

Day09

过滤器

案例

新建 maven 项目

修改 web.xml

新增 servlet

新增 Filter

添加Tomcat

配置 Tomcat

项目部署到 Tomcat

运行

课堂作业

案例分析

过滤器常用注解

过滤器细节

生命周期

doFilter 方法细节

filter执行了两遍

filter 返回

过滤器配置 (xml)

过滤器配置 (注解)

过滤器配置-初始化参数 (xml)

过滤器配置-初始化参数 (注解)

多个过滤器的执行顺序

过滤器的五种拦截行为

Filter 与 Servlet 的区别

课堂作业

Filter 案例

字符编码过滤器

过滤器

过滤器——Filter，它是 JavaWeb 三大组件之一。另外两个是 Servlet 和 Listener。

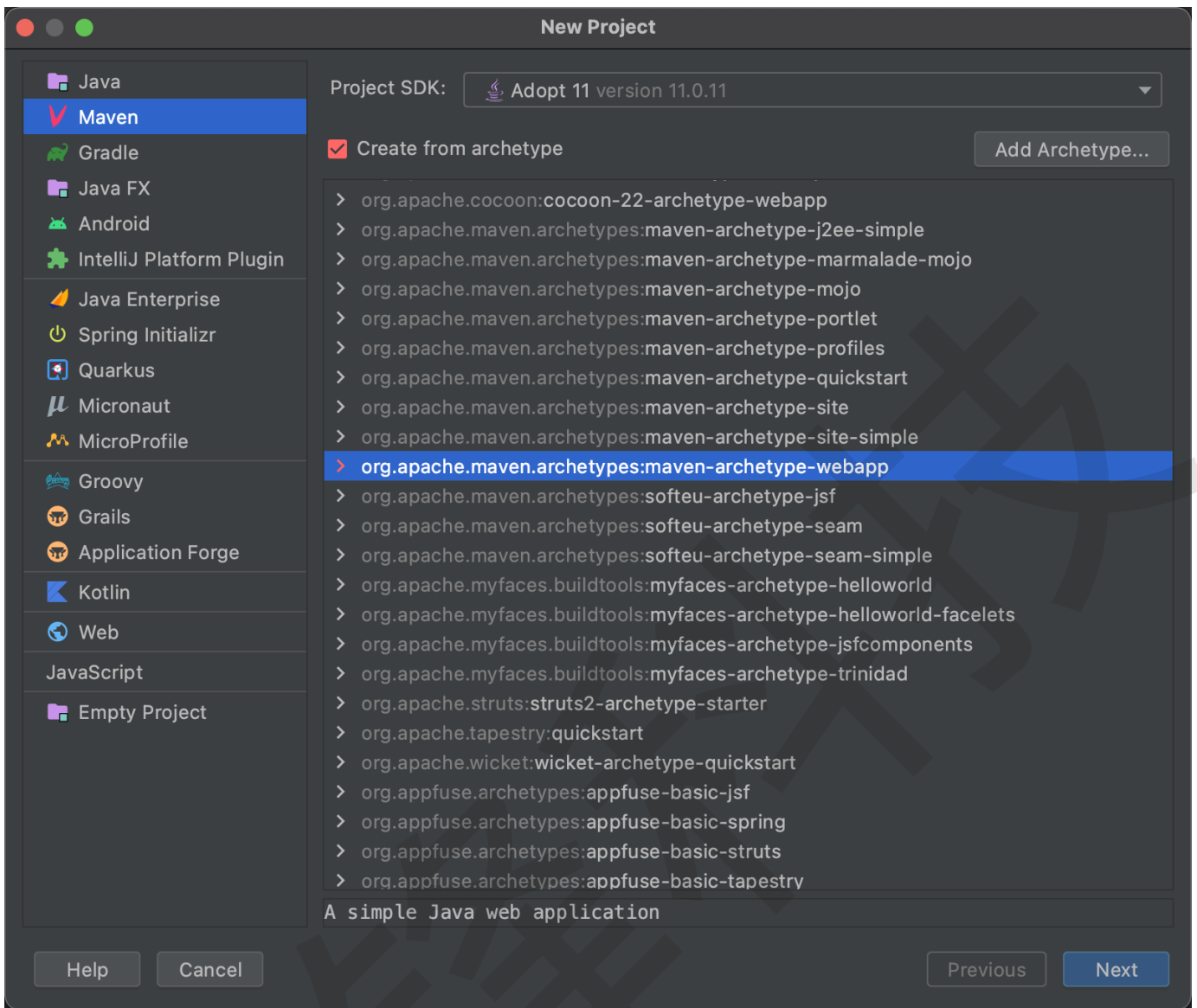
它是在 2000年发布的 Servlet 2.3 规范中加入的一个接口。是 Servlet 规范中非常实用的技术。

