

When is StringIO used?

Using StringIO as string buffer is slower than using list as buffer.

When is StringIO used?

```
from io import StringIO

def meth1(string):
    a = []
    for i in range(100):
        a.append(string)
    return ''.join(a)

def meth2(string):
    a = StringIO()
    for i in range(100):
        a.write(string)
    return a.getvalue()

if __name__ == '__main__':
    from timeit import Timer
    string = "This is test string"
    print(Timer("meth1(string)", "from __main__ import meth1, string").timeit())
    print(Timer("meth2(string)", "from __main__ import meth2, string").timeit())
```

Results:

```
16.7872819901
18.7160351276
```

python stringio

edited May 25 '15 at 18:41



nbro

4,541 6 24 66

asked Jan 19 '11 at 9:39



simha

351 1 5 11

1 Do you possibly mean "When" instead of "Where" above? – Lennart Regebro Jan 19 '11 at 10:16

4 Answers

If you measure for speed, you should use `cStringIO` .

From the [docs](#):

The module `cStringIO` provides an interface similar to that of the `StringIO` module. Heavy use of `StringIO.StringIO` objects can be made more efficient by using the function `StringIO()` from this module instead.

But the point of `StringIO` is to be a *file-like object*, for when something expects such and you don't want to use actual files.

Edit: I noticed you use `from io import StringIO` , so you are probably on Python ≥ 3 or at least 2.6. The separate `StringIO` and `cStringIO` are gone in Py3. Not sure what implementation they used to provide the `io.StringIO`. There is `io.BytesIO` too.

edited Jan 19 '11 at 9:59

answered Jan 19 '11 at 9:51



plundra

9,734 3 19 21

Try it with `cStringIO` . Results: List: 17, `cString`: 33. – user225312 Jan 19 '11 at 9:59

2 `io.StringIO` is a C implementation, if that exists on your platform. If not it uses a Python implementation fallback. The reason it's slower is because he is doing something that he doesn't need `StringIO` for in the first place. – Lennart Regebro Jan 19 '11 at 10:14

The main advantage of StringIO is that it can be used where a file was expected. So you can do for example:

```
import sys
import StringIO

out = StringIO.StringIO()

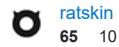
sys.stdout = out

print "hi, I'm going out"

sys.stdout = sys.__stdout__

print out.getvalue()
```

edited May 29 at 15:17



[ratskin](#)
65 10

answered Jan 19 '11 at 9:51



[TryPyPy](#)
4,580 4 25 61

Can it be used with `with` in python 2 ? From what I see here no: bugs.python.org/issue1286 – [Mr_and_Mrs_D](#) Dec 29 '14 at 22:19

@Mr_and_Mrs_D please see <http://bugs.python.org/issue1286#msg176512> which states that it will work from 2.5 up. What more do you want, blood on it? :D – [Mark Lawrence](#) Jul 2 '16 at 22:45

@MarkLawrence: no it won't - reread the comment you linked - you have to roll *your own* context manager – [Mr_and_Mrs_D](#) Jul 3 '16 at 13:19

@Mr_and_Mrs_D the link I gave gives an example of a context manager. You clearly do want blood on it. What did your last skivvy die of, overwork? – [Mark Lawrence](#) Jul 27 '16 at 8:01

Well, I don't know if I would like to call that using it as a "buffer", you are just multiplying a string a 100 times, in two complicated ways. Here is an uncomplicated way:

```
def meth3(string):
    return string * 100
```

If we add that to your test:

```
if __name__ == '__main__':

    from timeit import Timer
    string = "This is test string"
    # Make sure it all does the same:
    assert(meth1(string) == meth3(string))
    assert(meth2(string) == meth3(string))
    print(Timer("meth1(string)", "from __main__ import meth1, string").timeit())
    print(Timer("meth2(string)", "from __main__ import meth2, string").timeit())
    print(Timer("meth3(string)", "from __main__ import meth3, string").timeit())
```

It turns out to be way faster as a bonus:

```
21.0300650597
22.4869811535
0.811429977417
```

If you want to create a bunch of strings, and then join them, `meth1()` is the correct way. There is no point in writing it to StringIO, which is something completely different, namely a string with a file-like stream interface.

answered Jan 19 '11 at 10:10



[Lennart Regebro](#)
85.5k 23 157 211

I know this thread is old, but I stumbled across it through Google - I hope this answer helps others as well.

If your aim is string concatenation then the `"+="` operator is better than both of the options:

```
def meth3(string):
    a = ''
    for i in range(100):
        a += string
    return a
```

Results:

7/31/2017

python - When is StringIO used? - Stack Overflow

17.406924963
157.963402033
13.0571110249

Note that I used Python2.6 and `from StringIO import StringIO` for the second method - which is slowest by far on my machine.

deleted by owner Jun 8 '13 at 21:42

answered Oct 13 '11 at 13:30



johndodo

6,185 6 49 74