Zxx:请将将Serializable的作用？

Java内存有一个对象，想持久化存储。之后又想把这个持久化存储的对象恢复到硬盘上 这个就是序列化和反序列化

Sun公司发现这种很常见，所以Sun公司就统一规定了序列化的文件的格式。所以 想使用的话，就类就是实现Sun的Seriallizable接口

注意 实现了这个java.io.Seriablizable接口的类在javac编译的时候，就会被识别，这样javac就会对这样的接口进行特殊的编译

之后使用sun的ObjectOutputStream.writeObject(obj)进行写



这个类可能持久化到本地 也可能持久化到网络传输到别的地方

所以 这个流是一个包装流



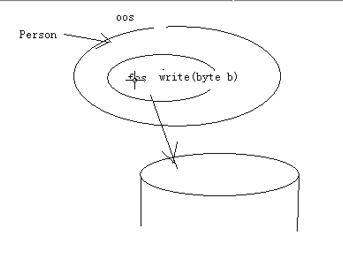
===== 包装流的思想：

物理设备（硬盘或者网络）

内存中有一个流对象 fos(fos直接进行系统调用) 和oos

对于进行系统调用的fos，有write(byte b) 或者是write(byte[] b) ---- 写的要么时候字符串 要么是字符数组【字符数组不就是字符串么 C语言中就是这么定义的】要么就是单一的字节

那么如果你要写入的是Java的内存对象，这个不是简单地字节了，那么就首先把这个交给oos ---- 由oos把复杂的Java内存对象转换成字节数组byte[] ---- 然后交给可以调用系统调用的fos ---- 把转换好的byte[]写入到磁盘上面



这样 就省的程序员自己把Java对象转换成字节数组

【String本身就是含有编码的byte[] 所以 String和 byte[]可以直接转换 但是 任意对象就不知道如何转换了】

这种包装流仅仅是对数据进行中转作用，交给另外一个流 所以包装流的构造方法必须传一个底层的关联的资源的流

场景：假如双方都想使用一个已经双方知道的字符串密码 来进行解密

刚才的场景：是代码中生成随机的一个密钥 供加密解密使用。但是 如果想把这个密钥固定 双方都知道 不是那种二进制的序列化文件 而是



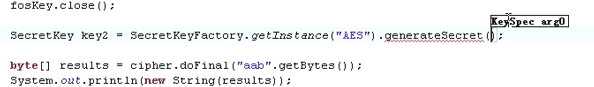
修改一下代码，Java程序加密的时候 仅仅认识key 123456这样的密码

Sun公司提供了一个工具 仅仅需要记住这个字符串 sun的工具可以把这个转成加密解密需要的Key的格式

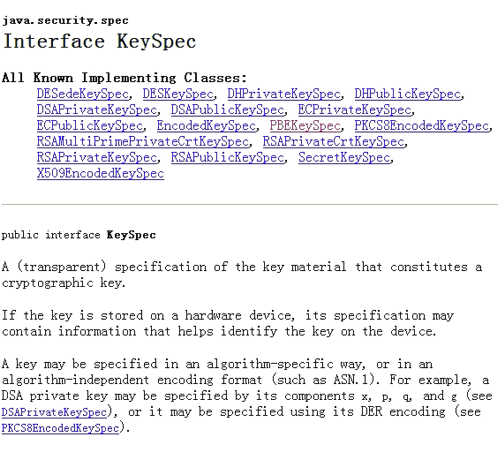


使用SecretKeyFactory来生成Key 这个arg0就是你的明文的密码

这个是创建密钥的工厂 用来产生密钥 generateSercret()

