컴포넌트 스캔

컴포넌트 스캔과 의존관계 자동 주입 시작하기

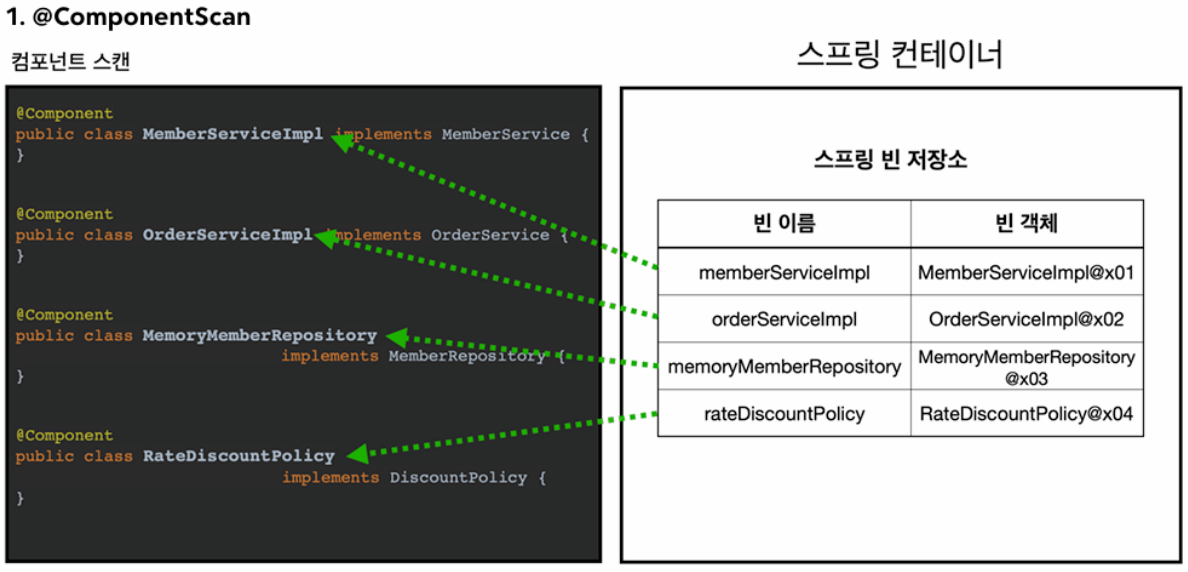
- 지금까지는 `@Bean`을 통해 직접 등록 → 귀찮음

⇒ 스프링은 설정 정보 없이 자동으로 스프링 빈을 등록하는 `컴포넌트 스캔` 기능 제공

⇒ 또 의존관계도 자동으로 주입하는 `@Autowired` 라는 기능도 제공

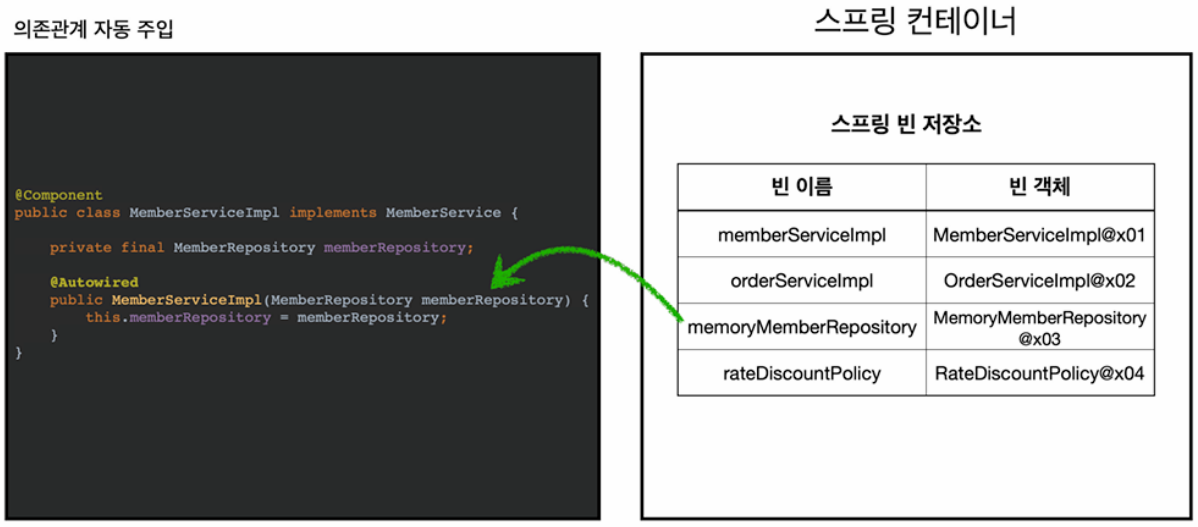
@ComponentScan

@Component가 붙은 클래스를 찾아 자동으로 스프링 빈으로 등록



- `@ComponentScan`은 `@Component`가 붙은 모든 클래스를 스프링 빈으로 등록한다.
- 이때 스프링 빈의 기본 이름은 클래스명을 사용하되 맨 앞글자만 소문자를 사용한다.
  - **빈 이름 기본 전략:** MemberServiceImpl 클래스➔ memberServiceImpl
