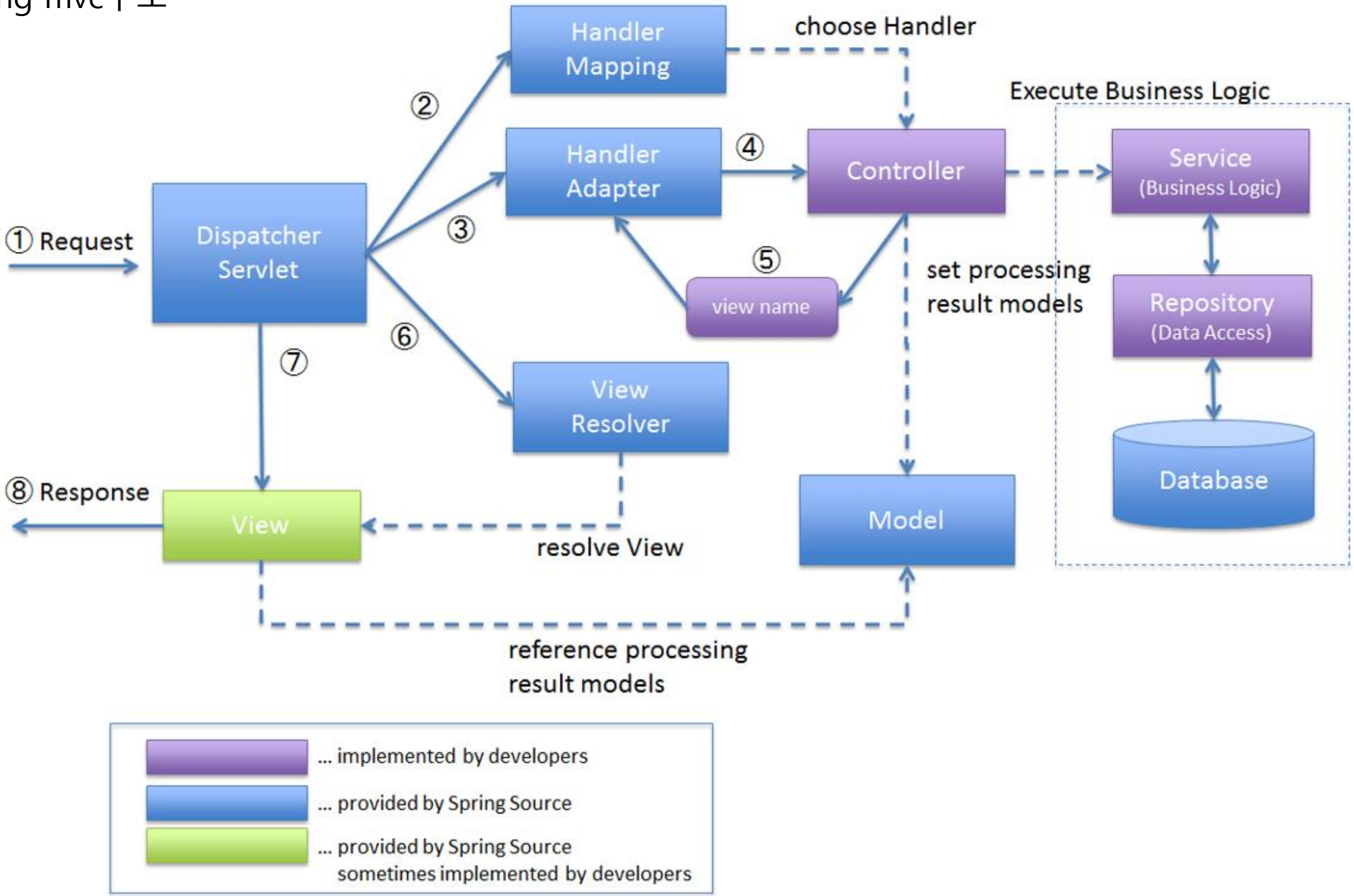
