# CAS

(Central Authentication Service)

# 开源

# 框架

# SSO

(Single Sign-on)

# 的

# 实

# 现

Qrh 工作室

# 目录


(一) SSO (Single Sign-on) 原理 (单点登录)......3
(二) CAS 的基本原理......3
✧ CAS 的结构体系......3
✧ CAS 协议......4
✧ CAS 如何实现 SSO......4
(三) 实践配置......5
(一) 添加安全证书......5
(二) 实现 CAS 系统......6
(四) 结合实际的环境的扩展......9
1 多个 web 应用如何实现单点登陆......9
2 认证业务方法的扩展：......10
3 如何在这取得用户名称......11
(五) CAS 安全性......11
■ TGC/PGT 安全性......11
■ Service Ticket/Proxy Ticket 安全性......12
附录一 名称解释......12
附录二 Keytool 的介绍......13
◆ 密钥与证书......13
◆ 创建一个证书......13
◆ 导出到证书文件......13
◆ 查看证书的信息......14
◆ 删除密钥库中的条目......14
◆ 修改证书条目口令......14
附录三 HTTPS 和 SSL 的介绍......14
1 . HTTPS 简介......14
2 . SSL(Server Socket Layer) 简介......14
3 . SSL 工作原理......14

# （一） SSO（Single Sign-on）原理（单点登录）

SSO 分为 Web-SSO 和桌面 SSO。桌面 SSO 体现在操作系统级别上。Web-SSO 体现在客户端，主要特点是：SSO 应用之间使用 Web 协议（如 HTTPS），并且只有一个登录入口。我们所讲的 SSO，指 Web SSO 。

SSO 的体系中，有下面三种角色：

- ✧ User（多个）
- ✧ Web 应用（多个）
- ✧ SSO 认证中心（一个）

SSO 实现模式千奇百怪，但万变不离其宗，包含以下三个原则：

- 所有的登录都在 SSO 认证中心进行。
- SSO 认证中心通过一些方法来告诉 Web 应用当前访问用户究竟是不是通过认证的用户。
- SSO 认证中心和所有的 Web 应用建立一种信任关系。

