

CAS 集成、认证流程和模拟登录

[向林 / cas-demo · GitLab](#)

CAS 集成

CAS Server 简单配置启动

[cas-overlay-template](#) 项目地址

[CAS](#) 文档地址

CAS 服务器启动步骤：

1. 从 [cas-overlay-template](#) 中获取 5.3 分支的项目文件，如图

```
1 git clone -b 5.3 git@github.com:apereo/cas-overlay-template.git
```

```
→ cas git clone -b 5.3 git@github.com:apereo/cas-overlay-template.git
Cloning into 'cas-overlay-template'...
remote: Enumerating objects: 1924, done.
remote: Counting objects: 100% (83/83), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 1924 (delta 40), reused 51 (delta 22), pack-reused 1841
Receiving objects: 100% (1924/1924), 10.67 MiB | 852.00 KiB/s, done.
Resolving deltas: 100% (1066/1066), done.
→ cas cd cas-overlay-template
→ cas-overlay-template git:(5.3) tree
.
├── LICENSE.txt
├── README.md
├── build.cmd
├── build.sh
├── etc
│   └── cas
│       └── config
│           ├── cas.properties
│           └── log4j2.xml
├── maven
│   ├── maven-wrapper.jar
│   └── maven-wrapper.properties
├── mvnw
├── mvnw.bat
└── pom.xml

4 directories, 11 files
```

2. CAS 默认使用 https，需要先生成证书

1. 可以使用项目中已提供的脚本生成，默认存储在 `/etc/cas/thekeystore`

```
1 ./build.sh gencert
```

2. 也可以手动生成，如图

```

1 # 生成证书
2 keytool -genkey -alias cas -keyalg RSA -keystore thekeystore -
  storepass changeit -deststoretype pkcs12
3 # 导出证书
4 keytool -export -file cas.crt -alias cas -keystore thekeystore -
  storepass changeit
5 # 导入证书到JVM的证书库
6 keytool -import -keystore
  "/usr/local/opt/openjdk@8/libexec/openjdk.jdk/Contents/Home/jre/l
  ib/security/cacerts" -file /Users/linxiang/Documents/cas/cas.crt
  -alias cas -storepass changeit

```

```

→ cas git:(5.3) keytool -genkey -alias cas -keyalg RSA -keystore thekeystore -s
torepass changeit
您的名字与姓氏是什么？
[Unknown]: cas.server.com
您的组织单位名称是什么？
[Unknown]:
您的组织名称是什么？
[Unknown]:
您所在的城市或区域名称是什么？
[Unknown]:
您所在的省/市/自治区名称是什么？
[Unknown]:
该单位的双字母国家/地区代码是什么？
[Unknown]:
CN=cas.server.com, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown是否正
确？
[否]: y

输入 <cas> 的密钥口令
(如果和密钥库口令相同，按回车):

```

```

→ cas git:(5.3) ✗ keytool -export -file cas.crt -alias cas -keystore thekeystor
e -storepass changeit
存储在文件 <cas.crt> 中的证书

```

3. 将 etc 文件夹复制到根目录下

```

1 cp -r etc/cas/config /etc/cas/config

```

可以修改 `cas.properties` 中的配置，比如静态认证默认的用户名密码，应用端口等

```
1 logging.config=file:/etc/cas/config/log4j2.xml
2 cas.authn.accept.users=admin::admin,xianglin::xianglin
3 server.ssl.key-store=file:/etc/cas/thekeystore
4 server.ssl.key-store-password=changeit
5 server.ssl.key-password=changeit
```

其中 `server.ssl.key-store` 配置了证书的存储位置。可以将第 2 步生成的证书复制到 `/etc/cas/thekeystore`，也可以修改 `key-store` 的值指向证书位置。

4. 使用 WAR 方式启动项目，访问认证页面 `https://cas.server.com:8443/cas/login`，如图

```
1 ./build.sh run
```

← → ↻ 🏠 ⚠ 不安全 | https://cas.server.com:8443/cas/login

CAS

Login

🔒

请输入您的用户名和密码.

用户名:

密码:

登录

🔗 [Forgot your password?](#)

出于安全考虑，一旦您访问过那些需要您提供凭证信息的应用时，请操作完成之后[登出](#)并关闭浏览器。

Static Authentication

CAS is configured to accept a static list of users for primary authentication. Please be advised that this is ONLY useful for demo purposes. It is recommended that you connect CAS to LDAP, JDBC, etc instead.

Links to CAS Resources

- ⚙ Dashboard
- 📖 Documentation
- 🔗 Pull Requests
- 👤 Contributor Guidelines
- 📧 Mailing Lists
- 💬 Chatroom

客户端集成 CAS

CAS 客户端示例

SpringBoot 配置

CAS 客户端启动步骤：

1. 修改 web.xml 配置，主要是：

casServerUrlPrefix：指向 CAS 服务器地址，

如：https://cas.server.com:8443/cas

serverName：CAS 认证后重定向的地址，即客户端地址，

如：https://localhost:9443/sample

完整 web.xml 示例：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
4
5     <filter>
6         <filter-name>CAS Single Sign Out Filter</filter-name>
7         <filter-
  class>org.jasig.cas.client.session.SingleSignOutFilter</filter-
  class>
8         <init-param>
9             <param-name>casServerUrlPrefix</param-name>
10            <param-value>https://cas.server.com:8443/cas</param-
  value>
11        </init-param>
12    </filter>
13
14    <listener>
15        <listener-
  class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener<
  /listener-class>
16    </listener>
17
18    <filter>
19        <filter-name>CAS Authentication Filter</filter-name>
```

