X.509 数字证书读入程序实现

17343128 幸赟

1.x.509 证书结构描述

（1）基本部分

1.1.版本号.

标识证书的版本（版本 1、版本 2 或是版本 3）。

1.2.序列号

标识证书的唯一整数，由证书颁发者分配的本证书的唯一标识符。

1.3.签名

用于签证书的算法标识，由对象标识符加上相关的参数组成，用于说明本证书所用的数字签名算法。例如，SHA-1 和 RSA 的对象标识符就用来说明该数字签名是利用 RSA 对 SHA-1 杂凑加密。

1.4.颁发者

证书颁发者的可识别名（DN）。

1.5.有效期

证书有效期的时间段。本字段由”Not Before”和”Not After”两项组成，它们分别由 UTC 时间或一般的时间表示（在 RFC2459 中有详细的时间表示规则）。

1.6.主体

证书拥有者的可识别名，这个字段必须是非空的，除非你在证书扩展中有别名。

1.7.主体公钥信息

主体的公钥（以及算法标识符）。

1.8.颁发者唯一标识符

标识符—证书颁发者的唯一标识符，仅在版本 2 和版本 3 中有要求，属于可选项。

1.9.主体唯一标识符

证书拥有者的唯一标识符，仅在版本 2 和版本 3 中有要求，属于可选项。

（2）证书扩展部分

可选的标准和专用的扩展（仅在版本 2 和版本 3 中使用），扩展部分的元素都有这样的结构：

```
Extension ::= SEQUENCE {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING }
```

extnID：表示一个扩展元素的 OID

critical：表示这个扩展元素是否极重要

extnValue：表示这个扩展元素的值，字符串类型。

扩展部分包括：

2.1.发行者密钥标识符

证书所含密钥的唯一标识符，用来区分同一证书拥有者的多对密钥。

2.2.密钥使用

一个比特串，指明（限定）证书的公钥可以完成的功能或服务，如：证书签名、数据加密等。

如果某一证书将 KeyUsage 扩展标记为“极重要”，而且设置为“keyCertSign”，则在 SSL 通信期间该证书出现时将被拒绝，因为该证书扩展表示相关私钥应只用于签写证书，而不应该用于 SSL。

2.3. CRL 分布点

指明 CRL 的分布地点。

2.4.私钥的使用期

指明证书中与公钥相联系的私钥的使用期限，它也有 Not Before 和 Not After 组成。若此项不存在时，公私钥的使用期是一样的。

2.5.证书策略

由对象标识符和限定符组成，这些对象标识符说明证书的颁发和使用策略有关。

2.6.策略映射

表明两个 CA 域之间的一个或多个策略对象标识符的等价关系，仅在 CA 证书里存在。

2.7.主体别名

指出证书拥有者的别名，如电子邮件地址、IP 地址等，别名是和 DN 绑定在一起的。

2.8. 颁发者别名

指出证书颁发者的别名，如电子邮件地址、IP 地址等，但颁发者的 DN 必须出现在证书的颁发者字段。

2.9. 主体目录属性

指出证书拥有者的一系列属性。可以使用这一项来传递访问控制信息。

（3）x.509 证书详细描述

```
Certificate ::= SEQUENCE {
        tbsCertificate       TBSCertificate, -- 证书主体
        signatureAlgorithm   AlgorithmIdentifier, -- 证书签名算法标识
        signatureValue       BIT STRING --证书签名值,是使用
                                        signatureAlgorithm 部分指
                                        定的签名算法对
                                        tbsCertificate 证书主题部分
                                        签名后的值.
        }
   TBSCertificate ::= SEQUENCE {
        version         [0] EXPLICIT Version DEFAULT v1, -- 证书版本号
```

serialNumber CertificateSerialNumber, -- 证书序列号，对同一CA 所颁发的证书，序列号唯一标识证书

signature AlgorithmIdentifier, --证书签名算法标识

issuer Name, --证书发行者名称

validity Validity, --证书有效期

subject Name, --证书主体名称

subjectPublicKeyInfo SubjectPublicKeyInfo,--证书公钥

issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,

-- 证书发行者 ID(可选)，只在证书版本 2、3 中才有

subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,

-- 证书主体 ID(可选)，只在证书版本 2、3 中才有

extensions [3] EXPLICIT Extensions OPTIONAL

-- 证书扩展段（可选），只在证书版本 3 中才有

}

Version ::= INTEGER { v1(0), v2(1), v3(2) }

CertificateSerialNumber ::= INTEGER

AlgorithmIdentifier ::= SEQUENCE {

algorithm OBJECT IDENTIFIER,

parameters ANY DEFINED BY algorithm OPTIONAL }

parameters:

