Spring week 4

유채린
chaerin.du.ub@gmail.com

해당 내용은 인프런 강의를 정리한 내용입니다.
잘못된 내용에 대하여 피드백주시면 감사하겠습니다!

(인프런 아이콘 클릭 시, 참고한 인프런 강의 사이트로 이동합니다.)

Index

- AppConfig
- IoC, DI, 컨테이너
- 스프링 컨테이너
- BeanFactory
- BeanDefinition

AppConfig

역할과 구현을 명확하게 분리

OCP Open/Closed Principle

개방-폐쇄 원칙

- 확장에는 열려 있지만 변경에는 닫혀 있어야 한다.
- **다형성** (Polymorphism) 활용해야 한다.

DIP dependency Inversion Principle

의존관계 역전 원칙

- 추상적인 것은 자신보다 구체적인 것에 의존하지 않고, 변화하기 쉬운 것에 의존해서는 안된다.
- 구현 클래스에 의존하지 말고, **인터페이스(추상화)에 의존하자**

AppConfig

- 관심사를 분리하자! (객체 생성, 연결 – 객체 사용)
- 애플리케이션의 전체 동작 방식을 구성(설정)하기 위해 **구현 객체를 생성하고, 연결하는 책임**을 가지는 별도의 설정 클래스

AppConfig 에서 생성한 객체 인스턴스의 참조 값을 생성자를 통해서 주입

→ 의존관계주입 (Dependency Injection)

- 클라이언트는 인터페이스에만 의존하면 되므로 DIP 만족
- 실행 코드가 바뀌는 경우 AppConfig에서만 수정하면 되므로 OCP 만족!

IoC, DI, 컨테이너

제어의 역전 IoC (Inversion of Control)

- AppConfig를 사용함으로써 프로그램의 제어 흐름에 대한 권한은 모두 AppConfig가 가져간다.

  서비스에서는 필요한 인터페이스들을 호출하지만 어떤 구현 객체들이 실행되는지 모른다.

- 프로그램의 제어 흐름을 직접 제어하는 것이 아니라 외부에서 관리하는 것!

의존관계 주입 DI (Dependency Injection)

- 애플리케이션 **실행 시점(런타임)**에 외부에서 실제 구현 객체를 생성하고 클라이언트에 전달해서 클라이언트와 서버의 실제 의존관계가 연결 되는 것

- 의존관계 주입을 사용하면 정적인 클래스 의존관계를 변경하지 않고, 동적인 객체 인스턴스 의존관계를 쉽게 변경할 수 있다.
  → 클래스 다이어그램, 애플리케이션 코드를 건드리지 않고 객체 다이어그램, AppConfig만 수정!

- 정적인 클래스 의존관계
  import코드만 보고 판단할 수 있는 의존관계
- 동적인 객체 인스턴스 의존관계
  애플리케이션 실행 시점에서 실제 생성된 객체 인스턴스의 참조가 연결된 의존관계

컨테이너

- 객체 생성, 관리, 의존관계 연결 (ex. AppConfig)

- IoC 컨테이너, DI 컨테이너, 어샘블러, 오브젝트 팩토리 등 불리기도 하지만 최근에는 의존관계 주입에 초점을 맞추어 주로 **DI 컨테이너**라고 불린다.

스프링 컨테이너

스프링 컨테이너

- AppConfig를 사용해서 직접 객체 생성하고 의존관계 주입하는 게 아닌, ApplicationContext(스프링 컨테이너)를 사용하여 스프링 적용한다.
- 스프링 컨테이너는 @Configuration이 붙은 클래스를 설정(구성) 정보로 사용한다.
- @Configuration에서 @Bean이라 적힌 메서드를 모두 호출해서 반환된 객체를 스프링 컨테이너에 등록하는데, 이 등록된 객체를 스프링 빈 이라고 한다.
- 스프링 컨테이너를 통해서 스프링 빈(객체)은 applicationContext.getBean() 메서드를 사용해서 찾을 수 있다.
- ApplicationContext는 인터페이스이다. (다형성 적용됨)
- XML기반, 애노테이션 기반의 자바 설정 클래스로 만들 수 있다. (요즘 애노테이션!, XML은 legacy)
- 빈 이름은 항상 다른 이름으로 부여할 것

