Spring MVC Form
File Upload & Download

524730-1
2021년 봄학기
5/26/2021
박경신

컨트롤러 구현

HomeController.java

```
@Controller  어노테이션
public class HomeController {
    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping(value = "/", method = RequestMethod.GET)  경로 매핑
    public String home(Locale locale, Model model) {
    logger.info("Welcome home! The client locale is {}.", locale);
    Date date = new Date();
    DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
    DateFormat.LONG, locale);
    String formattedDate = dateFormat.format(date);
    model.addAttribute("serverTime", formattedDate );
    // add more attribute
    String greetings = "Greetings, Spring MVC";
    model.addAttribute("message", greetings);
    return "home";    뷰 이름 리턴
    }
}
```

모델에 데이터 추가

serverTime과 message 속성이 모델에 추가됨

@RequestMapping 으로 요청 매핑하기

- 요청 URL은 다양
  - /springmvc/members
  - /springmvc/cart/emptycart
- 각각의 유의미한 URL에 대해서 이를 처리할 메소드를 구현하여 연결하는 작업 -> 요청 매핑
- 사용자가 localhost:8080/springmvc/members 연결 시 (GET메소드로) 메소드 실행하여 적절한 뷰로 연결

HomeController.java

```
@RequestMapping(value = "/members", method = RequestMethod.GET)  경로 매핑
public ModelAndView list(ModelAndView model) {
    list<Member> list = dao.list();
    model.addObject("memberList", list);   뷰에 넘겨줄 모델
    model.setViewName("home");   home.jsp 뷰 이름 리턴
    return model;
}
```

@RequestMapping 으로 요청 매핑하기

- 요청 URL경로 처리에 유의
  - 만약 우리 컨트롤러가 dispatcher에 의해 /main/*으로 매핑중이라면(web.xml)
  - 이 컨트롤러의 한 메소드가 @RequestMapping으로 /event/list를 처리한다면
- 실제로는 /springmvc/main/event/list로 접근시에 이 메소드가 실행

@RequestMapping 으로 요청 매핑하기

- @Controller와 함께 사용하기

```
@Controller
@RequestMapping(value = "/my")
public class HomeController {

@RequestMapping(value = "/home", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
    logger.info("Welcome home! The client locale is {}.", locale);
```

- 이 경우 /springmvc/컨트롤러 경로/my/home에 대해 home메소드 실행

@RequestMapping 으로 요청 매핑하기

- HTTP 메소드 선택 기능
  - GET, POST, PUT, PATCH, DELETE, TRACE, OPTIONS
- 이 경로에 접근하는 사용자의 의도가 무엇인지에 따라 다른 동작 할당 가능
- 웹 브라우저에서 보내지는 요청은 **GET/POST**만 가능
  - GET : **주소창에 경로**를 입력/링크를 클릭 등
  - POST : **<form method = "post">** 등의 태그를 통해 값을 전달

@GetMapping, @PostMapping

- 스프링 MVC는 별도 설정이 없으면 GET, POST 방식에 상관없이 @RequestMapping에 지정한 경로와 일치하는 요청을 처리
- 그 외에도 @GetMapping 또는 @PostMapping 사용 가능

```
@Controller
public class HomeController {

// /home/register 경로로 들어오는 요청 중 POST 방식만 처리
@PostMapping("/home/register")
public String register(Model model) {
    ....
}
// /home 경로로 들어오는 요청 중 GET 방식만 처리
@GetMapping("/home")
public String home() {
    return "redirect:/home/register";
}
```

리다이렉트 처리

- URL을 직접 입력하는 경우, redirect 사용
  - 웹어플리케이션을 기준으로 이동 경로를 생성
    - "redirect" 뒤의 문자열이 "/"로 시작하는 경우
    - 예를 들어, "redirect:/home/register"의 경우 웹 어플리케이션 경로와 합쳐져 "/springmvc/home/register"가 됨
  - 현재 경로를 기준으로 상대 경로를 이용
    - "/"로 시작하지 않을 경우
  - 절대 경로를 이용
    - 완전한 URL을 사용

