

Swiftレクチャーー3回目

溝口健太

村上ろまん

今回の内容

tableViewでセルを操る能力者になろう
(ほぼコーディングに時間を使います)

今回の完成形

結構ちゃんと作ったので作る前に、コーディングしてもらってどのようなアプリになるのか先に見せたいと思います。

tableview

- Dynamic Prototypes

tableViewのセルが可変する際に使う。Static Cellsと比べてコードを書く量が多い。`delegate`メソッドの実装を必ず行わなければいけない

- Static Cells

レイアウトが一切変わらないtableViewを使いたい時に使う。SB上でセルを作れるが可変性が無い

Dynamic Prototypesで作っていきます。主にセルの再利用をして作っていくので無駄なメモリを使わずに済むのでとても便利ですね。

tableView

まず初めに指定した2つの配列の中身
をtableViewに表示してみましょう

⚠️絶対にdelegate先の設定をしましょう⚠️

tableView

```
class ViewController: UIViewController, UITableViewDelegate,  
UITableViewDataSource{  
    @IBOutlet weak var tableView: UITableView!  
    @IBOutlet weak var navigationBar: UINavigationBar!  
  
    let George = ["sore","java","de","dekinaino?"] //要素の中身は  
    なんでも構いません  
    let always_Say = ["ok","3年生質問は?","2年生質問  
    は?","Hironakakun","ゲームは時間のMUDA"]  
  
    override func viewDidLoad() {  
  
        super.viewDidLoad()  
        // Do any additional setup after loading the view,  
        // typically from a nib.  
        /*tableView.delegate = self  
        tableView.dataSource = self*/ //SB上でも指定できるけどコード  
        //上に書く場合  
    }  
}
```

tableView

```
//MARK: -UITableViewDelegate-

func numberOfRowsInSection(tableView: UITableView) ->
    Int { //section(区切り)の数を決める
    return 2
}

//MARK: -UITableViewDataSource

func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int { //各セクションのセルの数を決めてい
    var count = 0
    switch section{
    case 0: count = George.count //セクションの初めのセル数
        break
    case 1: count = always_Say.count//次セクションのセル数
        break
    default:
        break
    }
}
```

tableView

```
return count
}

func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell { //  
tableViewに表示するセルを作つて表示させている  
var cell =  
    tableView.dequeueReusableCellWithIdentifier("Cell")  
//再利用できるセルがあれば再利用する  
if cell == nil{ //セルが再利用できない場合新しくセルを作る  
    cell = UITableViewCell(style: UITableViewCellStyle.Default, reuseIdentifier: "Cell") //style = 書  
式設定 reuseIdentifier = セルの定義  
}  
switch indexPath.section{ //indexPathにはセクション番号と  
行番号が入つてます
```

tableView

```
    case 0: cell?.textLabel?.text = George[indexPath.row]
        break
    case 1: cell?.textLabel?.text = always_Say[indexPath.row]
        ]
        break
    default:
        break
    }
    return cell!
}

}
```

tableView

これで2つの配列の中身をtableView上に表示することができましたね。

今回はセクションが2つに分かれているため、SB内のIB(Interface Build)の中のStyleの項目をPlain→Groupedに変更して実行してみましょう

書式の変更をしてみよう

書式を変更して1つのセルの中に2つのテキストを表示させてみよう

書式の変更をしてみよう

```
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {      //各セクションのセルの数を決めてい  
    /*var count = 0  
    switch section{  
        case 0: count = George.count      //セクションの初めのセル数  
            break  
        case 1: count = always_Say.count//次セクションのセル数  
            break  
        default:  
            break  
    }  
    return count*/  
    return George.count  
}
```

書式の変更をしてみよう

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell { //  
tableViewに表示するセルを作つて表示させている  
  
var cell =  
    tableView.dequeueReusableCell(withIdentifier: "Cell")  
    //再利用できるセルがあれば再利用する  
  
if cell == nil{ //セルが再利用できない場合新しくセルを作る  
    cell = UITableViewCell(style: UITableViewCellStyle.  
        Subtitle, reuseIdentifier: "Cell") //style =  
        書式設定 reuseIdentifier = セルの定義  
}  
/*switch indexPath.section{ //indexPathにはセクション番号  
    と行番号が入つてます  
case 0: cell?.textLabel?.text = George[indexPath.row]  
    break  
case 1: cell?.textLabel?.text =  
    always_Say[indexPath.row]  
    break  
default:  
    break  
*/
```

書式の変更をしてみよう