```
20         <filter-  
class>org.jasig.cas.client.authentication.AuthenticationFilter</filt  
er-class>  
21         <init-param>  
22             <param-name>casServerLoginUrl</param-name>  
23             <param-  
value>https://cas.server.com:8443/cas/login</param-value>  
24         </init-param>  
25         <init-param>  
26             <!--这是客户端的部署地址，认证时会带着这个地址，认证成功后会跳转到  
这个地址-->  
27             <param-name>serverName</param-name>  
28             <param-value>https://localhost:9443/sample</param-value>  
29         </init-param>  
30     </filter>  
31  
32     <filter>  
33         <filter-name>CAS Validation Filter</filter-name>  
34         <filter-  
class>org.jasig.cas.client.validation.Cas30ProxyReceivingTicketValid  
ationFilter</filter-class>  
35         <init-param>  
36             <param-name>casServerUrlPrefix</param-name>  
37             <param-value>https://cas.server.com:8443/cas</param-  
value>  
38         </init-param>  
39         <init-param>  
40             <param-name>serverName</param-name>  
41             <param-value>https://localhost:9443/sample</param-value>  
42         </init-param>  
43         <init-param>  
44             <param-name>redirectAfterValidation</param-name>  
45             <param-value>true</param-value>  
46         </init-param>  
47         <init-param>  
48             <param-name>useSession</param-name>
```

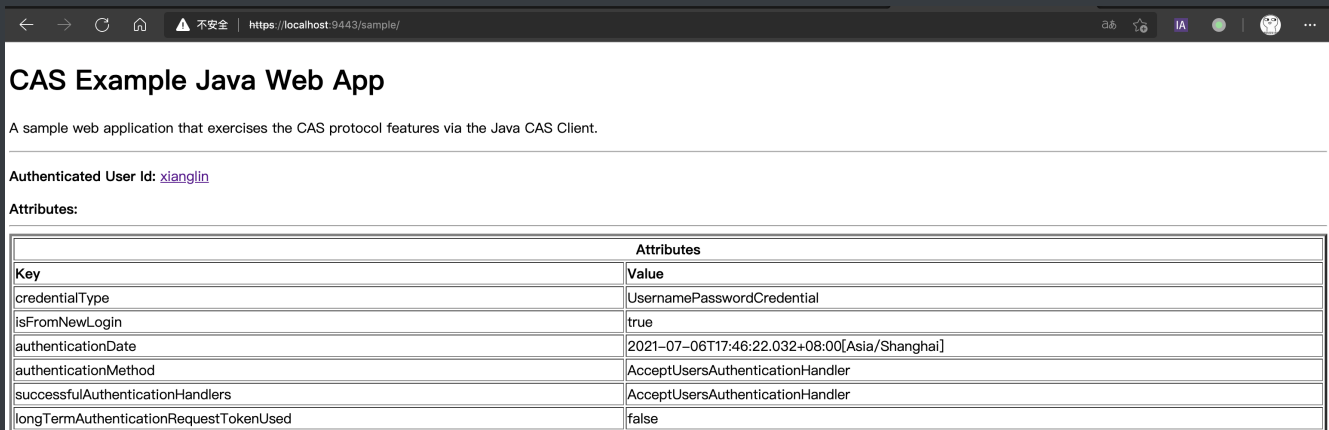
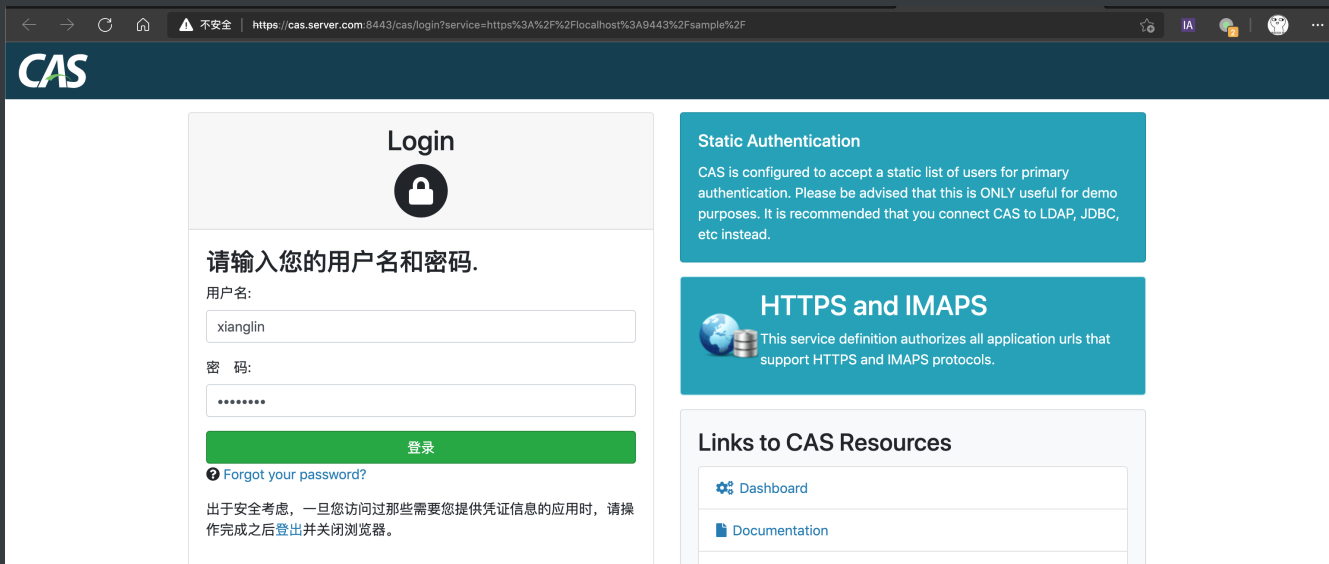
```
49         <param-value>true</param-value>
50     </init-param>
51     <init-param>
52         <param-name>authn_method</param-name>
53         <param-value>mfa-duo</param-value>
54     </init-param>
55 </filter>
56
57     <!-- 该过滤器负责实现HttpServletRequest请求的包裹，比如允许开发者通过
    HttpServletRequest的getRemoteUser()方法获得SSO登录用户的登录名，可选配置-->
58     <filter>
59         <filter-name>CAS HttpServletRequest Wrapper Filter</filter-
    name>
60         <filter-
    class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</fil
    ter-class>
61     </filter>
62
63     <filter-mapping>
64         <filter-name>CAS Single Sign Out Filter</filter-name>
65         <url-pattern>/*</url-pattern>
66     </filter-mapping>
67
68     <filter-mapping>
69         <filter-name>CAS Validation Filter</filter-name>
70         <url-pattern>/*</url-pattern>
71     </filter-mapping>
72
73     <filter-mapping>
74         <filter-name>CAS Authentication Filter</filter-name>
75         <url-pattern>/*</url-pattern>
76     </filter-mapping>
77
78     <filter-mapping>
```

```
79     <filter-name>CAS HttpServletRequest Wrapper Filter</filter-  
    name>  
80     <url-pattern>/*</url-pattern>  
81 </filter-mapping>  
82  
83     <welcome-file-list>  
84         <welcome-file>  
85             index.jsp  
86         </welcome-file>  
87     </welcome-file-list>  
88 </web-app>
```

2. 修改 Tomcat 配置, 见 [server.xml](#)

```
1  <!--<Connector port="8080" protocol="HTTP/1.1"  
2           connectionTimeout="20000"  
3           redirectPort="8443" /> -->  
4  <Connector port="9443"  
    protocol="org.apache.coyote.http11.Http11Protocol"  
5           maxThreads="150" SSLEnabled="true" scheme="https"  
    secure="true"  
6           clientAuth="false" sslProtocol="TLS"  
    keystoreFile="/Users/linxiang/Documents/cas/thekeystore"  
7           keystorePass="changeit"/>
```

3. 启动客户端, 如图



Spring Boot 应用简单集成CAS 认证

1. 在 pom.xml 中添加 CAS 客户端依赖

```
1 <dependency>
2   <groupId>org.jasig.cas.client</groupId>
3   <artifactId>cas-client-support-springboot</artifactId>
4   <version>3.6.2</version>
5 </dependency>
```

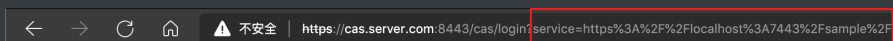
2. 在 application.properties 添加 CAS 配置

