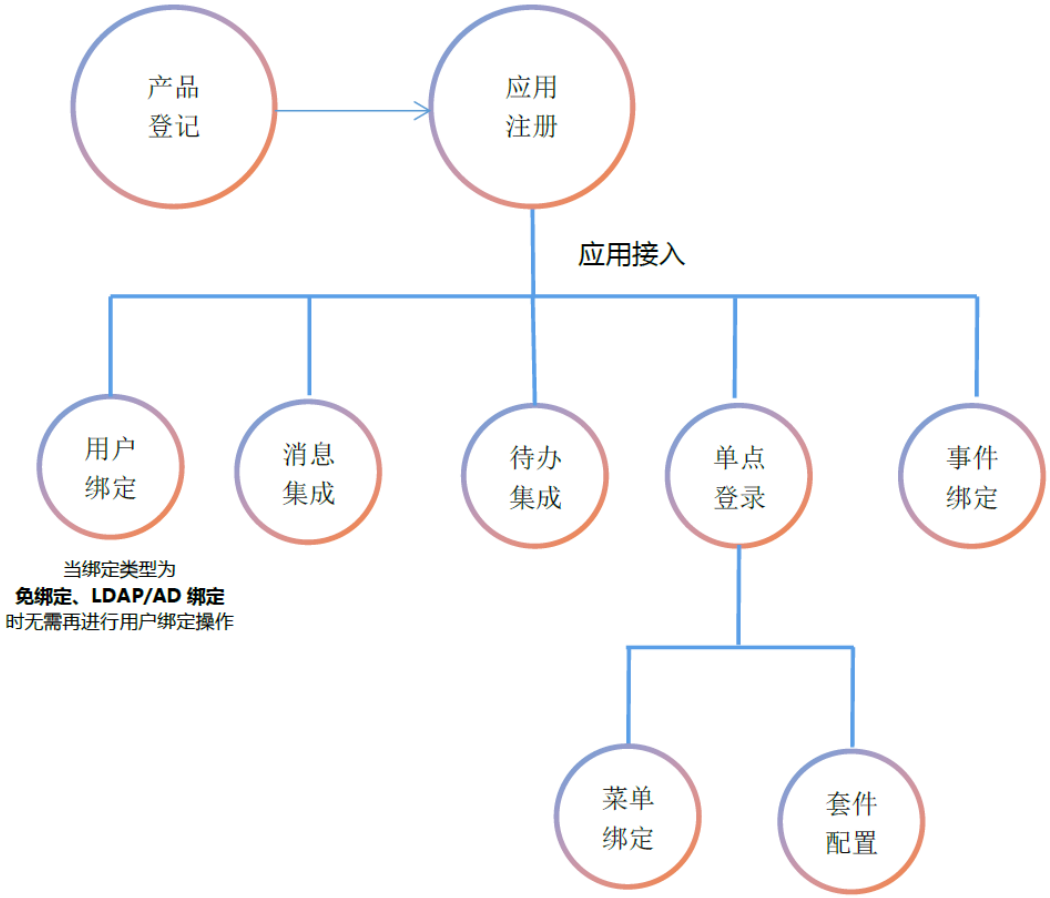
CIP[1]集成平台

> 注：本文档基于A8V7.0以上版本生效，低版本略微有差别，请自行调整，尤其是低版本不含人员免绑定功能，本文档中的代码均基于A8V71SP1版本。

一、概述

A8-V5协同集成依托协同平台，提供各种级别的企业集成要求。A8-V5协同集成包括4大平台接口支持、集成开发工具和标准集成插件。A8-V5不仅基于CIP支持主数据同步、业务数据集成、流程集成、应用接入、功能集成、门户集成等六大类典型集成应用场景，还提供丰富广泛的集成扩展支持，能够满足用户的各类集成需求。

CIP，协同集成平台，提供内外开放接口库、组织机构同步、消息/待办集成、单点登录、企业统一信息门户、数据交换等集成服务支持，还提供大量第三方标准集成插件，包括NC集成插件、EAS集成插件、SAP集成插件、U8集成插件、视频会议集成插件、携程商旅集成插件等。



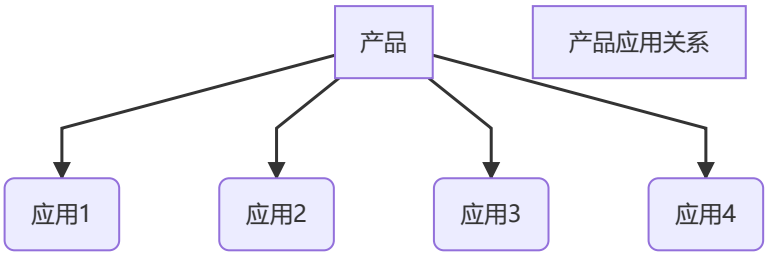
CIP采用分层服务模式提供开放、快捷的集成服务支持，同时对平台的集成任务和数据交换实施全面跟踪和监控。CIP以通用集成服务和整体业务组件为核心。在底层提供广泛的内外接口/事件支持，减少异构系统的技术和业务复杂度影响，降低集成门槛和实施成本；在应用层封装包含完整功能的通用应用服务，多外支持多个同类系统接入，对内提供统一接口，如通用组织机构同步、视频会议组件等.

本文档主要介绍CIP平台在第三方接口库、单点登录、消息/待办集成方面的功能，关于组织架构同步[2]和业务流程集成[3]可以参考其他相关的视频及文档。

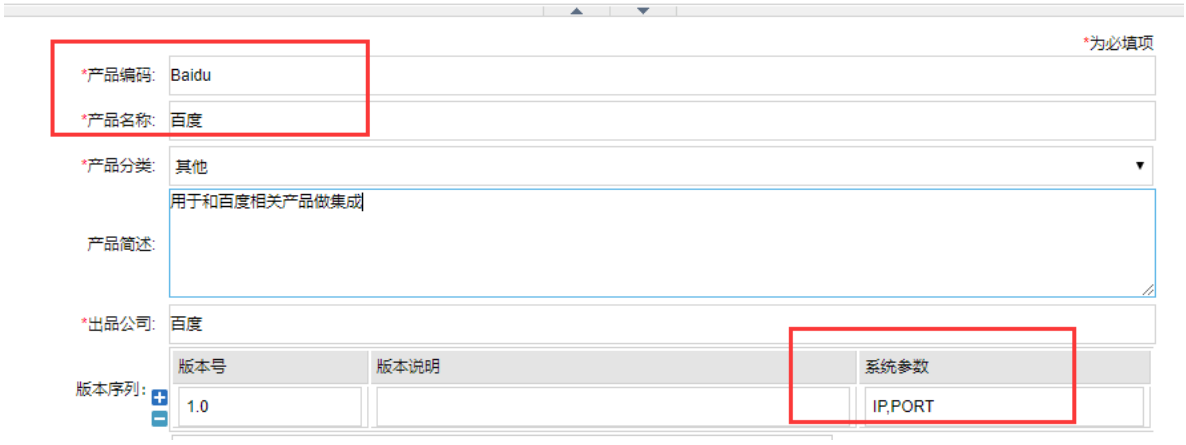
二、第三方应用

2.1 产品登记

当第三方系统要集成到A8中时，我们需要先进产品登记，这里的产品登记就标记了我们的后续做应用注册的分类，即产品->应用为1-N。



例如，我們要和百度做集成，那麼我們新增一個产品为Baidu：



注意，这里有系统参数，这里的系统参数可以用于后续的变量，我们可以设置为百度的相关地址和端口，方便后期的维护和环境的切换：

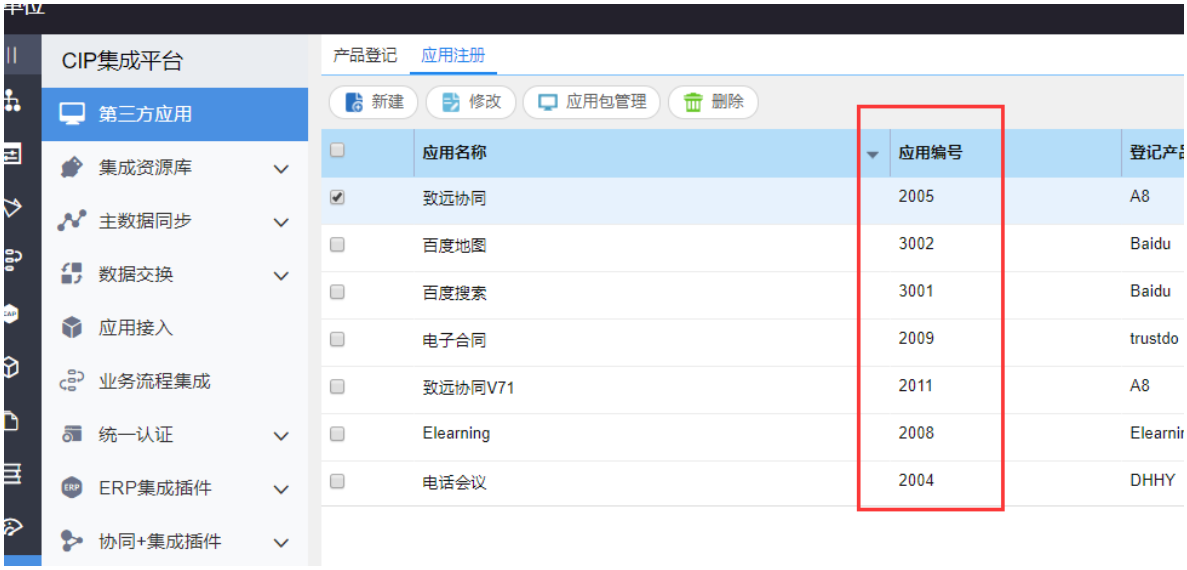
2.2 应用注册

产品新增完成后，可以给对应产品新增应用，这里继续对第一步的百度应用进行应用注册。这里新增两个应用：百度搜索和百度地图：

给应用上传好相应的图标，选择对应的产品，这里需要注意的一点是接入方式，这里的接入方式选择不同，第三方推送数据的时候参数也不同，详见待办和消息集成中的参数说明：

类型	说明
PC	仅在PC端 可以看见
移动URL接入应用	仅在移动端可以看见
本地H5应用	按照CMP开发规范开发的应用包
PC&移动URL接入应用	1/2结合
PC&本地H5应用	1/3结合
PC&原生应用	PC和原生APP应用

我们对产品进行应用注册后，都会自动生成一个应用编号，此编号即第三方系统调用借口使用的registerCode参数，需要由OA提供给第三方：



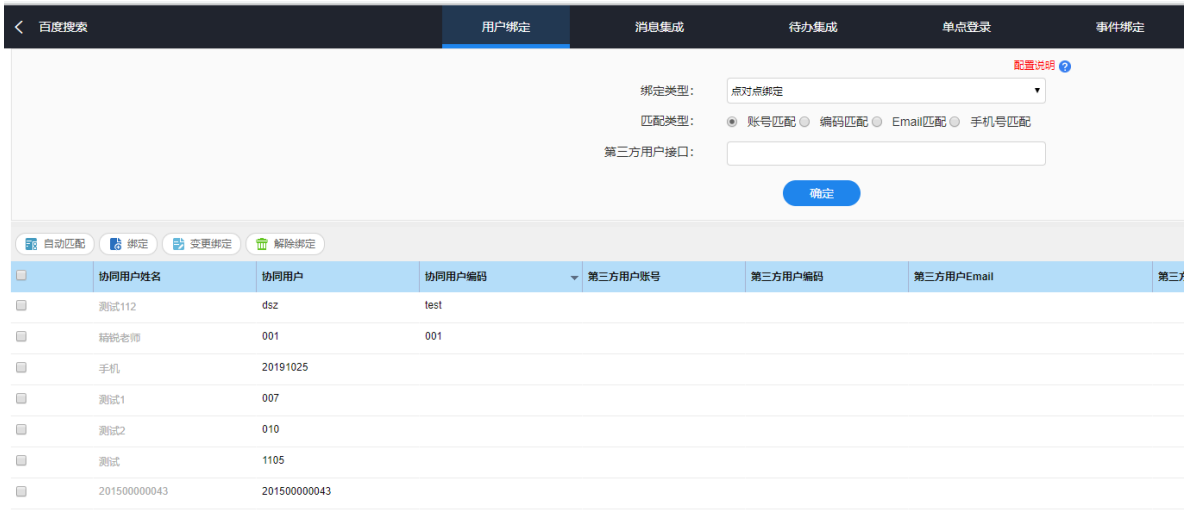
三、用户绑定

无论任何两个系统做集成，最关键的点都是用户的映射关系，如果不能满足用户的映射关系，那么系统集成性也无从谈起。

在A8V70以上版本中，产品标准支持了免绑定功能，如果和第三方的登录名、手机号、邮箱或者编码一直的话，无需再进行点对点的绑定操作。

3.1 点对点绑定

点对点绑定，即每个用户和第三方的用户进行中间表的映射（thirdparty_user_mapper），要实现此映射，我们需要第三方提供一个用户信息绑定接口，按照需要的格式返回数据



