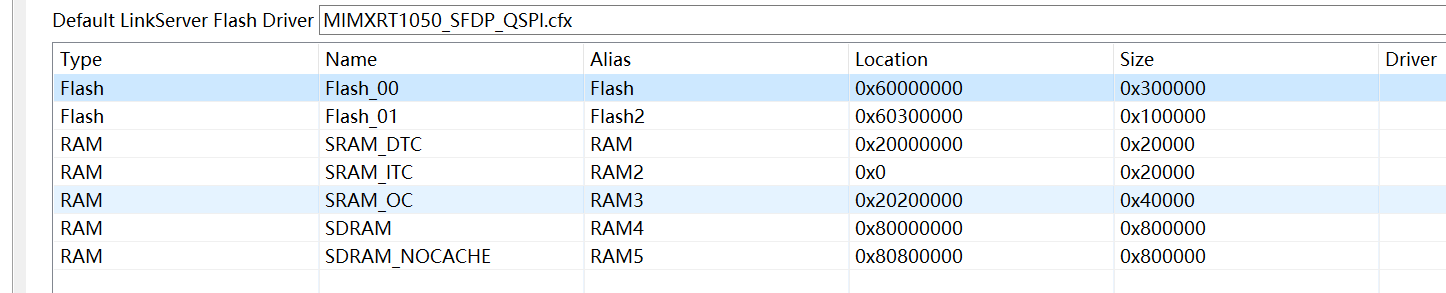
# 8.优化

### 方式之一 把函数、变量放在内存中



列表看一下各个存储器的时钟频率

|  |  |
| --- | --- |
| 存储器 | 时钟频率 |
| SRAM\_DTC | 600M |
| SRAM\_ITC | 600M |
| SRAM\_OC | 150M |
| SDRAM | 166M |
| FLASH | 133M |

ITCM：指令紧耦合RAM。与内核同频运行，通过64-bit的I-TCM总线访问，用来存放指令

DTCM：数据紧耦合RAM。与内核同频运行，通过2\*32-bit的D0/D1-TCM总线访问，用来存放只有内核访问的数据

OCRAM：片上RAM。以1/4内核频率运行，连接到总线矩阵SIM*M7的NIC接口，由于这个特性，OCRAM可以被连接到SIM*M7的其他master访问。比如DMA。当打开L1 Cache时，OCRAM会通过L1 Cache访问以加快访问速度。

所以说哪部分的存储器速度最快？DTC和ITC。那有没有什么办法可以把代码和变量放在这俩块地方呢？

编译器提供了一个指令\_\_attribute\_\_来控制编译，例如

\_\_attribute\_\_((aligned(n))) //设置n字节对齐  
\_\_attribute\_\_((section(".ramfunc.$SRAM\_ITC")))//将函数链接到.ramfunc.$SRAM\_ITC区

用法：

\_\_attribute\_\_((section(".bss.$SRAM\_ITC"))) uint32\_t UserVectors[256] \_\_attribute\_\_((aligned(1024)));

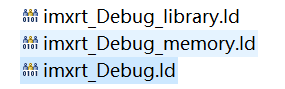
board.h中提供了这些宏定义，可字节使用

bss是未初始化的变量，在启动的时候会同意初始化为0

data是初始化了的变量

ramfunc就是在内存中运行的函数

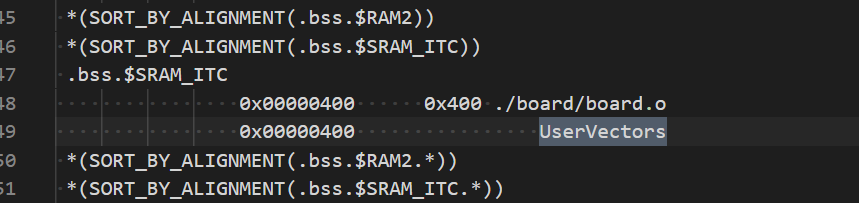
#define ALIGN(n) \_\_attribute\_\_((aligned(n)))  
  
