

Spring Framework

- 스프링 MVC를 이용한 웹 요청 처리_2

■ CONTENTS

- 파일 업로드
- @PathVariable을 이용한 경로 변수 처리
- HanndlerInterceptor
- 예외처리

■ 파일 업로드 처리

• MultipartResolver 설정

- Multipart 지원 기능을 사용하려면 **MultipartResolver**를 스프링 설정 파일에 등록
- Multipart 형식으로 데이터가 전송된 경우 해당 데이터를 스프링 MVC에서 사용 가능하도록 변환
- 스프링이 제공하는 MultipartResolver는 CommonsMultipartResolver이다
- **Commons FileUpload Api** 를 이용해서 Multipart를 처리해 준다.
- CommonsMultipartResolver를 MultipartResolver로 사용하기 위해서는
"multipartResolver" 이름의 빈으로 등록

```
<!-- 파일업로드 처리를 위한 multipartResolver bean 등록 -->
<beans:bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <beans:property name="defaultEncoding">
        <beans:value>utf-8</beans:value>
    </beans:property>
</beans:bean>
```

■ 파일 업로드 처리

– CommonsMultipartResolver 클래스의 프로퍼티

프로퍼티	타입	설명
maxUploadSize	long	최대 업로드 가능한 바이트 크기, -1은 제한이 없음을 의미 한다.
maxInMemorySize	int	디스크에 임시파일을 생성하기 전에 메모리에 보관할 수 있는 최대 바이트 크기, 기본값은 1024 바이트 이다.
defaultEncoding	String	요청을 파싱할 때 사용할 캐릭터 인코딩, 지정하지 않을 경우, <code>HttpServletRequest.setCharacterEncoding()</code> 메서드를 지정한 캐릭터 셋이 사용된다. 아무 값도 없을 경우 ISO-8859-1을 사용

pom.xml

```
<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.4</version>
    </dependency>

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.7</version>
    </dependency>
```

■ 파일 업로드 처리

- **@RequestParam 어노테이션을 이용한 업로드 파일 접근**
 - 업로드한 파일을 전달 받는 방법은 @RequestParam 어노테이션이 적용된 MultipartFile 타입의 파라미터를 사용.

```
package com.aia.firstspring.controller;

import java.io.File;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.multipart.MultipartHttpServletRequest;

import com.aia.firstspring.domain.ReportUploadRequest;

@Controller
public class FileUploadController {
    final String URI = "/uploadfile";
    @RequestMapping("/upload/uploadForm")
    public String uploadForm() {
        return "upload/uploadForm";
    }
}
```

■ 파일 업로드 처리

- **MultipartHttpServletRequest** 인터페이스는 스프링이 제공하는 인터페이스로, Multipart 요청이 들어올 때 내부적으로 원본 **HttpServletRequest** 대신 사용되는 인터페이스.
- 웹 요청 정보를 구하기 위한 **getParameter()**나 **getHeader()**와 같은 메서드를 사용, 추가로 **MultipartRequest** 인터페이스가 제공한 **Multipart** 관련 메서드를 사용할 수 있다.
- **MultipartRequest** 인터페이스가 제공하는 메서드

메서드	설명
Iterator<String> getFileNames()	Djqlhem 된 파일의 이름 목록을 제공하는 Iterator를 구함
MultipartFile getFile(String Name)	파라미터 이름이 name인 업로드 파일 정보를 구함
List<MultipartFile> getFiles(String name)	파라미터 이름이 name인 업로드 파일 정보 목록을 구함
MapM<String, MultipartFile> getFileMap()	파라미터이름을 키로 파라미터에 해당하는 파일 정보를 값으로 하는 Map을 구한다.

