SSL/TLS 配置

Quick Start

下列說明將使用變量名 $CATALINA\_BASE 來表示多數相對路徑所基於的基本目錄。如果沒有為 Tomcat 多個實例設置 CATALINA\_BASE 目錄，則 $CATALINA\_BASE 就會設定為 $CATALINA\_HOME 的值，也就是你安裝 Tomcat 的目錄。

在 Tomcat 中安裝並配置 SSL/TLS 支持，只需遵循下列幾步即可。詳細信息可參看文檔後續介紹。

1. 創建一個 keystore 文件保存服務器的私有密鑰和自簽名證書：  
   Windows：  
   "%JAVA\_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA  
   UNIX：  
   $JAVA\_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA

指定密碼為「changeit」。

1. 取消對 $CATALINA\_BASE/conf/server.xml 中 「SSL HTTP/1.1 Connector」 一項的註釋狀態。按照下文中[配置](http://wiki.jikexueyuan.com/project/tomcat/ssl-tls.html#configuration)一節所描述的方式進行修改。

SSL/TLS 簡介

安全傳輸層協議（TLS）與其前輩加密套接字（SSL）都是用於保證 Web 瀏覽器與 Web 服務器通過安全連接進行通信的技術。利用這些技術，我們所要傳送的數據會在一端進行加密，傳輸到另一端後再進行解密（在處理數據之前）。這是一種雙向的操作，服務器和瀏覽器都能在發送數據前對它們進行加密處理。

SSL/TLS 協議的另一個重要方面是認證。當我們初始通過安全連接與 Web 服務器進行通信時，服務器將提供給 Web 瀏覽器一組「證書」形式的憑證，用來證明站點的歸屬方以及站點的具體聲明。某些情況下，服務器也會請求瀏覽器出示證書，來證明作為操作者的「你」所宣稱的身份是否屬實。這種證書叫做「客戶端證書」，但事實上它更多地用於 B2B（企業對企業電子商務）的交易中，而並非針對個人用戶。大多數啟用了 SSL 協議的 Web 服務器都不需要客戶端認證。

SSL/TLS 與 Tomcat

一定要注意的是，通常只有當 Tomcat 是獨立運行的 Web 服務器時，才有必要去配置 Tomcat 以便利用加密套接字。具體細節可參看 [Security Considerations Document](http://tomcat.apache.org/tomcat-8.0-doc/security-howto.html)。當 Tomcat 以 Servlet/JSP 容器的形式在其他 Web 服務器（比如 Apache 或 Microsoft IIS）背後運行時，通常需要配置的是主 Web 服務器，用主服務器來處理與用戶的 SSL 連接。主服務器一般會利用所有的 SSL 相關功能，將任何只有 Tomcat 才能處理的請求進行解密後再傳給 Tomcat。同樣，Tomcat 會返回明文形式的響應，這些響應在被傳輸到用戶瀏覽器之前會被主服務器進行加密處理。在這種情境下，Tomcat 知道主服務器與客戶端的所有通信都是通過安全連接進行的（因為應用要求），但 Tomcat 自身無法參與到加密與解密的過程中。

證書

為了實現 SSL，Web 服務器必須對每一個接受安全連接的外部接口（IP 地址）配備相應的證書。這種設計方式的理論在於，服務器必須提供某種可信的保證（尤其對於接收敏感信息而言），保證它的擁有者是你所認為的角色。限於本章篇幅，不再贅述關於證書的詳細解釋，只需要把它認為成是一種 IP 地址的「數字駕照」即可。它聲明了與站點相關的組織，以及一些關於站點擁有者或管理者的基本聯繫信息。

這種「數字駕照」的持有者對其進行了加密簽名，從而使得它極難偽造。對於參與電子商務的站點或者其他一些身份驗證顯得非常重要的商業交易來說，證書通常都是從一些比較權威公正的 CA （ Certificate Authority，認證機構）購買的，比較知名的 CA 有 VeriSign 、Thawte 等。這些證書可經電子驗證。實際上，CA 會保證所頒發證書的真實性，所以你完全可以放心。

