Ах, эти строки...

Пару слов о себе

- работаю в «Леви9»
- пишу на Java
- качаю производительность

https://github.com/stsypanov

https://habr.com/ru/users/tsypanov/posts

Пару слов о докладе

О чём доклад?	Доклад об (не)эффективном использовании строк
Для кого доклад?	Доклад для разработчиков, занимающихся производительностью, и сочувствующих
Откуда всё это?	Что-то выловлено в коде проекта, что-то выловлено во фреймворках и библиотеках

Что, чем и на чём измеряли

Код	https://github.com/stsypanov/strings
JMH	1.23
Железо	Intel Core i7-7700
JDK	по умолчанию 11 (также 8 и 13)
Режим	среднее время выполнения + расход памяти (меньше = лучше)

Пакет java.lang

- чрезвычайно консервативный
- редко меняется

... за одним исключением

Чё там у соседушки по java.lang?

- JEP 192: String Deduplication in G1
- JEP 250: Store Interned Strings in CDS Archives
- JEP 254: Compact Strings
- JEP 280: Indify String Concatenation
- ~~JEP 326: Raw String Literals (Preview)~~
- JEP 355: Text Blocks (Preview)

Почему строки?

- они везде
- они съедают весомую часть кучи
- они [при неправильном использовании] могут ухудшить производительность
- когда им хорошо, то лучше становится всем

Два подхода к прокачиванию производительности

- добавление кода (например, кэширование)
  -- String.intern()
  -- слияние одинаковых строк (JEP 192: String Deduplication in G1)

Два подхода к прокачиванию производительности

- добавление кода (например, кэширование)
  -- String.intern()
  -- слияние одинаковых строк (JEP 192: String Deduplication in G1)

- удаление кода

Основная проблема строк

Любое* преобразование строк порождает новую строку.

https://habr.com/ru/post/457776

---

* почти

Разработчик, помни!

Стратегия: не использовать строки там, где они не нужны.

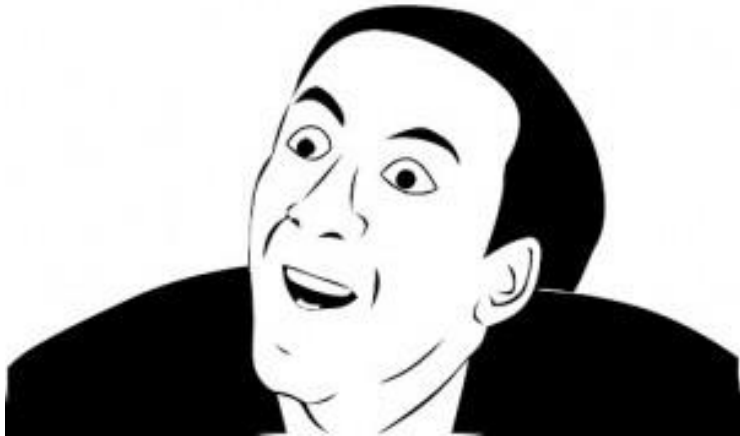
Тактика: преобразовывать строки только там, где требуется.

https://habr.com/ru/post/489554/

Ключи и словари

Строки используются в качестве ключей в словарях

```
Map<String, ?> map = new HashMap<>();
```



А так ли они нужны?

```
Map<String, ?> map = new HashMap<>();
```

- строки неизменяемы
- строки определяют equals() / hashCode()
- строки кэшируют хэш-код
- строки сериализуемы и реализуют java.lang.Comparable
- объект можно представить в виде строки вызовом obj.toString()

Баян

```
Map<String, ?> map = new HashMap<>();
```

```
class Constants {
  static final String MarginLeft    = "margl";
  static final String MarginRight   = "margr";
  static final String MarginTop     = "margt";
  static final String MarginBottom  = "margb";
}
```

EnumMap спешит на помощь!

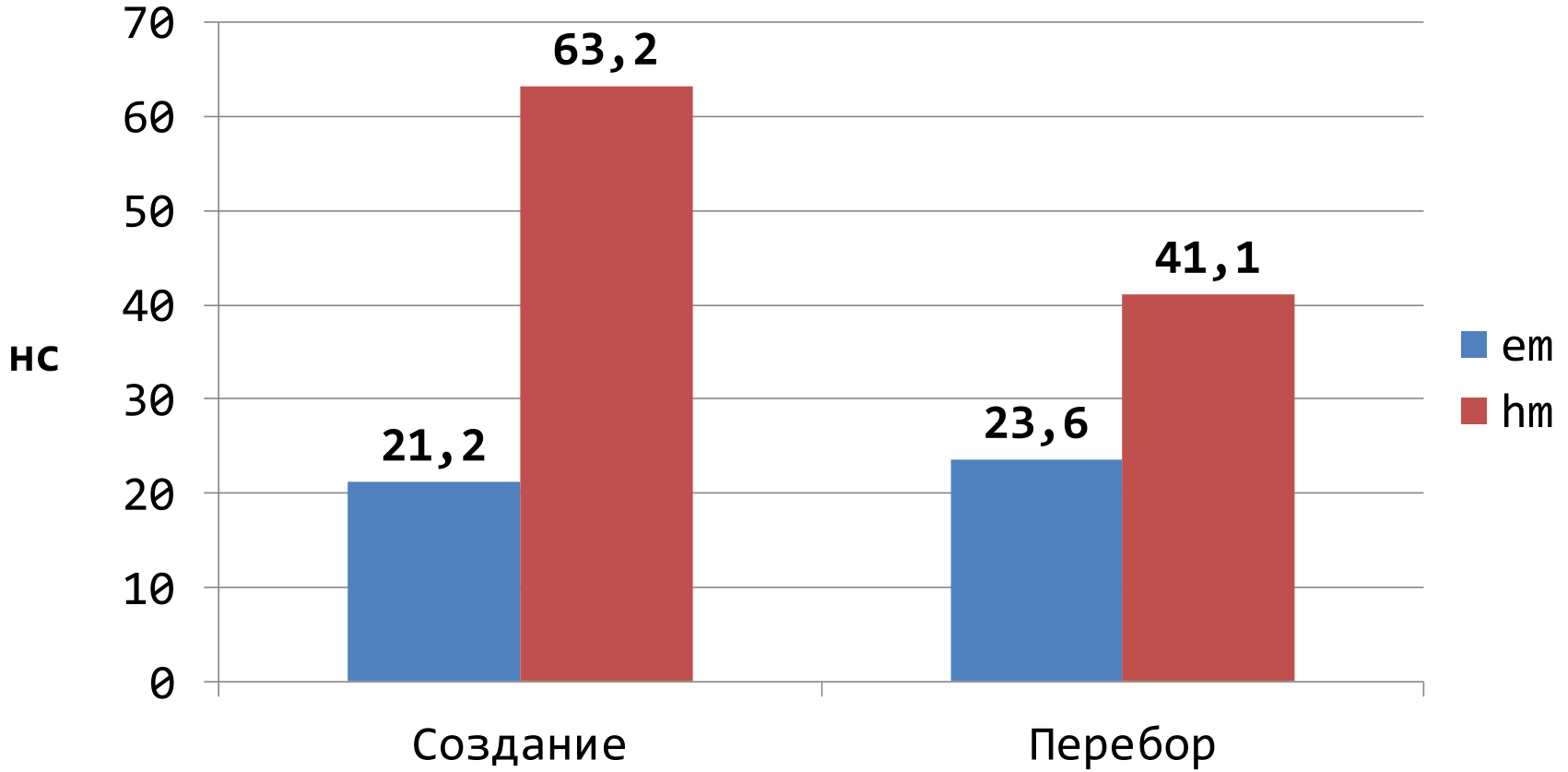
```
Map<Constants, ?> map = new EnumMap<>(Constants.class);


enum Constants {
  MarginLeft,
  MarginRight,
  MarginTop,
  MarginBottom
}
```

Стоит ли игра свеч?

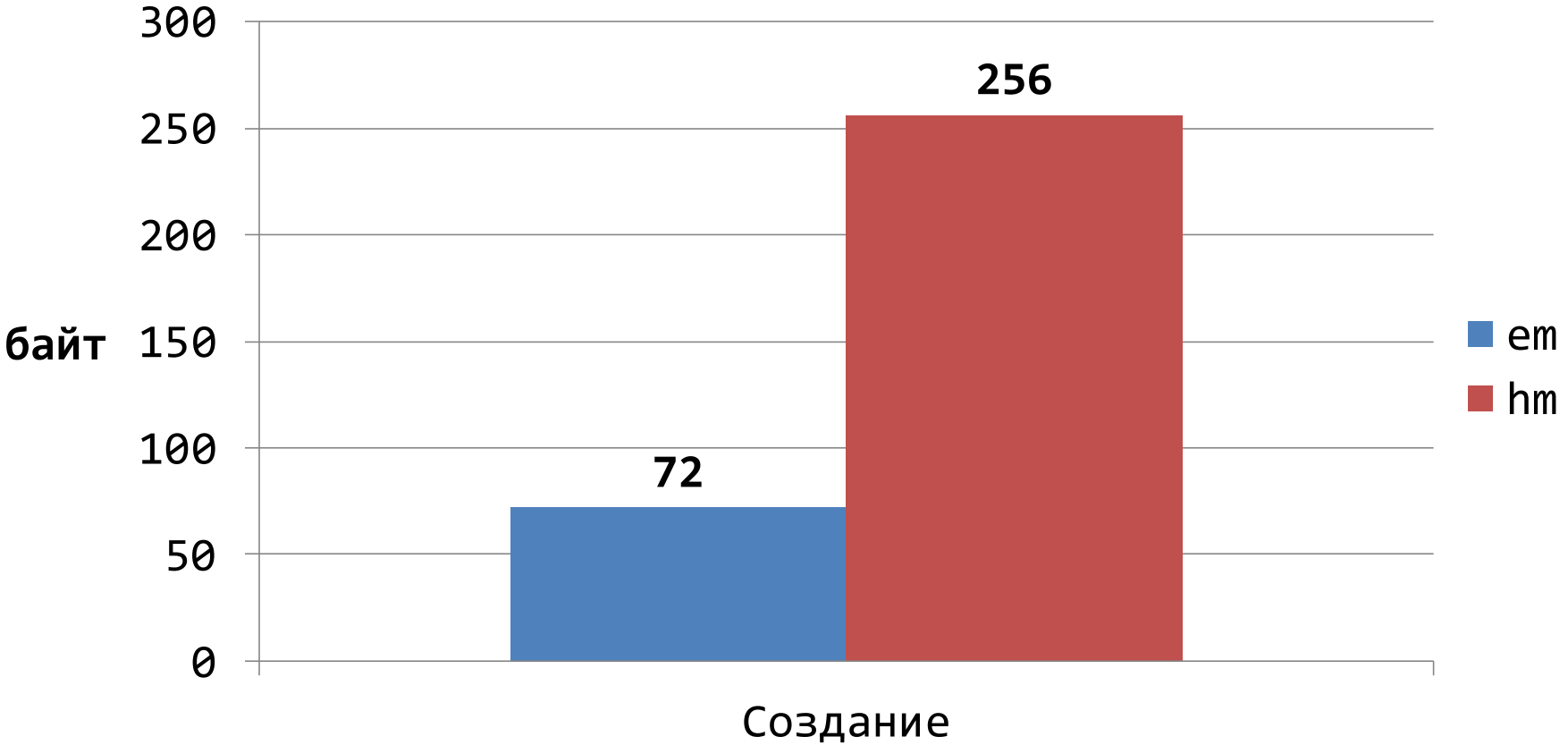
```java
@Benchmark
public Object hm() {
 var m = new HashMap<>();
 m.put(MarginLeft,   1);
 m.put(MarginRight,  2);
 m.put(MarginTop,    3);
 m.put(MarginBottom, 4);
 return m;
}
```

```java
@Benchmark
public Object em() {
 var m = new EnumMap<>(Constants.class);
 m.put(MarginLeft,   1);
 m.put(MarginRight,  2);
 m.put(MarginTop,    3);
 m.put(MarginBottom, 4);
 return m;
}
```

Посчитаем (время)



Посчитаем (память)



Не всё так просто: o.s.a.f.CglibAopProxy

```
Map<String, Integer> map = new HashMap<>();
```

Не всё так просто: o.s.a.f.CglibAopProxy

```
Map<String, Integer> map = new HashMap<>();


getCallbacks(Class<?> rootClass) {
  Method[] methods = rootClass.getMethods();

  for (int x = 0; x < methods.length; x++) {
    map.put(methods[x].toString(), x);
  }
}
```

Не всё так просто: o.s.a.f.CglibAopProxy

```
Map<String, Integer> map = new HashMap<>();


accept(Method method) {
  String key = method.toString();

  // key используется как ключ
}
```

Сколько тратим?

```
Method method = getClass().getMethod("toString");

@Benchmark
public String methodToString() { return method.toString(); }
```

Сколько тратим? А строка-то небольшая...

```
Method method = getClass().getMethod("toString");

@Benchmark
public String methodToString() { return method.toString(); }
```

```
methodToString                          85,4 ±  1,3   ns/op
methodToString:·gc.alloc.rate.norm       344 ±  0,0    B/op
```

```
"public java.lang.String java.lang.Object.toString()"
```

На больших строках теряем больше

```
Method method = getClass().getMethod("getInstance");

public MethodToStringBenchmark getInstance()
throws ArrayIndexOutOfBoundsException { /* ... */ }
```

```
methodToString                          186,4 ±  1,5   ns/op
methodToString:·gc.alloc.rate.norm       1075 ±  0,0    B/op
```

```
"public tsypanov.strings.reflection.MethodToStringBenchmark
tsypanov.strings.reflection.MethodToStringBenchmark.getInstance()
throws java.lang.ArrayIndexOutOfBoundsException"
```

CglibAopProxy: на первый взгляд совсем безнадёжно

```
Map<String, Integer> map = new HashMap<>();

getCallbacks(Class<?> rootClass) {
  Method[] methods = rootClass.getMethods();

  for (int x = 0; x < methods.length; x++) {
    map.put(methods[x].toString(), x);
  }
}

accept(Method method) {
  String key = method.toString(); /* key используется как ключ */
}
```

CglibAopProxy: на второй – не очень

```
Map<String, Integer> map = new HashMap<>();

getCallbacks(Class<?> rootClass) {
  Method[] methods = rootClass.getMethods();

  for (int x = 0; x < methods.length; x++) {
    map.put(methods[x].toString(), x);
  }
}

accept(Method method) {
  String key = method.toString(); /* key используется как ключ */
}
```

public final class java.lang.reflect.Method

+ реализует equals() / hashCode()
+ неизменяемый *

- не Comparable
- нетождественный hashCode() (и отсутствует его кэширование)

```
public int hashCode() {
  int h1 = getDeclaringClass().getName().hashCode();
  int h2 = getName().hashCode();
  return h1 ^ h2;
}
```

public final class java.lang.reflect.Method

+ реализует equals() / hashCode()
+ неизменяемый *

- не Comparable
- нетождественный hashCode() (и отсутствует его кэширование)

```
public int hashCode() {
  int h1 = getDeclaringClass().getName().hashCode();
  int h2 = getName().hashCode();
  return h1 ^ h2;
}
```

* setAccessible(**boolean**) не считается ;)