■ 파일 업로드 처리

- **@RequestParam 어노테이션을 이용한 업로드 파일 접근**
 - 업로드한 파일을 전달 받는 방법은 @RequestParam 어노테이션이 적용된 MultipartFile 타입의 파라미터를 사용.

```
@RequestMapping("/upload/upload1")
public String upload1(
    @RequestParam("sn") String sn,
    @RequestParam("report") MultipartFile report,
    Model model,
    HttpServletRequest request
) throws IllegalStateException, IOException {

    System.out.println(report.getOriginalFilename());

    model.addAttribute("sno", sn);
    model.addAttribute("reportFile", report.getOriginalFilename());

    // 파일의 저장
    report.transferTo(getFile(request, URI, report.getOriginalFilename()));

    return "upload/uploadComplete";
}
```


■ 파일 업로드 처리

- MultipartFile인터페이스는 스프링에서 업로드 한 파일을 표현 할 때 사용되는 인터페이스
- MultipartFile인터페이스를 이용해서 업로드 한 파일의 이름, 실제 데이터, 파일의 크기 등을 구할 수 있다.

■ 파일 업로드 처리

- **MultipartFile 인터페이스를 사용**

- MultipartFile 인터페이스는 업로드 한 파일 정보 및 파일 데이터를 표현하기 위한 용도로 사용.
- MultipartFile 인터페이스의 주요 메서드

메서드	설명
String getName()	파라미터 이름을 구한다.
String getOriginalFilename()	업로드 한 파일의 이름을 구한다.
boolean isEmpty()	업로드 한 파일이 존재하지 않는 경우 true를 리턴 한다.
long getSize()	업로드한 파일의 크기를 구한다.
byte[] getBytes() throws IOException	업로드 한 파일 데이터를 구한다.
InputStream getInputStream()	InputStream을 구한다.
void transferTo(File dest)	업로드 한 파일 데이터를 지정한 파일에 저장한다.

■ 파일 업로드 처리

- **MultipartHttpServletRequest** 이용한 업로드 파일 접근
 - MultipartHttpServletRequest 인터페이스를 사용.

```
@RequestMapping("/upload/upload2")
public String upload2(
    MultipartHttpServletRequest request,
    Model model
) throws IllegalStateException, IOException {

    String sn = request.getParameter("sn");
    MultipartFile report = request.getFile("report");

    System.out.println(report.getOriginalFilename());

    // 파일 업로드
    report.transferTo(getFile(request, URI, report.getOriginalFilename()));

    model.addAttribute("sno", sn);
    model.addAttribute("reportFile", report.getOriginalFilename());

    return "upload/uploadComplete";
}
```

■ 파일 업로드 처리

- 커맨드 객체를 통한 업로드 파일 접근
 - 커맨드 객체를 이용해도 업로드 한 파일을 전달받을 수 있다.
 - 커맨드 클래스에 파라미터와 동일한 이름의 **MultipartFile** 타입 프로퍼티를 추가해 주어야 함.

```
@RequestMapping("/upload/upload3")
public String upload3(
    ReportUploadRequest uploadRequest,
    Model model,
    HttpServletRequest request
) throws IllegalStateException, IOException {
    System.out.println(uploadRequest.getReport().getOriginalFilename());
    model.addAttribute("sno", uploadRequest.getSn());
    model.addAttribute("reportFile",
        uploadRequest.getReport().getOriginalFilename());

    uploadRequest.getReport().transferTo(getFile(request, URI,
        uploadRequest.getReport().getOriginalFilename()));

    return "upload/uploadComplete";
}
```

■ 파일 업로드 처리

스프링 설정 파일 등록

...

```
<!-- 파일업로드 처리를 위한 multipartResolver bean 등록 -->  
<beans:bean id="multipartResolver"  
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
  <beans:property name="defaultEncoding">  
    <beans:value>utf-8</beans:value>  
  </beans:property>  
</beans:bean>
```

....

■ 파일 업로드 처리

mvctest.controller.ReportCommand.java

```
package mvctest.controller;
```

```
import org.springframework.web.multipart.MultipartFile;
```

```
public class ReportCommand {
```

```
    private String studentNumber;
```

```
    private MultipartFile report;
```

```
    public String getStudentNumber() { return studentNumber; }
```

```
    public void setStudentNumber(String studentNumber) {  
        this.studentNumber = studentNumber;  
    }
```

```
    public MultipartFile getReport() { return report; }
```

```
    public void setReport(MultipartFile report) { this.report = report; }
```

```
}
```

