CAS Server 4.0二次开发说明文档

1. 环境搭建 2

2. 拓展后最终目录结构 7

3. 修改CAS Server支持HTTP协议 8

4. 修改CAS服务器端登录认证方式为数据库认证 9

4.1. 在pom中添加如下的依赖 9

4.2. 创建ValidUserQueryDBAuthenticationHandler 10

4.3. 修改deployerConfigContext.xml 10

5. 登录连续错误三次以上出现验证码校验 11

5.1. 添加kaptcha验证码依赖库 11

5.2. 配置web.xml 12

5.3. UsernamePasswordCaptchaCredentia 12

5.4. 修改cas登录页，添加验证码字段 12

5.5. 添加验证码校验 13

5.6. 自定义异常处理Handler 统计错误次数 16

5.7. 修改配置cas-servlet.xml 16

5.8. 修改配置login-webflow.xml 17

6. CAS服务器端返回多个属性字段 17

6.1. 创建MultipleAttributeUserDao模拟用户自定义属性 18

6.2. 修改deployerConfigContext.xml 18

6.3. 修改casServiceValidationSuccess.jsp 18

7. CAS Server集群部署时Ticket同步 21

7.1. 添加redis依赖 21

7.2. 创建RedisTicketRegistry 22

7.3. 创建TicketRedisTemplate 22

7.4. 修改ticketRegistry.xml 23

7.5. 修改web.xml 24

8. CAS Server登录页定制 24

8.1. 拷贝资源文件 24

8.2. 修改casLoginView.jsp 25

8.3. 基于theme方式 25

9. 修改CAS Server为移动端设备提供REST登录服务 25

9.1. 添加依赖 26

9.2. 修改webx.xml 27

9.3. 请求票据授权 27

9.4. 请求服务授权 28

9.5. Java Client测试 29

在生产环境中，官方提供的CAS Server功能远远不能满足我们的需求。这个时候我们就需要基于源码进行二次开发，以满足我们的线上使用。这里主要是基于CAS服务器端。本文档主要描述目前在CAS Server 4.0版本基础上做了哪些改进。

# 环境搭建

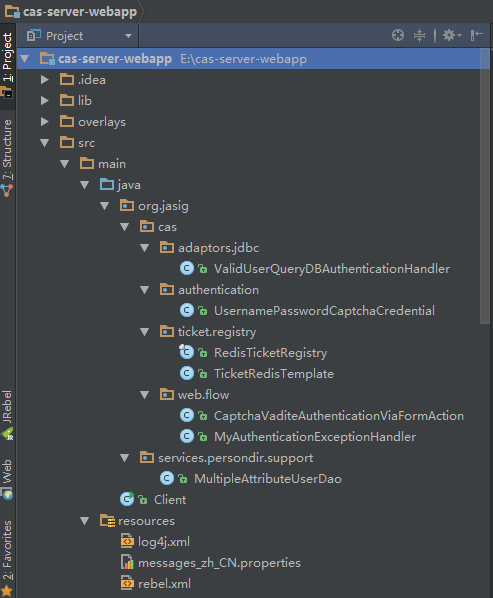
这里为了方便起见采用maven overlays特性（通路径下的同名文件会使用最新的资源文件替换已有war目录中下的资源文件）搭建，因此这里不用将cas-server-webapp模块的所有源码导入到工程里，只需将需要修改的文件按源码里面的目录层次结构排列，最终编译构建打包时会自动添加新特性或替换原有文件，这是一件很方便的事，修改和添加了哪些文件一目了然。

