信息管理系统-过滤器&监听器

课程目标

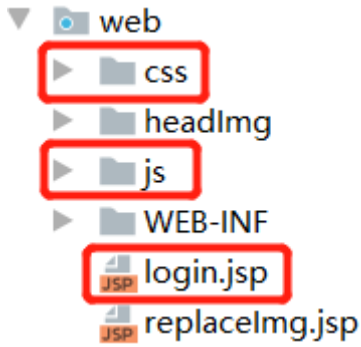
- 理解和掌握用户注销的逻辑操作。
- 理解和掌握记住账号的逻辑操作。
- 理解验证码的实现原理，能够使用验证码即可。
- 理解过滤器的作用
- 掌握编码和登录校验过滤器的简单版本，了解编码和登录校验过滤器的优化操作。
- 理解监听器的作用
- 掌握使监听器去创建第一个超级管理员账号。

一、用户注销

用户注销则为退出登录状态，将当前登录的用户信息销毁。而我们之前做的用户登录功能，是把当前登录用户信息是存在 session 中的，所以只需要清除 session 中的用户信息即可实现用户注销。

1、修改 login.jsp

为了登录页面好看，加入新的登录页面，按照给的**资料\模板页面**目录里面去拷贝即可（login.jsp，css 目录，js 目录）。



2、修改 list.jsp

修改产品的 list.jsp，提供给用户点击可以安全退出的链接，点击发送请求交由 Servlet 去处理。

```
欢迎：${USER_IN_SESSION.username} <a href="/logout"> 安全退出</a><br/>
```

3、编写 LogoutServlet.java

处理安全退出的请求，清除 session 中用户的登录信息，并重定到 login.jsp。

```
package cn.wolfcode.web.servlet;

@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
    @Override
```

```
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 方式一：删除用户共享数据
        // req.getSession().removeAttribute("USER_IN_SESSION");

        // 方式二：销毁整个 Session 对象
        req.getSession().invalidate();
        resp.sendRedirect("/login.jsp");
    }
}
```

二、记住账号

为了提升用户体验，在用户登录的时候可以选择记住账号信息，在指定时间范围内可以不用再重新填写账户信息。
因为需要将用户信息保存一段时间，放 session 的话浏览器关闭则会失效，所以选择使用 Cookie 来实现。

1、修改 login.jsp

提供勾选记住我复选框，点击登录时会传递参数 rememberMe 给 LoginServlet。

```
<label>
    <!-- 为了方便后台转换，value 加上勾选后的值为 true ，默认勾选值为 on -->
    <input type="checkbox" name="rememberMe" value="true"> 记住我
</label>
```

2、修改 LoginServlet.java

获取请求参数 rememberMe，若是选中（true），创建 Cookie，指定时间并响应给浏览器；若没有选中，清除 Cookie 对象。

```
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    req.setCharacterEncoding("UTF-8");
    // 接受请求参数
    String username = req.getParameter("username");
    String password = req.getParameter("password");
    try {
        // 调用业务方法来处理登录请求
        User user = userService.login(username, password);
        // 登录成功时往 Session 中加入一个登录成功的标识，给其它需要登录才可以操作的资源做判
断使用
        req.getSession().setAttribute("USER_IN_SESSION", user);

        //============== 是否记住我 ===============
        boolean rememberMe = Boolean.valueOf(req.getParameter("rememberMe"));
        Cookie cookie = new Cookie("username", username);
        cookie.setPath("/");
        if (rememberMe){
            // 设置 Cookie 有效时间为一周
            cookie.setMaxAge(60 * 60 * 24 * 7);
```

```
        }else{
            // 删除 Cookie
            cookie.setMaxAge(0);
        }
        resp.addCookie(cookie);

        // 控制跳转
        resp.sendRedirect("/product");
    }catch (Exception e){ // 有异常时表示登录失败
        e.printStackTrace();
        req.setAttribute("errorMsg", e.getMessage());
        req.getRequestDispatcher("/login.jsp").forward(req, resp);
    }
}
```

3、在登录页面回显账号和记住我

修改 login.jsp 使用 EL 表达式回显账号和记住我。

```
<input type="text" class="form-control" placeholder="请输入账号" name="username"
value="${cookie.username.value}">
```

```
<label>
    <input type="checkbox" name="remeberMe" value="true"
        ${cookie.username.value != null ? 'checked' : ''}> 记住我
</label>
```

