第 3 章

OpenHarmony 移植

3.1 轻量级内核移植

3.1.1 LiteOS 内核概述

操作系统是用来管理系统硬件、软件及数据资源，控制程序运行，并为其他应用软件提供支持的一种系统软件。根据不同的种类，又可分为实时操作系统、桌面操作系统、服务器操作系统等。在于一些小型的应用，在系统实时性要求高，硬件资源有限等情况下，应尽量避免使用复杂庞大的操作系统，使用小型的实时操作系统更能满足应用的需求。

OpenHarmony 包含轻量系统、小型系统和标准系统，其针对不同量级的系统，分别使用不同形态的内核，分别有 LiteOS 和 Linux，在轻量系统和小型系统上选择使用 LiteOS 内核，在小型系统和标准系统上选择使用 Linux 内核。

LiteOS 是华为面向物联网领域开发的一个基于实时内核的轻量级操作系统，是 OpenHarmony 体系的内核之一。从本质上说，它就是一个 RTOS，现有基础内核可支持任务管理、内存管理、时间管理、通信机制、中断管理、队列管理、事件管理、定时器等操作系统基础组件，能够更好地支持低功耗场景，支持 tickless 机制，支持定时器对齐。同时 LiteOS 提供端云协同能力，集成了 LwM2M、CoAP、mbedtls、LwIP 全套 IoT 互联协议栈，且在 LwM2M 的基础上，提供了 AgentTiny 模块，用户只需关注自身的应用，而不必关注 LwM2M 实现细节，直接使用 AgentTiny 封装的接口即可简单快速地实现与云平台安全可靠的连接。LiteOS 又分为 LiteOS-M 和 LiteOS-A，前者主要针对轻量系统，后者主要针对小型系统和标准系统。RK2206 是 MCU 级芯片，适合使用 LiteOS-M 内核，接下来讲解 LiteOS-M 如何移植到 RK2206 芯片上。

3.1.2 LiteOS 移植适配

LiteOS 有两种移植方案：接管中断和非接管中断方式。在接管中断方式下，由 LiteOS 创建并管理中断，需要修改 RK2206 芯片启动文件，移植比较复杂。RK2206 的中断管理做得很好，不用由 LiteOS 管理中断，所以下面介绍的移植方案，都是非接管中断方式的。中断

的使用与在裸机工程中是一样的。

另外，本书是基于 OpenHarmony 3.0LTS 源代码与 RK2206 芯片做的适配，因为 RK2206 芯片资源有限，所以采用 LiteOS-M 作为操作系统。下述的 OpenHarmony 默认就是 OpenHarmon 3.0LTS，LiteOS-M 默认就是 LiteOS。

1. 下载 OpenHarmony 源代码

OpenHarmony 源代码获取方式主要分为以下两种。

1）通过 repo＋ssh 下载

```
repo init -u git@gitee.com:openharmony/manifest.git -b refs/tags/OpenHarmony-v3.0-LTS
--no-repo-verify
repo sync -c
repo forall -c 'git lfs pull'
```

2）通过 repo＋https 下载

```
repo init -u https://gitee.com/openharmony/manifest.git -b refs/tags/OpenHarmony-v3.0-
LTS --no-repo-verify
repo sync -c
repo forall -c 'git lfs pull'
```

2. 创建工程

在 OpenHarmony 主目录下创建 device/rockchip 目录，如图 3.1.1 所示。

```
lingzhi@lingzhi:/svn/mouxian/gitee/lockzhiner-rk2206-openharmony3.0lts/device/rockchip$ tree -L 2
.
├── hardware
│   ├── BUILD.gn
│   ├── docs
│   ├── lib
│   └── libhardware.a
├── LICENSE
├── README_zh.md
├── rk2206
│   ├── adapter
│   ├── BUILD.gn
│   ├── README_zh.md
│   ├── sdk_liteos
│   └── third_party
└── tools
    ├── linux
    ├── package
    ├── pictures
    ├── README.md
    └── windows

12 directories, 7 files
```

图 3.1.1 device/rockchip 目录

- hardware 文件夹主要存放 RK2206 芯片的固件库。
- rk2206 文件夹主要存放主函数、移植配置文件和相关头文件。
- tools 文件夹主要存放烧写工具、打包工具等。

3. 配置 target_config.h

target_config 对裁剪整个 LiteOS-M 所需功能的宏均做了定义，有些宏定义被使能，有些宏定义被失能，开始时用户暂时只需要配置最简单的功能即可。要想随心所欲地配置 LiteOS 的功能，就必须掌握宏定义的功能。下面介绍宏定义的含义及其修改方法。

1）添加 link.h 头文件

```
/* RK2206 芯片的相关配置,包括内存分配等 */
#include "link.h"
#include "soc.h"
```

将 RK2206 芯片的 link.h 头文件添加到 target_config.h 中。后续一些 LiteOS 配置需要用到该头文件。

2）系统时钟模块配置

```
/* ================================================
系统时钟模块配置
================================================ */
#define OS_SYS_CLOCK                                    96000000UL
#define OSCFG_BASE_CORE_TICK_PER_SECOND                 (1000UL)
#define LOSCFG_BASE_CORE_TICK_HW_TIME                   0
① #define LOSCFG_BASE_CORE_TICK_WTIMER                 0
② #define LOSCFG_BASE_CORE_TICK_RESPONSE_MAX           0xFFFFFFUL
```

其中，OS_SYS_CLOCK 表示 LiteOS 的系统时钟周期，也就是 RK2206 芯片的系统时钟周期。

LOSCFG_BASE_CORE_TICK_PER_SECOND 表示 LiteOS 每秒 Tick 数量，即嘀嗒节拍数。在 LiteOS 中，系统延时和阻塞时间都是以 Tick 为单位的，配置 LOSCFG_BASE_CORE_TICK_PER_SECOND 的值可以改变中断的频率，从而间接改变 LiteOS 的时钟周期（$T=1/f$）。如果将 LOSCFG_BASE_CORE_TICK_PER_SECOND 的值设置为 1000，那么 LiteOS 的时钟周期为 1ms。过高的系统节拍中断频率意味着 LiteOS 内核将占用更多的 CPU 时间，因此会降低效率，一般将 LOSCFG_BASE_CORE_TICK_PER_SECOND 的值设置为 50～1000。

