

本示例将演示如何编写简单业务，输出“Hello World”，初步了解HarmonyOS 如何

运行在开发板上。

参考文档：

https://device.harmonyos.com/cn/docs/start/introduce/oem_wifi_start-0000001050168544

添加功能小结：

1、创建App

```
1 # 1、创建业务文件夹路径
2 ./applications/sample/wifi-iot/app/first_app
3 # 2、添加实现文件及源码
4 ./applications/sample/wifi-iot/app/first_app/*
5 #3、编写模块BUILD.gn文件
6 ./applications/sample/wifi-iot/app/first_app/build.gn
7 #4、修改app下build.gn
8
```

2、编译

```
1 python build.py wifiiot
```

修改源码

bugfix和新增业务两种情况，涉及源码修改。下面以新增业务（my_first_app）为例，向开发者介绍如何进行源码修改。

1. 确定目录结构。

开发者编写业务时，务必先在./applications/sample/wifi-iot/app路径下新建一个目录（或一套目录结构），用于存放业务源码文件。

例如：在app下新增业务my_first_app，其中hello_world.c为业务代码，BUILD.gn为编译脚本，具体规划目录结构如下：

```
1 .
2 └─ applications
3     └─ sample
4         └─ wifi-iot
5             └─ app
6                 └─ my_first_app
7                     └─ hello_world.c
8                     └─ BUILD.gn
9 └─ BUILD.gn
```

1. 编写业务代码。

新建./applications/sample/wifi-iot/app/my_first_app下的hello_world.c文件，在hello_world.c中新建业务入口函数HelloWorld，并实现业务逻辑。并在代码最下方，使用HarmonyOS启动恢复模块接口SYS_RUN()启动业务。（SYS_RUN定义在ohos_init.h文件中）

```
1 #include <stdio.h>
2 #include "ohos_init.h"
3 #include "ohos_types.h"
4
5 void HelloWorld(void)
6 {
7     printf("[DEMO] Hello world.\n");
8 }
9 SYS_RUN(HelloWorld);
```

1. 编写用于将业务构建成静态库的BUILD.gn文件。

新建./applications/sample/wifi-iot/app/my_first_app下的BUILD.gn文件，并完成如下配置。

如步骤1所述，BUILD.gn文件由三部分内容（目标、源文件、头文件路径）构成，需由开发者完成填写。。

- static_library中指定业务模块的编译结果，为静态库文件libmyapp.a，开发者根据实际情况完成填写。

- sources中指定静态库.a所依赖的.c文件及其路径，若路径中包含"//"则表示绝对路径（此处为代码根路径），若不包含"//"则表示相对路径。
- include_dirs中指定source所需要依赖的.h文件路径。

```
1 static_library("myapp") {
2     sources = [
3         "hello_world.c"
4     ]
5     include_dirs = [
6         "//utils/native/lite/include"
7     ]
8 }
```

1. 编写模块BUILD.gn文件，指定需参与构建的特性模块。

配置./applications/sample/wifi-iot/app/BUILD.gn文件，在features字段中增加索引，使目标模块参与编译。features字段指定业务模块的路径和目标，以my_first_app举例，features字段配置如下。

- my_first_app是相对路径，指向./applications/sample/wifi-iot/app/my_first_app/BUILD.gn。
- myapp是目标，指向./applications/sample/wifi-iot/app/my_first_app/BUILD.gn中的static_library("myapp")。

```
1 import("//build/lite/config/component/lite_component.gni")
2
3 lite_component("app") {
4     features = [
5         "my_first_app:myapp",
6     ]
7 }
```

调测验证

目前调试验证的方法有两种，分别为通过printf打印日志、通过asm文件定位panic问题，开发者可以根据具体业务情况选择。

由于本示例业务简单，采用printf打印日志的调试方式即可。下面开始介绍这两种调试手段的使用方法。

printf打印

代码中增加printf维测，信息会直接打印到串口上。开发者可在业务关键路径或业务异常位置增加日志打印，如下所示。

```
1 void HelloWorld(void)
2 {
3     printf("[DEMO] Hello world.\n");
4 }
```

根据asm文件进行问题定位

系统异常退出时，会在串口上打印异常退出原因调用栈信息，如下文所示。通过解析异常栈信息可以定位异常位置。

```
1 =====KERNEL PANIC=====
2 *****Call Stack*****
3 Call Stack 0 -- 4860d8 addr:f784c
4 Call Stack 1 -- 47b2b2 addr:f788c
5 Call Stack 2 -- 3e562c addr:f789c
6 Call Stack 3 -- 4101de addr:f78ac
7 Call Stack 4 -- 3e5f32 addr:f78cc
8 Call Stack 5 -- 3f78c0 addr:f78ec
9 Call Stack 6 -- 3f5e24 addr:f78fc
10 *****Call Stack end*****
```

