передаются по значению, передаваемые объекты должны быть Serializable. RMI основана на более ранней технологии удаленного вызова процедур Remote Procedure Call (RPC). Использование: Серверное приложение, как правило, создает удаленные объекты (remote objects), делает доступные ссылки на эти объекты и находится в ожидании вызова методов этих объектов. Клиентское приложение получает у сервера ссылку на удаленные объекты, после чего вызывает его методы. Технология RMI, обеспечивающая механизм взаимодействия клиента и сервера передачей между ними соответствующей информацией, реализована в виде java.rmi пакета, содержащего целый ряд вложенных подпакетов; один из наиболее важных подпакетов java.rmi.server реализует функции сервера RMI. RMI обеспечивает процесс преобразования информации данных по сети и позволяет java приложениям передавать объекты с помощью механизма сериализации объектов.
2. Хук useState() предназначен для управления локальным состоянием компонента. Хук «useContext()» позволяет извлекать значения из контекста в любом компоненте, обернутом в провайдер (provider). Flux — архитектура для создания приложений на React, в которой описывается, как хранить, изменять и отображать глобальные данные. Основные концепции: Dispatcher принимает события от представления и отправляет их на обработку хранилищу данных. Store знает, как менять данные. Напрямую из React-компонента их изменить нельзя. После изменения данных Store посылает события представлению, и оно перерисовывается. Redux — небольшая библиотека, реализующая упрощенный паттерн Flux. В Redux есть store — синглтон, хранилище состояние всего приложения. Изменения состояния производятся при помощи чистых функций. Они принимают на вход state и действиеи возвращают либо неизмененный state либо копию

1. Валидаторы в JSF. Создание, назначение и тд.
2. Реализация контроллера в Spring Web MVC
1. Валидаторы в JSF — реализации интерфейса Validator. Метод validate принимает FacesContext, UiComponent и значение. Осуществляется перед обновлением значения компонента на уровне модели. Класс, осуществляющий валидацию, должен реализовывать интерфейс javax.faces.validator.Validator. Существуют стандартные валидаторы для основных типов данных. DoubleRangeValidator, LengthValidator, RegexValidator, RequiredValidator. Собственные валидаторы с помощью аннотации @FacesValidator. Создание: параметры компонента, вложенный тег, <f:validatorvalidatorId=”com.example.MyValidator”/>, логика на уровне управлямого бина
2. Перехватывает входящие запросы, упаковывает данные в нужный формат, отправляет эти данные нужной модели, а затем ответ от модели передать обратно в DispatcherServlet. На методы контроллера лепятся разные аннотации(@GetMapping, @PostMapping, @PathVariable. @ResourcesVarible). @Controller ℵ public class HelloController { ℵ @RequestMapping(value = "/hello method = RequestMethod.GET)ℵ public String printHello(ModelMap model) {ℵ model.addAttribute("message "Hello Spring MVC Framework!");ℵ return "hello";}}

1. JNDI. JNDI в Java EE. Способы взаимодействия с JNDI. Их преимущества и недостатки.
2. React. Особенности. Архитектура
1. JNDI — API для доступа к объектам и ресурсам по их именам. Организовано в виде службы имен и каталогов. Чаще всего используется в enterprise. Главный юзкейс — настройка доступа к базе данных. Приложение знает только JNDI-имя, а сами детали подключения описываются администратором в веб контейнере. JNDI поддерживает разные реализации сервиса служб имен и каталогов. Некоторые из них: DNS, RMI, LDAP, COBRA. Преимущества JNDI: пароли к бд лежат отдельно от приложения; при изменении бд не нужно пересобирать приложение. Недостатки: зависимость от контейнера; при использовании старой версии log4j есть уязвимость (log4shell) основанная на jndi. Варианты использования: CDI аннотации, работает только в managed компонентах; прямой вызов API, работает везде. new InitialContext().lookup(“res”);
2. React — JS библиотека для разработки пользовательского интерфейса (SPA). Позволяет создавать свои собственные компоненты, с пропсами и стейтом. Компоненты рендерятся в HTML. Передача данных от родителя к детям. Виртуальный DOM. При изменении state происходит рендер компонента с обновлением вложенных компонентов/тэгов. Важная особенность React — использование JSX. Это расширение синтаксиса JavaScript, которое удобно использовать для описания интерфейса. Кроме того, React содержит виртуальный DOM — облегченное представление в памяти реального DOM. При написании сложных приложений, работающих с большим количеством данных, часто применяется архитектура Flux и библиотека Redux.