```
        cell?.textLabel?.text = George[indexPath.row]
        cell?.detailTextLabel?.text = always_Say[indexPath.row]
        return cell!
    }
```

配列always_Sayは要素数が1つ多いので入れるセルがありません(当たり前だけど)表示されません。

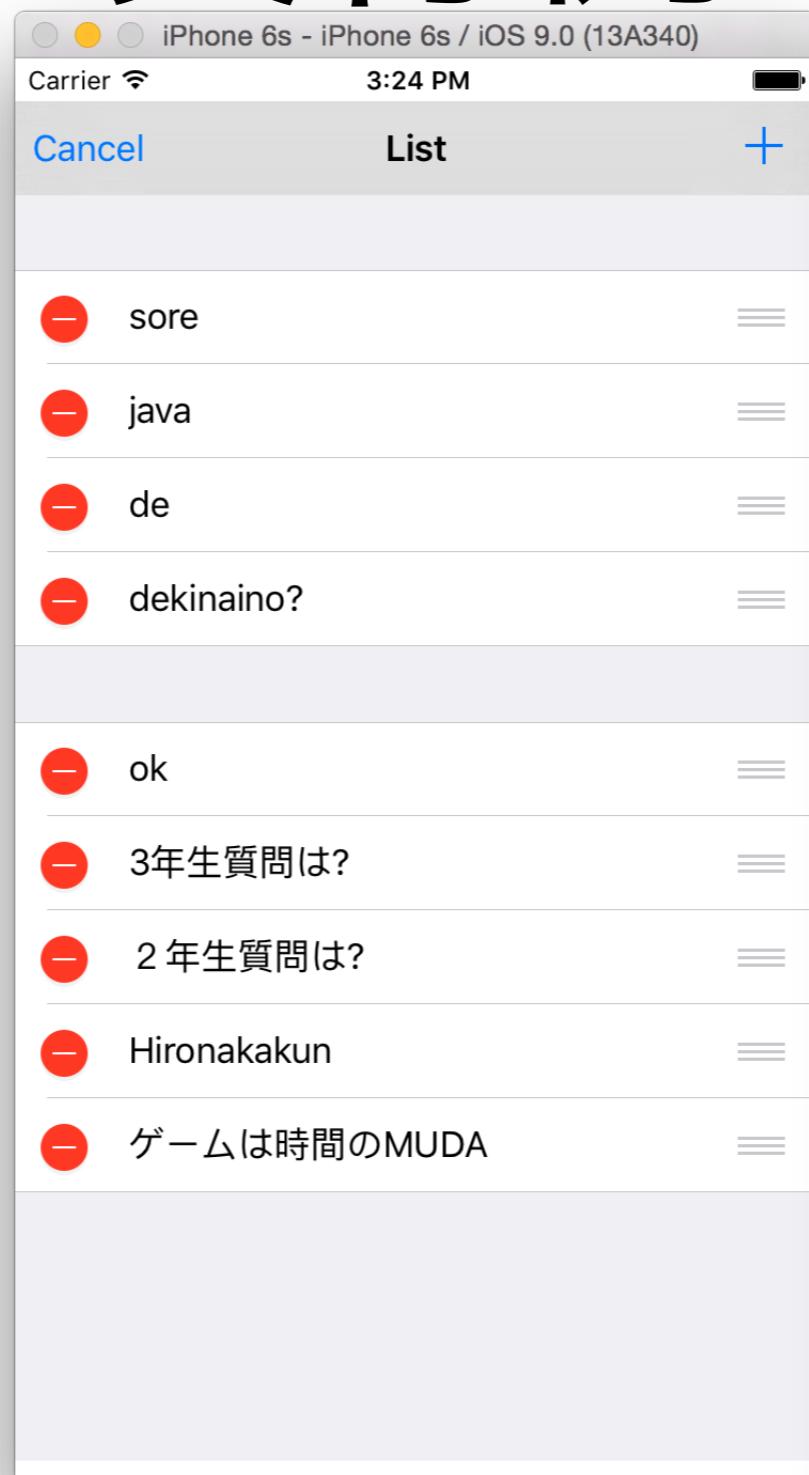
tableView

次にNavigationBarとBarButtonItemを設置して、セルの削除、ソートと(単純な)セルの追加を行います。

実行例



初期状態

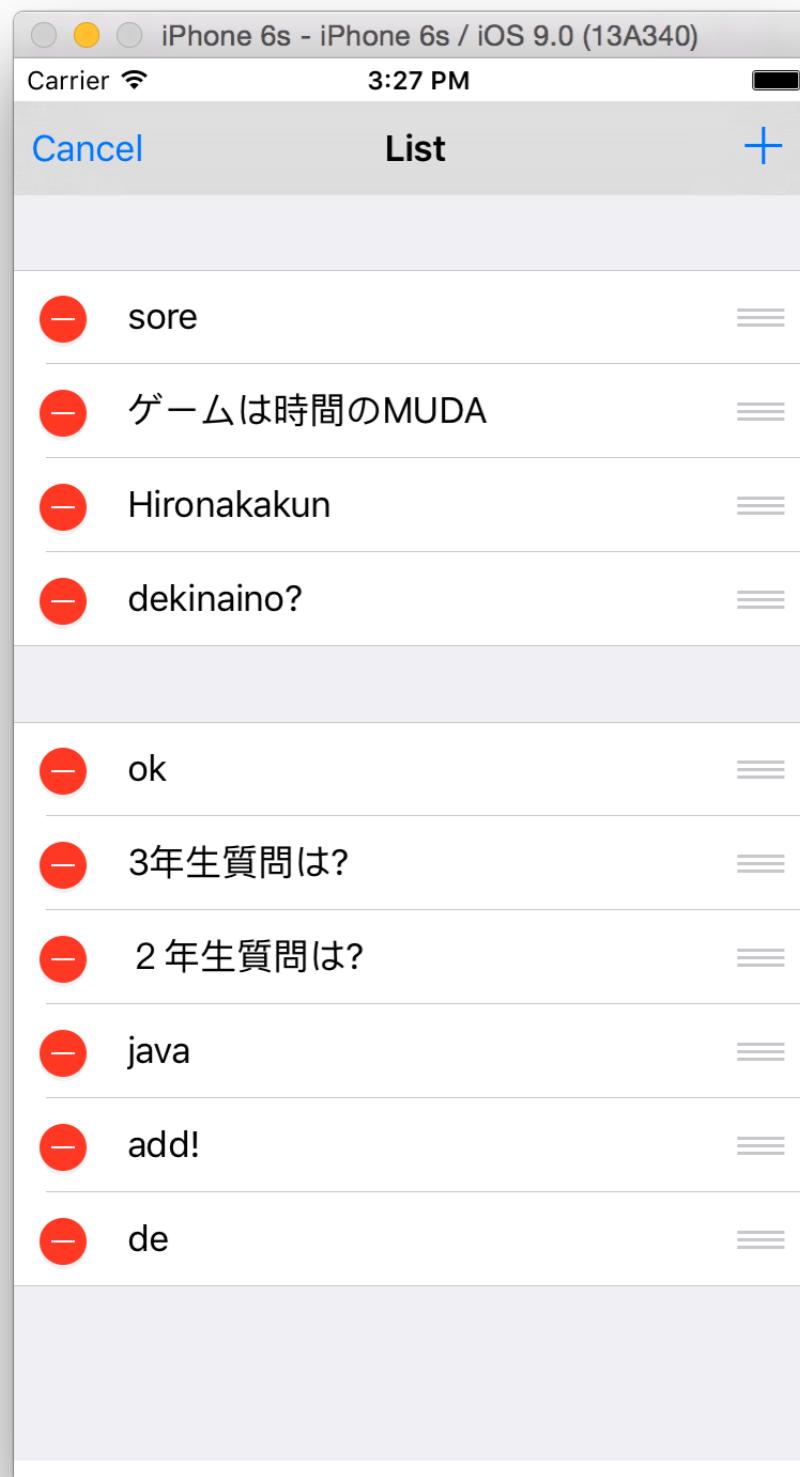


Editクリック

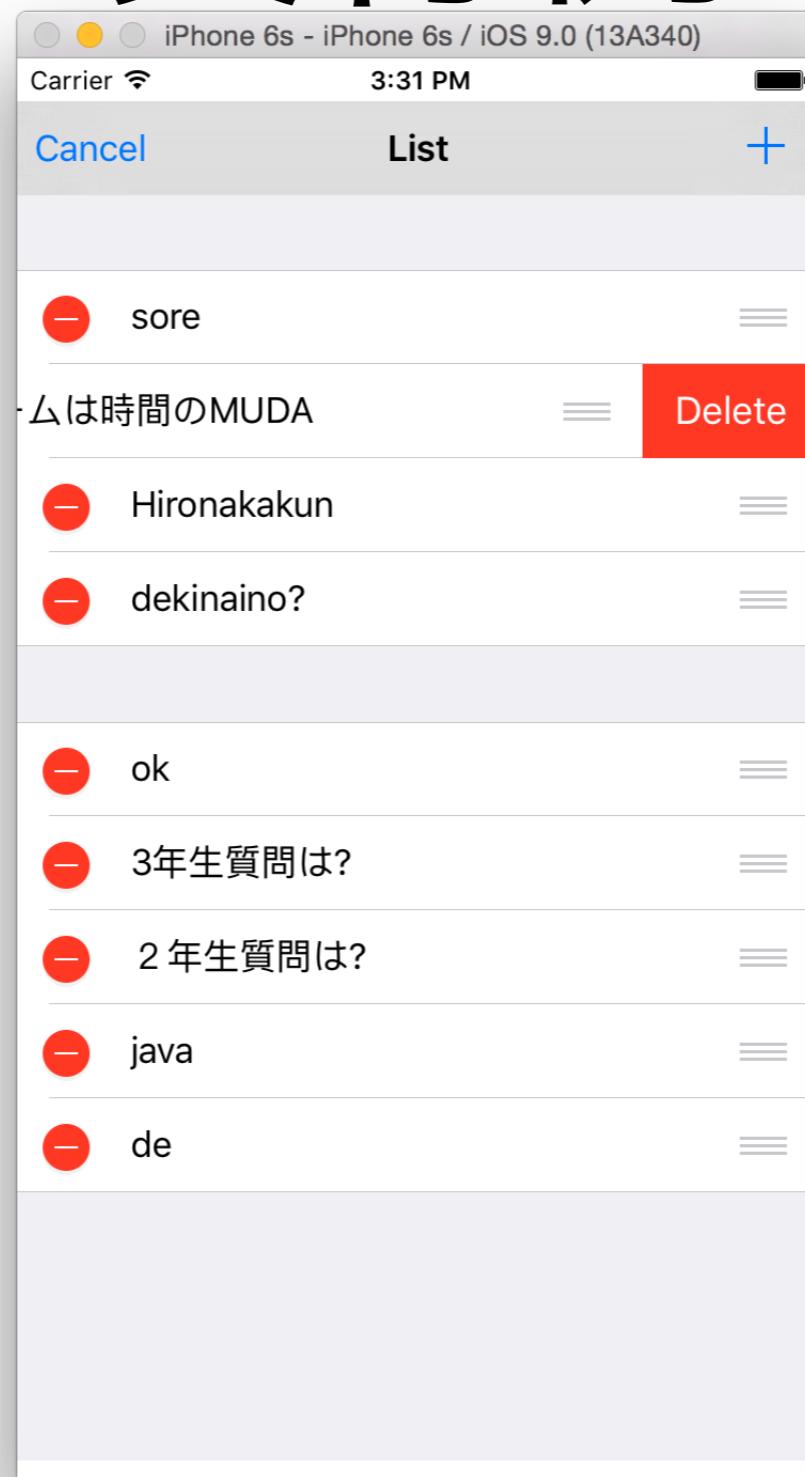


Addクリック

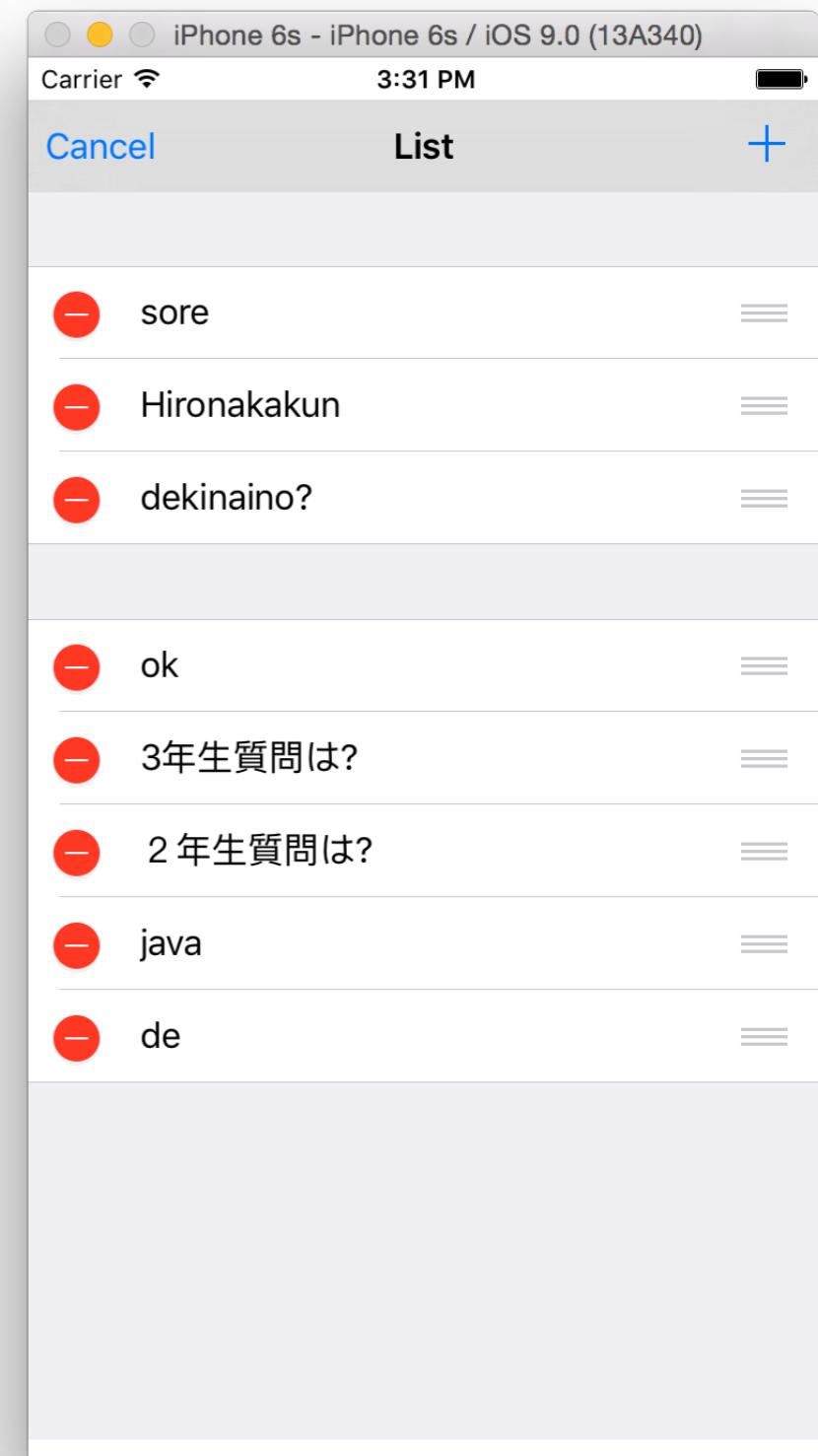
実行例



ソート



削除①



削除②

tableView

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource{  
    @IBOutlet weak var tableView: UITableView!  
    @IBOutlet weak var navigationBar: UINavigationBar!  
  
    @IBOutlet weak var add_Cell: UIBarButtonItem!  
    @IBOutlet weak var edit_Cell: UIBarButtonItem!  
  
    var George:NSMutableArray = ["sore","java","de","dekinaino?"] //可変配列であるこ  
    とを明示しておく しないとaddObjectできない  
    let always_Say:NSMutableArray = ["ok","3年生質問は?","2年生質問  
    は?","Hironakakun","ゲームは時間のMUDA"]
```

