(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 106599156 B
(45)授权公告日 2019.05.21

(21)申请号 201611118032.7

(22)申请日 2016.12.07

(65)同一申请的已公布的文献号
申请公布号 CN 106599156 A

(43)申请公布日 2017.04.26

(73)专利权人 国云科技股份有限公司
地址 523808 广东省东莞市松山湖高新技术产业开发区科汇路1号中科院云计算中心19楼

(72)发明人 钟健波 杨松 季统凯

(74)专利代理机构 广东莞信律师事务所 44332
代理人 余伦

(51)Int.Cl.
G06F 16/958(2019.01)
G06F 9/445(2018.01)

(56)对比文件
CN 106055385 A,2016.10.26,
CN 102567534 A,2012.07.11,
CN 104270403 A,2015.01.07,
CN 103150155 B,2016.08.03,
US 2013055242 A1,2013.02.28,

审查员 庞丽金

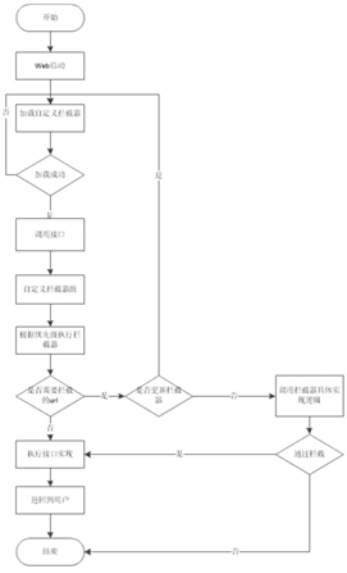
权利要求书1页 说明书5页 附图1页

(54)发明名称

一种基于struts2拦截器的动态拦截器管理方法

(57)摘要

本发明涉及web项目领域,特别是一种基于struts2拦截器的动态拦截器管理方法。本发明首先创建一个struts2拦截器,用于执行自定义的拦截器组;拦截器组是自定义的一个具有多种属性的抽象类,包括优先级,拦截url;然后创建加载器模块,用于动态加载web项目,并在线动态更新拦截器组中的拦截器。通过自定义的动态拦截器,可以在线升级web项目中的拦截模块,提供web项目安全性和提供服务稳定性;方便扩展,提高开发人员的开发效率。



CN 106599156 B

1.一种基于struts2拦截器的动态拦截器管理方法,其特征在于:所述的方法具体包括以下步骤:

步骤1:创建一个struts2拦截器,并配置好对应的struts配置文件,web启动时,拦截器能正常运行;

步骤2:创建自定义的拦截器抽象类;

步骤3:根据步骤2的抽象类,实现具体的拦截逻辑;

步骤4:创建加载模块,自动扫描web项目下特定包路径下的文件加载到内存;

步骤5:启动web项目,加载模块加载所有自定义拦截器到内存中;

步骤6:调用web应用接口,拦截器的调用模块获取内存中所有自定义拦截器,并根据优先级排序;

步骤7:循环每个拦截器,根据接口url判断是否匹配自定义拦截器url,若匹配,执行步骤8,否则执行步骤9;

步骤8:执行拦截器的拦截逻辑,若执行通过,返回true,否则抛错;

步骤9:执行原有接口逻辑,返回给用户;

步骤10:判断接口是否更新拦截器,若是执行步骤11,否则,执行步骤12;

步骤11:加载模块扫描指定包路径下文件,加载到内存中;

步骤12:结束。

2.根据权利要求1所述的方法,其特征在于:所述的拦截逻辑是当需要拦截的接口url为*时拦截所有的接口。

3.根据权利要求1所述的方法,其特征在于:所述的加载模块在web项目启动时执行,扫描指定包路径下的代码文件,判断是否自定义拦截器的实现类,若是则添加到内存中,否则,继续扫描;

首次加载完后,接口调用更新时,调用加载模块,对特定包路径代码文件扫描更新;更新前不会影响原来在内存中的拦截器;在更新时保证web应用能正常的运作、提供服务。

4.根据权利要求2所述的方法,其特征在于:所述的加载模块在web项目启动时执行,扫描指定包路径下的代码文件,判断是否自定义拦截器的实现类,若是则添加到内存中,否则,继续扫描;

首次加载完后,接口调用更新时,调用加载模块,对特定包路径代码文件扫描更新;更新前不会影响原来在内存中的拦截器;在更新时保证web应用能正常的运作、提供服务。

5.根据权利要求1至4任一项所述的方法,其特征在于:所述的优先级,是自定义的拦截器的属性,优先级为0,代表优先级最小,数字越大,优先级越大,先执行优先级高的拦截器。

6.根据权利要求1至4任一项所述的方法,其特征在于:拦截器中配置需要拦截的url,当匹配到正确的url才进行拦截,其余的直接返回;当配置为*时,拦截所有的url。

