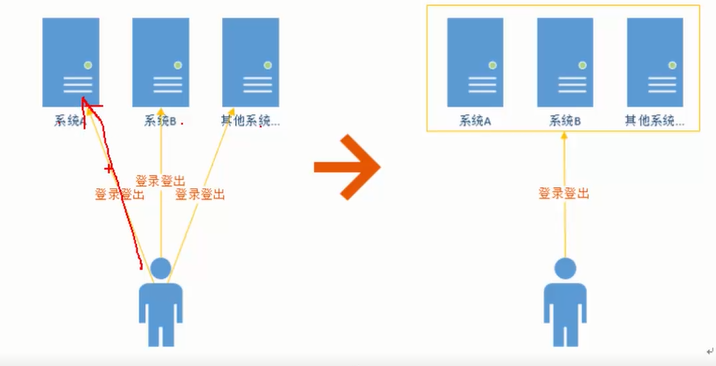
# 单点登录简介

## 什么是单点登录

单点登录（single Sign On），简称为SSO，是目前比较流行的企业业务整合的解决方案之一。SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。

## 为什么要使用单点登录

现在流行的web系统不断变的复杂，从最在的单系统单模块发展到现在的多系统多模块的应用群，用户在访问这些应用群中的各个系统的时候，是不是都要分别登录、登出？如果是这样，有N多系统，用户可能会疯掉。用户希望的是在这些系统中统一的登入和登出，换句话说，不管登录哪个系统之后，其他的子系统就无需在登陆了。



## 单点登录的分类

同域SSO

同一个域名下的系统 如：

Web1.x.com

Web2.x.com

Web3.x.com

跨域SSO

不同域名下的系统 如：

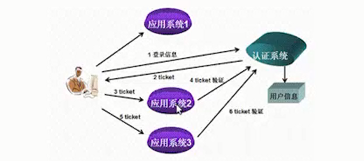
https://weibo.com/ 微博

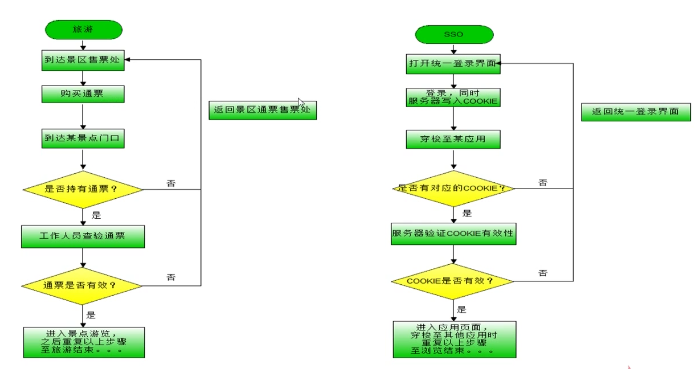
http://blog.sina.com.cn/ 博客

## 1.4.单点登录的技术实现机制

单点登录的机制其实是比较简单的，用一个现实中的例子作比较，颐和园是北京著名的旅游景点，在颐和园内部有许多独立的景点，例如“苏州街”，“佛香阁”和“德和园”，都可以在各个景点门口单独买票。很多游客需要游玩所有的景点，这种门票方式很不方便，需要在每个景点门口排队买票，钱包拿进拿出的，容易丢失，很不安全。于是绝大多数游客选择在门口买一张通票（也叫套票），就可以玩遍所有的景点而不需要重新再买票，他们只需要在每个景点门口出示一下刚才买的套票就能够被允许进入每个独立的景点。

单点登录的机制也一样，如下图所示，当用户第一次访问应用系统1的时候，因为还没有登录，（1）会被引导到认证系统中登录，根据用户提供的登录信息，认证系统进行身份校验，（2）如果通过校验，认证系统会返回给用户一个认证凭据—ticket。（3，5）用户在访问别的应用系统的时候就会将这个ticket带上，作为自己的认证凭据，应用系统接收到请求之后会把ticket送到认证系统进行校验，（4，6）检查ticket的合法性，如果通过校验，用户就可以在不用再次登录的情况下访问应用系统2和应用系统3了。





# HTTPS简介

## 什么是https

在说https之前先说说什么是http,http就是我们平时浏览网页时候使用的一种协议，http协议传输的数据都是未加密的，也就是明文的，因此使用http协议传输的隐私信息非常不安全。为了能够保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer安全套接层）协议用于对http协议传输的数据进行加密，从而就诞生了https。

SSL目前的版本是3.0，被IETF（Internet Engineering Task Force国际互联网工程任务组）定义在RFC（Request For Comments 一系列以编号排定的文件）6101中，之后IETF对SSL3.0进行了升级，于是出现了TLS（Transport Layer Security安全传输层协议）1.0，定义在RFC2246中。

实际上我们现在的https都是用的TLS协议，但是由于SSL协议出现的时间比较早，并且依旧被现在的浏览器所支持，因此SSL依然是https的代名词。

但无论是TLS还是SSL都是上个世纪的事情，SSL最后一个版本是3.0，今后TLS将会继承SSL优良血统继续为我们进行加密服务。

目前TLS的版本是1.2，定义在RFC5246中，暂时还没有被广泛的使用。

## https的工作原理

HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。

TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用了非对称加密，对称加密以及HASH算法，握手过程的简单描述如下：

