用SSL构建安全的SocketJava认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E7_94_A8SSL_E6_9E_84_E5_c104_644562.htm SSL(安全套接层)是 Netscape公司在1994年开发的，最初用于WEB浏览器，为浏览器与服务器间的数据传递提供安全保障，提供了加密、来源认证和数据完整性的功能。现在SSL3.0得到了普遍的使用，它的改进版TLS(传输层安全)已经成为互联网标准。SSL本身和TCP套接字连接是很相似的，在协议栈中，SSL可以被简单的看作是安全的TCP连接，但是某些TCP连接的特性它是不支持的，比如带外数据(out-of-bound)。 在构建基于 Socket的C/S程序时，通过添加对SSL的支持来保障数据安全和完整是不错的方法。完善的Java为我们提供了简单的实现方法：JSSE(Java 安全套接字扩展)。JSSE是一个纯Java实现的SSL和TLS协议框架，抽象了SSL和TLS复杂的算法，使安全问题变得简单。JSSE已经成为 J2SE1.4版本中的标准组件，支持SSL 3.0和TLS 1.0。我们将通过一个具体的例子演示JSSE的一些基本应用。例子中的服务器端将打开一个SSL Socket，只有持有指定证书的客户端可以与它连接，所有的数据传递都是加密的。 构造一个SSLSocket是非常简单的： SSLServerSocketFactory factory=(SSLServerSocketFactory)SSLServerSocketFactory.getDefault(). SSLServerSocket server = (SSLServerSocket) factory.createServerSocket(portNumber). SSLSocket socket = (SSLSocket). 但是执行这样的程序会产生一个异常，报告找不到可信任的证书。SSLSocket和普通的Socket是不一样的，它需要一个证书来进行安全认证。 一、 证书 生成一个CA证书

，在命令行下执行： keytool genkey keystore SSLKey keyalg rsa alias SSL 黑体部分是用户可以自己指定的参数，第一个参数是要生成的证书的名字，第二个参数是证书的别名。rsa指明了我们使用的加密方法。 系统会要求输入证书发放者的信息，逐项输入即可。 系统生成的文件命将会和证书名相同。证书可以提交给权威CA认证组织审核，如果通过审核，组织会提供信任担保，向客户担保你的连接是安全的。当然这不是必须的。在我们的例子中会把证书直接打包到客户端程序中，保证客户端是授权用户，避免伪造客户，所以不需要提交审核。 二、 服务器端 现在可以编写服务器端的代码，与普通的Socket代码不同，我们需要在程序中导入证书，并使用该证书构造SSLSocket。需要的说明的是： ●KeyStore ks=KeyStore.getInstance("JKS"). 访问Java密钥库，JKS是keytool创建的Java密钥库，保存密钥。 ● KeyManagerFactory kmf=KeyManagerFactory.getInstance("SunX509"). 创建用于管理JKS密钥库的X.509密钥管理器。 ● SSLContext sslContext=SSLContext.getInstance("SSLv3"). 构造SSL环境，指定SSL版本为3.0，也可以使用TLSv1，但是SSLv3更加常用。 ●sslContext.init(kmf.getKeyManagers(),null,null). 初始化SSL环境。第二个参数是告诉JSSE使用的可信任证书的来源，设置为null是从javax.net.ssl.trustStore中获得证书。第三个参数是JSSE生成的随机数，这个参数将影响系统的安全性，设置为null是个好选择，可以保证JSSE的安全性。 完整代码如下： /* *SSL Socket的服务器端 */ package org.ec107.ssl. import java.net.*. import javax.net.ssl.*. import java.io.*. import java.security.*. public class SSLServer { static int port=8266. //系统

将要监听的端口号,82.6.6是偶以前女朋友的生日^_^ static SSLServerSocket server. /* *构造函数 */ public SSLServer() { } /* *@param port 监听的端口号 *@return 返回一个SSLServerSocket对象 */ private static SSLServerSocket getServerSocket(int thePort) { SSLServerSocket s=null. try { String key="SSLKey". //要使用的证书名 char keyStorePass[]="12345678".toCharArray(). //证书密码 char keyPassword[]="12345678".toCharArray(). //证书别称所使用的主要密码 KeyStore ks=KeyStore.getInstance("JKS"). //创建JKS密钥库 ks.load(new FileInputStream(key),keyStorePass). //创建管理JKS密钥库的X.509密钥管理器 KeyManagerFactory kmf=KeyManagerFactory.getInstance("SunX509"). kmf.init(ks,keyPassword). SSLContext sslContext=SSLContext.getInstance("SSLv3"). sslContext.init(kmf.getKeyManagers(),null,null). //根据上面配置的SSL上下文来产生SSLServerSocketFactory,与通常的产生方法不同 SSLServerSocketFactory factory=sslContext.getServerSocketFactory(). s=(SSLServerSocket)factory.createServerSocket(thePort). }catch(Exception e) { System.out.println(e). } return(s). } public static void main(String args[]) { try { server=getServerSocket(port). System.out.println("在 ” port ” 端口等待连接..."). while(true) { SSLSocket socket=(SSLSocket)server.accept(). 