1. JSF Restore View phase
2. Spring Framework. Отличия и сходства с JavaEE
1. Restore View phase происходит до Apply Request Values Phase. JSF Runtime формирует представление (начиная cUIViewRoot): Создаются объекты дерева компонентов, начиная с UIViewRoot, назначаются слушатели, конверторы и валидаторы. Все элементы помещаются в FacesContext. Если клиент уже заходил на эту страницу, то состояния представления синхронизируются с клиентом. JSF начинает фазу восстановления представления, как только щелкает ссылка или кнопка и JSF получает запрос. На этом этапе JSF создает представление, связывает обработчики событий и средства проверки с компонентами пользовательского интерфейса и сохраняет представление в экземпляре FacesContext. Экземпляр FacesContext теперь будет содержать всю информацию, необходимую для обработки запроса. Если это первый запрос пользователя к странице JSF, то формируется пустое представление. Если это запрос к уже существующей странице, то JSF Runtime синхронизирует состояние компонентов представления с клиентом. Проверяется тип запроса, а после запускаются процессы рендера новых страниц(если это GET) и создание/изменение данных(если это POST).
2. Java EE грубо говоря, конструктор, она модульная, можно подключать и отключать совсем маленькие модули. В ней есть множество реализаций представлений, бинов и т.д. что делает ее идеальной для разработки монолитного масштабируемого приложения. Spring — также разделен на модули, но эти модули довольно крупные и скорее удобно дополняют друг-друга чем живут обособленно. Универсальный фреймворк для разработки приложений на Java. Этот фреймворк подходит для небольших веб приложений, либо для микросервисной архитектуры. Тот же ajax, который в JavaEE пишется 1 строчкой(благодаря JSF), в спринге нужно писать руками. Реализует паттерн IoC и механизмы CDI. Активно использует инфраструктурные решения Java / Jakarta EE. Базовая концепция Java EE – разделение обязанностей между контейнером и компонентом; «базовая» концепция Spring – IoC / CDI. Контейнер в Java EE включает в себя приложение; приложение в Spring включает в себя контейнер. Java EE – спецификация; Spring – фреймворк.

1. Java EE CDI Beans прерывание жизненного цикла (Interception)
2. Компоненты React. State and props. Умные и Глупые компоненты

1. В спецификации CDI предусмотрен механизм, который предоставляет возможность добавить к методу бина предобработку и постобработку. Для связи метода с интерсептором необходимо создать кастомную аннотацию, которая дополнительно помечается аннотацией @InterceptorBinding. Далее создать класс, который и будет в роли Интерсептора, повесить на него созданную нами аннотацию + @Interceptor. Создать метод для обработки, с параметром InvocationContext ctx и аннотацией @AroundInvoke. В самом методе, чтобы вызвать метод пишем ctx.proceed(). До этого или после мы можем описать дополнительную логику. Теперь осталось добавить нашу кастомную аннотацию методу, чью логику мы хотим расширить (если повесить аннотацию на класс, то все его методы будут прерываться)
2. Компоненты позволяют разбить интерфейс на независимые части. Их можно складывать вместе и использовать несколько раз. Они принимают произвольные входные данные так называемые «пропсы» и возвращают React-элементы. Props и state — это обычные JavaScript-объекты, содержат инфу влияющую на представление. props передаётся в компонент, в то время как state находится внутри компонента. "Умные"компоненты хранят в себе состояние и меняются в зависимости от него. Они управляют простыми компонентами, делают запросы на сервер и многое другое. "Глупые"компоненты - все их действия просты и однообразны, они всего лишь выводят данные, принимаемые ими от свойств (пропсов).

1. Построение интерфейсов на JSF. Иерархия компонентов JSF.
2. Java EE CDI Beans: принципы инъекции бинов.

