Enough java.lang.String to Hang Ourselves ...

Dr Heinz M. Kabutz

Last Updated 2018-06-19

Javaspecialists.eu
java training

Converting `int val` to a `String`?

1. **"" + val**
2. **Integer.toString(val)**
3. **Integer.valueOf(val)**
4. **String.valueOf(val)**
5. **Integer.getInteger(val)**

Which do you think is fastest?

```
public class StringAppender {
  public static String basic(String s1, String s2, String s3) {
    return "SELECT " + s1 + " FROM " + s2 + " WHERE " + s3;
  }

  public static String stringBuilder(String s1, String s2, String s3) {
    return new StringBuilder().append("SELECT ").append(s1)
        .append(" FROM ").append(s2).append(" WHERE ")
        .append(s3).toString();
  }

  public static String stringBuilderSize(String s1, String s2, String s3) {
    int len = 20 + s1.length() + s2.length() + s3.length();
    return new StringBuilder(len).append("SELECT ").append(s1)
        .append(" FROM ").append(s2).append(" WHERE ")
        .append(s3).toString();
  }
}
```

When the Dinosaurs Roamed the Earth - Java 1.0

- **Fields:**
  - **private char value[];**
  - **private int offset;**
  - **private int count;**
- **hashCode() used samples of chars if String was longer than 16**
- **equals() did not check if obj == this**
- **intern() used a static HashSet**
  - **Memory Leak**
- **StringBuffer a modifiable, thread-safe version**
  - **toString() shared the underlying char[] unless it was later modified**

hashCode() in String 1.0

```
public int hashCode() {
    int h = 0;
    int off = offset;
    char val[] = value;
    int len = count;

    if (len < 16) {
        for (int i = len ; i > 0; i--) {
            h = (h * 37) + val[off++];
        }
    } else {
        // only sample some characters
        int skip = len / 8;
        for (int i = len ; i > 0; i -= skip, off += skip) {
            h = (h * 39) + val[off];
        }
    }
    return h;
}
```

Early Hunter Gatherer - Java 1.1

- **Fields stayed the same**
- **hashCode() still sampling**
- **intern() moved to native code**
- **toUpperCase() added some weird edge cases such as ß ➜ SS**

Discovering Fire - Java 1.2

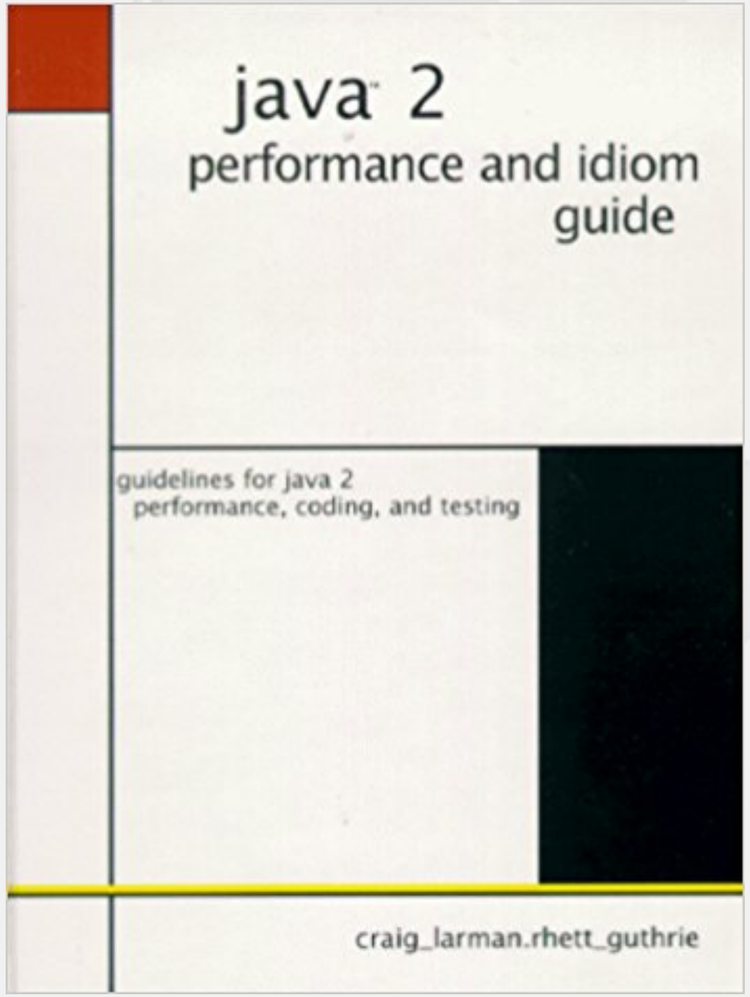
- **Fields still the same**
- **hashCode() changed to**

```
public int hashCode() {
    int h = 0;
    int off = offset;
    char val[] = value;
    int len = count;

    for (int i = 0; i < len; i++)
        h = 31*h + val[off++];

    return h;
}
```

