What is the easiest/best/most correct way to iterate through the characters of a string in Java?

Microsoft
Bruce Iverfer apps their Azure
Sign up in August and get FREE gear and a chance to win a trip to Seattle
Microsoft Azure
Ask a question or vote on questions you like.

asked 9 years, 6 months ago
viewed 1,251,201 times
active 4 months ago

Ben ijt toe een extra teampower?
Bekijk de vacatures
Google Chrome

Linked

- 1 Iterate through a string in Ruby
- 1 Iterating Characters from Strings in Java
- 1 How do I turn a String into a InputStreamReader in Java?
- 1 Fastest way to iterate over all the chars in a string
- 1 How do I apply the for each loop to every character in a string?
- 1 How can I iterate through the unicode codepoints of a Java String?
- 1 Java: How to get Iterator<Character> from String
- see more linked questions...

Related

- 1 How to generate a random alpha-numeric string
- 1 Using HashMap in Java to make a mirror code
- 1 Reading Characters from Strings in Java
- 1 How do I turn a String into a InputStreamReader in Java?
- 1 Fastest way to iterate over all the chars in a string
- 1 How do I apply the for each loop to every character in a string?
- 1 How can I iterate through the unicode codepoints of a Java String?
- 1 Java: How to get Iterator<Character> from String
- see more related questions...

Hot Network Questions

- Another guessing game kamoyo.com challenge (hard)
- Number of ways to take the numbers from 1 to 9 and combine them into a single number, such that the resulting number is even and is still not the same
- Are Kickstarter frequent or US-school gamers in the target audience?
- Why are measures one "bar"? Important?
- Do people know of good literature by Bogdanov?
- Sentinel Return uses too much awesome magic
- Shell Gas Item. Release it away or its easy up
- Why is there no indication of relative numbers on the Richter scale?
- Is it appropriate to email a professor saying you enjoyed their class, after doing well in it?
- Magpies 2 - command shortcut
- Do magic weapons bypass monster resistance?
- Secure way of receiving our sensitive information in a non-secure email
- Design Transistors on an existing creative
- Can my farmer eat me?
- How do I politely decline a request for a contract signature?
- What does "coming out of breath" mean?
- Sole derivative of signed function in neural networks
- Why can't I hang gate in MC?
- Why do use snacks to identify a pet?
- How to swap elements in a vector using an arbitrary permutation in Mathematica
- What's the easiest for implementation of Kaiser window?
- What is the difference between "incident", "fatigue" and "wear"?
- Merge with constant x substitution
- How are the off world colonies reached in Blade Runner?
- Question feed

Join Stack Overflow to learn, share knowledge, and build your career. Email Sign Up or Use an Network

I use a for loop to iterate the string and use `charAt()` to get each character to examine it. Since the String is implemented in the array, the `charAt()` method is a constant time operation.

```
String s = "...stuff...";
for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    ...handle c...
}
```

240

That's what I would do. It seems the easiest to me.

As far as correctness goes, I don't believe that exists here. It is all based on your personal style.

share · improve this answer — answered Oct 19 '08 at 14:22 — Matthew Flaschen — edited Oct 19 '08 at 14:25

- Does the compiler inline the length? and the charAt? — Lin Oct 19 '08 at 6:25
- It might inline length(), that is how the method behind that call is a free function, but its more efficient to do this anyway. — Matthew Flaschen Oct 19 '08 at 14:25 — Steve Cherney Oct 11 '09 at 17:14
- Collating Unicode code is a very performance gain. Please avoid this until you decide this area of code is speed critical. — Lin Oct 19 '08 at 6:25
- I usually add the length to a variable to make the intention more explicit. — Jason Oct 19 '08 at 14:25
- Note that this technique gives you characters, not code points, meaning you may get surrogates. — Gabe Oct 19 '08 at 15:12

show 6 more comments

Ben ijt toe een extra teampower?
Bekijk de vacatures
Google Chrome

Two options

```
for(int i = 0, n = s.length(); i < n ; i++) {
    char c = s.charAt(i);
}
```

or

```
for(char c : s.toCharArray()) {
    ...do something...
}
```

The first is probably faster, then 2nd is probably more readable.

share · improve this answer — answered Oct 19 '08 at 14:22 — Lin — edited Oct 19 '08 at 14:26

