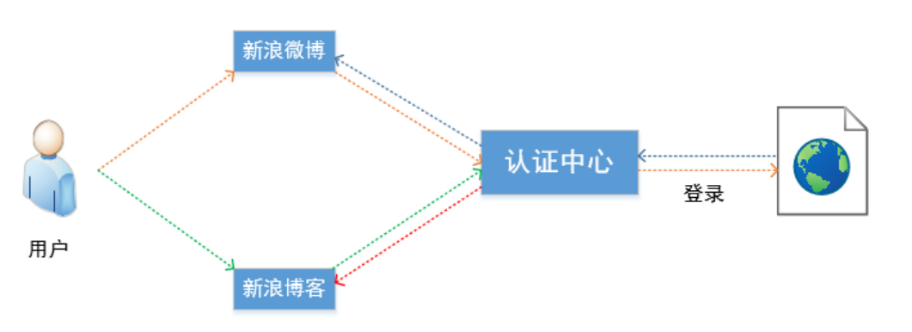
**CAS + Shiro实现单点登录方案**

1. **简介**

在多个相互信任的系统中，用户只需要登录一次就可以访问其他受信任的系统。例如新浪微博与新浪博客是相互信任的应用系统。



当用户首次访问新浪微博时，新浪微博识别到用户未登录，将请求重定向到认证中心，认证中心也识别到用户未登录，则将请求重定向到登录页。当用户已登录新浪微博访问新浪博客时，新浪博客识别到用户未登录，将请求重定向到认证中心，认证中心识别到用户已登录，返回用户的身份，此时用户无需登录即可使用新浪博客。只要多个系统使用同一套单点登录框架那么它们将是相互信任的。

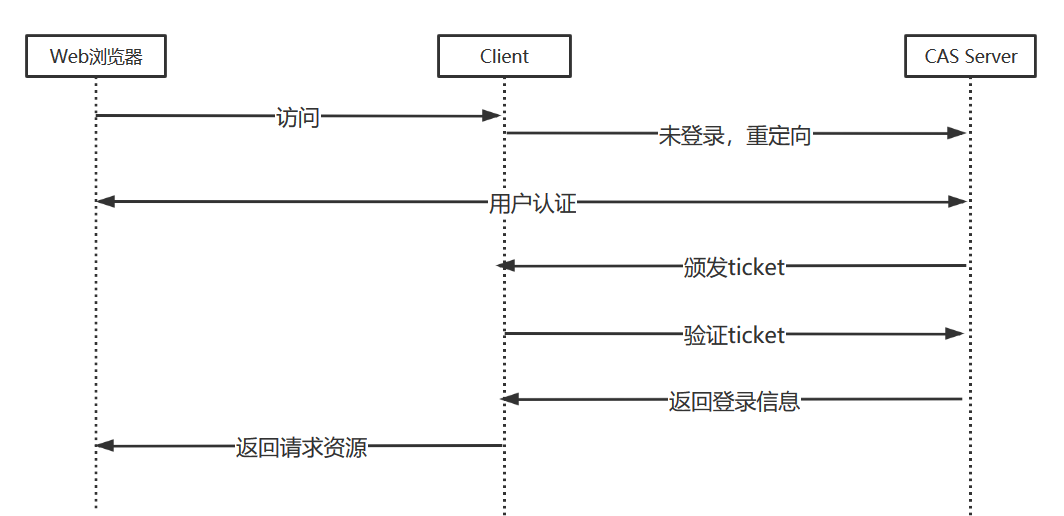
* 1. **CAS简介**

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。其包含CAS Client 和 CAS Server两部分。

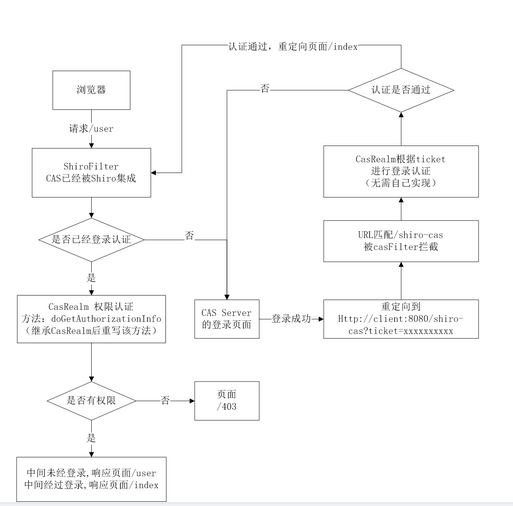
1. CAS Client：要使用单点登录的Web应用，将与同组下的Web应用构成相互信任的关系，只需在web应用中添加CAS提供的Listener和Filter即可成为CAS Client ，其主要负责对客户端的请求进行登录校验、重定向和校验ticket工作。
2. CAS Server：主要负责对用户的用户名/密码进行认证，颁发票据等，需要单独的进行部署。
   1. **Shiro简介**

Apache Shiro是一个强大且易用的Java安全框架,执行身份验证、授权、密码和会话管理。拥有三个核心组件：Subject, SecurityManager 和 Realms。Subject指的是当前用户，代表了当前用户的安全操作，SecurityManager则管理所有用户的安全操作，它是Shiro框架的核心，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务。Realm充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息。

1. **实现方案**
   1. **CAS访问流程**



1. 访问服务： SSO 客户端发送请求访问应用系统提供的服务资源。
2. 定向认证： SSO 客户端会重定向用户请求到 SSO 服务器。
3. 用户认证：用户身份认证。
4. 发放票据： SSO 服务器会产生一个随机的 Service Ticket 。
5. 验证票据： SSO 服务器验证票据 Service Ticket 的合法性，验证通过后，允许客户端访问服务。
6. 传输用户信息： SSO 服务器验证票据通过后，传输用户认证结果信息给客户端。
   1. **Shiro集成CAS后访问流程**

