FinTech-FUN iOS勉強会第3回 2017/06/12資料

前回はボタンを作って画面遷移，画面遷移で値を渡すというようにiOSアプリケーションに最も基本な部分を学んだ．前回はlabelでの値渡しやbuttonの操作だけであったが，Text FieldやPickerなど様々な機能の使い方を自ら学習してもらえるとありがたいです．
※今回からプログラムはかなり省いていて，機能を説明する重要な部分だけしか書いてないです．前回までのswiftファイルに追加したり，どうしたらうまくいくかを自分で考えたり，実際にプログラムを書いてみたりしてみてください．わからなくなったら，いつでも遠慮なく質問してね． 参考：
http://qiita.com/merrill/items/b3a57acf38afdd3023f0

配列 Arrayの使い方

- CやJavaと違って配列は常に動的な配列を使うことができる．
  - 動的な配列：配列の大きさを変えられる

```
var ar: [Int] = []     //空の配列を宣言
//var ar = [1, 2]         //配列をInt型の1,2で初期化
ar.append(1)           //要素の追加(Int型の1を追加)
ar.append(2)
print(ar[0])           //arの0番目を使う 1
print(ar)              //arの中身を使う [1, 2]
ar.remove(at: 0)       //配列要素の0番目を削除
```

- 配列を空で宣言するには，型を指定しないといけない．
- 初期化する時は，型推論が起こるので型指定は必要ない．
- 配列に要素を追加するには append を使う
- 配列から要素を削除するには remove を使う
  - (at: Int) で配列の要素番号を指定

for文の使い方

- for文は拡張for文の使い方がされる．

```
/*配列の中身を表示*/
for i in ar{
  print(i)
}

/*配列の中身をインデックスと表示*/
for (i, value) in ar.enumerated(){
  print("\(i) : \(value)")
}
```

- arの中身をiに代入しながらループする．
- swiftではfor文は配列の中身を操作する時に使うほうがいいかも　→　while文のほうがいい場合もある(無限ループさせるにはwhileが必要) whileの使い方は自分で調べてね．
- 以下の使い方で指定回数ループすることができる

```
for i in 0..<4 {
  print(i)
}

for i in 0...3{
```

```
  print(i)
}
```

Dictionary / JSON

- JSONとはデータを文字列で表現することができ，要素の取り出し/追加などが非常に便利である
- Swift3ではJSONをDictionaryを用いて表現する
- JSONはKeyとValueを結びつけて扱う

```
//let jsonObj = ["Name":"Taro"]       //静的な要素の初期化
var jsonObj = Dictionary<String,Any>()
        jsonObj["Name"] = "Taro"
        jsonObj["Age"] = 12
        //jsonObj["Name"] = nil
        //jsonObj.removeValue(forKey: "Name")
do {
    let jsonData = try JSONSerialization.data(withJSONObject: jsonObj, options: [])
    let jsonStr = String(bytes: jsonData, encoding: .utf8)!
    print(jsonStr)
} catch let error {
    print(error)
}
```

- JSONは動的に操作することが多いので，Dictionary型を用いるのが一般的
- Dictionary() JSONの形式にするためにStringとAnyを使う
  - Keyは必ずString，Valueは様々な型が入るのでAnyとなっている
- Dictionary型の追加は jsonObj["Name"] = "Taro" のようにkeyとvalueを設定することで追加できる
- nilを代入するかremoveValueでKeyを設定すれば，Dictionaryからその値を削除できるs
- JSONSerializationでJSONを作り，それをStringにすることでJSON文字列を作ることができる
  - JSONSerializationはエラー処理をしないといけないのでdo try catchが必要
- 参考 : http://dev.classmethod.jp/smartphone/swift3-json/
- 端末(ローカル)に保存したデータを再度扱うことはアプリケーション開発ではよくあること.
- その際にはJSON文字列で保存し，JSON文字列を読み込みDictionary型に変換するという手順でそれを実現する

```
var jsonObj = Dictionary<String,Any>()
jsonObj["Name"] = "Taro"
jsonObj["Age"] = 12
var jsonStrCon: String = ""
do {
    let jsonData = try JSONSerialization.data(withJSONObject: jsonObj, options: [])
    let jsonStr = String(bytes: jsonData, encoding: .utf8)!
    jsonStrCon = jsonStr
} catch let error {
    print(error)
}

let data = jsonStrCon.data(using: .utf8)!
do{
    let dic = try JSONSerialization.jsonObject(with: data, options: []) as? [String: Any]
    print(dic!)
    var Obj = dic!
    Obj["Pro"] = "Oh"
    print(Obj)
} catch let error {
    print(error)
```

```
}
```

- 上のプログラムの下部 jsonStrCon.data より下側でJSON文字列をDictionary型に変換して，KeyとValueを追加している
- JSONSerialization は let型 での初期化しかできない
- var型として変数で使いたのであれば，letとvarの2種類を併用してvarにletを代入する必要がある
- また，何度か '!' が付いている箇所があるが，nil(null)を許容するかしないかというOptionalの制御である
  - この '!' はOptionalをアンラップするもので，Optionalの中身を取り出す処理を行なっている
  - 上記プログラムのdicの実体はOptional(dic)というようになっており，dicをそのままprintしても Optional(dicの中身) という表示になってしまう．これをdic!にすればOptionalが外れて dicの中身 だけを表示することができる

簡単にJSON文字列とDictionaryの変換処理を表す図

データをローカルに保存する(キャッシュ)UserDefaults

- 通常，アプリケーションは一度終了するとデータが破棄され，次の起動時には初回起動と同じ状態に戻る．
- 何らかのデータを保存したい場合があるので，それを実現するための機能がキャッシュである．
- キャッシュはテキストファイルとは異なるので注意すること．
- 参考：http://dev.classmethod.jp/smartphone/swift-3-0-userdefaults/

```
var count = 0
let userDefaults = UserDefaults.standard      //ローカルに保存するためのインスタンスを生成
userDefaults.set(count, forKey: "count")      //インスタンスuserDefaultsにcountをセット
let ret = userDefaults.integer(forKey: "count")         //インスタンスuserDefaultsの"count"に保存されているintを取得
```

- UserDefaults.standard を使ってインスタンスを生成する
- .set(count, forKey: "count") は第1引数のcountを型推論でintと判断して，Keyである"count"と紐づけて保存する
- 取り出すには，.integer(forKey: _)を使ってintを取り出す．他にも，.string()や.bool()などを使うことで各種の型でデータを取り出すことができる．
- .objectを使うと様々な型のデータを取り出すことができる．
  - ただし，取り出した後の処理が面倒になるので，色々理解を深めておくといいかも．

JSONとUserDefaultsを組み合わせて利用することで，アプリケーション開発をより順調に進めることができる．