Token认证使用步骤

一、ECOLOGY系统配置

1、配置接口白名单

在ecology系统代码目录中找到以下配置文件：`ecology/WEB-INF/prop/weaver_session_filter.properties`

```
checkurl=/api/hrm/emmanager;/api/userPhrase;
uncheckurl=/api/ec/dev/app/getCheckSystemInfo;/api/ec/dev/app/emjoin;
uncheckssessionurl=/api/ec/dev/util/accesspage;.../api/loginportal/element/;/api/edc/fillin/;
```

在`uncheckssessionurl=`后面添加`/api/ec/dev/auth/regist;/api/ec/dev/auth/applytoken;`

2、发放许可证(appid)

在ecology系统数据库执行以下脚本示例（*请不要直接使用示例中的数据*）：

```
INSERT INTO ECOLOGY_BIZ_EC(ID,APPID,NAME) VALUES('123456','EEAA5436-7577-4BE0-8C6C-89E9D88805EA','上海泛微网络科技股份有限公司');
COMMIT;
```

字段描述：

- ID：数据库主键。保证与其它系统发放的许可证在数据库中的主键标识不冲突即可（`对应示例：123456`）
- APPID：许可证号码。最终发放给异构系统的许可证号码,多个许可证号码保证唯一（`对应示例：EEAA5436-7577-4BE0-8C6C-89E9D88805EA`）
- NAME：许可证名称。用于快速辨识许可证发放系统（`对应示例：上海泛微网络科技股份有限公司`）

二、异构系统编码实现认证

1、引入DES加密算法工具类(KB190601及以上版本已废弃此步骤)

```
/*
 *
 * Copyright (c) 2001-2018 泛微软件.
 * 泛微协同商务系统,版权所有.
 *
 */``
package com.weaver.test;

import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
```

```java

/**
 * Util_Security 简化版本的认证管理器
 * @version 1.0
 * @author AndyZhang
 */
public class Util_Security {
   /**
     * 混淆参数(ecology中默认混淆参数,不建议修改。)
     * 如需修改混淆参数，一定要保证ecology系统和异构系统混淆参数一致，联系技术开发人员进行修改。
     * @author AndyZhang 云商店-章称
     */
  private static String password = "9588028820109132570743325311898426347857298773549468758875018579537757772 1630"+

"8447887369944730603446620061641196057412243405946910023589270273686087290 1247123456";

    /**
     * DES解密算法
     * @param data 需要解密的密文数据
     * @return 解密完成的明文串，如果为null代表解密失败
     */
    public static String decoded_des(String data) {
        try{
            SecureRandom random = new SecureRandom();
            DESKeySpec desKey = new DESKeySpec(password.getBytes());
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey securekey = keyFactory.generateSecret(desKey);
            Cipher cipher = Cipher.getInstance("DES");
            cipher.init(Cipher.DECRYPT_MODE, securekey, random);
            return new String(cipher.doFinal(Util_Security.hexToByteArray(data)));
        }catch(Exception e){
            e.printStackTrace();
        }
        return null;
    }

    /**
     * DES加密算法 加密函数
     * @param data 需要加密的明文数据
     * @return 加密完成的密文串，如果为null代表加密失败
     */
    public static String encoded_des(String data) {
      try{
      SecureRandom random = new SecureRandom();
      DESKeySpec desKey = new DESKeySpec(password.getBytes());
      SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
      SecretKey securekey = keyFactory.generateSecret(desKey);
      Cipher cipher = Cipher.getInstance("DES");
      cipher.init(Cipher.ENCRYPT_MODE, securekey, random);
      return Util_Security.bytesToHex(cipher.doFinal(data.getBytes()));
    }catch(Throwable e){
      e.printStackTrace();
```

```
    }
    return null;
   }
  /**
   * byte转16进制字符
   * @param bytes
   * @return
   */
    public static String bytesToHex(byte[] bytes) {
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(bytes[i] & 0xFF);
            if(hex.length() < 2){
                sb.append(0);
            }
            sb.append(hex);
        }
        return sb.toString();
    }

    /**
     * 单个16进制字符转byte
     * @param inHex
     * @return
     */
    public static byte hexToByte(String inHex){
      return (byte)Integer.parseInt(inHex,16);
    }

    /**
     * 16进制字符串转byte数组
     * @param inHex
     * @return
     */
    public static byte[] hexToByteArray(String inHex){
        int hexlen = inHex.length();
        byte[] result;
        if (hexlen % 2 == 1){

            hexlen++;
            result = new byte[(hexlen/2)];
            inHex="0"+inHex;
        }else {

            result = new byte[(hexlen/2)];
        }
        int j=0;
        for (int i = 0; i < hexlen; i+=2){
            result[j]=hexToByte(inHex.substring(i,i+2));
            j++;
        }
        return result;
    }

}
```