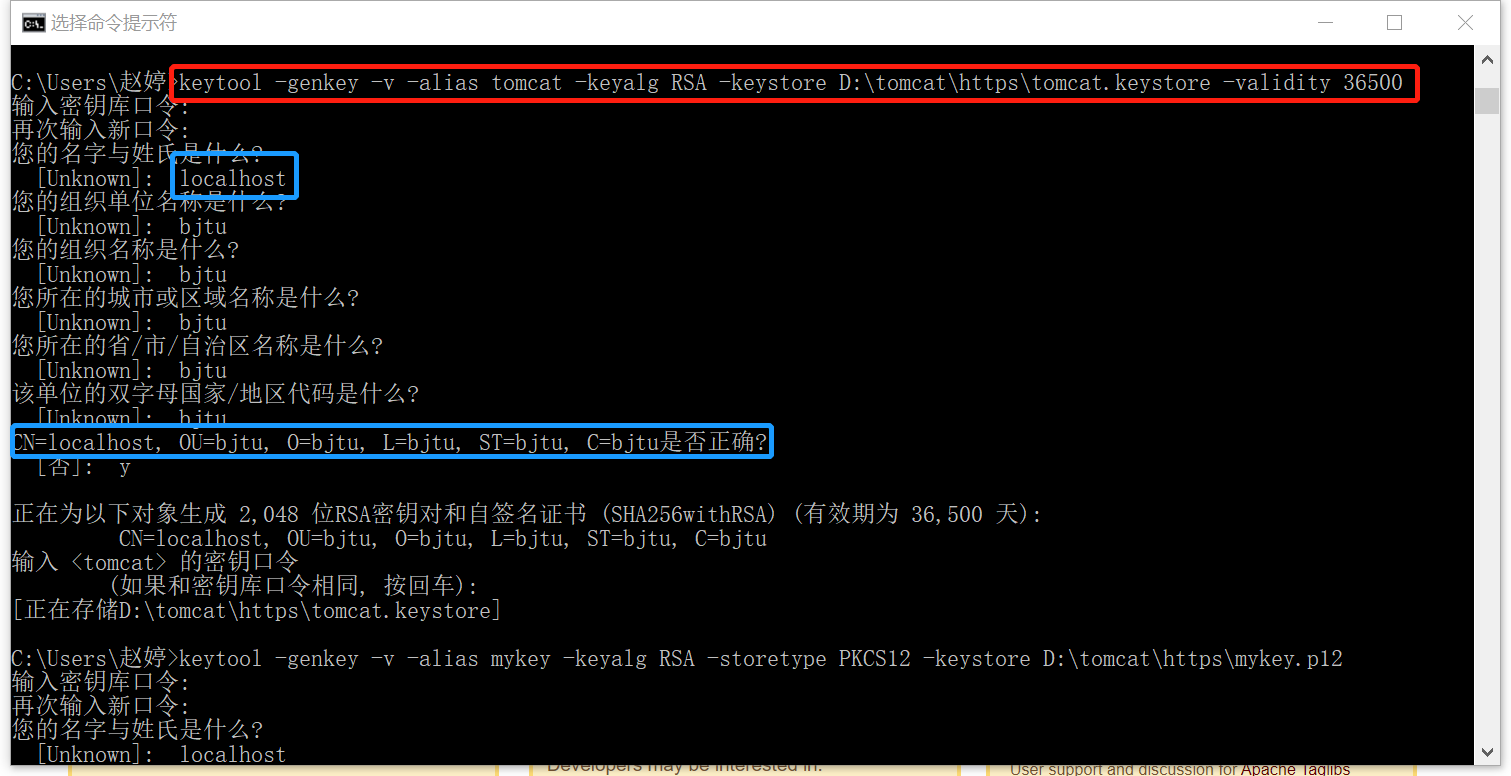


1. **搭建步骤**
   1. **配置https双向认证**

由于CAS一般使用的是https协议，在本次实现中，使用tomcat作为服务器，因此首先应该利用tomcat服务器配置https双向认证。

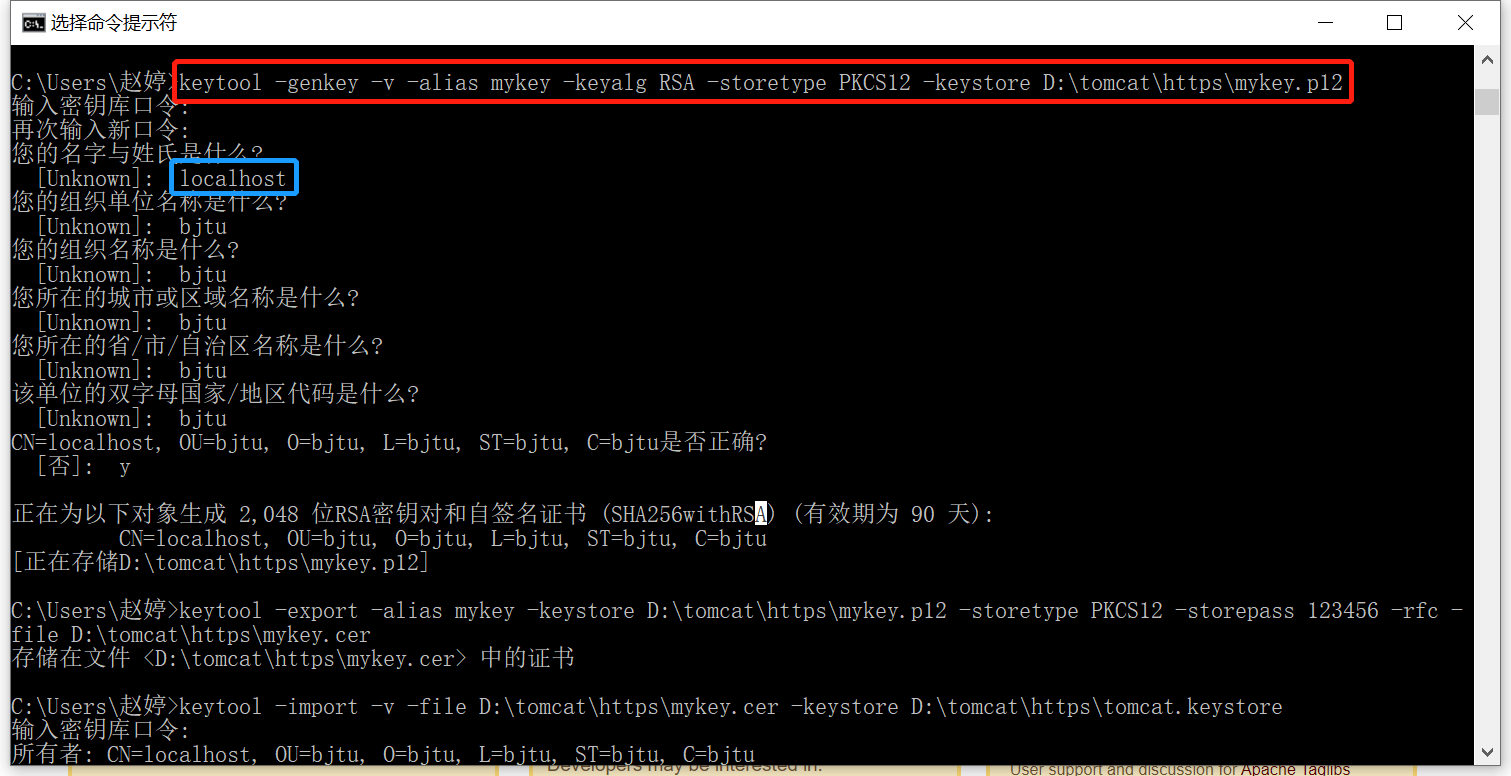
1. 为服务器生成证书

使用keytool为tomcat生成证书，假设目标机器的域名是"localhost",keystore文件存放在D:\tomcat\https\tomcat.keystore，口令为"password"。



