spring 구동순서

사전지식

Spring Framework의 특징

아래에는 Spring Framework의 여러가지 특징을 대략적으로 소개했다. 각 특징의 자세한 내용은 추가적인 포스팅을 통해 집중적으로 다룰 계획이다.

- **IoC (Inversion of Control) : 제어의 역행**

  제어의 주체가 개발자가 아닌 프레임워크라는 뜻으로 때에 따라 프레임워크가 작성된 코드를 호출하는 기술. 객체의 생명주기의 관리까지 모든 객체에 대한 제어권을 프레임워크가 가진다.

- **DI (Dependency Injection) : 의존성 주입**

  의존성 객체를 개발자가 생성하지 않고 클래스를 Bean으로 등록해놓으면 Bean으로 등록된 객체를 프레임워크가 찾아서 알아서 주입해주는 기술이다. 이를 통해 모듈간의 결합도를 낮출 수 있다.

  - 의존성
    - A 객체에서 B 객체의 변수나 메소드를 사용해야 할 경우, A라는 객체 생성자에서 new B();를 해야한다. 이 때 A는 B에 의존한다고 볼 수 있다.
  - 의존성 주입
    - A라는 객체에서 B를 생성하는 것이 아니라 외부에서 생성된 B를 A에 주입함으로써 의존 관계를 없앤다.
  - 의존성 주입 방법
    - XML 방법
      - 생성자<constructor-arg> 태그 + ref 속성
      - 속성 <property> + name 속성
    - Annotation 방법
      - @Autowired / @Resource

- **AOP (Aspect Oriented Programming) : 관점 지향 프로그래밍**

  각 코드마다 공통된 관심사를 분리하여 모듈화하는 프로그래밍 기법.

  객체지향적으로 프로그래밍을 했음에도 로그, 트랜잭션, 성능확인 등 공통적인 관심사가 중복되는 문제점을 해결하기 위해 프록시 패턴을 사용하여 코드를 분리하여 관리하는 기술이다.

- **PSA (Portable Service Abstraction) : 추상화를 통해 코드가 간결해진다.**

  POJO 프로그래밍을 지원하기 위해 다양하게 구현되어있는 인터페이스를 같은 방식으로 사용하도록 중간에 인터페이스 어댑터 역할을 해주는 레이어를 추가하는 방법이다. 실제로 일일이 구현을 해줘야 되는

것들을 스프링이 다 구현해놓고 어노테이션으로 만들어 놓아 어노테이션만 선언하면 내부적으로 다 동작이 되도록 추상화 한 것이다.

- **POJO : Plane Old Java Object (순수 자바 객체)**

  EJB 등에서 사용되는 Java Bean이 아닌 멤버 변수와 Getter와 Setter로 구성된 가장 순수한 형태의 기본 클래스이다.

  `"POJO는 getter 와 setter를 가진 단순한 자바 오브젝트"가 아니라 "getter와 setter를 가진 가장 단순한 자바 오브젝트는 POJO"다.`

POJO

- 스프링의 특징 중 하나
- 평범한 옛날 자바 객체
- Not POJO = Servlet의 특징
  - javax.servlet, javax.servlet.http 패키지를 import해야 한다.
  - Servlet, Generic Servlet, HttpServlet 중 하나를 상속해야 한다.
  - 생명주기에 해당하는 메소드를 재정의(Overriding)한다. (반드시 Servlet에서 요구하는 규칙에 맞게 클래스를 만들어야 실행됨 doGet, doPost 등)
  - 예전에는 자바로 웹애플리케이션을 개발하려면 Servlet클래스를 상속받아 구현해야 했는데 이 Servlet이 POJO가 아니었다. 그런데 스프링을 사용하면 POJO만으로 웹 애플리케이션을 구축할 수 있게 되었다. Servlet클래스를 모두 추상화하여 라이브러리로 들어갔기 때문에. XML이나 다른 설정으로 Servlet을 이요하면 된다.

