스프링 컨테이너

- `ApplicationContext`를 스프링 컨테이너라 한다.
- 기존에는 개발자가 `AppConfig`를 사용해서 직접 객체를 생성하고 DI를 했지만, 이제부터는 스프링 컨테이너를 통해서 사용한다.
- 스프링 컨테이너는 `@Configuration`이 붙은 `AppConfig`를 설정(구성) 정보로 사용한다. 여기서 `@Bean`이라 적힌 메서드를 모두 호출해서 반환된 객체를 스프링 컨테이너에 등록한다. 이렇게 스프링 컨테이너에 등록된 객체를 스프링 빈이라 한다.
- 스프링 빈은 `@Bean`이 붙은 메서드의 명을 스프링 빈의 이름으로 사용한다. (`memberService`, `orderService`)
- 이전에는 개발자가 필요한 객체를 `AppConfig`를 사용해서 직접 조회했지만, 이제부터는 스프링 컨테이너를 통해서 필요한 스프링 빈(객체)를 찾아야 한다. 스프링 빈은 `applicationContext.getBean()` 메서드를 사용해서 찾을 수 있다.
- 기존에는 개발자가 직접 자바코드로 모든 것을 했다면 이제부터는 스프링 컨테이너에 객체를 스프링 빈으로 등록하고, 스프링 컨테이너에서 스프링 빈을 찾아서 사용하도록 변경되었다.

스프링 컨테이너 생성

스프링 컨테이너가 생성되는 과정을 알아보자.

```
//스프링 컨테이너 생성
ApplicationContext applicationContext =
                        new
AnnotationConfigApplicationContext(AppConfig.class);
```

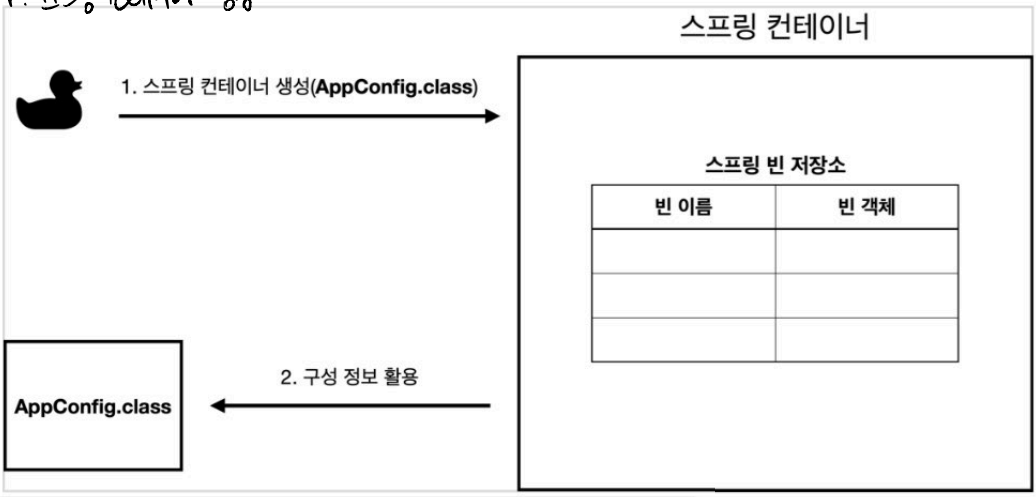
- `ApplicationContext`를 스프링 컨테이너라 한다.
- `ApplicationContext`는 인터페이스이다.
- 스프링 컨테이너는 XML을 기반으로 만들 수 있고, 애노테이션 기반의 자바 설정 클래스로 만들 수 있다. (거의 사용 X)
- 직전에 `AppConfig`를 사용했던 방식이 애노테이션 기반의 자바 설정 클래스로 스프링 컨테이너를 만든 것이다.
- 자바 설정 클래스를 기반으로 스프링 컨테이너(`ApplicationContext`)를 만들어보자.
  - `new AnnotationConfigApplicationContext(AppConfig.class);`
  - 이 클래스는 `ApplicationContext` 인터페이스의 구현체이다.

```java
public class OrderApp {

    public static void main(String[] args) {
//        AppConfig appConfig = new AppConfig();
//        MemberService memberService = appConfig.memberService();
//        OrderService orderService = appConfig.orderService();

        ApplicationContext applicationContext = new
AnnotationConfigApplicationContext(AppConfig.class);
        MemberService memberService =
applicationContext.getBean("memberService", MemberService.class);
        OrderService orderService = applicationContext.getBean("orderService",
OrderService.class);

        long memberId = 1L;
        Member member = new Member(memberId, "memberA", Grade.VIP);
        memberService.join(member);
```

