INTERVIEW JAVA DEVELOPER (ADVANCED) SPRING FRAMEWORK

Ответы на вопросы на собеседование Spring Framework (часть 1).

Vasyl K 18:02:00 Добавить комментарий

- **ОБЪЯСНИТЕ СУТЬ ПАТТЕРНА DI ИЛИ IOC.**

Dependency injection (DI) – паттерн проектирования и архитектурная модель, так же известная как Inversion of Control (IoC). DI описывает ситуацию, когда один объект реализует свой функционал через другой объект. Например, соединение с базой данных передается конструктору объекта через аргумент, вместо того чтобы конструктор сам устанавливал соединение. Существуют три формы внедрения (но не типа) зависимостей: сэттер, конструктор и внедрение путем интерфейса.
DI – это способ достижения слабой связанности. IoC предоставляет возможность объекту получать ссылки на свои зависимости. Обычно это реализуется через lookup-метод. Преимущество IoC в том, что эта модель позволяет отделить объекты от реализации механизмов, которые он использует. В результате мы получаем большую гибкость как при разработке приложений, так и при их тестировании.

- **КАКИЕ ПРЕИМУЩЕСТВА ПРИМЕНЕНИЯ DEPENDENCY INJECTION (DI)?**

К преимуществам DI можно отнести:

- Сокращение объема связующего кода. Одним из самых больших плюсов DI является возможность значительного сокращения объема кода, который должен быть написан для связывания вместе различных компонентов приложения. Зачастую этот код очень прост – при создании зависимости должен создаваться новый экземпляр соответствующего объекта.
- Упрощенная конфигурация приложения. За счет применения DI процесс конфигурирования приложения значительно упрощается. Для конфигурирования классов, которые могут быть внедрены в другие классы, можно использовать аннотации или XML-файлы.
- Возможность управления общими зависимостями в единственном репозитории. При традиционном подходе к управлению зависимостями в общих службах, к которым относятся, например, подключение к источнику данных, транзакция, удаленные

службы и т.п., вы создаете экземпляры (или получаете их из определенных фабричных классов) зависимостей там, где они нужны – внутри зависимого класса. Это приводит к распространению зависимостей по множеству классов в приложении, что может затруднить их изменение. В случае использования DI вся информация об общих зависимостях содержится в единственном репозитории (в Spring есть возможность хранить эту информацию в XML-файлах или Java классах).

- Улучшенная возможность тестирования. Когда классы проектируются для DI, становится возможной простая замена зависимостей. Это особенно полезно при тестировании приложения.
- Стимулирование качественных проектных решений для приложений. Вообще говоря, проектирование для DI означает проектирование с использованием интерфейсов. Используя Spring, вы получаете в свое распоряжение целый ряд средств DI и можете сосредоточиться на построении логики приложения, а не на поддерживающей DI платформе.

• КАКИЕ IOC КОНТЕЙНЕРЫ ВЫ ЗНАЕТЕ?

Spring является IoC контейнером. Помимо него существуют HiveMind, Avalon, PicoContainer и т.д.

• КАК РЕАЛИЗУЕТСЯ DI В SPRING FRAMEWORK?

Реализация DI в Spring основана на двух ключевых концепциях Java – компонентах JavaBean и интерфейсах. При использовании Spring в качестве поставщика DI вы получаете гибкость определения конфигурации зависимостей внутри своих приложений разнообразными путями (т.е. внешне в XML-файлах, с помощью конфигурационных Java классов Spring или посредством аннотаций Java в коде). Компоненты JavaBean (также называемые POJO (Plain Old Java Object – простой старый объект Java)) предоставляют стандартный механизм для создания ресурсов Java, которые являются конфигурируемыми множеством способов. За счет применения DI объем кода, который необходим при проектировании приложения на основе интерфейсов, снижается почти до нуля. Кроме того, с помощью интерфейсов можно получить максимальную отдачу от DI, потому что бины могут использовать любую реализацию интерфейса для удовлетворения их зависимости.

• КАКИЕ СУЩЕСТВУЮТ ВИДЫ DI? ПРИВЕДИТЕ ПРИМЕРЫ.

