

Anyframe File Upload Plugin



Version 1.0.2

저작권 © 2007-2011 삼성SDS

본 문서의 저작권은 삼성SDS에 있으며 Anyframe 오픈소스 커뮤니티 활동의 목적하에서 자유로운 이용이 가능합니다. 본 문서를 복제, 배포할 경우에는 저작권자를 명시하여 주시기 바라며 본 문서를 변경하실 경우에는 원문과 변경된 내용을 표시하여 주시기 바랍니다. 원문과 변경된 문서에 대한 상업적 용도의 활용은 허용되지 않습니다. 본 문서에 오류가 있다고 판단될 경우 이슈로 등록해 주시면 적절한 조치를 취하도록 하겠습니다.

I. Introduction	1
II. File Upload	2
1. FTP 서버를 이용한 파일 업로드	4
1.1. 파일 업로드	4
1.2. 파일 다운로드	6
1.3. 파일 삭제	9

I.Introduction

Spring MVC에서는 MultipartResolver라는 모듈을 사용하여 웹UI에서 파일을 업로드할 수 있도록 지원하고 있다. 또한 Apache Commons FileUpload [<http://jakarta.apache.org/commons/fileupload>] 오픈 소스를 위한 MultipartResolver 구현체를 제공한다. Anyframe fileupload plugin은 Spring MVC 기반의 웹 어플리케이션에서 이러한 클래스들을 활용하여 파일업로드 기능을 구현하는 방법에 대해서 가이드하고 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 fileupload plugin을 설치한다.

```
mvn anyframe:install -Dname=fileupload
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Dependent Plugins

Plugin Name	Version Range
core [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.0.3/reference/htmlsingle/core.html]	2.0.0 > *

II. File Upload

Spring MVC는 파일 업로드 기능을 지원하기 위하여 Commons 파일 업로드 [http://commons.apache.org/fileupload/] 라이브러리를 지원한다. commons 라이브러리를 사용하기 위해서는 commons-fileupload-x.x.jar 파일과 commons-io-x.x.jar파일이 필요하다. 이는 Anyframe 배포 라이브러리에 포함되어 있다. 파일 업로드 기능을 구현하기 위해서는 먼저 빈 설정 파일에 다음과 같이 MultipartResolver를 정의해야한다.

```
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <property name="maxUploadSize">
    <value>10000000</value>
  </property>
</bean>
```

또한 해당 컨트롤러의 property로 파일의 업로드 위치를 지정해주고 컨트롤러에서 setter injection을 통해 지정된 파일 업로드 위치를 불러올 수 있다. 사용예는 다음과 같다.

```
<bean id="movieController"
      class="org.anyframe.sample.fileupload.movie.MovieController">
  <property name="destinationDir" value="C:/Temp/fileupload/temp" />
  <property name="movieService" ref="movieService" />
</bean>
```

파일 업로드를 위해 JSP파일의 입력 폼 타입을 file로 지정하고 form의 enctype을 multipart/form-data로 지정한다.

```
<body>
<form name="fileForm" action="file.do" method="post" enctype="multipart/form-data">
  파일 : <input type="file" style="width:400" name="file"><br/>
  <input type="submit" value="upload" />
</form>
</body>
```

Spring MVC에서는 파일 업로드를 위해 MultipartFile이라는 객체 타입을 제공한다.

```
private MultipartFile file;
private Long size;
private String name;
private String filePath;
```

다음은 파일 업로드를 위해 Controller를 구현한 모습이다.

```
public class MovieController extends AbstractCommandController {

  private File destinationDir;

  /**
   * 파일업로드를 위한 빈 설정의 property로 지정받은
   * destinationDir setter injection
   */
  public void setDestinationDir(File destinationDir) {
    this.destinationDir = destinationDir;
  }

  ...중략...

  protected ModelAndView handle(HttpServletRequest request,
    HttpServletResponse response, Object command, BindException exception)
    throws Exception {
```

```

//전달 받은 Request값을 MultipartHttpServletRequest로 바인딩 시킨다.
MultipartHttpServletRequest multipartRequest
    = (MultipartHttpServletRequest) request;

//request의 "file"을 찾아 file 객체에 세팅한다.
MultipartFile file = multipartRequest.getFile("file");
String fileName = file.getOriginalFilename();
File destination = File.createTempFile("file", fileName, destinationDir);

//파일 카피
FileCopyUtils.copy(file.getInputStream(), new FileOutputStream(destination));

