

# 过滤器 Filter

## 今日内容介绍

- ◆ 案例：自动登录
- ◆ 案例：统一 GET 和 POST 中文乱码的处理

## 今日内容学习目标

- ◆ 了解过滤器执行原理
- ◆ 独立编写过滤器
- ◆ 知道如何在过滤器中对 request 进行增强

## 第1章 案例：自动登录

### 1.1 案例介绍

在完成登录时，如果用户勾选“自动登录”，将在下次登录时，自动完成登录功能，减少用户再次输入账号和密码繁琐的操作。此功能是对用户的操作体验进行优化，本案例将带领大家完成此功能。效果图如下：

会员登录 USER LOGIN

用户名 jack

密码 \*\*\*\*\*

验证码 l C K x

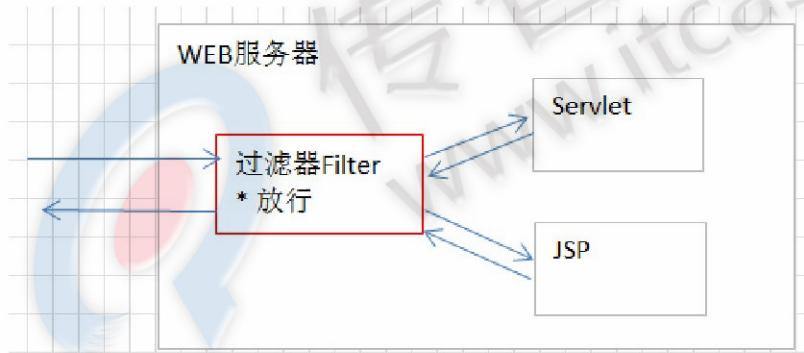
自动登录  记住用户名

登录

自动登陆：有程序帮助使用者，进行自动的登录。（输入用户名和密码，点击登录这个操作）

## 1.2 相关知识点：过滤器

- 什么是过滤器
  - 过滤器是一个运行在服务器端的程序，先于与之相关的 servlet 或 JSP 页面之前运行，实现对请求资源的过滤的功能。



登录：成功（session 作用域记录用户登录状态），失败（在 request 作用域记录用错误信息）

- 过滤器可附加到一个或多个 servlet 或 JSP 页面上，可以检查请求信息，也可以处理响应信息。
- Filter 的基本功能是对 Servlet 容器调用 Servlet 的过程进行拦截，从而在 Servlet 执行前后实现一些特殊的功能。

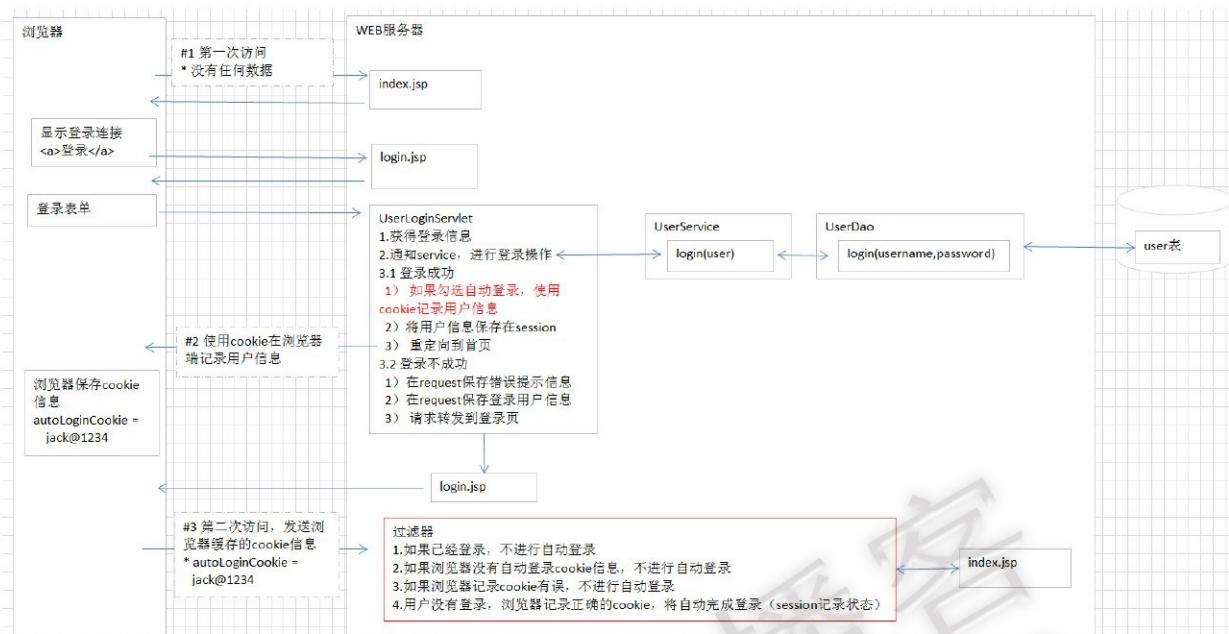
- 过滤器常用实例
  - 自动登录，解决全站乱码，屏蔽非法文字，进行响应数据压缩，等等
- 过滤器的编写流程
  - 实现类，需要实现接口 javax.servlet.Filter
  - 配置，在 web.xml 使用<filter>和<filter-mapping>进行配置

