PikaScript 软件使用说明书

目录

一、PikaScript 综述：

1.简介

pikascript 可以为 mcu 裸机 c 开发提供面向对象的脚本绑定。

支持裸机运行，可运行于内存 20Kb 以上的 mcu 中，如 stm32f103c8t6，esp32。

官方支持 api 源码生成器

https://github.com/mimilib/pikascript-compiler-rust

支持跨平台，可在 linux 环境开发、测试内核。

开箱即用，零配置，仅使用 C 标准库，几乎不使用宏，几乎不使用全局变量。

完整的单元测试。

堆空间零占用，栈空间少量占用，内存信息可观测。

尽可能的结构清晰（尽我所能）。

2.stm32 教程视频

https://www.bilibili.com/video/BV1mg411L72e

3.内核源码：

对象支持:https://github.com/mimilib/mimilib/tree/master/mimiObject

数据结构:https://github.com/mimilib/mimilib/tree/master/mimiData

4.用法示例：

```c
#include "sysObj.h"

/*
    被绑定的方法
    self 是对象指针，指向执行方法的对象
    args 是参数列表，用于传入传出参数
    （所有被绑定的方法均使用此形参）
*/
void add(MimiObj *self, Args *args)
{
    /*
        参数传递
        从参数列表中取出输入参数val1和val2
    */
    int val1 = args_getInt(args, "val1");
    int val2 = args_getInt(args, "val2");

    /* 实现方法的功能 */
    int res = val1 + val2;

    /* 将返回值传回参数列表 */
    method_returnInt(args, res);
}

/*
    定义测试类的构造器，一个构造器对应一个类
    通过构造器即可新建对象
    args是构造器的初始化参数列表
    MimiObj*是新建对象的指针
    （所有构造器均使用此形参）
*/
```

```c
MimiObj *New_MimiObj_test(Args *args)
{
    /*
        继承sys类
        只需要直接调用父类的构造器即可
    */
    MimiObj *self = New_MimiObj_sys(args);

    /*
        为test类绑定一个方法（支持重载）
        1.入口参数self：对象指针，指向当前对象
        2.传入的第二参数是被绑定方法的接口定义
        （此处使用typescript语法，简单的修改即可支持python格式）
        3.传入的第三个参数是被绑定方法的函数指针
    */
    class_defineMethod(self, "add(val1:int, val2:int):int", add);


    /* 返回对象 */
    return self;
}
```

```c
void main()
{
    /*
        新建根对象，对象名为“sys”
        传入对象名和构造器的函数指针
    */
    MimiObj *sys = newRootObj("sys", New_MimiObj_sys);

    /*
        新建test对象
        test对象作为子对象挂载在sys对象下（对象树）
    */
    obj_newObj(sys, "test", New_MimiObj_test);

    /*
        运行单行脚本。
        因为test对象挂在在sys对象下，
        因此可以通过test.add调用test对象的方法
        运行后会动态新建res属性，该属性属于sys对象
    */
    obj_run(sys, "res = test.add(val1 = 1, val2 = 2)");
    /*
        (也支持 "res = test.add(1, 2)"的调用方式)
    */

    /* 从sys对象中取出属性值res */
    int res = obj_getInt(sys, "res");

    /*
        析构对象
        所有挂载在sys对象下的子对象都会被自动析构
        本例中挂载了test对象，因此在析构sys对象前，
        test对象会被自动析构
    */
    obj_deinit(sys);

    /* 打印返回值 res = 3*/
    printf("%d\r\n", res);
}
```

二、PikaScript 原理解析：

1、什么是 PikaScript

在 IOT、智能终端等嵌入式应用场景中，脚本开发是一个方便快捷的解决方案。

说到嵌入式使用脚本语言开发，可能首先想到的就是 micropython，micropython 可以让工程师使用脚本语言 python 进行 mcu 开发，极大地降低了开发门槛。

但是使用 micropython 开发能够直接使用的开发板并不多，为没有现成 micropython 固件的 mcu 移植 micropython 显然也是一件工程浩大且门槛很高的工作。

而且 python 的运行效率较低，在资源紧缺的 mcu 中显得尤为明显，使用 python 开发也难以充分利用 mcu 的中断、dmp 等硬件特性。在高实时性的信号处理、数据采集、实时控制等应用中，python 难以成为真正落地于生产环境。

就目前而言，在 mcu 开发中，占 80%左右的开发仍然是使用 c 语言，c++也仅占不到 20%。但是无疑脚本语言的便利性是非常明显的。服务器端的开发者往往熟悉 python 和 JavaScript 等支持面向对象的脚本语言，如果能够直接用脚本语言调用 mcu 的功能，将明显降低开发难度。

那么，如果使用 c 语言进行 mcu 嵌入式开发，又向上位机或者服务器提供面向对象的脚本语言调用接口，不就可以兼顾 mcu 运行效率和开发效率了吗？

本文介绍的 mimisciprt 库正是可以起到这样的作用。

mimiscrpit 库可以为 c 语言开发的 mcu 工程提供面向对象的脚本语言调用接口。PikaScript 有以下几个特点：

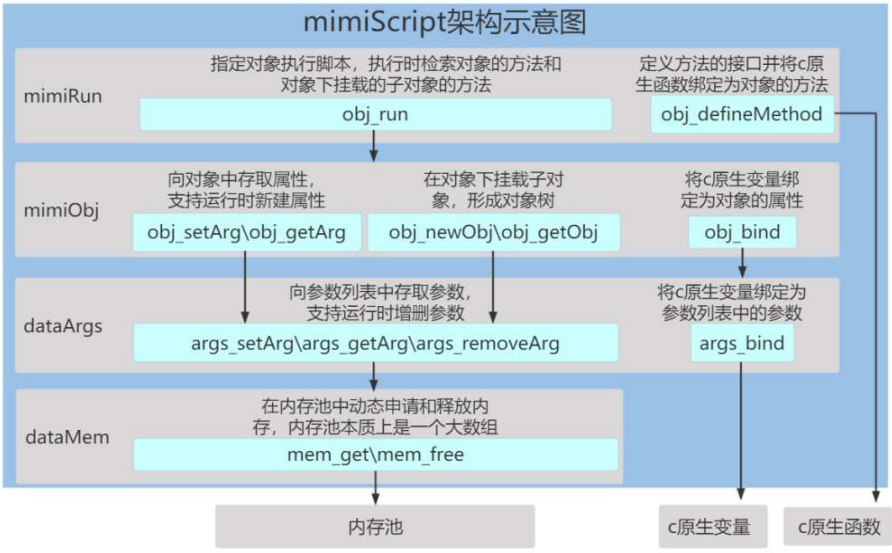
支持裸机运行，可运行于内存 40Kb 以上的 mcu 中，如 stm32f103，esp32。

支持跨平台，可运行于 linux 环境。

代码可读性强，仅使用 C 标准库，尽可能的结构清晰（尽我所能），几乎不使用宏。

2、PikaScript 的原理解析

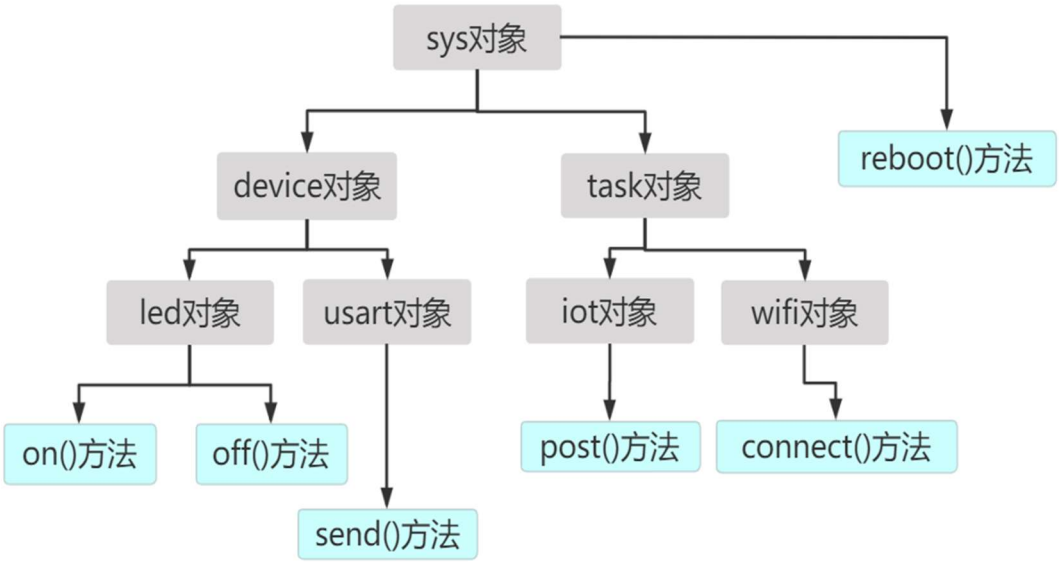
PikaScript 的架构示意图如下图所示。我们从上往下逐层分析。



1.mimiRun 脚本运行层

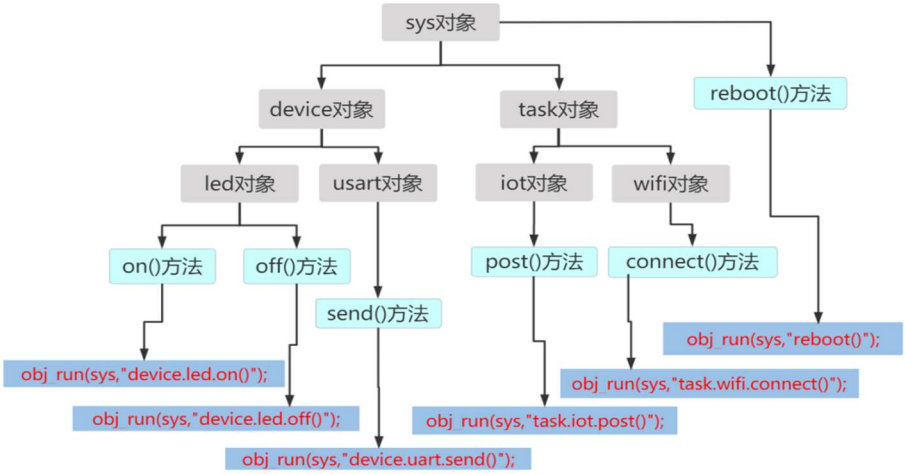
mimiRun 脚本运行层是 PikaScript 的最上层调用接口，只需要调用 obj_run 就可以实现脚本运行。在调用 obj_run 时，需要指定一个对象，脚本在运行时，会检索这个对象的方法和这个对象的子对象的方法。

下图所示的是一个常见的嵌入式开发中的对象结构，sys 是最上层的对象，sys 对象有 reboot() 方法，sys 对象下挂载了 device 子对象和 task 子对象，这两个对象下面又挂载了子对象，每个子对象有自己的方法。



这个时候，我们只需要在 obj_run 中传入最上面的 sys 对象的指针，就可以用如下图所示的

方法调用所有对象的所有方法。其中，reboot()方法直接属于 sys 对象，因此使用直接运行 obj_run(sys,"reboot()")就可以调用，而 led 对象则通过 obj_run(sys,"device.led.on()")进行调用。



在实际开发中，我们可以让 mcu 将串口接收到的数据直接作为脚本运行。

例如：

```
1  obj_run(sys, uartReciveBuff);
```

其中 uartReciveBuff 是串口接收到的数据。

这时向 mcu 的串口中发送"device.led.on()"，就可以点亮 led 灯。

2.mimiObj 对象支持层

如上一节所述，我们已经知道如何使用 PikaScript 在已有的对象结构中执行脚本。那么下一个问题就是，如何构造对象，又如何为对象定义属性和方法呢？

（1）构造器函数

PikaScript 通过一个构造器函数来构造对象。一个构造器函数对应一个 PikaScript 中的类。如下所示的就是一个 LED 的构造器函数。在 PikaScript 中，所有的构造器函数都使用相同的入口参数和返回参数。

