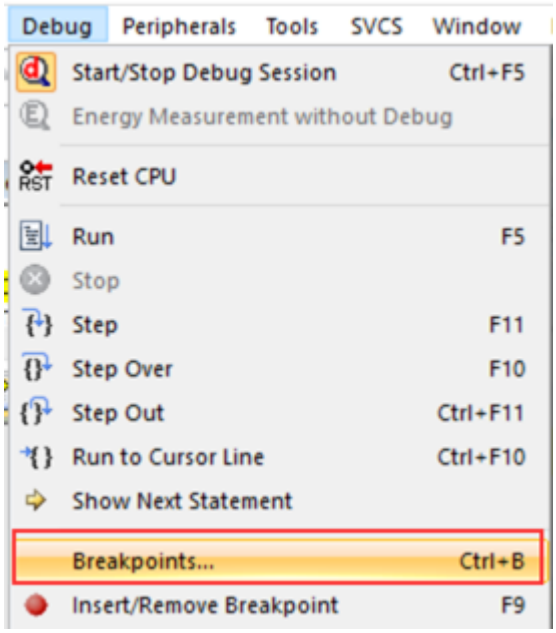
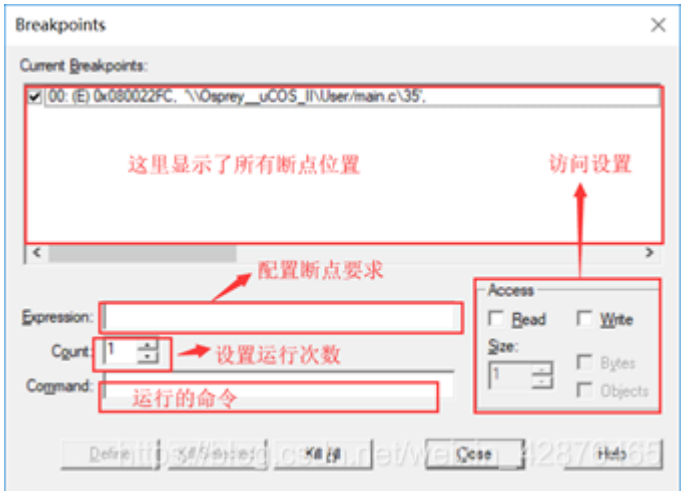
KEIL 的斷點偵錯窗口！