```
1  server:
2    port: 7443
3    ssl:
4      key-store: classpath:thekeystore
5      key-store-password: changeit
6      key-store-type: pkcs12
7      key-alias: cas
8
9  cas:
10   server-url-prefix: https://cas.server.com:8443/cas
11   server-login-url: https://cas.server.com:8443/cas/login
12   client-host-url: https://client1.server.com:7443/
```

3. 启用注解 @EnableCasClient

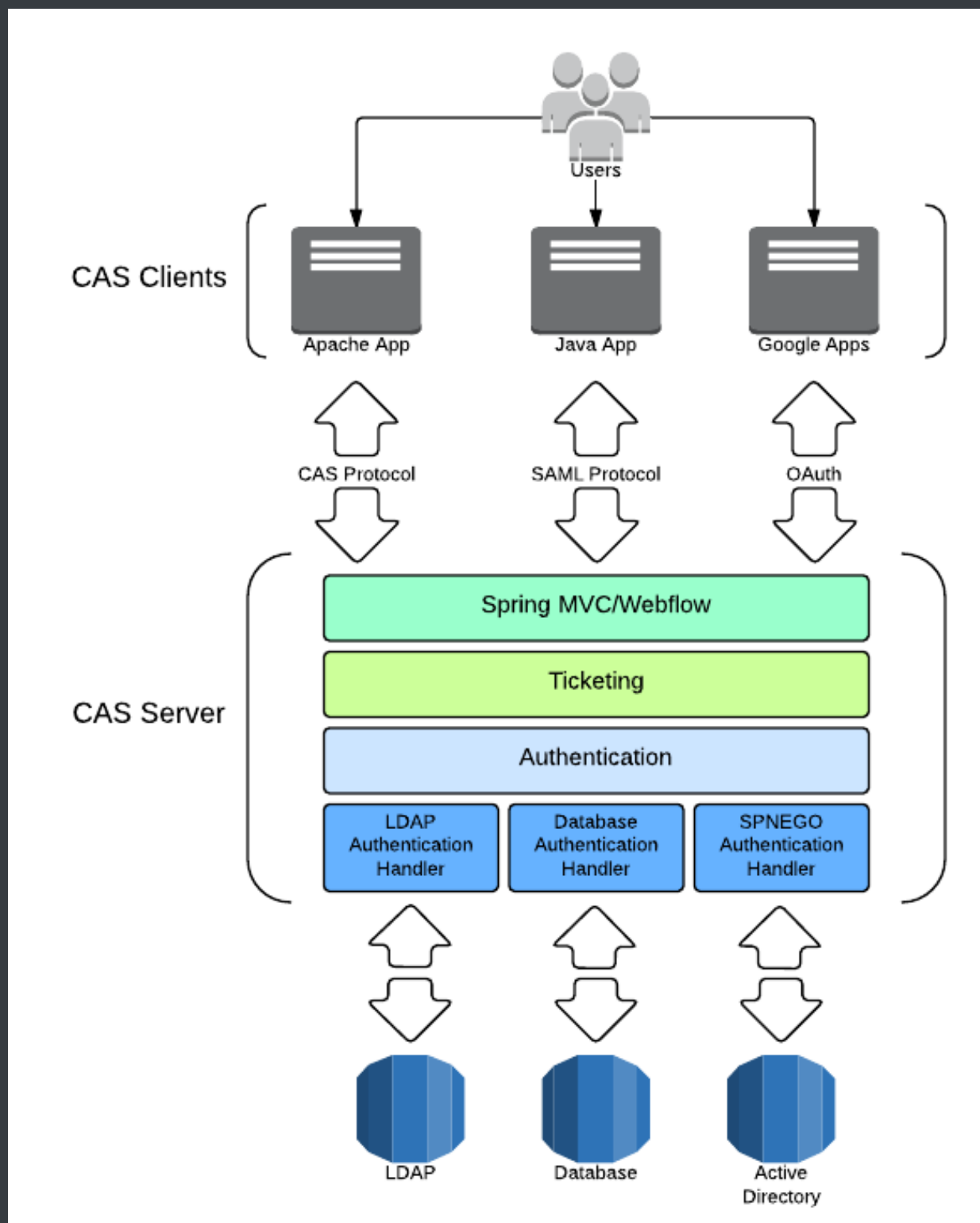
```
1  @SpringBootApplication
2  @EnableCasClient
3  public class SpringbootCasDemoApplication {
4
5      public static void main(String[] args) {
6          SpringApplication.run(SpringbootCasDemoApplication.class,
7              args);
8      }
9  }
```

4. 启动客户端，如图



← → ↻ 🏠 ⚠ 不安全 | https://cas.server.com:8443/cas/login?service=https%3A%2F%2Flocalhost%3A7443%2Fsample%2F

认证流程



CAS 名词

通过 Fiddler 抓包学习 CAS 的认证流程前，先熟悉几个名词（见 [CAS Protocol](#)）：

- TGC：（Ticket-granting Cookie），存放 SSO 用户身份凭据的 Cookie，类似 JSESSIONID。
- TGT：（Ticket Granting Ticket），在 SSO 中标识一位用户，类似 Session。

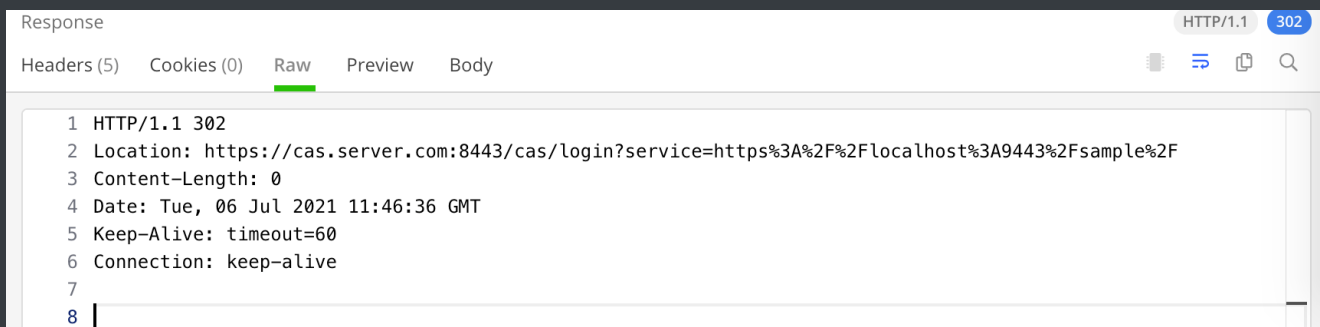
- ST : (Service Ticket), 作为一个参数跟在 url 后面, 代表 CAS 为某一用户访问 CAS 客户端签发的凭据。

首次登录

打开浏览器, 键入 `https://localhost:9443/sample/`, 输入口令完成登录, 在 Fiddler 中过滤无关请求, 需要学习的请求如图。

# ↑	URL	Result	Method	Process	Remote IP	Body Size	Comments
930	https://localhost:9443/sample/	302	GET	microsoft edge :3116	::1	0 bytes	
932	https://cas.server.com:8443/cas/login?service=https...	200	GET	microsoft edge :3116	127.0.0.1	7,654 bytes	
956	https://cas.server.com:8443/cas/login?service=https...	302	POST	microsoft edge :3116	127.0.0.1	0 bytes	
957	https://localhost:9443/sample/?ticket=ST-18-qx-ZRVk...	302	GET	microsoft edge :3116	::1	0 bytes	
958	https://localhost:9443/sample/?sessionId=D065FD88...	200	GET	microsoft edge :3116	::1	1,135 bytes	

1. 序号 930, 第一次访问 `https://localhost:9443/sample/`, 会经过 CAS 提供的过滤器 `org.jasig.cas.client.authentication.AuthenticationFilter`, 过滤器判断到没有登录, 将请求重定向到认证中心, 如图。



重定向的地址是在过滤器参数中配置的 `casServerLoginUrl`, `service` 参数值则是过滤器参数中配置的 `serverName`, 即认证后要跳转的地址。

2. 序号 932, CAS 服务器接收到登录请求, 返回登录页面。
3. 序号 956, 输入口令登录, CAS 服务器验证用户名密码是否有效 (这里使用静态认证), 如图。

Request

/cas/login?service=https%3A%2F%2Flocalhost%3A8443%2Fsample/ HTTP/1.1 POST

Headers (18) Params (1) Cookies (0) Raw Body

Text JSON XML Form-Data

Key	Value
username	xianglin
password	xianglin
	932a0a32-63d0-42c0-b74d-0d18848c7e5b_ZXlKaGJHY2lPaUpJVXpVeE1pSjkuU0xTK3AvYkKTUJGQ09rS3ZramJtcM13eUFkN00xNHBGeVJqU0ZCYzJUM0JnbXZzcTZWUGl1NkFMUHR5STVlcHcxYUenJiMm5Yakx5Z2ZNdV6Y2RtVGsyUkg0NjlbCtJNE

Response

HTTP/1.1 302

Cache-Control: no-cache, no-store, max-age=0, must-revalidate

Pragma: no-cache

Expires: 0

Strict-Transport-Security: max-age=15768000 ; includeSubDomains

X-Content-Type-Options: nosniff

X-Frame-Options: DENY

X-XSS-Protection: 1; mode=block

Set-Cookie: TGC=eyJhbGciOiJIUzUxMiJ9. ZXlKNmFYQWlPaUpFVlZaUxDSmhiR2NpT2lKa2FYSWlMQ0psYm1NaU9pSkJNVEk0UTBKRExVaFRNa1UySW4wLi5kwl9vc3pYV09ETXVMeXFBLTBkTGVRljdTTE1Z2NvsQVg2c1JTS3RwWkdUdDdRVHlaRmNITjlmUTHhdGx0RnpHdTFQWjBER2Fwc2FK0ENzdU9CRW9BT1JNUW5VbElpRUZBbDVSQmFhS0ZPZGF6Nk9RaUZN3VDX21LS0xabzNjZ1M1UGh0YUVMTRxv3lGMU5Vc3d0VjdDNURXc3FIWFZRQnNYZWl6NW9VakhhREozR0hoQmtUUUEwZl82X3VfTm1SY25jcUI0WmpNRllPUlpleFA3ckR2SkpZaV55dm5yM24wbzJJNHNWZWZxZUQ3VXRhc3UwdnZ3MnlzcGctUUJaclJNSDhVTFVKcm1wV1kzaTVQZkQ3ZkV1ZG5tM2hz0VN5SWZMT04tU09BTWt2bUp1aTduZDR2ck8wN1M0TGwxVWRvLlNwNDlhTU5EdHlwQ0xIdWJhZ0V0WEE=.w9A2ky6AzEAiZd5ao283D8T_v4cdu0yrhabIXeUorvosI3e9njxkopciyr2iHSNahj0svfRQzv0QBinRzi1tlw; Path=/cas/; Secure; HttpOnly

Location: https://localhost:9443/sample/?ticket=ST-18-qx-ZRVk0akH9M63m7xhjBZABVBsLindeMacBook-Pro

Content-Length: 0

Date: Tue, 06 Jul 2021 11:46:42 GMT

使用 POST 请求的方式提交登录表单，认证成功后，返回状态码 302，将请求重定向到最开始访问的地址。请求地址通过 querySpring 携带了一个 ticket 令牌。同时设置一个名为 TGC 的 Cookie，路径为 /cas/，下次访问 https://cas.server.com:8443 时会携带这个 Cookie，服务器根据这个 TGC，查找与之对应的 TGT。从而判断用户是否登录过，是否需要展示登录页面。TGT 与 TGC 的关系就像 SESSION 与 Cookie 中 JSESSIONID 的关系。

4. 序号 957，浏览器向 https://localhost:9443/sample/ 发起请求，携带 CAS 服务器签发的 ST，localhost:9443 在过滤器中获取到 ticket 值，通过 HTTP 请求方式调用 CAS 服务验证该 ticket 是否有效，过滤器全称 org.jasig.cas.client.validation.Cas30ProxyReceivingTicketValidationFilter，验证方法在 org.jasig.cas.client.validation.AbstractUrlBasedTicketValidator#validate，如图。