不過，在很多情況下，驗證並不是問題的關鍵。管理員可能只想保證服務器所傳輸與接收的數據是秘密的，不會被某些人通過連接來竊取。幸運的是，Java 提供了一個簡單的命令行工具：keytool。它能輕鬆創建一個「自簽名」的證書，這種自簽名證書是由用戶生成的一種證書，未經任何知名 CA 給予官方保證，因此它的真實性也無法確定。再說一次，是否認證，完全根據你的需求。

運行 SSL 通常需要注意的一些內容

當用戶首次訪問你站點上的安全頁面時，頁面通常會提供給他一個對話框，包含證書相關細節（比如組織及聯繫方式等），並且詢問他是否願意承認該證書為有效證書，然後再進行下一步的事務。一些瀏覽器可能會提供一個選項，允許永遠承認給出的證書的有效性，這樣就不會在用戶每次訪問站點時打擾他們了。但有些瀏覽器不會提供這種選項。一旦用戶承認了證書的有效性，那麼在整個的瀏覽器會話期間，證書都被認為是有效的。

雖然 SSL 協議的意圖是儘可能有助於提供安全且高效的連接，但從性能角度來考慮，加密與解密是非茶館耗費計算資源的，因此將整個 Web 應用都運行在 SSL 協議下是完全沒有必要的，開發者需要挑選需要安全連接的頁面。對於一個相當繁忙的網站來說，通常只會在特定頁面上使用 SSL 協議，也就是可能交換敏感信息的頁面，比如：登錄頁面、個人信息頁面、購物車結賬頁面（可能會輸入信用卡信息），等等。應用中的任何一個頁面都可以通過加密套接字來請求訪問，只需將頁面地址的前綴 http: 換成 https: 即可。絕對**需要**安全連接的頁面應該檢查該頁面請求所關聯的協議類型，如果發現沒有指定 https:，則採取適當行為。

最後，在安全連接上使用基於名稱的虛擬主機可能會造成一定的問題。這正是 SSL 協議的侷限。SSL 握手過程，即客戶端瀏覽器接收服務器證書，必須發生在 HTTP 請求被訪問前。因此包含虛擬主機名稱的請求信息不能先於認證而確定，也不可能為單個 IP 地址指定多個證書。如果單個 IP 地址的所有虛擬主機都需要利用同樣證書來認證的話，那麼多個虛擬主機不應該干涉服務器通常的 SSL 操作。但是要注意一點，多數客戶端瀏覽器會將服務器域名與證書中的多個域名（如果存在的話，）進行對比，如果域名出現不匹配，則瀏覽器會向用戶顯示警告信息。一般來說，生產環境中，通常只有使用基於地址的虛擬主機利用 SSL。

配置

1. 準備證書密鑰存儲庫

Tomcat 目前只能操作 JKS、PKCS11、PKCS12 格式的密鑰存儲庫。JKS 是 Java 標準的「Java 密鑰存儲庫」格式，是通過 keytool 命令行工具創建的。該工具包含在 JDK 中。PKCS12 格式一種互聯網標準，可以通過 OpenSSL 和 Microsoft 的 Key-Manager 來。

密鑰存儲庫中的每一項都通過一個別名字符串來標識。儘管許多密碼存儲庫實現都在處理別名時不區分大小寫，但區分大小寫的實現也是允許的。比如，PKCS11 規範需要別名是區分大小寫的。為了避免別名大小寫敏感的問題，不建議使用只有大小寫不同的別名。

為了將現有的證書導入 JKS 密碼存儲庫，請查閱關於 keytool 的相關文檔（位於 JDK 文檔包裡）。注意，OpenSSL 經常會在密碼前加上易於理解的註釋，但 keytool 並不支持這一點。所以如果證書裡的密碼數據前面有註釋的話，在利用 keytool 導入證書前，一定要清除它們。

