sourcemind

Advanced Java and Spring course

Traditional Java Web Applications

Course agenda

- Traditional Java Web Applications
- Spring Fundamentals
- Spring Boot
- Spring Web
- Spring Security
- Deployment

Lesson agenda

- Spring intro
- IoC Controller
- Spring Bean
- Dependency injection

Advantages of Spring

- Predefined Templates
- Loose Coupling
- Easy to test
- Lightweight
- Fast Development
- Powerful abstraction
- Declarative support

IoC Container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

There are two types of IoC containers:

- BeanFactory
- ApplicationContext

Spring bean

The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. Bean definition contains the information called configuration metadata, which is needed for the container to know the following:

- How to create a bean
- Bean's lifecycle details
- Bean's dependencies

Bean properties

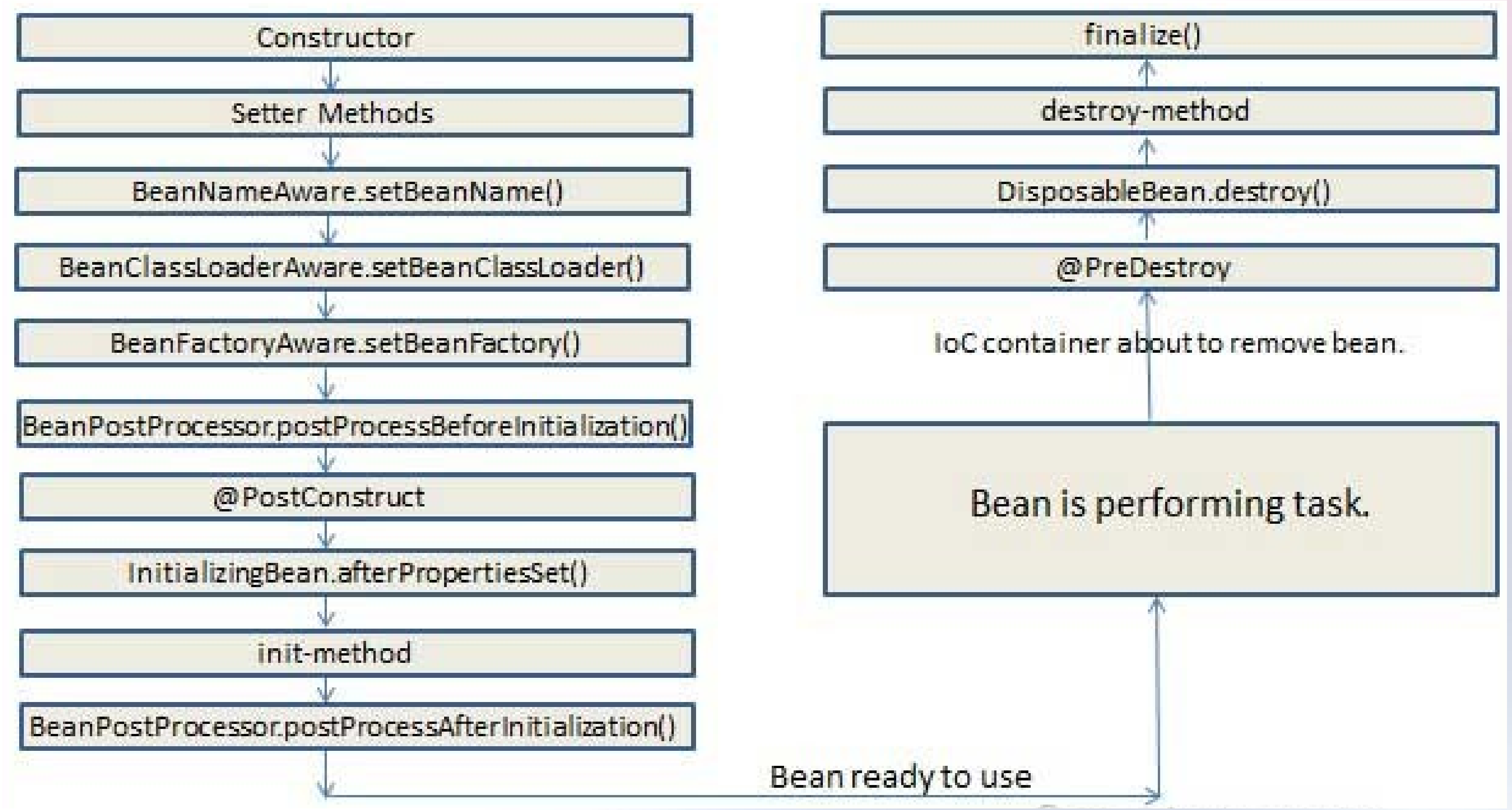
- class
- name
- scope
- constructor-arg
- properties
- autowiring mode
- lazy-initialization mode
- initialization method
- destruction method

