com.fr.web.utils.WebUtils

-
- //——FineReport V10 2020-07-18
- //
-
-

//——FineReport V10 2020-07-18

WebUtils.java

```
package com.fr.web.utils;

...

/**
 * web
 */
public class WebUtils {

    private WebUtils() {

    }


    /**
     * HTTP
     *
     * @param req      HTTP
     * @param paraName
     * @return
     */
    public static String getHTTPRequestParameter(HttpServletRequest req, String paraName) {
        return NetworkHelper.getHTTPRequestParameter(req, paraName);
    }

    /**
     *
     *
     * @param req       HTTP
     * @param paraNames
     * @return
     */
    public static String getHTTPRequestParameter(HttpServletRequest req, String... paraNames) {
        return NetworkHelper.getHTTPRequestParameter(req, paraNames);
    }

    /**
     * HTTP
     *
     * @param req      HTTP
     * @param paraName
     * @return
     */
    public static boolean getHTTPRequestBoolParameter(HttpServletRequest req, String paraName) {
        return NetworkHelper.getHTTPRequestBoolParameter(req, paraName);
    }

    /**
     * HTTP-1
     *
     * @param req      HTTP
```

```
     * @param paraName
     * @return
     */
    public static int getHTTPRequestIntParameter(HttpServletRequest req, String paraName) {
        return getHTTPRequestIntParameter(req, paraName, -1);
    }

    /**
     * HTTP
     *
     * @param req          HTTP
     * @param paraName
     * @param defaultValue
     * @return reqdefaultValue
     */
    public static int getHTTPRequestIntParameter(HttpServletRequest req, String paraName, int defaultValue) {
        return NetworkHelper.getHTTPRequestIntParameter(req, paraName, defaultValue);
    }

    /**
     *
     *
     * @param req HTTP
     * @return
     */
    public static String getReportTitleFromRequest(HttpServletRequest req) {
        String title = WebUtils.getHTTPRequestParameter(req, ParameterConstants.VIEWLET);

        if (title == null) {
            title = WebUtils.getHTTPRequestParameter(req, ParameterConstants.VIEWLETS);
        }

        if (title == null) {
            title = WebUtils.getHTTPRequestParameter(req, ParameterConstants.REPORTLET);
        }

        if (title == null) {
            title = WebUtils.getHTTPRequestParameter(req, ParameterConstants.REPORTLETS);
        }

        if (title == null) {
            title = WebUtils.getHTTPRequestParameter(req, ParameterConstants.RESULTLET);
        }

        return title;
    }

    /**
     * Servelt
     *
     * @param req HTTP
     * @return URL
     */
    public static String createServletURL(HttpServletRequest req) {
        return NetworkHelper.createServletURL(req);
    }

    /**
     *
     *
     * @param res HTTP
     * @return
     * @throws IOException
     */
    public static PrintWriter createPrintWriter(HttpServletResponse res) throws IOException {
        return NetworkHelper.createPrintWriter(res);
    }

    /**
     * JSON
     *
```

```
     * @param req  HTTP
     * @param res  HTTP
     * @param data
     * @throws Exception
     */
    public static void flushSuccessMessageAutoClose(HttpServletRequest req, HttpServletResponse res, JSONObject
data) throws Exception {
        WebUtils.printAsJSON(res, data);
    }

    /**
     * JSON
     *
     * @param req  HTTP
     * @param res  HTTP
     * @param data
     * @throws Exception
     */
    public static void flushSuccessMessageAutoClose(HttpServletRequest req, HttpServletResponse res, JSONArray
data) throws Exception {
        WebUtils.printAsJSON(res, data);
    }

    /**
     * JSONwriter
     *
     * @param req    HTTP
     * @param res    HTTP
     * @param writer
     * @param data
     * @throws Exception
     */
    public static void flushSuccessMessage(HttpServletRequest req, HttpServletResponse res, PrintWriter writer,
JSONArray data) throws Exception {
        writer.print(data.toString());
    }

    /**
     * JSONwriter
     *
     * @param req    HTTP
     * @param res    HTTP
     * @param writer
     * @param data
     * @throws Exception
     */
    public static void flushSuccessMessage(HttpServletRequest req, HttpServletResponse res, PrintWriter writer,
JSONObject data) throws Exception {
        writer.print(data.toString());
    }

    /**
     *
     *
     * @param req         HTTP
     * @param res         HTTP
     * @param errorCode
     * @param description
     * @throws Exception
     */
    public static void flushFailureMessageAutoClose(HttpServletRequest req, HttpServletResponse res, int
errorCode, String description) throws Exception {
        flushFailureMessageAutoClose(req, res, errorCode, description, null);
    }

    /**
     *
     *
     * @param req            HTTP
     * @param res            HTTP
     * @param errorCode
```

```
     * @param description
     * @param compatibleData
     * @throws Exception
     */
    public static void flushFailureMessageAutoClose(HttpServletRequest req, HttpServletResponse res, int
errorCode, String description, JSONObject compatibleData) throws Exception {
        if (StringUtils.isEmpty(description)) {
            throw new IllegalArgumentException("The description of error code cannot be empty!");
