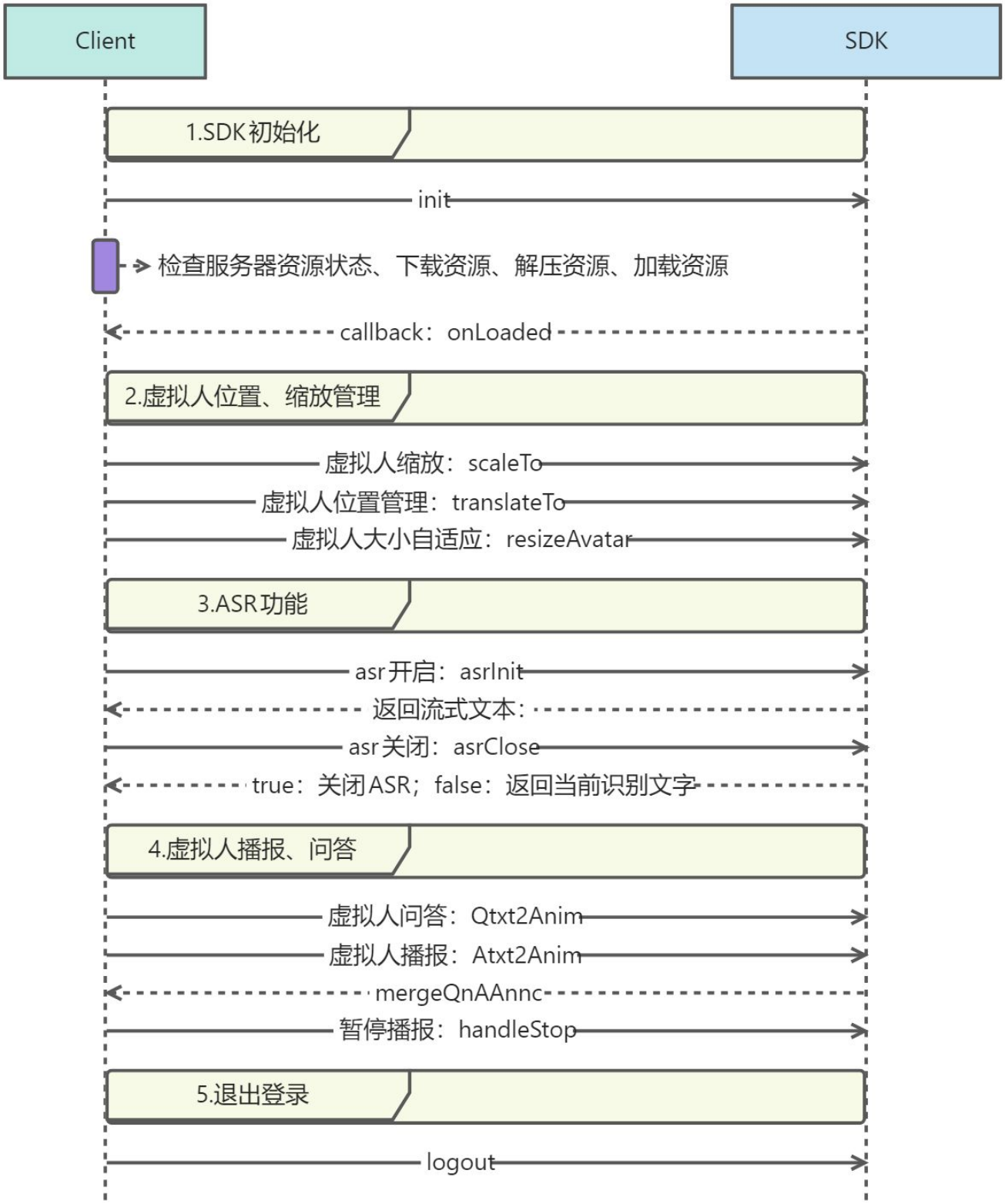
SDK接入指南

【HaiSDK_H5接入指南】

1. 概览

本WebSDK通过JavaScript语言把拟仁智能的虚拟人服务以简洁的方式开放给Web应用开发者，使得Web开发者能够很快在网页、网页应用、微信公众号中快速集成虚拟人形象及其智能服务，给目标用户带来新颖的体验。

1.1. SDK调用时序



2. SDK接入

2.1. 导入依赖模块

```JavaScript
import Haihuman from './haihuman.js'
```

3. SDK初始化

- hairenderer.wasm；hairenderer.js；transcode.worker.js文件需放置在web服务器的根目录下

```JavaScript
const appId = "57279......"
const appKey = "NsUk3kEn......"
const appSecret = "t8rTul......"
const avatarId = "11200010......"
const onLoaded = () => {}
const mergeQnAAnnc = (val:boolean, text:string) => {}
const onError = (err) => {}
const cbs = { mergeQnAAnnc, onLoaded, onError }
let a; // 全局使用
const init = () => {
  isAvatarOk = true // 当前加载完毕没有，没有加载完毕会弹出 正在初始化
  a = new Haihuman({
    appId,
    appKey,
    appSecret,
    avatarId,
    cbs
  }); // 实例化Haihuman对象
  a.init(); // 初始化Haihuman对象
}
// 调用
init();
```

4. 虚拟人位置、缩放、自适应管理

4.1. 虚拟人缩放

```JavaScript
const scaleTo = (i:number) => a.scaleTo(i) // 缩放比例
// 示例： scaleTo(1) || a.scaleTo(1)
```

4.2. 虚拟人位置管理

```JavaScript
const translateTo = (x:number, y:number) => a.translateTo(x, y)
// 示例：translateTo(1,1) || a.translateTo(1,1)
```

4.3. 大小自适应

```JavaScript
const resizeAvatar = () => a.resizeAvatar()
// 示例： window.addEventListener('resize', resizeAvatar) 监听窗口变化 调用人物等比例变化
// 示例： window.addEventListener('resize', a.resizeAvatar) 监听窗口变化 调用人物等比例变化
```

5. ASR功能

5.1. ASR开启

```JavaScript
const asrInit = (call) => a.asrInit(call)
/*
  示例：
  不返回文本 asrInit()
  返回文本 asrInit('', '', (value：string) => {
     // 返回识别到的文本 流式 console.log(value, );
  });
*/
```

5.2. ASR关闭

```JavaScript
const answer = document.getElementById('answer')
const answerText = document.getElementById('answerText')
const asrClose = (state:boolean) => a.asrClose(state)
/*
    state: true 不返回任何问题 不会自动调用问答接口;
           false 会返回当前asr识别文字内部自动调用问答接口
*/
asrClose(true) || a.asrClose(true) // 关闭ASR识别，不返回任何结果
asrClose(false, (value:string)=>{ // value: 返回当前asr识别文字
    if (value) {  // 判断当前 value 是否有值
      answerText.textContent = value;
      answer.style.display = 'flex'
    }
  }) || a.asrClose(false,(value)=>{})
```

6. 虚拟人问答、播报

6.1. 虚拟人问答（Qtxt2Anim）

```JavaScript
Qtxt2Anim: 问答接口
const Qtxt2Anim = (text:string) => a?.Qtxt2Anim(text)
// 示例: (Qtxt2Anim('你叫什么名字') || a?.Qtxt2Anim('你叫什么名字')) 回答: 我是小琪很高兴为您服务
```

6.2. 虚拟人播报（Atxt2Anim）

```JavaScript
Atxt2Anim: 播报接口
const Atxt2Anim = (text:string) => a?.Atxt2Anim(text)
// 示例: (Atxt2Anim('你叫什么名字') || a?.Atxt2Anim('你叫什么名字')) 回答: 你叫什么名字
```

6.2.1. 回调说明:mergeQnAAnnc

- 获取Qtxt2Anim及Atxt2Anim返回的结果

```JavaScript
mergeQnAAnnc：回调说明
const speech = document.getElementById('speech')
const answer = document.getElementById('answer')
const answerText = document.getElementById('answerText')
const mute = document.getElementById('mute')
const mergeQnAAnnc = (val:boolean, text:string) => {
/*
  val:  返回 true || false
        true代表当前在播放语音;
        false代表语音播放结束.
  text：返回当前语音回答文本
*/
  if (!val) {
    answer.style.display = 'none'
    mute.style.display = 'none'
    speech.style.display = 'block'
  } else {
    mute.style.display = 'block'
    speech.style.display = 'none'
    answerText.textContent = text
    answer.style.display = 'flex'
  }
}
```