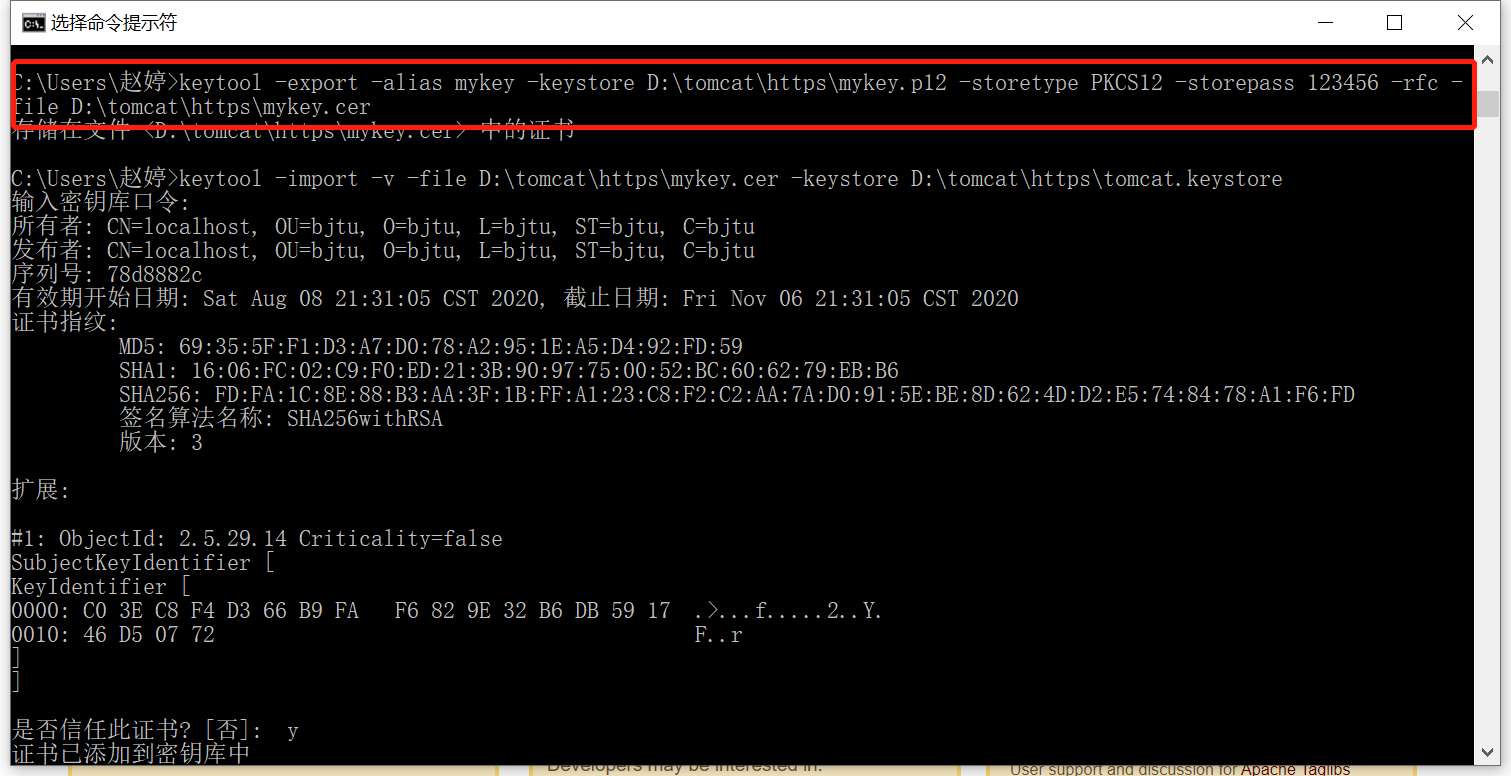
1. 为客户端生成证书

为浏览器生成证书，以便让服务器来验证它。

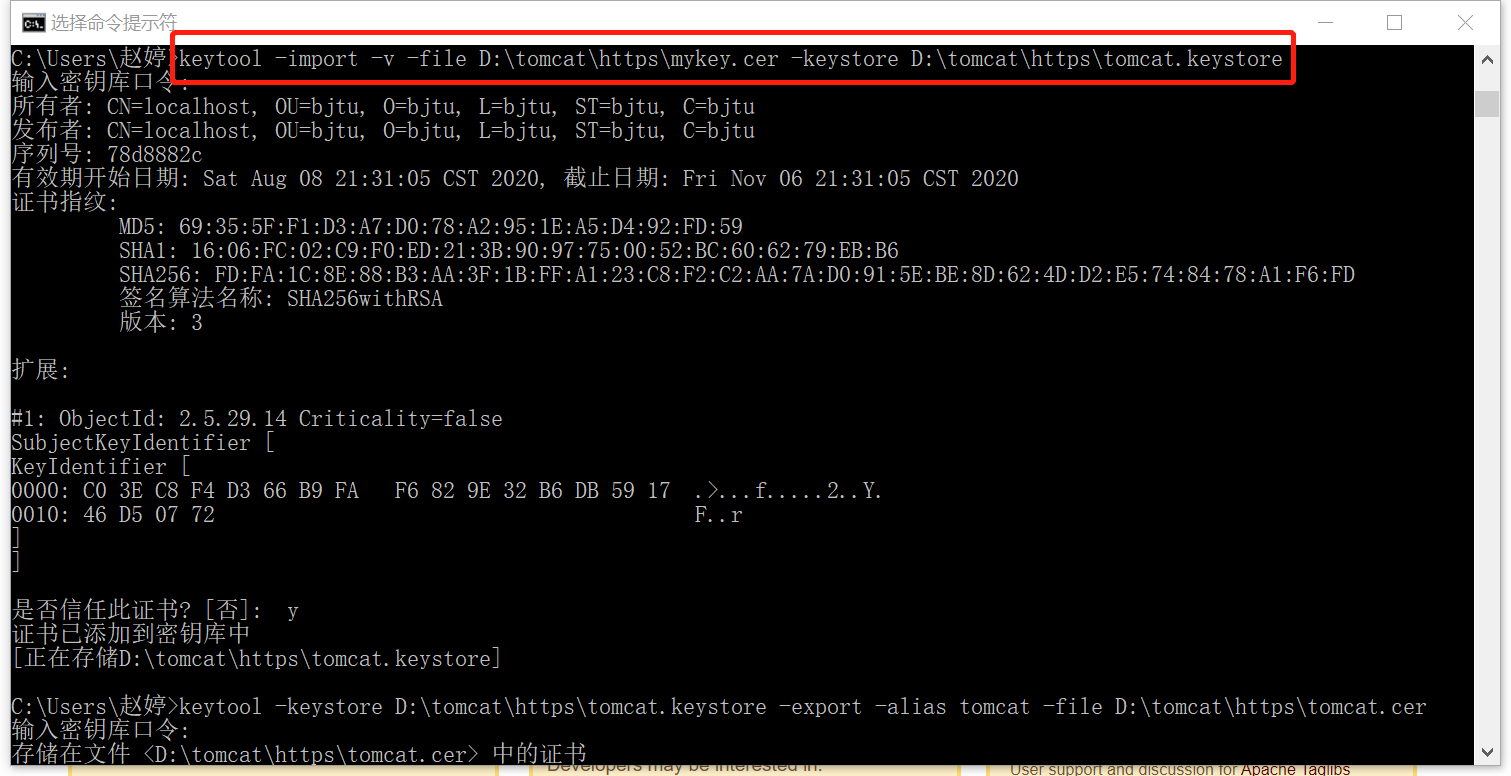


