Anyframe File Upload Plugin

Version 1.2.0

I. Introduction 1
II. File Upload 2
1. 파일 업로드 개요 3
1.1. 파일업로드 기본설정 3
1.1.1. Spring anyframe:annotation-driven 설정 3
1.1.2. fileupload properties 설정 3
1.2. 파일업로드 화면 UI Flow 4
1.3. 파일업로드 기능 6
1.4. Javascript Libarary 6
1.5. Java Libarary 7
2. 파일 업로드 설정 및 구현 9
2.1. 화면개발을 위한 javascript 라이브러리 url 설정 9
2.2. 화면 UI 생성함수 호출 및 설정 9
2.2.1. 함수로드 및 fileupload 생성 9
2.2.2. fileupload 생성할 html 태그 설정 10
2.3. File Size 구현 10
2.4. File Name 구현 12
2.5. File Upload 구현 13
2.6. File Progress 구현 14
3. 기타 16
3.1. 브라우저별 Size 제약사항 16
3.2. 참고 사이트 16

I.Introduction

fileupload plugin은 jQuery 기반의 라이브러리를 사용하여 웹에서 파일을 업로드할 수있는 기능의 UI를 제공한다. 또한 파일이 업로드될때 대용량 파일 처리를 위하여 스트리밍 방식으로 직접 Disk에 저장되도록 Apache Commons FileUpload [http://commons.apache.org/fileupload/] 의 StreamingAPI를 사용하였고, 동적으로 form 전송 시 브라우저 호환성 처리를 위하여 jquery form plugin의 오픈소스가 라이브러리에 구성되어 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.

II.File Upload

RIA (Flash, Silverlight 등) 및 ActiveX 와 같은 비표준 솔루션을 사용하지 않고, 순수 HTML/Javascript 기반의 파일 업로드 UI 및 기능을 제공하는 것을 목표로한다. 세부적인 기능으로는, 파일추가, 파일삭제, 멀티업로드, 파일 Size 표시, 파일 업로드, 업로드 시 Progress 처리 등의 기능을 제공하며, IE7 이상, Chrome, Safari, Firefox 의 브라우저에서 동작되도록 호환성을 제공한다.

1.파일 업로드 개요

제공하는 파일업로드 plugin은 크게 서버측의 로직을 구현한 java 라이브러리와 화면측의 UI와 관련로직을 구현한 javascript 라이브러리로 구성되어 있다. 본 chapter 에서는 파일업로드의 전반적인 설명을 위해서 컴포넌트를 사용하기 위한 기본설정, 화면 UI 흐름과 사용법, 서버측의 기능, 구성 라이브러리에 대한 설명을 하고자 한다.

1.1.파일업로드 기본설정

위의 introduction에서와 같이 플러그인을 설치한 후, 추가적인 기본사항을 설정하고 확인하기 위해서 다음의 과정이 필요하다.

1.1.1.Spring anyframe:annotation-driven 설정

: 내부적으로 form 전송을 동적으로 비동기 호출한다. 이 때, 연속적으로 호출되어야 하는 form 전송 및 ajax 호출이 수행되지 않는 문제가 있다. 따라서, core-servlet.xml 파일에서 다음과 같은 설정으로 수정해주어야 한다. (파일 경로 : /anyframe-fileupload-pi/src/main/resources/spring/core-servlet.xml)

```
<anyframe:annotation-driven synchronizeOnSession="false" />
```

이와같이 anyframe annotation 설정부분을 synchronizeOnSession 값을 false로 바꾸어주면 form 전송 및 ajax 호출이 정상적으로 수행될 것이다.

1.1.2.fileupload properties 설정

: 본 파일업로드는 disk에 파일을 저장하는 방식을 구현하였으며, 관련하여 구동전에 몇가지 환경정보를 설정하여야 한다. 반드시 입력되어야 할 정보로는 임시 저장위치 및 실제 저장위치가 있고, 그 외 업로드 될 화일의 한계 크기를 fileupload.properties 파일에 정의할 수 있다.

```
#Common
common.temp.path=C:/Temp
common.file.maxsize=-1

#Local Disk
disk.path=C:/Temp
```

각 설정 내용은 다음과 같다.

- common.temp.path : 파일이 streaming 되어 임시로 저장될 위치를 설정한다.
- common.file.maxsize : 업로드할 파일의 최대사이즈를 설정한다. (Byte 단위) 단, 브라우저별 제약사항이 존재한다. 참조 3.1
- disk.path : 파일의 실제 저장될 로컬 경로를 설정한다.