- I plus one for placing the length in the initialization expression. If anyone doesn't know why, it's because that is only evaluated once whereas if it was placed in the termination statement (ie: i < s.length), then s.length() would be evaluated every time through the loop. — Jason Oct 19 '08 at 14:27
- I thought compilers were smart enough to do that for you. — Ethan Oct 19 '08 at 14:28
- @Matthew You can the Jonas Jasas disassembler to see that the repeated calls to s.length() for loop termination expression are indeed inlined. Note that in the case OP posted the call to s.length() in the initialization expression, so the language semantics already guarantees that it will be called only once. — Tomalak Oct 19 '08 at 14:29
- @Jason Note that the language semantics guarantee the condition, NOT in the case files. Even if you was repeated calls to length() that doesn't include a runtime penalty, necessarily. — Isaac Dec 31 '10 at 20:42
- @SteveCherney, why would you define "n" in a variable? Isn't it just less code? i < (n = s.length()); n++{ ? — Lance Mar 21 '11 at 19:17

show 6 more comments

Note most of the other techniques described here break down if you're dealing with characters outside the Basic Multilingual Plane, for instance emoji. Such characters are stored in Unicode as pairs of UTF-16 units. They will only iterate properly if you iterate the code points outside that are mostly assigned to dead languages. But there are some useful characters outside this, for example some code points used for mathematical notation, and some used to encode proper names in Chinese.

In that case (your code will be):

```
String str = "...";
int offset = 0;
while (offset < str.length()) {
    int codePoint = str.codePointAt(offset);
    offset += Character.charCount(codePoint);
    // Do something with the codePoint
}
```

The `Character.charCount()` method requires Java 5+.

Source: http://mindprod.com/jgloss/codepoint.html

share · improve this answer — answered Jan 16 '14 at 11:45 — Suragch — edited Jan 16 '14 at 11:49

- Suragch Good. — Jonathan Drapeau Jan 16 '14 at 13:47 — Jonathan Drapeau Jan 16 '14 at 13:47

- I didn't get how you use anything but the Basic Multilingual Plane here. Lucifer is still 16 bits right? — Prof. Falken Feb 11 '14 at 22:00
- You either use an int to store the entire code point, or else each char will only store one out of the two surrogate pairs that define the character. — Jon Skeet Feb 11 '14 at 22:04
- I think I need to read up on code points and surrogate pairs. Thanks! — Prof. Falken Feb 11 '14 at 22:06
- It's unclear from the only sense that it's correct for Unicode chars outside of the BMP. — Jason S Jul 10 '14 at 20:00

Wrote some code to illustrate the concept of looping over codepoints (as opposed to chars): gist.github.com/mikemaccana/7... — Christopher Eze Oct 10 '14 at 12:12

show 1 more comment

I agree that String.Splitter is overkill here. Actually I tried out the suggestions above and took the time.

My best was fairly simple: create a StringSplitter with about a million characters, convert it to a String, and traverse each of them with charAt(). After converting to a char array I with a CharacterIterator a little faster. Not much faster, but making sure to do something on the string in the compiler can't optimize away the whole loop.

The result on my 2.4 GHz Powerbook (that's a mac :-)) and JDK 1.5:

- Test 1: charAt + String: ~2150ms
- Test 2: String converted to array: ~~9900ms~~
- Test 3: StringBuilder.charAt(): ~~3530ms~~
- Test 4: CharacterIterator and String: ~~2150ms~~

As the results are significantly different, the most straightforward way this sequence. Sorry to be that bastard, also interestingly, charAt() on StringBuilder seems to be slightly slower than the one of String.

BTW I suggest not to use CharacterIterator as I consider its abuse of the 'UATTF' character as 'end of iteration' a really awful hack. In big projects Sun's always has guys that use the same kind of hack for two different purposes and the code crashes really mysteriously.

Here's one of the tests:

```
int count = 1000;
while
{
    System.out.print("Test 1: charAt + String");
    long s = System.currentTimeMillis();
    int l = 0;
    for (int i=0; i<count; i++)
        String str = "...";
        for (int j=0; j<length; j++)
            if (str.charAt(j) == 'b')
            {
                l++;
            }
    s = System.currentTimeMillis()-s;
    System.out.println("result: " + l + " after " + s + "msec");
}
```

share · improve this answer — answered Oct 19 '08 at 21:08 — Peter Lawrey

- This has the same problem noted here: stackoverflow.com/questions/196830 ... — Emmanuel Ogeh Oct 10 '14 at 21:08

add a comment

There are some dedicated classes for this.

```
import java.text.*;

final CharacterIterator it = new StringCharacterIterator(s);
for(char c = it.first(); c != CharacterIterator.DONE; c = it.next()) {
    // process c
}
```

