2-7 抵御即跨站脚本（XSS）攻击

一、XSS攻击的危害

XSS攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是JavaScript，但实际上也可以包括Java、 VBScript、ActiveX、 Flash 或者甚至是普通的HTML。攻击成功后，攻击者可能得到包括但不限于更高的权限（如执行一些操作）、私密网页内容、会话和cookie等各种内容。

例如用户在发帖或者注册的时候，在文本框中输入 `<script>alert('xss')</script>` ，这段代码如果不经过转义处理，而直接保存到数据库。将来视图层渲染HTML的时候，把这段代码输出到页面上，那么 `<script>` 标签的内容就会被执行。

通常情况下，我们登陆到某个网站。如果网站使用 `HttpSession` 保存登陆凭证，那么 `SessionId` 会以 `Cookie` 的形式保存在浏览器上。如果黑客在这个网页发帖的时候，填写的 `JavaScript` 代码是用来获取 `Cookie` 内容的，并且把 `Cookie` 内容通过Ajax发送给黑客自己的电脑。于是只要有人在这个网站上浏览黑客发的帖子，那么视图层渲染HTML页面，就会执行注入的XSS脚本，于是你的 `Cookie` 信息就泄露了。黑客在自己的电脑上构建出 `Cookie` ，就可以冒充已经登陆的用户。

即便很多网站使用了JWT，登陆凭证（ `Token令牌` ）是存储在浏览器上面的。所以用XSS脚本可以轻松的从Storage中提取出 `Token` ，黑客依然可以轻松的冒充已经登陆的用户。

所以避免XSS攻击最有效的办法就是对用户输入的数据进行转义，然后存储到数据库里面。等到视图层渲染HTML页面的时候。转义后的文字是不会被当做JavaScript执行的，这就可以抵御XSS攻击。

二、导入依赖库

因为 `Hutool` 工具包带有XSS转义的工具类，所以我们要导入 `Hutool` ，然后利用 `Servlet` 规范提供的请求包装类，定义数据转义功能。

```
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.4.0</version>
</dependency>
```

三、定义请求包装类

我们平时写Web项目遇到的 `HttpServletRequest` ，它其实是个接口。如果我们想要重新定义请求类，扩展这个接口是最不应该的。因为 `HttpServletRequest` 接口中抽象方法太多了，我们逐一实现起来太耗费时间。所以我们应该挑选一个简单一点的自定义请求类的方式。那就是继承 `HttpServletRequestWrapper` 父类。

