腾讯云小程序平台

代码接入指引

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

代码接入指引
- Demo及SDK获取
- 使用 Android 设备接入
  - 步骤一：获取应用 SDK 配置
  - 步骤二：集成 SDK
  - 步骤三：使用 SDK
    - SDK 初始化
    - 小程序管理
    - 小程序引擎代理实现
    - 扩展 SDK
    - 自定义UI
    - 自定义API
    - 自定义原生组件
    - 自定义分享
    - 开放接口
  - 常见问题
- 使用 iOS 设备接入
  - 步骤一：获取应用SDK 配置
  - 步骤二：集成 SDK
  - 步骤三：使用 SDK
    - SDK 初始化
    - 小程序管理 API
    - 小程序引擎代理类实现
    - 小程序文件管理
    - 扩展 SDK
    - 自定义UI
    - 自定义 API
    - 自定义原生组件
    - 常见问题

代码接入指引
Demo及SDK获取

最近更新时间：2024-02-26 10:34:51

Android

- 在线集成方式 Demo 获取
- 离线集成方式 Demo 及 SDK 获取

iOS

> **说明：**
> 按步骤一获取配置文件，替换后运行，项目 BundleId 需与运营平台设置的 BundleId 保持一致。

- 通过 CocoaPods 集成 Demo 获取地址
- 手动集成 Demo 及 sdk 获取地址

使用 Android 设备接入
步骤一：获取应用 SDK 配置

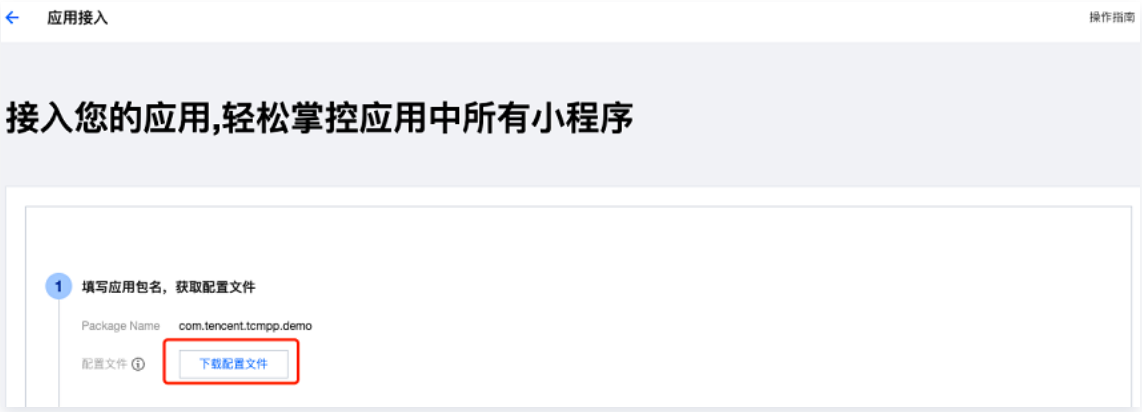
最近更新时间：2024-03-21 15:51:52

使用容器 SDK 需要申请配置信息，只有在 SDK 初始化的时候配置了正确的配置文件，才能初始化成功并正常使用。

新建应用获取配置信息

1. 登录腾讯云小程序平台-运营端 控制台，在左侧导航栏单击**概览**。
2. 在概览页单击**接入应用**，在创建应用的弹窗中，填写如下信息后，单击**下一步，集成小程序容器**，完成应用创建，进入接入向导。
   - 应用名称：仅支持20个字符以内的汉字，字母和数字，不支持特殊符号。
   - 接入端选择 Android 端。
   - 输入 Package Name。

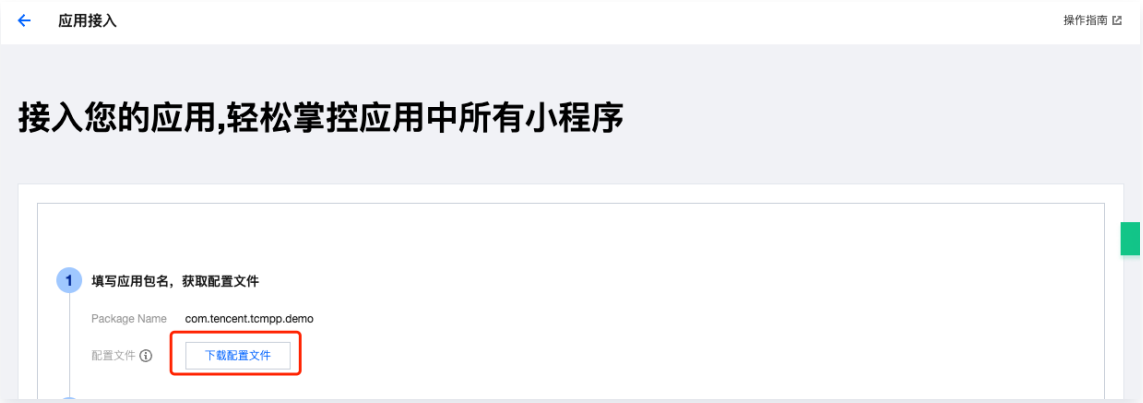
3. 在接入向导页面，单击**下载配置文件**。



从应用主页下载配置文件

1. 登录腾讯云小程序平台-运营端 控制台，在左侧导航栏单击**应用管理** > **应用主页**，进入应用基础信息页面。

2. 单击**查看详情**，进入接入向导页面后单击**下载配置文件**，完成下载。



步骤二：集成 SDK

最近更新时间：2023-12-28 11:32:51

前提条件

> **说明：**
> 搭载 minSdkVersion 21 环境。

集成方式

小程序容器集成

1. **配置仓库地址**

- **方式一：** Gradle 7.0以前版本。

  打开用户项目管理工具，在项目级 build.gradle 中添加如下配置。

```
buildscript {
    dependencies {
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:1.4.32'
    }
}

allprojects {
    repositories {
        maven {
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'
        }
    }
}
```

- **方式二：** Gradle 7.1及之后版本。

  在项目级 settings.gradle 中添加如下配置。

```
pluginManagement {
    repositories {
        maven {
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'
        }
    }
}
```

```
dependencyResolutionManagement {
    repositories {
        maven {
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'
        }
    }
}
```

2. 在应用级 build.gradle 中配置插件。

- **方式一：** Gradle 7.0以前版本：

```
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-kapt'
```

- **方式二：** Gradle 7.1及之后版本：

```
plugins {
    id "org.jetbrains.kotlin.android"
    id "org.jetbrains.kotlin.kapt"
}
```

3. 在应用级 build.gradle 中配置 packagingOptions。

```
android {
    defaultConfig {
        packagingOptions {
            pickFirst 'lib/arm64-v8a/libc++_shared.so'
            pickFirst 'lib/armeabi/libc++_shared.so'
            pickFirst 'lib/armeabi-v7a/libc++_shared.so'
            pickFirst 'lib/arm64-v8a/libmarsxlog.so'
            pickFirst 'lib/armeabi/libmarsxlog.so'
            pickFirst 'lib/armeabi-v7a/libmarsxlog.so'
            pickFirst 'lib/arm64-v8a/libv8jni.so'
        }
    }
}
```

4. 在应用级 build.gradle 中配置依赖项。

```
dependencies {
    implementation 'com.google.android.material:material:1.3.0-alpha03'
    implementation 'androidx.core:core-ktx:1.6.0'
```

```
    //gosn
    implementation 'com.google.code.gson:gson:2.8.6'
    // ok-http
    implementation 'com.squareup.okhttp3:okhttp:3.12.13'

    // mini app start
    kapt
'com.tencent.tcmpp.android:mini_annotation_processor:${version}'//版本
信息请参考SDK 更新动态
    implementation 'com.tencent.tcmpp.android:mini_core:${version}'//版
本信息请参考SDK 更新动态
    // mini app end
}
```

- 请将 ${version} 替换为 SDK 最新版本，版本信息请参考 SDK 更新动态 。
- 如果开发者原来的工程中使用了 annotationProcessor 注解处理器，需要将所有 annotationProcessor 改为 kapt。
- 如需使用x5内核，可参考 扩展 SDK > X5内核 。

5. 集成过程可能会遇到的问题

- 集成过程可能会遇到如下问题：

```
AAPT: error: attribute android:requestLegacyExternalStorage not found.
```

在 application 标签下添加如下代码：

```
<application
   android:theme="@style/AppTheme"
   tools:replace="android:icon"
     tools:remove="android:requestLegacyExternalStorage">
 /application>
```

- 集成过程中可能会出现如下错误：

```
Duplicate class android.support.v4.app.INotificationSideChannel found in
modules core-1.3.1-runtime (androidx.core:core:1.3.1) and support-v4-21.0.3-
runtime (com.android.support:support-v4:21.0.3)
```

在 `gradle.properties` 中添加如下代码：

```
android.useAndroidX=true
android.enableJetifier=true
```

6. 小程序 taro 框架支持

```
Uncaught DOMException: Failed to read the 'sessionStorage' property
from 'Window': Access is denied for this document.
    at <anonymous>:1207:96250
    at Array.forEach (<anonymous>)
    at Module.<anonymous> (<anonymous>:1207:96164)
    at Module.9 (<anonymous>:1207:113782)
    at l (<anonymous>:1203:566)
    at Module.204 (<anonymous>:1215:87806)
    at l (<anonymous>:1203:566)
    at t (<anonymous>:1203:435)
    at Array.r [as push] (<anonymous>:1203:298)
    at <anonymous>:1215:125
```

如果小程序基于 Taro 框架开发可能出现上面的错误，请添加如下依赖（未使用 taro 框架则无需添加）。

```
implementation 'com.tencent.tcmpp.android:mini_extra_v8:${version}'
```

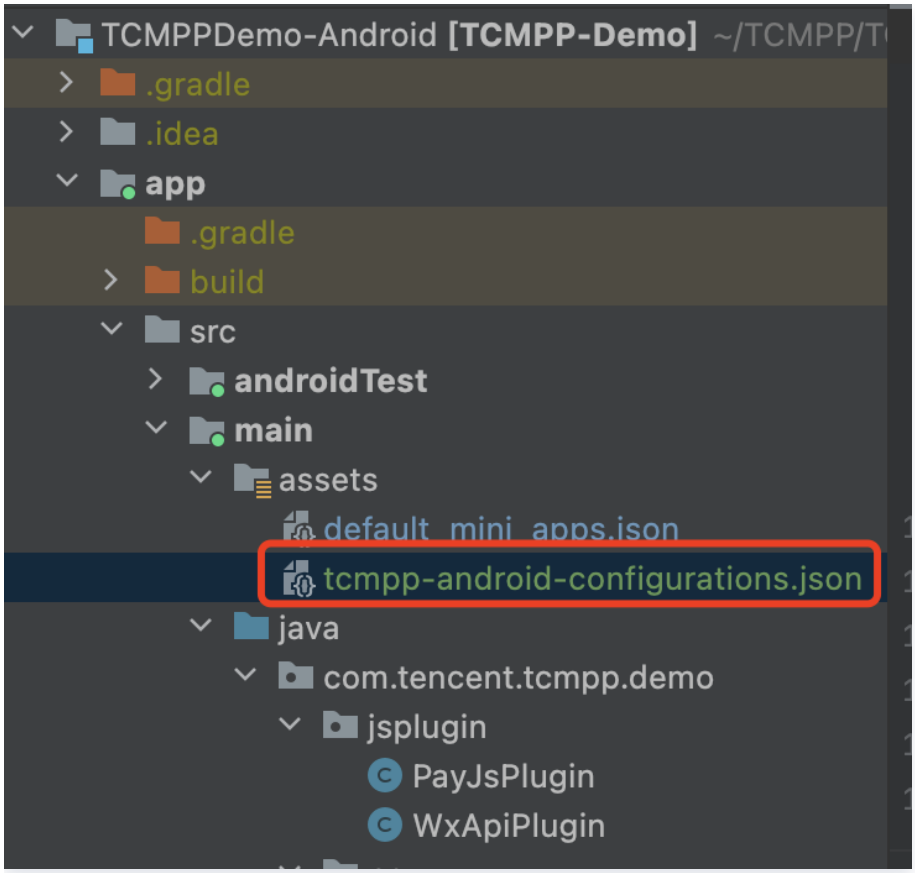
步骤三：使用 SDK

SDK 初始化

最近更新时间：2024-04-10 15:12:31

配置文件获取

开发人员从腾讯云小程序平台获取对应 App 的配置文件，该配置文件是一个 json 文件，包含该 App 使用小程序平台的所有信息，将配置文件引入到项目的 assets 根目录。获取配置文件操作，可参见 获取容器 SDK 凭证 。



注意：

应用包名必须与配置文件中 packageName 保持一致，否则运行时无法通过校验。

如果不一致，不能直接修改配置文件包名，应通过下面两种方式来修正：

- 修改工程中应用包名，与配置文件中包名一致。
- 控制台修改应用包名，与工程中应用包名保持一致，并重新下载配置文件。

```
{
  "customId": "T1682645488JDYWMH",
  "productId": "7422",
  "packageName": "com.tencent.tcmpp.showcase",
  "bundleId": "",
  "material": "01rg5oB0lgkX9bGM7/C8k7RBovDV4ByAGrQmKj7njQ2Jcs
  "shark": {
    "httpUrl": "https://api.tcmpp-staging.tmfcloud.com:443",
    "tcpHost": "api.tcmpp-staging.tmfcloud.com",
    "tcpPort": 30014,
    "appKey": "app-1lwbqz2nop",
    "symEnc": 2,
    "asymEnc": 1
  }
}
```


配置信息设置

- 实现 MiniConfigProxy 类
- 添加如下注解

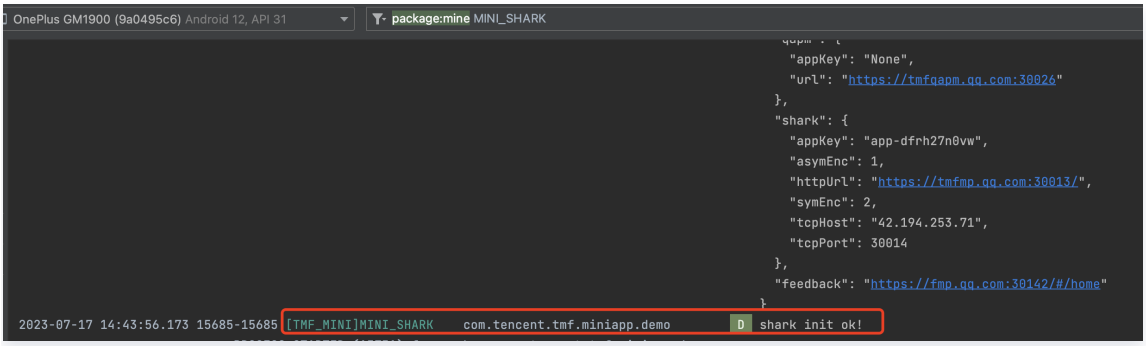
```
@ProxyService(proxy = MiniConfigProxy.class)
```

参考代码：

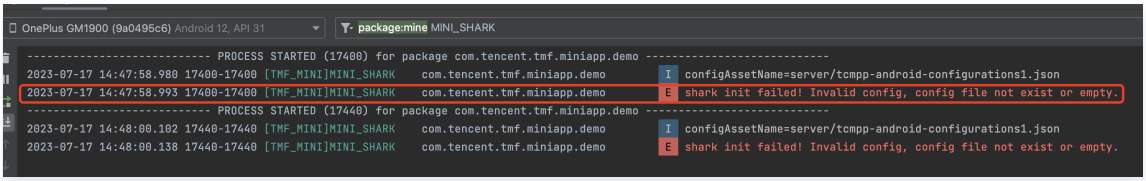
```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 应用Application
     * @return
     */
    @Override
    public Application getApp() {
        return "app Application";
    }

     /**
     * 创建初始化配置信息
     * @return
     */
    @Override
    public MiniInitConfig buildConfig() {
      MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
      MiniInitConfig config = builder
```

```
            .configAssetName("tcmpp-android-configurations.json") //assets中配置文件名称
            .imei("IMEI") //配置设备id， 于在管理平台上根据设备标识进 程序的灰度发布使(可选)
            .autoRequestPermission(true) //配置小程序使用到需要权限的API时是否自动向用户申请对应的系统权限
            .debug(true) //日志开关，默认关闭的
            .build();
    }
}
```

继承并实现 MiniConfigProxy 类后，就可以通过启动小程序接口打开小程序了，sdk 内部会自动进行初始化。从 mini_core 1.5.1.1版本之后，可通过 MINI_SHARK 过滤日志，确定初始化是否成功，初始化成功会输出：shark init ok!



