3P-5

No-method-found ハンドラとその応用

竹内 郁雄　　大里 延康

NTT ソフトウェア研究所

はじめに

メッセージ伝達式の評価において，メソッドが見つからないことを積極的な意味で使うことが可能である．本論文では，TAO/ELIS 上で，端末と計算機の結合を仮想化する detach/attach 機能をインプリメントする際に，これがどう役立ったかについて報告する．

1. No-method-found ハンドラ

TAO のメッセージ伝達式は次のような形をしている．

```
[receiver selector arg ...]
```

ここで，receiver はあるクラスのインスタンス (TAO では user defined object あるいは略して udo と呼ぶ) であり，selector はシンボルである．メソッドは，receiver の属するクラスが保持する二分探索表の中で探索される (この二分探索表はスーパークラスから継承されたメソッドをすべて含んでいる)．

従来，TAO はこの二分探索表にセレクタが登録されていない場合，ただちに no-method-found というエラーを引き起していた．Smalltalk-80 では，見つからなかったメッセージを引数とする doesNotUnderstand: というメッセージを再びレシーバに送る．これにより，そのレシーバが doesNotUnderstand: というセレクタを知っていれば，自分でエラー処理を行なうことができる．

TAO は，2 で述べる detach/attach 機能の実現のために，no-method-found ハンドラを導入した．これは基本的に doesNotUnderstand: と同じであるが，効率を上げるために，no-method-found ハンドラの呼び出しに二重のメソッド探索を使わない．すなわち，二分探索表のキーにシンボルの最小元 [TAO の場合は nil] を入れておき，メソッドが見つからなかったら，二分探索表の第 1 要素にアクセスすることにより直ちにハンドラを見つけるのである．このような効率重視は以下のような使い方では本質的である．

No-method-found ハンドラは次のように定義する．

```
(defmethod (CLASS) (sel &rest args) ...)
```

TAO の通常のメソッド定義は

```
(defmethod (CLASS SELECTOR) ARGS ...)
```

だから，no-method-found ハンドラは名無しのメソッドという格好になる．受け取る引数は，見つからなかったセレクタと，もとのメッセージの (評価済みの) 引数のリストである．

2. detach/attach 機能

多くのマルチユーザシステムでは，端末とログインセッションの結合が直接的である．単純で効率もよいからだろう．しかし，TOPS-20 のようにこの結合が間接的だと便利なことが多い．TOPS-20 では，ログインセッションを保持したまま，端末とホストの結合を切断すること (detach) が可能である．そして，別の端末からこのログインセッションに再結合すること (attach) が可能である．

この機能を使うとログインしたままで仕事の場所を変えることができる．また，端末が 1 台しかないときに，(ログインを切らないで) 端末の割り込み使用が可能になる．もっと有り難いのは，TELNET などを通じてリモートログインしていたとき，ネットワーク事故により，突然ホストとの接続が切られても，ホストが強制的に detach してくれれば，再接続後の attach により，それまでのセッションの結果を失わずに済むことである (この機能の欠如は，Unix が TOPS-20 に比べて決定的に劣る点の 1 つであろう)．このときローカル端末から attach して，必要最低限のセーブをすることももちろん可能である．

従来，TAO/ELIS は端末とログインセッションとの結合が直接的であった．これを detach/attach が可能な

No-method-found handler and its application

Ikuo TAKEUCHI and Nobuyasu OSATO

NTT Software Laboratories

ような間接結合に変更するのは，ほとんどシステム全体の作り直しに近い．理由は，

(1) 端末に対応する入出力ストリームがオブジェクト (udo) として表現されている．これらはオブジェクト指向の精神により，物理的な端末に直接対応している．単に *terminal-io* や *standard-input* の値を変更するだけでは済まない．物理的な端末を部品としている高度な入出力ストリーム (たとえば，ZEN エディタと同じ入出力インタフェースをもつ zen-eval-stream) などがデータ構造の中に (インスタンス変数の値として) それらを抱え込んでしまっている．
(2) 入出力デバイスをコントロールする ELIS の FEP と，ELIS の間のインタフェースで結合を間接化することも可能だが，すでに何種類かある FEP プログラムのすべてを作り直さないといけない．

3. メッセージのタライ回し

TAO/ELIS 上の detach/attach 機能は，オブジェクトを変身させることと，no-method-found ハンドラによるメッセージのタライ回しによって実現された．問題の所在を明確にするために次の場合を考えよう．ローカル端末から detach して，TELNET を通じて attach したとする．detach 直前の基本入出力ストリームは TAO の fundamental-stream (以下 FS) という基本的なストリームである．ところが attach したあとの基本入出力ストリームは telnet-server-stream (以下 TS) になる．TS は FS の孫クラスであり，120 個以上のインスタンス変数をもつ大きな udo である (ちなみに，FS のインスタンス変数は 14 個)．

上にも述べたように，入出力ストリームの udo はログインセッションの中の変数からばかりではなく，いろいろなデータから直接指されている．だから，もとの FS をすべて新しい TS にすり替えるのは，ゴミ集めのようなマーキングを行なわないかぎり，ほとんど不可能である．もとの FS の残骸を再利用する方策が必要である．そこで，deferred-stream というクラスを定義する (クラス変数，スーパークラスリストは () である)．

```
(defclass deferred-stream
    () (sys:stream-status sys:stream)
    () (:gettable sys:stream) )
```

もとの FS は上のように detach/attach されると，FS の udo のクラスへのポインタが deferred-stream クラスへのポインタに張り替えられる (生みの親であるクラスをすげ替えてしまう)．sys:stream というインスタンス変数の値は新たに開かれた TS になる．ここで，

```
(defmethod (deferred-stream)
    (selector &rest args)
    (if args
        (apply sys:stream
           (cons selector args) )
        (send sys:stream selector) ))
```

という no-method-found ハンドラを定義しておく．(注意: TAO は，評価済みの引数のリストをメッセージにつけたいとき，send ではなく apply を使う．) これによって，deferred-stream に変身した FS から，TS にメッセージがタライ回しされる様子は容易に理解されよう．代理人である deferred-stream はなにも知らないので，ほとんどのメッセージをそのまま sys:stream の値である本物のストリームへバイパスするわけである．

このほか，detach/attach のためにシステムテーブルの書き替えなどがあるが，問題の最も本質的な部分はこのような簡単なアイデアで完全に解決してしまった．この方法によると，通常のログインは結合が直接的なままなので効率がよいことに注意しておこう．また，Common Lisp の synonym-stream と deferred-stream とは意味がまったく異なることにも注意しておこう．

おわりに

大きなシステム変更を伴うと思われた detach/attach 機能の導入は，no-method-found ハンドラの活用とオブジェクトの変身によって，非常に簡単に実現することができた．実際，detach/attach のための変更は，マイクロコードの 2 ステップ追加と，login の手続きに対する attach コマンド付加 (5 行) だけで，残りは 300 行足らずの TAO のコードをシステムに追加しただけである (no-method-found ハンドラ自身のための変更ステップは除く)．すなわち，端末を通して入出力を行なう既存のすべてのシステムプログラムとアプリケーションプログラムにはなんの変更もなかったわけである．

オブジェクトの変身という荒業は使ったものの，これはオブジェクト指向パラダイムの有効性を示したいい例題であると考えられる．ここでは，no-method-found ハンドラの 1 つの応用を示したにすぎない．このほかにも面白い応用があり得るだろう．

[文献] A. Goldberg and D. Robson: Smalltalk-80, The language and its implementation, Addison-Wesley, 1983.