CommonsMultipartResolver 관련 주의사항

: 본 파일업로드를 다른 플러그인과 함께 사용할 경우, multipartResolver와 관련된 문제가 있다. 따라서, 다음의 경로에 선언되어 있는 multipartResolver를 지워줘야만 정상적으로 파일첨부 기능을 사용할 수 있다.

: src/main/resources/spring/jquery-servlet.xml or src/main/resources/spring/xp-query-servlet.xml

삭제 혹은 주석처리해야할 부분의 코드는 아래와 같다.

```
<!-- 삭제 혹은 주석처리 부분 -->
<!--
<bean id="multipartResolver"
   class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
   <property name="maxUploadSize">
     <value>10000000</value>
   </property>
</bean>
-->
```

1.2.파일업로드 화면 UI Flow

: 기본 UI, 파일 리스트 UI, 파일업로드 UI 의 각 업로드 과정에 따라 어떤 화면 요소들과 기능이 존재하는지 알려준다.

1. 기본 UI
   : 제공하는 기본 UI 요소와 기능, 실제 화면예시는 다음과 같다.

   - Add : 업로드하고자 하는 파일을 add 버튼으로 추가한다. 여러개 선택하여 추가가 가능하며, 추가된 파일은 리스트에 파일명으로 추가된다.
   - Upload : 한개 혹은 여러개 추가된 파일들에 대해 업로드를 시작한다.
   - Reset : 추가된 리스트들을 모두 취소하고자 할 때, Reset 버튼을 실행한다.

   위와 같은 UI 요소가 생성되기 위해서는 jquery, jquery ui 라이브러리가 필요하며, 이를 토대로 jsp 파일 내에 스크립트로 구현되어 있다. (관련된 jsp파일 내 url 설정 및 함수호출 부분은 chapter #2를 참조한다.)

2. 파일 리스트 UI
   : add 버튼으로 파일을 한개 또는 여러개 추가할 수 있으며, 파일이 추가될 때 나오는 리스트 정보들과 실제 화면예시는 다음과 같다.

   - Icon : 파일의 확장자명에 따라 총 7개의 아이콘과 default 아이콘이 존재하며, 사용맥락에 따라 추가적인 이미지도 사용가능하다.
   - File Name : 선택된 파일의 정보를 (filename).(extension) 과 같은 형식으로 보여준다 .

- Progressbar : 업로드 진행과 동시에 진행 정도를 보여주며, 실제 업로드 된 크기와 진행률을 UI 적으로 표현한다.
- File Size : 처음 파일 추가시 전체사이즈가 나오며, 진행상태에 따라 사이즈 값과 단위가 변화한다.
- Remove Icon : 업로드 시작 이전에 개별 리스트의 삭제가 가능하며, 이 삭제 버튼의 실행을 통해 동작하게 된다.
- Tooltip : 긴 파일명의 경우, 파일명에 마우스 오버시 tooltip의 형태로 보여준다.

파일 리스트로 나오는 모든 요소들은 jquery를 이용하여 모두 동적으로 HTML 구문이 생성되어 보여지는 것이므로, 위 정보들의 수정/추가/삭제가 필요하면 javascript의 구문을 직접 수정해야 한다.

3. 파일 업로드 UI
   : Upload버튼을 실행하여 추가된 파일들을 실제 서버로 업로드 한다.

- 파일 리스트 : 완료된 파일 리스트들에 대해서 비 활성화 되도록 처리하여 리스트삭제 등이 동작되지 않도록 한다.
- 완료 : 전체 파일리스트의 업로드가 완료되면 파일 전송에 성공하였다는 메시지를 업로드 버튼 옆에 표시한다.
- 미완료 : 파일업로드 도중 사이즈 등의 이유로 미완료 되었을 경우, 미완료된 해당 리스트에 대해 경고문과 함께 삭제링크가 나온다. 리스트를 삭제후 나머지리스트는 다시 진행가능하다.

1.3.파일업로드 기능

파일업로드 구현 시, 업로드 목표 위치에 대한 종류를 명시하여 처리하도록 구현되었다. 현재는 disk 타입만 존재하나 향후 이 부분을 확장하면 remote, db 등의 target 으로 확장가능하게 구현될 수 있다.

