

   直接上图，其中model 和view大家经常写，就不说了

有人可能并不清楚controller 到底是啥，其实就是经常写的 data source ,delegate,outlet什么的

先撇开那些乱七八糟的箭头单看他们之间的分界线，view与controller、model与controller之间的线是虚的，而model与view之间的是双黄线，考过驾照的知道越过这个线意味着什么吧。

控制器与视图之间的大部分连线都可以在Xcode中通过按住ctrl来连接。例如outlet，例如delegate。有了这些连线，他们之间的沟通算是畅通无阻了。

1）用户在view上的操作可以直接告诉控制器让控制器去响应

2）view需要的代理也可以指派给控制器让控制器去完成

3）view需要的数据源也需要控制器来提供

总之，控制器很忙，因为它非常被动。

控制器还需要一个角色的参与：model

顾名思义，模型。我觉得使用[asp.net](http://asp.net/)里mvc的domain来描述比较好理解，领域。例如你要开发一个图书管理系统，模型里描绘的就是图书的书名、售价、作者，谁借走了，几号借走的，一切关于这个应用领域的数据。

我们可以给模型一个简单的定义：它描述了一个应用所有需要抽象出来的数据结构和这些数据之间的关系，还有他们获取数据需要通过的途径（本地的、远程的）等。

有了模型，控制器想啥时候去拿数，想怎么拿都行。

仔细观察上面的图，有控制器到model的箭头，没有model到控制器的箭头。因为模型不需要知道控制器的存在，这样可以减少耦合，提高代码复用可能性。把你定义好的model挪到别的项目中，还可以使用。

但是问题来了，当model数据已经更新了，被修改了，怎么通知相关的控制器？注意model上方有个天线，高端！之所以使用无线的，是因为model不知道更改要发给谁，但是model可以广播，model的观点：谁关心我的更改让谁收听去，我对谁收听了我不感兴趣。

广播的方式有两种：Notification 和 KVO（都是设计模式，本系列接下来会详述）

好了，他们几个之间的关系基本有个概念了。mvc模式就是这样，把原本杂乱无章的类，分为三堆，严格监管，按规则行事。

还是那句话：

说到底一切都是为了使类之间的耦合性更松散。好的代码应该对**扩展开放**，对**修改关闭**。

以下是学院派喜欢看的概念

一、MVC模式概述

模型-视图-控制器（MVC模式）是一种非常经典的软件架构模式，在UI框架和UI设计思路中扮演着非常重要的角色。从设计模式的角度来看，MVC模式是一种复合模式，它将多个设计模式在一种解决方案中结合起来，用来解决许多设计问题。MVC模式把用户界面交互分拆到不同的三种角色中，使应用程序被分成三个核心部件：Model（模型）、View（视图）、Control（控制器）。它们各自处理自己的任务：

（1）模型：模型持有所有的数据、状态和程序逻辑。模型独立于视图和控制器。

（2）视图：用来呈现模型。视图通常直接从模型中取得它需要显示的状态与数据。对于相同的信息可以有多个不同的显示形式或视图。

（3）控制器：位于视图和模型中间，负责接受用户的输入，将输入进行解析并反馈给模型，通常一个视图具有一个控制器。

MVC模式将它们分离以提高系统的灵活性和复用性，不使用MVC模式，用户界面设计往往将这些对象混在一起。MVC模式实现了模型和视图的分离，这带来了几个好处。

（1）一个模型提供不同的多个视图表现形式，也能够为一个模型创建新的视图而无须重写模型。一旦模型的数据发生变化，模型将通知有关的视图，每个视图相应地刷新自己。

（2）模型可复用。因为模型是独立于视图的，所以可以把一个模型独立地移植到新的平台工作。

（3）提高开发效率。在开发界面显示部分时，你仅仅需要考虑的是如何布局一个好的用户界面；开发模型时，你仅仅要考虑的是业务逻辑和数据维护，这样能使开发者专注于某一方面的开发，提高开发效率。