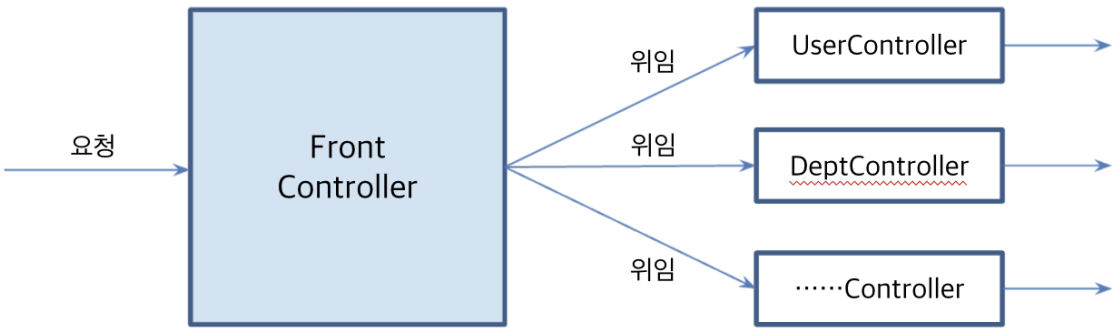
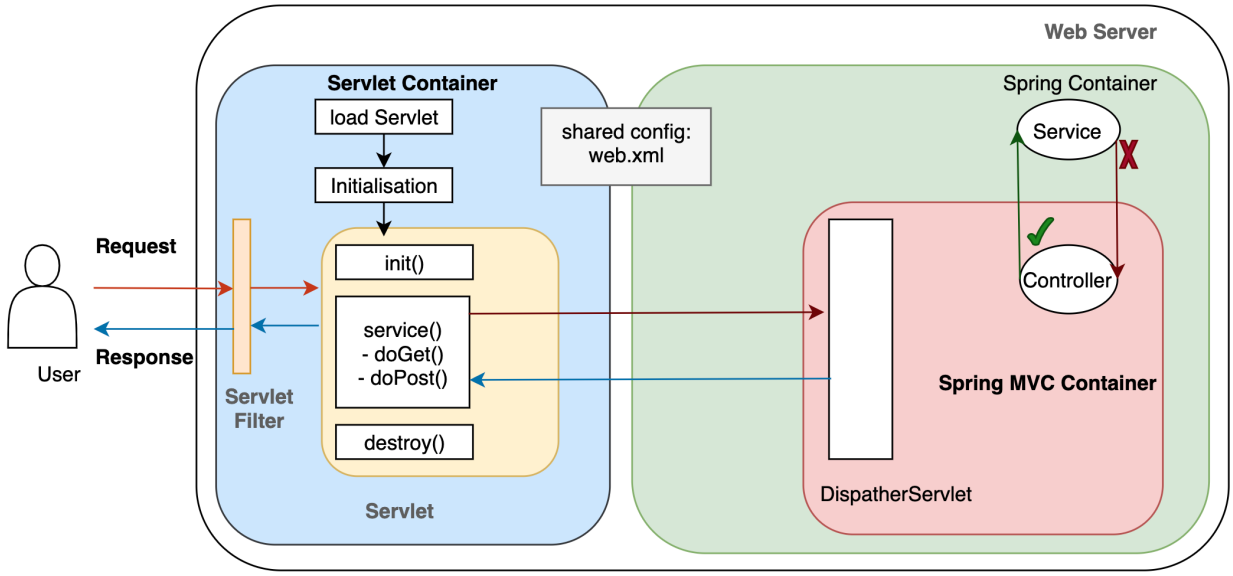
Spring MVC 1️⃣

