스프링 웹 MVC

이 강좌는 자바 서블릿(Servlet) 기반의 MVC 프레임워크인 스프링 웹 MVC(이하 스프링 MVC)에 대해 학습합니다.

자바 엔터프라이즈 에디션(Jave EE)은 웹 애플리케이션을 개발할 수 있는 HTTP Servlet(이하 서블릿)이라는 스팩과 API를 제공합니다. 스프링 MVC는 서블릿 API 기반 애플리케이션을 개발할 때 보다 쉽고 빠르게 개발할 수 있는 프레임워크를 제공하여 개발자가 서블릿 API 보다는 애플리케이션 로직에 집중할 수 있도록 도와줍니다. 현재 많은 회사에서 스프링 MVC 기반으로 엔터프라이즈 애플리케이션을 개발하고 배포하며 운영하고 있습니다.

이 강좌는 스프링 MVC 동작 원리와 설정 방법 그리고 애노테이션 기반 MVC 활용 방법에 대해 다루고 있으며 다음과 같은 특징이 있습니다

1. 아쉽지만 Webflux는 다루지 않습니다.
스프링 프레임워크 5 버전부터 리액티브 스택 기반으로 웹 애플리케이션을 개발할 때 사용할 수 있는 스프링 Webflux를 제공하지만 이번 강좌에서 다루는 기술 스택과 차이가 크기 때문에 이번 강좌에서 다루지 않습니다.

2. 서블릿에 대해 학습합니다.
이번 강좌에서는 스프링 MVC 동작 원리를 이해하는데 필요한 서블릿 기능에 대해 학습합니다. 따라서 서블릿 기반 웹 애플리케이션 개발이 처음이거나 스프링 MVC 동작 원리가 궁금했던 학생 또는 개발자에게 유용할 것입니다.

3. 타임리프를 주로 사용합니다.
스프링 MVC 기능 학습에 필요한 뷰를 만들 때 타임리프(Thymeleaf)를 사용합니다. 하지만 타임리프와 JSP(Java Server Pages) 또는 기타 다른 뷰 템플릿 엔진에 대해서 자세히 학습하지는 않습니다. 이번 강좌는 스프링 MVC에 집중하겠습니다.

4. 스프링 부트
이번 강좌도 스프링 부트를 사용하여 예제 프로젝트를 만들고 코딩하지만, 스프링 부트 없이 설정하는 방법도 학습합니다. 그러면 스프링 부트가 제공하는 자동 설정을 보다 잘 이해할 수 있을 뿐 아니라 스프링 MVC 설정을 원하는 대로 고쳐 사용할 수 있을 것입니다.

5. 테스트 친화적 개발
뷰를 만들고 요청을 보내는 방법으로 스프링 MVC 기능을 확인하고 학습할 수도 있지만 테스트 코드를 작성하여 확인하는 방법을 익히는 것 또한 효율적이며 중요합니다. 따라서 이번 강좌에서는 모든 스프링 MVC 기능을 코드로 테스트 하는 방법도 소개합니다.

원활한 학습을 위해 이번 강좌를 수강하기 전에 다음 강좌 수강을 고려해 주시기 바랍니다.

- 스프링 프레임워크 핵심 기술 (필수)

- 스프링 부트 개념과 활용 (선택)

학습 목표

- 애노테이션 기반 스프링 MVC의 동작 원리를 이해합니다.
- 스프링 MVC가 제공하는 다양한 기능을 이해합니다.
- 사용하는 스프링 MVC 기능에 대한 테스트 코드를 작성할 수 있습니다.
- 스프링 부트 없이도 스프링 MVC 애플리케이션을 개발할 수 있습니다.
- 스프링 부트가 제공하는 스프링 MVC 설정을 고쳐 사용할 수 있습니다.

예제 코드 저장소

- 1부 스프링 MVC 핵심 원리: https://github.com/keesun/javaservletdemo
- 2부 스프링 MVC 설정: https://github.com/keesun/demo-boot-web
- 3부 스프링 MVC 활용: https://github.com/keesun/demo-web-mvc

1. 강좌 소개

첫 페이지 참고

2. 강사 소개

백기선

- 현재 마이크로소프트 미국 본사에 근무 중. (그전에는 네이버와 아마존에서 일을 했습니다.)
- 2007년부터 개발자로 일했으며 이제 막 경력 10년이 조금 넘었네요.
- 자바, 스프링 프레임워크, JPA, 하이버네이트를 주로 공부하고 공유해 왔습니다.
- Youtube/백기선 채널에서 코딩 관련 정보를 영상으로 공유하고 있습니다.
- (예전에는 Whiteship.me 라는 블로그에 글도 많이 올렸지만 요즘은 잘 안써요.)
- (예전에는 책도 쓰고 번역도 하고 발표도 종종 했었지만 역시나.. 요즘은 거의 안합니다.)

1부. 스프링 MVC 동작 원리

3. 스프링 MVC 소개

스프링 MVC로 웹 애플리케이션 개발하기 소개

M: 모델
V: 뷰
C: 컨트롤러

모델: 평범한 자바 객체 POJO
뷰: HTML. JSP, **타임리프**, ...
컨트롤러: 스프링 @MVC

모델: 도메인 객체 또는 DTO로 화면에 전달할 또는 화면에서 전달 받은 데이터를 담고 있는 객체.
뷰: 데이터를 보여주는 역할. 다양한 형태로 보여줄 수 있다. HTML, JSON, XML, ...
컨트롤러: 사용자 입력을 받아 모델 객체의 데이터를 변경하거나, 모델 객체를 뷰에 전달하는 역할.

- 입력값 검증
- 입력 받은 데이터로 모델 객체 변경
- 변경된 모델 객체를 뷰에 전달

MVC 패턴의 장점

- 동시 다발적(Simultaneous) 개발 - 백엔드 개발자와 프론트엔드 개발자가 독립적으로 개발을 진행할 수 있다.
- 높은 결합도 - 논리적으로 관련있는 기능을 하나의 컨트롤러로 묶거나, 특정 모델과 관련있는 뷰를 그룹화 할 수 있다.
- 낮은 의존도 - 뷰, 모델, 컨트롤러는 각각 독립적이다.
- 개발 용이성 - 책임이 구분되어 있어 코드 수정하는 것이 편하다.
- 한 모델에 대한 여러 형태의 뷰를 가질 수 있다.

MVC 패턴의 단점

- 코드 네비게이션 복잡함
- 코드 일관성 유지에 노력이 필요함
- 높은 학습 곡선

참고

- https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
- https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html

4. 서블릿 소개

가장 기본적인 서블릿 애플리케이션 만들기

서블릿 (Servlet)
- 자바 엔터프라이즈 에디션은 웹 애플리케이션 개발용 스팩과 API 제공.
- 요청 당 쓰레드 (만들거나, **풀에서 가져다가**) 사용
- 그 중에 가장 중요한 클래스중 하나가 HttpServlet.

서블릿 등장 이전에 사용하던 기술인 CGI (Common Gateway Interface)
- 요청 당 프로세스를 만들어 사용

서블릿의 장점 (CGI에 비해)
- 빠르다.
- 플랫폼 독립적
- 보안
- 이식성

서블릿 엔진 또는 서블릿 컨테이너 (톰캣, 제티, 언더토, ...)
- 세션 관리
- 네트워크 서비스
- MIME 기반 메시지 인코딩 디코딩
- 서블릿 생명주기 관리
- ...

서블릿 생명주기
- 서블릿 컨테이너가 서블릿 인스턴스의 init() 메소드를 호출하여 초기화 한다.
    - 최초 요청을 받았을 때 한번 초기화 하고 나면 그 다음 요청부터는 이 과정을 생략한다.
- 서블릿이 초기화 된 다음부터 클라이언트의 요청을 처리할 수 있다. 각 요청은 별도의 쓰레드로 처리하고 이때 서블릿 인스턴스의 service() 메소드를 호출한다.
    - 이 안에서 HTTP 요청을 받고 클라이언트로 보낼 HTTP 응답을 만든다.
    - service()는 보통 HTTP Method에 따라 doGet(), doPost() 등으로 처리를 위임한다.
    - 따라서 보통 doGet() 또는 doPost()를 구현한다.
- 서블릿 컨테이너 판단에 따라 해당 서블릿을 메모리에서 내려야 할 시점에 destroy()를 호출한다.

