Вопросы и ответы на собеседование по Spring

(1-15)

1). Что такое Spring Framework?

Spring Framework (или просто **Spring**) - это популярный и мощный фреймворк для разработки приложений на языке Java. Он предоставляет комплексный набор инструментов и функций для упрощения разработки Java-приложений, особенно в области создания **веб-приложений** и **веб-сервисов**. Spring был создан для решения проблем, связанных с разработкой сложных и масштабируемых приложений, и он стал одним из самых популярных фреймворков для разработки на Java.

Spring Framework является открытым исходным кодом и имеет большое сообщество разработчиков, что делает его надежным и активно поддерживаемым фреймворком для создания разнообразных Java-приложений.

2). Из каких частей состоит Spring Framework?

Вот 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** – отвечает за context web-приложений.
- **Spring MVC framework** – реализация паттерна MVC для построения Web-приложений.

Read & Write Code

3). В чем преимущества Spring?

Вот некоторые из ключевых особенностей Spring Framework:

1.**IoC** (Inversion of Control): Spring предлагает механизм инверсии управления, который позволяет управлять жизненным циклом и зависимостями объектов в приложении. **Контейнер** Spring заботится о связывании зависимостей различных объектов вместо создания или поиска зависимых объектов.Это делает код более гибким и управляемым.

2.**DI** (Dependency Injection): Spring поддерживает внедрение зависимостей, что упрощает создание и управление зависимостями между компонентами приложения.

3.Аспектно-ориентированное программирование (**AOP**): Spring предоставляет поддержку AOP для реализации перехватчиков (aspects), таких как логирование, безопасность и другие аспекты приложения.

4.**Модульность**: Spring разделен на модули, что позволяет вам выбирать только необходимые части фреймворка для вашего проекта.

5.**Поддержка транзакций**: Spring предоставляет механизм управления транзакциями в приложении, что делает его отличным выбором для работы с базами данных.

6.**Поддержка веб-приложений**: Spring **MVC** - часть Spring Framework, предоставляющая мощные средства для разработки веб-приложений и API.

7.**Тестирование**: Spring обеспечивает хорошую поддержку для тестирования приложений с помощью модулей, таких как Spring Test и JUnit.

4). Объясните суть паттерна DI или IoC

Dependency injection (DI) – паттерн проектирования и архитектурная модель, так же известная как **Inversion of Control (IoC)**.
В полном соответствии с принципом единой обязанности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.
Суть DI заключается в том, что объекты не создают свои зависимости сами, а вместо этого получают их из внешних источников. 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. Термин “dependency injection” немного неправильный, т.к. внедряются не зависимости, а функциональность или ресурсы. 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.

5). 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. с помощью XML-файлов, 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.

Read & Write Code

5). Продолжение

К типам реализации внедрения зависимостей в Spring относят:

Constructor Dependency Injection — это тип внедрения зависимостей, при котором зависимости компонента предоставляются ему в его конструкторе (или конструкторах).

```
public class ConstructorInjection {

private Dependency dependency;

  public ConstructorInjection(Dependency dependency) {
  this.dependency = dependency;
  }
}
```

Setter Dependency Injection — контейнер IoC внедряет зависимости компонента в компонент через методы установки в стиле JavaBean.

```
public class SetterInjection {
private Dependency dependency;
  public void setDependency(Dependency dependency) {
  this.dependency = dependency;
  }
}
```

6). Объясните работу BeanFactory в Spring

BeanFactory – это реализация паттерна Фабрика, является основным интерфейсом, отвечающим за создание и управление бинами в Spring-приложениях. Он служит как контейнер для бинов. BeanFactory обычно настраивается с помощью конфигурационных файлов, таких как XML-файлы, аннотации или Java-конфигурации. Он понимает, какие бины должны быть созданы и как они связаны друг с другом.
Один из ключевых моментов - **ленивая инициализация**. BeanFactory создает бины только в тот момент, когда они действительно нужны, что способствует оптимизации использования ресурсов.
Существует несколько реализаций BeanFactory, самая используемся – “**org.springframework.beans.factory.xml.XmlBeanFactory**”. Она загружает бины на основе конфигурационного XML-файла. Чтобы создать XmlBeanFactory передайте конструктору InputStream, например:

```
BeanFactory factory = new XmlBeanFactory(new FileInputStream("myBean.xml"));
```

После этой строки фабрика знает о бинах, но их экземпляры еще не созданы. Чтобы инстанцировать бин нужно вызвать метод getBean().