DispatchServlet

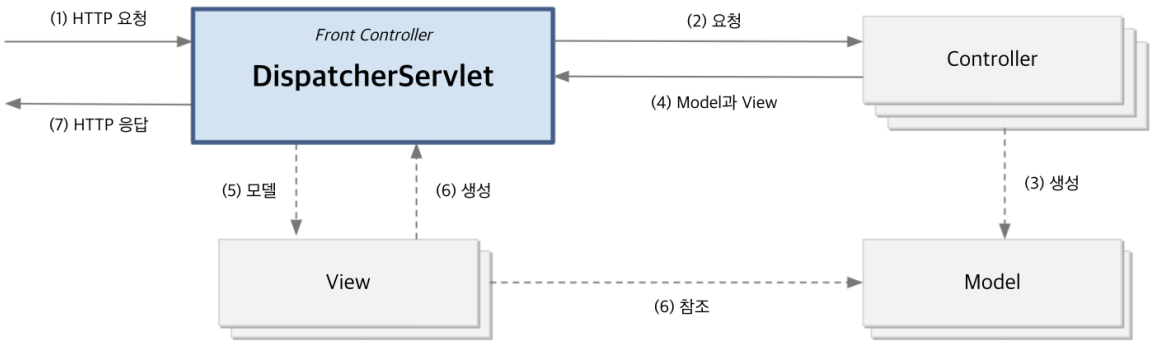
Front Controller Pattern





https://mossgreen.github.io/Servlet-Containers-and-Spring-Framework/

Spring MVC 처리 흐름



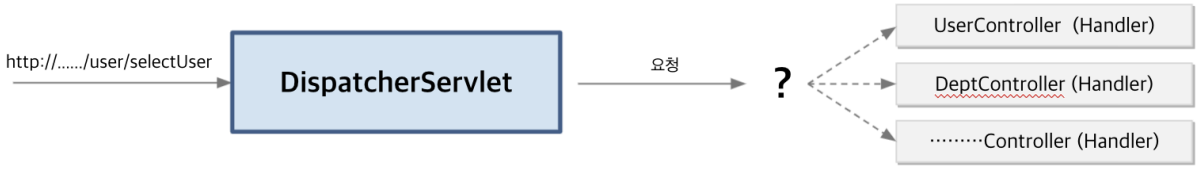
1. DispatcherServlet의 HTTP 요청 접수
2. DispatcherServlet에서 컨트롤러로 HTTP 요청 위임
3. 컨트롤러의 모델 생성과 정보 등록
4. 컨트롤러의 결과 리턴: 모델과 뷰
5. DispatcherServlet의 뷰 호출과 (6) 모델 참조
6. HTTP 응답 돌려주기

DispatcherServlet의 HTTP 요청 접수

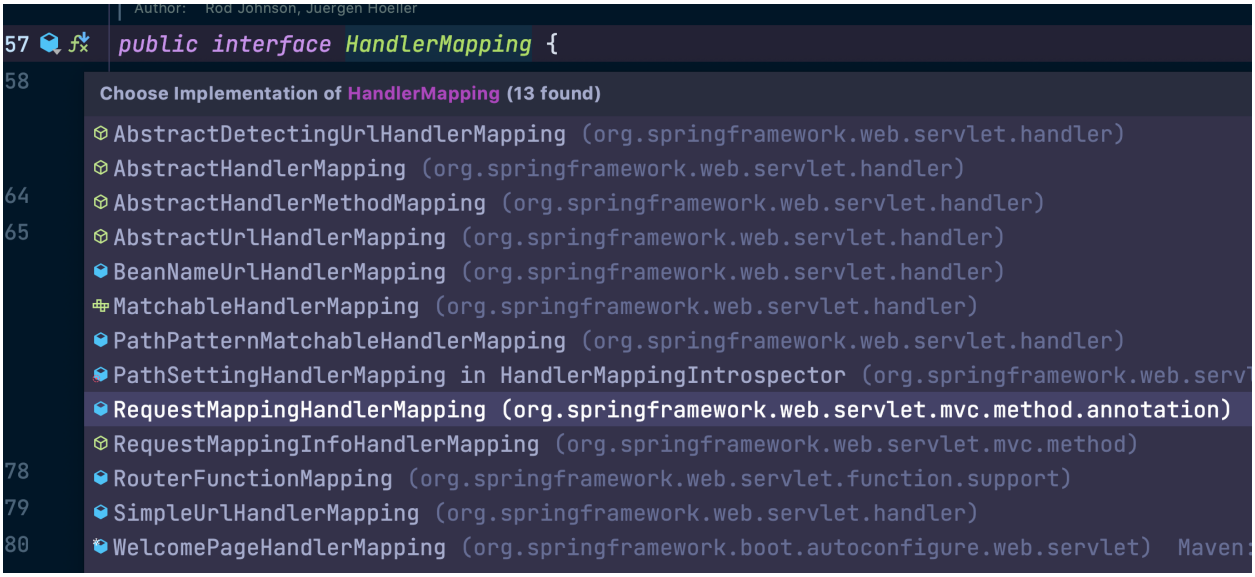
```
public class MyWebApplicationInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) {

        // Load Spring web application configuration
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplication
Context();
        context.register(AppConfig.class);

        // Create and register the DispatcherServlet
        DispatcherServlet servlet = new DispatcherServlet(context);
        ServletRegistration.Dynamic registration = servletContext.addServlet("app", servle
t);
        registration.setLoadOnStartup(1);
        registration.addMapping("/app/*");
    }
}
```