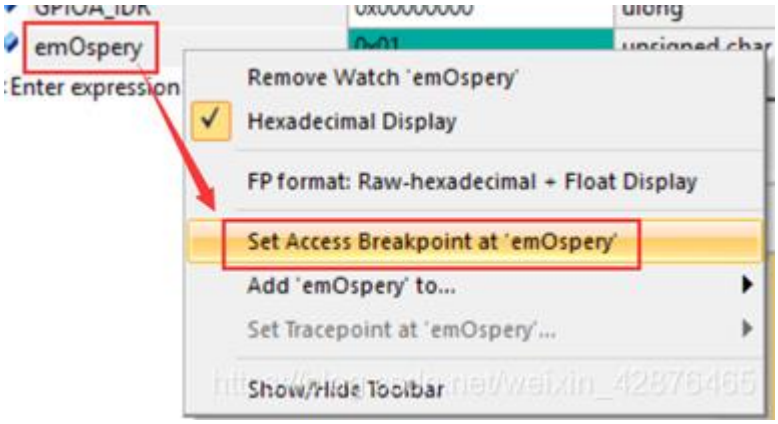
首先打開資料 `Breakpoint` 的窗口：



快速鍵是 `Ctrl + B`。
可以看到如下窗口：

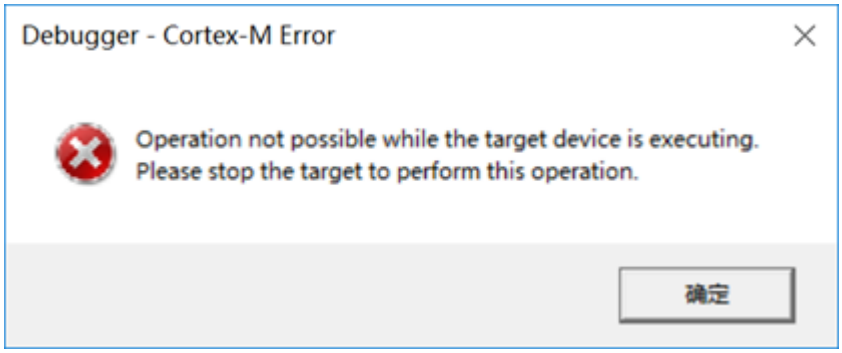


當然你也可以通過下面這種方式打開並設定：



從這裡你會發現，其實這個窗口就是用來管理你設定的斷點的。平常使用的設定斷點方法只是其中的一種特例罷了。

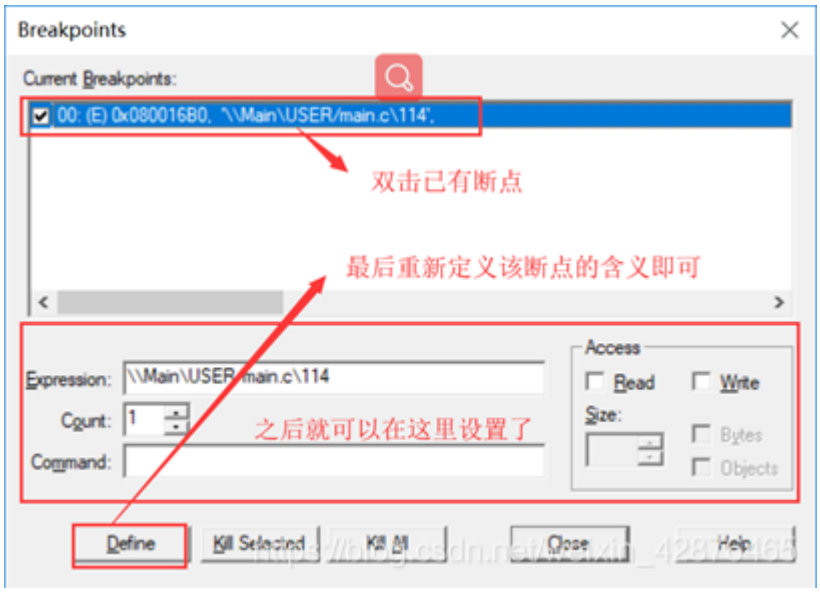
首先要知道的就是，偵錯程式支援的斷點數量是有限的，具體有多少視情況而定，一旦 KEIL 警告你設定斷點太多，那麼就要刪除一些斷點了：



常規用法

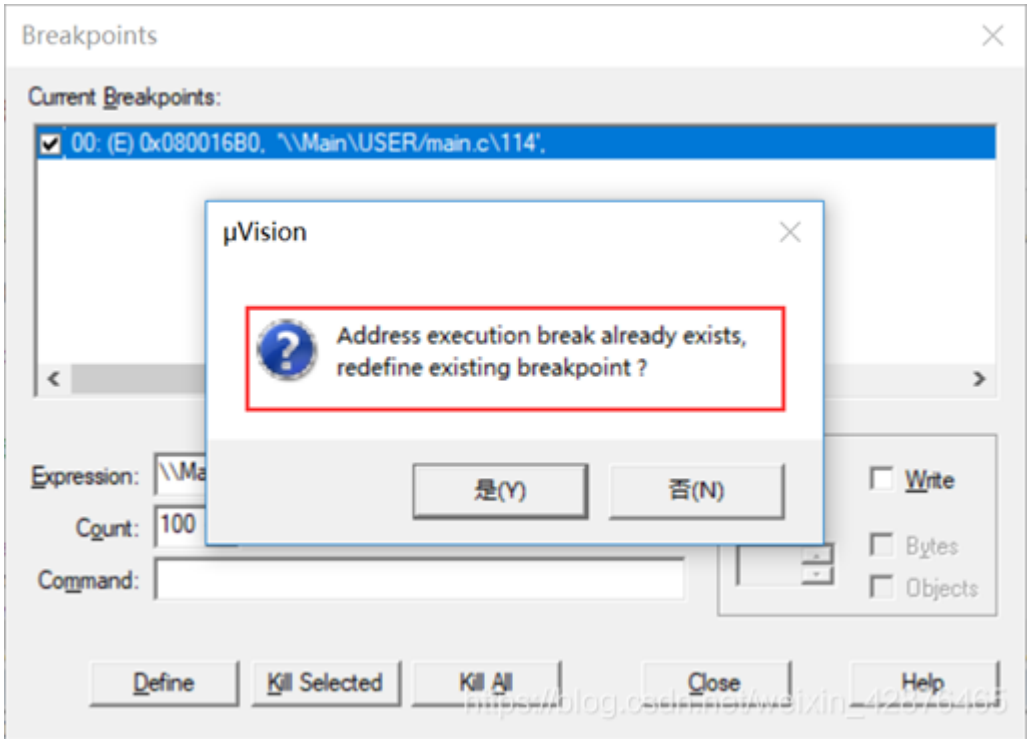
1、程式碼位置運行次數

有些時候我們想知道某些程式碼的運行次數，比如進入中斷處理函數的次數，尋常的斷點設定方式必然會讓程序停止在中斷程序中，但有些時候我們並不希望它停下來。這個時候，你只需要打開該窗口，找到已有的對應斷點位置，連按兩下之後就可以看到類似下面的窗口：



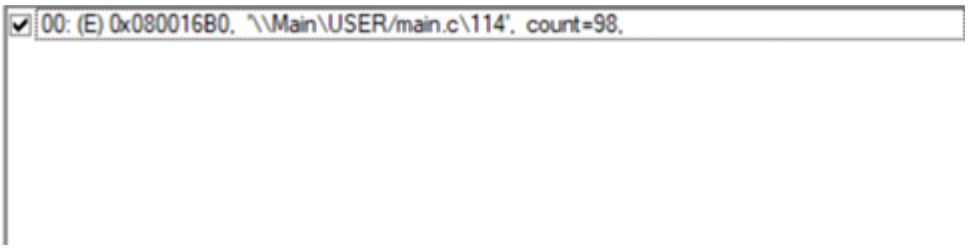
此時，你將 `Count` 的值設定的儘可能大一些，那麼就可以讓程式執行多次之後才停止。