三、验证码

1、作用

验证码（CAPTCHA）是“Completely Automated Public Turing test to tell Computers and Humans Apart”（**全自动区分计算机和人类的图灵测试**）的缩写，是一种**区分用户是计算机还是人的公共全自动程序。**可以**防止：恶意破解密码、刷票、论坛灌水，有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登录尝试**，实际上用验证码是现在很多网站通行的方式，我们利用比较简易的方式实现了这个功能。这个问题可以由计算机生成并评判，但是必须只有人类才能解答。由于计算机无法解答 CAPTCHA 的问题，所以回答出问题的用户就可以被认为是人类。

2、分类

- 图片验证码（最常用）。
- 手机短信验证码。
- 手机语音验证码。
- 视频验证码。

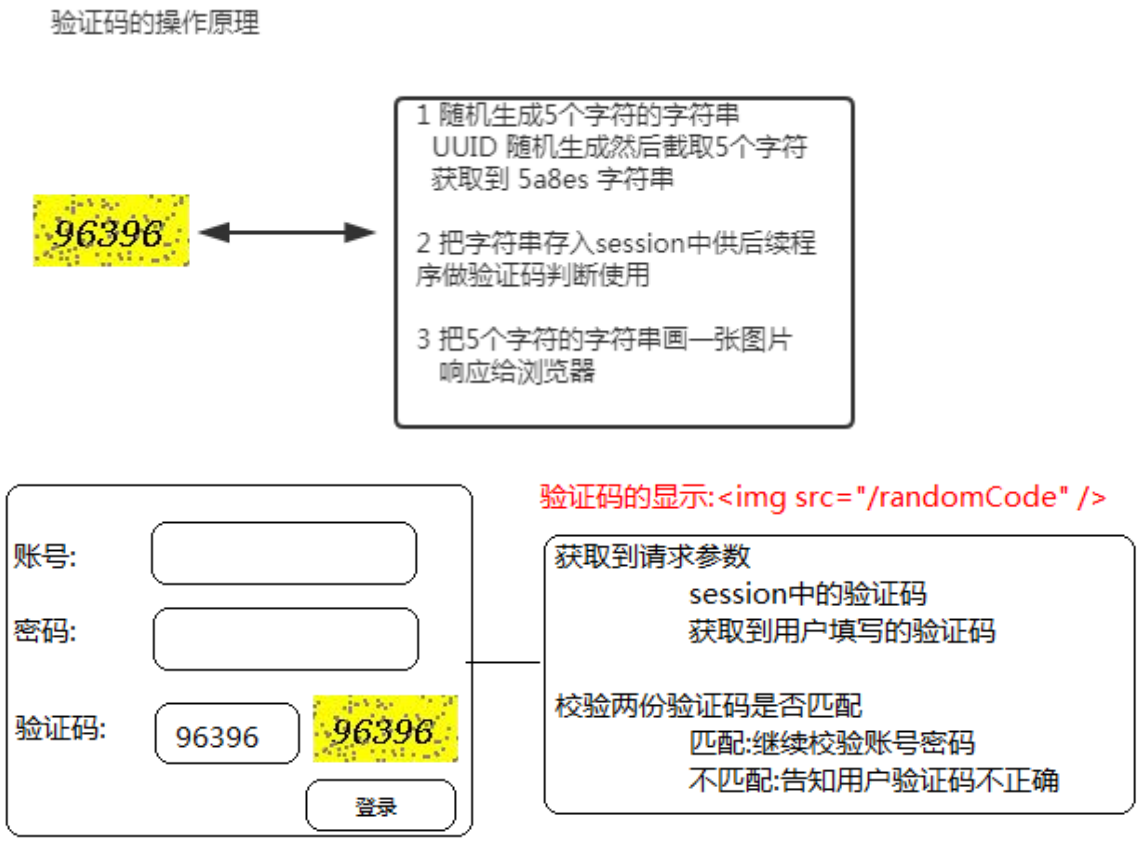
3、图片验证码原理

图片验证码功能分三个部分：

- A 程序生成随机内容的验证码图片展现给用户。
- 用户根据图片输入验证码文本。
- B 程序判断用户输入的验证码和 A 程序生成的验证是否相同。

问题：生成验证码的程序 A 和判断用户填写验证码的程序 B 不是同一个，如何在程序 B 中获取到程序 A 生成的验证码文本做判断？

解决：因为两个程序我们都是使用 Servlet 来实现的，所以可以把 A 程序生成 验证码之后存入 Session ，B 程序从 Session 中来获取。



4、图片验证码实现

需求：防止恶意登录，给登录加上验证码。

4.1、拷贝 RandomCodeServlet.java

这个负责生成图片验证码，并把验证码存在 Session 中。

4.2、修改在 login.jsp

提供 img 标签，显示验证码。

```
<div class="row">
    <div class="col-xs-8">
        <input type="text" class="form-control" placeholder="请输入验证码"
name="randomCode">
    </div>
    <div class="col-xs-4">
        <img style="width: 85px; height: 30px;" alt="验证码" src="/randomCode">
    </div>
</div>
```

4.3、修改 LoginServlet.java

加入验证码判断。

```
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    req.setCharacterEncoding("UTF-8");
    // 接受请求参数