LOSCFG_BASE_CORE_TICK_HW_TIME 是定时器剪裁的外部配置参数，未使用，所以这个宏定义为 0。

① 表示 Tick 参数是否可被修改。

② 表示 Tick 参数的最大数值。超过该数值，系统时钟将重新开始计数。

3）硬件外部中断模块配置

```
/* ============================================================================
硬件外部中断模块配置
============================================================================ */
#define LOSCFG_PLATFORM_HWI                          1
#define LOSCFG_USE_SYSTEM_DEFINED_INTERRUPT          1
#define LOSCFG_PLATFORM_HWI_LIMIT                    128
```

其中，LOSCFG_PLATFORM_HWI 表示 LiteOS 硬件中断定制配置参数，为 1 表示 LiteOS 接管了外部中断，为 0 表示 LiteOS 不接管中断。RK2206 芯片适配设置为 1，即 LiteOS 接管中断。

LOSCFG_USE_SYSTEM_DEFINED_INTERRUPT 表示使用 LiteOS 定义的中断函数，为 1 表示使用 LiteOS 定义的中断函数，为 0 表示未使用 LiteOS 定义的中断函数。RK2206 适配设置为 1，即使用 LiteOS 定义的中断函数。

LOSCFG_PLATFORM_HWI_LIMIT 表示 LiteOS 支持最大的外部中断数。RK2206 适配设置为 128，即 LiteOS 支持最大的外部中断数为 128 个。

4）任务模块配置

```
/* ============================================================================
任务模块配置
============================================================================ */
#define LOSCFG_BASE_CORE_TSK_LIMIT                   63
#define LOSCFG_BASE_CORE_TSK_IDLE_STACK_SIZE         (0x1000U)
#define LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE      (0x1000U)
#define LOSCFG_BASE_CORE_TSK_MIN_STACK_SIZE          (0x200U)
#define LOSCFG_BASE_CORE_TIMESLICE                   1
#define LOSCFG_BASE_CORE_TIMESLICE_TIMEOUT           20000
```

其中，LOSCFG_BASE_CORE_TSK_LIMIT 表示 LiteOS 支持最大任务个数（除去空闲任务），一般默认为 63。

LOSCFG_BASE_CORE_TSK_IDLE_STACK_SIZE 表示 LiteOS 空闲任务的栈大小，默认为 0x500U 字节。

LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE 表示 LiteOS 任务的默认栈大小。

LOSCFG_BASE_CORE_TSK_MIN_STACK_SIZE 表示 LiteOS 任务的最小栈大小。

LOSCFG_BASE_CORE_TIMESLICE 表示 LiteOS 是否使用时间片轮转方法，1 为使能，0 为禁用，建议为 1。

LOSCFG_BASE_CORE_TIMESLICE_TIMEOUT 表示 LiteOS 相同优先级的任务最

长执行时间，单位为时钟节拍周期。

5）信号量模块配置

```
/* ============================================================
    信号量模块配置
============================================================ */
#define LOSCFG_BASE_IPC_SEM         1
#define OSCFG_BASE_IPC_SEM_LIMIT    48          // LiteOS 最大支持信号量的个数
```

其中，LOSCFG_BASE_IPC_SEM 表示信号量模块是否启用，1 为启用，0 为禁用，建议为 1。

LOSCFG_BASE_IPC_SEM_LIMIT 表示 LiteOS 最大支持信号量的个数，建议为 48。

6）互斥锁模块配置

```
/* ============================================================
    互斥锁模块配置
============================================================ */
#define LOSCFG_BASE_IPC_MUX             1
#define LOSCFG_BASE_IPC_MUX_LIMIT       48
```

其中，LOSCFG_BASE_IPC_MUX 表示互斥锁模块是否启用，1 为启用，0 为禁用，建议为 1。

LOSCFG_BASE_IPC_MUX_LIMI 表示 LiteOS 最大支持互斥锁的个数，建议为 48。

7）消息队列模块配置

```
/* ============================================================
    消息队列模块配置
============================================================ */
#define LOSCFG_BASE_IPC_QUEUE           1
#define LOSCFG_BASE_IPC_QUEUE_LIMIT     48
```

其中，LOSCFG_BASE_IPC_QUEUE 表示消息队列模块是否启用，1 为启用，0 为禁用，建议为 1。

LOSCFG_BASE_IPC_QUEUE_LIMIT 表示 LiteOS 最大支持消息队列的个数，建议为 48。

8）软件定时器模块配置

```
/* ============================================================
    软件定时器模块配置
============================================================ */
#define LOSCFG_BASE_CORE_SWTMR          1
#define LOSCFG_BASE_CORE_SWTMR_ALIGN    0
#define LOSCFG_BASE_CORE_SWTMR_LIMIT    48
```

其中，LOSCFG_BASE_CORE_SWTMR 表示软件定时器模块是否启用，1 为启用，0 为禁用，建议为 1。

LOSCFG_BASE_CORE_SWTMR_ALIGN 表示软件定时器对齐功能，1 为对齐，0 为不对齐，建议为 0。

LOSCFG_BASE_CORE_SWTMR_LIMIT 表示 LiteOS 最大支持软件定时器的个数，建议为 48。

9）内存模块配置

```
/* ============================================================================
    内存模块配置
 ============================================================================ */
#define LOSCFG_MEM_MUL_POOL                         1
#define OS_SYS_MEM_NUM                              20
①#define LOSCFG_MEM_FREE_BY_TASKID                  1
②#define LOSCFG_BASE_MEM_NODE_INTEGRITY_CHECK       1
③#define LOSCFG_MEM_LEAKCHECK                       1
#define LOSCFG_SYS_EXTERNAL_HEAP                    1
④extern unsigned int _heap_start;
#define LOSCFG_SYS_HEAP_ADDR                        &_heap_start
#define LOSCFG_SYS_HEAP_SIZE              (PSRAM_SIZE - SYS_STACK_SIZE)
```