1. 服务器信任客户端证书

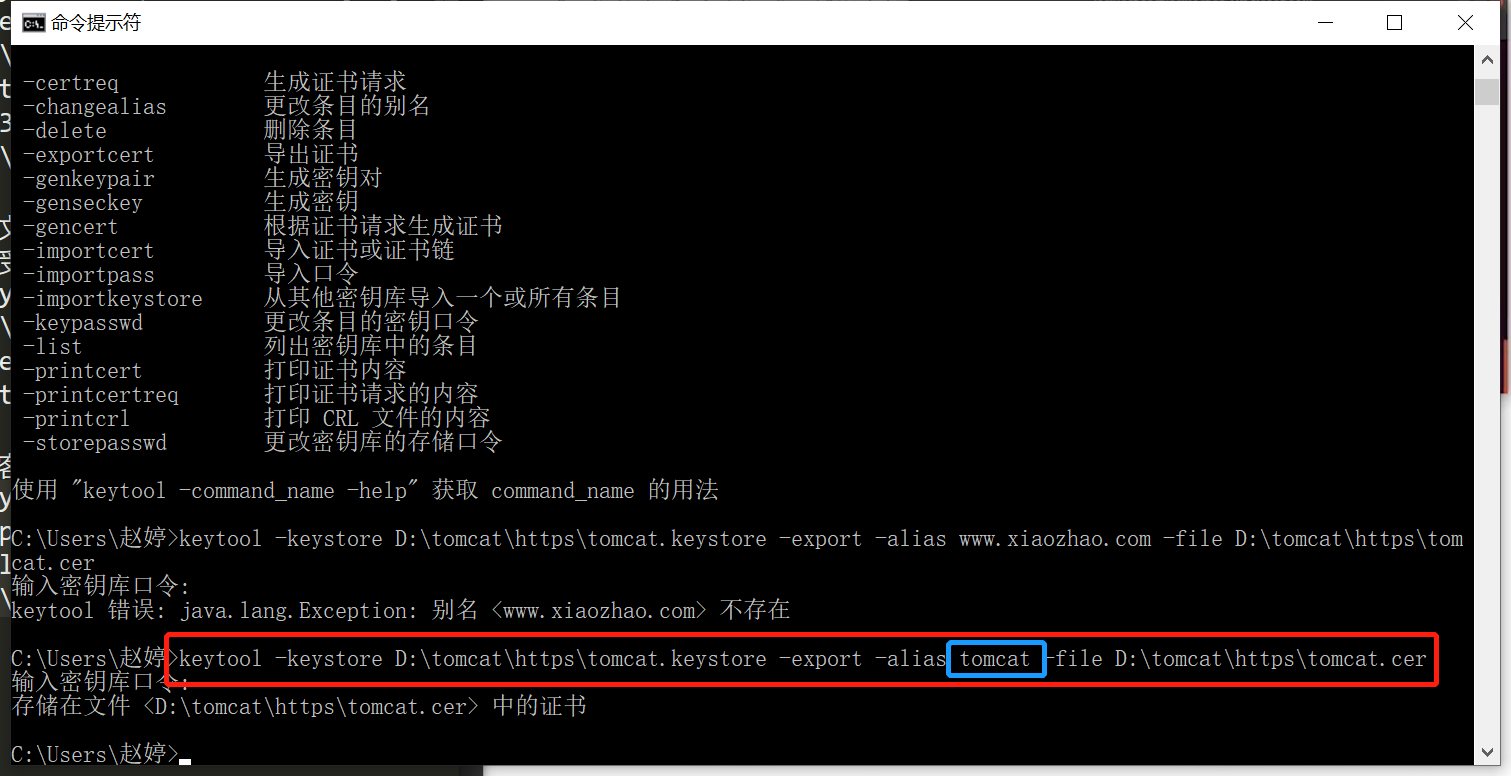
由于是双向的SSL认证，因此服务器必须要信任客户端证书，必须把客户端证书添加为服务器的信任认证，首先将客户端证书导出为一个单独的CER文件。



