第 4 章

他山之石，可以攻玉

在第3章中，我们了解了Java 6中与加密/解密相关的API，几乎各种常用加密算法都能找到对应的实现，但还是难免会有遗憾：受出口限制，密钥长度上不能满足需求；部分算法未能支持，如MD4、SHA-224等算法；API使用起来还不是很方便；一些常用的进制转换辅助工具未能提供，如Base64编码转换、十六进制编码转换等工具。

对于上述这些问题，该如何解决呢？他山之石，可以攻玉！

对于出口限制，Sun通过权限文件（local_policy.jar、US_export_policy.jar）做了相应限制。但幸运的是，Sun在官方主页上提供了替换文件，可减少相关的限制。当然，你需要结合本地进出口政策，合法使用该文件。

对于Java 6未能支持的加密算法，各大密码学机构以及以加密算法为核心的软件组织和软件公司都不遗余力地研究，并提供了相应的实现。但因为版权问题，我们很少能够使用到这些机构提供的加密软件包。在这样一个开源的时代，难道就没有其他的替代方案吗？当然不是，Bouncy Castle（http://www.bouncycastle.org/）提供了一系列算法支持实现，并可以跻身于JCE框架之下，以提供者的方式纳入其中。而且，Bouncy Castle是一款开源组件，你不必为版权问题而伤脑筋。此外，Bouncy Castle提供了Base64和十六进制编码转换相关的实现。

Commons Codec（http://commons.apache.org/codec/）是国际开源组织Apache（http://www.apache.org/）旗下的一款开源软件。它与Bouncy Castle不同，并未对Java 6提供扩展加密算法。仅仅是对Java 6提供的API做了改进，提供了更加易用的API。

4.1 加固你的系统

鉴于出口限制问题，我们得到的JDK安全强度不够高，主要是密钥长度不够。对于这一点，Sun在其官方网站上提供了无政策限制权限文件（Unlimited Strength Jurisdiction Policy Files），我们只需要将其部署在JRE环境中。

如果你所在的国家或地区不受该进出口限制，就完全可以使用该文件解除权限限制问题。

4.1.1 获得权限文件

Sun在其下载页面（http://java.sun.com/javase/downloads/index.jsp）上提供了该权限文件的下载地址，如图4-1所示，注意Other Downloads项。

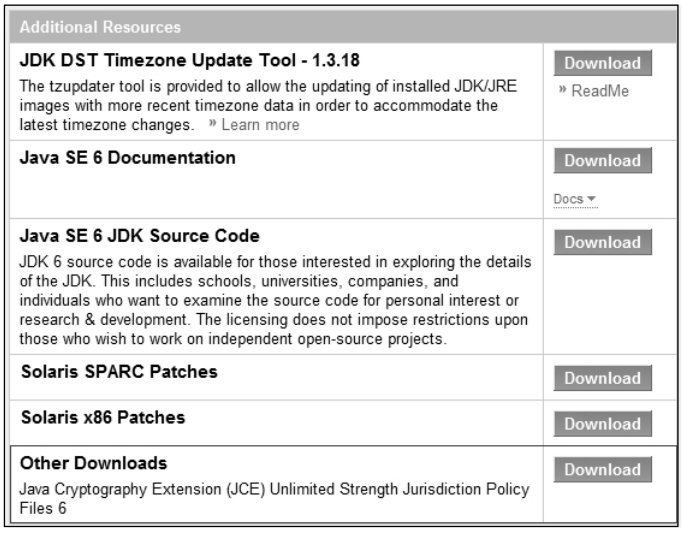


图4-1 权限文件下载页面

Other Downloads项提供了Java Cryptography Extension（JCE）Unlimited Strength Jurisdiction Policy Files 6的下载地址。很显然，它的版本是6，对应Java 6。下载后，会得到一个名为jce_policy-6.zip的文件。用WinRAR打开后，如图4-2所示。

图4-2 jce_policy-6.zip文件中jce目录列表

在这个压缩包中仅有一个目录，也就是jce目录。该目录中包含了4个文件：README.txt、COPYRIGHT.html、local_policy.jar和US_export_policy.jar。其中包含的两个jar文件正是此次配置中用到的文件。

4.1.2 配置权限文件

我们可以查看上述README.txt文件，你需要在JDK的JRE环境中，或者是JRE环境中配置

上述两个jar文件。

切换到%JDK_Home%\jre\lib\security目录下，对应覆盖local_policy.jar和US_export_policy.jar两个文件。同时，你可能有必要在%JRE_Home%\lib\security目录下，也需要对应覆盖这两个文件。

配置权限文件的最终目的是为了使应用在运行环境中获得相应的权限，可以加强应用的安全性。

通常，我们在应用服务器上安装的是JRE，而不是JDK。因此，这就很有必要在应用服务器的%JRE_Home%\lib\security目录下，对应覆盖这两个权限文件。很多开发人员往往忽略了这一点，导致事故发生。

4.1.3 验证配置

经过一番调整之后，如何验证我们的系统获得相应的权限呢？

修改权限配置的目的是为了获得更长的密钥。如果对于同一个加密算法，在修改权限配置前后系统所能提供的密钥的最长长度发生了变化，那么就说明配置有效！

AES算法是较为常用的对称加密算法之一，几乎是对称加密算法中安全级别最高的算法。Java 6支持AES算法的密钥长度为128位、192位或256位。但是，如果不加权限配置直接使用256位长度的密钥，就会得到“java.security.InvalidKeyException”异常，如下面代码所示：

```
KeyGenerator kg = KeyGenerator.getInstance(KEY_ALGORITHM);
// AES 要求密钥长度为128位、192位或256位。
kg.init(256);
// 生成秘密密钥
SecretKey secretKey = kg.generateKey();
byte[] key = secretKey.getEncoded();
```

如果我们未能对相应的权限配置文件作相应调整，则会得到“java.security.InvalidKeyException”异常。反之，将正常获得256位长度的密钥！

