iOS中MVC设计模式

在组织大型项目的代码文件时，我们常用MVC的思想。MVC的概念讲起来非常简单，就和对象（object）一样。但是理解和应用起来却非常困难。今天我们就简单总结一下MVC设计理念。

MVC(Model View Controller)模型(model)－视图(view)－控制器(controller)：

MVC本来是存在于Desktop程序中的，M是指数据模型，V是指用户界面，C则是控制器。使用MVC是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。比如一批统计数据你可以分别用柱状图、饼图来表示。C存在的目的则是确保M和V的同步，一旦M改变，V应该同步更新，从例子可以看出MVC就是Observer设计模式的一个特例。

MVC是一个设计模式，它强制性的使应用程序的输入、处理和输出分开。使用MVC应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。分层概念

（一）模型对象

模型对象封装了应用程序的数据，并定义操控和处理该数据的逻辑和运算。例如，模型对象可能是表示游戏中的角色或地址簿中的联系人。用户在视图层中所进行的创建或修改数据的操作，通过控制器对象传达出去，最终会创建或更新模型对象。模型对象更改时（例如通过网络连接接收到新数据），它通知控制器对象，控制器对象更新相应的视图对象。在MVC的三个部件中，模型拥有最多的处理任务。例如它可能用象EJBs和ColdFusion Components这样的构件对象来处理数据库。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

（二）视图对象

视图对象是应用程序中用户可以看见的对象。视图对象知道如何将自己绘制出来，并可能对用户的操作作出响应。视图对象的主要目的，就是显示来自应用程序模型对象的数据，并使该数据可被编辑。尽管如此，在 MVC 应用程序中，视图对象通常与模型对象分离。在iOS应用程序开发中，所有的控件、窗口等都继承自 UIView，对应MVC中的V。UIView及其子类主要负责UI的实现，而UIView所产生的事件都可以采用委托的方式，交给UIViewController实现。

（三）控制器对象

在应用程序的一个或多个视图对象和一个或多个模型对象之间，控制器对象充当媒介。控制器对象因此是同步管道程序，通过它，视图对象了解模型对象的更改，反之亦然。控制器对象还可以为应用程序执行设置和协调任务，并管理其他对象的生命周期。

控制器对象解释在视图对象中进行的用户操作，并将新的或更改过的数据传达给模型对象。模型对象更改时，一个控制器对象会将新的模型数据传达给视图对象，以便视图对象

代理，block，单例，通知）(3592)

3. IOS中的UINavigationController（导航控制器）(2816)

4. OC中NSDictionary（字典）、NSMutableDictionary（可变字典）、NSSet（集合）、NSMutableSet（可变集合）得常用方法(2234)

5. iOS开发UI中懒加载的使用方法(2047)

6. iOS中MVC设计模式(1996)

7. iOS高级编程之XML，JSON数据解析(1777)

8. [iOS]MVVM-框架介绍(1758)

9. iOS开发之Pch预编译文件的创建(1732)

10. iOS 项目中用到的一些开源库和第三方组件(1590)

推荐排行榜

1. iOS 页面间几种传值方式（属性，代理，block，单例，通知）(8)

2. iOS 项目中用到的一些开源库和第三方组件(8)

3. iOS App上架流程(7)

4. [iOS]MVVM-框架介绍(3)

5. Objective-C 编码规范(2)

6. iOS数据持久化存储之归档NSKeyedArchiver(2)

7. iOS七大手势之(平移、捏合、轻扫、屏幕边缘轻扫)手势识别器方法(2)

8. 浅析Objective-C字面量(2)

9. iOS高级编程之XML，JSON数据解析(2)

10. Swift（二）控制流(2)

可以显示它。

为什么要使用 MVC

首先，最重要的一点是多个视图能共享一个模型，现在需要用越来越多的方式来访问你的应用程序。对此，其中一个解决之道是使用MVC，无论你的用户想要Flash界面或是WAP 界面；用一个模型就能处理它们。由于你已经将数据和业务规则从表示层分开，所以你可以最大化的重用你的代码了。

由于模型返回的数据没有进行格式化，所以同样的构件能被不同界面使用。例如，很多数据可能用HTML来表示，但是它们也有可能要用Adobe Flash和WAP来表示。模型也有状态管理和数据持久性处理的功能，例如，基于会话的购物车和电子商务过程也能被Flash网站或者无线联网的应用程序所重用。

因为模型是自包含的，并且与控制器和视图相分离，所以很容易改变你的应用程序的数据层和业务规则。如果你想把你的数据库从MySQL移植到Oracle，或者改变你的基于RDBMS数据源到LDAP，只需改变你的模型即可。一旦你正确的实现了模型，不管你的数据来自数据库或是LDAP服务器，视图将会正确的显示它们。由于运用MVC的应用程序的三个部件是相互独立，改变其中一个不会影响其它两个，所以依据这种设计思想你能构造良好的松耦合的构件。

