实验二十三　音视频播放

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言，音视频播放
3. 编写代码
4. 编译运行
5. 在模拟器上运行

二、实验原理

1. 鸿蒙开发原理
2. ArkTS，ArkUI 开发原理
3. 鸿蒙应用运行原理

三、实验仪器材料

1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

四、实验步骤

这个实验我们学习音视频播放的方法：

- 使用 Media Kit

  https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/media-kit-intro-V5

- 对于 Video，可以使用组件 Video

  https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-media-components-video-V5

我们主要通过实例来学习。

1. 打开 DevEco Studio，点击 Create Project 创建工程

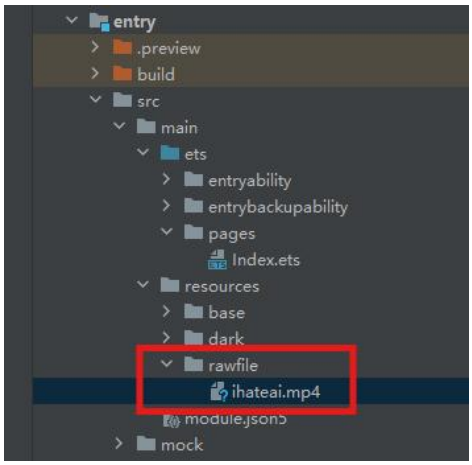
设置项目名称为 MediaDemo。

2. Video 组件

Video 组件只提供简单的视频播放功能，无法支撑复杂的视频播控场景。复杂开发场景推荐使用 AVPlayer 播控 API 和 XComponent 组件开发。

使用网络视频时，需要申请权限 ohos.permission.INTERNET。

找一个视频，放到 rawfile 目录下：



把 Index.ets 的代码改为：

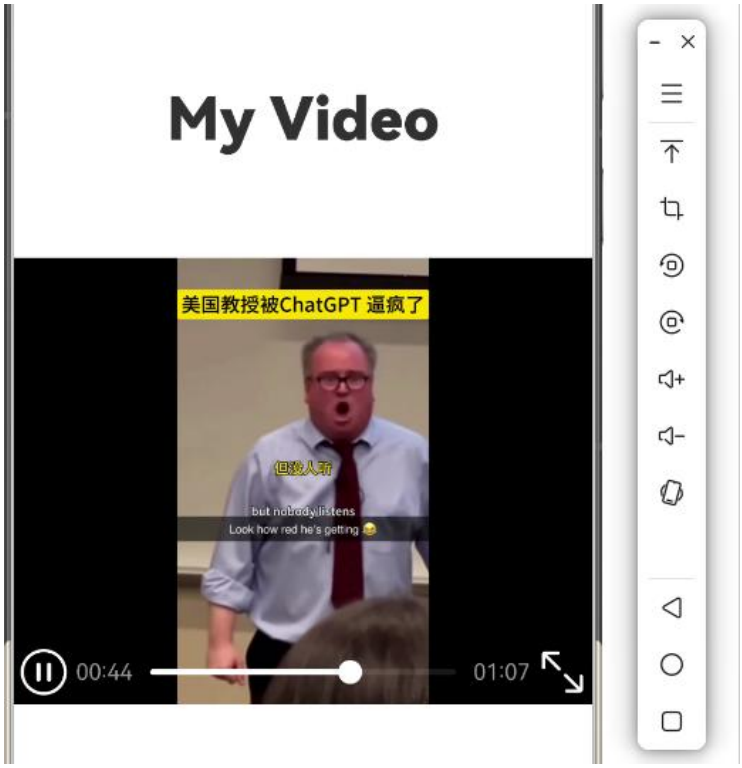
```
@Entry
@Component
struct Index {
  @State message: string = 'My Video'
  @State video: Resource = $rawfile('ihateai.mp4')
  controller: VideoController = new VideoController()
  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(40)
```

```
            .fontWeight(FontWeight.Bolder)
            .margin({bottom: 70})
        Video({
            src: this.video,
            currentProgressRate: 1,
            controller: this.controller
        }).height(300)
            .objectFit(ImageFit.Contain)
            .autoPlay(false)
            .controls(true)
            .onStart(() => {
                console.info('MediaDemo', "play video    success")
            })
            .id("videoCard")
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

可以看到，核心就是这个 Video 组件，最基本的设置是指定播放源，加上一个控制器 controller，然后是一些常见的参数设置如播放源之类。

Previewer 是不支持这个组件的：

在模拟器中运行：



可以正常播放，右下角全屏的功能也可以使用。

也可以加载网络视频，先在 module.json5 中添加权限许可：

```
    "requestPermissions": [
        {
            "name": "ohos.permission.INTERNET",
```

```
        "usedScene": {
          "abilities": [
            "EntryAbility"
          ],
          "when": "always"
        }
      }
    ],
```

然后，只需要把代码中的 video 改为 string 类型，地址输入：

```
  @State video: string =
'https:\/\/vd3.bdstatic.com\/mda-pdc2kmwtd2vxhiy4\/cae_h264\/1681502407203843413
\/mda-pdc2kmwtd2vxhiy4.mp4'
```

注意，/前面要加上\才能正确识别。

效果：

3. 使用 AVPlayer 播放音频

使用 AVPlayer 可以实现端到端播放原始媒体资源，播放的全流程包含：**创建 AVPlayer，设置播放资源，设置播放参数（音量/倍速/焦点模式），播放控制（播放/暂停/跳转/停止）**，重置，销毁资源。

官方文档：

https://developer.huawei.com/consumer/cn/doc/harmonyos-references/arkts-apis-media-avplayer

在进行应用开发的过程中，开发者可以通过 AVPlayer 的 state 属性主动获取当前状态或使用 **on('stateChange')方法监听状态变化**。如果应用在音频播放器处于错误状态时执行操作，系统可能会抛出异常或生成其他未定义的行为。

看一个简单的实例，清空 Index.ets，先加入代码：