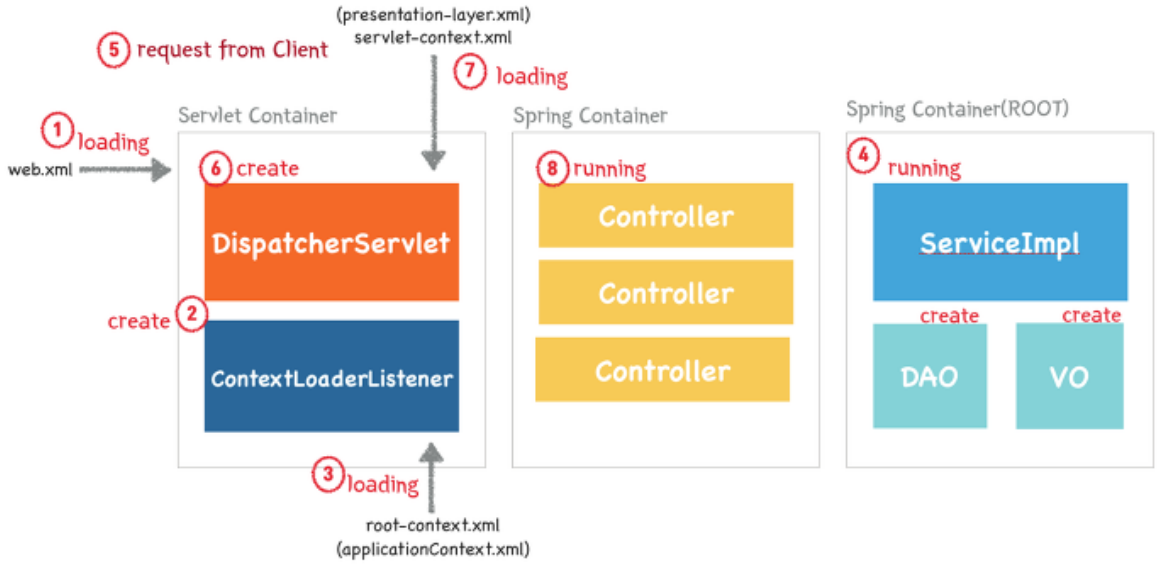
- Spring에서는 xml과 같은 설정을 통해 Servlet 사용
- Servlet이 복잡해서 시작한 것이 Spring

컨테이너 2 종류

- BeanFactory
  - 스프링 설정파일에 등록된 <bean> 객체를 생성하고 관리.
  - 컨테이너가 구동될 때 객체를 생성하는 것이 아니라 클라이언트로부터의 요청에 의해서만 객체를 생성 (Lazy Loading 방식)
  - > 일반적인 스프링 프로젝트에서 사용할 일이 없음
- ApplicationContext
  - BeanFactory가 제공하는 <bean> 객체 관리 기능 외에도 트랜잭션 관리나 메시지 기반의 다국어 처리 기능 지원
  - 컨테이너가 구동되는 시점에 <bean>에 등록되어 있는 클래스들을 객체화 하는 즉시로딩(Pre-Loading 방식)
    - <bean> option에 lazy-init="true"를 써서 클라이언트가 요청하는 시점에 구동할 수도 있음
  - ApplicationContext의 구현 클래스는 매우 다양하지만, 두 개만 알고 있으면 됨

    - GenericXmlApplicationContext : 대표적으로 ApplicationContext를 구현한 클래스, 파일 시스템이나 클래스 경로에 있는 xml 설정 파일을 로딩하여 구동하는 컨테이너
    - XmlWebApplicationContext : 웹 기반의 스프링 애플리케이션을 개발할 때 사용하는 컨테이너 (직접 개발 안하는 컨테이너)
- 정리 : 스프링 컨테이너는 bean 저장소에 해당하는 xml 설정파일을 참조하여 bean 생명주기 관리
- Servlet Container (Servlet Engine)
- 서블릿 실행
- 세션관리
- 네트워크 서비스
- 메시지 인코딩 디코딩
- 서블릿 생명주기 관리

Spring 실행순서



1. 웹 어플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml이 Loading

ServletContainer (ex: 톰캣 서버) -> URL 확인 -> 요청을 처리할 서블릿 찾아 실행

web.xml : 각종 설정을 위한 파일

2. web.xml에 등록되어 있는 ContextLoaderListener (Java Class) 생성

서블릿 컨테이너가 파일을 읽어서 구동될 때, ContextLoaderListener 가 자동으로 메모리에 생성된다. (Pre-Loading)

ContextLoaderListener 클래스는 ServletContextListener 인터페이스를 구현하고 있으며, ApplicationContext를 생성하는 역할을 수행한다.

