

## 前置要求：

操作系统：Ubuntu 22.04

CPU架构：x86\_64

参考Electron鸿蒙化指导文档git拉取Electron源代码

## 一、aki源码下载和编译

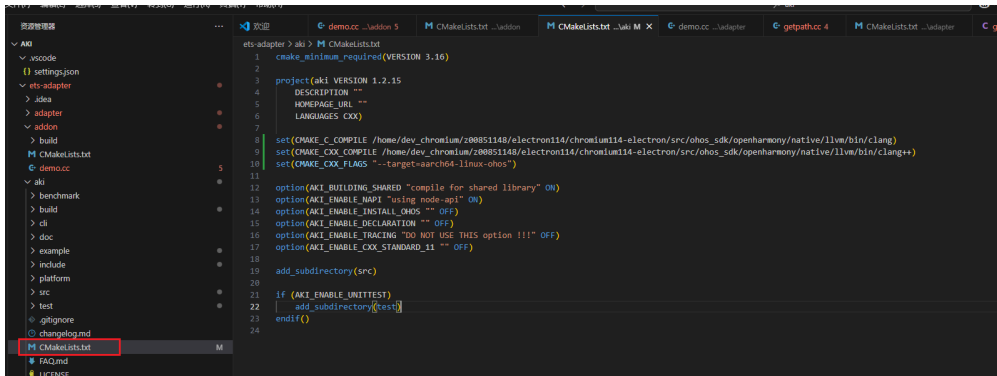
### 1 利用git下拉aki框架代码

git clone https://gitee.com/openharmony-sig/aki.git

### 2 配置cmake

进入aki目录，在cmake中增加以下编译配置（服务器中鸿蒙clang、clang++对应的路径，以下路径仅作参考）：

```
set(CMAKE_C_COMPILE /home/dev_chromium/z00851148/electron114/chromium114-electron/src/ohos_sdk/openharmony/native/llvm/bin/clang)
set(CMAKE_CXX_COMPILE /home/dev_chromium/z00851148/electron114/chromium114-electron/src/ohos_sdk/openharmony/native/llvm/bin/clang)
set(CMAKE_CXX_FLAGS "--target=aarch64-linux-ohos")
```



### 3 配置环境变量（clang、clang++换为实际路径）

```
export CC="/home/dev_chromium/z00851148/electron114/chromium114-electron/src/ohos_sdk/openharmony/native/llvm/bin/clang --target=aarch64-linux-ohos"
export CXX="/home/dev_chromium/z00851148/electron114/chromium114-electron/src/ohos_sdk/openharmony/native/llvm/bin/clang++ --target=aarch64-linux-ohos"
```

### 4 编译

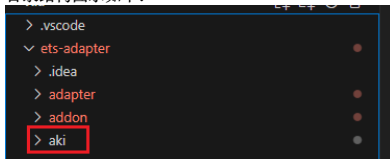
运行结束在build/src路径会生成最终的编译产物 libaki\_jsbind.so

```
cd aki
mkdir build
cd build
cmake ../ -DCMAKE_BUILD_TYPE=Release
make
```

## 二、应用adapter编写

### 1 样例目录结构

目录结构图示如下：



其中，aki文件夹为第一步git下来的aki。

### 2. 仿照adapter下的demo进行方法的注册

```
ets-adapter > adapter > demo.cc > getNativeContext(napi_env, napi_callback_info)
1 #include "third_party/electron_node/src/node.h"
2
3 #include "openharmony/native/sysroot/usr/include/napi/native_api.h"
4 #include "openharmony/native/sysroot/usr/include/js_native_api_types.h"
5 #include "aki/jsbind.h"
6 #include <string.h>
7 #include <stdio.h>
8 #define DECLARE_NAPI_METHOD(name, func) { name, 0, func, 0, 0, 0, napi_default, 0 }
9
10 using v8::FunctionCallbackInfo;
11 using v8::Isolate;
12 using v8::Local;
13 using v8::NewStringType;
14 using v8::Object;
15 using v8::String;
16 using v8::Value;
17
18 static napi_value getNativeContext(napi_env env, napi_callback_info info) {}
19
20 napi_value exports;
21 if (napi_create_object(env, &exports) != napi_ok) {
22     napi_throw_error(env, nullptr, "napi_create_object failed");
23     return exports;
24 }
25
26 aki::JSBind::BindSymbols(env, exports);
27 return exports;
28
29
30 static napi_value Initialize(napi_env env, napi_value exports) {
31     if (env == nullptr || exports == nullptr) {
32         return exports;
33     }
34     napi_property_descriptor desc[] = {
35         DECLARE_NAPI_METHOD("getNativeContext", getNativeContext)
36     };
37     if (napi_define_properties(env, exports, sizeof(desc) / sizeof(desc[0]),
38         desc) != napi_ok) {
39         return exports;
40     }
41     return exports;
42 }
43
44
45 static napi_module adaptertestModule = {
46     .nm_version = 1,
47     .nm_flags = 0,
48     .nm_filename = nullptr,
49     .nm_register_func = Initialize,
50     .nm_module_name = "adaptertest",
51     .nm_priv = ((void*)nullptr),
52     .reserved = {nullptr},
53 };
54
55 extern "C" __attribute__((constructor)) void RegisterAdaptertestModule(void) {
56     napi_module_register(&adaptertestModule);
57 }
```