@PathVariable로 경로 변수 만들기

- localhost:8080/springmvc/members/
- 각 멤버에 대해 매번 다른 메소드를 만들 수는 없으니…
  - @RequestMapping("/members/**{memberId}**")
    public String detail**(@PathVariable("memberId")** int memberId, Model model)
    - 메소드 안에서 memberId에 "Park"으로 사용 가능
    - 한 경로에 여러 **PathVariable** 사용 가능
      - /members/{memberId}/orders/{orderId}
    - 타입은 알아서 적절하게 변환됨(문자열->숫자)

Ant 패턴을 이용한 경로 표현

- Ant는 ?, *, ** 을 이용하여 경로 패턴을 명시
  - ? 1개 글자
  - * 0개 이상의 글자
  - ** 0개 이상의 디렉토리 경로
- m/category/files, m/category/sub/files, m/cat/sub/my/files 모두를 받은 경로를 만들려면?
  - "/m/**/files"
  - 실제 경로 값을 구하기 위해 request.getRequestUri()를 실행해야 함

@GetMapping("/member/?*.jsp") /member/로 시작하고 확장자가 .jsp로 끝나는 모든 경로

@PostMapping("/folders/**/files") /folders/로 시작하고 중간에 0개 이상 중간 경로가 존재하고 /files로 끝나는 모든 경로

@RequestMapping("/m/image?.htm") /m/image로 시작하고 1글자가 사이에 위치하고 .htm로 끝나는 모든 경로

처리 가능한 요청/응답 가능한 컨텐트 타입 제한

- 웹브라우저에서 input 태그를 이용해 폼을 전송할 때는 기본적으로 application/x-www-form-urlencoded 사용
- AJAX등의 등장에 따라 json/xml등을 전송하는 경우가 늘어남
- 요청이 json인 경우만 처리하는 매핑 :
  - @RequestMapping(value="...."... **consumes**="application/json")
- 응답이 json인 경우만 처리하는 매핑 :
  - @RequestMapping(value="...."... **produces**="application/json")

Model을 통한 컨트롤러에서 뷰로 데이터 전달

- 컨트롤러는 뷰가 응답 화면을 구성하는데 필요한 데이터를 생성해서 Model을 이용하여 전달
  - RequestMapping이 적용된 메소드의 파라미터로 Model을 추가
  - Model addAttribute(String attrName, Object attrValue);
  - Model addAllAttributes(Map<String, ?> attributes);
  - boolean containsAttribute(String attrName);

```
@RequestMapping(value = "/members/{id}", method = RequestMethod.GET)
public String detail(@PathVariable("id") int id, Model model) {
    Member member = dao.get(id);
    model.addAttribite("member" , member); // model에 데이터 추가
    return "/home/member"; // 뷰 이름을 리턴
}
```

ModelAndView를 통한 뷰선택과 모델 전달

- ModelAndView를 사용하여 뷰와 모델을 한번에 처리 가능
- ModelAndView addObject(String name, Object object);
- ModelAndView setViewName(String viewName);

```
@RequestMapping(value = "/members/add", method = RequestMethod.GET)
public ModelAndView detail(ModelAndView model) {
    Member member = new Member();
    model.addObject("member" , member); // model에 데이터 추가
    model.setViewName("addForm"); // 뷰 이름을 지정
    return model;
}
```

