# 目的

全站支持https访问

让多个tomcat的菜单连接挂在一个tomcat菜单下同样的支持https的访问；

注：由于SSL和CAS集成只能使用域名的方式访问

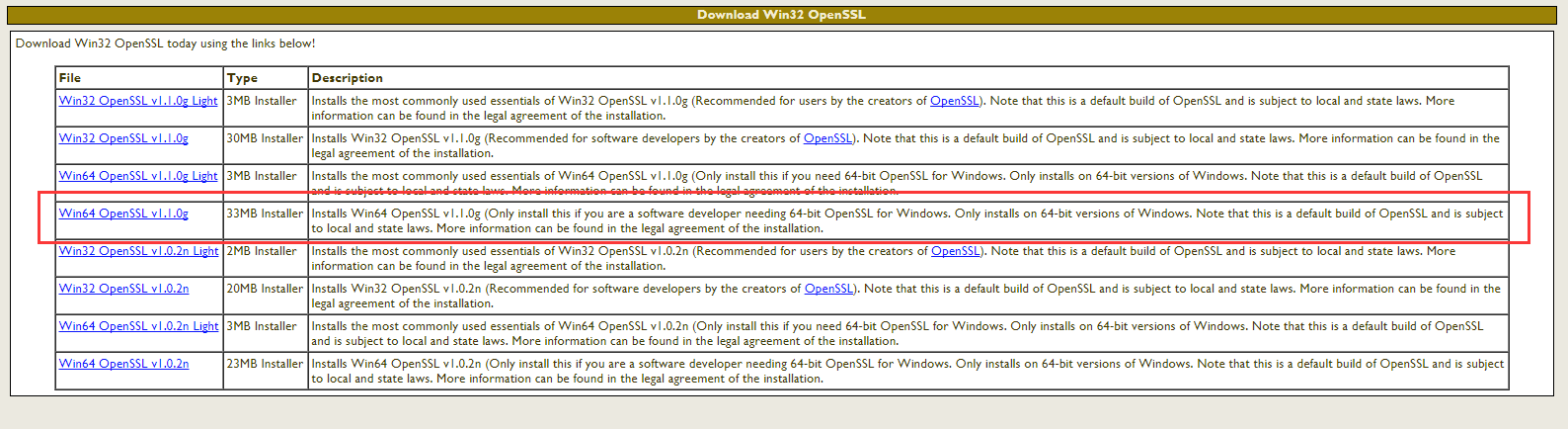
1. 使用工具

Nginx、tomcat 、openSSl。

1. 配置

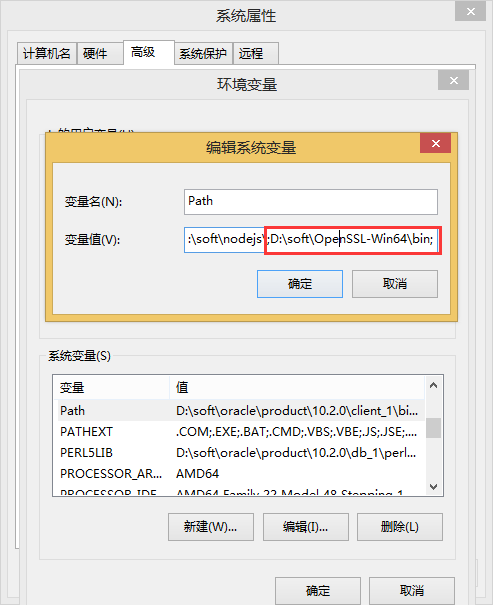
3.1安装openssl

1.下载地址：<http://slproweb.com/products/Win32OpenSSL.html>

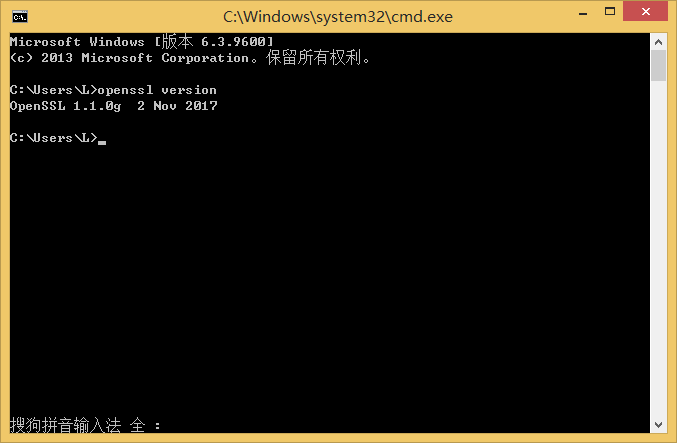


安装无特别选项，直接下一步，下一步。

2.安装完毕后，配置openssl的环境变量；



3.配置完成后使用 windows+R快捷键打开cmd窗口。使用openssl version命令查看是否安装成功；



3.2生成数字证书

在生成证书之前先创建一个hosts 主机域名映射，将在后面的数字证书创建中使用；

Hosts文件地址windows系统一般在：C:\Windows\System32\drivers\etc目录下，

打开hosts文件创建证书域名映射：



1. 为应用证书/中级证书生成私钥文件

|  |
| --- |
| openssl genrsa -des3 -out server.key 1024 |

2. 根据私钥文件，为应用证书/中级证书生成 csr 文件（证书请求文件）

|  |
| --- |
| 命令：openssl req -new -key server.key -out server.csr   关于证书请求文件中的DN字段 运行中会提示输入一些 Distinguished Name fields，即证书的识别名信息字段，简称为DN字段，如下  **DN字段名缩写说明填写要求**  Country NameC 证书持有者所在国家要求填写国家代码，用2个字母表示  State or Province