

스프링 빈과 의존 관계 주입



20210463 박연종

2024.05.22.

인터페이스에만 의존해야 하는 이유?

```
public interface OrderService { // ... }
public interface DiscountPolicy { // ... }

// 어떤 쇼핑몰에서 초창기에 고객 유치를 위해 VIP 고객에게 무조건 만 원 할인 정책을 적용
public class Vip10000FixDiscountPolicy implements DiscountPolicy {
    public int discount(Item item) {
        // ...
        return item.price > 10000 ? item.price - 10000 : 0;
    }
}

public class OrderServiceImpl implements OrderService {
    private final Vip10000FixDiscountPolicy discountPolicy = new Vip10000FixDiscountPolicy();

    public CartItem addCartItem(User user, Item item) {
        int discountPrice = discountPolicy.discount(user, item);
        return new CartItem(cart, item, discountPrice);
    }
}

public class Main {
    public static void main(String[] args) {
        OrderService orderService = new OrderServiceImpl();
        // ...
        CartItem cartItem = orderService.addCartItem(user, item);
    }
}
```


인터페이스에만 의존해야 하는 이유?

```
public interface OrderService { // ... }
public interface DiscountPolicy { // ... }
public class Vip10000FixDiscountPolicy implements DiscountPolicy {

    // 사용자 유치 후 VIP 고객에게 10만 원 이상 구매 시 각 상품을 10% 할인하는 정책을 적용
    public class Vip10PerRateDiscountPolicy implements DiscountPolicy {
        public int discount(User user, Item item) {
            // ...
            return item.price > 100000 ? item.price - item.price * 0.1 : item.price;
        }
    }
}

public class OrderServiceImpl implements OrderService {
    // private final Vip10000FixDiscountPolicy discountPolicy = new Vip10000FixDiscountPolicy();
    private final Vip10PerRateDiscountPolicy discountPolicy = new Vip10PerRateDiscountPolicy();

    public CartItem addCartItem(User user, Item item) {
        int discountPrice = discountPolicy.discount(user, item);
        return new CartItem(cart, item, discountPrice);
    }
}
```



인터페이스에만 의존해야 하는 이유?

쇼핑몰에서 정책을 변경할 때마다 개발자는

새로운 정책 클래스 추가와 함께

기존 정책을 사용하는 클래스에서

기존 정책의 객체 인스턴스를 새로운 정책의 객체 인스턴스로 바꾸어야 한다

만약 기존 정책을 사용하는 클래스가 수백 개라면?

인터페이스에만 의존해야 하는 이유?

```
public interface OrderService { // ... }
public interface DiscountPolicy { // ... }
public class FixDiscountPolicy implements DiscountPolicy {
public class Vip10PerRateDiscountPolicy implements DiscountPolicy { // ... }

public class OrderServiceImpl implements OrderService {
// private final Vip10PerRateDiscountPolicy discountPolicy = new Vip10PerRateDiscountPolicy();
private final DiscountPolicy discountPolicy;
// ...
}
```

변경이 용이하도록
인터페이스 타입으로 만들자!

구현체(객체 인스턴스)를 제거하자!

이대로 실행하면...

Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "DiscountPolicy.discount()" because "discountPolicy" is null

클래스 수백 개를 수정하기 싫어 구현체를 제거했더니 NPE 발생
무슨 짓을 해서라도 구현체를 전달해야 한다!

1

외부에서 구현체를 전달하는 방법

외부에서 구현체를 전달하자

```
public class OrderServiceImpl implements OrderService {
    private final DiscountPolicy discountPolicy;

    public OrderServiceImpl(DiscountPolicy discountPolicy) {
        this.discountPolicy = discountPolicy;
    }

    // ...
}
```

```
public class AppConfig {
    public DiscountPolicy discountPolicy() {
        return new Vip10PerRateDiscountPolicy();
    }
}
```

```
public OrderService orderService() {
    return new OrderServiceImpl(
        discountPolicy()
    );
}
```

```
}
```

```
public class Main {
    public static void main(String[] args) {
        AppConfig appConfig = new AppConfig();
        OrderService orderService = appConfig.orderService();
        // ...
        CartItem cartItem = orderService.addCartItem(user, item);
    }
}
```

각각의 클래스를 수정하지 않고 메서드 하나만 바꾸면 된다!

