分散加载+增量固件升级方案

Flash内存映射：



Bootloader：

存放在flash起始处的一块空间中，用于接收上位机发送的升级包对对应模块进行升级。

V0.1，升级包内容直接为程序内容，首先写入ram中的一个buffer内，合适时机将buffer内的数据考入对应的flash区域中。

V0.2，升级包内容为mdcd算法生成的固件包生成指令，首先根据指令在buffer内构建固件，然后将构建完成的固件写入对应的flash区域中。如果模块大小大于buffer的大小，将模块按buffer大小进行分割，分别计算固件生成命令。

V0.3 升级包内容为新算法生成的固件包升级指令，可以直接根据指令在flash中进行固件更新，并且不需要外的flash和内存空间，不受此限制。

Entry

Entry应用程序的入口模块，程序的中断向量表，初始化过程在该模块内，该模块位于应用程序所处flash的起始位置。

Entry模块中可以调用其他模块所暴露的函数，但是该模块不向外部暴露函数，没有函数跳转表。

Module\_desc

Module\_desc模块紧接着entry模块，用于存放模块信息表。模块信息表位于module\_desc模块的顶端位置，表中每一个元素为一个结构体，包含一个对应模块的入口地址，模块版本号，模块大小，模块的crc32校验值。

每进行程序更新时，该模块都会被更新（因为至少被更新模块的版本号发生了变化）

Module

普通模块的代码。模块的顶端存放着模块对外接口的函数表，函数表的地址即模块的起始地址（入口地址）。

说明：

* 需要使用分散加载文件，将每个模块的加载域和执行域分割开。模块之间不能直接引用全局变量，需要设置接口函数完成。
* 每个模块的头文件中声明了接口函数表的类型结构体，当一个模块需要引用另一个模块的接口函数时，首先包含该函数的头文件和应用描述，声明一个接口函数表的变量，然后从模块描述表中取出要引用的模块的入口地址，根据入口地址调用需要的函数。

（如何做到增加/删除模块时，不至于所有模块都要进行升级）

（这是一个问题，如果某个很基础的模块进行了升级，那么可能几乎所有模块都必须进行升级，同时如果该模块的改动很小，其他模块对应的改动也很小时，这样的变动使得升级数据量很大，这时就体现出增量升级的优势了）