1. Что такое AOP? Для чего оно используется?]

Аспекто-ориентированном программировании (далее – АОП). АОП

является одним из ключевых компонентов Spring. Смысл АОП заключается в том, что бизнес-логика приложения разбивается не на объекты, а на “отношения” (concerns).

Термином АОП нередко обозначают инструментальные средства для

реализации сквозной функциональности. Понятие сквозной

функциональности имеет отношение к логике, которая не может быть

отделена от остальной части приложения, что в конечном итоге приводит к дублированию кода и тесной связанности.

Ключевой единицей в ООП является объект, а ключевой единицей в

АОП – аспект. В качестве примера аспекта можно привести *безопасность,кэширование, логирование и т.д.* Внедрений зависимостей (DI) позволяет нам отделять объекты приложения друг от друга. АОП, в свою очередь, позволяет нам отделять сквозные проблемы (cross-cuttings) от объектов, к которым они относятся.

1. Определите понятия aspect, advice, poincut, join point, weaving в AOP?

**Точка соединения (Join point)-** точка в приложении, где мы можем

подключить аспект (как точка наблюдения). Другими словами, это место, где начинаются определённые действия модуля АОП в Spring.

**Совет (Advice) -** фактическое действие, которое должно быть

предпринято до и/или после выполнения метода или набор инструкций выполняемых на точках среза (Pointcut). Это конкретный код, который вызывается во время выполнения программы.

**Срез точек (Pointcut) -** несколько объединённых точек (join points)

или срез, в котором должен быть выполнен совет.

**Аспект** (aspect) - это сочетание совета и срезов, инкапсулированных в классе.

**Связывание или вплетение** (Weaving) - процесс связывания

аспектов с другими объектами приложения для создания совета. Может быть вызван во время компиляции, загрузки или выполнения приложения.

**Аспект (Aspect) -** модуль или класс, который имеет набор программных интерфейсов, которые обеспечивают сквозные требования (модуль в котором собраны описания Pointcut и Advice).

3. Какие есть подходы для реализации AOP?

В Spring поддерживаются 2 подхода для реализации АОП:

**Основанный на XML** (применяется конфигурация с помощью

конфигурационного XML-файла) и о**снованный на аннотациях @AspectJ**(применяется конфигурация с помощью аннотации)

4. Расскажите про архитектуру AOP?

Ключевая архитектура АОП в Spring основана на прокси. Когда вы хотите создать экземпляр класса, снабженный советом, то должны использовать класс ProxyFactory для создания прокси этого экземпляра, первым делом предоставив ProxyFactory со всеми аспектами, которые необходимо связать с прокси. Применение ProxyFactory - это чисто программный подход к созданию прокси АОП. По большей части использовать такой подход в своем приложении не обязательно; вместо этого можно положиться на механизмы

декларативной конфигурации. Это класс ProxyFactory Bean, пространство имен аор и аннотации в стиле @AspectJ, которые обеспечат декларативное создание заместителей.

Во время выполнения платформа Spring анализирует сквозную

функциональность, определенную для бинов в ApplicationContext, и

динамически генерирует прокси-бины (которые являются оболочками для лежащих в основе целевых бинов). Вместо обращения к целевому бину напрямую вызывающие объекты внедряют прокси-бин. Прокси-бин затем анализирует текущие условия (т.е. точку соединения, срез или совет) и соответствующим образом связывает подходящий совет.

В самом каркасе Spring поддерживаются две реализации заместителей: динамические заместители JDK и заместители CGLIВ. По умолчанию, когда целевой объект, снабженный советом, реализует какой-нибудь интерфейс, для получения экземпляров заместителя целевого объекта в Spring будет использован динамический заместитель JDK. Но если целевой объект,снабженный советом, не реализует интерфейс (например, потому, что он представляет конкретный класс), то для получения экземпляров заместителей

будет применяться библиотека CGLIВ. И объясняется это в основном тем, что динамический заместитель JDK поддерживает создание заместителей только для интерфейсов.

5. Что такое AspectJ и как он связан со Spring AOP?

*AspectJ* - это простая в использовании и изучении Java-совместимая среда для интеграции сквозных реализаций. AspectJ был разработан в PARC. Сегодня это одна из известных сред AOP, благодаря своей простоте и возможности поддерживать модульность компонентов. Его можно использовать для применения AOP к статическим или нестатическим полям, конструкторам, методам, которые являются частными, общедоступными или защищенными.

6. Какие AOP Фреймворки вы знаете?

*AspectWertz, AspectJ, JBoss AOP, Dynaop, CAESAR, Spring AOPs*

7. Как определяется конфигурационный файл spring при использовании аспектов?

для разработки аспекта на основе XML:

1. Выберите сквозную задачу, которая будет реализована

2. Напишите аспект.

3. Зарегистрируйте аспект как компонент в контексте Spring.

4. Напишите конфигурацию аспекта

8. Поясните назначение @Aspect, @Before, @After, @Around,

@Pointcut.

before - запускает совет перед выполнением метода.

after - запускает совет после выполнения метода, независимо от

результата его работы (кроме случая остановки работы JVM).

around - запускает совет до и после выполнения метода.

@Aspect – указывает, что класс аспект.

@Pointcut – указывает точку среза.

9. Как создать аспект в программе?

10. Как сконфигурировать advice (совет) который запускается до, после

выполнения метода?

(Before и After) или around

11. Как сконфигурировать advice (совет) который запускается до и после выполнения метода?

12. Как сконфигурировать advice (совет) который запускается после

успешного выполнения метода, после исключения?

After-throwing, after-returning

13. Как определить точки сопряжения при AOP?

@Pointcut;

11.12.2019 – может не быть лекции.