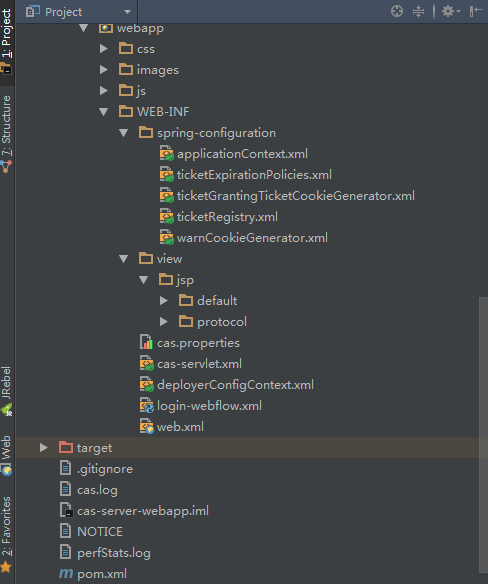
以下列出了本文档所涉及到所有拓展功能及最终编译打包所需的maven依赖。

pom.xml

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>net.aimeizi</groupId>  <artifactId>cas-server-webapp</artifactId>  <version>4.0.0</version>  <packaging>war</packaging>  <name>Jasig CAS Web Application</name>  <description>Jasig CAS Server Web Application</description>  <properties>  <!-- 文件拷贝时的编码 -->  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <!-- 编译时的编码 -->  <maven.compiler.encoding>UTF-8</maven.compiler.encoding>  </properties>  <dependencies>  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-webapp</artifactId>  <version>4.0.0</version>  <type>war</type>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-core</artifactId>  <version>4.0.0</version>  <type>test-jar</type>  </dependency>  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-webapp-support</artifactId>  <version>4.0.0</version>  <scope>compile</scope>  <exclusions>  <exclusion>  <groupId>\*</groupId>  <artifactId>\*</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>compile</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.1.2</version>  <type>jar</type>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>1.1.2</version>  <type>jar</type>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.18</version>  </dependency>  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-support-jdbc</artifactId>  <version>4.0.0</version>  </dependency>  <!-- cas server restlet -->  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-integration-restlet</artifactId>  <version>4.0.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet</artifactId>  <version>2.1.0</version>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.spring</artifactId>  <version>2.1.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-asm</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-expression</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.servlet</artifactId>  <version>2.1.0</version>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.slf4j</artifactId>  <version>2.1.0</version>  </dependency>  <!-- cas server restlet -->  <!-- redis -->  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-redis</artifactId>  <version>1.7.1.RELEASE</version>  <exclusions>  <exclusion>  <groupId>\*</groupId>  <artifactId>\*</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-pool2</artifactId>  <version>2.4.2</version>  </dependency>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.8.1</version>  </dependency>  <dependency>  <groupId>com.google.code.kaptcha</groupId>  <artifactId>kaptcha</artifactId>  <version>2.3.2</version>  <!--<scope>system</scope>-->  <!--<systemPath>${basedir}/lib/kaptcha-2.3.2.jar</systemPath>-->  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.4</version>  </dependency>  <!-- shiro crypto -->  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-core</artifactId>  <version>1.2.2</version>  </dependency>  <dependency>  <groupId>commons-httpclient</groupId>  <artifactId>commons-httpclient</artifactId>  <version>3.1</version>  </dependency>  </dependencies>  <build>  <finalName>cas</finalName>  <plugins>  </plugins>  </build>  </project> |

# 拓展后最终目录结构





# 修改CAS Server支持HTTP协议

修改src\main\webapp\WEB-INF\spring-configuration\ticketGrantingTicketCookieGenerator.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:p="http://www.springframework.org/schema/p"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  <!-- p:cookieSecure="true" 表示开启https协议,cookieSecure设置为false关闭https协议 -->  <bean id="ticketGrantingTicketCookieGenerator" class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"  p:cookieSecure="false"  p:cookieMaxAge="3600"  p:cookieName="CASTGC"  p:rememberMeMaxAge="3600"  p:cookiePath="/cas"/>  </beans> |

修改src\main\webapp\WEB-INF\spring-configuration\warnCookieGenerator.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:p="http://www.springframework.org/schema/p"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  <!-- p:cookieSecure="true" 表示开启https协议,cookieSecure设置为false关闭https协议 -->  <bean id="warnCookieGenerator" class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"  p:cookieSecure="false"  p:cookieMaxAge="-1"  p:cookieName="CASPRIVACY"  p:cookiePath="/cas"/>  </beans> |

# 修改CAS服务器端登录认证方式为数据库认证

Cas Server默认提供简单用户密码认证即（casuser/Mellon，该部分内容在src\main\webapp\WEB-INF\deployerConfigContext.xml文件中可以看到。），显然这不能满足我们的线上环境。在此我们需要修改使其支持基于数据库的用户名密码校验。要想使用数据库校验，需要为cas server提供相应的jdbc依赖。

## 在pom中添加如下的依赖

|  |
| --- |
| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.18</version>  </dependency>  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-support-jdbc</artifactId>  <version>4.0.0</version>  </dependency> |

## 创建ValidUserQueryDBAuthenticationHandler

在src/main/java目录下创建org.jasig.cas.adaptors.jdbc包，在该包下创建ValidUserQueryDBAuthenticationHandler.java文件。因内容较多，这里不再列出，具体请查看源码

## 修改deployerConfigContext.xml

|  |
| --- |
| <!-- 数据源 -->  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">  <property name="driverClassName" value="${dataSource.driver}"></property>  <property name="url" value="${dataSource.url}"/>  <property name="username" value="${dataSource.username}"/>  <property name="password" value="${dataSource.password}"/>  <property name="maxActive" value="${dataSource.maxActive}"/>  <property name="initialSize" value="${dataSource.initialSize}"/>  <property name="maxWait" value="${dataSource.maxWait}"/>  <property name="minIdle" value="${dataSource.minIdle}"/>  <property name="timeBetweenEvictionRunsMillis" value="${dataSource.timeBetweenEvictionRunsMillis}"/>  <property name="minEvictableIdleTimeMillis" value="${dataSource.minEvictableIdleTimeMillis}"/>  <property name="validationQuery" value="${dataSource.validationQuery}"/>  <property name="testWhileIdle" value="${dataSource.testWhileIdle}"/>  <property name="testOnBorrow" value="${dataSource.testOnBorrow}"/>  <property name="testOnReturn" value="${dataSource.testOnReturn}"/>  <property name="maxOpenPreparedStatements" value="${dataSource.maxOpenPreparedStatements}"/>  <property name="removeAbandoned" value="${dataSource.removeAbandoned}"/>  <property name="removeAbandonedTimeout" value="${dataSource.removeAbandonedTimeout}"/>  <property name="logAbandoned" value="${dataSource.logAbandoned}"/>  </bean>  <!-- 密码加密策略 -->  <bean id="MD5PasswordEncoder" class="org.jasig.cas.authentication.handler.DefaultPasswordEncoder"  c:encodingAlgorithm="MD5" p:characterEncoding="UTF-8"/>  <!-- 自定义基于数据库认证策略 -->  <bean id="primaryAuthenticationHandler" class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler">  <property name="dataSource" ref="dataSource"/>  <property name="sql" value="${auth.sql}"/>  <property name="passwordEncoder" ref="MD5PasswordEncoder"/>  </bean> |