### ● 实例：

```
//实现类
```

```
public class HelloFilter implements Filter {  
    public void init(FilterConfig filterConfig) throws ServletException {  
    }  
  
    public void doFilter(ServletRequest request, ServletResponse response,  
FilterChain chain)  
        throws IOException, ServletException {  
        //放行  
        chain.doFilter(request, response);  
    }  
  
    public void destroy() {  
    }  
}  
  
//xml 配置  
<!-- 1 注册：通知 tomcat 过滤器实现类  
      filter-name: 给已经注册的过滤器进行唯一命名  
      filter-class: 过滤器实现类，全限定类名  
-->  
<filter>  
    <filter-name>HelloFilter</filter-name>  
    <filter-class>cn.itcast.demo01.HelloFilter</filter-class>  
</filter>  
<!-- 2 使用：确定对那些程序使用过滤器  
      filter-name:已经注册的过滤器名称  
      url-pattern: 需要对那些指定路径进行过滤 。  
      /* 表示所有  
-->  
<filter-mapping>  
    <filter-name>HelloFilter</filter-name>  
    <url-pattern>/*</url-pattern>  
</filter-mapping>
```

## 1.3 案例分析



用户登录成功后，如果用户勾选复选框，将使用 cookie 记录用户信息，cookie 的值格式“用户名@密码”。

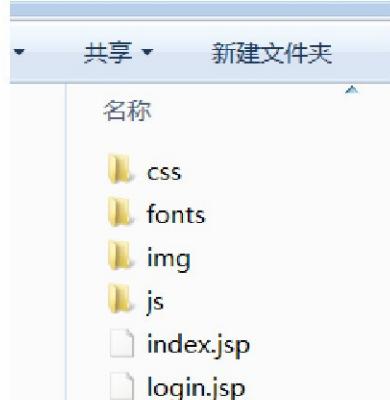
当用户第二次访问首页时，编写过滤器处理浏览器 cookie 记录的用户信息。

## 1.4 案例实现

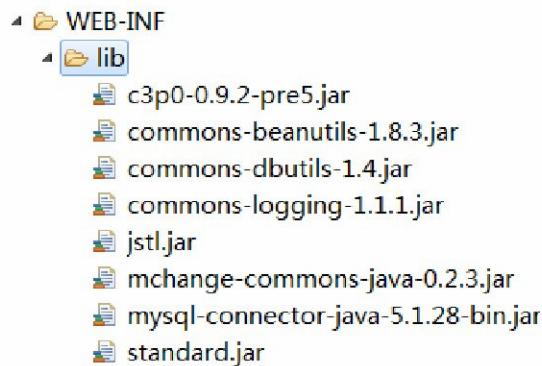
### 1.4.1 环境搭建

- 步骤 1：创建项目“day24\_auto\_login”
- 步骤 2：复制 html 代码

\day24--过滤器\资料\01.用户登录



- 步骤 3：拷贝 jar 包



● 步骤 4: 拷贝配置文件和工具类



```
<!-- 连接数据库的 4 项基本参数 -->
<property name="driverClass">com.mysql.jdbc.Driver</property>
<property name="jdbcUrl">jdbc:mysql://127.0.0.1:3306/day24_db</property>
<property name="user">root</property>
<property name="password">1234</property>
```

