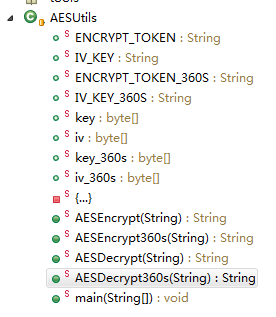
# 360 shuaji root中加密 解密总结

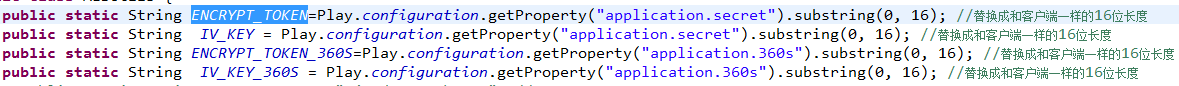
\*\*原则 数据量大 尽量使用**对称加密**来提高加密的效率 ---- 对称加密使用的算法主要是AES算法（变种就是 把密钥换一下 就是AES的360s版本 s:super）

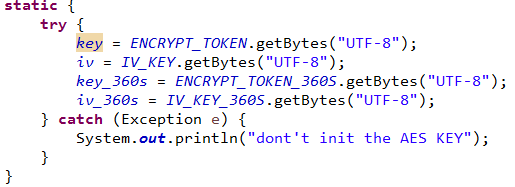
1. **加密解密使用的工具类介绍**

**(1). AESUtils.java**



**\*\*初始化内容**





**\*\*对应配置文件 ---- 测试服务配置**（生产和测试一致）

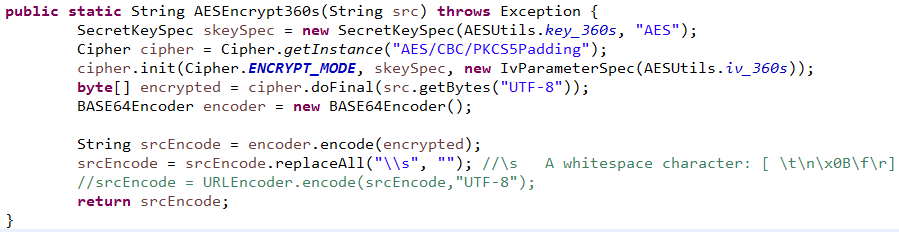


上面的application.secret是ENCRYPT\_TOKEN 是AES加密的密钥

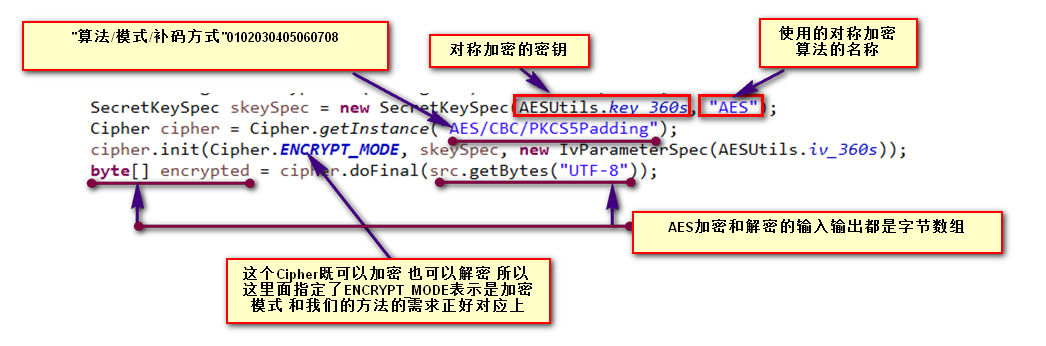
application.360s是AES360s加密的另一个密钥

**\*\*AESEncrypt360s**

---- AES360s加密的代码



--- 代码简要解释



【IvParameterSpec看做一个增强加密算法强度的参数就可以了】

后面几句代码：

因为AES加密算法的输入输出都是字节数组 所以 不能直接输出展示

这里面使用了Base64对二进制数据进行编码 编码之后 就变成了字符串 返回来

总结：

**我们项目中 AES加密的过程**：

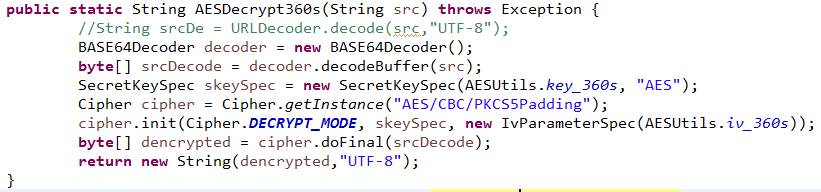
要加密的内容变成字节数组byte[] --->

使用标准的AES算法（加密模式）+密钥进行对称加密 获得加密结果 同样是byte[] --->

使用Base64对字节数组进行编码 获得最后的字符串形式表示 --->

去掉编码之后的空格字符 返回

**\*\*\*AESDecrypt360s**



**我们项目中 AES解密的过程**：

使用Base64对编码后的密文内容进行解码 获得字节数组byte[]--->