7.根据权利要求5所述的方法,其特征在于:拦截器中配置需要拦截的url,当匹配到正确的url才进行拦截,其余的直接返回;当配置为*时,拦截所有的url。

一种基于struts2拦截器的动态拦截器管理方法

技术领域

[0001] 本发明涉及web应用领域，特别是一种基于struts2拦截器的动态拦截器管理方法。

背景技术

[0002] Web应用中，需要对接口进行一些公共的操作，例如参数校验，权限控制等等。一般是通过新建一个拦截器进行统一的处理。拦截器的使用过程中还是会有遇到以下的一些问题：

[0003] 一是每增加一个拦截器，需要重新修改配置文件，增加对应的拦截器配置信息。

[0004] 二是增加了拦截器后需要重启web应用，拦截器才能正常的执行拦截的工作。

发明内容

[0005] 本发明解决的技术问题在于提供一种基于struts2拦截器的动态拦截器管理方法，解决了web应用在线更新，减少web应用的运维难度，提高web应用开发效率。

[0006] 本发明解决上述技术问题的技术方案是：

[0007] 所述的的方法具体包括以下步骤：

[0008] 步骤1:创建一个struts2拦截器，并配置好对应的struts配置文件，web启动时，拦截器能正常运行；

[0009] 步骤2:创建自定义的拦截器抽象类；

[0010] 步骤3:根据步骤2的抽象类，实现具体的拦截逻辑；

[0011] 步骤4:创建加载模块，自动扫描web项目下特定包路径下的文件加载到内存；

[0012] 步骤5:启动web项目，加载模块加载所有自定义拦截器到内存中；

[0013] 步骤6:调用web应用接口，拦截器的调用模块获取内存中所有自定义拦截器，并根据优先级排序；

[0014] 步骤7:循环每个拦截器，根据接口url判断是否匹配自定义拦截器url，若匹配，执行步骤8，否则执行步骤9；

[0015] 步骤8:执行拦截器逻辑，若执行通过，返回true，否则抛错；

[0016] 步骤9:执行原有接口逻辑，返回给用户；

[0017] 步骤10:判断接口是否更新拦截器，若是执行步骤11，否则，执行步骤12；

[0018] 步骤11:加载模块扫描指定包路径下文件，加载到内存中；

[0019] 步骤12:结束。

[0020] 所述的拦截逻辑是当需要拦截的接口url为*时拦截所有的接口。

[0021] 所述的加载模块在web项目启动时执行，扫描指定包路径下的代码文件，判断是否自定义拦截器的实现类，若是则添加到内存中，否则，继续扫描；

[0022] 首次加载完后，接口调用更新时，调用加载模块，对特定包路径代码文件扫描更新；更新前不会影响原来在内存中的拦截器；在更新时保证web应用能正常的运作、提供服

务。

[0023] 所述的优先级，是自定义的拦截器的属性，优先级为0，代表优先级最小，数字越大，优先级越大，先执行优先级高的拦截器。

[0024] 拦截器中配置需要拦截的url，当匹配到正确的url才进行拦截，其余的直接返回；当配置为*时，拦截所有的url。

[0025] 本发明解决了解决了web应用在线更新，解决了web应用在线更新，提高web应用开发效率。

附图说明

[0026] 下面结合附图对本发明进一步说明：

[0027] 图1为本发明方法拦截器工作流程图。

[0028] 具体实施方法

[0029] 如图1所示，本发明首先启动web应用，加载代码中拦截器实现类，实现类再进行工作：

[0030] 创建自定义的拦截器

```
public class MyInterceptor extends MethodFilterInterceptor {

    private static Logger LOG = Logger.getLogger(MyInterceptor.class);

    @Override
    protected String doIntercept(ActionInvocation arg0) throws Exception {
        // TODO Auto-generated method stub
        for (InterfaceInterceptor it : InterceptorManage.get()) {
            try {
                it.interceptor(arg0);
            } catch (Exception e) {
                LOG.error(e, e);
            }
        }
        String res = null;
```

[0031]

```
            try {
                res = arg0.invoke();
            } catch (Exception e) {
                LOG.error(e, e);
                res = errExceptionHandle(e);
            }
            return res;
```

[0032]

```
        }

        protected String errExceptionHandle(Exception ex) {
            return Action.SUCCESS;
        }

    }
```

[0033] 添加自定义的拦截器模板

```
    public class InterfaceInterceptor implements Comparator<InterfaceInterceptor> {
        public void interceptor(ActionInvocation arg0) {

        }

        public int getPriority() {
            return 0;
        }

        public String getUrl() {
            return "*";
        }
```

[0034]

```
            @Override
            public int compare(InterfaceInterceptor o1, InterfaceInterceptor o2) {
                // TODO Auto-generated method stub
                if (o1.getPriority() == o2.getPriority()) {
                    return 0;
                }

                if (o1.getPriority() < o2.getPriority()) {
                    return 1;
                }

                else {
                    return -1;
                }

            }
        }
```

[0035]