1. 浏览器将自己支持的一套加密规则发送给网站。
2. 网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址、加密公钥、以及证书的颁发机构等信息。
3. 获得网站证书后浏览器要做以下工作：
4. 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。
5. 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。
6. 使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。
7. 网站接收浏览器发来的数据之后要做以下的操作：
8. 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。
9. 使用密码加密一段握手消息，发送给浏览器。
10. 浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

这里浏览器与网站互相发送加密的握手消息并验证，目的是为了保证双方都获得了一致的密码，并且可以正常的加密解密数据，为后续真正数据的传输做一次测试。另外，HTTPS一般使用的加密与HASH算法如下：

非对称加密算法：RSA，DSA/DSS

对称加密算法：AES，RC4，3DES

HASH算法：MD5，SHA1，SHA256

其中非对称加密算法用于在握手过程中加密生成的密码，对称加密算法用于对真正传输的数据进行加密，而HASH算法用于验证数据的完整性。由于浏览器生成的密码是整个数据加密的关键，因此在传输的时候使用了非对称加密算法对其加密。非对称加密算法会生成公钥和私钥，公钥只能用于加密数据，因此可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心的保管自己的私钥，防止泄漏。

TLS握手过程中如果有任何错误，都会使加密连接断开，从而阻止了隐私信息的传输

# 3．Tomcat服务器配置https双向认证

## 3.1. 生成服务器证书

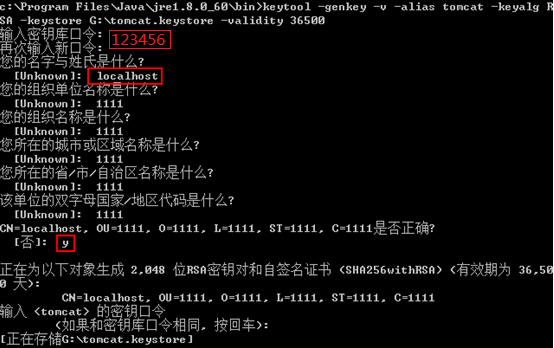
（1）打开命令控制台，进入%JAVA\_HOME%/bin目录

|  |
| --- |
| cd D:\projectFile\JDK\jre1.8.0\_161\lib |

（2）使用keytool为Tomcat生成证书

|  |
| --- |
| keytool -genkey -v -alias tomcat -keyalg RSA -keystore E:\test\tomcat-7.0\https\tomcat.keystore -validity 36500 |

（参数说明：“G:\tomcat.keystore”含义是将证书文件的保存路径，证书文件名称是tomcat.keystore； “-validity 36500”含义是证书有效期，36500表示100年，默认值是90天；“tomcat”为自定义证书名称）



**输入密钥库口令：**keystore密码（假设使用123456）

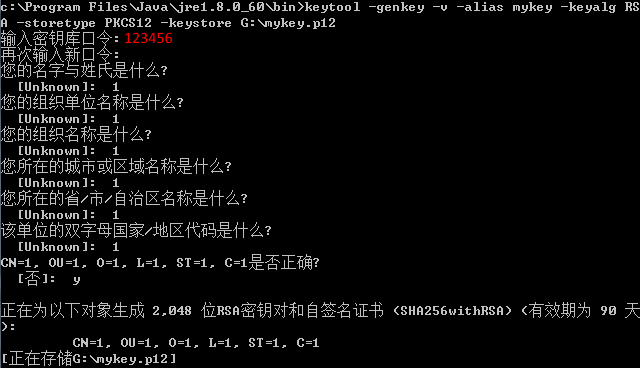
**您的名字与姓氏是什么：**必须是TOMCAT部署主机的域名或者IP[如：hangge.com 或者 10.1.25.119]（就是你将来要在浏览器中输入的访问地址），否则浏览器会弹出警告窗口，提示用户证书与所在域不匹配。在本地做开发测试时，应填入“localhost”。

**输入 <tomcat> 的密钥口令：**直接回车

## 3.2. 生成客户端证书

为浏览器生成证书，以便让服务器来验证它。为了能将证书顺利导入至IE和Firefox，证书格式应该是PKCS12，因此，使用如下命令生成：

|  |
| --- |
| keytool -genkey -v -alias mykey -keyalg RSA -storetype PKCS12 -keystore E:\test\tomcat-7.0\https\mykey.p12 |



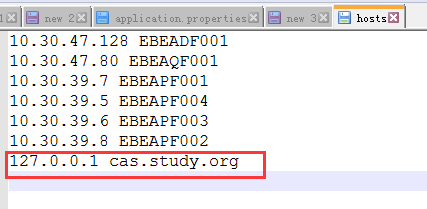
假设客户端证书密码“123456”，其他随便填。

## 3.3. 设置域名

SSL证书是基于域名的，如果您有自己的域名，请将域名指向到您的测试[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)。如果您没有自己的域名，那自己[模拟](https://www.baidu.com/s?wd=%E6%A8%A1%E6%8B%9F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)个域名吧，暂且使用 cas.study.org 。在测试的电脑上，设置hosts将该域名指向到当前服务器。具体设置方式如下：

