ORACLE

The Lord of the Strings

«How I spent my summer»

Aleksey Shipilëv
aleksey.shipilev@oracle.com, @shipilev

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Intro

Intro: Expectations

This talk is mostly about the JDK/JVM development experience for 2 features:

- **Compact Strings**: represent ASCII Strings with 1 byte/char
- **Indify String Concat**: move the String concat to run time

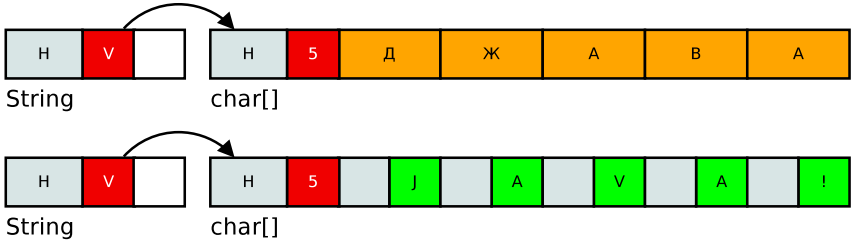
For each feature, we have three parts:

1. **Insight**: main idea and elevated hope
2. **Angst**: how many angels are dancing on JDK development pin
3. **Catarsis**: what this pain gives us in return

Or, «Why those [expletive] [expletive] [expletive] cannot do the feature in a month, but spend a year instead»[citation needed]

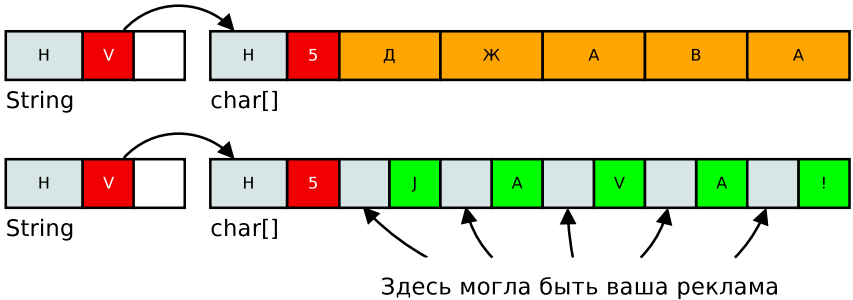
Compact Strings

Compact Strings: String internals



- Two objects: `String` и `char[]`
- Anything we can cut?

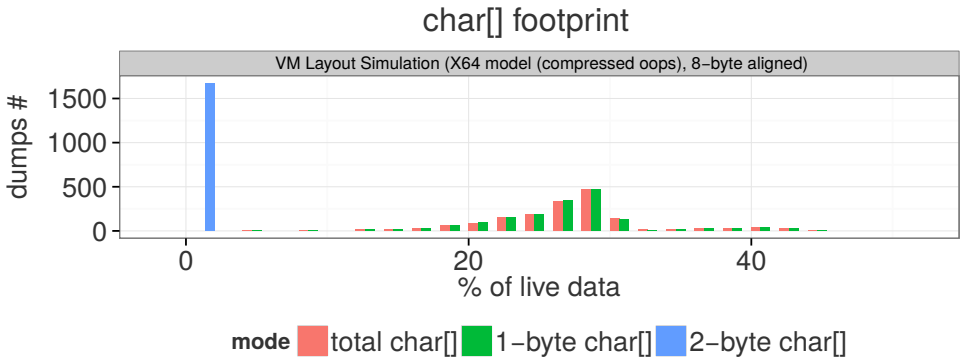
Compact Strings: String internals



- Lots of zeroes in those `char[]`-s
- Mostly, because an overwhelming number of `String`-s is Latin1

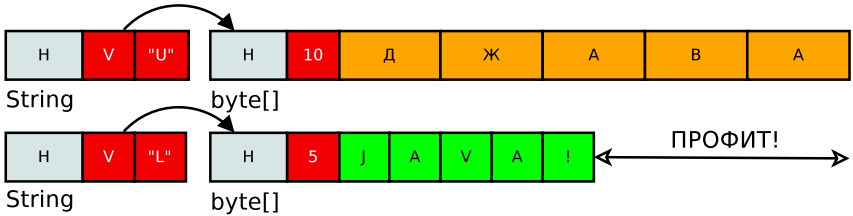
Compact Strings: Some RealWorld (tm) Data

Most applications have lots of `Strings`, most of them are Latin1:



Compact Strings: Wouldn't It Be Awesome...

...if `String`-s were like that:



- Don't store `char[]`, store `byte[]`
- The `String` itself is aware what «coder» to use for deciphering the data

Compact Strings: Thou Shalt Not Regress

Lots and lots of `charAt` calls in JDK:

```
# find -iname \*.java -exec grep charAt {} \; | wc -l
2157
```

Simplest use case:

```
for (int i = 0; i < s.length(); i++) {
   char c = s.charAt(i);
   System.out.println(c);
}
```


Compact Strings: Bird's Eye View

- String is compressed during construction
  - The largest trouble: `String(char[] value)` constructor
  - Optimistically try to compress in 1-byte. Failed? Copy as 2-byte...
  - Some shortcuts are possible, e.g. ISO-8851-1 fastpath

- Methods are able to operate on both 1-byte and 2-byte forms
  - `substring` for a 1-byte String is also 1-byte
  - `String(String other)` constructor does not repack
  - `char[] getChars()` needs to copy/unpack

Compact Strings: JDK Development In 5 Easy Steps

1. Check out the workspace
2. Hack, hack, hack

```
sed -i -e "s/char[]/byte[]/g" \
    java/lang/String.java
```

3. Run the tests
4. Fix the bugs (a few `if`s here and there)
5. BIKESHED A LOT

Who codes the coders?