Bean scopes

- Singleton: This scopes the bean definition to a single instance per Spring IoC container (default).
- Prototype: This scopes a single bean definition to have any number of object instances.
- Request: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
- Session: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
- Global-session: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

sourcemind

Bean lifecycle


Constructor
Setter Methods
BeanNameAware.setBeanName()
BeanClassLoaderAware.setBeanClassLoader()
BeanFactoryAware.setBeanFactory()
BeanPostProcessor.postProcessBeforeInitialization()
@PostConstruct
InitializingBean.afterPropertiesSet()
init-method
BeanPostProcessor.postProcessAfterInitialization()
Bean ready to use
Bean is performing task.
IoC container about to remove bean.
@PreDestroy
DisposableBean.destroy()
destroy-method
finalize()

Dependency Injection

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing.

Spring framework provides two ways to inject dependency

- By Constructor
- By Setter method

Java Based Configuration

```
<beans>
     <bean
          id="helloWorld"

class="com.sourcemind.HelloWorld"
     />
</beans>
```

XML to Annotations

```
@Configuration
public class HelloWorldConfig {
   @Bean
   public HelloWorld helloWorld()
{
      return new HelloWorld();
   }
}
```

ClassPathXmlApplicationContext

XML to Annotations

AnnotationConfigApplicationContext

sourcemind

Injecting Bean Dependencies

```
@Configuration
public class AppConfig {

    @Bean(name = ”constructor”)
    public Foo foo() {
        return new Foo(bar());
    }
    @Bean(name = ”setter”)
    public Foo foo() {
        Foo foo= new Foo();
        foo.setBar(bar());
        return foo;
    }
    @Bean
    public Bar bar() {
        return new Bar();
    }
}
```

The foo bean named **constructor** receives a reference to bar via the constructor injection.

The foo bean named **setter** receives a reference to bar via the setter injection.

sourcemind

Spring annotations

- The **@Import** Annotation
    - Allows loading @Bean definitions from another configuration class.
- Lifecycle Callbacks
    - The **@Bean** annotation supports specifying arbitrary initialization and destruction callback methods. @Bean**(initMethod = "init", destroyMethod = "cleanup").**
- Specifying Bean Scope
    - The default scope is singleton, but you can override this with the **@Scope** annotation.
- Lazy Initialization
    - By default, Spring creates all singleton beans eagerly at the startup/bootstrapping of the application context. **@Lazy** annotation allows overriding that and creating the bean whenever we request it.

sourcemind

@Component vs @Bean

- **@Component auto-detects and configures the beans using classpath scanning whereas @Bean explicitly declares a single bean,** So if you want to create a bean for the class then you need to create a method that returns the class instance and needs to declare that method with @Bean annotation.
- **@Component does not decouple the declaration of the bean from the class definition whereas @Bean decouples the declaration of the bean from the class definition.** So we can keep the class that is completely independent of spring dependency. Mainly the @Bean annotation is used for added spring beans for the class which is defined on external jars.
- **@Component is a class-level annotation whereas @Bean is a method level annotation and the name of the method serves as the bean name.**
- **@Component doesn’t need to be used with the @Configuration annotation whereas @Bean annotation has to be used within the class which is annotated with @Configuration.**
- **We cannot create a bean of a class using @Component if the class is outside the spring container whereas we can create a bean of a class using @Bean even if the class is present outside the spring container.**

sourcemind