ABOUT BYRON KIOURTZOGLOU

Byron is a master software engineer working in the IT and Telecom domains. He is an applications developer in a wide variety of applications/services. He is currently acting as the team leader and technical architect for a proprietary service creation and integration platform for both the IT and Telecom industries in addition to a in-house big data real-time analytics solution. He is always fascinated by SOA, middleware services and mobile development. Byron is co-founder and Executive Editor at Java Code Geeks.

Java Best Practices – Char to Byte and Byte to Char conversions

Posted by: Byron Kiourtzoglou in Core Java November 8th, 2010 5 Comments 3446 Views

Continuing our series of articles concerning proposed practices while working with the Java programming language, we are going to talk about String performance tunning. Especially we will focus on how to handle character to byte and byte to character conversions efficiently when the default encoding is used. This article concludes with a performance comparison between two proposed custom approaches and two classic ones (the "*String.getBytes()*" and the *NIO* ByteBuffer) for converting characters to bytes and vice – versa.

All discussed topics are based on use cases derived from the development of mission critical, ultra high performance production systems for the telecommunication industry.

Prior reading each section of this article it is highly recommended that you consult the relevant Java API documentation for detailed information and code samples.

All tests are performed against a Sony Vaio with the following characteristics :

- System : openSUSE 11.1 (x86_64)
- Processor (CPU) : Intel(R) Core(TM)2 Duo CPU T6670 @ 2.20GHz
- Processor Speed : 1,200.00 MHz
- Total memory (RAM) : 2.8 GB
- Java : OpenJDK 1.6.0_0 64-Bit

The following test configuration is applied :

- Concurrent worker Threads : 1
- Test repeats per worker Thread : 1000000
- Overall test runs : 100

Char to Byte and Byte to Char conversions

Character to byte and byte to character conversions are considered common tasks among Java developers who are programming against a networking environment, manipulate streams of byte data, serialize String objects, implementing communication protocols etc. For that reason Java provides a handful of utilities that enable a developer to convert a String (or a character array) to its byte array equivalent and vice versa.

The "*getBytes(charsetName)*" operation of the String class is probably the most commonly used method for converting a String into its byte array equivalent. Since every character can be represented differently according to the encoding scheme used, its of no surprise that the aforementioned operation requires a "*charsetName*" in order to correctly convert the String characters. If no "*charsetName*" is provided, the operation encodes the String into a sequence of bytes using the platform's default character set.

Another "classic" approach for converting a character array to its byte array equivalent is by using the ByteBuffer class of the NIO package. An example code snippet for the specific approach will be provided later on.

Both the aforementioned approaches although very popular and indisputably easy to use and straightforward greatly lack in performance compared to more fine grained methods. Keep in mind that ***we are not converting between character encodings***. For converting between character encodings you should stick with the "classic" approaches using either the "*String.getBytes(charsetName)*" or the *NIO* framework methods and utilities.

When all characters to be converted are ASCII characters, a proposed conversion method is the one shown below :

```java
public static byte[] stringToBytesASCII(String str) {
 char[] buffer = str.toCharArray();
 byte[] b = new byte[buffer.length];
 for (int i = 0; i < b.length; i++) {
  b[i] = (byte) buffer[i];
 }
 return b;
}
```

The resulted byte array is constructed by casting every character value to its byte equivalent since we know that all characters are in the ASCII range (0 – 127) thus can occupy just one byte in size.

Using the resulted byte array we can convert back to the original String, by utilizing the “classic” String constructor “*new String(byte[])*”

For the default character encoding we can use the methods shown below to convert a String to a byte array and vice – versa :

```java
public static byte[] stringToBytesUTFCustom(String str) {
 char[] buffer = str.toCharArray();
 byte[] b = new byte[buffer.length << 1];
 for(int i = 0; i < buffer.length; i++) {
  int bpos = i << 1;
  b[bpos] = (byte) ((buffer[i]&0xFF00)>>8);
  b[bpos + 1] = (byte) (buffer[i]&0x00FF);
 }
 return b;
}
```

Every character type in Java occupies 2 bytes in size. For converting a String to its byte array equivalent we convert every character of the String to its 2 byte representation.