기본적인 동작과정은 client-side에서 ajax로 호출 시 parameter로써, "utype" 값을 명시하면, java library(FileUploadAgent)에서 이 값을 받아 향후 업로드 되는 파일을 목표에 맞게 처리하는 형태로 이루어 진다. (단, 현재는 "disk" 형태만 지원)

- client-side 부분 : jQuery ajax의 option 중, data 부분에 다음과 같이 전송될 파라미터 값인 utype을 입력한다.(jquery.fileupload.js 에서 구현)

```
data: {"utype": "disk"}
```

  여기서 "disk"부분을 다른 저장형태로 바꾸어 값을 줄 수 있다. ("remote", "db" 와 같은 다른 방식으로 확장가능. 단, sever-side에서 처리부분이 구현되어 있어야한다.)

- server-side 부분 : utype값에 따라 다르게 처리가능하도록 다음의 부분을 추가 구현할 수 있다. (FileUploadAgent.java 에서 구현)

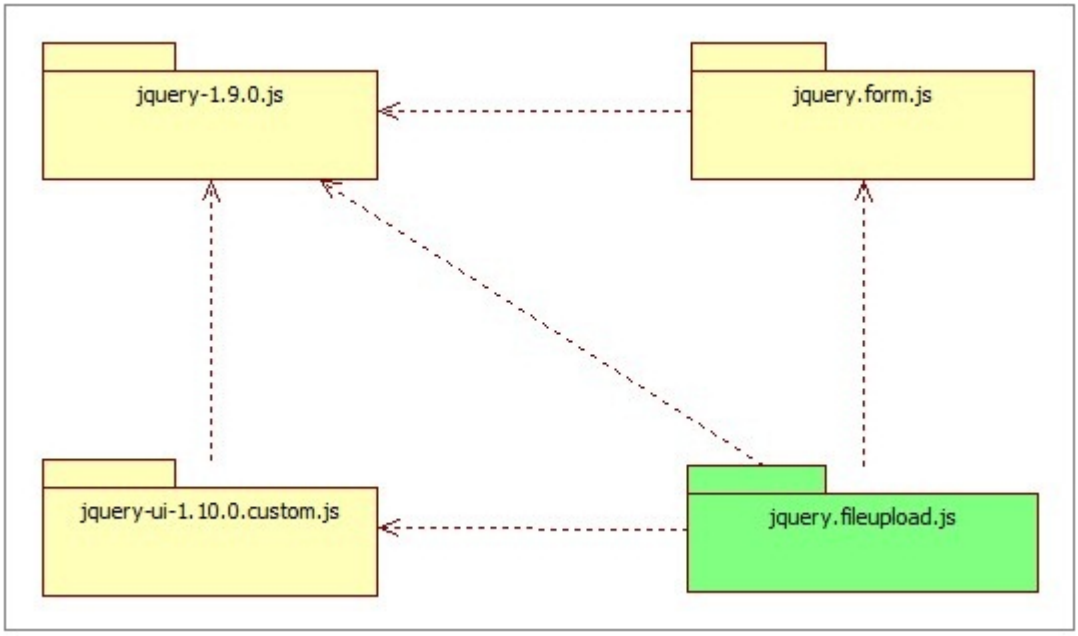
```
if ("disk".equals(utype.toLowerCase())) {
   iFu = new FileUploadLocalDisk();
}
else if ("remote".equals(utype.toLowerCase())) {
   //TODO: (추가 구현 토직..)
}
```

  즉, utype의 값에 따라 저장대상유형을 비교하여 분기 처리한다.

1.4.Javascript Libarary

파일업로드 생성 및 사용에 필요한 필수 javascript 라이브러리는 총 4개이며, 추가적으로 스타일 정의를 위한 css파일 2개, 실제 사용을 위해 필요한 JSP 파일 1개가 필요하다.

4개의 javascript 라이브러리는 각 jquery, jquery ui, form plugin, fileupload plugin 으로 구성되어 있으며, 이들간의 의존관계가 존재한다. 다음은 해당 의존관계를 보여주는 다이어그램이다.



jquery ui와 form plugin은 jquery core 라이브러리에 dependency가 존재하며, fileuplad plugin은 jquery, jquery ui, form plugin 모두에 dependency가 존재한다.

