GoKit3 二次开发-
代码自动生成工具介绍

机智云

编制人	Ture Zhang	审核人	Andy Gao	批准人	
产品名称		产品型号		文档编号	
会签日期			版本	V0.1.4	

GizWits

修改记录：

修改时间	修改记录	修改人	版本	备注
20160728	初建	TureZhang	0.1.0	
20160728	合并 MCU 与 SOC 介绍	TureZhang	0.1.1	
20160729	修改部分内容	TureZhang	0.1.2	
20160930	二版修改	TureZhang	0.1.3	
20161024	增加配置处理说明	AndyGao	0.1.4	

目录

1. 前文需知

1.1 什么是“代码自动生成工具”

为了降低开发者的开发门槛，缩短开发周期，降低开发资源投入，机智云推出了代码自动生成服务。云端会根据产品定义的数据点生成对应产品的设备端代码。

自动生成的代码实现了机智云通信协议的解析与封包、传感器数据与通信数据的转换逻辑，并封装成了简单的 API，且提供了多种平台的实例代码。当设备收到云端或 APP 端的数据后，程序会将数据转换成对应的事件并通知到应用层，开发者只需要在对应的事件处理逻辑中添加传感器的控制函数，就可以完成产品的开发。

使用自动生成的代码开发产品，就不必再处理协议相关的部分了，开发者可以将节省出来的精力集中在产品的核心功能开发上。

1.2 支持的平台

自动生成服务支持的硬件方案有：独立 MCU 方案、SOC 方案。其中独立 MCU 方案支持的硬件平台有：stm32f103c8x 平台、通用平台（即“其他平台”）；SOC 方案支持的硬件平台有：ESP8266 平台。

MCU 方案与 SOC 方案区别：

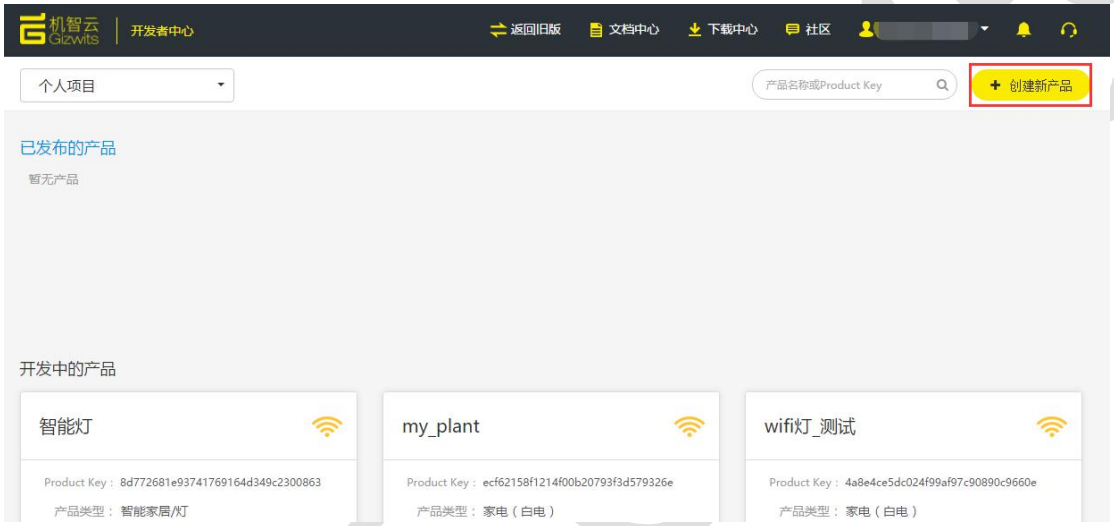
方案	说明
MCU 方案	模组负责与云端信息的交互，通过串口与主控板（即 MCU）进行通信，需要在 MCU 上进行协议解析与外设控制的开发。
SoC 方案	节省一颗 MCU 芯片，利用模组内部资源完成传感器操作和产品逻辑。

MCU 方案中除了支持 **STM32** 平台，还可以将我们生成好的**通用平台版代码**移植到符合条件的任意平台，从而实现机智云所提供的各种功能（详细移植过程请查看《GoKit3 二次开发--通用平台版移植说明》）。

2. “代码自动生成工具”的使用

2.1 创建产品

登录机智云开发者中心（新版）：http://dev.gizwits.com/



点击右上角创建新产品

输入相应的产品信息后点击“保存”。

2.2 添加数据点

添加相应的数据点

添加成功后点击“应用”