```
@Entry
```

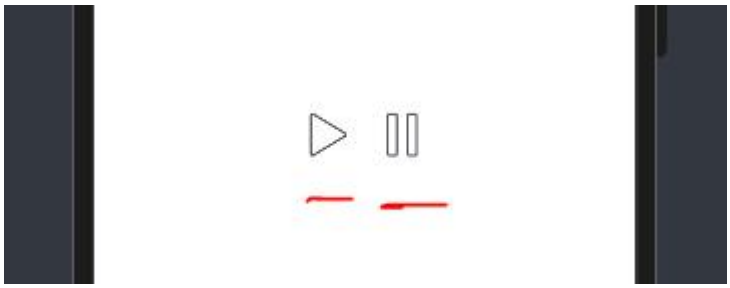
```
@Component
struct Index {
  build() {
    Row() {
      Button({type: ButtonType.Circle, stateEffect: true}) {
        Image($r("app.media.play")).width(35).height(35)
      }
      .onClick(() => {

      })
      .id('btnAudioPlay')
      .backgroundColor(0xFFFFFF)
      .margin({right: 15})

      Button({type: ButtonType.Circle, stateEffect: true}) {
        Image($r("app.media.pause")).width(35).height(35)
      }
      .onClick(() => {
      })
      .id('btnAudioStop')
      .backgroundColor(0xFFFFFF)
    }
```

```
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }
}
```

极简的一个界面，就两个控制按钮：



一个是播放，一个是暂停。

先创建 AVPlayer，在@Entry 前面添加代码：

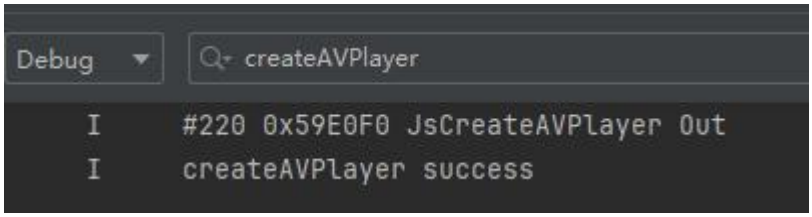
```
import { media } from '@kit.MediaKit'
import { common } from '@kit.AbilityKit';
import { BusinessError } from '@kit.BasicServicesKit'

const TAG: string = 'MediaDemo Audio#'

let player: media.AVPlayer;
media.createAVPlayer().then((audio: media.AVPlayer) => {
  if (audio != null) {
    player = audio;
    console.info('createAVPlayer success');
```

```
  } else {
    console.error('createAVPlayer fail');
  }
}).catch((error: BusinessError) => {
  console.error(`AVPlayer catchCallback, error message:${error.message}`);
});
```

在模拟器上运行，查看 Log：



可以看到，创建 AVPlayer 成功了。

下一步是设置播放资源，在 build() 前面加入代码：

```
  @State hasInit: boolean = false;
  private context?: common.UIAbilityContext;
  stateChangeCallback: Function | undefined = undefined;

  aboutToAppear() {
    this.context = getContext(this) as common.UIAbilityContext
  }

  initAudioPlayer() {
    if(this.context){
```

```
        this.setCallback(player)
        this.context.resourceManager.getRawFd("music_wukong.mp3").then(value => {
          player.fdSrc = value
        })
      }
    }

    setCallback(player: media.AVPlayer) {
      player.on('stateChange', this.stateChangeCallback = async (state: string, reason:
media.StateChangeReason) => {
        switch (state) {
          case 'idle':
            console.info('state idle called');
            break;
          case 'initialized':
            console.info(TAG, 'initialized prepared called')
            this.hasInit = true;
            player.prepare().then(() => {
              player.play()
            })
            break;
          case 'playing':
```

```
            console.info(TAG, 'audio play success')
            break;
        case 'paused':
            console.info(TAG, 'audio paused success')
            break;
        case 'error':
            console.error('state error called');
            break;
        default:
            console.info('unkown state :' + state);
            break;
      }
    })
  }
```

这里是在做准备的工作：

- setCallback 函数是通过 AVPlayer 的 state 属性主动获取当前状态或使用 on('stateChange')方法监听状态变化
- initAVPlayer 函数是绑定播放资源，我们在 rawfile 里面放了一个 music_wukong.mp3 文件

现在，给播放按钮的 onClick 事件添加代码：

```
      .onClick(() => {
        console.info(TAG, "play button clicked, hasInit: " + this.hasInit)
        console.log("this.hasInit",this.hasInit)
        if (this.hasInit) {
```

```
            player.play()
        } else {
            this.initAudioPlayer()
        }
    })
```

意思是，如果第一次点击，此时的 hasInit 是 false，就做一次初始化。否则就播放。

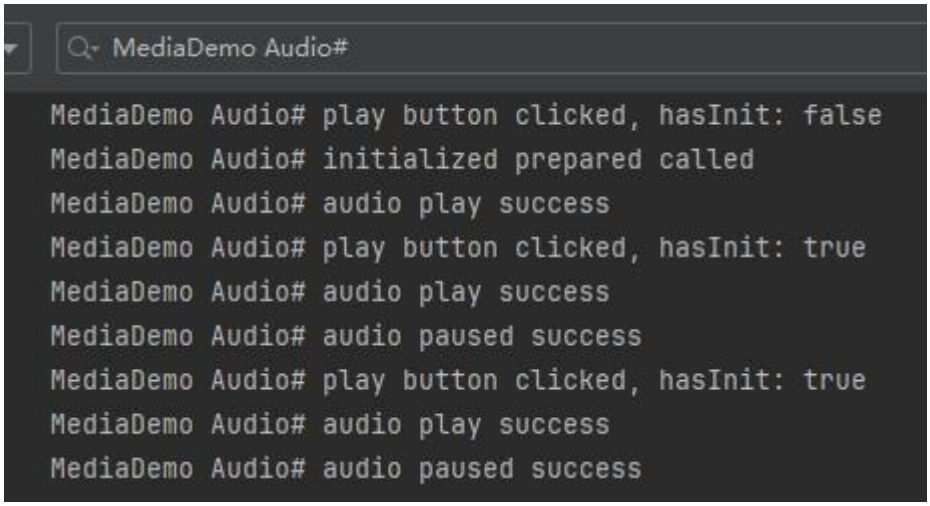
再给暂停按钮添加 onClick 事件的代码：

```
    .onClick(() => {
        player.pause()
    })
```

这个很简单，就是调用 pause()函数。

再在模拟器上运行，随机点击播放（第一次务必点播放初始化播放器），暂停按钮：

