IoC 컨테이너와 DI

스프링으로 객체지향 프로그래밍을 한다는 것은?

스프링이란?

- 자바언어를 사용하는 애플리케이션 프레임워크
  - 스프링 컨테이너 또는 애플리케이션 컨텍스트라는 스프링 런타임 엔진을 제공
- IoC/DI 를 프레임워크를 기본으로 사용
  - 객체지향 설계원칙과 디자인 패턴의 원리가 녹아 있음
- 개발에 바로 사용할 수 있는 다양한 기술 API들을 제공
  - 개발자는 조립하듯이 개발할 수 있음

스프링의 장점

- 단순함
  - 단순한 객체지향 개발모델인 POJO를 지원한다
  - 객체간의 관계를 간단한 JAVA Class로 표현할 수 있다

- 유연성
  - 프레임워크 근간을 유지하면서 확장해서 사용가능
  - Network : netty, mina
  - DB : hibernate, mybatis

- 범용적 확장성
  - 전자정부프레임워크, SDS, CNS, C&C, 금융권 등 다양한 분야에서 스프링을 커스터마이징 해서 사용 중

관심사의 분리

- 추상화 수준에서 다양한 객체에 대해 모델링 했을 때 분리와 확장을 고려해서 설계하는 것
- 관심이 같은 것끼리는 하나의 객체관점으로 공통 분모를 만든다
- 관심이 다른 것은 서로 떨어지게 해서 영향을 주지 않도록 하는 것

인터페이스를 사용할 줄 안다는 것

- 높은 응집도
  - 변화가 일어날 때 모듈 or 클래스에서 공통적으로 변하는 부분이 많다는 것
  - 관심사를 하나로 집중해서 묶어 둔 것

- 낮은 결합도
  - 객체간 관계가 간접적으로 연결되어 서로 독립적으로 구분되거나 구현부분에 대한 종속 없이 껍데기만 가지고 있는 것
  - DI를 통해 낮은 결합도로 결합

제어권 이전을 통한 관계 설정

- 절차적(능동적) 관계 VS 수동적 관계
  - 능동적인 생성과 사용 VS 객체의 생성과 사용에 대해 알 수 없음

- 라이브러리 VS 프레임워크
  - API 함수 사용 VS 컴포넌트의 생성과 관계설정, 생명주기를 관장하는 Manager 존재 필요

스프링의 IoC

- 빈(제어권을 가진 오브젝트)
- 빈 팩토리(빈의 생성과 관계설정)
- 애플리케이션 컨텍스트(설정정보를 가지고 있는 IoC 엔진 (유일한 능동적 객체))
  - @Configuration 사용(설정정보라는 표시)
  - @Bean 사용(오프젝트 생성을 담당하는 IoC 메소드라는 표시)

