第22回

Storyboardで開発効率アップ!

スマートフォンの認知度を一般に広めただけでなく、ソフトウェア開発においても新しい波を作り出してしまったiOS。開発者たちは何を見、どう考えているのか。毎回入れ替わりでiOS向けアプリケーション開発に関わるエンジニアに登場いただき、企画・開発のノウハウやアプリの使いこなし術などを披露してもらいます。

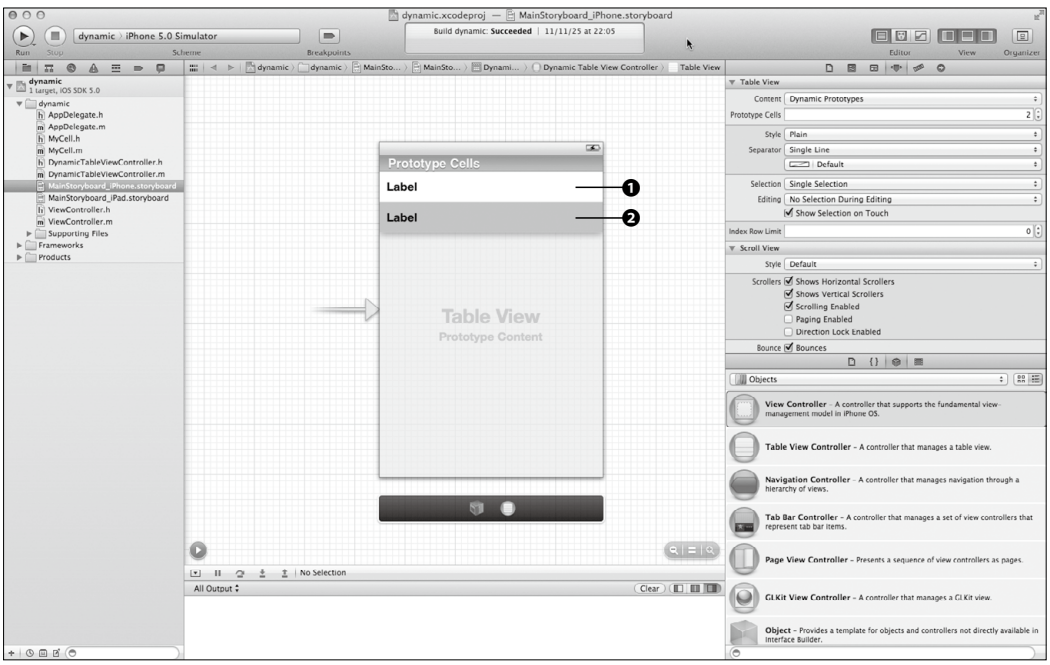
吉田 悠一　YOSHIDA Yuichi　㈱デンソーアイティーラボラトリ
http://sonson.jp/

はじめに

iPhone 4Sの発売と同時に4度目のメジャーバージョンアップとなる、iOS 5がリリースされました。iCloud、Twitter連携、ワイヤレス同期など、多くの機能がiOSに追加されました。これまでiOSのメジャーバージョンアップでは、ユーザが使う機能やApple謹製のアプリケーションに加えて、開発者にも多くの新機能やAPIが提供されてきました。今回のバージョンアップも例外ではなく、開発者向けに多くの機能が追加されています。

本稿で紹介するのは、その中でも開発者の生産性に最も貢献すると考えるXcodeのInterface Builder(Interface BuilderはXcodeに組み込まれたので、これ以降はXcodeに統一します)の新機能Storyboardです。Storyboard機

▼図1　XcodeでUITableViewを編集している状態



能が追加されたことによって、Xcodeは、ビューコントローラやビューといった部品に分けて開発するのではなく、アプリケーション全体で画面遷移を含めてプログラミングできる強力なツールとなりました。

本稿では、Storyboardによる恩恵が最も大きいUITableViewのコーディングについて説明します(**図1**)。また本稿は、XcodeやInterface Builderの基本的な使い方を知っているプログラマを対象とします。なお、本稿で前提とするXcodeのバージョンは4.2.1です。

UITableViewの開発効率の向上

新しいXcodeを使った開発の最も大きな利点は、UITableViewにおける開発効率の向上にあると考えます。今回紹介するのは、次の2点です。

- iOSの設定アプリのような複雑なUITableViewを作るときのコード量が劇的に減る
- オリジナルのセルをビューコントローラ上で直接設計できる

複雑なセルを持つUITableViewの作成

従来のXcodeによる実装

従来のXcodeでは、iOSの設定アプリのように、スイッチやスライダーや単純なボタンなどたくさんの種類のセルが並ぶ場合、セルのセクションと行をif文やswitch文により分けて、それぞれに合ったセルを生成して、UITableViewに渡すコードを延々と書かなければなりませんでした。