这里将数据源配置中的属性放到了cas.properties中

# 登录连续错误三次以上出现验证码校验

为了防止cas server被恶意登录及防暴力登录。这里采用验证码校验。为了良好的用户体验，设定当连续登录错误次数达到3次及以上需验证验证码登录。

这里使用开源的kaptcha来做验证码

## 添加kaptcha验证码依赖库

<dependency>

<groupId>com.google.code.kaptcha</groupId>

<artifactId>kaptcha</artifactId>

<version>2.3.2</version>

<!--<scope>system</scope>-->

<!--<systemPath>${basedir}/lib/kaptcha-2.3.2.jar</systemPath>-->

</dependency>

该库并没有上传到maven中央仓库，因此我们需要自行发布到私服，或者使用上面的scope结合systemPath引用，使用这种方式时，在打包的时候不会打到war包里面，因此为了方便起见建议下载下来自行上传到maven私服或安装到本地仓库。

使用下面的命令即可安装到本地仓库

mvn install:install-file -DgroupId=com.google.code.kaptcha -DartifactId=kaptcha -Dversion=2.3.2 -Dfile=E:\cas-4.0.0\cas-server-webapp\lib\kaptcha-2.3.2.jar -Dpackaging=jar -DgeneratePom=true

## 配置web.xml

在src\main\webapp\WEB-INF\web.xml中添加如下内容

|  |
| --- |
| <!-- Captcha -->  <servlet>  <servlet-name>Kaptcha</servlet-name>  <servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>  <init-param>  <param-name>kaptcha.border</param-name>  <param-value>no</param-value>  </init-param>  <init-param>  <param-name>kaptcha.textproducer.char.space</param-name>  <param-value>5</param-value>  </init-param>  <init-param>  <param-name>kaptcha.textproducer.char.length</param-name>  <param-value>5</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>Kaptcha</servlet-name>  <url-pattern>/captcha.jpg</url-pattern>  </servlet-mapping> |

## UsernamePasswordCaptchaCredentia

创建UsernamePasswordCaptchaCredential继承自org.jasig.cas.authentication.RememberMeUsernamePasswordCredential

主要是拓展了一个验证字段captcha.

@NotNull

@Size(min = 1, message = "required.captcha")

private String captcha;

## 修改cas登录页，添加验证码字段

修改src\main\webapp\WEB-INF\view\jsp\default\ui\casLoginView.jsp添加验证码及记住密码字段

|  |
| --- |
| <div class="re\_password">  <!--  <p><a href="#">忘记密码？点击重置</a></p>  -->  <div class="remember">  <span>记住登录状态</span>  <div class="rem\_btn">  <img src="images/rem\_btn.png">  </div>  <input type="hidden" name="rememberMe" id="rememberMe" value="true"/>  </div>  </div>  <c:if test="${not empty count && count >= 3}">  <div class="verify">  <div class="verify\_box">  <spring:message code="screen.welcome.label.captcha.accesskey" var="authcaptchaAccessKey" />  <input class="input\_box" type="text" placeholder="请输入验证码" maxlength="10" name="captcha">  <input type="hidden" name="showCaptcha" value="true"/>  </div>  <div class="ver\_img">  <img alt="<spring:message code="required.captcha" />"  onclick="this.src='captcha.jpg?'+Math.random()" src="captcha.jpg">  </div>  </div>  </c:if> |

注意表单中的name字段与UsernamePasswordCaptchaCredential中的字段对应。这里在验证码这块使用了c标签判断了错误次数count 。该count在下文会有交代。

## 添加验证码校验

在org.jasig.cas.web.flow包下创建CaptchaVaditeAuthenticationViaFormAction并继承自AuthenticationViaFormAction

其具体内容如下：

|  |
| --- |
| package org.jasig.cas.web.flow;  import org.jasig.cas.authentication.Credential;  import org.jasig.cas.authentication.UsernamePasswordCaptchaCredential;  import org.jasig.cas.web.support.WebUtils;  import org.springframework.binding.message.MessageBuilder;  import org.springframework.binding.message.MessageContext;  import org.springframework.util.StringUtils;  import org.springframework.webflow.execution.RequestContext;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpSession;  /\*\*  \* 验证码校验  \* Created by fengjing on 2016/4/27.  \*/  public class CaptchaVaditeAuthenticationViaFormAction extends AuthenticationViaFormAction {  private static final String CAPTCHA\_REQUIRED\_MSG = "required.captcha";// 验证码必填  private static final String CAPTCHA\_ERROR\_MSG = "error.captcha";// 验证码不正确  public final String validate(final RequestContext context, final Credential credentials,  final MessageContext messageContext) throws Exception {  final HttpServletRequest request = WebUtils.getHttpServletRequest(context);  HttpSession session = request.getSession();  // 用户输入错误次数累计，次数大于3次则显示验证码  int count;  try {  count = (Integer) context.getFlowScope().get("count");  } catch (Exception e) {  count = 0;  }  // 获取用户名密码  String username = request.getParameter("username");  String password = request.getParameter("password");  // 用户名非空验证  if (StringUtils.isEmpty(username)) {  populateErrorsInstance("required.username", messageContext);  count++;  context.getFlowScope().put("count", count);  return "error";  }  // 密码非空验证  if (StringUtils.isEmpty(password)) {  populateErrorsInstance("required.password", messageContext);  count++;  context.getFlowScope().put("count", count);  return "error";  }  // 验证码判断  String showCaptcha = request.getParameter("showCaptcha");  // 判断验证码是否显示  if (StringUtils.hasText(showCaptcha)) {  // 从session中获取验证码  String authcode = (String) session.getAttribute(com.google.code.kaptcha.Constants.KAPTCHA\_SESSION\_KEY);  session.removeAttribute(com.google.code.kaptcha.Constants.KAPTCHA\_SESSION\_KEY);  UsernamePasswordCaptchaCredential upc = (UsernamePasswordCaptchaCredential) credentials;  // 获取用户输入的验证码  String submitAuthcode = upc.getCaptcha();  // 验证码非空判断  if (!StringUtils.hasText(submitAuthcode) || !StringUtils.hasText(authcode)) {  populateErrorsInstance(CAPTCHA\_REQUIRED\_MSG, messageContext);  count++;  context.getFlowScope().put("count", count);  return "error";  }  // 验证码正确性判断  if (submitAuthcode.equals(authcode)) {  return "success";  } else {  populateErrorsInstance(CAPTCHA\_ERROR\_MSG, messageContext);  count++;  context.getFlowScope().put("count", count);  return "error";  }  }  return "success";  }  private void populateErrorsInstance(final String errorCode, final MessageContext messageContext) {  try {  messageContext.addMessage(new MessageBuilder().error().code(errorCode).defaultText(errorCode).build());  } catch (final Exception fe) {  logger.error(fe.getMessage(), fe);  }  }  } |