打开文件C:\Windows\System32\drivers\etc\hosts添加如下行

|  |
| --- |
| 127.0.0.1 cas.study.org |



生成证书 指定一个位置

|  |
| --- |
| keytool -genkey -alias cas -keyalg RSA -keysize 2048 -keypass changeit -storepass changeit -keystore casexample.keystore -dname "CN=http://cas.study.org/,OU=casexample.com,O=casexample,L=casexample,ST=casexample,C=CN" -deststoretype pkcs12 |

Tomcat开启HTTPS服务，导入证书

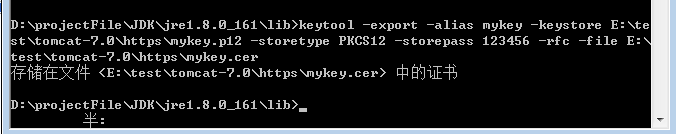
Tomcat默认情况下是打开HTTP服务的，HTTPS的服务是需要手动开发的，具体打开步骤如下：

## 3.3. 让服务器信任客户端证书

服务器要信任客户端证书，必须把客户端证书添加为服务器的信任认证。

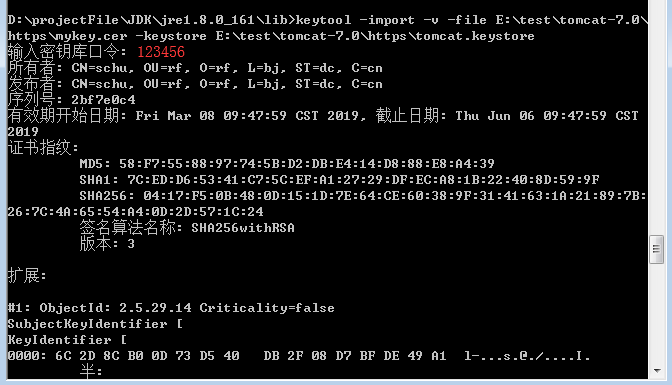
（1）由于不能直接将PKCS12格式的证书库导入，必须先把客户端证书导出为一个单独的CER文件，使用如下命令：（下面要用到客户端证书密码“123456”）

|  |
| --- |
| keytool -export -alias mykey -keystore E:\test\tomcat-7.0\https\mykey.p12 -storetype PKCS12 -storepass 123456 -rfc -file E:\test\tomcat-7.0\https\mykey.cer |



（2）将该文件导入到服务器的证书库，添加为一个信任证书使用命令如下：

|  |
| --- |
| keytool -import -v -file E:\test\tomcat-7.0\https\mykey.cer -keystore E:\test\tomcat-7.0\https\tomcat.keystore |



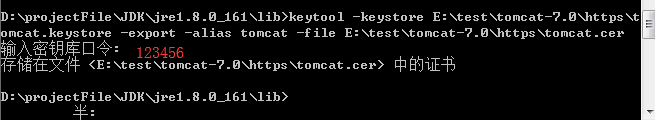
（3）通过**list**命令查看服务器的证书库，可以看到两个证书，一个是服务器证书，一个是受信任的客户端证书：

|  |
| --- |
| keytool -list -keystore E:\test\tomcat-7.0\https\tomcat.keystore |

## 3.4. 让客户端信任服务器证书

由于是双向SSL认证，客户端也要验证服务器证书。把服务器证书导出为一个单独的CER文件提供给客户端，使用如下命令：

|  |
| --- |
| keytool -keystore E:\test\tomcat-7.0\https\tomcat.keystore -export -alias tomcat -file E:\test\tomcat-7.0\https\tomcat.cer |



## 3.5. 经过上面操作，我们生成如下证书：



## 3.6. 服务器tomcat的配置

打开Tomcat根目录下的/conf/server.xml，找到Connector port="8443"配置段，修改为如下：

|  |
| --- |
| <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"  SSLEnabled="true" maxThreads="150" scheme="https"  secure="true" clientAuth="true" sslProtocol="TLS"  keystoreFile="E:\\test\\tomcat-7.0\\https\\tomcat.keystore" keystorePass="123456"  truststoreFile="E:\\test\\tomcat-7.0\\https\\tomcat.keystore" truststorePass="123456" /> |

**属性说明：**

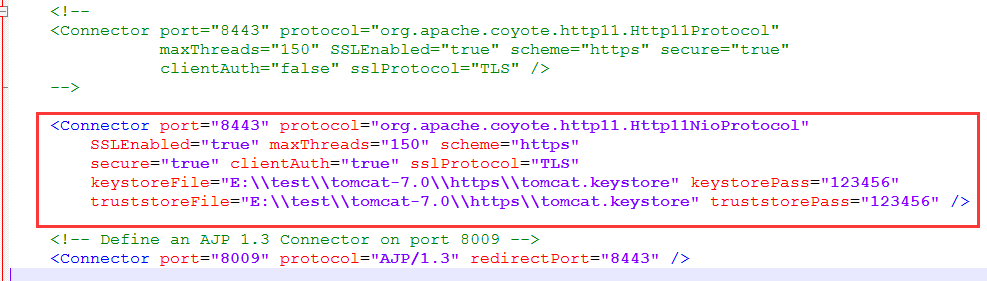
clientAuth:设置是否双向验证，默认为false，设置为true代表双向验证