```
#220 0x59E0F0 JsCreateAVPlayer Out
createAVPlayer success
```



音频正确播放，暂停正确。

详细的指引：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/using-avplayer-for-playback-V5

4. 使用 AVPlayer 播放视频

两种视频播放开发的方案的比较：

- **AVPlayer：功能较完善**的音视频播放 ArkTS/JS API，**集成了流媒体和本地资源解析，媒体资源解封装，视频解码和渲染功能**，适用于对媒体资源进行端到端播放的场景，可直接播放 mp4、mkv 等格式的视频文件。
- **Video 组件**：封装了视频播放的**基础能力**，只需要**设置数据源**以及**基础信息**即可播放视频，但**相对扩展能力较弱**。Video 组件由 ArkUI 提供能力，相关指导请参考 UI 开发文档-Video 组件。

Video 组件我们前面已经尝试过。对于使用 AVPlayer 的方式，播放的全流程包含：**创建 AVPlayer，设置播放资源和窗口，设置播放参数（音量/倍速/缩放模式），播放控制（播放/暂停/跳转/停止），重置，销毁资源**。在进行应用开发的过程中，开发者可以通过 AVPlayer 的 state 属性主动获取当前状态或使用 **on('stateChange')方法监听状态变化**。如果应用在视频播放器处于错误状态时执行操作，系统可能会抛出异常或生成其他未定义的行为。

开发步骤：

(1) 调用 createAVPlayer()创建 AVPlayer 实例，初始化进入 idle 状态

(2) 设置业务需要的监听事件，搭配全流程场景使用

(3) 设置资源：设置属性 url，AVPlayer 进入 initialized 状态

(4) 设置窗口：获取并设置属性 SurfaceID，用于设置显示画面

(5) 应用需要从 XComponent 组件获取 surfaceID，获取方式请参考 XComponent

(6) 准备播放：调用 prepare()，AVPlayer 进入 prepared 状态，此时可以获取 duration，设置缩放模式、音量等

(7) 视频播控：播放 play()，暂停 pause()，跳转 seek()，停止 stop() 等操作

(8) （可选）更换资源：调用 reset()重置资源，AVPlayer 重新进入 idle 状态，允许更换资源 url

(9) 退出播放：调用 release()销毁实例，AVPlayer 进入 released 状态，退出播放

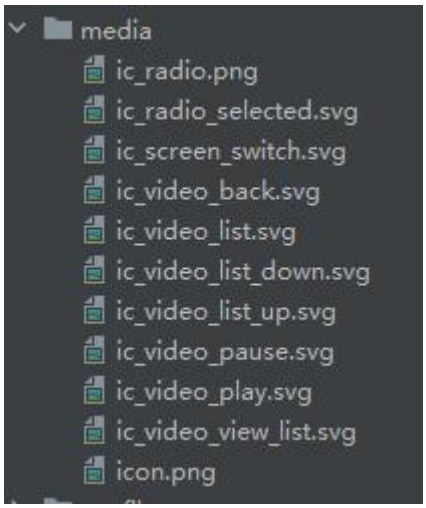
我们通过逐步实现这个例子来理解这个过程：

https://gitee.com/harmonyos_samples/video-play/

1） 首先，准备 resources 目录里面的资源：

- 在 rawfile 目录下面放置两个 mp4 文件（请要求老师发出来）

- 在 media 目录下，拷贝图片：（请要求老师发出来）



- 在 element 目录下，修改 color.json 为：

```
{
  "color": [
    {
      "name": "start_window_background",
      "value": "#FFFFFF"
    },
    {
      "name": "slider_selected",
      "value": "#007DFF"
```

```
    },
    {
        "name": "speed_dialog",
        "value": "#33bab4b4"
    },
    {
        "name": "video_play",
        "value": "#333333"
    },
    {
        "name": "video_play_selected",
        "value": "#5c5c5c"
    }
  ]
}
```

- 在 element 目录下，修改 float.json 为：

```
{
  "float": [
    {
        "name": "size_zero_five",
        "value": "0.5"
    },
```

```
    {
        "name": "size_zero_six",
        "value": "0.6"
    },
    {
        "name": "size_zero",
        "value": "0"
    },
    {
        "name": "size_1",
        "value": "1"
    },
    {
        "name": "size_5",
        "value": "5"
    },
    {
        "name": "size_10",
        "value": "10"
    },
    {
        "name": "size_12",
```

```
        "value": "12"
    },
    {
        "name": "size_15",
        "value": "15"
    },
    {
        "name": "size_16",
        "value": "16"
    },
    {
        "name": "size_18",
        "value": "18"
    },
    {
        "name": "size_20",
        "value": "20"
    },
    {
        "name": "size_down_20",
        "value": "-20"
    },
```

```
    {
        "name": "size_24",
        "value": "24"
    },
    {
        "name": "size_25",
        "value": "25"
    },
    {
        "name": "size_30",
        "value": "30"
    },
    {
        "name": "size_32",
        "value": "32"
    },
    {
        "name": "size_35",
        "value": "35"
    },
    {
        "name": "size_40",
```

```
        "value": "40"
    },
    {
        "name": "size_45",
        "value": "45"
    },
    {
        "name": "size_48",
        "value": "48"
    },
    {
        "name": "size_50",
        "value": "50"
    },
    {
        "name": "size_64",
        "value": "64"
    },
    {
        "name": "size_75",
        "value": "75"
    },
```

```
    {
        "name": "size_80",
        "value": "-80"
    },
    {
        "name": "size_254",
        "value": "254"
    }
  ]
}
```

- 在 element 目录下，修改 string.json 为：

```
{
  "string": [
    {
        "name": "module_desc",
        "value": "module description"
    },
    {
        "name": "entryAbility_desc",
        "value": "description"
    },
    {
```

```
        "name": "entryAbility_label",
        "value": "video play"
    },
    {
        "name": "video_res_1",
        "value": "test1.mp4"
    },
    {
        "name": "video_res_2",
        "value": "test2.mp4"
    },
    {
        "name": "video_res_3",
        "value": "network.mp4"
    },
    {
        "name": "video_speed_1_0X",
        "value": "1.0X"
    },
    {
        "name": "video_speed_1_25X",
        "value": "1.25X"
```

```
    },
    {
        "name": "video_speed_1_75X",
        "value": "1.75X"
    },
    {
        "name": "video_speed_2_0X",
        "value": "2.0X"
    },
    {
        "name": "video_warn",
        "value": "Please check if the network is connected or available!"
    },
    {
        "name": "dialog_cancel",
        "value": "Cancel"
    },
    {
        "name": "dialog_play_speed",
        "value": "Playback speed"
    },
    {
```