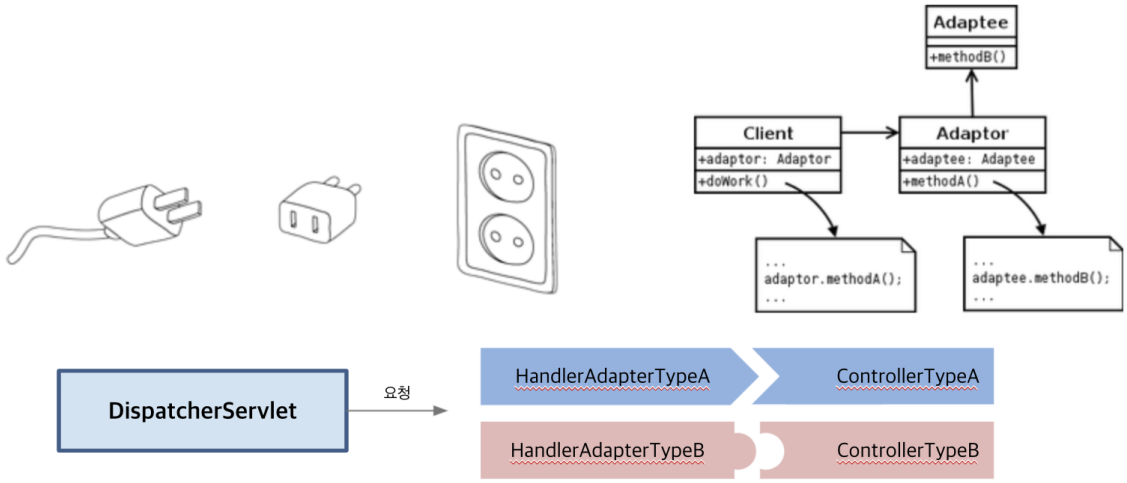
```
<web-app>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-cl
ass>
    </listener>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/app-context.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>app</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value></param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>app</servlet-name>
        <url-pattern>/app/*</url-pattern>
    </servlet-mapping>

</web-app>
```