5. 서블릿 애플리케이션 개발

준비물: 메이븐, 톰캣

서블릿 구현

```
public class HelloServlet extends HttpServlet {

  @Override
  public void init() throws ServletException {
    System.out.println("init");
  }

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    System.out.println("doGet");
    resp.getWriter().write("Hello Servlet");
  }

  @Override
  public void destroy() {
    System.out.println("destroy");
  }
}
```

서블릿 등록

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
 <display-name>Archetype Created Web Application</display-name>

 <servlet>
  <servlet-name>hello</servlet-name>
  <servlet-class>me.whiteship.HelloServlet</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>hello</servlet-name>
  <url-pattern>/hello</url-pattern>
 </servlet-mapping>
</web-app>
```

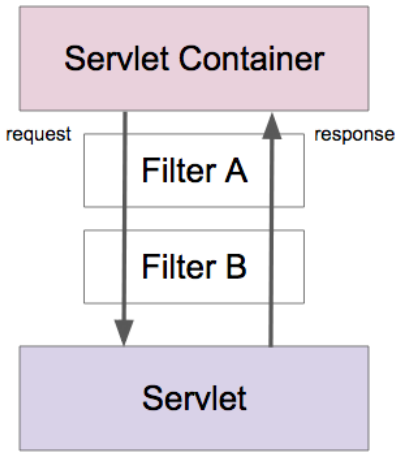
6. 서블릿 리스너와 필터

서블릿 리스너

- 웹 애플리케이션에서 발생하는 주요 이벤트를 감지하고 각 이벤트에 특별한 작업이 필요한 경우에 사용할 수 있다.
    - 서블릿 컨텍스트 수준의 이벤트
        - 컨텍스트 라이프사이클 이벤트
        - 컨텍스트 애트리뷰트 변경 이벤트
    - 세션 수준의 이벤트
        - 세션 라이프사이클 이벤트
        - 세션 애트리뷰트 변경 이벤트

서블릿 필터

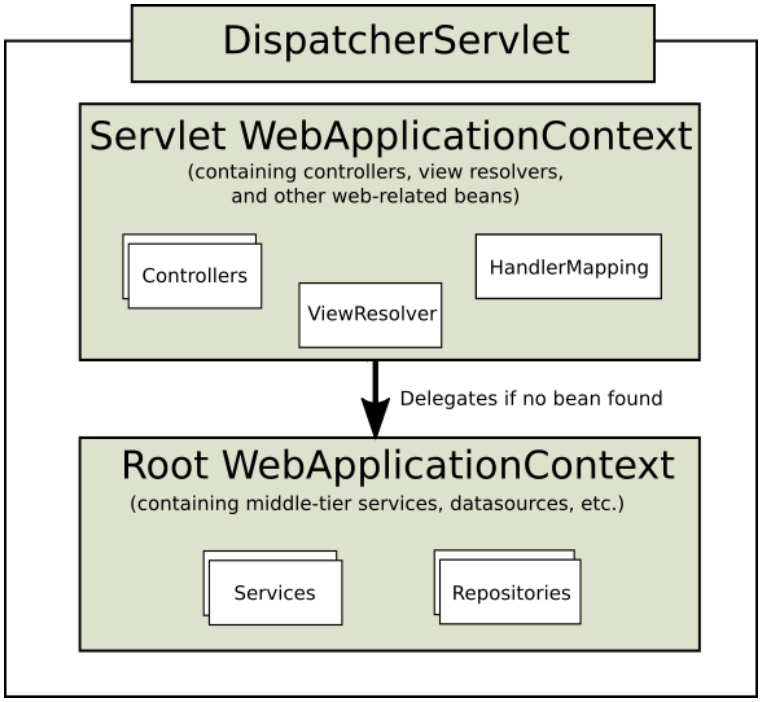
- 들어온 요청을 서블릿으로 보내고, 또 서블릿이 작성한 응답을 클라이언트로 보내기 전에 특별한 처리가 필요한 경우에 사용할 수 있다.
- 체인 형태의 구조



참고

- https://docs.oracle.com/cd/B14099_19/web.1012/b14017/filters.htm#i1000654

7. 스프링 IoC 컨테이너 연동



(출처: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc)

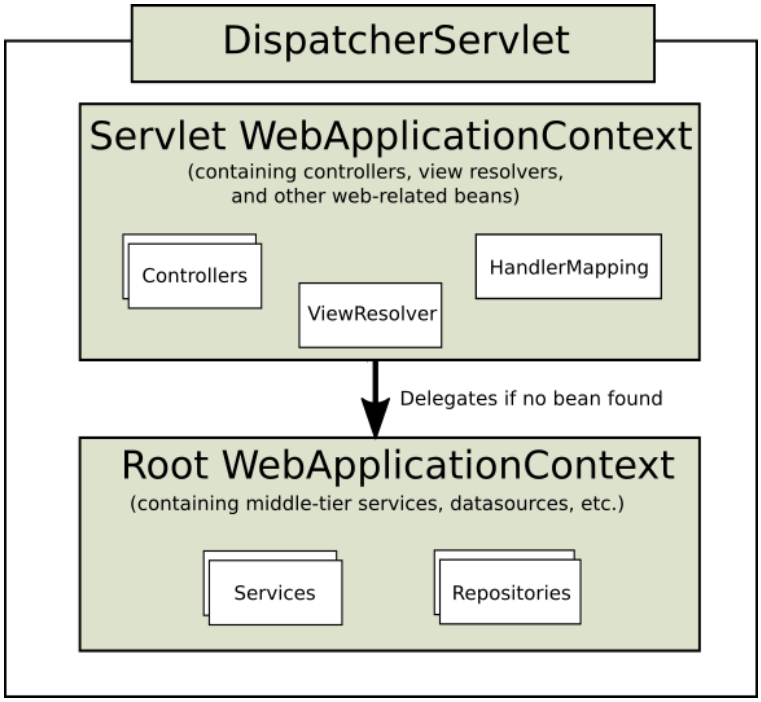
서블릿 애플리케이션에 스프링 연동하기

- **서블릿에서 스프링이 제공하는 IoC 컨테이너 활용하는 방법**
- 스프링이 제공하는 서블릿 구현체 DispatcherServlet 사용하기

ContextLoaderListener

- 서블릿 리스너 구현체
- ApplicationContext를 만들어 준다.
- ApplicationContext를 서블릿 컨텍스트 라이프사이클에 따라 등록하고 소멸시켜준다.
- 서블릿에서 IoC 컨테이너를 ServletContext를 통해 꺼내 사용할 수 있다.

8. 스프링 MVC 연동



(출처: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc)

서블릿 애플리케이션에 스프링 연동하기

- 서블릿에서 스프링이 제공하는 IoC 컨테이너 활용하는 방법
- **스프링이 제공하는 서블릿 구현체 DispatcherServlet 사용하기**

DispatcherServlet

- 스프링 MVC의 핵심.
- Front Controller 역할을 한다.

참고

- http://www.corej2eepatterns.com/FrontController.htm
- https://www.oracle.com/technetwork/java/frontcontroller-135648.html
- https://martinfowler.com/eaaCatalog/frontController.html

9. DispatcherServlet 동작 원리 1부

DispatcherServlet 초기화

- 다음의 특별한 타입의 빈들을 찾거나, 기본 전략에 해당하는 빈들을 등록한다.
- HandlerMapping: 핸들러를 찾아주는 인터페이스
- HandlerAdapter: 핸들러를 실행하는 인터페이스
- HandlerExceptionResolver
- ViewResolver
- ...

DispatcherServlet 동작 순서

1. 요청을 분석한다. (로케일, 테마, 멀티파트 등)
2. (핸들러 맵핑에게 위임하여) 요청을 처리할 핸들러를 찾는다.
3. (등록되어 있는 핸들러 어댑터 중에) 해당 핸들러를 실행할 수 있는 “핸들러 어댑터”를 찾는다.
4. 찾아낸 “핸들러 어댑터”를 사용해서 핸들러의 응답을 처리한다.
    - 핸들러의 리턴값을 보고 어떻게 처리할지 판단한다.
        - 뷰 이름에 해당하는 뷰를 찾아서 모델 데이터를 랜더링한다.
        - @ResponseEntity가 있다면 Converter를 사용해서 응답 본문을 만들고.
5. (부가적으로) 예외가 발생했다면, 예외 처리 핸들러에 요청 처리를 위임한다.
6. 최종적으로 응답을 보낸다.

HandlerMapping

- RequestMappingHandlerMapping

HandlerAdapter

- RequestMappingHandlerAdapter

10. DispatcherServlet 동작 원리 2부: SimpleController

HandlerMapping

- BeanNameUrlHandlerMapping

HandlerAdapter

- SimpleControllerHandlerAdapter

