

编写mBlock扩展



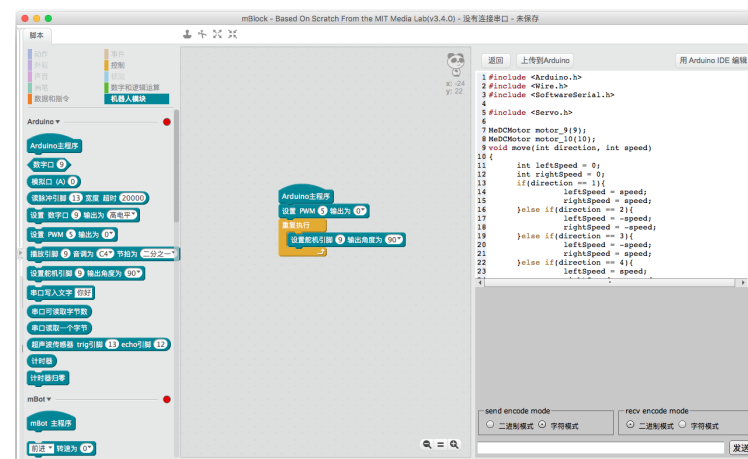
什么是mBlock扩展

mBlock扩展能为mBlock添加新的语句块。您可以用扩展来支持您喜欢的Arduino传感器，或者Lego, LittleBits等机器人和电子模块产品。

每个人都可以为mBlock撰写扩展。这使得mBlock成为机器人和电子创作的理想工具。

Scratch模式和Arduino模式

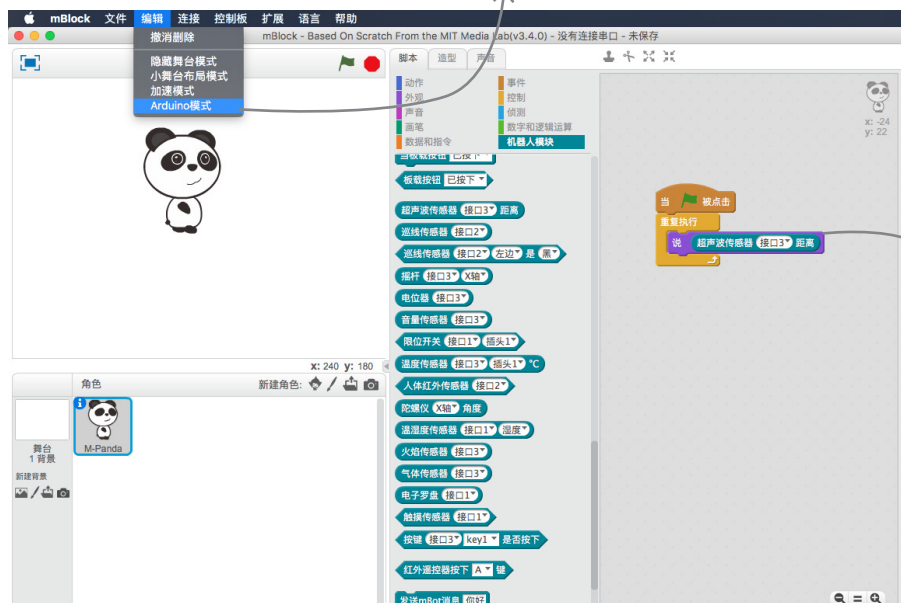
mBlock的每个语句块都有两个模式：Scratch模式和Arduino模式。了解它们的区别对于开发扩展很有必要。