# （二） CAS 的基本原理

CAS官方网址：*http://www.ja-sig.org/*

CAS(Central Authentication Service) 是 Yale 大学发起的构建 Web SSO 的 Java 开源项目。

✧ CAS 的结构体系

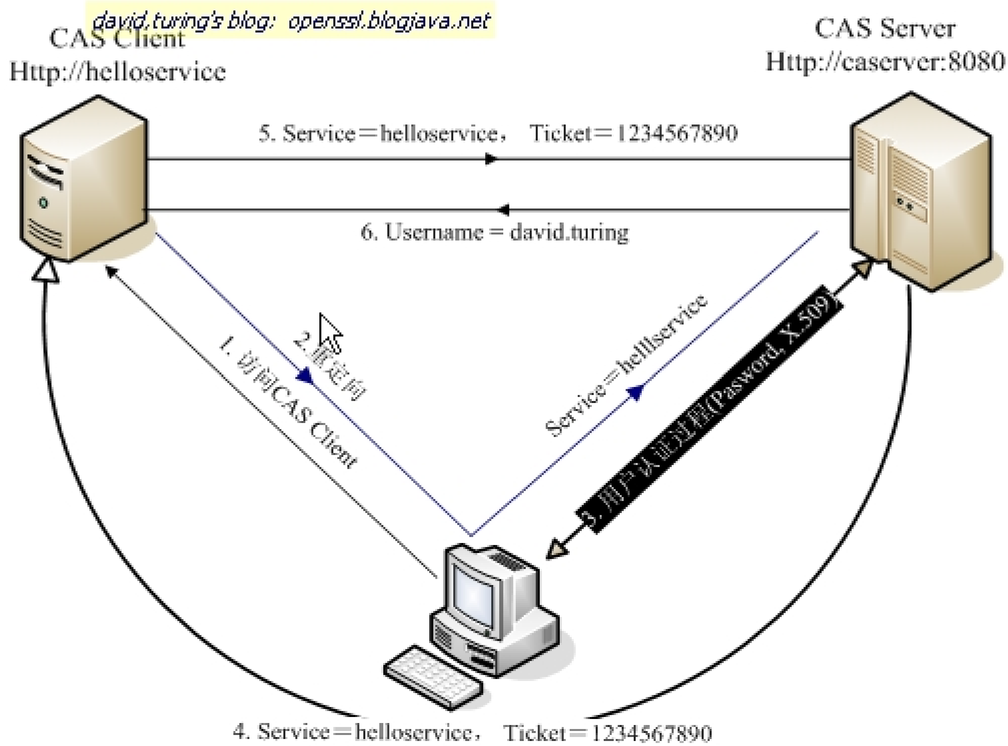
i. CAS Server

CAS Server 负责完成对用户信息的认证，需要单独部署，CAS Server 会处理用户名 / 密码等凭证 (Credentials) 。

ii. CAS Client

CAS Client 部署在客户端，当有对本地 Web 应用受保护资源的访问请求，并且需要对请求方进行身份认证，重定向到 CAS Server 进行认证。

- ✧ CAS 协议
  - i. 基础协议




上图是一个基础的 CAS 协议， CAS Client 以 过滤器的方式保护 Web 应用的受保护资源，过滤从客户端过来的每一个 Web 请求，同时， CAS Client 会分析 HTTP 请求中是否包请求 Service Ticket( 上图中的 Ticket) ，如果没有，则说明该用户是没有经过认证的， CAS Client 会重定向用户请求到 CAS Server （ Step 2 ）。 Step 3 是用户认证过程，如果用户提供了正确的认证信息 ， CAS Server 会产生一个随机的 Service Ticket ，会向 User 发送一个 Ticket granting cookie (TGC) 给 User 的浏览器，并且重定向用户到 CAS Client （附带刚才产生的 Service Ticket)，Step 5 和 Step6 是 CAS Client 和 CAS Server 之间完成了一个对用户的身份核实，用 Ticket 查到 Username ，认证通过。

- ✧ CAS 如何实现 SSO

当用户访问 Helloservice2 再次被重定向到 CAS Server 的时候， CAS Server 会主动获到这个 TGC cookie ，然后做下面的事情：

- 如果 User 的持有 TGC 且其还没失效，那么就走基础协议图的 Step4 ，达到了 SSO 的效果。
- 如果 TGC 失效，那么用户还是要重新认证 ( 走基础协议图的 Step3) 。

## （三） 实践配置

下面我们以 apache-tomcat-6.0.18 为例进行说明(这里，我将 Server 和 Client 同时放在了同一个 Tomcat 服务器下)。

软件环境：apache-tomcat-6.0.18　　jdk1.6.0_10　　MySQL Server 5.0

下载　　Server 服务器端　cas-server-3.3.3-release.zip

Server 端需要的 jar 库：

cas-server-support-jdbc-3.3.3.jar

mysql-connector-java-5.1.7-bin.jar

org.springframework.jdbc-3.0.0.M3.jar

Client 客户端需要的 jar 库：

casclient-2.1.1.jar

commons-logging-1.0.4.jar[1]

下载官网

*http://www.ja-sig.org/*

*http://www.mysql.com*

*http://tomcat.apache.org/*

(一) 添加安全证书

将一个或者一些页面进行支持 HTTPS 传输协议（意义：对某些页面进行了安全传输）（**重点掌握**）

- 产生 SERVER 的证书库文件

keytool -genkey -alias tomcat　-keyalg RSA [-keystore keystore-file]

*注：默认生成到当前用户的文件下.keystore 文件*

并将证书文件放在 web 容器的 conf 目录下。

- (在 server 端) 修改服务端 Tomcat 配置文件，启用 SSL 和 HTTPS

*$CATALINA_HOME/conf/server.xml* 里取消下面代码的注释

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"/>
```

在其中添加　keystorePass="8903239" keystoreFile="/conf/.keystore"

【配置好的例子】

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
               keystorePass="8903239" keystoreFile="/conf/.keystore"/>
```

*注意：keystorePass="8903239"（这个问证书库文件的密码，也就是上面配置产生的一个密码）keystoreFile="/conf/.keystore"（这是证书库文件的存放路径，其中根目录“/”为 tomcat 的安装路径）*