HTTP request 처리하기

- Get/Post 전송된 요청 파라미터 값을 사용하기 위한 방법
  - **HttpServletRequest**를 직접 이용
  - **@RequestParam** 어노테이션을 사용

```
@RequestMapping(value = "/detail", method = RequestMethod.GET)
public ModelAndView detail(HttpServletRequest request) {
    String name = request.getParameter("name");
    ....
}
```

```
@RequestMapping(value = "/detail", method = RequestMethod.GET)
public ModelAndView detail(@RequestParam("name") String name) {
   // 스프링에서 지원하는 변환기에서 지원되는 모든 타입을 변환 가능
    ....
}
```

```
@RequestMapping(value = "/detail", method = RequestMethod.GET)
public ModelAndView detail(@RequestParam(value="id" defaultValue="0") int id,
@RequestParam("name") String name) {
....
}
```

HTTP request 처리하기

- 웹 페이지에서 서버로 다양한 값이 전달됨
  - GET방식/POST 방식
  - 로그인 데이터, 게시판에 글을 쓴 데이터, ...

```
@RequestMapping("/detail")
public String detail(HttpServletRequest request, Model model) throws IOException {
    String id = request.getParameter("id");
    if (id == null)
        return REDIRECT_EVENT_LIST;
    Long eventId = null;
    try {
        eventId = Long.parseLong(id);
    } catch (NumberFormatException e) {
        return REDIRECT_EVENT_LIST;
    }
    Event event = getEvent(eventId);
    if (event == null)
        return REDIRECT_EVENT_LIST;
    model.addAttribute("event", event);
    return "event/detail";
}
```

/event/detail?id=1123

id==1123

<input name="id">1123</input>

HTTP Request 처리하기

- **@RequestParam** 어노테이션을 이용해 동일한 기능 구현 가능

```
@RequestMapping("/detail2")
public String detail2(@RequestParam("id") long eventId, Model model) {
    Event event = getEvent(eventId);
    if (event == null)
        return REDIRECT_EVENT_LIST;
    model.addAttribute("event", event);
    return "event/detail";
}
```

- RequestParam("id", required=false)를 이용해 id자리에 null을 넣을 수도 있음
  - defaultValue 속성을 이용해 null 대신 기본값도 사용 가능

Command 객체를 이용한 폼 전송 처리

- <form>
  <input type="text" name="email">...
  <input type="text name="name">....
- 여러 값을 한꺼번에 parameter로 받는 경우 @RequestParam을 여러 번 사용해야 함
- MemberRegRequest와 같은 **커맨드 객체**(자바빈즈 규약 준수)를 만들고 컨트롤러 메소드의 parameter에 전달 가능
  - 모델에도 자동으로 포함되므로 매우 편리
  - "${memberRegRequest.name}님 환영합니다" 와 같이 뷰 JSP 코드에서 커맨드 객체 사용 가능

```
public class MemberRegRequest{
    private String email, name, password;
    public String getEmail(){...}
    public String getName(){...}
... }
```

Command 객체를 이용한 폼 전송 처리

- **@ModelAttribute** 어노테이션으로 커맨드 객체 폼 전송
  - 커맨드 객체에 접근할 때 사용할 속성 이름을 @ModelAttribute 어노테이션으로 사용해서 변경
  - 스프링은 <form:form> 태그와 <form:input> 태그를 제공
  - <form:form> 태그를 사용하려면 커맨드 객체가 존재해야 함

```
@RequestMapping(value="/update", method=Request.POST)
public String edit(@ModelAttribute MemberRegRequest memberRegRequest) {
    .... memberRegRequest
}
```

memberRegRequest 객체를 모델에 넣어야 함

```
<!– editForm.jsp -->
<form:form action="update" method="post" modelAttribute="memberRegRequest">
<label>Email:<br><form:input path="email" value="${memberRegRequest.email}"
/></label>
<label>Name:<br><form:input path="name" value="${memberRegRequest.name}"
/></label>
<label>Password:<br> <form:input path="password"
value="${memberRegRequest.password}" /> </label>
</form:form>
```