```
@org.springframework.stereotype.Controller("/simple")
public class SimpleController implements Controller {

  @Override
  public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
response) throws Exception {
      return new ModelAndView("/WEB-INF/simple.jsp");
  }
}
```

11. DispatcherServlet 동작 원리 3부: 커스텀 ViewResolver

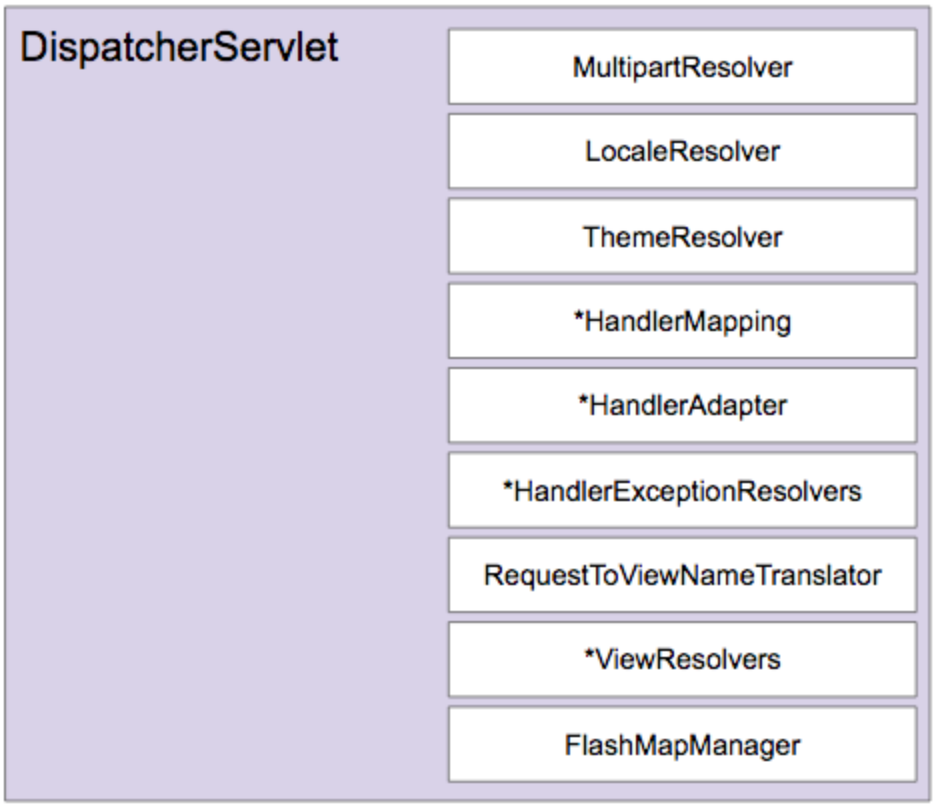
ViewResolver
- InternalResourceViewResolver

InternalResourceViewResolver
- Prefix
- Suffix

```
@Configuration
@ComponentScan
public class WebConfig {

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

}
```

```
@org.springframework.stereotype.Controller("/simple")
public class SimpleController implements Controller {

  @Override
  public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
response) throws Exception {
     return new ModelAndView("/WEB-INF/simple.jsp");
  }
}
```

12. 스프링 MVC 구성 요소



DispatcherSerlvet의 기본 전략

- DispatcherServlet.properties

MultipartResolver

- 파일 업로드 요청 처리에 필요한 인터페이스
- HttpServletRequest를 MultipartHttpServletRequest로 변환해주어 요청이 담고 있는 File을 꺼낼 수 있는 API 제공.

LocaleResolver

- 클라이언트의 위치(Locale) 정보를 파악하는 인터페이스
- 기본 전략은 요청의 accept-language를 보고 판단.

ThemeResolver

- 애플리케이션에 설정된 테마를 파악하고 변경할 수 있는 인터페이스
- 참고: https://memorynotfound.com/spring-mvc-theme-switcher-example/

HandlerMapping

- 요청을 처리할 핸들러를 찾는 인터페이스

HandlerAdapter

- HandlerMapping이 찾아낸 “핸들러”를 처리하는 인터페이스
- 스프링 MVC **확장력**의 핵심

HandlerExceptionResolver

- 요청 처리 중에 발생한 에러 처리하는 인터페이스

RequestToViewNameTranslator

- 핸들러에서 뷰 이름을 명시적으로 리턴하지 않은 경우, 요청을 기반으로 뷰 이름을 판단하는 인터페이스

ViewResolver

- 뷰 이름(string)에 해당하는 뷰를 찾아내는 인터페이스

FlashMapManager

- FlashMap 인스턴스를 가져오고 저장하는 인터페이스
- FlashMap은 주로 리다이렉션을 사용할 때 요청 매개변수를 사용하지 않고 데이터를 전달하고 정리할 때 사용한다.
- redirect:/events

13. 스프링 MVC 동작원리 정리

결국엔 (굉장히 복잡한) 서블릿.
= DispatcherServlet

DispatcherServlet 초기화

1. 특정 타입에 해당하는 빈을 찾는다.
2. 없으면 기본 전략을 사용한다. (DispatcherServlet.properties)

스프링 부트 사용하지 않는 스프링 MVC

- 서블릿 컨네이너(ex, 톰캣)에 등록한 웹 애플리케이션(WAR)에 DispatcherServlet을 등록한다.
    - web.xml에 서블릿 등록
    - 또는 WebApplicationInitializer에 자바 코드로 서블릿 등록 (스프링 3.1+, 서블릿 3.0+)
- 세부 구성 요소는 빈 설정하기 나름.

스프링 부트를 사용하는 스프링 MVC

- 자바 애플리케이션에 내장 톰캣을 만들고 그 안에 DispatcherServlet을 등록한다.
    - 스프링 부트 자동 설정이 자동으로 해줌.
- 스프링 부트의 주관에 따라 여러 인터페이스 구현체를 빈으로 등록한다.

2부. 스프링 MVC 설정

14. 스프링 MVC 구성 요소 직접 빈으로 등록하기

@Configuration을 사용한 자바 설정 파일에 직접 @Bean을 사용해서 등록하기

15. @EnableWebMvc

애노테이션 기반 스프링 MVC를 사용할 때 편리한 웹 MVC 기본 설정

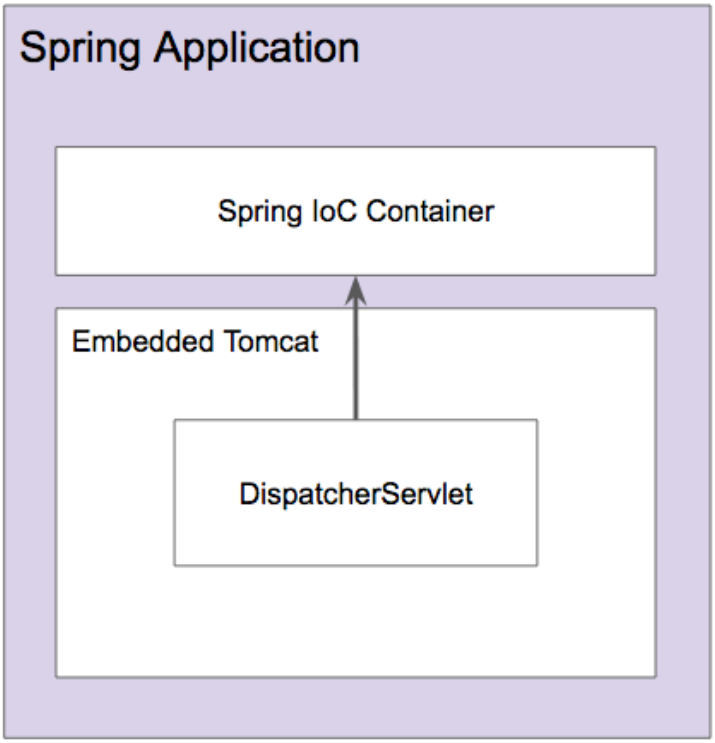
```
@Configuration
@EnableWebMvc
public class WebConfig {

