视图

多层架构

创建视图

视图控制器

MVC应用

应用实例

视图

视图是构建iOS app界面的基础和核心。

所有的app都是由一个或多个视图组成的。

视图控制器：管理视图所有相关事务。

包括：定义视图的用户界面、处理视图间的交互以及管理视图与数据的交互。

本节只涉及视图的相关概念和基础知识。

后续课程还会涉及其它的常用视图控件：

文本编辑框

导航控制器

集合视图

Web视图

表单

应用生命周期

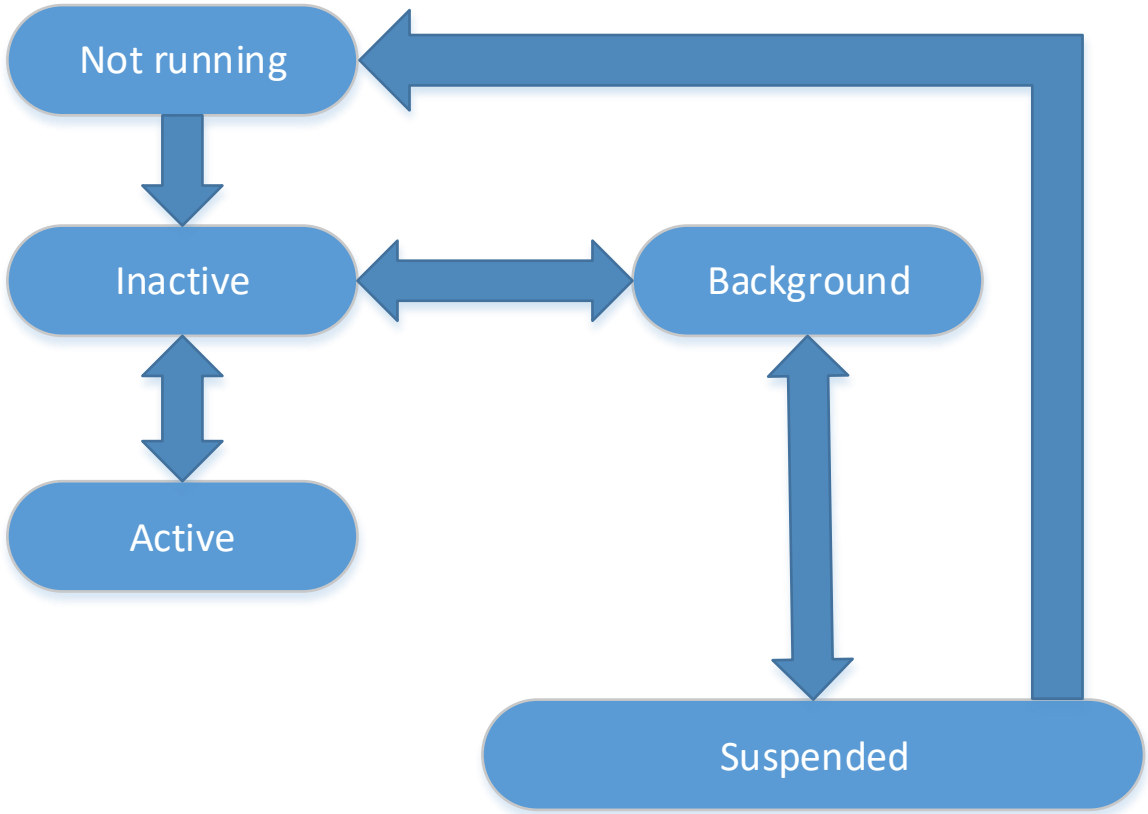
苹果应用总是处于5个状态之中。

Not running（未运行）
Inactive（未激活）
Background（后台运行）
Active（激活）
Suspended（被挂起）

系统根据外界动作将应用从一个状态切换到另一个。

代理委托机制

通过应用中的代理委托机制，大部分的状态转移发生时，都会调用应用中的相关处理方法。通过这些方法，开发人员可以对应用的状态变化进行适当处理。



视图的多层结构：Windows和Views

通过Windows对象和Views（视图）对象将应用系统界面呈现到屏幕上。

Windows

Windows是Views的基础容器，是不可见的。

Views

Views定义了屏幕上可见的部分。

任何一个应用至少有一个Window和一个View。

UIKit中的View

系统自带的UIKit框架提供了大量可以直接使用的View：button、label、table view、picker view、scroll view等。

如有特殊需求，可自行定制View。

视图的多层结构：父子视图

UIView是View的基类。

View负责管理应用系统的矩形区域。

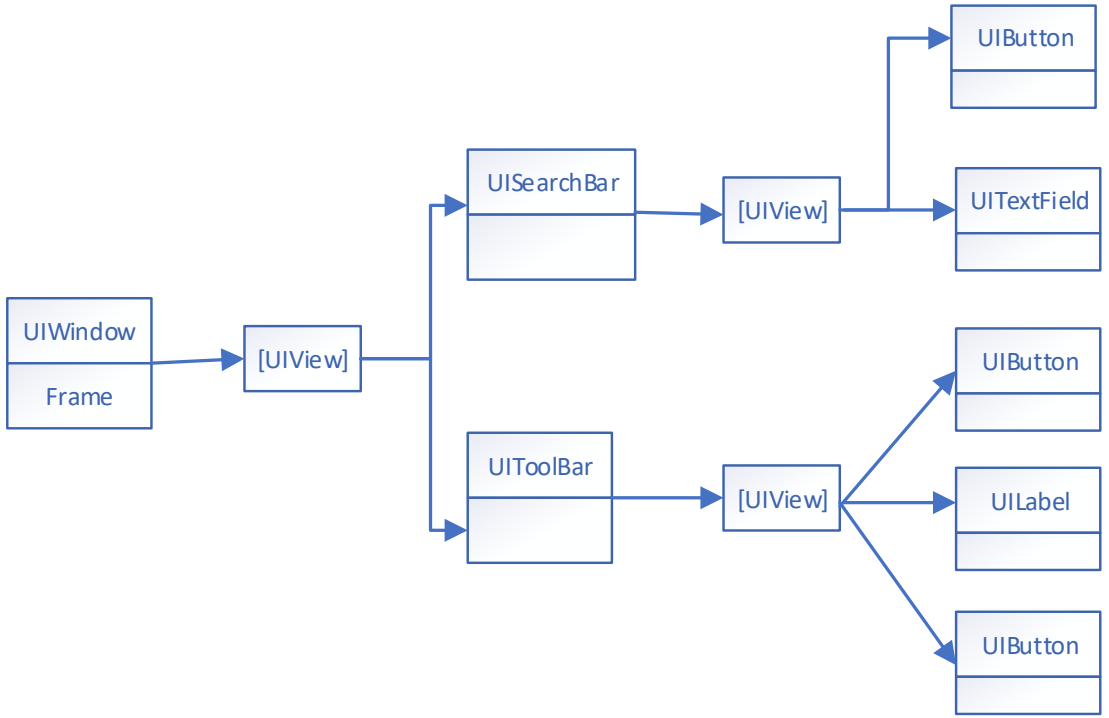
包括：绘制界面、处理多触点事件以及子视图subview的布局。

视图容器

一个视图除了显示自己的内容外，还可以作为其它视图的容器，用来构建复杂的显示内容。

父子视图

当一个视图包含另一个视图时，称为父子关系。
作为容器的视图称为父视图，加入容器中的视图称为子视图。
在编码中，子视图表示为subview，而父视图表示为superview。
在界面显示的时候，如果subview是不透明的，那么superview中subview所占的部分会被覆盖。如果改变一个superview的尺寸，那么superview中所有的subview的尺寸和相对位置都会受到影响。