```

public final Assertion validate(String ticket, String service) throws TicketValidationException {
    String validationUrl = this.constructValidationUrl(ticket, service);
    this.logger.debug("Constructing validation url: {}", validationUrl);

    try {
        this.logger.debug("Retrieving response from server.");
        String serverResponse = this.retrieveResponseFromServer(new URL(validationUrl), ticket);
    }
}

```

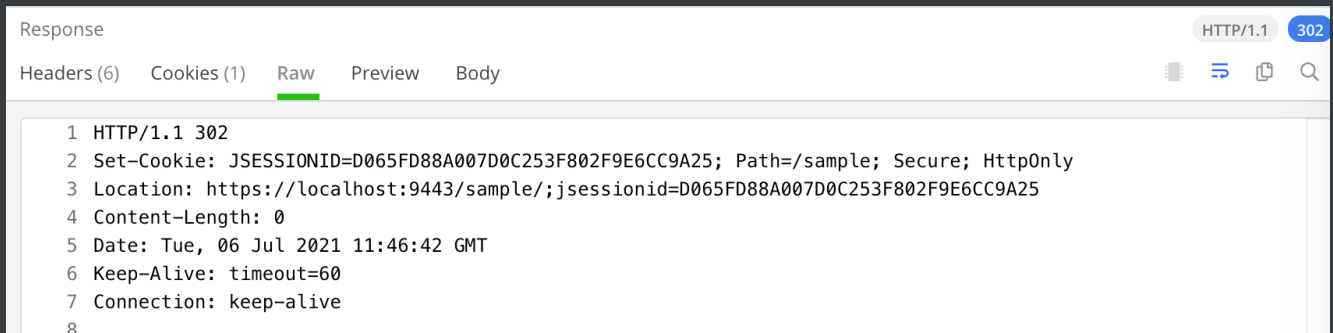
此时，CAS 服务器日志如下图。

```

2021-07-06 19:46:42,687 INFO [org.apereo.inspektr.audit.support.Slf4jLoggingAuditTrailManager] - <Audit trail record BEGIN
=====
WHO: xianglin
WHAT: ST-18-qx-ZRVk0akH9M63m7xhjBZABVBsLindeMacBook-Pro for https://localhost:9443/sample/
ACTION: SERVICE_TICKET_VALIDATED
APPLICATION: CAS
WHEN: Tue Jul 06 19:46:42 CST 2021
CLIENT IP ADDRESS: 127.0.0.1
SERVER IP ADDRESS: 127.0.0.1
=====

```

localhost:9443 收到 CAS 服务器返回，知道用户合法，即可正常响应请求，展示页面，如图。

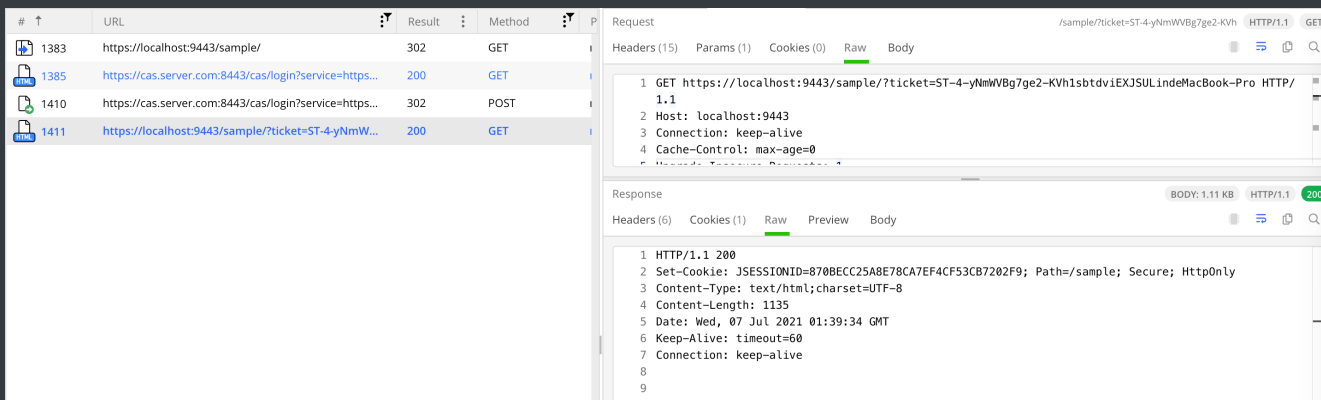


5. 序号 958，ST 验证通过后再次重定向是在 `Cas30ProxyReceivingTicketValidationFilter` 过滤器中使用了 `redirectAfterValidation` 参数，将其配置成 `false` 即可在验证用户后返回正常页面，如图。

```

this.onSuccessfulValidation(request, response, assertion);
if (this.redirectAfterValidation) {
    this.logger.debug("Redirecting after successful ticket validation.");
    response.sendRedirect(this.constructServiceUrl(request, response));
    return;
}