■ 파일 업로드 처리

views/report/uoloadForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    <style type="text/css">
        body{ line-height: 240%;}
    </style>
</head>
<body>
<form action="submitReport1" method="post" enctype="multipart/form-data">
    학번 <input type="text" name="studentNumber"> <br>
    이름 <input type="text" name="studentName"> <br>
    리포트 파일 <input type="file" name="report"> <br>
    <input type="submit">
</form>
```

■ 파일 업로드 처리

views/report/uoloadForm.jsp

<h3>MultipartHttpServletRequest 사용</h3>

<form action="submitReport2" method="post" enctype="multipart/form-data">

학번: <input type="text" name="studentNumber" />

리포트파일: <input type="file" name="report" />

<input type="submit" />

</form>

<h3>커맨드 객체 사용</h3>

<form action="submitReport3" method="post" enctype="multipart/form-data">

학번: <input type="text" name="studentNumber" />

리포트파일: <input type="file" name="report" />

<input type="submit" />

</form>

</body>

■ 파일 업로드 처리

views/report/uploadOk.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>
학번 : ${studentNumber}<br>
이름 : ${studentName}<br>
파일이름 : ${fileName}<br>
파일사이즈 : ${fileSize}<br>

</h1>
</html>
```

■ openproject 사진 업로드 적용

```
@RequestMapping(method = RequestMethod.POST)
public String joinSubmit(Member member, HttpServletRequest request) throws
IllegalStateException, IOException {

    member.setRegdate(new Date());
    /*업로드 폴더 시스템 물리적 경로 찾기*/
    String uploadURI = "/uploadfile/memberphoto";
    String dir = request.getSession().getServletContext().getRealPath(uploadURI);
    System.out.println(dir);

    // 업로드 파일의 물리적 저장
    if (!member.getPhotoFile().isEmpty()) {
        member.getPhotoFile().transferTo(new File(dir, member.getMemberid()));
        member.setPhoto(member.getMemberid());
    }

    joinService.join(member);

    return "redirect:/main";
}
```

■ openproject 사진 업로드 적용

servlet-context.xml

```
<!-- Handles HTTP GET requests for /resources/** by efficiently serving  
up static resources in the ${webappRoot}/resources directory -->  
<resources mapping="/resources/**" location="/resources/" />  
  
<!-- 사진 저장 폴더를 리소스 폴더로 등록 -->  
<resources mapping="/uploadfile/**" location="/uploadfile/" />
```

■ openproject 사진 업로드 적용

views/mypage/mypage.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>
        ${loginInfo.member_id} / ${loginInfo.photo}
    </h1>

    

</body>
</html>
```

■ @PathVariable을 이용한 경로 변수 처리

- ID 가 10인 회원의 정보를 조회하기 위해한 URL을 구성할 때 다음과 같은 방법으로 요청 URL 경로에 포함시킬 수 있다.

<http://localhost/open/member/mypage/10>

- 경로의 특정 위치 값이 고정되지 않고 달라질 때 사용하는 것이 @PathVariable