要绑定上面的数据有两种方式，一种第三方进行推送，或者获取第三方的数据进行主动绑定：

3.1.1 第三方推送

我们系统提供了两个post接口供第三方进行调用推送数据：

批量绑定用户地址:/seeyon/rest/thirdpartyUserMapper/binding

单个用户绑定地址:/seeyon/rest/thirdpartyUserMapper/binding/singleUser

注意：调用A8的接口都需要先进行token认证。对rest接口不了解的可以先学习产品rest相关内容

具体参数如下：

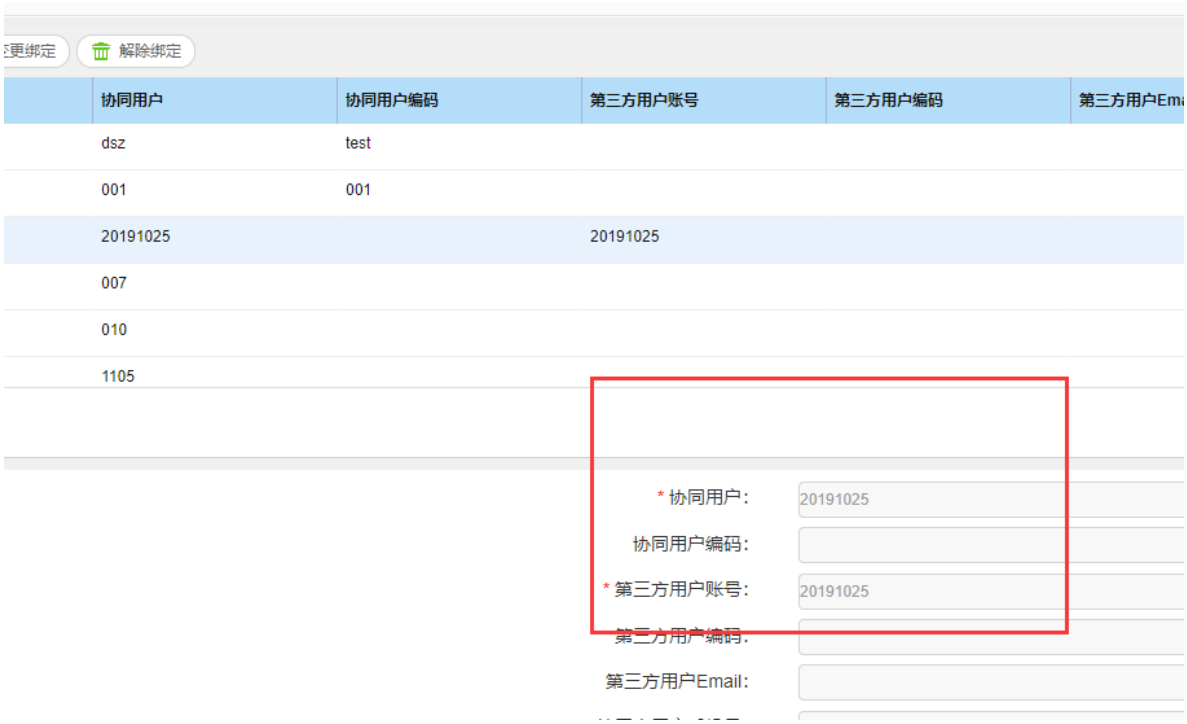
```
// 批量绑定数据
{
    "userlist": [
        {
            "registerCode": "注册系统编码（必须参数）",
            "thirdUserId": "第三方系统用户主键（必须参数）",
            "thirdLoginName": "第三方登录名",
            "thirdName": "",
            "thirdCode": "",
            "thirdMobile": "",
            "thirdEmail": "",
            "param0": "扩展参数或者属性",
            "param1": "扩展参数或者属性",
        }
    ]
}
// 单条绑定数据：
{
    "registerCode": "注册系统编码（必须参数）",
    "thirdUserId": "第三方系统用户主键（必须参数）",
    "thirdLoginName": "第三方登录名",
    "thirdName": "",
    "thirdCode": "",
    "thirdMobile": "",
    "thirdEmail": "",
    "param0": "扩展参数或者属性",
    "param1": "扩展参数或者属性",
}
```

下面我们设置点对点绑定为账号绑定模式，通过接口绑定一个用户：

```
/**
     * 以绑定单个用户为例
     * @throws Exception
     */
    @Test
    public void bindUser() {
        String url = URL + "/thirdpartyUserMapper/binding/singleUser?token=" +
token;
        BindUser user = new BindUser();
        user.setRegisterCode("3001");
        user.setThirdUserId("1");
        // 根据点对点绑定中设置的类型进行设置
        user.setThirdLoginName("20191025");
        // 其余字段可以不用设置
        String res;
        try {
            res = HttpKit.post(url, JSONUtil.toJSONString(user));
            System.out.println(res);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

上面代码中注意，我们根据设置的类型需要给第三方人员的相关属性进行赋值，设置账号绑定，则需要传递thirdLoginName字段，邮箱绑定则需要thirdEmail，绑定成功结果：

```
{
  "errorMsgs" : [ ],
  "success" : true
}
```



绑定失败结果如下：

```
{
  "errorMsgs" : [ {
    "errorDetail" : "第三方人员主键：20191025",
    "errorCode" : 102,
    "errorType" : "第三方系统人员账号为空"
  } ],
  "success" : false
}
```

3.1.2 OA主动拉取

实现这种方式，我们需要先到CIP接口库中配置用户信息接口，需要第三方按照上面的数据格式提供相应的返回数据，后面放到CIP接口库中进行讲解，这里不过多说明。

如果第三方不愿意做开发，那么可以在自己的接口中取调用他们的接口，返回A8需要的数据，然后配置到CIP中，甚至可以直接单独开发定时任务，调用相关的内部方法进行绑定，可以参考产品代码的实现方法：

```
// ThirdpartyUserMapperManagerImpl.java
@Override
    public CIPErrorMessage transAutoMapper(String
registerCode,List<ThirdpartyUserMapperImporter> importer) throws
BusinessException {
        if (CollectionUtils.isEmpty(importer)) {
            LOG.info("需要匹配第三方用户信息集合为空，不执行");
            return null;
        }
        //人员绑定首先需要依据注册编码取人员绑定配置信息，如果是免绑定或ldap/ad绑定则不处理绑
定信息，否则走点对点映射表方式
        CIPGlobalEnum.AutoBindMapperType autoBindingEnum =
CIPGlobalEnum.AutoBindMapperType.account;

        CIPUserBindingConfig configItem
=cipUserConfigManager.getCacheThirdpartyConfig(registerCode);
        if (configItem != null) {
            //免绑定不再走匹配模式，直接返回错误信息

if(CIPUserBindingTypeEnum.Point2Point!=CIPUserBindingTypeEnum.getCIPBindingType(
configItem.getBindingType())){
                LOG.warn("CIP非点对点模式不再执行匹配操作");
                CIPErrorMessage error = new CIPErrorMessage();
                error.addErrorMsg(MessageStatus.userBindingTypeError,"不再执行绑定
操作");

                return error;
            }
            autoBindingEnum =
CIPGlobalEnum.AutoBindMapperType.getEnum(configItem.getMapperType());
        }

        Map<String, UserMapperAction> actionMap =
AppContext.getBeansOfType(UserMapperAction.class);
        UserMapperAction action = null;
        for (Map.Entry<String, UserMapperAction> entry : actionMap.entrySet()) {
            UserMapperAction a = entry.getValue();
```

```
            if (a.getAutoMapperType() == autoBindingEnum) {
                // 自动匹配模式
                action = a;
                break;
            }
        }
        if(null!=action){
            Long userId =-1L;
            User user = AppContext.getCurrentUser();
            if(user!=null){
                userId =user.getId();
            }
            CIPErrorMessage erro = action.transMapper(registerCode, importer);
            RegisterPO po = registerManager.getRegisterIdByCode(registerCode);

            if (!erro.isSuccess()) {
                List<ErrorMessage> messages = erro.getErrorMsgs();
                StringBuilder sb = new StringBuilder();
                for (ErrorMessage errorMessage : messages) {
                    sb.append(errorMessage.getErrorType() + " " +
errorMessage.getErrorDetail());
                }
                LOG.warn("系统注册匹配失败信息：" + sb.toString());
                Map<String,Object> param=new LinkedHashMap<String,Object>();
                param.put("link", true);
                CIPActionLogFactory.createLog(LogApplicationEnum.APPINTEGRATION,
(po!=null?po.getAppName()+":":"")+"执行绑定第三方用户",
                        "绑定第三方用户失败信息：" + sb.toString(), userId, null,
param);
                return erro;
            }
            CIPActionLogFactory.createLog(LogApplicationEnum.APPINTEGRATION,
(po!=null?po.getAppName()+":":"")+"执行绑定第三方用户", "成功绑
定"+importer.size()+"人", userId, null, null);

        }
        return new CIPErrorMessage();
    }
```

3.1.3 注意事项

通过上面的接口我们看到，第三方推送过来的情况实际上也是绑定了用户的编号、登录名、邮箱或者手机号之一，实际如果是这种情况，我们支持采用免绑定即可，一般情况下是由于没有映射关系才会采用这种方式，例如第三方的信息是加密的，需要做一些解密操作等。

3.2 免绑定（推荐）

对于用户的免绑定，只要双方系统有对应的关系即可，登录名、邮箱、手机号及编号任意一个对应即可，我们也强烈推荐这种方式，无论是对开发或者对客户来说都能够更好的管理用户信息，否则系统一旦多了就会出现多个用户名，难以维护。

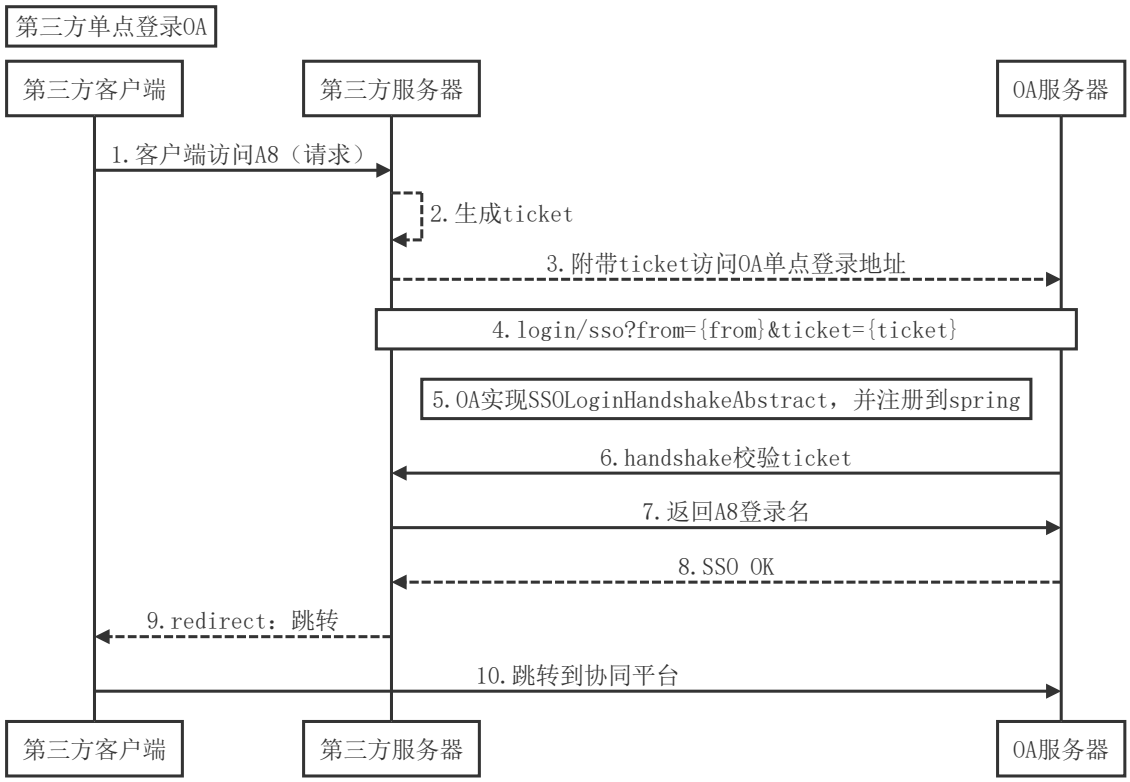
四、单点登录

4.1 单点登录类型

从整体来说，单点登录无非两种，第三方单点登录OA以及OA单点登录第三方，但是各个厂商的实现方式都有一定的区别，这里列举一些常见的场景。

4.2 第三方单点登录OA解决方案

为便于理解，第三方单点登录到OA的整体流程图如下：



上述流程图可以解析为三个步骤：

1. 用户登录第三方后访问OA平台。（第三方开发，生成ticket访问A8单点地址）
2. OA实现一个握手类，根据第三方传过来的ticket获取用户信息（A8开发）。
3. ticket解密成功后返回登录名，通过login/sso?from={from}&ticket={ticket}跳转到OA首页。

务必注意：ticket的解析必须在服务端，否则暴露在前端解析会有安全问题。

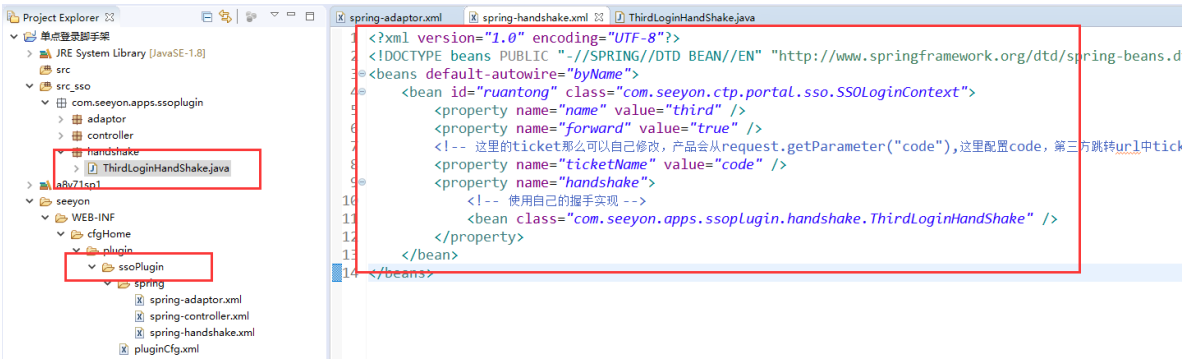
4.2.1 第三方开发

这里以自己开发的一个简单系统为例，新增下面的一个html页面（index.html）：

```
<!DOCTYPE html>
<html>
        <head>
             <meta charset="utf-8">
             <title>第三方首页</title>
        </head>
        <body>
        <h1>欢迎登陆：测试账号（A8登陆名为dsz）</h1>
        下面的ticket为随机数,这里随意写的，实际情况由第三方系统生成<br><br><br><br>
         <a href="http://127.0.0.1/seeyon/login/sso?
from=third&code=45613241654986498">单点登录到OA</a>
        </body>
</html>
```

4.2.2 A8系统开发

从上面的页面可以看到提供给第三方的跳转的地址为from=third，code=xxx，那么我们需要到A8中实现对应的握手类并注册到spring中，按照A8插件开发机制[5]进行开发：