<AppConfig를 사용해 직접 객체를 생성하고 DI 하는 코드>

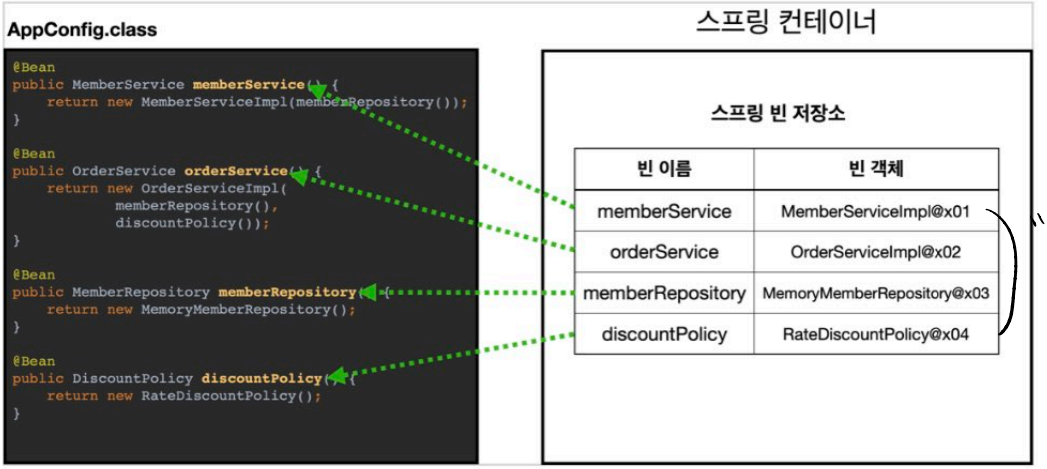
// @Configuration 이 붙은 AppConfig.class 넘겨줌

1. 스프링 컨테이너 생성



- `new AnnotationConfigApplicationContext(AppConfig.class)` 스프링컨테이너 생성
- 스프링 컨테이너를 생성할 때는 구성 정보를 지정해주어야 한다.
- 여기서는 `AppConfig.class` 를 구성 정보로 지정했다.

2. 스프링 빈 등록

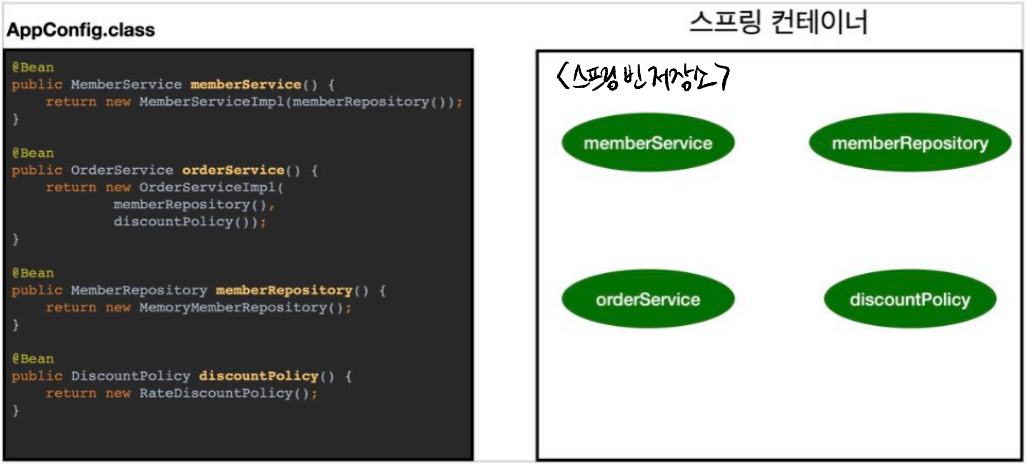


- 스프링 컨테이너는 파라미터로 넘어온 설정 클래스 정보를 사용해서 스프링 빈을 등록한다.

