Kahuaプログラミングスタイル

〜あるいはKahua Inside Out〜

Kahuaプロジェクト/タイムインターメディア
備前 達矢(び)

2008年9月13日
Gauche/Kahuaセミナー2008 Fall

Kahuaプログラミングスタイル

- このセミナーの話が出た時、とりあえずどうにでも転ばせられるタイトルにしてみた。
- でも、いつものKahuaアプリケーションプログラミングの紹介というのでは芸がない。
- Kahuaの内部を少し詳しく紹介してみる。
- Kahuaの中身を覗いて、ちょっとエグいGaucheの使い方に目覚めてもらえると嬉しいな。
- もちろん、**Kahuaをつかってもらえると**もっといいな。

お前誰よ(お約束)

- Kahuaプロジェクトのメインプログラマ
- 途中参加(2006年5月から)
- それ以前は単なるユーザ
- Kahua創成期を知らぬ世代
- コンセプチュアルな部分より、基盤部分の強化を中心に開発を進めた
- 最近、あまり活発とは言えない

Kahuaって何よ(お約束)

- アプリケーションサーバ/フレームワーク
- Scheme(Gauche)で書かれている
- Gaucheでアプリケーションコードを書く
- Web専用ではないが現実的にはWeb向き
- オープンソースソフトウェア(修正BSDL)
- 継続ベース: CPSで書くための便利機能
- 組み込みオブジェクトデータベース
- S式でプロセス間通信/ストレージ
- 動的でインクリメンタルな開発が可能

Kahuaの中身でネタになりそうなのは...

- 継続ベース(継続渡しスタイル)
- オブジェクトデータベース
- 高階タグ手続き

要はController、Model、View

別に無理にMVCに当てはめて考える必要はないけど

Kahuaにおける継続渡しスタイル

継続ベース

- Kahua最大の謳い文句
- 継続渡しスタイル(Continuation Passing Style/CPS)でアプリケーションコードを書くから、書きやすいから**継続ベースサーバ/フレームワークと謳う**

継続渡しスタイルのおさらい

普通の関数呼び出しスタイル

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

継続渡しスタイル

```
(define (first-proc cont)
  (let1 value (...) ;; first-procの主な処理
    (cont value)))  ;; 主な処理の値を継続処理に渡して末尾呼び出し

(first-proc second-proc) ;; second-procを継続処理としてfirst-procに渡す
```

Webにおける継続渡しスタイル

継続渡しスタイルなら、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、残りの処理を継続手続きに仕立てる。
- 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(サーバに制御を渡す)。

Kahuaにおける継続渡しスタイル

- 継続手続きをクロージャ(実際には引数を取らないthunk)として表現
- 継続手続きを特定する**継続ID**を割り当てる
- **継続IDからURI**を組み立てる
- レスポンスデータ内に埋め込まれた継続手続きをURIで置き換える
- Webクライアントがそのに URIにリクエストを発行すると、継続手続きが実行される

継続手続きとしてのクロージャ

メリット

- Schemeとしては自然(無名手続き、静的スコープ、無限エクステント、マクロ etc...)
- 動的環境に状態を保存しなくても、自然変数として状態を保存できる
- 関数的に書いてあげればリプレイやクローンにも対応可能

継続手続きとしてのクロージャ

デメリット

- 継続手続きはプロセスローカルになる
- 同一アプリケーションを複数プロセスで動かしている時、継続を埋め込んだプロセスが必ずその継続手続きを実行しなければならない
- プロセスを再起動すると、全ての継続手続きが消えてしまう。

継続手続きを作る(無名継続)

Kahuaの継続手続きはthunk(引数を取らないクロージャ)

```
(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)))
```

- 単なるlambda式で記述できる
- 継続IDやURIはKahuaに自動的に割り当てられ、予測できない

  → 無名継続
- クエリパラメータや付加されたパスは処理できない

継続手続きを作る(無名継続)

URI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))))