DispatcherServlet에서 컨트롤러로 HTTP 요청 위임

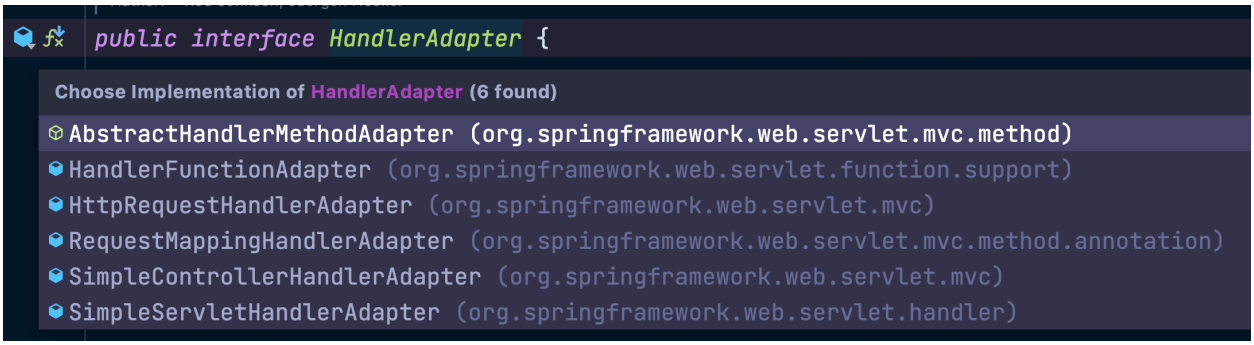


- 스프링에서는 컨트롤러를 `핸들러`라고도 부름
- 사용자 요청을 기준으로 어떤 핸들러에게 작업을 위임할지를 결정해주는 것 => **핸들러 매핑 전략**





- 요청이 전달되는 방법은 해당 컨트롤러 오브젝트의 메소드를 호출하는 방법 뿐
- 제각각 다른 메소드와 포맷을 가진 컨트롤러를 DispatcherServlet이 어떻게 알아서호출할 수 있을까? => `핸들러 어댑터 전략`