Существует два типа DI: через сэттер и через конструктор.
Через сэттер: обычно во всех java beans используются гэттеры и сэттеры для их свойств:

```java
public class NameBean {
    String name;
    public void setName(String a) {
        name = a;
    }
    public String getName() {
        return name;
    }
}
```

Мы создаем экземпляр бина NameBean (например, bean1) и устанавливаем нужное свойство, например:

```java
bean1.setName("Marfa");
```

Используя Spring реализация была бы такой:

```xml
<bean id="bean1" class="NameBean">
    <property name="name">
        <value>Marfa</value>
    </property>
</bean>
```

Это и называет DI через сэттер. Пример внедрения зависимости между объектами:

```xml
<bean id="bean1" class="bean1impl">
    <property name="game">
        <ref bean="bean2" />
    </property>
</bean>
<bean id="bean2" class="bean2impl" />
```

Через конструктор: используется конструктор с параметрами. Например:

```java
public class NameBean {
    String name;
    public NameBean(String name) {
        this.name = name;
    }
}
```

Теперь мы внедряем объект на этапе создания экземпляра класса, т.е.

```java
bean1 = new NameBean("Marfa");
```

Используя Spring это выглядело бы так:

```xml
<bean id="bean1" class="NameBean">
    <constructor-arg>
        <value>Marfa</value>
    </constructor-arg>
</bean>
```

- **ЧТО ТАКОЕ SPRING? ИЗ КАКИХ ЧАСТЕЙ СОСТОИТ SPRING FRAMEWORK?**

Spring – фреймворк с открытым исходным кодом, предназначеный для упрощения разработки enterprise-приложений. Одним из главным преимуществом Spring является его слоистая архитектура, позволяющая вам самим определять какие компоненты будут использованы в вашем приложении. Модули Spring построены на базе основного контейнера, который определяет создание, конфигурация и менеджмент бинов.

Основные модули:

- Основной контейнер – предоставляет основной функционал Spring. Главным компонентом контейнера является BeanFactory – реализация паттерна Фабрика. BeanFactory позволяет разделить конфигурацию приложения и информацию о зависимостях от кода.
- Spring context – конфигурационный файл, который предоставляет информация об окружающей среде для Spring. Сюда входят такие enterprise-сервисы, как JNDI, EJB, интернационализация, валиадция и т.п.
- Spring AOP – отвечает за интеграцию аспектно-ориентированного программирования во фреймворк. Spring AOP обеспечивает сервис управления транзакциями для Spring-приложения.
- Spring DAO – абстрактный уровень Spring JDBC DAO предоставляет иерархию исключений и множество сообщений об ошибках для разных БД. Эта иерархия упрощает обработку исключений и значительно уменьшает количество кода, которое вам нужно было бы написать для таких операций, как, например, открытие и закрытие соединения.
- Spring ORM – отвечает за интеграцию Spring и таких популярных ORM-фреймворков, как Hibernate, iBatis и JDO.
- Spring Web module – классы, которые помогают упростить разработку Web (авторизация, доступ к бинам Spring-а из web).
- Spring MVC framework – реализация паттерна MVC для построения Web-приложений.

• НАЗОВИТЕ НЕКОТОРЫЕ ИЗ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ, ИСПОЛЬЗУЕМЫХ В SPRING FRAMEWORK?

Spring Framework использует множество шаблонов проектирования, например:

- Singleton Pattern: Creating beans with default scope.
- Factory Pattern: Bean Factory classes
- Prototype Pattern: Bean scopes
- Adapter Pattern: Spring Web and Spring MVC
- Proxy Pattern: Spring Aspect Oriented Programming support
- Template Method Pattern: JdbcTemplate, HibernateTemplate etc
- Front Controller: Spring MVC DispatcherServlet
- Data Access Object: Spring DAO support
- Dependency Injection and Aspect Oriented Programming

- **КАКОВЫ НЕКОТОРЫЕ ИЗ ВАЖНЫХ ОСОБЕННОСТЕЙ И ПРЕИМУЩЕСТВ SPRING FRAMEWORK?**

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring Framework не всецело связан с платформой Java Enterprise, несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