- 在 web 应用的 *WEB-INF/web.xml* 文件中增加

[1] *若无 commons-logging-*.jar 服务器启动时会出现错误----Error FilterStart*

```
<security-constraint>
    <web-resource-collection >
        <web-resource-name >SSL</web-resource-name> <!--名字随便取-->
        <url-pattern>/jsp/jsp2/el/*</url-pattern>  <!--针对 Tomcat 自带的 examples 应用-->
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

*解释：transport-guarantee 元素指定了客户端和服务端的通信关系，有 NONE，INTEGRAL,CONFIDENTIAL。NONE 表示着应用不需要任何传输保障。INTEGRAL 表示着在数据在客户端到服务端的过程中不能有任何改变。CONFIDENTIAL 表示在传输过程中防止其他传输内容的干扰。在使用 SSl 时常用的就 INTEGRAL 或 CONFIDENTIL。*

- 进行访问测试（如图），点击 *添加一个例外*

（二） 实现 CAS 系统

1. 产生 SERVER 的证书库文件

keytool -genkey -alias tomcat  -keyalg RSA [-keystore keystore-file]

并将证书文件放在 web 容器的 conf 目录下。

2. （在 server 端）修改服务端 Tomcat 配置文件，启用 SSL 和 HTTPS

*$CATALINA_HOME/conf/server.xml* 里取消注释

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"/>
```

添加  keystorePass="8903239" keystoreFile="/conf/.keystore"

【配置好的例子】

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
               keystorePass="8903239" keystoreFile="/conf/.keystore"/>
```

*注意：keystorePass="8903239"（这个问证书库文件的密码，也就是上面配置产生的一个密码） keystoreFile="/conf/.keystore"（这是证书库文件的存放路径，其中根目录“/”为 tomcat 的安装路径）*

3. 将 *cas-server-3.3.3-release.zip* 解压，并将 *\cas-server-3.3.3\modules cas-server-webapp-3.3.3.war* 拷贝到 webapps 下并改名为 *cas.war*。

4. 将 *casclient-2.1.1.jar* 和 *commons-logging-1.0.4.jar*[2] 拷贝到client服务器上(这里为同一tomcat)的web应用的 *WEB-INF/lib* 目录下（如果没有就建一个）

5. 修改 web 应用的 *WEB-INF/web.xml*

在要使用 CAS 的客户端应用里设置（以 examples 这个 Tomcat 自带 APP 为例，在应用时，所有客户端均进行类似配置），我们使用 ServletFilter(CAS client 里提供的)来实现 SSO 的检查。

修改 *examples/WEB-INF/web.xml*

```
<filter>
  <filter-name>CASFilter</filter-name>
    <filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
    <init-param>
        <param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
        <param-value>https://server:port/cas/login</param-value>
    </init-param> <!--这里是 CAS server 的 loginURL-->

    <init-param>
        <param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
        <param-value>https://server.name:port/cas/proxyValidate</param-value>
    </init-param>      <!--这里是 CAS server 的 URL 验证器-->
      <init-param>
        <param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
        <param-value>client:port </param-value>
       </init-param> <!--client:port 就是需要 CAS 需要拦截的地址和端口，一般就是这个
                     TOMCAT 所启动的 IP（默认为 localhost）和 port(默认 8080)-->
  </filter>
  <filter-mapping>
      <filter-name>CASFilter</filter-name>
      <url-pattern>/*</url-pattern>
  </filter-mapping>
```

[2] *若无 commons-logging-*.jar 服务器启动时会出现错误----Error FilterStart*

【配置好的例子】

```
<filter>
   <filter-name>CASFilter</filter-name>
   <filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
   <init-param>
        <param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
        <param-value>https://localhost:8443/cas/login</param-value>
   </init-param>
   <init-param>
        <param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
        <param-value>https://localhost:8443/cas/proxyValidate</param-value>
   </init-param>
   <init-param>
             <param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
             <param-value>localhost:8080</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>CASFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

6. 导出 Server 端的的证书文件（证书文件只包含公钥）

   keytool -export -file myserver.crt -alias aliasName -keystore keystoreFile

例： keytool -export -file myserver.crt -alias tomcat -keystore .keystore

7. 在客户端的 JVM 里的证书库 *cacerts* 中导入信任的 SERVER 的证书(根据情况有可能需要管理员权限)

   keytool -import -keystore cacerts -file myserver.crt -alias aliasName（别名）

例： keytool -import -keystore cacerts -file myserver.crt -alias tomcat

然后将 *cacerts* 复制到 *%JAVA_HOME%/jre/lib/security/* 目录下覆盖原文件

8. 测试.

