iOS 9 音频应用开发基础教程

（内部资料 v1.0）

大学霸
www.daxueba.net

前　言

在 iOS 应用中，音频可以用来充当很多的角色。例如，在游戏应用中，音频可以用来充当背景音乐、或者是特殊的音效；在 QQ 音乐中，音频就是纯粹用来播放的；在短信应用中，音频可以用来提醒用户。在一个应用中恰当的使用音频可以提供应用程序的用户体验。

现在绝大多数的 iOS 应用都会与音频有关。到目前为止，市面上对音频的介绍很少，更别提使用 Swift 2 语言去开发音频应用了。

基于以上不可忽略的事实，本书决定着眼于讲解 Swift 2 语言去开发 iOS 中的音频应用程序，并且详细的讲解了在音频应用中出现的各种功能，相信这样会更好的帮助读者去学习本书。。

1.学习所需的系统和软件

- ❑ 安装 Mac OS 10.11 操作系统
- ❑ 安装 Xcode 7.0

2.学习建议

大家学习之前，可以致信到 xxxxxxxx，获取相关的资料和软件。如果大家在学习过程遇到问题，也可以将问题发送到该邮箱。我们尽可能给大家解决。

目　录

第 2 章　播放音频

QQ 音乐、虾米音乐、百度音乐这样应用都是用来实现音频播放的。在 iOS 中实现音频播放最为简单和常用的就是 AVFoundation 框架中的 AVAudioPlayer 类。虽然 AVAudioPlayer 类不能播放网络上的音频文件，但是它可以播放本地音频文件，以及缓冲区的文件。本章将讲解最为基础的音频播放——本地音频文件的播放。

2.1　基本功能

实现音频的播放需要使用到 AVAudioPlayer 类。AVAudioPlayer 是 AVFoundation.framework 框架里面最基本的一个音频播放器的类。使用此类可以实现单个音频的播放、暂停以及停止等功能。本节将使用 AVAudioPlayer 类实现一个音频文件的播放。

2.1.1　AVAudioPlayer 类简介

AVAudioPlayer 是 iOS 2.2 之后引入的 AVFoundation.framework 框架中的一个类，使用此类可以实现音频的播放。表 2-1 中总结了 AVAudioPlayer 类中使用到的方法。

表 2-1　AVAudioPlayer类的方法

方法名	功能
init(contentsOfURL:)	使用本地文件的地址初始化播放器
init(data:)	使用NSData初始化播放器，注意使用此方法时必须文件格式和文件后缀一致，否则出错。
init(contentsOfURL:fileTypeHint:)	使用本地文件的地址初始化播放器。该文件可以没有后缀名
init(data:fileTypeHint:)	使用NSData初始化播放器。该文件可以没有后缀名
play()	播放音频文件
playAtTime()	在指定的时间开始播放音频
pause()	暂停播放
stop()	停止播放
prepareToPlay()	加载音频文件到缓冲区，注意即使在播放之前音频文件没有加载到缓冲区程序也会隐式调用此方法。
averagePowerForChannel(_:)	获得指定声道的分贝平均值
peakPowerForChannel(_:)	获得指定声道的分贝峰值
updateMeters()	更新音频测量值

表 2-2 总结了了 AVAudioPlayer 类中使用到的属性。

表 2-2　AVAudioPlayer类的属性

属性	功能
playing	音频是否正在播放，只读
volume	音量大小
pan	立体声平衡
rate	播放速率
enableRate	是否允许改变播放速度

numberOfLoops	循环播放次数。如果为0，则不循环；如果小于0，则无限循环；如果大于0，则表示循环次数
delegate	委托设置
settings	音频播放设置信息，只读
numberOfChannels	音频声道数，只读
duration	音频时长
currentTime	当前播放时长
deviceCurrentTime	输出设备播放音频的时间，注意如果播放中被暂停此时间也会继续累加
url	音频文件路径，只读
data	音频数据，只读
meteringEnabled	是否启用音频测量

注意：在 Xcode 6.0 之后，在创建的项目中，AVFoundation.framework 框架默认是被添加到项目中的。如果 AVFoundation.framework 框架没有被添加到项目中，就需要开发者手动进行添加。手动添加框架的具体操作步骤如下：

（1）选择导航窗口中的项目名称，打开目标窗口，如图 2.1 所示。

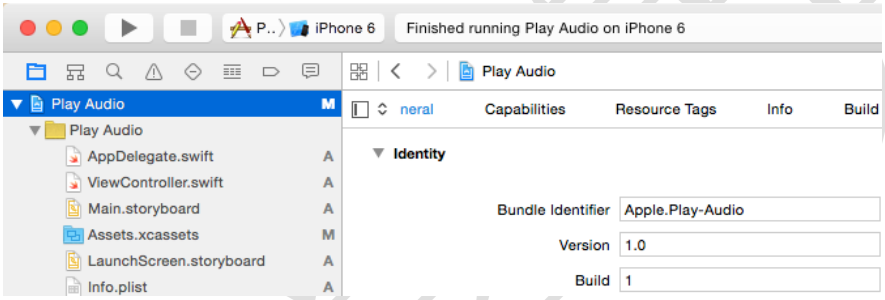

图 2.1　目标窗口

（2）选择目标窗口中的 Build Phases 选项，打开 Build Phases 面板，如图 2.2 所示。

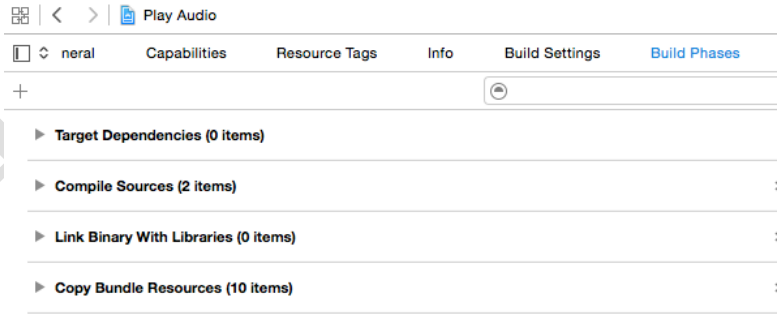

图 2.2　Build Phases 面板

（3）将 Link Binary With Libraries(0 items)打开，会看到一个加号按钮，如图 2.3 所示。

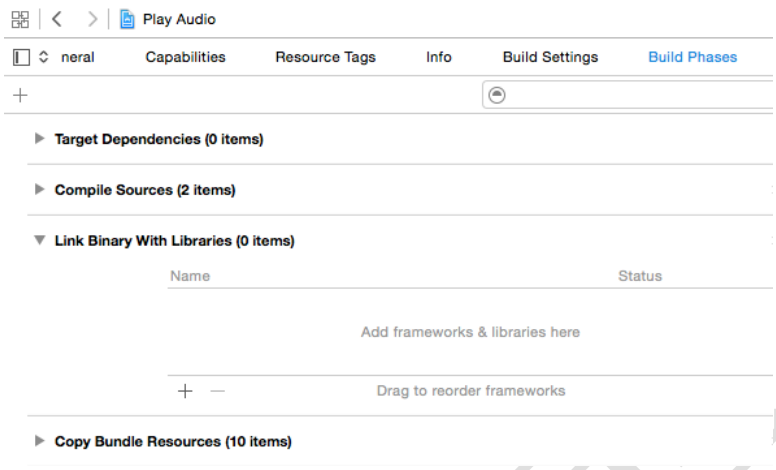

图 2.3　Link Binary With Libraries(0 items)

（4）选择加号按钮后，会弹出一个 Choose frameworks and libraries to add 对话框，如图 2.4 所示。

（5）在对话框中找到 AVFoundation.framework 框架，单击 Add 按钮。此时，就会在 Link Binary With Libraries 中出现添加的框架，如图 2.5 所示。这表明此框架已经添加到了项目中。

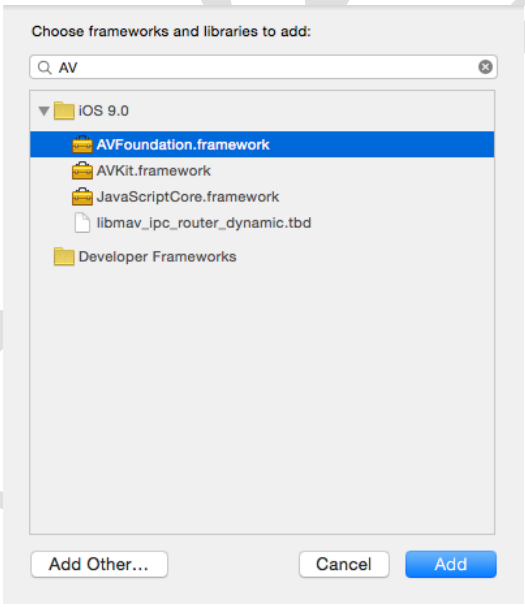

图 2.4　Choose frameworks and libraries to add 对话框

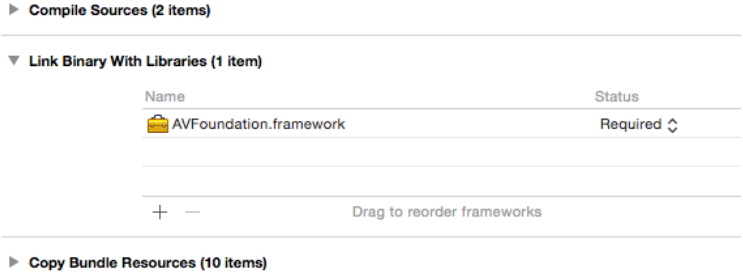

图 2.5　添加框架

2.1.2　准备素材文件

在 iOS 的游戏应用中（如例如超级玛丽）或者是酷狗音乐打开时，经常会有音频的播放。这些音频都是应用程序预置的，也就是素材文件。在实现播放本地音频文件之前，首先需要将预置的音频文件添加到创建的项目中。添加音频文件的具体操作步骤如下。

