云地图服务

开发指南

文档版本	01
发布日期	2025-01-20

华为云计算技术有限公司

商标声明


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

注意

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

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

目 录

1 WebARSDK 使用手册

1.1 开发概述

WebARSDK 简介

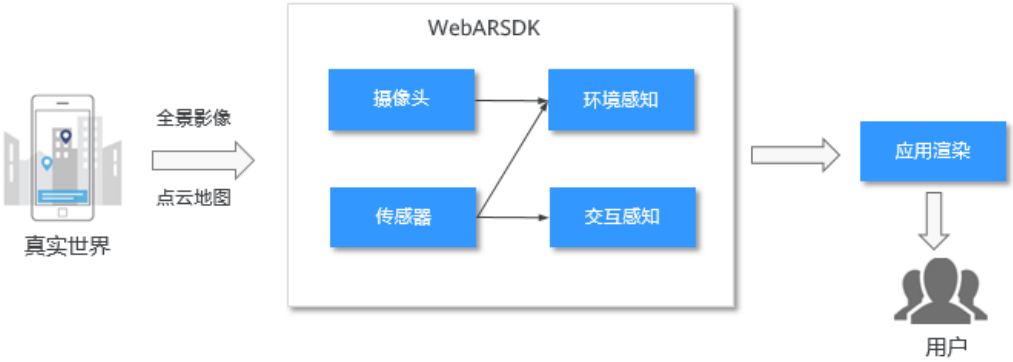
WebARSDK是一款轻量化JS-SDK，支持租户通过SDK快速集成和使用KooMap AR地图运行服务，开放能力的内存增加不大于3MB，运行时不依赖于Native App，可通过主流Web浏览器平台、主流终端设备的App内嵌WebView，实现WebAR体验。

WebARSDK提供的API，可以将用户、空间及数字内容连接在一起 ，达到厘米级的定位能力和1度以内的定姿能力，实现随时随地高精度的空间计算。在AR地图覆盖区域，用户可进行3D实景步行导引，无需担心GPS信号弱的环境。

工作原理

WebARSDK通过设备传入的图像及传感器数据，调用视觉定位（VPS）和导航服务，以及SLAM（同步定位与地图构建）算法，为您的应用提供环境感知与交互感知能力，再经应用渲染给用户呈现虚实融合体验。

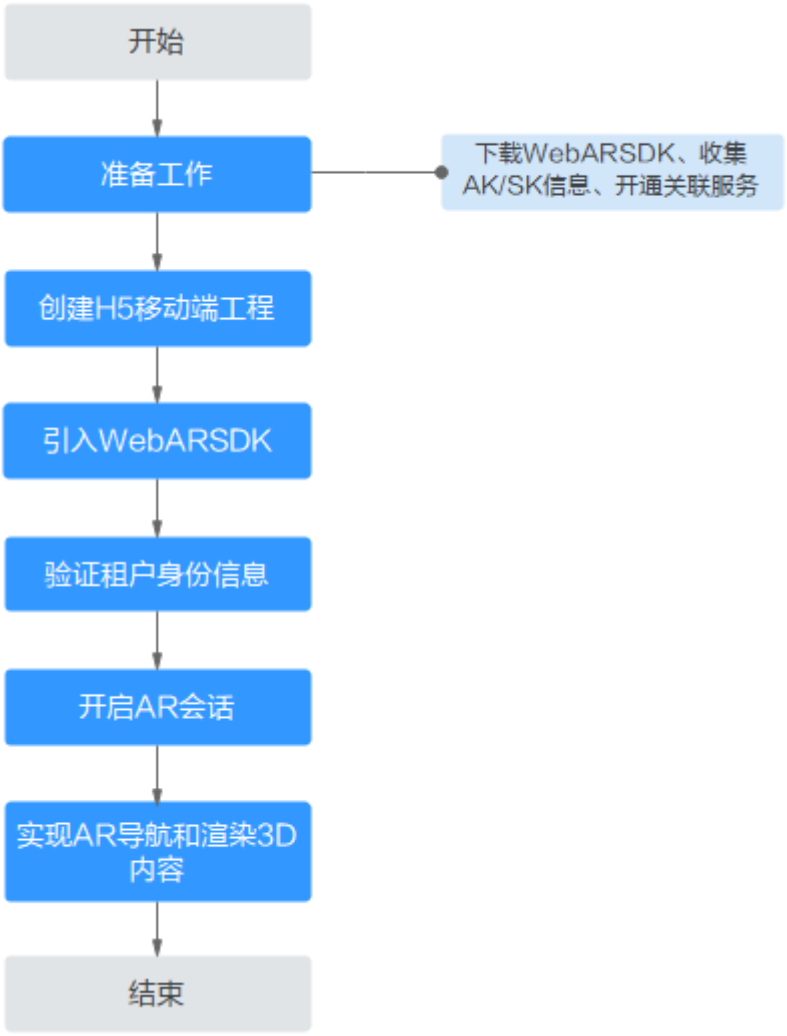
图 1-1 WebARSDK 工作原理



1.2 总体开发思路

使用WebARSDK开发移动端应用的工作流程如图1-2所示。

图 1-2 工作流程



1. 准备工作。

   下载WebARSDK、**收集AK/SK信息**、**开通关联服务**。

2. 创建H5移动端工程。

   使用原生或者流行框架（如Vue、React）创建H5移动端工程。

3. 引入WebARSDK文件。

   下载、解压WebARSDK包，并将**下载WebARSDK**包放入工程目录，通过外部引入脚本的方式**引入WebARSDK文件**。

4. 验证租户身份信息。

   通过**hwar.setAKSK方法**将**AK/SK**传入SDK，验证租户身份信息。

5. 开启AR会话。

   获取图像及传感器数据，初始化SLAM，准备相关环境。

6. 实现AR导航和渲染3D内容。

   依靠视觉定位，获取当前设备位置，以及请求导航路径，实现空间位置追踪导航；开发者基于SDK输出的图像数据、相机矩阵、路径信息，实现3D数字内容渲染。

1.3 准备工作

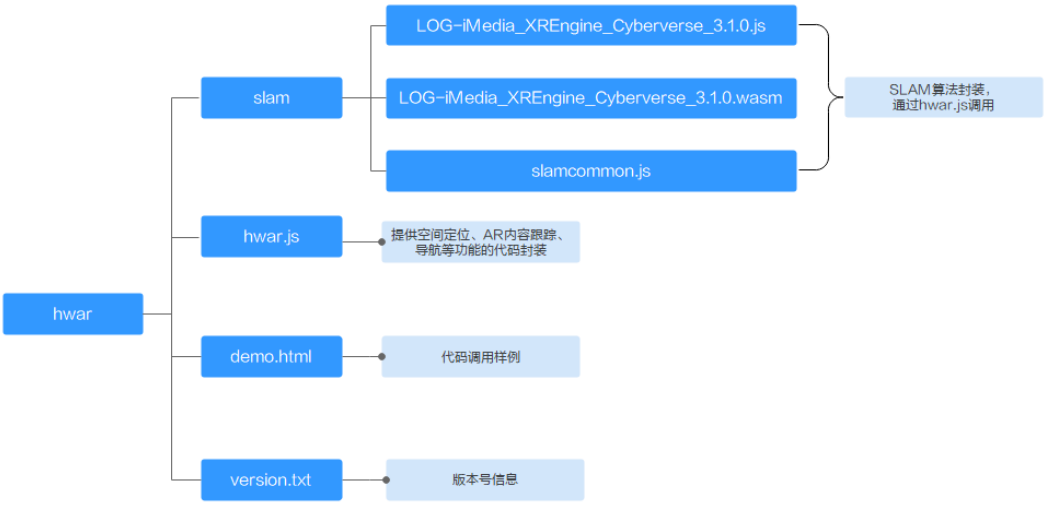
开发技能要求

- 具备TypeScript/JavaScript开发基础。
- 熟悉Web移动端开发。
- 熟悉Three.js、LayaAir等渲染库，可实现基于WebGL的3D渲染。

下载 SDK

请下载**WebARSDK软件包**和**软件包的完整性校验文件**，并解压软件包、核对文件目录。

图 1-3 SDK 包目录结构



收集信息

表 1-1 收集信息

信息项	说明
AK/SK	访问密钥。包含访问密钥ID（Access Key ID，AK）和秘密访问密钥（Secret Access Key，SK）两部分，是您在华为云的长期身份凭证。华为云通过AK识别访问用户的身份，通过SK对请求数据进行签名验证，用于确保请求的机密性、完整性和请求者身份的正确性。 获取方法请参见**访问密钥**。

开通关联服务

开通AR地图运行服务。

说明

AR地图运行服务可在**KooMap管理控制台**开通。

环境要求

- 已安装配置NodeJS环境。
- 已安装代码编辑器，如Visual Studio Code。
- 已安装Web服务器，如http-server、VSCode插件live-server、Tomcat等。
- Windows系统电脑已安装Chrome浏览器或基于Chromium内核的浏览器（例如Edge）；Mac系统电脑已安装Safari浏览器。
- 移动设备已配置后置摄像头、陀螺仪、加速度传感器、GPS芯片等器件。
- 移动设备已安装支持WebRTC浏览器，如Chrome、Safari、华为浏览器等。

1.4 快速开始

以工程Project为例，通过引入SDK脚本、传入AK/SK、调用API，验证工程中SDK是否引入正常，为后续功能开发做准备。

操作步骤

步骤1 创建工程（Project），并将WebARSDK包解压后放入工程目录。

```
Project
├── index.html
├── src
│   ├── index.js
├── hwar
│   ├── slam
│   │   └── LOG-iMedia_XREngine_Cyberverse_3.1.0.js
│   │   └── LOG-iMedia_XREngine_Cyberverse_3.1.0.wasm
│   │   └── slamcommon.js
│   ├── hwar.js
```

步骤2 通过外部脚本引入WebARSDK的JS文件。

```
<script src="hwar/slam/LOG-iMedia_XREngine_Cyberverse_3.1.0.js"></script>
<script src="hwar/slam/slamcommon.js"></script>
<script src="hwar/hwar.js"></script>
```

步骤3 传入AK/SK。

```
// 认证用的AK、SK硬编码到代码中或明文存储都有很大的安全风险，建议在代码中配置加密后的AK/SK，解密后传入hwar.setAKSK方法中。
hwar.setAKSK({
    AK:"解密后的变量",
    SK:"解密后的变量"
});
```

步骤4 调用API（以“hwar.searchArea”）验证引入效果。

```
// 根据GPS判断地图服务区域
hwar.searchArea({
    "location": { "lon": xxx, "lat": xxx },
    "radius": 50
}).then((res) => {
    console.log("根据GPS获取局点：" + JSON.stringify(res));
});
```

步骤5 查看控制台输出，如果能够打印区域信息，则说明引入成功。

----结束

后续步骤

创建H5工程后，您可以继续开发AR功能。

- 开启AR会话，具体请参见**开启AR会话**。
- 实现AR导航，具体请参见**实现AR导航**。

1.5 开启 AR 会话

1.5.1 概述

您可开启AR会话获取设备的图像、GPS坐标及传感器数据，启动SLAM从而进行视觉定位。

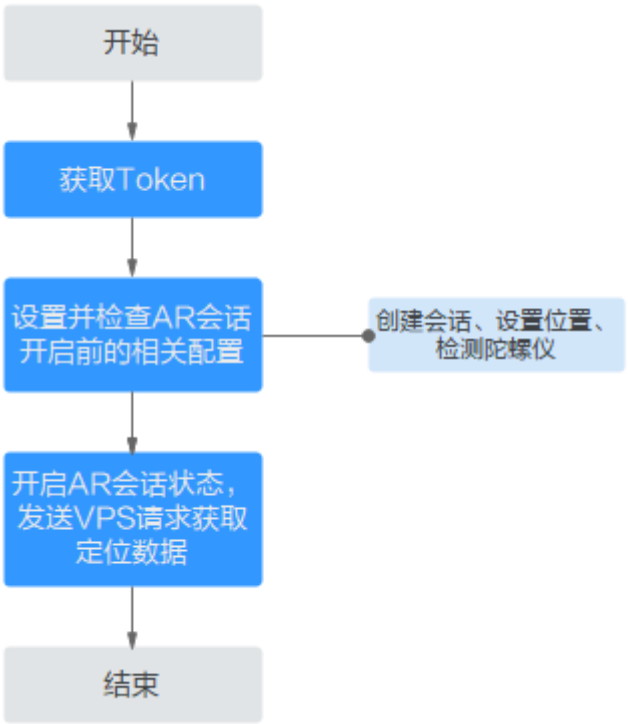
说明

传感器包括陀螺仪、加速度计、磁力计。

1.5.2 开发指导

开发流程

图 1-4 开启 AR 会话开发流程



开发过程

步骤1 获取Token。

成功获取Token后才能进行定位请求。

```
// 向服务器请求签名
hwar.getToken().then((res) => {
    // 获取成功
}).catch((err) => {
    // 获取失败
});
```

步骤2 设置并检查AR会话开启前的相关配置。

配置操作包括创建会话、设置位置、检测陀螺仪。

说明

- 创建会话：用于启动相机及SLAM。
- 设置位置：用于设置已知地图服务区域的GPS坐标，提高后续视觉定位的精度。
- 检测陀螺仪：用于判断设备陀螺仪的有效性。

```
// 并发异步请求
Promise.all([hwar.createARSession(),
    setAreaGPSFun({ latitude: xxx, longitude: xxx, altitude: xxx }),
    checkGyro()
]).then(([res1, res2, isCheckGyro]) => {
    if (isCheckGyro) {
        // 会话建立成功
    } else {
        // 该机型暂不支持
    }
}).catch((err) => {
    // 根据错误信息err.message的不同取值，处理不同事件。
});

// 设置已知地图服务区域的GPS坐标
function setAreaGPSFun(gps) {
    return new Promise((resolve, reject) => {
        hwar.setAreaGPS(gps);
        resolve(true);
    })
}

// 检测陀螺仪有效性，evt.data.gyro值为true/false
function checkGyro() {
    return new Promise((resolve, reject) => {
        hwar.addEventListener("GYRO_STATUS", (evt) => {
            resolve(evt.data.gyro);
        });
    })
}
```

对于并发异步请求捕获的错误信息（err.message），这里仅提供创建会话（hwar.createARSession）的错误信息，如**表1-2**所示。您可以对并发请求中的其他Promise对象抛出自定义错误信息，用于处理不同情况下的异常事件。

表 1-2 hwar.createARSession 中错误信息说明

错误信息	说明
SLAM_INIT_FAIL	SLAM启动失败。
PHONE_TYPE_NOT_SURPORT	设备机型暂不支持。

步骤3 开启AR会话状态，发送VPS请求获取定位数据。

通过回调函数（hwar.registerGetNewVpsPose）获取定位数据，如果获取失败，可通过侦听事件“VPS_FAIL”处理。


```
// 设置AR状态，第一个参数设为true表示开始发送VPS请求
hwar.setARStatus(true, true, true);

// 定位成功回调函数
hwar.registerGetNewVpsPose((dictPoseData, vpsOffset) => {
    // 获取定位成功数据
});

// 定位失败事件侦听
hwar.addEventListener("VPS_FAIL", () => {
    // 处理失败逻辑
});
```