스프링 적용

스프링 기반으로 변경 → Annotation 기반의 자바 설정 클래스

```java
@Configuration
public class AppConfig {

    @Bean
    public MemberService memberService() { return new MemberServiceImpl(memberRepository()); }

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

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

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

스프링 컨테이너 적용

```java
public class MemberApp {

    public static void main(String[] args) {

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

        Member member = new Member( id: 1L,  name: "memberA", Grade.VIP);
        memberService.join(member);
```

ApplicationContext applicationContext
= new AnnotationConfigApplicationContext(AppConfig.class);

→ AppConfig.class에 있는걸 스프링 빈으로 등록하여 스프링 컨테이너가 관리해줌

MemberService memberService
= applicationContext.getBean("memberService", MemberService.class);

→ getBean(빈 이름(default: 메소드 이름), 타입(return 값))

BeanFactory

Feat. ApplicationContext

BeanFactory

- BeanFactory는 스프링 컨테이너의 최상위 인터페이스
- 스프링 빈을 관리하고 조회하는 역할, 대부분의 기능 BeanFactory가 제공하는 기능 사용

- 왜 굳이 BeanFactory 기능을 모두 상속받는 ApplicationContext를 쓸까요?

```
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory,
        MessageSource, ApplicationEventPublisher, ResourcePatternResolver {
```

Application은 다른 인터페이스도 상속받고 있어 BeanFactory보다 더 많은 부가기능을 제공한다.

- MessageSource: 한국에서 접속 시 한국어로, 영어권에서 접속 시 영어로 출력
- EnvrionmentCapable: 로컬서버, 개발서버, 운영서버 등 구분해서 처리
- ApplicationEventPublisher: 이벤트를 발행하고 구독하는 모델 편리하게 지원
- ResourcePatternResolver: ResourceLoader의 확장된 인터페이스로, 외부에서 파일을 읽어와 쓸 때 추상화해서 편리하게 제공해준다.