（1）在创建项目的导航窗口中右击，弹出快捷菜单。

（2）在弹出的快捷菜单中选择 Add Files to "***"…命令。弹出选择文件对话框，如图 2.6 所示。

（3）选择音频文件后，单击 Add 按钮，此时被选择的音频文件就添加到了创建的项目中。

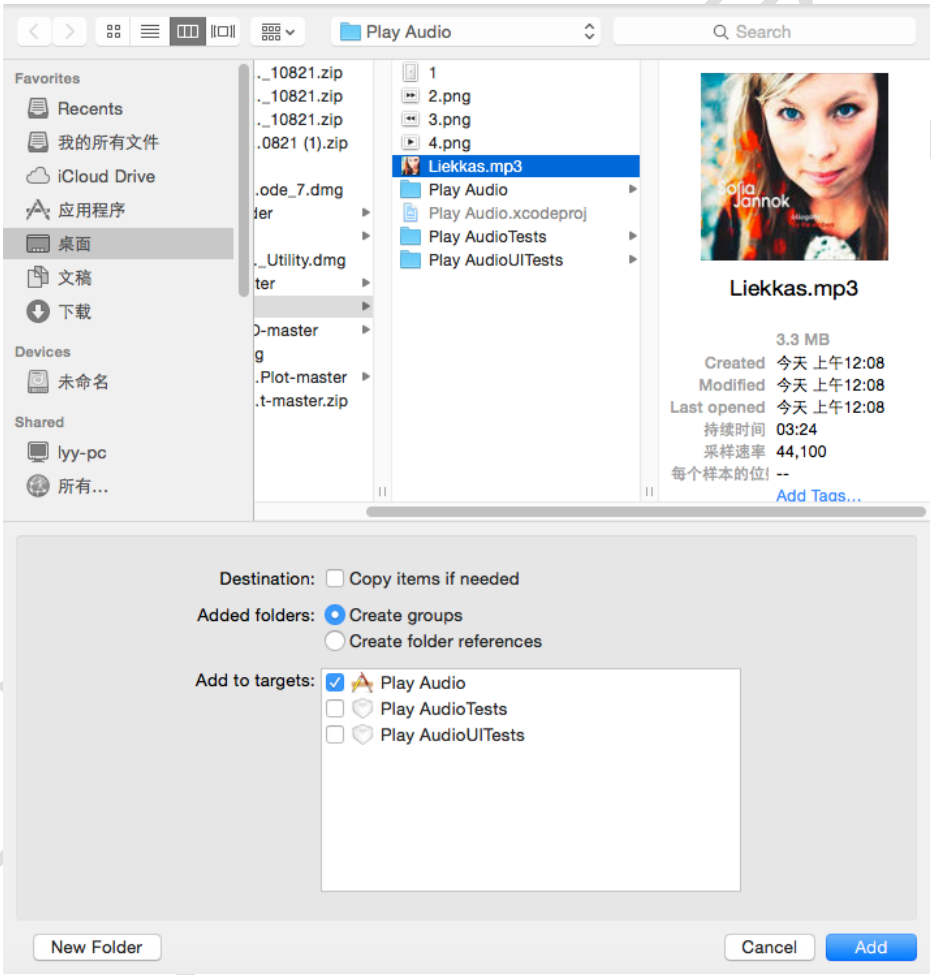

图 2.6　添加音频文件

注意：添加音频文件到创建的项目中除了以上讲解的方式外，还有一种方式，具体的操作步骤如下。

（1）找到需要添加到项目中的音频文件，将其拖动到创建的项目中，弹出 Choose options for adding these files:对话框架，如图 2.7 所示。

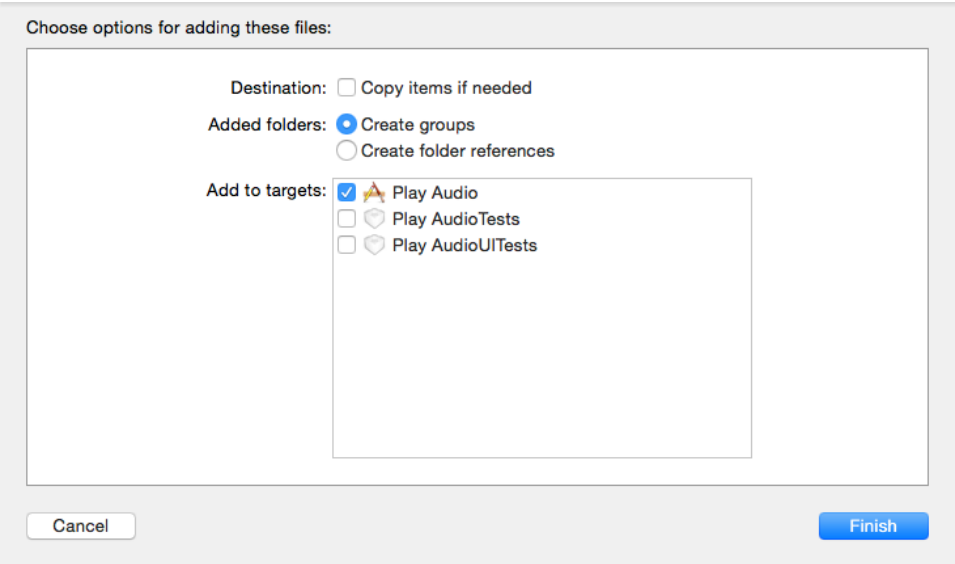

图 2.7　添加音频文件

（2）单击 Finish 按钮后，音频文件就添加到了创建的项目中。

注意：在添加音频文件中提到的两种方式，除了可以添加音频文件外，还可以添加其他文件，如图像文件、字体文件等。

2.1.3　加载音频文件

加载音频文件是为了对 AVAudioPlayer 类进行实例化，实例化时需要使用到 init()方法，根据 init()方法参数需求的不同，可以分为 4 种形式，分别为 init(contentsOfURL:)、init(data:)、init(contentsOfURL:fileTypeHint:)和 init(data:fileTypeHint:)。以下就是对这四种形式的讲解。

1.init(contentsOfURL:)

init(contentsOfURL:)是使用本地音频文件的地址对 AVAudioPlayer 类进行实例化。其语法形式如下：

```
init(contentsOfURL url: NSURL) throws
```

其中，参数 url 是 NSURL 对象。NSURL 是一个数据类型，实际上就是一个地址，很多开发人员会有疑惑，地址就是一个字符串，为什么还有 NSURL 类型，那是因为地址分为绝地路径（描述目标文件夹的位置）和相对路径（显示文件的完整路径）。对于绝对路径来说，地址的字符串都会比较复杂，包括很多请求参数。这样，在请求过程中需要解析出来每个部分，所以封装一个 NSURL。所以需要将字符串转换为 NSURL，对于 NSURL 的转换其实可以使用它的实例化方法 init 来实现，这个方法中会有一个字符串参数，也就是使用字符串去实例化 NSURL 方法，其语法形式如下：

```
init(fileURLWithPath path: String)
```

其中，参数 path 是一个字符串。此字符串是一个地址（一般来说不同部分之间以斜线（/）分隔），这个地址可以是相对路径，也可以是绝对路径。

注意：参数 path 是只可以是本地文件的地址，不可以是网络地址。

2.init(data:)

在前面的内容中提到 AVAudioPlayer 类不可以播放从网络上获取音频文件，但是可以播放从网络上下载到内存缓冲区的音频。要实现内存缓冲区音频的播放，需要使用 init()方法的 init(data:)方式，其语法形式如下：

```
init(data data: NSData) throws
```

其中，参数 data 是一个 NSData 对象，表示一个音频数据。由于是要播放内存缓冲区的音频，所以 NSData 对象在创建时需要指定音频文件的地址，此时需要使用到 init()方法的 init(contentsOfFile:)形式。

```
init?(contentsOfFile path: String)
```

其中，参数 path 是一个字符串，代码表示文件的绝对路径。

注意：参数 data 必须是一个完整的文件。

3.init(contentsOfURL:fileTypeHint:)

方法 init(contentsOfURL:)和 init(data:)都是在具有后缀名的情况下使用的形式，但是在网络上下载的音频文件可能由于某些原因没有后缀名。这时就需要使用到 init()方法的 init(contentsOfURL:fileTypeHint:)形式以及 init(data:fileTypeHint:)。init(contentsOfURL:fileTypeHint:)的语法形式如下：

```
init(contentsOfURL url: NSURL,
    fileTypeHint utiString: String?) throws
```

其中，参数 url 是 NSURL 对象，此对象的实例化在前面讲解过。参数 utiString 用来指定文件类型的提示，用于提示系统该文件属于什么类型。其中，文件类型的提示如表 2-3 所示。

表 2-3　文件类型的提示

类型类型的提示	功能
AVFileType3GPP	用于3GPP文件格式，通常文件采用.3gp、.3gpp和.sdv等进行定义
AVFileType3GPP2	用于3GPP2文件格式，通常文件采样.3g2、.3gp2等进行定义
AVFileTypeAIFC	用于AIFC音频文件格式，通常文件采样.aifc、.cdda等进行定义
AVFileTypeAIFF	用于AIFF音频文件格式，通常文件采样.aif、.aiff等进行定义
AVFileTypeCoreAudioFormat	用于CoreAudio文件格式，通常文件采样.caf等进行定义
AVFileTypeAppleM4V	用于iTunes视频文件格式，通常文件采样.m4v等进行定义
AVFileTypeMPEG4	用于MPEG-4音频文件格式，通常文件采样.mp4等进行定义
AVFileTypeAppleM4A	用于m4a音频文件格式，通常文件采样.m4a等进行定义
AVFileTypeQuickTimeMovie	用于QuickTime电影文件格式，通常文件采样.mov、.qt等进行定义
AVFileTypeWAVE	用于WAVE音频文件格式，通常文件采样.wav、.wave和.bwf等进行定义
AVFileTypeAMR	用于multi-rate音频文件格式，通常文件采样amr等进行定义
AVFileTypeAC3	用于AC-3音频文件格式，通常文件采样.ac3等进行定义
AVFileTypeMPEGLayer3	用于MPEG layer 3音频文件格式，通常文件采样mp3等进行定义
AVFileTypeSunAU	用于Sun/NeXT音频文件格式，通常文件采样.au、.snd等进行定义

4.init(data:fileTypeHint:)

init(data:fileTypeHint:)形式的语法形式如下：

```
init(data data: NSData,
    fileTypeHint utiString: String?) throws
```

其中，参数 url 是 NSURL 对象，此对象的实例化在前面讲解过。参数 utiString 用来指定文件类型的提示，这些提示可以参考表 2-3。

2.1.4　添加音频文件到缓冲区

对于音频文件来说，读取的次数会过于频繁，这样不仅可能导致存储卡损耗，还会增加 CPU 的负担。为了解决这一问题，我们可以将加载的音频文件存入到缓冲区中（缓冲区为暂时放置输出或输入数据的地方）。而这一过程需要使用到 AVAudioPlayer 的 prepareToPlay()方法，其语法形式如下

```
func prepareToPlay() -> Bool
```

其中，该方法的返回值类型为 Bool，即布尔类型。当值为真时，表示将文件放入缓存；反之，则不

放入缓存。

2.1.5 播放音频

最后一步就是播放音频，此功能的实现需要使用到 AVAudioPlayer 的 play()方法，其语法形式如下：

```
func play() -> Bool
```

