```
CK_BBOOL            sw_false = 0;
CK_SESSION_HANDLE hSession = ...; // 见 C_OpenSession 调用
char                private_key_label[]="private key label";
CK_ULONG            ulObjectCount;
CK_ATTRIBUTE        attrtemplate[1][5] = {   {
        {CKA_CLASS,      &private_key_class, sizeof(private_key_class)},
        {CKA_KEY_TYPE, &keyType_rsa, sizeof(keyType_rsa)},
        {CKA_TOKEN,      &sw_false, sizeof(sw_false)},
        {CKA_PRIVATE,   &sw_false, sizeof(sw_false)},
        {CKA_LABEL,       private_key_label, sizeof(private_key_label)-1}
}   };
CK_MECHANISM        rsa_Mechasm = {CKM_SHA1_RSA_PKCS, NULL_PTR, 0};
CK_BYTE             pSignature[1024];          /* gets the signature */
CK_ULONG            ulSignatureLen = sizeof(pSignature);   /* gets signature length */
char     *data=...;
int       dalen = ....;
rv = C_FindObjectsInit(hSession, attrtemplate[0], 5);
if (rv != CKR_OK) {       return -2;       }
ulObjectCount = 0;
do {
     rv = C_FindObjects(hSession, &key, 1, &ulObjectCount);
     if (rv != CKR_OK) {
          C_FindObjectsFinal(hSession);
          return -3;
     }
} while (ulObjectCount == 1);
rv = C_FindObjectsFinal(hSession);
if(rv != CKR_OK) {       return -4;       }
// 进行签名操作
rv = C_SignInit(hSession, &rsa_Mechasm, key);
if (rv != CKR_OK) {       return -5;       }
rv = C_Sign(hSession, data, dalen, pSignature, &ulSignatureLen);
if (rv != CKR_OK) {       return   -6;      }
// C_Sign 完成后，pSignature 存储了签名结果，ulSignatureLen 保存了签名值长度
```

12.3　JCA/JCE

12.3.1　JCA/JCE 简介

JCA（Java Cryptography Architecture，Java 密码体系结构）和 JCE（Java Cryptography Extension，Java 密码扩展包）是 Java 平台提供的用于安全加密服务的两组 API，但它们并不实现任何算法，它们只是连接应用程序和实际算法实现程序的一组接口。软件开发商可以根据 JCE 接口把各种算法实现后，打包成一个安全实现提供者，动态地加载到 Java 运行环境中。

GUOXU.nokiaMS

根据美国出口限制的规定，JCA 可出口（JCA 和 Sun 的一些默认实现包含在 Java 发行版中），但 JCE 对部分国家是限制出口的。因此，要实现一个完整的安全功能，就需要一个或多个第三方厂商提供 JCE 实现产品，其称为安全提供者。

安全提供者是承担特定安全机制实现的第三方，有些提供者是完全免费的，而另一些提供者则需要付费。安全提供者的公司有 Sun、BouncyCastle 等。Sun 提供了如何开发安全提供者的细节。BouncyCastle 提供了可以在 J2ME/ J2EE/J2SE 平台应用的 API，而且 BouncyCastle 是免费的。

在 JDK 1.4 之前，JCE 并不随 JDK 一起发布，正因如此，JCA 和 JCE 经常被称为独立的不同组件。随着 JCE 捆绑在 JDK 1.4 以后版本中一起发布，JCA 和 JCE 的区别变得越来越不明显，由于 JCE 使用与 JCA 相同的架构，JCE 越来越被认为是 JCA 的一个组成部分。

在 JDK 中的 JCA 包括两个软件组件：密码服务框架的定义和密码服务提供者。密码服务框架包括 java.security、javax.crypto、javax.crypto.spec 和 javax.crypto.interfaces 等软件包。密码服务提供者包括 Sun、SunRsaSign、SunJCE 等。每当提及一个特定的 JCA 提供者时，总是引用提供者的名称。

1. JCA 体系架构

JCA 遵循如下设计原则：实现上的独立性和互操作性，算法上的独立性和可扩展性。基于这两个原则，采用了如下实现技术。

（1）实现上的独立性

应用程序不需要自己实现安全算法，它们只需要在 Java 平台中请求安全服务即可。其中，安全服务的实现是在提供者中完成的，而提供者通过标准的接口以插件形式存在于 Java 平台中。因此，实现上的独立性是通过这种基于提供者的架构而取得的。提供者又称为 CSP（Cryptographic Service Provider，密码服务提供者）。

（2）实现上的互操作性

提供者在各种应用程序之间可以彼此协作，也就是应用程序可以使用不同的提供者，同时，一个提供者也可以为多个程序服务。

（3）算法上的独立性

Java 平台定义了一些不同类型的与密码相关的引擎（Engines）或者叫服务（Services），然后定义了为该引擎提供功能的类，叫作引擎类。比如，MessageDigest、Signature、KeyFactory、Cipher 等。

