Commandlet用法和独立进程的调用

用处

CommandLet可以实现UE4编辑器功能的独立调用。例如在不启动可视化编辑器时调用某个引擎任务或功能；希望某个任务异步调用时，可以通过启动一个新进程完成异步任务。

实现

1. 创建继承UCommandlet的子类，例如UTestCommandlet
2. 实现该类继承的抽象方法virtual int32 Main(const FString& Params) override;
3. 外部调用和执行：

   ```
   "D:/Epic Games/UE_4.26/Engine/Binaries/Win64/UE4Editor-Cmd.exe" D:/Unreal
   Project/MyProject/MyProject.uproject -run=Test
   ```

   注： -run后跟的参数为调用的任务，一般为所实现类的类名除去Commandlet的名字

更多

1. 参数解析

方法一：

```
int32 UTestCommandlet::Main(const FString& Params)
{
    FString Param1, Param2;
    FParse::Value(*Params, TEXT("param1="), Param1);   //解析key=value类型的参数
    FParse::Value(*Params, TEXT("param2="), Param2);
}
```

方法二：

```
TArray<FString> Tokens;                     //形如 BuildCook 的参数，纯文本
TArray<FString> Switches;                   //形如 -bSkipSoftReferences 的参数，带-
TMap<FString, FString> ParamMap;            //形如 key=value 的参数，key/value

UCommandlet::ParseCommandLine(*Params, Tokens, Switches, ParamMap);
```

2. UE4调用

一般调用：

```
/**
*   @param URL executable name
*   @param Parms command line arguments
*   @param bLaunchDetached  if true, the new process will have its own window 新进程是否开启一个窗口
*   @param bLaunchHidden    if true, the new process will be minimized in the task bar 新进程是否最小化到任务栏
*   @param bLaunchReallyHidden  if true, the new process will not have a window or be in the task bar 是否完全隐藏新进程
*   @param OutProcessId if non-NULL, this will be filled in with the ProcessId
*   @param PriorityModifier 2 idle, -1 low, 0 normal, 1 high, 2 higher
*   @param OptionalWorkingDirectory Directory to start in when running the program, or NULL to use the current working directory
*   @param PipeWrite    Optional HANDLE to pipe for redirecting output
*/
//第一个参数为UE4Editor-cmd.exe的路径，第二参数为项目路径+commandlet任务参数+额外参数
FProcHandle Handle =  FPlatformProcess::CreateProc(*EngineExePath, *Params, false, false, true, nullptr, 0, nullptr, null);
```

官方文档：https://docs.unrealengine.com/4.27/en-US/API/Runtime/Core/GenericPlatform/FGenericPlatformProcess/CreateProc/

等待任务完成：

```
FPlatformProcess::WaitForProc(Handle);
```

获取返回值：

```
int32 ReturnCode;
FPlatformProcess::GetProcReturnCode(Handle, &ReturnCode);
```

获取并打印日志和消息：

```
void * PipeRead = nullptr;
void * PipeWrite = nullptr;
verify(FPlatformProcess::CreatePipe(PipeRead, PipeWrite));

FProcHandle Handle =  FPlatformProcess::CreateProc(*EngineExePath, *Params, true, false, false, nullptr, 0, nullptr, PipeWrite);

//获取并打印日志和消息
while(FPlatformProcess::IsProcRunning(Handle))
{
    FString ReadStr = FPlatformProcess::ReadPipe(PipeRead);
    UE_LOG(LogTemp, Display, TEXT("%s"), *ReadStr);
    FPlatformProcess::Sleep(0.01f);
}

FPlatformProcess::ClosePipe(PipeRead, PipeWrite);
```