CS, Coders: OOP

Reference a «coder» implementation:

- Almost idiomatic OOP!
- Quadratisch.
- Praktisch.
- Gut.

Problems?

```
class String {
  byte[] val;
  Coder coder;

  char charAt(int idx) {
    return coder.charAt(val, idx);
  }
}

interface Coder {
  char charAt(byte[] val, int idx);
}
```


CS, Coders: Object?

Masking `byte[]` or `char[]`:

- «Ever so slight» step away from OOP idiomatics
- Typechecks are cheap, right?

Problems?

```
class String {
  Object val;
  char charAt(int idx) {
    // FIXME: Range checks!
    if (val instanceof byte[]) {
      byte[] v = (byte[])val;
      return toChar(v[idx]);
    } else {
      // Totally safe!
      char[] v = (char[])val;
      return v[idx];
    }
  }
}
```


CS, Coders: Hide Me Under The Tree

Hide coder as zero-th element:

- Oh wow, it's like Pascal all over again!
- Seriously, we can just compare the arrays now

Problems?

```
class String {
  byte[] val;
  char charAt(int idx) {
    // FIXME: Range checks!
    byte coder = val[0];
    if (coder == LATIN1) {
      return toChar(val[idx + 1]);
    } else {
      return toChar(
         (val[idx*2] << 8)
               + val[idx*2+1]);
    }
  }
}
```

CS, Coders: Tagged Arrays

Steal a bit from address, maybe?

- We are... RUNTIME.
- We can do whatever the hell we want

Problems?

```
class String {
  byte[] val;
  char charAt(int idx) {
    // FIXME: Range checks!
    byte coder = VM_MAGIC(val);
    if (coder == LATIN1) {
      return val[idx];
    } else {
      return getChar(val, idx);
    }
  }
}
```


CS, Coders: Thou Shalt Not Overcomplicate

```
***** 32-bit VM: ************************************************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0     8        (object header)                N/A
      8     4 char[] String.value                   N/A
     12     4    int String.hash                    N/A
Instance size: 16 bytes

***** 64-bit VM, compressed references enabled: *****************
java.lang.String object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                    VALUE
      0    12        (object header)                N/A
     12     4 char[] String.value                   N/A
     16     4    int String.hash                    N/A
     20     4        (loss due to the next object alignment)
Instance size: 24 bytes
```


CS, Coders: Just A Field

Coder is binary:

- 0 - Latin1, «all high bytes are zero»
- 1 - UTF16, «at least one high byte is not zero»

Allows you do to tricks:

- Compare Strings without touching `val`
- Concat result coder = «OR» over argument coders

```
class String {
  byte[] val;
  byte coder;
  char charAt(int idx) {
    // FIXME: Range checks!
    if (coder == LATIN1) {
      return val[idx];
    } else {
      return getChar(val, idx);
    }
  }
}
```


CS, Coders: In Much More Detail

Magnum Opus on method dispatch performance:
`http://shipilev.net/blog/2015/black-magic-method-dispatch/`

More details on choosing where to put coders:
`http://cr.openjdk.java.net/~shade/density/double-selection.txt`

Stable zero and other perversions

CS, Zeroes: What Kind of Sourcery Is This?

Why is this optimized like that?

```
static final String S;

static {
  S = "Foo";
}

@Benchmark
int testLength() {
  return S.length();
}
```

```
testLength():
  # ... some mumbo-jumbo ...
  # THE ANSWER IS "3"!
  mov $0x3$, %eax
  # ... some mumbo-jumbo ...
  ret
```


CS, Zeroes: Truth or Dare

```
static final MyClass finallie    = new MyClass();
static       MyClass nonFinallie = new MyClass();

class MyClass {
  final int trustMe = 42;
}

int read_finallie() {
  return finallie.trustMe; // folded to "42", or not?
}

int read_nonFinallie() {
  return nonFinallie.trustMe; // folded to "42", or not?
}
```


CS, Zeroes: Truth or dare

```
static final MyClass finallie = new MyClass();

class MyClass {
  @Stable
  final int trustMe = 42;
}

int read_finallie() {
  return finallie.trustMe; // folded to "42", or not?
}
```


CS, Zeroes: Truth or Dare

```
class String {
  @Stable final byte coder;
  @Stable final byte[] value;
  int length() { return value.length >> coder);
}

int len_Latin1() {
  return "Foo".length(); // folded or not?
}

int len_UTF16() {
  return "Φyy".length(); // folded or not?
}
```


CS, Zeroes: Gonna Try Something Else

- Find the compiler's stash of trusted classes
- Find the special treatment for `final` fields in those classes
- Add `String` to trust its `final` fields

Feature Kill Switch:
`-XX:-CompactStrings`

CS, Kill Switch: Naively...

```
class String {
  static final boolean COMPACT_STRINGS = ...;

  String(char[] value) {
     if (COMPACT_STRINGS) {
        byte[] v = tryCompress(value);
        if (v != null) {
           this.value = v; this.coder = LATIN1; return;
        }
     }
     this.value = copy(value); this.coder = UTF16;
  }
}
```


CS, Kill Switch: Working...

```
class String {
  static final boolean COMPACT_STRINGS = ...;

  char charAt(int idx) {
    if (coder == LATIN1) { // branch, nevertheless
      return val[idx];
    else
      return getChar(val, idx);
  }
}
```

Everything is awesome, everyone is coming with `(coder == UTF16)`?

CS, Kill Switch: Working...

```
class String {
  static final boolean COMPACT_STRINGS = ...;

  char charAt(int idx) {
    if ((coder == LATIN1) && COMPACT_STRINGS) {
      return val[idx];
    else
      return getChar(val, idx);
  }
}
```

Compilers are good at folding, right?