2.3 生成目标平台代码

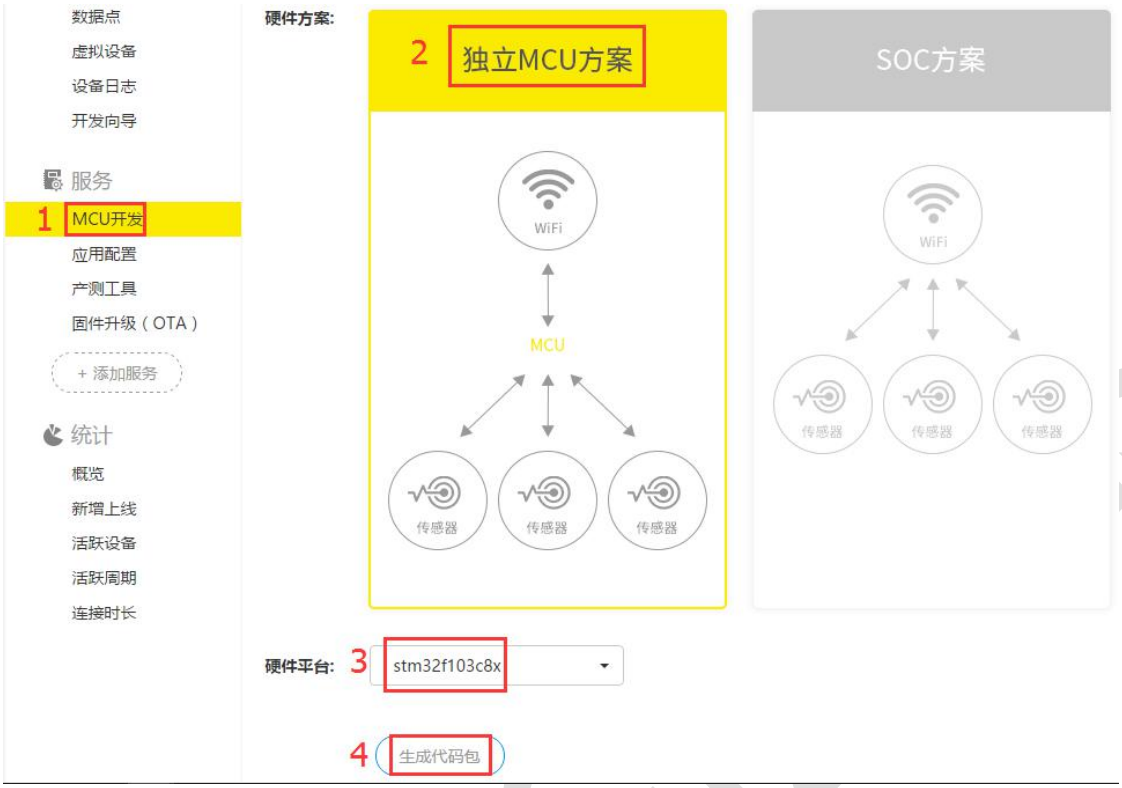
注：如果之前没有定义数据点则无法使用自动生成代码服务。

2.3.1 生成 MCU 方案代码

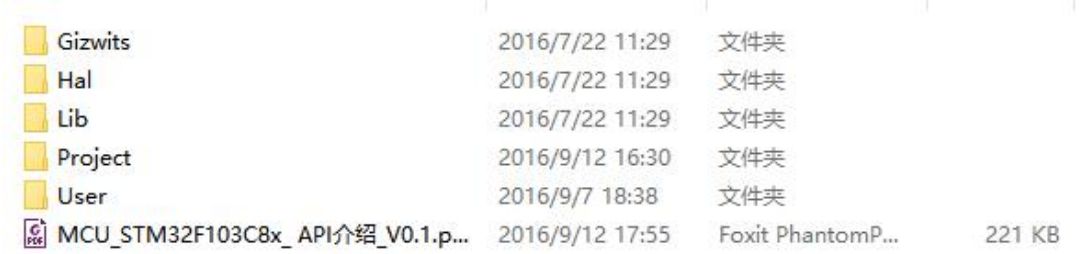
定义好产品后，选择左侧服务中的“MCU 开发”(假设采用的 MCU 是 STM32F103C8x)，选中硬件方案中的“独立 MCU 方案”，再选择“硬件平台”中的“stm32f103c8x”，最后点击“生成代码包”，等待生成完毕下载即可。

