

Ответы на вопросы на собеседовании Spring Framework (часть 2).

👤 Vasyl K ⌚ 8:24:00

💬 Добавить комментарий

• ЧТО ТАКОЕ ЖИЗНЕННЫЙ ЦИКЛ SPRING 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`.

```
1  @PostConstruct
2  public void init() {
3      System.out.println("Bean init method called");
4  }
5
6  @PreDestroy
7  public void destroy() {
8      System.out.println("Bean destroy method called");
9  }
```

Или через xml конфигурацию:

```
1  <bean name="myBean" class="jsehelper.spring.beans.MyBean" init-method="init" destroy-method="destroy">
2      <property name="someProp" ref="someProp"></property>
3  </bean>
```

• ОБЪЯСНИТЕ РАБОТУ BEANFACTORY В SPRING.

BeanFactory – это реализация паттерна Фабрика, его функциональность покрывает создание бинов. Так как эта фабрика знает многое об объектах приложения, то она может создавать связи между объектами на этапе создания экземпляра. Существует несколько реализаций BeanFactory, самая используемая – 

"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:

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

• 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.

```
1 | @Autowired  
2 | ServletContext servletContext;
```

• 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:

- возможность получения текстовых сообщений, в том числе поддержку интернационализации;
- общий механизм работы с ресурсами;
- события для бинов, которые зарегистрированы 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

• КАК ВЫГЛЯДИТ ТИПИЧНАЯ РЕАЛИЗАЦИЯ МЕТОДА ИСПОЛЬЗУЯ SPRING?

Для типичного Spring-приложения нам необходимы следующие файлы:

- Интерфейс, описывающий функционал приложения
- Реализация интерфейса, содержащая свойства, сэттеры-гэттеры, функции и т.п.
- Конфигурационный XML-файл Spring'a.
- Клиентское приложение, которое использует функцию.

- **ЧТО ТАКОЕ СВЯЗЫВАНИЕ В SPRING И РАССКАЖИТЕ ОБ АННОТАЦИИ @AUTOWIRED?**

Процесс внедрения зависимостей в бины при инициализации называется Spring Bean Wiring. Считается хорошей практикой задавать явные связи между зависимостями, но в Spring предусмотрен дополнительный механизм связывания @Autowired. Аннотация может использоваться над полем или методом для связывания по типу. Чтобы аннотация заработала, необходимо указать небольшие настройки в конфигурационном файле спринг с помощью элемента context:annotation-config.

- **КАКОВЫ РАЗЛИЧНЫЕ ТИПЫ АВТОМАТИЧЕСКОГО СВЯЗЫВАНИЯ В SPRING?**

Существует четыре вида связывания в спринг:

- autowire byName,
- autowire byType,
- autowire by constructor,
- autowiring by @Autowired and @Qualifier annotations

- **ПРИВЕДИТЕ ПРИМЕР ЧАСТО ИСПОЛЬЗУЕМЫХ АННОТАЦИЙ SPRING.**

@Controller – класс фронт контроллера в проекте Spring MVC.

@RequestMapping – позволяет задать шаблон маппинга URI в методе обработке контроллера.

@ResponseBody – позволяет отправлять Object в ответе. Обычно используется для отправки данных формата XML или JSON.

@PathVariable – задает динамический маппинг значений из URI внутри аргументов метода обработчика.

@Autowired – используется для автоматического связывания зависимостей в spring beans.

@Qualifier – используется совместно с @Autowired для уточнения данных связывания, когда возможны коллизии (например одинаковых имен\типов).

@Service – указывает что класс осуществляет сервисные функции.

@Scope – указывает scope у spring bean.

@Configuration, @ComponentScan и @Bean – для java based configurations.

AspectJ аннотации для настройки aspects и advices, @Aspect, @Before, @After, @Around, @Pointcut и др.

