Spring Framework Explained

What Makes The Magic Work

Mikaël Francoeur

Introduction

- ▶ Bean/BeanDefinition
- ▶ BeanFactory
- ▶ ApplicationContext
- ▶ ClassPathBeanDefinitionScanner
- ▶ BeanPostProcessor
- ▶ Proxies

Bean and BeanDefinition

- Bean
  - An Object that is managed by the container
  - Often a singleton, but not always
  - Often instantiated by the container, but not always

- BeanDefinition
  - A set of metadata indicating how to create a bean
  - getName()
  - getBeanClassName()
  - getFactoryMethodName()

BeanFactory

- "the root interface for accessing a Spring container" (javadoc)

```java
public interface BeanFactory {
  <T> T getBean(String name, Class<T> requiredType) throws BeansException;
  // ...
}
```

- Related interfaces
  - SingletonBeanRegistry
  - ListableBeanRegistry
  - BeanDefinitionRegistry
- Implementations
  - DefaultListableBeanFactory
  - SimpleJndiBeanFactory

```java
@Data static class MyBeanRequiresInjection {
    private MyInjectedBean myField;
}


static class MyInjectedBean {
    @Getter private final String myValue = "myInjectedProperty";
}


@Test void test() {
    BeanFactory beanFactory = new DefaultListableBeanFactory();

    ((BeanDefinitionRegistry) beanFactory).registerBeanDefinition( beanName: "myInjectedBean",
        BeanDefinitionBuilder
            .genericBeanDefinition(MyInjectedBean.class)
            .getBeanDefinition());


    ((BeanDefinitionRegistry) beanFactory).registerBeanDefinition( beanName: "myBean",
        BeanDefinitionBuilder
            .genericBeanDefinition(MyBeanRequiresInjection.class)
            .addAutowiredProperty("myField")
            .getBeanDefinition());

    MyBeanRequiresInjection myPojo = beanFactory.getBean( name: "myBean", MyBeanRequiresInjection.class);
    System.out.println(myPojo.getMyField().getMyValue()); // myInjectedProperty
}
```

Demo!

Inspecting the BeanFactory

ApplicationContext

Manages the BeanFactory
- Controls the BeanFactory lifecycle
- Provides extension points
- Initializes singletons eagerly

Adds Enterprise features
- Events
- Profiles and Properties
- Resources
- Messages and i18n

AnnotationConfigApplicationContext (and variations)
- Uses a ClassPathBeanDefinitionScanner to register bean definitions

ClassPathBeanDefinitionScanner

▶ Registers BeanDefinitions with the BeanFactory

▶ Detects @Component, @Service, @Repository, @ManagedBean, @Named

▶ Parses classes in bytecode, without loading them

```java
private void scan(String... basePackages) {
    for (String basePackage : basePackages) {
        Set<BeanDefinition> candidates = findCandidateComponents(basePackage);

        for (BeanDefinition candidate : candidates) {
            // augment BeanDefinition with metadata from component class

            registerBeanDefinition(candidate, beanDefinitionRegistry);
        }
    }
}
```

BeanPostProcessor

```java
public interface BeanPostProcessor {
    Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException;
    Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException;
}
```

▶ Some implementations

- ▶ ConfigurationPropertiesBindingPostProcessor (@ConfigurationProperties)

- ▶ AutowiredAnnotationBeanPostProcessor (@Autowired, @Value, @Inject, @Lookup)

- ▶ MethodValidationPostProcessor (@Valid, @NotNull, @NotEmpty)

Proxies

- ▶ JDK Proxy
  - ▶ Interfaces only
  - ▶ Part of JDK
- ▶ CGLIB
  - ▶ Classes and interfaces

```java
Person person = new Person();
person.setName("Mikaël");


Person personProxy = (Person) Enhancer.create(
        Person.class,
        (MethodInterceptor) (_, method, args, _) -> {

                if (method.getReturnType().equals(String.class)) {
                    String result = (String) method.invoke(person, args);
                    return result.toUpperCase();
                }


                return method.invoke(person, args);
        });


System.out.println(personProxy.getName()); // MIKAËL
```

Demo!

- Inspecting proxies

Conclusion

- We saw:

  - Bean/BeanDefinition

  - BeanFactory

  - ApplicationContext

  - ClassPathBeanDefinitionScanner

  - BeanPostProcessor

  - Proxying (JDK and CGLIB)