今からサンプルを作成しますが、このサンプルを従来のXcodeで実装した場合のコードの一部を**リスト1**に示します。これは、冗長極まるコードです。このため、このコードをXMLから自動的に生成するコードを書くプログラマもいる始末です。さらに、スイッチなどのコントロールを持つセルからユーザの操作イベントを取得しようとすると、そのためにUITableViewCellのサブクラスを作成して、イベント処理のコードを書いて……と、従来のSDKではここでもコードが肥大化していきます。

▼リスト1　従来のXcodeにより実装した場合のコードの例

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    if (indexPath.section == 0) {
        if (indexPath.row == 0) {
            // スイッチを含むセルを生成、
            // あるいはキューから取り出す
            // セルのタイトルなどを設定
            return cell;
        }
        if (indexPath.row == 1) {
            // スライダを持つセルを生成、
            // あるいはキューから取り出す
            // セルのタイトルなどを設定
            return cell;
        }
        if (indexPath.row == 2) {
            // ラベルを含むセルを生成する
            // すでに生成している場合は生成しない
            return cell;
        }
        if (indexPath.row == 3) {
            // ラベルを含むセルを生成する
            // すでに生成している場合は生成しない
            return cell;
        }
    }
    else if (indexPath.section == 1) {
        // 次のセクション……
    }
    else if (indexPath.section == 2) {
        // 次のセクション……

    }
    return cell;
}
```

新しいXcodeによる実装

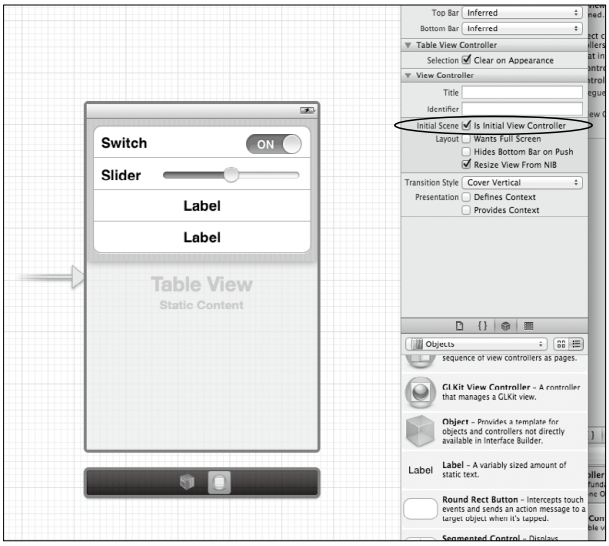
新しいXcodeを使うと、同様の機能をコードをほぼ書かずに実現できます。Xcodeのシンプルビューのサンプルプロジェクトを作成してStoryboardファイルを開き、既存のViewControllerを削除してTableViewControllerを追加し、さらにそれを初期状態のビューコントロー

ラに設定します(**図2**の囲み)。追加したTable ViewControllerは、MyTableViewControllerというUITableViewControllerのサブクラスにします。Storyboardファイルで、MyTableView Controllerに含まれるUITableViewを選択し、Contentを Static Cellsにセットします。こうすると、MyTableViewControllerはセルを動的に生成するのではなく、Xcode上で設計したセルを使うようになります。

たとえば、図2に示す画面デザインのように、ボタンなどを配置したセルを作ってみましょう。1つめのセルにはUITextLabelとUISwitch、2つめのセルにはUITextLabelとUISlider、3つめと4つめのセルにはUITextLabelを追加します。そして、MyTableViewControllerは、UISwitchとUISliderに対するユーザの操作結果を、3つめと4つめのセルのラベルに表示するように実装します(**リスト2**)。また、UISwitchとUISliderはそれぞれchangedSwitchとchangedSliderに、2つのラベルはそれぞれlabelOn3rdRowとlabelOn4thRowに、Xcode上で接続しておきます。

UITableViewを Static Cellsにしたときは、UITableViewDelegateや UITable

▼図2　Xcode(Interface Builder)でのアウトレットの接続



▼リスト2　MyTableViewControllerの実装

```
@interface MyTableViewController : UITableViewController

@property (strong, nonatomic) IBOutlet UILabel *labelOn3rdRow;
@property (strong, nonatomic) IBOutlet UILabel *labelOn4thRow;