入口参数 args 是一个参数列表，args 内部基于链表，可以传入任意个数、任意类型的参数，此处 args 是构造器的初始化参数，在进行含参构造时将会用到。

构造器函数的返回值是一个 MimiObj 对象。

```c
MimiObj * New_LED(Args *args)
{
    // 继承自MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    // 为对象定义属性
    obj_setInt(self, "isOn", 0);
    // 为LED1对象绑定on()方法
    obj_defineMethod(self, "on()", onFun);
    return self;
}
```

构造器的第一行代码用于类继承，LED 类继承自 mimiobj 基类，mimiobj 基类是所有类的源头。

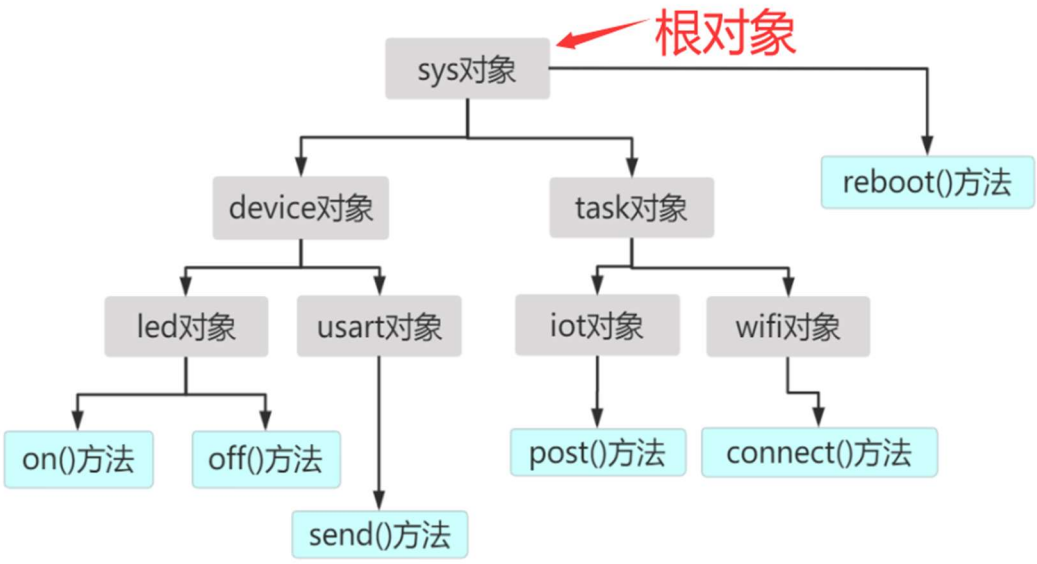
obj_setInt 为 LED 类定义一个属性，属性名为"isOn"，初始值为 0。

obj_defineMethod 为 LED 类绑定一个方法，绑定的方法为 on()方法。onFun 是 on()方法所绑定的 c 原生函数的函数指针。onFun 函数具体的编写方式在第三章和第四章中介绍。

（2）构造对象

构造对象有两种方法，一种是构造 obj_run 传入的对象，称为根对象，如下图中的 sys 对象，其他对象则是一般对象，挂载在根对象之下。

一般一个项目中只构造一个根对象。



newRootObj 函数用于构造根对象。构造根对象需要传入对象名"led"和构造器函数指针，newRootObj 的返回值是根对象的指针。

```
MimiObj *led = newRootObj("led", New_LED);
```

一般对象的构造在父对象的构造器中完成，如果要在 sys 对象下挂载 led 子对象，就可以这样编写 SYS 类的构造器函数：

```
MimiObj * New_SYS(Args *args)
{
    // 继承自MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    // 通过LED类的构造器导入LED类
    obj_import(self, "LED", New_LED);
    // 使用LED类新建led对象，led对象作为sys对象的子对象
    obj_newObj(self, "led", "LED");
    return self;
}
```

obj_import 通过构造函数的函数指针导入一个类，上面代码中导入的类名为 LED。

obj_newObj 通过导入的类新建对象，新建对象作为子对象挂载在当前类下。

这时，通过调用下面的函数，就可以得到一个挂载了 led 对象的 sys 根对象了。

```
MimiObj *sys = newRootObj("sys", New_SYS);
```

3.dataArgs 动态参数列表

dataArgs 是基于链表的动态参数列表，其结构体是 Args，dataArgs 在运行时动态地申请和释放内存，所以可以在运行时增删改查参数，mimiobj 的属性和方法信息的存取均基于 dataArgs 参数列表实现。

dataArgs 支持整形、浮点型、字符串、指针类型的参数，也支持将原生的 c 语言变量绑定为 dataArgs 中的参数。

下面的例子是 Args 的基本使用方法。dataArgs 的实现原理会在后续的文章中介绍，在此文中不作重点。

```c
// 新建一个参数列表
Args *args = New_Args();
// 向参数列表中存入一个整形参数a，值为1
args_setInt(args, "a", 1);
// 取出参数a,值为1
int a = args_getInt(args, "a");
// 修改a的值为2
args_setInt(args, "a", 2);
// 再取出a,值为2
a = args_getInt(args, "a");
// 新建一个c语言的原生变量
float b = 3.0;
// 将变量b绑定为args中的参数
args_bindFloat(args ,"b", &b);
// 取出参数b，值为3.0
float b2 = args_getFloat(args, "b");
// 销毁参数列表
args_deinit(args);
```

4.dataMemory

dataMemory 为 dataArgs 提供动态内存申请和释放，在此文中不做重点。

3、用 PikaScript 点一个灯

鲁迅曾说过：“点灯是嵌入式领域的 helloworld”。

那我们就点一点灯，看一看实际工程中 PikaScript 如何为 mcu 提供面向对象的脚本支持。

我们以 STM32 的 HAL 库为例，假设在管脚 PA8 上连接了一个 LED 灯，我们称之为 led1，PA8 拉高时灯亮，拉低时灯灭。

那么点亮灯 led1 需要使用下面的 c 语言代码：

```c
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,SET)
```

我们希望使用如下的面向对象脚本更优雅地点灯~

```
led1.on()
```

下面我们看看如何使用 PikaScript 实现这个需求。

1.编写一个 onFun()函数。

```
void onFun(MimiObj *self, Args *args)
{
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,SET);
}
```

这个函数将会被作为一个方法注册到脚本对象里面，注册之后就不会再由开发者在 c 语言开发中调用了，只会在脚本运行时被脚本解释器调用。

onFun()函数的入口参数有 self 和 args，其中 self 是对象的指针，args 是传入和传出的参数列表（在第四章会用到）。

在 PikaScript 中，所有被绑定为方法的函数都使用这两个入口参数。

2.编写 LED1 类的构造器。

```
MimiObj * New_LED1(Args *args)
{
    // 继承自MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    // 为LED1对象绑定on()方法
    obj_defineMethod(self, "on()", onFun);
    return self;
}
```

obj_defineMethod 用于将编写的 c 语言函数绑定为脚本对象的方法。这里将 c 语言的原生函数 onFun()的函数指针作为参数注册进对象中，"on()"字符串声明了脚本调用时的方法名和参数，此处"on()"方法没有参数，含参数的方法绑定在第四章介绍。

3.编写根对象的构造器。

```c
MimiObj * New_MYROOT(Args *args)
{
    // 继承自MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    // 导入LED1类
    obj_import(root, "LED1", New_LED1);
    // 构造子对象"led1", "led1"的类是"LED1"
    obj_newObj(root, "led1", "LED1");
    return self;
}
```

obj_import 通过构造函数的函数指针导入 LED1 类。

obj_newObj 通过导入的 LED1 类新建 led1 对象，led1 对象作为子对象挂载在 MYROOT 类下。

4.创建根对象并监听串口的输入数据。当获得整行数据后直接当作脚本执行。

```c
int uartReciveOk; //串口单行接收完成的标志位
char uartReciveBuff[256];// 串口接收到的单行数据
int main()
{
    // 硬件的初始化代码略

    // 创建根对象
    MimiObj *myRoot = newRootObj("myRoot", New_MYROOT);
    while(1)
    {
        // 串口已经接收到单行数据
        if(uartReciveOk)
        {
            // 执行串口输入的单行数据
            obj_run(myRoot, uartReciveBuff);
            // 清除串口接收标志位
            uartReciveOk = 0;
        }
    }
}
```

5.这时只要向 mcu 的串口发送 led1.on()，灯就亮了（神奇不~）

4、用 PikaScript 实现一个加法函数。

上面的例子中的方法没有输入输出，下面的例子中，我们会定义一个 TEST 类，并为 TEST 类添加一个 add 方法，实现加法功能，看一看如何绑定一个带有输入输出的方法。

1.编写一个 add()函数。

和上次 onFun 函数一样，这次被绑定的函数是 addFun 函数。

```c
void addFun(MimiObj *self, Args *args)
{
    //取出输入参数
    int val1 = args_getInt(args, "val1");
    int val2 = args_getInt(args, "val2");
    //实现方法功能
    int res = val1 + val2;
    //将返回值传回参数列表
    method_returnInt(args, res);
}
```

args_getInt 用来从参数列表中取出整型参数，此处从参数列表中取出输入参数 val1 和 val2。参数列表还支持 float 类型、字符串类型和指针类型。

method_return Int 用来传递方法的返回值，同样可以返回 float 类型、字符串类型和指针类型。

2.定义测试类的构造器

```c
MimiObj *New_MimiObj_test(Args *args)
{
    //继承MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    //绑定方法
    obj_defineMethod(self, "add(val1:int, val2:int)->int", addFun);
    return self;
}
```

这次用 obj_defineMethod 绑定一个带有输入输出参数的方法。

"add(val1:int,val2:int)->int"是 python 的带类型函数声明语法，表示 add 方法有 int 类型的 val1、val2 两个输入参数，输出参数也是 int 类型。同样，传入 addFun 函数的函数指针。

3.编写根对象的构造器。

```c
MimiObj * New_MYROOT(Args *args)
{
    // 继承自MimiObj基本类
    MimiObj *self = New_MimiObj(args);
    // 导入TEST类
    obj_import(root, "TEST", New_MimiObj_test);
    // 构造子对象"test"，"test"的类是"TEST"
    obj_newObj(root, "test", "TEST");
    return self;
}
```

在根对象中挂载 test 子对象。

4.创建对象并测试运行脚本

```c
void main()
{
    //新建根对象
    MimiObj *root = newRootObj("root", New_MYROOT);
    //运行脚本(也支持 "res = test.add(val1 = 1, val2= 2)"的调用方式)
    obj_run(root , "res = test.add(1, 2)");
    //从root 对象中取出属性值res
    int res = obj_getInt(root, "res");
    //销毁根对象
    obj_deinit(root);
    /* 打印返回值 res = 3*/
    printf("%d\r\n", res);
}
```

obj_run 运行脚本后会动态新建 res 属性，该属性属于 root 对象。obj_deinit 用于销毁对象，所有挂载在 root 对象下的子对象都会被自动销毁。本例中 root 对象挂载了 test 对象，因此在销毁 root 对象前，test 对象会被自动销毁。

