网龙华渔教育

101 统一身份管理平台
接入说明

网龙网络控股有限公司
福建省华渔教育科技有限公司

目录

概述

本文档主要作为第三方业务系统如何接入101统一身份管理平台（以下简称“IDaaS”）帮助说明，包括平台接入技术说明、接口详细介绍、各类语言开发的应用接入案例技术性指导说明。

规范说明

参考文献《SAML_2.0》：https://en.wikipedia.org/wiki/SAML_2.0

为兼容基于各种开发语言、数据库等开发的平台，本规范采取通用标准的WEB协议格式，适用于任何基于B/S架构的WEB应用系统平台与“IDaaS”进行整合，实现用户数据同步以及其他基础数据获取与交互，详见以下章节。本规范基于标准HTTP通讯协议，其中统一认证登录体系采用基于XML标准的SAML安全断言标语言（全称:Security Assertion Markup Language）进行系统间的数据交换认证。为避免各平台之间参数编码差异造成不便，本规范要求整合的各平台之间接口数据包均采用统一的UTF-8编码。各参与整合的平台需要根据自身业务需求实现本规范中要求的接口，并向“IDaaS”提供实现该接口的请求地址。“IDaaS”在对应功能操作时将自动向该整合平台的接口实现地址发送请求（如账号绑定），整合子平台自行在接口实现中处理相应业务（如登录较验session记录等）。

相关术语

名称	解释
SP	Service Provider，服务源。接入的第三方应用为用户提供的服务资源。
IDP	Identity Provider，身份源。接入的第三方应用提供用户身份的认证、授权、管理功能。
SAMLRequest	SAML认证协议的请求消息。
SAMLResponse	SAML认证协议的响应消息。
SAMLArtifact	SAML认证信息标识，在Artifact Binding模式下，用于标识替代敏感信息的传输。

接入步骤

1、接入业务向IDaaS提供应用相关信息，进行应用登记，获得应用接入所需的公钥和私钥。
2、按照SDK集成说明以及接入案例说明，在测试环镜完成接入相关开发联调。
3、完成接入联调后，将相关配置信息改为生产环境信息，发布上线。

接入登记

在进行IDaaS授权登录接入之前，接入方首先要在IDaaS平台登记应用接入，并获取SAML2.0协议签名的公钥和私钥（联调与生产环境各一对）。

接入应用需要提供以下信息：

* 应用标识 - 用于与区分应用做唯一标识区分，业务方自己定义

* 应用名称 - 应用的名称，该名称会在登陆过程中做为用户提示显示

* 应用主页 - 如http://www.nd.com.cn

* 应用Logo - 用于IDaaS应用列表中显示，格式：JPG、PNG，尺寸建议大小：600 x 400 像素，大小：200KB 以内

* 应用端信息
    * 应用端 - 可选有web端、移动端
    * clientName - 端名称
    * 认证回调地址 - IDaaS认证授权成功后，通知接入应用的地址
      该值为默认值，即客户端在构建登录请求协议时认证回调地址参数不为空，则优先使用客户端设定的值。
    * 账号绑定地址 - IDaaS账号与接入应用未绑定时，调起接入应用绑定验证的入口地址
      该值为默认值，即客户端在构建登出请求协议时登出回调地址参数不为空，则优先使用客户端设定的值。
    * 登出回调地址 - IDaaS退出成功后，通知接入应用的地址
    * 平台 - 应用端为移动端时，需要选平台参数：Android 或 iOS

* 联系方式

申请信息示意图


创建应用
基本信息
应用Logo：
上传Logo
建议上传尺寸为600x400像素logo，仅支持JPG、PNG图片，文件小于200KB
*应用名称：
应用ID：
*应用代码：
*应用Host：
新用户生成接口地址：
IDP配置
*认证类型：SAML2.0-Artifact
*应用IDP选择：其它
*IDP元数据描述：
公钥：
私钥：
应用端配置
添加端：
+ 添加web端
+ 添加移动端
web端1：
clientId：
*clientNa...
*认证回调...
*账号绑定...
*登出回调...
移动端1：
clientId：
*平台：Android
*clientNa...
*认证回调...
*账号绑定...
*登出回调...
保存
取消

IDaaS相关服务地址：

测试环境

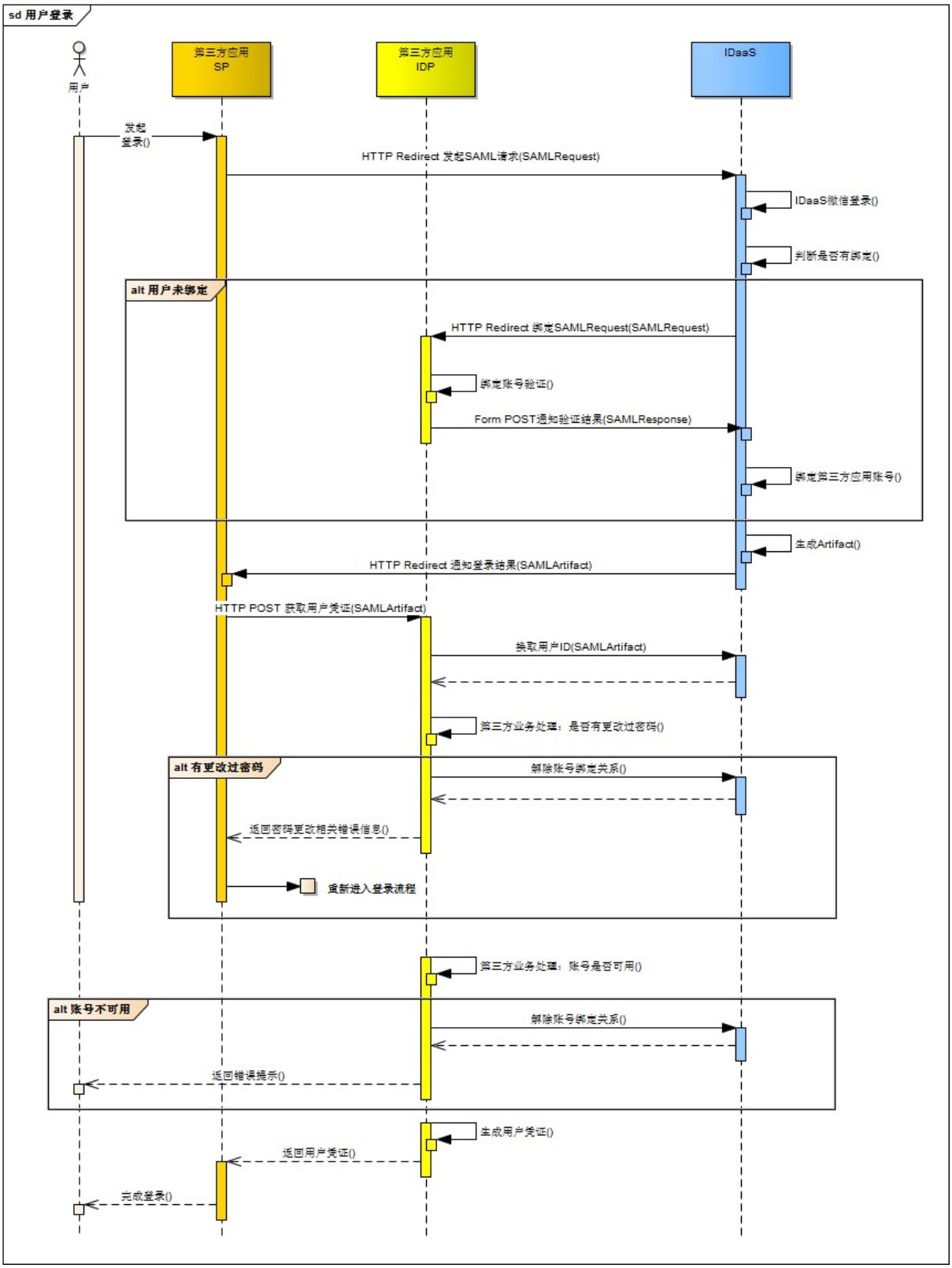
名称	说明	值
IDaaS Service	IDaaS 服务平台地址	http://idaas.beta.web.sdp.101.com/
IDaaS 登录地址	IDaaS 登录地址	http://idaas-portal.beta.101.com/bg/index.html#/index
IDaaS 登出地址	IDaaS 登出地址	http://idaas-portal.beta.101.com/bg/index.html#/logout
绑定回调地址	接入应用完成绑定验证后，回调通知IDaaS的地址	http://idaas-portal.beta.101.com/idp/response

生产环境

名称	说明	值
IDaaS Service	IDaaS 服务平台地址	http://idaas.sdp.101.com/
IDaaS 登录地址	IDaaS 登录地址	http://idaas-portal.sdp.101.com/bg/index.html#/index
IDaaS 登出地址	IDaaS 登出地址	http://idaas-portal.sdp.101.com/bg/index.html#/logout
绑定回调地址	接入应用完成绑定验证后，回调通知IDaaS的地址	http://idaas-portal.sdp.101.com/idp/response

用户登录集成

接入概要时序



接入方需要完成的开发事项：

登录主流程接入

1、第三方应用SP发起SAMLRequest请求，HTTP Redirect 到 IDaaS：

```
如：http://idaas-portal.beta.101.com/bg/index.html#/index?SAMLRequest={SAMLRequest}&pro
duct={product}[&lang={lang}]

* SAMLRequest : SAML请求协议报文件（必要参数），组成要素：

    Signature : 协议签名
    Issuser : 应用申请成功后，对应的clientId
    ProtocolBinding : SP接收IDaaS登录认证成功返回协议地址，如：http://[接入应用域名]/idaasS
uccess

* product : 应用所属的产品代码（必要参数）

  lang : 语言参数（可选参数），可选值：zh、en
```