对我来说，控制器也提供了一个好处，就是可以使用控制器来联接不同的模型和视图去完成用户的需求，这样控制器可以为构造应用程序提供强有力的手段。给定一些可重用的模型和视图，控制器可以根据用户的需求选择模型进行处理，然后选择视图将处理结果显示给用户。

MVC的优点

（一）、低耦合性

视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

（二）、高重用性和可适用性

随着技术的不断进步，现在需要用越来越多的方式来访问应用程序。MVC模式允许你使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，而这些表示所需要的命令是改变视图层的实现方式，而控制层和模型层无需做任何改变。

（三）、较低的生命周期成本

MVC使开发和维护用户接口的技术含量降低。

（四）、可维护性

分离视图层和业务逻辑层也使得应用更易于维护和修改。

（五）、有利于软件工程化管理

由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化管理程序代码。

MVC的缺点：

MVC的缺点是由于它没有明确的定义，所以完全理解MVC并不是很容易。使用MVC需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。

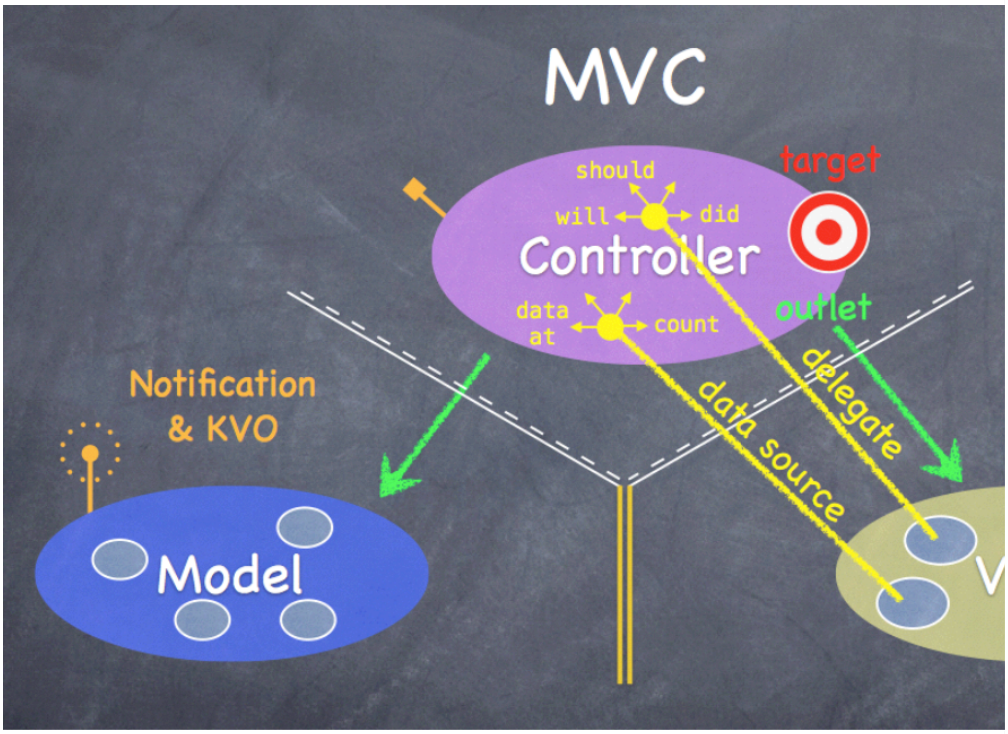
你将不得不花费相当可观的时间去考虑如何将MVC运用到你的应用程序，同时由于模型和视图要严格的分离，这样也给调试应用程序带来了一定的困难。每个构件在使用之前都需要经过彻底的测试。一旦你的构件经过了测试，你就可以毫无顾忌的重用它们了。

根据开发者经验，由于开发者将一个应用程序分成了三个部件，所以使用MVC同时也意味着你将要管理比以前更多的文件，这一点是显而易见的。这样好像我们的工作量增加了，但是请记住这比起它所能带给我们的好处是不值一提。

MVC并不适合小型甚至中等规模的应用程序，花费大量时间将MVC应用到规模并不是很大的应用程序通常会得不偿失。

MVC设计模式是一个很好创建软件的途径，它所提倡的一些原则，像内容和显示互相分离可能比较好理解。但是如果你要隔离模型、视图和控制器的构件，你可能需要重新思考你的应用程序，尤其是应用程序的构架方面。如果你肯接受MVC，并且有能力应付它所带来的额外的工作和复杂性，MVC将会使你的软件在健壮性，代码重用和结构方面上一个新的台阶。

IOS MVC设计模式：



图中有几条线把这三部分划分开，有黄线，虚线，和白色的实线。我们把它们想象成路标。你可以看到，在M和V之间有两条黄线，这表示什么呢？它意味着你不能 穿越这黄线，任何一个方向都不行，即M和V完全分离。在图的上部，你可以看到白色的虚线，它意味着你可以自由的穿越它，只要是安全的。那白色的实线呢？它代表你可以穿越，但你必须要买票，或者交点过路费。