ServletContextListener 는 웹 어플리케이션의 시작, 종료 이벤트에 대한 이벤트 리스너이다. 핸들러 메서드에서는 ServletContext에 대한 참조를 얻을 수 있다.

💡 리스너 등록 및 처리 과정을 이해하기 쉽게 설명하자면 다음과 같다.

1. 개발자는 처리하고싶은 이벤트에 관련된 이벤트 리스너 인터페이스를 구현하고 web.xml에 설정을 등록한다.
2. 톰캣과 같은 서블릿 컨테이너는 web.xml 설정을 참고하여 개발자가 구현해둔 리스너 객체를 생성하고 관리하게 된다.
3. 특정 이벤트가 발생했을때 그 이벤트와 관련된 리스너가 있는지 확인하여 해당 리스너에게 알려줍니다. 예를 들어 A라는 이벤트가 발생하면 자신이 생성하여 관리하던 리스너 중 A 이벤트와 관련된 리스너에게 알려주는것이다.
4. 리스너에는 개발자가 구현해둔 이벤트 핸들링 메서드가 있으므로 해당 메서드가 실행된다. (이벤트 처리)

ContextLoaderListener 클래스는 Servlet의 생명주기를 관리해준다.

Servlet을 사용하는 시점에 서블릿 컨텍스트에 ApplicationContext 등록, Servlet이 종료되는 시점에 ApplicationContext 삭제

3. 생성된 ContextLoaderListener는 root-context.xml을 Loading

- ContextLoaderListener 객체는 src/main/resources 소스 폴더에 있는 applicationContext.xml 파일을 로딩하여 스프링 컨테이너를 구동하는데 이를 Root 컨테이너 라고 한다.
- 원래는 ContextLoaderListener가 WEB-INF 밑에 있는 파일을 먼저 로딩하도록 되어있으나 유지보수 상의 관계로 src/main/resources 에 파일을 넣어 놓는 경우가 많다. 때문에 src/main/resources 에 있는 파일을 가져다 로딩시키려면 web.xml에 -param으로 설정해서 사용한다.

4. root-context.xml에 등록되어 있는 Spring Container가 구동

root-context.xml에는 주로 view 지원을 제외한 공통 bean을 설정한다. (web과 관련된 bean들은 등록해주지 않음)

예시로 spring properties 파일을 로컬과 서버용으로 구분지을 때 여기서 property value를 설정해준다.

(검색해보니 databaseDataSource, repository 설정을 주로함)

5. 클라이언트로부터 웹 어플리케이션 요청이 옴

최초의 클라이언트 요청에 의해 DispatcherServlet 생성

6. DispatcherServlet이 생성됨

DispatcherServlet 객체는 WEB-INF/config 폴더에 있는 servelt-context.xml 파일을 로딩하여 두번째 스프링 컨테이너를 구동한다. 이 두 번째 스프링 컨테이너가 Controller 객체를 메모리에 생성한다.

DispatcherServlet은 FrontController의 역할을 수행한다. 클라이언트로부터 요청 온 메시지를 분석하여 알맞은 PageController에게 전달하고 응답을 받아 요청에 따른 응답을 어떻게 할지 결정한다.

