Atitit 通用Servlet CommonServlet v2 s66

主治 通用参数

/hislog/src/main/java/com/attilax/prj/hislog/commServletV2.java

**package** com.attilax.prj.hislog;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.lang.reflect.InvocationTargetException;

**import** java.net.URLEncoder;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.ServletException;

**import** javax.servlet.ServletOutputStream;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.commons.io.IOUtils;

**import** org.apache.commons.lang3.reflect.ConstructorUtils;

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

**import** com.alibaba.fastjson.JSON;

**import** com.alibaba.fastjson.JSONObject;

**import** com.attilax.ast.ClassInstanceCreation;

**import** com.attilax.ast.Expression;

**import** com.attilax.ast.MethodInvocation;

**import** com.attilax.ast.SimpleName;

**import** com.attilax.core.astExpressInterpreter;

**import** com.attilax.device.perfMonitorService;

**import** com.attilax.io.pathx;

**import** com.attilax.net.reqUtil;

**import** com.attilax.ref.refServiceV2s530;

**import** com.attilax.shell.CliShellProcessService;

**import** com.attilax.shell.ProcessUtil;

**import** com.attilax.util.ExUtil;

**import** com.attilax.web.respService;

**import** com.google.common.collect.Lists;

/\*\*

\* v3 add relt path mode 基于本地代理的文件绝对路径选择器 内嵌式ftp server集成测试 文件上传下载与一些移动复制操作方便实施

\* 以及目录浏览

\*

\* **@author** attilax

\*

\*/

// http://localhost:9999/ShellServlet3?cmdtextbox=/libExe/FileOpenDialog.exe

@WebServlet(name = "commServletV2", urlPatterns = { "/commServletV2", "/commServletprjhislogxxx" }, loadOnStartup = 1)

**public** **class** commServletV2 **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html;charset = utf-8");

**boolean** **isNew** = (reqUtil.*hasOption*("new", request));

String **clsname** = reqUtil.*getOptionValue*("class", request);

List<Object> **con\_paramsList** = get\_con\_paramsList(request);

String **meth** = reqUtil.*getOptionValue*("method", request);

List<Object> **paramsList** = getParams(request);

Object **mi\_rzt**;

mi\_rzt = getRztV2\_retAst(isNew, clsname, meth, paramsList, con\_paramsList);

System.***out***.println(JSON.*toJSONString*(mi\_rzt, **true**));

Object **interpret\_rzt** = **new** astExpressInterpreter().interpret((Expression) mi\_rzt);

respService.*out\_asjson*(response, interpret\_rzt);

} **catch** (Exception **e**) {

ExUtil.*throwExV2*(e);

}

}

**private** Object getRztV2\_retAst(**boolean** isNew, String clsname, String meth, List<Object> paramsList,

List<Object> con\_paramsList) {

MethodInvocation **mi** = **new** MethodInvocation();

Expression **miexp**;

Object **rzt**;

**if** (isNew) {

ClassInstanceCreation **cic** = **new** ClassInstanceCreation();

cic.jsonname = "ClassInstanceCreation";

cic.name = clsname;

cic.arguments = con\_paramsList;

miexp = cic;

} **else** {

SimpleName **sn** = **new** SimpleName(clsname);

miexp = sn;

}

mi.Exp = miexp;

mi.Name = meth;

mi.arguments = paramsList;

mi.jsonname = "MethodInvocation";

**return** mi;

}

//

// private static List get\_con\_paramsList(CommandLine cmd) {

// if (cmd.hasOption("conargs")) {

// List con\_paramsList ;//= get\_con\_paramsList();

// return Lists.newArrayList();

//

// } else

//

// return Lists.newArrayList();

// }

**private** Object getRzt(Class cls, String meth, List<Object> paramsList)

**throws** NoSuchMethodException, IllegalAccessException, InvocationTargetException, InstantiationException {

Object **rzt**;

**if** (reqUtil.~~hasOption~~("new")) {

Object **obj**;

**if** (reqUtil.~~hasOption~~("conargs")) {

List **con\_paramsList** = get\_con\_paramsList();

obj = ConstructorUtils.*invokeConstructor*(cls, con\_paramsList.toArray());

} **else**

obj = cls.newInstance();

rzt = MethodUtils.*invokeMethod*(obj, meth, paramsList.toArray());

} **else**

rzt = MethodUtils.*invokeStaticMethod*(cls, meth, paramsList.toArray());

**return** rzt;

}

**private** List get\_con\_paramsList() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**private** List get\_con\_paramsList(HttpServletRequest request) {

// **TODO** Auto-generated method stub

**return** **null**;

}

**private** List<Object> getParams(HttpServletRequest request) {

List<Object> **paramsList** = Lists.*newArrayList*();

**if** (request.getParameter("args") == **null**)

**return** Lists.*newArrayList*();

**if** (request.getParameter("args").length() > 0) {

String **arguments** = request.getParameter("args");

String **paramtypes** = request.getParameter("paramtypes");

String[] **a** = arguments.split(",");

**int** **n** = 0;

**for** (String **string** : a) {

Object **p** = refServiceV2s530.*getArg*(string, n, paramtypes.split(","));

paramsList.add(p);

n++;

}

}

**return** paramsList;

}

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

doGet(request, response);

}

// public static String filenameEncoding(String filename, HttpServletRequest

// request) throws IOException {

// String agent = request.getHeader("User-Agent"); //鑾峰彇娴忚鍣�

// if (agent.contains("Firefox")) {

// BASE64Encoder base64Encoder = new BASE64Encoder();

// filename = "=?utf-8?B?"

// + base64Encoder.encode(filename.getBytes("utf-8"))

// + "?=";

// } else if(agent.contains("MSIE")) {

// filename = URLEncoder.encode(filename, "utf-8");

// } else {

// filename = URLEncoder.encode(filename, "utf-8");

// }

// return filename;

// }

}