注：

如果是其他 MCU 芯片，请选择“其他平台”选项，然后将生成的代码包移植到使用的平台，移植方法参考《GoKit3 二次开发-通用平台版移植说明》。



下载完成后解压如下

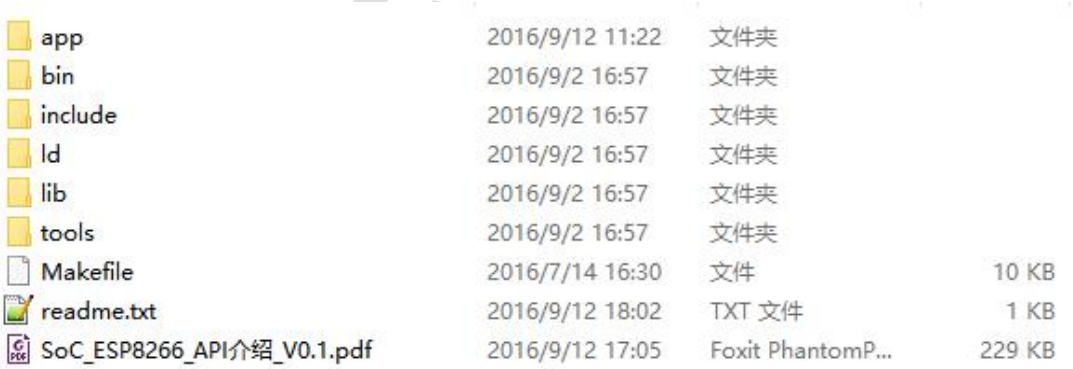


2.3.2 生成 SoC 方案代码

定义好产品后，选择左侧服务中的“SoC 开发”(假设使用的 SoC 芯片是 esp8266)，选中硬件方案中的“SoC 方案”，则选择“硬件平台”中的“esp8266”，最后点击“生成代码包”，等待生成完毕下载即可。


产品信息
基本信息
数据点
虚拟设备
设备日志
开发向导
服务
1
MCU开发
应用配置
产测工具
固件升级（OTA）
+ 添加服务
统计
概览
新增上线
活跃设备
活跃周期
连接时长
MCU开发
硬件方案:
独立MCU方案
2
SOC方案
WiFi
MCU
传感器
硬件平台:
3
esp8266
4
生成代码包

下载完成后解压如下


app 2016/9/12 11:22 文件夹
bin 2016/9/2 16:57 文件夹
include 2016/9/2 16:57 文件夹
ld 2016/9/2 16:57 文件夹
lib 2016/9/2 16:57 文件夹
tools 2016/9/2 16:57 文件夹
Makefile 2016/7/14 16:30 文件 10 KB
readme.txt 2016/9/12 18:02 TXT 文件 1 KB
SoC_ESP8266_API介绍_V0.1.pdf 2016/9/12 17:05 Foxit PhantomP... 229 KB

3. 自动生成代码说明

3.1 STM32 平台文件说明

```
1  ├── Gizwits
2  │   ├── gizwits_product.c ---------->//产品相关处理函数，如：eventProcess()
3  │   ├── gizwits_product.h ---------->//gizwits_product.c 头文件，定义产品软硬件版本号
4  │   ├── gizwits_protocol.c --------->//gizwits协议相关的处理模块，API的封装模块
5  │   └── gizwits_protocol.h --------->//gizwits_protocol.c头文件，产品相关数据结构体
6  ├── Hal
7  │   └── Hal_key
8  │       └── Hal_key.h ------------->//Hal_key.c头文件
9  ├── Lib
10 ├── Project
11 │   ├── gokit_mcu_stm32.uvopt
12 │   └── gokit_mcu_stm32.uvproj ---->//工程管理入口文件
13 ├── README.txt
14 └── User
15     ├── delay.c ------------------->//延时函数
16     ├── delay.h
17     ├── main.c -------------------->//程序入口函数main()所在文件，包括各模块的初始化
18     ├── stm32f10x_conf.h ---------->//stm32平台相关资源管理
19     ├── stm32f10x_it.c ------------>//stm32平台相关中断函数
20     └── stm32f10x_it.h ------------>//stm32f10x_it.c头文件
```

主要文件说明：

文件	说明
gizwits_product.c	该文件为产品相关处理函数，如 gizEventProcess() 平台相关硬件初始化，如串口、定时器等
gizwits_product.h	该文件为 gizwits_product.c 的头文件，存放产品相关宏定义 如： HARDWARE_VERSION 、SOFTWARE_VERSION
gizwits_protocol.c	该文件为 SDK API 接口函数定义文件
gizwits_protocol.h	该文件为 gizwits_protocol.c 对应头文件,相关 API 的接口声明均在此文件中