1. 将文件导入服务器证书库，添加为一个受信任的证书

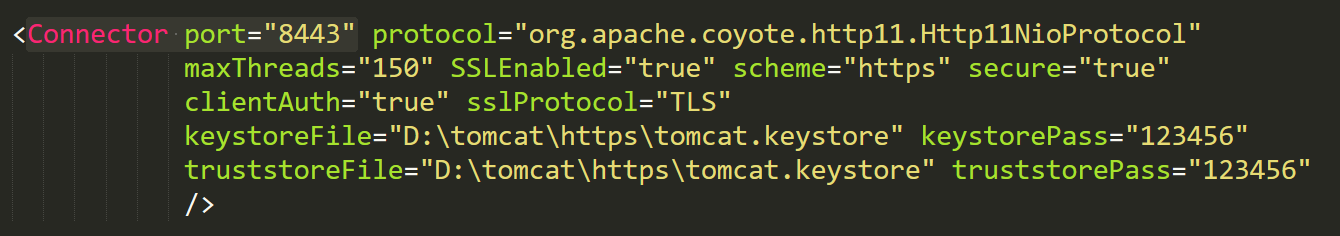


1. 让客户端信任服务端证书



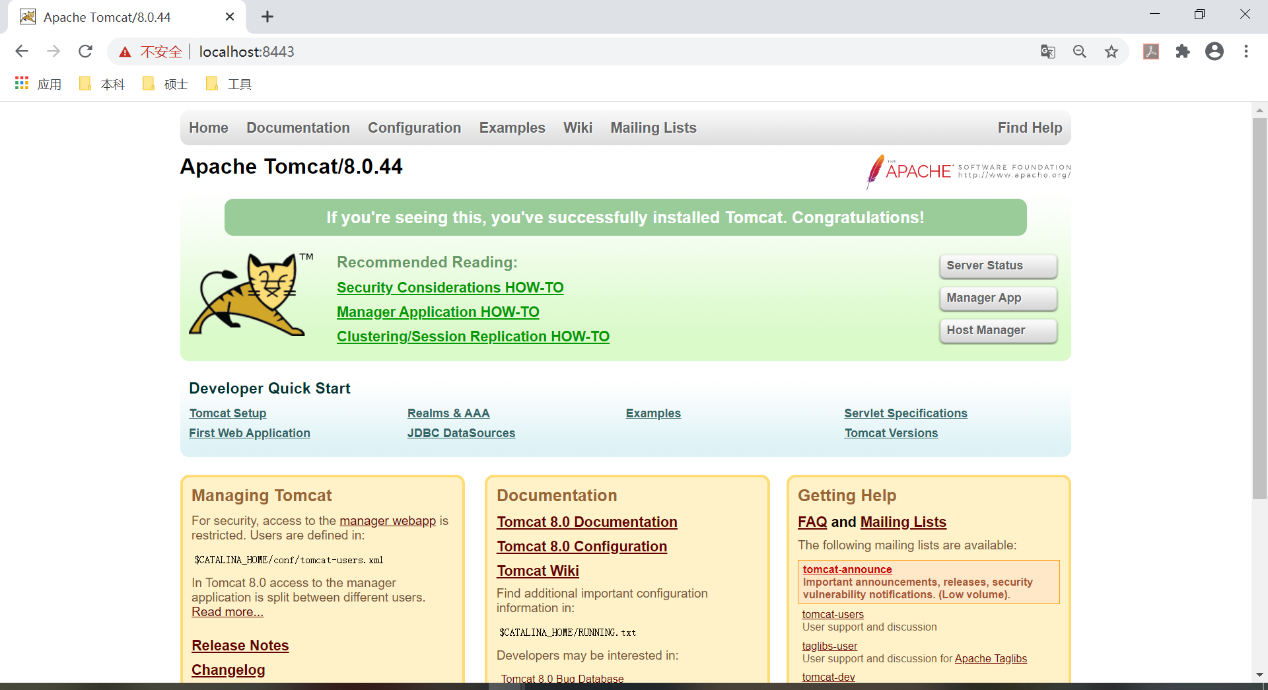
1. 配置tomcat服务器

打开tomcat根目录下的/conf/server.xml，将Connector port="8443"的配置段，修改成如下内容：



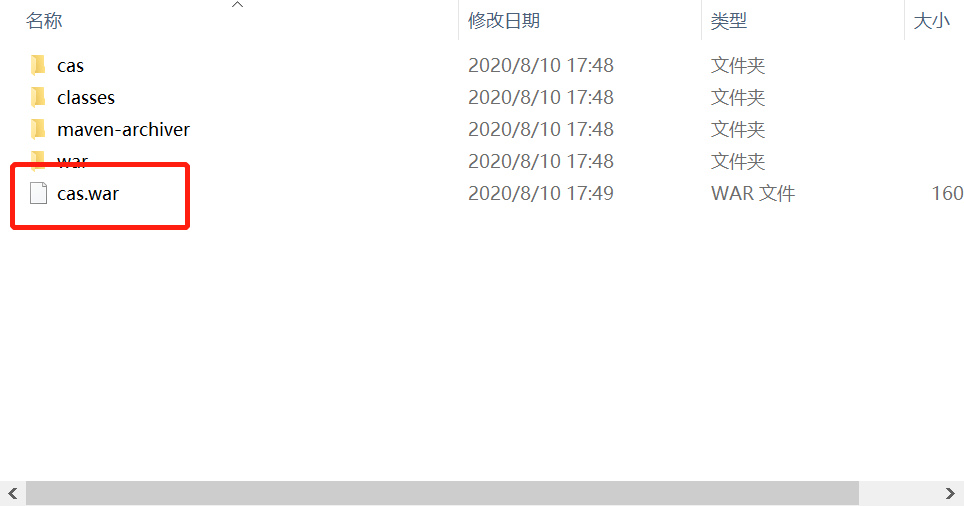
1. 测试

启动tomcat，在浏览器中导入我们生成的证书，然后输入localhost:8443，成功显示tomcat页面即成功。

