記　号　処　理　47－3
（１９８８．９．２２）

オブジェクトトレーサ

大久保　清貴　ohkubo@pfrad.pf.fujitsu.junet
（株）ＰＦＵ 研究開発部

通常のトレーサはプログラムの特定の点にかけて、そこを通過する時のデータを表示する。それに対し本文で述べるオブジェクトトレーサはデータにかけ、そのデータがどう操作されるかを追う。このトレーサはユーザによって選択された Smalltalk のバーチャルイメージ内のオブジェクトを対象にし、そのオブジェクトに送られてくるメッセージをトレースする。このツールは Smalltalk のウィンドウを構成するコントローラのような複雑な働きをするオブジェクトの役割りを理解するのに非常に便利である。本文で述べるオブジェクトトレーサは次のような特長を持っている。

- ほとんど全てのオブジェクトをトレースの対象にでき、またほとんど全てのメッセージをトレースできる。
- トレースを指示したオブジェクトと同じクラスの他のインスタンスに影響を与えない。そしてトレース中のオブジェクトが動かない限りパフォーマンスの低下を引き起こさず、また副作用もない。

ＯｂｊｅｃｔＴｒａｃｅｒ

Kiyoki OHKUBO

Research & Development Center, PFU limited
1841-1, Tsuruma 19-Gou, Machida-Shi, Tokyo 194, Japan

The ObjectTracer is an object-oriented tracer which traces how the data are manipulated. This tool is implemented in Smalltalk, and the target of trace is an object user designated in the Smalltalk virtual image. This tool takes advantages of the significant features of Smalltalk, i.e. all things are object and uses message passing in everywhere to the extreme. The approach is completely different than the stepper which is invoked from the debugger, but utilizies doesNotUnderstand: mechanism and only affects the objects which are tracing, so there doesn't occur performance degradation but messages are actually sent to the tracing object. The target of trace may be any object except the two categories of objects; one of them are the unique object such instances as SmallInteger and Symbol, and the other is an object which constructs tracer itself. With regard to message, almost all messages can be traced excepting only four; these are methods constructing tracer itself and executed directly. In this paper I present the technical details and the source codes are also included in the appendix.

１．　オブジェクトトレーサの基本方式

Smalltalk システムにはステッパがあり、実行の流れをソースプログラムを表示させながら追って行ける。ステッパの核は、バイトコードという Smalltalk 向きの機械語を Smalltalk 自体が解釈実行する部分である。この機能を使ってもトレーサの実現は可能であろうが、普段ハードウェアの機械語で実行しているバイトコードを Smalltalk で実行するため２桁程度の速度低下が起こるという難点がある。Lisp ではインタープリタで実行する場合は、apply フックや eval フックを使うことができる。またコンパイルしてある場合は、関数はシンボルの関数セルに入っているので、トレースしたい関数を一時的にトレース用の関数で置き換えて、引数などを出力した後、元の関数を起動すればよい。

ここで述べるオブジェクトトレーサの基本的方式は、トレースを指示されたオブジェクトをしばらくの間『偽オブジェクトと置き換え』、『その偽オブジェクトがメッセージをインターセプトし』トレース情報を出力後、本来実行されるべきメソッドを起動する。Smalltalk では、『偽オブジェクトと置き換え』るメカニズムは、become: で実現できる。また、『その偽オブジェクトがメッセージをインターセプトし』という箇所は、オブジェクトがメッセージを理解できないとバーチャルマシンがそのオブジェクトに doesNotUnderstand: というメッセージを再送してくれるので、これを利用すればよい。あくまで偽オブジェクトであるから看破られる可能性はあるが、めったに看破られないのが良い偽オブジェクトである。ここで簡単に become: と doesNotUnderstand: の働きをふりかえってみる。

（1）　become:

become: は　「object1 become: object2」　のように使い、object1 と object2 のオブジェクトポインタを入れ換えるために使う。オブジェクトポインタとは、オブジェクトを指す単一のポインタであり、それぞれのオブジェクトは固有のオブジェクトポインタで代表される。従って、上のようにオブジェクトポインタを入れ換えると、それまで object1 を参照していた全てのオブジェクトは object2 を参照するようになり、逆に object2 を参照していた全てのオブジェクトは object1 を参照するようになる〔Goldberg83, p.247〕。

（2）　doesNotUnderstand:

いま、クラス Object のサブクラスにＡが、そしてＡのサブクラスにＢがあり、ｂというクラスＢのインスタンスがあるとしよう。オブジェクトｂにメッセージｍを送ると、B、A、Object という順序でｍという名前のメソッドが捜される。もしクラス Object まで行ってもそのメソッドが無いと、今度は、ｍを引数として doesNotUnderstand: というメッセージがオブジェクトｂに送られる。通常 doesNotUnderstand: というメソッドはユーザプログラムでは定義されていないので、クラス Object まで行ってそこにある doesNotUnderstand: を実行しノーティファイヤというウィンドウ

をオープンする〔Goldberg83, p.589〕。

2. オブジェクトトレーサの仕様

トレースをしたいオブジェクトをｘとすると、『x asXObject』のように asXObject というメッセージを送ると、そのオブジェクトに送られてくるほとんど全てのメッセージについてトレース情報をシステムトランスクリプトに表示する。例外は doesNotUnderstand:、instVarNames、allInstVarNames、perform:withArguments: である。これらは偽オブジェクトが元々知っているメッセージなので直接実行されるのでトレースされない。トレース情報は次の通りである（実行例は附録の図を参照）。

メッセージ　レシーバのクラス　（メッセージによって起動されるメソッドの属するクラス）　<< メッセージを送ったメソッドの名前　センダーのクラス　（メッセージを送ったメソッドの属するクラス）

() の部分はクラス DefaultObject のメソッド doesNotUnderstand: の最初で、verbose という変数に true を設定すれば出力される。センダーはメッセージを送ったオブジェクトで、レシーバはメッセージを受け取ったオブジェクトである。『レシーバのクラス』は asXObject されたオブジェクトのクラス名が出力されるが、self または super で自分にメッセージを送った場合は、self または super と出力される。『メッセージによって起動されるメソッドの属するクラス』は、『レシーバのクラス』またはそのスーパークラスであり、『メッセージを送ったメソッドの属するクラス』は『センダーのクラス』またはそのスーパークラスである。トレースしていたオブジェクトを元に戻すには、『x asNormalObject』を実行すればよい。

最初、偽オブジェクトは doesNotUnderstand:、instVarNames、allInstVarNames、perform:withArguments: という４つのメソッドしか知らないが、新しいメッセージが送られてくる度に元のオブジェクトが知っていたメソッドをコピーしてきて、だんだんメソッドを覚えて行く。これらのメソッドはブラウザを通して見れるようにしている。これらは、XObjects というクラスカテゴリの元に作られる（附録の図参照）。

このオブジェクトトレーサでトレースできないオブジェクトも少しある。一つはユニークなオブジュクトであり、もう一つは Transcript や Display などトレーサ自体がメッセージを送るオブジェクトである。ユニークなオブジェクトには、SmallInteger のインスタンス、シンボル、true、false、nil などがある。ユニークなオブジェクトは偽オブジェクトが作れないのでトレースできない。ユニークなオブジェクトであるかどうかは、shallowCopy を実行しても元のオブジェクトを返すかどうかで調べられる。

3. オブジェクトトレーサの構成

偽オブジェクトのクラス名は元のオブジェクトのクラス名の前に X- を付けたものになる。これは asXObject で自動的に行なわれる。そして偽オブジェクトの共通のスーパークラスとして DefaultObject を定義し、トレーサの核であるメソッド doeNot Understand: を DefaultObject に用意する。クラス DefaultObject を定義する時は、一時的にクラス Object をスーパークラスにするが、一旦定義できるとクラスの初期化でスーパークラスを nil にする (DefaultObject のクラスメソッド initialize を参照) 。これは、もし偽オブジェクトのクラスのスーパークラスにクラス Object があると、Object が持っているメソッドは直接実行されトレースされないからである。Smalltalk ではクラス Object が、クラス Object 自身を除く全てのクラスのスーパークラスになっている。しかしクラス DefalutObject のスーパークラスは、クラス Object と同じく nil である。従って DefaultObject は通常の Smalltalk のクラスオブジェクトではない。例えばブラウザを使ってクラス DefalutObject を削除することができない。削除するには「superclass ← Object」を実行してスーパークラスを Object にする必要がある。オブジェトトレーサを実現するために次のようなメソッドを用意している。