● 步骤 4: 数据库和表

```
#创建数据库
create database day24_db;
#使用数据库
use day24_db;
### 用户表
CREATE TABLE `user` (
    `uid` varchar(32) PRIMARY KEY,
    `username` VARCHAR(20) ,
    `password` VARCHAR(20) ,
    `gender` VARCHAR(10) , #性别
    `telephone` VARCHAR(20) ,
    `email` VARCHAR(50) ,
    `introduce` VARCHAR(100), #自我介绍
    `activeCode` VARCHAR(50) , #激活码
    `state` INT(11) , #激活状态 0:未激活 1:激活
    `role` VARCHAR(10) DEFAULT '普通用户' , #角色（权限管理）
    `registTime` DATETIME
);

insert into user(uid,username,password) values('u001','jack','1234');
insert into user(uid,username,password) values('u002','rose','1234');
insert into user(uid,username,password) values('u003','张三','1234');
```

● 步骤 5: 创建 JavaBean

```
public class User {
```

```
private String uid;
private String username;
private String password;
private String gender;
private String telephone;
private String email;
private String introduce;
private String activeCode;
private String state;
private String role;
private String registTime;
```

## 1.4.2 登录

- 步骤 1：编写 dao，提供 `find(username,password)`方法，通过用户名和密码查询用户

```
/*
 * 通过用户名和密码查询用户
 * @param username
 * @param password
 * @return
 */
public User find(String username, String password) {
    try {
        //1 核心类
        QueryRunner queryRunner = new QueryRunner(C3P0Utils.getDataSource());
        //2 准备sql语句
        String sql = "select * from user where username = ? and password = ?";
        //3 准备参数
        Object[] params = {username, password};
        //4 执行
        return queryRunner.query(sql, new BeanHandler<User>(User.class), params);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

- 步骤 2：编写 service，提供 `login(User)`进行用户登录

```
public class UserService {

    private UserDao userDao = new UserDao();
    /**
     * 用户登录
     * @param user
```

```
* @return
*/
public User login(User user){
    return userDao.find(user.getUsername(), user.getPassword());
}
```

● 步骤 3：编写 jsp 表单

```
<form action="${pageContext.request.contextPath}/userLoginServlet" method="post" >
    <input type="text" name="username" placeholder="请输入用户名">
    <input type="password" name="password" placeholder="请输入密码">
    <input type="checkbox" name="autoLogin"> 自动登录
    <input type="submit" width="100" value="登录">
```

● 步骤 4：编写 servlet 实现类，完成登录

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    //0 编码
    request.setCharacterEncoding("UTF-8");

    //1 获得数据，并封装
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    User user = new User(username, password);

    //2 用户登录
    UserService userService = new UserService();
    User loginUser = userService.login(user);

    //3 处理信息
    if(loginUser != null) {
        // 3.1 用户登陆成功
        // * 将用户信息保存在 session 作用域，记录登录状态
        request.getSession().setAttribute("loginUser", loginUser);
        // * 重定向到首页
        response.sendRedirect(request.getContextPath() + "/index.jsp");
    } else {
        // 3.2 登录不成功
        // * 用户错误提示信息
        request.setAttribute("msg", "用户名和密码不匹配");
        // * 请求转发到登录页
        request.getRequestDispatcher("/login.jsp").forward(request, response);
    }
}
```