比如我們設定 `Count` 的值為 `100` 次，那麼必須在該程式碼位置運行 `100` 次才會讓程序暫停。當你設定完後點選【`Define`】後，就會詢問你是否需要重新定義，你選擇“是”即可。



這樣你的斷點變成了這樣：

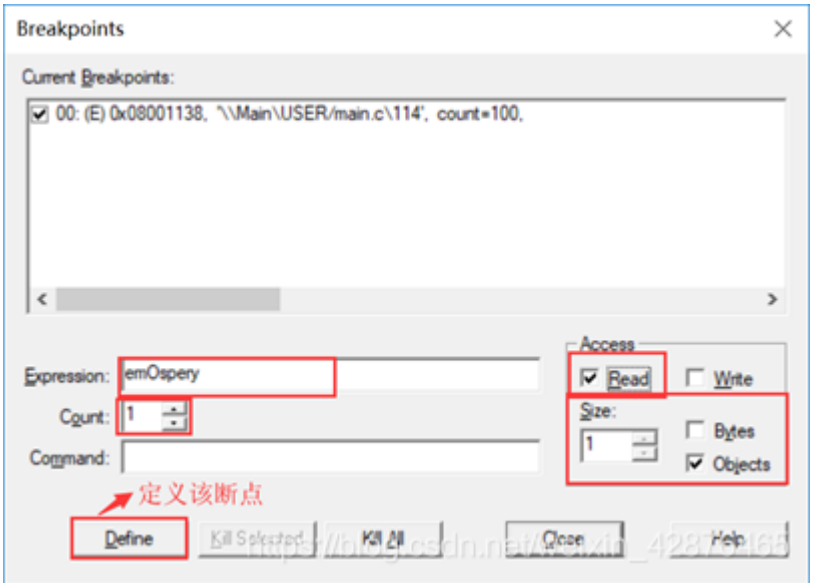
後面的 count=100 表示剩餘運行次數為 100，運行 100 次後將停止程序。前面的 00 代表斷點號，E 代表這是一個執行斷點，0x080016B0 代表程式碼地址，後面的是原始碼位置。



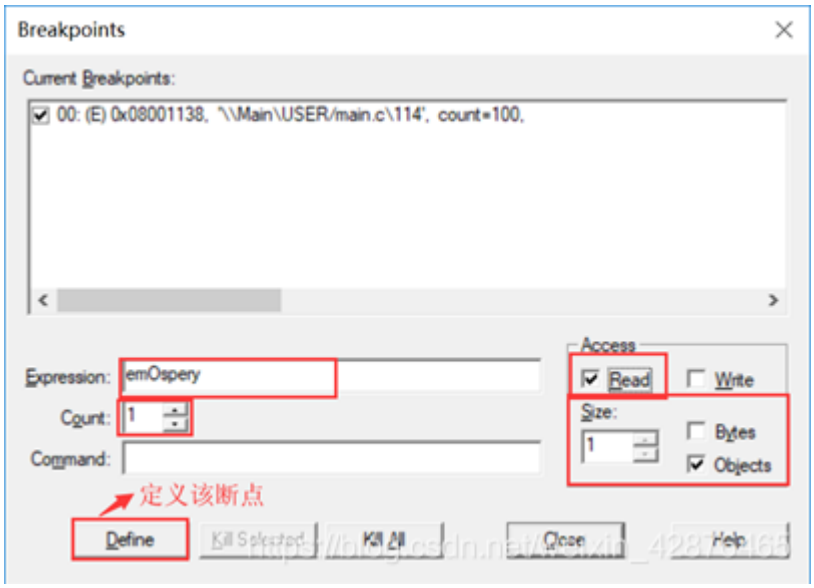
當這個斷點位置運行了 2 次，重新打開該窗口（刷新資料），發現這個數變成了 98，從而可以推算出，已經運行了多少了。如果說你想讓這段程式碼運行 2 次後停止，那麼你只需要一開始設定 Count 的值為 2 即可。

2、資料訪問

有些時候我們需要知道一些變數會在哪裡被訪問，那麼你可以設定該變數的訪問條件。比如魚鷹想知道 emOspery 變數會在哪裡被讀取？那麼你只需設定如下：



定義之後就是這樣：



因為 Count 值設定為 1，所以每一次讀取 emOspery 的操作都將使程序停止。比如這段程式碼：

```
//    // 延时一段时间，不能连续传输太多！否则只能重新.
    delay_ms(10);
    emOspery++;
//    printf("please input number:\n");
//    scanf("%u",&emOspery);
    printf("You input the number is %u\n",emOspery);
  }
```