(initialize-main-proc (lambda () (counter 0)))  [3]
```

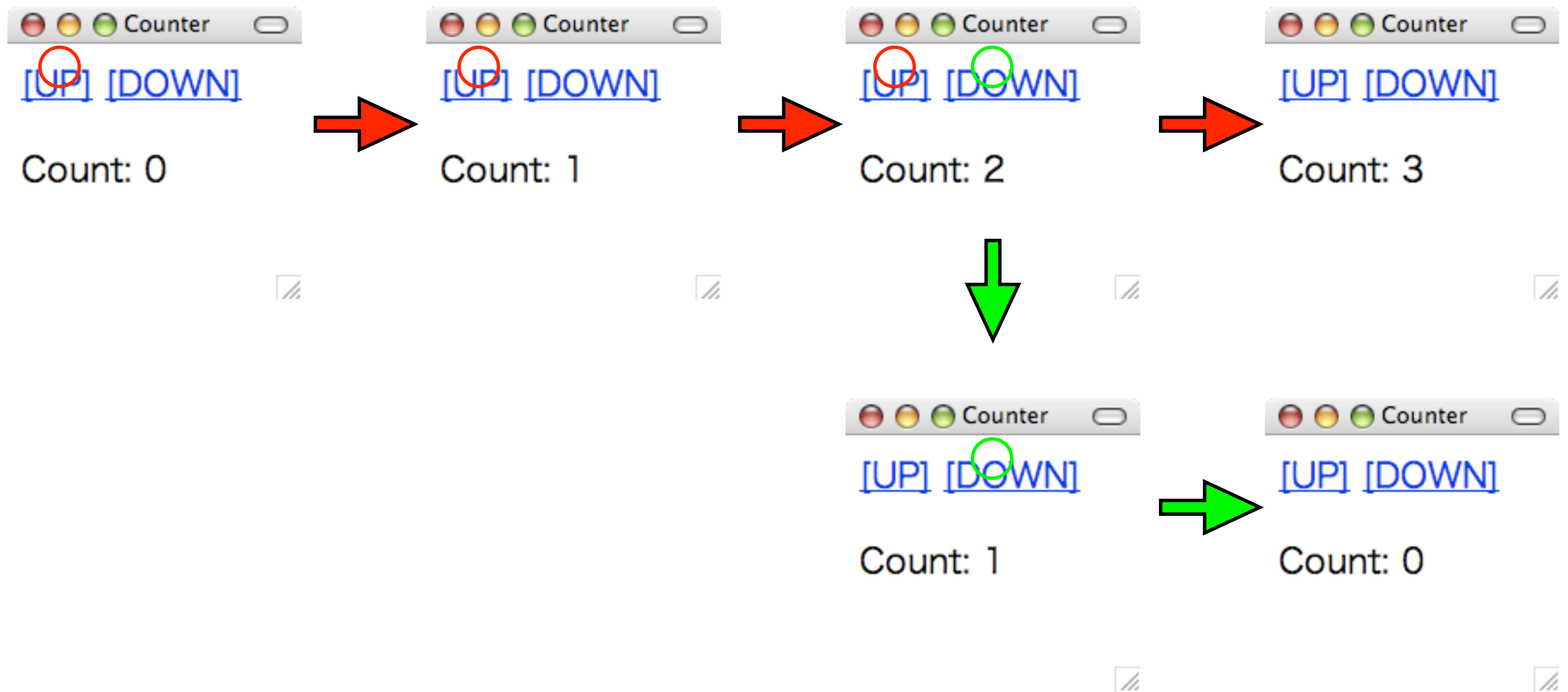
[1] 継続IDが指定される: http://localhost/アプリ名/**1-g8x:4dxlf-1khbug**

[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

継続手続きを作る(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。



継続手続きを作る(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)))
```

継続手続きを作る(無名継続)

クエリパラメータを扱う継続手続き

```
(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 (entry-lambda (:keyword init)
                        (counter (if init (x->integer init) 0))))
```

- entry-lambda式で使用するクエリパラメータを宣言的に定義できる
- それ以外はlambda式と同じ
- http://localhost/アプリ名?init=10