1. Интерфейсы веб приложений на JSF описываются в XHTML файлах. Там могут быть как обычные HTML-элементы (div, p, h1, img), так и JSF компоненты. Компоненты — это классы наследники UIComponent, образуют иерархию. Корень - UIViewRoot. У каждого компонента (кроме UIViewRoot) есть родитель, а также могут быть дети. Плюс компонентов в том, что они могут инкапсулировать сложную верстку и логику на JS за одним XHTML тегом. Существуют сторонние библиотеки компонентов, такие как PrimeFaces и IceFaces, которые упрощают построение интерфейса обширным набором готовых компонентов. Можно создавать свои компоненты
2. Для того чтобы внедрить бины можно использовать аннотацию @Inject, тогда контейнер найдет у себя подходящий бин и сам создаст его. Так же если подходит несколько бинов, то будет выброшено исключение, чтобы избежать этого можно использовать аннотацию @Alternative Внедрение зависимостей работает с полями класса и с конструктором. Механизм выбора подходящего бина учитывает запрашиваемый класс или интерфейс, название бина (@Named) и альтернативы (@Alternative). @Qualifier используется когда надо конкретизировать какой именно бин внедрить. В бины внедряется не оригинальный класс бина-зависимости, а класс-прокси, который создается на лету самим контейнером и позволяет реализовывать перехватчики.

1. REST в спринге: методы и аргументы
2. Навигация в React. React Router

1. @RestController = @Controller + @ResponseBody REST можно реализовать в обычном Spring MVC контроллере, используя аннотацию @Controller. На каждый метод-обработчик, возвращающий сериализованный ответ в теле, нужно добавить аннотацию @ResponseBody. Для удобства в Spring сделали аннотацию @RestController, которая по умолчанию применяет @ResponseBody к всем методам, помеченными аннотацией @RequestMapping. Специальными аннотациями @Get/Post/Put/DeleteMapping помечаются методы для обработки http запросов. В аргументах аннотации можно указать path, по которому можно обратиться к данному методу. Методы-обработчики запросов могут принимать параметры из URL через аннотацию @PathVariable, параметры GET-запроса (query string) через @RequestParam, десериализованные данные из тела запроса через @RequestBody. Spring автоматически определит формат данных, исходя из заголовка Content-Type и десериализует их, при наличии библиотеки (для JSON используется Jackson) @ResponseBody - сереализует в JSON для передачи клиенту
2. React Router - система маршрутизации, позволяющая делать навигацию между компонентами, а также позволяет сопоставлять запросы к с компонентами. В React браузер всегда показывает одну и ту же страницу. Содержимое страницы меняется динамически. Router определяет набор маршрутов и выполняет сопоставление запроса с маршрутами. Выбирает маршрут для обработки запроса по URL. Routes содержит набор маршрутов и позволяет выбрать первый попавшийся маршрут по нужному URL и его использовать для обработки. Каждый маршрут представляет объект Route. Для маршрута устанавливаются атрибуты: path - шаблон адреса; element - отвечает за обработку запроса по этому маршруту;

1. REST контроллеры в спринге. Сериализация и десериализация
2. Архитектура Angular приложения. Модули, компоненты, представление, сервисы

1. Для обработки запросов написать контроллер с аннотацией @RestController в котором будет обработка клиентских запросов. Специальными аннотациями Get/Post/Put/Delete Mapping помечаются методы для обработки запросов. В аргументах аннотации можно указать path. Десериализация: если присутствует тело запроса, прописан заголовок Content-Type, и обработчик запроса принимает аргумент, помеченный аннотацией @RequestBody, Spring автоматически десериализует данные, используя Jackson. Из коробки доступен формат JSON, но можно установить поддержку XML. Помимо тела запроса, данные могут приходить как часть URL: их можно вытащить через @PathVariable, или как часть GET-параметра, используя @RequestParam. Сериализация: из обработчика возвращается объект, а Spring его автоматически сериализует. Формат выбирается исходя из HTTP заголовка Accept. Формат можно прописать вручную в свойстве produces аннотации @RequestMapping.
2. Приложение разбивается на модули, которые могут импортировать друг в друга. В модулях определяются компоненты — строительные блоки интерфейса, инкапсулирующие верстку, стили, и логику приложения, можно переиспользовать внутри других компонентов. Компонент состоит из TS класса, помеченного декоратором @Component, HTML шаблона, CSS стилей. Компонент вместе с шаблоном образуют представление, которое образуют иерархию. Существует двухсторонняя связь между классом компонента и представлением — при изменении данных в компоненте обновляется представление и наоборот. Задачи приложения, которые не касаются представления, выносятся в сервисы (загрузка данных с сервера, валидация данных, фоновые процессы, логирование). Angular поддерживает внедрение сервисов в компоненты - @Injectable.