其中，该方法的返回值类型为 Bool 即布尔类型。当值为真时，表示音频文件播放成功；反之，则失败。

2.2 第一个实例

为了让开发者可以对上面的内容有更加深入的了解，本节将实现播放音频的第一个实例。在此实例中会涉及到项目的创建、界面设计、关联以及功能代码等内容。

2.2.1 创建项目

在 iOS 开发中所有的应用程序都会存在一个项目，在项目中存放了 iOS 开发所需的各种文件。在 Xcode 7.0 中创建项目会与之前的有所不同，具体的操作步骤如下：

（1）打开 Xcode，弹出 Welcome to Xcode 对话框，如图 2.8 所示。

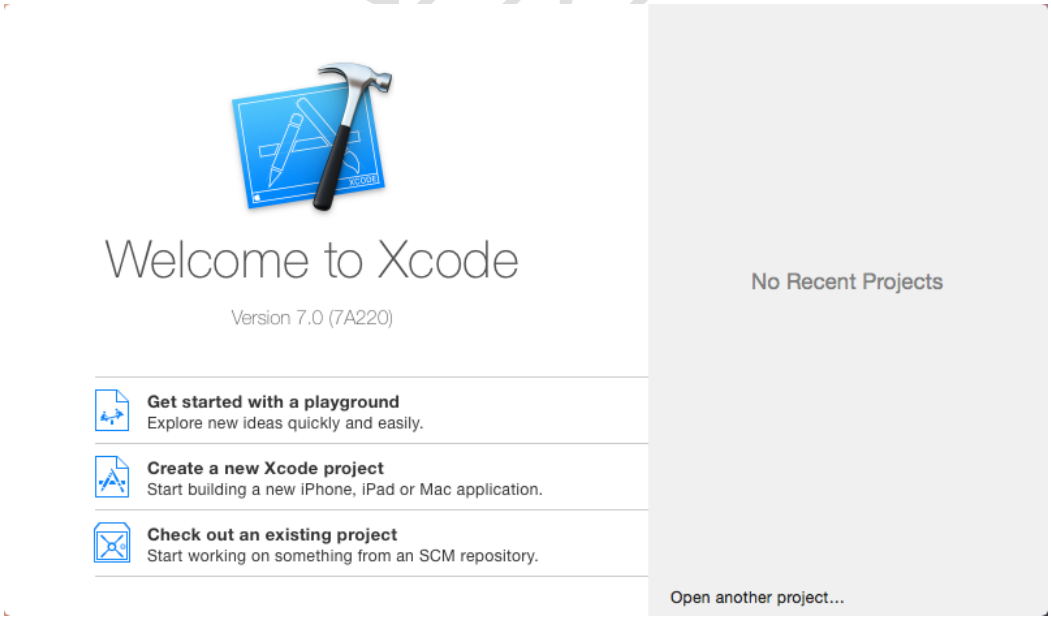

图 2.8 Welcome to Xcode 对话框

（2）选择 Create a new Xcode project 选项，弹出 Choose a template for your new project:对话框，如图 2.9 所示。

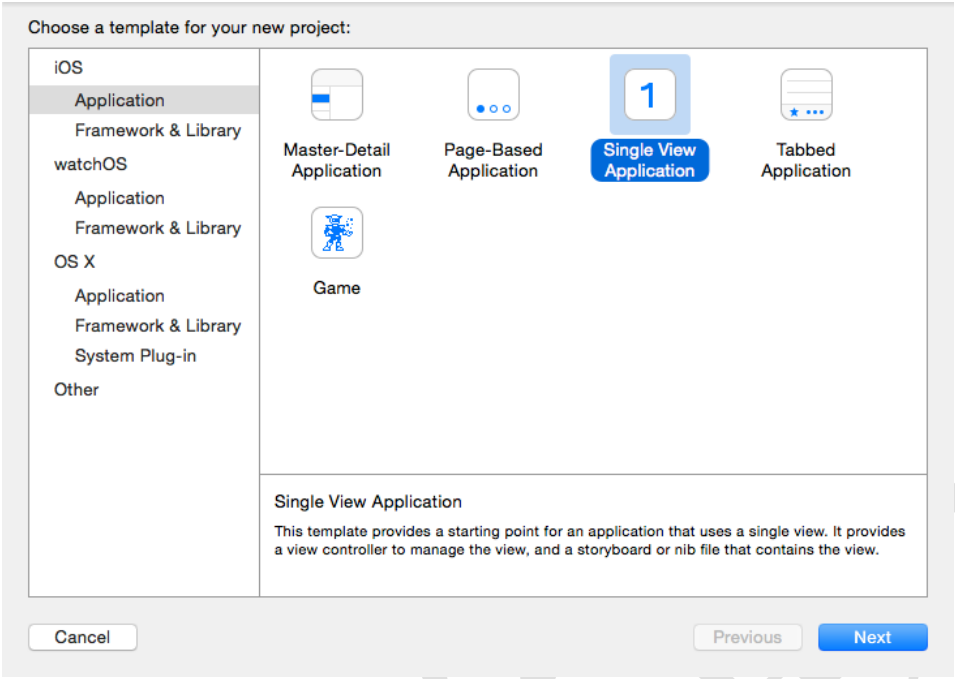

图 2.9　Choose a template for your new project:对话框

（3）选择 iOS|Application 中的 Single View Application 模板，单击 Next 按钮后，弹出 Choose options for your new project:对话框，如图 2.10 所示。

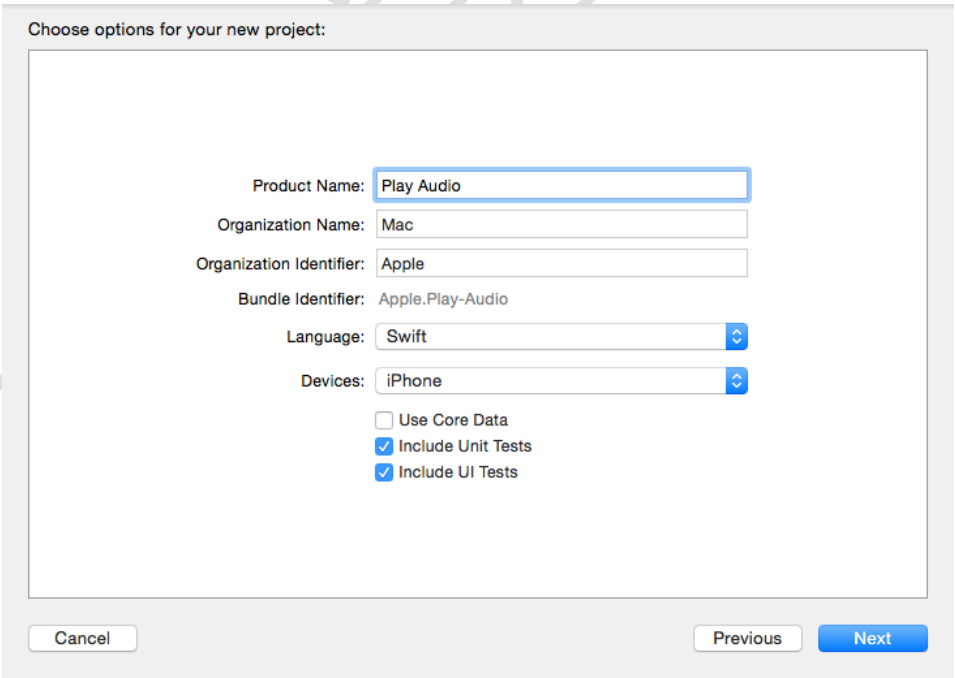

图 2.10　Choose options for your new project:对话框

注意：在图 2.10 中出现的 UI Tests 是 Xcode 7.0 新增的内容。UI Tests 是一个自动测试 UI 与交互的 Testing 组件。它可以通过编写代码、或者是记录开发者的操作过程并代码化，来实现自动点击某个按钮、视图，或者自动输入文字等功能。

（4）填入 Product Name（项目名）、Organization Identifier（标识符）信息以及选择 Language（编

程语言）和设备 Devices（设备），如表 2-4 所示。

表 2-4　填写的内容

需要填写的项	填入的内容
Product Name:	Play Audio
Organization Identifier	Apple
Language	Swift
Devices	iPhone

（5）内容填写完毕后，单击 Next 按钮，打开项目的保存位置对话框，如图 2.11 所示。

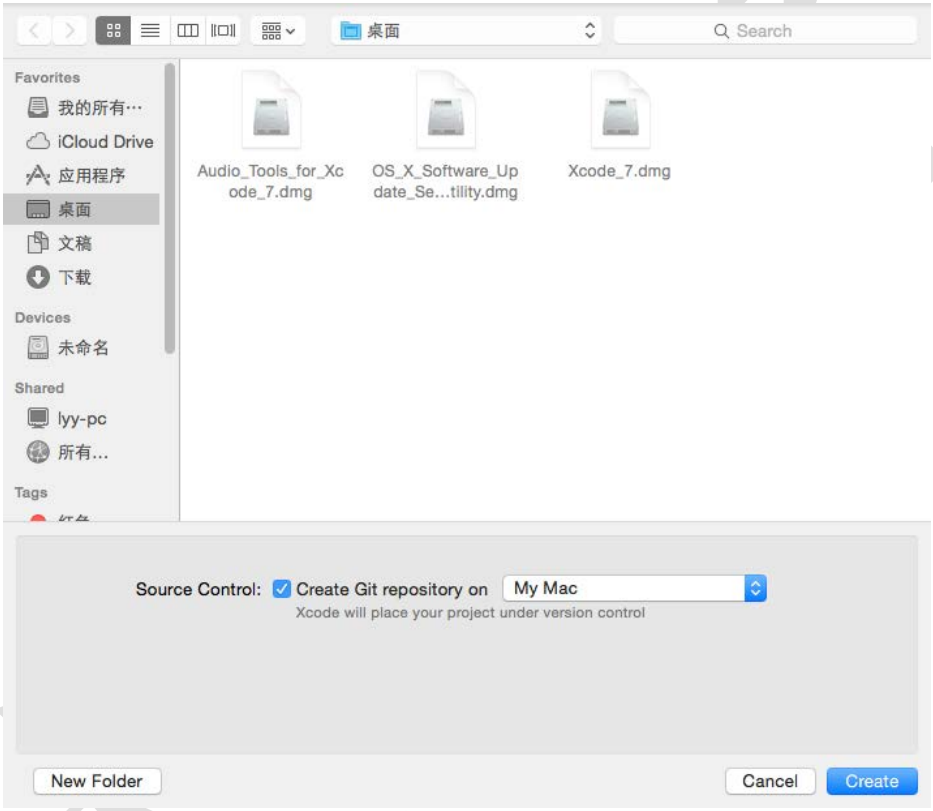

图 2.11　项目的保存位置对话框

（6）单击 Create 按钮，这时一个项目名为 Play Audio 的项目就创建好了。

2.2.2　界面设计

为了让用户可以对此应用中的音频进行控件，也为了讲解的需要，下面对应用程序的界面进行设计。具体的操作步骤如下。

（1）添加图像和 backgroundImage.png、forwardImage.png、backwardImage.png、playImage.png、moreSettingImage.png 和 pauseImage.png 到创建的项目中，如图 2.12 所示。