三、PikaScript 部署指南

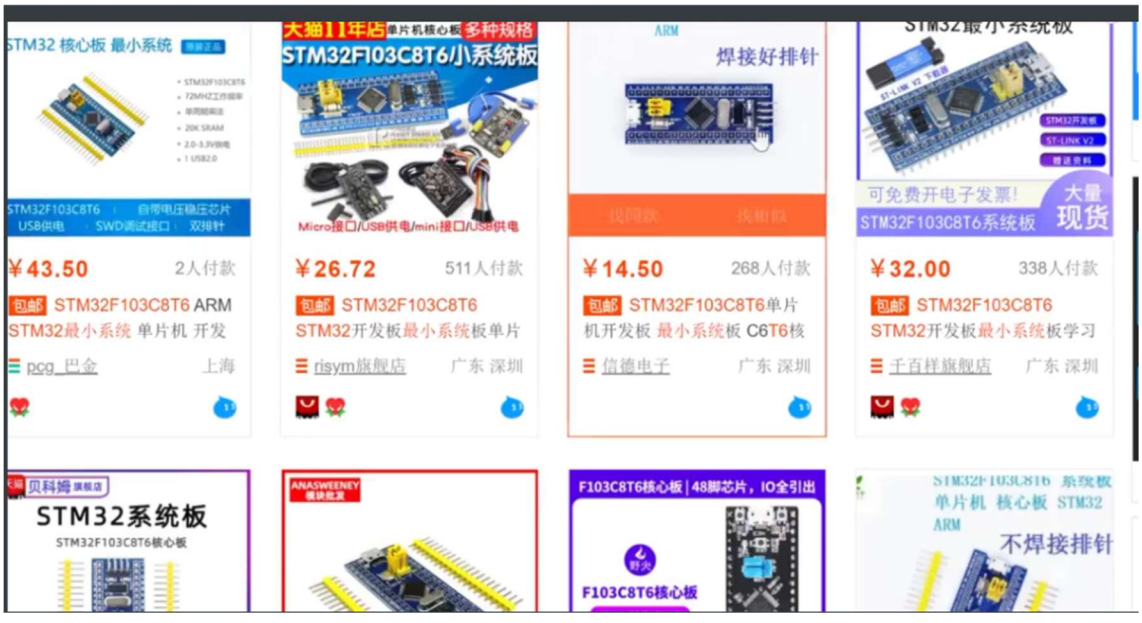
本指南改编自教学视频【手把手 mimiScript 教程】轻松在 stm32f103 跑 python

bilibili 搜索"手把手 mimiscript"即可找到。

后续的视频会分 p 放在这个视频底下。

视频中有更多细节，动手试验时请参考视频。

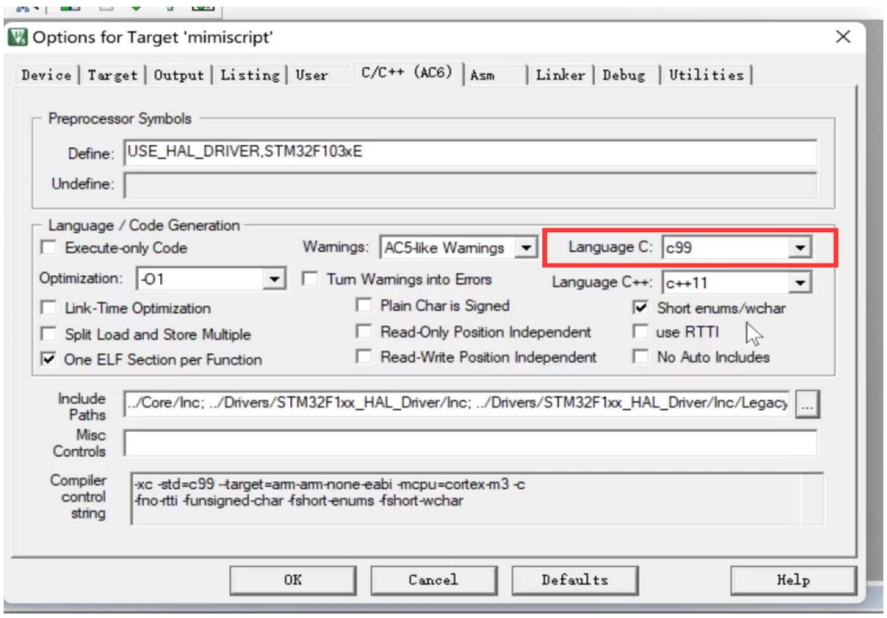
指南使用的单片机的平台是 sm32f103c8t6，一个最小系统，大概长这样。



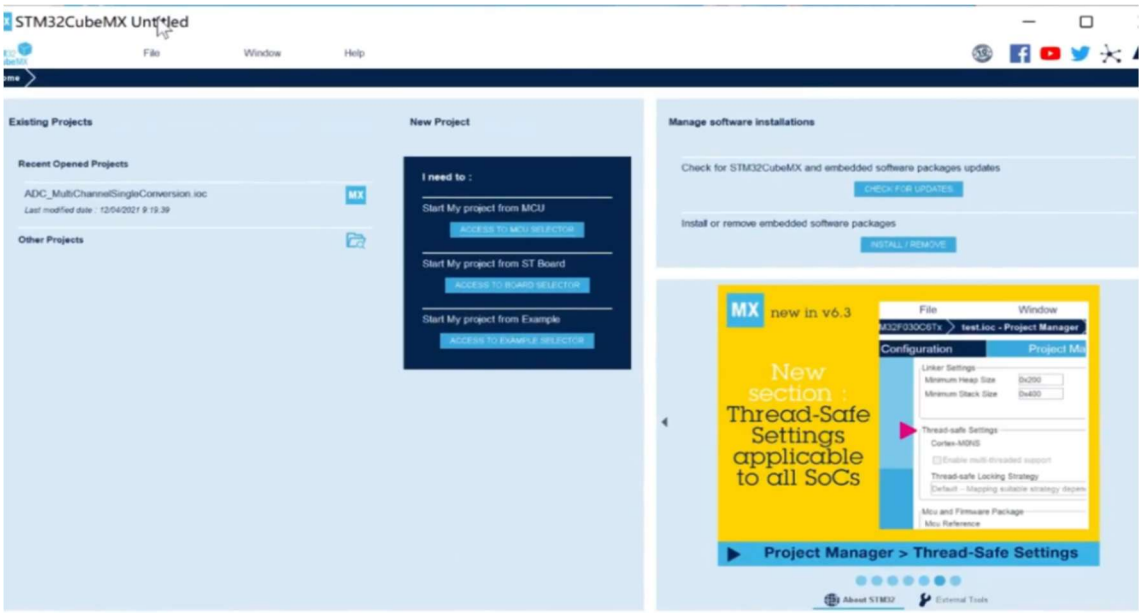
不是广告。随便哪一个都是可以的。

调试器使用 dap-link。

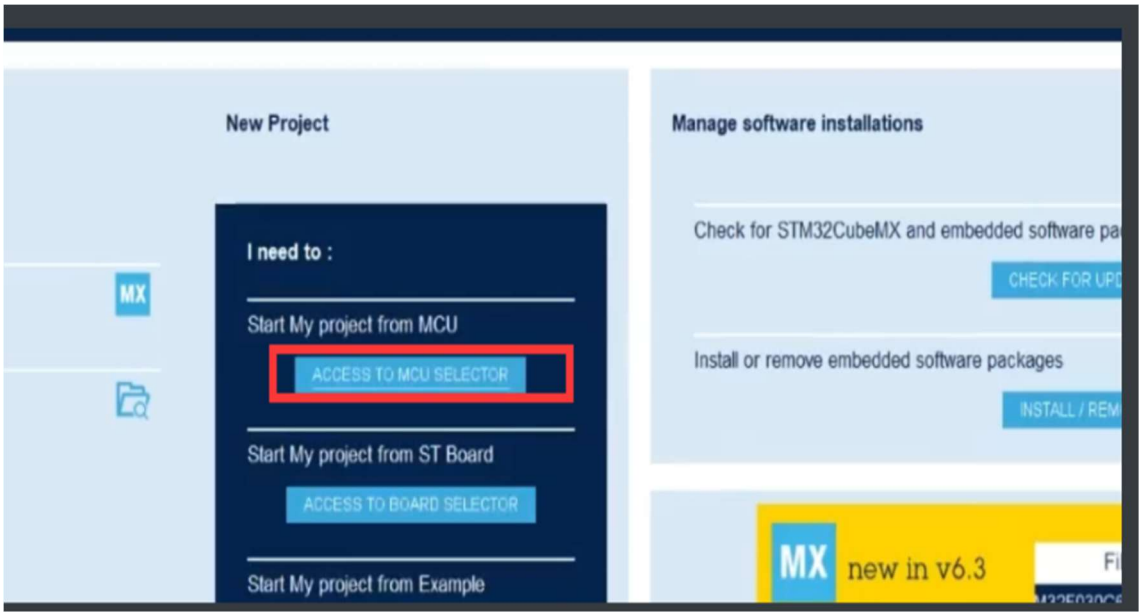
使用的开发环境是 mdk-arm keil。



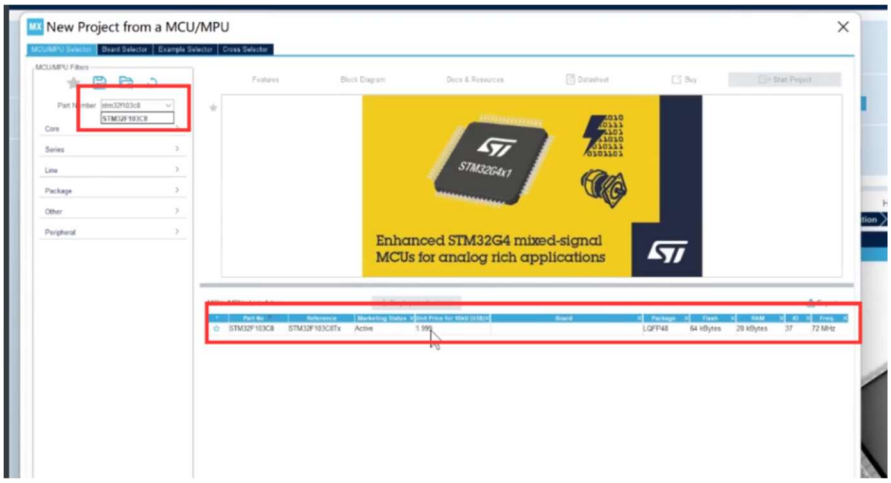
然后新建工程的话也是用现在比较流行的方法，使用 stm32cubeMX 来新建工程和跟初始化代码。