1. Managed bean: назначение, конфигурация, использование в xhtml
2. Архитектура и состав Spring Web MVC

1. Managed beans - обычные JAVA классы управляемые JSF. Хранят состояние JSF-приложения. Содержат параметры и методы для обработки данных, получаемых из компонентов. Занимаются обработкой событий. Настройка происходит в faces-config.xml или при помощи аннотаций. У Managed Beans есть скоуп – время, в которое бин будет создан и будет доступен. Скоупы: NoneScoped - жизненным циклом управляют другие бины; RequestScoped - контекст - запрос; ViewScoped - контекст-страница(компонент создается один раз при обращении к странице); SessionScoped - контекст - сессия; ApplicationScoped - контекст - приложение; CustomScoped - компонент создается и сохраняется в коллекции типа Map. Областью жизни управляет программист. В JSF обращаться к ManagedBean можно через EL:# {myBean.property}
2. Model инкапсулирует данные приложения для формирования представления. View формирует HTML страницу. Фрэймворк не специфицирует жестко технологию на которой будет построено представление. Можно использовать Thmyleaf, Freemaker, реализовывать представление вне спринга на JS. Controller обрабатывает запрос пользователя, связывает модель с представлением, управляет состоянием модели. В шаблоне мы можем читать свойства модели и отображать их на странице. Класс и его методы могут быть помечены аннотациями привязывающими его к HTTP методам или URL. DispatcherServlet - сервлет, который принимает все запросы и передает управление контроллерам, написанными программистом. HandlerMapper — интерфейс для поиска подходящего контроллера. Контроллер — класс с аннотацией @Controller, который занимается обработкой запросов. В нем реализуется некая бизнес логика для подготовки данных. ViewResolver — интерфейс для поиска подходящего представление.

1. Класс FacesServlet - назначение, особенности конфигурации
2. Принципы IoC и CDI. Реализация в Java EE

1. FacesServlet — главный сервлет, который занимается жизненным циклом обработки запросов в приложениях, построенных на JSF. Обрабатывает запросы с браузера. Формирует объекты события и вызывает методы-слушатели. Является частью фреймворка. Конфигурация через файл web.xml. Там настраиваются правила навигации, регистрируются ManagedBeans, конвертеры, валидаторы, компоненты пользовательского интерфейса. Многие из этих настроек также доступны через аннотации. В разделе <servlet> регистрируется сам серверлет и имя, а в разделе <servlet-mapping> правила url навигации.
2. Инверсия управления — принцип используемый для уменьшения связности кода. Заключается в том, что самописным кодом управляет общий фреймворк, занимающийся жизненным циклом компонентов и коммуникацией между ними. Чаще всего реализуется посредством внедрения зависимостей. Внедрение зависимостей - паттерн проектирования (сокращенно DI), который позволяет создавать объект, использующий другие объекты. При этом поля объекта настраиваются внешней сущностью, что позволяет убрать зависимость компонента от контейнера. Не требуется реализации каких-либо интерфейсов. Не нужны прямые вызовы API. Реализуется через аннотации. В жава ее для впрыскивания @Inject, для выдачи имени @Named Конкретные реализации передаются через конструктор, поля, или сеттеры. В Java существует несколько реализаций паттернов IoC и CDI: EJB, CDI, Spring. Все они очень похожи. В качестве компонентов выступают бины — классы, написанные по определенным правилам, жизненных цикл которых управляется контейнером.

1. JavaServer Faces. Особенности, недостатки, преимущества.
2. CDI бины - что это, зачем нужны, если есть EJB и ManagedBeans

1. Особенности: Компонентно-ориентированная структура. Интерфейс строится из компонентов, которые могут быть вложены друг в друга. Рендерятся в HTML элементы. Для отображения данных используются JSP или XML-шаблоны (facelets). Бизнес логика выносится в Java бины. Написан поверх Servlet API. Входит в JAVA EE Преимущества: разделение бизнес логики от представления (реализует MVC); Управление обменом данными на уровне компонент; программисту нужно писать меньше JS кода; простота реализации AJAX; работа с событиями на стороне сервера; расширяемость(доп наборы компонент, можно определять свои); поддержка в IDE Недостатки: Плохо масштабируется. Сложно реализовывать не предусмотренную авторами функциональность и компоненты; Не подходит для высокопроизводительных приложений; learning curve.
2. CDI - бины, которые позволяют разработчику использовать концепцию внедрения зависимостей. В отличие от MB, CDI бины намного мощнее и гибче, они могут использовать перехватчики, стереотипы, декораторы и многое другое, а также смешиваться с другими бинами. EJB же обладают некоторыми особенностями, недоступными для CDI (например, транзакционные функции, таймеры, асинхронность, удаленность). Однако, в целом, EJB и CDI схожи, и их можно даже инжектить друг в друга.