Command 객체를 이용한 폼 전송 처리

- editForm.jsp 뷰를 호출하는 컨트롤러 코드에 memberRegRequest 커맨드 객체를 넣어줘야 함

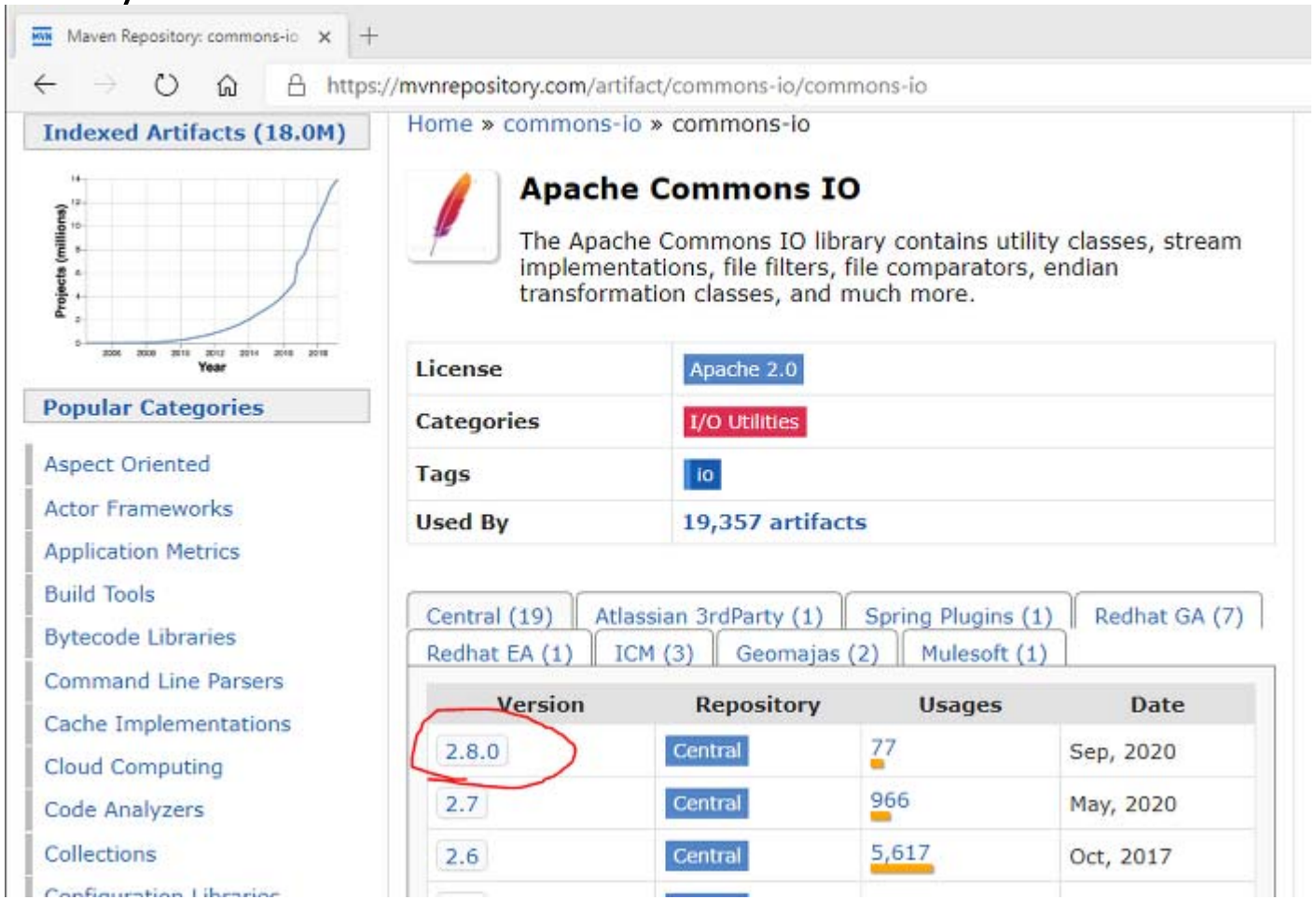
```
@RequestMapping(value = "/edit", method = RequestMethod.GET)
public ModelAndView editPet(HttpServletRequest request) {
    String name = request.getParameter("name");
    ...
    ModelAndView model = new ModelAndView("editForm");
    model.addObject("memberRegRequest", memberRegRequest);
    ...
    return model;
}
```

