sign up log in tour help careers 2.0

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Take the 2-minute tour ×

Why are strings immutable in many programming languages? [duplicate]

AdMob by Google — Improve the performance of your app globally — Sign up

Possible Duplicate:
Why can't strings be mutable in Java and .NET?
Why .NET String is immutable?

Several languages have chosen for this, such as C#, Java, and Python. If it is intended to save memory or gain efficiency for operations like *compare*, what effect does it have on concatenation and other modifying operations?

java c++ string immutability

edited Jun 8 at 1:30
Demetri
715 1 4 17

asked Mar 3 '12 at 6:51
user1087373
404 4 9

marked as duplicate by Mehrdad, FredOverflow, Bo Persson, Marvin Pinto, Anthony Pegram Mar 3 '12 at 22:14

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

1 stackoverflow.com/questions/2365272/why-net-string-is-immutable – Mehrdad Mar 3 '12 at 7:15

16 C++'s `std::string` is *not* immutable. – FredOverflow Mar 3 '12 at 7:48

add a comment

4 Answers

Immutable types are a good thing generally:

- They work better for concurrency (you don't need to lock something that can't change!)
- They reduce errors: mutable objects are vulnerable to being changed when you don't expect it which can introduce all kinds of strange bugs ("action at a distance")
- They can be safely shared (i.e. multiple references to the same object) which can reduce memory consumption and improve cache utilisation.
- Sharing also makes copying a very cheap O(1) operation when it would be O(n) if you have to take a defensive copy of a mutable object. This is a big deal because copying is an incredibly common operation (e.g. whenever you want to pass parameters around....)

As a result, it's a pretty reasonable language design choice to make strings immutable.

Some languages (particularly functional languages like Haskell and Clojure) go even further and make pretty much everything immutable. This enlightening video is very much worth a look if you are interested in the benefits of immutability.

There are a couple of minor downsides for immutable types:

- Operations that create a changed string like concatenation are more expensive because you need to construct new objects. Typically the cost is O(n+m) for concatenating two immutable Strings, though it can go as low as O(log (m+n)) if you use a tree-based string data structure like a Rope. Plus you can

always use special tools like Java's StringBuilder if you really need to concatenate Strings efficiently.
- A small change on a large string can result in the need to construct a completely new copy of the large String, which obviously increases memory consumption. Note however that this isn't usually a big issue in garbage-collected languages since the old copy will get garbage collected pretty quickly if you don't keep a reference to it.

Overall though, the advantages of immutability vastly outweigh the minor disadvantages. Even if you are only interested in performance, the concurrency advantages and cheapness of copying will in general make immutable strings much more performant than mutable ones with locking and defensive copying.

edited Mar 3 '12 at 10:00 answered Mar 3 '12 at 6:56
mikera
64k 10 127 276

Good points. But, every coin has two sides. You just listed the upsides. What about the downsides? As it is immutable, we can't concatenate two strings simply by pending one to another; We also have to create a new string even it differs as little as only a char from an existing string, which means a cpu-consuming allocation of memory. – user1087373 Mar 3 '12 at 7:10

4 If Strings were mutable, given the massive usage, it would create a lot of hard-to-detect bugs in large projects. The cost of these bugs simply make string concatenation CPU-consuming concern negligible. Moreover, in many cases, you still need to create a different string when making slight modification even if strings are mutable. – Bob Wang Mar 3 '12 at 7:24

I've expanded my answer a little. But note that in particular with your appending example, the old immutable string will generally get GC'd pretty quickly, so the impact on memory usage is only very temporary. – mikera Mar 3 '12 at 7:25

2 @Nawaz: It makes perfects sense. In the highly likely case that your String is in the young generation, then it will get cleared very quickly. Also on the JVM at least you may get it allocated on the stack via escape analysis for instant freeing. And even if it didn't free quickly, you *don't care* because it is guaranteed to be freed before you need the memory again. The GC is your friend, people really need to learn to stop worrying about it. – mikera Mar 4 '12 at 3:45

3 @Nawaz: Depends on your definition of fast, but absolutely yes if you interpret fast as "fast enough that you have no reason to care in practice". If you want to be pedantic and say "as fast as possible" then no, but that's clearly a pointless academic argument. Latency doesn't matter if it's not a bottleneck. – mikera Mar 4 '12 at 3:52

show **4** more comments

It's mainly intended to prevent programming errors. For example, Strings are frequently used as keys in hashtables. If they could change, the hashtable would become corrupted. And that's just one example where having a piece of data change while you're using it causes problems. Security is another: if you checking whether a user is allowed to access a file at a given path before executing the operation they requested, the string containing the path better not be mutable...

It becomes even more important when you're doing multithreading. Immutable data can be safely passed around between threads while mutable data causes endless headaches.

Basically, immutable data makes the code that works on it easier to reason about. Which is why purely functional languages try to keep *everything* immutable.

answered Mar 3 '12 at 7:03
Michael Borgwardt
185k 26 260 496

add a comment

In Java not only String but all primitive Wrapper classes (Integer, Double, Character etc) are immutable. I am not sure of the exact reason but I think these are the basic data types on which all the programming schemes work. If they change, things could go wild. To be more specific, I'll use an example: Say you have opened a socket connection to a remote host. The host name would be a String and port would be Integer. What if these values are modified after the connection is established.

As far as performance is concerned, Java allocates memory to these classes from a separate memory section called Literal Pool, and not from stack or Heap. The Literal Pool is indexed and if you use a string "String" twice, they point to the same object from Literal pool.

answered Mar 3 '12 at 7:17

Ameya
309 1 3

Great point. Thanks. – user1087373 Mar 3 '12 at 9:08

The "literal" pool is only used in certain cases. For numeric wrapper classes this is only used for numbers in a certain range (for int I think -128 to 127) and only when doing boxing operations (i.e. new Integer(5) will still create a new `Integer` object; but `Integer n = 5` will reference an `Integer` object in the intern pool). For `String` , this only happens with actual literals; if you create a string dynamically (e.g. `String a = "x"; String b = a + "abc";` (here b is on heap)) then it will be a new object on the normal heap(s) unless you intentionally intern it with `String.intern()` . – Kevin Brock Mar 3 '12 at 13:14

@user1087373: I don't think you should accept this answer. This explains nothing. On the contrary, it provides incorrect rationale. – Nawaz Mar 4 '12 at 3:20

@Nawaz, yes, Actually, the second answer is more preferable to me. I did not realize only one answer can be chosen as accepted until your remind. Thanks – user1087373 Mar 5 '12 at 2:13

add a comment

Having strings as immutable also allows the new string references easy, as the same/similar strings will be readily available from the pool of the Strings previously created. Thereby reducing the cost of new object creation.

answered Mar 3 '12 at 11:38
saurabytes
1 1

add a comment

Not the answer you're looking for? Browse other questions tagged java c++ string immutability or ask your own question.