```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-autowire="byName">
    <bean id="ruantong" class="com.seeyon.ctp.portal.sso.SSOLoginContext">
        <property name="name" value="third" />
        <property name="forward" value="true" />
        <!-- 这里的ticket那么可以自己修改，产品会从request.getParameter("code"),这里配置code，第三方跳转url中ticket=xxx就需要改成code=xxx -->
        <property name="ticketName" value="code" />
        <property name="handshake">
            <!-- 使用自己的握手实现 -->
            <bean class="com.seeyon.apps.ssoplugin.handshake.ThirdLoginHandShake" />
        </property>
    </bean>
</beans>
```

开发完成后，我们将补丁包部署到A8中，重启服务。

注意，我上面的配置文件中ticketName配置了code这里一定需要和提供给第三方的进行对应，否则握手类中无法获取到ticket参数，这种场景一般用于第三方的ticket参数已经被使用的情况。其中handshake需要替换为自己的握手实现类，这里的示例为ThirdLoginHandShake：

```
package com.seeyon.apps.ssoplugin.handshake;

import com.seeyon.ctp.portal.sso.SSOLoginHandshakeAbstract;

public class ThirdLoginHandShake extends SSOLoginHandshakeAbstract {

    /**
     * 单点登录握手方法，返回A8登录名
     */
    @Override
    public String handshake(String ticket) {
        // 根据ticket去获取第三方的登录人员，调用接口或者解密等
        System.out.println("第三方传递的ticket为：" + ticket);
        // 建议都采用调用接口的方式，更加安全， 解密 的话一定要保证每个用户名不同时期的加密结果不一致
        return "dsz";
    }

    /**
```

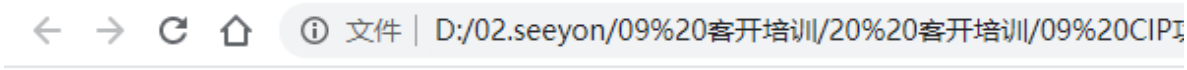
```
     * 单点登录退出通知第三方
     */
    @Override
    public void logoutNotify(String ticket) {

    }
}
```

由于这里仅仅是为了演示效果，所以握手类中直接返回了我本地环境中已有的一个用户登录名，实际情况中需要在这里面实现验证逻辑，例如钉钉的接口验证：

```
package com.seeyon.apps.dingding.handshake;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.seeyon.apps.dingding.kit.HttpKit;
import com.seeyon.ctp.portal.sso.SSOLoginHandshakeAbstract;

/**
 * @author kkdo
 * @date 2019年6月24日 下午1:40:20
 * <pre>单点登录握手类</pre>
 */

public class ScanLoginHandShakeImpl extends SSOLoginHandshakeAbstract {

    @Override
    public String handshake(String ticket) {
        try {
            String tokenurl = "https://oapi.dingtalk.com/sso/gettoken?corpid=" +
Constants.CorpId + "&corpsecret=" + Constants.corpsecret;
            String tokenRes = HttpKit.get(tokenurl);
            JSONObject obj = JSON.parseObject(tokenRes);
            String token = obj.getString("access_token");
            String userUrl = "https://oapi.dingtalk.com/sso/getuserinfo?
access_token=" + token + "&code=" + ticket;
            String useRes = HttpKit.get(userUrl);
            JSONObject user = JSON.parseObject(useRes);
            ticket = user.getJSONObject("user_info").getString("userid");
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 以钉钉的userid作为OA的登录名，实际情况可能是手机号或者其他情况
        return ticket;
    }

    @Override
    public void logoutNotify(String arg0) {

    }

}
```

4.2.3 单点登录测试

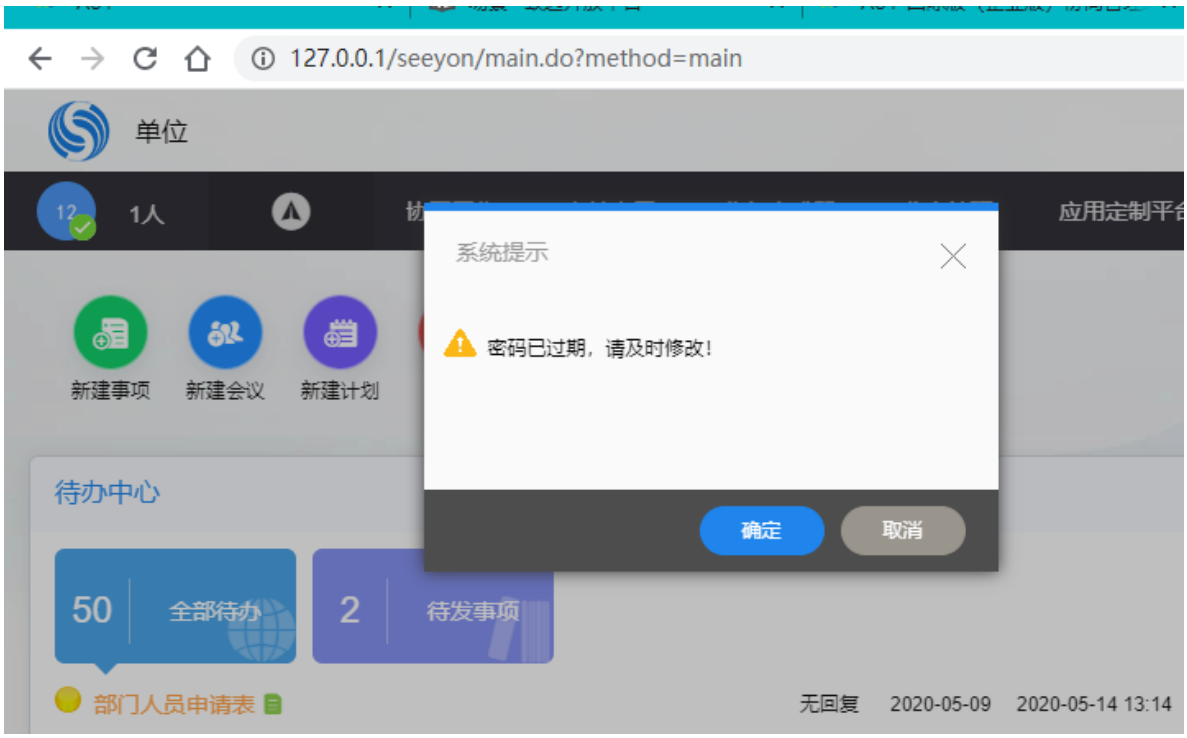
A8重启完成后，我们访问第三方系统的index.html页面，并点击单点登录到OA：



欢迎登陆：测试账号（A8登陆名为dsz）

下面的ticket为随机数,这里随意写的，实际情况由第三方系统生成

单点登录到OA



可以看到成功登录到OA系统中，说明单点登录成功！

4.2.4 常见错误

1. ticket不能直接简单加密或者直接采用无加密的登录名，系统会报错：[ticket] prohibit same user login name!
2. 注意第三方系统的地址和tickeName的对应关系，如果不正确会报错：Parameter 'code' is not available.看到此错误则说明第三方地址中不含code参数。

4.2.5 特殊情况

上述的方案是说明了有第三方配合的情况下的单点登录的开发方式，接下来介绍几种特殊的单点登录情况。例如：

1. 钉钉或者微信扫码登录到OA系统中
2. 硬件集成，获取U-key等硬件的用户信息单点登录
3. OAuth2.0登录认证

如果遇到上述情况，我们需要自行修改login.jsp，查看钉钉的官方文档，在页面中引入钉钉的二维码生成js，并且将二维码展示到A8中，钉钉扫码后，会将带有用户信息的code返回到回调方法中，我们需要自己在js回调方法中访问（location.href):**login/sso?from=third&code={code}**，通过后端调用接口解析获取用户信息，返回A8登录名进行单点登录。

4.3 OA单点登录第三方解决方案

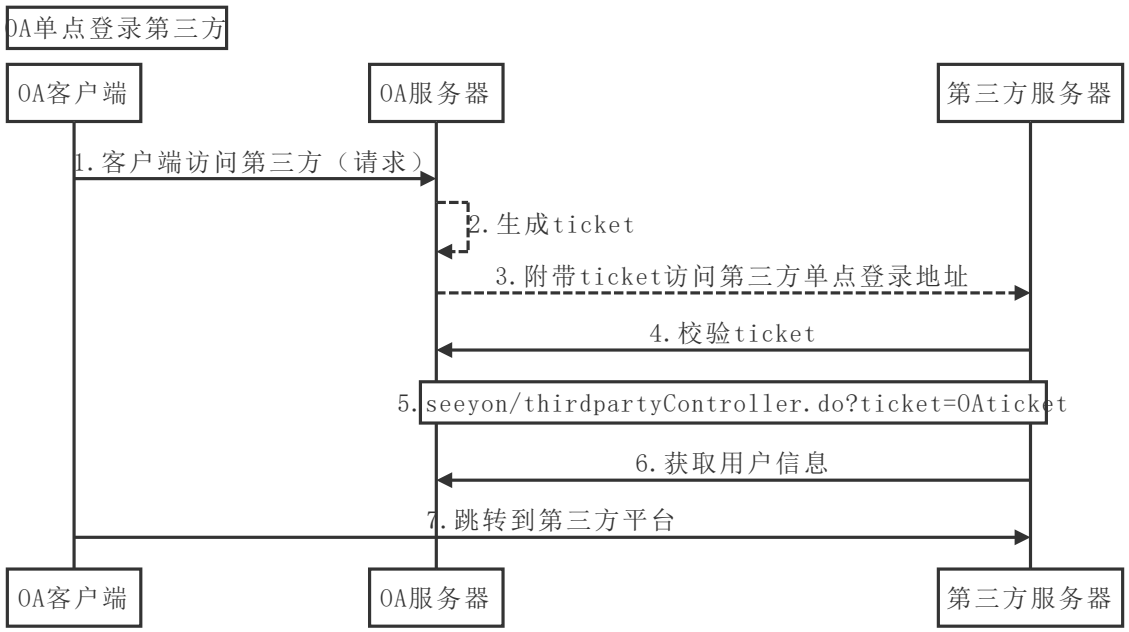
在A8系统中，对于OA单点登录到第三方系统提供了标准的解决方案，但是也会存在一些特殊情况，这里分别做一些说明。

首先看一下OA的单点登录分类：

其中门户认证机制为A8对外提供的标准模式，由第三方进行开发，同时还支持第三方认证机制、用户密码认证以及第三方token认证机制和CAS认证等。

4.3.1 A8标准方案（门户认证机制）

首先，我们介绍门户认证机制的过程：



根据上面的图可以看到，和之前第三方单点登录到OA过程反了一下，由OA提供了校验接口，以及生成的ticket，第三方开发对应的握手类进行登录逻辑。

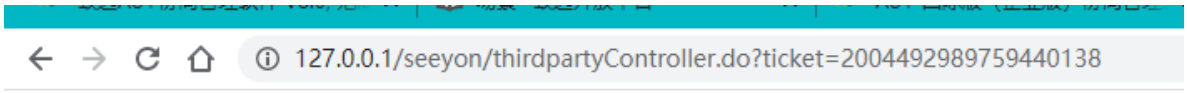
第三方需要开发类似下面的单点登录地址提供给A8：

```
/**
* 提供给OA登陆地址
* 接收一个ticket参数
*/
@GetMapping("/oalogin")
public String login(String ticket) {
    if (null != ticket) {
        // OA提供根据ticket获取用户名的接口地址
        String getLoginNameUrl = ConstConfig.OAURL +
"seeyon/thirdpartyController.do?ticket=" + ticket;
        String username;
        try {
            // 如果要获取其他信息可以在header中取
            username = HttpKit.get(getLoginNameUrl).trim(); // ..........
            // 建议先清除session 然后再执行登录逻辑
            // 执行第三方系统的登录逻辑
            // ..........
            return redirect("/");
        } catch (Exception e) {
        }
    }
    // 登录失败 直接跳转到登录页面
    return LOGIN_REALPATH;
}
```

A8拿到此地址后配置到单点登录中即可，例如我这里配置https://www.baidu.com,那么我访问的时候就会访问https://www.baidu.com?ticket=xxxxx，百度拿到ticket之后，调用seeyon/thirdpartyController.do?ticket=" + ticket获取OA的实际登录名，进行登录认证。

通过接口获取实际的登录名：

```
http://127.0.0.1/seeyon/thirdpartyController.do?ticket=2004492989759440138
```



dsz

PC端我们可以在栏目中看到配置的单点登录系统，如果是移动端，在V8.0以前需要用户手动开通，V8.0+可以直接开通，无需用户再次开通：

管理员可以给应用设置到不同的分类下面显示。

单点登录OA后，OA在退出时可以通知对方；

```
/**
 * 单点登录退出通知第三方  BUG  暂时无效
 */
@Override
public void logoutNotify(String ticket) {

}
```

4.3.2 第三方认证机制

单点登录的第三方认证机制一般是第三方有自己的一套验证机制，加密、ticket、token等方式，这种情况，我们一般不能直接进行跳转，需要做一个中转。

> 需求：某系统需要将用户名通过DE方式进行加密，放到url中，然后再访问就能进行单点认证。

4.3.2.1 通过产品的接口实现（推荐）

产品提供了第三方的认证机制的实现类，我们只需要实现产品接口并注册到spring中即可实现。

我们以上面注册的百度为例，例如要登录百度首页，需要在后面附带user=DES加密的user，那么实现过程如下：第一步，实现认证接口ThirdpartyAuthenticationPortal：

```
/**
     * 第三方系统认证page获取 ，返回自定义的参数，如aa=***&bb=***系统会自动依据PC登录地址和H5登录地址进行拼接跳转
     * @param memberId
     * @param loginAccountId
     * @return
     */
    String getpageUrl(long memberId, long loginAccountId);

