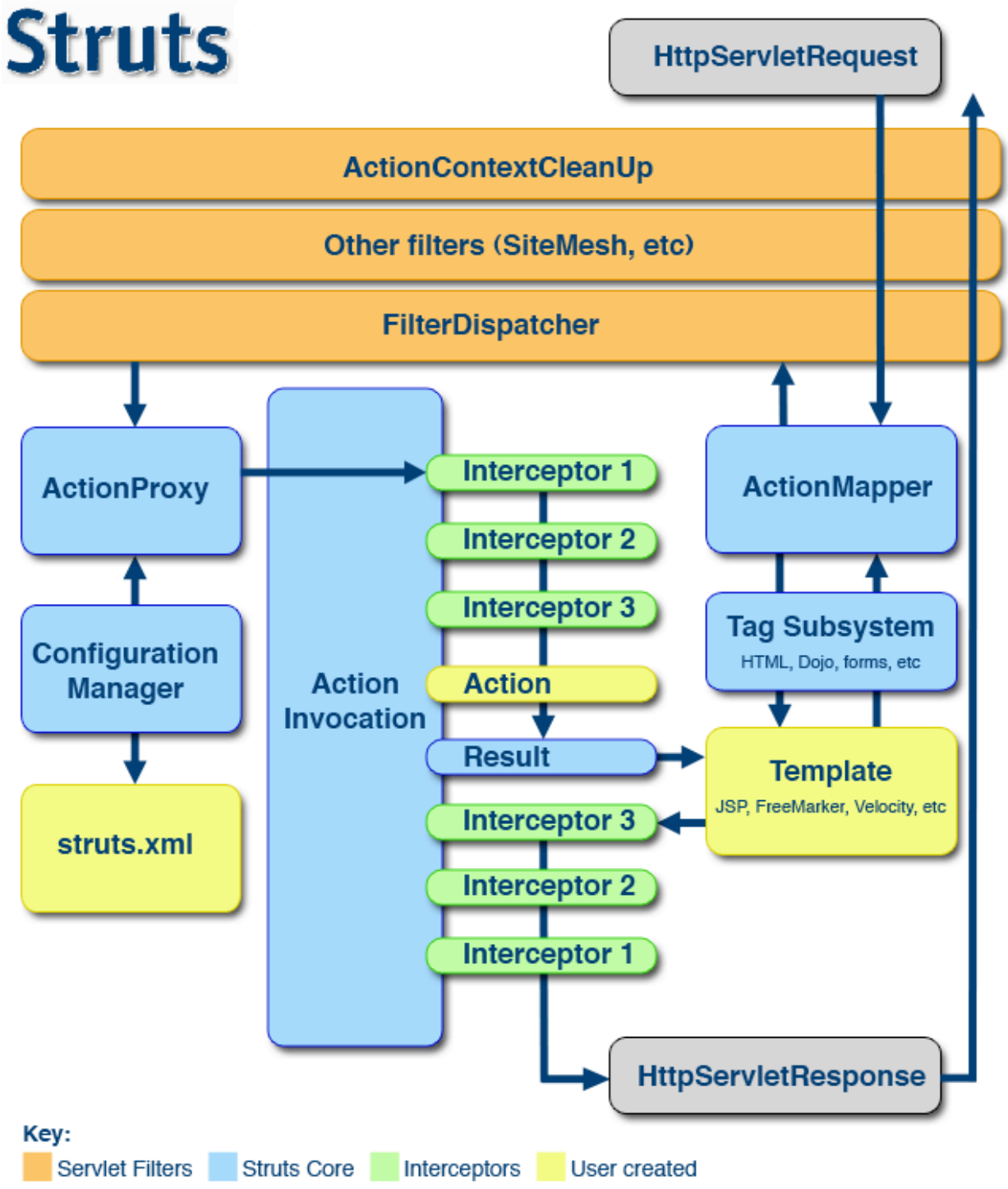
Struts

一 Struts 介绍

1.1 介绍

> Struts2是Struts1的下一代产品，是在 struts1和WebWork的技术基础上进行了合并的全新的Struts 2框架。
> 其全新的Struts 2的体系结构与Struts 1的体系结构差别巨大。
> Struts 2以WebWork为核心，采用拦截器的机制来处理用户的请求，这样的设计也使得业务逻辑控制器能够与ServletAPI完全脱离开，所以Struts 2可以理解为WebWork的更新产品。
> Struts2是一个基于MVC设计模式的Web层框架,通过类似于 web.xml 中地址映射的方式来快速定位到执行类,但是区别于 servlet，它的类可以是仁义类,不需要必须实现或者继承自某个类,他通过 filter 的方式进行路径拦截并分析转发