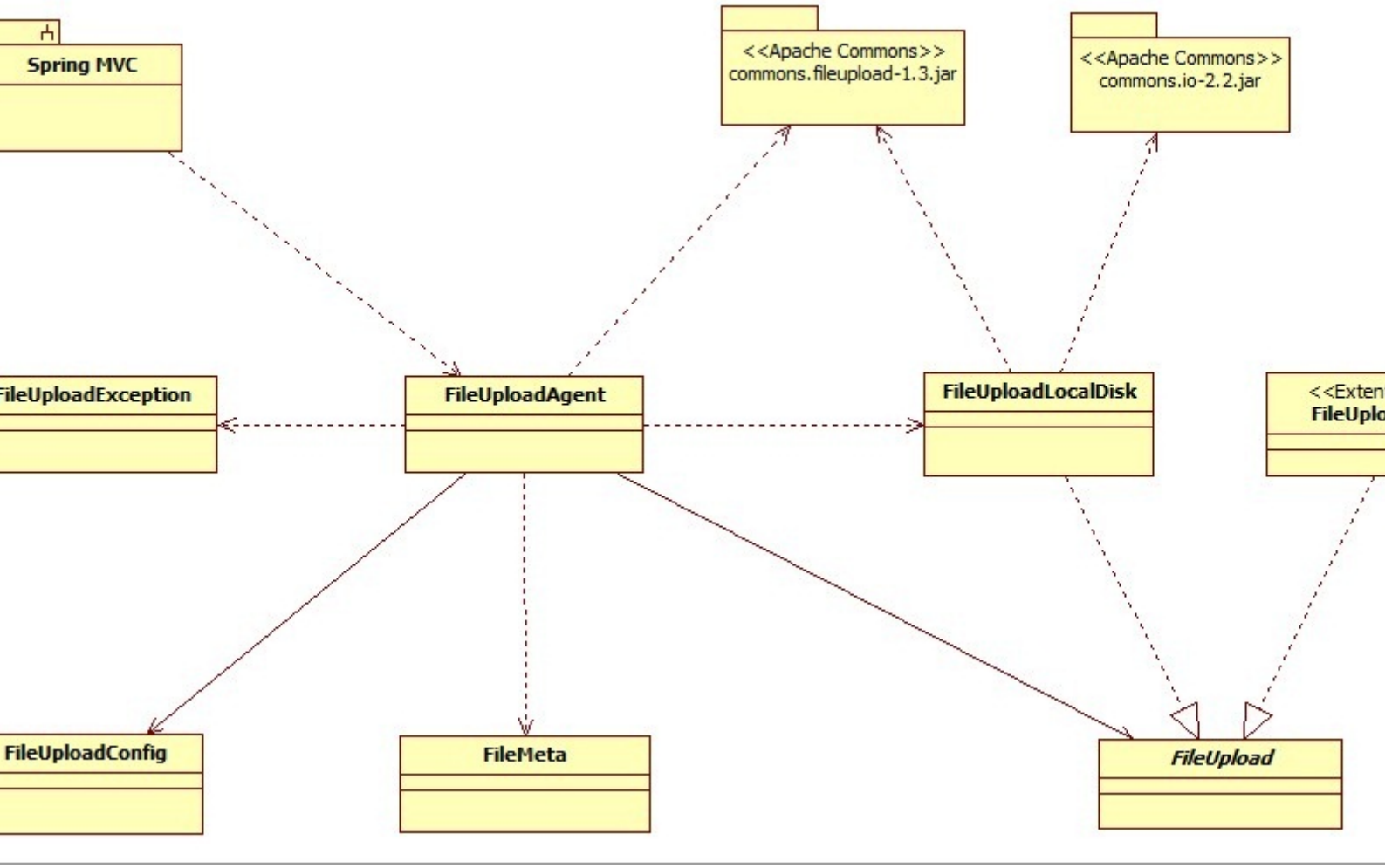
javascript 라이브러리를 포함, 각 관련 파일들의 상세내용은 다음의 표를 참조한다.

파일명	설명
jquery-1.9.0.js	jquery core 함수들이 존재하는 라이브러리로 파일업로드의 전반적인 UI 및 기능을 구현하는데 사용된 JS 파일
jquery-ui-1.10.0.custom.js	jquery ui core 함수들이 존재하며, 파일업로드 UI 내부의 button, progressbar, tooltip 의 ui를 구현하는데 사용된 JS 파일
jquery.form.js	form 형식의 ajax 통신관련된 부분을 지원하는 jquery plugin 형태의 JS 파일
jquery.fileupload.js	파일업로드의 화면단의 여러기능을 구현하여 지원하는 JS 파일
jquery_ui_style.css	파일업로드에서 쓰이는 공통적인 ui style을 재정의한 CSS 파일
fileuploader_style.css	파일업로드 UI 스타일을 정의한 CSS 파일
form.jsp	파일업로드의 UI를 생성하여 사용하는 JSP 파일

1.5.Java Libarary

파일업로드 생성 및 사용에 필요한 필수 java 라이브러리는 총 7개이다.

