Билет 218

1. Java EE CDI Beans: “перехватчики” событий жизненного цикла (Interceptors).
2. Redux Toolkit: что такое, для чего нужен, если есть Redux.
3. Интерфейс на JSF, формирующий список новостей. Каждая новость состоит из заголовка, имени автора, даты, иллюстраций, краткой аннотации и полного текста новости, показываемого по клику. Список новостей должен являться динамическим и браться из Managed Bean.

Ответы:

1. В спецификации CDI предусмотрен механизм «перехватчиков», который предоставляет возможность добавить к методу бина предобработку и постобработку.

Interceptorы - классы, реагирующие на определённые события жизненного цикла бинов

Для связи метода с интерсептором необходимо создать кастомную аннотацию, которая дополнительно помечается аннотацией @InterceptorBinding.  
Как внедрить дополнительную логику до и после вызываемого метода:

* 1. Создаем кастомную аннотацию, над которой надо указать @InterceptorBinding
  2. Далее создать класс, который и будет в роли Интерсептора, повесить на него созданную нами аннотацию + @Interceptor
  3. Создать метод для обработки, с параметром InvocationContext ctx пометить его аннотацией @AroundInvoke. В самом методе, чтобы вызвать метод пишем ctx.proceed();. До этого или после мы можем описать дополнительную логику.
  4. Теперь осталось добавить нашу кастомную аннотацию методу, чью логику мы хотим расширить (если повесить аннотацию на класс, то все его методы будут прерываться)

1. Redux Toolkit (RTK) - это официальная библиотека для упрощения процессов разработки приложений, использующих Redux. Она предоставляет набор инструментов и вспомогательных функций, которые помогают создать Redux хранилище и упрощают работу с состоянием приложения.
   1. Упрощение процесса создания Redux-хранилища: RTK предоставляет встроенные функции, такие как configureStore, что избавляет от необходимости писать множество шаблонного кода.
   2. Упрощение работы с редьюсерами: С помощью функции createSlice можно легко создавать редьюсеры и действия, объединяя их в одном месте.
   3. Улучшенное управление побочными эффектами: RTK включает поддержку для createAsyncThunk, облегчая обработку асинхронных действий.
   4. Интеграция с DevTools: RTK имеет встроенную поддержку для Redux DevTools, упрощая процесс отладки.
   5. Оптимизация производительности: Библиотека включает функции для предотвращения лишних перерисовок компонентов.

Хотя Redux уже предоставляет мощные инструменты для управления состоянием, Redux Toolkit делает разработку быстрее, проще и менее подверженной ошибкам. Он помогает разработчикам сосредоточиться на логике приложения, а не на шаблонном коде, что ускоряет процесс разработки и делает его более приятным.

Билет 219

1. JPA: что такое, какое место в иерархии абстракций для работы с БД занимает.
2. Пакетные менеджеры: какие задачи решают. Пример команд для работы с пакетным менеджером npm.
3. Интерфейс на JSF (xhtml + CDI-бин), реализующий сервис по отчислению студентов. Студенты должны браться из коллекций на стороне сервера, напротив каждой фамилии должна находиться кнопка “Отчислить”, клик по которой должен убирать отчисляемого студента из списка. Список должен обновляться с помощью AJAX.

Ответы:

1. Java Persistence API (JPA) — это спецификация для работы с постоянными данными в Java-приложениях, предоставляющая стандартный способ отображения объектов Java на реляционные базы данных. JPA описывает механизм, который позволяет разработчикам работать с объектами (сущностями) вместо того, чтобы манипулировать SQL-запросами напрямую. Это приводит к более чистому, удобному и объектно-ориентированному коду.

Место JPA в иерархии абстракций для работы с базами данных:

1. Приложение / Слой бизнес-логики:

Здесь находятся ваши бизнес-классы и логика. Этот слой использует JPA (обычно через DAO или репозитории) для взаимодействия с базой данных.

1. DAO / Репозиторий:

Уровень доступа к данным, который использует JPA для извлечения и управления данными. DAO (Data Access Object) абстрагирует механизм доступа к данным от остальных слоев приложения. Используя репозитории, разработчики могут работать с обьектами и выполнять операции, такие как создание, чтение, обновление и удаление (CRUD).

1. JPA (Java Persistence API):

Это API, которое определяет, как управлять данными и их состоянием. JPA сам по себе не является реализацией, а представляет собой интерфейс для работы с объектами и их отношениями с базой данных. Она определяет аннотации и интерфейсы, которые дают возможность конфигурировать и управлять объектами-сущностями.

