

Re: "False exceptions?" (was Re: theme of the week: tools

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-10/0022.html>

From: David Bolen (db3l_at_fitlinxx.com)

Date: 10/01/04

Date: 30 Sep 2004 19:13:52 -0400

"Dan Perl" <danperl@rogers.com> writes:

> (...) *In my experience with sporadic problems like the ones you are
> describing, debuggers are usually not much help, except after the fact, for
> instance debugging a core of a C/C++ program. If the problem is caused by a
> race condition, a debugger can be useless because it affects the scheduling
> of threads. How does the loaded stub work in cases like that? Does it
> affect threads in any way?*

Yes, once the debugger is actually activated (you start up the IDE and it attaches) it's like a normal Python debugger, so it uses the trace hook and will have some impact on runtime. You can finesse this a little by controlling where you import the stub and/or when you connect to the process.

Of course, in my experience just about anything you do to try to work with a subtle threading or race condition bug is just as likely to affect the problem (including adding a new line of code), so you can still hope that the problem will simply exhibit itself as well – even if slightly differently – with the debugger attached. Obviously, no guarantees.

(the "sre" exception)

> *It was the only one. My point was that even just this particular exception
> should happen very often. So even volume, not only significance, is
> relative. I, for one, was not comfortable ignoring an exception, especially
> without a very clear and detailed explanation on why the false positive
> happened in the first place and what happens with the exception if I ignore
> it.*

I don't really disagree – it would probably be good if this was a well-defined one that Wing came configured to ignore.

>> (...)

>>> *I would need to first see an example of generating an exception without
>>> knowing it and that exception still being relevant although it is being*

> >> *handled.*
> >
> > *I'd split that into two parts – (a) generating the exception without*
> > *knowing it, and (b) it being relevant even if handled.*
> >
> > *For (a), I can't seem to come up with a good concise example, so maybe*
> > *I'll just agree that you may know about, but not be able to act on it*
> > (...)
> > *Note however, that knowing about it in many cases may only be through*
> > *stderr traceback, so to know it other than manually inspecting it at*
> > *runtime would require trapping stderr in general and logging that*
> > *somewhere, and then having something to recognize tracebacks in that*
> > *log.*
> >
> > *(b) is certainly possible though, although at the risk of overuse I'll*
> > *fall back to wxPython again. In wxPython any exception that occurs in*
> > *an event handler will be handled by the wxPython extension, and is*
> > *thus invisible to top level code.*
> > (...)
> >
> > *You're making a very good case, assuming that the exception is at the root*
> > *of the problem. However, it's hard to say how many such problems are caused*
> > *by an exception. Even if there is an exception, the root problem may be way*
> > *before that and you still have to go and do some normal debugging. So many*
> > *times I've had memory corruption crashing an application in C++ and I had to*
> > *look back at a method that was called WAY BEFORE the crash because that was*
> > *usually corrupting memory at exit.*

Yep, although if you're lucky the form of the corruption at the point of error can help point the way. Also, since pure memory corruption tends to be a rare occurrence in Python (buggy extension modules notwithstanding), even if an exception is due to malformed code earlier on, the actual memory state of the process is generally in good enough shape to go poking around reliably in all your data structures once the exception triggers.

> *So, in your example of a button, my general debugging approach would still*
> *be to put a breakpoint in the event handler of the button, because the root*
> *of the problem can be so many different things.*

Sure, but if the button works sometimes and not others, it's just more of a pain to hit the breakpoint over and over again until it happens to get to a state where the problem would occur.

> *In your experience with Python, what percentage of problems manifested*
> *themselves with exceptions (especially right at the root of the problem) as*
> *opposed to simply flawed logic that gets through without any exceptions?*

I'd have to separate by older and newer code again. The newer code following a TDD pattern definitely has different characteristics than older legacy code written using more traditional methods. The newer

stuff is much less likely to generate either kind of problem except during active development at which point the current test generally points right to the area of potential problem.

But I'd have to say that at least for me, for more traditional methods, I do think there's a higher percentage of problems leading to exceptions than to more subtle failures. I'm not entirely sure why except that I expect bugs that in other environments might lead to subtle mistakes (or harder to track problems like eventual crashes from prior memory corruption) quickly end up in some state where you are trying to interpret any object incorrectly in your code and yield an exception. For example, a wrong pathway not constructing an appropriate object type or contents which is then passed into another function anticipating it. In effect, the "duck typing" approach that is commonly used can yield exceptions in pretty quick order if you start messing up the objects involved in your system.

Note however that I'm not saying all those exceptions require a debugger or anything more than logging them and/or observing them during development.

> *I think we are actually very much agreeing. The main difference between us*
> *is how much weight we put in debugging exceptions versus debugging all kinds*
> *of problems and how much we are willing to put up with in exchange for that*
> *precious advantage in a few cases.*

Yes (to the agreement). And don't let me over-emphasize things, as I'm not necessarily suggesting that the "wait for the exception" is extremely important. It is, however, useful enough to me that I consider supporting it to be a "pro" feature of any Python debugger, and at least for me, yes worth some consequences such as the occasional false positive.

> *You are saying a solid debugger "can be worth its weight in gold". How much*
> *does Wing IDE weigh? ;-)*

Heh – good question. 'Course, with gold where it is at the moment, it probably wouldn't have to weigh too much anyway :-)

-- David