----结束

1.5.3 注意事项

视频流无法正常获取原因排查

如果您无法获取视频流，建议您排查以下问题：

- 检查是否存在浏览器多个标签页同时申请获取视频流的情况，建议您关闭其他标签页。
- 检查浏览器是否支持WebRTC，如不支持，建议您更换浏览器。
- 检查应用相机权限是否打开。
- 检查移动设备后置摄像头是否能正常拍摄画面。

无法进行视觉定位原因排查

如果您无法进行视觉定位，建议您排查以下问题：

- 检查是否竖屏正向手持设备。
  - 侦听事件“LANDSCAPE_BY_ROLL”检查当前屏幕是否竖屏。
  - 侦听事件“ORIENTATION_NORMAL”检查屏幕正向状态。

  说明

  验证视觉定位效果时，您需竖屏正向手持设备扫描周边环境。
- 检查定位区域是否在地图服务范围内。

  按“F12”打开DevTools，在“Network”页签中查看“vps”的状态码。如果状态码为“400”，请在“Response”页签中查看“error_msg”内容，如显示“access denied, out of your service zone”，则表示位置不在您的地图服务范围。

1.6 实现 AR 导航

1.6.1 概述

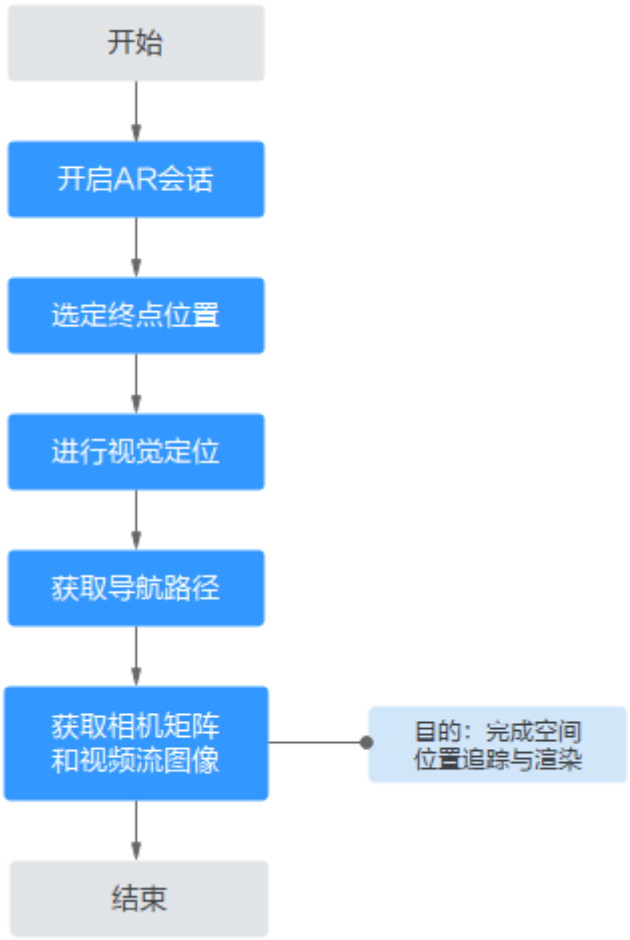
AR导航功能依赖视觉定位，提供空间位置追踪能力。

您可以依赖图像数据、相机矩阵及路径点信息进行3D导航路径渲染。

1.6.2 开发指导

开发流程

图 1-5 AR 导航启动流程



开发过程

步骤1 **开启AR会话**。

步骤2 选定终点位置。

通过POI智能搜索（hwar.searchPoi）选定终点坐标。

```
// POI智能搜索
hwar.searchPoi({
    "floor": "1F",
    "address": "xxx",
    "title": "xxx",
    "pageNum": 0,
    "pageSize": 50
}).then((res) => {
    // POI信息列表，包含POI的UTM坐标等信息
});
```

步骤3 进行视觉定位。

通过设置AR会话状态（hwar.setARStatus）或者触发手动定位（hwar.requestVps）发送定位请求，定位成功结果通过回调函数返回。

```
// 触发手动定位请求
hwar.requestVps();

// 设置定位成功回调函数
hwar.registerGetNewVpsPose((dictPoseData, vpsOffset) => {
    // 可计算与上次定位的差值、与终点的距离。当有3D内容时可重置世界坐标原点重新渲染内容
});
```

步骤4 获取导航路径。

传入起始点与终点的UTM坐标请求导航（hwar.requestNavi），获取该段路程的路径点。

```
// 请求导航路径
hwar.requestNavi("xxx",
    [xxx, xxx, xxx],
    [xxx, xxx, xxx],
    1
).then((res) => {
    // 获取导航路径点信息
}).catch((err) => {
    // 获取失败
})

// 路径点结构：其中xyz是UTM坐标，渲染路径时需要转换成渲染坐标（TS语言）
export interface INaviPoint {
    linkID: number;
    x: number;
    y: number;
    z: number;
    floor: string;
    building: string;
    status: number;
}
```

步骤5 获取相机矩阵和视频流图像，完成空间位置追踪与渲染。

通过回调函数“hwar.registerRefreshCameraMat”获取相机矩阵和视频流图像。当步骤3定位成功后输出相机矩阵，包含当前虚拟相机的位姿信息。视频流图像可通过Three.js、LayaAir等WebGL渲染引擎渲染成背景画面。

说明

- 您可以根据相机矩阵、视频流图像及路径点，自定义渲染导航路径、虚拟数字内容。
- 自定义渲染导航路径时，建议您分段处理路径，及时移除失效路径点。渲染的每帧仅计算当前相机的位置和距离相机最近的点，且仅渲染当前路段。

```
// 设置接收相机矩阵的回调函数
hwar.registerRefreshCameraMat((cameraMat, imgData) => {
    if (cameraMat) {
        //VPS定位成功，输出相机矩阵
        console.log(`相机矩阵：${cameraMat}`);
    }
    // 图像预览流默认宽:高=480:640，不同尺寸屏幕请自行适配
    console.log(`接收到视频流图像：${imgData}`);
});

//Three.js r144版本 相机渲染代码示例（TS语言）
//图像作为3D场景的背景,渲染器的设置如下：
this.renderer = new THREE.WebGLRenderer({ alpha: true, preserveDrawingBuffer: true, antialias: true });
this.renderer.autoClearColor = true;
this.renderer.sortObjects = false;
this.renderer.setPixelRatio(window.devicePixelRatio);
this.renderer.setSize(window.innerWidth, window.innerHeight);
this.renderer.localClippingEnabled = true;
```



```
this.renderer.autoClear = false;
this.renderer.debug.checkShaderErrors = false;
document.body.appendChild(this.renderer.domElement);

public updateARCameraMatrix(cameraMat: THREE.Matrix4, imgData: ImageData): void {
    if (cameraMat) {
        this.camera.matrix.fromArray(cameraMat.toArray());
        this.camera.updateMatrixWorld(true);
    }
    if (imgData) {
        let buffer: Uint8Array = new Uint8Array(imgData.data.buffer);
        let tex: DataTexture = new DataTexture(buffer,
            imgData.width,
            imgData.height,
            THREE.RGBAFormat
        );
        tex.needsUpdate = true;
        tex.matrixAutoUpdate = false;
        tex.magFilter = THREE.LinearFilter;
        tex.flipY = true;
        tex.encoding = THREE.sRGBEncoding;
        this.scene.background = tex;
    }
}

//LayaAir 2.12版本 相机渲染代码示例（TS语言）
//图像作为舞台的背景，将video: Laya.Sprite添加到舞台，添加3D场景，并将场景相机设置如下：
Laya.stage.addChild(this.video);
this.mainScene.mouseThrough = false;
this.mainCamera = this.mainScene.getChildByName("Main Camera") as Laya.Camera;
this.mainCamera.enableHDR = false;
this.mainCamera.clearColor = new Laya.Vector4(0, 0, 0, 255);
this.mainCamera.clearFlag = Laya.CameraClearFlags.DepthOnly;
this.mainCamera.fieldOfView = 66;
this.mainCamera.depthTextureMode = Laya.DepthTextureMode.None;
Laya.stage.addChild(this.mainScene);

public updateARCameraMatrix(cameraMat: { elements: Float32Array, isMatrix4: boolean }, imgData:
ImageData): void {
    if (cameraMat) {
        let mat: Laya.Matrix4x4 = new Laya.Matrix4x4();
        mat.elements = cameraMat.elements;
        this.mainCamera.transform.worldMatrix = mat;
    }
    if (imgData) {
         if (this.stageHeight !== Laya.stage.height) {
            // 请自行对渲染尺寸(宽:高=480:640)做屏幕适配
            this.onResize();
         }
         if (this.videoTexture && this.videoTexture.bitmap) {
            this.videoTexture.disposeBitmap();
            this.videoTexture2D.destroy();
            this.videoTexture = null;
            this.videoTexture2D = null;
         }
         let buffer: Uint8Array = new Uint8Array(imgData.data.buffer);
         this.videoTexture2D = new Laya.Texture2D(imgData.width, imgData.height);
         this.videoTexture2D.setPixels(buffer);
         this.videoTexture = Laya.Texture.create(this.videoTexture2D, 0, 0, imgData.width, imgData.height);
         this.video.texture = this.videoTexture;
    }
}
```

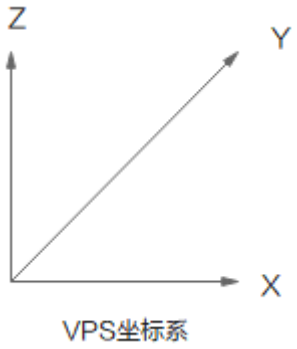
----结束

1.6.3 注意事项

了解坐标系

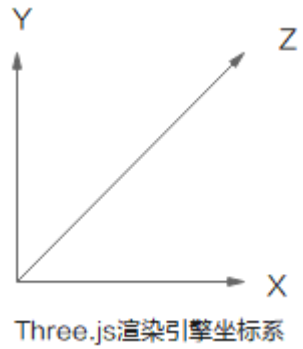
- VPS坐标系
  VPS坐标系是基于UTM坐标定义的，北向Y轴正向和东向X轴正向如图1-6所示。

  图 1-6 VPS 坐标系示意图

- Three.js渲染引擎坐标系
  摆放虚拟物体时需要注意：Three.js的相机朝向是Z轴的反向。

  图 1-7 Three.js 渲染引擎坐标系示意图

坐标系转换

SDK提供了UTM坐标与渲染引擎（如Three.js、LayaAir）坐标之间的转换方法。

```
// UTM坐标转成渲染坐标
let vecInit = hwar.utm2Render([utm.x, utm.y, utm.z]);

// 渲染坐标转成UTM坐标
let renderPosUtm = hwar.render2Utm([render.x, render.y, render.z]);
```

1.7 API 列表

1.7.1 调用前须知

调用API前，您需提前了解如下内容：

- WebARSDK封装类名为“hwar”，所有API均直接通过类“hwar”来调用，无需实例化。
- 下文API中的GPS坐标系为WGS84坐标系。
- utmCode指UTM（Universal Transverse Mercator Grid System，通用横墨卡托格网系统）投影带号。
- 图像流约束为宽:高=480:640，您需根据实际情况自行适配设备屏幕尺寸。
- 参数类型使用了TypeScript语言的声明规范。

1.7.2 视觉定位

接口列表

视觉定位需要使用的接口如表1-3所示。

表 1-3 视觉定位接口列表

接口	描述	参数名	参数类型	参数说明	返回值
getToken	向服务器请求签名。	-	-	-	Promise<any>

setAKSK	验证租户身份信息。	requestParams	{
			AK: string;
			SK: string;
			}
				访问密钥ID（AK）和秘密访问密钥（SK）。	boolean
					如果AK/SK有一个是空值就会返回false。

setAreaGPS	设置已知地图服务区域的GPS坐标到AR系统中，便于提高后续视觉定位的精度。GPS信息可以从局点配置中获取，也可以通过请求浏览器的位置接口获取。	gps	{
			latitude: number;
			longitude: number;
			altitude: number;
			}
				GPS坐标的经纬度及海拔高度。	void

接口	描述	参数名	参数类型	参数说明	返回值

searchArea	根据GPS判断地图服务区域。	areaData	{
			location: {
			lon: number;
			lat: number;
			};
			radius: number;
			}
				GPS经纬度坐标和查询半径（单位：米）。	Promise<any>
					返回查询范围内的局点信息。

getAreaConfig	获取区域配置信息。	address	string	局点名称。	Promise<any>
convertUTMtoGPS	UTM转GPS。	utm	number[]	utm坐标，[x, y, z]。	number[]
		isSouthHemi	boolean	是否在南半球。	
		zone	number	utmCode的数值部分。定位成功会返回utmCode。	

convertGPStoUTM	GPS坐标转UTM XYZ。	gps	{
			longitude: number;
			latitude: number;
			altitude?: number;
			}
				GPS坐标。	{
					utmX: number;
					utmY: number;
					utmZ: number;
					utmCode: string;
					}

接口	描述	参数名	参数类型	参数说明	返回值
registerGetNewVpsPose	设置定位成功回调函数。	funGetNewPose	Function	回调函数。 需要接收两个参数： VPS请求接口的返回数据和世界坐标原点。 ● dictPoseData: { utmcode: string; vpsFloor: string; pose: { r: number[], t: number[]} } – dictPoseData.pose.r是VPS姿态矩阵 – dictPoseData.pose.t是当前的位置的UTM坐标 ● vpsOffset: Vector3	void
requestVps	触发手动定位请求。定位结果会通过回调函数返回。	-	-	-	void

接口调用示例

```
// 向服务器请求签名
hwar.getToken().then((res) => {
    // 进行后续操作（无需获取具体签名值）
}).catch((err) => {
    // 获取失败
});

// 验证租户身份信息
// 认证用的AK、SK硬编码到代码中或明文存储都有很大的安全风险，建议在代码中配置加密后的AK/SK，解密后
```



```
传入hwar.setAKSK方法中。
let result:boolean = hwar.setAKSK({
    AK: "解密后的变量",
    SK: "解密后的变量"
});

// 设置已知地图服务区域的GPS坐标
hwar.setAreaGPS({
    latitude:xxx,
    longitude:xxx,
    altitude:xxx
});

// 根据GPS判断地图服务区域
hwar.searchArea({
    "location": { "lon": xxx, "lat": xxx },
    "radius": 50
}).then((res) => {
    // 查询范围内的局点信息
});

// 获取区域配置信息
hwar.getAreaConfig("xxx").then((data) => { });

// UTM 转 GPS
let gps = hwar.convertUTMtoGPS([xxx, xxx, xxx], false, xxx);

// GPS 坐标 转 UTM XYZ
let utm = hwar.convertGPStoUTM({
    latitude: xxx,
    longitude: xxx,
    altitude: xxx
});

// 设置定位成功回调函数
hwar.registerGetNewVpsPose((dictPoseData: any, vpsOffset: Vector3) => { });

// 触发手动定位请求
hwar.requestVps();
```

1.7.3 空间位置追踪与渲染

接口列表

空间位置追踪与渲染需要使用到的接口如表1-4所示。

表 1-4 空间位置追踪与渲染接口列表

接口	描述	参数名	参数类型	参数说明	返回值
createARSession	创建AR会话。	-	-	-	Promise<void>
setARStatus	设置AR状态。	isAR	boolean	是否开启AR。 • true：开启会话，开始发送请求。 • false：关闭会话，停止发送请求。	void