애플리케이션 컨텍스트의 동작 방식

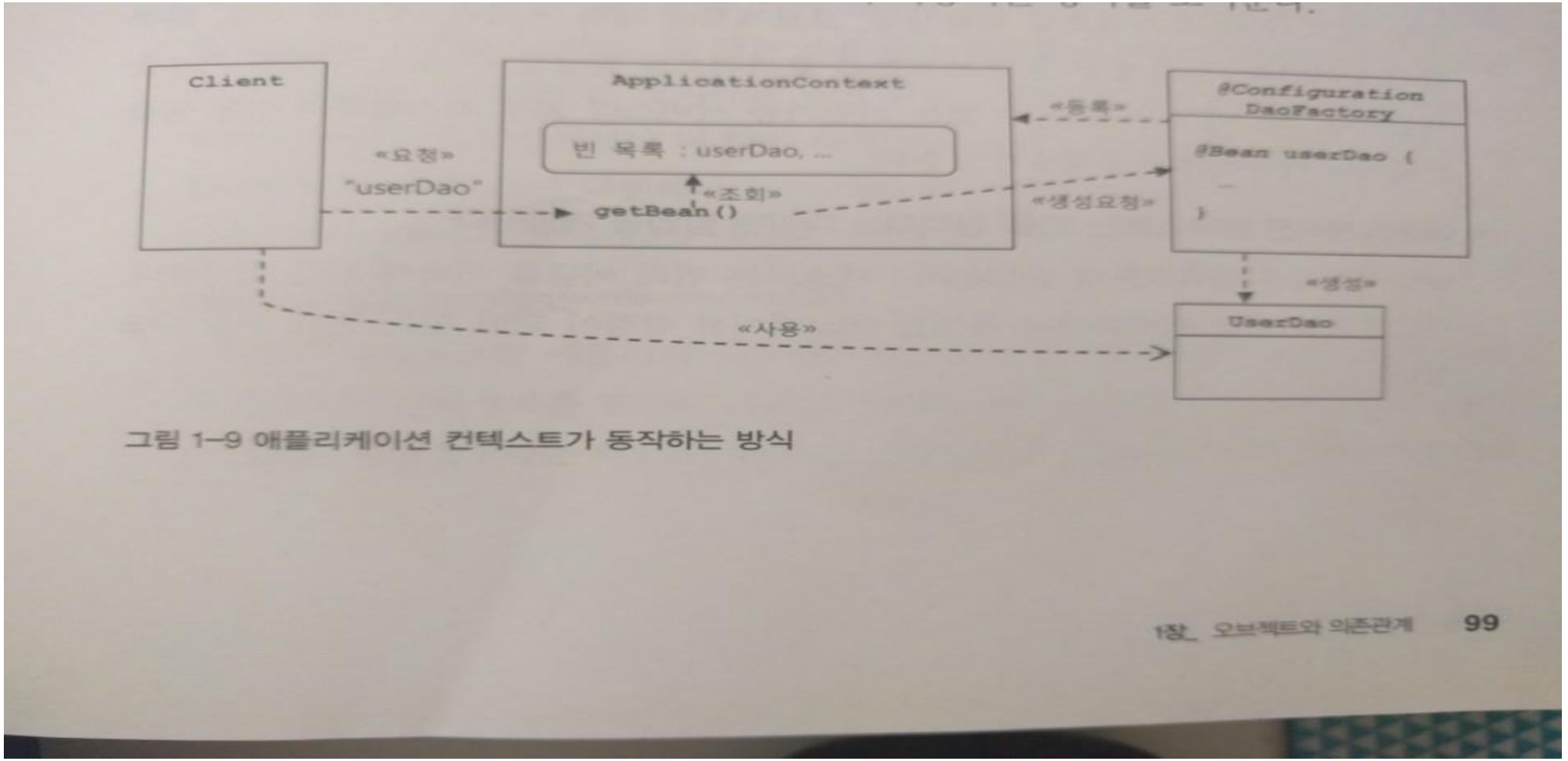


그림 1-9 애플리케이션 컨텍스트가 동작하는 방식

스프링의 DI

- 의존 관계
  - A ------> B
  - A가 B에 의존한다 즉, B가 변하면 A에 영향을 끼친다

- 스프링의 IoC 컨테이너는 동작방식은 인터페이스를 통해 인스턴스를 생성하는 다이나믹한 방법이지만 의존관계를 미리 설정함으로 의존관계를 주입한다고 할 수 있다

- 인터페이스가 변하면 그 구현체나 의존성이 주입된 객체에 영향을 미치나 인터페이스를 사이에 둔 두 객체간 결합도는 낮다

의존 관계 주입의 핵심

- 클래스 모델이나 코드에서 런타임 시점에는 의존관계가 주입되지 않음(다이나믹 속성)
- 런타임 시점의 의존관계는 컨테이너나 팩토리 같은 제 3의 존재가 결정
- 의존관계는 사용할 오브젝트에 대한 인터페이스를 주입해줌으로 만들어짐

