非同期プログラミング（C#）

2013.7.5

ニューラル株式会社

目的

Windowsストアアプリの開発環境（WinRT）では、多くのAPIを非同期化しています。I/O以外でもウインドウ表示など重い処理は非同期化されています。今回は.NET上の非同期プログラミングの手法を取り上げたいと思います。

※ Windowsストアアプリ: Windows8/WindowsRT(ARM向け)のMetroUIを使用したアプリケーション。win32APIの利用は制限される。

非同期プログラミング（C言語, ioctl,select）

シングルスレッドで動作する。ポーリングで並行処理を行う。

```
ioctl(fd, FIONBIO, &val);     // fdをNon-Blockに変更
while (1) {
  n = read(fd, buf, sizeof(buf));
  if (n >= 1) {
    …                                  // Read処理
  } else if (errno == EAGAIN) {
    …                                  // Read以外の他の処理
  } else {
    … // エラー
  }
}
```

非同期プログラミング（C#, APM）

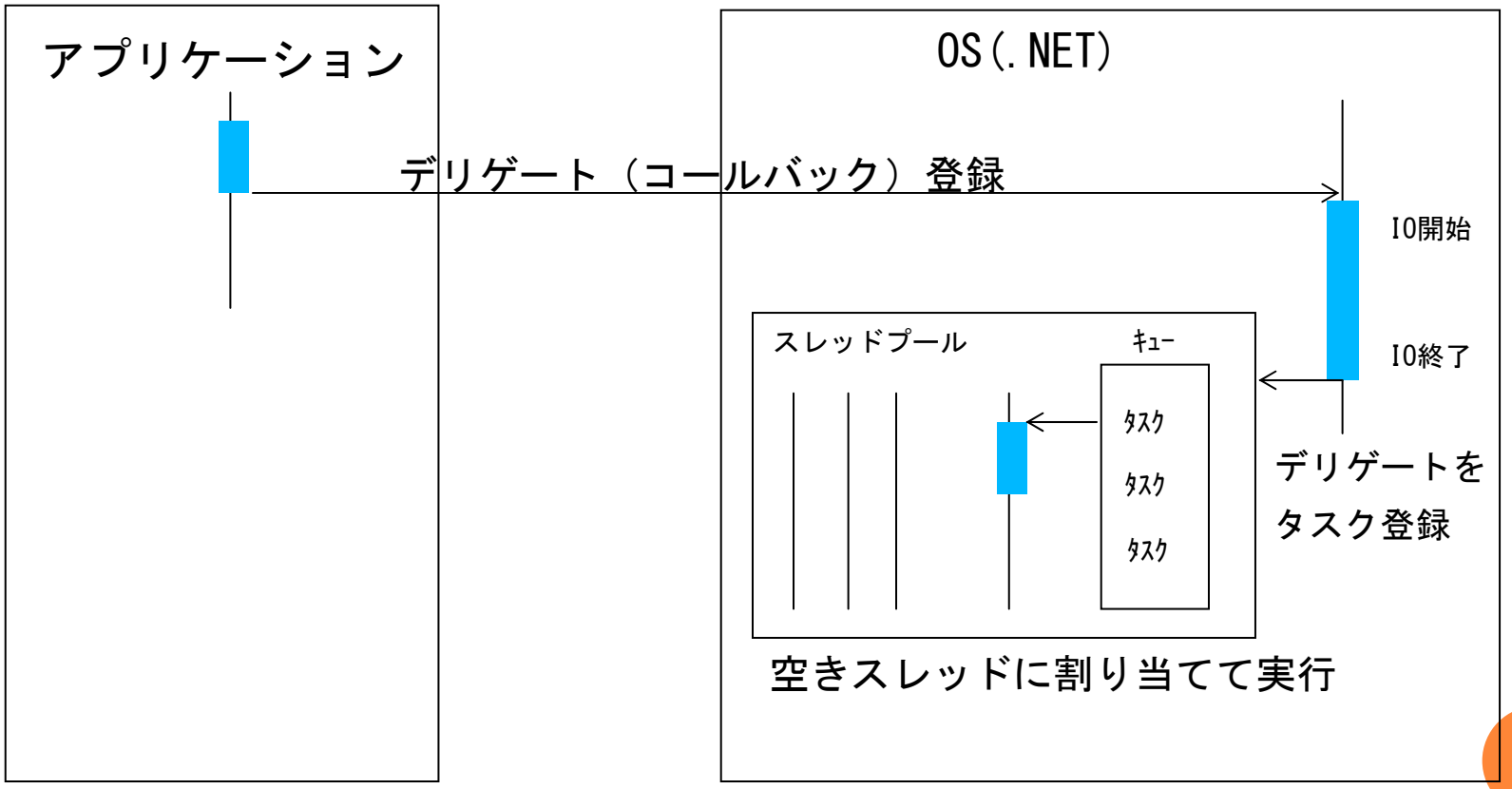
Begin～関数、End～関数を使って非同期IO処理ができる。
APM(Asynchronous Programming Model)と呼ばれる。

```
static void Main()
{
  ios = …    // iostreamオブジェクトを取得
  ios.BeginRead(buf, 0, buf.Length, ReadFunc, null);  // 非同期読込開始
  …          // カレントスレッドは別の処理を実行
}

void ReadFunc(IAsyncResult ar)    // 読込完了後に呼ばれるデリゲート(Callback)
{
  int n = ios.EndRead(ar);        // EndReadは読込サイズを返す
  …                               // 読込データの処理
  ios.BeginRead(buf, 0, buf.Length, ReadFunc, null);  // 継続データがあれば
                                                      // 再度非同期読込
}
```

