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

Dr Heinz M. Kabutz, Dmitry Vyazelenko

geecon

Kraków, 15-17 May 2019

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?

```
String appendBasic(String question, String answer1, String answer2) {
  return "<h1>" + question + "</h1><ol><li>" + answer1 +
      "</li><li>" + answer2 + "</li></ol>";
}
String appendStringBuilder(String question, String answer1, String answer2) {
  return new StringBuilder().append("<h1>").append(question)
      .append("</h1><ol><li>").append(answer1)
      .append("</li><li>").append(answer2)
      .append("</li></ol>").toString();
}
String appendStringBuilderSize(String question, String answer1, String answer2) {
  int len = 36 + question.length() + answer1.length() + answer2.length();
  return new StringBuilder(len).append("<h1>").append(question)
      .append("</h1><ol><li>").append(answer1)
      .append("</li><li>").append(answer2)
      .append("</li></ol>").toString();
}
String appendFormat(String question, String answer1, String answer2) {
  return String.format("<h1>%s</h1><ol><li>%s</li><li>%s</li></ol>",
      question, answer1, answer2);
}
```

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 Hashtable**
  - **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;
}
```

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

Dr Heinz M. Kabutz, ~~Dmitry Vyazelenko~~

geecon

Kraków, 15-17 May 2019

Who's Who

- **Dmitry Vyazelenko @DVyazelenko**
  - **Software engineer, JCrete Chief Disorganizer**
- **Heinz Kabutz @heinzkabutz**
  - **JCrete Unfounder**
  - **Java Specialists Newsletter**
    - **www.javaspecialists.eu**

tinyurl.com/geecon-string

Early Hunter Gatherer - Java 1.1

- Fields stayed the same
- hashCode() still sampling
- intern() moved to native code
  - Not necessarily better than Java
- toUpperCase() added some weird edge cases such as ß ➜ SS

tinyurl.com/geecon-string

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**

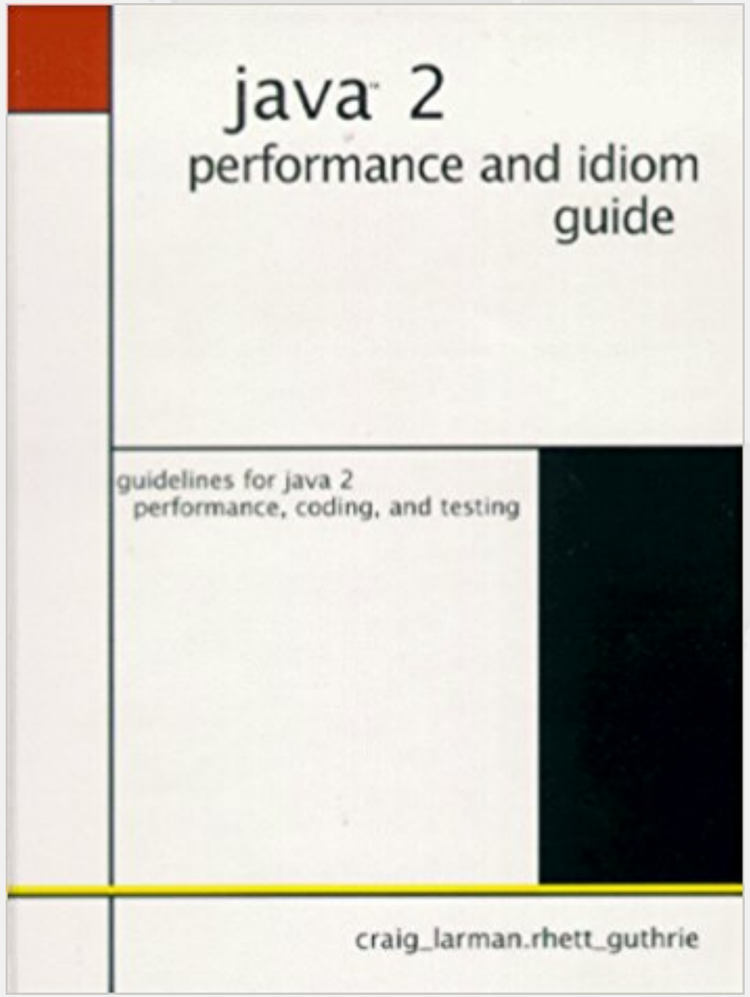
Old Hash vs New Hash Calculation Performance

- **Java 1.0 and 1.1 calculation was O(1) - constant time**
- **Java 1.2 calculation is O(n) - linear time**

tinyurl.com/geecon-string

Java 2 Performance and Idiom Guide

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



Why Caching Hash Codes Seldom Helps

- How long to put() 30x?
- How long to get() 30x?
  - If you are going to keep keys for maps around, why not just keep the values?