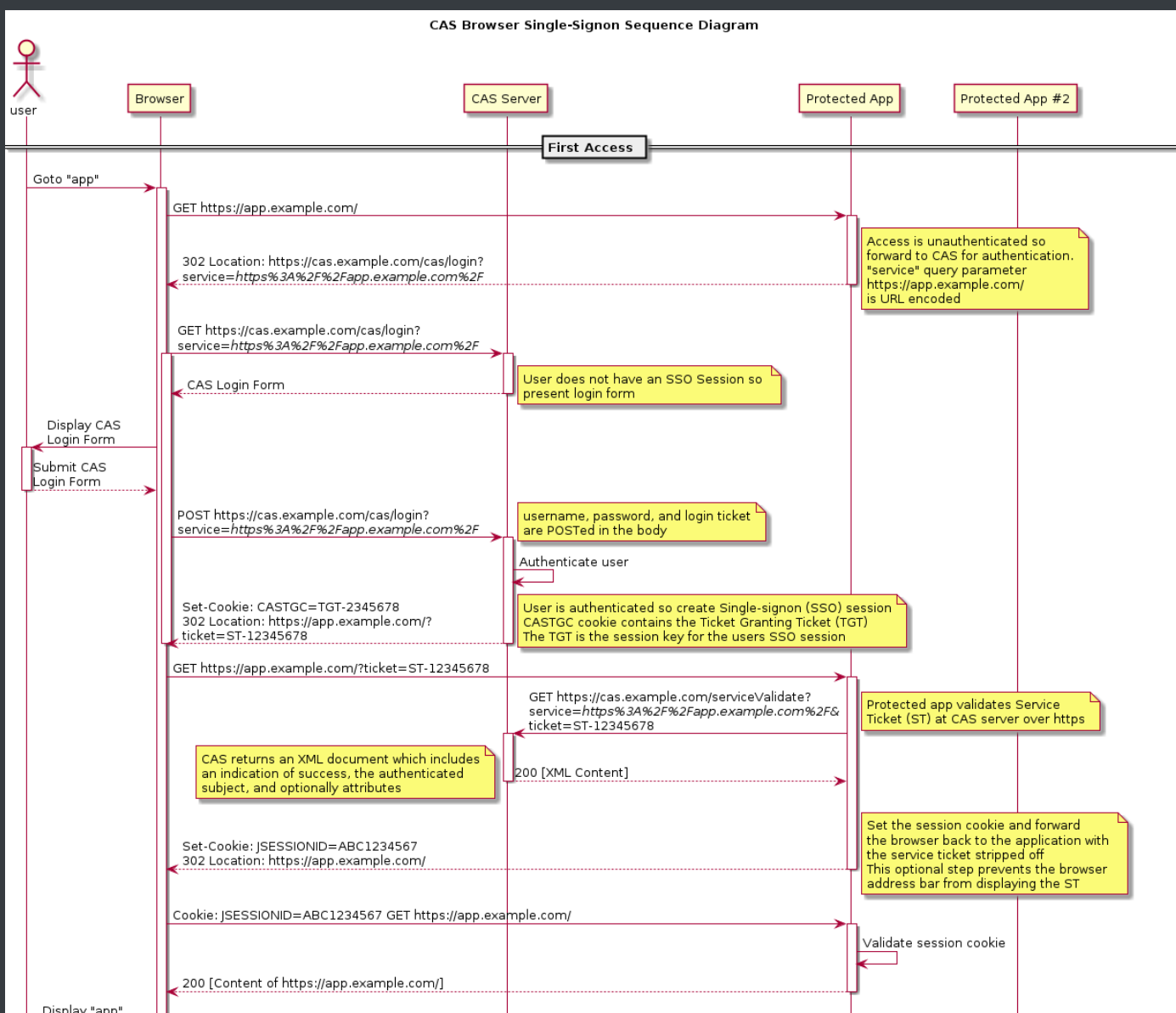
```

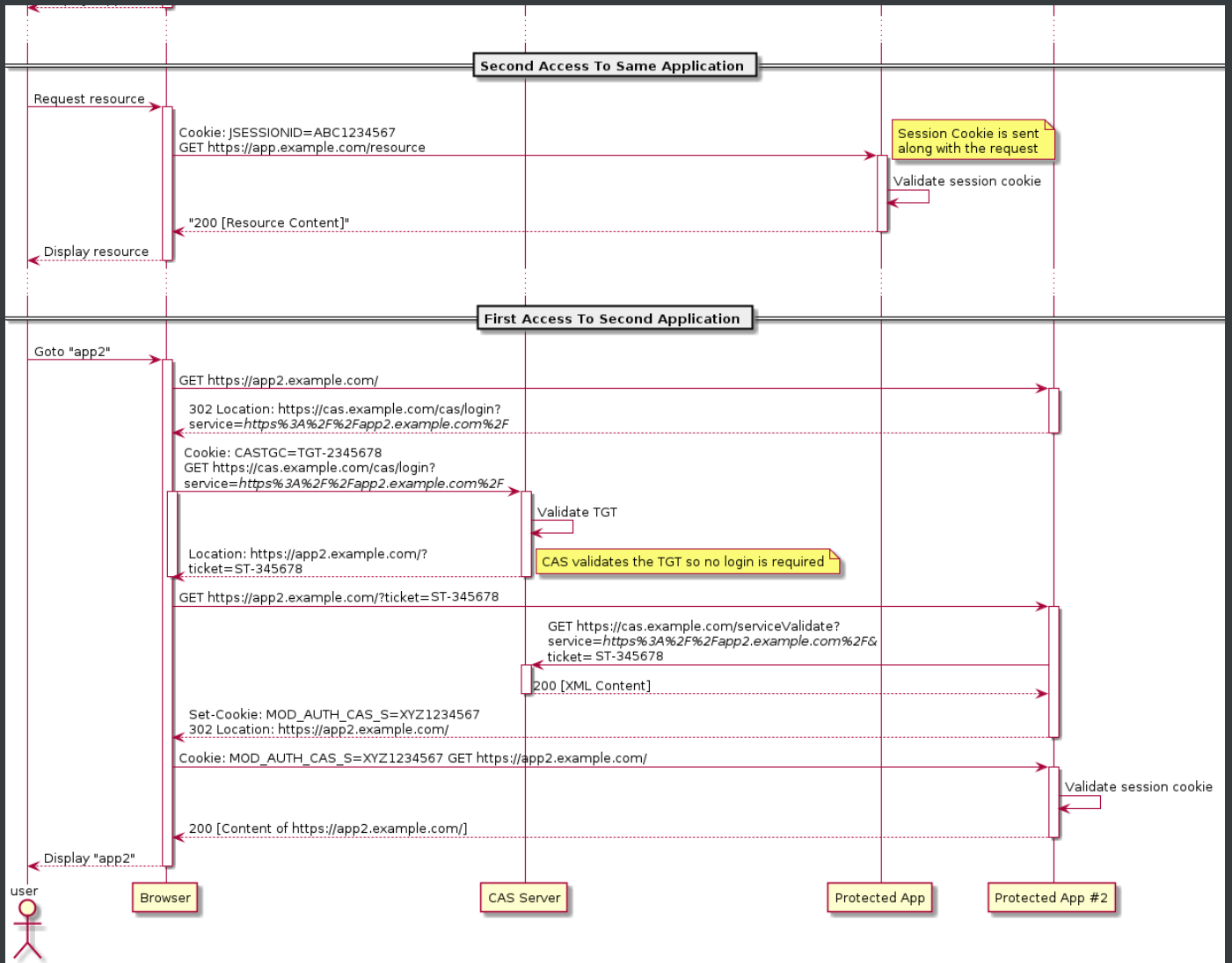


多个子系统登录

第一次访问 localhost:9443 的步骤已梳理完成，保持用户在 SSO 的会话，尝试访问 localhost:7443，少了 CAS 服务器登录这个步骤，剩余步骤和首次登录一致。

官方文档中的时序图如下，见 [CAS - CAS Protocol](#)。





模拟登录

示例代码见 [Simulation.java](#)

使用 HttpClient 模拟登录

参照首次登录流程，模拟登录时只需要处理好 CAS 认证、重定向到客户端两个步骤，就可以获取包含 JSESSIONID 的 Cookie，后续使用此 Cookie 便可以完成其它请求的模拟，简单的示例代码如下：

