

13강 스프링 MVC 자바 설정

Java 설정

- Com.green.ex00j로 새로운 프로젝트를 생성한다.
- 기존 프로젝트에서 모든 자바코드와 뷰페이지를 복사해온다.
- 기존 프로젝트에서 POM.xml의 내용을 복사한 후 메이븐 업데이트로 각종 라이브러리를 추가한다.

Java 설정

- 기존의 Web.xml 설정에서 xml 설정파일을 읽어오는 내용을 자바 클래스를 읽어오는 방식으로 변경한다.

```
<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.AnnotationConfigWebApplicationContext
    </param-value>
  </init-param>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      spring.config.MvcConfig
      spring.config.MemberConfig
      spring.config.ControllerConfig
    </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>
```

```
<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>
```

Java 설정

- 기존 xml 스프링 설정 파일을 자바 파일로 변환해보자
 - spring-mvc.xml => MvcConfig.java
 - spring-controller.xml => ControllerConfig.java
 - spring-member.xml => MemberConfig.java

Java 설정

- 기존 xml 스프링 설정 파일을 자바 파일로 변환해보자
- spring.config 패키지에 ControllerConfig, MemberConfig, MvcConfig 클래스 파일을 만든다.
- 모든 클래스는 설정파일이라는 의미로 @Configuration 애노테이션을 붙인다.

```
@Configuration  
public class ControllerConfig
```


```
@Configuration  
public class MvcConfig
```

```
@Configuration  
public class MemberConfig
```

MvcConfig 클래스 설정

- xml에서 <mvc:annotation-driven />을 이용해서 필요한 설정을 자동으로 생성했다.
- java에서는 @EnableWebMvc 을 사용한다.

<mvc:annotation-driven/>



```
@Configuration
@EnableWebMvc
public class MvcConfig
```

MvcConfig 클래스 설정

- mvc 네임 스페이스는 <mvc:annotation-driven />, <mvc:default-servlet-handler />, <mvc:view-resolvers> 태그등을 제공하는데 이런 설정을 java 설정코드에서는 WebMvcConfigurer인터페이스를 활용해서 추가설정을 진행한다.

--기존의 코드--

```
@Bean
public WebMvcConfigurer mvcConfigurer(){
    return new WebMvcConfigurer() {
        @Override
        public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
            configurer.enable();
        }
    }
}
```

MvcConfig 클래스 설정

- mvc 네임 스페이스는 <mvc:annotation-driven />, <mvc:default-servlet-handler />, <mvc:view-resolvers> 태그등을 제공하는데 이런 설정을 java 설정코드에서는 WebMvcConfigurer인터페이스를 활용해서 추가설정을 진행한다.


-- 현재 코드 : WebMvcConfigurerAdapter클래스를 상속받아서 구현한다.

```
public class MvcConfig extends WebMvcConfigurerAdapter{  
  
    @Override  
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {  
        configurer.enable();  
    }  
}
```


MvcConfig 클래스 설정

- <mvc:default-servlet-handler /> 를 설정해보자

```
<mvc:default-servlet-handler/>
```

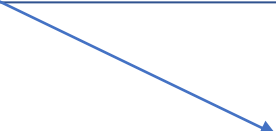


```
@Override  
public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {  
    configurer.enable();  
}
```

MvcConfig 클래스 설정

- JSP를 위한 ViewResolver 설정

```
<mvc:view-resolvers>  
  <mvc:jsp prefix="/WEB-INF/views/" />  
</mvc:view-resolvers>
```



```
@Override  
public void configureViewResolvers(ViewResolverRegistry registry) {  
    registry.jsp().prefix("/WEB-INF/views/").suffix(".jsp");  
}
```

- suffix()는 기본값이 jsp이므로 생략이 가능하다.

```
@Override  
public void configureViewResolvers(ViewResolverRegistry registry) {  
    registry.jsp().prefix("/WEB-INF/views/");  
}
```

MvcConfig 클래스 설정


- 스프링 4.0 이전 버전은 위 방식을 지원하지 않으므로 다음과 같은 방식을 사용한다.

```
@Bean
public ViewResolver viewResolver(){
    InternalResourceViewResolver result = new InternalResourceViewResolver();
    result.setPrefix("/WEB-INF/view/");
    result.setSuffix(".jsp");
    return result;
}
```

MvcConfig 클래스 설정

- 메시지 라벨을 읽어오기 위한 코드

```
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>classpath:message/label</value>
        </list>
    </property>
    <property name="defaultEncoding" value="UTF-8"/>
</bean>
```




```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource msgSrc =
        new ResourceBundleMessageSource();

    msgSrc.setBasenames("message.label");
    msgSrc.setDefaultEncoding("UTF-8");
    return msgSrc;
}
```

MemberConfig 클래스 설정

- 데이터베이스 연결을 위한 DataSource 빈을 설정한다.

