支付系统中的常用工具

作者：Sysecho
原文链接：https://ld246.com/article/1498744223151
来源网站：链滴

- StringUtils.java

  处理常用字符串：判断是否为空isEmpty(String value);

  按字典排序并拼接参数：createLinkString(Map params);

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;

public class StringUtils {
    /**
     * 判断字符数组是否为空
     */
    public static boolean areNotEmpty(String... values) {
        boolean result = true;
        if (values == null || values.length == 0) {
            result = false;
        } else {
            for (String value : values) {
                result &= !isEmpty(value);
            }
        }
        return result;
    }

    /**
     * 把数组所有元素排序，并按照"name1=value1&name2=value2..."字符拼接成字符串
     *
     * @param params
     *            需要排序并参与字符拼接的参数组
     * @return 拼接后字符串
     */
    public static String createLinkString(Map<String, String> params) {

        List<String> keys = new ArrayList<String>(params.keySet());
        Collections.sort(keys);

        String prestr = "";

        for (int i = 0; i < keys.size(); i++) {
            String key = keys.get(i);
            String value = params.get(key);

            if (i == keys.size() - 1) {
                prestr = prestr + key + "=" + value;
            } else {
                prestr = prestr + key + "=" + value + "&";
            }
        }

        return prestr;
```

```
    }

    /**
     * 判断字符串是否为空
     * <ul>
     * <li>isEmpty(null) = true</li>
     * <li>isEmpty("") = true</li>
     * <li>isEmpty("   ") = true</li>
     * <li>isEmpty("abc") = false</li>
     * </ul>
     *
     * @param value
     *            目标字符串
     * @return true/false
     */
    public static boolean isEmpty(String value) {
        int strLen;
        if (value == null || (strLen = value.length()) == 0) {
            return true;
        }
        for (int i = 0; i < strLen; i++) {
            if ((Character.isWhitespace(value.charAt(i)) == false)) {
                return false;
            }
        }
        return true;
    }
}
```

- 支付系统中的签名和验签SignUtils.java

```
import java.io.InputStream;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.apache.commons.codec.binary.Base64;
/**
```

-

```
*/
```

```
public class SignUtils {


/**  */
// 缓存公钥和私钥
public static Map<String, Object> certMap = new java.util.concurrent.ConcurrentHashMap<S
ring, Object>();

/**
 *
 *
 * @param sArray
 * @return
 */
public static Map<String, String> paraFilter(Map<String, String> sArray) {

    Map<String, String> result = new HashMap<String, String>();

    if (sArray == null || sArray.size() <= 0) {
        return result;
    }

    for (String key : sArray.keySet()) {
        String value = sArray.get(key);
        if (value == null || StringUtils.isEmpty(value)
                || key.equalsIgnoreCase("sign")) {
            continue;
        }
        result.put(key, value);
    }

    return result;
}


/**
 *
 *
 * @param sortedParams
 * @return
 */
public static String getSignContent(Map<String, String> sortedParams) {
    StringBuffer content = new StringBuffer();
    List<String> keys = new ArrayList<String>(sortedParams.keySet());
    Collections.sort(keys);
    int index = 0;
    for (int i = 0; i < keys.size(); i++) {
        String key = keys.get(i);
        String value = sortedParams.get(key);
        if (StringUtils.areNotEmpty(key, value)) {
            content.append((index == 0 ? "" : "&") + key + "=" + value);
            index++;
        }
    }
```

```
    return content.toString();
}


/**
 *
 * 签名
 * @param params 业务参数
 * @param charset 编码
 * @param pfxCertFileInputStream 证书输入流
 * @param rsaPassword  私钥pkcs12证书密码
 * @param algorithmName rsa算法名
 * @return
 * @throws Exception
 */
public static String rsaSign(Map<String, String> params, String charset,
        InputStream pfxCertFileInputStream,String rsaPassword, String algorithmName) throws E
ception {
    String signContent = getSignContent(params);

    return rsaSign(signContent, charset, pfxCertFileInputStream,rsaPassword, algorithmName);
}

