Deutsche Bank
Deutsche Bank Technology Centre, LLC

Escape Analysis & Scalar Replacement

Черемин Руслан, Deutsche Bank

Passion to Perform

Общеизвестно что…

Escape Analysis:

статический анализ *времени жизни* (области достижимости) объекта

Scalar Replacement

Проецирование полей объекта в локальные переменные

Общеизвестно что...

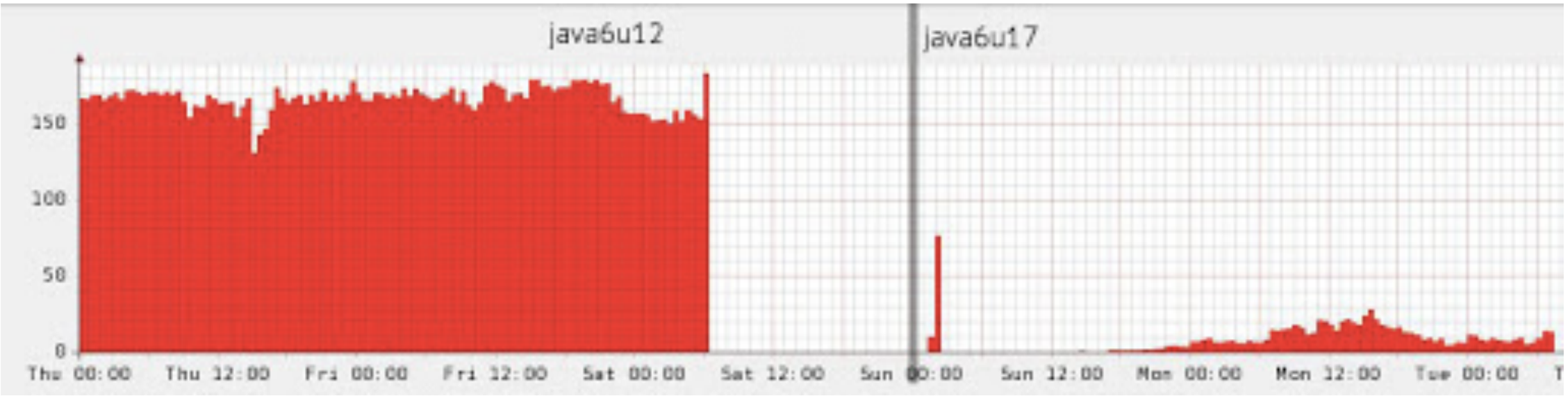
GC:

Reachability Analysis -> Memory Reclamation

JIT:

Escape Analysis -> Scalar Replacement

Результаты вдохновляют



dolzhenko.blogspot.ru/2011/01/lock-coarsening-biased-locking-escape.html

За 2009-2016 я узнал

1. `Point p = new Point(x,y)`
2. `ArrayList.iterator()`
3. `sum(int... numbers)`

Цели

- Ментальная модель (чуйка) EA/SR
  - Где надежно работает
  - Где надежно не работает
  - Где надо проверять
- Набор кейсов
  - Методика их тестирования
  - Фреймворк для тестирования

Эксперименты!
Эксперименты!!!

…Но сначала немного теории

EA/SR как часть C2

```
a = new A()
b = new B()
c = new C()
b.setField1(a)
setGlobal(c)
return b.f()
```

→

Inlining
De-virtualization
Constant folding
uncommon traps
…

→

```
a = calloc(A)
b = calloc(B)
c = calloc(C)
b.field1 = a
global.c = c
return b.field1
```

EA/SR как часть C2

```
a = new A()
b = new B()
c = new C()
b.setField1(a)
setGlobal(c)
return b.f()
```

→

```
Inlining
De-virtualization
Constant folding
uncommon traps
…
```

→

```
a = calloc(A)
b = calloc(B)
c = calloc(C)
b.field1 = a
global.c = c
return b.field1
```

↓

Escape Analysis

←

```
a = calloc(A)
b = calloc(B)   [*]
c = calloc(C)
b.field1 = a
global.c = c
return b.field1
```

EA/SR как часть C2

```
a = new A()
b = new B()
c = new C()
b.setField1(a)
setGlobal(c)
return b.f()
```