//새로운 파일 속성 세팅
HelloVO vo = (HelloVO) command;
vo.setFilePath(destination.getAbsolutePath());
vo.setName(file.getOriginalFilename());
vo.setSize(file.getSize());
vo.setFile(file);
helloWorldService.getMessage1(vo);
return new ModelAndView("result", "message", vo);
}
}

```

위와 같이 간단한 파일 업로드를 실행시켜 볼 수 있다. 위의 예제는 화면에서 업로드된 파일을 MultipartFile타입으로 받았기 때문에 별다른 바인딩 작업이 필요하지않았다. 하지만 화면에서 업로드된 파일을 String 타입으로 바인딩하려면 StringMultipartEditor, byte 타입의 배열로 바인딩 하려면 ByteArrayMultipartEditor를 사용하여 Controller에 다음과 같이 initBinder 메소드를 오버라이드하여 구현해 줄 수 있다.

- StringMultipartEditor

```

protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
    throws ServletException {
    binder.registerCustomEditor(String.class, new StringMultipartFileEditor());
}

```

- ByteArrayMultipartEditor

```

protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
    throws ServletException {
    binder.registerCustomEditor(byte[].class, new ByteArrayMultipartFileEditor());
}

```

1. FTP 서버를 이용한 파일 업로드

이제까지 Apache Commons Fileupload와 연계하여 Spring MVC의 파일 업로드 기능을 구현하는 기본적인 방법에 대해 알아보았다. 이번 장에서는 파일을 FTP 서버에 업로드하는 방법에 대해서 알아 보도록 한다. 이 때, Apache Commons VFS(<http://commons.apache.org/vfs/>)를 이용해서 FTP 서버에 접근하도록 한다. Apache Commons VFS를 이용하여 FTP 또는 SFTP 프로토콜을 사용한 서버 접근 로직을 구현할 때에는 아래와 같은 참조 라이브러리가 필요하다.

Dependency	Required For
Commons Net(http://commons.apache.org/net/) Version 2.0 or later.	FTP
JSch Version 0.1.42 or later.(http://www.jcraft.com/jsch/)	SFTP

File Upload Plugin 설치 시 Commons VFS에서 제공하는 API를 사용하여 SFTP 서버에 파일을 업로드, 다운로드, 삭제하는 로직을 살펴 볼 수 있다. 이 때, 해당 Plugin에서 제공되는 SftpService 인터페이스를 통해 FTP 서버에 접근할 수 있으며 이를 구현한 SftpServiceImpl 클래스에서 실제 Commons VFS API를 호출하는 로직을 담고 있다. 이러한 SftpService를 재사용하거나 확장 및 Customizing하여 실제 프로젝트에 적용할 수 있다. 이에 대한 상세한 내용은 아래와 같다.

1.1. 파일 업로드

Anyframe File Upload Plugin에서 제공하고 있는 업로드 로직에 대해 알아보도록 한다. 먼저, JSP는 file 타입의 다중 파일 업로드를 가능하도록 구현되어 있다. 실제 코드는 아래와 같다.

```
<c:forEach var="attachedFile" items = "${attachedFiles}">
  <a href="<c:url value='/download.do'/">${attachedFile.name}</a>  (${attachedFile.fileSize}
  Byte)<br/>
</c:forEach>
```

```
var gFiles = 0;
function addFile() {
  var tr = document.createElement('tr');
  tr.setAttribute('id', 'file-' + gFiles);
  var td = document.createElement('td');
  //var removeFileId = "file"+gFiles
  td.innerHTML = '<input type="file" name="file"><span onclick="removeFile(\`file-' + gFiles
  + '\`)" style="cursor:pointer;">Delete</span>'
  tr.appendChild(td);
  document.getElementById('files-root').appendChild(tr);
  gFiles++;
}
function removeFile(aId) {
  var obj = document.getElementById(aId);
  obj.parentNode.removeChild(obj);
}
```

위와 같이 구현하여 다중 파일 업로드를 가능하게 화면을 구성하였다.

다음은 Controller 구현 부분이다. Spring MVC에서는 파일업로드를 위해 MultipartFile로 File 객체를 처리할 수 있다. Controller 구현부에서는 별도의 로직없이 MultipartFile 객체만 Service로 넘겨주고 있다.