如果初始化失败会输出：shark init failed! 及具体错误原因。



如果初始化失败，可参考如下常见原因进行排查。

工程编译失败排查

- 原因一：出现如下错误

```
> A failure occurred while executing org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction
   > java.lang.reflect.InvocationTargetException (no error message)

* Try:
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Exception is:
org.gradle.api.tasks.TaskExecutionException: Execution failed for task ':libs:libopensdk:kaptEnvTestKotlin'. <33 internal lines>
Caused by: org.gradle.workers.internal.DefaultWorkerExecutor$WorkExecutionException: A failure occurred while executing
 org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction <126 internal lines>
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.gradle.internal.KaptExecution.run(KaptWithoutKotlincTask.kt:285)
    at org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction.execute(KaptWithoutKotlincTask.kt:240)
      <27 internal lines>
    ... 2 more
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing(annotationProcessing.kt:90)
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing$default(annotationProcessing.kt:31)
    at org.jetbrains.kotlin.kapt3.base.Kapt.kapt(Kapt.kt:47)
    ... 34 more
Caused by: com.sun.tools.javac.processing.AnnotationProcessingError Create breakpoint : java.lang.NoClassDefFoundError: 
com/tencent/tmfmini/sdk/annotation/ProxyService
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.callProcessor(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.discoverAndRunProcs(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment$Round.run(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.doProcessing(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.main.JavaCompiler.processAnnotations(Unknown Source)
    ... 40 more
Caused by: java.lang.NoClassDefFoundError: com/tencent/tmfmini/sdk/annotation/ProxyService
    at com.tencent.tmfmini.sdk.annotation.processor.ProxyServiceProcessor.process(ProxyServiceProcessor.java:48)
    at org.jetbrains.kotlin.kapt3.base.incremental.IncrementalProcessor.process(incrementalProcessors.kt:90)
    at org.jetbrains.kotlin.kapt3.base.ProcessorWrapper.process(annotationProcessing.kt:188)
    ... 45 more
Caused by: java.lang.ClassNotFoundException: com.tencent.tmfmini.sdk.annotation.ProxyService
    ... 48 more
```

检查工程中是否有如下配置，如果有则去掉。

```
kapt.include.compile.classpath=false
```

小程序启动失败排查

- 原因一：配置文件路径错误，configAssetName 设置的是 assets 目录下文件的完整路径，如果配置文件在

子目录下需要追加目录路径，例如：server/tcmpp-android-configurations.json。

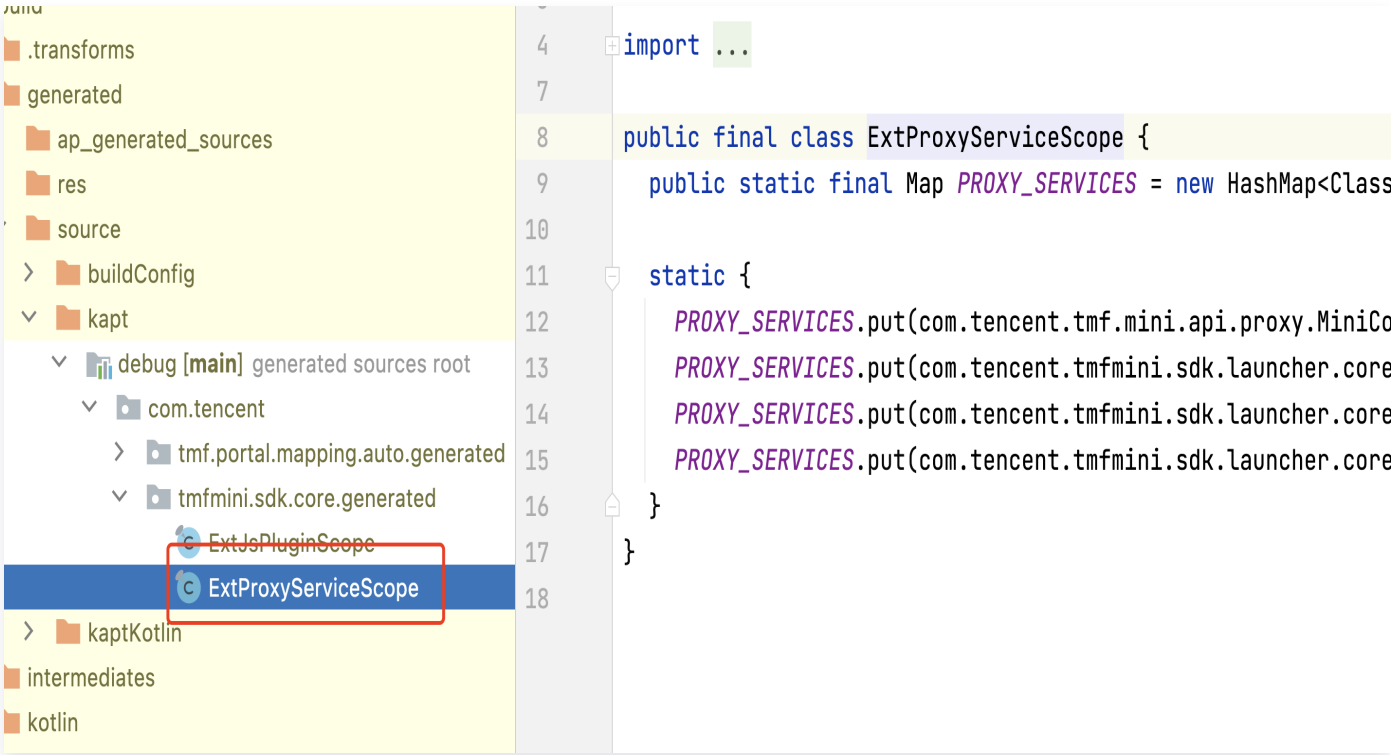
- 原因二：不允许修改小程序配置文件内容，否则小程序无法正常运行。
- 原因三：配置文件中的 **packageName** 必须与应用的 **applicationId** 保持一致，否则 App 运行失败，因为初始化时会校验配置文件中的 **packageName**，可以通过如下设置不进行包名校验。

```
MiniInitConfig config = builder
        .verifyPkg(false)//忽略包名校验
        .build();
```


```
{
  "channel": "Android",
  "customId": "9ab50ac0-be06-11ec-a34e-278d66d394b5",
  "material": "01X7UBa0sAdvWstZOsTT0mdoDnZghwoG145kRt8B3i0vvTj31DVuIuXrmEZFtnnyryyJngGkBm1I4AckFtfAIs2cYkaWRawCnwteX
  "packageName": "com.tencent.tmf.miniapp.demo",
  "productId": "7686",
  "qapm": {
    "appKey": "None",
    "url": "https://tmfqapm.qq.com:30026"
  },
  "shark": {
    "appKey": "app-dfrh27n0vw",
    "asymEnc": 1,
    "httpUrl": "https://tmfmp.qq.com:30013/",
    "symEnc": 2,
    "tcpHost": "42.194.253.71",
    "tcpPort": 30014
  }
}
```

```
android {
    compileSdkVersion 30
    defaultConfig {
        applicationId "com.tencent.tmf.miniapp.demo"
        //feed back 测试
//        applicationId "tmf.tencent.tmf.demo.db"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        ndk { NdkOptions it -> abiFilters "arm64-v8a" }
```

- 原因四：确定上面初始化注解@ProxyService是否生成了如下类ExtProxyServiceScope



隐私合规说明

sdk 的初始化是在用户调用 TmfMiniSDK 类的方法时调用的，所以在用户同意隐私合规授权之后，再调用 TmfMiniSDK 类中提供的方法即可。

其它初始化动作(非必须设置)

设置地区或者账号，方便进行灰度推送时使用。

```
/**
 * 设置账号信息
 * @param userId
 */
 public static void setUserId(String userId)

/**
 * 设置位置信息
 * @param country
 * @param province
 * @param city
 */
 public static void setLocation(String country, String province, String city)
```

X5初始化监听(非必须设置)

如果需要监听X5初始化是否成功，可通过如下方式获取初始化回调。

```
@ProxyService(proxy = IX5EventProxy.class)
public class X5EventProxyImpl implements IX5EventProxy {
    /**
     * X5初始化成功回调
     * @param isX5
     */
    @Override
    public void init(boolean isX5) {

    }
}
```

小程序管理

最近更新时间：2024-05-07 11:31:11

打开小程序

启动选项请参考 MiniStartOptions
启动场景请参考 MiniScene

```
/**
 * 启动小程序
 * @param activity
 * @param appId     小程序Id（从控制台获取）
 * @param options
 */
public static void startMiniApp(Activity activity, String appId, MiniStartOptions options)


/**
 * 启动小程序
 * @param activity
 * @param appId     小程序Id （从控制台获取）
 * @param scene     不同场景下打开小程序设置不同参数,参见MiniScene
 * @param appVerType 小程序的版本类型(预览\开发版本),参见MiniApp
 * @param options
 */
public static void startMiniApp(Activity activity, String appId, int scene, int
appVerType, MiniStartOptions options) {
```

> **说明：**
> 如果有本地缓存版本，则先打开本地版本，同时检查更新并下载，新版本在下次冷启动时生效。

打开正式版小程序

如下方式打开**已上线**最新版小程序。

```
TmfMiniSDK.startMiniApp(this, appId, options);
//或者
TmfMiniSDK.startMiniApp(this, appId, MiniScene.LAUNCH_SCENE_MAIN_ENTRY,
MiniApp.TYPE_ONLINE, options);
```

打开预览和开发版正式小程序

当本地已存在预览\开发版信息（通过 getRecentList 返回本地已存在小程序列表），可以通过此方法打开；否则，需要通过 startMiniAppByLink 扫描二维码打开。

```
TmfMiniSDK.startMiniApp(this, appId, MiniScene.LAUNCH_SCENE_MAIN_ENTRY,
appVerType, options);
```

打开搜索结果小程序

打开通过搜索 TmfMiniSDK.searchMiniApp 接口返回的列表小程序。

```
TmfMiniSDK.startMiniApp(this, appId, MiniScene.LAUNCH_SCENE_SEARCH,
MiniApp.TYPE_ONLINE, options);
```

miniStartOptions.resultReceiver 可用于接收小程序启动错误情况。

```
private ResultReceiver mResultReceiver = new ResultReceiver(new Handler()) {
    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
      if (resultCode != MiniCode.CODE_OK) {
                String errMsg = resultData.getString("errMsg");
                Toast.makeText(mActivity, errMsg + resultCode,
Toast.LENGTH_SHORT).show();
                }
        }
};
```

打开二维码小程序

TMF 内置扫码模块，通过 scan 接口启动扫码，**在 onActivityResult 中调用 getScanResult 对扫码结果进行处理。**

启动选项请参考 MiniStartLinkOptions

```
/**
 * 启动扫码
  *
  * @param activity
  */
public static void scan(Activity activity)
  /**
   * 获取扫码结果
   *
   * @param requestCode  onActivityResult中的requestCode
   * @param intent       onActivityResult中的intent
```

```
  * @return
  */
public static JSONObject getScanResult(int requestCode, Intent intent)
  /**
   * 通过扫码打开小程序
   *
   * @param activity
   * @param link          从getScanResult获取
   * @param resultReceiver 接收小程序启动过程中错误情况
   */
public static void startMiniAppByLink(Activity activity, String link,
MiniStartLinkOptions resultReceiver)
```

扫码结果格式：

```
{
   "scanType": "类型",
   "result": "扫码结果数据，startMiniAppByLink中link参数来源",
   "charset": "编码"
}
```

搜索小程序

搜索选项请参考 SearchOptions

```
/**
* 小程序搜索
*
* @param searchOptions
* @param callback
*/
public static void searchMiniApp(SearchOptions searchOptions,
MiniCallback<List<MiniApp>> callback)
```

获取最近访问小程序列表

```
/**
 * 获取最近访问小程序列表
 * @param callback
 */
public static void getRecentList(IRecentMiniCallback callback)
```

删除小程序

由于小程序的运行，会将小程序包和小程序信息缓存在本地，以便下次更快速的打开。如果想要将小程序的所有信息都删除，可以调用以下 API 删除某个小程序或者删除所有小程序。

```
/**
 * 根据appId删除小程序(正式、开发、预览版都会删除)
 * @param appId
 */
public static void deleteMiniApp(String appId)
/**
* 删除指定类型和版本小程序
* @param appId
* @param appVerType
* @param version
*/
public static void deleteMiniApp(String appId, int appVerType, String version)
```

停止小程序

```
/**
 * 停止小程序
 *
 * @param context
 * @param appId
 */
public static void stopMiniApp(Context context, String appId)

/**
 * 停止所有小程序
 *
 * @param context 上下文
 */
public static void stopAllMiniApp(Context context)
```

预下载小程序

预下载参数请参考 PreDownloadInfo

```
/**
 * 预下载小程序主包
 *
 * @param preDownloadInfo 预下载信息
```

```
 * @param callback 下载回调
 */
public static void preDownloadPkg(PreDownloadInfo preDownloadInfo,
IDownloadCallback callback)

 /**
 * 预下载小程序主包
 * @param preDownloadInfos 预下载信息
 * @param callback 下载回调
 */
public static void preDownloadPkg(List<PreDownloadInfo> preDownloadInfos,
IDownloadCallback callback)
```

当前进程是否是小程序进程

```
/**
 * 当前运行的进程是否是小程序进程
 * @param context
 * @return
 */
public static boolean isMiniProcess(Context context)
```

预加载小程序进程

```
/**
 * 预加载进程, 仅主进程调用才会生效
 *
 * @param bundle 拓展参数，可传null
 */
public static void preloadMiniApp(Context context, Bundle bundle)
```

小程序授权管理

授权状态请参考 MiniAuthState

```
/**
 * 获取小程序授权列表
 * @param appId
 * @param appVerType 小程序版本类型
 * @return
 */
public static List<MiniAuthState> getAuthStateList(String appId, int appVerType)
```

```
/**
 * 设置授权状态
 * @param appId
 * @param appVerType 小程序版本类型
 * @param scopeName 权限名
 * @param grant 是否授权
 */
public static void setAuthState(String appId, int appVerType, String scopeName,
boolean grant)
```

小程序引擎代理实现

最近更新时间：2024-01-16 15:46:41

通过如下设置，宿主需要自定义小程序的一些定制化功能。

```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxy{}
```

定义实现类并继承 BaseMiniAppProxyImpl，并使用上面的注解进行修饰。

自定义胶囊

```
/**
 * 点击胶囊按钮的关闭选项
 * 调用环境：子进程
 *
 * @param miniAppContext 小程序运行环境(小程序进程，非主进程)
 * @param onCloseClickedListener 点击小程序关闭时回调
 * @return 不支持该接口，请返回false
 */
public abstract boolean onCapsuleButtonCloseClick(IMiniAppContext miniAppContext,
        DialogInterface.OnClickListener onCloseClickedListener);

/**
 * 返回胶囊更多面板的按钮，扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添加无
效
 * 调用环境：子进程
 *
 * @param builder
 * @return
 */
public abstract ArrayList<MoreItem> getMoreItems(MoreItemList.Builder builder);

/**
 * 返回胶囊更多面板按钮点击监听器
 * 调用环境：子进程
 *
 * @return 监听器
 */
public abstract OnMoreItemSelectedListener getMoreItemSelectedListener();
```

getMoreItems 实现参考示例

```
@Override
public ArrayList<MoreItem> getMoreItems(IMiniAppContext miniAppContext,
MoreItemList.Builder builder) {
    MoreItem item1 = new MoreItem();
    item1.id = ShareProxyImpl.OTHER_MORE_ITEM_1;
    item1.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_other1);
    item1.drawable = R.mipmap.mini_demo_about;

    MoreItem item2 = new MoreItem();
    item2.id = ShareProxyImpl.OTHER_MORE_ITEM_2;
    item2.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_other2);
    item2.shareKey = SHARE_TWITTER;//自定义分享的key,必须设置且唯一，与小程序端调用
控制设置时会使用到
    item2.drawable = R.mipmap.mini_demo_about;

    MoreItem item3 = new MoreItem();
    item3.id = DemoMoreItemSelectedListener.CLOSE_MINI_APP;
    item3.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_float_app);
    item3.drawable = R.mipmap.mini_demo_about;

    MoreItem item4 = new MoreItem();
    item4.id = ShareProxyImpl.OTHER_MORE_ITEM_INVALID;
    item4.text = getString(miniAppContext,
R.string.applet_mini_proxy_impl_out_of_effect);
    item4.drawable = R.mipmap.mini_demo_about;

    // 自行调整顺序。
    builder.addMoreItem(item1)
            .addMoreItem(item2)
            .addShareQQ("QQ", R.mipmap.mini_demo_channel_qq)
            .addMoreItem(item3)
            .addShareQzone(getString(miniAppContext,
R.string.applet_mini_proxy_impl_Qzone),
                    R.mipmap.mini_demo_channel_qzone)
            .addShareWxFriends(getString(miniAppContext,
R.string.applet_mini_proxy_impl_wechat_friend),
                    R.mipmap.mini_demo_channel_wx_friend)
            .addShareWxMoments(getString(miniAppContext,
R.string.applet_mini_proxy_impl_wechat_group),
                    R.mipmap.mini_demo_channel_wx_moment)
            .addRestart(getString(miniAppContext,
R.string.applet_mini_proxy_impl_restart),
                    R.mipmap.mini_demo_restart_miniapp)
            .addAbout(getString(miniAppContext, R.string.applet_mini_proxy_impl_about),
                    R.mipmap.mini_demo_about)
```

```
            .addDebug(getString(miniAppContext, R.string.mini_sdk_more_item_debug),
                    R.mipmap.mini_demo_about)
            .addMonitor(getString(miniAppContext,
R.string.applet_mini_proxy_impl_performance),
                    R.mipmap.mini_demo_about)
            .addComplaint(getString(miniAppContext,
R.string.applet_mini_proxy_impl_complain_and_report),
                    R.mipmap.mini_demo_browser_report)
            .addSetting(getString(miniAppContext, R.string.mini_sdk_more_item_setting),
                    R.mipmap.mini_demo_setting);

    return builder.build();
}

private String getString(IMiniAppContext miniAppContext, int id) {
    return miniAppContext.getContext().getString(id);
}
```

getMoreItemSelectedListener 实现参考

```
/**
 * 胶囊更多面板按钮点击监听器
 *
 * @return
 */
@Override
public OnMoreItemSelectedListener getMoreItemSelectedListener() {
    return new DemoMoreItemSelectedListener();
}

public class DemoMoreItemSelectedListener extends
DefaultMoreItemSelectedListener {
    public static final int CLOSE_MINI_APP = 150;

    @Override
    public void onMoreItemSelected(IMiniAppContext miniAppContext, int moreItemId)
{
        //处理开发者自定义点击事件(自定义分享事件除外)
        switch (moreItemId) {
            case CLOSE_MINI_APP:
                close(miniAppContext);
                return;
            case OTHER_MORE_ITEM_1:
                miniAppContext.getAttachedActivity().runOnUiThread(new Runnable() {
                    @Override
```

```
                public void run() {
                    Toast.makeText(miniAppContext.getAttachedActivity(), "custom menu
click", Toast.LENGTH_SHORT).show();
                }
            });
            return;
        }

        //处理内置分享和开发者自定义分享，例如：微博、twitter等
        super.onMoreItemSelected(miniAppContext, moreItemId);
    }

    public void close(IMiniAppContext miniAppContext) {
        Activity activity = miniAppContext.getAttachedActivity();
        if (activity != null && !activity.isFinishing()) {
            boolean moved = activity.moveTaskToBack(true);
            if (!moved) {
                QMLog.e("Demo", "moveTaskToBack failed, finish the activity.");
                activity.finish();
            }
        }
    }
}
```

相册选择与图片预览

```
/**
 * 打开选图界面
 * 调用环境：子进程
 *
 * @param context 当前Activity
 * @param maxSelectedNum 允许选择的最大数量
 * @param listner 回调接口
 * @return 不支持该接口，请返回false
 */
public abstract boolean openChoosePhotoActivity(Context context, int
maxSelectedNum, IChoosePhotoListner listner);

/**
 * 打开图片预览界面
 * 调用环境：子进程
 *
 * @param context 当前Activity
 * @param selectedIndex 当前选择的图片索引
```

```
 * @param pathList 图片路径列表
 * @return 不支持该接口，请返回false
 */
public abstract boolean openImagePreview(Context context, int selectedIndex,
List<String> pathList);
```

自定义授权 UI

当小程序调用的 API 需要授权时，SDK 提供如下默认的授权 UI 样式，开发者也可以通过如下方法自定义授权 UI 样式。

```
/**
 * 自定义授权弹窗view
 * 调用环境：子进程
 *
 * @param context
 * @param authInfo
 * @param authView
 * @return true:自定义授权view;false:使用内置
 */
@Override
public boolean authView(Context context, MiniAuthInfo authInfo, IAuthView authView)
{
    return true;
}
```

自定义授权用户信息

实现 BaseMiniAppProxyImpl 如下代码，可以自定义用户授权信息中用户昵称和头像。

```
/**
 * 获取scope.userInfo授权用户信息
 * 调用环境：子进程
 *
 * @param appId
 * @param result
 */
@Override
public void getUserInfo(String appId, AsyncResult result) {
    JSONObject jsonObject = new JSONObject();
    try {
        //返回昵称
        jsonObject.put("nickName", "userInfo测试");
        //返回头像url
        jsonObject.put("avatarUrl",
"https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fimg.daimg.com%2Fuplo
ads%2Fallimg%2F210114%2F1-
210114151951.jpg&refer=http%3A%2F%2Fimg.daimg.com&app=2002&size=f9999,1
0000&q=a80&n=0&g=0n&fmt=auto?
sec=1673852149&t=e2a830d9fabd7e0818059d92c3883017");
        result.onReceiveResult(true, jsonObject);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

可参考如下方法实现加载头像：

```
public Drawable getDrawable(Context context, String source, int width, int hight,
Drawable defaultDrawable)
```

小程序数据根据账号隔离存储

当进行账号切换时，如当前账号下已有运行的小程序，建议先调用 stopAllMiniApp 停止运行的小程序，然后再进行账号切换。

```
/**
 * 用户账号,必须唯一，设置后数据会按账号隔离存储,不建议使用用户敏感信息
 * 调用环境：主进程
 */
@Override
public String getAccount() {
    return "tmf_test";
```

