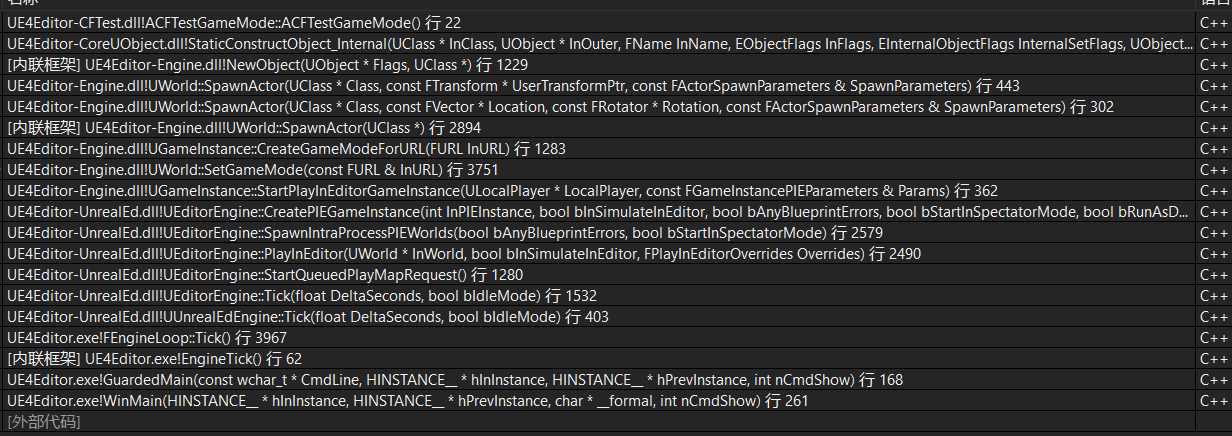
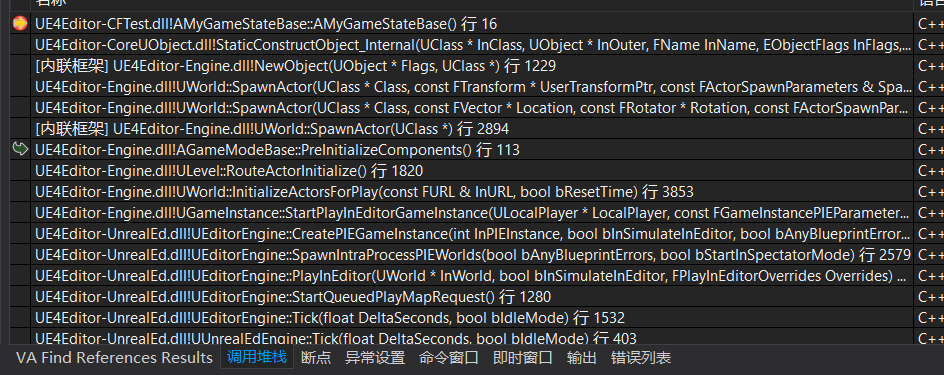
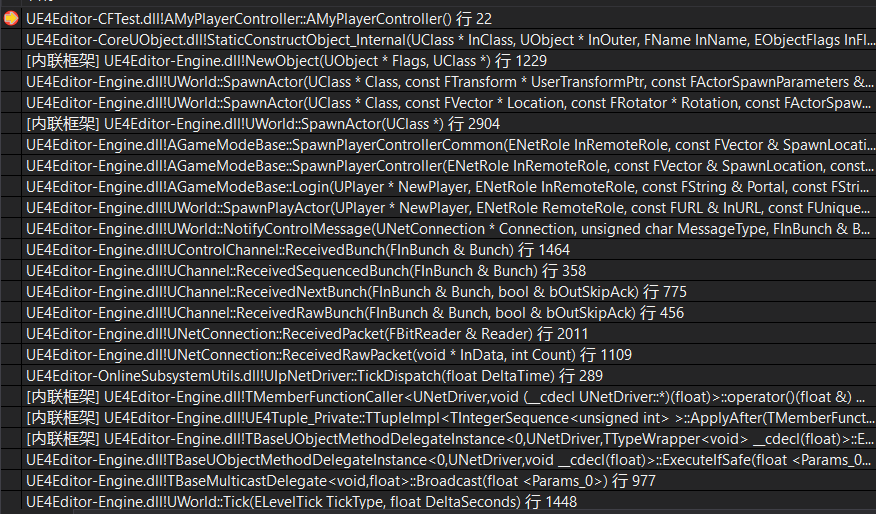
World有很多种

GameInstance是在GameEngine里创建的







所以本质上来说，UE的编辑器其实也是个游戏！我们是在编辑器这个游戏里面创造我们自己的另一个游戏。话虽如此，但比较编辑器和游戏还是有一定差别的，所以UE会在不同模式下根据编译环境而采用不同的具体Engine类，而在基类UEngine里通过一个WorldList保存了所有的World。

Persisitent的意思是一开始就加载进World，Streaming是后续动态加载的意思。Levels里保存有所有的当前已经加载的Level，StreamingLevels保存整个World的Levels配置列表。PersisitentLevel和CurrentLevel只是个快速引用。在编辑器里编辑的时候，CurrentLevel可以指向其他Level，但运行时CurrentLevel只能是指向PersisitentLevel。

**思考：哪些逻辑应该放在GameInstance？**  
第二个惯例的问题是，这一层应该写些什么逻辑。顾名思义，既然是作为游戏中全局唯一的长者，我们就应该给他全局的控制权。在逻辑层面，GameInstance往下看是：

1. Worlds，Level的切换实际发生地是Engine，而GameInstance可以说是UE之神其下的唯一代言人，所以GameInstance也可以代之管理World的切换等。我们可以在GameInstance里实现各种逻辑最后调用Engine的OpenLevel等接口。
2. Players，虽然一般来说我们直接控制Players的机会不多，都是配置好了就行。但要是到了需要的时候，GameInstance也实现了许多的接口可以让你动态的添加删除Players。
3. UI，UE的UI是另一套World之外的系统，虽然同属于Viewport的显示之下，但是控制结构跟Actor们并不一样。所以我们常常会需要控制UI各种切换的业务逻辑，虽然在Widget的Graph里也可以写些简单的切换，但是要想复用某些切换逻辑的时候，在特定的Wdiget里就不合适了，而GameMode一方面局限于Level，另一方面又只存在于Server；PlayerController也是会切换掉的，同时又只存在于World中，所以最后比较合适的就剩下GameInstance了，以后当然有可能了可能会扩展出个UI的业务逻辑Manger类，不过那是后话了。
4. 全局的配置，也常常需要根据平台改变一些游戏的配置，Execute一些ConsoleCommand，GameInstance也是这些命令的存放地。
5. 游戏的额外第三方逻辑，如果你的游戏需要其他一些控制，比如自己写的网络通信、自定义的配置文件或者自己的一些程序算法，如果简单的话，GameInstance也可以一放，等复杂起来了，也可以把GameInstance当作一个模块容器，你可以在里面再扩展出来其他的子逻辑模块。当然如果是插件的话，还是在自己的插件Module里面自行管理逻辑，然后把协调工作交给GameInstance来做。