還有後面的列印函數也使用 emOsprey 變數，所以也會導致程式執行停止。可能你會感到奇怪，為什麼 emOsprey++這樣的操作也會涉及到讀取？事實上你理解了 CPU 暫存器存在的意義也就明白了。

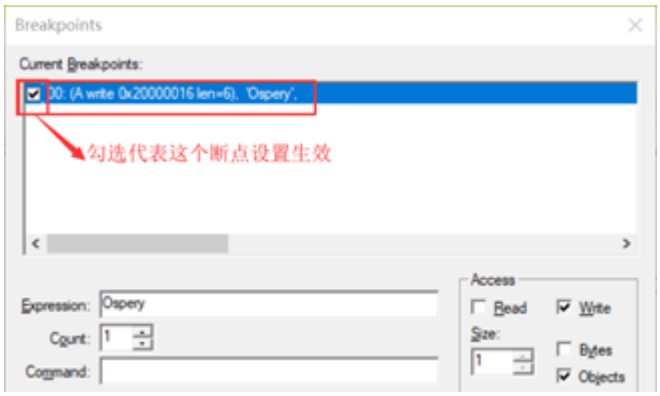
而當你設定為寫（Write）訪問時，你會發現從復位程序開始運行後，程序會停止在某個地方，這是為什麼？當你知道全域變數會在進入 main 函數之前被初始化時，你也就明白為什麼了。

在這裡我們選擇使用 Objects 訪問，即按整個變數對象進行訪問，上面的 emOsprey 變數實際上是 uint16_t，所以 len 為 2，即位元組大小。也就說，如果你設定為 Objects 訪問，那麼它會根據實際的情況設定訪問範圍。

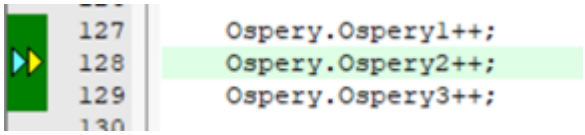
為了更好的說明這一點，我構造一個結構體。

```
typedef struct
{
  uint16_t Ospery1;
  uint16_t Ospery2;
  uint16_t Ospery3;
}Ospery_Typedef;
```