С этим можно жить, поэтому в 5.2 M1 теперь так

```
Map<String, Integer> map = new HashMap<>();
Map<Method, Integer> map = new HashMap<>();

getCallbacks(Class<?> rootClass) {
  Method[] methods = rootClass.getMethods();

  for (int x = 0; x < methods.length; x++) {
    map.put(methods[x].toString(), x);
    map.put(methods[x], x);
  }
}
```

https://github.com/spring-projects/spring-framework/pull/22258

AspectConfig

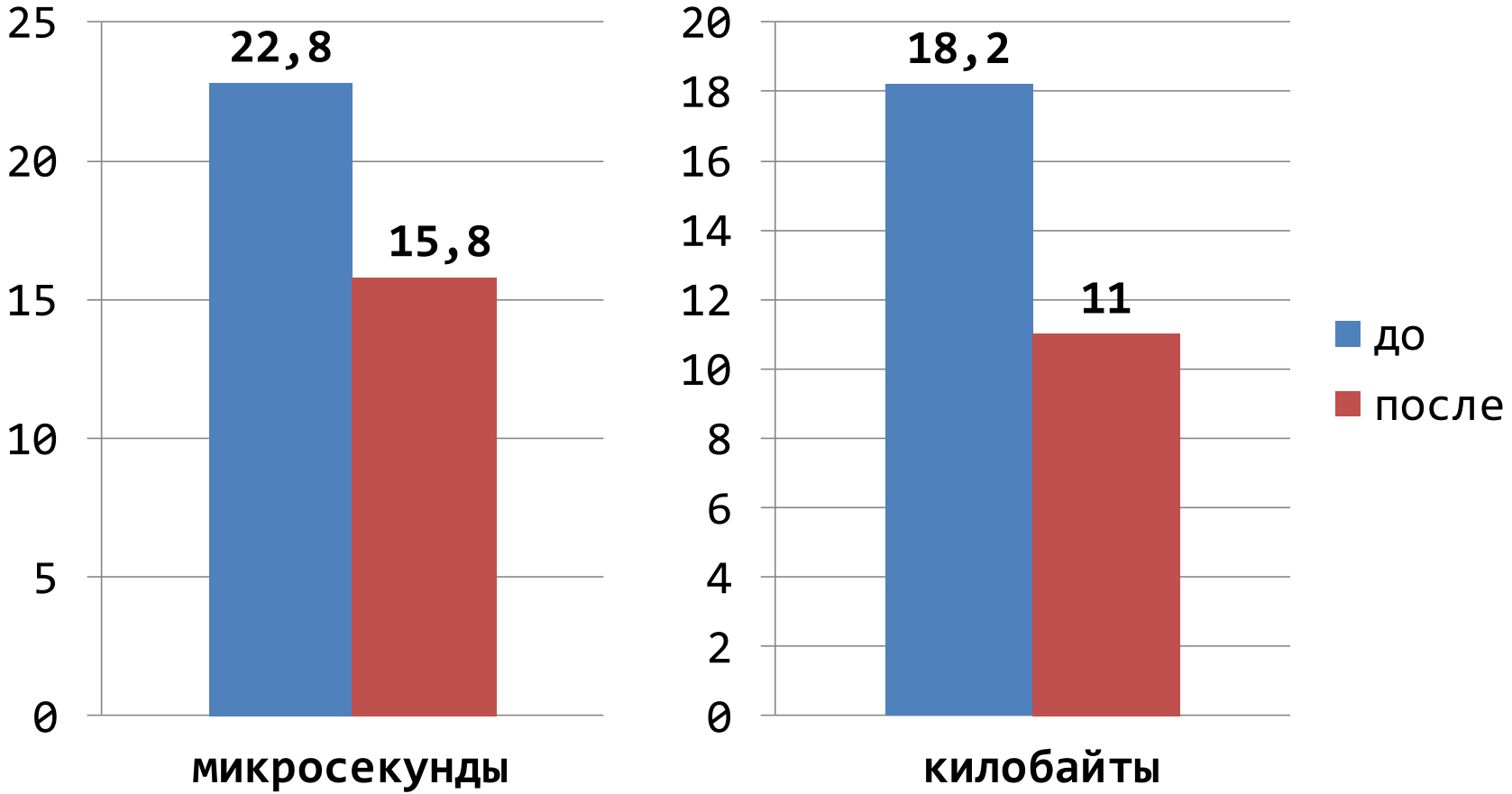
```
@Bean
ServiceAspect serviceAspect()     { return new ServiceAspect(); }

@Bean @Scope(SCOPE_PROTOTYPE)
AspectedService aspectedService() { return new ServiceImpl();   }

@Bean
AbstractAutoProxyCreator proxyCreator() {
  var factory = new AnnotationAwareAspectJAutoProxyCreator();
  factory.setProxyTargetClass(true);
  factory.setFrozen(true);
  return factory;
}
```

Сколько выиграли?

Создание 1 прототипа с 1 аспектом на 1 метод, JDK 11, Spring 5.2.*

AspectPrototypeBenchmark
https://github.com/stsypanov/spring-benchmark

Считаем


25
20
15
10
5
0
22,8
15,8
микросекунды
20
18
16
14
12
10
8
6
4
2
0
18,2
11
килобайты
до
после

Разработчик, помни!

Прежде чем вызывать toString() и использовать строку как ключ, подумай – действительно ли тебе это нужно.

Возможно, объект сам может стать ключом.

Составные ключи

Есть в JDK класс ObjectStreamClass...

```
class ObjectStreamClass {

  private static class Caches {
    Map<FieldReflectorKey, Reference<?>> reflectors = new CHM<>();
  }

  private static class FieldReflectorKey extends {
    /* ... */
  }

}
```

...а в нём FieldReflectorKey

```
private final String sigs;
private final int hash;

FieldReflectorKey(Class<?> cl, ObjectStreamField[] fields) {
  StringBuilder sb = new StringBuilder();
  for (ObjectStreamField f : fields) {
    sb.append(f.getName()).append(f.getSignature());
  }
  sigs = sb.toString();
  hash = System.identityHashCode(cl) + sigs.hashCode();
}
```

...а в нём загвоздка JDK-6996807

```
private final String sigs;    <---
private final int hash;

FieldReflectorKey(Class<?> cl, ObjectStreamField[] fields) {
  StringBuilder sb = new StringBuilder();
  for (ObjectStreamField f : fields) {
    sb.append(f.getName()).append(f.getSignature());
  }
  sigs = sb.toString();
  hash = System.identityHashCode(cl) + sigs.hashCode();
}
```

Посчитаем

```
@Setup
public void setup() {
  String signature = "Ljava/lang/String;";
  fields    = new ObjectStreamField[3];
  fields[0] = new ObjectStreamField("groupId",    signature);
  fields[1] = new ObjectStreamField("artifactId", signature);
  fields[2] = new ObjectStreamField("version",    signature);
}

@Benchmark
public Object original() { return new FieldReflectorKey(fields); }
```

Создание FieldReflectorKey весьма недёшево

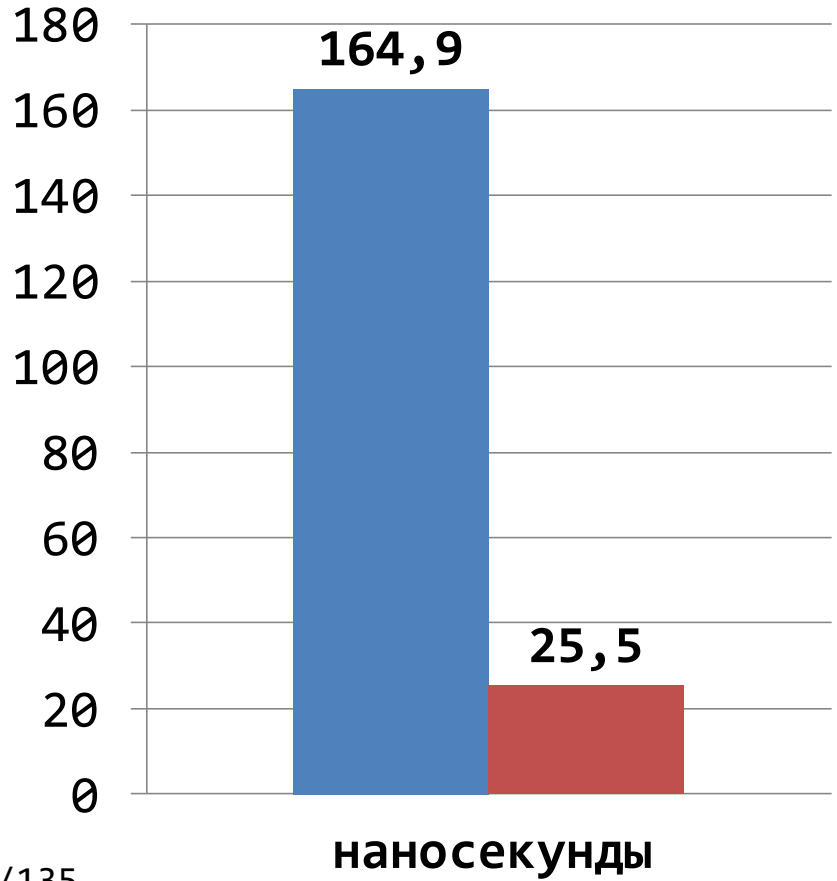
original	**164,9**	**± 3,5**	ns/op
original:·gc.alloc.rate.norm	**504**	**± 0,0**	B/op

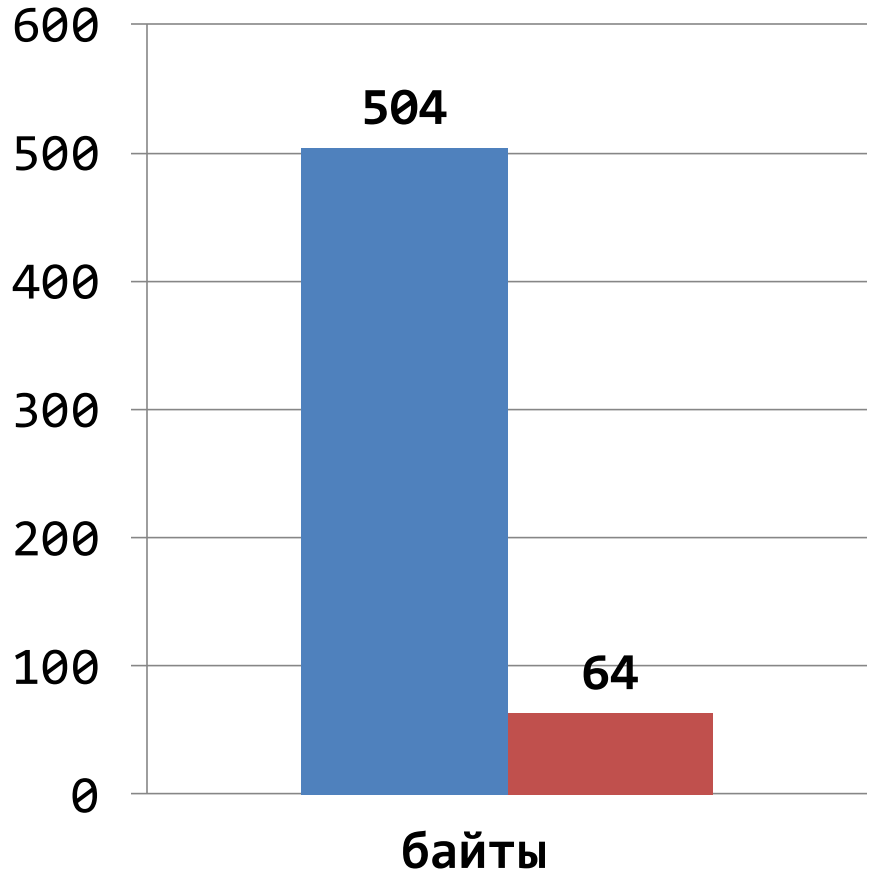
...полкилобайта на ключ – ни в какие рамки!

Пересядь с иглы StringBuilder-а на массив

```
private final String[] sigs;    <---
private final int hash;

FieldReflectorKey(Class<?> cl, ObjectStreamField[] fields) {
  sigs = new String[2 * fields.length];
  for (int i = 0; i < fields.length; i++) {
    ObjectStreamField f = fields[i];
    sigs[2 * i] = f.getName();
    sigs[2 * i + 1] = f.getSignature();
  }
  hash = System.identityHashCode(cl) + Arrays.hashCode(sigs);
}
```

Посчитаем (JDK 13)





Для внеклассного чтения

https://bugs.openjdk.java.net/browse/JDK-6996807

http://cr.openjdk.java.net/~igerasim/6996807/00/webrev/

В сухом остатке

...SPECjvm2008:serial improves a little bit with this patch, and the allocation rate is down ~5%.

https://bugs.openjdk.java.net/browse/JDK-6996807

Снова Спринг (с сокращениями)

```
class StaticMessageSource {
  Map<String, String> messages = new HashMap<>();

  String resolve(String code, Locale locale) {
    return messages.get(code + '_' + locale);
  }

  void addMessage(String code, Locale locale, String msg) {
    messages.put(code + '_' + locale, msg);
  }

}
```

Посчитаем: дороговато

```
String code = "code1";
Locale locale = Locale.getDefault();

Map<String, Object> map = new HashMap<>();

@Benchmark
public String concatenation() {
  return map.get(code + '_' + locale);
}
```

```
concatenation                          41,9 ±  4,4   ns/op
concatenation:·gc.alloc.rate.norm       120 ±  0,0    B/op
```

Решение: составной ключ

```
@EqualsHashCode
@RequiredArgsConstructor
private static final class Key {
  private final String code;
  private final Locale locale;
}
```

Решение: составной ключ

```
@EqualsHashCode
@RequiredArgsConstructor
private static final class Key {
  private final String code;
  private final Locale locale;
}
```

```
Arrays.asList(code, locale);
```

Решение: составной ключ или вложенный словарь

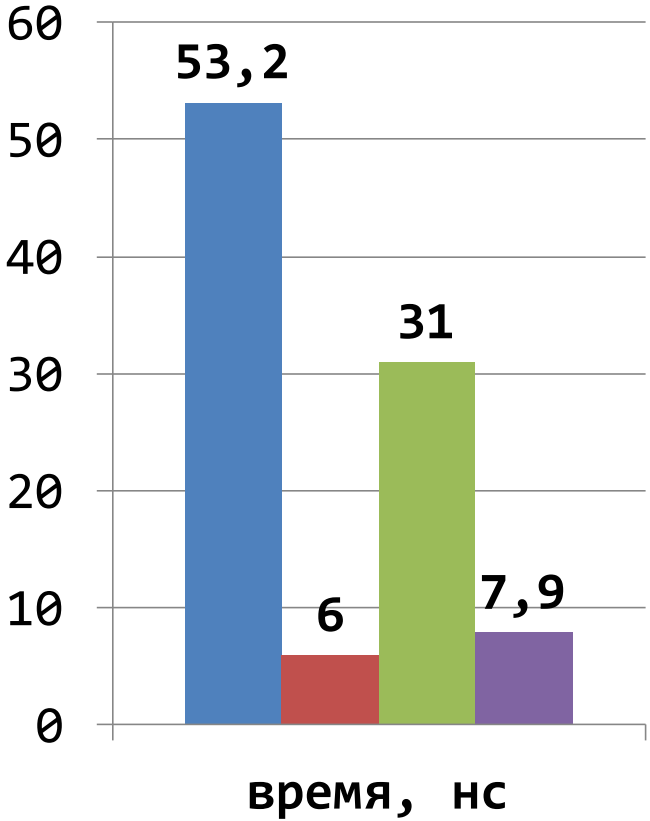
```
@EqualsHashCode
@RequiredArgsConstructor
private static final class Key {
  private final String code;
  private final Locale locale;
}
```

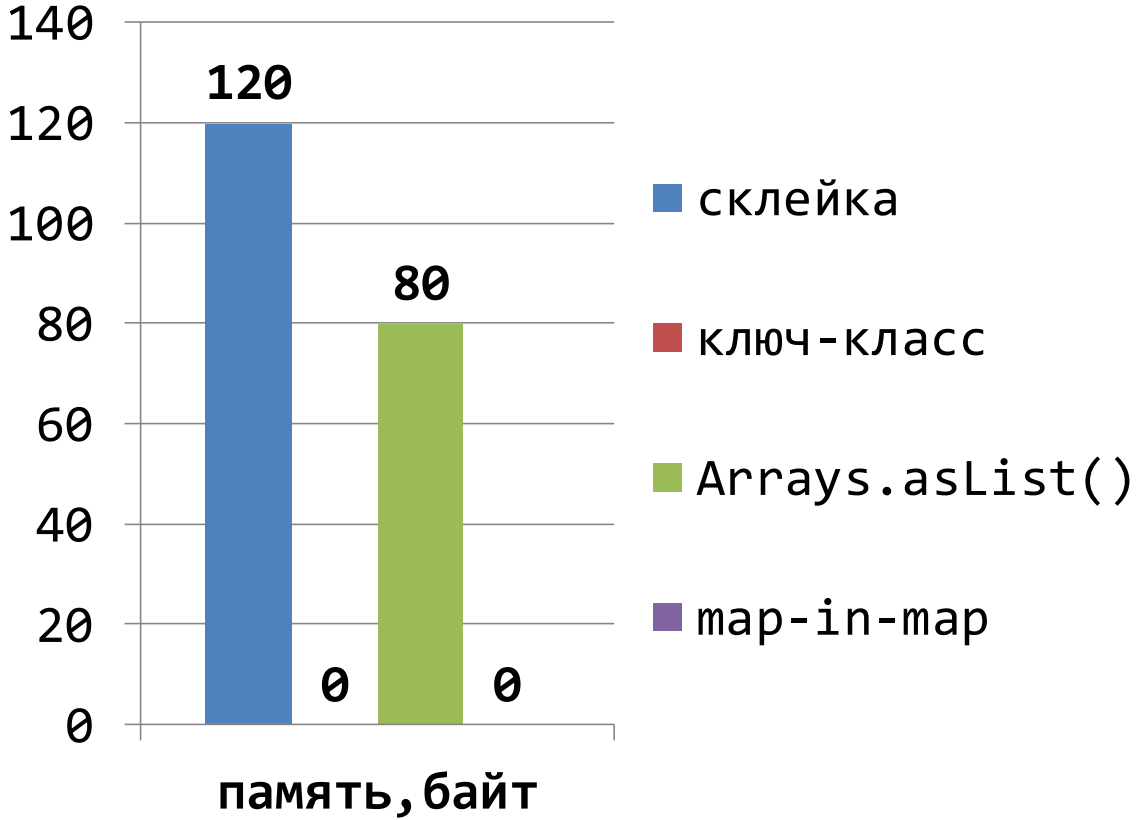
```
Arrays.asList(code, locale);
```

```
Map<String, Map<Locale, MessageHolder>> msgMap = new HashMap<>();
```

https://github.com/spring-projects/spring-framework/pull/22451

Посчитаем


60
50
40
30
20
10
0
53,2
6
31
7,9
время, нс


140
120
100
80
60
40
20
0
120
0
80
0
память,байт
склейка
ключ-класс
Arrays.asList()
map-in-map

Разработчик, помни!

