# 阿里云Java SDK 使用手册

<https://help.aliyun.com/document_detail/52740.html?spm=a2c4g.11174283.3.2.3d9926d1LJhmT3>

# 《实验:使用KMS加密系统配置文件》

实验操作手册

### 目录

一、实验准备.................................................................................................................................3

1实验概述.................................................................................................................................3

2实验目的.................................................................................................................................3

3实验要求.................................................................................................................................3

二、实验步骤.................................................................................................................................3

1.开通密钥管理服务.................................................................................................................3

2.创建密钥.................................................................................................................................5

3.创建JAVA项目并下载配置SDK........................................................................................6

1)创建JAVA Maven项目...........................................................................................6

2)配置Maven依赖....................................................................................................9

3)查看下载的jar包..................................................................................................10

4.使用KMS SDK进行加解密...................................................................................................11

1)创建KMSSample类..................................................................................................11

2)定义全局常量...........................................................................................................12

3)添加加密方法kmsEncrypt.......................................................................................12

4)添加解密方法Kms Decrypt......................................................................................13

5)测试加解密方法.......................................................................................................14

5.直接加解密配置文件信息..................................................................................................15

实验:使用KMS加密系统配置文件实验操作手册

1)添加base64编码方法base64 encode...............................................................15

2)添加base64解码方法base64 decode...............................................................15

3)添加读写文件的方法writeFile和readFile........................................................15

4)加解密配置文件内容..........................................................................................16

6.通过数字信封加解密配置文件信息..............................................................................17

1)添加DES加密方法..............................................................................................17

2)添加DES解密方法desDecrypt...........................................................................18

3)获取数据密钥方法kmsGenerateDataKey...........................................................19

4)使用数据信封的方式加解密配置文件信息.......................................................20

三、附录..................................................................................................................................21

Pom.xml:...................................................................................................................................21

KMSSample：..........................................................................................................................23