有关AES算法相关实现细节，请读者阅读第7章。

4.2 加密组件Bouncy Castle

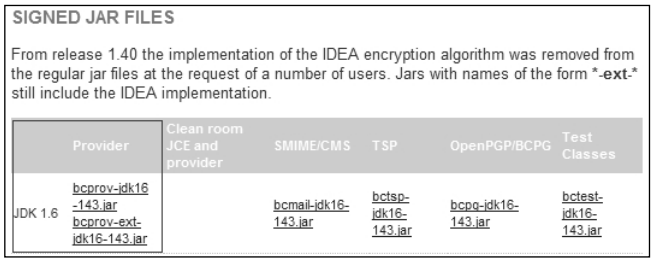
Java 6提供了多种算法支持，但并不完善。许多加密强度较高的算法，Java 6未能提供。在本书第3章中，我们提到了java.security文件，它位于%JDK_HOME%\jre\lib\security\ext目录下，用于提供者配置。

如果你需要使用Java 6不支持的算法，如MD4和IDEA（International Data Encryption Algorithm，国际数据加密算法）等。你可以在继续沿用Java 6的API前提下，通过在JRE环境中配置开源组件包Bouncy Castle，加入对应的提供者，获得相应的算法支持。关于Bouncy Castle支持的算法，请参见附录。

4.2.1 获得加密组件

Bouncy Castle目前提供的加密组件包的版本是1.43。自1.40版本开始，Bouncy Castle提供了对IDEA算法的支持。

我们可以通过Bouncy Castle提供的下载地址（http://www.bouncycastle.org/latest_releases.html），下载最新的加密组件包，主要是bcprov-jdk16-143.jar和bcprov-ext-jdk16-143.jar两个文件，如图4-3所示。

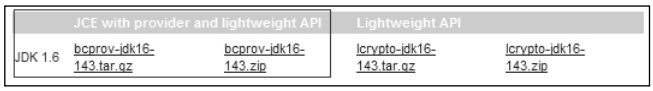

SIGNED JAR FILES

From release 1.40 the implementation of the IDEA encryption algorithm was removed from the regular jar files at the request of a number of users. Jars with names of the form *-ext-* still include the IDEA implementation.

	Provider	Clean room JCE and provider	SMIME/CMS	TSP	OpenPGP/BCPG	Test Classes
JDK 1.6	bcprov-jdk16-143.jar bcprov-ext-jdk16-143.jar		bcmail-jdk16-143.jar	bctsp-jdk16-143.jar	bcpg-jdk16-143.jar	bctest-jdk16-143.jar

图4-3 Bouncy Castle可执行二进制文件下载页面

当然，你有可能需要相应的源代码和API文档，它位于bcprov-jdk16-143.zip文件中，如图4-4所示。



	JCE with provider and lightweight API		Lightweight API	
JDK 1.6	bcprov-jdk16-143.tar.gz	bcprov-jdk16-143.zip	lcrypto-jdk16-143.tar.gz	lcrypto-jdk16-143.zip

图4-4 Bouncy Castle文档及源代码文件下载页面

获得bcprov-jdk16-143.jar和bcprov-ext-jdk16-143.jar两个文件后，我们就可以在Java 6的基础上扩充算法支持了。后面详细讲解这两个文件的使用方式。

Bouncy Castle加密组件包支持的算法不仅广泛，而且开源，常常用做J2ME的加密实现，其主页上也提供了J2ME版本的下载链接（http://www.bouncycastle.org/download/lcrypto-j2me-143.zip）。

4.2.2 扩充算法支持

对于Bouncy Castle提供的扩充算法支持，我们有两种方案可选：

1）配置方式。通过配置JRE环境，使其作为提供者（Provider）提供相应的算法支持，在代码实现层面只需指定要扩展的算法名称。

2）调用方式。在调用Java API初始化相应的密钥工厂、密钥生成器等引擎类之前，通过代码将Bouncy Castle提供者引入，获得扩展算法支持。

1. 配置方式

配置Bouncy Castle并不复杂，有点类似于4.1节中权限文件的配置，需要在%JDK_Home%和%JRE_Home%目录中做相应调整。

我们以%JDK_Home%目录配置为例。

❑ 使用步骤

首先，我们需要修改配置文件（java.security）。

在第3章中提到配置%JDK_Home%\ jre\lib\security\java.security文件，通过加入支持的方式获得更多的算法支持。

在这个文件中，我们可以很清晰地看到Java 6中有如下9种安全提供者：

```
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
```

Java 7很快就要问世了，作者查看了该文件的相应配置，并无变化。

上述这些配置是按照以下方式来配置的：

```
#security.provider.<n>=<className>
```

很显然，为了加入Bouncy Castle加密组件的安全提供者只需要这样做：

```
#增加BouncyCastleProvider
security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider
```

最后，我们需要将bcprov-ext-jdk16-143.jar文件导入。

切换至%JDK_Home%\jre\lib\ext目录下，我们能够看到 sunjce_provider.jar这个文件。SunJCE就是由这个文件提供的。同理，要将Bouncy Castle加密组件扩展包导入其中，只需要将4.2.1节中获得的bcprov-ext-jdk16-143.jar文件放到这里即可。

%JRE_Home%目录的相应配置与上述%JDK_Home%目录配置相类似。

对应修改%JRE_Home%\lib\security\java.security文件，并将bcprov-ext-jdk16-143.jar文件放置到%JRE_Home%\lib\ext目录中即可。

❑ 应用举例

Java 6不支持MD4算法，做了上述配置后，如果要使用MD4算法可参考如下代码：

```
/**
* MD4加密
* @param data
* @return
* @throws Exception
*/
public static byte[] encodeMD4(byte[] data) throws Exception {
    MessageDigest md = MessageDigest.getInstance("MD4");
    md.update(data);
    return md.digest();
}
```

这是一种对使用者透明的使用方式，你无须关心MD4算法的提供者是谁，代码很清晰。

2. 调用方式

有时候，我们需要通过明显的代码调用方式引入支持者，这完全依赖于Security类的addProvider()方法，详见3.2节。

❑ 使用步骤

首先，我们需要将bcprov-jdk16-143.jar文件导入工程。相信读者对于这一步操作一定都不陌生，这里就不详细介绍了。

接着，我们需要将以下两个类导入（import）你的代码中：

```
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

当然，如果你使用Eclipse，可以在下述代码写完后，使用快捷键Ctrl+Shift+O直接导入所需类。

最后，我们只需要在初始化密钥工厂、密钥生成器等引擎类之前，调用如下代码：