        }
        WebUtils.printAsJSON(res, generateErrorCodeObject(errorCode, description, compatibleData));
    }

    /**
     *
     *
     * @param req            HTTP
     * @param writer
     * @param errorCode
     * @param description
     * @param compatibleData
     * @throws Exception
     */
    public static void flushFailureMessage(HttpServletRequest req, PrintWriter writer, int errorCode, String
description, JSONObject compatibleData) throws Exception {
        if (StringUtils.isEmpty(description)) {
            throw new IllegalArgumentException("The description of error code cannot be empty!");
        }
        writer.print(generateErrorCodeObject(errorCode, description, compatibleData).toString());
    }

    private static JSONObject generateErrorCodeObject(int errorCode, String description, JSONObject
compatibleData) throws Exception {
        JSONObject data = JSONObject.create().put("errorCode", errorCode).put("errorMsg", description);
        if (compatibleData != null) {
            Iterator it = compatibleData.keys();
            while (it.hasNext()) {
                String key = GeneralUtils.objectToString(it.next());
                data.put(key, compatibleData.get(key));
            }
        }
        return data;
    }

    /**
     *
     *
     * @param res         HTTP
     * @param charsetName
     * @return
     * @throws IOException
     */
    public static PrintWriter createPrintWriter(HttpServletResponse res, String charsetName) throws IOException
{
        return NetworkHelper.createPrintWriter(res, charsetName);
    }

    /**
     * JSON
     *
     * @param res HTTP
     * @param jo  JSON
     * @throws Exception
     */
    public static void printAsJSON(HttpServletResponse res, JSONObject jo) throws Exception {
        printAsString(res, jo.toString());
    }

    /**
     * JSON
     *
     * @param res HTTP
```

```
     * @param ja  JSON
     * @throws Exception
     */
    public static void printAsJSON(HttpServletResponse res, JSONArray ja) throws Exception {
        printAsString(res, ja.toString());
    }

    /**
     * JSON
     *
     * @param res HTTP
     * @param jo  JSON
     * @throws Exception
     */
    public static void printAsString(HttpServletResponse res, String jo) throws Exception {
        PrintWriter pw = WebUtils.createPrintWriter(res);
        pw.print(jo);
        pw.flush();
        pw.close();
    }


    /**
     * JSON
     *
     * @param object
     * @return JSON
     * @throws JSONException
     */
    public static Object object2JSONable(Object object) throws JSONException {
        // alex:DateImage,JSON
        return object2JSONable(object, -1, -1);
    }

    /**
     * JSON
     *
     * @param object
     * @param cellwidth  k
     * @param cellheight g
     * @return JSON
     * @throws JSONException
     */
    public static Object object2JSONable(Object object, int cellwidth, int cellheight) throws JSONException {
        if (object instanceof Date) {
            return new JSONObject().put("date_milliseconds", new Long(((Date) object).getTime()));
        } else if (object instanceof Image) {
            if (cellheight > 0 && cellwidth > 0) {
                return AttachmentHelper.addAttachment((Image) object, AttachmentHelper.DEFAULT_IMAGE_FORMAT,
                        cellwidth, cellheight, AttachmentScope.DEFAULT).toConfig();
            } else {
                return AttachmentHelper.addAttachment((Image) object, AttachmentScope.DEFAULT).toConfig();
            }
        } else if (object instanceof Double) {
            if (((Double) object).isInfinite() || ((Double) object).isNaN()) {
                return GeneralUtils.objectToString(object);
            }
        } else if (object instanceof Float) {
            if (((Float) object).isInfinite() || ((Float) object).isNaN()) {
                return GeneralUtils.objectToString(object);
            }
        } else if (object instanceof FArray) {
            return object;
        }
        return JSONHolder.hold(object);
    }

    /**
     *
     *
     * @param resource
```

```
     * @param response HTTP
     * @throws IOException
     */
    public static void dealWithTemplate(String resource, HttpServletResponse response) throws IOException {
        writeOutTemplate(resource, response, java.util.Collections.EMPTY_MAP);
    }

    /**
     *
     *
     * @param resource
     * @param response HTTP
     * @param map
     * @throws IOException
     */
    public static void writeOutTemplate(String resource, HttpServletResponse response, Map map) throws
IOException {

        //debugjscss
        if (StableUtils.isDebug()) {
            StableFactory.refreshTypeStyleFiles();
            StableFactory.refreshJavaScriptFiles();
        }
        PrintWriter writer = WebUtils.createPrintWriter(response);
        //bug45146weblogic/resincontentTypecontentTypetext/htmlresource
        WebUtils.setResourceContentType(resource, response);
        TemplateUtils.dealWithTemplate(resource, writer, map);
        writer.flush();
        writer.close();
    }

    /**
     * @deprecated use {@link BaseWebUtils#invalidResourcePath} instead.
