String Comparison : individual comparison Vs appended string comparison

Asked 15 years, 11 months ago Modified 14 years, 11 months ago Viewed 1k times

2

I have six string variables say str11, str12, str13, str21, str21 and str23.

I need to compare combination of these variables.

The combinations I have to check is str11 -- str12 -- str13 as one group and str21 -- str22 -- str23 as other group. I have to compare these two groups.

Now I'm in confusion which method should I use for comparison?

Can I append strings of same group and compare, which is only one comparison say `( str11 append str12 append str13 )  eqauls ( str21 append str22 append str23 )`

Or

Should I go for individual 3 comparisons?

```
if( str11 equals str21 ) {

    if( str12 equals str22 ) {

        if( str13 equals str23 ) {

        }

    }

}
```

What is performance factor which costs me because of string length when I do string comparison? Lets us assume all strings are of same(approx) length.

java string comparison

Share

Improve this question

Follow

edited Jan 6, 2009 at 13:32

Binary Worrier

51.7k • 20 • 142 • 186

asked Jan 6, 2009 at 13:24

pramodc84

1,614 • 2 • 26 • 33

Why are you worrying about performance here? Is the comparison a bottleneck? – orip Jan 6, 2009 at 14:06

8 Answers

Sorted by: Highest score (default)

10

I'd test individually.

Is "AB" "CD" "EF" equal to "ABC" "DE" "F"?

Me thinks not.

P.S. If it is, then it's a VERY special case, and if you decide to code it that way (as a concatenated comparison) then comment the hell out of it.

Share Improve this answer Follow

answered Jan 6, 2009 at 13:35

Binary Worrier
51.7k • 20 • 142 • 186

8

Splitting the comparison into three if statements is definitely not necessary. You could also simply do an AND with your comparisons, eg

```
if (  str11 equals str21
   && str12 equals str22
   && str13 equals str23) ...
```

Share

Improve this answer

Follow

edited Mar 3, 2009 at 8:14

Hosam Aly
42.4k • 37 • 146 • 182

answered Jan 6, 2009 at 13:31

Adam Bellaire
110k • 19 • 152 • 165

3

Your variable names indicate a major code smell. It sounds like instead of having six variables, you should instead have two arrays, each containing three strings. In other words, something like this initially would have been much better:

```
String[][] strs = new String[2][3];
strs[0][0] = str11;
strs[0][1] = str12;
...
```

Chances are that depending where you obtained the six strings from, you will not need to do this manually immediately before the comparison, but can likely pass in your arguments in a format that is more friendly.

If you do wish to do this by comparing arrays of the string objects, and you are using Java 1.5 or above, remember that you have access to the java.util.Arrays.equals() methods for array equality. Using library methods as much as possible is a great way to avoid extra effort reinventing the wheel, and possible implementation mistakes (both submitted implementations so far have bugs, for example).

The exact route you take probably depends on the domain you are writing for - if your particular problem requires you to always compare 3-tuples, then writing the code to explictly compare groups of three strings would not be such a good idea, as it would probably be more immediately understandable than code that compared arrays of arbitrary length. (If you are going this route, then by all means us a single if() conditional with && instead of nested if blocks, as Adam Bellaire demonstrated).

In general though, you'll have a much more reuable block of code if you set it up to work with arrays of arbitrary length.

Share Improve this answer Follow

answered Jan 7, 2009 at 16:51

Andrzej Doyle
104k ● 33 ● 191 ● 231

1

Appending the strings together and comparing will not work. For instance, strings 1 and 2 could be empty and string 3 could contain "gorps", while string 4 contains "gorps" and 5 and 6 are empty. A comparison of the appended results would return true, though that would be a false positive. You would have to come up with a delimiter you guarantee would not be contained in any string to get this to work, and that could get messy.

I would just do the comparison the way you are doing it. It's readable and straightforward.

Share Improve this answer Follow

answered Jan 6, 2009 at 13:34

Tom Moseley
591 ● 7 ● 15

@Tom I have guaranteed delimiter, then? – pramodc84 Jan 6, 2009 at 13:36

If you decide to add a delimiter, then you're extending the cost of the string concatination (you have n-1 additional concatinations to perform for n strings) – Binary Worrier Jan 6, 2009 at 13:38

If you have a guaranteed delimiter, that scheme will work, but string concatenation involves the creation of additional string objects. As I said, I'd do the individual comparisons. The other way would work, but would be more costly. – Tom Moseley Jan 6, 2009 at 13:42

@Binary - I know. I presented this as the only condition under which the concat approach would work. As I said, I'd do the individual comparison method. – Tom Moseley Jan 6, 2009 at

13:43

@Tom: I hear you, it was more as a warning to reader, I reckoned from your response you though it was a bad idea. Have a goodun dude :) – Binary Worrier Jan 6, 2009 at 14:08

1

The iteration over over one large char[] is probably faster than iteration over n separate string of a total equal length. This is because data is very local and the CPU has an easy time to prefetch data.