要想把一個已有的由你自己的 CA 所簽名的證書導入使用 OpenSSL 的 PKCS12 密碼存儲庫，應該執行如下命令：

openssl pkcs12 -export -in mycert.crt -inkey mykey.key

-out mycert.p12 -name tomcat -CAfile myCA.crt

-caname root -chain

更複雜的實例，請參考 [OpenSSL 文檔](https://www.openssl.org/)的相關內容。

下面這個實例展示的是如何利用終端命令行，從零開始創建一個新的 JKS 密碼存儲庫，該密碼庫包含一個自簽名的證書。

Windows：

"%JAVA\_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA

Unix：

$JAVA\_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA

（RSA 算法應該作為首選的安全算法，這同樣也能保證與其他服務器和組件的大體的兼容性。）

該命令將在用戶的主目錄下創建一個新文件：.keystore。要想指定一個不同的位置或文件名，可以在上述的 keytool命令上添加 -keystore 參數，後跟到達 keystore 文件的完整路徑名。你還需要把這個新位置指定到 server.xml 配置文件上，見後文介紹。例如：

Windows：

"%JAVA\_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA

-keystore \path\to\my\keystore

Unix：

$JAVA\_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA

-keystore /path/to/my/keystore

執行該命令後，首先會提示你提供 keystore 的密碼。Tomcat 默認使用的密碼是 changeit（全部字母都小寫），當然你可以指定一個自定義密碼（如果你願意）。同樣，你也需要將這個自定義密碼在 server.xml 配置文件內進行指定，稍後再予以詳述。

接下來會提示關於證書的一般信息，比如組織、聯繫人名稱，等等。當用戶試圖在你的應用中訪問一個安全頁面時，該信息會顯示給用戶，所以一定要確保所提供的信息與用戶所期望看到的內容保持一致。

最後，還需要輸入**密鑰密碼**（key password），這個密碼是這一證書（而不是存儲在同一密碼存儲庫文件中的其他證書）的專有密碼。keytool 提示會告訴你，如果按下回車鍵，則自動使用密碼存儲庫 keystore 的密碼。當然，除了這個密碼，你也可以自定義自己的密碼。如果選擇自定義密碼，那麼不要忘了在 server.xml 配置文件中指定這一密碼。

如果操作全部正常，我們現在就會得到一個服務器能使用的有證書的密碼存儲庫文件。

2. 編輯 Tomcat 配置文件

Tomcat 能夠使用兩種 SSL 實現：

* JSSE 實現，它是Java 運行時（從 1.4 版本起）的一部分。
* APR 實現，默認使用 OpenSSL 引擎。

詳細的配置信息依賴於所用的實現方式。如果通過指定通用的 protocol="HTTP/1.1" 來配置連接起，那麼就會自動選擇 Tomcat 所要用到的實現方式。如果安裝使用的是 [APR](http://wiki.jikexueyuan.com/project/tomcat/%E9%9C%80%E8%A6%81%E4%B8%AD%E6%96%87%E9%A1%B5%E9%9D%A2%E9%93%BE%E6%8E%A5)（比如你安裝了 Tomcat 的原生庫），那麼它將使用 APR 的 SSL 實現，否則就將使用 Java 所提供的 JSSE 實現。

由於這兩種 SSL 實現在 SSL 支持的配置屬性上有很大差異，所以**強烈建議**不要自動選擇實現方式。選擇實現應該採取這種方式：在[連接器](http://tomcat.apache.org/tomcat-8.0-doc/config/http.html)的 protocol 屬性中，通過指定類名來確立實現方式。