CS, Kill Switch: Working...

```
class String {
  static final boolean COMPACT_STRINGS = ...;

  char charAt(int idx) {
    if (COMPACT_STRINGS && (coder == LATIN1)) {
      return val[idx];
    else
      return getChar(val, idx);
  }
}
```

Nope... only this one works reliably.[1]

[1] https://bugs.openjdk.java.net/browse/JDK-8087309

CS, Kill Switch: Working...

```
class String {
  static final boolean COMPACT_STRINGS = ...;
  boolean isLatin1() { return COMPACT_STRINGS && (coder == LATIN1); }

  char charAt(int idx) {
    if (isLatin1()) {
      return val[idx];
    else
      return getChar(val, idx);
  }
}
```

Or this?

CS, Kill Switch: Will This Work?

```
class String {
  static final boolean COMPACT_STRINGS =
    Boolean.getBoolean("compactStrings");
}
```


CS, Kill Switch: Will This Work?

```
class String {
  static final boolean COMPACT_STRINGS =
    Boolean.getBoolean("compactStrings");
}
```

Circular dependencies:

```
Error occurred during initialization of VM
java.nio.charset.IllegalCharsetNameException: UTF-8
at java.nio.charset.Charset.checkName(Charset.java:316)
    ...
at java.lang.StringCoding.decode(StringCoding.java:334)
at java.lang.String.<init>(String.java:592)
at java.lang.String.<init>(String.java:614)
at java.lang.System.initProperties(Native Method)
at java.lang.System.initializeSystemClass(System.java:1162)
```


CS, Kill Switch: What About This One?

```
class String {
  static final boolean COMPACT_STRINGS =
    Unsafe.isCompactStrings();
}
```


CS, Kill Switch: What About This One?

```
class String {
  static final boolean COMPACT_STRINGS =
    Unsafe.isCompactStrings();
}
```

Even better...

```
# A fatal error has been detected by the JRE:
#
#  SIGSEGV (0xb) at pc=0x0000000000000000, pid=..., tid=...
#
# JRE version:  (9.0) (build ...)
# Java VM: OpenJDK 64-Bit Server VM ...
# Problematic frame:
# C  0x0000000000000000
```


CS, Kill Switch: It's ALIVE!

```
class String {
  static final boolean COMPACT_STRINGS = isCompactStrings();
  private static native boolean isCompactStrings();
}

JNIEXPORT jboolean JNICALL
Java_java_lang_String_isCompactStrings(JNIEnv *env, jclass cls) {
    return JVM_IsCompactStrings();
}

JNIEXPORT jboolean JNICALL JVM_IsCompactStrings(void) {
  JVMWrapper("JVM_IsCompactStrings");
  return CompactStrings;
}
```


CS, Kill Switch: Will This Do?

```
class String {
  static final boolean COMPACT_STRINGS = true;
}

void VM::super_secret_VM_method() {
  find_String_field_and_set_it_to(CompactStrings);
}
```


CS, Kill Switch: Or This?

```
class String {
  static final boolean COMPACT_STRINGS =
    new Boolean(true).booleanValue();
}

void VM::super_secret_VM_method() {
  find_String_field_and_set_it_to(CompactStrings);
}
```


Arrays of Trickery Ahead

CS, Arrays: `char[]`

```
char[] tryWithChar(int size) {
  return new char[size];
}
```

What `size` would fail?

CS, Arrays: char[]

```
char[] tryWithChar(int size) {
  return new char[size];
}
```

What size would fail?

```
tryWithChar(Integer.MAX_VALUE):
```

```
Exception in thread "main" java.lang.OutOfMemoryError:
Requested array size exceeds VM limit
at org.openjdk.BoundsTest.testWith(BoundsTest.java:30)
at org.openjdk.BoundsTest.main(BoundsTest.java:25)
```


CS, Arrays: char[]

```
char[] tryWithChar(int size) {
  return new char[size];
}
```

What size would fail?

```
tryWithChar(Integer.MAX_VALUE - 2); // works
```


CS, Arrays: String

```
String tryWithString(int size) {
  return new String(new char[size]);
}
```

```
tryWithString(Integer.MAX_VALUE - 2):
```


CS, Arrays: String

```
String tryWithString(int size) {
  return new String(new char[size]);
}
```

```
tryWithString(Integer.MAX_VALUE - 2):
```

```
java.lang.OutOfMemoryError:
UTF16 String size is 2147483645, should be less than 1073741807
at java.lang.StringUTF16.rangeCheck(StringUTF16.java:56)
at java.lang.StringUTF16.compress(StringUTF16.
```


getChar,
or `char[]` view over `byte[]`, anyone?

CS, getChar: What Could Possibly Be Simpler

```
char getChar_UTF16(byte[] val, int idx) {
  idx *= 2;
  return toChar(val[idx]) + toChar(val[idx + 1]) << 8;
}
```


CS, getChar: What Could Possibly Be Simpler

```
char getChar_UTF16(byte[] val, int idx) {
  idx *= 2;
  return toChar(val[idx]) + toChar(val[idx + 1]) << 8;
}
```

Endianness, you freakish hardware you?

