イベントドリブンプログラムの関数型的書き方

2012年9月1日　　小笠原　啓

IT Planning, Inc.

アジェンダ

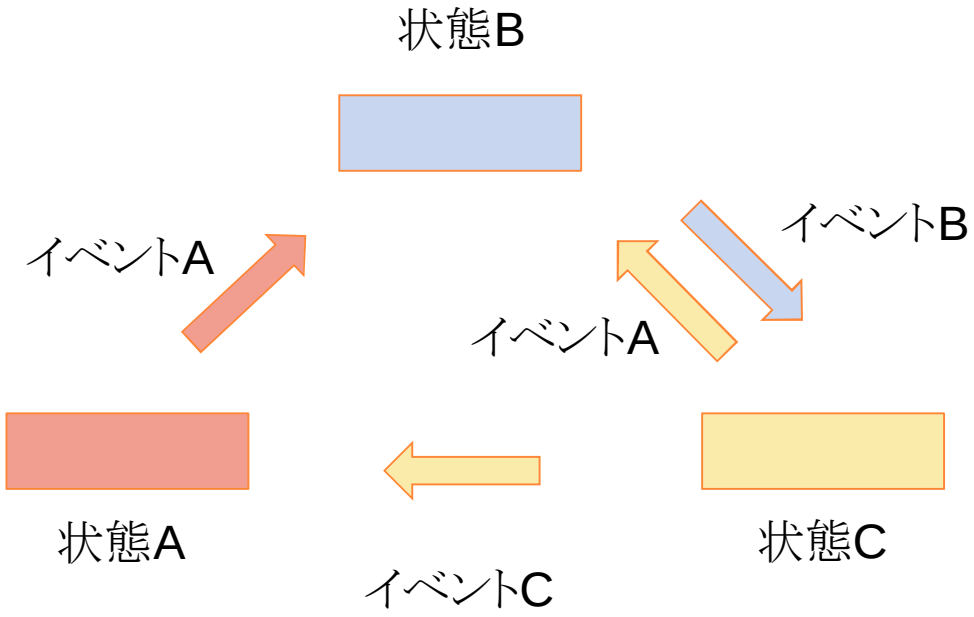
- 状態を減らそう！
- 意味イベントを構築しよう！

第1章
状態を減らそう！

IT Planning, Inc.

イベントドリブンなプログラムとは

- イベントが発生した事を**発端**に、何か処理が走る。この繰り返しで大部分が構成されるようなプログラム。
- イベント処理では現在の状態を確認し、次の状態を作り出す。これは**状態遷移プログラミング**。



例えばこんな例題。

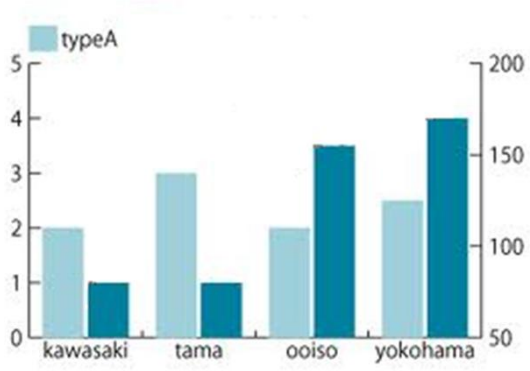
- 棒グラフボタン、折れ線グラフボタン、散布図ボタンがあり、ボタンを押すと排他的に対応するグラフが表示される。
- 重ね描画モードがONの時は、三つのボタンは独立したトグルボタンになる。