   把 server 和 client 分别起来(这里为同一个 Tomcat,实际应用时可以在多个服务器上,且 client 可以为多个应用)，检查启动的 LOG 是否正常，如果一切 OK，就访问

   *http://localhost:8080/examples/jsp/jsp2/el/basic-arithmetic.jsp*

   系统会自动跳转到一个验证页面（如图），随便输入一个相同的账号,密码，验证通过后就会访问

   到真正的 *basic-arithmetic.jsp* 了

# （四） 结合实际的环境的扩展

1　多个 web 应用如何实现单点登陆

（！——大家思考一下：如果我想在配置一个客户端，需要什么步骤?）

下面以 tomcat 自带 web 应用 examples 为例子，进行下面得阐述：

- 在 */webapps/examples/WEB-INF/web.xml* 文件中进行配置：

```
<filter>
  <filter-name>CASFilter</filter-name>
  <filter-class>edu.yale.its.tp.cas.client.filter.CASFilter</filter-class>
  <init-param>
      <param-name>edu.yale.its.tp.cas.client.filter.loginUrl</param-name>
      <param-value>https://localhost:8443/cas/login</param-value>
  </init-param>
  <init-param>
      <param-name>edu.yale.its.tp.cas.client.filter.validateUrl</param-name>
      <param-value>https://localhost:8443/cas/proxyValidate</param-value>
  </init-param>
  <init-param>
          <param-name>edu.yale.its.tp.cas.client.filter.serverName</param-name>
          <param-value>localhost:8080</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CASFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

（2）在 *webapps/examples/WEB-INF/lib* 加入， *casclient.jar* 和 *commons-logging-1.0.4.jar*。

2　认证业务方法的扩展：

2.1.1 配置 CAS 使用数据库进行验证

① 在MySql中的Test库中新建app_user表

```
Create TABLE `app_user` (
    `username` varchar(30) NOT NULL default '',
    `password` varchar(45) NOT NULL default '',
    PRIMARY KEY  (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

② 添加以下用户：

```
Insert INTO `app_user` (`username`,`password`) VALUES
    ('test','test'),
    ('test1','test1');
```

③ 修改cas项目中的 *deployerConfigContext.xml* 文件

```
<!--
    <bean class="org.jasig.cas.authentication.handler.support.SimpleTestUsernamePasswordAut
    henticationHandler" />
--> <!— 注释掉该行，并加入以下代码 -->
<bean class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler">
    <property name="sql" value="select password from app_user where username=?" />
    <property name="dataSource" ref="dataSource" />
</bean>
```

④ 在 beans 节点下加入以下bean（dataSource）

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource"
destroy-method="close">
    <property name="driverClassName">
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property name="url">
        <value>jdbc:mysql://localhost:3306/test</value>
    </property>
    <property name="username">
        <value>root</value>
    </property>
    <property name="password">
        <value>8903239</value>
    </property>
</bean>
```

2.1.2 拷贝 *cas-server-support-jdbc-3.3.3.jar*和 *mysql-connector-java-5.1.7-bin.jar*和 *org.springframework.jdbc-3.0.0.M3.jar* 到 webapps/cas/WEB-INF/lib 下。

3 如何在这取得用户名称

```
<%@page contentType="text/html;charset=GBK"%>
<%
    String username=(String)session.getAttribute("edu.yale.its.tp.cas.client.filter.user");
%>
<p>当前得登陆用户：<%=username%></p>
    <%
        username = (String)session.getAttribute("edu.yale.its.tp.cas.client.filter.user");
    %>
```

# （五） CAS 安全性

## ■ TGC/PGT 安全性

TGC 也有自己的存活周期。下面是 CAS 的 */WEB-INF/spring-configuration/ticketExpirationPolicies.xml* 中，通过 *TimeoutExpirationPolicy* 来设置 CAS TGC 存活周期的参数，参数默认是 120 分钟，在合适的范围内设置最小值，太短，会影响 SSO 体验，太长，会增加安全性风险。

```
<bean id="grantingTicketExpirationPolicy"
      class="org.jasig.cas.ticket.support.TimeoutExpirationPolicy">
      <!-- This argument is the time a ticket can exist before its considered expired.  -->
      <constructor-arg  index="0"    value="7200000" />    <!-- 单位为：毫秒 -->
</bean>
```

■ Service Ticket/Proxy Ticket 安全性

设用户拿到 Service Ticket 之后，他请求 helloservice 的过程又被中断了， Service Ticket 就被空置了，事实上，此时， Service Ticket 仍然有效。 CAS 规定 Service Ticket 只能存活一定的时间，然后 CAS Server 会让它失效。通过在 applicationContext.xml 中配置下面的参数，可以让 Service Ticket 在访问多少次或者多少秒内失效。

下面是 CAS 的 /WEB-INF/spring-configuration/ticketExpirationPolicies.xml 中，

