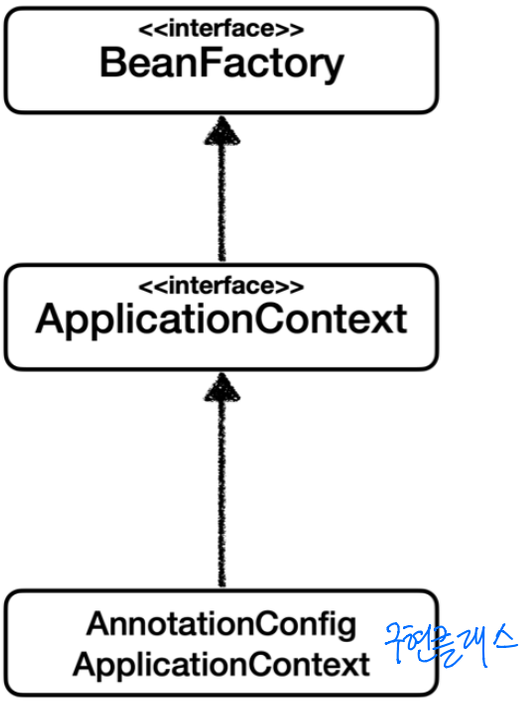
```
        public DiscountPolicy fixDiscountPolicy() {
            return new FixDiscountPolicy();
        }
    }

}
```

BeanFactory와 ApplicationContext

beanFactory와 ApplicationContext에 대해서 알아보자.



BeanFactory

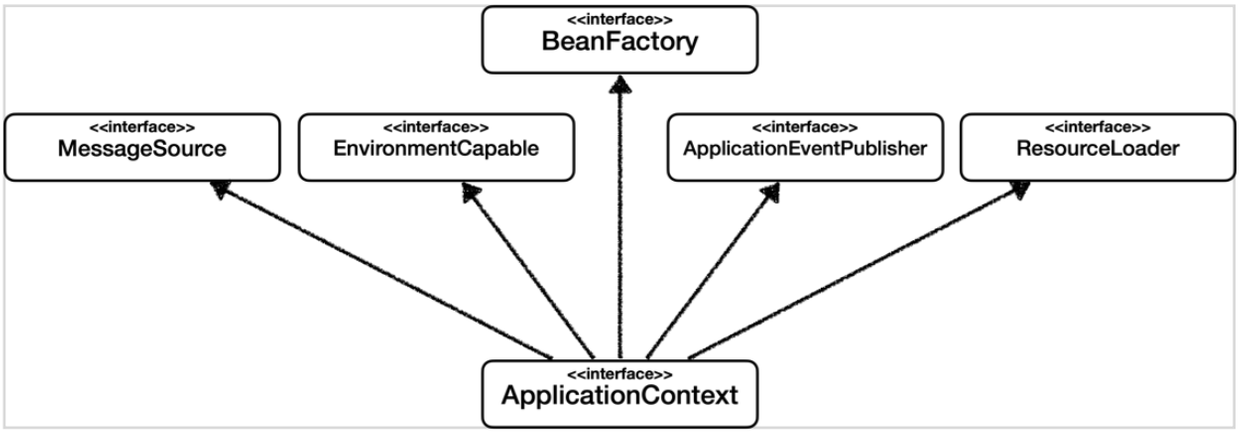
- 스프링 컨테이너의 최상위 인터페이스다.
- 스프링 빈을 관리하고 조회하는 역할을 담당한다.
- `getBean()`을 제공한다.
- 지금까지 우리가 사용했던 대부분의 기능은 BeanFactory가 제공하는 기능이다.

ApplicationContext

- BeanFactory 기능을 모두 상속받아서 제공한다.
- 빈을 관리하고 검색하는 기능을 BeanFactory가 제공해주는데, 그러면 둘의 차이가 뭘까?
- 애플리케이션을 개발할 때는 빈은 관리하고 조회하는 기능은 물론이고, 수 많은 부가기능이 필요하다.