However, when you concatenate multiple strings in Java you will use StringBuilder/Buffer and then convert i back to a String in several cases. This will cause increased memory allocation due to how SB.append() works and Java String being immutable, which in turn can create a memory bottleneck and slow down your application significantly.

I would recommend keeping the Strings as is and do separate comparison. The gain in performance due to a longer char[] most likely is far less than the problems you can run in to with the higher allocation rate.

Share Improve this answer Follow

answered Jan 6, 2009 at 14:11

sfriberg

55 • 6

1

With all respect: I think your code and question not only smells a bit, but almost stinks (big smiley here).

1) the variable names indicate actually having string-vectors around; as already mentioned
2) the question of individual compares vs. a concatenated compare raises the question of how you define equality of your string-tuples; also already mentioned.

But what strikes me most:

3) To me that looks like a typical case of "premature optimization" and counting CPU cycles at the wrong place.

If you really care for the performance, forget about the cost of 3 individual compares against a single compare. Instead:

How about the added overhead of creating two concatenated strings ?

```
(str11 + str12 + str13) = (str21 + str22 + str23)
```

Lets analyze that w.r.t. to the memory manager and operations to be done. On the low level, that translates is 4 additional memory allocations, 2 additional strcpy's, and either another 4 additional strcat or strcpy (depending on how the VM does it; but most would use another strcpy) operations. Then a single compare is called for, which does not first count the characters using strlen; instead it either knows the size in advance (if the object header also includes the number of chars, which is likely) or it simply runs up to a 0-byte. That is called once vs. 3 times. The actual number of chars to compare is roughly the same (forget about the extra 0-bytes). That leaves us with 2 additional calls to strcmp (a few nS), vs. the overhead I described above (a few uS). If we add up the GC reclamation overhead (0 allocations vs. 4), I'd say that your "optimized" solution can easily be a 100 to 1000 times slower than the 3 strcmps !

Additional Notice:
Theroretically, the JITter could optimize it or some of it, and actually generate code as suggested by Adam Bellaire, but I doubt that any JIT-developer cares to optimize such code. By the way, the system's string routines (aka String operations) are usually MUCH faster than handcoding, so do not start to loop over individual characters yourself.

answered Jan 13, 2009 at 16:22

blabla999
3,200 ● 23 ● 24

I would add the two groups in two arrays, and then loop over the arrays to compare the individual strings in that array. A good example is already in the ansewers, given by Markus Lausberg.

I would not be concerned about performance costs. Just write it in the most readable way possible. The Java compiler is very good in performance optimizations.

Example method:

```
    public boolean compareGroups(String[] group1, String[] group2){
    if (group1.length != group2.length ){
        return false;
    }

    for (int i = 0; i < group1.length; i++) {
        if (!group1[i].equals(group2[i])){
            return false;
        }
    }

    return true;
}
```

And calling the method is ofcourse simple:

```
    String[] group1 = new String[]{"String 1", "String 2", "String 3"};
String[] group2 = new String[]{"String 1", "String 2", "String 3"};

boolean result = compareGroups(group1, group2);
```

Share

Improve this answer

Follow

edited Jan 6, 2009 at 14:25

answered Jan 6, 2009 at 14:19

Rolf

7,268 ● 6 ● 42 ● 57

@Rolf Yes ofcourse. I'm not changing existing question. This is one more question if im not wrong – pramodc84 Jan 6, 2009 at 14:44

0

i would use the simple way

dynamic run over all array elements of both arrays.

```
        boolean isEqual = true;
        for(int n = 0;n<str1.length;++n){
            isEqual &= str1[n].equals(str2[n]);
        }

        return isEqual;
```

Share

Improve this answer

Follow

edited Jan 6, 2009 at 15:09

answered Jan 6, 2009 at 13:35

Markus Lausberg

12.3k ● 6 ● 42 ● 66

I don't think that will compile. – Grant Wagner Jan 6, 2009 at 14:07

-1 overengineered, not optimized (large n and str1[0] is not equal to str2[0]) – eljenso Jan 6, 2009 at 15:19