継続手続きを作る(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 = ()

entry-lambdaの実装

kahua.serverモジュール内でマクロとして定義

```
(define-macro (entry-lambda args body1 . bodys)
  ;; 実際にはここにヘルパー手続きparse-argsの定義がある
  (receive (ps ks ms rs) (parse-args args)
    `(,make-parameterized-entry-closure
      ',ps ',ks ',ms ',rs (lambda (,@ps ,@ks ,@ms . ,rs)
                              ,body1 ,@bodys))))
```

- 手続きparse-argsでentry-lambdaのパラメータ定義を解釈して各種パラメータ名をリスト(rsのみシンボル)にまとめる
- make-parameterized-entry-closureの呼び出しを組み立ててそれを実行する

entry-lambdaの実装

```
(define (make-parameterized-entry-closure pargs kargs mvkargs rarg
proc)
  (let ((kargs-str (map x->string kargs))
    (mvkargs-str (map x->string mvkargs)))
     (lambda ()
       (let1 path-info (kahua-context-ref "x-kahua-path-info" '())
         (apply proc
                (append (map-with-index (lambda (ind parg)
                                          (list-ref path-info ind #f))
                                        pargs)
                        (map kahua-context-ref kargs-str)
              (map kahua-context-ref* mvkargs-str)
                        (if rarg
                          (drop* path-info (length pargs))
                          '()))))))
    ))
```

無名継続への継続IDの割り当て

- 無名継続への継続ID割り当てとURIへの変換は、レスポンスドキュメントをXHTMLにレンダリングする処理の中で行われる
- lambda式で作った継続手続きもentry-lambdaで作った継続手続きも、扱いは同じ

無名継続への継続IDの割り当て

kahua.sessionモジュール内の手続きsession-cont-register

```
(define (session-cont-register cont . maybe-id)
  (and (procedure? cont)
       (let ((entry (cont-closure->session cont))
             (given-id (get-optional maybe-id #f)))
         (if entry
           (if (or (null? maybe-id)
                   (equal? given-id (ref entry 'key)))
             (ref entry 'key)
             (replace-key entry given-id))
           (ref (apply make <session-cont>
                       :closure cont
                       (if given-id
                         `(:key ,given-id :permanent? #t)
                         '()))
                'key)))))
```

無名継続への継続IDの割り当て

- オプショナル引数が与えられていたら、それを固定の継続IDとして使用(後述)
- なければすでにその継続手続きが登録済かどうかを調べ、登録済なら登録された継続IDを返す(cont-closure->session)
- そうでなければ継続手続きを登録し、対応する継続IDを生成して(cont-closure->session内のmake-cont-key)、それを返す

固定継続IDの割り当て

- 無名継続の継続IDはリクエスト処理の度に自動的に生成される。
- すなわち、継続手続きに結びつけられたURIがクライアント側から予測不可能
- パーマネントリンクが実現できない
- パーマネントリンクを実現するには、継続IDを固定で割り当てる仕組みがあればよい

これを**有名継続**と呼んでいる

固定継続IDの割り当て

define-entry: 固定継続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。

固定継続IDの割り当て

```
;; ここは変更なし
(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))))

;; start という名前の有名継続を宣言
(define-entry (start :keyword init)
  (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)
```

継続IDとURIとのマッピング(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(パラメータあり): http://localhost/アプリ名/**start?init=10**
- **[3]** 有名継続(パラメータなし): http://localhost/アプリ名/**start**
- **[4]** 継続ID指定なし(パラメータなし): http://localhost/アプリ名
- **[5]** 継続ID指定なし(パラメータあり): http://localhost/アプリ名**?init=10**

define-entryの実装

kahua.serverモジュール内でマクロとして定義

```
(define-syntax define-entry
  (syntax-rules ()
    ((define-entry (name . args) . body)
     (define-entry name (entry-lambda args . body)))
    ((define-entry name expr)
     (define name
       (let* ((closure expr)
              (x (lambda ()
                   (parameterize ((kahua-current-entry-name
                                   (symbol->string 'name)))
             (closure)))))
         (add-entry! 'name x)
         x)))
    ))
```

define-entryの実装

- define-entryに与えた継続ID(継続名)と同じ名前を継続手続きに束縛する
- この継続手続きを与えた継続IDにて登録する(add-entry!)

```
;; kahua.serverモジュール内
(define (add-entry! name proc)
  ;; session-cont-registerのオプショナル引数として固定継続IDが渡っている
  (session-cont-register proc (symbol->string name)))
```