## 自定义异常处理Handler 统计错误次数

在org.jasig.cas.web.flow包下创建MyAuthenticationExceptionHandler。重写handle方法，着重在这里处理计算错误次数

|  |
| --- |
| public String handle(final RequestContext context, final AuthenticationException e, final MessageContext messageContext) {  int count;  try {  count = (Integer) context.getFlowScope().get("count");  } catch (Exception ex) {  count = 0;  }  if (e != null) {  for (final Class<? extends Exception> kind : this.errors) {  for (final Class<? extends Exception> handlerError : e.getHandlerErrors().values()) {  if (handlerError != null && handlerError.equals(kind)) {  final String messageCode = this.messageBundlePrefix + handlerError.getSimpleName();  messageContext.addMessage(new MessageBuilder().error().code(messageCode).build());  count++;  context.getFlowScope().put("count", count);  return handlerError.getSimpleName();  }  }  }  }  final String messageCode = this.messageBundlePrefix + UNKNOWN;  logger.trace("Unable to translate handler errors of the authentication exception {}. Returning {} by default...", e, messageCode);  messageContext.addMessage(new MessageBuilder().error().code(messageCode).build());  count++;  context.getFlowScope().put("count", count);  return UNKNOWN;  } |

## 修改配置cas-servlet.xml

修改src\main\webapp\WEB-INF\cas-servlet.xml

|  |
| --- |
| <!-- 自定义验证码校验的CaptchaVaditeAuthenticationViaFormAction -->  <bean id="authenticationViaFormAction" class="org.jasig.cas.web.flow.CaptchaVaditeAuthenticationViaFormAction"  p:centralAuthenticationService-ref="centralAuthenticationService"  p:warnCookieGenerator-ref="warnCookieGenerator"  p:ticketRegistry-ref="ticketRegistry"/>  <!-- 自定义异常处理Handler 统计错误次数 -->  <bean id="authenticationExceptionHandler" class="org.jasig.cas.web.flow.MyAuthenticationExceptionHandler"/> |

配置在上文提到的CaptchaVaditeAuthenticationViaFormAction及MyAuthenticationExceptionHandler

## 修改配置login-webflow.xml

修改src\main\webapp\WEB-INF\login-webflow.xml文件，该文件采用了spring-web-flow.在这里定义描述了整个cas server的登录校验认证流程。接下来，需要在登录属性表单中添加验证码字段及添加修改有关涉及到验证码校验的流程

替换原来的credential配置为UsernamePasswordCaptchaCredential

|  |
| --- |
| <!--<var name="credential" class="org.jasig.cas.authentication.UsernamePasswordCredential" />-->  <!-- 添加用于接收前台表单验证码字段captcha -->  <var name="credential" class="org.jasig.cas.authentication.UsernamePasswordCaptchaCredential"/> |

在<view-state id="viewLoginForm" view="casLoginView" model="credential">中添加验证码字段

|  |
| --- |
| <view-state id="viewLoginForm" view="casLoginView" model="credential">  <binder>  <binding property="username"/>  <binding property="password"/>  <!-- 前台添加表单添加验证码字段captcha -->  <binding property="captcha"/>  <!-- rememberMe -->  <binding property="rememberMe"/>  </binder>  .....  </view-state> |

# CAS服务器端返回多个属性字段

通常情况下，cas server认证成功后返回的属性字段很少，但在很多场合下登录后需要获取到许多字段属性。这时候我们就需要个性化来定制cas server返回的字段属性了。这里为了方便演示期间，创建了一个MultipleAttributeUserDao类，并重写了IPersonAttributes方法，在这里模拟了一些字段属性。在生产环境时，可将该部分代码替换，根据业务需求换成从数据库中查询的业务逻辑代码。

## 创建MultipleAttributeUserDao模拟用户自定义属性

|  |
| --- |
| package org.jasig.services.persondir.support;  import org.jasig.services.persondir.IPersonAttributes;  import java.util.Collections;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  /\*\*  \* 扩展用户自定义属性  \* Created by Administrator on 2016/4/30 0030.  \*/  public class MultipleAttributeUserDao extends StubPersonAttributeDao {  @Override  public IPersonAttributes getPerson(String uid) {  Map<String, List<Object>> attributes = new HashMap<String, List<Object>>();  attributes.put("userid", Collections.singletonList((Object) uid));  attributes.put("username", Collections.singletonList((Object) "测试username"));  attributes.put("email", Collections.singletonList((Object) "test@163.com"));  return new AttributeNamedPersonImpl(attributes);  }  } |