接口	描述	参数名	参数类型	参数说明	返回值
		apkUuidIsUpdate	boolean	是否更新会话。可选。 默认为false。 ● true：重置会话，重新进行VPS请求。 ● false：不重置会话。	
		multiFrame	boolean	是否连续请求定位。可选。 默认false。 ● true：连续请求定位，直到请求成功。 ● false：不连续请求定位。	
registerRefreshCameraMat	设置接收相机矩阵的回调函数。	funCallBack	Function	回调函数。需要接收cameraMat（相机矩阵）和imgData（视频流图片）两个参数。 cameraMat和imgData参数类型： ● cameraMat：{ elements: Float32Array; isMatrix4: boolean } ● imgData：ImageData	void
utm2Render	utm坐标转成渲染坐标。	listUtm	number[]	坐标数组。	Vector3
render2Utm	渲染坐标转成UTM坐标。	listRender	number[]	坐标数组。	Vector3

接口调用示例

```
// 创建AR会话
hwar.createARSession().then((res) => { });

// 设置AR状态
hwar.setARStatus(true, true, true);

// 设置接收相机矩阵的回调函数（TS语言，以Three.js为例）
hwar.registerRefreshCameraMat((cameraMat: THREE.Matrix4, imgData: ImageData) => { });

// UTM坐标转成渲染坐标（TS语言，以Three.js为例）
let vecInit: THREE.Vector3 = hwar.utm2Render([xxx, xxx, xxx]);
```

```
// 渲染坐标转成UTM坐标（TS语言，以Three.js为例）
let renderPosUtm：THREE.Vector3 = hwar.render2Utm([xxx, xxx, xxx]);
```

1.7.4 导航

导航需要使用到的接口如表1-5所示。

表 1-5 导航接口列表

接口	描述	参数名	参数类型	参数说明	返回值

searchPoi	POI智能搜索。	searchData	{
			floor?: string;
			address?: string;
			title?: string;
			tag?: string;
			pageNum?: number;
			pageSize?: number;
			}
				搜索信息。
				• floor：楼层，例如1F，2F，B1。
				• address：局点名称。
				• title：目标POI名称的关键字。
				• tag：目标POI的标签。
				• pageNum：当前页码，从0开始。
				• pageSize：每页POI数量。
					Promise<{
					count: number;
					nextStartIndex: number;
					total: number;
					list: PoiInfo[];
					}>
					返回信息列表，list包含POI名称、坐标、缩略图等信息。
					PoiInfo: {
					arMapInfo: any;
					poi: any;
					poiContent: any;
					}

getPoiImgUrl	获取可直接使用的POI缩略图的url。	url	string	searchPoi接口返回的POI缩略图的url。	Promise<any> 返回可直接使用的POI缩略图的url。

接口	描述	参数名	参数类型	参数说明	返回值

requestNavi	请求导航。
	路径点的状态分别表示："未知"、"直行"、 "左转"、"左前方"、"左后方"、"右转"、"右前方"、"右后方"、"电梯"、"结束"、"扶梯"以及"楼梯"。
		strUtmCode	string	UTM投影带号。
				例如：50N、49N
					Promise<{
					listNavi: INaviPoint[];
					policyMsg: string;
					policy: number;
					}>
					INaviPoint: {
					x: number;
					y: number;
					z: number;
					floor: string;
					building: string;
					status: number;
					}

		listFrom	number[]	起始点的UTM坐标[x, y, z]。	
		listTo	number[]	终点的UTM坐标[x, y, z]。	
		strNaviPolicy	number	路线选择策略。 • 1：最短距离 • 2：电梯优先 • 3：扶梯优先 • -2：不坐电梯 • -3：不走扶梯	
isUseManyNaviPolicy	判断是否使用多路径规划配置值。	-	-	-	boolean
onReachFloor	导航出电梯后到达指定楼层；调用此方法通知启动AR。	floor	number	楼层索引，可以从导航路径中获取。	void
onReachElevator	导航到达电梯。 因为到达电梯需要暂停AR以降低功耗。	-	-	-	void

接口	描述	参数名	参数类型	参数说明	返回值
getAngle	获取陀螺仪的alpha角。 围绕z轴（z轴垂直于屏幕表面，远离屏幕的方向为正）旋转设备将使alpha角度值发生变化。alpha为360°时表示设备的顶部正指北极方向。	-	-	-	number

接口调用示例

```
// POI智能搜索
hwar.searchPoi({
    "floor": "1F",
    "address": "xxx",
    "title": "xxx",
    "pageNum": 0,
    "pageSize": 50
}).then((res) => {
    //POI信息列表
});

// 获取可直接使用的POI缩略图的url
hwar.getPoiImgUrl("xxx").then((res) =>{ });

// 请求导航
hwar.requestNavi("xxx", [xxx, xxx, xxx], [xxx, xxx, xxx], 1).then((res) =>{ });

// 判断是否使用多路径规划配置值
hwar.isUseManyNaviPolicy();

// 导航出电梯后到达指定楼层
hwar.onReachFloor(xxx);

// 导航到达电梯
hwar.onReachElevator();

// 获取陀螺仪的alpha角
hwar.getAngle();
```

1.7.5 事件管理

事件管理使用到的接口如**表1-6**所示。

表 1-6 事件管理接口列表

接口	描述	参数名	参数类型	参数说明	返回值
dispatchEvent	事件派发。	strId	string	侦听事件id。	void

		data	{
			[key: string]: DataType
			}
				携带的数据。	

addEventListener	事件侦听。	strId	string	侦听事件id。	void
		listener	Function	侦听函数，会接收到一个参数：event: { data: Object }。	
removeEventListener	移除侦听事件。	strId	string	侦听事件id。	void
		listener	Function	侦听函数。	
registerTick	添加定时器。	strName	string	事件名称。	void
		nInterval	number	时间间隔（单位：毫秒）。	
		funCallback	Function	回调函数，没有接收参数。	
registerOnce	只执行一次的定时器。	nInterval	number	时间间隔（单位：毫秒）。	void
		funCallback	Function	回调函数，没有接收参数。	
unregisterTick	删除定时器。	strName	string	要删除的定时器名称。	void

接口调用示例

```
// 事件派发
hwar.dispatchEvent("xxx", { msg: "xxx" });

// 事件侦听
hwar.addEventListener("xxx", (evt) => { });

// 移除侦听事件
hwar.removeEventListener("xxx");

// 添加定时器
hwar.registerTick("xxx", 500, () => { });

// 只执行一次的定时器--倒计时
hwar.registerOnce(500, () => { });

// 删除定时器
hwar.unregisterTick("xxx");
```

1.7.6 SDK 内部派发事件

SDK内部派发事件如表1-7所示。

表 1-7 SDK 内部派发事件列表

事件名称	描述	携带的数据	数据类型	事件说明
GYRO_STATUS	陀螺仪有效性检测。	gyro	boolean	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.gyro表示设备陀螺仪是否正常。

WEB_INFO	版本信息。	{
		slam: string;
		phone: string;
		}
			Object	侦听函数，会接收到一个参数：event: { data: Object }
				• event.data.slam表示当前AR位置追踪算法的版本号。
				• event.data.phone表示当前设备型号。

UPDATE_CAMERA_FOV	刷新渲染相机FOV。只需要相机内参fy值，计算透视相机的fov值。	fy	number	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.fy表示设备相机内参fy值。
LANDSCAPE_BY_ROLL	当前设备是否横屏。	isLandscapeByRoll	boolean	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.isLandscapeByRoll表示当前设备是否横屏。
AUGULAR_VELOCITY_BIG	当设备角速度过大时会接收到此通知。	str	string	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.str表示提示信息的内容。

事件名称	描述	携带的数据	数据类型	事件说明
VPS_CHECK_INFO	视觉定位时检测信息提示。 ● 定位失败，请左右移动手机重新定位 。 ● 连续定位失败，请尝试更换场景。	str	string	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.str表示提示信息的内容。
ORIENTATION_NORMAL	手机俯仰角是否正常。视觉定位时需要竖屏正向手持设备。	isNormal	boolean	需要侦听回调，回调函数会接收到一个参数：event: { data: Object } event.data.isNormal表示当前设备的俯仰角是否正常。
VIDEO_STREAM_GET_ERROR	视频流获取失败。 可能原因是： ● 浏览器不支持WebRTC。 ● 请求打开摄像头权限失败。	-	-	需要侦听回调。回调函数无event参数。
VIDEO_STREAM_GET_NONE_PIXEL	视频流为空白像素。通常原因是浏览器多个标签页同时申请获取视频流。	-	-	需要侦听回调。回调函数无event参数。
NO_SRTEAM	获取设备的相机时出错。一般是设备没有后置摄像头。	-	-	需要侦听回调。回调函数无event参数。

事件名称	描述	携带的数据	数据类型	事件说明
VPS_FAIL	VPS定位失败。可能是定位不在地图范围、定位超时等原因。	-	-	需要侦听回调。回调函数无event参数。

接口调用及相机 fov 设置示例

```
// 以事件名"UPDATE_CAMERA_FOV"为例
hwar.addEventListener("UPDATE_CAMERA_FOV", (evt) => {
    resolve(evt.data.fy);//请根据返回的fy来设置渲染相机的fov
});

// Three.js r144版本 相机fov设置代码示例（TS语言）
// 获取设备标定参数后设置相机参数
public updateFov(fy): void {
    this.nVpsFy = fy;
    let nFov: number = ThreeUtils.calculateVerticalFov(RenderConfig.RenderCameraWidth, this.nVpsFy);
    let fov: number = ThreeUtils.radianToDegree(nFov);
    this.camera.fov = fov;
}
// ThreeUtils.ts
static calculateVerticalFov(nImageHeight: number, nFocal: number) {
    return 2 * Math.atan(0.5 * nImageHeight / nFocal);
}
static radianToDegree(nRadian: number) {
    return nRadian / Math.PI * 180;
}

// LayaAir 2.12版本 相机fov设置代码示例（TS语言）
// 获取设备标定参数后设置相机参数
public updateFov(fy): void {
    this.nVpsFy = fy;
    let nFov: number = LayaUtils.calculateVerticalFov(RenderConfig.RenderCameraWidth, this.nVpsFy);
    let fov: number = LayaUtils.radianToDegree(nFov);
    this.mainCamera.fieldOfView = fov;
}
// LayaUtils.ts
static calculateVerticalFov(nImageHeight: number, nFocal: number) {
    return 2 * Math.atan(0.5 * nImageHeight / nFocal);
}
static radianToDegree(nRadian: number) {
    return nRadian / Math.PI * 180;
}
```

1.8 代码示例

开启AR会话功能的代码样例，具体见WebARSDK包中hwar目录下的demo.html。

说明

WebARSDK包的目录请参考**图1-3**。

1.9 常用调试方法

1.9.1 调试工具

常用的调试工具包括：

- 基于Chromium内核的浏览器（例如Edge、Chrome）的开发者工具（DevTools）
- Safari浏览器

1.9.2 调试方法

准备工作

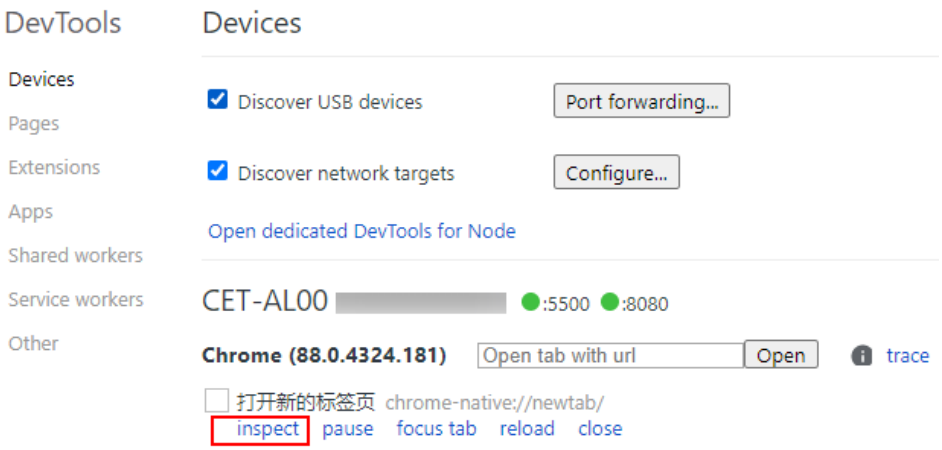
您可根据实际情况选择如下任一组设备进行调试：

- 安卓手机、数据线、Windows系统电脑。
- 苹果手机、数据线、Mac系统电脑。

调试步骤

- 选择安卓手机调试。
  a. 用数据线连接手机、电脑，手机开启USB调试模式。
  b. 在Chrome浏览器输入：chrome://inspect/#devices，手机打开需要调试的页面停留片刻会出现调试的设备，如图1-8即表示连接成功，单击“inspect”进入调试页面。

  图 1-8 DevTools 成功连接安卓设备

  c. 在DevTools中的Console页签查看打印输出情况，在“Network”页签查看网络请求发送情况，在“Sources”页签对工程文件进行断点调试。
- 选择苹果手机调试。
  a. 在Mac电脑中，选择“Safari 浏览器 > 偏好设置”，单击“高级”，勾选“在菜单栏中显示‘开发’菜单”。
  b. 苹果手机开启调试模式。
  c. 苹果手机连接Mac电脑，打开手机Safari浏览器并运行Web页面。
  d. 在Mac电脑的Safari浏览器“开发”菜单下选择已连接的苹果手机，单击手机打开的Web页面进行调试。

2 HTSDK 使用手册

2.1 开发概述

HTSDK是接入KooMap AR地图运行服务的端侧SDK，助力于快速构建真实和虚拟融合世界。HTFoundation是适配HTSDK的示例代码，供用户在Unity3D中开发业务。

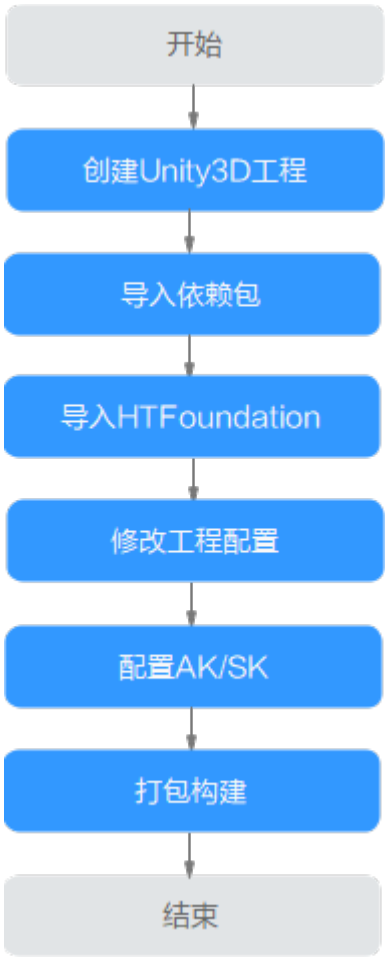
HTFoundation示例实现了真实世界构建、世界位姿定位和跟踪、虚实世界呈现。基于HTFoundation示例，用户可构建虚拟世界，包括虚拟对象设计、行为设计实现和业务逻辑开发。