1.2 struts 执行流程



二 入门案例

2.1 依赖包

```
<!-- https://mvnrepository.com/artifact/org.apache.struts/struts2-core -->
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>2.5.14.1</version>
</dependency>
```

2.2 Action 类(用于处理请求)

```
public class Demo1 {
    /**
     * 这个方法用于处理某个请求
     * Action 方法的要求,
     *      必须是 public
     *      返回值必须是 String
     *      不得有参数
     * 代码块就是你当前请求该执行的操作,比如获取到请求的参数,处理数据,返回数据
     *  返回值  用于视图逻辑处理,比如跳转页面等等 可以随便写
     * null 和 none 都是一样的代表不需要返回结果
     * 一般建议 方法的返回值 和方法名保持一致,除了简单的只需要返回 success 或者其他的建
议名字之外
     *
     * @return
     */
    public String helloWorld(){
        System.err.println("hello moto");
        return "success";
    }
}
```

2.3 struts.xml配置文件

> 在项目的 src 目录先创建 struts.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">



<struts>
    <!--开启 debug 模式 修改了当前配置文件 会自动重新加载 -->
    <constant name="struts.devMode" value="true"></constant>
```

```
    <!--定义一个包,当前的地址是从项目/开始 ,具体访问地址等于 namespace+action name   -
->
    <package name="p1"  namespace="/">
        <!-- 配置请求路径和类之间的映射关系 就像 在 web.xml 文件中配置 servlet 一样 -->
        <!-- action动作名,代表的就是请求的地址 只写最后一部分,不需要写项目名之类 class
当前请求的处理类全限定名称 method 处理这个请求的是哪个方法 action 和方法名不一定一样 -->
        <action name="helloWorld" class="com.qianfeng.struts.action.Demo1"
            method="helloWorld">
            <!-- 返回的视图逻辑 name 代表的是返回逻辑视图的内容,就是 action 方法的返回
值 如果 name 与方法的返回值不一致或者不写 result 会出错,出错的异常 No result defined
for action com.qianfeng.struts.action.Demo1 and result success ,最后的 success 代表
方法返回值是success 但是 action 配置里面没有一个 result
                那么为success 有两个特殊值是不需要配置 result 一个叫 null 一个叫 none
                一个 aciton 可以有多个 result
                 -->
            <result name="success">/test1.jsp</result>
        </action>
    </package>
</struts>
```

2.4 配置 web.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <!-- 启用 struts -->
     <filter>
        <filter-name>struts2</filter-name>
        <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filte
r-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

2.5 启动测试

2.6 更多配置,如 struts package 的继承等,请参考2.7配置说明

2.7 struts 配置文件说明

```
1. <package>标签，如果要配置<Action>的标签，那么必须要先配置<package>标签，代表的包的概念
     包含的属性
    name              -- 包的名称，要求是唯一的，管理action配置

    extends           -- 继承其他的包,继承后该包就包含了被继承包的功能，一般都是继承struts-default
    namespace         -- 名称空间，一般与<action>标签中的name属性共同决定访问路径常见的属性如下:
          namespace="/"      -- 根名称空间
          namespace="/aaa"   -- 带有名称的名称空间
          abstract              -- 抽象的。这个属性基本很少使用，值如果是true，那么编写的包是被继承的

2. <action>标签

     代表配置action类，包含的属性

          name          -- 和<package>标签的namespace属性一起来决定访问路径的

          class         -- 配置Action类的全路径（默认值是ActionSupport类）

          method        -- Action类中执行的方法，如果不指定，默认值是execute



3. <result>标签

     action类中方法执行，返回的结果跳转的页面

          name       -- 结果页面逻辑视图名称

          type       -- 结果类型（默认值是转发，也可以设置其他的值）
```

```
<?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<!-- struts.action.extension=action,, 请求的扩展名配置 默认是 action 和没有
struts.devMode
    = false 开发模式 -->