## 修改deployerConfigContext.xml

修改src\main\webapp\WEB-INF\deployerConfigContext.xml文件，添加配置我们自定义的MultipleAttributeUserDao

<bean id="attributeRepository" class="org.jasig.services.persondir.support.MultipleAttributeUserDao"/>

## 修改casServiceValidationSuccess.jsp

至此为了能获取到我们的自定义属性还不需要修改src\main\webapp\WEB-INF\view\jsp\protocol\2.0\casServiceValidationSuccess.jsp和src\main\webapp\WEB-INF\view\jsp\protocol\3.0\casServiceValidationSuccess.jsp文件。添加获取用户自定义属性的代码块。

src\main\webapp\WEB-INF\view\jsp\protocol\2.0\casServiceValidationSuccess.jsp

|  |
| --- |
| <%--  <%@ page session="false" contentType="application/xml; charset=UTF-8" %>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>  <cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>  <cas:authenticationSuccess>  <cas:user>${fn:escapeXml(assertion.primaryAuthentication.principal.id)}</cas:user>  <!-- TODO 获取用户自定义属性 -->  <c:if test="${fn:length(assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.attributes) > 0}">  <cas:attributes>  <c:forEach var="attr"  items="${assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.attributes}">  <cas:${fn:escapeXml(attr.key)}>${fn:escapeXml(attr.value)}</cas:${fn:escapeXml(attr.key)}>  </c:forEach>  </cas:attributes>  </c:if>  <!-- TODO 获取用户自定义属性 -->  <c:if test="${not empty pgtIou}">  <cas:proxyGrantingTicket>${pgtIou}</cas:proxyGrantingTicket>  </c:if>  <c:if test="${fn:length(assertion.chainedAuthentications) > 1}">  <cas:proxies>  <c:forEach var="proxy" items="${assertion.chainedAuthentications}" varStatus="loopStatus" begin="0"  end="${fn:length(assertion.chainedAuthentications)-2}" step="1">  <cas:proxy>${fn:escapeXml(proxy.principal.id)}</cas:proxy>  </c:forEach>  </cas:proxies>  </c:if>  </cas:authenticationSuccess>  </cas:serviceResponse> |

src\main\webapp\WEB-INF\view\jsp\protocol\3.0\casServiceValidationSuccess.jsp

|  |
| --- |
| <%@ page session="false" contentType="application/xml; charset=UTF-8" %>  <%@ page import="java.util.Map.Entry" %>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>  <cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>  <cas:authenticationSuccess>  <cas:user>${fn:escapeXml(assertion.primaryAuthentication.principal.id)}</cas:user>  <c:if test="${not empty pgtIou}">  <cas:proxyGrantingTicket>${pgtIou}</cas:proxyGrantingTicket>  </c:if>  <c:if test="${fn:length(assertion.chainedAuthentications) > 1}">  <cas:proxies>  <c:forEach var="proxy" items="${assertion.chainedAuthentications}" varStatus="loopStatus" begin="0"  end="${fn:length(assertion.chainedAuthentications)-2}" step="1">  <cas:proxy>${fn:escapeXml(proxy.principal.id)}</cas:proxy>  </c:forEach>  </cas:proxies>  </c:if>  <c:if test="${fn:length(assertion.primaryAuthentication.principal.attributes) > 0}">  <cas:attributes>  <c:forEach var="attr"  items="${assertion.primaryAuthentication.principal.attributes}"  varStatus="loopStatus" begin="0"  end="${fn:length(assertion.primaryAuthentication.principal.attributes)}"  step="1">  <%-- ${attr.value['class'].simpleName} fails for List: use scriptlet instead --%>  <%  Entry entry = (Entry) pageContext.getAttribute("attr");  Object value = entry.getValue();  pageContext.setAttribute("isAString", value instanceof String);  %>  <c:choose>  <%-- it's a String, output it once --%>  <c:when test="${isAString}">  <cas:${fn:escapeXml(attr.key)}>${fn:escapeXml(attr.value)}</cas:${fn:escapeXml(attr.key)}>  </c:when>  <%-- if attribute is multi-valued, list each value under the same attribute name --%>  <c:otherwise>  <c:forEach var="attrval" items="${attr.value}">  <cas:${fn:escapeXml(attr.key)}>${fn:escapeXml(attrval)}</cas:${fn:escapeXml(attr.key)}>  </c:forEach>  </c:otherwise>  </c:choose>  </c:forEach>  </cas:attributes>  </c:if>  </cas:authenticationSuccess>  </cas:serviceResponse> |

# CAS Server集群部署时Ticket同步

当cas server被正式用在生产环境时，她需要支撑各个应用系统的登录服务。而cas server只进行一处部署时，这里就存在一个单点故障的问题。一旦出现该问题，就会导致各个应用系统无法正常登录。次问题是极为严重的。因此我们在线上使用的时候就需要将cas server进行集群部署，可cas server相对于其他web应用集群部署有所不同。我们知道cas server的用户身份校验是基于cas ticket的。Cas默认的ticket管理策略是不具备集群部署时ticket同步的。这里就需要我们自行拓展，以达到集群部署时ticket也能同步。

## 添加redis依赖

在这里我们使用redis缓存策略来同步管理cas ticket。因此需要添加redis依赖

|  |
| --- |
| <!-- redis -->  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-redis</artifactId>  <version>1.7.1.RELEASE</version>  <exclusions>  <exclusion>  <groupId>\*</groupId>  <artifactId>\*</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-pool2</artifactId>  <version>2.4.2</version>  </dependency>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.8.1</version>  </dependency> |