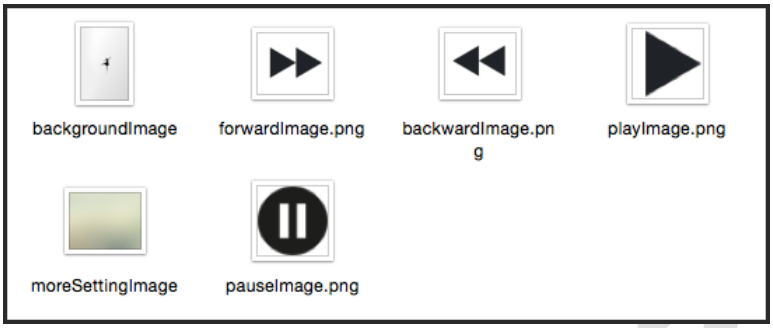

图 2.12　添加的图像

注意：添加图像文件的步骤开发者可以参考 2.1.2 小节中的操作步骤。

（2）添加音频文件 Liekkas.mp3 到创建的项目中。

（3）打开 Main.storyboard 文件，对主视图进行设计，如图 2.13 所示。

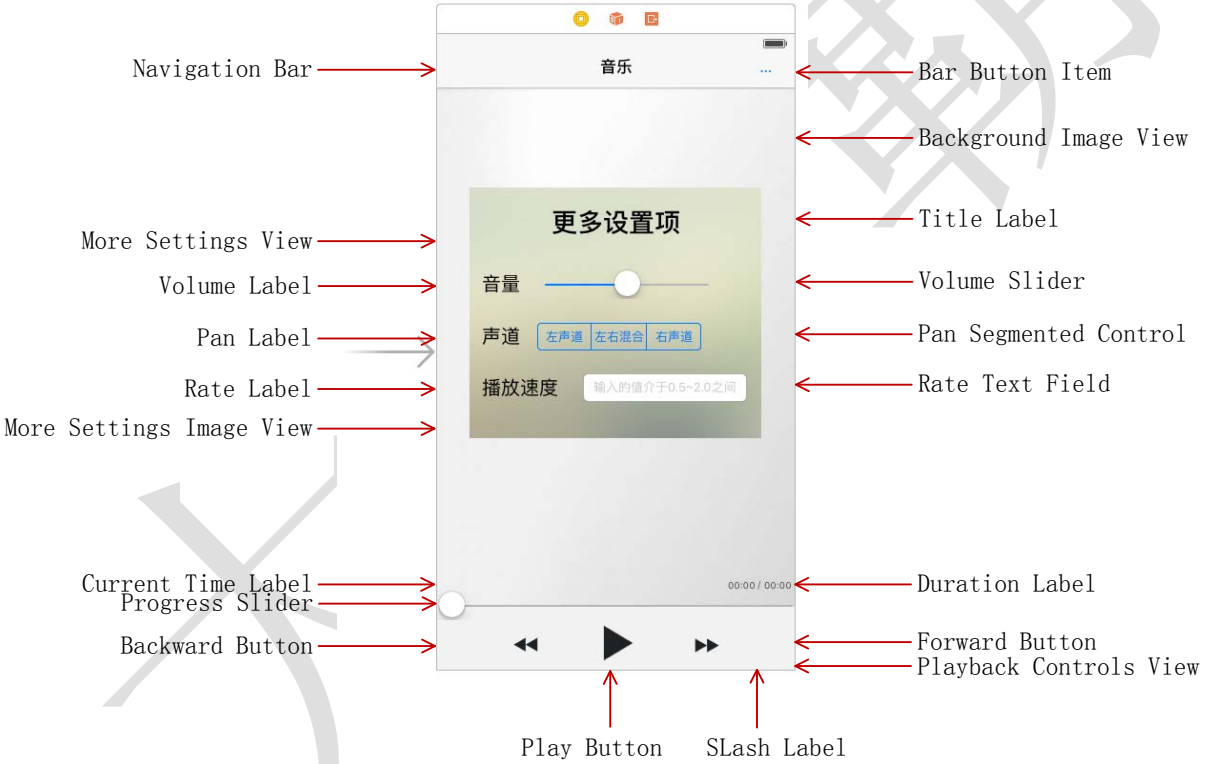

图 2.13　主视图的效果

注意：一些资深的开发人员在设计界面时都会使用代码进行，但是本人认为使用故事面板设计界面更加直接，并且可以减轻开发者的代码量。

在界面中会看到很多控制音频文件的控件。在这里，我们需要讲解的播放功能使用到的控件为 Play Button 按钮控件，对于其他的控件我们就会后面进行讲解。

2.2.3　关联

对于在主视图上添加的视图或控件，在使用它们时必须要与插座变量进行关联。插座变量其实就是

为主视图中的视图或者控件起的别名，类似于实例化的对象。将主视图中的 Play Button 按钮控件与插座变量 playButton 进行关联。具体的操作步骤如下：

（1）使用设置编辑器的三个视图方式的图标，如图 2.14 所示，将 Xcode 的界面调整为如图 2.15 所示的效果。

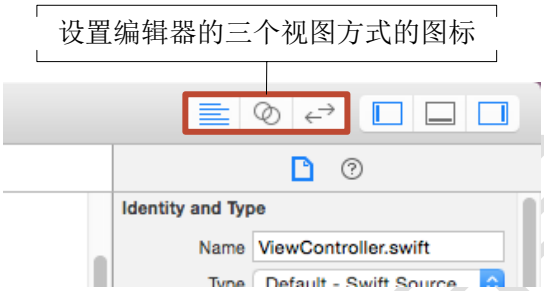

图 2.14　编辑器的三个视图方式的图标

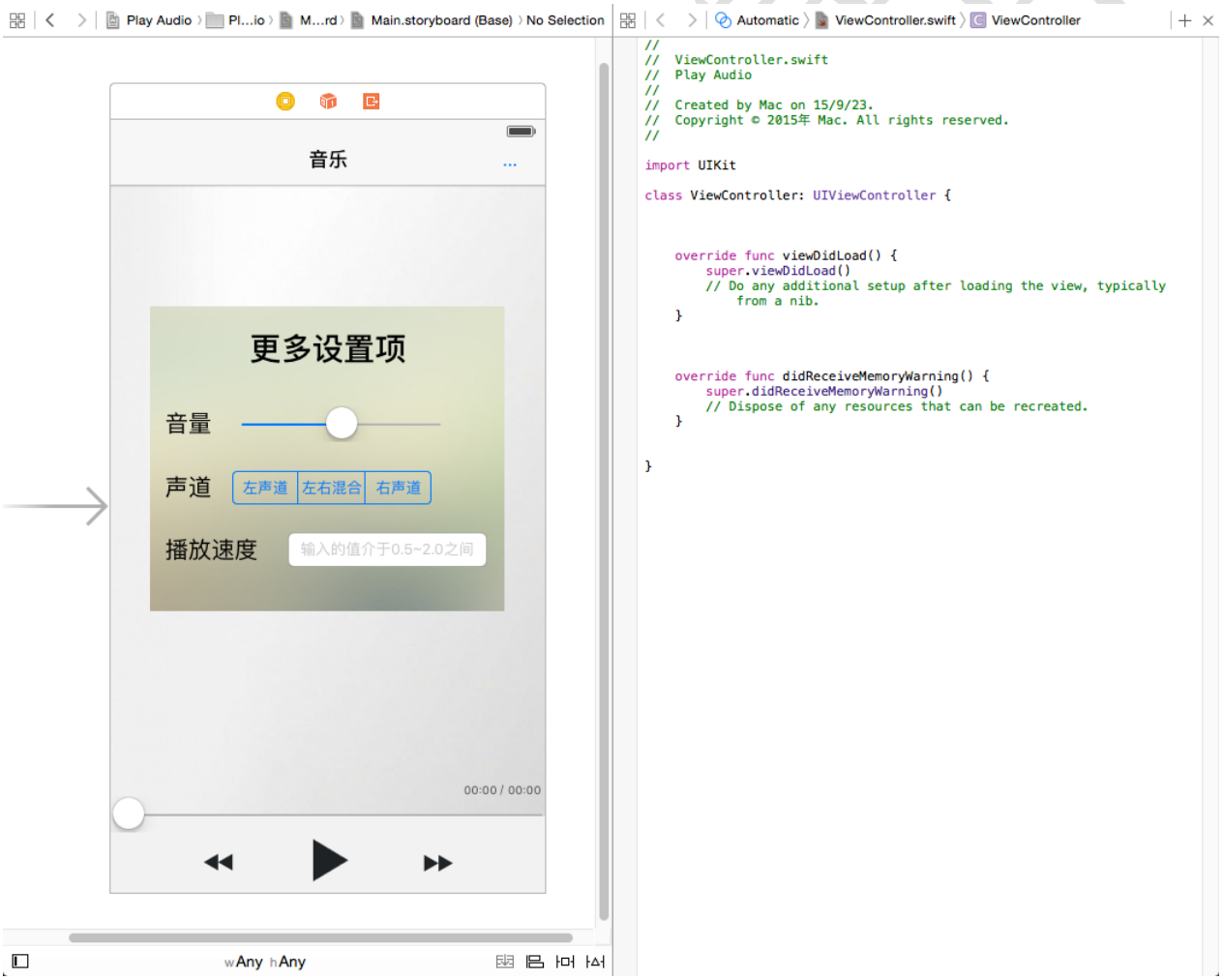

图 2.15　调整界面

（2）按住 Ctrl 键拖动主视图中的 Play Button 对象，这时会出现一个蓝色的线条，将这个蓝色的线条拖动到 ViewController.swift 文件中，如图 2.16 所示。