```
@RequestMapping(params = "method=create")
public String create( @RequestParam("file") MultipartFile[] files ,
  Movie movie, BindingResult results, SessionStatus status) throws Exception {
  this.moviesService.create(movie, files);
}
```

```

status.setComplete();

return "redirect:/fileuploadMovieFinder.do?method=list";
}

```

Service에서는 유일한 File reference ID와 File ID를 생성하여 Movie, AttachedFile 객체에 대한 DB insert 실행 한 후 SftpService의 upload() 메소드를 호출하여 실제 FTP 서버에 파일을 업로드하는 일을 하게 된다.

```

public void create(Movie movie, MultipartFile[] files) throws Exception {

    if (files.length > 0) {
        String fileRefId = "";
        SimpleDateFormat formatter = new SimpleDateFormat(
            "yyyyMMddHHmmssSSS", new Locale("ko", "KR"));
        String formattedValue = formatter.format(new Date());
        fileRefId = fileRefIdPrefix + formattedValue;

        movie.setFileRefId(fileRefId);

        movieDao.create(movie);

        for(int i = 0; i < files.length; i++){

            MultipartFile file = files[i];
            if(file.getSize() > 0){
                String id = "";
                SimpleDateFormat formatter_file = new SimpleDateFormat(
                    "yyyyMMddHHmmssSSS", new Locale("ko", "KR"));
                String formattedValue_file = formatter_file.format(new Date());

                id = fileIdPrefix + formattedValue_file;

                AttachedFile attachedFile = new AttachedFile();

                attachedFile.setRefId(fileRefId);
                attachedFile.setId(id);
                attachedFile.setName(file.getOriginalFilename());
                attachedFile.setFileSize(file.getSize());

                uploadInfoDao.create(attachedFile);

                //create local file
                File localFile = createLocalFile(id);
                file.transferTo(localFile);

                //SFTP 서버에 업로드
                sftpService.upload(localFile, id);
            }
        }
    }
}

```

SftpService의 업로드 로직은 아래와 같이 구현되어 있다.

```

public void upload(File localFile, String fileId)
    throws Exception {
    FileSystemOptions fsOptions = null;
    DefaultFileSystemManager fsManager = null;
    SftpFileProvider sftp = null;
}

```

```

try {
    fsOptions = new FileSystemOptions();
    SftpFileSystemConfigBuilder.getInstance()
        .setStrictHostKeyChecking(fsOptions, "no");
    fsManager = new DefaultFileSystemManager();
    sftp = new SftpFileProvider();
    DefaultLocalFileProvider fspath = null;

    fspath = new DefaultLocalFileProvider();

    //initialize
    fspath.init();
    sftp.init();

    fsManager.addProvider("sftp", sftp);
    fsManager.addProvider("file", fspath);
    fsManager.init();

    SftpFileObject fo = null;

    try {
        fo = (SftpFileObject) fsManager.resolveFile(ftpPath + fileId, fsOptions);
        FileObject localObject = fsManager.toFileObject(localFile);
        fo.copyFrom(localFileObject, Selectors.SELECT_SELF);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        logger.debug(fo.getName() + " File이 SFTP 서버에 생성되었습니다.");
        if (fo != null)
            fo.close();
        localFile.delete();
        logger.debug("local File 삭제가 완료되었습니다.");
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (sftp != null)
        sftp.close();
    if (fsManager != null)
        fsManager.close();
}
}

```

위의 코드에서는 FileObject를 생성하여 Controller에서 넘어온 File 타입의 object를 SFTP 프로토콜을 사용하여 서버에 복사하고 있다. 이 때, SFTP 서버의 경로 및 ID/password는 설정 파일에서 따로 관리하도록 한다. 다음은 이를 정의한 context.propertis 파일의 일부이다.

```
ftpPath = sftp://username:password@server.ip/folder/
```

SFTP 서버에 대한 경로를 프로젝트에 맞게 고쳐서 설정해주도록 한다.

1.2.파일 다운로드

파일 다운로드도 업로드와 마찬가지로 SftpService를 사용한다. 이 때, 파일의 ID를 가지고 FileObject를 얻어오는 방법과 특정 위치에 해당 파일을 다운로드하는 방법을 사용할 수 있다.