7. DispatcherServlet은 servlet-context.xml을 Loading

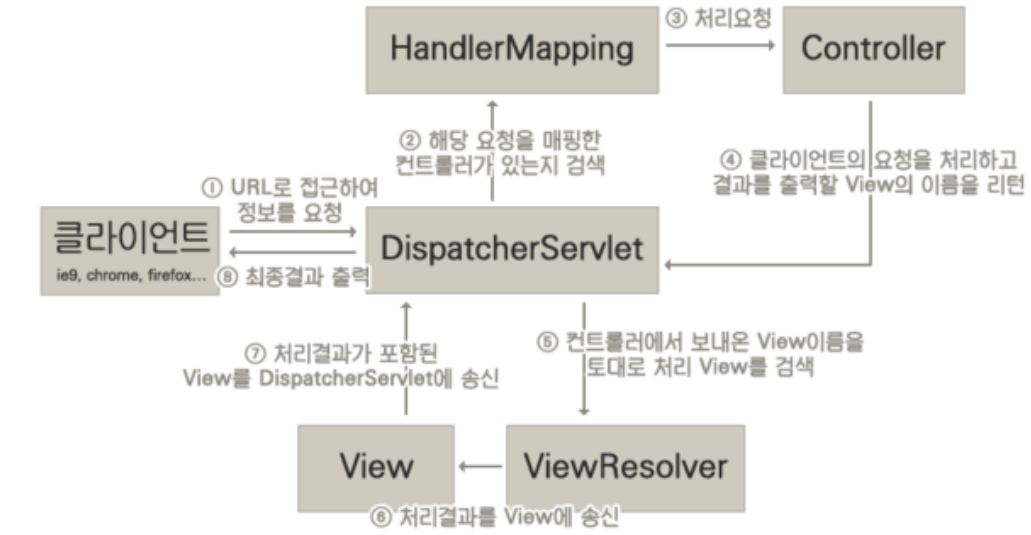
web.xml을 보면 DispatcherServlet이 servlet-context.xml을 가리키고 있다.

```
<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

  <param-name>contextConfigLocation</param-name>

  <param-value>

        classpath:/conf/spring/appServlet/servlet-context.xml

 </param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>
```

servlet-context.xml을 보면 어노테이션을 스캔하여 bean 객체로 등록함을 볼 수 있다.

```
<annotation-driven /> --어노테이션을 사용함을 선언


<mvc:annotation-driven> --

    <mvc:message-converters register-defaults="true">

        <beans:bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">

            <beans:property name="objectMapper">

                <beans:bean class="com.github.miemiedev.mybatis.paginator.jackson2.PageListJsonMapper" />

            </beans:property>

        </beans:bean>

    </mvc:message-converters>

</mvc:annotation-driven>


<context:component-scan base-package="com.company.*" />

-- 베이스 패키지 하위의 모든 어노테이션을 스캔해서 빈으로 등록
```

8. 구동순서



① 클라이언트가 해당 어플리케이션에 접근하면 접근한 URL 요청을 DispatcherServlet이 가로챔

```
DEBUG org.springframework.web.servlet.DispatcherServlet - DispatcherServlet with name 'appServlet' processing POST request for [/project_domain/main]
```

② RequestMappingHandlerMapping이 해당 요청을 처리할 컨트롤러를 찾음

```
DEBUG org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Looking up handler method for path /main
```

Request Mapping Handler : @Controller("/url") // 어노테이션으로 매핑

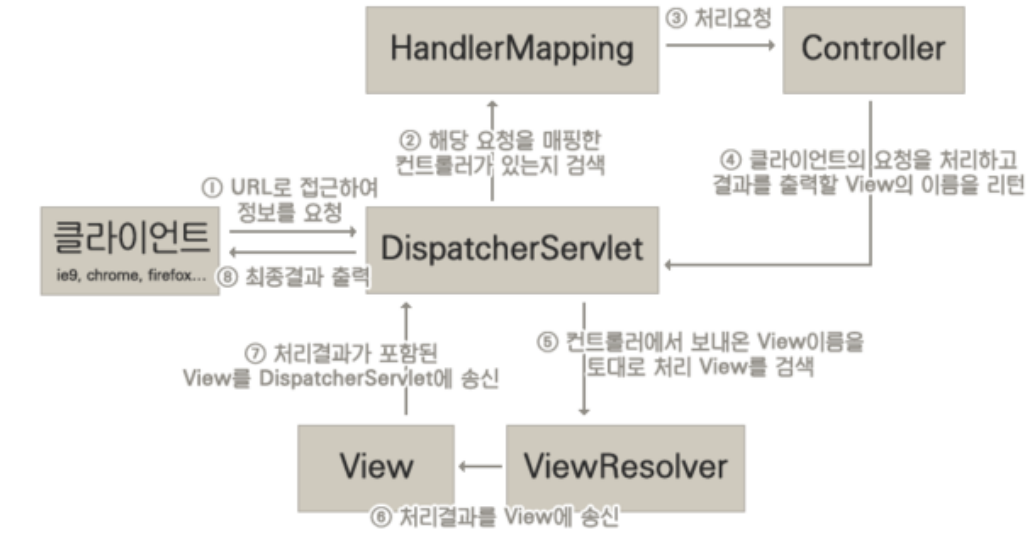
BeanNameMapping Handler : **<class-url>**</class- // 클래스 이름으로 매핑