<struts>
    <!-- 修改了当前配置文件 会自动重新加载 -->
```

```
    <constant name="struts.devMode" value="true"></constant>
    <!-- 修改访问的后缀名称,一般不改 <constant name="struts.action.extension"
value="do"></constant> -->
    <!-- 定义一个包 包名随便写,但是不允许重复 因为以后一个项目会有很多个模块,如果把所有的
东西写在一个地方会显得特别乱 包的目的是用于区分当前这个包里面的
        action 是做什么的 包与包之间可以继承 struts-default 是框架自带的一个包,里面配置
了一些默认的拦截器, 我们的包必须直接或者是间接的继承自struts-default
        包的继承, A 包继承了 B 包, B 包里的所有配置都会被 A 给继承 一般继承不会随便继承,
比如这个被继承的 B 包里面一般不会有 action
        被继承的包里面 一般都是一些的通用的配置,比如一些自定义的拦截器,全局的结果视图等等
abstract 声明为抽象包,如果包里面没有任何 action
        可以声明为抽象的,目的就是为了继承 namespace 命名空间, action 的实际访问地址 是
由 命名空间+action 动作名来拼接完成
        比如 namespace="/abc" action 名为helloWorld 那么访问的地址就是
/abc/helloWorld 包 一般建议
        先写一个空包 继承自struts-default ,然后其他的包都继承自空包 目的,因为你暂时不知
道你是否会有相同的配置的情况,为了以防万一,先写一个空的,以后有需要的时候直接在空包里面写就
行了
 如果一个 package 的目的就是被继承的,建议添加属性 abstract="true"
-->
    <package name="mydefault" extends="struts-default">

    </package>
    <!--
    namespace 默认值是空白, 空白和/是不一样的

    如果遇到奇怪的问题,比如 有一个/的明明控件和空白的命名空间
    访问了一个 action.如果访问的 action 里面没有命名空间,会走/的.如果访问的里面有命名空
间,但是没找到,会走空白的命名空间
    -->
    <package name="p23" extends="mydefault" namespace="">
        <action name="helloWorld" class="com.qianfeng.struts.action.Demo3"
            method="helloWorld">
            <result name="success">/test3.jsp</result>
        </action>
    </package>

    <package name="p1" extends="mydefault" namespace="/">
        <!-- 配置请求路径和类之间的映射关系 就像 在 web.xml 文件中配置 servlet 一样 -->
        <!-- action动作名,代表的就是请求的地址 只写最后一部分,不需要写项目名之类 class
当前请求的处理类全限定名称 method
            处理这个请求的是哪个方法 action 和方法名不一定一样 -->
        <action name="helloWorld" class="com.qianfeng.struts.action.Demo1"
            method="helloWorld">
            <!-- 返回的视图逻辑 name 代表的是返回逻辑视图的内容,就是 action 方法的返回
值 如果 name 与方法的返回值不一致或者不写
                result 会出错,出错的异常 No result defined for action
com.qianfeng.struts.action.Demo1
```

```
                and result success 最后的 success 代表方法返回值是success 但是
action 配置里面没有一个 result
                那么为success 有两个特殊值是不需要配置 result 一个叫 null 一个叫 none
                一个 aciton 可以有多个 result
                 -->
            <result name="success">/test1.jsp</result>
            <result name="error">/test2.jsp</result>
        </action>
    </package>


    <package name="p2" extends="mydefault" namespace="/abc/bcd">
        <action name="helloWorld" class="com.qianfeng.struts.action.Demo2"
            method="helloWorld">
            <result name="success">/test2.jsp</result>
        </action>
    </package>

</struts>
```

2.8 struts 的常量配置

```
#我们可以在 struts.xml 中通过constant标签来对 struts 的全局常量配置进行更改,也可以通过
struts.properties 文件进行修改
#struts.properties的配置和constant一样,设置属性-值即可
#比如:<constant name="struts.devMode" value="true"></constant>等价于
struts.devMode = true
```

