Atitit 开发效率 脚本语言 php jsp js

Jsp脚本通用性貌似更强

由于JSP页面的内置脚本语言是基于Java编程语言的，而且所有的JSP页面都被编译成为Java Servlet，JSP页面就具有Java技术的所有好处，包括健壮的存储管理和安全性。

作为J

Jasper是Tomcat自带的JSP编译器，它可以将JSP文件转换成Java源文件和class文件供Tomcat加载并使用。网上关于Jasper编译器如何使用的文章少之又少，在我翻阅了Tomcat官方文档后领悟到了使用方法。如果你想尝试自己写一个像Tomcat那样的Servlet容器，那么这篇文章适合你

如果你想只编译一个JSP文件，在提交前加上

jspc.setJspFiles("index.jsp");

取得 JSP 執行後的字串內容

為了取得執行後的字串內容，我們需要建立 Response 並且替換 PrintWriter，這樣才有辦法取得執行後的內容。

{

//

//

//

//

// private StringWriter writer = new StringWriter();

//

// @Override

// public PrintWriter getWriter() throws UnsupportedEncodingException {

// /\* 用 StringWriter 作為輸出的容器 \*/

// return new PrintWriter(writer);

// }

////

//// @Override

//// public boolean isCommitted() {

//// /\* true 是為了讓 View 可以採用 include 方式 Render 到 Response \*/

//// return true;

//// }

//

// @Override

// public String getContentAsString() throws UnsupportedEncodingException {

// /\* 取得 Render 後的內容 \*/

// return writer.getBuffer().toString();

// }

//

//

//

// };

package script;

import java.io.File;

import java.io.PrintWriter;

import java.io.StringWriter;

import java.io.UnsupportedEncodingException;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.URLClassLoader;

import javax.servlet.Servlet;

import javax.servlet.jsp.JspFactory;

import javax.servlet.jsp.PageContext;

import javax.tools.JavaCompiler;

import javax.tools.ToolProvider;

import org.apache.commons.io.FileUtils;

import org.apache.commons.io.FilenameUtils;

import org.apache.commons.lang3.reflect.MethodUtils;

import org.apache.jasper.JspC;

import org.apache.jasper.runtime.JspFactoryImplAti;

import org.springframework.mock.web.MockHttpServletRequest;

import org.springframework.mock.web.MockHttpServletResponse;

import org.springframework.mock.web.MockServletConfig;

public class JspUti {

public static void main(String[] **args**)

throws Exception {

String jspfileDir = "C:\\Users\\jun\\eclipse-workspace\\deadlockchk\\scrptJsp"; //\\t.jsp

String classesPathDir = "C:\\Tomcat\_work2";

String jspRetStr = *invokeJsp*(jspfileDir,"t.jsp", classesPathDir);

System.***out***.println("-----------------jsp ret start==========");

System.***out***.println(jspRetStr);

System.***out***.println("-----------------jsp ret end==========");

//

// System.out.println( MethodUtils.invokeMethod(newInstance,"halo"));

// //this invoke ites ok。。good

System.***out***.println("--f");

}

private static String invokeJsp(String **jspfileDir**, String **jspfile**, String **classesPathDir**) throws Exception {

// compiler jsp to class file

JspC jspc = new JspC(); // 创建一个Jasper编译器对象

// jspc.setUriroot(s);

jspc.setUriroot(**jspfileDir**); // WEB应用存放路径 batch cmplt

jspc.setJspFiles(**jspfile**); // only one file cmplt

jspc.setOutputDir(**classesPathDir**); // JSP编译结果输出路径

jspc.setCompile(true); // 是否将JSP转换后的.java源代码文件转换成.class文件

jspc.execute(); // 确认提交

System.***out***.println("f");

*loadClass*(**classesPathDir**);

String bscname = FilenameUtils.*getBaseName*(**jspfile**);

String className = "org.apache.jsp." + bscname + "\_jsp";

String jspRetStr = *invokeJspClass*(className);

return jspRetStr;

}

private static String invokeJspClass(String **className**) throws Exception {

// 文件名称

// String className = subFile.getAbsolutePath();

// className = className.substring(clazzPathLen, className.length() - 6);

// className = className.replace(File.separatorChar, '.');

// 加载Class类

Class cls = Class.*forName*(**className**);

Object newInstance = cls.newInstance();

MockServletConfig MockServletConfig1 = new MockServletConfig();

MockHttpServletRequest mockHttpServletRequest = new MockHttpServletRequest();

mockHttpServletRequest.setMethod("GET");

MockHttpServletResponse mockHttpServletResponse = *newMockResp*();

// MockServletConfig msc = new MockServletConfig();

// // JspFactory.setDefaultFactory(new JspFactoryImplAti());

// MethodUtils.invokeStaticMethod(JspFactory.class, "setDefaultFactory", new JspFactoryImplAti());

Object r = MethodUtils.*invokeMethod*(newInstance, "\_jspService", mockHttpServletRequest,

mockHttpServletResponse);

String jspRetStr = mockHttpServletResponse.getContentAsString();

return jspRetStr;

}

private static void loadClass(String **classesPathDir**)

throws Exception {

//must bef load..bcz its static ini

JspFactory.*setDefaultFactory*(new JspFactoryImplAti());

// 加载class load class from class file

Method method = URLClassLoader.class.getDeclaredMethod("addURL", URL.class);

method.setAccessible(true);

// 设置类加载器

ClassLoader systemClassLoader = ClassLoader.*getSystemClassLoader*();

URLClassLoader classLoader = (URLClassLoader) systemClassLoader;

File clazzPath = new File(**classesPathDir**);

// 将当前类路径加入到类加载器中

method.invoke(classLoader, clazzPath.toURI().toURL());

}

private static MockHttpServletResponse newMockResp() {

MockHttpServletResponse mockHttpServletResponse = new MockHttpServletResponse()

{

public StringWriter writer = new StringWriter();

*@Override*

public PrintWriter getWriter() throws UnsupportedEncodingException {

/\* 用 StringWriter 作為輸出的容器 \*/

return new PrintWriter(writer);

// return null;

}

//

*@Override*

public boolean isCommitted() {

/\* true 是為了讓 View 可以採用 include 方式 Render 到 Response \*/

return true;

}

*@Override*

public String getContentAsString() throws UnsupportedEncodingException {

/\* 取得 Render 後的內容 \*/

return writer.getBuffer().toString();

}

// getServletConfig

};

return mockHttpServletResponse;

}

private static void invokeJava(String **javafile**, String **out\_classesPathDir**) throws Exception {

JavaCompiler javac = ToolProvider.*getSystemJavaCompiler*();

File javaFile=new File(**javafile**);

//JavaCompiler最核心的方法是run, 通过这个方法编译java源文件, 前3个参数传null时,

//分别使用标准输入/输出/错误流来 处理输入和编译输出. 使用编译参数-d指定字节码输出目录.

int compileResult = javac.run(null, null, null, "-d", new File(**out\_classesPathDir**).getAbsolutePath(), javaFile.getAbsolutePath());

//run方法的返回值: 0-表示编译成功, 否则表示编译失败

if(compileResult != 0) {

System.***err***.println("编译失败!!");

return;

}

String className = "script.Bean2";

Class cls = Class.*forName*(className);

MethodUtils.*invokeStaticMethod*(cls, "main", (Object)new String[]{});

return;

}

}