Chapter 5. 스프링 컨테이너와 스프링 빈

BeanFactory와 ApplicationContext

applicationContext = BeanFactory + a

- BeanFactory
  - 스프링 컨테이너의 최상위 인터페이스
  - 스프링 빈을 관리하고 조회하는 역할 담당 (getBean 이런거)
  - 지금까지 사용한 대부분 기능은 BeanFactory가 제공하는 기능
- ApplicationContext
  - beanFactory의 기능을 모두 상속받아서 사용
  - public ~~ extends BeanFactory { }
  - 빈을 관리하고 검색하는 기능을 BeanFactory가 제공해주는데,, 이 둘의 차이는?
  - 애플리케이션을 개발할 때에는 빈을 관리하고 조회하는 기능 뿐만 아니라 **수많은 부가 기능이 필요**
  - public interface ApplicationContext extends EnviromentCapable, ListableBeanFactory, HierarchicalBea, MessageSource, ApplicationEventPublisher, ResourcePatternResolver {}
  - **“메시지소스를 활용한 국제화 기능"** : _ko, _en …
  - “**환경변수"** : 로컬, 개발, 운영 등을 구분해서 처리
  - **“애플리케이션 이벤트"** : 이벤트를 발행하고 구독하는 모델을 편리하게 지원
  - **“편리한 리소스 조회"** : 파일, 클래스패스, 외부 등에서 리소스를 편리하게 조회
- 정리

  ApplicationContext는 BeanFactory의 기능을 상속 받는다.

  ApplicationContext는 빈 관리 기능 + 부가 기능 제공

  BeanFactory를 직접 사용할 일은 거의 없음. ApplicationContext 만 거의 사용한다고 보면 됨

  이 둘을 모두 <스프링 컨테이너> 라고 한다.

다양한 설정 형식 지원 - 자바 코드, XML

- 스프링 컨테이너가 받아올 수 있는 설정 정보의 다양한 형식
  - 자바코드, XML, Groovy 등
- ApplicationContext를 구성방식
  - AnnotationConfig ApplicationContext

    → AppConfig.class를 사용함
  - GenericXml ApplicationContext

    → XML 문서를 설정정모로 사용함 (ex_appConfig.xml)
  - 애노테이션 기반 자바 코드 설정

    → new AnnotationConfigApplication(AppConfig.class)

```java
package hello.core.beanfind;

import ...

class ApplicationContextBasicFindTest {
    AnnotationConfigApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);

    @Test
    @DisplayName("빈 이름으로 조회")
    void findBeanByName() {
        MemberService memberService = ac.getBean( name: "memberService", MemberService.class);
        Assertions.assertThat(memberService).isInstanceOf(MemberServiceImpl.class);
    }
    @Test
```

- XML 설정 사용
  - 컴파일 없이 빈 설정 정보를 변경할 수 있는 장점도 있다.
  - 기존 AppConfig.java 와의 비교

```java
@Configuration
public class AppConfig {
    // 메소드를 가지고 호출하는 순간 역할과 구현 클래스를 알 수 있음
    // 메모리 -> DB로 바뀐다면 memberRepository만 고치면 됨
    // 할인 정책이 바뀐다면 discountPolicy만 고치면 됨
    @Bean   // spring 컨테이너에 등록
    public MemberService memberService() { return new MemberServiceImpl(memberRepository()); }

    @Bean
    public MemoryMemberRepository memberRepository() { return new MemoryMemberRepository(); }

    @Bean
    public OrderService orderService() { return new OrderServiceImpl(memberRepository(), discountPolicy()); }

    @Bean
    public DiscountPolicy discountPolicy() { return new RateDiscountPolicy(); }
}
```

AppConfig.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="memberService" class="hello.core.member.MemberServiceImpl">
        <constructor-arg name="memberRepository" ref="memberRepository" />
    </bean>