```
        "name": "playing",
        "value": "Playing"
    },
    {
        "name": "reason_internet",
        "value": "Used for accessing the Internet during video playback"
    },
    {
        "name": "reason_get_network_info",
        "value": "Used for retrieving network information during video playback"
    }

  ]
}
```

- 将 en_US > element 下面的 string.json 改为：

```
{
  "string": [
    {
        "name": "module_desc",
        "value": "module description"
    },
    {
```

```
        "name": "entryAbility_desc",
        "value": "description"
    },
    {
        "name": "entryAbility_label",
        "value": "video play"
    },
    {
        "name": "video_res_1",
        "value": "test1.mp4"
    },
    {
        "name": "video_res_2",
        "value": "test2.mp4"
    },
    {
        "name": "video_res_3",
        "value": "network.mp4"
    },
    {
        "name": "video_speed_1_0X",
        "value": "1.0X"
```

```
    },
    {
        "name": "video_speed_1_25X",
        "value": "1.25X"
    },
    {
        "name": "video_speed_1_75X",
        "value": "1.75X"
    },
    {
        "name": "video_speed_2_0X",
        "value": "2.0X"
    },
    {
        "name": "video_warn",
        "value": "Please check if the network is connected or available!"
    },
    {
        "name": "dialog_cancel",
        "value": "Cancel"
    },
    {
```

```
        "name": "dialog_play_speed",
        "value": "Playback speed"
    },
    {
        "name": "playing",
        "value": "playing"
    },
    {
        "name": "reason_internet",
        "value": "Used for accessing the Internet during video playback"
    },
    {
        "name": "reason_get_network_info",
        "value": "Used for retrieving network information during video playback"
    }


  ]
}
```

- 将 zh_CN > element 下面的 string.json 改为：

```
{
  "string": [
    {
```

```
        "name": "module_desc",
        "value": "模块描述"
    },
    {
        "name": "entryAbility_desc",
        "value": "description"
    },
    {
        "name": "entryAbility_label",
        "value": "视频播放"
    },
    {
        "name": "video_warn",
        "value": "请检查网络是否连接或可用！"
    },
    {
        "name": "video_speed_1_0X",
        "value": "1.0X"
    },
    {
        "name": "video_speed_1_25X",
        "value": "1.25X"
```

```
    },
    {
        "name": "video_speed_1_75X",
        "value": "1.75X"
    },
    {
        "name": "video_res_1",
        "value": "test1.mp4"
    },
    {
        "name": "video_res_2",
        "value": "test2.mp4"
    },
    {
        "name": "video_res_3",
        "value": "network.mp4"
    },
    {
        "name": "video_speed_2_0X",
        "value": "2.0X"
    },
    {
```

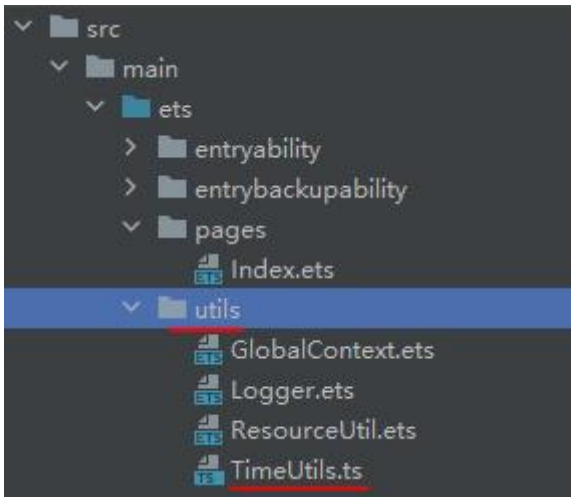
```
        "name": "dialog_cancel",
        "value": "取消"
    },
    {
        "name": "dialog_play_speed",
        "value": "播放倍速"
    },
    {
        "name": "playing",
        "value": "当前播放"
    },
    {
        "name": "reason_internet",
        "value": "用于视频播放场景使用 Internet 网络"
    },
    {
        "name": "reason_get_network_info",
        "value": "用于视频播放场景获取网络信息"
    }
  ]
}
```

2) 在 module.json5 中添加权限：

```
  "requestPermissions": [
    {
      "name": "ohos.permission.INTERNET",
      "reason": "$string:reason_internet",
      "usedScene": {
        "abilities": [
          "EntryAbility"
        ],
        "when": "always"
      }
    },
    {
      "name": "ohos.permission.GET_NETWORK_INFO",
      "reason": "$string:reason_get_network_info",
      "usedScene": {
        "abilities": [
          "EntryAbility"
        ],
        "when": "always"
      }
    }
  ]
```

上面这些内容，相信大家学到现在应该都能理解。

3) 在 src > main > ets 下创建目录 utils，创建几个文件：



注意，TimeUtils 的后缀是 ts，而不是 ets。

- Logger.ets 代码：

```
import { hilog } from '@kit.PerformanceAnalysisKit';


class Logger {
  private domain: number;
  private prefix: string;
  private format: string = '%{public}s, %{public}s';


  constructor(prefix: string) {
    this.prefix = prefix;
    this.domain = 0xFF00;
  }
```

```
  debug(...args: string[]): void {
    hilog.debug(this.domain, this.prefix, this.format, args);
  }

  info(...args: string[]): void {
    hilog.info(this.domain, this.prefix, this.format, args);
  }

  warn(...args: string[]): void {
    hilog.warn(this.domain, this.prefix, this.format, args);
  }

  error(...args: string[]): void {
    hilog.error(this.domain, this.prefix, this.format, args);
  }
}

export default new Logger('[VideoPlayDemo]');
```

这里做的事情是封装一下 hilog 的函数，创建并导出一个 Logger 实例。

- TimeUtils.ets 代码：

```
const TIME_ONE = 60000;
```

```
const TIME_TWO = 1000;

const TIME_THREE = 10;


export function timeConvert(time: number): string {
  let min: number = Math.floor(time / TIME_ONE);
  let second: string = ((time % TIME_ONE) / TIME_TWO).toFixed(0);
  return `${min}:${(+second < TIME_THREE ? '0' : '') + second}`;
}
```

这个代码是用在视频的时间显示的时候，把分钟和秒中间加一个冒号。

- ResourceUtil.ets 代码：