● 步骤 5：编写 servlet xml 配置

```
<servlet>
    <servlet-name>UserLoginServlet</servlet-name>
    <servlet-class>cn.itcast.web.servlet.UserLoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>UserLoginServlet</servlet-name>
    <url-pattern>/userLoginServlet</url-pattern>
</servlet-mapping>
```

● 步骤 6：完善登陆页 login.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${not empty msg}">
    <div class="form-group">
        <label for="username" class="col-sm-2 control-label"></label>
        <div class="col-sm-10">
            <font style="color: #f00; font-weight: bold;">${msg}</font>
        </div>
    </div>
</c:if>
```

login.jsp

```
133
134<c:if test="${not empty msg}"> 如果“错误提示信息”不为空，则显示
135    <div class="form-group">
136        <label for="username" class="col-sm-2 control-label"></label>
137        <div class="col-sm-10">
138            <font style="color: #f00; font-weight: bold;">${msg}</font>
139        </div>
140    </div>
141</c:if>
```

会员登录 USER LOGIN

用户名和密码不匹配

用户名 jack

● 步骤 7：完善 首页 index.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${empty loginUser}">
    <a href="login.jsp">登录</a>
</c:if>
<c:if test="${not empty loginUser}">
    ${loginUser.username} ,
    <a href="#">退出</a>
</c:if>
```

- 步骤 8：登录不成功，标签回显。将用户填写的信息存放在 request 作用域，jsp 标签的 value 手动的填写内容。

```

UserLoginServlet.java
37     response.sendRedirect(request.getContextPath() + "/index.jsp"
38 } else {
39     // 3.2 登录不成功
40     // * 用户错误提示信息
41     request.setAttribute("msg", "用户名和密码不匹配");
42     // * 存放用户填写的登录信息
43     request.setAttribute("user", user);
44     // * 请求转发到登录页
45     request.getRequestDispatcher("/login.jsp").forward(request, r
AE

<input name="username" placeholder="请输入用户名" value="${user.username}">
<input name="password" placeholder="请输入密码" value="${user.password}">
效果

```



### 1.4.3 自动登录

- 步骤 0：复制 “day24\_auto\_login”，重名为 “day24\_auto\_login2”

- 步骤 1：确定登录表单复选框 login.jsp

```
<input type="checkbox" name="autoLogin"> 自动登录
```

- 步骤 2：完善 servlet，在登录成功后，判断是否勾选自动登录复选框，如果勾选 cookie 记录登录信息

```

/**自动登录 start*/
// #1 获得复选框，如果没有设置 value 值，默认获得 on
String autoLogin = request.getParameter("autoLogin");
if(autoLogin != null){
    //#2 使用 cookie 记录用户信息，使用@拼凑
    Cookie cookie = new Cookie("autoLoginCookie", username + "@" + password);
    cookie.setPath("/");
    cookie.setMaxAge(60*60);
    response.addCookie(cookie);
}
/**自动登录 end*/

```



```

31 //3 处理信息
32 if(loginUser != null) {
33     /**自动登录start*/
34     // #1 获得复选框, 如果没有设置value值, 默认获得on
35     String autoLogin = request.getParameter("autoLogin");
36     if(autoLogin != null){
37         //#2 使用cookie记录用户信息, 使用@拼凑
38         Cookie cookie = new Cookie("autoLoginCookie", username + "@" + password);
39         cookie.setPath("/");
40         cookie.setMaxAge(60*60);
41         response.addCookie(cookie);
42     }
43     /**自动登录end*/
44     // 3.1 用户登陆成功
45 }

```

- 步骤 3：编写过滤器实现类，在下一次访问时，进行自动登录

1. 如果已经登录，不进行自动登录
2. 如果浏览器没有自动登录 cookie 信息，不进行自动登录
3. 如果浏览器记录 cookie 有误，不进行自动登录
4. 用户没有登录，浏览器记录正确的 cookie，将自动完成登录（session 记录状态）

```

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
throws IOException, ServletException {
    //0 强转
    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) resp;

    //1 用户登录信息
    User loginUser = (User) request.getSession().getAttribute("loginUser");

    //2 如果已经登录，放行，不需要自动登录
    if(loginUser != null){
        chain.doFilter(request, response);
        return; //程序结束
    }

    //3 获得 自动登录 cookie 信息
    Cookie[] allCookie = request.getCookies();
    Cookie userCookie = null;
    if(allCookie != null){
        for (Cookie c : allCookie) {
            if("autoLoginCookie".equals(c.getName())){
                userCookie = c;
                break;
            }
        }
    }
}

