Class Thread < Object

Thread encapsulates the behavior of a thread of execution, including the main thread of the Ruby script. See the tutorial in Chapter 12, beginning on page 184.

In the descriptions that follow, the parameter *symbol* refers to a symbol, which is either a quoted string or a Symbol (such as :name).

Class methods

abort_on_exception

Thread.abort_on_exception → true or false

Returns the status of the global "abort on exception" condition. The default is false. When set to true or if the global $DEBUG flag is true (perhaps because the command-line option -d was specified), all threads will abort (the process will exit(0)) if an exception is raised in any thread. See also Thread.abort_on_exception=.

abort_on_exception=

Thread.abort_on_exception= *bool*→ true or false

When set to true, all threads will abort if an exception is raised. Returns the new state.

```
Thread.abort_on_exception = true
t1 = Thread.new do
  puts  "In new thread"
  raise "Exception from thread"
end
sleep(1)
puts "not reached"
```

produces:

```
In new thread
prog.rb:4:in `block in <main>': Exception from thread (RuntimeError)
```

current

Thread.current → *thread*

Returns the currently executing thread.

```
Thread.current   # =>   #<Thread:0x0ac684 run>
```

exclusive

Thread.exclusive { *block* } → *obj*

1.9

Executes the block and returns whatever the block returns. Internally uses a Mutex so that only one thread can be executing code under control of Thread.exclusive at a time.

exit

Thread.exit

Terminates the currently running thread and schedules another thread to be run. If this thread is already marked to be killed, exit returns the Thread. If this is the main thread, or the last thread, exits the process.

fork

Thread.fork { *block* } → *thread*

Synonym for Thread.start.

kill

Thread.kill(*thread*)

Causes the given thread to exit (see Thread.exit).

```
count = 0
a = Thread.new { loop { count += 1 } }
sleep(0.1)       # =>  0
Thread.kill(a)   # =>  #<Thread:0x0a3764 aborting>
count            # =>  967979
# give it time to die...
sleep 0.01
a.alive?         # =>  false
```

list

Thread.list → *array*

Returns an array of Thread objects for all threads that are either runnable or stopped.

```
Thread.new { sleep(200) }
Thread.new { 1000000.times {|i| i*i } }
Thread.new { Thread.stop }
Thread.list.each {|thr| p thr }
```

produces:

```
#<Thread:0x0ac684 run>
#<Thread:0x0a3b88 sleep>
#<Thread:0x0a3a70 run>
#<Thread:0x0a39a8 sleep>
```

main

Thread.main → *thread*

Returns the main thread for the process.

```
Thread.main   # =>   #<Thread:0x0ac684 run>
```

new

Thread.new(⟨ *arg* ⟩*) {| *args* | *block* } → *thread*

Creates and runs a new thread to execute the instructions given in *block*. Any arguments passed to Thread.new are passed into the block.

```
x = Thread.new { sleep 0.1; print "x"; print "y"; print "z" }
a = Thread.new { print "a"; print "b"; sleep 0.2; print "c" }
x.join; a.join # wait for threads to finish
```

produces:

```
abxyzc
```

pass

Thread.pass

Invokes the thread scheduler to pass execution to another thread.

```
a = Thread.new { print "a"; Thread.pass; print "b" }
b = Thread.new { print "x"; Thread.pass; print "y" }
a.join; b.join
```

produces:

```
axby
```

start Thread.start(⟨ *args* ⟩*) {| *args* | *block* } → *thread*

Basically the same as Thread.new. However, if class Thread is subclassed, then calling start in that subclass will not invoke the subclass's initialize method.

stop Thread.stop

Stops execution of the current thread, putting it into a "sleep" state, and schedules execution of another thread. Resets the "critical" condition to false.

```
a = Thread.new { print "a"; Thread.stop; print "c" }
Thread.pass
print "b"
a.run
a.join
```

produces:

```
a
b
c
```

Instance methods

[ ] *thr*[*symbol*] → *obj* or nil

Attribute Reference—Returns the value of a thread-local variable, using either a symbol or a string name. If the specified variable does not exist, returns nil.

```
a = Thread.new { Thread.current["name"] = "A"; Thread.stop }
b = Thread.new { Thread.current[:name]  = "B"; Thread.stop }
c = Thread.new { Thread.current["name"] = "C"; Thread.stop }
Thread.list.each {|x| puts "#{x.inspect}: #{x[:name]}" }
```

produces:

```
#<Thread:0x0ac684 run>:
#<Thread:0x0a2ecc run>:
#<Thread:0x0a2e7c run>:
#<Thread:0x0a2cb0 run>:
```

[ ]= *thr*[*symbol*] = *obj*→ *obj*

Attribute Assignment—Sets or creates the value of a thread-local variable, using either a symbol or a string. See also Thread#[].

abort_on_exception *thr*.abort_on_exception → true or false

Returns the status of the thread-local "abort on exception" condition for *thr*. The default is false. See also Thread.abort_on_exception=.

abort_on_exception= *thr*.abort_on_exception= true or false→ true or false

When set to true, causes all threads (including the main program) to abort if an exception is raised in *thr*. The process will effectively exit(0).

alive?

thr.alive? → true or false

Returns true if *thr* is running or sleeping.

```
thr = Thread.new { }
thr.join                # =>  #<Thread:0x0a49ac dead>
Thread.current.alive?   # =>  true
thr.alive?              # =>  false
```

exit

thr.exit → *thr* or nil

Terminates *thr* and schedules another thread to be run. If this thread is already marked to be killed, exit returns the Thread. If this is the main thread, or the last thread, exits the process.