实验:使用KMS加密系统配置文件实验操作手册

### 一、实验准备

1实验概述

本实验是基于阿里云的密钥管理服务，加解密网站配置文件中的配置信息。

2实验目的

1. 掌握阿里云的密钥管理服务的基本操作
2. 理解数字信封加密技术的基本过程
3. 使用信封加密技术在本地加密、解密配置文件信息

3实验要求

1)需具备基本的Java知识

2)需安装Java 集成开发环境Eclipse

注：实验中的相关配置、代码文本参见附录部分

### 二、实验步骤

### 1、开通密钥管理服务

如果已经开通阿里云密钥管理服务，可直接进入第二步开始创建密钥，如果未开通密钥服务，

那么按照下面说明进行开通。进入阿里云管理控制台，如果左侧菜单没有“密钥管理服务”，点击设置图标，如下图所示：



在弹出页面选择“监控与管理”，然后选择“密钥管理服务”，进行添加，如下所示：



点击左侧“密钥管理服务”菜单，在服务未开通的情况，弹出如下页面，点击“去开通”，如下图所示：



在密钥管理服务页面，点击“立即开通”，如下图所示：



服务正常开通后，显示如下图所示：



### 创建密钥

进入密钥管理服务控制台，点击“创建密钥”，如下图所示：



在创建密钥弹出框中输入描述信息，然后单击“确定”如下图所示

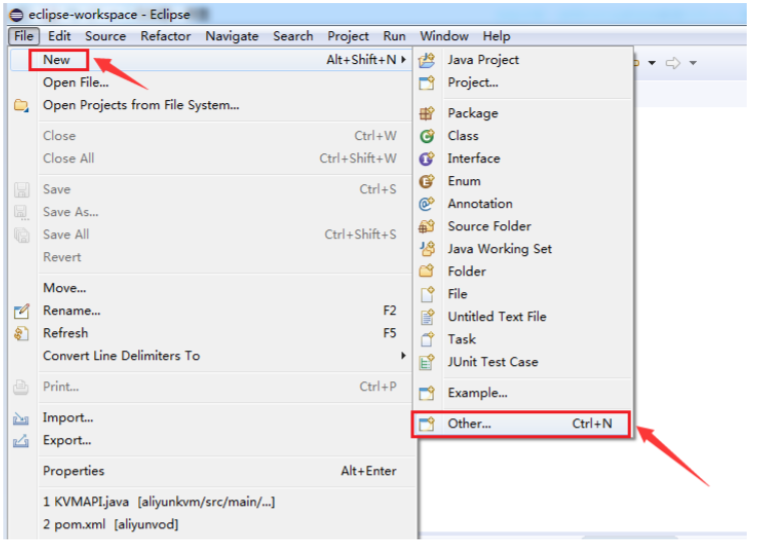


在密钥列表页面，可以查看到新创建密钥ID，创建日期，密钥状态，如下图所示

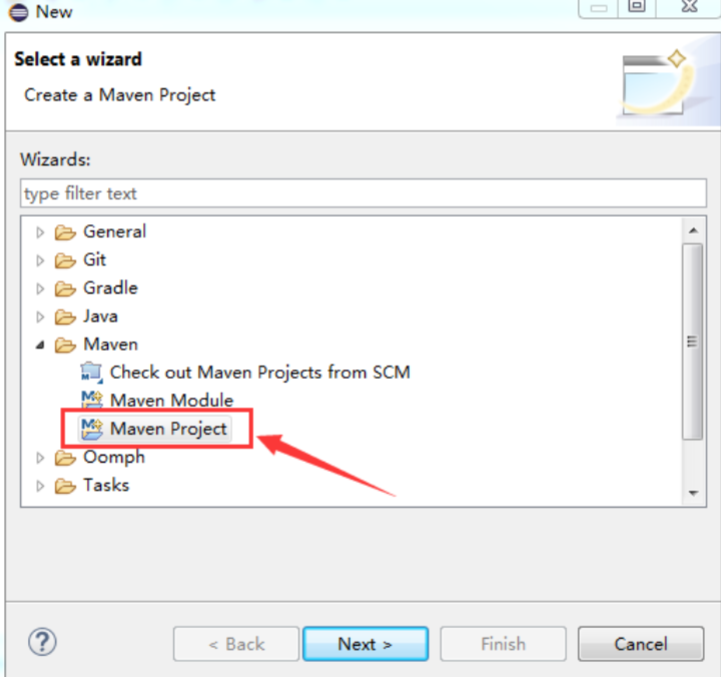


### 3、创建JAVA项目并下载配置SDK

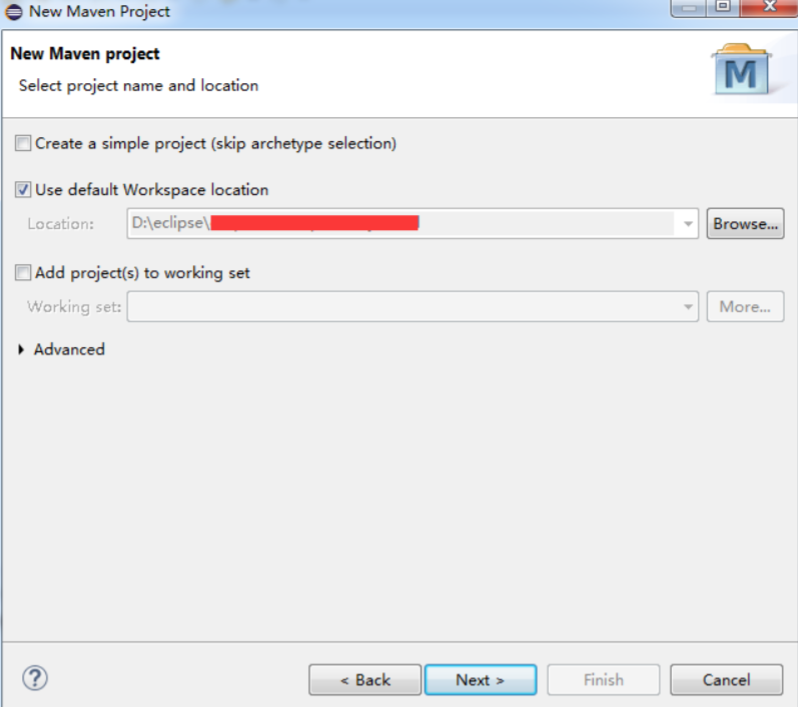
1)创建JAVA Maven项目打开eclipse，依次选择file>New>Other,如下图所示



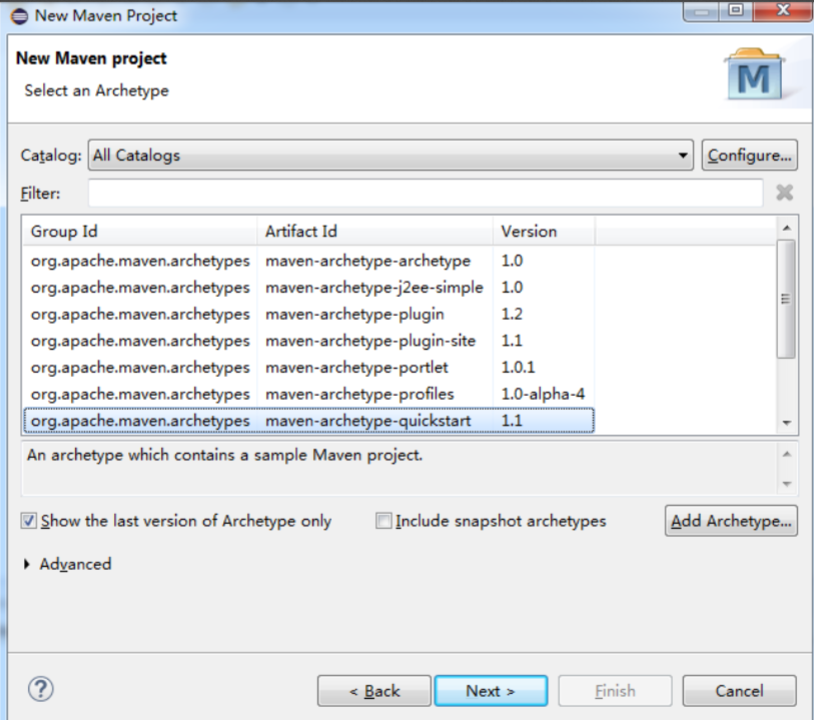
在弹出的对话框中，选择“Maven Project”，单击“Next”，如下图所示