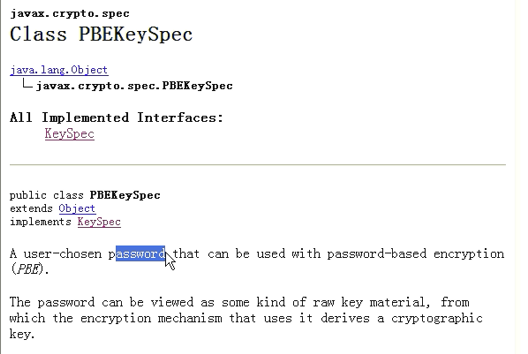


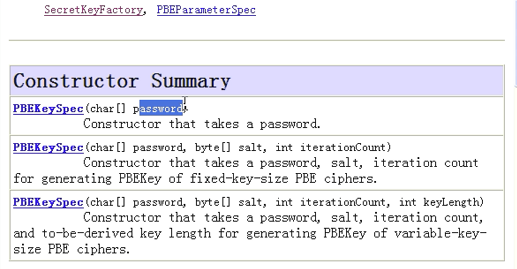
注意到这个参数是KeySpec类型的参数 这个新的类 那就要看一下文档：



这个有一个子类 PBEKeySpec ----- 基于密码的加密

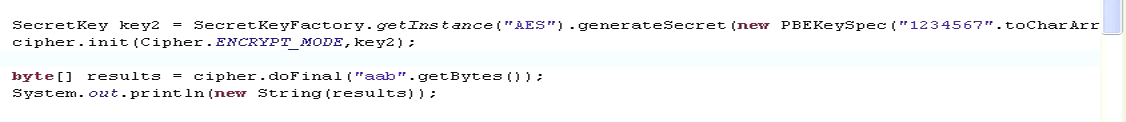
java.crypto.spec.PBEKeySpec





这个PBEKeySpec() ---参数是字符串数组

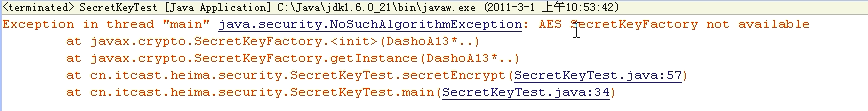
基于密码的验证

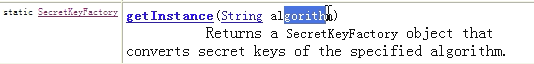


下面来解密

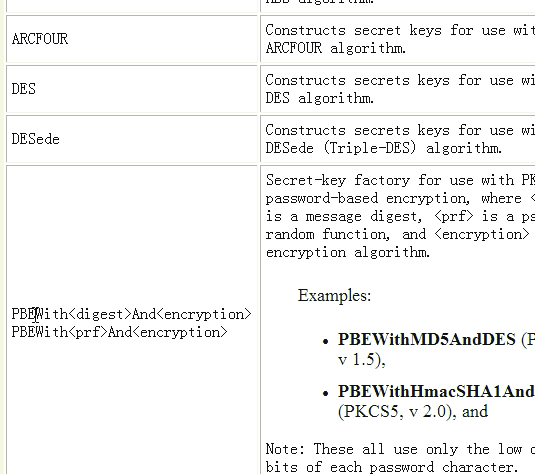
Key还是用key2

这样两边记住的都是密码

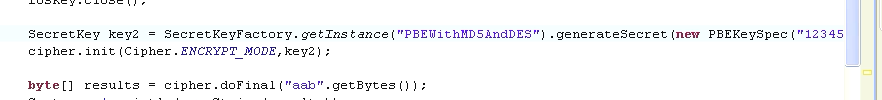


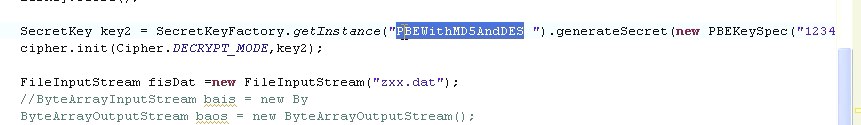


算法的名字 如果瞎写 就会报错



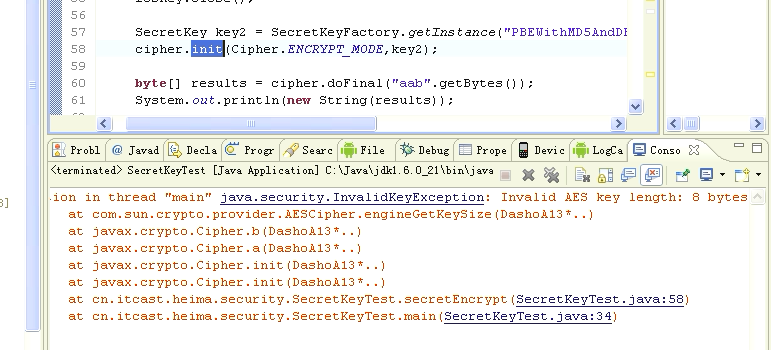
这个有一个PBEWithxxx



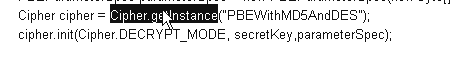


报错 说key的长度小：