### 3. 仿照getPatch.cc进行方法的绑定

此处提供同步和异步两种方式，供应用自己选择：

```
ets-adapter > adapter > getpath.cc > PathAdapter > GetDir(std::string R)
1 #include "getpath.h"
2
3 #include <functional>
4 #include "aki/jsbind.h"
5
6 namespace PathAdapter {
7     std::string GetDir(std::string& getDirFuncName) {}
8
9     // 同步方式
10     if (auto getDirFunc = aki::JSBind::GetJSFunction(getDirFuncName)) {
11         auto path = getDirFunc->Invoke<std::string>();
12         return path;
13     }
14     return "";
15
16     // 异步方式
17     // std::promise<std::string> promise;
18     // std::function<void(std::string)> callback = [&promise](std::string path) {
19     //     promise.set_value(path);
20     // };
21
22     // if (auto jsFunc = aki::JSBind::GetJSFunction(getDirFuncName)) {
23     //     jsFunc->Invoke<void>(callback);
24     // } else {
25     //     return "";
26     // }
27
28     // auto future = promise.get_future();
29     // auto status = future.wait_for(std::chrono::seconds(5));
30     // if (status == std::future_status::timeout) {
31     //     return "timeout";
32     // }
33     // return future.get();
34 }
35 }
```

### 4. CMakeLists编写

```
# Project Name
SET(TARGETFILENAME "adaptertest")
PROJECT(${TARGETFILENAME})

# CMake minimum version requirement setting
cmake_minimum_required(VERSION 3.8)

# set electron path
set(ELE_PATH /home/dev_chromium/z00851148/electron114/chromium114-electron)

# ohos
set(CMAKE_C_COMPILER ${ELE_PATH}/src/ohos_sdk/openharmony/native/llvm/bin/clang)
set(CMAKE_CXX_COMPILER ${ELE_PATH}/src/ohos_sdk/openharmony/native/llvm/bin/clang++)
set(CMAKE_CXX_FLAGS "--target=aarch64-linux-ohos")

# 设置包含目录
# ohos
include_directories(${ELE_PATH}/src/)
include_directories(${ELE_PATH}/src/third_party/electron_node/deps/v8/include/)
include_directories(${ELE_PATH}/src/ohos_sdk/)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/../aki/include)

message(${CMAKE_CURRENT_SOURCE_DIR})

set(src demo.cc getpath.cc)
set(headers getpath.h)

# 生成可执行文件
#add_executable(${TARGETFILENAME} demo.cpp)

# 生成静态库
#ADD_LIBRARY(${TARGETFILENAME} STATIC demo.cpp)

# 生成动态库或共享库
```

```

ADD_LIBRARY (${TARGETFILENAME} SHARED
    ${src})
)
target_compile_features(${TARGETFILENAME} PUBLIC cxx_std_17)
target_compile_definitions(${TARGETFILENAME} PUBLIC JSBIND_USING_NAPI=1)
target_compile_definitions(${TARGETFILENAME} PUBLIC AKI_BUILDING_SHARED=1)

# 设置链接库文件-ohos
target_link_libraries(${TARGETFILENAME} PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/../aki/build/src/libaki_jsbind.so)
target_link_libraries(${TARGETFILENAME} PUBLIC ${ELE_PATH}/src/out/musl_64/libelectron.so)

# 修改后缀名为 .node
# SET_TARGET_PROPERTIES(${TARGETFILENAME} PROPERTIES SUFFIX ".node")

