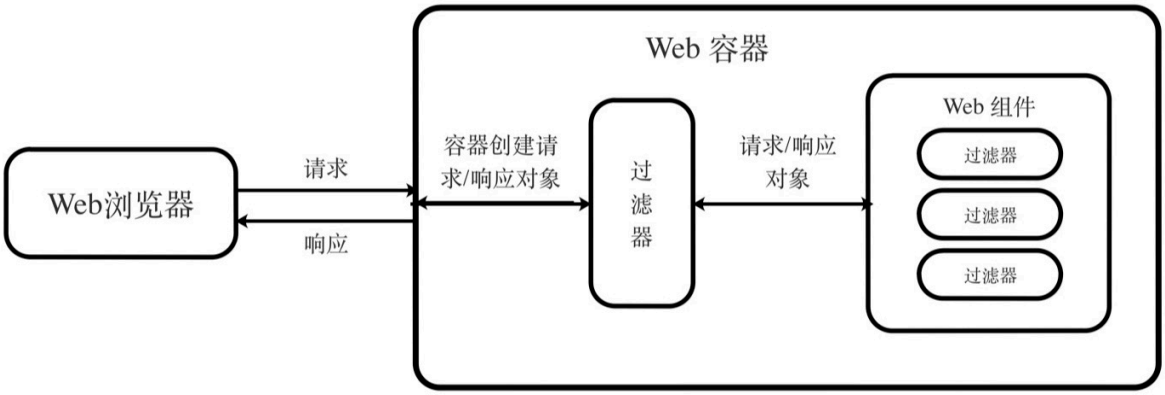
1. Filter 过滤器

1.1. 概述

对访问的请求和相应进行拦截，实现特殊的功能

- 例如验证用户访问权限、记录用户操作

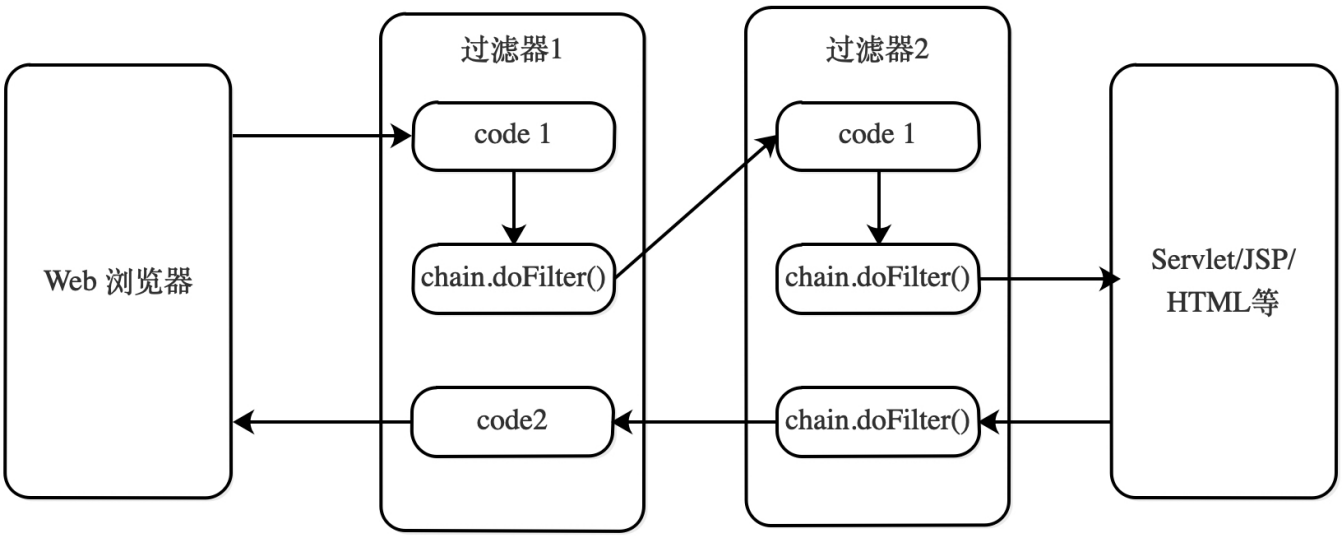
运行原理



- 请求到达指定网页之前，借助过滤器改变这些请求，称为预处理
- 执行结果到达用户之前经过过滤器修改相应输出的内容，称为后处理
- 过滤器链
- 过滤器运行步骤
    - Web 容器判断接收的请求资源是否有与之匹配的过滤器，如果有，容器将请求交给相应过滤器进行处理;
    - 在过滤器预处理过程中,可以改变请求的内容,或者重新设置请求的报头信息,然后将请求发给目标资源;
    - 目标资源对请求进行处理后作出响应;
    - 容器将响应转发回过滤器;
    - 在过滤器后处理过程中，可以根据需求对响应的内容进行修改;
    - Web 容器将响应发送回客户端。