Склейка строк для создания ключа – это плохо, ибо

- требует много времени
- много мусора (StringBuilder и иже с ним)
- выход есть – отдельный класс для составного ключа
  в крайнем случае подойдёт массив или Arrays.asList()

Склейка

Как клеить строки?

~~Кто знает самый лучший способ?~~

Это неправильный вопрос.

Правильный вопрос звучит так: **А нужно ли вообще их клеить?**

Исполнение: o.s.core.ResolvableType::toString

```
StringBuilder result = new StringBuilder(resolved.getName());

if (hasGenerics()) {
  result.append('<');
  result.append(arrayToDelimitedString(getGenerics(), ", "));
  result.append('>');
}

return result.toString();
```

Исполнение 1: условие истинно

```
StringBuilder result = new StringBuilder(resolved.getName());

if (hasGenerics()) {
  result.append('<');
  result.append(arrayToDelimitedString(getGenerics(), ", "));
  result.append('>');
}

return result.toString();
```

Исполнение 2: условие ложно

```
StringBuilder result = new StringBuilder(resolved.getName());

if (hasGenerics()) {
  result.append('<');
  result.append(arrayToDelimitedString(getGenerics(), ", "));
  result.append('>');
}

return result.toString();
```

Так-то лучше (5.2 M1)

```
if (hasGenerics()) {
  return resolved.getName()
          + '<'
          + arrayToDelimitedString(getGenerics(), ", ")
          + '>';
}

return resolved.getName();
```

https://github.com/spring-projects/spring-framework/pull/22593

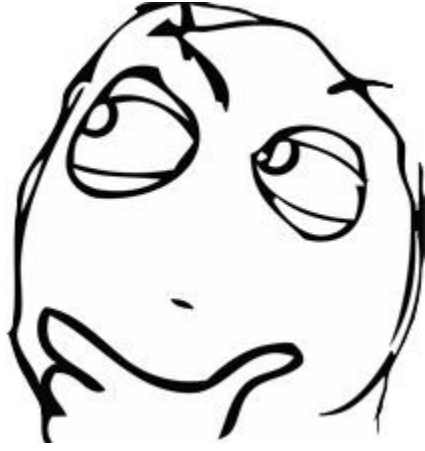
Разработчик, помни!

Заставь дурака Б-гу молиться – он и лоб расшибёт.

Русская пословица

Блин, наверное, будет тормозить

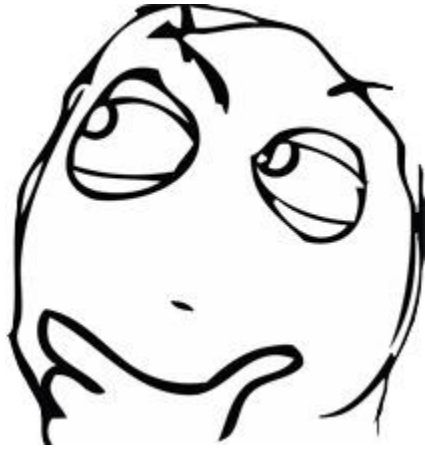
```
public String concat(Data data) {
  return data.str1 + '/' + data.str2;
}
```



Нет склейки – нет проблем

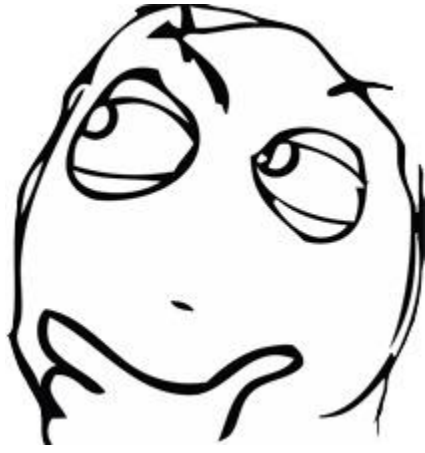
```
public String concat(Data data) {
  return data.str1 + '/' + data.str2;
}
```

```
public String format(Data data) {
  return String.format("%s/%s", data.str1, data.str2);
}
```

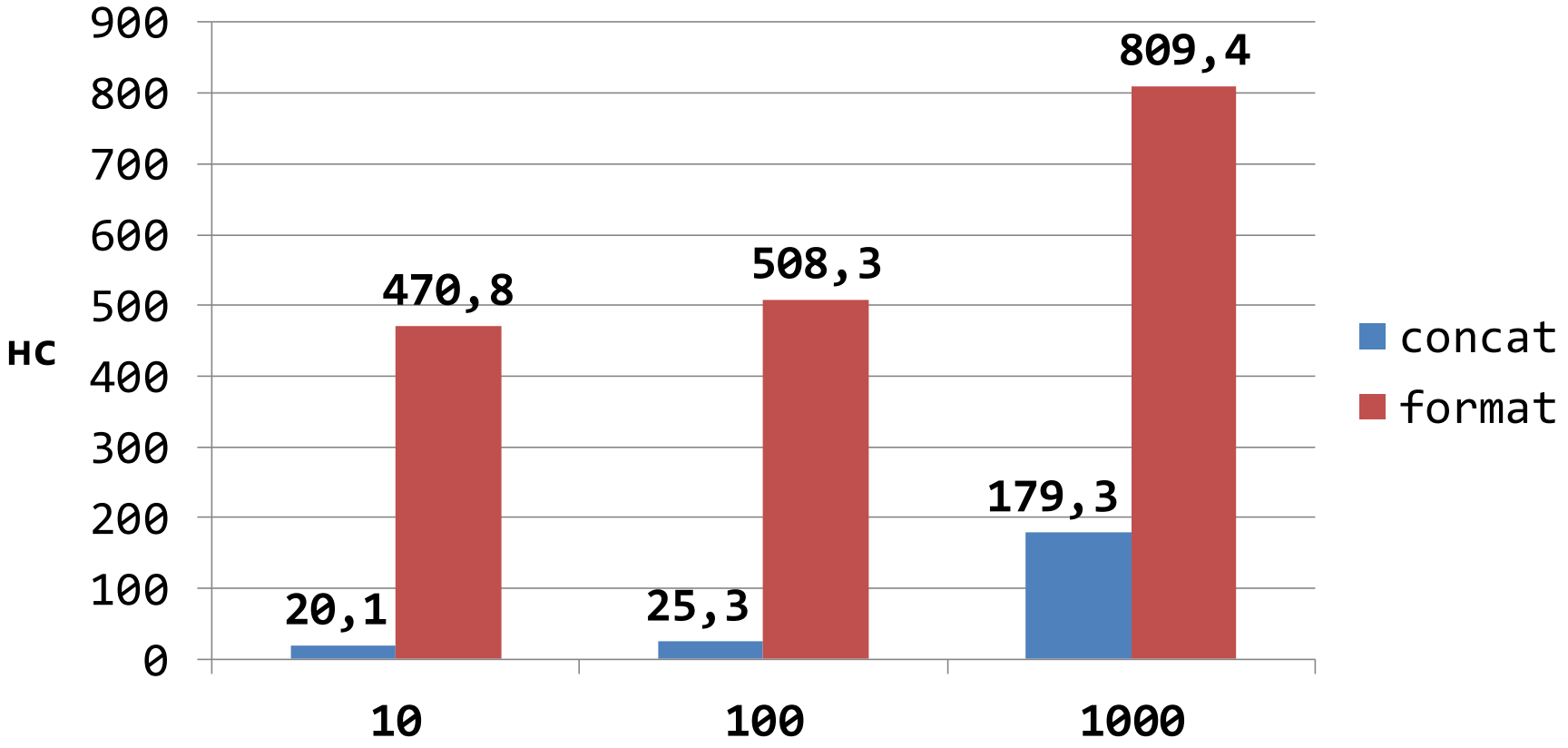


Нет склейки – нет проблем (а вот и есть)

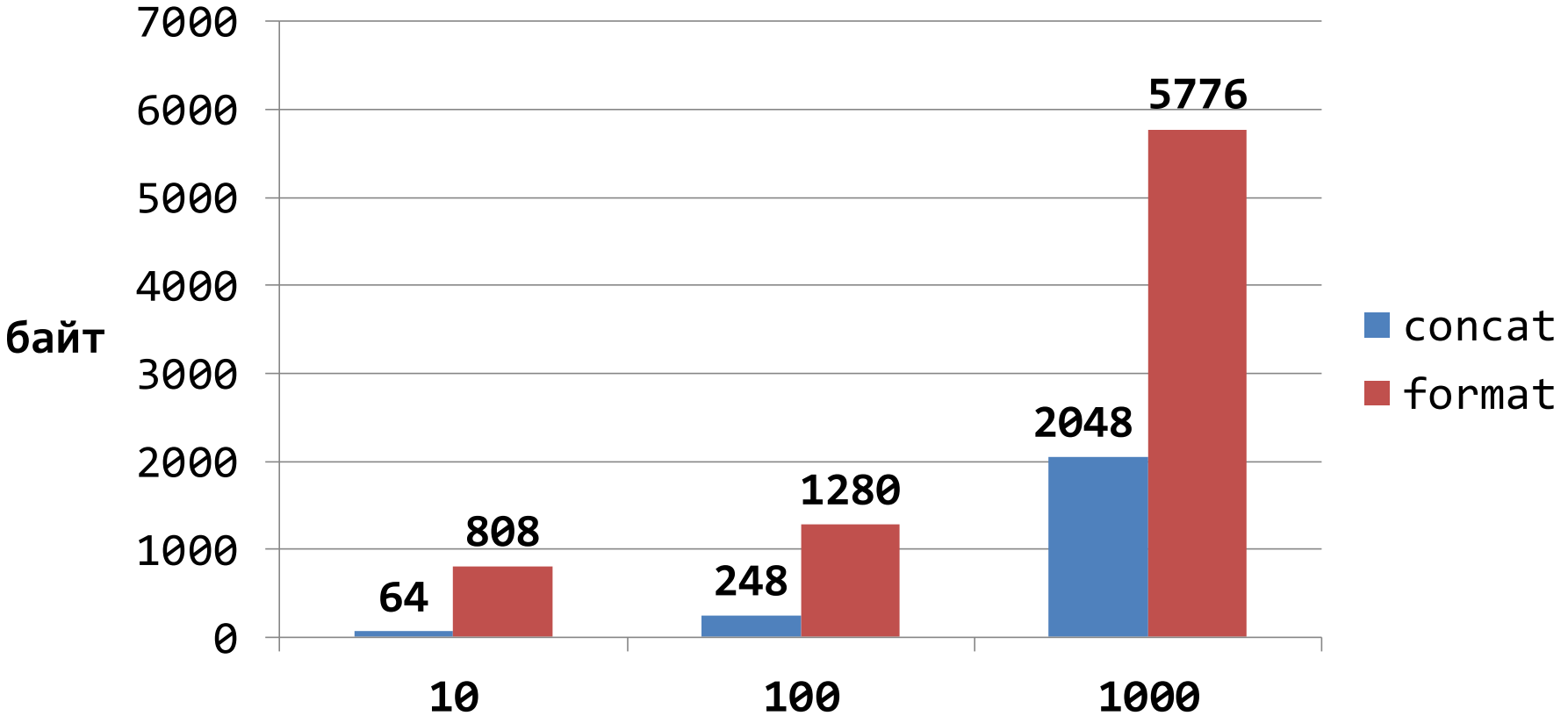
```
@Benchmark
public String concat(Data data) {
  return data.str1 + '/' + data.str2;
}

@Benchmark
public String format(Data data) {
  return String.format("%s/%s", data.str1, data.str2);
}
```



Посчитаем (время)



Посчитаем (память)



Склейка: если всё-таки нужно

Задача: сущность и p6spy

```java
@Entity
public class ReportEntity {
  @Id
  private long id;

  @Lob
  private byte[] reportContent;
}
```

https://github.com/p6spy/p6spy

https://p6spy.readthedocs.io/en/latest

Задача: сущность и p6spy

```java
@Entity
public class ReportEntity {
  @Id
  private long id;

  @Lob
  private byte[] reportContent;
}
```

```
insert into reports(id, content) values(123, 'A34C4DE31F...')
```

com.p6spy.engine.common.Value

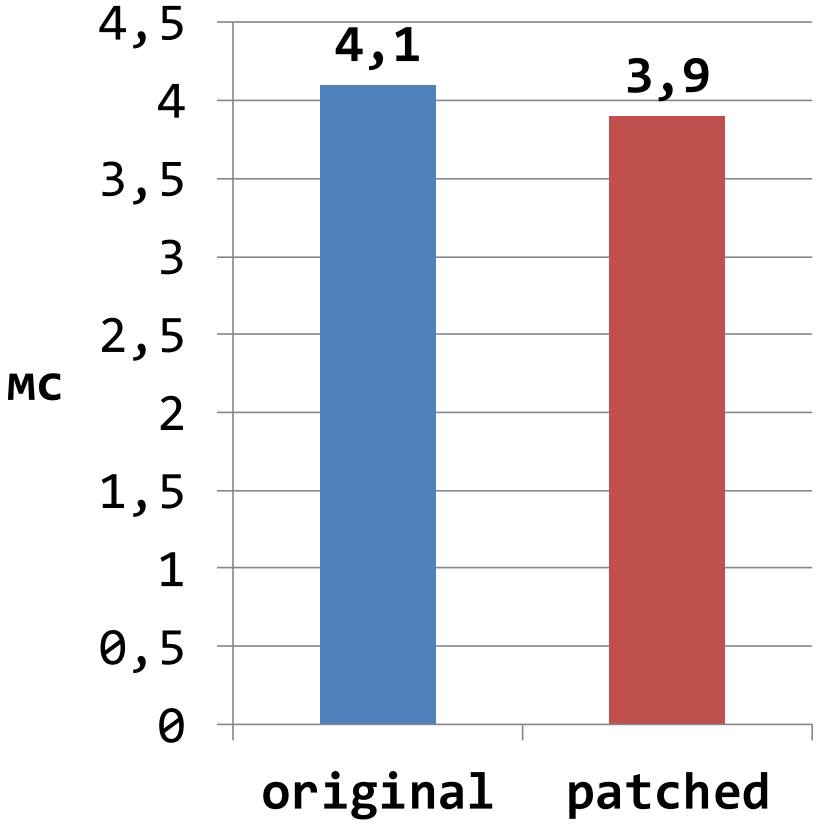
```
private String toHexString(byte[] bytes) {
  StringBuilder sb = new StringBuilder();

  for (byte b : bytes) {
    int temp = (int) b & 0xFF;
    sb.append(HEX_CHARS[temp / 16]);
    sb.append(HEX_CHARS[temp % 16]);
  }

  return sb.toString();
}
```

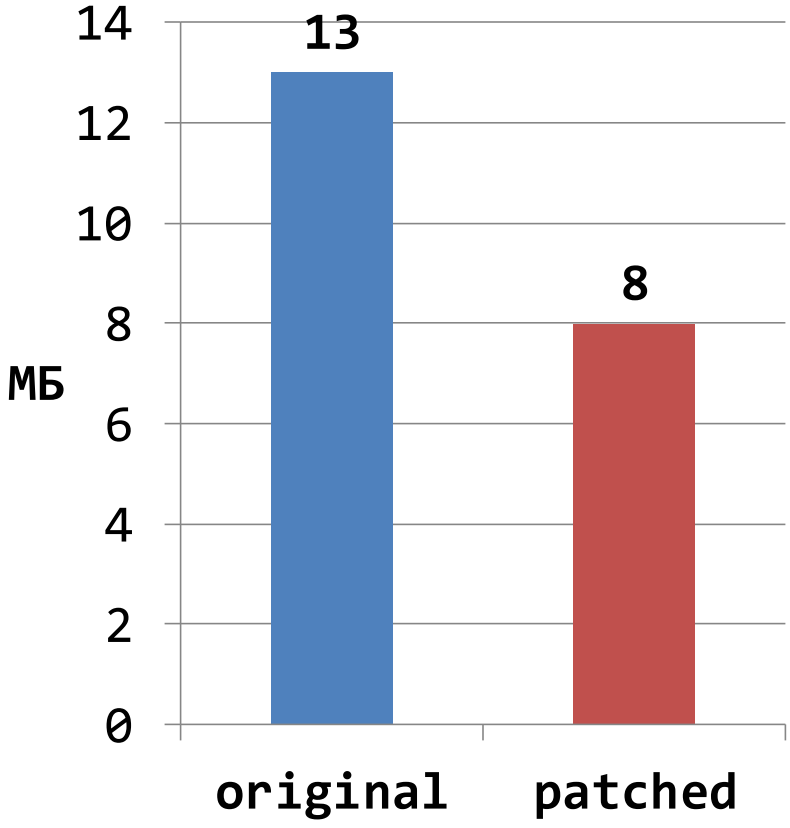
Задача про буферизацию: семь раз отмерь (3.8.0)

```
private String toHexString(byte[] bytes) {
  StringBuilder sb = new StringBuilder(bytes.length * 2);

  for (byte b : bytes) {
    int temp = (int) b & 0xFF;
    sb.append(HEX_CHARS[temp / 16]);
    sb.append(HEX_CHARS[temp % 16]);
  }

  return sb.toString();
}
```

https://github.com/p6spy/p6spy/pull/450

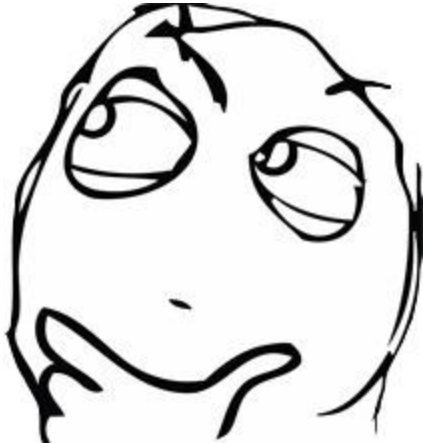
По времени выигрыш невелик (JDK 8, отчёт 1 Мб)