2、第三方应用SP接收IDaaS登认证成功返回协议

```
从IDaaS返回的协议是以 HTTP Redirect方式返回，参数为SAMLArtifact，如：

http://[接入应用域名]/idaasSuccess?SAMLArtifact=xxxxx
```

3、第三方应用SP使用SAMLArtifact作为参数，HTTP POST请求第三方平台IDP服务。

4、第三方应用IDP服务接收SAMLArtifact，发送HTTP POST请求到IDaaS换取IDP用户ID，IDP服务为获取到的用户ID生成登录凭证，返回给SP。

5、第三方应用IDP服务验证用户密码是否发生过变更，如果发生过变更，调用SDK方法，解除绑定关系，通过第三方应用SP，重新发起登录流程，引导用户验证新密码。（可选步骤）

6、第三方应用IDP服务验证用户账号是否为可用状态，如果账号不可用，调用SDK方法，解除绑定关系，并提示用户账号被禁用。（可选步骤）

注：第三方应用也可以参照步骤**5**、**6**，增加其他自定义逻辑处理，根据业务情况，判断是否需要调用**SDK**方法，解除用户账号绑定关系。

7、第三方应用SP收到IDP的用户凭证信息后完成后续处理（如生成cookie、session或local storage等）。

账号绑定

账号绑定为登录流程的一个异常子流程。

1、向IDaaS提交第三方应用IDP实现的绑定验证模块地址，如：http://[接入应用域名]/idaasLogin

2、第三方应用SP发起登录请求，IDaaS完成用户微信登录后，如果发现IDaaS账号未与第三方应用账号进行过绑定，IDaaS发起绑定请求到第三方平台IDP。请求为HTTP Redirect，参数为SAMLRequest，如：

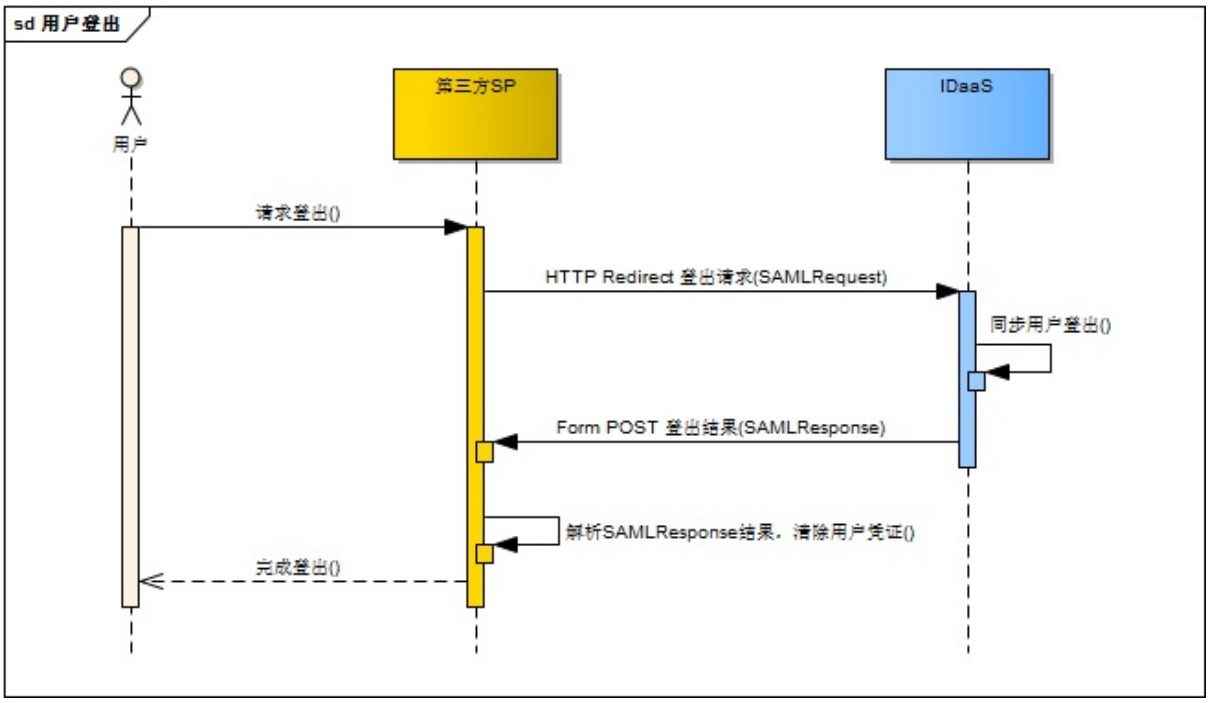
```
http://[接入应用域名]/idaasLogin?SAMLRequest=...
```

3、第三方应用IDP接收SAMLRequest，校验签名合法后，显示用户登录绑定界面。用户输入账号信息提交后，IDP完成账号信息校验。生成SAMLResponse，以FORM POST方式，回调通知IDaaS。

```
FORM表单格式：

<form id="idpResponse" method="post" action="http://idaas-portal.beta.101.com/idp/respo
nse" ...>
  <input type="hidden" name="SAMLResponse" value="..." />
  <input type="hidden" name="RelayState" value="..." />
</form>
<script>
  document.getElementById("idpResponse").submit();
</script>
```

用户登出集成

登出流程：



需要实现的功能：

1、第三方应用SP组合登出请求，HTTP Redirect 到IDaaS，参数为SAMLRequest，如：

```
http://idaas-portal.beta.101.com/bg/index.html#/logout?SAMLRequest=...

SAMLRequest协议要素：
    Signature ： 协议签名
    Issuser ： 申请应用时提交的应用标识
    ProtocolBinding ： SP接收IDaaS登出成功返回协议地址，如：http://[接入应用域名]/idaasLogo
utSuccess
```

2、第三方应用SP接收IDaaS登出成功通知，通知方式为Form POST，参数有SAMLResponse，ReplayState如：

```
<form id="logoutForm" action="http://[接入应用域名]/idaasLogoutSuccess" method="POST">

   <input type="hidden" name="SAMLResponse" value="..." />

   <input type="hidden" name="ReplayState" value="..." />
```

```
</form>;
```

第三方应用解析SAMLResponse参数后，得到第三方应用IDP的用户ID，与当前会话的用户ID进行对比，如果相同，则清除会话，完成退出登录操作。

JAVA SDK 集成

目前IDaaS提供了JAVA语言SDK开发包，JDK版本要求1.7及以上。

获取方式：

下载包：

下载地址：http://cdncs.101.com/v0.1/static/idaas_public/idaas_saml2_sdk-0.5.1.1-20180425.083450-13.jar?serviceName=idaas_public&attachment=true

Maven依赖：

```
        <dependency>
            <groupId>com.nd.idaas</groupId>
            <artifactId>idaas_saml2_sdk</artifactId>
            <version>0.6.1-SNAPSHOT</version>
            <scope>system</scope>
            <systemPath>${basedir}/lib/idaas_saml2_sdk-0.6.1-20180524.075656-5.jar</sys
temPath>
        </dependency>
```

结构说明

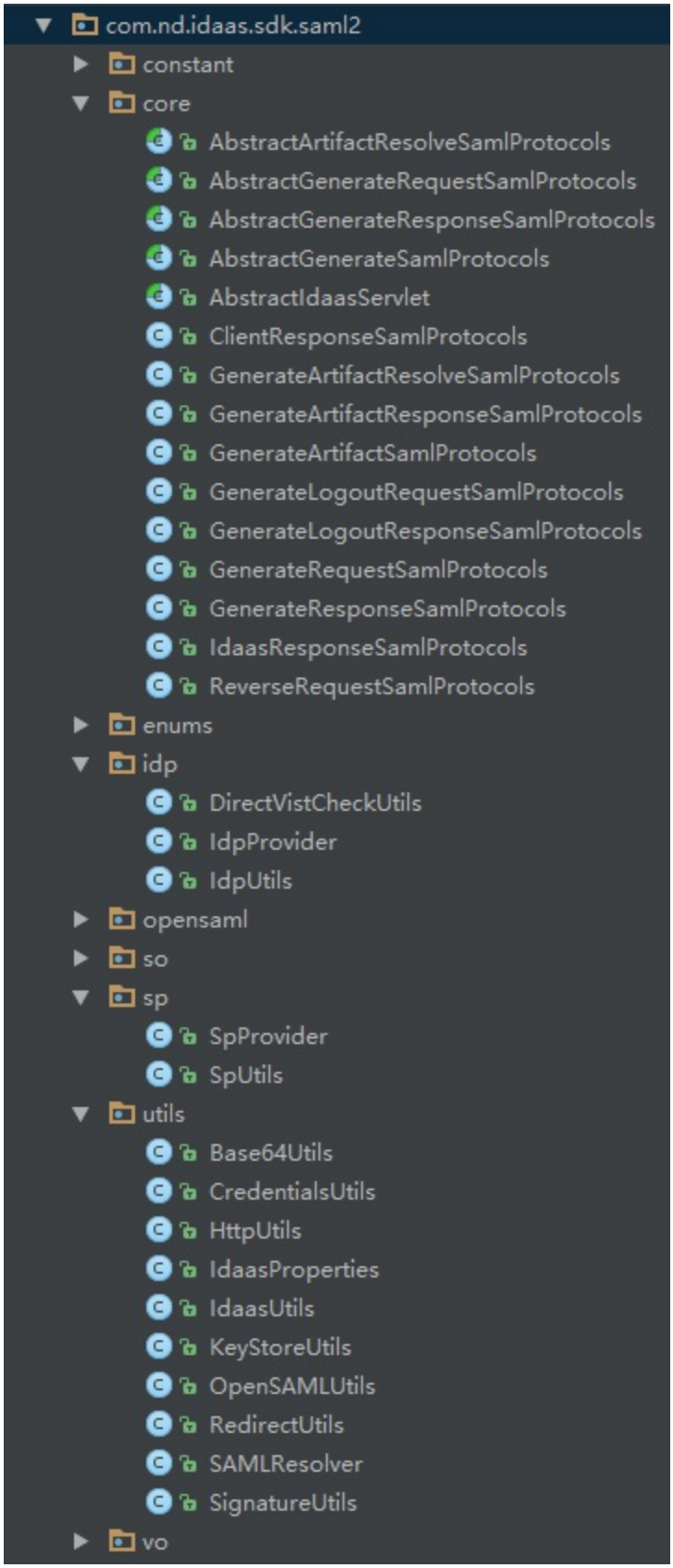
SDK依赖库

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
        <!-- openSAML -->
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-core</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-saml-api</artifactId>
            <version>${opensaml.version}</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-saml-impl</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-messaging-api</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-messaging-impl</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-soap-api</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.opensaml</groupId>
            <artifactId>opensaml-soap-impl</artifactId>
            <version>${opensaml.version}</version>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>1.9.5</version>
            <scope>test</scope>
        </dependency>