     * */
    @Deprecated
    public static boolean invalidResourcePath(String resourcePath) {
        return BaseWebUtils.invalidResourcePath(resourcePath);
    }

    /**
     *
     *
     * @param resourceName
     * @param res          http
     * @return
     */
    public static ResourceType setResourceContentType(String resourceName, HttpServletResponse res) {
        if (StringUtils.isBlank(resourceName)) {
            return ResourceType.NONE;
        }
        return getResourceType(res, resourceName.toLowerCase());
    }

    private static ResourceType getResourceType(HttpServletResponse res, String lowerName) {

        String charset = ServerConfig.getInstance().getServerCharset();
        if (lowerName.endsWith(".png")) {
            res.setContentType("image/png");
        } else if (lowerName.endsWith(".gif")) {
            res.setContentType("image/gif");
        } else if (lowerName.endsWith(".jpg")) {
            res.setContentType("image/jpeg");
        } else if (lowerName.endsWith(".js")) {
            res.setContentType("text/javascript;charset=" + charset);
            return ResourceType.FILE;
        } else if (lowerName.endsWith(".json")) {
            res.setContentType("text/plain;charset=" + charset);
            return ResourceType.FILE;
        } else if (lowerName.endsWith(".map")) {
            return ResourceType.FILE;
        } else if (lowerName.endsWith(".css")) {
```

```
            res.setContentType("text/css;charset=" + charset);
            return ResourceType.FILE;
        } else if (lowerName.endsWith(".xml")) {
            res.setContentType("text/xml;charset=" + charset);
            return ResourceType.NONE;
        } else if (lowerName.endsWith(".swf")) {
            res.setContentType("application/x-shockwave-flash");
        } else if (lowerName.endsWith(".jnlp")) {
            res.setContentType("application/x-java-jnlp-file");
        } else if (lowerName.endsWith(".xls")) {
            res.setContentType("application/vnd.ms-excel");
        } else if (lowerName.endsWith(".exe")) {
            res.setContentType("application/x-msdownload");
            res.setHeader("extension", "exe");
            res.setHeader("Content-disposition", "attachment; filename=AcrobatReader.exe");
        } else if (lowerName.endsWith(".ttf")) {
            res.setContentType(".ttf");
        } else if (lowerName.endsWith(".eot")) {
            res.setContentType(".eot");
        } else if (lowerName.endsWith(".woff")) {
            res.setContentType(".woff");
        } else if (lowerName.endsWith(".cur")) {
            res.setContentType(".cur");
        } else if (lowerName.endsWith(".htc")) {
            res.setContentType("text/x-component");
        } else if (lowerName.endsWith(".html")) {
            res.setContentType("text/html;charset=" + charset);
            return ResourceType.FILE;
        } else {
            res.setContentType("text/html;charset=" + charset);
            return ResourceType.NONE;
        }
        return ResourceType.OTHER;
    }

    /**
     * map
     *
     * @param req HTTP
     * @return map
     * @throws Exception
     */
    public static Map createSessionIDParameterMap(HttpServletRequest req) throws Exception {
        HashMap parameterMap = new HashMap();

        Enumeration parameterNames = req.getParameterNames();
        while (parameterNames.hasMoreElements()) {
            String parameterName = (String) parameterNames.nextElement();
            // "reportlet",.
            if (StringUtils.isBlank(parameterName)) {
                continue;
            }

            // .
            if (isSpecificParameters(parameterName)) {
                continue;
            }

            // Encoding text.
            parameterMap.put(parameterName, WebUtils.getHTTPRequestParameter(req, parameterName));
        }

        return parameterMap;
    }

    private static boolean isSpecificParameters(String parameterName) {
        return "reportlet".equalsIgnoreCase(parameterName) || "format".equalsIgnoreCase(parameterName) || "op".
equalsIgnoreCase(parameterName);
    }

    /**
```

```
     * carl:Map
     * json
     *
     * <p/>
     * Map
     *
     * @param paramMap map
     * @return Map
     */
    public static Map<String, Object> dealWithExecuteParamMap(Map<String, Object> paramMap) {
        // alex:executeMap,Map,sessionIDInforMap,
        Map<String, Object> map4Execute = new HashMap<String, Object>();
        if (paramMap != null) {
            for (String key : paramMap.keySet()) {
                Object ob = paramMap.get(key);
                // richer:JSONObject
                if (ob instanceof JSONObject) {
                    for (Map.Entry<String, Object> entry : (JSONObject)ob) {
                        String name = entry.getKey();
                        map4Execute.put(name, ((JSONObject) ob).getValue(name));
                    }
                }
            }
            map4Execute.putAll(paramMap);
        }

        // remove server parameters
        for (int i = 0; i < ParameterConstants.ALL.length; i++) {
            map4Execute.remove(ParameterConstants.ALL[i]);
        }

        return map4Execute;
    }


    /**
     * webgetRealPath
     *
     * @param servletContext the servletContext
     * @return return the WEB-INF Path
     */
    public static String getWebINFPath(ServletContext servletContext) {