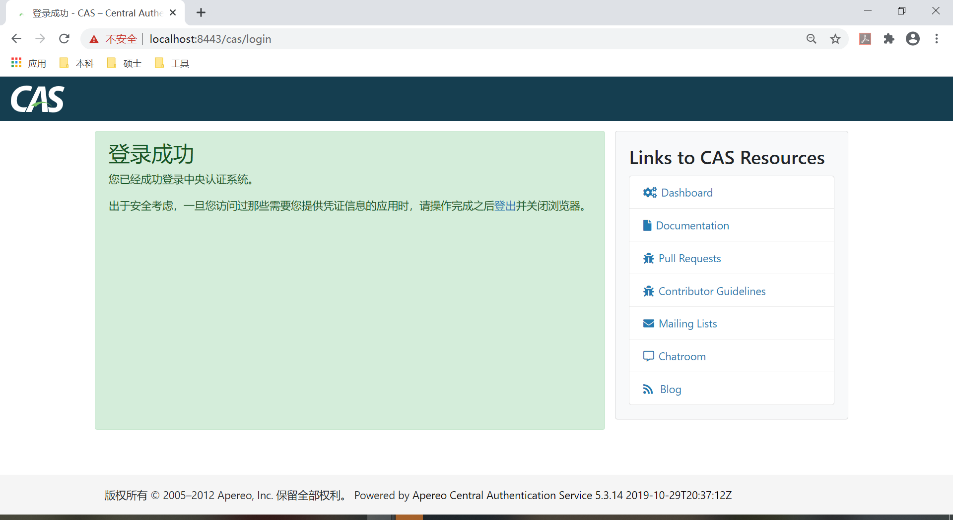
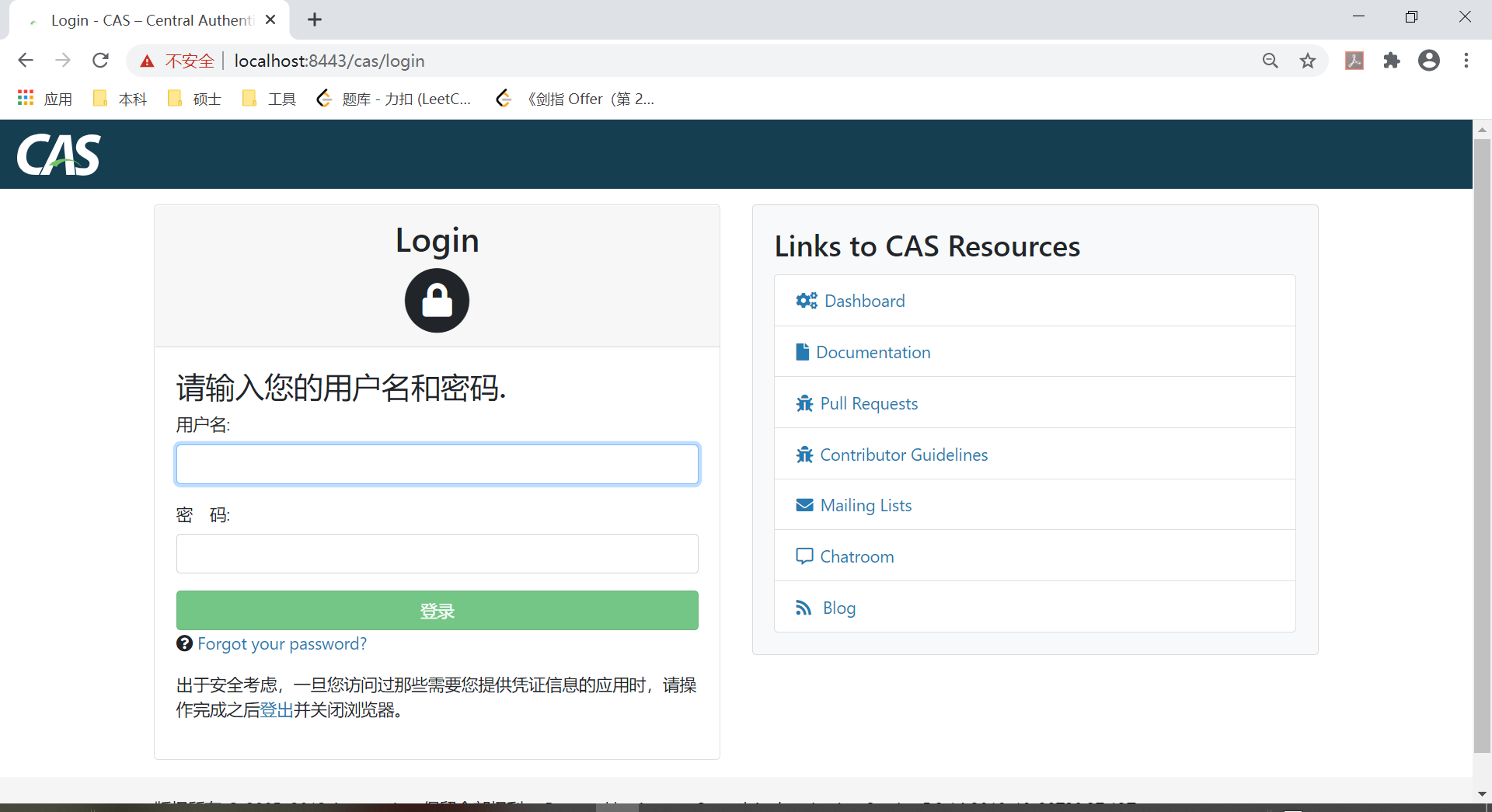


* 1. **CasServer搭建**

在git上下载cas-overlay-template（<https://github.com/apereo/cas-overlay-template>），在解压后的文件路径下执行maven命令build package，会下载依赖构建war包，在target文件下生成war包，复制cas.war到tomcat的webapp目录下，然后重启tomcat。完整项目地址：<https://github.com/smilezt/sso-cas>