其中，LOSCFG_MEM_MUL_POOL 表示内存模块内存池功能，1 为启用，0 为禁用，建议为 1。

OS_SYS_MEM_NUM 表示 LiteOS 支持最大内存数量，建议为 20。

① 表示由任务来释放内存。

② 表示是否进行内存节点完整性检查。1 为启用，0 为禁用。

③ 表示是否开启内存泄漏检测机制。1 为启用，0 为禁用。开启该功能时，内存机制会自动记录申请内存时的函数调用关系，一旦出现内存泄漏，系统会将这些信息打印出来。

LOSCFG_SYS_EXTERNAL_HEAP 表示 LiteOS 是否支持外部堆，1 为启用，0 为禁用，建议为 1。

④ 表示定义堆的起始地址。

LOSCFG_SYS_HEAP_ADDR 表示 LiteOS 定义内存堆的起始地址。

LOSCFG_SYS_HEAP_SIZE 表示 LiteOS 定义内存堆的内存总大小。

10）任务栈监控模块配置

```
/*  ============================================================================
    任务栈监控模块配置
 ============================================================================ */
#define LOSCFG_BASE_CORE_TSK_MONITOR 1 // 任务栈监控模块功能,1 为打开,0 为关闭
①#define LOSCFG_BASE_CORE_CPUP          1
// 任务执行过滤器钩子函数配置,默认为 1
#define LOSCFG_BASE_CORE_EXC_TSK_SWITCH  1
```

其中，LOSCFG_BASE_CORE_TSK_MONITOR 表示任务栈监控模块功能，1 为打开，0 为关闭，建议为 1。

LOSCFG_BASE_CORE_TSK_MONITOR 用于表示开启 CPUP 模块初始化。1 表示启用，0 表示禁用。其中 CPUP（Central Processing Unit Percentage，CPU 占用率）主要用于获知当前系统中各个任务的 CPU 占用情况。

LOSCFG_BASE_CORE_EXC_TSK_SWITCH 表示监控栈的任务执行过滤器钩子函数配置，建议为 1。

编辑完毕后，将 target_config.h 放入到 sdk_liteos/include 目录中。

4. 配置 config.gni

在 sdk_liteos 目录下创建 config.gni 文件，指明整个工程编译器相关信息，包括编译器、编译选项以及编译模块内容。

```
# Copyright (c) 2022 FuZhou Lockzhiner Electronic Co., Ltd. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Kernel type, e.g. "linux", "liteos_a", "liteos_m".
kernel_type = "liteos_m"
# Kernel version.
kernel_version = "1.0.1"
# Board CPU type, e.g. "cortex-a7", "riscv32".
board_cpu = "cortex-m4"
# Board arch, e.g. "armv7-a", "rv32imac".
board_arch = ""
# Toolchain name used for system compiling.
# E.g. gcc-arm-none-eabi, arm-linux-harmonyeabi-gcc, ohos-clang, riscv32-unknown-
elf.
# Note: The default toolchain is "ohos-clang". It's not mandatory if you use the default
toolchain.
board_toolchain = "arm-none-eabi-gcc"
use_board_toolchain = true
# The toolchain path installed, it's not mandatory if you have added toolchain path to your
~/.bashrc.
board_toolchain_path = ""
```

```
# Compiler prefix.
board_toolchain_prefix = "arm-none-eabi-"
# Compiler type, "gcc" or "clang".
board_toolchain_type = "gcc"
# Board related common compile flags.
board_cflags = [
    "-mcpu=cortex-m4",
    "-mthumb",
    "-Wall",
    "-fdata-sections",
    "-ffunction-sections",
    "-DUSE_HAL_DRIVER",
    "-D_STORAGE_LITE_",
    "-D__LITEOS_M__",
    "-D_BSD_SOURCE",
    "-D_GNU_SOURCE",
]

board_cxx_flags = board_cflags
board_ld_flags = board_cflags
# Board related headfiles search path.
board_include_dirs = [
    "//device/rockchip/rk2206/sdk_liteos/include",
    "//device/rockchip/rk2206/adapter/include",
    "//device/rockchip/rk2206/sdk_liteos/board/fs",
    "//device/rockchip/hardware/lib/CMSIS/Device/RK2206/Include",
    "//third_party/cmsis/CMSIS/Core/Include",
    "//kernel/liteos_m/kal/posix/include",
    "//kernel/liteos_m/components/fs/fatfs",
    "//kernel/liteos_m/kernel/include",
    "//kernel/liteos_m/kernel/arch/include",
    "//kernel/liteos_m/kernel/arch/arm/cortex-m4/gcc",
    "//kernel/liteos_m/utils",
    "//kernel/liteos_m/kal/cmsis",
    "//kernel/liteos_m/components/net/lwip-2.1/porting/include",
    "//third_party/FatFs/source",
    "//third_party/musl/arch/arm",
    "//third_party/lwip/src/include",
    "//utils/native/lite/include",
    "//foundation/communication/wifi_aware/interfaces/kits",
    "//foundation/communication/wifi_lite/interfaces/wifiservice",
    "//foundation/communication/ipc_lite/interfaces/kits",
    "//foundation/communication/softbus_lite/interfaces/kits/discovery/",
    "//foundation/communication/softbus_lite/interfaces/kits/transport/",
    "//base/startup/syspara_lite/interfaces/kits",
```

```
    "//base/security/huks/interfaces/innerkits/huks_lite",
    "//utils/native/lite/include",
    "//base/hiviewdfx/hilog_lite/interfaces/native/kits",
    "//third_party/openssl/crypto/ec",
    "//third_party/openssl/include",
    "//kernel/liteos_m/components",
    "//third_party/bounds_checking_function/include",
]
# Board adapter dir for OHOS components.
board_adapter_dir = ""
# Sysroot path.
board_configed_sysroot = ""
# Board storage type, it used for file system generation.
storage_type = "spinor"
rk_third_party_dir = product_config.rk_third_party_dir
```