- Относительная легкость в изучении и применении фреймворка в разработке и поддержке приложения.
- Внедрение зависимостей (DI) и инверсия управления (IoC) позволяют писать независимые друг от друга компоненты, что дает преимущества в командной разработке, переносимости модулей и т.д..
- Spring IoC контейнер управляет жизненным циклом Spring Bean и настраивается наподобие JNDI lookup (поиска).
- Проект Spring содержит в себе множество подпроектов, которые затрагивают важные части создания софта, такие как вебсервисы, веб программирование, работа с базами данных, загрузка файлов, обработка ошибок и многое другое. Всё это настраивается в едином формате и упрощает поддержку приложения.

- **КАКОВЫ ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ SPRING TOOL SUITE?**

Для упрощения процесса разработки основанных на Spring приложений в Eclipse (наиболее часто используемая IDE-среда для разработки Java-приложений), в рамках Spring создан проект Spring IDE. Проект бесплатный. Он интегрирован в Eclipse IDE, Spring IDE, Mylyn (среда разработки в Eclipse, основанная на задачах), Maven for Eclipse, AspectJ Development Tool.

- **ЧТО ТАКОЕ AOP? КАК ЭТО ОТНОСИТЬСЯ К IOC?**

Аспектно-ориентированное программирование (АОП) - парадигма программирования, основанная на идее разделения функциональности для улучшения разбиения программы на модули. AOP и Spring - взаимодополняющие технологии, которые позволяют решать сложные проблемы путем разделения функционала на отдельные модули. АОП предоставляет возможность реализации сквозной логики - т.е. логики, которая применяется к множеству частей приложения - в одном месте и обеспечения автоматического применения этой логики по всему приложению. Подход Spring к АОП заключается в создании "динамических прокси" для целевых объектов и "привязывании" объектов к конфигурированному совету для выполнения сквозной логики.

- **ЧТО ТАКОЕ ASPECT, ADVICE, POINTCUT, JOINTPOINT И ADVICE ARGUMENTS В АОП?**

Основные понятия АОП:

- Аспект (англ. aspect) – модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.
- Совет (англ. advice) – фрагмент кода, который должен выполняться в отдельной точке соединения. Существует несколько типов советов, совет может быть выполнен до, после или вместо точки соединения.
- Точка соединения (англ. joinpoint) – это четко определенная точка в выполняемой программе, где следует применить совет. Типовые примеры точек соединения включают обращение к методу, собственно Method Invocation, инициализацию класса и создание экземпляра объекта. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.
- Срез (англ. pointcut) – набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в AspectJ применяются Java-сигнатуры) и позволяют их повторное использование с помощью переименования и комбинирования.
- Связывание(англ. weaving) представляет собой процесс действительной вставки аспектов в определенную точку кода приложения. Для решений АОП времени компиляции это делается на этапе компиляции, обычно в виде дополнительного шага процесса сборки. Аналогично, для решений АОП времени выполнения связывание происходит динамически во время выполнения. В AspectJ поддерживается еще один механизм связывания под названием связывание во время загрузки (load-time weaving – LTW), который перехватывает лежащий в основе загрузчик классов JVM и обеспечивает связывание с байт-кодом, когда он загружается загрузчиком классов.
- Цель(англ. target) – это объект, поток выполнения которого изменяется каким-то процессом АОП. На целевой объект часто ссылаются как на объект, снабженный советом.
- Внедрение (англ. introduction, введение) – представляет собой процесс, посредством которого можно изменить структуру объекта за счет введения в него дополнительных методов или полей, изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (англ. metaobject protocol, MOP).

- **В ЧЕМ РАЗНИЦА МЕЖДУ SPRING AOP И ASPECTJ АОП?**

AspectJ де-факто является стандартом реализации АОП. Реализация АОП от Spring имеет некоторые отличия:

- Spring AOP немного проще, т.к. нет необходимости следить за процессом связывания.
- Spring AOP поддерживает аннотации AspectJ, таким образом мы можем работать в спринг проекте похожим образом с AspectJ проектом.