```
}
```

小程序图片加载

小程序中需要加载网络或本地图片时，需要实现如下接口：

```
/**
 * 获取Drawable对象
 * 调用环境：子进程
 *
 * @param context        上下文
 * @param source         来源，可以是本地或者网络
 * @param width          图片宽度
 * @param hight          图片高度
 * @param defaultDrawable 默认图片,用于加载中和加载失败
 */
@Override
public Drawable getDrawable(Context context, String source, int width, int hight,
Drawable defaultDrawable)
```

demo 中参考示例

```
@Override
public Drawable getDrawable(Context context, String source, int width, int hight,
Drawable defaultDrawable) {
    //接入方接入自己的ImageLoader
    //demo里使用开源的universalimageloader
    UniversalDrawable drawable = new UniversalDrawable();
    if (TextUtils.isEmpty(source)) {
        return drawable;
    }
    drawable.loadImage(context, source);
    return drawable;
}
```

自定义虚拟域名

小程序打用到默认的虚拟域名，这个域名并非真实的域名，在浏览器中是无法访问的，如果需要自定义可以按照如下配置进行设置。

DevTools

Devices
Pages
Extensions
Apps
Shared workers
Service workers
Other

Devices

☑ Discover USB devices Port forwarding...

☑ Discover network targets Configure...

Open dedicated DevTools for Node

Remote Target #LOCALHOST

SM-G8870 #2CE0221CA21D7ECE

WebView in com.tencent.tmf.miniapp.demo (77.0.3865.120) trace

untitled https://appservice.tmf.qq.com/page-frame.html
empty
inspect pause

untitled https://appservice.tmf.qq.com/page-frame.html
empty
inspect pause

untitled about:blank
empty
inspect pause

local_api_demo:page/component/index.html:VISIBLE https://appservice.tmf.qq.com/page-frame.html
at (0, 252) size 1080 × 1907
inspect pause

untitled about:blank
empty
inspect pause

```
@Override
public MiniConfigData configData(Context context, int configType, JSONObject
params) {
    if(configType == MiniConfigData.TYPE_DOMAIN) {
        //虚拟域名配置
        MiniConfigData.DomainConfig domainConfig = new
MiniConfigData.DomainConfig();
        domainConfig.domain = "test.com";

        return new MiniConfigData
                .Builder()
                .domainConfig(domainConfig)
                .build();
    }

    return new MiniConfigData
                .Builder()
                .build();
}
```

自定义 userAgent

可以通过如下方式自定义 webview的userAgent。

```
@Override
public MiniConfigData configData(Context context, int configType, JSONObject
params) {
    if(configType == MiniConfigData.TYPE_WEBVIEW) {
        //webView userAgent
        String ua =
params.optString(MiniConfigData.WebViewConfig.WEBVIEW_CONFIG_UA);
        MiniConfigData.WebViewConfig webViewConfig = new
MiniConfigData.WebViewConfig();
        //设置开发者需要追加的userAgent，注意：不要把原始的ua设置进去了
        webViewConfig.userAgent = "key/value";

        return new MiniConfigData
                .Builder()
                .webViewConfig(webViewConfig)
                .build();
    }

    return new MiniConfigData
            .Builder()
            .build();
}
```

自定义扫码

小程序的扫码 API 功能可以使用平台提供的扫码扩展库进行扫码，也支持自定义扫码。

```
/**
 * 扫码二维码
 *
 * @param context        上下文
 * @param onlyFromCamera 只允许摄像头扫码
 * @param result         扫描结果
 * @return true:自定义扫码;false:使用内置扫码
 */
@Override
public boolean enterQRCode(Context context, boolean onlyFromCamera, AsyncResult
result) {
    return false;
}
```

保存文件目录

```
/**
 * 指定小程序API保存图片、视频到系统相册中的目录
 *
 */
public abstract String getSaveFileDir();
```

小程序生命周期

```
/**
 * 小程序生命周期事件回调
 * 调用上下文环境：主进程UI线程
 *
 * @param appState 事件状态
 * @param event 事件
 */
public abstract void onAppStateChange(@AppState int appState, MiniAppEvent
event);
```

基础库更新

```
/**
 * 基础库检测到更新时，是否进行更新
 * @param context
 * @param data 基础库信息数据
 * @return true:更新;false:不更新，默认为 true
 */
public abstract boolean isUpdateBaseLib(Context context, JSONObject data);
```

扩展 SDK

最近更新时间：2024-02-26 10:34:51

本文将为您介绍，通过扩展库的方式集成组件，支持的组件可参见下文。

X5内核扩展 SDK

如果开发者原来的工程中已集成了腾讯X5内核，小程序 SDK 支持多种版本X5内核，客户需根据自身工程实际情况进行选择。

> **注意：** 集成小程序 SDK 后开发者切记不要自己调用X5的初始化，因为小程序 SDK 会进行初始化，如果开发者自己调用初始化可能导致X5初始化失败，进而影响小程序同层渲染能力。

- **公网版本 X5 内核**

  公网版本 X5 内核为腾讯 X5对外提供使用的内核，它属于共享内核（共享微信、QQ 等内核），且会访问腾讯相关后台服务。

```
implementation 'com.tencent.tbs:tbssdk:${version}' //版本信息请参考SDK 更新动态
implementation 'com.tencent.tcmpp.android:mini_extra_public_x5:${version}' //版本信息请参考SDK 更新动态
```

- **静态内核**

  静态 x5 内核将整个内核打包的 aar 中，同时支持 armeabi、arm64-v7a、arm64-v8a 多种架构，但 aar 包体积比较大；**并支持小程序同层渲染。**

```
//在工程中引入依赖
implementation 'com.tencent.tmf.android:tbscore:${version}' //版本信息请参考SDK 更新动态
implementation 'com.tencent.tcmpp.android:mini_extra_static_x5_new:${version}' //版本信息请参考SDK 更新动态

//工程的application中开启extractNativeLibs配置，必须设置为true，否则内核初始化失败
<application
    android:extractNativeLibs="true">
</application>
```

  开发者选择过滤打包后的 apk 支持的架构，来减小 apk 的大小。

```
ndk { abiFilters "armeabi" "armeabi-v7a" "arm64-v8a" }
```

配置内核 LicenseKey，配置代码详细说明参考 SDK 初始化，LicenseKey 需要找相关人员进行申请。

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
    * 创建初始化配置信息
    * @return
    */
    @Override
    public MiniInitConfig buildConfig() {
      MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
      MiniInitConfig config = builder
              .x5LicenseKey("")//设置LicenseKey
              .build();
    }
}
```

- **动态版 X5 内核**

动态内核集成到 app 的 sdk 体积小，内核从服务端下载。

```
implementation 'com.tencent.tmf.android:dynamicx5:${version}'
implementation 'com.tencent.tcmpp.android:mini_extra_dynamic_x5:${version}'
//版本信息请参考SDK 更新动态
```

使用开发动态内核的开发者依然需要设置 LicenseKey 和内核下载地址，配置代码详细说明参考 SDK 初始化。

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
    * 创建初始化配置信息
    * @return
    */
    @Override
    public MiniInitConfig buildConfig() {
      MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
      MiniInitConfig config = builder
              .x5LicenseKey("")//设置LicenseKey
              .coreUrl32("")//32位内核下载地址
              .coreUrl64("")//64位内核下载地址
```

```
            .build();
    }
}
```

部署内核文件到服务器，并按照规则生成对应的 URL 地址。
内核文件 URL 地址的生成规则如下：

```
( http 或 https ) ://domain/path/{versionCode}/{tbscore.tbs}
```

- {versionCode}：为获取内核时提供的内核版本号 。
- {tbscore.tbs}：为内核文件。

  需要保证 URL 以" / "截取时，内核文件为最后一位，versionCode 为倒数第二位。
  示例如下：
  - 32位内核下载地址
  - 64位内核下载地址

  本例当中，内核版本号为 46471，内核文件为 tbs_core_release.tbs。

X5 内核事件回调

```
@ProxyService(proxy = IX5EventProxy.class)
public class X5EventProxyImpl implements IX5EventProxy {
    /**
     * 动态内核下载进度
     * @param progress
     */
    @Override
    public void onDownloadProgress(int progress) {
        Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadProgress=" +
progress);
    }

    /**
     * 动态内核下载失败
     * @param code
     * @param msg
     */
    @Override
    public void onDownloadFailed(int code, String msg) {
        Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadFailed=" + code + " " +
msg);
    }
```

```
    /**
     * 动态内核下载完成
     */
    @Override
    public void onDownloadFinish() {
        Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadFinish");
    }

    /**
     * 内核初始化回调
     * @param isX5 true:x5初始化成功;false：x5初始化失败
     */
    @Override
    public void init(boolean isX5) {
        Log.d(ModuleApplet.TAG, "X5EventProxyImpl isX5=" + isX5);
    }
}
```

扫码扩展 SDK

- 组件说明：开发者小程序如果使用了小程序扫码能力，则需要添加如下 SDK 支持扫码功能；如未使用，建议暂不添加，这样可以减小 App 包大小。
- 集成方法：按照如下的方式添加扫码扩展库依赖：

```
//扫码扩展组件
implementation 'com.tencent.tcmpp.android:mini_extra_qrcode:${version}' // 版本信息
请参考SDK 更新动态
```

添加扫码扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.scanCode	调起客户端扫码界面进行扫码

涉及权限：

权限	描述
相机权限	需要申请相机权限用于扫码
文件读写权限	需要申请文件读写权限用于识别本地图片中的二维码

腾讯定位地图扩展 SDK

- 组件说明：针对中国大陆地区 App 开发，开发者小程序如果使用了小程序地图能力，则需要添加如下 SDK 支持腾讯地图功能；如未使用，建议暂不添加，这样可以减小 App 包大小。
- 集成方法：按照如下的方式添加地图扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_map:${version}'// 版本信息请参考SDK 更新动态
implementation 'com.tencent.map:tencent-map-vector-sdk:4.5.10'  //版本信息参考 腾讯地图文档
implementation 'com.tencent.map:sdk-utilities:1.0.7'
implementation 'com.tencent.map.geolocation:TencentLocationSdk-openplatform:7.4.7'
```

您需要在您的腾讯位置服务控制台配置项目，并获取访问腾讯地图服务所需要的 API 密钥，详细操作，请参考 开发指南 。
完成上述操作后，您需要在 Android 工程中配置您的 API 密钥。在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置：

```
<application
    ...
    <meta-data
    android:name="TencentMapSDK"
    android:value="(YOUR_API_KEY)" />
    ...
</application>
```

添加腾讯地图扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
地图	支持地图相关接口，包括地图展示，使用地图选择位置以及查询 POI 等

涉及权限：

权限	描述
定位权限	需要使用定位权限用于显示地图定位

Google 及华为定位地图扩展 SDK

- 组件说明：针对境外 App 开发，开发者小程序如果使用了小程序地图能力，需要添加如下 SDK 支持 Google Map 功能；如未使用，建议暂不添加，这样可以减小 App 包大小。

- 集成方法：按照如下的方式添加地图扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_google_map:${version}'//版本信息请参考SDK 更新动态
implementation 'com.google.android.gms:play-services-maps:18.1.0'        //版本信息参考谷歌地图文档（需要外网访问）
implementation 'com.google.maps.android:android-maps-utils:2.3.0'
```

由于部分华为设备不支持内嵌 Google Map，可能导致地图无法显示。您可以额外接入 Petal Map 作为补充方案，小程序框架将在华为设备上优先使用 Petal Map。

```
repositories {
  maven {url 'https://developer.huawei.com/repo/'}
}
implementation 'com.tencent.tcmpp.android:mini_extra_huawei_map:${version}'//版本信息请参考SDK 更新动态
implementation 'com.huawei.hms:maps:6.9.0.300'                    //版本信息参考华为地图文档
implementation 'com.huawei.hms:maps-basic:6.9.0.300'
implementation 'com.huawei.hms:site:6.5.1.300'
```

使用 Google Map 的情形，您需要在您的 Google Cloud Console 配置 Google Cloud 项目，并获取访问 Google 地图服务所需要的 API 密钥，具体操作步骤请参考在 **Google Cloud Console 中进行设置**以及**使用 API 密钥**。
完成上述操作后，您需要在 Android 工程中配置您的 API 密钥。在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置：

```
<application
   ...
   <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="(YOUR_API_KEY)" />
    ...
</application>
```

使用 Petal Map 的情形，您需要在您的华为管理控制台建立项目、开通地图以及位置服务并获取位置服务所使用的 API 密钥，具体操作步骤，请参考 配置 AppGallery Connect 。然后按照 集成 HMS Core SDK 的引导下载“agconnect-services.json”文件至您的项目中并配置华为 AGC 插件。
您需要在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置以正常使用华为的位置服务：

```
<application
    ...
    <meta-data
    android:name="HuaweiApiKey"
    android:value="(YOUR_API_KEY)" />
    ...
</application>
```

> **注意：**
> 出于安全考虑，建议您为位置服务单独申请 API 密钥。

添加 Google、华为地图扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
地图	支持地图相关接口以及组件，包括地图展示，使用地图选择位置以及查询 POI 等

涉及权限：

权限	描述
定位权限	需要使用定位权限用于显示地图定位

直播组件扩展SDK

- 组件说明：如果您需要使用直播组件（live-player 和 live-pusher）进行直播推、拉流相关场景的开发，需要添加如下 SDK 以支持直播组件相关的功能的实现。
- 集成方法：添加直播组件依赖：

```
//直播组件支持库
implementation 'com.tencent.tcmpp.android:mini_extra_trtc_live:${version}'//版本信息请参考SDK 更新动态
//直播组件库
implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release' //版本信息参考 腾讯云文档
```

除了完成以上依赖的添加，您还需要重写实现 BaseMiniAppProxyImpl 的如下方法，提供直播组件需要的 LicenseUrl 和 LicenseKey，以完成直播组件的初始化信息配置；如果您未配置正确的 LicenseUrl 和 LicenseKey，会导致直播组件功能不可用。

说明：
LicenseUrl 和 LicenseKey 的获取方式可参考 新增与续期 License 。

```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxyImpl {
    @Override
   public MiniConfigData configData(Context context, int configType, JSONObject
params) {
      if(configType == MiniConfigData.TYPE_LIVE) {
         //Live直播配置
         MiniConfigData.LiveConfig liveConfig = new MiniConfigData.LiveConfig();
         //下面的key和url仅可用于demo
         liveConfig.licenseKey = "";
         liveConfig.licenseUrl = "";

         return new MiniConfigData
               .Builder()
               .liveConfig(liveConfig)
               .build();
      }

      return null;
   }
}
```

添加直播扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.createLivePusherContext	创建直播推流端 context
LivePusherContext	支持 LivePusherContext 相关接口
wx.createLivePlayerContext	创建直播拉流端 context
LivePlayerContext	支持 LivePlayerContext 相关接口
组件	–
live-pusher	推流标签
live-player	播放标签

涉及权限如下：

权限名称	描述
相机权限	–
录音权限	–

LBS 扩展 SDK

- 组件说明：LBS 组件提供位置信息、罗盘、加速计、定位、设备方向的相关的能力。
- 集成方法：按照如下的方式添加 LBS 扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_lbs:${version}'//版本信息请参考SDK 更新动态
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
位置信息	支持位置信息相关接口
罗盘	支持罗盘相关接口
加速计	支持加速计相关接口
设备方向	支持设备方向相关接口
陀螺仪	支持陀螺仪相关接口

LBS 扩展 SDK 涉及权限如下：

权限	说明
定位	获取定位依赖定位权限

蓝牙扩展 SDK

- 组件说明：添加蓝牙扩展库之后，即可使用蓝牙相关的 API。
- 集成方法：按照如下的方式添加蓝牙扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_bluetooth:${version}'//版本信息请参考SDK 更新动态
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API	说明
蓝牙-通用	蓝牙通用接口
蓝牙-低功耗外围设备	外围设备相关接口
蓝牙-低功耗中心设备	中心设备相关接口
蓝牙-信标	蓝牙信标相关接口

蓝牙扩展 SDK 涉及权限如下：

权限	说明
蓝牙	操作蓝牙需要申请蓝牙权限
定位	蓝牙设备搜索依赖定位权限

NFC 扩展 SDK

- 组件说明：添加 NFC 扩展 SDK，能够实现 NFC 读写相关的能力。
- 集成方法：按照如下的方式添加 NFC 扩展 SDK 依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_nfc:${version}'//版本信息请参考SDK 更新动态
```

添加 NFC 扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.getNFCAdapter	获取 NFC 操作管理实例对象
NFCAdapter	支持 NFCAdapter 相关接口
NFC 实例（NfcA、NfcB、NfcV、NfcF、Ndef、IsoDep、MifareUltralight、MifareClassic）	支持 NFC 标签实例相关接口

所涉及权限：

权限名称	描述
NFC 权限	需要获取 NFC 权限

日历扩展 SDK

- 组件说明：日历扩展 SDK 提供日程创建相关的功能
- 集成方法：按照如下的方式添加日历扩展扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_calendar:${version}'//版本信息请参考SDK 更新动态
```

添加日历扩展 SDK 后，增加支持的小程序 API 列表如下：

API名称	说明
wx.addPhoneCalendar	添加日程
wx.addPhoneRepeatCalendar	添加重复日程

涉及权限：

权限	描述
日历权限	需要授予日历读写权限

生物认证扩展 SDK

- 组件说明：生物认证扩展 SDK 提供设备指纹、人脸识别相关的能力。
- 集成方法：按照如下的方式添加生物认证扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_soter:${version}'//版本信息请参考SDK 更新动态
```

添加生物认证扩展 SDK 后，增加支持的小程序 API 列表如下：

API名称	说明
wx.startSoterAuthentication	–
wx.checkIsSupportSoterAuthentication	–
wx.checkIsSoterEnrolledInDevice	–

涉及权限：

权限	描述
指纹访问	需要申请指纹访问权限

剪切板扩展 SDK

- 组件说明：提供剪切板访问的能力
- 集成方法：按照如下的方式添加扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_clipboard:${version}'//版本信息请参考SDK 更新动态
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API名称	说明
wx.getClipboardData	–
wx.setClipboardData	–

涉及权限：

权限	描述
剪贴板权限	需要申请剪切板访问权限

通讯录扩展 SDK

- 组件说明：提供联系人访问相关能力。
- 集成方法：按照如下的方式添加扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_contact:${version}'//版本信息请参考SDK 更新动态
```

添加通讯录扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.addPhoneContact	添加联系人
wx.chooseContact	选择联系人

涉及权限：

权限	描述
联系人读写权限	需要申请联系人访问、写入权限

文档引擎扩展 SDK

- 组件说明：提供文档（pdf、word、excel、ppt等）打开能力。
- 集成方法：按照如下的方式添加扩展库依赖：

```
implementation 'com.tencent.tmf.android:tbs-doc-support:${version}'//版本信息请参考SDK 更新动态
implementation 'com.tencent.tcmpp.android:mini_extra_doc:${version}'//版本信息请参考SDK 更新动态
```

配置文档引擎LicenseKey，配置代码详细说明参考 SDK 初始化，LicenseKey需要找相关人员进行申请；

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
    * 创建初始化配置信息
    * @return
    */
   @Override
   public MiniInitConfig buildConfig() {
     MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
     MiniInitConfig config = builder
             .docLicenseKey("")//设置LicenseKey
             .build();
   }
}
```

媒体扩展 SDK

- 组件说明：提供 chooseMedia、previewMedia 的默认实现。
- 集成方法：按照如下的方式添加扩展库依赖：

```
implementation 'com.tencent.tcmpp.android:mini_extra_media_support:${version}'//版本信息请参考SDK 更新动态
```