Например:

```
myBean bean1 = (myBean)factory.getBean("myBean");
```

Read & Write Code

7). В чем роль ApplicationContext в Spring?

В то время, как BeanFactory используется в простых приложениях, **Application Context** – это более сложный контейнер. Как и BeanFactory он может быть использован для загрузки и связывания бинов, но еще он предоставляет:

1. возможность получения текстовых сообщений, в том числе поддержку интернационализации;
2. общий механизм работы с ресурсами;
3. события для бинов, которые зарегестрированы как слушатели.

Из-за большей функциональности **рекомендуется** использование Application Context вместо BeanFactory. Последний используется только в случаях нехватки ресурсов, например при разработке для мобильных устройств.

Существует 3 основных реализации Application context:

1. **ClassPathXmlApplicationContext** – получает информацию из xml-файла, находящегося в classpath. Пример: ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);
2. **FileSystemXmlApplicationContext** – получает информацию из xml-файла. Например: ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);
3. **XmlWebApplicationContext** – получает информацию из xml-файла за пределами web-приложения.

Read & Write Code

8). Что такое AOP? Как это относиться к IoC?

AOP (Aspect-Oriented Programming) - это методология программирования, которая позволяет выделять и абстрагировать "аспекты" поведения приложения, такие как логирование, безопасность, транзакции и другие поперечно-распределенные задачи, и внедрять их в приложение независимо от основной логики программы.
AOP решает проблему размещения кода, который не является частью бизнес-логики, внутри классов, что может сделать их менее читаемыми и сложными для поддержки. Вместо этого, вы можете создать аспекты, которые содержат этот поперечно-распределенный код, и затем применять эти аспекты к классам или методам, где он должен выполняться.
Как это связано с IoC (Inversion of Control)? Внедрение зависимостей (DI) и управление контейнером (IoC) являются фундаментальными концепциями Spring, которые позволяют разделить создание объектов и их зависимостей от основной логики приложения. AOP в Spring интегрируется с DI и IoC для обеспечения более гибкого управления поведением вашего приложения.
Как это работает:

1. **Аспекты**: В Spring AOP вы можете создать аспекты, которые описывают поведение, которое вы хотите выделить. Например, аспект для логирования.
2. Цели (**Targets**): Цели - это компоненты вашего приложения, к которым вы хотите применить аспект. Это могут быть классы или методы.
3. Советы (**Advices**): Аспекты содержат советы, которые представляют собой фрагменты кода, выполняющиеся в определенные моменты времени, например, до выполнения метода, после выполнения метода и так далее.
4. Точки среза (**Pointcuts**): Точки среза определяют, когда и где должны выполняться советы. Они определяют условия для применения аспекта.

Read & Write Code

9). Что такое DAO?

Spring предоставляет различные возможности и механизмы для поддержки **DAO (Data Access Object)** в приложениях. DAO - это шаблон проектирования, который используется для абстрагирования доступа к данным и обеспечения уровня абстракции между приложением и базой данных. Вот как Spring поддерживает DAO:

1). JDBC Template: Spring предоставляет класс JdbcTemplate, который упрощает работу с JDBC (Java Database Connectivity). Он обрабатывает открытие и закрытие соединений с базой данных, управление транзакциями и обработку исключений. Вы можете использовать JdbcTemplate для выполнения SQL-запросов и обновления данных в базе данных.
Пример:

```
@Autowired
private JdbcTemplate jdbcTemplate;

public void insertData(String data) {
 jdbcTemplate.update("INSERT INTO my_table (data) VALUES (?)", data);
 }
```