  - **빈 이름 직접 지정:** 만약 스프링 빈의 이름을 직접 지정하고 싶으면 `@Component("memberService2")` 이런식으로 이름을 부여하면 된다.

2. @Autowired 의존관계 자동 주입

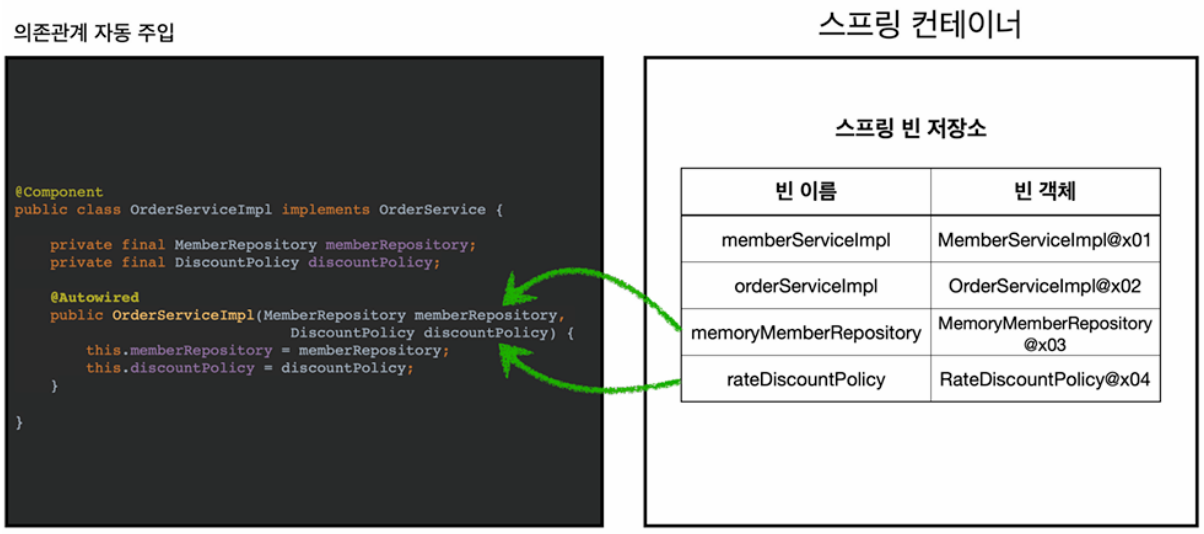



```java
@Component
public class MemberServiceImpl implements MemberService {

    private final MemberRepository memberRepository;

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

- 생성자에 `@Autowired`를 지정하면, 스프링 컨테이너가 자동으로 해당 스프링 빈을 찾아서 주입한다.
- 이때 기본 조회 전략은 타입이 같은 빈을 찾아서 주입한다.
  - `getBean(MemberRepository.class)`와 동일하다고 이해하면 된다.
  - 더 자세한 내용은 뒤에서 설명한다.