- (IBAction)changedSwitch:(id)sender;
- (IBAction)changedSlider:(id)sender;

@end

@implementation MyTableViewController

@synthesize labelOn3rdRow = _labelOn3rdRow;
@synthesize labelOn4thRow = _labelOn4thRow;

- (IBAction)changedSwitch:(id)sender {
    if (((UISwitch*)sender).on)
        self.labelOn3rdRow.text = [NSString stringWithFormat:@"On"];
    else
        self.labelOn3rdRow.text = [NSString stringWithFormat:@"Off"];
}

- (IBAction)changedSlider:(id)sender {
     self.labelOn4thRow.text =
        [NSString stringWithFormat:@"%f", ((UISlider*)sender).value];
}
@end
```

ViewDatasourceで宣言されているセルやセクションの数を返すメソッド、セルのインスタンスを返すメソッドを実装しないほうが良いでしょう。Xcode上で設定した値と矛盾すると、アプリケーションがクラッシュしたり、想定外の動作をするなどが起こりうるので注意が必要です。

▼リスト3　MyCellの実装

```
@interface MyCell : UITableViewCell
@property (strong, nonatomic) IBOutlet UILabel *label;
@end

@implementation MyCell
@synthesize label = _label;
@end
```

このサンプルで、スイッチやスライダーを操作すると、3つめと4つめのセルのテキストが更新されるはずです。ほとんどコードを書くことなく、複雑なUITableViewを使ったインターフェースをXcodeで実装することができました。MyTableViewControllerクラスに設定を反映したり、保存したりするコードを実装すれば、iOSの設定アプリのような複雑な画面を簡単に作成できることがわかると思います。

ここまでは、XcodeのStatic Cellsというモードで、簡単に複雑なUITableViewを構築する方法を紹介しました。続いて、DynamicPrototypesというモードで、UITableViewのコーディングを行う方法を紹介します。

UITableViewCellのビジュアルプログラミング

Dynamic Prototypesの利用

UITableViewのContentには、Static Cells以外に、Dynamic Prototypesがあります。Static Cellsは、誤解を恐れずに言うと、UITableViewで見せたいセルはすべてXcode上で1つずつ作成しなければならないモードです。それに対して、Dynamic Prototypesは、セルを大量に使い回すときのモードです。次に、このモードを使った場合のコーディングスタイルを紹介します。

■UITableViewCellのサブクラスの作成

まず最初にXcodeでインターフェースを設計する前に、UITableViewCellとUITableViewControllerのサブクラスとして、それぞれMyCellとDynamicTableViewControllerをプロジェクトに追加します。次にUITableViewCellのサブクラス、MyCellの実装です(**リスト3**)。とくにコーディングは必要なく、UILabelをアウトレットとして保持できるようにアクセサを定義しておきます。

■ UITableViewControllerのサブクラスの作成

セルを表示するUITableViewControllerのサブクラス、DynamicTableViewControllerのコードについて説明します(**リスト4**)。こちらのコードは、今までと多少異なります。

異なる点として、UITableViewにセルを渡すメソッド、「- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath;」を書き換えます。まず、セルの行数が奇数か偶数かによって分岐させ、奇数の場合はdequeueReusableCellWithIdentifier:cellIdentifierを呼び出すときのIdentifierを“OddCell”に、偶数の場合は“EvenCell”にします。

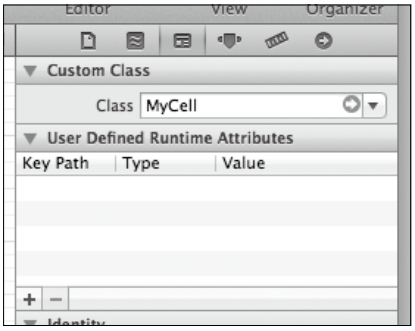
また、通常、dequeueReusableCellWithIdentifier:cellIdentifierがnilを返した場合は自前でセルをallocメソッドで作成しますが、XcodeでContentをDynamic Prototypesにした場合はすべてランタイムがStoryboardの設定に基づいてセルを生成してくれるので、プログラマがallocで割り当てる必要はなくなります。これだけでも、かなり便利なことがわかると思います。

あとは、MyCellの中にあるUILabelのテキストに、セルの行数をNSStringに変換して渡します。

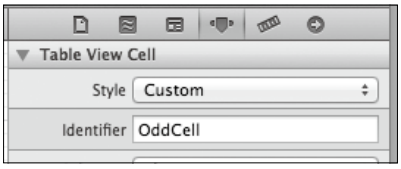
▼リスト4　DynamicTableViewControllerの実装

