**서버로 실행 시키려면~~**

Window – Web browser – chrome 설정 하고,

프로젝트 잡고 마우스 우클릭 run on server를 하면, 창이 뜨는데, 거기서

Choose an existing server, tomcat v9.0, always use this server when running this project 클릭후,

Next 누르고, Configured에 사용할 프로젝트를 넣는다, finish. 하면 크롬에 작업한것들이 뜬다.

(기본적으로 만든 legacy project에는 주어진 기본 파일 있다. 실행해봐도 무관.)

**코딩이나 run on server를 하기 전!!!**

프로젝트 잡고 Properties.

Web Content Setting -document type : html5, CSS profile: CSS3: level3

Targeted Runtimes, Server -Tomcat 9.0

JSP Fragment -Language: java Content Type: text/html

Web Project setting -root: /

**Pom.xml에서~~**

Scope가 compile로 되어있는 경우, 지우는게 관례. 지워도 어차피 default로 compile로 지정된다. Scope는 특수한 경우에만 넣는다.

Dependencies 에서 수정을 할 경우, 귀찮아서 버전만 바꾸는 경우가 많지만... 역시 사이트에 들어가서 코드를 복붙하는게 제일 좋다. 혹여나 코드들이 바뀔 가능성이 있으므로!

**Log4j오류~~**

Log4j.xml에서 오류가 나는데, 이는, <DOCTYPE> 파트가 구버젼이라 오류가 나는것이므로, 신버젼 log4j의 DOCTYPE으로 바꿔줘야된다.(쌤꺼에서 가져오든지 어디서 가져와서 수정)

**스프링 프레임워크 설정~~**

스프링프레임워크에서는 3버전 이전에는 xml파일을 썻으나, 3버전 이후 xml파일대신 자바 클래스파일로 대체하였다. 그러므로, maven기반의 pom.xml파일빼고, 나머지는 다 삭제하고, 자바클래스 파일로 대체한다(log4j는 제외)

이를 연습하기 위해, jex03을 만든다. GroupID – com.zeork///artifactID – myapp… 이후 Web.xml, root-context.xml, servlet-context.xml 삭제~! 하면! Pom.xml에서 오류가 뜬다.

Pom.xml에서 <build>에 <plugin>을 찾아서, 플러그인을 하나 추가한다. 추가할때, artifactID는 maven-war-plugin을 넣고, API에서 검색을 해서, 최신 버전을 확인하고 수정을 하며, configuration part를 추가한다.

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>3.3.1</version>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

이것이 web.xml이 없어도 pom.xml에 오류가 안 뜨도록 한다( failOnMissingWebXml 을 false로 놓아서.)

Root-context.xml ->RootConfig.java

Servlet-context.xml은 ->ServletConfig.java

Web.xml은 ->WebConfig.java

이후, src/main/java에 있는 패키지 com.zerock.myapp밑에, com.zerock.myapp.config라는 패키지 만들기. 그리고 그 안에 클래스 RootConfig를 만들자. 이것이 root-context.xml을 대체한다.

어노테이션으로 @Configuration을 넣는다.(참고자료 0 참조)

**참고자료 0(RootConfig)**

**package** com.zerock.myapp.config;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public** **class** RootConfig {

} //end class

같은 패키지 안에, ServletConfig를 만들고, implements WebMvcConfigurer를 해주고, source에서 override/implements를 눌러서, addResourceHandlers, configureDefaultServletHandling, configureViewResolvers를 추가한다(참고자료 1 참조)

**참고자료 1(ServletConfig)**

**package** com.zerock.myapp.config;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;

**import** org.springframework.web.servlet.config.annotation.EnableWebMvc;

**import** org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

**import** org.springframework.web.servlet.config.annotation.ViewResolverRegistry;

**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@EnableWebMvc

@ComponentScan(basePackages= {"com.zerock.myapp"})

**public** **class** ServletConfig **implements** WebMvcConfigurer {

@Override

**public** **void** configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {

configurer.enable();

} //configureDefaultServletHandling

@Override

**public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/\*\*").addResourceLocations("/resources/");

} //addResourceHandlers

@Override

**public** **void** configureViewResolvers(ViewResolverRegistry registry) {

registry.jsp("/WEB-INF/views/", ".jsp");

} //configureViewResolvers

} //end class

같은 패키지 안에, WebConfig를 만들고, 이번에는 어노테이션이 아니라 상속으로, extends AbstractAnnotationConfigDispatcherServletInitializer넣고, ctrl shift o 하고, 추상메소드를 불러온다. 이것이 Web.xml을 대체한다. 3개의 추상 메소드에서 getRootConfigClasses() 메소드의 return이 null로 되어있는데, 이것을, Class<?>[] {RootConfig.class};로 바꾸어준다. (참고자료2 참조)

**참고자료 2(WebConfig)**

**package** com.zerock.myapp.config;

**import** org.springframework.web.servlet.support.

AbstractAnnotationConfigDispatcherServletInitializer;

**public** **class** WebConfig **extends**

AbstractAnnotationConfigDispatcherServletInitializer{

@Override

**protected** Class<?>[] getRootConfigClasses() {

//배열을 리턴할때, 그 값을 모르면... new 타입 {}.

//int[] a = new int[] {1,2,4} 랑 같은 원리!!

// return new Class[] {RootConfig.class}; //얘도 된다.

**return** **new** Class<?>[] {RootConfig.**class**}; //제네릭을 알든 모르든, 이것이 더 낫다.

} //getRootConfigClasses

@Override

**protected** Class<?>[] getServletConfigClasses() {

**return** **new** Class<?>[] {ServletConfig.**class**};

} //getServletConfigClasses

@Override

**protected** String[] getServletMappings() {

**return** **new** String[] {"/"};

} //getServletMappings

} //end class

Clazz 객체를 얻어내는 3가지 방법

1. 클래스이름.class
2. 참조변수.getClass()
3. Class.forName()

Beans class:

필드의 값 마다 getter/setter가 필요하고, default생성자(매개변수가 없는)있어야함(다른 생성자가 있어도 되지만 default constructor는 필수) Bean은 POJO(Plain Old Java Object 아주평볌한 클래스)이다.