**FreeRTOS学习之二：二值信号量**

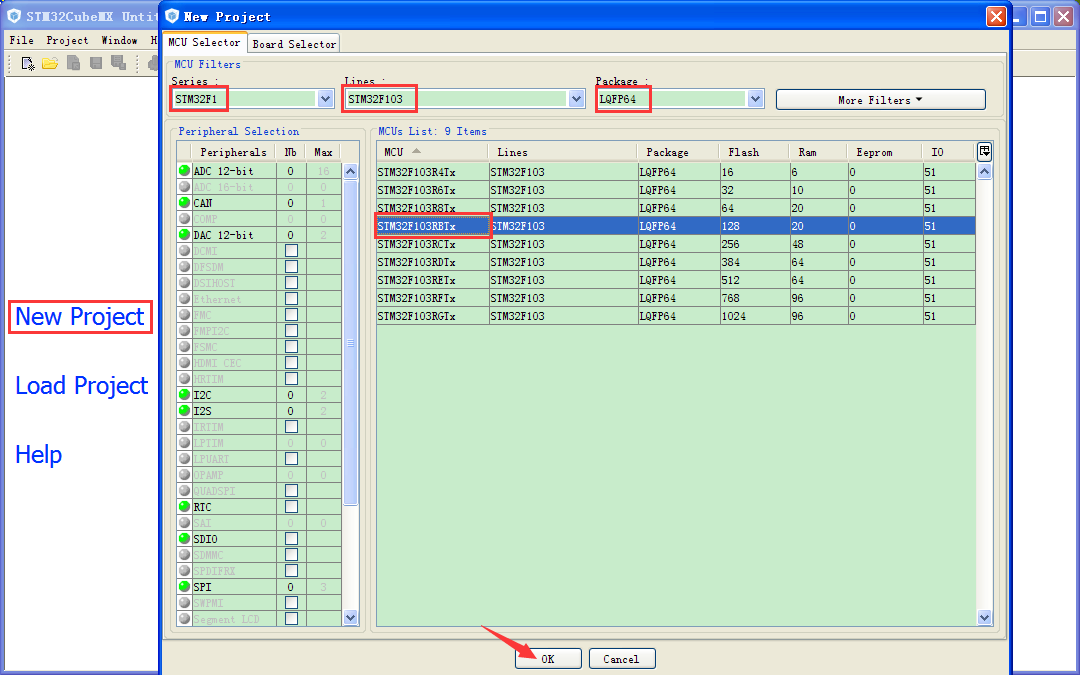
前提：默认已经装好MDK V5和STM32CubeMX，并安装了STM32F1xx系列的支持包。