视图的多层结构

用户界面的构成

在构建用户界面时，往往要使用多个View来搭建，每一个View负责界面的一部分区域和显示特殊的内容。

视图的多层结构

例如：UIView、UISearchBar和UIToolBar都是容器。这些View共同显示一个完整的用户界面，它们之间的嵌套关系构成视图的多层结构。

事件的传递和处理

在多层架构的视图中，当一个subview区域内发生了一个触碰动作，系统会将这个触碰事件信息直接发送给这个subview来处理。

如果subview没有处理这个动作，那么这个触碰事件将会向上传递给它的superview来处理。以此类推，触碰事件会一级一级向上传递，直到被处理。

视图的坐标系统

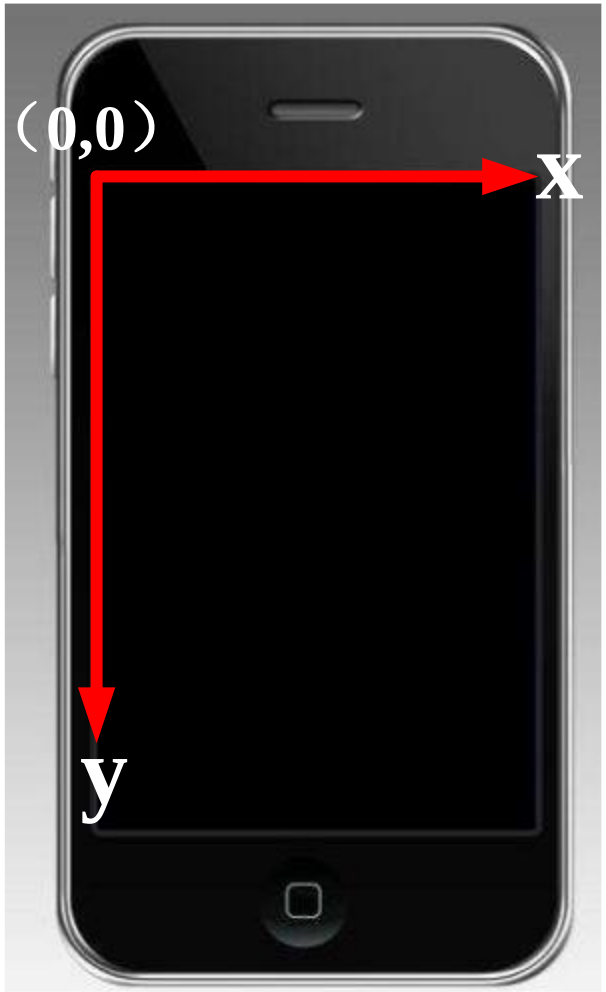
坐标原点

坐标原点的坐标为（0,0），在屏幕的左上角顶点，x坐标轴为自左向右，y坐标轴为自上而下。

坐标值是浮点数，从而支持精确的定位。

绝对坐标和相对坐标

每一个视图，无论是superview还是subview，在视图内部都遵循坐标系统定义，不同的是，屏幕的坐标是绝对坐标，而子视图的坐标是相对坐标。



视图的创建

编码方式创建视图

相对繁琐

对视图的控制精确，非常灵活

需要非常熟悉视图的实现机制

Interface Builder创建视图

强大的“所见即所得”应用程序界面构建工具

最简便的方式，备受开发者青睐

这两种方式创建视图的效果和底层代码都是一样的。

类似于：高级语言和汇编语言的关系

视图的创建：Interface Builder

创建方法

在Interface Builder中，可以直接向界面中添加视图，并且设置视图在界面中所处的层次。

选中要编辑的目标视图，然后通过视图的属性编辑面板来设置视图的各项属性值，以及将视图的各种行为与编写好的行为处理代码关联起来。

底层实现机制

在Interface Builder中编辑的视图都是某个视图类的实例，因此在设计阶段对其进行的编辑结果（属性和行为关联）将会被保存到一个nib文件（保存对象状态和属性的文件）中，从而保证视图在设计时和运行时保持一致。

视图控制器与MVC架构

MVC设计模式

“M”是模型Model

“V”是视图Views

“C”是视图控制器View Controllers。

视图控制器

视图控制器是MVC架构中的核心，也是应用程序的关键所在。

任何一个应用程序都有一个或多个视图控制器。

每一个视图控制器负责管理部分的用户界面以及用户界面行为与数据的交互。

视图控制器还负责不同用户界面之间的切换动作。

应用程序的核心设计就在于视图控制器代码的编写。

视图控制器：UIViewController

UIViewController

该类定义了一系列的方法和属性，用来管理视图、处理事件、切换视图控制器以及协同应用程序的各个模块。

自定义视图控制器

在设计应用程序时，可以通过继承UIViewController生成子类subclass，并在子类中添加自定义代码来实现应用程序的独特行为。

内容和容器视图控制器

内容视图控制器，管理所有相关视图，包括根视图及子视图。

容器视图控制器，只管理它自己的视图，即：根视图。

视图控制器的作用

数据调度

管理多层结构的视图

管理视图资源的创建和释放

解决视图在不同硬件环境下显示的适配问题

处理所有相关视图（根视图和子视图）的用户交互事件

管理多层结构视图

根视图

每一个视图控制器都对应唯一的根视图。

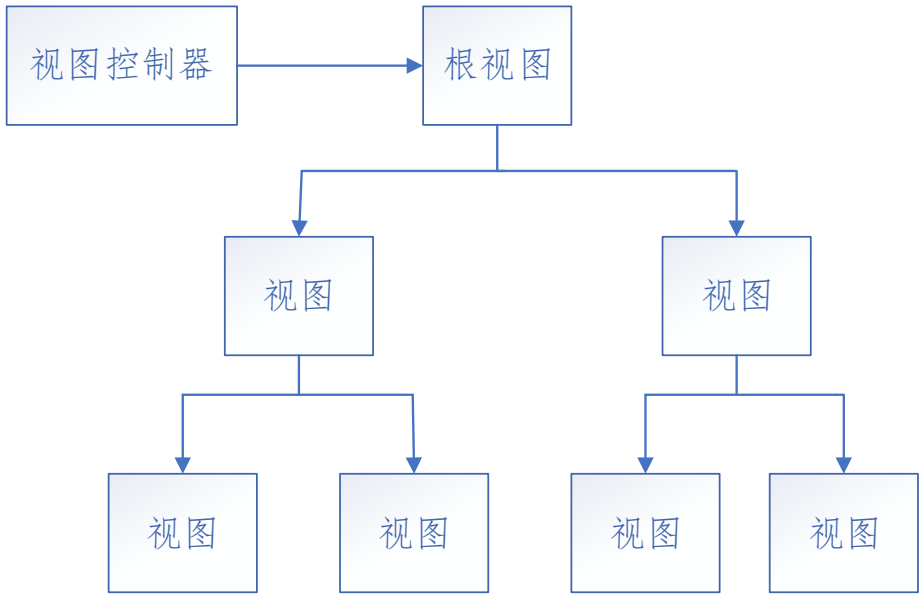
根视图通常对应着多个子视图。

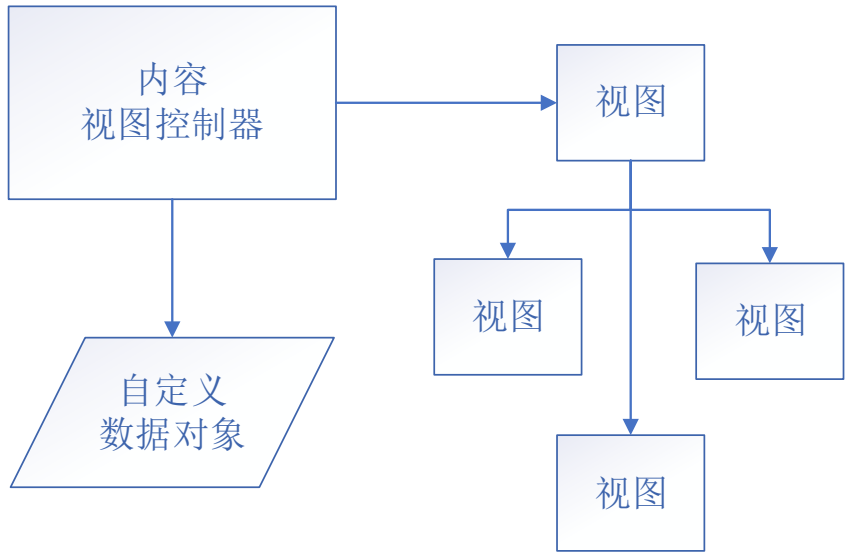
视图控制器对视图的管理

视图控制器总有一个引用指向它的根视图。

每个视图都有一个或多个强引用指向其子视图。

通过这种方式，视图控制器来实现对多层结构视图的管理。





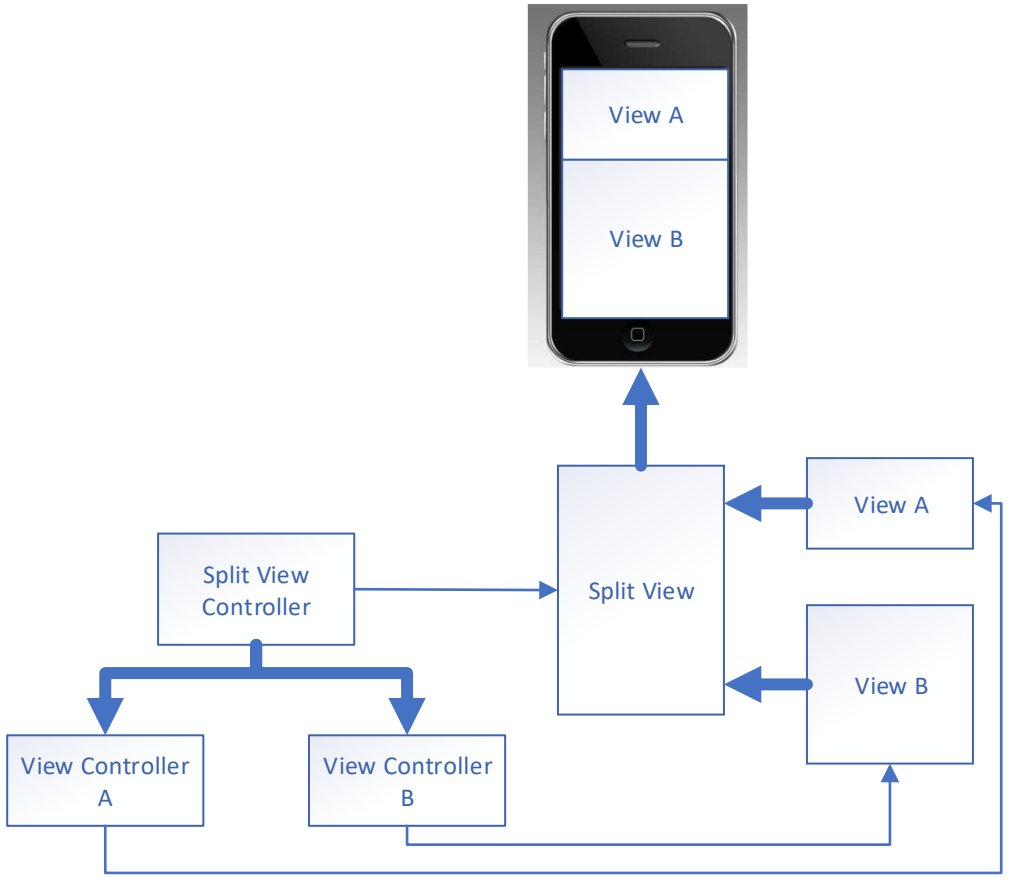
数据调度

视图控制器是视图和数据模型之间的媒介

视图控制器的基类UIViewController提供了一系列属性和方法来管理应用程序界面的可视化呈现。

在开发视图控制器时，通过继承UIViewController来生成子类，然后在子类中定义需要使用的新的数据变量。

视图控制器实例


View A
View B
Split View Controller
Split View
View A
View B
View Controller A
View Controller B

