ToDoList项目报告

学号	姓名
191220162	张乐简

功能概述

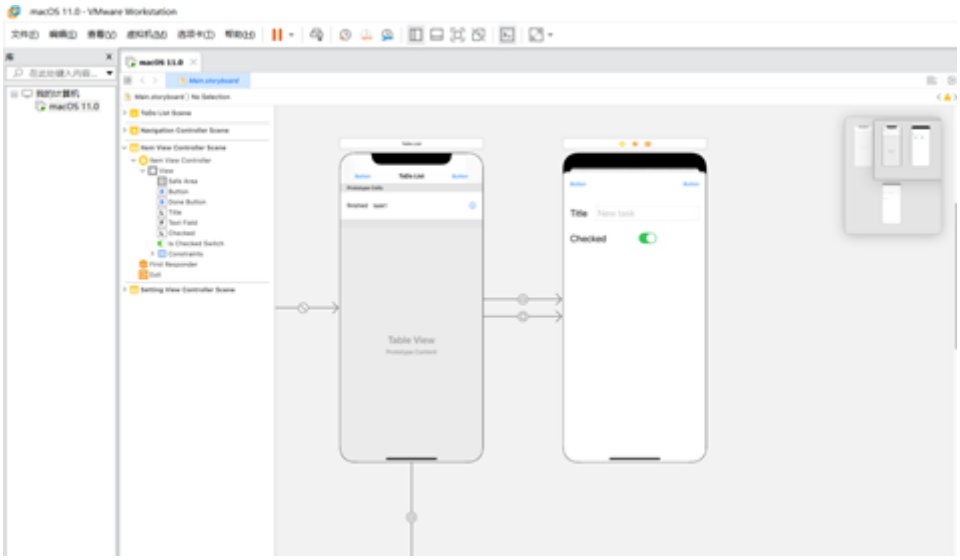
该App用表格UITableView显示待办事项，并提供了待办事项的添加、删除、修改的基本功能。另外，增加了调整App背景颜色，按事项的完成与否进行排序、以及显示、隐藏已完成事项的功能。最后，添加了保存当前待办事项和文件设置的功能。

实现过程

该部分分成三部分，依次介绍基本功能的实现、事项排序功能的实现以及App设置的实现。

基本功能

依照教学视频，采取委托设计模式完成。



基本功能的页面仅设计App的初始页面和添加或修改App的界面，不需要添加太多复杂的约束，实现过程较为简单。页面之间的切换同样依照视频利用segue实现。

事项排序

一般来说，人们更关注未完成的事项，因此我希望令未完成的事项被添加时处于已完成事项的上方，同时若一个事项的完成/等待状态被修改，改变他的位置。

为实现这一功能，首先注意到只有在新事项加入或事项被更新时才需要考虑改变事项的位置，而这两个事件已经有委托在处理，因此修改这几个处理函数即可。

```
extension ToDoTableViewController:AddItemDelegate{
    func addItem(item:ToDoItem){
        if item.isChecked==true{
            items.append(item)
        }else{
            var i:Int = 0
            while i<items.count && items[i].isChecked==false{
                i=i+1
            }
            if i==items.count{
                items.append(item)
            }else{
                items.insert(item, at: i)
            }
        }
        self.tableView.reloadData()
    }
}
extension ToDoTableViewController:EditItemDelegate{
    func editItem(newItem:ToDoItem,itemIndex:Int){
        items.remove(at: itemIndex)
        self.addItem(item: newItem)
        self.tableView.reloadData()
    }
}
```