        return BaseWebUtils.getWebINFPath(servletContext);
    }

    /**
     * map
     *
     * @param req
     * @return map
     */
    public static Map<String, Object> parameters4SessionIDInforContainMPCache(HttpServletRequest req) {
        Map<String, Object> map = parameters4SessionIDInfor(req);
        HttpSession session = req.getSession(false);
        if (session != null) {
            Object info = session.getAttribute(ParameterConstants.__MPCACHEID__);
            if (info != null) {
                Map<String, Object> tMap = (Map<String, Object>) info;
                map.putAll(tMap);
                session.removeAttribute(ParameterConstants.__MPCACHEID__);
            }
        }

        return map;
    }

    /**
     * HTTPServletRequest,Map,,SessionIDInfor
     *
     * @param req  q
     * @return map  c
```

```
     */
    public static Map<String, Object> parameters4SessionIDInfor(HttpServletRequest req) {
        Map<String, Object> parameterMap = new HashMap<String, Object>();
        if (req == null) {
            return parameterMap;
        }
        ExtraClassManagerProvider pluginProvider = PluginModule.getAgent(PluginModule.ExtraCore);
        RequestParameterCollector collector = null;
        if (pluginProvider != null) {
            collector = pluginProvider.getSingle(RequestParameterCollector.XML_TAG);
        }
        if (collector == null) {
            collector = DefaultRequestParameterCollector.getInstance();
        }
        parameterMap.putAll(collector.getParametersFromSession(req));
        parameterMap.putAll(collector.getParametersFromAttribute(req));
        parameterMap.putAll(collector.getParametersFromParameter(req));
        parameterMap.putAll(collector.getParametersFromJSON(req, parameterMap));
        parameterMap.putAll(collector.getParametersFromReqInputStream(req));

        // denny:
        parameterMap.put(Constants.__LOCALE__, WebUtils.getLocale(req));

        // carl:urlsession
        //
        if (parameterMap.get(ParameterConstants.REDIRECT_FROM) == null) {
            String[] FILTER_PARAMS = BaseSessionFilterParameterManager.getFilterParameters();
            for (int i = 0; i < FILTER_PARAMS.length; i++) {
                parameterMap.remove(FILTER_PARAMS[i]);
            }
        }
        SimpleUser user = getCurrentSimpleUserFromRequest(req);
        if (user == null) {
            return parameterMap;
        } else {
            dealWithDetailUserInfo(user, UserInfoHandleFactory.getUserDetailInfo(), parameterMap);
        }
        return parameterMap;
    }

    /**
     * map
     */
    private static void dealWithDetailUserInfo(SimpleUser user, UserDetailInfo userDetailInfoHandler,
Map<String, Object> parameterMap) {
        String username = user.getUsername();
        String displayName = user.getDisplayName();
        parameterMap.put(ParameterConstants.FINE_USERNAME, username);
        parameterMap.put(ParameterConstants.FINE_OLD_USERNAME, username);
        parameterMap.put(ParameterConstants.FINE_DISPLAY_NAME, displayName);
        if (userDetailInfoHandler != null) {
            FArray depAndPost = userDetailInfoHandler.getDepartmentAndPost(username);
            FArray customRole = userDetailInfoHandler.getCustomRole(username);
            if (depAndPost != null) {
                parameterMap.put(ParameterConstants.FINE_POSITION, depAndPost);
                parameterMap.put(ParameterConstants.FINE_OLD_POSITION, depAndPost);
            }
            if (customRole != null) {
                parameterMap.put(ParameterConstants.FINE_ROLE, customRole);
                parameterMap.put(ParameterConstants.FINE_OLD_ROLE, customRole);
            }
        }
    }

    /**
     * {@link ServerConfig#isTokenFromCookie()}truecookie
     * Parameter(REPORT-15581 TokenResource.HEADER) -> Header -> Cookie
     * @param req request
     * @return token
     */
    private static String getToken(HttpServletRequest req) {
```

```
    String token;
    if (ServerConfig.getInstance().isTokenFromCookie()) {
        token = NetworkHelper.getTokenFromCookie(req);
        if (!StringUtils.isEmpty(token)) {
            return token;
        }
    }
    token = NetworkHelper.getTokenFromParameter(req);
    if (!StringUtils.isEmpty(token)) {
        return token;
    }
    token = NetworkHelper.getTokenFromHeader(req);
    return StringUtils.isEmpty(token) ? NetworkHelper.getTokenFromCookie(req) : token;
}

/**
 *
 *
 * @param request http
 * @return
 */
private static SimpleUser getCurrentSimpleUserFromRequest(HttpServletRequest request) {
    try {
        String token = getToken(request);
        if (StringUtils.isNotBlank(token)) {
            Claims claims = JwtUtils.parseJWT(token);
            return SimpleUser.create(claims.getSubject(), claims.getDescription());
        }
    } catch (Exception e) {
        FineLoggerFactory.getLogger().info("can't find username from request: {}", e.getMessage());
    }
    return null;
}


/**
 * HTTPURL
 *
 * @param req HTTP
 * @return URL
 */
public static String getOriginalURL(HttpServletRequest req) {
    return NetworkHelper.getOriginalURL(req);
}

/**
 * IP
 *
 * @param req HTTP
 * @return IP
 */
public static String getIpAddr(HttpServletRequest req) {
    String ip = req.getHeader("x-forwarded-for");
    if (StringUtils.isEmpty(ip) || "unknown".equalsIgnoreCase(ip)) {
        ip = req.getHeader("Proxy-Client-IP");
    }