根视图控制器与Window

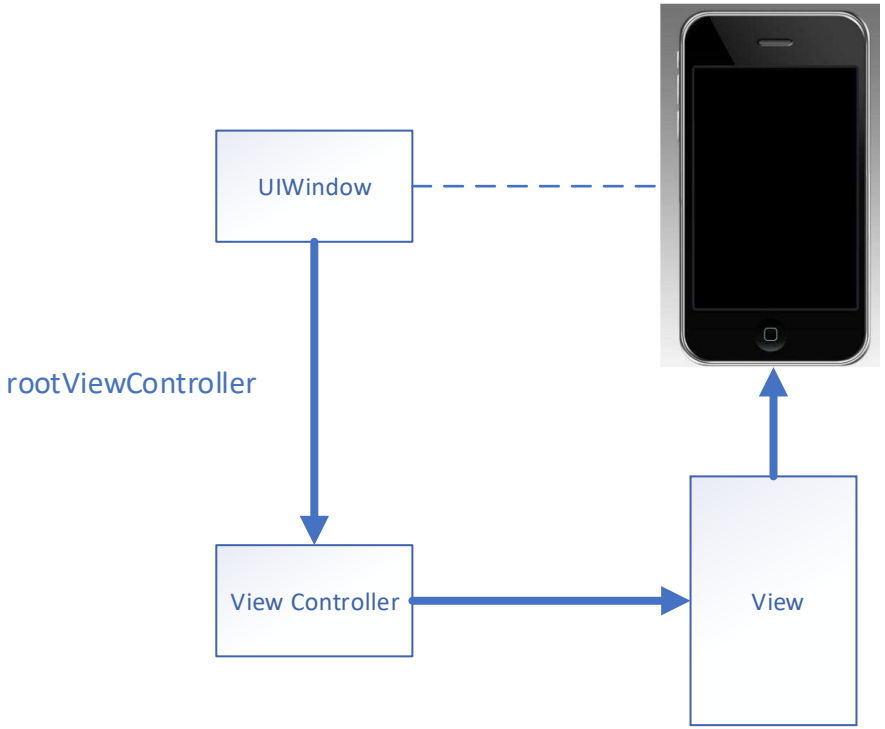
视窗Window

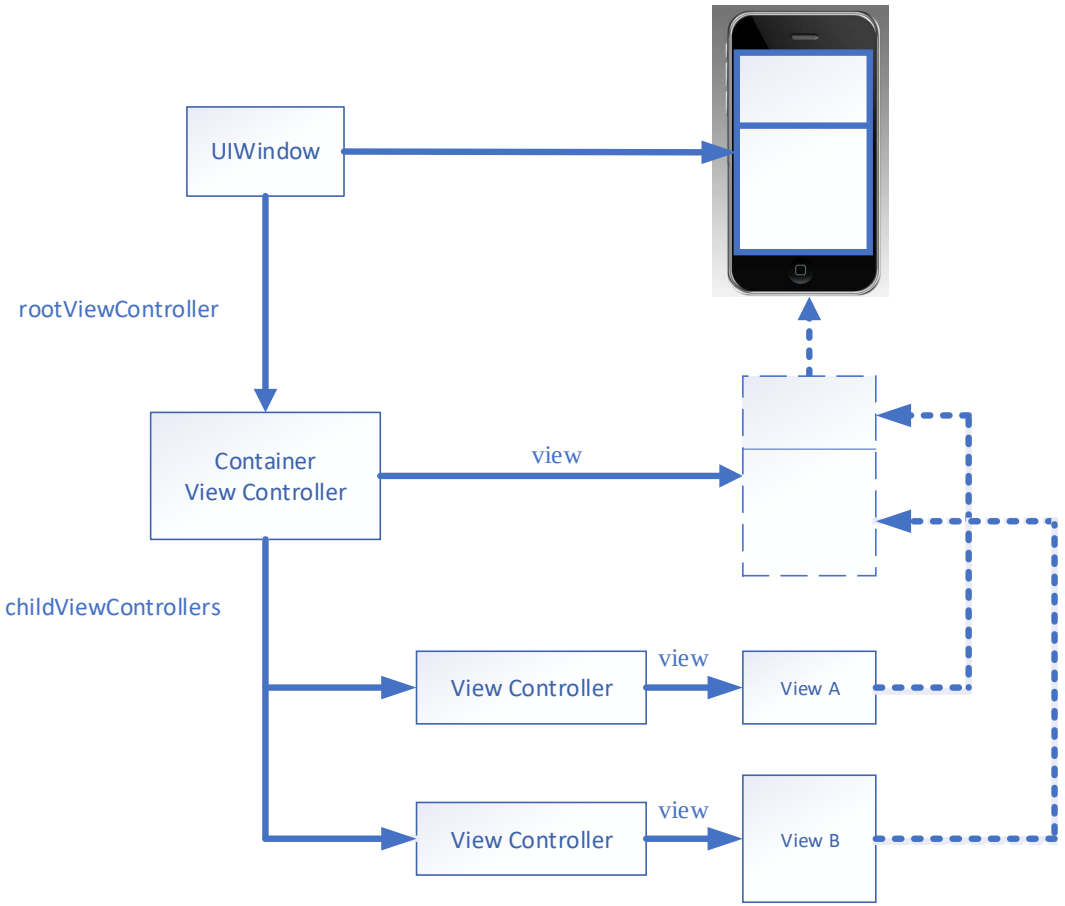
每个应用程序对应一个Window。

Window本身是不可见的，也没有可显示的内容。

每个视窗Window都仅有一个根视图控制器，它负责用根视图来绘制整个视窗的界面。

根视图控制器定义了应用程序启动时呈现给用户的初始界面内容。





容器的视图控制器

容器视图控制器

主要用来组合复杂的用户界面，并且使该界面易于维护和重用。

容器视图控制器将一个或多个子视图控制器的内容组合到一起，形成最终的用户界面。

常见的容器视图控制器

UINavigationController

UISplitViewController

UIPageViewController

容器视图总是将其所在的空间占满。

容器视图控制器在视窗中的作用

通常作为根视图控制器

也可以作为内容视图来呈现

或作为别的容器视图的一部分来使用

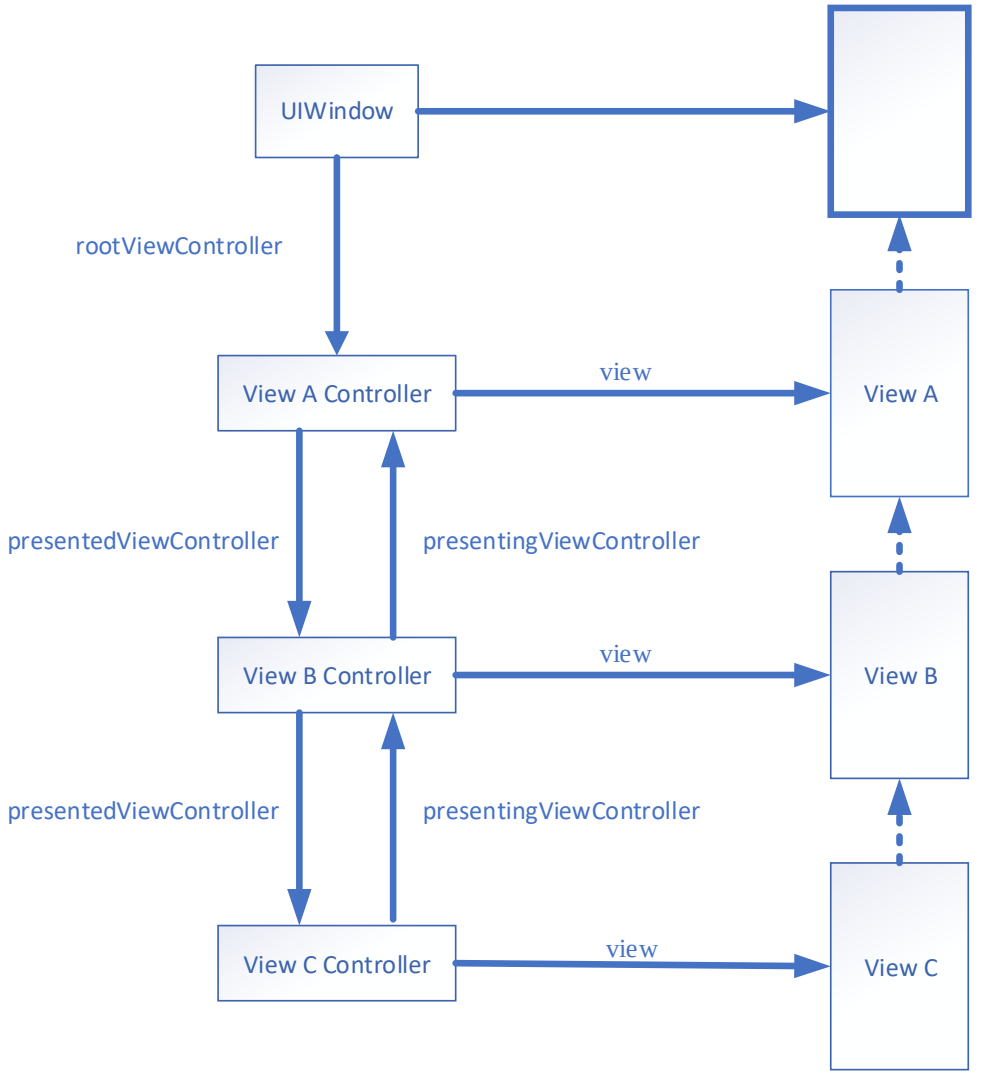
切换当前视图控制器的过程

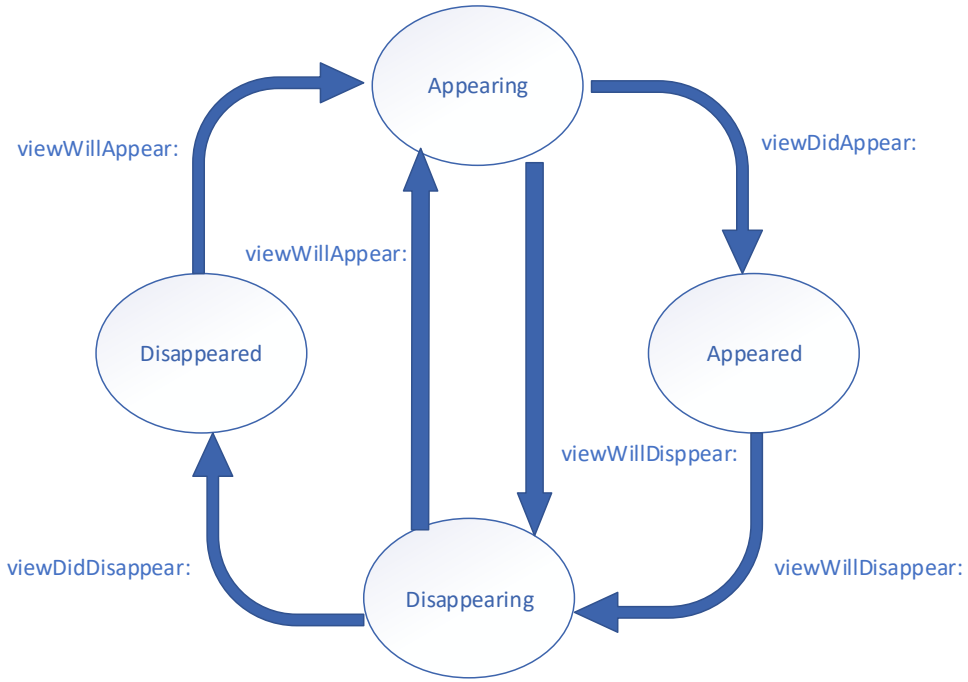
当前界面的切换

应用程序运行时，常常会切换不同的显示界面，用一个新的界面替换当前正在显示的界面。

切换界面是为了显示不同内容而频繁进行的动作

例如，为了登录系统，常常需要在当前界面上弹出一个用户输入账号的页面。当切换当前显示界面时，就会创建一个即将显示的视图与当前视图之间的引用关系。





视图的状态切换

在App运行时，处于显示状态的视图会不断切换。

视图的控制器会自动调用不同的方法来响应视图当前的状态变化。

当视图将要显示到屏幕上时调用方法viewWillAppear(_:)来为视图做一些准备工作；

当视图将要从屏幕上消失时调用方法viewWillDisappear(_:)来保存变化或者状态信息等。

当视图显示到屏幕后将调用方法viewDidAppear(_:)，而当视图从当前显示状态切换到后台时将调用方法viewDidDisappear(_:)。

MVC设计模式

MVC定义

Model-View-Controller:模型-视图-控制器

一种经典的设计模式，有广泛的应用，特别是在面向对象的程序设计中。

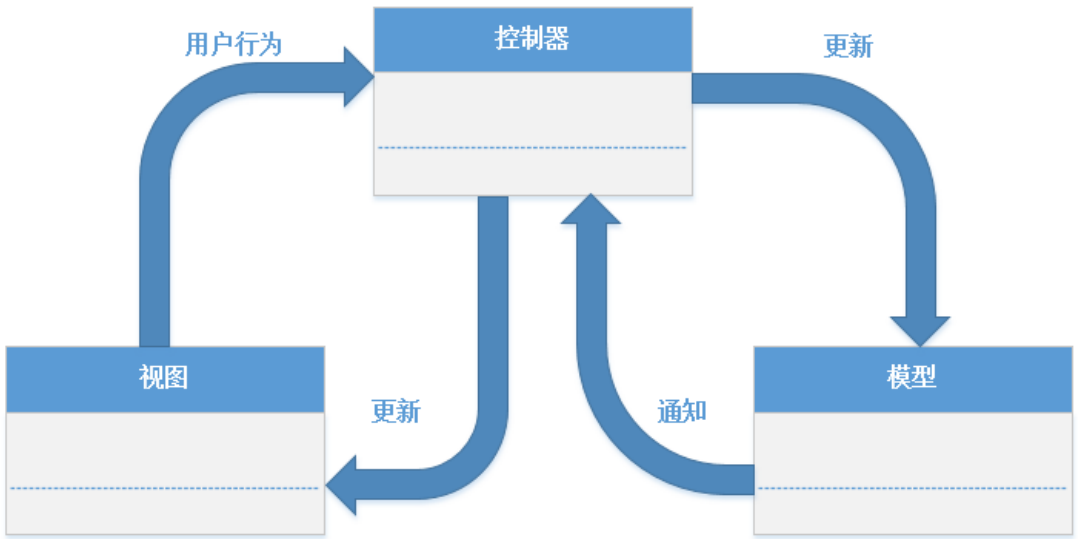
