分享文档系列

Jakarta Commons FileUpload 用户指南

作者：

日期：2007-3-22

版本：V1.0(2007-03-22)

较对：hunqiu

前　言

什么也不写。

目　录

第1章　使用 FileUpload

FileUpload 能以多种方式使用，这取决于你的应用需求。举个简单的例子，你可能调用一个单独的方法来解析 servelt 的请求，并且处理那些项目。 从另一个方面来讲，你可能想自定义 FileUpload 来完全控制个别项目的存贮；例如，你想流化那些内容，并存到数据库里去。

这里我们会介绍使用FileUpload的基础原则，并描述一些简单的通用的使用模式。我们会在在其它地方介绍关于FileUpload的自定义。

FileUpload依赖于一些公用的IO，因此，要确保在你继续之前，你的classpath里已经有依赖页面里提起的那些版本。

第2章 FileUpload 如何工作？

一个上传请求由一系列根据 RFC1867（"Form-based File Upload in HTML".）编码的项目列表组成。FileUpload 可以解析这样的请求，并为你的应用提供那些已上传的项目的列表。每一个这样的项目都实现了 FieIItem 接口，我们不用管它们的底层实现。

这个页面描述了commons fileupload库的常用API。这些常用API是非常方便的途径。然而，为了最好的性能，你可能更喜欢最快的Streaming API 。

每一个文件项目有一些自己的属性，这些属性也许正是你的应用程序感兴趣的地方。例如，每个项目有个一个名字和内容类型，并且可以提供一个输入流来访问它们的数据。另一方面来看，你可能需要用不同方式来处理不同的项目，这就依赖于那些项目是否是一个正常的表单字域，也就是说，这些数据来自于一个普通的文本框或类似 HTML 的字域，还是一个要上传文件字段。FileItem 接口提供一些方法来做这样一个决定，并且用最合适的方法访问这些数据。

FileUpload 使用 FileItemFactory 创建一个新的文件项目。这将会给 FileUpload 最好的灵活性。工厂最终控制每个项目如何被创建。默认的工厂在内存或者硬盘里存储项目的数据，这依赖于项目的大小（例如，有多少字节的数据。）。不过，为了适用于你的应用，你还是可以自定义这种行为的。

第3章 servlets and portlets

从 V1.1 版开始，FileUpload 就开始支持 servlet 和 portlet 的文件上传请求。这两种环境的用法基本上差不多，因此，文档的剩下部分都将是在 servlet 环境里。

如果你正在构建一个 portet 应用，那么下面两个差别是你在读文档时应注意的：

你在哪里引用了ServletFileUpload类，就用PortletFileUpload类来替代它。

你在哪里引用了HttpServletRequest类，就用ActionRequest类替代它。

第4章　解析请求

在你同那些上传的项目一起工作前，你需要先解析请求本身。以确保这个请求确实是一个文件上传请求。FileUpload 是通过调用一个静态方法来实现的。

```
// Check that we have a file upload request
boolean isMultipart = ServletFileUpload.isMultipartContent(request);
```

现在，我们已经准备好解析请求里的项目了。

4.1　最简单的例子

下面是一些简单的使用场景：

- 上传项目只要足够小，就应该保留在内存里。
- 较大的项目应该被写在硬盘的临时文件上。
- 非常大的上传请求应该避免。
- 限制项目在内存中所占的空间，限制最大的上传请求，并且设定临时文件的位置。

处理这个场景的请求很简单：

```
// Create a factory for disk-based file items
FileItemFactory factory = new DiskFileItemFactory();
// Create a new file upload handler
ServletFileUpload upload = new ServletFileUpload(factory);
// Parse the request
```

```
List /* FileItem */ items = upload.parseRequest(request);
```

这就是我们所需要的全部代码了！

解析的结果就是一个项目的List,每个项目都实现了FileItem接口。我们将在下面讨论如何处理这些项目。

4.2 训练如何控制

如果你的使用场景和上面那个简单的例子很接近，但是你又需要一点点控制，那么你可以很容易地定义 upload 处理器或者文件项目工厂的行为。下面这个例子显示了几个配置选项。

```
// Create a factory for disk-based file items
DiskFileItemFactory factory = new DiskFileItemFactory();
// Set factory constraints
factory.setSizeThreshold(yourMaxMemorySize);
factory.setRepository(yourTempDirectory);
// Create a new file upload handler
ServletFileUpload upload = new ServletFileUpload(factory);
// Set overall request size constraint
upload.setSizeMax(yourMaxRequestSize);
// Parse the request
List /* FileItem */ items = upload.parseRequest(request);
```

当然，每个配置方法是独立于其它任意一个的。但是如果你想一次性配置他们，你可以用 parseRequest()的另一个重载方法，像这样：

```
// Create a factory for disk-based file items
DiskFileItemFactory factory = new DiskFileItemFactory(
        yourMaxMemorySize, yourTempDirectory);
```

