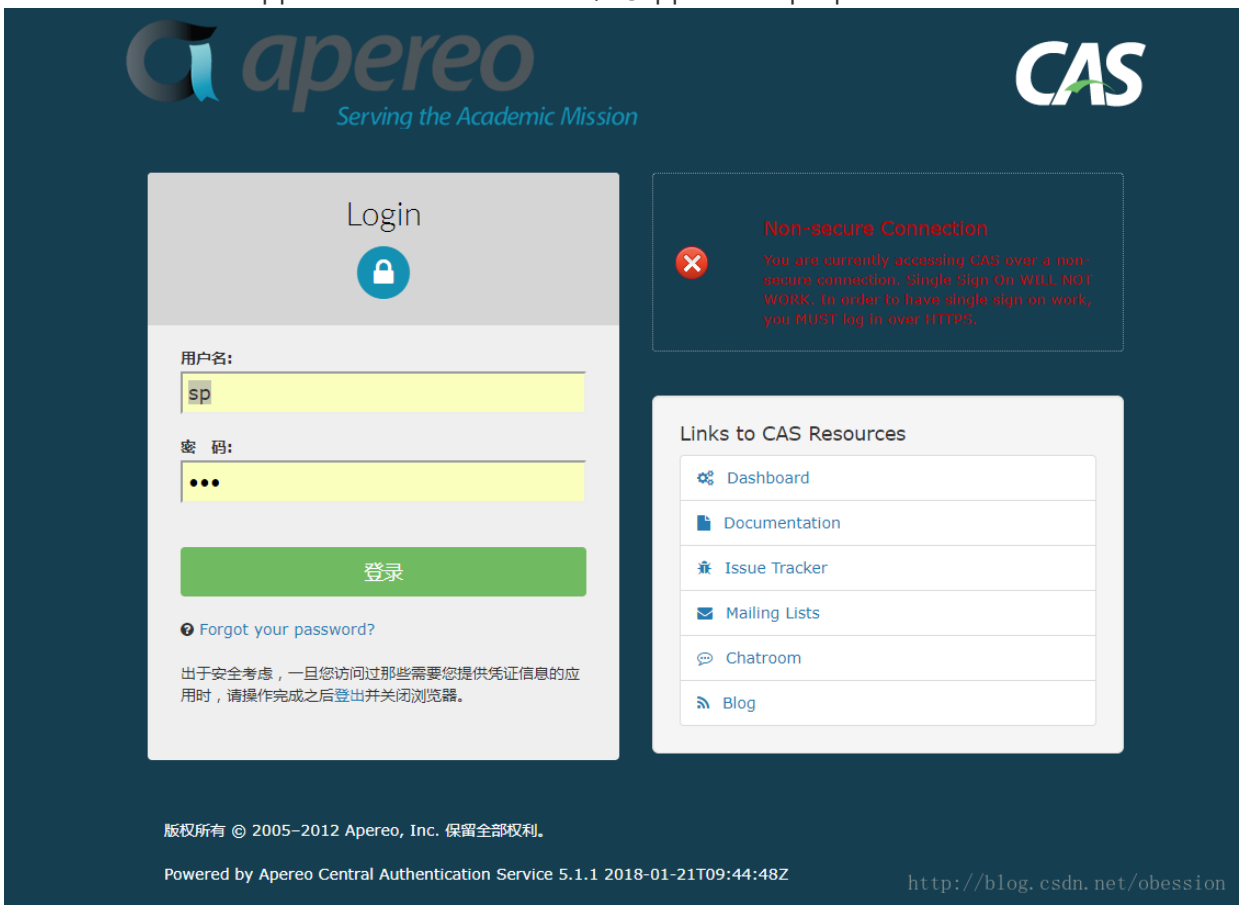
CasServer搭建

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 客户端配置官方文档：https://github.com/apereo/java-cas-client application.properties配置官方文档CAS Configuration-Properties： https://apereo.github.io/cas/5.0.x/installation/Configuration-Properties.html CAS REST-protocol认证官方文档：https://apereo.github.io/cas/5.0.x/protocol/REST-Protocol.html CAS 类说明 http://static.javadoc.io/org.apereo.cas/cas-server/5.1.0-RC1/overview-summary.html CAS 5.3.X 官网： https://apereo.github.io/cas/5.3.x/index.htm