Spring MVC와의 연결을 구현한 JFileUploadController와 서버측 기능을 구현한 FileUploadAgent, FileUploadException, FileuploadConfig, FileMeta, FileUpload와 실제 local disk로의 저장기능을 구현한 FileUploadLocalDisk 으로 구성되어 있으며, 각 클래스간의 연결관계는 다음과 같다.



메인 클래스인 FileUploadAgent 를 중심으로, 파일의 메타정보를 표현하는 FileMeta, 기본 설정정보를 처리하는 FileUploadConfig 등을 호출한다. 실제 파일업로드를 위해서는 FileUpload 라는 인터페이스를 각 타겟에 맞게 구현하도록 설계되었다.

각 java 라이브러리 파일들의 개별 상세내용은 다음과 같다.

파일명	설명
FileUploadAgent.java	파일 업로드 기능을 처리하는 메인 클래스. 파일 HttpRequest 객체로부터 파일 메타정보를 얻어 값을 설정, 초기화, 저장.
FileUploadAgentException.java	파일 업로드 시 나타나는 오류를 처리하는 Exception 클래스
FileUploadConfig.java	파일 업로드가 처리되기 위한 기본적인 환경정보를 처리하는 클래스
FileMeta.java	파일 사이즈, 파일명 등의 메타정보 표현하는 클래스
JFileUploadController.java	파일업로드를 위해 FileUploadAgent 에 입력 정보를 넘겨주는 역할을 하는 Controller 클래스
FileUpload.java	파일업로드 기능에서 생성, 초기화, 저장, 진행상태 등의 공통 메소드 인터페이스를 갖는 추상 클래스
FileUploadLocalDisk.java	디스크 저장방식에 맞게 FileUpload 인터페이스를 구현한 클래스(설정, 초기화, 저장, 진행상태)

2.파일 업로드 설정 및 구현

화면에서 파일업로드 컴포넌트를 생성하기 위해 라이브러리 파일을 incude 하거나 필수적으로 설정해야하는 부분이 있다. 본 chapter에서는 이와 관련된 설정과 파일업로드의 개별 기능에 대한 상세 구현 내용에 대해 설명하고자 한다.

2.1.화면개발을 위한 javascript 라이브러리 url 설정

fileupload를 사용하기 위해서 jquery, jquery-ui, jquery-form plugin, jquery-fileupload plugin 과 관련 css파일이 필요하다. 따라서 다음과 같이 url을 설정해 주어야 한다.

```
<!-- FileUpload javascript libraries
 ------------------------------------------------------------>
<script type="text/javascript" src="<c:url value='/fileupload/jquery/jquery-1.9.0.js'/>"></
script>
<script type="text/javascript" src="<c:url value='/fileupload/jquery/jquery-
ui-1.10.0.custom.js'/>"></script>
<script type="text/javascript" src="<c:url value='/fileupload/jquery/jquery.form.js'/>"></
script>
<script type="text/javascript" src="<c:url value='/fileupload/jquery/jquery.fileupload.js'/
>"></script>

<!--  FileUpload css files
 ------------------------------------------------------------------------>
<link rel="stylesheet"  type="text/css" href ="<c:url value='/fileupload/css/
jquery_ui_style.css'/>" />
<link rel="stylesheet"  type="text/css" href ="<c:url value='/fileupload/css/
fileuploader_style.css'/>" />
```

참고

* 각 라이브러리간의 dependency가 존재하므로, 라이브러리를 참조하는 순서가 다음과 같이 정의되어야 한다.

: jQuery -> jQuery UI -> form plugin > fileupload plugin

2.2.화면 UI 생성함수 호출 및 설정

2.2.1.함수로드 및 fileupload 생성

ready() 함수는 DOM이 완전히 로드된 후에 실행되는 함수로, 위의 모든 라이브러리를 로드한 후 실행되게 된다. 파일업로드를 사용하기 위해서 context 와 id 값을 설정한 후, init함수를 호출하여 위의 javscript를 참조, 파일업로드를 기본UI를 만든다. 다음과 같이 현재 실행되는 JSP의 context 정보를 FILEUPLOAD.CONTEXT 에 지정, html 태그에서 사용한 id값은 FILEUPLOAD.ID 에 지정해 주어야 한다.

```
$(document).ready(function() {

 FILEUPLOAD.CONTEXT = "${ctx}";
 FILEUPLOAD.ID = "fileuploader";
```

```
 fileupload.init();

});
```