NameST证书持有者所在州或省份填写全称，可省略不填  Locality NameL 证书持有者所在城市可省略不填  Organization NameO证书持有者所属组织或公司最好还是填一下  Organizational Unit  NameOU证书持有者所属部门可省略不填  Common NameCN 证书持有者的通用名必填。 对于非应用证书，它应该在一定程度上具有惟一性； 对于应用证书，一般填写服务器域名或通配符样式的域名。  Email Address 证书持有者的通信邮箱可省略不填 |

3.复制key文件

|  |
| --- |
| copy server.key server.key.org |

4. 为CA的私钥文件去除保护口令

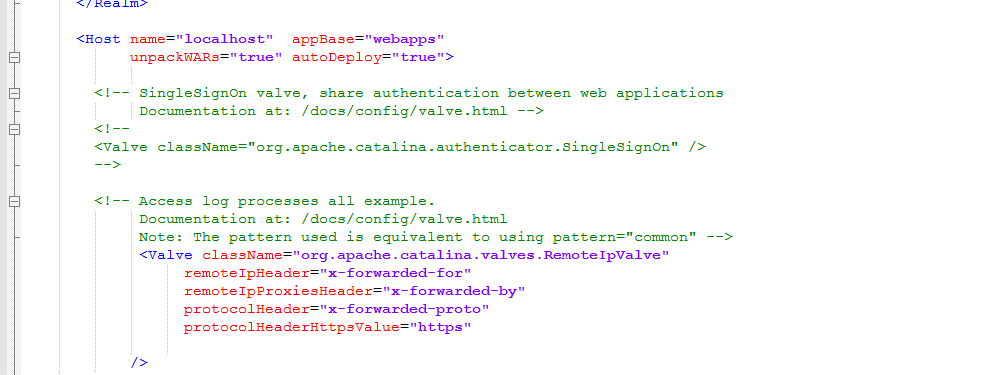
|  |
| --- |
| openssl rsa -in server.key.org -out server.key |

5. 使用CA的公私钥文件给 csr 文件签名，生成中级证书，有效期10年

|  |
| --- |
| openssl x509 -req -days 3650 -in server.csr -signkey server.key -out server.crt |

3.3配置tomcat 端

整个访问流程是 浏览器通过https 访问nginx，在nginx代理使用http访问tomcat,所以需要给tomcat 加上https 支持。在tomcat的server.xml中增加如何配置；



|  |
| --- |
| <Valve className="org.apache.catalina.valves.RemoteIpValve"  remoteIpHeader="x-forwarded-for"  remoteIpProxiesHeader="x-forwarded-by"  protocolHeader="x-forwarded-proto"  protocolHeaderHttpsValue="https"    /> |