}
```

16. WebMvcConfigurer 인터페이스

@EnableWebMvc가 제공하는 빈을 커스터마이징할 수 있는 기능을 제공하는 인터페이스

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

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

17. 스프링 부트의 스프링 MVC 설정



스프링 부트의 “주관”이 적용된 자동 설정이 동작한다.

- JSP 보다 Thymeleaf 선호
- JSON 지원
- 정적 리소스 지원 (+ 웰컴 페이지, 파비콘 등 지원)

스프링 MVC 커스터마이징

- application.properties
- **@Configuration + Implements WebMvcConfigurer: 스프링 부트의 스프링 MVC 자동설정 + 추가 설정**
- @Configuration + @EnableWebMvc + Imlements WebMvcConfigurer: 스프링 부트의 스프링 MVC 자동설정 사용하지 않음.

18. 스프링 부트에서 JSP 사용하기

“If possible, JSPs should be avoided. There are several known limitations when using them with embedded servlet containers.”

- https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-spring-mvc-template-engines

제약 사항

- JAR 프로젝트로 만들 수 없음, WAR 프로젝트로 만들어야 함
- Java -JAR로 실행할 수는 있지만 “실행가능한 JAR 파일”은 지원하지 않음
- 언더토우(JBoss에서 만든 서블릿 컨테이너)는 JSP를 지원하지 않음
- Whitelabel 에러 페이지를 error.jsp로 오버라이딩 할 수 없음.

참고

- https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-jsp-limitations
- https://github.com/spring-projects/spring-boot/tree/v2.1.1.RELEASE/spring-boot-samples/spring-boot-sample-web-jsp (샘플 프로젝트)

의존성 추가

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```

태그 선언

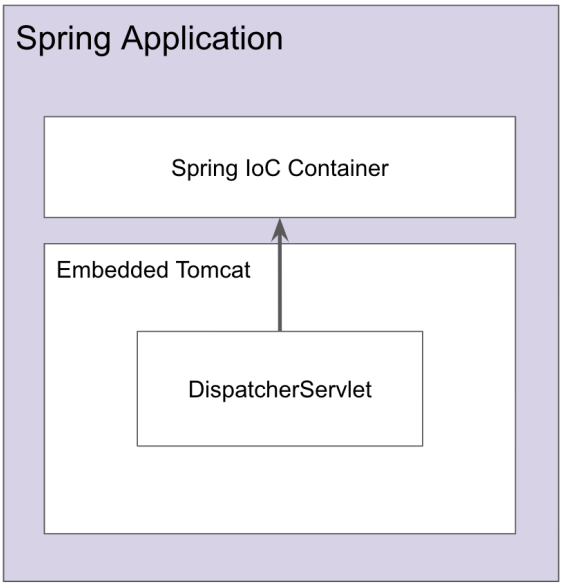
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

application.properties

```
spring.mvc.view.prefix=/WEB-INF/jsp
spring.mvc.view.suffix=.jsp=
```

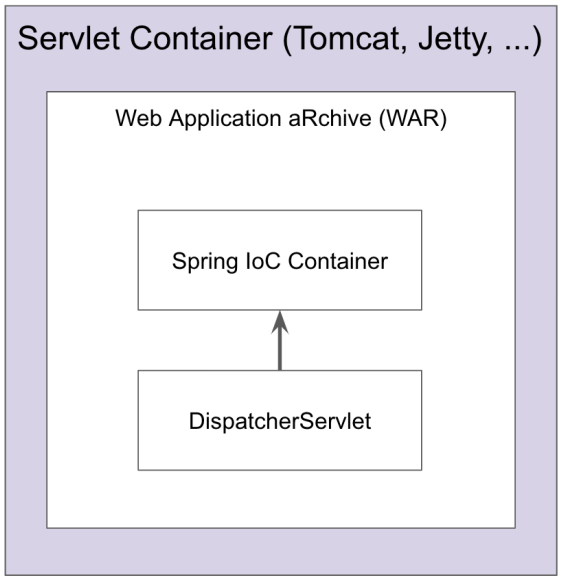
19. WAR 파일 배포하기

java -jar를 사용해서 실행하기



- SpringApplication.run 사용하기

서블릿 컨테이너에 배포하기



- SpringBootServletInitializer (WebApplicationInitializer) 사용하기

20. 포매터 추가하기

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html#addFormatters-org.springframework.format.FormatterRegistry-

Formatter

- Printer: 해당 객체를 (Locale 정보를 참고하여) **문자열**로 어떻게 출력할 것인가
- Parser: 어떤 **문자열**을 (Locale 정보를 참고하여) 객체로 어떻게 변환할 것인가

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/format/Formatter.html

포매터 추가하는 방법 1

- WebMvcConfigurer의 addFormatters(FormatterRegistry) 메소드 정의

포매터 추가하는 방법 2 (스프링 부트 사용시에만 가능 함)

- 해당 포매터를 빈으로 등록

21. 도메인 클래스 컨버터 자동 등록

스프링 데이터 JPA는 스프링 MVC용 도메인 클래스 컨버터를 제공합니다.

도메인 클래스 컨버터

- 스프링 데이터 JPA가 제공하는 Repository를 사용해서 ID에 해당하는 엔티티를 읽어옵니다.

의존성 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>
```

엔티티 맵핑

```
@Entity
public class Person {

  @Id @GeneratedValue
  private Integer id;
...
```

리파지토리 추가

```
public interface PersonRepository extends JpaRepository<Person, Integer> {
}
```

테스트 코드 수정

- 테스트용 이벤트 객체 생성
- 이벤트 리파지토리에 저장
- 저장한 이벤트의 ID로 조회 시도

22. 핸들러 인터셉터 1부: 개념

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html#addInterceptors-org.springframework.web.servlet.config.annotation.InterceptorRegistry-

HandlerInterceptor

- 핸들러 맵핑에 설정할 수 있는 인터셉터
- 핸들러를 실행하기 전, 후(아직 랜더링 전) 그리고 완료(랜더링까지 끝난 이후) 시점에 부가 작업을 하고 싶은 경우에 사용할 수 있다.
- 여러 핸들러에서 반복적으로 사용하는 코드를 줄이고 싶을 때 사용할 수 있다.
    - 로깅, 인증 체크, Locale 변경 등...

boolean preHandle(request, response, **handler**)

- 핸들러 실행하기 전에 호출 됨
- “핸들러"에 대한 정보를 사용할 수 있기 때문에 서블릿 필터에 비해 보다 세밀한 로직을 구현할 수 있다.
- 리턴값으로 계속 다음 인터셉터 또는 핸들러로 요청,응답을 전달할지(true) 응답 처리가 이곳에서 끝났는지(false) 알린다.

void postHandle(request, response, **modelAndView**)

- 핸들러 실행이 끝나고 아직 뷰를 랜더링 하기 이전에 호출 됨
- “뷰"에 전달할 추가적이거나 여러 핸들러에 공통적인 모델 정보를 담는데 사용할 수도 있다.
- 이 메소드는 인터셉터 역순으로 호출된다.
- 비동기적인 요청 처리 시에는 호출되지 않는다.

void afterCompletion(request, response, handler, ex)

- 요청 처리가 완전히 끝난 뒤(뷰 랜더링 끝난 뒤)에 호출 됨
- preHandler에서 true를 리턴한 경우에만 호출 됨
- 이 메소드는 인터셉터 역순으로 호출된다.
- 비동기적인 요청 처리 시에는 호출되지 않는다.

vs 서블릿 필터

- 서블릿 보다 구체적인 처리가 가능하다.
- 서블릿은 보다 일반적인 용도의 기능을 구현하는데 사용하는게 좋다.