```
import { abilityDelegatorRegistry } from '@kit.TestKit'


let context: Context


export async function getString(str: Resource) {
  if (context == null) {
    const abilityDelegator = abilityDelegatorRegistry.getAbilityDelegator()
    context = await abilityDelegator.getAppContext()
  }
  let manager = context.resourceManager
  return await manager.getStringValue(str);
}
```

这个代码的核心其实是最后一句，通过这个可以将 Resource 类型转换为 string 类型。

- GlobalContext.ets 代码：

```
export class GlobalContext {
  private constructor() {
  }

  private static instance: GlobalContext;
  private _objects = new Map<string, Object>();

  public static getContext(): GlobalContext {
    if (!GlobalContext.instance) {
      GlobalContext.instance = new GlobalContext();
    }
    return GlobalContext.instance;
  }

  getObject(value: string): Object | undefined {
    return this._objects.get(value);
  }

  setObject(key: string, objectClass: Object): void {
    this._objects.set(key, objectClass);
```

```
  }
}
```

这里定义了一个 GlobalContext 类，自带一个 Map 数据类型，可以定义 key-value 对，如果定义一个 key 为 context，把当前的 context 值赋给它，然后就可以在 context 的基础上拓展自己的键值对。

4) 将 entryability 目录下的 EntryAbility.ets 代码修改为（红色字体为新增的代码）：

```
import { AbilityConstant, UIAbility, Want } from '@kit.AbilityKit';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { window } from '@kit.ArkUI';
import { emitter } from '@kit.BasicServicesKit';
import { GlobalContext } from '../utils/GlobalContext';

export default class EntryAbility extends UIAbility {
  private tag = 'VideoPlayDemo';

  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
    GlobalContext.getContext().setObject('abilityWant', want)
    GlobalContext.getContext().setObject('context', this.context)
    if (want.parameters) {
      if (want.parameters.currentTime) {
        GlobalContext.getContext().setObject('currentTime',
want.parameters.currentTime);
```

```
            console.info(this.tag, 'time: ' + want.parameters.currentTime);
        }
    }
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onCreate');
  }

  onDestroy(): void {
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onDestroy');
  }

  onWindowStageCreate(windowStage: window.WindowStage): void {
    // Main window is created, set main page for this ability
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onWindowStageCreate');
    windowStage.getMainWindow().then((win: window.Window) => {
      win.setWindowKeepScreenOn(true);
      win.setWindowSystemBarProperties({
        statusBarColor: '#000000',
        statusBarContentColor: '#FFFFFF'
      });
      win.setWindowLayoutFullScreen(true);
      win.on('windowSizeChange', (newSize: window.Size) => {
        let eventWHData: emitter.EventData = {
```

```
            data: {
              'width': newSize.width,
              'height': newSize.height
            }
          };
          let innerEventWH: emitter.InnerEvent = {
            eventId: 3,
            priority: emitter.EventPriority.HIGH
          };
          emitter.emit(innerEventWH, eventWHData);
        });
      });
      windowStage.loadContent('pages/Index', (err) => {
        if (err.code) {
          hilog.error(0x0000, 'testTag', 'Failed to load the content. Cause: %{public}s',
JSON.stringify(err) ?? '');
          return;
        }
        hilog.info(0x0000, 'testTag', 'Succeeded in loading the content.');
      });
    }
```

```
  onWindowStageDestroy(): void {
    // Main window is destroyed, release UI related resources
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onWindowStageDestroy');
  }

  onForeground(): void {
    // Ability has brought to foreground
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onForeground');
  }

  onBackground(): void {
    // Ability has back to background
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onBackground');
  }
}
```

几个重点：

- 导入了 emitter，这个是线程之间通信的方法， Emitter 主要提供线程间发送和处理事件的能力，包括对持续订阅事件或单次订阅事件的处理、取消订阅事件、发送事件到事件队列等。具体参考这里：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/itc-with-emitter-V5

发送调用 emit，比如这里：

```
emitter.emit(innerEventWH, eventWHData);
```

就是发送了事件“innerEventWH”，携带了数据“eventWHData”，就是窗口的宽度和高度发生了变化。

- GlobalContext 的使用

这句话很关键，通过这个把 context 里面的东西都用上了：

```
GlobalContext.getContext().setObject('context', this.context)
```

然后我们还可以增加一些键值对，比如：

```
GlobalContext.getContext().setObject('currentTime',
want.parameters.currentTime);
```

- 对窗口的一些配置

- setWindowKeepScreenOn：保持屏幕一直是 on 的状态

- setWindowSystemBarProperties：设置状态条的属性

- setWindowLayoutFullScreen：设置全屏

- win.on('windowSizeChange',：响应窗口变化事件

```
windowStage.getMainWindow().then((win: window.Window) => {
  win.setWindowKeepScreenOn(true);
  win.setWindowSystemBarProperties({
    statusBarColor: '#000000',
    statusBarContentColor: '#FFFFFF'
  });
  win.setWindowLayoutFullScreen(true);
  win.on('windowSizeChange', (newSize: window.Size) => {
    let eventWHData: emitter.EventData = {
      data: {
        'width': newSize.width,
        'height': newSize.height
      }
    };
    let innerEventWH: emitter.InnerEvent = {
      eventId: 3,
      priority: emitter.EventPriority.HIGH
    };
    emitter.emit(innerEventWH, eventWHData);
```

5) 在 src > main > ets 下创建目录 videomanager，创建文件 AvPlayManager.ets，添加代码：

```
import { media } from '@kit.MediaKit';
import { resourceManager } from '@kit.LocalizationKit';
import { emitter } from '@kit.BasicServicesKit';
```

```
import Logger from '../utils/Logger';
import { GlobalContext } from '../utils/GlobalContext';
import { common } from '@kit.AbilityKit';

export default class AvPlayManage {
  private tag: string = 'AVPlayManage';
  private flag: boolean = false;
  private avPlayer: media.AVPlayer | null = null;
  private surfaceID: string = '';
  private mgr: resourceManager.ResourceManager = {} as resourceManager.ResourceManager;
  private currentTime: number = 0; // Current time of the video.
  private durationTime: number = 0; // Total video length
  private speedSelect: number = 0; // Playback rate selection
  private fileDescriptor: resourceManager.RawFileDescriptor | null = null;
  private videoSrc: string = '';
  private fileSrc: string = '';

  async initPlayer(surfaceId: string, callback: (avPlayer: media.AVPlayer) => void): Promise<void> {
    Logger.info(this.tag, 'initPlayer==initCamera surfaceId== ${surfaceId}');
    this.surfaceID = surfaceId;
```