```

```
        }

    }

    //4 判断自动登录 cookie 是否存在,如果没有 cookie, 不需要自动
    if(userCookie == null){
        chain.doFilter(request, response);
        return;
    }

    //5 通过用户 cookie 中记录信息, 查询用户
    // 5.1 获得用户信息
    String[] u = userCookie.getValue().split("@");
    String username = u[0];
    String password = u[1];
    User user = new User(username, password);
    // 5.2 执行登录
    UserService userServic = new UserService();
    loginUser = userServic.login(user);

    //6 如果没有查询 (修改密码)
    if(loginUser == null){
        chain.doFilter(request, response);
        return ;
    }

    // 7 自动登录
    request.getSession().setAttribute("loginUser", loginUser);
    //放行
    chain.doFilter(request, response);
}
```

#### ● 步骤 4：编写过滤器 xml 配置

```
<!-- 登录拦截器 start -->
<filter>
    <filter-name>loginFilter</filter-name>
    <filter-class>cn.itcast.web.filter.LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>loginFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- 登录拦截器 end -->
```

## 1.5 总结

- Filter 生命周期：过滤器从创建到销毁的过程

- 服务器启动的时候，服务器就会创建过滤器的对象，每次访问被拦截目标资源，过滤器中的 doFilter 的方法就会执行。当服务器关闭的时候，服务器就会销毁 Filter 对象。
- 服务器在启动时执行初始化方法， init
- 访问资源被拦截时执行拦截方法， doFilter 。放行： chain.doFilter(request,response)
- 服务器关闭时执行销毁方法， destroy

```
Filter
  ● init(FilterConfig) : void
  ● doFilter(ServletRequest, ServletResponse, FilterChain) : void
  ● destroy() : void
```

- FilterConfig 对象

```
FilterConfig
  ● getFilterName() : String
  ● getServletContext() : ServletContext
  ● getInitParameter(String) : String
  ● getInitParameterNames() : Enumeration<String>
```

```
// 获得初始化参数：过滤器的初始化参数。
```

```
String username = fConfig.getInitParameter("username");
String password = fConfig.getInitParameter("password");
System.out.println("初始化参数"+username+" "+password);
```

```
// 获得所有的初始化参数的名称：
```

```
Enumeration<String> names = fConfig.getInitParameterNames();
while(names.hasMoreElements()){
    String name = names.nextElement();
    String value = fConfig.getInitParameter(name);
    System.out.println(name+" "+value);
}
```

```
// 获得过滤器的配置的名称：
```

```
String filterName = fConfig.getFilterName();
System.out.println("过滤器名称"+filterName);
```

- FilterChain：过滤器链

过滤器链中的过滤器的执行的顺序跟<filter-mapping>的配置顺序有关

```
FilterChain
  ● doFilter(ServletRequest, ServletResponse) : void
```

- 过滤器的配置

- url-pattern 的配置与 servlet 中的配置一样：

\* 三种配置：

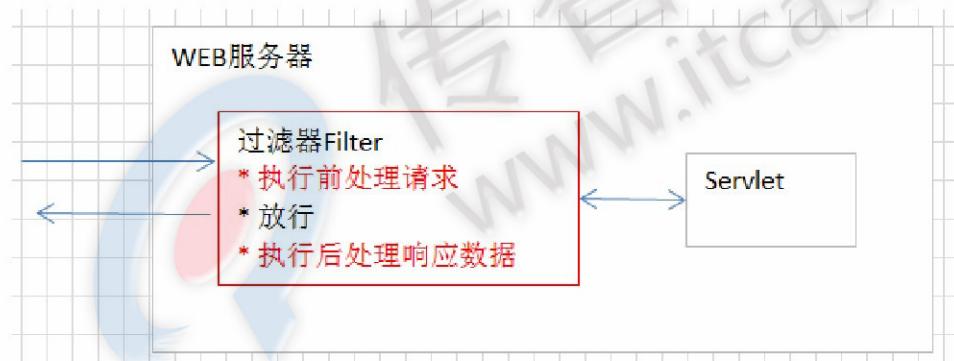
- \* 完全路径匹配：以 / 开始 /aaa /aaa/bbb
- \* 目录匹配：以 / 开始 /\* /aaa/\*