将要解密的内容使用标准的AES算法（解密模式）+密钥进行对称解密 获得明文 同样是byte[] --->

对字节数组byte[]直接使用UTF-8编码 返回

**(2). RSAUtil.java**

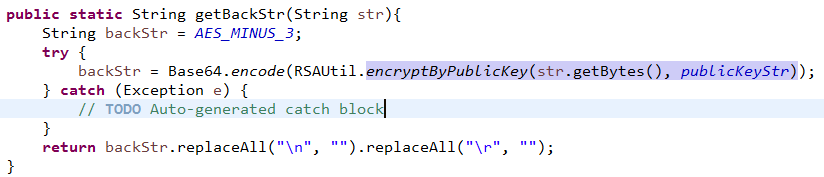




这个公钥和私钥是事先使用方法生成好的字符串 --- 加密解密的时候 直接使用就可以了

这个类中 我们用到两个方法：

**\*\*RSA加密方法**

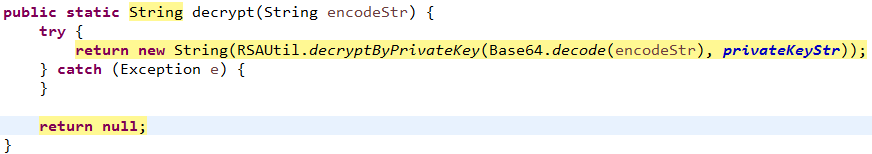


代码简介：

RSAUtil.encryptByPublicKey(str.getBytes(), publicKeyStr) ----- 就是使用公钥对输入的字符串转换的字节数组进行RSA算法加密(非对称加密)

加密之后 还是使用Base64进行编码 ---- 去掉回车和换行之后 返回给客户端

**\*\*RSA解密方法**



首先就是通过Base64进行解码 获得byte[]数组 ----- 前面是公钥加密 这里面就是私钥解密 最后 把获取到的byte[]编码为UTF-8字符串进行输出

**(3). 对称加密 非对称加密 单向加密的特点**

---- 对称加密：加密解密速度快 严重依赖密钥 容易被截获并被破解 导致信息泄露

---- DES DES3 AES RC4

---- 非对称加密：加密解密速度慢 由于公钥加密之后 需要私钥解密 私钥不公开 安全性好 ----- RSA DSA

---- 单向加密（单向Hash算法）：定长输出 不可逆 微小的输入变化会产生巨大的输出结果

----- MD5

1. **涉及到的业务逻辑的加密解密**

**\*\*接口的请求**

由于我们的接口是典型的高并发业务场景，请求是非常频繁的。所以接口接收到用户加密之后的请求应该迅速的解密用户的请求，因此加密的请求全部是AES加密（可能使用不同的密钥 所以 有一个AES加密解密 另一个是AES360s加密解密）

