Re:ゼロから始めるアプリ開発

— Xamarin ってなぁに？ —

ここあ　著

2019 年 9 月 22 日　 ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

目次

第1章 Xamarinってなぁに

はい，いきなりサブタイ回収しました．

Xamarin っていうのは，Android なら `Java` もしくは `Kotlin` のコードを，iOS なら `Swift` のコードをラッピングして `C#`を用いて開発することのできるフレームワークです．

こいつを使うことで，各 OS 間でのロジックに関するコードを共通化することができるってわけなんですね．

さらにさらに，(ちょいと制限があるけど)UI すらも共通化することも可能です．

とりあえず実際にアプリをいくつか作っていきながら Xamarin での開発に慣れていきましょうね．

そうそう，今回は Android アプリに関してのみ触れたいと思います．

そうそう，本著では Xamarin の導入方法とかプロジェクトの作成方法は解説しないことにします．

めんどくさいんだもん．

第2章 キッチンタイマー

まず手始めに定番とも言えるキッチンタイマーを作ってみましょう.
キッチンタイマーにはどのような機能が必要でしょうか.

- タイマーをスタートする
- 終了時に知らせる

本当に最低限の要求仕様はこのくらいではないでしょうか.
てことで開発に移っていきましょう！

2.1 レイアウトを実装する

まずは，アプリのレイアウトを決めましょう．プロジェクトを作成した時に activity_main.xml というファイルが作成されると思うので，そちらをいじっていきましょう.
ここはシンプルに TextView と Button でいこうではありませんか.

▼activity_main.xml

```
 1: <?xml version="1.0" encoding="utf-8"?>
 2: <RelativeLayout
 3:  xmlns:android="http://schemas.android.com/apk/res/android"
 4:  xmlns:app="http://schemas.android.com/apk/res-auto"
 5:  xmlns:tools="http://schemas.android.com/tools"
 6:  android:layout_width="match_parent"
 7:  android:layout_height="match_parent">
 8:  <TextView
 9:   android:layout_width="wrap_content"
10:   android:layout_height="wrap_content"
11:   android:id="@+id/remains"
12:   android:textAppearance="?android:attr/textAppearanceLarge"
13:   android:textSize="80sp"
14:   android:text="03:00"
15:   android:layout_centerInParent="true"
16:   />
17:  <Button
18:   android:layout_width="wrap_content"
19:   android:layout_height="wrap_content"
20:   android:id="@+id/start"
21:   android:text="start"
22:   android:layout_below="@id/remains"
23:   android:layout_centerHorizontal="true"/>
24: </RelativeLayout>
```

2.2　プログラムを書いていこう

最低限キッチンタイマー的な動作をするプログラムを書いて見ましょう..
先ほどの要件定義を満たすために，カウントダウン機構とスタート動作，ビープ音の生成・発音機能を実装していくことにします.