6.3. 暂停播报

```JavaScript
const handleStop = () => a.handleStop()
// 示例：handleStop || a.handleStop()
```

7. 获取热键列表

```JavaScript
// 获取热键列表
const list = a.hotkeys
```

8. 获取跑马灯列表

```JavaScript
// 获取跑马灯列表
  const list = a.tickerList

// 获取热键列表
const list = a.hotkeys

/**
 * 定义一个热键调用函数
 * @param {string} key - 热键名称
 * @returns 调用指定的热键操作
 */
const callHotkeys = (key) => a.callHotkeys(key)
```

9. 事件绑定

9.1. onLoaded

```JavaScript
onLoaded：人物加载完毕
// 人物加载完毕 执行一定操作
const onLoaded = () => {
  translateTo(0.82, -0.6)  // 调整人物位置
  scaleTo(1.5) // 设置人物大小
  haihuman.style.display = isShow ? 'block' : 'none'; // 显示或隐藏人物
  isAvatarOk = false // 人物加载完毕
}
```

9.2. logout

- 退出页面的时候调用

```JavaScript
// 退出接口
const logout = () => a.logout()
// 页面离开调用
window.addEventListener('beforeunload', logout());
```

10. 示意代码

JavaScript

```javascript
const appId = "57279......"
const appKey = "NsUk3kEn......"
const appSecret = "t8rTul......"
const avatarId = "11200010......"
const answer = document.getElementById('answer')
const answerText = document.getElementById('answerText')
const mute = document.getElementById('mute')
const msg = document.getElementById('msg')
const onLoaded = () => {}
const showValue = (val:true, text:string) => {}
const cbs = { showValue, onLoaded }
let a; // 全局使用
const init = () => {
  isAvatarOk = true // 当前加载完毕没有，没有加载完毕会弹出 正在初始化
  a = new Haihuman({
    appId,
    appKey,
    appSecret,
    avatarId,
    cbs
  }); // 实例化Haihuman对象
  a.init(); // 初始化Haihuman对象
}
// 调用
init();

// 虚拟人缩放
const scaleTo = (i:number) => a.scaleTo(i) // 缩放比例
// 示例： scaleTo(1)

// 人物大小自适应
const resizeAvatar = () => a.resizeAvatar()
// 示例： window.addEventListener('resize', resizeAvatar) 监听窗口变化 调用人物等比例变化

// 虚拟人位置管理
const translateTo = (x:number, y:number) => a.translateTo(x, y)
// 示例：translateTo(1,1)

// ASR开启
const asrInit = (call) => a.asrInit(call)
/*
  示例：
  asrInit()
  asrInit((value: string) => {
```

```
      // 返回识别到的文本 流式 console.log(value, );
  });
*/

// ASR关闭
const asrClose = (state:boolean) => a.asrClose(state)
/*
    state: true 不返回任何问题 不会自动调用问答接口;
           false 会返回当前asr识别文字内部自动调用问答接口
*/
asrClose(true) // 关闭ASR识别，不返回任何结果
asrClose(false, (value:string)=>{ // value: 返回当前asr识别文字
    if (value) {  // 判断当前 value 是否有值
      answerText.textContent = value;
      answer.style.display = 'flex'
    }
  })

// 虚拟人问答
const Qtxt2Anim = (text:string) => a?.Qtxt2Anim(text)
// 示例: Qtxt2Anim('你叫什么名字') 回答: 我是小琪很高兴为您服务

// 虚拟人播报
const Atxt2Anim = (text:string) => a?.Atxt2Anim(text)
// 示例: Atxt2Anim('你叫什么名字') 回答: 你叫什么名字

// 回调说明showValue
const showValue = (val:boolean, text:string) => {
  /*
    val:  返回 true || false
          true代表当前在播放语音;
          false代表语音播放结束.
    text: 返回当前语音回答文本
  */
    if (!val) {
      answer.style.display = 'none'
      mute.style.display = 'none'
      speech.style.display = 'block'
    } else {
      mute.style.display = 'block'
      speech.style.display = 'none'
      answerText.textContent = text
      answer.style.display = 'flex'
    }
  }

// 暂停播报
const handleStop = () => a.handleStop()
```

```
// 示例：handleStop()

// onLoaded 回调说明
// 人物加载完毕 执行一定操作
const onLoaded = () => {
  translateTo(0.82, -0.6)  // 调整人物位置
  scaleTo(1.5) // 设置人物大小
  haihuman.style.display = isShow ? 'block' : 'none'; // 显示或隐藏人物
  isAvatarOk = false // 人物加载完毕
}

// logout 退出登录
const logout = () => a.logout()
// 页面离开调用 示例 window.addEventListener('beforeunload', logout());
```

【HaiSDK_Unity接入指南】

1.概览

对虚拟人基础生命周期管理，对交互接口说明

2.SDK的初始化、更新和退出

2.1 SDK的初始化

```C#
/**
 * 从启动到退出的一次完整操作流程中，SDK的初始化流程只需在主程序启动时执行一次。
 * @param useInnerAudioFlag 是否启用内部音频控制器，如果是false，sdk的调用方可以重写sdk抛出的音频处理回调来达到重写音频控制代码。
 * @param persistPath 需要用户手动设置一个可读写的文件路径给sdk，sdk内部会用这个目录存放存放一些依赖资源。
 * @param animMode 0表示默认动画方式，1是升级动画方式
 * @param authHost  这些参数都是和sdk本身的授权有关，这些参数需要向拟仁申请。
 * @param appKey    这些参数都是和sdk本身的授权有关，这些参数需要向拟仁申请。
 * @param appID     这些参数都是和sdk本身的授权有关，这些参数需要向拟仁申请。
 * @param appSecret 这些参数都是和sdk本身的授权有关，这些参数需要向拟仁申请。
 * @param usrID usrID
 * @param usrID 所有传递给回调函数的用户数据指针
 * @param initEndCallback  结束回调接口
 */
HAISDK_CALL void haisdk_init(int useInnerAudioFlag,
                             const char *persistPath,
                             int animMode,
                             const char *authHost,
                             const char *appKey,
                             const char *appID,
                             const char *appSecret,
                             const char *usrID,
                             HAI_PVOID userData,
                             HAI_FPTR_INIT_END_CALLBACK initEndCallback);
```

2.2 SDK的更新

SDK提供Update来完成SDK的更新

```C#
/**
 * sdk全局更新接口
 */
HAISDK_CALL void haisdk_update();
```

2.3 获取授权的token

```C#
/**
 * 获取授权的token
 */
HAISDK_CALL const char *haisdk_get_authtoken();
```

2.4 SDK的退出

SDK提供了以下调用函数来完成SDK的退出过程

```C#
/**
 * 从启动到退出的一次完整操作流程中，SDK的退出流程只需在主程序退出时执行一次。
 * SDK将会在此函数被调用时完成SDK注销、清理SDK内存占用等一系列操作。
 */
HAISDK_CALL void haisdk_quit();
```

3.渲染画布管理

3.1 添加画布

```C#
/**
 * 添加一个绘制表面
 * @param surface  绘制表面句柄
 * @param width 绘制表面对应的像素宽度
 * @param height 绘制表面对应的像素高度
 * @return 返回值是默认的surfaceID(为非负值)，其他值表示API执行有错误。
 */
HAISDK_CALL int haisdk_add_draw_surface(void *surface,
                                        int width,
                                        int height);
```

3.2 删除画布

```C#
/**
 * 根据一个绘制表面的句柄id，删除一个绘制表面
 * @param surfaceID 绘制表面的句柄id
 */
HAISDK_CALL void haisdk_remove_draw_surface(int surfaceID);
```

3.3 画布背景管理

3.3.1 缩放画布

```C#
/**
 * 缩放一个绘制表面
 * @param width 新的绘制表面对应的像素宽度
 * @param height 新的绘制表面对应的像素高度
 */
HAISDK_CALL int haisdk_resize_draw_surface(int surfaceID,
                                           int width,
                                           int height);
```

3.3.2 启用颜色背景

```C#
/**
 * 启用颜色背景，分别传递rgba的值，与SetBackgroundImage互斥，只有一个会显示。
 * @param r255 0-255整形值
 * @param g255 0-255整形值
 * @param b255 0-255整形值
 * @param a255 0-255整形值
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_background_show_color(int r255,
                                             int g255,
                                             int b255,
                                             int a255);
```