```

1 public class Simulation {
2     private static final String SERVER_LOGIN_URL =
        "https://cas.server.com:8443/cas/login";
3     private static final String CLIENT_HOST_URL =
        "https://client1.server.com:7443/";

```



```
4     private static final CloseableHttpClient client;
5
6     static {
7         client = HttpClients.createDefault();
8     }
9
10    /**
11     * 模拟登录，返回包含 JSESSIONID 的 Cookie
12     *
13     * @param userName userName
14     * @param password password
15     * @return CookieValue
16     * @throws IOException IOException
17     */
18    public String simulationLogin(String userName, String password)
19    throws IOException {
20        CloseableHttpResponse response;
21        String url = SERVER_LOGIN_URL + "?service=" + CLIENT_HOST_URL;
22        // 1、获取 CAS 登录页面
23        HttpGet httpGet = new HttpGet(url);
24        response = client.execute(httpGet);
25        StatusLine statusLine = response.getStatusLine();
26        int statusCode = statusLine.getStatusCode();
27        if (HttpStatus.SC_OK != statusCode) {
28            System.out.println("获取 CAS 登录页面返回错误! " + statusLine);
29            return null;
30        }
31        String casLoginPageHtml =
32        EntityUtils.toString(response.getEntity(), StandardCharsets.UTF_8);
33        IOUtils.closeQuietly(response);
34        // 使用 Jsoup 解析表单
35        Document document = Jsoup.parse(casLoginPageHtml);
36        Element fm1Element = document.getElementById("fm1");
37        Elements inputElements = fm1Element.getElementsByTag("input");
38        // 提交参数
39        List<NameValuePair> casLoginBody = new ArrayList<>(8);
```

```
38         for (Element inputElement : inputElements) {
39             String type = inputElement.attr("type");
40             if ("submit".equals(type)) {
41                 continue;
42             }
43             String name = inputElement.attr("name");
44             String value = inputElement.attr("value");
45             if ("username".equals(name)) {
46                 value = userName;
47             }
48             if ("password".equals(name)) {
49                 value = password;
50             }
51             casLoginBody.add(new BasicNameValuePair(name, value));
52         }
53         // 2、提交登录表单
54         HttpEntity loginEntity = new UrlEncodedFormEntity(casLoginBody,
StandardCharsets.UTF_8);
55         HttpPost httpPost = new HttpPost(url);
56         httpPost.setEntity(loginEntity);
57         response = client.execute(httpPost);
58         statusLine = response.getStatusLine();
59         if (HttpStatus.SC_MOVED_TEMPORARILY !=
statusLine.getStatusCode()) {
60             System.out.println("登录 CAS 返回错误! " + statusLine);
61             return null;
62         }
63         // 处理响应头
64         Header cookie = response.getFirstHeader("Set-Cookie");
65         System.out.println("cookie: " + cookie.getName() + " => " +
cookie.getValue());
66         Header location = response.getFirstHeader("Location");
67         String locationUrl = location.getValue();
68         System.out.println("locationUrl : " + locationUrl);
69         IOUtils.closeQuietly(response);
70         // 3、重定向到客户端
```

```
71         httpGet = new HttpGet(locationUrl);
72         response = client.execute(httpGet);
73         statusLine = response.getStatusLine();
74         if (HttpStatus.SC_MOVED_TEMPORARILY !=
statusLine.getStatusCode()) {
75             System.out.println("登录客户端返回错误! " + statusLine);
76             return null;
77         }
78         // 处理响应头, 返回 Cookie
79         cookie = response.getFirstHeader("Set-Cookie");
80         System.out.println("cookie: " + cookie.getName() + " => " +
cookie.getValue());
81         location = response.getFirstHeader("Location");
82         locationUrl = location.getValue();
83         System.out.println("locationUrl : " + locationUrl);
84         IOUtils.closeQuietly(response);
85         return cookie.getValue();
86     }
87 }
```

常见问题

忽略 SSL 证书

CAS 默认不支持 HTTP 请求, 使用 HttpClient 模拟登录时可能会遇到如下错误。

```
1 Exception in thread "main" javax.net.ssl.SSLHandshakeException: PKIX
  path building failed:
  sun.security.provider.certpath.SunCertPathBuilderException: unable to
  find valid certification path to requested target
2   at java.base/sun.security.ssl.Alert.createSSLException(Alert.java:131)
3   .....
4 Caused by: sun.security.validator.ValidatorException: PKIX path building
  failed: sun.security.provider.certpath.SunCertPathBuilderException:
  unable to find valid certification path to requested target
5   at
  java.base/sun.security.validator.PKIXValidator.doBuild(PKIXValidator.jav
  a:439)
```

有两种方式解决这个问题：

1. 使用如下语句将证书导入 JVM 的证书库

```
1 keytool -import -keystore "JAVA_HOME/jre/lib/security/cacerts" -file
  <path-to-cert> -alias cas -storepass changeit
```

2. 设置 HttpClient 忽略 SSL 证书，如示例代码

```
1 static {
2     // client = HttpClients.createDefault();
3     SSLContext sslContext;
4     try {
5         sslContext = SSLContexts.custom().loadTrustMaterial(null,
6             (x509Certificates, s) -> true).build();
7     } catch (Exception e) {
8         throw new RuntimeException(e);
9     }
10    client = HttpClients
11        .custom()
12        .setSslcontext(sslContext)
13        .build();
14 }
```

忽略主机名称验证

HttpClient 在 SSL 握手时验证对方主机名称，防止链接被重定向到其他地址上去，若服务器证书上的 HostName 和实际的 URL 不匹配时，会抛出如下异常，参考 [java - javax.net.ssl.SSLPeerUnverifiedException: Host name does not match the certificate subject provided by the peer - Stack Overflow](#)。

```
1 Exception in thread "main" javax.net.ssl.SSLPeerUnverifiedException:
  Host name 'client1.server.com' does not match the certificate subject
  provided by the peer (CN=cas.server.com, OU=Unknown, O=Unknown,
  L=Unknown, ST=Unknown, C=Unknown)
```

这里是因为 CAS 登录后会重定向到客户端 client1.server.com，可以通过如下配置忽略这个问题：

```
1 static {
2     // client = HttpClient.createDefault();
3     client = HttpClient.custom()
4         .setSSLHostnameVerifier(NoopHostnameVerifier.INSTANCE)
5         .build();
6 }
```

NoopHostnameVerifier 在 verify 方法中返回 true 忽略主机名验证。

```
1 public boolean verify(String s, SSLSession sslSession) {
2     return true;
3 }
```

关闭 HttpClient 自动重定向

simulationLogin 方法期望返回 client.server.com 响应头中的 Cookie 信息，实际执行时控制台打印如下。

```
1 cookie: Set-Cookie =>
  TGC=eyJhbGciOiJIUzUxMiJ9.ZXlKNmFYQWlPaUpFUlVZaUxDSmhiR2NpT2lKa2FYSWlMQ0p
  sYm1NaU9pSkJNVEk0UTBKRExVaFRNaUySW4wLi5lZGRGZVhZT3A1ekVscy10aTF1RFhnLm1
  wemwtdnFxMEo0UDM2cGp3SFZnMTN4cVc3UWxlUUtfSzdHNkFOUHI4TFkwRFZDUnYzby13VUE
  2V2FCMG1zamQ3OGVNcmh4MVAxTDdpdDZBMWhveUZ5dDJMZkxTTEpNbJRNhT3lHTWxERjV
  5SXZhd0xvQWlxNVZsc1R4S0dPY3J5S3FXMF9wdl9aa09TT3RoMmVuT2NLSFg4VkMzMkhiaEN
  yRHFGcWdYNlBtQ2RZdUNRMWl6aUVYVTM5X2tMdy45RVdvdVgzaVBQbEhzd2lCQlB4Q1Zn.YK
  SYFjFhm1TXH0r1Bzrb3z9-
  oSDsGECpJK7k0MeTz4oAe0F8Nd2WYii6oV1kb9Bzat4w7NhPxUxjvLmPjoe8Jg;
  Path=/cas/; Secure; HttpOnly
2 locationUrl : https://client1.server.com:7443/?ticket=ST-16-
  eYyv0XKLcocogqqtV84HVwxhuuMLindeMacBook-Pro
3 登录客户端返回错误! HTTP/1.1 200
4 cookie => null
```

“3、重定向到客户端”这一步并没有像在“认证流程”中分析的那样返回 302 状态，这是因为 HttpClient 默认自动处理重定向，可以通过 fiddler 抓包观察到这一点。200 状态的返回响应头中并没有 Cookie 信息，想要获取 Cookie 可以配置 HttpClient 不自动重定向，代码如下。