## 创建RedisTicketRegistry

在org.jasig.cas.ticket.registry包下创建RedisTicketRegistry，提供一些基于redis对Ticket的增删改查操作。这里涉及内容较多，具体不一一列出，请查看源码。

## 创建TicketRedisTemplate

在org.jasig.cas.ticket.registry包下创建TicketRedisTemplate，用来管理redis的序列化策略

|  |
| --- |
| package org.jasig.cas.ticket.registry;  import org.jasig.cas.ticket.Ticket;  import org.springframework.data.redis.connection.RedisConnectionFactory;  import org.springframework.data.redis.core.RedisTemplate;  import org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;  import org.springframework.data.redis.serializer.RedisSerializer;  import org.springframework.data.redis.serializer.StringRedisSerializer;  /\*\*  \* CasServer集群环境下使用Redis缓存Ticket  \* Created by fengjing on 2016/4/27.  \*/  public class TicketRedisTemplate extends RedisTemplate<String, Ticket> {  public TicketRedisTemplate() {  RedisSerializer<String> string = new StringRedisSerializer();  JdkSerializationRedisSerializer jdk = new JdkSerializationRedisSerializer();  setKeySerializer(string);  setValueSerializer(jdk);  setHashKeySerializer(string);  setHashValueSerializer(jdk);  }  public TicketRedisTemplate(RedisConnectionFactory connectionFactory) {  this();  setConnectionFactory(connectionFactory);  afterPropertiesSet();  }  } |

## 修改ticketRegistry.xml

修改src\main\webapp\WEB-INF\spring-configuration\ticketRegistry.xml文件，配置我们自定义的RedisTicketRegistry

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:p="http://www.springframework.org/schema/p"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  <description>  Configuration for the default TicketRegistry which stores the tickets in-memory and cleans them out as specified  intervals.  </description>  <!-- Ticket Registry -->  <!--<bean id="ticketRegistry" class="org.jasig.cas.ticket.registry.DefaultTicketRegistry" />-->  <bean id="ticketRegistry" class="org.jasig.cas.ticket.registry.RedisTicketRegistry">  <constructor-arg index="0" ref="redisTemplate"/>  <constructor-arg index="1" value="28800"/>  <constructor-arg index="2" value="7200"/>  </bean>  <bean id="jedisConnFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"  p:hostName="${redis.hostname}" p:port="${redis.port}" p:database="${redis.database}"  p:usePool="${redis.usePool}"/>  <bean id="redisTemplate" class="org.jasig.cas.ticket.registry.TicketRedisTemplate"  p:connectionFactory-ref="jedisConnFactory">  </bean>  <!--Quartz -->  <!-- TICKET REGISTRY CLEANER -->  <!--  <bean id="ticketRegistryCleaner" class="org.jasig.cas.ticket.registry.support.DefaultTicketRegistryCleaner"  p:ticketRegistry-ref="ticketRegistry"  p:logoutManager-ref="logoutManager" />  <bean id="jobDetailTicketRegistryCleaner" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"  p:targetObject-ref="ticketRegistryCleaner"  p:targetMethod="clean" />  <bean id="triggerJobDetailTicketRegistryCleaner" class="org.springframework.scheduling.quartz.SimpleTriggerBean"  p:jobDetail-ref="jobDetailTicketRegistryCleaner"  p:startDelay="20000"  p:repeatInterval="5000000" />  -->  </beans> |

这里关于redis的配置放在了cas.properties文件中。

## 修改web.xml

最后一步，记得在web.xml中添加<distributable/>。使其支持集群部署。

关于tomcat集群部署及整合nginx或apache负载均衡session同步的内容这里不做赘述，可参考我之前写的搭建配置文档。

使用Redis或Memcached实现Tomcat7+Nginx集群及Session共享

<https://github.com/sxyx2008/tomcat7-nginx-redis-memcached-cluster>

Apache、Tomcat7集群session共享及负载均衡

<https://github.com/sxyx2008/apache2-tomcat7-cluster>

# CAS Server登录页定制

这部分内容较为简单。

## 拷贝资源文件

事先让美工设计一套登录页面。然后将css、image、js拷贝到src\main\webapp\css src\main\webapp\images src\main\webapp\js目录下。

## 修改casLoginView.jsp

修改src\main\webapp\WEB-INF\view\jsp\default\ui\casLoginView.jsp文件，直接拷贝原来的spring form表单组件替换现有模板对应的表单元素。注意其name属性的值。

下图为自定义的cas server登录页



## 基于theme方式

以上是一种快捷便利的修改方式，官方也提供基于theme的修改方案。

按原有目录结构拷贝src\main\webapp\WEB-INF\view\jsp\default一份并为其重命名。该名称即为theme name。然后修改src\main\webapp\WEB-INF\cas.properties进行如下配置

cas.viewResolver.basename=default\_views

# cas定制登录页主题lamppa 主题文件路径src\main\webapp\WEB-INF\view\jsp\lamppa

#cas.themeResolver.defaultThemeName=cas-theme-lamppa

#cas.viewResolver.basename=lamppa\_views

接着修改对应的文件即可。

# 修改CAS Server为移动端设备提供REST登录服务

<http://apereo.github.io/cas/4.0.x/protocol/REST-Protocol.html>

Cas Server主要为web应用提供单点登录解决方案。可在现实中有时候我们也需要为移动端设备提供登录服务。这时候难道还是去请求cas server的统一登录页吗？这有点太low了。Cas官方为我们提供了cas rest风格的登录支持。要使用该功能，就必须得添加相应的依赖。