我们直接从头开始。

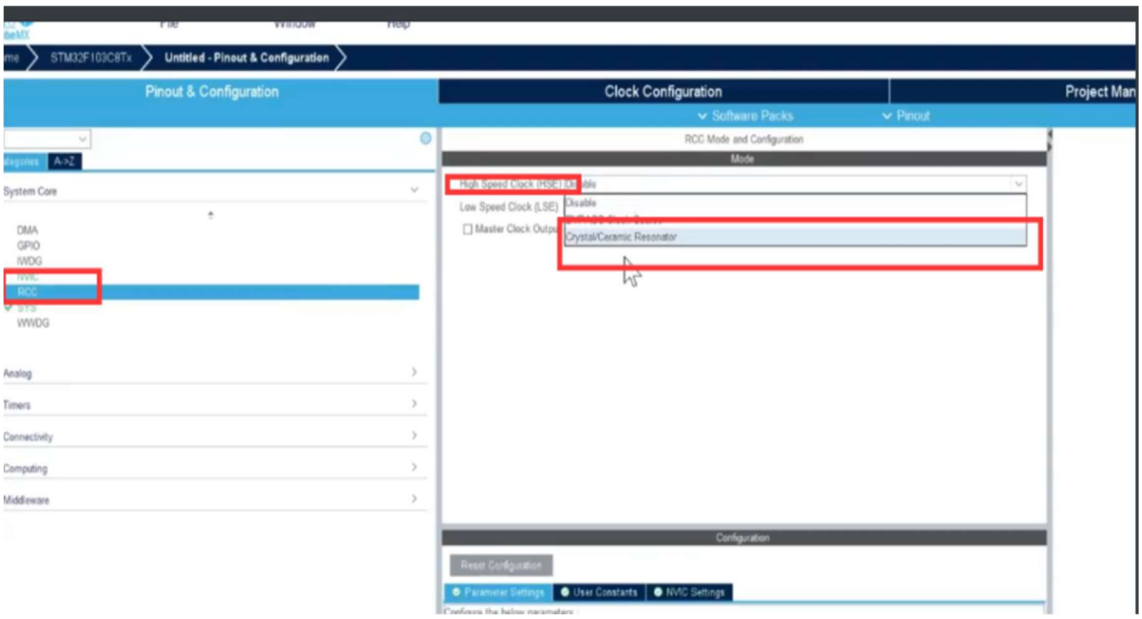


我们就从最开始的选 mcu 开始。



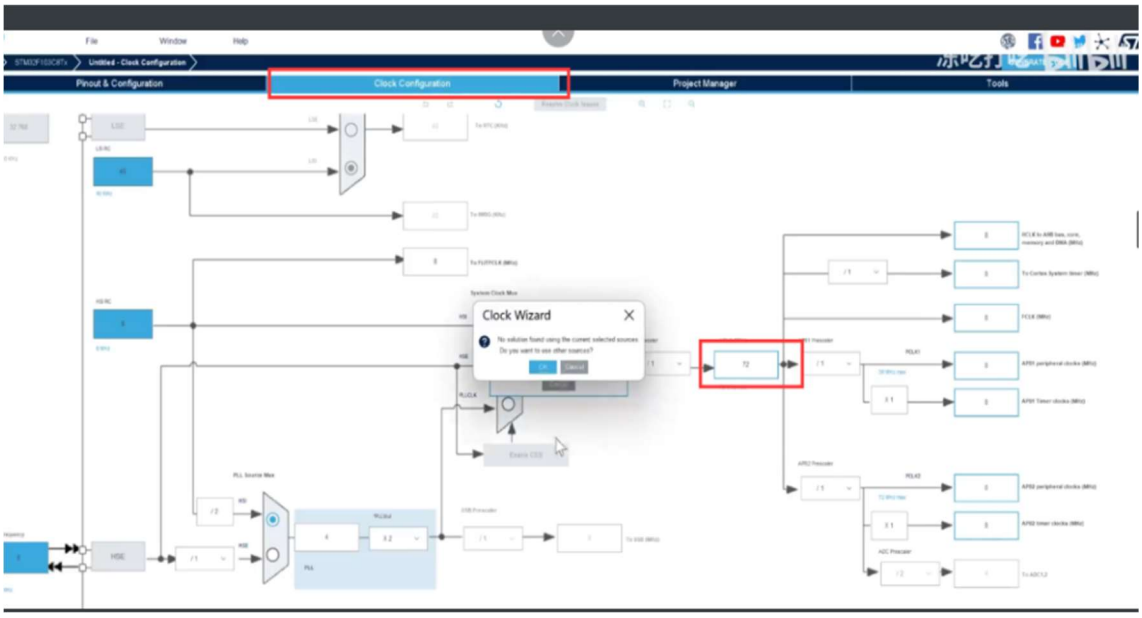
mcu 选择 stm32f103c8。

首先惯例还是先把这个时钟给配上

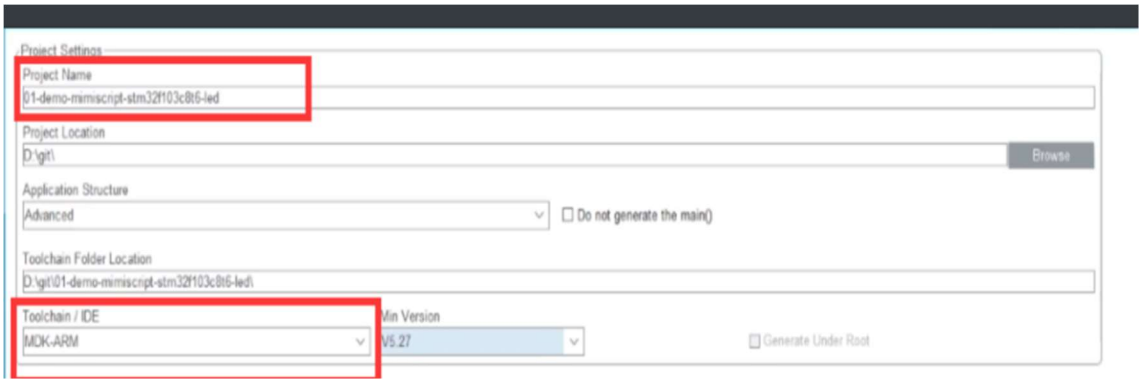


然后打开 swd 的调试选项

然后主频给到最大 72MHz，晶振的输入是 8MHz。



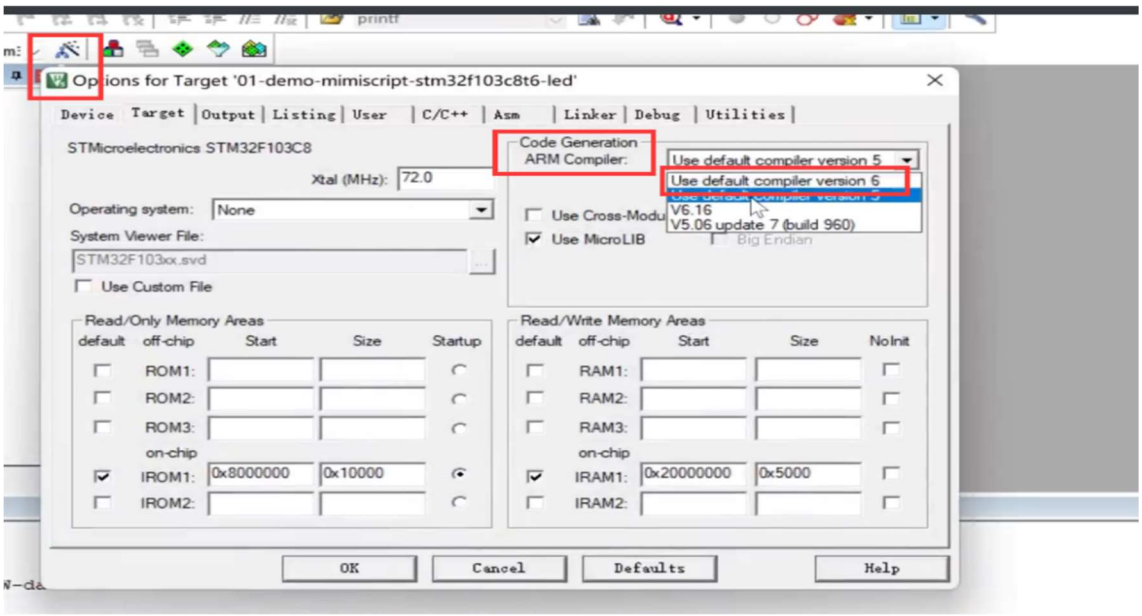
然后输入 project 名称



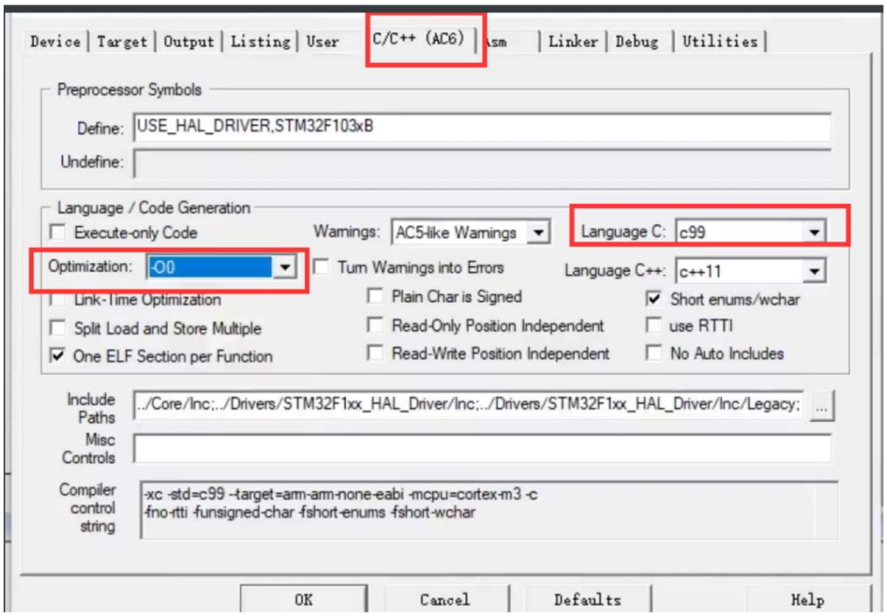
关于这个 location 我放在我的 D 盘的 git 文件夹里面，然后结构的话是选择 advance 的结构，其实都可以。ide 要选 MDK-ARM。

这个地方比较关键，就是堆栈的设置，这个地方 PikaScript 以前的版本是需要修改的，现在不用改，默认就可以了。选择 stm32 一些用到的一些包，选择第二个仅仅拷贝需要的库文件。这里我比较习惯的是让它生成单独的驱动文件。

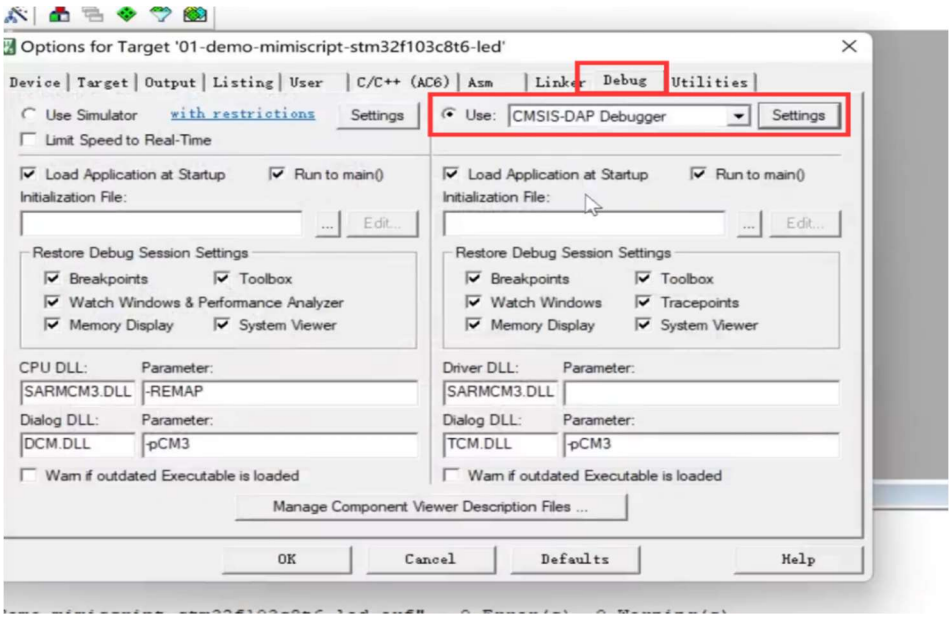
然后 generate。



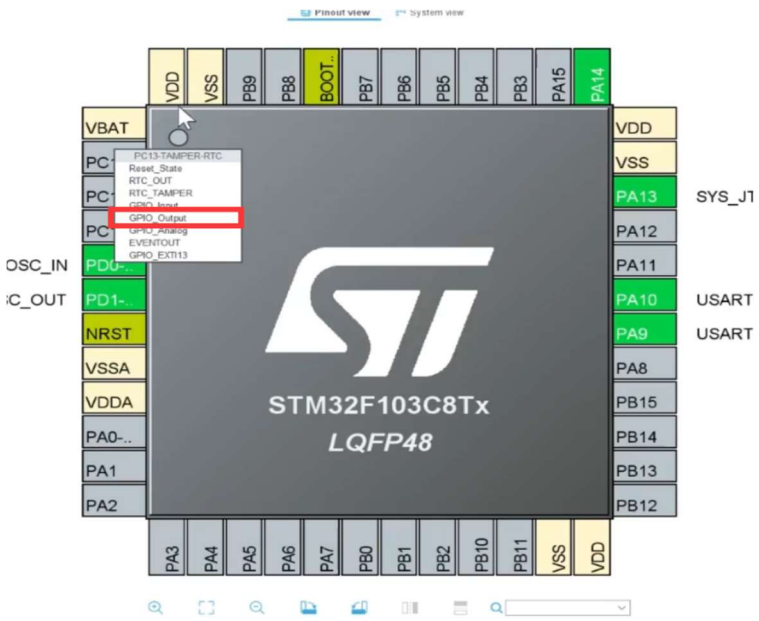
编译器的话，既然有 6 还是用 6，6 的话更快。5 也没问题都是支持的，然后再把 c 标准选为 c99，然后这个优化关掉，不关掉可能有问题，找到-O0 关掉优化，不然它会把一些关键的指针给它优化掉，不知道为什么把指针优化掉就没了。



