IoC(Inversion of Control) – 역제어, 제어역전

|  |  |
| --- | --- |
| 기존 방식 | IoC 사용 시 |
| 자바객체를 생성하고 서로간의 의존관계를 연결시키는 작업에 대한 제어권이 개발되는 어플리케이션에 있다. | 제어권이 컨테이너에게 넘어가 객체에 대한 생명주기를 컨테이너가 전담함 |

\* 문제 제기

컴포넌트간의 결합도가 높아서 컴포넌트의 확장 및 재사용성이 어려운 문제가 발생

\* 스프링이 IoC를 사용하는 이유

컴포넌트간의 결합도가 낮아서 컴포넌트의 재사용이 가능하고, 확장도 쉽고, 컴포넌트 간의 의존관계 변경도 쉽게 된다.

스프링 ioc컨테이너에는 – **어플리케이션 컨텍스트**(빈 팩토리를 포함하여 더 많은 기능을 가지고 있음), 빈 팩토리(메이지 로딩-늦은 인스턴스화)가 있다.

Spring은 XLM 기반의 Bean 속성을 정의하기 위해 <property />와 <constructor-arg />내의 하위태그를 지원한다.

이러한 하위태그는 Collection유형의 Bean 속성을 정의하는 태그를 제공함

|  |  |
| --- | --- |
| <value> | 사람이 읽을 수 있는 형태의 문자열 |
| <ref> | 다른 빈을 참조하는 방법의 하나 |
| <ref bean=  ”someBean”/> | 동일한 xml 파일 내에 정의된 다른 Bean 참조 |

\* AnnotationConfigApplicationContext

>> @Bean(메소드 앞에)

@Configuration(클래스 선언 앞에)

@Override

@Controller

@Autowried

@Service

@Repository

\* web.xml – DD – 배치서술자

deployement descriptor

\* log4j에 대한 선언을 <context-param>에 한 이유

로그 출력이 공통된 관심사이기 때문! (세팅 후 중간에 바뀔 일이 거의 없음)

getter/setter

>> 더 이상 req,res에 종속적인 코딩은 하지 않는다.

상속-HttpServlet

스프링 – DispatcherServlet

환경설정하는 방법

1..xml

spring-servlet.xml (모든 업무의 컨트롤 클래스가 정의되어 있음)

Spring0service.xmo (XXXLogic클래스의 정의)

Spring-data.xm l(JDBC, myBatis, Dao)

2. @annotation-java코드 작성

SI – spring boot – gradle/kotlin

SM – spring4, spring5 – maven

\* DI, IoC 🡪 MVN 🡪 굳이 왜 XXXLogic.java로 관리?

>> AOP!!!!!!!!

- AOP: 수평관계, 공통된 관심사를 분리하는 기술(API)

- OOP: 상하관계 위아래, 상속

XXXController - @Controller

XXXLogic - @Service

XXXDao - @Service(직관적, 반자동), @Repository(추상적처리 – 인터페이스 방식 사용)

HikariCP 커넥션풀 API 사용 설정 >> db연결

@Override대상 – 원형을 바꿀 수 없다.

리턴타입, 파라미터타입, 파라미터 개수, 메소드 이름까지!!! >> 결합도가 높음

\*\* 게시판 이관작업 하는 데에 필요한 어노테이션

**@Controller – 컨트롤 계층을 조립**

@RequestMapping(“/board/\*”) – 클래스 선언 앞에 올 수 있다. – 작업 폴더명을 미리 빼놓기!

위치는? 클래스 앞! 멀티라인으로도 작성 가능

@GetMapping(“/”) or @PostMapping(“/”) 단위테스트가 어려움.

**@Service – 모델 계층을 조립**

@Repository와 차이점은 스프링에서 제공되는 DataAccessException을 지원받음 (SQLException 종류)

생략가능한 어노테이션

[/WEB-INF/views/testList.jsp] – 404에러

- @RequestMapping : 업무명에 대한 공통 이름 등록

- @GetMapping : 실제 메소드와 매핑되는 어노테이션

- @PostMapping : 위와 동일(post방식)

\* 컨트롤 계층에서 화면 출력을 위해 주입되는 scope가 request인 클래스이름은 무엇일까 >> **Model**

\*\* POJO > SPRING 이관작업 \*\*

트랜잭션 처리 대상이 아님.

공통점: SELECT, forward, request – 유지문제

ModelAndView사용 – 리턴타입으로 처리

Model사용 – 파라미터로 주입

차이점: 출력페이지가 다르다

> 그래서 메소드 이름을 분리!

**boardList** – list.jsp

**boardDetail** – read.jsp

-------------------------------------------------------------

**boardInsert** > 커밋대상 >

(boardList.sp 경유 > forward) > list.j네

Insert into 테이블명(컬럼명, 컬럼명, …) values(?, ?, ..)

**boardUpdate** (insert와 동일)

Update 테이블명 set 컬럼명 = 값

Where 컬럼명 = 값

**boardDelete** (insert와 동일)

delete from 테이블명

where 컬럼명 = ?

? 자리는 파라미터!

ModelAndView XXX(req,res) >> Model, ModelMap 변화!

\*\* 스프링은 웹 개발 전용이 아니다.

(모바일 개발, 로컬 어플리케이션 등 가능하다. 왜? DI, IOC 적용되기 때문에)

- 웹 개발에서는 @Controller @RestController @Service @Repository (interface - 코드양은 줄지만, 직관적이지 않다.-사용지양하자)