

## 1. Компоненты в Angular: взаимодействие с представлениями и сервисами

### 2. Инициализация Spring Beans

1. В модульях 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. Задача приложения, которые не являются представлениями, 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` — переиспользуемые 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` - Область 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 `DispatcherServlet` 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, используя 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 реализуются 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 передается слушателем событий, JSF обрабатывает события также решается вопрос навигации; Формируются новые значения компонентов. (Вызывается метод `UIViewRoot#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 ee, 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 основных 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 разработку 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 вызывает конструктор с аргументами бинов, которые потом занесутся в класс; `Setter-based DI` - сначала контейнер вызывает конструктор бина без аргументов, после вызывает пометочные аннотации `@Autowired` сеттеры и вписки туда нужные зависимости; `Field-based DI` - контейнер через рефлексию будет в поля класса прописывать 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 создаем заглушка, реализующая тот же интерфейс что и серверный объект, отправляющая 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 изменять нельзя. После изменения данных `Store` посылает события представлению, и оно перерисовывается. `Redux` — небольшая библиотека, реализующая упрощенный паттерн `Flux`. В `Redux` есть `store` — синглтон, хранящий состояние всего приложения. Изменения состояния производятся при помощи чистых функций. Они принимают на вход `state` и действия 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` выбирает 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 серветы 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="кровоВТшников"/>}`

### 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` - Область 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. Инъекция может 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 инжектировать. 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 раятммом `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 фреймворк не специфицирует жестко технологию на которой будет построено представление. По умолчанию `JSP`. `Controller` обрабатывает запрос пользователя, создает соответствующую Модель и передает 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`. Преимущество `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 `jni`. Варианты использования: `CDI` аннотация, 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 ререндер 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 платформ 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 обрабатывались зависимости, уже с обработанными зависимостями вызывалось конструирование объекта, а потом добавились срезы. Аспекты - это "выкрапления", 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. 123

2. 123

1. Построение интерфейсов на JSF. Иерархия компонентов JSF.

2. Java EE CDI Beans: принципы инъекции бинов.

1. 123

2. 123

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. 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 управляемого 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, @ResourceVarible). @Controller N public class HelloController { N @RequestMapping(value = "/hello method" = RequestMethod.GET)N public String printHello(ModelMap model) {N model.addAttribute("message"Hello Spring MVC Framework!");N return "hello";}}

1. REST контроллеры в спринге. Сериализация и десериализация

2. Архитектура Angular приложения. Модули, компоненты, представление, сервисы

1. 123

2. 123

1. Managed bean: назначение, конфигурация, использование в xhtml

2. Архитектура и состав Spring Web MVC

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. 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 сUIViewRoot): 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. 123

2. 123

1. Контекст управляемых бинов. Конфигурация контекста бина

2. Шаблоны MVVM и MVP. Сходства и отличия от MVC

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