目标：

过滤器:

1. 过滤器
   1. 基于概念
   2. 写一个HelloWorld
   3. 总结Api及执行流程
2. 过滤器案例
3. 综合案例

# 过滤器

## 基本概念

(3W1H: why, what, where，how)

### 为什么需用到过滤器？

项目开发中，经常会涉及到重复代码的实现！

注册 ----🡪 Servlet 【1. 设置编码】 ----🡪 JSP

修改 ----🡪Servlet 【1. 设置编码】 ---🡪 JSP

其他，

如判断用户是否登陆，只有登陆才能有操作权限！

涉及到重复判断： 获取session，取出session数据，判断是否为空,为空说明没有登陆，不能操作； 只有登陆后，才能操作！

如何解决：

1. 抽取重复代码，封装
2. 每个用到重复代码的地方，手动的调用！

过滤器，设计执行流程：

1. 用户访问服务器
2. 过滤器： 对Servlet请求进行拦截
3. 先进入过滤器， 过滤器处理
4. 过滤器处理完后， 在放行， 此时，请求到达Servlet/JSP
5. Servlet处理
6. Servlet处理完后，再回到过滤器, 最后在由tomcat服务器相应用户；

(过滤器就像回家的门！)

### 过滤器,HelloWorld案例

Javax.servlet.\*;

|-- interface Filter 及过滤器

开发步骤：

1. 写一个普通java类，实现Filter接口
2. 配置过滤器

### 过滤器执行流程

OOAD 面向对象的分析与设计

使用RationRose 时序图



### 过滤器相关Api

|-- interface Filter 过滤器核心接口

Void init(filterConfig); 初始化方法，在服务器启动时候执行

Void doFilter(request,response,filterChain); 过滤器拦截的业务处理方法

Void destroy(); 销毁过滤器实例时候调用

|-- interface FilterConfig 获取初始化参数信息

|  |  |
| --- | --- |
| String | [**getInitParameter**](mk:@MSITStore:E:\API\servlet-api_en.chm::/javax/servlet/FilterConfig.html#getInitParameter(java.lang.String))(java.lang.String name) |
| Enumeration | [**getInitParameterNames**](mk:@MSITStore:E:\API\servlet-api_en.chm::/javax/servlet/FilterConfig.html#getInitParameterNames())**()** |

|-- interface FilterChain 过滤器链参数；一个个过滤器形成一个执行链；

void doFilter(ServletRequest request, ServletResponse response) ; 执行下一个过滤器或放行

|  |
| --- |
| /\*\*  \* 过滤器，测试  \* **@author** Jie.Yuan  \*  \*/  **public** **class** HelloFilter **implements** Filter{    // 创建实例  **public** HelloFilter(){  System.*out*.println("1. 创建过滤器实例");  }  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {  System.*out*.println("2. 执行过滤器初始化方法");    // 获取过滤器在web.xml中配置的初始化参数  String encoding = filterConfig.getInitParameter("encoding");  System.*out*.println(encoding);    // 获取过滤器在web.xml中配置的初始化参数 的名称  Enumeration<String> enums = filterConfig.getInitParameterNames();  **while** (enums.hasMoreElements()){  // 获取所有参数名称：encoding、path  String name = enums.nextElement();  // 获取名称对应的值  String value = filterConfig.getInitParameter(name);  System.*out*.println(name + "\t" + value);  }  }  // 过滤器业务处理方法： 在请求到达servlet之前先进入此方法处理公用的业务逻辑操作  @Override  **public** **void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** IOException, ServletException {  System.*out*.println("3. 执行过滤器业务处理方法");  // 放行 (去到Servlet)  // 如果有下一个过滤器，进入下一个过滤器，否则就执行访问servlet  chain.doFilter(request, response);    System.*out*.println("5. Servlet处理完成，又回到过滤器");  }  @Override  **public** **void** destroy() {  System.*out*.println("6. 销毁过滤器实例");  }  } |

|  |
| --- |
| <!-- 过滤器配置 -->  <filter>  <!-- 配置初始化参数 -->  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>path</param-name>  <param-value>c:/...</param-value>  </init-param>    <!-- 内部名称 -->  <filter-name>hello\_filter</filter-name>  <!-- 过滤器类的全名 -->  <filter-class>cn.itcast.a\_filter\_hello.HelloFilter</filter-class>  </filter>  <filter-mapping>  <!-- filter内部名称 -->  <filter-name>hello\_filter</filter-name>  <!-- 拦截所有资源 -->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