    String username = req.getParameter("username");
    String password = req.getParameter("password");

    // 判断验证码是否正确
    String randomCode = req.getParameter("randomCode");
    String sessionRandomCode = (String)
req.getSession().getAttribute("RANDOMCODE_IN_SESSION");
    if(!(StringUtil.hasLength(randomCode)
            && StringUtil.hasLength(sessionRandomCode)
            && randomCode.equalsIgnoreCase(sessionRandomCode))) {
        // 跳回登录页面重新填写验证码
        req.setAttribute("errorMsg", "验证码错误");
        req.getRequestDispatcher("/login.jsp").forward(req, resp);
        return;
    }
    // 说明验证码正确，可以做登录校验，可以把验证码清除，验证码仅限于一次登录使用
    req.getSession().removeAttribute("RANDOMCODE_IN_SESSION");

    // 后面登录操作省略 ......
}
```

4.4、修改验证码

生成的图片验证码可能存在用户识别不了的情况，可提供点击更换图片验证码功能。

思路: 当 img 标签的 src 属性值改变了，浏览器会自动发送新的请求。那么我们就给图片添加一个点击事件处理，点击了图片，修改该图片 src 的值即可，重新请求验证码图片。

修改 login.jsp：

```
<!-- 为图片绑定一个点击事件 -->
<img id="randomCode" style="width: 85px; height: 30px; cursor: pointer" alt="验证
码" src="/randomCode" onclick="changeImg()">
```

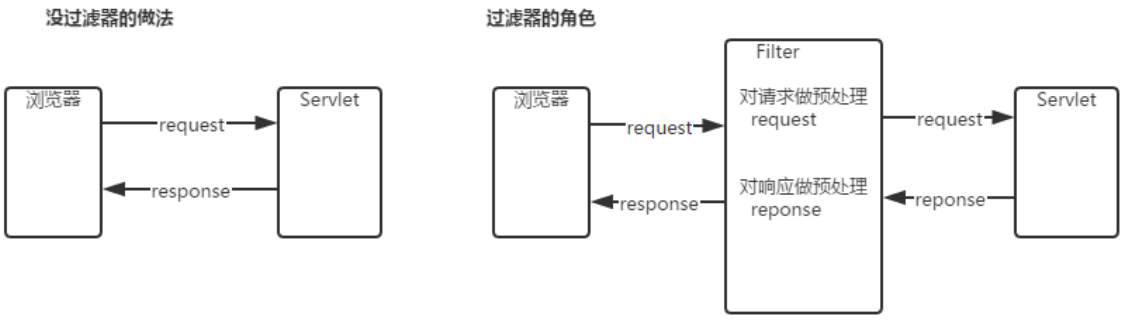
```
// 点击图片执行下面的函数
function changeImg() {
    // 修改验证码 img 标签的 src 属性值，为什么要带参数，禁用浏览器缓存
    document.getElementById("randomCode").src="/randomCode?" + new
Date().getTime();
}
```