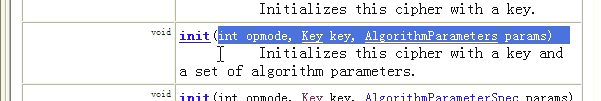


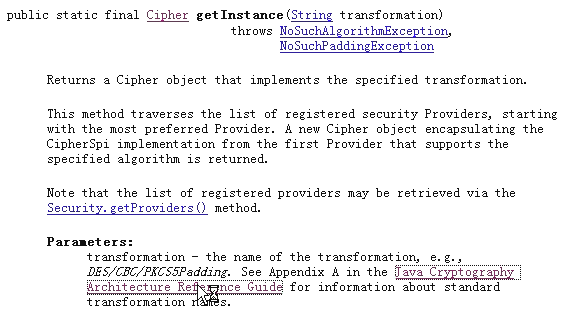


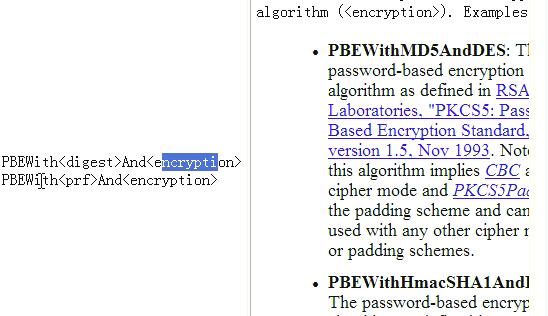
看了一下PPT 说这个加密的算法也要改 并且 加盐 parameterSpec



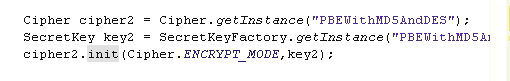
Cipher的新初始化方法 ---- 这个叫AlgorithmParameters params就是盐 为了确保加密更加安全





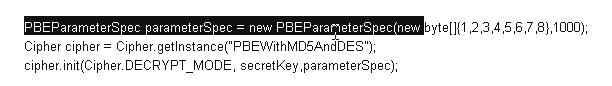


这样Cipher和PBEKeySpec中指定的算法是一模一样的



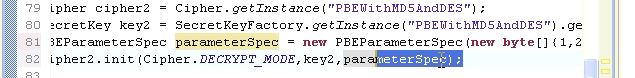
没加盐

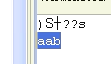




需要加盐

加密和解密都改一下



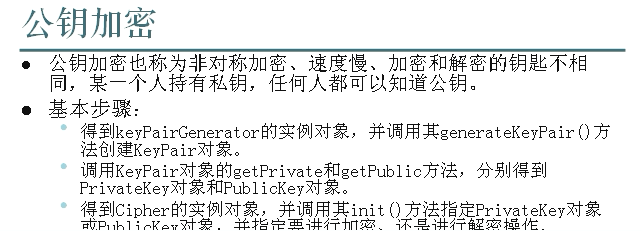


这种就是双方都记住字符串就可以了

下面是非对称加密

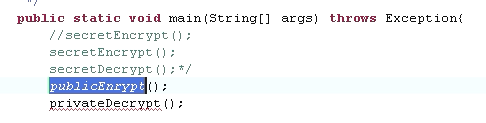
对称加密的key两种方式 --- 存在硬盘文件上（比如银行给的U盾） 另一种是以字符串记在脑子里（平时自己使用）