DispatcherServlet의 뷰 호출과 모델 참조

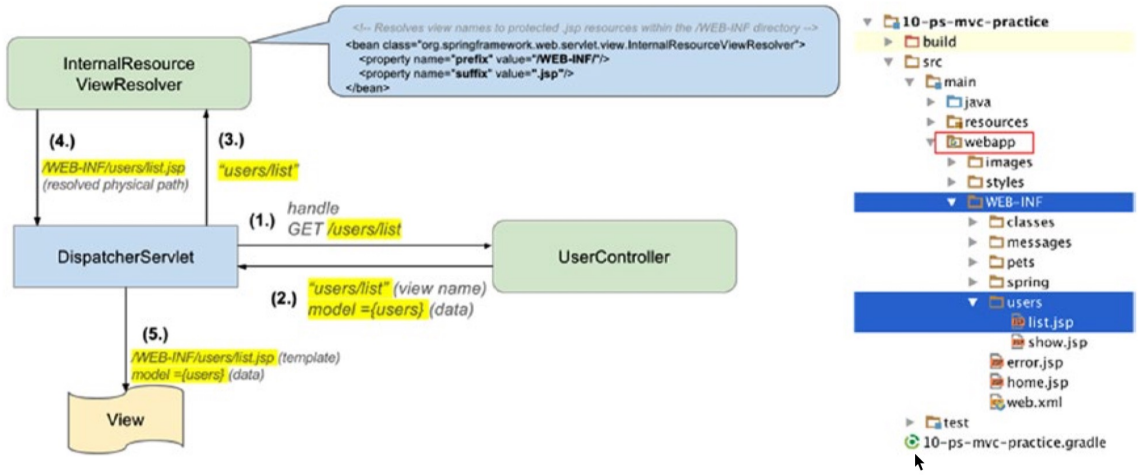