2.2.2.fileupload 생성할 html 태그 설정

html에서 fileupload를 생성하고자 하는 위치에 사용할 html 태그를 작성하고, id값을 함께 지정해주어야 한다. 이 id 값을 기준으로 파일업로드가 생성될 때 필요한 모든 내부태그와 스타일이 동적으로 작성, 적용되게 된다. 또한 태그와 같은 경우, 아래와 같이 div를 사용하도록 권장한다.

```
<div  id ="fileuploader"></div>
```

2.3.File Size 구현

파일업로드를 위해 사용자가 파일을 추가할 때마다 각 파일에 대한 용량 전체크기 정보가 필요하며, 이를 위해 파일 전체크기 정보를 요청하고 응답결과를 얻어오는 과정이 이루어지게 된다. 파일 전체크기 정보가 필요한 시점은 파일을 선택하여 놓는 순간이며, 아직 업로드가 되지 않은 상태에서 전체크기에 대한 정보를 사용자에게 알려주기 위한 게시용 정보라고 볼 수 있다.

1. 전체 사이즈 정보 요청하기
   전체 사이즈 값은 사용자가 파일은 선택하는 즉시 알 수 있도록 하기 위하여 업로드 전, 선택한는 시점에서 요청을 먼저 보내고 정보를 받아오는 형태로 이루어 진다. 다음은 Controller에서 method=getmetaFake 일 때 처리하는 부분이다.(JFileUploadController.java) 여기서 request객체를 FileUploadAgent의 getMetaFake함수에서 받아 요청을 처리하도록 되어있다. (getMetaFake 함수는 request 정보 중, contentLength함수로 파일크기 값을 얻고, 이를 FileMeta 정보형태에 맞게 매핑시킨다.)

```
@RequestMapping(params = "method=getmetaFake", method = RequestMethod.POST)
 public void getFileMetaFake(HttpServletRequest request, HttpServletResponse response,
 HttpSession session) throws JSONException, IOException {

  FileUploadAgent fuAgent = null;

  if ( (fuAgent = (FileUploadAgent)
 session.getAttribute(FileUploadAgent.SESSION_OBJECT_NAME)) == null )  {
   fuAgent = new FileUploadAgent();
   session.setAttribute(FileUploadAgent.SESSION_OBJECT_NAME, fuAgent);
  }

  try {
   fuAgent.getMetaFake(request);
   response.setStatus(HttpServletResponse.SC_ACCEPTED);
  } catch (FileUploadAgentException e) {
   response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, e.getMessage());
  }
 }
```

clinet-side에서 ajax 요청을 하는 부분은 다음과 같다.(jquery.fileupload.js)

```
// getSizeFake 요청
var options = {
```

```
 async : true,
 url : FILEUPLOAD.CONTEXT+"/fileuploadJFileUpload.do?method=getmetaFake",
 beforeSend : function(jqXHR, settings) {
  curjqXHR = jqXHR;
  return true;
 },
 success : function(data) {
  $.ajax(optionGetSize);
 },
 error : function(jqXHR, textStatus, errorThrown) {
  if (errorThrown == "abort")
   $.ajax(optionGetSize);
 }
};

$form.ajaxSubmit(options);

// getSizeReal 요청
var optionGetSize = {
   async : true,
   url : FILEUPLOAD.CONTEXT+"/fileuploadJFileUpload.do?method=getmetaReal",
   data : {
    filename : infilename
   },
   success : function(data) {
    var retJObj = $.parseJSON(data);
    callbackFileuploadObj.updateSize(vfIndex, retJObj.filesize);
   },
   error : function(jqXHR, textStatus, errorThrown) {
   }
  };

//요청 취소
function getRealSizeNCancelRequest() {
   if (curjqXHR != null) {
    clearInterval(hAbort);
    curjqXHR.abort("abort");
   }
}

var hAbort = setTimeout(getRealSizeNCancelRequest, FILEUPLOAD.GET_SIZE_INTERVAL);
```

- $form.ajaxSubmit(option) : $form은 동적으로 생성한 form 태그와 내부 input태그가 있는 모든 html을 묶어 처리한 선택자이다. 따라서 options에 설정한 각 async, url, beforeSend, success, error 옵션은 이 form 선택자에 대해 비동기식의 ajax 요청의 parameter로 보내지게 된다. success 의 경우, 아래의 optionGetSize와 같은 내용을 담은 ajax를 호출하게 된다.