keystoreFile:服务器证书文件路径

keystorePass:服务器证书密码

truststoreFile:用来验证客户端证书的根证书，此例中就是服务器证书

truststorePass:根证书密码



**测试服务端：（需要重新启动浏览器）**

浏览器输入https://localhost:8443/　（下图说明tomcat的https开启成功。报无效证书没关系，因为我们客户端还没导入证书）

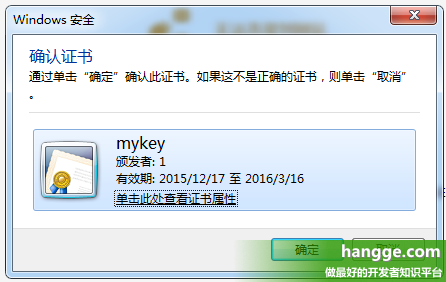


## 3.7. **客户端证书导入**

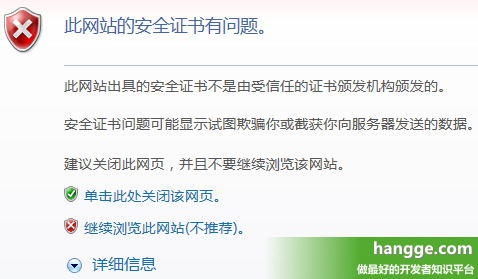
注意：每次操作后需要重新启动浏览器

（1）双击“**mykey.p12**”导入客户端证书

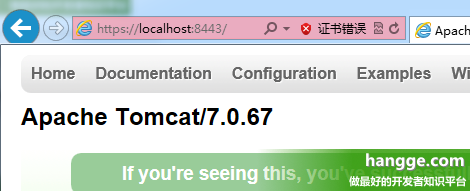
再次使用浏览器访问服务端，浏览器会让我们选择使用的证书。



但由于是自签名的证书，所以浏览器会警告我们不安全，选择继续好了：



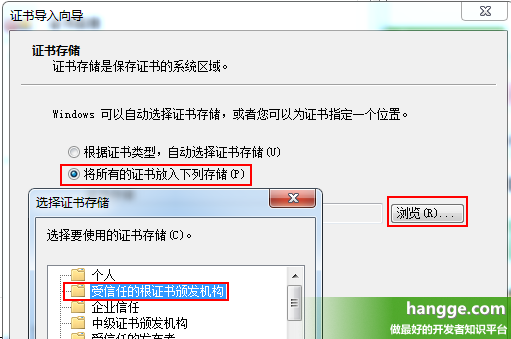
可以看到能成功访问了（地址栏“证书错误”说明目前还没双向验证，不过数据已经是使用HTTPS传输了）：



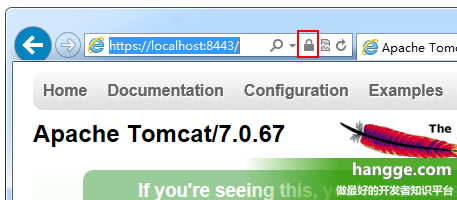
（2）导入服务器公钥证书（**tomcat.cer**）

由于是自签名的证书，为避免每次都提示不安全。这里双击tomcat.cer安装服务器证书。

注意：将证书填入到“受信任的根证书颁发机构”



再次重新访问服务器，会发现没有不安全的提示了，同时浏览器地址栏上也有个“锁”图标，表示本次会话已经通过HTTPS双向验证。



# 4.CAS简介

CAS官方建议采用overlay方式对cas进行个性化配置，有两处系统博文写得很好，推荐：

https://blog.csdn.net/u013825231/article/category/7517313

https://blog.csdn.net/weixin\_39206782/article/category/7641756

https://blog.csdn.net/u010588262/article/category/7548325

## 4.1.什么是CAS

随着SSO技术的流行，相关产品也比较多，其中CAS就是其中的一套方案。CAS（Central Authentication Service）中文翻译为统一身份认证服务或中央身份认证服务，它有服务端和客户端组成，实现了SSO，并且容易进行企业应用的集成。

CAS是Yale大学发起的一个开源项目，旨在为web应用系统提供一种可靠的单点登录方法，CAS在2004年12月正式成为JA-SIG的一个项目。

## 4.2.CAS的特点

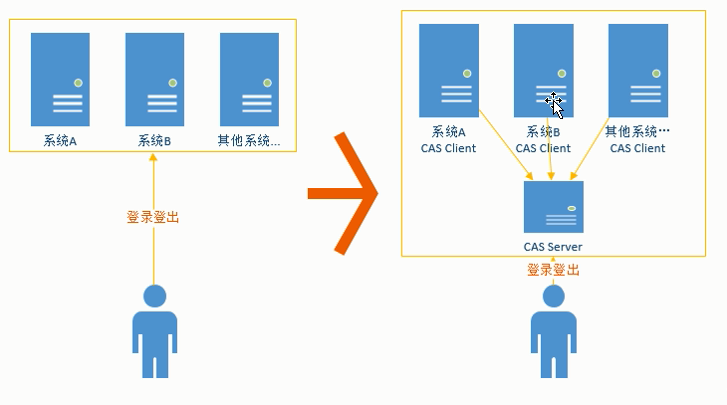
1、开源的企业级单点登录解决方案

2、CAS Server为独立部署的web应用。

3、CAS Client 支持非常多的客户端（这里指单点登录系统中的各个web应用），包括java、.Net、PHP、Perl、Apache、uPortal、Ruby等。