它可以对 web 应用中的所有资源进行拦截，并且在拦截之后进行一些特殊的操作。

常见应用场景：URL级别的权限控制；过滤敏感词汇；中文乱码问题等等。

案例

新建 maven 项目



修改 web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
6         version="3.1">
7
8 </web-app>
```

XML | 复制代码

新增 servlet

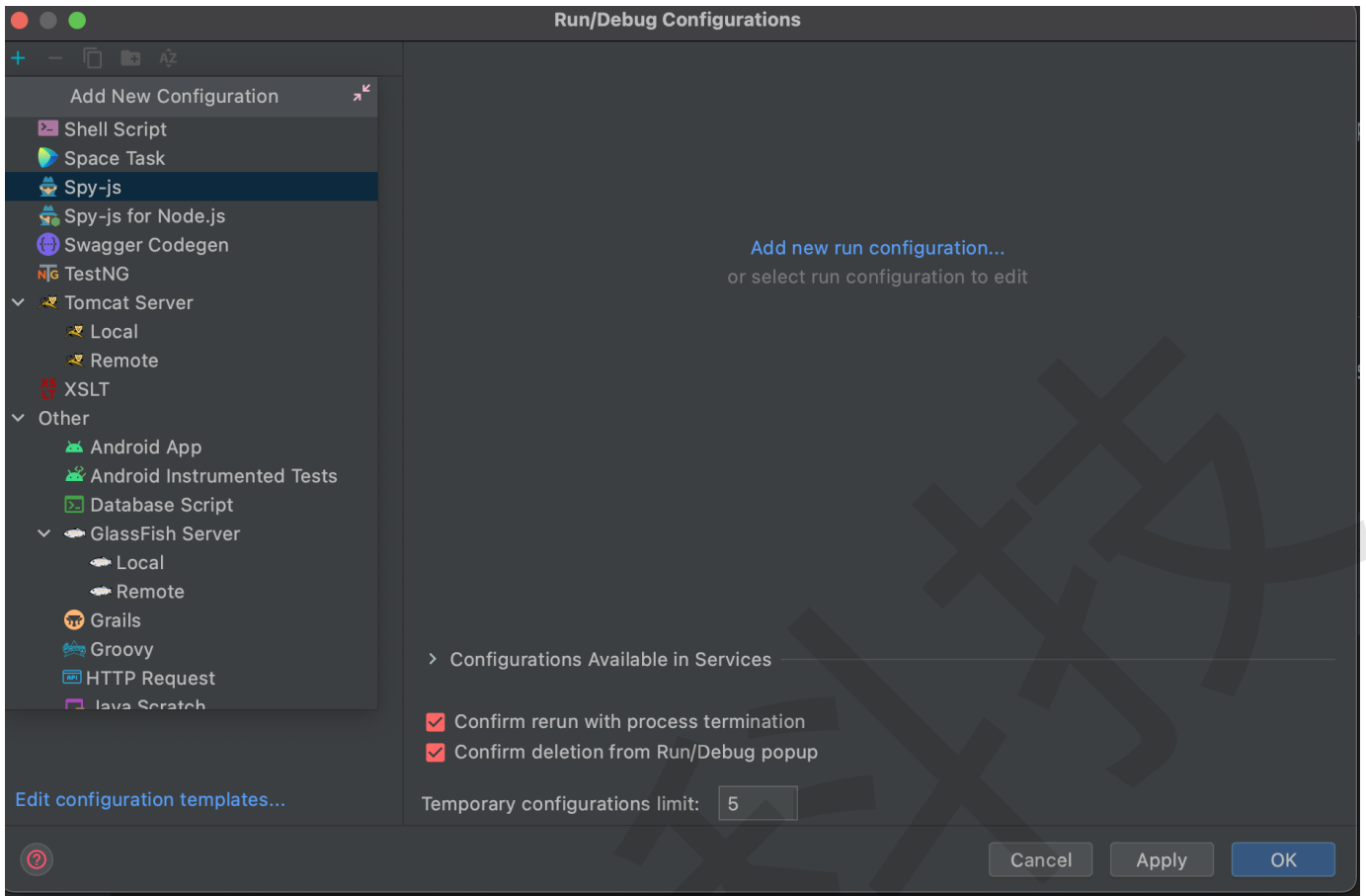
Java | 复制代码

