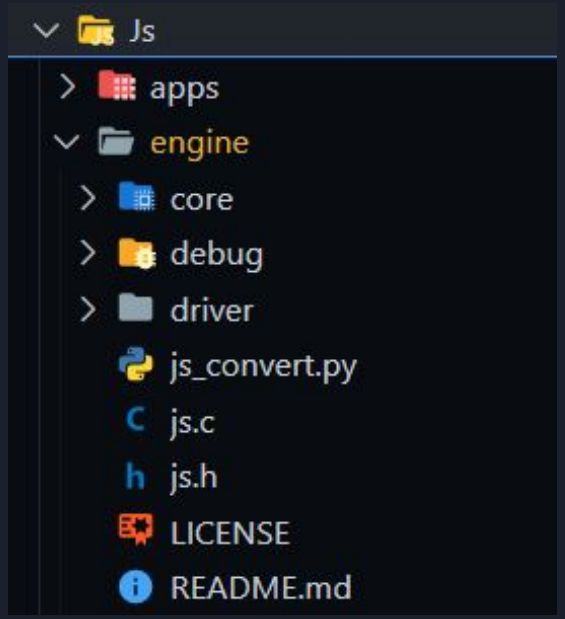
TencentOS-tiny 项目1 JS engine 移植

田祥安

项目简介

t_js 是一个用C 语言实现的JS引擎，为了更轻量化，只实现了ES6功能实用的一部分，可以在微处理器上运行。它是基于开源项目 Elk 二次开发的，所以继承了 Elk 的GNU General Public License v2.0 协议。t_js 在 Elk 的基础上进行了大规模重构，使得代码的模块更加清晰，利于维护。同时也修复了原代码中的若干缺陷。目前用户可以将写好的.js 文件通过 Python 脚本转换成.h文件中的字符串来运行，后续将支持从SD 卡读取和网络传输 .js 文件来运行。



功能支持

1. 运算符 : 除了 !=, ==, (使用!==, ===), a ? b : c, a[b] 的所有JS标准运算符
2. 查询变量类型 : typeof(a)
3. 循环 : while(...) { ... }
4. 条件 : if (...) { ... } else { ... }
5. 简单的变量类型 : let a = 1, b, c = 2.3, d = 'a', e = null, f = true, g = false，暂不支持 var，const
6. 函数申明 : let f = function(a, b) { return a + b; }，暂不支持 =>
7. 对象 : let obj = {f: function(x) {return x * 2}}; obj.f(2);
8. 导入 C 函数
9. 32 位和 64 位支持 (float, double)

原理简介

1. 申明固定大小的内存作为 JS engine 的运行内存
2. JS 脚本以字符串的形式进行实时解析：根据关键字、运算符、逻辑符号、数值进行运算，并将对象、属性、和字符串等存入内存中
3. 运行后将结果存入内存，并返回结果的地址
4. 垃圾回收机制会在每个scope 运行完和整个脚本执行完后进行。
5. 导入 C 函数时，会在内存尾部保存导入函数的指针

C 函数导入

t_js支持通过语言交互接口(FFI)来调用C函数，可以通过js_import()来导入C函数。

函数签名指示符：

b: 布尔类型，d: double 类型，

i: 整型，包括char, short, int, long

s: C字符串类型，j：jsval_t类型

m：现有的JS对象，p：指针类型

v：只适用于返回值，返回值为void

需导入的函数

```
void toggle_led()
{
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_5);
}
```

js_import导入

```
js_import(js, (uintptr_t)toggle_led, "v");
```

C 函数导入

全局导入

```
// 获取全局对象
jsval_t glb = js_glob(js);
// 导入C函数并返回导入后的在JS内存的地址
jsval_t func_addr = js_import(js, (uintptr_t)toggle_led, "v");
// 将函数导入global namespace
js_set(js, glb, "toggle_led", func_addr);
```

作为对象的方法

```
jsval_t printer_obj = js_mkobj(js);
js_set(js, js_glob(js), "printer", printer_obj);

js_set(js, printer_obj, "print", js_import(js, (uintptr_t)print, "v"));
```

使用场景

1. 用户直接在代码中输入 JavaScript 脚本
2. 用户在本地好 .js 文件后，通过转化为 .h 头文件运行
3. 用户远程编写好程序后，通过网络传输到设备运行