```
    Logger.info(this.tag, 'initPlayer==this.surfaceID surfaceId== ${this.surfaceID}');
    try {
      Logger.info(this.tag, 'initPlayer videoPlay avPlayerDemo');
      // Creates the avPlayer instance object.
      this.avPlayer = await media.createAVPlayer();
      // Creates a callback function for state machine changes.
      await this.setAVPlayerCallback(callback);
      Logger.info(this.tag, 'initPlayer videoPlay setAVPlayerCallback');
      this.mgr = (GlobalContext.getContext().getObject('context') as
(common.UIAbilityContext)).resourceManager;
      Logger.info(this.tag, 'initPlayer videoPlay this.mgr');
      this.fileDescriptor = await this.mgr.getRawFd('test1.mp4');
      Logger.info(this.tag, 'initPlayer videoPlay fileDescriptor =
${JSON.stringify(this.fileDescriptor)}');
      this.avPlayer.fdSrc = this.fileDescriptor;
    } catch (e) {
      Logger.error(this.tag, 'initPlayer initPlayer err: ${e}');
    }
  }

  // Registering the avplayer callback function
```

```
  async setAVPlayerCallback(callback: (avPlayer: media.AVPlayer) => void, videoSrc?:
string): Promise<void> {
    if (this.avPlayer === null) {
      return;
    }
    // Callback function of the seek operation result
    this.avPlayer.on('seekDone', (seekDoneTime) => {
      Logger.info(this.tag, 'setAVPlayerCallback AVPlayer seek succeeded, seek time is
${seekDoneTime}');
    });
    // error callback listening function. When an error occurs during the operation of the
avPlayer,
    // the reset interface is invoked to trigger the reset process.
    this.avPlayer.on('error', (err) => {
      if (this.avPlayer === null) {
        return;
      }
      Logger.error(this.tag, 'setAVPlayerCallback Invoke avPlayer failed, code is ${err.code},
message is ${err.message}');
      this.avPlayer.reset();
    });
    // Callback function for state machine changes
```

```
    this.avPlayer.on('stateChange', async (state, reason) => {
      if (this.avPlayer === null) {
        return;
      }
      switch (state) {
        case 'idle': // This state machine is triggered after the reset interface is successfully invoked.
          this.avPlayer.release();
          Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state idle called.');
          break;
        case 'initialized': // This status is reported after the playback source is set on the AVPlayer.
          Logger.info(this.tag, 'setAVPlayerCallback AVPlayerstate initialized called.');
          this.avPlayer.surfaceId = this.surfaceID; // Set the display screen. This parameter is not required when the resource to be played is audio-only.
          Logger.info(this.tag, 'setAVPlayerCallback this.avPlayer.surfaceId = ${this.avPlayer.surfaceId}');
          this.avPlayer.prepare();
          break;
        case 'prepared': // This state machine is reported after the prepare interface is successfully invoked.
          Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state prepared called.');
```

```
            this.durationTime = this.avPlayer.duration;
            this.currentTime = this.avPlayer.currentTime;
            this.avPlayer.play(); // Invoke the playback interface to start playback.
            Logger.info(this.tag, 'setAVPlayerCallback this.speedSelect =
${this.speedSelect}');
            callback(this.avPlayer);
            break;
        case 'playing': // After the play interface is successfully invoked, the state machine
is reported.
            Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state playing called.');
            let eventDataTrue: emitter.EventData = {
              data: {
                'flag': true
              }
            };
            let innerEventTrue: emitter.InnerEvent = {
              eventId: 2,
              priority: emitter.EventPriority.HIGH
            };
            emitter.emit(innerEventTrue, eventDataTrue);
            break;
```

```
            case 'completed': // This state machine is triggered to report when the playback
ends.
                Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state completed called.');
                let eventDataFalse: emitter.EventData = {
                  data: {
                    'flag': false
                  }
                };
                let innerEvent: emitter.InnerEvent = {
                  eventId: 1,
                  priority: emitter.EventPriority.HIGH
                };
                emitter.emit(innerEvent, eventDataFalse);
                break;
            default:
                Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state unknown called.');
                break;
        }
    });
    // Listening function for reporting time
    this.avPlayer.on('timeUpdate', (time: number) => {
      this.currentTime = time;
```

```
        Logger.info(this.tag, 'setAVPlayerCallback timeUpdate success,and new time is =
${this.currentTime}');
    });
  }

  getDurationTime(): number {
    return this.durationTime;
  }

  getCurrentTime(): number {
    return this.currentTime;
  }

  videoPlay(): void {
    if (this.avPlayer) {
      try {
        this.avPlayer.play();
      } catch (e) {
        Logger.error(this.tag, 'videoPlay = ${JSON.stringify(e)}');
      }
    }
  }
```

```
  videoPause(): void {
    if (this.avPlayer) {
      try {
        this.avPlayer.pause();
        Logger.info(this.tag, 'videoPause==');
      } catch (e) {
        Logger.info(this.tag, 'videoPause== ${JSON.stringify(e)}');
      }
    }
  }
  async videoRelease(): Promise<void> {
    if (this.avPlayer === null) {
      return;
    }
    this.avPlayer.release((err) => {
      if (err == null) {
        Logger.info(this.tag, 'videoRelease release success');
      } else {
        Logger.error(this.tag, 'videoRelease release filed,error message is =
${err.message}');
      }
```

```
    });
  }
}
```

回顾前面提到的开发步骤：

(1)调用 createAVPlayer()创建 AVPlayer 实例，初始化进入 idle 状态

(2)设置业务需要的监听事件，搭配全流程场景使用

(3)设置资源：设置属性 url，AVPlayer 进入 initialized 状态

查看 initPlayer 函数和 setAVPlayerCallback 函数，可以找到对应的点：

```
this.avPlayer = await media.createAVPlayer();
```

这一步是创建实例，此时状态是进入了 idle 状态。

```
await this.setAVPlayerCallback(callback);
```

这里设置 Callback，其实就是设置各种状态以及对应的监听事件：