這個結構體大小可以看出是 6 個位元組。
然後設定訪問該結構體的條件：

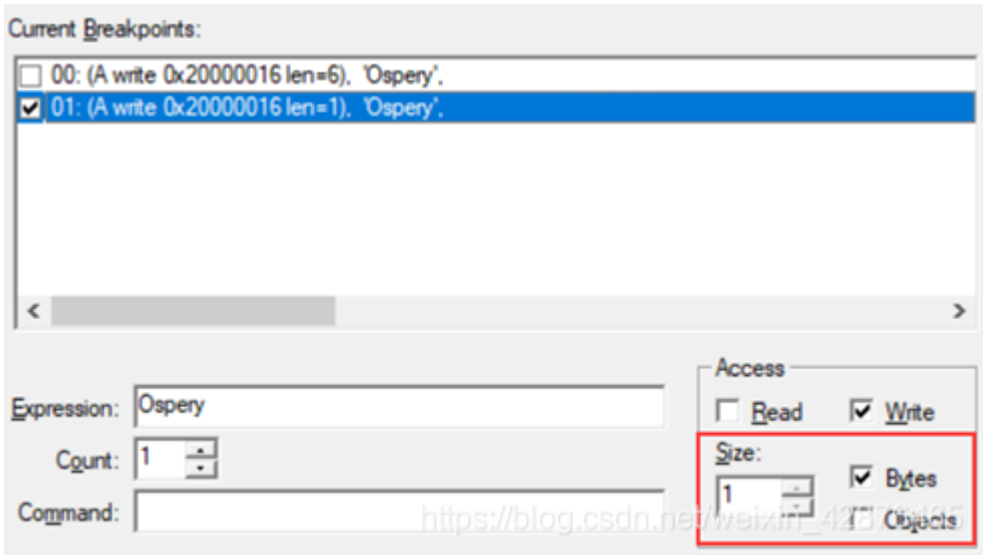


如果我們按 `Objects` 訪問的話，那麼下面的每一條語句都會導致程式執行的停止。

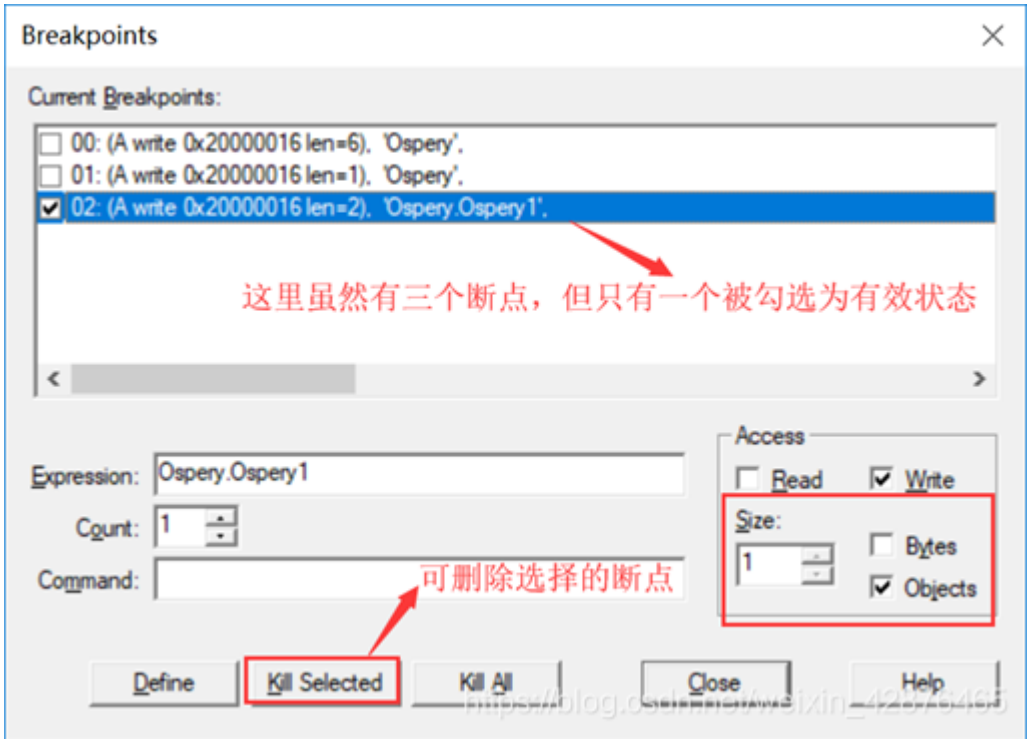


這是因為這些資料都在 `Osprey` 結構體的範圍內（從這裡也可以瞭解到，只要在 `len` 的範圍內的訪問都會導致程序停止運行，所以你可以試試將 `Size` 設定得更大）。

而如果設定為 `Byte` 訪問的話，那麼就只有第一條語句才會導致程序停止運行：

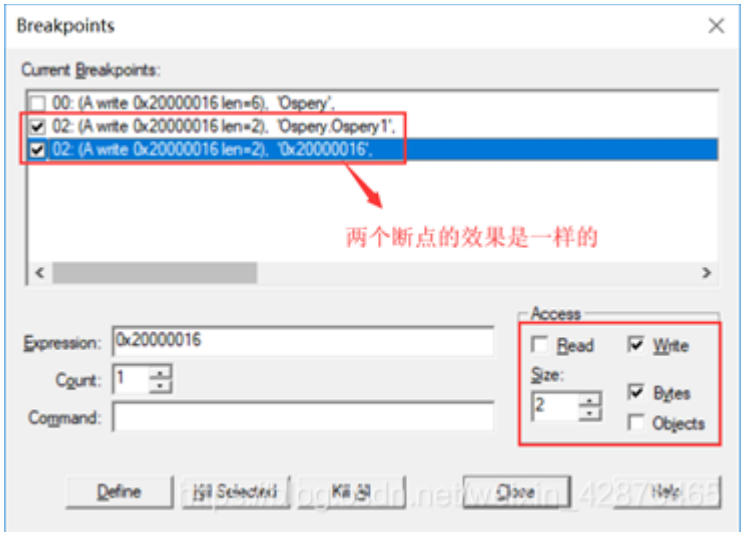


實際上如果你希望只在某個結構體成員變數被訪問時才停止，那麼直接這麼設定就可以：

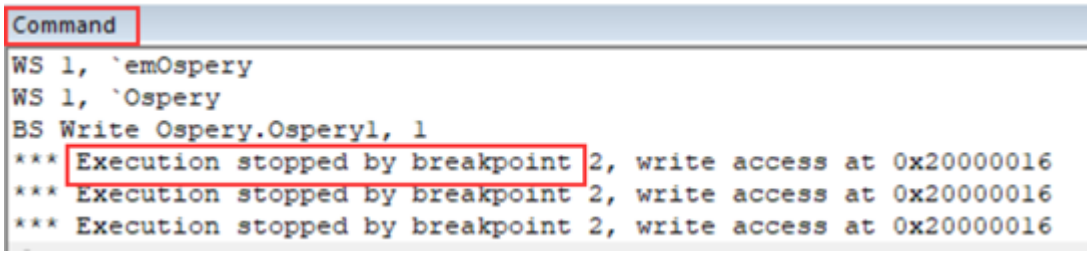


你會發現設定是如此之簡單。

實際上還有一種更為通用的訪問方式，即按地址訪問。
上面可以看出 `Ospery.Ospery1` 成員變數的地址為 `0x2000 0016`（由此我們知道也可以通過這個來看出一個結構體變數的地址是多少）。所以我們可以這樣設定：