（4）算法上的可扩展性

第三方的密码算法可以通过扩展 JCA 给出的某些类来方便地添加。

JCA 与应用程序的关系如图 12-4 所示，ProviderB 和 ProviderC 都实现了 MD5 算法。

java.security.Provider 是所有这些提供者的基础类。每个提供者包含该密码服务提供者的名字以及它所实现的所有算法的列表。当应用程序需要使用一个特定的算法时，它可以向 JCA 请求，JCA 框架就会访问所有可用的提供者，然后从中选中一个合适的提供者，并为该应用程序创建一个算法的实例。比如：

```
md = MessageDigest.getInstance("MD5");
md = MessageDigest.getInstance("MD5", "ProviderC");
```

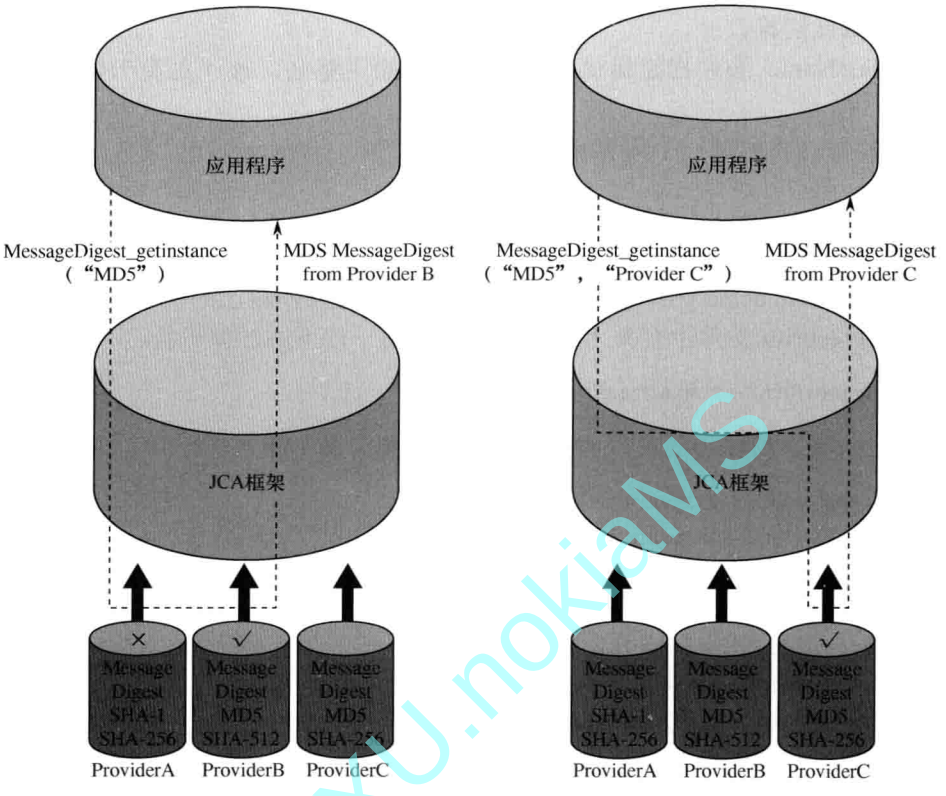

图 12-4　JCA 体系架构

对于上面两句代码，在图 12-4 中有 3 个加密服务提供者，分别是 ProviderA、ProviderB 和 ProviderC，应用程序向 JCA 请求“MD5”算法服务，JCA 通过搜索所有可用的加密提供者来返回合适的算法实现。对于第一句代码，见图 12-4 中左半部分，JCA 查找到 ProviderA，而 ProviderA 提供了“SHA-1”和“SHA-256”算法，并没有提供“MD5”算法，所以不合适，继续搜索，找到 ProviderB，因为它里面有“MD5”算法，故返回该算法的一个实例。

而对于第二句代码，见图 12-4 中右半部分，它指定了从 ProviderC 中获取“MD5”算法服务，故 JCA 直接返回 ProviderC 中“MD5”算法的一个实例。

2．安装密码服务提供者

首先要使类可用，以便需要时能找到它们。可以按 JAR 文件的形式提供提供者类。把提供者添加到被认可的提供者列表中。这可通过编辑 JDK 的 lib/security（在 Windows 中为 lib\security）目录下的 java.security 文件来完成。对于每个提供者，该文件都应有如下形式的声明：

```
security.provider.n=masterClassName
```

该语句声明了一个提供者并指定了它的优先顺序 n。优先顺序是当没有请求特定提供者时，为获得请求算法而查找提供者的顺序。优先顺序是从 1 开始的，1 享有最大优先权，