1. Провайдер JPA:

Практическая реализация JPA, которая выполняет всю логику взаимодействия с базой данных. Наиболее популярными реализациями являются:

* 1. Hibernate
  2. EclipseLink
  3. OpenJPA
  4. Каждая из этих реализаций может иметь свои особенности и расширения, но все они соответствуют спецификации JPA.

1. SQL и СУБД:

Это базовый уровень, где происходит фактическое хранение данных. JPA интерпретирует операции над объектами и преобразует их в соответствующие SQL-запросы, которые затем выполняются в реляционной базе данных.

* Упрощение работы с данными: JPA позволяет работать с объектами, что более естественно для объектно-ориентированного программирования.
* Портируемость: благодаря стандарту JPA разработчики могут легко переключаться между различными реализациями, не меняя свою бизнес-логику.
* Инкапсуляция деталей хранения: JPA абстрагирует детали работы с базой данных, что позволяет концентрироваться на бизнес-логике.
* Поддержка кэширования: многие реализации JPA поддерживают встроенные механизмы кэширования для повышения производительности.

1. Пакетные менеджеры — это инструменты, которые помогают управлять библиотеками и зависимостями в проектах, упрощая процессы установки, обновления, удаления и конфигурации библиотек. Они позволяют разработчикам эффективно управлять сторонними библиотеками и модулями, а также обеспечивают автоматизацию ряда задач, таких как следование зависимостям и управление версиями.

Основные задачи, которые решают пакетные менеджеры:

* 1. Установка и обновление пакетов: Позволяют легко загружать и устанавливать внешние библиотеки и модули, а также обновлять их до последних версий.
  2. Управление зависимостями: Пакетные менеджеры гарантируют, что все необходимые зависимости установлены и совместимы, позволяя избежать конфликтов версий.
  3. Удаление пакетов: Позволяют удалять ранее установленные пакеты и их зависимости, если они больше не нужны.
  4. Автоматизация процессов: Позволяют автоматизировать различные процессы разработки, такие как тестирование, сборка и развертывание, за счет использования скриптов и конфигурационных файлов.
  5. Управление версиями: Позволяют устанавливать пакеты определенной версии или версии, соответствующие определенным критериям.
  6. Совместная работа: Обеспечивают возможность создания файла конфигурации (например, package.json в npm), который фиксирует используемые зависимости, что упрощает совместную работу и развёртывание проекта на других машинах.

Билет 213

1. 3-й и 4-й этап жизненного цикла JSF вопрос-ответа.
2. Ssr, csr, ssg. Архитектура и различия.
3. Представить образ БД студента ВТ через JPA.

Ответы:

1. 3-й этап: Обработка событий (Process Events)

На этом этапе JSF обрабатывает все события, которые могут возникнуть в ходе взаимодействия пользователя с интерфейсом. Это может включать в себя различные действия, такие как нажатия на кнопки, изменение данных в полях ввода и другие взаимодействия.

Основные задачи:

* Обработка событий, инициированных пользователем.
* Вызывает соответствующие методы обработчиков событий, если такие имеются.
* Выполнение логики приложения, связанной с событиями (например, переходы на другие страницы после успешного выполнения действий).

4-й этап: Обновление модели (Update Model)

После обработки событий JSF обновляет модели данных. Это означает, что JSF берет значения из компонентов пользовательского интерфейса и обновляет соответствующие значения в управляемых бинах (например, в CDI-бинах или бинах JSF).

Основные задачи:

* Получение входных данных из компонентов пользовательского интерфейса (таких как текстовые поля, флажки и т. д.) и обновление значений соответствующих свойств в управляемых бинах.
* Валидация данных (перед обновлением модели) для проверки правильности введенных данных, если применяются какие-либо валидаторы.

Билет ?

1. Managed Bean Scopes & Конфигурация.
2. ORM и Angular: автодополнение.

Билет ?

1. Платформы Java. Особенности, сходства, различия.
2. pacaje.json: для чего нужен, что в нем можно писать.
3. На Angular написать код компонента часов, которые раз в секунду обновляются.

Ответы:

1. Основные платформы Java
2. **JSE Java Standard Edition (Java SE)**

* Описание: Это основная платформа для разработки десктопных приложений и приложений для серверов. Она включает в себя базовую библиотеку классов (Java API), инструменты разработки и стандартные библиотеки.
* Особенности:
* Поддержка многопоточности и работы с графическим интерфейсом пользователя через JavaFX и Swing.
* Расширяемая архитектура через использование библиотек и фреймворков.
* Сферы применения: Десктопные приложения, утилиты, серверные приложения.