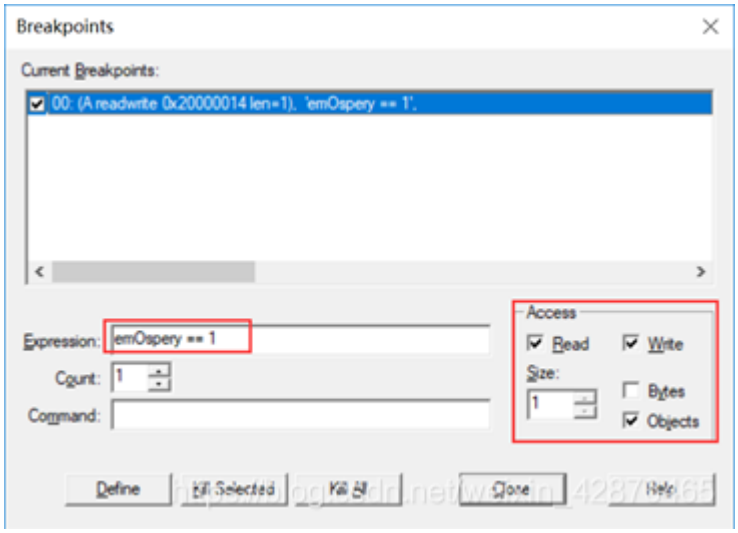


而程式碼位置的斷點設定亦是如此。
斷點太多，怎麼知道程序因何停止？看你的命令窗口就知道了：



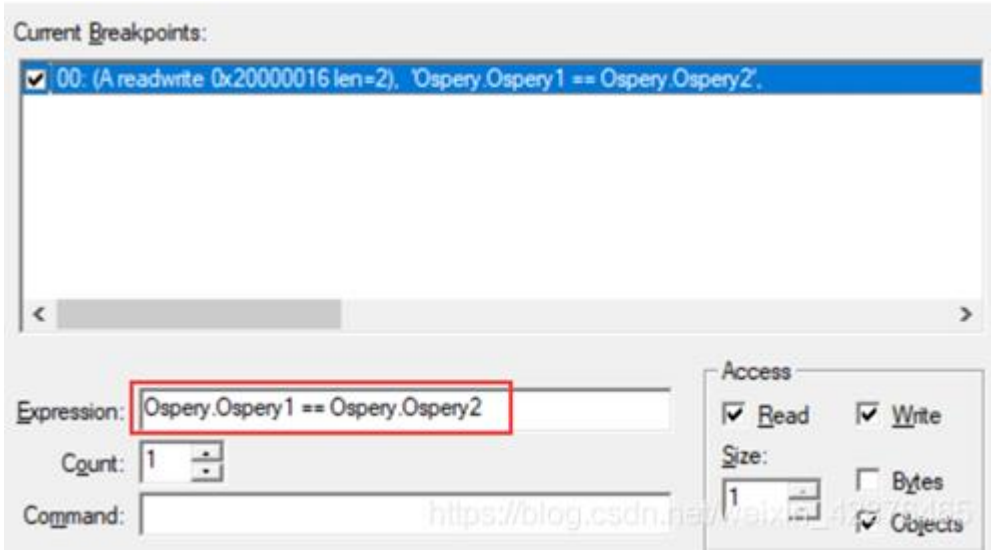
3、資料匹配

有些時候，我們並不關注地址訪問情況，而對變數的資料內容感興趣。比如說魚鷹想讓變數 `emOspery` 等於 `1` 時停下來，怎麼設定？

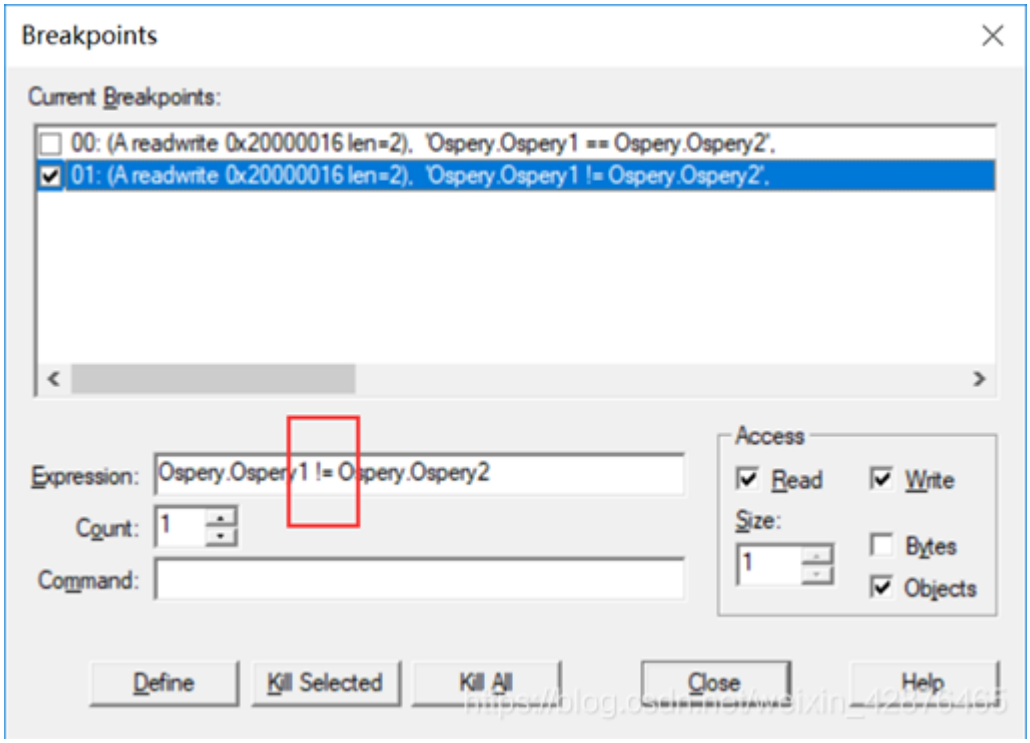


只要簡單的設定 `emOspery == 1` 即可（注意必須設定訪問條件，並且 `Size` 設定正確）。

事實上你也可以設定兩個變數相等作為條件：



設定為不等也是可以的：



當然還有其它支援的運算就靠你們自己去發現了（可支援運算：&，&&，<，<=，>，>= ，==，!=）。

注意：以上內容可以組合使用，比如讀、寫條件，計數器計數等可以同時設定。滿足條件時就會讓程式執行停止。

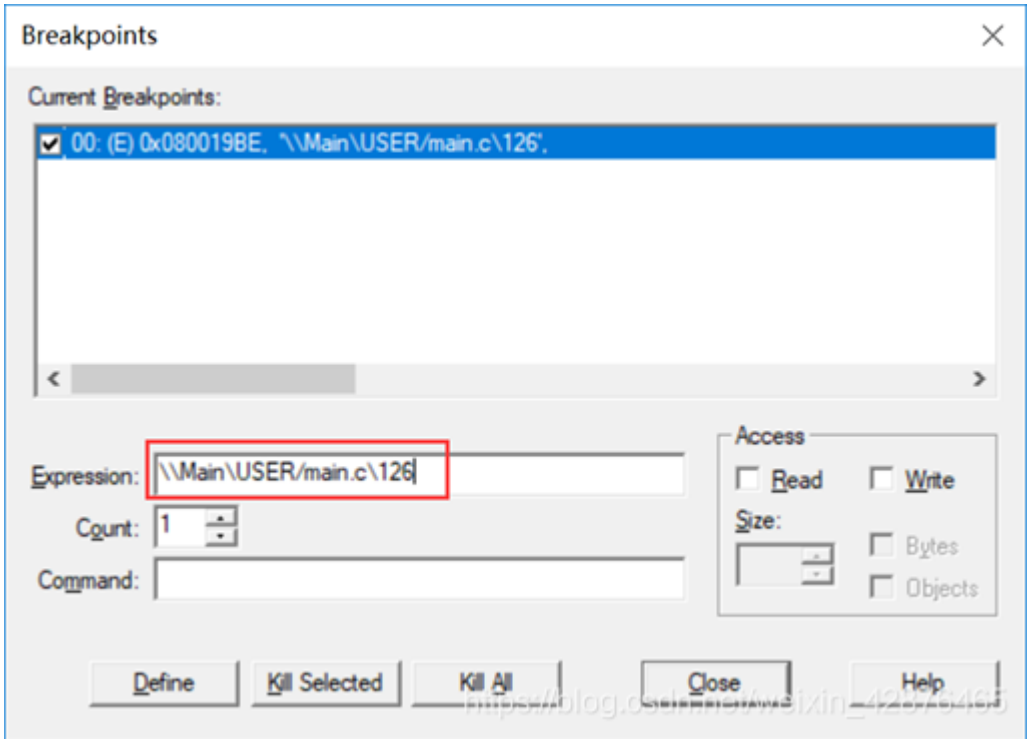
高級用法

以上為比較常規的偵錯功能，現在說說魚鷹剛學習的技能，這個技能的使用靈活性更大，而且對於解決疑難雜症更是不二之選。