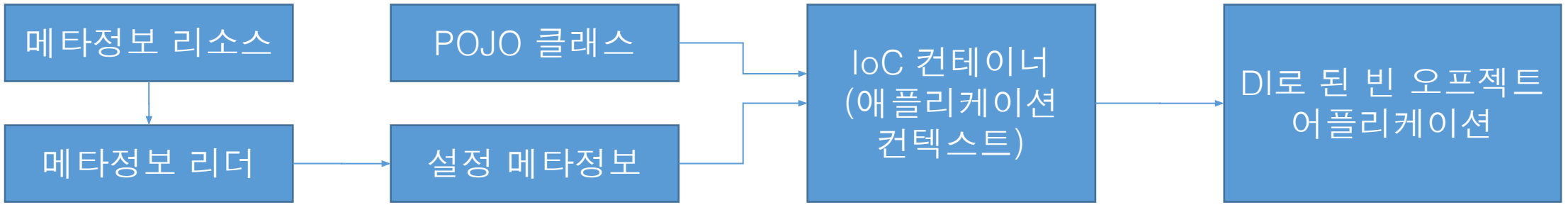
IoC 컨테이너와 DI

- 스프링의 IoC 컨테이너
  - 애플리케이션 컨텍스트
  - 빈 팩토리

- 스프링 애플리케이션은 최소한 하나 이상의 IoC 컨테이너 즉 애플리케이션 컨텍스트를 갖고 있다

POJO 클래스

- 코드로 구현된 Object
- 다른 Object와 관계를 가지는 것에 대해 느슨하게 결합되어야 한다 (Interface 필요)



설정 메타정보

- 스프링 컨테이너가 관리하는 Bean들을 어떻게 만들고 어떻게 동작하게 할 것인가에 관한 정보

- 빈 아이디, 이름, 별칭
- 클래스 또는 클래스 이름
- 스코프
- 프로퍼티 값 또는 참조
- 생성자 파라미터 값 또는 참조
- 지연된 로딩여부, 우선빈 여부, 자동와이어링 여부, 부모빈 정보, 빈팩토리 등

IoC 컨테이너 종류와 사용 방법

```java
public void registerBean() {
    StaticApplicationContext ac = new StaticApplicationContext();
    ac.registerSingleton("hello1", Hello.class);

    Hello hello1 = ac.getBean("hello1", Hello.class);
    assertThat(hello1, is(notNullValue()));

    BeanDefinition helloDef = new RootBeanDefinition(Hello.class);
    helloDef.getPropertyValues().addPropertyValue("name", "Spring");
    ac.registerBeanDefinition("hello2", helloDef);

    Hello hello2 = ac.getBean("hello2", Hello.class);
    assertThat(hello2.sayHello(), is("Hello Spring"));
    assertThat(hello1, is(not(hello2)));

    assertThat(ac.getBeanFactory().getBeanDefinitionCount(), is(2));
}
```

IoC 컨테이너 종류와 사용 방법

```java
public void registerBeanWithDependency() {
    StaticApplicationContext ac = new StaticApplicationContext();

    ac.registerBeanDefinition("printer", new RootBeanDefinition(StringPrinter.class));

    BeanDefinition helloDef = new RootBeanDefinition(Hello.class);
    helloDef.getPropertyValues().addPropertyValue("name", "Spring");
    helloDef.getPropertyValues().addPropertyValue("printer", new RuntimeBeanReference("printer"));
    ac.registerBeanDefinition("hello", helloDef);

    Hello hello = ac.getBean("hello", Hello.class);
    hello.print();

    assertThat(ac.getBean("printer").toString(), is("Hello Spring"));
}
```

WebApplicationContext

- 가장마니사용되는 애플리케이션 컨텍스트
- XML 설정파일을 사용하는 XmlWebApplication

- 웹 애플리케이션은 Main()가 없다 대신에 서블릿 컨테이너가 브라우저로 부터 오는 HTTP 요청을 인터넷을 타고 들어와 해당요청에 매핑되는 서블릿을 실행해주는 것으로 서블릿이 일종의 Main() 메소드와 같은 역할을 한다