本地 .js 文件转化为 .h 文件

loop.js

```
let a = 0;
while (a < 100){
    delay(100);
    print(a);
    a++;
}
```

```
#pragma once

const char js_app_loop[] = "let a = 0; while (a < 100) {     delay(200);     print(a);     a++; }";
```

```
void task_js_exec(void *pdata)
{
    printf("js_exec\n");
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    k_err_t err = tos_mutex_pend(&mutex_js_engine);
    if (err == K_ERR_NONE)
    {
        JS *js = js_create_static();
        js_driver_init(js);
        js_eval(js, js_app_loop, ~0);
        tos_mutex_post(&mutex_js_engine);
    }
    tos_sleep_ms(500);
}
```

演示：运算

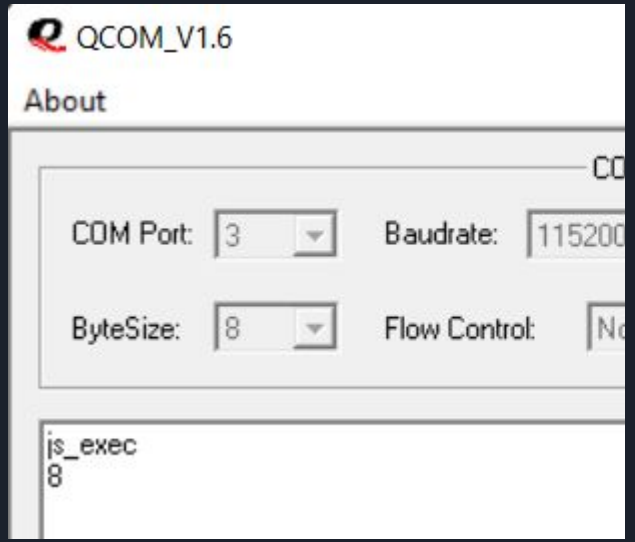
JS 文件

```
let a = 1 + 2 * 3.5;
print(a);
```

Task

```
void task_js_exec(void *pdata)
{
    printf("js_exec\n");
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    k_err_t err = tos_mutex_pend(&mutex_js_engine);
    if (err == K_ERR_NONE)
    {
        JS *js = js_create_static();
        js_driver_init(js);
        js_eval(js, js_app_calculation, ~0);
        tos_mutex_post(&mutex_js_engine);
    }
    tos_sleep_ms(500);
}
```

结果

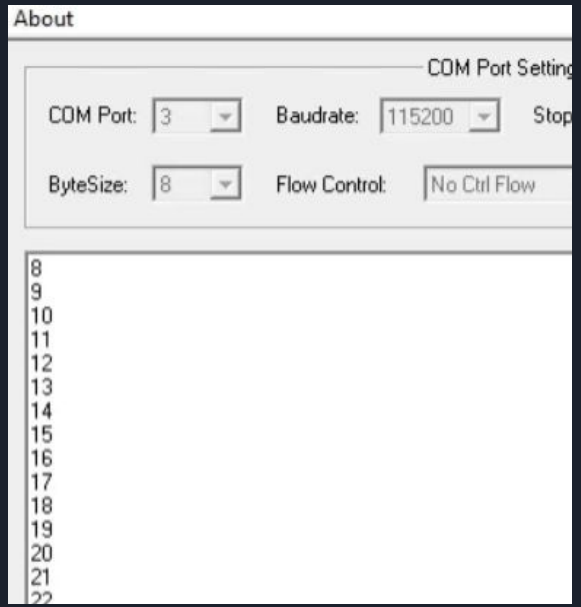


演示：循环

JS 文件

```
let a = 0;
while (a < 100) {
    delay(200);
    print(a);
    a++;
}
```

结果



演示：

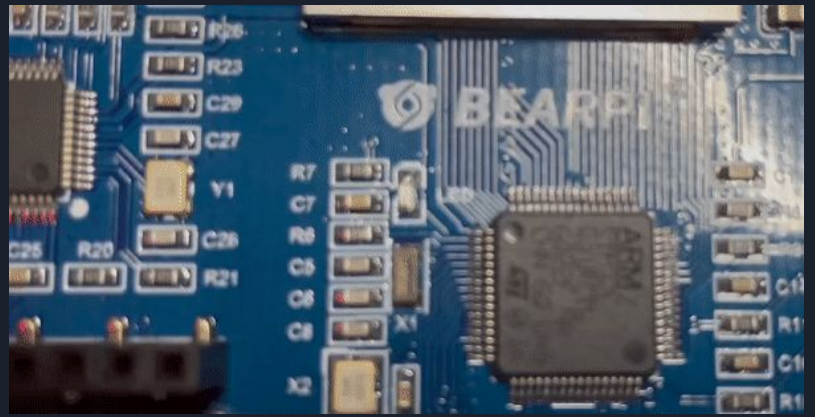
JS 文件

```
let time = 2000;

print(time);

while (true) {
    toggle_led();
    delay(time);
}
```

结果



视频演示:远程控制