2

스프링 컨테이너와 스프링 빈

스프링 컨테이너 생성과 설정 정보

```
public class OrderServiceImpl implements OrderService {
    private final DiscountPolicy discountPolicy;

    public OrderServiceImpl(DiscountPolicy discountPolicy) {
        this.discountPolicy = discountPolicy;
    }

    // ...
}

@Configuration
public class AppConfig {

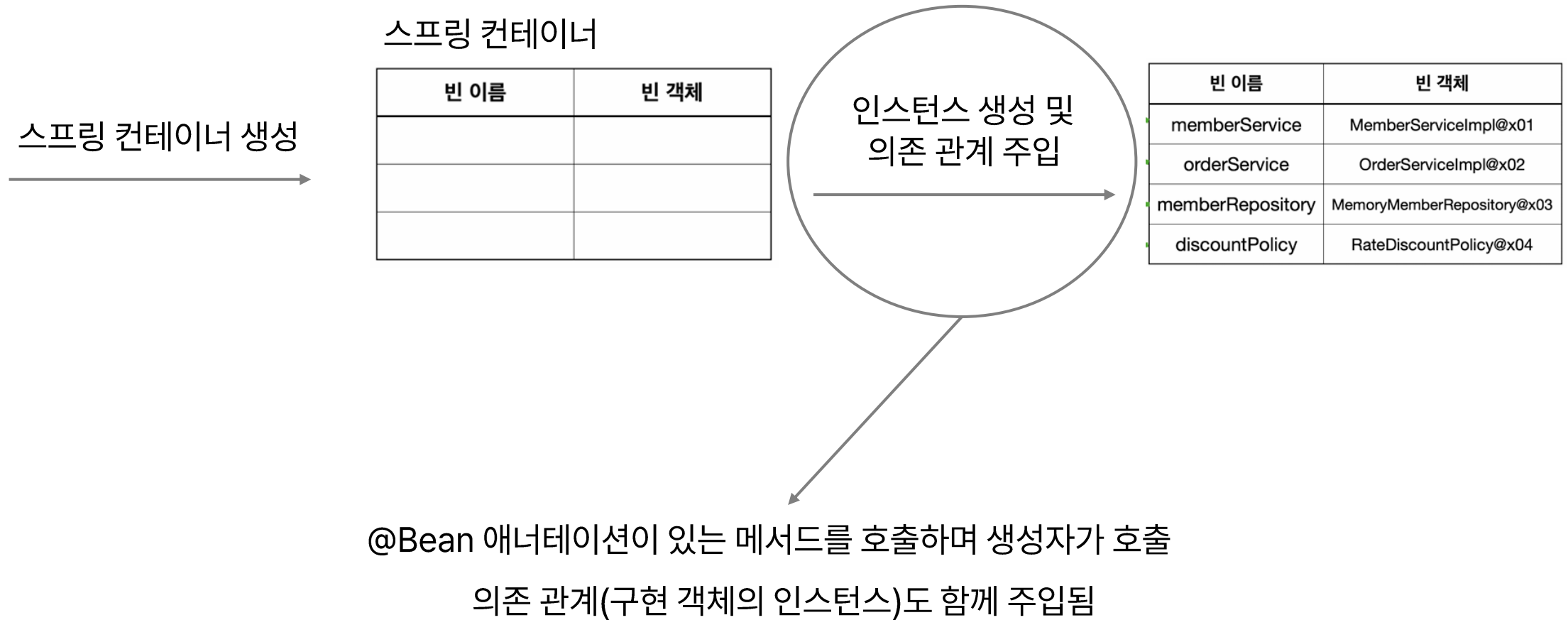
    @Bean
    public DiscountPolicy discountPolicy() {
        return new Vip10PerRateDiscountPolicy();
    }

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

public class Main {
    public static void main(String[] args) {
        ApplicationContext ac =
            new AnnotationConfigApplicationContext(AppConfig.class);
        OrderService orderService = ac.getBean(OrderService.class)
        // ...
        CartItem cartItem = orderService.addCartItem(user, item);
    }
}
```

설정 정보를 통한 스프링 컨테이너 생성 과정

```
ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
```



쇼핑몰에서 새로운 기능을 추가할 때마다 메서드를 추가하거나
일부 메서드의 파라미터를 조정하기
매우 귀찮다!