```
public class Key {
  private final int id;

  public Key(int id) { this.id = id; }

  public int hashCode() {
    try {
      Thread.sleep(100); // now THIS is slow
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
    return id;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof Key)) return false;
    return id == ((Key)obj).id;
  }
}
```

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;
}
```

What Do These Strings Have in Common?

```
"ARbyguv", "ARbygvW", "ARbyhVv", "ARbyhWW", "ARbzHuv", "ARbzHvW", "ARbzIVv", "ARbzIWW",
"ARcZguv", "ARcZgvW", "ARcZhVv", "ARcZhWW", "ASCyguv", "ASCygvW", "ASCyhVv", "ASCyhWW",
"ASCzHuv", "ASCzHvW", "ASCzIVv", "ASCzIWW", "ASDZguv", "ASDZgvW", "ASDZhVv", "ASDZhWW",
"bmgkAEs", "bmgkAFT", "bmhLAEs", "bmhLAFT", "bnHkAEs", "bnHkAFT", "bnILAEs", "bnILAFT",
"cNgkAEs", "cNgkAFT", "cNhLAEs", "cNhLAFT", "cOHkAEs", "cOHkAFT", "cOILAEs", "cOILAFT",
"Elcnfnz", "ElcngOz", "ElcoGnz", "ElcoHOz", "EldOfnz", "EldOgOz", "EldPGnz", "EldPHOz",
"EmDnfnz", "EmDngOz", "EmDoGnz", "EmDoHOz", "EmEOfnz", "EmEOgOz", "EmEPGnz", "EmEPHOz",
"FMcnfnz", "FMcngOz", "FMcoGnz", "FMcoHOz", "FMdOfnz", "FMdOgOz", "FMdPGnz", "FMdPHOz",
"FNDnfnz", "FNDngOz", "FNDoGnz", "FNDoHOz", "FNEOfnz", "FNEOgOz", "FNEPGnz", "FNEPHOz",
"Obdwdac", "ObdwdbD", "ObdweBc", "ObdweCD", "ObdxEac", "ObdxEbD", "ObdxFBc", "ObdxFCD",
"ObeXdac", "ObeXdbD", "ObeXeBc", "ObeXeCD", "ObeYEac", "ObeYEbD", "ObeYFBc", "ObeYFCD",
"OcEwdac", "OcEwdbD", "OcEweBc", "OcEweCD", "OcExEac", "OcExEbD", "OcExFBc", "OcExFCD",
"OcFXdac", "OcFXdbD", "OcFXeBc", "OcFXeCD", "OcFYEac", "OcFYEbD", "OcFYFBc", "OcFYFCD",
"PCdwdac", "PCdwdbD", "PCdweBc", "PCdweCD", "PCdxEac", "PCdxEbD", "PCdxFBc", "PCdxFCD",
"PCeXdac", "PCeXdbD", "PCeXeBc", "PCeXeCD", "PCeYEac", "PCeYEbD", "PCeYFBc", "PCeYFCD",
"PDEwdac", "PDEwdbD", "PDEweBc", "PDEweCD", "PDExEac", "PDExEbD", "PDExFBc", "PDExFCD",
"PDFXdac", "PDFXdbD", "PDFXeBc", "PDFXeCD", "PDFYEac", "PDFYEbD", "PDFYFBc", "PDFYFCD",
"Xwfaark", "XwfaasL", "XwfabSk", "XwfabTL", "XwfbBrk", "XwfbBsL", "XwfbCSk", "XwfbCTL",
"XwgBark", "XwgBasL", "XwgBbSk", "XwgBbTL", "XwgCBrk", "XwgCBsL", "XwgCCSk", "XwgCCTL",
"XxGaark", "XxGaasL", "XxGabSk", "XxGabTL", "XxGbBrk", "XxGbBsL", "XxGbCSk", "XxGbCTL",
"XxHBark", "XxHBasL", "XxHBbSk", "XxHBbTL", "XxHCBrk", "XxHCBsL", "XxHCCSk", "XxHCCTL",
"zsjpxah", "zsjpxbI", "zsjpyBh", "zsjpyCI", "zsjqYah", "zsjqYbI", "zsjqZBh", "zsjqZCI",
```