Теряем на проверках и доступе (JDK 8)

```
@Override
public AbstractStringBuilder append(char c) {
  ensureCapacityInternal(count + 1);
  value[count++] = c;
  return this;
}
```

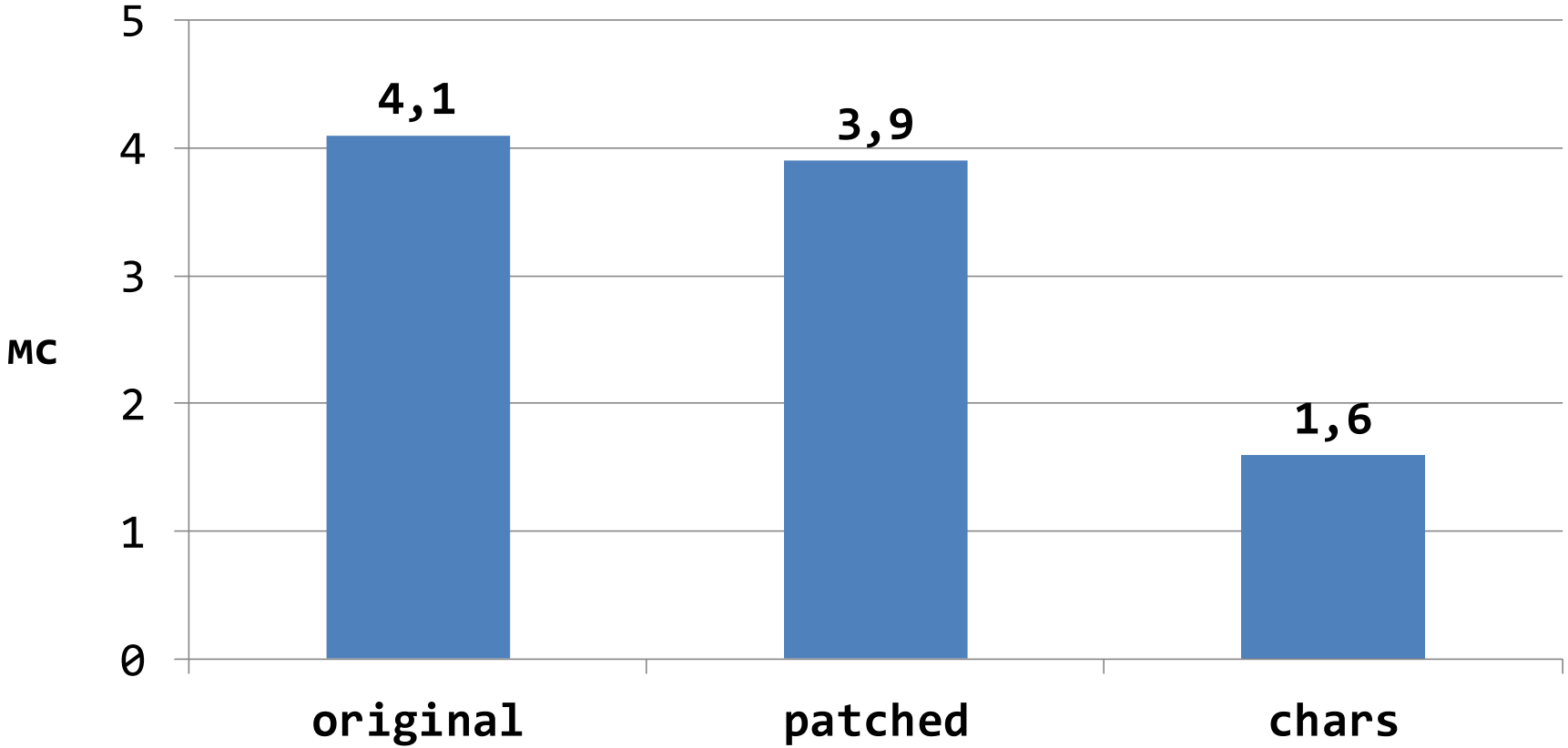


Один раз отрежь (3.8.2)

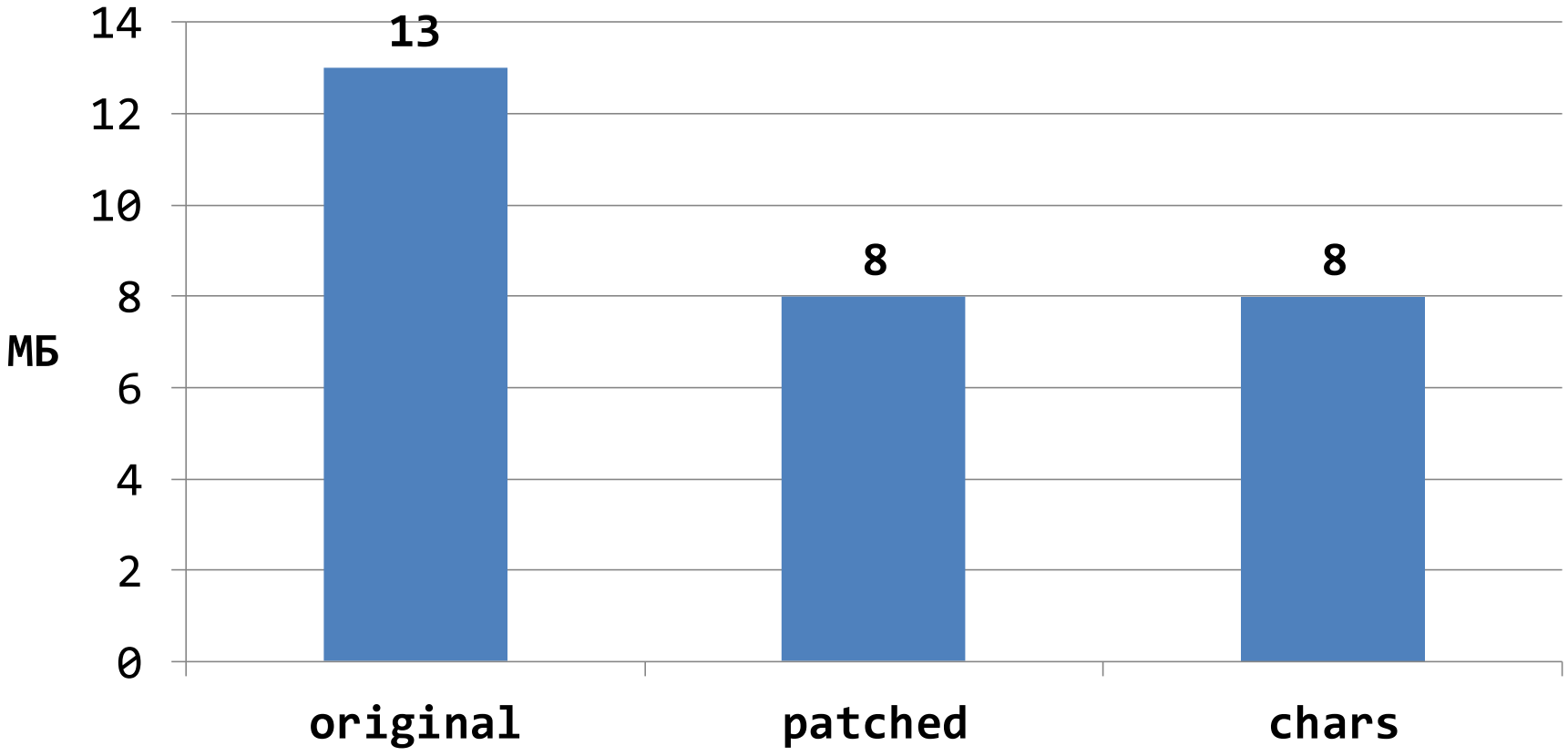
```
private String toHexString(byte[] bytes) {
  char[] result = new char[bytes.length * 2];
  int idx = 0;
  for (byte b : bytes) {
    int temp = (int) b & 0xFF;
    result[idx++] = HEX_CHARS[temp / 16];
    result[idx++] = HEX_CHARS[temp % 16];
  }

  return new String(result);
}
```

https://github.com/p6spy/p6spy/pull/463

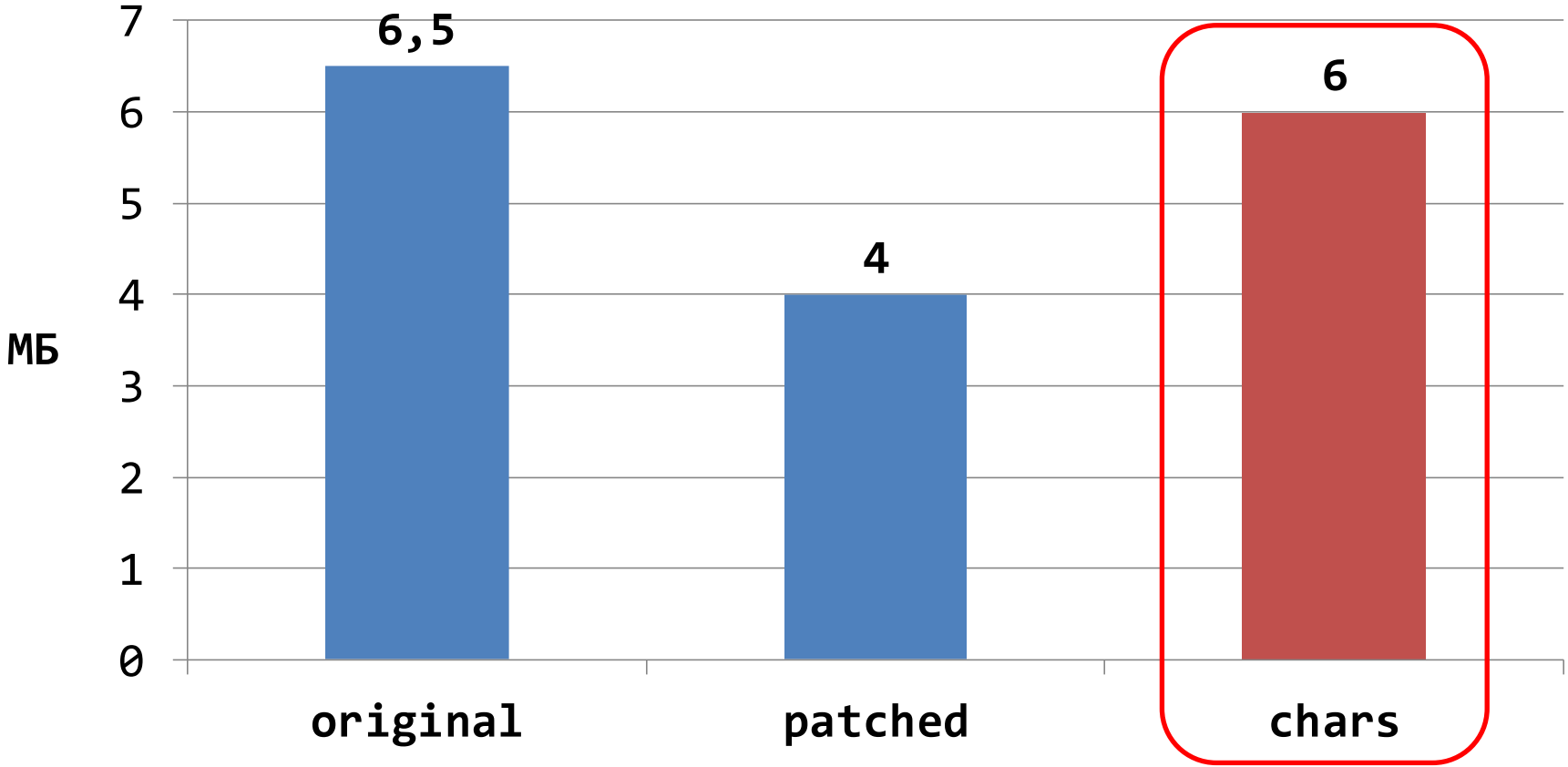
Задача про буферизацию: в сухом остатке (JDK 8)



Задача про буферизацию: в сухом остатке (JDK 8)



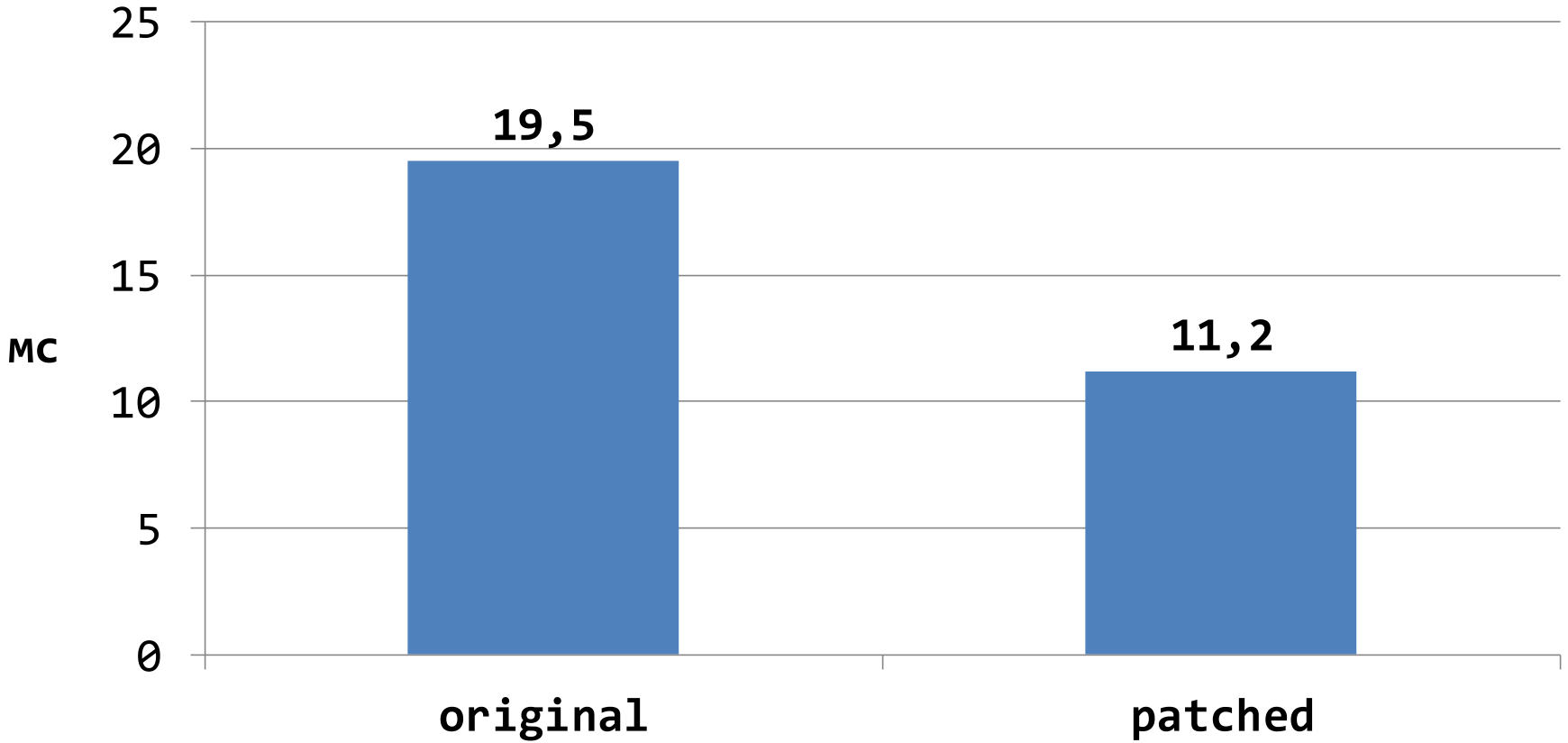
Задача про буферизацию: внезапно (JDK 13)



После JDK 9

```
abstract class AbstractStringBuilder {

  byte[] value;

  public AbstractStringBuilder append(char c) {
    ensureCapacityInternal(count + 1);
    if (isLatin1() && StringLatin1.canEncode(c)) {
      value[count++] = (byte)c;
    }
    //...
  }
}
```

Сохранение в БД сущности с отчётом весом 1 Мб



Разработчик, помни!