    if (StringUtils.isEmpty(ip) || "unknown".equalsIgnoreCase(ip)) {
        ip = req.getHeader("WL-Proxy-Client-IP");
    }
    if (StringUtils.isEmpty(ip) || "unknown".equalsIgnoreCase(ip)) {
        ip = req.getRemoteAddr();
    }
    //nullreturn, Apache,SquidIP
    return StringUtils.isEmpty(ip) ? "unknown" : ip;
}

/**
 * IP
 *
 * @param ip IP
 * @return truefalse
```

```
     */
    public static boolean isLocalHost(String ip) {
        return "127.0.0.1".equals(ip) || "localhost".equals(ip) || (ip != null && ip.startsWith("0:0:0:0:0:0:0:
1"));
    }

    /**
     * HTTP
     *
     * @param req q
     * @return y
     */
    public static int receivePageNumber(HttpServletRequest req) {
        int pn = 0;
        String pnValue = WebUtils.getHTTPRequestParameter(req, "pn");
        if (pnValue != null) {
            Number num = Utils.string2Number(pnValue);
            // alex:numnullintValue,,pnInteger.MAX_VALUE
            if (num == null || (pn = num.intValue()) < 0) {
                pn = Integer.MAX_VALUE;
            }
        }
        return pn;
    }

    /**
     *
     *
     * @param req HTTP
     * @return
     */
    public static Device getDevice(HttpServletRequest req) {
        return NetworkHelper.getDevice(req);
    }

    public static Format getResponseContentFormat(HttpServletRequest req) {
        return NetworkHelper.getResponseContentFormat(req);
    }

    /**
     * req
     *
     * @param req HTTP
     * @return
     * @date 2014-12-1-4:39:44
     */
    public static Locale getLocale(HttpServletRequest req) {
        if (req == null) {
            return GeneralContext.getLocale();
        }

        //fr_locale, js__fr__locale__
        String localeStr = WebUtils.getHTTPRequestParameter(req, Constants.__FR_LOCALE__);
        localeStr = localeStr == null ? WebUtils.getHTTPRequestParameter(req, "__fr_locale__") : localeStr;

        //, .
        if (StringUtils.isEmpty(localeStr) && "zh_CN".equals(String.valueOf(req.getLocale()))) {
            return Locale.CHINA;
        }

        return GeneralUtils.createLocale(replaceScript4Xss(localeStr), req.getLocale());
    }

    /**
     * Importcss, html
     *
     * @param ca
     * @return html
     * @date 2014-12-3-7:45:14
     */
    public static String getCSSLinks(Calculator ca) {
```

```
    return getCSSLinks(ca, null);
}

/**
 * Importcss, html
 *
 * @param ca
 * @param attr
 * @return html
 * @date 2014-12-3-7:45:14
 */
public static String getCSSLinks(Calculator ca, ImportJsCssProvider attr) {
    String[] csses = getImportedCSS(attr);
    StringBuilder cssBuffer = new StringBuilder();
    for (String importString : csses) {
        try {
            //
            importString = TemplateUtils.render(importString, ca);
            cssBuffer.append("<link rel=\"stylesheet\" type=\"text/css\" href=\"");
            if (IOUtils.isOnlineUrl(importString)) {
                cssBuffer.append(importString);
            } else {
                cssBuffer.append(SateVariableManager.get(WebServerConstants.FINE_CONTEXT_PATH))
                        .append(importString.startsWith(CoreConstants.SEPARATOR)
                                ? importString : CoreConstants.SEPARATOR + importString);
            }
            cssBuffer.append("\"></link>").append('\n');
        } catch (Exception e) {
            //  doNothing
        }
    }
    return cssBuffer.toString();
}

/**
 * Importjs, javascript
 *
 * @param ca
 * @return javascript
 * @date 2014-12-3-7:45:14
 */
public static String getJSLinks(Calculator ca) {
    return getJSLinks(ca, null);
}

/**
 * Importjs, javascript
 *
 * @param ca
 * @param attr
 * @return javascript
 * @date 2014-12-3-7:45:14
 */
public static String getJSLinks(Calculator ca, ImportJsCssProvider attr) {
    return getJSLinks(ca, attr, false);
}

/**
 * Importjs, htmljavascript
 *
 * @param ca
 * @param attr
 * @param isHtmlTag html
 * @return htmljavascript
 * @date 2014-12-3-7:45:14
 */
public static String getJSLinks(Calculator ca, ImportJsCssProvider attr, boolean isHtmlTag) {
    String[] jses = getImportedJS(attr);
    StringBuilder jsBuffer = new StringBuilder();
    for (String importString : jses) {
        try {
```

```
            //
            importString = TemplateUtils.render(importString, ca);
            jsBuffer.append("<script type=\"text/javascript\" src=\"");
            if (IOUtils.isOnlineUrl(importString)) {
                jsBuffer.append(importString);
            } else {
                jsBuffer.append(SateVariableManager.get(WebServerConstants.FINE_CONTEXT_PATH))
                        .append(importString.startsWith(CoreConstants.SEPARATOR)
                                ? importString : CoreConstants.SEPARATOR + importString);
            }
            jsBuffer.append("\"></script>").append('\n');
        } catch (Exception e) {
            //  doNothing
        }
    }
    return isHtmlTag ? jsBuffer.toString() : CodeUtils.javascriptEncode(jsBuffer.toString());
}

/**
 * css
 *
 * @param attr
 * @return css
 * @date 2014-12-3-7:47:10
 */
private static String[] getImportedCSS(ImportJsCssProvider attr) {
    java.util.Set<String> set = new java.util.LinkedHashSet<String>(); // Set
    ImportJsCssProvider globalAttr = getGlobalWebAttr();

    if (globalAttr != null) {
        set.addAll(Arrays.asList(globalAttr.getCSSImport()));
    }

    if (attr != null) {