## 4.3.CAS的使用

有了CAS，我们的系统框架就演变为下面这样的：



从架构上可以看出，CAS包含两个部分：CAS Server和CAS Client。CAS Server需要独立部署，主要负责对用户的认证工作；CAS Client负责处理对客户端受保护资源的访问请求，需要登录时，重定向到CAS Server。

# 5.Cas Server搭建

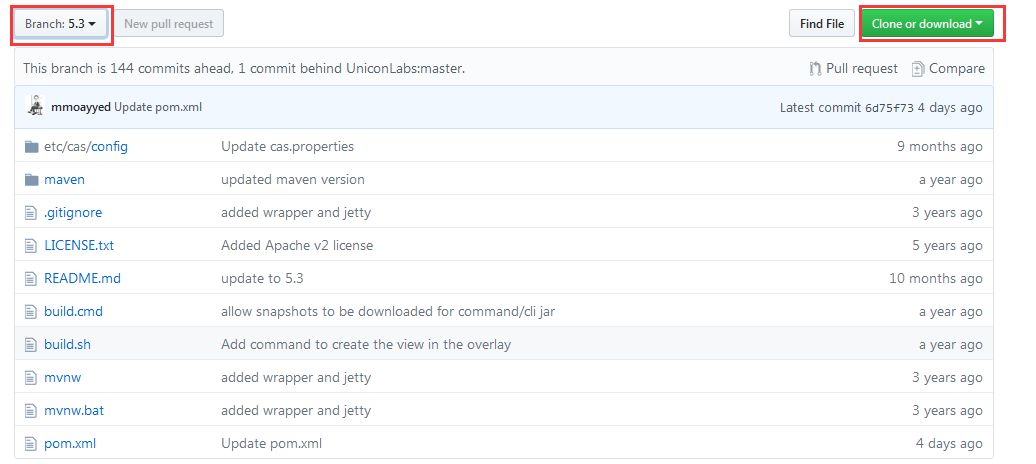
服务端官方下载：<https://github.com/apereo/cas-overlay-template>

客户端官方下载：<https://github.com/cas-projects/cas-sample-java-webapp>

## 4.1.**下载CAS5.3**

下载地址：[https://github.com/apereo/cas-overlay-template](https://github.com/apereo/cas-overlay-template" \t "_blank)

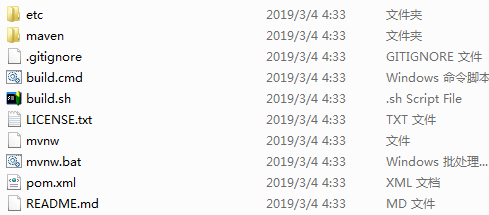
打开网址，点击左上角的Branch master可以选择要下载的版本



下载后



解压后



## 4.2. 服务端项目构建

注意：

. 以下操作都在win10下完成

. cas 5.3的编译jdk版本必须是1.8以上

. maven必须安装了和配置OK，这里使用的是3.6.0

. 构建是必须联网，因为要下载相关依赖

. 主要maven不要配置私服。

在cmd定义到文件夹cas-overlay-template-5.3中

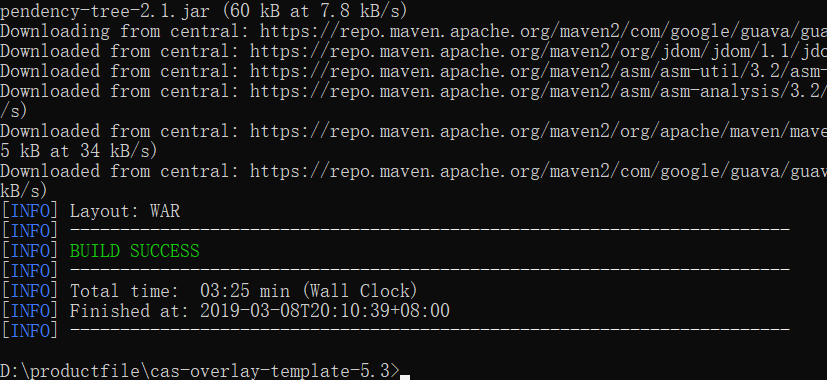
|  |
| --- |
| cd D:\productfile\cas-overlay-template-5.3 |

构建项目执行命令

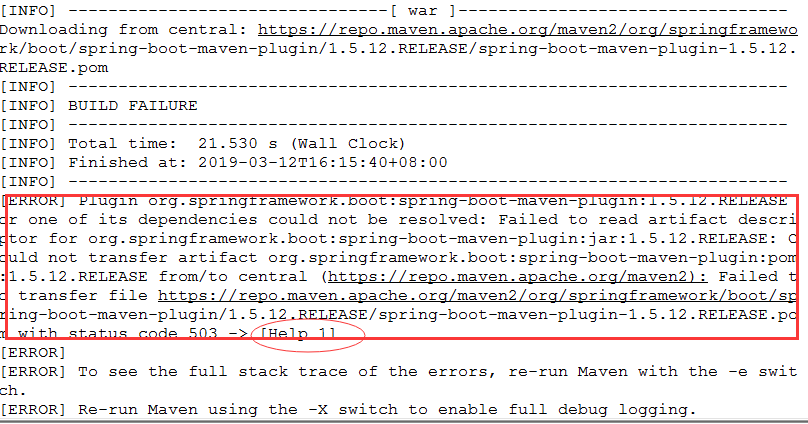
|  |
| --- |
| build.cmd package |



这个过程需要一点时间，执行完毕后，会有这样的提示：



注意：在这过程中如果报错可能是maven的问题



这个问题是网络限制的原因

解决方案:

1、使用阿里的云仓库，在maven的settings.xml文件的Mirrors中添加了如下内容：

|  |
| --- |
| <mirrors>    <!-- 阿里云仓库 -->  <mirror>  <id>alimaven</id>  <mirrorOf>central</mirrorOf>  <name>aliyun maven</name>  <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>  </mirror>    <!-- 中央仓库1 -->  <mirror>  <id>repo1</id>  <mirrorOf>central</mirrorOf>  <name>Human Readable Name for this Mirror.</name>  <url>http://repo1.maven.org/maven2/</url>  </mirror>    <!-- 中央仓库2 -->  <mirror>  <id>repo2</id>  <mirrorOf>central</mirrorOf>  <name>Human Readable Name for this Mirror.</name>  <url>http://repo2.maven.org/maven2/</url>  </mirror>    </mirrors> |

2.添加完成后，发现部分资源 可以下载了，但仍然有许多资源无法下载，后来参考了一些文档，发现pom.xml都是继承自super pom，super pom中的有如下内容：

|  |
| --- |
| <repositories>  <repository>  <id>central</id>  <name>Central Repository</name>  <url>http://repo.maven.apache.org/maven2</url>  <layout>default</layout>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories>    <pluginRepositories>  <pluginRepository>  <id>central</id>  <name>Central Repository</name>  <url>http://repo.maven.apache.org/maven2</url>  <layout>default</layout>  <snapshots>  <enabled>false</enabled>  </snapshots>  <releases>  <updatePolicy>never</updatePolicy>  </releases>  </pluginRepository>  </pluginRepositories> |

故当maven下载资源时，会优先使用[http://repo.maven.apache.org/maven2](http://repo.maven.apache.org/maven2" \t "_blank)去下载。

最终解决方案是在自定义的pom.xml中加入如下内容：

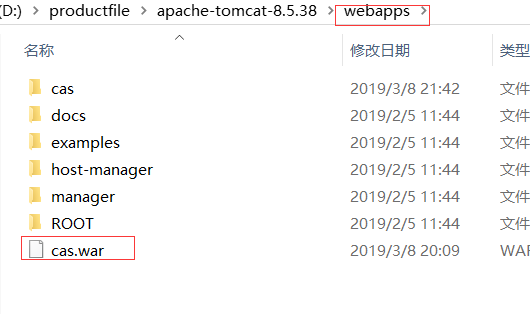
|  |
| --- |
| <repositories>  <repository>  <id>central</id>  <url>http://host:port/content/groups/public</url>  </repository>  </repositories>    <pluginRepositories>  <pluginRepository>  <id>central</id>  <url>http://host:port/content/groups/public</url>  </pluginRepository>  </pluginRepositories> |

最后的结果会生成一个target文件，文件中包含cas.war :