참고:

- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/HandlerInterceptor.html
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/AsyncHandlerInterceptor.html

- http://forum.spring.io/forum/spring-projects/web/20146-what-is-the-difference-between-using-a-filter-and-interceptor (스프링 개발자 Mark Fisher의 서블릿 필터와의 차이점에 대한 답변 참고)

23. 핸들러 인터셉터 2부: 만들고 등록하기

핸들러 인터셉터 구현하기

```
public class GreetingInterceptor implements HandlerInterceptor {

  @Override
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
    System.out.println("preHandle 1");
    return true;
  }

  @Override
  public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) throws Exception {
    System.out.println("postHandle 1");
  }

  @Override
  public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
Object handler, Exception ex) throws Exception {
    System.out.println("afterCompletion 1");
  }

}
```

핸들러 인터셉터 등록하기

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

  @Override
  public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new GreetingInterceptor()).order(0);
    registry.addInterceptor(new AnotherInterceptor())
        .addPathPatterns("/keeun")
        .order(-1);
  }
}
```

- 특정 패턴에 해당하는 요청에만 적용할 수도 있다.
- 순서를 지정할 수 있다.

24. 리소스 핸들러

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html#addResourceHandlers-org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry-

이미지, 자바스크립트, CSS 그리고 HTML 파일과 같은 정적인 리소스를 처리하는 핸들러 등록하는 방법

디폴트(Default) 서블릿
- 서블릿 컨테이너가 기본으로 제공하는 서블릿으로 정적인 리소스를 처리할 때 사용한다.
- https://tomcat.apache.org/tomcat-9.0-doc/default-servlet.html

스프링 MVC 리소스 핸들러 맵핑 등록
- 가장 낮은 우선 순위로 등록.
    - 다른 핸들러 맵핑이 “/” 이하 요청을 처리하도록 허용하고
    - 최종적으로 리소스 핸들러가 처리하도록.
- DefaultServletHandlerConfigurer

리소스 핸들러 설정
- 어떤 요청 패턴을 지원할 것인가
- 어디서 리소스를 찾을 것인가
- 캐싱
- ResourceResolver: 요청에 해당하는 리소스를 찾는 전략
    - 캐싱, 인코딩(gzip, brotli), WebJar, ...
- ResourceTransformer: 응답으로 보낼 리소스를 수정하는 전략
    - 캐싱, CSS 링크, HTML5 AppCache, ...

스프링 부트
- 기본 정적 리소스 핸들러와 캐싱 제공

참고
- https://www.slideshare.net/rstoya05/resource-handling-spring-framework-41

25. HTTP 메시지 컨버터 1부: 개요

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html#configureMessageConverters-java.util.List-

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html#extendMessageConverters-java.util.List-

HTTP 메시지 컨버터
- 요청 본문에서 메시지를 읽어들이거나(@RequestBody), 응답 본문에 메시지를 작성할 때(@ResponseBody) 사용한다.

기본 HTTP 메시지 컨버터
- 바이트 배열 컨버터
- 문자열 컨버터
- Resource 컨버터
- Form 컨버터 (폼 데이터 to/from MultiValueMap<String, String>)
- (JAXB2 컨버터)
- (Jackson2 컨버터)
- (Jackson 컨버터)
- (Gson 컨버터)
- (Atom 컨버터)
- (RSS 컨버터)
- ...

설정 방법
- 기본으로 등록해주는 컨버터에 새로운 컨버터 추가하기: extendMessageConverters
- 기본으로 등록해주는 컨버터는 다 무시하고 새로 컨버터 설정하기: configureMessageConverters
- **의존성 추가로 컨버터 등록하기 (추천)**
  - **메이븐 또는 그래들 설정에 의존성을 추가하면 그에 따른 컨버터가 자동으로 등록된다.**
  - **WebMvcConfigurationSupport**
  - (이 기능 자체는 스프링 프레임워크의 기능임, 스프링 부트 아님.)

참고
- https://www.baeldung.com/spring-httpmessageconverter-rest

26. HTTP 메시지 컨버터 2부: JSON

스프링 부트를 사용하지 않는 경우

- 사용하고 싶은 JSON 라이브러리를 의존성으로 추가
- GSON
- JacksonJSON
- JacksonJSON 2

스프링 부트를 사용하는 경우

- 기본적으로 JacksonJSON 2가 의존성에 들어있다.
- **즉, JSON용 HTTP 메시지 컨버터가 기본으로 등록되어 있다.**

참고

- JSON path 문법
- https://github.com/json-path/JsonPath
- http://jsonpath.com/

27. HTTP 메시지 컨버터 3부: XML

OXM(Object-XML Mapper) 라이브러리 중에 스프링이 지원하는 의존성 추가

- JacksonXML
- JAXB

스프링 부트를 사용하는 경우

- **기본으로 XML 의존성 추가해주지 않음.**

JAXB 의존성 추가

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
</dependency>
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-oxm</artifactId>
  <version>${spring-framework.version}</version>
</dependency>
```

Marshaller 등록

```
@Bean
public Jaxb2Marshaller marshaller() {
  Jaxb2Marshaller jaxb2Marshaller = new Jaxb2Marshaller();
  jaxb2Marshaller.setPackagesToScan(Event.class.getPackageName());
  return jaxb2Marshaller;
}
```

도메인 클래스에 @XmlRootElement 애노테이션 추가

참고

- Xpath 문법
- https://www.w3schools.com/xml/xpath_syntax.asp
- https://www.freeformatter.com/xpath-tester.html

28. 그밖에 WebMvcConfigurer 설정

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurer.html

CORS 설정
- Cross Origin 요청 처리 설정
- 같은 도메인에서 온 요청이 아니더라도 처리를 허용하고 싶다면 설정한다.

리턴 값 핸들러 설정
- 스프링 MVC가 제공하는 기본 리턴 값 핸들러 이외에 리턴 핸들러를 추가하고 싶을 때 설정한다.

아큐먼트 리졸버 설정
- 스프링 MVC가 제공하는 기본 아규먼트 리졸버 이외에 커스텀한 아규먼트 리졸버를 추가하고 싶을 때 설정한다.

뷰 컨트롤러
- 단순하게 요청 URL을 특정 뷰로 연결하고 싶을 때 사용할 수 있다.

비동기 설정
- 비동기 요청 처리에 사용할 타임아웃이나 TaskExecutor를 설정할 수 있다.

뷰 리졸버 설정
- 핸들러에서 리턴하는 뷰 이름에 해당하는 문자열을 View 인스턴스로 바꿔줄 뷰 리졸버를 설정한다.

Content Negotiation 설정
- 요청 본문 또는 응답 본문을 어떤 (MIME) 타입으로 보내야 하는지 결정하는 전략을 설정한다.

29. 스프링 MVC 설정 마무리

스프링 MVC 설정은 즉 DispatcherServlet이 사용할 여러 빈 설정.

- HandlerMapper
- HandlerAdapter
- ViewResolver
- ExceptionResolver
- LocaleResolver
- ...

일일히 등록하려니 너무 많고, 해당 빈들이 참조하는 또 다른 객체들까지 설정하려면... 설정할게 너무 많다.

@EnableWebMvc

- 애노테이션 기반의 스프링 MVC 설정 간편화
- WebMvcConfigurer가 제공하는 메소드를 구현하여 커스터마이징할 수 있다.

스프링 부트

- **스프링 부트 자동 설정을 통해 다양한 스프링 MVC 기능을 아무런 설정 파일을 만들지 않아도 제공한다.**
- WebMvcConfigurer가 제공하는 메소드를 구현하여 커스터마이징할 수 있다.
- @EnableWebMvc를 사용하면 스프링 부트 자동 설정을 사용하지 못한다.

스프링 MVC 설정 방법

- 스프링 부트를 사용하는 경우에는 application.properties 부터 시작.
- WebMvcConfigurer로 시작
- @Bean으로 MVC 구성 요소 직접 등록

3부. 스프링 MVC 활용

30. 스프링 MVC 핵심 기술 소개

애노테이션 기반의 스프링 MVC
- 요청 맵핑하기
- 핸들러 메소드
- 모델과 뷰
- 데이터 바인더
- 예외 처리
- 글로벌 컨트롤러

사용할 기술
- 스프링 부트
- 스프링 프레임워크 웹 MVC
- 타임리프

학습 할 애노테이션
- @RequestMapping
    - @GetMapping, @PostMapping, @PutMapping, ...
- @ModelAttribute
- @RequestParam, @RequestHeader
- @PathVariable, @MatrixVariable
- @SessionAttribute, @RequestAttribute, @CookieValue
- @Valid
- @RequestBody, @ResponseBody
- @ExceptionHandler
- @ControllerAdvice

참고:
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-controller

31. HTTP 요청 맵핑하기 1부: 요청 메소드

HTTP Method
- GET, POST, PUT, PATCH, DELETE, ...