图 2-1 工作原理

2.2 总体开发思路

使用HTFoundation示例开发移动端应用的工作流程如图2-2所示。

图 2-2 工作流程



1. 创建一个新的工程目录。

   使用Unity3D引擎创建基础工程，基础工程里面会有一个空的工程目录。详情见**创建Unity项目**。

2. 导入ARFoundation插件包。

   在Unity的Packages文件夹中找到manifest.json，输入指定命令，Unity3D将会自动导入所添加的插件包。详情见**添加插件**。

3. 导入HTFoundation示例。

   解压Cyberverse-HTFoundation.zip并将文件夹复制到Unity工程对应目录。详情见**导入HTFoundation示例**。

4. 修改工程配置。

   单击菜单“Custom Tools > Set HTFoundation Config”自动设置工程所需配置项。详情见**配置项目选项**。

5. 配置AK/SK。

   包含访问密钥ID（AK）和秘密访问密钥（SK）两部分，是用户在华为云的长期身份凭证。华为云通过AK识别访问用户的身份，通过SK对请求数据进行签名验证。详情见**配置域名及密钥**。

6. 打包构建。

   在“HTFoundation > Samples”下找到“HTFoundationTester”，将其添加到打包场景中进行打包构建。详情见**验证结果**。

2.3 准备工作

开通关联服务

开通AR地图运行服务。

说明

AR地图运行服务可在**KooMap管理控制台**开通。

开发技能要求

- 熟悉C#语言开发。
- 熟悉Unity3D开发。

下载 SDK

请下载**Cyberverse-HTFoundation安装包**和**安装包完整性校验文件**。

搭建开发环境

- 准备Windows，安装Unity2020或以上版本以及Visual Studio。
- 准备iOS环境，安装Xcode。
- 准备Android环境，安装Android Studio。

说明

如果您需要在华为手机上使用SDK，请咨询**华为云专业服务团队**。

2.4 快速开始

步骤1 创建Unity项目。

新建一个Unity Project，“Template”选择“3D”。

步骤2 导入ARFoundation包，并配置ARCore XR Plugin和ARKit XR Plugin。

1. 打开工程里自动创建的Packages文件夹目录下的manifest.json文件，在manifest.json输入如下内容，Unity3D将会自动导入所添加的插件。

```
"com.unity.xr.arfoundation": "4.1.13",
"com.unity.xr.arcore": "4.1.13",
"com.unity.xr.arkit": "4.1.13",
```

说明

您也可以在菜单栏“Windows”里找到“Package Manager”，单击“Package Manager”直接选择需要添加的插件。

2. 插件包导入完成后配置平台支持ARCore XR Plugin和ARKit XR Plugin。

在菜单栏“Edit > ProjectSettings > XR Plug-in Management”路径下：

- 选择  平台，勾选“ARCore”。
- 选择iOS平台，勾选“ARKit”。

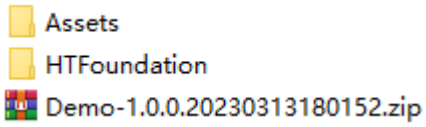
步骤3 导入HTFoundation示例。

说明

请先在“File > Build Settings > Player Settings > Player > Other Settings”中找到“Allow 'unsafe' Code”，并将其勾选。

1. 解压SDK压缩包Cyberverse-HTFoundation.zip。压缩包下载地址请参考**下载SDK**。
2. 在解压的文件夹中找到CloudBuildOutput文件夹，并将该文件夹里面的Demo-xx.zip解压到同级目录，解压后得到一个Assets文件夹。

图 2-3 解压 Demo 压缩包



3. 在解压的“Cyberverse-HTFoundation > CloudBuildOutput”下，找到HTFoundation文件夹，并将其合并到“Cyberverse-HTFoundation > CloudBuildOutput > Assets”的HTFoundation文件夹。
4. 将以上合并后的Assets文件夹合并到**步骤1**新建的Unity工程的Assets目录。

步骤4 配置Plugins文件夹里的文件。

在新建的Unity工程的“Assets > HTFoundation”找到Plugins文件夹，文件夹中包含Android、iOS文件夹以及HT开头的SDK功能模块dll，请依照如下指引进行配置：

- 选中Android文件夹下的so文件，在Inspector窗口中，“Select platforms for plugin”勾选“Android”，“Platform settings”中的“CPU”选择“ARM64”。
- 选中iOS文件夹下后缀为.h、.mm和.a的文件，在Inspector窗口中，“Select platforms for plugin”勾选“iOS”。
- 选中文件名包含Android的dll，在Inspector窗口中，“Select platforms for plugin”勾选“Android”。
- 选中文件名包含iOS的dll，在Inspector窗口中，“Select platforms for plugin”勾选“iOS”。

步骤5 配置项目选项。

完成以上操作后，找到菜单栏的“Custom Tools”并单击“Set HTFoundation Config”配置项目选项，Unity将会自动进行以下设置：

- 通用设置：
  - 添加Tag：Photo、Object。
  - 添加Layer：
    - Layer6：HTUI
    - Layer17：Danger
  - 不使用垂直同步。
  - 默认使用竖屏。
  - 允许不安全代码。

  - 关闭代码裁剪。
- iOS设置：
  - iOS相机权限描述。
  - iOS定位权限描述。
  - IL2CPP脚本运行时环境。
- Android设置：
  - 使用OpenGLES3图形API。
  - 关闭多线程渲染。
  - 使用最小的SDK版本29。
  - 使用目标的SDK版本29。
  - IL2CPP脚本运行时环境。
  - 支持ARM64位处理器。

步骤6 配置域名及密钥。

工程运行需要证书校验，请将申请到的AK/SK以及固定的BASE_URL填写在新建Unity工程的“Assets > HTFoundation > Scripts > BaseUtils > HTStringRes.cs”中的HTStringConfig里面。

说明

- 申请AK/SK详情请参考**获取AK/SK**。
- 固定BASE_URL为：https://koomap.cn-north-4.myhuaweicloud.com。

```
// 认证用的AK、SK硬编码到代码中或明文存储都有很大的安全风险，建议在代码中配置加密后的AK、SK，在使用的地方解密。
public class HTStringConfig
{
    public const string AK = "加密后的内容";
    public const string SK = "加密后的内容";
    public const string BASE_URL = "";
}
```

步骤7 验证结果。

在菜单栏单击“File > Build Settings > Add open Scenes”并添加HTFoundationTester场景：

- Unity导出Android apk：“Platform”选择“Android”，单击“Switch platform”。不勾选“Export Project”，单击“build”导出apk。
- Unity导出Android项目：“Platform”选择“Android”，单击“Switch platform”。勾选“Export Project”，单击“Export”导出项目。
- Unity导出iOS项目：“Platform”选择“iOS”，单击“Switch platform > Export”导出项目。

----结束

2.5 创建 HTFoundation 会话

2.5.1 概述

SDK功能集成在HTFoundationOrigin预制体中，默认开启相机预览流、定位、资源加载与调试面板功能，主要功能包含导航、白膜显隐等。其主要是由HTSysEntrance和Canvas构成：

- HTSysEntrance：负责提供SDK功能并维护生命周期。
- Canvas：负责提供示例界面UI。

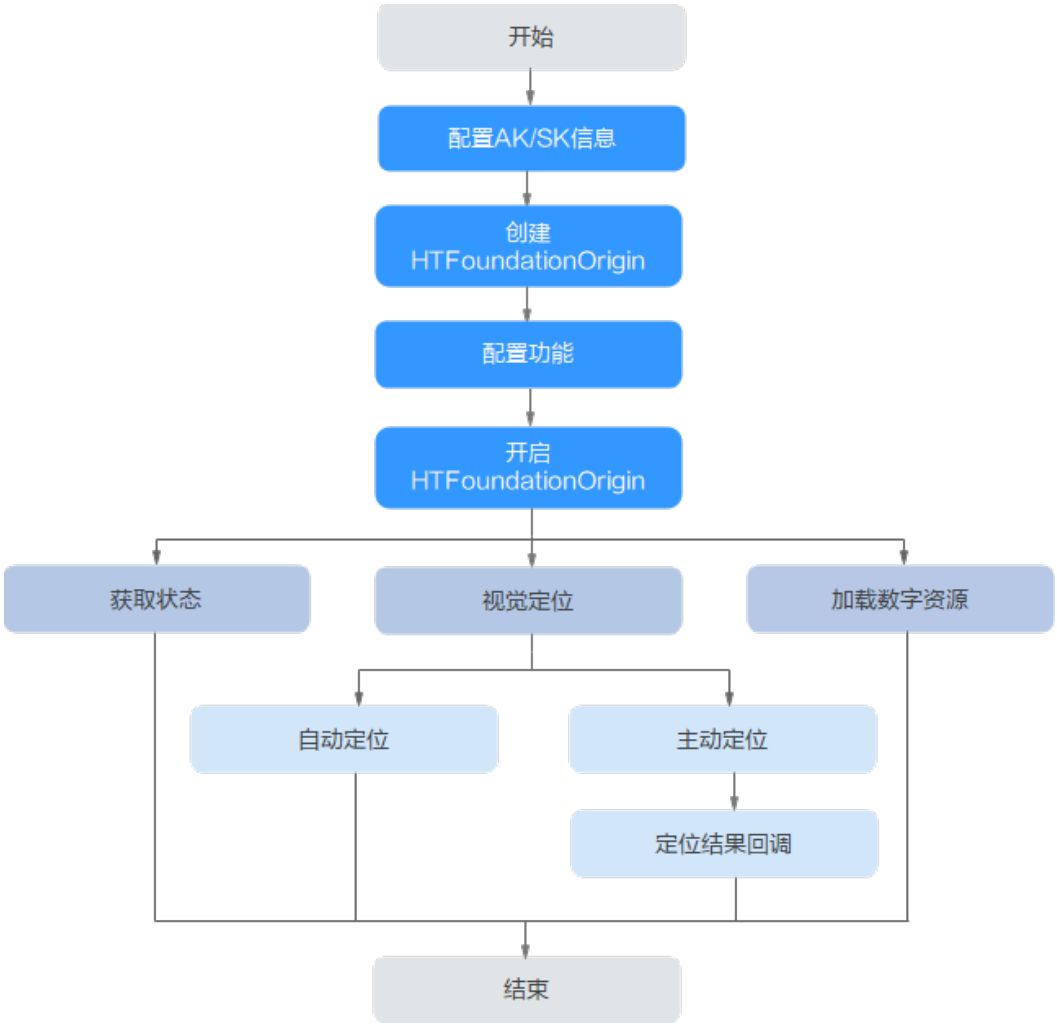
在新建的Unity工程“Assets > HTFoundation > Scenes”路径下包含一个基础场景HTFoundationScene，它是一个只包含HTFoundationOrigin预制体的场景。您可以直接在这个场景进行二次开发。

2.5.2 开发指导

开发流程

创建和开发流程如图2-4所示。

图 2-4 流程图



开发过程

步骤1 配置AK/SK信息。

在新建的Unity工程“Assets > HTFoundation > Scripts > BaseUtils”中找到HTStringRes.cs，并在HTStringRes.cs配置AK/SK信息以及输入固定的BASE_URL。

说明

- 申请AK/SK详情请参考**获取AK/SK**。
- 固定BASE_URL为：https://koomap.cn-north-4.myhuaweicloud.com。

```
// 认证用的AK、SK硬编码到代码中或明文存储都有很大的安全风险，建议在代码中配置加密后的AK、SK，在使用的地方解密。
public class HTStringConfig
{
    public const string AK = "加密后的内容";
    public const string SK = "加密后的内容";
    public const string BASE_URL = "";
}
```

步骤2 创建HTFoundationOrigin。

单击Unity菜单栏“Custom Tools > Create”中的“HTFoundationOrigin”，Unity会自动创建Prefab。

步骤3 配置功能。

展开“HTFoundationOrigin”目录并单击“HTSysEntrance”，根据实际情况配置需要的功能。其中默认配置HTRes（资源加载）、HTNavi（导航功能）、HTGvps（定位功能）。

配置完成后，启动当前场景，自动获取状态信息，开启视觉定位和加载数字资源：

- 视觉定位：自动定位和手动定位。
  - 自动定位：

    系统默认开启自动定位功能。在定位丢失状态下，每隔1秒刷新一次定位，直到成功。定位成功状态下，每隔15秒刷新一次定位。您可根据实际需求设置刷新时间。

    ```
    // 第一个参数是定位丢失时的定位刷新时间，第二个参数是定位成功后的定位刷新时间
    mHtSystem.Start(HTSystemManager.FastLocinterval,
    HTSystemManager.NormalLocaIntervalNonVpp);
    ```

  - 手动定位：

    i. 创建HTGvpsRequestOpExt实例。

    ```
    protected HTGvpsRequestOpExt mGvpsOp = new HTGvpsRequestOpExt();
    ```

    ii. 调用HTGvpsRequestOpExt的RequestGvpsLocaiton方法进行一次手动定位请求。

    ```
    mGvpsOp.RequestGvpsLocaiton(HTSystemManager.GetHTSystem(),
    HTRequestPriority.ForceRequest, HTVpsRequestType.ManualClick, OnLocation);
    ```

    iii. 通过HTGvpsRequestOpExt类的Addobserver方法将定位添加到SDK系统的监听池中。

    定位结果将通过HTGvpsRequestOpExt类中的OnLocation方法返回。

    ```
    mGvpsOp.AddObserver(HTSystemManager.GetHTSystem());
    ```

    说明

    您可在Unity新建工程的“Assets > HTFoundation > Samples > HTFoundationTester > Scripts”中找到HTManualReq.cs并查看参考实例。

- 加载数字资源：

  HTResManager为场景中挂载的类，控制着HTResSession类的创建以及资源加载流程的生命周期。

  HTResSession构造了资源管理类HTResPumper和资源渲染类HTResRender，主要对外提供了dll的生命周期，相对于Unity生命周期的映射及内容回调。

  对资源模块的信息配置及功能调整，都可以在这里进行。

  HTResCtro主要进行资源材质的二次赋值，能很好的修复PC端加载资源出现材质丢失的问题，主要方法为FixShader(GameObject obj)。

  核心方法代码如下：

```
public void Create(bool isRequestDigital = true, bool isUseSecondaryOffset = true)
    {
        Debug.Log("HTResSession Create HTPerformance +");
        if (mHTBehaviourFactory == null)
        {
            mHTBehaviourFactory = new HTBehaviourFactory();
        }
        bool isResLstOnline = true;
        bool isResLstDump = false;
        bool isResCacherDe = false;
        mResRender = new HTResRender(mOwner, mHTBehaviourFactory);
        mResPumper = new HTResPumper(
            HTSystemManager.GetHTSystem().GetAppConfig(),
            HTSystemManager.GetHTSystem().GetAppEnv(),
            HTSystemManager.GetHTSystem().GetWorldControl(),
            HTSystemManager.GetHTSystem().GetSniffer(),
            HTSystemManager.GetHTSystem().GetJniAttach(),
            mResRender,
            mSysSession.mHostUrl,
            Application.temporaryCachePath,
            mSysSession.mPlaformType,
            isResLstOnline,
            isResLstDump,
            isResCacherDe,
            isRequestDigital,
            isUseSecondaryOffset
            );
        mResRender.SetRealWordMatPath(HTStringMat.WORLD_REAL);
        ((IHTActor)(mResPumper)).Create("resPumper");
        Debug.Log("HTResSession Create HTPerformance -");
    }
```