1.服务器搭建

https://github.com/apereo/cas-overlay-template 下载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文件夹下生成cas.war 复制cas.war包到tomcat的webapp目录下，运行start.up，访问http://localhost:8080/cas/login； 可以看到，登录用户名和密码可以输入casuser和Mellon，这是初始配置文件默认的，文件位于apache-tomcat-8.5.16\webapps\cas\WEB-INF\classes下的application.properties



登录成功可以看到：

2.数据库连接

pom依赖

```
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.40</version>
            </dependency>
            <!-- https://mvnrepository.com/artifact/org.apereo.cas/cas-server-
support-jdbc -->
            <dependency>
                <groupId>org.apereo.cas</groupId>
                <artifactId>cas-server-support-jdbc</artifactId>
                <version>5.3.2</version>
            </dependency>
```

配置application.properties文件属性； **可以在 overlay 下面自己创建 src 目录创建resources,将程序 war 里面的application.properties复制到这里面,然后修改属性,这样我们再打包后会用我们的文件覆盖掉默认文件** 注释掉cas.authn.accept.users=casuser::Mellon

```
##
# CAS Authentication Credentials
#
#注释掉默认的ID, password
#cas.authn.accept.users=casuser::Mellon
```

添加jdbc认证的配置：

```
#连接端口和数据库名
cas.authn.jdbc.query[0].url=jdbc:mysql://localhost:3306/casserver?
useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&useSSL=false
cas.authn.jdbc.query[0].user=root
```

```
cas.authn.jdbc.query[0].password=qishimeiyoumima
#查询用户的信息的 sql 语句,因为是开发中密码肯定是和用户输入的不一致,所以只要根据用户名查询即可
cas.authn.jdbc.query[0].sql=select * from cas where username=?
#用户的密码在数据库表中的列名
cas.authn.jdbc.query[0].fieldPassword=password
cas.authn.jdbc.query[0].driverClass=com.mysql.jdbc.Driver

#指定过期字段，1为过期，若过期不可用
#cas.authn.jdbc.query[0].fieldExpired=expired
#为不可用字段段，1为不可用，需要修改密码
#cas.authn.jdbc.query[0].fieldDisabled=disabled

#cas.authn.jdbc.query[0].healthQuery=
#cas.authn.jdbc.query[0].isolateInternalQueries=false
#cas.authn.jdbc.query[0].failFast=true
#cas.authn.jdbc.query[0].isolationLevelName=ISOLATION_READ_COMMITTED
#cas.authn.jdbc.query[0].dialect=org.hibernate.dialect.MySQLDialect
#cas.authn.jdbc.query[0].leakThreshold=10
#cas.authn.jdbc.query[0].propagationBehaviorName=PROPAGATION_REQUIRED
#cas.authn.jdbc.query[0].batchSize=1
#cas.authn.jdbc.query[0].healthQuery=
#cas.authn.jdbc.query[0].isolateInternalQueries=false
#cas.authn.jdbc.query[0].failFast=true
#cas.authn.jdbc.query[0].isolationLevelName=ISOLATION_READ_COMMITTED
#cas.authn.jdbc.query[0].dialect=org.hibernate.dialect.MySQLDialect
#cas.authn.jdbc.query[0].leakThreshold=10
#cas.authn.jdbc.query[0].propagationBehaviorName=PROPAGATION_REQUIRED
#cas.authn.jdbc.query[0].batchSize=1
#cas.authn.jdbc.query[0].ddlAuto=create-drop
#cas.authn.jdbc.query[0].maxAgeDays=180
#cas.authn.jdbc.query[0].autocommit=false
#cas.authn.jdbc.query[0].idleTimeout=5000
# cas.authn.jdbc.query[0].credentialCriteria=
# cas.authn.jdbc.query[0].name=
# cas.authn.jdbc.query[0].order=0
# cas.authn.jdbc.query[0].dataSourceName=
# cas.authn.jdbc.query[0].dataSourceProxy=false

# cas.authn.jdbc.query[0].fieldExpired=
# cas.authn.jdbc.query[0].fieldDisabled=
# cas.authn.jdbc.query[0].principalAttributeList=sn,cn:commonName,givenName

#cas.authn.jdbc.query[0].passwordEncoder.type=DEFAULT
#cas.authn.jdbc.query[0].passwordEncoder.type=com.example.CustomPasswordEncoder
#cas.authn.jdbc.query[0].passwordEncoder.characterEncoding=UTF-8
#cas.authn.jdbc.query[0].passwordEncoder.encodingAlgorithm=MD5

#cas.authn.jdbc.query[0].passwordEncoder.secret=
```

```
#cas.authn.jdbc.query[0].passwordEncoder.strength=16

# cas.authn.jdbc.query[0].principalTransformation.suffix=
#
cas.authn.jdbc.query[0].principalTransformation.caseConversion=NONE|UPPERCASE|LOWERCASE
# cas.authn.jdbc.query[0].principalTransformation.prefix=
```