```
 struts.i18n.encoding=UTF-8            -- 指定默认编码集,作用于HttpServletRequest的
setCharacterEncoding方法
 struts.action.extension=action,,      -- 该属性指定需要Struts 2处理的请求后缀，该属
性的默认值是action，即所有匹配.action的请求都由Struts2处理。如果用户需要指定多个请求后
缀，则多个后缀之间以英文逗号（,）隔开
 struts.serve.static.browserCache=true     -- 设置浏览器是否缓存静态内容,默认值为
true(生产环境下使用),开发阶段最好关闭
 struts.configuration.xml.reload=false     -- 当struts的配置文件修改后,系统是否自动重
新加载该文件,默认值为false(生产环境下使用)
 struts.devMode = false                    -- 开发模式下使用,这样可以打印出更详细的错
误信息
```

2.9 struts 加载配置文件的顺序

1. default.properties 在 org.apache.struts包中 无法修改
2. struts-default.xml 在 struts jar 包中,无法修改
3. struts-plugin.xml 在 struts 的一些插件包中,无法修改
4. struts.xml：在应用的构建路径顶端。自己定义的Struts配置文件（推荐）

5. struts.properties:在应用的构建路径顶端。程序员可以编写（不推荐）
6. web.xml 配置过滤器时，指定参数。程序员可以编写（不推荐）

```
可以通过查看StrutsPrepareAndExecuteFilter的源码发现
init_DefaultProperties();  -- 加载org/apache/struts2/default.properties
init_TraditionalXmlConfigurations();      -- 加载struts-default.xml,struts-
plugin.xml,struts.xml
init_LegacyStrutsProperties();            -- 加载自定义的struts.properties
init_CustomConfigurationProviders();      -- 加载用户自定义配置提供者
init_FilterInitParameters() ;             -- 加载web.xml
```

2.10 struts 配置文件的拆分

```
1．在大部分应用里，随着应用规模的增加，系统中Action的数量也会大量增加，导致struts.xml配
置文件变得非常臃肿。
    为了避免struts.xml文件过于庞大、臃肿，提高struts.xml文件的可读性，我们可以将一个
struts.xml配置文件分解成多个配置文件，然后在struts.xml文件中包含其他配置文件。但是必须在
src 下保留一个 struts.xml 文件,可以将通用的配置放在基础 struts.xml 中
2．可以在<package>标签中，使用<include>标签来引入其他的struts_xx.xml的配置文件。例如：
    <struts>
        <!--导入同级目录下的 xml 文件-->
        <include file="struts-part1.xml"/>


        <include file="struts-part2.xml"/>

    </struts>
3．注意注意注意导入其他包下的配置文件：<include file="com/qianfeng/demo2/struts-
part1.xml"/>
```

三 Action 类的写法

```
在 Struts 中 Action有三种写法,和 struts.xml 的配置无关
1．直接是任意类
2．实现 Action 接口
3．继承 ActionSupport 类
```

3.1 实现 Action 接口

```
/**
 * 实现 Action 接口的好处，可以使用 struts 默认 提供的一些方法的返回值常量
 *  以下值是框架建议使用,同时在框架内部的一些判断 返回的结果也是这些值
 *  SUCCESS 建议的成功的返回值内容
```

```
 *  ERROR  action在执行期间出现了错误 异常之类的会建议返回 ERROR, 内部默认返回也是
ERROR(框架内部的代码如果出现错误就返回 error)
 *  NONE 代表不需要任何返回值
 *  INPUT 代表输入错误的时候跳转到输入页面,具体页面需要指定(框架内部返回 INPUT 我们在
action 的 result 必须定义一个结果接收 input 设置跳转的页面)
 *  LOGIN  代表需要登陆,需要在 action 的结果中定义一个 result 来接收 LOIGIN 指定需要跳
转到登陆页面
 *
 * @author jackiechan
 *
 */
public class ActionDemo2 implements Action{

    /**
     * 默认执行的方法 可以空实现,但是如果用到了必须得写具体逻辑
     *
     */
    @Override
    public String execute() throws Exception {
        System.err.println("实现 Action 接口的方式");
        return SUCCESS;
    }
}
```

3.2 继承自 ActionSupport

```
/**
 *  继承ActionSupport的好处
 *  ActionSupport 实现 Action 接口,所有 Action 接口的好处它都有
 *  可以实现参数的自动封装
 *  可以对数据进行验证
 *  国际化可以直接使用
 * @author jackiechan
 *
 */
public class ActionDemo3 extends ActionSupport{

    public String hello(){
        System.err.println("继承 ActionSupport 的方式");
        return NONE;
    }
}
```

3.2.1 默认 Action 类

> 如果 我们指定的 action 并没有配置 class, 那么struts 给我们默认执行了一个类 就是 com.opensymphony.xwork2.ActionSupport ,然后执行内部指定的execute 方法,默认返回 success, 注意在没有指定类的情况下,不能指定方法