相关参数解释如下：

- HTBehaviourFactory：资源的行为脚本厂类，SDK提供了一套通用的行为供用户使用，用户也可以创建自己的行为厂类，需继承IHTBehaviourFactory接口类。
- isResLstOnline：是否加载在线资源（加载离线资源需先缓存Dump，并且把缓存的TXT放在Application.temporaryCachePath下的ResLstCache文件夹内）。
- isResLstDump：是否缓存资源列表的TXT。
- GetAppConfig：获取应用鉴权参数。
- GetAppEnv：获取App环境参数。
- GetWorldControl：获取资源世界的参数，例如是否在服务区、获取坐标偏移量等信息。
- GetSniffer：获取定位及传感器信息。
- GetJniAttach：子线程向Android客户端发送信息时需要使用。
- mResRender：HTResRender.dll的实例。
- mHostUrl：存储云端数字资源的域名。

- mPlaformType：当前移动设备的平台（Android或iOS）。
- isRequestDigital：是否需要加载数字资源。
- isUseSecondaryOffset：是否开启**二次偏移**。

说明

资源模块运行时，判断当前位置是否在服务区，可通过如下接口进行判断：

```
int serviceid = HTSystemManager.GetHTSystem().GetWorldControl().GetServiceCode();
```

其中，返回的serviceid参数解释如下：

- serviceid > 0：在服务区
- serviceid = 0：未知
- serviceid < 0：不在服务区

----结束

2.5.3 注意事项

- 模块之间启动销毁存在相关依赖需要按照一定顺序执行，HTSysEntrance需要最先执行。脚本执行顺序可见“Assets > HTFoundation > Scripts”中的SessionsManager.cs文件。
- 由于SDK的虚拟世界比较庞大，导致距离坐标世界原点较远的虚拟资源坐标值很大，可能会引起一些渲染上的异常（例如模型闪烁，抖动等），因此SDK会将资源进行二次偏移以减小资源的坐标值。

说明

如果您需要获取数字资源在虚拟世界中根节点的偏移量，请参考如下方法：

```
HTVector2f vector2f = HTSystemManager.GetHTSystem().GetWorldControl().GetSecondaryOffset();//（x轴和z轴，y轴无偏移）
```

2.6 AR 导航

2.6.1 概述

AR导航功能依赖视觉定位，进行空间位置跟踪。它将会根据当前位置信息及终点信息获取路径点信息，然后进行3D虚拟世界渲染操作。

其包含的功能主要有路线规划及导航指示：

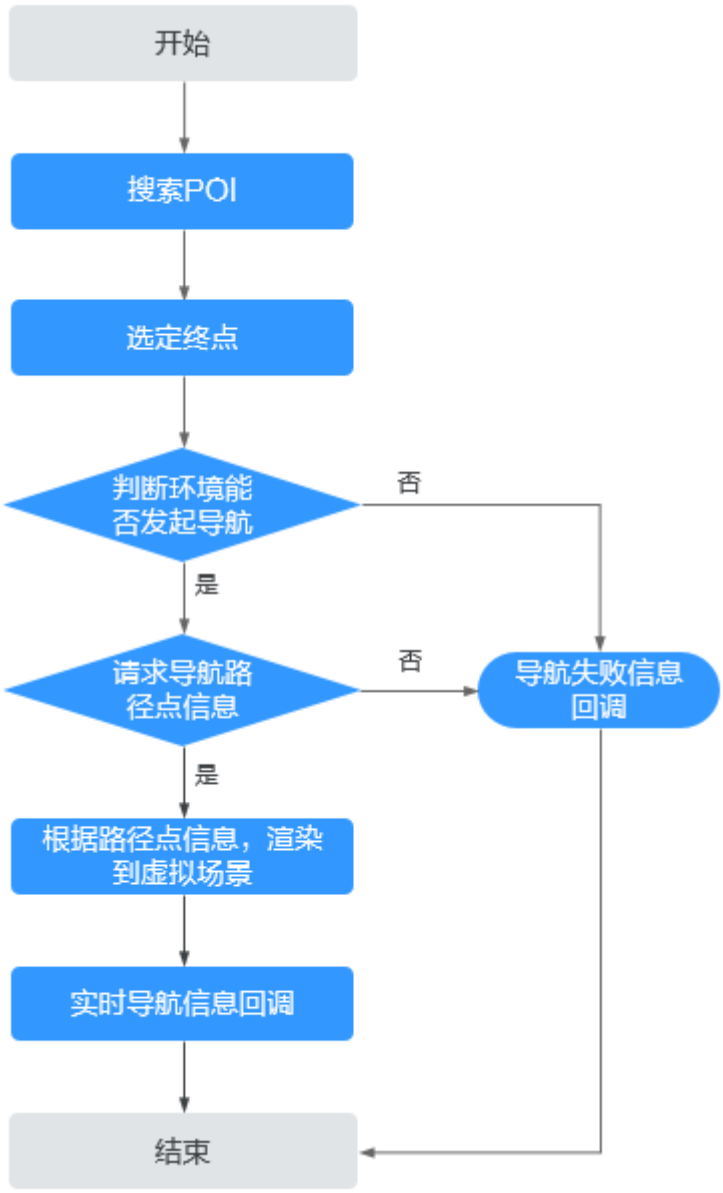
- 路线规划：根据目的地和当前位置，自动规划合适的路线。
- 导航指示：引导用户沿着设定的路线前进，并在需要时提供转弯提示等导航信息。

2.6.2 开发指导

开发流程

开发流程请参考**图2-5**。

图 2-5 开发流程图



开发过程

步骤1 搜索POI。

```
// 创建获取POI的异步任务栈
var mPOICloud = new HTPOICloud();

// 将生命周期绑定到unity的生命周期
mPOICloud.Create("NAVIPOI"); // 参数设置的是线程名
mPOICloud.Start();
mPOICloud.Stop();
mPOICloud.Destroy();

// 设置系统的基本参数
mPOICloud.SetParams(HTSystemManager.GetHTSystem().GetAppConfig(),
HTSystemManager.GetHTSystem().GetAppEnv(), hostUrl);

// 设置成功失败的回调监听
mPOICloud.AddListListener(SelecteDestination);
```

```
mPOICloud.AddErrorListener(OnPoiError);

// 根据gps获取附近的POI
double lon = HTSystemManager.GetHTSystem().GetSniffer().GetGpsLocation().mLongitude;
double lat = HTSystemManager.GetHTSystem().GetSniffer().GetGpsLocation().mLatitude;
mPOICloud.GetDataList(lon, lat);
```

步骤2 选定终点。

HTNaviManager会自动管理HTNaviSession中导航功能的生命周期，被默认挂载在预制体当中，使用时可以直接通过HTNaviSession的mNavi来调用各种方法。

```
// 导航终点坐标是虚拟世界坐标，可以直接带入POI上返回的坐标
mNaviSession.mNavi.SetDestination(447763.6348, 4418530.1394, 37.979218);
```

步骤3 发起导航。

```
mNaviSession.mNavi.StartNavi();
```

步骤4 回调监听。

```
// 导航实时信息监听
mNaviSession.mNaviInfoListener = OnNaviInfoStatus;
// 导航失败信息监听
mNaviSession.mNaviErrorListener = OnNaviError;
// 进入电梯信息监听
mNaviSession.mNaviElevatorListener = OnNaviElevator;
// 方向引导信息监听
mNaviSession.mNaviDirectionListener = OnNaviDirection;
// 偏移量信息监听
mNaviSession.mNaviYawListener = ShowYaw;
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Samples > HTFoundationTester > Scripts”中找到HTNaviCtro.cs并查看参考实例。

----结束

2.6.3 注意事项

创建导航

```
// HTNavigationProvider是为导航模块提供参数与配置的类，通过对这个类的参数调整，可以对导航功能进行详细配置
mNaviProvider = new HTNavigationProvider(HTSystemManager.GetHTSystem().GetARPlugin(), HTSystemManager.GetHTSystem().GetWorldControl());

// HTNavigationRender负责将路径节点渲染成导航路径
mNaviRender = new HTNavigationRender();

// HTNaviManage是sdk中导航功能的管理类，回调都是从这个类中接收的
mNavi = new HTNaviManage();
mNavi.Create(HTSystemManager.GetHTSystem().GetAppConfig(),HTSystemManager.GetHTSystem().GetAppEnv(),hostUrl,mNaviProvider,mNaviRender);
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Scripts > HTPlatform”中找到HTNaviSession.cs并查看参考实例。

HTNaviManager

导航的生命周期需要与Unity绑定，您可以参考“Assets > HTFoundation > Scripts > SessionsManager”中的HTNaviManager.cs文件。这里主要说明Update、导航回调和渲染的注意事项。

- Update：

```
// Update负责实时导航功能的实现，isTracking表示的是当前用户的定位状态
mNavi.Update(isTracking);
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Scripts > HTPlatform”中找到HTNaviSession.cs并查看参考实例。

- 导航回调：

```
// 这是一个必要监听
mNavi.AddNaviErrorListener(NaviErrorListener);
// 这是导航启动失败的错误，需重新开始导航
HTNaviErrorCode.LOAD_RES
HTNaviErrorCode.GVPS_DISABLE
HTNaviErrorCode.NET_DISABLE
// 这是请求导航路径失败，会中断导航可通过ContinueNavi继续
HTNaviErrorCode.NET_ERROR
// 这是提示性错误不会中断导航
HTNaviErrorCode.NAVIGATING
HTNaviErrorCode.LOST_TRACK

// 这是一个必要监听
// 进入电梯会中断导航，需要外部的重定位来继续导航
mNavi.AddNaviElevatorListener(NaviElevatorListener);
// 这个方法通知导航模块已经出电梯了
mNaviSession.mNavi.ContinueNavi();
// 这个方法通知导航模块已重新gvps定位
mNaviSession.mNavi.IsRefreshGvps();

// 这是一个可选监听
// 如果启动了导航的偏移量监听，在用户偏航的时候会根据返回的bool中断导航。中断后，需要调用RequestNaviInfo，获取最新的导航路线继续导航
mNavi.AddNaviYawListener(NaviYawListener);
mNaviSession.mNaviYawListener = ShowYaw;
public bool ShowYaw(float yaw)
{
    ReplaceBtnListener(mNaviInfoDialogPanel.GetComponentInChildren<Button>(), () =>
    {
        mNaviSession.mNavi.RequestNaviInfo();
        mNaviInfoDialogPanel.SetActive(false);
    });
    mNaviInfoDialogPanel.GetComponentInChildren<Text>().text = "偏航" + yaw + "m,为您重新规划路线";
    mNaviInfoDialogPanel.SetActive(true);
    return true;
}

// 这是一个可选监听
// 如果启动了导航路径监听，在获取导航路径的时候会中断导航，等待调用ResetNaviRoute选择导航路径继续导航
mNavi.AddNaviPathListener(NaviPathListener);
mNaviSession.mNavi.ResetNaviRoute(naviContent.path[0]);
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Scripts > SessionsManager”中找到HTNaviManager.cs并查看参考实例。

- 渲染：

由于导航默认会根据白膜调整导航高度，所以系统会返回白膜渲染完的通知。如果不需要等待通知可以直接提前调用IsRenderReady。

```
mNavi.IsRenderReady();
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Samples > HTFoundationTester > Scripts”中找到HTNaviCtro.cs并查看参考实例。

HTNavigationProvider

在HTNavigationProvider中可以配置很多导航模块参数，如果想进行更多的参数调整，可以通过调用IHTNavigationProvider接口或者继承HTNavigationProvider来实现。

```
public class HTNavigationProviderExt : HTNavigationProvider
{
  ...
}
mNaviProvider = new HTNavigationProviderExt(HTSystemManager.GetHTSystem().GetARPlugin(), HTSystemManager.GetHTSystem().GetWorldControl());
```

说明

您可在Unity新建工程的“Assets > HTFoundation > Scripts > HTPlatform”中找到HTNavigationProviderExt.cs并查看参考实例。

HTNavigationRender

HTNavigationRender主要负责对导航路径的渲染，通过继承和实现接口实现高度自定义。

说明

您可在Unity新建工程的“Assets > HTFoundation > Scripts > HTPlatform”中找到HTNavigationRenderExt.cs并查看参考实例。

2.7 代码示例

完整的示例代码在SDK压缩包**Cyberverse-HTFoundation.zip**的Demo.zip中。

您可在创建完成后的Unity项目中找到对应的示例代码。

2.8 常用调试方法

2.8.1 调试工具

调试工具包括：

- 安卓手机
- Windows系统电脑
- USB数据线

2.8.2 cfg.ini 调试方法

基本原理

您可以通过创建和修改应用目录下的cfg.ini文件调试信息配置，调试信息在配置文件中以键值对的形式存在。ini文件的配置目录如下：

- Unity：*C:|...|AppData|LocalLow|{company name}|{product name}|ini*
- Android：*/sdcard/android/data/{pakagename}/files/ini*

- iOS：手机连接Mac操作系统电脑，连接后在电脑的“iPhone > 文件”里面找到导入的应用名，应用名下有ini文件夹，请把配置了如下命令的ini文件拖入ini文件夹即可。

  配置样例：

