配置适用于正式使用环境下的 Tomcat Web 服务器双向 SSL 认证

关于如何使用 Tomcat 服务器实现双向 SSL 认证的文章很早就有了，比较实用的文章可以看看 IBM developerWorks 中国网站 2002 年 5 月 配置 Tomcat 4 使用 SSL(http://www-900.ibm.com/developerWorks/cn/security/se-tcssl/index.shtml)，使用 google 还可以搜索到很多类似的文章.

但 是现在看来，这些文章也有不足之处，一是只说明了在 Tomcat 4 服务器上的配置方法，与现在普遍使用的 Tomcat 5 系列的配置方法有一定的差异，其次，这些文章中的方法大多是作为一种实验性的操作介绍，如果用于实际 Web 服务器的产生证书环境，还是会存在一些问题的:

- 生成的 CA 私钥(ca-key.pem)以及自签名根证书(ca-cert.pem)均为没有加密的明文，由于一旦在服务器发布，以及客户端证书发布之后，根证书是不能随便修改的，那么在保管没有加密的 CA 私钥以及自签名根证书时就会存在安全性问题(如果别人得到你的 CA 根证书，他就可以替你生成客户端证书了);
- 生成的服务器端证书(server_keystore)不包括信任的 CA 根证书，信任的 CA 根证书被导入到 JSSE 的默认位置，如果进行证书操作的计算机和真正运行 Web 应用的不是同一台计算机，那么在安装服务器端证书的时候，还需要在服务器上将 CA 根证书导入 JSSE 默认位置;
- 生成客户端证书的过程比较烦琐，不方便批量进行客户端证书的生成;
- 没有统一的配置文件，生成证书的过程中需要执行相当多的命令，可重复性差，出错的可能性比较大.

针对以上的问题，在参考"配置 Tomcat 4 使用 SSL"这篇文章的基础上，整理出由 4 个批处理命令和一个配置文件组成的证书工具包，这个工具包考虑到在实际 Web 服务器环境下产生证书的需求，克服了上面提到的那些问题.

用到的软件包

- Tomcat 5.0.x
- JSSE 中的 keytool 工具，在 JDK 1.4 中已经自带了这个工具，因此，只需要安装 JDK1.4.x 即可
- openssl(openssl 已经被包含在这个工具包里面了，不需要另外安装)
- cygwin 的 sed 和 echo 命令(这些命令文件已经包含在工具包里面，不需要另外安装,)

配置及主要命令介绍

- **PATH** 环境变量
  - 安装 JDK 之后，需要将环境变量 JAVA_HOME 设置为 JDK 的安装目录，同时在 PATH 中加上 %JAVA_HOME%\bin.
- 工具包结构
  - **.bin**: 存放 openssl 以及 cygwin 的 sed 和 echo 命令的可执行执行文件;
  - **.etc**: 存放系统的配置文件，其中最主要的配置文件是 .etc/config.cmd;
  - **dist**: 这个目录存放产生的各种证书;
  - **work**: 产生证书时使用的临时文件的存放目录，为了安全起见，在每次证书相关操作结束后，建议清空这个目录;
  - **step0,1,2,3** 四个批处理命令：这四个命令分别用于证书各个方面的操作.
- **.etc/config.cmd**
  - 这个文件是系统的配置文件，主要配置 CA 根证书/服务器证书/客户端证书 的相关信息，例如证书的 distinguished name 信息等等，在生成证书之前，需要首先根据实际情况更新这个配置文件;
- **step0-ca-pfx.bat**
  - 该命令生成 CA 私钥以及自签名根证书，最后得到 PKCS12 格式的 CA 根证书 (PKCS12 格式的 CA 根证书受密码保护，因此有比较好的安全性);
  - 生成的 PKCS12 格式的 CA 根证书保存在 dist\ca-cert 目录下，在正式使用的系统中，这个证书文件(*.pfx)需要妥善保存，因为以后的服务器证书和客户端证书都需要依赖这个证书，尤其是客户端证书，如果丢失了 CA 根证书，就无法发布新的客户端证书了，而重新生成 CA 根证书，则需要重新生成服务器证书和客户端证书，在客户端用户较多的情况下，重新发布所有的客户端证书是有相当大的工作量;
  - 在执行这个命令的过程中，会提示输入证书保护密码，请注意不要遗忘或者泄漏这个密码，否则根证书的安全会受到威胁.
- **step1-ca-prepare.bat**
  - 该命令用于从 PKCS12 格式的 CA 根证书中导出 CA 私钥和未加密 CA 根证书;
  - 由于安全原因，CA 根证书平时以密码保护的 PKCS12 格式文件(*.pfx)存放，那么如果需要使用 CA 根证书来发布服务器证书或者客户端证书时，首先需要执行这个命令得到 CA 根证书的未加密形式;
  - 在执行这个命令的过程中，会出现两次提示输入根证书的保护密码，如果密码不正确，这个命令将不能执行成功，也就无法进行下面两步的发布证书的操作了.
- **step2-server.bat**
  - 该命令用于生成服务器端证书(keystore 文件);
  - 注意：CA 根证书同时也会被导入到证书的 keystore 文件中，这样在将这个证书应用到 Tomcat Web 服务器上的时候就不需要将 CA 根证书导入到 JSSE 的默认位置了;
  - 这个命令必须在执行 step1-ca-prepare.bat 之后才能正常运行.
- **step3-client.bat**
  - 这个命令用于发布客户端证书;
  - 为了便于批量生成客户端证书，这个命令支持命令行参数，第 1 到 3 个参数依次为:
    - 客户端证书的名称(Common Name)
    - 客户端证书所属的组织(Organizational Unit Name)
    - 产生的 PKS12 格式客户端证书的导入密码，这个密码可以保护证书只能被知道密码的用户导入到浏览器
  - 这个命令必须在执行 step1-ca-prepare.bat 之后才能正常运行.