→

```
Inlining
De-virtualization
Constant folding
uncommon traps
…
```

→

```
a = calloc(A)
b = calloc(B)
c = calloc(C)
b.field1 = a
global.c = c
return b.field1
```

↓

Escape Analysis

←

```
a = calloc(A)
b = calloc(B)   [*]
c = calloc(C)
b.field1 = a
global.c = c
return b.field1
```

←

Scalar Replacement

↓

```
a = calloc(A)

c = calloc(C)
b$field1 = a
global.c = c
return b$field1
```

Все-таки эксперименты

Инструменты

-Измерения

- `ThreadMBean.`**`getThreadAllocatedBytes(threadId)`**
- `GarbageCollectorMXBean.`**`getCollectionCount`**`()`
- Контроль: `-XX:-EliminateAllocations`

-Раскопки и расследования

- `-XX:+PrintCompilation`
- `-XX:+PrintInlining`
- `-XX:+LogCompilation` **-> JITWatch**

Инструменты: JMH?

JMH Sample_35:

```
> java -jar target/benchmarks.jar JMHSample_35.*Maps -prof gc
```

Benchmark	(type)	Mode	Cnt	Score		Error	Units
JMHSample_35_Profilers.Maps.test	hashmap	avgt	5	1553.201	±	6.199	ns/op
JMHSample_35_Profilers.Maps.test:Â·gc.alloc.rate	**hashmap**	**avgt**	**5**	**1257.046**	**±**	**5.675**	**MB/sec**
JMHSample_35_Profilers.Maps.test:Â·gc.alloc.rate.norm	**hashmap**	**avgt**	**5**	**2048.001**	**±**	**0.001**	**B/op**
JMHSample_35_Profilers.Maps.test:Â·gc.churn.PS_Eden_Space	hashmap	avgt	5	1259.148	±	315.277	MB/sec
JMHSample_35_Profilers.Maps.test:Â·gc.churn.PS_Eden_Space.norm	hashmap	avgt	5	2051.519	±	520.324	B/op
JMHSample_35_Profilers.Maps.test:Â·gc.churn.PS_Survivor_Space	hashmap	avgt	5	0.175	±	0.386	MB/sec
JMHSample_35_Profilers.Maps.test:Â·gc.churn.PS_Survivor_Space.norm	hashmap	avgt	5	0.285	±	0.629	B/op
JMHSample_35_Profilers.Maps.test:Â·gc.count	**hashmap**	**avgt**	**5**	**29.000**			**counts**
JMHSample_35_Profilers.Maps.test:Â·gc.time	hashmap	avgt	5	16.000			ms

№1.1: Basic

```
long allocateOnce( final ThreadLocalRandom rnd ) {
   final Vector2D v1 = randomVector( rnd );
   final Vector2D v2 = randomVector( rnd );
   final Vector2D v3 = randomVector( rnd );
   return ( long ) v1.add( v2 ).dot( v3 );
}

...
public class Vector2D {
   private double x;
   private double y;

   Vector2D randomVector( final ThreadLocalRandom rnd ) {
      return new Vector2D(
            rnd.nextDouble(),
            rnd.nextDouble()
      );
   }...
}
```

№1.1: Basic

```
long allocateOnce( final ThreadLocalRandom rnd ) {
   final Vector2D v1 = randomVector( rnd );
   final Vector2D v2 = randomVector( rnd );
   final Vector2D v3 = randomVector( rnd );
   return ( long ) v1.add( v2 ).dot( v3 );
}
```

```
JDK: 1.8.0_77 (25.77-b03), OS: 'Mac OS X' 10.11.4 arch: x86_64

12 runs, 5000 ms each

#0: ~= 0.02 b/call (2610600 bytes/131257344 calls)
#1: ~= 0.00 b/call (48 bytes/180697088 calls)
#2: ~= 0.00 b/call (48 bytes/177738752 calls)
#3: ~= 0.00 b/call (48 bytes/184651776 calls)
#4: ~= 0.00 b/call (48 bytes/178926592 calls)
#5: ~= 0.00 b/call (48 bytes/179334144 calls)
#6: ~= 0.00 b/call (48 bytes/180411392 calls)
...
```