浏览器有缓存，将 get 请求的结果缓存到本地磁盘，若请求相同的资源，不再发送新的请求，而是直接使用缓存中的结果。我们可以通过携带参数不一样来欺骗浏览器我们请求的资源不是同一个资源（实际上是同一个)。

四、过滤器 Filter

1、定义及应用

Filter 是 Servlet 2.3 新增加的功能，Java Web 组件之一。**使用户可以改变一个 request 和修改一个 response**。Filter 不是一个 Servlet，它不能产生一个 response，它能够在一个 request 到达 Servlet 之前预处理 request，也可以在 response 离开 Servlet 后处理 response。换种说法，Filter 其实是一个 "Servlet chaining"（Servlet 链）。



过滤器使用场景：

- 字符编码处理。
- 登录校验。
- 论坛敏感字过滤。
- 做前端框架的分发器。

2、Filter 的 Hello World

2.1、回顾 Servlet 开发步骤

回顾 Servlet 开发步骤，类比有助于学习 Filter，步骤如下：

- 定义类，并实现 javax.servlet.Servlet 接口。
- 覆写其中的 5 个方法，在 service 方法中编写处理请求做响应的代码。
- 将写好的 Servlet 交给 Tomcat 来管理（XML/注解）。

2.2、HelloServlet 代码演示

为了给大家更清楚地演示访问的执行流程，也写一个 HelloServlet。

2.2.1、编写 HelloServlet.java

```
package cn.wolfcode.web.servlet;

public class HelloServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("helloServlet");
        req.getRequestDispatcher("/hello.jsp");
    }
}
```

2.2.2、配置 HelloServlet

使用 XML 配置，在 web.xml 配置如下：

```
<servlet>
    <servlet-name>helloServlet</servlet-name>
    <servlet-class> cn.wolfcode.web.servlet.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>helloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

2.2.3、编写 hello.jsp

在 web 目录下新建 hello.jsp，内容如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>hello</title>
</head>
<body>
    欢迎
</body>
</html>
```

2.3、Filter 开发步骤

Filter 也是一个 Java Web 组件之一，所以开发步骤和 Servlet 一致，步骤如下：

- 定义类，实现 javax.servlet.Filter 接口。
- 覆写其中的 3 个方法。

- 将写好的 Filter 交给 Tomcat 来管理（XML/注解）。

2.3、HelloFilter 代码演示

2.3.1、编写 HelloFilter.java

```
package cn.wolfcode.web.filter;

public class HelloFilter implements Filter {

    // 初始化方法
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    // 过滤方法
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
```

```
        // 可以在这里做过滤操作
        System.out.println("进来过滤了...");
        // 目前直接放行
        filterChain.doFilter(servletRequest, servletResponse);
    }

    // 销毁方法
    @Override
    public void destroy() {

    }
}
```

2.3.2、配置 HelloFilter

使用 XML 配置，在 web.xml 配置如下：

```
<!-- 配置过滤器 -->
<filter>
    <filter-name>helloFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.HelloFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>helloFilter</filter-name>
    <!-- 需过滤的资源地址，即访问 /hello 会执行到过滤器 HelloFilter 的 doFilter 方法 --
>
    <url-pattern>/hello</url-pattern>
</filter-mapping>
```

2.3.3、测试

启动 Tomcat 访问 localhost/hello 测试一下。

3、过滤路径

过滤器的 url-pattern 是指 Filter 对哪一些资源做过滤操作。注意与 Servlet 区别：

- **Servlet 中的 url-pattern 的作用**：为当前 Servlet 起一个资源名称，外界通过该名字找到对应的 Servlet 对象。
- **Filter 中的 url-pattern 的作用**：指定对哪些资源做过滤。

配置过滤器路径举例说明：

- /hello ：说明当前 Filter 只会对 /hello 做拦截/过滤。
- /employee ：说明当前 Filter 只会对 /employee 资源做过滤。
- /system/* ：说明当前 Filter 只会对以 /system/ 作为前缀的资源路径做拦截。
- /* ：说明当前 Filter 会对所有的资源访问都会被拦截。

4、Filter 的生命周期

4.1、生命周期

Filter 的生命周期：指 Filter 从创建到销毁的整个过程。过程如下：

- **对象的创建**：在启动服务器的时候创建所有的 Filter 对象。
- **init 方法的执行**：在启动服务器的时候调用 Filter 对象中的 init 方法。