- \* 扩展名匹配：不能以 / 开始 \*.do \*.jsp \*.action
- servlet-name 的配置，通过 url-pattern 拦截一个 Servlet 的资源.也可以通过 servlet-name 标签进行拦截.
- dispatcher 的配置
  - \* REQUEST : 默认值.
  - \* FORWARD : 拦截转发
  - \* ERROR : 拦截跳转到错误页面.全局错误页面.
  - \* INCLUDE : 拦截在一个页面中包含另一个页面.

## 第2章 案例：GET和POST统一编码

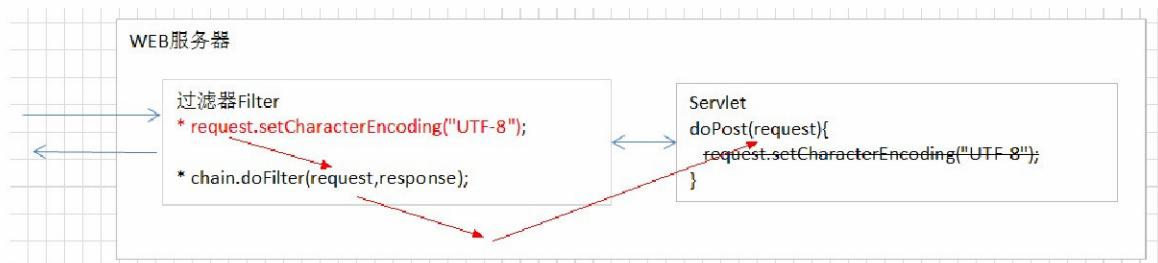
### 2.1 案例介绍

在完成功能时，我们发现 UserLoginServlet 有一行代码，在之前编写的 servlet 都出现了，那就是编码处理，在实际开发中，需要统一处理编码。且之前的程序 GET 请求需要单独处理，本案例将以上两个问题使用过滤器统一解决。也就是在 servlet 前后执行特定功能。



### 2.2 POST 请求方式编码处理

#### 2.2.1 案例分析



- 将 **Servlet** 中处理请求编码的语句，编写到过滤器放行语句之前。

## 2.2.2 案例实现

- 步骤 1：编写实现，在放行之前设置编码

```
public class EncodingFilter implements Filter {  
    public void init(FilterConfig fConfig) throws ServletException {  
    }  
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain  
chain)  
        throws IOException, ServletException {  
        //0 强转  
        HttpServletRequest request = (HttpServletRequest) req;  
        HttpServletResponse response = (HttpServletResponse) resp;  
  
        //1 设置编码  
        request.setCharacterEncoding("UTF-8");  
  
        //2 放行  
        chain.doFilter(request, response);  
    }  
    public void destroy() {  
    }  
}
```

- 步骤 2：web.xml 配置过滤器

```
<filter>  
    <filter-name>EncodingFilter</filter-name>  
    <filter-class>cn.itcast.web.filter.EncodingFilter</filter-class>  
</filter>  
<filter-mapping>  
    <filter-name>EncodingFilter</filter-name>  
    <url-pattern>/*</url-pattern>  
</filter-mapping>
```