2、引入RSA加密算法工具jar包到异构系统

在ecology系统代码目录中找到`ecology/WEB-INF/lib/RSA-0.0.1-SNAPSHOT.jar`文件引入到异构系统项目资源目录下。

3、向ECOLOGY系统注册许可证

```
public static void regist() throws Exception {
    //httpclient的使用请自己封装，可参考ECOLOGY中HttpManager类
    HttpManager http = new HttpManager();
    //请求头信息封装集合
    Map<String, String> heads = new HashMap<String, String>();
    //获取当前异构系统RSA加密的公钥
    String cpk = new RSA().getRSA_PUB();
    //当前异构系统用于向ECOLOGY注册时使用的账号密码通过DES加密后密文进行传输
      //kb1906及以上版本 已废弃账号密码校验
    String password = Util_Security.encoded_des("1");
    //封装参数到请求头
    heads.put("appid", "EEAA5436-7577-4BE0-8C6C-89E9D88805EA");
    heads.put("cpk", cpk);
        //kb1906及以上版本 已废弃账号密码校验
    heads.put("loginid", "sysadmin");
        //kb1906及以上版本 已废弃账号密码校验
    heads.put("pwd", password);
    //调用ECOLOGY系统接口进行注册
    String data = http.postDataSSL("http://ip:port/api/ec/dev/auth/regist",
null, heads);
    //返回的数据格式为json，具体格式参数格式请参考文末API介绍。
    //注意此时如果注册成功会返回秘钥信息,请根据业务需要进行保存。
    System.out.println(data);
    /**********************其它业务逻辑实现**********************/
  }
```

4、获取Token

```
public static void applytoken() throws Exception {
    //httpclient的使用请自己封装，可参考ECOLOGY中HttpManager类
    HttpManager http = new HttpManager();
    //请求头信息封装集合
    Map<String, String> heads = new HashMap<String, String>();
    //ECOLOGY返回的系统公钥
    String spk="注册接口成功时返回的apk参数值";
    RSA rsa = new RSA();
    //对秘钥进行加密传输，防止篡改数据
    String secret = rsa.encrypt(null, "调用注册接口返回的secret参数值", null,
"utf-8", spk, false);
    //封装参数到请求头
    heads.put("appid", "EEAA5436-7577-4BE0-8C6C-89E9D88805EA");
    heads.put("secret", secret);
    //调用ECOLOGY系统接口进行申请
    String data =
http.postDataSSL("http://ip:port/api/ec/dev/auth/applytoken", null, heads);
    //返回的数据格式为json，具体格式参数格式请参考文末API介绍。
    //注意此时如果申请成功会返回token,请根据业务需要进行保存。
    System.out.println(data);
    /**********************其它业务逻辑实现**********************/
  }
```

5、接口调用

- 普通接口调用

```
  public static void testRestful() throws Exception{
    //httpclient的使用请自己封装，可参考ECOLOGY中HttpManager类
    HttpManager http=new HttpManager();
    //请求头信息封装集合
    Map<String, String> heads=new HashMap<String, String>();
    //接口参数信息封装集合
    Map<String, Object> param=new HashMap<String, Object>();
    //ECOLOGY返回的系统公钥
    String spk="注册接口成功时返回的apk参数值";
    RSA rsa = new RSA();
    //对用户真实系统id进行加密传输，防止篡改数据
    String userid = rsa.encrypt(null, "真实的ECOLOGY用户ID", null, "utf-8", spk, false);
    //封装参数到请求头
    heads.put("token", "申请token接口成功时返回token值");
    heads.put("appid", "5583bc0e-220e-4a44-8e14-d838d47ad9b7");
    heads.put("userid", userid);
    //......
    //封装接口请求参数
    //......
    //调用接口相关系统接口
    String data=http.postDataSSL("http://127.0.0.1:9990/api/普通接口地址", param,heads);
    //返回接口响应参数
    System.out.println(data);
    /**********************其它业务逻辑实现************************/
  }
```

- "白名单"接口调用(KB190901版本及以上)

> "白名单"接口是指系统存在某些未直接配置接口白名单，且接口内部无需校验用户的普通REST接口，此时无需userid的参数。例如：获取系统信息，上传文件，同步数据，调整配置，检查状态等等接口。

```
  public static void testRestful() throws Exception{
    //httpclient的使用请自己封装，可参考ECOLOGY中HttpManager类
    HttpManager http=new HttpManager();
    //请求头信息封装集合
    Map<String, String> heads=new HashMap<String, String>();
    //接口参数信息封装集合
    Map<String, Object> param=new HashMap<String, Object>();
    //封装参数到请求头
    heads.put("token", "申请token接口成功时返回token值");
    heads.put("appid", "5583bc0e-220e-4a44-8e14-d838d47ad9b7");
    //该参数于190901版本增加
    heads.put("skipsession", "1");
    //......
    //封装接口请求参数
    //......
    //调用接口相关系统接口
```

```
    String data=http.postDataSSL("http://127.0.0.1:9990/api/白名单接口地址", param,heads);
    //返回接口响应参数
    System.out.println(data);
    /**********************其它业务逻辑实现**********************/
  }
```