首先， 我们来看C和M之间的绿色箭头，这箭头的方向就代表着“发起对话”的方向，也就是说，发起对话的是C，而做出回答的是M。C可以问M各种各样的问题，但M 只是回答C的问题或要求，它不可以主动的向C要求什么。还记得虚线是畅通无阻的意思吧，所以，C知道M的所有的事情,如果用代码来说明这件事情，就是 说，C可以导入M的头文件或是M的接口（API）。因为C可以通过M的API，所以它就可以肆无忌惮的向M要求这要求那了。

我们再来看看另外的一个绿色箭头，它是在C和V之间，和前一个绿色箭头的意义一样，它代表C可以直接地向V进行交流。你可以想想，C要把V放到屏幕 上，并设置V的属性，告诉它们什么时候从屏幕上消失，把它们分成组等等。如果C不能自由的向V发号施令的话，程

序的显示将会多么的困难,所以，C可以毫无 限制地向V说话。

可能你已经注意到了，这个箭头上还有outlet(输出口)，outlet可以看作是从C指向V的指针，它在C中被定义。outlet给我们提供了很大的 方便，它使我们在C的内部就可以轻松准确地向V施令。C可以拥有很多的outlet，可以不止一个，这也使它可以更高效的和V进行交流。

那M和V之间可以交流么？还记得黄线的意思么？完全不可以通过，所以我们是不允许M和V进行交流的。这是因为我们不希望这三部分之间有过多的交流，你想想，假如V在显示时出现了问题，比如有一个图形没有显示出来，我们就要去查找错误，因为C可以和V交流，M也可以和V交流的话，我们就要去检查两个部分。 相反的，只有C可以和V交流的话，在出错时，我们就只需要去C那里查找原因，这样查找错误不就很是简单了么？所以，我们不允许M和V之间有直接的联系，这 也是在它们两之间有两根黄线的原因。 总结下来也就是以下三点：

（1）、Model和View永远不能相互通信，只能通过Controller传递。

（2）、Controller可以直接与Model对话（读写调用Model），Model通过Notification和KVO机制与Controller间接通信。

（3）、Controller可以直接与View对话，通过outlet,直接操作View,outlet直接对应到View中的控件,View通过action向Controller报告事件的发生(如用户Touch我了)。Controller是View的直接数据源（数据很可能是Controller从Model中取得并经过加工了）。Controller是View的代理（delegate),以同步View与Controller。

我们接下来讨论V是如何向C发送信息的。V对C的交流有三种不同的方式：

第一种我们称为目标操作(target-action)。

它是这样工作的，C会在自己的内部“悬挂”一个目标(target)，如图中的红白相间的 靶子，对应的，它还会分发一个操作(action，如图中的黄色箭头)给将要和它交流的视图对象(可能是屏幕上的一个按钮)，当按钮被按时，action 就会被发送给与之对应的target，这样V就可以和C交流了。但是在这种情况下，V只是知道发送action给对应的target,它并不知道C中的 类，也不知道它到底发送了什么。target-action是我们经常使用的方法。

第二种方式我们叫做委托(delegate)。

有时候，V需要和C进行同步，你知道，用户交互不仅仅是什么按按钮，划滑块，还有很多种形式。好了， 让我们来看看图中的delegate黄色箭头，你发现箭头上又分出了四个小箭头：should，did，will，还有一个没标注的。绝大部分的 delegate信息都是should，will，did这三种形式。和英文意思相对应，should代表视图对象将询问C中的某个对象“我应该这么做 么？”，举个例子，有一个web视图，有人点击了一个链接，web视图就要问“我应该打开这个链接么？这样做安全么？”。这就是should信息。那 will和did呢？will就是“我将要做这件事了”，did就是“我已经做了这件事”。C把自己设置为V的委托(delegate),它让V知道：如 果V想知道更多的关于将如何显示的信息的话，就向C发送delegate信息。通过接受V发过来的delegate信息，C就会做出相应的协调和处理。还有一点，每个V只能有一个delegate。

第三种方式就是数据源(datasource)，

V不能拥有它所要显示的数据，记住这点非常重要。V希望别人帮助它管理将要显示的数据，当 它需要数据时，它就会请求别人的帮助,把需要的数据给它。再者，iphone的屏幕很小，它不能显示包含大量信息的视图。看图中的datasource箭 头，和delegate类似，V会发送cout，data at信息给C来请求数据。

对于不同的UIView，有相应的UIViewController，对应MVC中的C。例如在iOS上常用的UITableView，它所对应的Controller就是UITableViewController。

分类: UI, iOS

标签: IOS, 移动开发, UI, MVC, 设计模式

Running2Snail
关注 - 31
粉丝 - 86
+加关注

0 0

« 上一篇: iOS开发UI中懒加载的使用方法
» 下一篇: iOS 页面间几种传值方式（属性，代理，block，单例，通知）

posted @ 2015-05-02 18:56 Running2Snail 阅读(1997) 评论(0) 编辑 收藏