```
1 static {
2     // client = HttpClient.createDefault();
3     final RequestConfig config = RequestConfig
4         .custom()
5         .setRedirectsEnabled(false)
6         .build();
7     client = HttpClient.custom()
8         .setDefaultRequestConfig(config)
9         .setSSLHostnameVerifier(NoopHostnameVerifier.INSTANCE)
10        .build();
11 }
```

再次执行，即可获取到 Cookie 信息，如图。

```
/Library/Java/JavaVirtualMachines/openjdk-8.jdk/Contents/Home/bin/java ...
cookie: Set-Cookie => T6C=eyJhbGciOiJIUzUxMiJ9.ZXlKNmFYQWLPaUpFULVZaUxDSmhiR2NpT2lKa2FYSWLMQ0psYm1NaU9pS
LocationUrl : https://client1.server.com:7443/?ticket=ST-17-bgJYgfarTIwaanyQC-4-RkZhr0QLindeMacBook-Pro
cookie: Set-Cookie => JSESSIONID=DECD2F69F863605598466606F3C34A3; Path=/; Secure; HttpOnly
LocationUrl : https://client1.server.com:7443/?jsessionId=DECD2F69F863605598466606F3C34A3
cookie => JSESSIONID=DECD2F69F863605598466606F3C34A3; Path=/; Secure; HttpOnly
```

HttpClient 自动重定向默认不支持 POST 请求，这也是有“3、重定向到客户端”这一步的原因，HttpClient 使用 RedirectStrategy 来完成重定向的处理，默认实现类 DefaultRedirectStrategy 中只允许 GET、HEAD 方法的重定向，如果需要使用 POST 可以使用 LaxRedirectStrategy，如下。

```
1 client = HttpClients.custom()
2   .setRedirectStrategy(LaxRedirectStrategy.INSTANCE)
3   .build();
```

使用 Fiddler 观察 HttpClient 请求

正常情况下 Fiddler 是无法抓取 HttpClient 请求的，可以通过设置代理的方式，如下。

```
1 HttpHost httpHost = new HttpHost("127.0.0.1", 8866);
2 final RequestConfig config = RequestConfig.custom()
3   .setRedirectsEnabled(false)
4   .setProxy(httpHost)
5   .build();
6 client = HttpClients.custom()
7   .setDefaultRequestConfig(config)
8   .setSSLHostnameVerifier(NoopHostnameVerifier.INSTANCE)
9   .build();
```

获取 Cookie 信息

HttpClient 自动重定向时，可能无法在响应头中获取到 Cookie，而且自己解析请求头获取 Cookie 也比较繁琐。可以借助 CookieStore 来管理 Cookie，如下。

```

1 final CookieStore cookieStore = new BasicCookieStore();
2 client = HttpClients.custom()
3     .setDefaultCookieStore(cookieStore)
4     .build();
5
6 List<Cookie> cookieList = cookieStore.getCookies();
7 for (Cookie cookie1 : cookieList) {
8     System.out.println(cookie1.getName() + " => " + cookie1.getValue() + "
9     => " + cookie1.getPath());
10 }

```

打印结果如下，第一个是 client1.server.com 的 Cookie，第二个是 cas.server.com 的 Cookie。

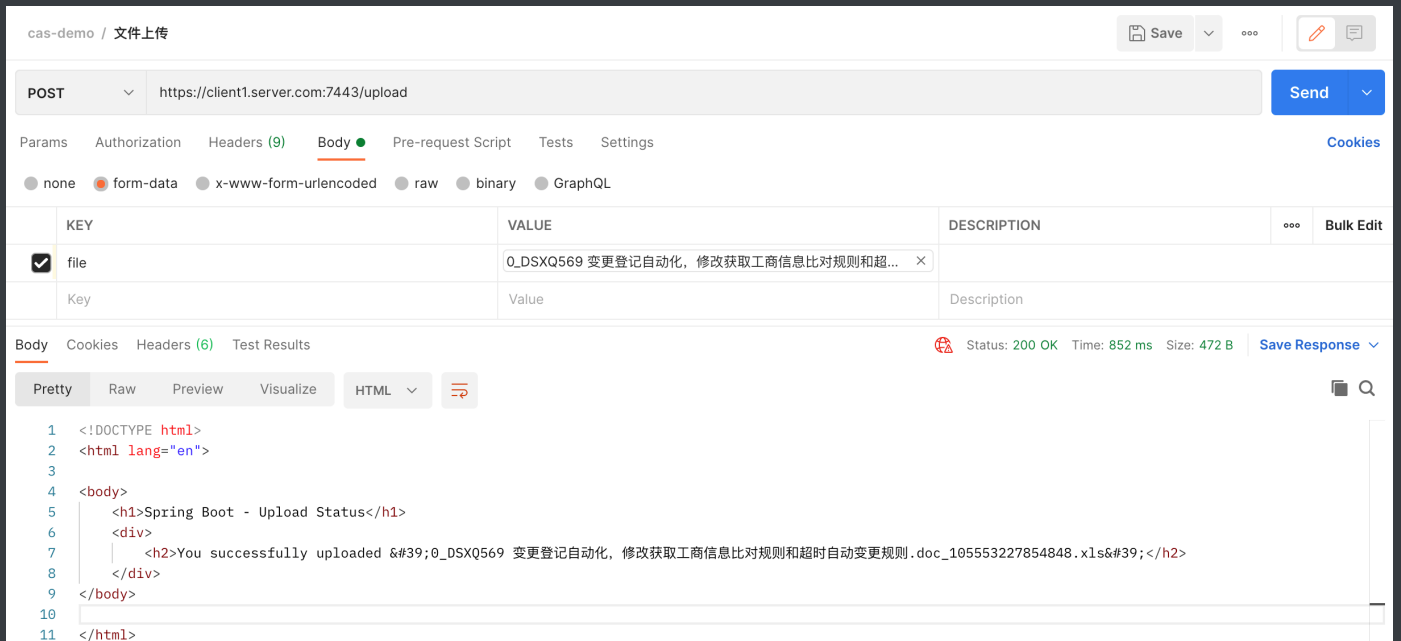
```

1 JSESSIONID => A07EF954C7591C5F1F724EFAA8141E86 => /
2 TGC =>
   eyJhbGciOiJIUzUxMiJ9.ZXlKNmFYQWlPaUpFVLVZaUxDSmhiR2NpT2lKa2FYSWlMQ0psYm1
   NaU9pSkJNVEk0UTBKRExVaFRNa1UySW4wLi4ycXMwX2lJZzYya1o1bDN5a1cyTFpnLnVjOWd
   lR2VlbjJpM3lmdTZQd25jYzZqLUxp0ThGNk5zTzE4d2NFaHBvdmI3REMtaG16TmVEbXhhNGh
   pVDJNU2FqWUdsNU5CSm16d3FfVkgzVFZCZG9rcG5HUFVBZkkyUE1LZ2pkOWthMTZYOVlfdnE
   1S3ZwdWlSdmE3QTJSSTlWM0pBS3IzS2p6aUVIUUpLQUQzLTM0Rmd3R3VQVUViak5LM3psZHA
   1UnVfRlktS0lMUmZPaFNyY2tqaF9wbGRiVS42Uy1CWTFTaUozNkN5b2tEV2VsbzVR.CsMVzX
   EYTWg5v70yh_DvosrY4mhFaQWTUqyn5Vzg2Spx3EMwhul12u9sovG8HS5lh73T5bKshw8UUt
   L82e1LnQ => /cas/

