java安全之java-web

> 前一段时间，系统学习了javaweb相关的知识，比较全面，而且是注重web开发方面的，但是对于java安全人员来说，不需要特别细的去学习java-web开发。所以才有了这一篇文章。
>
> 如果需要特别细的学习一下javaweb，推荐哔哩上的javaweb入门到实战

接下来正式开始我们的java安全之java-web

01.Java Web 基础

`Java EE`指的是Java平台企业版（`Java Platform Enterprise Edition`），之前称为`Java 2 Platform, Enterprise Edition`(`J2EE`)，2017 年的 9 月Oracle将`Java EE`捐赠给 Eclipse 基金会，由于Oracle持有Java商标原因，Eclipse基金于2018年3月将`Java EE`更名为Jakarta EE。

Java EE和Servlet版本

Java EE历史版本:

Java SE/JDK版本	Java EE版本	Servlet版本	发布时间
/	/	Servlet 1.0	(1997年6月)
JDK1.1	/	Servlet 2.0	/
/	/	Servlet 2.1	(1998年11月)
JDK1.2	J2EE 1.2	Servlet 2.2	(1999年12月12日)
JDK1.2	J2EE 1.3	Servlet 2.3	(2001年9月24日)
JDK1.3	J2EE 1.4	Servlet 2.4	(2003年11月11日)
JDK1.5	Java EE 5	Servlet 2.5	(2006年5月11日)
JDK1.6	Java EE 6	Servlet 3.0	(2009年12月10日)
/	Java EE 7	Servlet 3.1	(2013年5月28日)
/	Java EE 8	Servlet 4.0	(2017年8月31日)
/	Jakarta EE8	Servlet 4.0	(2019年8月26日)

由上表可知`Java EE`并不是`Java SE`的一部分(JDK不自带)，`Java EE`的版本也不完全是对应了JDK版本，我们通常最为关注的是`Java EE`对应的`Servlet`版本。不同的`Servlet`版本有着不一样的特性，`Servlet`容器(如`GlassFish/Tomcat/Jboss`)也会限制部署的`Servlet`版本。Java流行的`Spring MVC`(基于Servlet机制实现)、`Struts2`(基于Filter机制实现)等Web框架也是基于不同的`Java EE`版本封装了各自的框架。

Servlet 3.0 规范、Servlet 3.1 规范、Servlet 4.0 规范

Tomcat Servlet版本

Tomcat版本	Java EE 版本	Servlet 版本	JSP版本	发布时间
Tomcat 5.0.0 +	J2EE 1.4	Servlet 2.4	JSP 2.0	(2003年11月24日)
Tomcat 6.0.0 - Tomcat 6.0.44	Java EE 5	Servlet 2.5	JSP 2.1	(2006年5月11日 - 2007年9月11日)
Tomcat 7.0.0 - Tomcat 7.0.25	Java EE 6	Servlet 3.0	JSP 2.2	(2009年12月10日 - 2011年2月6日)
Tomcat 8.0.0 +	Java EE 7	Servlet 3.1	JSP 2.3	(2013年5月28日)
Tomcat 9.0.0 +	Java EE 8	Servlet 4.0	JSP 2.3	(2017年2月5日)
Tomcat 10.0.0 +	Jakarta EE8	Servlet 4.0	JSP 2.3	/

参考: Web Application Specifications

上面简单的介绍了一下javaweb的历史，下面会介绍javaweb中最重要的`servlet`

02. Servlet

`Servlet`是在`Java Web`容器中运行的`小程序`,通常我们用`Servlet`来处理一些较为复杂的服务器端的业务逻辑。`Servlet`是`Java EE`的核心,也是所有的MVC框架的实现的根本!

基于Web.xml配置

`Servlet3.0`之前的版本都需要在`web.xml`中配置`servlet标签`，`servlet标签`是由`servlet`和`servlet-mapping`标签组成的,两者之间通过在`servlet`和`servlet-mapping`标签中同样的`servlet-name`名称来实现关联的。