10

share · improve this answer — answered Oct 19 '08 at 21:39 — David De Florinier — edited Dec 2 '13 at 14:57

- Looks like an overkill for something as simple as iterating over immutable char array. — dimitrov Oct 19 '08 at 21:39

- I don't see why this is overkill. Iterators are the most pure and easy to read way of looping - iterative. The StringCharacterIterator is bound to take full advantage of immutability. — John Oct 17 '10 at 8:21

Agree with @dimitrov. Iterators are overkill. The only reason to use an iterator would be to take advantage of foreach, which is a bit easier to "read" than a for loop. If you're going to write a conventional for loop anyway, there's no reason to use an iterator. — Jeff Ryan Dec 2 '13 at 15:04

- The iterator class hierarchy is probably the only correct way to deal with Unicode characters. Unicode strings require more space than a Java `char`: provides A Java `char` contains 16 bit and can hold Unicode characters in the range U+0000 to U+FFFF. Unicode characters in the range U+10000 to U+10FFFF are encoded in results in a variable length character encoding. Most answers on this page assume that the Java encoding is a constant length encoding, which is wrong. — Joeng Jun 13 '13 at 20:04

- Glowing it does not seem that a character iterator is going to help you with non-BMP characters. stackoverflow.com/questions/9515491/iterating-over-a-string-with-non-bmp-unicode — Bruce De Fraine Jun 13 '13 at 20:08

show 6 more comments

If you have Guava on your classpath, the following is a pretty readable alternative. Guava even has a fairly sensible custom List implementation for strings, so this should be sufficient.

```
for(Char c : Lists.charactersOf(yourString) {
    // do whatever you want
}
```

UPDATE: As @Jens noted, with Java 8 there's also `CharSequence.chars`. to use. Even the type is IntStream, so it can be mapped to chars like:

```
yourString.chars()
    .mapToObj(c -> (Character.isLetterOf(char) ?
        (char)(65 - 32 + c) : (char)c))
    .forEach(System.out::print);
```

share · improve this answer — answered Oct 19 '08 at 11:23 — Jens Schauder — edited Jun 20 '14 at 9:32

- Guava is a very good library but not every time Stringliterator is not recommended INCLUDING a question tagged with "performance" — Bruce De Fraine Jun 13 '13 at 20:04 stackoverflow.com/q/9515491 for a starting point with interesting. Upvoted to offset — Pharaoh Jun 13 '13 at 20:10

- Thanks for. Bernman. . . I take it that the cited block quote should have been created clear, where she should probably note that active flag lives well for a comments to String/Stringbuilder — Ben Oct 11 '13 at 20:42

add a comment

In **Java 8** we can solve it as

```
String str = "...";
int[] codePoints = str.codePoints().toArray();
int[] codePoints1 = str.codePoints().toArray();
```

The method chars() returns an `IntStream`, as mentioned in doc.

Returns a stream of int zero-extending the char values from this sequence. Any char which maps to a surrogate code point is passed through uninterpreted. If the sequence is mutated while the stream is being read, the result is undefined.

The method `codePoints()` also returns an `IntStream` as per doc:

Returns a stream of code point values from this sequence. Any surrogate pairs encountered in the sequence are combined as if by Character.toCodePoint and the result is passed to the stream. Any other code units, including ordinary BMP characters, unpaired surrogates, and unidefined code units, are zero-extended to int values which then are passed to the stream.

How is char and code point different? As mentioned in this article:

Unicode 3.1 added supplementary characters, bringing the total number of characters to more than 65,536, the maximum number of combinations that can fit in 16 bits. This means that a value no longer has a one-to-one mapping to the fundamental semantic unit in Unicode. JDK 5 was updated to support the larger set of character values. Instead of changing the definition of the `char` type, some of the new supplementary characters are represented by a surrogate pair of two `char` values. To reduce naming confusion, a code point will be used to refer to the number that represents a particular Unicode character, including supplementary ones.

Finally why `forEachOrdered` and not `forEach`. ?

The behaviour of `forEach` is explicitly nondeterministic where as the `forEachOrdered` performs an action for each element of this stream, in the **encounter order of the stream** if the stream has a defined encounter order. `forEach` does not guarantee that the order would be kept. Also check this question for more.