- Spring AOP поддерживает только proxy-based АОП и может использовать только один тип точек соединения – Method Invocation. AspectJ поддерживает все виды точек соединения.
- Недостатком Spring AOP является работа только со своими бинами, которые существуют в Spring Context.

- **ЧТО ТАКОЕ IOC КОНТЕЙНЕР SPRING?**

По своей сути IoC, а, следовательно, и DI, направлены на то, чтобы предложить простой механизм для предоставления зависимостей компонента (часто называемых коллабораторами объекта) и управления этими зависимостями на протяжении всего их жизненного цикла. Компонент, который требует определенных зависимостей, зачастую называют зависимым объектом или, в случае IoC, целевым объектом. IoC предоставляет службы, через которые компоненты могут получать доступ к своим зависимостям, и службы для взаимодействия с зависимостями в течение их времени жизни. В общем случае IoC может быть расщеплена на два подтипа: инверсия управления (Dependency Injection) и инверсия поиска (Dependency Lookup). Инверсия управления – это крупная часть того, делает Spring, и ядро реализации Spring основано на инверсии управления, хотя также предоставляются и средства Dependency Lookup. Когда платформа Spring предоставляет коллабораторы зависимому объекту автоматически, она делает это с использованием инверсии управления (Dependency Injection). В приложении, основанном на Spring, всегда предпочтительнее применять Dependency Injection для передачи коллабораторов зависимым объектам вместо того, чтобы заставлять зависимые объекты получать коллабораторы через поиск.

- **ЧТО ТАКОЕ SPRING БИН?**

Термин бин (англ. Bean) – в Spring используется для ссылки на любой компонент, управляемый контейнером. Обычно бины на определенном уровне придерживаются спецификации JavaBean, но это не обязательно особенно если для связывания бинов друг с другом планируется применять Constructor Injection. Для получения экземпляра бина используется ApplicationContext. IoC контейнер управляет жизненным циклом спринг бина, областью видимости и внедрением.

- **КАКОЕ ЗНАЧЕНИЕ ИМЕЕТ КОНФИГУРАЦИОННЫЙ ФАЙЛ SPRING BEAN?**

Конфигурационный файл спринг определяет все бины, которые будут инициализированы в Spring Context. При создании экземпляра Spring ApplicationContext будет прочитан конфигурационный xml файл и выполнены указанные в нем необходимые инициализации. Отдельно от базовой конфигурации, в

файле могут содержаться описание перехватчиков (interceptors), view resolvers, настройки локализации и др…

- **КАКОВЫ РАЗЛИЧНЫЕ СПОСОБЫ НАСТРОИТЬ КЛАСС КАК SPRING BEAN?**

Существует несколько способов работы с классами в Spring:
XML конфигурация:

```
<bean name="myBean" class="jsehelper.spring.beans.MyBean"></bean>
```

Java based конфигурация. Все настройки и указания бинов прописываются в java коде:

```
@Configuration
@ComponentScan(value = "jsehelper.spring.main")
public class MyConfiguration {

    @Bean
    public MyService getService() {
        return new MyService();
    }
}
```

Для извлечения бина из контекста используется следующий подход:

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(MyConfiguration.class);
MyService service = ctx.getBean(MyService.class);
```

Annotation based конфигурация. Можно использовать внутри кода аннотации @Component, @Service, @Repository, @Controller для указания классов в качестве спринг бинов. Для их поиска и управления контейнером прописывается настройка в xml файле:

```
<context:component-scan base-package="jsehelper.spring"/>
```

- **КАКИЕ ВЫ ЗНАЕТЕ РАЗЛИЧНЫЕ SCOPE У SPRING BEAN?**

В Spring предусмотрены различные области времени действия бинов:

- singleton – может быть создан только один экземпляр бина. Этот тип используется спрингом по умолчанию, если не указано другое. Следует осторожно использовать публичные свойства класса, т.к. они не будут потокобезопасными.
- prototype – создается новый экземпляр при каждом запросе.
- request – аналогичен prototype, но название служит пояснением к использованию бина в веб приложении. Создается новый экземпляр при каждом HTTP request.
- session – новый бин создается в контейнере при каждой новой HTTP сессии.
- global-session: используется для создания глобальных бинов на уровне сессии для Portlet приложений.