公钥加密 在大型的网站 企业用 android用得少

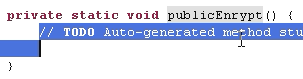


公钥加密的缺点就是慢一点

直接写代码 公钥加密和私钥解密



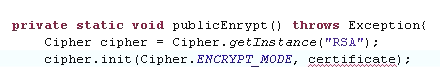
测试代码中构造的方法



还是使用Cipher 只是其中的算法变了

看一下Cipher里面的算法 DES AES都是对称加密

RSA是非对称加密



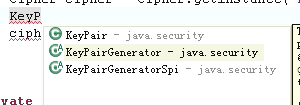
这个时候的key是公钥

产生公钥的办法

是一对钥匙



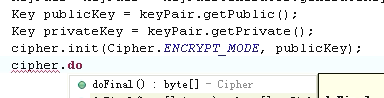
这个还有KeyGenerator



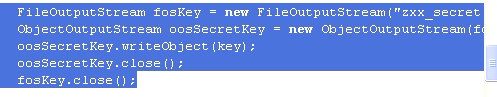
因为是一对 所以使用KeyPairGenerator

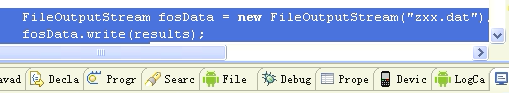


获取一下私钥个公钥

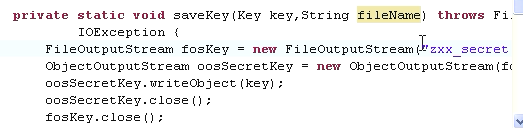


【\*\*Cipher加密解密的方法的输入都是二进制数据】





把这个保存密码的代码 和数据的方法抽取成方法

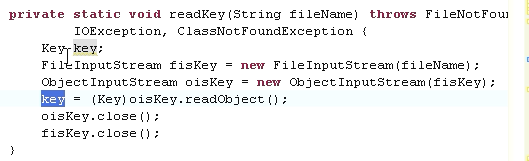


还有一个saveData 保存数据

 ---保存私钥

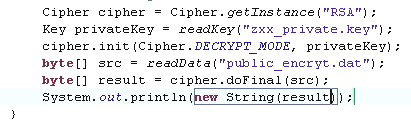
现在进行私钥解密

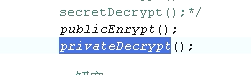
首先恢复key【也就是从文件中读出来这个key】



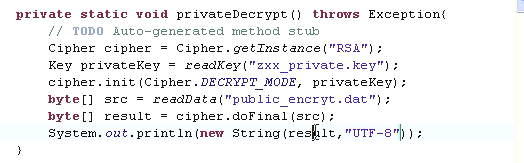
readKey(), readData() ---- 这两个方法

读出来之后 进行解密



公钥加密 私钥解密



指定编码

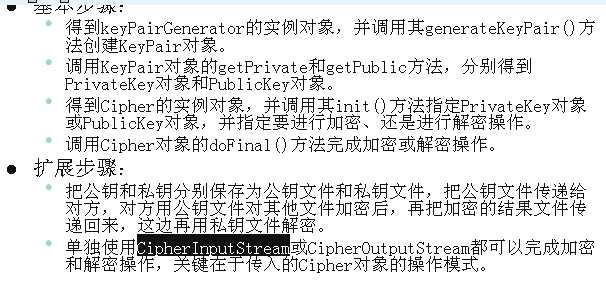
如果不告诉 就会使用操作系统默认的字符集GBK



运行的时候 会感觉到有一点慢

\*\*上面就是对称加密和非对称加密

【保存密码的过程就相当于传输了这个密钥到另一方】



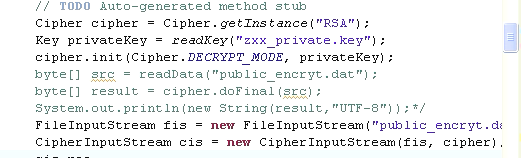
扩展 ---- 还有一个CipherInputStream和CipherOutputStream这两个可以完成加密和解密操作

把一个文件装载到一个字节数组中 很多人写不出来 绝对不是定义一个很大的数组【我以前也不会写】

如果觉得这个难，可以使用这两个流来替代



现在 对这个文件进行解密



这样 通过文件+解密器构造构造一个加密输入流 ---- 这样加密读取文件的工作都在这里面了 并且 进行了解密或者加密

