视频点播

# 服务端 SDK 参考

| | |
|---|---|
| 文档版本 | 01 |
| 发布日期 | 2026-03-09 |

华为云计算技术有限公司

### 商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

### 注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## 华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

# 目 录



# 1 SDK 概述

## VOD SDK 概述

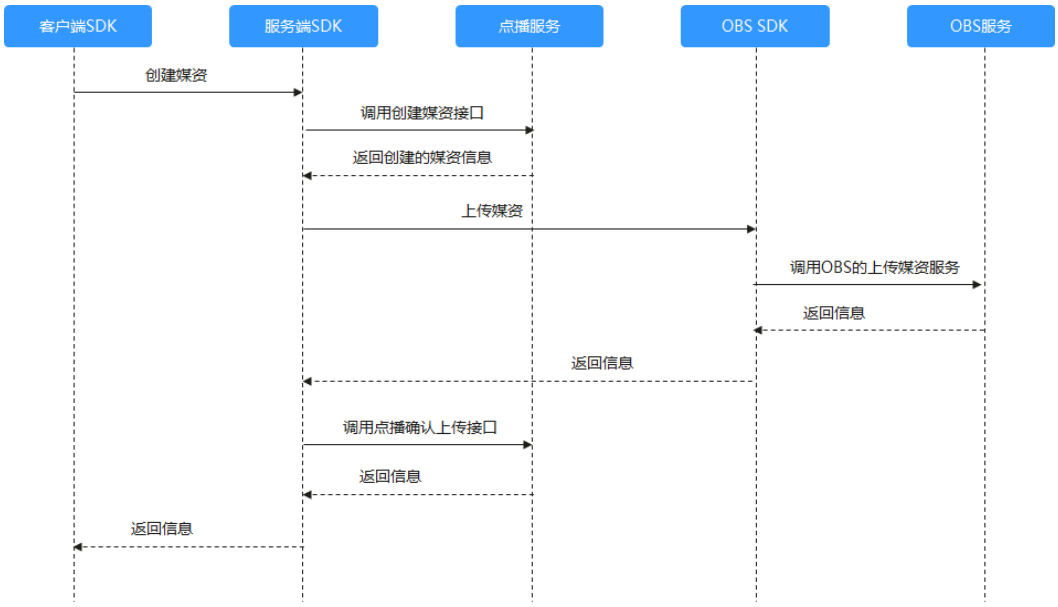
VOD SDK是对点播服务接口请求的封装，请您在使用SDK前务必先查看点播服务的接口文档，了解相关接口的功能、参数、规则和使用方法。

原SDK 2.x版本已全面下线，现推出全新设计的V3版本SDK。最新版本的SDK支持**Java**、**Python**、**Go**、**NodeJs**、**.Net**、**PHP**和**C++**开发语言，您可以登录**SDK中心**下载对应开发语言的SDK。