CS, getChar: What Could Possibly Be Simpler

```
char getChar_UTF16(byte[] val, int idx) {
  idx *= 2;
  return toChar(val[idx]) + toChar(val[idx + 1]) << 8;
}
```

Endianness, you freakish hardware you?

```
char getChar_UTF16(byte[] val, int idx) {
  idx *= 2;
  return toChar(val[idx + 1]) + toChar(val[idx]) << 8;
}
```


CS, getChar: Running With Scissors

```
// Got a hammer:
static final Unsafe U = Unsafe.getUnsafe();
static final int base = U.ARRAY_CHAR_BASE_OFFSET;
static final int scale = U.ARRAY_CHAR_INDEX_SCALE;

// Everything looks like a nail:
char getChar_UTF16(byte[] val, int idx) {
  return U.getChar(val, base + scale*idx);
}
```


CS, getChar: Running With Scissors

```
// Got a hammer:
static final Unsafe U = Unsafe.getUnsafe();
static final int base = U.ARRAY_CHAR_BASE_OFFSET;
static final int scale = U.ARRAY_CHAR_INDEX_SCALE;

// Everything looks like a nail:
char getChar_UTF16(byte[] val, int idx) {
  return U.getChar(val, base + scale*idx);
}
```

Hint:

$$\lim_{idx \to MAX} getChar(val, idx) =< garbage >$$

CS, getChar: Running With Longer Scissors

```
// Got a longer hammer:
static final Unsafe U = Unsafe.getUnsafe();
static final long base = U.ARRAY_CHAR_BASE_OFFSET;
static final long scale = U.ARRAY_CHAR_INDEX_SCALE;

// Everything looks like a longer nail:
char getChar_UTF16(byte[] val, int idx) {
  return U.getChar(val, base + scale*idx);
}
```

[2] https://bugs.openjdk.java.net/browse/JDK-8074124

CS, getChar: Running With Longer Scissors

```
// Got a longer hammer:
static final Unsafe U = Unsafe.getUnsafe();
static final long base = U.ARRAY_CHAR_BASE_OFFSET;
static final long scale = U.ARRAY_CHAR_INDEX_SCALE;

// Everything looks like a longer nail:
char getChar_UTF16(byte[] val, int idx) {
  return U.getChar(val, base + scale*idx);
}
```

Except that 32-bit platforms are down:[2]

```
UnsafeConvBench.plain                3835.953 ± 48.240 ns/op
UnsafeConvBench.unsafe_field_scale   7611.268 ± 72.331 ns/op
```

[2] https://bugs.openjdk.java.net/browse/JDK-8074124

CS, getChar: Recap

`(get|put)Char` problems are solvable with three options:

1. Make compiler prove no overflow in regular expressions

2. Drill more holes in `Unsafe`

```
<T> T Unsafe.get_T_Indexed(Object,int)
```

3. Make more intrinsics in C1 and C2:

```
char StringUTF16.getChar(byte[] val, int idx)
void StringUTF16.putChar(byte[] val, int idx, char c)
```


What Is Left?: ...and few things

- Redo the same for all important `String` methods
- Redo the same for other platforms
- Fix up VM support for native `Stringss`
- Moar Intinriscs for Intrinsics GOD
- Encoders/Decoders handling for new `String` forms
- Fix up library: `AbstractStringBuilder`-s!
- Fix up `-XX:+OptimizeStringConcat`

What Is Left?: ...and few things

- Redo the same for all important `String` methods
- Redo the same for other platforms
- Fix up VM support for native `Stringss`
- Moar Intinriscs for Intrinsics GOD
- Encoders/Decoders handling for new `String` forms
- Fix up library: `AbstractStringBuilder`-s!
- Fix up `-XX:+OptimizeStringConcat`

CS, Catharsis: LogLineBench

```
public class LogLineBench {
  @Param
  int size;

  String method = generateString(size);

  @Benchmark
  public String work() throws Exception {
      return "[" + System.nanoTime() + "] " +
             Thread.currentThread().getName() + ":" +
             "Calling an application method \"" + method +
             "\" without fear and prejudice.";
  }
}
```


CS, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
8u66 Baseline	148 ± 2	147 ± 2	227 ± 3	888	904	1680
9b92 Baseline	149 ± 3	153 ± 4	231 ± 4	888	904	1680
9b93 -XX:-CompactStrings	152 ± 3	150 ± 4	230 ± 6	888	904	1680
9b103 -XX:-CompactStrings	132 ± 4	135 ± 6	224 ± 4	888	904	1680
9b93 -XX:+CompactStrings	142 ± 2	139 ± 3	169 ± 4	504	512	904
9b103 -XX:+CompactStrings	130 ± 5	130 ± 4	155 ± 6	504	512	904

- 1.36x better throughput, 1.85x less garbage
- Kill Switch is working, for the cases when something goes wrong

CS, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
8u66 Baseline	148 ± 2	147 ± 2	227 ± 3	888	904	1680
9b92 Baseline	149 ± 3	153 ± 4	231 ± 4	888	904	1680
9b93 -XX:-CompactStrings	152 ± 3	150 ± 4	230 ± 6	888	904	1680
9b103 -XX:-CompactStrings	132 ± 4	135 ± 6	224 ± 4	888	904	1680
9b93 -XX:+CompactStrings	142 ± 2	139 ± 3	169 ± 4	504	512	904
9b103 -XX:+CompactStrings	130 ± 5	130 ± 4	155 ± 6	504	512	904

- 1.36x better throughput, 1.85x less garbage
- Kill Switch is working, for the cases when something goes wrong

CS, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
8u66 Baseline	148 ± 2	147 ± 2	227 ± 3	888	904	1680
9b92 Baseline	149 ± 3	153 ± 4	231 ± 4	888	904	1680
9b93 -XX:-CompactStrings	152 ± 3	150 ± 4	230 ± 6	888	904	1680
9b103 -XX:-CompactStrings	132 ± 4	135 ± 6	224 ± 4	888	904	1680
9b93 -XX:+CompactStrings	142 ± 2	139 ± 3	169 ± 4	504	512	904
9b103 -XX:+CompactStrings	130 ± 5	130 ± 4	155 ± 6	504	512	904

