在C++中，编写用户态协程框架首先需要理解两个关键的概念：栈管理和上下文切换。

1. 栈管理：协程运行时需要一个私有的栈空间，用于存放局部变量等数据。当从一个协程切换到另一个协程时，需要保存当前栈的状态，并恢复目标协程的栈状态。
2. 上下文切换：协程的运行需要依赖于寄存器的状态，特别是程序计数器（PC）和栈指针（SP）。因此，在切换协程时，需要保存当前协程的寄存器状态，并恢复目标协程的寄存器状态。

这两个关键的概念在C++中可以通过ucontext.h库中的函数来实现。具体来说，可以使用getcontext和setcontext函数来实现上下文的保存和恢复，使用makecontext函数来创建新的上下文。在创建新的上下文时，需要提供一个新的栈空间，这个栈空间可以通过malloc函数来分配。

以下是一个简单的协程框架的示例：

#include <ucontext.h>

#include <stdio.h>

#include <stdlib.h>

#define STACK\_SIZE 1024\*64

*// 创建两个ucontext\_t结构体，一个用于主流程，一个用于协程*

ucontext\_t main\_context,co\_context;

void co\_func()

{

printf("Hello, I am coroutine!\n");

*// 切换回主流程*

swapcontext(&co\_context,&main\_context);

}

int main()

{

*// 初始化协程上下文*

getcontext(&co\_context);

co\_context.uc\_stack.ss\_sp=malloc(STACK\_SIZE); *// 分配栈空间*

co\_context.uc\_stack.ss\_size=STACK\_SIZE;

co\_context.uc\_link=&main\_context; *// 协程结束后返回到主流程*

makecontext(&co\_context,co\_func,0); *// 设置协程的入口函数*

printf("Hello, I am main!\n");

*// 切换到协程，保存当前上下文*

swapcontext(&main\_context,&co\_context);

printf("Back to main!\n");

return 0;

}

以上代码在主流程中创建并切换到一个协程，该协程打印一条消息后又切换回主流程。这就是一个最简单的协程框架。

需要注意的是，以上代码不包含对并发控制和调度的实现，如果需要实现这些，还需要借助于其他的工具和技术，比如使用队列来管理协程的调度，使用锁来控制对共享资源的访问等。