tableView

```
func get_Data(moveRowAtIndexPath sourceIndexPath:NSIndexPath,toIndexPath  
destinationIndexPath:NSIndexPath,sortArrayName1 array1:NSMutableArray,  
sortArrayName2 array2:NSMutableArray){ //主にソートの処理を行っています  
let get_Data = array1.objectAtIndex(sourceIndexPath.row) //配列に挿入する要  
素(移動元)を作成  
if sourceIndexPath.section == destinationIndexPath.section{ //同じセクショ  
ンでのソートの場合  
array1.removeObjectAtIndex(sourceIndexPath.row) //移動元のセルを削除し  
ます  
array1.insertObject(get_Data, atIndex: destinationIndexPath.row) //移  
動先にセルを挿入  
}else{ //異なるセクションでのソートの場合  
array1.removeObjectAtIndex(sourceIndexPath.row) //移動元のセルを削除し  
ます  
array2.insertObject(get_Data, atIndex: destinationIndexPath.row) //移  
動先にセルを挿入  
}  
}
```

tableView

```
//MARK: -UITableViewDelegate-

func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {    //セルの選択ができるようになる
}

func numberOfSectionsInTableView(tableView: UITableView) -> Int {    //section(区切り)の数を決める
    return 2
}

func tableView(tableView: UITableView, editingStyleForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCellEditingStyle {
    if tableView.editing{    //
        return UITableViewCellEditingStyle.Delete    //編集モード中であればセルの削除を可能にする
    }else{
        return UITableViewCellEditingStyle.None    //編集モード中でなければ左スワイプでの削除をやらせない
    }
}
```

tableView

```
func tableView(tableView: UITableView, commitEditingStyle editingStyle:  
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)  
{    //編集モード中にも呼び出され、左スワイプするとdeleteボタンが出現する素晴らしいデリ  
ゲートメソッド  
    if editingStyle == UITableViewCellEditingStyle.Delete{  
        if indexPath.section == 0{ //選択したセルの配列の要素を削除するよ  
            George.removeObjectAtIndex(indexPath.row)  
        }else{  
            always_Say.removeObjectAtIndex(indexPath.row)  
        }  
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:  
        UITableViewRowAnimation.Automatic) //削除する際にアニメーションで消すこ  
        とができるよ indexPathにはsectionとrowの2つが入ってるから選択したsectionと  
rowと一致するセルを消すイメージ  
        //tableView.reloadData() このメソッドを使えばdeleteRowsAtIndexPathsを使わな  
        くてもうまくいきます  
    }  
}
```

tableView

```
// MARK: -UITableViewDataSource- さっき書いたtableView(numberOfRowsInSection)
tableView(cellForRowAtIndexPath)メソッドは省略していますので消さないように

func tableView(tableView: UITableView, moveRowAtIndexPath sourceIndexPath:
NSIndexPath, toIndexPath destinationIndexPath: NSIndexPath) {    //セルの並び
替えをすることができます。コーディングしなければ見た目上並べ替え出来ていますが実際は配列の
要素の並べ替えは行わないのでここで実装すること
if destinationIndexPath.section == sourceIndexPath.section{ /*ここでの
sourceIndexPathが移動元 destinationIndexPathが移動先 同じセクション同士の場合
の処理を書いています*/
if destinationIndexPath.section == 0{ //sourceIndexPath.sectionでも構い
ません(同じなため)
get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:
destinationIndexPath,sortArrayName1:George,sortArrayName2:
always_Say) //レクチャーの1回目でこの書き方はやりましたね。同じ処理をい
ちいち書くのは面倒なので関数作って呼び出し
```

tableView

```
 }else{ //次のセクション同士のソートの場合
    get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:
        destinationIndexPath,sortArrayName1:always_Say,sortArrayName2:
        George)
}
}else{ //section1→section2(逆も然り)の移動を行った際の処理 これ書かないとビルト
成功しても移動した際に落ちるため
if sourceIndexPath.section == 0{ //section1→section2
    get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:
        destinationIndexPath,sortArrayName1:George,sortArrayName2:
        always_Say)
}else{ //section2→section1
    get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:
        destinationIndexPath,sortArrayName1:always_Say,sortArrayName2:
        George)
}
}
print(always_Say) //コンソーラに表示して配列もきちんとソートできているか確認してみよ
う!
print(George)
}
```

これで基本機能はできましたが

いちいちEditボタンを押して編集を行い、削除ボタンでセルの消去は少し古臭いと思いませんか？（昔のiPhone/iPodの機能みたいで）

これで基本機能はできましたが

スワイプでセルの削除ができるように
なれば

Stylish

だと思いませんか？

これで基本機能はできましたが

よってコードに少し手を加えていきた
いと思います。

スワイプした際にセルの編集もできる
ようにコーディングしていきます。
ボタンを押しても削除/編集も一応可能
のままで続けます(個人的には必要)