        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.18</version>
            <scope>provided</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.7</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
        <dependency>
            <groupId>com.alibaba</groupId>
```

```
            <artifactId>fastjson</artifactId>
            <version>1.2.40</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>dom4j</groupId>
            <artifactId>dom4j</artifactId>
            <version>1.6.1</version>
        </dependency>
```

目录结构



关键类、方法说明

com.nd.idaas.sdk.saml2.core

com.nd.idaas.sdk.saml2.core.GenerateRequestSamlProtocols ：用户登录请求生成SAMLRequest报文的协议类。

com.nd.idaas.sdk.saml2.core.GenerateArtifactResolveSamlProtocols ：Artifact模式下，通过IDaaS返回的Artifact向IDaaS兑换真实用户数据的协议类。

com.nd.idaas.sdk.saml2.core.GenerateLogoutRequestSamlProtocols：用户登出请求SAMLRequest报文的协议类。

com.nd.idaas.sdk.saml2.core.IdaasResponseSamlProtocols：生成用户绑定成功SAMLResponse报文的协议类。

com.nd.idaas.sdk.saml2.core.GenerateResponseSamlProtocols：通过从IDaaS请求到的用户凭证，向IDP兑换token等信息的协议类。

com.nd.idaas.sdk.saml2.core.ReverseRequestSamlProtocols：用户反向登录请求生成SAML报文的协议类。当第三方系统想通过自身已登录的用户访问IDaaS其他应用时使用。

com.nd.idaas.sdk.saml2.utils.RedirectUtils： 将**Protocols**进行编码重组的工具类

主要方法如下：

```
getUrlSignedRequestEncodedUrl：根据Request类协议构造的进行了参数数字签名的可供跳转的URL
redirectUrlSignedRequest：在getUrlSignedRequestEncodedUrl方法的基础上自动执行重定向操作
getArtifactUrl：根据Artifact类协议构造的可供跳转的URL
getUrlUnsignedResponseEncodedMessage：根据Response类协议构造的未进行参数数字签名的SAML加密参数
```

com.nd.idaas.sdk.saml2.utils.IdaasProperties：IDaaS私有properties读取工具类。

com.nd.idaas.sdk.saml2.utils.KeyStoreUtils：密钥辅助工具类。

com.nd.idaas.sdk.saml2.utils.IdaasUtils：Idaas辅助工具类。

com.nd.idaas.sdk.saml2.utils.SignatureUtils：数字签名辅助工具类。

com.nd.idaas.sdk.saml2.utils.SAMLResolver：将**SAML**请求进行解码读取有效信息的工具类

主要方法如下：

```
decodeUrlSignedResponse：解码进行了参数数字签名的请求，并得到SAMLResponse对象
decodeUrlUnsignedResponseEncodedMessage：解码未进行参数数字签名的SAML加密参数，并得到SAMLResponse对象
decodeUrlUnsignedResponse：解码未进行参数数字签名的请求，并得到SAMLResponse对象
decodeUrlSignedRequestEncodedUrl：解码进行了参数数字签名的URL，并得到SAMLRequest对象
decodeUrlSignedRequest：解码进行了参数数字签名的请求，并得到SAMLRequest对象
decodeAssertion：解码加密的断言，并得到Assertion对象
```

com.nd.idaas.sdk.saml2.sp：承担**sp**角色的服务端主要使用的方法

com.nd.idaas.sdk.saml2.sp.SpProvider：sp服务配置提供者

```
init：服务配置初始化
```

com.nd.idaas.sdk.saml2.sp.SpUtils：sp服务工具类，进一步封装了utils包里的部分方法

```
buildLoginRequestProtocol：构建登录请求协议
buildReverseLoginRequestProtocol：构建反向登录请求协议
buildArtifactResolveProtocol：构建Artifact兑换协议
buildLogoutRequestProtocol：构建登出请求协议
buildExchangeTokenResponseProtocol：构建向idp兑换token的协议
isStatusSuccessful：判断response类的响应码是否正确
resolveArtifactResponse：解析Artifact模式下从IDaaS获取的ArtifactResponse
resolveResponse：解析Response
resolveLogoutResponse：解析登出响应的返回信息
```

com.nd.idaas.sdk.saml2.idp：承担**idp**角色的服务端主要使用的方法

com.nd.idaas.sdk.saml2.idp.IdpProvider：idp服务配置提供者

```
init：服务配置初始化
```

com.nd.idaas.sdk.saml2.idp.IdpUtils：idp服务工具类，进一步封装了utils包里的部分方法

```
resolveResponse：解析response
verifyUrlSignedRequest：校验进行了参数数字签名的URL，并得到SAMLRequest
buildIdaasResponseProtocol：构建返回给idaas的响应协议
```

DEMO 下载

下载地址：http://cs.101.com/v0.1/static/idaas_public/idaas-demo-20180502-3922.rar?serviceName=idaas_public&attachment=true

接入案例**1**：前后端一体

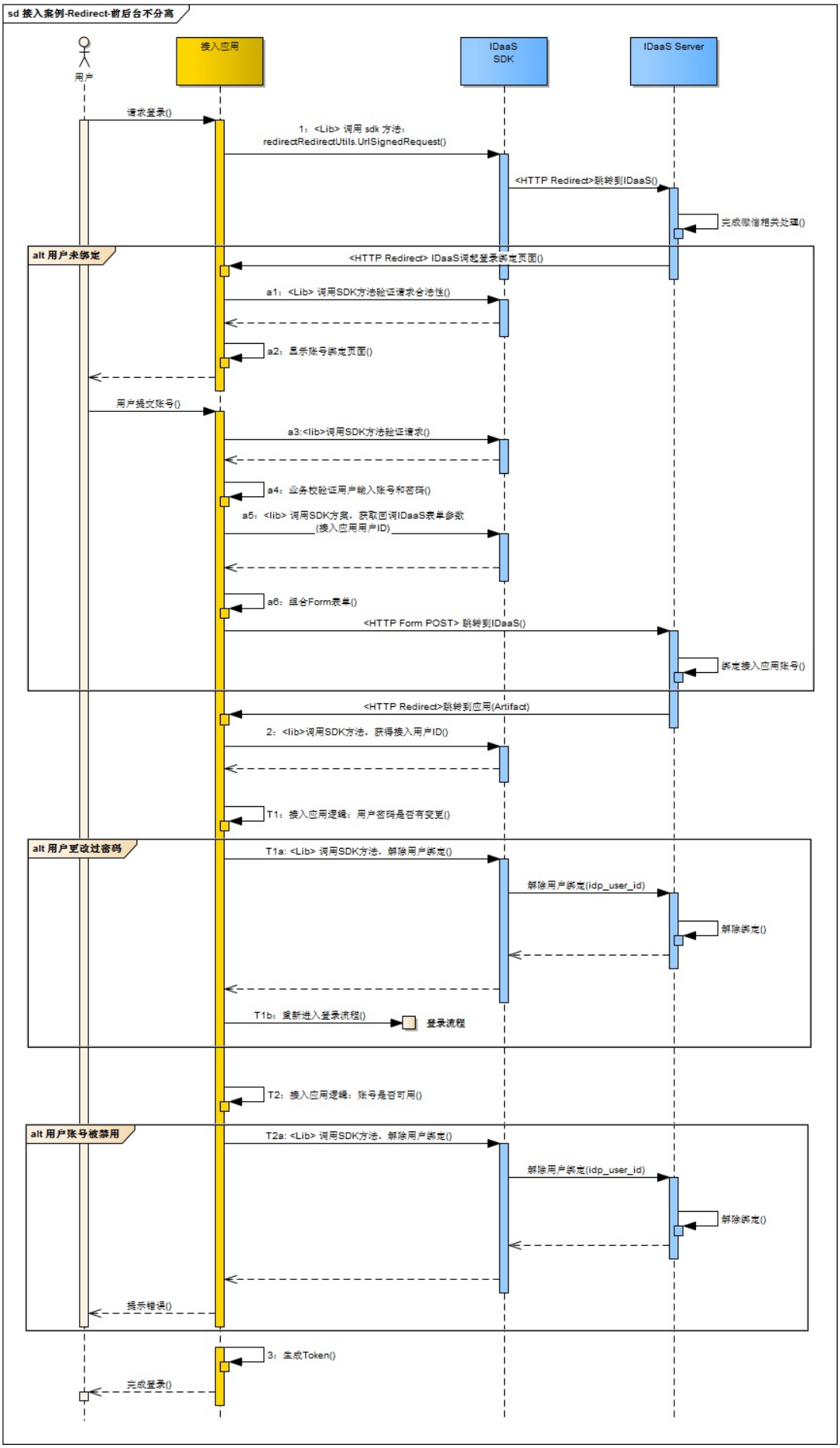
目前IDaaS实现了SAML的Redirect-Binding和Artifact-Binding两种模式，接入业务可以按照自己的需要，选择一种模式接入。

前后端一体接入案例

- 绑定模式：Artifact-Binding

用户登录

登录认证时序图


sd 接入案例-Redirect-前后台不分离
用户
接入应用
IDaaS SDK
IDaaS Server
请求登录()
1：<Lib> 调用 sdk 方法：redirectRedirectUtils.UrlSignedRequest()
<HTTP Redirect>跳转到IDaaS()
完成微信相关处理()
alt 用户未绑定
<HTTP Redirect> IDaaS调起登录绑定页面()
a1：<Lib> 调用SDK方法验证请求合法性()
a2：显示账号绑定页面()
用户提交账号()
a3:<lib>调用SDK方法验证请求()
a4：业务校验证用户输入账号和密码()
a5：<lib> 调用SDK方案，获取回调IDaaS表单参数(接入应用用户ID)
a6：组合Form表单()
<HTTP Form POST> 跳转到IDaaS()
绑定接入应用账号()
<HTTP Redirect>跳转到应用(Artifact)
2：<lib>调用SDK方法，获得接入用户ID()
T1：接入应用逻辑：用户密码是否有变更()
alt 用户更改过密码
T1a: <Lib> 调用SDK方法，解除用户绑定()
解除用户绑定(idp_user_id)
解除绑定()
T1b：重新进入登录流程()
登录流程
T2：接入应用逻辑：账号是否可用()
alt 用户账号被禁用
T2a: <Lib> 调用SDK方法，解除用户绑定()
解除用户绑定(idp_user_id)
解除绑定()
提示错误()
3：生成Token()
完成登录()

注册配置信息

调用方法SpProvider.init()和IdpProvider.init()方法，注册配置信息：