JavaEE只是一个标准，具体的实现由各家应用服务器厂商来完成。比如说`Tomcat`在实现`Servlet`规范的时候，就自定义了`HttpServletRequest`接口的实现类。同时JavaEE规范还定义了`HttpServletRequestWrapper`，这个类是请求类的包装类，用上了装饰器模式。不得不说这里用到的设计模式真的非常棒，无论各家应用服务器厂商怎么去实现`HttpServletRequest`接口，用户想要自定义请求，只需要继承`HttpServletRequestWrapper`，对应覆盖某个方法即可，然后把请求传入请求包装类，装饰器模式就会替代请求对象中对应的某个方法。用户的代码和服务器厂商的代码完全解耦，我们不用关心`HttpServletRequest`接口是怎么实现的，借助于包装类我们可以随意修改请求中的方法。同学们，如此优雅的代码设计，有时间你真该认真学习设计模式。

```java
package com.example.emos.wx.config.xss;
import cn.hutool.core.util.StrUtil;
import cn.hutool.http.HtmlUtil;
import cn.hutool.json.JSONUtil;
import javax.servlet.ReadListener;
import javax.servlet.ServletInputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import java.io.*;
import java.nio.charset.Charset;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;
public class XssHttpServletRequestWrapper extends HttpServletRequestWrapper {
    public XssHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);
    }
    @Override
    public String getParameter(String name) {
        String value = super.getParameter(name);
        if (!StrUtil.hasEmpty(value)) {
            value = HtmlUtil.filter(value);
        }
        return value;
    }
    @Override
    public String[] getParameterValues(String name) {
        String[] values = super.getParameterValues(name);
        if (values != null) {
            for (int i = 0; i < values.length; i++) {
                String value = values[i];
                if (!StrUtil.hasEmpty(value)) {
                    value = HtmlUtil.filter(value);
                }
                values[i] = value;
            }
        }
        return values;
    }
    @Override
    public Map<String, String[]> getParameterMap() {
        Map<String, String[]> parameters = super.getParameterMap();
        Map<String, String[]> map = new LinkedHashMap<>();
        if (parameters != null) {
            for (String key : parameters.keySet()) {
                String[] values = parameters.get(key);
                for (int i = 0; i < values.length; i++) {
                    String value = values[i];
                    if (!StrUtil.hasEmpty(value)) {
                        value = HtmlUtil.filter(value);
                    }
                    values[i] = value;
                }
                map.put(key, values);
            }
```

```
        }
        return map;
    }
    @Override
    public String getHeader(String name) {
        String value = super.getHeader(name);
        if (!StrUtil.hasEmpty(value)) {
            value = HtmlUtil.filter(value);
        }
        return value;
    }
    @Override
    public ServletInputStream getInputStream() throws IOException {
        InputStream in = super.getInputStream();
        StringBuffer body = new StringBuffer();
        InputStreamReader reader = new InputStreamReader(in, Charset.forName("UTF-8"));
        BufferedReader buffer = new BufferedReader(reader);
        String line = buffer.readLine();
        while (line != null) {
            body.append(line);
            line = buffer.readLine();
        }
        buffer.close();
        reader.close();
        in.close();
        Map<String, Object> map = JSONUtil.parseObj(body.toString());
        Map<String, Object> resultMap = new HashMap(map.size());
        for (String key : map.keySet()) {
            Object val = map.get(key);
            if (map.get(key) instanceof String) {
                resultMap.put(key, HtmlUtil.filter(val.toString()));
            } else {
                resultMap.put(key, val);
            }
        }
        String str = JSONUtil.toJsonStr(resultMap);
        final ByteArrayInputStream bain = new ByteArrayInputStream(str.getBytes());
        return new ServletInputStream() {
            @Override
            public int read() throws IOException {
                return bain.read();
            }
            @Override
            public boolean isFinished() {
                return false;
            }
            @Override
            public boolean isReady() {
                return false;
            }
            @Override
            public void setReadListener(ReadListener listener) {
            }
        };
    }
```

```
111. }
```

四、创建过滤器，把所有请求对象传入包装类

为了让刚刚定义的包装类生效，我们还要在`com.example.emos.wx.config.xss`中创建`XssFilter`过滤器。过滤器拦截所有请求，然后把请求传入包装类，这样包装类就能覆盖所有请求的参数方法，用户从请求中获得数据，全都经过转义。

```
1. package com.example.emos.wx.config.xss;
2. import javax.servlet.*;
3. import javax.servlet.annotation.WebFilter;
4. import javax.servlet.http.HttpServletRequest;
5. import java.io.IOException;
6. @WebFilter(urlPatterns = "/*")
7. public class XssFilter implements Filter {
8.     public void init(FilterConfig config) throws ServletException {
9.     }
10.     public void doFilter(ServletRequest request, ServletResponse response, FilterChain
    chain)
11.             throws IOException, ServletException {
12.         XssHttpServletRequestWrapper xssRequest = new XssHttpServletRequestWrapper(
13.                 (HttpServletRequest) request);
14.         chain.doFilter(xssRequest, response);
15.     }
16.     @Override
17.     public void destroy() {
18.     }
19. }
```

五、给主类添加注解

给SpringBoot主类添加`@ServletComponentScan`注解。

六、测试拦截XSS脚本

1. 把`TestSayHelloForm`中的正则表达式验证给去掉，因为`name`字段只可以是中文，所以无法接收XSS脚本。

2. 在Swagger中，执行`sayHello()`方法，向name属性传入`<script>HelloWorld</script>`，然后观察返回的结果