实现 MediaImageLoaderProxy 代理，使用自定义的图片加载实现，用于 mini_extra_media_support 库的图片加载。

> **说明：**
> 可以通过实现 MediaChooseJsProxy 代理，实现自定义的 chooseMedia 逻辑。

```
@ProxyService(proxy = MediaImageLoaderProxy.class)
public class CustomMediaImageLoaderProxy implements MediaImageLoaderProxy {
    private GlideImageEngine glideImageEngine = new GlideImageEngine();

    @Override
    public ImageEngine getCustomImageEngine() {
        return glideImageEngine;
    }

    static class GlideImageEngine implements ImageEngine {

        @Override
        public void loadPhoto(@NonNull Context context, @NonNull Uri uri, @NonNull
ImageView imageView) {
            Glide.with(context).load(uri).transition(withCrossFade()).into(imageView);

        }

        @Override
        public void loadGifAsBitmap(@NonNull Context context, @NonNull Uri gifUri,
@NonNull ImageView imageView) {
            Glide.with(context).asBitmap().load(gifUri).into(imageView);

        }

        @Override
        public void loadGif(@NonNull Context context, @NonNull Uri gifUri, @NonNull
ImageView imageView) {

Glide.with(context).asGif().load(gifUri).transition(withCrossFade()).into(imageView);

        }

        @Override
        public Bitmap getCacheBitmap(@NonNull Context context, @NonNull Uri uri, int
width, int height)
                throws Exception {
            return Glide.with(context).asBitmap().load(uri).submit(width, height).get();
        }
    }
}
```

自定义UI

最近更新时间：2023-12-06 15:14:25

通过如下设置，宿主可以自定义小程序 UI。

```
@ProxyService(proxy = IMiniUiProxy.class)
public class MiniUiProxyImpl extends AbsMiniUiProxy
```

定义实现类并继承 AbsMiniUiProxy，并使用上面的注解进行修饰。

胶囊 UI

```
/**
 * 自定义导航栏返回icon, 宽高比要求=24x43
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色, 1:black 0:white
 * @return
 */
@DrawableRes
int navBarBackRes(int mode);

/**
 * 导航栏返回主页图标icon, 宽高比要求=48x48
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色, 1:black 0:white
 * @return
 */
@DrawableRes
int homeButtonRes(int mode);

/**
 * 胶囊更多图标icon, 宽高比要求=80x59
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色, 1:black 0:white
 * @return
 */
@DrawableRes
int moreButtonRes(int mode);
```

```
/**
 * 胶囊关闭图标icon, 宽高比要求=80x59
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色, 1:black 0:white
 * @return
 */
@DrawableRes
int closeButtonRes(int mode);

/**
 * 胶囊按钮中间分割线背景颜色
 * 调用环境：子进程
 *
 * @return
 */
@DrawableRes
int lineSplitBackgroundColor();
```

小程序授权 UI

当小程序调用的 API 需要授权时，SDK 提供如下默认的授权 UI 样式，开发者也可以通过如下方法自定义授权 UI 样式。

```
/**
 * 自定义授权弹窗view
 * 调用环境：子进程
 *
 * @param context
 * @param authInfo
 * @param authView
 * @return true:自定义授权view;false:使用内置
 */
@Override
public boolean authView(Context context, MiniAuthInfo authInfo, IAuthView authView)
{
    return true;
}
```

授权用户信息 UI

可以自定义用户授权信息中用户昵称和头像。

```
/**
 * 获取scope.userInfo授权用户信息
 * 调用环境：子进程
 *
 * @param appId
 * @param result
 */
@Override
public void getUserInfo(String appId, AsyncResult result) {
    JSONObject jsonObject = new JSONObject();
    try {
        //返回昵称
        jsonObject.put("nickName", "userInfo测试");
        //返回头像url
        jsonObject.put("avatarUrl",
"https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fimg.daimg.com%2Fuploads%2Fallimg%2F210114%2F1-210114151951.jpg&refer=http%3A%2F%2Fimg.daimg.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=auto?sec=1673852149&t=e2a830d9fabd7e0818059d92c3883017");
        result.onReceiveResult(true, jsonObject);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

为了加载头像，还需要实现 BaseMiniAppProxyImpl 如下方法（参考 小程序宿主自定义 ）。

```
public Drawable getDrawable(Context context, String source, int width, int hight,
Drawable defaultDrawable)
```

小程序 Loading

小程序打开过程中有检查更新和启动加载loading，可以通过如下方式自定义 loading。

```
/**
 * 自定义小程序检查更新loading页面
 * 调用环境：主进程
 *
 * @param context
 * @return
 */
public abstract IMiniLoading updateLoadingView(Context context);

/**
 * 自定义小程序加载loading页面
 * 调用环境：子进程
 *
 * @param activityWeakRef Activity引用
 * @param app 小程序信息
 * @return 返回小程序loading UI
 */
public abstract IMiniLoading startLoadingView(WeakReference<Activity>
activityWeakRef, MiniAppLoading app);
```

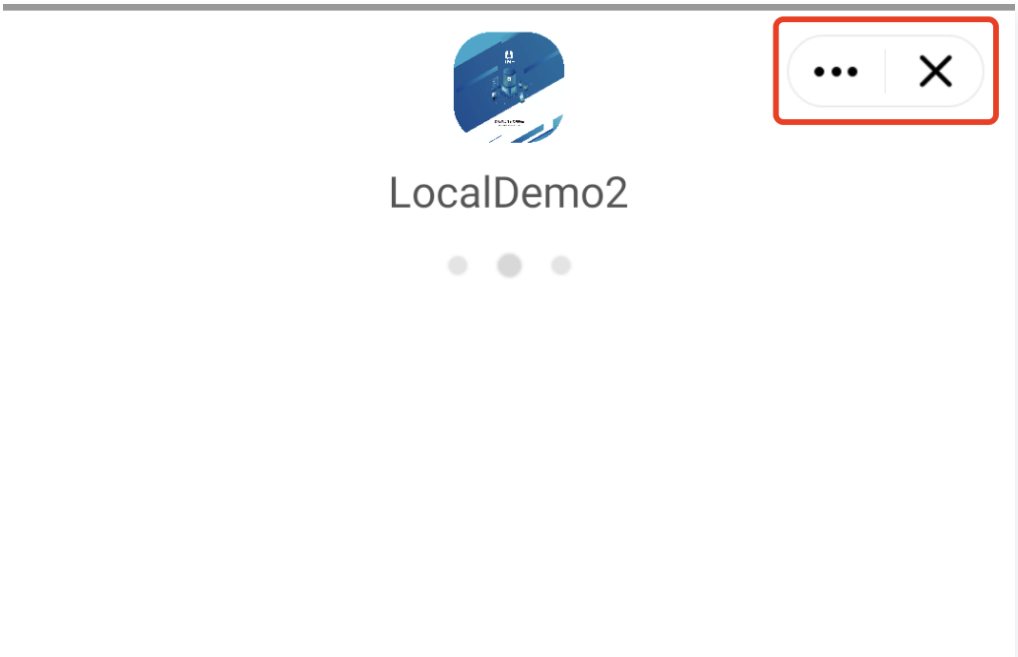
示例：

```
@Override
public IMiniLoading updateLoadingView(Context context) {
    return new IMiniLoading() {
        @Override
        public View create() {
            return
LayoutInflater.from(context).inflate(R.layout.applet_activity_custom_update_loading,
null);
        }

        @Override
        public void show(View v) {

        }

        @Override
        public void stop(View v) {
```

```
        }
    };
}
```

小程序加载 loading 页面右上角胶囊

```
/**
 * 是否隐藏小程序加载loading页面右上角胶囊
 *
 * @return true:隐藏;false:不隐藏（默认值）
 */
boolean hideLoadingCapsule();
```



自定义API

最近更新时间：2023-12-28 11:32:51

创建 API

参考示例：

```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("customAsyncEvent")
    public void custom(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "test");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }

    @JsEvent("customSyncEvent")
    public String custom1(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //同步返回数据
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "value");
        } catch (JSONException e) {
            throw new RuntimeException(e);
        }
        return req.okSync(jsonObject);
    }

    /**
     * 测试覆盖系统API
     * @param req
     */
    @JsEvent("getAppBaseInfo")
```

```
    public void getAppBaseInfo(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "test");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }

    @JsEvent("testState")
    public void testState(final RequestEvent req) {

      try {
          //回调中间状态
          req.sendState(req, new JSONObject().put("progress", 1));
          req.sendState(req, new JSONObject().put("progress", 30));
          req.sendState(req, new JSONObject().put("progress", 60));
          req.sendState(req, new JSONObject().put("progress", 100));
      } catch (JSONException e) {
          e.printStackTrace();
      }

      JSONObject jsonObject = new JSONObject();
      try {
          jsonObject.put("key", "test");
          req.ok(jsonObject);
      } catch (JSONException e) {
        throw new RuntimeException(e);
      }
    }
}
```

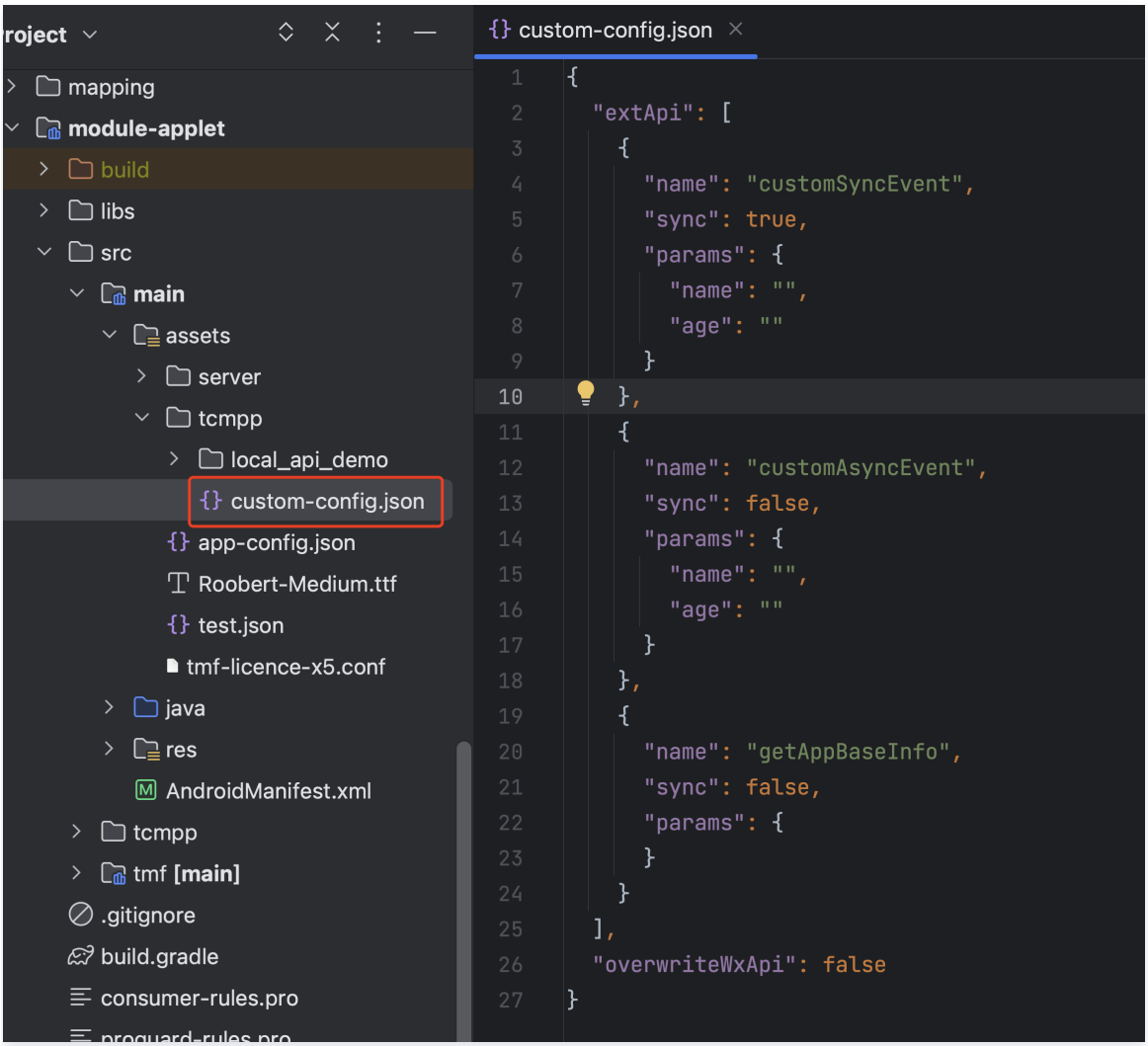
说明：

- 继承 BaseJsPlugin 并用注解进行定义 @JsPlugin(secondary = true)。
- 定义一个方法，方法只能有一个参数且参数必须是 RequestEvent 类型。
- 然后在方法上定义注解@JsEvent("事件名")，当小程序 js 调用“事件名”时就会调用到@JsEvent修饰的对应方法。
- @JsEvent 支持定义多个事件名。

- 支持同步或异步返回数据（同一事件只能选择一种方式）。
- 可以通过调用 sendState 给小程序端多次返回中间状态，sendState 调用结束后必须调用 ok 或 fail 标识整个流程结束。

定义 API 配置

```
{
 "extApi": [
  {
   //事件名，与“创建API”示例中@JsEvent定义的事件保持一致
   "name": "customSyncEvent",
   //是否同步调用，与“创建API”示例数据返回方式保持一致
   "sync": true,
   //定义参数
   //a.json格式可以嵌套
   //b.字符串参数value设置为""即可
   //c.数字参数value设置为0即可
   "params": {
    "name": "",
    "age": 0,
    "object": {
     "key": "",
    }
   }
  }
 ],
 //true：当自定义的API事件名与小程序SDK内置的方法名，会覆盖SDK内置的
 //API，最终调用会调用的开发者自定义的API中
 "overwriteWxApi": false
}
```

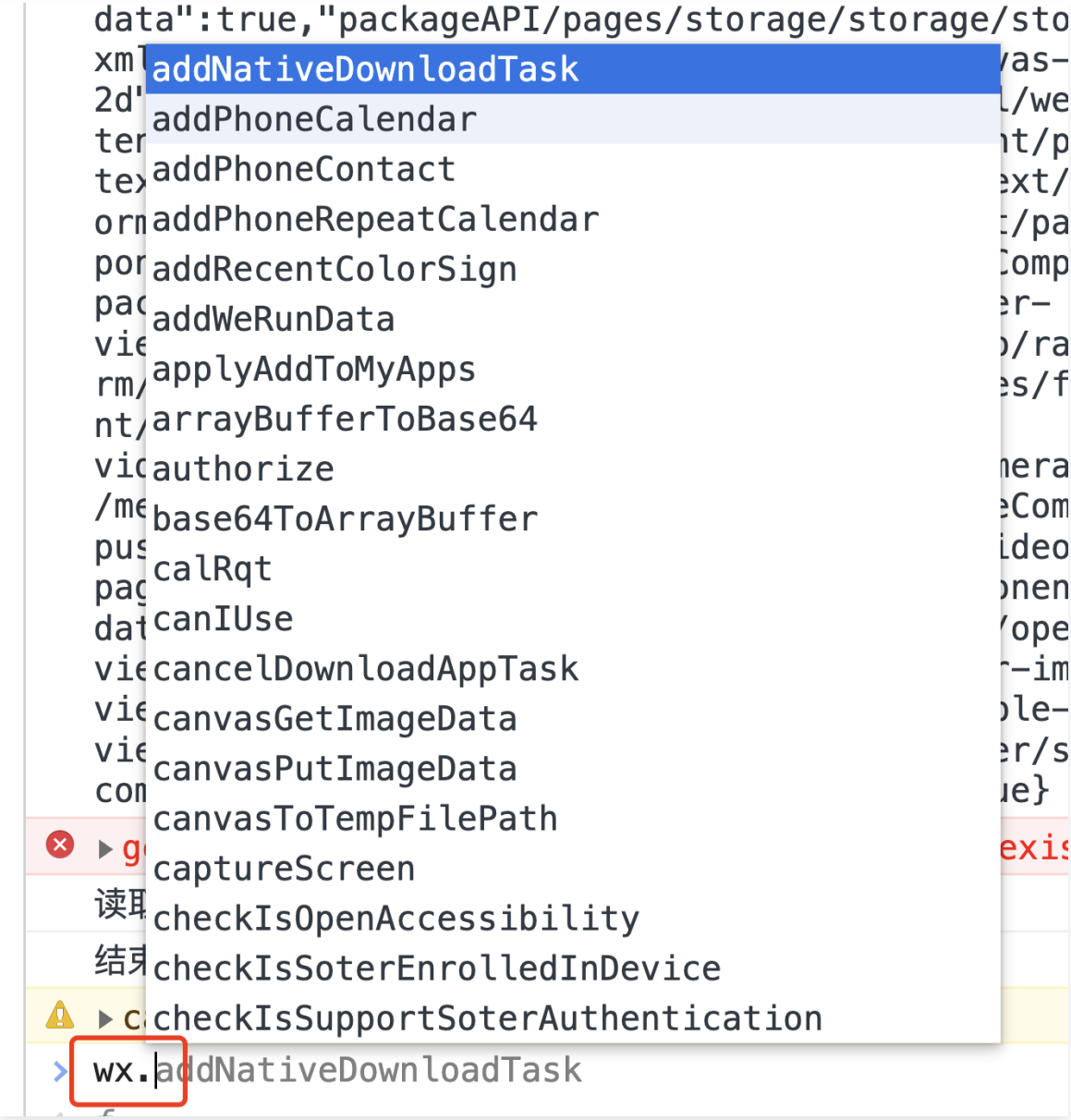


将定义的 API 配置文件存放到工程的 assets 目录下（文件名和路径开发者可以自己定义），然后按照如下方法设置 API 配置文件路径。

```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxyImpl {
    @Override
   public MiniConfigData configData(Context context, int configType, JSONObject
params) {
      if(configType == MiniConfigData.TYPE_CUSTOM_JSAPI) {
         //自定义JsApi配置
         MiniConfigData.CustomJsApiConfig customJsApiConfig = new
MiniConfigData.CustomJsApiConfig();
         customJsApiConfig.jsApiConfigPath = "tcmpp/custom-config.json";

         return new MiniConfigData
               .Builder()
               .customJsApiConfig(customJsApiConfig)
               .build();
      }
```

```
    return null;
}
```

在搜索栏输入 `wx.` 时，会自动弹出展示小程序系统内置的 API 接口，如下图所示：



Json API 配置参考示例：

```
{
 "extApi": [
  {
   "name": "customSyncEvent",
   "sync": true,
   "params": {
    "name": "",
    "age": ""
```

```
      }
    },
    {
      "name": "customAsyncEvent",
      "sync": false,
      "params": {
        "name": "",
        "age": ""
      }
    },
    {
      "name": "getAppBaseInfo",
      "sync": false,
      "params": {
      }
    }
  "overwriteWxApi": true
}
```

小程序端调用代码

上述示例中，代码在小程序中的调用如下：

```
wx.customAsyncEvent({"name":"123","age":"18"})
wx.getAppBaseInfo()//会覆盖系统API，然后调用到自定义API中
```

自定义 JSAPI 错误排查

- 原因一：确定是否生成了如下类 XxxJsPluginScope。

- 原因二：检查客户端定义的事件与小程序调用方法名是否一致，事件名区分大小写的。

插件中启动 Activity

插件中启动新 Activity 并获取获取 Activity 返回的数据。

