### Servlet, Servlet Context, Application Context 등 개념정리

### Servlet

### public abstract class HttpServlet extends GenericServlet {

### *//...*

### protected void doGet(HttpServletRequest req, HttpServletResponse resp){...}

### protected void doPost(HttpServletRequest req, HttpServletResponse resp){...}

### *//...*

### }

### 자바를 사용하여 웹 페이지를 동적으로 생성하는 서버측 프로그램 혹은 그 사양입니다.

### 쉽게 말해, "웹 서버 프로그래밍을 하기 위한 사양을 갖춘 자바 코드" 라고 할 수 있습니다.

### HttpServlet 클래스를 상속한 클래스

### Servlet은 Servlet Container에 의해 관리, 실행됩니다. HTTP Server + Servlet Container가 웹 서버 역할에 필요한 대부분을 구현해두었고, 개발자는 Servlet을 만들어 HTTP 요청을 받아 처리하는 부분을 구현하는 것입니다.

### 메서드를 참고하면 알 수 있듯 요청(Request)과 응답(Response) 즉, Http 웹 서버 기능 동작이 가능합니다.

### 톰캣(Tomcat)

### 웹 애플리케이션 서버(WAS)중 하나로 Servlet Container, Servlet Engine이라고 표현할 수 있으며 자바 웹 프로그래머가 작성한 Servlet을 관리합니다.

### Servlet을 관리한다는 말은 클라이언트가 어떤 요청(Request)을 했을 때, 어떤 Servlet을 실행할 것인지 제어해준다는 것입니다.

### 톰캣은 Servlet을 관리해주는 주체이기 때문에 아무 클래스가 아니라 Servlet(HttpServlet 클래스를 상속한 클래스)이어야 합니다.

### web.xml

### WAS(e.g. tomcat)는 Servlet을 생성하고 어떤 Servlet이 어떤 요청을 담당할 것인지(mapping), 어떤 요청이 인증과정을 거칠 것인지 등의 제어 기능을 지원해줍니다. 그러려면 WAS에게 Servlet에 대한 정보를 줘야하는데 이때 쓰이는 파일이 web.xml(Deployment Descriptor)입니다.

### servlet3.0부터는 web.xml에서만 Servlet에 대해 정의하지 않고, 자바 소스 설정(java config)으로도 가능합니다. 요즘은 학습할 때 빼고는 대부분 자바 소스 설정으로 하는 것으로 추정됩니다.

### <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

### xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

### xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

### http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

### version="3.1">

### <servlet>

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

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

### </servlet>

### <servlet-mapping>

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

### <url-pattern>/</url-pattern>

### </servlet-mapping>

### <filter>

### <filter-name>encodingFilter</filter-name>

### <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

### <init-param>

### <param-name>encoding</param-name>

### <param-value>utf-8</param-value>

### </init-param>

### </filter>

### <filter-mapping>

### <filter-name>encodingFilter</filter-name>

### <url-pattern>/\*</url-pattern>

### </filter-mapping>

### </web-app>

### DispatcherServlet

### dispatch는 '급파하다', '신속히 보내다', '특파하다', ... 뭐 이런 뜻이 있습니다. dispatcher는 '항공기 운항 관리자(관제사)', '배차 담당자', ... 이런 뜻이 있습니다.

### Servlet에 대입해보면, "Servlet Container(e.g. tomcat)으로부터 들어오는 요청을 관제하는 컨트롤러다" 라고 할 수 있습니다. [Spring MVC](https://jeong-pro.tistory.com/96)에서 요청을 받는 부분이라고 할 수 있습니다.

### Servlet Container(e.g. tomcat)에 여러 매핑 정보를 가진 여러 Servlet을 생성하고 관리할 수도 있지만 일반적인 경우(?)에는 Servlet Container에는 DispatcherServlet만 등록해놓고 DispatcherServlet이 HandlerMapping을 통해 적절한(개발자가 만든) Controller로 매핑하도록 하는 것으로 알고 있습니다.