- **doFilter 方法的执行**：每次请求对应的资源的时候都会执行，只要路径符合。
- **destroy 方法的执行**：正常关闭服务器的时候，执行销毁操作，非正常关闭不会执行。

4.2、代码验证

修改 HelloFilter.java，提供构造器，并在其他初始化方法和销毁方法里面打印，修改完启动 Tomcat，访问 localhost/hello 测试一下，看打印结果。

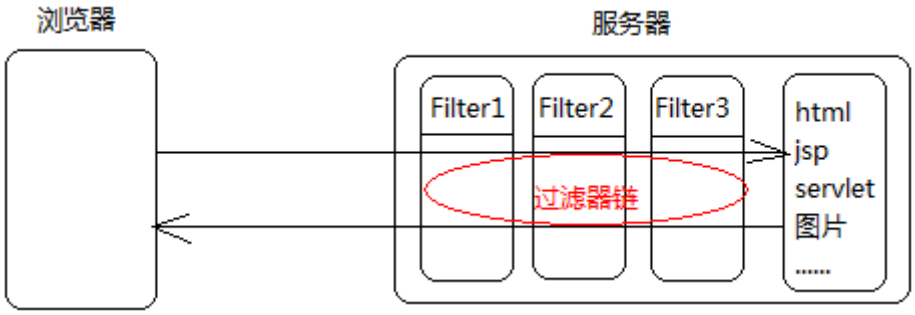
```
package cn.wolfcode.web.filter;

public class HelloFilter implements Filter {

    public HelloFilter() {
        System.out.println("HelloFilter 对象被创建");
    }

    // 初始化方法
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // filterConfig 可以获取到配置初始化参数
        System.out.println("初始化方法");
    }

    // 过滤方法
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        // 可以在这里做过滤操作
        System.out.println("进来 HelloFilter 过滤了..");
        // 目前直接放行
        filterChain.doFilter(servletRequest, servletResponse);
    }

    // 销毁方法
    @Override
    public void destroy() {
        System.out.println("销毁方法");
    }
}
```

4.3、执行顺序

创建对象（一次）---> init 方法（一次）---> doFilter 方法（N 次）---> destroy 方法(一次/ 0 次)

5、FilterChain（过滤器链）

在开发中会存在并配置了多个过滤器时，多个过滤器按照一定的顺序，排列起来，多个 Filter 组合在一起都形成一个 Filter 链，使用 FilterChain 对象来做牵引关联。



程序中，存在多个过滤器的时候，过滤器的先后执行顺序由谁来决定?

- 由在 web.xml 中配置的 `<filter-mapping>` 的先后顺序来决定。
- 注解配置时则是由 Filter 的名称的字母先后顺序来决定。

6、过滤方式（了解）

默认过滤器只对请求操作做过滤，针对转发是没有做过滤的，若要对转发方式等做过滤，需要设置过滤方式。过滤方式如下：

- REQUEST： 一次全新的请求，只有全新的请求才会经过过滤器（默认）。
- FORWARD： 请求转发。
- ERROR： 错误页面跳转。

6.1、过滤转发

在 web.xml 修改 HelloFilter 的过滤器方式，如下：

```
<filter-mapping>
    <filter-name>helloFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <!-- 过滤请求 -->
    <dispatcher>REQUEST</dispatcher>
    <!-- 过滤转发 -->
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

6.2、过滤器错误

6.2.1、编写 404.jsp 和 500.jsp

在 web 目录下新建 404.jsp 和 500.jsp，404.jsp 内容提示用户访问的资源不存在，500.jsp 内容提示用户程序员正在努力修复程序。

6.2.2、配置通用的错误页面

在 web.xml 中配置如下：

```
<error-page>
    <error-code>404</error-code>
    <location>/404.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/500.jsp</location>
</error-page>
```

6.3.2、修改过滤器过滤方式

在 web.xml 修改 HelloFilter 的过滤器方式，如下：

```
<filter-mapping>
    <filter-name>helloFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <!-- 过滤请求 -->
    <dispatcher>REQUEST</dispatcher>
    <!-- 过滤转发 -->
    <dispatcher>FORWARD</dispatcher>
    <!-- 过滤错误 -->
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

