## 逻辑控制模块

逻辑控制模块是整个后台逻辑的基础。所以，基于 Pygame 的特点和简化编码以及维护出发，做出如下要求：

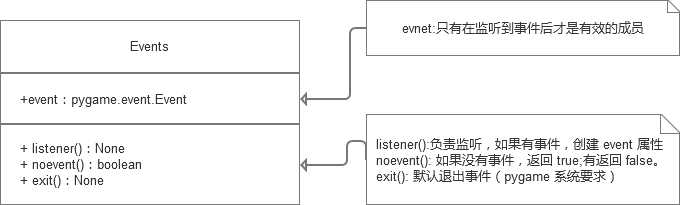
1. 逻辑控制的总体要求是：有一个类（ PyClass ）负责总控制：初始化，加载资源，处理事件，更新表面，销毁退出（有python内省控制）。有一个运行入口函数( AppRun ) 作为整个程序的入口函数，利用 PyClass 组织整个程序接口。
2. 所有渲染对象都是 “精灵”，每个精灵负责自己的“触发事件”，“更新”。渲染逻辑在精灵类中实现，字总控制类中被调用

### 事件监控 --- event模块

也就是说，主控程序不需知道各个渲染对象的逻辑，只需要维护相关接口即可。简化主控的复杂性。基于此，事件监听应该满足以下特点：

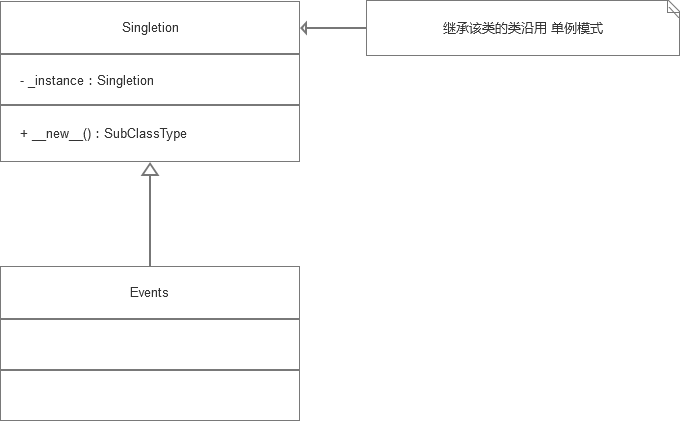
1. 事件监听在主控中只发生一次（Pygame中多处监听会导致其内部出错）
2. 事件监听的作用域在整个程序生命周期内。
3. 事件监听结果可以在任意的地方获取，并且处理后从事件中消除。

所以，事件类 Events 类图如下：



为了让事件的监听只产生一次就可以在全局内生效，所以Events类应该是一个单列模式。经管 python 中每个模块都是一个简化的单例模式，但是为了整个系统在后期不出现bug，所以对Event 显示规定为 单例模式。（参考：<http://python.jobbole.com/87791/>）

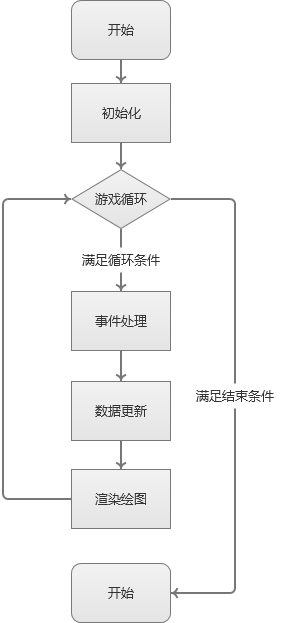
Singletion 类和与 Events类的关系图如下。Events 类继承 Singletion类，沿用 单例模式。在整个程序中有且只有一个 Events 类实例。为了简化 Events 类的使用。Events 在模块内实例化一个对象 “mEvents” 这个是Events 类的对象。也是整个程序中时间监听和反馈相关的入口。



### 程序主控 --- pyclass 模块

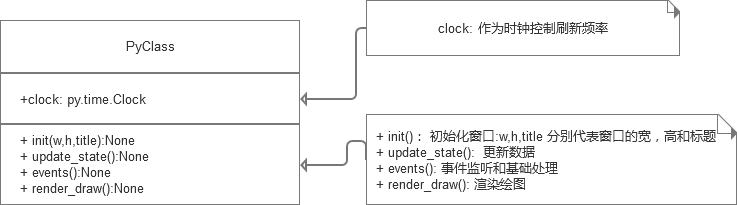
主控模块应该包含一个 PyClass 类作为主控制的基础类和一个入口函数 AppRun（） 函数.

入口函数的逻辑是：



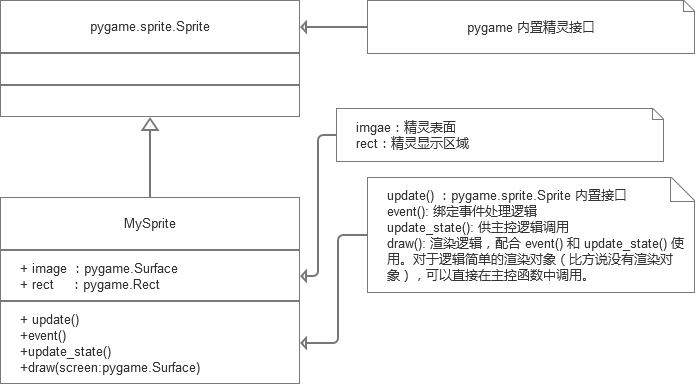
参考入口函数逻辑：PyClass 类应该也至少包含以下几个方法：

1. 初始化方法：init（），初始化窗口，和相关数据。
2. 事件处理：events（）, 开启事件，并做一些必要处理。
3. 数据更新：update\_state（func），响应用户输入，更新游戏状态。这个类应该接受一个 函数作为参数，响应精灵的触发事件。
4. 渲染绘图：render\_draw()：更新渲染窗口，并且控制刷新频率。PyClass 类图如下：



### 精灵类：渲染对象的接口要求

根据上面的主控逻辑和事件处理方式，定义如下精灵类：



默认继承自 pygame.sprite.Sprite 精灵接口，以便后期进行优化和更新。同理 image 、rect和update（） 也是这个接口的重要对象。使用 update\_state() 自定义的 更新区别与 默认接口的 update(),由 主控逻辑调用。