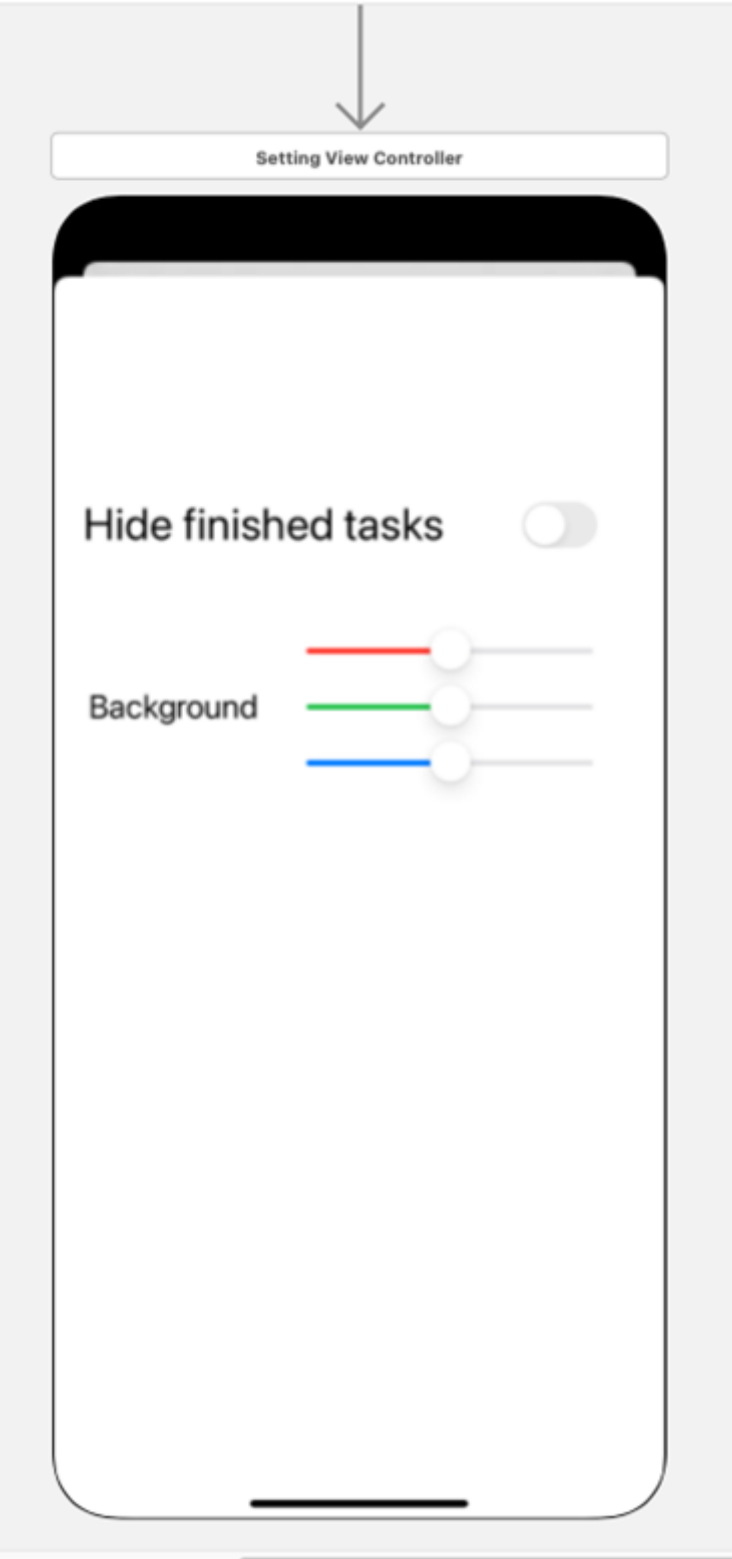
如上，在添加新事项时，首先判断他的完成状态。如果已经完成，按设计目的无论如何都会排在最后面，直接append即可；如果没有完成，假设之前的操作全部保证了未完成事项在已完成事项的前面，可以直接遍历当前的未完成事项，将其insert到它们之后。

在修改事项时，由于对事项的处理不变，干脆直接删除待修改事项，再添加一个新的修改后事项，以减少重复代码。

主要修改的代码在ToDoTableViewController，即显示待办事项的初始页面。

设置

设置功能主要包括两部分，即调整背景颜色和设置是否显示已经完成的事项。他们位于同一个新的SettingView上，由SettingViewController控制，同时都同添加事项、修改事项一样，采用委托模式实现。如下图，调整背景颜色由3个Slider、设置是否显示已完成事项由一个Switch作为用户接口。




```
protocol SettingAgent{
    func changeHideFinishedTasks()
    func setColor(r:CGFloat,g:CGFloat,b:CGFloat)
}
```