```
<!-- Expiration policies -->
<bean id="serviceTicketExpirationPolicy"
     class="org.jasig.cas.ticket.support.MultiTimeUseOrTimeoutExpirationPolicy">
     <!-- This argument is the number of times that a ticket can be used before its considered
          expired. -->
     <constructor-arg
          index="0"
          value="1" />

     <!-- This argument is the time a ticket can exist before its considered expired.  -->
     <constructor-arg
          index="1"
          value="300000" />//单位：毫秒
</bean>
```

*注：该参数在业务应用的条件范围内，越小越安全。*

# 附录一 名称解释

CAS(Central Authentication Service)

TGT(Ticket Granting Ticket)

ST(Service Ticket)

PGT(Proxy Granting Ticket)

Ticket Granting Cookie （简称 TGC）

Service Ticket（简称 ST）

# 附录二 Keytool 的介绍

◆ 密钥与证书

keytool JAVA 是个密钥和证书管理工具。它使用户能够管理自己的公钥/私钥对及相关证书，用于（通过数字签名）自我认证（用户向别的用户/服务认证自己）或数据完整性以及认证服务。它还允许用户储存他们的通信对等者的公钥（以证书形式）。通过 keytool –help 查看其用法，详细信息可以参考 http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html

创建证书 Java 中的 keytool.exe （位于 JDK/Bin 目录下）可以用来创建数字证书，所有的数字证书是以一条一条(采用别名区别)的形式存入证书库的中，证书库中的一条证书包含该条证书的私钥，公钥和对应的数字证书的信息。证书库中的一条证书可以导出数字证书文件，数字证书文件只包括主体信息和对应的公钥。

每一个证书库是一个文件组成，它有访问密码，在首次创建时，它会自动生成证书库，并要求指定访问证书库的密码。

在创建证书的的时候，需要填写证书的一些信息和证书对应的私钥密码。这些信息包括 CN=xx,OU=xx,O=xx,L=xx,ST=xx,C=xx，它们的意思是：

Ø CN（Common Name - 名字与姓氏）：其实这个“名字与姓氏”应该是域名，比如说 localhost 或是 blog.devep.net 之类的。输成了姓名，和真正运行的时候域名不符，会出问题。浏览器访问时，弹出一个对话框，提示“安全证书上的名称无效，或者与站点名称不匹配”，用户选择继续还是可以浏览网页。但是用 http client 写程序访问的时候，会抛出类似于“javax.servlet.ServletException: HTTPS hostname wrong: should be ”的异常。

Ø OU（Organization Unit - 组织单位名称）

Ø O（Organization - 组织名称）

Ø L（Locality - 城市或区域名称）

Ø ST（State - 州或省份名称）

Ø C（Country - 国家名称）

可以采用交互式让工具提示输入以上信息，也可以采用参数，如：-dname “CN=xx,OU=xx,O=xx,L=xx,ST=xx,C=xx”来自动创建。

◆ 创建一个证书

指定证书库为 D:/keystore/test，创建别名为 Tomcat 的一条证书，它指定用 RSA 算法生成，且指定密钥长度为 1024，证书有效期为 1 年：

keytool -genkey -alias Tomcat -keyalg RSA -keysize 1024 -keystore C:/keystore/test -validity 365

显示证书库中的证书使用如下命令： keytool -list -keystore C:/keystore/test 将显示 C:/keystore/test 证书库的的所有证书列表

◆ 导出到证书文件

使用命令：keytool -export -alias Tomcat -file C:/keystore/TC.cer -keystore C:/keystore/test 将把证书库 C:/keystore/test 中的别名为 Tomcat 的证书导出到 TC.cer 证书文件中，它包含证书主体的信息及证书的公钥，不包括私钥，可以公开。

导出的证书文件是以二进制编码文件，无法用文本编辑器正确显示，可以加上 -rfc 参数以一种可打印的编者编码输出。 如：