1. Контекст управляемых бинов. Конфигурация контекста бина
2. Шаблоны MVVM и MVP. Сходства и отличия от MVC

1. Контекст определяет, к чему будет привязан бин и его время жизни. Конфигурировать можно через аннотации, либо через faces-config.xml: <managed-bean-scope>application</managed-bean-scope>.
NoneScoped - контекст не определён, жизненным циклом управляют другие бины; по умолчанию RequestScoped - контекст - запрос; ViewScoped - контекст-страница(компонент создается один раз при обращении к странице, и используется ровно столько, сколько пользователь находится на странице); SessionScoped - контекст - сессия; ApplicationScoped - контекст - приложение; CustomScoped - компонент создается и сохраняется в коллекции типа Map. Областью жизни управляет программист.
2. MVVM - Model-View-ViewModel: Смысл в том, что ViewModel не связан напрямую с View, а общается с ним с помощью простых команд, и вью подписывается на его изменение. Сама же ViewModel содержит модель, преобразованную к представлению, а также команды, через которые представление обращается к модели. MVP, или Model-View-Presenter - шаблон, созданный много позже MVC. Вместо Контроллера-Презентер. отвечает за отрисовку и обновление View, а за обновление Model. содержит логику интерфейса и отвечает за синхронизацию Model и View В MVC input идёт на controller, который далее взаимодействует с View и Model. (Причём один контроллер может взаимодействовать со многими View)

1. Конвертеры JSF, создание и назначение
2. Реализация model в Spring web MVC

1. Классы, реализующие интерфейс Converter. Используются для преобразования данных компонента в заданный формат (дата, число и т. д.). Существуют стандартные конвертеры для основных типов данных (например DateTimeConverter). Можно создавать собственные конвертеры. Чтобы зарегистрировать свой конвертер, необходимо прописать его в faces-config.xml либо воспользоваться аннотацией FacesConverter. Назначение в JSF: автоматическое(на основе типа данных) <h:inputText value="#{user.age}"/> ; с помощью атрибута converter: <h:inputText converter="#{javax.faces.DateTime}"/> ; с помощью вложенного тега: <h:outputText value="#{user.birthDay}» <f:converter converterId="#{javax.faces.DateTime}"/> </h:outputText>
2. Model в Spring MVC — интерфейс, для работы с данными необходимыми для формирования представления. Основные методы модели — addAttribute, getAttribute, asMap. В модель можно класть любой объект(POJO), и доставать его по строковому ключу. Через модель представление получает доступ к данным приложения, которые необходимо вывести на странице. Упрощенная реализация - ModelMap Существует аннотация @ModelAttribute, которая позволяет задать в контроллере метод, заполняющий модель атрибутами, которые потом будут доступны из всех обработчиков. Также можно пометить аргумент обработчика как @ModelAttribute. В таком случае, наш объект будет создан используя параметры из запроса, а затем будет положен в модель. В общем случае, реализует интерфейс org.springframework.ui.Model.

1. CDI Beans
2. Angular DI

1. Универсальные компоненты уровня бизнес- логики. Общая идея – «отвязаться» от конкретного фреймворка при создании бизнес-логики внутри приложения. В большинстве случаев их можно использовать вместо JSF Managed Beans и EJB. По реализации очень похожи на JSF Managed Beans. CDI bean — класс, удовлетворяющим следующим требованиям: статический и должен иметь конструктор без аргументов. CDI бины поддерживают внедрение зависимостей. Жизненным циклом бинов управляет CDI контейнер. Реализация происходит через аннотации: @Named("name"); указывается скоуп, например - @SessionScoped; далее можно сделать инжект например в класс едпоинта @Path("/path") public class NameEndpoint {@Inject private @Named("name") NameBean nameBean;

2. Dependency Injection - паттерн проектирования, который позволяет создавать объект, использующий другие объекты. При этом поля объекта настраиваются внешней сущностью. Angular поддерживает DI. В компоненты внедряют сервисы, в которых реализуется бизнес логика, не связанная с представлением. Например логирование, общение с API. (с помощью DI легче тестировать). Можно писать свои классы помеченные аннотацией @Injectable и внедрять их. Зависимости передаются ввиде параметров конструктора. Принципы: приложение содержит глобальный инжектор (root) который занимается DI; Injector создает зависимости и передает их экземпляры контейнеру; Provider - обръект который сообщает Injectory как получить или создать экземпляр зависимости; Обычно провайдер сервиса - сам его класс;