Задача про буферизацию:

- проверка границ
- расширение хранилища
- копирование данных

Решение:

> Семь раз отмерь – один раз отрежь.

Выше описан довольно редкий случай

- известно количество проходов
- известно сколько данных записывается на каждом проходе

Когда всё прозаично

сложение

```
String str = s1 + s2 + s3;
```

всё с новой строчки

```
StringBuilder sb = new StringBuilder();
sb.append(s1);
sb.append(s2);
sb.append(s3);
String str = sb.toString();
```

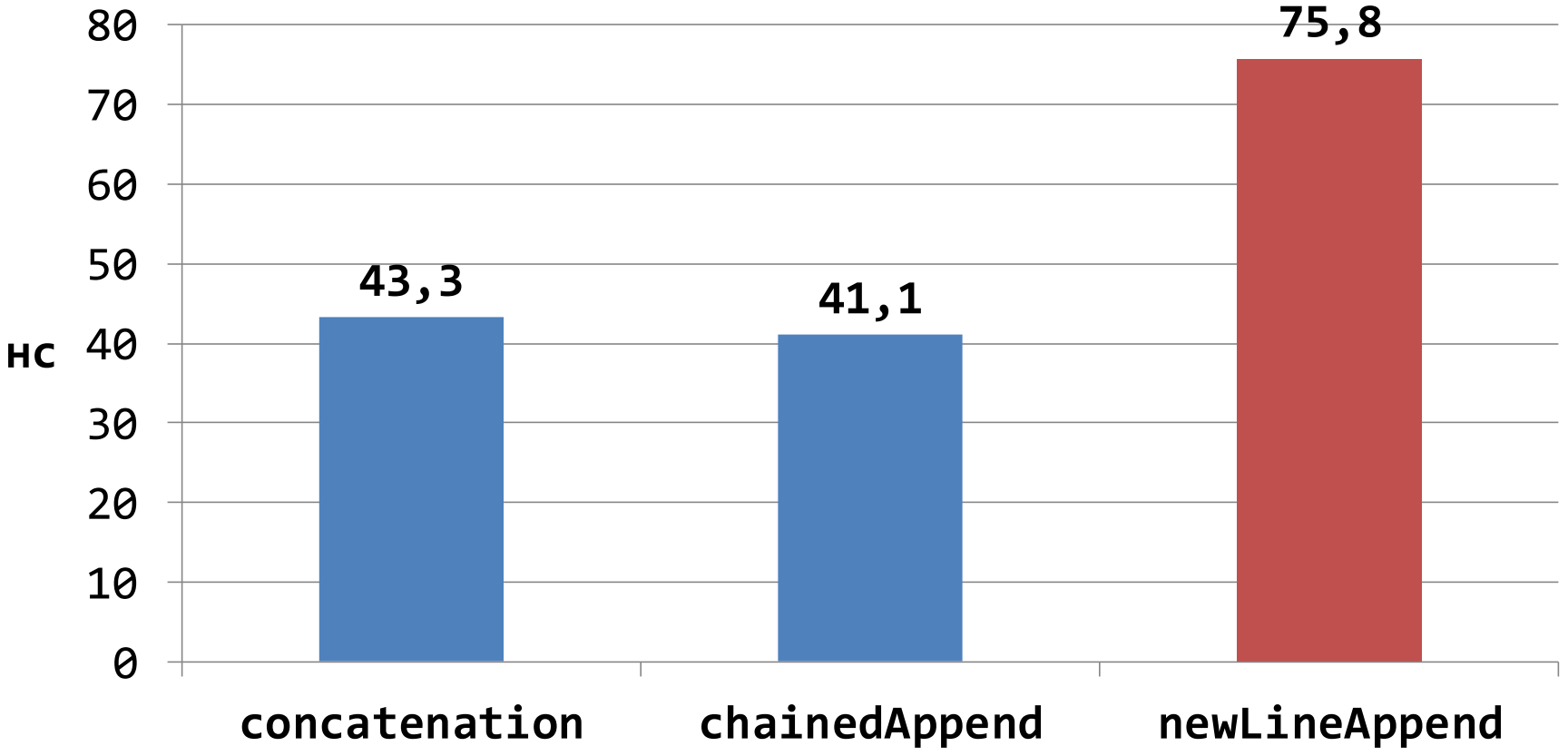
цепочка

```
String str = new StringBuilder()
                    .append(s1)
                    .append(s2)
                    .append(s3)
                    .toString();
```

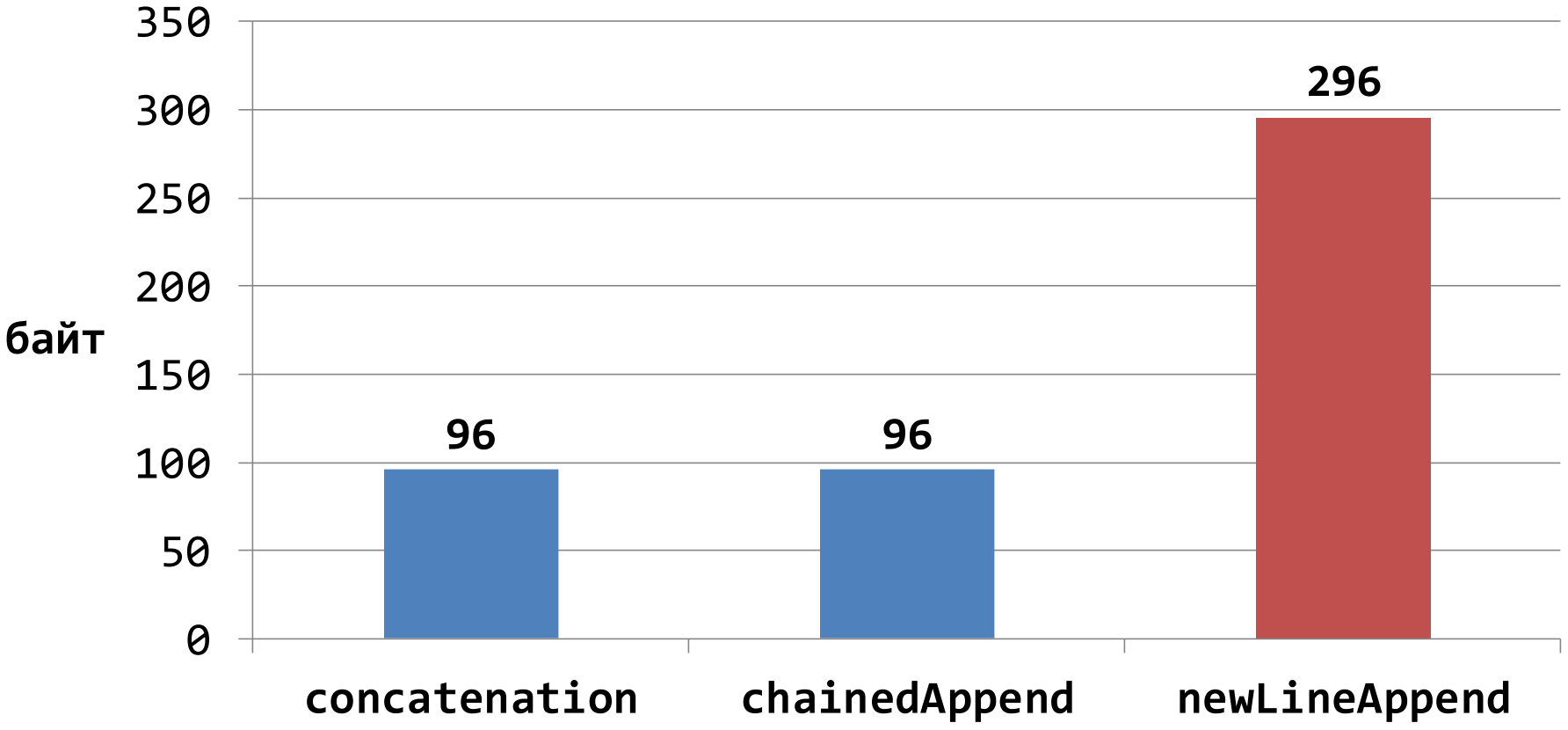
Посчитаем

```
private final String str1 = "1".repeat(10);
private final String str2 = "2".repeat(10);
private final String str3 = "3".repeat(10);
private final String str4 = "4".repeat(10);
private final String str5 = "5".repeat(10);


@Benchmark public String concatenation() { /*...*/ }

@Benchmark public String chainedAppend() { /*...*/ }

@Benchmark public String newLineAppend() { /*...*/ }
```

Посчитаем (время)



Посчитаем (память)



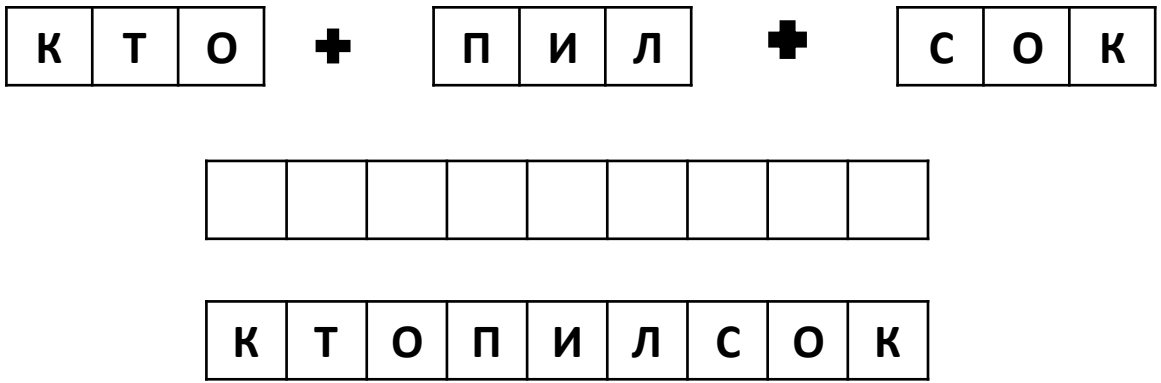
Сделаем очевидный вывод

Цепочка StringBuilder.append() или сложение через «+» значительно быстрее StringBuilder.append() с новой строки.

Сделаем очевидный вывод

Цепочка StringBuilder.append() или сложение через «+» значительно быстрее StringBuilder.append() с новой строки.

```
String str = s1 + s2 + s3;
```



Поставим вопрос иначе

цепочка цельная

```
String str = new StringBuilder()
                    .append(s1)
                    .append(s2)
                    .append(s3)
                    .toString();
```

Поставим вопрос иначе

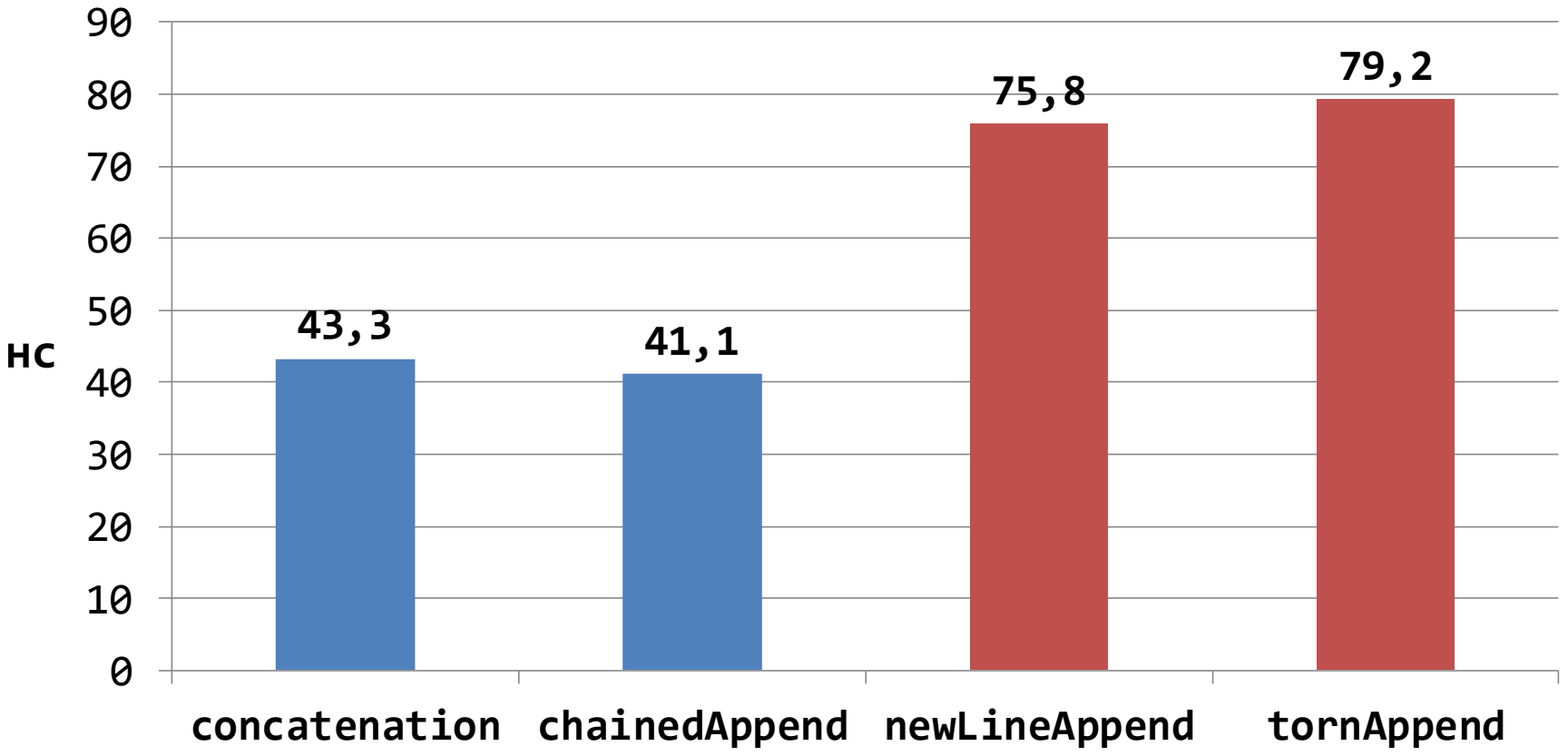
цепочка цельная

```
String str = new StringBuilder()
                    .append(s1)
                    .append(s2)
                    .append(s3)
                    .toString();
```

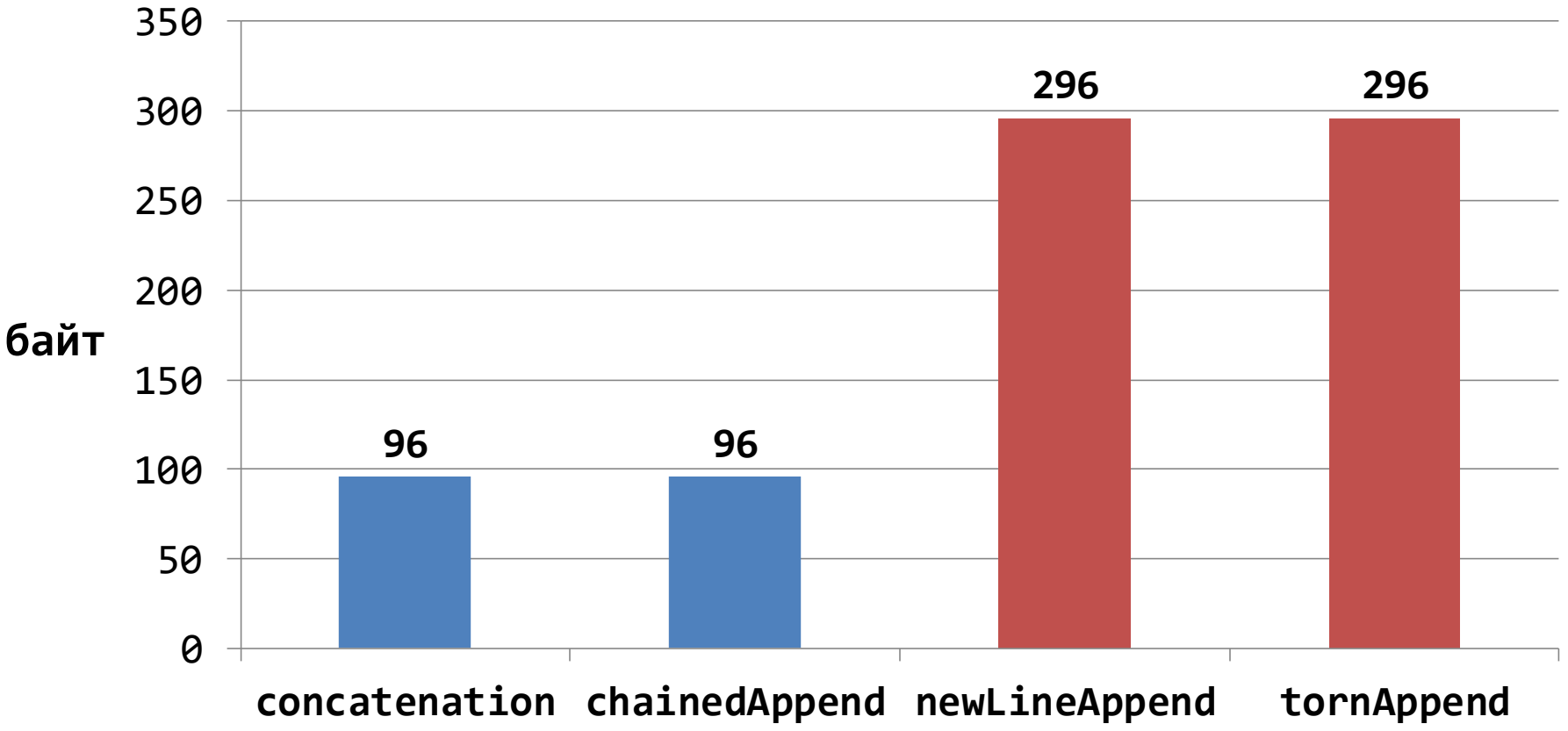
цепочка разорванная

```
StringBuilder sb = new StringBuilder()
                    .append(s1)
                    .append(s2);

if (smthTrue)  sb.append(s3);

String str = sb.toString();
```