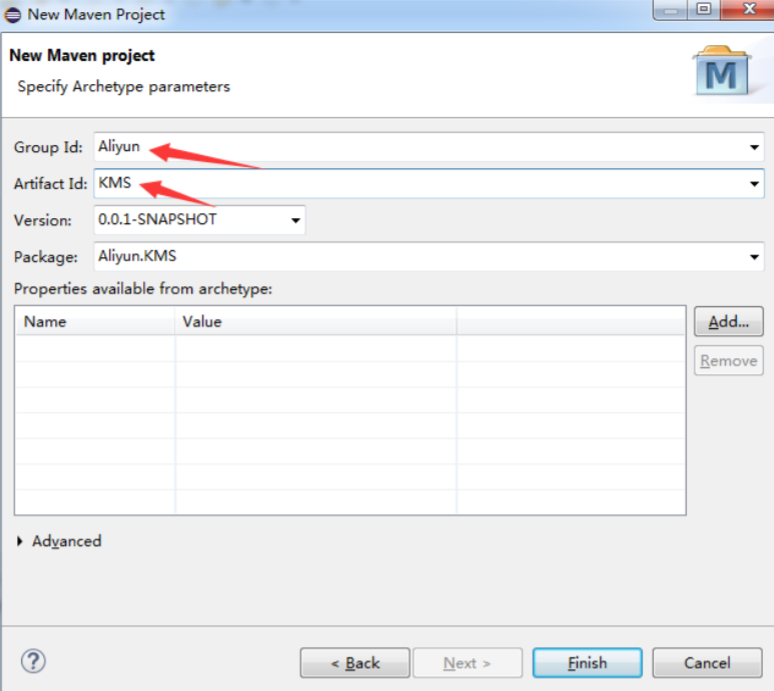
选择默认的工作空间，然后点击“Next”，如下图所示



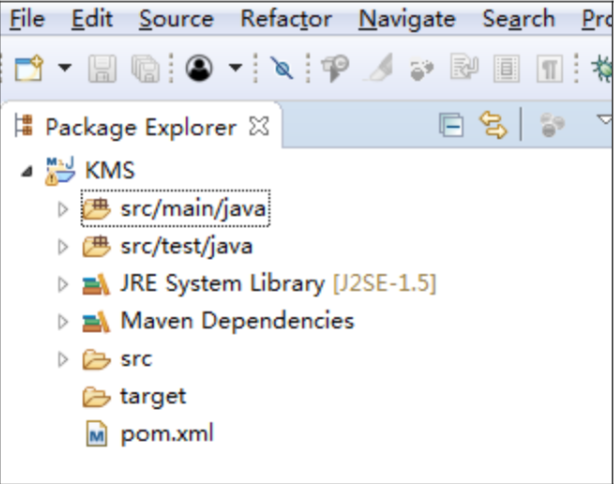
选择默认的配置，然后点击“Next”，如下图所示：



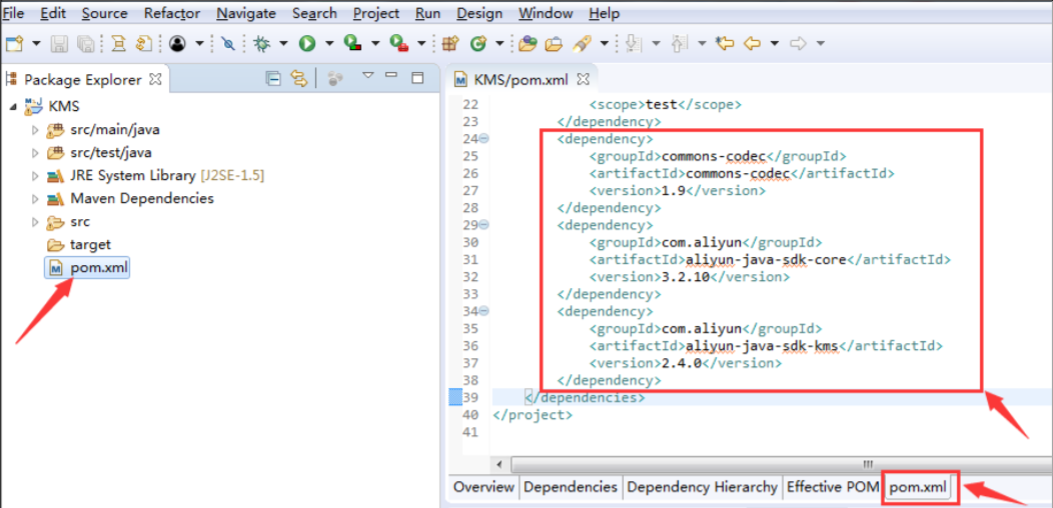
输入Group Id:Aliyun,ArtifactId:KMS,然后点击“Finish”，如下图所示



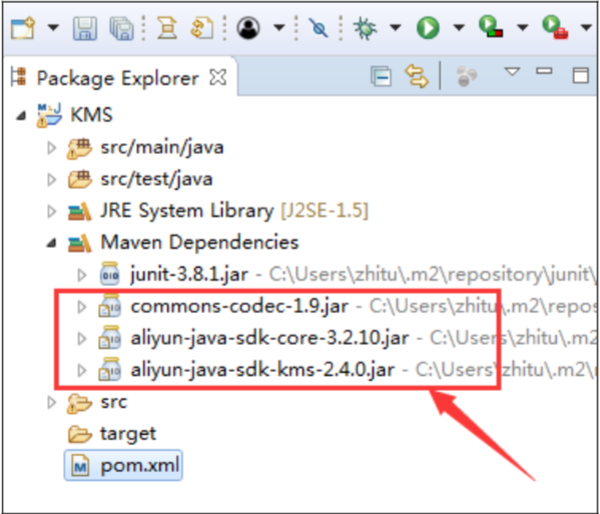
至此完成KMS项目的创建，如下图所示



配置Maven依赖打开pom.xml文件，进入编辑页面，添加commons-codec 、aliyun-java-sdk-core以及aliyun-java-sdk-kms包，如下所示

