[Spring Framework 실행순서](https://javannspring.tistory.com/231)

**사전지식**

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)
* 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

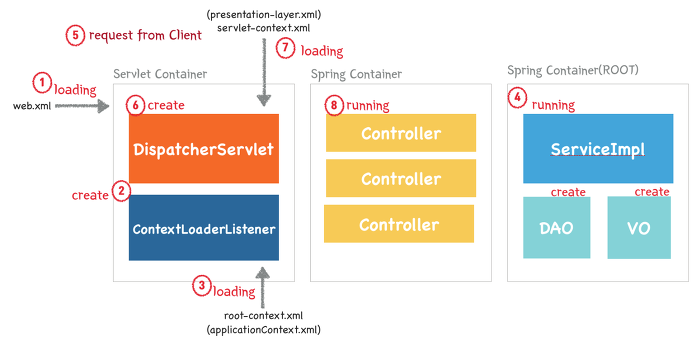
-세션관리

-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

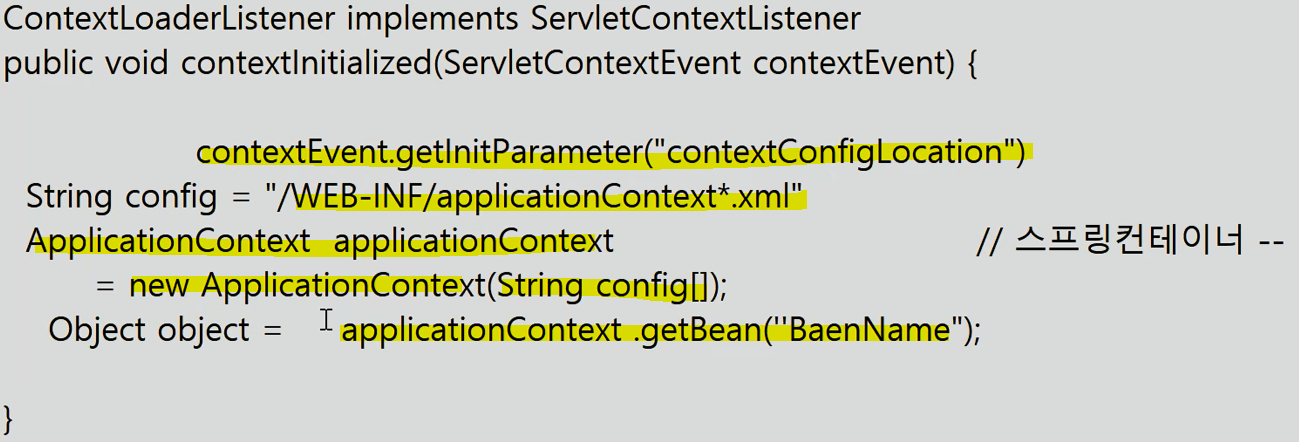
2. web.xml에 등록되어 있는 ContextLoaderListener (Java Class) 생성

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

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

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(저장소) 설정**을 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'

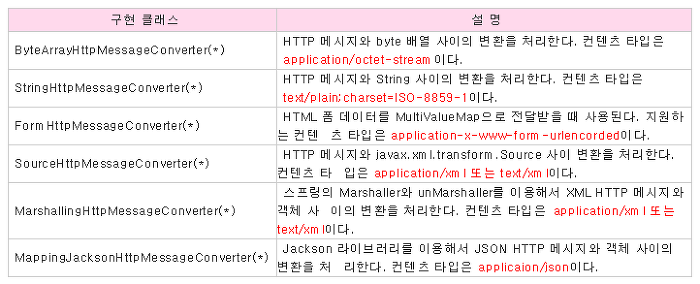
④ RequestResponseBodyMethodProcessor가 servlet-context.xml 에 선언해놓은 MessageConverter을 이용하여 요청 바디(파라미터)를 읽음

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



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