```
// 加入BouncyCastleProvider支持
Security.addProvider(new BouncyCastleProvider());
```

或者，在初始化密钥工厂、密钥生成器等引擎类时，采用如下方式：

```
MessageDigest md = MessageDigest.getInstance("MD4", "BC");
```

每个提供者都有简称，Bouncy Castle提供者简称“BC”，因此我们可以通过上述方式使用BouncyCastleProvider。

执行以下代码，我们可以获得Bouncy Castle提供者的算法详细描述。

```
Provider provider = Security.getProvider("BC");
System.err.println(provider);
for (Map.Entry<Object, Object> entry : provider.entrySet()) {
    System.err.println(entry.getKey() + " - " + entry.getValue());
}
```

在控制台中我们可以看到以下内容：

```
BC version 1.43
    Alg.Alias.Signature.SHA224withCVC-ECDSA - SHA224WITHCVC-ECDSA
    AlgorithmParameters.DES - org.bouncycastle.jce.provider.JDKAlgorithm
    Parameters$IVAlgorithmParameters
    KeyGenerator.2.16.840.1.101.3.4.22 - org.bouncycastle.jce.provider.
    symmetric.AES$KeyGen192
    Alg.Alias.Cipher.RSA//ISO9796-1PADDING - RSA/ISO9796-1
    AlgorithmParameterGenerator.NOEKEON - org.bouncycastle.jce.provider.
    symmetric.Noekeon$AlgParamGen
    Alg.Alias.Cipher.RSA//NOPADDING - RSA
```

```
    Alg.Alias.Mac.IDEA - IDEAMAC
    Alg.Alias.AlgorithmParameters.PBEWITHSHAAND3-KEYTRIPLEDES - PKCS12PBE
    Alg.Alias.Mac.HMAC/SHA1 - HMACSHA1
    Alg.Alias.AlgorithmParameterGenerator.1.2.410.200004.1.4 - SEED
    AlgorithmParameterGenerator.ELGAMAL - org.bouncycastle.jce.provider.
    JDKAlgorithmParameterGenerator$ElGamal
......
```

上述内容并未完全展现Bouncy Castle所支持的算法，本文做了简要摘录。

通常我们也可以通过上述方式检查系统支持的加密算法。

❑ 应用举例

Java 6未能支持MD4算法，也未能支持SHA-224算法。依照本文显式调用代码的方式，需要将bcprov-jdk16-143.jar文件导入工程，同时导入相关类（Security和BouncyCastleProvider），并通过Security类的addProvider()方法将BouncyCastleProvider类导入，见如下代码：

```
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
// ... 省略
/**
* SHA-224加密
*
* @param data
* @return
* @throws Exception
*/
public static byte[] encodeSHA224(byte[] data) throws Exception {
    // 加入BouncyCastleProvider支持
    Security.addProvider(new BouncyCastleProvider());
    MessageDigest md = MessageDigest.getInstance("SHA-224");
    md.update(data);
    return md.digest();
}
```

多了几行代码，多少有点别扭。但这种方式让人心知肚明，很清楚自己使用了哪些类。

3. 两种方式对比

配置和调用两种方法都有可取之处：前者对代码无需改动，只需提供Bouncy Castle支持的算法名称，但开发者需要知道Bouncy Castle已作为提供者部署在JRE中，也就是说这种方式需要依赖环境；后者需要对代码做改动，将Bouncy Castle作为提供者在代码中调用，但对环境依赖程度较小。

作者对于这两种方式并没有一个绝对的评价，它们各有特色。读者可根据需要，选择合适的方式获得扩展算法支持。

为了引起读者的注意，避免不必要的误导，本文在后续内容中将采用第2种方式介绍Bouncy Castle相关扩展算法。

4.2.3 相关API

Bouncy Castle的API主要包含了以下几个方面：

❑ JCE工具及其扩展包

仅包括org.bouncycastle.jce包。这是对JCE框架的支持。其中定义了一些扩展算法的接口与实现，如ECC和ElGamal算法。

❑ JCE支持者和测试包

包括org.bouncycastle.jce.provider包及其子包。本章提到的Bouncy Castle 的安全提供者BouncyCastleProvider就位于该包中。

❑ 轻量级加密包

包括org.bouncycastle.crypto包及其子包。我们可以认为这个包完成了扩展算法的实现。

❑ OCSP和OpenSSL PEM支持包

包括org.bouncycastle.ocsp包及其子包和org.bouncycastle.openssl包及其子包。这两个包都是与数字证书相关的支持包。OCSP（Online Certificate Status Protocol，在线证书状态协议）用于鉴定所需证书的（撤销）状态。具体协议内容请查看RFC 2560（http://www.ietf.org/rfc/rfc2560.txt）。OpenSSL用于管理数字证书，包括证书的申请和撤销等。

❑ ASN.1编码支持包

包括org.bouncycastle.asn1包及其子包，该包体积最为庞大。标准的ASN.1编码规则有基本编码规则（Basic Encoding Rules, BER）、规范编码规则（Canonical Encoding Rules, CER）、唯一编码规则（Distinguished Encoding Rules, DER）、压缩编码规则（Packed Encoding Rules, PER）和XML编码规则（XML Encoding Rules, XER）。我们在导出数字证书时，常常会使用到DER编码。

❑ 工具包

包括org.bouncycastle.util包及其子包。提供了很多与编码转换有关的工具类，如Base64编码和十六进制编码。

❑ 其他包

包括org.bouncycastle.mozilla包及其子包和org.bouncycastle.x509包及其子包。org.bouncycastle.mozilla包用于支持基于Mozilla（网景）浏览器的公钥签名和身份认证。org.bouncycastle.x509包用于基于支持X.509格式的数字证书。

相信不论是谁，在看了上述介绍后都会汗颜。当然，作者也不例外。作者有意避开Bouncy Castle支系庞大的API，建议读者将Bouncy Castle作为支持的方式来使用其扩展算法。这样可以降低学习难度，这也是Sun构建JCE架构最初的目的。

本书将简要介绍org.bouncycastle.util.encoders包中的内容。

1. Base64

Base64类是用于Base64编码的工具类。当然，Sun内部实现了Base64算法，但相关实现并未在API中体现，请读者参考第5章的内容。