硬件平台：STM32F1xx系列。

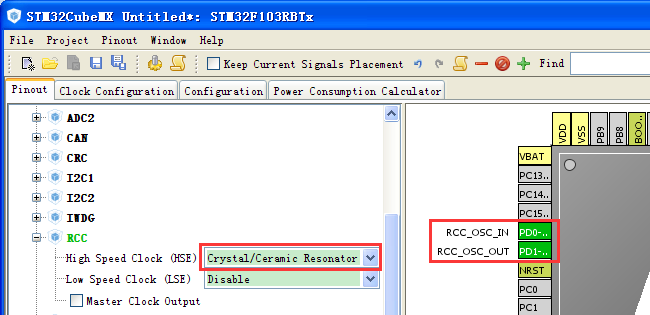
目的：学习使用二值信号量进行任务同步。

二值信号量主要用于任务的同步。本文例子使用STM32CubeMX配置创建两个任务，一个任务每秒钟发送一次信号量，另一个等待信号量并控制LED的输出状态。

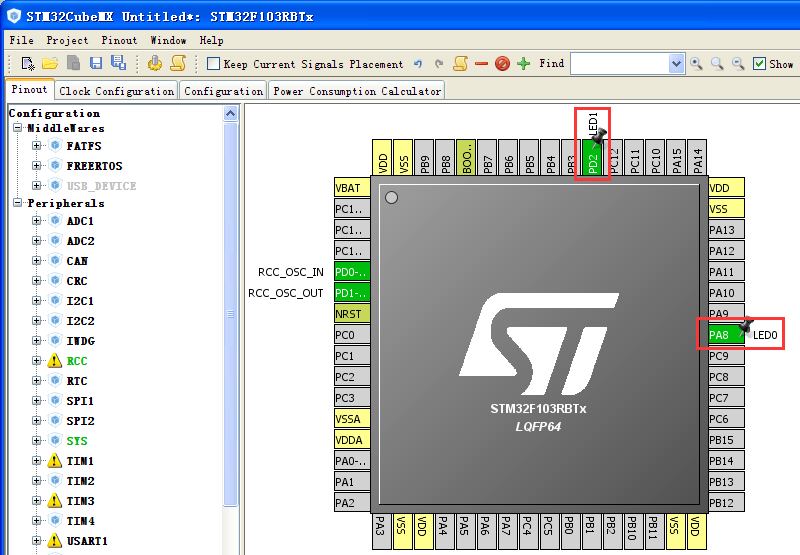
Step1.打开STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



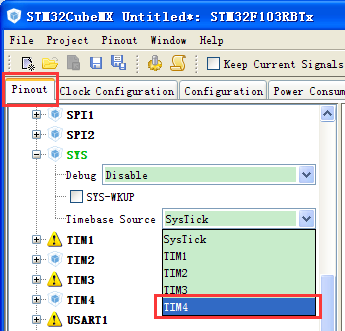
Step2.配置时钟引脚。



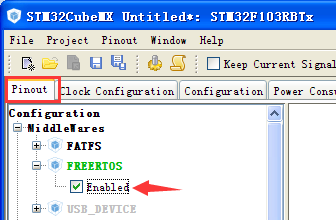
Step3.配置PA8和PD2为Output，并把用户标签分别改为LED0，LED1。