Using the resulted byte array we can convert back to the original String, by utilizing the method provided below :

```java
public static String bytesToStringUTFCustom(byte[] bytes) {
 char[] buffer = new char[bytes.length >> 1];
 for(int i = 0; i < buffer.length; i++) {
  int bpos = i << 1;
  char c = (char)(((bytes[bpos]&0x00FF)<<8) + (bytes[bpos+1]&0x00FF));
  buffer[i] = c;
 }
 return new String(buffer);
}
```

We construct every String character from its 2 byte representation. Using the resulted character array we can convert back to the original String, by utilizing the “classic” String constructor “*new String(char[])*”

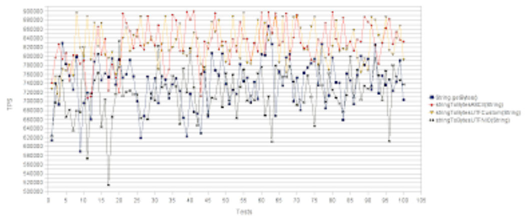
Last but not least we provide two example methods using the NIO package in order to convert a String to its byte array equivalent and vice – versa :

```java
public static byte[] stringToBytesUTFNIO(String str) {
 char[] buffer = str.toCharArray();
 byte[] b = new byte[buffer.length << 1];
 CharBuffer cBuffer = ByteBuffer.wrap(b).asCharBuffer();
 for(int i = 0; i < buffer.length; i++)
  cBuffer.put(buffer[i]);
 return b;
}
```

```java
public static String bytesToStringUTFNIO(byte[] bytes) {
 CharBuffer cBuffer = ByteBuffer.wrap(bytes).asCharBuffer();
 return cBuffer.toString();
}
```

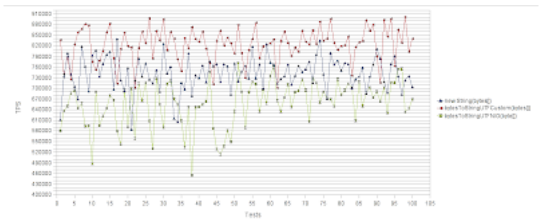
For the final part of this article we provide the performance comparison charts for the aforementioned String to byte array and byte array to String conversion approaches. We have tested all methods using the input string “***a test string***”.

First the String to byte array conversion performance comparison chart :



The horizontal axis represents the number of test runs and the vertical axis the average transactions per second (TPS) for each test run. Thus higher values are better. As expected, both “*String.getBytes()*” and “*stringToBytesUTFNIO(String)*” approaches performed poorly compared to the “*stringToBytesASCII(String)*” and “*stringToBytesUTFCustom(String)*” suggested approaches. As you can see, our proposed methods achieve almost 30% increase in TPS compared to the “classic” methods.

Lastly the byte array to String performance comparison chart :



The horizontal axis represents the number of test runs and the vertical axis the average transactions per second (TPS) for each test run. Thus higher values are better. As expected, both "*new String(byte[])*" and "*bytesToStringUTFNIO(byte[])*" approaches performed poorly compared to the "*bytesToStringUTFCustom(byte[])*" suggested approach. As you can see, our proposed method achieved almost 15% increase in TPS compared to the "*new String(byte[])*" method, and almost 30% increase in TPS compared to the "*bytesToStringUTFNIO(byte[])*" method.

In conclusion, when you are dealing with character to byte or byte to character conversions and you do not intent to change the encoding used, you can achieve superior performance by utilizing custom – fine grained – methods rather than using the "classic" ones provided by the String class and the *NIO* package. Our proposed approach achieved an overall of 45% increase in performance compared to the "classic" approaches when converting the test String to its byte array equivalent and vice – versa.

Happy coding

Justin

P.S.

After taking into consideration the proposition from several of our readers to utilize the "*String.charAt(int)*" operation instead of using the "*String.toCharArray()*" so as to convert the String characters into bytes, I altered our proposed methods and re-executed the tests. As expected, further performance gains where achieved. In particular, an ***extra*** 13% average increase in TPS was recorded for the "*stringToBytesASCII(String)*" method and an ***extra*** 2% average increase in TPS was recorded for the "*stringToBytesUTFCustom(String)*". So you should use the altered methods as they perform even better than the original ones. The updated methods are shown below :

```java
public static byte[] stringToBytesASCII(String str) {
 byte[] b = new byte[str.length()];
 for (int i = 0; i < b.length; i++) {
  b[i] = (byte) str.charAt(i);
 }
 return b;
}
```

```java
public static byte[] stringToBytesUTFCustom(String str) {
 byte[] b = new byte[str.length() << 1];
 for(int i = 0; i < str.length(); i++) {
  char strChar = str.charAt(i);
  int bpos = i << 1;
  b[bpos] = (byte) ((strChar&0xFF00)>>8);
  b[bpos + 1] = (byte) (strChar&0x00FF);
 }
 return b;
}
```