```
<struts>
<!--
动态方法调用,直接在 action 后面加!方法名 可以直接调用 action 类里面的对应方法
    坚决反对使用,防止方法的非法调用
    <constant name="struts.enable.DynamicMethodInvocation" value="true">
</constant>
-->
    <constant name="struts.devMode" value="true"></constant>
    <package name="mydefault" extends="struts-default">
    </package>

    <package name="p23" extends="mydefault">
    <!--
    只配置了一个 action 名
    class 有默认值 com.opensymphony.xwork2.ActionSupport

    方法有默认方法execute  这个方法有个固定的返回值  success
     -->
        <action name="helloWorld" >
            <result name="success">/test3.jsp</result>
        </action>
        <action name="m1" class="com.qianfeng.struts.action.Demo1" method="m1">
        </action>
        <!-- <action name="m2" class="com.qianfeng.struts.action.Demo1"
method="m2">
        </action>
        <action name="m3" class="com.qianfeng.struts.action.Demo1" method="m3">
        </action>
        <action name="m76" class="com.qianfeng.struts.action.Demo1" method="m76">
        </action> -->

        <action name="testimpl" class="com.qianfeng.struts.action.ActionDemo2">
            <result name="success">/test3.jsp</result>
        </action>
        <action name="hello" class="com.qianfeng.struts.action.ActionDemo3"
method="hello">
            <result name="success">/test2.jsp</result>
        </action>
    </package>

</struts>
```

四 通配符

> struts支持使用通配符来批量设置访问路径

4.1 struts.xml

```
<struts>
    <constant name="struts.devMode" value="true"></constant>
    <package name="p1" extends="struts-default">
        <!-- 通配符 *代表通配符 {1} 取的是第一个*的值 取值从1开始  add_User 第一个*add
第二个* User 方法应该是 addUser
            /add.jsp 如果需要切割请求的时候 用通配符需要切割,不要将通配符连起来 比如**
相当于* Class clzzz= Class.forname(com.qianfeng.struts.action.ActionDemo1);
            Object obj =clazz.newInstance();
clazz.getMethod("addUser").invoke(obj);
            绝对匹配具有最高的优先级
            如果有多个通配符能匹配目标 action 的时候,谁先出现谁先执行
            通配符.匹配范围越小的应该出现的越早,以免被大范围的覆盖掉
            用到通配符说 action 和方法名甚至类名以及返回的结果视图 命名名必须规范
            -->

        <action name="add_User" class="com.qianfeng.struts.action.ActionDemo2"
            method="addUser"></action>

        <action name="*" class="com.qianfeng.struts.action.ActionDemo3"
            method="findUser"></action>
        <!--访问路径中出现一个下划线,那么最终的路径的下划线前面的部分将作为方法的前半部
分,下划线后面的部分作为方法的后半部分 区分大小写-->
        <action name="*_*" class="com.qianfeng.struts.action.ActionDemo1"
            method="{1}{2}">
          <!--将下划线前面的部分作为跳转页面的 jsp 文件的名字-->
            <result name="success">/{1}.jsp</result>
        </action>
    </package>

</struts>
```

4.2 Action 类

```
public class ActionDemo1 extends ActionSupport {
    public String addUser() {
        System.err.println("addUser");
        return SUCCESS;
    }

    public String findUser() {
```

```
        System.err.println("findUser");
        return SUCCESS;
    }

    public String updateUser() {
        System.err.println("updateUser");
        return SUCCESS;
    }
}
```

五 访问 Servlet 中的 api

5.1 方式一 ServletActionContext

> struts 给我们提供了一个ServletActionContext类,内部封装有静态方法,可以帮我们获取 request ,response,session 和 servletcontext 对象

```
/**
 * 通过前面的配置,我们可以让一个请求执行 Action 类里面的对应的方法,但是因为方法没有参数,无法获取请求的数据
 * @author jackiechan
 *
 */
public class ActionDemo1 extends ActionSupport {

    public String helloWorld() throws IOException {
        //获取用户传递参数
        //方式1 ServletActionContext 不是单例对象,每次请求都创建一个对象,随着 action 走
            String name = ServletActionContext.getRequest().getParameter("name");

ServletActionContext.getResponse().setContentType("text/html;charset=utf-8");
            ServletActionContext.getResponse().getWriter().write("这是返回的内容");//自己处理返回结果,不需要 struts 处理
            //没有解决乱码问题
            System.err.println(name);
        return NONE;
    }