```
1 package com.rushuni.servlet;
2
3 import jakarta.servlet.ServletException;
4 import jakarta.servlet.annotation.WebServlet;
5 import jakarta.servlet.http.HttpServlet;
6 import jakarta.servlet.http.HttpServletRequest;
7 import jakarta.servlet.http.HttpServletResponse;
8
9 import java.io.IOException;
10
11 /**
12  * @author rushuni
13  * @date 2021年07月22日 10:48 上午
14  */
15 @WebServlet("/demoServlet01")
16 public class Demo01Servlet extends HttpServlet {
17     @Override
18     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
19     throws ServletException, IOException {
20         System.out.println("demoServlet01 执行中...");
21         // resp.setContentType("text/html;charset=UTF-8");
22         resp.getWriter().write("<h1>demoServlet01 执行中...</h1>");
23     }
24
25     @Override
26     protected void doPost(HttpServletRequest req, HttpServletResponse resp)
27     throws ServletException, IOException {
28         doGet(req, resp);
29     }
30 }
```

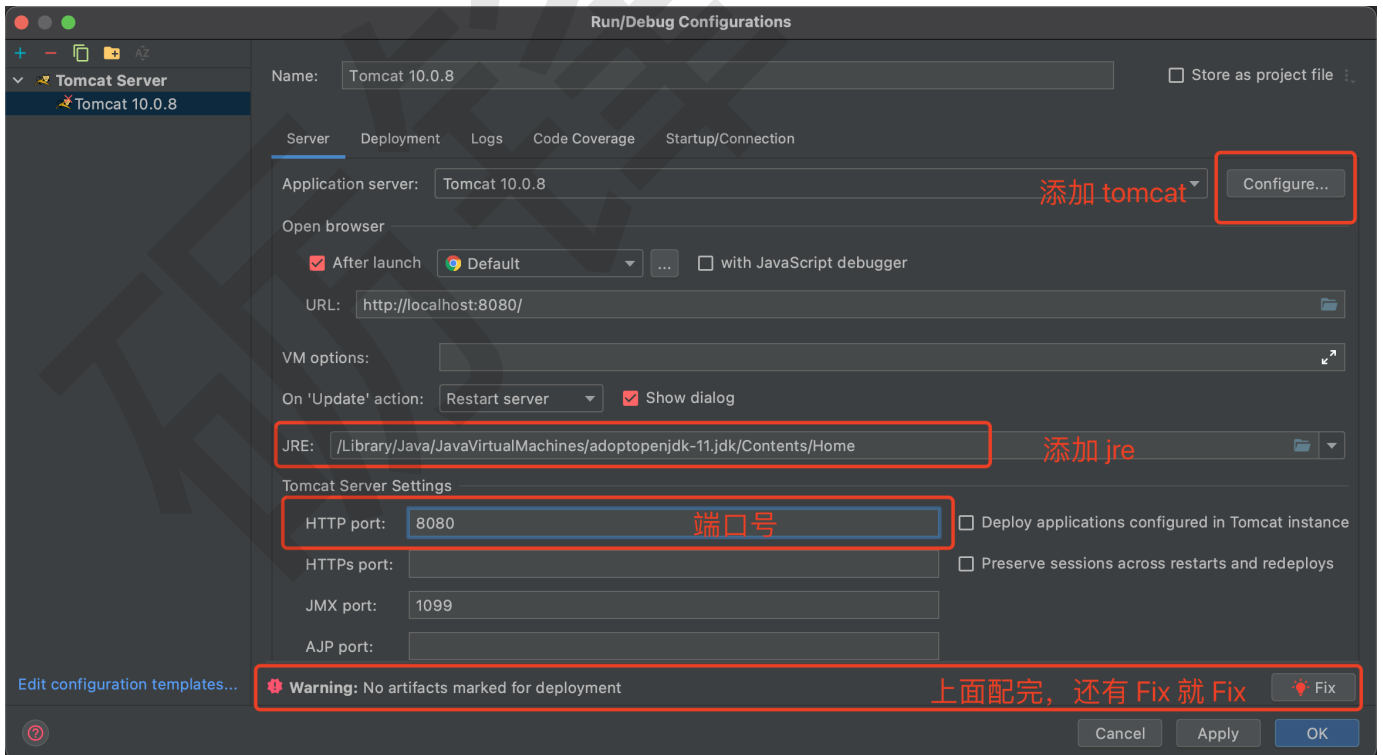
新增 Filter

