检验检测模块 IWork电子用印后回调

简介:检测报告在用印环节会调用iwork接口发起用印流程。iwork在用印完毕后，调用接口返回结果。

基础地址

测试系统地址：http://172.16.46.63:30081/admin-api/databus/api/portal

生产系统地址：待定

版本:V1

1.总体说明

先调用【上传附件】接口，上传盖章后的文件。然后调用【用印回调】接口，返回用印结果。接口调用时需要进行签名和加密，详见【接口签名】章节。

2.上传附件接口

接口地址: `/qms_uploadAtt/v1`

请求方式: `POST`

请求数据类型: `application/json`

请求参数:

参数名称	参数说明	数据类型	是否必须
base64Content	文件内容	string	是
fileName	文件名称	string	是
directory	文件目录	long	否
encrypt	是否加密（默认false）	bool	否

响应参数:

参数名称	参数说明	类型
code	返回代码	integer(int32)
msg	返回处理消息	string
data	返回数据对象	object
id	id	string

参数名称	参数说明	类型
path	文件存储路径	string
name	文件名	string
url	文件访问url	string
previewUrl	文件预览url	string
isEncrypted	是否加密	bool
type	类型	string
size	大小	long
createTime	创建时间	时间戳

3.用印回调接口

接口地址: `/qms_sealReply/v1`

请求方式: `POST`

请求数据类型: `application/json`

请求参数:

参数名称	参数说明	请求类型	是否必须
mainId	报告id	string	是
fileIds	附件id，多值用半角逗号分隔	string	是

响应参数:

参数名称	参数说明	类型
code	错误码：0-成功，其他值-失败	integer(int32)
msg	返回处理消息	string

4.接口签名

接口签名和加密参考以下代码：

APP_ID和APP_SECRET请联系相关人员获取

```
public class ApiForIworkExample {
    public static final String TIMESTAMP = Long.toString(System.currentTimeMillis());
    private static final String APP_ID = "";
    private static final String APP_SECRET = "";
```

```
    private static final String ENCRYPTION_TYPE = CryptoSignatureUtils.ENCRYPT_TYPE_AES;
    private static final String TARGET_API = "http://172.16.46.63:30081/admin-api/databus/api/portal/";
    private static final HttpClient HTTP_CLIENT = HttpClient.newBuilder().connectTimeout(Duration.ofSeconds(5)).build();
    private static final ObjectMapper OBJECT_MAPPER = new ObjectMapper();
    private static final PrintStream OUT = buildConsolePrintStream();
    public static final String ZT_APP_ID = "ZT-App-Id";
    public static final String ZT_TIMESTAMP = "ZT-Timestamp";
    public static final String ZT_NONCE = "ZT-Nonce";
    public static final String ZT_SIGNATURE = "ZT-Signature";
    public static final String ZT_AUTH_TOKEN = "ZT-Auth-Token";
    public static final String CONTENT_TYPE = "Content-Type";

    private ApiForIworkExample() {}

    public static void main(String[] args) throws Exception {
        executePostExample();
    }

    private static void executePostExample() throws Exception {
        Map<String, Object> queryParams = new LinkedHashMap<>();
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("mainId", "1983446576685900000");
        jsonObject.put("fileIds", "1983446576685900001,1983446576685900002");
        String bodyJson = jsonObject.toJSONString();
        Map<String, Object> bodyParams = parseBodyJson(bodyJson);
        String signature = generateSignature(queryParams, bodyParams);
        String url = TARGET_API + "qms_sealReply/v1";
        URI requestUri = buildUri(url, queryParams);
        String nonce = randomNonce();
        String cipherBody = encryptPayload(bodyJson);
        OUT.println("原始 Request Body: " + bodyJson);
        OUT.println("加密 Request Body: " + cipherBody);
        HttpRequest request = HttpRequest.newBuilder(requestUri)
                .timeout(Duration.ofSeconds(10))
                .header(ZT_APP_ID, APP_ID)
                .header(ZT_TIMESTAMP, TIMESTAMP)
                .header(ZT_NONCE, nonce)
                .header(ZT_SIGNATURE, signature)
//                .header(ZT_AUTH_TOKEN, "82e5c281ddfa4386988fa4074e8794d7")
                .header(CONTENT_TYPE, "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(cipherBody, StandardCharsets.UTF_8))
                .build();
        HttpResponse<String> response = HTTP_CLIENT.send(request, HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF
        printResponse(response);
    }

    private static String encryptPayload(String plaintext) {
        try {
            return CryptoSignatureUtils.encrypt(plaintext, APP_SECRET, ENCRYPTION_TYPE);
        } catch (Exception ex) {
            throw new IllegalStateException("Failed to encrypt request body", ex);
        }
    }

    private static void printResponse(HttpResponse<String> response) {
        OUT.println("HTTP Status: " + response.statusCode());
        String cipherText = response.body();
        OUT.println("加密 Response: " + cipherText);
        String plain = tryDecrypt(cipherText);
        OUT.println("原始 Response: " + normalizePotentialMojibake(plain));
    }

    private static String randomNonce() {
        return UUID.randomUUID().toString().replace("-", "");
    }
```

```
private static URI buildUri(String baseUrl, Map<String, Object> queryParams) {
    if (queryParams == null || queryParams.isEmpty()) {
        return URI.create(baseUrl);
    }
    StringBuilder builder = new StringBuilder(baseUrl);
    builder.append(baseUrl.contains("?") ? '&' : '?');
    boolean first = true;
    for (Map.Entry<String, Object> entry : queryParams.entrySet()) {
        if (!first) {
            builder.append('&');
        }
        first = false;
        builder.append(URLEncoder.encode(entry.getKey(), StandardCharsets.UTF_8));
        builder.append('=');
        builder.append(URLEncoder.encode(String.valueOf(entry.getValue()), StandardCharsets.UTF_8));
    }
    return URI.create(builder.toString());
}

private static String generateSignature(Map<String, Object> queryParams, Map<String, Object> bodyParams) {
    TreeMap<String, Object> sorted = new TreeMap<>();
    if (queryParams != null) {
        queryParams.forEach((key, value) -> sorted.put(key, normalizeValue(value)));
    }
    if (bodyParams != null) {
        bodyParams.forEach((key, value) -> sorted.put(key, normalizeValue(value)));
    }
    sorted.put(ZT_APP_ID, APP_ID);
    sorted.put(ZT_TIMESTAMP, TIMESTAMP);
    StringBuilder canonical = new StringBuilder();
    sorted.forEach((key, value) -> {
        if (value == null) {
            return;
        }
        if (canonical.length() > 0) {
            canonical.append('&');
        }
        canonical.append(key).append('=').append(value);
    });
    OUT.println("原始 签名串: " + canonical);
    String md5Hex = md5Hex(canonical.toString());
    OUT.println("原始签名: " + md5Hex);
    return md5Hex;
}

private static Object normalizeValue(Object value) {
    if (value == null) {
        return null;
    }
    if (value instanceof Map || value instanceof Iterable) {
        try {
            return OBJECT_MAPPER.writeValueAsString(value);
        } catch (JsonProcessingException ignored) {
            return value.toString();
        }
    }
    return value;
}

private static Map<String, Object> parseBodyJson(String bodyJson) {
    if (bodyJson == null || bodyJson.isBlank()) {
        return Map.of();
    }
    try {
        return OBJECT_MAPPER.readValue(bodyJson, new TypeReference<Map<String, Object>>() { });
    } catch (IOException ex) {
```

```
            throw new IllegalArgumentException("Failed to parse request body JSON", ex);
        }
    }

    private static String md5Hex(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            byte[] bytes = digest.digest(input.getBytes(StandardCharsets.UTF_8));
            StringBuilder hex = new StringBuilder(bytes.length * 2);
            for (byte b : bytes) {
                String segment = Integer.toHexString(b & 0xFF);
                if (segment.length() == 1) {
                    hex.append('0');
                }
                hex.append(segment);
            }
            return hex.toString();
        } catch (NoSuchAlgorithmException ex) {
            throw new IllegalStateException("MD5 algorithm not available", ex);
        }
    }

    private static String tryDecrypt(String cipherText) {
        if (cipherText == null || cipherText.isBlank()) {
            return cipherText;
        }
        try {
            // Databus 会在凭证开启加密时返回密文，这里做一次解密展示真实响应。
            return CryptoSignatureUtils.decrypt(cipherText, APP_SECRET, ENCRYPTION_TYPE);
        } catch (Exception ex) {
            return "<unable to decrypt> " + ex.getMessage();
        }
    }

    // 解决控制台打印 乱码问题
    private static String normalizePotentialMojibake(String value) {
        if (value == null || value.isEmpty()) {
            return value;
        }
        long suspectCount = value.chars().filter(ch -> ch >= 0x80 && ch <= 0xFF).count();
        long highCount = value.chars().filter(ch -> ch > 0xFF).count();
        if (suspectCount > 0 && highCount == 0) {
            try {
                byte[] decoded = value.getBytes(StandardCharsets.ISO_8859_1);
                String converted = new String(decoded, StandardCharsets.UTF_8);
                if (converted.chars().anyMatch(ch -> ch > 0xFF)) {
                    return converted;
                }
            } catch (Exception ignored) {
                return value;
            }
        }
        return value;
    }

    /**
     * 输出流编码与当前控制台保持一致，避免中文字符再次出现编码差异。
     */
    private static PrintStream buildConsolePrintStream() {
        try {
            String consoleEncoding = System.getProperty("sun.stdout.encoding");
            if (consoleEncoding != null && !consoleEncoding.isBlank()) {
                return new PrintStream(System.out, true, Charset.forName(consoleEncoding));
            }
            return new PrintStream(System.out, true, Charset.defaultCharset());
        } catch (Exception ignored) {
```

```
            return System.out;
        }
    }
}
```