非同期処理の実行スレッド

デリゲートの実行スレッドは呼び出しスレッドと異なる。
スレッドプールにキューイングされてから実行される。



非同期プログラミング（C#, EAP）

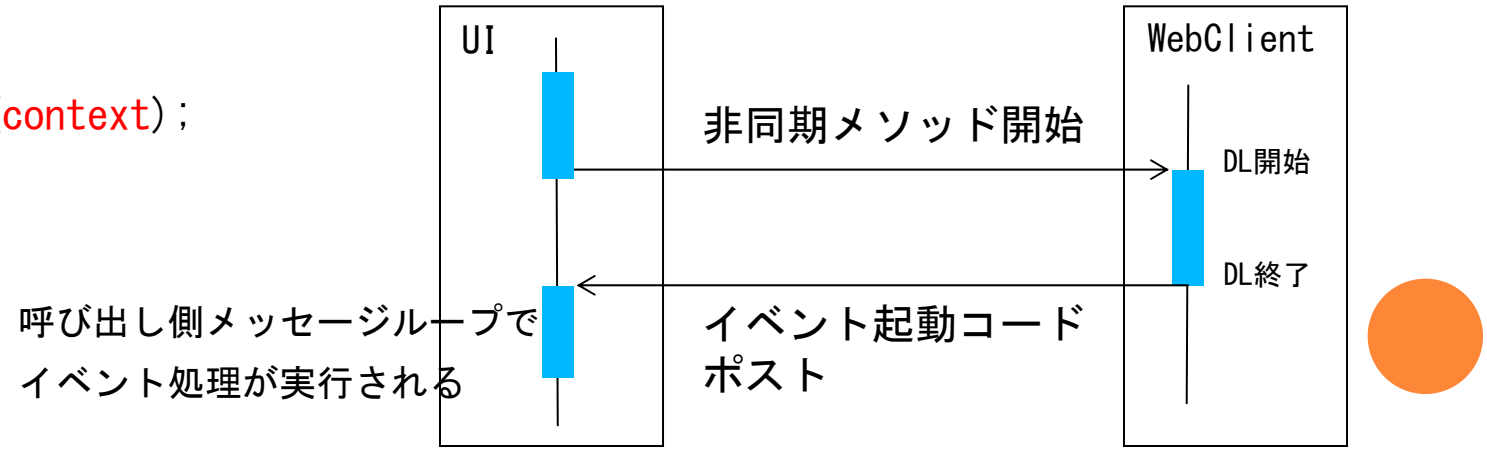
～Async関数、～Completedイベントを使って非同期処理を行う。
EAP(Event-based Asynchronous Pattern)と呼ばれる。

```
void Button_Click(…)
{
  var client = New WebClient();
  // ダウンロード完了イベントにデリゲートを登録
  client.DownloadStringCompleted += (sender, e) => {
      textbox1.Text = e.Result.ToString();   // UI更新処理
  };
  client.DownloadStringAsync(new Uri(…));   // 非同期ﾀﾞｳﾝﾛｰﾄﾞ開始
  …   // カレントスレッドは別の処理を実行
}
```