BeanFactory 뿐만아니라 여러가지 인터페이스를 받고있다

ApplicatonContext가 제공하는 부가기능



- **메시지소스를 활용한 국제화 기능**
  - 예를 들어서 한국에서 들어오면 한국어로, 영어권에서 들어오면 영어로 출력
- **환경변수**
  - 로컬, 개발, 운영등을 구분해서 처리
- **애플리케이션 이벤트**
  - 이벤트를 발행하고 구독하는 모델을 편리하게 지원
- **편리한 리소스 조회**
  - 파일, 클래스패스, 외부 등에서 리소스를 편리하게 조회

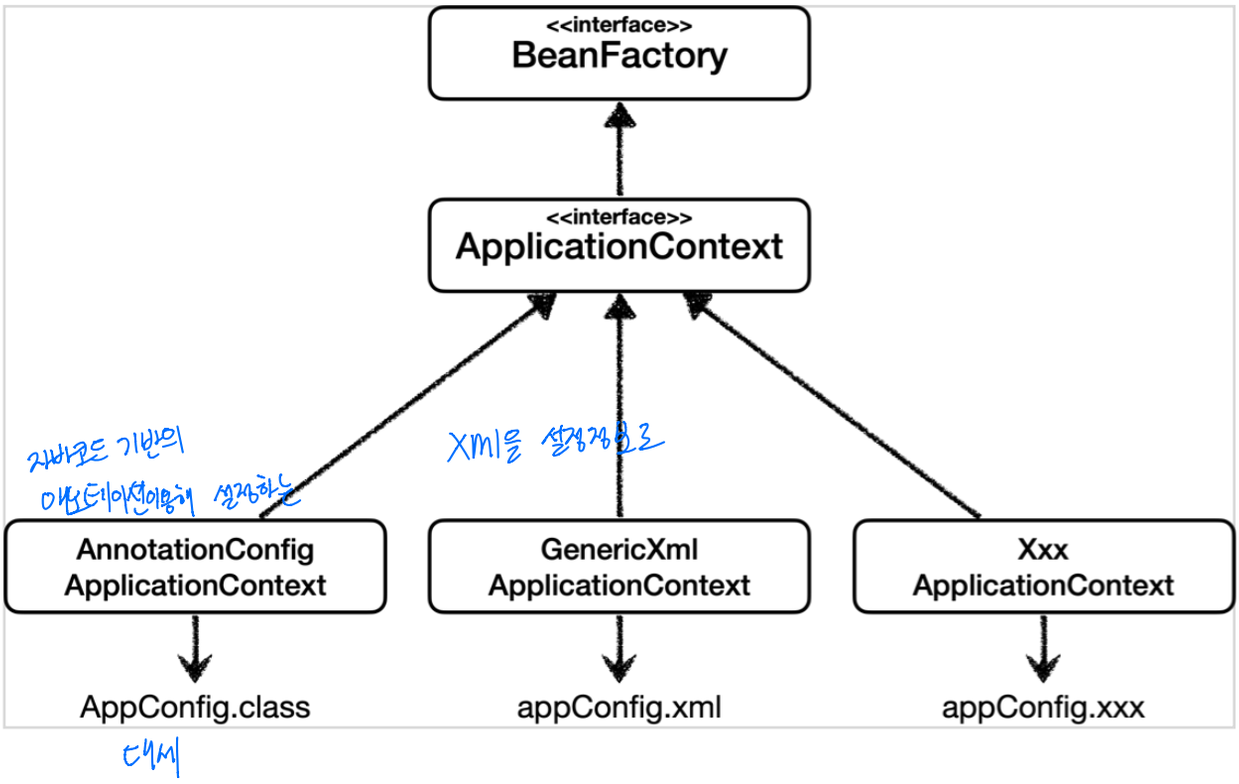
정리

- ApplicationContext는 BeanFactory의 기능을 상속받는다.
- ApplicationContext는 빈 관리기능 + 편리한 부가 기능을 제공한다.
- BeanFactory를 직접 사용할 일은 거의 없다. 부가기능이 포함된 ApplicationContext를 사용한다.
- BeanFactory나 ApplicationContext를 스프링 컨테이너라 한다.

다양한 설정 형식 지원 - 자바 코드, XML

- 스프링 컨테이너는 다양한 형식의 설정 정보를 받아드릴 수 있게 유연하게 설계되어 있다.

- 자바 코드, XML, Groovy 등등



애노테이션 기반 자바 코드 설정 사용

- 지금까지 했던 것이다.
- `new AnnotationConfigApplicationContext(AppConfig.class)`
- `AnnotationConfigApplicationContext` 클래스를 사용하면서 자바 코드로된 설정 정보를 넘기면 된다.

XML 설정 사용

- 최근에는 스프링 부트를 많이 사용하면서 XML기반의 설정은 잘 사용하지 않는다. 아직 많은 레거시 프로젝트 들이 XML로 되어 있고, 또 XML을 사용하면 컴파일 없이 빈 설정 정보를 변경할 수 있는 장점도 있으므로 한번쯤 배워두는 것도 괜찮다.
- `GenericXmlApplicationContext` 를 사용하면서 `xml` 설정 파일을 넘기면 된다.