3、配置MD5认证加密

配置application.properties文件属性

MD5

```
cas.authn.jdbc.query[0].passwordEncoder.type=DEFAULT
#cas.authn.jdbc.query[0].passwordEncoder.type=com.example.CustomPasswordEncoder
cas.authn.jdbc.query[0].passwordEncoder.characterEncoding=UTF-8
cas.authn.jdbc.query[0].passwordEncoder.encodingAlgorithm=MD5
```

4、 MD5盐值次数加密

> 注意参数的名字和只 MD5校验的不一致

```
##
# 加盐MD5用户jdbc验证
##
#配置数据库连接
cas.authn.jdbc.encode[0].driverClass=com.mysql.jdbc.Driver
cas.authn.jdbc.encode[0].url=jdbc:mysql://localhost:3306/p2p?
useUnicode=true&characterEncoding=UTF-8&autoReconnect=true&useSSL=false
cas.authn.jdbc.encode[0].user=root
cas.authn.jdbc.encode[0].password=root
#加密迭代次数
cas.authn.jdbc.encode[0].numberOfIterations=1024
#如果迭代次数是随机或者存放在表中,则需要通过声明列明来代替,该列名的值可替代上面的值，但对密码加密时必须取该值进行处理
#cas.authn.jdbc.encode[0].numberOfIterationsFieldName=
#动态盐值用的字段
cas.authn.jdbc.encode[0].saltFieldName=PasswordSalt
#静态盐值
#cas.authn.jdbc.encode[0].staticSalt=.
cas.authn.jdbc.encode[0].sql=SELECT * FROM member_user WHERE name =?
#对处理盐值后的算法,注意和上面的只用 md5不一样
cas.authn.jdbc.encode[0].algorithmName=MD5
#哪个字段作为密码字段,注意这个地方和单纯查询方式不一样
cas.authn.jdbc.encode[0].passwordFieldName=Password
```

5、客户端搭建

下载实例：https://github.com/cas-projects/cas-sample-java-webapp 然后进入web.xml a.更改成你的服务器和客户端的地址：我改成了HTTP访问方式

```
    <!-- 认证拦截 -->
    <filter>
        <filter-name>CAS Authentication Filter</filter-name>
        <!--<filter-
class>org.jasig.cas.client.authentication.Saml11AuthenticationFilter</filter-
class>-->
        <filter-
class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>
        <init-param>
        <!-- 要跳转的服务器地址 -->
            <param-name>casServerLoginUrl</param-name>
           <!--
            <param-value>https://mmoayyed.unicon.net:8443/cas/login</param-value>
            -->
             <param-value>http://localhost:8080/cas/login</param-value>
        </init-param>
        <init-param>
        <!-- 客户端项目所在地址，会议service参数get提交到服务器的地址 -->
            <param-name>serverName</param-name>
            <!--
            <param-value>https://mmoayyed.unicon.net:9443</param-value>
            -->
            <param-value>http://localhost:8118</param-value>
        </init-param>
    </filter>
```