№1.1: Контроль

```
long allocateOnce( final ThreadLocalRandom rnd ) {
   final Vector2D v1 = randomVector( rnd );
   final Vector2D v2 = randomVector( rnd );
   final Vector2D v3 = randomVector( rnd );
   return ( long ) v1.add( v2 ).dot( v3 );
}
```

JDK: 1.8.0_77 (25.77-b03) **-XX:-EliminateAllocations**

12 runs, 5000 ms each

```
#0: ~= 128.00 b/call (1
#1: ~= 128.00 b/call (14763163696 bytes/115337216 calls)
#2: ~= 128.00 b/call (13795459120 bytes/107777024 calls)
#3: ~= 128.00 b/call (14887944240 bytes/116312064 calls)
#4: ~= 128.00 b/call (15298199600 bytes/119517184 calls)
...
```

4 * sizeof(Vector2D)

№1.2: Loop accumulate

```
public static final int SIZE = ...;

long accumulateInLoop( final ThreadLocalRandom rnd ) {
   final Vector2D v = randomVector( rnd );
   for( int i = 0; i < SIZE; i++ ) {
      final Vector2D addition = randomVector( rnd );
      v.addAccumulate( addition );
   }
   return ( long ) v.length();
}
```

SIZE = …
bytes/call = ?

№1.2: Loop accumulate

```
public static final int SIZE = ...;

long accumulateInLoop( final ThreadLocalRandom rnd ) {
   final Vector2D v = randomVector( rnd );
   for( int i = 0; i < SIZE; i++ ) {
      final Vector2D addition = randomVector( rnd );
      v.addAccumulate( addition );
   }
   return ( long ) v.length();
}
```

```
JDK: 1.8.0_77  (25.77-b03)

[SIZE=64]: 12 runs, 5000 ms each

#0: ~= 0.88 b/call (5228296 bytes/5957632 calls)
#1: ~= 0.00 b/call (48 bytes/8574976 calls)
#2: ~= 0.00 b/call (48 bytes/8552448 calls)
#3: ~= 0.00 b/call (48 bytes/8382464 calls)
#4: ~= 0.00 b/call (48 bytes/8547328 calls)
...
```

№1.3: Replace in loop

```
public static final int SIZE = ...;

long replaceInLoop( final ThreadLocalRandom rnd ) {
   Vector2D v = randomVector( rnd );
   for( int i = 0; i < SIZE; i++ ) {
      v = v.mul( i );
   }
   return ( long ) v.length();
}
```

SIZE = …
bytes/call = ?

№1.3: Replace in loop

```
public static final int SIZE = ...;

long replaceInLoop( final ThreadLocalRandom rnd ) {
   Vector2D v = randomVector( rnd );
   for( int i = 0; i < SIZE; i++ ) {
      v = v.mul( i );
   }
   return ( long ) v.length();
}
```

JDK: 1.8.0_77 (25.77-b03)

[SIZE:**64**]: 12 runs, 5000 ms each

```
#0: ~= 2080.01 b/call (14264129576 bytes/6857728 calls)
#1: ~= 2080.00 b/call (19495157808 bytes/9372672 calls)
#2: ~= 2080.00 b/call (19710279728 bytes/9476096 calls)
#3: ~= 2080.00 b/call (21256601648 bytes/10219520 calls)
#4: ~= 2080.00 b/call (21806121008 bytes/10483712 calls)
...
```

№1.3: Replace in loop

```
public static final int SIZE = ...;

long replaceInLoop( final ThreadLocalRandom rnd ) {
   Vector2D v = randomVector( rnd );
   for( int i = 0; i < SIZE; i++ ) {
      v = v.mul( i );
   }
   return ( long ) v.length();
}
```

JDK: 1.8.0_77 (25.77-b03)

[SIZE:**64**]: 12 runs, 5000 ms each

```
#0: ~= 2080.01 b/call (
#1: ~= 2080.00 b/call
#2: ~= 2080.00 b/call (
#3: ~= 2080.00 b/call (
#4: ~= 2080.00 b/call (21806121008 bytes/10483712 calls)
...
```

= 32 * (SIZE + 1)

№1.4: Control flow

```
long assignConditionally( final ThreadLocalRandom rnd ) {
   final Vector2D v;
   if( rnd.nextBoolean() ) {
      v = new Vector2D( 1, rnd.nextDouble() );
   } else {
      v = new Vector2D( rnd.nextDouble(), 1 );
   }
   return ( long ) v.length();
}
```

bytes/call = ?