3.3.3 启用贴图背景

```C#
/**
 * 启用贴图背景，传递背景贴图的路径，与SetBackgroundColor互斥，只有一个会显示。
 * @param backgroundImagePath 背景贴图的路径
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_background_show_image(const char *backgroundImagePath);
```

3.3.4 隐藏已显示的背景

```C#
/**
 * 隐藏已经显示的背景
 */
HAISDK_CALL void haisdk_background_hide();
```

3.4 启用单一surface模式

```C#
/**
 * 启用单一surface显示模式，这个模式与MultiSurfaces互斥，只会一个enable
 * @param surfaceID 绘制表面的句柄id
 */
HAISDK_CALL void haisdk_enable_single_surfacemode(int surfaceID);
```

3.5 设置SDK内部的渲染相机的投影方式

3.5.1 透视方式

```C#
/**
 * 设置sdk的内部的渲染相机的投影方式，透视方式。
 */
HAISDK_CALL void haisdk_set_camera_ortho();
```

3.5.2 正交投影

```C#
/**
 * 设置sdk的内部的渲染相机的投影方式，正交投影。
 */
HAISDK_CALL void haisdk_set_camera_perspective(float fov);
```

3.6 全屏动画管理

3.6.1 播放全屏UV动画

```C#
/**
 * 播放一个全屏UV动画
 * @param picPath 图片路径
 * @param uSpeed u方向的动画速度
 * @param vSpeed v方向的动画速度
 * @param targetAlpha uv图片的alpha值
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_picturefx_show_uvanim(const char *picPath,
                                             float uSpeed,
                                             float vSpeed,
                                             float targetAlpha);
```

3.6.2 播放全屏渐变动画

```C#
/**
 * 播放一个全屏的渐变动画
 * @param picPath 图片路径
 * @param blendInTime 融合显示时长
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_picturefx_show_blendanim(const char *picPath,
                                                float blendInTime);
```

3.6.3 停止显示全屏动画

```C#
/**
 * 停止显示一个全屏的图片动画
 */
HAISDK_CALL void haisdk_picturefx_hide();
```

4.角色实例管理

4.1 创建角色实例

```C#
/**
 * 创建数字虚拟人接口
 * @param instId
 * @param avatarID  表示insId对应的avatarID
 * @param langID   多语言id，指定虚拟人是按照什么语言交流
 *                      cn, en, es, fr, de, it, hi
 * @param usrData 所有传递给回调函数的用户数据指针
 * @param fptrStopAudio 音频控制的回调接口，用于API调用层重写音频控制函数，只有Init函数里面useInnerAudioFlag为false的时候，才有可能有回调。
 * @param fptrPauseAudio 音频控制的回调接口，用于API调用层重写音频控制函数，只有Init函数里面useInnerAudioFlag为false的时候，才有可能有回调。
 * @param fptrPlayAudioBuff 音频控制的回调接口，用于API调用层重写音频控制函数，只有Init函数里面useInnerAudioFlag为false的时候，才有可能有回调。
 * @param fptrPlayAudioFile 音频控制的回调接口，用于API调用层重写音频控制函数，只有Init函数里面useInnerAudioFlag为false的时候，才有可能有回调。
 * @param createAvatarCallback 虚拟人创建是否成功的异步回调。这个回调函数会返回两个参数，一个参数表示Instance是否创建成功，另一个表示创建成功的Instance实例ID。
 */
HAISDK_CALL void haisdk_avatar_create(const char *insID,
                                      const char *avatarID,
                                      const char *langID,
                                      HAI_PVOID usrData,
                                      HAI_FPTR_STOP_AUDIO_CALLBACK fptrStopAudio,
                                      HAI_FPTR_PAUSE_AUDIO_CALLBACK fptrPauseAudio,
                                      HAI_FPTR_PLAY_PCM_CALLBACK fptrPlayAudioBuff,
                                      HAI_FPTR_PLAY_AUDIOFILE_CALLBACK fptrPlayAudioFile,
                                      HAI_FPTR_CREATE_AVATAR_CALLBACK createAvatarCallback);
```

4.3 加载或替换全身

```C#
/**
 * 加载或者替换全身
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param wholeAssetName 新身体的资源名称
 * @return
 */
HAISDK_CALL int haisdk_avatar_load_whole_asset(int ava_inst_id,
                                               const char *wholeAssetName);
```

4.4 卸载全身

```C#
/**
 * 卸载全身
 * @param ava_inst_id haisdk_avatar_create返回的id值
 */
HAISDK_CALL int haisdk_avatar_unload_whole_asset(int ava_inst_id);
```

4.4 删除角色实例

```C#
/**
 * 删除数字虚拟人
 * @param ava_inst_id haisdk_avatar_create返回的id值
 */
HAISDK_CALL void haisdk_avatar_destroy(int ava_inst_id);
```

5.角色实例管理

5.1. 获取虚拟人实例的位置参数

```C#
/**
 * 获取虚拟人实例的位置参数
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @return float[7] 0:scale, [1..3]:rotation, [4..6]: position
 */
HAISDK_CALL float *haisdk_avatar_get_transformation(int ava_inst_id);
```

5.1.1 指定虚拟人走到某个相对位置

```C#
/**
 * 指定虚拟人走动到某个相对位置
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param moveMotionName 设定角色再移动的过程中的动作
 * @param delta_x 设定角色在水平左右方向上相对x轴向值;
 * @param duration 设定角色走动的时长;
 * @param moveUserData 所有传递给回调函数的用户数据指针
 * @param moveEndCallback 行走结束回调;
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_startwalkby(int ava_inst_id,
                                          float delta_x,
                                          float duration,
                                          HAI_PVOID moveUserData,
                                          HAI_FPTR_VOID_PVOID moveEndCallb
ack);
```

5.1.2 将角色由当前位置直接移动到指定的3d世界坐标位置

```C#
/**
 * 将角色由当前位置直接移动到指定的3d世界坐标位置
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param x 设定角色在水平方向上的3D位置坐标;
 * @param y 设定角色在垂直方向上的3D位置坐标;
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_translate_to(int ava_inst_id,
                                           float x,
                                           float y);
```

5.1.3 将角色由当前位置直接移动到当前角色的3d世界坐标+(deltax, deltay)的位置处

```C#
/**
 * 将角色由当前位置直接移动到当前角色的3d世界坐标+(deltax, deltay)的位置处。
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param deltax 沿水平方向的移动距离，当值为正时，表示向右移动，否则向左移动;
 * @param deltay 沿垂直方向的偏移距离，当值为正时，表示向上移动，否则向下移动;
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_translate_by(int ava_inst_id,
                                           float deltax,
                                           float deltay);
```

5.1.4 正交相机条件下将角色坐标按照窗口大小比率进行移动

```C++
/**
 * 正交相机条件下将角色坐标按照窗口大小比率进行移动
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param x 范围[-1, 1],x=0时，角色处于窗口x轴中心位置，x=-1时，处于窗口最左边，x=1时,处于窗口最右边
 * @param y 范围[-1, 1],y=0时，角色处于窗口y轴中心位置，y=-1时，处于窗口最底部，y=1时,处于窗口最顶部
 * @return 返回值大于0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_unit_translate_to(int ava_inst_id,
                                                float x,
                                                float y);
```

5.2 角色旋转管理

5.2.1 将角色基于当前方位旋转指定角度

```C#
/**
 * 将角色基于当前方位旋转指定角度
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param angle 相对的旋转角度值，当值为正时，表示逆时针转动（俯视视角），否则表示顺时针转动。
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_rotate_by(int ava_inst_id,
                                        float angle);
```

5.2.2 将角色基于当前方位旋转至指定角度

```C#
/**
 * 将角色基于当前方位旋转至指定角度
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param angle 设定的角度值，取值范围是（-180，180）。
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_rotate_to(int ava_inst_id,
                                        float angle);
```

5.3 角色缩放管理

5.3.1 将角色基于现有大小缩放至指定的大小

```C++
/**
 * 将角色基于现有大小缩放至指定的大小
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param percentage 相对于原始模型大小的缩放系数，当值大于1时，表示放大，否则表示缩小。
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_scale_to(int ava_inst_id,
                                       float percentage);
```

