springboot下使用过滤器、拦截器和监听器

过滤器

简介

Filter也称之为过滤器,过滤器是对数据进行过滤，预处理。开发人员可以对客户端提交的数据进行过滤处理，比如敏感词，也可以对服务端返回的数据进行处理。还有就是可以验证用户的登录情况，权限验证，对静态资源进行访问控制，没有登录或者是没有权限时是不能让用户直接访问这些资源的。类似的过滤器还有很多的功能，比如说编码，压缩服务端给客户端返回的各种数据，等等。

运作原理

Java为我们提供了一个**Filter**接口，我们只需要实现这个接口就能实现自定义过滤器，然后添加一些必要的配置让过滤器生效。过滤器只能初始化一次，并且过滤器只会在项目停止或者是重新部署的时候才销毁。我们可以实现的这个Filter接口，里面最重要的是一个**doFilter**方法，当我们编写好Filter，并配置好对那个URL资源进行拦截时，每一次请求这个资源之前就会调用这个doFilter方法。并且在这个doFilter方法里面也有着一个**FilterChain**的对象参数,这个对象里面也有一个doFilter方法，是否调用这个方法决定了这个过滤器是否能调用后面的资源或者是执行后面的过滤器。也就是相当于目标资源。所以在过滤器里面可以进行一些什么操作呢？可以在调用目标资源之前，进行权限等的处理；判断是否调用目标资源；也可以在调用目标资源之后进行一些响应消息进行处理。

过滤器配置的两种方法

注解配置

```
@WebFilter(urlPatterns = "/*", filterName = "filterUser")
public class FilterUser implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("用户过滤器开始初始化");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        System.out.println("用户过滤器开始工作");
        HttpServletRequest httpRequest =
(HttpServletRequest)servletRequest;
```

```
        HttpServletResponse httpResponse =
(HttpServletResponse)servletResponse ;
        httpResponse.setHeader("admin","admin");
        filterChain.doFilter(httpRequest,httpResponse);
    }

    @Override
    public void destroy() {
        System.out.println("用户过滤器销毁...");
    }
}
```

代码说明：

1. @**WebFilter**注解，**filterName**属性表示filter的名称，urlPatter表示要拦截的URL资源，可以是一个或者多个。
2. **@Order(1)**表示如果有多个拦截器的话就是设置这个拦截器的运行等级，数字越小，越先执行
3. **init()**方法只会执行一次，初始化过滤器
4. **doFilter()**核心方法，配置过滤器的逻辑代码
5. **destroy()**只会在项目停止或者是项目重新部署的时候才会执行

配置完上面的之后我们还需要在启动类加上一个扫描包的注解，开启包扫描。**@ServletComponentScan("com.*.filter")**，当然你也可以不用写包的具体地址，不传参数，但是建议是传参数，并且这个采参数也可以传多个的。

代码注册

```
public class FilterResource implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("资源过滤器开始初始化...");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest)
servletRequest;
        System.out.println("资源过滤器开始工
作:"+httpServletRequest.getRequestURL());
        filterChain.doFilter(servletRequest,servletResponse);
    }

    @Override
    public void destroy() {
```

```
        System.out.println("资源过滤器开始销毁...");
    }
}

@Configuration
public class FilterConfig {
    @Bean
    // spring boot 会按照order值的大小，从小到大的顺序来依次过滤
    @Order(2)
    public FilterRegistrationBean<FilterResource> configFilter() {
        FilterRegistrationBean<FilterResource> filterRegistrationResource
= new FilterRegistrationBean<>();
        filterRegistrationResource.setFilter(new FilterResource());
        filterRegistrationResource.addUrlPatterns("/*");
        filterRegistrationResource.setName("resourceFilter");
        return filterRegistrationResource;
    }
}
```

拦截器Interceptor

> 拦截器作用类似于过滤器，都可以对一个请求进行拦截处理

常见应用场景

1、日志记录，可以记录请求信息的日志，以便进行信息监控、信息统计等。
2、权限检查：如登陆检测，进入处理器检测是否登陆，如果没有直接返回到登陆页面。
3、性能监控：典型的是慢日志。