```
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory,
        MessageSource, ApplicationEventPublisher, ResourcePatternResolver {
```

- ListableBeanFactory: BeanFactory의 확장된 인터페이스
- HierarchicalBeanFactory: 계층구조의 일부가 될 수 있는 bean factory들에 의해 실행되는 서브 인터페이스

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/ListableBeanFactory.html

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/HierarchicalBeanFactory.html

BeanDefinition

스프링 빈 설정 메타 정보

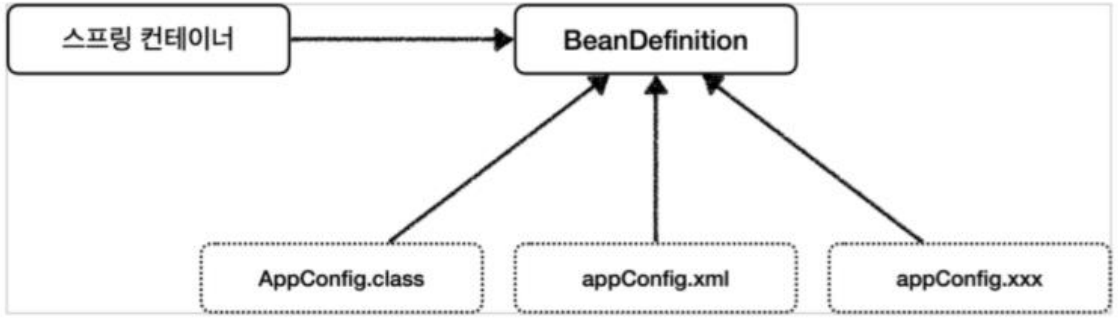
BeanDefinition

- BeanDefinition을 직접 생성해서 스프링 컨테이너에 등록할 수 있다.

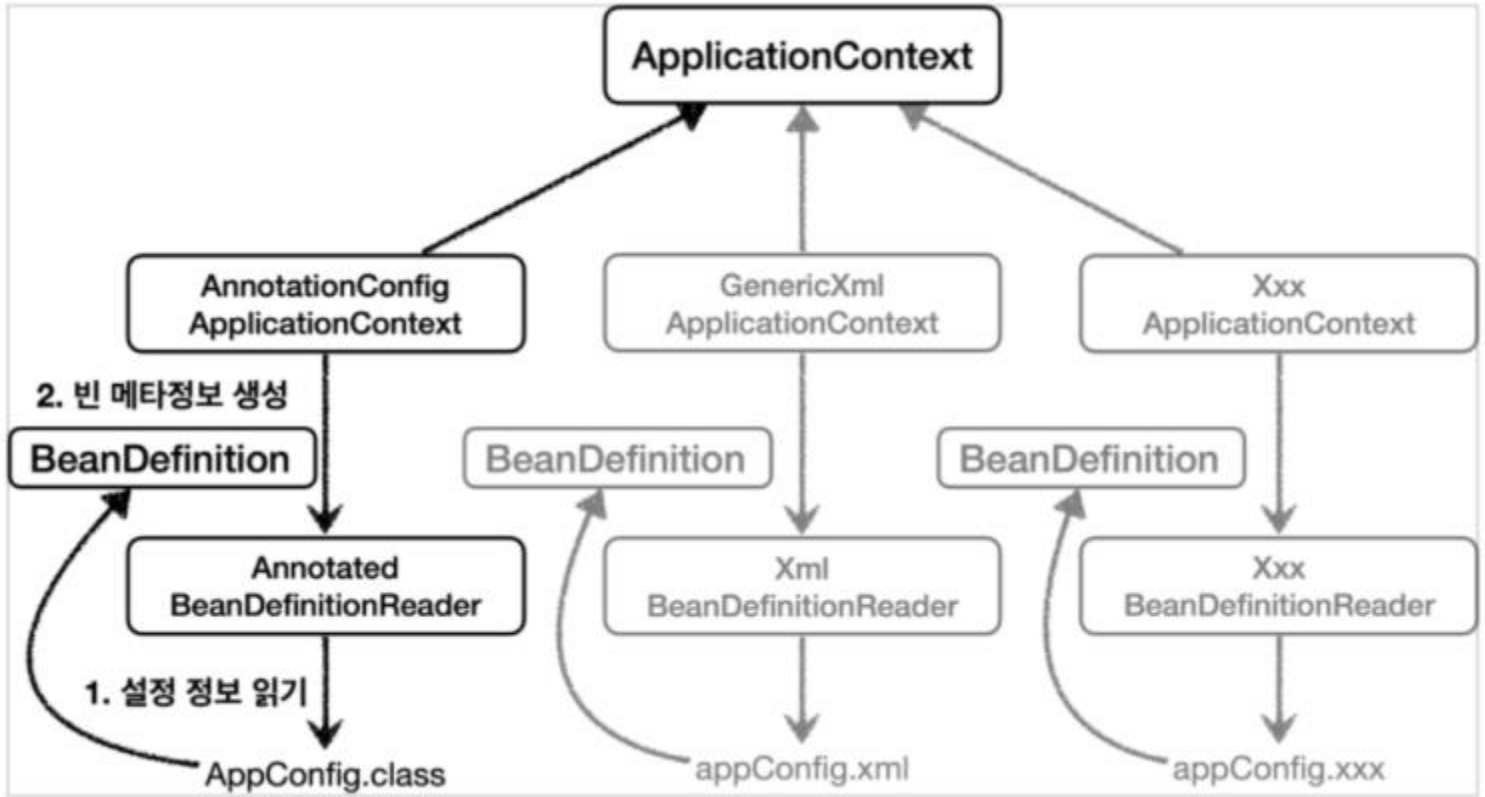
- 스프링 컨테이너는 오직 BeanDefinition만 알면 된다.

  (BeanDefinition 기반으로 스프링 빈 생성)

  → 추상화에만 의존!



- Bean 설정 메타 정보
  - @Bean(annotation), <bea>(xml)당 각각 하나씩 메타정보가 생성된다.


ApplicationContext
AnnotationConfig
ApplicationContext
GenericXml
ApplicationContext
Xxx
ApplicationContext
2. 빈 메타정보 생성
BeanDefinition
BeanDefinition
BeanDefinition
Annotated
BeanDefinitionReader
Xml
BeanDefinitionReader
Xxx
BeanDefinitionReader
1. 설정 정보 읽기
AppConfig.class
appConfig.xml
appConfig.xxx

감사합니다.