Getting an int representation of a String

Asked 16 years, 3 months ago Modified 11 years ago Viewed 17k times

10

I am looking for a way to create an int\long representation of an arbitrary alpha-numeric String. Hash codes won't do it, because I can't afford hash collisions i.e. the representation must be unique and repeatable.

The numeric representation will be used to perform efficient (hopefully) compares. The creation of the numeric key will take some time, but it only has to happen once, whereas I need to perform vast numbers of comparisons with it - which will hopefully be much faster than comparing the raw Strings.

Any other idea's on faster String comparison will be most appreciated too...

java performance string

Share

Improve this question

Follow

edited Dec 2, 2013 at 13:09

e-sushi

14.2k ● 10 ● 39 ● 57

asked Sep 5, 2008 at 16:11

jase

159 ● 1 ● 2 ● 6

14 Answers

Sorted by: Highest score (default)

12

Unless your string is limited in length, you can't avoid collisions.

There are 4294967296 possible values for an integer (2^32). If you have a string of more than 4 ASCII characters, or more than two unicode characters, then there are more possible string values than possible integer values. You can't have a unique integer value for every possible 5 character string. Long values have more possible values, but they would only provide a unique value for every possible string of 8 ASCII characters.

Hash codes are useful as a two step process: first see if the hash code matches, then check the whole string. For most strings that don't match, you only need to do the first step, and it's really fast.

Share Improve this answer Follow

edited Sep 5, 2008 at 20:35

answered Sep 5, 2008 at 16:23

Don Kirkby

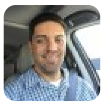
56.5k ● 27 ● 219 ● 301

Can't you just start with a hash code, and if the hash codes match, do a character by character comparison?

10

Share Improve this answer Follow

answered Sep 5, 2008 at 16:15



Patrick McElhaney
59.1k ● 41 ● 137 ● 169

6

How long are the strings? If they are very short, then a unique ID can be generated by considering the characters as digits in base 36 (26 + 10) that form a *n*-digits number where *n* is the length of the string. On the other hand, if the strings are short enough to allow this, direct comparison won't be an issue anyway.

Otherwise you'll have to generate a collision-free hash and this can only be done when the complete problem space is known in advance (i.e. if you know all strings that can possibly occur). You will want to have a look at perfect hashing, although the only feasible algorithm to find a perfect hash function that I know is probabilistic so collisions are still theoretically possible.

There might be other ways to find such a function. Knuth called this a “rather amusing … puzzle” in TAoCP but he doesn't give an algorithm either.

In general, you give way too few information to find an algorithm that doesn't require probing the whole problem space in some manner. This does invariably mean that the problem has exponential running time but could be solved using machine-learning heuristics. I'm not sure if this is advisable in your case.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:20

Konrad Rudolph
545k ● 138 ● 956 ● 1.2k

2

Perhaps:

```
String y = "oiu291981u39u192u3198u389u28u389u";
BigInteger bi = new BigInteger(y, 36);
System.out.println(bi);
```

Share Improve this answer Follow

answered Sep 5, 2008 at 16:21

toolkit
50.2k ● 18 ● 110 ● 138

2

At the end of the day, a single alphanumeric character has at least 36 possible values. If you include punctuation, lower case, etc then you can easily pass 72 possible values.

A non-colliding number that allows you to quickly compare strings would necessarily grow exponentially with the length of the string.

So you *first* must decide on the longest string you are expecting to compare. Assuming it's N characters in length, and assuming you ONLY need uppercase letters and the numerals 0-9 then you need to have an integer representation that can be as high as 36^N

For a string of length 25 (common name field) then you end up needing a binary number with 130 bits.

If you compose that into 32 bit numbers, you'll need 4. Then you can compare each number (four integer compares should take no time, compared to walking the string). I would recommend a big number library, but for this specialized case I'm pretty sure you can write your own and get better performance.

If you want to handle 72 possible values per character (uppercase, lowercase, numerals, punctuation...) and you need 10 characters, then you'll need 62 bits - two 32 bit integers (or one 64 bit if you're on a system that supports 64 bit computing)

If, however, you are not able to restrict the numbers in the string (ie, could be any of the 256 letters/numbers/characters/etc) and you can't define the size of the string, then comparing the strings directly is the only way to go, but there's a shortcut.

