StrutsUpload

Overview

Presently the Struts code does the following: (1) wrap the request in a multipart request that has to be continually unwrapped prior to use and prior to being initialized anyway; (2) parse the request inside the population of the ActionForm, i.e. in RequestUtils.

This is excellent, since the deferment of anything substantial until RequestUtils allows the Struts Team, should you choose to take this assignment, the opportunity to make Struts v1.3 accessible to people wanting to build upload applications.

As things stand, you have to change Struts to build an upload application with any sophistication. Making Struts more flexible is rather easy. Just use the following three interfaces.

An example of how someone might want to use these interfaces is also given following the interfaces.

If the Struts Team wants to stay with the present idea of providing a multipart request to the Action for backwards compatibility, then this can be achieved by populating both such a request, which is really useless, and a MultipartData class.

Framework Code

MultipartFile

```
public interface MultipartFile
    extends Serializable {
  public long        getSize();
  public void        setSize(long fileSize);
  public String      getName();
  public void        setName(String fileName);
  public String      getContentType();
  public void        setContentType(String fileContentType);
  public byte[]      getData();
  public InputStream getInputStream();
  public void        reset();
}
```

MultipartData

```
public interface MultipartData {
  public Iterator getParameterNames();
  public String   getParameter(String name);
  public String[] getParameterValues(String name);
  public Map      getFiles();
}
```

MultipartHandler

```
public interface MultipartHandler {
  public void          handleRequest(Object [] params) throws IOException;
  public ActionMapping getMapping();
  public void          setMapping(ActionMapping mapping);
  public ActionServlet getServlet();
  public void          setServlet(ActionServlet servlet);
}
```

Framework Code Sample Implementation

UploadMultipartFile

```
public class UploadMultipartFile
    implements MultipartFile {
  private String   contentType;
  private String   name;
  private long     size;
```

```
private FileItem fi;

public UploadMultipartFile() {
}

public UploadMultipartFile(String name,
                           String contentType,
                           long   size) {
  this.name        = name;
  this.contentType = contentType;
  this.size        = size;
}

public UploadMultipartFile(FileItem fi) {
  this.fi = fi;
}

public long getSize() {
  return size;
}

public void setSize(long size) {
  this.size = size;
}

public String getName() {
  return name;
}

public void setName(String name) {
  this.name = name;
}

public String getContentType() {
  return contentType;
}

public void setContentType(String contentType) {
  this.contentType = contentType;
}

public byte[] getData() {
  InputStream is = getInputStream();
  if(is != null) {
    int i = (int)getSize();
    if(i > 0) {
      byte data[] = new byte[i];
      try {
        is.read(data);
        is.close();
      } catch(IOException ioe) { }
      return data;
    } else {
      return null;
    }
  } else {
    return null;
  }
}

public void reset() {
  if(fi != null) {
    fi.delete();
  }
}

public InputStream getInputStream() {
  if(fi == null) {
    return null;
  }
```

```
    try {
      return new BufferedInputStream(fi.getInputStream());
    } catch (IOException ioe) {
      return null;
    }
  }

  public FileItem getFileItem() {
    return fi;
  }
}
```

UploadMultipartData

```
public class UploadMultipartData
    implements MultipartData {
  private Map parameterNames;
  private Map files;

  public UploadMultipartData(HttpServletRequest req,
                             List               monitors,
                             String             encoding,
                             int                maxFileSize)
      throws UploadException,
             IOException {

    if(req == null) {
      new UploadException(UploadConstant.INVALID_REQUEST);
    }

    parameterNames = Collections.synchronizedMap(new HashMap(89));
    files          = Collections.synchronizedMap(new HashMap(89));

    MultipartHandler handler = new UploadMultipartHandler();

    Object [] objects = new Object [] { req,                              // 0
                                        encoding,                         // 1
                                        monitors,                         // 2
                                        parameterNames,                   // 3
                                        files,                            // 4
                                        UploadConstant.SYSTEM_TEMPDIR,    // 5
                                        new Integer(maxFileSize) };       // 6
    handler.handleRequest(objects);
  }

  public UploadMultipartData(HttpServletRequest req,
                             List               monitors,
                             int                maxFileSize)
      throws UploadException,
             IOException {
    this(req, monitors, UploadConstant.DEFAULT_ENCODING, maxFileSize);
  }

  public UploadMultipartData(HttpServletRequest req,
                             List               monitors)
      throws UploadException,
             IOException {
    this(req, monitors, UploadConstant.MAX_FILE_SIZE);
  }

  public UploadMultipartData(HttpServletRequest req)
      throws UploadException,
             IOException {
    this(req, null, UploadConstant.MAX_FILE_SIZE);
  }

  public Iterator getParameterNames() {
    return this.parameterNames.keySet().iterator();
  }
```

```
  public String getParameter(String name) {
    List parameterValues = (List)parameterNames.get(name);

    if(parameterValues == null || parameterValues.size() == 0) {
      return null;
    }

    return (String)parameterValues.get(parameterValues.size() - 1);
  }

  public String[] getParameterValues(String name) {
    List parameterValues = (List)parameterNames.get(name);
    if(parameterValues == null || parameterValues.size() == 0) {
      return null;
    }
    return (String [])parameterValues.toArray();
  }

  public Map getFiles() {
    return (Map)files;
  }
}
```