MVC设计模式通过灵活的接口定义，为程序提供了很好的可重用性和可扩展性。

MVC的三层架构

MVC设计模式将一个应用中的对象按照其作用分配到三个不同抽象层中，即：模型层、视图层和控制层。

MVC不仅规定了应用中每一个抽象层对象的行为，也定义了不同抽象层对象之间的通讯方式。

MVC的一个抽象层就是同类对象的集合。



MVC的运行机制

控制层

即：控制器，负责与视图层和模型层的通讯。

视图层

视图层是应用系统与用户交互的窗口，它收集用户行为，并将其汇报给控制器。控制器根据视图层汇报过来的用户行为，将信息更新（或查询）的指令传达给模型层。

模型层

模型层根据控制器发过来的指令对相关数据进行更新（或查询），并将处理结果通知给控制器。控制器再根据模型层反馈的信息，发送信息更新的指令给视图层。最终，由视图层将更新的结果展示给用户。

应用MVC

两点注意

MVC中的每一个抽象层通常都是由一系列对象构成的（极端情况为每一层仅有一个对象）。不同抽象层之间的通讯，实际是不同抽象层中对象之间的通讯。

MVC将应用系统的行为划分成三个抽象层次：

缺点：增加了系统复杂度和代码规模。

优点：提高了系统的可重用性、可维护性以及可扩展性。

为什么要用MVC架构

App开发中需求常常变化频繁，要求快速迭代开发，因此系统中大部分对象都要有很好的可重用性和可维护性。

应用系统还要有很好的可扩展性。

苹果官方的Cocoa体系架构就是基于MVC设计模式的。在苹果应用系统的开发过程中，会大量使用Cocoa提供的API，为了实现App无缝的与Cocoa API协同，App本身也要采用MVC架构。

MVC：模型层

模型层由一系列模型对象组成的。

模型对象

模型对象封装了对数据的定义、计算以及各种操作。

一个模型对象可以描述一个系统设置信息，也可以表示通讯录中的一个联系人信息。

模型对象关系图

一个模型对象与其它模型对象之间有一对一或者一对多的关系。因此，模型层可以表示为一系列模型对象的关系图。

应用系统中大部分数据都应该存储于模型对象中，这些数据将在应用运行过程中被反复使用。

MVC：视图层

视图层是由一系列用户可见的视图对象组成的。

视图对象能够绘制自身和相应用户的动作。

主要作用

向用户展示模型层的数据，并提供给用户操作这些数据的接口。

标准视图库

MVC架构中的视图层是从模型层解耦出来的，视图对象具有很好的可重用性和可配置性，因而苹果官方可以提供给开发者包含有标准化视图对象集合的开发框架UIKit和AppKit。