```
// Callback function for state machine changes
this.avPlayer.on('stateChange', async (state, reason) => {
  if (this.avPlayer === null) {
    return;
  }
  switch (state) {
    case 'idle':...
    case 'initialized':...
    case 'prepared':...
    case 'playing':...
    case 'completed':...
    default:
      Logger.info(this.tag, 'setAVPlayerCallback AVPlayer state unkr
      break;
  }
});
```

回到 initPlayer:

```
this.mgr = (GlobalContext.getContext().getObject('context') as
(common.UIAbilityContext)).resourceManager;
```

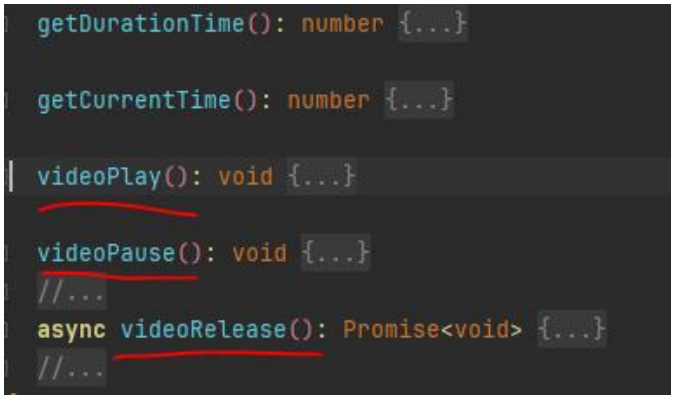
```
        Logger.info(this.tag, 'initPlayer videoPlay this.mgr');
        this.fileDescriptor = await this.mgr.getRawFd('test1.mp4');
        Logger.info(this.tag, 'initPlayer videoPlay fileDescriptor = ${JSON.stringify(this.fileDescriptor)}');
        this.avPlayer.fdSrc = this.fileDescriptor;
```

这里就是设置资源，由于我们现在的场景是本地文件，所以用的是 fdSrc， AVPlayer 进入 initialized 状态。对应的 Callback 里面，此时需要处理：

```
case 'initialized': // This status is reported af
  Logger.info(this.tag, 'setAVPlayerCallback AVPl
  this.avPlayer.surfaceId = this.surfaceID; // Se
  Logger.info(this.tag, 'setAVPlayerCallback this
  this.avPlayer.prepare();
  break;
case 'prepared': // This state machine is reporte
  Logger.info(this.tag, 'setAVPlayerCallback AVPl
  this.durationTime = this.avPlayer.duration;
  this.currentTime = this.avPlayer.currentTime;
  this.avPlayer.play(); // Invoke the playback in
```

需要用到 surfaceID，然后就进入 prepared 状态，就可以进行播放了，看下面的 play 函数。

在这个 AvPlayManager.ets 里面，我们还定义了一些函数：



```
getDurationTime(): number {...}

getCurrentTime(): number {...}

videoPlay(): void {...}

videoPause(): void {...}
//...
async videoRelease(): Promise<void> {...}
//...
```

这些都比较好理解。

6) 回到 Index.ets，代码改为：

```
import avPlayManage from '../videomanager/AvPlayManager';
import { media } from '@kit.MediaKit';
```

```
import { display } from '@kit.ArkUI';
import { emitter } from '@kit.BasicServicesKit';
import { timeConvert } from '../utils/TimeUtils';
import { GlobalContext } from '../utils/GlobalContext';

const PROPORTION = 0.99; // Screen Percentage
const SURFACEW = 0.9; // Surface width ratio
const SURFACEH = 1.78; // Surface height ratio
const TIMEOUT = 0; // Timer ID
const SET_TIME_OUT = 8000; // Interval: 8s
const SET_INTERVAL = 100;

class innerInfo {
  eventId: number = 0
  priority: emitter.EventPriority = 0
}

let innerEventFalse: innerInfo = {
  eventId: 1,
  priority: emitter.EventPriority.HIGH
};
```

```
let innerEventTrue: innerInfo = {
  eventId: 2,
  priority: emitter.EventPriority.HIGH
};

let innerEventWH: innerInfo = {
  eventId: 3,
  priority: emitter.EventPriority.HIGH
};

@Entry
@Component
struct Index {
  private surfaceId: string = '';
  private timeout: number = 0; // Timer ID
  private xComponentController: XComponentController = new XComponentController();
  @State avPlayManage: avPlayManage = new avPlayManage();
  @State isSwiping: boolean = false;
  @State isClickScreen: boolean = false;
  @State flag: boolean = true; // Pause Playback
  @State XComponentFlag: boolean = false;
  @State speedSelect: number = 0;
```

```
  @State videoListSelect: number = 0;
  @State durationTime: number = 0;
  @State currentTime: number = 0;
  @State surfaceW: number = 0;
  @State surfaceH: number = 0;
  @State show: boolean = false; // Indicates whether the videoPanel component is displayed.
  @State videoSelect: number = 0;
  @State percent: number = 0;
  @State windowWidth: number = 300;
  @State windowHeight: number = 300;
  @State isCalcWHFinished: boolean = false;
  @StorageLink('videoName') videoName: Resource = $r('app.string.video_res_1');
  @StorageLink('videoIndex') videoIndex: number = 0;

  setTimer(): void {
    let that = this;
    this.timeout = setTimeout(() => {
      that.isClickScreen = false; // Hide the operation panel
    }, SET_TIME_OUT); // Hide in 8 seconds
  }
```

```
  clearTimer(): void {
    if (this.timeout !== TIMEOUT) {
      clearTimeout(this.timeout);
      this.timeout = TIMEOUT;
    }
  }

  aboutToAppear() {
    this.windowWidth = display.getDefaultDisplaySync().width;
    this.windowHeight = display.getDefaultDisplaySync().height;
    this.surfaceW = (GlobalContext.getContext().getObject('windowWidth') as number) *
SURFACEW;
    this.surfaceH = this.surfaceW / SURFACEH;
    this.flag = true;
    AppStorage.setOrCreate('avPlayManage', this.avPlayManage);
  }

  aboutToDisappear() {
    this.avPlayManage.videoRelease();
    emitter.off(innerEventFalse.eventId);
  }
```

```
  onPageHide() {
    this.avPlayManage.videoPause();
    this.flag = false;
  }

  onPageShow() {
    emitter.on(innerEventTrue, (res) => {
      if (res.data) {
        this.flag = res.data.flag;
        this.XComponentFlag = res.data.flag;
      }
    });
    emitter.on(innerEventFalse, (res) => {
      if (res.data) {
        this.flag = res.data.flag;
      }
    });
    emitter.on(innerEventWH, (res) => {
      if (res.data) {
        this.windowWidth = res.data.width;
        this.windowHeight = res.data.height;
        this.setVideoWH();
```