### 对指定的请求拦截

|  |
| --- |
| **/\* 表示拦截所有的请求** |
| <filter-mapping>  <filter-name>hello\_filter2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

默认拦截的类型：(直接访问或者重定向)

<dispatcher>REQUEST</dispatcher>

拦截转发：

<dispatcher>FORWARD</dispatcher>

拦截包含的页面(RequestDispatcher.include(/page.jsp); 对page.jsp也执行拦截)

<dispatcher>INCLUDE</dispatcher>

拦截声明式异常信息：

<dispatcher>ERROR</dispatcher>

|  |
| --- |
| <!-- 配置第二个过滤器 -->  <!-- 演示： 拦截指定的请求 -->  <filter>  <filter-name>hello\_filter2</filter-name>  <filter-class>cn.itcast.a\_filter\_hello.HelloFilter2</filter-class>  </filter>  <filter-mapping>  <filter-name>hello\_filter2</filter-name>  <!-- 1. 拦截所有  <url-pattern>/\*</url-pattern>  -->    <!-- 2. 拦截指定的jsp  <url-pattern>/index.jsp</url-pattern>  <url-pattern>/list.jsp</url-pattern>  -->  <!-- 拦截所有的jsp  <url-pattern>\*.jsp</url-pattern>  -->  <!-- 3. 根据servlet的内部名称拦截  <servlet-name>IndexServlet</servlet-name>  -->  <!-- 拦截指定的servlet  <url-pattern>/index</url-pattern>  -->    <!-- 4. 指定拦截指定的类型 -->  <url-pattern>/\*</url-pattern>  <dispatcher>REQUEST</dispatcher>  <dispatcher>FORWARD</dispatcher>  </filter-mapping> |

共性问题：

1. 过滤器：方法参数没有自动命名，说明没有关联源码

--🡪 关联tomcat或servlet源代码

2. 连接池： 多刷新几次，报错！

-🡪 连接没关

QueryRunner qr = new QueryRunner();

qr.update(con,sql);

// 这里con一定要关闭

-🡪 注意：dataSource 确定一个项目创建一次

QueryRunner qr = new QueryRunner(dataSource);

🡪 修改连接池参数配置

3 . 编码

// 设置POST提交的请求的编码

request.setCharacterEncoding("UTF-8");

// 设置相应体的编码

response.setCharacterEncoding("UTF-8");

// 设置页面打开时候时候的编码格式、 设置相应体的编码

response.setContentType("text/html;charset=UTF-8");

开发中：

工作区间编码、项目编码、request/response、数据库编码一致！

# 2.案例

## 过滤器-编码统一处理

几乎每一个Servlet都要涉及编码处理：处理请求数据中文问题！

【GET/POST】

每个servlet都要做这些操作，把公用的代码抽取-过滤器实现！

代码实现思路:

1. Login.jsp 登陆，输入“中文”
2. LoginServlet.java 直接处理登陆请求
3. EncodingFilter.java 过滤器处理请求数据编码：GET/POST

|  |
| --- |
| 过滤器：  /\*\*  \* 编码处理统一写到这里(servlet中不需要再处理编码)  \* **@author** Jie.Yuan  \*  \*/  **public** **class** EncodingFilter **implements** Filter {  // 过滤器业务处理方法：处理的公用的业务逻辑操作  @Override  **public** **void** doFilter(ServletRequest req, ServletResponse res,  FilterChain chain) **throws** IOException, ServletException {    // 转型  **final** HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) res;    // 一、处理公用业务  request.setCharacterEncoding("UTF-8"); // POST提交有效  response.setContentType("text/html;charset=UTF-8");    /\*  \* 出现GET中文乱码，是因为在request.getParameter方法内部没有进行提交方式判断并处理。  \* String name = request.getParameter("userName");  \*  \* 解决：对指定接口的某一个方法进行功能扩展，可以使用代理!  \* 对request对象(目标对象)，创建代理对象！  \*/  HttpServletRequest proxy = (HttpServletRequest) Proxy.*newProxyInstance*(  request.getClass().getClassLoader(), // 指定当前使用的累加载器  **new** Class[]{HttpServletRequest.**class**}, // 对目标对象实现的接口类型  **new** InvocationHandler() { // 事件处理器  @Override  **public** Object invoke(Object proxy, Method method, Object[] args)  **throws** Throwable {  // 定义方法返回值  Object returnValue = **null**;  // 获取方法名  String methodName = method.getName();  // 判断：对getParameter方法进行GET提交中文处理  **if** ("getParameter".equals(methodName)) {    // 获取请求数据值【 <input type="text" name="userName">】  String value = request.getParameter(args[0].toString()); // 调用目标对象的方法    // 获取提交方式  String methodSubmit = request.getMethod(); // 直接调用目标对象的方法    // 判断如果是GET提交，需要对数据进行处理 (POST提交已经处理过了)  **if** ("GET".equals(methodSubmit)) {  **if** (value != **null** && !"".equals(value.trim())){  // 处理GET中文  value = **new** String(value.getBytes("ISO8859-1"),"UTF-8");  }  }  **return** value;  }  **else** {  // 执行request对象的其他方法  returnValue = method.invoke(request, args);  }    **return** returnValue;  }  });    // 二、放行 (执行下一个过滤器或者servlet)  chain.doFilter(proxy, response); // 传入代理对象  }  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {    }  @Override  **public** **void** destroy() {    }  } |
| 过滤器配置：  <!-- 编码处理过滤器配置 -->  <filter>  <filter-name>encoding</filter-name>  <filter-class>cn.itcast.a\_loginFilter.EncodingFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>encoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| Servlet： |
| **public** **class** LoginServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  // 获取请求数据  String name = request.getParameter("userName");  System.*out*.println("用户：" + name);  } |

## 过滤器-无效数据过滤

模拟：论坛过滤敏感词汇！

实现思路：

1. Dis.jsp 讨论区页面
2. DisServlet.java 处理提交

---》 获取请求参数

---》 保存到request域

-----》 跳转dis.jsp 【从request取数据显示(处理后)】

1. DataFilter.java 过滤器

----》编码

---》 无效数据处理

即： 在上一个案例基础上，再添加无效数据过滤的相关代码!

JSP引入ckeditor组件：客户端组件，便于用户输入内容！

|  |
| --- |
| JSP |
| <!-- 引入ckeditor组件(给用户输入提供方便) -->  <script src=*"*${pageContext.request.contextPath }*/ckeditor/ckeditor.js"*></script>  <link rel=*"stylesheet"* href=*"*${pageContext.request.contextPath }*/ckeditor/samples/sample.css"*>    <body>  ${requestScope.content }    <form name=*"frmDis"* action=*"*${pageContext.request.contextPath }*/dis"* method=*"post"*>  发表评论: <textarea class=*"ckeditor"* rows=*"6"* cols=*"30"* name=*"content"*></textarea>    <br/>  <input type=*"submit"* value=*"评论"* >  </form>  </body> |
| Filter: |
| 在上个过滤器案例的基础上，增加如下代码： |
| // 中文数据已经处理完： 下面进行无效数据过滤  //【如何value中出现dirtyData中数据，用\*\*\*\*替换】  **for** (String data : dirtyData) {  // 判断当前输入数据(value), 是否包含无效数据  **if** (value.contains(data)){  value = value.replace(data, "\*\*\*\*\*");  }  } |

## 登陆权限判断

登陆， 提交到登陆Servlet处理其业务！

-🡪登陆成功, 跳转到首页，显示欢迎信息 + 列表信息

-🡪登陆失败，跳转到登陆！

要求：

只有登陆后，才可以访问首页, 显示列表

如果没有登陆，直接访问首页列表，要跳转到登陆!

实现思路：

1. Login.jsp 登陆页面
2. List.jsp 列表显示
3. LoginServlet.java 登陆处理servlet
4. IndexServlet.java 首页列表查询Servlet
5. LoginFilter.java 登陆验证过滤器

(用之前的表：

admin存储登陆用户, 登陆用

employee 存储员工信息，列表显示用！

)

实现步骤：

1. 建库、建表、建项目、引入jar文件
2. entity
   1. Admin.java
   2. Employee.java
3. Dao
   1. AdminDao
   2. EmployeeDao
4. Servcie
5. Servlet
6. Jsp

<http://localhost:8080/emp_sys/login.jsp> 可以直接访问

<http://localhost:8080/emp_sys/login> 可以直接访问

<http://localhost:8080/emp_sys/index> 不能直接访问

<http://localhost:8080/emp_sys/list.jsp> 不能直接访问