Dss-Parms ::= SEQUENCE { -- parameters ，DSA(DSS)算法时的 parameters,

RSA 算法没有此参数

p INTEGER,

q INTEGER,

g INTEGER }

signatureValue：

Dss-Sig-Value ::= SEQUENCE { -- sha1DSA 签名算法时,签名值

```
                r       INTEGER,
                s       INTEGER }

  Name ::= CHOICE {
    RDNSequence }
  RDNSequence ::= SEQUENCE OF
RelativeDistinguishedName
  RelativeDistinguishedName ::=
    SET OF AttributeTypeAndValue
  AttributeTypeAndValue ::= SEQUENCE {
    type     AttributeType,
    value    AttributeValue }
  AttributeType ::= OBJECT IDENTIFIER
  AttributeValue ::= ANY DEFINED BY AttributeType

  Validity ::= SEQUENCE {
      notBefore      Time,  -- 证书有效期起始时间
      notAfter       Time  -- 证书有效期终止时间
      }
  Time ::= CHOICE {
      utcTime        UTCTime,
      generalTime    GeneralizedTime }
```

```
UniqueIdentifier ::= BIT STRING
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm           AlgorithmIdentifier, -- 公钥算法
    subjectPublicKey    BIT STRING           -- 公钥值
    }
subjectPublicKey:
RSAPublicKey ::= SEQUENCE { -- RSA 算法时的公钥值
     modulus            INTEGER, -- n
     publicExponent     INTEGER -- e -- }

Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
    extnID      OBJECT IDENTIFIER,
    critical    BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING }
```

2.数据结构

本次程序主要调用了 java 的 security 库来解决，其中读文件用到了 FileInputStream，存文件证书用到了 Certificate，转换为 x509 证书用到了 X509Certificate，然后就是存公钥用到的 PublicKey。

3．Java 源代码分析

本次没有选择用 c++写，因为 c++写起来很繁琐，读取出来后无法处理，所以在百度之后选择了 java。因为 java 中已经有了证书的一些方法，也有 x509 封装好的一些函数可以直接调用，很方便。

首先是一些库的引用

```
import java.security.*;
import java.io.*;
import java.security.cert.*;
import java.security.cert.Certificate;
```

然后是一些变量的声明，比如读文件，获取证书等等

```
CertificateFactory factory ;
try {
    FileInputStream filepath = new FileInputStream("test.cer") ;
    factory = CertificateFactory.getInstance("X.509") ;
    Certificate certificate = factory.generateCertificate(filepath);
    X509Certificate x509 = (X509Certificate)certificate ;
    filepath.close();
```

再是调用封装好的函数来输出证书的各个部分内容

```
System.out.println("版本号:" + x509.getVersion()) ;
System.out.println("序列号:" + x509.getSerialNumber()) ;
System.out.println("使用的签名算法："+ x509.getSigAlgName()) ;
System.out.println("颁发者:"+ x509.getIssuerDN()) ;
System.out.println("有效起始日期："+ x509.getNotBefore()) ;
System.out.println("有效终止日期："+ x509.getNotAfter()) ;
System.out.println("主体名称："+ x509.getSubjectDN()) ;
System.out.println("证书签名："+ x509.getSignature()) ;
PublicKey key = x509.getPublicKey() ;
byte [] Key_encode = key.getEncoded() ;
System.out.print("公钥:") ;
for(int i = 0 ; i < Key_encode.length ; i++) {
    System.out.print(Key_encode[i]+",");
}
```

最后是一个异常处理，因为在 io 的时候需要进行 io 的异常处理，

java 中自带的没有这个异常处理

```
public static void main(String args[]) throws IOException {
    CertificateFactory factory ;
    try {

    }
    catch (CertificateException e) {
        e.printStackTrace();
    }
```