  - **Broke a bunch of code**
- **Introduced the Comparable interface**

Java 2 Performance and Idiom Guide

- **Proposed wrapping String with own object and caching hash code**



Stone Age - Java 1.3

- **Fields:**
  - **private char value[];**
  - **private int offset;**
  - **private int count;**
  - **private int hash; ⬅**
- **So is String really immutable?**

```
public int hashCode() {
    int h = hash;
    if (h == 0) {
        int off = offset;
        char val[] = value;
        int len = count;

        for (int i = 0; i < len; i++)
            h = 31*h + val[off++];
        hash = h;
    }
    return h;
}
```

Brief History Lesson of String - Java 1.4

- **Fields same as 1.3**
- **Introduced CharSequence interface**
- **Regular expressions**
  - **Methods like matches(), split(), etc.**

Before we go on ...

- **Adding Strings together**

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello " + args[0]);
  }
}
```

- **Became (Java 1.0 - 1.4)**

```
public class Hello {
  public static void main(String[] args) {
    System.out.println(new StringBuffer().append("Hello ")
      .append(args[0]).toString());
  }
}
```

- **new StringBuffer() would create an array of 16 characters**

Brief History Lesson of String - Java 1.5

- Fields same as 1.3, but marked final (except for hash)
- Code points introduced
  - 32-bit characters
- StringBuilder as unsynchronized StringBuffer
  - char[] no longer shared with created Strings
- Needed to recompile all code
  - And hand-crafted StringBuffer code would now typically be slower than +

Brief History Lesson of String - Java 1.6

- **Not much changed since 1.5**
- **-XX:+OptimizeStringConcat**
- **-XX:+UseCompressedStrings**
  - **byte[] when 7-bit ASCII**
  - **otherwise char[]**

Brief History Lesson of String - Java 1.7

- **Fields:**
  - **private final char value[];**
  - **private int hash;**
  - **private transient int hash32 = 0; // used to avoid DOS attacks on HashMap**
- **new constructor String(char[], boolean unshared)**
  - **SharedSecrets.getJavaLangAccess().newStringUnsafe(char[])**
    - **Demo**
    - **Moved out of harm's way since Java 9**
- **String.substring() now created new char[]s**
  - **SubbableString alternative - demo**
  - **Newsletter 230 - https://www.javaspecialists.eu/archive/Issue230.html**

Brief History Lesson of String - Java 1.8

- **Fields:**
  - **private final char value[];**
  - **private int hash;**
- **static methods for joining several Strings**
- **Deduplication of char[]s**
- **Hash Maps use trees in case of too many bucket collisions**

String Deduplication

- **Java 1.8.0_20 can replace char[]s of duplicate strings**
  - **Only works for the G1 collector -XX:+UseStringDeduplication**
  - **Threshold when deduplicated -XX:StringDeduplicationAgeThreshold**

```
public class DeduplicationDemo {
  public static void main(String... args) throws Exception {
    char[] heinz = {'h', 'e', 'i', 'n', 'z'};
    String[] s = {new String(heinz), new String(heinz),};
    Field value = String.class.getDeclaredField("value");
    value.setAccessible(true);
    System.out.println("Before GC");
    System.out.println(value.get(s[0]));
    System.out.println(value.get(s[1]));
    System.gc(); Thread.sleep(100);
    System.out.println("After GC");
    System.out.println(value.get(s[0]));
    System.out.println(value.get(s[1]));
  }
}
```