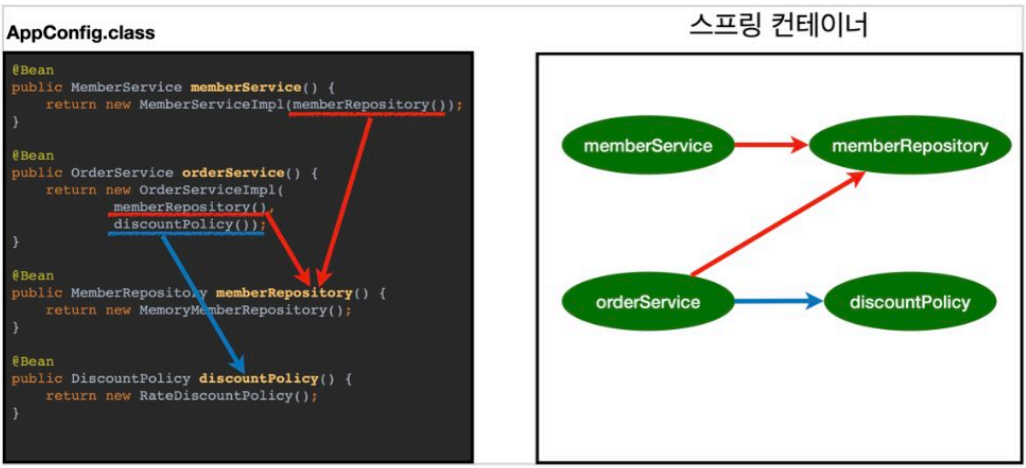
빈 이름

- 빈 이름은 메서드 이름을 사용한다.
- 빈 이름을 직접 부여할 수 도 있다.
  - `@Bean(name="memberService2")` //빈이름은 겹복되면 X

3. 스프링 빈 의존관계 설정 - 준비



4. 스프링 빈 의존관계 설정 - 완료

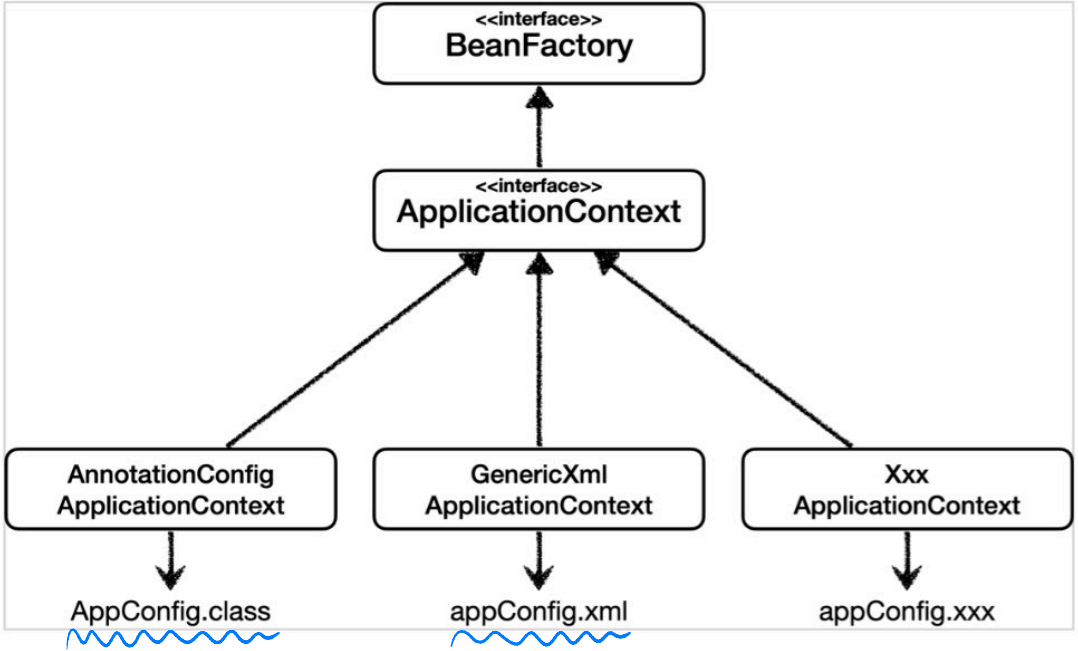


- 스프링 컨테이너는 설정 정보를 참고해서 의존관계를 주입(DI)한다.
- 단순히 자바 코드를 호출하는 것 같지만, 차이가 있다. 이 차이는 뒤에 싱글톤 컨테이너에서 설명한다.