说明**:**当 Web 服务器开始正式运行以后，step0-ca-pfx.bat 命令是不能再次执行的，如果需要重新发布服务器证书，或者发布新的客户端证书，在执行 step2-server.bat 和 step3-client.bat 命令前，可以通过 step1-ca-prepare.bat 重新从保存的 PKCS12 格式 CA 根证书中导出 CA 私钥和未加密 CA 根证书.

Tomcat 5 服务器配置

参考配置方法如下:

- 将 "step2-server.bat" 命令产生的 dist\server 目录下的 keystore 文件(如果使用本工具包的默认配置, 这个文件叫做"ssl-test.net-tomcat.keystore")复制到 Tomcat 安装目录的 conf 目录下;
- 修改 Tomcat 安装目录的 conf 目录下的 "server.xml" 文件, 修改 <Service name="Catalina"> 包含的 "Connector" 元素, 示例如下(仅供参考):

```
<Service name="Catalina">
 <Connector URIEncoding="UTF-8"
       acceptCount="100" connectionTimeout="20000" disableUploadTimeout="true"
       port="8080" redirectPort="8443"
       maxSpareThreads="75" maxThreads="150" minSpareThreads="25">
 </Connector>
 <Connector URIEncoding="UTF-8" port="8443"
       maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
       enableLookups="false" disableUploadTimeout="true"
       acceptCount="100" debug="0" scheme="https" secure="true"
       clientAuth="true" sslProtocol="TLS"
       keystoreFile="${catalina.home}/conf/ssl-test.net-tomcat.keystore"
       keystorePass="openssl"
       truststoreFile="${catalina.home}/conf/ssl-test.net-tomcat.keystore"
       truststorePass="openssl"/>
 ......
```

启用双向 SSL 时 Web 应用程序的配置

- 要启用双向 SSL 认证, 在 Web 应用程序的 web.xml 中需要如下增加一些配置: auth-method=CLIENT-CERT 说明是"以客户端数字证书来确认用户的身份", transport-guarantee=CONFIDENTIAL 表示应用程序要求数据必须在一种"防止其他实体看到传输的内容的方式中传送".

```
<login-config>
<!-- Authorization setting for SSL -->
  <auth-method>CLIENT-CERT</auth-method>
  <realm-name>Client Cert Users-only Area</realm-name>
</login-config>
<security-constraint>
<!-- Authorization setting for SSL -->
  <web-resource-collection >
    <web-resource-name >SSL</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

- 经过以上的配置之后, 那么即使用户是通过 http 访问 Web 应用程序的, 浏览器也会自动切换到 https 方式并弹出选择客户端证书的对话框.

如何使用客户端证书进行用户验证

- 查看一些资料上提到客户端证书内容中 subject 的 CN 部分可以和 Tomcat 的 Realm 中的用户集成, 不过一直没有尝试成功;
- 在 web 应用程序中, 可以通过 java 代码从 request 对象中获得, 根据 Servlet Specifications, 使用 `request.getAttribute("javax.servlet.request.X509Certificate")` 就可以得到 https 请求的客户端证书链信息, 其中第一个元素就是客户端证书, 相应的示例代码如下:

```
  String certSubject = null;
    X509Certificate[] certChain=
        (X509Certificate[])request.getAttribute("javax.servlet.requ
est.X509Certificate");
    int len=certChain.length;
    if (len>0){
        X509Certificate cert = (X509Certificate)certChain[0];
        Principal pSubject = cert.getSubjectDN();
        certSubject = pSubject.getName();
    }
```

- 客户端证书的 subject 是类似 `CN=client, OU=web, O=ssl-test.net, L=your_locality, ST=your_province, C=CN` 这样的字符串, 其中 CN=... 就是客户端证书的用户名称, Web 应用程序可以通过这个字段来验证 https 请求对应的用户身份了.

END