五、字符编码过滤器

1、编码过滤器引入

之前编码处理的代码是直接写 Servlet 中的，那么它存在以下问题：

- 代码重复，无法复用，从增加维护成本。

目标：一处设置，到处有效（设置一次，所有 Servlet 共用）。
方案：访问到 Servlet 之前对请求中的编码做处理（Filter）。

2、代码实现

2.1、修改所有 Servlet

删除其中对请求编码的的处理的代码。

2.2、编写 CharacterEncodingFilter.java

统一在过滤器的 doFilter 方法中设置好编码为 UTF-8。

```
package cn.wolfcode.web.filter;

public class CharacterEncodingFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        // 设置请求编码
        servletRequest.setCharacterEncoding("UTF-8");
        // 发行
        filterChain.doFilter(servletRequest, servletResponse);
    }
```

```
    @Override
    public void destroy() {

    }
}
```

2.3、配置 CharacterEncodingFilter

在 web.xml 配置此过滤器，如下：

```
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CharacterEncodingFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3、字符编码可以配置

在过滤器中，我们将请求参数的编码设值为 UTF-8，此时编码是写死在 Java 代码中的，若后期需要修改，那么这个 UTF-8 就是硬编码，不可以灵活配置。

3.1、配置字符编码

Filter 类似 Servlet 一样，也可以配置初始化参数。在 web.xml 配置如下：

```
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.2、修改 CharacterEncodingFilter.java

在初始化方法里从 FilterConfig 对象中获取配置的 encoding 的初始化参数值。

```
package cn.wolfcode.web.filter;

public class CharacterEncodingFilter implements Filter {

    private String encoding;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.encoding = filterConfig.getInitParameter("encoding");
```

```
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        if(StringUtil.hasLength(encoding) {
            // 设置请求编码
            servletRequest.setCharacterEncoding(this.encoding);
        }
        // 放行
        filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {

    }
}
```

4、字符编码覆盖设置

因为可能存在多个过滤器且都设置字符编码，可以让开发者能够选择是否覆盖之前的字符编码，这种设计才是最灵活的。

4.1、配置字符编码是否覆盖

在 web.xml 配置如下：

```
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>force</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

4.2、修改 characterEncodingFilter.java

在初始化方法里从 FilterConfig 对象中获取配置的 encoding 的初始化参数值。

```
package cn.wolfcode.web.filter;

public class CharacterEncodingFilter implements Filter {

    private String encoding;
    private boolean force;
```

```
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.encoding = filterConfig.getInitParameter("encoding");
        this.force = Boolean.valueOf(filterConfig.getInitParameter("force"));
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        if(StringUtil.hasLength(encoding)) {
            // 若配置强制或者之前设置过请求字符编码的话
            if(force || servletRequest.getCharacterEncoding() == null) {
                servletRequest.setCharacterEncoding(this.encoding);
            }
        }
        filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {

    }
}
```

六、登录校验过滤器

1、登录校验过滤器引入

在实际开发中，我们项目中会存在很多的资源都需要在登录之后才能访问，所以我们需要在请求到达这些资源之前先判断当前用户是否登录，如果没有应该跳转到登录页面。之前我们是在需要登录访问的的资源加上登录判断代码，代码如下：

```
Object obj = req.getSession().getAttribute("USER_IN_SESSION");
if(obj == null){
    resp.sendRedirect("/login.jsp");
    return;
}
```

以上操作能够解决登录检查的需求，但存在大量重复代码，在每个资源的代码中都会编写检查登录的代码，从增加维护成本。

目标：一处设置，到处有效（设置一次，所有 Servlet 共用）。
方案：访问到 Servlet 之前对请求进行登录校验处理（Filter）。

2、代码实现

2.1、修改所有 Servlet

删除其中对请求登录校验处理的代码。

2.2、编写 CheckLoginFilter.java

过滤器请求，做登录校验。

```
package cn.wolfcode.web.filter;

public class CheckLoginFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        // 把请求和响应转化为符合 HTTP 协议的对象
        HttpServletRequest req = ((HttpServletRequest) servletRequest);
        HttpServletResponse resp = ((HttpServletResponse) servletResponse);