其中，kernel_type 表示内核类型，建议为 liteos-m。

board_cpu 表示芯片型号，建议为 cortex-m4。

board_toolchain 表示编译器，建议为 arm-none-eabi-gcc。

board_cflags 表示 gcc 预编译选项，建议为

```
-mcpu=cortex-m4 -mthumb -Wall -fdata-sections -ffunction-sections -DUSE_HAL_
DRIVER -D_STORAGE_LITE_ -D__LITEOS_M__ -D_BSD_SOURCE -D_GNU_SOURCE
```

board_cxx_flags 表示 g++ 预编译选项。

board_ld_flags 表示编译链接选项。

board_include_dirs 表示编译头文件路径。

5. 编写 Makefile

在 device/rockchip/rk2206/sdk_liteos 目录下创建 Makefile，用于将所有编译好的静态库链接成 bin 文件，并打包成镜像文件，具体代码如下：

```
TARGET = liteos
BUILD_DIR = $(OUTDIR)
BOARD_DIR = $(shell pwd)
PREFIX = arm-none-eabi-
CC = $(PREFIX)gcc
AS = $(PREFIX)gcc -x assembler-with-cpp
CP = $(PREFIX)objcopy
SZ = $(PREFIX)size
HEX = $(CP) -O ihex
BIN = $(CP) -O binary -S
#########################################
# CFLAGS
#########################################
# cpu
```

```
CPU = -mcpu=cortex-m4
# fpu
FPU = -mfpu=fpv4-sp-d16
# float-abi
FLOAT-ABI = -mfloat-abi=soft
# mcu
MCU = $(CPU) -mthumb # $(FPU) $(FLOAT-ABI)
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# LDFLAGS
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
LDSCRIPT = board.ld
boot_LIBS = -lbootstrap -lbroadcast
hardware_LIBS = -lhal_iothardware -lhardware
hdf_LIBS = -lhdf_config -lhdf_core -lhdf_osal_lite -lhdf_platform_lite
app_LIBS =
xts_LIBS = -lmodule_ActsBootstrapTest -lmodule_ActsDfxFuncTest -lmodule_ActsHieventLiteTest
-lhuks_test_common -lmodule_ActsHuksHalFunctionTest -lmodule_ActsKvStoreTest \
-lmodule_ActsLwipTest -lmodule_ActsParameterTest -lmodule_ActsSamgrTest -lmodule_
ActsUpdaterFuncTest -lmodule_ActsUtilsFileTest \
-lmodule_ActsWifiIotTest -lmodule_ActsWifiServiceTest -lhctest -lmbedtls -lhal_update_
static -lhota

common_LIBS = -larch -lcjson_static -ldump_static -lhal_wifiaware -lhuks_3.0_sdk \
-lnative_file -lsec_static -lwifiaware -lauthmanager -lcmsis -lexchook -lkernel -lpm \
-lsysparam -lwifiservice -lbacktrace -lcppsupport -lhal_file_static -lhichainsdk \
-lposix -ltoken_static -lboard -lcpup -lhievent_lite -lmbedtls -lsamgr -ltrans_
service \
-ldiscovery -lhal_sysparam -lhilog_lite -lmusl-c -lsamgr_adapter -lutils \
-llwip -lhal_token_static -lhiview_lite -lmusl-m -lsamgr_source -lutils_kv_store \
-lpahomqtt_static -llzlittlefs
LIBS = -Wl,--start-group \
        -Wl,--whole-archive $(boot_LIBS) $(hardware_LIBS) $(hdf_LIBS) $(app_LIBS) \
-Wl,--no-whole-archive \
    $(common_LIBS) \
    -Wl,--end-group

LIB_DIR = -L$(BUILD_DIR)/libs

LDFLAGS = $(MCU) \
    --specs=nosys.specs \
    -T$(LDSCRIPT) \
    $(LIB_DIR) \
    -Wl,--start-group $(LIBS) -Wl,--end-group \
    -Wl,-Map=$(BUILD_DIR)/$(TARGET).map,--cref \
    -Wl,--gc-sections
```

```
all:  $(BUILD_DIR)/$(TARGET).elf  $(BUILD_DIR)/$(TARGET).hex  $(BUILD_DIR)/
$(TARGET).bin
$(BUILD_DIR)/$(TARGET).elf:
    $(CC) $(LDFLAGS) -o $@
$(SZ) $@

$(BUILD_DIR)/%.hex: $(BUILD_DIR)/%.elf
$(HEX) $< $@

$(BUILD_DIR)/%.bin: $(BUILD_DIR)/%.elf
$(BIN) $< $@

-include $(wildcard $(BUILD_DIR)/*.d)
```

其中，TARGET 表示编译最终目标为 LiteOS 的 bin 文件。

PREFIX、CC 表示指明交叉编译器。

CPU 表示指定编译 CPU 类型，即 cpu=cortex-m4。

LDSCRIPT 表示指定 RK2206 芯片的 ld 链接脚本。

LIBS 表示指定编译需要链接的静态库。其中，如果有需要调用 LiteOS 的开启自启动机制的静态库，需要在编译时将该静态库添加在-Wl、--whole-archive 和-Wl、--no-whole-archive 之间。