    /**
     * 注册第三方系统编码
     * @return
     */
    String registerCode();
```

实现接口：public class BaiduAuthentication implements ThirdpartyAuthenticationPortal，主要实现代码如下：

```
public class BaiduAuthentication implements ThirdpartyAuthenticationPortal {

    private RegisterManager registerManager;

    public void setRegisterManager(RegisterManager registerManager) {
        this.registerManager = registerManager;
    }

    @SuppressWarnings("unchecked")
    @Override
    public String getpageUrl(long memberId, long loginAccountId) throws
BusinessException {
        // 获取注册信息
        RegisterPO registerIdByCode =
registerManager.getRegisterIdByCode(registerCode());
```

```
        // 获取应用注册的相关参数
        String paramValue = registerIdByCode.getParamValue();
        Map<String, String> paramValueMap = (Map<String, String>)
JSONUtil.parseJSONString(paramValue);
        System.out.println(paramValueMap);
        // String IP = paramValueMap.get("IP");
        // String PORT = paramValueMap.get("PORT");
        // 获取用户
        //V3xOrgMember member = Functions.getMember(memberId);
        // 获取第三方的用户名    如果是免绑定 则无需获取，直接用对应的属性即可
        String username = AppContext.getCurrentUser().getLoginName();
        Long registerId =
registerManager.getRegisterIdByCode(registerCode()).getId();
        String thirdLoginName =
ThirdpartyUserTransformationManager.getInstance()
                .getThirdLoginName(registerId.toString(), username);
        if (Strings.isNotBlank(thirdLoginName)) {
            username = thirdLoginName;
        }
        // 逻辑处理：进行加密
        try {
            String user = DESUtil.encrypt(username);
            return "user=" + user;
        } catch (Exception e) {
        }
        // 会将参数拼接到url后面
        return "user=null";
    }

    @Override
    public String registerCode() {
        return "3001";
    }
}
```

然后我们将代码注册到spring中（按照插件开发 注册）：

```
<bean id="baiduAuthentication"
class="com.seeyon.apps.ssoplugin.third.BaiduAuthentication"></bean>
```

我们上面的接口的返回值为user=xxx，那么我们配置https://ww.baidu.com,访问的时候会自动根据注册编码调用上面的实现类，获取user=xxx，附带到链接后面。即：https://www.baidu.com?user=xxxx。

系统获取实现类的方式：

```
ThirdpartyAuthenticationPortal portal =
CIPUtil.getAuthThirdpartyPortal(String.valueOf(register.getAppCode()));
```

实际访问地址：

```
https://www.baidu.com/?
user=k%2OzOLNrEpQ8=&from=A8&ticket=-8000310828009959609&interneturl=http://127.0
.0.1
```

4.3.2.2 自定义插件实现

我们除了可以采用第一步，我们在OA中按照插件开发[2]模式，开发一个controller，用于后台加密及跳转，主体实现逻辑如下：

```
public class SsoPluginController extends  BaseController {

    private static final Log log = LogFactory.getLog(SsoPluginController.class);

    private static ConcurrentHashMap<String, SsoPluginAdaptor> ADAPTOR_MAP = new
ConcurrentHashMap<>();

    public ModelAndView index(HttpServletRequest request, HttpServletResponse
response) {
        String ssoname = request.getParameter("ssoname");
        //String ticket = request.getParameter("ticket");
        if(null == ssoname || "".equals(ssoname)) {
            try {
                rendJavaScript(response, "alert('链接有误，务必包含ssoname参数');");
            } catch (IOException e) {
                log.error("输出js发生异常：", e);
            }
            return null;
        }

        SsoPluginAdaptor adaptor = getAdaptor(ssoname);

        if(null == adaptor) {
```

```
            try {
                rendJavaScript(response, "alert('适配器有误，没有找到对应的适配器，请
联系开发人员');");
            } catch (IOException e) {
                // TODO 输出js发生异常
                log.error("输出js发生异常：", e);
            }
            return null;
        }

        String url = adaptor.getRedirectUrl(request);
        // 进行url校验
        log.error("获取到的url为：" + url);
        try {
            response.sendRedirect(url);
        } catch (IOException e) {
            log.error("跳转发生异常：", e);
        }
        return null;
    }

    /**
     * 获取适配器
     * @param name
     * @return
     */
    private SsoPluginAdaptor getAdaptor(String name) {
        SsoPluginAdaptor adaptor = ADAPTOR_MAP.get(name);
        if(null == adaptor) {
            adaptor = (SsoPluginAdaptor) AppContext.getBean(name);
            if(null == adaptor) {
                adaptor = (SsoPluginAdaptor) AppContext.getBean(name +
"Adaptor");
            }
            if(null != adaptor) {
                ADAPTOR_MAP.put(name, adaptor);
            }
        }
        return adaptor;
    }
}
```

为了便于以后的扩展，有B系统、C系统等其他系统需要集成，我们无需开发多个controller，通过单点登录SsoPluginAdaptor适配器的方式给不同的系统生成不同的url，进行单点跳转：adapter定义如下：

```
/**
* @param request 可以增加一些参数的传递，获取request中的参数，传递到适配器中进行处理
* @return    返回单点登录的url地址    如：http://open.seeyon.com
*/
String getRedirectUrl(HttpServletRequest request);
```

继续实现上面的需求，我们实现此接口：

```
public class BaiduSsoAdaptor implements SsoPluginAdaptor {
```

```
    @Override
    public String getRedirectUrl(HttpServletRequest request) {
        String user = AppContext.currentUserName();
        try {
            // 按照实际需要进行加密
            user = DESUtil.encrypt(user);
        } catch (Exception e) {
            return null;
        }
        // 执行调用第三方接口的逻辑
        return "https://www.baidu.com?user=" + user;
    }

}
```

完成之后，我们将代码部署到系统中，重启后进入单点登录配置界面：

```
<!-- spring bean 的配置 -->
<bean id="baiduSso" class="com.seeyon.apps.ssoplugin.adaptor.BaiduSsoAdaptor" />
```

可见。我们上面配置的springbean id为 baiduSso，那么我们对应配置的单点登录地址为：

> http://127.0.0.1/seeyon/thirdSso.do?ssoname=baiduSso¶m0=xxx

其中param0等可以自行添加，然后到adapter中进行解析即可。

最后，第三方通过获取加密参数，按照他自己的方式解密即可。

4.3.3 用户密码认证

略。

4.3.4 第三方token认证

以腾讯邮箱的登录方式为例，腾讯邮箱单点登录需要先获取access_token，然后调用接口根据用户邮箱获取单点登录地址。

这边不对CIP接口的配置方式做详细说明，大家可以参考文档中附带的腾讯邮箱.zip，将此zip导入到应用接入中即可查看相应的配置方式。

不熟悉的人员，这里推荐采用第三方认证机制方式进行开发，实现一个适配器，返回登录地址即可。

```
@Override
public String getRedirectUrl(HttpServletRequest request) {
    V3xOrgMember member = Functions.getMember(AppContext.currentUserId());
    String email = member.getEmailAddress();
    if (null == email) {
        log.error("没有设置邮箱号，无法进行单点登录!");
        return null;
    }
    try {
        String url = ExMailInterfaceKit.getSsoUrl(email);
        response.sendRedirect(url);
    } catch (Exception e) {
        log.error("单点登录失败:", e);
        super.rendJavaScript(response, "alert('单点登录失败:" + e.getMessage() +
"!');");
        return null;
    }
    return null;
}
```

```
public class ExMailInterfaceKit {
```

```
    private static Token token;

    public synchronized static String getToken(String key) throws Exception {
        // token不存在或者已经失效
        if(null == token || !token.isInvalid(-100)) {
            // 重新获取token
            String secret = "";
            if(key.equals(ExMailConstants.SSO_KEY)) {
                secret = ExMailConstants.SSO_SECRET;
            } else {
                secret = ExMailConstants.UNREAD_SECRET;
            }
            String url = ExMailConstants.TOKEN_URL + "?corpid=" +
ExMailConstants.CORP_ID + "&corpsecret=" + secret;
            try {
                String res = HttpKit.get(url);
                JSONObject obj = JSON.parseObject(res);
                int code = obj.getIntValue("code");
                if (code == 0) {
                    token = new Token(new Date(),
obj.getString("access_token"));
                } else {
                    throw new Exception("获取token失败：" +
obj.getString("errmsg"));
                }
            } catch (Exception e) {
                throw new Exception("获取token失败:" + key);
            }
        }
        return token.getAccessToken();
    }

    public static String getSsoUrl(String email) throws Exception {

        String url = ExMailConstants.SSO_URL + "?access_token=" +
getToken(ExMailConstants.SSO_KEY) + "&userid=" + email;
        try {
            String res = HttpKit.get(url);
            JSONObject obj = JSON.parseObject(res);
            int code = obj.getIntValue("code");
            if (code == 0) {
                return obj.getString("login_url");
            } else {
                throw new Exception("单点登录失败：" + obj.getString("errmsg"));
            }
        } catch (Exception e) {
            throw new Exception("单点登录失败" + e.getMessage());
        }
    }

}
```

4.3.5 CAS模式认证

略，参考word文档：CAS-统一认证.docx。

4.4 第三方单点到A8对应页面

如果要直接跳转到A8的对应页面中，我们需要先进行单点登录，然后再进行跳转，产品提供了相应的包装方法：

```
String ticket = request.getParameter("ticket");
String url="collaboration/collaboration.do?
method=summary&openFrom=listPending&affairId=-932843950278492";
TicketInfo ticketInfo =
com.seeyon.ctp.portal.sso.SSOTicketBean.getTicketInfo(ticket);
if(ticketInfo != null) {
  response.sendRedirect(com.seeyon.ctp.portal.sso.SSOTicketBean.makeURLOfSSOTick
et(ticket, url));
}
```

其中，ticket是用户在单点登录时握手类中的ticket。

如果上面的情况无法满足需求，这里给出A8改造示例：

> 需求：由于A8的token中可以附带用户信息，第三方想要传递token进行认证。

实现上述需求，我们需要修改源码，注意不要影响到原先的逻辑：

```
//ThirdpartyController.java
@NeedlessCheckLogin
    public ModelAndView access(HttpServletRequest request, HttpServletResponse
response) throws Exception {
        // 略
        //因为微信汉字名ticket可能存在+号被屏蔽为空格，这里增加转换
        if(ticketInfo==null){
            ticket = ticket.replaceAll(" ", "+");
            ticketInfo=SSOTicketBean.getTicketInfoByticketOrname(ticket);
        }
        // 新增代码  所有产品分支都走完了 未获取到用户信心，那么从token中获取
        if(null == ticketInfo) {
            // 客开 ：从token中获取用户信息
            String token = request.getParameter("token");
            User u = ServiceManager.getBindUser(token);
            if(null != u) {
                memberId = u.getId();
            } else {
                mv.addObject("ExceptionKey", "token中不含用户信息，无法单点登录");
                //mv.addObject("ExceptionKey", "链接");
                return mv;
            }
        }else {
            loginTime = ticketInfo.getCreateDate().getTime();
            if("weixin".equals(ticketInfo.getFrom())) {
                userAgentFrom = login_useragent_from.weixin;
            }
            if (ticketInfo != null) {
                memberId = ticketInfo.getMemberId();
            }
        }
        // 新增tokne逻辑完成
        // 略
}
```

4.5 A8配置单点系统显示位置

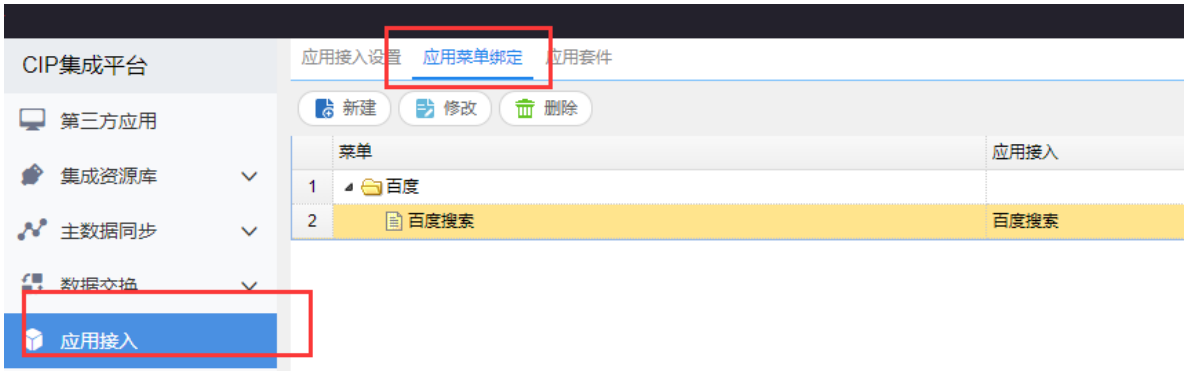
4.4.1 门户空间显示

我们系统配置完成之后，给相应的用户授权，用户即可在空间中看到。

4.5.2 应用菜单配置

可以生成一个第三方的系统菜单，将单点系统挂载入到菜单上。

如果要用于移动端，则需要新增应用套件。



4.5.3 应用磁贴配置

可以将单点系统配置到应用磁贴上。

五、消息集成

和人员绑定一样，A8对于消息集成提供了两种模式，主动和被动接收，两种模式都需要按照消息内容格式提供数据。

单条消息格式如下：

```
{
    "thirdpartyRegisterCode": "注册系统编码",
    "thirdpartyMessageId": "第三方系统消息主键",
    "messageContent": "消息内容",
    "thirdpartySenderId": "第三方系统发送者主键",
    "thirdpartyReceiverId": "第三方系统接收人主键",
    "creation_date": "消息创建日期，格式yyyy-MM-dd HH:mm",
    "downloadUrl": "原生app的下载地址",
    "messageH5URL": "H5穿透地址",
    "messageURL": "PC穿透地址",
    "noneBindingSender":"登录名称/人员编码/手机号/电子邮件",
    "noneBindingReceiver":"登录名称/人员编码/手机号/电子邮件"
}