下载器选择 CMSISDAP-link 来 debug。



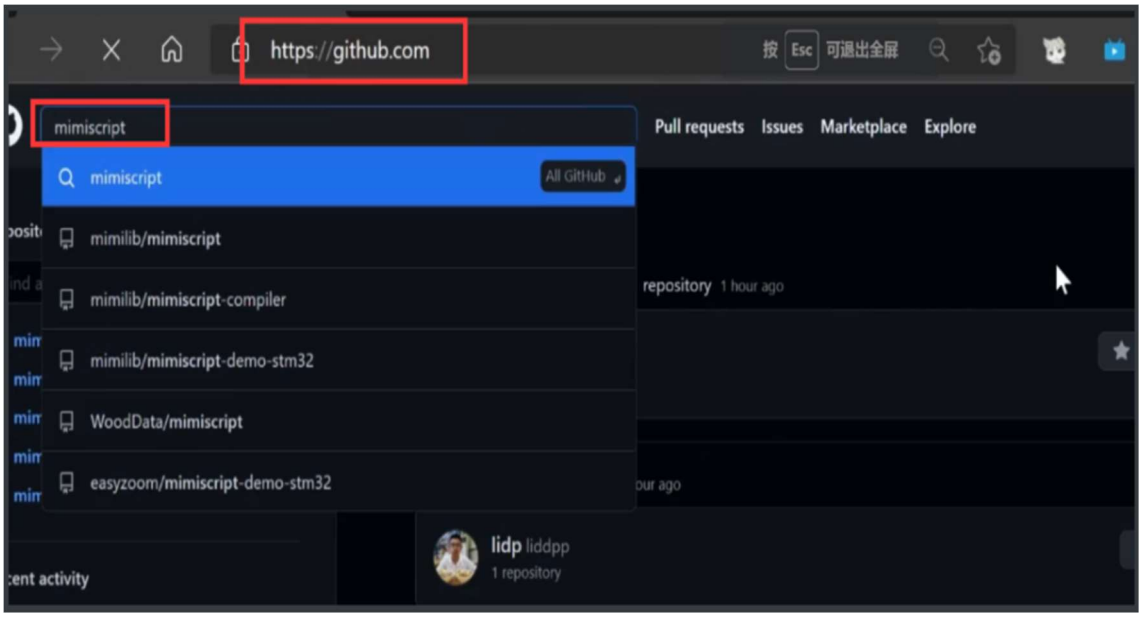
我们再开一个灯，板子上它的灯是 PC13。

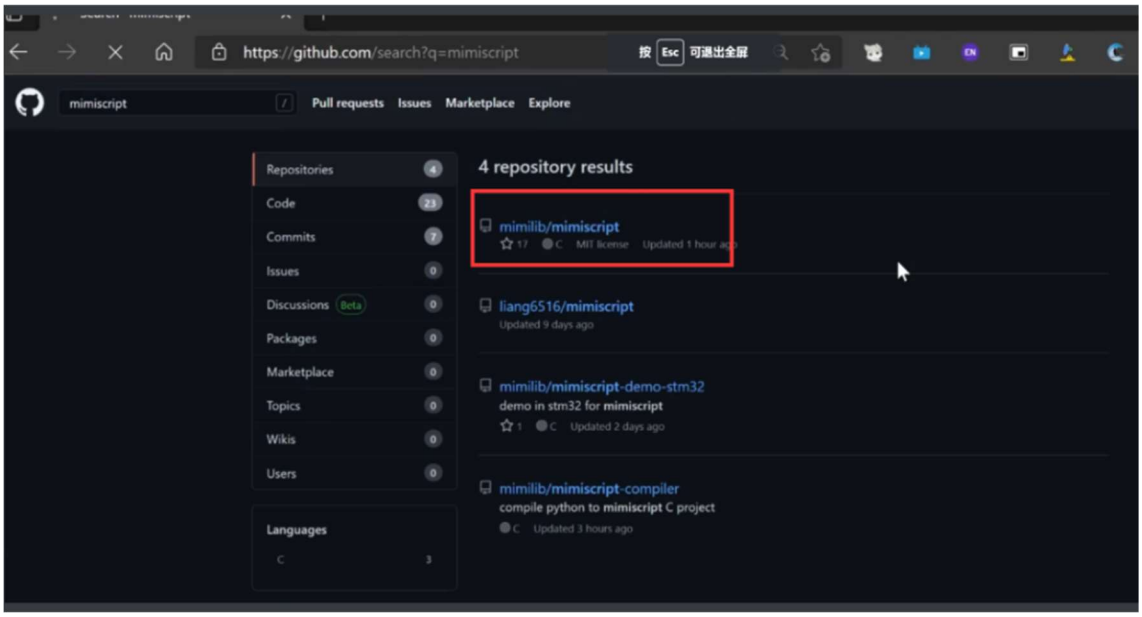


好，现在我们得到的就是一个纯 cubeMX 生成的工程了，然后下面我们来开始给工程里面添加 PikaScript，首先让我们打开浏览器，然后打开 github，然后搜索 PikaScript，第一个就是，大家可以点一个 star 关注最新进展。

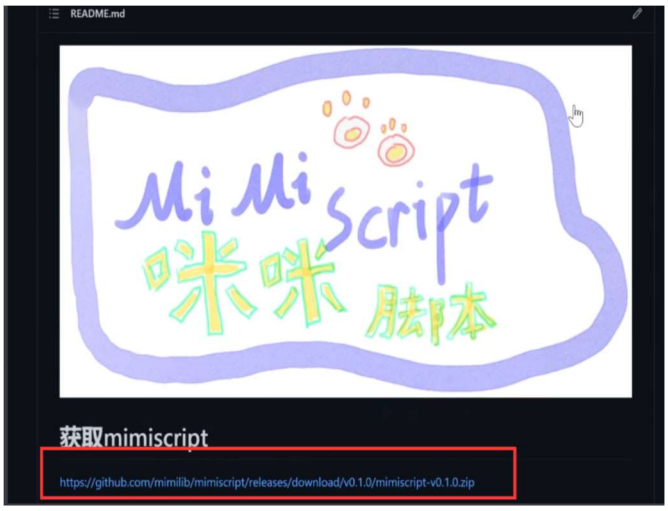
仓库链接：

https://github.com/mimilib/PikaScript

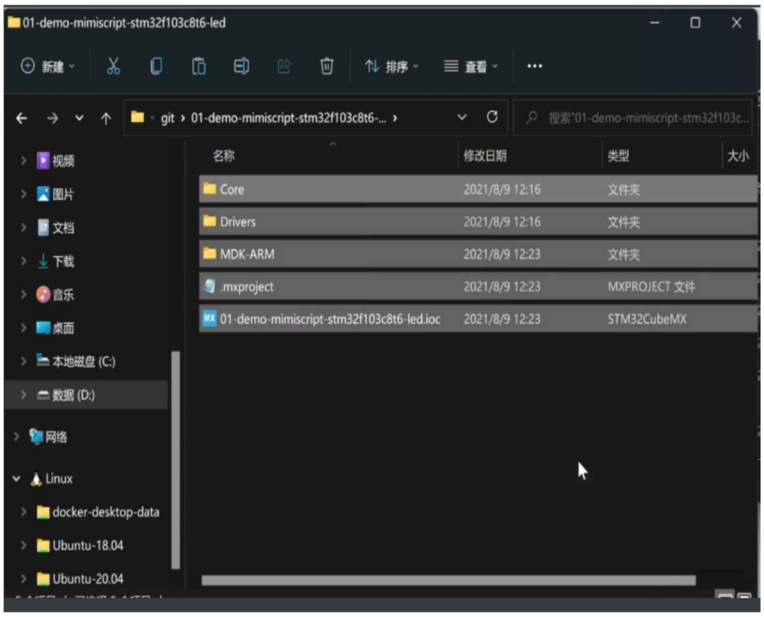




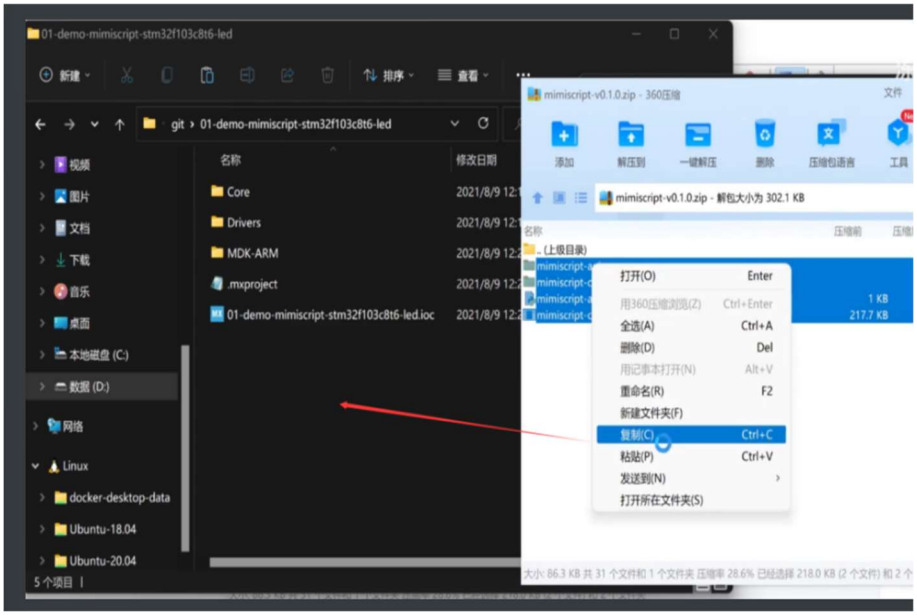
然后再给大家展示一下我新画的漂亮的典雅的 logo，然后这个是获得最新版本的 PikaScript 的源码包。