5.3.2 正交相机条件下将角色按照窗口高度比率进行缩放

```C++
/**
 * 正交相机条件下将角色按照窗口高度比率进行缩放
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param surfaceID 绘制表面的句柄id
 * @param percentage 缩放到窗口高度的比率，范围(0, 1.0]
 * @return 返回值为大于0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_scale_to_surface_percent(int ava_inst_id,
                                                       int surfaceID,
                                                       float percentage);
```

6.通用动画管理

6.1 动画播放控制–播放原子动画

C#

```
/**
 * 将角色行为状态切换到播放原子级的分成动画状态
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param animName 动画名称
 * @param loop 循环标志位
 * @param forcePlayClipState true表示中断其他虚拟人状态，直接进入单纯播放motion的状态
 * @param usrData 所有传递给回调函数的用户数据指针
 * @param playCallback 通知动画播放结束时间的回调处理函数；原子动画不含音频信息，因此不支持通知开始播放wav音频时间的回调处理函数。如果原子动画设置了循环播放的标志位，结束回调就不会再调用了
 * @return 返回非负值为当前要播放的动画实例ID，负值表示播放错误
 */
HAISDK_CALL int haisdk_avatar_anim_playclip(int ava_inst_id,
                                            const char *animName,
                                            int loop,
                                            int forcePlayClipState,
                                            HAI_PVOID usrData,
                                            HAI_FPTR_VOID_PVOID_INT playCallback);
```

6.2 角色口型与行为情绪管理

```C#
/**
 * 将角色行为状态切换到动态播报状态，同时输出口型对齐结果和行为情绪分析结果。
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param text 播报的文本，可以是问题文本，也可以是答案文本。 *如果只是答案文本，请将openNLP设置成false。后面角色播报的时候，只是播报这个输入的文本。* 如果是问题文本，请将openNLP设置成true。后面角色播报的时候，会播报针对这个问题经过自然语言处理之后的答案文本。
 * @param openNLP 是否启用自然语言处理。
 * @param openMotionTag 是否启用行为动作分析处理。
 * @param inLangID    输入文本的语言id，指定虚拟人是按照什么语言交流
 *                    cn, en, es, fr, de, it, hi
 * @param outLangID   目标播报语言id，指定虚拟人是按照什么语言交流
 *                    cn, en, es, fr, de, it, hi
 * @param tags 可以主动设置一些tag。
 * @param chatUserData 所有传递给回调函数的用户数据指针
 * @param chatEndCallback 结束播报的回调函数。
 * @param chatTagCallback 中间结果，返回的tag回调函数。
 * @param chatTextCallback 中间结果，返回的文本答案回调函数。
 * @param chatRichCallback 中间结果，返回的富文本回调函数。
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_anim_playdynamic(int ava_inst_id,
                                               const char *text,
                                               int openNLP,
                                               int openMotionTag,
                                               const char *inLangID,
                                               const char *outLangID,
                                               const char *tags,
                                               HAI_PVOID chatUserData,
                                               HAI_FPTR_VOID_PVOID chatEndCallback,
                                               HAI_FPTR_VOID_PVOID_PCHAR chatTagCallback,
                                               HAI_FPTR_VOID_PVOID_PCHAR chatTextCallback,
                                               HAI_FPTR_VOID_PVOID_PCHAR chatRichCallback);

```

7.定制逻辑动画

虚拟人SDK还提供了一些方便外部调用的、具有高度定制化特征的逻辑动画调用接口，这类动画接口的共同特点是：

(1)使用频率高：例如在语音交互过程中，需要频繁地让虚拟形象在倾听、回答、待机等多个动画状态之间来回切换；再如在停止状态下，经常需要让虚拟形象以走路的形式移动到终端屏幕的某个位置。

(2)定制性强：理论上，所有定制逻辑动画接口，都可以由逻辑程序员通过调用上一节中介绍的通用动画管理接口予以实现（也可能需要与角色方位管理接口配合）；虚拟人SDK中提供这类接口的主要目的，是方便逻辑程序员的调用，并起到一定的代码示范作用。

(3)可自行编写或扩展：仿照定制逻辑动画接口的实现方法，逻辑端程序员完全可以自己在java层实现更多、更复杂的定制逻辑动画函数，从而满足自身项目的、千变万化的实际需求。

以下针对SDK中已经提供的定制逻辑动画接口分别予以介绍

7.1.将角色行为状态切换到Idle状态

以下接口用于调用待机动画，通常用于在语音交互过程中打断原本正在播报的其他动画，为接收新的用户指令做好准备。

C#

```
/**
 * 将角色行为状态切换到 Idle 状态
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_do_standby(int ava_inst_id);
```

7.2.将角色行为状态切换到 Listen 状态

C++

```
/**
 * 将角色行为状态切换到 Listen 状态
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_do_listen(int ava_inst_id);
```

7.3.将角色行为状态切换到 Boring 状态

```C++
/**
 * 将角色行为状态切换到 Boring 状态
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @return 返回值为0时，表示调用接口正确返回。
 */
HAISDK_CALL int haisdk_avatar_do_boring(int ava_inst_id);
```

8.触发逻辑管理

8.1.虚拟人触发反馈

在没有虚拟人的界面，是可以点击按钮或设定位置去激活事件。虚拟人会被触碰产生反馈效果的调用，完全依附于代码设定触碰的优先级和设定方式。例如：在待机界面，虚拟人可以被触发反馈，但在首页界面不可以触发。

虚拟人身体部分会做出相应的行为动作，以下为触碰虚拟人的反馈函数。

```C#
/**
 * 点击数字虚拟人的身体部分，会出发虚拟人做出行为动作。
 * @param ava_inst_id haisdk_avatar_create返回的id值
 * @param posX 点击的可视区域的事件x坐标
 * @param posY 点击的可视区域的事件y坐标
 * @param usrData 所有传递给回调函数的用户数据指针
 * @param playCallback 通知动画播放结束时间的回调处理函数；如果动画设置了循环播放的标志位，结束回调就不会再调用了。
 * @return 返回非负值为当前要播放的动画实例ID，负值表示播放错误.
 */
HAISDK_CALL int haisdk_avatar_interaction_click_play_animation(int ava_inst_id,
                                                               float posX,
                                                               float posY,
                                                               HAI_PVOID usrData,
                                                               HAI_FPTR_VOID_PVOID_INT playCallback);
```

9.资源更新管理

如前所述，终端主程序可以通过调用云服务接口来获取当前应用所包含的角色，在虚拟人SDK中，角色资源更新检查和下载都是以单个角色为单位进行的。

另一方面，资源更新管理是虚拟人SDK系统中一个相对复杂的代码模块，对由于突然断网或网速过慢等状况而引起的异常情况都需要进行周全的处理，因此必须严格依照SDK定义的调用流程，才能取得预期的实现效果。

9.1.资源更新流程

一个完整的资源更新流程，应包含云端更新检查、更新文件下载、下载状态监测、下载资源生效等若干处理步骤；鉴于在资源传输过程中，可能出现网络中断等各种复杂的情况，因此资源更新流程也必须针对各种可能发生的异常情况做最保守的处理，从而保证此项功能的稳定性。SDK通过以下函数的组合调用来实现资源更新和下载的流程，资源生效的处理方法将在下一节中予以介绍。

9.1.1 检查服务器资源状态

C++

```
/**
 * SDK初始化完毕之后，可以调用这个接口，调用时需要有网络，这个接口主要就是检查服务器资源状态，检查结束会异步返回值。
 * @param avatarID  avatarID
 * @param resourceType 0表示默认资源包，1表示经常更新的资源包
 * @param userData 所有传递给回调函数的用户数据指针
 * @param checkCallback 资源检查的回调接口。
 */
HAISDK_CALL int haisdk_check_server_asset(const char *avatarID,
                                          int resourceType,
                                          HAI_PVOID userData,
                                          HAI_FPTR_CHECK_RESOURCE_CALLBACK checkCallback);
```

9.1.2 下载虚拟人资源

```C++
/**
 * 调用haisdk_check_server_asset之后，理论上可以调用这个接口了，调用时需要有网络，这个接口主要就是下载资源，
 * 但是这个接口依赖haisdk_check_server_asset的返回状态。
 * 如果haisdk_check_server_asset返回状态是HAS_UPDATE_IGNORABLE或者SHOULD_UPDATE，调用这个接口就会下载最新资源。
 * 如果haisdk_check_server_asset返回状态是其他值，调用这个接口其实没有意义。
 * @param userData 所有传递给回调函数的用户数据指针
 * @param progressCallback 资源下载的进度回调接口。
 * @param endCallback 资源下载的结束回调接口。
 */
HAISDK_CALL int haisdk_download_server_asset(HAI_PVOID userData,
                                             HAI_FPTR_DOWNLOAD_RESOURCE_PROGRESS_CALLBACK progressCallback,
                                             HAI_FPTR_DOWNLOAD_RESOURCE_END_CALLBACK endCallback);
```