XmlAppConfig 사용 자바 코드

```
package hello.core.xml;


import hello.core.member.MemberService;
import org.junit.jupiter.api.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;
```

```
import static org.assertj.core.api.Assertions.*;

public class XmlAppContext {

    @Test
    void xmlAppContext() {
        ApplicationContext ac = new
GenericXmlApplicationContext("appConfig.xml");

        MemberService memberService = ac.getBean("memberService",
MemberService.class);
        assertThat(memberService).isInstanceOf(MemberService.class);
    }
}
```

xml 기반의 스프링 빈 설정 정보 재귀기 이번은 resources

`src/main/resources/appConfig.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" />
```

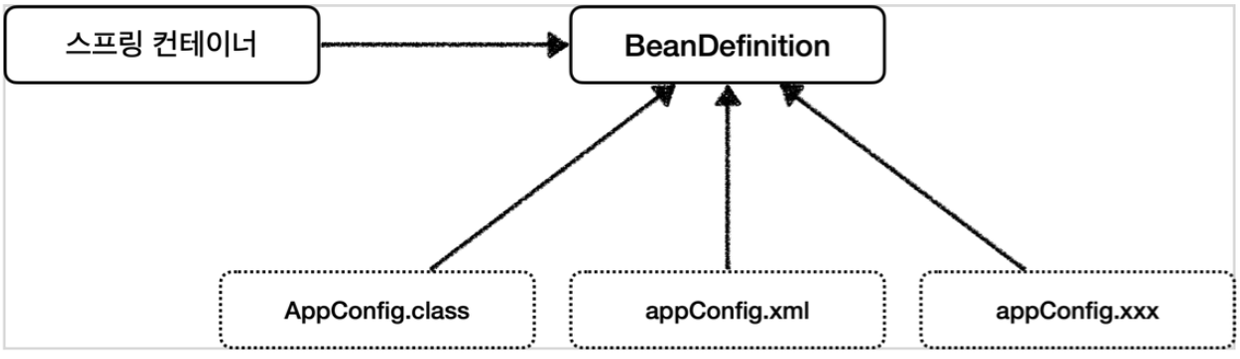
```
</beans>
```

- xml 기반의 `appConfig.xml` 스프링 설정 정보와 자바 코드로 된 `AppConfig.java` 설정 정보를 비교해보면 거의 비슷하다는 것을 알 수 있다.
- xml 기반으로 설정하는 것은 최근에 잘 사용하지 않으므로 이정도로 마무리 하고, 필요하면 스프링 공식 레퍼런스 문서를 확인하자.
  - https://spring.io/projects/spring-framework

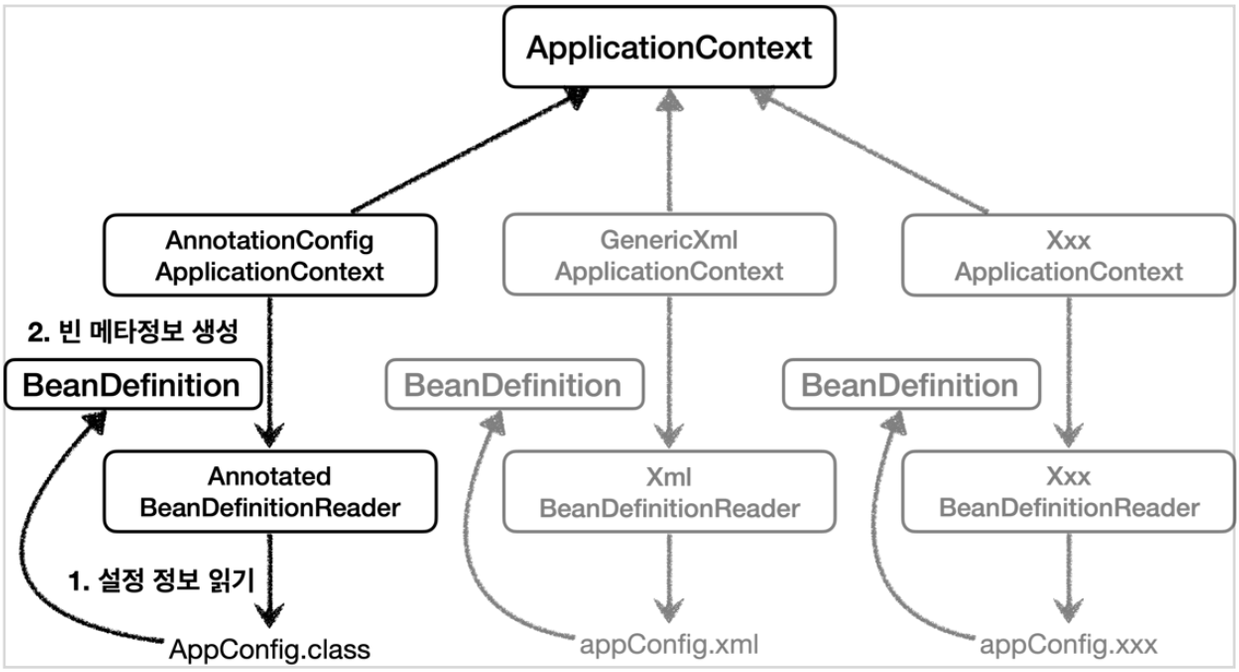
스프링 빈 설정 메타 정보 - BeanDefinition

빈정보를 추상화 시킴!

- 스프링은 어떻게 이런 다양한 설정 형식을 지원하는 것일까? 그 중심에는 `BeanDefinition` 이라는 추상화가 있다.
- 쉽게 이야기해서 **역할과 구현을 개념적으로 나눈 것**이다!
  - XML을 읽어서 BeanDefinition을 만들면 된다.
  - 자바 코드를 읽어서 BeanDefinition을 만들면 된다.
  - 스프링 컨테이너는 자바 코드인지, XML인지 몰라도 된다. 오직 BeanDefinition만 알면 된다.
- `BeanDefinition` 을 빈 설정 메타정보라 한다.
  - `@Bean` , `<bean>` 당 각각 하나씩 메타 정보가 생성된다.
- 스프링 컨테이너는 이 메타정보를 기반으로 스프링 빈을 생성한다.



코드 레벨로 조금 더 깊이 있게 들어가보자.



- `AnnotationConfigApplicationContext`는 `AnnotatedBeanDefinitionReader`를 사용해서 `AppConfig.class`를 읽고 `BeanDefinition`을 생성한다.
- `GenericXmlApplicationContext`는 `XmlBeanDefinitionReader`를 사용해서 `appConfig.xml` 설정 정보를 읽고 `BeanDefinition`을 생성한다.
- 새로운 형식의 설정 정보가 추가되면, XxxBeanDefinitionReader를 만들어서 `BeanDefinition`을 생성하면 된다.