```
// 用于Base64编码/解码转换。
public class Base64
extends Object
```

❑ 方法详述

我们先来看编码方法：

```
// 返回Base64编码结果。
public static byte[] encode(byte[] data)
// 向输出流中写入Base64编码结果。
public static int encode(byte[] data, int off, int length, OutputStream out)
// 向输出流中写入Base64编码结果。
public static int encode(byte[] data, OutputStream out)
```

以下是对应的解码方法：

```
// 返回Base64解码结果。
public static byte[] decode(byte[] data)
// 返回Base64解码结果，空格将被忽略。
public static byte[] decode(String data)
// 向输出流中写入Base64解码结果，空格将被忽略。
public static int decode(String data, OutputStream out)
```

其实，Base64类在内部实现时调用了Base64Encoder类来完成相应的编码/解码操作。我们只需要知道如何使用Base64类就可以了。

❑ 实现示例

我们通过代码清单4-1来演示如何进行Base64编码和解码操作。

代码清单4-1 Base64编码/解码1

```
String str = "base64编码";
System.err.println("原文:\t" + str);
byte[] input = str.getBytes();
// Base64编码
byte[] data = Base64.encode(input);
System.err.println("编码后:\t" + new String(data));
// Base64解码
byte[] output = Base64.decode(data);
System.err.println("解码后:\t" + new String(output));
```

查看控制台输出如下：

```
原文:       Base64 编码
编码后:     QmFzZTY0IOe8lueggQ==
解码后:     Base64 编码
```

编码后的内容中，出现“=”符号，这是Base64编码的标志性符号。

2. UrlBase64

Base64算法最初用于电子邮件系统，后经演变成为显式传递Url参数的一种编码算法，通

常称为“Url Base64”。它是Base64算法的变体，将字符映射表中用做补位的“=”换成“.”，并用“-”和“_”分别替代“+”和“/”，使得Base64编码符合Url参数规则，可以将二进制的数据以Get方式进行传输。有关Base64算法请参见第5章。

```
// 用于Url Base64编码/解码转换。
public class UrlBase64
extends Object
```

❑ 方法详述

以下为编码方法：

```
// 返回Url Base64编码。
public static byte[] encode(byte[] data)
```

以下方法将编码结果输出至输出流中：

```
// 向输出流中写入Url Base64编码。
public static int encode(byte[] data, OutputStream out)
```

以下为解码方法：

```
// 返回Url Base64解码，空格忽略。
public static byte[] decode(byte[] data)
// 返回Url Base64解码，空格忽略。
public static byte[] decode(String data)
```

以下方法可以将解码结果输出至输出流中：

```
// 向输出流中写入Url Base64解码，空格忽略。
public static int decode(byte[] data, OutputStream out)
// 向输出流中写入Url Base64解码，空格忽略。
public static int decode(String data, OutputStream out)
```

UrlBase64类在内部实现时，调用了UrlBase64Encoder类来完成相应的编码/解码操作。这是Bouncy Castle一贯的代码风格。

❑ 实现示例

我们对上述Base64算法实现稍作调整，改为Url Base64类完成编码和解码操作，如代码清单4-2所示。

代码清单4-2 Url Base64编码/解码

```
String str = "Base64 编码";
System.err.println("原文:\t" + str);
byte[] input = str.getBytes();
// Url Base64编码
byte[] data = UrlBase64.encode(input);
System.err.println("编码后:\t" + new String(data));
// Url Base64解码
byte[] output = UrlBase64.decode(data);
System.err.println("解码后:\t" + new String(output));
```

观察控制台输出的内容：

```
原文:       Base64编码
编码后:     QmFzZTY0IOe8lueggQ..
解码后:     Url Base64编码
```

编码后的内容中，出现了“.”符号，替换掉了“=”符号，符合了Url参数规则。当然，有些系统会对这个“.”符号较为敏感，如在文件系统中这可能导致一些错误。关于这个问题，请读者参考第5章的相关内容。

3. Hex

不用介绍，读者就能从字面上判断出Hex类和十六进制相关。Hex类用于十六进制转换。常配合消息摘要算法处理摘要值，以十六进制形式公示。

```
// 用于十六进制编码/解码操作。
public class Hex
extends Object
```

❑ 方法详述

以下是十六进制编码方法：

```
// 返回十六进制编码结果。
public static byte[] encode(byte[] data)
// 返回十六进制编码结果。
public static byte[] encode(byte[] data, int off, int length)
```

以下方法将编码结果输出至输出流中：

```
// 向输出流中写入十六进制编码结果。
public static int encode(byte[] data, int off, int length, OutputStream out)
// 向输出流中写入十六进制编码结果。
public static int encode(byte[] data, OutputStream out)
```

以下是十六进制解码方法：

```
// 返回十六进制解码结果。
public static byte[] decode(byte[] data)
// 返回十六进制解码结果，空格忽略。
public static byte[] decode(String data)
```

以下方法将解码结果输出至输出流中：

```
// 向输出流中写入十六进制解码结果，空格忽略。
public static int decode(String data, OutputStream out)
```

❑ 实现示例

我们继续对上述代码进行改造，如代码清单4-3所示：

代码清单4-3 Hex编码/解码1

```
String str = "Hex 编码";
System.err.println("原文:\t" + str);
```

```
byte[] input = str.getBytes();
// Hex编码。
byte[] data = Hex.encode(input);
System.err.println("编码后:\t" + new String(data));
// Hex解码。
byte[] output = Hex.decode(data);
System.err.println("解码后:\t" + new String(output));
```

观察控制台输出的内容：

```
原文:       Hex编码
编码后:     48657820e7bc96e7a081
解码后:     Hex编码
```

这时候看到编码后的字符串样式就很眼熟了，在各大软件厂商的下载页面上都很常见，通常用做MD5的十六进制表示。

Sun虽然在Java API中提供了简单的进制转换方法，但并不方便。相应的进制转换方法均包含在封装类中，如Long类的toHexString()方法可将长整型转换为十六进制字符串，并通过parseLong()方法，指定进制参数将十六进制字符串转换为长整型，参考代码如下：

