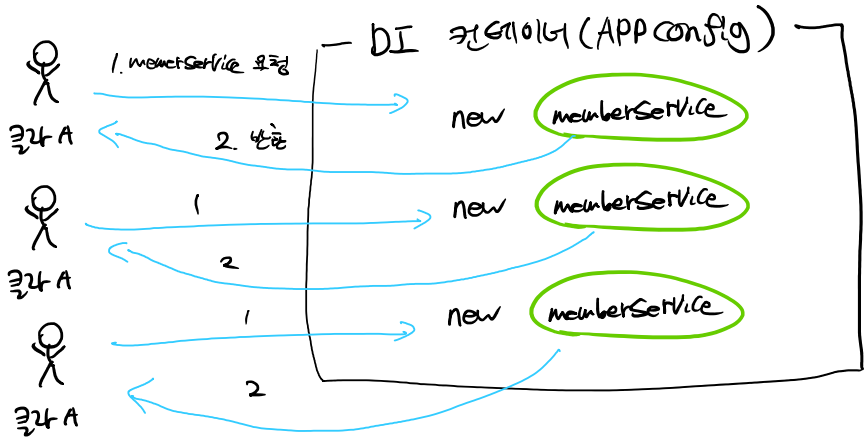
스프링핵심원리-싱글톤 컨테이너

2021년 8월 9일 월요일 오후 2:35

웹 어플리케이션과 싱글톤

스프링은 근본이 기업용 온라인 서비스 기술 지원을 목적
대부분이 웹 어플리케이션, But 아닌 경우도 가능
웹 어플리케이션은 보통 여러 고객이 동시에 요청



계속 객체를 이렇게 여러 고객들이 요청할 때마다 만드는 것은 효율적이지 못함
스프림 없는 순수한 DI 컨테이너는 위와 같이 요청할 때마다 객체 새로 생성
=> 메모리 낭비가 극심했음
=> 객체를 1개만 생성하고 그것을 공유하도록 설계 (== 싱글톤 패턴)

싱글톤 패턴
클래스의 인스턴스가 딱 1개만 생성되는 것을 보장하는 디자인 패턴
How?
=> 객체 인스턴스를 2개 이상 생성하지 못하게 막아야함
=> private 생성자를 이용하여 외부에서 임의로 new 키워드 사용을 막는다

```java
public class singletonService {
    //1. static 영역에 객체를 딱 1개만 생성해둔다
    private static final singletonService instance = new singletonService();

    //2. public으로 열어서 객체 인스턴스가 필요하면 이 static 메소드를 통해서만 조회하도록 허용
    public static singletonService getInstance() {
        return instance;
    }
    //3. 생성자를 private으로 선언해서 외부에서 new 키워드를 사용한 객체 생성을 못하게 막는다
    private singletonService() {

    }
```

static 을 사용하면 클래스의 레벨에 올라가서 딱 하나만 존재

1. static 영역에 객체 인스턴스를 미리 하나 생성해서 올려둠
2. 이 객체 인스턴스가 필요하면 오직 getInstance() 메소드를 통해서만 조회할 수 있음
   이 메소드를 호출하면 항상 같은 인스턴스를 반환한다
3. 딱 1개의 객체 인스턴스만 존재해야 하므로, 생성자를 private으로 막아서 혹시라도
   외부에서 new 키워드로 객체 인스턴스가 생성되는 것을 막는다.