同时我们的用户请求数据没有那么重要，就算被破解，也没有太大的损失。

基于上述两点，我们的客户端使用AES加密请求服务端 ---- 服务端的接口使用AES解密获取用户的原始请求

**\*\*接口的响应**

这个根据产品来决定。一般情况 AES是对要传输的大量数据进行加密解密比较合适

RSA是对简短的内容 比如AES的加密的密钥进行加密来使用

但是 如果我们的返回内容非常重要 怕被破解，通常要使用RSA这种对称加密来进行

$$ /c/cr.html接口 ---- 360sCDN下发：仅仅使用AES加密返回密文结果

$$ /c/getroote 接口 ---- 获取关键字的root文件接口：仅仅使用RSA加密返回密文结果 ---- 这个里面可以做很多事情 重要程度不确定 所以 RSA加密

$$ /c/getsolutione 接口 ---- 获取密文方案：这个是ROOT的核心文件 --- 方案文件，所以 为了防止被竞品截获并破解，重要程度不言而喻，因此使用RSA加密返回密文

===== 前期使用的是AES对称加密 ----- 但是 被破解了 所以 历史原因 最后使用了AES+RSA两种方式进行加密

这三个接口返回的数据通常量都很大 ---- 所以 RSA会耗费一定的服务器性能

**\*\*缓存中使用的key**

我们很多接口由于判断的逻辑比较复杂，这样一个web应用服务器如果直接响应用户的请求，就无法满足大量的用户的请求，所以这个时候使用了缓存Memcache对用户的请求-响应进行缓存以提高服务器的性能。  
 Memcache使用的是key-value的形式进行数据缓存的，因此我们需要一个唯一的键值对来作为用户访问的key

但是有的时候我们的用户访问的参数可能过多，会导致我们直接缓存用户请求的明文过长。因此需要一种hash算法来对需要存入缓存的结果进行映射 ----- 我们的工程选用了MD5

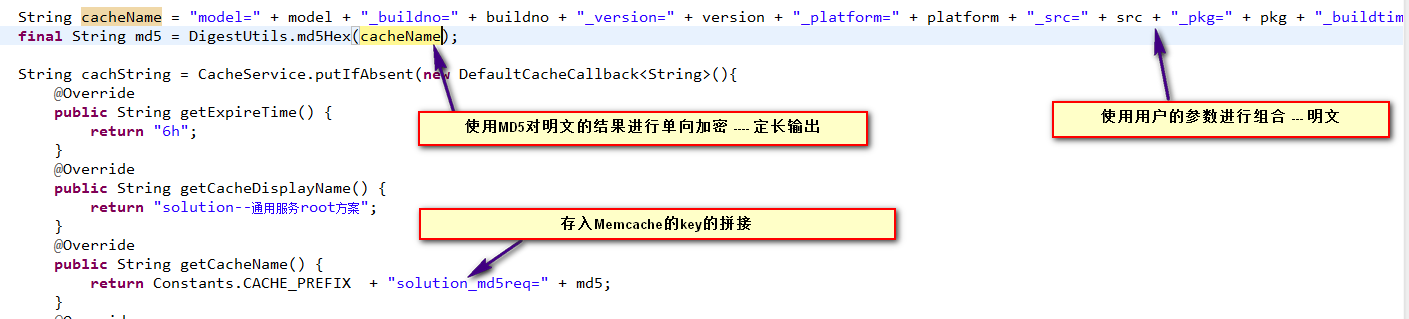
基于MD5这种单向加密的有雪崩效应+定长输出的特点 -----

雪崩效应 可以对我们的请求有微小的变化 但是 映射之后会有巨大的变换结果

定长输出 无论输入的请求和参数组合有多长 都会有定长输出 这样符合我们Memcache对key的长度限制要求

---- /c/getroote 和 /c/getsolutione 这两个接口使用了缓存

**下面是getsolutione这个接口使用缓存的key的构造情况**



1. **相关的业务总结加密解密代码参考（吕品协助）**