3.2 ESP8266 平台文件说明

```
├── app
│   ├── driver
│   │   ├── hal_key.c                //按键驱动程序
│   ├── gen_misc.bat
│   ├── gen_misc.sh                  //编译工具，执行./gen_misc.sh
│   ├── Gizwits
│   │   ├── gizwits_product.c        //产品相关的处理函数，如gizEventProcess()
│   │   ├── gizwits_product.h        //gizwits_product.c头文件，主要定义软硬件版本号
│   │   ├── gizwits_protocol.c       //gizwits协议相关的处理模块，API的封装等
│   │   ├── gizwits_protocol.h       //gizwits_protocol.c头文件，包括协议相关结构体，数据点相关结构体等
│   ├── include
│   │   ├── driver
│   │   │   ├── hal_key.h            //hal_key.c头文件
│   │   ├── ssl
│   └── user
│       └── user_main.c              //程序入口函数user_init()所在文件，包括各模块的初始化，task创建等
├── bin
│   ├── at
│   │   ├── 1024+1024
│   │   ├── 512+512
│   │   ├── noboot
│   ├── _temp_by_dltool
│   └── upgrade
│       ├── user1.4096.new.6.bin     //编译生成的执行文件，烧录使用
├── include
│   ├── gagent_external.h            //gagent接口头文件
├── ld
├── lib
│   ├── libgagent.a                  //gagent封装库文件
├── tools
└── 开发指南V0.1.pdf                 //使用说明
```

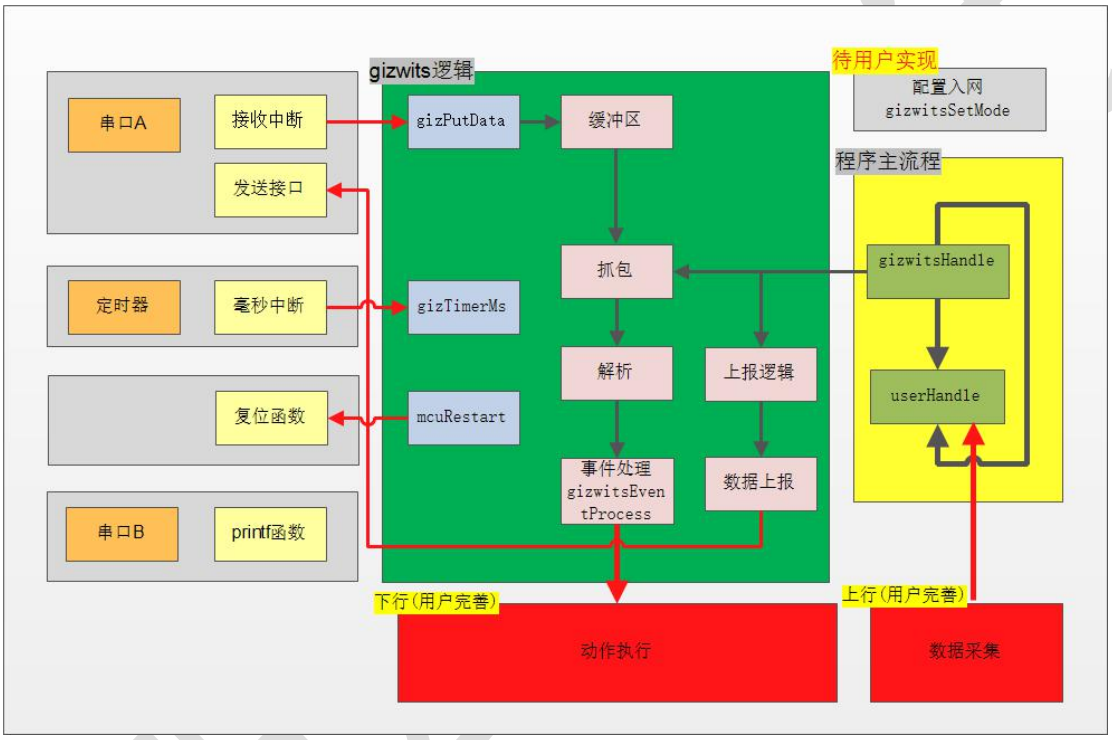
主要文件说明：

文件	说明
libgagent.a	该文件为机智云设备接入协议库文件,文件位于 lib 目录下
gagent_external.h	该文件为 libgagent.a 对应头文件,两个文件配合使用
gizwits_product.c	该文件为平台相关处理文件，存放事件处理 API 接口函数，即 ***gizwitsEventProcess***()
gizwits_product.h	该文件为 gizwits_product.c 的头文件，存放产品相关宏定义如： HARDWARE_VERSION 、SOFTWARE_VERSION
gizwits_protocol.c	该文件为协议实现文件，存放 SDK API 接口函数
gizwits_protocol.h	该文件为 gizwits_protocol.c 对应头文件，协议相关宏定义即 API 接口声明均在此文件中。