LIB_DIR 表示指定链接静态库的路径。

6. 编写 build.sh

创建 device/rockchip/rk2206/sdk_liteos/build.sh，该 shell 脚本将负责以下 3 部分的工作：

（1）编译 ld 链接脚本；

（2）调用 Makefile 编译 bin 文件，即 LiteOS 二进制文件；

（3）运行打包脚本，打包成引导程序和镜像文件。

build.sh 具体脚本如下：

```
# error out on errors
set -e
CPUs=`sed -n "N;/processor/p" /proc/cpuinfo|wc -l`

OUT_DIR="$1"
SDK_DIR=$(pwd)
TOOLS_DIR=${SDK_DIR}/../../tools/package

PART_SYSTEM_BLOCKS=$(cat include/link.h | grep "#define PART_SYSTEM_BLOCKS" | awk -F' '
{print $3}')
PART_LOADER_BLOCKS=$(cat include/link.h | grep "#define PART_LOADER_BLOCKS" | awk -F' '
'{print $3}')
```

```
PART_LITEOS_BLOCKS = $(cat include/link.h | grep "#define PART_LITEOS_BLOCKS" | awk -F' '
'{print $3}')
PART_ROOTFS_BLOCKS = $(cat include/link.h | grep "#define PART_ROOTFS_BLOCKS" | awk -F' '
'{print $3}')
PART_USERFS_BLOCKS = $(cat include/link.h | grep "#define PART_USERFS_BLOCKS" | awk -F' '
'{print $3}')

LDSCRIPT = board.ld
CFLAGS = "SDK_DIR = ${SDK_DIR} LDSCRIPT = ${LDSCRIPT}"

function main()
{
    # make ld script
    make ${CFLAGS} -f build/link.mk
    # make liteos
    make OUTDIR = ${OUT_DIR} -j${CPUs}
    # make images
    ${TOOLS_DIR}/mkimage.sh ${PART_SYSTEM_BLOCKS} ${PART_LOADER_BLOCKS} ${PART_
LITEOS_BLOCKS} ${PART_ROOTFS_BLOCKS} ${PART_USERFS_BLOCKS}
}

main "$@"
```

其中，make ${CFLAGS} -f build/link.mk 表示编译 ld 链接脚本，该链接脚本的源文件来自 device/rockchip/rk2206/sdk_liteos/build/link.mk。

7. 添加主程序

创建 device/rockchip/rk2206/sdk_liteos/board 文件夹，创建相关源代码文件，具体如图 3.1.2 所示。

```
lingzhi@lingzhi:/svn/mouxian/gitee/lockzhiner-rk2206-openharmony3.0lts/device/rockchip/rk2206/sdk_liteos/board$ tree
.
├── BUILD.gn
├── dprintf.c
├── fs
│   ├── ff_gen_drv.c
│   ├── ff_gen_drv.h
│   └── fs_config.h
├── include
│   └── config_network.h
├── iotmain.c
├── main.c
├── src
│   └── config_network.c
└── startup
    └── startup_rk2206.S

4 directories, 10 files
```

图 3.1.2 board 文件夹

其中，dprintf.c 为将 printf 函数重定向，将串口 1 定义为调试串口信息输出，具体代码如下：

```
/*
 * Copyright (c) 2022 FuZhou Lockzhiner Electronic Co., Ltd. All rights reserved.
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <stdarg.h>
#include <stdio.h>
#include "lz_hardware.h"

#define DEBUG_PORT                  1

int printf(char const *fmt, ...)
{
    va_list ap;
    char buffer[256];

    va_start(ap, fmt);
    vsnprintf(buffer, 256, fmt, ap);
    DebugWrite(DEBUG_PORT, (const unsigned char *)buffer, strlen(buffer));
    DebugPutc(DEBUG_PORT, '\r');
    va_end(ap);
    return 0;
}
```

其中，src/config_network.c 是 WiFi/AP 网络配置，主要用于初始化 LwIP 协议栈，设置 WiFi/AP。该部分源代码与 LiteOS 移植关系不大，故不详述。

startup/startup_rk2206.S 是启动连接脚本。

main.c 的功能包括固件库初始化、LiteOS 初始化、创建任务以及 HDF 服务初始化等，具体内容如下：

```
/*
 * Copyright (c) 2022 FuZhou Lockzhiner Electronic Co., Ltd. All rights reserved.
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

```
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include "los_tick.h"
#include "los_task.h"
#include "los_config.h"
#include "los_interrupt.h"
#include "los_debug.h"
#include "los_compiler.h"
#include "lz_hardware.h"
#include "config_network.h"

#define MAIN_TAG                "MAIN"
int DeviceManagerStart();
void IotInit(void);

/*****************************************************************************
Function     : main
Description  : Main function entry
Input        : None
Output       : None
Return       : None
*****************************************************************************/
LITE_OS_SEC_TEXT_INIT int Main(void)
{
    int ret;
    LZ_HARDWARE_LOGD(MAIN_TAG, "%s: enter ...", __func__);

    HalInit();

    ret = LOS_KernelInit();
    if (ret == LOS_OK) {
        IotInit();
        OHOS_SystemInit();
        /* 开启驱动管理服务 */
        DeviceManagerStart();
        LZ_HARDWARE_LOGD(MAIN_TAG, "%s: LOS_Start ...", __func__);
```

```
        LOS_Start();
    }

        while (1) {
            __asm volatile("wfi");
        }
    }
```

其中，HalInit 用于固件库初始化，包括时钟频率配置、调试串口初始化等。

LOS_KernelInit()主要是初始化 LiteOS 内核。

OHOS_SystemInit()主要是初始化了一些相关模块、系统，包括调用宏函数 MODULE_INIT、SYS_INIT 和函数 SAMGR_Bootstrap()的初始化启动。

DeviceManagerStart()主要是开启驱动管理服务。

LOS_Start()主要是开启 LiteOS 操作系统。

8. 编写 BUILD.gn

创建 device/rockchip/rk2206/sdk_liteos/BUILD.gn，具体内容如下：