## 添加依赖

|  |
| --- |
| <!-- cas server restlet -->  <dependency>  <groupId>org.jasig.cas</groupId>  <artifactId>cas-server-integration-restlet</artifactId>  <version>4.0.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet</artifactId>  <version>2.1.0</version>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.spring</artifactId>  <version>2.1.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-asm</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  </exclusion>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring-expression</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.servlet</artifactId>  <version>2.1.0</version>  </dependency>  <dependency>  <groupId>org.restlet.jee</groupId>  <artifactId>org.restlet.ext.slf4j</artifactId>  <version>2.1.0</version>  </dependency> |

## 修改webx.xml

在web.xml中添加如下配置

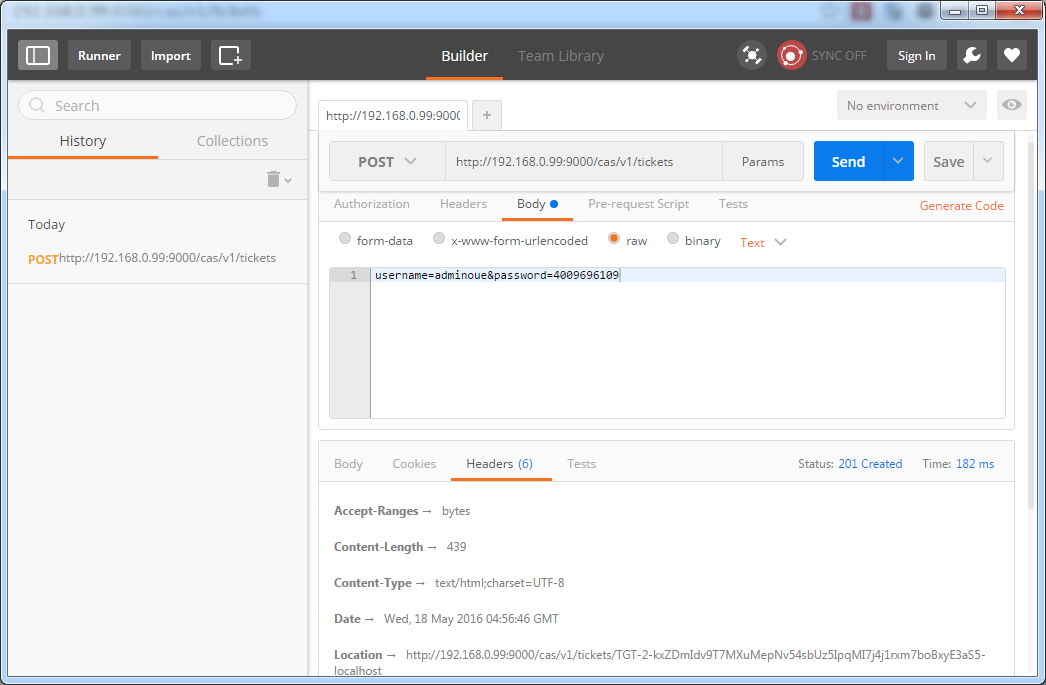
|  |
| --- |
| <servlet>  <servlet-name>restlet</servlet-name>  <servlet-class>org.restlet.ext.spring.RestletFrameworkServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>restlet</servlet-name>  <url-pattern>/v1/\*</url-pattern>  </servlet-mapping> |

## 请求票据授权

POST <http://192.168.0.99:9000/cas/v1/tickets> username=adminoue&password=4009696109

发送一个POST请求到/cas/v1/tickets，传递请求参数为username=adminoue&password=4009696109。请求成功后会返回http://192.168.0.99:9000/cas/v1/tickets/TGT-7-U71jox1ykWosLwQQSpcc5g47jdKul9ccgjgxgIchl64rSTpa1K-localhost。

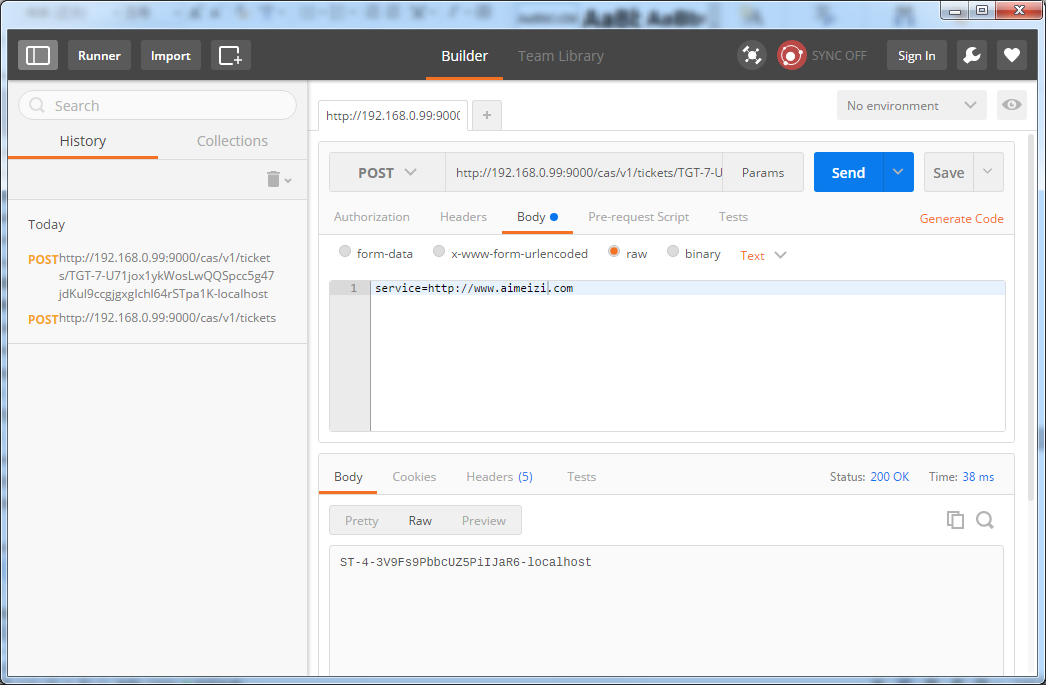
其TGT-7-U71jox1ykWosLwQQSpcc5g47jdKul9ccgjgxgIchl64rSTpa1K-localhost为服务器端返回的请求票根。下文在请求服务的时候需携带该信息。