```
// 长整型转换十六进制字符串。
String s = Long.toHexString(new Long(100));
// 十六进制字符串转换长整型。
long l = Long.parseLong(s, 16);
```

如果输入参数为字节数组，就不能使用上述方式来操作了。

对于一个长整型、整型、浮点型等数据类型来说，二进制转换虽然不复杂，我们完全有能力实现，但是转换过程中要考虑的细节很多，往往容易在补码、转码等问题上出现纰漏，这时就不如使用现成的开源实现。也许“不要重复制造轮子”的经典论断就是这么来的。

4.3 辅助工具Commons Codec

Commons Codec是Apache旗下的一款开源软件，主要用于编码格式的转换，如Base64、二进制、十六进制、字符集和Url编码的转换。甚至，Commons Codec还提供了语音编码的转换。除此之外，Commons Codec还对Java原生的消息摘要算法做了良好的封装，提高了方法的易用性。

4.3.1 获得辅助工具

Commons Codec目前提供的加密组件包的版本是1.4，这是一次重大的改进。作者明显感受到Commons Codec对Base64算法和消息摘要算法的支持便利性，是对上一个版本（1.3版）的一次升华。

我们可以直接登录Commons Codec官网（http://commons.apache.org/codec/），下载最新的组件包（http://commons.apache.org/codec/download_codec.cgi）。组件包分为两种压缩格式：

tar.gz和zip。我们以zip格式为例，commons-codec-1.4-bin.zip包含了二进制可执行文件、源代码和文档，并且在压缩包中都是以jar格式出现的，如图4-5所示。此外还可以获得纯粹的源代码文件commons-codec-1.4-src.zip。

图4-5 commons-codec-1.4-bin.zip文件中包含的相关文件

4.3.2 相关API

Commons Codec的API主要包含了以下几个方面：

- org.apache.commons.codec：该包内主要定义了一些编码转换的接口。
- org.apache.commons.codec.binary：该包内主要完成了编码转换实现，如Base64、二进制、十六进制和字符集编码。
- org.apache.commons.codec.digest：该包内仅有一个实现类DigestUtils，它是对Java原生消息摘要实现的改进。
- org.apache.commons.codec.language：该包内主要完成了语言和语音编码器实现。
- org.apache.commons.codec.net：该包内主要完成了网络相关的编码和解码，如Url编码/解码。

本书将主要阐述用于Base64和十六进制编码的转换实现，以及消息摘要算法相关实现。

1. Base64

Commons Codec也提供了用于Base64算法的实现类，与Bouncy Castle同名——Base64类。与Bouncy Castle提供的Base64类有所不同的是，Commons Codec提供的Base64类遵循RFC 2045（http://www.ietf.org/rfc/rfc2045.txt）。RFC全称为“Request For Comments”，意为请求注解。有关细节，请参见第5章的内容。

同时，这个Base64类实现了Bouncy Castle提供的UrlBase64类的Url编码功能。

```
// 用于Base64编码/解码转换。
public class Base64
implements BinaryEncoder, BinaryDecoder
```

❑ 方法详述

这个Base64类提供了相关的构造方法，用于控制构建一般Base64算法或Url Base64算法。以下两个构造方法互为映射：

```
// 构建Base64算法实现
public Base64()
// 构建Url Base64算法实现
public Base64(boolean urlSafe)
```

以下构造方法指定了每行字符个数：

```
// 构建指定每行字符数的Base64算法实现。
public Base64(int lineLength)
```

在上述构造方法的基础上，加入了回车换行符的控制：

```
// 构建指定每行字符数的Base64算法实现，并控制每行字符数和行末符号。
public Base64(int lineLength, byte[] lineSeparator)
```

以下构造方法提供了是否支持Url Base64算法的支持：

```
/* 构建指定每行字符数的Base64算法实现，并控制每行字符数和行末符号及是否支持Url Base64算法。*/
public Base64(int lineLength, byte[] lineSeparator, boolean urlSafe)
```

我们先来了解Base64类的静态方法。

下面是最为常用的Base64编码方法：

```
// 以字节数组形式返回Base64编码结果。
public static byte[] encodeBase64(byte[] binaryData)
```

这个方法返回的是一般Base64编码，与之不同的是下面两种方法：

```
/* 以字节数组形式返回Base64编码结果，输出结果中每76个字符追加一个回车换行符。*/
public static byte[] encodeBase64Chunked(byte[] binaryData)
// 以字符串形式返回Base64编码结果，输出结果中每76个字符追加一个回车换行符。
public static String encodeBase64String(byte[] binaryData)
```

在内部实现上，上述方法均调用了以下这种方法：

```
/* 以字节数组形式返回Base64编码结果，对输出结果中每76个字符追加一个回车换行符可控。*/
public static byte[] encodeBase64(byte[] binaryData, boolean isChunked)
```

对应上述编码方法，以下是相应的解码方法：

```
// 以字节数字形式返回Base64解码结果。
public static byte[] decodeBase64(byte[] base64Data)
// 以字节数字形式返回Base64解码结果。
public static byte[] decodeBase64(String base64String)
```

在4.2.3节中，我们提到了Url Base64算法。主要差别是将原Base64字符映射表中的“+”和“/”替换为“-”和“_”。Commons Codec的Base64类也完成了同样的实现，提供了如下方法：

```
// 以字节数组形式返回Url Base64编码结果。
public static byte[] encodeBase64URLSafe(byte[] binaryData)
```

```
// 以字符串形式返回Url Base64编码结果。
public static String encodeBase64URLSafeString(byte[] binaryData)
```

上述两种方法实际上调用了如下方法：

```
/* 以字节数组形式返回Base64编码结果，是否加入回车换行符可控，是否使用Url Base64编码可控。*/
public static byte[] encodeBase64(byte[] binaryData, boolean isChunked, boolean urlSafe)
```

下述方法则更为细致一些：