**2. JEE Java Enterprise Edition (Java EE)**

* Описание: Это расширение Java SE, предназначенное для разработки корпоративных приложений, которые требуют высокой надежности и масштабируемости.
* Особенности:
* Включает множество API для работы с веб-сервисами, базами данных, системой управления транзакциями (JTA), а также поддерживает архитектуры MVC и ORM.
* Поддержка контейнеров сервлетов (например, Apache Tomcat, JBoss).
* Сферы применения: Веб-приложения, распределенные системы, микросервисы.

**3. JME Java Micro Edition (Java ME)**

* Описание: Это платформа, ориентированная на устройства с ограниченными ресурсами, такие как мобильные телефоны и встраиваемые системы.
* Особенности:
* Предоставляет уменьшенный набор API, оптимизированных для работы на слабых устройствах.
* Поддержка различных профилей для разных классов устройств.
* Сферы применения: Мобильные приложения, IoT-устройства, встраиваемые решения.

4. **JavaFX**

* Описание: Это платформа для создания богатых клиентских приложений с графическим интерфейсом.
* Особенности:
* Позволяет легко создавать сложные пользовательские интерфейсы с помощью CSS и FXML.
* Поддержка мультимедиа и графики.
* Сферы применения: Десктопные приложения с высоко интерактивным интерфейсом.

**Сходства между платформами:**

* **Общий язык программирования:** Все платформы используют язык программирования Java, что позволяет разработчикам применять аналогичные подходы и концепции.
* **Байтовый код и JVM:** Все платформы компилируются в байтовый код, который исполняется на JVM, что обеспечивает платформонезависимость.
* **Объектно-ориентированное программирование**: Все платформы поддерживают принципы ООП, такие как наследование, полиморфизм и инкапсуляция.
* **Значительное количество библиотек:** Каждая платформа Java предоставляет широкий выбор библиотек и API, что облегчает разработку.

**Различия между платформами:**

* **Цели и области применения:**
* Java SE ориентирована на общие приложения и настольные программы, Java EE — на корпоративные и многослойные системы, Java ME — на мобильные и встраиваемые устройства.
* **API и библиотеки:**
* Java EE предлагает гораздо более широкий набор библиотек по сравнению с Java SE и Java ME.
* **Среда выполнения:**
* Java ME работает в ограниченных средах с минимальным набором ресурсов, тогда как Java EE и SE требуют более мощных серверов и приложений.
* **Разработка интерфейсов:**
* JavaFX предлагает современные инструменты для работы с графическим интерфейсом, тогда как Swing и AWT (входящие в Java SE) являются более старыми и менее гибкими.

1. package.json — это файл конфигурации для проектов, разработанных на языке JavaScript с использованием Node.js, а также для проектов, использующих такие инструменты, как npm (Node Package Manager) или yarn. Этот файл содержит информацию о проекте и его зависимостях, а также различные скрипты и метаданные, которые необходимы для управления проектом.

Основные поля package.json

* 1. name: Имя вашего проекта. Обычно это уникальное имя, которое используется при публикации пакета в npm.
  2. version: Версия вашего проекта, следуя семантическому версионированию (например, 1.0.0).
  3. description: Краткое описание вашего проекта.
  4. main: Основной файл вашего проекта (например, index.js), который будет исполняемым по умолчанию при импорте вашего пакета.
  5. scripts: Объект, который содержит команды, которые могут быть запущены с помощью npm run <имя\_скрипта>.
  6. dependencies: Список библиотек и пакетов, которые необходимы вашему проекту для работы.
  7. devDependencies: Список библиотек и пакетов, которые необходимы только для разработки (например, тестирование и сборка), и не нужны в производственной среде.
  8. repository: Ссылка на репозиторий вашего проекта. Обычно это URL на GitHub или другой платформе.
  9. keywords: Массив ключевых слов, которые описывают ваш проект. Это может помочь другим людям найти ваш пакет.
  10. author и license: Информация об авторе и лицензии проекта.

Зачем нужен package.json:

* Управление зависимостями: Позволяет точно указать, какие зависимости необходимы для разработки и запуска приложения, а также их версии.
* Скрипты сборки: Позволяет определять скрипты для автоматизации задач, таких как тестирование, сборка, запуск сервера и т.д.
* Идентификация проекта: Содержит метаданные о проекте, которые могут быть использованы для его идентификации и управления.
* Совместимость: Помогает другим разработчикам легко понять, какие пакеты и версии нужны для работы с вашим проектом.