1.2. 过滤器链

- 在一个 Web 应用中可以部署多个过滤器，形成一个过滤器链
- 在客户端响应过程，并不需要经过所有的过滤链，而是根据过滤链中的过滤条件匹配需要过滤的资源



1.3. 过滤器的生命周期

- 加载和实例化
  - Web 容器启动时,会根据 `@WebFilter` 属性 `filterName` 所定义的类名的大小写拼写顺序,或者 `web.xml` 中声明的 Filter 顺序依次实例化 Filter。
- 初始化
  - Web 容器调用 `init(FilterConfig config)` 方法来初始化过滤器。容器在调用该方法时，向过滤器传递 `FilterConfig` 对象。实例化和初始化的操作只会在容器启动时执行,并且只会执行一次。
- `doFliter()` 方法的执行
  - 当客户端请求目标资源的时候, 容器会筛选出符合过滤器映射条件的 Filter, 并按照 `@WebFilter` 属性 filterName 所定义的类名的大小写拼写顺序,或者 `web.xml` 中声明的 `filter-mapping` 的顺序依次调用这些过滤器的 `doFilter()` 方法。
  - 在这个链式调用过程中,可以调用 `FilterChain` 对象的 `doFilter(ServletRequest,ServletResponse)` 方法将请求传给下一个过滤器(或目标资源),也可以直接向客户端返回响应信息,或者利用请求转发或重定向将请求转向到其它资源。需要注意的是,这个方法的请求和响应参数的类型是 `ServletRequest` 和 `ServletResponse` ,也就是说,过滤器的使用并不依赖于具体的协议。
- 销毁
  - Web 容器调用 `destroy()` 方法指示过滤器的生命周期结束。在这个方法中，可以释放过滤器使用的资源。

1.4. FilterConfig 接口

- 容器将其实例作为参数传入过滤器对象的初始化方法 `init()` 中，来获取过滤器的初始化参数和 `Servlet` 的相关信息
- `getFilterName()`：获取配置信息中指定的过滤器的名字
- `getInitParamter(String name)`：获取配置信息中名为 `name` 的过滤器初始化参数
- `getInitParameterNames()`：获取过滤器所有初始化参数的枚举集合
- `getServletContext()`：获取 Servlet 上下文对象

1.5. FilterChain 接口

- 由容器实现，将其实例作为参数传入 `doFilter()` 方法，调用过滤器链中的下一个过滤器，如果是最后一个资源就调用目标资源
- `doFilter(ServletRequest req, ServletResponse resp)`

1.6. 过滤器声明配置

- `@WebFliter` 注解 / `Web.xml`
- 匹配
    - 路径匹配：`urlPattern/value` 对指定的URL 匹配模式进行指定地址拦截
        - `/*`：本应用目录下所有
    - 地址匹配：`*.jsp` 扩展名
- `dispatcherTypes`
    - `REQUEST`：直接对网页请求时
    - `FORWARD`：`RequestDispatcher` 对象的 `forward()` 方法调用
    - `INCLUDE`：`RequestDispatcher` 对象的 `include()` 方法调用
    - `ERROR`：某个页面使用 page 指令指定了 error 属性，出现异常时调用
    - `ASYNC`：指异步处理调用此页面

过滤器应用

- 做统一的认证处理;
- 对用户的请求进行检查和更精确的记录;
- 监视或对用户所传递的参数做前置处理,例如:防止数据注入攻击;
- 改变图像文件的格式;
- 对请求和响应进行编码;
- 对响应做压缩处理;
- 对XML的输出使用XSLT来转换。

2. 监听器

对Web 应用特殊事件设置响应，事件发生时，回调监听器事件实现功能。

2.1. 与Servlet 上下文相关的接口

- `ServletContextListener`
  - 监听 `ServletContext` 对象的创建和销毁
  - 每个 Web 应用对应一个 `ServletContext`，Web 容器启动时创建，关闭时销毁
  - Web 容器创建销毁时产生一个 `ServletContextEvent` 事件
  - 当 Web 应用程序中声明了一个实现 `ServletContextListener` 接口的事件监听器后，Web 容器在创建或销毁此对象时就会产生一个 `ServletContextEvent` 事件对象，然后再执行监听器中的相应事件处理方法，并将 `ServletContextEvent` 事件对象传递给这些方法。
  - 方法
    - `contextInitialized( ServletContextEvent sce )`
    - `contextDestroyed( ServletContextEvent sce )`
  - `ServletContextEvent` 为一个事件类，用于通知 Web 应用程序上下文的改变
- `ServletContextAttributeListener`
  - 监听 `ServletContext` application 范围内属性的改变
  - Web 容器在 `ServletContext` 应用域属性发生改变时产生一个 `ServletContextAttributeEvent` 对象
    - `attributeAdd( ServletContextAttriibuteEvent event )`
    - `attributeRemove(ServletContextAttriibuteEvent event )`
    - `attributeReplaced(ServletContextAttriibuteEvent event )`
  - `ServletContextAttributeEvent` 方法
    - `getName()`：返回属性名
    - `getValue()`：返回属性值

2.2. 与会话相关监听器接口

- `HttpSessionListener`：监听会话的创建和销毁
  - 每个浏览器与服务器的会话状态对于一个 `HttpSession` 对象，浏览器与服务器开始会话时创建，结束时销毁
  - `sessionCreated(HttpSessionEvent se)`，创建 `session` 对象时候
  - `sessionDestroyed(HttpSessionEvent se)`，销毁 `session` 对象时候
- `HttpSessionAttributeListener`
- 监听会话域属性的改变
  - `attributeAdd( HttpSessionBindingEvent event )`
  - `attributeRemove(HttpSessionBindingEvent event)`
- `attributeReplaced(HttpSessionBindingEvent event )`

2.3. 与请求相关的监听器接口

- `ServletRequestListener`：监听请求的产生和结束
  - 浏览器的每次访问请求分别对应一个 `ServletRequest` 对象，访问请求开始创建，结束销毁
  - `requestInitialized(ServletRequestEvent event)`

    - `requestDestroyed(ServletRequestEvent event)`
    - `ServletRequestEvent`
        - `getServletRequest()`
- `ServletRequestAttributeListener`：监听 ServletRequest 范围内属性的变化
    - `attributeAdd(ServletRequestAttributeEvent event )`
    - `attributeRemove(ServletRequestAttributeEvent event )`
    - `attributeReplaced(ServletRequestAttributeEvent event  )`
    - `ServletRequestEvent`
        - `getName()`
        - `getValue()`