(1) 〈asXObject〉 on Object
通常のオブジェクトを、トレース情報を出力する偽オブジェクトに変換する。まず元のオブジェクトのクラス名をAとするとそのサブクラス X－A を、新しいインスタンス変数を追加しないで作る。クラス X－A にはインスタンス変数を返すためのメソッド instVarNames と allInstVarNames をダイナミックに定義する。この2つのメソッドはトレース機能には直接関係ないが、クラスX－Aにトレース中に動いたメソッドをコピーしてくるので、それらのメソッドをブラウザを通して見れるようにするために定義する。次にそのインスタンスａを作り、元のオブジェクトの内容をａにコピーする。このコピーはクラス Object に定義されている shallowCopy と同じである。そしてクラスX－Aのスーパークラスを DefaultObject に変更する。最後に become: を使ってａを元のオブジェクトと入れかえる。Smalltalk ではオブジェクトのクラスは変えられないが、クラスのスーパークラスは変えられるので、このようにして偽オブジェクトを作る。

(2) 〈perform: aSelector withArguments: anArray〉 on DeaultObject
このメソッドはクラス Object の持っているメソッドをそのままコピーしたものであり、メッセージ aSelector に引数 anArray を付けて送る。Lisp の apply と同様な働きをする。偽オブジェクトの中から本来のメソッドを起動する為に使う。

(3) 〈doesNotUnderstand: aMessage〉 on DefalutObject
このメソッドは実行すべきメッセージを引数 aMessage に渡される。このメソッドの中の一時変数は次のような意味を持つ。

- receiverClass は元のオブジェクトのクラスである。これは偽オブジェクトの

クラス名 X－A の頭の X－ を除けてクラス名を求め、グローバル変数を持っているディクショナリ Smalltalk から求めることができる。

・ receiverMethodClass は実際に起動すべきメソッドが属するクラスである。これは aMessage のセレクタと receiverClass からメソッドの記述を求め、そのメソッドがどのクラスに属しているかを調べて求める。この部分はメソッドサーチに相当する。注意しなければならないのは、super にメッセージを送っている場合で、この時はメソッドサーチを receiverClass のスーパークラスから開始しなければならない。super へのメッセージセンドであるかどうかを調べるためにここでは呼出し元のメソッドのバイトコードを調べている。receiverMethodName は receiverClass と同じであるか、またはそのスーパークラスである。

・ senderClassName は、現在実行中のメソッドの結果を待っているオブジェクトのクラスである。Smalltalk では現在実行中のコンテキストは thisContext という擬似変数にはいっていて、これも通常のオブジェクトと同じ扱いが可能である。そこで senderClassName は 「thisContext sender receiver class name」により求められる。

・ senderMethodClassName は、メッセージを送ったメソッドの所属するクラスであり、senderClassName と同じであるか、またはそのスーパークラスである。まず「thisContext sender method」でメッセージを送ったメソッド求め、先程と同様なメソッドサーチにより、そのメソッドが属するクラスを求める。

これらの情報を出力した後、実際に起動すべきメソッドを tryCopyingCodeFor: でコピー（実際はコンパイル）する。このメソッドを呼ぶ時の引数は、「クラス名．セレクタ名」である。このメソッドはこの名前をクラス名とセレクタ名に分解して、そのクラスの中のそのセレクタを持ったメソッドをコンパイルしてメソッドを定義してくれる。tryCopyingCodeFor: に指定する名前のクラス名の部分は現在実行中のオブジェクトのクラスのどれかのスーパークラスの名前でなければならないので、クラスX－Aのスーパークラスを superclass: により一時的に本来実行すべきメソッドの属するクラスに換えている。このメソッドは現在実行中のオブジェクト、即ち偽オブジェクトのクラスに定義される。ただしメソッドの名前は「クラス名．セレクタ名」なので、偽オブジェクトに再び今インターセプトした名前のメッセージが送られても doesNotUnderstand: が呼ばれるようにしている。しかし以前にインターセプトした名前のメッセージが来た時はコピーはしないですむようにしている。こうしてメソッドを偽オブジェクトのクラスに作った後、スーパークラスを DefaultClass に戻し、perform:withArguments: を使ってコピーしたメソッドを実行する。

（4） 〈asNormalObject〉 on DefaultObject

このメソッドは asXObject の逆で、偽オブジェクトを本来のオブジェクトに戻す。

（5） 〈initialize〉 on DefalutObject class

Object のサブクラスから DefaultObject を削除し、DefaultObject のスーパークラ

スを nil にする。クラスメソッド initialize は fileIn を使ってファイルからロードする場合は自動的に実行されるが、ブラウザを使ってプログラムを作った場合は、プログラマが doIt しなければならない。