주요 폼 태그

- 스프링 MVC는 <form:form>, <form:input> 등 폼과 커맨드 객체를 연동하기 위한 JSP 태그 라이브러리를 제공
- <form:form> 태그
  - 커맨드 객체의 이름이 기본값인 "command"가 아니라면 modelAttribute 속성을 사용해 설정
  - 커맨드 객체를 이용해 이전에 입력한 값을 출력 가능
- <form:input> 태그
  - <form:input> 태그는 text 입력에 사용. path 속성을 사용해 연결할 커맨드 객체의 프로퍼티를 지정
- <form:select>, <form:options>, <form:option> 태그
  - <form:select> 태그는 선택 옵션을 제공할 때 사용. items 속성에 옵션목록(Array 또는 List)을 Model을 통해 전달하면 간단하게 생성

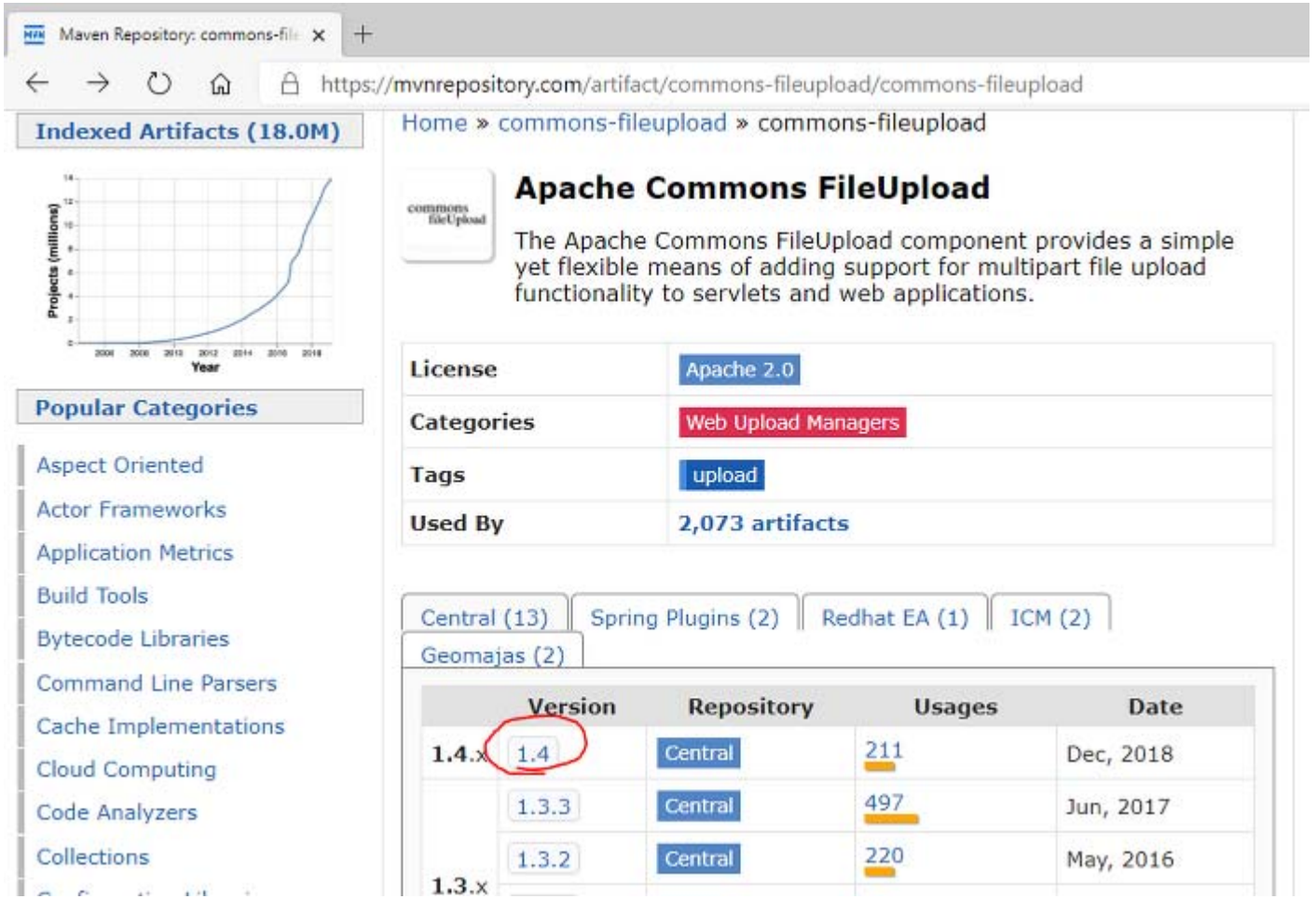
File Upload

- https://mvnrepository.com/artifact/commons-io/commons-io



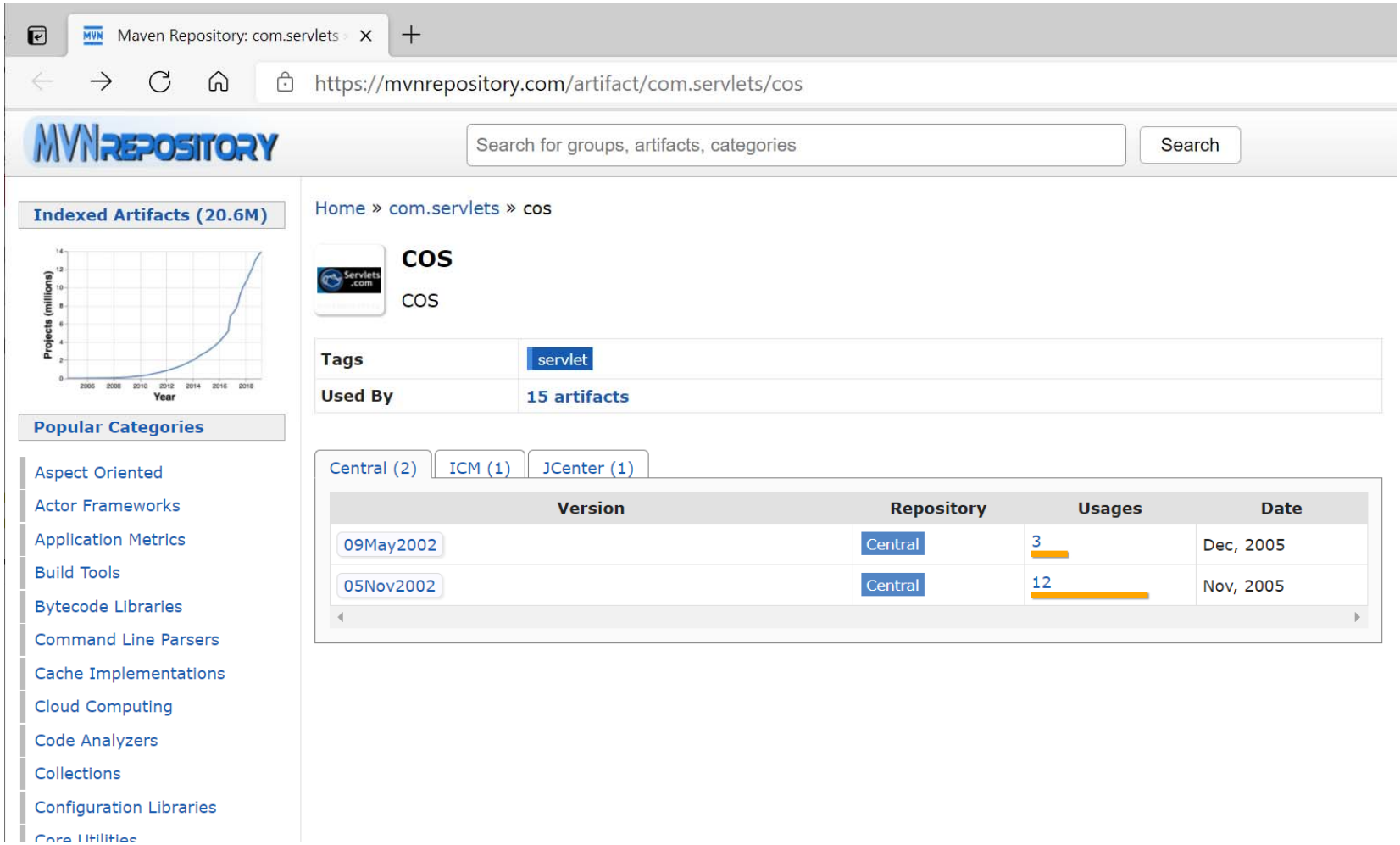
File Upload

- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload



File Upload

- https://mvnrepository.com/artifact/com.servlets/cos