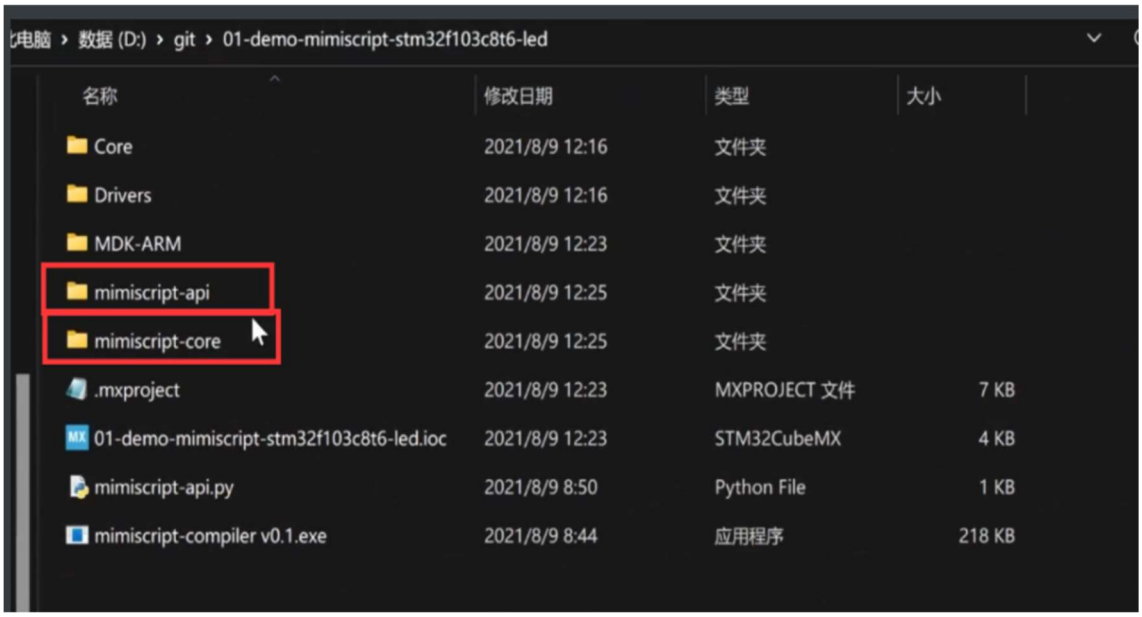
我们把工程的文件夹打开。

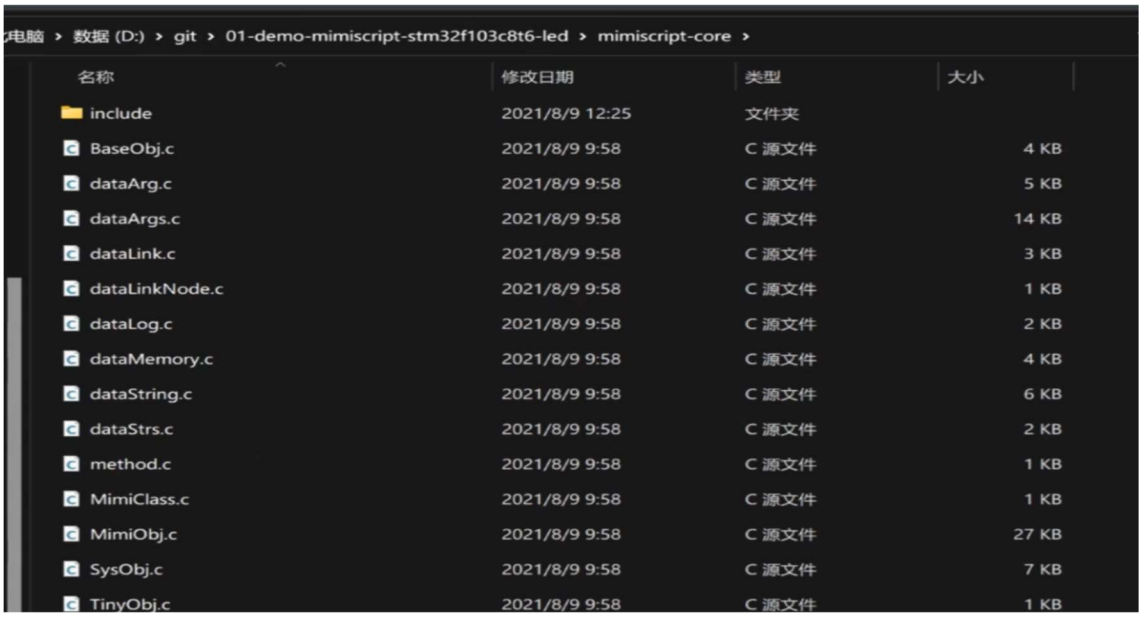


然后就直接把源码全部包括它的可执行文件全部复制过来，复制到工程所在的根目录里面就可以了。



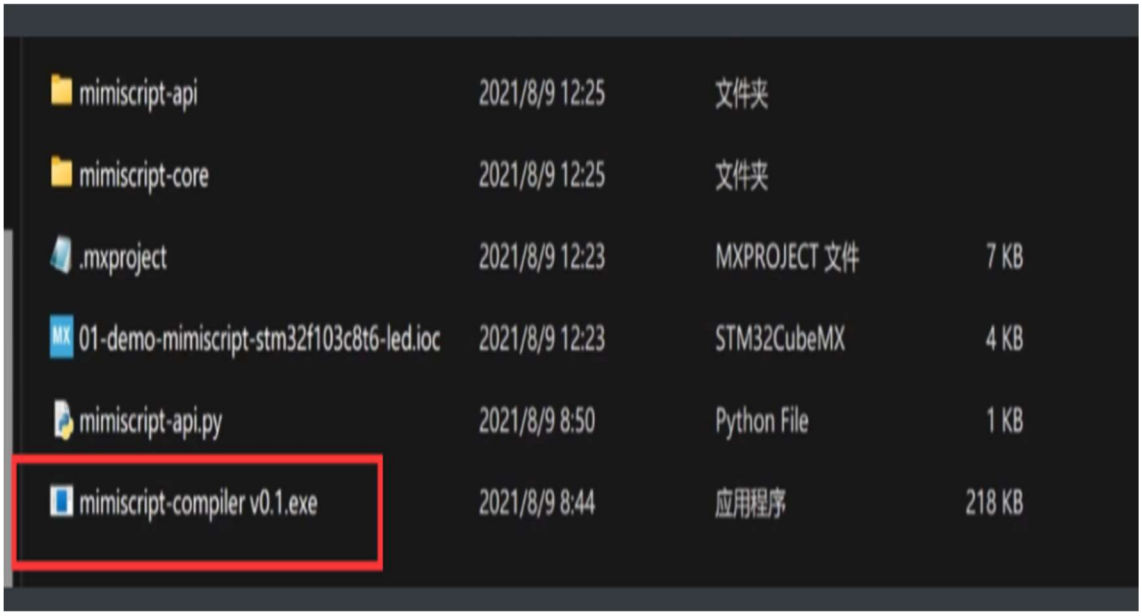
然后这几个文件夹再简单说一下，PikaScript-core 是 PikaScript 的解释器内核和对象模型支持。



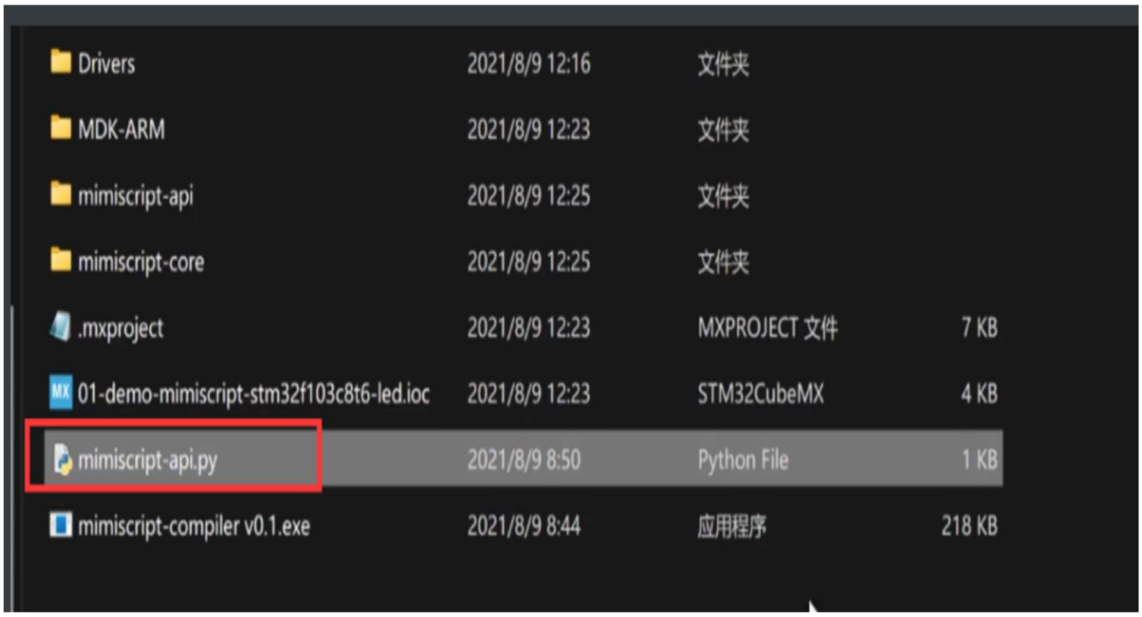


然后打开 PikaScript-api，你看他是空的对吧？

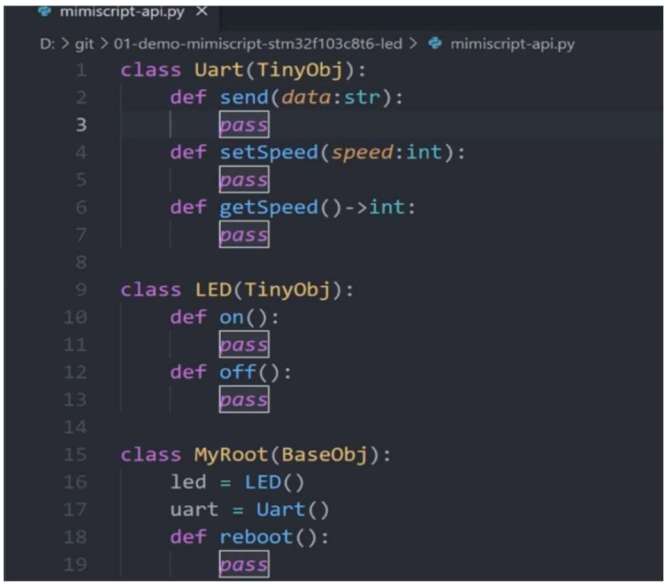
因为这边的 API 本来是要我们自己编写的，有了 PikaScript-compiler 预编译器，或者叫代码生成器，它就可以自动生成 PikaScript 的 API 的 C 源码。



然后我们用 vscode 打开 PikaScript-api.py。这个 python 随便一个编辑器打开都是可以的。



大家可以看一下里面的内容，里面内容的话就是一个类的接口定义，就是完全是使用 Python 的语法。



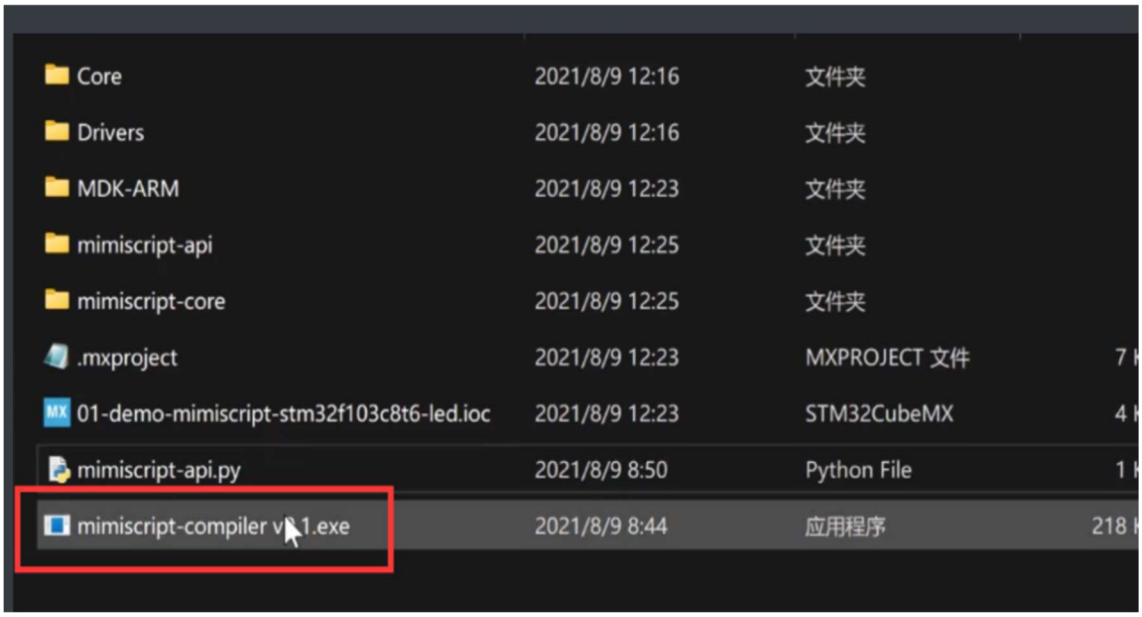
第一行表示新建一个类，类名是 Uart，然后他继承是继承自 TinyObj，这是 PikaScript 很基本的一个类，这个是最小的类，几乎没有什么功能，但是占用内存是最少的。串口和 led 都是从 TinyObj 里面继承，然后 MyRoot 就是根对象的类，我们一般都会做把 PikaScript 固件做成一个对象树，把其他的一些对象挂载在这根对象上面。

```
class MyRoot(BaseObj):
    led = LED()
    uart = Uart()
    def reboot():
        pass
```