GET 요청
- 클라이언트가 서버의 리소스를 요청할 때 사용한다.
- 캐싱 할 수 있다. (조건적인 GET으로 바뀔 수 있다.)
- 브라우저 기록에 남는다.
- 북마크 할 수 있다.
- 민감한 데이터를 보낼 때 사용하지 말 것. (URL에 다 보이니까)
- idempotent

POST 요청
- 클라이언트가 서버의 리소스를 수정하거나 새로 만들 때 사용한다.
- 서버에 보내는 데이터를 POST 요청 본문에 담는다.
- 캐시할 수 없다.
- 브라우저 기록에 남지 않는다.
- 북마크 할 수 없다.
- 데이터 길이 제한이 없다.

PUT 요청
- URI에 해당하는 데이터를 새로 만들거나 수정할 때 사용한다.
- POST와 다른 점은 “URI”에 대한 의미가 다르다.
    - POST의 URI는 보내는 데이터를 처리할 리소스를 지칭하며
    - PUT의 URI는 보내는 데이터에 해당하는 리소스를 지칭한다.
- Idempotent

PATCH 요청
- PUT과 비슷하지만, 기존 엔티티와 새 데이터의 차이점만 보낸다는 차이가 있다.
- Idempotent

DELETE 요청
- URI에 해당하는 리소스를 삭제할 때 사용한다.
- Idempotent

스프링 웹 MVC에서 HTTP method 맵핑하기
- @RequestMapping(method=RequestMethod.GET)
- @RequestMapping(method={RequestMethod.GET, RequestMethod.POST})
- @GetMapping, @PostMapping, ...

참고

- https://www.w3schools.com/tags/ref_httpmethods.asp
- https://tools.ietf.org/html/rfc2616#section-9.3
- https://tools.ietf.org/html/rfc2068

32. HTTP 요청 맵핑하기 2부: URI 패턴 맵핑

URI, URL, URN 햇갈린다

- https://stackoverflow.com/questions/176264/what-is-the-difference-between-a-uri-a-url-and-a-urn

요청 식별자로 맵핑하기

- @RequestMapping은 다음의 패턴을 지원합니다.
- ?: 한 글자 (“/author/???” => “/author/123”)
- *: 여러 글자 (“/author/*” => “/author/keesun”)
- **: 여러 패스 (“/author/** => “/author/keesun/book”)

클래스에 선언한 @RequestMapping과 조합

- 클래스에 선언한 URI 패턴뒤에 이어 붙여서 맵핑합니다.

정규 표현식으로 맵핑할 수도 있습니다.

- /{name:정규식}

패턴이 중복되는 경우에는?

- 가장 구체적으로 맵핑되는 핸들러를 선택합니다.

URI 확장자 맵핑 지원

- 이 기능은 권장하지 않습니다. (스프링 부트에서는 기본으로 이 기능을 사용하지 않도록 설정 해 줌)
    - 보안 이슈 (RFD Attack)
    - URI 변수, Path 매개변수, URI 인코딩을 사용할 때 할 때 불명확 함.

RFD Attack

- https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/reflected-file-download-a-new-web-attack-vector/
- https://www.owasp.org/index.php/Reflected_File_Download
- https://pivotal.io/security/cve-2015-5211

33. HTTP 요청 맵핑하기 3부: 미디어 타입 맵핑

특정한 타입의 데이터를 담고 있는 요청만 처리하는 핸들러

- @RequestMapping(**consumes**=MediaType.APPLICATION_JSON_UTF8_VALUE)
- Content-Type 헤더로 필터링
- 매치 되는 않는 경우에 415 Unsupported Media Type 응답

특정한 타입의 응답을 만드는 핸들러

- @RequestMapping(**produces**="application/json")
- Accept 헤더로 필터링 (하지만 살짝... 오묘함)
- 매치 되지 않는 경우에 406 Not Acceptable 응답

문자열을 입력하는 대신 MediaType을 사용하면 상수를 (IDE에서) 자동 완성으로 사용할 수 있다.

클래스에 선언한 @RequestMapping에 사용한 것과 조합이 되지 않고 메소드에 사용한 @RequestMapping의 설정으로 덮어쓴다.

Not (!)을 사용해서 특정 미디어 타입이 아닌 경우로 맵핑 할 수도 있다.

34. HTTP 요청 맵핑하기 4부: 헤더와 매개변수

특정한 헤더가 있는 요청을 처리하고 싶은 경우

- @RequestMapping(**headers** = “key”)

특정한 헤더가 없는 요청을 처리하고 싶은 경우

- @RequestMapping(**headers** = “**!**key”)

특정한 헤더 키/값이 있는 요청을 처리하고 싶은 경우

- @RequestMapping(**headers** = “key=value”)

특정한 요청 매개변수 키를 가지고 있는 요청을 처리하고 싶은 경우

- @RequestMapping(**params** = “a”)

특정한 요청 매개변수가 없는 요청을 처리하고 싶은 경우

- @RequestMapping(**params** = “**!**a”)

특정한 요청 매개변수 키/값을 가지고 있는 요청을 처리하고 싶은 경우

- @RequestMapping(**params** = “a=b”)

35. HTTP 요청 맵핑하기 5부: HEAD와 OPTIONS 요청 처리

우리가 구현하지 않아도 스프링 웹 MVC에서 자동으로 처리하는 HTTP Method

- HEAD
- OPTIONS

HEAD

- GET 요청과 동일하지만 **응답 본문을 받아오지 않고 응답 헤더만** 받아온다.

OPTIONS

- 사용할 수 있는 HTTP Method 제공
- 서버 또는 특정 리소스가 제공하는 기능을 확인할 수 있다.
- 서버는 Allow 응답 헤더에 사용할 수 있는 HTTP Method 목록을 제공해야 한다.

참고

- https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
- https://github.com/spring-projects/spring-framework/blob/master/spring-test/src/test/java/org/springframework/test/web/servlet/samples/standalone/resultmatchers/HeaderAssertionTests.java

36. HTTP 요청 맵핑하기 6부: 커스텀 애노테이션

@RequestMapping 애노테이션을 메타 애노테이션으로 사용하기
- @GetMapping 같은 커스텀한 애노테이션을 만들 수 있다.

메타(Meta) 애노테이션
- 애노테이션에 사용할 수 있는 애노테이션
- 스프링이 제공하는 대부분의 애노테이션은 메타 애노테이션으로 사용할 수 있다.

조합(Composed) 애노테이션
- 한개 혹은 여러 메타 애노테이션을 조합해서 만든 애노테이션
- 코드를 간결하게 줄일 수 있다.
- 보다 구체적인 의미를 부여할 수 있다.

@Retention
- 해당 애노테이션 정보를 언제까지 유지할 것인가.
- Source: 소스 코드까지만 유지. 즉, 컴파일 하면 해당 애노테이션 정보는 사라진다는 이야기.
- Class: 컴파인 한 .class 파일에도 유지. 즉 런타임 시, 클래스를 메모리로 읽어오면 해당 정보는 사라진다.
- Runtime: 클래스를 메모리에 읽어왔을 때까지 유지! 코드에서 이 정보를 바탕으로 특정 로직을 실행할 수 있다.

@Target
- 해당 애노테이션을 어디에 사용할 수 있는지 결정한다.

@Documented
- 해당 애노테이션을 사용한 코드의 문서에 그 애노테이션에 대한 정보를 표기할지 결정한다.

메타 애노테이션
- https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-meta-annotations
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/core/annotation/AliasFor.html

37. HTTP 요청 맵핑하기 7부: 맵핑 연습 문제

다음 요청을 처리할 수 있는 핸들러 메소드를 맵핑하는 @RequestMapping (또는 @GetMapping, @PostMapping 등)을 정의하세요.

1. GET /events
2. GET /events/1,
   GET /events/2,
   GET /events/3,
   ...
3. POST /events CONTENT-TYPE: application/json ACCEPT: application/json
4. DELETE /events/1,
   DELETE /events/2,
   DELETE /events/3,
   ...
5. PUT /events/1 CONTENT-TYPE: application/json ACCEPT: application/json,
   PUT /events/2 CONTENT-TYPE: application/json ACCEPT: application/json,
   ...