- **МОЖЕМ ЛИ МЫ ПОСЛАТЬ ОБЪЕКТ КАК ОТВЕТ МЕТОДА ОБРАБОТЧИКА КОНТРОЛЛЕРА?**

Да, это возможно. Для этого используется аннотация `@ResponseBody`. Так можно отправлять ответы в виде JSON, XML в `restful` веб сервисах.

- **ЯВЛЯЕТСЯ ЛИ SPRING BEAN ПОТОКОБЕЗОПАСНЫМ?**

По умолчанию бин задается как синглтон в Spring. Таким образом все публичные переменные класса могут быть изменены одновременно из разных мест. Так что – нет, не является. Однако поменяв область действия бина на `request`, `prototype`, `session` он станет потокобезопасным, но это скажется на производительности.

- **КАК СОЗДАТЬ APPLICATIONCONTEXT В ПРОГРАММЕ JAVA?**

В независимой Java программе `ApplicationContext` можно создать следующим образом: `AnnotationConfigApplicationContext` – при использовании Spring в качестве автономного приложения можно создать инициализировать контейнер с помощью аннотаций. Пример:

```
1 | ApplicationContext context = new AnnotationConfigApplicationContext("bean.xml");
```

`ClassPathXmlApplicationContext` – получает информацию из `xml`-файла, находящегося в `classpath`. Пример:

```
1 | ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
```

`FileSystemXmlApplicationContext` – получает информацию из `xml`-файла, но с возможностью загрузки файла конфигурации из любого места файловой системы. Пример:

```
1 | ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
```

`XmlWebApplicationContext` – получает информацию из `xml`-файла за пределами `web`-приложения.

- **МОЖЕМ ЛИ МЫ ИМЕТЬ НЕСКОЛЬКО ФАЙЛОВ КОНФИГУРАЦИИ SPRING?**

С помощью указания `contextConfigLocation` можно задать несколько файлов конфигурации Spring. Параметры указываются через запятую или пробел:

```
1 | <servlet>
2 |   <servlet-name>appServlet</servlet-name>
3 |   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4 |   <init-param>
5 |     <param-name>contextConfigLocation</param-name>
6 |     <param-value>/WEB-INF/spring/appServlet/servlet-context.xml,/WEB-INF/spring/appServlet/servlet-jdbc.xml</param-value>
7 |   </init-param>
8 |   <load-on-startup>1</load-on-startup>
9 | </servlet>
```

Поддерживается возможность указания нескольких корневых файлов конфигурации Spring:

```
1 <context-param>
2   <param-name>contextConfigLocation</param-name>
3   <param-value>/WEB-INF/spring/root-context.xml /WEB-INF/spring/root-security.xml</param-value>
4 </context-param>
```

Файл конфигурации можно импортировать:

```
1 <beans:import resource="spring-jdbc.xml"/>
```

- **КАК ВНЕДРИТЬ JAVA.UTIL.PROPERTIES В SPRING BEAN?**

Для возможности использования Spring EL для внедрения свойств (properties) в различные бины необходимо определить propertyConfigurer bean, который будет загружать файл свойств.

```
1 <bean id="propertyConfigurer" class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">
2   <property name="location" value="/WEB-INF/application.properties">
3 </property></bean>
4
5 <bean class="jsehelper.spring.dao.impl.EmployeeDaoImpl">
6   <property name="maxReadResults" value="${results.read.max}">
7 </property></bean>
```

Или через аннотации:

```
1 @Value("${maxReadResults}")
2 private int maxReadResults;
```

- **КАК НАСТРАИВАЕТСЯ СОЕДИНЕНИЕ С БД В SPRING?**

Используя datasource

"org.springframework.jdbc.datasource.DriverManagerDataSource". Пример:

```
1 <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
2   <property name="driverClassName">
3     <value>org.hsqldb.jdbcDriver</value>
4   </property>
5   <property name="url">
6     <value>jdbc:hsqldb:db/appfuse</value>
7   </property>
8   <property name="username"><value>sa</value></property>
9   <property name="password"><value></value></property>
10 </bean>
```