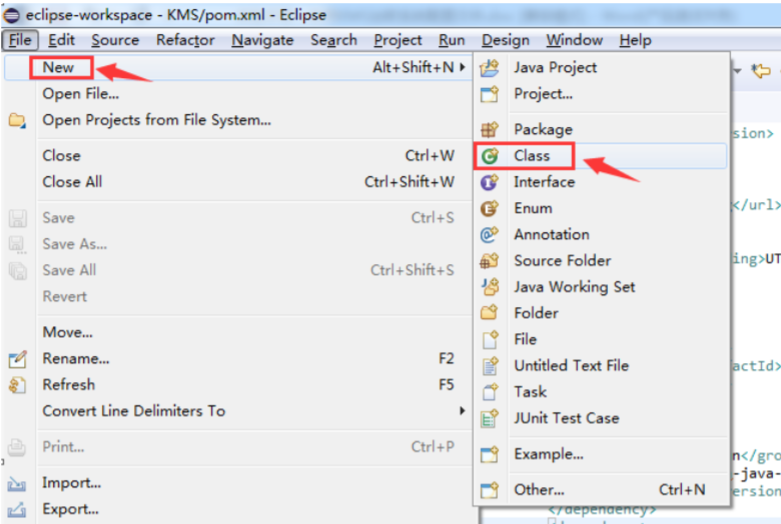


查看下载的jar包打开Maven Dependencies中查看下载完成的jar包，如下所示

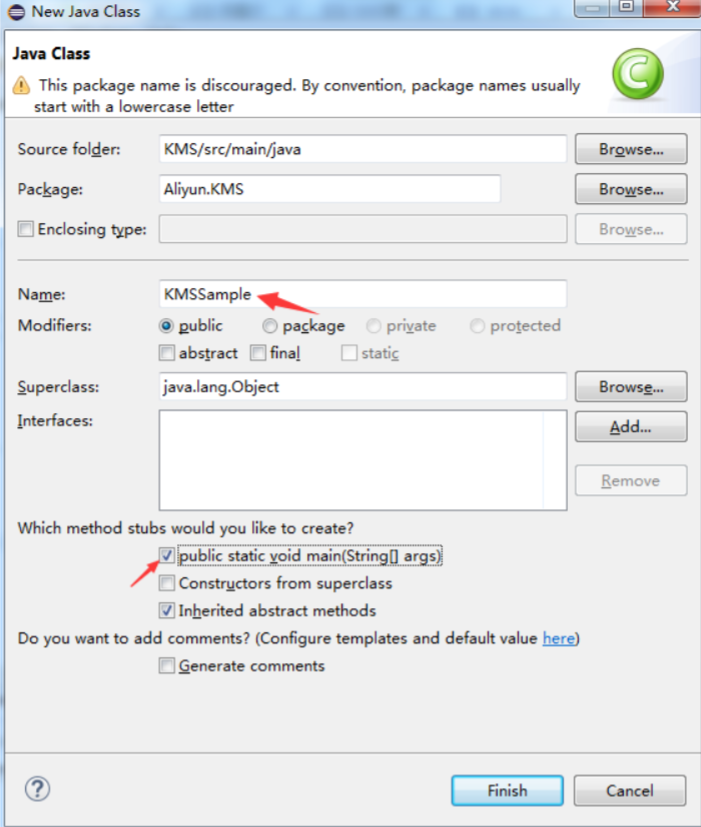


使用KMS SDK进行加解密

1)创建KMSSample类依次选择file>New>Class进行创建，如下图所示

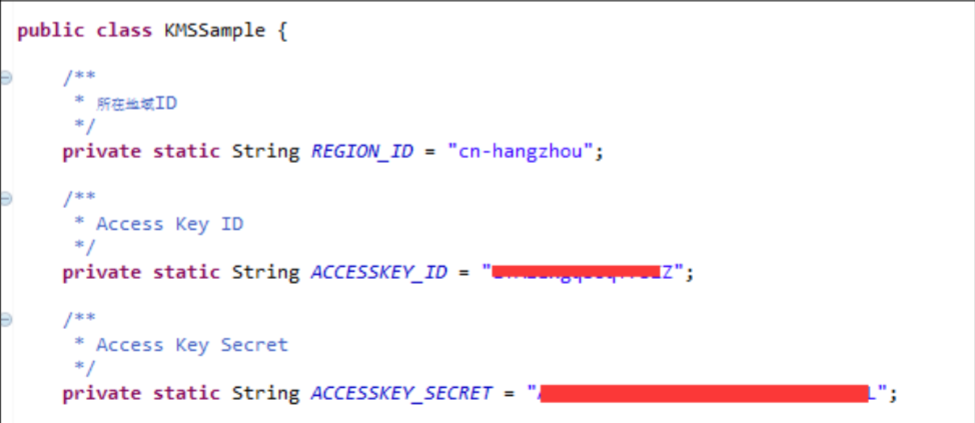


输入类名KMSSample，勾选上“public static void main...”，然后点击“Finish”，完成类创建，如下图所示



### 定义全局常量

定义阿里云API调用的常量，所在地域，Access Key ID，Access Key Secret，如下图所示

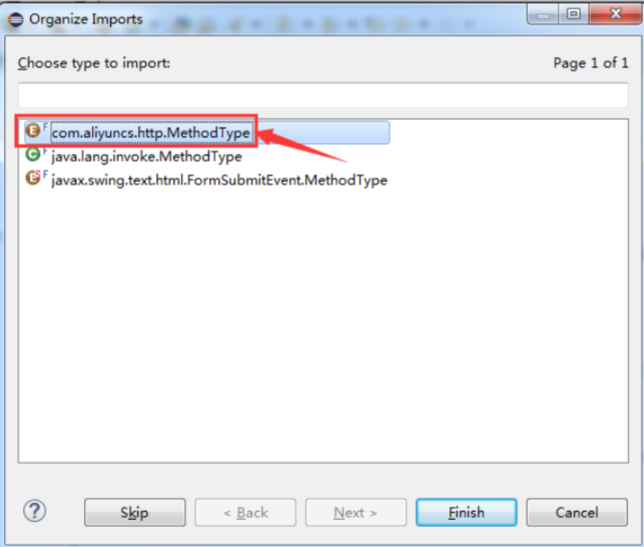


### 添加加密方法kmsEncrypt

添加使用KMS加密方法kmsEncrypt，如下所示



可使用按CTRL + SHIFT + O快捷键,添加import里面的包，在弹出页面选择“com.aliyuncs.http.MethodType”,如下图所示



### 添加解密方法kmsDecrypt

添加使用KMS解密方法kmsDecrypt，如下所示



### 测试加解密方法

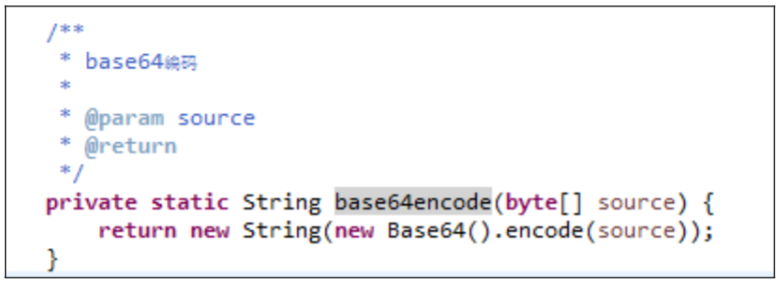
在main函数中对kmsEncrypt和kmsDecrypt进行测试，将“Hello,KMS”进行加密输出，然后再进行解密输出，其中keyId为步骤2中创建的密钥ID，如下图所示