1. JSF: ключевые особенности, преимущества, недостатки
2. CDI-бины: что такое и зачем нужны, когда есть EJB и Managed Beans

1. JSF — фреймворк для создания веб приложений на Java, является частью стандарта Java EE. Он абстрагирует программиста от работы с http протоколом напрямую. Компонентно-ориентированная структура. Интерфейс строится из компонентов, которые могут быть вложены друг в друга. Рендерятся в HTML элементы. Для отображения данных используются JSP или XML-шаблоны (facelets). Бизнес логика выносится в Java бины Написан поверх Servlet API. входит в JAVA EE. Преимущества: разделение бизнес логики от представления (реализует MVC); Управление обменом данными на уровне компонент; программисту нужно писать меньше JS кода; простота реализации AJAX; работа с событиями на стороне сервера; расширяемость(доп наборы компонент, можно определять свои); поддержка в IDE. Недостатки: Плохо масштабируется. сложно реализовывать не предусмотренную авторами функциональность и компоненты; Не подходит для высокопроизводительных приложений.

2. CDI бины - специальные бины, которые позволяют разработчику использовать концепцию внедрения зависимостей. CDI даёт возможность управлять bean-компонентами. В отличие от MB, CDI бины намного мощнее и гибче, они могут использовать перехватчики, стереотипы, декораторы и многое другое, а также смешиваться с другими бинами. EJB же обладают некоторыми особенностями, недоступными для CDI (например, транзакционные функции, таймеры, асинхронность, удаленность). Однако, в целом, EJB и CDI схожи, и их можно даже инжектить друг в друга.

1. Шаблоны и представление в Angular
2. Dependency Lookup Spring

1. Представление компонента задается с помощью шаблонов. Шаблоны похожи на обычный html, взаимодействуют с классом компонента через data binding. Представления группируются иерархически. Компонент может содержать иерархию представлений, которая содержит встроенные представления из других компонентов. Поддерживается интерполяция: {{ value }}. [attr]=”value” — одностороннее связывание, [(attr)]=”value” — двухстороннее связывание, @event=”handler” — обработчик событий. Для условной отрисовки используется директива *ngIf, для циклов — *ngFor. Шаблоны могут содержать фильтры и директивы. Директивы - инструкции по преобразованию DOM. Фильтры - могут преобразовывать данные в нужный формат. Поддерживаются pipe chains, также могут принимать аргументы.

2. Dependency Lookup — подход при котором компонент напрямую просит у контейнера передать ему зависимость. Противопоставляется DI, который происходит автоматически. Не рекомендуется к использованию, но необходимо когда DI дает сбой(например достает не тот бин). В Spring для реализации DL нужно сначала получить контекст приложения (например ClassPathXmlApplicationContext). Затем: context.getBean(“name”) — вернет нужный бин. Также можно указать конкретный класс или интерфейс: context.getBean(“name”, MyBean.class). ApplicationContext appContext = new ClassPathXmlApplicationContext(“/applicationcontext.xml”); MyBean bean = appContext.getBean(“myBean”);

1. Java EE CDI Beans стереотипы
2. Разметка страницы в React - приложениях. JSX

1. Stereotype — аннотация, включающая в себя много аннотаций. В которой указано: Область видимости по умолчанию; Ноль или более Interceptorов; По желанию, аннотация @Named, гарантирующая именование EL по умолчанию; Необязательно, аннотация @Alternative, указывающая, что все компоненты с этим стереотипом являются альтернативами Бин, аннотированный определённым стереотипом, всегда будет использовать указанные аннотации, так что не нужно применять одни и те же аннотации ко многим бинам. Мы можем создать свой стереотип и использовать его: @ApplicaitonScoped @Named @Secure public @interface myStereotype(){}. Также существуют стандартные стереотипы, например @Model (@RequestScoped + @Named) При применении аннотации @MyStereotype будут включатся все перечисленные аннотации. Один бин может использовать несколько стереотипов. Если у вас будут разные scope в стереотипах, то у вас не скомпилится, либо вам надо будет указать скоуп прямо перед бином. В стереотипах нельзя задавать имена бинов.