```
1 package com.rushuni.filter;
2
3 import jakarta.servlet.*;
4 import jakarta.servlet.annotation.WebFilter;
5
6 import java.io.IOException;
7
8 /**
9  * @author rushuni
10  * @date 2021年07月22日 10:54 上午
11  */
12 @WebFilter("/*")
13 public class Demo01Filter implements Filter {
14
15     @Override
16     public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
17         System.out.println("DemoFilter01...");
18
19         //处理乱码
20         servletResponse.setContentType("text/html;charset=UTF-8");
21
22         //放行
23         filterChain.doFilter(servletRequest, servletResponse);
24     }
25
26 }
```

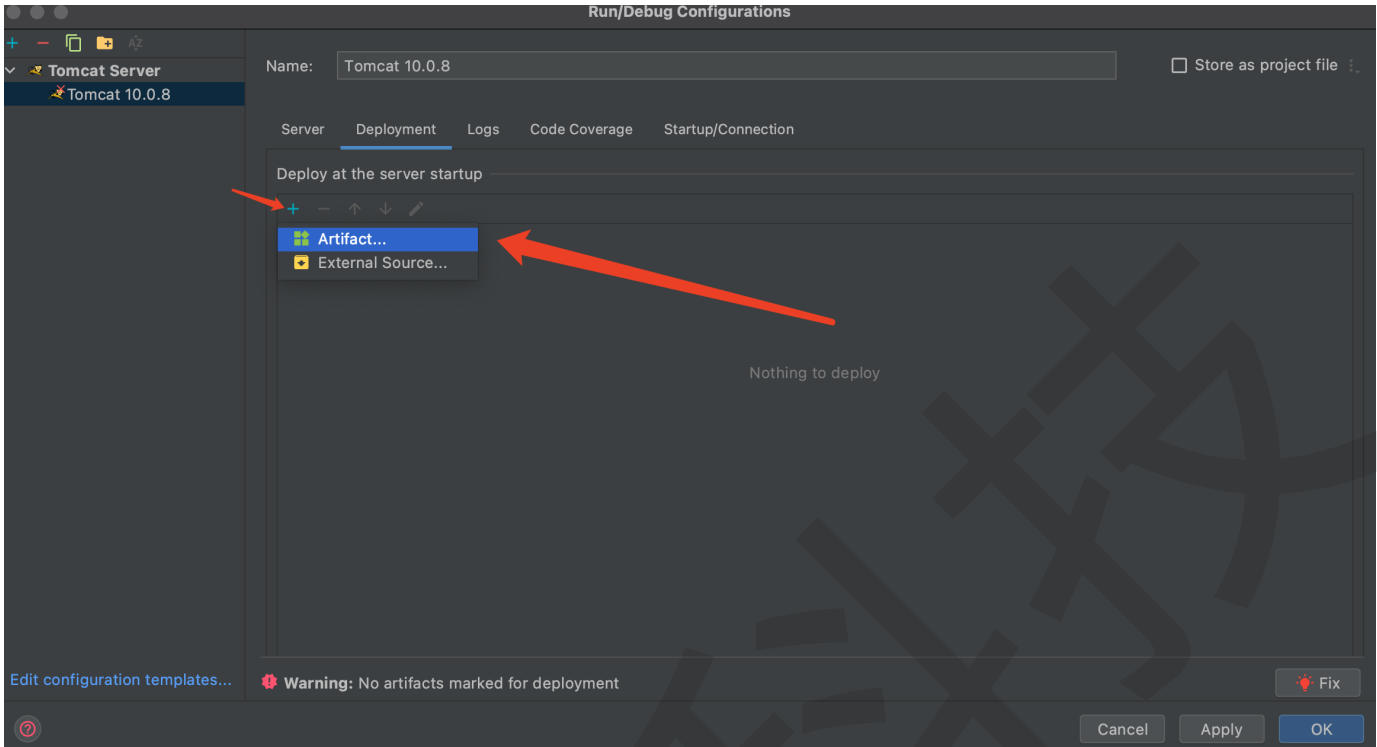
添加Tomcat



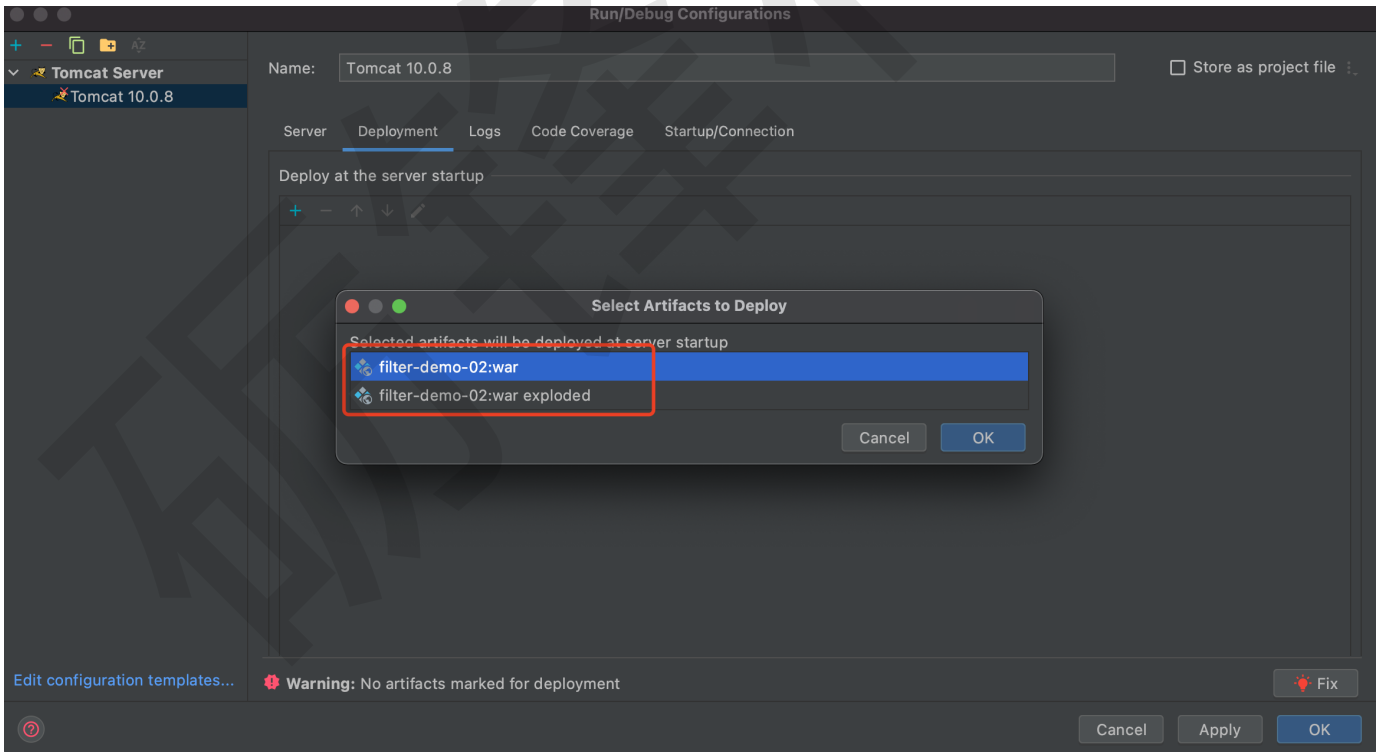
配置 Tomcat



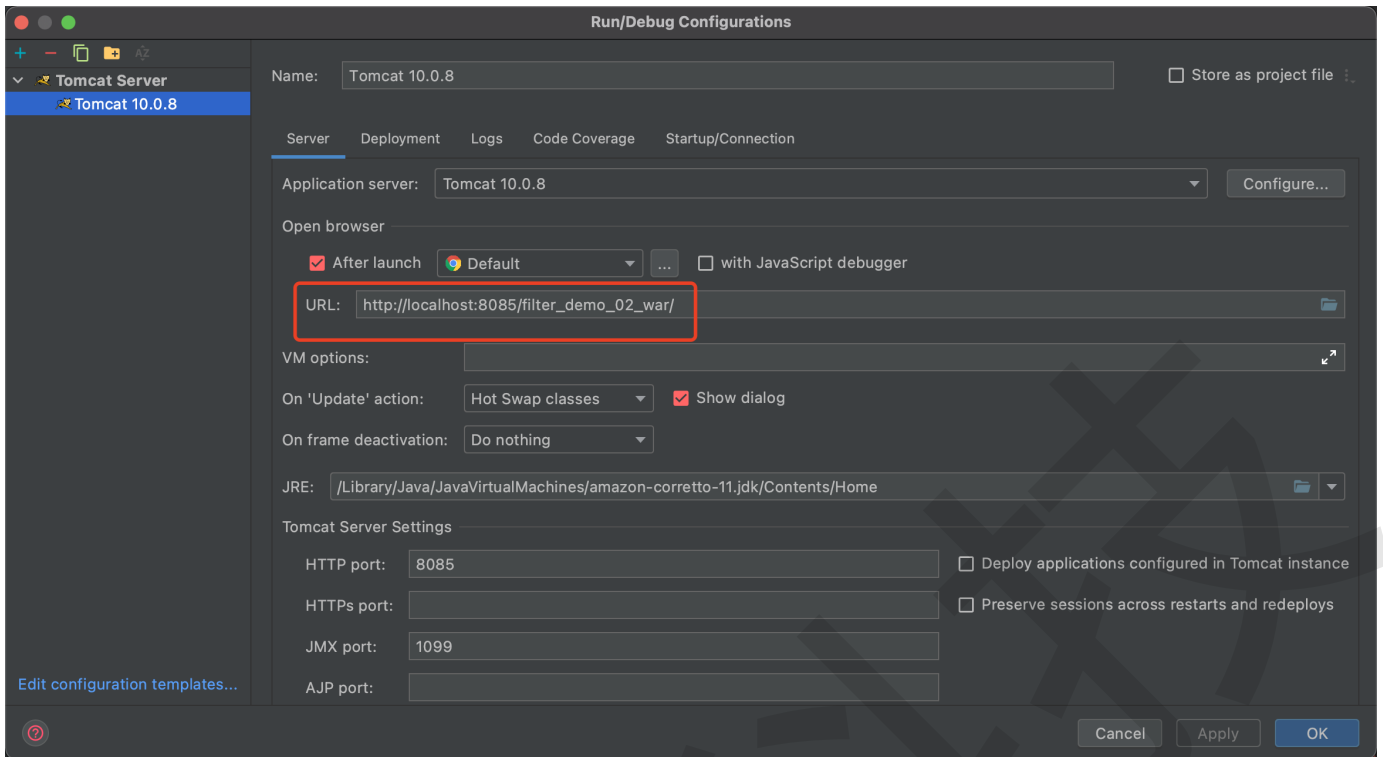
项目部署到 Tomcat



选择 Artifact... 后进行选择：



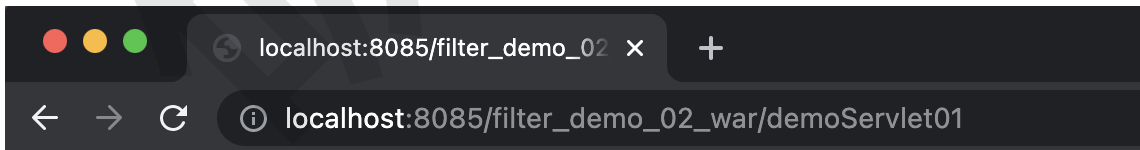
正确配完：



运行



浏览器打开:



demoServlet01 执行中...

课堂作业

- 创建 Web 项目，完成过滤器案例

案例分析

当我们启动服务，在地址栏输入访问地址后，发现浏览器任何内容都没有，控制台却输出了：
Demo01Filter拦截到了请求。

这说明案例中访问任何资源的时候，都需要先经过过滤器。

这是因为：我们在配置过滤器的拦截规则时，使用了 `/*`，表明访问当前应用下任何资源，此过滤器都会起作用。

除了这种全部过滤的规则之外，它还支持特定类型的过滤配置。

比如：`@WebFilter("/demoServlet01")`，这样将不会拦截所有请求。

过滤器常用注解

Servlet3.0 提供 `@WebFilter` 注解将一个实现了 `javax.servlet.Filter` 接口的类定义为过滤器。

这样我们在web应用中使用过滤器时，不需要在 `web.xml` 文件中配置过滤器的相关描述信息了。

`WebFilter` 的常用属性介绍：

`filterName` `String` 指定过滤器的 `name` 属性，等价于 `<filter-name>`

`value` `String[]` 该属性等价于 `urlPatterns` 属性。但是两者不应该同时使用。

`urlPatterns` `String[]` 指定一组过滤器的 URL 匹配模式。等价于 `<url-pattern>` 标签。

`servletNames` `String[]` 指定过滤器将应用于哪些 Servlet。取值是 `@WebServlet` 中的 `name` 属性的取值，或者是 `web.xml` 中 `<servlet-name>` 的取值。

过滤器细节

生命周期

- 出生——活着——死亡

出生：当应用加载的时候执行实例化和初始化方法。

活着：只要应用一直提供服务，对象就一直存在。

死亡：当应用卸载时，或者服务器宕机时，对象消亡。

Filter 的实例对象在内存中也只有一份。所以也是单例的。

doFilter 方法细节


filter执行了两遍

如果跳转网页，会发现过滤器执行了两遍，这是因为favicon.ico 的资源请求也被过滤器拦截了。

添加如下代码即可：