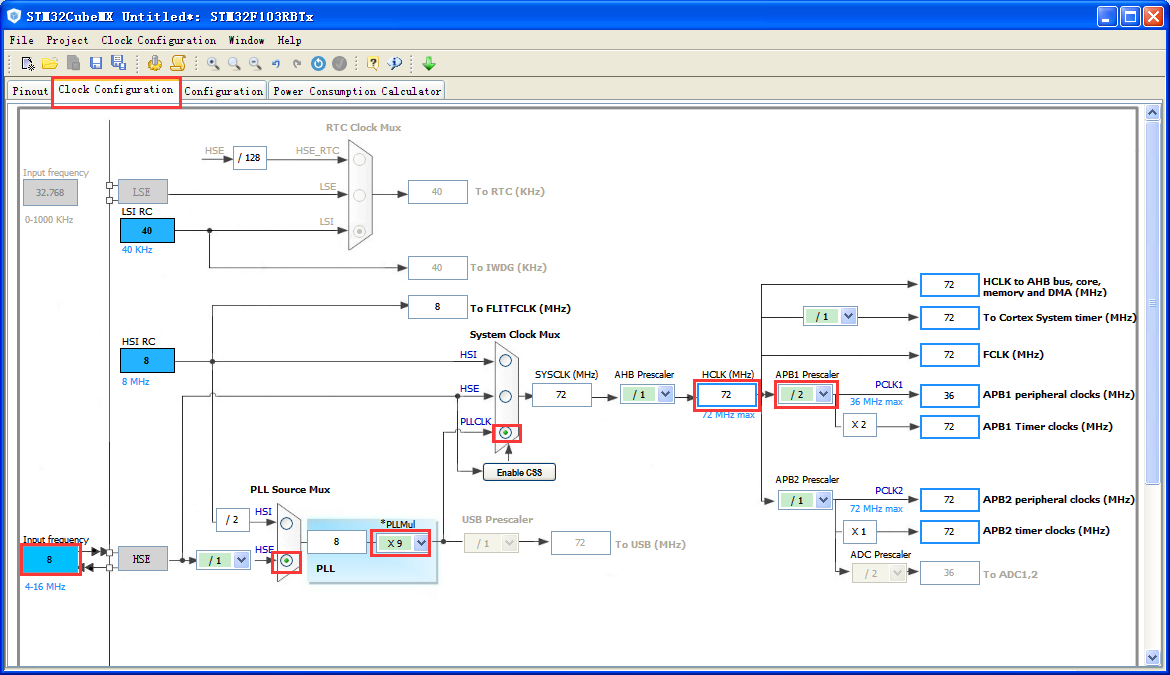
Step4.将系统时基源改为TIM4。



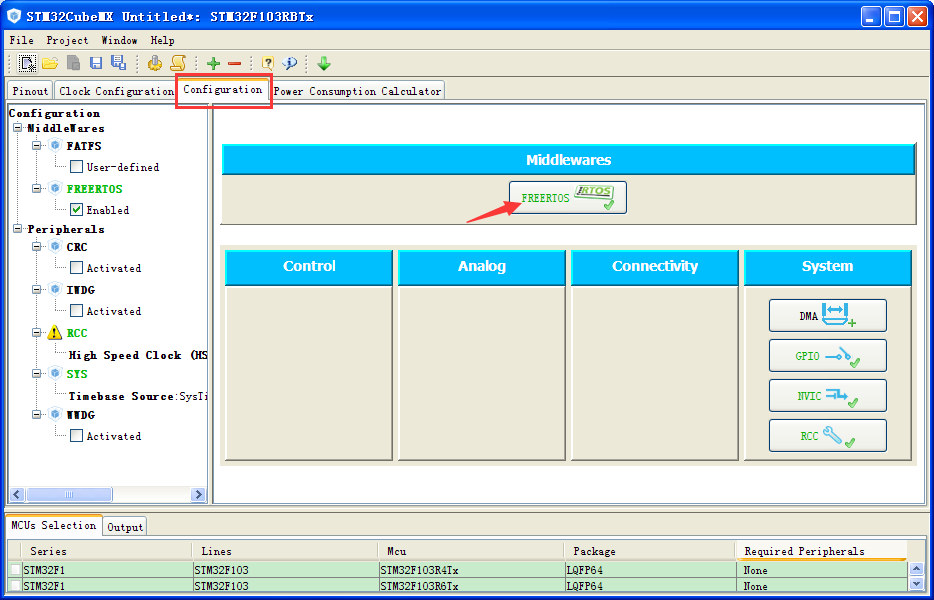
Step5.使能FreeRTOS。



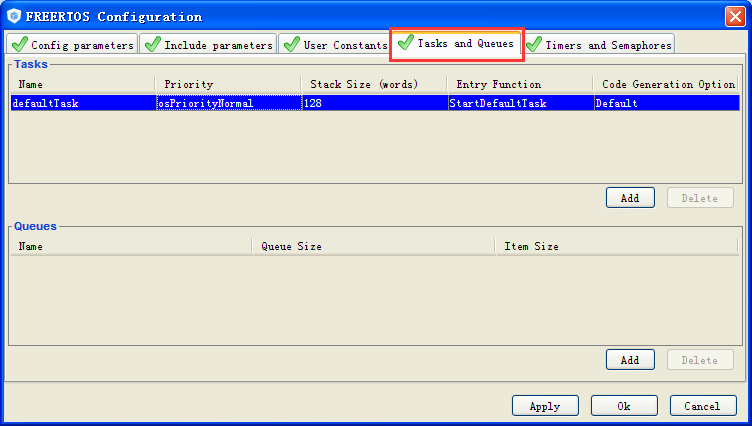
Step6.配置时钟树。8M输入时，通过PLL得到72M内部时钟。



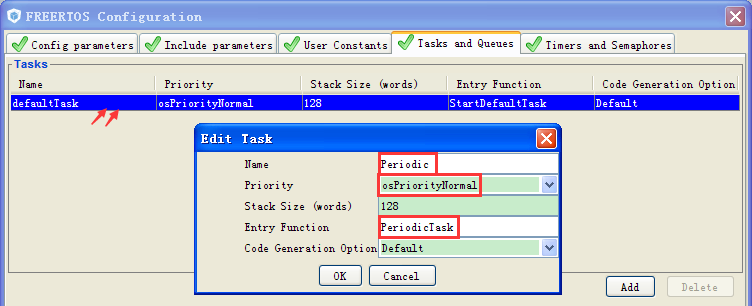
Step7.配置FreeRTOS。



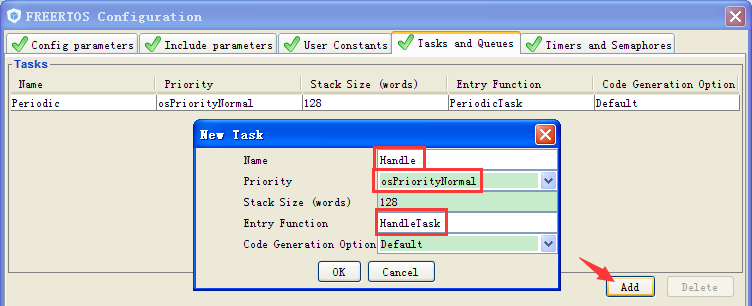
在Tasks and Queues选项卡中，默认配置了一个名为defaultTask的任务，其优先级为普通，任务堆栈大小为128字，任务函数名为StartDefaultTask。