三、API文档在线说明

1、简要描述：向OA系统发送许可证信息进行注册认证

- 许可证注册接口

请求URL：

- `http://xx.com/api/ec/dev/auth/regist`

请求方式：

- POST

请求头部参数：

参数名	必选	类型	说明
appid	是	string	许可证号码
loginid	是	string	OA系统账号登录名(KB1906版本及以上已废除)
pwd	是	string	账号密码加密后的密文(KB1906版本及以上已废除)
cpk	是	string	异构系统公钥

接口参数：无

返回示例

```
{
    "msg": "ok",
    "errcode": "0",
    "code": 0,
    "msgShowType": "none",
    "secrit": "e0ab4a73-e9c6-4ae5-9dac-f21a7807991a",
    "errmsg": "ok",
    "status": true,
    "spk":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmNIfHNjAvtvkSKtAj79mkHOk5/bDwrGVykRJ7saI1oLHPp8lEQjBP5Ae4Te5chzflLLEj4WP24JnY2ahzb8xnPU/HNt/lwhykjoxVPoczdlXL4jcdMuvgitgCQdrqPgtpzrKNOoKbTz9vagMl0LbzGlSzGPF4PAlGDw+OrgOMsiWXyG1EsQWOW5zfwnnGfvnPZlpSKfDKT3m+9oPL/cWQv6FU1A1kHO4ADT+Np8Q2n33mMJC1Sc+YiFjNIblGoBOAME7MOhb+Ln4b+n+23jsBBeZhfBdmRdLY31AALWIuJH1+8KZvaaMSOOwkBdCfZhF4chWYSRAV4yjCx19Ax/wBQIDAQAB"
}
```

返回参数说明

参数名	类型	说明
status	boolean	响应状态。true:成功,false:失败
code	String	响应码。0代表成功
errcode	String	错误码。0代表成功（可忽略）
msg	String	响应信息
msgShowType	String	信息显示类型。默认“none”
secrit	String	秘钥信息。注意此处secrit单词拼写错误（原词为：secret），请使用 secrit获取结果
spk	String	系统公钥信息

返回错误示例一

```
{
    "msg": "ok",
    "errcode": "1",
    "code": -1,
    "msgShowType": "none",
    "errmsg": "passowrd error:null & sysadmin",
    "status": false
}
```

返回错误示例二

```
{
    "msg": "ok",
    "errcode": "1",
    "code": 0,
    "msgShowType": "none",
    "errmsg": "注册失败没有在找到正确的APPID:EEAA5436-7577-4BE0-8C6C-89E9D888",
    "status": false
}
```

2、简要描述：向OA系统发送获取token请求

- 获取token接口

请求URL：

- `http://xx.com/api/ec/dev/auth/applytoken`

请求方式：

- POST

请求头部参数：

参数名	必选	类型	说明
appid	是	string	许可证号码
secret	是	string	注册许可证时返回的公钥spk对秘钥信息secrit进行加密。（注意此处参数名单词拼写正确）

接口参数：无

返回示例

```
{
    "msg": "获取成功!",
    "code": 0,
    "msgShowType": "none",
    "status": true,
    "token": "770a10d8-0b15-492b-a614-b2c537b512e5"
}
```

返回参数说明

参数名	类型	说明
status	boolean	响应状态。true:成功,false:失败
code	String	响应码。0代表成功
msg	String	响应信息
msgShowType	String	信息显示类型。默认“none”
token	String	认证通过的token信息。（默认30分钟内有效）

返回错误示例一

```
{
    "msg": "认证信息错误！",
    "code": -1,
    "msgShowType": "none",
    "status": false
}
```

返回错误示例二

```
{
    "msg": "解密失败！",
    "code": -1,
    "msgShowType": "none",
    "status": false
}
```

3、简要描述：通过token的方式认证调用ECOLOGY系统普通接口

- token使用演示接口,该API没有实际意义

请求URL：

- `http://xx.com/api/接口地址`

请求方式：

- GET | POST | PUT | DELETE

请求头部参数：

参数名	必选	类型	说明
token	是	string	token信息
appid	是	string	许可证号码
userid	否	string	通过注册许可时返回spk公钥对userid进行加密生成的密文
skipsession	否	string	是否跳过session拦截(用于白名单接口)，1代表是，0代表否（缺省）

接口参数：按照接口规范编写

返回示例

```
{
    "msg": "ok",
    "code": 0,
    "msgShowType": "none",
    "status": true
}
```

返回参数说明

参数名	类型	说明
status	boolean	响应状态。true:成功,false:失败
code	String	响应码。0代表成功
msg	String	响应信息
msgShowType	String	信息显示类型。默认“none”

返回错误示例一

```
{
    "msg": "token:不存在或者超时a25ed7ba-6027-424d-96ec-99daef56a51d",
    "code": -1,
    "msgShowType": "none",
    "status": false
}
```