// 如果是多条，那么外面的key为messages
```

5.1 A8主动拉取 6

主动拉取分为url和接口两种类型，主要区别在于url无需配置参数，并且无需映射字段，接口可以配置参数并且可以映射字段。

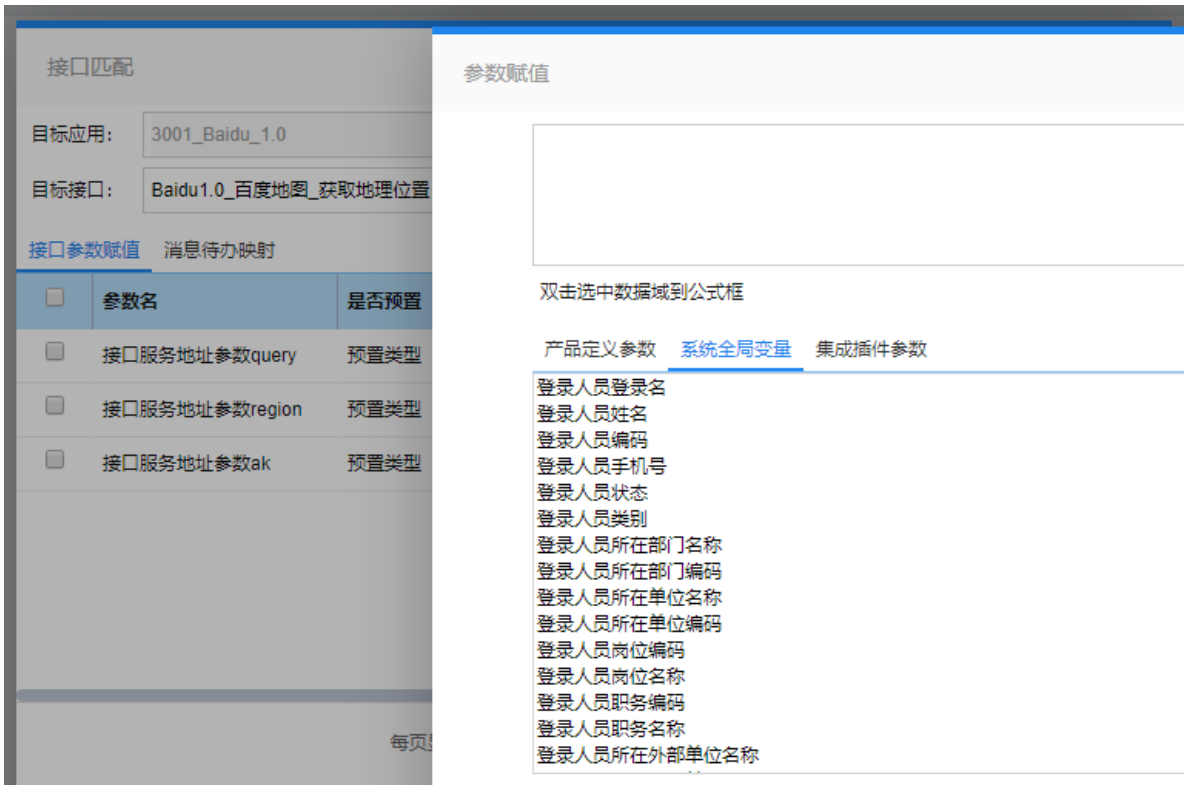
接口类型	接口地址	请求内容类型	输入参数	返回参数
URL	自定义	application/json	无需配置	消息对象list
接口	自定义	application/json		消息对象list

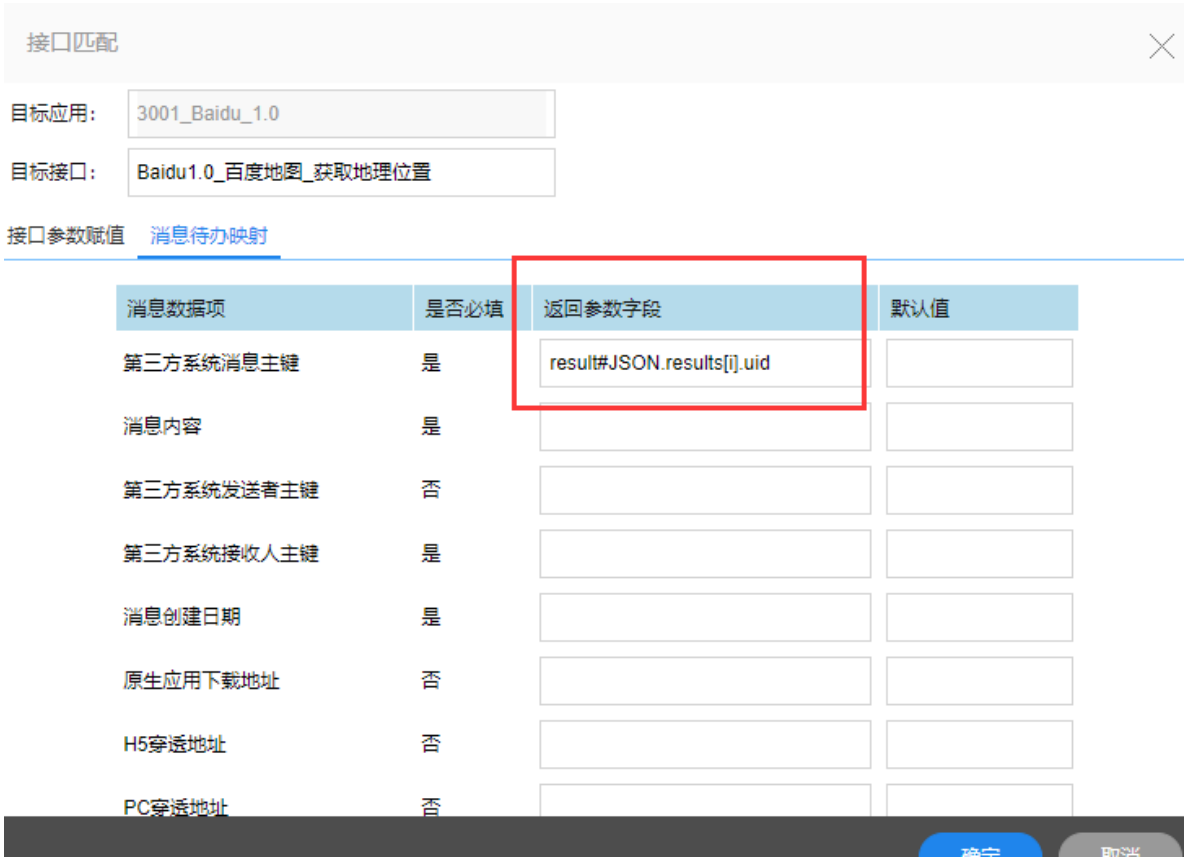
除了上述方式，依然可以通过插件开发方式后台生成定时任务，通过内部API生成消息：

```
// ThirdpartyMessageApiImpl.java
@Override
public CIPErrorMessage receiveListMessage(String
registerCodeGlobe,List<MessageReceiveVo> listVo) throws BusinessException {
    CIPErrorMessage error = new CIPErrorMessage();
```

```
    List<MessageReceiveVo> availableMessage = new ArrayList<MessageReceiveVo>();
    //点对点时模式校验接收人
    boolean isPoint = CIPUtil.isPonint2Point(registerCodeGlobe);
    for (int i = 0; i < listVo.size(); i++) {
        CIPErrorMessage msg = checkMessage(isPoint,
listVo.get(i),registerCodeGlobe);
        if(!msg.isSuccess()){
            error.addAllErrorMsg(msg.getErrorMsgs());
            continue;
        }
        availableMessage.add(listVo.get(i));
    }
    if(!error.isSuccess()){
        Map<String,Object> param=new LinkedHashMap<String,Object>();
        param.put("link", true);

        LogMessage actionMessage = new
LogMessage(LogRecodeTypeEnum.Action,"10001","消息");
        LogMessage detailMessage = new
LogMessage(LogRecodeTypeEnum.Detail,"10003",error.toString());

 CIPActionLogFactory.createLog(LogApplicationEnum.APPINTEGRATION,actionMessage,
detailMessage, -1L, null, param);
    }
    CIPErrorMessage msgd =
messageOrPendingConfigManager.sendReceiveMessage(registerCodeGlobe,
availableMessage,GetModeEnum.PUSH);
    if (!msgd.isSuccess()) {
        error.addAllErrorMsg(msgd.getErrorMsgs());
    }
    return error;
}
```

取数接口可以配置如下：





5.2 第三方推送（推荐）

A8提供了单条消息和多条消息推送的接口，第三方系统可以按照A8的rest接口规范[4]进行调用推送数据。

多条消息接口地址：/seeyon/rest/thirdpartyMessage/receive/messageList

单条消息接口地址：/seeyon/rest/thirdpartyMessage/receive/singleMessage

示例代码如下：

```
@Test
public void push() throws Exception {
    OaMessage msg = new OaMessage();
    msg.setThirdpartyRegisterCode("3001");  //我们系统提供，应用注册里面的编码
    msg.setThirdpartyMessageId("5555"); // 消息的唯一标识
    msg.setMessageContent("您有一条公文待处理。");
    msg.setThirdpartySenderId("");// 用户的唯一标识
    msg.setThirdpartyReceiverId("");//接收用户的唯一标识
    msg.setThirdpartySenderName(""); //免绑定不用传，如果不是则需要传绑定的账号
    msg.setThirdpartyReceiverName(""); //免绑定不用传，如果不是则需要传绑定的账号
    msg.setNoneBindingReceiver("dsz");  // 改成发送用户的登录名 我这里的设置的是登录名
免绑定，所以传登录名，如果其他情况则对应修改
    msg.setNoneBindingSender("dsz");  //消息接收者的登录名 同上
    msg.setCreationDate(new Date());
    // messageType  0：PC；1：移动URL接入应用；2：本地H5应用；3：本地原生应用；4：PC&移动
URL接入应用；5：PC&本地H5应用；6：PC&本地原生应用    必填
    // 和应用注册中的类型有关
    msg.setMessageType(4);
    msg.setDownloadUrl("");
    msg.setMessageURL("http://www.baidu.com");
    msg.setMessageH5URL("http://www.baidu.com");
```

```
    /*msg.setAppParam("sycmpiphone://m1services?serviceParams=
{'module':'authentication/home','username':'{$userName$}','archiveID':'-1','from
url':'com.others.path'}\r\n"+"sycmpandroidphone://m1services?serviceParams=
{'module':'authentication/home','username':'{$userName$}','archiveID':-1,'fromur
l':'com.others.path}|das|sdsa|asdsa");*/
    String res = HttpKit.post(URL + "/thirdpartyMessage/receive/singleMessage?
token=" + token, JSONUtil.toJSONString(msg));
    System.out.println(res);
}
```

测试消息推送：

```
{
  "errorMsgs" : [ {
    "errorDetail" : "第三方注册编码：3001",
    "errorCode" : 206,
    "errorType" : "消息配置已经停用"
  } ],
  "success" : false
}
```

可以看到我们还没有启用消息推送的配置，回到单位管理员中，进入对应应用设置消息推送启用：


```
{
  "errorMsgs" : [ ],
  "success" : true
}
```

登录对应人员账号，可以成功看到消息通知：

5.3 消息认证模式

待办和消息都采用了统一的消息认证模式，总共有两大类：v5认证和自定义认证。自定义认证中包含了第三方认证、cas认证等方式。

5.3.1 v5认证

我们在单点登录中讲到，如果采用A8的标准单点登录模式，会给单点登录附带一个参数ticket，第三方拿到ticket参数后回调A8接口seeyon/thirdpartyController.do?ticket=${ticket}获取实际的用户信息，消息和待办也相同，唯一的区别在于，单点登录附带的参数名称为ticket，而待办和消息附带的参数名称为v5ticket。

A8会我们设置的免绑定类型返回对应的绑定用户信息：

```
String username = null;
ThirdpartyTicketManager.TicketInfo ticketInfo =
ThirdpartyTicketManager.getInstance().getTicketInfo(ticket);

if (ticketInfo == null) {
    if(log.isDebugEnabled()){
        log.debug("ticket is not from thirdpartticket:"+ticket);
    }
    TicketInfo ssoTicketInf =
SSOTicketManager.getInstance().getTicketInfo(ticket);
    if (ssoTicketInf != null) {
        username = ssoTicketInf.getUsername();
        if(log.isDebugEnabled()){
            log.debug("ticket is from ssoTicket,userName:"+username);
        }
    }
}
ThirdpartySpace tpSpace = null;
if (ticketInfo != null) {
    if(log.isDebugEnabled()){
        log.debug("ticket is from thirdpartticket:"+ticket);
    }
    tpSpace = thirdpartySpaceManagerApi.get(ticketInfo.getSpaceId());

    if (tpSpace != null) {
        username = ticketInfo.getLoginName();
        if(log.isDebugEnabled()){
            log.debug("ticket is from thirdpartticket,userName:"+username);
        }
    }
}
if (username != null) {
    V3xOrgMember member = orgManager.getMemberByLoginName(username);
    // 做用户映射或做处理后才能返回第三方登录账号，否则返回OA登录账号
    String thirdUsername = username;
    if (null != tpSpace) {
        thirdUsername = tpSpace.getThirdpartyLoginName(username);
    }
    PrintWriter out = response.getWriter();
    response.addHeader("LoginName", java.net.URLEncoder.encode(thirdUsername,
"UTF-8"));
    if (null != member) {
        response.addHeader("MemberId", String.valueOf(member.getId()));
        response.addHeader("MemberName",
java.net.URLEncoder.encode(member.getName(), "UTF-8"));
    } else {
        log.warn("OA集成第三方系统通过登录名称找人失败：" + username + " " +
thirdUsername);
```

