零死角玩转STM32

MDK的编译过程及文件类型全解

淘宝：firestm32.taobao.com

论坛： www.firebbs.cn

扫描进入淘宝店铺

主讲内容

MDK的编译过程及文件类型全解

程序的组成、存储与运行

CODE、RO、RW、ZI Data域及堆栈空间

在工程的编译提示输出信息中有一个语句“Program Size：Code=xx RO-data=xx RW-data=xx ZI-data=xx”，它说明了程序各个域的大小，编译后，应用程序中所有具有同一性质的数据(包括代码)被归到一个域，程序在存储或运行的时候，不同的域会呈现不同的状态，这些域的意义如下：

- **Code：**即代码域，它指的是编译器生成的机器指令，这些内容被存储到ROM区。
- **RO-data：**Read Only data，即只读数据域，它指程序中用到的只读数据，这些数据被存储在ROM区，因而程序不能修改其内容。例如C语言中const关键字定义的变量就是典型的RO-data。

MDK的编译过程及文件类型全解

CODE、RO、RW、ZI Data域及堆栈空间

- **RW-data：** Read Write data，即可读写数据域，它指初始化为“非0值”的可读写数据，程序刚运行时，这些数据具有非0的初始值，且运行的时候它们会常驻在RAM区，因而应用程序可以修改其内容。例如C语言中使用定义的全局变量，且定义时赋予“非0值”给该变量进行初始化。
- **ZI-data：** Zero Initialie data，即0初始化数据，它指初始化为“0值”的可读写数据域，它与RW-data的区别是程序刚运行时这些数据初始值全都为0，而后续运行过程与RW-data的性质一样，它们也常驻在RAM区，因而应用程序可以更改其内容。例如C语言中使用定义的全局变量，且定义时赋予“0值”给该变量进行初始化(若定义该变量时没有赋予初始值，编译器会把它当ZI-data来对待，初始化为0)；

MDK的编译过程及文件类型全解

CODE、RO、RW、ZI Data域及堆栈空间

- **ZI-data的栈空间(Stack)及堆空间(Heap)：**在C语言中，函数内部定义的局部变量属于栈空间，进入函数的时候从向栈空间申请内存给局部变量，退出时释放局部变量，归还内存空间。而使用malloc动态分配的变量属于堆空间。在程序中的栈空间和堆空间都是属于ZI-data区域的，这些空间都会被初始值化为0值。编译器给出的ZI-data占用的空间值中包含了堆栈的大小(经实际测试，若程序中完全没有使用malloc动态申请堆空间，编译器会优化，不把堆空间计算在内)。

MDK的编译过程及文件类型全解

CODE、RO、RW、ZI Data域及堆栈空间

综上所述，以程序的组成构件为例，它们所属的区域类别如下表：

程序组件	所属类别
机器代码指令	Code
常量	RO-data
初值非0的全局变量	RW-data
初值为0的全局变量	ZI-data
局部变量	ZI-data栈空间
使用malloc动态分配的空间	ZI-data堆空间