  ```
  loglevel:UPDATE
  debugger_ctrl_switch:ON
  fixed_location_switch:ON
  longitude:11.111111
  latitude:22.2222222
  ```

设置调试选项

此功能必须在Unity的工程配置宏定义UNITY_HTDEBUG后方可使用，设置步骤如下：

步骤1 打开“Unity > Edit”，并找到“Project Settings”。

步骤2 在“Project Settings”中找到“Player”并单击打开。

步骤3 找到“Other Settings”，往下拉，可见“Script Compilation”下方的“Scripting Define Symbols”。

步骤4 在“Scripting Define Symbols”输入框中输入“UNITY_HTDEBUG”。

步骤5 单击“Apply”，设置成功。

----结束

调试选项

- loglevel：UPDATE|INFO|WARNING|ASSERT|EXCEPTION|ERROR|NONE

  日志级别左边包含右边的日志信息，即UPDATE级别会打印所有日志信息，使用该字段会打印HTDebug的日志，并写日志到此文件的上一级文件目录。NONE表示不保存log。
- debugger_ctrl_switch：ON|OFF

  是否在日志中显示调试面板，只对HTAppUIType中mNativeAppIds包含的应用包名有效。
- fixed_location_switch：ON|OFF

  是否使用自定义的固定位置。
- longitude

  经度，填写固定位置经纬的值，只在“fixed_location_switch”为“ON”时有效。
- latitude

  纬度，填写固定位置纬度的值，只在“fixed_location_switch”为“ON”时有效。

3 XRLightSDK 使用手册

3.1 开发概述

XRLightSDK 简介

XRLightSDK是一款基于微信小程序平台开发的SDK，支持租户通过SDK快速集成和使用KooMap AR地图运行服务。借助小程序平台免安装即可运行，易于推广。

XRLightSDK提供的API，可以实现世界位姿定位和跟踪，将用户、空间及数字内容连接在一起，助力用户构建真实和虚拟融合的世界。

工作原理

用户需要新建微信小程序工程，然后导入XRLightSDK软件包，通过XRLightSDK调用AR地图运行服务，在微信小程序工程中，添加自定义的业务逻辑，并对小程序进行调试，确保所有功能正常运行。调试通过后，将小程序提交审核并发布上线。

图 3-1 XRLightSDK 工作原理

3.2 快速开始

使用微信开发者工具创建小程序工程，导入SDK。

说明

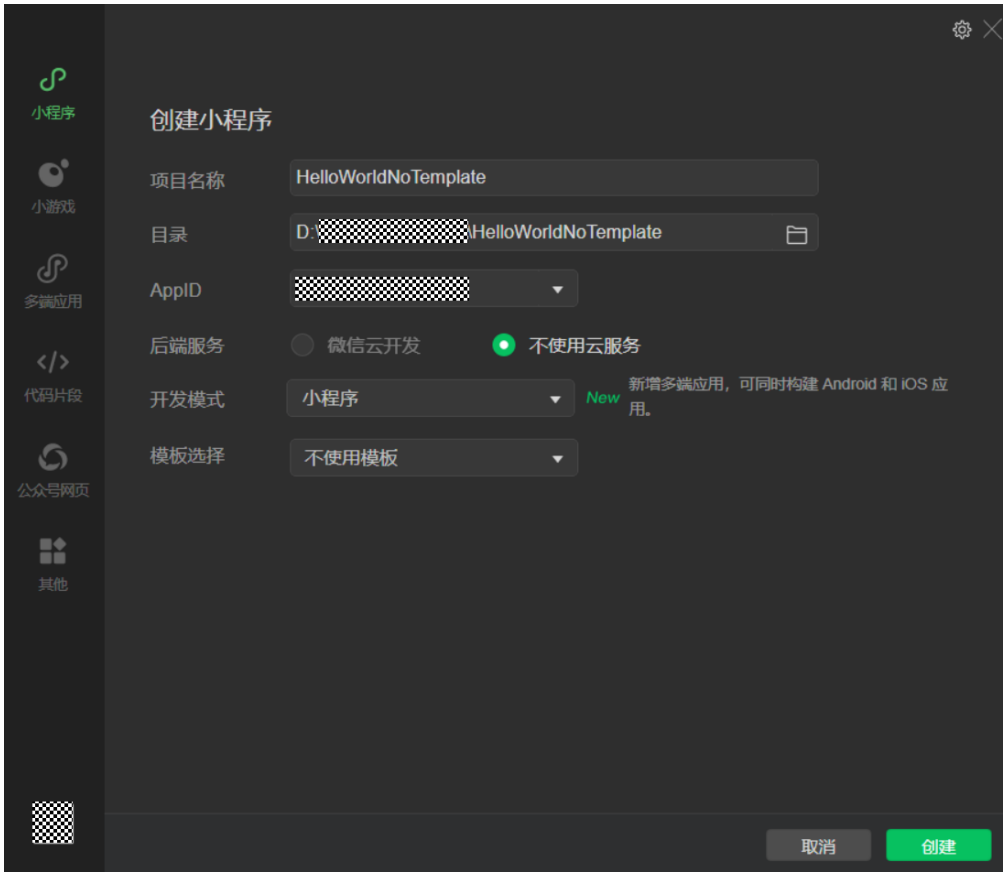
本文档使用的微信开发者工具版本为Nightly 1.06.2411282。

步骤1 使用微信号登录微信开发者工具。

步骤2 新建微信小程序工程。

1. 单击“新建项目”，进入“创建小程序”页面。

图 3-2 创建小程序



- 在“项目名称”文本框中填写项目名称。
- “目录”选择项目存放的路径。
- “AppID”选择“测试号”。
- “后端服务”选择“不使用云服务”。
- “开发模式”选择“小程序”。
- “模板选择”选“不使用模板”。

2. 单击“创建”，完成小程序工程创建。创建完成后，工程目录如图3-3。

图 3-3 目录名称

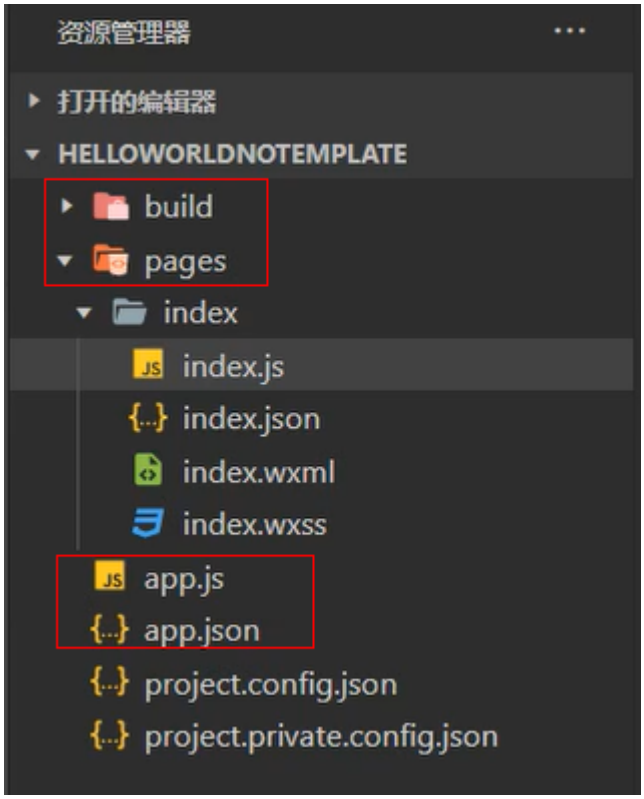
步骤3 新建文件并导入SDK。

1. 在工具的资源管理器，单击右键，选择“新建文件”，新建文件“app.json”，并在app.json中配置页面（pages），代码如下：

```
{
 "pages": [
  "pages/index/index"
 ]
}
```

2. 单击右键，选择“新建文件”，新建文件“app.js”。
3. 按Ctrl+S组合键，保存并编译，工具会自动生成pages文件夹及文件夹下的文件。
4. 下载并解压**CyberverseXRLightSDK软件包**。
5. 打开工程目录，将解压后得到的build文件夹复制到pages同级目录。

图 3-4 导入 SDK 后目录



步骤4 调用SDK，获取版本号。

1. 在“index.js”文件的第一行导入XRClient。

```
import { XRClient } from "../../build/XRClient";
```

2. 在“index.js”文件的onLoad方法里打印SDK版本号。

```
onLoad(options) {
  console.log(XRClient.getVersion())
},
```

步骤5 真机调试，验证结果。

说明

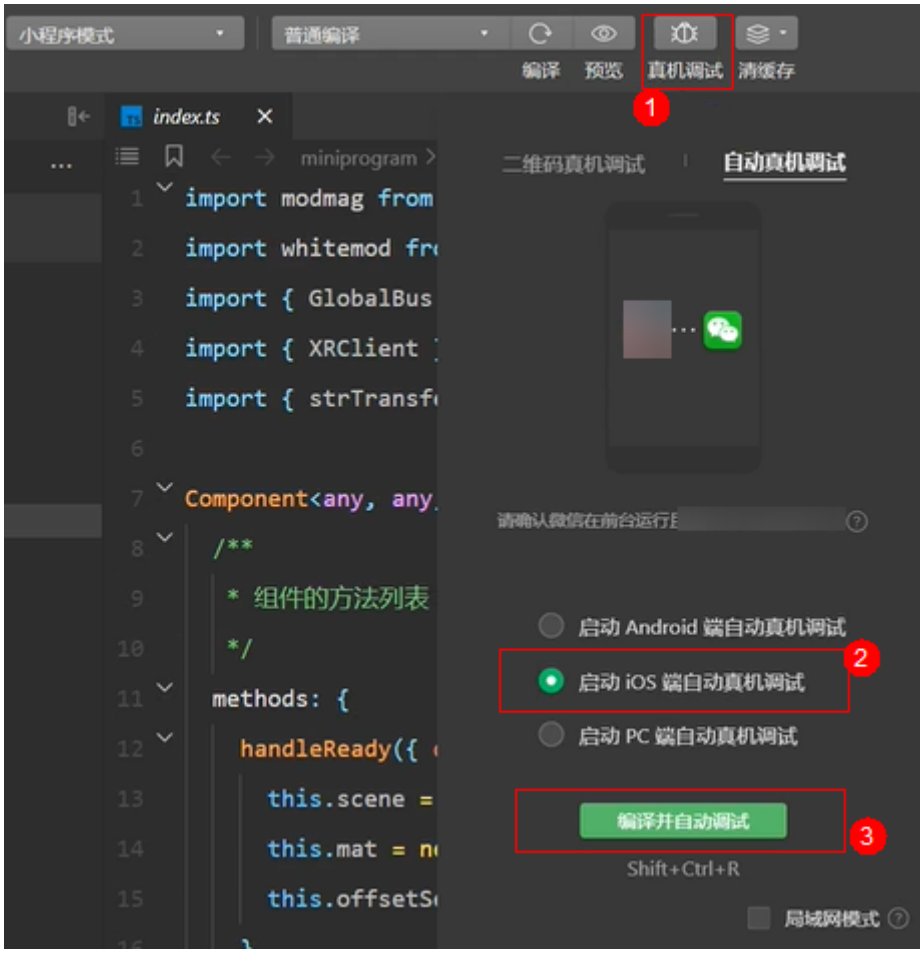
自动调试前，确保移动端设备的微信在前台运行，且与开发者工具登录的是同一个账号。

1. 在微信开发者工具中，单击“真机调试”。
2. 根据移动端设备情况选择“启动 Android 端自动真机调试”或 “启动 iOS 端自动真机调试”。

XRLightSDK是基于XRFrame方案的AR应用，不支持PC端调试。

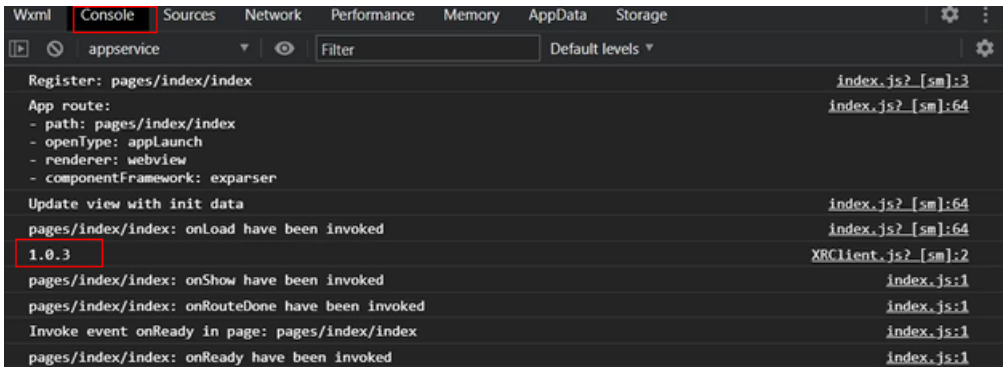
3. 单击“编译并自动调试”，弹出“真机调试”窗口。

图 3-5 真机调试



4. 在“Console”页签下，查看输出的内容，如果打印了SDK版本号信息，说明引入SDK成功。

图 3-6 查看 SDK 版本信息



----结束

3.3 开启 AR 会话

3.3.1 开启 AR 会话方案概述

您可开启AR会话获取设备的图像、GPS坐标及传感器数据，启动SLAM从而进行视觉定位。

视觉定位成功后，您可获取到当前位置的坐标信息，并在附近渲染数字内容，从而构建虚实融合的世界。

说明

传感器包括陀螺仪、加速度计、磁力计。

3.3.2 准备工作

开发技能要求

- 具备TypeScript/JavaScript开发基础。
- 熟悉XRFrame的XR/3D应用解决方案。

下载并解压 XRLightSDK

请下载**XRLightSDK软件包**和**软件包的完整性校验文件**，并解压软件包。

说明

SDK软件包中的目录结构：

- workers文件夹：工程引用的worker。
- XRClient.d.ts文件：申明文件。
- XRClient.js：SDK的核心逻辑。

收集 AK/SK 信息

表 3-1 收集信息

信息项	说明
AK/SK	访问密钥。包含访问密钥ID（Access Key ID，AK）和秘密访问密钥（Secret Access Key，SK）两部分，是您在华为云的长期身份凭证。华为云通过AK识别访问用户的身份，通过SK对请求数据进行签名验证，用于确保请求的机密性、完整性和请求者身份的正确性。 获取方法请参见**访问密钥**。

开通关联服务

开通AR地图运行服务。

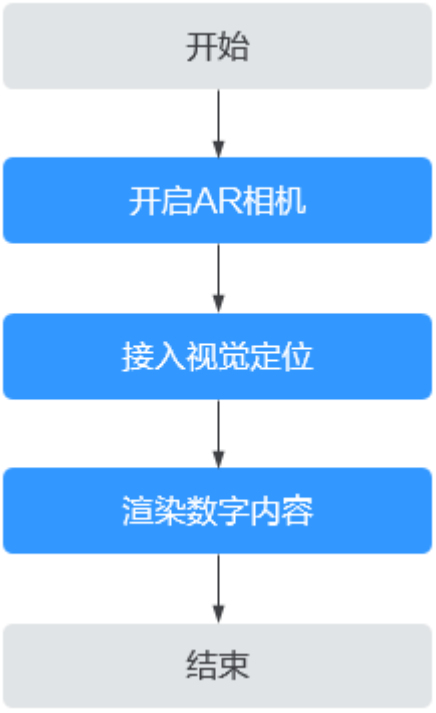
环境要求

- 已安装微信开发者工具。
- 移动设备已配置后置摄像头、陀螺仪、加速度传感器、GPS芯片等器件。

3.3.3 开发指导

开发流程

图 3-7 开启 AR 会话开发流程



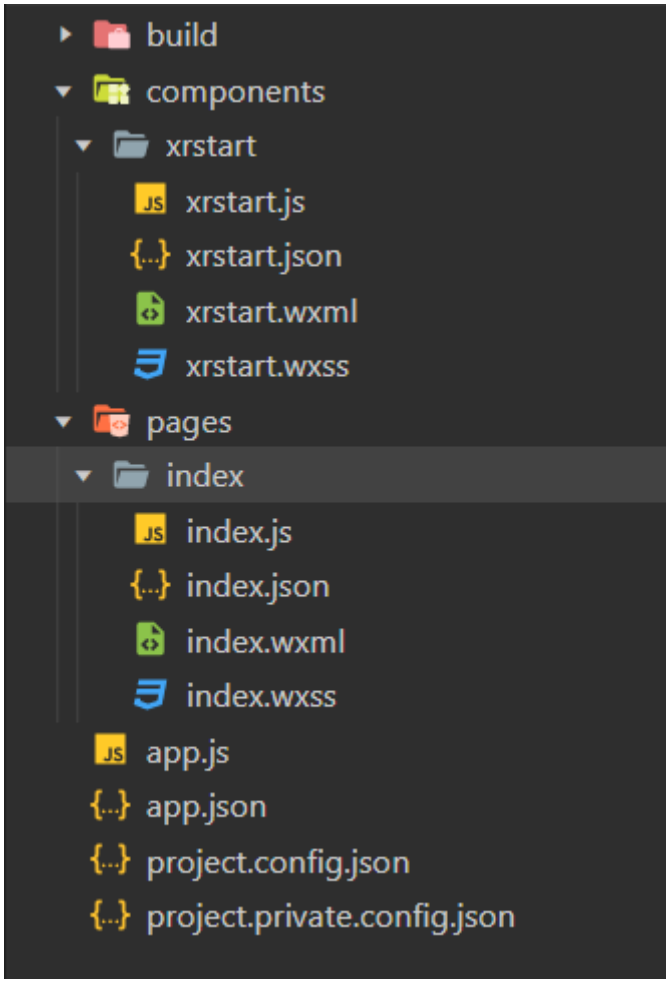
开发过程

步骤1 开启AR相机。

1. 在工具的资源管理器，单击右键，选择“新建文件夹”，命名为“components”。
2. 选择“components”文件夹，单击右键，选择“新建文件夹”，命名为“xrstart”。
3. 选择“xrstart”文件夹，单击右键，选择“新建Component”，命名为“xrstart”。

   Component建完后，目录结构如图3-8。

图 3-8 目录结构



4. 在xrstart.json文件中，添加renderer配置项。