```
        //CIPActionLogFactory.createLog(LogApplicationEnum.APPINTEGRATION, "单点登录","OA集成第三方系统通过登录名称找人失败：" + username + " " + thirdUsername, null, null, null);
    }
    if (AppContext.hasPlugin("m3")) {

 response.addHeader("M3URL",java.net.URLEncoder.encode(PNSPropertyUtils.getInstance().getM3URL(), "UTF-8"));
    }
    out.println(java.net.URLEncoder.encode(thirdUsername, "UTF-8"));
}
```

例如，第三方推送的地址为https://www.baidu.com，那么我们访问此地址时，实际上会访问https://www.baidu.com?v5ticket=xxxxxxx，第三方从url中获取到v5ticket参数，回调A8接口，拿到实际的用户后再跳转到对应的待办（已办）详情中。

5.3.2 第三方认证

同单点登录第三方认证。

如果是系统对于保密性要求不高，可以让第三方对用户名进行直接的加密，直接放到跳转链接中，此方式不安全，无论谁取到此地址都能够进行单点登录，不建议使用。

如果有一些特殊的方式，例如，第三方需要先进行单点登录然后再跳转到系统对应的事项中，我们需要自己对url进行包装，实现方式可以参考A8的单点登录跳转方式，通过ThirpartyContoller.java中的access方法进行实现。

总之，为了安全性，最好采用ticket的方式，如果第三方有自己的方式，那么可以按照第三方的规则，我们自己实现controller进行包装，然后到自己的跳转方法中按照第三方需要的方式进行拼接加密参数等实现单点跳转！

六、待办集成

待办的数据格式说明：

```
{
    "registerCode": "注册系统编码",
    "taskId": "第三方系统待办主键",
    "title": "待办标题",
    "senderName": "待办发起人姓名",
    "classify": "类别",
    "contentType": "内容类型",
    "state": "状态0待办，1已办",
    "thirdSenderId": "第三方系统发送者主键",
    "thirdReceiverId": "第三方系统接收人主键",
    "creationDate": "待办发起日期，格式yyyy-MM-dd HH:mm",
    "content": "原生app的下载地址",
    "h5url": "H5穿透地址",
    "url": "PC穿透地址",
    "noneBindingSender":"登录名称/人员编码/手机号/电子邮件",
    "noneBindingReceiver":"登录名称/人员编码/手机号/电子邮件"
}
```

6.1 A8主动拉取[6]

和消息推送类似，这里不再说明。

6.2 第三方推送（推荐）

A8提供了单条待办和多条待办推送的接口，第三方系统可以按照A8的rest接口规范[4]进行调用推送数据。

单条待办接口地址：/seeyon/rest/thirdpartyPending/receive

多条待办接口地址：/seeyon/rest/thirdpartyPending/receive/pendings

示例代码如下：

```
@Test
public void push() throws Exception {
    OaPending pend = new OaPending();
    pend.setTaskId("123456789");  // 第三方系统待办的唯一标识
    pend.setRegisterCode("3001");       //我们系统提供，应用注册里面的编码
    pend.setTitle("关于学习十八大精神的会议纪要");
    pend.setSenderName("");    // 免绑定不用传，如果不是则需要传绑定的账号
    pend.setThirdReceiverId("");  // 同上
    pend.setNoneBindingReceiver("dsz"); // 改成发送用户的登录名 我这里的设置的是登录名 免绑定，所以传登录名，如果其他情况则对应修改
    pend.setNoneBindingSender("dsz");  //消息接收者的登录名 同上
    pend.setCreationDate(new Date());
    pend.setState(0);
    pend.setSubState(null);
    pend.setUrl("http://www.baidu.com");    //处理地址
    pend.setH5url("http://www.baidu.com");
    String res = HttpKit.post(URL + "/thirdpartyPending/receive?token=" + token, JSONUtil.toJSONString(pend));
    System.out.println(res);
}
```

进行待办的推送测试，同样的，我们需要先启用待办的推送：

```
{
  "errorMsgs" : [ ],
  "success" : true
}
```

得到上面的结果说明推送成功，登录账号查看相关待办，需要注意的是，CIP的待办和A8的标准待办并非在同一个待办栏目，需要用管理员或者个人账号进行配置：

6.3 更新待办状态

更新待办状态接口地址：/seeyon/rest/thirdpartyPending/updatePendingState

字段名称	描述	是否必填
taskId	第三方待办主键（保证唯一）	必填
registerCode	系统注册编码	必填
state	状态：0:未办理；1:已办理	必填
subState	处理后状态：0/1/2/3同意已办/不同意已办/取消/驳回	必填

需要注意的是，由于目前产品问题，暂时未考虑到已办和待办的地址可能存在差别，所以未更新pc端地址和移动端H5的参数，需要修改A8源码进行适配。

除了调用上述接口外，我们可以调用待办推送接口，只要保证taskId一致即可，state和substate改成对应的状态即可。

具体修改代码位置如下：

```
//ThirdPendingManagerImpl
@Override
public boolean saveOrupdatePendingByTaskId(ThirdPendingPO pending) {
    List<ThirdPendingPO> pendings = thirdPendingDao
```

```
        .findPending(pending.getRegisterId(),pending.getTaskId());
    try {
        if (pendings.isEmpty()) {
            // 新增
            pending.setIdIfNew();
            // receiberId是必填字段，暂时还不确定用户绑定过程，先用预留
            //              pending.setReceiverId(1l);
            pending.setCreateDate(new Date());
            pending.setUpdateDate(new Date());
            thirdPendingDao.savePending(pending);
            // 添加全文检索
            if (AppContext.hasPlugin("index")) {
                try {
                    indexApi.add(pending.getId(),
ApplicationCategoryEnum.ThirdPartyIntegration.getKey());
                } catch (BusinessException e) {
                    log.error("从indexManager添加第三方待办检索项。", e);
                }
            }
        } else {
            // 更新
            // 如果要更新地址等参数，这里需要自行修改！！
            // 只更新state和substate状态值
            pendings.get(0).setState(pending.getState());
            pendings.get(0).setSubState(pending.getSubState());
            pendings.get(0).setUpdateDate(new Date());
            thirdPendingDao.updatePending(pendings.get(0));
            if(pending.getState()==0){
                // 删除全文检索
                if (AppContext.hasPlugin("index")) {
                    try {
                        indexApi.delete(pending.getId(),
ApplicationCategoryEnum.ThirdPartyIntegration.getKey());
                    } catch (BusinessException e) {
                        log.error("从indexManager删除第三方待办检索项。", e);
                    }
                }
            }
        }
        return true;
    } catch (Exception e) {
        log.error(e.getMessage(), e);
    }
    return false;
}
```

6.4 待办认证模式

同消息，见5.3.

七、CIP接口库

第三方应用接口库支持的接口远程协议包括：WebService/Rest/JCO/Http.

具体规范：

1、WebService是根据WSDL文件进行自动解析的，所有必须具有WSDL文件

2、WebService自动解析不支持复杂类型的参数

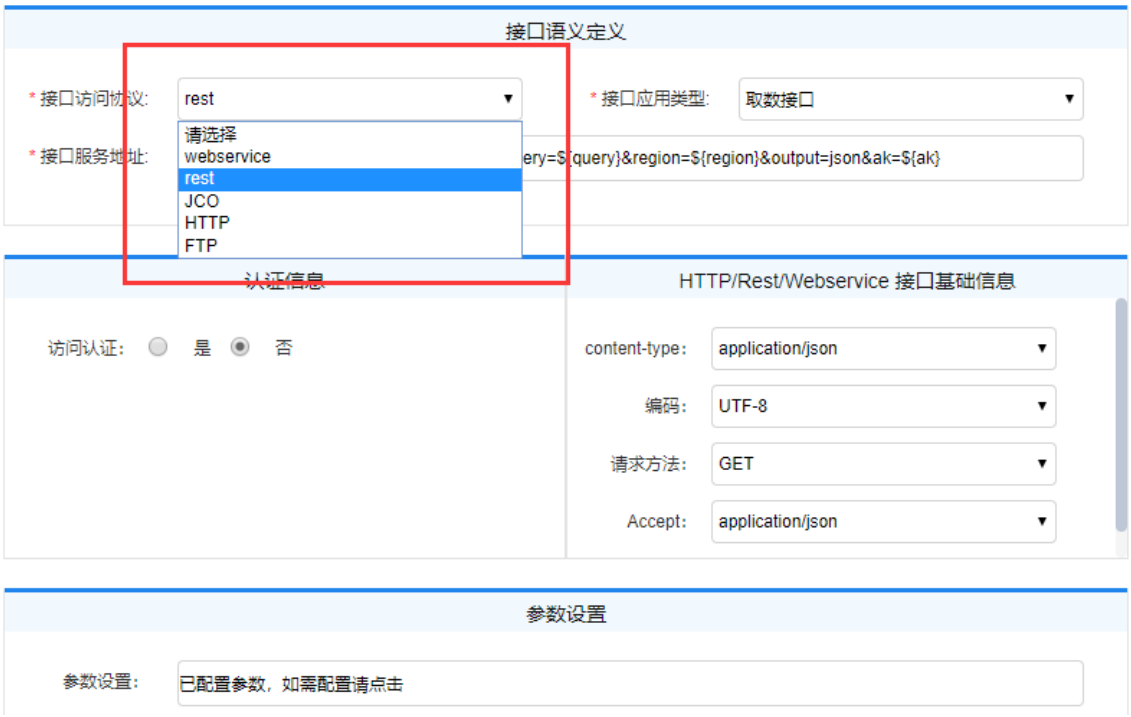
3、HTTP/REST接口下返回参数最多只能配置一个(可以是xml或JSON等复杂类型)

4、Content-type为text/xml或application/json时，传入参数最多只能配置一个(可以是xml或JSON等复杂类型)

5、认证引用不能多次引用同一个认证接口

6、不能交叉引用认证接口，比如接口A（本身为认证接口）引用了认证接口B，则B不能再引用认证接口A

CIP接口库主要用于配置第三方的接口，实现0代码的方式做到数据同步、消息、待办集成，流程事件等功能，一般情况来说除非双方数据结构分成吻合，否则都是需要开发的。



接口类型包含导出接口、取数接口、认证接口和页面跳转接口四种类型，前面两种主要用于数据操作，认证接口常用于第三方接口调用前需要先获取token等，页面跳转常用于单点登录。

我们这里以百度地图的一个api为例，根据地址和省份获取地址信息：

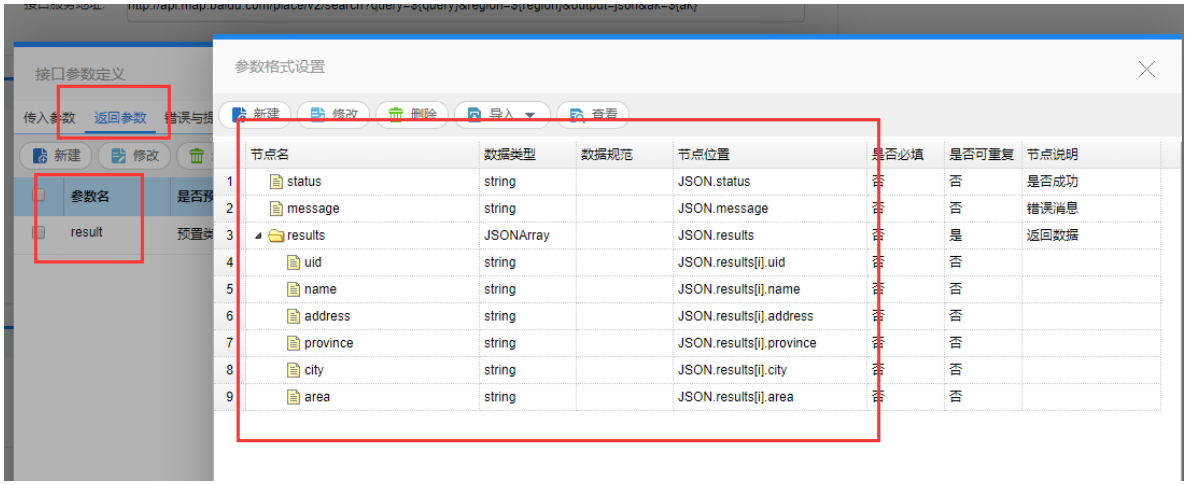
> http://api.map.baidu.com/place/v2/search?query=ATM机&region=北京&output=json&ak=您的ak //GET请求

那我们配置的接口服务地址为：