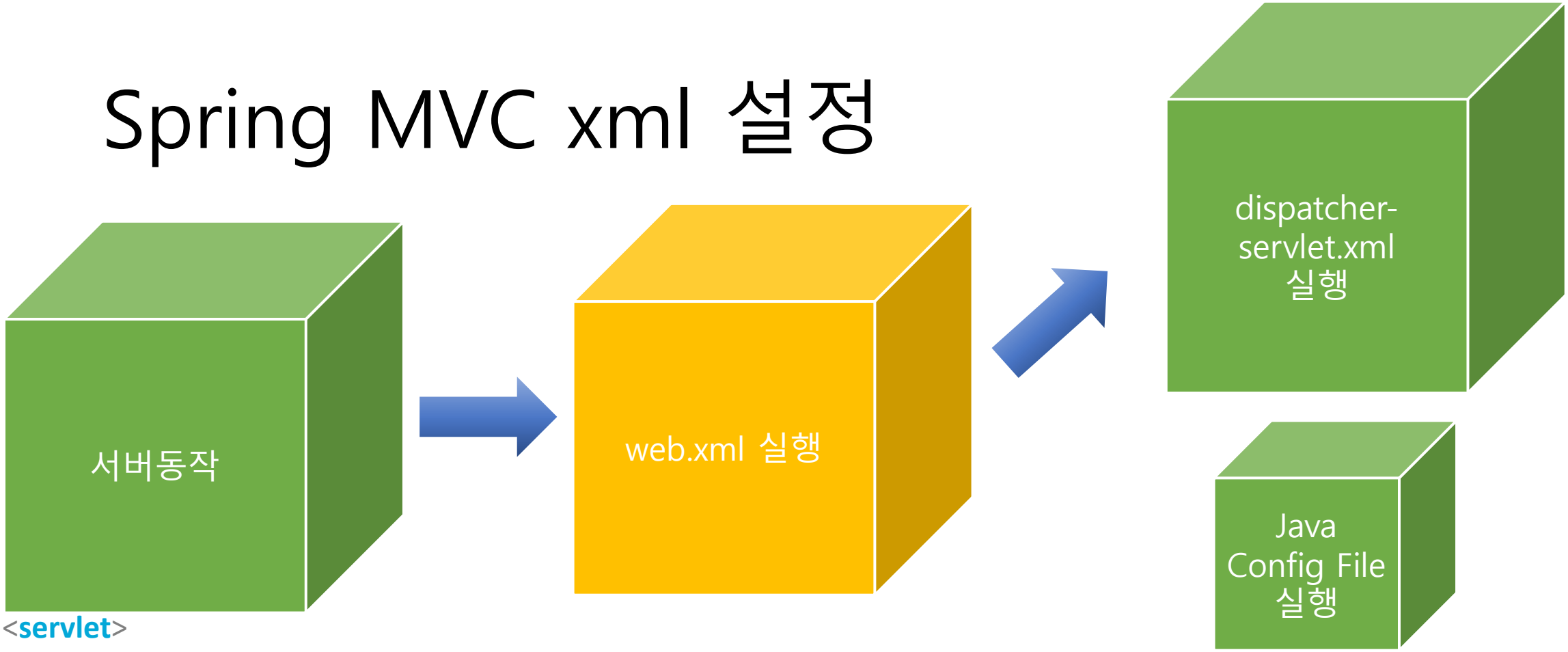
2. SPRING WEB MVC