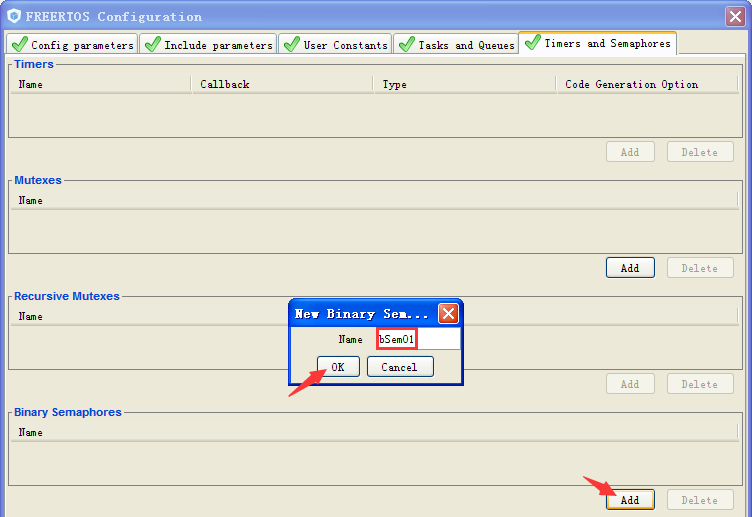
双击蓝色的地方，弹出对话框，将任务名修改为Periodic，将任务函数名修改为PeriodicTask。