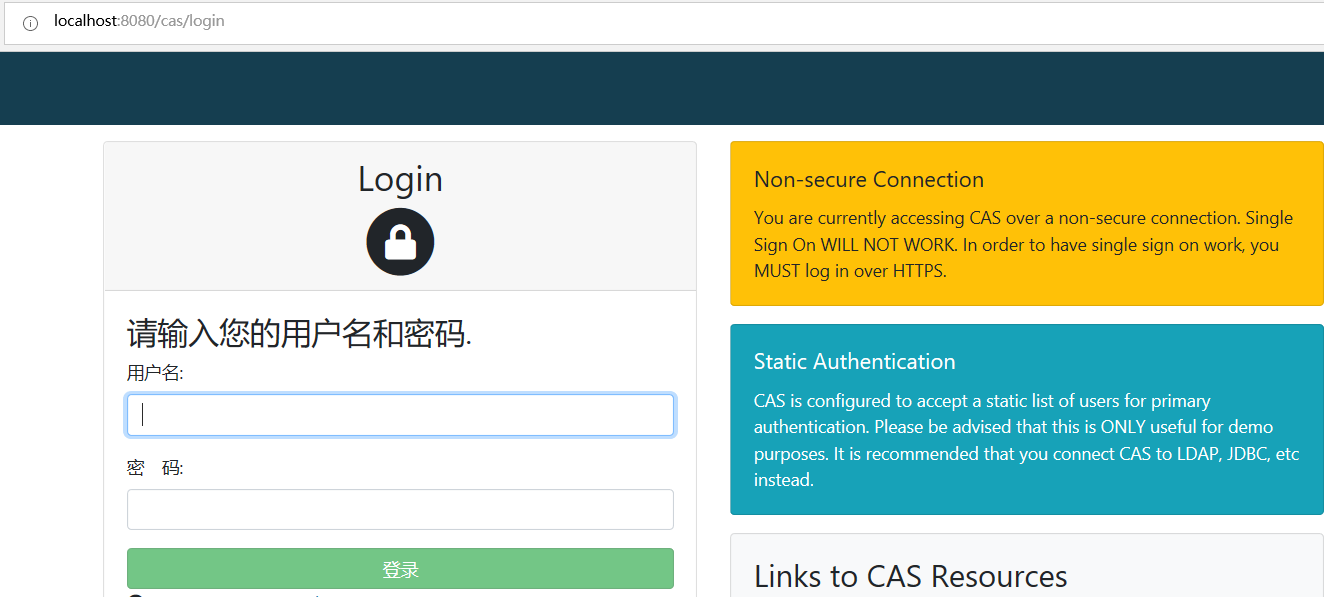


## 4.3. **CAS5.3的安装部署**

注意：tomcat版本必须为8.0以上

把生成的.war文件拷贝到[webapps](https://www.baidu.com/s?wd=webapps&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)目录下面，启动tomcat,会自动将war包解压 

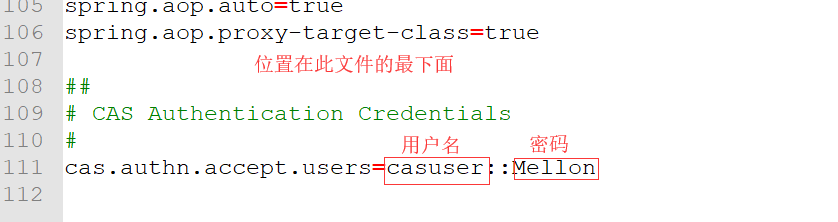
通过浏览器访问：http://localhost:8080/cas/login



查看上图表示安装成功。

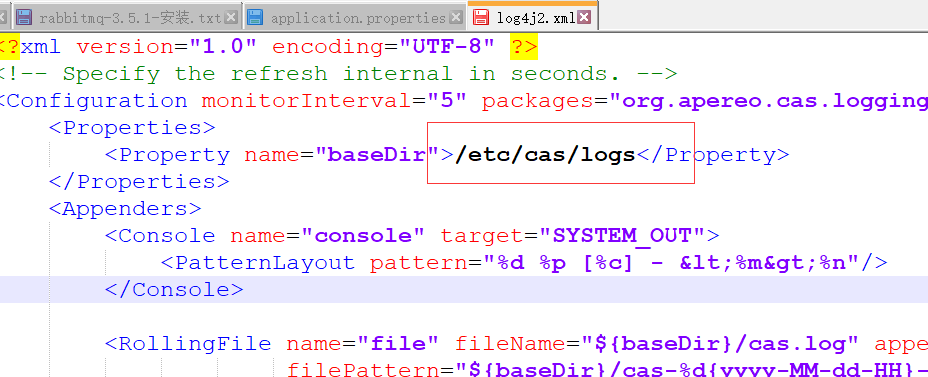
1.查找默认的用户名密码

默认的用户名密码在解压的项目cas\WEB-INF\classes\ application.properties中



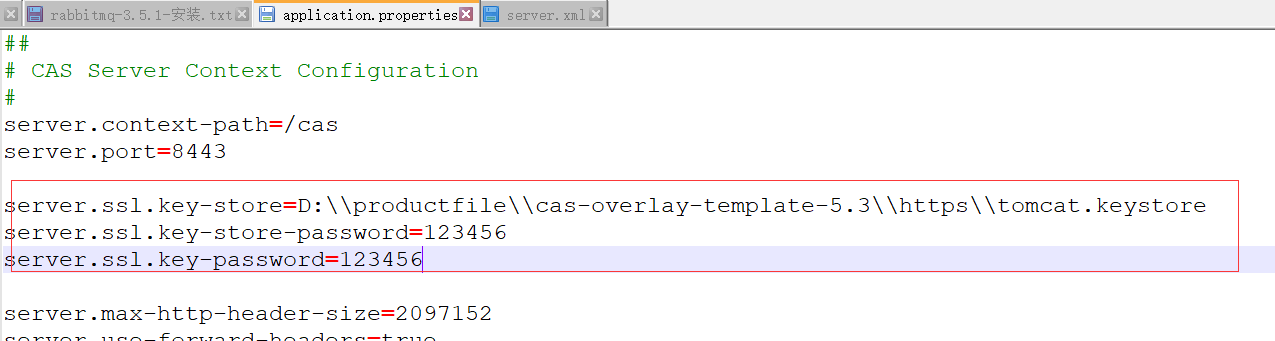
2.日志存放的位置

在webapps\cas\WEB-INF\classes\ log4j2.xml中默认存储在（当前磁盘下的这个位置，如：D:/etc/cas/logs），可进行修改。



1. 设置https证书

位置在webapps\cas\WEB-INF\classes\ application.properties中



# 6. cas服务器连接数据库

## 6.1.mysql数据库

pom.xml中添加数据库驱动、数据库包

|  |
| --- |
| <dependency>  <groupId>org.apereo.cas</groupId>  <artifactId>cas-server-support-jdbc-drivers</artifactId>  <version>${cas.version}</version>  </dependency>  <dependency>  <groupId>org.apereo.cas</groupId>  <artifactId>cas-server-support-jdbc</artifactId>  <version>${cas.version}</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>6.0.6</version>  </dependency> |