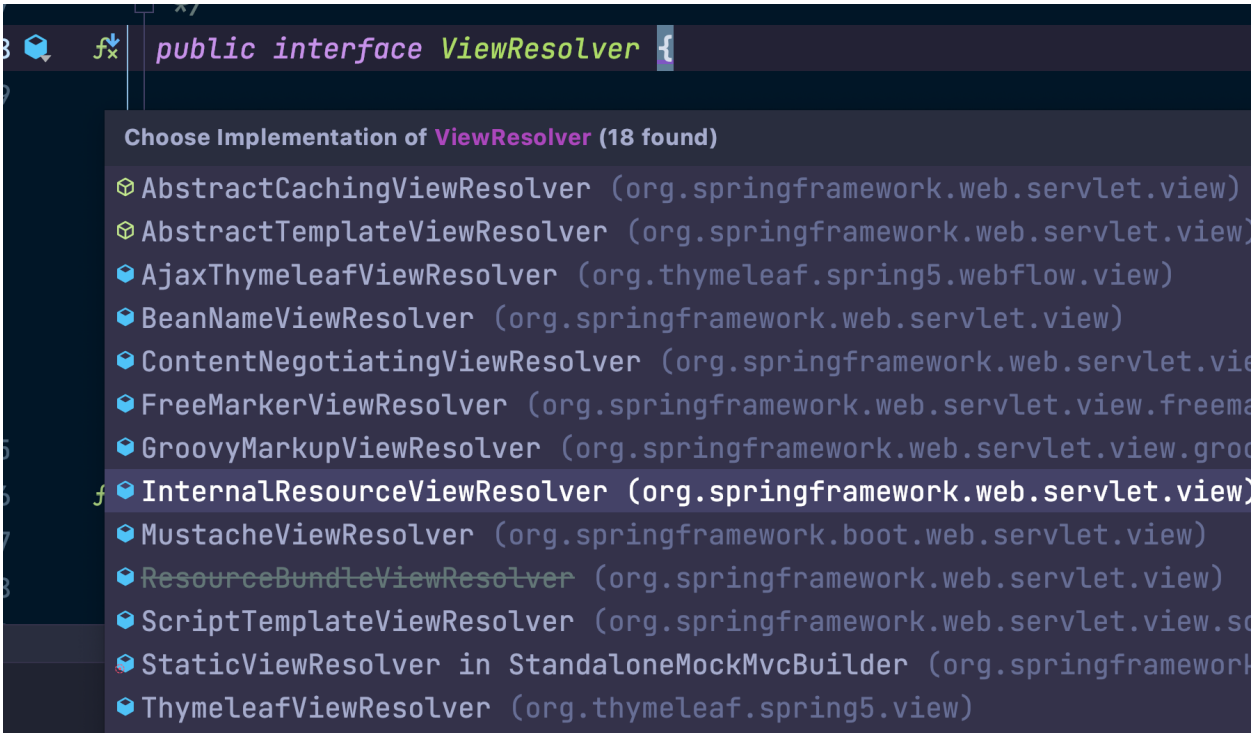
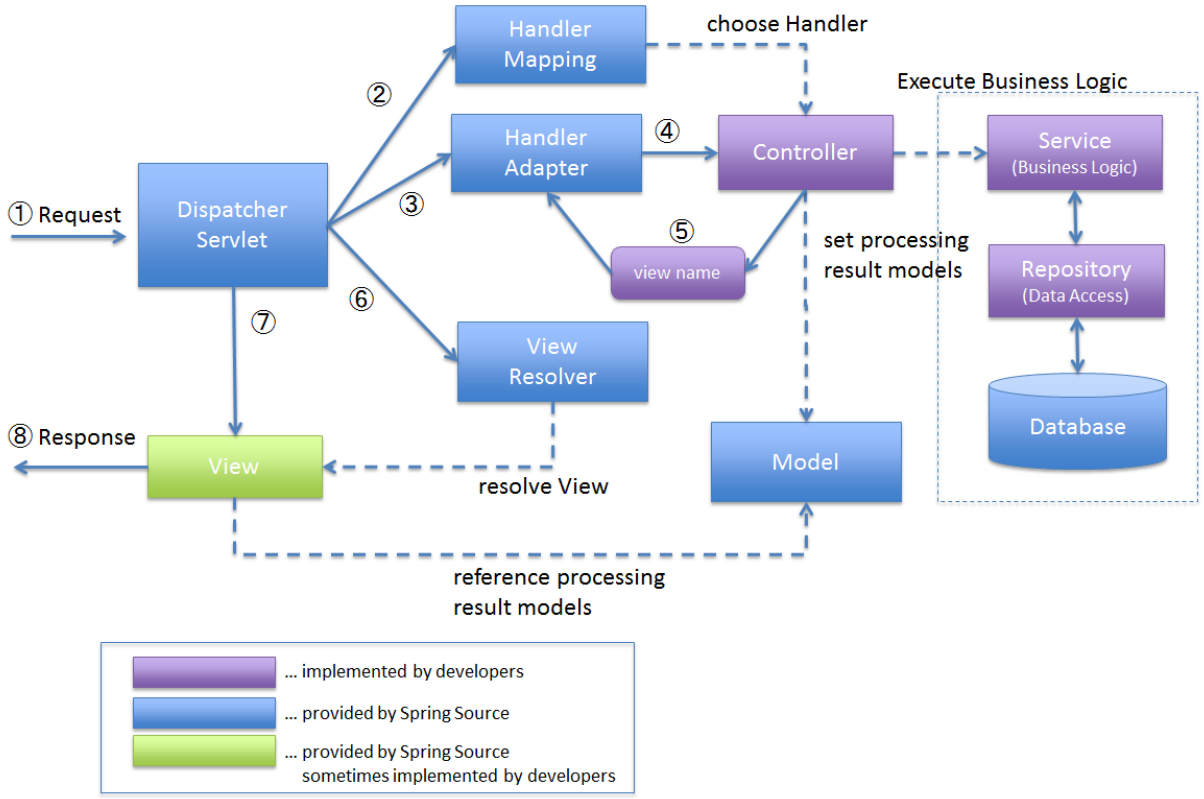