数据通讯

视图层通过控制层来与模型层进行通讯。视图对象通过控制器对象获得模型对象中数据信息，并呈现给用户。

MVC：控制层

控制层是由一系列控制器对象组成的。

通讯媒介

一个控制器对象是一个或多个视图对象与一个或多个模型对象之间进行通讯的媒介。

控制层的作用

视图对象通过控制器对象获得模型对象中的数据信息，并呈现出来。
模型层则通过控制层将数据操作结果反馈给视图对象。
控制器对象还负责创建和协同应用中的任务，并对其它对象进行生命周期管理。

应用实例

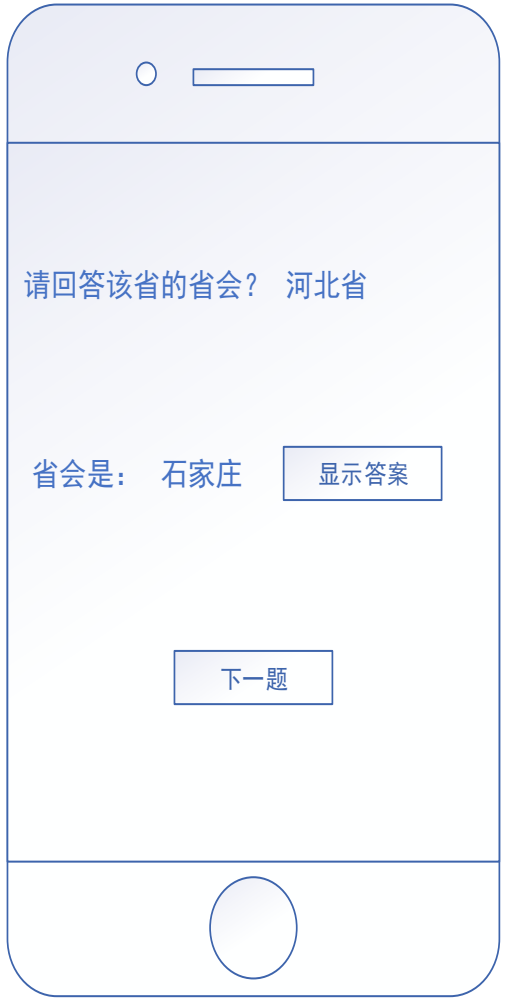
界面设计

在界面上有三行文字:

第一行文字提出问题: “请回答以下省份的省会城市？”，然后显示省份的名称。

第二行的内容是显示“省会城市是:”，然后后面紧跟着省会城市的答案，初始状态显示“？？？”，当点击后面的按钮“显示答案”时，将会显示省会城市的名称。

第三行是一个按钮“下一题”，用来切换到下一个省份的名称。



功能设计

视图层

根据app的设计草图搭建界面：

用四个UILabel视图的实例来分别显示：问题文本、省名文本、省会提示文本以及省会城市名文本

用两个UIButton视图的实例来分别用来实现：显示答案和切换题目。

这六个视图实例都在同一个容器视图UIView的实例View中，它们负责与用户进行交互，共同构成了应用的视图层，即MVC中的Views Layer。

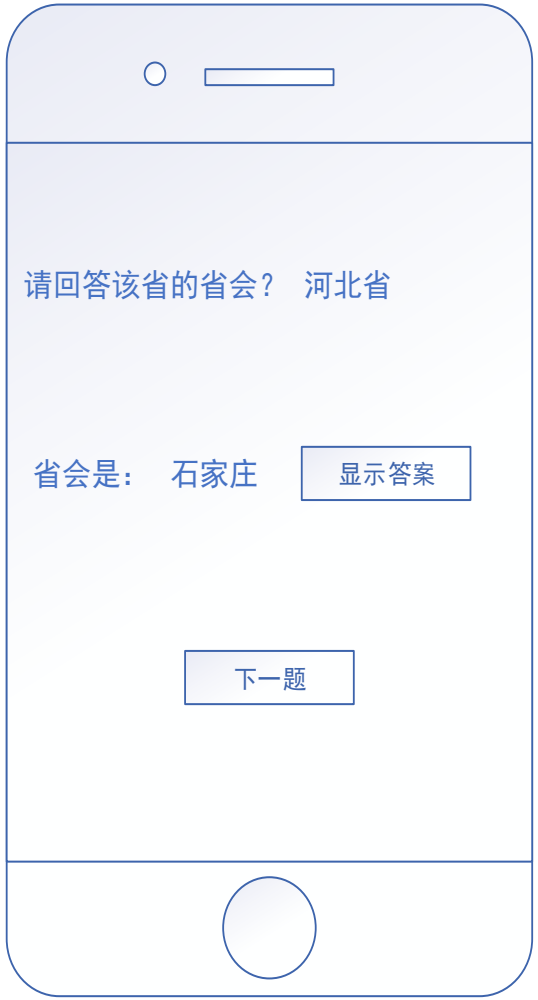
模型层

相关数据是省份名和省会名，可以分别用两个字符串数组来保存。

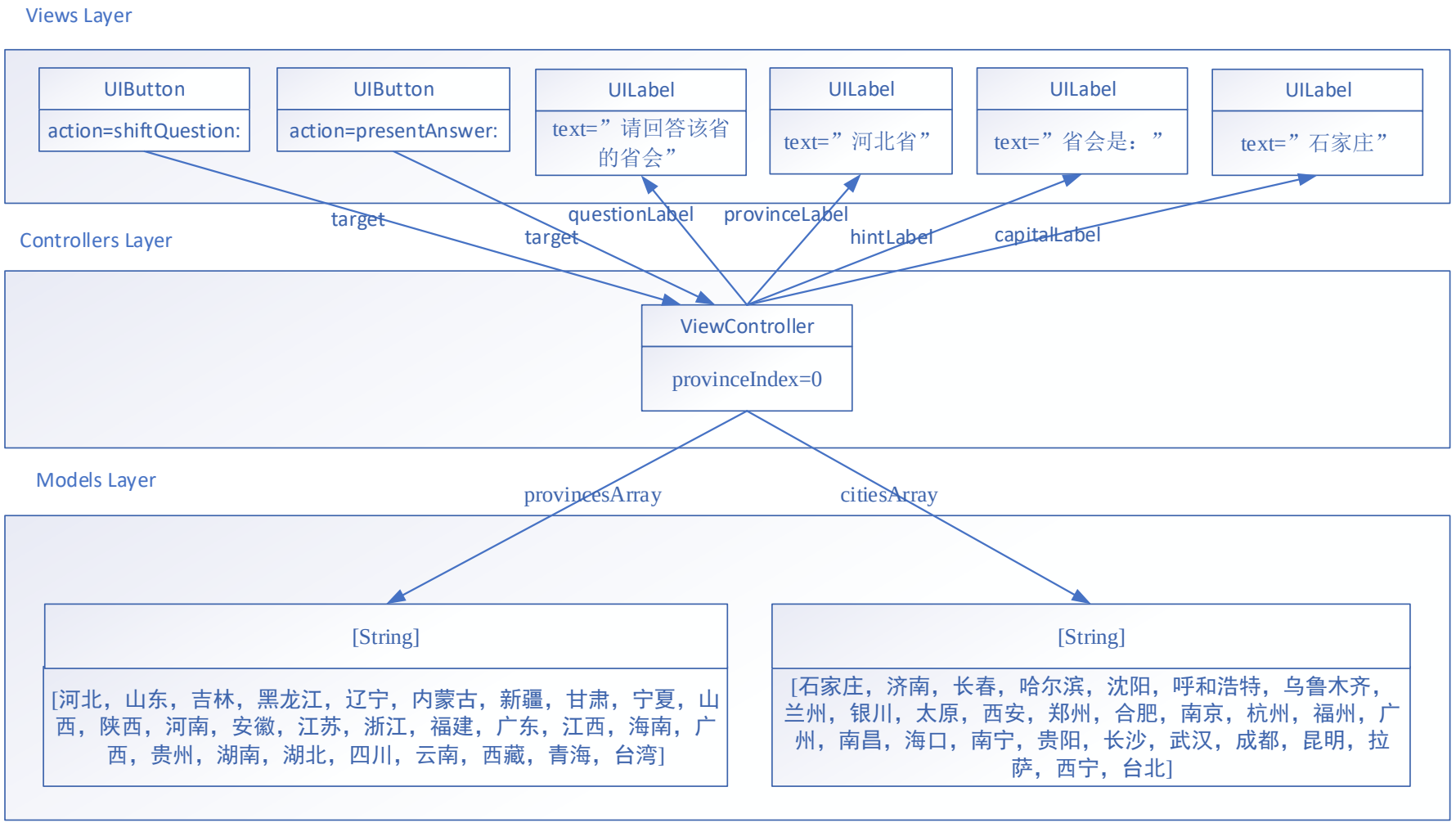
控制层

为了实现用户界面和数据模型之间的通讯，还需要构建应用的控制层，即MVC中的Controllers Layer。

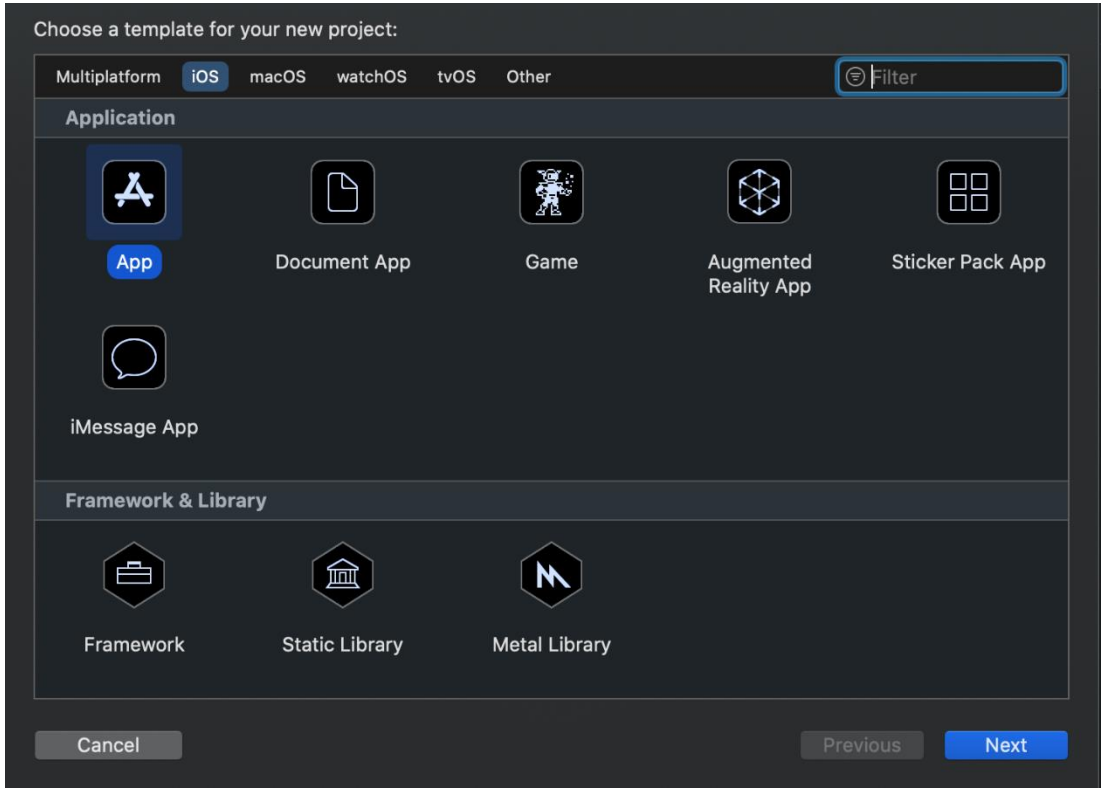
这里视图结构比较简单，只需要一个容器视图View，因此使用一个UIViewController的实例ViewController，所有用户界面与数据模型之间的交互，都由ViewController作为中介来协调处理。



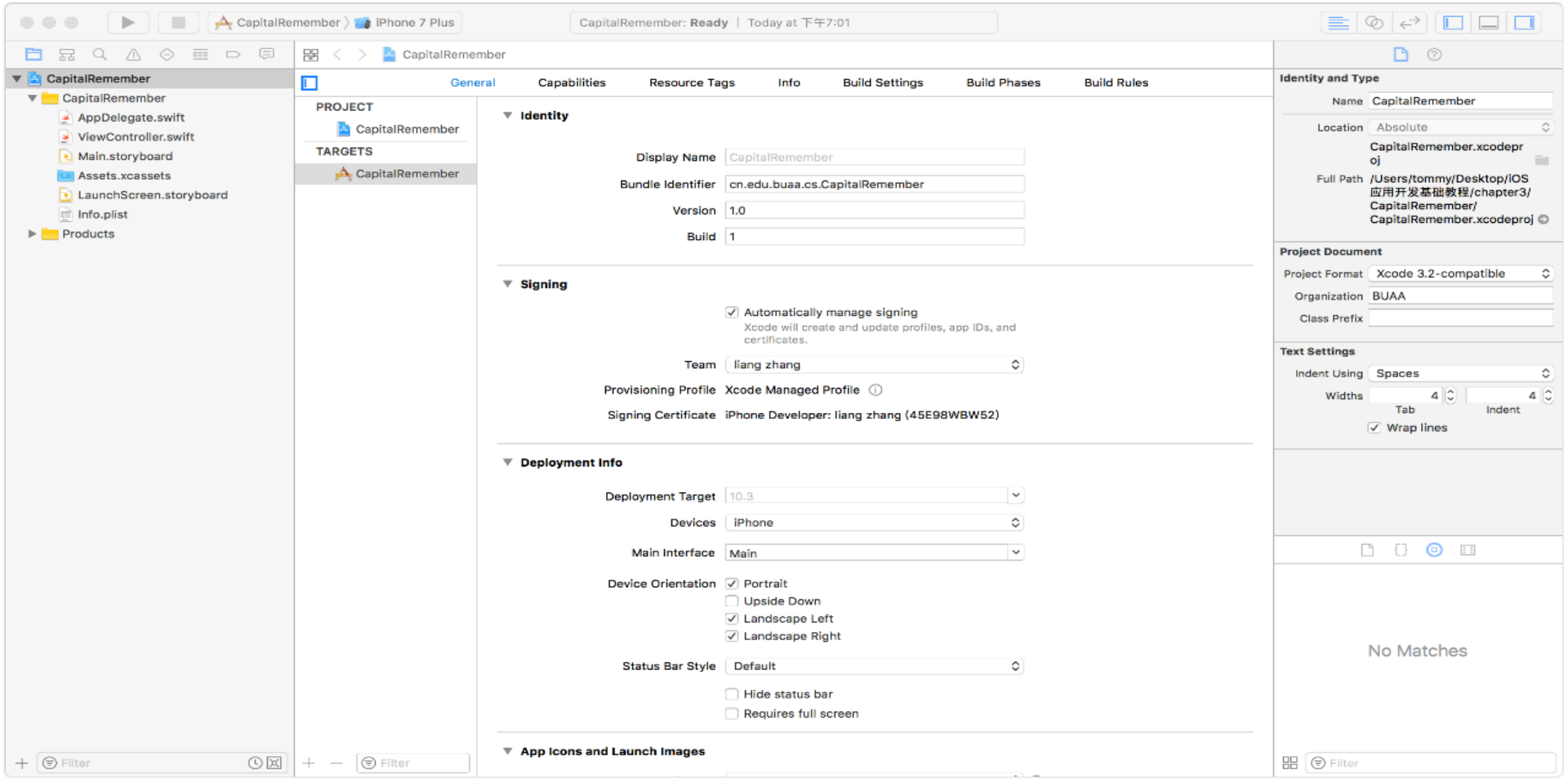
MVC架构图


Views Layer
UIButton
action=shiftQuestion:
UIButton
action=presentAnswer:
UILabel
text=”请回答该省的省会”
UILabel
text=”河北省”
UILabel
text=”省会是：”
UILabel
text=”石家庄”
target
target
questionLabel
provinceLabel
hintLabel
capitalLabel
Controllers Layer
ViewController
provinceIndex=0
Models Layer
provincesArray
citiesArray
[String]
[河北，山东，吉林，黑龙江，辽宁，内蒙古，新疆，甘肃，宁夏，山西，陕西，河南，安徽，江苏，浙江，福建，广东，江西，海南，广西，贵州，湖南，湖北，四川，云南，西藏，青海，台湾]
[String]
[石家庄，济南，长春，哈尔滨，沈阳，呼和浩特，乌鲁木齐，兰州，银川，太原，西安，郑州，合肥，南京，杭州，福州，广州，南昌，海口，南宁，贵阳，长沙，武汉，成都，昆明，拉萨，西宁，台北]

创建应用项目


Choose a template for your new project:
Multiplatform
iOS
macOS
watchOS
tvOS
Other
Filter
Application
App
Document App
Game
Augmented Reality App
Sticker Pack App
iMessage App
Framework & Library
Framework
Static Library
Metal Library
Cancel
Previous
Next