}
```

5.2 实现指定接口

> struts 包含了ServletRequestAware,ServletResponseAware,ServletContextAware三个接口,实现后重写方法, struts 在找到 action 类的时候,会帮我们将对应的对象通过方法注入进来, 我们只需要声明一个全局变量,在方法内部接收变量即可

```
/**
 * 通过依赖注入的方式获取请求和响应对象
 *
 * @author jackiechan
 *
 */
public class ActionDemo2 extends ActionSupport implements
ServletRequestAware,ServletResponseAware,ServletContextAware {

    private HttpServletRequest request;//请求对象
    private HttpServletResponse response;
    private ServletContext context;
    public String helloWorld() throws IOException {
        String name = request.getParameter("name");
        System.err.println(name);
        return NONE;
    }
    /**
     * 注入请求对象到当前 action
     * request 就是当前的请求对象
     *
     * 实现方式, Struts 过滤器,检查配置文件,找到 Action 对应的类.然后反射创建对象,将这个对象一层一层传递到每一个拦截器里面
     *  每一个拦截器在其对应的intercept方法内部进行相应的处理
     * 这些方法会在真正的 action 方法之前进行调用
     */
    @Override
    public void setServletRequest(HttpServletRequest request) {
        this.request=request;
    }
    /**
     * 注入响应对象
     */
    @Override
    public void setServletResponse(HttpServletResponse response) {
        this.response=response;
    }
    /**
     * 注入 servletcontext  application
     */
    @Override
    public void setServletContext(ServletContext context) {
        this.context=context;
    }
```

```
}
```

5.2.1 实现原理

> struts 内部有一个servletConfig拦截器, struts 在处理映射地址找到 action 类后,在调用方法之前,会判断当前 action 是否属于指定接口的类型,如果属于,就调用对应的方法将对象注入进来,这样子就调用到了我们的 action 上面实现的方法, 最终对象注入进来

```
public class ServletConfigInterceptor extends AbstractInterceptor implements
StrutsStatics {

    private static final long serialVersionUID = 6052617778586766 38L;

    /**
     * Sets action properties based on the interfaces an action implements. Things
like application properties,
     * parameters, session attributes, etc are set based on the implementing
interface.
     *
     * @param invocation an encapsulation of the action execution state.
     * @throws Exception if an error occurs when setting action properties.
     */
    public String intercept(ActionInvocation invocation) throws Exception {
        final Object action = invocation.getAction();//获取要执行的 action
        final ActionContext context = invocation.getInvocationContext();

        if (action instanceof ServletRequestAware) {//判断action 是否属于
ServletRequestAware,属于则获取request对象并调用setServletRequest方法
            HttpServletRequest request = (HttpServletRequest)
context.get(HTTP_REQUEST);
            ((ServletRequestAware) action).setServletRequest(request);
        }

        if (action instanceof ServletResponseAware) {//同上
            HttpServletResponse response = (HttpServletResponse)
context.get(HTTP_RESPONSE);
            ((ServletResponseAware) action).setServletResponse(response);
        }

        if (action instanceof ParameterAware) {//同上
            context.getParameters().applyParameters((ParameterAware) action);
        }

        if (action instanceof HttpParametersAware) {//同上
            ((HttpParametersAware) action).setParameters(context.getParameters());
        }
```

```
        if (action instanceof ApplicationAware) {//同上
            ((ApplicationAware) action).setApplication(context.getApplication());
        }

        if (action instanceof SessionAware) {//同上
            ((SessionAware) action).setSession(context.getSession());
        }

        if (action instanceof RequestAware) {//同上
            ((RequestAware) action).setRequest((Map) context.get("request"));
        }

        if (action instanceof PrincipalAware) {//同上
            HttpServletRequest request = (HttpServletRequest)
context.get(HTTP_REQUEST);
            if(request != null) {
                // We are in servtlet environment, so principal information
resides in HttpServletRequest
                ((PrincipalAware) action).setPrincipalProxy(new
ServletPrincipalProxy(request));
            }
        }
        if (action instanceof ServletContextAware) {//同上
            ServletContext servletContext = (ServletContext)
context.get(SERVLET_CONTEXT);
            ((ServletContextAware) action).setServletContext(servletContext);
        }
        return invocation.invoke();
    }
}
```