【HaiSDK_Unreal】

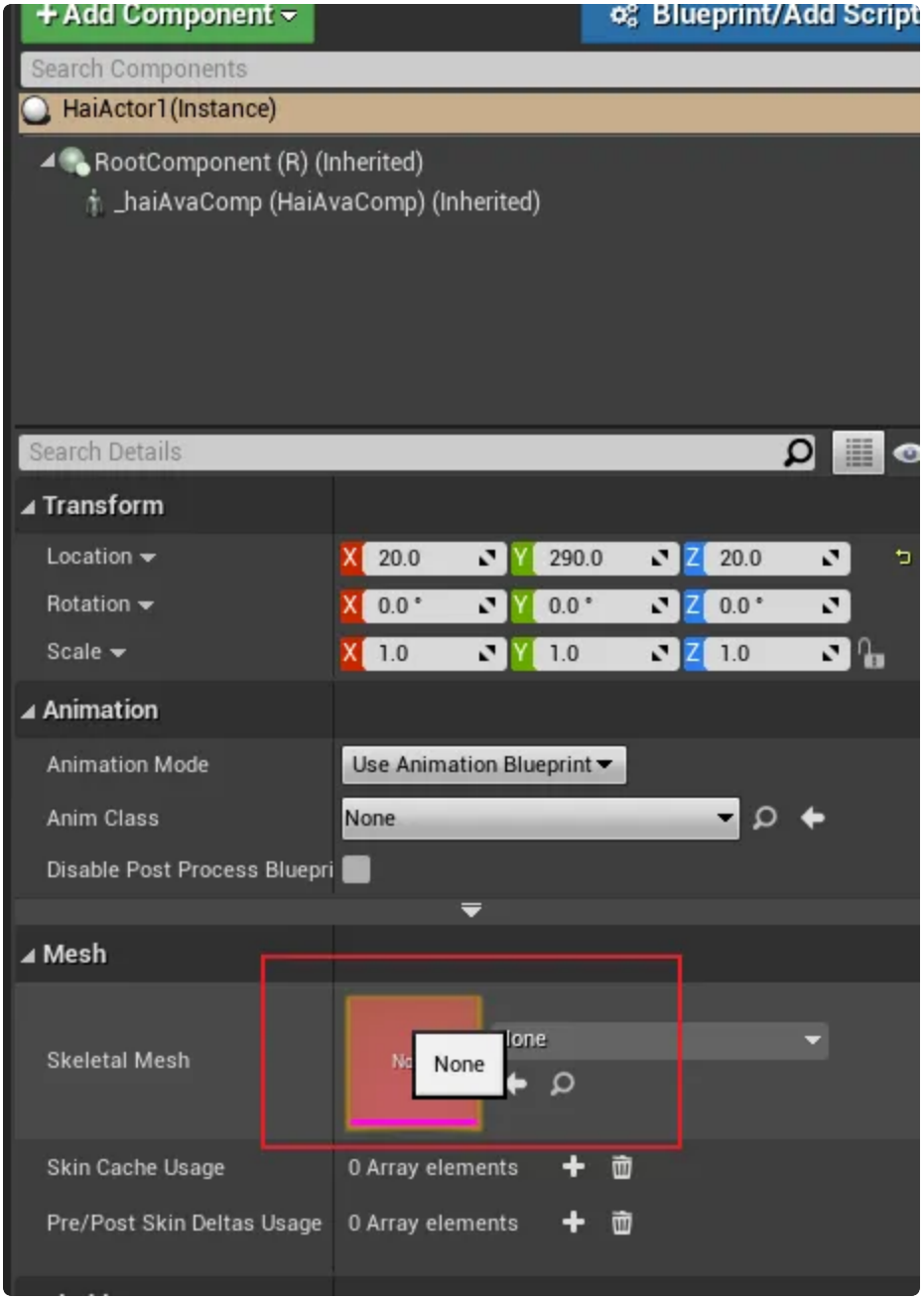
1.概览

HaiSDK_Unreal版是plugin的形式给到的，我们也提供整个测试工程。

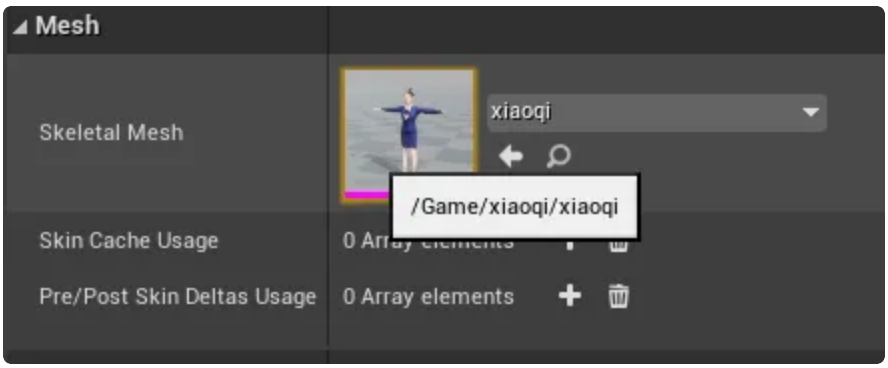
我们sdk层面，提供了一个HaiActor，这个是个可视化对象。也是我们在ue场景中添加虚拟人的关键的actor。