```

## 5. 获取编译产物

执行命令，可获取编译产物libadaptestest.so

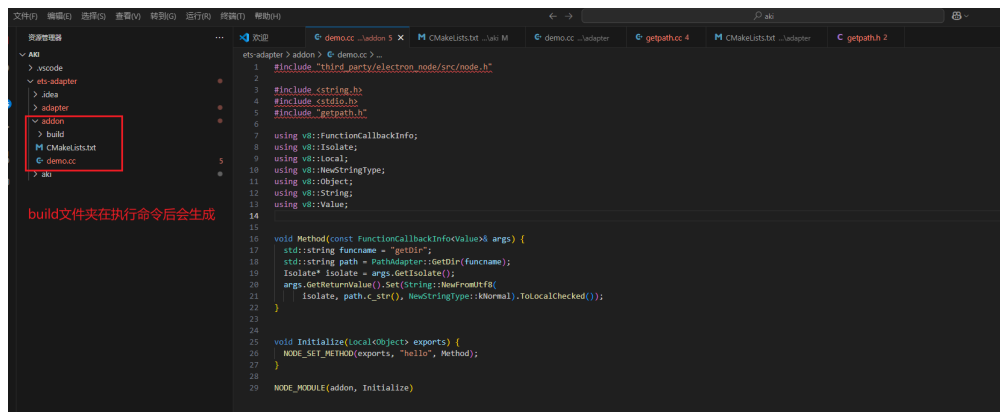
```

mkdir build
cd build
cmake ../
make

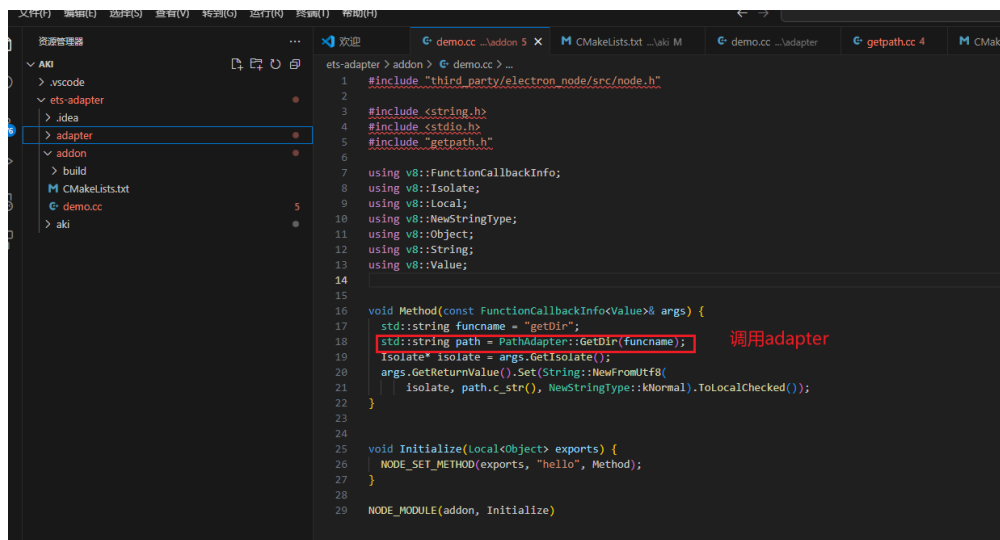
```

# 三、应用addon编写

## 1 样例目录结构



## 2 调用adapter



## 3 CMakeLists编写

```

# Project Name
SET(TARGETFILENAME "addon")
PROJECT(${TARGETFILENAME})

# CMake minimum version requirement setting
cmake_minimum_required(VERSION 3.8)

# set electron path
set(ELE_PATH /home/dev_chromium/z00851148/electron114/chromium114-electron)

# ohos
set(CMAKE_C_COMPILER ${ELE_PATH}/src/ohos_sdk/openharmony/native/llvm/bin/clang)
set(CMAKE_CXX_COMPILER ${ELE_PATH}/src/ohos_sdk/openharmony/native/llvm/bin/clang++)
set(CMAKE_CXX_FLAGS "--target=aarch64-linux-ohos")

# 设置包含目录

# ohos

include_directories(${ELE_PATH}/src/)
include_directories(${ELE_PATH}/src/third_party/electron_node/deps/v8/include/)

```

```

include_directories(${ELE_PATH}/src/ohos_sdk/)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/../adapter/)

message(${CMAKE_CURRENT_SOURCE_DIR})

set(src demo.cc)

# 生成可执行文件

#add_executable(${TARGETFILENAME} demo.cpp)

# 生成静态库

#ADD_LIBRARY(${TARGETFILENAME} STATIC demo.cpp)

# 生成动态库或共享库
ADD_LIBRARY (${TARGETFILENAME} SHARED
    ${src}
)

# 设置链接库文件-ohos
target_link_libraries(${TARGETFILENAME} PUBLIC ${ELE_PATH}/src/out/musl_64/libelectron.so)
target_link_libraries(${TARGETFILENAME} PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/../adapter/build/libadaptestest.so)
target_compile_definitions(${TARGETFILENAME} PUBLIC NODE_MODULE_VERSION=116)

# 修改后缀名为 .node
SET_TARGET_PROPERTIES(${TARGETFILENAME} PROPERTIES SUFFIX ".node")

