使用JavaServlets2.4来执行过滤 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/242/2021_2022__E4_BD_BF_E7_94_A8Java_c97_242697.htm Servlet API 很久以前就已成为企业应用开发的基石，而 Servlet 过滤器则是对 J2EE 家族的相对较新的补充。在 J2EE 探索者 系列文章的最后一篇中，作者 Kyle Gabhart 将向您介绍 Servlet 过滤器体系结构，定义过滤器的许多应用，并指导您完成典型过滤器实现的三个步骤。他还会透露 bean 的一些激动人心的变化，预计刚发布的 Java Servlet 2.4 规范会引入这些变化。 Servlet 过滤器是可插入的 Web 组件，它允许我们实现 Web 应用程序中的预处理和后期处理逻辑。过滤器支持 servlet 和 JSP 页面的基本请求处理功能，比如日志记录、性能、安全、会话处理、XSLT 转换，等等。 过滤器最初是随 Java Servlet 2.3 规范发布的，最近定稿的 2.4 规范对它进行了重大升级。在这 J2EE 探索者 系列文章的最后一篇中，我将向您介绍 Servlet 过滤器的基础知识 比如总体的体系结构设计、实现细节，以及在 J2EE Web 应用程序中的典型应用，还会涉及一些预计最新的 Servlet 规范将会提供的扩展功能。Servlet 过滤器是什么？ Servlet 过滤器是小型的 Web 组件，它们拦截请求和响应，以便查看、提取或以某种方式操作正在客户机和服务器之间交换的数据。过滤器是通常封装了一些功能的 Web 组件，这些功能虽然很重要，但是对于处理客户机请求或发送响应来说不是决定性的。典型的例子包括记录关于请求和响应的数据、处理安全协议、管理会话属性，等等。过滤器提供一种面向对象的模块化机制，用以将公共任务封装到可插入的组件中，这些组件通过一个

配置文件来声明，并动态地处理。 Servlet 过滤器中结合了许多元素，从而使得过滤器成为独特、强大和模块化的 Web 组件。也就是说，Servlet 过滤器是：声明式的：过滤器通过 Web 部署描述符（web.xml）中的 XML 标签来声明。这样允许添加和删除过滤器，而无需改动任何应用程序代码或 JSP 页面。动态的：过滤器在运行时由 Servlet 容器调用来拦截和处理请求和响应。灵活的：过滤器在 Web 处理环境中的应用很广泛，涵盖诸如日志记录和安全等许多最公共的辅助任务。过滤器还是灵活的，因为它们可用于对来自客户机的直接调用执行预处理和后期处理，以及处理在防火墙之后的 Web 组件之间调度的请求。最后，可以将过滤器链接起来以提供必需的功能。模块化的：通过把应用程序处理逻辑封装到单个类文件中，过滤器从而定义了可容易地从请求/响应链中添加或删除的模块化单元。可移植的：与 Java 平台的其他许多方面一样，Servlet 过滤器是跨平台和跨容器可移植的，从而进一步支持了 Servler 过滤器的模块化和可重用本质。可重用的：归功于过滤器实现类的模块化设计，以及声明式的过滤器配置方式，过滤器可以容易地跨越不同的项目和应用程序使用。透明的：在请求/响应链中包括过滤器，这种设计是为了补充（而不是以任何方式替代）servlet 或 JSP 页面提供的核心处理。因而，过滤器可以根据需要添加或删除，而不会破坏 servlet 或 JSP 页面。 所以 Servlet 过滤器是通过一个配置文件来灵活声明的模块化可重用组件。过滤器动态地处理传入的请求和传出的响应，并且无需修改应用程序代码就可以透明地添加或删除它们。最后，过滤器独立于任何平台或者 Servlet 容器，从而允许将它们容易地部署到任何相容的 J2EE