1.1.将haiactor拖到场景中

1.2.给这个haiactor1添加skeletalMesh



我们也提供了测试角色，例如



3.后面就可以使用c++获取这个actor使用它的接口了，或者直接使用蓝图节点来控制虚拟人了。

2.C++层接口说明

C++调用说明 C++

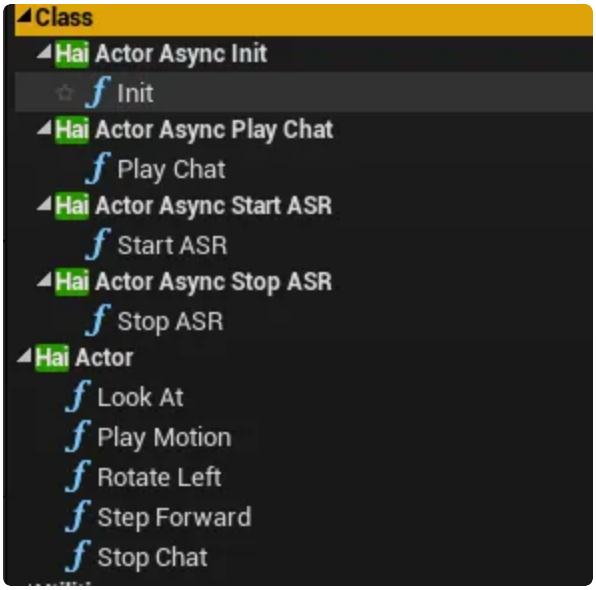
```cpp
//--------------------------------------------------------------------------------
//blueprint层可调用节点
/**
 * @brief 驱动虚拟人播放动画
 * @param motionName 动画名称
 * @param loop 是否动画循环
 */
UFUNCTION(BlueprintCallable, Category = "HaiActor")
void PlayMotion(const FString& motionName, bool loop);

/**
 * @brief 设置虚拟人向前或者向后走动多少单位距离
 * @param duration 走动的时间
 * @param distance 走动的距离
 */
UFUNCTION(BlueprintCallable, Category = "HaiActor")
void StepForward(float duration, float distance);

/**
 * @brief 设置虚拟人旋转角度
 * @param duration 旋转的时间
 * @param angle 角度值，正数为左转，负数为右转，单位是角度
 */
UFUNCTION(BlueprintCallable, Category = "HaiActor")
void RotateLeft(float duration, float angle);

/**
 * @brief 设置虚拟人身体朝向某个点
 * @param pos 朝向点位置
 */
UFUNCTION(BlueprintCallable, Category = "HaiActor")
void LookAt(FVector pos);

/**
 * @brief 停止虚拟人说话的蓝图接口
 *
 */
UFUNCTION(BlueprintCallable, Category = "HaiActor")
void StopChat();

//--------------------------------------------------------------------------------
//C++层调用接口
/**
```

```
 * @brief haisdk初始化接口
 * @param onInitCallback 只有回调返回值为true的时候，才能表示初始化成功。
 */
void AsyncInit(std::function<void(bool)> onInitCallback);

/**
 * @brief 虚拟人对话接口
 * @param useNLP 是否启用问答模式，
 *               true：将输入的inputUTF8text当做问题，经过我们智能后台的语义分析，找到合适的答案给虚拟人播报。
 *               false：将输入的inputUTF8text当做待播报的文本，让虚拟人直接播报。
 * @param useAutoGenMotion 是否启用智能动作分析
 *               true：表示智能后台会动态的组装虚拟人在播报文字的时候的行为动作。
 *               false：虚拟人播报的时候，没有行为动作，只有嘴型。
 * @param inputUTF8text 待输入给虚拟人的文本
 * @param onChatEndCallback 虚拟人播报结束的回调函数
 * @param onWalkTargetIDCallback 虚拟人播报时候，有的时候会有设置虚拟人站立位置，相机的位置，朝向。
 *                               这些参数都是通过这个函数通知返回的，
 *                               第1个返回参数是 targetID，字符类型
 *                               第2个参数是相机位置参数，FVector类型
 *                               第3个参数是相机的目标点位置，FVector类型
 * @param onVideoUriCallback 虚拟人播报时候，有的时候会返回视频信息的uri地址
 * @param onImageUriCallback 虚拟人播报时候，有的时候会返回图片信息的uri地址
 */
void AsyncPlayChat(bool useNLP,
                   bool useAutoGenMotion,
                   const std::string& inputUTF8text,
                   std::function<void()> onChatEndCallback,
                   std::function<void(const FString&,const FVector& cameraPos,const FVector& cameraTarget)> onWalkTargetIDCallback,
                   std::function<void(const FString&)> onVideoUriCallback,
                   std::function<void(const FString&)> onImageUriCallback);

/**
 * @brief 启动一句话录音转文字识别功能
 * @param typeID 默认传0，以后底层会支持多种识别算法
 * @param asrStartedCallback 录音转换功能是否开启，或者遇到异常的回调
 * @param asrEndCallback     录音转换完成之后回调，这个回调会将转换之后的文字返回
 */
void AsyncStartASR(int typeID,
                   std::function<void (bool started)> asrStartedCallback,
                   std::function<void (const FString&)> asrEndCallback);

/**
 * @brief 强行中断一句话录音转文字识别功能
 * @param asrStoppedCallback 只有这个回调返回才能确保停止了该功能
```

```
 */
void AsyncForceStopASR(std::function<void (bool stopped)> asrStoppedCallback);
```

