스프링 MVC FileUpload

FileUpload 구 현

- 스프링은 서블릿 3.0 사양에서 제공하는 방식과 Apache Commons FileUpload 컴포넌트를 사용하는 방식 모두 지원
- 서블릿 3.0 방식 설정
  - 파일 업로드 설정 (web.xml)

```
<servlet>
  <servlet-name>product</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/product-context.xml</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
  <multipart-config>
    <max-file-size>20848820</max-file-size>
    <max-request-size>418018841</max-request-size>
    <file-size-threshold>1048576</file-size-threshold>
  </multipart-config>
</servlet>
```

  - 파일 업로드 처리 빈 등록(빈 설정 파일)

```
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.support.StandardServletMultipartResolver"/>
```

FileUpload 구 현

- Apache Commons FileUpload 컴포넌트 사용 설정
  - 의존성 패키지 등록 (pom.xml)

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

  - 파일 업로드 처리 빈 등록 (빈 설정 파일)

```
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <beans:property name="maxUploadSize" value="20848820"/>
</beans:bean>
```

FileUpload 구 현

- 파일 업로드 처리 (공통)

```
@RequestMapping(value="/edit.do", method=RequestMethod.POST)
public String add(@Valid @ModelAttribute("product") ProductModel model,
        BindingResult bindingResult,
        @RequestParam(value="image", required=false) MultipartFile image,
        HttpServletRequest request) {
    if(bindingResult.hasErrors())
        return "edit";
    try {
        if(!image.isEmpty()) {
            if(!image.getContentType().equals("image/jpeg")) {
                throw new ImageUploadException("JPEG 이미지만 선택해주세요.");
            }
            String webRootPath = request.getSession().getServletContext().getRealPath("/");
            String filePath = webRootPath +"resources/" + image.getOriginalFilename();
            File file = new File(filePath);
            FileUtils.writeByteArrayToFile(file, image.getBytes());
            logger.info("업로드 파일 : " + filePath);
        }
    }
    catch(Exception e) {
        bindingResult.reject(e.getMessage());
        return "edit";
    }
    productService.saveProduct(model.buildDomain());
    return "result";
}
```

사용자 정의 View

- InternalResourceViewResolver는 JstlView를 사용해서 응답 컨텐츠 생성
  - JstlView는 일종의 JSP Parser 역할 ➔ServletContainer이 JSP 처리 기능과 동일
- JSP 기반의 HTML 응답 컨텐츠가 아닌 다른 종류의 응답 컨텐츠를 만들어야 할 경우 적절한 다른 View를 사용하거나 직접 View 클래스 구현 필요
- 사용자 정의 View 클래스 구현 방법
  - View 인터페이스 구현 (render 메서드)
  - AbstractView 추상 클래스 상속 (renderMergedOutputModel)

파 일 다 운 로 드

- 파일 다운로드는 HTML이 아닌 컨텐츠 응답
  - 사용자 정의 View를 통해 구현 가능

```
public class DownloadView extends AbstractView {

    @Override
    protected void renderMergedOutputModel(
            Map<String, Object> params,
            HttpServletRequest request, HttpServletResponse response) throws Exception {

        //다운로드 처리
        //1. 브라우저에게 처리할 수 없는 컨텐츠로 알려주어서 다운로드 처리하도록 지정
        response.setContentType("application/octet-stream");

        //2. 다운로드 처리할 때 필요한 정보 제공
        response.addHeader( "Content-Disposition", "Attachment;Filename=\"web.xml\"" );

        //3. 파일을 응답스트림에 기록
        String file2 = request.getSession().getServletContext().getRealPath("/WEB-INF/web.xml");
        BufferedInputStream istream = new BufferedInputStream(new FileInputStream(file2));
        BufferedOutputStream ostream = new BufferedOutputStream(response.getOutputStream());
        while (true) {
            int data = istream.read();
            if (data != -1) ostream.write(data);
            else break;
        }
        istream.close();
        ostream.close();
    }

}
```