点击Add按钮，增加一个任务Handle，优先级设置为Normal，函数名为HandleTask。

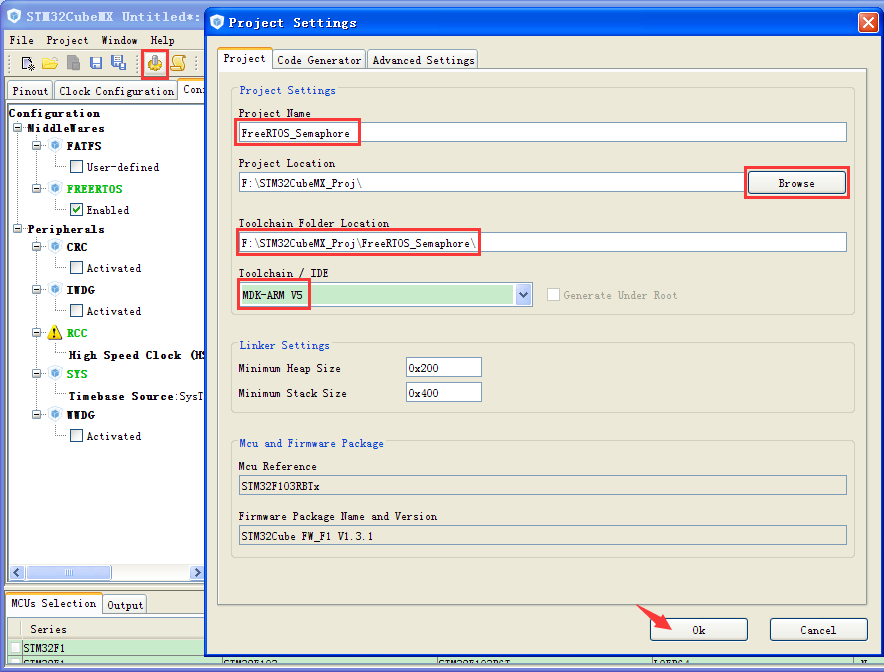


在Timers and Semaphores选项卡，点击Binary Semaphores项右边的“Add”按钮，添加一个信号量，名称改为bSem01。

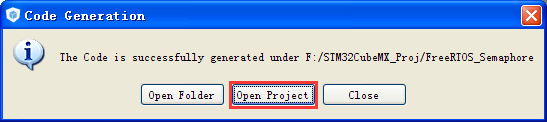


注：该步骤中，除了添加任务和信号量，其他的都使用默认参数。

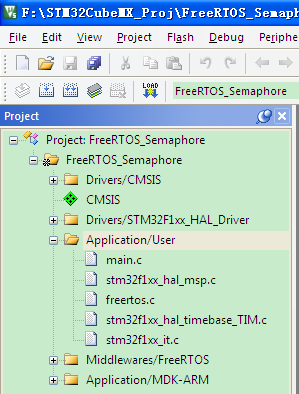
Step8.生成代码。



等完成后直接打开工程。

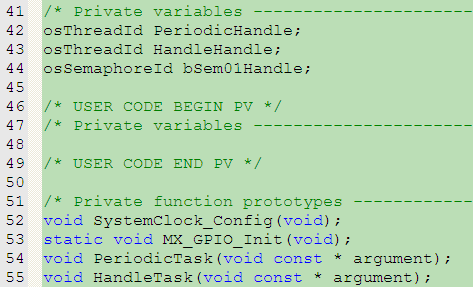


工程基本组织结构如下图，其中Application/User组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。

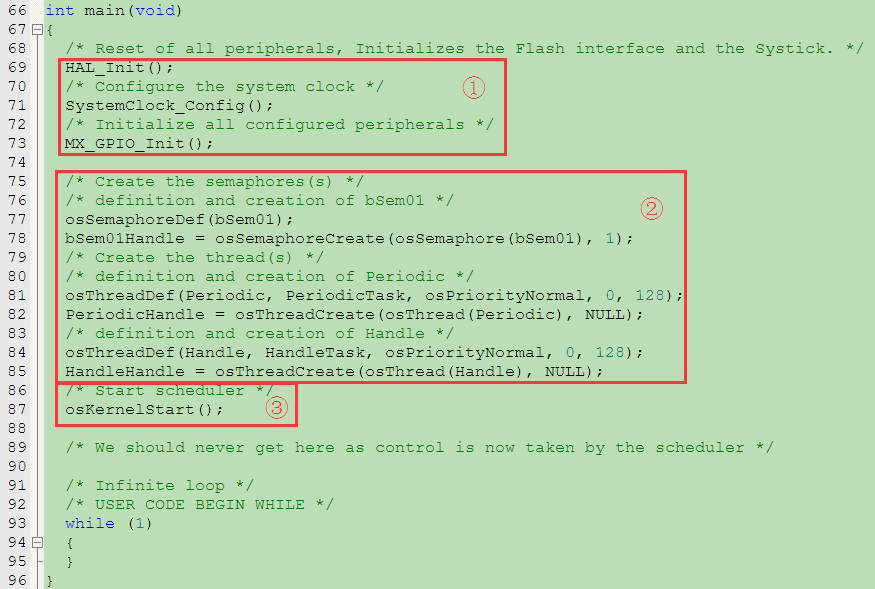


Step9.分析程序结构。

在进入main函数之前，先定义了几个变量，声明了几个函数。



再看main函数。将main函数整理，删除很多注释之后，得到下图所示内容。



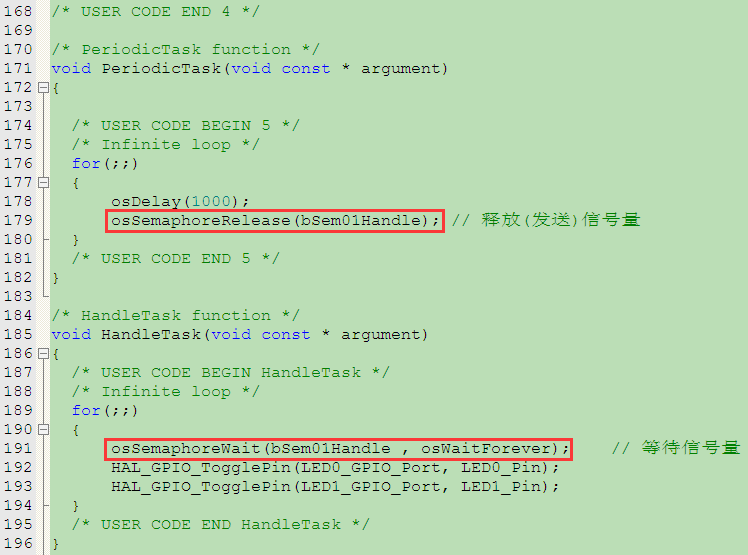
其中第①部分，是硬件配置；第②部分，创建一个信号量和两个任务；第③部分，启动调度器。

