Bean 탐구하기

스터디 2회차 발표 - ES

들어가기 전

```java
@Test
@DisplayName("애플리케이션 빈 출력하기")
void findAllBean(){
    String[] beanDefinitionNames = ac.getBeanDefinitionNames();
    for (String beanDefinitionName : beanDefinitionNames) {
        int role = ac.getBeanDefinition(beanDefinitionName).getRole();
        BeanDefinition beanDefinition = ac.getBeanDefinition(beanDefinitionName);
        if(beanDefinition.getRole() == beanDefinition.ROLE_APPLICATION){
            Object bean = ac.getBean(beanDefinitionName);
            System.out.println("role = " + role + " name = " + beanDefinitionName + " bean = " + bean);
        }
    }
}
```

```
Tests passed: 1 of 1 test – 26 ms
23:58:10.555 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.576 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.578 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.578 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.580 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.587 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
role = 0 name = appConfig bean = hello.core.order.AppConfig@6f7923a5
```

- 애플리케이션 빈 출력하기
  - 스프링이 내부에서 사용하는 빈은 제외하고, 내가 등록한 빈만 출력해보자.
  - 스프링이 내부에서 사용하는 빈은 `getRole()` 로 구분할 수 있다.
    - `ROLE_APPLICATION` : 일반적으로 사용자가 정의한 빈
    - `ROLE_INFRASTRUCTURE` : 스프링이 내부에서 사용하는 빈

빈의 종류

애플리케이션 로직 빈

- 애플리케이션 로직을 담고 있는 주요 클래스의 오브젝트 (DAO, Service, Controller)

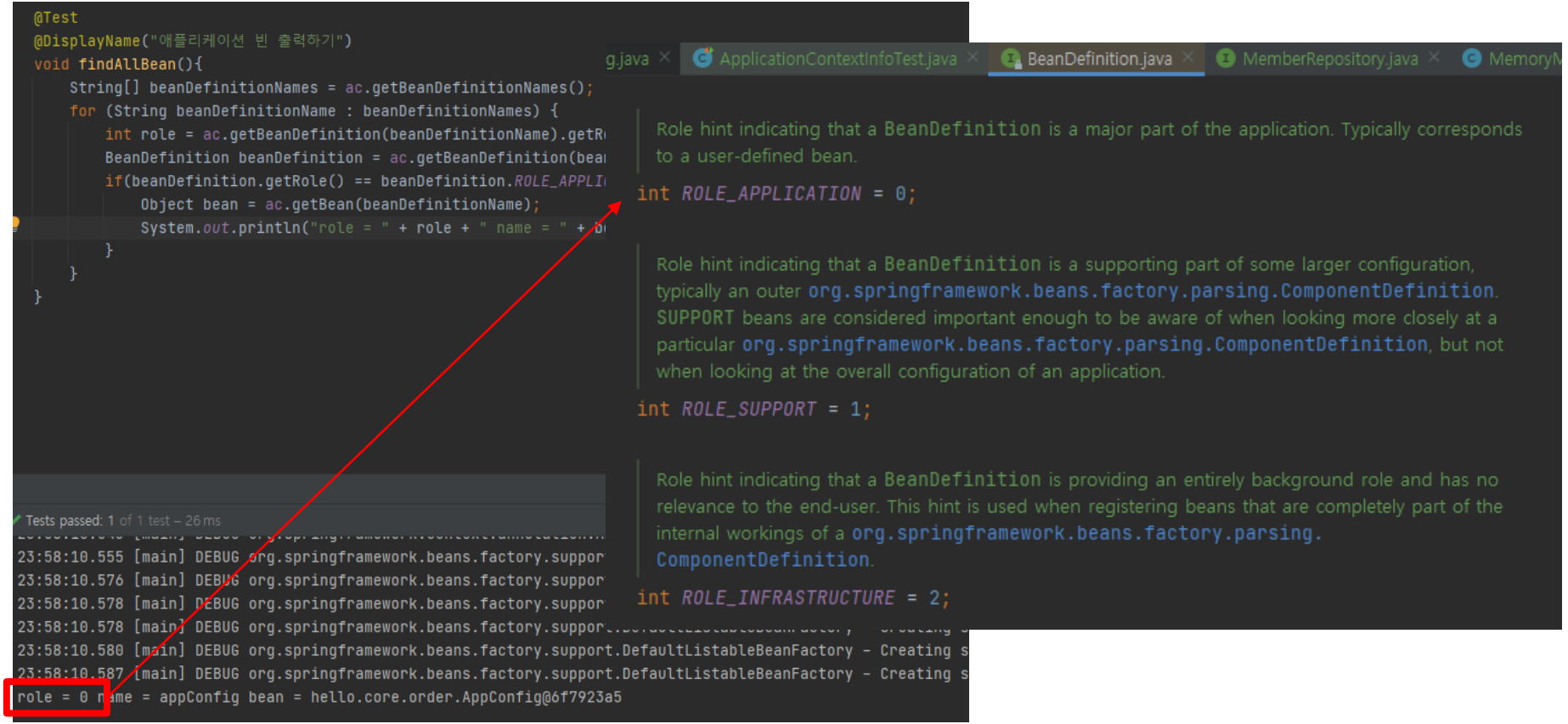
애플리케이션 인프라(infrastructure) 빈

- 애플리케이션 로직 빈을 지원
- 애플리케이션 동작에 직접 참여 but 개발자가 직접 작성한 로직 X
- ex) DataSource, DataSourceTransactionManager