```

// 특정 위치에 FTP 파일 저장
public void download(File localFile, String fileId)
    throws Exception {

```

```
FileSystemOptions fsOptions = null;
DefaultFileSystemManager fsManager = null;
SftpFileProvider sftp = null;

try {
    fsOptions = new FileSystemOptions();
    SftpFileSystemConfigBuilder.getInstance()
        .setStrictHostKeyChecking(fsOptions, "no");
    fsManager = new DefaultFileSystemManager();
    sftp = new SftpFileProvider();
    DefaultLocalFileProvider fspath = null;

    fspath = new DefaultLocalFileProvider();

    fspath.init();
    sftp.init();

    fsManager.addProvider("sftp", sftp);
    fsManager.addProvider("file", fspath);
    fsManager.init();

    SftpFileObject fo = null;

    try {
        fo = (SftpFileObject) fsManager.resolveFile(ftpPath
            + fileId, fsOptions);

        FileObject localFileObject = fsManager
            .toFileObject(localFile);
        localFileObject.copyFrom(fo, Selectors.SELECT_SELF);
        logger.debug(localFileObject.getName() + " 파일이 저장되었습니다.");

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (fo != null)
            fo.close();
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (sftp != null)
        sftp.close();
    if (fsManager != null)
        fsManager.close();
}
}

// FTP 서버에 있는 해당 FileObject를 얻어옴
public SftpFileObject getSftpFileObject(String fileId)
throws Exception {
    FileSystemOptions fsOptions = null;
    DefaultFileSystemManager fsManager = null;
    SftpFileProvider sftp = null;

    SftpFileObject fo = null;

    try {
        fsOptions = new FileSystemOptions();
        SftpFileSystemConfigBuilder.getInstance()
            .setStrictHostKeyChecking(fsOptions, "no");
        fsManager = new DefaultFileSystemManager();
```

```

sftp = new SftpFileProvider();
DefaultLocalFileProvider fspath = null;

fspath = new DefaultLocalFileProvider();

fspath.init();
sftp.init();

fsManager.addProvider("sftp", sftp);
fsManager.addProvider("file", fspath);
fsManager.init();

try {
    fo = (SftpFileObject) fsManager.resolveFile(ftpPath
        + fileId, fsOptions);
} catch (Exception e) {
    e.printStackTrace();
} finally {
    logger.debug(fo.getName() + " 파일을 얻어오는 데 성공 하였습니다.");
    if (fo != null)
        fo.close();
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (sftp != null)
        sftp.close();
    if (fsManager != null)
        fsManager.close();
}
return fo;
}

```

File Upload Plugin 샘플에서는 FileObject를 사용하여 직접 InputStream을 받아 파일을 출력해주는 로직으로 구현되어 있다. 다음은 이를 구현한 DownlodController의 일부이다.

```

//SFTP 다운로드 로직 추가(FileObject)
SftpFileObject fo = sftpService.getSftpFileObject(id);

BufferedInputStream fin = null;
BufferedOutputStream outs = null;

try{
    if (fo.getType() == FileType.FILE) {
        //FileObject의 InputStream 사용
        fin = new BufferedInputStream( fo.getInputStream());
        outs = new BufferedOutputStream(response.getOutputStream());
        int read = 0;
        while ((read = fin.read()) != -1){
            outs.write(read);
        }
    }
} catch (Exception e){
    e.printStackTrace();
} finally{
    if (outs != null){
        outs.flush();
        outs.close();
    }
    if (fin != null){

```

```
    fin.close();
  }
}
```

1.3.파일 삭제

SFTP 서버에 있는 파일을 삭제할 경우에도 파일의 ID를 가지고 SftpService를 사용하여 서버에 있는 파일을 삭제 할 수 있다. 기존 Controller 코드와 Service 코드에 추가적인 코딩은 불필요 하며 SftpService의 remove() 메소드를 콜해주어 해당 작업을 수행할 수 있다.

```
public void remove(String fileId) throws Exception {

    FileSystemOptions fsOptions = null;
    DefaultFileSystemManager fsManager = null;
    SftpFileProvider sftp = null;

    try {
        fsOptions = new FileSystemOptions();
        SftpFileSystemConfigBuilder.getInstance()
            .setStrictHostKeyChecking(fsOptions, "no");
        fsManager = new DefaultFileSystemManager();
        sftp = new SftpFileProvider();
        DefaultLocalFileProvider fspath = null;

        fspath = new DefaultLocalFileProvider();

        fspath.init();
        sftp.init();

        fsManager.addProvider("sftp", sftp);
        fsManager.addProvider("file", fspath);
        fsManager.init();

        SftpFileObject fo = null;

        try {
            fo = (SftpFileObject) fsManager.resolveFile(ftpPath + fileId, fsOptions);
            fo.delete();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            logger.debug("파일이 정상적으로 삭제되었습니다.");
            if (fo != null)
                fo.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (sftp != null)
            sftp.close();
        if (fsManager != null)
            fsManager.close();
    }
}
```