关于头文件中的 static inline函数

> 整理自 **关于头文件中的 static inline函数**

头文件中常见static inline函数，于是思考有可能遇到的问题，如头文件经常会被包含会不会产生很多副本？网上说法不一。于是自己验证。经过arm-none-eabi-gcc下测试后得出结论。

> inline 关键字实际上仅是**建议内联并不强制内联**，gcc中O0优化时是不内联的，即使是O2以上，如果该函数被作为函数指针赋值，那么他也不会内联，也**必须产生函数实体**，以获得该函数地址。经测试c文件中的仅inline函数即使Os优化也不内联，因为没有static，编译认他是全局的，因此像普通函数一样编译了，本c文件也一样通过bl inline_func 这样的方式调用，不像网上别人说的，本c会内联，其他c文件则通过bl inline_func 方式。加入static 后则内联了。（Os优化等级测试）
>
> ***所以在头文件中用inline时务必加入static，否则当inline不内联时就和普通函数在头文件中定义一样，当多个c文件包含时就会重定义。所以加入static代码健壮性高，如果都内联了实际效果上是一样的。*** （gcc下验证过O0级别includes.h中仅定义inline的函数，编译失败，Os编译成功）

虽然知道了头文件中用inline函数时要加入static，但是为什么要在头文件中定义函数呢?

```
// main.c
inline void open(void)
{
    vfs_open();
}
```

一些简单的封装接口函数，如：

```
 open() { vfs_open() }
```

仅仅是为了封装一个接口，我们不希望耗费一次函数调用的时间，解决方法一是宏，但是**作为接口，宏不够清晰**。那选择inline，但是如果在c文件中写函数体，在头文件中加声明，外部要使用则不会内联的， 因为编译器有个原则，以c文件为单位进行逐个编译obj，每个c文件的编译是独立的，该c文件用到的外部函数都在编译时预留一个符号，只有等到所有obj生成后链接时才给这些符号地址（链接脚本决定地址），所以其他c文件编译时只会看到这个函数的声明而无法知道她的实体，就会像普通函数一样通过bl 一个函数地址，等链接的时候再填入该地址了，他做不到内联展开。

> 所以要内联则必须在每个用到它的c文件体现实体，那就只有在头文件了，**所以会把这类希望全局使用又希望增加效率的函数实现在头文件中**static inline。

static inline 的坏处

因为inline 是C99才有的关键字，C89没有，有部分编译器不支持，或者部分支持，如支持**inline 或** inline__等，所以我们一般会用一个宏定义inline 如：

```
#define INLINE    static inline
//不支持inline时:
#define INLINE    static
```

但是这样如果编译器不支持inline 即意味着之前 static inline的函数全部被修改为 static，在头文件中写static会有什么后果呢?

经过测试果然和我们想的一样，每个c文件包含了该头文件后全部都有了该函数副本。这无疑增大了很多代码量。比如在 include.h 这样的大头文件，几乎每个c文件我们都会包含他，相当于每一C文件都会加入一个 static void func(void){...} 实体。如果是函数宏则不会有这种问题，函数宏是没有实际代码的，没调用他时代码不存在。这就是头文件中用static inline 函数的坏处。但是可以通过优化解决，经过测试，O0优化下在头文件中定义static 函数包含该头文件的三个c文件的确都有了该函数，但是在Os优化下则只有调用了该函数的C文件才有实体。这是由编译器对static函数的特性决定的。总之他的法则和我们想的一致，就是头文件仅仅是单纯的展开，而每个C独立编译，不会因为知道其他个C文件定义了该函数，这个c文件就把他当外部bl了。

关于c文件中的static函数

static函数除了文件内使用这个功能外，在优化上也有作用， static定义后如果该文件没有函数调用他，那么他意味着没有用，其他文件不可能调用，所以开优化就被优化掉了（已验证），无优化时则还在。这点一定要注意中断函数最好不要写static，中断函数如果没有中断服务函数的，即没有被调用，static可能被优化，当然也不一定，因为没有中间服务函数的独立中断服务函数必须在链接脚本体现，可能需要再链接脚本上加入KEEP参数应该就没问题。

这里排除非gcc编译器，如code worrior编译器则不同，code worrior是定制型的，通过识别main函数，因此他有主函数枝干可以判断哪些函数被调用，哪些是无用的，但是gcc不同，没有所谓的main，所以三个c文件如果没有static的函数，即使开优化也都会编译出来，他并不知道哪些函数有用，哪些函数无用。而static后他就一定知道他没被调用即无用，所以可以优化掉。

以上均指arm-none-eabi-gcc环境下测试的结论，其他编译器有些不同。但是c语言的编译规则还是以gcc为标准，即使其他编译器有些不同我们平时编程时还是以这个为标准。

在codeworrior 编译器上测试，即使0优化依然会把inline内联，有main函数，没有被主干调用的都是无用函数全部被标记UNUSED，不编译，因此上面讨论的当不支持inline时static函数实体过多的问题它也不存在。