```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
   @JsEvent("testStartActivityForResult")
   public void testStartActivityForResult(final RequestEvent req) {
     Activity activity = req.activityRef.get();
     TmfMiniSDK.addActivityResultListener(new IActivityResultListener() {
        @Override
        public boolean doOnActivityResult(int requestCode, int resultCode, Intent data)
{
           TmfMiniSDK.removeActivityResultListener(this);

           Log.i(ModuleApplet.TAG, data.getStringExtra("key"));
           req.ok();
           return true;
        }
     });

     //注意：requestCode必须>=1000000，否则可能与内部requestCode冲突，引起未知问题
     activity.startActivityForResult(new Intent(activity, TransActivity.class), 1000000);
  }
}
```

添加 Activity 返回监听，并在监听处理完成后移除监听：

```
TmfMiniSDK.addActivityResultListener()
TmfMiniSDK.removeActivityResultListener()
```

启动新的 Activity。

```
//注意：requestCode必须>=1000000，否则可能与内部requestCode冲突，引起未知问题
activity.startActivityForResult(new Intent(activity,  Xxxx.class), 1000000);
```

插件中调起第三方 App

当在插件中调用其它第三方 App（分享、支付、登录）等，当这些操作完成后能直接返回小程序而不是返回 App。

- 首先开发者创建透明的辅助Activity

  如下示例 TransActivity 为定义的透明辅助 activity（透明 activity 开发者自行配置，因为主题的设置与应用 Activity 继承的系统 Activity 有关）

```
<activity android:name=".activity.TransActivity"
    android:exported="false"
    android:screenOrientation="portrait"
    android:theme="@style/TransparentTheme"/>
```

- 然后在 TransActivity 中调用业务逻辑。

```
public class TransActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.applet_activity_tran);

        /*************************
            业务开发
        ************************/

        //业务开发完成后返回数据并调用finish
        Intent intent = new Intent();
        intent.putExtra("key", "value");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

- 插件启动 Activity 并监听数据返回。

```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("testStartActivityForResult")
    public void testStartActivityForResult(final RequestEvent req) {
      Activity activity = req.activityRef.get();
      TmfMiniSDK.addActivityResultListener(new IActivityResultListener() {
          @Override
          public boolean doOnActivityResult(int requestCode, int resultCode, Intent
data) {
              TmfMiniSDK.removeActivityResultListener(this);

              Log.i(ModuleApplet.TAG, data.getStringExtra("key"));
              req.ok();
              return true;
          }
      });
```

```
    //注意：requestCode必须>=1000000，否则可能与内部requestCode冲突，引起未知
问题
    activity.startActivityForResult(new Intent(activity, TransActivity.class),
1000000);
  }
}
```

自定义原生组件

最近更新时间：2024-04-28 17:24:31

小程序 SDK 的使用者可以自定义客户端原生组件为小程序提供支持。小程序开发者通过特定的小程序 API 能够在小程序页面中创建并操作原生组件以及与原生组件进行通信。Android 的客户端原生组件分为两种类型：

- 非同层原生组件：原生组件渲染在小程序页面之上，不支持 zIndex，始终位于小程序其他组件上层并会遮挡小程序内容。该类型组件以普通 View 的形式进行绘制。
- 同层原生组件：原生组件渲染在小程序页面内，支持 zIndex 以及页面内的组件层级关系。该类型组件需要以 Surface 的形式绘制在小程序页面中。

小程序向页面中插入客户端自定义的组件需要先在 wxml 中引入一个 external-element 节点：

```
<external-element
   id="comp1"
   type="maTestView"
   _insert2WebLayer="true"
   style="width: 200px;height: 100px;"
   bindexternalelementevent="handleEvent"
></external-element>
```

此处 type 为与应用约定创建的组件类型名称一致，_insert2WebLayer 表示该组件为同层组件或者非同层组件（true 为同层，需客户端实现同层组件代理；false 为非同层，需要客户端实现非同层代理），bindexternalelementevent 可捕获 native 传递的 onExternalElementEvent 或者。onXWebExternalElementEvent，回调参数包括：

```
{
   target,
   currentTarget,
   timeStamp,
   touches,
   detail, // native传递的参数
}
```

之后小程序通过 id 创建一个与该节点相关联的上下文：

```
this.ctx = wx.createExternalElementContext('comp1');
```

该方法会通知应用在节点的位置创建对应的原生组件。小程序后续可以通过该上下文向原生组件发送事件，对原生组件进行操作：

```
this.ctx.call({
  params1: {
    name: 'name1',
    age: 11
  },
  params2: {
    name: 'name2',
    age: 22
  },
  success: (e) => {
    console.log('====operate success=====', e)
  },
  fail: (e) => {
    console.log('====operate fail=====', e)
  },
  complete: (e) => {
    console.log('====operate complete=====', e)
  }
})
```

应用的开发者需要实现特定的代理，在小程序调用自定义组件 API 时创建原生组件以及响应对原生组件的操作。以下会针对非同层组件以同层组件分别进行说明。

自定义非同层原生组件

通过如下设置覆写代理，宿主可以自定义创建非同层原生组件：

```
@ProxyService(proxy = ExternalElementProxy.class)
public class MyExternalElementProxy extends ExternalElementProxy{}
```

代理需要实现以下内容：

1. 插入非同层组件。当小程序向页面中插入非同层原生组件时会调用此方法。开发者需要实现此方法，并将自定义组件作为子 View 加入到 parent 参数提供的容器中。

```
/**
 * 创建非同层组件，当小程序创建自定义非同层原生组件时将会调用此接口
 *
 * @param widgetId 小程序创建的组件唯一ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 * @param type 小程序创建的组件类型名称
 * @param parent 承载原生组件的父容器
 * @param params 小程序创建组件时传递的参数
 */
```

```
public abstract void handleInsertElement(long widgetId, ExternalWidgetContext
widgetContext,
                                String type, ViewGroup parent, JSONObject params);
```

2. 更新非同层组件。当原生组件在小程序中的位置以及大小发生变化时会调用此方法通知应用。原生组件的父容器布局会按照变化后的样式进行自适应，原生组件本身则可以根据需要对自身进行调整。

```
/**
 * 更新非同层组件样式，当小程序更新自定义非同层原生组件的样式时将会调用此接口
 *
 * @param widgetId 小程序组件ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 * @param params 小程序更新组件时传递的参数
 */
public abstract void handleUpdateElement(long widgetId, ExternalWidgetContext
widgetContext,
                                JSONObject params);
```

3. 操作非同层组件。当小程序向原生组件发送事件时会调用此方法（例如按钮点击，状态变更），事件的具体内容需要开发者自行在 params 中进行定义。

```
/**
 * 操作非同层组件，当小程序需要向非同层组件发送指令或者调用特有方法的时候将会调用此
接口
 *
 * @param widgetId 小程序组件ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 * @param params 小程序更新组件时传递的参数
 */
public abstract void handleOperateElement(long widgetId, ExternalWidgetContext
widgetContext,
                                JSONObject params);
```

4. 删除非同层组件。当小程序删除已添加的原生组件时将调用此方法通知应用。此时组件的父容器将被移除，应用应当对组件进行销毁处理。

```
/**
 * 删除非同层组件
 *
 * @param widgetId 小程序组件ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 */
```

```
public abstract void handleRemoveElement(long widgetId, ExternalWidgetContext
widgetContext);
```

小程序非同层原生组件上下文

小程序组件的上下文包含了使原生组件能够向对应的小程序组件发送消息的方法。onExternalElementEvent 方法会直接向小程序发送 onExternalElementEvent 事件，小程序应该捕获并处理该事件；callbackSuccess、callbackFail 则是在小程序调用 API 向应用发送事件时对当次小程序 API 调用传入的 success 或者 fail 方法进行回调。

```
/**
 * 向小程序发送 onExternalElementEvent 事件
 *
 * @param jsonObject 事件携带的JSON数据
 */
public final void onExternalElementEvent(JSONObject jsonObject);

/**
 * 调用小程序提供的success回调方法
 *
 * @param jsonObject 回调携带的JSON数据，可以为空
 */
public final void callbackSuccess(JSONObject jsonObject);

/**
 * 调用小程序提供的fail回调方法
 *
 * @param jsonObject jsonObject 回调携带的JSON数据，可以为空
 * @param message 错误信息描述
 */
public final void callbackFail(JSONObject jsonObject, String message);
```

自定义同层原生组件

如果小程序请求创建的是同层原生组件，则需要应用实现以下代理：

```
@ProxyService(proxy = EmbeddedExternalElementProxy.class)
public class MyExternalEmbeddedElementProxy extends
EmbeddedExternalElementProxy{}
```

1. 与非同层组件不同，同层组件由于涉及到嵌入页面的 Surface 创建以及销毁，因此会提供额外的组件生命周期回调。开发者可以覆写这些方法对同层组件的生命周期事件进行处理：

```
/**
 * 初始化同层组件DOM节点
 *
 * @param context 页面上下文
 * @param widgetId 同层组件ID
 * @param tagName 同层组件在DOM中的TAG
 * @param attributes 同层组件的Attributes
 */
public abstract void onInit(Context context, long widgetId, String tagName,
Map<String, String> attributes);

/**
 * 同层组件Surface创建完毕回调
 *
 * @param widgetId 同层组件ID
 * @param surface 同层组件关联的Surface
 */
public abstract void onSurfaceCreated(long widgetId, Surface surface);

/**
 * 同层组件Surface销毁回调
 *
 * @param widgetId 同层组件ID
 * @param surface 同层组件关联的Surface
 */
public abstract void onSurfaceDestroyed(long widgetId, Surface surface);

/**
 * 同层组件触摸事件回调
 *
 * @param widgetId 同层组件ID
 * @param event 触摸事件
 * @return 同层组件是否消费触摸事件
 */
public abstract boolean onTouchEvent(long widgetId, MotionEvent event);

/**
 * 同层组件绘制区域变化回调
 *
 * @param widgetId 同层组件ID
 * @param rect 新的绘制区域
 */
public abstract void onRectChanged(long widgetId, Rect rect);

/**
```

```
 * 同层组件请求重绘
 *
 * @param widgetId 同层组件ID
 */
public abstract void onRequestRedraw(long widgetId);

/**
 * 同层组件可见性发生变化
 *
 * @param widgetId 同层组件ID
 * @param visibility 是否可见
 */
public abstract void onVisibilityChanged(long widgetId, boolean visibility);

/**
 * 同层组件变为可用
 *
 * @param widgetId 同层组件ID
 */
public abstract void onActive(long widgetId);

/**
 * 同层组件变为不可用
 *
 * @param widgetId 同层组件ID
 */
public abstract void onDeActive(long widgetId);

/**
 * 同层组件节点被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void onDestroy(long widgetId);

/**
 * 当前小程序页面进入pause状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void webViewPause(long widgetId);

/**
 * 当前小程序页面进入resume状态
 *
```

```
 * @param widgetId 同层组件ID
 */
public abstract void webViewResume(long widgetId);

/**
 * 当前小程序页面被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void webViewDestroy(long widgetId);

/**
 * 小程序进入resume状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void nativeResume(long widgetId);

/**
 * 小程序进入pause状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void nativePause(long widgetId);

/**
 * 小程序被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void nativeDestroy(long widgetId);
```

2. 当小程序真正向页面插入同层组件时会调用应用的 handleInsertXWebExternalElement 接口。客户端开发应当实现该方法，根据 type 提供的组件类型将相应的组件绘制到 widgetId 关联的 Surface 上。

```
/**
 * 插入同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param type 同层组件类型
 * @param req 同层组件透传参数
 */
```

```
public abstract void handleInsertXWebExternalElement(long widgetId,
ExternalWidgetContext widgetContext, String type, JSONObject req);
```

3. 组件的大小以及样式发生变化时，会通过该方法通知应用：

```
/**
 * 更新同层组件样式
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param req 小程序传递的参数
 */
public abstract void handleUpdateXWebExternalElement(long widgetId,
XWebExternalWidgetContext widgetContext, JSONObject req);
```

4. 当小程序组件向原生组件发送事件需要操作原生组件时会通过该方法通知应用：

```
/**
 * 操作同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param req 小程序传递的参数
 */
public abstract void handleOperateXWebExternalElement(long widgetId,
XWebExternalWidgetContext widgetContext, JSONObject req);
```

5. 当小程序删除同层原生组件时会通过该方法通知应用：

```
/**
 * 移除同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 */
public abstract void handleRemoveXWebExternalElement(long widgetId,
XWebExternalWidgetContext widgetContext);
```

小程序同层原生组件上下文

与非同层组件类似，同层组件可以通过小程序组件的上下文向小程序发送 onXWebExternalElementEvent 事件；或是在小程序调用操作原生组件的 API 时回调 API 提供的 callback 方法。

```java
/**
 * 向小程序发送 onXWebExternalElementEvent 事件
 *
 * @param jsonObject 事件携带的JSON数据
 */
public final void onXWebExternalElementEvent(JSONObject jsonObject);

/**
 * 调用小程序提供的success回调方法
 *
 * @param jsonObject 回调携带的JSON数据，可以为空
 */
public final void callbackSuccess(JSONObject jsonObject);

/**
 * 调用小程序提供的fail回调方法
 *
 * @param jsonObject jsonObject 回调携带的JSON数据，可以为空
 * @param message 错误信息描述
 */
public final void callbackFail(JSONObject jsonObject, String message);
```

自定义分享

最近更新时间：2023-10-20 15:19:13

操作步骤

1. 通过 MiniAppProxyImpl，在胶囊控制面板中添加自定义分享 Item。

```
private static final String SHARE_TWITTER = "twitter";

/**
 * 返回自定义分享数据Map
 * 调用环境：子进程
 *
 * key:与getMoreItems方法中添加的MoreItem.id一致
 * value:与getMoreItems方法中添加的MoreItem.shareKey一致
 * @return
 */
@Override
public Map<String, Integer> getCustomShare(){
    Map<String, Integer> objects = new HashMap<>();
    objects.put(SHARE_TWITTER, ShareProxyImpl.OTHER_MORE_ITEM_2);
    return objects;
}

/**
 * 返回胶囊更多面板的按钮，扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添
加无效
 * 调用环境：子进程
 *
 * @param miniAppContext 小程序运行环境(小程序进程，非主进程)
 * @param builder
 * @return
 */
@Override
public ArrayList<MoreItem> getMoreItems(IMiniAppContext miniAppContext,
MoreItemList.Builder builder) {
  MoreItem item2 = new MoreItem();
  item2.id = ShareProxyImpl.OTHER_MORE_ITEM_2;
  item2.text = getString(miniAppContext,
                  R.string.applet_mini_proxy_impl_other2);
  item2.shareKey = SHARE_TWITTER;//自定义分享的key,必须设置且唯一，与小程序端调
用控制设置时会使用到
  item2.drawable = R.mipmap.mini_demo_about;
```

```
  builder.addMoreItem(item2)
  return builder.build();
}
```

2. 如需创建胶囊，可在**更多面板**中，单击**监听器**。

```
/**
 * 返回胶囊更多面板按钮单击监听器
 *
 * @return
 */
@Override
public OnMoreItemSelectedListener getMoreItemSelectedListener() {
    return new DemoMoreItemSelectedListener();
}

public class DemoMoreItemSelectedListener extends
DefaultMoreItemSelectedListener {
    public static final int CLOSE_MINI_APP = 150;

    @Override
    public void onMoreItemSelected(IMiniAppContext miniAppContext, int
moreItemId) {
        //处理开发者自定义点击事件(自定义分享事件除外)
        switch (moreItemId) {
            case CLOSE_MINI_APP:
                close(miniAppContext);
                return;
            case OTHER_MORE_ITEM_1:
                miniAppContext.getAttachedActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(miniAppContext.getAttachedActivity(), "custom
menu click", Toast.LENGTH_SHORT).show();
                    }
                });
                return;
        }

        //处理内置分享和开发者自定义分享，例如：微博、twitter等
        super.onMoreItemSelected(miniAppContext, moreItemId);
    }
}
```

3. 按照如下类型接收分享，单击**事件**，开发人员在 share 方法中可以获取到分享数据，并调用第三方 SDK 实现分享。

> **说明：**
> 如果在胶囊菜单自定义按钮，请参见 自定义胶囊 。

```
@ProxyService(proxy = ShareProxy.class)
public class ShareProxyImpl extends BaseShareProxy {
  /**
   * 分享
   *
   * @param shareData 分享数据
   */
  @Override
  public void share(Activity activity, ShareData shareData) {
  }
}
```

开放接口

最近更新时间：2023-10-20 15:19:14

微信小程序如下方法与 Native 对应关系，当开发者在小程序开发过程中调用对应方法时，会调用到 native 对应事件，开发者需要监听事件并返回数据。

微信方法	native事件
wx.login	login
wx.getUserInfo	getUserInfo
wx.getUserProfile	getUserProfile
wx.getPhoneNumber	getPhoneNumber

```
@JsPlugin(secondary = true)
public class WxApiPlugin extends BaseJsPlugin {
    /**
     * 对应小程序wx.login调用
     * 调用环境：子进程
     *
     * @param req
     */
    @JsEvent("login")
    public void login(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        Log.d(ModuleApplet.TAG, "login=" + req.jsonParams);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.login");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }

    /**
     * 对应小程序wx.getUserInfo调用
```

```
     * 调用环境：子进程
     *
     * @param req
     */
    @JsEvent("getUserInfo")
    public void getUserInfo(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        Log.d(ModuleApplet.TAG, "getUserInfo=" + req.jsonParams);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.getUserInfo");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }

    /**
     * 对应小程序wx.getUserProfile调用
     * 调用环境：子进程
     *
     * @param req
     */
    @JsEvent("getUserProfile")
    public void getUserProfile(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        Log.d(ModuleApplet.TAG, "getUserProfile=" + req.jsonParams);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.getUserProfile");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }

    /**
```

```
     * 对应小程序wx.getPhoneNumber调用
     * 调用环境：子进程
     *
     * @param req
     */
    @JsEvent("wx.getPhoneNumber")
    public void getPhoneNumber(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        Log.d(ModuleApplet.TAG, "getPhoneNumber=" + req.jsonParams);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.getPhoneNumber");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        req.ok(jsonObject);
    }
}
```

常见问题

最近更新时间：2024-04-28 17:24:31

1. 集成相关

1.1 android.support 与 AndroidX 冲突问题

TCMPP SDK 依赖了 AndroidX，客户在接入过程中可能会遇到与 android.support 相关冲突，遇到类似错误：

```
Duplicate class android.support.v4.app.INotificationSideChannel found in modules
core-1.3.1-runtime (androidx.core:core:1.3.1) and support-v4-21.0.3-runtime
(com.android.support:support-v4:21.0.3)
```

解决办法：
在 gradle.properties 中添加如下属性。

```
android.useAndroidX=true
android.enableJetifier=true
```

1.2 模块化工程支持

开发者在模块化工程中多个模块同时使用了注解 @JsPlugin 或 @ProxyService 时，编译工程时会出现如下错误：

```
Execution failed for task ':DEMO:mergeLibDexDebug'.
> A failure occurred while executing com.android.build.gradle.internal.tasks.Workers$ActionFacade
   > com.android.builder.dexing.DexArchiveMergerException: Error while merging dex archives:
     Learn how to resolve the issue at https://developer.android
      .com/studio/build/dependencies#duplicate_classes.
     Program type already present: com.tencent.tmfmini.sdk.core.generated.ExtJsPluginScope

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log
 output. Run with --scan to get full insights.
```