b.和ticket认证的地址

```
<!-- ticket认证 -->
    <filter>
        <filter-name>CAS Validation Filter</filter-name>
        <!--<filter-
class>org.jasig.cas.client.validation.Saml11TicketValidationFilter</filter-class>-
->
        <filter-
class>org.jasig.cas.client.validation.Cas30ProxyReceivingTicketValidationFilter</f
ilter-class>
        <init-param>
        <!-- 认证服务中心的地址 -->
            <param-name>casServerUrlPrefix</param-name>
            <param-value>http://localhost:8080/cas/</param-value>
        </init-param>
                <!-- 客户端项目所在地址，会议service参数get提交到服务器的地址 -->
```

```
        <init-param>
            <param-name>serverName</param-name>
            <param-value>http://localhost:8118</param-value>
        </init-param>
```

5.1 https 认证

因为CAS默认的HTTPS访问方式，而我们的 tomcat 是 HTTP, 所以要添加HTTP方式，要在**服务端**添加HTTP的服务 webapps\cas\WEB-INF\classes\services下的HTTPSandIMAPS-10000001文件更改正则表达式 添加HTTP

```
"serviceId" : "^(https|imaps|http)://.*",
```

然后在application.properties文件添加认证JSON服务的配置(都是**服务端**的设置)

```
#添加认证服务
cas.tgc.secure=false
cas.serviceRegistry.initFromJson=true
```

不然会报错:

然后跑起eclipse的客户端 认证成功 客户端返回成功的界面

CAS Example Java Web App

A sample web application that exercises the CAS protocol features via the Java CAS Client.

Authenticated User Id: lqy

Attributes:

	Attributes
Key	**Value**
isFromNewLogin	true
authenticationDate	2018-02-05T17:55:20.064+08:00[Asia/Shanghai]
authenticationMethod	QueryDatabaseAuthenticationHandler
successfulAuthenticationHandlers	QueryDatabaseAuthenticationHandler
longTermAuthenticationRequestTokenUsed	false

6、实现两个客户端的单点登录

从eclipse导出客户端的war包部署在一个新的tomcat下，不把两个客户端放在eclipse下跑的原因是，有可能会共享一个session 不便于测试单点登录和全登出的效果。 这里服务器和客户端1，客户端2 的TOMCAT的shutdown,http,ajp的端口一定都设成不一样的，才能都在localhost下跑起来。 实现效果，客户端1进行了ID,密码验证后 ，进入客户端2不需要再输入账号密码。

7、全登出的配置

参考：http://blog.csdn.net/u010475041/article/details/78234027 和http://blog.csdn.net/u010475041/article/details/78018100 首先全登出的原理是服务端选择登出后，先清除自己的session，再依次通知客户端清除客户端自己的session。 CAS是有全登出的配置，且默认打开。官方文档： https://apereo.github.io/cas/5.1.x/installation/Logout-Single-Signout.html

service： 服务端通过service来允许每一个客户端接入认证，如HTTPSandIMAPS-10000001.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^(https|imaps)://.*",
  "name" : "HTTPS and IMAPS",
  "id" : 10000001,
  "description" : "This service definition authorizes all application urls that
support HTTPS and IMAPS protocols.",
  "evaluationOrder" : 10000
}
```

serviceId就是允许去服务端认证的客户端的IP，”^(https|imaps)://.*”，因为没有HTTP，所以通过HTTP协议来访问服务端的就没有权限。所以需要对每一个客户端单独配置一个service。

实现Logout and Single Logout (SLO)，需要： a.首先配置客户端的web.xml的登出拦截器。

```
<!-- 登出拦截器 -->
    <filter>
        <filter-name>CAS Single Sign Out Filter</filter-name>
        <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-
class>
        <init-param>
            <param-name>casServerUrlPrefix</param-name>
            <param-value>http://localhost:8080/cas</param-value>
        </init-param>
    </filter>
```

b.然后在服务端添加服务，在apache-tomcat-8.5.16\webapps\cas\WEB-INF\classes\services下有几个json格式的service,要保证json的service被加载，需要在application.properties文件添加认证JSON服务的配置添加认证服务