```

## 4 获取编译产物

执行下述命令后，可获取编译产物addon.node

```

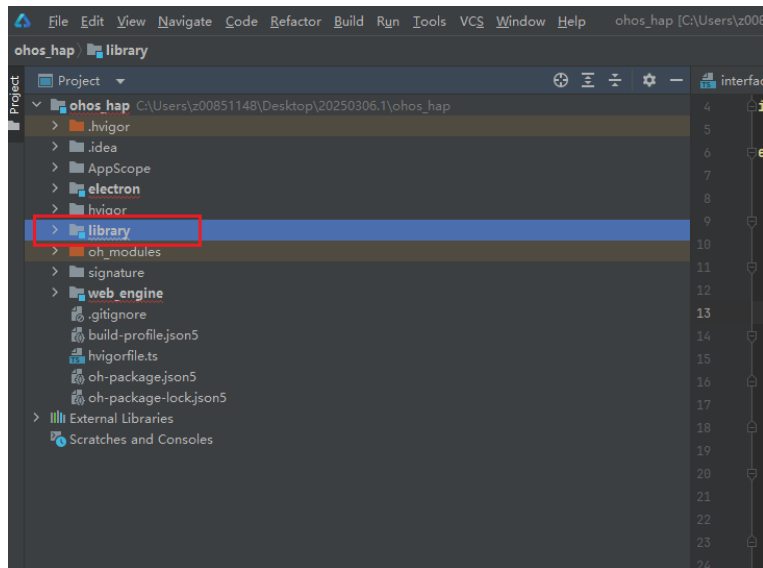
mkdir build
cd build
cmake ../
make

```

# 四、ets调用及har包构建

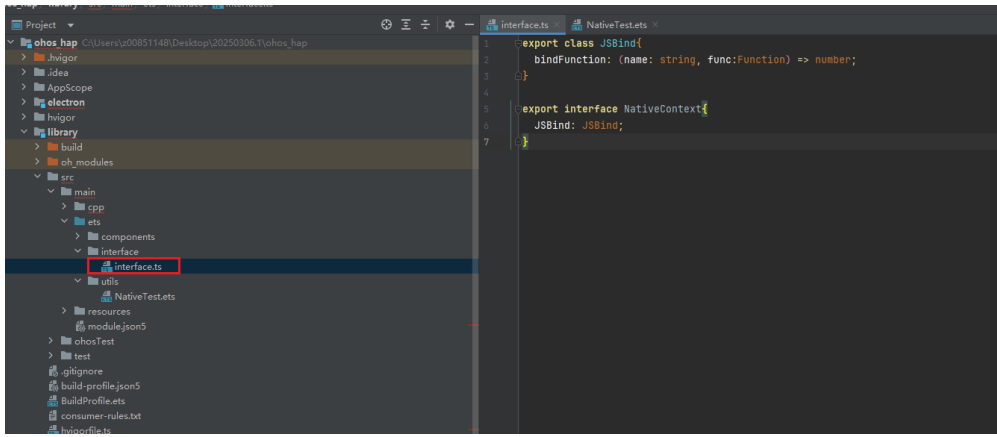
## 1 创建工程

deveco中点击File ==> New ==> Moudule ==> Static Library，创建一个Library



## 2 将adapter中的注册进行声明和初始化

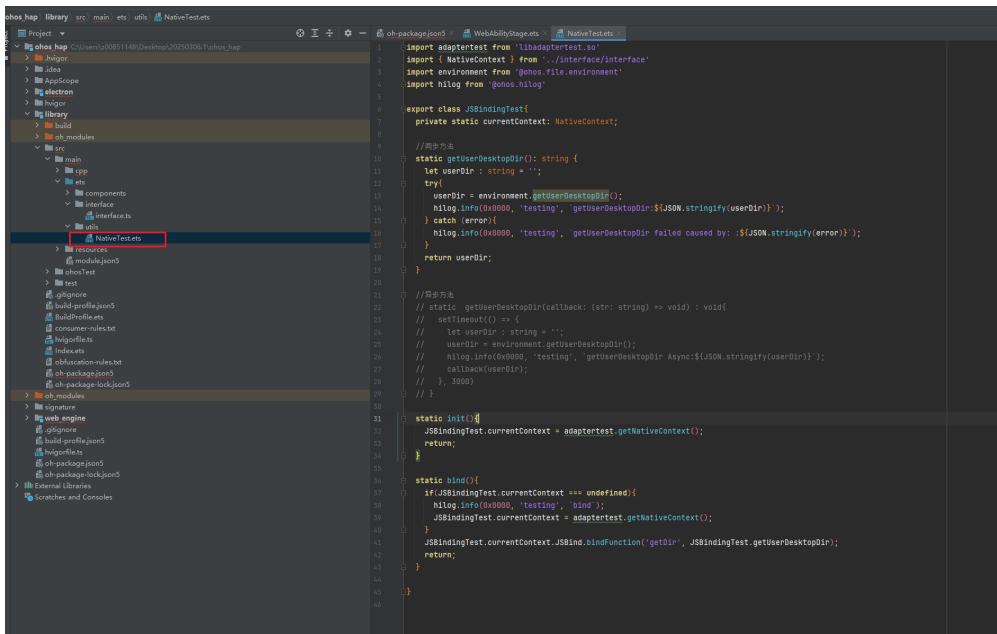
### 2.1 adapter中的注册进行声明



```
export class JSBind{
  bindFunction: (name: string, func:Function) => number;
}
```

```
export interface NativeContext{
  JSBind: JSBind;
}
```