重ね描画モード

棒グラフ

折れ線

散布図

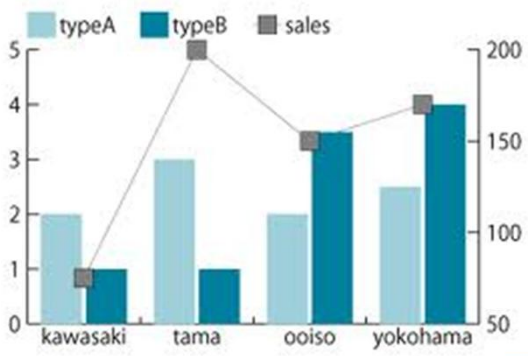


重ね描画モード

棒グラフ

折れ線

散布図



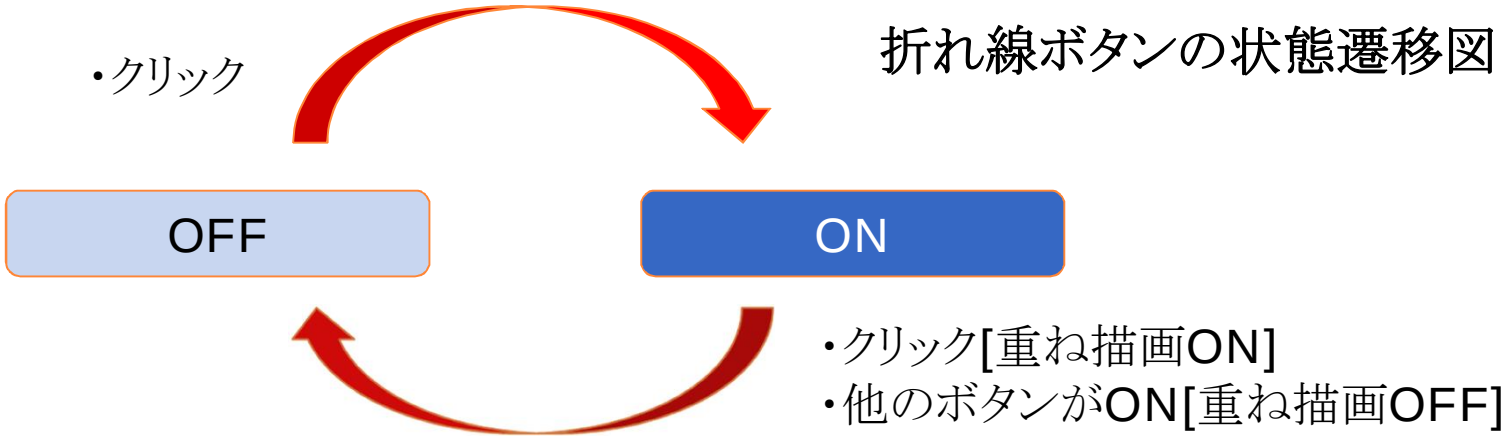
状態を定義してみる。

【関数型言語(OCaml)】

```
(* 全部で4つのON/OFF状態を定義。初期値も付与. *)
let overwrite_mode = ref false
let bar_graph = ref true
let line_graph = ref false
let scatter_diagram = ref false
```

状態遷移図を記述。(折れ線ボタン)

- 重ね描画モードがONの時、
  - 既に折れ線ボタンがONなら、これをOFFに。
  - 折れ線ボタンがOFFなら、これをONに。
- 重ね描画モードがOFFの時、
  - 既に折れ線ボタンがONなら、何もしない。
  - 折れ線ボタンがOFFなら、これをONに。他のONのボタンをOFFに。



折れ線ボタンの状態遷移図

コーディングすると、こんな感じ。

```
let on_bargraph_click () =  (*  折れ線クリック *)
  if  !overwrite_mode then
      bar_graph := not !bar_graph (*  反転 *)
  else
      if  !bar_graph then ()
      else begin
          bar_graph := true;
          line_graph := false;          (*  排他的動作 *)
          scatter_diagram := false;  (*  排他的動作 *)
      end
```

ＯＫ、何か問題が？

問題点

- ボタンは3つ。折れ線ボタン以外のもう2つのボタンもほぼ同じロジックなのに、コピペする？同様に、4つ目のボタンが増えた時、全てのボタンのロジックを変更する？
- overwrite_modeがOFFの時は、ボタンのどれか一つがONである事を覚えておけばいいのに、全てのボタンのON/OFFを管理している。冗長な上にミスの元じゃない？

代数的データ型で状態を見直してみよう。

```
type button = Bar | Line | Scatter
type state =
    OverWrite of button list   (* 重ね描画ON *)
  | Normal of button           (* 重ね描画OFF *)
```

- 重ね描画モードONの時には、ONになっているボタンのリストを保持する。
- 重ね描画モードOFFの時には、3つのボタンの内どれか一つが必ずON。

管理すべき状態が4つから2つに半減！
（しかも同時には1つの状態のみ）

ボタンを一般化した状態遷移。

```
let on_button_click b = function
  | OverWrite bs when List.exists ((=) b) bs ->
        OverWrite (List.filter ((<>) b) bs)
  | OverWrite bs ->  OverWrite (b :: bs)
  | Normal _ -> Normal b  (* 排他的ON *)
```

- 一般化したのに、たったの4行。
- パターンマッチによる見通しのいい場合分け。
- 網羅性チェックも働く。

代数的データ型のご利益

- 代数的データ型は管理すべき状態を減らすことができる。
- イベントドリブンプログラム（状態遷移系）は、代数的データ型の大活躍できるフィールド。

第2章
意味イベントを構築しよう！

IT Planning, Inc.