        Object obj = req.getSession().getAttribute("USER_IN_SESSION");
        // 没有登录，重定向到登录页面
        if(obj == null) {
            resp.sendRedirect("/login.jsp");
            return;
        }
        // 登录过，放行访问
        filterChain.doFilter(req, resp);
    }

    @Override
    public void destroy() {

    }
}
```

2.3、配置 CheckLoginFilter

在 web.xml 配置此过滤器，如下：

```
<filter>
    <filter-name>checkLoginFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CheckLoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>checkLoginFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2.4、匿名访问资源

匿名访问资源指的是不需要登录也可以访问的资源，比如 /login.jsp，/login，/randomCode 和静态资源等等。若不排除，那么当浏览器中请求这些资源，也需要登录，这就会造成这些资源访问不到了。

针对上面的情况，只需要指定 CheckLoginFilter 对哪些资源做登录校验处理或者不对哪些资源做登录校验处理。

2.4.1、方式一

指定 CheckLoginFilter 不对哪些资源做登录校验处理。

修改 CheckLoginFilter，配置初始化参数，指定哪些资源不做登录校验，如下：

```
<filter>
    <filter-name>checkLoginFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CheckLoginFilter</filter-class>
    <init-param>
        <param-name>unCheckUri</param-name>
        <param-value>/login.jsp;/login;/randomCode</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>checkLoginFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

```
package cn.wolfcode.web.filter;

public class CheckLoginFilter implements Filter {

    private List<String> unCheckUriList;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.unCheckUriList =
Arrays.asList(filterConfig.getInitParameter("unCheckUri").split(";"));
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest req = ((HttpServletRequest) servletRequest);
        HttpServletResponse resp = ((HttpServletResponse) servletResponse);

        Object obj = req.getSession().getAttribute("USER_IN_SESSION");
        // 获取请求的资源路径
        String uri = req.getRequestURI();
        // 没有登录且访问的不是匿名资源，重定向到登录页面
        if(obj == null && !unCheckUriList.contains(uri)) {
            resp.sendRedirect("/login.jsp");
            return;
        }
        // 登录过，放行访问
        filterChain.doFilter(req, resp);
    }

    @Override
    public void destroy() {

    }
}
```

2.4.1、方式二

因为会发现，上面那样处理还是会导致静态资源不可以访问到，而实际开发中，我们需要排除的资源较多的话，尤其是像静态资源，会很麻烦。所以这次指定 CheckLoginFilter 对哪些资源做登录校验处理。将需要受检查的资源通通存放到 check 路径下。

修改 CheckLoginFilter 的过滤路径，如下：

```
<filter>
    <filter-name>checkLoginFilter</filter-name>
    <filter-class>cn.wolfcode.web.filter.CheckLoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>checkLoginFilter</filter-name>
    <!-- 只有在 check 路径下的资源访问会做登录的校验 -->
    <url-pattern>/check/*</url-pattern>
</filter-mapping>
```

修改 CheckLoginFilter，如下：

```
package cn.wolfcode.web.filter;

public class CheckLoginFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest req = ((HttpServletRequest) servletRequest);
        HttpServletResponse resp = ((HttpServletResponse) servletResponse);

        Object obj = req.getSession().getAttribute("USER_IN_SESSION");
        // 没有登录，重定向到登录页面
        if(obj == null) {
            resp.sendRedirect("/login.jsp");
            return;
        }
        // 登录过，放行访问
        filterChain.doFilter(req, resp);
    }

    @Override
    public void destroy() {

    }
}
```

最后只要修改需要做登录校验的资源，加上 /check 路径即可（记得把 list.jsp 查询产品的路径和登录成功重定向的的路径改了），如下：

```
@WebServlet("/check/product")
public class ProductServlet extends HttpServlet {
    // 省略......
}
```

七、监听器 Listener

Listener 是 Java Web 组件之一，主要的作用是用于**监听作用域对象**的创建和销毁动作以及**作用域属性值**的改变动作。若用户触发了这些动作，那么会立即执行相应的的监听器的操作。

1、监听什么及分类

监听的对象：作用域对象，作用域属性。
监听的动作：作用域对象的创建和销毁，作用域属值的增删改。

监听器分类

- 按作用域对象：
  - ServletRequestListener
  - HttpSessionListener
  - ServletContextListener
- 按作用域属性分：
  - ServletRequestAttributeListener
  - HttpSessionAttributeListener
  - ServletContextAttributeListener

2、开发监听器的步骤

- 创建一个类，根据需求实现对应的接口。
- 实现其中的方法。
- 将监听器交给 Tomcat 管理。

3、统计游客数量

案例：统计游客数量。

3.1、编写 VisitorListener.java