All Those Strings Have hashCode() == 0

- Plus any combination of these Strings also have hashCode of 0
  - Thus we can produce an endless sequence of such Strings
    - "zsjpyCIcOHkAEsObeXeCDASCzlVv".hashCode() == 0
- Why is this so bad?
- github.com/kabutz/string-performance
  - eu.javaspecialists.playground.hasher.StringDOS

Bucket Collisions

- Can attack server by sending lots of Strings with same hashCode
  - Very easy to do when == 0
- Both put() and get() become linear

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:+UseCompressedStrings
  - byte[] when 7-bit ASCII
  - otherwise char[]
- -XX:+OptimizeStringConcat
  - char[] could in some cases be shared between StringBuilder/Buffer and String

Quiz 2: StringAppenderBenchmark.append Strings

	1.6.0_113	1.7.0_191	1.8.0_172	11
appendBasic	61 ns/op 208 B/op	56 ns/op 200 B/op	58 ns/op 200 B/op	75 ns/op 120 B/op
appendStringBuilder	61 ns/op 208 B/op	56 ns/op 200 B/op	58 ns/op 200 B/op	75 ns/op 120 B/op
appendStringBuilderSize	57 ns/op 208 B/op	57 ns/op 200 B/op	58 ns/op 200 B/op	75 ns/op 120 B/op

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**

-Djdk.map.althashing.threshold=512

- **new constructor String(char[], boolean unshared)**
  - **SharedSecrets.getJavaLangAccess().newStringUnsafe(char[])**
    - **Moved out of harm's way since Java 9**
- **String.substring() now created new char[]s**
  - **SubbableString alternative**
  - **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
```

Deduplication vs intern() vs Roll Own

- new String("Hello World!") in Java 8 64-bit compressed OOPS
  - String: 12 (header) + 4 (value) + 4 (hash) ≈ 24 bytes
  - char[ ]: 12 (header) + 24 (12 characters) + 4 (length) = 40 bytes
  - Total: 64 bytes
- Deduplication saves 40 bytes automagically
- intern() saves 64 bytes, but at high cost
  - Until Java 10, intern table did not grow
  - jcmd in Java 9+ can show details with VM.stringtable
- Own ConcurrentHashMap with putIfAbsent(s, s) saves 64 bytes
  - But potential memory leak as unused Strings never deleted

Brief History Lesson of String - Java 9 / 10 / 11 / 12

- **Fields:**
  - **private final byte[] value;**
  - **private final byte coder;**
  - **private int hash;**

Indify String Concatenation Java 9+

- **+ is no longer compiled to StringBuilder**
  - **StringConcatFactory**
  - **Demo and look at benchmarks: https://github.com/kabutz/string-performance**

StringAppenderBenchmark Mixed Parameters

	1.6.0_113	1.7.0_191	1.8.0_172	11
plus	319 ns/op 896 B/op	317 ns/op 864 B/op	333 ns/op 864 B/op	127 ns/op 152 B/op
sb_sized	220 ns/op 536 B/op	230 ns/op 504 B/op	245 ns/op 504 B/op	259 ns/op 280 B/op
sb	320 ns/op 896 B/op	316 ns/op 864 B/op	332 ns/op 864 B/op	303 ns/op 488 B/op
concat	644 ns/op 2024 B/op	576 ns/op 1712 B/op	590 ns/op 1664 B/op	368 ns/op 960 B/op
format	4088 ns/op 3560 B/op	3541 ns/op 3504 B/op	3208 ns/op 3304 B/op	3855 ns/op 1896 B/op

JEP 280: Indify String Concatenation

- Uses invokedynamic for String concatenation
- Bytecode generator
  - BC_SB - like old Java 5 + concatenation
  - BC_SB_SIZED
  - BC_SB_SIZED_EXACT
- MethodHandles
  - MH_SB_SIZED
  - MH_SB_SIZED_EXACT
  - MH_INLINE_SIZED_EXACT (default)
    - Converts non-primitives, float and double to String using StringifierMost
    - Uses StringConcatHelper#mixLen to compute exact sizes for other primitives

StringAppenderBenchmark.plus

- **-Djava.lang.invoke.stringConcat=...**

	plus with mixed values
MH_INLINE_SIZED_EXACT	127 ns/op, 152 B/op
BC_SB_SIZED_EXACT	157 ns/op, 208 B/op
MH_SB_SIZED	251 ns/op, 328 B/op
BC_SB_SIZED	255 ns/op, 328 B/op
MH_SB_SIZED_EXACT	290 ns/op, 408 B/op
BC_SB	301 ns/op, 512 B/op

JEP 254: Compact Strings

- char[] replaced with byte[]
- Saves space if characters fit into a byte (i.e. Latin1)
- kill switch -XX:-CompactStrings
- Max String length is now half of what it was
  - Whether compact Strings disabled or not Latin1 String

StringAppenderBenchmark +/- CompactStrings

	+CompactStrings	-CompactStrings
plus	127 ns/op, 152 B/op	135 ns/op, 264 B/op
sb_sized	259 ns/op, 304 B/op	247 ns/op, 528 B/op
sb	302 ns/op, 512 B/op	316 ns/op, 888 B/op
concat	369 ns/op, 1224 B/op	408 ns/op, 1928 B/op
format	3855 ns/op, 1872 B/op	3647 ns/op, 2296 B/op

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 11