Figure 6-6. *Custom Internal Resource View Resolver Example*

https://mossgreen.github.io/Spring-Certification-Spring-MVC/





https://terasolunaorg.github.io/guideline/public_review/Overview/SpringMVCOverview.html

실습

Setup

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

JSTL의 기본 개념과 사용방법 정리

일반적으로 알고있는 JSTL이란 JSTL + EL 의 조합을 말한다. 예전에는 스크립틀릿을 많이 사용했지만 가독성이 떨어지고, 뷰와 비즈니스로직의 분리로 인해 현재는 JSTL을 많이 사용하는 추세다. JSTL과 EL은 보통 함께 사용하는데 그 이유와 각각의 개념, 각각의 차이와 문법을 정리해보도록 하자. JSTL의 정식 명칭은 자바서버 페이지 표준 태그 라이브러리

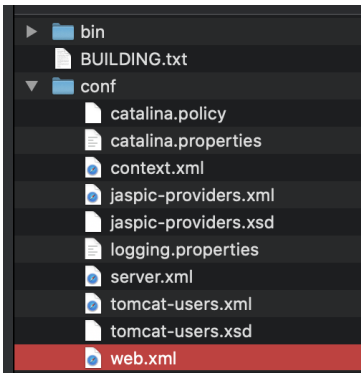
https://daesuni.github.io/jstl/

Spring Boot With JavaServer Pages (JSP) | Baeldung

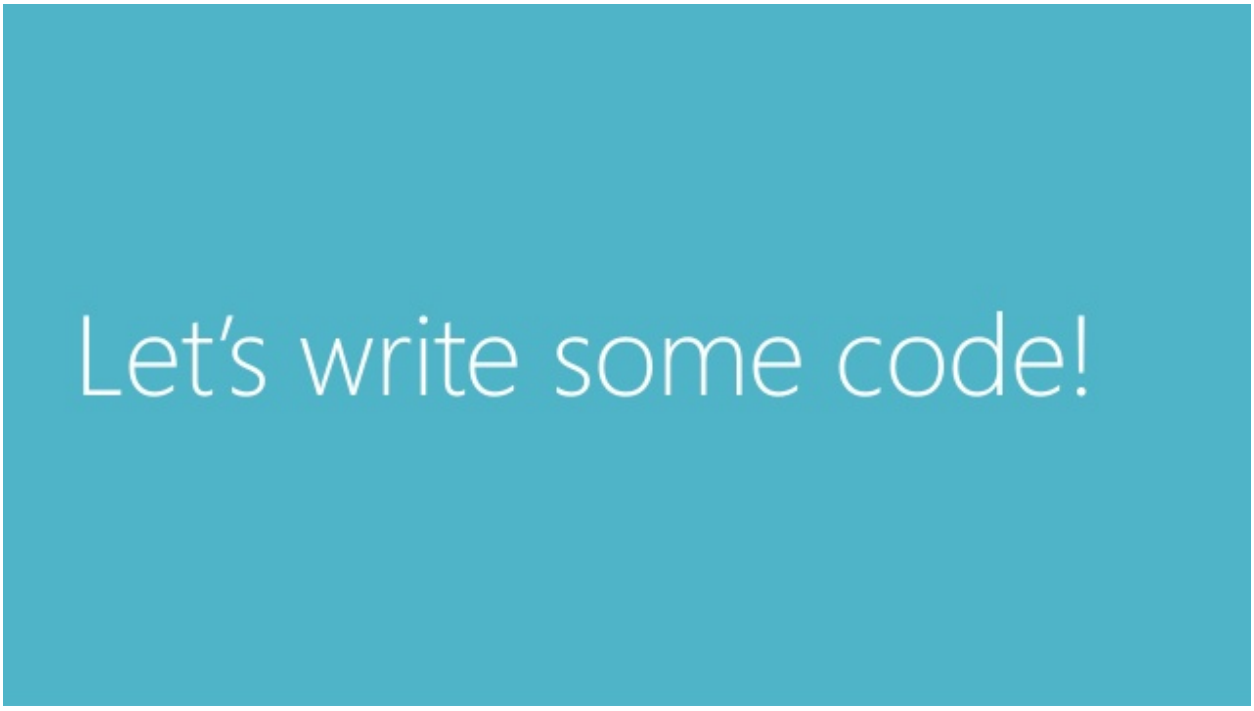
When building Web Applications, JavaServer Pages (JSP) is one option we can use as a templating mechanism for our HTML pages. On the other hand, Spring Boot is a popular framework we can use

https://www.baeldung.com/spring-boot-jsp

Static Resource 처리하기




```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```



Thymeleaf

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Expression

- 변수 식: ${OGNL}

- OGNL (Object-Graph Navigation Language) 표현식은 객체의 속성에 값을 가져오고 설정하는데 사용됩니다. 이전에 스트럿츠2라는 웹 프레임워크에서 사용되었고 마이바티스 등에서 사용되었습니다.

```
<p>Today is: <span th:text="${today}">13 february 2011</span>.</p>
==> ctx.getVariable("today");
```

> 여기서 ctx는 Model이 됩니다.

- 메시지 식: #{코드}
  - 스프링 메시지소스와 연동해서 메시지 코드를 넣으면 해당 메시지가 출력됩니다. 다국어 처리에 용이합니다.
- 링크 식: @{링크}

```
<!-- Will produce 'http://localhost:8080/gtvg/order/details?orderId=3' (plus rewritin
g) -->
<a href="details.html"
   th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/details?orderId=3' (plus rewriting) -->
<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/3/details' (plus rewriting) -->
<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a
>
```

> / 로 시작하는 패스는 자동으로 애플리케이션 컨텍스트 네임이 앞에 붙게됩니다.

- 선택 변수 식: *{OGNL}
  - th:object로 선택한 객체에 한해서 필드에 접근하게 됩니다.

https://www.baeldung.com/thymeleaf-in-spring-mvc

Let's write some code!