```
import("//build/lite/config/component/lite_component.gni")
import("//build/lite/config/subsystem/lite_subsystem.gni")

declare_args() {
  enable_hos_vendor_wifiiot_xts = false
}

lite_subsystem("wifiiot_sdk") {
  subsystem_components = [ ":sdk" ]
}

build_ext_component("liteos") {
  exec_path = rebase_path(".", root_build_dir)
  outdir = rebase_path(root_out_dir)
  command = "sh ./build.sh $ outdir"
  deps = [
    ":sdk",
    "//build/lite:ohos",
  ]
  if (enable_hos_vendor_wifiiot_xts) {
    deps += [ "//build/lite/config/subsystem/xts:xts" ]
  }
}

lite_component("sdk") {
  features = [ ]
```

```
        deps = [
        "//device/rockchip/rk2206/sdk_liteos/board:board",
        "//device/rockchip/hardware:hardware",
        "../third_party/littlefs:lzlittlefs",
        "//build/lite/config/component/cJSON:cjson_static",
        "//vendor/lockzhiner/rk2206/hdf_config:hdf_config",
        "//vendor/lockzhiner/rk2206/hdf_drivers:hdf_drivers",
        "//drivers/adapter/khdf/liteos_m/test/sample_driver:sample_driver",
        "//drivers/adapter/uhdf/manager:hdf_manager",
        "//drivers/adapter/uhdf/posix:hdf_posix",
      ]
    }
```

其中，build_ext_component("liteos")主要是执行当前目录下的 build.sh 脚本。在执行 build.sh 脚本之前，先执行 sdk 和//build/lite：ohos。

sdk 主要是编译指定的功能模块，包括 board(主程序)、hardware(固件库)、lzlittlefs(littlefs 文件系统)、hdf(HDF 驱动服务和 HDF 驱动)。

9. 添加 vendor 文件夹

按照 OpenHarmony 官方目录规范要求，vendor/厂商/开发板名称。在 vendor 文件夹下可创建开发板文件夹，具体如图 3.1.3 所示。

```
lingzhi@lingzhi:/svn/mouxian/gitee/lockzhiner-rk2206-openharmony3.0lts/vendor$ tree -L 2
.
└── lockzhiner
    ├── LICENSE
    ├── README_zh.md
    └── rk2206

2 directories, 2 files
```

图 3.1.3　vendor 文件夹

在 vendor/lockzhiner/rk2206 目录下创建 config.json 文件，具体内容如下：

```
{
    "product_name": "lockzhiner - rk2206",
    "ohos_version": "OpenHarmony 3.0",
    "device_company": "rockchip",
    "board": "rk2206",
    "kernel_type": "liteos_m",
    "kernel_version": "1.0.1",
    "subsystems": [
      {
        "subsystem": "applications",
        "components": [
          { "component": "wifi_iot_sample_app", "features":[ ] }
        ]
```

```
        },
        {
          "subsystem": "iot_hardware",
          "components": [
            { "component": "iot_controller", "features":[ ] }
          ]
        },
        {
          "subsystem": "hiviewdfx",
          "components": [
            { "component": "hilog_lite", "features":[ ] },
            { "component": "hievent_lite", "features":[ ] },
            { "component": "blackbox", "features":[ ] },
            { "component": "hidumper_mini", "features":[ ] }
          ]
        },
        {
          "subsystem": "distributed_schedule",
          "components": [
            { "component": "samgr_lite", "features":[ ] }
          ]
        },
        {
          "subsystem": "security",
          "components": [
            { "component": "hichainsdk", "features":[ ] },
            { "component": "huks", "features":
              [
                "disable_huks_binary = false",
                "disable_authenticate = false",
                "huks_use_lite_storage = true",
                "huks_use_hardware_root_key = true",
                "huks_config_file = \"hks_config_lite.h\""
              ]
            }
          ]
        },
        {
          "subsystem": "startup",
          "components": [
            { "component": "bootstrap_lite", "features":[ ] },
            { "component": "syspara_lite", "features":[ ] }
          ]
        },
        {
          "subsystem": "communication",
```

```
          "components": [
            { "component": "wifi_lite", "features":[ ] },
            { "component": "softbus_lite", "features":[ ] },
            { "component": "wifi_aware", "features":[ ]}
          ]
        },
        {
          "subsystem": "iot",
          "components": [
            { "component": "iot_link", "features":[ ] }
          ]
        },
        {
          "subsystem": "utils",
          "components": [
            { "component": "file", "features":[ ] },
            { "component": "kv_store", "features":[ ] },
            { "component": "os_dump", "features":[ ] }
          ]
        },
        {
          "subsystem": "vendor",
          "components": [
             { "component": "rk2206_sdk", "target": "//device/rockchip/rk2206/sdk_liteos:
wifiiot_sdk", "features":[ ] }
          ]
        },
        {
          "subsystem": "test",
          "components": [
            { "component": "xts_acts", "features":[ ] },
            { "component": "xts_tools", "features":[ ] }
          ]
        }
      ],
      "vendor_adapter_dir": "//device/rockchip/rk2206/adapter",
      "third_party_dir": "//third_party",
      "rk_third_party_dir": "//device/rockchip/rk2206/third_party",
      "product_adapter_dir": "//vendor/lockzhiner/rk2206/hals",
      "ohos_product_type":"",
      "ohos_manufacture":"",
      "ohos_brand":"",
      "ohos_market_name":"",
      "ohos_product_series":"",
      "ohos_product_model":"",
      "ohos_software_model":"",
```

```
    "ohos_hardware_model":"",
    "ohos_hardware_profile":"",
    "ohos_serial":"",
    "ohos_bootloader_version":"",
    "ohos_secure_patch_level":"",
    "ohos_abi_list":""
  }
```

创建 BUILD.gn 文件，具体内容如下：

```
# Copyright (c) 2022 FuZhou Lockzhiner Electronic Co., Ltd. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

group("rk2206") {
}
```

10. 编译工程

在 OpenHarmony 主目录下，配置编译主目录，输入命令：

```
hb set -root
```

选择编译工程，输入命令：

```
hb set
```

选择 lockzhiner-rk2206 编译工程，如图 3.1.4 所示。

```
[OHOS INFO] hb root path: /svn/mouxian/svn/lockzhiner-rk2206-openharmony3.0lts
OHOS Which product do you need?  (Use arrow keys)

lockzhiner
 › lockzhiner-rk2206
```

图 3.1.4　选择 lockzhiner-rk2206 编译工程

编译工程，输入命令：

