LaTeX

1. Компоненты в Angular: взаимодействие с представлениями и сервисами
2. Инициализация Spring Beans

1. В модулях определяются компоненты. Компоненты — строительные блоки интерфейса, кирпичики, 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. Angular поддерживает DI: сервисы внедряются в компоненты (аннотация @Injectable).
2. Инициализация. Разные способы применяются в таком порядке: Метод бина с аннотацией @PostConstruct из стандарта JSR-250 (рекомендуемый способ); Метод afterPropertiesSet() бина реализующего интерфейс 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. Могут быть использованы в JSF путём конфигурации в faces-config.xml

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 - контекст - запрос; @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 - контекст - сессия; @ApplicationScoped - контекст - приложение; @ConversationScoped - Областью жизни управляет программист. Управление осуществляется через инъекцию объекта javax.enterprise.context.Conversation; @Dependent. Используется по умолчанию. ЖЦ определяется тем где он был использован;

1. Spring Web MVC: View Resolvers
2. Angular: ключевые особенности, отличия от AngularJS

1. Контроллер после обработки запроса возвращает имя представления на которое нужно направить пользователя. 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 ViewResolver. Существует много реализаций ViewResolver для разных технологий построения представления. В одном приложении можно использовать несколько ViewResolver'ов
2. Angular - написана 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. 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.

1. Фаза Apply request values JSF
2. Реализация IoC и CDI 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.
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

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. XML-файл, в котором вручную прописываются все бины, путь до класса, свойства, конструкторы. Загружается этот файл из classpath; (class based config). Создаем класс с аннотацией Configuration, внутри помечаем методы аннотацией Bean; (annotation based config) добавляя к классам аннотации Component и т.д. 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 веб-служб 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; Field-based DI - контейнер через рефлексию будет в поля класса пропихивать зависимости.

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. Использование: Регистрируем серверный объект RMI Registry; Далее создается заглушка, реализующая тот же интерфейс что и серверный объект, отправляется клиенту и притворяется, что все методы есть; При вызове происходит поиск объекта сервера; Клиенту прилетает ответ; Происходит обмен данными;
2. 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 неизменный state либо копию

1. Spring MVC: handler mapping
2. JSX. Применение в реакте. Пример синтаксиса

1. Когда DispatcherServlet получает запрос, он на основании конфигурации HandlerMapping выбрать на какой контроллер пойдет запрос. Этот mapping - механизм, позволяющий распределять запросы по различным обработчикам. Помимо основного Handler'а, в обработке запроса могут участвовать реализаций интерфейса HandlerInterceptor. 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) является 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"/>}

1. MVC в JSF
2. Главные аннотации в CDI beans java EE

1. Model — бины, в которых содержится бизнес логика. 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. Controller — реализуется самим фреймворком. Это класс FacesServlet, который занимается диспетчеризацией и управлением жизненным циклом.
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; Конфигурация - faces-config.xml (опционально); Дескриптор развертывания - как и для любого веб приложения;
2. Spring Web MVC – фреймворк в составе Spring для разработки веб-приложений. Основан на паттерне MVC. 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.

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. При написании сложных приложений, работающих с большим количеством данных, часто применяется архитектура Flux и библиотека Redux.

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. 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. 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. 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. 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 компонентов. Назначаются слушатели событий, конвертеры и валидаторы. Все элементы представления помещаются в FacesContext. 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.
2. Универсальный фреймворк для разработки приложений на Java. 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. 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. **REST в спринге: методы и аргументы**
2. **Навигация в React. React Router**
    1. 123
    2. 123

1. CDI Beans
2. Angular DI
1. 123
2. 123

1. Класс FacesServlet - назначение, особенности конфигурации
2. Vue.js - ключевые особенности, преимущества и недостатки
1. 123
2. 123

1. Шаблоны и представление в Angular
2. Dependency Lookup Spring
1. 123
2. 123

1. Конвертеры JSF, создание и назначение
2. Реализация model в Spring web MVC
1. 123
2. 123

1. Process validations phase, Update module values phase
2. Жизненный цикл Spring-приложения
1. 123
2. 123