早期的MVC模式，其中三者的定义可以简单的理解为：

Model，负责的是数据，这里的“数据”不仅限于数据本身，还包括处理数据的逻辑。

View，负责数据的表现形式，将数据及数据的变化呈现给用户。

Controller，负责用户的输入，将用户的命令转化成消息传递给model或view，是一个翻译者。

Model & View

Model和View的关系，也可以分为两种形式：

push model：View在model上将自己注册为数据的监听者，model的数据发生变化时会发送通知，view接到通知后用新数据更新自己。

pull model：View负责在它需要的时候调用model来获取当前最新的数据。

但不管是那种模式，model对于view都是没有感知的，push model是在数据变化的时候简单的发送广播，告之所有对该数据变化感兴趣的监听者，而pull model是view对model进行调用。因此在任何一种模式下，model都是不能直接操作view。

这种model-view模式也就是俗称的观察者模式，model是被观察者，view是观察者，当被观察者发送变化的时候，通知注册在它上面的所有观察者。这样设计带来的另一个好处就在于多个view可以监听同一个model的变化。

Controller & View

关于Controller和View的关系，Controller是绑定在View上面的，意思是用户任何在View上的操作（例如点击按钮等）都会调用Controller上的一个回调方法。其实也意味着View是持有Controller的引用的，当用户做相应操作时，是由View来调用合适的Controller方法的，而Controller对View的操作在早期概念中则不太明确。

Controller & Model

Controller是可以向Model直接通信的。例如用户点击了删除按钮，那么Controller将用户的这个操作翻译成“用户需要删除这条数据”的消息传递给Model，Model负责删除数据，然后通知View来更新页面以告知用户数据的变化。

现代MVC模式

与经典MVC模式不同，很多现代系统设计中，如Apple Cocoa框架，最大的改变在于将Controller的位置放在了Model和View之间。

主要区别就在于Controller的位置变了：Controller将消息传递给Model，M处理完数据后是先将数据的变化通知给C，再由C来通知View来变化视图的。也就是说Controller变成了在Model和View之间双向传递数据的中间协调者，关系变成了： View <-> Controller <-> Model 。

Model & View

Model和View之间没有了任何关系，所有通信都是通过Controller传递。

Model & Controller & View

View通过Controller向model传递用户操作的消息，而model在处理完数据后通过Controller来向View来传递结果。Controller从经典MVC模式中的单向翻译官变成了双向的中间人。