#define RAMFUNC\_ITC \_\_attribute\_\_((section(".ramfunc.$SRAM\_ITC")))  
#define RAMFUNC\_OC \_\_attribute\_\_((section(".ramfunc.$SRAM\_OC")))  
#define RAMFUNC\_SDRAM \_\_attribute\_\_((section(".ramfunc.$SRAM\_SDRAM")))  
#define RAMFUNC\_SDRAM\_NOCACHE \_\_attribute\_\_((section(".ramfunc.$SDRAM\_NOCACHE")))  
  
#define DATA\_DTC \_\_attribute\_\_((section(".data.$SRAM\_DTC")))  
#define DATA\_OC \_\_attribute\_\_((section(".data.$SRAM\_OC")))  
#define DATA\_SDRAM \_\_attribute\_\_((section(".data.$SDRAM")))  
#define DATA\_SDRAM\_NOCACHE \_\_attribute\_\_((section(".data.$SDRAM\_NOCACHE")))  
  
#define BSS\_DTC \_\_attribute\_\_((section(".bss.$SRAM\_DTC")))  
#define BSS\_OC \_\_attribute\_\_((section(".bss.$SRAM\_OC")))  
#define BSS\_SDRAM \_\_attribute\_\_((section(".bss.$SDRAM")))  
#define BSS\_SDRAM\_NOCACHE \_\_attribute\_\_((section(".bss.$SDRAM\_NOCACHE")))

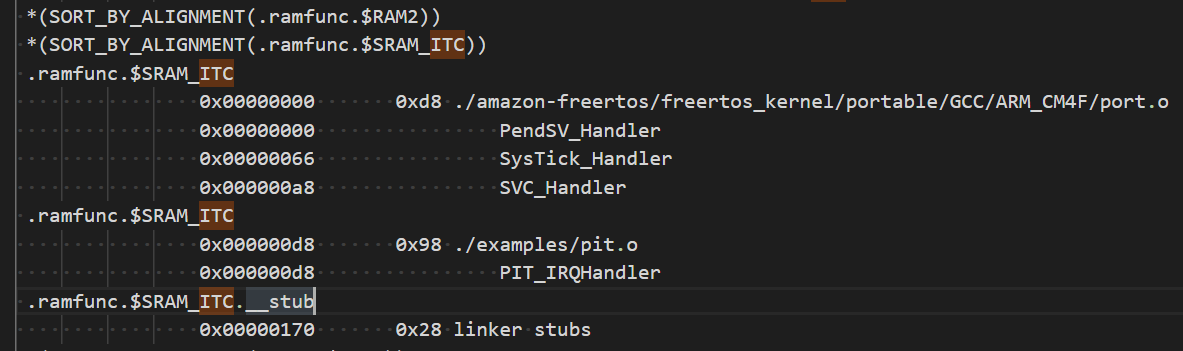
至于为什么双引号内的.data.$SRAM\_DTC指的就是DTCM块，那是由ide自动生成的ld链接脚本控制的。

不过链接到哪里，东西都保存在flash中，启动之后是如何跑到内存当中的呢？在startup/startup\_mimxrt1052.cpp中有相关拷贝数据的代码。

以上面UserVectors的例子说明一下就是，attribute指令告诉编译器链接到Itc区,ld脚本告诉编译器itc区在0x0-0x20000，然后单片机启动之后，在startup\_mimxrt1052.cpp执行的启动代码会把flash中属于itc的数据拷贝到itc中。

