**Unity中任务模块化工作流框架的实现与使用**

最近又看到一些公司会使用声明式的方式定义一些行为，例如在游戏初始化时声明需要下载哪些东西，因为时声明式的写法所以看起来非常清晰，就是一个列表一样的，先做啥，然后做啥一目了然。

然后之前也有想到使用模块化工作流的方式在写工具的时候可以很方便地制作一些小工具。例如，拷贝、解压缩之类的，全部都是很简单的命令，但是我们每次都要通过自己写重复代码来实现。

所以模块化的工作流可以大大提高我们的效率。甚至可以通过编辑器来编辑我们的定制工作流。

看下面的例子：

我们有以下任务：Task1、Task2、Task3、Task4.

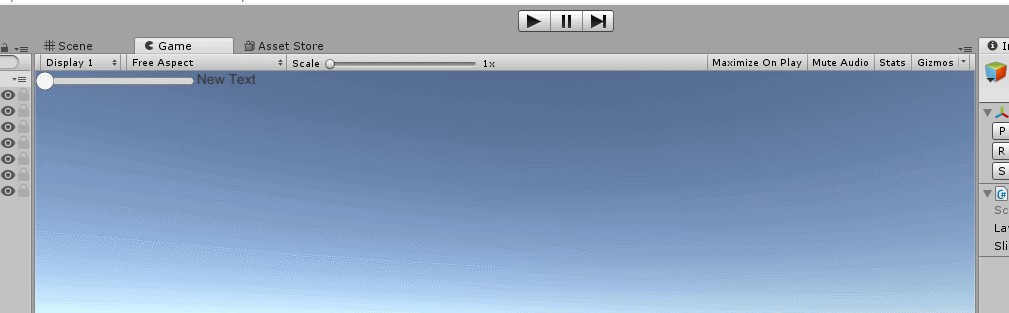
就打个比方吧，Task3是资源解压缩，Task2是数据初始化，Task4是加载Bundle，Task1是开始游戏。

