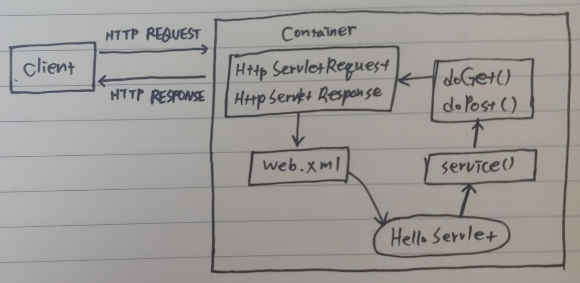
**서블릿 컨테이너와 스프링 컨테이너**

<https://minwan1.github.io/2017/10/08/2017-10-08-Spring-Container,Servlet-Container/>

<https://minwan1.github.io/2018/11/21/2018-11-21-jsp-springboot-%EB%8F%99%EC%9E%91%EA%B3%BC%EC%A0%95/>

서블릿 컨테이너는 개발자가 웹서버와 통신하기 위하여 소켓을 생성하고, 특정 포트에 리스닝하고, 스트림을 생성하는 등의 복잡한 일들을 할 필요가 없게 해준다. **컨테이너는 servlet의 생성부터 소멸까지의 일련의 과정(Life Cycle)을 관리한다. 서블릿 컨테이너는 요청이 들어올때마다 새로운 자바 스레드를 만든다.** 우리가 알고 잇는 대표적인 Servlet Container가 Tomcat이다. 톰캣같은 was가 java파일을 컴파일해서 Class로 만들고 메모리에 올려 servlet 객체를 만든다.

**서블릿 동작과정**



1. 사용자가 URL을 클릭하면 HTTP Request를 Servlet Container에 보낸다.

**2. Servlet Container는 HttpServletRequest, HttpServletResponse 두 객체를 생성한다.**

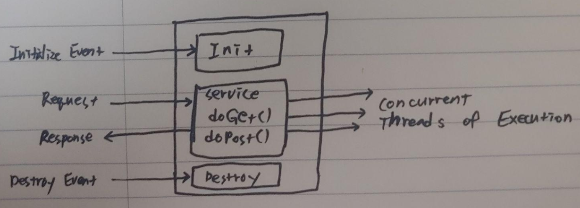
3. 사용자가 요청한 URL을 분석하여 어느 서블릿에 대한 요청인지 찾는다.

4. 컨테이너는 서블릿 service() 메소드를 호출하며, POST/GET 여부에 따라 doGet() 또는 doPost()가 호출된다.

**5. doGet() 이나 doPost() 메소드는 동적인 페이지를 생성한 후 HttpServletResponse 객체에 응답을 보낸다.**

6. 응답이 완료되면 HttpServletRequest, HttpServletResponse 두 객체를 소멸시킨다.

**서블릿 생명주기 (Servlet Lifecycle)**



init() - 서버가 켜질때 한번만 실행

service - 모든 유저들의 요청들을 받는다.

destroy() - 서버가 꺼질때 한번만 실행

**스프링 컨테이너 (Spring Container)**

먼저 **Spring Container**를 이해하기 위해서는 **IOC**와 **DI**를 이해해야 한다. **Spring Container는 Bean들의 생명주기를 관리**한다. Spring Container는 어플리케이션을 구성하는 **Bean들을 관리하기 위해 IoC를 사용**한다. Spring Container 종류에는 **BeanFactory**와 이를 상속한 **ApplicationContext**가 존재한다. 이 두개의 컨테이너로 의존성 주입된 빈들을 제어하고 관리할 수 있다. 아래는 스프링 웹애플리케이션 동작원리이다.

1. 웹 애플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml이 로딩된다. **(load-on-startup 으로 톰캣시작시 servlet 생성가능하도록 설정 가능)**

2. web.xml에 등록되어 있는 ContextLoaderListener(Java Class)가 생성된다. ContextLoaderListener 클래스는 ServletContextListener 인터페이스를 구현하고 있으며, **ApplicationContext를 생성하는 역할**을 수행한다.

3. 생성된 ContextLoaderListener는 applicationContext.xml을 로딩한다.

4. applicationContext.xml에 등록되어 있는 설정에 따라 Spring Container가 구동된다. **이때 개발자가 작성한 비즈니스 로직에 대한 부분과 DAO, VO 객체들이 생성된다.**

5. 클라이언트로부터 웹애플리케이션 요청이 온다.

6. DispatcherServlet(Servlet)이 생성된다. DispatcherServlet은 FrontController의 역할을 수행한다. **클라이언트로부터 요청 온 메시지를 분석하여 알맞은 PageController에게 전달하고 응답을 받아 요청에 따른 응답을 어떻게 할지 결정만 한다.** 실질적인 작업은 PageController에서 이뤄지기 때문이다. 이러한 클래스들을 HandlerMapping, ViewResolver 클래스라고 한다.

