Kahuaによる継続渡しスタイルWebプログラミング

Kahua プロジェクト

備前 達矢(び)

継続渡しスタイル(CPS)とは?

普通の手続き呼び出しスタイルでは…

- 手続きは値を戻す。
- 手続きの後で行いたい処理は、手続きを呼んだ側が手続きから値を受け取って実行する。

```
(let ((value (a-proc)))  ;; a-procから値を受け取って
  (next-proc value))     ;; その値をnext-procに渡して実行する
```

継続渡しスタイル(CPS)とは?

継続渡しスタイル(Continuation-Passing Style / CPS)では...

- 手続きは戻らない(呼んだら呼びっぱなし)。
- 手続きの後で行いたい処理は、その手続きに引数として渡す(継続渡し)。
- 手続きは処理の最後に、渡された継続を呼び出す(末尾呼び出し)。

```
(a-proc next-proc)  ;; a-procが中でnext-procを末尾呼び出しする
```

なぜWebプログラミングにCPSなのか

Webアプリケーションは、処理の流れを分断する形でrequest-responseサイクルが入る。

- 例えばショッピングカート。
- セッション開始から買い物完了までの間にrequest-responseサイクルが何度か回る。
- requestでサーバサイドに、responseでクライアントサイドに制御が渡る。
- その間、サーバ側で状態を保持する必要がある。

なぜWebプログラミングにCPSなのか

CPSなら処理の途中で相手(Webクライアント)に制御を渡すのが簡単

- サーバは制御を返したいところで、残りの処理をクロージャで包んで継続手続きとする。
- この継続手続きをレスポンスデータにリンクとして埋め込んでクライアントに渡す(制御をいったん返す)。
- クライアントはリンクをつつくことでサーバに継続手続きを起動するよう伝える(サーバに制御を返す)。

なぜWebプログラミングにCPSなのか

クロージャを継続手続きに使用すると...

- クロージャは自由変数として状態を保持できる。
- 従って、状態をセッションオブジェクトにわざわざ保存する必要がない。
- 関数的に書けばリプレイやクローンにも対応可能。

要求される言語要素は?

- CPSは言語によらない
- ただし、クロージャを継続として使用するなら:
  - 無名手続き(lambda)
  - 静的スコープ
  - 無限エクステント
  - マクロもあればさらに楽ができる

Kahuaとは?

- Scheme処理系Gauche上に構築された継続ベースのアプリケーションフレームワーク/サーバ。
- オープンソースソフトウェア(修正BSDライセンス)
- 「継続ベース」を名乗るのは、CPSでアプリケーションを書くための様々な機能を持っているから。
- その他、組み込みオブジェクトデータベース、S式によるプロトコルなどなど。
- http://www.kahua.org/

KahuaにおけるCPSの実現

- 継続手続きを引数を取らない手続き(thunk)として表現。
- 継続手続きをクライアントに返すHTMLやXMLに埋め込むと、「継続ID」を割り当てる。
- 継続IDからさらにURLを組み立て、ドキュメント内にリンクとして埋め込む。これをクライアントに返す。
- クライアント側でそのリンクをつつくと、URLに結びつけられた継続手続きが起動する。

Kahuaにおける継続手続き

- 無名継続
- 有名継続
- 部分継続

Kahuaにおける継続(1) - 無名継続

- 継続IDとURLはKahuaが自動的に割り当てる。
- クライアント側からはURLの予測がつかない(無名)。
- HTMLにリンクとして埋め込むか、POST先として使われることが多い。
- Kahuaの継続の基礎

```
(define (counter cnt)
  (html/
   (head/ (title/ "Counter"))
   (body/
    (p/ (a/cont/ (@@/ (cont (lambda () (counter (+ cnt 1))))) "[UP]")
    " "
        (a/cont/ (@@/ (cont (lambda () (counter (- cnt 1))))) "[DOWN]"))
    (p/ "Count: " cnt))))

(initialize-main-proc (lambda () (counter 0)))
```

Kahuaにおける継続(1) - 無名継続

URLの構成: http://localhost/アプリケーション名/継続ID/...

```
(define (counter cnt)
  (html/
   (head/ (title/ "Counter"))
   (body/                          [1]
    (p/ (a/cont/ (@@/ (cont (lambda () (counter (+ cnt 1))))) "[UP]")
    " "                            [2]
        (a/cont/ (@@/ (cont (lambda () (counter (- cnt 1))))) "[DOWN]"))
    (p/ "Count: " cnt))))
                             [3]
(initialize-main-proc (lambda () (counter 0)))
```