## 集成开发流程

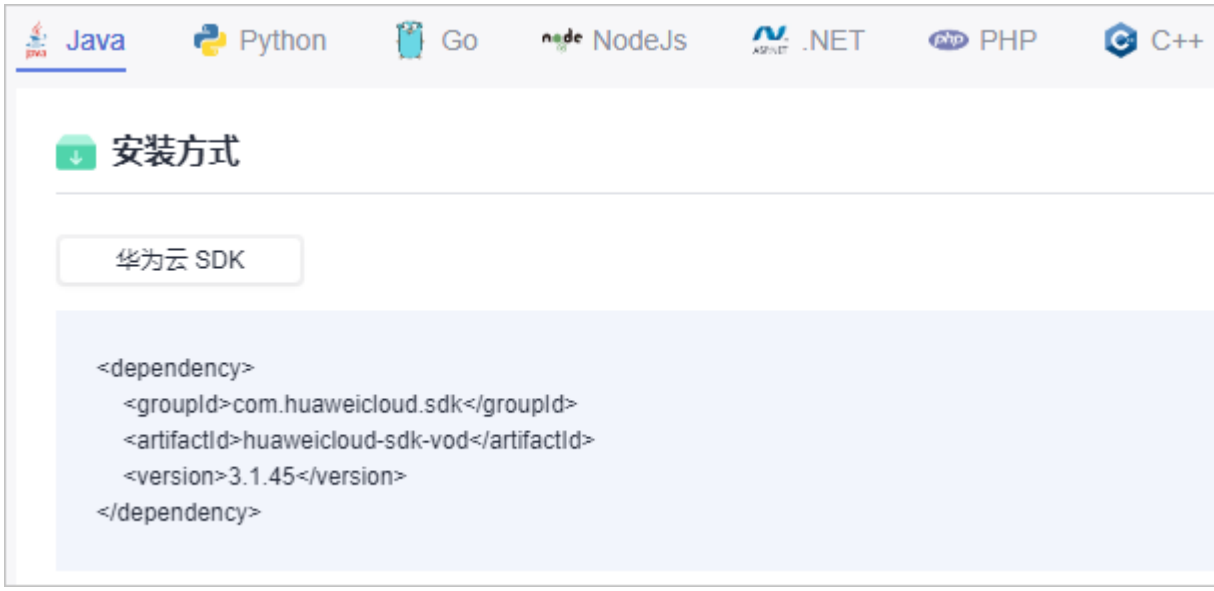

## SDK 工作流程

除点播服务SDK外，服务端SDK与其它SDK的交互关系如下：



# 2 SDK 下载

华为云视频点播服务提供了**Java**、**Python**、**Go**、**NodeJs**、**.Net**、**PHP**和**C++**语言版本的服务端SDK，您可以在**SDK中心**下载如下对应语言的SDK。



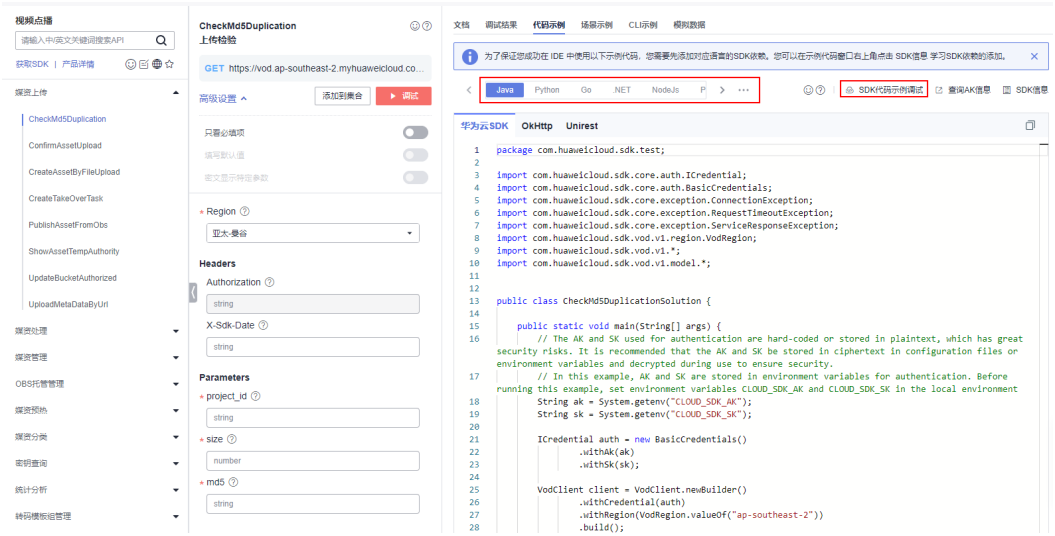
# 3 SDK 开发说明

V3版本SDK对点播服务提供的所有API进行了封装，您可以通过点播**API Explorer**调试接口。同时在代码示例处，会生成各种语言的demo供您参考。您也可以使用“SDK代码示例调试”，直接调试代码。

**说明**

在使用SDK V3版本过程中，如您有进一步疑问和建议，欢迎您**提交工单**进行交流反馈。

**图 3-1** API Explorer




## 媒资上传

当您需要使用服务端SDK上传本地媒资时，可参考《视频点播 API参考》中的**应用示例1**或**应用示例2**进行操作。其中，应用示例中对应的“创建媒资：上传方式”和“确认媒资上传”步骤，可参考SDK中的**创建媒资：上传方式**和**确认媒资上传**方法。在PUT媒资文件时，您可以使用**HTTP PUT**方式，将媒资文件PUT到对应的URL中即可。