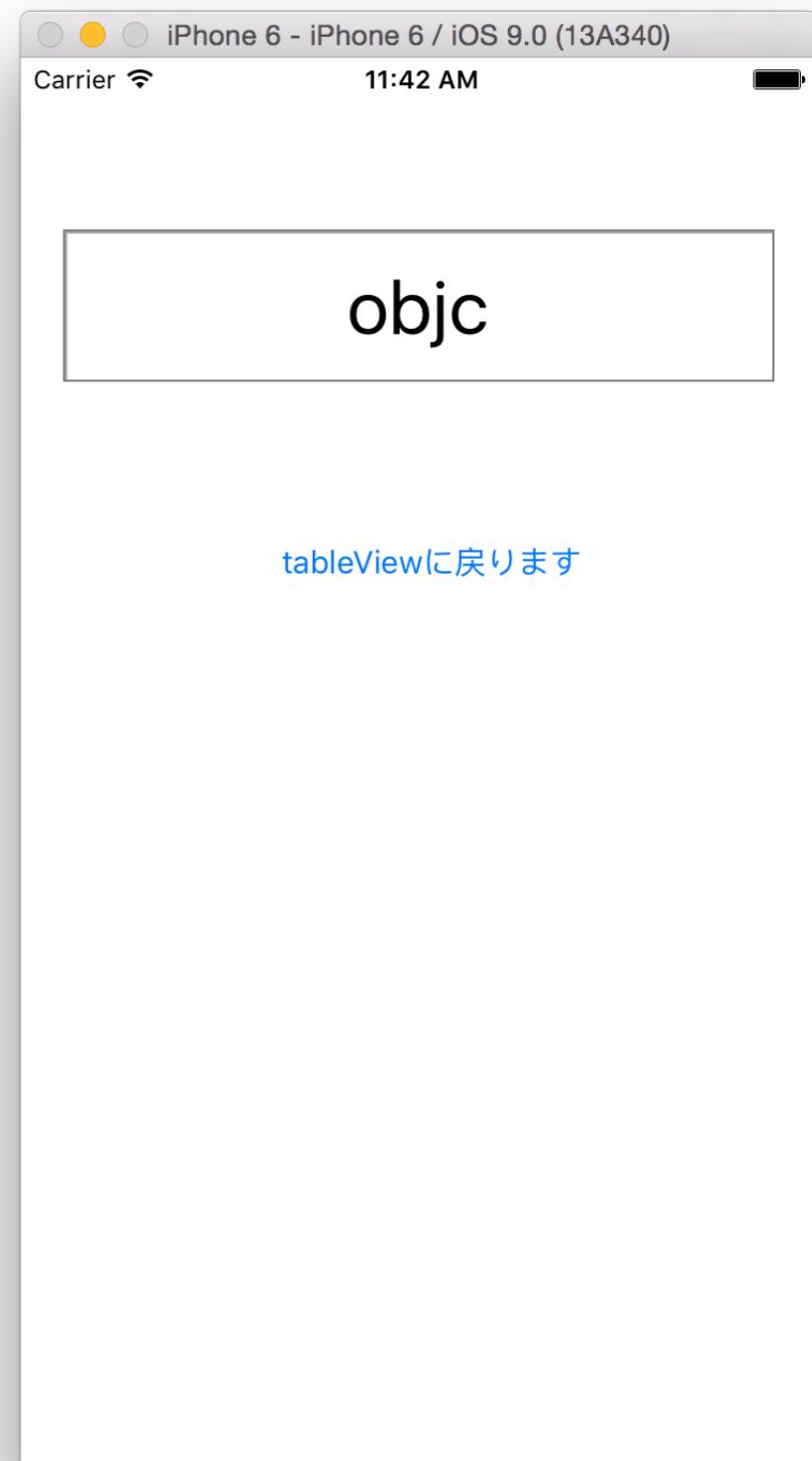
実行例



編集①



編集②



編集③

実行例



編集④

削除もきちんと出来ます

コーディングの前に

tableViewDelegateメソッド内の2つのメソッドをコメントにする所をきちんと確認して変更してください

```
/*func tableView(tableView: UITableView, editingStyleForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCellEditingStyle {
    if tableView.editing{
        return UITableViewCellEditingStyle.Delete //編集モード中であればセルの削除
            を可能にする
    }else{
        return UITableViewCellEditingStyle.None //編集モード中でなければ左スワイプでの削除をやらせない
    }
    return UITableViewCellEditingStyle.None
}*/
```

コーディングの前に

```
func tableView(tableView: UITableView, commitEditingStyle editingStyle:  
    UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)  
{    //編集モード中にも呼び出され、左スワイプするとdeleteボタンが出現する素晴らしいデリ  
    ゲートメソッド  
/*if editingStyle == UITableViewCellEditingStyle.Delete{  
    if indexPath.section == 0{ //選択したセルの配列の要素を削除するよ  
        George.removeObjectAtIndex(indexPath.row)  
    }else{  
        always_Say.removeObjectAtIndex(indexPath.row)  
    }  
    tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:  
        UITableViewRowAnimation.Automatic) //削除する際にアニメーションで消すこ  
    とができるよ indexPathにはsectionとrowの2つが入ってるから選択したsectionと  
    rowと一致するセルを消すイメージ  
    //tableView.reloadData() このメソッドを使えばdeleteRowsAtIndexPathsを使わな  
    くてもうまくいきます  
}*/  
}
```

メソッド自体は絶対にコメントにしないでください

tableView

ViewController.swift

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource,  
updateDelegate{  
  
    @IBOutlet weak var tableView: UITableView!  
    @IBOutlet weak var navigationBar: UINavigationBar!  
  
    @IBOutlet weak var add_Cell: UIBarButtonItem!  
    @IBOutlet weak var edit_Cell: UIBarButtonItem!  
  
    var George:NSMutableArray = ["sore","java","de","dekinaino?"] //可変配列であるこ  
    とを明示しておく しないとaddObjectできない  
    let always_Say:NSMutableArray = ["ok","3年生質問は?","2年生質問  
    は?","Hironakakun","ゲームは時間のMUDA"]  
    var sendStr:String? = nil //初期化しない場合はこれでもOK sendStrの値はnil(初期値な  
    し) 遷移先に送る文字列  
    var select:Int? = nil //編集の際に使います
```

tableView

ViewController.swift

```
func get_Data(moveRowAtIndexPath sourceIndexPath:NSIndexPath,toIndexPath
destinationIndexPath:NSIndexPath,sortArrayName1 array1:NSMutableArray,
sortArrayName2 array2:NSMutableArray){ //主にソートの処理を行っています
let get_Data = array1.objectAtIndex(sourceIndexPath.row) //配列に挿入する要
素(移動元)を作成
if sourceIndexPath.section == destinationIndexPath.section{ //同じセクショ
ンでのソートの場合
array1.removeObjectAtIndex(sourceIndexPath.row) //移動元のセルを削除し
ます
array1.insertObject(get_Data, atIndex: destinationIndexPath.row) //移
動先にセルを挿入
}else{ //異なるセクションでのソートの場合
array1.removeObjectAtIndex(sourceIndexPath.row) //移動元のセルを削除し
ます
array2.insertObject(get_Data, atIndex: destinationIndexPath.row) //移
動先にセルを挿入
}
}
```

tableView

ViewController.swift

```
@IBAction func edit_Cell(sender: AnyObject) {    //編集ボタンを押したら編集モードへ
    tableView.setEditing(!tableView.editing, animated: true)    //編集モードへ
    if tableView.editing{        //編集モード中なら
        edit_Cell.title = "Cancel" //このまま文字列を入れてもedit_Cell.textは初期化処理を行っていないので(見た目上は見えているけど値としては入っていません)新しくString型の変数を作り、""内の文字列を入れて条件処理を抜けた後に入れるか、IBでBarItemのtitleに文字列を追加させてください。どちらでもできるのでお好きな方で
    }else{        //編集モードでなければ
        edit_Cell.title = "Edit"
    }
}

override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "editSegue"{
        let editVC = segue.destinationViewController as! editViewController
        editVC.delegate = self
        editVC.name = sendStr
    }
}
```

tableView

ViewController.swift