Kahuaにおける継続渡しスタイル

- lambda式による無名継続
- entry-lambda式による無名継続
- define-entry式による有名継続

を適宜使い分けることになる

本当は部分継続(call/pc)もあるけど今回は割愛

Kahuaオブジェクトデータベース

Kahuaオブジェクトデータベース

- GaucheのMOPに依存している
- プログラマは永続オブジェクトを明示的に保存する必要がない(新規オブジェクトや更新されたオブジェクトは自動的に保存される)
- ファイルシステムベースのストレージ(Pure Gaucheで実装)の他、MySQLやPostgreSQLをストレージとしてサポートしている(要DBDモジュール)
- 拡張されたクラス再定義機構

永続クラスの定義

<kahua-persistent-base>クラスを継承する

```
(define-class <bookmark-entry> (<kahua-persistent-base>)
  ((url :init-keyword :url :allocation :persistent :index :unique)
   (title :init-keyword :title :allocation :persistent :index :any)
   (score :init-value 1 :allocation :persistent)
   (count :init-value 0 :allocation :persistent)))
```

永続クラスの定義

保存するスロットを :allocation :persistent スロットオプションで明示する

```
(define-class <bookmark-entry> (<kahua-persistent-base>)
  ((url :init-keyword :url :allocation :persistent :index :unique)
   (title :init-keyword :title :allocation :persistent :index :any)
   (score :init-value 1 :allocation :persistent)
   (count :init-value 0 :allocation :persistent)))
```

永続クラスの定義

検索に使用するスロットを :index :unique もしく

は :index :any スロットオプションで指定する

```
(define-class <bookmark-entry> (<kahua-persistent-base>)
  ((url :init-keyword :url :allocation :persistent :index :unique)
   (title :init-keyword :title :allocation :persistent :index :any)
   (score :init-value 1 :allocation :persistent)
   (count :init-value 0 :allocation :persistent)))
```

- 重複を許さない場合は :index :unique
- 重複を許す場合には :index :any
- いずれも、実体化されたオブジェクトは、オンメモリのインデックステーブルにキャッシュされる

<kahua-persistent-base>

```
(define-class <kahua-persistent-base> ()
  ((%kahua-persistent-base::id :init-keyword
      :%kahua-persistent-base::id
      :init-form (%kahua-db-unique-id) :final #t)
   (%kahua-persistent-base::removed?
      :init-keyword :%kahua-persistent-base::removed?
      :init-value #f :final #t) ; It's removed?
   (%kahua-persistent-base::db :init-form (current-db) :final #t)
   (%hidden-slot-values :init-keyword :%hidden-slot-values
      :init-value '())
   (%persistent-generation
      :init-keyword :%persistent-generation :init-value 0)
   (%floating-instance :init-keyword :%floating-instance
      :init-value #t)
   (%transaction-id :init-value -1)
   (%in-transaction-cache :init-value '())
   (%persistent-key :init-value #f)
   (%modified-index-slots))
  :metaclass <kahua-persistent-meta>)
```

永続クラスに関わるその他のクラス

<kahua-persistent-meta>: 永続化を司るメタクラス

```
(define-class <kahua-persistent-meta> (<class>)
  ((class-alist :allocation :class :init-value '()
                :accessor class-alist-of)
   (source-id   :init-keyword :source-id :init-value "")
   (metainfo      :init-value #f)
   (generation    :init-value 0)
   (persistent-generation :init-value #f)
   (write-syncer  :init-value :ignore
                  :init-keyword :write-syncer)
   (read-syncer   :init-value :auto
                  :init-keyword :read-syncer)
   (index-cache   :init-value #f)))
```