这里面就是解密 那么read出来的结果就是解密后的结果

[记得把流关掉]

老张这里面由于数据量小 偷懒了 使用了byte[1024]

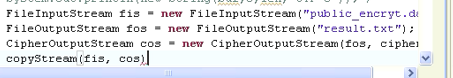
 ----- 这段代码数据量很小 没关系 但是很大的情况 是不能用的

下面使用CipherOutputStream这个类来实现解密

前面使用的是CipherInputStream



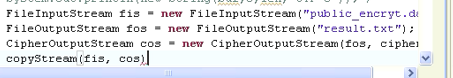
这个就是解密之后的数据 写到fos关联的磁盘文件中



那么数据从哪里来呢？

【\*\*思路 使用CipherOutputStream解密的方式

因为CipherOutputStream是一种输出流 ----- 所以 数据是可以写到输出流 ---- 这样 公钥加密的数据可以通过输入流写出到Cipher输出流中，然后CipherOutputStream会根据自身的加密器cipher来解密流中的数据

 ---- 所以 有一个copy(fis, cos)这样的一个数据传递的过程

\*\*思路 使用CipherInputStream解密的方式

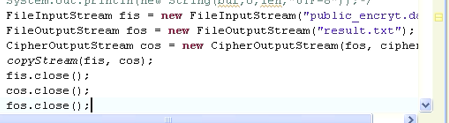
因为CipherInputStream是一种输入流 ---- 所以 CipherInputStream构造方式一定能有一个文件输入流传递进来 这样 解密或者加密都不愁数据的来源 本身又有一个cipher 这个是对数据进行加密或者解密的工具 ---- 只不过加密解密出来的数据都通过read方法读到了内存中】

【同样道理 CipherInputStream可以实现加密

所以 CipherInputStream或者CipherOutputStream本身加密或者解密不是输入流或者输出流来决定的 是 本身的持有的cipher对象的模式决定的！

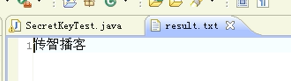
这个输入流和输出流的好处就在于，把数据从磁盘加载到内存/把内存的数据存储到磁盘的过程 是封装在这两个包装流里面就可以了 ----- 实际上 把未知大小的文件读到磁盘的大小固定的数组中 是一件有相对有难度的工作

因此CipherInputStream和CipherOutputStream这两个类都能实现加密或者解密 --- 因为CipherXXx中的Cipher功能就是双向的】



【copyStream里面的代码实际上是调用了fis的read(bytep[]) 🡪 cos.write(byte[]) 这样 数据源源不断从fis关联的磁盘文件转移到了CipherOutputStream中】

打开文件



CipherInputStream --- 读的过程中就加密或者解密了

CipherOutputStream ----写的过程中就加密或者解密了