```
{
    // 服务初始化。作为SP服务端，所有IDaaS服务使用的外部依赖参数均通过该方法注入
    // init最后的参数支持传多个app_client_id，即支持单应用多客户端部署在同一项目中，sdk的方法也提供了根据传参顺序选择使用哪个client配置
    SpProvider.init(DebugConfig.PUBLIC_KEY, DebugConfig.PRIVATE_KEY, DebugConfig.IDAAS_HOST,
                DebugConfig.IDAAS_WEB_HOST, DebugConfig.APP_CLIENT_ID, DebugConfig.APP_MOBILE_CLIENT_ID);

    // 注册idp：无用户解绑需求
    // IdpProvider.init(DebugConfig.IDAAS_HOST, DebugConfig.IDAAS_WEB_HOST);
    // 注册idp：有用户解绑需求
    IdpProvider.init(DebugConfig.APP_CLIENT_ID, DebugConfig.PRIVATE_KEY, DebugConfig.IDAAS_HOST, DebugConfig.IDAAS_WEB_HOST);
}
```

SpProvider.init()参数说明：

```
/**
 * 初始化sp相关设置.
 * <p>Description: sp功能使用前需调用该方法进行初始化操作</p>
 * <p>Create Time: 2017/12/5 0005 10:30 </p>
 * @author 910204
 * @param appClientId 注册idaas时分配的客户端ID.
 * @param publicKey idaas分配的公钥.
 * @param privateKey idaas分配的私钥.
 * @param serviceHost idaas服务端host.
 * @param webServiceHost idaasWeb端host.
 */
public static void init(String publicKey, String privateKey, String serviceHost, String
 webServiceHost, String... appClientId) {
    if (isInit) {
        return;
    }
    initEnvironment();
    idaasAppClientIds = appClientId;
    spPublicKey = publicKey;
    spPrivateKey = privateKey;
    idaasHost = serviceHost;
    idaasWebHost = webServiceHost;
    isInit = true;
}
```

IdpProvider.init()参数说明：

```
public static void init(String issue, String privateKey, String serviceHost, String web
ServiceHost) {
    if (isInit) {
        return;
    }
    initEnvironment();
    idaasHost = serviceHost;
    idaasWebHost = webServiceHost;
    isInit = true;
    appIssue = issue;
    appPrivateKey = privateKey;
}
```

主干步骤说明

1：第三方应用收到用户登录请求，调用SDK方法： SpUtils.buildLoginRequestProtocol()构建协议，通过SDK方法RedirectUtils.redirectUrlSignedRequest()生成请求的完整URL并重定向到IDaaS。示例代码：

```
@RequestMapping(value = "/launch", method = RequestMethod.GET)
public void spStep1(HttpServletResponse httpServletResponse) {
    // 构建请求协议
    GenerateRequestSamlProtocols protocols = SpUtils.buildLoginRequestProtocol("http://
spResponseUri");
    // 生成请求的完整URL
    String idaasRequestUriFull = RedirectUtils.getUrlSignedRequestEncodedUrl(protocols)
;
    // 拼接产品编码
    String retUri = new StringBuilder(idpRequestUriFull).append("&product=").append(Deb
ugConfig.PRODUCT_AUTO_ID)
                .toString();
    // ================================================
    // TODO : 接入应用完成后续处理
    // ================================================

}
```

2：IDaaS处理用户登录成功后，会回调通知第三方应用。第三方应用收到通知后，调用SDK方法：使用SpUtils.buildArtifactResolveProtocol()验证请求有效性、使用IdaasUtils.soapGetArtifactResponse()方法soap请求idaas，兑换IDP用户信息、使用SpUtils.resolveArtifactResponse()解析ArtifactResponse，获得第三方应用用户ID。示例代码：

```
@RequestMapping(value = "/sp_response_uri", method = RequestMethod.GET)
```

```
public Object spStep4(@RequestParam(value = "SAMLArtifact") String samlArtifact, HttpSe
rvletResponse response) throws Exception {
    // 校验artifact，兑换真实idp token，若是UC登录模式则直接获取token// STEP 4.1.3.1.
    // artifact模式下，通过artifact兑换idpUserId（若IDP为UC，则同时返回UC token）
    SAMLResponse samlResponseObj = this.exchangeArtifact(samlArtifact, response);
    // 构建请求自建IDP的协议
    String idpUserId = samlResponseObj.getSubjectForNameID()));
    // ============================================================
    // TODO : 接入应用为该用户生成相关授权信息以及无成后续步骤。
    // ============================================================
}
```

3：第三方应用根据步骤2获得的用户ID，生成用户登录凭证，完成登录。

可选步骤说明

T1：第三方应用验证用户密码是否发生过变更，如果发生过变更。 T1a：如果T1结果为用户更改过密码，那么调用SDK方法：IdpUtils.unbind(String idpUserId)，解除用户绑定。 T1b：T1a操作成功后，重新进入登录流程，即进入步骤1。

T2：第三方应用验证用户账号是否为可用状态 T2a：如果T2结果为账号不可用，那么调用SDK方法：IdpUtils.unbind(String idpUserId)，解除用户绑定，并提示用户相应错误，流程终止。

注：第三方应用也可以参照步骤**T1**、**T2**，增加其他自定义逻辑处理，根据业务情况，判断是否需要调用**SDK**方法，解除用户账号绑定关系，和自定义相关用户提示信息。

示例代码：

```
String privateKey = "MIIEvQIBADANBgkqhki...";
IdpProvider.init("fjjyxinxihua2017", privateKey, "http://localhost:8081", "http:xxxx");
try {
    IdpUtils.unbind("vcnlwokrqxonkwa");
} catch (Exception e) {
    System.out.print(e);
}
```

异常分支步骤说明**(**第三方应用用户账号与**IDaaS**绑定**)**

a1：IDaaS收到第三方应用登录请求（步骤1），完成用户在IDaaS登录（微信登录）后，如果IDaaS判断到该用户未与第三方应用的账号进行过绑定，那么IDaaS会向第三方应用发起授权绑定请求。第三方应用接收到请求后，调用SDK方法IdpUtils.verifyUrlSignedRequest()，验证请求的有效性：

```
@RequestMapping(value = "/bind_uri", method = RequestMethod.GET)
public Object spStep2(HttpServletRequest request) {
    // 校验请求参数是否合法
    try {
```

```
        SAMLRequest samlRequestObj = IdpUtils.verifyUrlSignedRequest(request, DebugConf
ig.APP_CLIENT_ID);
    } catch (Exception e {
        // ========================================
        // TODO: 校验异常，接入业务根据自身需要进行相关处理
        // ========================================
    }

    // ================================
    // TODO：校验通过，正常显示登录界面
    // ================================
    return null;
}
```

a2：接a1，第三方应用显示用户认证绑定界面，如：

a3：用户在界面输出第三方应用账号信息并提交后，第三方应用调用SDK方法IdpUtils.verifyUrlSignedRequest()验证合法性。

a4：接步骤a3，第三方应用校验用户提交的账号信息是否正确，如果不正确，抛出相应异常提示。

a5：第三方应用调用SDK方法RedirectUtils.getUrlUnsignedResponseEncodedMessage()，获得组合IDaaS回调表单的参数数据：