```
        }
    });

    if (this.flag == false) {
        this.clearTimer();
    }
  }

  setVideoWH(): void {
    if (this.percent >= 1) { // Horizontal video
        this.surfaceW = Math.round(this.windowWidth * PROPORTION);
        this.surfaceH = Math.round(this.surfaceW / this.percent);
    } else { // Vertical video
        this.surfaceH = Math.round(this.windowHeight * PROPORTION);
        this.surfaceW = Math.round(this.surfaceH * this.percent);
    }
  }

  @Builder
  CoverXComponent() {
    XComponent({
        // Loading the video container
```

```
        id: 'xComponent',
        type: XComponentType.SURFACE,
        controller: this.xComponentController
    })
        .visibility(this.XComponentFlag ? Visibility.Visible : Visibility.Hidden)
        .onLoad(() => {
          this.surfaceId = this.xComponentController.getXComponentSurfaceId();
          this.avPlayManage.initPlayer(this.surfaceId, (avPlayer: media.AVPlayer) => {
            this.percent = avPlayer.width / avPlayer.height;
            this.setVideoWH();
            this.isCalcWHFinished = true;
            this.durationTime = this.avPlayManage.getDurationTime();
            setInterval(() => { // Update the current time.
              if (!this.isSwiping) {
                this.currentTime = this.avPlayManage.getCurrentTime();
              }
            }, SET_INTERVAL);
          })
        })
        .height(this.isCalcWHFinished ? `${this.surfaceH}px` : '100%')
        .width(this.isCalcWHFinished ? `${this.surfaceW}px` : '100%')
  }
```

```
build() {
    Stack() {
        Column() {
            this.CoverXComponent()
        }
        .align(Alignment.TopStart)
        .margin({ top: $r('app.float.size_80') })
        .id('Video')
        .justifyContent(FlexAlign.Center)

        Text()
            .height(`${this.surfaceH}px`)
            .width(`${this.surfaceW}px`)
            .margin({ top: $r('app.float.size_80') })
            .backgroundColor(Color.Black)
            .opacity($r('app.float.size_zero_five'))
            .visibility(this.isSwiping ? Visibility.Visible : Visibility.Hidden)

        Row() {
            Text(timeConvert(this.currentTime))
                .fontSize($r('app.float.size_24'))
```

```
                .opacity($r('app.float.size_1'))
                .fontColor($r("app.color.slider_selected"))
            Text("/" + timeConvert(this.durationTime))
                .fontSize($r('app.float.size_24'))
                .opacity($r('app.float.size_1'))
                .fontColor(Color.White)
        }
        .margin({ top: $r('app.float.size_80') })
        .visibility(this.isSwiping ? Visibility.Visible : Visibility.Hidden)
    }
    .onClick(() => {
        this.isClickScreen = !this.isClickScreen;
        if (this.isClickScreen) {
            this.setTimer();
        } else {
            this.clearTimer();
        }
    })
    .backgroundColor(Color.Black)
    .height('100%')
    .width('100%')
    .padding({ top: '36vp', bottom: '28vp'})
```

```
  }
}
```

这里最核心的代码已经用红色字体标出来，其中最最核心的代码是下面红色字体：

```
  @Builder
  CoverXComponent() {
    XComponent({
      // Loading the video container
      id: 'xComponent',
      type: XComponentType.SURFACE,
      controller: this.xComponentController
    })
      .visibility(this.XComponentFlag ? Visibility.Visible : Visibility.Hidden)
      .onLoad(() => {
        this.surfaceId = this.xComponentController.getXComponentSurfaceId();
        this.avPlayManage.initPlayer(this.surfaceId, (avPlayer: media.AVPlayer) => {
          this.percent = avPlayer.width / avPlayer.height;
          this.setVideoWH();
          this.isCalcWHFinished = true;
          this.durationTime = this.avPlayManage.getDurationTime();
          setInterval(() => { // Update the current time.
            if (!this.isSwiping) {
              this.currentTime = this.avPlayManage.getCurrentTime();
```

```
            }
        }, SET_INTERVAL);
      })
    })
    .height(this.isCalcWHFinished ? `${this.surfaceH}px` : '100%')
    .width(this.isCalcWHFinished ? `${this.surfaceW}px` : '100%')
  }
```

关于 XComponent，参考这里：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/napi-xcomponent-guidelines-V5#arkts-xcomponent%E5%9C%BA%E6%99%AF

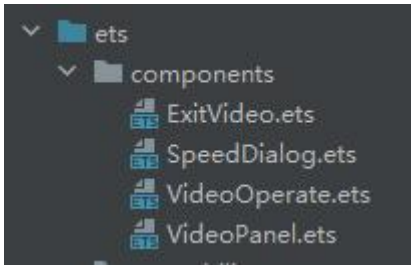
同学们需要认真阅读文档，并参考代码以及借助其他资料，认真阅读和理解相关概念。

通过这个实例，基本上就是：

- 调用 XComponent 组件，设置为 SURFACE 类型
- 生成一个 surfaceId
- 在 onLoad 函数里面，调用咱们的 initPlayer 函数，传入 surfaceId，在回调函数里面设置好宽度和高度等信息
- 函数 initPlayer 调用了之后，才会开始创建 AVPlayer 实例，才会加载内容，才会继续下去，开始播放视频。

到这一步，在模拟器上运行，可以看到视频终于被播放了：

后续的步骤，请自己添加 components 目录，在里面加上四个子组件：



然后在 Index.ets 调用这些子组件。

这些就留给大家自己探索。最后的效果：

上方多了一个导航栏，其实就是结束 APP，用子组件 ExitVideo.ets 中的代码实现。

下方多了一个控制条，包括控制播放的倍数，用 VideoOperate.ets 以及 SpeedDialog.ets 中的子组件实现。点击播放倍数的按钮效果：

还多了一个 Panel，可以选择另外一个视频，或者网络视频：

比如我选择了网络视频，见到了大佬：

非常有趣的实例，请自行探索。

五、实验注意事项

1. 注意教师的操作演示。

2. 学生机与教师机内网连通，能接收和提交实验结果。

3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

六、思考题

1. 通过这个实验，你学到了什么？