使用 编辑/Arduino模式菜单在Scratch模式和Arduino模式之间切换

► Scratch模式

在Scratch模式，机器人Arduino控制板必须一直连接电脑，也可以使用Scratch丰富的图形功能。



▲ Arduino模式

在Arduino模式下，程序c将被上传到机器人，这使得机器人能够脱离电脑工作。但因为没有电脑，也就无法使用Scratch的图形功能了。

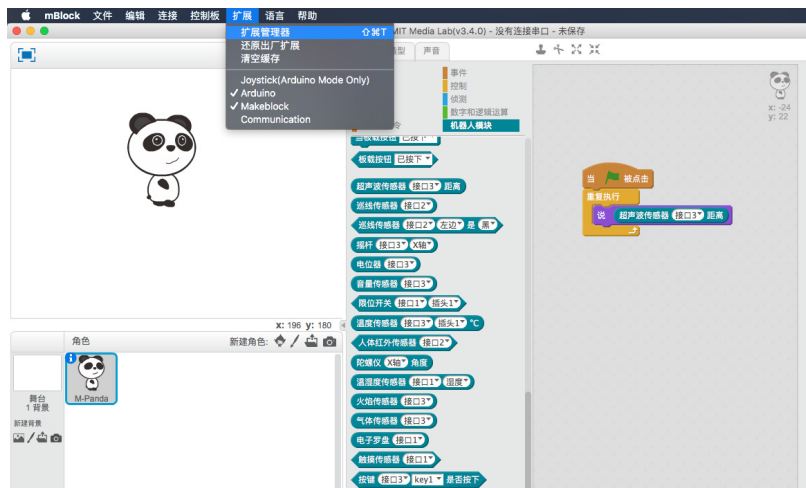
“说”指令块只能在Scratch模式下用，但“重复执行”指令块在Scratch和Arduino模式下都可以使用。

使用扩展

使用扩展中心，您可以浏览现有的扩展，找到自己想要的，然后一键下载安装。

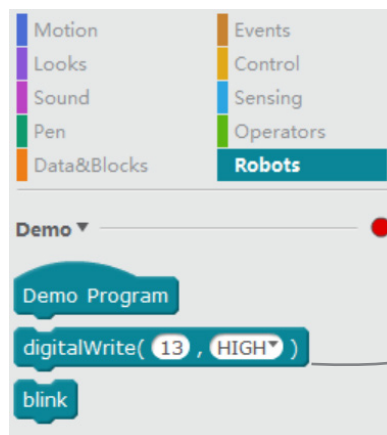
下面的说明讲解了怎样向mBlock添加一个扩展。

- 1 在“扩展”菜单下，选择“扩展管理器”，打开扩展管理器。



- 3 刚刚下载的扩展，会出现在“机器人”群组中。

- 2 可以通过关键字来搜索扩展。找到想要的扩展后，点击右侧“下载”按钮下载安装。



编写扩展

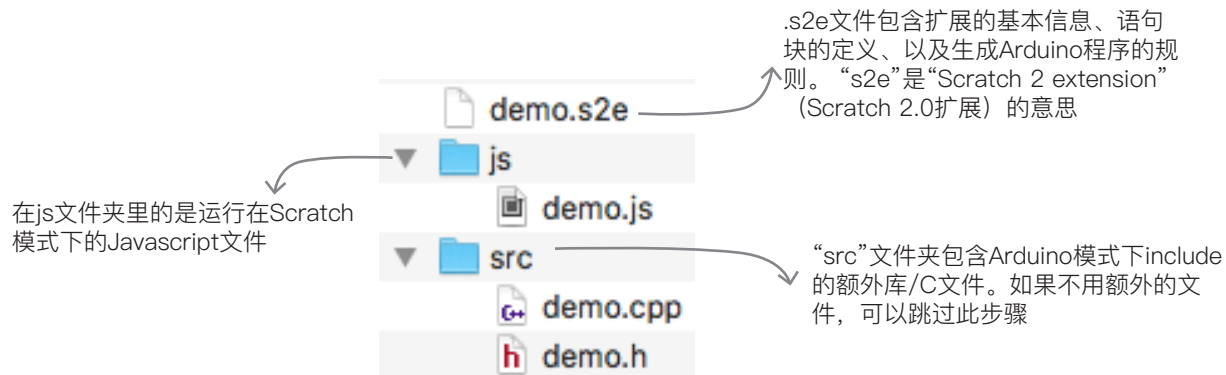
编写扩展没有想象的难。大部分工作是编辑一些文本文件。如果来实现Arduino模式，需要一些Arduino知识；而Scratch模式需要对Javascript有所了解。您可以跳过其中一个模式，这样的话用户在被跳过的模式下操作会没有效果。

下面是开发扩展需要做的一些事情：

- 撰写基本信息，比如扩展名称和作者
- 定义语句块的外观形状
- 告诉mBlock怎样产生Arduino代码
- 编写在Scratch模式下执行的代码
- 包含额外的Arduino C文件