直接加解密配置文件信息

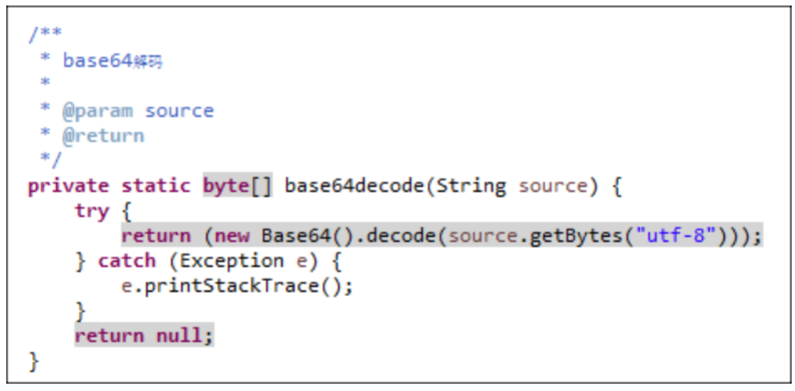
1. 添加base64编码方法base64encode

添加使用base64编码方法base64encode，如下所示



### 添加base64解码方法base64decode

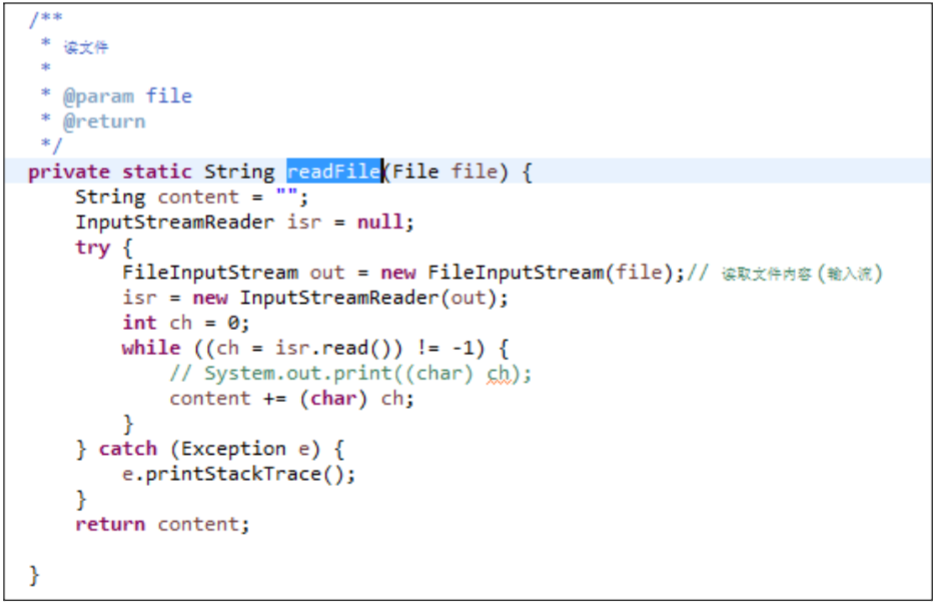
添加使用base64解码方法base64decode，如下所示



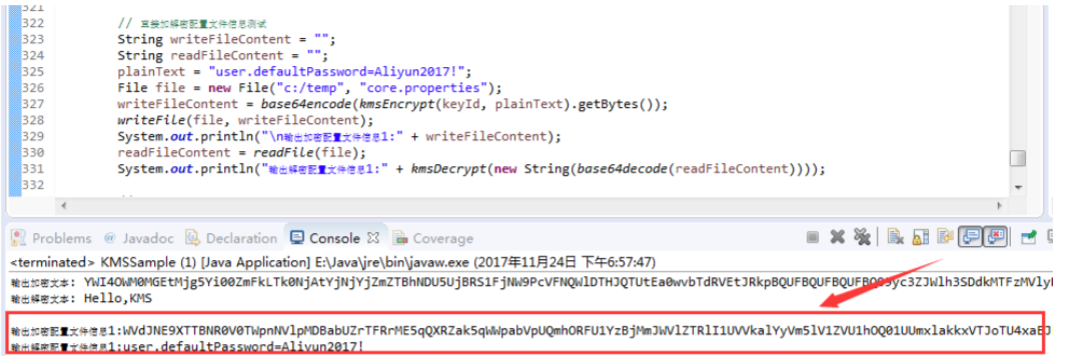
### 添加读写文件的方法writeFile和readFile

添加写文件的方法writeFile和读文件的方法readFile，如下所示

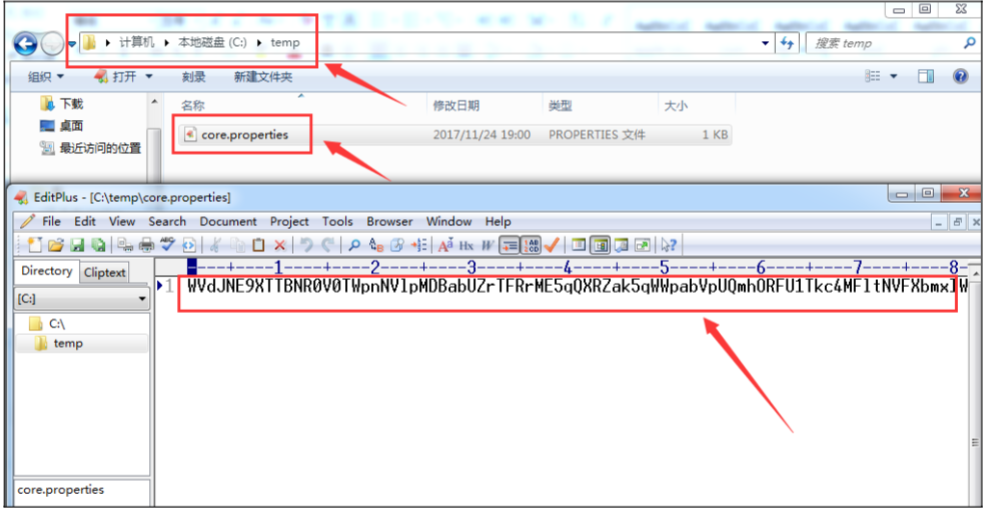




加解密配置文件内容在main函数中，使用base64编解码方法以及读写文件的方法，对配置文件的内容进行加解密操作，将加密的文本存储在配置文件中，再从配置文件中读取加密的文本进行解密，如下所示：