```
@RequestMapping(value = "/bind_uri", method = RequestMethod.POST)
public Object login(HttpServletRequest request, HttpServletResponse response, @RequestB
ody JSONObject loginBody)
        throws Exception {
    // 校验请求参数是否合法:接入应用根据情况选择是否进行二次校验
    // SAMLRequest samlRequest = IdpUtils.verifyUrlSignedRequest(request, DebugConfig.A
```

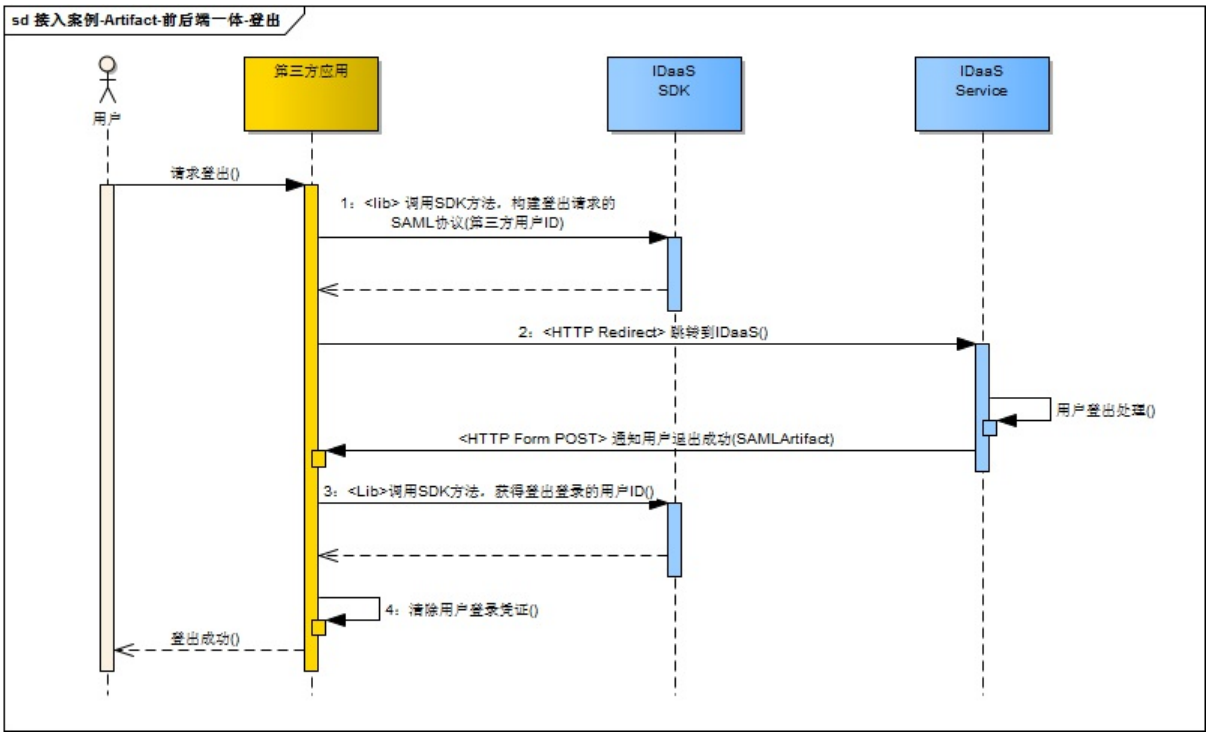
```
PP_CLIENT_ID);
    // ====================================================
    // TODO:idp校验账密，接入应用根据自身系统修改下面校验代码
    String idpUserId = DebugConfig.getIDaaSUserIdByLoginName(loginBody.getString("login
_name"));
    // ====================================================

    // 构建协议
    IdaasResponseSamlProtocols protocols = IdpUtils.buildIdaasResponseProtocol(samlRequ
est, idpUserId);
    // 生成saml协议密文
    String urlEncodeSaml = URLEncoder.encode(RedirectUtils.getUrlUnsignedResponseEncode
dMessage(protocols),
            "UTF-8");
    JSONObject retObj = new JSONObject();
    retObj.put("idp_response_target", DebugConfig.IDAAS_WEB_HOST + IdaasWebServiceUri.I
DP_LOGIN_RESPONSE_PATH);
    retObj.put("saml_response", urlEncodeSaml);
    retObj.put("product", DebugConfig.PRODUCT_AUTO_ID);
    return retObj;
}
```

a6：第三方应用组合Form表单，并触发POST提交到IDaaS，完成绑定授权。Form表单组合提交示例(详细请参考DEMO)：

```
<html>
<head>
    <title>第一个 JSP 程序</title>
</head>
<body>
<form action="${idp_response_target!}" method="POST" id="idaasForm">
    <input type="hidden" name="SAMLResponse" value="${saml_response!}" />
    <input type="hidden" name="product" value="${product}" />
</form>
<script type="application/javascript">
    document.getElementById("idaasForm").submit();
</script>
</body>
</html>
```

用户登出

用户登出时序图



步骤说明

1：第三方应用收到用户登出请求后，调用SDK方法SpUtils.buildLogoutRequestProtocol()，生成IDaaS登出请求地址：

```
@RequestMapping(value = "/sp/logout/step_1", method = RequestMethod.GET)
    public Object spLogoutStep1() {
        GenerateLogoutRequestSamlProtocols protocols = SpUtils.buildLogoutRequestProtoc
ol("http://spResponseUri", "idpUserId");
        // 生成请求的完整URL
        String idpLogoutRequestUriFull = RedirectUtils.getUrlSignedRequestEncodedUrl(pr
otocols);
        // TODO:跳转
    }
```

其中，“http://spResponeUri”位置处理为IDaaS完成登出请求响应后的回调地址，其优先级高于IDaaS应用登记时配置的默认回调地址。

2：第三方应用跳转至步骤1获得的IDaaS登出请求地址。

3：IDaaS处理完登出后，会回调通知第三方应用。第三方应用接到收请求后，调用SDK方法SpUtils.resolveLogoutResponse()获得第三方用户ID：

```
@RequestMapping(value = "/sp/logout/step_2", method = RequestMethod.POST)
public Object resolveLogoutResponse(HttpServletRequest request, HttpServletResponse htt
pServletResponse) throws Exception{
    SAMLResponse samlResponse = SpUtils.resolveLogoutResponse(request);
```

```
    String idpUserId = samlResponse.getSubjectForNameID();
    //TODO：清除用户凭证
}
```

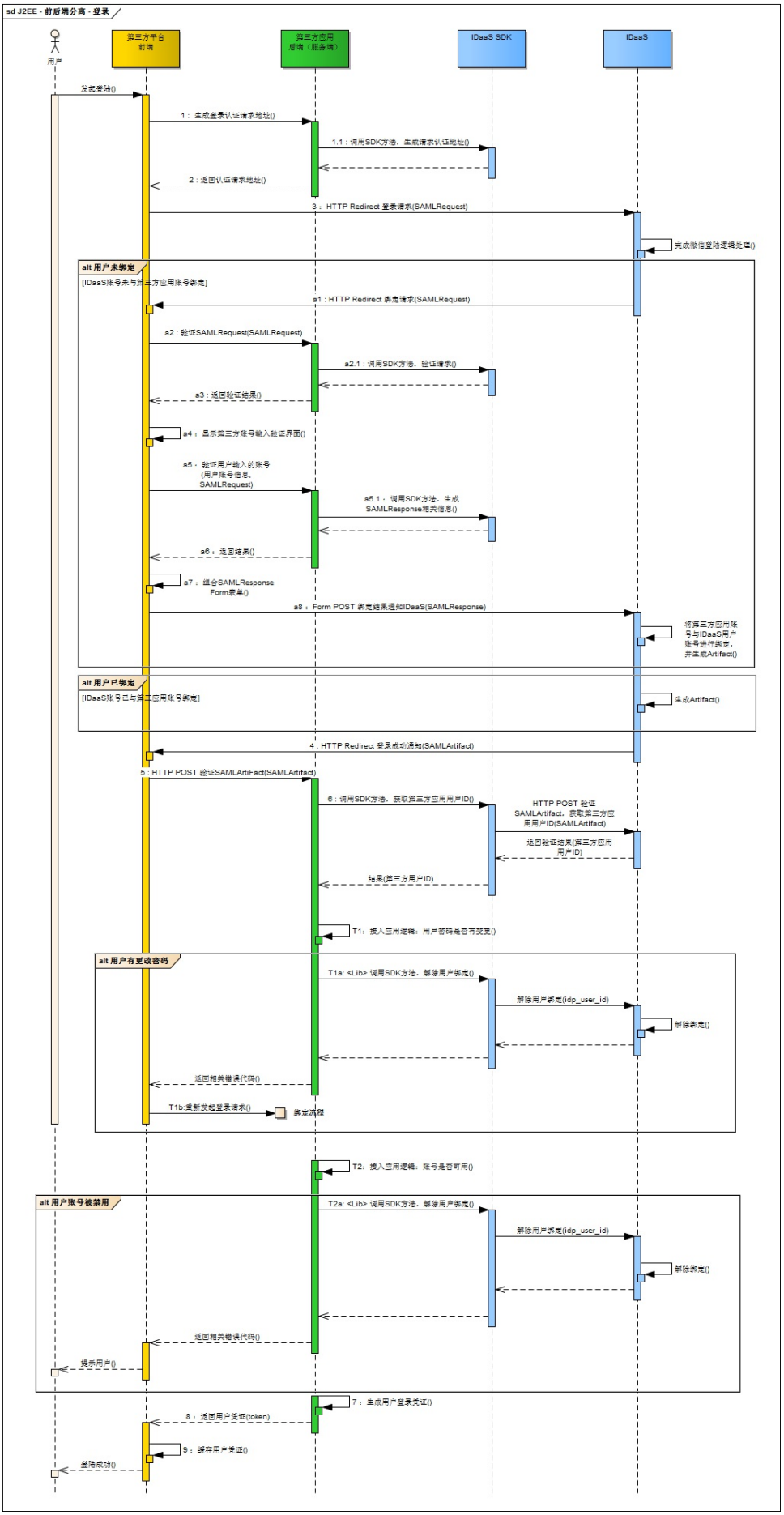
4：第三方应用根据步骤3获得的用户ID，清除用户登录凭证信息，完成用户登出。

接入案例**2**：前后端分离

- 绑定模式：Artifact-Binding

用户正向登录

正向登录认证时序图


sd J2EE - 前后端分离 - 登录
用户
第三方平台前端
第三方应用后端（服务端）
IDaaS SDK
IDaaS
发起登陆()
1：生成登录认证请求地址()
1.1：调用SDK方法，生成请求认证地址()
2：返回认证请求地址()
3：HTTP Redirect 登录请求(SAMLRequest)
完成微信登陆逻辑处理()
alt 用户未绑定
[IDaaS账号未与第三方应用账号绑定]
a1：HTTP Redirect 绑定请求(SAMLRequest)
a2：验证SAMLRequest(SAMLRequest)
a2.1：调用SDK方法，验证请求()
a3：返回验证结果()
a4：显示第三方账号输入验证界面()
a5：验证用户输入的账号（用户账号信息，SAMLRequest)
a5.1：调用SDK方法，生成SAMLResponse相关信息()
a6：返回结果()
a7：组合SAMLResponse Form表单()
a8：Form POST 绑定结果通知IDaaS(SAMLResponse)
将第三方应用账号与IDaaS用户账号进行绑定，并生成Artifact()
alt 用户已绑定
[IDaaS账号已与第三应用账号绑定]
生成Artifact()
4：HTTP Redirect 登录成功通知(SAMLArtifact)
5：HTTP POST 验证SAMLArtiFact(SAMLArtifact)
6：调用SDK方法，获取第三方应用用户ID()
HTTP POST 验证SAMLArtifact，获取第三方应用用户ID(SAMLArtifact)
返回验证结果(第三方应用用户ID)
结果(第三方用户ID)
T1：接入应用逻辑：用户密码是否有变更()
alt 用户有更改密码
T1a: <Lib> 调用SDK方法，解除用户绑定()
解除用户绑定(idp_user_id)
解除绑定()
返回相关错误代码()
T1b:重新发起登录请求()
绑定流程
T2：接入应用逻辑：账号是否可用()
alt 用户账号被禁用
T2a: <Lib> 调用SDK方法，解除用户绑定()
解除用户绑定(idp_user_id)
解除绑定()
返回相关错误代码()
提示用户()
7：生成用户登录凭证()
8：返回用户凭证(token)
9：缓存用户凭证()
登陆成功()

主干步骤说明：

1：第三方应用前端通过HTTP POST(Ajax)向第三方应用服务端获取登录认证请求地址。

2：第三方应用服务端 调用IDaaS SDK方法，生成登录认请求地址，并返回给前端：