扩展的文件结构

每个扩展都是一个由文件夹压缩的.zip压缩文件。下面是Demo扩展解压之后的样子：



提示

现有扩展的源代码是学写扩展的最好资源。在mBlock的扩展中心可以看到已有扩展的源代码。



“查看源代码”选项打开插件源代码的文件夹。您可以试着修改扩展代码，修改会在下次mBlock启动的时候生效(这可以作为调试的一种手段)。然而，在这里做的任何修改，会在升级mBlock或者执行“清除缓存”指令时丢失。

1 填写基本信息

从此处下载作为模板的Demo扩展项目：

<http://www.mblock.cc/site-images/Demo.zip>

下一步是编辑s2e文件。您需要一个文本编辑器。虽然用系统自带的“记事本”程序也可以，但推荐使用专门用于编辑代码的编辑器，比如Github Atom, Visual Studio Code, Brackets或者Sublime text.

在s2e文件的开头是一些基本信息。您需要告诉用户这个扩展是做什么的，它是由谁撰写的。下面是s2e文件的开头几行：



Tips

.s2e文件是扩展的主要文件。除基本信息外，.s2e文件还定义语句块，告诉mBlock下拉菜单是什么样子，以及完成多语言翻译工作(如果有的话)

如果您熟悉Javascript，您会发现它是一个正常的JSON文件。



2 定义语句块

“blockSpecs”板块告诉mBlock语句块长什么样，并且为Scratch模式和Arduino模式提供重要的信息

```

"blockSpecs": [
  ["h","Demo Program","runArduino"],
  [
    "w",
    "digitalWrite( %n , %d.digital )",
    "digitalWrite",
    "13",
    "HIGH",
    {
      "setup":"pinMode({0},OUTPUT); \n",
      "inc":"","",
      "def":"","",
      "work":"digitalWrite({0},{1});\n",
      "loop":""
    }
  ],
  [
    "w",
    "blink",
    "blink",
    {
      "setup":"","",
      "inc":"#include \"demo.h\"",
      "def":"DemoClass demo; \n",
      "work":"demo.blink(); \n",
      "loop":""
    }
  ]
],

```

每个语句块是由一个 Javascript数组定义的

▼ 语句块类型是用一个字母来表示的:

“h” 代表“头语句”; 它们在自定义语句块中很少使用

“w” 代表“写语句”; 他们向硬件发送写指令, 但却不读取返回数据

“r” 代表“读语句”; 在Scratch模式下, 它会让程序暂停执行来等待一个返回值; “R” 是异步读的意思 – 程序无法直接获得读取的 (传感器) 等返回值, 而是在一段时间之后通过一个函数 (回调函数) 被调用来获得的. “r” 和 “R” 类型语句块看起来一样, 在Arduino模式也没有区别.

“b” 是“布尔指令”的意思; 他们返回“是或否”的二元信息. 和“R”相同“B”是异步读取布尔指令的意思

▼ 在语句块上显示的文字

由“%”开始的部分是参数 – 用户可以在此输入文字或者用其它语句块填充. 该语句块 (“digitalWrite(%n , %d.digital)”) 看起来像这样:

digitalWrite(13 , HIGH)

%n 展现一个可以用来填数字的圆圈, 在Arduino模式下也会输出一个数字.

%d.name 展现一个下拉列表, 在Arduino模式下会转化为一个数字. 下拉列表的内容是由同文件“menus”板块定义的, name是它的标识符.

其它类型的参数包括:

serial write text hello

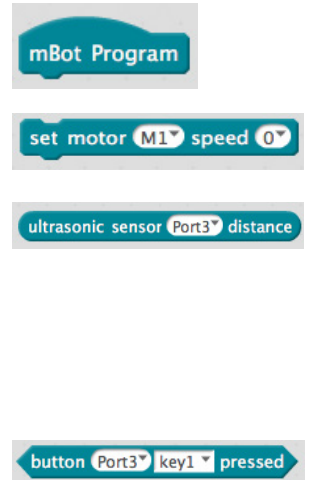
%s 展现一个用来填字符串的方框

ir remote A pressed

%m.name 展现一个方形的下拉列表, 用来选择字符串

touching color ?