```
{
  "component": true,
  "renderer": "xr-frame",
  "usingComponents": {}
}
```

5. 在xrstart.wxml文件中，添加camera、light节点。

```
<xr-scene ar-system="modes:Plane;" >
  <xr-node node-id="cameraParent" position="0 0 0" rotation="0 0 0">
    <xr-camera id="camera" node-id="camera" clear-color="0.925 0.925 0.925 1" background="ar"
far="2000" is-ar-camera></xr-camera>
  </xr-node>
  <xr-node node-id="lights">
    <xr-light type="ambient" color="1 1 1" intensity="1" />
    <xr-light type="directional" rotation="180 0 0" color="1 1 1" intensity="3" />
  </xr-node>
</xr-scene>
```

6. 在index.json文件中，引用新建的xrstart组件。

```
{
  "usingComponents": {
    "xr-start": "../../components/xrstart/xrstart"
  }
}
```

7. 在index.js文件中，配置高度和宽度参数。

```
import { XRClient } from "../../build/XRClient";
Page({
  data: {
```



```
    width: 300,
    height: 300,
    renderWidth: 300,
    renderHeight: 300
  },
  onLoad() {
    console.log(XRClient.getVersion())
    const info = wx.getSystemInfoSync();
    const width = info.windowWidth;
    const height = info.windowHeight;
    const dpi = info.pixelRatio;
    this.setData({
      width, height,
      renderWidth: width * dpi,
      renderHeight: height * dpi
    });
  }
})
```

8. 在index.wxml中设置xr-start组件高度和宽度。

```
<view>
  <xr-start disable-scroll
  id="main-frame"
  width="{{renderWidth}}"
  height="{{renderHeight}}"
  style="width:{{width}}px;height:{{height}}px;"
  />
</view>
```

9. 在app.json中配置lazyCodeLoading。

```
{
  "pages": [
    "pages/index/index"
  ],
  "lazyCodeLoading": "requiredComponents"
}
```

10. 单击“真机调试”，小程序会开启AR相机，在手机上可看到相机拍摄到的现实环境画面。

图 3-9 开启 AR 相机

步骤2 接入视觉定位。

1. 在工具的资源管理器，单击右键，选择“新建文件夹”，命名为“utils”。

2. 选择utils文件夹，单击右键，选择“新建文件”，命名为“GlobalBus.ts”。
3. 在GlobalBus.ts文件中，配置事件码。

```
export class GlobalBus {
  //vps定位结果事件码
  public static VPS_RESULT: string = 'VPS_RESULT';
  //vps状态事件码
  public static VPS_TRACKING: string = 'VPS_TRACKING';
  //scene实例事件码
  public static SCENE_INSTANCE: string = 'SCENE_INSTANCE';
}
```

4. 选择utils文件夹，单击右键，选择“新建文件”，命名为“Logger.ts”。
5. 在Logger.ts文件中，实现ILog接口。

```
export class Logger implements ILog {
  private logLevel: number = 3;
  private isSaveLog: boolean = false;
  private logSaveFolderPath: string = `${wx.env.USER_DATA_PATH}/miniprogramARSDK/`;
  private logSaveFileName: string = "";
  init() {
    let time:any= this.formatTime(new Date())
    time=time.replaceAll("/","_").replaceAll(" ","_").replaceAll(":","_")
    this.logSaveFileName=time+".log"
    this.createLogFile();
  }
  setIsSaveLog(isSave: boolean) {
    this.isSaveLog = isSave;
  }
  setLogOutputLevel(level: number) {
    this.logLevel = level;
  }
  error<T>(message: T) {
    if (this.logLevel < 1) return;
    const formartLog= this.formatLog("ERROR",message)
    console.error(formartLog);
    this.logToFile( formartLog);
  }
  warn<T>(message: T) {
    if (this.logLevel < 2) return;
    const formartLog= this.formatLog("WARN",message)
    console.warn(formartLog);
    this.logToFile( formartLog);
  }
  log<T>(message: T) {
    if (this.logLevel < 3) return;
    const formartLog= this.formatLog("LOG",message)
    console.log(formartLog);
    this.logToFile(formartLog);
  }
  createLogFile() {
    if(!this.isSaveLog)return;
    let self = this;
    let fsm = wx.getFileSystemManager()
    fsm.access({
      path: self.logSaveFolderPath,
      success() { },
      fail() {
        fsm.mkdirSync(self.logSaveFolderPath, true)
      }
    })
    fsm.access({
      path: self.logSaveFolderPath + self.logSaveFileName,
      success() { },
      fail() {
        fsm.writeFileSync(self.logSaveFolderPath + self.logSaveFileName, "", `utf-8`);
      }
    })
    console.log(`create logfile:${self.logSaveFolderPath + self.logSaveFileName}`)
  }
```

```
  logToFile(message: string) {
    if (!this.isSaveLog) return;
    let fsm = wx.getFileSystemManager()
    let self = this;
    fsm.access({
      path: self.logSaveFolderPath + self.logSaveFileName,
      complete() {
        fsm.appendFile({
          filePath: self.logSaveFolderPath + self.logSaveFileName,
          data: message,
          encoding: 'utf8',
          success: () => { },
          fail: (res) => { console.log(res.errMsg); }
        })
      }
    })
  }
  formatLog<T>(level: string, message: T): string {
    const date = new Date();
    const formattedDate=this.formatTime(date)
    const formattedMessage: string = `[${formattedDate}][${level}]: ${message}\n`;
    return formattedMessage
  }
  formatNumber = (n: number) => {
    const s = n.toString()
    return s[1] ? s : '0' + s
  }
  formatTime = (date: Date) => {
    const year = date.getFullYear()
    const month = date.getMonth() + 1
    const day = date.getDate()
    const hour = date.getHours()
    const minute = date.getMinutes()
    const second = date.getSeconds()

    return (
      [year, month, day].map(this.formatNumber).join('/') +
      ' ' +
      [hour, minute, second].map(this.formatNumber).join(':')
    )
  }
}
```

6. 在app.js文件中初始化logger。

```
import { Logger } from "./utils/Logger";
import { XRClient } from "./build/XRClient";
App({
  onLaunch() {
    const logger=new Logger()
    XRClient.initLogger(logger)
    XRClient.log('onLaunch');
  },
})
```

7. 在xrstart.js文件中，初始化Vps，侦听Vps结果并设置回调函数。

```
import { XRClient } from "../../build/XRClient";
import { GlobalBus } from "../../utils/GlobalBus";
Component({
  methods: {
    handleReady({ detail }) {
      this.scene = detail.value;
      this.offsetSetted = false;
      XRClient.log("handleReady");
    },
    handleARReady: function ({ }) {
      XRClient.log('arReady: arVersion ' + this.scene.ar.arVersion);
      XRClient.log('arReady: version ' + this.scene.ar.version);
      // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量
中密文存放，使用时解密，确保安全；
      // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量
```



```
HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    let config = {
      access: process.env.HUAWEICLOUD_SDK_AK,
      secret: process.env.HUAWEICLOUD_SDK_SK,
      beforeLocateDelta: 15,
      afterLocateDelta: 60,
      autoRequest: true,
      workerPath: 'build/workers/ImageProcessWorker.js',
      url:''
    }
    XRClient.init(config);
    XRClient.dispatchEvent(GlobalBus.SCENE_INSTANCE, { info: this.scene });
    XRClient.addEvent(GlobalBus.VPS_RESULT, this.onVpsResult, this);
    this.arInited = true;
  },
  handleTick() {
    XRClient.updatePerFrame();
  },
  onVpsResult(evt) {
    XRClient.log("onVpsResult status" + evt.data.status);
    XRClient.log("onVpsResult info" + evt.data.info);
    if (evt.data.status) {
      XRClient.log("VPS success");
      wx.showToast({
        title: '定位成功',
        icon: 'success',
        duration: 2000
      })
    } else {
      XRClient.log("VPS failure");
      wx.showToast({
        title: '定位失败',
        icon: 'none',
        duration: 2000
      })
    }
  },
},
})
```

说明

代码中url固定为：https://koomap.cn-north-4.myhuaweicloud.com。

8. 在xrstart.wxml文件里绑定函数。

```
<xr-scene ar-system="modes:Plane;" bind:ready="handleReady" bind:ar-ready="handleARReady"
bind:tick="handleTick">
```

9. 在app.json文件里添加地理位置permission并配置workers。

```
{
  "pages": [
    "pages/index/index"
  ],
  "permission": {
    "scope.userLocation": {
      "desc": "为了给您提供更好的服务，请授权您的地理位置信息"
    }
  },
  "workers": "build/workers",
  "lazyCodeLoading": "requiredComponents"
}
```

10. 在project.config.json文件的setting中，配置useCompilerPlugins。

```
"useCompilerPlugins": [
    "typescript"
]
```

11. 单击“真机调试”，小程序会自动发起定位，根据定位结果后，会弹出Toast提示是否定位成功。如定位失败，可参见**常见问题排查**章节确定原因。

步骤3 渲染数字内容。

1. 选择utils文件夹，单击右键，选择“新建文件”，命名为“ModelMag.ts”
2. 在“ModelMag.ts”文件中，添加渲染逻辑。

```
import { XRClient } from "../build/XRClient";
import { GlobalBus } from "./GlobalBus";
let xrFrameSystem
class ModelMag {
  private scene: any;
  modelRoot: any;
  gltfModel: any[] = [];
  digitalInfo: any;
  count = 0;
  private utmPositionJson = "[{\"position\":{\"x\":0,\"y\":0,\"z\":0}},{\"position\":{\"x\":0,\"y\":0,\"z
\":2}}]"; //在utmPositionJson中配置数字内容的坐标，定位成功后接口会返回用户当前位置的坐标信息，
可以将数字内容坐标设置在用户附近，以便于调试。
  constructor(scene: any) {
    this.scene = scene;
    xrFrameSystem = wx.getXrFrameSystem();
    const jsonData = JSON.parse(this.utmPositionJson);
    let node = this.scene.createElement(xrFrameSystem.XRNode);
    this.modelRoot = node;
    this.scene.addChild(node);
    this.digitalInfo = jsonData;
    this.setCube();
    XRClient.addEvent(GlobalBus.VPS_TRACKING, this.onVpsTrackingState, this);
  }
  public setPosition(xOffset, yOffset, zOffset) {
    let trs = this.modelRoot.getComponent(xrFrameSystem.Transform);
    trs.setData({ visible: true })
    XRClient.log("set digital root" + xOffset + " " + yOffset + " " + zOffset);
    trs.position.setValue(xOffset, yOffset, zOffset);
  }
  setCube() {
    const geometry = this.scene.assets.getAsset('geometry', 'cube');//以方块为例，也可替换为.glb格式的
模型
    const effect = this.scene.assets.getAsset('effect', 'standard');
    const beOccMaterial = this.scene.createMaterial(effect);
    beOccMaterial.setVector('u_baseColorFactor', xrFrameSystem.Vector4.createFromNumber(1.0, 0, 0,
1.0));
    this.modelRoot.getComponent(xrFrameSystem.Transform).setData({ visible: false })
    for (let i = 0; i < this.digitalInfo.length; i++) {
      let item = this.digitalInfo[i];
      const cube = this.scene.createElement(xrFrameSystem.XRNode);
      cube.addComponent(xrFrameSystem.Mesh, {
        geometry: geometry,
        material: beOccMaterial,
      });
      this.modelRoot.addChild(cube);
      const trs = cube.getComponent(xrFrameSystem.Transform);
      trs.position.setValue(item.position.x, item.position.y, -item.position.z);
      XRClient.log("set " + i + " cube posi:" + trs.position.x + " " + trs.position.y + " " + trs.position.z);
    }
  }
  onVpsTrackingState(evt) {
    if (evt.data.status) {
      XRClient.log("show digital");
      let modelRootTrs = this.modelRoot.getComponent(xrFrameSystem.Transform);
      modelRootTrs.setData({ visible: true })
    } else {
      XRClient.log("hide digital");
      let modelRootTrs = this.modelRoot.getComponent(xrFrameSystem.Transform);
      modelRootTrs.setData({ visible: false })
    }
  }
  public destroy(){
    XRClient.removeEvent(GlobalBus.VPS_TRACKING, this.onVpsTrackingState);
  }
```



```
}
export default ModelMag;
```

3. 在index.js文件中实例化ModelMag。

```
import { XRClient } from "../../build/XRClient";
import { GlobalBus } from "../../utils/GlobalBus";
import modmag from "../../utils/ModelMag";
Component({
  methods: {
    handleReady({ detail }) {
      this.scene = detail.value;
      this.offsetSetted = false;
      XRClient.log("handleReady");
    },
    handleARReady: function ({ }) {
      XRClient.log('arReady: arVersion ' + this.scene.ar.arVersion);
      XRClient.log('arReady: version ' + this.scene.ar.version);
      // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
      // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
      let config = {
        access: process.env.HUAWEICLOUD_SDK_AK,
        secret: process.env.HUAWEICLOUD_SDK_SK,
        beforeLocateDelta: 15,
        afterLocateDelta: 60,
        autoRequest: true,
        workerPath: 'build/workers/ImageProcessWorker.js',
        url:''
      }
      XRClient.init(config);
      XRClient.dispatchEvent(GlobalBus.SCENE_INSTANCE, { info: this.scene });
      XRClient.addEvent(GlobalBus.VPS_RESULT, this.onVpsResult, this);
      this.arInited = true;
      let mod = new modmag(this.scene);
      this.mod = mod;
    },
    handleTick() {
      XRClient.updatePerFrame();
    },
    onVpsResult(evt) {
      XRClient.log("onVpsResult status" + evt.data.status);
      XRClient.log("onVpsResult info" + evt.data.info);
      if (evt.data.status) {
        XRClient.log("VPS success");
        wx.showToast({
          title: '定位成功',
          icon: 'success',
          duration: 2000
        })
        if (!this.offsetSetted) {
          this.offsetSetted = true;
          let arr = XRClient.getCameraOffset();
          this.mod.setPosition(-arr[0], -arr[2], arr[1]);
        }
      } else {
        XRClient.log("VPS failure");
        wx.showToast({
          title: '定位失败',
          icon: 'none',
          duration: 2000
        })
      }
    },
  },
  lifetimes:{
    attached(){
      XRClient.log("attached");
    },
    detached(){
```