4． 移植上の注意

このプログラムを移植または拡張しようとする人は次の点に注意しなければならない。まず DefaultObject のスーパークラスは nil なので通常の Smalltalk の世界の常識が通用しない場合がある。例えば doesNotUnderstand: のなかでエラーになると shift-ctl-C さえもきかない場合がある。このような問題を避けるためには、DefaultObject のスーパークラスを Object のままにしておき、安全に動くことを確認してから nil にするとよい。このプログラムは Techtronix の Smalltalk で実行する場合はこのままでよいが、Parc Place Systems の Smalltalk (VI2.2) で実行する場合は次の点を修正しなければならない。

- asXObject の中間あたりにある２つの「CompiledMethod nullSourceDescriptor」を「#(0 0 0)」に変更する。
- doesNotUnderstand: の中間あたりにある「context method instructions」を「context method」に変更する。
- DefaultObject のインスタンスメソッドとして、Object に定義されている class をコピーする。
- StandardSystemController のインスタンスメソッド intersectsDisplayBox: aController の最初に「view isNil ifTrue: [↑false] .」を挿入する。

このトレーサが使えないオブジェクト、すなわち既にトレース中のオブジェクトとユニークなオブジェクトは、このトレーサがチェックしてはじいている。しかしトレーサが情報を出力するために使っているオブジェクト、すなわち Transcript や Display もトレースの対象にしてはならない。もしこれらを対象にするとトレーサがトレーサを呼ぶことになり無限ループに陥ってしまう。同様にこのトレーサを拡張しようとして、トレーサから呼ばれたメソッドの中からトレース中のオブジェクトにメッセージを送ってはならない。このツールはインスペクタのメニューで起動できるようにしておくと便利である。

参考文献

[Goldberg83] Goldberg,A. and Robson,D.: Smalltalk-80: The Language and its Implementation, Addison-Wesley, 1983

Object

Instance Protocols For: XN object conversion

asXObject

```
    | className aClass collection instVarNames index newObject |
    ('X-*' match: self class name)
        ifTrue: [self error: 'Cyclically XObjectify not allowed'].
    self == self shallowCopy ifTrue: [self error: 'Unique object can"t XObjectify'].
    className ← ('X-' , self class name printString) asSymbol.
    aClass ← Smalltalk at: className ifAbsent: [].
    aClass isNil
        ifTrue:
            [aClass ← self class
                        subclass: className
                        instanceVariableNames: ''
                        classVariableNames: ''
                        poolDictionaries: ''
                        category: 'XObjects'.
            collection ← self class allSuperclasses reverse.
            collection add: self class.
            instVarNames ← '#('.
            collection do: [:eachClass |
                eachClass instVarNames do: [:eachInstVar |
                    instVarNames ← instVarNames , '"' , eachInstVar, '" ']].
            aClass class
                compile: 'allInstVarNames
↑ ' , instVarNames , ')'
                notifying: nil
                trailer: CompiledMethod nullSourceDescriptor.
            aClass class
                compile: 'instVarNames
↑ ' , instVarNames , ')'
                notifying: nil
                trailer: CompiledMethod nullSourceDescriptor.
"in ParcPlace St [CompiledMethod nullSourceDescriptor] above two methods should be replaced
by [#(0 0 0)]"
            collection do: [:eachClass |
                eachClass classPool keys do: [:eachKey |
                    aClass addClassVarName: eachKey asString]].
            collection ← self class allSuperclasses.
            collection add: self class.
            collection do: [:eachClass |
                eachClass classPool keys do: [:eachKey |
                    aClass classPool at: eachKey put: (eachClass classPool at: eachKey)]]].
    aClass isVariable
        ifTrue:
            [index ← self basicSize.
            newObject ← aClass basicNew: index.
            [index > 0]
                whileTrue:
                    [newObject basicAt: index put: (self basicAt: index).
                    index ← index - 1]]
        ifFalse: [newObject ← aClass basicNew].
    index ← aClass instSize.
    [index > 0]
        whileTrue:
            [newObject instVarAt: index put: (self instVarAt: index).
            index ← index - 1].
    self class removeSubclass: aClass.
    aClass superclass: DefaultObject.
    self become: newObject
```