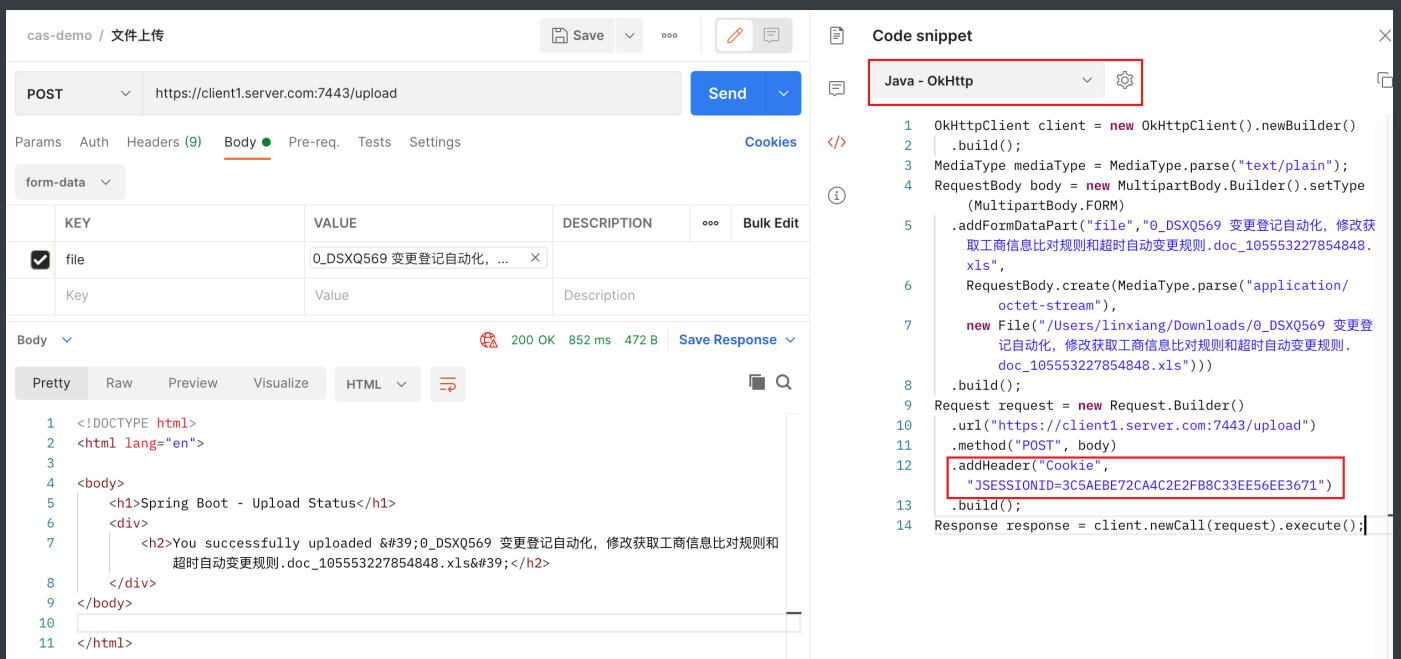
```

借助 Postman 简化编码

遇到不会写或者不愿写的请求，可以借助 Postman 的示例代码快速完成开发。比如我不会使用 HttpClient 上传文件，可以先使用 Postman 完成文件上传的验证，如图。



在右侧工具栏找到 Code 标签，Code snippet 中选择“Java - OkHttpClient”，即可复用代码，如图。



将其复制到自己的代码中，简化开发，Cookie 可以使用上面提到的方法获取，详细代码如下。

```
1 public void upload(String cookie, String filePath) throws IOException {
2   OkHttpClient client = new
3     OkHttpClient().newBuilder().hostnameVerifier(NoopHostnameVerifier.INSTANCE)
4     .build();
5   MediaType mediaType = MediaType.parse("text/plain");
6   RequestBody body = new MultipartBody.Builder().setType
7     (MultipartBody.FORM)
8     .addFormDataPart("file", filePath,
9       RequestBody.create(MediaType.parse("application/octet-stream"),
10         new File(filePath)))
11     .build();
12   Request request = new Request.Builder()
13     .url("https://client1.server.com:7443/upload")
14     .method("POST", body)
15     .addHeader("Cookie", cookie)
16     .build();
17   Response response = client.newCall(request).execute();
18 }
```

```
3     .build();
4     RequestBody body = new
    MultipartBody.Builder().setType(MultipartBody.FORM)
5     .addFormDataPart("file",
    filePath.substring(filePath.lastIndexOf("/")),
6
    RequestBody.create(MediaType.parse("application/octet-stream"),
7
                                new File(filePath)))
8     .build();
9     Request request = new Request.Builder()
10    .url(CLIENT_HOST_URL + "upload")
11    .method("POST", body)
12    .addHeader("Cookie", cookie)
13    .build();
14    Response response = client.newCall(request).execute();
15    ResponseBody responseBody = response.body();
16    if (responseBody != null) {
17        String string = responseBody.string();
18        System.out.println(string);
19    }
20    IOUtils.closeQuietly(response);
21    IOUtils.closeQuietly(responseBody);
22 }
```

配合 `simulationLogin` 可以完整的实现模拟 CAS 登录并上传文件的需求，结果如图。

```

1 public class App {
2     public static void main(String[] args) throws IOException {
3         String filePath = "/Users/linxiang/Documents/cas/cas-
demo/server.xml";
4         Simulation simulation = new Simulation();
5         String cookie = simulation.simulationLogin("xianglin",
"xianglin");
6         System.out.println("cookie => " + cookie);
7         simulation.upload(cookie, filePath);
8     }
9 }

```

```

/Library/Java/JavaVirtualMachines/openjdk-8.jdk/Contents/Home/bin/java ...
cookie: Set-Cookie => TGC=eyJhbGciOiJIUzUxMiJ9.ZXlKNmFYQWLPaUpFULVZaUxDSmhiR2NpT2lKa2FYSWLMQ0psYm1NaU9pSkJNVEk0UTBKRExVaFRNaUySW4wLi4
locationUrl : https://client1.server.com:7443/?ticket=ST-3-ifRq9VgppWwEPC9HLWH16QmwN3oLindeMacBook-Pro
cookie: Set-Cookie => JSESSIONID=6DF171D5E3D86E6A41BBFD02E5016701; Path=/; Secure; HttpOnly
locationUrl : https://client1.server.com:7443/;jsessionid=6DF171D5E3D86E6A41BBFD02E5016701
=====
JSESSIONID => 6DF171D5E3D86E6A41BBFD02E5016701 => /
TGC => eyJhbGciOiJIUzUxMiJ9.ZXlKNmFYQWLPaUpFULVZaUxDSmhiR2NpT2lKa2FYSWLMQ0psYm1NaU9pSkJNVEk0UTBKRExVaFRNaUySW4wLi4tcTVMbGNFcEJQTGtZcm
cookie => JSESSIONID=6DF171D5E3D86E6A41BBFD02E5016701; Path=/; Secure; HttpOnly
<!DOCTYPE html>
<html lang="en">
<body>
<h1>Spring Boot - Upload Status</h1>
<div>
    <h2>You successfully uploaded &#39;/server.xml&#39;</h2>
</div>
</body>
</html>

进程已结束，退出代码为 0

```