[1] 継続IDが指定される: http://localhost/アプリ名/**1-g8x:4dxlf-1khbug**

[2] 継続IDが指定される: http://localhost/アプリ名/**1-g8x:4dxlf-39xmzo**

[3] 継続IDが指定されない: http://localhost/アプリ名

Kahuaにおける継続(1) - 無名継続

自由変数cntを状態として使用している。

```
(define (counter cnt)
  (html/
   (head/ (title/ "Counter"))
   (body/
    (p/ (a/cont/ (@@/ (cont (lambda () (counter (+ cnt 1))))) "[UP]")
    " "
        (a/cont/ (@@/ (cont (lambda () (counter (- cnt 1))))) "[DOWN]"))
    (p/ "Count: " cnt))))

(initialize-main-proc (lambda () (counter 0)))
```

Kahuaにおける継続(1) - 無名継続

この例は関数的に書かれているのでクローンも自在。

Counter
[UP] [DOWN]
Count: 0

Counter
[UP] [DOWN]
Count: 1

Counter
[UP] [DOWN]
Count: 2

Counter
[UP] [DOWN]
Count: 3

Counter
[UP] [DOWN]
Count: 1

Counter
[UP] [DOWN]
Count: 0

Kahuaにおける継続(1) - 無名継続

entry-lambda: クエリパラメータやパス要素を宣言的に使うための構文

```
(entry-lambda ([path1 path2 ...]
               [:keyword param1 param2 ...]
               [:mvkeyword mvparam1 mvparam2 ...]
               [:rest restpaths])
  ...)
```

http://localhost/アプリ名/継続ID/a/b/c/d?param1=k&mvparam1=m&mvparam1=n

- → path1 = “a”
- → path2 = “b”
- → restpaths = (“c” “d”)
- → param1 = “k”
- → param2 = #f
- → mvparam1 = (“m” “n”)
- → mvparam2 = ()

Kahuaにおける継続(2) - 有名継続

- クライアントが予測できないURLだけでは不便:
→ パーマネントリンクが欲しい。

- より宣言的にURLと継続とのマッピングを行いたい。

→ 宣言的に継続IDを割り当てる機構が必要

Kahuaにおける継続(2) - 有名継続

define-entry: 継続IDを宣言的に割り当てる構文

```
[1] (define-entry 継続ID thunk)
[2] (define-entry (継続ID ...)
      ...)
    ≡
   (define-entry 継続ID
     (entry-lambda (...)
       ...))
```

```
(define-entry start (lambda () (counter 0)))
```

手続き start を定義し、継続ID startを恒常的に手続きstartに割り当てる。

http://localhost/アプリ名/**start**

というURLで継続 **(lambda () (counter 0))** を呼び出すことができる。

Kahuaにおける継続(2) - 有名継続

さきほどの無名継続の例に手を加えてみる。

```
;; ここは変更なし
(define (counter cnt)
  (html/
   (head/ (title/ "Counter"))
   (body/                            [1]
    (p/ (a/cont/ (@@/ (cont (lambda () (counter (+ cnt 1))))) "[UP]")
    " "                              [2]
        (a/cont/ (@@/ (cont (lambda () (counter (- cnt 1))))) "[DOWN]"))
    (p/ "Count: " cnt))))

;; start という名前の有名継続を宣言
(define-entry (start init) [3]
  (counter (if init (x->integer init) 0)))

;; 継続IDが指定されない場合には継続start(thunkでもある)を起動するよう登録
(initialize-main-proc start) [4]
```

Kahuaにおける継続(2) - 有名継続

この場合、URLとの対応は...