- 1.36x better throughput, 1.85x less garbage
- Kill Switch is working, for the cases when something goes wrong

Indify String Concat

Indify String Concat: See Any Concat In Bytecode?

```
String m(String a, int b) {  return a + "(" + b + ")"; }
```


Indify String Concat: See Any Concat In Bytecode?

```
String m(String a, int b) {  return a + "(" + b + ")"; }
```

```
 0: new           #2   // class java/lang/StringBuilder
 3: dup
 4: invokespecial #3   // Method java/lang/StringBuilder."":()V
 7: aload_1
 8: invokevirtual #4   // Method java/lang/StringBuilder.append:(Ljava/lang
11: ldc           #5   // String (
13: invokevirtual #4   // Method java/lang/StringBuilder.append:(Ljava/lang
16: iload_2
17: invokevirtual #6   // Method java/lang/StringBuilder.append:(I)Ljava/la
20: ldc           #7   // String )
22: invokevirtual #4   // Method java/lang/StringBuilder.append:(Ljava/lang
25: invokevirtual #8   // Method java/lang/StringBuilder.toString:()Ljava/l
28: areturn
```

Indify String Concat: OptimizeStringConcat

Compiler is not stupid, it has `-XX:+-OptimizeStringConcat`[3]

1. Match the IR for something that looks like a sane `StringBuilder::append` and `StringBuider::toString` chains
2. Squeak happily, and replace the entire thing with:
   - Compute the argument(s) lengths, and result length
   - Allocate the storage
   - Do the in-place copies, where possible, e.g. `Integer.getChars(...)`
   - Do the efficient arraycopies for `Strings`
3. Profit!

[3] `http://hg.openjdk.java.net/jdk9/jdk9/hotspot/file/tip/src/share/vm/opto/stringopts.cpp`

Indify String Concat: Caveat #1: «Transparency»

```
int next() { return (id++) & 0xFF };

@Benchmark
public String infix() {
  return "Hello, sir, your SS number is " + next() +
    ", and you have a problem with your tax report.";
}

@Benchmark
public String prefix() {
  int luckyBoy = next();
  return "Hello, sir, your SS number is " + luckyBoy +
    ", and you have a problem with your tax report.";
}
```


Indify String Concat: Caveat #1: «Transparency»[4]

```
int x;

// Optimizeable:
new StringBuilder().append("Foo: ").append(x).toString();

// NOPE. NOPE. NOPE.
new StringBuilder().append("Foo: ").append(x++).toString();

// Optimizeable again:
x++;
new StringBuilder().append("Foo: ").append(x).toString();
```

[4] https://bugs.openjdk.java.net/browse/JDK-8043677

Indify String Concat: Caveat #2: «Universality»

```
int i;
long l;
double d;

// Optimizeable:
new StringBuilder().append("Foo: ").append(i).toString();

// NOPE, LOL
new StringBuilder().append("Foo: ").append(l).toString();

// YOUR WISH, BROTHA, ROFL
new StringBuilder().append("Foo: ").append(d).toString();
```


Indify String Concat: Caveat #3: «Reliability»[5]

```
String s1, s2;

// JDK 9 Baseline, optimizeable:
new StringBuilder(s1.length() + s2.length())
    .append(s1).append(s2).toString();

// JDK 9 Compact Strings... NOPE, WHAT ARE U DOIN', STAHP
class String {
  int length() { return value.length >> coder; }
}

new StringBuilder(s1.length() + s2.length())
    .append(s1).append(s2).toString();
```

[5] https://bugs.openjdk.java.net/browse/JDK-8136469

Indify String Concat: Insight

Want to match runtime support?
You **have to** control the bytecode.

How to hande this mess? Let's change the bytecode emitted by `javac`, d'uh:

- Ask users to recompile with newer `javac`?
- It is easy to make a simple «mistake» in bytecode, so that compiler would be unable to match
- M bytecode variants, N compiler versions
  $\Rightarrow$ M*N configurations to test

Indify String Concat: Late Binding

Level out the impedance between the Java *language* and Java *bytecode*.
If only we could delay the decision on what concat really does:

```
package java.lang;

class StringConcat {
  String concat(String first, String... moar) {
    // TODO: Actually implement this.
  }
}
```


Indify String Concat: Late Binding

Level out the impedance between the Java *language* and Java *bytecode*.
If only we could delay the decision on what concat really does:

```
package java.lang;

class StringConcat {
  String concat(String first, String... moar) {
    // TODO: Actually implement this.
  }
}
```

Dang it, this only works with Strings...

Indify String Concat: Late Binding

Level out the impedance between the Java *language* and Java *bytecode*.
If only we could delay the decision on what concat really does:

```
package java.lang;

class StringConcat {
  String concat(Object first, Object... moar) {
    // TODO: Actually implement this.
  }
}
```


Indify String Concat: Late Binding

Level out the impedance between the Java *language* and Java *bytecode*.
If only we could delay the decision on what concat really does:

```
package java.lang;

class StringConcat {
  String concat(Object first, Object... moar) {
    // TODO: Actually implement this.
  }
}
```

Dang it, boxing. Dang it, allocating for varargs...