如上，两个功能共用一个SettingAgent。在用户修改设置时，SettingViewController接受修改事件，如下

```
    @IBAction func onSwitch(_ sender: Any) {
        settingAgent?.changeHideFinishedTasks()
    }
    @IBAction func onSliderChange(){
        redColor=CGFloat(redSlider.value)
        greenColor=CGFloat(greenSlider.value)
        blueColor=CGFloat(blueSlider.value)
        settingAgent?.setColor(r: redColor, g: greenColor, b: blueColor)
        self.view.backgroundColor=UIColor(red: redColor, green: greenColor,
blue: blueColor, alpha: 1)
    }
```

接着，它在事件响应函数里通过outlet解析出对应的参数，传入settingAgent，剩余事务由ToDoTableViewController代为处理。

控制已完成事项的显示

给ToDoTableViewController添加一个成员hidefinishedTasks，并根据该成员在View加载完毕时调整SettingView的内容。

```
    var hidefinishedTasks:Bool=false
    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between
presentations
        // self.clearsSelectionOnViewWillAppear = false

        // Uncomment the following line to display an Edit button in the
navigation bar for this view controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem
        self.navigationController?.navigationBar.prefersLargeTitles=true
        self.tableView.backgroundColor=UIColor(red: 1, green: 1, blue: 1, alpha:
1)
        load()
    }
```

这样便可以根据ToDoViewController的状态（由hidefinishedTasks决定），在TableView询问表格里cell数量时回应不同的值。由于已经完成的事项总是在未完成的下面，只要修改这一处便能控制已完成事项的显示与隐藏。

```
    override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        // #warning Incomplete implementation, return the number of rows
        if hidefinishedTasks==false{
            return items.count
        }
        else{
            var count:Int=0
            for item in items {
                if item.isChecked==false{
                    count=count+1
                }
            }
            return count
        }
```

```
    }
```

调整背景颜色

类似的，在viewDidLoad中令SettingView中的三个Slider显示为当前的背景颜色，上面已经提到，这里不再重复。在处理调整背景色事件时，固然要调整self.tableView.backgroundColor，也需要在切换到其他页面时，将其他页面的颜色改变。

于是，在切换至SettingView时，在prepare函数中添加对应处理。

```
        }else if segue.identifier=="setting"{
            let settingViewController=segue.destination as!
SettingViewController

            settingViewController.settingAgent=self
            settingViewController.InitScene(bColor:
self.tableView.backgroundColor!)
        }
```

其中，InitScene给SettingView的背景色赋值。

这些设置的保存和事项的保存没有区别，均采用Json实现，只是将对外的接口saveData、loadData变为save，load。

```
    func saveSettings(){
        do{
            let data=try JSONEncoder().encode(hidefinishedTasks)
            try data.write(to: configFilePath(),options: .atomic)

            var color:[Float]=[]
            let r=self.tableView.backgroundColor?.cgColor.components![0]
            let g=self.tableView.backgroundColor?.cgColor.components![1]
            let b=self.tableView.backgroundColor?.cgColor.components![2]
            color.append(Float(r!))
            color.append(Float(g!))
            color.append(Float(b!))
            let data2=try JSONEncoder().encode(color)
            try data2.write(to: configcolorFilePath(),options: .atomic)
        }catch{
            print("Cannot save: \(error.localizedDescription)")
        }
    }
    func loadSettings(){
        let path=configFilePath()
        if let data=try?Data(contentsOf: path){
            do{
                hidefinishedTasks=try JSONDecoder().decode(Bool.self, from:
data)
            }catch{
                print("Error decoding list:\(error.localizedDescription)")
            }
        }
        let path2=configcolorFilePath()
        if let data2=try?Data(contentsOf: path2){
            do{
                let color=try JSONDecoder().decode([Float].self, from: data2)
                let r=CGFloat(color[0])
```

```
                let g=CGFloat(color[1])
                let b=CGFloat(color[2])
                self.tableView.backgroundColor=UIColor(red: r, green: g, blue:
b, alpha: 1)
            }catch{
                print("Error decoding list:\(error.localizedDescription)")
            }
        }
    }
```

项目反思

委托模式提高了类的内聚性，让它们只需要专注于自己的事务即可，这是一项很好的性质，然而也提高了类之间通信的难度，需要根据情况选择设计模式。另外，待办事项还应可添加给事项分类的功能，之后可以实现。还有，保存设置时创建了太多Json文件，应考虑将其合并为1个。