Cast the pointer of the string to a 32 bit unsigned integer array, and compare the string 4 bytes at a time (or 64 bits/8bytes at a time on a 64 bit processor). This means that a 100 character string only requires 25 compares maximum to find which is greater.

You may need to re-define the character set (and convert the strings) so that the characters with higher precedence are assigned values closer to 0, and lower precedence values closer to 255 (or vice versa, depending on how you are comparing them).

Good luck!

-Adam

answered Sep 5, 2008 at 16:36

Adam Davis
93.5k ● 60 ● 271 ● 333

2

As long as it's a hash function, be it String.hashCode(), MD5 or SHA1, collision is unavoidable unless you have a fixed limit on the string's length. It is mathematically impossible to have one-to-one mapping from an infinite group to a finite group.

Stepping back, is collision avoidance *absolutely* necessary?

answered Sep 5, 2008 at 22:25

ckpwong
2,129 ● 1 ● 17 ● 17

If the string length is fixed, how collision is unavoidable? can you please explain? – Swamy Apr 5, 2013 at 12:00

1

A few questions in the beginning:

1. Did you test that simple string comparison is too slow?
2. How the comparison looks like ('ABC' == 'abc' or 'ABC' != 'abc')?
3. How many string do you have to compare?
4. How many comparison do you have to do?
5. How your strings look like (the length, letter case)?

As far as I remember String in Java is an object and two identical strings point to the same object.

So, maybe it would be enough to compare objects (probably string comparison is already implemented in this way).

If it doesn't help you can try to use Pascal implementation of string object when first element is length and if your strings have various length this should save some CPU time.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:22

Grzegorz Gierlik
11.2k ● 4 ● 48 ● 57

0

How long are your strings? Unless you choose an int representation that's longer than the string, collisions will always be possible no matter what conversion you're using. So if you're using a 32 bit integer, you can only uniquely represent strings of up to 4 bytes.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:14

Chris Upchurch
15.5k ● 6 ● 52 ● 66

0

How big are your strings? Arbitrarily long strings cannot be compressed into 32/64 bit format.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:15

Pramod
9,456 ● 4 ● 27 ● 28

0

If you don't want collisions, try something insane like SHA-512. I can't guarantee there won't be collisions, but I don't think they have found any yet.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:16

Thomas Owens
116k ● 99 ● 317 ● 436

0

Assuming "alphanumeric" means letters and numbers, you could treat each letter/number as a base-36 digit. Unfortunately, large strings will cause the number to grow rapidly and you'd have to resort to big integers, which are hardly efficient.

If your strings are usually different when you make the comparison (i.e. searching for a specific string) the hash might be your best option. Once you get a potential hit, you can do the string comparison to be sure. A well-designed hash will make collisions exceedingly rare.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:18

Kyle Cronin
79k ● 45 ● 151 ● 167

0

It would seem that an MD5 hash would work fine. The risk of a hash collision would be extremely unlikely. Depending on the length of your string, a hash that generates an int/long would run into max value problems very quickly.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:19

Todd
3,363 ● 3 ● 22 ● 12

0

Why don't you do something like 1stChar + (10 x 2ndChar) + 100 x (3rdChar) ...., where you use the simple integer value of each character, i.e. a = 1, b = 2 etc, or just the integer value if it's not a letter. This will give a unique value for each string, even for 2 strings that are just the same letters in a different order.

Of course if gets more complicated if you need to worry about Unicode rather than just ASCII and the numbers could get large if you need to use long string.

Are the standard Java string comparison functions definitely not efficient enough?

Share Improve this answer Follow

answered Sep 5, 2008 at 16:20

Matt Warren
10.3k ● 7 ● 51 ● 63

0

> String length may vary, but let's say 10 characters for now.

In that case, in order to guarantee uniqueness you'd have to use some sort of big integer representation. I doubt that doing comparisons on big integers would be substantially faster than doing string comparisons in the first place. I'll second what other's have said here, use some sort of hash, then in the event of a hash match check the original strings to weed out any collisions.

In any case, If your strings are around 10 characters, I doubt that comparing, say, a bunch of 32 bit hashes will be all that much faster than direct string comparisons. I think you have to ask yourself if it's it really worth the additional complexity.

Share Improve this answer Follow

answered Sep 5, 2008 at 16:24

Chris Upchurch
15.5k ● 6 ● 52 ● 66