```
package cn.wolfcode.web.listener;

public class VisitorListener implements HttpSessionListener {

    private static long totalCount = 0;

    // Session 对象创建会执行下面的方法
    @Override
    public void sessionCreated(HttpSessionEvent httpSessionEvent) {
        totalCount++;
        System.out.println("在线人数：" + totalCount);
    }

    // Session 对象销毁会执行下面的方法
    @Override
    public void sessionDestroyed(HttpSessionEvent httpSessionEvent) {
        totalCount--;
    }
}
```

3.2、配置 VisitorListener

可以使用注解配置（直接在自己写监听器类上贴 @WebListener 即可），也可以是 XML 配置，XML 配置如下：

```
<listener>
    <listener-class>cn.wolfcode.web.listener.VisitorListener</listener-class>
</listener>
```

4、创建系统默认管理员

一般系统最开始部署到用户的服务器时，需要在用户的表中添加一个默认的超级管理员账户，再由该管理员来管理其他的账户的添删改查操作。

4.1、思考的问题

- 超级管理员什么时候创建？服务器启动的时候创建。
- 哪些技术可以在服务器启动做操作？
  答案：可以使用 Servlet 和 Filter 来实现，也可以使用 ServletContextListener 这个监听器来完成，这个监听器可以用来监听 ServletContext 对象的创建，而这个对象的创建就是在 Tomcat 启动的时候。

4.2、编写并配置 CreateSuperUserListener

在其 contextInitialized 方法中创建管理员账户：

- 查询用户表中是否存在管理员的账户。
- **存在则不做操作，不存在则创建账号。**

```
package cn.wolfcode.web.listener;

@WebListener
public class CreateSuperUserListener implements ServletContextListener {

    private IUserDAO userDAO = new UserDAOImpl();

    // 容器启动的时候会执行下面的方法
    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        // 判断是否存在超级管理员，存在说明不是第一次部署，不能再创建账号
        User user = userDAO.checkUser("admin");
        if(user == null) {
            // 保存管理员账号
            User adminUser = new User();
            adminUser.setUsername("admin");
            adminUser.setPassword("admin");
            userDAO.insert(adminUser);
        }
    }

    @Override
    public void contextDestroyed(ServletContextEvent servletContextEvent) {

    }
}
```

4.3、修改 IUserDAO 和 UserDAOImpl

```
void insert(User user);
```

```
@Override
public void insert(User user) {
    SqlSession session = MyBatisUtil.getSession();
    session.insert("cn.wolfcode.mapper.UserMapper.insert", user);
    session.commit();
    session.close();
}
```

4.4、修改 UserMapper.xml

```
<insert id="insert" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO user(username, password, headImg) VALUES(#{username}, #
{password}, #{headImg})
</insert>
```

八、页面美化（拓展）

操作步骤：

1. 拷贝前端写的页面

   根据给的目录把对应的文件拷贝到项目下，css js login.jsp common/input.jsp common/list.jsp
2. 修改 list.jsp 来显示列表的数据

   保留行中的编辑和修改按钮（带有样式），加入遍历到的货品数据
3. 加入高级查询的表单数据
4. 修改分页条的 totalPage 和 currentPage （common/page.jsp）
5. 加入 input.jsp -> 修改 ProductServlet 加入分发器，可以跳转到 input.jsp
6. 修改 input.jsp，一块一块表单控件的 div 拷贝修改即可
7. 把编辑删除，提交完成后台功能即可
8. 把用户头像修改加入项目中

练习

- 在之前 Web CRUD 项目基础上，提供用户注销和记住账号的功能。
- 在之前 Web CRUD 项目基础上，提供对请求编码过滤器和登录校验过滤器。
- 在之前 Web CRUD 项目基础上，完成在项目启动时初始化第一个超级管理员账号。