UploadMultipartHandler

```
public class UploadMultipartHandler
    implements MultipartHandler {
  private ActionMapping mapping;
  private ActionServlet servlet;

  public UploadMultipartHandler() {
  }

  public void handleRequest(Object [] params)
      throws IOException {
    handleRequest((HttpServletRequest)params[0],              // req
                  (String)params[1],              // encoding
                    (List)params[2],              // monitors
                     (Map)params[3],              // parameterNames
                     (Map)params[4],              // files
                  (String)params[5],              // repositoryPath
                 ((Integer)params[6]).intValue()); // maxFileSize
  }

  private void handleRequest(HttpServletRequest req,
                             String             encoding,
                             List               monitors,
                             Map                parameterNames,
                             Map                files,
                             String             repositoryPath,
                             int                maxFileSize)
      throws IOException {
    UploadFileItemFactory ufiFactory = new UploadFileItemFactory();

    ufiFactory.setMonitors(monitors);
    ufiFactory.setContentLength(req.getContentLength());

    DiskFileUpload dfu = new DiskFileUpload();

    dfu.setFileItemFactory(ufiFactory);
    dfu.setSizeMax(maxFileSize);
    dfu.setSizeThreshold(UploadConstant.BUFFER_SIZE);
    dfu.setRepositoryPath(repositoryPath);

    if(encoding != null) {
      dfu.setHeaderEncoding(encoding);
    }
```

```
    List list = null;

    try {
      list = dfu.parseRequest(req);
    } catch(FileUploadException cfue) {
      throw new IOException(cfue.getMessage());
    }

    Object obj = null;

    for(Iterator iterator = list.iterator(); iterator.hasNext();) {
      FileItem fi        = (FileItem)iterator.next();
      String   fieldName = fi.getFieldName();

      if(fi.isFormField()) {
        String data = null;
        if(encoding != null) {
          data = fi.getString(encoding);
        } else {
          data = fi.getString();
        }

        List names = (List)parameterNames.get(fieldName);

        if(names == null) {
          names =  Collections.synchronizedList(new LinkedList());
          parameterNames.put(fieldName, names);
        }

        names.add(data);
      } else {
        String name        = fi.getName();
        String ext         = name;
        String contentType = null;
        int    flag        = -99;
        if(name != null) {
          MultipartFile mf = new UploadMultipartFile(fi);
          mf.setName(name);
          mf.setContentType(contentType = fi.getContentType());
          mf.setSize(fi.getSize());
          files.put(fieldName, mf);
        }
      }
    }
  }

  public ActionServlet getServlet() {
    return this.servlet;
  }

  public void setServlet(ActionServlet servlet) {
    this.servlet = servlet;
  }

  public ActionMapping getMapping() {
    return this.mapping;
  }

  public void setMapping(ActionMapping mapping) {
    this.mapping = mapping;
  }
}
```

MultipartUtil

```
public class MultipartUtil {
  public  static final String MULTIPART_FORM_DATA = "multipart/form-data";
  public  static final String CONTENT_TYPE        = "Content-Type";

  public static boolean isMultipartFormData(HttpServletRequest req) {
    String contentType        = null;
    String headerContentType  = req.getHeader(CONTENT_TYPE);
    String requestContentType = req.getContentType();

    if(headerContentType == null && requestContentType != null) {
      contentType = requestContentType;
    } else if(requestContentType == null && headerContentType != null) {
      contentType = headerContentType;
    } else if(headerContentType != null && requestContentType != null) {
      contentType = headerContentType.length() <= requestContentType.length() ? requestContentType :
headerContentType;
    }

    return contentType != null && contentType.toLowerCase().startsWith(MULTIPART_FORM_DATA);
  }
}
```

Sample Application Code Pieces

UploadOutputStream

```
public class UploadOutputStream
    extends DeferredFileOutputStream {
  private List  monitors;
  private boolean isFormField;

  public UploadOutputStream(int     threshold,
                            File    outputFile,
                            List    monitors,
                            boolean isFormField) {
    super(threshold, outputFile);
    this.monitors = monitors;
    this.isFormField = isFormField;
  }

  public void write(byte data[], int i, int j)
      throws IOException {
    super.write(data, i, j);
    if((monitors != null) && (! isFormField)) {
      for(int k = 0; k < monitors.size(); k++) {
        Monitor monitor = (Monitor)monitors.get(k);
        monitor.read(j);
      }
    }
  }
}
```