<kahua-proxy>と<kahua-wrapper>

<kahua-proxy>: 他の永続オブジェクトへの参照を保存

```
(define-class <kahua-proxy> ()
  ((class :init-keyword :class)     ;; 永続クラス
   (ident   :init-keyword :ident))) ;; 永続オブジェクトID
```

<kahua-wrapper>: 他のオブジェクトへの参照評価を遅延

```
(define-class <kahua-wrapper> ()
  ((value :init-keyword :value)))
```

永続オブジェクトのスロットが他の永続オブジェクトへの参照の場合、スロット値そのものが参照されるまでスロット値の実体化を保留する

永続化の書式

define-reader-ctorで登録したリーダ(<kahua-persistent-base>を継承したクラスのオブジェクトなら手続きkahua-object2-read)で読み込める形式で永続化される

```
#,(kahua-object2                ;; リーダーマクロに割り当てたタグ
(<bookmark-entry> 0)            ;; クラス名とクラス定義の世代番号
1                               ;; オブジェクトID(db内でユニーク)
#f                              ;; 削除フラグ
(url . "http://www.kahua.org") ;; 以下はスロット名と値
(title . "Kahua Project")
(score . 3)
(count . 1)
)
```

永続化の書式

```
(define-class <tags> (<kahua-persistent-base>)
  ((name :init-keyword :name :init-value ""
         :allocation :persistent)))

(define-class <bookmark-entry> (<kahua-persistent-base>)
  ((url :init-keyword :url :allocation :persistent :index :unique)
   (title :init-keyword :title :allocation :persistent :index :any)
   (score :init-value 1 :allocation :persistent)
   (count :init-value 0 :allocation :persistent)
   (tags :init-form (make <tags>) :allocation :persistent)))
```

永続オブジェクトへの参照を保持するスロットが存在する場合

永続化の書式

```
#,(kahua-object2
(<bookmark-entry> 1)             ;; クラス定義の世代番号が上がる
1                                ;; オブジェクトIDは不変
#f
(url . "http://www.kahua.org")
(title . "Kahua Project")
(score . 5)
(count . 1)
(tags . #,(kahua-proxy <tags> 5)) ;; kahua-proxy-readが読める書式
)
```

- #,(kahua-proxy ...)を読み込むと、<kahua-proxy>のインスタンスをくるんだ<kahua-wrapper>のインスタンスがスロット値としてセットされる
- tagsスロットが参照されると<kahua-wrapper>と<kahua-proxy>が剥がされて参照先のオブジェクトが実体化される

スロットアクセスのカスタマイズ

- 永続オブジェクトの永続スロット (:allocation :persistent なスロット)は、値が変更されたらデータベースに書き出さなければならない
- <kahua-wrapper>で実体化が遅延されているオブジェクト参照は、そのスロットへのアクセス時に実体化しなければならない

→ メソッド compute-get-n-set をオーバーライド

compute-get-n-set(アクセッサを生成)

```
(define-method compute-get-n-set ((class <kahua-persistent-meta>) slot)
  ;; 実際には delete-slot-definition-allocationの定義が入る
  (let ((alloc (slot-definition-allocation slot)))
    (case alloc
      ((:persistent)
       (let1 acc (let1 slot (delete-slot-definition-allocation slot)
                   (compute-slot-accessor class slot (next-method class slot)))
         (inc! (slot-ref class 'num-instance-slots))
         (list (make-kahua-getter acc class slot)
               (make-kahua-setter acc slot)
               (make-kahua-boundp acc)
               #t)))
      (else (next-method)))))
```

アクセッサ自体はmake-kahua-{getter,setter,boundp}で生成している

make-kahua-getterが生成するgetter

- データベース上のデータとの整合性の確認
- そのスロットの値が<kahua-wrapper>のインスタンスだったら、参照先の永続オブジェクトを実体化
- 参照先の永続オブジェクトに削除フラグが立っていたら、スロット値に#fを設定して更新リストに永続オブジェクトを追加

make-kahua-setterが生成するsetter

- データベース上のデータとの整合性の確認
- インデックススロットだったらオンメモリキャッシュを更新
- 更新リストに永続オブジェクトを追加

make-kahua-boundpが生成するbound?