怎么知道有没有正确的链接呢？在Debug/xxx.map中会记录各个变量函数的链接情况

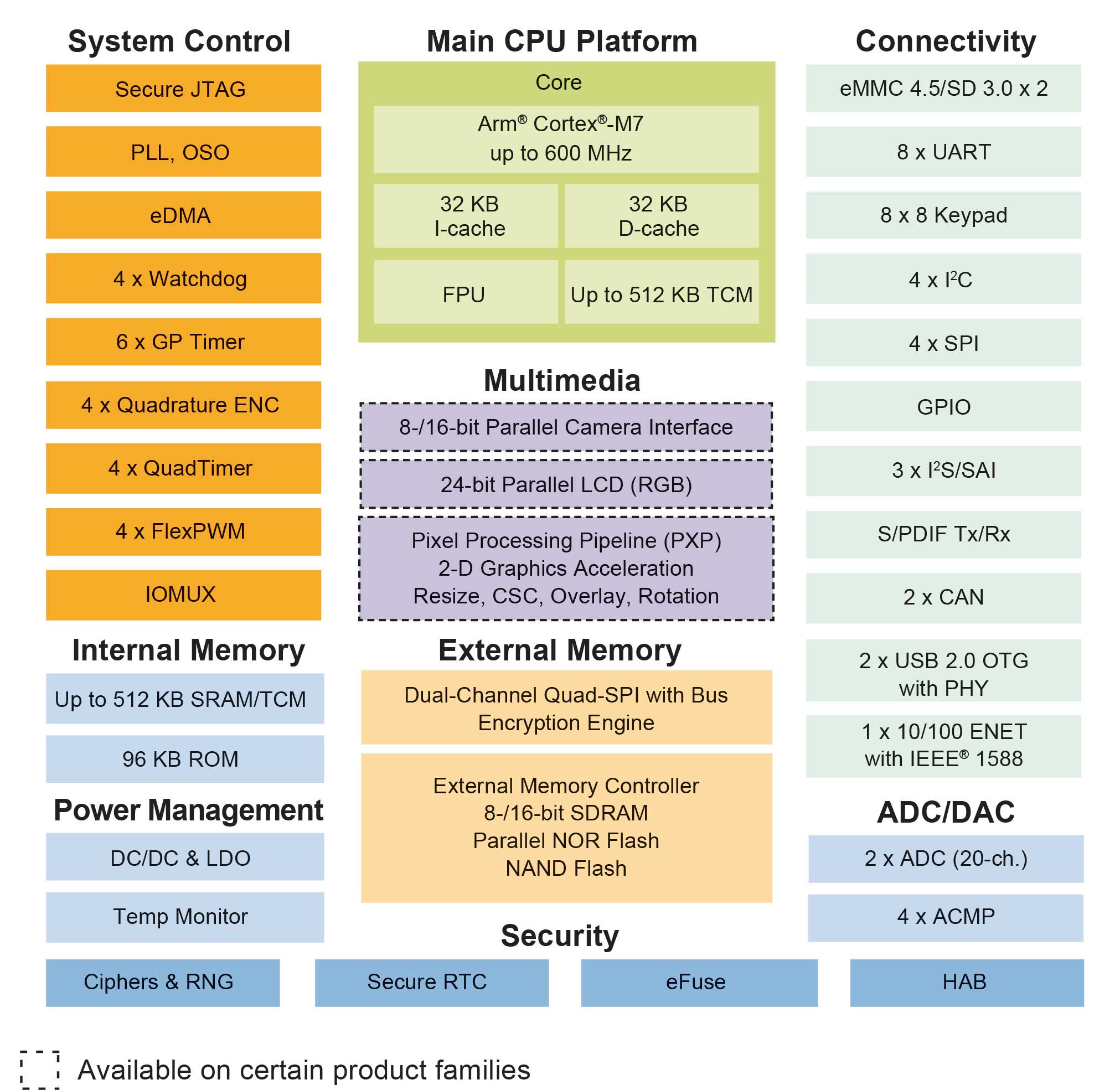




还有就是如果要调试在内存中的函数的话，断电类型得是硬件断点才能生效，否则会出现无法调试的状况。

Flash因为不能同时读上面的指令和读写上面的数据，说以flash底层的读写都是链接到内存当中的。注意itcm和dtcm不能被dma访问

### 方式之二 利用cache加速



核心部分有i-cache和d-cache，分别是指令和数据的高速缓存，启用这俩部分的话会加速经常使用的代码和变量，避免cpu频繁读写低速的内存。

在board.h中提供了BOARD\_ConfigMPU();函数用于启用Cache。

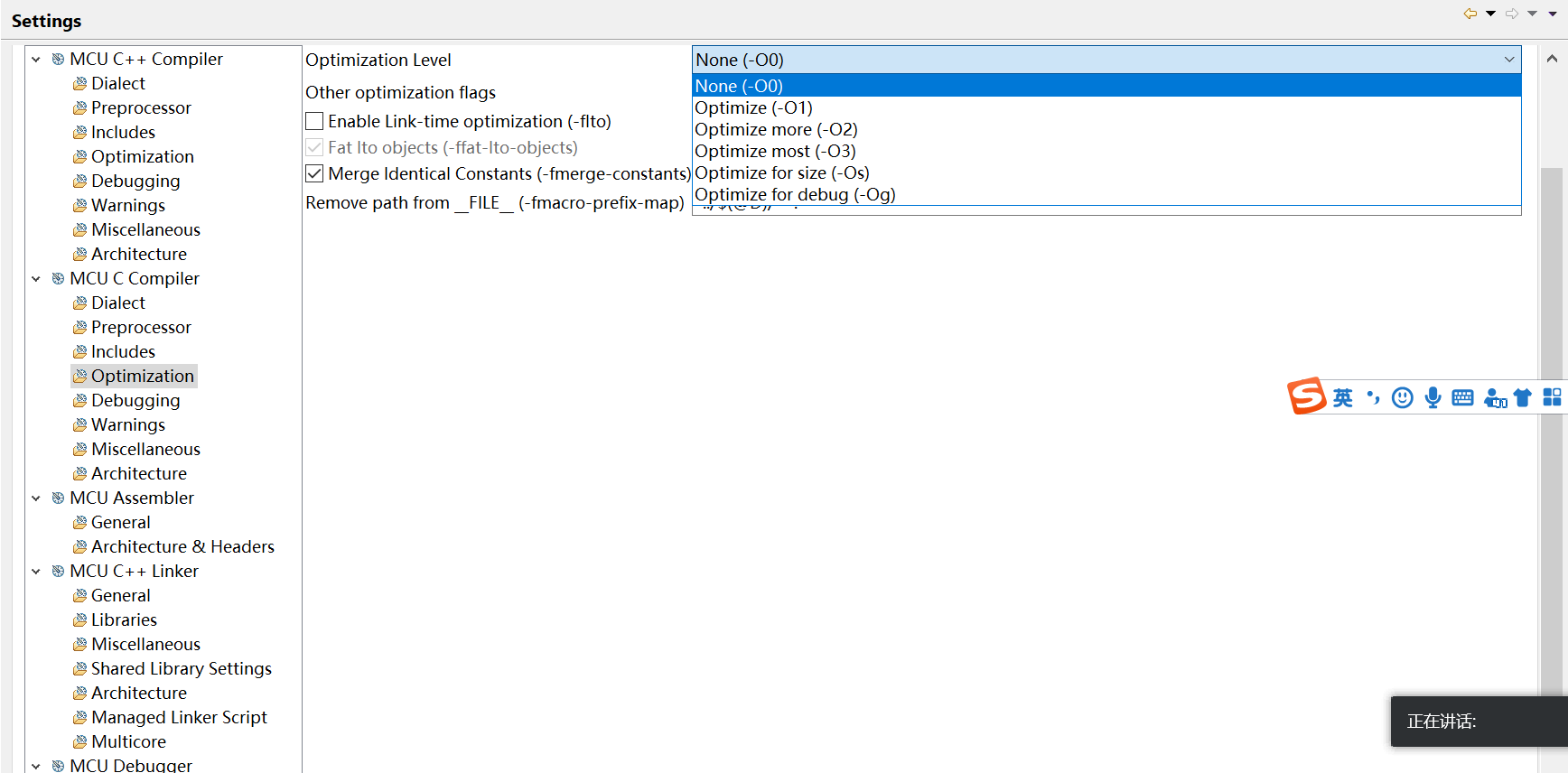
还要注意图像数组这些超大的数据必须不能放在启用了cache的内存上，这样可以避免cache就会被这些数组占满其他变量就没法命中。

启用cache还有**内存一致性**的问题，比如Dma传完数据，cpu去访问这个数据，cpu先看cache中有没有缓存到，有的话直接从cache读取，那么读取的数据就不是dma传输完的数据。可能出现不一致的问题时，要先调用fsl*cache.h中的L1CACHE*InvalidateDCache或者L1CACHE\_InvalidateDCacheByRange使缓存中的数据失效。

16m内存分为了等大小的俩块，一块启用cache，一块没有启用cache，后者用来保存大数组和一些不需要cache的数据。

### 方式之三 编译器优化

这是通用的方法，对于任意程序都可以使用。优化之后，可以做到下标访问和指针访问一样快。

在这里

可以选择优化等级，默认的是没有优化，程序没有bug之后尝试打开编译器优化。