UploadFileItem

```
public class UploadFileItem
    implements FileItem {
  private static volatile int number = 0;
  private UploadOutputStream  uos;
  private File                tempDir;
  private List                monitors;
  private String              fieldName;
  private String              contentType;
  private String              fileName;
  private byte []             data;
  private int                 threshold;
```

```
private int                totalSize;
private boolean            isFormField;

public UploadFileItem(String  fieldName,
                      String  contentType,
                      boolean isFormField,
                      String  fileName,
                      List    monitors,
                      File    tempDir,
                      int     threshold,
                      int     totalSize) {
  this.fieldName   = fieldName;
  this.contentType = contentType;
  this.isFormField = isFormField;
  this.fileName    = fileName;
  this.monitors    = monitors;
  this.tempDir     = tempDir;
  this.threshold   = threshold;
  this.totalSize   = totalSize;
}

public InputStream getInputStream()
    throws IOException {
  if(!uos.isInMemory()) {
    return new FileInputStream(uos.getFile());
  }

  if(data == null) {
    data = this.uos.getData();
  }

  return new ByteArrayInputStream(data);
}

public String getContentType() {
  return contentType;
}

public String getName() {
  return fileName;
}

public boolean isInMemory() {
  return uos.isInMemory();
}

public long getSize() {
  if(data != null) {
    return (long)data.length;
  }

  if(uos.isInMemory()) {
    return (long)uos.getData().length;
  } else {
    return uos.getFile().length();
  }
}

public byte[] get() {
  if(uos.isInMemory()) {
    if(data == null) {
      data = uos.getData();
    }

    StdOut.log("log.devel","UploadFileItem uos.isInMemory() data = " + data);
    return data;
  }

  byte fisData[] = new byte[(int)getSize()];
  FileInputStream fis = null;
  try {
```

```
    fis = new FileInputStream(uos.getFile());
    fis.read(fisData);
  } catch(IOException ioe) {
    fisData = null;
  } finally {
    if(fis != null) {
      try {
        fis.close();
      } catch(IOException ioe1) { }
    }
  }
  return fisData;
}

public String getString(String encoding)
    throws UnsupportedEncodingException {
  return new String(get(), encoding);
}

public String getString() {
  return new String(get());
}

public void write(File file)
    throws Exception {
  if(isInMemory()) {
    FileOutputStream fos = null;
    try {
      fos = new FileOutputStream(file);
      fos.write(get());
    } finally {
      if(fos != null) {
        fos.close();
      }
    }
  } else {
    File outputFile = uos.getFile();

    if(outputFile != null) {
      if(!outputFile.renameTo(file)) {
        BufferedInputStream  bis = null;
        BufferedOutputStream bos = null;
        try {
          bis = new BufferedInputStream(new FileInputStream(outputFile));
          bos = new BufferedOutputStream(new FileOutputStream(file));
          byte streamData[] = new byte[UploadConstant.STREAM_BUFFER_SIZE];
          for(int i = 0; (i = bis.read(streamData)) != -1;)
            bos.write(streamData, 0, i);
        } finally {
          try {
            bis.close();
          } catch(IOException ioe) { }
          try {
            bos.close();
          } catch(IOException ioe1) { }
        }
      }
    } else {
      throw new IOException(UploadConstant.WRITE_ERROR);
    }
  }
}

public void delete() {
  data = null;
  File file = uos.getFile();
  if(file != null && file.exists()) {
    file.delete();
  }
}
```

```
  public String  getFieldName() {
    return fieldName;
  }

  public void setFieldName(String fieldName) {
    this.fieldName = fieldName;
  }

  public boolean isFormField() {
    return isFormField;
  }

  public void setFormField(boolean isFormField) {
    this.isFormField = isFormField;
  }

  protected void finalize() {
    File file = uos.getFile();
    if(file != null && file.exists()) {
      file.delete();
    }
  }

  public OutputStream getOutputStream()
      throws IOException {
    if(uos == null) {
      File file = tempDir;

      if(file == null) {
        file = new File(UploadConstant.SYSTEM_TEMPDIR);
      }

      String fileName = UploadConstant.FILE_PREFIX + (number++) + UploadConstant.TMP_EXT;
      File   outputFile = new File(file, fileName);

      outputFile.deleteOnExit();

      uos = new UploadOutputStream(threshold, outputFile, monitors, isFormField());

      if(monitors != null && !isFormField()) {
        Iterator iter = monitors.iterator();
        while(iter.hasNext()) {
          ((Monitor)iter.next()).init(outputFile, totalSize, getContentType());
        }
      }
    }
    return this.uos;
  }
}
```

UploadFileItemFactory

```
public class UploadFileItemFactory
    extends DefaultFileItemFactory {
  private List monitors;
  private int  contentLength;

  public UploadFileItemFactory() {
  }

  public FileItem createItem(String  fieldName,
                             String  contentType,
                             boolean isFormField,
                             String  fileName) {
    return new UploadFileItem(fieldName,
                              contentType,
                              isFormField,
                              fileName,
                              monitors,
                              getRepository(),
                              getSizeThreshold(),
                              contentLength);
  }

  public void setContentLength(int contentLength) {
    this.contentLength = contentLength;
  }

  public int  getContentLength() {
    return contentLength;
  }

  public void setMonitors(List monitors) {
    this.monitors = monitors;
  }

  public List getMonitors() {
    return this.monitors;
  }
}
```

---

*I have a full application, but no sense putting it all here if there is no interest.