DefaultObject

class name	**DefaultObject**
superclass	**Object**
instance variable names	*none*
class variable names	*none*
pool dictionaries	*none*
category	**Kernel-Objects**

Instance Protocols For: message handling

perform: aSelector withArguments: anArray

```
    "copied from Object"
    < primitive: 84 >
```

Instance Protocols For: error handling

doesNotUnderstand: aMessage

```
    | verbose spaces context receiverClass receiverClassName receiverMethodClass senderMethodName
    senderMethodClassName senderClassName instructions newMessage |
    verbose ← false.
    spaces ← ''.
    context ← thisContext sender.
    [context notNil]
        whileTrue:
            [spaces ← spaces , ' '.
            context ← context sender].
    context ← thisContext sender.
    receiverClass ← Smalltalk at: (self class name copyFrom: 3 to: self class name size) asSymbol.
    senderMethodName ← context receiver class
                                    selectorAtMethod: context method
                                    setClass: [:senderMethodClass | senderMethodClass].
    senderMethodName isCompound
        ifTrue:
            [senderMethodClassName ← senderMethodName classPart.
            senderMethodName ← senderMethodName selectorPart].
    instructions ← context method instructions.
"in ParcPlace St [context method instructions] above should be replaced
by [context method]"
    context receiver == self
        ifTrue: [((context pc > 2 and: [(instructions at: context pc - 2)
                = 133])
            or: [context pc > 3 and: [(instructions at: context pc - 3)
                    = 134]])
            ifTrue:
                [receiverClassName ← 'super'.
                receiverClass ← (Smalltalk at: senderMethodClassName) superclass]
            ifFalse: [receiverClassName ← 'self']]
        ifFalse: [receiverClassName ← receiverClass name printString].
    senderMethodClassName isNil ifTrue: [senderMethodClassName ← senderMethodClass name].
    receiverMethodClass ← (receiverClass methodDescriptionAt: aMessage selector) whichClass.
    Transcript cr; show: spaces , aMessage selector printString , ' ' , receiverClassName.
    verbose & (receiverMethodClass ~ = receiverClass) ifTrue:
        [Transcript show: '(' , receiverMethodClass printString , ')'].
    senderClassName ← context receiver class name.
    (context receiver class superclass = DefaultObject) ifTrue:
        [senderClassName ←
            (senderClassName copyFrom: 3 to: senderClassName size) asSymbol].
    Transcript show: ' < < ' , senderMethodName printString , ' ' , senderClassName printString.
    verbose & (senderMethodClassName ~ = senderClassName) ifTrue:
        [Transcript show: '(' , senderMethodClassName printString , ')'].
    self class superclass: receiverMethodClass.
    newMessage ← (receiverMethodClass printString , '.' , aMessage selector printString) asSymbol.
    (self class selectors includes: newMessage)
        ifFalse:
            [(self class tryCopyingCodeFor: newMessage)
                == #OK ifFalse: [self halt].
            self class organization classify: newMessage under: receiverMethodClass printString].
    self class superclass: DefaultObject.
    ↑ self perform: newMessage withArguments: aMessage arguments
```

Instance Protocols For: XN object conversion

asNormalObject

```
    | originalClassName i aClass index newObject |
    ('X-*' match: self class name)
        ifFalse: [self error: 'It"s not an XObject'].
    originalClassName ← self class name copyFrom: 3 to: self class name size.
    aClass ← Smalltalk at: originalClassName asSymbol
                    ifAbsent: [self error: 'Original class missing'].
    aClass isVariable
        ifTrue:
            [index ← self basicSize.
            newObject ← aClass basicNew: index.
```

DefaultObject

```
            [index > 0]
                whileTrue:
                    [newObject basicAt: index put: (self basicAt: index).
                    index ← index - 1]]
        ifFalse: [newObject ← aClass basicNew].
    index ← aClass instSize.
    [index > 0]
        whileTrue:
            [newObject instVarAt: index put: (self instVarAt: index).
            index ← index - 1].
    self become: newObject

"- - - - - - - - - - - - - - - - - - -"

DefaultObject class
instanceVariableNames: ''
```

Class Protocols For: class initialization

initialize

```
    "DefaultObject initialize"
    Object removeSubclass: self.
    superclass ← nil
```

DefaultObject initialize **doIt**