Which loop has better performance? Why?

Asked 16 years, 3 months ago Modified 11 years, 4 months ago Viewed 9k times

51

```
String s = "";
for(i=0;i<....){
    s = some Assignment;
}
```

or

```
for(i=0;i<..){
    String s = some Assignment;
}
```

I don't need to use 's' outside the loop ever again. The first option is perhaps better since a new String is not initialized each time. The second however would result in the scope of the variable being limited to the loop itself.

EDIT: In response to Milhous's answer. It'd be pointless to assign the String to a constant within a loop wouldn't it? No, here 'some Assignment' means a changing value got from the list being iterated through.

Also, the question isn't because I'm worried about memory management. Just want to know which is better.

java performance string garbage-collection

Share

Improve this question

Follow

edited Apr 13, 2009 at 20:24

community wiki
7 revs, 6 users 64%
Vivek Kodira

It's not unusual to iterate over a collection of strings specified as literals. For example, the column headings of a table might hard-coded as a String[]. What's important, though, is that the same assignment happens in both cases, and so it doesn't affect the answer. – erickson Sep 21, 2008 at 6:33

Another comment: don't forget, that if you're not going to change the value of `s`, you *should* declare it `final`. Many Java programmers forget that all to often. – Aleksandar Dimitrov Sep 21, 2008 at 9:03

8 Answers

Sorted by: Highest score (default)

109

Limited Scope is Best

Use your second option:

```
for ( ... ) {
  String s = ...;
}
```

Scope Doesn't Affect Performance

If you disassemble code the compiled from each (with the JDK's `javap` tool), you will see that the loop compiles to the exact same JVM instructions in both cases. Note also that Brian R. Bondy's "Option #3" is identical to Option #1. Nothing extra is added or removed from the stack when using the tighter scope, and same data are used on the stack in both cases.

Avoid Premature Initialization

The only difference between the two cases is that, in the first example, the variable `s` is unnecessarily initialized. This is a separate issue from the location of the variable declaration. This adds two wasted instructions (to load a string constant and store it in a stack frame slot). A good static analysis tool will warn you that you are never reading the value you assign to `s`, and a good JIT compiler will probably elide it at runtime.

You could fix this simply by using an empty declaration (i.e., `String s;`), but this is considered bad practice and has another side-effect discussed below.

Often a bogus value like `null` is assigned to a variable simply to hush a compiler error that a variable is read without being initialized. This error can be taken as a hint that the variable scope is too large, and that it is being declared before it is needed to receive a valid value. Empty declarations force you to consider every code path; don't ignore this valuable warning by assigning a bogus value.

Conserve Stack Slots

As mentioned, while the JVM instructions are the same in both cases, there is a subtle side-effect that makes it best, at a JVM level, to use the most limited scope possible. This is visible in the "local variable table" for the method. Consider what happens if you have multiple loops, with the variables declared in unnecessarily large scope:

```
void x(String[] strings, Integer[] integers) {
  String s;
  for (int i = 0; i < strings.length; ++i) {
    s = strings[0];
    ...
  }
  Integer n;
  for (int i = 0; i < integers.length; ++i) {
    n = integers[i];
    ...
  }
}
```

The variables `s` and `n` could be declared inside their respective loops, but since they are not, the compiler uses two "slots" in the stack frame. If they were declared inside the loop, the compiler can reuse the same slot, making the stack frame smaller.

What Really Matters

However, most of these issues are immaterial. A good JIT compiler will see that it is not possible to read the initial value you are wastefully assigning, and optimize the assignment away. Saving a slot here or there isn't going to make or break your application.

The important thing is to make your code readable and easy to maintain, and in that respect, using a limited scope is clearly better. The smaller scope a variable has, the easier it is to comprehend how it is used and what impact any changes to the code will have.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:02

Community Bot

1 • 1

answered Sep 21, 2008 at 6:24

erickson

269k • 59 • 401 • 497

1 I'm still not fully convinced, but I deleted my answer because it was wrong, but for reference of this question. Here is what I had: {//Don't remove braces String s; for(i=0;i<....){ s = some Assignment; } } – Brian R. Bondy Sep 21, 2008 at 13:34

I'd vote up this comment twice if I could. I'd also tag the question "early optimization". – Trenton Oct 11, 2008 at 6:27

What an excellent answer; I'd also vote it up multiple times if I could. – L. Cornelius Dol May 1, 2009 at 6:25

I hadn't played with javap before, so I checked a for loop that gets the date and sets String s to the date.toString. The disassembly I did, showed that the code is different. You can see that s is being set each loop in the inner one. Is this something that the jit compiler might fix? – Philip Tinney Oct 12, 2010 at 17:41

@Philip T. - I'm not sure I understand what you are describing. If you mean that the same value was assigned to the `String` on each iteration of the loop, and you are asking whether

that calculation can be "hoisted", yes, it's possible, but it will depend on the JVM. – erickson Oct 12, 2010 at 17:59

22

In *theory*, it's a waste of resources to declare the string inside the loop. In *practice*, however, both of the snippets you presented will compile down to the same code (declaration outside the loop).

So, if your compiler does any amount of optimization, there's no difference.

