

【Codelab】懒人“看”书新法—鸿蒙语音播报，到底如何实现？

现代社会节奏较快，人们看书可能不仅仅只用眼睛，有时候也会用耳朵来“听书”，语音播报由此诞生，并通过人工智能语音识别引擎实现。HarmonyOS 基于华为智慧引擎 (HUAWEI HiAI Engine) 中的语音识别引擎，向开发者提供人工智能应用层 API，该技术提供将文本转换为语音并进行播报的能力，可应用于以下两种场景：

•实时语音交互

生成音频信息用于语音交互，例如与智能音箱或手机智能助手的交互，后台会将回答的信息以音频方式进行语音播报。

•超长文本播报

用于小说、新闻等较长文本的自动朗读。

本期我们就为大家带来超长文本播报场景下的基于 AI 语音播报能力的 Codelab。当用户输入相关文本内容时，点击“语音播放”按钮，程序即对文本进行播报并同步记录语音播报的耗时时长，并呈现在页面上，是不是能满足计时“听书”的需求呢？让我们一起来看看吧。

首先，让我们梳理一遍开发要点：

1) UI 页面的构建

- 2) 语音播报接口调用
- 3) 计时器的创建
- 4) 线程间通信处理机制的使用

请注意，由于需要时刻进行观察，在逻辑代码实现中我们会穿插 HiLog 日志打印，下面我们会逐一指出。

在正式开始敲代码之前，开发者们需要先下载安装 Huawei DevEco Studio，如果对这个流程不甚熟悉，可以参照官网的教程来操作。Huawei DevEco Studio 安装指南：

[https://developer.harmonyos.com/cn/docs/documentation/doc-guides/software_i
nstall-0000001053582415](https://developer.harmonyos.com/cn/docs/documentation/doc-guides/software_install-0000001053582415)

【注意】本次 Codelab 针对的是步骤拆解和重点讲解，限于篇幅原因不会展示完整代码，开发者们可在文末 **【阅读原文】** 中获取完整代码哦~

我们打开 Huawei DevEco Studio，选择 Phone 中的 Empty Feature Ability(Java)模板工程，本次 Codelab 我们将在该模板下完成。有如下操作：

1. 我们将在 \entry\src\main\resources\base\layout\ability_main.xml 中构建 UI 页面；
2. 我们在 \entry\src\main\resources\base\graphic\ 目录下新建 background_button.xml 文件用于实现“语音播报”按钮的样式优化；

3. 文 中 的 逻 辑 代 码 我 们 将 在
\entry\src\main\java\com\example\ailistener\slice\MainAbilitySlice.java 文件中实
现；让我们马上开始。

1: Project

Project

- Allistener D:\Allistener
 - .gradle
 - .idea
 - build
 - entry
 - build
 - libs
 - src
 - main
 - java
 - com.example.allistener
 - slice
 - MainAbilitySlice
 - MainAbility
 - MyApplication
 - resources
 - base
 - element
 - graphic
 - background_ability_main.xml
 - background_button.xml
 - layout
 - ability_main.xml
 - media
 - profile
 - rawfile
 - config.json
 - ohosTest
 - test
 - .gitignore
 - build.gradle
 - proguard-rules.pro
 - gradle
 - .gitignore
 - build.gradle

1) UI 界面构建

纵观这个页面，主要分为以下几个部分：

•标题

即“AI 语音播报”这几个字，这里我们使用 Text 组件。

•文本输入框

可供用户输入想要播报的文本内容，最大不超过 100,000 个字符。为了便于大家理解，这里我们已经给大家准备了一段文本，我们使用 TextField 组件来完成。

•播报按钮

此处展示的文本是“语音播报”，使用的是 Button 组件。值得注意的是，这里需要优化按钮样式，如添加阴影及优化其为胶囊按钮，让按钮更为醒目美观。

如前面提到的，我们将在 background_button.xml 文件中优化按钮样式，通过 color 设置按钮背景颜色，通过 radius 的半径实现圆角，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>

<shape xmlns:ohos="http://schemas.huawei.com/res/ohos"
       ohos:shape="rectangle">

    <corners
```

```
    ohos:radius="40"/>

    <solid

        ohos:color="#e9e9e9"/>

    </shape>
```

•计时文本

用于显示“播报耗时：0 s”文本，同样使用 Text 组件完成。



2) 语音播报接口调用

构建完了页面，我们来到今天的重头戏之一，也就是使用 AI 语音播报能力开发程序。语音播报（Text to Speech，以下简称 TTS），提供将文本转换为语音并进行播报的能力。

•语音播报官网资料

来源：HarmonyOS 微信号 https://mp.weixin.qq.com/s/tYiCGO-WRXOC9_pfopbiyw

<https://developer.harmonyos.com/cn/docs/documentation/doc-guides/ai-tts-overview-0000001050724400>

这里我们主要分三个部分实现，创建 TTS 客户端、TTS 客户端的初始化和调用相关方法对文本进行播报。下面我们来详细看看各个接口如何调用。

1. TTS 客户端创建

调用 void create 接口创建 TTS 客户端。

```
private void initTtsEngine() {  
  
    TtsClient.getInstance().create(this, ttsListener);  
  
}
```

2. TTS 客户端的初始化

当 TTS 客户端创建成功，即 eventType 取值

TtsEvent.CREATE_TTS_CLIENT_SUCCESS 时，进行 TTS 客户端的初始化。

```
public void onEvent(int eventType, PacMap pacMap) {  
  
    HiLog.info(LABEL_LOG, "onEvent...");  
  