%c 展现一个颜色选择器



3 制作菜单

“digitalWrite” 板块有一个菜单来方便选择高低电平。它是这样定义的：



```
"blockSpecs": [
  ["h","Demo Program","runArduino"],
  [
    "w",
    "digitalWrite( %n , %d.digital )",
    "digitalWrite",
    "13",
    "HIGH",
    {
      "setup":"pinMode({0},OUTPUT); \n",
      "inc":"",
      "def":"",
      "work":"digitalWrite({0},{1});\n",
      "loop":""
    }
  ],
  [
    "w",
    "blink",
    "blink",
    {
      "setup":"",
      "inc":"#include \"demo.h\"",
      "def":"DemoClass demo; \n",
      "work":"demo.blink(); \n",
      "loop":""
    }
  ]
],
"menus": {
  "digital":["HIGH","LOW"]
},
"values":{
  "HIGH":1,
  "LOW":0
},

```

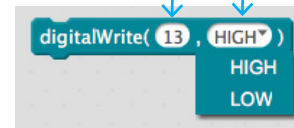
“%d.digital” 表示菜单信息是存储在“menus”列表的“digital”项里的

4 完成语句块定义

命名一个对应到 Scratch模式的 Javascript 函数名字

```
"blockSpecs": [
  ["h","Demo Program","runArduino"],
  [
    "w",
    "digitalWrite( %n , %d.digital )",
    "digitalWrite",
    "13",
    "HIGH",
    {
      "setup":"pinMode({0},OUTPUT); \r\n",
      "inc":"",
      "def":"",
      "work":"digitalWrite({0},{1});\r\n",
      "loop":""
    }
  ],
  [
    "w",
    "blink",
    "blink",
    {
      "setup":"",
      "inc":"#include \"demo.h\"",
      "def":"DemoClass demo; \n",
      "work":"demo.blink(); \n",
      "loop":""
    }
  ]
],

```



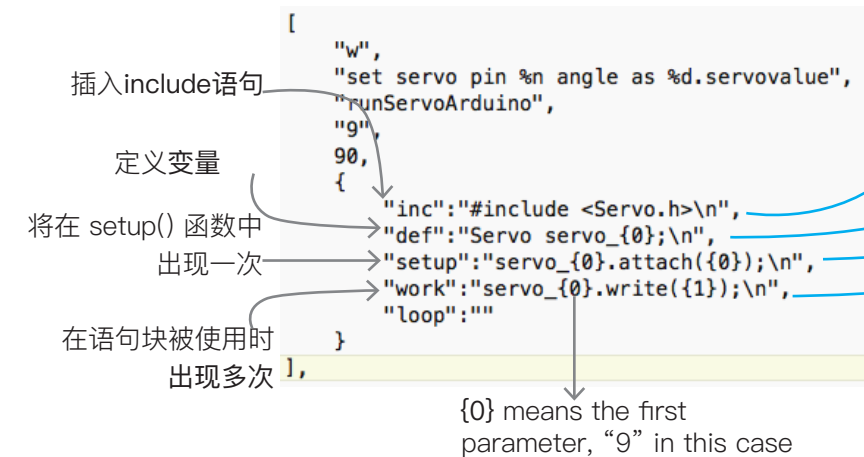
为你刚定义的语句块提供默认值

它代表菜单有两个选项：“HIGH” and “LOW”

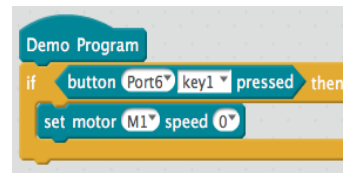
在Arduino模式下，“HIGH”产生数字1，“LOW”产生数字0。Scratch模式不受这个影响