```
http://api.map.baidu.com/place/v2/search?query=${query}&region=${region}&output=json&ak=${ak}
```

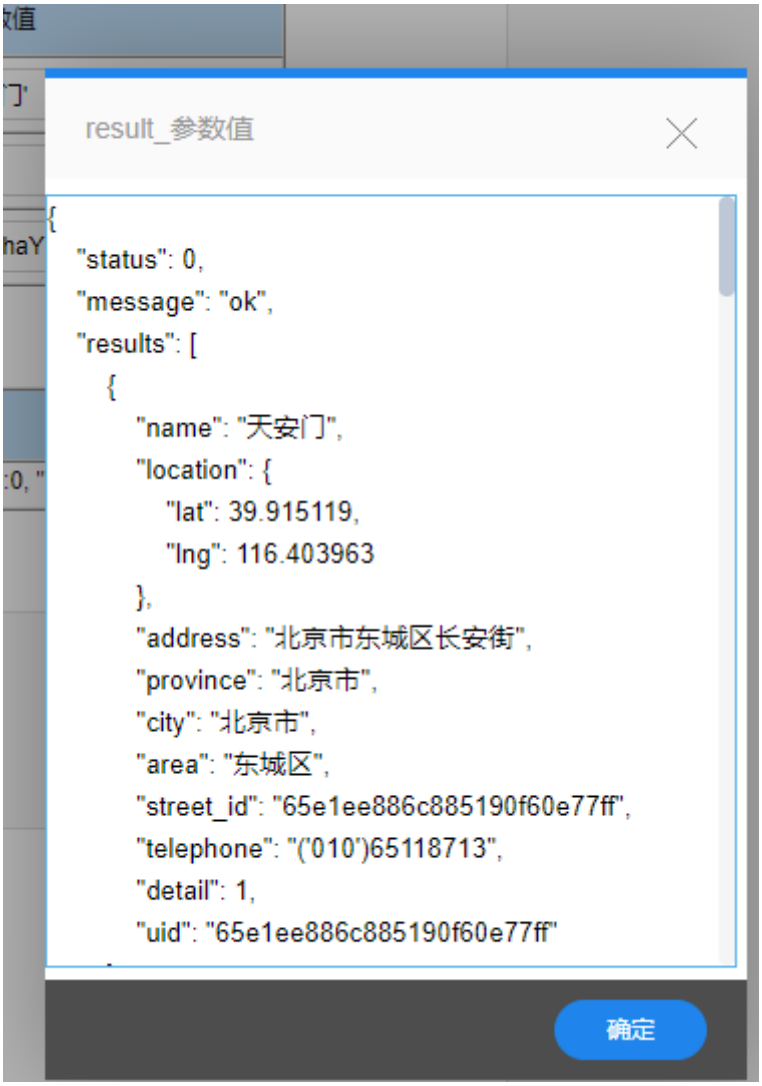
url中的变量我们可以直接使用${变量名}的方式进行声明，如果是post接口，那么我们再下面的参数设置中设置body里面的信息，注意content-type设置为正确的类型。

然后根据接口返回值设置返回参数：



配置完成后，我们可以进行测试：

设置好对应的参数后，我们执行调试：



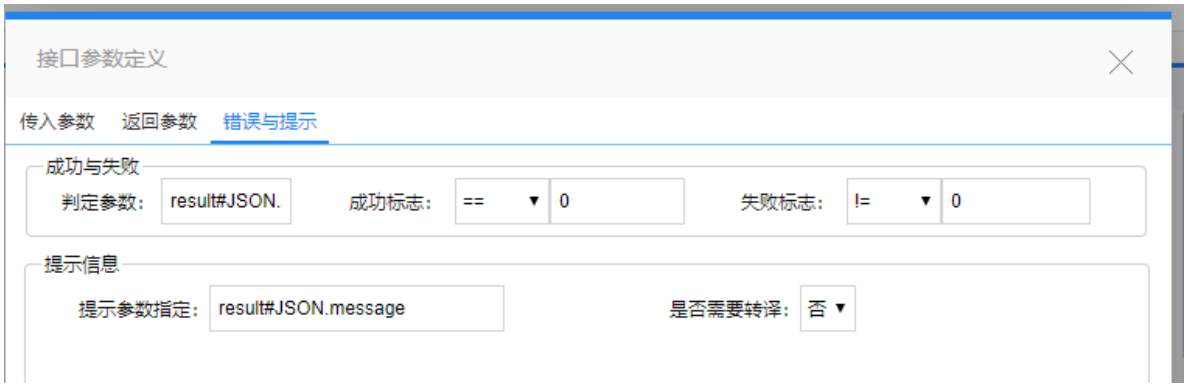
可以成功获取到相应的结果。

7.1 流程事件

CIP接口可以用于配置流程事件，例如，想要在流程发起前调用第三方接口进行校验。

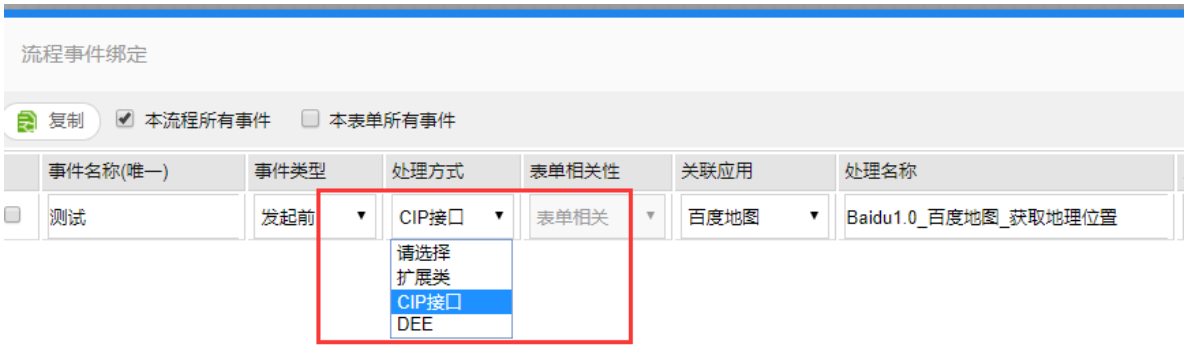
> 需求：我们调用百度地图的API，如果调用失败，则返回错误消息。

在实现此功能之前，我们需要先对接口进行错误提示配置：



根据返回的数据设置好相关的成功和失败判断规则，上方设置了status==0位成功，其余失败。

流程事件可以配置三种类型的任务：CIP接口、扩展类[7]和DEE任务[8]。



如果采用使用类，我们只需要实现com.seeyon.ctp.workflow.event.AbstractWorkflowEvent抽象类的相关方法即可：

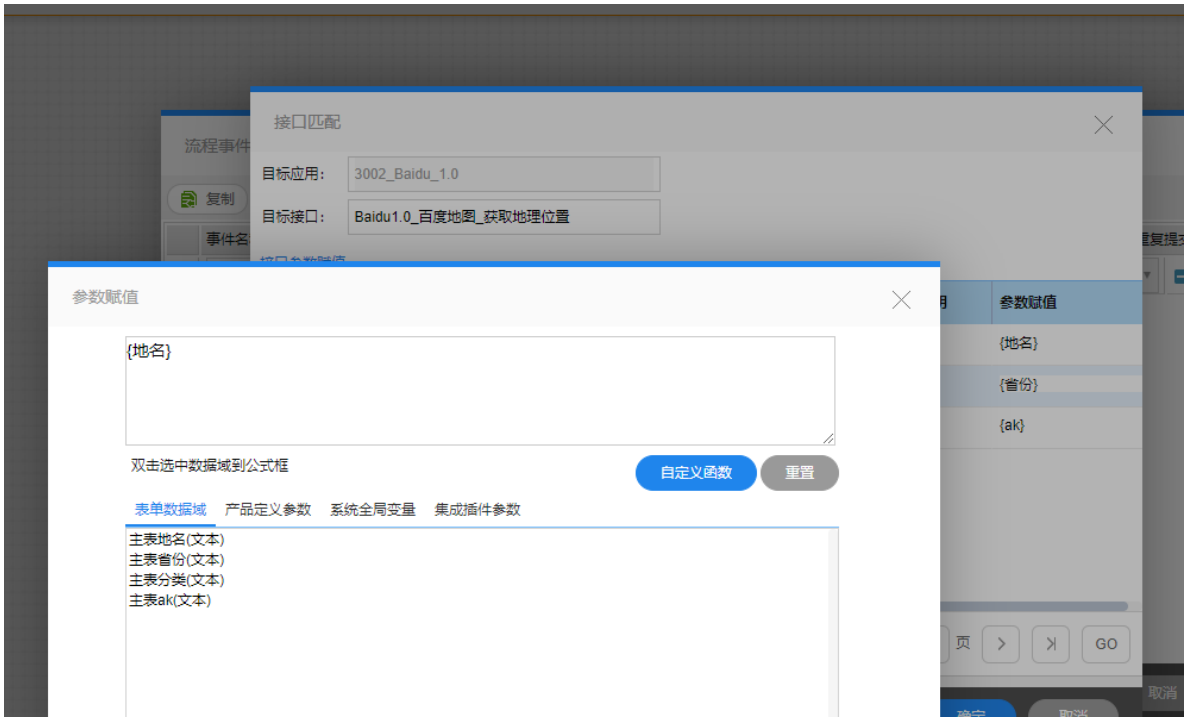
注意：在7.1sp1版本一定要实现getType()方法，如果用于表单的话请返回WorkflowEventConstants.WorkflowEventType.form，否则，无法选择到对应的事件。

```
public abstract class AbstractWorkflowEvent {

    /**
     * 唯一标示，一旦生成，不可变更
     * @return
     */
    public abstract String getId();

    /**
     * 事件显示名称
     * @return
     */
    public abstract String getLabel();

    /**
     * 必须是WorkflowEventConstants.WorkflowEventType中的枚举值，默认为扩展类型Ext。
     * 如果用于表单，请返回form
     * 7.1sp1以前版本无此接口
     * @return
     */
    public WorkflowEventType getType(){
        return WorkflowEventConstants.WorkflowEventType.Ext;
    }

    /**
     *
     * 高级扩展的时候，可以进行事件的高级设置
     * 高级开发的设计界面 - 当事件类型为DEETrigger的时候，可以点击按钮弹出一个事件自己的页面，然后设置保存到自己的数据库中。
     * 页面的返回值必须json的格式，并且包含name,id
     * <pre>
     * eg.
     * {
     *  id : xx, //外部页面设置结果的唯一标记
     *  label : yy //外部页面设置以后回填高级开发界面的的输入框的显示值
     * }
     * </pre>
     * @return
     */
    public WorkflowEventAdvancePageInfo getOpenPageInfo() {
        return null;
```

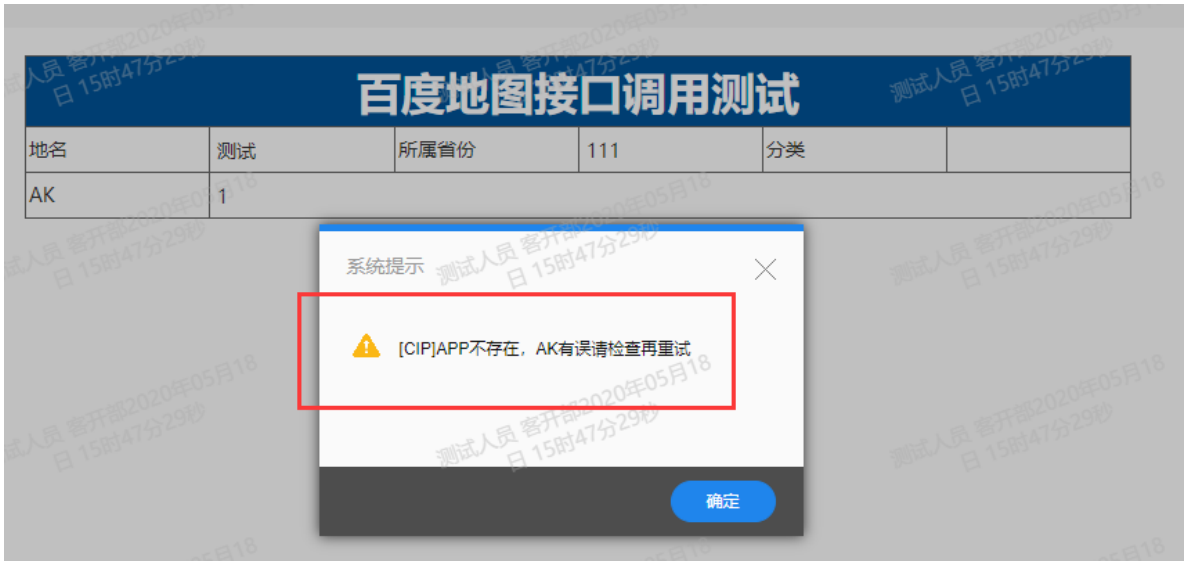
```
    }

    //发起前事件
    public WorkflowEventResult onBeforeStart(WorkflowEventData data){return
null;}
    //发起事件
    public void onStart(WorkflowEventData data){}

    //处理前事件
    public WorkflowEventResult onBeforeFinishWorkitem(WorkflowEventData data)
{return null;}
    //处理事件
    public void onFinishWorkitem(WorkflowEventData data){}

    //终止前事件
    public WorkflowEventResult onBeforeStop(WorkflowEventData data){return
null;}
    //终止事件
    public void onStop(WorkflowEventData data){}

    //回退前事件
    public WorkflowEventResult onBeforeStepBack(WorkflowEventData data){return
null;}
    //回退事件
    public void onStepBack(WorkflowEventData data){}

    //撤销前事件
    public WorkflowEventResult onBeforeCancel(WorkflowEventData data){return
null;}
    //撤销事件
    public void onCancel(WorkflowEventData data){}

    //取回前事件
    public WorkflowEventResult onBeforeTakeBack(WorkflowEventData data){return
null;}
    //取回事件
    public void onTakeBack(WorkflowEventData data){}