定義 Java（JSSE）連接器，不管 APR 庫是否已經加載，都可以使用下列方式：

<!-- Define a HTTP/1.1 Connector on port 8443, JSSE NIO implementation -->

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"

port="8443" .../>

<!-- Define a HTTP/1.1 Connector on port 8443, JSSE NIO2 implementation -->

<Connector protocol="org.apache.coyote.http11.Http11Nio2Protocol"

port="8443" .../>

<!-- Define a HTTP/1.1 Connector on port 8443, JSSE BIO implementation -->

<Connector protocol="org.apache.coyote.http11.Http11Protocol"

port="8443" .../>

另一種方法，指定 APR 連接器（APR 庫必須可用），則使用：

<!-- Define a HTTP/1.1 Connector on port 8443, APR implementation -->

<Connector protocol="org.apache.coyote.http11.Http11AprProtocol"

port="8443" .../>

如果使用 APR，則會出現一個選項，從而可以配置另一種 OpenSSL 引擎。

<Listener className="org.apache.catalina.core.AprLifecycleListener"

SSLEngine="someengine" SSLRandomSeed="somedevice" />

默認值為：

<Listener className="org.apache.catalina.core.AprLifecycleListener"

SSLEngine="on" SSLRandomSeed="builtin" />

所以要想使用 APR 實現，一定要確保 SSLEngine 屬性值不能為 off。該屬性值默認為 on，如果指定的是其他值，它也會成為一個有效的引擎名稱。

SSLRandomSeed 屬性指定了一個熵源。生產系統需要可靠的熵源，但熵可能需要大量時間來採集，因此測試系統會使用非阻塞的熵源，比如像「/dev/urandom」，從而能夠更快地啟動 Tomcat。

最後一步是在 $CATALINA\_BASE/conf/server.xml 中配置連接器，$CATALINA\_BASE 表示的是 Tomcat 實例的基本目錄。Tomcat 安裝的默認 server.xml 文件中包含一個用於 SSL 連接器的 <Connector> 元素的範例。為了配置使用 JSSE 的 SSL 連接器，你可能需要清除註釋並按照如下的方式來編輯它。

<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->

<Connector

protocol="org.apache.coyote.http11.Http11NioProtocol"

port="8443" maxThreads="200"

scheme="https" secure="true" SSLEnabled="true"

keystoreFile="${user.home}/.keystore" keystorePass="changeit"

clientAuth="false" sslProtocol="TLS"/>

APR 連接器會使用很多不同的屬性來設置 SSL，特別是密鑰和證書。 APR 配置範例如下：

<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->

<Connector

protocol="org.apache.coyote.http11.Http11AprProtocol"

port="8443" maxThreads="200"

scheme="https" secure="true" SSLEnabled="true"

SSLCertificateFile="/usr/local/ssl/server.crt"

SSLCertificateKeyFile="/usr/local/ssl/server.pem"

SSLVerifyClient="optional" SSLProtocol="TLSv1+TLSv1.1+TLSv1.2"/>

每個屬性所用的配置信息與選項都是強制的，可查看 [HTTP 連接器](http://tomcat.apache.org/tomcat-8.0-doc/config/http.html#SSL_Support)配置參考文檔中的 SSL 支持部分。一定要確保對所使用的連接器採用正確的屬性。BIO、NIO 以及 NIO2 連接器都使用 JSSE，然而APR以及原生的連接器則使用 APR。

port屬性指的是 Tomcat 用以偵聽安全連接的 TCP/IP 端口號。你可以隨意改變它，比如改成 https 的默認端口號 443。但是在很多操作系統中，在低於 1024 的端口上運行 Tomcat 必須進行一番特殊的配置，限於篇幅，本文檔不再贅述。