## 生成鉴权 URL

若您使用SDK时需要生成鉴权URL，可在点播控制台中使用**Key防盗链**算法生成，也可以参考以下Demo生成。

```
package AuthUrlDemo;
import org.apache.commons.codec.binary.Hex;
import org.apache.commons.codec.digest.DigestUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

class AuthUrlDemo {
    // url是未带加密信息的原始播放url, key是在点播控制台配置的Key值
    // 加密算法A
    public static String createAuthInfoUrlByAlgorithmA(String url, String key, String method) {
        try {
            checkParam(url, key);

            long timestamp = Instant.now().getEpochSecond();
            String randUid = UUID.randomUUID().toString().replaceAll("-", "");
            String uid = "0";
            String tmpRandKey = timestamp + "-" + randUid + "-" + uid;

            URL originUrl = new URL(url);
            String originHashStr = originUrl.getPath() + "-" + tmpRandKey + "-" + key;
            String hashStr = hashString(originHashStr, method);
            String authInfo = "auth_key=" + tmpRandKey + "-" + hashStr;

            return StringUtils.isEmpty(originUrl.getQuery()) ? url + "?" + authInfo : url + "&" + authInfo;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    // 加密算法B
    public static String createAuthInfoUrlByAlgorithmB(String url, String key, String method) {
        try {
            checkParam(url, key);

            URL originUrl = new URL(url);
            String filePath = originUrl.getPath();
            String dateStr = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMddHHmm"));
            String originHashStr = key + dateStr + filePath;
            String hashStr = hashString(originHashStr, method);

            return originUrl.getProtocol() + "://" + originUrl.getHost() + "/"
                + dateStr + "/" + hashStr + originUrl.getFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    // 加密算法C
    public static String createAuthInfoUrlByAlgorithmC(String url, String key, String method) {
        try {
            checkParam(url, key);

            URL originUrl = new URL(url);
```

```
            String filePath = originUrl.getPath();
            String hexTime = Long.toHexString(Instant.now().getEpochSecond()).toUpperCase(Locale.ENGLISH);
            String originHashStr = key + filePath + hexTime;
            String hashStr = hashString(originHashStr, method);

            return originUrl.getProtocol() + "://" + originUrl.getHost() + "/" + hashStr + "/" + hexTime +
originUrl.getFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    // 加密算法D
    public static String createAuthInfoUrlByAlgorithmD(String url, String key, String pliveDuration, String
previewDuration) {
        try {
            isDigital("试看时长", previewDuration);
            isDigital("伪直播开始时间", pliveDuration);
            checkParam(url, key, previewDuration, pliveDuration);

            String authArg = "";

            URL originUrl = new URL(url);
            String urlPath = originUrl.getPath();
            String pathInUrl = urlPath.substring(0, urlPath.lastIndexOf("/") + 1);
            String data = encodeUrl(pathInUrl) + "$" + getUtcTime();

            if (previewDuration != null && !previewDuration.isEmpty()) {
                data += "$" + previewDuration;
                authArg = "&exper=" + previewDuration;
            }

            if (pliveDuration != null && !pliveDuration.isEmpty()) {
                data += "$" + pliveDuration;
                authArg = "&plive=" + pliveDuration;
            }
            String encryptInfo = aesCbcEncrypt(data, key, true);

            String authInfoStr = "auth_info=" + URLEncoder.encode(encryptInfo, StandardCharsets.UTF_8);
            if (url.contains("?")) {
                return originUrl + "&" + authInfoStr + authArg;
            }
            return originUrl + "?" + authInfoStr + authArg;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    public static String createAuthInfoUrlByAlgorithmE(String url, String key, String pliveDuration, String
previewDuration) {
        try {
            isDigital("试看时长", previewDuration);
            isDigital("伪直播开始时间", pliveDuration);
            checkParam(url, key, previewDuration, pliveDuration);

            URL originUrl = new URL(url);
            String urlPath = originUrl.getPath();

            String currentTimestamp = String.valueOf(new Date().getTime() / 1000);

            String authArg = "";
            String authStr = key + urlPath + currentTimestamp;

            if (previewDuration != null && !previewDuration.isEmpty()) {
                authArg = "&exper=" + previewDuration;
                authStr = authStr + previewDuration;