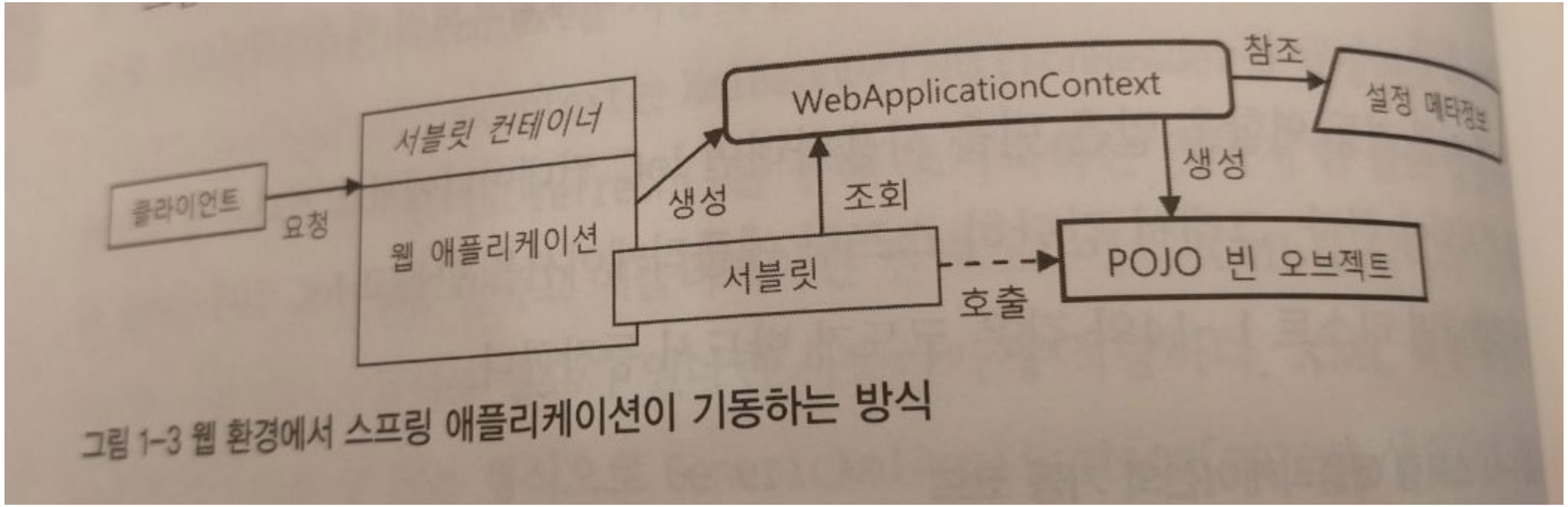


그림 1-3 웹 환경에서 스프링 애플리케이션이 기동하는 방식

IoC 컨테이너 계층구조

- 루트 어플리케이션 컨텍스트

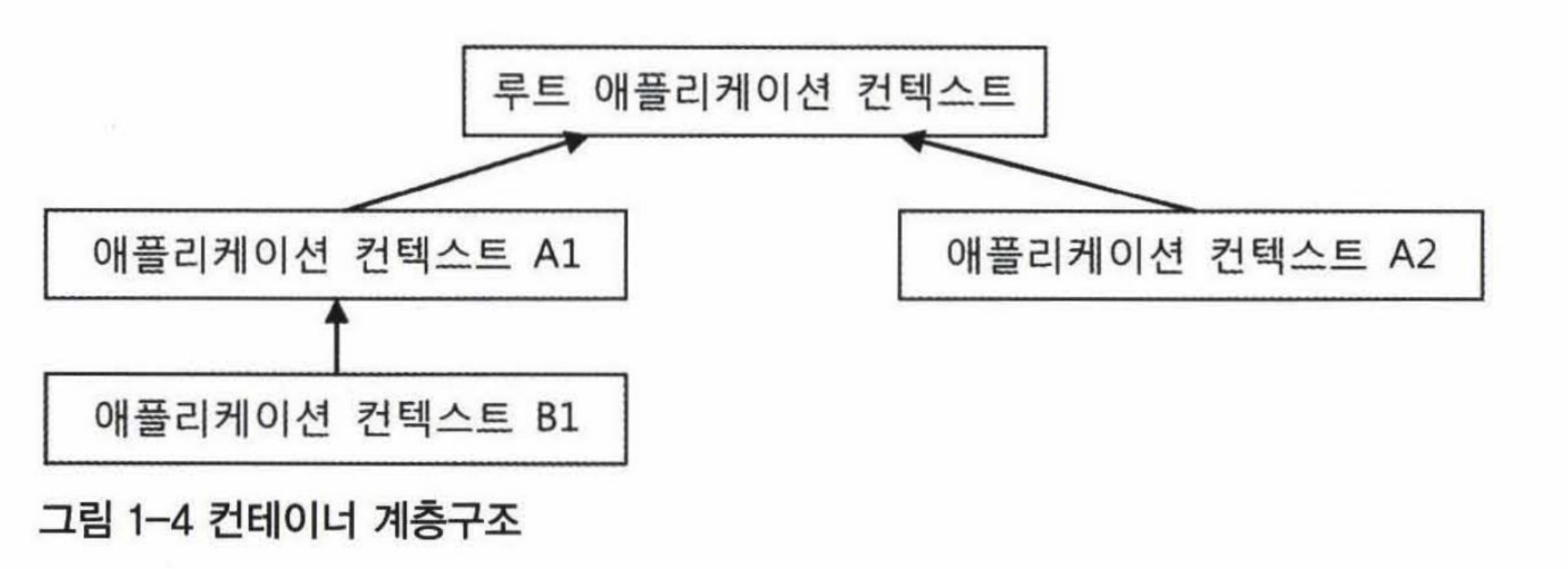


그림 1-4 컨테이너 계층구조

웹 어플리케이션의 IoC 컨테이너 구성

- 몇 개의 서블릿이 중앙집중식으로 핸들러라는 녀석을 통해 요청을 받아 처리하는 방식
  - 프론트 컨트롤러 패턴

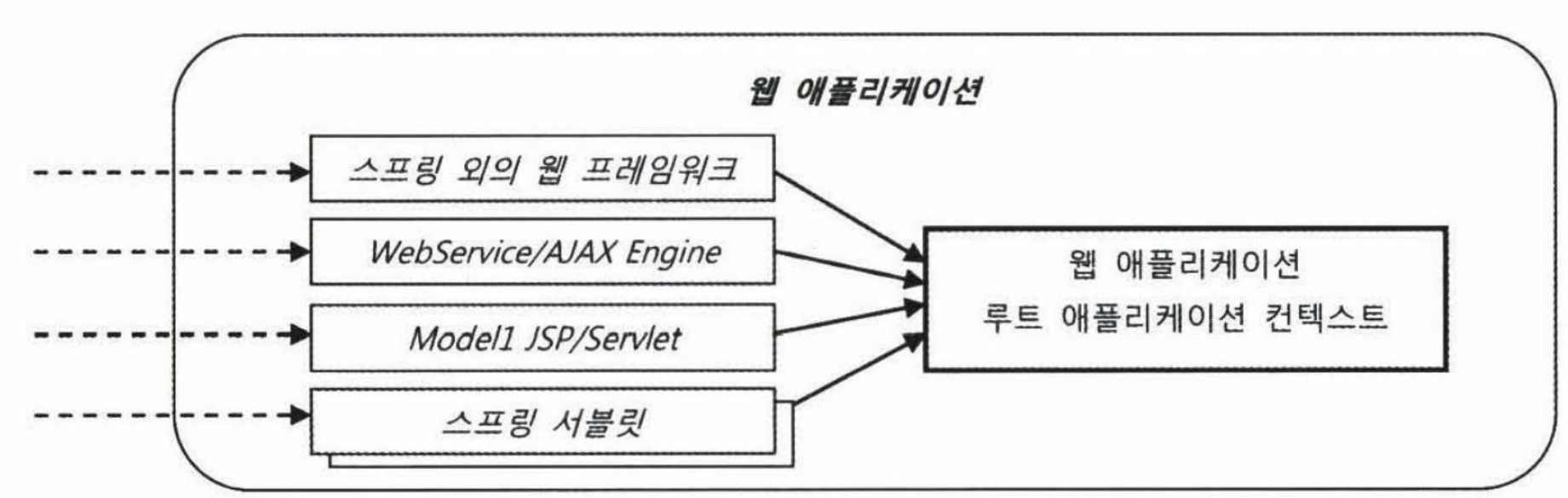


그림 1-7 스프링 웹 애플리케이션의 다양한 구성 방법

빈 등록(메타정보로 저장)

- IoC 컨테이너의 가장 기본적인 역할은 애플리케이션을 구성하는 오브젝트를 생성하고 관리하는 것 (POJO로 만들어진 클래스)
- 빈을 등록하는 방법과 관리 or 관계에 대해 프레임웍에서 임무가 주어짐
- XML 파일 or 애노테이션을 통한 규칙을 정하고 그 규칙에 따라 IoC컨테이너가 활용
- BeanDefinition 타입의 오브젝트(Interface)와 같은 것으로 관리