Share Improve this answer Follow

edited Jan 24, 2010 at 3:40

answered Sep 21, 2008 at 2:48

Esteban Araya
29.6k ● 25 ● 111 ● 141

The reference is just put inside the stack frame for this method call, right? – Thomas Sep 21, 2008 at 3:08

I think you misinterpreted what 1800 INFORMATION wrote. The immutability of Java strings is irrelevant here: the "some Assignment" will produce a new String each time, whether or not String is immutable. – Thomas Sep 21, 2008 at 3:30

Have updated the Question. Assigning a hard coded string would be illogical. – Vivek Kodira Sep 21, 2008 at 5:52

5 Which theory is it, which tells you that *declaring* a variable wastes resources? – jrudolph Sep 21, 2008 at 13:31

jrudolph: Creating a new empty string creates a new garbage collected object, if I remember my Java correctly. – TraumaPony Sep 21, 2008 at 13:59

17

In general I would choose the second one, because the scope of the 's' variable is limited to the loop. Benefits:

- This is better for the programmer because you don't have to worry about 's' being used again somewhere later in the function
- This is better for the compiler because the scope of the variable is smaller, and so it can potentially do more analysis and optimisation
- This is better for future readers because they won't wonder why the 's' variable is declared outside the loop if it's never used later

Share Improve this answer Follow

answered Sep 21, 2008 at 2:51

Greg Hewgill
990k ● 191 ● 1.2k ● 1.3k

6

If you want to speed up for loops, I prefer declaring a max variable next to the counter so that no repeated lookups for the condidtion are needed:

instead of

```
for (int i = 0; i < array.length; i++) {
  Object next = array[i];
}
```

I prefer

```
for (int i = 0, max = array.lenth; i < max; i++) {
  Object next = array[i];
}
```

Any other things that should be considered have already been mentioned, so just my two cents (see ericksons post)

Greetz, GHad

Share Improve this answer Follow

answered Sep 21, 2008 at 13:55

GHad

9,621 ● 6 ● 36 ● 40

4

To add on a bit to @Esteban Araya's answer, they will both require the creation of a new string each time through the loop (as the return value of the `some Assignment` expression). Those strings need to be garbage collected either way.

Share

Improve this answer

Follow

edited May 23, 2017 at 11:46

Community Bot

1 ● 1

answered Sep 21, 2008 at 2:54

1800 INFORMATION

135k ● 30 ● 163 ● 242

Not necessarily. We don't know what is being assigned to "s" inside the loop. Perhaps its a string constant that is allocated in PermGen space and will never be garbage collected. All we know is that whatever it is, its the same in both cases, so it doesn't matter. – erickson Sep 21, 2008 at 6:28

@erickson: I agree that it could be a string constant, but in that case, I expect the compiler would probably use constant propagation to move s out of the body of the loop. I was assuming it wasn't a constant because a sensible programmer would have done the same. – 1800 INFORMATION Sep 21, 2008 at 8:25

I didn't mean the same string constant in every iteration. See my comment on the OP. – erickson Sep 21, 2008 at 15:44

Yes in that case there would be no optimsation – 1800 INFORMATION Sep 21, 2008 at 20:25

3

I know this is an old question, but I thought I'd add a bit that is *slightly* related.

I've noticed while browsing the Java source code that some methods, like String.contentEquals (duplicated below) makes redundant local variables that are merely copies of class variables. I believe that there was a comment somewhere, that implied that accessing local variables is faster than accessing class variables.

In this case "v1" and "v2" are seemingly unnecessary and could be eliminated to simplify the code, but were added to improve performance.

```
public boolean contentEquals(StringBuffer sb) {
    synchronized(sb) {
        if (count != sb.length())
            return false;
        char v1[] = value;
        char v2[] = sb.getValue();
        int i = offset;
        int j = 0;
        int n = count;
        while (n-- != 0) {
            if (v1[i++] != v2[j++])
                return false;
        }
    }
    return true;
}
```

Share

Improve this answer

Follow

answered Apr 13, 2009 at 20:13

community wiki
Randy Stegbauer

That was probably more useful on old VMs than it is now. I have a hard time believing that makes any difference on HotSpot. – Michael Myers ♦ Apr 13, 2009 at 20:27

1

It seems to me that we need more specification of the problem.

The

```
s = some Assignment;
```

is not specified as to what kind of assignment this is. If the assignment is

```
s = "" + i + "";
```

then a new sting needs to be allocated.

but if it is

```
s = some Constant;
```

s will merely point to the constants memory location, and thus the first version would be more memory efficient.

Seems i little silly to worry about to much optimization of a for loop for an interpreted lang IMHO.

Share Improve this answer Follow

answered Sep 21, 2008 at 5:32

Milhous
14.6k ● 16 ● 66 ● 83

1

When I'm using multiple threads (50+) then i found this to be a very effective way of handling ghost thread issues with not being able to close a process correctly ....if I'm wrong, please let me know why I'm wrong:

```
Process one;
BufferedInputStream two;
try{
one = Runtime.getRuntime().exec(command);
two = new BufferedInputStream(one.getInputStream());
}
}catch(e){
e.printstacktrace
}
finally{
//null to ensure they are erased
one = null;
two = null;
//nudge the gc
System.gc();
}
```

Share

Improve this answer

Follow

answered Jul 29, 2013 at 21:10

community wiki
Petro