```
1  HttpServletRequest req = (HttpServletRequest) servletRequest;
2  String requestURI = req.getRequestURI();
3  if (requestURI.contains("favicon.ico")) {
4      return;
5  }
```

Java

 复制代码

filter 返回

如下代码：

```

12 @WebFilter("/demoServlet01")
13 public class Demo01Filter implements Filter {
14
15     ....@Override
16     public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
17         ....
18         ....System.out.println("Demo01Filter 过滤器执行中...");
19
20         ....//处理乱码
21         ....servletResponse.setContentType("text/html;charset=UTF-8");
22
23         ....//放行
24         ....filterChain.doFilter(servletRequest,servletResponse);
25         ....System.out.println("Demo01Filter 放行之后, 又回到了 doFilter 方法");
26     ....}
27
28 }

```

过滤器放行之后执行完目标资源，仍会回到过滤器中：

```

[2021-07-22 11:37:01,382] Artifact filter-demo-02:war: Deploy took 614 milliseconds
Demo01Filter 过滤器执行中...
demoServlet01 执行中...
Demo01Filter 放行之后, 又回到了 doFilter 方法

```

过滤器配置 (xml)

```

1 <filter>
2   <filter-name>filterDemo01</filter-name>
3   <filter-class>com.rushuni.filter.DemoFilter01</filter-class>
4 </filter>
5 <filter-mapping>
6   <filter-name>filterDemo01</filter-name>
7   <url-pattern>/*</url-pattern>
8 </filter-mapping>

```

XML

复制代码

过滤器配置 (注解)

```

1 @WebFilter("/*")
2 public class DemoFilter01 implements Filter {
3
4 }

```

Java

复制代码

过滤器配置-初始化参数 (xml)

XML | 复制代码

```
1 <filter>
2   <filter-name>filterDemo02</filter-name>
3   <filter-class>com.rushuni.filter.DemoFilter02</filter-class>
4   <init-param>
5     <param-name>username</param-name>
6     <param-value>zhangsan</param-value>
7   </init-param>
8   <init-param>
9     <param-name>password</param-name>
10    <param-value>123</param-value>
11  </init-param>
12 </filter>
13 <filter-mapping>
14   <filter-name>filterDemo02</filter-name>
15   <url-pattern>/*</url-pattern>
16 </filter-mapping>
```

过滤器配置-初始化参数 (注解)

Java | 复制代码

```
1 @WebFilter(filterName = "filterDemo02",
2           urlPatterns = "/*",
3           initParams={@WebInitParam(name="username", value="张三"),
4                        @WebInitParam(name="password", value="123")})
5 public class DemoFilter02 implements Filter {
6     // ...
7 }
```

多个过滤器的执行顺序

当配置了多个过滤器，过滤器执行的前后顺序决定过滤器的调用顺序，也就是由映射配置顺序决定。

当 xml 配置和注解配置同时存在，先调用 xml 配置，再注解配置。

如果 xml 的过滤器和注解的过滤器名称相同，只会跑 xml 的过滤器。

过滤器的五种拦截行为

我们的过滤器目前拦截的是请求，但是在实际开发中，我们还有请求转发和请求包含，以及由服务器触发调用的全局错误页面。默认情况下过滤器是不参与过滤的，要想使用，需要我们配置。

XML:

```

1
2 <filter>
3   <filter-name>filterDemo03</filter-name>
4   <filter-class>com.rushuni.filter.DemoFilter03</filter-class>
5   <!--配置开启异步支持，当dispatcher配置ASYNC时，需要配置此行-->
6   <async-supported>true</async-supported>
7 </filter>
8 <filter-mapping>
9   <filter-name>filterDemo03</filter-name>
10  <url-pattern>/*</url-pattern>
11  <!--过滤请求：默认值。-->
12  <dispatcher>REQUEST</dispatcher>
13  <!--过滤全局错误页面：当由服务器调用全局错误页面时，过滤器工作-->
14  <dispatcher>ERROR</dispatcher>
15  <!--过滤请求转发：当请求转发时，过滤器工作。-->
16  <dispatcher>FORWARD</dispatcher>
17  <!--过滤请求包含：当请求包含时，过滤器工作。它只能过滤动态包含，jsp的include指令是静态包
    含-->
18  <dispatcher>INCLUDE</dispatcher>
19  <!--过滤异步类型，它要求我们在filter标签中配置开启异步支持-->
20  <dispatcher>ASYNC</dispatcher>
21 </filter-mapping>
22
23 <!--配置全局错误页面-->
24 <error-page>
25   <exception-type>java.lang.Exception</exception-type>
26   <location>/error.jsp</location>
27 </error-page>
28 <error-page>
29   <error-code>404</error-code>
30   <location>/error.jsp</location>
31 </error-page>

```

注解：