解决办法：

1. 在每个使用了 @JsPlugin 或 @ProxyService 注解的 module 的 build.gradle 中添加如下代码：

```
android {
    defaultConfig {
        javaCompileOptions {
            annotationProcessorOptions {
                //配置模块名：开发者自己定义唯一名称,模块名遵循android类名定义规范
                arguments = [tcmppModuleName: "Demo"]
            }
        }
    }
}
```

2. 初始化代码中注册模块：

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();

        //将上面定义的所有module都进行注册，registerModule参数值与上面
tcmppModuleName定义的保持一致
        return builder
                .registerModule("Demo")
                .registerModule("Test")
                .build();
    }
}
```

> **说明：**
> 如果开发者只有一个模块使用了注解 @JsPlugin 或 @ProxyService，上面配置可以忽略，SDK 内部会有默认模块名。

1.3 常见编译错误

- 情况一

```
> A failure occurred while executing org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction
   > java.lang.reflect.InvocationTargetException (no error message)

* Try:
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Exception is:
org.gradle.api.tasks.TaskExecutionException: Execution failed for task ':libs:libopensdk:kaptEnvTestKotlin'. <33 internal lines>
Caused by: org.gradle.workers.internal.DefaultWorkerExecutor$WorkExecutionException: A failure occurred while executing
 org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction <126 internal lines>
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.gradle.internal.KaptExecution.run(KaptWithoutKotlincTask.kt:285)
    at org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction.execute(KaptWithoutKotlincTask.kt:240)
      <27 internal lines>
    ... 2 more
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing(annotationProcessing.kt:90)
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing$default(annotationProcessing.kt:31)
    at org.jetbrains.kotlin.kapt3.base.Kapt.kapt(Kapt.kt:47)
    ... 34 more
Caused by: com.sun.tools.javac.processing.AnnotationProcessingError Create breakpoint : java.lang.NoClassDefFoundError: ↵
↖com/tencent/tmfmini/sdk/annotation/ProxyService
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.callProcessor(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.discoverAndRunProcs(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment$Round.run(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.doProcessing(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.main.JavaCompiler.processAnnotations(Unknown Source)
    ... 40 more
Caused by: java.lang.NoClassDefFoundError: com/tencent/tmfmini/sdk/annotation/ProxyService
    at com.tencent.tmfmini.sdk.annotation.processor.ProxyServiceProcessor.process(ProxyServiceProcessor.java:48)
    at org.jetbrains.kotlin.kapt3.base.incremental.IncrementalProcessor.process(incrementalProcessors.kt:90)
    at org.jetbrains.kotlin.kapt3.base.ProcessorWrapper.process(annotationProcessing.kt:188)
    ... 45 more
Caused by: java.lang.ClassNotFoundException: com.tencent.tmfmini.sdk.annotation.ProxyService
    ... 48 more
```

解决办法：

检查工程中是否有如下配置，如果有则去掉。

```
kapt.include.compile.classpath=false
```

1.4 Taro框架支持相关

使用 Taro 框架开发的小程序， 启动时逻辑线程可能报如下错误：

```
Uncaught DOMException: Failed to read the 'sessionStorage' property from
'Window': Access is denied for this document.
    at <anonymous>:1207:96250
    at Array.forEach (<anonymous>)
    at Module.<anonymous> (<anonymous>:1207:96164)
    at Module.9 (<anonymous>:1207:113782)
    at l (<anonymous>:1203:566)
    at Module.204 (<anonymous>:1215:87806)
    at l (<anonymous>:1203:566)
    at t (<anonymous>:1203:435)
    at Array.r [as push] (<anonymous>:1203:298)
    at <anonymous>:1215:125
```

解决办法：

Taro 框架开发的小程序，部分特性需依赖 v8 扩展库，请在 app build.gradle 中添加如下依赖。

```
implementation 'com.tencent.tcmpp.android:mini_extra_v8:${version}'
```

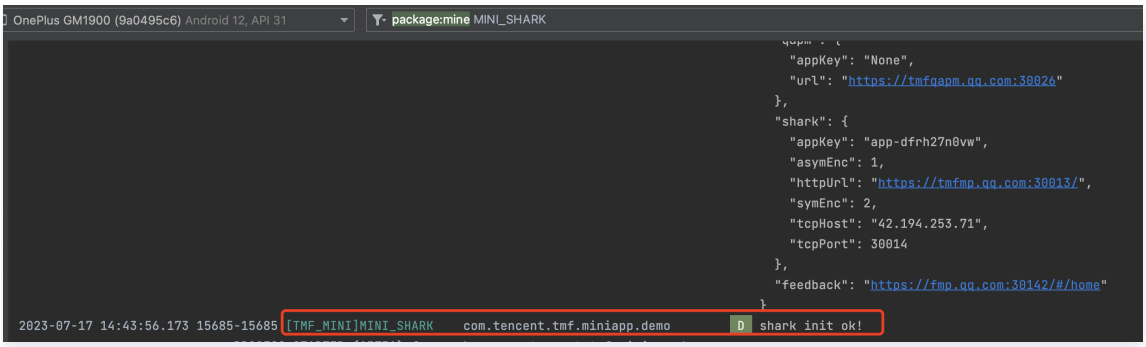
2. 如何跳过小程序域名和隐私 API 校验

正式版小程序不支持跳过域名及 API 校验，如果想在开发测试阶段跳过校验，需打开非正式版 小程序调试（VConsole）

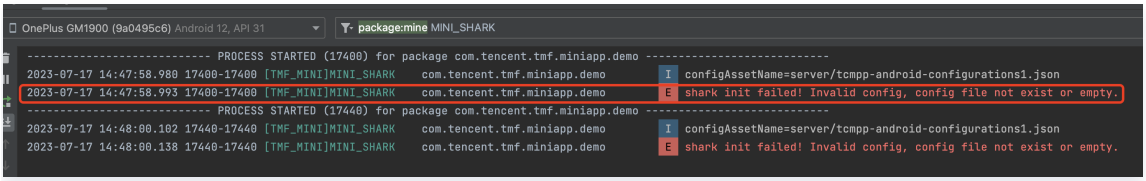
3. 如何判断初始化是否成功

使用打开小程序、扫码、搜索等小程序 API 后，SDK 内部会自动进行初始化，可通过 MINI_SHARK 过滤日志，确定初始化是否成功。

初始化成功会输出：shark init ok!



如果初始化失败会输出：shark init failed! 及具体错误原因。



初始化失败常见原因

- 原因一：配置文件路径错误，configAssetName 设置的是 assets 目录下文件的完整路径，如果配置文件在子目录下需要追加目录路径，例如：server/tcmpp-android-configurations.json。
- 原因二：不允许修改小程序配置文件内容，否则小程序无法正常运行。
- 原因三：配置文件中的 **packageName** 必须与应用的 **applicationId** 保持一致，否则 App 运行失败，因为初始化时会校验配置文件中的 **packageName**，

```
{
  "channel": "Android",
  "customId": "9ab50ac0-be06-11ec-a34e-278d66d394b5",
  "material": "01X7UBa0sAdvWstZOsTT0mdoDnZghwoG145kRt8B3i0vvTj31DVuIuXrmEZFtnnyryyJngGkBm1I4AckFtfAIs2cYkaWRawCnwteX
  "packageName": "com.tencent.tmf.miniapp.demo",
  "productId": "7686",
  "qapm": {
    "appKey": "None",
    "url": "https://tmfqapm.qq.com:30026"
  },
  "shark": {
    "appKey": "app-dfrh27n0vw",
    "asymEnc": 1,
    "httpUrl": "https://tmfmp.qq.com:30013/",
    "symEnc": 2,
    "tcpHost": "42.194.253.71",
    "tcpPort": 30014
  }
}
```

```
android {
    compileSdkVersion 30
    defaultConfig {
        applicationId "com.tencent.tmf.miniapp.demo"
        //feed back 测试
//        applicationId "tmf.tencent.tmf.demo.db"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        ndk { NdkOptions it -> abiFilters "arm64-v8a" }
```

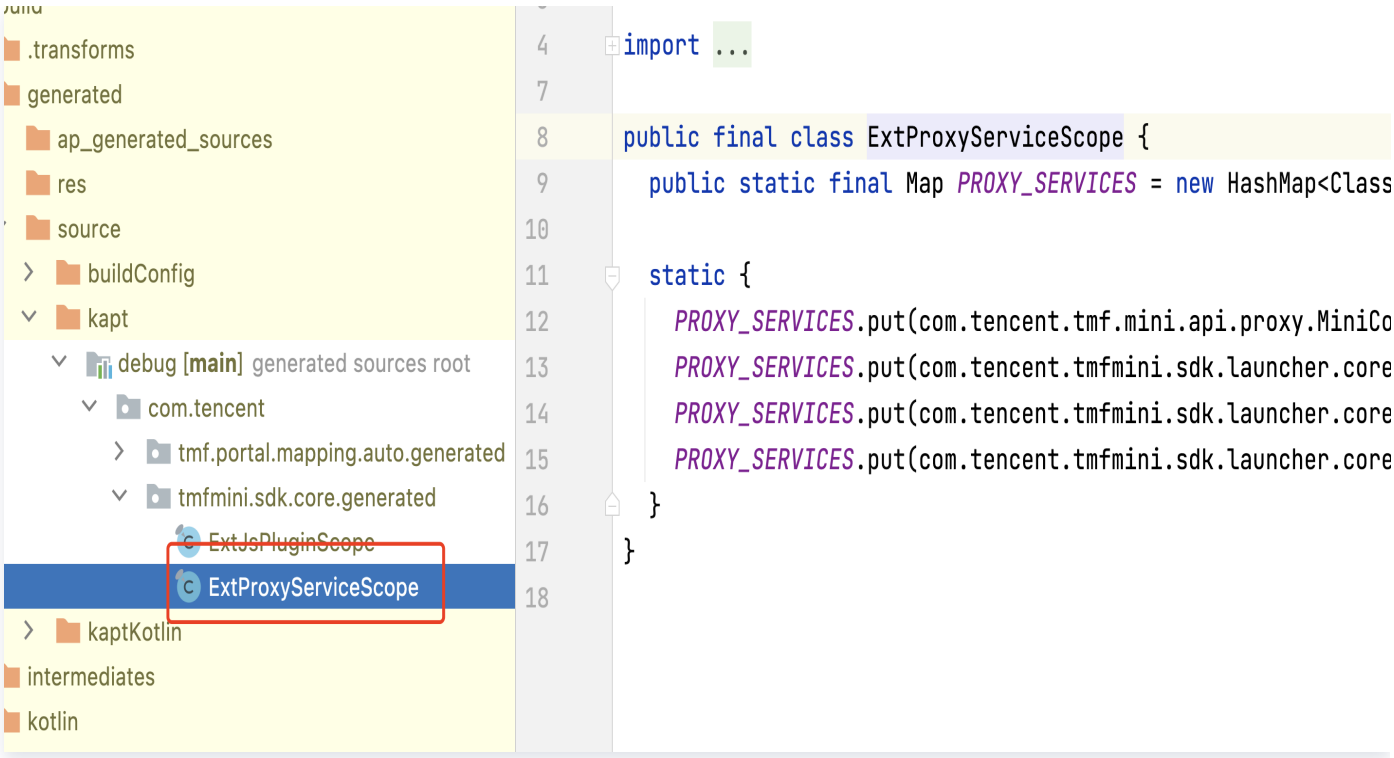
此外，也可以通过如下设置不进行包名校验：

```
MiniInitConfig config = builder
        .verifyPkg(false)//忽略包名校验
        .build();
```

原因四：MiniConfigProxy 实现类未正确生成映射关系。

确定方法如下图：

1. 是否正常生成了 ExtProxyServiceScope。
2. ExtProxyServiceScope 中是否正常生成了 MiniConfigProxy 与实现类的映射关系。



未正确生成原因：

1. app module 是否正确依赖了**mini_annotation_processor**

```
kapt 'com.tencent.tcmpp.android:mini_annotation_processor:${version}'//
版本信息请参考SDK 更新动态
```

2. MiniConfigProxy 实现类是否正确添加了@ProxyService 注解

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
}
```

4. 小程序启动失败常见原因

- 小程序与 App 未建立绑定关系。
- 小程序与 App 建立绑定关系之后未发布版本。

使用 iOS 设备接入
步骤一：获取应用SDK 配置

最近更新时间：2024-03-21 15:51:52

使用容器 SDK 需要申请配置信息，只有在 SDK 初始化 的时候配置了正确的配置文件，才能初始化成功并正常使用。

步骤一：创建应用

1. 登录腾讯云小程序平台-运营端，在左侧导航栏单击**概览**。
2. 在概览页，单击**接入应用**，在创建应用的弹窗中，填写应用名称、Bundle ID后，单击**下一步，集成小程序容器**，完成应用创建。

步骤二：获取配置文件

若要获取对应应用的配置文件，请选择对应 Bundle ID 右侧的详情操作，单击**下载**，即可下载应用配置文件到本地。

步骤二：集成 SDK

最近更新时间：2024-02-26 10:34:51

前置条件

环境要求

- iOS >= 9.0
- Xcode >= 10.0

组件依赖

- tars
- MQQTcc
- MQQComponents
- TMFSSL
- TMFShark
- TMFProfile
- SSZipArchive
- PromiseObjC
- MJRefresh
- Masonry
- SocketRocket
- Brotli
- CocoaAsyncSocket
- Lame
- TMFBaseCore
- TMFJSBridge
- TMFUploader

集成方式

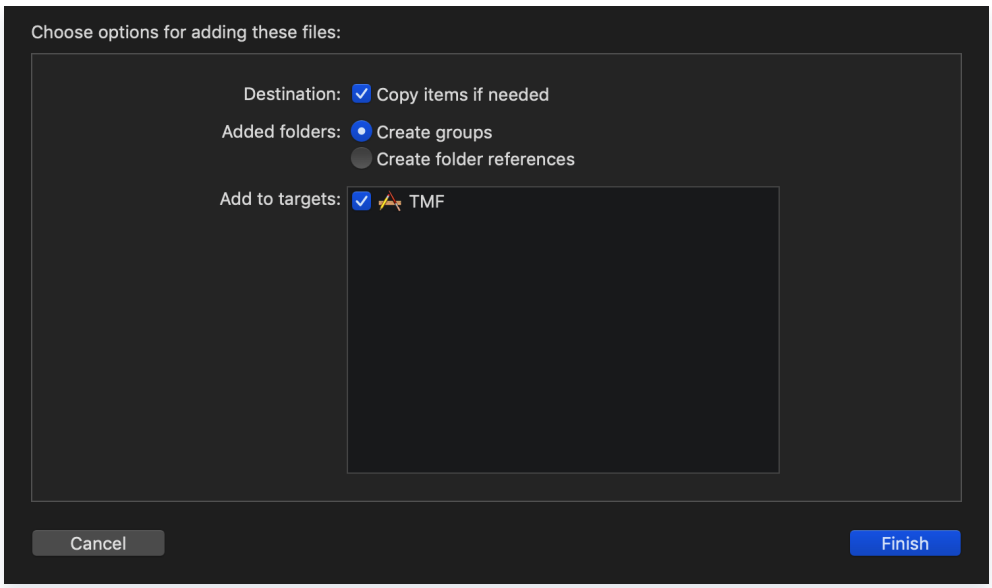
TCMPPSDK 的集成方式有以下2种，可选择其一进行集成：

- CocoaPods 集成 SDK
- 手动集成 SDK

CocoaPods 集成 SDK

1. 在您项目中的 Podfile 文件里添加 tmf 源及小程序依赖模块：

```
#TCMPP Pods 仓库
source 'https://e.coding.net/tmf-work/tmf/tmf-repo.git'

target 'YourTarget' do
    # ——— TCMPP -----
————————————
```



2. **添加依赖的 SDK**

   把 TCMPPSDK 依赖的所有组件添加到项目中，依赖的组件列表，请参见前置条件中的 组件依赖 。

3. **添加依赖的系统库**

   把 TCMPPSDK 依赖的系统库添加到项目中，在 Xcode 中打开 project 设置页，选中相关的 target，单击 **General**，在“Linked Frameworks and Libraries”中进行添加。

4. **系统库依赖如下：**
   - Foundation.framework
   - CoreTelephony.framework
   - CFNetwork.framework
   - Security.framework
   - SystemConfiguration.framework
   - CoreService.framework
   - CoreFoundation.framework
   - libz.tdb
   - libc++.tbd
   - libc.tbd
   - libz2.tbd
   - libsqlite3.0.tbd

5. **project 设置**

   添加 TCMPPSDK 后，需要进行相关的 Project 设置。在 Xcode 中打开 Project 设置页，选中相关的 target，进行以下设置：

   选择 **Build Settings** > **Linking** > **Other Linker Flags**，增加： `-ObjC` 。

   选择 **Build Settings** > **Apple Clang – Custom Compiler Flags** > **Other C Flags**，增加：

```
-fshort-wchar
-D__FIXWCHART__
```

步骤三：使用 SDK

SDK 初始化

最近更新时间：2023-12-06 15:14:26

配置文件获取

开发人员从开放平台获取对应 App 的配置文件，该配置文件是一个 json 文件，包含该 App 使用小程序开发平台的所有信息，将配置文件引入到项目中，并且作为资源设置在打包内容。具体操作，请参见 获取配置文件 。

配置信息设置

根据配置文件初始化 TMFAppletConfig 对象，并使用 TMFAppletConfig 初始化 TCMPP 小程序引擎。sdk 可以支持直接进行引擎初始化，提前准备网络链接，并更新基础库信息和配置信息，加速后小程序加载，也可以支持在需要的时候进行初始化。

参考代码：

```
//配置使用环境
    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"tcmpp-ios-configurations" ofType:@"json"];
    if(filePath) {
        TMAServerConfig *config  = [[TMAServerConfig alloc] initWithFile:filePath];
        //直接初始化
        [[TMFMiniAppSDKManager sharedInstance] setConfiguration:config];

        /*
        //准备配置信息，在需要的时候调用initServer进行初始化或者打开小程序时自动初始化
        [[TMFMiniAppSDKManager sharedInstance] prepareConfig:config];

        //需要时机进行初始化
        [[TMFMiniAppSDKManager sharedInstance] initServer];
        /*
    }
```

其它初始化动作

使用者可根据需要，设置开放接口实现实例。如果需要集成扩展模块时，初始化扩展接口准备。

```
//设置小程序引擎代理类实现
[TMFMiniAppSDKManager sharedInstance].miniAppSdkDelegate = [MIniAppDemoSDKDelegateImpl sharedInstance];
```

小程序管理 API

最近更新时间：2023-12-06 15:14:26

打开小程序

打开普通小程序

打开小程序时，会先判断本地是否有缓存的小程序。

- 如果没有，则会自动从远程服务器上下载小程序，然后打开。
- 如果有缓存的小程序，则会先打开本地小程序，同时后台校验服务器端是否有新版本。

> **说明：**
> 如果有新版本，则下载新版小程序，下次打开时，就会使用新版小程序

```
/// 通过小程序id打开小程序
/// @param appID 小程序ID
/// @param scene 场景值
/// @param firstPage 打开页面
/// @param paramsStr 带入参数
/// @param parentVC 从哪个vc呼起
/// /// @param completion 错误回调
- (void)startUpMiniAppWithAppID:(NSString *)appID
                          scene:(TMAEntryScene)scene
                      firstPage:(NSString * _Nullable)firstPage
                       paramsStr:(NSString * _Nullable)paramsStr
                      parentVC:(UIViewController *)parentVC
                    completion:(void (^)(NSError * _Nullable))completion;
```