        set.addAll(Arrays.asList(attr.getCSSImport()));
    }

    return set.toArray(new String[0]);
}

/**
 * web
 *
 * @return web
 * @date 2014-12-4-8:51:38
 */
private static ImportJsCssProvider getGlobalWebAttr() {
    Class globalAttr = StableFactory.getMarkedClass(ImportJsCssProvider.XML_TAG, ImportJsCssProvider.class);
    return (ImportJsCssProvider) ConfigContext.getConfigInstance(globalAttr);
}

/**
 * js
 *
 * @param attr
 * @return js
 * @date 2014-12-3-7:47:10
 */
private static String[] getImportedJS(ImportJsCssProvider attr) {
    java.util.Set<String> set = new java.util.LinkedHashSet<String>(); // Set
    ImportJsCssProvider globalAttr = getGlobalWebAttr();

    if (globalAttr != null) {
        set.addAll(Arrays.asList(globalAttr.getJSImport()));
    }

    if (attr != null) {
        set.addAll(Arrays.asList(attr.getJSImport()));
    }
    return set.toArray(new String[0]);
}
```

```
}
```

NetworkHelper.java

```
package com.fr.data;

import com.fr.base.ServerConfig;
import com.fr.base.TemplateUtils;
import com.fr.general.ComparatorUtils;
import com.fr.general.GeneralUtils;
import com.fr.general.http.HttpClient;
import com.fr.general.web.ParameterConstants;
import com.fr.plugin.injectable.PluginModule;
import com.fr.stable.CodeUtils;
import com.fr.stable.EncodeConstants;
import com.fr.stable.StringUtils;
import com.fr.stable.fun.PrintWriterProcessor;
import com.fr.stable.fun.RequestParameterHandler;
import com.fr.stable.fun.ServletURLTransformer;
import com.fr.stable.plugin.ExtraClassManagerProvider;
import com.fr.stable.web.Device;
import com.fr.stable.web.Format;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

/**
 * @author richie
 * @date 14/11/17
 * @since 8.0
 */
public class NetworkHelper {
    /**
     *
     */
    @SuppressWarnings("WeakerAccess")
    public static final String POST_EMPTY_SESSION = "POST_EMPTY_SESSION";

    private static final String DEPRECATED_DEVICE_KEY = "__device__";
    private static final String DEVICE_KEY = "deviceType";

    /**
     * Servelt
     *
     * @param req HTTP
     * @return URL
     * @date 2014-9-25-7:46:46
     */
    public static String createServletURL(HttpServletRequest req) {
        return createServletURL(req, ServerConfig.getInstance().getServletName());
    }

    /**
     * Servelt
     *
     * @param req HTTP
     * @return URL
     * @date 2014-9-25-7:46:46
```

```
 */
public static String createServletURL(HttpServletRequest req, String servletName) {
    StringBuilder urlBuf = new StringBuilder();

    // peter:
    // IP
    // IP
    urlBuf.append(req.getContextPath());
    urlBuf.append("/").append(servletName);

    ExtraClassManagerProvider extra = PluginModule.getAgent(PluginModule.ExtraCore);
    if (extra != null) {
        Set<ServletURLTransformer> set = extra.getArray(ServletURLTransformer.XML_TAG);
        for (ServletURLTransformer transformer : set) {
            if (transformer.accept(req)) {
                return transformer.transform(req, urlBuf);
            }
        }
    }
    return urlBuf.toString();
}

/**
 * req InputStream
 *
 * @param req HTTP
 * @return
 * @date 2014-9-29-3:21:14
 */
public static InputStream getRequestInputStream(HttpServletRequest req) {
    //reqREQ_IN
    byte[] bytes = (byte[]) req.getAttribute(DataBaseUtils.REQ_IN);
    if (bytes == null || bytes.length == 0) {
        //
        if (req.getAttribute(HttpClient.CLOSED) == null) {
            try {
                //req
                return req.getInputStream();
            } catch (IOException e) {
                throw new RuntimeException("Network transfer exception", e);
            }
        } else {
            bytes = new byte[0];
        }
    }
    return new ByteArrayInputStream(bytes);
}

/**
 * HTTP
 *
 * @param req      HTTP
 * @param paraName
 * @return
 */
public static String getHTTPRequestParameter(HttpServletRequest req, String paraName) {
    /*
     *  BrowsercjkEncode
     * cjkEncode
     * paraNamecjkEncodereq
     */
    return getHTTPRequestEncodeParameter(req, paraName, true);
}

/**
 *
 *
 * @param req       HTTP
 * @param paraNames
 * @return
 */
```

```
public static String getHTTPRequestParameter(HttpServletRequest req, String... paraNames) {
    if (paraNames != null) {
        for (String paraName : paraNames) {
            String result = getHTTPRequestParameter(req, paraName);
            if (result != null) {
                return result;
            }
        }
    }
    return StringUtils.EMPTY;
}

/**
 * HTTP
 *
 * @param req      HTTP
 * @param paraName
 * @return boolean
 */
public static boolean getHTTPRequestBoolParameter(HttpServletRequest req, String paraName) {
    return Boolean.parseBoolean(NetworkHelper.getHTTPRequestParameter(req, paraName));
}

/**
 * HTTP-1
 *
 * @param req      HTTP
 * @param paraName
 * @return int
 */
public static int getHTTPRequestIntParameter(HttpServletRequest req, String paraName) {
    return getHTTPRequestIntParameter(req, paraName, -1);
}

/**
 * HTTP
 *
 * @param req          HTTP
 * @param paraName
 * @param defaultValue
 * @return reqdefaultValue
 */
public static int getHTTPRequestIntParameter(HttpServletRequest req, String paraName, int defaultValue) {

    if (req == null) {
        return defaultValue;
    }
    String parameterValue = getHTTPRequestParameter(req, paraName);
    Number num = GeneralUtils.string2Number(parameterValue);
    if (num == null) {