자동으로 해주면 좋을 텐데...

컴포넌트 스캔

```
@Component
public class Vip10PerRateDiscountPolicy implements DiscountPolicy { // ... }
```

```
@Component
public class OrderServiceImpl implements OrderService { // ... }
```

```
@Configuration
@ComponentScan
public class AppConfig { }

public class Main {
    public static void main(String[] args) {
        ApplicationContext ac =
            new AnnotationConfigApplicationContext(AppConfig.class);
        OrderService orderService = ac.getBean(OrderService.class)
        // ...
        CartItem cartItem = orderService.addCartItem(user, item);
    }
}
```

@Component 애너테이션이 있는 클래스는 알아서 스프링 빈으로 등록해준다!

그런데 의존 관계를 어떻게 넣어주지?

의존 관계 주입은 어떻게?

@Autowired로 자동으로 연결해 드립니다

3

자동으로 주입해 주는 여러가지 방법

생성자 주입

- 생성자 호출 시점에 한 번만 호출되는 것이 보장됨
- 나중에 다른 객체로 변하지 않거나 꼭 필요한 객체에 사용됨

@Component

```
public class OrderServiceImpl implements OrderService {  
    private final MemberRepository memberRepository;  
    private final DiscountPolicy discountPolicy;
```

@Autowired

```
public OrderServiceImpl(MemberRepository memberRepository, DiscountPolicy discountPolicy) {  
    this.memberRepository = memberRepository;  
    this.discountPolicy = discountPolicy;  
}
```

```
}
```

수정자 주입

- 필드 값을 변경하는 메서드를 통해 의존 관계를 주입하는 방법
- 나중에 다른 객체로 변할 수 있거나 메서드에 따라 필요한 객체가 다른 경우 사용됨

@Component

```
public class OrderServiceImpl implements OrderService {  
    private MemberRepository memberRepository;  
    private DiscountPolicy discountPolicy;
```

@Autowired

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

@Autowired

```
public void setDiscountPolicy(DiscountPolicy discountPolicy) {  
    this.discountPolicy = discountPolicy;  
}
```

```
}
```

필드 주입

- 필드에 값(의존 관계)을 바로 주입하는 방법
- 외부에서 객체를 변경할 수 없어 테스트할 때 불편함

```
@Component
public class OrderServiceImpl implements OrderService {

    @Autowired
    private MemberRepository memberRepository;

    @Autowired
    private DiscountPolicy discountPolicy;

}
```


정리

[인터페이스에만 의존해야 하는 이유]

- 정책 변경으로 인해 기존 정책 객체를 새로운 정책 객체로 바꿔야 할 필요가 있음
- 기존 정책 객체를 사용하는 클라이언트 객체가 여러 개이면 바꾸기 쉽지 않음

⇒ 클라이언트 객체에서 바로 정책 객체를 사용하지 않고 외부에서 정책 객체를 알려주면 된다!

[스프링 컨테이너와 스프링 빈]

- 개발자 대신 스프링이 객체의 생성과 소멸을 대신 하게 함

[자동으로 주입해 주는 여러가지 방법]

- 스프링 빈을 스프링이 알아서 등록하게 하면 객체 생성 시 필요한 다른 객체를 전달해야 함

⇒ @Autowired 애너테이션을 적용해 스프링이 알아서 의존 관계를 주입하도록 한다!

감사합니다



Email / park@duck.com

Insta / [@yeonjong.park](https://www.instagram.com/yeonjong.park)

GitHub / [patulus](https://github.com/patulus)