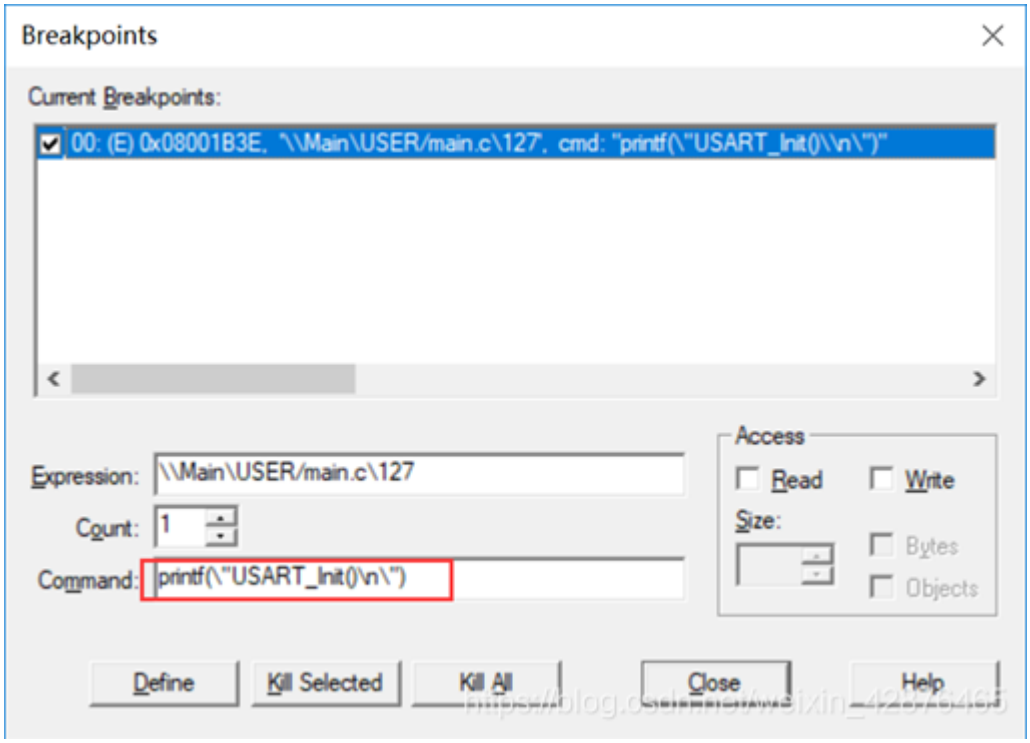
首先設定一個你需要的斷點：

```
123  int main(void)
124  {
125    delay_init(72);
126    USRAT_Init(9600);
127    USART1->DR =1;
128    printf("hello word!\n");
129    PWM_Init(900,0);
130    while(1)
```

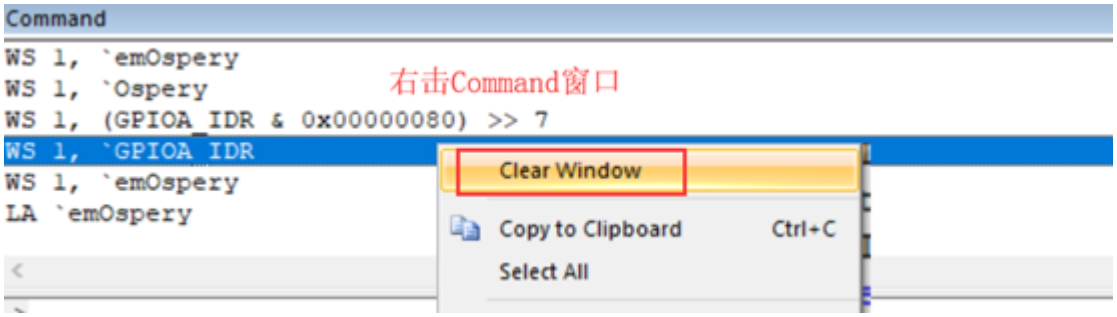
打開斷點窗口，並連按兩下你之前設定的斷點：



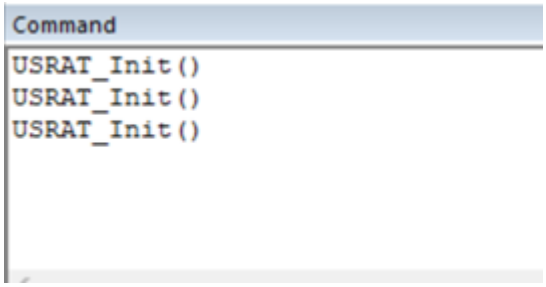
設定 Command 為【printf(“USRAT_Init()\n”)】（注意\n，否則可能不能輸出，這個應該是 KEIL 的一個 bug）。最後【Define】：



清空你之前的命令（如果你不嫌亂的話，也可以不清空）：



那麼你的程序每次運行到這個程式碼位置都會在 Command 窗口輸出一條資訊：



但是你的程序並不會停止。