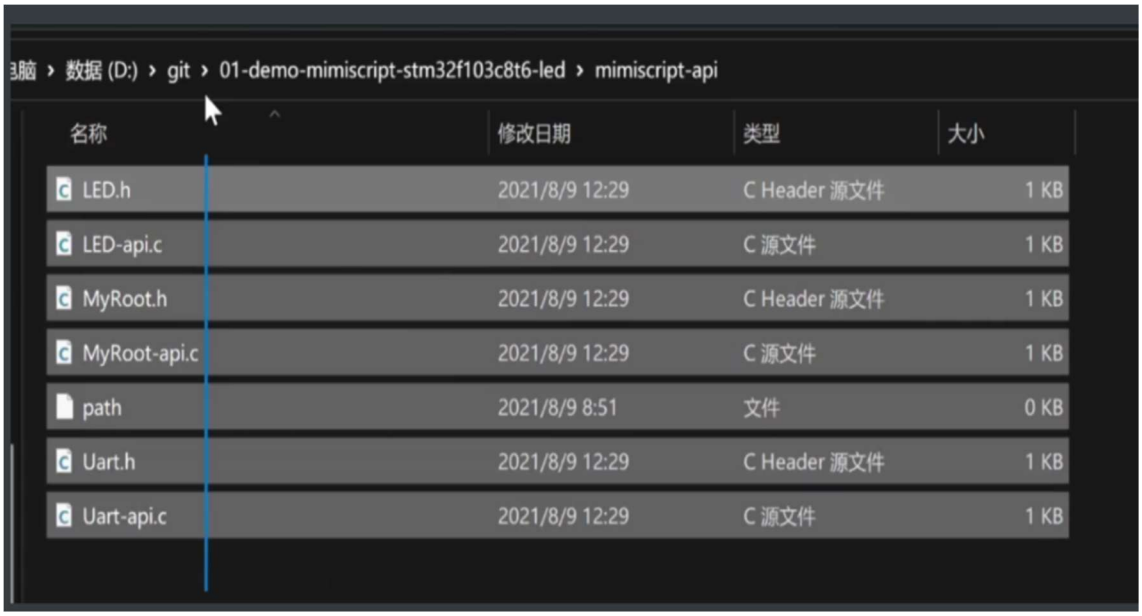
MyRoot 类继承的是 BaseObj，BaseObj 也很小，没什么功能，但是它比 TinyObj 多了一个功能，就是它可以挂载子对象。这个地方就挂在了一个子对象 led，然后 led 就从 LED 类构造一个子对象，uart 是通过 Uart 类新建的一个子对象，然后 def 就是定义的一些方法的接口，这个是给 uart 定义了一个 send 接口，send 接口它里面有传入参数，那个参数名字叫 data，类型是 str，就是字符串类型。speed 是 int 类型，然后一个箭头指 int，这个是指返回的是 int 型，这个就是 Python 带类型的函数声明方法，这个都是标准 Python 语法。把这个参数名写在前面，类型写在后面，然后中间是冒号隔开，然后返回值的话是用一个箭头指回来。

```
def setSpeed(speed:int):
    pass

def getSpeed()->int:
    pass
```

有了这个 PikaScript-api.py 之后，我们就可以使用 PikaScript-compiler 自动生成 PikaScript-api 源码了，双击运行 PikaScript-compiler。



现在打开 PikaScript-api 文件夹，可以看到已经生成了 C 源码了。

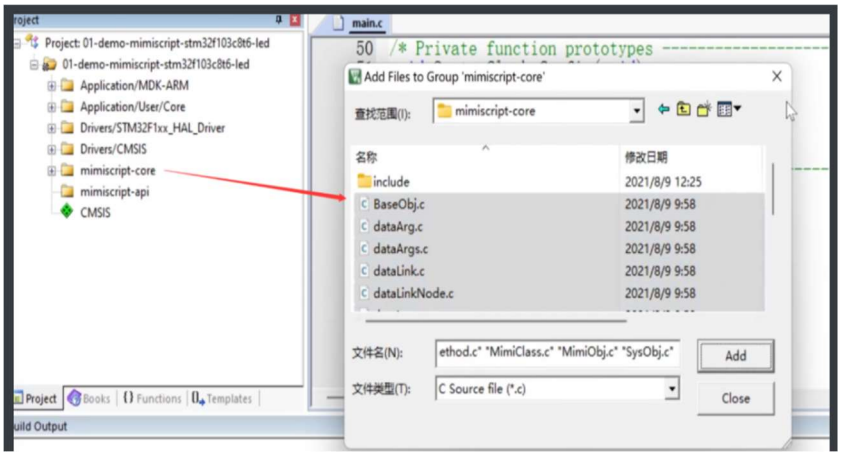


我们把它删掉，然后来写我们的 LED 灯的例程。

首先把串口删掉，我们历程只用到 LED 灯，然后灯的话就一个 on 方法，一个 off 方法，然后 UART 也不需要了。

```
class LED(TinyObj):
    def on():
        pass
    def off():
        pass

class MyRoot(BaseObj):
    led = LED()
```

好，我们的 PikaScript-api.py 就写完了，就是我们现在用到的 API，有一个根对象，然后根对象里面挂载了一个类型为 LED 的一个灯，然后它里面有一个 on 方法和一个 off 方法，没有传入参数，也没有返回参数，我们再用 PikaScript-compiler 编译一次。编译完之后，我们把内核的源码和 API 都添加到工程里面，我们两新建两个组，一个是 PikaScript-core，一个是 PikaScript-api。

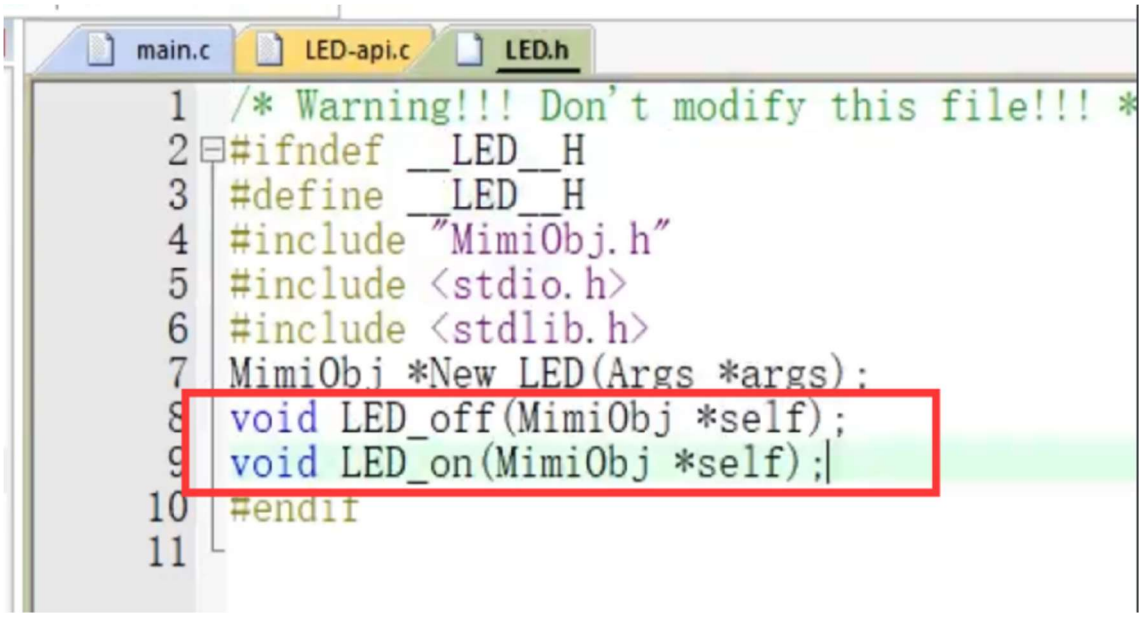


PikaScript-core 添加 PikaScript-core 文件夹的所有.c 源码，然后再添加 include 路径。PikaScript-api 也要添加源码，然后添加路径。

然后我们编译一次，看到报错提示，没有定义 LED_off 函数和 LED_on 函数。

```
Error: L6218E: Undefined symbol LED_off (referred from led-api.o).
Error: L6218E: Undefined symbol LED_on (referred from led-api.o).
```

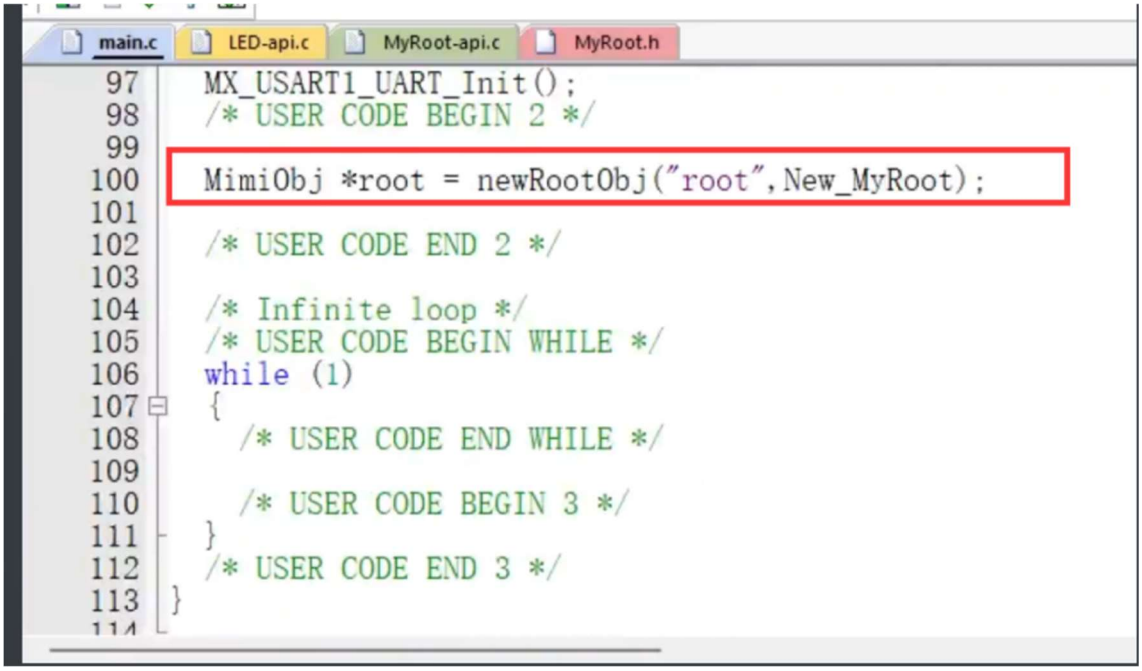
这是正常的，这两个函数是 c 里面被绑定的原生函数，我们是需要自己去实现的，实现完之后由脚本解释器内核来调用，我们怎么看哪些 C 的原生函数需要我们去定义呢？我们可以打开 API 的头文件，比如这个 LED.h，就可以看到声明了。



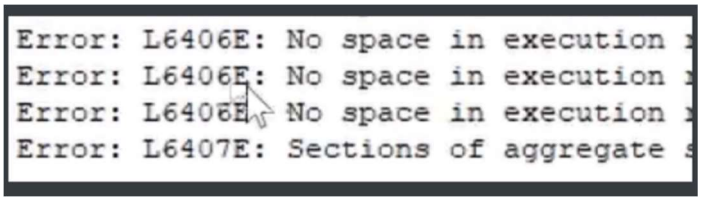
这两个函数是 c 的原生函数，它们的声明由 PikaScript-compiler 自动生成，会自动绑定到这个 PikaScript 里面，我们需要自己去实现它。这个例程里面我们在 main.c 里面实现这两个函数。

```
/* USER CODE BEGIN 0 */

void LED_off(MimiObj *self)
{
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
}
void LED_on(MimiObj *self)
{
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
}

/* USER CODE END 0 */
```