        return defaultValue;
    }
    return num.intValue();
}

/**
 * sessionID
 *
 * @param req HTTP
 * @return session
 */
public static String getHTTPRequestSessionIDParameter(HttpServletRequest req) {
    return getHTTPRequestParameter(req, ParameterConstants.SESSION_ID);
}

/**
 *
 *
 * @param req HTTP
 * @return
 */
```

```
    public static String getHTTPRequestFileNameParameter(HttpServletRequest req) {
        return getHTTPRequestParameter(req, ParameterConstants.__FILENAME__);
    }

    /**
     * HTTP
     *
     * @param req
     * @param paraName
     * @param encode    url-decode
     * @return String
     */
    public static String getHTTPRequestEncodeParameter(HttpServletRequest req, String paraName, boolean encode)
{

        ExtraClassManagerProvider provider = PluginModule.getAgent(PluginModule.ExtraCore);
        RequestParameterHandler handler;
        if (provider == null) {
            handler = DefaultRequestParameterHandler.getInstance();
        } else {
            handler = provider.getSingle(RequestParameterHandler.XML_TAG);
            if (handler == null) {
                handler = DefaultRequestParameterHandler.getInstance();
            }
        }
        Object returnValue = handler.getParameterFromHeader(req, paraName);
        if (returnValue == null) {
            returnValue = handler.getParameterFromRequest(req, paraName);
        }
        if (returnValue == null) {
            returnValue = handler.getParameterFromAttribute(req, paraName);
        }
        // alex:reqtypeSensitive
        if (returnValue == null) {
            returnValue = handler.getParameterFromJSONParameters(req, paraName);
        }
        //wei : bug10637,session
        if (returnValue == null) {
            returnValue = handler.getParameterFromSession(req, paraName);
        }

        if (returnValue == null) {
            // p:decode.[4554], name.
            // alex:[9853]req
            paraName = CodeUtils.cjkEncode(paraName);
            returnValue = handler.getParameterFromRequest(req, paraName);
            if (returnValue == null) {
                returnValue = handler.getParameterFromAttribute(req, paraName);    // peter:Struts,.
                if (returnValue == null) {
                    returnValue = handler.getParameterFromSession(req, paraName);
                }
            }
        }

        return encode ? checkURLDecode(returnValue) : GeneralUtils.objectToString(returnValue);
    }

    /**
     * post, session, sessionid, return null.
     * return POST_EMPTY_SESSION, sessionid != null && xxxxx(sessionid)
     *
     * @param returnValue
     * @return  {@link NetworkHelper#POST_EMPTY_SESSION}
     */
    @SuppressWarnings("WeakerAccess")
    protected static boolean isPostEmptySession(Object returnValue) {
        return POST_EMPTY_SESSION.equals(returnValue);
    }

    private static String checkURLDecode(Object str) {
        if (str == null) {
```

```
            return null;
        }

        //urlencodeURIComponent(encodeURIComponent("+"))
        //decodeText //bug
        String temp = CodeUtils.decodeText(String.valueOf(str));

        try {
            //8125 29122 jsencodeURIComponent(utf-8)
            // neil:encode, ascii(STR_ENC1), encodeweb,
            // req.getParameter GBK  UTF-8  ISO-8859-1  [STR_ENC1],
            // jsencode, utf-8(tomcat), , javaURLDecoder.decode
            return URLDecoder.decode(temp, EncodeConstants.ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
            return null;
        } catch (IllegalArgumentException e) {
            //URLDecoder
            return temp;
        }
    }

    /**
     * HTTPURL
     *
     * @param req HTTP
     * @return URL
     */
    public static String getOriginalURL(HttpServletRequest req) {
        return getOriginalURL(req, true);
    }

    /**
     * HTTPURL
     *
     * @param req               req HTTP
     * @param decodeQueryString queryString
     * @return URL
     */
    public static String getOriginalURL(HttpServletRequest req, boolean decodeQueryString) {
        return getOriginal(req, true, decodeQueryString);
    }

    /**
     * uri()
     *
     * @param request           http
     * @param decodeQueryString
     * @return String  uri
     */
    @SuppressWarnings("WeakerAccess")
    public static String getOriginalURI(HttpServletRequest request, boolean decodeQueryString) {
        return getOriginal(request, false, decodeQueryString);
    }

    private static String getOriginal(HttpServletRequest req, boolean withPrefix, boolean decodeQueryString) {
        if (req == null) {
            return "";
        }
        StringBuffer sb = withPrefix ? req.getRequestURL() : new StringBuffer(req.getRequestURI());
        Map pMap = req.getParameterMap();
        Iterator itr = pMap.entrySet().iterator();
        boolean isFirst = !sb.toString().contains("?");
        while (itr.hasNext()) {
            Map.Entry entry = (Map.Entry) itr.next();
            if (isFirst) {
                sb.append('?');
                isFirst = false;
            } else {
                sb.append('&');
            }
            sb.append(entry.getKey().toString());
```

```
            sb.append('=');
            sb.append(getHTTPRequestEncodeParameter(req, entry.getKey().toString(), decodeQueryString));
        }

        // encode, cjk([])encodeURIComponent(url&a=123),
        // url, encodeURI.