```

1 @WebFilter(filterName = "Demo03",
2           urlPatterns = "/*",
3           dispatcherTypes = {DispatcherType.REQUEST,
4                               DispatcherType.ERROR,
5                               DispatcherType.FORWARD,
6                               DispatcherType.INCLUDE,
7                               DispatcherType.ASYNC})
8 public class DemoFilter03 implements Filter {
9     //...
10 }

```

Filter 与 Servlet 的区别

方法/类型	Servlet	Filter	备注
初始化 方法	<code>void init(ServletConfig);</code>	<code>void init(FilterConfig);</code>	几乎一样，都是在web.xml中配置参数，用该对象的方法可以获取到。
提供服务方法	<code>void service(request,response);</code>	<code>void dofilter(request,response,FilterChain);</code>	Filter比Servlet多了一个FilterChain，它不仅能完成Servlet的功能，而且还可以决定程序是否能继续执行。所以过滤器比Servlet更为强大。在 Struts2 中，核心控制器就是一个过滤器。
销毁方法	<code>void destroy();</code>	<code>void destroy();</code>	

课堂作业

- 完成 filter 的 xml 和注解的配置案例，熟悉 filter 的特性。

Filter 案例

字符编码过滤器

```
1 package com.rushuni.filter;
2
3 import javax.servlet.*;
4 import javax.servlet.annotation.WebFilter;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import java.io.IOException;
8
9 /**
10  * @author rushuni
11  * @date 2021年07月22日 1:58 下午
12  */
13 @WebFilter("/*")
14 public class DemoFilter04 implements Filter {
15
16     @Override
17     public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) {
18         try{
19             //1.将请求和响应对象转换为和HTTP协议相关
20             HttpServletRequest request = (HttpServletRequest) servletRequest;
21             HttpServletResponse response = (HttpServletResponse)
servletResponse;
22
23             //2.设置编码格式
24             request.setCharacterEncoding("UTF-8");
25             response.setContentType("text/html;charset=UTF-8");
26
27             //3.放行
28             filterChain.doFilter(request,response);
29         } catch (Exception e) {
30             e.printStackTrace();
31         }
32     }
33 }
34
35 }
```

登录校验拦截过滤器

```
1 package com.rushuni.filter;
2
3 import javax.servlet.Filter;
4 import javax.servlet.FilterChain;
5 import javax.servlet.ServletException;
6 import javax.servlet.ServletResponse;
7 import javax.servlet.annotation.WebFilter;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 /**
12  * @author rushuni
13  * @date 2021年07月22日 1:58 下午
14  */
15 @WebFilter(value = {"/list.jsp", "/listServlet"})
16 public class DemoFilter05 implements Filter {
17
18     @Override
19     public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) {
20         try{
21             // 1.将请求和响应对象转换为和HTTP协议相关
22             HttpServletRequest request = (HttpServletRequest) servletRequest;
23             HttpServletResponse response = (HttpServletResponse)
servletResponse;
24
25             // 2.获取会话域对象中数据
26             Object username = request.getSession().getAttribute("username");
27
28             // 3.判断用户名
29             if(username == null || "".equals(username)) {
30                 //重定向到登录页面
31                 response.sendRedirect(request.getContextPath() +
"/login.jsp");
32                 return;
33             }
34
35             // 4.放行
36             filterChain.doFilter(request,response);
37
38         } catch (Exception e) {
39             e.printStackTrace();
40         }
41     }
42
43 }
```



```
1 package com.rushuni.servlet;
2
3 import javax.servlet.ServletException;
4 import javax.servlet.annotation.WebServlet;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import java.io.IOException;
9
10 /**
11  * @author rushuni
12  * @date 2021年07月22日 2:45 下午
13  */
14 @WebServlet("/loginServlet")
15 public class LoginServlet extends HttpServlet{
16     @Override
17     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
18     throws ServletException, IOException {
19         // 1. 获取用户名和密码
20         String username = req.getParameter("username");
21         String password = req.getParameter("password");
22
23         // 2. 判断用户名
24         if(username == null || "".equals(username)) {
25             //2.1 用户名为空 重定向回登录页面
26             resp.sendRedirect(req.getContextPath() + "/login.jsp");
27             return;
28         }
29
30         // 2.2 用户名不为空 将用户名存入会话域中
31         req.getSession().setAttribute("username",username);
32
33         // 3. 重定向到首页index.jsp
34         resp.sendRedirect(req.getContextPath() + "/index.jsp");
35     }
36
37     @Override
38     protected void doPost(HttpServletRequest req, HttpServletResponse resp)
39     throws ServletException, IOException {
40         doGet(req,resp);
41     }
42 }
```

login.jsp

```
1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <html lang="zh-CN">
3  <head>
4      <title>登录</title>
5  </head>
6  <body>
7      <form action="${pageContext.request.contextPath}/loginServlet" method="get"
        autocomplete="off">
8          姓名: <input type="text" name="username"> <br>
9          密码: <input type="password" name="password"> <br>
10         <button type="submit">登录</button>
11     </form>
12 </body>
13 </html>
```