```
hb build -f
```

编译完成后，出现如图 3.1.5 所示信息表示编译成功。

```
[OHOS INFO] [830/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/sdk.stamp
[OHOS INFO] [831/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/wifiiot_sdk.stamp
[OHOS INFO] [832/834] STAMP obj/build/lite/ohos.stamp
[OHOS INFO] [833/834] ACTION //device/rockchip/rk2206/sdk_liteos:liteos(//build/lite/toolchain:arm-none-eabi-gcc)
[OHOS INFO] [834/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/liteos.stamp
[OHOS INFO] /svn/mouxian/svn/lockzhiner-rk2206-openharmony3.0lts/vendor/lockzhiner/rk2206/fs.yml not found, stop pa
es not need to be packaged, ignore it.
[OHOS INFO] lockzhiner-rk2206 build success
[OHOS INFO] cost time: 0:00:03
```

图 3.1.5 编译成功

编译后，相关镜像文件在 out/rk2206/lockzhiner-rk2206/images/目录下，如图 3.1.6 所示。

```
-rw-rw-r-- 1 lingzhi lingzhi 2097152 4月  12 10:16 Firmware.img
-rw-rw-r-- 1 lingzhi lingzhi      16 4月  12 10:16 Firmware.md5
-rw-rw-r-- 1 lingzhi lingzhi   35093 4月  12 10:16 rk2206_db_loader.bin
```

图 3.1.6 相关文件

注意：如果在上述命令的运行过程中出现错误，则可以输入：

```
python3 -m pip install build/lite
```

3.2 轻量级内核移植测试

3.2.1 测试目的

为了测试轻量级内核移植是否正常，本节用一个小程序测试调试串口和 LiteOS 任务管理。

3.2.2 程序设计

为了完成内核移植测试的目的，在 main()函数中调用一个 task_example()函数，创建两个任务：一个任务每隔 1s 打印一次“Hello World”字符串；另一个任务每隔 2s 打印一次“Hello OpenHarmony”字符串。下面具体说明程序设计步骤。

1. 创建 a0_hello_world 文件夹

在 OpenHarmony 源代码主目录 vendor/lockzhiner/rk2206/samples 下创建 a0_hello_world 文件夹。

2. 创建 hello_world.c 文件

在 a0_hello_world 文件夹下创建 hello_world.c 文件。其中，task_example()函数负责创建两个任务，分别为 task_helloworld 和 task_openharmony，task_helloworld 负责每隔 1s 打印一次“Hello World”字符串，task_openharmony 负责每隔 2s 打印一次“Hello OpenHarmony”字符串的任务。具体代码如下。

```
#include "los_task.h"                                    // OpenHarmony LiteOS 的任务管理头文件

/*****************************************************************
 * 函数名称: task_helloworld
 * 说    明: 线程函数 helloworld
 * 参    数: 无
 * 返 回 值: 无
 *****************************************************************/
void task_helloworld()
{
    while (1)
    {
        printf("Hello World\n");
        /* 睡眠 1s.该函数为 OpenHarmony LiteOS 内核睡眠函数,单位:ms */
        LOS_Msleep(1000);
    }
}

/*****************************************************************
 * 函数名称: task_openharmony
 * 说    明: 线程函数
 * 参    数: 无
 * 返 回 值: 无
 *****************************************************************/
void task_openharmony()
{
    while (1)
    {
        printf("Hello OpenHarmony\n");
        /* 睡眠 1s.该函数为 OpenHarmony 内核睡眠函数,单位:ms */
        LOS_Msleep(2000);
    }
}

/*****************************************************************
 * 函数名称: task_example
 * 说    明: 内核任务创建例程
 * 参    数: 无
 * 返 回 值: 无
 *****************************************************************/
void task_example()
{
    /* 任务 id */
    unsigned int thread_id1;
    unsigned int thread_id2;
    /* 任务参数 */
    TSK_INIT_PARAM_S task1 = {0};
    TSK_INIT_PARAM_S task2 = {0};
    /* 返回值 */
```

```
    unsigned int ret = LOS_OK;

    /* 创建 HelloWorld 任务 */
    task1.pfnTaskEntry = (TSK_ENTRY_FUNC)task_helloworld;   // 运行函数入口
    task1.uwStackSize = 2048;                               // 堆栈大小
    task1.pcName = "task_helloworld";                       // 函数注册名称
    task1.usTaskPrio = 24;                                  // 任务的优先级,为 0～63
    ret = LOS_TaskCreate(&thread_id1, &task1);              // 创建任务
    if (ret != LOS_OK)
    {
        printf("Failed to create task_helloworld ret:0x%x\n", ret);
        return;
    }

    task2.pfnTaskEntry = (TSK_ENTRY_FUNC)task_openharmony;  // 运行函数入口
    task2.uwStackSize = 2048;                               // 堆栈大小
    task2.pcName = "task_openharmony";                      // 函数注册名称
    task2.usTaskPrio = 25;                                  // 任务的优先级,为 0～63
    ret = LOS_TaskCreate(&thread_id2, &task2);              // 创建任务
    if (ret != LOS_OK)
    {
        printf("Failed to create task_openharmony ret:0x%x\n", ret);
        return;
    }
}
```

3. 创建 BUILD.gn

在 a0_hello_world 文件夹下创建 BUILD.gn 文件。BUILD.gn 负责将 hello_world.c 文件编译成静态库 libtask_helloworld.a。BUILD.gn 的语法为 gn 语法,对 gn 语法感兴趣的读者可以访问 gn 官网阅读相关文档,具体代码如下:

```
static_library("task_helloworld") {
    sources = [
        "hello_world.c",
    ]

    include_dirs = [
        "//utils/native/lite/include",
    ]
}
```

4. 修改 main.c 文件

修改 OpenHarmony 主目录 device/rockchip/rk2206/sdk_liteos/board 文件夹下的 main.c。该文件为 OpenHarmony 操作系统的主函数。在 main.c 文件中添加运行 hello_world.c 文

件的 task_example()函数，具体代码如下：

```
#include "los_tick.h"
#include "los_task.h"
#include "los_config.h"
#include "los_interrupt.h"
#include "los_debug.h"
#include "los_compiler.h"
#include "lz_hardware.h"
#include "config_network.h"

#define MAIN_TAG                        "MAIN"
int DeviceManagerStart();
void IotInit(void);
void task_example();                    // 声明 task_example 函数