Посчитаем (время)



Посчитаем (память)



Посчитаем (и сделаем менее очевидный вывод)

Объединение разорванной цепочки даёт конский прирост.

o.s.a.interceptor.AbstractMonitoringInterceptor

```
String createInvocationTraceName(MethodInvocation inv) {
 StringBuilder sb = new StringBuilder(getPrefix());
 Method method = inv.getMethod();
 Class<?> clazz = method.getDeclaringClass();
 if (logTargetClassInvocation && clazz.isInstance(inv.getThis())) {
  clazz = invocation.getThis().getClass();
 }
 sb.append(clazz.getName());
 sb.append('.').append(method.getName());
 sb.append(getSuffix());
 return sb.toString();
}
```

Делай раз

```
String createInvocationTraceName(MethodInvocation inv) {
 StringBuilder sb = new StringBuilder(getPrefix());
 Method method = inv.getMethod();
 Class<?> clazz = method.getDeclaringClass();
 if (logTargetClassInvocation && clazz.isInstance(inv.getThis())) {
  clazz = invocation.getThis().getClass();
 }
 sb.append(clazz.getName());
 sb.append('.').append(method.getName());
 sb.append(getSuffix());
 return sb.toString();
}
```

Делай два

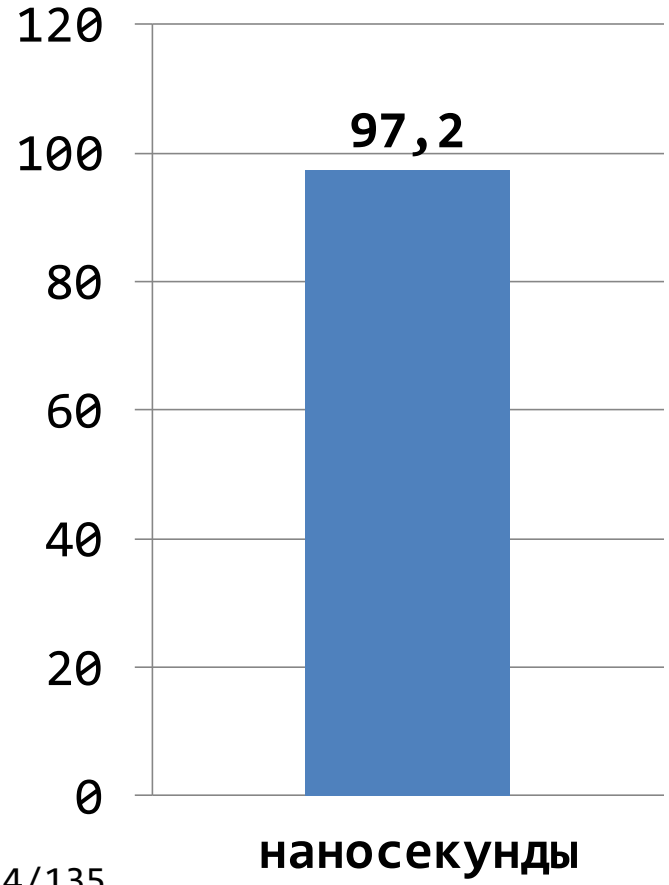
```
String createInvocationTraceName(MethodInvocation inv) {
 Method method = inv.getMethod();
 Class<?> clazz = method.getDeclaringClass();
 if (logTargetClassInvocation && clazz.isInstance(inv.getThis())) {
   clazz = invocation.getThis().getClass();
 }
 StringBuilder sb = new StringBuilder(getPrefix());
 sb.append(clazz.getName());
 sb.append('.').append(method.getName());
 sb.append(getSuffix());
 return sb.toString();
}
```

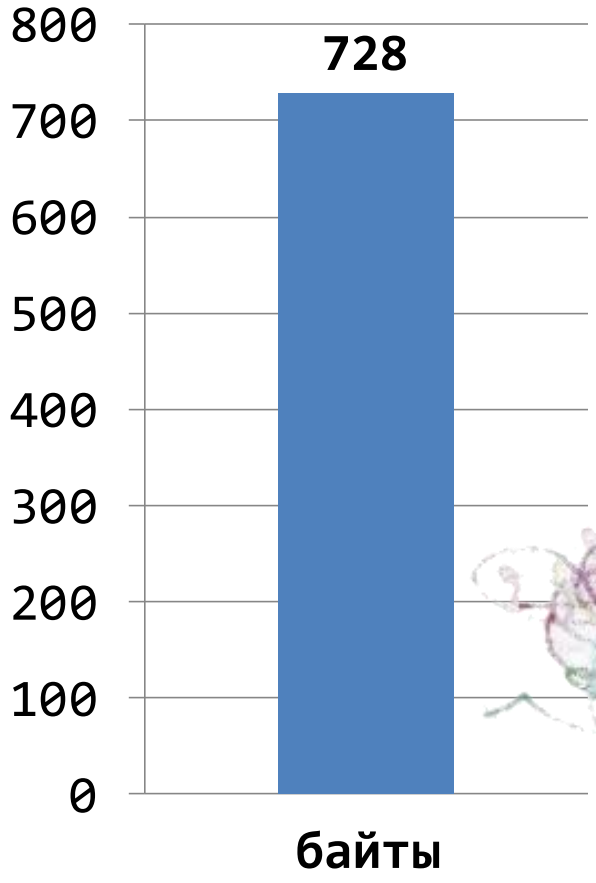
Делай три (5.2.0.RC2)

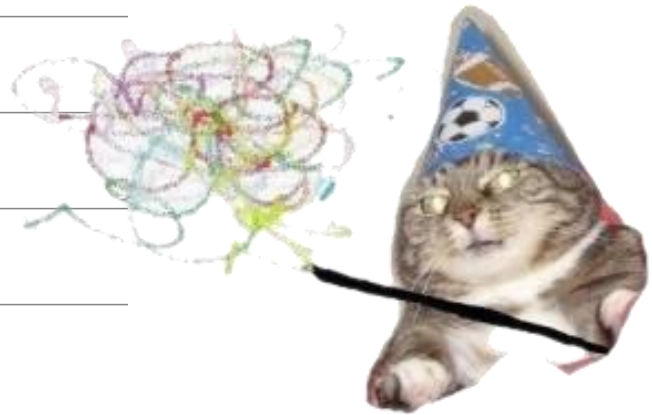
```
String createInvocationTraceName(MethodInvocation inv) {
 Method method = inv.getMethod();
 Class<?> clazz = method.getDeclaringClass();
 if (logTargetClassInvocation && clazz.isInstance(inv.getThis())) {
   clazz = invocation.getThis().getClass();
 }
 return getPrefix() + clazz.getName() + '.'
                    + method.getName() + getSuffix();
}
```

https://github.com/spring-projects/spring-framework/pull/22773

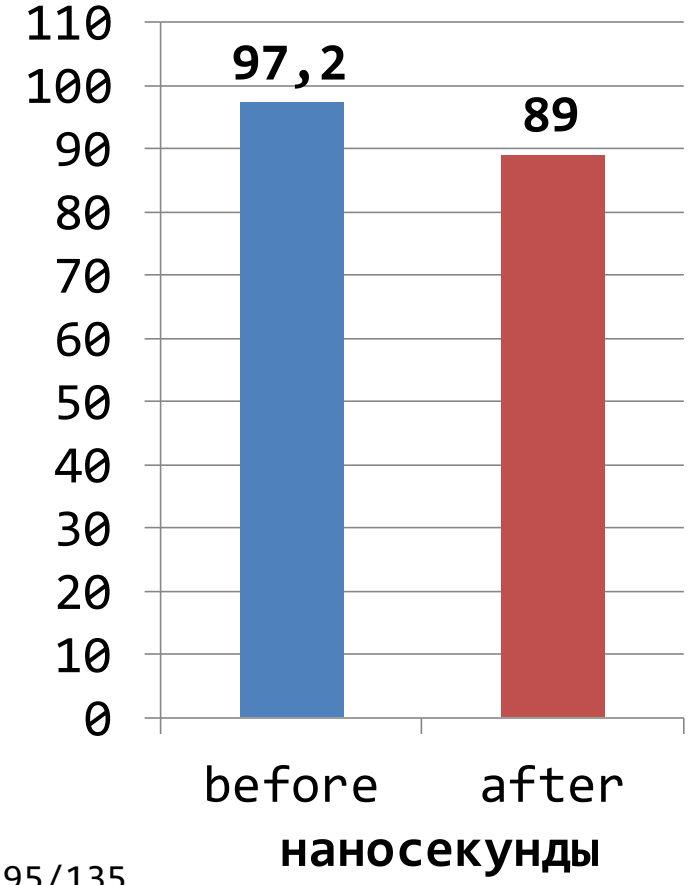
JDK 8: вжух!

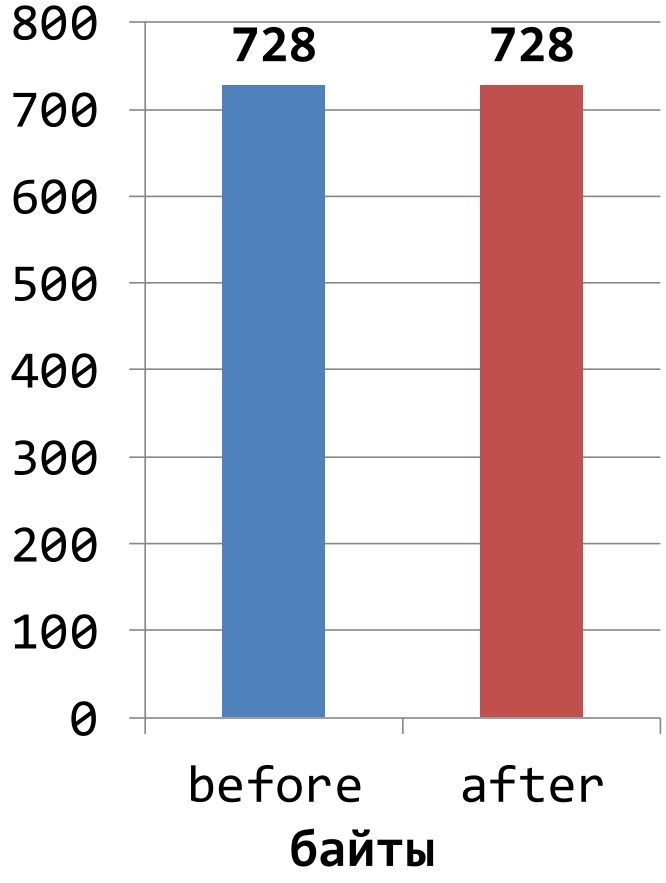

120
100
80
60
40
20
0
97,2
наносекунды


800
700
600
500
400
300
200
100
0
728
байты



JDK 8: и не вжух...





Что-то пошло не так...

```
String createInvocationTraceName(MethodInvocation inv) {

 //...

 return getPrefix() + clazz.getName() + '.'
                    + method.getName() + getSuffix();

}
```

...ужимаем

BrokenConcatenationBenchmark

```
@Benchmark
return "class " + data.clazz.getName();

@Benchmark
String clazzName = data.clazz.getName();
return "class " + clazzName;
```

```
@Setup
public void setup() {
  clazz.getName();
}
```

java.lang.Class

```
public String getName() {
  String name = this.name;
  if (name == null)
    this.name = name = getName0();
  return name;
}

// cache the name to reduce the number of calls into the VM
private transient String name;

private native String getName0();
```

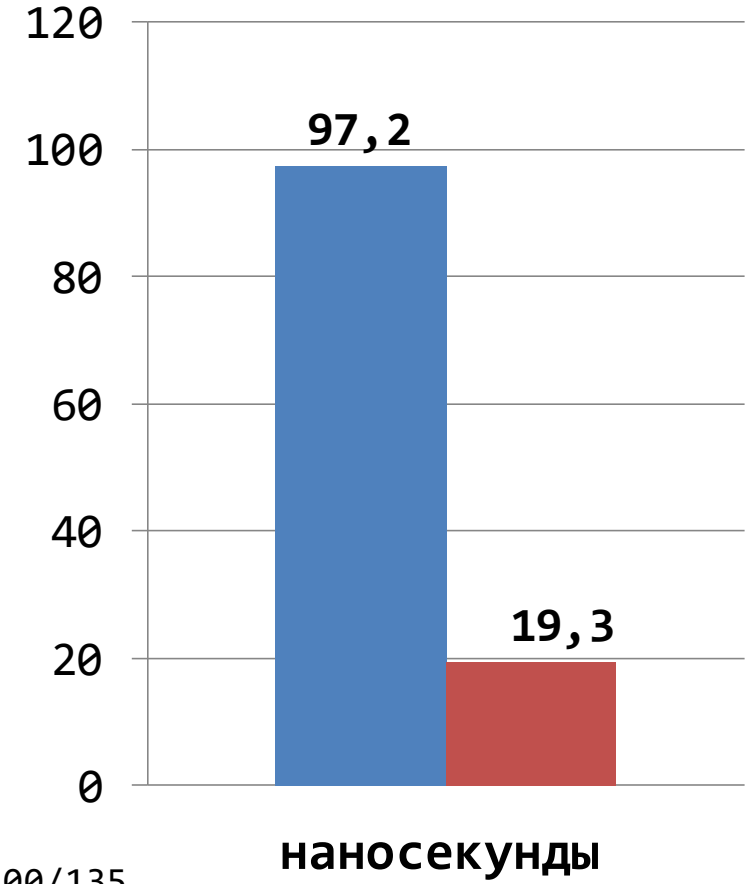
Отравление профиля

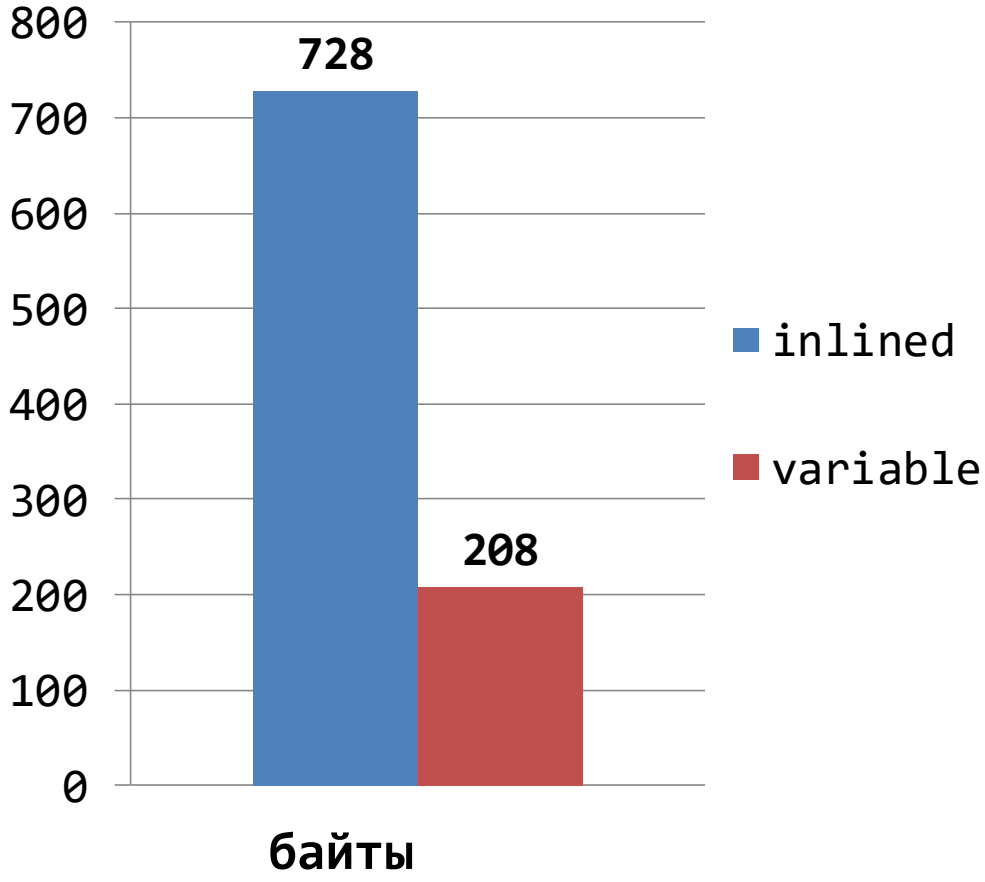
```
public String getName() {
  String name = this.name;
  if (name == null)
    this.name = name = getName0();
  return name;
}
```