**如果在這裡，你更改了端口號，那麼也應該在 非 SSL 連接器的 redirectPort 屬性值。從而使 Tomcat 能夠根據 Servlet 規範，自動對訪問帶有安全限制（指定需要 SSL）頁面的用戶進行重定向。**

配置完全部信息後，你應該像往常一樣，重新啟動 Tomcat，從而能夠利用 SSL 來訪問任何 Tomcat 所支持的 Web 應用了。比如：

https://localhost:8443/

你將看到跟往常一樣的 Tomcat 主頁面（除非你修改了 ROOT 應用）。如果出現問題，這樣做沒有任何效果，請看下面的故障排除小節。

從 CA 處安裝證書

如果想從 CA（比如 verisign.com、thawte.com、trustcenter.de）處獲取並安裝證書，請閱讀之前的內容，然後按照下列操作進行：

創建一個本地證書籤名請求（CSR）

為了從選擇的 CA 處獲取證書，必須創建一個證書籤名請求（CSR）。CA 通過 CSR 來創建出一個證書，用來證明你的網站是「安全的」。創建 CSR 的步驟如下：

* 創建一個本地自簽名證書（如前文所述）：

keytool -genkey -alias tomcat -keyalg RSA

-keystore <your\_keystore\_filename>

注意：在某些情況下，為了創建一個能耐生效的證書，你必須在「first- and lastname」字段中輸入網站的域名（比如：`www.myside.org`）。

* 然後利用下列代碼創建 CSR：

keytool -certreq -keyalg RSA -alias tomcat -file certreq.csr

-keystore <your\_keystore\_filename>

現在，我們得到了一個 certreq.csr 的文件，可以把它提交給 CA 了（CA 的網站上應有關於如何提交的文檔），審核通過後就會收到一個證書。

導入證書

接下來可以把證書導入到本地密鑰存儲庫中。首先你需要把鏈證書（Chain Certificate）或根證書（Root Certificate）導入到密鑰存儲庫中。然後繼續導入證書。

* 從獲得證書的 CA 那裡下載鏈證書。 如選擇 Verisign.com 的商業證書，則點擊：<http://www.verisign.com/support/install/intermediate.html>。  
  如選擇 Verisign.com 的試用證書，則點擊：<http://www.verisign.com/support/verisign-intermediate-ca/Trial_Secure_Server_Root/index.html>。  
  如選擇 Trustcenter.de，則點擊：<http://www.trustcenter.de/certservices/cacerts/en/en.htm#server>。  
  如選擇 Thawte.com，則點擊：<http://www.thawte.com/certs/trustmap.html>。
* 將鏈證書導入密鑰存儲庫：

keytool -import -alias root -keystore <your\_keystore\_filename>

-trustcacerts -file <filename\_of\_the\_chain\_certificate>

* 最後導入你的新證書：

keytool -import -alias tomcat -keystore <your\_keystore\_filename>

-file <your\_certificate\_filename>

疑難排解

以下列出了一些在設置 SSL 通信時經常會遇到的問題及其解決方法：

* 當 Tomcat 啟動時，出現這樣的異常信息：「ava.io.FileNotFoundException: {some-directory}/{some-file} not found.」

這很有可能是因為 Tomcat 無法在指定位置處找到密鑰存儲庫。默認情況下，密鑰存儲庫文件應以 .keystore 為後綴，位於用戶的 home 目錄下（也許很你的設置不同）。如果密鑰存儲庫文件在別處，則需要在 [Tomcat 配置文件](http://wiki.jikexueyuan.com/project/tomcat/ssl-tls.html#TomcatConfigFile)的 <Factory> 元素中添加一個 keystoreFile 屬性。

* 當 Tomcat 啟動時，出現這樣的異常信息：「java.io.FileNotFoundException: Keystore was tampered with, or password was incorrect.」

假如排除了有人惡意篡改密鑰存儲庫文件的因素，那麼出現這樣的異常，最有可能是因為 Tomcat 現在所用的密碼不同於你當初創建密鑰存儲庫文件時所用密碼。為瞭解決這個問題，你可以[重新創建密鑰存儲庫文件](http://wiki.jikexueyuan.com/project/tomcat/ssl-tls.html#Prepare Certificate Keystore)，或者在[Tomcat 配置文件](http://wiki.jikexueyuan.com/project/tomcat/ssl-tls.html#TomcatConfigFile)的 <Connector> 元素中添加或更新一個 keystoreFile 屬性。**注意**：密碼都是區分大小寫的。