Spring mvc구조

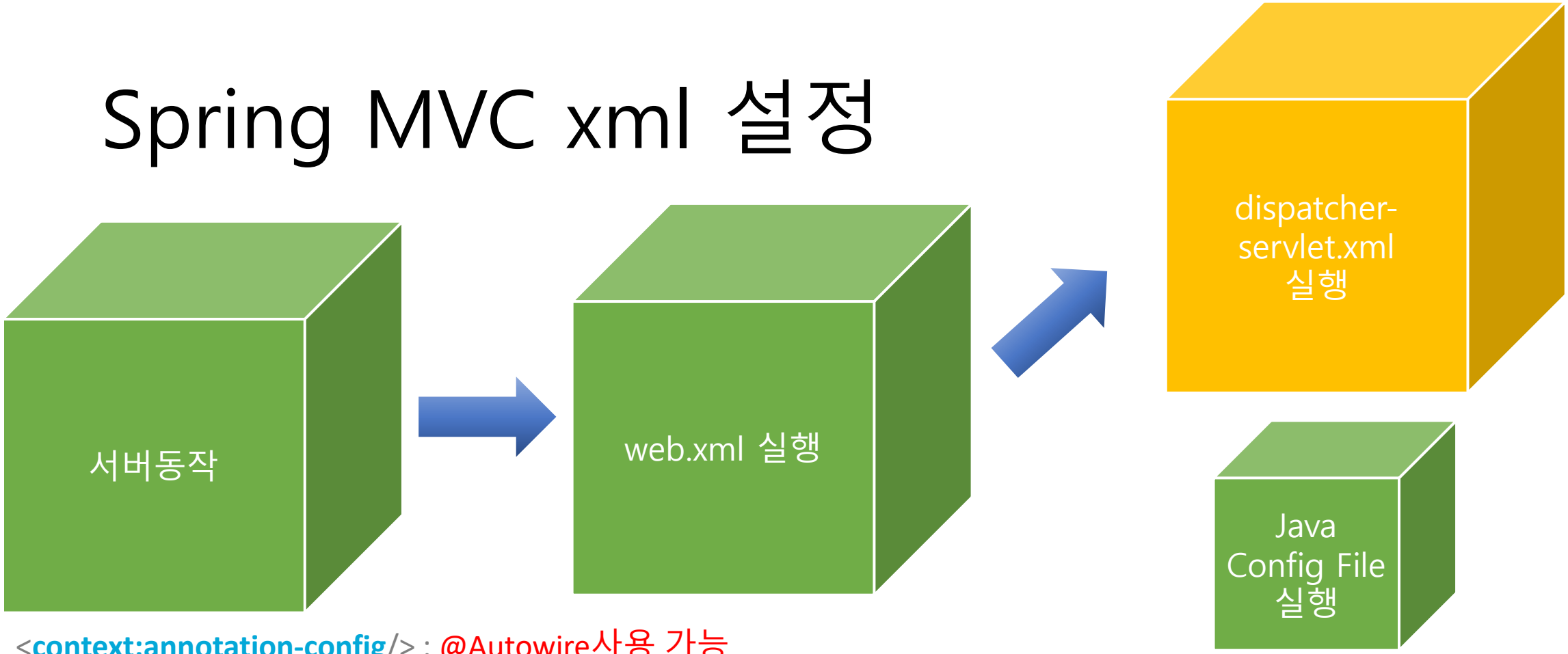


Spring MVC xml 설정



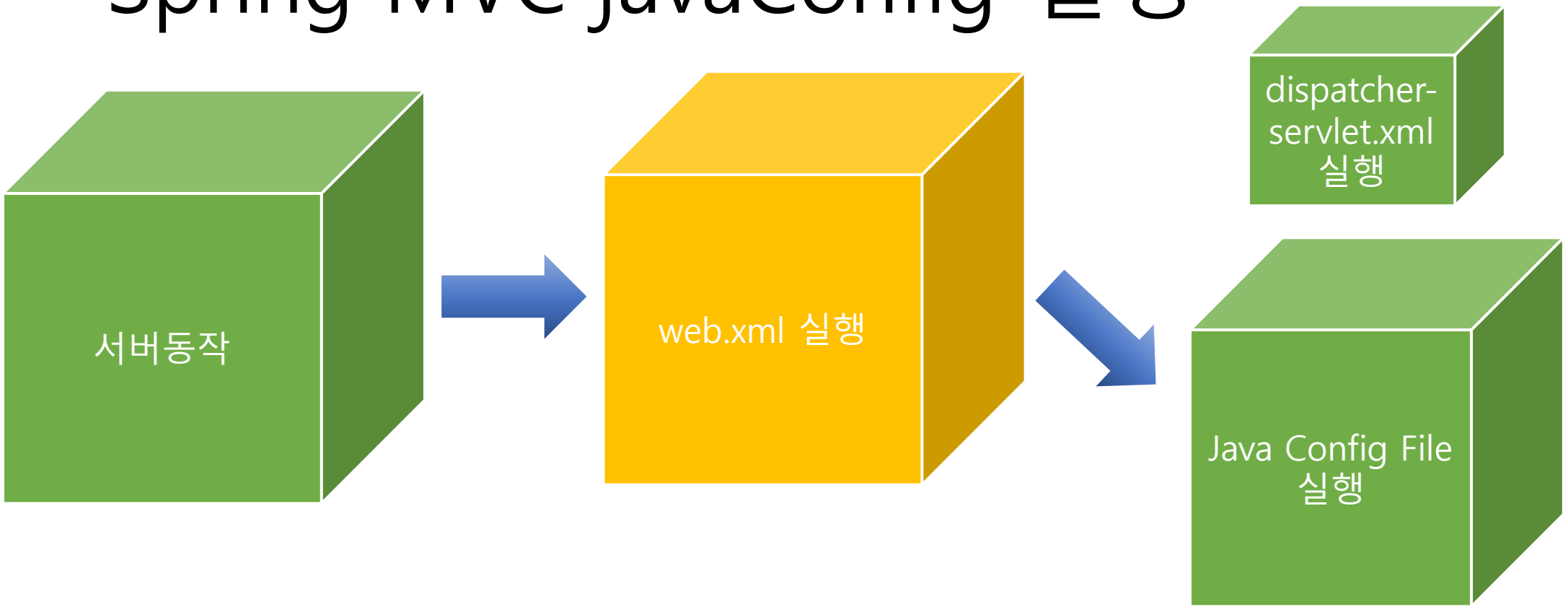

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Spring MVC xml 설정



<context:annotation-config/> : @Autowire사용 가능
<context:component-scan base-package="Controller"/> : @Controller 스캔
<bean : ViewResolver등록 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="order" value="2" />
<property name="prefix" value="/WEB-INF/view/" />
<property name="suffix" value=".jsp" />
</bean>