https://stackoverflow.com/questions/59157085/java-8-class-getname-slows-down-string-concatenation-chain

https://github.com/spring-projects/spring-framework/pull/24153

После починки



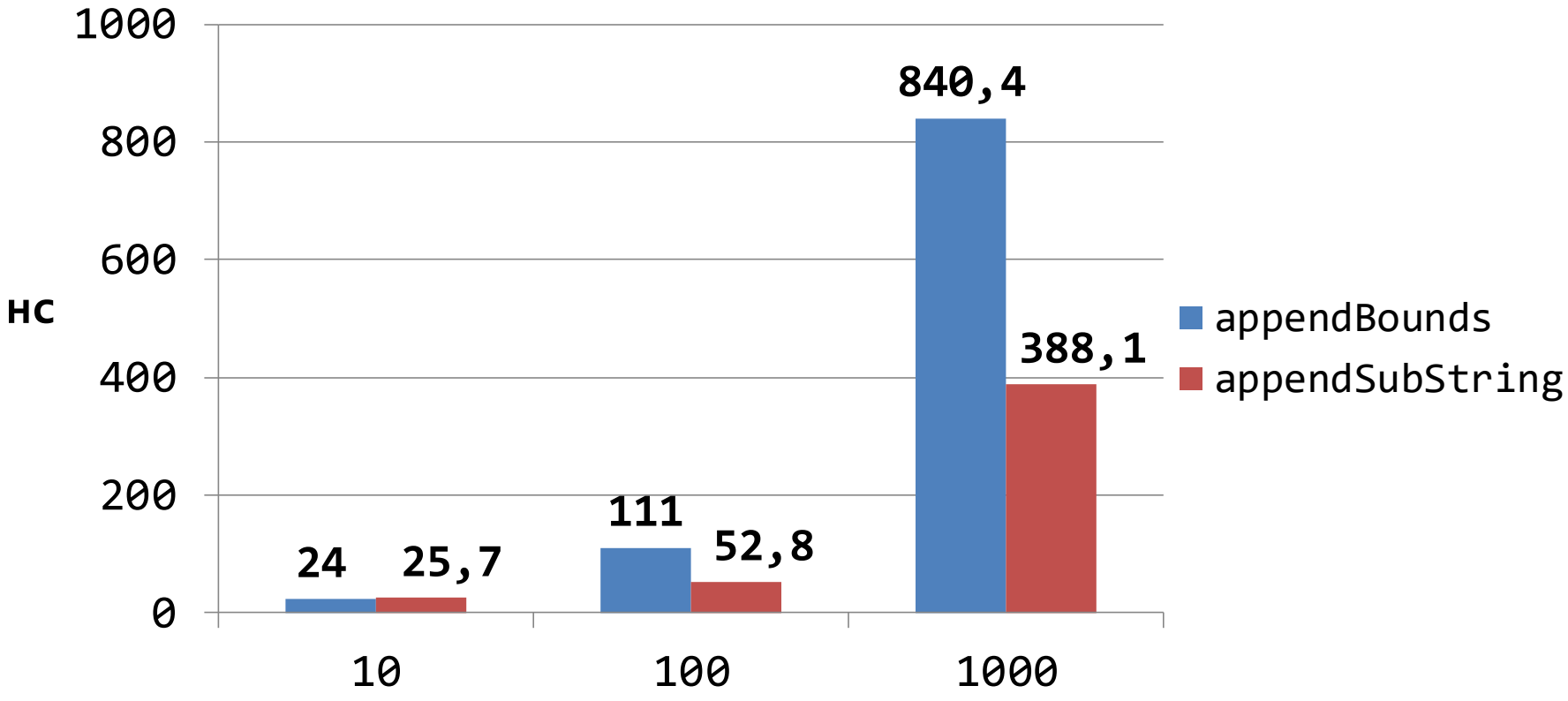


Ещё тонкости: что быстрее?

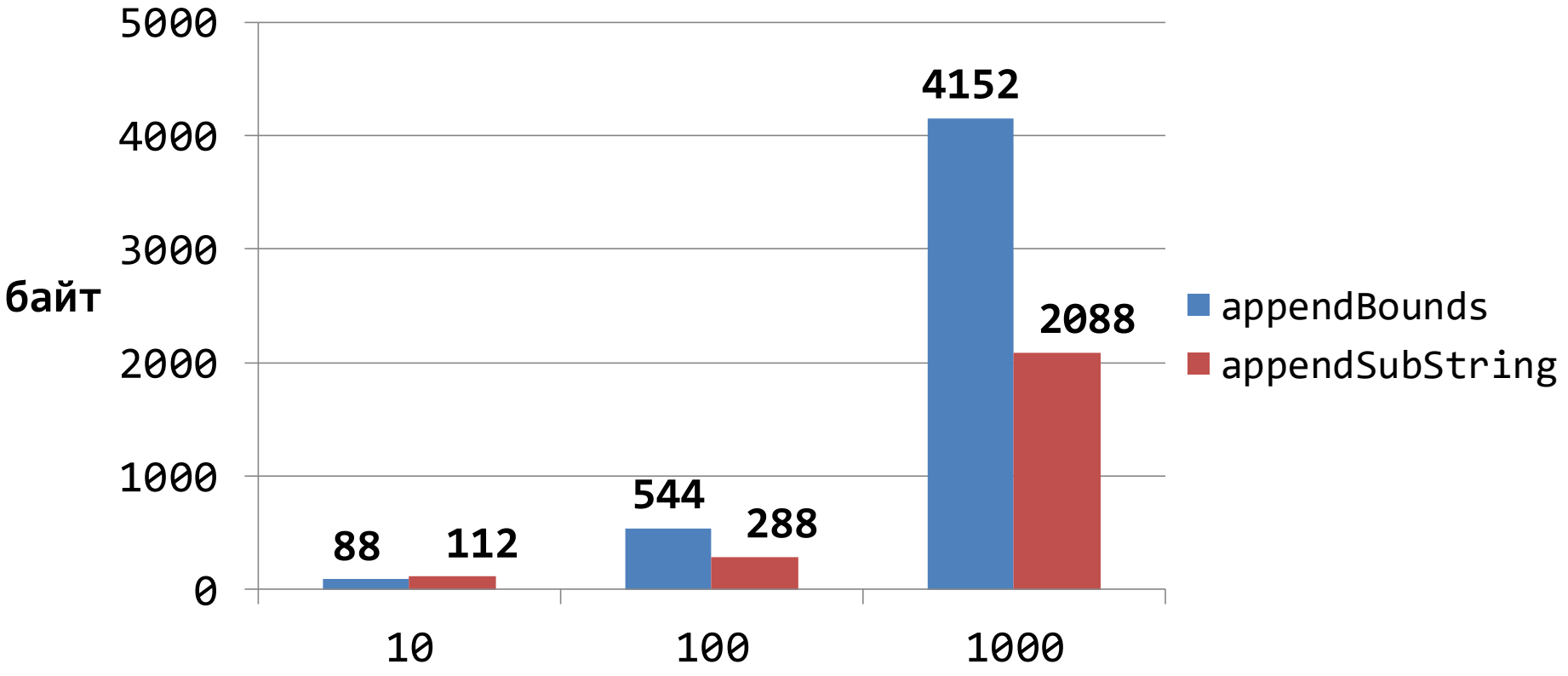
```
String s = new StringBuilder()
    .append('L')
    .append(str, from, to)
    .append(';')
    .toString();
```

```
String s = new StringBuilder()
    .append('L')
    .append(str.substring(from, to))
    .append(';')
    .toString();
```

Внезапно (время)



Внезапно (память)



Почему так?

```
sb.append(str.substring(from, to));     // интринсик
```

```
sb.append(str, from, to);               // цикл в ASB-е
```

https://habr.com/ru/post/436746/

Что с этим делать?

https://bugs.openjdk.java.net/browse/JDK-8217675

~~https://bugs.openjdk.java.net/browse/JDK-8224986~~ (JDK 13)

~~https://youtrack.jetbrains.com/issue/IDEA-205676~~ (2019.1)

Разработчик, помни!

Сшивание цепочки = упрощение кода + производительность

В старых изданиях JDK (<9) держи в уме угловые случаи

Склейка:

среди if-ов

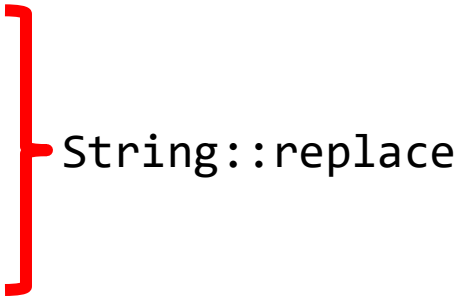
как среди рифов

На сцене ASM: выразить одним словом

```
void appendDescriptor(Class<?> clazz, StringBuilder sb) {
  //...
  String name = clazz.getName();
  int nameLength = name.length();
  for (int i = 0; i < nameLength; ++i) {
    char car = name.charAt(i);
    sb.append(car == '.' ? '/' : car);
  }

  //...
}
```

На сцене ASM: замена

```
void appendDescriptor(Class<?> clazz, StringBuilder sb) {
  //...
  String name = clazz.getName();
  int nameLength = name.length();
  for (int i = 0; i < nameLength; ++i) {
    char car = name.charAt(i);
    sb.append(car == '.' ? '/' : car);
  }

  //...
}
```



На сцене ASM: String::replace

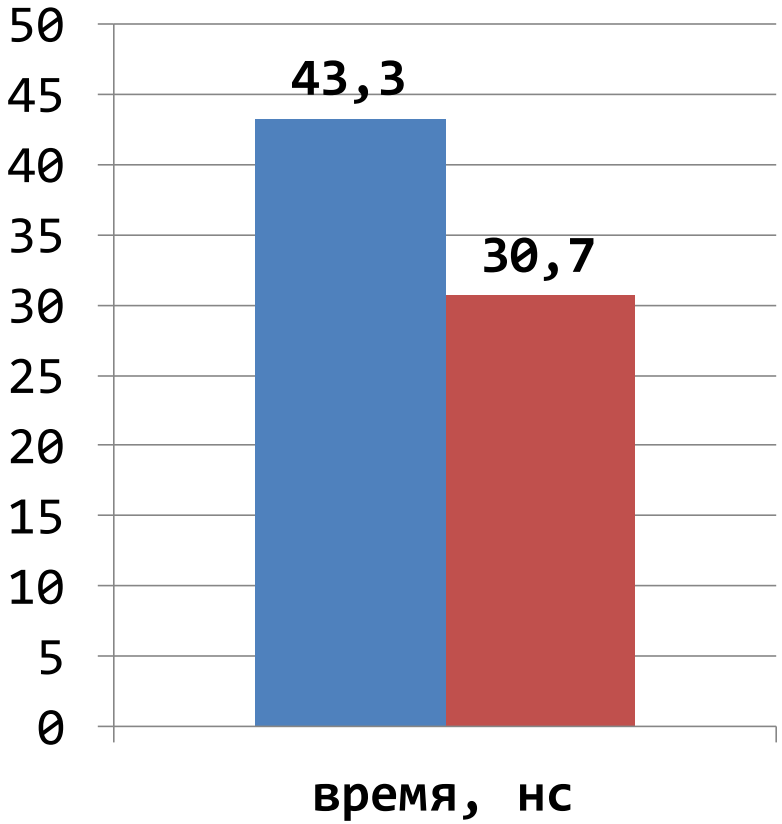
```
void appendDescriptor(Class<?> clazz, StringBuilder sb) {
  //...
  String name = clazz.getName();
  int nameLength = name.length();
  for (int i = 0; i < nameLength; ++i) {
    char car = name.charAt(i);
    sb.append(car == '.' ? '/' : car);
  }
  sb.append(clazz.getName().replace('.', '/'));
  //...
}
```

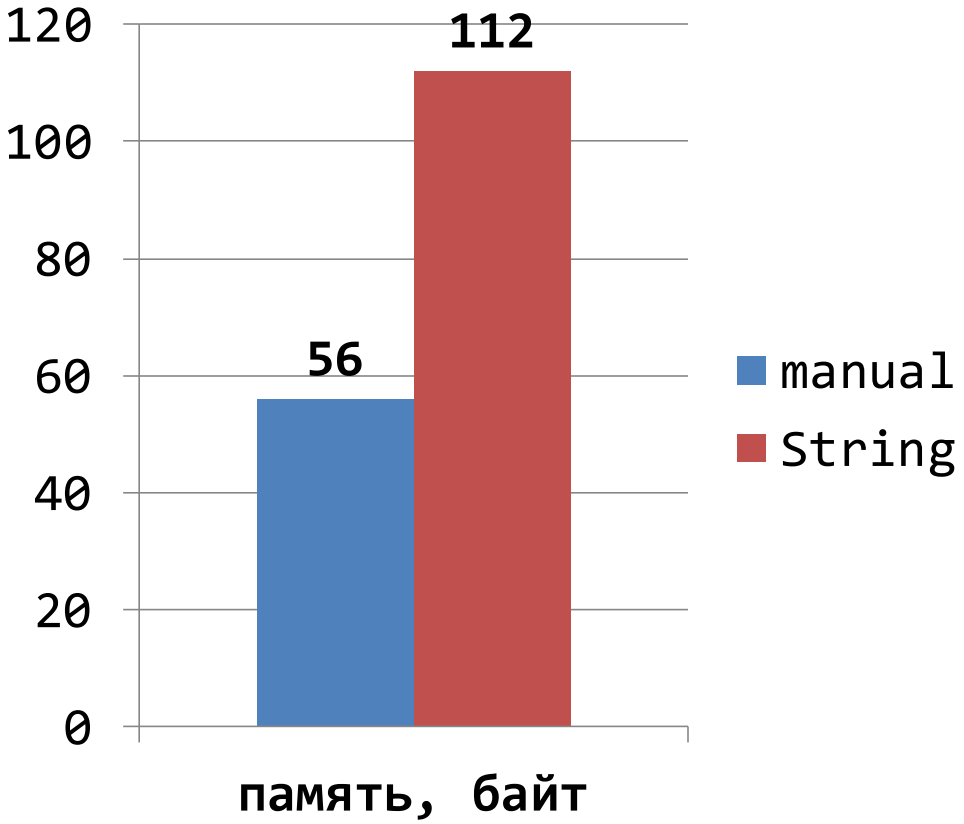
https://gitlab.ow2.org/asm/asm/merge_requests/235