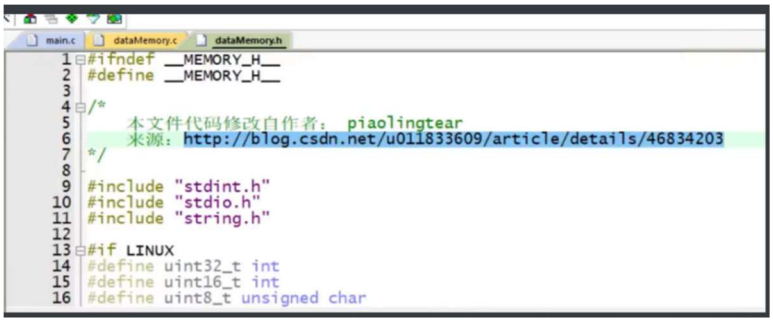
然后我们来试一下运行脚本，看看这两个函数有没有被调用到。我们首先需要新建一个 PikaScript 根对象。



然后编译一次，发现还是报错了。

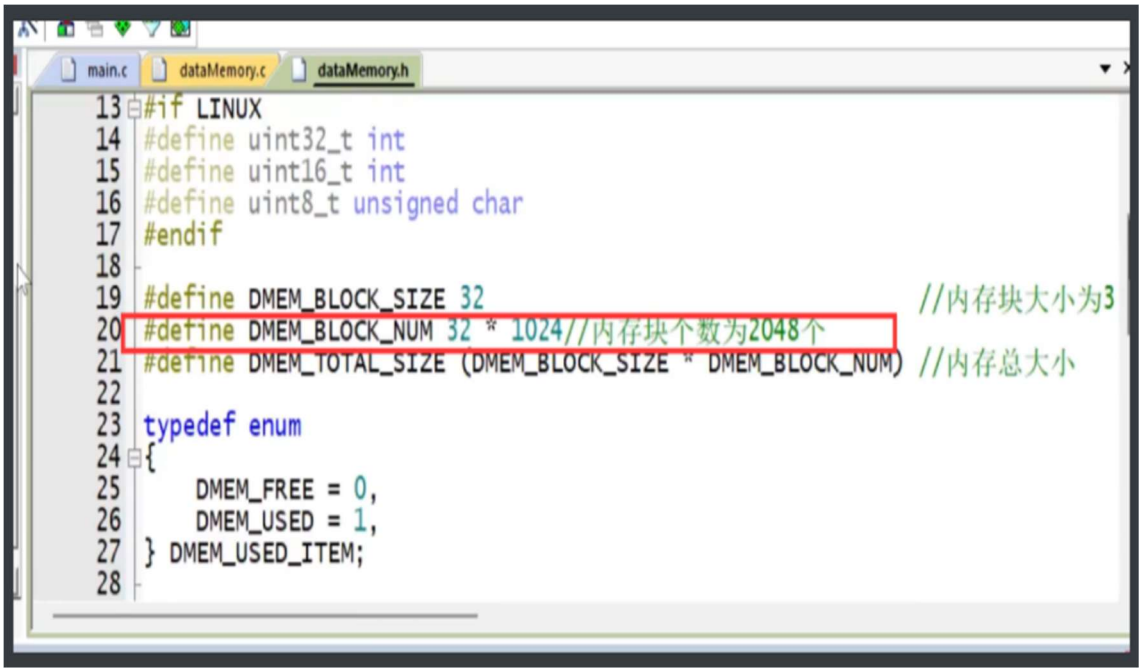


链接的时候没有空间了。这是因为我默认的给 PikaScript 分配的内存是 1M，但是 stm32f103c8t6 只有 20K 内存，所以说它显然是没有那么多内存就给 PikaScript 分配的。那么内存的大小在哪里更改呢，这也是 PikaScript 唯一需要进行一个配置，就是在 dataMemory.h。



dataMemory 是管内存申请释放的，然后参考了作者 piaolingtear 的源码，就在这个地方，大家可以去学习了解一下。现在用到的文件在他的基础上进行了修改。

要改的就是这么一个定义。内核唯一需要修改的仅有这里，一个宏需要修改，DMEM_BLOCK_SIZE 就是一个代码块所占的字节，这个是不用改的，我们就改这个代码块的数量 DMEM_BLOCK_NUM。



32 是一个代码块所占的字节，然后*32 的话，正好是 1024 字节，就是 1K。我默认写的是又乘了个 1024，所以说是 1k×1024 是 1M。现在我们给 stm32 就分配 10k，再编译一下，就可以了。

```
16 ne uint8_t unsigned char
17 f
18
19 ne DMEM_BLOCK_SIZE 32                                    //内存块大小为
20 ne DMEM_BLOCK_NUM 32 * 10//内存块个数为2048个
21 ne DMEM_TOTAL_SIZE (DMEM_BLOCK_SIZE * DMEM_BLOCK_NUM) //内存总大小
22
23 ef enum
24
25 MEM_FREE = 0,
26 MEM_USED = 1,
27 M_USED_ITEM;
28
29 ef struct
```

然后我们使用 obj_run 来运行脚本，指定 root 对象为根对象，执行“led.on()"。如果脚本解释器执行正常的它会先遍历根对象，然后找到里面的子对象和方法，最后调用绑定过的本地函数然后执行。

```
98  /* USER CODE BEGIN 2 */
99
100 MimiObj *root = newRootObj("root",New_MyRoot);
101 obj_run(root, "led.on()");
102 /* USER CODE END 2 */
```

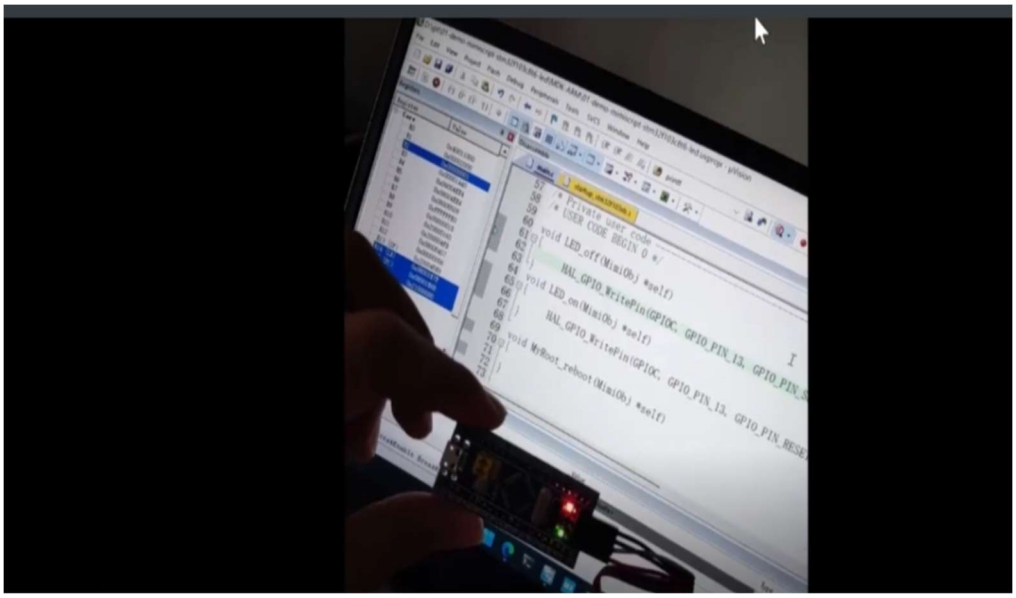
好，我们请大家看看它会不会执行进来，我在 LED_on 函数打一个断点，发现可以执行进来，这就说明脚本解释器运行正常了。

```
main.c | MimiObj.c | TinyObj.c | startup_stm32f103xb.s | MyRoot-api.c | MyRoot.h
61 {
62     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
63 }
64 void LED_on(MimiObj *self)
65 {
66     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
67 }
68
69 void MyRoot_reboot(MimiObj *self)
70 {
71
72 }
```

然后我们来试试写个流水灯。

```
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_Delay(200);
    obj_run(root, "led.on()");
    HAL_Delay(200);
    obj_run(root, "led.off()");
    /* USER CODE END WHILE */
```

LED 已经闪起来了，第一个例程就这么愉快的结束了，还是挺愉快的是吧？



在以往的版本的话，其实我们是需要自己去编写 API 函数的，API 函数里面包括一个构造器 New_MyRoot，然后 obj_import 是导入 LED 类，然后 obj_newObj 新建 led 对象，对应的是 PikaScript-api.py 里面的“led=LED()”，然后 class_defineMethod 是绑定的一个方法，对应的是“defon()”。

```
MimiObj *New_MyRoot(Args *args){
    MimiObj *self = New_BaseObj(args);
    obj_import(self, "LED", New_LED);
    obj_newObj(self, "led", "LED");
    class_defineMethod(self, "reboot()", MyRoot_rebootMethod);
    return self;
}
```

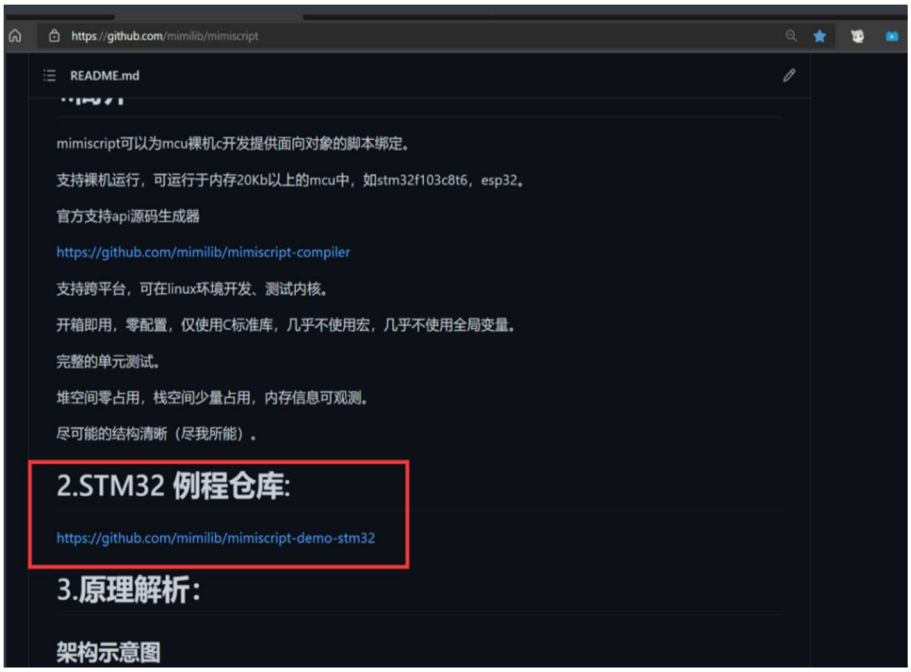
然后被绑定的方法是这样子的，它的接口的话是指向本对象的 MimiObjectself 指针和动态参数列表 Args。绑定完之后，内核就会调用这个方法，然后这个方法又调用了一个本地函数 LED_on()。

```
void LED_offMethod(MimiObj *self, Args *args){
    LED_off(self);
}

void LED_onMethod(MimiObj *self, Args *args){
    LED_on(self);
}
```

对于 LED_on()这个函数，API 生成器只会去生成一个声明，他不会去帮他实现，因为从原理上说，这是需要你在 c 里面去编写的，比如 LED_on()中使用了 HAL 库，这是没有办法通过 PikaScript-api.py 来生成的。然后我就把这个例程上传到 github 的例程仓库里面。

https://github.com/mimilib/PikaScript-demo-stm32

感兴趣的同学可以自己试试自己写一些其他的 demo，然后 pullrequest 到这个仓库里面，体验一下开源社区 github。



好，第一个 demo 到这里就结束了，就是这么简单。然后大家可以回去下载一下试试。