/*****************************************************************
Function    : main
Description : Main function entry
Input       : None
Output      : None
Return      : None
*****************************************************************/
LITE_OS_SEC_TEXT_INIT int Main(void)
{
    int ret;
    LZ_HARDWARE_LOGD(MAIN_TAG, "%s: enter ...", __func__);

    HalInit();

    ret = LOS_KernelInit();
    if (ret == LOS_OK) {
        IotInit();
        task_example();                 // 调用 task_example 函数
        OHOS_SystemInit();
        ClkDevInit();
        /* 开启驱动管理服务 */
        //DeviceManagerStart();
        //ExternalTaskConfigNetwork();
        LZ_HARDWARE_LOGD(MAIN_TAG, "%s: LOS_Start ...", __func__);
        LOS_Start();                    // 开启 OpenHarmony 操作系统
    }
    while (1) {
        __asm volatile("wfi");
    }
}
```

5. 修改 Makefile

修改 OpenHarmony 主目录 device/rockchip/rk2206/sdk_liteos 文件夹下的 Makefile 文件。该 Makefile 文件负责将编译好的静态库和可执行文件打包成 bin 文件。将固件库 libtask_helloworld.a 添加到 Makefile 中，让其最终编译成 bin 文件，具体修改如下：

```
################################################################
# LDFLAGS
################################################################
LDSCRIPT = board.ld

boot_LIBS = -lbootstrap -lbroadcast
hardware_LIBS = -lhal_iothardware -lhardware -ltask_helloworld
```

在 Makefile 文件的 hardware_LIBS 变量中添加-ltask_helloworld。

6. 修改 BUILD.gn

修改 OpenHarmony 主目录 device/rockchip/rk2206/sdk_liteos 文件夹下的 BUILD.gn。该 BUILD.gn 文件负责编译各个组件，包括静态库 task_helloworld，具体代码如下：

```
import("//build/lite/config/component/lite_component.gni")
import("//build/lite/config/subsystem/lite_subsystem.gni")

declare_args() {
  enable_hos_vendor_wifiiot_xts = false
}

lite_subsystem("wifiiot_sdk") {
  subsystem_components = [ ":sdk" ]
}

build_ext_component("liteos") {
  exec_path = rebase_path(".", root_build_dir)
  outdir = rebase_path(root_out_dir)
  command = "sh ./build.sh $outdir"
  deps = [
    ":sdk",
    "//build/lite:ohos",
  ]
  if (enable_hos_vendor_wifiiot_xts) {
    deps += [ "//build/lite/config/subsystem/xts:xts" ]
  }
}

lite_component("sdk") {
  features = [ ]
```

```
    deps = [
      "//device/rockchip/rk2206/sdk_liteos/board:board",
      "//device/rockchip/hardware:hardware",
      # xts
      "//device/rockchip/rk2206/adapter/hals/update:hal_update_static",
      # xts
      "//base/update/ota_lite/frameworks/source:hota",
      "../third_party/littlefs:lzlittlefs",
      "//build/lite/config/component/cJSON:cjson_static",
      "../third_party/lwip:rk2206_lwip",
      "//kernel/liteos_m/components/net/lwip-2.1:lwip",
      "//vendor/lockzhiner/rk2206/samples:app",
      "//third_party/paho_mqtt:pahomqtt_static",
      "//vendor/lockzhiner/rk2206/hdf_config:hdf_config",
      "//vendor/lockzhiner/rk2206/hdf_drivers:hdf_drivers",
      "//drivers/adapter/khdf/liteos_m/test/sample_driver:sample_driver",
      "//vendor/lockzhiner/rk2206/samples/a0_hello_world:task_helloworld",
      ]
}
```

注意：倒数第3行内容就是添加task_helloworld。其中，“:”前面为需要编译的路径，“//”表示OpenHarmony源代码主目录；“:”后面为需要编译的选项，即//vendor/lockzhiner/rk2206/samples/a0_hello_world/BUILD.gn文件中的编译选项。

3.2.3 编译程序

终端命令行回到OpenHarmony主目录。

1. 设置主目录路径

输入以下命令：

```
hb set -root
```

2. 设置编译项目

输入以下命令：

```
hb set
```

选择lockzhiner-rk2206项目。具体如图3.2.1所示。

```
[OHOS INFO] hb root path: /svn/mouxian/svn/lockzhiner-rk2206-openharmony3.0lts
OHOS Which product do you need?  (Use arrow keys)

lockzhiner
 › lockzhiner-rk2206
```

图3.2.1 选择项目

3. 编译 OpenHarmony

输入以下命令：

```
hb build -f
```

编译结束后如图 3.2.2 所示。

```
[OHOS INFO] [830/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/sdk.stamp
[OHOS INFO] [831/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/wifiiot_sdk.stamp
[OHOS INFO] [832/834] STAMP obj/build/lite/ohos.stamp
[OHOS INFO] [833/834] ACTION //device/rockchip/rk2206/sdk_liteos:liteos(//build/lite/toolchain:arm-none-eabi-gcc)
[OHOS INFO] [834/834] STAMP obj/device/rockchip/rk2206/sdk_liteos/liteos.stamp
[OHOS INFO] /svn/mouxian/svn/lockzhiner-rk2206-openharmony3.0lts/vendor/lockzhiner/rk2206/fs.yml not found, stop pa
es not need to be packaged, ignore it.
[OHOS INFO] lockzhiner-rk2206 build success
[OHOS INFO] cost time: 0:00:03
```

图 3.2.2　编译结果

4. 烧写程序

烧写程序步骤请参考第 2 章内容，此处不再赘述。

3.2.4　实验结果

设备重新上电后，调试串口显示如下信息表示移植成功。

```
entering kernel init...
[IOT:D]IotInit: start ...
hilog will init.
[MAIN:D]Main: LOS_Start ...
Entering scheduler
[IOT:D]IotProcess: start ...
Hello World success.
Hello OpenHarmony
Hello World
Hello OpenHarmony
Hello World
Hello World
Hello OpenHarmony
Hello World
Hello World
Hello OpenHarmony
Hello World
Hello World
Hello OpenHarmony
Hello World
Hello World
```