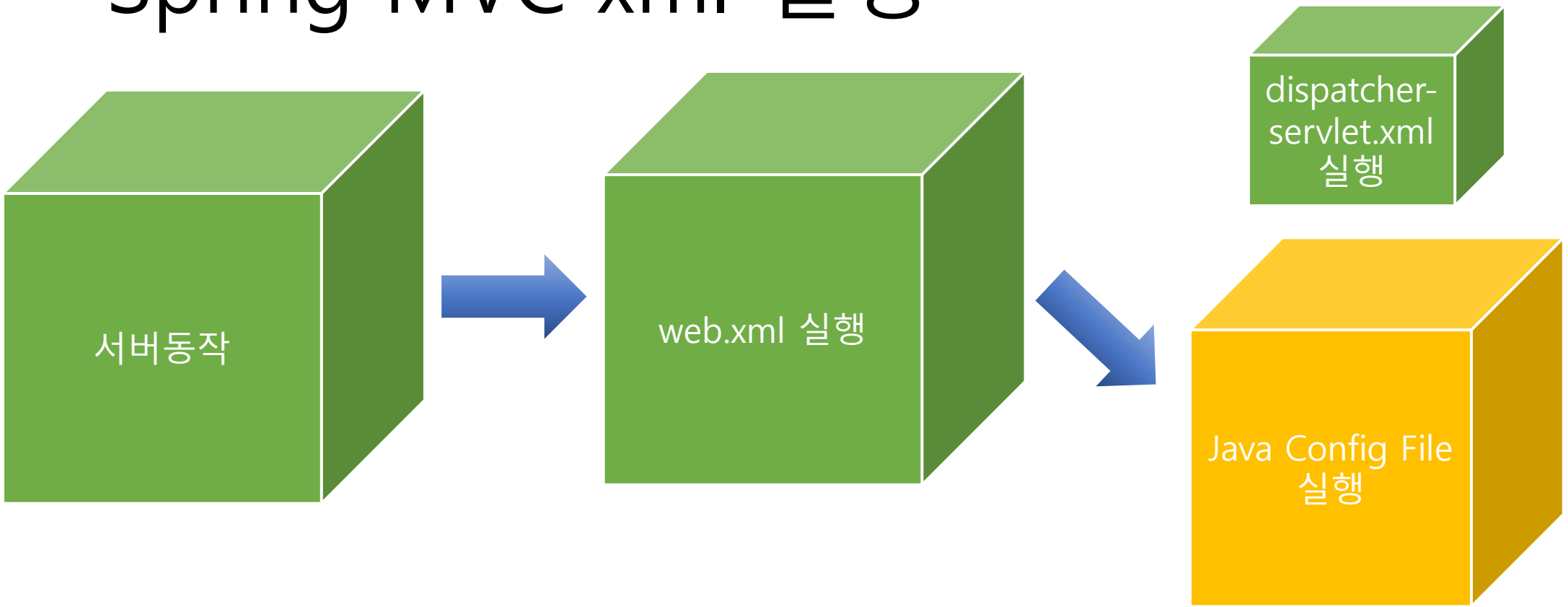
Spring MVC javaConfig 설정



web.xml

```
<servlet>
 <servlet-name>dispatcher</servlet-name>
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
자바 설정파일을 사용하기 위해 스프링 파일 도움설정
 <init-param>
    <param-name>contextClass</param-name>
    <param-value>
org.springframework.web.context.support.AnnotationConfigWebApplicationContext
</param-value>
  </init-param>
  사용자가 정의한 WebMvcContextConfiguration 위치 지정
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>springmvc.MvcConfig</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
 <servlet-name>dispatcher</servlet-name>
 <url-pattern>/</url-pattern>
 </servlet-mapping>
```