```
{
    // 服务初始化。作为SP服务端，所有IDaaS服务使用的外部依赖参数均通过该方法注入
    // init最后的参数支持传多个app_client_id，即支持单应用多客户端部署在同一项目中，sdk的方法也提供了根据传参顺序选择使用哪个client配置
    SpProvider.init(DebugConfig.PUBLIC_KEY, DebugConfig.PRIVATE_KEY, DebugConfig.IDAAS_HOST,
            DebugConfig.IDAAS_WEB_HOST, DebugConfig.APP_CLIENT_ID,DebugConfig.APP_MOBILE_CLIENT_ID);
}

/**
 * DebugConfig.PUBLIC_KEY : 公钥
 * DebugConfig.PRIVATE_KEY : 私钥
 * DebugConfig.IDAAS_HOST : IDaaS Service服务地址
 * DebugConfig.IDAAS_WEB_HOST : IDaaS Web组件地址
 * DebugConfig.APP_CLIENT_ID : 接入应用端clientId
 */

/**
 * 生成SAMLRequest的接口.
 *
 * @return CommonResult common result
 */
@RequestMapping(value = "/launch", method = RequestMethod.GET)
public CommonResult launch(@RequestParam(value = "org_type", required = false, defaultValue = "1") Integer orgType,
        @RequestParam(value = "org_mark", required = false, defaultValue = "nd") String orgMark) {
    // 构建登录请求协议，参数为请求IDaaS完成绑定后回调SP服务端的地址
    GenerateRequestSamlProtocols protocols = SpUtils
            .buildLoginRequestProtocol(DebugConfig.SP_CLIENT_HOST + "/sp/sp_response_uri");
    // 生成请求的完整URL
    String idpRequestUriFull = RedirectUtils.getUrlSignedRequestEncodedUrl(protocols);
    // 拼接产品编码
    String retUri = new StringBuilder(idpRequestUriFull).append("&product=").append(DebugConfig.PRODUCT_AUTO_ID)
                .toString();
    // ================================================
    // TODO : 接入应用完成后续处理
    // ================================================
```

```
}
```

3：第三方应用前端重定向到步骤2获得的登录请求地址。

4：第三方应用前端接收登录成功通知，接收参数SAMLArtifact。

> 此地址与步骤2请求参数sp_response_uri地址一致。

5：第三方应用前端HTTP POST(Ajax)向第三方应用服务端 获取登录凭证，请求参数为SAMLArtifact。

6-8：第三方应用服务端调用IDaaS SDK，验证SAMLArtifact，换取用户ID，并为换取的用户ID生成登录凭证，并返回给前端：

```
/**
 * 代理解析IDaaS返回的SAMLArtifact
 */
@RequestMapping(value = "/101uc/sp/saml/response", method = RequestMethod.GET)
public JSONObject resolveArtifact(HttpServletRequest request, HttpServletResponse respo
nse) {
    try {
        // 校验artifact，兑换真实idp token，若是UC登录模式则直接获取token// STEP 4.1.3.1.
        // artifact模式下，通过artifact兑换idpUserId（若IDP为UC，则同时返回UC token）
        SAMLResponse samlResponseObj = this.exchangeArtifact(samlArtifact, response);
        // ================================================
        // TODO : 接入应用根据获得的接入应用用户ID完成授权后续处理
        String idpUserId = samlResponseObj.getSubjectForNameID();
        // ================================================
    } catch (Exception e) {
        //================================================
        // TODO : 接入应用根据自身实际情况，处理异常
        // ================================================
    }
}

//http://idpExchangeTokenUri：sp与idp约定的兑换token的接口
```

9：第三方应用前端获取后步骤8返回的用户凭证，生成前端缓存，返回登录成功。

可选步骤说明

T1：第三方应用验证用户密码是否发生过变更，如果发生过变更。
T1a：如果T1结果为用户更改过密码，那么调用SDK方法：IdpUtils.unbind(String idpUserId)，解除用户绑定。
T1b：T1a操作成功后，重新进入登录流程，即进入步骤1。

T2：第三方应用验证用户账号是否为可用状态
T2a：如果T2结果为账号不可用，那么调用SDK方法：IdpUtils.unbind(String idpUserId)，解除用户绑定，并提示用户相应错误，流程终止。

注：第三方应用也可以参照步骤**T1**、**T2**，增加其他自定义逻辑处理，根据业务情况，判断是否需要调用**SDK**方法，解除用户账号绑定关系，和自定义相关用户提示信息。

示例代码：

```
String privateKey = "MIIEvQIBADANBgkqhki...";
IdpProvider.init("fjjyxinxihua2017", privateKey, "http://localhost:8081", "http:xxxx");
try {
    IdpUtils.unbind("vcnlwokrqxonkwa");
} catch (Exception e) {
    System.out.print(e);
}
```

异常分支步骤说明**(**第三方应用用户账号与**IDaaS**绑定**)**：

a1：第三方应用前端接收绑定请求，接收参数SAMLRequest。

```
此地址为第三方应用申请接入IDaaS所提供的登录地址。
```

a2：第三方应用前端HTTP POST(Ajax)向第三方应用服务端请求验证SAMLRequest合法性。

a2.1：第三方应用服务端调用IDaaS SDK，验证SAMLRequest的合法性：

```
@RequestMapping(value = "/bind_uri", method = RequestMethod.GET)
public Object spStep2(HttpServletRequest request) {
    // 校验请求参数是否合法
    try {
        SAMLRequest samlRequestObj = IdpUtils.verifyUrlSignedRequest(request, DebugConf
ig.APP_CLIENT_ID);
    } catch (Exception e {
        // ==========================================
        // TODO: 校验异常，接入业务根据自身需要进行相关处理
        // ==========================================
    }

    // ================================
    // TODO：校验通过，正常显示登录界面
    // ================================
    return null;
}
```

a3：第三方应用服务端返回验证结果给第三应用前端。

a4：第三方应用前端判断步骤a3的结果，如果请求合法，显示第三方应用用户账号的输入框，如：

a5：第三方应用前端HTTP POST(Ajax)提交用户输入的账号信息（用户名、密码）及SAMLRequest给第三方应用服务端，获取绑定结果信息。

a5.1：第三方应用服务端验证用户信息后，调用IDaaS SDK，将用户ID封装到SAMLResponse响应用报文中：

```
/**
 * 101UC登录代理
 */
@RequestMapping(value = "/bind_uri", method = RequestMethod.POST)
public Object login(HttpServletRequest request, HttpServletResponse response, @RequestBody JSONObject loginBody)
        throws Exception {
    // 校验请求参数是否合法:接入应用根据情况选择是否进行二次校验
    // SAMLRequest samlRequest = IdpUtils.verifyUrlSignedRequest(request, DebugConfig.APP_CLIENT_ID);
    // ====================================================
    // TODO:idp校验账密，接入应用根据自身系统修改下面校验代码
    String idpUserId = DebugConfig.getIDaaSUserIdByLoginName(loginBody.getString("login_name"));
    // ====================================================