5 生成Arduino代码

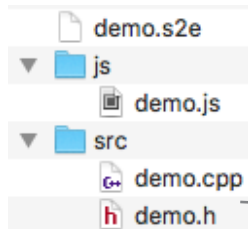
在定义语句块的同时, .s2e 文件也决定了该语句块的Arduino代码是如何生成的。下面是定义的方法:



```
Back Upload to Arduino Edit with Arduino IDE
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include <SoftwareSerial.h>
4
5 #include <Servo.h>
6
7 double angle_rad = PI/180.0;
8 double angle_deg = 180.0/PI;
9 Servo servo_9;
10
11 void setup(){
12   servo_9.attach(9);
13   for(int __i__=0; __i__<10; ++__i__)
14   {
15     servo_9.write(90);
16   }
17 }
18
19 void loop(){
20   _loop();
21 }
22
23 void _delay(float seconds){
24   long endTime = millis() + seconds * 1000;
25   while(millis() < endTime)_loop();
26 }
27
28 void _loop(){
29 }
```



```
Back Upload to Arduino Edit with Arduino IDE
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include <SoftwareSerial.h>
4
5 double angle_rad = PI/180.0;
6 double angle_deg = 180.0/PI;
7 Me4Button buttonSensor_6(6);
8 MeDCMotor motor_9(9);
9
10 void setup(){
11   if((buttonSensor_6.pressed()==1)){
12     motor_9.run(0);
13   }
14 }
15
16 void loop(){
17   _loop();
18 }
19
20 void _delay(float seconds){
21   long endTime = millis() + seconds * 1000;
22   while(millis() < endTime)_loop();
23 }
24
25 void _loop(){
26   buttonSensor_6.pressed();
27 }
```



如果引用了其他Arduino源文件, 请确定已复制到“src”文件夹

6 编程Scratch Mode

Javascript文件很长，建议在Demo.js上做必要的修改。如果你的扩展不支持Scratch模式，可以跳过此步骤(这样语句块在Scratch模式下运行时什么都不会发生)

demo.js

```
// demo.js

(function(ext) {
  var device = null; // 不要变动

  var levels = {
    HIGH:1, // 定义你自己的变量
    LOW:0
  };

  ext.resetAll = function(){}; // 扩展初始化代码

  ext.runArduino = function(){};

  ext.digitalWrite = function(pin, level) { // 在这里写语句块的对应代码
    device.send([pin, levels[level]])
  };

  ext.blink = function(){
    device.send([0x22, 0x23]) // 使用字节数组向串口发送字节数据
  }

  function processData(bytes) { // 每次计算机从串口接受到数据的时候都会调用这个函数
    trace(bytes);
  }

  // Extension API interactions
  var potentialDevices = [];
```

demo.js (结尾)

```
ext._getStatus = function() {
  if(!device) return {status: 1, msg: 'demo disconnected'};
  return {status: 2, msg: 'demo connected'};
}

var descriptor = {};
ScratchExtensions.register('demo', descriptor, ext, {type: 'serial'});
})();
```

改成你的扩展的名字。

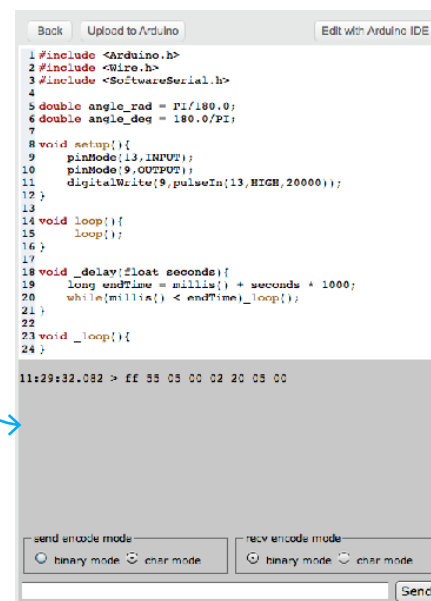
demo.s2e

```
{
  "extensionName": "Demo2",
  "description": "A Demo Extension for Arduino",
  "version": "1.1",
  "author": "Test Author(test@makeblock.com)",
  "homepage": "makeblock.com",
  "sort": 0,
  "javascriptURL": "js/demo.js",
}
```

demo.s2e

```
"blockSpecs": [
  ["h", "Demo Program", "runArduino"],
  [
    "w",
    "digitalWrite( %n, %d.digital )",
    "digitalWrite",
    "13",
    "HIGH",
    {
      "setup": "pinMode({0}, OUTPUT); \n",
      "inc": "",
      "def": "",
      "work": "digitalWrite({0}, {1}); \n",
      "loop": ""
    }
  ],
]
```