Indify String Concat: `invokedynamic` magic

```
String m(String a, int b) {  return a + "(" + b + ")"; }
```


Indify String Concat: `invokedynamic` magic

```
String m(String a, int b) {  return a + "(" + b + ")"; }

java.lang.String m(java.lang.String, int);
  0: aload_1
  1: ldc           #2       // String (
  3: iload_2
  4: ldc           #3       // String )
  6: invokedynamic #4,  0   // InvokeDynamic #0:makeConcat
                            //   (String,String,int,String)String
 11: areturn

BootstrapMethods:
  0: #19 invokestatic java/lang/invoke/StringConcatFactory.makeConcat...
```


Indify String Concat: `invokedynamic` magic, #2

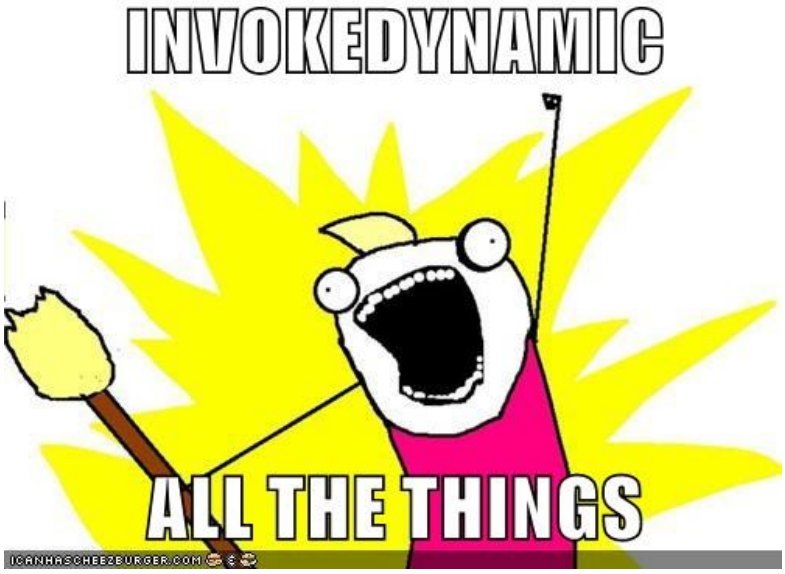
```
String m(String a, int b) {  return a + "(" + b + ")"; }
```


Indify String Concat: `invokedynamic` magic, #2

```
String m(String a, int b) {  return a + "(" + b + ")"; }
```

```
java.lang.String m(java.lang.String, int);
  0: aload_1
  1: iload_2
  2: invokedynamic #2,  0   // InvokeDynamic #0:makeConcat
                            //   (String,int)String;
  7: areturn

BootstrapMethods:
  0: #15 invokestatic
  java/lang/invoke/StringConcatFactory.makeConcatWithConstants:..
    Method arguments:
      #16 \u0001(\u0001)
```

ISC, Init: Oops

```
Error occurred during initialization of VM
java.lang.StackOverflowError
at java.lang.Throwable.toString(Throwable.java:481)
at java.lang.BootstrapMethodError.<init>(BootstrapMethodError.java:77
at java.lang.Throwable.toString(Throwable.java:481)
...
at sun.nio.cs.StandardCharsets.lookup(StandardCharsets.java:1100)
at sun.nio.cs.StandardCharsets.charsetForName(StandardCharsets.java:1
```


ISC, Init: Oops

```
Error occurred during initialization of VM
java.lang.StackOverflowError
at java.lang.Throwable.toString(Throwable.java:481)
at java.lang.BootstrapMethodError.<init>(BootstrapMethodError.java:77
at java.lang.Throwable.toString(Throwable.java:481)
...
at sun.nio.cs.StandardCharsets.lookup(StandardCharsets.java:1100)
at sun.nio.cs.StandardCharsets.charsetForName(StandardCharsets.java:1
```

Concat needs `java.lang.invoke`, that uses concat, that needs `java.lang.invoke`, that uses concat, that needs `java.lang.invoke`, that uses concat, that needs `java.lang.invoke`, ...

ISC, Init: Thank You, Dear Jigsaw

Before JEP 201 «Modular Source Code»[6]:

1. `java.lang.invoke` can potentially call anything in JDK
2. Ergo, no JDK classes are allowed to use Indy String Concat

[6] `http://openjdk.java.net/jeps/201`

ISC, Init: Thank You, Dear Jigsaw

Before JEP 201 «Modular Source Code»[6]:

1. `java.lang.invoke` can potentially call anything in JDK
2. Ergo, no JDK classes are allowed to use Indy String Concat

After JEP 201:

1. `java.lang.invoke` can potentially call anything in `java.base`
2. Ergo, no `java.base` class can use Indy String Concat
3. Thank You, Dear Jigsaw!

[6] `http://openjdk.java.net/jeps/201`

But the initial `invokedynamic` linkage should cost us something? How's startup?

ISC, Startup: Hello Cruel String Concat World

```
public class Concat {
  static String a = "a";
  static String b = "b";
  static String c = null;
  static int d = 42;

  public static void main(String... args) {
    System.out.println(a + b + c + d + null);
  }
}
```


ISC, Startup: Take This `HelloWorld` Example

JDK 9 Baseline:

```
-XstringConcat:inline:  200.1 ± 1.5 ms
-XstringConcat:indy:    201.2 ± 2.0 ms
```

Very nice! El-cheapo `java.lang.invoke`!

[7] `https://bugs.openjdk.java.net/browse/JDK-8136854`

ISC, Startup: Take This `HelloWorld` Example

JDK 9 Baseline:

```
-XstringConcat:inline:  200.1 ± 1.5 ms
-XstringConcat:indy:    201.2 ± 2.0 ms
```

Very nice! El-cheapo `java.lang.invoke`!
Oh shi... G1 performance bug.[7]

[7] `https://bugs.openjdk.java.net/browse/JDK-8136854`

ISC, Startup: Take This `HelloWorld` Example

JDK 9 Baseline:

```
-XstringConcat:inline:  200.1 ± 1.5 ms
-XstringConcat:indy:    201.2 ± 2.0 ms
```

Very nice! El-cheapo `java.lang.invoke`!
Oh shi... G1 performance bug.[7]

`-XX:+UseParallelGC`:

```
-XstringConcat:inline:  31.1 ± 1.5 ms
-XstringConcat:indy:    58.5 ± 2.0 ms
```

[7] https://bugs.openjdk.java.net/browse/JDK-8136854

ISC, Startup: Hello Cruel String Concat World, #2

```
public class Concat2 {
  static String a = "a";
  static String b = "b";
  static String c = null;
  static int d = 42;

  public static void main(String... args) {
    System.out.println(a + b + c + d + null);         // 1
    System.out.println(a + b + c + d + null + null); // 2
  }
}
```

```
(1):    58.5 ± 2.0 ms
(1+2):  59.5 ± 2.0 ms
```


ISC, Startup: Necessary Evil?