    // 构建协议
    IdaasResponseSamlProtocols protocols = IdpUtils.buildIdaasResponseProtocol(samlRequest, idpUserId);
    // 生成saml协议密文
    String urlEncodeSaml = URLEncoder.encode(RedirectUtils.getUrlUnsignedResponseEncodedMessage(protocols),
```

```
            "UTF-8");
    JSONObject retObj = new JSONObject();
    retObj.put("idp_response_target", DebugConfig.IDAAS_WEB_HOST + IdaasWebServiceUri.I
DP_LOGIN_RESPONSE_PATH);
    retObj.put("saml_response", urlEncodeSaml);
    retObj.put("product", DebugConfig.PRODUCT_AUTO_ID);
    return retObj;
}
```

a6：第三方应用服务端返回绑定结果给前端，绑定成功情况下，应包含数据：

```
{
    "idp_response_target":"http://idaas-portal.beta.101.com/idp/response", //绑定成功通
知IDaaS地址
    "saml_response":"...", //SAMLResponse 报文
    "product":"..." //产品代码
}
```

a7：第三方应用前端使用步骤a6的结果，组合Form POST表单：

```
<form id="idpResponse" action="http://idaas-portal.beta.101.com/idp/response" method="P
OST" style="display:none">
<input type="hidden" name="SAMLResponse" value="..." />
<input type="hidden" name="product" value="..." />
</form>

// method 必须是 POST
// target 必须是 _parent
```

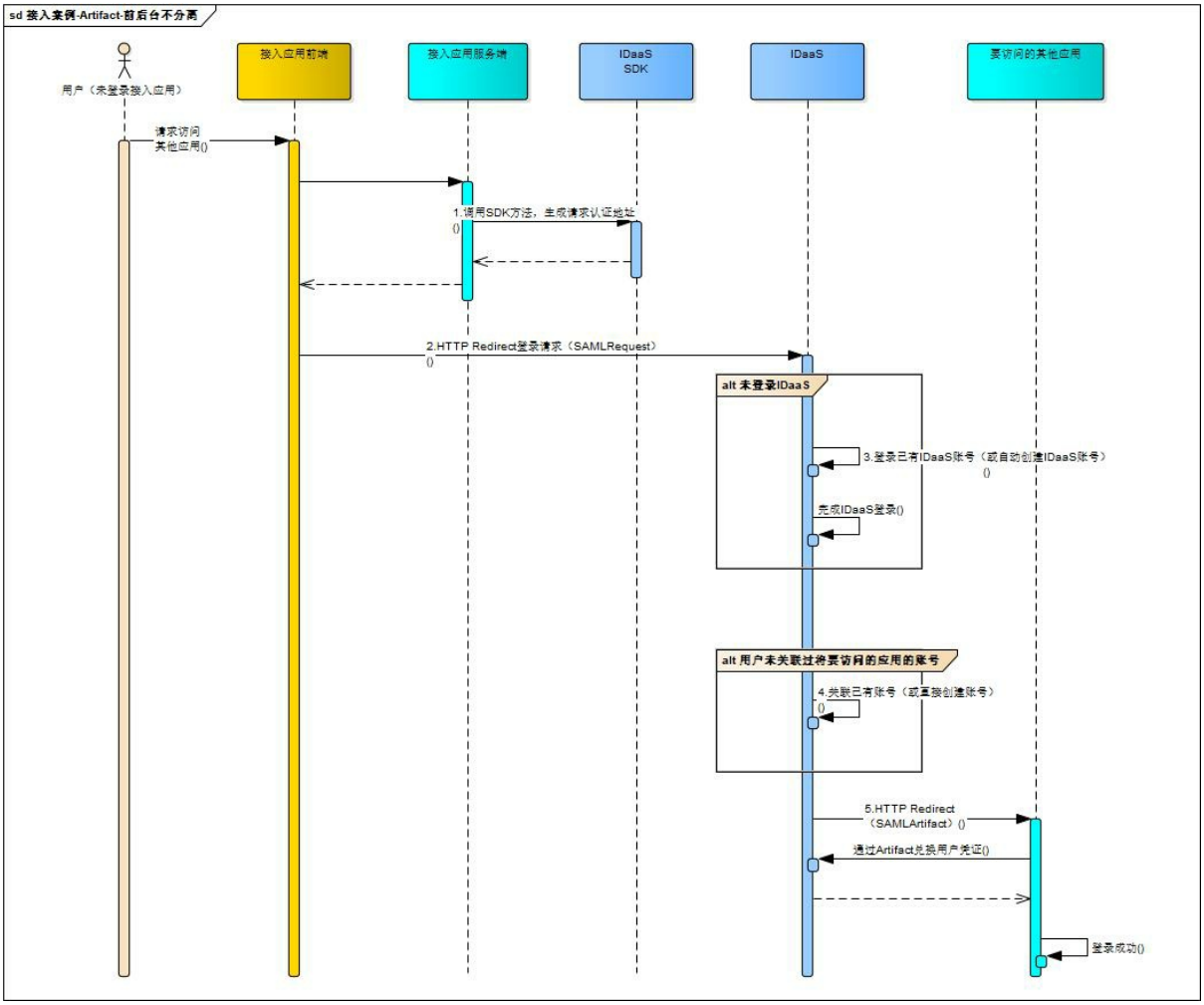
a8：第三方应用前端触发提交步骤a7生成的Form表单：

```
<script type="text/javascript">
    document.getElementById("idpResponse").submit();
</script>
```

用户反向登录

反向登录流程时序图

用户未登录：



步骤说明：

1：用户未登录时，实际发起的是用户正向登录，并在链接后加上参数“to_client_id”，表示将要访问的应用的客户端ID。

```
    // 构建登录请求协议，参数为请求IDaaS完成绑定后回调SP服务端的地址
    GenerateRequestSamlProtocols protocols = SpUtils
            .buildLoginRequestProtocol(DebugConfig.SP_CLIENT_HOST + "/sp/sp_respons
e_uri");
    // 生成请求的完整URL
    String idpRequestUriFull = RedirectUtils.getUrlSignedRequestEncodedUrl(protocol
s);
    // 拼接产品编码
    String retUri = new StringBuilder(idpRequestUriFull).append("&product=").append
(DebugConfig.PRODUCT_AUTO_ID)
                .append("&to_client_id=").append("此处请填写将要访问的应用客户端ID，如
client10000001").toString();

    // ================================================
```

```
        // TODO : 接入应用完成后续处理
        // ================================================
```

2：使用步骤1中生成的链接进行页面重定向

3：由IDaaS进行当前登录状态的判断

3.1： 若已登录IDaaS，则跳过步骤3；

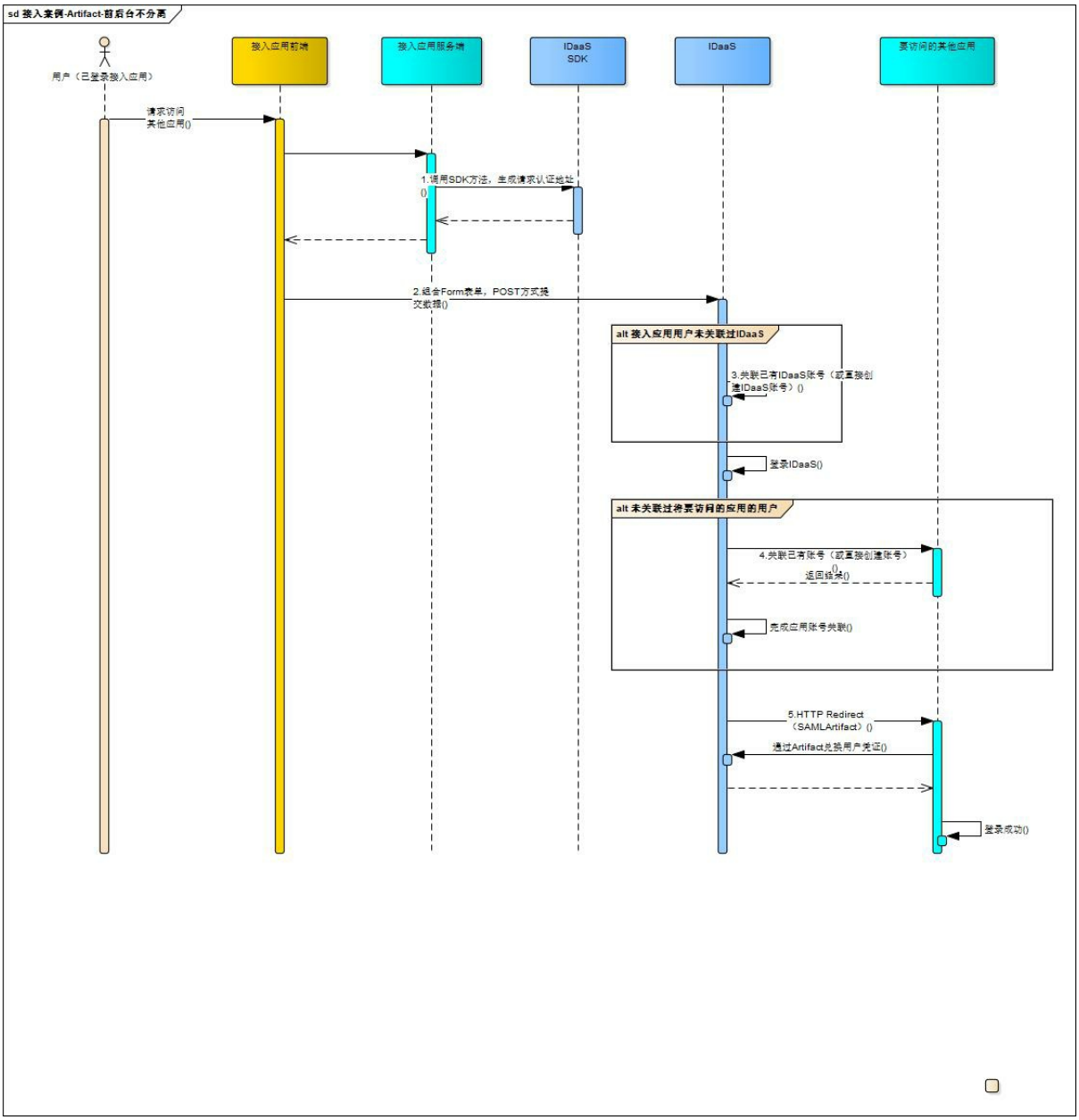
3.2： 若未登录IDaaS，用户需通过提供的登录方式中的一种进行登录。

4：由IDaaS进行当前IDaaS用户与将要访问的应用之间是否关联的判断

4.1 若已关联过，则页面携带SAMLArtifact参数重定向到要访问的应用，由应用完成后续操作。

4.2 若未关联过，用户可以选择和已有的应用账号绑定。如应用配置了“自动创建用户”的功能（需在IDaaS平台配置），用户也可选择“自动创建应用账号”。完成应用账号绑定后，后续操作同4.1。

用户已登录：



步骤说明

1：用户已登录第三方自己的登录体系时，实际发起的是用户反向登录，并在链接后加上参数“to_client_id”，表示将要访问的应用的客户端ID。