keytool -export -alias Tomcat -file C:/keystore/TC.cer -keystore C:/keystore/test –rfc

◆ 查看证书的信息

通过命令: keytool -printcert -file D:/keystore/TC.cer 可以查看证书文件的信息。 也可以在 Windows 资源管理器中双击产生的证书文件直接查看。

◆ 删除密钥库中的条目

keytool -delete -alias Tomcat -keystore C:/keystore/test

这条命令将 C:/keystore/test 库中的 Tomcat 这一条证书删除了。

◆ 修改证书条目口令

keytool -keypasswd -alias Tomcat -keystore C:/keystore/test，可以以交互的方式修改 C:/keystore/test 证书库中的条目为 Tomcat 的证书。

Keytool -keypasswd -alias Tomcat -keypass oldpasswd -new newpasswd -storepass storepasswd -keystore C:/keystore/test 这一行命令以非交互式的方式修改库中别名为 Tomcat 的证书的密码为新密码 newpasswd,行中的 oldpasswd 是指该条证书的原密码， storepasswd 是指证书库的密码。

# 附录三 HTTPS 和 SSL 的介绍

1 . HTTPS 简介

HTTPS（Secure Hypertext Transfer Protocol）安全超文本传输协议 它是由 Netscape 开发并内置于其浏览器中，用于对数据进行压缩和解压操作，并返回网络上传送回的结果。HTTPS 实际上应用了 Netscape 的完全套接字层（SSL）作为 HTTP 应用层的子层。（HTTPS 使用端口 443，而不是象 HTTP 那样使用端口 80 来和 TCP/IP 进行通信。）

2 . SSL(Server Socket Layer) 简介

在网络上信息在源 - 宿的传递过程中会经过其它的计算机。一般情况下，中间的计算机不会监听路过的信息。但在使用网上银行或者进行信用卡交易的时候有可能被监视，从而导致个人隐私的泄露。由于 Internet 和 Intranet 体系结构的原因，总有某些人能够读取并替换用户发出的信息。随着网上支付的不断发展，人们对信息安全的要求越来越高。因此 Netscape 公司提出了 SSL 协议，旨在达到在开放网络 (Internet) 上安全保密地传输信息的目的,这种协议在 WEB 上获得了广泛的应用。之后 IETF(ietf.org) 对 SSL 作了标准化,即 RFC2246 ,并将其称为 TLS（ Transport Layer Security ）,从技术上讲，TLS1.0 与 SSL3.0 的差别非常微小。

3 . SSL 工作原理

SSL 协议使用不对称加密技术实现会话双方之间信息的安全传递。可以实现信息传递的保密性、完整性，并且会话双方能鉴别对方身份。不同于常用的 http 协议，我们在与网站建立 SSL 安全连接时使用 https 协议，即采用 https://ip:port/ 的方式来访问。当我们与一个网站建立 https 连接时，我们的浏览器与 Web Server 之间要经过一个握手的过程来完成身份鉴定与密钥交换，从而建立安全连接。具体过程如下：

用户浏览器将其 SSL 版本号、加密设置参数、与 session 有关的数据以及其它一些必要信息发送到服务器。 服务器将其 SSL 版本号、加密设置参数、与 session 有关的数据以及其它一些必要信息发送给浏览器，同时发给浏览器的还有服务器的证书。如果配置服务器的 SSL 需要验证用户身份，还要发出请求要求浏览器提供用户证书。客户端检查服务器证书，如果检查失败，提示不能建立 SSL 连接。如果成功，那么继续。客户端浏览器为本次会话生成 pre-master secret ，并将其用服务器公钥加密后发送给服务器。如果服务器要求鉴别客户身份，客户端还要再对另外一些数据签名后并将其与客户端证书一起发送给服务器。 如果服务器要求鉴别客户身份，则检查签署客户证书的 CA 是否可信。如果不在信任列表中，结束本次会话。如果检查通过，服务器用自己的私钥解密收到的 pre-master secret ，并用它通过某些算法生成

本次会话的 master secret 。 客户端与服务器均使用此 master secret 生成本次会话的会话密钥（ 对称密钥 ）。在双方 SSL 握手结束后传递任何消息均使用此会话密钥。这样做的主要原因是对称加密比非对称加密的运算量低一个数量级以上，能够显著提高双方会话时的运算速度。 客户端通知服务器此后发送的消息都使用这个会话密钥进行加密。并通知服务器客户端已经完成本次 SSL 握手。服务器通知客户端此后发送的消息都使用这个会话密钥进行加密。并通知客户端服务器已经完成本次 SSL 握手。 本次握手过程结束，会话已经建立。双方使用同一个会话密钥分。