```
cas.serviceRegistry.initFromJson=true
```

c.默认的两个json文件：Apereo-10000002.json和HTTPSandIMAPS-10000001.json，删除。我实验的是两个客户端，地址分别是：http://localhost:8118/cas-sample-java-webapp和http://localhost:8208/cas-sample-java-webapp。所以配置的两个service文件如下：

```
{
  "@class": "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "http://localhost:8118/cas-sample-java-webapp.*",
  "name": "Clientdemo1",
  "id": 10000001,
   "description" : "Clientdemo1",
  "evaluationOrder" : 10000,
   "logoutType" : "BACK_CHANNEL",
  "logoutUrl" : "http://localhost:8118/cas-sample-java-webapp/logout.jsp"
}
```

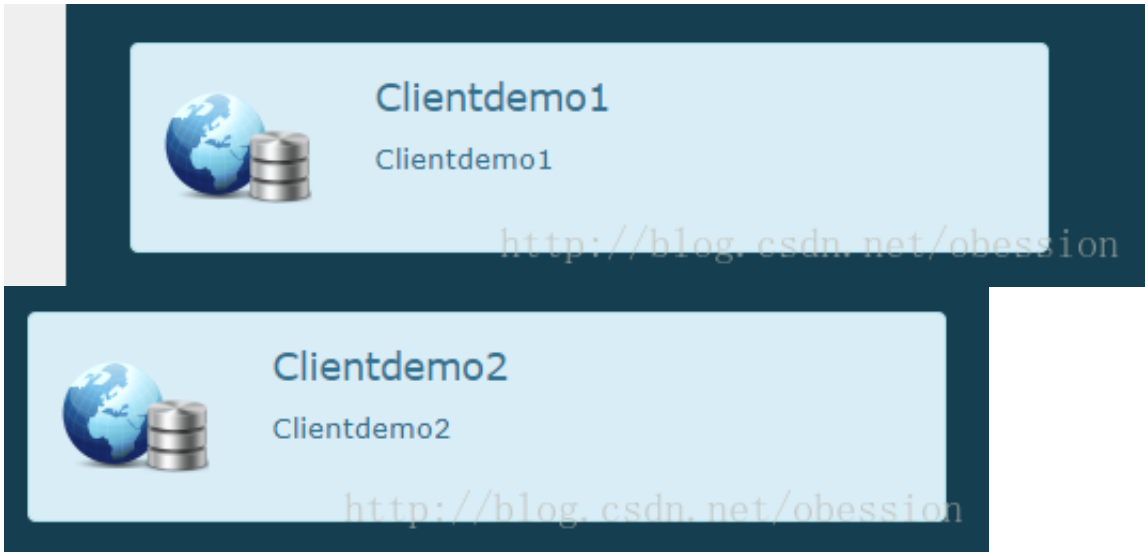
和

```
{
  "@class": "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "http://localhost:8208/cas-sample-java-webapp.*",
  "name": "Clientdemo2",
  "id": 10000002,
  "description" : "Clientdemo2",
  "evaluationOrder" : 10001,
  "logoutType" : "BACK_CHANNEL",
  "logoutUrl" : "http://localhost:8208/cas-sample-java-webapp/logout.jsp"
}
```

d.另外注意：json文件名字规则为name−name−{id}.json,id必须为json文件内容id一致 json文件解释：

@class：必须为org.apereo.cas.services.RegisteredService的实现类，对其他属性进行一个json反射对象，常用的有RegexRegisteredService，匹配策略为id的正则表达式 serviceId：唯一的服务id name： 服务名称，会显示在默认登录页 id：全局唯一标志 description：服务描述，会显示在默认登录页 evaluationOrder： 匹配争取时的执行循序

如果没问题，客户端1和客户端2的认证界面都能看到配置到的service：



实现效果： 并且在服务端登出后，客户端1和客户端2会自动登出。

8 java导入证书

> 将d:\software\AKAZAM-Mail.cer中的证书导入到cacerts这个文件中,然后替换到java自带的文件

```
keytool -import -alias tomcat -keystore cacerts -file d:\software\AKAZAM-Mail.cer
```