```
@interface DynamicTableViewController : UITableViewController
@end

@implementation DynamicTableViewController

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    // セクションの数を返す
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section {
    return 40;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    NSString *CellIdentifier = @"OddCell";
    if (indexPath.row %2 == 0)
        CellIdentifier = @"EvenCell";

    MyCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    cell.label.text = [NSString stringWithFormat:@"%d rows", indexPath.row];

    return cell;
}

(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    // 何もしない
}

@end
```

▼図3　セルのクラスをMyCellに設定



▼図4　セルにOddCellというIdentiferを設定



■Storyboardファイルの設定

次に、Storyboardファイルについて説明します。XcodeでStoryboardファイルを開き、DynamicTableViewControllerの中のUITableViewをクリックし、プロパティの中のContentをDynamic Prototypesに設定し、Prototype Cellsの値を2に設定します。そうすると、UITableViewの中にセルが2つ表示されるはずです。これらを両方とも、**図3**のようにクラスをMyCellにしておきます。

次に、この2つのMyCellのIdentifierを設定します。リスト4のコード例で記述した2つのIdentifierです。それぞれのIdentifierには、1つめのセルにOddCell(**図4**)、2つめのセルにEvenCellを設定します。これで、おおかたの準備は整いました。

次に、この2種類のセルをStoryboardで編集します。1つめのセルを奇数行のセルとして、2つめのセルを偶数行のセルとしてデザインします。偶数と奇数をわかりやすくするため、偶数行のセルの背景に色を変えたUIViewを追加し(図1の❶と❷)、両方のセルにタイトルのためのUILabelを追加することにします。そして、それぞれのUILabelを、それぞれのMyCellのアウトレットであるLabelに接続します。また、セルの選択時の見栄えを良くするために、それらのUILabelのハイライト時のテキストの色を白色に変えておきます。

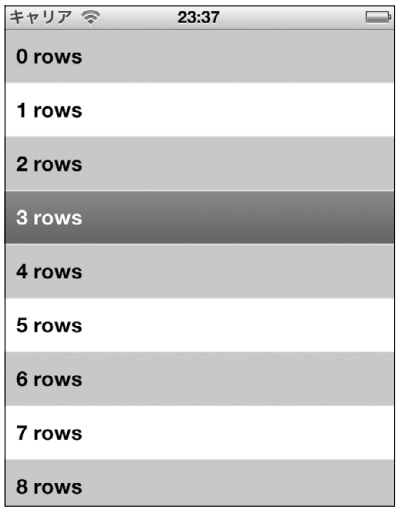
このプロジェクトを実行すると、**図5**のように奇数と偶数でセルの背景色が違う結果が得られるはずです。新しいXcodeによって、まったく同じクラスであるにもかかわらず、見栄えが違うセルを、見栄えに関するコードをいっさい書かずに実装することができました。

従来のXcodeでは、セルとテーブルビューのインターフェースを同時に編集できませんでした。しかし、新しいXcodeは、このようにテーブルとセルを1つのファイルで直感的にビジュアルでプログラミングできるだけでなく、リスト4のようにIdentifierを使った動的なリソースの振り分けも可能です。また、TableViewController側にIBActionを接続できるため、セルに加えたUIButtonのタップアクションのアウトレットとしてUITableViewControllerを接続でき、セルからテーブルにタップアクションを伝えるためのデリゲートをセルに持たせる必要がなくなるといった利点もあります。これらの機能を使いこなすことで、手間を軽減できる点は他にもたくさんあると思います。

まとめ

Storyboard機能が追加された新しいXcodeを使うことで、UITableView周りの開発効率を上げられる例を2点紹介しました。従来、開発コストが高くなりがちであったUITableViewにまつわるコードを改善できる点はかなり有用であると言えます。このStoryboard機能は、他の開発現場でもいろいろと役立つシーンがあると筆者は考えます。まず、UITableView周りからはじめてみて、それから応用にトライして、この利便性を他のViewController構造に活かしてみてはいかがでしょう。SD

▼図5　2つのプロトタイプセルを追加したUITableViewController



● 吉田 悠一（よしだ ゆういち）　㈱デンソーアイティーラボラトリ

本業はコンピュータビジョン、ヒューマンインターフェースの研究。iOS用2ちゃんねるビューア「2tch」の中の人。共著書にiOSのAPIハックを集めた『iOS SDK HACKS』（吉田 悠一、高山 征大、UICoderz著、オライリージャパン、2010年、ISBN978-4-8731-1472-9）がある。