## 2.2 定义初始化和bind函数



```
import adapterTest from 'libadaptestest.so'
import { NativeContext } from '../interface/interface'
import environment from '@ohos.file.environment'
import hilog from '@ohos.hilog'
```

```
export class JSBindingTest{
  private static currentContext: NativeContext;

  //同步方法
  static getUserDesktopDir(): string {
    let userDir : string = '';
    try{
      userDir = environment.getUserDesktopDir();
      hilog.info(0x0000, 'testing', `getUserDesktopDir:${JSON.stringify(userDir)}`);
    } catch (error){
      hilog.info(0x0000, 'testing', `getUserDesktopDir failed caused by: :${JSON.stringify(error)}`);
    }
    return userDir;
  }

  //异步方法
  // static getUserDesktopDir(callback: (str: string) => void) : void{
  //   setTimeout(() => {
  //     let userDir : string = '';
  //     userDir = environment.getUserDesktopDir();
  //     hilog.info(0x0000, 'testing', `getUserDesktopDir Async:${JSON.stringify(userDir)}`);
  //     callback(userDir);
  //   }, 3000)
  // }

  static init(){
    JSBindingTest.currentContext = adapterTest.getNativeContext();
    return;
  }

  static bind(){
    if(JSBindingTest.currentContext === undefined){
      hilog.info(0x0000, 'testing', `bind`);
      JSBindingTest.currentContext = adapterTest.getNativeContext();
    }
  }
}
```

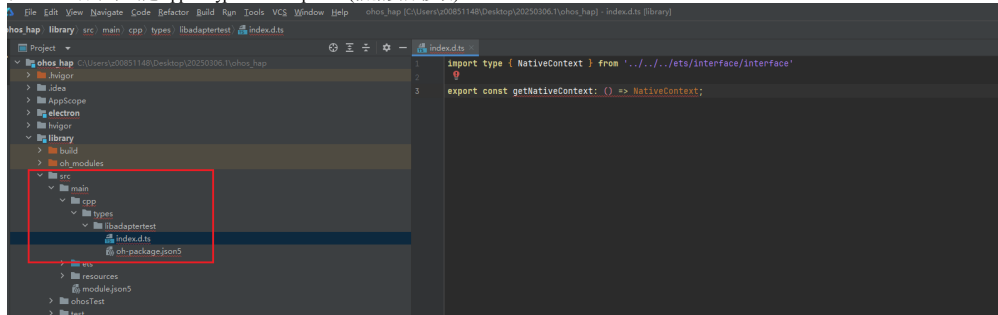
```

    }
    JSBindingTest.currentContext.JSBind.bindFunction('getDir', JSBindingTest.getUserDesktopDir);
    return;
  }
}

```

### 2.2.3 增加共享包导出声明入口

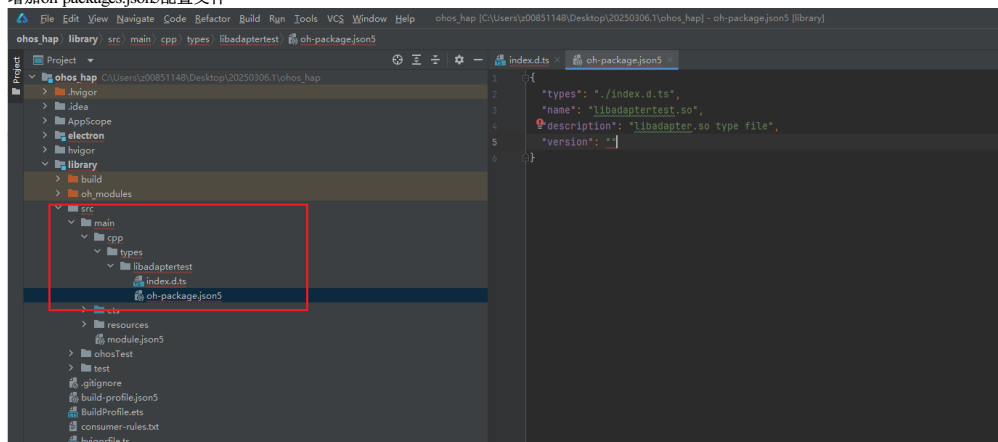
在src/main目录下创建 cpp -> types -> libadaptestest(根据实际情况修改)



```
import type { NativeContext } from '../../ets/interface/interface'
```

```
export const getNativeContext: () => NativeContext;
```