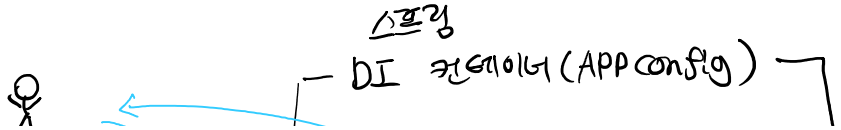
스프링컨테이너는 기본적으로 객체 싱글톤으로 관리해줌

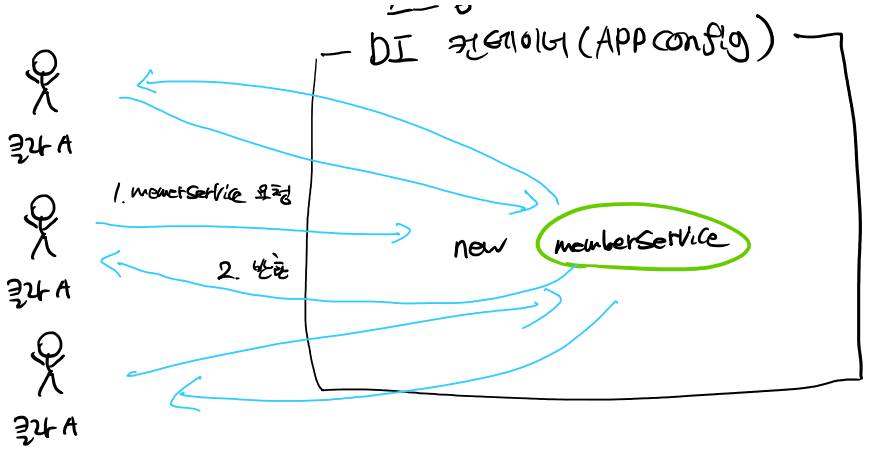
싱글톤 패턴의 문제점:
1. 싱글톤 구현 위한 코드 필요
2. 의존관계상 클라이언트가 구체 클래스에 의존 -> DIP 위반, OCP 위반 가능성
3. 테스트가 어려움
4. 내부 속성 변경, 초기화 어려움
5. private 생성자로 자식 클래스 만들기 어려움
6. 즉 유연성이 떨어지고 안티패턴으로 불리기도 함

스프링이 이러한 문제점을 전부 해결해버림... How?

싱글톤 컨테이너

스프링 컨테이너는 싱글톤 패턴 적용하지 않고 싱글톤으로 관리함
컨테이너는 객체를 하나만 생성해서 관리함
스프링 컨테이너는 싱글톤 컨테이너 역할을 함. 이렇게 싱글톤 객체를 생성하고 관리하는 기능을 싱글톤 레지스트리라 함
스프링 컨테이너의 이러한 기능으로 싱글톤 패턴의 모든 단점을 해결하면서 객체를 싱글톤으로 유지할 수 있다.





싱글톤 방식의 주의점

싱글톤 패턴이든, 스프링 같은 싱글톤 컨테이너를 사용하든, 객체 인스턴스를 하나만 생성해서 공유하는 싱글톤 방식은 여러 클라이언트가 하나의 같은 객체 인스턴스를 공유함
=> 싱글톤 객체는 상태를 유지(stateful)하게 설계하면 안됨
즉 무상태(stateless) 하게 설계해야됨

무상태(stateless)란

- 특정 클라이언트에 의존적인 필드 있으면 안됨
- 특정 클라이언트가 값을 변경할 수 있는 필드가 있으면 안됨
- 가급적 읽기만 가능해야함 (가급적 값 수정X)
- 필드 대신에 자바에서 공유되지 않는, 지역변수, 파라미터, ThreadLocal 등을 사용해야

스프링 빈의 필드에 공유 값을 설정하면 정말 큰 장애 발생할 수 있음

@Configuration과 바이트코드 조작

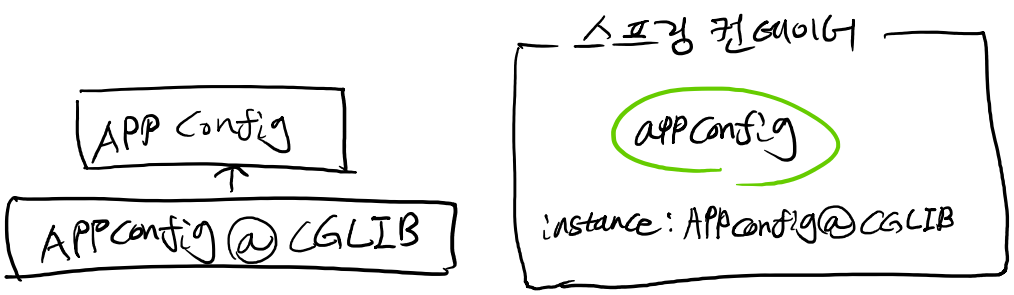
스프링은 클래스의 바이트코드를 조작하는 라이브러리를 사용함
@Configuration

```java
@Test
void configurationDeep() {
    ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
    AppConfig bean = ac.getBean(AppConfig.class);

    System.out.println("bean = " + bean.getClass());
}
```

```
bean = class hello.core.AppConfig$$EnhancerBySpringCGLIB$$121335e9
```

내가 만든 클래스가 아니라 스프링이 CGLIB이라는 바이트코드 조작 라이브러리를 사용해서 AppConfig 클래스를 상속받은 임의의 다른 클래스를 만들고, 그것을 스프링 빈으로 등록한것



@Bean 만 사용해도 스프링 빈으로 등록되지만, 싱글톤 보장X
EX) memberRepository() 처럼 의존관계 주입이 필요해서 메소드 직접 호출할 때 싱글톤 보장X
스프링 설정정보(Appconfig)는 항상 @Configuration 사용하자