    // 定义 TTS 客户端创建成功的回调函数  
  
    if (eventType == TtsEvent.CREATE_TTS_CLIENT_SUCCESS) {  
  
        TtsParams ttsParams = new TtsParams();  
  
        ttsParams.setDeviceId(UUID.randomUUID().toString());  
  
        initItsResult = TtsClient.getInstance().init(ttsParams);  
  
    }  
}
```

```
}
```

同时我们引入 HiLog 日志打印，便于观察相关情况。

3.调用相关方法对文本进行播报

这里我们调用 `TtsClient.getInstance().speakText()` 方法对文本进行播报，同样也引入 HiLog 日志打印用于观察初始化是否成功。

```
private void readText(Component component) {  
  
    if (initItsResult) {  
  
        HiLog.info(LABEL_LOG, "initItsResult is true, speakText");  
  
        TtsClient.getInstance().speakText(infoText.getText(), null);  
  
    } else {  
  
        HiLog.error(LABEL_LOG, "initItsResult is false");  
  
    }  
}
```

3) 计时器的创建

本 Codelab 将以秒为单位对 AI 语音播报速度进行计时，故而我们需要一个计时器。在 HarmonyOS 中，我们通过计时器 `Timer` 和计时器任务 `TimerTask` 类来实现。这里使用到的是构建和取消两种方法，比较简单。大家可以通过官网资料进一步了解。

•Timer

<https://developer.harmonyos.com/cn/docs/documentation/doc-references/timer-0000001054358579>

•TimerTask

<https://developer.harmonyos.com/cn/docs/documentation/doc-references/timertask-0000001054558601>

同样我们使用 HiLog 日志打印来观察文本语音播报的开始和结束。

4) 线程间通信处理机制的使用

接下来我们将提到本 Codelab 另外一个重头戏——线程间通信处理机制的使用。在启动应用时，系统会为该应用创建一个称为“主线程”的执行线程。该线程随着应用创建或消失，是应用的核心线程。具体到本 Codelab，UI 界面的显示和更新等操作，就是更新播报耗时的界面，是在主线程上进行的，因此主线程也称为 UI 线程。示例中分配的是 9015，如图所示：

```
02-20 11:26:57.117 9015-9015/com.huawei.codedemo I 01100/MainAbilitySlice: 播报耗时: 0 s
...
02-20 11:27:09.575 9015-9015/com.huawei.codedemo I 01100/MainAbilitySlice: 播报耗时: 12 s
```

然而在实际项目中，开发者可能面临许多耗时的操作，比如说下载文件、查询数据库，具体到本 Codelab，就是语音播报功能和计时器功能，这些复杂的操作会阻塞 UI 线程，导致界面无响应，带来非常不好的用户体验。

因此，我们需要将这些耗时操作放到子线程中，避免阻塞主线程，比如在示例中，我们把 AI 语音播报放在子线程 9275 中执行：

```
02-20 11:26:57.111 9015-9275/com.huawei.codedemo I 01100/MainAbilitySlice: onSpeechStart...
```

但同时，我们又需要把操作的结果数据反馈给 UI 线程，这个时候就必须引入线程间通信处理机制。因此，HarmonyOS 给 Java 应用开发提供了 EventHandler 机制，可以通过 EventRunner 创建新线程，将耗时的操作放到新线程上执行。这样既不阻塞原来的线程，任务又可以得到合理的处理。

每一个 EventHandler 和指定的 EventRunner 所创建的新线程绑定，并且该新线程内部有一个事件队列。EventHandler 可以投递指定的 InnerEvent 事件或 Runnable 任务到这个事件队列。

EventRunner 从事件队列里循环地取出事件：

- 1) 如果取出的事件是 InnerEvent 事件，将在 EventRunner 所在线程执行 processEvent 回调；
- 2) 如果取出的事件是 Runnable 任务，将在 EventRunner 所在线程执行 Runnable 的 run 回调。

•线程间通信开发概述

<https://developer.harmonyos.com/cn/docs/documentation/doc-guides/inter-thread-overview-0000000000038958>

在本例中，开始发音的时候发送 EVENT_MSG_TIME_COUNT 事件，此时程序开始计时并更新 UI 页面，示例代码如下所示：

```
@Override  
public void onSpeechStart(String utteranceId) {  
    // 开始计时  
    HiLog.info(LABEL_LOG, "onSpeechStart...");  
    if (timer == null && timerTask == null) {  
        timer = new Timer();  
        timerTask = new TimerTask() {  
            public void run() {  
                handler.sendEvent(EVENT_MSG_TIME_COUNT);  
            }  
        };  
        timer.schedule(timerTask, 0, 1000);  
    }  
}
```

此时取出的事件是 Runnable，需要将 Runnable 任务投递到新的线程，在 EventRunner

所在线程执行 Runnable 的 run 回调，并按照优先级和延时进行处理，。这里是同步更新

UI 页面，代码如下所示：

```
private EventHandler handler = new EventHandler(EventRunner.current()) {  
  
    @Override  
  
    protected void processEvent(InnerEvent event) {  
  
        switch (event.eventId) {  
  
            case EVENT_MSG_TIME_COUNT:  
  
                getUITaskDispatcher().delayDispatch(new Runnable() {  
  
                    @Override  
  
                    public void run() {  
  
                        time = time + 1;  
  
                        HiLog.info(LABEL_LOG, " 播 报 耗 时 : " +  
                        Integer.toString(time) + " s");  
  
                        timeText.setText("播报耗时: " + Integer.toString(time) + "  
s");  
  
                    }  
  
                }, 0);  
  
                break;  
  
            default:  
  
                break;  
  
        }  
    }  
}
```

```
};
```

至此，我们已经完成本次 Codelab 的所有关键步骤。通过这个 Codelab，大家可以学习到 AI 语音播报、线程间通信和计时器的使用方法。