1. Перечислите Spring модули и их назначение.

**Spring Framework** (или Spring Core) Ядро платформы, предоставляет

базовые средства для создания приложений — управление компонентами

(бинами, **beans**), внедрение зависимостей, MVC фреймворк, транзакции,

базовый доступ к БД. В основном это низкоуровневые компоненты и

абстракции. По сути, неявно используется всеми другими компонентами.

**Spring MVC** (часть Spring Framework) Оперирует понятиями

контроллеров, маппингов запросов, различными HTTP абстракциями и т.п. Со

Spring MVC интегрированы шаблонные движки, типа Thymeleaf, Freemaker,

Mustache, плюс есть сторонние интеграции. JSP или JSF писать не нужно.

**Spring Data** Доступ к данным: реляционные и нереляционные БД, KV

хранилища и т.п.

**Spring Cloud** Много полезного для микросервисной архитектуры —

service discovery, трасировка и диагностика, балансировщики запросов, circuit

breaker-ы, роутеры и т.п.

**Spring Security** Авторизация и аутентификация, доступ к данным,

методам и т.п. OAuth, LDAP, и куча разных провайдеров.

**Spring Integration** Обработка данных из разных источников.

Особняком стоит отметить **Spring Boot**. Boot позволяет быстро создать

и сконфигурить (т.е. настроить зависимости между компонентами)

приложение, упаковать его в исполняемый самодостаточный артефакт. Это то связующее звено, которое объединяет вместе набор компонентов в готовое приложение. Не использует XML для конфигурации. Все конфигурится через аннотации.

1. Расскажите о составе Spring Framework.

**Основной контейнер (Core Container)** включает в себя Beans, Core,

Context и SpEL (expression language).

***Beans*** отвечает за BeanFactory которая является сложной реализацией

паттерна Фабрика (GoF) Это– базовые примитивы Spring namespace,

включая декларирование бинов и как они должны быть связаны.

***Модуль Core*** обеспечивает ключевые части фреймворка, включая

свойства IoC и DI.

***Context*** построен на основе Beans и Core и позволяет получить доступ

к любому объекту, который определён в настройках. Ключевым элементом

модуля *Context* является интерфейс ApplicationContext.

**Модуль *SpEL*** обеспечивает язык выражений для манипулирования

объектами во время исполнения.

***AOP*** реализует аспекто-ориентированное программирование .

**Модуль *Aspects*** обеспечивает интеграцию с AspectJ.

***Instrumentation*** отвечает за поддержку class instrumentation и class

loader, которые используются в серверных приложениях.

Модуль ***Messaging*** обеспечивает поддержку STOMP.

Модуль ***Test*** обеспечивает тестирование с использованием TestNG

или JUnit Framework.

Контейнер **Data Access/Integration** состоит из JDBC, ORM, OXM,

JMS и модуля Transatcions.

***JDBC обеспечивает*** абстрактный слой JDBC.

***ORM* обеспечивает** интеграцию с ORM, например Hibernate, JDO,

JPA и т.д.

**Модуль *OXM*** отвечает за связь Объект/XML – XMLBeans, JAXB и

т.д.

**Модуль *JMS*** *(Java Messaging Service)* отвечает за создание, передачу

и получение сообщений.

***Transactions*** поддерживает управление транзакциями для классов,

которые реализуют определённые методы.

**Web.** Этот слой состоит из Web, Web-MVC, Web-Socket, Web-Portlet

Модуль ***We****b* обеспечивает такие функции, как загрузка файлов и т.д.

***Web-MVC*** содержит реализацию Spring MVC для веб-приложений.

**Spring MVC** cтоит упомянуть отдельно, т.к. мы будем вести речь в

основном о веб-приложениях. Он оперирует понятиями контроллеров,

маппингов запросов, различными HTTP абстракциями и т.п. Со Spring

MVC интегрированы нормальные шаблонные движки, типа Thymeleaf,

Freemaker, Mustache, плюс есть сторонние интеграции с кучей других. Так

что никаких JSP или JSF писать не нужно.

***Web-Socket*** обеспечивает поддержку связи между клиентом и

сервером, используя Web-Socket-ы в веб-приложениях.

***Web-Portlet*** обеспечивает реализацию MVC с среде портлетов.

1. Объясните принцип IoC (Inversion of Control)?

►Зависимый объект или целевой объект-компонент, который требует определенных зависимостей

►Инверсия управления -конфигурирование компонентов приложений и управление жизненным циклом Java-объектов.