[0036] Web应用启动时候扫描特定包中实现代码添加到内存中

[0037]

```
private static List<InterfaceInterceptor> interceptor = new ArrayList<InterfaceInterceptor>();

    public static void registe(Class c) {
        try {
            if (InterfaceInterceptor.class.isAssignableFrom(c)) {
                InterfaceInterceptor it = (InterfaceInterceptor) c.newInstance();
                if (!interceptor.contains(it)) {
                    interceptor.add(it);
                    LOG.debug(String.format("register the request message observer:%s",
```

```
            it.getClass().getSimpleName()));
                        sort();
                    }
                }
            } catch (InstantiationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public static List<InterfaceInterceptor> get() {
            return interceptor;
        }

        public static void sort() {
            Collections.sort(interceptor, new InterfaceInterceptor());
        }。
```

[0038]

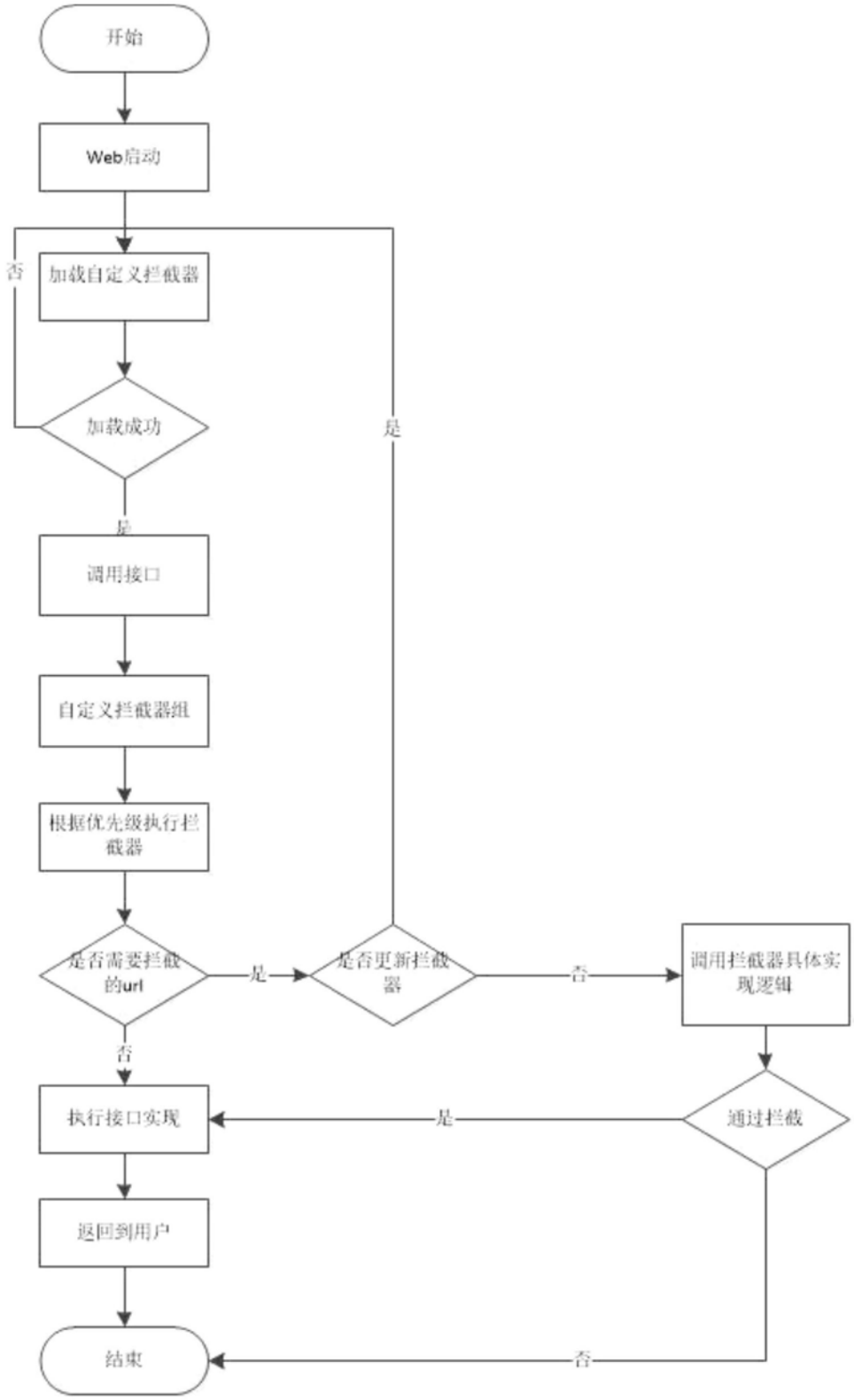

开始
Web启动
加载自定义拦截器
否
加载成功
是
是
调用接口
自定义拦截器组
根据优先级执行拦截器
是否需要拦截的url
是
是否更新拦截器
否
调用拦截器具体实现逻辑
否
执行接口实现
是
通过拦截
返回到用户
结束
否

图1