**7. DispatcherServlet은 servlet-context.xml(spring-mvc.xml)을 로딩한다.**

8. 두번째 Spring Container가 구동되면 응답에 맞는 PageController들이 동작한다. 이때 첫번째 Spring Container가 구동되면서 생성된 DAO, VO, ServiceImpl 클래스들과 협업하여 알맞은 작업을 처리하게 된다.

서블릿 컨테이너는 웹애플리케이션 서버중에서 HTTP 요청을 받아 처리하는 기초 역할을 맡고있다. **대부분의 웹프레임워크가 제공하는 기능은 서블릿 컨테이너 위에서 동작하는 서블릿, 필터, 이벤트 리스너 등을 적절하게 구현한 것이다. 따라서 사용자가 웹 프레임워크로 작성한 웹 애플리케이션은 결국 서블릿 컨테이너 위에서 동작한다.** 서블릿 컨테이너의 종류로는 아파치 톰캣, 제티 등이 서블릿 컨테이너로 널리 사용되고 있다.

 서블릿 컨테이너에 의해 프로그램이 실행되기 위해서는 표준 즉 Servlet Interface를 구현해줘야 한다. **사용자 정의 서블릿은 서블릿 컨테이너 내에 등록된 후 서블릿 컨테이너에 의해 생성, 호출, 소멸이 이루어진다.**

**때로는 서블릿은 자신의 상태 변경 시점을 알아내 적절한 리소스 획득/반환 등의 처리를 해야하므로 Servlet 인스터페이스에 init/destroy 메서드가 정의된다.** 다시 말해 서블릿 컨테이너는 서블릿의 생명주기에 따라 서블릿의 상태를 변경하면서 서블릿 인터페이스에 정의된 각 메서드를 불러준다.

 HTTP 프로토콜로 전달된 메시지는 서블릿 컨테이너에서 해석되고 재조합되어 웹프로그래머가 작성한 서블릿으로 전다로디는 과정을 거친다.

**1) Servlet**

Servlet은 서블릿 프로그램을 개발할때 반드시 구현해야 하는 메서드를 선언하고 있는 인터페이스이다. **이 표준을 구현해야 서블릿 컨테이너가 해당 서블릿을 실행 할수 있다.**

**2) GenericServlet**

GenericServlet은 Servlet 인터페이스를 상속하여 클라이언트-서버 환경에서 서버단의 애플리케이션으로서 필요한 기능을 구현한 추상클래스이다. service() 메서드를 제외한 모든 메서드를 재정의하여 적절한 기능으로 구현했다. GenericServlet 클래스를 상속하면 애플리케이션의 프로토콜에 따라 메서드 재정의 구문을 적용해야 한다.

**3) HttpServlet**

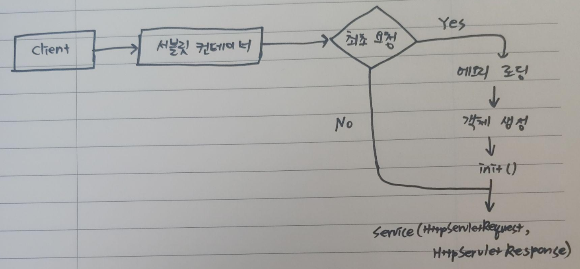
일반적으로 서블릿이라하면 거의 대부분 HttpServlet을 상속받은 서블릿을 의미한다. HttpServlet은 GenericServlet을 상속받았으며, GenericServlet의 유일한 추상 메서드인 service를 HTTP 프로토콜 요청 메서드에 적합하게 재구현해놨다.

 이미 DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE를 처리하는 메소드가 모두 정의되어 있다.

**서블릿 실행순서**

서블릿의 실행 순서는 개발자가 관리하는게 아닌 서블릿 컨테이너가 관리를 한다. 즉 서블릿에 의해 사용자가 정의한 서블릿 객체가 생성되고 호출되고 사라진다. 즉 이렇게 개발자가 아닌 프로그램에 의해 객체들이 관리되는 것을 IoC(Inversion of Control)이라고 한다.

다음은 서블릿 컨테이너의 생명주기를 도식화한 것이다.



- 서블릿 컨테이너는 클라이언트로부터 **처음 요청이 들어오면 현재 실행할 서블릿이 최초의 요청인지 판단하고 없으면 해당 서블릿을 새로 생성**한다. 이 작업은 최초 1회만 일어난다.

- init() 메소드는 해당 사용자 서블릿이 최초 생성되고 바로 호출되는 메소드이다.

- service() 메소드는 최초의 요청이든 2번째 요청이든 계속 호출되는 메소드이다.

 여기에서 서블릿컨테이너가 종료된다면 사용자 정의 HttpServlet의 destroy() 메서드가 호출될 것이다.