[Apache Commons 1](#_Toc27153)

[ Apache Commons Proper 1](#_Toc30351)

[1. FileUpload 1](#_Toc4052)

[ The Commons Sandbox 3](#_Toc9978)

[ The Commons Dormant 3](#_Toc6934)

[Guava 4](#_Toc10952)

# Apache Commons

Apache Commons是一个Apache项目，专注于所有层面的可重用Java组件。该项目由三部分组成：

## **Apache Commons Proper**

目标只有一个，创建并维护可重用的Java组件。Commons的开发人员会努力确保他们的组件尽量少的依赖其它库。因此，这些组件很容易部署。另外，该组件会保证其接口尽可能的平稳，因此使用者不需要担心组件以后会不会改变。

### FileUpload

这个包让servlet轻松实现简单粗暴，而且高效的文件上传功能。但HTTP请求得以post方法提交，并且内容类型为multipart/form-data 。表单的默认类型（enctype）是application/x-www-form-urlencoded

总之最后要改成这样：

<form action=””method=”post”enctype=”multipart/form-data”>

</form>

但，改完类型后，就不能再像以前那样用request.getParameter()来获取值了，必须要用fileupload来抽取。

往下的话，我个人是这样理解的，fileupload会处理整个request请求，并返回一个list，类型为List<FileItem>。这个list里不仅包括用户想上传的文件，还包括普通的表单域（如果有的话），因此该list不是到手后就能直接用的，要判断。下面详细讲这个过程：

// 这个类是fileupload自带的，不用想太多，就是获得一个工厂

DiskFileItemFactory factory = new DiskFileItemFactory();

// 临界值，大于这个值，文件会被保存到临时文件夹。设置值得方式为

500\*1024 这个表示500k，单位为1 byte

factory.setSizeThreshold(yourMaxMemorySize);

// 设置临时存储区域，当然不一定非得用ServletContext里的，不过从硬盘角度上讲都一样。

ServletContext servletContext = this.getServletConfig().getServletContext();

File repository = (File)servletContext.getAttribute("javax.servlet.context.tempdir");

factory.setRepository(repository);

// 创建处理器，准备解析器请求

ServletFileUpload upload = new ServletFileUpload(factory);

// 如果需要请求头的话，就把这行加上

upload.setHeaderEncoding(“UTF-8”);

// 可上传文件的最大体积，设置方式同上

upload.setSizeMax(yourMaxRequestSize);

// 解析请求，得到文件list

List<FileItem> items = upload.parseRequest(request);

// 对items的操作肯定是放在循坏里。最近比较喜欢foreach。

for(FileItem item : items){

if(item.isFormField()){ // 是否表单域

// 获取表单域的name，与之相对应的是getName()，获取文件名

String fieldName = item.getFieldName();

// 以字符串的形式返回文件内容，很明显是针对表单域的，抽空可以对文件试试这个方法，不知道会是神马效果，另外，那个“utf-8”是可选的，有重载。

String value = item.getString(“UTF-8”);

.... // 然后你爱干嘛干嘛

} else {

// 非表单域那就是正儿八经的文件了。这里，能一口气获取的属性还是蛮多的。

String fieldName = item.getFieldName();  
    String fileName = item.getName();  
    String contentType = item.getContentType();  
    boolean isInMemory = item.isInMemory();  
    long sizeInBytes = item.getSize();

// 保存文件，这是个很便利的方法，当然也可以像I/O那样用流保存文件

File uploadedFile = new File(“D:\\fileDir\\”+fileName);  
    item.write(uploadedFile);

// 获取输入流，保存文件，又或者查看内容。

InputStream uploadedStream = item.getInputStream();

uploadedStream.close();

}

} //:~

## **The Commons Sandbox**

## **The Commons Dormant**

# Guava

Google出品的一套java工具，可以极大程度上精简我们的代码，提高开发效率。

// 其实就是帮你省下写各种for循坏，各种判断的功夫。

## **eventbus**

大概瞄了一眼，目前是这么理解的：~~事件（一个对象，类似model类，比较单纯），通过某种方式注册给监听者，然后监听者通过某种方式处理这个事件。~~

现在看来不是这么一回事啊。

定义好一个事件，事件是一个很简单的model类。然后再定义监听者，监听者内必须包含一个用@Subscriber修饰的方法。（没有这种方法，定义这个监听者又有什么用呢）接着，把监听者注册给EventBus这个类的对象，紧接着，再用EventBus类的对象来发布事件。

**public** **class** Work {

**public** **static** **void** main(String[] args) {

EventBus bus = **new** EventBus();

bus.register(**new** SimpleSubscriber("订阅者1"));

bus.register(**new** SimpleSubscriber("订阅者2"));

bus.register(**new** SimpleSubscriber("订阅者3"));

**for** (**int** i = 0; i < 50; i++) {

bus.post(**new** SimpleEvent("事件" + i));

}

}

}

**class** SimpleEvent {

**private** String name;

**public** SimpleEvent(String name) {

**super**();

**this**.name = name;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**class** SimpleSubscriber {

**private** String name;

**public** SimpleSubscriber(String name) {

**this**.name = name;

}

@Subscribe

**public** **void** handle(SimpleEvent event) {

System.***out***.println(name + " 开始处理 " + event.getName());

}

}

输出结果令我很惆怅。

订阅者1 开始处理 事件0

订阅者2 开始处理 事件0

订阅者3 开始处理 事件0

订阅者1 开始处理 事件1

订阅者2 开始处理 事件1

订阅者3 开始处理 事件1

订阅者1 开始处理 事件2

订阅者2 开始处理 事件2

订阅者3 开始处理 事件2

订阅者1 开始处理 事件3

订阅者2 开始处理 事件3

订阅者3 开始处理 事件3

订阅者1 开始处理 事件4

订阅者2 开始处理 事件4

订阅者3 开始处理 事件4

订阅者1 开始处理 事件5

订阅者2 开始处理 事件5

订阅者3 开始处理 事件5

老子要的是多线程的效果啊，挨个都处理一遍是作甚。

这东西我搞的不是很明白，总之不适合我多线程爬取鼠绘漫画资源的需求。

## **collection**

### Lists

Lists.transform()

用于将一个list转换成另一个list，初见时惊为天人，代码还能这么写。不过这两个月下来，倒也不那么盲目了。

这东西的源码我还没见过，但有一点还是比较肯定的，它不是立马做转换的。只有切切实实要用到转换后的list中的对象时，它才会帮你把这个对象转出来。

而且，这个转换不可控，因为有时候我们可能想在转换过程中过滤一部分元素，用transform方法的话，顶多return一个null。这样后续的迭代会显得比较麻烦，不如一开始就自己写for循环。