Related Articles :

- Java Best Practices – DateFormat in a Multithreading Environment
- Java Best Practices – High performance Serialization
- Java Best Practices – Vector vs ArrayList vs HashSet
- Java Best Practices – String performance and Exact String Matching
- Java Best Practices – Queue battle and the Linked ConcurrentHashMap

Tagged with: BYTE ARRAY CHARACTER JAVA BEST PRACTICES STRING

(***0*** *rating,* ***0*** *votes*)

You need to be a registered member to rate this. 5 Comments 3446 Views Tweet it!

I agree to the Terms and Privacy Policy

Sign up

LIKE A CODE GEEKS

5

Join the discussion...

4 1 0 5

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Subscribe

newest oldest most voted

Mehmet Nuri Deveci

Guest

Thanks for the article but I have a question.

I have tested your last stringToBytesUTFCustom method. It works much more faster than String.getBytes(Charset). But the problem is, I am using some unicode characters ('u258E' and 'u0374') and it produces invalid data. For example:

String.getBytes(UTF-8) [-30, -106, -114, -51, -76]
stringToBytesUTFCustom [37, -114, 3, 116]

Am I missing something here?

0 Reply 6 years ago

Byron Kiourtzoglou

Guest

Hello Mehmet,

Our custom method produces a byte array using the default character encoding in Java which utilizes a 2 bytes per character scheme. By using UTF-8 encoding a single character can occupy up to 4 bytes of data. Thats why the two methods return different number of bytes for the same two unicode characters. Nevertheless If you examine the returned byte array for our stringToBytesUTFCustom() method you will see that the resulted bytes are correct!

BRs

0 Reply 6 years ago

Duarte

Guest

Hi Byron, I've a similar implementation so that I can hold a stream of bytes in a String, and then convert it back to bytes. The objective would be to not change in any case the data. Your implementation has a problem if the number of bytes is odd (1 byte will be cut off). My solution was to pad the last char with 0x00 if the number of bytes is odd, and have always a last char signaling if the data is padded or not. This has the disadvantage of using always 2 extra bytes (1 char), and to… Read more »

0 Reply 5 years ago

John DeRegnaucourt

Guest

Bryon,

In your final versions, you have a for loop like this:

for(int i = 0; i < str.length(); i++)

The str.length() method is called on every iteration. You can see this by looking at the bytecode (Intellij now makes that easy).

If you want additional speed, use:

```
int len = str.length();
for(int i = 0; i < len; i++) ...
```

This removes a method call per every iteration of the loop.

+ 0 − Reply 3 years ago

Guest

4sskick

hi I want to ask about snippet code I got from internet,

in some parts I found something like this

```
String input ==> from user
char[] stat = new char[256];
for (int i = 0; i < input.length(); i++) {
stat[input.charAt(i)]++;
}
```

I don't get it what exactly "stat[input.charAt(i)]++;" piece of code, I was tried to print value inside array stat[] but nothing. Someone please give me explanation what exactly the does

+ 0 − Reply 2 years ago

KNOWLEDGE BASE

- Courses
- Examples
- Minibooks
- Resources
- Tutorials

PARTNERS

- Mkyong

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

HALL OF FAME

- "Android Full Application Tutorial" series
- 11 Online Learning websites that you should check out
- Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons
- Android Google Maps Tutorial
- Android JSON Parsing with Gson Tutorial
- Android Location Based Services Application – GPS location
- Android Quick Preferences Tutorial
- Difference between Comparator and Comparable in Java
- GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
- Java Best Practices – Vector vs ArrayList vs HashSet

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creat
ultimate Java to Java developers resource center; targeted at the technical ar
technical team lead (senior developer), project manager and junior developer
JCGs serve the Java, SOA, Agile and Telecom communities with daily news wri
domain experts, articles, tutorials, reviews, announcements, code snippets an
source projects.

DISCLAIMER