* 當 Tomcat 啟動時，出現這樣的異常：「java.net.SocketException: SSL handshake error javax.net.ssl.SSLException: No available certificate or key corresponds to the SSL cipher suites which are enabled.」

出現這樣的異常，很有可能是因為 Tomcat 無法在指定的密鑰存儲庫中找到服務器密鑰的別名。查看一下 [Tomcat 配置文件](http://wiki.jikexueyuan.com/project/tomcat/ssl-tls.html#TomcatConfigFile)的 <Connector> 元素中所指定的 keystoreFile 和 keyAlias 值是否正確。**注意**：keyAlias 值可能區分大小寫。

如果出現了其他問題，可以求助 **TOMCAT-USER** 郵件列表，你可以在該郵件列表內找到之前消息的歸檔文件，以及訂閱和取消訂閱的相關信息。Tomcat 郵件列表的鏈接是：<http://tomcat.apache.org/lists.html>。

在應用中使用 SSL 跟蹤會話

這是一個 Servlet 3.0 規範中的新功能。由於它將 SSL 會話 ID 關聯到物理的客戶端服務器連接上，所以導致了一些約束與侷限：

* Tomcat 必須有一個屬性 **isSecure** 設為 true 的連接器。
* 如果 SSL 連接器通過代理或硬件加速器來管理，則它們必須填充 SSL 請求報頭（參見 [SSLValve](http://tomcat.apache.org/tomcat-8.0-doc/config/valve.html)），從而使 SSL 會話 ID 可見於 Tomcat。
* 如果 Tomcat 終止了 SSL 連接，則不可能採用會話重複，因為 SSL 會話 ID 在兩個端點處都是不一樣的。

為了開啟 SSL 會話跟蹤，只需使用上下文偵聽器將上下文的跟蹤模式設定為 SSL 即可（如果還開啟了其他跟蹤模式，則會優先使用它）。如下所示：

package org.apache.tomcat.example;

import java.util.EnumSet;

import javax.servlet.ServletContext;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

import javax.servlet.SessionTrackingMode;

public class SessionTrackingModeListener implements ServletContextListener {

@Override

public void contextDestroyed(ServletContextEvent event) {

// Do nothing

}

@Override

public void contextInitialized(ServletContextEvent event) {

ServletContext context = event.getServletContext();

EnumSet<SessionTrackingMode> modes =

EnumSet.of(SessionTrackingMode.SSL);

context.setSessionTrackingModes(modes);

}

}

注意：SSL 會話跟蹤是針對 BIO、NIO 以及 NIO2 連接器來實現的，目前還沒有針對 APR 連接器的實現。

其他技巧

要想從請求中訪問 SSL 會話 ID，可使用：

String sslID = (String)request.getAttribute("javax.servlet.request.ssl\_session\_id");

關於這一方面的其他內容，可參看[Bugzilla](https://bz.apache.org/bugzilla/show_bug.cgi?id=22679)。

為了終止 SSL 會話，可以使用：

// Standard HTTP session invalidation

session.invalidate();

// Invalidate the SSL Session

org.apache.tomcat.util.net.SSLSessionManager mgr =

(org.apache.tomcat.util.net.SSLSessionManager)

request.getAttribute("javax.servlet.request.ssl\_session\_mgr");

mgr.invalidateSession();

// Close the connection since the SSL session will be active until the connection

// is closed

response.setHeader("Connection", "close");

注意：由於使用了 SSLSessionManager 類，所以這段代碼是專門針對 Tomcat 的。當前只適用於 BIO、NIO 以及 NIO2 連接器，不適合 APR/原生連接器。