Выиграем или нет, вот в чём вопрос

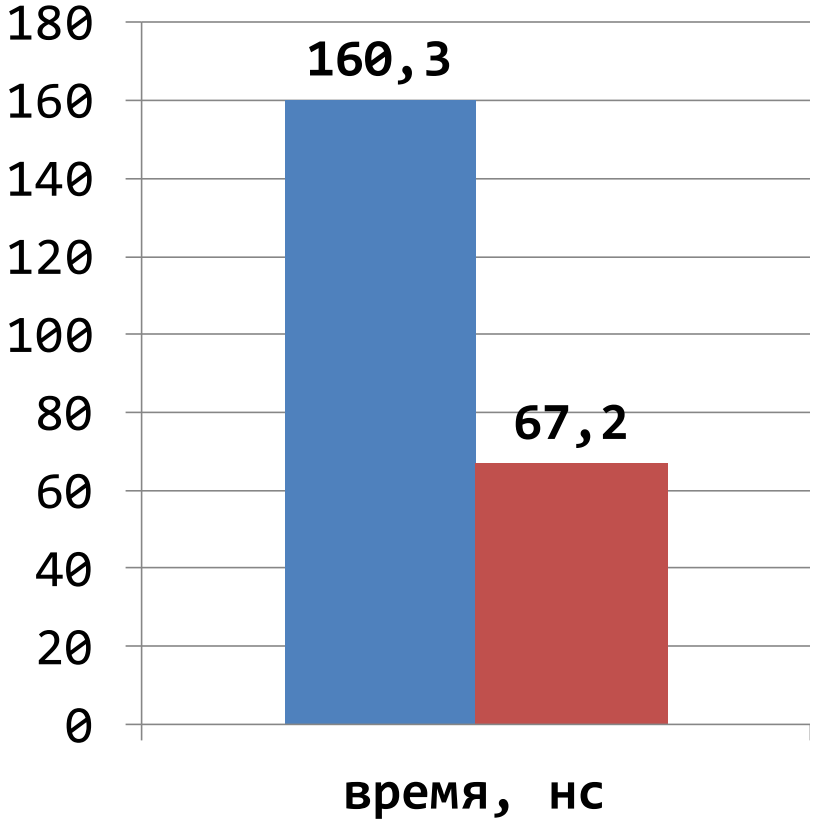
```
void appendDescriptor(Class<?> clazz, StringBuilder sb) {
  //...
  String name = clazz.getName();
  int nameLength = name.length();
  for (int i = 0; i < nameLength; ++i) {
    char car = name.charAt(i);
    sb.append(car == '.' ? '/' : car);
  }
  sb.append(clazz.getName().replace('.', '/'));
  //...
}
```

java.lang.String – [почти] получилось

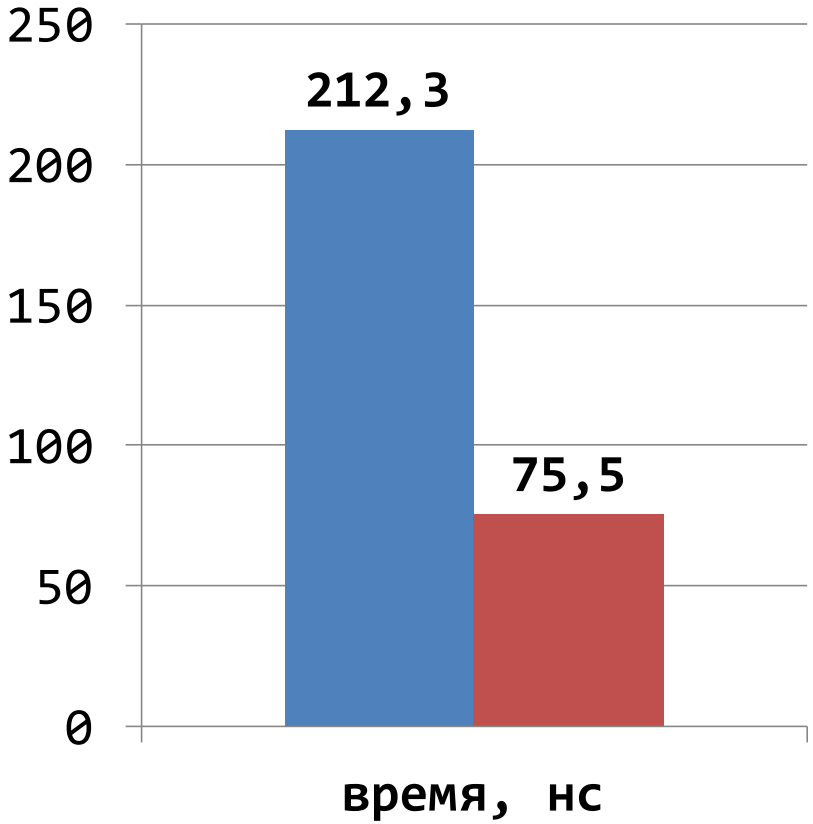


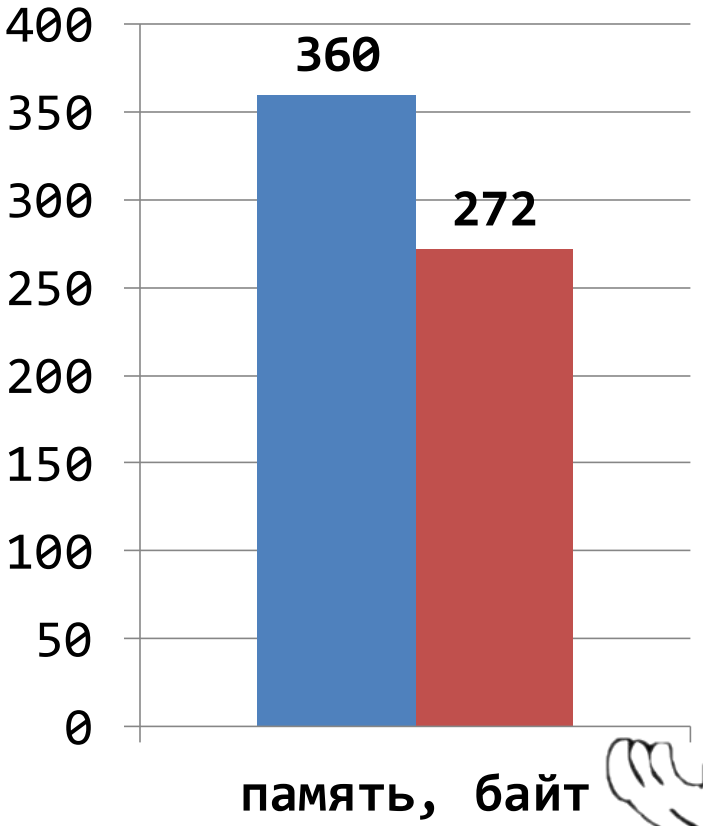


t.s.b.s.CharacterReplaceBenchmark.class



o.s.o.i.p.PercSerializationInstantiator.class





manual

String

По-одному: выделение памяти

```
for (int i = 0; i < nameLength; ++i) {
  char car = name.charAt(i);
  sb.append(car == '.' ? '/' : car);
}
```

java.lang.String	16 знаков – 16 ячеек
t.s.b.s.CharacterReplaceBenchmark	58 знаков – 70 ячеек
o.s.o.i.p.PercSerializationInstantiator	77 знаков – 142 ячейки

Всё сразу: выделение памяти

```
sb.append(clazz.getName().replace('.', '/'));
```

java.lang.String	16 знаков – 16 ячеек
t.s.b.s.CharacterReplaceBenchmark	58 знаков – 58 ячеек
o.s.o.i.p.PercSerializationInstantiator	77 знаков – 77 ячейки

Когда же это всё-таки нужно?

```
for (int i = 0; i < nameLength; ++i) {
  char car = name.charAt(i);
  sb.append(car == '.' ? '/' : car);
}
```

Когда же это всё-таки нужно: когда замен много

```
char c = text.charAt(i);
switch (c) {
  case GROUP_SEPARATOR:
  case GROUPS_SEPARATOR:
  case GROUP_VALUE_SEPARATOR:
  case '\'':
  case '\"':
  case '=' :
    escaped.append(' ');    break;
  default:
    escaped.append(c);      break;
}
```

com.intellij.internal.statistic.beans.ConvertUsagesUtil

Когда же это всё-таки нужно: иначе накладно

```
return text
  .replace(GROUP_SEPARATOR, ' ')
  .replace(GROUPS_SEPARATOR, ' ')
  .replace(GROUP_VALUE_SEPARATOR, ' ')
  .replace('\'', ' ')
  .replace('\"', ' ')
  .replace('=' , ' ');
```

Итого:

- 6 новых строк (в худшем случае)
- 6 полных переборов (в любом случае)

Когда же это всё-таки нужно: не так уж и редко

c.i.internal.statistic.beans.ConvertUsagesUtil::ensureProperKey

c.i.openapi.projectRoots.JdkUtil::quoteArg

com.maddyhome.idea.copyright.pattern.EntityUtil::encode

org.jetbrains.git4idea.ssh.SSHConfig::compilePattern

Разработчик, помни!

- разовое выделение памяти лучше, чем последовательное
- неразрывное действие лучше, чем цикл
- массовые операции в 99/100 выигрывают у одиночных
- из любого правила бывают исключения в 1/100

StringJoiner:

склеивание с

с разделителем

StringJoiner: JDK 8 --> JDK 9

```
public final class StringJoiner {
  private final String prefix;
  private final String delimiter;
  private final String suffix;

  private StringBuilder value;
}
```

```
public final class StringJoiner {
  private final String prefix;
  private final String delimiter;
  private final String suffix;

  private String[] elts;

  private int size;
  private int len;
}
```

https://bugs.openjdk.java.net/browse/JDK-8054221

Главное во всём этом – StringJoiner.toString()

```
char[] chars = new char[len + addLen];
int k = getChars(prefix, chars, 0);
if (size > 0) {
    k += getChars(elts[0], chars, k);
    for (int i = 1; i < size; i++) {
        k += getChars(delimiter, chars, k);
        k += getChars(elts[i], chars, k);
    }
}
k += getChars(suffix, chars, k);
return new String(chars);
```

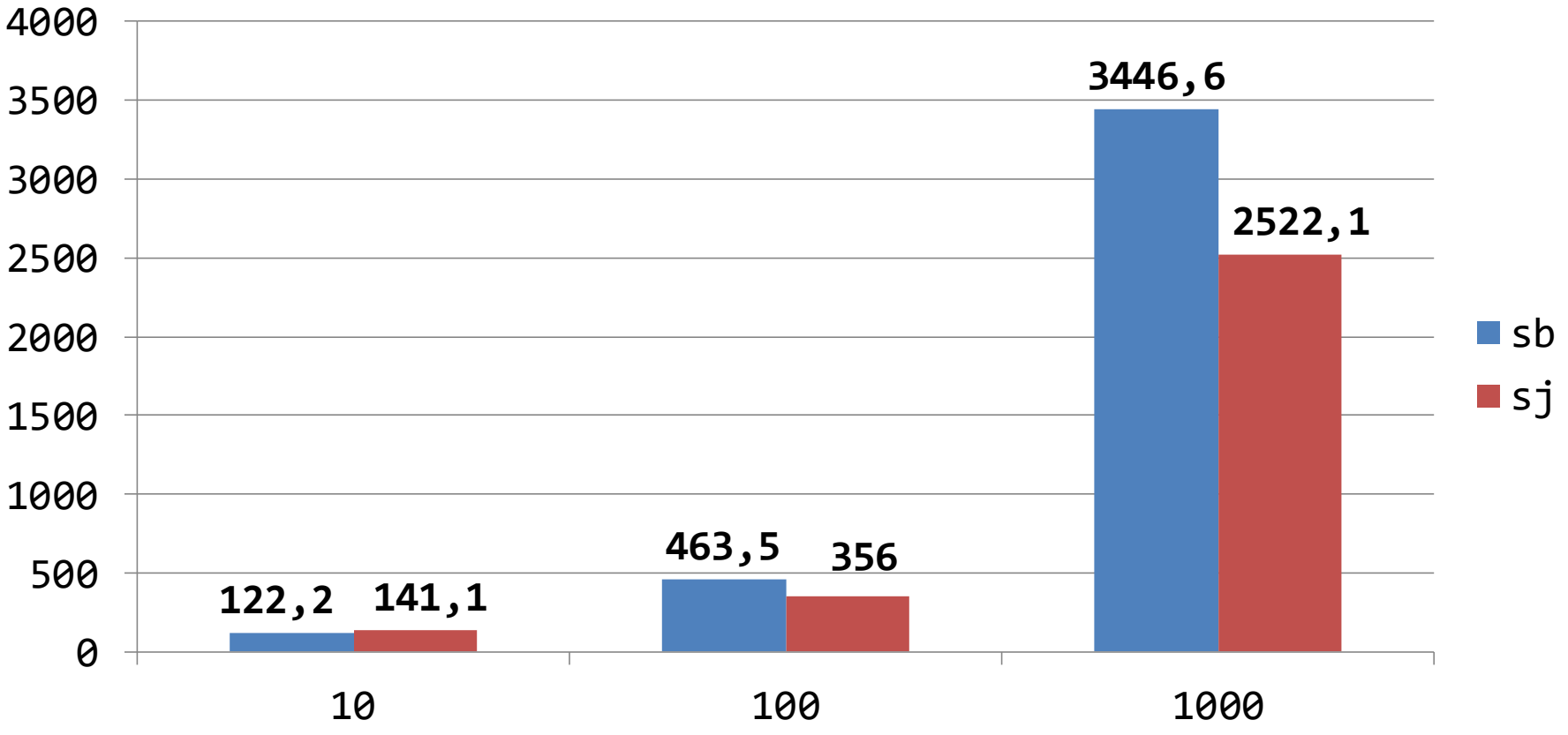
Нужно больше производительности!

```
StringBuilder pathBuilder = new StringBuilder();

for (PathComponent pathComponent : this.pathComponents) {
  pathBuilder.append(pathComponent.getPath());
}

return pathBuilder.toString();
```

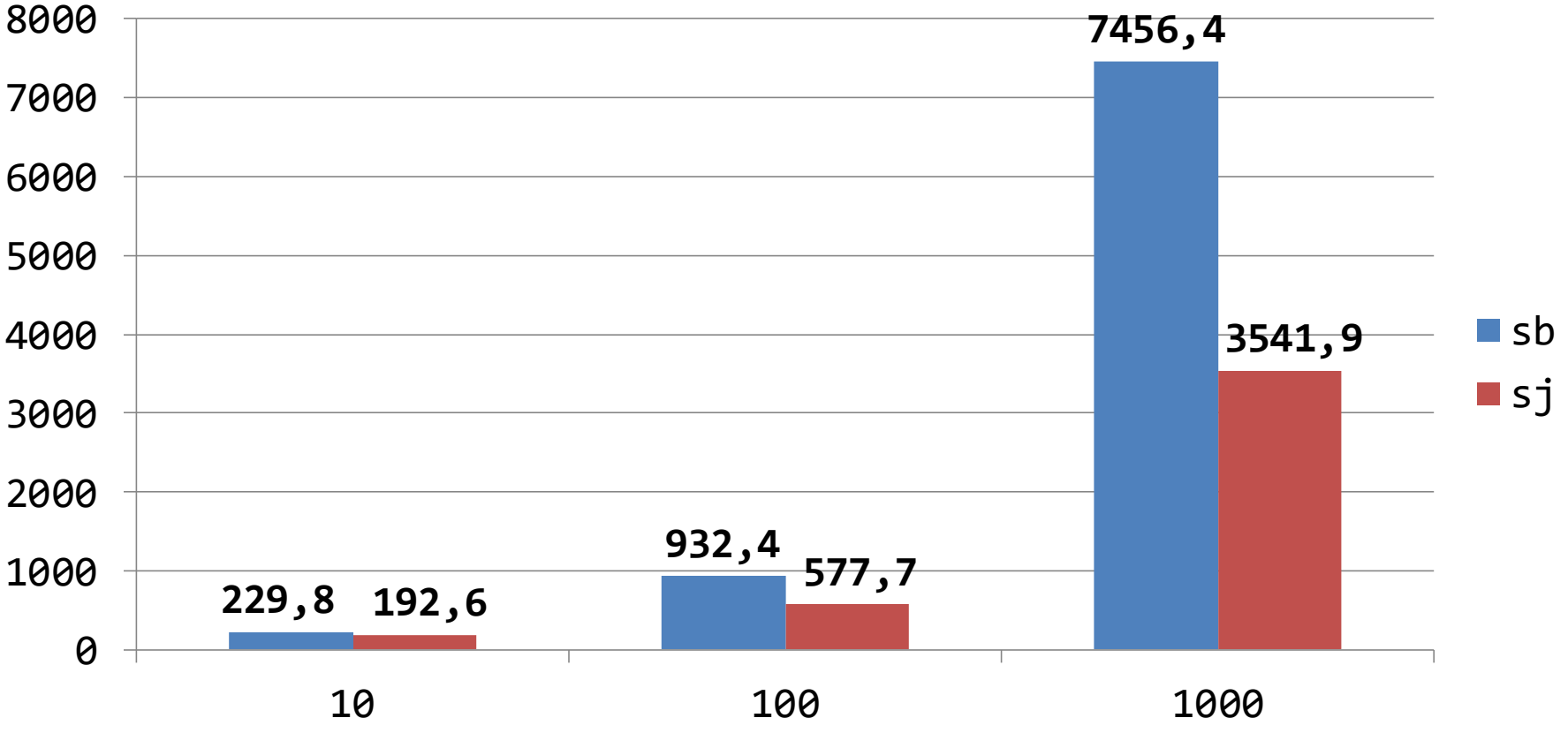
Нужно больше производительности!

```
StringJoiner  pathBuilder = new StringJoiner("");

for (PathComponent pathComponent : this.pathComponents) {
  pathBuilder.add(pathComponent.getPath());
}

return pathBuilder.toString();
```

Стоит ли игра свеч (10 латинских строк)?



Стоит ли игра свеч (10 кириллических строк)?



SJ.toString(), вопрос для любознательных:

```
char[] chars = new char[len + addLen];
int k = getChars(prefix, chars, 0);
if (size > 0) {
    k += getChars(elts[0], chars, k);
    for (int i = 1; i < size; i++) {
        k += getChars(delimiter, chars, k);
        k += getChars(elts[i], chars, k);
    }
}
k += getChars(suffix, chars, k);
return new String(chars);
```

SJ.toString(), вопрос для любознательных:

```
char[] chars = new char[len + addLen];    // почему char[] ?!!
int k = getChars(prefix, chars, 0);
if (size > 0) {
    k += getChars(elts[0], chars, k);
    for (int i = 1; i < size; i++) {
        k += getChars(delimiter, chars, k);
        k += getChars(elts[i], chars, k);
    }
}
k += getChars(suffix, chars, k);
return new String(chars);
```

Краткий ответ: из-за пакета

```
java.lang.String

java.util.StringJoiner
```

https://habr.com/ru/post/488874/

https://bugs.openjdk.java.net/browse/JDK-8148937

Выводы

- `map.get(/* new String */)` – избегайте
- составной ключ вида `"_" + smth` почти всегда можно заменить
- при буферизации обращайте внимание на объём данных и буфера
- клейте строки через «+», ~~скрипач~~ StringBuilder зачастую не нужен
- одиночные преобразования *почти* всегда проигрывают массовым
- помните о StringJoiner-е и используйте его для типовых задач

Благодарю за внимание

sergei.tsypanov@yandex.ru