▼MainActivity.cs

```
 1: using Android.App;
 2: using Android.Media;
 3: using Android.OS;
 4: using Android.Support.V7.App;
 5: using Android.Widget;
 6:
 7: namespace KitchenTimer {
 8:  [Activity(Label = "@string/app_name", Theme = "@style/AppTheme.NoActionBar"
    ), MainLauncher = true)]
 9:  public class MainActivity : AppCompatActivity {
10:
11:   // 残り秒数
12:   private int sec_ = 180;
13:
14:   Handler handler_;
15:   TextView tv_;
16:   AudioTrack audio_;
17:
18:   // Beep音用の変数群
19:   const double amplification_ = 0.4;
20:   const int sampleRate_ = 44100; // [samples / sec]
21:   const short bitRate_ = 16; // [bits / sec]
22:   const short freq_ = 440; // [Hz] = [1 / sec]
23:   const double duration_ = 0.5; // [sec]
24:   short[] audioBuf_;
25:
26:   protected override void OnCreate(Bundle savedInstanceState) {
27:    base.OnCreate(savedInstanceState);
28:    Xamarin.Essentials.Platform.Init(this, savedInstanceState);
29:    SetContentView(Resource.Layout.activity_main);
30:
31:    handler_ = new Handler();
32:
33:    // (1)
34:    // 時間を表示させるviewの取得
35:    tv_ = FindViewById<TextView>(Resource.Id.remains);
36:
37:    // (2)
38:    // Buttonのクリック動作を設定
39:    // どうせ保持していても使わないので直接構築
40:    // 1000ミリ秒経過後にタスクを実行するように設定しているぞ
41:    FindViewById<Button>(Resource.Id.start)
42:     .Click += (sender, e) => {
43:      handler_.PostDelayed(() => Action(), 1000);
```

```
44:     };
45:   }
46:
47:   /// @brief : Resume時に呼び出されるhook
48:   /// @return : None
49:   protected override void OnResume() {
50:    base.OnResume();
51:
52:    // [samples / sec] * [sec] = [samples]
53:    int samples = (int)(sampleRate_ * duration_);
54:    audioBuf_ = new short[samples];
55:
56:    // (3)
57:    // Beep音の生成
58:    for(int point = 0;point < samples;point++) {
59:     // pointの最大値はsamplesと同値．すなわち発音時間でのsample数
60:     // すなわち，point / sampleRate_は時間位置(time / freq)と等価的存在
61:     audioBuf_[point] = (short)((amplification_ * short.MaxValue) *
62:      System.Math.Sin(2.0 * System.Math.PI * freq_ * point / sampleRate_));
63:    }
64:
65:    audio_ = new AudioTrack(Stream.Music,
66:                sampleRate_,
67:                ChannelOut.Mono,
68:                Encoding.Pcm16bit,
69:                audioBuf_.Length * bitRate_ / 8,
70:                AudioTrackMode.Static);
71:    audio_.Write(audioBuf_, 0, audioBuf_.Length);
72:   }
73:
74:
75:   // (4)
76:   /// @brief : 一定時間ごとに行うタスク
77:   /// @return : None
78:   void Action() {
79:    handler_.RemoveCallbacks(Action);
80:    sec_--;
81:
82:    if(sec_ > 0)
83:     handler_.PostDelayed(Action, 1000);
84:    else
85:     Beep();
86:
87:    // (5)
88:    // この関数が別スレッドで動いているので，
89:    // UIスレッドを明示的に指定
90:    RunOnUiThread(() => {
91:     tv_.Text = (sec_ / 60).ToString("D2") +
92:     ":" +
93:     (sec_ % 60).ToString("D2");
94:    });
95:
96:   }
```

```
 97:
 98:   ///  @brief  : Beep音を鳴らす
 99:   ///  @return : None
100:   void Beep() {
101:    audio_.Stop();
102:    audio_.ReloadStaticData();
103:    audio_.Play();
104:   }
105:  }
106: }
```

(1), **(2)** ではそれぞれのコントロールを取得してます.

両者を比べてみれば一目瞭然ですが，**(1)** ではインスタンスを生成しているのに対して **(2)** では，生成せずに直接操作をしてます.

別に，以降もそのコントロールを使用するなら保存しておけばいいですし，その場限りでしか操作しないなら直接構築してあげればいいかなと思います.

そしてそして，**Xamarin では，コントロールのアクションはラムダ式で登録できる** のです！！！

(3) では，カウント終了時に鳴らす音声を生成しています.

思いっきり高校物理の波動の分野ですね．覚えてますか？？ 笑

(4) の `Action` という関数は，ボタンクリック時に 1000[ms] 遅延で実行するタスク，つまりタイマーカウントを担っています，遅延動作をネストしたかったのでわざわざ関数化しました.

(5) の部分は，タイマーの残り時間をフォアグラウンドで実行するためのコードです. `RunOnUiThread()` を用いる事で，UI の更新をスレッドセーフ (?) に行うことができるのです.

さて，これだけで本当に簡単なキッチンタイマーができてしまいました.

しかしながらこれでは，スタートしてカウント終了したらそれっきりです．これでは使い物になりませんよね.

て事で，こいつに機能をじゃんじゃん追加していって本格的なキッチンタイマーを作っていくこととしましょう.

2.3　機能を追加する

- **♣ リセット機能**
- **♣ 任意の時間を指定**
- **♣ 一時停止機能**
- **♣ 最近使用したタイマー**
- **♣ UI をカッコよくする**

Re:ゼロから始めるアプリ開発

Xamarin ってなぁに？

2019 年 9 月 22 日　ver 1.0

著　者　ここあ
印刷所　○○印刷所

© 2019 カウプラン機関極東支部

(powered by Re:VIEW Starter)