1. Какво представляват Beans в Spring?
2. PostConstruct анотация?
3. PreDestroy анотация?
4. Има ли разлика между appilcation.properties и application.yaml (отнася се за предназначението).
5. Какви Beans scopes познавате, дайте примери и какви са разликите между тях?
6. Какво е Spring Boot?
7. Само на Java ли се пишат спринг приложения?

1.Какво представляват Beans в Spring?

1. Overview

Различни обекти между които обаче общото е че се менажират от Спринг. Тоест те живеят във Application context спринг ги инициализира и той ги дестройва. Или не използваме New оператора когато

Ги викаме. За да получим достъп до тях използваме Inversion of control and Dependency Injection.

Bean is a key concept of the Spring Framework. So, understanding this notion is crucial to get the hang of the framework and use it in an effective way. Unfortunately, there aren't clear answers to the simple question of what a Spring bean really is. Some explanations go to such a low level that the big picture is missed, whereas others are too vague. This tutorial will try to shed light on the topic, starting with a description in the official documentation.

Bean е ключова концепция на Spring Framework. Така че разбирането на това понятие е от решаващо значение, за да овладеете рамката и да я използвате по ефективен начин. За съжаление, няма ясни отговори на простия въпрос какво всъщност представлява Spring Bean . Някои обяснения стигат до толкова ниско ниво, че голямата картина се пропуска, докато други са твърде неясни. Този урок ще се опита да хвърли светлина върху темата, като се започне с описание в официалната документация.

2. Bean Definition

Here's a definition of beans in [the Spring Framework documentation](https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-introduction):

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

This definition is concise and gets to the point but fails to elaborate on an important element: the Spring IoC container. Let's take a closer look to see what it is and the benefits it brings in.

Ето дефиниция за bean в документацията на Spring Framework:

В Spring обектите, които формират гръбнака на вашето приложение и които се управляват от Spring IoC контейнера, се наричат beans. Bean е обект, който се инстанцира, сглобява и управлява по друг начин от Spring IoC контейнер.

Тази дефиниция е кратка и стига до същността, но не успява да разработи важен елемент: контейнера Spring IoC. Нека разгледаме по-отблизо, за да видим какво представлява и ползите, които носи.

3. Inversion of Control

Simply put, [Inversion of Control](https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring) (IoC) is **a process in which an object defines its dependencies without creating them.** This object delegates the job of constructing such dependencies to an IoC container.

Най-просто казано, инверсията на контрола (IoC) е процес, при който обект дефинира своите зависимости, без да ги създава. Този обект делегира работата по конструирането на такива зависимости на IoC контейнер.

Resource: <https://www.baeldung.com/spring-bean>

2.PostConstruct анотация?

Слага се върху метод който трябва да бъде извикан след като е инициализиран един спринг Bean.

Spring calls the methods annotated with @PostConstruct only once, just after the initialization of bean properties. Keep in mind that these methods will run even if there's nothing to initialize.

The method annotated with @PostConstruct can have any access level, but it can't be static.

One possible use of @PostConstruct is populating a database. For instance, during development, we might want to create some default users:

Spring извиква методите, анотирани с @PostConstruct само веднъж, непосредствено след инициализацията на свойствата на bean. Имайте предвид, че тези методи ще се изпълняват, дори ако няма нищо за инициализиране.

Методът, анотиран с @PostConstruct, може да има всяко ниво на достъп, но не може да бъде статичен.

Едно възможно използване на @PostConstruct е попълването на база данни. Например, по време на разработката, може да искаме да създадем някои потребители по подразбиране:

@Component

**public** **class** **DbInit** {

@Autowired **private** UserRepository userRepository;

@PostConstruct **private** **void** **postConstruct**() {

**User** admin = **new** **User**("admin", "admin password");

**User** normalUser = **new** **User**("user", "user password"); userRepository.save(admin, normalUser);

}

}

The above example will first initialize *UserRepository* and then run the *@PostConstruct* method.

Горният пример първо ще инициализира UserRepository и след това ще изпълни метода @PostConstruct.

3.PreDestroy анотация?

Се слага върху метод който се извиква от контейнера когато се дестройва някакъв Bean или при Shut down на Application server.

A method annotated with @PreDestroy runs only once, just before Spring removes our bean from the application context.

Same as with @PostConstruct, the methods annotated with @PreDestroy can have any access level but can't be static.

Метод, анотиран с @PreDestroy, се изпълнява само веднъж, точно преди Spring да премахне нашия bean от контекста на приложението.

Както при @PostConstruct, методите, анотирани с @PreDestroy, могат да имат всяко ниво на достъп, но не могат да бъдат статични.

Resource: <https://www.baeldung.com/spring-postconstruct-predestroy>

For reading and more information: https://www.baeldung.com/running-setup-logic-on-startup-in-spring

4.Има ли разлика между appilcation.properties и application.yaml (отнася се за предназначението).

Не и двата файла са с еднакво предназначение като формата на информация в тях е различен, като application.yml е по удобен и по четим за ХОРА.

Относно предназначението няма разлика. Разликата е във формата

Повече информация тук: https://www.baeldung.com/spring-boot-yaml-vs-properties

5.Какви Beans scopes познавате, дайте примери и какви са разликите между тях?

Singleton , Prototype разликата е че първия се инициализира един единствен път а втория се създава всеки път. Има и други скоупове.

**@Bean**

**@Scope("singleton") <- Can be omitted**

**public Student student() {**

**return new Student();**

**}**

* + Singleton - When we define a bean with the singleton scope, the container creates a single instance of that bean; all requests for that bean name will return the same object, which is cached. Any modifications to the object will be reflected in all references to the bean. This scope is the default value if no other scope is specified.

Когато дефинираме bean с единичния обхват, контейнерът създава единичен екземпляр на този bean; всички заявки за това име на бийн ще връщат един и същ обект, който е кеширан. Всички модификации на обекта ще бъдат отразени във всички препратки към бина. Този обхват е стойността по подразбиране, ако не е посочен друг обхват.

* + Prototype - Will return a different instance every time it is requested from the container

Ще връща различен екземпляр всеки път, когато бъде поискан от контейнера

Другите са достъпни само в контекста на уеб-ориентирано приложение. Използват се по-рядко на практика.

* + Request - The request scope creates a bean instance for a single HTTP request.

Обхватът на заявката създава екземпляр на bean за единична HTTP заявка

* + Session - the session scope creates a bean instance for an HTTP Session

Докато обхватът на сесията създава екземпляр на bean за HTTP сесия.

Resource: https://www.baeldung.com/spring-bean-scopes

6.Какво е Spring Boot?

Начин по който можем да си сетнем нашето приложение.

Spring Boot is an open source, microservice-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase. The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

Spring Boot е базирана на микросервизи Java уеб рамка с отворен код. Рамката Spring Boot създава напълно готова за производство среда, която е напълно конфигурируема с помощта на предварително изградения си код в своята кодова база. Микросервизната архитектура предоставя на разработчиците напълно затворено приложение, включително вградени сървъри на приложения.

7.Само на Java ли се пишат спринг приложения?

Не , може и на:

Kotlin, Apache Groovy and Dynamic Language Support like JRuby, Groovy 1.8+, BeanShell 2.0 .

More interview question for spring : <https://www.interviewbit.com/spring-boot-interview-questions/>