    <bean id="memberRepository" class="hello.core.member.MemoryMemberRepository"/>

    <bean id="orderService" class="hello.core.order.OrderServiceImpl">
        <constructor-arg name="memberRepository" ref="memberRepository"/>
        <constructor-arg name="discountPolicy" ref="discountPolicy" />
    </bean>
    <bean id="discountPolicy" class="hello.core.discount.RateDiscountPolicy"/>
</beans>
```

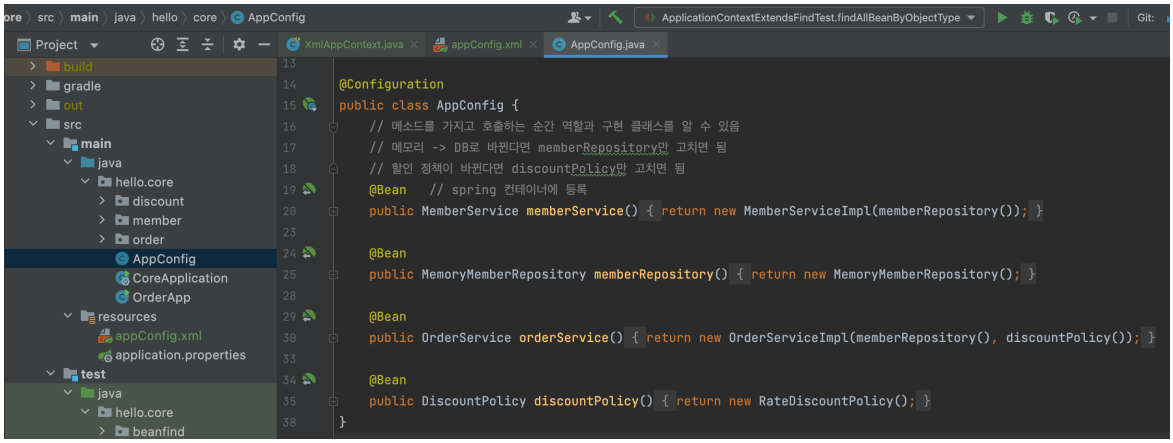
appConfig.xml

스프링 빈 설정 메타 정보 - BeanDefinition

- 스프링은 어떻게 위와 같이 다양한 설정 형식을 지원할 수 있을까?

  → BeanDefinition 이라는 추상화때문에 가능
- 역할과 구현을 개념적으로 나눈 것
  - 자바 코드를 읽어서 BeanDefinition을 만든다
  - XML을 읽어서 BeanDefinition을 만든다
  - 스프링 컨테이너는 자바 코드인지, XML 인지 몰라도 된다.

    BeanDefinition만 알면 됨
- BeanDefinition : 빈 설정 메타 정보
- @Bean, <bean> 당 각각 하나씩 메타 정보가 생성됨
- 스프링 컨테이너는 이 메타정보를 기반으로 스프링 빈을 생성함



AppConfig.java를 설정 정보를 읽듯이 읽는다.

- BeanDefinition 정보 → 요론 정보들이 있다 정도만 이해
  - getBeanDefinitionName : 빈 설정 메타정보 확인

```
00:45:50.508 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
00:45:50.509 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
beanDefinitionName = appConfigbean Definition = Generic bean: class [hello.core.AppConfig$$EnhancerBySpringCGLIB$$2fd30374]; s
beanDefinitionName = memberServicebean Definition = Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowireMo
beanDefinitionName = memberRepositorybean Definition = Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowi
beanDefinitionName = orderServicebean Definition = Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowireMo
beanDefinitionName = discountPolicybean Definition = Root bean: class [null]; scope=; abstract=false; lazyInit=null; autowireM

Process finished with exit code 0
```

  - BeanClassName
  - factoryBeanName
  - factoryMethodName
  - Scope
  - lazInit
  - InitMethodName

    - DestroyMethodName
    - Constructor arguments, Properties

- 정리
    - BeanDefinition을 직접 생성해서 스프링 컨테이너에 등록할 수도 있다. (실무는 x)
    - 요점은 스프링이 다양한 형태의 설정 정보를 BeanDefinition으로 추상화 해서 사용한다 !!

- 참고
    - ApplicationContext 로 한다면, getBeanDefinition 을 사용 못 함
    - xml config → java config로 바뀌면서 class 정보가 직접적으로 드러나지 않음
    - **java config를 통해서 bean 정보를 등록하는 것을 factoryBean을 통해서 등록한다고 표현함**