**近两天来由于项目需要使用上传组件，于是我仔细分析了Cos和FileUpload的源代码，并对它们的性能进行了测试比较，使用2M、20M、45M、200M的上传大小测试三种组件所花费的时间（单位为ms）结果是：**

2M

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 第1次 | 第2次 | 第3次 | 第4次 | 平均 |
| Cos | 297 | 203 | 234 | 245 | 245 |
| FileUpload | 281 | 312 | 281 | 312 | 297 |
| SmartUpload | 531 | 594 | 485 | 532 | 536 |

20M

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 第1次 | 第2次 | 第3次 | 第4次 | 平均 |
| Cos | 2562 | 2109 | 2719 | 2172 | 2391 |
| FileUpload | 4062 | 4140 | 5360 | 3922 | 4371 |
| SmartUpload | 3453 | 3094 | 3078 | 3547 | 3293 |

45M

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 第1次 | 第2次 | 第3次 | 第4次 | 平均 |
| Cos | 4860 | 4844 | 5125 | 5171 | 5000 |
| FileUpload | 9000 | 8391 | 10375 | 10078 | 9461 |
| SmartUpload | 8265 | 9187 | 8672 | 8856 | 8745 |

200M

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 第1次 | 第2次 | 第3次 | 第4次 | 平均 |
| Cos | 55813 | 52282 | 54796 | 51187 | 53520 |
| FileUpload | 76343 | 68531 | 80954 | 79031 | 76215 |
| SmartUpload | 内存堆栈溢出 |  |  |  |  |

从上述的表格对比中可以看出Cos始终保持着良好的性能。在上传量较小（容量<2M，这是最常出现的情况）时，Cos比FileUpload性能并没有好很多，但SmartUpload就已经开始显出弱势。

   随着容量的增大，FileUpload和SmartUpload的性能下降非常快，直到200M容量时，SmartUpload已经不堪重负崩溃了，而Cos此时的花费时间比FileUpload少了20多秒，不能不说在本次的评测中，Cos的性能位居第一。

通过对三种流行的上传组件进行对比，我认为选用Cos是比较好的。在实际的项目中，可以把上传的文件放到文件夹，文件路径存于数据库中以便于管理。

   如果需要把文件上传到数据库也很简单，从Cos中已经得到了上传文件（java.io.File），其后的操作很平常所做的一样：  通过File得到inputStream，存到数据库的blob或Clob字段即可。

   对于使用Struts的项目，我觉得还是使用FileUpload比较好，因为Struts天生集成了FileUpload的功能，使用FileUpload会带来很多的便利。而如果想要开发独立的上传组件，则用Cos是最好的选择，可以在Cos的基础上封装一层，暴露给业务程序员的只是一些简单易用的API，而且可以给这些API加上自定义的javaDoc，这对于实际的开发和将来的扩展都是非常方便的。  
   下面对FileUpload的上传机制作一些分析，基本上，上传一个文件的过程在FileUpload中可以分为三个部分：  
   1.由客户端把要上传的文件生成request数据流，与服务器端建立连接  
   2. 在服务器端接收request流，将流缓存到内存或磁盘中（具体缓存到什么地方，将由DiskFileUpload的 setSizeThreshold(int cacheMax)方法来决定，当文件大小<cacheMax时，文件将被缓存到内存，否则将被缓存到磁盘的临时文件）  
   3.由服务器端的内存或是临时文件中把文件输出到指定的目录（这个目录才是指定的文件上传目录）.  
     
  
   上述的第一步由浏览器完成，不用过多理会，重点是第二和第三步。  
   第 二步时，由DiskFileUpload的parseRequest(...)方法（其实这个方法是继承于FileUploadBase类，真正起解析 request流作用的类是FileUploadBase）解析request流。在parseRequest(...)方法中，新建了一个 MultipartStream实例，由此实例的readBodyData（）方法将上传文件的流读到FileItem实例中，FileItem实例根据 设置好的cacheMax大小，引用一个内存中的数据流或是一个磁盘上的数据流，注意此时文件已经上传到了服务器，但仍然没有传到设定的上传目录。     
   第三步时，调用FileItem实例 的write(File file)方法，将已经存在于内存或是磁盘上的上传文件流拷贝到设定好的上传目录，至此上传仍未结束，因为磁盘中很可能保存了上传文件的临时文件（当设定 的cacheMax<文件大小时），如何删除这些临时文件？有两种方法：1.显示调用FileItem实例的delete（）方法。2.不调用任何 方法，当FileItem被垃圾回收时，由finalize（）方法删除临时文件。