[1] 無名継続: http://localhost/アプリ名/1-g8x:4dxlf-1khbug

[2] 無名継続: http://localhost/アプリ名/1-g8x:4dxlf-39xmzo

[3] 有名継続(パス要素あり): http://localhost/アプリ名/**start/10**

[3] 有名継続(パス要素なし): http://localhost/アプリ名/**start**

[4] 継続ID指定なし: http://localhost/アプリ名

Kahuaにおける継続(3) - 部分継続

CPSではないロジックをCPSに書き換えるのは面倒な場合がある。

- 暗黙の継続を捕捉して使えないか?
- Schemeで暗黙の継続と言えばcall/ccだが、Kahuaではcall/ccをこの用途には使えない。

Kahuaにおける継続(3) - 部分継続

なぜcall/ccではまずいか(Kahuaの事情)

- KahuaはSchemeで書かれたWebアプリケーションフレームワーク/サーバ。
- アプリケーションコードの中でcall/ccを使って捕捉した継続には、捕捉時に使用していたポート(ソケットポート)に対する出力という処理も含まれている。
- その継続が次のrequest-responseサイクルで起動された時には、そのポートはすでにクローズ/破棄されてしまっている。

Kahuaにおける継続(3) - 部分継続

アプリケーションロジック限定の継続を捕捉したい

→ 部分継続

reset/pc: 部分継続を捕捉する範囲を規定する構文

```
(reset/pc ...)
```

call/pc: 部分継続を捕捉するための手続き

```
(call/pc proc)
;; ただし、procは1引数の手続き; 引数は部分継続を表す手続き
```

このふたつをセットで使用する。

Kahuaにおける継続(3) - 部分継続

部分継続の制御の流れを例で見る。

```
;; (1) call/pcがなければreset/pcは効果を持たない
(list 1 (reset/pc (list 2 3 4 5)))
→ (1 (2 3 4 5))

;; (2) call/pcの引数となる手続きは、直接reset/pcから返る
(list 1 (reset/pc (list 2 (call/pc (lambda (k) (list 3 4))) 5)))
→ (1 (3 4)

;; (3) kが呼ばれると、その引数がcall/pcから返り、reset/pcに
;; 包まれた継続を実行してkから返る
(list 1 (reset/pc (list 2 (call/pc (lambda (k)
                                     (list (k 3) 4)))
                        5)))
→ (1 ((2 3 5) 4))
```

Kahuaにおける継続(3) - 部分継続

```
;; 部分継続kとして補足されている範囲
(list 1 (reset/pc (list 2 (call/pc (lambda (k)
                                        (list (k 3) 4)))
                          5)))
```

Kahua固有の注意点:

- reset/pcやcall/pcはKahua固有の名前。Kahuaの実装の元になった論文では、reset、shiftという構文を使用していた。
- Kahuaアプリケーションの中では、reset/pcを書くことはない。なぜなら、Kahua側で、継続手続きを起動する直前にreset/pcしているから。

Kahuaにおける継続(3) - 部分継続

Kahuaでの使用例: 必要に応じてプロンプトを出す

```
(define (get-number)
  (call/pc
   (lambda (k)
     (html/
      (head/ (title/ "Enter your number"))
      (body/
       (p/ "Enter your number")
       (form/cont/ (@@/ (cont (entry-lambda (:keyword number)
                                (k (x->integer number)))))
           (input/ (@/ (type "text") (name "number")))
           (input/ (@/ (type "submit") (value "submit")))))))))

;; パス要素として初期値が与えられなかったら入力画面から入力を受ける
(define-entry (start init)
  (counter (if init
               (x->integer init)
               (get-number))))
```

Kahuaの過去、現在、未来

- CPSに関係する基盤部分はかなり初期にほぼ出来上がっていた。
- 昨年は実用できるレベルの安定性を目指して開発を進め、年末に初の安定版をリリース。
- 現在はさらなる安定性と可用性、開発効率のよさを目指して開発継続中。

Kahuaの過去、現在、未来

CPSに関わる構想:

- 現在の継続手続きはプロセスローカル。
- 継続手続きを永続化できれば、プロセスを再起動しても、その継続を起動できる。
- プロセス間やサーバ間で継続を移送できれば、柔軟性や可用性が向上する。
- ただし、Gauche側に対応が必要かも。

参考文献

- Martin Gasbichler and Michael Sperber: Final Shift for Call/cc: Direct Implementation of Shift and Reset, ICFP 02, October 2002.

  http://citeseer.ist.psu.edu/gasbichler02final.html

- 川合史朗: 第1回Kahuaセミナー資料「継続渡しによるWebアプリケーション」

  http://www.kahua.org/kahua/docserv/seminar200402/cps/

ご清聴ありがとうございました