

【济南中心】JavaEE就业班同步笔记第一阶段：动态代理

小鲁哥哥 • 2017-3-26 20:39:29

**【济南中心】JavaEE就业班同步笔记第一阶段：
JavaWeb之基础加强--动态代理****1. 案例三：使用动态代理的方式统一网站的字符集编码****1.1 需求：**

在一个表单中分别使用get和post提交到Servlet中，在Servlet中直接调用getParameter方法解决中文乱码的问题！！

1.2 分析：**1.2.1 技术分析：****【动态代理】**

增强一个类中的某个方法.对程序进行扩展.Spring框架中AOP.

什么是代理：

【入门案例】

```
[mw_shl_code=java,true]class MyInvocationHandler implements InvocationHandler{
    private Waiter waiter;
    public MyInvocationHandler(Waiter waiter) {
        this.waiter = waiter;
    }
    @Override
    // 执行目标对象的任何一个方法 都相当于执行了invoke方法.
    /**
```

回帖

* Object[]:在调用的方法的参数.

*/

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    // System.out.println("aaaaa");
```

```

        // System.out.println(method.getName());
        if("server".equals(method.getName())){
            // 增强server.
            System.out.println("微笑...");
            return method.invoke(waiter, args);
        }else{
            // 不增强:
            return method.invoke(waiter, args);
        }
        // return null;
    }
}[/mw_shl_code]

```

1.3 代码实现:

```

[mw_shl_code=java,true]package com.itheima.encoding;
import java.io.IOException;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

```

```

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

```

```

@WebFilter(urlPatterns={"/*"})
public class GenericCharacterEncodingFilter implements Filter{

```

```

    @Override

```

```

public void init(FilterConfig filterConfig) throws ServletException {
}
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain
)
    throws IOException, ServletException {
    final HttpServletRequest req = (HttpServletRequest) request;
    // 对req产生代理对象:
    HttpServletRequest myReq = (HttpServletRequest) Proxy.newProxyInstance(req.getCl
ass().getClassLoader(),req.getClass().getInterfaces(), new InvocationHandler() {

        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
        {
            // 增强getParameter:
            if("getParameter".equals(method.getName())){
                // 增强.
                // 根据请求方式:
                String type = req.getMethod();
                if("get".equalsIgnoreCase(type)){
                    // 调用原有的getParameter:
                    String value = (String)method.invoke(req, args);
                    String s = new String(value.getBytes("ISO-8859-1"),"UTF-8");
                    return s;

                }else if("post".equalsIgnoreCase(type)){
                    req.setCharacterEncoding("UTF-8");
                    return method.invoke(req, args);
                }
            }

            // 不增强:

```

```

        return method.invoke(req, args);
    }
});
chain.doFilter(myReq, response);
}
@Override
public void destroy() {
}
}[/mw_shl_code]

```

1.4 总结:

1.4.1 【类的加载器: 了解】

类加载器就是将class文件加载到内存.

JDK中提供的类加载器:

- * 引导/系统类加载器 : Java\jre7\lib\rt.jar
- * 扩展类加载器 : Java\jre7\lib\ext*.jar
- * 应用类加载器 : 自定义的类,类路径下的所有class文件.

类的加载器的机制 : 全盘委托机制.

引导类加载器

|

扩展类加载器

|

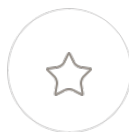
应用类加载器

```

[mw_shl_code=java,true]class A{
    String s;
}[/mw_shl_code]

```

class文件由应用类加载器获得到,没有加载,向上一层委托向扩展类加载器委托,向上一层进行委托委托给引导类加载器.引导类加载器查看class哪些它负责,将自己负责的这个class进行加载.不是其负责的就向下传递扩展类加载器.扩展类加载器查看是否是其管理的class,如果是加载,不是就再向下到应用类加载器.



回帖

— 2条回帖 —



zhouxiaoyang

这么赞6666

沙发 • 2017-3-26 20:57:55

回帖



来自宇宙超级黑马专属苹果客户端



liujinlong666

谢谢分享!!

藤椅 • 2017-6-1 00:05:25

回帖