- $.ajax(optionGetSize) : optionGetSize는, sync, url, data, success, error의 옵션내용으로 구성되어 있으며, 이 옵션의 successs 경우, 사용되는 변수 retJObj는 parameter로 얻어오는 data를 JSON형태로 변환하여 받아오는 값이다. (data는 byte단위의 filesize와 filename으로 이루어져 있다.) client-side 에서는 이렇게 retJObj로 받아온 파일사이즈 정보를 실제 화면에서 숫자값으로 나타나게 한다.(updateSize 함수호출 부분)

- getRealSizeNCancelRequest 메소드 : 요청이 처리되지 못할 때, 자동적으로 취소처리를 하는 부분이다. setTimeOut으로 설정한 것에 대해 clear하고, abort시켜서 취소 동작이 이루어지도록 한다.

2. 전체 사이즈 정보 구하기
   위의 client-side에서 ajax로 요청한 결과, 성공하였을 경우, "method=getmetaReal" 인 요청을 하고, 그 응답결과를 넘겨주는 처리를 한다. 또한, JSON 형태로 데이터를 넘겨주기 위해 결과값을 filename,

filesize의 key값에 각 value를 매핑시켜주도록 하고 있다. 다음은 Controller에서 그러한 내용을 포함하고 있는 부분이다.(JFileUploadController.java)

```
@RequestMapping(params = "method=getmetaReal", method = RequestMethod.GET)
public void getFileMetaReal(HttpServletRequest request, HttpServletResponse response,
 HttpSession session) throws JSONException, IOException {

 FileUploadAgent fuAgent = (FileUploadAgent)
 session.getAttribute(FileUploadAgent.SESSION_OBJECT_NAME);
 String filename =
 Normalizer.normalize( request.getParameter(FileUploadAgent.GETMETAREAL_PARAM),
 Normalizer.NFC );

 if (filename == null || filename.length() <= 0) {
  response.sendError(HttpServletResponse.SC_BAD_REQUEST);
  return;
 }

 try {
  FileMeta fm = fuAgent.getMeta(filename);

  JSONObject jsonResult = new JSONObject();
  jsonResult.put("filename", filename);
  jsonResult.put("filesize", fm.getSize()); //TODO: check null

  response.setContentType("text/JSON");
  response.setCharacterEncoding("utf-8");
  response.getWriter().write(jsonResult.toString());
 }
 catch (Exception e) {
  response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, e.getMessage());
 }

}
```

2.4.File Name 구현

client-side에서 type=file 인 input의 value로 전체 파일 경로 정보를 얻게 된다. 이 때, 전체 파일경로 정보로부터 단순 (파일이름.확장자) 와 같은 형식의 값을 얻어내기 위한 처리가 필요하다. 경로구분자인 " \"로 string을 분리시킨 후, 마지막 배열로 저장되는 값을 사용하게 된다. (ex> path = "D:\dev\workspace\anyframe-fileupload-pi.zip" 와 같은 파일경로인 경우, 리턴되는 결과값은 fileName = "anyframe-fileupload-pi.zip" 이다.) 다음은 파일명을 얻어내기 위한 메소드 _getFileName 이다.

```
_getFileName : function(formId) {

  var path = $("#"+formId).children("input[type='file']").val();

  var arryParams = path.split("\\"),
     fileName = arryParams[ arryParams.length - 1 ];

  return fileName;
},
```

2.5.File Upload 구현

파일업로드에서 여러가지 업로드의 저장방식 중 본 파일업로드 플러그인은 disk 저장방식을 구현하고 있다. FileUploadAgent의 upload 함수로 요청을 처리하며, 그 요청결과에 대한 응답을 JSON 타입으로 넘겨주도록 하고 있다. 다음은 Controller에서 method=upload 일 때 처리하는 부분이다. (JFileUploadController.java)

```
@RequestMapping(params = "method=upload", method = RequestMethod.POST)
 public void upload(HttpServletRequest request, HttpServletResponse response, HttpSession
 session) throws JSONException, IOException {

  FileUploadAgent fuAgent = (FileUploadAgent)
 session.getAttribute(FileUploadAgent.SESSION_OBJECT_NAME);

  if (fuAgent == null) {
   response.sendError(HttpServletResponse.SC_BAD_REQUEST);
   return;
  }

  JSONObject jsonResult = new JSONObject();
  response.setContentType("text/JSON");
  response.setCharacterEncoding("utf-8");

  try {
   fuAgent.upload(request);
   jsonResult.put("succ", true);
   response.getWriter().write(jsonResult.toString());
  } catch (FileUploadAgentException e) {
   jsonResult.put("succ", false);
   jsonResult.put("msg", e.getMessage());
   response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, jsonResult.toString());
  }
 }
```