```
@RequestMapping(value = "/logined/create_request_uri", method = RequestMethod.GET)
public JSONObject loginHtml()throws UnsupportedEncodingException {
    // TODO 获取当前的登录用户信息
    String idpUserId = "TODO";
    String idpUserNickName = "TODO";
    String toClientId = "TODO";
```

```
        // 构建协议
        ReverseRequestAttribute attribute = new ReverseRequestAttribute();
        attribute.setNickName(idpUserNickName );
        // 回调地址
        attribute.setSpResponseUri(DebugConfig.SP_CLIENT_HOST + "/sp/sp_response_uri");
        // 将要访问的应用的客户端ID（由IDaaS提供）
        attribute.setToClientId(toClientId);
        ReverseRequestSamlProtocols protocols = SpUtils.buildReverseRequestProtocol(idp
UserId, attribute);
        // 生成saml协议密文
        String urlEncodeSaml = URLEncoder.encode(RedirectUtils.getUrlUnsignedResponseEn
codedMessage(protocols),
                "UTF-8");
        JSONObject retObj = new JSONObject();
        retObj.put("idp_response_target", DebugConfig.IDAAS_WEB_HOST + IdaasWebServiceU
ri.IDP_REVERSE_REQUEST_PATH);
        retObj.put("saml_response", urlEncodeSaml);
        retObj.put("product", DebugConfig.PRODUCT_AUTO_ID);
        return retObj;
    }
```

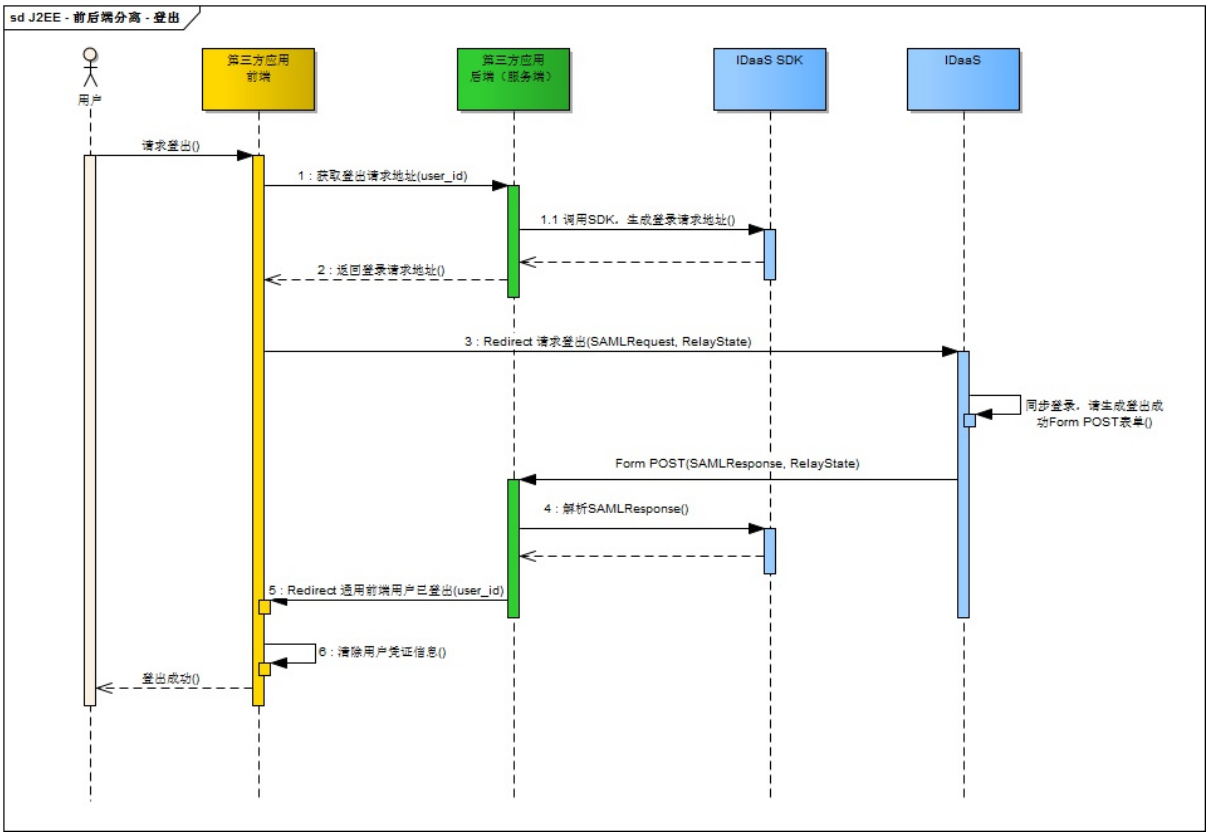
2：由前端将参数组合成form表单，通过post方式提交到IDaaS指定地址。

```
    var form = $("<form method='post'></form>");
    form.attr({"action": data.idp_response_target});
    var my_input = $('<input type="text" name="SAMLResponse" />');
    my_input.val(data.saml_response);
    var my_input2 = $('<input type="text" name="product" />');
    my_input2.val(data.product);
    form.append(my_input);
    form.append(my_input2);
    form.appendTo(document.body);
    form.submit();
```

3：后续步骤同“用户未登录”流程。

用户登出

登出时序图



步骤说明：

1：第三方应用前端HTTP POST(Ajax)向第三方应用服务端获取登出请求地址，参数为第三方应用用户ID。

1.1：第三方应用服务端调用IDaaS SDK，生成登出请求地址：

```
/**
 * 生成用户登出请求
 */
@RequestMapping(value = "/sp/logout/request", method = RequestMethod.GET)
public LogoutRequestResponseVo logoutAgent(@RequestParam(value = "user_id") String user
Id) {
    GenerateLogoutRequestSamlProtocols protocols = SpUtils.buildLogoutRequestProtocol(s
pResponseUri, userId);
    String idpLogoutRequestUriFull = RedirectUtils.getUrlSignedRequestEncodedUrl(protoc
ols);
    return new LogoutRequestResponseVo(idpLogoutRequestUriFull);
}

//spResponseUri ： IDaaS同步登出成功后通知第三方应用的回调地址
```

其中，“http://spResponeUri”位置处理为IDaaS完成登出请求响应后的回调地址，其优先级高于IDaaS应用登记时配置的默认回调地址。

2：第三方服务端返回登出请求地址给前端：

```
{
    "idp_logout_request_uri_full":"http://idaas-portal.beta.101.com/bg/index.html#/logo
ut?SAMLRequest=..."
}
```

3：第三方应用前端HTTP Redirect跳转到步骤2获得的地址。

4：第三应用服务端接收IDaaS登出成功通知（地址为步骤1中的回调址）后，调用IDaaS SDK，解析SAMLResponse参数：

```
/**
 * 获取登出的用户标识
 */
@RequestMapping(value = "/idp/logout/response", method = RequestMethod.POST)
public void resolveLogoutResponse(HttpServletRequest request, HttpServletResponse httpS
ervletResponse) {
    SAMLResponse samlResponse = SpUtils.resolveLogoutResponse(response);
    try {
        //第三方应用用户ID : samlResponse.getSubjectForNameID()
        httpServletResponse.sendRedirect(logoutSuccessWebUri + "?user_id=" +
        samlResponse.getSubjectForNameID());
    } catch (IOException ex) {
        throw new WebportalException("登录跳转异常");
    }
}

//logoutSuccessWebUri : 第三方应用前端接收登出成功通知地址
```

5：第三方应用服务端HTTP Redirect到第三方应用前端接收登出成功通知地址，参数为第三方应用用户ID。

6：第三方应用前端接收登出地址接收到通知后，对比通知参数中的用户ID与缓存中的用户ID，如果一致，则清除用户凭证缓存信息，完成登出操作。