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
  <property name="driverClass" value="oracle.jdbc.OracleDriver"/>
  <property name="jdbcUrl" value="jdbc:oracle:thin:@db.interstander.com:41521:XE"/>
  <property name="user" value="greenJSP"/>
  <property name="password" value="jsp1234"/>
  <property name="maxPoolSize" value="30"/>
</bean>
```



```
@Bean
public DataSource dataSource() {
    ComboPooledDataSource ds = new ComboPooledDataSource();

    try {
        ds.setDriverClass("oracle.jdbc.OracleDriver");
    } catch (PropertyVetoException e) {
        throw new RuntimeException(e);
    }


    ds.setJdbcUrl("jdbc:oracle:thin:@db.interstander.com:41521:XE");
    ds.setUser("greenJSP");
    ds.setPassword("jsp1234");

    return ds;
}
```

MemberConfig 클래스 설정

- transactionManager의 빈을 설정한다.

```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <property name="dataSource" ref="dataSource"/>  
</bean>
```

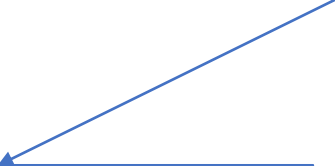


```
@Bean  
public DataSourceTransactionManager transactionManager() {  
    DataSourceTransactionManager txMgr = new DataSourceTransactionManager();  
    txMgr.setDataSource(dataSource());  
    return txMgr;  
}
```

MemberConfig 클래스 설정

- transactionManager의 빈을 설정한다.

```
<tx:annotation-driven transaction-manager="transactionManager"/>
```




```
@Configuration  
@EnableTransactionManagement  
public class MemberConfig {
```

ControllerConfig 클래스 설정

- 요청과 JSP를 바로 연결하는 코드를 설정한다.

```
<mvc:view-controller path="/main" view-name="main"/>
```

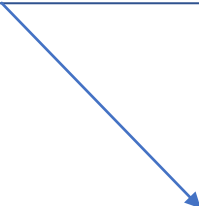


```
@Override  
public void addViewControllers(ViewControllerRegistry registry) {  
    registry.addViewController("/main").setViewName("main");  
}
```


ControllerConfig 클래스 설정

- 인터셉터 설정 - addInterceptors()메서드 사용

```
<mvc:interceptors>
  <mvc:interceptor>
    <!-- 어디에 -->
    <mvc:mapping path="/edit/**"/>
    <!-- 어떤 인터셉터 -->
    <bean class="spring.interceptor.AuthCheckInterceptor" />
  </mvc:interceptor>
</mvc:interceptors>
```



```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new AuthCheckInterceptor()).addPathPatterns("/edit/**");
}
```

ControllerConfig 클래스 설정

- addInterceptors()메서드는 인터셉터로 사용할 객체를 전달하고 , addPathPatterns()메서드는 인터셉터로 적용할 경로 패턴을 지정한다. 두개 이상의 경로를 지정할때는 ,를 사용한다.

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new AuthCheckInterceptor()).addPathPatterns("/edit/**","/member/**");
}
```

ControllerConfig 클래스 설정

- 제외해야 할 경로를 지정할 때 경우 excludePathPatterns()메서드

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new AuthCheckInterceptor()).addPathPatterns("/edit/**", "/member/**")
        .excludePathPatterns("/edit/help/**");
}
```

ControllerConfig 클래스 설정

- 스프링 빈을 인터셉터 객체로 사용하고 싶다면 해당 빈을 매개변수로 주입하거나 @Autowired로 주입받고 해당 필드를 사용한다.

```
@Bean
public AuthCheckInterceptor authCheckInterceptor() {
    return new AuthCheckInterceptor();
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(authCheckInterceptor()).addPathPatterns("/edit/**");
}
```

```
@Autowired
private AuthCheckInterceptor authCheckInterceptor;

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(authCheckInterceptor).addPathPatterns("/edit/**");
}
```