First `invokedynamic` user pays for everyone:[8]

- Indify String Concat
- Lambda MetaFactory
- Nashorn, JRuby
- Jigsaw

Early Bird Gets All The Regression Bugs

[8] `https://bugs.openjdk.java.net/browse/JDK-8086045`

ISC, Strategies: Basic

***_SB**:
«Delegate to `StringBuilder` to handle everything»

Recipe:

1. Shove in `StringBuilder.append`-s
2. Let C2 to handle the rest in OptimizeStringConcat

Bottom line: Almost 1:1 to what javac currently does. This is a fallback/testing strategy: as non-intrusive as it can possibly be.

ISC, Strategies: More advanced

***_SB_SIZED**:
«*Guess the final size*, delegate to `StringBuilder`»

Recipe:

1. Call `toString()` on all reference types eagerly
2. Poll `length()` over all references, assume the max length for primitives
3. Pre-size `StringBuilder`
4. Shove in `StringBuilder.append`-s
5. Let C2 to handle the rest in OptimizeStringConcat

Bottom line: Pre-sizes SB exactly for non-primitive args.

ISC, Strategies: Even more advanced

***_SB_SIZED_EXACT**:
«*Figure out the final size*, delegate to `StringBuilder`»

Recipe:

1. Call `toString()` on *all args* eagerly
2. Poll `length()` over *all args*
3. Pre-size `StringBuilder`
4. Shove in `StringBuilder.append`-s
5. Let C2 to handle the rest in OptimizeStringConcat

Bottom line: Pre-sizes SB exactly for all args, but wastes a trip on primitives.

ISC, Strategies: Something completely different

MH_INLINE_SIZED_EXACT:
«I'm gonna build my own string concat, with private APIs and who cares?»

Recipe:

1. Call `toString()` on all reference args
2. Call `length()` *or* `T.stringSize(T t)` on everything
3. Call-and-or `coder()` on all reference args
4. Allocate `byte[]`, copy all args, convert primitives in-place
5. Invoke a private `String` constructor, handing over the array

Bottom line: OptoStringConcat benefits without OptoStringConcat mess.

ISC, Catharsis: LogLineBench

```
public class LogLineBench {
  @Param
  int size;

  String method = generateString(size);

  @Benchmark
  public String work() throws Exception {
      return "[" + System.nanoTime() + "] " +
             Thread.currentThread().getName() + ":" +
             "Calling an application method \"" + method +
             "\" without fear and prejudice.";
  }
}
```


ISC, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	134 ± 7	135 ± 9	220 ± 9	888	904	1680
BC_SB_SIZED	100 ± 2	99 ± 4	132 ± 7	560	592	960
BC_SB_SIZED_EXACT	73 ± 3	76 ± 2	83 ± 5	336	352	536
MH_SB_SIZED	101 ± 4	99 ± 3	135 ± 5	560	592	960
MH_SB_SIZED_EXACT	106 ± 3	109 ± 3	138 ± 4	600	632	1000
MH_INLINE_SIZED_EXACT	77 ± 3	78 ± 3	86 ± 4	288	304	488

- BC_SB dies bit does not regress: `invokedynamic` rules
- Most optimal ISC strategies do 2.6x better, and 3.4x less garbage

ISC, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	134 ± 7	135 ± 9	220 ± 9	888	904	1680
BC_SB_SIZED	100 ± 2	99 ± 4	132 ± 7	560	592	960
BC_SB_SIZED_EXACT	73 ± 3	76 ± 2	83 ± 5	336	352	536
MH_SB_SIZED	101 ± 4	99 ± 3	135 ± 5	560	592	960
MH_SB_SIZED_EXACT	106 ± 3	109 ± 3	138 ± 4	600	632	1000
MH_INLINE_SIZED_EXACT	77 ± 3	78 ± 3	86 ± 4	288	304	488

- BC_SB dies bit does not regress: `invokedynamic` rules
- Most optimal ISC strategies do 2.6x better, and 3.4x less garbage

ISC, Catharsis: Performance

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	134 ± 7	135 ± 9	220 ± 9	888	904	1680
BC_SB_SIZED	100 ± 2	99 ± 4	132 ± 7	560	592	960
BC_SB_SIZED_EXACT	73 ± 3	76 ± 2	83 ± 5	336	352	536
MH_SB_SIZED	101 ± 4	99 ± 3	135 ± 5	560	592	960
MH_SB_SIZED_EXACT	106 ± 3	109 ± 3	138 ± 4	600	632	1000
MH_INLINE_SIZED_EXACT	77 ± 3	78 ± 3	86 ± 4	288	304	488

- BC_SB dies bit does not regress: `invokedynamic` rules
- Most optimal ISC strategies do 2.6x better, and 3.4x less garbage

