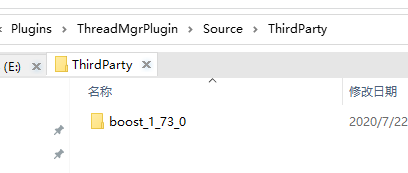
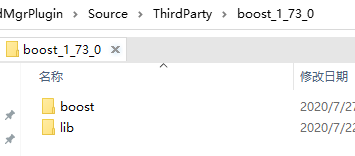
## boost接入说明

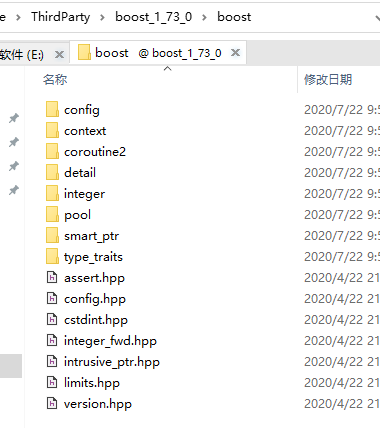
协程使用了boost中的coroutine2库，需要接入boost库。先从boost官网下载1.73版本



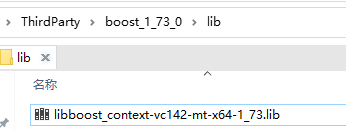
boost\_1\_73\_0 文件夹中的结构如下



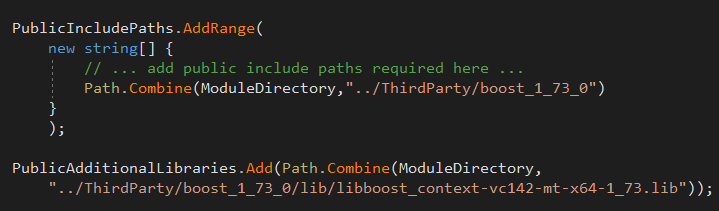
其中boost是coroutine2相关的**头文件**



lib是编译boost产生的lib库

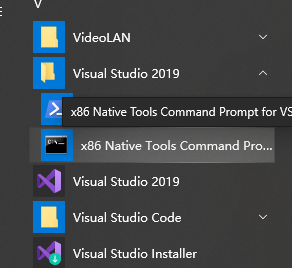


在ThreadMgrPlugin.Build.cs中指定**PublicIncludePaths**和**PublicAdditionalLibraries**

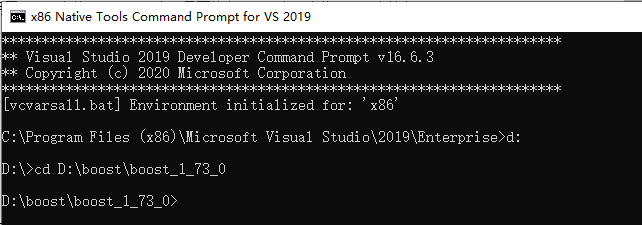


## lib编译方法

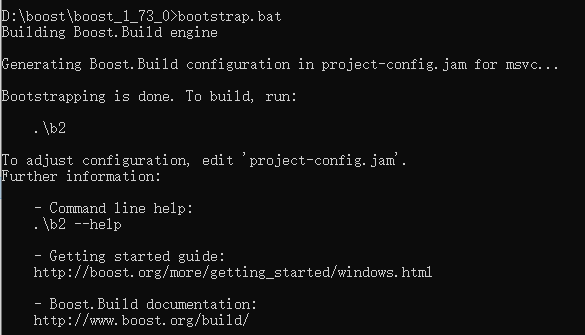
1、启动vs命令行工具



2、cd 到boost目录

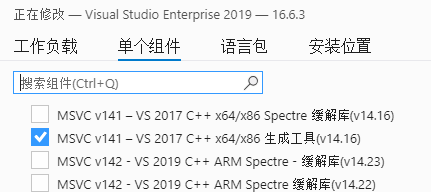


3、运行bootstrap.bat

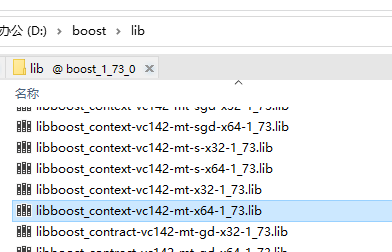


4、运行b2 stage --toolset=msvc-14.2 --without-python --stagedir="d:\boost\lib" link=static

注意：这里使用的是vs2019 msvc为14.2如果需要使用vs2017的msvc141，可以通过visual stuido Installer安装msvc141。然后在这里指定--toolset=msvc-14.1



5、生成完成后拷贝**libboost\_context-vc142-mt-x64-1\_73.lib**到前面所说的文件夹Plugins\ThreadMgrPlugin\Source\ThirdParty\boost\_1\_73\_0\lib中

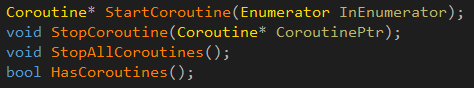


注意：如果是vs2017需要拷贝libboost\_context-vc141-mt-x64-1\_73.lib

## 协程使用方法

协程和线程都封装在ThreadMgr中，ThreadMgr是一个Actor，直接放置在场景中，获得ThreadMgr引用即使用

**协程主要接口：**



Enumerator是一个std::function<void (CoroPush&)>对象，可以这样定义：

Enumerator ATestCoroutineActor::TaskA()

{

return [=](CoroPush& yield\_return)

{

UKismetSystemLibrary::PrintString(GetWorld(), "TaskA A1");

yield\_return(nullptr);

UKismetSystemLibrary::PrintString(GetWorld(), "TaskA A2");

};

}

如果需要延迟指定时间：

Enumerator ATestCoroutineActor::TaskB(float Param)

{

return [=](CoroPush& yield\_return)

{

while (true) {

PrintString("TaskB B1", 5.f, FLinearColor::Red);

yield\_return(new WaitForSeconds(Param));

PrintString("TaskB B2", 5.f, FLinearColor::Red);

yield\_return(new WaitForSeconds(Param));

PrintString("TaskB B3", 5.f, FLinearColor::Red);

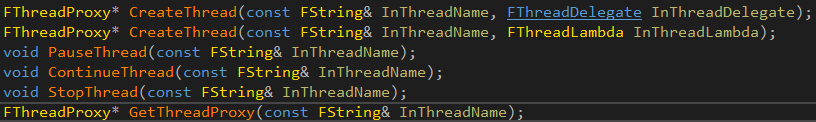
}

};

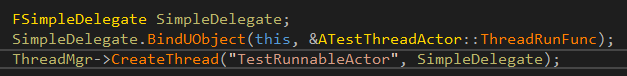
}

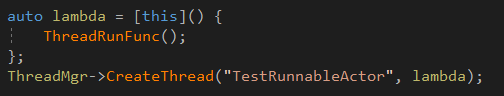
## 线程使用方法

**线程主要接口：**



说明：

1. 线程创建需要提供线程名和线程运行函数，这个运行函数可以是委托或lambda表达式
2. 委托形式：  
   
3. lambda形式；



1. 可以对线程进行pause和continue操作
2. 如果线程不需要使用时，需要调用StopThread

注意：为了防止开启线程后，游戏退出后，创建方没有关闭线程，在ThreadMgr 销毁时，会销毁所有通过ThreadMgr创建出的线程。