当nginx监听的非80端口和443端口需要在Value 中加上端口号。

3.4配置nginx

整个nginx的配置

|  |
| --- |
| #user nobody;  worker\_processes 1;#配置服务cpu的个数  #error\_log logs/error.log;  #error\_log logs/error.log notice;  #error\_log logs/error.log info;  #pid logs/nginx.pid;  events {  worker\_connections 1024;#配置连接数，根据你的cpu的个数计算。  }  http {  include mime.types;  default\_type application/octet-stream;  #log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '  # '$status $body\_bytes\_sent "$http\_referer" '  # '"$http\_user\_agent" "$http\_x\_forwarded\_for"';  #access\_log logs/access.log main;  sendfile on;  #tcp\_nopush on;  #keepalive\_timeout 0;  keepalive\_timeout 65;  #gzip on;  # another virtual host using mix of IP-, name-, and port-based configuration    server {  listen 80;  server\_name localhost;  rewrite ^(.\*)$ https://$host$1 permanent; #用户习惯用http访问，加上80让它自动跳到443端口  }  # HTTPS server  #nginx 配置了 https tomcat 不同配置  server {  listen 443 ssl; #监听https 443端口  server\_name www.fjdc.com; #域名访问地址    ssl\_certificate F:/test/server.crt; #配置数字签名  ssl\_certificate\_key F:/test/server.key; #配置数字签名    location /AM/ { #项目需要部署在webapp AM为你webapp的项目名称    proxy\_pass http://192.168.20.175:8080; #代理访问地址    proxy\_pass\_request\_headers on;  proxy\_set\_header Host $host:$server\_port;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_connect\_timeout 240;  proxy\_send\_timeout 240;  proxy\_read\_timeout 240;  }    location /alk/ {  proxy\_pass http://192.168.20.175:9090;  proxy\_redirect off;  proxy\_pass\_request\_headers on;  proxy\_set\_header Host $host:$server\_port;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_connect\_timeout 240;  proxy\_send\_timeout 240;  proxy\_read\_timeout 240;  }    location /cas/ { #单点的代理地址  proxy\_pass http://192.168.0.151:8888;    proxy\_redirect off;  proxy\_pass\_request\_headers on;  proxy\_set\_header Host $host:$server\_port;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_connect\_timeout 240;  proxy\_send\_timeout 240;  proxy\_read\_timeout 240;  }  }    } |

3.5单点集成配置

因为单点做了代理所以需要修改对应的单点配置，让单点的server和client 都走我们的代理地址；

|  |
| --- |
| <!--用于单点退出，该过滤器用于实现单点登出功能 -->  <listener>  <listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-class>  </listener>  <!--该过滤器用于实现单点登出功能。 -->  <filter>  <filter-name>CASSingle Sign OutFilter</filter-name>  <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>CASSingle Sign OutFilter</filter-name>  <!-- url-pattern 不需要修改 -->  <url-pattern>/\*</url-pattern>  </filter-mapping>    <filter>  <filter-name>CASFilter</filter-name>  <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>  <init-param>  <param-name>casServerLoginUrl</param-name>  <!-- SSO SERVER的IP+端口+上下文+登录地址 -->  <param-value>https://www.fjdc.com:443/cas</param-value>  </init-param>  <init-param>  <param-name>serverName</param-name>  <!--客户端的IP+端口-->  <param-value>https:// www.fjdc.com:443</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CASFilter</filter-name>  <!-- url-pattern 不需要修改 -->  <url-pattern>/\*</url-pattern>  </filter-mapping>    <!--该过滤器负责对Ticket的校验工作，必须启用它 -->  <filter>  <filter-name>CASValidationFilter</filter-name>  <filter-class> org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter  </filter-class>  <init-param>  <param-name>casServerUrlPrefix</param-name>  <!-- SSO SERVER的IP+端口+上下文 -->  <param-value>https://www.fjdc.com:443/cas</param-value>  </init-param>  <init-param>  <param-name>serverName</param-name>  <!--客户端的IP+端口 -->  <param-value>https://www.fjdc.com:443</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CASValidationFilter</filter-name>  <!-- url-pattern 不需要修改 -->  <url-pattern>/\*</url-pattern>  </filter-mapping>    <!-- 该过滤器负责实现HttpServletRequest请求的包裹， 比如允许开发者通过HttpServletRequest的getRemoteUser()方法获得SSO登录用户的登录名，可选配置。 -->  <filter>  <filter-name>CASHttpServletRequest WrapperFilter</filter-name>  <filter-class>  org.jasig.cas.client.util.HttpServletRequestWrapperFilter  </filter-class>  </filter>  <filter-mapping>  <filter-name>CASHttpServletRequest WrapperFilter</filter-name>  <url-pattern>\*.vm</url-pattern>  </filter-mapping>    <!-- 该过滤器使得开发者可以通过org.jasig.cas.client.util.AssertionHolder来获取用户的登录名。 比如AssertionHolder.getAssertion().getPrincipal().getName()。 -->  <filter>  <filter-name>CASAssertion Thread LocalFilter</filter-name>  <filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>CASAssertion Thread LocalFilter</filter-name>  <url-pattern>\*.vm</url-pattern>  </filter-mapping>  <!-- SSO 登录后，业务系统获取登录用户信息 SSO-Client Filter-->  <filter>  <filter-name>SSOClientFilter</filter-name>  <filter-class>ths.jdp.project.filter.SSOClientFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>SSOClientFilter</filter-name>  <url-pattern>\*.vm</url-pattern>  </filter-mapping>  <!-- ========================单点登录结束 ======================== --> |