在初始化数据和加载Bundle之前我们需要解压缩资源，然后开始游戏之前我们需要初始化数据和加载Bundle。

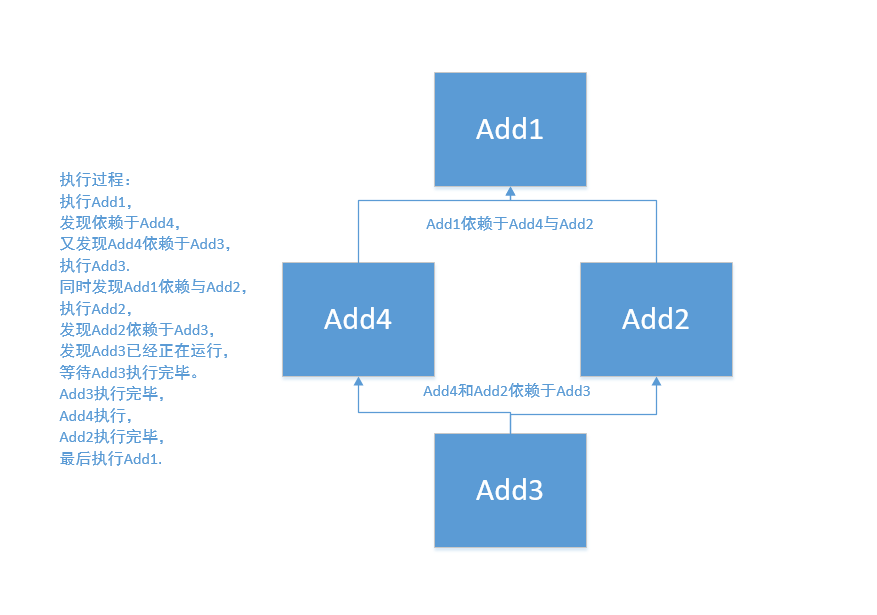
Task3-》Task2、Task4-》Task1

过程分成3段，先执行Task3，然后同时执行Task2和Task4，最后执行Task1

如图：

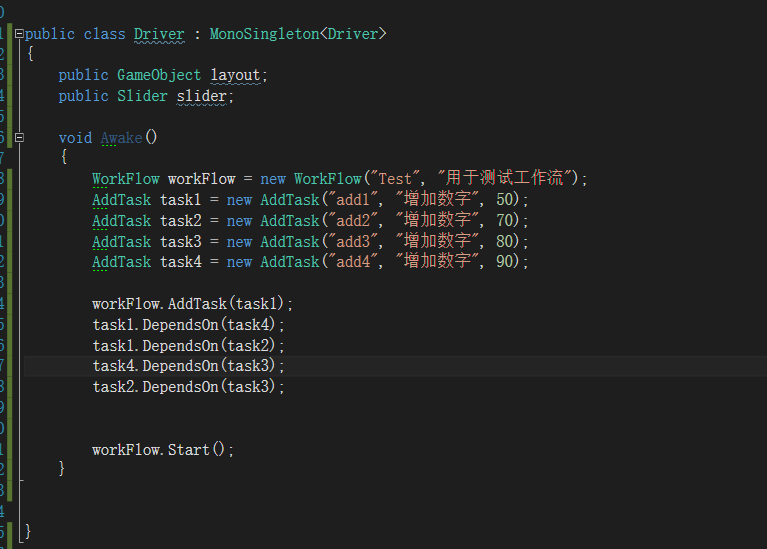


过程如下图：

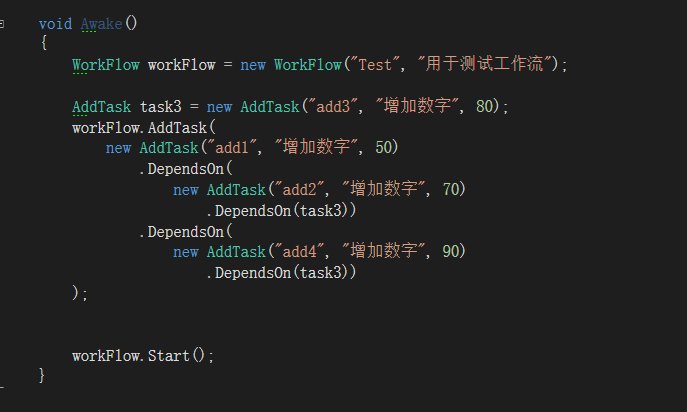


过程大概是这样的。

其中代码是这样的：



当然，我也尽量将设计变得尽量可格式化：



由于Task被两个Task共同依赖了所以需要提前定义。

在我的设计当中，我将工作流分成三个主要组成部分：

**WorkFlow：作为整个工作流的容器，包含了所有的任务以及属性**

**Task：组成工作流的组件，其中组件我们可以复用，我们可以定义自己的Task来供以后使用**

**Property：定义了WorkFlow的一些属性，供Task使用**

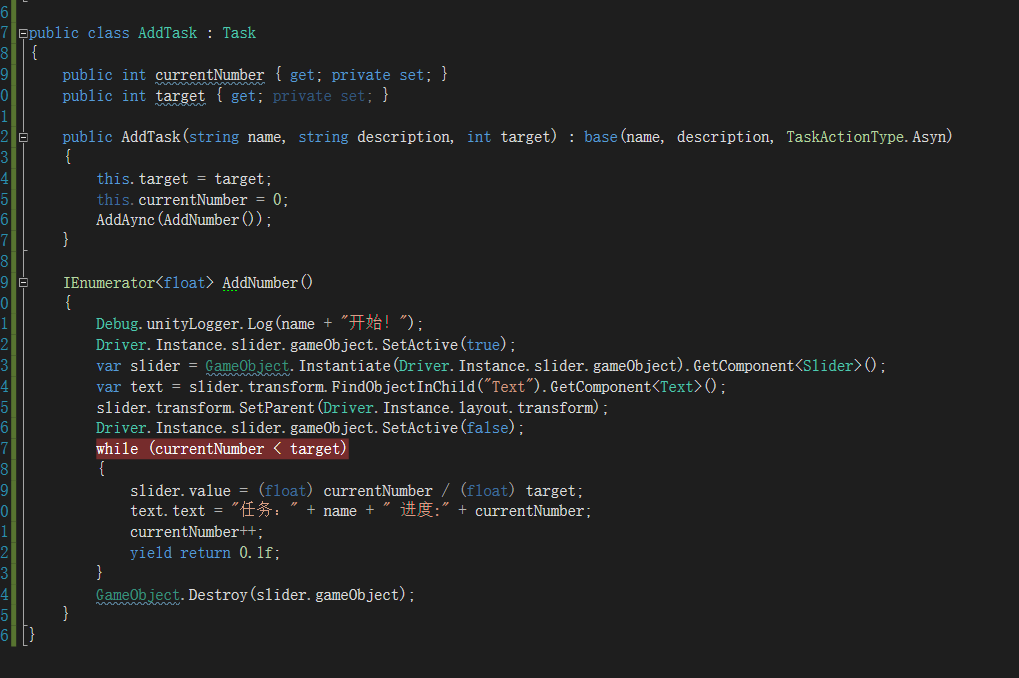
大家可以看到，在这个例子当中，我定义了一个AddTask用于显示于增加数字（实际开发当中建议尽量将数据与界面尽量分离，这里只是为了演示方便）

我首先创建了WorkFlow，然后添加了Task1，然后添加了Task1所依赖的Task，以此类推，直到将所有需要使用的Task都加入工作流当中。

做过Java或者Android开发的同学可能一下就看出来了，这不就是Gradle里面的Project、Task和Property吗？没错，我确实参考了Gradle一些设计，不过也只是抄了非常浅显的一些东西而已，而且不包含构建的所有东西，所以实现起来也不废劲。

由于没有办法像Grooy这种动态语言一样直接用点操作符来调用不存在的属性，所以我就只能使用Dictionary来存了，丑是丑，但是可行，要是以后做出了图形界面的话也用不着管丑不丑了。

我们可以先看看我所自定义的AddTask：



我们可以看到我们重写了构造函数，在构造函数中调用了AddAync，以便加入异步行为。

我所使用的协程是我自己编写的而并非Unity原生的协程，这是因为原生的协程开销较大并且无法判断协程是否已经完成，但是我自己编写的协程则可以自由扩展：

例如等待所有加入的协程完成、等待异步行为等等。详细可以看我以前的文章：[自己实现一个零GC的高效率协程](http://www.resetoter.cn/?p=66)

例如我想要编写一个下载任务，那么我只需要继承Task，并且编写自己的下载异步方法就可以在不同的使用场景中不断复用这个下载任务了。

这样的好处不仅仅是复用，而且相当于实现了一种工作流的数据结构，我可以将整个流程序列化到文件当中，可以通过文件来更新逻辑，更进一步地可以制作出工作流编辑器，类似于Jinkins或者是utomate。大大提高我们的工作效率，以完成持续集成或者规范化导入等等的工作。

这里只是提供了一个解决问题的思路，具体的代码细节并没有给出，以后心情好的话可以写一写。（不过八成是懒得写了，SEO都没做，都没什么人看，只能自嗨一下了