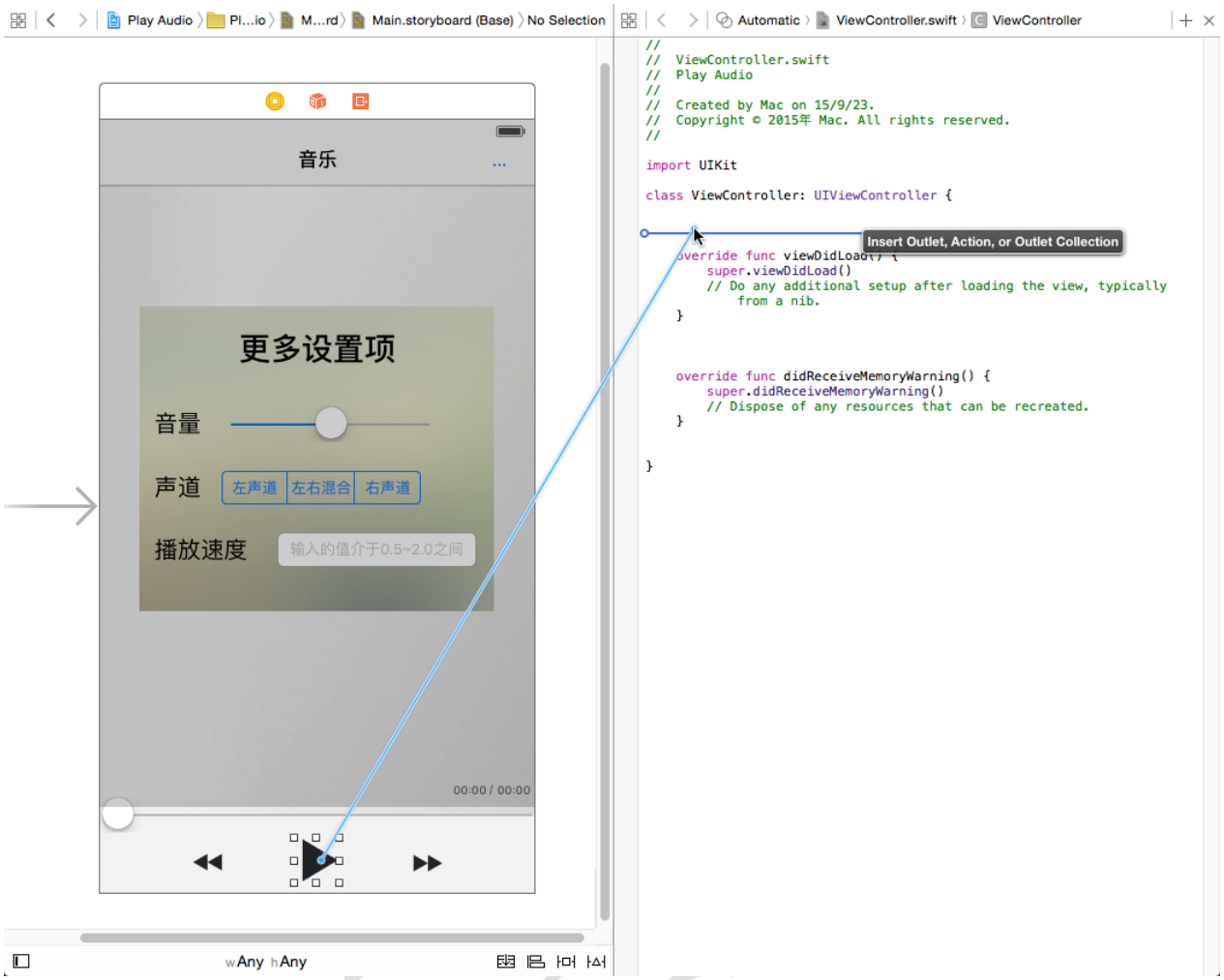

图 2.16　出现蓝色的线条

（3）松开鼠标后，会弹出一个对话框，如图 2.17 所示。

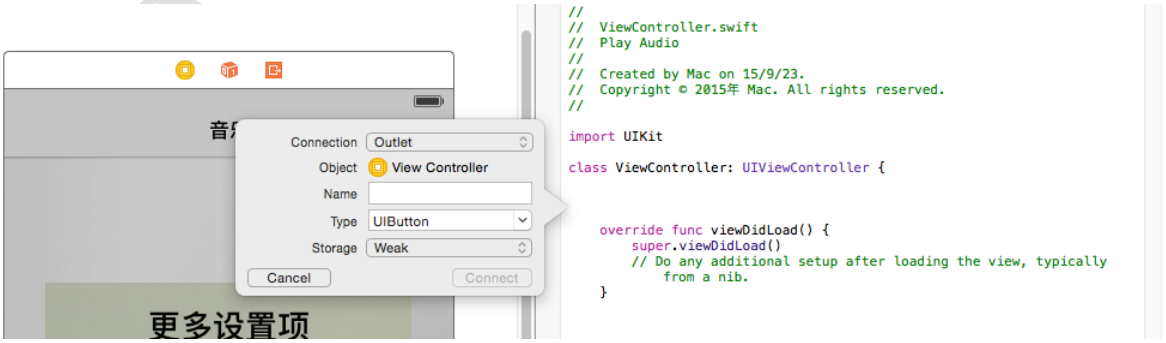

图 2.17　弹出对话框

（4）在 Name 文本框中输入名称 playButton，如图 2.18 所示。

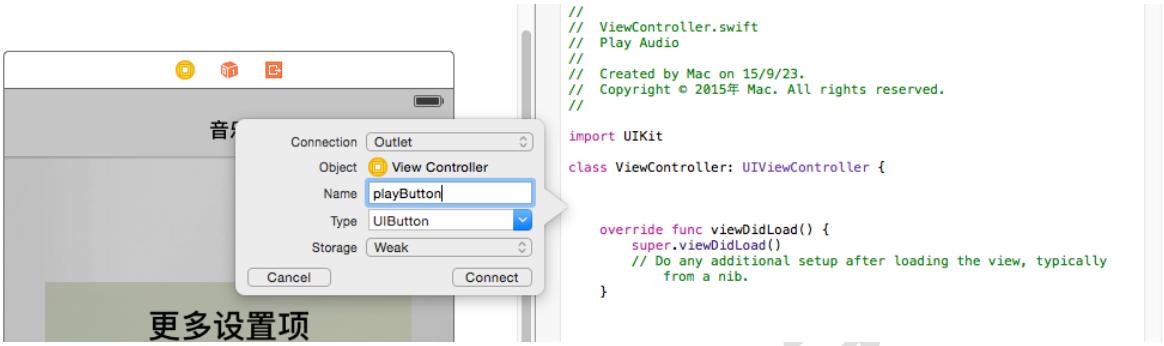

图 2.18　输入名称

注意：Name 这一项输入的名称是任意的。

（5）选择 Connect 按钮，关闭对话框，这时在 ViewController.swift 文件中自动生成一行代码，如图 2.19 所示。

图 2.19　操作变量

注意：生成的代码被叫做插座变量。

将主视图中的 More Settings View 视图与插座变量 moreSettingsView 进行关联。将主视图中的 Play Button 按钮控件与动作 playAudio 进行关联（动作其实就是方法，一般使用在控件中，如按钮，开关、滑块等）。具体的操作步骤如下：

（1）使用设置编辑器的三个视图方式的图标，将 Xcode 的界面调整为和图 2.15 一样的效果。

（2）按住 Ctrl 键拖动主视图中的 Play Button 对象，这时会出现一个蓝色的线条，将这个蓝色的线条拖动到 ViewController.swift 文件中，如图 2.20 所示。

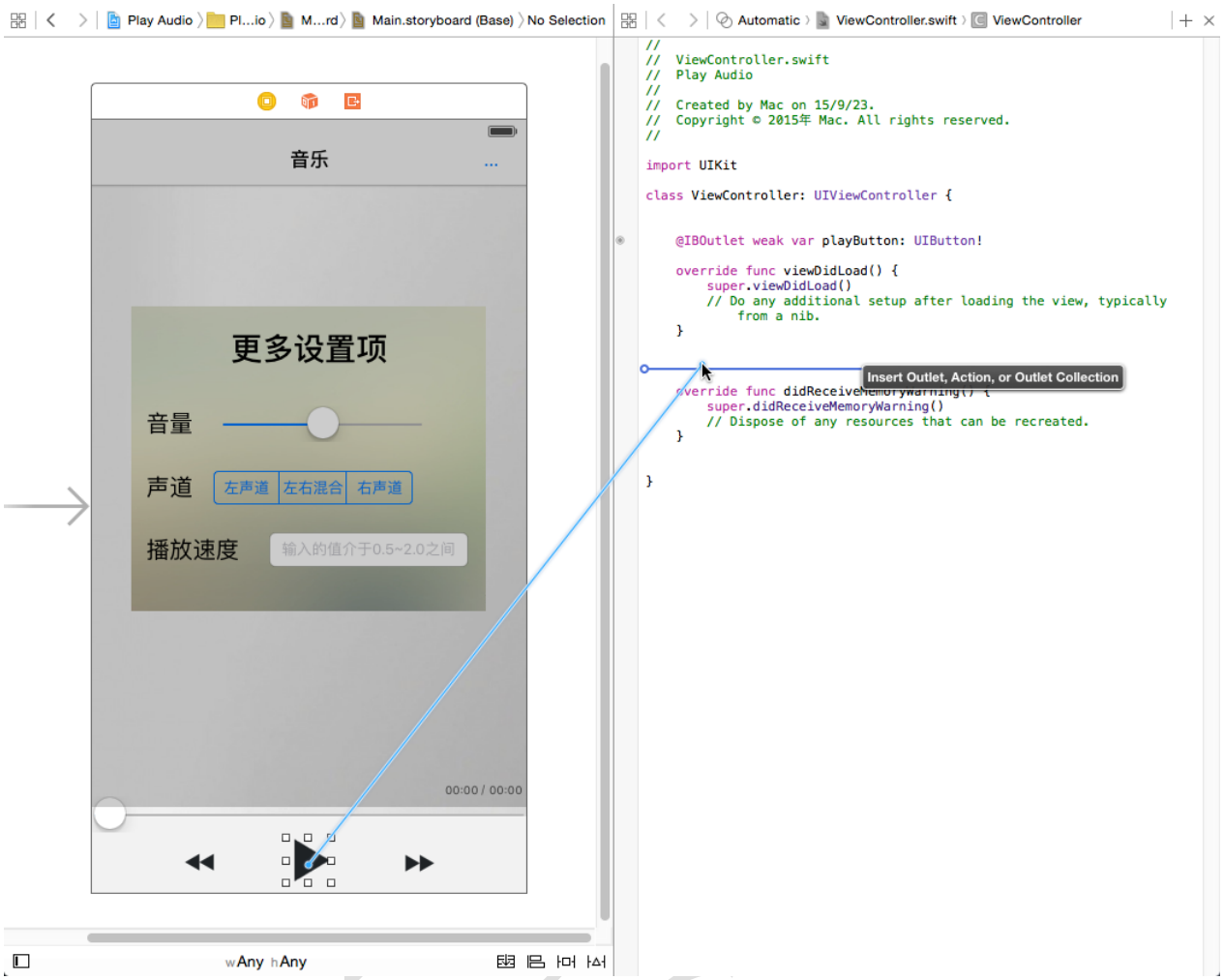

图 2.20　按住 Ctrl 键拖动界面中的按钮对象

（3）松开鼠标后，会弹出声明关联插座变量一起进行的对话框。将 Connection 选项设置为 Action，表示关联的是一个动作；将 Name 设置为 playAudio，表示关联的动作名为 playAudio，如图 2.21 所示。