启动调度器后，程序就由FreeRTOS的调度器管理了，后面的while(1)是不会执行到的。

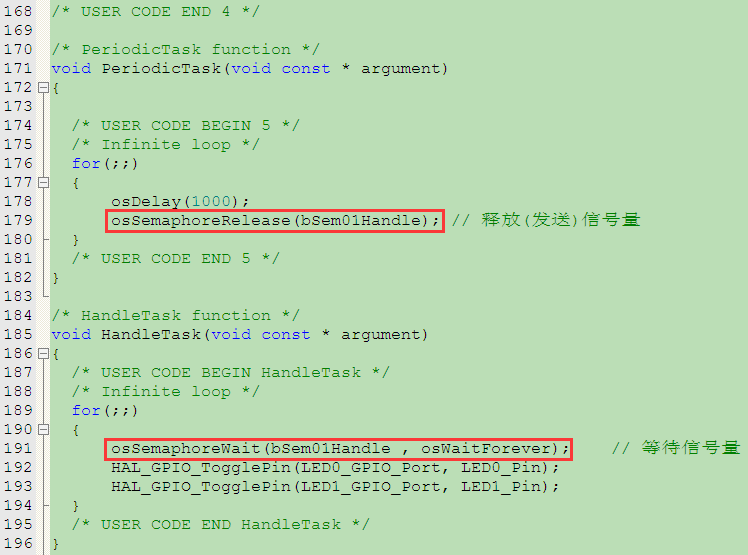
Step10.添加代码。

在main.c文件中，找到前面配置添加的两个任务函数，并在其中分别添加代码。

PeriodicTask的功能是，每1秒钟发送一次信号量。



HandleTask的功能的，等待信号量，然后翻转LED0和LED1的输出状态。



Step11.编译下载运行。LED0和LED1分别闪烁，周期都是2秒。

程序分析：

1.分析语句：osSemaphoreDef(bSem01);

和osThreadDef(…)类似，osSemaphoreDef(…)并不是一个函数，而是一个宏。

其定义在cmsis\_os.h文件中，作用是定义一个osSemaphoreDef\_t结构体。



2.分析语句：bSem01Handle = osSemaphoreCreate(osSemaphore(bSem01), 1);

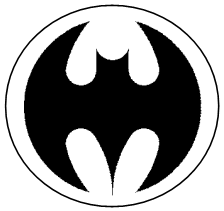
同样的，osSemaphore (…)也是一个宏定义，在cmsis\_os.h文件中可查到。

该语句的作用就是创建一个信号量，最后一个参数=1时，创建的就是一个二值信号量。

3.HandleTask任务在执行osSemaphoreWait(bSem01Handle , osWaitForever);语句后，进入阻塞状态，等待PeriodicTask任务释放信号量。在实际运用中，二值信号量主要用于任务同步。

如果把上面的PeriodicTask任务换成硬件定时器的ISR函数，那么HandleTask任务就相当于定时器的延迟处理函数。这是RTOS种常用的方法。因为RTOS调度本身使用了较低优先级的定时器中断实现，如果在硬件ISR中执行过多的代码，就会造成RTOS响应受到很大影响。因此，在ISR中只发送信号量，然后在其延迟处理函数中实现真正的用户功能。这本质上就是，将任务函数和硬件中断进行同步。

需要注意的一点是，在原生的FreeRTOS中，信号量的发送和获取在ISR中要使用以FromISR结尾的特定函数。在STM32Cube生成的代码中，ST的工程师已经把接口进行了统一，在相应的函数中，通过查询程序状态寄存器判断当前是在ISR中还是在普通函数中。所以用户在使用时，统一使用osSemaphoreWait()和osSemaphoreRelease()函数操作即可。同样的，FreeRTOS的其他通信方式如队列、互斥量等，都进行了这样的处理。



S.D.Lu 于 深圳

2016年8月