在application.properties配置文件里面加上如下简单的配置就可以达到数据库的访问了

|  |
| --- |
| cas.authn.jdbc.query[0].url=jdbcmysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&useSSL=false cas.authn.jdbc.query[0].user=root  cas.authn.jdbc.query[0].password=root  cas.authn.jdbc.query[0].sql=select \* from test\_table where name=?  cas.authn.jdbc.query[0].fieldPassword=pd  cas.authn.jdbc.query[0].driverClass=com.mysql.jdbc.Driver |

## 6.2.Oracle数据库

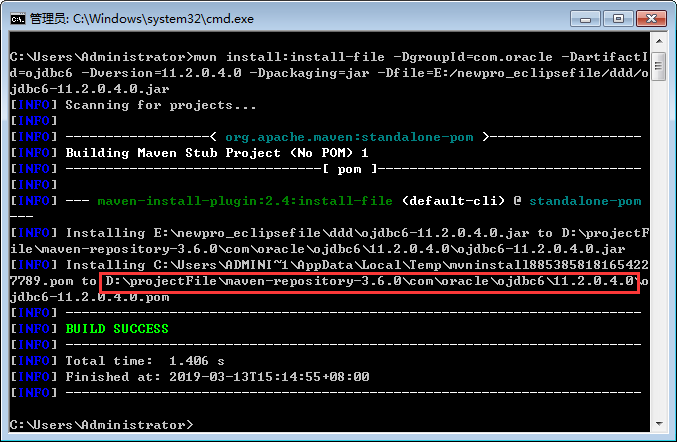
在pom文件中加入对应的依赖:

|  |
| --- |
| <dependency>  <groupId>org.apereo.cas</groupId>  <artifactId>cas-server-support-jdbc</artifactId>  <version>${cas.version}</version>  </dependency>  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc6</artifactId>  <version>11.2.0.4.0</version>  </dependency> |

本地下载ojdbc6-11.2.0.4.0.jar 到目录E:/newpro\_eclipsefile/ddd

Oracle驱动需要手动添加依赖

|  |
| --- |
| mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0.4.0 -Dpackaging=jar -Dfile=E:/newpro\_eclipsefile/ddd/ojdbc6-11.2.0.4.0.jar |



执行后他会本地maven仓库添加Oracle驱动

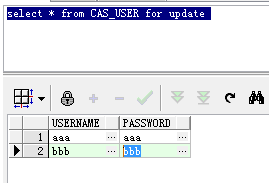
重新生成war包

|  |
| --- |
| build.cmd package |

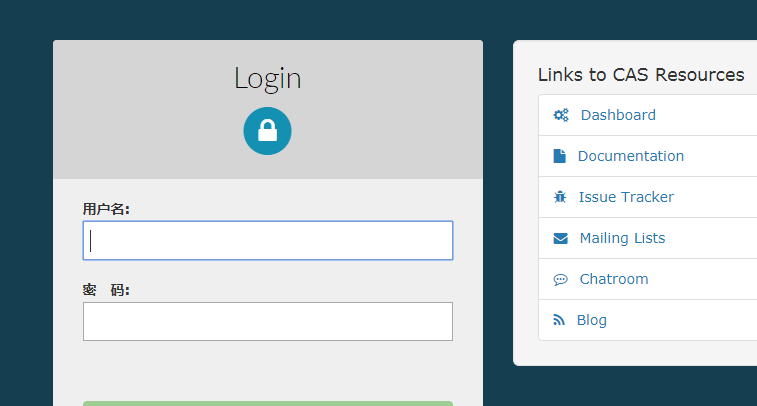
配置application.properties文件:

|  |
| --- |
| cas.authn.jdbc.query[0].url=jdbc:oracle:thin:@127.0.0.1:1521:orcl  cas.authn.jdbc.query[0].user=gjyw  cas.authn.jdbc.query[0].password=123456  cas.authn.jdbc.query[0].sql=select \* from CAS\_USER where username = ?  cas.authn.jdbc.query[0].fieldPassword=password  cas.authn.jdbc.query[0].driverClass=oracle.jdbc.driver.OracleDriver |

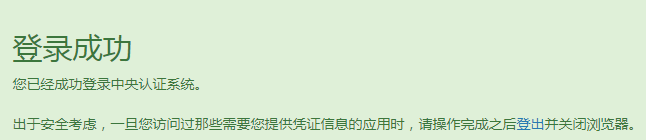
在oracle库中创建表CAS\_USER



启动Tomcat查看是否连接Oracle数据库成功



输入用户密码

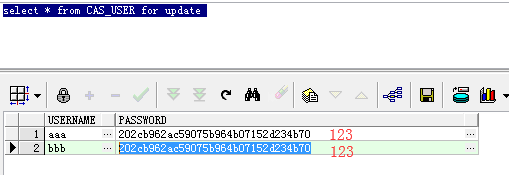


# 7.设置密码MD5加密

在application.properties文件中

|  |
| --- |
| cas.authn.jdbc.query[0].passwordEncoder.type=DEFAULT  cas.authn.jdbc.query[0].passwordEncoder.characterEncoding=UTF-8  cas.authn.jdbc.query[0].passwordEncoder.encodingAlgorithm=MD5 |

数据库密码改为MD5加密后的



启动Tomcat 输入密码为123 查看是否能登陆成功