CUOXU.nokiaMS

紧接着是 2，依此类推。

masterClassName 必须指定提供者的“主类”的完整名，该“主类”始终是 Provider 类的子类。

安装 JDK 时，将包括一个内置提供者，即“SUN”。java.security 文件仅提供如下提供者规范：

```
security.provider.1=sun.security.provider.Sun
```

假定主类是 com.acme.provider 包中的 Acme 类，并且使自己的提供者优先顺序为 2。为此，在 java.security 文件中有关“SUN”提供者的一行下添加如下行：

```
security.provider.2 = com.acme.provider.Acme
```

在 JDK 1.6 中，所提供的密码服务提供者的配置信息如下：

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.8=sun.security.smartcardio.SunPCSC
security.provider.9=sun.security.mscapi.SunMSCAPI
```

3. 引擎（Engine）类和算法

一个“引擎类”以一种抽象方式（无具体的实现方法）定义一种密码服务。密码服务总是与某个特定的算法或类型相关联，它或者提供密码运算（如数字签名或报文摘要的运算），生成或提供密码运算所需要的密码信息（密钥或参数），或者生成安全封装的密钥（这些密钥可用于密码运算中）的数据对象。例如，Signature 和 KeyFactory 类就是两个引擎类。

引擎类提供了两种类型操作：

① 密码操作（加密、数字签名、消息摘要等）；

② 生成器或密码参数转换器（密钥和算法参数），或密码对象（如密钥库或证书），密码对象是对加密数据的封装，使密码数据可以在更高抽象层次使用。

JDK 1.6 中定义了常用的引擎类，如表 12-3 所示。

表 12-3 JDK 常用引擎类

引 擎 类	说 明
SecureRandom	用于生成随机数或伪随机数
MessageDigest	用于计算数据的报文摘要
Signature	用于对数据进行签名和校验数字签名
Cipher	用密钥进行初始化，用于加密/解密数据。支持多种类型算法：对称块加密（例如 AES、DES、DESede、Blowfish、IDEA），流加密（如 RC4），非对称加密（例如 RSA），以及基于密码的加密法（PBE）

GUOXU.nokiaMS

（续表）

引　擎　类	说　　明
MAC	消息认证码，如 MessageDigests 一样，也产生哈希值。它先用密钥进行初始化，以保护信息的完整性
KeyFactory	用于把类型为 Key 的密钥数据转换为规范密钥，或进行相反的转换
SecretKeyFactory	用于把类型为 SecretKey 的密钥数据转换成规范密钥，或进行相反的转换。SecretKey Factorys 是 KeyFactorys 的子类，用于创建对称密钥
KeyPairGenerator	用于生成与指定算法相匹配的公钥和私钥对
KeyGenerator	用于生成指定算法的新的对称密钥
KeyAgreement	由两方或多方商定，指定一个密钥进行加密操作
AlgorithmParameters	用于存储某一特定算法的参数，包括编码和解码参数
AlgorithmParameterGenerator	用于生成一组与指定算法相匹配的参数
KeyStore	用于创建和管理密钥库。密钥库是密钥的数据库。密钥库中的私钥有一个与之关联的证书链，用于认证对应的公钥。密钥库还含有来自可信实体的证书
CertificateFactory	用于创建公钥证书和证书撤销列表（CRL）
CertPathBuilder	用于构建证书链（也称为证书路径）
CertPathValidator	用于验证证书链的有效性
CertStore	用于检索证书和证书撤销列表（CRL）

注意，生成器创建全新内容的密码对象，而工厂是从已有数据（如一个已经编码好的证书）创建密码对象。所有的引擎类没有公共构造方法，必须使用 getInstance 方法以获得一个密码对象实例。

12.3.2　使用证书

证书和私钥都存放在密钥库（KeyStore）中，为了演示方便，先使用 keytool 工具产生一个测试密钥库，然后使用代码访问产生的测试密钥库。

12.3.2.1　使用 keytool 工具产生测试密钥库

1．keytool 工具说明

keytool 是一个密钥和证书管理实用程序。它使用户能够管理自己的公私钥对及相关证书，或数据完整性和认证服务，以及使用数字签名。它还允许用户缓存与其通信的另一方的公钥（以证书形式）。该工具还允许用户管理在对称加密/解密（如 DES）中使用的秘密密钥。keytool 常用命令参数如表 12-4 所示。

表 12-4　JDK keytool 常用命令参数

参　　数	说　　明
-genkey	在用户主目录中创建一个默认文件“.keystore”，并会产生一个 mykey 的别名，mykey 中包含用户的公钥、私钥和证书（在没有指定生成位置的情况下，keystore 会存在于用户系统默认目录中）
-alias	产生别名。缺省值为“mykey”
-keystore	指定密钥库的名称（产生的各类信息将不在.keystore 文件中）
-keyalg	指定密钥的算法（如 RSA、DSA，如果不指定则默认采用 DSA）

GUOXUEKAIS