4. 二次开发介绍

4.1 代码二次开发需知

自动生成的代码已经根据用户定义的产品数据点信息，并针对 STM32、ESP8266 等平台，生成了对应的机智云串口协议层代码，用户只需要调用相应的 API 接口或添加相应的逻辑处理即可。代码框架如下图所示：



需要开发的部分为：

A. 下行处理：例如 LED 灯开关、电机转速控制等。

B. 上行处理：例如温湿度数据采集，红外传感器状态获取等。

C. 配置处理：配置入网及恢复出厂设置。

4.2 二次开发举例

自动生成的代码在各平台之间使用了统一的协议封装，故二次开发所完成的修改也几乎相同，下面以 **STM32 平台**为例。

4.2.1 下行处理

首先要完成的是传感器驱动开发，然后在 **Gizwits 目录**下的 **gizwits_product.c** 文件中的 **gizwitsEventProcess()**函数中处理相应事件即可（如下例中的 **ledRgbControl()**，功能是控制 RGB 灯的颜色）。

下面以控制 RGB LED 为例，代码示例如下：

修改前：

```
if(0x01 == currentDataPoint.valueLED_ONOFF)
{
  //user handle
}
else
{
  //user handle
}
break;
```

修改后：

```
if(0x01 == currentDataPoint.valueLED_ONOFF)
{
  //user handle
    ledRgbControl(254,0,0);
}
else
{
  //user handle
    ledRgbControl(0,0,0);
}
break;
```

4.2.2 上行处理

首先要完成的是传感器驱动开发，然后在 **user 目录**下 **main.c** 文件中的 **userHandle()**函数中实现传感器数据采集，用户只需并将采集到的数值赋值给对应**用户区的设备状态结构体数据位**即可（如下例中的：**currentDataPoint.valueInfrared = irHandle();**）。

下面以红外传感器的数据获取为例（**只读型数据点**的操作会被云端自动生成），如下：

修改前：

```
void userHandle(void)
{
    /*
    currentDataPoint.valueInfrared = ;//Add Sensor Data Collection
    */
}
```

修改后：

```
void userHandle(void)
{
    currentDataPoint.valueInfrared = irHandle();
}
```

特别提醒：

userHandle()被 while 循环调用，执行速度较快，需要针对不同的需求，用户可调整数据点数据的采集周期和接口实现位置，预防由于传感器数据采集过快引发的不必要的问题。但不建议在 userHandle()中调用延时函数来降低执行频率。正确方法如下：

```
void userHandle(void)
{
    static uint32_t irLastTimer = 0;

    if((gizGetTimerCount()-irLastTimer ) > SAMPLING_TIME_MAX)
    {
       currentDataPoint.valueInfrared = irHandle();

       irLastTimer = gizGetTimerCount();
    }
}
```

4.2.3 配置处理

除了数据的上行与下行处理外，还需要一些配置操作需要完成，如下：

1、配置入网

2、恢复出厂配置

我们提供了一个 API 接口，实现上述操作，定义如下。

int32_t gizwitsSetMode(uint8_t mode)

参数 mode 支持 WIFI_RESET_MODE、WIFI_SOFTAP_MODE、WIFI_AIRLINK_MODE 三种(详情见 gizwits_protocol.h 文件的 WIFI_MODE_TYPE_T)，分别完成恢复出厂配置，进入 softap 配置模式，进入 airlink 配置模式操作。您可以根据产品的定义实现不同的配置操作，

如按键触发进入配置模式。

5. 相关支持

1) 如果您是开发者

GoKit 是面向智能硬件开发者限量免费开放，注册我们的论坛或关注我们的官方微信均可发起申请即可。

开发者论坛： http://club.gizwits.com/forum.php

文档中心：http://docs.gizwits.com/hc/

2) 如果您是团体

GizWits 针对团体有很多支持计划，您可以和 GizWtis 联系，快速得到 GoKit 以及技术支持；

网站地址：http://www.gizwits.com/about-us

官方二维码：