- **options 支持的参数列表：**

名称	必须	类型	作用
appID	YES	NSString	打开指定小程序的小程序 id
scene	YES	TMAEntryScene	打开小程序使用的场景值
firstPage	NO	NSString	打开页面
paramsStr	NO	NSString	打开传递参数
parentVC	YES	UIViewController	从哪个 VC 呼起

completion	YES	block	错误回调

打开二维码小程序

TCMPP 支持识别二维码的内容打开小程序。

```
/// 通过二维码呼起小程序
/// @param qrData 二维码内容
/// @param parentVC 从哪个vc呼起
/// @param completion 错误回调
- (void)startUpMiniAppWithQrData:(NSString *)qrData
                    parentVC:(UIViewController *)parentVC
                   completion:(void (^)(NSError * _Nullable error))completion;
```

关闭小程序

关闭指定小程序

```
/// 关闭指定的小程序对象
/// @param appID 要关闭的 appID
- (void)closeMiniAppWithAppID:(NSString *)appID;
```

关闭当前小程序

```
/// 关闭当前正在运行的小程序对象
- (void)closeCurrentApplet;
```

关闭所有打开的小程序

```
///关闭所有内存中正在运行的小程序
- (void)closeAllApplications;
```

终止小程序

终止指定小程序

```
/// 终止指定的小程序对象
/// @param appID 要终止的 appID，默认删除所有版本
- (void)terminateMiniAppWithAppID:(NSString *)appID;
```

终止当前小程序

```
/// 终止当前正在运行的小程序对象
- (void)terminateCurrentApplet;
```

终止所有小程序

```
/// 终止栈上与保活中的小程序和小游戏
- (void)terminateAllApplications;
```

清理缓存

由于小程序的运行，会将小程序包和小程序信息缓存在本地，以便下次更快速的打开。如果想要将小程序的所有信息都删除，可以调用以下 API 删除某个小程序或者删除所有小程序。

```
/// 移除小程序所有相关缓存 包含资源包、基础库、小程序/小游戏沙箱数据
- (void)clearMiniAppCache;
```

清除指定小程序缓存

```
/// 删除本地缓存小程序
/// @param appID 要删除的 appID，默认删除所有版本
- (void)clearCacheWithAppID:(NSString *)appID;
```

获取小程序信息

获取当前正在运行的小程序信息

```
/// 获取当前正在运行的小程序对象
/// @return TMFAppletInfo 小程序信息
- (TMFMiniAppInfo *)currentApplet;
```

获取最近打开过的所有小程序信息

```
//获取最近打开的所有小程序信息
///@return 小程序数组<TMFMiniAppInfo>
- (NSArray *)loadAppletsFromCache;
```

搜索小程序

可以根据关键词，搜索目前小程序平台上已经发布的小程序列表。

```
/// 搜索小程序
/// @param name 搜索名称关键词
/// @param completion 搜索结果
- (void)searchAppletsWithName:(NSString *)name
                completion:(void (^)(NSArray<TMFAppletSearchInfo *> * _Nullable,
NSError * _Nullable))completion;
```

预下载小程序

提前把小程序下载到本地，可以减少初次启动小程序的耗时。

```
///预准备小程序信息
///@param appIds  要准备小程序信息
///@param isDownload 是否直接下载
///@param complete 批量更新小程序回调
- (void)preloadMiniApps:(NSArray *)appIds isDownload:(BOOL)isDownload complete:
(void (^)(NSArray * _Nullable results, NSError *_Nullable error))complete;
```

小程序引擎代理类实现

最近更新时间：2023-12-06 15:14:26

TCMPPSDK 提供一系列的开放接口，可以方便宿主 App 进行定制及使用宿主 App 已有的能力，实现步骤如下：

新建类实现 TMFMiniAppSDKDelegate 协议

```
@interface MIniAppDemoSDKDelegateImpl : NSObject <TMFMiniAppSDKDelegate>

@end
```

通过小程序管理类设置进小程序引擎

```
[TMFMiniAppSDKManager sharedInstance].miniAppSdkDelegate =
[MIniAppDemoSDKDelegateImpl sharedInstance];
```

目前已经支持可扩展的接口列表

宿主 App 相关接口

本部分内容，为 App 和小程序引擎提供运行时需要的内容，如版本、名称、语言等信息，建议实现。

```
/**
 * @brief SDK宿主应用名称
 * 主要用于文案提示使用
 */
- (NSString *)appName;

/**
 * @brief SDK宿主平台版本
 * @return 返回的是小写字符串，例如1.0.0
 */
- (NSString *)getAppVersion;

/**
 * @brief SDK宿主平台QUA
 */
- (NSString *)getAppQUA;

/**
```

```
 * @brief SDK宿主平台的网络状态
 */
- (TMANetWorkStatus)getAppNetworkStatus;

/**
 * @brief SDK宿主平台的机型信息
 */
- (NSString *)getAppIPhoneModel;


/**
 * @brief 宿主设置的当前语言
 */
- (NSString *)getCurrentLocalLanguage;

/**
 *  @brief 剪贴板频控
 */
- (NSNumber *)getClipboardInterval;

/// 定制的UA
/// @param defaultUserAgent 默认使用的 ua
- (NSString *)customUserAgent:(NSString *)defaultUserAgent;
```

用户相关 API 实现

本部分内容，为用户相关内容实现，供小程序使用，建议实现，尤其是 getAppUID。

```
/**
 * @brief SDK宿主平台的用户昵称,默认返回"TMF小程序"
 */
- (NSString *)getAppNickName;

/**
 * @brief SDK宿主平台的用户头像,默认返回demo图片
 */
- (UIImage *)getAppAvatar;


/**
 * @brief 获取SDK宿主平台的当前用户账号标识，一般填uin或openid
 *
 * 注意：返回nil会导致SDK内某些缓存失效。如果没有登录，可以填个设备号id来避免缓存失效
 */
- (NSString *_Nonnull)getAppUID;
```

开放接口列表

本部分内容需要宿主 App 实现后才可以正常使用。

```
/// 发起支付
/// @param app 小程序/小游戏实例
/// @param params 参数
/// @param completionHandler 回调结果
- (void)requestPayment:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;

/// login
/// @param app 小程序/小游戏实例
/// @param params 参数
/// @param completionHandler 回调结果
- (void)login:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;


/// checkSession
/// @param app 小程序/小游戏实例
/// @param params 参数
/// @param completionHandler 回调结果
- (void)checkSession:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;

/// getAccountInfoSync
/// @param app 小程序/小游戏实例
/// @param params 参数
/// @param completionHandler 回调结果
- (void)getAccountInfoSync:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;


/// getUserProfile
/// @param app 小程序/小游戏实例
/// @param params 参数
/// @param completionHandler 回调结果
- (void)getUserProfile:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;


/// getUserInfo
/// @param app 小程序/小游戏实例
```

```
/// @param params 参数
/// @param completionHandler 回调结果
- (void)getUserInfo:(TMFMiniAppInfo *)app params:(NSDictionary *)params
completionHandler:(MACommonCallback)completionHandler;
```

UI 相关元素

本部分内容，在 TMF 小程序引擎中已经有默认实现，宿主 App 可以根据自己的需要重新实现覆盖原默认实现。

```
/// 展示loading
/// @param infos loading信息
- (void)showLoading:(TMALoadingInfo * _Nullable)infos;


/// 隐藏loading
- (void)hideLoading;


/// 展示toast
/// @param infos toast信息
- (void)showToast:(TMAToastInfo *)infos;


/// 隐藏toast
- (void)hideToast;


/// 展示actionSheet
/// @param title 标题
/// @param cancelButtonTitle 取消按钮标题
/// @param cancelAction 点击取消按钮触发的动作
/// @param otherButtonTitleAndActions 其他按钮及响应操作
/// @param dismissBlock actionSheet收起后需要执行的操作（一定要调用以保证功能正
确！！！）
/// @param presentingViewController 展示actionSheet的vc
- (void)showActionSheetWithTitle:(nullable NSString *)title
            cancelButtonTitle:(nullable NSString *)cancelButtonTitle
                cancelAction:(nullable dispatch_block_t)cancelAction
    otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitleAndActions
                dismissBlock:(nullable dispatch_block_t)dismissBlock
      presentingViewController:(UIViewController *)presentingViewController;


/// 分享面板
```

```
/// 如果此方法不实现，则会调用
showActionSheetWithTitle:cancelButtonTitle:cancelAction:otherButtonTitleAndActions:
dismissBlock:presentingViewController:
/// @param title 标题
/// @param cancelAction 取消操作
/// @param otherButtonTitleAndActions 其他按钮及响应操作
/// @param dismissBlock 面板收起后需要执行的操作（一定要调用以保证功能正确！！！）
/// @param parentVC 呼起面板的vc
- (void)showShareViewWithTitle:(nullable NSString *)title
                 cancelAction:(nullable dispatch_block_t)cancelAction
    otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitleAndActions
                 dismissBlock:(nullable dispatch_block_t)dismissBlock
                    parentVC:(UIViewController *)parentVC;

/// 点击胶囊按钮呼起的面板
/// 如果此方法不实现，则会调用
showActionSheetWithTitle:cancelButtonTitle:cancelAction:otherButtonTitleAndActions:
dismissBlock:presentingViewController:
/// @param app 小程序信息
/// @param cancelButtonTitle 取消标题
/// @param cancelAction 取消操作
/// @param otherButtonTitleAndActions 其他按钮及响应操作
/// @param dismissBlock 面板收起后需要执行的操作（一定要调用以保证功能正确！！！）
- (void)showMoreButtonActionSheetWithApp:(TMFMiniAppInfo *)app
                       cancelButtonTitle:(nullable NSString *)cancelButtonTitle
                           cancelAction:(nullable dispatch_block_t)cancelAction
                 otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitleAndActions
                           dismissBlock:(nullable dispatch_block_t)dismissBlock;

/// 收起所有小程序/小游戏呼起的actionSheet
- (void)clearAllActionSheet;


/// 呼起alert弹框
/// @param title 标题
/// @param message 信息
/// @param actionTitleAndblocks 按钮标题及响应操作
/// @param presentingViewController 呼起alert弹框的vc
- (void)showAlertWithTitle:(nullable NSString *)title
              withMessage:(nullable NSString *)message
             actionBlocks:(nullable NSArray<AlertActionInfo *> *)actionTitleAndblocks
  presentingViewController:(UIViewController *)presentingViewController;

/**
 * @brief 展示弹窗
```

```
 *
 * @param title 标题
 * @param message 信息
 * @param actionTitleAndblocks 按钮标题及响应操作
 * @param isDismissWhenBackground 当小程序退到后台时，弹窗是否消失
 * @param dismissOnClickBlank 当点击空白背景区域时是否消失
 * @param presentingViewController 呼起alert弹框的vc
 */
- (void)showPopupWithTitle:(NSString *)title
                  message:(NSString *)message
             actionBlocks:(NSArray<AlertActionInfo *> *)actionTitleAndblocks
   isDismissWhenBackground:(BOOL)isDismissWhenBackground
      dismissOnClickBlank:(BOOL)dismissOnClickBlank
  presentingViewController:(UIViewController *)presentingViewController;



/// 宿主App可以自定义分享途径、决定展示顺序，目前使用在点击更多按钮、button组件(open-
type="share")呼起的ActionSheet中
/// 1、默认渠道：QQ好友、QQ空间、微信、朋友圈（具体type参见
MAUIDelegateShareViewType），由开发商决定，宿主App只能更改展示顺序
/// 2、自定义渠道：宿主App自定义（type填
MAUIDelegateShareViewTypeCustomizedShare）
/// 以上两种渠道展示顺序支持混排
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare;


/// 宿主App是否处于DarkMode模式
- (BOOL)isDarkMode;

/// 宿主App可以返回一个自定义的WKWebView，目前用于
/// 1. web-view组件：需要解决web-view组件中用xhr发送post请求body丢失问题
- (WKWebView *)webViewWithConfiguration:(WKWebViewConfiguration
*)configuration;
```

媒体相关内容

本部分内容，小程序引擎或者扩展引擎中有默认实现，宿主 App 可以根据自己的需要重新实现。

```
/// 扫码调用客户端的扫码模块scancode
/// @param scanPrams 扫码参数字典
/// @param navigationController 从那个页面呼起vc
/// @param completionHandler 回调结果
- (void)scanCode:(NSDictionary *)scanPrams
navigationController:(UINavigationController *)navigationController
```

```
completionHandler:(MACommonCallback)completionHandler;

/// 从选图器选择媒体
/// @param model 配置
/// @param parentVC vc
/// @param completionBlock 选择完毕后需要回传数据 根据所选类型接受
TMAPickerImageModel TMAPickerVideoModel
- (void)selectMediaFromPickerWithModel:(TMAMediaChooseConfigModel *)model
                          parentVC:(UIViewController *)parentVC
                     completionBlock:(void(^)(NSArray * _Nullable medias, NSError *
_Nullable error))completionBlock;

/// 拍摄媒体
/// @param model 配置
/// @param parentVC vc
/// @param completionBlock 选择完毕后需要回传数据，根据所选类型接受
TMACameraImageModel TMACameraVideoModel
- (void)selectMediaFromCameraWithModel:(TMAMediaChooseConfigModel *)model
                          parentVC:(UIViewController *)parentVC
                     completionBlock:(void(^)(id _Nullable media, NSError * _Nullable
error))completionBlock;

/// 从PHAsset中获取图片数据
/// @param phAsset 媒体对象
/// @param needCompress 是否需要压缩
- (NSData *)imageDataFromPhAsset:(PHAsset *)phAsset needCompress:
(BOOL)needCompress;

/// 图片预览
/// @param navigationController 呼起图片预览的导航栏
/// @param currentAbsoluteUrl 当前页面地址
/// @param absUrlsInPreviewArray 需要预览的图片
- (void)navigationController:(UINavigationController *)navigationController
  presentImageWithCurrentUrl:(NSString *)currentAbsoluteUrl
              imageUrlArray:(NSArray *)absUrlsInPreviewArray;
```

事件通知回调

本部分内容，为小程序生命周期函数中部分事件通知回调，宿主 App 可以在对应的事件发生时添加自己的处理逻辑。

```
// 处理启动过程中发生的错误
- (void)handleStartUpError:(NSError *_Nonnull)error
                     app:(NSString *_Nullable)app
                 parentVC:(UIViewController *_Nonnull)parentVC;
```

```
// 小程序启动成功
- (void)handleStartUpSuccessWithApp:(TMFMiniAppInfo *_Nonnull)app;
```

日志输出

```
/// 打印Log
/// @param level log级别，参考PLTLogLevel
/// @param msg log信息
- (void)log:(MALogLevel)level msg:(NSString *)msg;
```

自定义配置

```
/// 处理当前app对于小程序引擎的一些配置项

/// @param key 配置项key及支持的功能,参见TMAConfigDefine.h定义
/// 自定义胶囊按钮关闭图标实例
- (NSString*)stringWithConfigKey:(NSString *)key {
    if([key isEqualToString:TMA_SK_MINIAPP_CloseButton]) {
        return [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"white_close-circle.png"];
    } else if([key isEqualToString:TMA_SK_MINIAPP_CloseButtonDark]) {
        return [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"dark_close-circle.png"];
    }
    return nil;
}
```

截屏、录屏事件和添加水印

```
// 录屏状态回调
- (void)applet:(TMFMiniAppInfo *)appletInfo screenCaptureStatusChanged:
(BOOL)isCapture atPagePath:(NSString *)pagePath;

// 截屏回调
- (void)appletDidTakeScreenshot:(TMFMiniAppInfo *)appletInfo atPagePath:(NSString
*)pagePath;

// 添加水印
- (nullable UIView *)appletCustomizeWatermarkView:(TMFMiniAppInfo *)appletInfo;
```

小程序文件管理

最近更新时间：2023-11-10 15:39:29

在小程序临时目录创建文件

SDK 支持原生在小程序的缓存目录创建文件，然后返回文件的本地路径，以供宿主使用。
代码示例：

```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance]
getFileManagerWithAppID:appInfo.appId];
NSString *tmpPath = [fm createMediaTmpPathWithFileName:@"a.pdf"
type:TMATmpPathTypeFile needTimestamp:YES];
```

绝对路径转换成 wxfile 路径

SDK 支持原生在小程序的缓存目录中创建的文件路径，转换成小程序使用的文件路径返回，供小程序内部使用。
代码示例：

```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance]
getFileManagerWithAppID:appInfo.appId];
NSString *tmpPath = [fm createMediaTmpPathWithFileName:@"a.pdf"
type:TMATmpPathTypeFile needTimestamp:YES];

NSString *wxPath = [fm translateAnyPathToWxfilePath:tmpPath];
```

wxfile 路径转换为绝对路径

SDK 支持原生在小程序的缓存目录中创建的文件路径，转换成小程序使用的文件路径返回，供小程序内部使用。
代码示例：

```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance]
getFileManagerWithAppID:appInfo.appId];

NSString *filePath = [fm translateWxfilePathToAbsolutePath:wxPath];
```

版权所有：腾讯云计算（北京）有限责任公司

扩展 SDK

最近更新时间：2024-04-10 15:12:31

TCMPPSDK 引擎提供核心模块及扩展模块，方便使用者根据自己的情况进行接入。

扩展 SDK 接入及使用

扩展 SDK 是对核心 SDK 的补充，所以要使用扩展 SDK，也必须依赖核心 SDK。为了保证 SDK 的安全稳定性，将需要权限的 API 尽可能放到扩展 SDK，TCMPPSDK 引擎将 SDK 拆分为核心 SDK 与扩展 SDK，后者是前者的补充，因此使用扩展 SDK 也必须依赖核心 SDK。

TCMPPSDKExtMedia

TCMPPSDKExtMedia 提供 chooseMedia，chooseVideo，chooseImage 三个接口的默认实现，如果宿主 App 已经有对应能力，建议在开放接口中实现，如果需要使用 TMF 提供的多媒体选择插件，需要使用该插件。
使用方式：

```
pod 'TCMPPSDK'
pod 'TCMPPSDKExtMedia'
```

TCMPPSDKExtScanCode

TCMPPSDKExtScanCode 提供 wx.scanCode 的处理逻辑，如果宿主 App 本身已经有扫码识别能力，建议通过。

```
TMFMiniAppSDKDelegate.scanCode:(NSDictionary *)scanPrams navigationController:
(UINavigationController *)navigationController completionHandler:
(MACommonCallback)completionHandler;
```

对接已经正常使用的扫码模块，如果需要使用 TMF 提供的扫码功能，可以使用该插件。
使用方式：

```
pod 'ncnn'
pod 'TMFCodeDetector'
pod 'TCMPPSDK'
pod 'TCMPPSDKExtScanCode'
```

TCMPPSDKExtQMap

TCMPPSDKExtQMap 为中国大陆地区的用户提供腾讯地图 SDK 的相关能力。
使用方式：

```
pod "TCMPPSDKExtQMap"
```

在腾讯地图开放平台申请 appkey 并在代码中进行设置：

```
[TMFAppletQMapComponent setQMapApiKey:@"XXXXXXXXXXX"];
```

TCMPPSDKExtLive

如果您需要使用直播组件（live-player 和 live-pusher）进行直播推、拉流相关场景的开发，需要添加如下 SDK 以支持直播组件相关的功能的实现。

```
pod "TCMPPSDKExtLive"
pod 'TXLiteAVSDK_Professional', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_Professional.podspec'
```