BeanDefinition 살펴보기

BeanDefinition 정보

- BeanClassName: 생성할 빈의 클래스 명(자바 설정 처럼 팩토리 역할의 빈을 사용하면 없음)
- factoryBeanName: 팩토리 역할의 빈을 사용할 경우 이름, 예) appConfig
- factoryMethodName: 빈을 생성할 팩토리 메서드 지정, 예) memberService
- Scope: 싱글톤(기본값)
- lazyInit: 스프링 컨테이너를 생성할 때 빈을 생성하는 것이 아니라, 실제 빈을 사용할 때 까지 최대한 생성을 지연처리 하는지 여부
- InitMethodName: 빈을 생성하고, 의존관계를 적용한 뒤에 호출되는 초기화 메서드 명
- DestroyMethodName: 빈의 생명주기가 끝나서 제거하기 직전에 호출되는 메서드 명
- Constructor arguments, Properties: 의존관계 주입에서 사용한다. (자바 설정 처럼 팩토리 역할의 빈을 사용하면 없음)

```
package hello.core.beandefinition;
```

```java
import hello.core.AppConfig;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.MutablePropertyValues;
import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.beans.factory.config.ConstructorArgumentValues;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class BeanDefinitionTest {

    AnnotationConfigApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
//    GenericXmlApplicationContext ac = new
GenericXmlApplicationContext("appConfig.xml");
    @Test
    @DisplayName("빈 설정 메타정보 확인")
    void findApplicationBean() {
        String[] beanDefinitionNames = ac.getBeanDefinitionNames();
        for (String beanDefinitionName : beanDefinitionNames) {
            BeanDefinition beanDefinition =
ac.getBeanDefinition(beanDefinitionName);

            if (beanDefinition.getRole() == BeanDefinition.ROLE_APPLICATION) {
                System.out.println("beanDefinitionName" + beanDefinitionName +
                                   " beanDefinition = " + beanDefinition);
            }
        }
    }

}
```

정리

- BeanDefinition을 직접 생성해서 스프링 컨테이너에 등록할 수 도 있다. 하지만 실무에서 BeanDefinition을 직접 정의하거나 사용할 일은 거의 없다. ➜ 어려우면 그냥 넘어가면 된다^^!