```
//MARK: -updateDelegate-

func editedCell(returnStr:AnyObject){    //デリゲートで渡ってきた文字列を編集しています
    var element = 0 //element? = nilにしたかったけど面倒だったので
    if select == 0{ //編集しているセルがsection1の場合(editActionsForRowAtIndexPath
        関数で指定しています)
        element = George.indexOfObject(sendStr!)      //配列の要素番号を取り出しています
        George.replaceObjectAtIndex(element,withObject:returnStr) //取り出した
        要素番号は編集したいのでデリゲートで受け取った文字列を、この要素番号へ格納させます
    }else{
        element = always_Say.indexOfObject(sendStr!)
        always_Say.replaceObjectAtIndex(element,withObject:returnStr)
    }
}
```

tableView

ViewController.swift

```
tableView.reloadData() //セルデータの更新 ここを書かなければ配列の変更があつただけで  
画面上では表示されません  
} //ここの処理はなるべくfind()とか使ってパパッとやりたかったけど、うまくいかなかったのでこ  
のやり方にしました。配列をループで回して文字列が一致してたらデリゲートを渡った文字列をそこ  
に入れてもできると思います
```

tableView

ViewController.swift

```
func tableView(tableView: UITableView, editActionsForRowAtIndexPath indexPath: NSIndexPath) -> [UITableViewRowAction]? {    //編集モードの詳細設定をすることができます
    if indexPath.section == 0{    //遷移させる際に値渡しのための配列とその要素を決めています
        sendStr = George[indexPath.row] as? String    //配列の型が異なるのでキャストしています
        select = 0
    }else{
        sendStr = always_Say[indexPath.row] as? String
        select = 1
    }
    let edit = UITableViewRowAction(style: .Normal, title: "編集", handler:
    {(action:UITableViewRowAction(indexPath:IndexPath) in
        self.performSegueWithIdentifier("editSegue", sender: nil)    //編集画面へ遷移させる
    )})
    edit.backgroundColor = UIColor.greenColor()
```

tableView

ViewController.swift -UITableViewDelegate-

```
let delete = UITableViewRowAction(style: .Normal, title: "削除", handler:  
{(action:UITableViewRowAction, indexPath:NSIndexPath) in  
if indexPath.section == 0{ //  
    tableView(commitEditingStyle, forRowAtIndexPath)メソッドでやってたセルの  
    削除の処理をhandlerに入る handlerの引数はクロージャー  
    self.George.removeObjectAtIndex(indexPath.row)  
}else{  
    self.always_Say.removeObjectAtIndex(indexPath.row)  
}  
    tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:  
    UITableViewRowAnimation.Automatic)  
})  
delete.backgroundColor = UIColor.redColor()  
return [edit,delete]  
}
```

クロージャーとは

関数を引数に指定して別の関数に渡して利用するしくみのこと

```
{ (引数1: 型1, 引数2: 型2, ...) -> 戻り値の型 in  
    return 戻り値 }
```

クロージャーとは

```
func myFunc1(a:Int,b:Int,closure:(Int,Int)->Int){  
    print(closure(a,b))  
}  
var addClosure1:(Int,Int)->Int = { (a : Int , b : Int) -> Int in  
    return a + b  
}  
var addClosure2:(Int,Int)->Int = { (a : Int , b : Int) -> Int in  
    return a * b  
}  
  
myFunc1(3, b: 3, closure: addClosure1)  
myFunc1(3, b: 3, closure: addClosure2)
```

```
func myFunc1(a:Int,b:Int,closure:(Int,Int)->Int){  
    print(closure(a,b))  
}
```

```
myFunc1(3, b: 3, closure: { (a : Int , b : Int) -> Int in return a + b } )  
myFunc1(3, b: 3, closure: { (a : Int , b : Int) -> Int in return a * b } )
```

tableView

ViewController.swift -UITableViewDataSource

```
func tableView(tableView: UITableView, moveRowAtIndexPath sourceIndexPath:  
    NSIndexPath, toIndexPath destinationIndexPath: NSIndexPath) { //セルの並び  
    替えをすることができます。コーディングしなければ見た目上並べ替え出来ていますが実際は配列の  
    要素の並べ替えは行わないのでここで実装すること  
    if destinationIndexPath.section == sourceIndexPath.section{ /*ここでの  
        sourceIndexPathが移動元 destinationIndexPathが移動先 同じセクション同士の場合  
        の処理を書いています*/  
        if destinationIndexPath.section == 0{ //sourceIndexPath.sectionでも構い  
            ません(同じなため)  
            get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:  
                destinationIndexPath,sortArrayName1:George,sortArrayName2:  
                always_Say) //レクチャーの1回目でこの書き方はやりましたね。同じ処理をい  
                ちいち書くのは面倒なので関数作って呼び出し  
        }else{ //次のセクション同士のソートの場合  
            get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:  
                destinationIndexPath,sortArrayName1:always_Say,sortArrayName2:  
                George)  
        }  
    }
```

tableView

ViewController.swift -UITableViewDataSource

```
 }else{      //section1→section2(逆も然り)の移動を行った際の処理 これ書ないとビルド  
    成功しても移動した際に落ちるため  
    if sourceIndexPath.section == 0{ //section1→section2  
        get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:  
            destinationIndexPath,sortArrayName1:George,sortArrayName2:  
            always_Say)  
    }else{ //section2→section1  
        get_Data(moveRowAtIndexPath:sourceIndexPath,toIndexPath:  
            destinationIndexPath,sortArrayName1:always_Say,sortArrayName2:  
            George)  
    }  
}  
}
```

tableView

以上で遷移元のコーディングが終了しました。

次は、遷移先のコーディングに移ります。

tableView

editViewController.swift

```
protocol updateDelegate{      //遷移元に戻る際に編集するセルの文字列とセルデータの再更新をデリゲートとして渡します
    func editedCell(returnStr:AnyObject)
}

class editViewController: UIViewController,UITextFieldDelegate{
    @IBOutlet weak var chengeCellName: UITextField!
    var name:String? = nil
    var delegate:updateDelegate? = nil //ここ書くの忘れないように!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        chengeCellName.text = name
        chengeCellName.delegate = self //2回目でやったけどTextFieldのデリゲート先を指定しています
    }
}
```

tableView

editViewController.swift

```
@IBAction func backBtn(sender: AnyObject) {
    delegate?.editedCell(chengeCellName.text as! AnyObject) //textFieldに入っている文字列を遷移元に渡します
    dismissViewControllerAnimated(true, completion: nil) //遷移元へ戻る
}