项目文件树

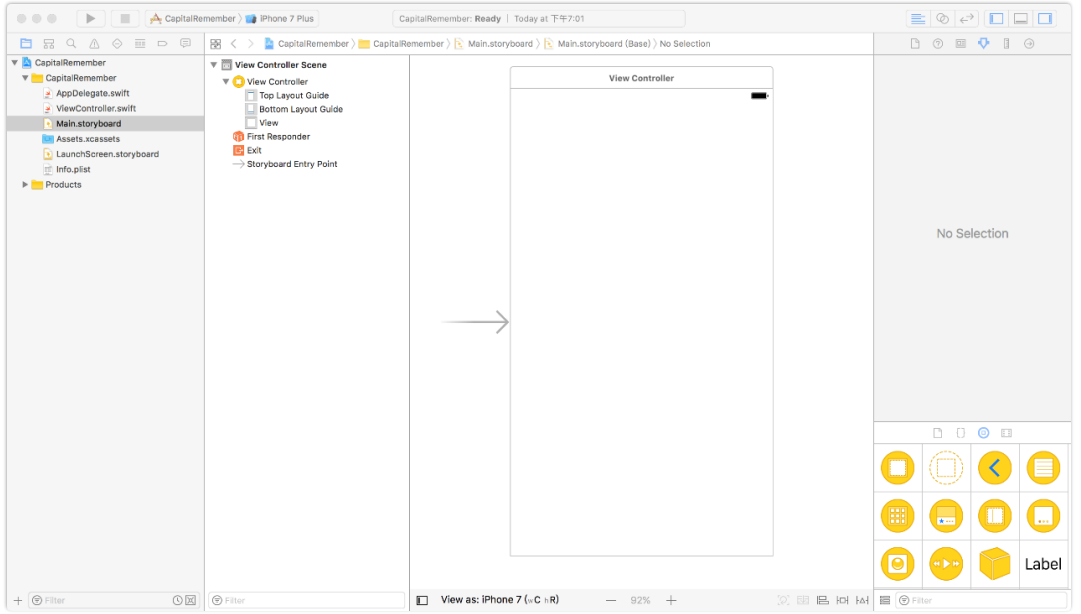

CapitalRemember
iPhone 7 Plus
CapitalRemember: Ready | Today at 下午7:01
CapitalRemember
CapitalRemember
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
CapitalRemember
TARGETS
CapitalRemember
Identity
Display Name CapitalRemember
Bundle Identifier cn.edu.buaa.cs.CapitalRemember
Version 1.0
Build 1
Signing
Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.
Team liang zhang
Provisioning Profile Xcode Managed Profile
Signing Certificate iPhone Developer: liang zhang (45E98WBW52)
Deployment Info
Deployment Target 10.3
Devices iPhone
Main Interface Main
Device Orientation
Portrait
Upside Down
Landscape Left
Landscape Right
Status Bar Style Default
Hide status bar
Requires full screen
App Icons and Launch Images
Identity and Type
Name CapitalRemember
Location Absolute
CapitalRemember.xcodeproj
Full Path /Users/tommy/Desktop/iOS应用开发基础教程/chapter3/CapitalRemember/CapitalRemember.xcodeproj
Project Document
Project Format Xcode 3.2-compatible
Organization BUAA
Class Prefix
Text Settings
Indent Using Spaces
Widths 4 4
Tab Indent
Wrap lines
No Matches
Filter

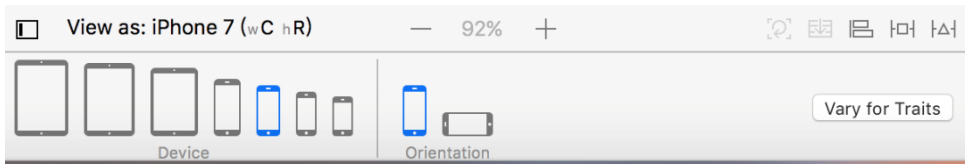
定义视图层

打开故事版文件

向画板中添加View控件

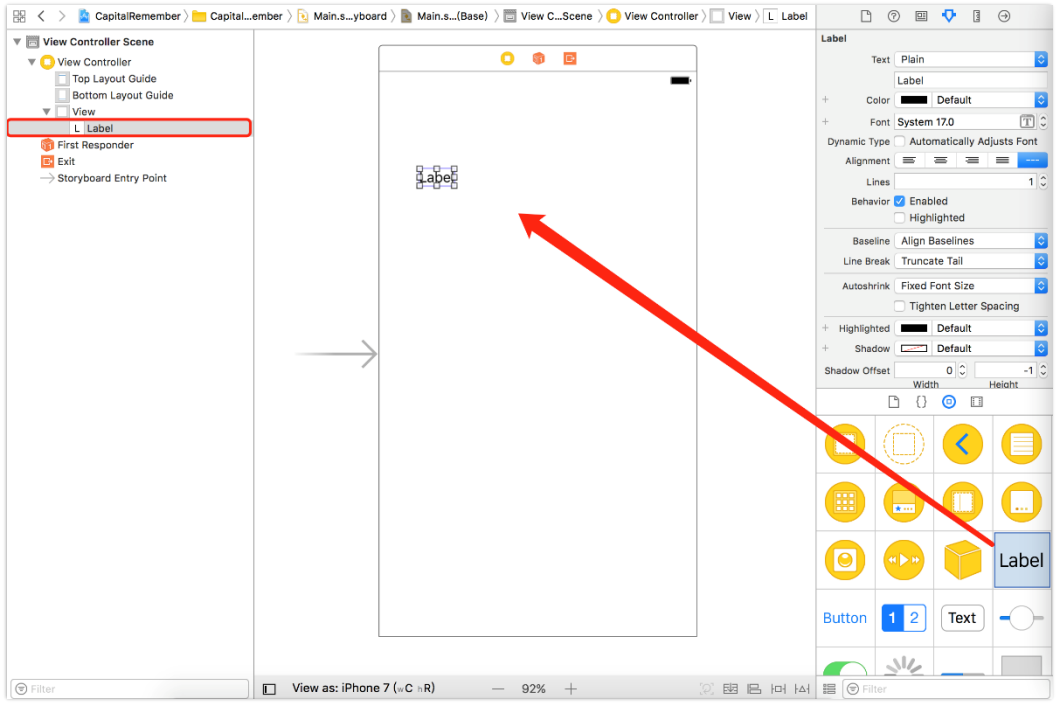
选择硬件设备的型号来适配屏幕尺寸

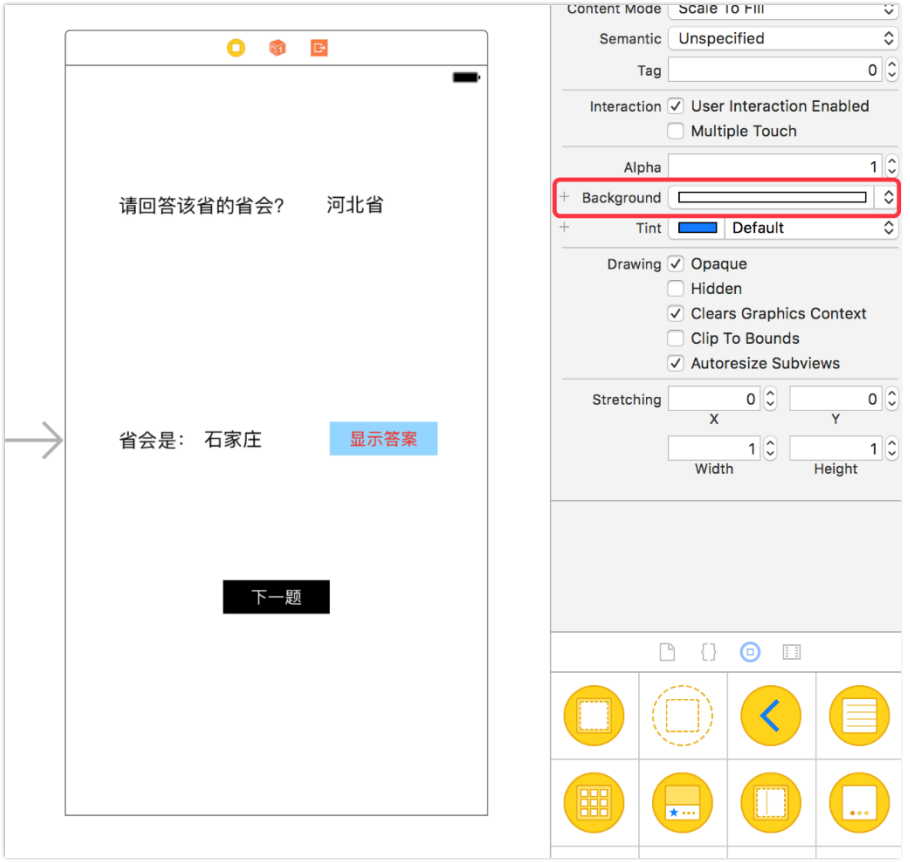




定义视图层

向视图中添加各种控件，设置相应属性





关联视图层与控制层

自动创建视图控制器

本项目采用了“Single View Application”模板，缺省创建了一个View及其ViewController，因此不需要手动创建ViewController。打开故事板文件可以查看到这两个文件，并且已经建立了关联关系。