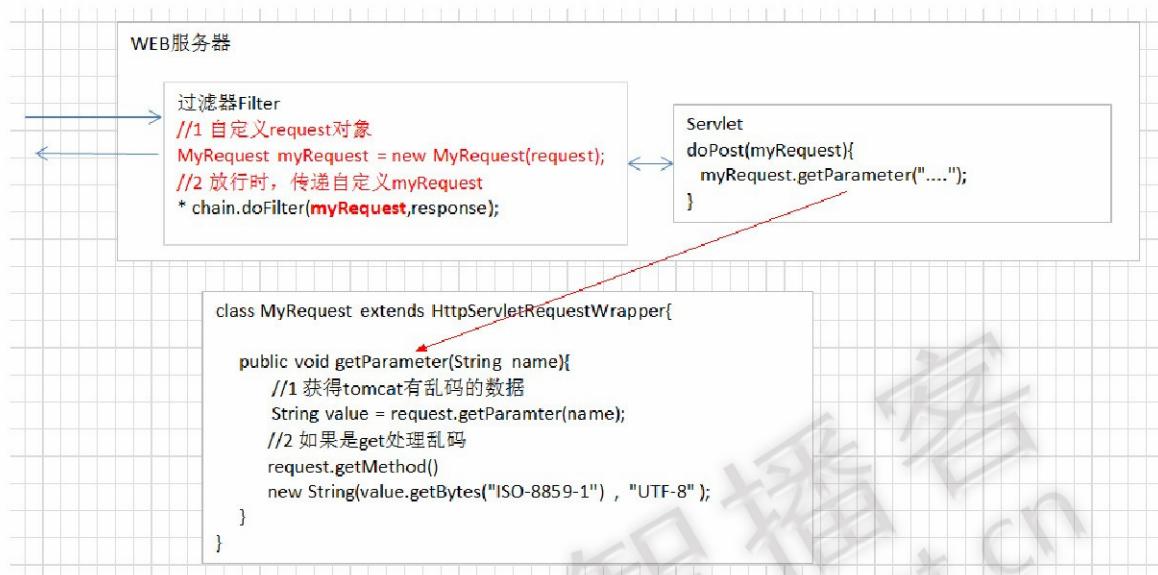
## 2.3 GET 请求方式编码处理

### 2.3.1 案例相关知识

编写 Request 装饰者实现类，理论需要实现 **HttpServletRequest** 接口，然后提供成员变量和构造方法，对接口中所有的方法进行处理（增强或不增强），但 **HttpServletRequest** 接口中方法过多，使

操作虽然简单，但比较繁琐。JavaEE 规范提供了 `HttpServletRequest` 接口的装饰者实现基类 `HttpServletRequestWrapper`，及按照装饰者编写实现类，所有方法都不增强。我们如果需要对方法进行增强，只需要继承该类，然后复写对应方法即可。

### 2.3.2 案例分析



### 2.3.3 案例实现

- 步骤 1：使用装饰者设计模式，编写自定义 Request 对象。

```

public class MyRequest extends HttpServletRequestWrapper {

    public MyRequest(HttpServletRequest request) {
        super(request);
    }

    /**
     * 获得指定名称的第一个参数
     */
    @Override
    public String getParameter(String name) {
        String value = super.getParameter(name);
        try {
            if ("GET".equalsIgnoreCase(super.getMethod())) {
                value = new String(value.getBytes("ISO-8859-1"), "UTF-8");
            }
        } catch (Exception e) {

```

```
        throw new RuntimeException(e);
    }
    return value;
}

}
```

- 步骤 2：修改过滤器，使用自定义 request 实现类

```
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
    throws IOException, ServletException {
    //0 强转
    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) resp;

    //1 设置编码
    request.setCharacterEncoding("UTF-8");

    //2 创建自定义 request
    MyRequest myRequest = new MyRequest(request);

    //3 放行，使用自定义 request
    chain.doFilter(myRequest, response);
}
```

- 步骤 3：修改 servlet，删除编码处理代码(注释掉)

```
//0 编码
//request.setCharacterEncoding("UTF-8");
```

- 步骤 4：测试，修改表单提交方式

```
<%--登录表单 start --%>
<form action="${pageContext.request.contextPath}/userLoginServlet" method="get" >
```

## 第3章 总结