函数名称定义在这里



发布扩展

完成扩展之后，你需要把它打包成一个.zip文件。在MacOS下，右键点击文件夹并选择“压缩 xxx...”；在Windows下，右键点击文件夹并选择“发送到”，“压缩文件夹”。

之后就可以通过扩展管理器的“添加扩展”按钮来添加扩展了。但如果将扩展上传到在线扩展中心的话，用户使用起来会更加方便。

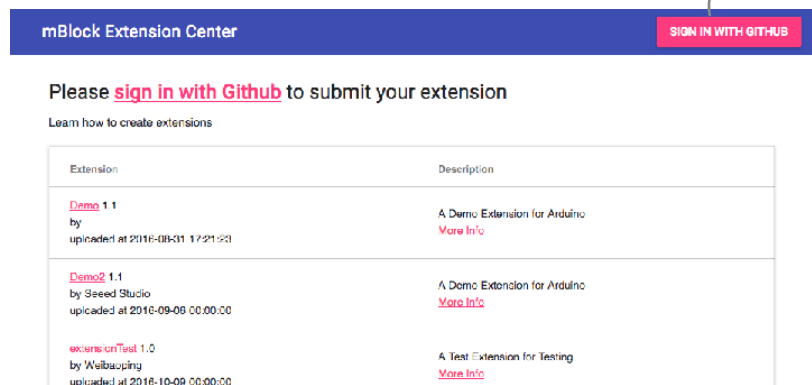
1 使用Github账户登录

在扩展中心网站：

<http://www.mblock.cc/extensions/>

点击“Sign-in with Github”。如果你还没有Github帐号，可以注册一个。

点击此处登录



mBlock Extension Center

[SIGN IN WITH GITHUB](#)

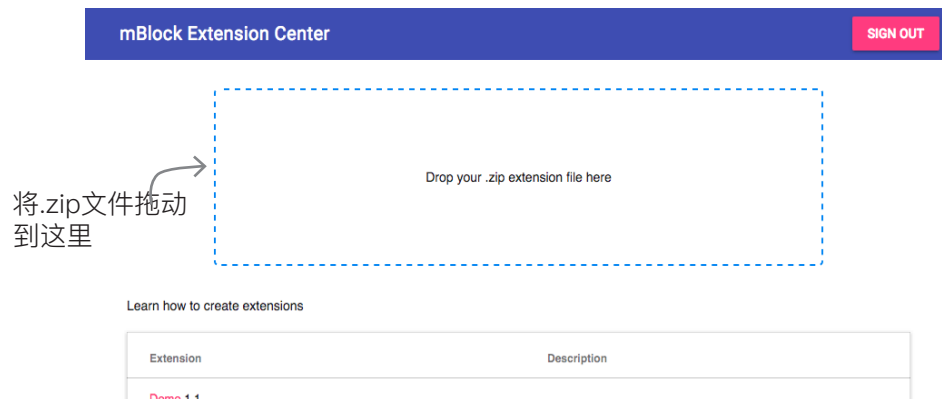
Please [sign in with Github](#) to submit your extension

Learn how to create extensions

Extension	Description
Demo 1.1 by by uploaded at 2016-09-01 17:21:53	A Demo Extension for Arduino More Info
Demo2 1.1 by Seed Studio uploaded at 2016-09-09 00:00:00	A Demo Extension for Arduino More Info
extensionTest 1.0 by Wulabaojun uploaded at 2016-10-09 00:00:00	A Test Extension for Testing More Info

2 上传扩展

登录之后，将打包后的.zip扩展文件拖动到虚线框内（或者点击虚线框来浏览文件）



mBlock Extension Center

[SIGN OUT](#)

Drop your .zip extension file here

Learn how to create extensions

Extension	Description
Demo 1.1	

恭喜！您已经成功上传了扩展，并且为国际mBlock社群做出了贡献！

提示

如果想要更新自己的扩展，只要上传一个版本号更新的就可以了。

提示

Makeblock作为运营mBlock的公司，保留以任何理由删除任何扩展的权利。但是我们欢迎面向各种硬件的各类扩展，并且希望审核工作只针对垃圾信息（包括以“test123”之类命名的扩展）以及有不适合内容的扩展。