다음은 실제 저장하는 기능 서비스를 구현한 save 메소드이다.(FileUploadLocalDisk.java) Apache Commons의 Streaming API를 사용하여 스트리밍방식으로 저장되는 방식을 택하고 있다.

```
public boolean save(HttpServletRequest request)  throws FileUploadException, IOException {
...
...
        FileItemIterator itor;

   itor = hUpload.getItemIterator(request);

   while (itor.hasNext()) {

          FileItemStream fiStrm = itor.next();

          String fname = FileMeta.getFileNameOnly(fiStrm.getName());

          if (!fiStrm.isFormField()) {
           InputStream inStrm = fiStrm.openStream();

           File file  = new File(targetPath + "/" + fname);
           FileOutputStream fo = new FileOutputStream(file);
```

```
            Streams.copy(inStrm, fo, true);
            inStrm.close();
           }
          }
...
}
```

2.6.File Progress 구현

파일을 업로드할 때, progressbar의 상태가 업데이트 되도록 지속적으로 progress 정보를 요청, 응답받아오도록 하고 있다. disk 저장방식으로 서비스를 구현한 FileUploadLoaclDisk 클래스에서 getProgress 메소드는 전체 파일 크기 중 업로드완료되어진 파일의 크기를 백분율로 계산하여 매핑시키고 있다. 다음은 FileUploadLoaclDisk에서 getProgress 메소드의 구현부분이다.(FileUploadLoaclDisk.java)

```
public long getProgress(HashMap<String, Long> map) {

 long nProgress = (content_length <= 0) ? 0 : (int)(((float)readbyte_length /
 (float)content_length) * 100L);

 map.put(FileUploadAgent.PROGRESS_PARAM1_PROGRESS, nProgress);
 map.put(FileUploadAgent.PROGRESS_PARAM2_CONTENTBYTE, content_length);
 map.put(FileUploadAgent.PROGRESS_PARAM3_READBYTE, readbyte_length);

 return nProgress;
}
```

clinet-side에서는 요청이 성공하였을 경우, 응답받은 progress값으로 실제 화면에서 size값과 progressbar의 value 값이 바뀌게 되면서 상태가 업데이트되도록 한다. 또한, 진행이 모두 완료되었을 경우와 아닌 경우를 나누어 더이상 요청이 이루어지지 않고, 완료되었을 때 필요한 다른 액션이 일어나게 된다. 다음은 이러한 과정에 대한 구현내용이다.(jquery.fileupload.js)

```
var options2 = {
  async : true,
  url : FILEUPLOAD.CONTEXT+"/fileuploadJFileUpload.do?method=progressReal",
  success : function(data) {
   var retJObj = $.parseJSON(data);
   fileupload.updateCurrentSize(vfIndex, retJObj.readbyte, retJObj.progress);
   progressResult = retJObj;
  },
  error : function(data) {
   clearInterval(hAbort);
   curjqXHR.abort();
  }
 };

function checkProgress() {
  if (progressResult != undefined && progressResult.progress == 100) {
   clearInterval(hAbort);

   fileupload.completeUnitList(vfIndex);

   if (vfIndex == (vf.length - 2)) {
    fileupload.completeAll(true);
   }
```

```
    if ((vfIndex + 1) < (vf.length - 1)) {
     setTimeout(VFUpload, 0);
    }
   } else {
    $.ajax(options2);
   }
}
```

3.기타

3.1.브라우저별 Size 제약사항

파일 업로드 시, Size 에 따른 브라우저 별 제약사항에 대하여 아래와 같이 정리한다.

Browser	size < 1G	size < 2G	size > 2G	비고
IE7	pass	pass	fail	
IE8	pass	pass	fail	
IE9	pass	pass	pass	4G 미만인 경우에만 pass
Chrome	pass	pass	pass	
Safari	pass	pass	fail	
Firefox	pass	pass	pass	

* 위에서 언급한 Size 는 실제 파일 크기 + Form 전송 시 Http Message 에 추가되는 내용을 더한 크기를 의미한다.

3.2.참고 사이트

- Commons FileUpload http://commons.apache.org/proper/commons-fileupload/
- jQuery http://api.jquery.com/
- jQuery UI http://jqueryui.com/
- jQuery Form Plugin http://plugins.jquery.com/form/