- **КАК СКОНФИГУРИРОВАТЬ JNDI НЕ ЧЕРЕЗ DATASOURCE В APPLICATIONCONTEXT.XML?**

Используя "org.springframework.jndi.JndiObjectFactoryBean". Пример

```
1 <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
2   <property name="jndiName">
3     <value>java:comp/env/jdbc/appfuse</value>
4   </property>
5 </bean>
```

- **КАКИМ ОБРАЗОМ МОЖНО УПРАВЛЯТЬ ТРАНЗАКЦИЯМИ В SPRING?**

Транзакциями в Spring управляют с помощью Declarative Transaction Management (программное управление). Используется аннотация @Transactional для описания необходимости управления транзакцией. В файле конфигурации нужно добавить настройку transactionManager для DataSource.

```
1 <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
2   <property name="dataSource" ref="dataSource">
3 </property></bean>
```

- **КАКИМ ОБРАЗОМ SPRING ПОДДЕРЖИВАЕТ DAO?**

Spring DAO предоставляет возможность работы с доступом к данным с помощью технологий вроде JDBC, Hibernate в удобном виде. Существуют специальные классы: JdbcDaoSupport, HibernateDaoSupport, JdoDaoSupport, JpaDaoSupport.

Класс HibernateDaoSupport является подходящим суперклассом для Hibernate DAO. Он содержит методы для получения сессии или фабрики сессий. Самый популярный метод – getHibernateTemplate(), который возвращает HibernateTemplate. Этот темплейт оборачивает checked-исключения Hibernate в runtime-исключения, позволяя вашим DAO оставаться независимыми от исключений Hibernate.

Пример:

```
1 public class UserDAOHibernate extends HibernateDaoSupport {
2     public User getUser(Long id) {
3         return (User) getHibernateTemplate().get(User.class, id);
4     }
5     public void saveUser(User user) {
6         getHibernateTemplate().saveOrUpdate(user);
7         if (log.isDebugEnabled()) {
8             log.debug("userId set to: " + user.getID());
9         }
10    }
11    public void removeUser(Long id) {
12        Object user = getHibernateTemplate().load(User.class, id);
13        getHibernateTemplate().delete(user);
14    }
15 }
```

- **КАК ИНТЕГРИРОВАТЬ SPRING И HIBERNATE?**

Для интеграции Hibernate в Spring необходимо подключить зависимости, а так же настроить файл конфигурации Spring. Т.к. настройки несколько отличаются между проектами и версиями, то смотрите официальную документацию Spring и Hibernate для уточнения настроек для конкретных технологий.

- **КАК ЗАДАЮТСЯ ФАЙЛЫ МАППИНГА HIBERNATE В SPRING?**

Через applicationContext.xml в web/WEB-INF. Например:

```

1 <property name="mappingResources">
2   <list>
3     <value>org/appfuse/model/User.hbm.xml</value>
4   </list>
5 </property>

```

- **КАК ДОБАВИТЬ ПОДДЕРЖКУ SPRING В WEB-ПРИЛОЖЕНИЕ**

Достаточно просто указать ContextLoaderListener в web.xml файле приложения:

```

1 <listener>
2   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
3 </listener>

```

- **МОЖНО ЛИ ИСПОЛЬЗОВАТЬ XYZ.XML ВМЕСТО APPLICATIONCONTEXT.XML?**

ContextLoaderListener – это ServletContextListener, который инициализируется когда ваше web-приложение стартует. По-умолчанию оно загружает файл WEB-INF/applicationContext.xml. Вы можете изменить значение по-умолчанию, указав параметр contextConfigLocation. Пример:

```

1 <listener>
2   <listener-class>org.springframework.web.context.ContextLoaderListener
3     <context-param>
4       <param-name>contextConfigLocation</param-name>
5       <param-value>/WEB-INF/xyz.xml</param-value>
6     </context-param>
7   </listener-class>
8 </listener>

```