增加oh-packages.json5配置文件

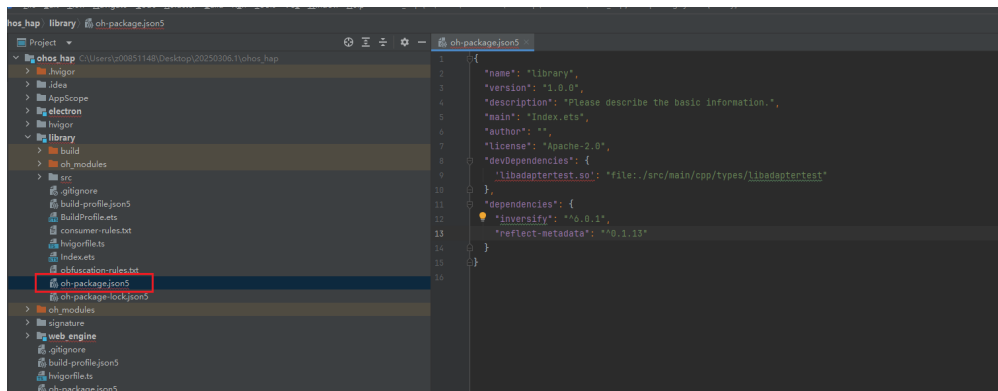


```

{
  "types": "../index.d.ts",
  "name": "libadaptestest.so",
  "description": "libadapter.so type file",
  "version": ""
}

```

### 2.2.4 修改library根目录下的oh-packages.json5配置文件，增加引用依赖

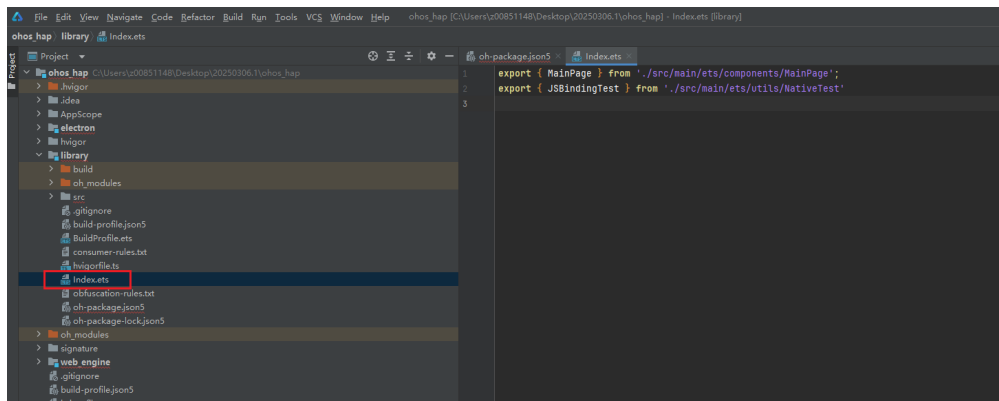


```

{
  "name": "library",
  "version": "1.0.0",
  "description": "Please describe the basic information.",
  "main": "Index.ets",
  "author": "",
  "license": "Apache-2.0",
  "devDependencies": {
    "libadaptestest.so": "file:../src/main/cpp/types/libadaptestest"
  },
  "dependencies": {
    "inversify": "^6.0.1",
    "reflect-metadata": "^0.1.13"
  }
}

```

### 2.2.5 修改library根目录下的Index.ets配置文件，增加导出方法

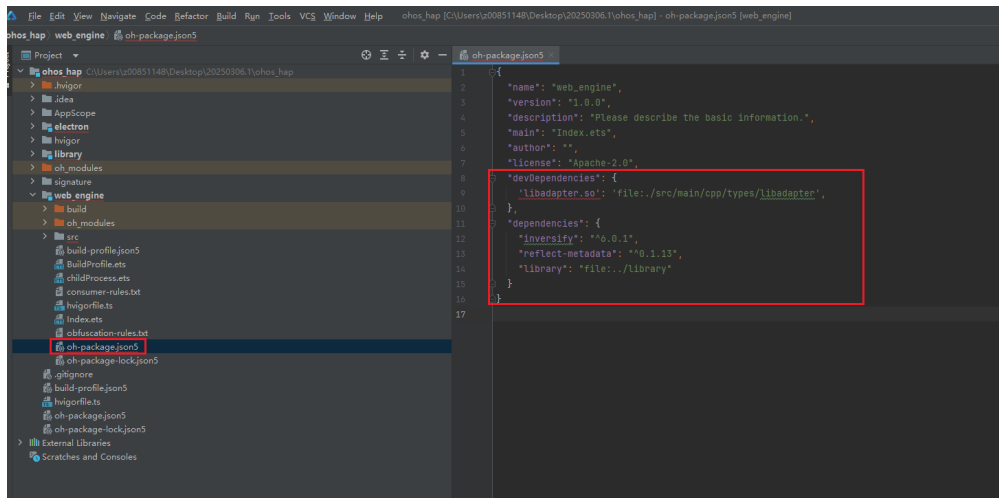


```
export { MainPage } from './src/main/ets/components/MainPage';
export { JSBindingTest } from './src/main/ets/Utils/NativeTest';
```