为解析上述调用栈信息，需要使用到Hi3861_wifiiot_app.asm文件，该文件记录了代码中函数在Flash上的符号地址以及反汇编信息。asm文件会随版本大包一同构建输出，存放在./out/wifiiot/路径下。

1. 将调用栈CallStack信息保存到txt文档中，以便于编辑。（可选）
2. 打开asm文件，并搜索CallStack中的地址，列出对应的函数名 信息。通常只需找出前几个栈信息对应的函数，就可明确异常代码方向。

```
1 Call Stack 0 -- 4860d8 addr:f784c -- WadRecvCB
2 Call Stack 1 -- 47b2b2 addr:f788c -- wal_sdp_process_rx_data
3 Call Stack 2 -- 3e562c addr:f789c
4 Call Stack 3 -- 4101de addr:f78ac
5 Call Stack 4 -- 3e5f32 addr:f78cc
```

```
6 Call Stack 5 -- 3f78c0 addr:f78ec
7 Call Stack 6 -- 3f5e24 addr:f78fc
```

1. 根据以上调用栈信息，可以定位WadRecvCB函数中出现了异常。

```
276537 0048607a-<WadRecvCB>:
276538 ..48607a:→fb0742ef.....→jal>t0,3fa82a-<__riscv_save_4>
276539 ..48607e:→0011d7b7.....→lui>a5,0x11d
276540 ..486082:→89b6.....→mv→s3,a3
276541 ..486084:→7c07a683.....→lw→a3,1984(a5).#.11d7c0-<__stack_chk_guard>
276542 ..486088:→1101.....→addi→sp,sp,-32
276543 ..48608a:→843e.....→mv→s0,a5
276544 ..48608c:→ce36.....→sw→a3,28(sp)
276545 ..48608e:→c509.....→beqz→a0,486098-<WadRecvCB+0x1e>
276546 ..486090:→84ae.....→mv→s1,a1
276547 ..486092:→c199.....→beqz→a1,486098-<WadRecvCB+0x1e>
276548 ..486094:→8932.....→mv→s2,a2
276549 ..486096:→ee19.....→bnez→a2,4860b4-<WadRecvCB+0x3a>
276550 ..486098:→0049.7afc.051f.....→l.li→a0,0x497afc
276551 ..48609e:→c0e740ef.....→jal>ra,3fa4ac-<printf>
276552 ..4860a2:→4781.....→li→a5,0
276553 ..4860a4:→853e.....→mv→a0,a5
276554 ..4860a6:→4772.....→lw→a4,28(sp)
276555 ..4860a8:→7c042783.....→lw→a5,1984(s0)
276556 ..4860ac:→02f70763.....→beq>a4,a5,4860da-<WadRecvCB+0x60>
276557 ..4860b0:→908700ef.....→jal>ra,3f61b8-<__stack_chk_fail>
276558 ..4860b4:→862a.....→mv→a2,a0
276559 ..4860b6:→4699.....→li→a3,6
276560 ..4860b8:→4599.....→li→a1,6
276561 ..4860ba:→000e.2e34.051f.....→l.li→a0,0xe2e34
276562 ..4860c0:→c63a.....→sw→a4,12(sp)
276563 ..4860c2:→f60750ef.....→jal>ra,3fb822-<memcpy_s>
276564 ..4860c6:→57fd.....→li→a5,-1
276565 ..4860c8:→fd71.....→bnez→a0,4860a4-<WadRecvCB+0x2a>
276566 ..4860ca:→4732.....→lw→a4,12(sp)
276567 ..4860cc:→864e.....→mv→a2,s3
276568 ..4860ce:→85ca.....→mv→a1,s2
276569 ..4860d0:→86ba.....→mv→a3,a4
276570 ..4860d2:→8526.....→mv→a0,s1
276571 ..4860d4:→eb5ff0ef.....→jal>ra,485f88-<WadBusinessProc>
276572 ..4860d8:→b7e9.....→j→4860a2-<WadRecvCB+0x28>
276573 ..4860da:→6105.....→addi→sp,sp,32
276574 ..4860dc:→f827406f.....→j→3fa85e-<__riscv_restore_4>
```

2. 完成代码排查及修改。

运行结果

示例代码编译、烧录、运行、调测后，在串口界面会显示如下结果：

```
1 ready to OS start
2 FileSystem mount ok.
3 wifi init success!
4 [DEMO] Hello HarmonyOS! ! .
```