```
    XRClient.log("detached");
    if(this.arInited)
    {
      XRClient.destroyVps();
      XRClient.removeEvent(GlobalBus.VPS_RESULT, this.onVpsResult);
      this.mod.destroy();
    }
  }
 }
})
```

4. 单击“真机调试”，定位成功后，在空间中显示红色方块。

图 3-10 显示红色方块

----结束

3.3.4 注意事项

如果您无法获取视频流，建议排查以下问题：

- 检查小程序相机权限是否打开。
- 检查移动设备后置摄像头是否能正常拍摄画面。

3.4 API 列表

3.4.1 调用前准备工作

XRLightSDK封装类名为“XRClient”，所有API均直接通过类“XRClient”来调用，无需实例化。

在调用SDK前，需在文件头部引入XRClient类。

```
import { XRClient } from "../../SDK/XRClient";
```

3.4.2 视觉定位

接口列表

表 3-2 视觉定位接口列表

接口	描述	参数名	参数类型	参数说明	返回值

init	初始化视觉定位模块。	config	{
			access: string,
			secret: string,
			beforeLocateDelta: number,
			afterLocateDelta: number,
			autoRequest: boolean,
			workerPath: string,
			url: string
			}
				初始化配置参数。
				● access：认证用的AK。
				● secret：认证用的SK。
				● beforeLocateDelta：首次定位成功前每隔几秒触发一次自动请求。
				● afterLocateDelta：定位成功后每隔几秒触发一次自动请求。
				● autoRequest：是否开启Vps自动请求。
				● workerPath：Vps模块用到的worker文件路径。
				● url：触发定位请求时使用的域名。
					void

requestVps	触发定位请求。	-	-	-	void
updatePerFrame	更新相机位姿。	-	-	-	void
getCameraOffset	获取相机偏移量。	-	-	-	number[3]

接口	描述	参数名	参数类型	参数说明	返回值
getUTMPosition	获取UTM（UNIVERSAL TRANSVERSE MERCARTOR GRID SYSTEM，通用横墨卡托格网系统）坐标。	-	-	-	number[3]
destroyVps	销毁视觉定位实例。	-	-	-	void
mockVps	模拟定位。	position	Vector3	位置，使用UTM坐标系。	void
		rotation	Vector3	朝向，以弧度制表示的欧拉角。	

接口调用示例

- **初始化视觉定位模块**

  设置初始化参数：

```
handleReady({ detail }) {
    this.scene = detail.value; //获取场景实例
},
handleARReady: function ({ }) {
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
     let config = {
       access: process.env.HUAWEICLOUD_SDK_AK,
       secret: process.env.HUAWEICLOUD_SDK_SK,
       beforeLocateDelta: 15,
       afterLocateDelta: 60,
       autoRequest: false,
       workerPath: 'SDK/workers/ImageProcessWorker.js',
       url:''
     }
     XRClient.init(config);//初始化视觉定位模块
     XRClient.dispatchEvent(GlobalBus.SCENE_INSTANCE, { info: this.scene });//派发scene实例事件
     XRClient.addEvent(GlobalBus.VPS_RESULT, this.onVpsResult,this);
},
```

  设置相机的节点：需在xr-start组件的.wxml文件中，设置ar-system，并配置ready、ar-ready和tick回调函数。

```
<xr-scene ar-system="modes:Plane;" bind:ready="handleReady" bind:ar-ready="handleARReady" bind:tick="handleTick">
  <xr-node node-id="cameraParent" position="0 0 0" rotation="0 0 0">
    <xr-camera id="camera" node-id="camera" clear-color="0.925 0.925 0.925 1" background="ar" far="2000" is-ar-camera></xr-camera>
  </xr-node>
</xr-scene>
```

- **触发定位请求**

  手动触发定位请求，获取当前位置。**在手动触发请求时，如果自动请求正在处理中，则此次手动请求将视为无效。**

  ```
  XRClient.requestVps();//返回定位结果。
  ```

- **更新相机位姿**

  在xr-start组件的handleTick里调用该函数，更新相机位姿。

  ```
  handleTick() {
      XRClient.updatePerFrame(); //结合定位结果、AR算法更新相机的位置和姿态以及计时处理。
  },
  ```

- **获取相机偏移量**

  将相机和数字内容等量偏移至较小值，避免过大的坐标值引起一些渲染上的异常（例如模型闪烁，抖动等）。

  ```
  let arr = XRClient.getCameraOffset(); //返回记录的偏移量，使用该偏移量对数字内容进行偏移，得到正确的渲染结果。
  this.mod.setPosition(arr[0], arr[1], arr[2]);
  ```

- **获取UTM坐标**

  ```
  XRClient.getUTMPosition();//函数返回当前时刻的UTM坐标，数组长度为3。
  ```

- **销毁视觉定位实例**

  ```
  lifetimes:{
      detached(){
        if(this.arInited){
          XRClient.destroyVps();//销毁Vps实例，在xr-start组件的detached方法中调用。
          XRClient.removeEvent(GlobalBus.VPS_RESULT, this.onVpsResult);
        }
      }
  },
  ```

- **模拟定位**

  如果开发者在办公区进行调试，但办公区没有进行地图采集，无法定位成功时，可使用模拟定位接口。

  ```
  let posi = { x: 366676, y: 3457911, z: 43 };
  let rot = { x: 0, y: 1.57, z: 0 };
  XRClient.mockVps(posi, rot);
  ```

  说明

  - 模拟定位情况下，使用输入的位置和旋转作为定位结果。
  - XRLightSDK会关闭自动定位，避免频繁跳变而影响调试。

3.4.3 事件

接口列表

表 3-3 事件接口列表

接口	描述	参数名	参数类型	参数说明	返回值
addEvent	侦听事件。	evtName	string	事件ID。 根据实际情况选择Vps定位结果事件ID、Vps状态事件ID或scene实例事件ID。	void

接口	描述	参数名	参数类型	参数说明	返回值
		cb	Function	侦听到事件后，触发的回调函数。	
		cbThis	any	当前实例。	
removeEvent	移除侦听事件。	evtName	string	事件ID。 根据实际情况选择Vps定位结果事件ID、Vps状态事件ID或scene实例事件ID。	void
		cb	Function	回调函数。	
dispatchEvent	派发事件。	evtName	string	事件ID。 根据实际情况选择Vps定位结果事件ID、Vps状态事件ID或scene实例事件ID。	void

		params	{
			key: value,
			key2: value2
			}
				事件携带的信息，键值对形式。	

接口调用示例

- **侦听事件**

  用于侦听事件，并触发侦听函数。

```
XRClient.addEvent(GlobalBus.VPS_RESULT, this.onVpsResult,this);
```

  定义侦听函数。

```
onVpsResult(evt) {
    XRClient.log("onVpsResult status" + evt.data.status);
    XRClient.log("onVpsResult info" + evt.data.info);
    if (evt.data.status) {
      XRClient.log("VPS success");
      wx.showToast({
        title: 'success',
        icon: 'success',
        duration: 2000
      })
    } else {
      XRClient.log("VPS failure");
      wx.showToast({
        title: 'failure',
        icon: 'none',
        duration: 2000
      })
    }
},
```

- **移除侦听事件**

  用于移除事件，移除后侦听函数不再触发。

  ```
  XRClient.removeEvent(GlobalBus.VPS_RESULT, this.onVpsResult);
  ```

- **派发事件**

  用于派发事件，配合侦听事件使用，接收到此处派发的事件，并做相应处理。

  ```
  XRClient.dispatchEvent('user_define', { status: false, info: 'test' } );
  ```

SDK 内部事件

表 3-4 SDK 内部事件列表

事件名称	描述	携带数据	数据类型	数据说明	事件说明
VPS_RESULT	Vps定位结果事件。	status	boolean	定位是否成功。	Vps定位结果返回时触发该事件。
		info	string	具体定位结果信息。	
VPS_TRACKING	Vps跟踪状态事件。	status	boolean	状态是否为跟踪中。	Vps跟踪状态变化时触发。
SCENE_INSTANCE	scene实例事件。	info	object	scene实例。	初始化视觉定位模块时，派发scene实例事件。

3.4.4 日志

接口列表

表 3-5 日志接口列表

接口	描述	参数名	参数类型	参数说明	返回值
initLogger	初始化日志模块	logger	ILog	日志的实例	void
setIsSaveLog	日志存储	isSave	boolean	是否存储日志	void
setLogOutputLevel	设置日志打印级别	level	number	打印日志的级别	void
log<T>	打印普通日志	message	T	日志信息	void

接口	描述	参数名	参数类型	参数说明	返回值
warn<T>	打印警告日志	message	T	日志信息	void
error<T>	打印错误日志	message	T	日志信息	void

接口调用示例

- **初始化日志模块**
  ```
  const logger=new Logger() //创建Logger实例
  XRClient.initLogger(logger)//初始化日志模块
  ```
- **日志存储**
  ```
  XRClient.setIsSaveLog(true)
  ```
  - 设置日志存储为“true”，打印日志的同时会写入日志文件到手机（**仅Android支持，iOS暂不支持**）。
  - 设置日志存储为“false”，不写入日志文件到手机。
- **设置日志打印级别**
  ```
  XRClient.setLogOutputLevel(1)
  ```
  日志级别包括：0（不输出）、1（error）、2（warn）、3（log）
  - 设置日志级别为0：不输出日志。
  - 设置日志级别为1：只输出error级别日志。
  - 设置日志级别为2：输出error和warn二个级别日志。
  - 设置日志级别为3：输出error、warn、log三个级别日志。
- **打印普通级别的日志**
  ```
  XRClient.log("打印内容")
  ```
- **打印警告级别的日志**
  ```
  XRClient.warn("打印内容")
  ```
- **打印错误级别的日志**
  ```
  XRClient.error("打印内容")
  ```

3.4.5 其他

接口列表

表 3-6 其他接口列表

接口	描述	参数名	参数类型	参数说明	返回值
requestNavi	请求导航路径。	utmCode	string	当前utm区域。	void
		from	number[3]	导航出发点utm坐标。	
		to	number[3]	导航目的地utm坐标。	

接口	描述	参数名	参数类型	参数说明	返回值
		policy	number	路线选择策略。 1：最短距离 2：电梯优先 3：扶梯优先 -2：不坐电梯 -3：不走扶梯	
		callback	Function	回调函数，附带导航路径点信息。	
getVersion	获取SDK版本号。	-	-	-	string

接口调用示例

- **请求导航路径**

```
const utmCode = "51N";
const from: number[] = [0, 0, 0];
const to: number[] = [100, 0, 0];
const policy: number = 0;
XRClient.requestNavi(utmCode, from, to, policy, this.RequestNaviCallBack);//返回从出发地到目的地的路径
```

- **获取SDK版本号**

```
XRClient.getVersion();
```

3.5 调试方法

调试工具

- 手机或平板
- 微信开发者工具

调试步骤

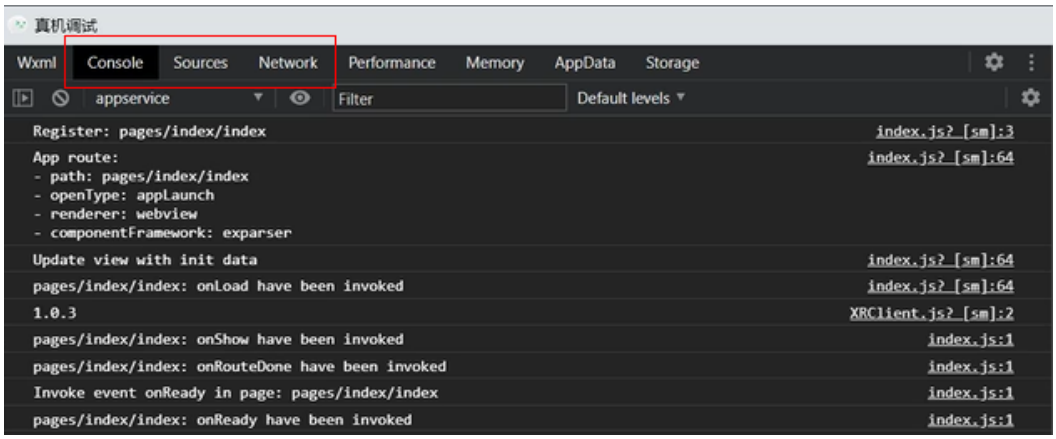
步骤1 登录微信开发者工具。

步骤2 单击“真机调试”，弹出“真机调试”窗口。

步骤3 您可根据实际需求进行操作。

- 在“Console”页签查看日志信息。
- 在“Network”页签查看网络请求发送情况。
- 在“Sources”页签进行断点调试。

图 3-11 真机调试界面



----结束

3.6 常见问题

3.6.1 定位失败，定位结果显示“Incorrect IAM”

检查初始化视觉定位模块config参数中配置的AK、SK是否正确。获取方法请参见**访问密钥**。

3.6.2 定位失败，定位结果显示“OutOfService”

登录**KooMap管理控制台**，检查是否开通了AR地图运行服务，如未开启，请**开通AR地图运行服务**。

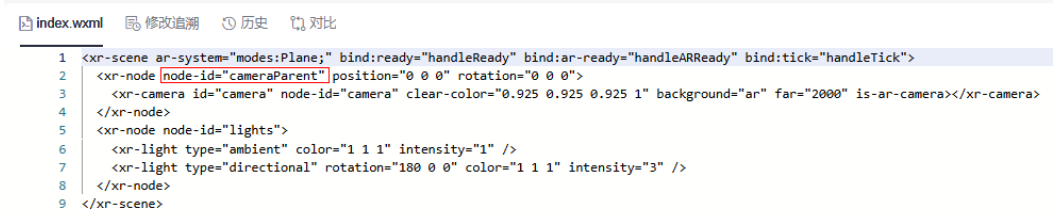
3.6.3 定位失败，定位结果显示“invalid url”

检查初始化视觉定位模块config参数中配置的url是否正确。url为https://koomap.cn-north-4.myhuaweicloud.com。

3.6.4 定位后报错，日志显示“Cannot read properties of undefined（reading ‘position‘）”

在components/xr-start/index.wxml文件里找到xr-camera ，然后在它的上一行加上node-id="cameraParent"的节点，如**图3-12**所示。

图 3-12 增加节点



3.6.5 定位失败，定位结果显示“FewPnPInliers”或“ErrorMeanReprojectionErrTooLarge”

- **原因一：可能是发起定位时使用的图片纹理特征不够明显。**
  解决方案：在移动端，对着纹理丰富的区域单击“手动请求Vps”重新发起定位。
- **原因二：可能是平面识别未初始化完成。**
  解决方案：
  a. 查看camera的position和rotation数值。
     如果position数值为0，且rotation值不更新，说明平面识别未初始化完成。
  b. 对着平面左右平移提升初始化速度，等待初始化完成。
  c. 单击“手动请求Vps”重新发起定位。
- **原因三：移动端不支持V2平面接口。**
  解决方案：
  查看日志确认当前移动端的arVersion版本，如果不是V2版本，则需更换移动端。