## 2.2.6 构建har包

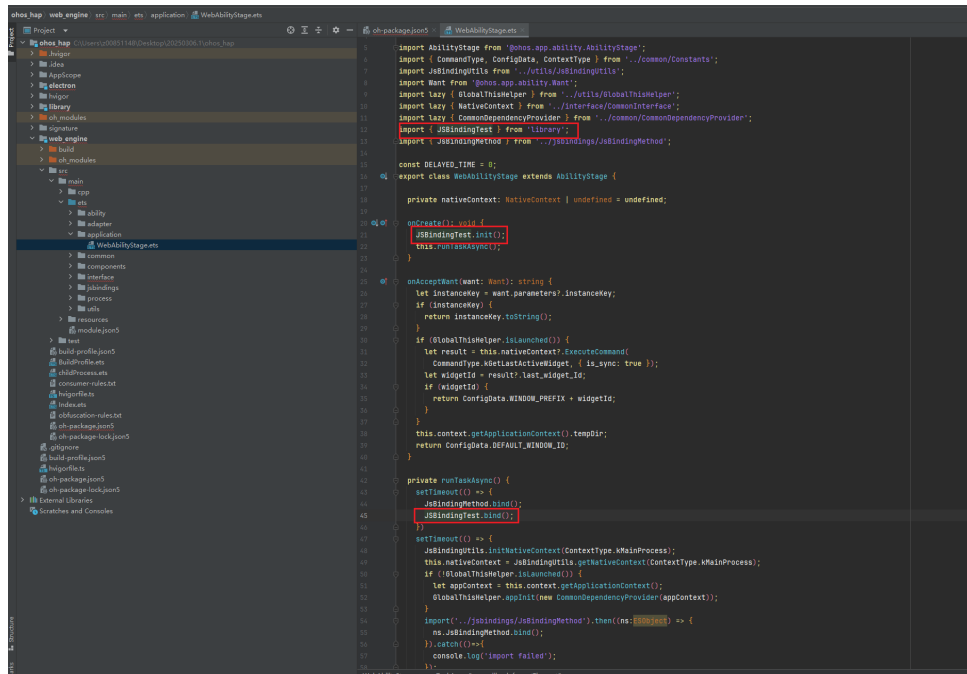
参考文档: [引用共享包-开发及引用共享包-应用/元服务开发-DevEco Studio - 华为HarmonyOS开发者](#)

1. 在web\_engine模块中引用此HAR包 webengine/oh-packages.json5 中增加引用配置



```
{
  "name": "web_engine",
  "version": "1.0.0",
  "description": "Please describe the basic information.",
  "main": "Index.ets",
  "author": "",
  "license": "Apache-2.0",
  "devDependencies": {
    "libadapter.so": "file:./src/main/cpp/types/libadapter",
  },
  "dependencies": {
    "inversify": "^6.0.1",
    "reflect-metadata": "^0.1.13",
    "library": "file:../library"
  }
}
```