Rendezvous

Rendezvous: Compact Strings + Indy String Concat

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	125 ± 2	126 ± 3	155 ± 4	504	512	904
BC_SB_SIZED	92 ± 5	94 ± 2	98 ± 3	312	328	512
BC_SB_SIZED_EXACT	81 ± 2	81 ± 2	83 ± 3	200	208	296
MH_SB_SIZED	88 ± 2	90 ± 1	95 ± 2	312	328	512
MH_SB_SIZED_EXACT	99 ± 1	101 ± 1	105 ± 2	344	360	536
MH_INLINE_SIZED_EXACT	75 ± 1	77 ± 1	78 ± 1	168	176	264

- Even the basic BC_SB is better because of Compact Strings
- Most optimal ISC strategies do 2.9x better, and 6.4x less garbage

Rendezvous: Compact Strings + Indy String Concat

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	125 ± 2	126 ± 3	155 ± 4	504	512	904
BC_SB_SIZED	92 ± 5	94 ± 2	98 ± 3	312	328	512
BC_SB_SIZED_EXACT	81 ± 2	81 ± 2	83 ± 3	200	208	296
MH_SB_SIZED	88 ± 2	90 ± 1	95 ± 2	312	328	512
MH_SB_SIZED_EXACT	99 ± 1	101 ± 1	105 ± 2	344	360	536
MH_INLINE_SIZED_EXACT	75 ± 1	77 ± 1	78 ± 1	168	176	264

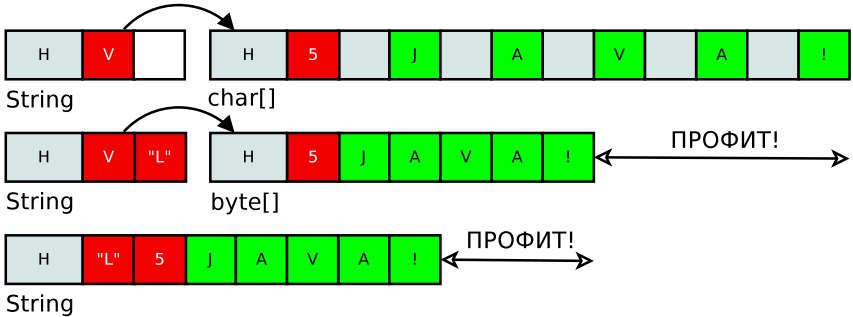
- Even the basic BC_SB is better because of Compact Strings
- Most optimal ISC strategies do 2.9x better, and 6.4x less garbage

Rendezvous: Compact Strings + Indy String Concat

	throughput, ns/op			allocated, b/op		
	1	10	100	1	10	100
9b103 -XX:-CS	132 ± 5	135 ± 4	224 ± 6	888	904	1680
BC_SB	125 ± 2	126 ± 3	155 ± 4	504	512	904
BC_SB_SIZED	92 ± 5	94 ± 2	98 ± 3	312	328	512
BC_SB_SIZED_EXACT	81 ± 2	81 ± 2	83 ± 3	200	208	296
MH_SB_SIZED	88 ± 2	90 ± 1	95 ± 2	312	328	512
MH_SB_SIZED_EXACT	99 ± 1	101 ± 1	105 ± 2	344	360	536
MH_INLINE_SIZED_EXACT	75 ± 1	77 ± 1	78 ± 1	168	176	264

- Even the basic BC_SB is better because of Compact Strings
- Most optimal ISC strategies do 2.9x better, and 6.4x less garbage

Rendezvous: String Fusion



- All is left is to throw away `byte[]`, and merge it into `String`
- String Fusion should be easier after Project Panama concludes

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
`Baseline (8u66, 9b92)`	224 ±4	1338 ±4	1680

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
Baseline (8u66, 9b92)	224 ±4	1338 ±4	1680
Compact Strings	155 ±4	763 ±4	904

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
`Baseline (8u66, 9b92)`	224 ±4	1338 ±4	1680
`Compact Strings`	155 ±4	763 ±4	904
`Indy String Concat (best)`	86 ±4	384 ±4	488

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
Baseline (8u66, 9b92)	224 ±4	1338 ±4	1680
Compact Strings	155 ±4	763 ±4	904
Indy String Concat (best)	86 ±4	384 ±4	488
CS + ISC (best)	78 ±4	210 ±4	264

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
`Baseline (8u66, 9b92)`	224 ±4	1338 ±4	1680
`Compact Strings`	155 ±4	763 ±4	904
`Indy String Concat (best)`	86 ±4	384 ±4	488
`CS + ISC (best)`	78 ±4	210 ±4	264
«Garbage Free» boundary			264

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
Baseline (8u66, 9b92)	224 ±4	1338 ±4	1680
Compact Strings	155 ±4	763 ±4	904
Indy String Concat (best)	86 ±4	384 ±4	488
CS + ISC (best)	78 ±4	210 ±4	264
«Garbage Free» boundary			264
CS + ISC (best) + String Fusion	???	???	???

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Rendezvous: Future

Configuration	Throughput, ns/op		Allocated, b/op
	Free[9]	Saturated[10]	
`Baseline (8u66, 9b92)`	224 ±4	1338 ±4	1680
`Compact Strings`	155 ±4	763 ±4	904
`Indy String Concat (best)`	86 ±4	384 ±4	488
`CS + ISC (best)`	78 ±4	210 ±4	264
«Garbage Free» boundary			264
`CS + ISC (best) + String Fusion`	???	???	???
«So Much Data» boundary	???	???	235

[9] 1 mutator, 8 GC threads

[10] 8 mutators, 8 GC threads

Conclusion

Conclusion: What About Real Applications?

We know it helps.on our applications **a lot**.

How much does it (help|regress) on yours? Tell us:
`https://jdk9.java.net/download/`

Conclusion: JEPs

Compact Strings:
http://openjdk.java.net/jeps/254

Indify String Concat:
http://openjdk.java.net/jeps/280

Q/A