```
/*  以字节数组形式返回Base64编码结果，对结果中每行多少个字符、是否加入回车换行符可控，是否使用
Url Base64编码可控。*/
public static byte[] encodeBase64(byte[] binaryData, boolean isChunked, boolean
urlSafe, int maxResultSize)
```

在这个Base64类中，除了提供了对于字节数组和字符串形式的编码/解码转换，同时也提供了BigInteger类型和字节数组形式的编码/解码转换。

```
// 以字节数组形式返回Base64编码结果。
public static byte[] encodeInteger(BigInteger bigInt)
// 以BigInteger形式返回Base64解码结果。
public static BigInteger decodeInteger(byte[] pArray)
```

如果需要判断一个字节数组（或一个字节）中是否包含了Base64字符映射表中的字符，可以使用如下方法：

```
// 测试输入的字节数组是否包含Base64字符映射表中的字符。
public static boolean isArrayByteBase64(byte[] arrayOctet)
// 判别输入字节是否在Base64字符映射表中。
public static boolean isBase64(byte octet)
```

在对上述静态方法了解一番后，下述方法理解起来就相对容易多了。它们绝大部分调用了上述静态方法。

以下方法互为Base64编码/解码实现：

```
// 以字节数组形式返回Base64编码结果。
public byte[] encode(byte[] pArray)
// 以字节数组形式返回Base64解码结果。
public byte[] decode(byte[] pArray)
```

如果需要字节数组和字符串之间的Base64编码/解码实现，可使用如下方法：

```
// 以字节数组形式返回Base64解码结果。
public byte[] decode(String pArray)
// 以字符串形式返回Base64编码结果。
public String encodeToString(byte[] pArray)
```

以下方法用于对象形式的Base64编码/解码操作：

```
// 以对象形式返回Base64编码结果。
public Object encode(Object pObject)
// 以对象形式返回Base64解码结果。
public Object decode(Object pObject)
```

此外，完成Base64初始化后，你可以通过如下方法获知当前实例化对象是否支持Url Basea64算法。

```
// 判别是否是Url Base64编码。
public boolean isUrlSafe()
```

❑ 实现示例

对应4.2.3节中Base64算法的编码解码实现如代码清单4-4所示。

代码清单4-4 Base64编码/解码2

```
String str = "Base64 编码";
System.err.println("原文:\t" + str);
byte[] input = str.getBytes();
// Base64编码
byte[] data = Base64.encodeBase64(input);
System.err.println("编码后:\t" + new String(data));
// Base64解码
byte[] output = Base64.decodeBase64(data);
System.err.println("解码后:\t" + new String(output));
```

控制台输出与4.2.3节中一致：

```
原文:       Base64 编码
编码后:     QmFzZTY0IOe8lueggQ==
解码后:     Base64 编码
```

对应4.2.3节中Url Base64算法的编码解码实现如下：

```
String str = "Base64 编码";
System.err.println("原文:\t" + str);
byte[] input = str.getBytes();
// Url Base64编码
byte[] data = Base64.encodeBase64URLSafe(input);
System.err.println("编码后:\t" + new String(data));
// Url Base64解码
byte[] output = Base64.decodeBase64(data);
System.err.println("解码后:\t" + new String(output));
```

注意控制台输出结果为：

```
原文:       Base64 编码
编码后:     QmFzZTY0IOe8lueggQ
解码后:     Base64 编码
```

读者需要注意这一点，Bouncy Castle和Commons Codec对于Url Base64算法的理解上有所不同。关于Url Base64算法，本身没有一个统一的标准，没有像RFC 2045这样的明文规定。

两者不同之处在于对原Base64算法中补位概念的理解，如下所示：

❑ Bouncy Castle：使用“.”符号替代“=”进行补位。

❑ Commons Codec：不进行补位。

在使用Url Base64算法时，需要注意选择合适的实现。

2. Base64InputStream

Commons Codec在1.4版本中，加入了对Base64输入/输出流的支持，包含Base64InputStream和Base64OutputStream两个类。

```
// 完成Base64编码/解码输入流操作。
public class Base64InputStream
extends FilterInputStream
```

❑ 方法详述

既然是流操作，就一定有相应的构造方法。Base64InputStream类可通过如下构造方法获得实例化对象：

```
// 通过输入流构造，默认不支持Base64编码。
public Base64InputStream(InputStream in)
```

如果需要支持Base64编码，需要使用如下方法，将doEncode参数设置为true。

```
// 通过输入流构造，可设置是否支持Base64编码。
public Base64InputStream(InputStream in, boolean doEncode)
```

以下方法还指定了每行字符数及行末符号：

```
// 通过输入流构造，可设置是否支持Base64编码，并指定每行字符数及行末符号。
public Base64InputStream(InputStream in, boolean doEncode, int lineLength,
byte[] lineSeparator)
```

获得输入流最主要的操作就是进行读操作了，Base64InputStream类提供如下两种方法：

```
// 每次读一个字节。
public int read()
// 按偏移量读入字节数组。
public int read(byte[] b, int offset, int len)
```

下述方法在当前版本未实现：

```
// 用于测试是否支持标记和重置操作，未实现返回值为false。
public boolean markSupported()
```

其余FilterInputStream类提供的方法，Base64InputStream类未覆盖。

❑ 实现示例

如果在通信环境中，消息收发双方使用Base64算法对消息隐蔽就可以使用Base64InputStream和Base64OutputStream类了。我们通过代码清单4-5来演示如何接收发送方传递过来的Base64编码消息。

代码清单4-5 Base64输入流操作

```
// 实例化Base64InputStream，用作Base64解码。
Base64InputStream input = new Base64InputStream(is, false);
// 使用DataInputStream包装Base64InputStream。
DataInputStream dis = new DataInputStream(input);
```

```
// 信息体
byte[] data = new byte[contentLength];
// 读入
dis.readFully(data);
// 关闭流
dis.close();
// 控制台输出
System.err.println(new String(data));
```

其中，is为网络输入流，contentLength为网络流长度。通过上述实现，在控制台中，我们获得如下内容：

```
Base64 解码
```

在网络流完成读取后，获得了解码结果。数据是如何编码并转换为网络流呢？请读者关注下面的内容。

3. Base64OutputStream

Base64OutputStream类自然是对Base64算法输出流的支持。