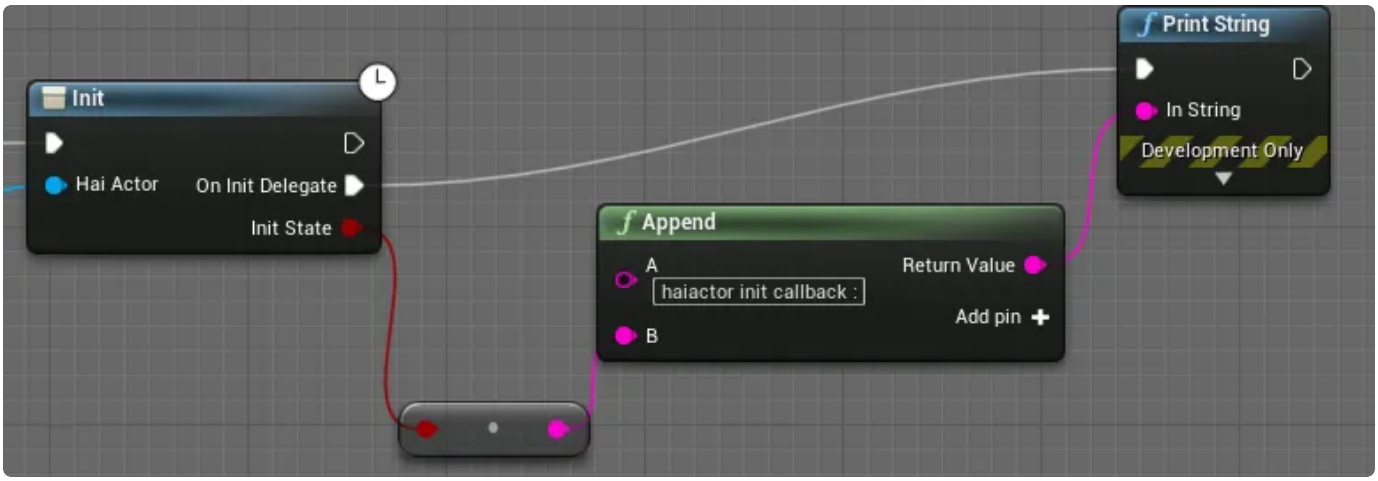
3.蓝图调用说明

我们提供的蓝图节点就是对c++的封装，具体节点列表如下



HaiActorAsyncInit函数节点

拟仁的sdk使用需要鉴权和初始化，初始化完成后有一个OnInitDelegate的回调返回，如果初始成功 Int State值为true



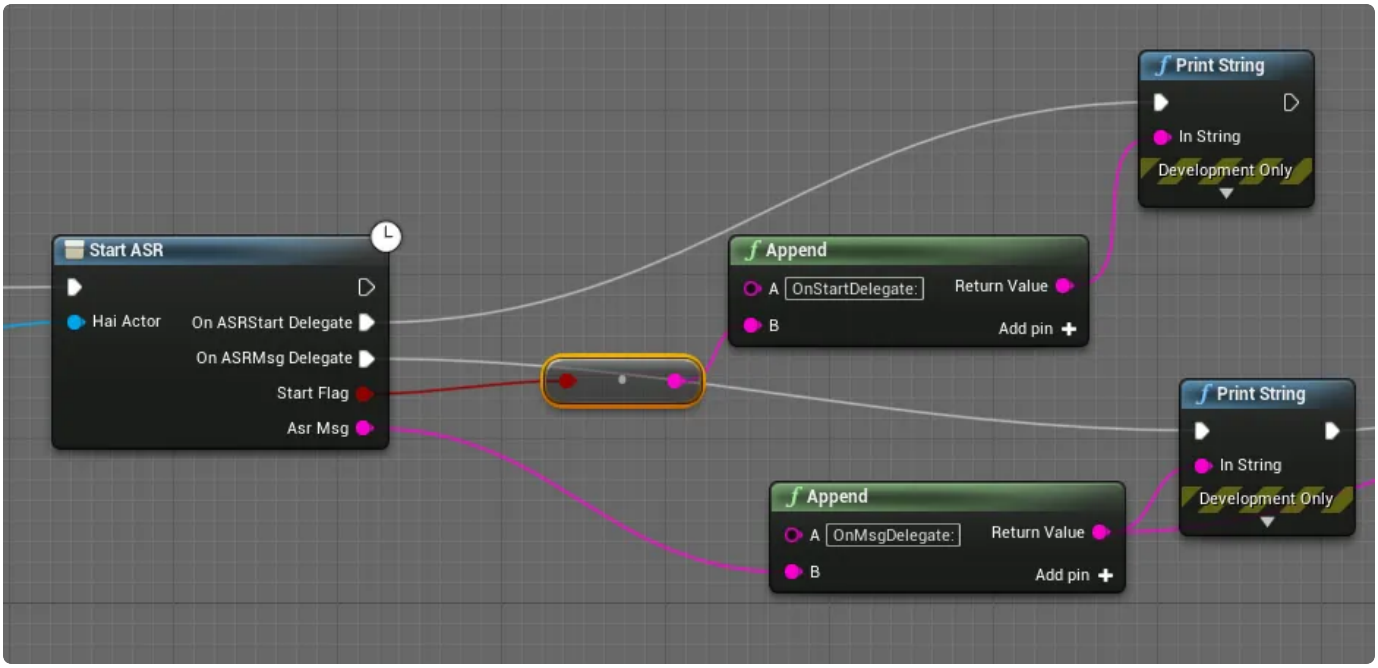
HaiActorAsyncStartASR函数节点

开启语音转文字功能

OnASRStartedDelegate:调用后会有一个bool的回调返回，true为开启成功，false为开启失败。

OnASRMsgDelegate：如果成功识别到你说的一句话后，会有一个语音识别的结果的String类型的回调返回。

· 注：Sample中是使用按键1来进行调用

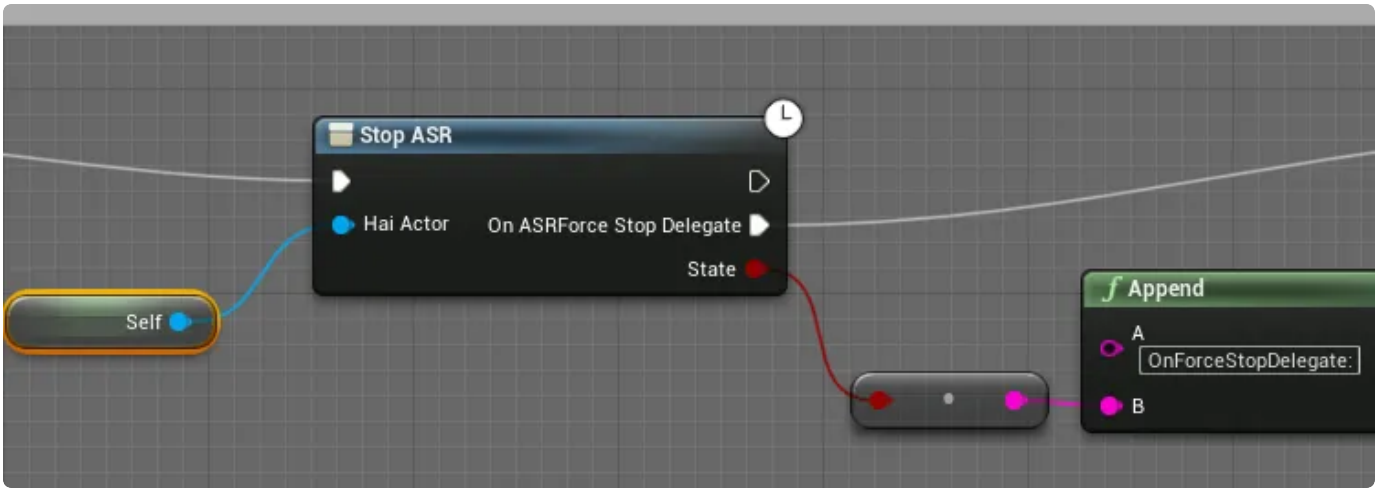


HaiActorAsyncStopASR函数节点

主动停止ASR的调用

OnASRForceStopDelegate：ASR是否停止成功的回调，State为true是停止成功，false为停止失败。

· 注：Sample中是使用按键2来进行调用



HaiActorAsyncPlayChat函数节点

USeNlp：是否使用Nlp。true为使用nlp进行回答，false为虚拟人重复inpuText的内容

InputText：和虚拟人交谈的内容

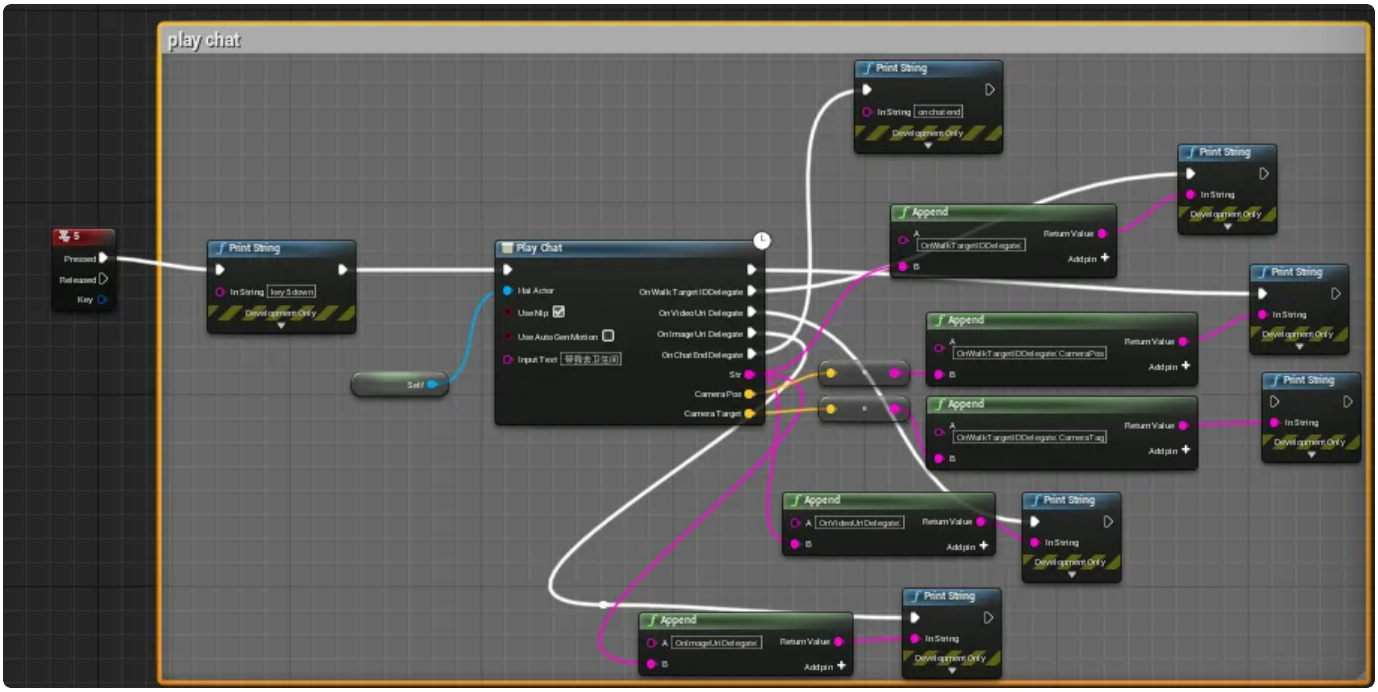
OnChatEndDelegate：虚拟人说话结束的回调

OnWalkTargetIDDelegate：虚拟人返回的Sting类型的WalkTargetId,CameraPos和CameraTarget

OnVideoUrlDelegate：虚拟人返回的String类型的视频的URL地址

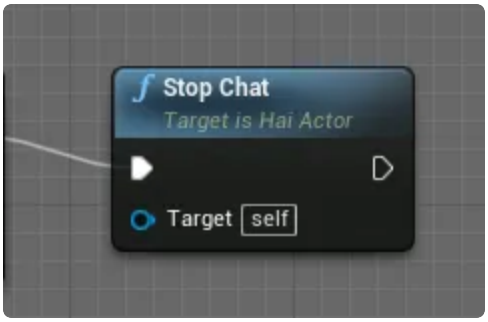
OnImageUrlDelegate：虚拟人返回的String类型的图片的URL地址