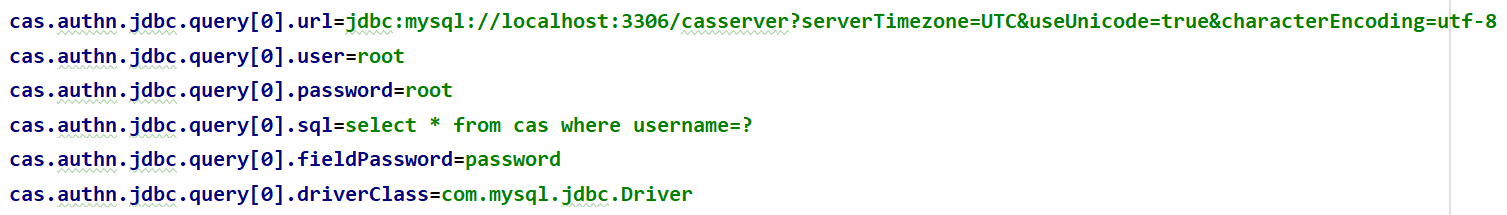
在浏览器访问<https://localhost:8443/cas/login>，输入默认的用户名和密码，即可登录CAS系统。



* 1. **配置JDBC连接数据库**

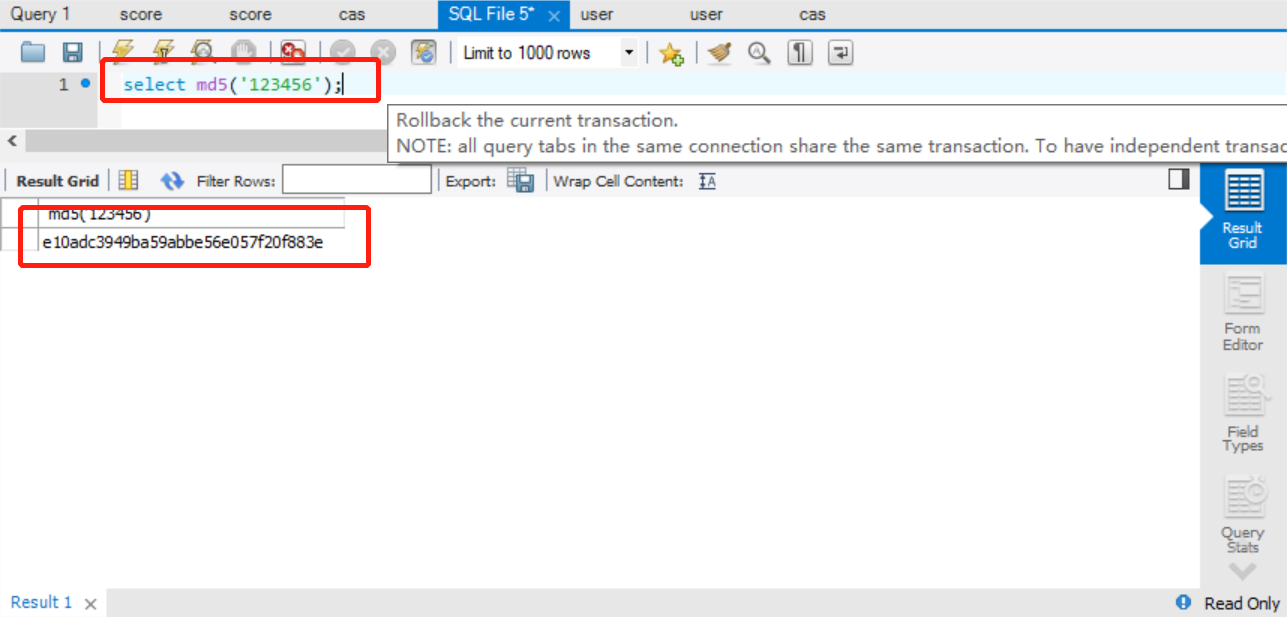
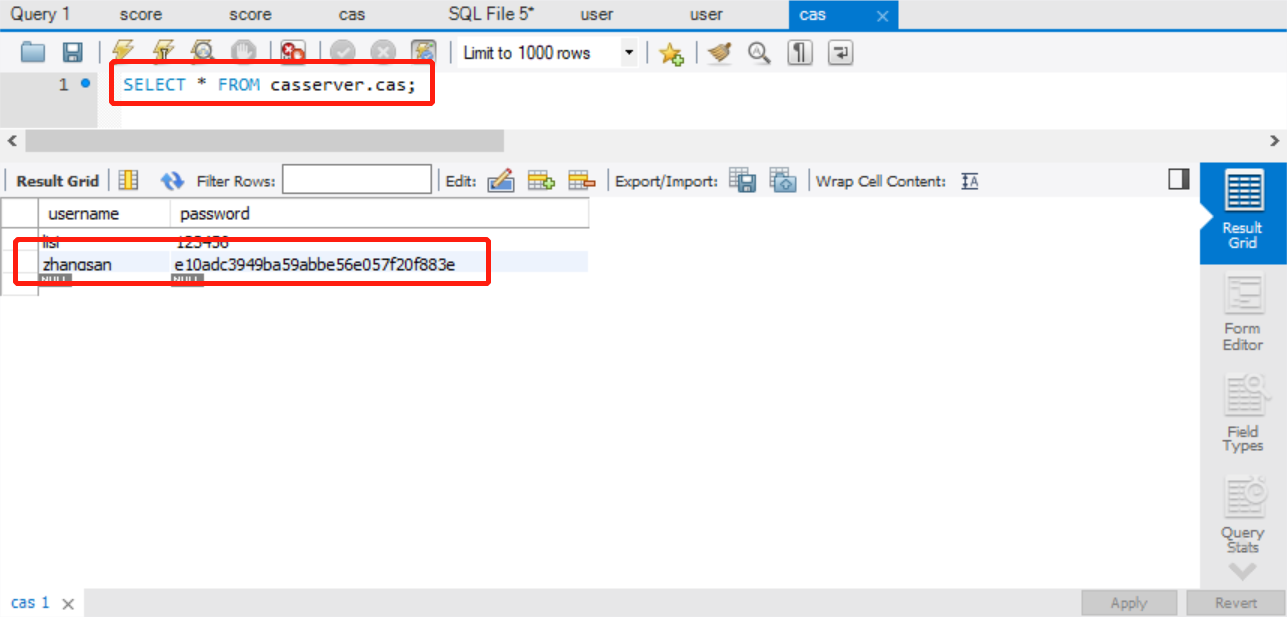
在项目中添加所需要的依赖后，在配置文件中添加如下配置，之后在数据库中建立相应的数据表插入数据后，就可以在登录时通过数据库来验证用户的登录有效性。