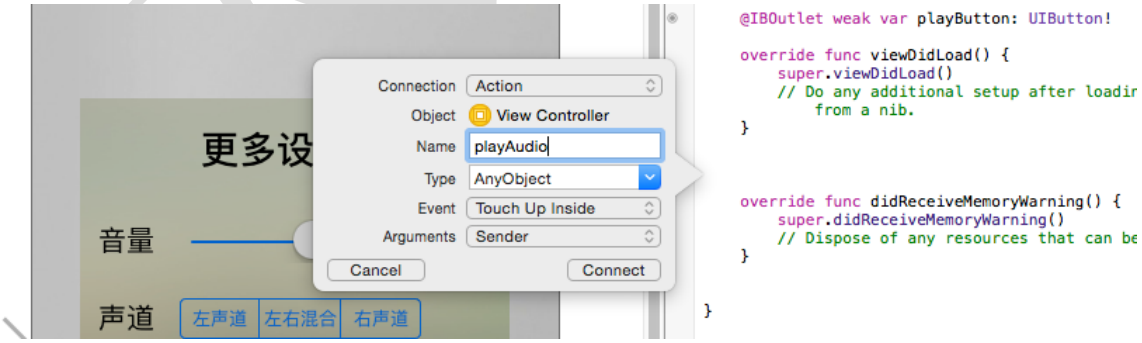

图 2.21　填写对话框

（4）将 Connection 选项设置为 Action，表示关联的是一个动作；将 Name 设置为 playAudio，表示关联的动作名为 playAudio，如图 2.22 所示。

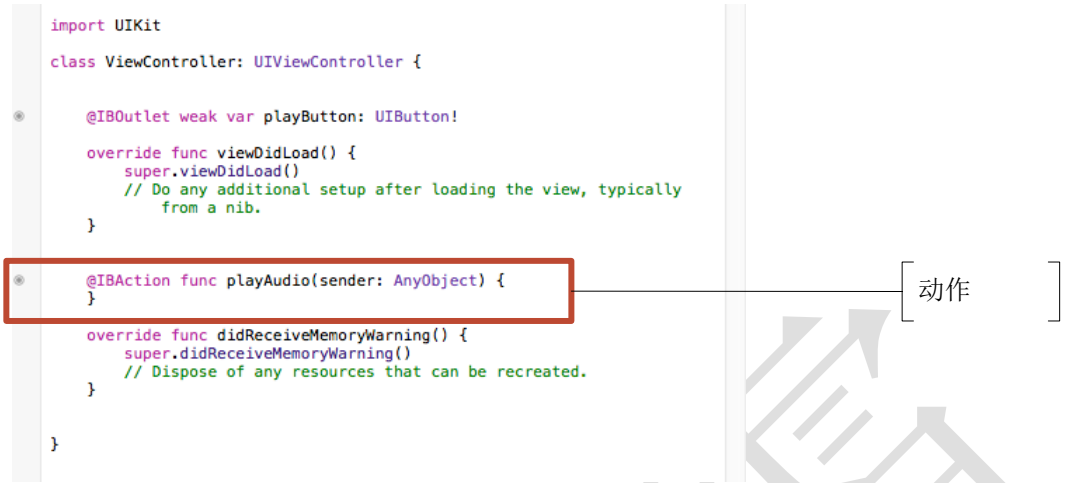

图 2.22　动作

此时，当用户轻拍 Play Button 按钮后，一个叫 playAudio()的方法就会被触发。

2.2.4　功能代码

最后就是播放音频文件的代码实现部分，也就是需要使用到在 2.1 节中所讲解的内容。打开 ViewController.swift 文件，编写代码，此代码实现音频播放的功能。代码如下：

```
import UIKit
import AVFoundation
class ViewController: UIViewController {
    @IBOutlet weak var playButton: UIButton!
    @IBOutlet weak var moreSettingsView: UIView!
    var audioEffect:AVAudioPlayer?=nil
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        moreSettingsView.hidden=true
        let path=NSBundle.mainBundle().pathForResource("Liekkas", ofType: "mp3")
        let pathURL=NSURL(fileURLWithPath: path!)
        //打开音频文件
        do {
            audioEffect=try AVAudioPlayer(contentsOfURL: pathURL)
        } catch _ {
            audioEffect = nil
        }
        audioEffect?.prepareToPlay()                                    //添加音频文件到缓存中
    }
    @IBAction func playAudio(sender: AnyObject) {
        //没有播放
        audioEffect?.play()                                             //播放声音
        playButton.setBackgroundImage(UIImage(named: "pauseImage.png"), forState: UIControlState.Normal)
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
```

```
            // Dispose of any resources that can be recreated.
        }
    }
```

此时运行程序，会看到如图 2.23 所示的效果。当开发者轻拍开始按钮，此时音频文件就会进行播放，并且开始按钮变为了暂停按钮，如图 2.24 所示。

图 2.23　初始状态

图 2.24　播放音乐

2.3　播放控制

在 2.1.1 小节中，我们通过表 2-1 和表 2-2 可以看到 AVAudioPlayer 类有很多的属性以及方法。本节将 AVAudioPlayer 类中常使用到的属性和方法进行详细的讲解。

2.3.1　暂停/停止

在音乐应用程序中都会有一个使音乐停止播放的按钮。当用户轻拍该按钮，正在播放的音乐就会停止。在 iOS 要想要正在播放的音频停止下来，可以使用 AVAudioPlayer 类中的 pause()方法和 stop()方法。

1.暂停

pause()方法的功能是暂停正在播放的音频文件，音频文件仍然准备从暂停处离开恢复播放。其语法形式如下：

```
func pause()
```

2.停止

stop()方法的功能是停止正在播放的音频文件，并撤消所需的播放设置，即如果恢复播放，音频文

件就需要重新开始播放。其语法形式如下：

```
func stop()
```

注意：stop()方法在 Xcode 6.0 之后和 pause 的功能是一样的，但是帮助文档中对 stop()的简介还是以前的。

【示例 2-1】以下将以第一个实例为基础，在播放音频的基础上实现暂停音频播放的功能。代码如下：

```
@IBAction func playAudio(sender: AnyObject) {
    audioEffect?.play()                                                    //播放声音
    playButton.setBackgroundImage(UIImage(named: "pauseImage.png"), forState: UIControlState.Normal)
    playButton.addTarget(self, action: ("pauseAudio"), forControlEvents:   UIControlEvents.TouchUpInside)
    }
func pauseAudio(){
    audioEffect?.pause()                                                   //暂停
    playButton.setBackgroundImage(UIImage(named: "playImage.png"), forState: UIControlState.Normal)
    playButton.addTarget(self, action: ("playAudio:"), forControlEvents:   UIControlEvents.TouchUpInside)
}
```

此时运行程序，在出现的模拟器界面中轻拍播放按钮，此时音频文件就会播放，并且播放按钮会变为暂停按钮；当开发者轻拍暂停按钮，播放的音频文件就会暂停播放，并且暂停按钮就变为原来的播放按钮。

注意：由于在大多数的音乐播放器中都不会出现停止控件，所在在此示例中也就不进行代码实现了。

2.3.2 前进/后退

在 AVAudioPlayer 类中有一个 currentTime 属性。该属性可以用来对当前播放的音频文件的时长进行设置的，即对当前播放的音频文件的播放位置进行设置。该属性值以秒为单位的。其语法形式如下：

```
var currentTime: NSTimeInterval
```

通过对 currentTime 属性的设置，我们可以实现在音乐应用中的前进后退功能。

1.前进

前进功能就是让正在播放的音频文件前进几秒进行播放。它的实现就是让 currentTime 属性设置的值进行固定的加法运算。

【示例 2-2】以下将以第一个实例为基础，实现前进的功能。具体的操作步骤如下：

（1）将主视图中的 Forward Button 按钮与动作 forwardAudio 进行关联。

（2）打开 ViewController.swift 文件，编写代码，实现前进功能。代码如下：

```
@IBAction func forwardAudio(sender: AnyObject) {
   //判断音频文件是否正在播放
    if(audioEffect!.playing){
        //音频文件正在播放，实现前进
        let desiredTime=(self.audioEffect?.currentTime)!+20.0
        //判断设置的时间是否小于音频文件的总时间
        if(desiredTime<self.audioEffect?.duration){
            self.audioEffect?.currentTime=desiredTime                      //设置当前播放的时间
        }
    }else{
        let alertController = UIAlertController(title: "提示", message: "音乐没有开始播放", preferredStyle:
UIAlertControllerStyle.Alert)
```

```
            let action = UIAlertAction(title: "知道了", style: UIAlertActionStyle.Default, handler: nil)
            alertController.addAction(action)
            self.presentViewController(alertController, animated: true, completion: nil)
        }
    }
```

此时运行程序后，在模拟器界面中轻拍播放按钮，此时音频文件就会开始播放，当开发者轻拍前进按钮后，音频文件就会快进 20 秒然后进行播放。（每轻拍一次就会快进 20 秒的进度）。

注意：如果开发者没有轻拍播放按钮，而是直接轻拍前进按钮，那么就是弹出“音乐没有开始播放”的警告视图，如图 2.25 所示。

图 2.25　警告视图

2.后退

后退功能就是让正在播放的音频文件后退几秒进行播放。它的实现就是让 currentTime 属性设置的值进行固定的减法运算。

【示例 2-3】以下将以第一个实例为基础，实现后退的功能。具体的操作步骤如下：

（1）将主视图中的 Backward Button 按钮与动作 backwardAudio 进行关联。

（2）打开 ViewController.swift 文件，编写代码，实现后退功能。代码如下：

```
@IBAction func backwardAudio(sender: AnyObject) {
    if(audioEffect!.playing){
        //实现后退
        let desiredTime=(self.audioEffect?.currentTime)!-20.0
        //判断指定的时间是否为 0
        if(desiredTime<0){
            self.audioEffect?.currentTime=0.0
        }else{
            self.audioEffect?.currentTime=desiredTime
        }
    }else{
        let alertController = UIAlertController(title: "提示", message: "音乐没有开始播放", preferredStyle:
UIAlertControllerStyle.Alert)
        let action = UIAlertAction(title: "知道了", style: UIAlertActionStyle.Default, handler: nil)
        alertController.addAction(action)
        self.presentViewController(alertController, animated: true, completion: nil)
        }
    }
}
```

此时运行程序后，在模拟器界面中轻拍播放按钮，此时音频文件就会开始播放。当开发者轻拍后退按钮后，音频文件就会后退 20 秒然后进行播放。（每轻拍一次就会后退 20 秒的进度）。

注意：如果开发者没有轻拍播放按钮，而是直接轻拍后退按钮，那么就是弹出“音乐没有开始播放”的警告视图。

2.3.3 音量设置

音量又称响度、音强，是指人耳对所听到的声音大小强弱的主观感受，其客观评价尺度是声音的振幅大小。在 iOS 的应用中，经常会出现播放的音乐音量过大或者过小。此时开发者可以使用 AVAudioPlayer 类中的 volume 属性对音频文件的音量进行设置。其语法形式如下：

```
var volume: Float
```