3.6配置证书加入jre的security 目录

此配置是解决cas单点登录https访问的时候出现的证书错误的问题；

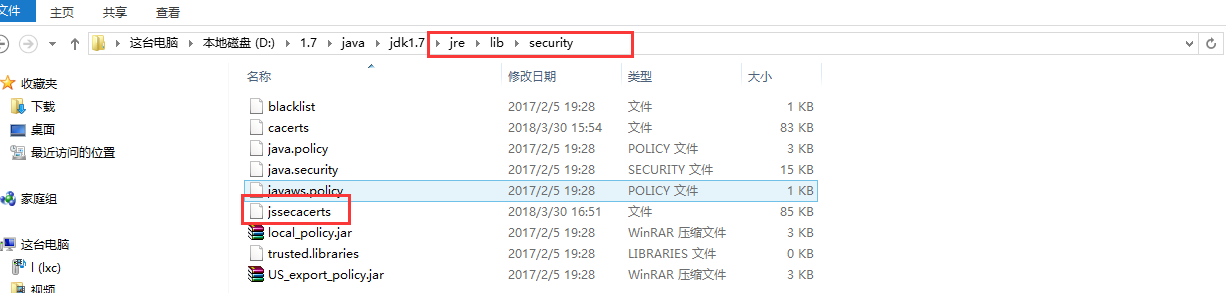
1.将此类编译

|  |
| --- |
| package cn.ths.liuxc.check;  /\*  \* Copyright 2006 Sun Microsystems, Inc. All Rights Reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* - Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* - Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the distribution.  \*  \* - Neither the name of Sun Microsystems nor the names of its  \* contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS  \* IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,  \* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  \* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  \* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  \* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  \* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  \* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  \* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  \* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  \* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*/    import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.io.OutputStream;  import java.security.KeyStore;  import java.security.MessageDigest;  import java.security.cert.CertificateException;  import java.security.cert.X509Certificate;    import javax.net.ssl.SSLContext;  import javax.net.ssl.SSLException;  import javax.net.ssl.SSLSocket;  import javax.net.ssl.SSLSocketFactory;  import javax.net.ssl.TrustManager;  import javax.net.ssl.TrustManagerFactory;  import javax.net.ssl.X509TrustManager;    public class InstallCert {    public static void main(String[] args) throws Exception {  args=new String[]{"www.gavin.com"};  String host;  int port;  char[] passphrase;  if ((args.length == 1) || (args.length == 2)) {  String[] c = args[0].split(":");  host = c[0];  port = (c.length == 1) ? 443 : Integer.parseInt(c[1]);  String p = (args.length == 1) ? "changeit" : args[1];  passphrase = p.toCharArray();  } else {  System.out  .println("Usage: java InstallCert <host>[:port] [passphrase]");  return;  }    File file = new File("jssecacerts");  if (file.isFile() == false) {  char SEP = File.separatorChar;  File dir = new File(System.getProperty("java.home") + SEP + "lib"  + SEP + "security");  file = new File(dir, "jssecacerts");  if (file.isFile() == false) {  file = new File(dir, "cacerts");  }  }  System.out.println("Loading KeyStore " + file + "...");  InputStream in = new FileInputStream(file);  KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());  ks.load(in, passphrase);  in.close();    SSLContext context = SSLContext.getInstance("TLS");  TrustManagerFactory tmf = TrustManagerFactory  .getInstance(TrustManagerFactory.getDefaultAlgorithm());  tmf.init(ks);  X509TrustManager defaultTrustManager = (X509TrustManager) tmf  .getTrustManagers()[0];  SavingTrustManager tm = new SavingTrustManager(defaultTrustManager);  context.init(null, new TrustManager[] { tm }, null);  SSLSocketFactory factory = context.getSocketFactory();    System.out  .println("Opening connection to " + host + ":" + port + "...");  SSLSocket socket = (SSLSocket) factory.createSocket(host, port);  socket.setSoTimeout(10000);  try {  System.out.println("Starting SSL handshake...");  socket.startHandshake();  socket.close();  System.out.println();  System.out.println("No errors, certificate is already trusted");  } catch (SSLException e) {  System.out.println();  e.printStackTrace(System.out);  }    X509Certificate[] chain = tm.chain;  if (chain == null) {  System.out.println("Could not obtain server certificate chain");  return;  }    BufferedReader reader = new BufferedReader(new InputStreamReader(  System.in));    System.out.println();  System.out.println("Server sent " + chain.length + " certificate(s):");  System.out.println();  MessageDigest sha1 = MessageDigest.getInstance("SHA1");  MessageDigest md5 = MessageDigest.getInstance("MD5");  for (int i = 0; i < chain.length; i++) {  X509Certificate cert = chain[i];  System.out.println(" " + (i + 1) + " Subject "  + cert.getSubjectDN());  System.out.println(" Issuer " + cert.getIssuerDN());  sha1.update(cert.getEncoded());  System.out.println(" sha1 " + toHexString(sha1.digest()));  md5.update(cert.getEncoded());  System.out.println(" md5 " + toHexString(md5.digest()));  System.out.println();  }    System.out  .println("Enter certificate to add to trusted keystore or 'q' to quit: [1]");  String line = reader.readLine().trim();  int k;  try {  k = (line.length() == 0) ? 0 : Integer.parseInt(line) - 1;  } catch (NumberFormatException e) {  System.out.println("KeyStore not changed");  return;  }    X509Certificate cert = chain[k];  String alias = host + "-" + (k + 1);  ks.setCertificateEntry(alias, cert);    OutputStream out = new FileOutputStream("jssecacerts");  ks.store(out, passphrase);  out.close();    System.out.println();  System.out.println(cert);  System.out.println();  System.out  .println("Added certificate to keystore 'jssecacerts' using alias '"  + alias + "'");  }    private static final char[] HEXDIGITS = "0123456789abcdef".toCharArray();    private static String toHexString(byte[] bytes) {  StringBuilder sb = new StringBuilder(bytes.length \* 3);  for (int b : bytes) {  b &= 0xff;  sb.append(HEXDIGITS[b >> 4]);  sb.append(HEXDIGITS[b & 15]);  sb.append(' ');  }  return sb.toString();  }    private static class SavingTrustManager implements X509TrustManager {    private final X509TrustManager tm;  private X509Certificate[] chain;    SavingTrustManager(X509TrustManager tm) {  this.tm = tm;  }    public X509Certificate[] getAcceptedIssuers() {  throw new UnsupportedOperationException();  }    public void checkClientTrusted(X509Certificate[] chain, String authType)  throws CertificateException {  throw new UnsupportedOperationException();  }    public void checkServerTrusted(X509Certificate[] chain, String authType)  throws CertificateException {  this.chain = chain;  tm.checkServerTrusted(chain, authType);  }  }    } |

2.执行改程序；java [www.fjdc.com，输入1](http://www.fjdc.com，输入1)、

3.执行程序完成之后，会在相对路径之下生产一个“jssecacerts” 文件。

4.将这个文件拷贝到对应的jdk的jre下的lib下security目录下；



以上配置完成之后，就可以直接通过我们的<https://www.fjdc.com/AM>的方式访问项目了。