```java
@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;
    }

}
```

- 생성자에 파라미터가 많아도 다 찾아서 자동으로 주입한다.

탐색 위치와 기본 스캔 대상

```
@Configuration
@ComponentScan(
        basePackages = "hello.core.member",
        basePackageClasses = AutoAppConfig.class,
        excludeFilters = @ComponentScan.Filter(type = Filte
rType.ANNOTATION, classes = Configuration.class)
        // 지금까지 만들었던 AppConfig와 충돌 피하기 위함
)
public class AutoAppConfig {


}
```

basePackageClasses 를 지정하지 않으면 hello.core부터 모든 패키지를 다 뒤짐

권장하는 방법
패키지 위치를 지정하지 않고, 설정 정보 클래스의 위치를 프로젝트 최상단에 두는 것.

예를 들어서 프로젝트가 다음과 같이 구조가 되어 있으면

- `com.hello`
- `com.hello.serivce`
- `com.hello.repository`

`com.hello` ➜ 프로젝트 시작 루트, 여기에 AppConfig 같은 메인 설정 정보를 두고, @ComponentScan 애노테이션을 붙이고, `basePackages` 지정은 생략한다.

컴포넌트 스캔 기본 대상

- `@Component` : 컴포넌트 스캔에서 사용
- `@Controller` : 스프링 MVC 컨트롤러에서 사용
- `@Service` : 스프링 비즈니스 로직에서 사용
- `@Repository` : 스프링 데이터 접근 계층에서 사용
- `@Configuration` : 스프링 설정 정보에서 사용

애노테이션은 상속관계라는 게 없음.

애노테이션은 java 언어와 관계 X 스프링이 지원하는 기능

컴포넌트 스캔의 용도 뿐만 아니라 다음 애노테이션이 있으면 스프링은 부가 기능을 수행한다.

- `@Controller` : 스프링 MVC 컨트롤러로 인식
- `@Repository` : 스프링 데이터 접근 계층으로 인식하고, 데이터 계층의 예외를 스프링 예외로 변환해준다.
- `@Configuration` : 앞서 보았듯이 스프링 설정 정보로 인식하고, 스프링 빈이 싱글톤을 유지하도록 추가 처리를 한다.
- `@Service` : 사실 `@Service` 는 특별한 처리를 하지 않는다. 대신 개발자들이 핵심 비즈니스 로직이 여기에 있겠구나 라고 비즈니스 계층을 인식하는데 도움이 된다.

필터

중복 등록과 충돌

- 컴포넌트 스캔에서 같은 빈 이름을 등록하면?

1. 자동 빈 등록 vs 수동 빈 등록
2. 자동 빈 등록 vs 자동 빈 등록

자동 빈 등록 vs 자동 빈 등록

- 컴포넌트 스캔에 의해 자동으로 스프링 빈이 등록되는데, 그 이름이 같은 경우 스프링은 오류를 발생시킨다.
  - `ConflictingBeanDefinitionException` 예외 발생

수동 빈 등록 vs 자동 빈 등록

만약 수동 빈 등록과 자동 빈 등록에서 빈 이름이 충돌되면 어떻게 될까?

```
@Component
public class MemoryMemberRepository implements MemberRepository {}
```

```
@Configuration
@ComponentScan(
        excludeFilters = @Filter(type = FilterType.ANNOTATION, classes =
Configuration.class)
)
public class AutoAppConfig {

    @Bean(name = "memoryMemberRepository")
    public MemberRepository memberRepository() {
        return new MemoryMemberRepository();
    }
}
```

이 경우 수동 빈 등록이 우선권을 가진다.
(수동 빈이 자동 빈을 오버라이딩 해버린다.)

수동 빈 등록시 남는 로그

```
Overriding bean definition for bean 'memoryMemberRepository' with a different
definition: replacing
```

물론 개발자가 의도적으로 이런 결과를 기대했다면, 자동 보다는 수동이 우선권을 가지는 것이 좋다. 하지만 현실은 개발자가 의도적으로 설정해서 이런 결과가 만들어지기 보다는 여러 설정들이 꼬여서 이런 결과가 만들어지는 경우가 대부분이다!
그러면 정말 잡기 어려운 버그가 만들어진다. 항상 잡기 어려운 버그는 애매한 버그다.
그래서 최근 스프링 부트에서는 수동 빈 등록과 자동 빈 등록이 충돌나면 오류가 발생하도록 기본 값을 바꾸었다.

수동 빈 등록, 자동 빈 등록 오류시 스프링 부트 에러

```
Consider renaming one of the beans or enabling overriding by setting
spring.main.allow-bean-definition-overriding=true
```

스프링 부트인 `CoreApplication`을 실행해보면 오류를 볼 수 있다.