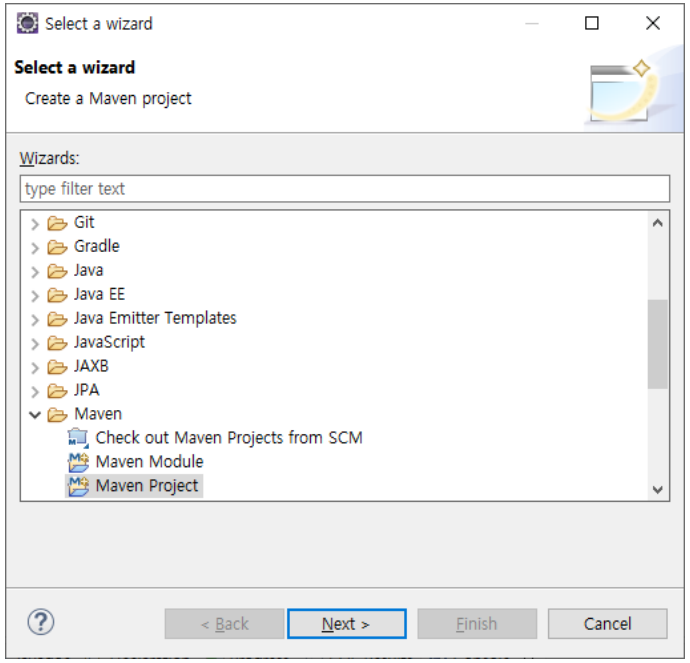
Spring MVC xml 설정

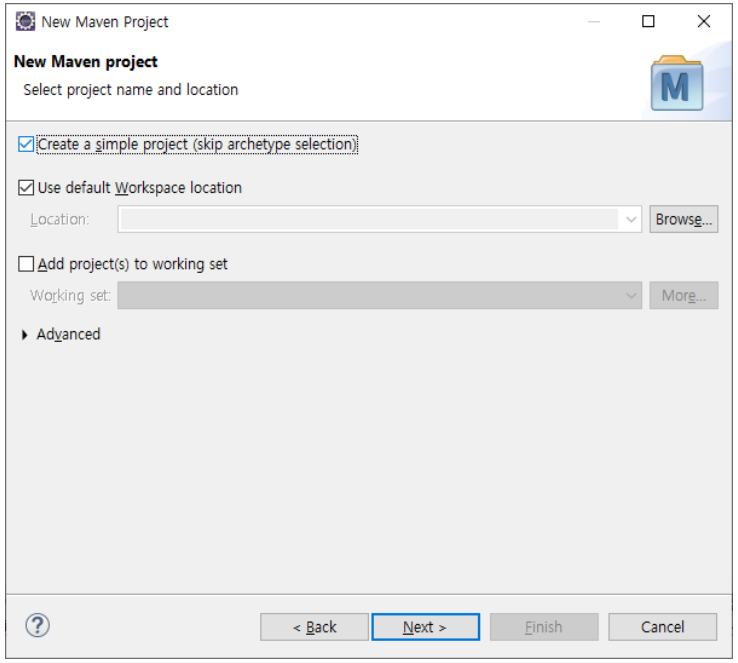


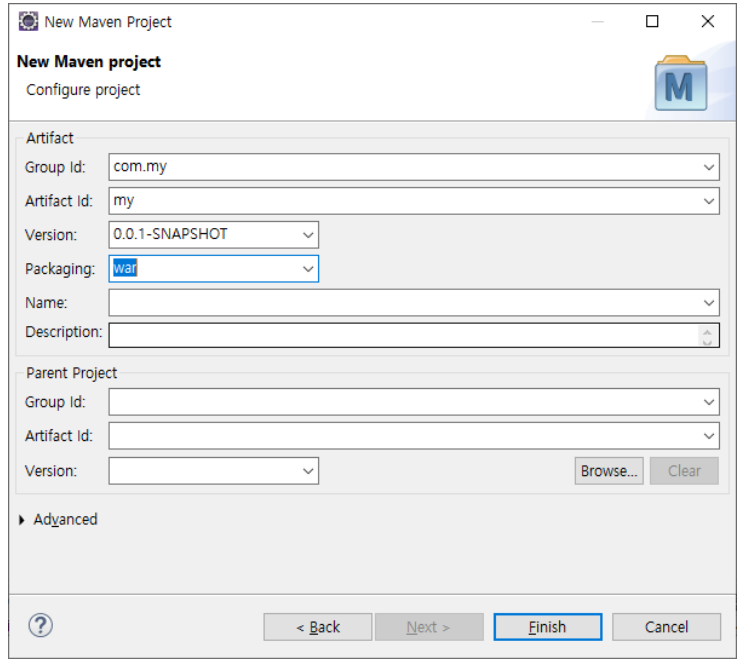
```
@Configuration  //이파일은xml파일과 같은 설정파일
@ComponentScan(basePackages = { "Controller" })   @Controller 스캔
//두개이상
//@ComponentScan(basePackages= {"springProject1","패키지명"})
public class MvcConfig implements WebMvcConfigurer {

//ViewResolver등록
   @Bean
      public InternalResourceViewResolver resolver() {
         InternalResourceViewResolver resolver =
new InternalResourceViewResolver();
         resolver.setViewClass(JstlView.class);
         resolver.setPrefix("/WEB-INF/view/");
         resolver.setSuffix(".jsp");
         return resolver;
      }


}
```

Web mvc

Maven web project







War파일 선택

Pom.xml설정

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.my</groupId>
<artifactId>springweb</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<build>
<plugins>
<plugin>
<artifactId>maven-war-plugin</artifactId>
<version>3.2.3</version>
</plugin>
</plugins>
</build>
<properties>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
```

아래의 버전이 업데이트가 되지 않으면
서버실행시 오류발생
1.8이상의 버전이여야함.

Webapp/index.jsp파일생성

- 파일 생성시 오류발생(윗줄에 servlet이 없다고 메시지 나타남)
- Pom.xml에서 등록

```
<dependency>
<groupId>org.apache.tomcat</groupId>
<artifactId>tomcat-jsp-api</artifactId>
<version>9.0.71</version>
</dependency>
```

Web.xml