```
_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;
```

java.lang.String#equals

```
public boolean equals(Object anObject) {
  if (this == anObject) {
    return true;
  }
  if (anObject instanceof String) {
    String aString = (String)anObject;
    if (coder() == aString.coder()) {
      return isLatin1() ? StringLatin1.equals(value, aString.value)
                        : StringUTF16.equals(value, aString.value);
    }
  }
  return false;
}
```

java.lang.StringLatin1#equals

```
@HotSpotIntrinsicCandidate
public static boolean equals(byte[] value, byte[] other) {
  if (value.length == other.length) {
    for (int i = 0; i < value.length; i++) {
      if (value[i] != other[i]) {
        return false;
      }
    }
    return true;
  }
  return false;
}
```

Using Arrays.mismatch()

```
// Probably how it is implemented in the intrinsic
private static boolean equals(byte[] value, byte[] other) {
  if (value.length == other.length) {
    return Arrays.mismatch(value, other) == -1;
  }
  return false;
}
```

Note: Deduplication not taken into account with String.equals

	length=4	length=16	length=64	length=256
string_equals	7 ns/op	9 ns/op	8 ns/op	15 ns/op
mismatch_equals	8 ns/op	9 ns/op	11 ns/op	19 ns/op
hand_rolled	7 ns/op	14 ns/op	29 ns/op	97 ns/op

Lessons from Today

Please vote

- **Use + instead of StringBuilder where possible**
  - **Recompile classes for Java 9+**
  - **In loops still use StringBuilder.append()**
- **Avoid intern() in your code**
  - **use String Deduplication or own cache instead**
- **Hashing on Strings can be particularly expensive**
  - **Especially dangerous if the hash resolves to 0**
- **Strings since Java 9 use byte[]**
  - **Might use less memory. Shorter maximum String if not Latin1**

`tinyurl.com/geecon-string`

Quiz 1: NumberToStringBenchmark

	1.6.0_113	1.7.0_191	1.8.0_172	11
int_plus	45 ns/op 72 B/op	50 ns/op 64 B/op	53 ns/op 64 B/op	**33 ns/op** **56 B/op**
int_toString	61 ns/op 72 B/op	60 ns/op 64 B/op	59 ns/op 64 B/op	**34 ns/op** **56 B/op**
long_plus	122 ns/op 248 B/op	118 ns/op 216 B/op	127 ns/op 216 B/op	**53 ns/op** **64 B/op**
long_toString	85 ns/op 88 B/op	63 ns/op 80 B/op	65 ns/op 80 B/op	**53 ns/op** **64 B/op**