如果你还想使用更多的控制，比如存储项目到其它地方（如，数据库），那么你可以看 FileUpload 自定义介绍。

第5章　处理上传的项目

一旦解析完成，那么你会得到一个待处理的文件项目列表。很多的情况下，你会想用不同的方式来处理文件上传域和正常的表单域，因此，你可以这样做：

```
// Process the uploaded items
Iterator iter = items.iterator();
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();
    if (item.isFormField()) {
        processFormField(item);
    } else {
        processUploadedFile(item);
    }
}
```

对于普通的表单域来说，你可能对项目的名称和字符型值　很感兴趣。就像你希望的那样，照下面的做：

```
// Process a regular form field
if (item.isFormField()) {
    String name = item.getFieldName();
    String value = item.getString();
    ...
```

```
}
```

对于上传文件，这里就有很多不同啦~你可能想知道更多其它的内容。下面是个例子，里面包含了不少你感兴趣的方法。

```
// Process a file upload

if (!item.isFormField()) {

String fieldName = item.getFieldName();

String fileName = item.getName();

String contentType = item.getContentType();

boolean isInMemory = item.isInMemory();

long sizeInBytes = item.getSize();

...

}
```

对于上传的文件，你肯定不希望总是通过内存来访问它，除非它很小，或者你实在没有别的选择余地了。你很希望使用流来处理文件内容或者将文件保存到它的最终位置。FileUpload 提供简单的方式来完成两方面的需求。

```
// Process a file upload

if (writeToFile) {

File uploadedFile = new File(...);

    item.write(uploadedFile);

} else {
```

```
InputStream uploadedStream = item.getInputStream();
    ...
uploadedStream.close();
}
```

注意：在 FileUpload 的默认实现中 wirte()方法应该值得关注，如果数据还在临时文件里没有移除，那么这个方法就会试图重命名这个文件为相应的目标文件。事实上如果重命名失败了的话，数据就仅仅被拷贝。

如果你需要访问内存中的上传数据，你可以用 get()方法来获得数据的二进制数组形式。

```
// Process a file upload in memory
byte[] data = item.get();
...
```

第6章　清除资源

这一节只适用于你使用了DiskFileItem。换句话说，它只适用于你在处理上传文件之前将上传文件写入过临时文件这种情形。

像这种临时文件会被自动删除，如果它们不再被使用（更确切地说，java.io.File 的实例已经被 GC 掉了。　）这是由 org.apache.commons.io.FileCleaner 类在后台完成的，它会启动一个收割机线程。

这个收割机线程在它不再被需要时会被停止。在 servlet 环境里，这是通过指定一个名叫 FileCleanerCleanup 的 servlet 上下文监听器来实现的。要做到这里，在你的 web.xml 增加下面的代码：

```
<web-app>
  ...
  <listener>
    <listener-class>
      org.apache.commons.fileupload.servlet.FileCleanerCleanup
    </listener-class>
  </listener>
  ...
</web-app>
```

不幸的是，事情到这里还没完。如果你和下面的情况一样，那么你就只需要按照上面的做，就可以清除资源了。

你使用的是 commons-io 1.3 或者更晚的版本。

你是从 web 应用的 web-inf/lib 里载入 commons-io 的，并不是从其它位置，如 Tomcat 的 common/lib 下。

如果 commons-io 1.3 是从你的 WEB 容器的 classpath 里载入的，那么，下面的情况可能会出现：

建议你运行两个应用，一个叫 A，一个叫 B。（这两个应用可能是完全一样，只不过上下文名称不一样。）这两个应用都使用了 FileCleanerCleanup。现在，如果你终止应用 A，B 还在运行，这时，A 会终止 B 的收割机线程。换言之，你要十分仔细地考虑是使用 FileCleanerCleanup，还是不使用。

第7章　观察上传进度

如果你希望可以上传很大的文件，这时，你可能想将上传的状态告诉用户，如已经接收了多少。

观察上传进度需要通过一个处理监听器来实现。

```
//Create a progress listener
ProgressListener progressListener = new ProgressListener(){
    public void update(long pBytesRead, long pContentLength, int pItems) {
        System.out.println("We are currently reading item " + pItems);
        if (pContentLength == -1) {
            System.out.println("So far, " + pBytesRead + " bytes have been
read.");
        } else {
                System.out.println("So far, " + pBytesRead + " of " +
pContentLength
                        + " bytes have been read.");
        }
    }
};
upload.setProgressListener(progressListener);
```

上面这个监听器是有问题的。因为它非常频繁地被调用。这会带来性能问题。一个比较好的解决办法是，减少调用。例如，如果 megabytes 被改变，那么就发出一个消息。

```
//Create a progress listener
ProgressListener progressListener = new ProgressListener(){
    private long megaBytes = -1;
    public void update(long pBytesRead, long pContentLength, int pItems) {
        long mBytes = pBytesRead / 1000000;
        if (megaBytes == mBytes) {
            return;
        }
        megaBytes = mBytes;
        System.out.println("We are currently reading item " + pItems);
        if (pContentLength == -1) {
            System.out.println("So far, " + pBytesRead + " bytes have been
read.");
        } else {
                System.out.println("So far, " + pBytesRead + " of " +
pContentLength
                                + " bytes have been read.");
        }
```

```
    }

};
```