```
@RequestMapping("/mypage/mypage/{id}")
public String mypageView(@PathVariable("id") Long memID, Model model) {
    Member member = dao.selectById(id);
    if ( member == null ) {
        throw new MemberNotFoundException();
    }
    return "mypage/mypage";
}
```

■ HandlerInterceptor를 통한 요청 가로채기

- 핸들러 인터셉터는 DispatcherServlet 이 컨트롤러를 호출하기 전과 후에 요청과 응답을 참조하거나 가공할 수 있는 일종의 필터
- 스프링이 기본적으로 제공하는 HandlerMapping은 HandlerInterceptor를 이용해서 컨트롤러가 요청을 처리하기 전과 처리한 후에 알맞은 기능을 수행 할 수 있도록 하고 있다.
- 조건에 따라 컨트롤러에 요청을 전달하지 않고 싶거나 컨트롤러가 요청을 처리한 후에 ModelAndView 객체를 조작하고 싶은 경우에 HandlerInterceptor를 사용하면 됨.
- EX) 로그인 여부 체크

■ HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor 인터페이스의 메서드.**
 - boolean **preHandle**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler)
 - void **postHandle**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler, **ModelAndView** modelAndView)
 - void **afterCompletion**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler, **Exception** ex)

■ HandlerInterceptor를 통한 요청 가로채기

– preHandle()

컨트롤러가 호출되기 전에 실행된다. handler 파라미터는 핸들러 매핑이 찾아 준 컨트롤러 빈 오브젝트다. 컨트롤러 실행 이전에 처리해야 할 작업이 있거나, 요청정보를 가공하거나 추가하는 경우에 사용할 수 있다. 또는 요청에 요청에 대한 로그를 남기기 위해 사용하기도 한다.

리턴 값이 true 이면 핸들러 실행 체인의 다음 단계로 진행되지만, false 라면 작업을 중단하고 리턴하므로 컨트롤러와 남은 인터셉터들은 실행되지 않는다.

– postHandle()

컨트롤러를 실행하고 난 후에 호출된다. 이 메소드에는 컨트롤러가 돌려준 ModelAndView 타입의 정보가 제공 되서 컨트롤러 작업 결과를 참조하거나 조작할 수 있다.

– afterCompletion()

이름 그대로 모든 뷰에서 최종 결과를 생성하는 일을 포함한 모든 작업이 모두 완료된 후에 실행된다. 요청처리 중에 사용한 리소스를 반환해주기에 적당한 메소드다.

- 핸들러 인터셉터는 하나 이상을 등록할 수 있다. preHandle() 은 인터셉터가 등록된 순서대로 실행된다. 반면에 postHandle() 과 afterCompletion() 은 preHandle() 이 실행된 순서와 반대로 실행된다.

■ HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor 인터페이스의 구현**

- 핸들러 인터셉터는 **HandlerInterceptor** 인터페이스를 구현해서 만든다. 이 인터페이스를 구현 할 경우 사용하지 않는 메서드도 구현 해주어야 한다.
- 이러한 불편함을 줄여주기 위해 **HandlerInterceptorAdaptor** 클래스를 제공.
- **HandlerInterceptor** 인터페이스를 구현해야 하는 클래스는 **HandlerInterceptorAdaptor** 클래스를 상속 받은 뒤 필요한 메서드만 오버라이딩 하여 사용.

■ HandlerInterceptor를 통한 요청 가로채기

- HandlerMapping에 HandlerInterceptor 설정 하기

- HandlerInterceptor를 구현 한 뒤에는 `<mvc:interceptors>` 의 interceptors 프로퍼티를 사용해서 **HandlerInterceptorAdapter** 를 등록해 주면 된다.

```
<!-- 인터셉터 이용한 로그인 체크 -->

<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/mypage/**" />
        <beans:bean
class="com.bitcamp.memberboard.interceptor.AuthCheckInterceptor" />
        </mvc:interceptor>
    </mvc:interceptors>
```

■ HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor의 실행 순서**

- HandlerMapping에는 한 개 이상의 **HandlerInterceptor**를 등록 할 수 있음.

1. 컨트롤러 실행 전 : 등록된 순서대로 preHandle() 실행
2. 컨트롤러 실행 후 : 등록된 순서 반대로 postHand() 실행
3. 처리 완료 후(뷰 생성 후) : 등록된 순서의 반대로 afterCompletion() 실행

```
<mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/mypage1/**" />
    <beans:bean
      class="com.bitcamp.open.interceptor.AuthCheckInterceptor1" />
  </mvc:interceptor>
  <mvc:interceptor>
    <mvc:mapping path="/mypage2/**" />
    <beans:bean
      class="com.bitcamp.open.interceptor.AuthCheckInterceptor2" />
  </mvc:interceptor>
  <mvc:interceptor>
    <mvc:mapping path="/mypage3/**" />
    <beans:bean
      class="com.bitcamp.open.interceptor.AuthCheckInterceptor3" />
  </mvc:interceptor>
</mvc:interceptors>
```

■ HandlerInterceptor를 통한 요청 가로채기

4. preHandle() 실행:
 - a. Interceptor1.preHandle()
 - b. Interceptor2.preHandle()
 - c. Interceptor3.preHandle()
5. 컨트롤러 handleRequest() 메서드 실행
6. postHand() 실행
 - a. Interceptor3.postHand()
 - b. Interceptor2.postHand()
 - c. Interceptor1.postHand()
7. 뷰 객체의 render() 메서드 실행에서 응답 결과 생성
8. afterCompletion() 실행
 - a. Interceptor3.afterCompletion()
 - b. Interceptor2.afterCompletion()
 - c. Interceptor1.afterCompletion()