如果說你想讓斷點程式碼位置運行多次之後才輸出一條資訊也是可以的，只要設定 Count 即可。

這裡可能你會問，這 printf 不就是我們寫的列印函數嗎？事實上，是，也不是。

這個函數是列印函數沒錯，但是這是 KEIL 呼叫的列印函數，輸出位置是 Command 窗口，和你自己寫的程式碼沒一點關係，每次觸發條件時 KEIL 都會呼叫該函數進行列印，而不會讓你的程序暫停運行。事實上斷點窗口的這個 Command 絕不僅僅只是設定 printf 這麼簡單，如果真是這樣我也不會如此推崇它了，感興趣的可以去官網尋找關於偵錯命令的使用方法。

因為是利用 KEIL 去執行列印任務，所以對你的程序幾乎沒有任何影響，並且在你設定斷點後也不用擔心刪除程式碼問題，可以放心飲用。還有一個額外的好處就是，對於所有能設定偵錯斷點的單晶片都適用，因此對於偵錯程式也就沒有過多的要求了，比如說，不管你是用 JLINK、ST-LINK 還是 CMSIS-DAP（CMSIS-DAP 不能使用 ITM，所以魚鷹才會想著用別的方式替代。總算是找到了，而且它在某些方面更出色），都可以這麼用。

原文链接：https://blog.csdn.net/weixin_42876465/article/details/97823112