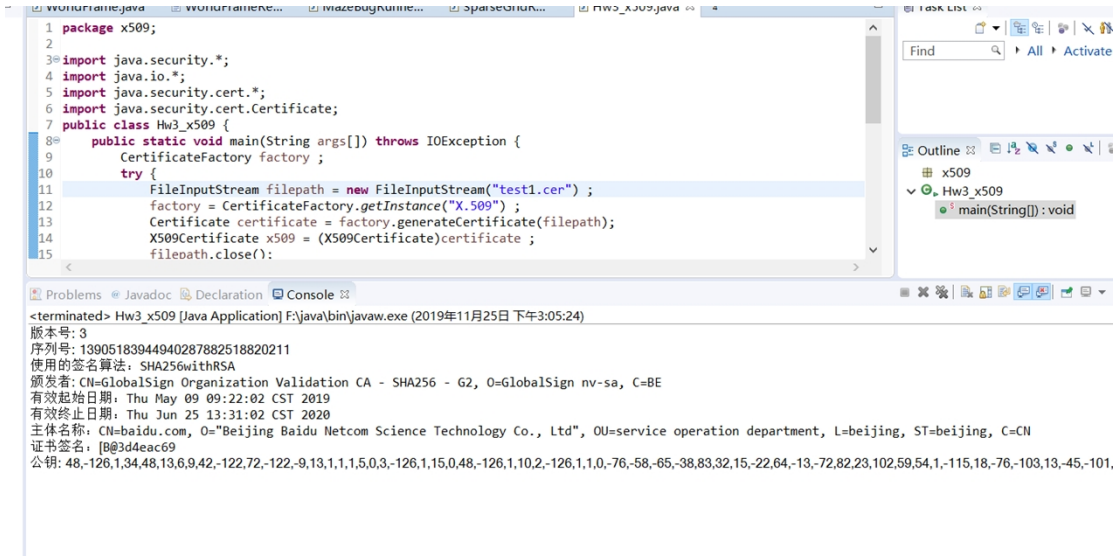
整个程序大概就这样，主要调用了 java 的 security 库以及其中的 Certificate 部分来解决。

4.运行截图

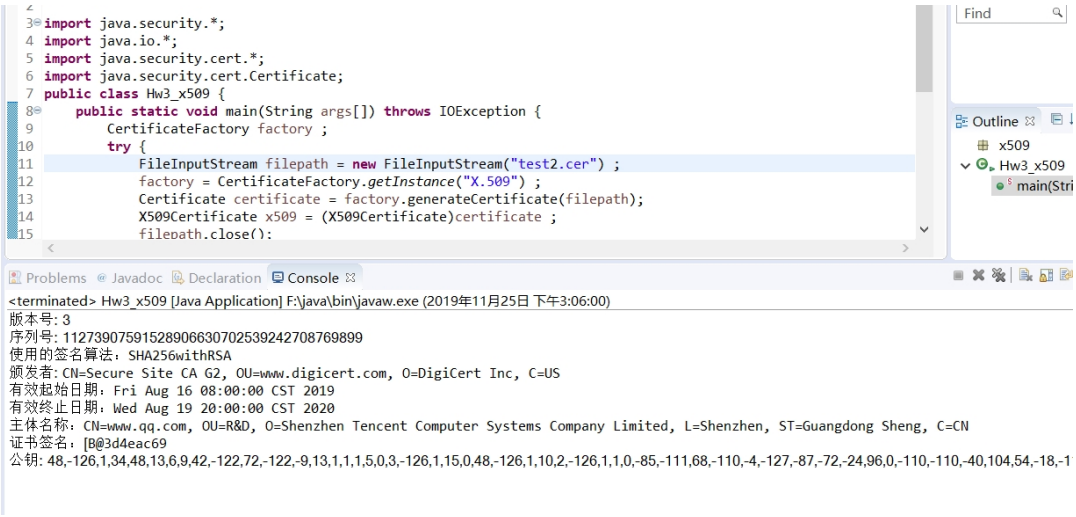
代码包里有 3 个测试的证书分别为 test.cer，test1.cer，test2.cer

第一个是 csdn 网站的 x509 证书

然后是百度的 x509 证书



最后是腾讯的 x509 证书



都可以一一读取出来，打印出证书的内容。