除了完成以上依赖的添加，您还需要实现 TMFMiniAppSDKDelegate 中的如下方法，提供直播组件需要的 LicenseUrl 和 LicenseKey，以完成直播组件的初始化信息配置；如果您未配置正确的 LicenseUrl 和 LicenseKey，会导致直播组件功能不可用。

> **说明：**
> LicenseUrl和LicenseKey的获取方式可以参考《新增与续期 License》。

```
- (NSString *)setLiveLicenceURL {
    return @"https://xxx.license";
}

- (NSString *)setLiveLicenceKey {
    return @"xxx";
}
```

TCMPPExtAuthentication

TCMPPExtAuthentication 提供生物认证相关的能力。

集成方式：

```
pod  "TCMPPExtAuthentication"
```

使用说明：

在 info.plist 文件中添加：

- Privacy - Face ID Usage Description

API 列表：

api 名称	api 描述信息
checkIsSupportSoterAuthentication	获取本机支持的生物认证方式
checkIsSoterEnrolledInDevice	获取设备内是否录入如指纹等生物信息的接口
startSoterAuthentication	开始生物认证

TCMPPExtBLE

TCMPPExtBLE 提供低功耗蓝牙及信标相关的能力。

集成方式：

```
pod "TCMPPExtBLE"
```

使用说明：

在 info.plist 文件中添加：

- Privacy - Bluetooth Always Usage Description
- Privacy - Bluetooth Peripheral Usage Description

API列表：

api 名称	api 描述信息
蓝牙-通用	一系列的api
蓝牙-低功耗中心设备	一系列的api
蓝牙-低功耗外围设备	一系列的api
蓝牙-信标	一系列的api

TCMPPExtCalendar

TCMPPExtCalendar 提供日历相关的能力。

集成方式：

```
pod "TCMPPExtCalendar"
```

使用说明：

在info.plist文件中添加：

- Privacy - Calendars Usage Description
- Privacy - Reminders Usage Description

API列表：

api名称	api描述信息
addPhoneRepeatCalendar	向系统日历添加重复事件
addPhoneCalendar	向系统日历添加事件

TCMPPExtClipBoard

TCMPPExtClipBoard 提供系统剪切板相关的能力。

集成方式：

```
pod "TCMPPExtClipBoard"
```

API 列表：

api 名称	api 描述信息
setClipboardData	设置系统剪贴板的内容
getClipboardData	获取系统剪贴板的内容

TCMPPExtContact

TCMPPExtContact 提供日历相关的能力。

集成方式：

```
pod "TCMPPExtContact"
```

使用说明：

在 info.plist文件中添加：

- Privacy - Contacts Usage Description

API 列表：

api 名称	api 描述信息
chooseContact	选择联系人
addPhoneContact	添加手机通讯录联系人

TCMPPExtLBS

TCMPPExtLBS 提供系统定位、系统地图、罗盘、加速计、设备方向、陀螺仪相关的能力。

集成方式：

```
pod "TCMPPExtLBS"
```

使用说明：

在 info.plist 文件中添加：

- Privacy - Location Always and When In Use Usage Description
- Privacy - Location Always Usage Description
- Privacy - Location Usage Description
- Privacy - Location When In Use Usage Description
- Privacy - Motion Usage Description

API 列表：

api 名称	api 描述信息
定位	一系列的api
系统地图	一系列的api
罗盘、加速计、设备方向、陀螺仪	一系列的api

TCMPPExtMDNS

TCMPPExtMDNS 提供局域网通信的能力。

集成方式：

```
pod "TCMPPExtMDNS"
```

使用说明：

在info.plist 文件中添加：

- Privacy - Local Network Usage Description
- Privacy - Bonjour services

API 列表：

api 名称	api 描述信息
MDNS	一系列的 api

TCMPPExtNetwork

TCMPPExtNetwork 提供 TCP/UDP 通信的能力。

集成方式：

```
pod "TCMPPExtNetwork"
```

使用说明：

在 info.plist 文件中添加：

- App Transport Security Settings
- Allow Arbitrary Loads - YES
- Privacy - Bonjour services

API 列表：

api 名称	api 描述信息
TCP	一系列的 api
UDP	一系列的 api

TCMPPExtMp3Encoder

TCMPPExtMp3Encoder 提供使用 RecorderManager 时保存为 mp3格式的能力。

集成方式：

```
pod "TCMPPExtMp3Encoder"
```

使用说明：

TCMPP 提供的录音保存成 mp3格式依赖 Lame 库，Lame 库开源协议基于**GNU Library or Lesser General Public License version 2.0 (LGPLv2), GNU General Public License version 2.0 (GPLv2)** 可参考：LAME，使用者可以根据需要集成。

自定义UI

最近更新时间：2023-11-13 15:46:32

设置加载页 UI

TCMPP 小程序引擎支持宿主 App 重新定义小程序加载时的 loading 页面，来代替 sdk 内容默认的加载页，具体实现是通过实现 TMFMiniAppSDKDelegate 协议中的 customLoadingViewWithAppInfo，参考代码如下：

```
- (UIView *)customLoadingViewWithAppInfo:(TMFMiniAppInfo *)appInfo frame:
(CGRect)frame {
    UIView *view = [[UIView alloc] initWithFrame:frame];
    //todo 设置具体的 view 相关内容

    return view;
}
```

设置小程序导航栏资源

TCMPP 小程序引擎支持宿主 App 重新定义小程序导航栏资源来替换默认资源，来实现自己的风格，具体实现是通过实现 TMFMiniAppSDKDelegate 协议中的 stringWithConfigKey，目前支持以下内容的设置：

Key	描述
TMA_SK_MINIAPP_CloseButton	关闭按钮 icon
TMA_SK_MINIAPP_CloseButtonDark	关闭按钮 icon 深色主题
TMA_SK_MINIAPP_HomeButton	主页按钮 icon
TMA_SK_MINIAPP_HomeButtonDark	主页按钮 icon 深色主题
TMA_SK_MINIAPP_BackButton	返回按钮 icon
TMA_SK_MINIAPP_BackButtonDark	返回按钮 icon 深色主题
TMA_SK_MINIAPP_MoreButton	更多按钮 icon
TMA_SK_MINIAPP_MoreButtonDark	更多按钮 icon 深色主题
TMA_SK_MINIAPP_RecordButton	录音按钮 icon
TMA_SK_MINIAPP_RecordButtonDark	录音按钮 icon 深色主题

版权所有：腾讯云计算（北京）有限责任公司

TMA_SK_MINIAPP_MoreBackground	胶囊部分背景图片
TMA_SK_MINIAPP_MoreBackgroundDark	胶囊部分背景图片深色主题

参考代码如下：

```
- (NSString *)stringWithConfigKey:(NSString *)key {
    //设置浅色模式下的关闭按钮
    if([key isEqualToString:TMA_SK_MINIAPP_CloseButton]) {
        return [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"white_close-circle.png"];
    } else if([key isEqualToString:TMA_SK_MINIAPP_CloseButtonDark]) {
        return [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"dark_close-circle.png"];
    }


    return nil;
}
```

设置小程序更多菜单

TCMPP 小程序引擎支持宿主 App 重新定义小程序导航栏上更多按钮弹出胶囊的处理，包括：

1. 在胶囊页面增加自定义的菜单项

通过 TMFMiniAppSDKDelegate 协议中的 customizedConfigForShare 可以实现在胶囊视图中增加内容，增加的选项分为两类：
MAUIDelegateShareViewTypeCustomizedShare：分享类型，会走小程序内分享逻辑，在 TMFMiniAppSDKDelegate 中触发 shareMessageWithMod 实现的分享操作。
MAUIDelegateShareViewTypeCustomizedAction：自定义操作类型，可以自定义回调事件。
参考代码如下：

```
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare {
    NSMutableArray *arrays = [[NSMutableArray alloc] init];
    TMASheetItemInfo *item1 = [[TMASheetItemInfo alloc] initWithTitle:@"More
sharing" type:MAUIDelegateShareViewTypeCustomizedShare shareTarget:100
shareKey:@"my"];
    item1.icon = [UIImage imageNamed:@"icon_moreOperation_shareChat"];
    [arrays addObject:item1];

    TMASheetItemInfo *item2 = [[TMASheetItemInfo alloc] initWithTitle:@"click"
type:MAUIDelegateShareViewTypeCustomizedAction action:^(TMASheetActionParams
```

```
* _Nullable params) {
        NSLog(@"click 点击");
    }];

    item2.icon = [UIImage imageNamed:@"icon_moreOperation_collect"];
    [arrays addObject:item2];

    return arrays;
}
```

2. 自定义弹出胶囊视图

通过 TMFMiniAppSDKDelegate 协议中的 showShareViewWithTitle 可以实现胶囊视图的自定义显示。

```
/// 分享面板
/// 如果此方法不实现，则会调用
showActionSheetWithTitle:cancelButtonTitle:cancelAction:otherButtonTitleAndActions:
dismissBlock:presentingViewController:
/// @param title 标题
/// @param cancelAction 取消操作
/// @param otherButtonTitleAndActions 其他按钮及响应操作
/// @param dismissBlock 面板收起后需要执行的操作（一定要调用以保证功能正确！！！）
/// @param parentVC 呼起面板的vc
- (void)showShareViewWithTitle:(nullable NSString *)title
                cancelAction:(nullable dispatch_block_t)cancelAction
    otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitleAndActions
                dismissBlock:(nullable dispatch_block_t)dismissBlock
                   parentVC:(UIViewController *)parentVC;
```

设置小程序转场动画

通过 TMFMiniAppSDKDelegate 协议中的 getTMFSlideAnimationType 可以实现小程序启动时的转场动画，目前支持下进上出、上进下出、左进右出、右进左出和默认类型（下进下出）。

```
// 设置小程序启动时转场动画为下进上出
- (TMFSlideAnimationType)getTMFSlideAnimationType{
    return TMFSlideAnimationTypeBottomToTop;
}
```

设置小程序权限弹框

通过 TMFMiniAppSDKDelegate 协议中的 createAuthorizeAlertViewWithFrame 可以实现小程序授权窗口的自定义，参数中包括小程序及需要申请的权限，宿主可以按照自己的风格实现对应的 view 并返回 。

```
/**
 * @brief 创建自定义的授权窗口
 *
 * @param frame 窗口大小
 * @param scope 参考微信授权 scope
 * @param title 权限名称
 * @param desc 权限描述信息
 * @param privacyApi 当前调用的api
 * @param appInfo 当前小程序信息
 * @param allowBlock 允许回调
 * @param denyBlock 拒绝回调
 */

- (UIView *)createAuthorizeAlertViewWithFrame:(CGRect)frame
                        scope:(NSString *)scope
                        title:(NSString *)title
                        desc:(NSString *)desc
                        privacyApi:(NSString *)privacyApi
                       appInfo:(TMFMiniAppInfo *_Nullable)appInfo
                        allowBlock:(void (^)(void))allowBlock
                         denyBlock:(void (^)(void))denyBlock;
```

设置小程序内部 UI

TCMPP 也支持小程序内部使用的 ui 自定义显示，在 TMFMiniAppSDKDelegate 实现对应的方法即可，目前支持的内容如下：

小程序 api	TMFMiniAppSDKDelegate 方法
wx.showLoading	`- (void)showLoading:(TMALoadingInfo * _Nullable)infos;`
wx.hideLoading	`- (void)hideLoading;`
wx.showToast	`- (void)showToast:(TMAToastInfo *)infos;`

wx.hideToast	`- (void)hideToast;`
wx.showActionSheet	–
wx.showModal	`- (void)showAlertWithTitle:(nullable NSString *)title withMessage:(nullable NSString *)message actionBlocks:(nullable NSArray<AlertActionInfo*> *)actionTitleAndblocks presentingViewController: (UIViewController*)presentingViewController;`

自定义 API

最近更新时间：2023-10-20 15:19:14

TCMPP SDK 引擎提供扩展机制，允许宿主 App 自定义 API 供小程序调用。

实现步骤

1. 自定义类。
2. 类实现中引入头文件 #import "TMAExternalJSPlugin.h"。
3. 注册 api。

参考用例

```
#import "NativePluginTest.h"
#import "TMAExternalJSPlugin.h"
#import "TMFMiniAppInfo.h"

@implementation NativePluginTest

TMA_REGISTER_EXTENAL_JSPLUGIN;

//自定义同步 api
TMAExternalJSAPI_IMP(testSync) {
    TMFMiniAppInfo *appInfo = context.tmfAppInfo;
    NSDictionary *data = params[@"data"];

    NSLog(@"************ invokeNativePlugin testSync,appId:%@,data is
%@",appInfo.appId, data);

    TMAExternalJSPluginResult *pluginResult = [TMAExternalJSPluginResult new];
    pluginResult.result = @{};
    return pluginResult;
}

TMAExternalJSAPI_IMP(test) {
    TMFMiniAppInfo *appInfo = context.tmfAppInfo;
    NSDictionary *data = params[@"data"];

    NSLog(@"************ invokeNativePlugin test,appId:%@,data is
%@",appInfo.appId, data);

    //异步处理，在异步回调中把结果返回给小程序
    //{
```

```
    //  TMAExternalJSPluginResult *pluginResult = [TMAExternalJSPluginResult new];
    //  pluginResult.result = @{@"result" : result.data};
    // [context doCallback:pluginResult];
    // }

    return nil;
}

@end
```

小程序中使用。

```
//异步api调用
var opts = {
  api_name: 'test',
  success: function(res) {},
  fail: function(res) {},
  complete: function(res) {},
  data: { // 入参
    name : 'kka',
    age : 22
  }
 }
 wx.invokeNativePlugin(opts);

//同步api调用
var opts = {
  api_name: 'testSync',
  sync:true
}
var rst = wx.invokeNativePlugin(opts);
```

进阶使用

自定义 api 支持在终端 app 配置文件的方式进行配置，在小程序中通过直接调用wx.api的方式来调用。

1. 将 app 中实现的配置文件统一放在tcmpp-custom-config.json中，并将tcmpp-custom-config.json文件设置打包到 app 中，参考内容如下：

```
{
   "extApi":[{
         "name": "test",
         "sync": false,
         "params": {
```

```
            "data": ""
        }
    },
    {
        "name": "testSync",
        "sync": true,
        "params": {
            "name": "",
            "title": ""
        }
    }
  ]
}
```

2. 小程序中调用。

```
//异步api调用
var opts = {
   success: function(res) {},
   fail: function(res) {},
   complete: function(res) {},
   data: {
     name : 'kka',
     age : 22
   }
}
wx.test(opts);

//同步api调用
var rst = testSync(opts);
```

自定义原生组件

最近更新时间：2023-12-21 17:06:33

使用自定义原生组件

TCMPP 支持自定义 UI 组件，小程序端的自定义组件为 <external-element>。如果您想使用自定义 UI 组件，请遵循以下步骤：

1. 客户端接入 SDK 后新建继承自 UIView 的类，例如 QMATestView，导入"TMAExternalJSPlugin.h"文件，使 QMATestView 遵循“TMAExternalElementView”协议。
2. 调用 TMARegisterExternalElement 方法，注册 QMATestView 类为 maTestView。
3. 实现“TMAExternalElementView”协议中的 createWithParams 和 operateWithParams 方法。

```
#import "QMATestView.h"
#import "TMAExternalJSPlugin.h"

@interface QMATestView () <TMAExternalElementView>

@end

@implementation QMATestView {
    UILabel *_textLabel;
    UIButton *_clickButton;
    id<TMAExternalJSContextProtocol> _context;
}

TMARegisterExternalElement(maTestView);
+ (UIView *)createWithParams:(NSDictionary *)params context:
(id<TMAExternalJSContextProtocol>)context {
    QMATestView *testView = [[QMATestView alloc] initWithFrame:CGRectZero];
    NSDictionary *testViewParams = QQ_Dict_DictValue(params, @"params");
    [testView setText:QQ_Dict_StringValue(testViewParams, @"text")];
    testView->_context = context;
    return testView;
}

//接收小程序端的调用事件进行处理
- (void)operateWithParams:(NSDictionary *)param context:
(id<TMAExternalJSContextProtocol>)context {
    NSDictionary *data = QQ_Dict_DictValue(param, @"data");
    NSDictionary *params1 = QQ_Dict_DictValue(data, @"params1");
    NSInteger age = [QQ_Dict_NumberValue(params1, @"age") integerValue];
    NSString *name = QQ_Dict_StringValue(params1, @"name");
```

```
    qq_weakify(self);
    [MAUtils executeOnThread:[NSThread mainThread] block:^{
      qq_strongify(self);
      if (self) {
        self->_textLabel.text = [NSString stringWithFormat:@"name = %@ , age =
%ld",name,(long)age];
        // 把结果返回给小程序端
        TMAExternalJSPluginResult *result = [TMAExternalJSPluginResult new];
        result.result = @{@"result":@"success"};
        [context doCallback:result];
      }
    }];
}
```

发送事件给小程序端

在自定义原生组件里如果想发送事件给小程序端，先在 createWithParams:context: 方法中记录 context：

```
_context = context;
```

发送事件时这样写：

```
- (void)onClickButton:(UIButton *)button {
    _textLabel.text = @"What do you want me to do";
    // 组装数据 发送事件
    NSString *data = [MAUtils
JSONStringify:@{@"externalElementId":_elementId,@"type": @"elvisgao callback"}];
    [_context doSubscribe:kTMAOnExternalElementEvent data:data];
}
```

小程序端使用：

1. 在小程序 wxml 中引入：

```
<external-element
    id="comp1"
    type="maTestView"
    _insert2WebLayer
    style="width: 200px;height: 100px;"
    bindexternalelementevent="handleEvent"
></external-element>
```

> **说明：**
> type 需与 native 端约定，如非同层则不设置 _insert2WebLayer 属性。

bindexternalelementevent 可监听 native 传递的操作，回调参数包括：

```
{
target,
currentTarget,
timeStamp,
touches,
detail, // native传递的参数
}
```

2. 小程序 js 中创建实例。

```
this.ctx = wx.createExternalElementContext('comp1');
```

> **说明：**
> wx.createExternalElementContext 参数为 wxml 中元素 id。

3. 在小程序中调用实例方法：

```
this.ctx.call({
params1: {
name: 'name1',
age: 11
},
params2: {
name: 'name2',
age: 22
},
success: (e) => {
console.log('====operate success=====', e)
},
fail: (e) => {
console.log('====operate fail=====', e)
},
complete: (e) => {
console.log('====operate complete=====', e)
```

```
}
})
```

说明：
实例提供 call 方法，success、fail、complete 回调；通过参数区分调用方法，基础库会调用 invoke 方法透传参数到 native。

常见问题

最近更新时间：2023-12-28 11:32:51

xcode15 上编译后，在 ios14 及以下设备上出现崩溃问题

在项目编译选项中添加 -Wl,-ld_classic 配置。

Xcode 15 RC Release Notes

Linking

New Features

A new linker has been written to significantly speed up static linking. It's the default for all macOS, iOS, tvOS and visionOS binaries and anyone using the "Mergeable Libraries" feature. The classic linker can still be explicitly requested using `-ld64`, and will be removed in a future release. (108915312)

Known Issues

Binaries using symbols with a weak definition crash at runtime on iOS 14/macOS 12 or older. This impacts primarily C++ projects due to their extensive use of weak symbols. (114813650) (FB13097713)

Workaround: Bump the minimum deployment target to iOS 15, macOS 12, watchOS 8 or tvOS 15, or add `-Wl,-ld_classic` to the `OTHER_LDFLAGS` build setting.