打开配置文件core.properties，可以查看到加密内容，如下图所示



通过数字信封加解密配置文件信息

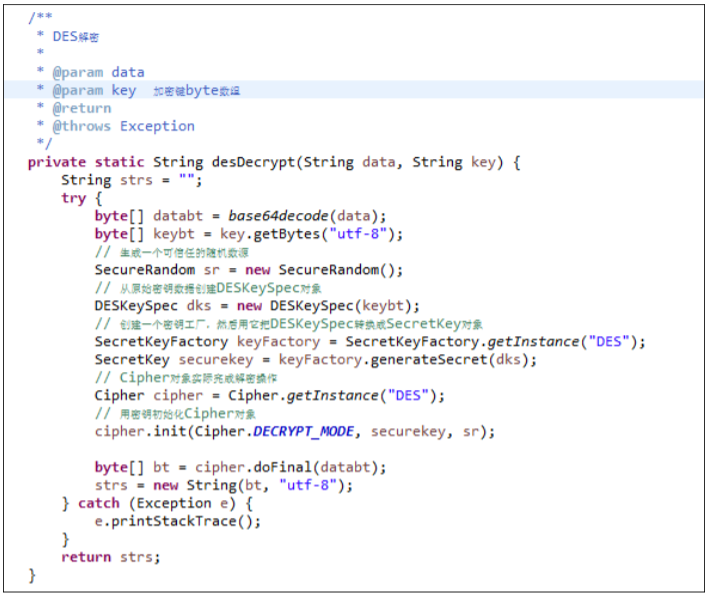
1. 添加DES加密方法

添加使用DES加密方法desEncrypt，如下所示



### 添加DES解密方法desDecrypt

添加使用DES解密方法desDecrypt，如下所示：



### 获取数据密钥方法kmsGenerateDataKey

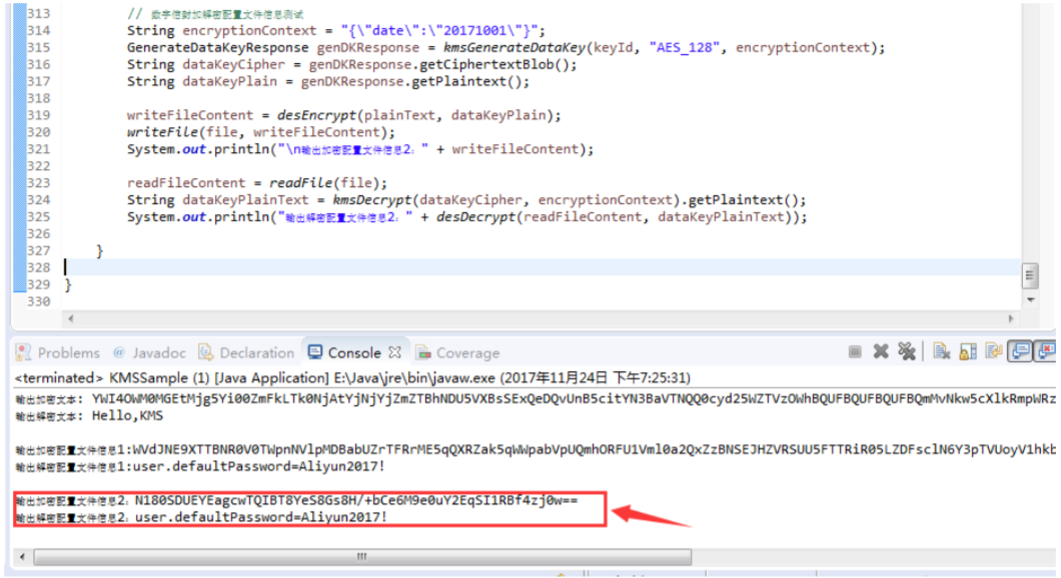
添加KMS获取数据加密的明文密钥方法kmsGenerateDataKey，如下所示



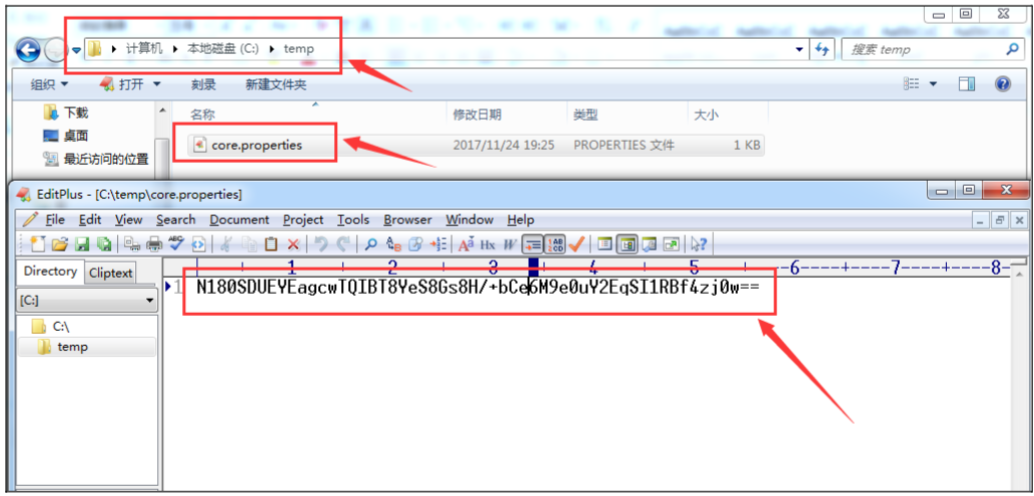
### 使用数据信封的方式加解密配置文件信息

在main函数中，使用DES加解密方法以及

获取数据加密的明文密钥方法，通过数据信封的方式加解密配置文件的内容，如下所示



同样打开配置文件core.properties，可以查看到加密内容，如下图所示



## 附录

### pom.xml:

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>Aliyun</groupId>  <artifactId>KMS</artifactId>  <version>1.0-SNAPSHOT</version>  <packaging>jar</packaging>  <name>KMS</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.9</version>  </dependency>  <dependency>  <groupId>com.aliyun</groupId>  <artifactId>aliyun-java-sdk-core</artifactId>  <version>3.2.10</version></dependency>  <dependency>  <groupId>com.aliyun</groupId>  <artifactId>aliyun-java-sdk-kms</artifactId>  <version>2.4.0</version>  </dependency>  </dependencies> </project> |