            }
```

```
        if (pliveDuration != null && !pliveDuration.isEmpty()) {
            authArg = "&plive=" + pliveDuration + authArg;
            authStr = authStr + pliveDuration;
        }

        authStr = sha256DigestAsHex(authStr);
        authArg = "auth_key=" + authStr + "&timestamp=" + currentTimestamp + authArg;

        return url.contains("?") ? originUrl + "&" + authArg : originUrl + "?" + authArg;

    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

private static void checkParam(String url, String key) {
    if (StringUtils.isAnyEmpty(url, key)) {
        throw new IllegalArgumentException("url or key is illegal");
    }
}

private static void checkParam(String url, String key, String preview, String plive) {
    if (StringUtils.isAnyEmpty(url, key)) {
        throw new IllegalArgumentException("url or key is illegal");
    }

    if (StringUtils.isNoneEmpty(preview, plive)) {
        throw new IllegalArgumentException("preview and plive cannot be enabled all ");
    }
}

private static String aesCbcEncrypt(String data, String key, boolean hasPoint) throws Exception {
    checkParam(data, key);

    byte[] realKey = get128BitKey(key);
    SecureRandom secureRand = new SecureRandom();
    byte[] ivBytes = new byte[16];
    secureRand.nextBytes(ivBytes);

    if (hasPoint) {
        return aesCbcEncrypt(data, ivBytes, realKey) + "." + bytesToHexString(ivBytes);
    } else {
        return aesCbcEncrypt(data, ivBytes, realKey) + bytesToHexString(ivBytes);
    }
}

private static String aesCbcEncrypt(String data, byte[] ivBytes, byte[] key) throws Exception {
    SecretKeySpec sk = new SecretKeySpec(key, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

    if (ivBytes != null) {
        cipher.init(Cipher.ENCRYPT_MODE, sk, new IvParameterSpec(ivBytes));
    } else {
        cipher.init(Cipher.ENCRYPT_MODE, sk);
    }

    return Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes(StandardCharsets.UTF_8)));
}

private static byte[] get128BitKey(String key) {
    byte[] result = null;

    if (key != null) {
        result = new byte[16];
        byte[] origin = key.getBytes();

        System.arraycopy(origin, 0, result, 0, Math.min(origin.length, 16));
    }
```

```
    return result;
}

private static String encodeUrl(String str) {
    try {
        if (StringUtils.isNotEmpty(str)) {
            StringBuilder encodeStr = new StringBuilder(32);
            String[] tmpArray = str.split("/");
            for (String s : tmpArray) {
                encodeStr.append(URLEncoder.encode(s, StandardCharsets.UTF_8)).append("/");
            }
            return encodeStr.toString();
        }
    } catch (Exception e) {
        throw new RuntimeException(String.format("Encode fail %s", e.getMessage()));
    }
    return str;
}

public static String sha256DigestAsHex(String plainText) {
    MessageDigest messageDigest;
    String encodeStr;
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
        byte[] hash = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));
        encodeStr = Hex.encodeHexString(hash);
    } catch (NoSuchAlgorithmException e) {
        throw new IllegalStateException("Could not find MessageDigest with algorithm SHA-256", e);
    }
    return encodeStr;
}

private static String getUtcTime() {
    SimpleDateFormat foo = new SimpleDateFormat("yyyyMMddHHmmss");
    java.util.Calendar cal = java.util.Calendar.getInstance();
    int zoneOffset = cal.get(java.util.Calendar.ZONE_OFFSET);
    int dstOffset = cal.get(java.util.Calendar.DST_OFFSET);
    cal.add(java.util.Calendar.MILLISECOND, -(zoneOffset + dstOffset));

    return foo.format(new Date(cal.getTimeInMillis()));
}

private static String bytesToHexString(byte[] src) {
    StringBuilder stringBuilder = new StringBuilder();
    if (src == null || src.length == 0) {
        return null;
    }
    for (byte b : src) {
        int v = b & 0xFF;
        String hv = Integer.toHexString(v);
        if (hv.length() < 2) {
            stringBuilder.append(0);
        }
        stringBuilder.append(hv);
    }
    return stringBuilder.toString();
}

private static void isDigital(String paramName, String value) throws Exception {
    if (value != null && !value.isEmpty() && !value.matches("\\d+")) {
        throw new Exception(paramName + "仅支持数字");
    }
}

private static String hashString(String oriStr, String algorithm) {
    return algorithm.equals("MD5") ? DigestUtils.md5Hex(oriStr.getBytes(StandardCharsets.UTF_8))
        : sha256DigestAsHex(oriStr);
```

```
    }
}
```