4. Объясните принцип DI (Dependency injection)? В чем между ними

разница?

Внедрения зависимостей (DI) -является специализированной формой IоС

Формы:

5. Какие существуют формы внедрения DI?

►ConstructorDependencyJnjection(Внедрение зависимостей через конструктор)

►SetterDependencylnjection(Внедрение зависимостей через метод установки).

6. Что такое Spring Boot и для чего он используется?

Особняком стоит отметить **Spring Boot**. Boot позволяет быстро создать

и сконфигурить (т.е. настроить зависимости между компонентами)

приложение, упаковать его в исполняемый самодостаточный артефакт. Это то связующее звено, которое объединяет вместе набор компонентов в готовое приложение. Не использует XML для конфигурации. Все конфигурится через аннотации.

Чтобы ускорить процесс управления зависимостями, Spring Boot неявно упаковывает необходимые сторонние зависимости для каждого типа приложения на основе Spring и предоставляет их разработчику посредством так называемых starter-пакетов (spring-boot-starter-web, spring-boot-starter-data jpa и т.д.)

Starter-пакеты представляют собой набор удобных дескрипторов

зависимостей, которые можно включить в свое приложение. Это позволит получить универсальное решение для всех, связанных со Spring технологий, избавляя нас от лишнего поиска примеров кода и загрузки из них требуемых дескрипторов зависимостей.

Например, если вы хотите начать использовать Spring Data JPA для

доступа к базе данных, просто включите в свой проект зависимость springboot-

starter-data-jpa и все будет готово (вам не придется искать совместимые

драйверы баз данных и библиотеки Hibernate)

Если вы хотите создать Spring web-приложение, просто добавьте

зависимость spring-boot-starter-web, которая подтянет в проект все

библиотеки, необходимые для разработки Spring MVC-приложений, таких

как spring-webmvc, jackson-json, validation-api и Tomcat

Другими словами, Spring Boot собирает все общие зависимости и

определяет их в одном месте.

Следовательно, при использовании Spring Boot, файл pom.xml содержит

намного меньше строк, чем при использовании его в Spring-приложениях

Второй превосходной возможностью **Spring Boot** является

автоматическая конфигурация приложения. После выбора

подходящего **starter**-пакета, **Spring Boot** попытается автоматически

настроить Spring-приложение на основе добавленных вами **jar**-зависимостей

Например, если вы добавите **Spring-boot-starter-web**, Spring Boot

автоматически сконфигурирует такие зарегистрированные бины,

как **DispatcherServlet**, **ResourceHandlers**, **MessageSource**

Если вы используете **spring-boot-starter-jdbc**, **Spring**

**Boot** автоматически регистрирует бины **DataSource**, **EntityManagerFactory**,

**TransactionManager** и считывает информацию для подключения к базе

данных из файла **application.properties**

7. Как создать и зарегистрировать контроллер?

Класс, помеченный как @Controller автоматически регистрируется в

MVC роутере, а используя аннотации @(Get|Post|Put|Patch)Mapping можно

регистрировать разные пути.

8. Поясните значения аннотаций: @Configuration, @Bean, @Component,

@Service, @Repository, @Controller.

@Configuration сообщает Spring что данный класс является

конфигурационным, содержит определения и зависимости bean-компонентов.

Бины (bean) — это объекты, которые управляются Spring'ом. Для определения бина используется аннотация @Bean.

@Component–показывает, что класс является Spring компонентом (обобщенный – все остальное специализация)

@Controller-показывает,чтокласс определяет Spring MVC контроллер.

@Repository–показывает,что класс определяет хранилище данных.

@Service–показывает,что класс предоставляет сервис.

9. Что такое ModelAndView?

ModelAndView (модель – сами данные; view (представление) – как эти данные отображать)

10. Поясните как работает DispatcherServlet, HandlerMapping,

ViewResolver?

Диспетчер сервлетов DisptacherSevlet Spring’а с помощью Handler Mapping определяет какой контроллер он должен использовать для определенного запроса request

ViewResolver — интерфейс, реализуемый объектами, которые способны находить представления View по имени View Name.

Состояние представления не изменяется во время выполнения приложения и поэтому в реализации возможно применять кэширование.

Реализации могут поддерживать интернационализацию, т.е. локализировать представления.

11. Расскажите как использовать SLF4J (Simple Logging Facade for Java).

@Slf4j аннотация классу и потом использовать log и его методы

12. Поясните как можно использовать @RequestMapping, @GetMapping.