データのポーリングがしたい

- 数秒に一回、サーバーに新しいデータが無いかどうか確認する。(ポーリング)
- ただし、ユーザーが「更新ボタン」を押すと、即座に問い合わせを行う。

数秒に一度のタイマーによるチェック

サーバーに最新データの
問い合わせ

サーバー

ユーザーによる強制更新

普通に書くとこんな感じ。

```
let do_polling () =  (*  共通イベント処理 *)
  if !locking then () (*  二重問い合わせを防ぐ *)
  else begin
      locking := true;
      polling ~post:(fun () -> locking := false);
  end
(*  生イベントとの接続 *)
let on_timer () = do_polling ()
let on_reload_button () = do_polling ()
```

やりたい処理

ＯＫ、何か問題が？

もし「ポーリング開始イベント」があったら…

```
let _ =    (*  polling_eventに処理を登録する *)
    Event.listen polling_event (fun () ->
      locking := true;
      polling ~post:(fun () -> locking := false))
```

- ポーリング開始イベントに「やりたい処理」を接続するだけ。読む方も意味が取りやすい。
- タイマーイベントとリロードボタンクリックイベントはあるが、ポーリング開始イベントは無い。
- ポーリング開始イベントはどうやって作る？

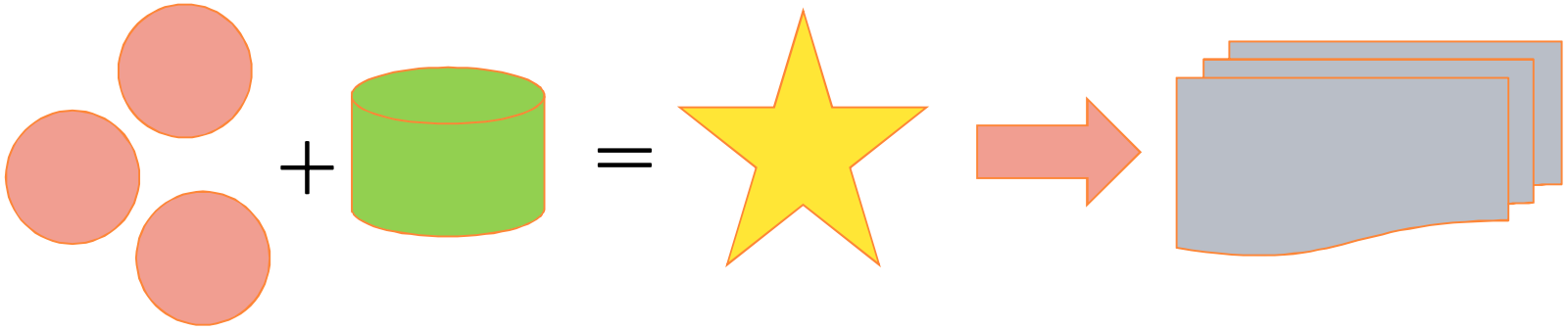
そこで第一級のイベントとコンビネーター

```
let polling_event =  (*  所望のポーリング開始イベント *)
    Event.choose [ reload_button_click;
                                timer_expire; ]
    |> Event.filter (fun () -> !locking = false)
```

- Event.chooseは指定された複数のイベントから、その内のどれかが発火すれば発火するイベントを作りだす関数。
- Event.filterはイベントから指定された条件が満たされないと発火しないイベントを作り出す関数。
- タイマーイベントとリロードイベントをchooseしてロック条件でfilterすれば、ポーリングを開始するイベントの出来上がり！

意味イベントを構築する

- 生イベントから「やりたい処理」に近い意味イベントを作り出す。
- 一塊のイベント処理が、「生イベントから意味イベントへの変換」と「やりたい処理」とに分離でき、見通しが良くなる。
- 生イベントから意味イベントへの変換部分だけテストすることも可能。



便利なイベントコンビネーター達

- map : 'a event -> ('a -> 'b) -> 'b event
- choose : 'a event list -> 'a event
- filter : 'a event -> ('a -> bool) -> 'a event
- fold : ('a -> 'b -> 'a) -> 'b -> 'b event -> 'a signal
- switch : a event event -> 'a event
- never : 'a event
- once : 'a event -> 'a event
- zip : 'a event -> 'b event -> ('a * 'b) event
- delay : int -> 'a event -> 'a event
- take_while : ('a -> bool) -> 'a event -> 'a event

イベントコンビネーターライブラリ

- Microsoft The Reactive Extensions(Rx)
  - *..is a library to compose asynchronous and event-based programs using observable collections and LINQ-style query operators.*
- Haskell Libraries
  - Sodium
  - Reactive
- OCaml Libraries
  - React
  - PEC
- Scheme(Racket)
  - FrTime

まとめ

- イベントドリブンプログラミングは、多くの状態や複雑な条件分岐が登場する、やっかいな状態遷移プログラミング。
- 代数的データ型とパターンマッチをうまく使うと、管理すべき状態が減り、網羅性もチェックできる。
- 第一級のイベントとコンビネーターを使うと、生のイベントを意味イベントに翻訳する事ができる。意味イベントと「やりたい処理」を結び付ければ、イベントドリブンプログラムは分かりやすくなる。

Let’s event driven development!

ご清聴ありがとうございました。

IT Planning, Inc.