/**
 *
 *
 * @param content
 * @param charset
 * @param pfxCertFileInputStream
 * @param rsaPassword
 * @param algorithmName
 * @return
 * @throws Exception
 */
public static String rsaSign(String content, String charset,
        InputStream pfxCertFileInputStream,String rsaPassword, String algorithmName) throws E
ception {
    try {
        PrivateKey priKey = getPrivateKeyFromPKCS12(
                rsaPassword,
                pfxCertFileInputStream);

        java.security.Signature signature = java.security.Signature
                .getInstance(algorithmName);

        signature.initSign(priKey);

        if (StringUtils.isEmpty(charset)) {
            signature.update(content.getBytes());
        } else {
            signature.update(content.getBytes(charset));
        }

        byte[] signed = signature.sign();
```

```
            String sign = new String(Base64.encodeBase64(signed), charset);


            return sign;
        } catch (Exception e) {
            throw new Exception("RSAcontent = " + content + "; charset = "
                    + charset, e);
        }
    }

    /**
     *
     *
     * @param publicCertFileInputStream
     * @param params
     * @param sign
     * @param charset
     * @param algorithmName
     * @return
     * @throws Exception
     */
    public static boolean rsaCheckContent(
            InputStream publicCertFileInputStream, Map<String, String> params,
            String sign, String charset,String algorithmName) throws Exception {
        String content = StringUtils.createLinkString(SignUtils
                .paraFilter(params));

        return rsaCheckContent(publicCertFileInputStream, content, sign,
                charset, algorithmName);
    }


    /**
     *
     *
     * @param publicCertFileInputStream
     * @param content
     * @param sign
     * @param charset
     * @param algorithmName
     * @return
     * @throws Exception
     */
    public static boolean rsaCheckContent(
            InputStream publicCertFileInputStream, String content, String sign,
            String charset, String algorithmName) throws Exception {
        boolean bFlag = false;
        try {
            java.security.Signature signetcheck = java.security.Signature
                    .getInstance(algorithmName);
            signetcheck
                    .initVerify(getPublicKeyFromCert(publicCertFileInputStream));
            signetcheck.update(content.getBytes(charset));
```

```
        if (signetcheck.verify(Base64.decodeBase64(sign.getBytes(charset)))) {
            bFlag = true;
            System.out.println("签名成功！");
        }else{
            System.out.println("签名失败！");
        }
    } catch (Exception e) {
        throw new Exception("验证签名异常");
    }

    return bFlag;
}

/**
 * 读取公钥，x509格式.
 *
 * @param ins
 * @return
 * @throws Exception
 */
public static PublicKey getPublicKeyFromCert(InputStream ins)
        throws Exception {
    PublicKey pubKey = (PublicKey) certMap.get("PublicKey");
    if (pubKey != null) {
        return pubKey;
    }

    try {
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        Certificate cac = cf.generateCertificate(ins);
        pubKey = cac.getPublicKey();
        certMap.put("PublicKey", pubKey);
    } catch (Exception e) {
        if (ins != null)
            ins.close();
        throw e;
    } finally {
        if (ins != null) {
            ins.close();
        }
    }

    return pubKey;
}

/**
 * 读取PKCS12格式的key（私钥）pfx格式.
 *
 * @param rsaPassword
 * @param ins
 * @return
 * @throws Exception
 */
public static PrivateKey getPrivateKeyFromPKCS12(String rsaPassword,
```

```
        InputStream ins) throws Exception {
    PrivateKey priKey = (PrivateKey) certMap.get("PrivateKey");
    if (priKey != null) {
        return priKey;
    }

    KeyStore keystoreCA = KeyStore.getInstance("PKCS12");
    try {
        // 读取CA根证书
        keystoreCA.load(ins, rsaPassword.toCharArray());

        Enumeration<?> aliases = keystoreCA.aliases();
        String keyAlias = null;
        if (aliases != null) {
            while (aliases.hasMoreElements()) {
                keyAlias = (String) aliases.nextElement();
                // 获取CA私钥
                priKey = (PrivateKey) (keystoreCA.getKey(keyAlias,
                        rsaPassword.toCharArray()));
                if (priKey != null) {
                    certMap.put("PrivateKey", priKey);
                    break;
                }
            }
        }
    } catch (Exception e) {
        if (ins != null)
            ins.close();
        throw e;
    } finally {
        if (ins != null) {
            ins.close();
        }
    }

    return priKey;
}

}
```