拦截器配置

```
public class InterceptorResource implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
        String requestUtl = request.getRequestURI();
        System.out.println("前置拦截器拦截资源，请求URL：" + requestUtl);
        return true;
    }

    @Override
```

```
    public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler, ModelAndView modelAndView)
{
        System.out.println("拦截资源拦截器postHandle之后执行");

    }

    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {
        System.out.println("拦截资源拦截器所有执行之后执行");
    }
}

public class InterceptorUser implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
        String requestUtl = request.getRequestURI();
        System.out.println("前置拦截器拦截用户，请求URL：" + requestUtl);
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler, ModelAndView modelAndView)
{
        System.out.println("拦截用户拦截器postHandle之后执行");

    }

    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {
        System.out.println("拦截用户拦截器所有执行之后执行");
    }
}


@Configuration
public class InterceptorConfig implements WebMvcConfigurer  {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
```

```
        // 注册第一个拦截器
        InterceptorRegistration registrationResource =
registry.addInterceptor(new InterceptorResource());
        // 配置拦截器路径
        registrationResource.addPathPatterns("/**");
        // 配置不拦截的路径
        registrationResource.excludePathPatterns("/**.html");

        // 注册其他拦截器，执行顺序和配置顺序有关

        //注册第二个拦截器
        InterceptorRegistration registrationUser =
registry.addInterceptor(new InterceptorUser());
        registrationUser.addPathPatterns("/**");
        registrationUser.excludePathPatterns("/**.html");

    }
}
```

运行结果

```
资源过滤器开始初始化...
资源过滤器开始工作:http://127.0.0.1:8088/hello
前置拦截器拦截资源，请求URL：/hello
前置拦截器拦截用户，请求URL：/hello
拦截用户拦截器postHandle之后执行
拦截资源拦截器postHandle之后执行
拦截用户拦截器所有执行之后执行
拦截资源拦截器所有执行之后执行
```

拦截器和过滤器区别

- Filter是依赖于Servlet容器，属于Servlet规范的一部分，而拦截器则是独立存在的，可以在任何情况下使用。
- Filter的执行由Servlet容器回调完成，而拦截器通常通过动态代理的方式来执行。
- Filter的生命周期由Servlet容器管理，而拦截器则可以通过IoC容器来管理，因此可以通过注入等方式来获取其他Bean的实例，因此使用会更方便。

监听器

> Java的监听器，也是系统级别的监听。监听器随web应用的启动而启动。Java的监听器在c/s模式里面经常用到，它会对特定的事件产生产生一个处理。监听在很多模式下用到，比如说观察者模式，就是一个使用监听器来实现的，在比如统计网站的在线人数。

分类

按监听的对象划可划分为：

ServletContext对象监听器
HttpSession对象监听器
ServletRequest对象监听器

按监听的事件划分为：

对象自身的创建和销毁的监听器
对象中属性的创建和消除的监听器

session中的某个对象的状态变化的监听器

ServletContextListener

- 注解方式

```
@WebListener
public class ListenerServlet implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("ListenerServlet初始化..");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {

    }
}

@SpringBootApplication
@ServletComponentScan
public class BasicApplication {

    public static void main(String[] args) {
        SpringApplication.run(BasicApplication.class, args);
    }
```

```
}
```

- 代码方式

```
public class ListenerResource implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("资源监听器初始化。。。。。。。。。。。。");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("资源监听器销毁。。。。。。。。。。。。");
    }
}

@Configuration
public class ListenerConfig {

    @Bean
    public ServletListenerRegistrationBean<ListenerResource>
resourceListenerRegistrationBean() {
        ServletListenerRegistrationBean<ListenerResource> slrBean = new
ServletListenerRegistrationBean<>();
        slrBean.setListener(new ListenerResource());
        return slrBean;
    }

}
```

HttpSessionListener

```
@Component
public class ListenerUser implements HttpSessionListener {
    public Integer count = 0;
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("用户监听器初始化...");
        count++;
        se.getSession().getServletContext().setAttribute("count", count);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("用户监听器销毁...");
```

```
    }
}
```

ServletContextListener

> 同ServletContextListener