环境中。 在接下来的几小节中，我们将进一步考察 Servlet 过滤器机制的总体设计，以及实现、配置和部署过滤器所涉及的步骤。我们还将探讨 Servlet 过滤器的一些实际应用，最后简要考察一下模型-视图-控制器（MVC）体系结构中包含的 Servlet 过滤器，从而结束本文的讨论。Servlet 过滤器体系结构正如其名称所暗示的，Servlet 过滤器 用于拦截传入的请求和/或传出的响应，并监视、修改或以某种方式处理正在通过的数据流。过滤器是自包含、模块化的组件，可以将它们添加到请求/响应链中，或者在无需影响应用程序中其他 Web 组件的情况下删除它们。过滤器仅只是改动请求和响应的运行时处理，因而不应该将它们直接嵌入 Web 应用程序框架，除非是通过 Servlet API 中良好定义的标准接口来实现。 Web 资源可以配置为没有过滤器与之关联（这是默认情况）、与单个过滤器关联（这是典型情况），甚至是与一个过滤器链相关联。那么过滤器究竟做什么呢？ 像 servlet 一样，它接受请求并响应对象。然后过滤器会检查请求对象，并决定将该请求转发给链中的下一个组件，或者中止该请求并直接向客户机发回一个响应。如果请求被转发了，它将被传递给链中的下一个资源（另一个过滤器、servlet 或 JSP 页面）。在这个请求设法通过过滤器链并被服务器处理之后，一个响应将以相反的顺序通过该链发送回去。这样就给每个过滤器都提供了根据需要处理响应对象的机会。 当过滤器在 Servlet 2.3 规范中首次引入时，它们只能过滤 Web 客户机和客户机所访问的指定 Web 资源之间的内容。如果该资源然后将请求调度给其他 Web 资源，那就不能向幕后委托的任何请求应用过滤器。2.4 规范消除了这个限制。Servlet 过滤器现在可以应用于 J2EE

Web 环境中存在请求和响应对象的任何地方。因此，Servlet 过滤器可以应用在客户机和 servlet 之间、servlet 和 servlet 或 JSP 页面之间，以及所包括的每个 JSP 页面之间。这才是我所称的强大能力和灵活性！实现一个 Servlet 过滤器 他们说“ 好事多磨 ”。我不知道“ 他们 ”指的是谁，或者这句古老的谚语究竟有多真实，但是实现一个 Servlet 过滤器的确要经历三个步骤。首先要编写过滤器实现类的程序，然后要把该过滤器添加到 Web 应用程序中（通过在 Web 部署描述符 /web.xml 中声明它），最后要把过滤器与应用程序一起打包并部署它。我们将详细研究这其中的每个步骤。1. 编写实现类的程序 过滤器 API 包含 3 个简单的接口（又是数字 3！），它们整洁地嵌套在 javax.servlet 包中。那 3 个接口分别是 Filter、FilterChain 和 FilterConfig。从编程的角度看，过滤器类将实现 Filter 接口，然后使用这个过滤器类中的 FilterChain 和 FilterConfig 接口。该过滤器类的一个引用将传递给 FilterChain 对象，以允许过滤器把控制权传递给链中的下一个资源。FilterConfig 对象将由容器提供给过滤器，以允许访问该过滤器的初始化数据。 为了与我们的三步模式保持一致，过滤器必须运用三个方法，以便完全实现 Filter 接口： init()：这个方法在容器实例化过滤器时被调用，它主要设计用于使过滤器为处理做准备。该方法接受一个 FilterConfig 类型的对象作为输入。 doFilter()：与 servlet 拥有一个 service() 方法（这个方法又调用 doPost() 或者 doGet()）来处理请求一样，过滤器拥有单个用于处理请求和响应的方法doFilter()。这个方法接受三个输入参数：一个 ServletRequest、response 和一个 FilterChain 对象。 destroy()：正如您想像的那样，这个方法执

行任何清理操作，这些操作可能需要在自动垃圾收集之前进行。