@GetMapping и др.

@RequestMapping адресация для всех типов запроса

@GetMapping адресация для Get запросов

13. Перечислите и охарактеризуйте все аннотации валидации.

Для задания даты позднее текущей

@NotNull @Future

Диапазон

@Size(min = 2, max = 30)

Есть еще много полезных аннотаций валидации. Например для даты:

@DateTimeFormat(style=**"S-"**)

Для валюты:

@NumberFormat(style= NumberFormat.Style.***CURRENCY***)

@Pattern(regexp=**"^(0?[1-9]|[12][0-9]|3[01])[\\/\\-](0?[1-**

**9]|1[012])[\\/\\-]\\d{4}$"**,message=**"Must be formatted DD/MM/YYYY"**)

@NotNull –не нулевое

@NotBlank – не пустое

14. Как создать пользовательский валидатор.

@Documented

@Constraint(validatedBy = PhoneValidator.**class**)

@Target ({ElementType.***FIELD***})

@Retention(RetentionPolicy.***RUNTIME***)

**public** @**interface** CellPhone {

String message() **default "{Phone}"**;

Class<?>[] groups() **default** {};

Class<? **extends** Payload>[] payload() **default** {};

}

**public class** PhoneValidator **implements** ConstraintValidator<CellPhone, String> {

@Override

**public void** initialize(CellPhone paramA) {

}

@Override

**public boolean** isValid(String phoneNo, ConstraintValidatorContext ctx) {

**if**(phoneNo == **null**){

**return false**;

}

*//задание номера телефона в формате "123456789"*

**if** (phoneNo.matches(**"\\d{9}"**))

**return true**;

*//номер телефона может разделяться -, . или пробелом*

**else if**(phoneNo.matches(**"\\d{2}[-\\.\\s]\\d{3}[-\\.\\s]\\d{2}[-**

**\\.\\s]\\d{2}"**))

**return true**;

*//может быть код оператора в скобках ()*

**else if**(phoneNo.matches(**"\\(\\d{2}\\)\\d{3}\\d{2}\\d{2}"**))

**return true**;

*//может быть код страны в скобках ()*

**else if**(phoneNo.matches(**"\\(\\d{3}\\)\\d{2}\\d{3}\\d{4}"**))

**return true**;

*//return false если ничего не подходит*

**else return false**;

}

}

15. Что такое Spring Bean? Как получить экземпляр бина?

**Spring Bean**

Любой объект в фреймворке Spring, который инициализируется через Spring IoC контейнер называется Spring Bean. Любой Java POJO класс может быть Spring бином если его сконфигурировать.

Метод getbean();

16. Какие существуют способы настройки класса в качестве spring bean?

Аннотация и xml

17. Как прописать конфигурацию, чтобы выполнялся автоматический

поиск бинов и управление контейнером?

Автоматическое обнаружение бинов в контекст приложения добавить элемент <context:component-scan>

18. В каком месте и для чего может использоваться аннотация

@Autowired и @Qualifier?

Метод, класс, поле.

@Autowired – аннотация автопривязки по типу из beans

@Qualifier – аннотация уточения автопривязки по имени из beans

19. В каком месте и для чего может использоваться аннотация

@Configuration, @ComponentScan?

@ComponentScan сообщает Spring где искать компоненты, которыми он должен управлять, т.е. классы, помеченные аннотацией @Component или ее производными, такими как @Controller, @Repository, @Service. Эти аннотации автоматически определяют бин класса.

20. Какие scope может иметь Spring Bean?

singleton-тип бина,при котором,создается одна сущность на Spring-контейнер(значение по умолчанию)

prototype–тип бина, при котором каждый раз создается новая сущность бина.

request-тип бина, при котором сущность бина создается одна на request.Такой тип бина справедлив для контекста веб-приложения.

session–тип бина,при котором сущность бина создается только одна на объект httpsession.Такой тип бина справедлив для контекста веб-приложения.

application–тип бина, задаваемый в единственном экземпляре на контекст сервлета. Действителен для контекста веб-приложения.

21. Для чего используют BeanFactory и ApplicationContext?

BeanFactory обеспечивает структуру конфигурации и основные функциональные возможности

ApplicationContext

Подинтерфейс BeanFactory и Spring IoC контейнер ответственный за создание, настройку и сборку bean

►Обеспечивает

простую интеграцию с функциями AOP Spring;

обработку ресурсов сообщений ,

публикацию событий;

+ специфичные для прикладного уровня контексты