1.2 IoC/DI를 위한 빈 설정 메타정보 작성

IoC 컨테이너의 가장 기본적인 역할은 코드를 대신해서 애플리케이션을 구성하는 오브젝트를 생성하고 관리하는 것이다. POJO로 만들어진 애플리케이션 클래스와 서비스 오브젝트들이 그 대상이다. 이렇게 컨테이너가 빈이라고 불리는 오브젝트를 만들고 관리하게 하는 것이 DI를 비롯한 스프링의 주요 기술을 적용하기 위한 전제조건이다.

그렇다면 컨테이너는 어떻게 자신이 만들 오브젝트가 무엇인지 알 수 있을까? 1장 앞부분에서 살펴봤듯이 컨테이너는 빈 설정 메타정보를 통해 빈의 클래스와 이름을 제공받는다. 빈을 만들기 위한 설정 메타정보는 파일이나 애노테이션 같은 리소스로부터 전용 리더를 통해 읽혀서 BeanDefinition 타입의 오브젝트로 변환된다. 이 BeanDefinition 정보를 IoC 컨테이너가 활용하는 것이다.

IoC 컨테이너가 직접 사용하는 BeanDefinition은 순수한 오브젝트로 표현되는 빈 생성 정보다. 따라서 그 정보가 담긴 리소스의 종류와 작성 방식에 독립적이다. 적절한 리더나 BeanDefinition 생성기를 사용할 수만 있다면 빈 설정 메타정보를 담은 소스는 어떤 식으로 만들어도 상관없다. 그림 1-8은 다양한 소스로부터 생성되는 BeanDefinition 메타정보가 컨테이너에 의해 사용되는 방식을 나타낸다.

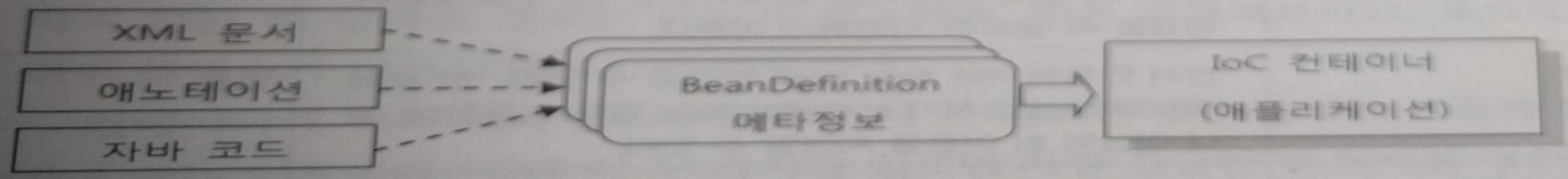


그림 1-8 컨테이너가 활용하는 빈 설정 메타정보

먼저 IoC 컨테이너가 사용하는 빈 생성을 위한 메타정보에 담긴 내용이 어떤 것인지를

빈 등록 방법

- XML 문서
  - Bean 태그 사용

- 애노테이션
  - @Component (빈 스캐너를 통해 등록) – 자동인식
  - @Configuration 클래스 @Bean 메소드 – 자바코드

빈 의존관계 설정 방법

- DI(의존성 주입)에 대한 메타정보를 등록

- XML (Bean) 태그
- 스키마를 가진 전용 태그
- 애노테이션
- 자바코드에 의한 직접적인 DI

- 의존 관계 주입이란 Lazy Loding / Dynamic 생성과 같다
- 의존성에 대한 관계를 위의 방법으로 설정한 후 해당 생성자 /프로퍼티에서 의존관계에 명시된 Bean 객체가 주입되면 자동으로 빈이 생성된다 즉 DI할 빈의 ID와 이름같은걸로 연결짓는 것이다

프레임 워크란?

- 개발프레임 워크
  - 공통점
  - 관리와 통제
- 통합개발환경
  - 프레임워크 개발자

- 과거의 개발
  - HW중심
  - 기능 중심

- 프로그래밍 언어의 활용(왜 언어는 도구인가?)에 대한 가설과 검증