1. Filter
2. 概述

Filter过滤器，当访问服务器的资源时 ，过滤器可以将请求拦截下来，完成一些特殊的功能。

1. 作用

过滤器一般完成通用的操作，如：登录验证，统一编码处理，敏感字符过滤

1. 过滤器的使用
2. 定义一个类，实现Filter接口
3. 重写接口中的方法
4. 通过注解@WebFilter配置拦截路径，即 访问该路径的资源之前，都会执行该过滤器。

或者通过web.xml配置拦截路径：

<filter>

<filter-name>demo1</filter-name>

<filter-class>cn.pro.filter.FilterDemo1</filter-class>

</filter>

<filter-mapping>

<filter-name>demo1</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

1. 在doFilter方法中要对请求进行放行操作：filterChain.doFilter(servletRequest, ServletResponse)
2. 过滤器的执行流程
3. 浏览器在请求服务器的资源时，要经过过滤器，过滤器对request对象请求消息进行增强处理。执行放行代码段之前的代码。
4. 过滤器处理后，将request请求对象进行放行。执行放行代码段。
5. 服务器将response发往浏览器的过程中，经过过滤器，过滤器对response对象进行增强处理。执行的是放行下面的代码段。

System.*out*.println("filterDemo1执行了");  
//放行  
chain.doFilter(req, resp);  
  
System.*out*.println("filterDemo2回来了");

1. 过滤器的生命周期方法
2. init方法

在服务器启动后，会创建Filter对象，然后调用init方法，只执行一次，用于加载资源。

1. doFilter方法

每一次请求资源被拦截时，会执行doFilter方法

1. destroy方法

在服务器关闭后，Filter对象被销毁，如果服务器是正常关闭，则会执行destroy方法，用于释放资源。

1. 过滤器的配置
2. 过滤器的拦截路径配置
3. 具体资源路径

/index.jsp，只有当访问index.jsp资源时，过滤器才会被执行

1. 拦截目录

/user/\* ，只有访问/user下的所有资源，过滤器才会被执行

1. 后缀名拦截

\*.jsp，访问所有后缀名为jsp资源时，过滤器都会被执行

1. 拦截所有资源

/\* ，访问所有资源时，过滤器都会被执行

2. 拦截方式的配置

可以通过注解配置或者web.xml进行配置。

注解配置中，设置dispatcherTypes属性，该属性可选值：

1. REQUEST

默认值。表示浏览器直接请求资源

@WebFilter(“/\*”, dispatcherTypes=DispatcherType.REQUEST)

1. FORWARD

表示转发访问资源。

1. INCLUDE

包含访问资源。

1. ERROR

表示错误跳转资源。

1. ASYNC

异步访问资源。

如果有多个限制条件，则用大括号括起来，如：

dispatcherTypes = (DispatcherType.REQUEST, DispatcherType.ERROR)

1. 过滤器链

当有多个过滤器链存在时，过滤器的执行顺序是怎样的？

如；有两个过滤器1，过滤器2，则执行顺序是：

过滤器1 -> 过滤器2 -> 资源执行 -> 过滤器2 -> 过滤器1

1. 过滤器执行顺序问题
2. 注解配置

在注解中配置过滤器时，按照类名的字符串比较规则进行比较，值小的先执行

1. web.xml配置

在web,xml中配置过滤器时，哪个过滤器的<filter-mapping>标签定义在上面，谁就先执行。

1. 过滤器实例
2. 登录验证
3. 需求

过滤器验证是否登录了，如果登录了，则直接放行；如果没有登录，则跳转到登录界面，输出提示。

1. 思路
2. 由于我们访问资源时可能直接访问登陆相关的资源，那么此次请求就不应该被阻拦，应该直接放行。因此首先判断是否是登录相关的资源，如果是，则直接放行；如果不是，则判断是否登录。
3. 要判断当前用户是否已经登录，就要通过Session中是否有User来判断，如果有，则放行；如果没有，则跳转到登录界面。
4. 代理模式
5. 概述

代理对象通过代理真实对象，以达到增强真实对象功能的目的。

1. 实现方式
2. 静态代理

有一个类文件描述代理模式

1. 动态代理

在内存中形成代理类，没有一个类文件代表代理对象。

实现步骤：

1. 代理对象和真实对象实现相同的接口
2. 代理对象 = Proxy.newInstance();
3. 使用代理对象调用方法
4. 增强方法
5. 动态代理的使用
6. 创建真实对象
7. 创建动态代理对象增强真实对象。使用Proxy.newProxyInstance()方法创建动态代理对象，传入的参数为：

类加载器：真是对象.getClass().getClassLoader()

接口数组：真实对象.getClass().getInterfaces()

处理器对象：new InvocationHandler(){}

处理器为一个匿名对象，其中存放着代理增强的逻辑代码在invoke()方法中，代理对象调用的所有方法都会触发invoke()方法的执行：

public Object invoke(Object proxy, Method method, Object[] args){}

proxy:代理对象

method:代理对象调用的方法，被封装为对象

args:代理对象调用方法时，传递的真实参数

1. 增强的方式
2. 增强参数

通过args参数数组可以获取参数，因而可以修改参数。并重新通过真实方法将该参数传入调用invoke()方法。

double money = (double)args[0];

money = money \* 0.85;

Object obj = method.invoke(Lenovo,money);

return obj;

1. 增强返回值

可以通过修改返回值来增强。

return obj + “\_鼠标垫”;

1. 增强方法体

在方法体中添加操作，以增强真实对象。

1. 监听器
2. 概述

监听器，顾名思义用于监听事件的发生。

1. 监听机制
2. 事件

某个事件，如按下键盘

1. 事件源

事件发生的地方

1. 监听器

用于监听事件发生的对象。

1. 注册监听

将事件，事件源，监听器绑定在一起，当事件源上发生某个事件后，执行监听器的代码。

1. ServletContextListener监听器

监听ServletContext对象的创建和销毁。

1. 方法

void contextDestroyed(ServletContextEvent sce)

ServletContext对象被销毁前会调用该方法

void contextInitialized(ServletContextEvent sce)

ServletContext对象创建后会调用该方法

1. 使用步骤
2. 定义一个类，事项ServletContextListener接口
3. 重写接口中的方法
4. 配置文件

通过web.xml或者注解都可以进行配置：

web.xml：

<listener>

<listener-class>全类名</listener-class>

</listener>

注解：

@WebListener