```
Before GC
[C@76ed5528
[C@2c7b84de
After GC
[C@2c7b84de
[C@2c7b84de
```

Brief History Lesson of String - Java 9 / 10

- **Fields:**
  - **private final byte value[];**
  - **private final byte coder;**
  - **private int hash;**
- **Latin1 are one byte per character, everything else two bytes**
  - **Most methods are a lot more complicated**
- **chars() returns an IntStream of contents of String**
- **+ is no longer compiled to StringBuilder**
  - **StringConcatFactory**
  - **Demo and look at benchmarks: https://github.com/kabutz/string-performance**

Intrinsics in Java 8 (https://github.com/apangin)

```
_compareTo                    int String.compareTo(String)
_indexOf                      int String.indexOf(String)
_equals                       boolean String.equals(Object)
_String_String                String(String)
_StringBuilder_void           StringBuilder()
_StringBuilder_int            StringBuilder(int)
_StringBuilder_String         StringBuilder(String)
_StringBuilder_append_char    StringBuilder StringBuilder.append(char)
_StringBuilder_append_int     StringBuilder StringBuilder.append(int)
_StringBuilder_append_String  StringBuilder StringBuilder.append(String)
_StringBuilder_toString       String StringBuilder.toString()

// similarly for StringBuffer
```

Intrinsics in Java 9

```
_compressStringC              StringUTF16.compress([CI[BII)I
_compressStringB              StringUTF16.compress([BI[BII)I
_inflateStringC               StringLatin1.inflate([BI[CII)V
_inflateStringB               StringLatin1.inflate([BI[BII)V
_toBytesStringU               StringUTF16.toBytes([CII)[B
_getCharsStringU              StringUTF16.getChars([BII[CI)V
_getCharStringU               StringUTF16.getChar([BI)C
_putCharStringU               StringUTF16.putChar([BII)V
_compareToL                   StringLatin1.compareTo([B[B)I
_compareToU                   StringUTF16.compareTo([B[B)I
_compareToLU                  StringLatin1.compareToUTF16([B[B)I
_compareToUL                  StringUTF16.compareToLatin1([B[B)I
_indexOfL                     StringLatin1.indexOf([B[B)I
_indexOfU                     StringUTF16.indexOf([B[B)I
_indexOfUL                    StringUTF16.indexOfLatin1([B[B)I
_indexOfIL                    StringLatin1.indexOf([BI[BII)I
_indexOfIU                    StringUTF16.indexOf([BI[BII)I
_indexOfIUL                   StringUTF16.indexOfLatin1([BI[BII)I
_indexOfU_char                StringUTF16.indexOfChar([BIII)I
_equalsL                      StringLatin1.equals([B[B)Z
_equalsU                      StringUTF16.equals([B[B)Z
_String_String                String.<init>(LString;)V
_hasNegatives                 StringCoding.hasNegatives([BII)Z
_encodeByteISOArray           StringCoding.implEncodeISOArray([BI[BII)I
_StringBuilder_void           StringBuilder.<init>()V
_StringBuilder_int            StringBuilder.<init>(I)V
_StringBuilder_String         StringBuilder.<init>(LString;)V
_StringBuilder_append_char    StringBuilder.append(C)LStringBuilder;
_StringBuilder_append_int     StringBuilder.append(I)LStringBuilder;
_StringBuilder_append_String  StringBuilder.append(LString;)LStringBuilder;
_StringBuilder_toString       StringBuilder.toString()LString;
```

Biggest Constant String?

- **What is the longest constant String "..." that Java can support?**
- **Let's try it out**

intern()

- Constant Strings in different classes point to same object
  - We can use the same constant table with intern()
- Debugging intern table
  - View events with -XX:+PrintStringTableStatistics
  - Extend table with -XX:StringTableSize=n
    - 1009 up until Java 6, 60013
  - jcmd in Java 9 can show details with VM.stringtable

Lessons from Today

- Use + instead of StringBuilder where possible
  - += still needs the StringBuilder
- Avoid intern() in your code
  - use String Deduplication instead
- Hashing on Strings can be particularly expensive
  - Especially if the hash resolves to 0
- Strings in Java 9 use byte[]
  - Might use less memory. Shorter maximum String if not Latin1