pom.xml 수정

pom.xml

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.8.0</version>
</dependency>

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

root-context.xml 수정

src/main/webapp/WEB-INF/spring/root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Root Context: defines shared resources visible to all other web components -->

<!-- MultipartResolver -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <!-- max file size in bytes -->
    <property name="maxUploadSize" value="2000000" />
    <!-- other properties... -->
</bean>

</beans>
```

File Upload

- 파일 업로드 (File Upload)
  - 웹 브라우저에서 서버로 파일을 전송하여 서버에 저장하는 것
  - 서버로 업로드할 수 있는 파일
    - 텍스트 파일, 바이너리 파일, 이미지 파일, 문서 등
  - 웹 브라우저에서 서버로 파일을 전송하기 위해 JSP 페이지에 폼 태그 사용
  - 전송된 파일을 서버에 저장하기 위해 오픈 라이브러리를 이용

파일 전송을 위한 Form 설정

- 파일 업로드 (File Upload)를 위한 JSP 페이지
  - 웹 브라우저에서 서버로 파일을 전송하기 위해 JSP 페이지에 폼 태그를 작성할 때 몇 가지 중요한 규칙
    - form 태그의 method 속성은 반드시 **POST** 방식으로 설정
    - form 태그의 enctype 속성은 반드시 **multipart/form-data**로 설정
    - form 태그의 action 속성은 파일 업로드를 처리할 JSP 파일로 설정
    - 파일 업로드를 위해 input 태그의 **type** 속성을 **file**로 설정
      - 만약 여러 파일을 업로드하려면 2개 이상의 input 태그를 사용하고 name 속성에 서로 다른 값을 설정

uploadForm.jsp

```
<form action="fileupload" method="POST" enctype="multipart/form-data">
    ...
  <input type="file" name="uploadfile" />
  <input type="submit" value= "upload" >
</form>
```

파일 전송을 위한 Form 설정

- 여러 개 파일 업로드 (File Upload)를 위한 JSP 페이지

uploadForm.jsp