其中，该属性设置的值是浮点类型，范围在 0.0 到 1.0 之间。当设置的值为 0.0 时，表示静音；当设置的为为 1.0 时，表示最大音量。

注意：如果开发者想要让耳机系统发挥出最佳状态，谨记把音量调到最合适的水平，如果你对现场真实的音量不熟悉，那么就多找机会去听现场！

【示例 2-4】以下将以第一个实例为基础，实现音量的调节。具体的操作步骤如下：

1.显示更多设置项

（1）将主视图中的 Bar Button Item 控件与动作 showMoreSettingsView 进行关联。

（2）打开 ViewController.swift 文件，编写代码，实现显示更多设置项，代码如下：

```
@IBAction func showMoreSettingsView(sender: AnyObject) {
    UIView.beginAnimations("", context: nil)
    UIView.setAnimationDuration(2)
    moreSettingsView.hidden=false
    UIView.commitAnimations()
}
```

2.设置音量

（1）将主视图中的 Volume Slider 与插座变量 volumeSlider 进行关联。

（2）将主视图中的 Volume Slider 与动作 setVolume 进行关联。

（3）打开 ViewController.swift 文件，编写代码，首先需要在 viewDidLoad()方法中添加一行代码，实现对音量的设置，这个音量是初始音量，代码如下：

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    moreSettingsView.hidden=true
    let path=NSBundle.mainBundle().pathForResource("Liekkas", ofType: "mp3")
    ……
    audioEffect?.prepareToPlay()
    audioEffect?.volume=volumeSlider.value                    //设置音量大小
}
```

（4）在 ViewController.swift 文件中的 setVolume()动作编写代码，实现通过滑块调节音频大小的功能。代码如下：

```
@IBAction func setVolume(sender: AnyObject) {
    audioEffect?.volume=volumeSlider.value
    self.performSelector(("hideMoreSettingsView"), withObject: self, afterDelay: 3)
}
//隐藏更多设置项界面
func hideMoreSettingsView(){
    moreSettingsView.hidden=true
}
```

此时运行程序，在模拟器界面中轻拍播放按钮，此时音频文件就会开始播放，当开发者轻拍导航栏

中的更多按钮后，会弹出更多设置项的界面，如图 2.26 所示。当开发者可以滑动滑块来控件音量的大小。

图 2.26　设置音量

2.3.4　声道设置

为了让声音可以还原真实的效果，所以有了声道的产生。声道是指声音在录制或播放时在不同空间位置采集或回放的相互独立的音频信号。通过声道，开发人员可以调整声音的强弱以及延迟模拟各种真实的声音效果。一般情况下声音在录制时采用设备的不同，可以分为单声道、立体声、3D 环绕声、四声环绕、5.1 声道和 7.1 声道。

1.单声道

所谓的单声道，就是声音只由一只音箱产生，听众可以很明显地听出声音的来源就是音箱所摆放的位置，其本身的表现力较为平淡；当通过两个扬声器回放单声道信息的时候，我们可以明显感觉到声音是从两个音箱正中间传递到我们耳朵里的。这种缺乏位置感的录制方式用现在的眼光看自然是很落后的，但在声卡刚刚起步时，已经是非常先进的技术了。

2.立体声

单声道缺乏对声音的位置定位，而立体声技术则彻底改变了这一状况。它利用了两个独立声道进行录音，整个过程不加任何的声音处理。立体声系统的再现需要一对音箱来完成，它通过调整系统中两只音箱发出声音的大小，让我们误认为声源来自两只音箱之间直线段中的任意位置。特别是当使用耳机的时候，由于左右两边的声音串音情况很少发生，所以声音的定位比较准确；再加上比较真实的音场感觉，它的表现力比单声道真实得多。立体声虽然可以改变单声道缺乏对声音的位置定位功能，但是它的不足也是很明显的。例如，对音箱的位置摆放要求较高，摆位的不好会直接影响声音的表达。

3.3D 环绕声

3D 环绕声有时也称作 3D 增强立体声（3D Enhancement）。它是一种模拟环绕声系统。左、右声道的立体声信号，经过数字信号处理后，通过左、右两路音箱，产生三维的环绕声场效果。

4.四声环绕

四声道环绕规定了 4 个发音点，分别为前左、前右，后左、后右。听众则被包围在这中间。同时还建议增加一个低音音箱，以加强对低频信号的回放处理。这也就是如今 4.1 声道音箱系统广泛流行的原因。就整体效果而言，四声道系统可以为听众带来来自多个不同方向的声音环绕，可以获得身临其境的听觉感受，给用户以全新的体验。如今四声道技术已经广泛融入于各类中高档声卡的设计中，成为未来发展的主流趋势。

5.5.1 声道

5.1 声道已广泛运用于各类传统影院和家庭影院中。一些比较知名的声音录制压缩格式，譬如杜比 AC-3（Dolby Digital）、DTS 等都是以 5.1 声音系统为技术蓝本的。其中“.1”声道，则是一个专门设计的超低音声道，这一声道可以产生频响范围 20～120Hz 的超低音。其实 5.1 声音系统来源于 4.1 环绕，不同之处在于它增加了一个中置单元。这个中置单元负责传送低于 80Hz 的声音信号，在欣赏影片时有利于加强人声，把对话集中在整个声场的中部，以增加整体效果。

6.7.1 声道

7.1 声道系统的作用简单来说就是在听者的周围建立起一套前后声场相对平衡的声场。

通过不同声道模式采集的声音（录音的声音）在播放时由于选择的播放声道的模式不同，会产生不一样的播放效果。声音在播放时的声道模式有 3 种，分别为左声道、右声道以及左右混合。以下就是对这 3 种在播放时声道模式的介绍。

- 左声道：电子设备中模拟人类左耳的听觉范围产生的声音输出。
- 右声道：电子设备中模拟人类右耳的听觉范围产生的声音输出。
- 左右混合：电子设备中模拟人类左右耳的听觉范围产生的声音输出。

在 iOS 中对于声音在播放时的声道模式是可以进行设置的，此时需要使用到 pan 属性，其语法形式如下：

```
var pan: Float
```

其中，该属性设置的值可以为-1.0、0.0 以及 1.0。这 3 个值的说明如下：

- -1.0：表示左声道。
- 0.0：表示左右混合。
- 1.0：表示右声道。

【示例 2-5】以下将以第一个实例为基础，实现声道模式的设置。具体的操作步骤如下：

（1）将主视图中的 Pan Segmented Control 与插座变量 panSegmentedControl 进行关联。

（2）将主视图中的 Pan Segmented Control 与动作 setPan 进行关联。

（3）打开 ViewController.swift 文件，编写代码，首先需要在 viewDidLoad()方法中添加一行代码，实现对声道模式的设置，这个声道是初始声道。代码如下：

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    moreSettingsView.hidden=true
    let path=NSBundle.mainBundle().pathForResource("Liekkas", ofType: "mp3")
    ……
    audioEffect?.volume=volumeSlider.value
    audioEffect?.pan = -1.0                                    //将声道设置为左声道
```

```
}
```

（4）在 ViewController.swift 文件中的 setPan 动作编写代码，实现通过声道模式的设置。代码如下：

```
@IBAction func setPan(sender: AnyObject) {
    let index=panSegmentedControl.selectedSegmentIndex
    if(index==0){
        audioEffect?.pan = -1.0                    //将声道设置为左声道
        hideMoreSettingsView()
    }else if(index==1){
        audioEffect?.pan = 0.0                     //将声道设置为左右混合
        hideMoreSettingsView()
    }else if(index==2){
        audioEffect?.pan = 1.0                     //将声道设置为右声道
        hideMoreSettingsView()
    }
}
```

此时运行程序，在模拟器界面中轻拍播放按钮，音频文件就会开始播放，此时听到的声音是来自左声道的；当开发者轻拍导航栏中的更多按钮后，会弹出更多设置项的界面。在声道这一项中开发者可以选择声音进行播放时所使用的声道模式。

2.4　控制播放速度

音频文件在播放时是以一定的速度进行的。这个速度是可以进行更改的，从而实现音频文件的快速播放和慢速播放功能。要实现播放速度的更改需要使用 AVAudioPlayer 类中的 rate 属性实现。其语法形式如下：

```
var rate: Float
```

其中，该属性设置的值为浮点类型，范围在 0.5 到 2.0 之间。如果该属性的值设置为 1.0 表示正常播放，它也是默认值。2.0 表示以最快的速度进行播放，0.5 表示以最慢的速度进行播放。

注意：如果开发者要实现播放速度的改变则必须要对 enableRate 属性进行设置，该属性的功能是否允许改变播放速度。其语法形式如下：

```
var enableRate: Bool
```

其中，该属性设置的值为布尔类型。将值为 true 时，表示允许改变播放速度，反之，则不运行改变播放速度。

【示例 2-6】以下将以第一个实例为基础，让用户可以控制音频文件的播放速度。具体的操作步骤如下：

（1）将主视图中的 Rate Text Field 文本框与插座变量 rateTextField 进行关联。

（2）将主视图中的 Rate Text Field 文本框与动作 setRate 进行关联。

（3）右击主视图中的 Rate Text Field 文本框，在弹出的 Rate Text Field 对话框，如图 2.27 所示。

（4）选择 Sent Events 下的 Did End On Exit 选项，将此选项和 dock 中的 View Controller 进行关联，如图 2.28 所示。

（5）Did End On Exit 选项和 dock 中的 View Controller 进行关联后，会弹出当前声明的方法，如图 2.29 所示。

图 2.27　弹出 Rate Text Field 对话框

图 2.28　关联

（6）选择其中的 setRate:方法，此时 Did End On Exit 就与 setRate:方法进行关联了，如图 2.30 所示。

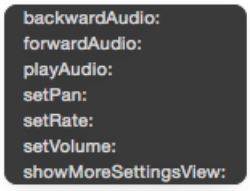

图 2.29　弹出方法对话框

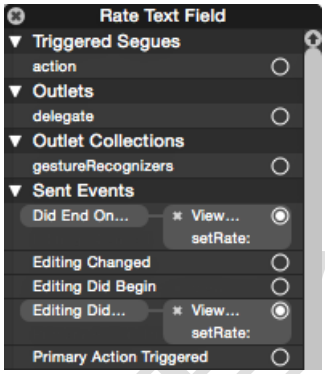

图 2.30　关联后的效果

（7）打开 ViewController.swift 文件，编写代码，实现播放速度的控制。代码如下：

```
@IBAction func setRate(sender: AnyObject) {
    rateTextField.resignFirstResponder()                                    //关闭键盘
    //判断音频文件是否没有开始播放
    if(audioEffect!.playing==false && audioEffect?.currentTime == 0.0){
        //没有开始播放
        audioEffect?.enableRate=true                                        //允许改变播放速度
        let rateValue=NSString(string: rateTextField.text!).floatValue
        audioEffect?.rate=rateValue                                         //设置播放速度
        hideMoreSettingsView()
    }else{
        //开始播放
        let alertController = UIAlertController(title: "提示", message: "音乐开已经播放，设置无效", preferredStyle:
UIAlertControllerStyle.Alert)
        //关闭更多设置项
        let action = UIAlertAction(title: "知道了", style: UIAlertActionStyle.Default){
            (action: UIAlertAction!) -> Void in
            self.hideMoreSettingsView()
        }
        alertController.addAction(action)
        self.presentViewController(alertController, animated: true, completion: nil)
    }
}
```