手动关联新增控件与视图控制器

现只需要连接视图层中新增加的视图控件到ViewController类中，ViewController就可以管理视图中的新增控件了。

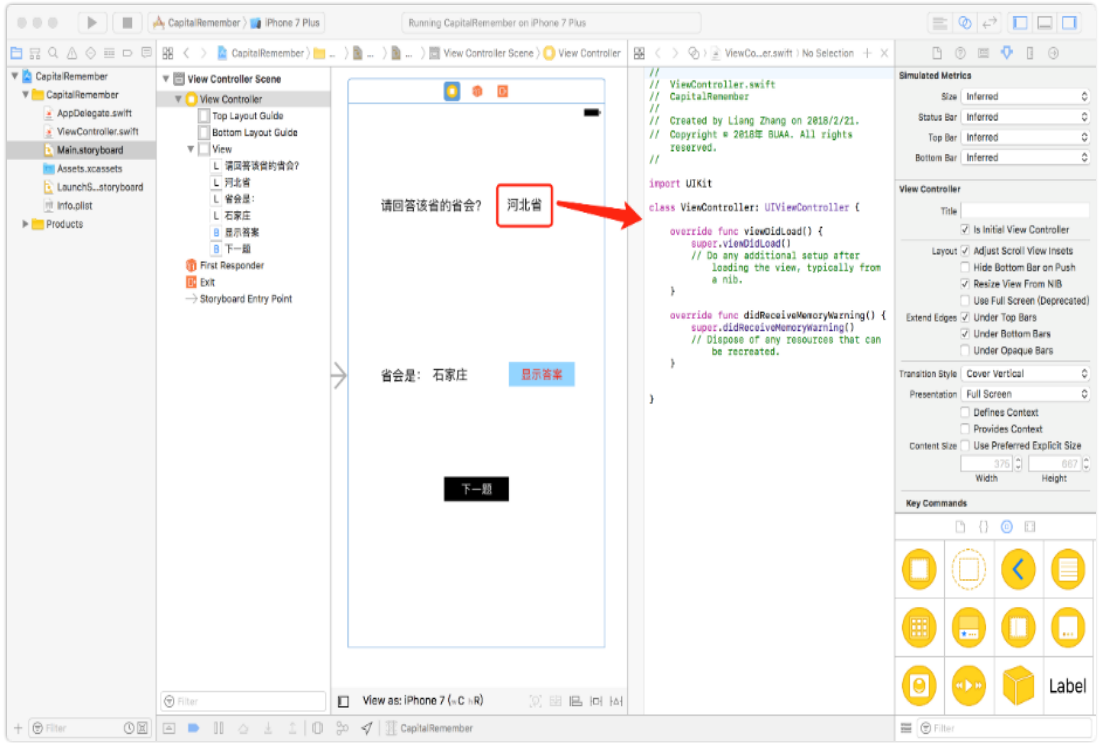
有多种方法可以实现视图层控件与控制层的连接，这里介绍其中一种。

关联视图层与控制层：实例

连接Label与ViewController

具体操作

选中Label图形后，按住Ctrl键不放，同时按住鼠标左键由Label出发，拉出一根线到右侧的ViewController类的定义部分的空白处，然后松开鼠标左键，会弹出一个窗口。



关联视图层与控制层：实例

设置连接关系的属性

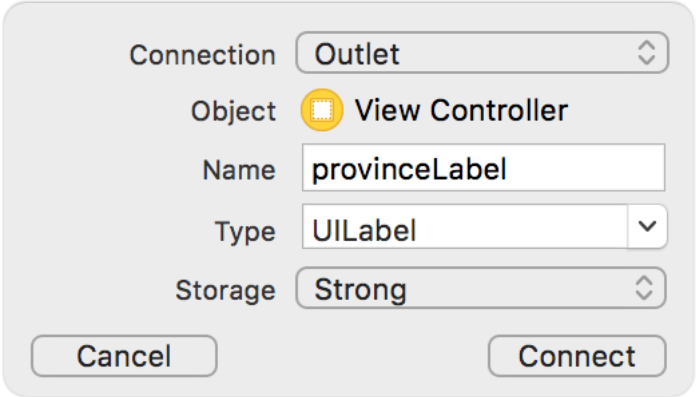
具体操作

在连接属性设置对话框中定义连接的属性：

Connection为出口Outlet类型，一般用于变量；

Name为provinceLabel，这是显示省份名称的Label在ViewController类里对应的变量名。

设置完以后点击Connect，建立连接关系。



关联视图层与控制层：实例

完成连接后的结果

在该类的声明部分增加了一行provinceLabel的变量定义。

注意：

在变量定义的左侧有一个圆圈，圆圈表示该变量为一个可以建立连接关系的变量。

如果没有与某个视图控件建立连接关系，则为空心圆圈，反之为实心圆圈。

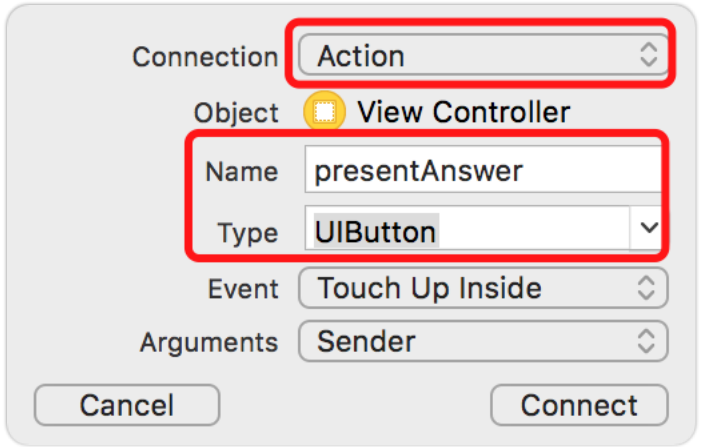
关联视图层与控制层：实例

连接Button与ViewController

建立连接关系时，Button和Label不同之处：

Label需要在运行过程中根据用户操作来改变text属性值，因此需要建立Outlet连接，在ViewController中增加一个变量定义；

Button需要接受到用户的点击动作，并做相应的事件处理。因此，在连接属性设置窗口中，Connection设置为Action，表示该连接为一个动作处理函数；Name为presentAnswer，这是动作处理函数名；Type为UIButton，表示发送方的类型。



关联视图层与控制层：实例

全部连接完成后的ViewController

在声明部分多了两个变量provinceLabel和cityLabel的定义

增加了两个函数的定义：

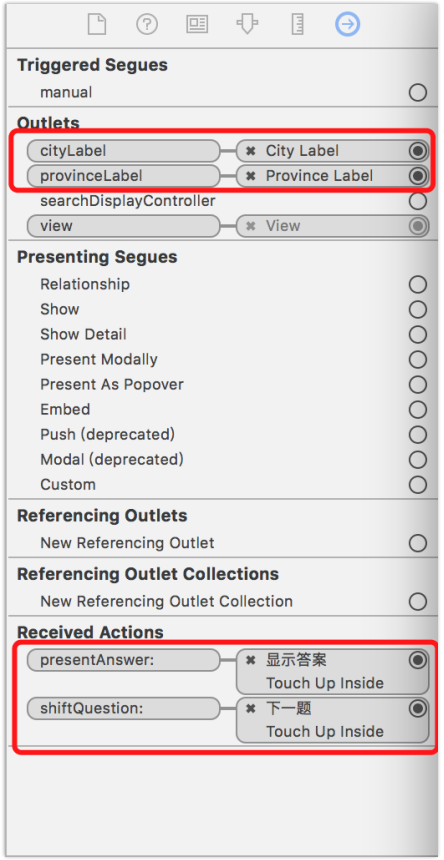
presentAnswer(_ sender: UIButton)

shiftQuestion(_ sender: UIButton)

函数体均为空（这里还没有编写事件的处理逻辑）

检查连接关系的正确性

```swift
//
//  ViewController.swift
//  CapitalRemember
//
//  Created by Liang Zhang on 2018/2/21.
//  Copyright © 2018年 BUAA. All rights
    reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet var provinceLabel: UILabel!

    @IBOutlet var cityLabel: UILabel!


    @IBAction func presentAnswer(_ sender:
        UIButton) {
    }

    @IBAction func shiftQuestion(_ sender:
        UIButton) {
    }


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after
            loading the view, typically from
            a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can
            be recreated.
    }


}
```



定义模型层

数据模型层由两个字符串数组构成。

这两个字符串数组用来储存省份信息和省会信息。

具体操作

可以直接在ViewController类的定义中，

增加字符串数组provincesArray和citiesArray的定义；

增加一个provinceIndex整型变量，用来记录当前显示的省份在数组provincesArray中的序号，以便对应到正确的省会城市。

```
class ViewController: UIViewController {

    @IBOutlet var provinceLabel: UILabel!

    @IBOutlet var cityLabel: UILabel!

    let provincesArray: [String] = ["河北","山东","吉林","黑龙江","辽宁","内蒙古","新疆",
        "甘肃","宁夏","山西","陕西","河南","安徽","江苏","浙江","福建","广东","江西",
        "海南","广西","贵州","湖南","湖北","四川","云南","西藏","青海","台湾"]

    let citiesArray: [String] = ["石家庄","济南","长春","哈尔滨","沈阳","呼和浩特","乌鲁木齐",
        "兰州","银川","太原","西安","郑州","合肥","南京","杭州","福州","广州","南昌","海口",
        "南宁","贵阳","长沙","武汉","成都","昆明","拉萨","西宁","台北"]

    var provinceIndex: Int = 0

    @IBAction func presentAnswer(_ sender: UIButton) {
    }

    @IBAction func shiftQuestion(_ sender: UIButton) {
    }


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

定义控制层

在控制层中添加控制逻辑

点击按钮“下一题”的动作处理函数

点击按钮“显示答案”的动作处理函数

初始化provinceLabel的text属性

```
@IBAction func shiftQuestion(_ sender: UIButton) {

    provinceIndex += 1

    if provinceIndex == provincesArray.count {
        provinceIndex = 0
    }

    let province: String = provincesArray[provinceIndex]
    provinceLabel.text = province

    cityLabel.text = "???"

}
```

```
@IBAction func presentAnswer(_ sender: UIButton) {

    let city: String = citiesArray[provinceIndex]
    cityLabel.text = city

}
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.

    provinceLabel.text = provincesArray[provinceIndex]

}
```

运行结果


iPhone 7 Plus – iOS 10.3 (14E8301)
Carrier
8:05 PM
请回答该省的省会?
山东
省会是：
???
显示答案
下一题
iPhone 7 Plus – iOS 10.3 (14E8301)
Carrier
8:05 PM
请回答该省的省会?
山东
省会是：
济南
显示答案
下一题

特别说明

示例的开发环境是Xcode10，最新版本是Xcode12.

在添加可视化组件、组件与代码的关联的添加方法略有不同，建议同学们自行探索一下！