```
<!-- spring dispatcher를 등록 - 서블릿 등록-서블릿 등록 xml파일을 이용하여 bean생성 -
->
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
```

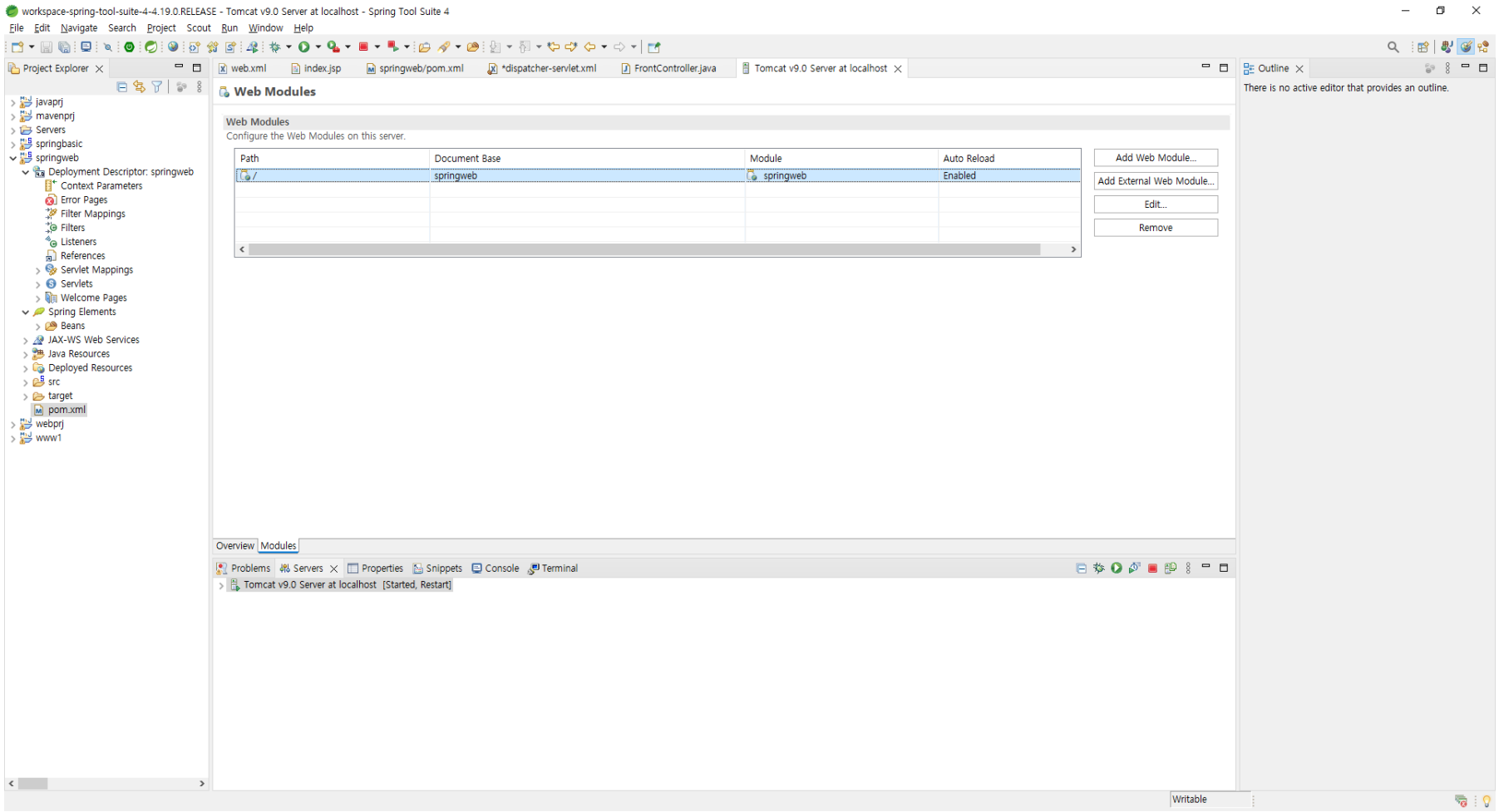
pom.xml에 등록

```
<dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>5.3.8</version>
</dependency>
</dependencies>
```

Controller생성

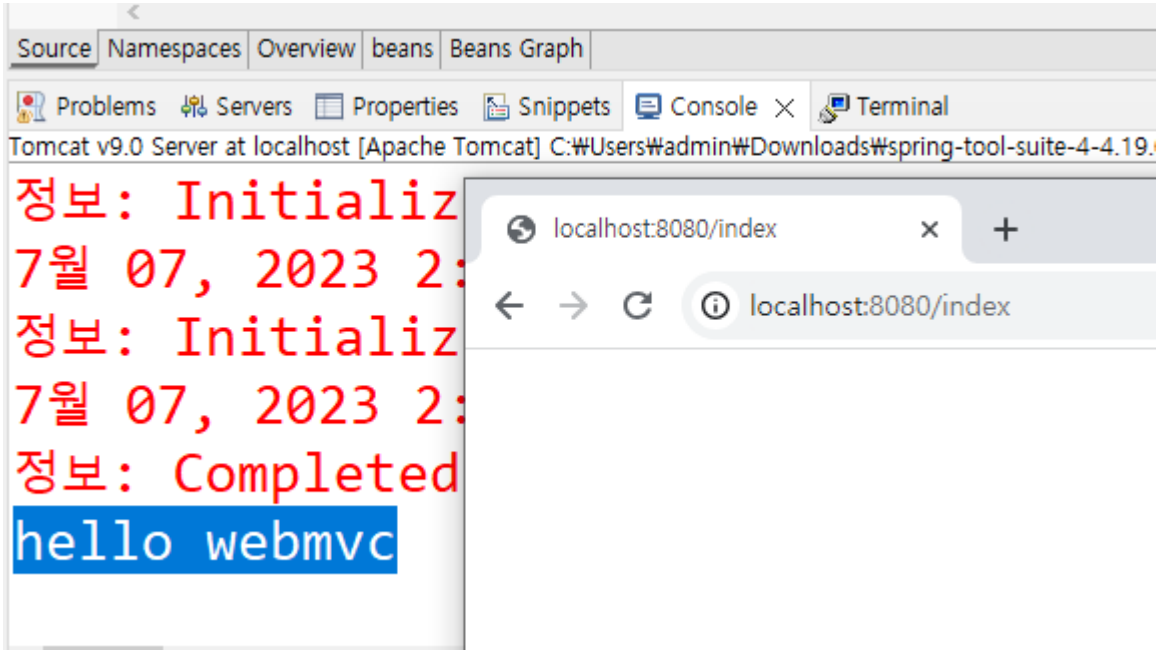
```
//spring으로 컨트롤러를 구현할 때 서블릿으로 생성하지 않는다.
//일반자바파일로 생성하고 spring에서 지원하는 Contoller인터페이스로
구현한다.