### KMSSample：

|  |
| --- |
| package Aliyun.KMS;  import com.aliyuncs.DefaultAcsClient; import com.aliyuncs.exceptions.ClientException; import com.aliyuncs.exceptions.ServerException; import com.aliyuncs.http.FormatType; import com.aliyuncs.http.MethodType; import com.aliyuncs.http.ProtocolType; import com.aliyuncs.kms.model.v20160120.\*; import com.aliyuncs.profile.DefaultProfile; import com.aliyuncs.profile.IClientProfile; import org.apache.commons.codec.binary.Base64; import javax.crypto.Cipher; import javax.crypto.SecretKey; import javax.crypto.SecretKeyFactory; import javax.crypto.spec.DESKeySpec; import java.io.\*; import java.security.SecureRandom;  */\*\*  \* 测试 KMS  \* 阿里云的密钥管理服务  \*/* public class KMSSample {    */\*\*  \* 所在地域ID  \*/* // private static String REGION\_ID = "填入您的密钥所在地域ID";  private static String *REGION\_ID* = "cn-shanghai";    // AccessKey ID和AccessKey Secret是您访问阿里云API的密钥，具有该账户完全的权限，请您妥善保管。  */\*\*  \* Access Key ID  \*/* private static String *ACCESSKEY\_ID* = "\*\*\*\*\*\*\*\*"; // private static String ACCESSKEY\_ID = "填入您的Access Key ID";    */\*\*  \* Access Key Secret  \*/* private static String *ACCESSKEY\_SECRET* = "\*\*\*\*\*\*\*\*"; // private static String ACCESSKEY\_SECRET = "填入您的Access Key Secret";    */\*\*  \* 加密方法  \** ***@return*** *加密文本  \** ***@paramkeyId*** *密钥ID  \** ***@paramplainText*** *明文  \*/* private static String kmsEncrypt(String keyId, String plainText) {  IClientProfile profile = DefaultProfile.*getProfile*(*REGION\_ID*, *ACCESSKEY\_ID*, *ACCESSKEY\_SECRET*);  DefaultAcsClient client = new DefaultAcsClient(profile);  String cipherBlob = "";  try {  final EncryptRequest encReq = new EncryptRequest();  encReq.setProtocol(ProtocolType.*HTTPS*);  encReq.setAcceptFormat(FormatType.*JSON*);  encReq.setMethod(MethodType.*POST*);  encReq.setKeyId(keyId);  encReq.setPlaintext(plainText);  final EncryptResponse encResponse = client.getAcsResponse(encReq);  cipherBlob = encResponse.getCiphertextBlob();  } catch (ClientException eResponse) {  eResponse.printStackTrace();  }  return cipherBlob;  }    */\*\*  \* 获取数据密钥  \** ***@param*** *keyId  \** ***@param*** *keyDesc  \** ***@param*** *encryptionContext  \** ***@return*** *\*/* private static GenerateDataKeyResponse kmsGenerateDataKey(String keyId, String keyDesc, String encryptionContext) {  IClientProfile profile = DefaultProfile.*getProfile*(*REGION\_ID*, *ACCESSKEY\_ID*, *ACCESSKEY\_SECRET*);  DefaultAcsClient kmsClient = new DefaultAcsClient(profile);  final GenerateDataKeyRequest genDKReq = new GenerateDataKeyRequest();  genDKReq.setProtocol(ProtocolType.*HTTPS*);  genDKReq.setAcceptFormat(FormatType.*JSON*);  genDKReq.setMethod(MethodType.*POST*);  genDKReq.setKeySpec(keyDesc);  genDKReq.setKeyId(keyId);  // genDKReq.setNumberOfBytes(numOfBytes);  if (!"".equals(encryptionContext)) {  genDKReq.setEncryptionContext(encryptionContext);  }  GenerateDataKeyResponse genDKRes = null;  try {  genDKRes = kmsClient.getAcsResponse(genDKReq);  } catch (ServerException e) {  e.printStackTrace();  } catch (ClientException e) {  e.printStackTrace();  }  return genDKRes;  }    */\*\*  \* 解密方法  \** ***@param*** *cipherBlob  \** ***@param*** *encryptionContext  \** ***@return*** *\*/* private static DecryptResponse kmsDecrypt(String cipherBlob, String encryptionContext) {  IClientProfile profile = DefaultProfile.*getProfile*(*REGION\_ID*, *ACCESSKEY\_ID*, *ACCESSKEY\_SECRET*);  DefaultAcsClient kmsClient = new DefaultAcsClient(profile);  final DecryptRequest decReq = new DecryptRequest();  decReq.setProtocol(ProtocolType.*HTTPS*);  decReq.setAcceptFormat(FormatType.*JSON*);  decReq.setMethod(MethodType.*POST*);  decReq.setCiphertextBlob(cipherBlob);  if (!"".equals(encryptionContext)) {  decReq.setEncryptionContext(encryptionContext);  }  DecryptResponse decResponse = null;  try {  decResponse = kmsClient.getAcsResponse(decReq);  } catch (ServerException e) {  e.printStackTrace();  } catch (ClientException e) {  e.printStackTrace();  }  return decResponse;  }    */\*\*  \* 調用自己的解密方法  \** ***@param*** *cipherBlob  \** ***@return*** *\*/* private static String kmsDecrypt(String cipherBlob) {  String plainText = *kmsDecrypt*(cipherBlob, "").getPlaintext();  return plainText;  }    */\*\*  \* base64编码  \** ***@param*** *source  \** ***@return*** *\*/* private static String base64encode(byte[] source) {  return new String(new Base64().encode(source));  }    */\*\*  \* base64解码  \** ***@param*** *source  \** ***@return*** *\*/* private static