- 現状では特別な処理はなく、通常の束縛チェックのみ

スロットの定義のカスタマイズ

- :final スロットオプションのサポート
  → #tだったらオーバーライドを禁止する

→ メソッド compute-slots をオーバーライド

compute-slots(スロット定義を生成)

```
(define-method compute-slots ((class <kahua-persistent-meta>))
  (receive (slots vs)
      (fold2 (lambda (class slots vs)
           (fold2 (lambda (s slots vs)
            (if (assq (slot-definition-name s) slots)
                (if (slot-definition-option s :final #f)
                (values slots (cons s vs))
                (values slots vs))
                (values (cons s slots) vs)))
              slots vs
              (class-direct-slots class)))
         '() '()
         (class-precedence-list class))
    (unless (null? vs)
      (errorf "Class ~s attempted to override slot(s) ~a, ..."
           class (string-join
                   (reverse! (map (lambda (s) (symbol->string (slot-definition-name
s))) vs)) ", ")))
    (reverse! slots)))
```

クラスの再定義

- クラスが再定義されると、既存のオブジェクトはその定義に追随して自動的に変換される
- 別のプロセスが古いクラス定義を使用し続けても、そのクラス定義に基づいて永続オブジェクトが実体化される。

永続クラスの定義情報

<kahua-persistent-metainfo>: 永続クラスの定義の一部を保存

```
(define-class <kahua-persistent-metainfo> (<kahua-persistent-base>)
  ((name                  :allocation :persistent :init-value #f
                          :init-keyword :name)
   (generation            :allocation :persistent :init-value 0
                          :init-keyword :persistent-generation)
   (previous-generation :allocation :persistent :init-value 0
         :init-keyword :previous-generation)
   (signature             :allocation :persistent :init-value '()
                          :init-keyword :signature)
   (signature-alist       :allocation :persistent :init-value '()
                          :init-keyword :signature-alist)
   (source-id-map         :allocation :persistent :init-value '()
                          :init-keyword :source-id-map)
   (translator-alist      :allocation :persistent :init-value '()
                          :init-keyword :translator-alist)
   (index-translator :init-value '() :accessor index-translator-of)))
```

メソッド ensure-metainfo

メソッド ensure-metainfo でデータベース上の<kahua-persistent-metainfo>インスタンスとプロセス内での対応するクラス定義とをつきあわせる

```
(define-method ensure-metainfo ((class <kahua-persistent-meta>))
  (unless (slot-ref class 'metainfo)
    (persistent-class-bind-metainfo class))
  (ref class 'metainfo))
```

実際の処理を行っているのはpersistent-class-bind-metainfo

メソッド persistent-class-bind-metainfo

kahua.persistenceモジュール内で定義される長大なメソッド

- クラスのスロット定義とmetainfoに保存されているシグネチャ(スロット定義のダイジェスト情報)を比較する
- 現在のシグネチャと一致しない場合、過去のシグネチャリストに一致するものがあればそれに基づいてmetainfoを扱う
- 一致するものがなければ世代をひとつ進める

高階タグ手続き

Kahua

高階タグ手続き

- 手軽なページ記述言語として機能している
- 本来の構想では、ページを構築する過程でページ内のコンテキスト処理を並行して行うことになっていた(らしい)
- 現在の実装は、単にページを表すSXMLを組み立てるだけで、コンテキスト処理は行っていない
- 一応、ステートモナドとして実装されているが、それを活かしていないということ

高階タグ手続き

よって今回は割愛します。

ごめん。

ご清聴ありがとうございました