EAPのUI同期方法

EAPでは同期コンテキストを用いて、完了イベントの処理を呼び出し側スレッドで実行していると思われる。

```
void DownLoadStringAsync(…)
{
  var context = SynchronizationContext.Current; // 呼び出し側ｽﾚｯﾄﾞの同期ｺﾝﾃｷｽﾄ
  Thread t = new Thread((ctx) => {              // ﾀﾞｳﾝﾛｰﾄﾞ処理を行うｽﾚｯﾄﾞ
    …
    ctx.Post(() => {     // イベント起動コード(のデリゲート)を同期CTXに投げる
      this.DownLoadStringCompleted(…);
    },null);
  });
  t.Start(context);
}
```



非同期プログラミング(C#,TAP)

タスク並列ライブラリ TPL(Task Parallel Library)を使用した
TAP(Task-based Asynchronous Pattern)プログラミング。

```
static void Main(string[] args)
{
  Task.Factory.StartNew(Go,  “One”);       // Go に”One”を渡してｽﾚｯﾄﾞ開始
  Task.Factory.StartNew(Go,  “Two”);
  Task.Factory.StartNew(() => Go(“Three”));  // ラムダ式も使える
  …
}

static void Go(Object name)
{
  for (int i = 0; i < 100; i++) {
  {
    Console.WriteLine(“{0} : {1}”, name, i);
  }
}
```

非同期プログラミング(C#,TPL)

TPLはスレッド操作が高機能で使いやすい。

待ち合わせ

```
Task t1 = Task.Factory.StartNew(…);
Task t2 = Task.Factory.StartNew(…);
Task.WaitAll(t1, t2);
```

親子関係

```
Task.Factory.StartNew(() => { // 親タスク
  Task.Factory.StartNew(…, TasckCreateionOption.AttachedToParent); // 子1
  Task.Factory.StartNew(…, TasckCreateionOption.AttachedToParent); // 子2
});
```

継続

```
Task.Factory.StartNew(…).ContinueWith(…);
```

UIディスパッチ

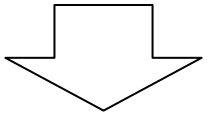
```
void OnButtonClick(…) {
  var uiSched = TaskScheduler.FromCurrentSynchronizationContext();
  Task.Factory.StartNew(…).ConutinueWith(…, uiSched);
}  // ContinueWithで実行する処理はUIスレッドで実行する
```

非同期プログラミング(C#,async,await)

async修飾子, await演算子を使うと、非同期処理間の同期を簡単に記述できる。

（例）ボタンクリックで、AsyncProcを非同期実行した後、
UpdateUIProcをUIスレッドで実行したい

```
void OnButtonClick(…) {
  var uiSched = TaskScheduler.FromCurrentSynchronizationContext();
  Task.Factory.StartNew(AsyncProc).ConutinueWith(UpdateUIProc, uiSched);
}
```



async/awaitを使って書く

```
async void OnButtonClick(…) {
  await Task.Factory.StartNew(AsyncProc);
  UpdateUIFunc();     // このパスが実行されるのはawaitの処理が終わってから。
}
```

今回扱わなかったが関連するテーマ

- C#同期メカニズム
  - lock構文
  - Monitorクラス
  - SpinLockクラス
  - Mutexクラス
- マルチコアCPUとの関連