byte[] base64decode(String source) {  try {  return (new Base64().decode(source.getBytes("utf-8")));  } catch (Exception e) {  e.printStackTrace();  }  return null;  }    */\*\*  \* DES加密  \** ***@param*** *data  \** ***@param*** *key 加密键byte数组  \** ***@return*** *\** ***@throws*** *Exception  \*/* private static String desEncrypt(String data, String key) {  String strs = "";  try {  byte[] databt = data.getBytes("utf-8");  byte[] keybt = key.getBytes("utf-8");  // 生成一个可信任的随机数源  SecureRandom sr = new SecureRandom();  // 从原始密钥数据创建DESKeySpec对象  DESKeySpec dks = new DESKeySpec(keybt);  // 创建一个密钥工厂，然后用它把DESKeySpec转换成SecretKey对象  SecretKeyFactory keyFactory = SecretKeyFactory.*getInstance*("DES");  SecretKey securekey = keyFactory.generateSecret(dks);  // Cipher对象实际完成加密操作  Cipher cipher = Cipher.*getInstance*("DES");  // 用密钥初始化Cipher对象  cipher.init(Cipher.*ENCRYPT\_MODE*, securekey, sr);  strs = *base64encode*(cipher.doFinal(databt));  } catch (Exception e) {  e.printStackTrace();  }  return strs;  }    */\*\*  \* DES解密  \** ***@param*** *data  \** ***@param*** *key 加密键byte数组  \** ***@return*** *\** ***@throws*** *Exception  \*/* private static String desDecrypt(String data, String key) {  String strs = "";  try {  byte[] databt = *base64decode*(data);  byte[] keybt = key.getBytes("utf-8");  // 生成一个可信任的随机数源  SecureRandom sr = new SecureRandom();  // 从原始密钥数据创建DESKeySpec对象  DESKeySpec dks = new DESKeySpec(keybt);  // 创建一个密钥工厂，然后用它把DESKeySpec转换成SecretKey对象  SecretKeyFactory keyFactory = SecretKeyFactory.*getInstance*("DES");  SecretKey securekey = keyFactory.generateSecret(dks);  // Cipher对象实际完成解密操作  Cipher cipher = Cipher.*getInstance*("DES");  // 用密钥初始化Cipher对象  cipher.init(Cipher.*DECRYPT\_MODE*, securekey, sr);  byte[] bt = cipher.doFinal(databt);  strs = new String(bt, "utf-8");  } catch (Exception e) {  e.printStackTrace();  }  return strs;  }    */\*\*  \* 写文件  \** ***@param*** *file  \** ***@param*** *content  \*/* private static void writeFile(File file, String content) {  try {  if (!file.exists()) {  file.createNewFile(); // 创建文件  }  } catch (IOException e) {  e.printStackTrace();  }  byte bt[] = new byte[1024];  bt = content.getBytes();  try {  FileOutputStream in = new FileOutputStream(file);// 向文件写入内容(输出流)  try {  in.write(bt, 0, bt.length);  in.close();  // System.out.println("写入文件成功");  } catch (IOException e) {  e.printStackTrace();  }  } catch (FileNotFoundException e) {  e.printStackTrace();  }  }    */\*\*  \* 读文件  \** ***@param*** *file  \** ***@return*** *\*/* private static String readFile(File file) {  String content = "";  InputStreamReader isr = null;  try {  FileInputStream out = new FileInputStream(file);// 读取文件内容(输入源)  isr = new InputStreamReader(out);  int ch = 0;  while ((ch = isr.read()) != -1) {  // System.out.print((char) ch);  content += (char) ch;  }  } catch (Exception e) {  e.printStackTrace();  }  return content;  }    public static void main(String[] args) {  // 加解密方法测试  String keyId = "\*\*\*\*\*\*\*\*"; // 必须和 所在地域ID在同一个区域 // String keyId = "填入您的密钥ID";  String plainText = "Hello,KMS";  String encryptText = *kmsEncrypt*(keyId, plainText);  System.*out*.println("输出加密文本: " + encryptText);  String decryptText = *kmsDecrypt*(encryptText);  System.*out*.println("输出解密文本: " + decryptText);   // 直接加解密配置文件信息测试  String writeFileContent = ""; // 初始变量  String readFileContent = ""; // 初始变量  plainText = "user.defaultPassword=Aliyun2017!";  File file = new File("c:/temp", "core.properties");  writeFileContent = *base64encode*(*kmsEncrypt*(keyId, plainText).getBytes());  *writeFile*(file, writeFileContent);  System.*out*.println("\n输出加密配置文件信息1：" + writeFileContent);  readFileContent = *readFile*(file);  System.*out*.println("输出解密配置文件信息1：" + *kmsDecrypt*(new String(*base64decode*(readFileContent))));   // 数字信封加解密配置文件信息测试  String encryptionContext = "{\"date\":\"20181212\"}"; // String encryptionContext = "{\"date\":\"20171001\"}";  GenerateDataKeyResponse genDKResponse = *kmsGenerateDataKey*(keyId, "AES\_128", encryptionContext);  String dataKeyCipher = genDKResponse.getCiphertextBlob();  String dataKeyPlain = genDKResponse.getPlaintext();  writeFileContent = *desEncrypt*(plainText, dataKeyPlain);  *writeFile*(file, writeFileContent);  System.*out*.println("\n输出加密配置文件信息2：" + writeFileContent);  readFileContent = *readFile*(file);  String dataKeyPlainText = *kmsDecrypt*(dataKeyCipher, encryptionContext).getPlaintext();  System.*out*.println("输出解密配置文件信息2：" + *desDecrypt*(readFileContent, dataKeyPlainText));  } } |