## 请求服务授权

POST <http://192.168.0.99:9000/cas/v1/tickets/TGT-7-U71jox1ykWosLwQQSpcc5g47jdKul9ccgjgxgIchl64rSTpa1K-localhost> service=http://www.aimeizi.net即可，最终返回一串ST-4-3V9Fs9PbbcUZ5PiIJaR6-localhost表示登录成功。

这里需要携带上文请求后返回的请求票据，请求该票据并传递service=你的应用地址最终返回登录成功后的票据信息。



## Java Client测试

这里使用commons-httpclient来模拟上述中的请求流程。

|  |
| --- |
| import org.apache.commons.httpclient.HttpClient;  import org.apache.commons.httpclient.NameValuePair;  import org.apache.commons.httpclient.methods.GetMethod;  import org.apache.commons.httpclient.methods.PostMethod;  import java.io.IOException;  import java.net.URLEncoder;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  /\*\*  \* Created by Administrator on 2016/5/11 0011.  \*/  public class Client {  public static String getTicket(final String server, final String username, final String password,  final String service) {  notNull(server, "server must not be null");  notNull(username, "username must not be null");  notNull(password, "password must not be null");  notNull(service, "service must not be null");  return getServiceTicket(server, getTicketGrantingTicket(server, username, password), service);  }  /\*\*  \* 取得ST  \*  \* @param server  \* @param ticketGrantingTicket  \* @param service  \*/  private static String getServiceTicket(final String server, final String ticketGrantingTicket, final String service) {  if (ticketGrantingTicket == null)  return null;  final HttpClient client = new HttpClient();  final PostMethod post = new PostMethod(server + "/" + ticketGrantingTicket);  post.setRequestBody(new NameValuePair[]{new NameValuePair("service", service)});  try {  client.executeMethod(post);  final String response = post.getResponseBodyAsString();  switch (post.getStatusCode()) {  case 200:  return response;  default:  warning("Invalid response code (" + post.getStatusCode() + ") from CAS server!");  info("Response (1k): " + response.substring(0, Math.min(1024, response.length())));  break;  }  } catch (final IOException e) {  warning(e.getMessage());  } finally {  post.releaseConnection();  }  return null;  }  /\*\*  \* @param server  \* @param username  \* @param password  \*/  private static String getTicketGrantingTicket(final String server, final String username, final String password) {  final HttpClient client = new HttpClient();  final PostMethod post = new PostMethod(server);  post.setRequestBody(new NameValuePair[]{new NameValuePair("username", username),  new NameValuePair("password", password)});  try {  client.executeMethod(post);  final String response = post.getResponseBodyAsString();  info("TGT=" + response);  switch (post.getStatusCode()) {  case 201: {  final Matcher matcher = Pattern.compile(".\*action=\".\*/(.\*?)\".\*").matcher(response);  if (matcher.matches())  return matcher.group(1);  warning("Successful ticket granting request, but no ticket found!");  info("Response (1k): " + response.substring(0, Math.min(1024, response.length())));  break;  }  default:  warning("Invalid response code (" + post.getStatusCode() + ") from CAS server!");  info("Response (1k): " + response.substring(0, Math.min(1024, response.length())));  break;  }  } catch (final IOException e) {  warning(e.getMessage());  } finally {  post.releaseConnection();  }  return null;  }  private static void ticketValidate(String serverValidate, String serviceTicket, String service) {  notNull(serviceTicket, "paramter 'serviceTicket' is not null");  notNull(service, "paramter 'service' is not null");  final HttpClient client = new HttpClient();  GetMethod post = null;  try {  post = new GetMethod(serverValidate + "?" + "ticket=" + serviceTicket + "&service=" + URLEncoder.encode(service, "UTF-8"));  client.executeMethod(post);  final String response = post.getResponseBodyAsString();  info(response);  switch (post.getStatusCode()) {  case 200: {  info("成功取得用户数据");  }  default: {  }  }  } catch (Exception e) {  warning(e.getMessage());  } finally {  //释放资源  post.releaseConnection();  }  }  private static void notNull(final Object object, final String message) {  if (object == null)  throw new IllegalArgumentException(message);  }  public static void main(final String[] args) throws Exception {  final String server = "http://192.168.0.99:9000/cas/v1/tickets";  final String username = "adminoue";  final String password = "4009696109";  final String service = "http://www.aimeizi.net";  final String proxyValidate = "http://192.168.0.99:9000/cas/proxyValidate";  ticketValidate(proxyValidate, getTicket(server, username, password, service), service);  }  private static void warning(String msg) {  System.out.println(msg);  }  private static void info(String msg) {  System.out.println(msg);  }  } |

最终可以看到控制台请求到cas server服务器返回的数据。