· 注：Sample中是使用按键5来进行调用



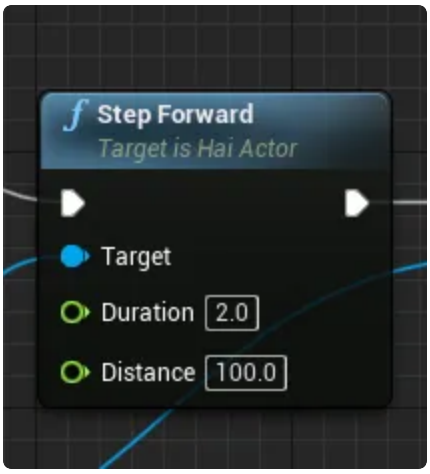
HaiActorStopChat函数节点

主动暂停虚拟人的播放或说话



HaiActorStepForward函数节点

驱动虚拟人向前步行多少距离，并可以设置多少时间到达目的点。



HaiActorRotateLeft函数节点

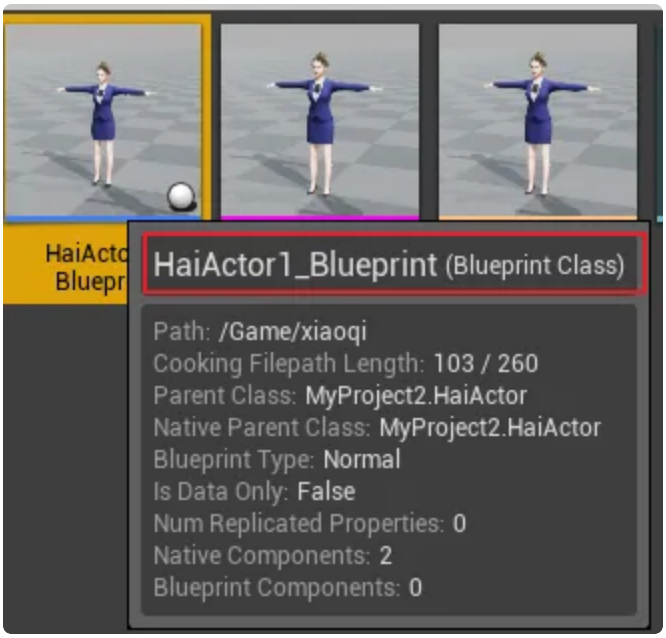
驱动虚拟人向左或者像走旋转多少角度，并可以设置多少时间旋转到位。

HaiActorLookAt函数节点

驱动虚拟人面朝哪个点位置。

4.SDK测试场景

我们也提供了一些测试sdk的蓝图，拖到场景中就可以。



里面做了一个可以按键测试sdk的逻辑。

5.使用注意事项

1. sdk使用的时候需要安装预安装包，在目录_precondition里面

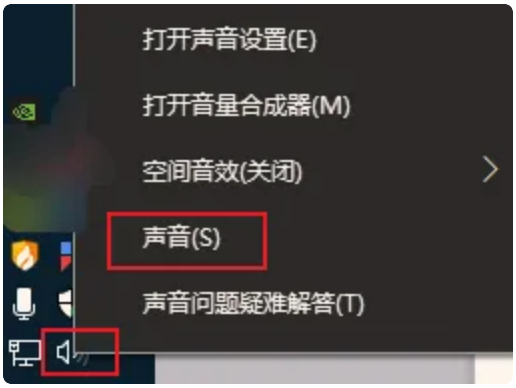
2. sdk依赖的资源在Saved\hai_conf文件夹中，conf.json可以配置sdk依赖的网络连接地址和端口。

3. Sdk是需要音频输出设备和输入设备的，默认的输出设备选择是除了“NVIDIA High Definition Audio”之外的第一个输出设备。默认输入设备是除了“Stereo Mix”之外的第一个输入设备。

当然可以手动禁用输入和输出设备，保持一个有用即可。

3.1 禁用输入设备方法如下

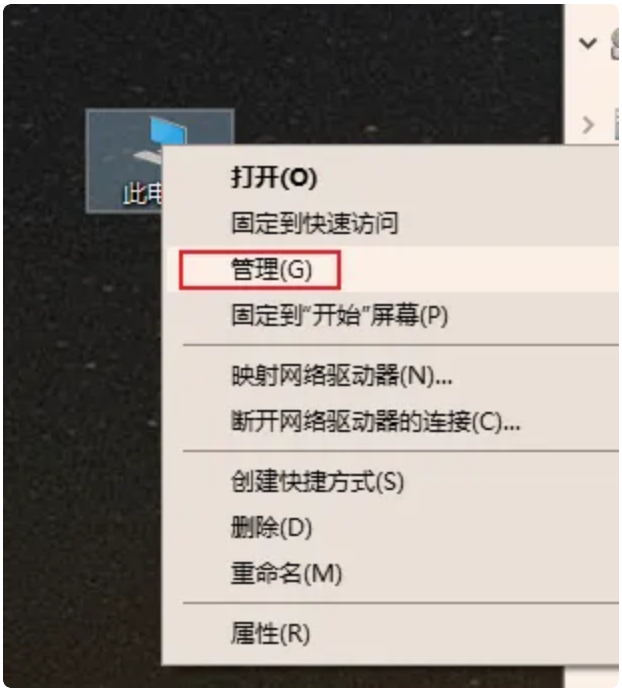
>右键点击任务栏上的小喇叭，弹出菜单，选择“声音(S)”



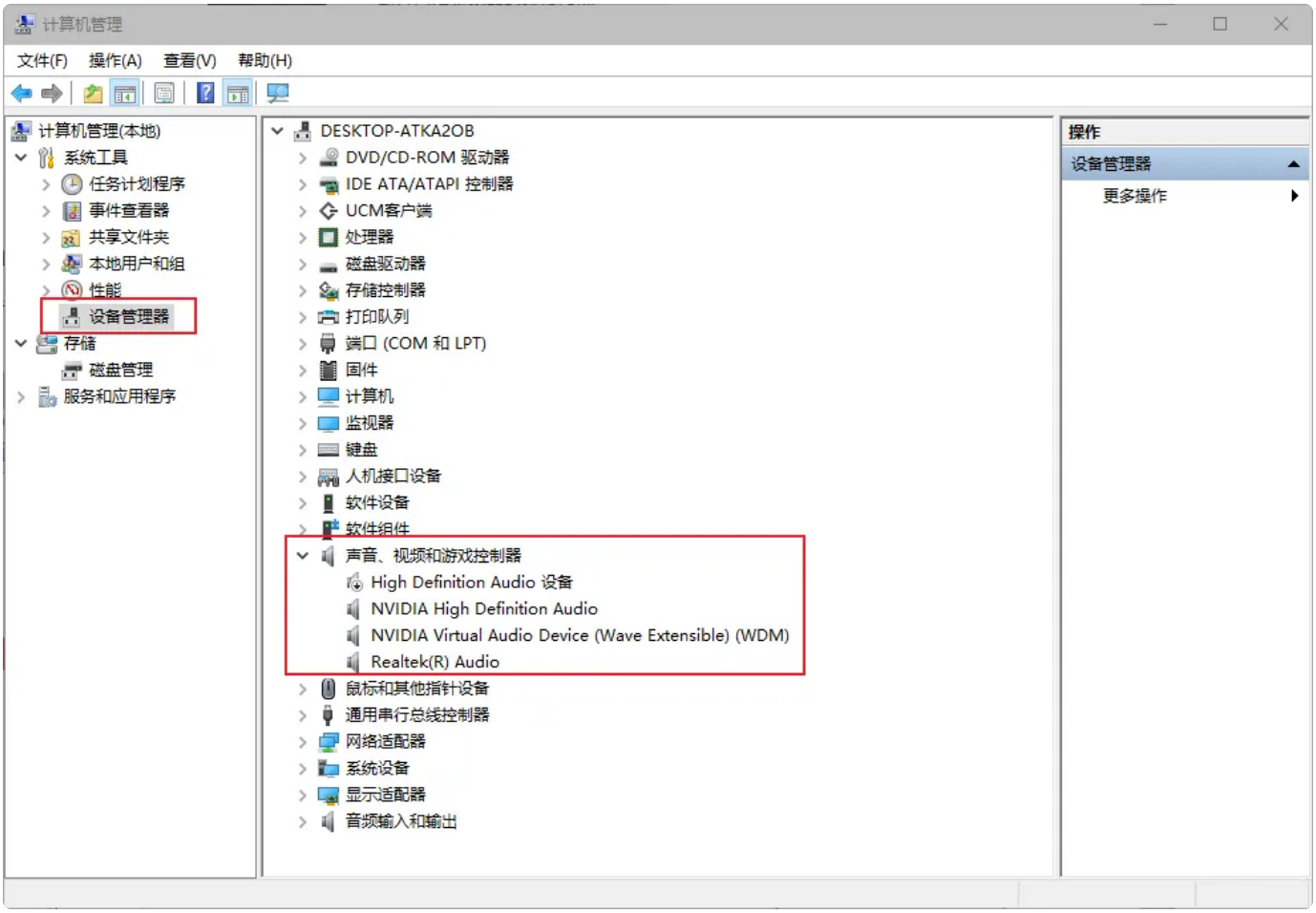
>然后选择一些无用的输入设备，选择“禁用”。

3.2禁用输出设备方法如下

- 右键选择我的电脑，选择“管理”



- 选择“设备管理器”，点击“声音，视频和游戏控制器”展开，然后理论可以只保留“Realtek（R）Audio”，别的都可以右键选择卸载。




软件组件
声音、视频和游戏控制器
High Definition Audio 设备
NVIDIA High Definition Audio
NVIDIA Virtual Audio Devi
Realtek(R) Audio
鼠标和其他指针设备
通用串行总线控制器
网络适配器
系统设备
显示适配器
音频输入和输出
更新驱动程序(P)
禁用设备(D)
卸载设备(U)
扫描检测硬件改动(A)
属性(R)