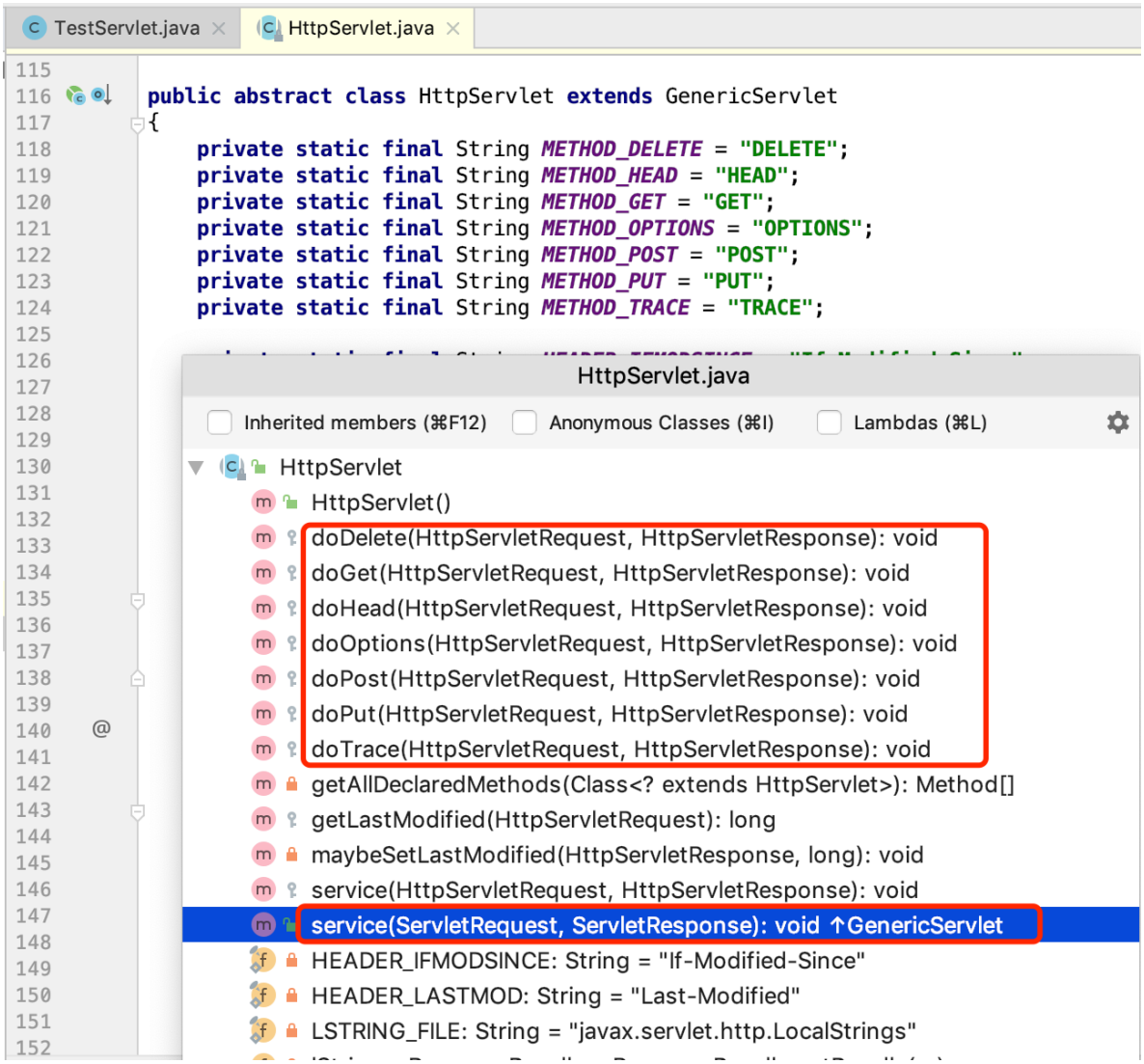
Servlet的定义

定义一个 Servlet 很简单，只需要继承`javax.servlet.http.HttpServlet`类并重写`doXXX`(如`doGet、doPost`)方法或者`service`方法就可以了，其中需要注意的是重写`HttpServlet`类的`service`方法可以获取到上述七种Http请求方法的请求。

javax.servlet.http.HttpServlet:

在写`Servlet`之前我们先了解下`HttpServlet`,`javax.servlet.http.HttpServlet`类继承于`javax.servlet.GenericServlet`，而`GenericServlet`又实现了`javax.servlet.Servlet`和`javax.servlet.ServletConfig`。`javax.servlet.Servlet`接口中只定义了`servlet`基础生命周期方法：`init(初始化)`、`getServletConfig(配置)`、`service(服务)`、`destroy(销毁)`,而`HttpServlet`不仅实现了`servlet`的生命周期并通过封装`service`方法抽象出了`doGet/doPost/doDelete/doHead/doPut/doOptions/doTrace`方法用于处理来自客户端的不一样的请求方式，我们的Servlet只需要重写其中的请求方法或者重写`service`方法即可实现`servlet`请求处理。

javax.servlet.http.HttpServlet类:

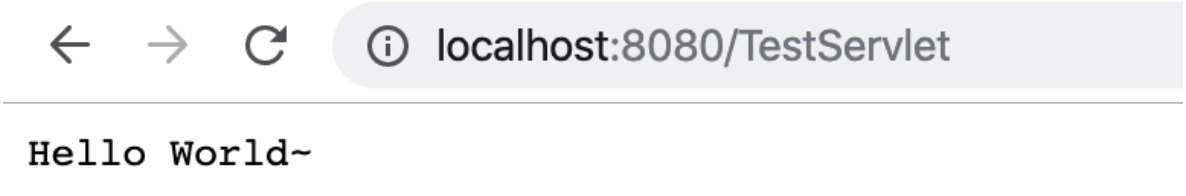


TestServlet示例代码:

```java
package com.anbai.sec.servlet;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Creator: yz
 * Date: 2019/12/14
 */
// 如果使用注解方式请取消@WebServlet注释并注释掉web.xml中TestServlet相关配置
//@WebServlet(name = "TestServlet", urlPatterns = {"/TestServlet"})
public class TestServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    doPost(request, response);
  }

  @Override
```

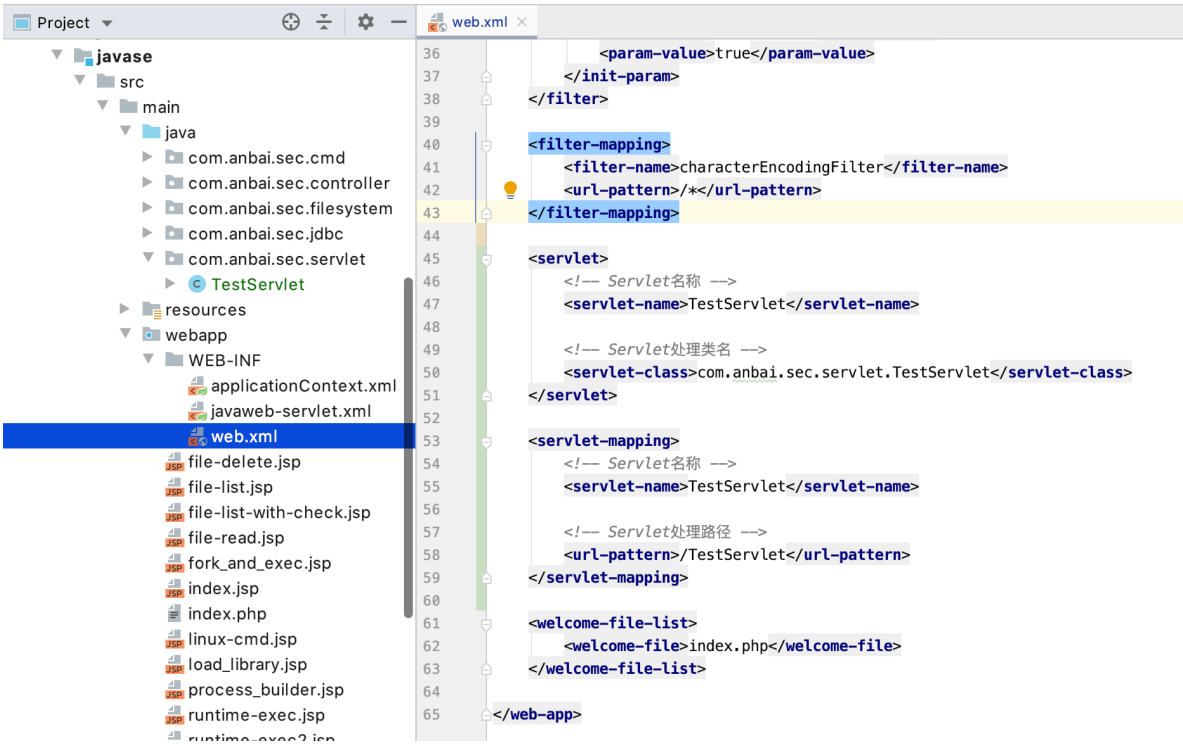
```
23    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException {
24        PrintWriter out = response.getWriter();
25        out.println("Hello World~");
26        out.flush();
27        out.close();
28    }
29
30 }
```

请求`TestServlet`示例:



Servlet Web.xml配置

定义好了Servlet类以后我们需要在`web.xml`中配置servlet标签才能生效。



Servlet 3.0 基于注解方式配置

基于注解的Servlet:

值得注意的是在 Servlet 3.0 之后(Tomcat7+)可以使用注解方式配置 Servlet 了,在任意的Java类添加`javax.servlet.annotation.WebServlet`注解即可。

基于注解的方式配置Servlet实质上是对基于`web.xml`方式配置的简化，极大的简化了Servlet的配置方式，但是也提升了对Servlet配置管理的难度，因为我们不得不去查找所有包含了`@WebServlet`注解的类来寻找Servlet的定义，而不再只是查看`web.xml`中的`servlet`标签配置。

```
/**
 * @author yz
 */
@WebServlet(name = "TestServlet", urlPatterns = {"/TestServlet"})
public class TestServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        doPost(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("Hello World~");
        out.flush();
        out.close();
    }

}
```

Servlet3.0开始支持使用注解方式配置Servlet

Servlet 3.0 特性

1. 新增动态注册`Servlet`、`Filter`和`Listener`的API(`addServlet`、`addFilter`、`addListener`)。
2. 新增`@WebServlet`、`@WebFilter`、`@WebInitParam`、`@WebListener`、`@MultipartConfig`注解。
3. 文件上传支持，`request.getParts()`。
4. `非阻塞 IO`，添加`异步 IO`。
5. 可插拔性(`web-fragment.xml`、`ServletContainerInitializer`)。

03.Filter

`javax.servlet.Filter`是`Servlet2.3`新增的一个特性,主要用于过滤URL请求，通过Filter我们可以实现URL请求资源权限验证、用户登陆检测等功能。

Filter是一个接口，实现一个Filter只需要重写`init`、`doFilter`、`destroy`方法即可，其中过滤逻辑都在`doFilter`方法中实现。

`Filter`的配置类似于`Servlet`，由和两组标签组成，如果Servlet版本大于3.0同样可以使用注解的方式配置Filter。

```java
/**
 * @author yz
 */
@WebFilter(filterName = "TestFilter", urlPatterns = {"/*"})
public class TestFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        String str = request.getParameter("password");

        if ("023".equals(str)) {
            chain.doFilter(request, response);
        } else {
            PrintWriter out = response.getWriter();
            out.println("Login error password error!");
            out.flush();
            out.close();
        }
    }

    @Override
    public void destroy() {

    }

}
```

Servlet3.0开始Filter支持使用注解方式配置

传递给下一个Filter链处理

输出密码错误信息到页面,没有执行chain.doFilter
后面的Servlet或Filter将不再执行

04. Filter和Servlet的总结

对于基于`Filter`和`Servlet`实现的简单架构项目，代码审计的重心集中于找出所有的`Filter`分析其过滤规则，找出是否有做全局的安全过滤、敏感的URL地址是否有做权限校验并尝试绕过`Filter`过滤。第二点则是找出所有的`Servlet`，分析`Servlet`的业务是否存在安全问题,如果存在安全问题是否可以利用？是否有权限访问？利用时是否被Filter过滤等问题，切勿看到`Servlet`、`JSP`中的漏洞点就妄下定论，不要忘了`Servlet`前面很有可能存在一个全局安全过滤的`Filter`。

`Filter`和`Servlet`都是`Java Web`提供的API，简单的总结了下有如下共同点。

1. `Filter`和`Servlet`都需要在`web.xml`或`注解`(`@WebFilter`、`@WebServlet`)中配置，而且配置方式是非常的相似的。
2. `Filter`和`Servlet`都可以处理来自Http请求的请求，两者都有`request`、`response`对象。
3. `Filter`和`Servlet`基础概念不一样，`Servlet`定义是容器端小程序，用于直接处理后端业务逻辑，而`Filter`的思想则是实现对Java Web请求资源的拦截过滤。
4. `Filter`和`Servlet`虽然概念上不太一样，但都可以处理Http请求，都可以用来实现MVC控制器(`Struts2`和`Spring`框架分别基于`Filter`和`Servlet`技术实现的)。
5. 一般来说`Filter`通常配置在`MVC`、`Servlet`和`JSP`请求前面，常用于后端权限控制、统一的Http请求参数过滤(`统一的XSS`、`SQL注入`、`Struts2命令执行`等攻击检测处理)处理，其核心主要体现在请求过滤上，而`Servlet`更多的是用来处理后端业务请求上。

05. JSP基础

`JSP`(`JavaServer Pages`)是与`PHP`、`ASP`、`ASP.NET`等类似的脚本语言，`JSP`是为了简化`Servlet`的处理流程而出现的替代品，早期的`Java EE`因为只能使用`Servlet`来处理客户端请求而显得非常的繁琐和不便，使用JSP可以快速的完成后端逻辑请求。