```
<form action="multifileupload" method="POST" enctype="multipart/form-data">
    ...
  <input type="file" name="uploadfiles" placeholder="Select File" multiple />
  <input type="submit" value= "upload" >
</form>
```

MultipartRequest를 이용한 파일 업로드

- MultipartRequest
  - 웹 페이지에서 서버로 업로드되는 파일 자체만 다루는 클래스.
  - 웹 브라우저가 전송한 multipart/form-data 유형과 POST 방식의 요청 파라미터 등을 분석한 후 일반 데이터와 파일 데이터를 구분하여 파일 데이터에 접근
  - 한글 인코딩 값을 얻기 쉽고, 서버의 파일 저장 폴더에 동일한 파일명이 있으면 파일명을 자동으로 변경
  - 오픈 라이브러리 cos.jar를 배포 사이트에서 직접 다운로드해서 사용
    - 배포 사이트: http://servlets.com/cos/
    - JSP 페이지에 page 디렉티브 태그의 import 속성을 사용하여 패키지 com.oreilly.servlet.*을 설정

MultipartRequest를 이용한 파일 업로드

HomeController.java

```
@RequestMapping(value = "/fileupload",method = RequestMethod.POST)
public String fileupload(@RequestParam("uploadfile") MultipartFile uploadfile, Model
model) throws IOException{
    String result = saveFile(uploadfile);
    if (result != null) { // file save success
        model.addAttribute("result", result + " is uploaded successfully!");
    } else { // file save failed
        model.addAttribute("result", "file upload is failed!");
    }
    return "result"; // result.jsp
}
private final String UPLOAD_PATH = "c:" + File.separator + "temp" + File.separator;
private String saveFile(MultipartFile file) throws IOException{
    String saveName = UUID.randomUUID() + "_" + file.getOriginalFilename();
    File saveFile = new File(UPLOAD_PATH, saveName); // folder, filename
    try {
        file.transferTo(saveFile); // file transfer
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
    return saveName;
}
```

MultipartRequest를 이용한 파일 업로드

```
@RequestMapping(value = "/multifileupload",method = RequestMethod.POST)
public String multiupload(@RequestParam("uploadfiles") MultipartFile[] files, Model
model) throws IOException {
    String result = "";
    for (MultipartFile file : files) {
        result += saveFile(file) + "₩n";
        model.addAttribute("result", result);
    }
    return "result"; // result.jsp
}
```

HomeController.java

다운로드의 구현

- 다운로드 구현 시 고려 사항
  - 응답 컨텐츠 타입은 application/octet-stream
  - Content-Disposition 헤더로 파일명 지정
    - 파일명 설정시 ISO-8859-1로 인코딩 변환해서 설정
- 실제 파일 전송
  - response.getOutputStream()으로 구한 OutputStream에 파일 데이터 출력

다운로드의 구현

HomeController.java

```
@RequestMapping(value = "/filedownload",method = RequestMethod.GET)
public void filedownload(HttpServletRequest request,
        HttpServletResponse response) throws IOException{
    ServletContext context = request.getSession().getServletContext();
    String appPath = context.getRealPath("");
    // construct the complete absolute path of the file
    String fullPath = appPath + filePath;
    File downloadFile = new File(fullPath);
    FileInputStream inputStream = new FileInputStream(downloadFile);
    // get MIME type of the file
    String mimeType = context.getMimeType(fullPath);
    if (mimeType == null) {
        // set to binary type if MIME mapping not found
        mimeType = "application/octet-stream";
    }
    // set content attributes for the response
    response.setContentType(mimeType);
    response.setContentLength((int) downloadFile.length());
    // set headers for the response
    String headerKey = "Content-Disposition";
    String headerValue = String.format("attachment; filename=|"%s|"",
                                       downloadFile.getName());
    response.setHeader(headerKey, headerValue);
```

다운로드의 구현

HomeController.java

```
    // get output stream of the response
    OutputStream outStream = response.getOutputStream();

    byte[] buffer = new byte[BUFFER_SIZE];
    int bytesRead = -1;

    // write bytes read from the input stream into the output stream
    while ((bytesRead = inputStream.read(buffer)) != -1) {
        outStream.write(buffer, 0, bytesRead);
    }

    inputStream.close();
    outStream.close();
}

private static final int BUFFER_SIZE = 4096;
private String filePath = "/resources/SpringProject.zip";
```