MDK的编译过程及文件类型全解

程序的存储与运行

RW-data和ZI-data它们仅仅是初始值不一样而已，为什么编译器非要把它们区分开？这就涉及到程序的存储状态了，应用程序具有静止状态和运行状态。

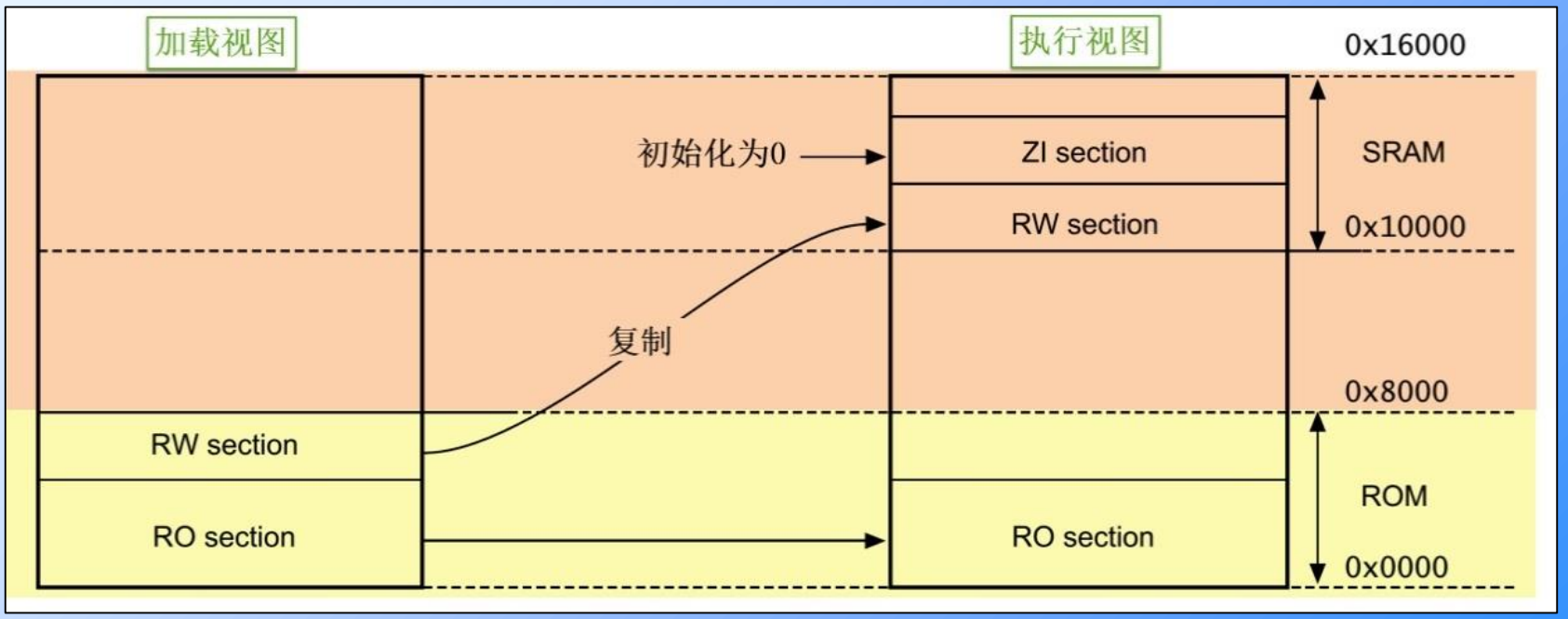
静止态的程序被存储在非易失存储器中，如STM32的内部FLASH，因而系统掉电后也能正常保存。

但是当程序在运行状态的时候，程序常常需要修改一些暂存数据，由于运行速度的要求，这些数据往往存放在内存中(RAM)，掉电后这些数据会丢失。

MDK的编译过程及文件类型全解

程序的存储与运行

因此，程序在静止与运行的时候它在存储器中的表现是不一样的，如下图：



MDK的编译过程及文件类型全解

程序的存储与运行

图中的左侧是应用程序的存储状态，右侧是运行状态，而上方是RAM存储器区域，下方是ROM存储器区域。

程序在存储状态时，RO节(RO section)及RW节都被保存在ROM区。当程序开始运行时，内核直接从ROM中读取代码，并且在执行主体代码前，会先执行一段加载代码，它把RW节数据从ROM复制到RAM， 并且在RAM加入ZI节，ZI节的数据都被初始化为0。加载完后RAM区准备完毕，正式开始执行主体程序。

编译生成的RW-data的数据属于图中的RW节，ZI-data的数据属于图中的ZI节。是否需要掉电保存，这就是把RW-data与ZI-data区别开来的原因，因为在RAM创建数据的时候，默认值为0，但如果有的数据要求初值非0，那就需要使用ROM记录该初始值，运行时再复制到RAM。

MDK的编译过程及文件类型全解

程序的存储与运行

STM32的RO区域不需要加载到SRAM，内核直接从FLASH读取指令运行。计算机系统的应用程序运行过程很类似，不过计算机系统的程序在存储状态时位于硬盘，执行的时候甚至会把上述的RO区域(代码、只读数据)加载到内存，加快运行速度，还有虚拟内存管理单元(MMU)辅助加载数据，使得可以运行比物理内存还大的应用程序。而STM32没有MMU，所以无法支持Linux和Windows系统。

当程序存储到STM32芯片的内部FLASH时(即ROM区)，它占用的空间是Code、RO-data及RW-data的总和，所以如果这些内容比STM32芯片的FLASH空间大，程序就无法被正常保存了。当程序在执行的时候，需要占用内部SRAM空间(即RAM区)，占用的空间包括RW-data和ZI-data。

MDK的编译过程及文件类型全解

程序的存储与运行

应用程序在各个状态时各区域的组成如下表：

程序状态与区域	组成
程序执行时的只读区域(RO)	Code + RO data
程序执行时的可读写区域(RW)	RW data + ZI data
程序存储时占用的ROM区	Code + RO data + RW data

在MDK中，我们建立的工程一般会选择芯片型号，选择后就有确定的FLASH及SRAM大小，若代码超出了芯片的存储器的极限，编译器会提示错误，这时就需要裁剪程序了，裁剪时可针对超出的区域来优化。

THANKS

论坛： www.firebbs.cn

淘宝：firestm32.taobao.com

扫描进入淘宝店铺