```
// 完成Base64编码/解码输出流操作。
public class Base64OutputStream
extends FilterOutputStream
```

❑ 方法详述

Base64OutputStream类可通过如下构造方法获得实例化对象：

```
//通过输入流构造，默认不支持Base64编码。
public Base64OutputStream(OutputStream out)
```

如果需要Base64编码需要使用如下构造方法，并将doEncode设置为true：

```
// 通过输入流构造，可设置是否支持Base64编码。
public Base64OutputStream(OutputStream out, boolean doEncode)
```

同时，我们还可以在构造操作时指定每行字符数及行末符号，如下方法所示：

```
// 通过输入流构造，可设置是否支持Base64编码，并指定每行字符数及行末符号。
public Base64OutputStream(OutputStream out, boolean doEncode, int lineLength,
byte[] lineSeparator)
```

既然是输出流，最主要的就是写操作了，也就是如下方法：

```
// 写操作，写入b[]中。
public void write(byte[] b, int offset, int len)
// 写操作，写入b[]中。
public void write(int i)
```

完成写操作后，需要执行如下操作：

```
// 清空流
public void flush()
// 关闭流
public void close()
```

❑ 实现示例

在消息发送时对输出流做相应实现如代码清单4-6所示。

代码清单4-6 Base64输出流操作

```
// 实例化Base64OutputStream，用作Base64编码。
Base64OutputStream output = new Base64OutputStream(os, true);
// 使用DataOutputStream包装Base64OutputStream。
DataOutputStream dos = new DataOutputStream(output);
// 写操作
dos.write(data);
// 清空
dos.flush();
// 关闭流
dos.close();
```

其中，os是网络输出流，data中是我们传输的消息。

相信读者已经猜到作者传送什么内容了，参见如下代码：

```
String str = "Base64 编码";
byte[] data = str.getBytes();
```

4. Hex

Commons Codec也提供了用于十六进制编码/解码转换的实现类，与Bouncy Castle所提供的类同名，也叫做Hex类。所不同的是，Commons Codec提供了更为细致的API。

```
// 用于十六进制编码/解码转换。
public class Hex
implements BinaryEncoder, BinaryDecoder
```

❑ 方法详述

Hex类与Base64类相类似，既提供静态方法，也提供对应接口实现的方法。以下为该类的构造方法：

```
// 空构造方法，使用默认字符集。
public Hex()
// 根据指定字符集构造。
public Hex(String csName)
```

我们先来了解Hex类的静态方法。

Hex类提供了4种静态方法，以下是编码操作方法：

```
// 以字符数组形式返回十六进制编码结果。
public static char[] encodeHex(byte[] data)
// 以字符串形式返回十六进制编码结果。
public static String encodeHexString(byte[] data)
```

有时候我们对输出的十六进制编码结果有要求，字母必须大写/小写，可以使用如下方法：

```
// 以字符数组形式返回十六进制编码结果，对结果字符大小写可控。
public static char[] encodeHex(byte[] data, boolean toLowerCase)
```

上述3种方法中，最为常用的非encodeHexString()方法莫属了。

尽管编码方法很多，但解码方法只有这一个：

```
// 以字节数组形式返回十六进制解码结果。
public static byte[] decodeHex(char[] data)
```

以下两种方法，实际上是调用了上述相应的静态方法：

```
// 以字节数组形式返回十六进制编码结果。
public byte[] encode(byte[] array)
// 以字节数组形式返回十六进制解码结果。
public byte[] decode(byte[] array)
```

以下是对对象的编码/解码方法：

```
// 以对象的形式返回对对象十六进制编码结果。
public Object encode(Object object)
// 以对象的形式返回对对象十六进制解码结果。
public Object decode(Object object)
```

对于对象的编码/解码操作，实际上是先判断输入参数是否是String类型，如果不是则强制转换为byte[]类型。如果转换失败，则抛出异常。

此外，Hex类还提供了如下方法：

```
// 获得字符集名称
public String getCharsetName()
// 转换字符串输出
public String toString()
```

❑ 实现示例

参考4.2.3节中十六进制编码/解码实现示例，做相应修改，如代码清单4-7所示。

代码清单4-7 Hex编码/解码2

```
String str = "Hex 编码";
System.err.println("原文:\t" + str);
byte[] input = str.getBytes();
// Hex编码
String data = Hex.encodeHexString(input);
System.err.println("编码后:\t" + data);
// Hex解码
byte[] output = Hex.decodeHex(data.toCharArray());
System.err.println("解码后:\t" + new String(output));
```

观察控制台输出的内容，一定是一样的结果：

```
原文:       Hex 编码
编码后:     48657820e7bc96e7a081
解码后:     Hex 编码
```

5. DigestUtils

DigestUtils类是对Sun提供的MessageDigest类进行了一次封装，提供了更为实用的算法支

持，弥补了消息摘要结果十六进制编码转换的缺憾。

```
// 用于实现MD5和SHA系列的消息摘要算法。
public class DigestUtils
```

❑ 方法详述

DigestUtils类是一个工具类，它提供了MD5和SHA系列消息摘要算法的实现。

以下是DigestUtils类提供的用于完成MD5算法的静态方法：

```
// 以字节数组形式返回MD5消息摘要信息。
public static byte[] md5(byte[] data)
// 以字节数组形式返回MD5消息摘要信息。
public static byte[] md5(InputStream data)
// 以字节数组形式返回MD5消息摘要信息。
public static byte[] md5(String data)
```

下述方法提供了MD5算法十六进制字符串形式的结果：

```
// 以十六进制字符串形式返回MD5消息摘要信息。
public static String md5Hex(byte[] data)
// 以十六进制字符串形式返回MD5消息摘要信息。
public static String md5Hex(InputStream data)
// 以十六进制字符串形式返回MD5消息摘要信息。
public static String md5Hex(String data)
```

DigestUtils类提供了Java 6所支持的全部SHA系列算法，包括SHA-1、SHA-256、SHA-384和SHA-512算法。

以下是DigestUtils类提供的用于完成SHA-1算法的静态方法：