2). Интеграция с **ORM** (Object-Relational Mapping): Spring обеспечивает интеграцию с различными ORM-фреймворками, такими как Hibernate, JPA (Java Persistence API), и JDO (Java Data Objects). Это позволяет вам создавать DAO, которые работают с объектами Java, а не с SQL-запросами напрямую. Spring управляет жизненным циклом сессии или Entity Manager'a и обеспечивает транзакционность.
Пример с Hibernate:

```
@Repository
public class MyEntityDao {
 @Autowired
 private SessionFactory sessionFactory;

 public void save(MyEntity entity) {
 Session session = sessionFactory.getCurrentSession();
session.save(entity);
   }
}
```

9). Продолжение

3). Декларативная транзакционность: Spring позволяет определять транзакции для методов DAO с помощью аннотаций или XML-конфигурации. Это обеспечивает управление транзакциями на уровне методов без необходимости явно управлять транзакциями.
Пример с аннотациями:

```
@Transactional
public class MyEntityService {
 @Autowired
 private MyEntityDao entityDao;

  public void saveEntity(MyEntity entity) {
entityDao.save(entity);
   }
}
```

4). **Поддержка исключений**: Spring предоставляет возможность определять, какие исключения должны считаться checked или unchecked (что влияет на поведение транзакции) при работе с DAO.

5). Поддержка JNDI и DataSource: Spring упрощает настройку и использование источников данных (DataSource), что делает его более гибким при переносе приложения между разными средами.

10). Что такое Been?

Spring Beans — это объекты Java, которые инициализируются контейнером Spring IoC. Spring контейнер обеспечивает удобный способ управления их жизненным циклом и конфигурацией.

Read & Write Code

11). Какие вы знаете различные scope у Spring Bean?

Какие вы знаете различные scope у Spring Bean?

В Spring предусмотрены различные области времени действия бинов:

1. **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.
2. **prototype** — создается новый экземпляр при каждом запросе.
3. **request** — аналогичен prototype, но название служит пояснением к использованию бина в веб приложении. Создается новый экземпляр при каждом HTTP request.
4. **session** — новый бин создается в контейнере при каждой новой HTTP сессии.
5. **global-session**: используется для создания глобальных бинов на уровне сессии для Portlet приложений.

По умолчанию Spring Bean инициализируется как **singleton**.

12) Что такое жизненный цикл Bean?

Жизненный цикл Spring бина — время существования класса. Spring бины инициализируются при инициализации Spring контейнера и происходит внедрение всех зависимостей. Когда контейнер уничтожается, то уничтожается и всё содержимое. Если нам необходимо задать какое-либо действие при инициализации и уничтожении бина, то нужно воспользоваться методами **init**() и **destroy**(). Для этого можно использовать аннотации **@PostConstruct** и **@PreDestroy()**.

```
 @PostConstruct
 public void init(){
 System.out.println("Bean init method called");
}
```

```
@PreDestroy
 public void destroy(){
 System.out.println("Bean destroy method called");
}
```

Или через xml конфигурацию:

```
<bean name="myBean" class="ru.readandwritecode.spring.MyBean"
 init-method="init" destroy-method="destroy">
 <property name="someProp" ref="someProp"></property>
</bean>
```

13). Как получить объекты ServletContext и ServletConfig внутри Spring Bean?

Доступны два способа для получения основных объектов контейнера внутри бина:

- Реализовать один из Spring*Aware (ApplicationContextAware, ServletContextAware, ServletConfigAware и др.) интерфейсов.
- Использовать автоматическое связывание @Autowired в спринг. Способ работает внутри контейнера спринг.

```
@Autowired
ServletContext servletContext;
```

14). Являются ли Singleton Beans потокобезопасными?

Нет, singleton bean-компоненты **не** потокобезопасны, поскольку потокобезопасность связана с выполнением, тогда как singleton — это шаблон проектирования, ориентированный на создание.

Безопасность потоков зависит только от самой реализации компонента.

15). Перечислите некоторые важные аннотации

Важные аннотации:

@Required;
@Autowired;
@Qualifier;
@Resource;
@PostConstruct;
@PreDestroy;
@Component;
@Controller;
@Repository;
@Service;
@Transactional.