正因为在`JSP`中可以直接调用Java代码来实现后端逻辑的这一特性，黑客通常会编写带有恶意攻击的JSP文件(俗称`WebShell`)来实现对服务器资源的恶意请求和控制。

现代的MVC框架(如：`Spring MVC 5.x`)已经完全抛弃了`JSP`技术，采用了`模板引擎(如：Freemark)`或者`RESTful`的方式来实现与客户端的交互工作,或许某一天`JSP`技术也将会随着产品研发的迭代而彻底消失。

JSP 三大指令

1. `<%@ page ... %>` 定义网页依赖属性，比如脚本语言、error页面、缓存需求等等
2. `<%@ include ... %>` 包含其他文件（静态包含）
3. `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>` 引入标签库的定义

JSP 表达式(EL)

`EL表达式(Expression Language)`语言,常用于在jsp页面中获取请求中的值，如获取在Servlet中设置的`Attribute`:`${名称}`。使用EL表达式可以实现命令执行，我们将会在后续EL表达式章节中详细讲解。

JSP 标准标签库(JSTL)

JSP标准标签库（JSTL）是一个JSP标签集合，它封装了JSP应用的通用核心功能。

JSTL支持通用的、结构化的任务，比如迭代，条件判断，XML文档操作，国际化标签，SQL标签。 除了这些，它还提供了一个框架来使用集成JSTL的自定义标签。

JSP 九大对象

从本质上说 JSP 就是一个Servlet，JSP 引擎在调用 JSP 对应的 jspServlet 时，会传递或创建 9 个与 web 开发相关的对象供 jspServlet 使用。 JSP 技术的设计者为便于开发人员在编写 JSP 页面时获得这些 web 对象的引用，特意定义了 9 个相应的变量，开发人员在JSP页面中通过这些变量就可以快速获得这 9 大对象的引用。

如下:

变量名	类型	作用
pageContext	PageContext	当前页面共享数据，还可以获取其他8个内置对象
request	HttpServletRequest	客户端请求对象，包含了所有客户端请求信息
session	HttpSession	请求会话
application	ServletContext	全局对象，所有用户间共享数据
response	HttpServletResponse	响应对象，主要用于服务器端设置响应信息
page	Object	当前Servlet对象,`this`
out	JspWriter	输出对象，数据输出到页面上
config	ServletConfig	Servlet的配置对象
exception	Throwable	异常对象

06. JSP、Servlet之间的关系

JSP、JSPX 文件是可以直接被 Java 容器直接解析的动态脚本， jsp 和其他脚本语言无异，不但可以用于页面数据展示，也可以用来处理后端业务逻辑。

从本质上说 JSP 就是一个`Servlet`，因为 jsp 文件最终会被编译成 class 文件，而这个 class 文件实际上就是一个特殊的`Servlet`。

JSP文件会被编译成一个java类文件，如`index.jsp`在Tomcat中`Jasper`编译后会生成`index_jsp.java`和`index_jsp.class`两个文件。而`index_jsp.java`继承于`HttpJspBase`类，`HttpJspBase`是一个实现了`HttpJspPage`接口并继承了`HttpServlet`的标准的`Servlet`，`__jspService`方法其实是`HttpJspPage`接口方法，类似于`Servlet`中的`service`方法，这里的`__jspService`方法其实就是`HttpJspBase`的`service`方法调用。

07. Cookie 和 Session 对象

`Cookie`是最常用的Http会话跟踪机制，且所有`Servlet容器`都应该支持。当客户端不接受`Cookie`时，服务端可使用`URL重写`的方式作为会话跟踪方式。会话`ID`必须被编码为URL字符串中的一个路径参数，参数的名字必须是`jsessionid`。

浏览器和服务端创建会话(`Session`)后，服务端将生成一个唯一的会话ID(`sessionid`)用于标识用户身份，然后会将这个会话ID通过`Cookie`的形式返回给浏览器，浏览器接受到`Cookie`后会在每次请求后端服务的时候带上服务端设置`Cookie`值，服务端通过读取浏览器的`Cookie`信息就可以获取到用于标识用户身份的会话ID，从而实现会话跟踪和用户身份识别。

因为`Cookie`中存储了用户身份信息，并且还存储于浏览器端，攻击者可以使用`XSS`漏洞获取到`Cookie`信息并盗取用户身份就行一些恶意的操作。