다양한 설정 형식 지원 - 자바 코드, XML → 요즘사용 X

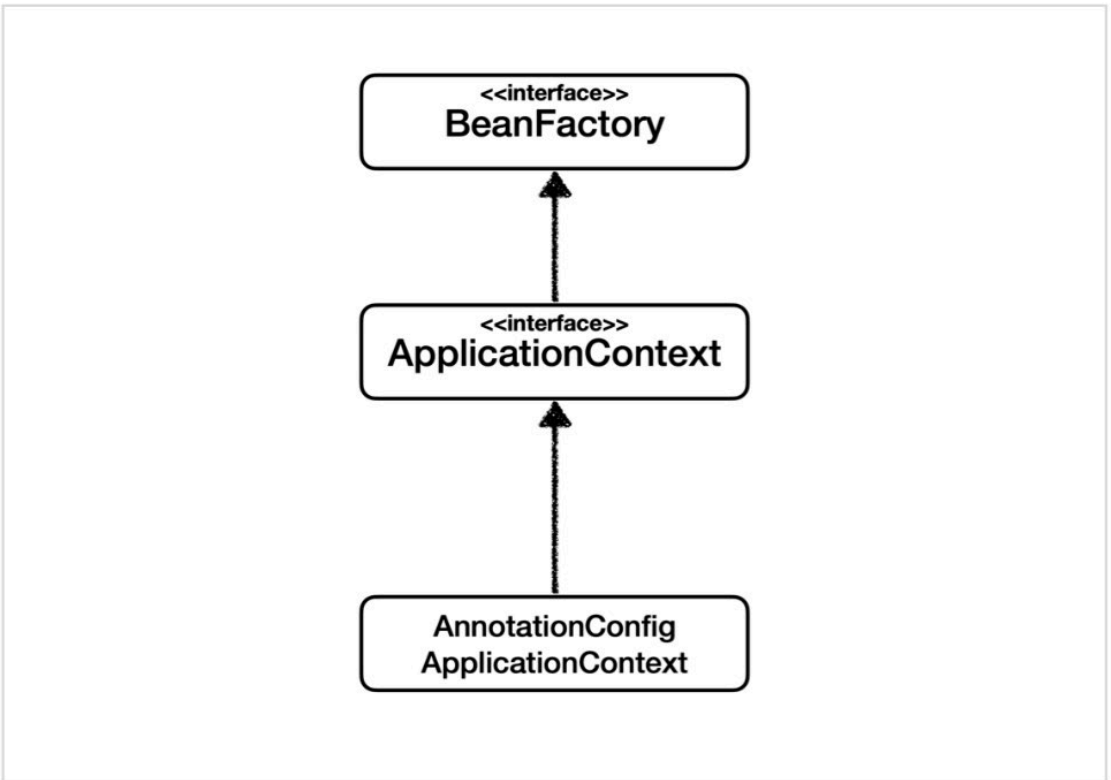
- 스프링 컨테이너는 다양한 형식의 설정 정보를 받아드릴 수 있게 유연하게 설계되어 있다.

- 자바 코드, XML, Groovy 등등



BeanFactory와 ApplicationContext

beanFactory와 ApplicationContext에 대해서 알아보자.



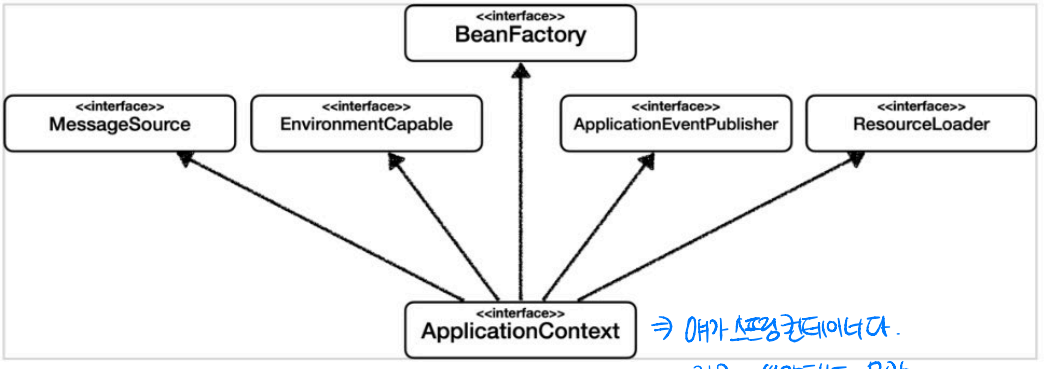
BeanFactory

- 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.

ApplicatonContext가 제공하는 부가기능



⇒ 얘가 스프링컨테이너다.
라고 생각해도 무관

- **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.

스프링 빈 설정 메타 정보 - BeanDefinition

- 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.