    //结束事件
    public void onProcessFinished(WorkflowEventData data){}
}
```

流程前事件我们可以进行阻塞，返回WorkflowEventResult，其中包含alertMessage即可。

配置到流程事件中，我们选择发起前事件，我们可以对参数设置为表单上面的数据，也可以设置一些全局参数等：

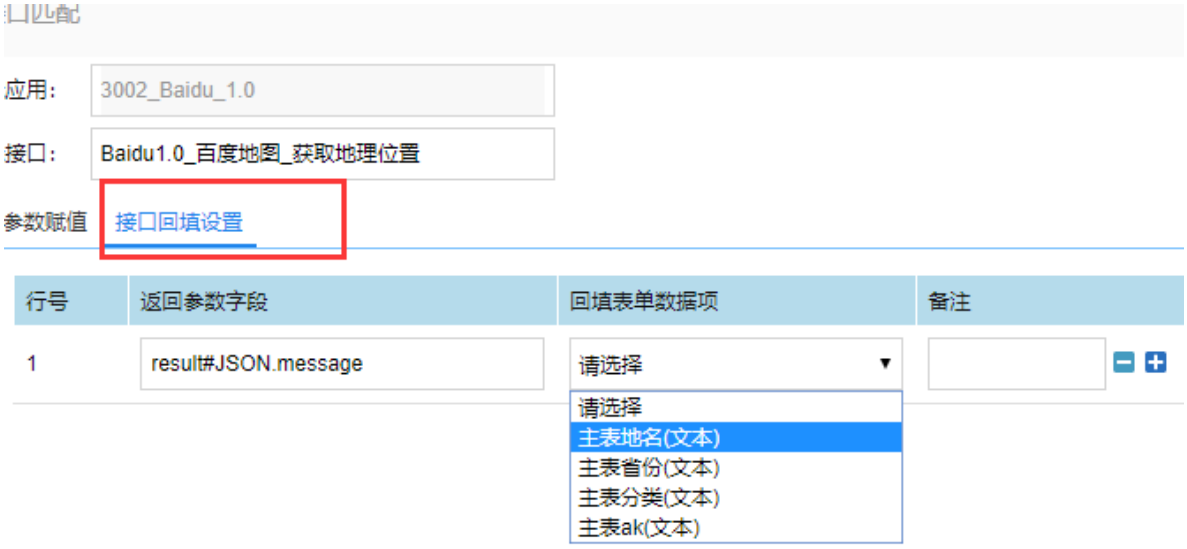


配置完成后，可以到前台发起测试：



以上就是通过CIP接口实现0代码实现流程的发起前校验。

发起后，提交前后，结束这3种可以支持回填。



7.2 产品事件

除了将CIP接口应用于流程事件，我们还可以将接口配置到产品的相关事件中。

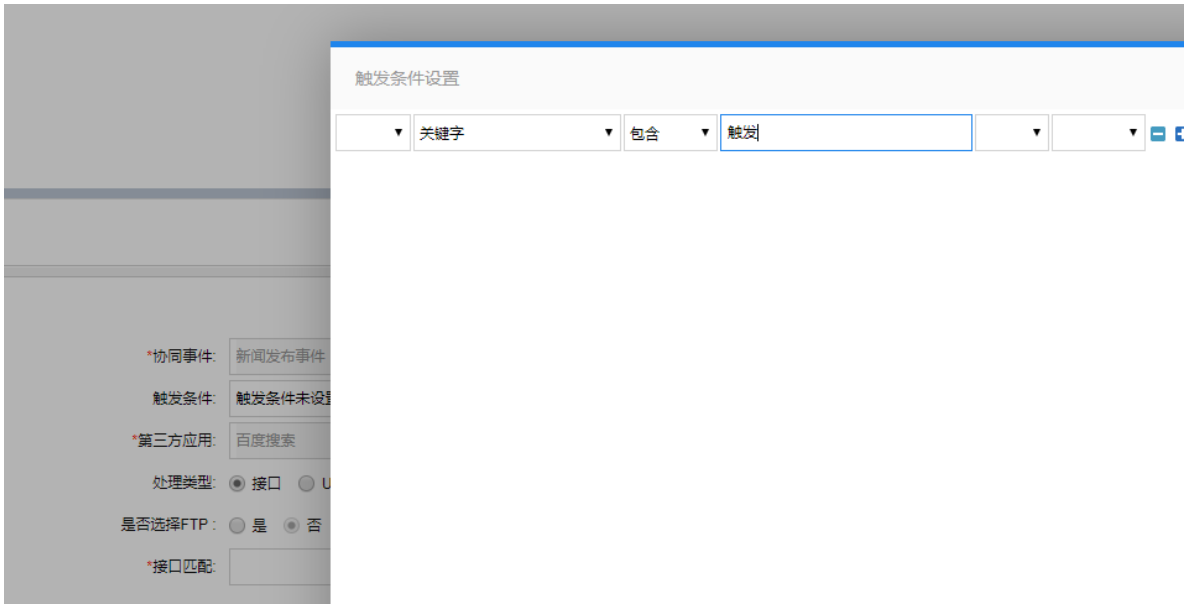
> 需求：发起新闻的时候触发一个流程。

我们通过A8系统里面已有的相关接口和事件即可完成配置工作。

由于新闻中可能存在附件，所以我们这里先配置一个ftp服务器接口（需要先在windows上配置ftp服务），如果有问题请参考：ftp服务器配置

首先，我们先新增一个事件绑定：

新增事件，可以设置一些简单的触发条件：



注意：应用接入和上面的ftp服务我们都是配置到A8下面的。否则这边选择不到。

参数赋值

	节点名	数据类型	数据规范	是否必填	节点位置	是否可重复	节点说明	节点赋值
1	templateCode	string		是	JSON.templateCode	否	模板编号	'test'
2	senderLoginName	string		是	JSON.senderLoginNam	否	发起者的登录名	{登录人员登
3	subject	string		是	JSON.subject	否	协同的标题	{新闻标题}
4	data	string		否	JSON.data	否	HTML正文流程为	{正文信息}
5	attachments	JSONArr		否	JSON.attachments	是	附件，Long型Lis	可赋值
6	param	string		是	JSON.param	否	为控制是否流程	'0'
7	transfertype	string		是	JSON.transfertype	否	data格式，xml:	'json'
8	formContentAtt	JSONArr		否	JSON.formContentAtt	是	表单附件 组件传	可赋值

发起新闻进行测试：

注意，上传ftp产品代码存在一点bug，需要做微调：

```
//FTPProcessor
localI = org.apache.commons.io.FileUtils.openInputStream(file);
                String storeFileName = Datetimes.format(new Date(),
"yyyyMMddHHmmss") + "_" + v3xFile.getFilename();
                isTrue = ftpclient.storeFile(new
String(storeFileName.getBytes(), "iso-8859-1"), localI);
```

以上是事件的触发。

7.3 超级节点

超级节点主要用于执行一些第三方的数据推送，为了保证事务的一致性，我们通常采用超级节点的方式，我们通过配置一个表单流程发起接口，来测试超级节点的作用。

首先，进入CIP接口库配置一个表单流程发起接口（标准的是HTML流程，不符合要求，我们需要自己封装一下，第三方的接口也可以参考），配置前，先在应用中设置好对应的参数（ip/port/restuser/restpwd):

我们以下面的一个简单的表单为例：

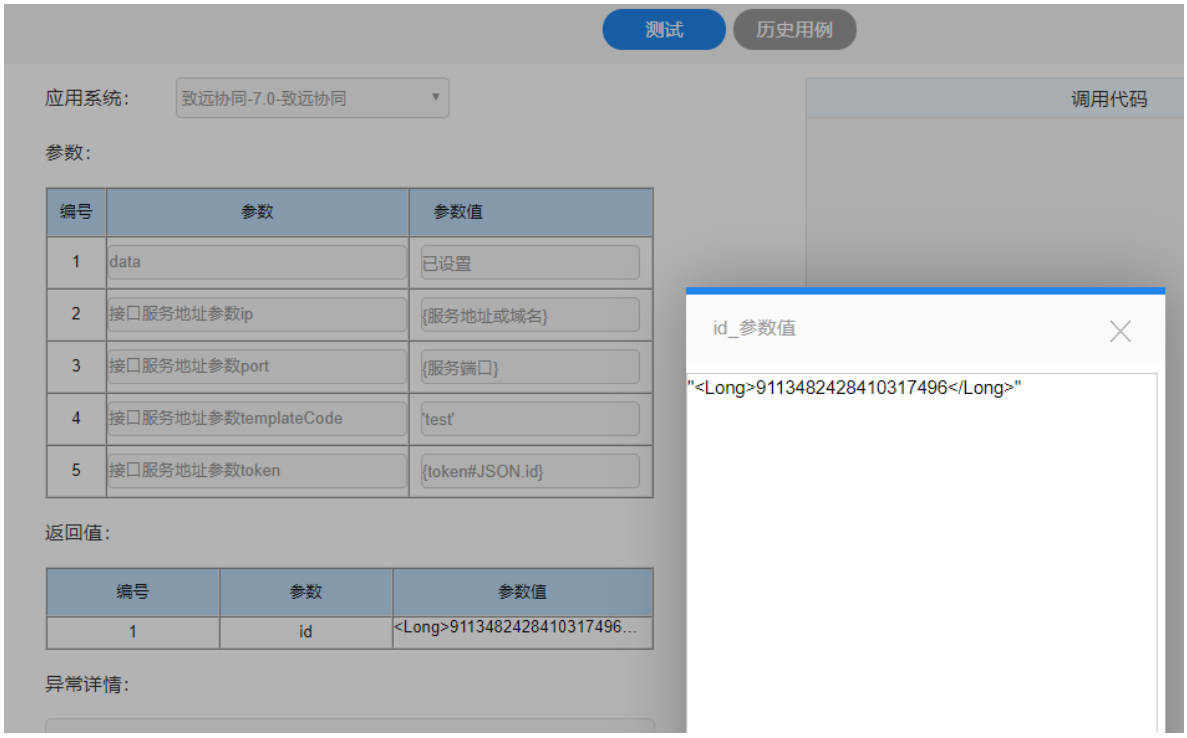
百度地图接口调用测试					
地名		所属省份		分类	
AK					

接口配置如下，服务地址为：

```
http://${ip}:${port}/seeyon/rest/flow/${templateCode}?token=${token}
```

	节点名	数据类型	数据规范	节点位置	是否必填	是否可重复	节点说明
1	data	JSON		JSON.data	否	否	
2	地名	string		JSON.data.地名	否	否	
3	省份	string		JSON.data.省份	否	否	
4	所属	string		JSON.data.所属	否	否	
5	ak	string		JSON.data.ak	否	否	
6	subject	string		JSON.subject	否	否	标题
7	senderLoginName	string		JSON.senderLoginName	否	否	流程发起者
8	transfertype	string		JSON.transfertype	否	否	数据格式

配置完成后，进行接口测试：



接口配置完成后，我们到业务流程集中任意找到一个表单，配置一个超级节点进行超级节点的引用：

然后登录前台用户，发起对应的流程进行测试，如果接口测试失败，则会回退，并且将错误信息放在评论区：

我们修正接口的错误后，重新发起，可以看到流程成功触发：

回到之前的 流程中也能看到相关的信息正确的回填了：

申请部门		申请人	测试112
会议主题	超级节点发起流程		上周会表
会议时间		至	
会议室			
负责人	测试2	主办部门	
联系人	测试1	联系电话	173134564
与会人员		出席人数	
与会范围			
会议室所需设备	话筒 电脑 有线网络 无线网络 投影仪 扩音器		
会议内容	-4405468102023364438		
附件			
备注	测试备注		

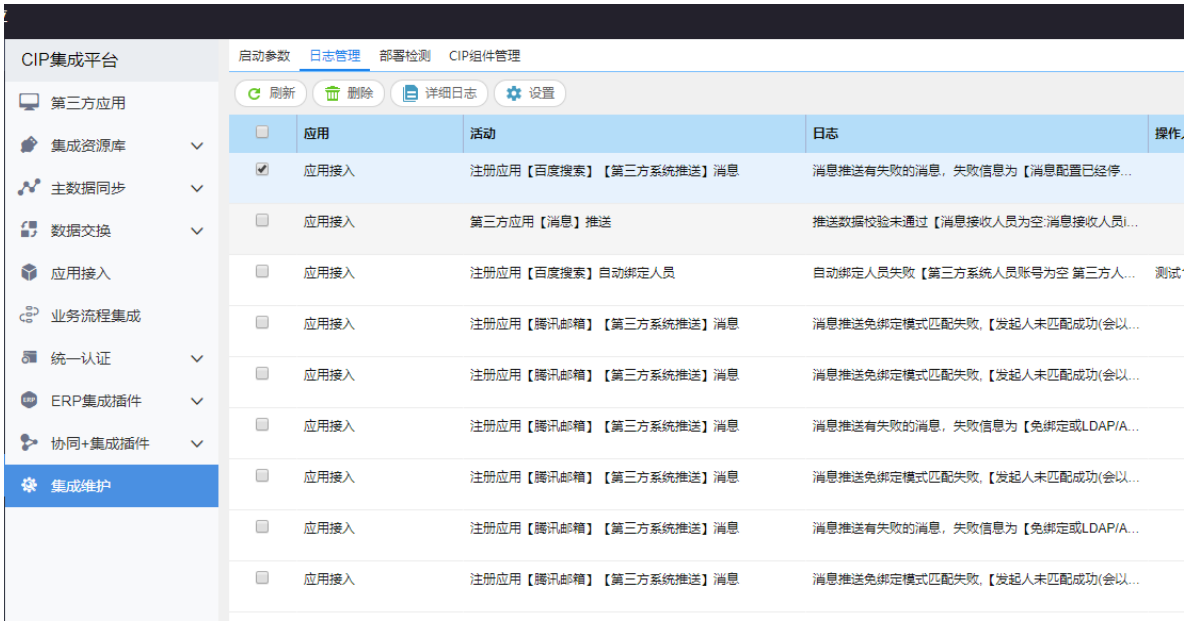
以上就是CIP接口在超级节点中的运用。

注：由于产品的发起流程表单接口的返回值并不符合接口规范，成功返回的是文本（long流程id），失败有时候返回的是json，有时候是500，一般来说这种情况都是不允许的，应该有特定的返回数据格式，所有这边配置成功和失败的条件在现有的CIP中难以控制，如果接口较为标准则不会有这种情况。

八、日志说明

8.1 V7.1版本日志

我们可以在集成维护中查看到CIP相关执行记录：



8.2 V8版本日志（待编写）

8.0对于日志有了进一步的增强。

九、说明

文档中使用到的代码可参考CIP应用集成平台.zip以及单点登录脚手架工程，都需要引入A8的jar包。

十、QA

10.1 CIP待办栏目的刷新机制，是否有定时刷新?

V7.1SP1最新版（二月修复包版本），8.0版本的PC和移动端都实现了关闭刷新。

10.2 第三方认证的方式如何实现?

ThirdpartyAuthenticationPortalManager

10.3 待办的接口读取是否只能配置到CIP中?

也可以实现类，如果没有配置url或者接口，就会找实现类，不推荐。

10.4 CIP和DEE的超级节点暂存待办?

cip和dee的暂存待办暂时不支持暂存待办。

附:

1. CIP全称为Collaboration Integration Platform，即协同集成平台。↩
2. 可以采用客开标准插件，联系对应区域进行 ctp-studio授权↩↩
3. 在客开培训云盘上有更多的详细资料。↩
4. 可以参考官方的rest说明，所有接口调用前都需要先获取token，或者参考客开培训的rest说明文档↩↩↩
5. 插件开发请参考官方的插件开发文档或者查看客开资料中的插件开发说明文档↩
6. 使用主动拉取的模式都会存在一定的性能问题，所以不建议采用。↩↩
7. 本文档主要介绍cip，对扩展类仅做简单介绍，要详细了解可以参考流程事件相关资料或者视频。↩
8. 可以参考之前发布的dee开发说明文档。↩