        return sb.toString();
    }

    /**
     *
     *
     * @param resource
     * @param response HTTP
     * @param map
     * @throws java.io.IOException e
     */
    public static void writeOutTemplate(String resource, HttpServletResponse response, Map map) throws
IOException {
        PrintWriter writer = createPrintWriter(response);
        TemplateUtils.dealWithTemplate(resource, writer, map);
        writer.flush();
        writer.close();
    }

    /**
     *
     *
     * @param res HTTP
     * @return
     * @throws java.io.IOException e
     */
    public static PrintWriter createPrintWriter(HttpServletResponse res) throws IOException {
        return createPrintWriter(res, ServerConfig.getInstance().getServerCharset());
    }

    /**
     *
     *
     * @param res         HTTP
     * @param charsetName
     * @return
     * @throws java.io.IOException e
     */
    public static PrintWriter createPrintWriter(HttpServletResponse res, String charsetName) throws IOException
{
        PrintWriterProcessor processor = null;
        ExtraClassManagerProvider provider = PluginModule.getAgent(PluginModule.ExtraCore);
        if (provider != null) {
            processor = provider.getSingle(PrintWriterProcessor.MARK_STRING);
        }
        if (processor == null) {
            processor = DefaultPrintWriterProcessor.getInstance();
        }
        return processor.createPrintWriter(res, charsetName);
    }

    /**
     *
     *
     * @param res HTTP
     */
    public static void setCacheSettings(HttpServletResponse res) {
        //marks:ie6
        //marks:cachehttps
        // carl:3bug0004207.
        res.setHeader("Cache-Control", "public");
        res.setHeader("Cache-Control", "max-age=3");
        // reset,buffer.
        // Weblogicbuffer.
        res.reset();
```

```
}

/**
 *
 *
 * @param req HTTP
 * @return
 */
public static Device getDevice(HttpServletRequest req) {
    String device = getDeviceType(req);
    if (StringUtils.isNotBlank(device)) {
        return Device.parse(device);
    }
    //wei: ,,.
    return Device.parse(NetworkHelper.getHTTPRequestParameter(req, DEPRECATED_DEVICE_KEY));
}

public static Format getResponseContentFormat(HttpServletRequest req) {
    Device device = getDevice(req);
    if (device.isMobile()) {
        return Format.JSON;
    }
    return Format.parse(NetworkHelper.getHTTPRequestParameter(req, "__format__"));
}

/**
 * Cookietoken
 * @param request request
 * @return token
 */
public static String getTokenFromCookie(HttpServletRequest request) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if (ComparatorUtils.equals(cookie.getName(), ParameterConstants.TOKEN_NAME)) {
                return cookie.getValue();
            }
        }
    }
    return null;
}

/**
 * Header Authorizationtoken
 * @param request request
 * @return token
 */
public static String getTokenFromHeader(HttpServletRequest request) {
    String authorization = request.getHeader(ParameterConstants.AUTHORIZATION_HEADER);
    if (StringUtils.isNotEmpty(authorization)) {
        return authorization.substring(ParameterConstants.AUTHORIZATION_PREFIX.length());
    } else {
        return StringUtils.EMPTY;
    }
}

/**
 * Parametertoken
 * @param request request
 * @return token
 */
public static String getTokenFromParameter(HttpServletRequest request) {
    return getHTTPRequestParameter(request, ParameterConstants.TOKEN_NAME);
}

private static String getDeviceType(HttpServletRequest req) {
    String deviceType = req.getHeader(DEVICE_KEY);
    if (StringUtils.isNotBlank(deviceType)) {
        return deviceType;
    }
    deviceType = (String) req.getAttribute(DEVICE_KEY);
```

```
        if (StringUtils.isNotBlank(deviceType)) {
            return deviceType;
        }
        deviceType = NetworkHelper.getHTTPRequestParameter(req, DEVICE_KEY);
        if (StringUtils.isNotBlank(deviceType)) {
            return deviceType;
        }
        deviceType = req.getHeader(DEPRECATED_DEVICE_KEY);
        if (StringUtils.isNotBlank(deviceType)) {
            return deviceType;
        }
        return (String) req.getAttribute(DEPRECATED_DEVICE_KEY);
    }

}
```

//

WebUtils

com.fr.stable.fun.RequestParameterHandler