2. 在WebAbilityStage.ets文件中初始化、绑定



// Copyright (c) 2024 Huawei Device Co., Ltd. All rights reserved.  
// Use of this source code is governed by a BSD-style license that can be  
// found in the LICENSE file.

```
import AbilityStage from '@ohos.app.ability.AbilityStage';
import { CommandType, ConfigData, ContextType } from '../common/Constants';
import JsBindingUtils from '../utils/JsBindingUtils';
import Want from '@ohos.app.ability.Want';
import lazy { GlobalThisHelper } from '../utils/GlobalThisHelper';
import lazy { NativeContext } from '../interface/CommonInterface';
import lazy { CommonDependencyProvider } from '../common/CommonDependencyProvider';
import { JSBindingTest } from 'library';
import { JsBindingMethod } from '../jsbindings/JsBindingMethod';

const DELAYED_TIME = 0;
export class WebAbilityStage extends AbilityStage {

  private nativeContext: NativeContext | undefined = undefined;

  onCreate(): void {
    JSBindingTest.init();
    this.runTaskAsync();
  }

  onAcceptWant(want: Want): string {
    let instanceKey = want.parameters?.instanceKey;
    if (instanceKey) {
      return instanceKey.toString();
    }
    if (GlobalThisHelper.isLaunched()) {
      let result = this.nativeContext?.ExecuteCommand(
        CommandType.kGetLastActiveWidget, { is_sync: true });
      let widgetId = result?.last_widget_Id;
      if (widgetId) {
        return ConfigData.WINDOW_PREFIX + widgetId;
      }
    }
    this.context.getApplicationContext().tempDir;
    return ConfigData.DEFAULT_WINDOW_ID;
  }

  private runTaskAsync() {
    setTimeout(() => {
      JsBindingMethod.bind();
      JSBindingTest.bind();
    })
    setTimeout(() => {
      JsBindingUtils.initNativeContext(ContextType.kMainProcess);
      this.nativeContext = JsBindingUtils.getNativeContext(ContextType.kMainProcess);
      if (!GlobalThisHelper.isLaunched()) {
        let appContext = this.context.getApplicationContext();
        GlobalThisHelper.appInit(new CommonDependencyProvider(appContext));
      }
      import('../jsbindings/JsBindingMethod').then((ns: typeof JsBindingMethod) => {
        ns.JsBindingMethod.bind();
      }).catch(() => {
        console.log('import failed');
      });
    })
  }
}
```



```

GlobalThisHelper.appInit(new CommonDependencyProvider(appContext));
}
import('../jsbindings/JsBindingMethod').then((ns:ESObject) => {
    ns.JsBindingMethod.bind();
}).catch(()=>{
    console.log('import failed');
});
}), DELAYED_TIME)
}
}

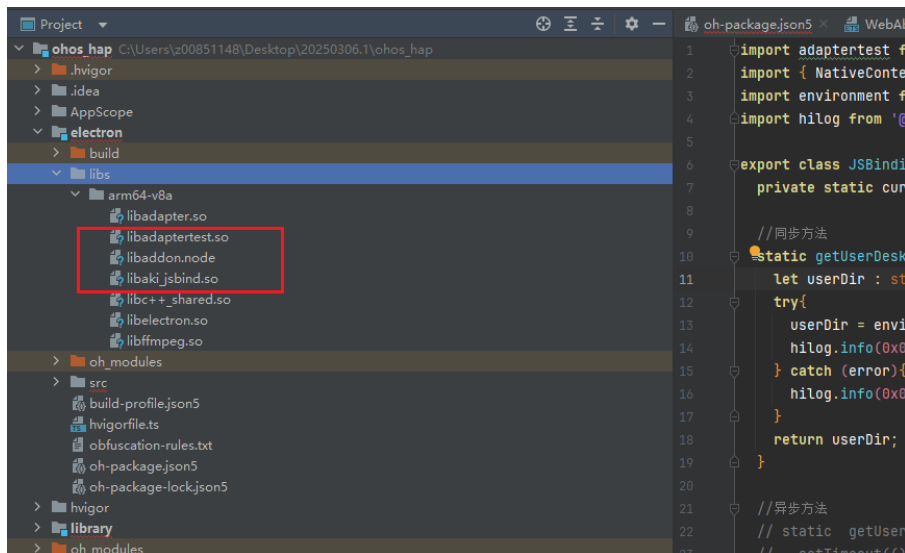
```

## 2.2.7 附加：部分目录结构介绍

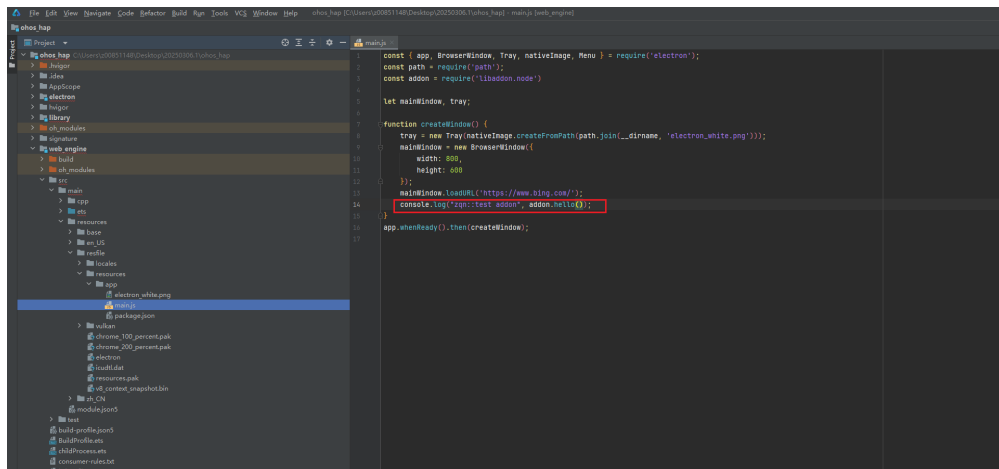
- **libs**: 用于存放.so文件。
- **src > main > cpp > types**: 用于存放 C++ API 描述文件，子目录按照 so 维度进行划分。
- **src > main > cpp > types > liblibrary > Index.d.ts**: 描述 C++ 接口的方法名、入参、返回参数等信息。
- **src > main > cpp > types > liblibrary > oh-package.json5**: 描述 so 三方包声明文件入口和 so 包名信息。
- **src > main > cpp > CMakeLists.txt**: CMake 配置文件，提供 CMake 构建脚本。
- **src > main > cpp > napi\_init.cpp**: 共享包 C++ 代码源文件。
- **Index.ets**: 共享包导出声明的入口。

## 五、最终调用

### 1 HAP包目录结构



### 2 JS代码调用



### 3 运行结果

