Profile Dis PDB

A brief look at profiling, disassembling and debugging your Python code.

Brett Langdon

- http://brett.is/
- brett@blangdon.com
- Software Engineer – Magnetic Media Online
- Github.com/brettlangdon
- @brett_langdon

Companion Code

- http://github.com/PythonBuffalo/Profile-Dis-PDB

Disassembly

- Manually generating Python Byte Code for your scripts
- Python is a virtual machine
- Byte Code is the Assembly code
- Why do we care?

Hello World Byte Code

```
print “hello world”
```

```
1           0 LOAD_CONST               0 ('hello world')
            3 PRINT_ITEM
            4 PRINT_NEWLINE
            5 LOAD_CONST               1 (None)
            8 RETURN_VALUE
```

Live Demo

Profiling

- Inspect the runtime of your scripts
- Determine where time is most spent
- Helps to identify bottlenecks in your code
- Helps identify areas of optimization

cProfile and profile

- Both are profilers built into standard library
- cProfile is a C extension
- profile is pure Python
- cProfile generally produces less overhead in profiling

```python
# sample.py
import time


def i_am_slow(n):
    time.sleep(0.1)
    return n - 1


def i_am_fast(n):
    return n - 1


def parent(total):
    for n in xrange(total):
        if n % 2 == 0:
            i_am_slow(n)
        else:
            i_am_fast(n)


parent(100)
```

Sample Profiling

```
python -m cProfile sample.py

         153 function calls in 5.055 seconds

   Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.002    0.002    5.055    5.055 sample.py:1(<module>)
     1    0.001    0.001    5.053    5.053 sample.py:13(parent)
    50    0.001    0.000    5.052    0.101 sample.py:4(i_am_slow)
    50    0.000    0.000    0.000    0.000 sample.py:9(i_am_fast)
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
    50    5.051    0.101    5.051    0.101 {time.sleep}
```

pstats

- Library used to make sense of profiling output
- Have cProfile/profile output results to file rather than stdout
- Use pstats to read in results file and manipulate

pstats Sorting Example

```
# sort_stats.py
import pstats

stats = pstats.Stats("stats.out")
stats.strip_dirs().sort_stats("cumulative").print_stats()
```

```
python –m cProfile –o stats.out sample.py
python sort_stats.py

Thu Oct  3 16:47:08 2013    stats.out

         153 function calls in 5.051 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.001    0.001    5.051    5.051 sample.py:1(<module>)
        1    0.001    0.001    5.050    5.050 sample.py:13(parent)
       50    0.001    0.000    5.049    0.101 sample.py:4(i_am_slow)
       50    5.048    0.101    5.048    0.101 {time.sleep}
       50    0.000    0.000    0.000    0.000 sample.py:9(i_am_fast)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Debugging

- Following executions paths of your scripts
- Helps to find bugs in execution of your scripts

Live Demo

Pdb

- The Python Debugger
- Debugger built into Python standard library
- Has the same/similar commands and usage as GDB
- Pdb repl is also just a Python repl

Running the debugger

```
# from cli
python –m pdb simple_code.py

# manually set breakpoint in code
print "before breakpoint"
import pdb
pdb.set_trace()
print "after breakpoint"

python test.py
before breakpoint
> simple_example.py(4)<module>()
-> print "after breakpoint"
(Pdb)
```

Pdb Commands

- l(ist) – show source code for where pdb currently is
- s(tep) – step into the current function call
- n(ext) – move to the next line of execution
- b(reak) <args> - set a breakpoint in the code
- c(ontinue) – continue to next breakpoint or end of script
- h(elp) – show help information

Live Demo

Questions?

Thanks!