```
// 以字节数组形式返回SHA消息摘要信息。
public static byte[] sha(byte[] data)
// 以字节数组形式返回SHA消息摘要信息。
public static byte[] sha(InputStream data)
// 以字节数组形式返回SHA消息摘要信息。
public static byte[] sha(String data)
```

下述方法提供了SHA-1算法十六进制字符串形式的结果：

```
// 以十六进制字符串形式返回SHA消息摘要信息。
public static String shaHex(byte[] data)
// 以十六进制字符串形式返回SHA消息摘要信息。
public static String shaHex(InputStream data)
// 以十六进制字符串形式返回SHA消息摘要信息。
public static String shaHex(String data)
```

以下是DigestUtils类提供的用于完成SHA-256算法的静态方法：

```
// 以字节数组形式返回SHA-256消息摘要信息。
public static byte[] sha256(byte[] data)
// 以字节数组形式返回SHA-256消息摘要信息。
public static byte[] sha256(InputStream data)
```

```
// 以字节数组形式返回SHA-256消息摘要信息。
public static byte[] sha256(String data)
```

下述方法提供了SHA-256算法十六进制字符串形式的结果：

```
// 以十六进制字符串形式返回SHA-256消息摘要信息。
public static String sha256Hex(byte[] data)
// 以十六进制字符串形式返回SHA-256消息摘要信息。
public static String sha256Hex(InputStream data)
// 以十六进制字符串形式返回SHA-256消息摘要信息。
public static String sha256Hex(String data)
```

以下是DigestUtils类提供的用于完成SHA-384算法的静态方法：

```
// 以字节数组形式返回SHA-384消息摘要信息。
public static byte[] sha384(byte[] data)
// 以字节数组形式返回SHA-384消息摘要信息。
public static byte[] sha384(InputStream data)
// 以字节数组形式返回SHA-384消息摘要信息。
public static byte[] sha384(String data)
```

下述方法提供了SHA-384算法十六进制字符串形式的结果：

```
// 以十六进制字符串形式返回SHA-384消息摘要信息。
public static String sha384Hex(byte[] data)
// 以十六进制字符串形式返回SHA-384消息摘要信息。
public static String sha384Hex(InputStream data)
// 以十六进制字符串形式返回SHA-384消息摘要信息。
public static String sha384Hex(String data)
```

以下是DigestUtils类提供的用于完成SHA-512算法的静态方法：

```
// 以字节数组形式返回SHA-512消息摘要信息。
public static byte[] sha512(byte[] data)
// 以字节数组形式返回SHA-512消息摘要信息。
public static byte[] sha512(InputStream data)
// 以字节数组形式返回SHA-512消息摘要信息。
public static byte[] sha512(String data)
```

下述方法提供了SHA-512算法十六进制字符串形式的结果：

```
// 以十六进制字符串形式返回SHA-512消息摘要信息。
public static String sha512Hex(byte[] data)
// 以十六进制字符串形式返回SHA-512消息摘要信息。
public static String sha512Hex(InputStream data)
// 以十六进制字符串形式返回SHA-512消息摘要信息。
public static String sha512Hex(String data)
```

❑ 实现示例

我们以MD5算法摘要处理为例，如代码清单4-8所示。

代码清单4-8 MD5摘要处理

```
String inputStr = "MD5消息摘要";
```

```
System.err.println("原文：\t" + inputStr);
// 执行MD5消息摘要
String md5Hex = DigestUtils.md5Hex(inputStr);
System.err.println("加密后：\t" + md5Hex);
```

注意控制台的输出：

```
原文：      MD5消息摘要
加密后：    aa24863099cf24696d4eb0f82c918849
```

原文MD5处理后，获得了一个32个字符的十六进制字符串。

有关消息摘要算法相关内容，请参见第6章。

4.4 小结

受限于美国出口限制，我们获得的JDK（或JRE）本身不具备很高的加密强度。在当地的相应进出口政策允许的前提下，我们可以从Sun官方网站上下载相应的权限文件，在一定范围内提高JDK（JRE）的加密强度。

获得Sun官方网站提供的权限文件后，我们只需要在对应的目录下，进行权限覆盖即可。

Bouncy Castle作为一款开源加密组件，极大地丰富了Java世界的加密算法支持。

Sun提供了JCE框架用于支持各种加密算法。基于这个架构，Bouncy Castle提供了相应的安全提供者，扩展了Java 6所不支持的多种算法，如MD4、SHA-224、HmacMD4和HmacSHA-224等消息摘要算法，IDEA对称加密算法以及ECDSA数字签名算法。

使用Bouncy Castle提供者有两种方式：第一种是配置方式，修改JDK（JRE）环境，也就是修改%JDK_Home%\ jre\lib\security\java.security文件（%JRE_Home%\ lib\security\java.security文件），增加Bouncy Castle提供者；第二种是显示调用方式，在使用Bouncy Castle所提供的扩展算法时，先要在各种密钥工厂、密钥对生成器等引擎初始化前，使用Security类的addProvider()方法加入Bouncy Castle提供者。

两种方式各具优势：前者需要依赖环境，但代码十分简洁。开发者必须知道环境对这些扩展算法已支持，无须关注由谁提供。在做迁移时，如项目上线时需要注意相关环境配置；后者需要依赖代码修改，需要使用者十分清楚所需要的扩展算法如何使用及由谁提供。

从架构和开发的角度来考虑：前者属于架构层面，更注重整体；后者注重开发细节，注重单模块。读者需根据开发需求选择合理的实现方式。

对于Base64、十六进制转换算法，Bouncy Castle也提供了相应的支持。此外，Bouncy Castle提供了用于Url编码的UrlBase64类。

Commons Codec是国际开源组织Apache旗下的一款开源组件，提供了各种编码转换实现，如Base64、二进制、十六进制、语音等多种编码转换，并且在Sun提供的消息摘要算法的基础上做了进一步封装，提高了各种算法操作的易用性。

Commons Codec提供的Base64类除了完成一般Base64算法实现，同时实现了Url Base64算法实现。

鱼和熊掌不能兼得，请读者根据相应需求选择Bouncy Castle或Commons Codec。