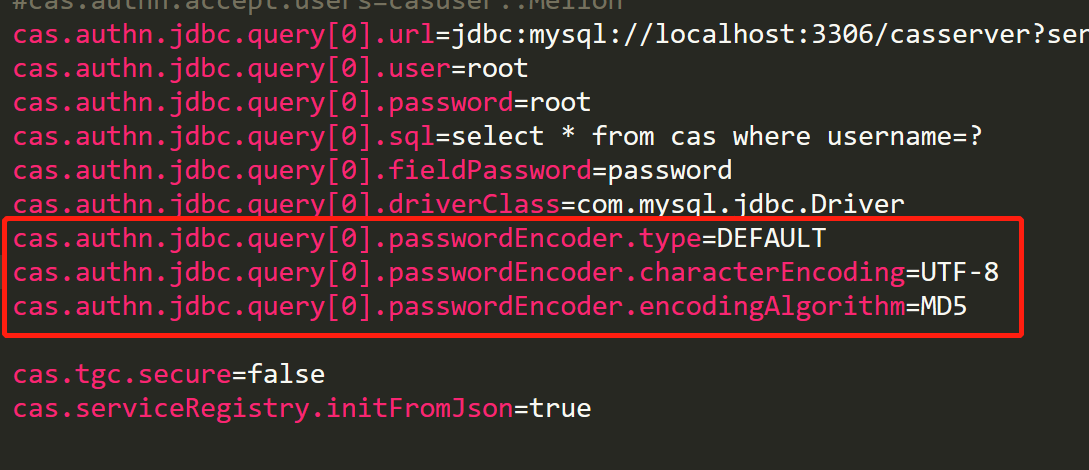


* 1. **CAS密码MD5值**

在数据库中生成密码原文的MD5值，然后存储到用户表中。

之后在CAS服务器的配置文件中增加如下配置，就可以在CAS系统中对密码进行一定的保护。



* 1. **Shiro端搭建**

使用Spring简单集成Shiro框架，搭建了一个实现登录功能的demo，由于之前介绍Shiro中主要通过Realms来实现认证和授权，因此，最核心部分就是需要写一个Realm结合CAS进行用户的认证和授权。完整项目地址：<https://github.com/smilezt/sso-shiro> 下面是ShiroCasRealm类的代码。

ShiroCasRealm.java

|  |
| --- |
| **package** com.test.shiro.realm;  **import** org.apache.shiro.authc.AuthenticationException; **import** org.apache.shiro.authc.AuthenticationInfo; **import** org.apache.shiro.authc.AuthenticationToken; **import** org.apache.shiro.authc.SimpleAuthenticationInfo; **import** org.apache.shiro.authz.AuthorizationInfo; **import** org.apache.shiro.cas.CasRealm; **import** org.apache.shiro.cas.CasToken; **import** org.apache.shiro.subject.PrincipalCollection; **import** org.apache.shiro.subject.SimplePrincipalCollection; **import** org.apache.shiro.util.CollectionUtils; **import** org.apache.shiro.util.StringUtils; **import** org.jasig.cas.client.authentication.AttributePrincipal; **import** org.jasig.cas.client.validation.Assertion; **import** org.jasig.cas.client.validation.TicketValidationException; **import** org.jasig.cas.client.validation.TicketValidator;  **import** java.util.List;  **public class** ShiroCasRealm **extends** CasRealm {   */\*\*  \* 用户身份认证  \*/* @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  *// 此时接收的AuthenticationToken对象实际上是CasToken类型* CasToken casToken = (CasToken) token; *// 现在需要返回的是CAS认证标记* **if** (casToken == **null**) { *// 如果现在没有返回token标记* **return null**; *// 当前的登录失败* }   *// CAS一定要返回给用户一个票根，所以需要取得这个票根的内容* String ticket = (String) casToken.getCredentials();  *// 需要对票根的有效性进行验证* **if** (!StringUtils.*hasText*(ticket)) { *// 票根验证失败* **return null**; *// 当前的登录失败* }   *// 如果现在票根验证的格式正确，那么需要进行票根的有效性验证* TicketValidator ticketValidator = **super**.ensureTicketValidator();   **try** {  *// 首先需要针对于票根的CAS做一个验证处理* Assertion casAssertion = ticketValidator.validate(ticket, **super**.getCasService());  *// 当验证处理完成之后，应该通过CAS取得用户信息* AttributePrincipal casPrincipal = casAssertion.getPrincipal();  String mid = casPrincipal.getName(); *// 取出当前登录的用户名  // 取出用户名之后需要将所有的相关信息（包括CAS相关信息）一起进行一个列表的创建* List principals = CollectionUtils.*asList*(mid, casPrincipal.getAttributes());  PrincipalCollection principalCollection = **new** SimplePrincipalCollection(principals,**super**.getName());  **return new** SimpleAuthenticationInfo(principalCollection, ticket);  } **catch** (TicketValidationException e) {  e.printStackTrace();  }  **return super**.doGetAuthenticationInfo(token);  }   */\*\*  \* 用户授权  \*/* @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  *// 没有权限验证体系，所以直接返回* **return super**.doGetAuthorizationInfo(principals);  } } |

1. **结果展示**

链接：https://pan.baidu.com/s/1YNWqAqZJSK8Fkn2svBkEkg

提取码：8esn

1. **总结**

使用此方式实现单点登录，优势在于使用开源的cas服务，只需要在其基础上进行修改，如页面和一些其他个性服务，不需要开发登录验证的代码，且此系统的验证流程较为完善和安全。但是正因为如此此方式不适用于集成到现有的网站中，适合于开发一个新网站。