public class FrontController implements Controller{


@Override
public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response) throws Exception {
System.out.println("hello webmvc");
return null;
}
```

Server의 path설정을 /로



WEB-INF/dispatcher-servlet.xml

```
<!-- spring id의 역할 객체변수 web spring id의 역할 url역할 -->
<bean id="/index" class="controller.FrontController"></bean>
```



Controller를 구현할 경우의 코드

```
public class FrontController implements Controller{


@Override
public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response) throws Exception {
System.out.println("hello webmvc");
ModelAndView mv=new ModelAndView();
mv.addObject("data","1234");
mv.setViewName("/WEB-INF/view.jsp");
return mv;
}
}
```

view.jsp

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
${data}
```

어노테이션을 이용할 경우의 코드

```
@Controller
public class MainController {
@RequestMapping("/")
public String index() {
return “view";
}
}
```

요약정리
1.maven web 프로젝트 생성(packing-jar선택)

2.pom.xml (버전확인 후 추가)
```
<build>
    <plugins>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.3</version>
      </plugin>
    </plugins>
  </build>
<properties>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

3.webpp.index.jsp파일 생성하면 오류 발생
오류의 원인은 servlet -api가 없어서 발생하는 문제
pom.xml에 라이브러를 추가
```
<dependency>
<groupId>org.apache.tomcat</groupId>
<artifactId>tomcat-jsp-api</artifactId>
<version>9.0.71</version>
</dependency>
```

4.spring으로 서블릿을 사용함.web.xml에 서블릿을 등록
```
<!-- spring dispatcher를 등록 - 서블릿 등록-서블릿 등록 xml파일을 이용하여 bean생성 -->
  <servlet>
                    <servlet-name>dispatcher</servlet-name>
                    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
  </servlet-mapping>
```

5.spring라이브가 존재하지 않으므로 pom.xml에 라이브러리 추가
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.8</version>
</dependency>
```
주의사항)톰캣버전에 따라 spring버전이 달라지므로 jakarta를 시작으로 하는 spring라이브러리는 톰캣10버전이상에서 사용을 해야한다.

6. 4에서 spring 서블릿을 등록해 놓았으므로 Controller를 생성하여 spring 설정파일에 등록을 해야한다.
컨트롤러를 만들 때 주의할 사항
1)서블릿으로 생성하지 않고 자바파일로 생성하고
2)spring Controller 인터페이스를 활용해야하므로 Controller이라는 이름의 클래스를 사용할 수 없다.

```
public class FrontController implements Controller{

		@Override
		public ModelAndView handleRequest(HttpServletRequest request,
HttpServletResponse response) throws Exception {
				System.out.println("hello webmvc");
				ModelAndView mv=new ModelAndView();
				mv.addObject("data","1234");
				mv.setViewName("/WEB-INF/view.jsp");
				return mv;
		}

}
```

7.Controller를 사용하기 위해서는 bean객체를 생성해야한다.
bean생성하는 설정파일은 이미 이름이 정해져있다. WEB-INF/dispachter-servlet.xml (파일 생성)
6에서 만든 컨트롤러를 xml에서 bean으로 생성해야함.(spring configure파일로 생성할 것)

```
<!-- spring id의 역할 객체변수 web spring id의 역할 url역할 -->
<bean id="/index" class="controller.FrontController"></bean>
```

8.서버의 context path를 반드시 확인하고 테스트 진행
/WEB-INF/view.jsp파일을 만들고
${data} -> http://localhost:8080/index

```
@Controller
public class MainController {
@RequestMapping("/index")
public String index() {
return "index";
}
}
```