此时运行程序，在出现的模拟器界面中轻拍更多按钮，弹出更多设置项，在播放速度对应的文本框中输入数字，按下 return 键后，退出更多设置项。轻拍播放按钮，音频文件就会进行播放了。此时音频文件播放的速度与用户输入的数字有关。

注意：如果开发者在音频文件播放一段时间后轻拍更多按钮，在弹出的更多设置项中输入播放速度，此时输入的速度是无效的，它不可以控制音频文件的播放速度，所以在按下 return 键后，会弹出“音乐已经播放，设置无效”的警告视图，如图 2.31 所示。

图 2.31　警告视图

2.5　播放进度

音频文件在播放后经过了多久以及还有多久才可以播放完毕，想必是用户所关注的问题。为了解决这一问题，在很多的音乐播放器中都会有一个播放进度，如图 2.32 所示。本节将讲解两种查看播放进度的方法。

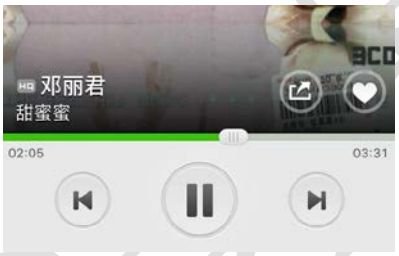

图 2.32　QQ 音乐

2.5.1　通过进度时间查看进度

通过时间查看进度就是在图 2.32 中出现的两个时间，它有一个当前播放的时间，还有一个音频文件的总时间。通过这两个时间，用户就可以知道音频文件播放了多久，还需要多久播放完毕。

1.显示音频总时间

对于音频文件中总时间的获取可以使用到 AVAudioPlayer 类的 duration 属性，其语法形式如下：

```
var duration: NSTimeInterval { get }
```

其中，该属性是一个只读属性，获取的值是一个 NSTimeInterval 类型（又名 Double）的值，它是以秒为单位的。如果要想出现图 2.32 中时间的效果，就需要将该值进行转换，转换的具体步骤如下：

（1）获取分钟数：由于使用 duration 属性获取的值为以秒为单位的，不是用户所理解的时间。由于音频文件往往都是几分钟，所以就不需要小时。将秒转换为分钟，只需要将 duration 属性获取的值除以 60 就可以了。其转换格式如下：

```
AVAudioPlayer 对象名.duration/60
```

注意：用户所理解的时间格式如下：

```
时:分:秒
```

（2）将分钟数转换为整型：由于 duration 是 NSTimeInterval 类型即双精度类型，而分钟数往往没有小数点，所以需要将此类型的值转换为整型，转换格式如下：

```
Int(分钟数)
```

（3）获取秒数：对于秒数的获取就直接使用音频文件的总时间减去转换为整型的分钟数，其语法形式如下：

```
AVAudioPlayer 对象名.duration - Int(分钟数)
```

（4）将秒数转为为整型：和为分钟数转换类型一样，需要将获取的秒数转换为整型，转换格式如下：

Int(秒数)

【示例 2-7】以下将以第一个实例为基础，获取音频文件的总时间，并显示在 Duration Label 标注中。具体的操作步骤如下：

（1）将主视图中的 Duration Label 标签控件与插座变量 durationTime 进行关联。

（2）打开 ViewController.swift 文件，编写代码，实现音频总时间的获取。代码如下：

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    moreSettingsView.hidden=true
    let path=NSBundle.mainBundle().pathForResource("Liekkas", ofType: "mp3")
    ……
    audioEffect?.pan = -1.0
    let durationMinutes:Int=Int(audioEffect!.duration/60)                    //获取分钟数
    let durationSeconds=Int(audioEffect!.duration - Double(durationMinutes * 60))    //获取秒数
    durationTime.text="\(durationMinutes):\(durationSeconds)"
}
```

此时运行程序，会看到如图 2.33 所示的效果。

图 2.33　主视图

2.获取当前播放的时间

获取当前播放的时间，我们在 2.3.2 小节中提到过需要使用到 AVAudioPlayer 类中的 currentTime 属性。

注意：此时获取的当前播放时间也是 NSTimeInterval 类型（又名 Double）。所以也需要进行转换，类型与音频文件总时间的转换。

【示例 2-8】下面将以第一个实例为基础，获取音频文件当前播放的时间，并显示在 Current Time Label 标注中。具体的操作步骤如下：

（1）将主视图中的 Current Time Label 标签控件与插座变量 currentTime 进行关联。

（2）打开 ViewController.swift 文件，声明一个时间定时器，代码如下：

```
var timer:NSTimer?=nil
```

（3）在 playAudio()动作中，添加一行代码，实现定时器的创建，代码如下：

```
@IBAction func playAudio(sender: AnyObject) {
    audioEffect?.play()                                                    //播放声音
    playButton.setBackgroundImage(UIImage(named: "pauseImage.png"), forState: UIControlState.Normal)
    playButton.addTarget(self, action: ("pauseAudio"), forControlEvents:    UIControlEvents.TouchUpInside)
    //创建定时器
    timer=NSTimer.scheduledTimerWithTimeInterval(1.0, target: self, selector: ("updateProgress"),
userInfo: nil, repeats: true)
}
```

（4）添加 updateProgress()方法，实现当前播放时间的更新。代码如下：

```
func updateProgress(){
    let  currentTimeMinutes=Int(self.audioEffect!.currentTime/60)                    //获取分钟数
    let  currentTimeSeconds=Int(audioEffect!.currentTime - Double(currentTimeMinutes*60))//获取秒数
    currentTime.text="\(currentTimeMinutes):\(currentTimeSeconds)"
}
```

此时运行程序，会看到如图 2.34 所示的效果。当轻拍播放按钮后，会看到当前时间在不停的进行更新，如图 2.35 所示。

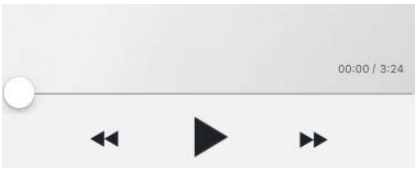

图 2.34　初始状态

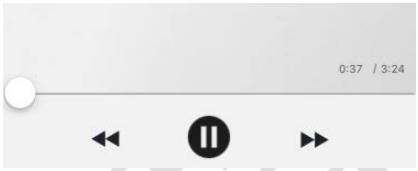

图 2.35　更新当前播放时间

注意：当用户轻拍暂停按钮后，此时的音频文件就会暂停播放，当前播放的时间停止更新，此时需要在 pauseAudio()动作中添加一行代码，实现此功能，代码如下：

```
func pauseAudio(){
    audioEffect?.stop()                                                    //暂停
    playButton.setBackgroundImage(UIImage(named: "playImage.png"), forState: UIControlState.Normal)
    playButton.addTarget(self, action: ("playAudio:"), forControlEvents:   UIControlEvents.TouchUpInside)
    timer?.invalidate()                                                    //让定时器失效
}
```

2.5.2　通过进度条查看进度

通过进度条查看进度可以让用户更为直接的看出音频文件播放了多少，还剩余多少，在图 2.32 中也使用到了此方法。在 iOS 中有两个控件可以实现进度条的功能：一种是 Progress View 控件；另一种是 Slider 控件。在本书中为了更为直观的看到音频文件多放了多少，还剩多少没有播放，我们采用 Slider 实现进度条的功能。

【示例 2-9】以下将以第一个实例为基础，实现通过进度条查看进度的功能。具体的操作步骤如下：

（1）将主视图中的 Progress Slider 滑块控件与插座变量 progressSlider 进行关联。

（2）打开 ViewController.swift 文件，在 updateProgress()方法中添加一行代码，对 Slider 控件的值进行设置，代码如下：

```
func updateProgress(){
    let  currentTimeMinutes=Int(self.audioEffect!.currentTime/60)
    let  currentTimeSeconds=Int(Double(audioEffect!.currentTime) - Double(currentTimeMinutes*60))
    currentTime.text="\(currentTimeMinutes):\(currentTimeSeconds)"
    self.progressSlider.value=Float(self.audioEffect!.currentTime/self.audioEffect!.duration)
}
```

此时运行程序，会看到如图 2.36 所示的效果。当轻拍播放按钮后，会看到当前时间在不停的进行更新，如图 2.37 所示。

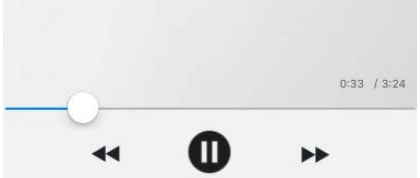

图 2.36　初始状态　　图 2.37　更新进度

2.5.3　拖动进度条播放

在音乐应用中，用户都可以以拖动进度条的方式指定音频文件播放的位置。此功能的实现还需要使用到 currentTime 属性。

【示例 2-10】以下将以第一个实例为基础，实现拖动进度条实现播放的功能。具体的操作步骤如下：

```
@IBAction func dragPlayAudio(sender: AnyObject) {
    if(audioEffect!.playing){
        //设置当前播放的时间
        audioEffect!.currentTime=Double(progressSlider.value) * Double(audioEffect!.duration)
    }else{
        let alertController = UIAlertController(title: "提示", message: "音乐没有播放，拖动无效", preferredStyle:
UIAlertControllerStyle.Alert)
        let action = UIAlertAction(title: "知道了", style: UIAlertActionStyle.Default){
            (action: UIAlertAction!) -> Void in
            self.audioEffect?.currentTime=0.0
            self.progressSlider.value=0.0
        }
        alertController.addAction(action)
        self.presentViewController(alertController, animated: true, completion: nil)
    }
}
```

此时运行程序，在出现的模拟器界面中轻拍播放按钮，此时音频文件就会播放。当开发者拖动进度条中的滑块后，音频文件就会播放滑块所在位置处的声音。

注意：拖动进度条播放并需要在音频文件播放时实现，否则就会出现“音乐没有播放，拖动无效”的警告视图，如图 2.38 所示。

图 2.38　警告视图