컨테이너 인프라 빈

- 애플리케이션 로직 X (애플리케이션 로직 빈)
- 다른 애플리케이션 로직 빈과 관계 맺고 외부 서비스 사용하는데 도움 X (애플리케이션 인프라 빈)
- only 스프링 컨테이너 기능에 관여

빈의 역할


@Test
@DisplayName("애플리케이션 빈 출력하기")
void findAllBean(){
String[] beanDefinitionNames = ac.getBeanDefinitionNames();
for (String beanDefinitionName : beanDefinitionNames) {
int role = ac.getBeanDefinition(beanDefinitionName).getR
BeanDefinition beanDefinition = ac.getBeanDefinition(bea
if(beanDefinition.getRole() == beanDefinition.ROLE_APPLI
Object bean = ac.getBean(beanDefinitionName);
System.out.println("role = " + role + " name = " + b
}
}
}
g.java
ApplicationContextInfoTest.java
BeanDefinition.java
MemberRepository.java
Role hint indicating that a BeanDefinition is a major part of the application. Typically corresponds to a user-defined bean.
int ROLE_APPLICATION = 0;
Role hint indicating that a BeanDefinition is a supporting part of some larger configuration, typically an outer org.springframework.beans.factory.parsing.ComponentDefinition. SUPPORT beans are considered important enough to be aware of when looking more closely at a particular org.springframework.beans.factory.parsing.ComponentDefinition, but not when looking at the overall configuration of an application.
int ROLE_SUPPORT = 1;
Role hint indicating that a BeanDefinition is providing an entirely background role and has no relevance to the end-user. This hint is used when registering beans that are completely part of the internal workings of a org.springframework.beans.factory.parsing.ComponentDefinition.
int ROLE_INFRASTRUCTURE = 2;
Tests passed: 1 of 1 test – 26 ms
23:58:10.555 [main] DEBUG org.springframework.beans.factory.suppor
23:58:10.576 [main] DEBUG org.springframework.beans.factory.suppor
23:58:10.578 [main] DEBUG org.springframework.beans.factory.suppor
23:58:10.578 [main] DEBUG org.springframework.beans.factory.suppor
23:58:10.580 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
23:58:10.587 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating s
role = 0 name = appConfig bean = hello.core.order.AppConfig@6f7923a5

빈의 역할

- ROLE_APPLICATION
  - 애플리케이션 로직 빈, 애플리케이션 인프라 빈
  - 애플리케이션 동작 중 사용되는 빈
- ROLE_SUPPORT
  - 복합 구조 빈을 정의할 때 보조적으로 사용되는 빈의 역할 지정
  - 실제론 거의 사용되지 않음
- ROLE_INTRASTRUCTURE
  - 컨테이너 인프라 빈
  - ex. <context:annotation-config> 같은 전용태그로 등록되는 것들
- @Role
  - 개발자가 빈 정의할 때 역할값 직접 지정 가능한 어노테이션

IoC / DI 설정 방법의 history

- 스프링 1.x
  - 세 가지 종류의 빈 모두 <bean> 태그로 등록
  - XML 설정 파일 보고 애플리케이션 구성 파악 어려움
- 스프링 2.0
  - 컨테이너 인프라 빈의 전용 태그 등장 → 구분의 용이함
- 스프링 2.5
  - 빈 자동등록 방식, 어노테이션 기반의 의존관계 설정(@Component)
  - XML엔 애플리케이션 인프라 빈, 전용태그 사용하는 컨테이너 인프라 빈만 남게 됨

IoC / DI 설정 방법의 history

- 스프링 3.0
  - 자바 코드로 빈 설정정보, 빈 설정 코드 작성 가능
  - 애플리케이션 로직 빈, 애플리케이션 인프라 빈 → 자바 코드 O
  - 여전히 컨테이너 인프라 빈 등록엔 XML 전용 태그 필요
- 스프링 3.1
  - 컨테이너 인프라 빈도 자바 코드 등록 가능
  - 모든 종류의 빈 설정 방식을 XML, 자바 코드 중 자유롭게 선택 가능

Bean 주입 방법

- 생성자 주입

```
@Component
public class OrderServiceImpl implements OrderService{

  private final MemberRepository memberRepository;
  public OrderServiceImpl(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }
}
```

Bean 주입 방법

- 필드 주입

```
@Component
public class OrderServiceImpl implements OrderService{

  @Autowired
  private MemberRepository memberRepository;
}
```

Bean 주입 방법

- 수정자 주입(Setter Injection)

```
@Component
public class OrderServiceImpl implements OrderService{

  private MemberRepository memberRepository;

  @Autowired
  public void setMemberRepository(MemberRepository memberRepository){
    this.MemberRepository = memberRepository;
  }
}
```

생성자 주입 권장 이유

- 순환 참조 방지
  - 수정자 주입, 필드 주입은 애플리케이션 오류/경고 없이 구동
  - 생성자 주입일 경우 BeanCurrentlyInCreationException 과 함께 애플리케이션 구동 X
  - 빈을 주입하는 순서가 다르기 때문
    - 생성자 주입만 객체 생성 시점에 빈을 주입
- 테스트를 편리하게 작성 가능
- 의존성이 명시적으로 드러나 코드의 품질 ↑
- final 선언 가능 → 런타임 시점에 변경X 오류 방지
- Lombok으로 편리하게 사용가능

```
@Component
@RequiredArgsConstructor //final 변수를 위한 생성자 생성
public class OrderServiceImpl implements OrderService{
  private final MemberRepository memberRepository;
}
```