我们的UI框架是基于MVC实现的，C持有M和V，M层存放所有数据，V层获取所有组件，C层添加各种事件的监听，M层监听服务器发过来修改数据的消息，我们的C和V是挂在预制件上的，M层是从C层new出来的，在初始化C层时new出来M层、Get到V层和添加对监听事件、向服务器发送消息获取数据或者从缓存中读取数据，当我们打开面板时判断管理类中的保存面板的字典有没有这个面板，在没有的情况下加载面板获取面板上的C层，通过UIManager管理所有的C层，因为我们C层持有M和V，所以我们初始化数据时通过C层调用M层数据和V层的组件，然后将各个组件进行赋值显示，M层有一个公开的只能获取数据的方法，C层可以传入一个Id来获取具体的信息，有一个单独赋值和全部刷新的方法，比如背包格子 刷新格子时 格子上有自己格子物品的信息通过V层一个所有格子的集合进行查找，找到就刷新，而全部赋值就是刷新全部格子，

Ui框架中的层级划分，我们框架中的层级分为5层，第一层是我们的主界面的层 它主要显示主界面包括ui的九宫格；第二层是ui面板的层级，第三层是提示框的层级，第四层是3D层 对话框所展示的2D或者3D的模型所在的层，最后一层是引导层 新手引导所在的层

之后我们将所有的ui元素都做成了prefab，层级的预制件在游戏运行时加载一次之后就不删除了，在加载ui时我们有所有UI的配置表，里面包含了ui的Id（唯一），名字等，其中ui的Id和框架中的ui面板的（int）枚举值是保持一致的，这样可以通过Id去查找名字，当初始化时将配置表所有的信息读取出来，保存到字典当中，当我们去打开一个面板时，就从这个字典当中去查找这个面板的信息，从Ab包中加载出来

我们面板中的互斥是通过堆栈来实现 当我们打开一个面板时 在这个面板之前的面板就要区隐藏 将这个隐藏的面板加入到栈中 依次类推 所以打开过并没有手动关闭的面板都将存入栈中，当我们关闭正在显示的这个面板时、我们从栈顶中拿出一个面板进行显示，直到栈中没有面板

数据主要分为两部分 一种是你本地的数据 一种的服务器发过来的数据，本地数据一般是保存下来的配置表，例如所有的装备，道具等公共的数据，像服务器的数据 比如说背包这类服务器保存的数据 我们在点击开始游戏时服务器会将数据发送过来，之后根据数据格式解析数据，将数据保存到缓存区，在打开面板的时候就往缓存里找数据进行刷新

框架中的事件是我们将所有的事件都注册到一个类中，通过UIManager实例化这个类 那些用的时候通过初始化时添加事件的监听，在删除面板时将事件的监听给注销

对UI的优化

我们所有的UI中的图片都是打成图集来使用的，如果不使用图集的话，我们有上百张Image，那么就需要调用上百次DrawCall，那么这时候用图集的就可以减少DrawCall的数量，我们对图集的管理是通过单个面板来进行管理，较小的图集更容易管理和更新，避免一次性加载过多的资源导致性能下降和因单个图集问题影响整个资源

Graphic Rebuild

整个UI的rebuild（图像重构）问题，ReBuild调用多的话会导致性能的下降，因为每次Rebuild被调用都会重新计算Canvas层中的渲染状态，我们在对图像的纹理、颜色、材质等属性发生改变时都会触发Rebuild，所以我们要尽量对Rebuild少调用，对于要隐藏的面板这些UI，我们可以通过将面板移出Canvas（或通过对CanvasGroup的Alpha的透明度修改为0 使用该面板时移回或修改透明度、还有一种方法是SetActive设置True和Flase，但是设置为True时还是会调用Rebuild，所以这个方法我们会少用，这里SetActive只是关于UI方面）