For **difference between a character, a code point, a glyph and a grapheme** check this question.

share · improve this answer — answered Oct 19 '17 at 6:44 — Ankur Shanbhag — edited Oct 19 '17 at 7:04

- Jens Schauder Jens Schauder — Jens Oct 19 '17 at 7:04

add a comment

See The Java Tutorials: Strings.

```
public class StringDemo {
    public static void main(String[] args) {
        String[] anArray = { "Hello", "World",
            int len = palindrome.length();
            char[] tempCharArray = new char[len];
            char[] charArray = new char[len];
            // put original string in an array of chars
            for (int i = 0; i < len; i++) {
                tempCharArray[i] = palindrome.charAt(i);
            }
            // reverse array
            int j = len - 1;
            int tmpLen = len / 2;
            for (int i = 0; i < tmpLen; i++) {
                charArray[i] = tempCharArray[j--];
            }
            String reversePalindrome = new String(charArray);
            System.out.println(reversePalindrome);
    }
}
```

Put the length into `int len` and use `for` loop.

share · improve this answer — answered Oct 19 '08 at 14:24 — Eugene Yokota — edited Mar 1 '18 at 17:25

- I'm starting to feel a bit spoiled here... if there's such a word. I But this solution also has the problem outlined here. This has the same problem outlined here. stackoverflow.com/questions/196830 ... — Emmanuel Ogeh Oct 10 '14 at 21:08

add a comment

StringTokenizer is totally unsuited to the task of breaking a string into its individual characters. With StringBuilder[1] you can do that easily by using a regex that matches nothing, e.g.:

```
String[] theChars = str.split("(?=.)");
```

But StringTokenizer doesn't use regexes, and there's no delimiter string you can specify that will match the nothing between characters. There is one rule little hack you can use to accomplish the task, though: you can specify the empty string (meaning the delimiter string character is the delimiter) and have it return the delimiter:

```
StringTokenizer st = new StringTokenizer(str, str, true);
```

However, I only mention these options for the purpose of demeaning them. Both techniques break down if you need to deal with supplementary characters (Unicode characters outside the range that fits into 16 bits). Instead, use the method of object creation and string manipulation. Compare that to using charAt(), in a for loop, which moves virtually no overhead.

share · improve this answer — answered Oct 19 '08 at 15:24 — Alan Moore

- Alan Moore Alan Moore — Jens Oct 19 '08 at 15:24

add a comment

Elaborating on this answer and this answer:

Above answers point out the problem of many of the solutions here which don't iterate by code point but by code unit (most flexible with any surrogate chars). The best place also confirms the issue from case "Unicode Character Representations". Anyhow, here's some code that uses some actual surrogate chars from the supplementary Unicode set, and converts them back to a String. Note that the output of the code shows that the chars are indeed surrogates, you'll necessarily have two chars. The code should work for any Unicode character.

```
String supplementary = "Some Supplementary: U0070B";
supplementary.codePoint(0).toCharArray();
    System.out.print(new StringCharacterIterator(c));
```

share · improve this answer — answered Dec 19 '14 at 19:08 — Community ♦ — 1 revision

- Christopher Parker Christopher Parker — Community ♦ Community ♦ — 1 revision

add a comment

This Example Code will help you out!

```
import java.util.Iterator;
import java.lang.*;
import java.io.PrintWriter;
import java.io.IOException;

public class Iterations {
    public static void main(String[] args) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        map.put("a", 001);
        map.put("b", 002);
        map.put("c", 003);
        map.put("d", 004);
        System.out.println(map);

        for (Character ch : getAllCharsIn(map)
            System.out.print(ch+"\n");
    }

    public static Iterable<Character> getAllCharsIn(
        for (Character ch : new TreeSet(new ToCharComparator(
            if (ch.toLowerCase(ch).equals(
            return ch+"";
        }
    }
}

class ToCharComparator implements Comparator {
    Map map;

    public ToCharComparator(Map m) {
        this.map = map;
    }

    public int compare(Object lhs, Object rhs) {
        Integer lhsValue = (Integer)map.get(lhs);
        Integer rhsValue = (Integer)map.get(rhs);
    }
}
```

share · improve this answer — answered Dec 17 '10 at 6:24 — Christopher Parker — edited Dec 17 '10 at 6:29

add a comment

Your Answer

Sign up or Log in
Sign up using Google
Sign up using Facebook
Sign up using Email and Password
Post as a guest
Name
Email
Required, but never shown
Post Your Answer
By posting your answer, you agree to the privacy policy and terms of service.

Not the answer you're looking for? Browse other questions tagged: java string iteration character
Interesting or ask your own question.

site design / logo © 2014 Stack Exchange Inc; user contributions licensed under cc by-sa 3.0 with attribution required. rev 2014.12.23.2141