38. 핸들러 메소드 1부: 지원하는 메소드 아규먼트와 리턴 타입

핸들러 메소드 아규먼트: 주로 요청 그 자체 또는 요청에 들어있는 정보를 받아오는데 사용한다.

핸들러 메소드 아규먼트	설명
WebRequest NativeWebRequest ServletRequest(Response) HttpServletRequest(Response)	요청 또는 응답 자체에 접근 가능한 API
InputStream Reader OutputStream Writer	요청 본문을 읽어오거나, 응답 본문을 쓸 때 사용할 수 있는 API
PushBuilder	스프링 5, HTTP/2 리소스 푸쉬에 사용
HttpMethod	GET, POST, ... 등에 대한 정보
Locale TimeZone ZoneId	LocaleResolver가 분석한 요청의 Locale 정보
@PathVariable	URI 템플릿 변수 읽을 때 사용
@MatrixVariable	URI 경로 중에 키/값 쌍을 읽어 올 때 사용
@RequestParam	서블릿 요청 매개변수 값을 선언한 메소드 아규먼트 타입으로 변환해준다. 단순 타입인 경우에 이 애노테이션을 생략할 수 있다.
@RequestHeader	요청 헤더 값을 선언한 메소드 아규먼트 타입으로 변환해준다.
...	...

참고:
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-arguments

핸들러 메소드 리턴: 주로 응답 또는 모델을 랜더링할 뷰에 대한 정보를 제공하는데 사용한다.

@ResponseBody	리턴 값을 HttpMessageConverter를 사용해 응답 본문으로 사용한다.
HttpEntity	응답 본문 뿐 아니라 헤더 정보까지, 전체 응답을 만들 때

ReponseEntity	사용한다.
String	ViewResolver를 사용해서 뷰를 찾을 때 사용할 뷰 이름.
View	암묵적인 모델 정보를 랜더링할 뷰 인스턴스
Map Model	(RequestToViewNameTranslator를 통해서) 암묵적으로 판단한 뷰 랜더링할 때 사용할 모델 정보
@ModelAttribute	(RequestToViewNameTranslator를 통해서) 암묵적으로 판단한 뷰 랜더링할 때 사용할 모델 정보에 추가한다. 이 애노테이션은 생략할 수 있다.
...	...

참고:
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-return-types

39. 핸들러 메소드 2부: URI 패턴

@PathVariable

- 요청 URI 패턴의 일부를 핸들러 메소드 아규먼트로 받는 방법.
- 타입 변환 지원.
- (기본)값이 반드시 있어야 한다.
- Optional 지원.

@MatrixVariable

- 요청 URI 패턴에서 키/값 쌍의 데이터를 메소드 아규먼트로 받는 방법
- 타입 변환 지원.
- (기본)값이 반드시 있어야 한다.
- Optional 지원.
- 이 기능은 기본적으로 비활성화 되어 있음. 활성화 하려면 다음과 같이 설정해야 함.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

  @Override
  public void configurePathMatch(PathMatchConfigurer configurer) {
    UrlPathHelper urlPathHelper = new UrlPathHelper();
    urlPathHelper.setRemoveSemicolonContent(false);
    configurer.setUrlPathHelper(urlPathHelper);
  }
}
```

참고:
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-typeconversion
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-matrix-variables

40. 핸들러 메소드 3부: @RequestMapping

@RequestParam

- 요청 매개변수에 들어있는 단순 타입 데이터를 메소드 아규먼트로 받아올 수 있다.
- 값이 반드시 있어야 한다.
    - required=false 또는 Optional을 사용해서 부가적인 값으로 설정할 수도 있다.
- String이 아닌 값들은 타입 컨버전을 지원한다.
- Map<String, String> 또는 MultiValueMap<String, String>에 사용해서 모든 요청 매개변수를 받아 올 수도 있다.
- 이 애노테이션은 생략 할 수 잇다.

요청 매개변수란?

- 쿼리 매개변수
- 폼 데이터

참고:

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-requestparam

41. 핸들러 메소드 4부: 폼 서브밋 (타임리프)

폼을 보여줄 요청 처리

- GET /events/form
- 뷰: events/form.html
- 모델: “event”, new Event()

타임리프

- @{}: URL 표현식
- ${}: variable 표현식
- *{}: selection 표현식

참고

- https://www.thymeleaf.org/doc/articles/standarddialect5minutes.html
- https://www.getpostman.com/downloads/

42. 핸들러 메소드 5부: @ModelAttribute

@ModelAttribute

- 여러 곳에 있는 단순 타입 데이터를 복합 타입 객체로 받아오거나 해당 객체를 새로 만들 때 사용할 수 있다.
- 여러 곳? URI 패스, 요청 매개변수, 세션 등
- 생략 가능

값을 바인딩 할 수 없는 경우에는?

- BindException 발생 400 에러

바인딩 에러를 직접 다루고 싶은 경우

- BindingResult 타입의 아규먼트를 바로 오른쪽에 추가한다.

바인딩 이후에 검증 작업을 추가로 하고 싶은 경우

- @Valid 또는 @Validated 애노테이션을 사용한다.

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-modelattrib-method-args

43. 핸들러 메소드 6부: @Validated

스프링 MVC 핸들러 메소드 아규먼트에 사용할 수 있으며 validation group이라는 힌트를 사용할 수 있다.

@Valid 애노테이션에는 그룹을 지정할 방법이 없다.

@Validated는 스프링이 제공하는 애노테이션으로 그룹 클래스를 설정할 수 있다.

44. 핸들러 메소드 7부: 폼 서브밋 (에러 처리)

바인딩 에러 발생 시 Model에 담기는 정보

- Event
- BindingResult.event

타임리프 사용시 바인딩 에러 보여주기

- https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html#field-errors

```
<p th:if="${#fields.hasErrors('limit')}" th:errors="*{limit}">Incorrect date</p>
```

Post / Redirect / Get 패턴

- https://en.wikipedia.org/wiki/Post/Redirect/Get
- Post 이후에 브라우저를 리프래시 하더라도 폼 서브밋이 발생하지 않도록 하는 패턴

타임리프 목록 보여주기

- https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html#listing-seed-starter-data

```
<a th:href="@{/events/form}">Create New Event</a>
<div th:unless="${#lists.isEmpty(eventList)}">
   <ul th:each="event: ${eventList}">
      <p th:text="${event.Name}">Event Name</p>
   </ul>
</div>
```

45. 핸들러 메소드 8부: @SessionAttributes

모델 정보를 HTTP 세션에 저장해주는 애노테이션

- HttpSession을 직접 사용할 수도 있지만
- 이 애노테이션에 설정한 이름에 해당하는 모델 정보를 자동으로 세션에 넣어준다.
- @ModelAttribute는 세션에 있는 데이터도 바인딩한다.
- 여러 화면(또는 요청)에서 사용해야 하는 객체를 공유할 때 사용한다.

SessionStatus를 사용해서 세션 처리 완료를 알려줄 수 있다.

- 폼 처리 끝나고 세션을 비울 때 사용한다.

46. 핸들러 메소드 9부: 멀티 폼 서브밋

세션을 사용해서 여러 폼에 걸쳐 데이터를 나눠 입력 받고 저장하기

- 이벤트 이름 입력받고
- 이벤트 제한 인원 입력받고
- 서브밋 -> 이벤트 목록으로!

완료된 경우에 세션에서 모델 객체 제거하기

- SessionStatus

47. 핸들러 메소드 10부: @SessionAttribute

HTTP 세션에 들어있는 값 참조할 때 사용

- HttpSession을 사용할 때 비해 타입 컨버전을 자동으로 지원하기 때문에 조금 편리함.
- HTTP 세션에 데이터를 넣고 빼고 싶은 경우에는 HttpSession을 사용할 것.

@SessionAttributes와는 다르다.

- @SessionAttributes는 해당 컨트롤러 내에서만 동작.
    - 즉, 해당 컨트롤러 안에서 다루는 특정 모델 객체를 세션에 넣고 공유할 때 사용.
- @SessionAttribute는 컨트롤러 밖(인터셉터 또는 필터 등)에서 만들어 준 세션 데이터에 접근할 때 사용한다.

48. 핸들러 메소드 11부: RedirectAttributes

리다이렉트 할 때 기본적으로 Model에 들어있는 primitive type 데이터는 URI 쿼리 매개변수에 추가된다.

- 스프링 부트에서는 이 기능이 기본적으로 비활성화 되어 있다.
- Ignore-default-model-on-redirect 프로퍼티를 사용해서 활성화 할 수 있다.

원하는 값만 리다이렉트 할 때 전달하고 싶다면 RedirectAttributes에 명시적으로 추가할 수 있다.

리다이렉트 요청을 처리하는 곳에서 쿼리 매개변수를 @RequestParam 또는 @ModelAttribute로 받을 수 있다.

49. 핸들러 메소드 12부: Flash Attributes

주로 리다이렉트시에 데이터를 전달할 때 사용한다.

- 데이터가 URI에 노출되지 않는다.
- 임의의 객체를 저장할 수 있다.
- 보통 HTTP 세션을 사용한다.

리다이렉트 하기 전에 데이터를 HTTP 세션에 저장하고 리다이렉트 요청을 처리 한 다음 그 즉시 제거한다.

RedirectAttributes를 통해 사용할 수 있다.

XPath

- https://www.w3schools.com/xml/xpath_syntax.asp
- https://www.freeformatter.com/xpath-tester.html#ad-output

50. 핸들러 메소드 13부: MultipartFile

MultipartFile

- 파일 업로드시 사용하는 메소드 아규먼트
- MultipartResolver 빈이 설정 되어 있어야 사용할 수 있다. (스프링 부트 자동 설정이 해줌)
- POST multipart/form-data 요청에 들어있는 파일을 참조할 수 있다.
- List<MultipartFile> 아규먼트로 여러 파일을 참조할 수도 있다.

파일 업로드 폼

