
Reflection II

doesNotUnderstand:

Contexts

exception handling

concurrency

Changing Method Lookup

If an object does not have or inherit a method for a message then the interpreter will send the "doesNotUnderstand:" message to it.

You can redefine doesNotUnderstand: in any class. The argument is an object of class Message that knows the selector and the arguments.

doesNotUnderstand:

Trick: make the superclass of a new class be nil. Then the class doesn't inherit any methods from Object.

Uses:

- persistent objects
- atomic objects
- distributed objects

Contexts

- context stores temporaries, method arguments, program counter, and return address
- thisContext gives method its own context
- used by debugger, exception handling, processes, backtracking

Contexts

<i>PolylineFigure</i> <i>displayOn:</i>
<i>CompositeFigure</i> <i>displayOn:</i>
<i>Drawing</i> <i>displayOn:</i>

Handling Errors

Meaning of error often depends on how it is being used.

Possible error: dictionary does not contain a key.

aDictionary

at: x

ifAbsent: [aDictionary at: x put: nil]

How to Handle Errors

- 1) Pass in block to handle error
- 2) Provide an exception system

Key Idea

Program handles a *particular* error signal.

[object doWork]

on: aSignal

do: [:theException | ...]

Server raises *the same* signal.

aSignal raise

Exception Handling

[x / y]

on: Number divisionByZeroSignal

do: [:theException | theException returnWith:
0]

/ signals an error by

Number divisionByZeroSignal raiseSignal

Exception Handling

The "raiseSignal" method of Signal creates an Exception, finds a on:do: message on the Signal, and evaluates the second argument with the Exception as the argument.

One class of Signals, many classes of Exceptions.

Each Signal knows the class of Exception that it creates.

Signals and Exceptions

Signal describes type of error,
Exception describes a particular
instance of error.

Example of “Type Object” pattern.

Exceptions have an error string, an
originator, and an argument.

Signal

raiseSignal

raiseSignal: aMessageString

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

773

Exception Protocol

resume: - return from message that
signaled exception

retryUsing: - abort exception handler and
reevaluate its exception block

return: - return from block protected by
active exception handler

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

774

Multiple Handlers

Exception travels down stack of contexts, looking for a handler to the Signal. It evaluates the handle block of the first match.

S raise
...
on: S do:
...
on: S: do:

Exception Protocol

reject will cause an Exception to look for the next handler to the Signal.

return will throw away all contexts above the handler.

Signal Hierarchy

Parent of a Signal is another Signal that will trap all exceptions raised for it.

Object errorSignal is the parent or grandparent of all other signals, so it will catch any error.

Signal Hierarchy

Signal new creates a new signal with Object errorSignal as a parent.

aSignal newSignal creates a new signal with aSignal as a parent.

Unwind Protection

Problem: program can get blown away by an exception while it is in the middle of making delicate changes. This is a problem with returns in general, even without exceptions.

Unwind Protection

Solution:

```
[self dangerousCode] ifCurtailed: [self  
  cleanUp]
```

The cleanup block is used if the execution stack is cut back for any reason.

Unwind Protection

ensure: evaluates the cleanup block after the receiver is evaluated.

Semaphore critical: aBlock
self wait.
aBlock ensure: [self signal]

An example

processFile: aFile
"Read a catalog card from the first comment.
Trap and report all errors."
aFile isReadable ifFalse: [^self].
[self processWithErrorsFile: aFile]
on: ErrorSignal
do: [:ex | ...]

The Handle Block

```
[ :ex |  
  Transcript show: ex errorString.  
  Transcript show: ' for ' , aFile asString.  
  Transcript cr.  
  Dialog confirm: ex errorString.  
  ex return ]
```

processWithErrorsFile: aFile

"Read a catalog card from the first comment."

```
| aStream string document |  
aStream := aFile readStream.  
[...] ensure: [aStream close]
```

If we encounter an error

[Transcript show: 'bad file ', aFile
asString.

^Transcript cr].

Also, system generates errors for files that
are not readable.

Naming Signals

Signals are all instances of same class.

Signals are distinguished by storing them in
class variables.

Make an accessing message for each
signal.

Look at senders to find where signal is
raised and handled.

Signals from Object

InformationSignal
ControlInterruptedSignal
UserInterruptSignal
HaltSignal
NotifySignal

More Signals from Object

ErrorSignal
NotFoundSignal
IndexNotFoundSignal
SubscriptOutOfBoundsSignal
NonIntegerIndexSignal
SubclassResponsibilitySignal
MessageNotUnderstoodSignal

Subclasses of Exception

105 subclasses of Exception.

Major categories are:

Error - can't be resumed

Notification - can be resumed

Halt

halt

"This is a simple message to use for inserting breakpoints during debugging.

The debugger is opened by raising a signal. This gives a chance to restore invariants related to multiple processes."

Halt

Object haltSignal

raiseRequestWith: thisContext

errorString: 'Halt encountered.'

Creating Processes

[self doIt] newProcess

[self doIt] fork

Operations on processes

resume - put process on run queue
suspend - take process off run queue
terminate - kill process
priority priority: - 1 (lowest) to 100
- ProcessorScheduler defines priorities

Semaphores

Semaphore new - initializes with no
"signals"

aSemaphore wait

aSemaphore signal

aSemaphore critical: [$x := x + 1$]

Promise

next

| result |

mutex critical:

[result := [self computeNextWith:
promise value] promise.

promise := result]

^promise

Delay

Delay forSeconds: 1

Delay forMilliseconds: 20

Delay methods:

wait - suspend process until delay is
finished

startup - start the clock running

disable - turn off the clock

SharedQueue

Collections and Streams are not safe when you have multiple processes.

SharedQueue has a semaphore and a collection to hold its contents.

Protocol

next

nextPut

next

| isEmpty value |

readSynch wait.

^[accessProtect critical:

 [(isEmpty := contents isEmpty)

 ifTrue: [nil]

 ifFalse: [contents removeFirst]]]

valueUninterruptably.

SharedQueue

nextPut: value

[accessProtect critical: [contents
addLast: value]] valueUninterruptably.

readSynch signal.

^value

Processes in VisualWorks GUI

Put "self poll" in every loop in a controller.

controlLoop

[self poll.

self isControlActive]

whileTrue:

[self controlActivity]