어노테이션을 이용한 자바파일

```java
//해당자바파일을 설정파일로 사용하기 위한 기본설정
@EnableWebMvc
@Configuration
//다른어노테이션을 스캔
@ComponentScan(basePackages = {"controller"})
public class AppConfig implements WebMvcConfigurer{
//registry를 이용한 viewResolver처리 : @bean이 필요없음
@Override
public void configureViewResolvers(ViewResolverRegistry registry) {
InternalResourceViewResolver bean = new InternalResourceViewResolver();
bean.setViewClass(JstlView.class);
bean.setPrefix("/WEB-INF/");
bean.setSuffix(".jsp");
registry.viewResolver(bean);
}
//css, js mapping설정
//registry등록 관련은 bean생성이 별도 필요없음
public void addResourceHandlers(ResourceHandlerRegistry registry) {
registry.addResourceHandler("/**").addResourceLocations("/");
}
}
```

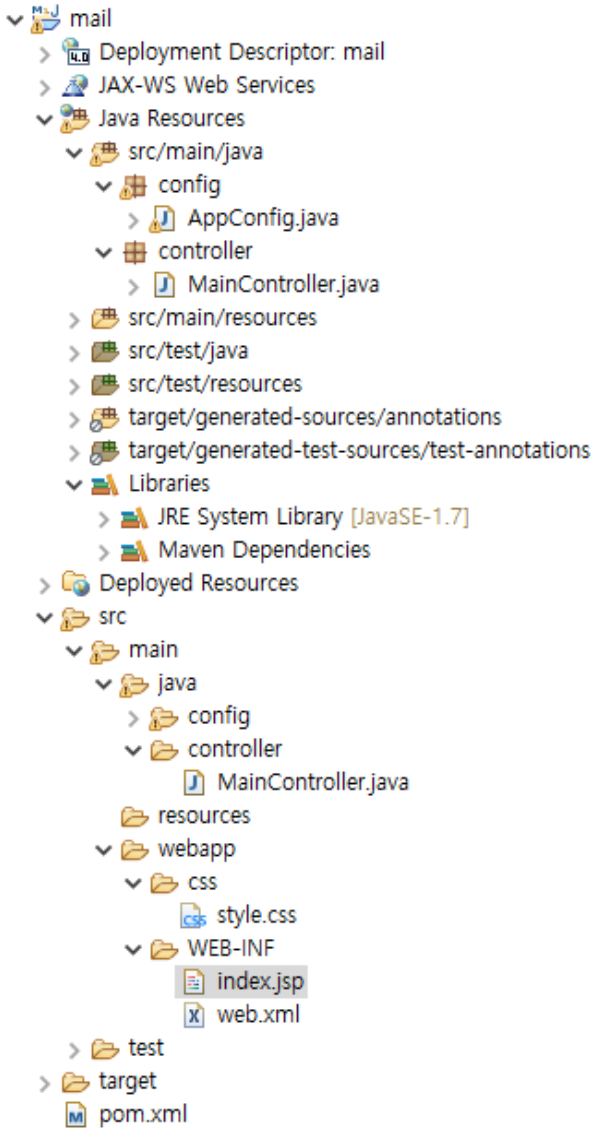
```java
//viewResolver설정 @bean을 이용하는 방법
/*
@Bean
public InternalResourceViewResolver resolver() {
InternalResourceViewResolver resolver=new InternalResourceViewResolver();
resolver.setViewClass(JstlView.class);
resolver.setPrefix("/WEB-INF/");
resolver.setSuffix(".jsp");
return resolver;
}
*/
```

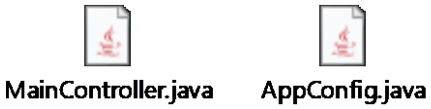
AppConfig.java

Web.xml 등록

```
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>


<init-param>
<param-name>contextClass</param-name>
<param-
value>org.springframework.web.context.support.AnnotationConfigWebApplicationC
ontext</param-value>
</init-param>


<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>config.AppConfig</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>



<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
```

기타파일


mail
Deployment Descriptor: mail
JAX-WS Web Services
Java Resources
src/main/java
config
AppConfig.java
controller
MainController.java
src/main/resources
src/test/java
src/test/resources
target/generated-sources/annotations
target/generated-test-sources/test-annotations
Libraries
JRE System Library [JavaSE-1.7]
Maven Dependencies
Deployed Resources
src
main
java
config
controller
MainController.java
resources
webapp
css
style.css
WEB-INF
index.jsp
web.xml
test
target
pom.xml


MainController.java
AppConfig.java


style.css
web.xml
index.jsp
MainController.java

Url 전달받기

- localhost/find?id=2
- Url의 모든 파라메다는 문자로 전송되지만 spring에서는 함수의 입력값 만으로 처리가능하다.
- 유의점은 변수명가 파라메다명이 같아야한다.
- 정수로 자동으로 변환이 가능하다.
- **public** ModelAndView selectById(**int** id) { }

Spring post전송시 한글깨짐 필터로 처리

Web.xml

```
<filter>
<filter-name>characterEncodingFilter</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>
<init-param>
<param-name>forceEncoding</param-name>
<param-value>true</param-value>
</init-param>
</filter>

<filter-mapping>
<filter-name>characterEncodingFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```