group

thr.group → *thread_group*

1.9

Returns the ThreadGroup owning *thr*, or nil.

```
thread = Thread.new { sleep 99 }
Thread.current.group.list   # =>  [#<Thread:0x0ac684 run>,
                                   #<Thread:0x0a3764 run>]
new_group = ThreadGroup.new
thread.group.list           # =>  [#<Thread:0x0ac684 run>,
                                   #<Thread:0x0a3764 run>]
new_group.add(thread)
thread.group.list           # =>  [#<Thread:0x0a3764 run>]
Thread.current.group.list   # =>  [#<Thread:0x0ac684 run>]
```

join

thr.join → *thr*
thr.join(*limit*) → *thr*

1.9

The calling thread will suspend execution and run *thr*. Does not return until *thr* exits or until *limit* seconds have passed. If the time limit expires, nil will be returned; otherwise, *thr* is returned.

Any threads not joined will be killed when the main program exits. If *thr* had previously raised an exception and the abort_on_exception and $DEBUG flags are not set (so the exception has not yet been processed), it will be processed at this time.

```
a = Thread.new { print "a"; sleep(10); print "b"; print "c" }
x = Thread.new { print "x"; Thread.pass; print "y"; print "z" }
x.join # Let x thread finish, a will be killed on exit.
```

produces:

```
axyz
```

The following example illustrates the *limit* parameter.

```
y = Thread.new { loop { sleep 0.1; print "tick...\n" }}
y.join(0.25)
puts "Gave up waiting..."
```

produces:

```
tick...
tick...
Gave up waiting...
```

keys

thr.keys → *array*

1.9

Returns an array of the names of the thread-local variables (as symbols).

```
thr = Thread.new do
  Thread.current[:cat] = 'meow'
  Thread.current["dog"] = 'woof'
end
thr.join   # =>   #<Thread:0x0a44e8 dead>
thr.keys   # =>   [:cat, :dog]
```

key?

thr.key?(*symbol*) → true or false

Returns true if the given string (or symbol) exists as a thread-local variable.

```
me = Thread.current
me[:oliver] = "a"
me.key?(:oliver)    # =>   true
me.key?(:stanley)   # =>   false
```

kill

thr.kill

Synonym for Thread#exit.

priority

thr.priority → *int*

Returns the priority of *thr*. The default is zero; higher-priority threads will run before lower-priority threads.

```
Thread.current.priority   # =>   0
```

priority=

thr.priority= *int* → *thr*

Sets the priority of *thr* to *integer*. Higher-priority threads will run before lower-priority threads. If you find yourself messing with thread priorities to get things to work, you're doing something wrong.

```
count_high = count_low = 0
Thread.new do
  Thread.current.priority = 1
  loop { count_high += 1 }
end
Thread.new do
  Thread.current.priority = -1
  loop { count_low += 1 }
end

sleep 1
count_high   # =>   7504330
count_low    # =>   1861069
```

raise

thr.raise
thr.raise(*message*)
thr.raise(*exception* ⟨ , *message* ⟨ , *array* ⟩ ⟩)

1.9

Raises an exception (see Kernel.raise on page 575 for details) from *thr*. The caller does not have to be *thr*.

```
Thread.abort_on_exception = true
a = Thread.new { sleep(200) }
a.raise("Gotcha")
a.join
```

produces:

```
prog.rb:2:in `sleep': Gotcha (RuntimeError)
from /tmp/prog.rb:2:in `block in <main>'
```

run

thr.run → *thr*

Wakes up *thr*, making it eligible for scheduling. If not in a critical section, then invokes the scheduler.

```
a = Thread.new { puts "a"; Thread.stop; puts "c" }
Thread.pass
puts "Got here"
a.run
a.join
```

produces:

```
a
b
c
```

safe_level

thr.safe_level → *int*

Returns the safe level in effect for *thr*. Setting thread-local safe levels can help when implementing sandboxes that run insecure code.

```
thr = Thread.new { $SAFE = 3; sleep }
Thread.current.safe_level   # =>   0
thr.safe_level              # =>   0
```

status

thr.status → *string*, false or nil

Returns the status of *thr*: sleep if *thr* is sleeping or waiting on I/O, run if *thr* is executing, aborting if *thr* is aborting, false if *thr* terminated normally, and nil if *thr* terminated with an exception.

```
a = Thread.new { raise("die now") }
b = Thread.new { Thread.stop }
c = Thread.new { Thread.exit }
a.status                # =>  nil
b.status                # =>  "sleep"
c.status                # =>  false
Thread.current.status   # =>  "run"
```

stop?

thr.stop? → true or false

Returns true if *thr* is dead or sleeping.

```
a = Thread.new { Thread.stop }
b = Thread.current
Thread.pass
a.stop?   # =>   false
b.stop?   # =>   false
```

terminate

thr.terminate

Synonym for Thread#exit.

value

thr.value → *obj*

Waits for *thr* to complete (via Thread#join) and returns its value.

```
a = Thread.new { 2 + 2 }
a.value   # =>   4
```

wakeup

thr.wakeup → *thr*

Marks *thr* as eligible for scheduling (it may still remain blocked on I/O, however). Does not invoke the scheduler (see Thread#run).