// MARK: -UITextFieldDelegate-

func textFieldShouldReturn(textField: UITextField) -> Bool { //リターンキーが押された際の処理
    view.endEditing(true) //キーボードを閉じます
    return true //もしかしたら実機じゃないからキーボード出ない気がする
}
```

なぜデリゲートを使ったのかについて

ここで、もしかしたらなぜデリゲートを使う必要があったのかと疑問にもつ人(そもそもよくデリゲートがよくわかっていない人も)がいるかもしれないで少しだけ説明を加えようと思う

なぜデリゲートを使ったのかについて

Segueで遷移先へ
→dismissViewControllerAnimated()

で遷移元に戻る

(流石に)これはわかるよね?

やってみよう

editViewControllle.swiftの
Action接続したボタンの処理内
のdelegate?editedCell()をコ
メントにしてデリゲートをさせ
ないで実行してみよう

やってみよう

確認してわかったと思いますが、
dismissViewControllerAnimated()は遷移する
前の画面の状態に戻るだけのため、全セルの更
新を行ったかったのでデリゲートを利用したん
ですね。あとセルの変更もできないし
要はデリゲートは便利ってことを伝えたかった

次に

これで少しあはアプリっぽくなりました
が、まだセルの追加がただ同じ処理を
繰り返して増やしているのでここを変
更していきたいと思います。

次に

変更点は+ボタンをクリックすると画面遷移を行い。遷移先でどちらのセクションなのかを選ばせて選択したセクションの最後尾にセルを追加させます

実行例



section1の追加

実行例



section2の追加

先ほどの画面元への移動の相違点

遷移先へsegueで遷移するのは同じ
遷移元への値渡しをdelegateで行って
遷移しているか、同じように遷移元へ
segueで遷移して値渡ししているかの
違いです。

tableView

ViewController.swift

```
@IBAction func add_Cell(sender: AnyObject) {
    performSegueWithIdentifier("addSegue", sender: nil)
}

override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "editSegue"{
        let editVC = segue.destinationViewController as! editViewController
        editVC.delegate = self
        editVC.name = sendStr
    }else{
        let addVC = segue.destinationViewController as! addViewController
        addVC.getArray1 = George      //セクションの追加先は遷移先で決めるので配列ごと値渡
                                    //します
        addVC.getArray2 = always_Say
    }
}
```

tableView

addViewController.swift

```
class addViewController: UIViewController {
    @IBOutlet weak var selectedAddCell: UISegmentedControl!
    @IBOutlet weak var addCellName: UITextField!

    var getArray1:NSMutableArray = [],getArray2:NSMutableArray = [] //  
George(always_Sayを格納する配列を宣言(空配列))
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        addCellName.placeholder = selectedAddCell.titleForSegmentAtIndex(0) //初めは  
section1になっているから、TextFieldのプレースホルダー(説明文)をsection1に明示しま  
す
    }
}
```

tableView

addViewController.swift

```
@IBAction func selectedAddCell(sender: AnyObject) {
    if selectedAddCell.selectedSegmentIndex == 0{ //選択中のセグメント番号でプレースホルダーの名前を変更しています
        addCellName.placeholder = selectedAddCell.titleForSegmentAtIndex(0)
    }else{
        addCellName.placeholder = selectedAddCell.titleForSegmentAtIndex(1)
    }
}

@IBAction func backBtn(sender: AnyObject) {
    if addCellName.text! == ""{ //追加処理をしなかったクソユーザのために設定
        return
    }else{
        if selectedAddCell.selectedSegmentIndex == 0{
            getArray1.addObject(addCellName.text!)
        }else{
            getArray2.addObject(addCellName.text!)
        }
    }
}
```

tableView

addViewController.swift

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {  
    if segue.identifier == "backSegue" {  
        let VC = segue.destinationViewController as! ViewController  
        VC.George = getArray1  
        VC.always_Say = getArray2  
    }  
}
```