2. React представляет собой дерево из компонентов. Точкой входа (корнем) являет index.js который определяет компоненты. Каждый компонент включает в себя другие компоненты. В React логика рендеринга и разметка живут вместе в одном месте - в компонентах. В React приложениях разметка пишется в JSX файлах. JSX — надстройка над JS, которая позволяет вкраплять HTML-синтаксис в код. Можно использовать стандартные HTML элементы (такие как div, span, h1, input) так и кастомные React компоненты. JSX код: text компилируется в вызов функции React.createElement(“div”, className: “foo”, “text”)

1. Process validations phase, Update model values phase
2. Жизненный цикл Spring-приложения

1. Process Validation Phase: На этом этапе реализация JavaServer Faces обрабатывает все валидаторы, зарегистрированные в компонентах в дереве, используя свой метод validate (processValidators). Он изучает атрибуты компонента, которые определяют правила проверки, и сравнивает эти правила с локальным значением, хранящимся для компонента. Если значение компонента не проходит валидацию, формируется сообщение об ошибке, которое сохраняется в FacesContext. Update Model Values Phase: Реализация JavaServer Faces устанавливает соответствующие свойства объекта на стороне сервера для локальных значений компонентов. В этот момент, если приложению необходимо перенаправить на другой ресурс веб-приложения или сгенерировать ответ, который не содержит никаких компонентов JavaServer Faces, оно может вызвать метод FacesContext.responseComplete.

2. 1. Конфигурация, написанная на xml или в виде Java кода, попадает в BeanDefinitionReader, который парсит конфигурацию. После считывания он выдает BeanDefinition-ы. BeanDefinition хранит: каким образом спрингу создавать бины, как бин определен, какие зависимости в себе содержит, какие аннотацию и другие настройки. 2. Если Bean настроен как Singleton (у спринга по умолчанию), то он в конвейере будет обработан сразу, будет создан объект и он попадет в IoC контейнер. Если Singleton, но Lazy Bean - не будет создан сразу, только по запросу. Prototype - не попадают в IoC контейнер. Спринг управляет им до момента как мы его заполучили. Как только он создан и отдан, он перестает управлять его жизненным циклом. 3. Создание кастомных FactoryBean 4. BeanFactory создает экземпляры бинов делегируя создание FactoryBean, если мы его определили. 5. Устанавливаются системные переменные, property для того, чтобы в дальнейшем правильно создать бины. Жизненный цикл бина: 1. Техническое начало жизни бина, работа конструктора его класса; 2. Установка свойств из конфигурации бина, внедрение зависимостей; 3. Нотификация aware-интерфейсов. Обновление контекста: вызов постпроцессоров. 4. Пре-инициализация – метод postProcessBeforeInitialization() интерфейса BeanPostProcessor; 5. Инициализация. Метод бина с аннотацией @PostConstruct из стандарта; Метод afterPropertiesSet() бина; Init-метод. 6. Пост-инициализация – метод postProcessAfterInitialization() интерфейса BeanPostProcessor. Уничтожение: 1. Метод с аннотацией @PreDestroy; 2. Метод с именем, которое указано в свойстве destroyMethod определния бина; 3. Метод destroy() интерфейса DisposableBean.

1. Location Transparency в Java EE
2. Spring MVC часть представления

1. Принцип Location Transparency (прозрачность местоположения) означает, что благодаря CDI мы можем добиться того, что нам станет не важно, где физически расположен вызываемый компонент (локально, удалённо, вне/внутри контейнера) - за его вызов отвечает контейнер. Если мы, к примеру, с JNDI ищем DataSource, нам не будет важно, где находится объект, ссылку на который мы получим. Таким образом, мы можем одинаково обращаться как к локальному, так и к удалённому объекту. Его получением занимается сервер приложений. В Java EE в первую очередь реализуется через JNDI — API для предоставления доступа к объектам по имени, нежели по их физической локации.

2. Фреймворк не специфицирует жёстко технологию, на которой должно быть построено представление. Вариант «по-умолчанию» – JSP. Можно использовать Thymeleaf, FreeMarker, Velocity etc. Можно реализовать представление вне контекста Spring – целиком на JS. Представление отвечает за то, как будут визуализироваться данные в браузере пользователя. За поиск представления по имени отвечает интерфейс ViewResolver. ViewResolver - интерфейс, при помощи которого DispatcherServlet определяет какое представление нужно использовать на основании имени.