### Servlet Filter

### Servlet 실행 전, 후에 어떤 작업을 하고자할 때 Servlet Filter 를 사용한다.

### Interceptor를 사용할 수 있겠지만 차이점은 실행시점(handler전, 후)에 차이가 있습니다.

### Filter 는 Servlet Container에 등록하고 Interceptor는 스프링 컨테이너에 등록합니다.

### javax.servlet.Filter 인터페이스의 구현체입니다.

### Servlet Context

### Servlet 단위로 생성되는 Context입니다.

### Servlet Container(e.g. tomcat)에 DispatcherServlet과 같은 servlet을 등록하면 해당 servlet이 갖는 하나의 작은 컨테이너 역할을 하는 객체입니다.

### 스프링을 이용하는 경우, 스프링 컨테이너(Application Context)를 부모 Context로 사용합니다.

### Application Context와 Servlet Context에 같은 id로 된 Bean이 있으면 ServletContext에 있는 Bean을 우선 사용합니다. (Bean을 찾는 순서가 Servlet에서 ServletContext를 확인한 후에 부모인 ApplicationContext를 확인하기 때문입니다.

### Application Context

### Root Context이자 스프링에 의해 생성되는 Bean에 대한 Spring IoC Container입니다.

### BeanFactory를 상속받는 Context

### 여러 Servlet에서 공통으로 사용할 Bean을 등록하는 Context입니다.

### @Transactional 으로 트랜잭션을 이용해야할 때 ApplicationContext에 있는 Service에서만 트랜잭션이 정상 작동합니다.

### WebApplicationInitializer

### public interface WebApplicationInitializer{

### void onStartup(ServletContext servletContext) throws ServletException;

### *//onStartup()메소드의 파라미터인 ServletContext는 Servlet Container를 가리킵니다.*

### }

### Servlet Context를 프로그래밍적으로 설정하기 위한 인터페이스입니다. (web.xml을 대체하기 위함)

### 스프링에 ServletContainerInitializer 를 구현한 클래스(SpringServletContainerInitializer)가 있고 그 클래스가 WebApplicationInitializer 인터페이스를 구현한 클래스를 찾아 초기화 작업을 위임하도록 구현해놨습니다.

### 설정하는 방식마다 차이가 있어서 좀 눈여겨봐야할 필요가 있고, 추가되는 내용을 정리해야할 필요가 있습니다...

### ContextLoaderListener

### Servlet Container(e.g. tomcat)에 루트 웹 애플리케이션 컨텍스트(Application Context)를 등록하는 방법을 제공합니다.

### Servlet Container의 시작과 종료 시에 발생하는 이벤트를 처리하는 리스너를 등록하기 위해 ServletContextListener 인터페이스를 구현한 리스너를 사용하는데 그 구현체가 바로 ContextLoaderListener 입니다.

### Application Context에 대한 실제 초기화 작업을 수행합니다.

### 이 리스너(Listener)만 등록하면 자동으로 디폴트 루트 애플리케이션 컨텍스트를 생성해줍니다.

### 디폴트 설정

### XmlWebApplicationContext

### XML 설정파일 위치 : /WEB-INF/applicationContext.xml

### RequestContextListener

### 현재 스레드에 요청(Request)을 노출하는 Servlet Listener입니다.

### RequestContextListener 를 등록하면 LocaleContextHolder, RequestContextHolder를 통해서 HttpServletRequest에 접근할 수 있게 합니다.

### AnnotationConfigWepApplicationContext

### Component 클래스를 입력값으로 받는 WebApplicationContext 인터페이스의 구현체입니다.

### Component 클래스는 @Configuration, @Component, @Inject annotation을 사용하는 클래스를 포함합니다.

### 패키지 경로를 스캔하여 컴포넌트를 Context에 등록합니다.

### AnnotationConfigWebApplicationContext 인스턴스는 스프링에서 DispatcherServlet이나 ContextLoaderListener에 주입되는 경우에 많이 사용합니다. (web.xml을 대체하여 WebApplicationInitializer 를 이용한 자바 코드 기반 설정을 할 때)

### XmlWebApplicationContext와 달리 디폴트로 ConfigurationLocation을 지정하지 않습니다. 따라서 ContextLoader에 대한 context-param= 'contextConfigLocation' 또는 Servlet에 대한 init-param = 'contextConfigLocation'을 반드시 설정해야합니다.

### 참고사이트

### <https://victorydntmd.tistory.com/148>

### <https://jaehun2841.github.io/2018/10/21/2018-10-21-spring-context/>

### <https://www.baeldung.com/spring-web-contexts>