更に追加機能をつけよう

セルの削除 追加 編集の機能は
実装できました。次は何を追加
させると思いますか？

更に追加機能をつけよう

セルの検索を行えるようにして
いきたいと思います。

更に追加機能をつけよう

下スクロールで検索画面を表示
にして、検索ワードを入力で結果を表示するように実装します

更に追加機能をつけよう

UIでSearchBarがありますが、動的に
作ります。

ちなみにSearch Bar and Search
Display Controllerは現在非推奨なの
で使わないようにしましょう

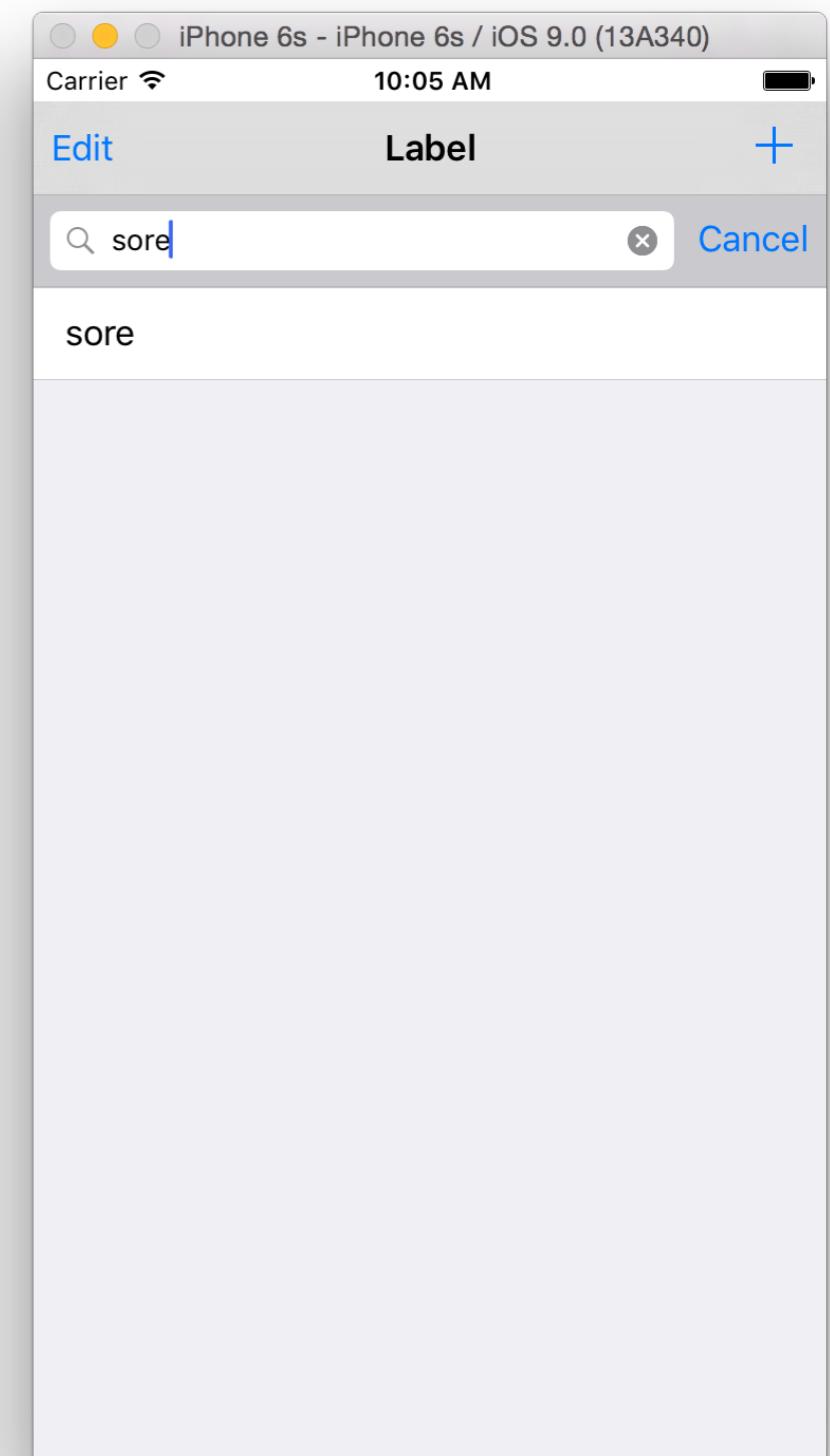
実行例



初期状態



検索バーの表示



検索①

実行例



検索②



検索③

tableView

ViewController.swift

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource,  
UISearchResultsUpdating, updateDelegate{  
  
var searchController: UISearchController!  
var searchedArray1: [String] = [] //検索結果を格納する配列  
var searchedArray2: [String] = []  
  
override func viewDidLoad() {  
  
    super.viewDidLoad()  
    // Do any additional setup after loading the view, typically from a nib.  
/*tableView.delegate = self  
tableView.dataSource = self*/ //SB上でも指定できるけどコード上に書く場合  
setSearchController()  
}  
}
```

tableView

ViewController.swift

```
func setSearchController(){
    searchController = UISearchController(searchResultsController:nil)
    tableView.contentOffset =
        CGPointMake(0.0,searchController.searchBar.bounds.size.height); //検索
    バーを始めは隠す処理
    searchController.searchResultsUpdater = self //デリゲート先の設定
    searchController.hidesNavigationBarDuringPresentation = false
    searchControllerdimsBackgroundDuringPresentation = false //検索中に画面が少
    し暗くなるのをさせない
    searchController.searchBar.searchBarStyle = UISearchBarStyle.Default //検
    索バーの種類の変更
    searchController.searchBar.placeholder = "検索ワードを入力しろ"
    searchController.searchBar.sizeToFit() //searchBarをViewのサイズに合わせる
    tableView.tableHeaderView = searchController.searchBar //searchBarを
    tableViewのヘッダー部に設置
}
```

tableView

```
func updateSearchResultsForSearchController(searchController:  
    UISearchController) { //検索バーを操作する際に呼ばれます  
    if searchController.searchBar.text?.characters.count > 0{ //検索バーに入力が  
        あった場合  
        searchedArray1.removeAll();searchedArray2.removeAll()  
        let searchPredicate = NSPredicate(format: "SELF CONTAINS %@",  
            searchController.searchBar.text!) //検索バーに入力した文字をselfから検出  
            する(VC内の配列等から)  
        let array1 = George.filteredArrayUsingPredicate(searchPredicate)  
        let array2 = always_Say.filteredArrayUsingPredicate(searchPredicate)  
        searchedArray1 = array1 as! [String] // [AnyObject]→[String] のキャスト  
        searchedArray2 = array2 as! [String] // [AnyObject]→[String] のキャスト  
        tableView.reloadData()  
    }else{ //入力がない場合  
        searchedArray1.removeAll();searchedArray2.removeAll()  
        searchedArray1 = George as NSArray as! [String] //NSMutableArray→Array  
            のキャスト  
        searchedArray2 = always_Say as NSArray as! [String]  
        tableView.reloadData()  
    }  
}
```

-UISearchResultsUpdating-

tableView

```
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) ->
    Int {      //各セクションのセルの数を決めている
    var count = 0
    if searchController.active { //検索中の場合
        switch section{
        case 0: count = searchedArray1.count
            break
        case 1: count = searchedArray2.count
            break
        default:
            break
        }
    }else{
        switch section{
        case 0: count = George.count      //セクションの初めのセル数
            break
        case 1: count = always_Say.count//次セクションのセル数
            break
        default:
            break
        }
    }
    return count
}
```

-UITableViewDataSource-

tableView

```
func tableView(tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {    //tableViewに表示するセルを作つて表示させてい  
    var cell = tableView.dequeueReusableCell(withIdentifier: "Cell") //再利用でき  
    るセルがあれば再利用する  
    if cell == nil{        //セルが再利用できない場合新しくセルを作る  
        cell = UITableViewCell(style: UITableViewCellStyle.Default,  
        reuseIdentifier: "Cell") //style = 書式設定 reuseIdentifier = 再利  
        用のためのセル名の定義  
    }  
    if searchController.isActive{        //検索中の場合  
        switch indexPath.section{        //indexPathにはセクション番号と行番号が入ってます  
        case 0: cell?.textLabel?.text = searchedArray1[indexPath.row]  
            break  
        case 1: cell?.textLabel?.text = searchedArray2[indexPath.row]  
            break  
        default:  
            break  
    }  
}
```

-UITableViewDataSource-

tableView

```
 }else{
    switch indexPath.section{ //indexPathにはセクション番号と行番号が入っています
        case 0: cell?.textLabel?.text = George[indexPath.row] as? String //可
                    变配列はanyObject型なのでStringへアップキャストする
                    break
        case 1: cell?.textLabel?.text = always_Say[indexPath.row] as? String
                    break
        default:
                    break
    }
    return cell!
}
```

-UITableViewDataSource-

今回はここまで

これで検索バーに入力した文字を表示してくれるようになりました。しかし、検索結果を削除したり編集する処理は行っていませんのでアプリがクラッシュするので悪しからず。

実機で動かそう

まずiOSの端末をMacに接続します。
シミュレーターの種類を接続している端末にします。
ここでxcodeのverと端末のverが合わないよってエ
ラーがきっと出るのでdeployment Infoの
deployment targetの値を自分の端末のOSのverと
一致させれば終了

実機で動かそう

あれ? 実機でのレイアウトが崩れてる!?

実機で動かそう

シミュレータ上と実機で動かした際の画面のサイズ等
は一致しません(例 シミュレータはiPhone5sサイズ
実機はiPhone6)

実機で動かした際のレイアウトの決まりごと(auto
layout)を設定しましょう!