```
<form method="POST" enctype="multipart/form-data" action="#" th:action="@{/file}">
    File: <input type="file" name="file"/>
    <input type="submit" value="Upload"/>
</form>
```

파일 업로드 처리 핸들러

```
    @PostMapping("/file")
    public String uploadFile(@RequestParam MultipartFile file,
                             RedirectAttributes attributes) {
        String message = file.getOriginalFilename() + " is uploaded.";
        System.out.println(message);
        attributes.addFlashAttribute("message", message);
        return "redirect:/events/list";
    }
```

메시지 출력

```
<div th:if="${message}">
    <h2 th:text="${message}"/>
</div>
```

파일 업로드 관련 스프링 부트 설정

- MultipartAutoConfiguration
- MultipartProperties

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-multipart-forms
- https://spring.io/guides/gs/uploading-files/

51. 핸들러 메소드 14부: ResponseEntity

파일 리소스를 읽어오는 방법

- 스프링 ResourceLoader 사용하기

파일 다운로드 응답 헤더에 설정할 내용

- Content-Disposition: 사용자가 해당 파일을 받을 때 사용할 파일 이름
- Content-Type: 어떤 파일인가
- Content-Length: 얼마나 큰 파일인가

파일의 종류(미디어 타입) 알아내는 방법

- http://tika.apache.org/

ResponseEntity

- 응답 상태 코드
- 응답 헤더
- 응답 본문

```
    @GetMapping("/file/{filename}")
    @ResponseBody
    public ResponseEntity<Resource> downloadFile(@PathVariable String filename) throws
IOException {
        Resource resource = resourceLoader.getResource("classpath:" + filename);
        File file = resource.getFile();
        Tika tika = new Tika();
        String type = tika.detect(file);
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachement; filename=\"" +
resource.getFilename() + "\"")
                .header(HttpHeaders.CONTENT_TYPE, type)
                .header(HttpHeaders.CONTENT_LENGTH, String.valueOf(file.length()))
                .body(resource);
    }
```

참고

- https://spring.io/guides/gs/uploading-files/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition
- https://www.baeldung.com/java-file-mime-type

52. 핸들러 메소드 15부: @RequestBody & HttpEntity

@RequestBody

- 요청 본문(body)에 들어있는 데이터를 HttpMessageConveter를 통해 변환한 객체로 받아올 수 있다.
- @Valid 또는 @Validated를 사용해서 값을 검증 할 수 있다.
- BindingResult 아규먼트를 사용해 코드로 바인딩 또는 검증 에러를 확인할 수 있다.

HttpMessageConverter

- 스프링 MVC 설정 (WebMvcConfigurer)에서 설정할 수 있다.
- configureMessageConverters: 기본 메시지 컨버터 대체
- extendMessageConverters: 메시지 컨버터에 추가
- 기본 컨버터
    - WebMvcConfigurationSupport.addDefaultHttpMessageConverters

HttpEntity

- @RequestBody와 비슷하지만 추가적으로 요청 헤더 정보를 사용할 수 있다.

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-requestbody
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-httpentity

53. 핸들러 메소드 16부: @ResponseBody & ResponseEntity

@ResponseBody

- 데이터를 HttpMessageConverter를 사용해 응답 본문 메시지로 보낼 때 사용한다.
- @RestController 사용시 자동으로 모든 핸들러 메소드에 적용 된다.

ResponseEntity

- 응답 헤더 상태 코드 본문을 직접 다루고 싶은 경우에 사용한다.

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-responsebody
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-responseentity

54. 핸들러 메소드 17부: 정리

다루지 못한 내용

- @JsonView: https://www.youtube.com/watch?v=5QyXswB_Usg&t=188s
- PushBuidler: HTTP/2, 스프링 5

과제

- 프로젝트 코드 분석
- https://github.com/spring-projects/spring-petclinic
- 컨트롤러 코드 위주로...

55. 모델: @ModelAttribute 또 다른 사용법

@ModelAttribute의 다른 용법

- @RequestMapping을 사용한 핸들러 메소드의 아규먼트에 사용하기 (이미 살펴봤습니다.)
- @Controller 또는 @ControllerAdvice (이 애노테이션은 뒤에서 다룹니다.)를 사용한 클래스에서 모델 정보를 초기화 할 때 사용한다.
- @RequestMapping과 같이 사용하면 해당 메소드에서 리턴하는 객체를 모델에 넣어준다.
    - RequestToViewNameTranslator

@ModelAttribute 메소드

```
@ModelAttribute
public void subjects(Model model) {
    model.addAttribute("subjects", List.of("study", "seminar", "hobby", "social"));
}
```

56. DataBinder: @InitBinder

특정 컨트롤러에서 바인딩 또는 검증 설정을 변경하고 싶을 때 사용

```
@InitBinder
public void initEventBinder(WebDataBinder webDataBinder) {
    webDataBinder.setDisallowedFields("id");
}
```

바인딩 설정

- webDataBinder.setDisallowedFields();

포매터 설정

- webDataBinder.addCustomFormatter();

Validator 설정

- webDataBinder.addValidators();

특정 모델 객체에만 바인딩 또는 Validator 설정을 적용하고 싶은 경우

- @InitBinder(“event”)

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-initbinder
- https://github.com/spring-projects/spring-petclinic/blob/master/src/main/java/org/springframework/samples/petclinic/owner/PetController.java

57. 예외 처리 핸들러: @ExceptionHandler

특정 예외가 발생한 요청을 처리하는 핸들러 정의

- 지원하는 메소드 아규먼트 (해당 예외 객체, 핸들러 객체, ...)
- 지원하는 리턴 값
- REST API의 경우 응답 본문에 에러에 대한 정보를 담아주고, 상태 코드를 설정하려면 ResponseEntity를 주로 사용한다.

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-exceptionhandler

58. 전역 컨트롤러: @(Rest)ControllerAdvice

예외 처리, 바인딩 설정, 모델 객체를 모든 컨트롤러 전반에 걸쳐 적용하고 싶은 경우에 사용한다.

- @ExceptionHandler
- @InitBinder
- @ModelAttributes

적용할 범위를 지정할 수도 있다.

- 특정 애노테이션을 가지고 있는 컨트롤러에만 적용하기
- 특정 패키지 이하의 컨트롤러에만 적용하기
- 특정 클래스 타입에만 적용하기

참고

- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-controller-advice

59. 스프링 MVC 강좌 마무리

시간 관계 상 살펴보지 못한 내용

- 비동기 요청 처리
- CORS 설정
- HTTP/2
- 웹 소켓
- 웹 플럭스
- ...

하지만 이번 강좌가 여러분이 직접 학습해 나갈 수 있는 디딤돌이 되었길 바랍니다.

감사합니다.