③ DefaultListableBeanFactory 가 mainContoller 를 쓰면 된다고 알려줌

```
DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'mainController'
```

④ RequestResponseBodyMethodProcessor가 servlet-context.xml 에 선언해 놓은 MessageConverter을 이용하여 요청 바디(파라미터)를 읽음

```
DEBUG org.springframework.web.servlet.mvc.method.annotation.RequestResponseBodyMethodProcessor - Reading [java.util.Map<java.lang.String, java.lang.Object>] as "application/json;charset=UTF-8" using [org.springframework.http.converter.json.MappingJackson2HttpMessageConverter@11eb521a]
```



HttpMessageConverter :

자바 객체와 HTTP 요청/응답 몸체 사이의 변환을 처리

HttpMessageConverter 종류

구현 클래스	설 명
ByteArrayHttpMessageConverter(*)	HTTP 메시지와 byte 배열 사이의 변환을 처리한다. 컨텐츠 타입은 application/octet-stream 이다.
StringHttpMessageConverter(*)	HTTP 메시지와 String 사이의 변환을 처리한다. 컨텐츠 타입은 text/plain;charset=ISO-8859-1이다.
FormHttpMessageConverter(*)	HTML 폼 데이터를 MultiValueMap으로 전달받을 때 사용된다. 지원하는 컨텐 츠 타입은 application-x-www-form-urlencorded이다.
SourceHttpMessageConverter(*)	HTTP 메시지와 javax.xml.transform.Source 사이 변환을 처리한다. 컨텐츠 타 입은 application/xml 또는 text/xml이다.
MarshallingHttpMessageConverter(*)	스프링의 Marshaller와 unMarshaller를 이용해서 XML HTTP 메시지와 객체 사 이의 변환을 처리한다. 컨텐츠 타입은 application/xml 또는 text/xml이다.
MappingJacksonHttpMessageConverter(*)	Jackson 라이브러리를 이용해서 JSON HTTP 메시지와 객체 사이의 변환을 처 리한다. 컨텐츠 타입은 applicaion/json이다.

서비스 진입...

⑤ DataSourceTransactionManager 로 DB 접속

```
DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Acquired Connection [jdbc:...DEBUG org.mybatis.spring.transaction.SpringManagedTransaction - JDBC Connection...
```

⑥ 응답결과 받아다 파라미터에 써줌

```
DEBUG org.springframework.web.servlet.mvc.method.annotation.RequestResponseBodyMethodProcessor - Written...
```

DispatcherServlet 다시 등장

⑦ 컨트롤러에서 view를 리턴하면 ViewResolver가 먼저 받아 해당 view가 존재하는지 검색

```
DEBUG org.springframework.web.servlet.DispatcherServlet - Null ModelAndView returned to DispatcherServlet with name 'appServlet': assuming HandlerAdapter completed request handling
```

⑧ DispatcherServlet은 ViewResolver를 통해 접두사와 접미사가 붙은 JSP 파일의 이름과 경로를 리턴받아 최종적으로 JSP를 실행.

view에 결과를 보낸 후 DispatcherServlet은 최종 결과를 클라이언트에 전송

DEBUG org.springframework.web.servlet.DispatcherServlet - Successfully completed request

(핸들러 매핑이나 리졸버는 대략적인 흐름만 알고 있다가 context에 등록시키면 됨)

참고용