```
com.bitcamp.open.interceptor.AuthCheckInterceptor
```

```
package com.bitcamp.open.interceptor;
```

```
public class AuthCheckInterceptor extends HandlerInterceptorAdapter {
```

```
    @Override
```

```
    public boolean preHandle(HttpServletRequest request,  
        HttpServletResponse response, Object handler) throws Exception {
```

```
        HttpSession session = request.getSession(false);
```

```
        if (session != null) {
```

```
            Object authInfo = session.getAttribute("loginInfo");
```

```
            if (authInfo != null) {
```

```
                return true;
```

```
            }
```

```
        }
```

```
        response.sendRedirect(request.getContextPath()+"member/login");
```

```
        return false;
```

```
    }
```

```
}
```

스프링 설정

...

```
<!-- 인터셉터 이용한 로그인 체크 -->
<!-- 여러 핸들러인터셉터를 생성 -->
<mvc:interceptors>
    <!-- 한개의 핸들러인터셉터를 생성 -->
    <mvc:interceptor>
        <!-- 핸들러인터셉터를 적용할 경로 설정 -->
        <mvc:mapping path="/mypage/**" />
        <!-- 경로중 일부 경로를 제외하고 싶을 때 -->
        <!-- <mvc:exclude-mapping path="/mypage/help/**"/> -->
        <beans:bean
            class="com.bitcamp.open.interceptor.AuthCheckInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>
```

...

■ 예외처리

- 컨트롤러 클래스의 **@RequestMapping**이 선언된 메소드는 모든 타입의 예외를 발생 시킬 수 있음.
- 서블릿 컨테이너가 생성한 에러 페이지가 아니라 예외 타입에 따라 스프링 MVC와 연동된 뷰를 이용해서 에러 페이지를 출력하고자 한다면, 스프링이 제공하는 **HandlerExceptionResolver** 인터페이스를 사용.
 - **AnnotationMethodHandlerExceptionResolver**
@ExceptionHandler 어노테이션이 적용된 메소드를 찾아 예외처리
 - **DefaultHandlerExceptionResolver**
스프링에서 내부적으로 발생하는 주요 예외를 처리해주는 표준 예외처리 로직을 담고 있다.
 - **SimpleMappingExceptionResolver**
예외 타입 별로 뷰 이름을 지정할 때 사용

■ 예외처리

- 예외처리 방법

- **HandlerExceptionResolver**

HandlerExceptionResolver 인터페이스의 resolveException() 메서드는 발생한 예외 객체를 파라미터로 전달받음

- **@ExceptionHandler**

@Controller 어노테이션이 적용된 클래스에 @ExceptionHandler 어노테이션이 적용된 메서드 구현

- **설정 파일을 이용한 선언적 예외 처리**

SimpleMappingExceptionHandler 클래스를 이용하기 위해 설정 파일에 설정

■ 예외처리

- **@ExceptionHandler** 어노테이션을 이용한 처리
 - AnnotationMethodHandlerExceptionResolver 클래스는 **@Controller** 어노테이션이 적용된 클래스에서 **@ExceptionHandler** 어노테이션이 적용된 메서드를 이용해서 예외 처리한다.
- @ModelAttribute** 에서 **@ExceptionHandler** 에서 지정한 예외가 발생하면 **@ExceptionHandler**가 적용된 메서드를 이용해서 뷰를 지정.

```
import org.springframework.web.bind.annotation.ExceptionHandler;

...

    @ExceptionHandler(NullPointerException.class)
    public String handleNullPointerException(NullPointerException ex) {
        return "error/nullException";
    }

...
```

■ 예외처리

views/error/nullException.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>예외 발생</title>
</head>
<body>
요청을 처리하는 과정에서 예외(null)가 발생하였습니다.
</body>
</html>
```

■ [과제] 회원 관리 프로그램

- 회원가입 시에 사진 업로드 처리
- 인터셉터를 통해 특정 경로의 요청에 로그인 여부 확인 처리
ex) 마이페이지, 회원관리 페이지