№1.4: Control flow

```
long assignConditionally( final ThreadLocalRandom rnd ) {
   final Vector2D v;
   if( rnd.nextBoolean() ) {
      v = new Vector2D( 1, rnd.nextDouble() );
   } else {
      v = new Vector2D( rnd.nextDouble(), 1 );
   }
   return ( long ) v.length();
}
```

```
JDK: 1.8.0_77 (25.77-b03)

12 runs, 5000 ms each

#0: ~= 32.00 b/call (4206352424 bytes/131446784 calls)
#1: ~= 32.00 b/call (4456185904 bytes/139255808 calls)
#2: ~= 32.00 b/call (3771924528 bytes/117872640 calls)
#3: ~= 32.00 b/call (4445503536 bytes/138921984 calls)
#4: ~= 32.00 b/call (4250140720 bytes/132816896 calls)
...
```

№1.5: Control flow

```
long assignUnConditionally( final ThreadLocalRandom rnd ) {
   if( rnd.nextBoolean() ) {
      final Vector2D v = new Vector2D( 1, rnd.nextDouble() );
      return ( long ) v.length();
   } else {
      final Vector2D v = new Vector2D( rnd.nextDouble(), 1 );
      return ( long ) v.length();
   }
}
```

bytes/call = ?

№1.5: Control flow

```
long assignUnConditionally( final ThreadLocalRandom rnd ) {
   if( rnd.nextBoolean() ) {
      final Vector2D v = new Vector2D( 1, rnd.nextDouble() );
      return ( long ) v.length();
   } else {
      final Vector2D v = new Vector2D( rnd.nextDouble(), 1 );
      return ( long ) v.length();
   }
}
```

```
JDK: 1.8.0_77 (25.77-b03)

#0: ~= 0.00 b/call (805224 bytes/226723840 calls)
#1: ~= 0.00 b/call (48 bytes/268607488 calls)
#2: ~= 0.00 b/call (48 bytes/253534208 calls)
#3: ~= 0.00 b/call (48 bytes/271968256 calls)
#4: ~= 0.00 b/call (48 bytes/275472384 calls)
...
```

№1.*: “Merge points”

```
v1 = new V(x1, y1)
```

```
v2 = new V(x2, y2)
```

```
v3 = (…) ? (v1) : (v2)

return v3.x
```

№1.*: “Merge points”

```
v1$x = x1
v1$y = y1
```

```
v2$x = x2
v2$y = y2
```

```
v3 = (…) ? (v1) : (v2)

return v3.x
```

?

№1.*: “Merge points”

…The test case shows limitation of the current EA implementation. Objects will not be eliminated if there is merge point **in which it is undefined which object is referenced**…

bugs: JDK-6853701

…Or the address may **point to more then one object**. This may produce the false positive result (set not scalar replaceable) since the flow-insensitive escape analysis can't separate the case when stores overwrite the field's value from the case when stores happened on different control branches.

/src/share/vm/opto/escape.cpp#l1732

№1.*: Интуитивно

Скаляризации не будет, если

- Ссылочная переменная может указывать *более, чем на один объект*
- Даже если такое может случиться только в разных сценариях исполнения

№2: EqualsBuilder

```
import org.apache.commons.lang3.builder.EqualsBuilder;

...

public boolean equals( final Object o ) {
    ...
   return new EqualsBuilder()
        .append( integer, other.integer )
        .append( string, other.string )
        .append( date, other.date )
        .isEquals();
}
```

№2.1: EqualsBuilder

```
long equalsWithBuilder() {
    final EqualsBuilder builder = new EqualsBuilder();
    return builder.append( 123, 456 )
                  .isEquals() ? 1 : 0;
}
```

bytes/call = ?

№2.1: EqualsBuilder

```
long equalsWithBuilder() {
   final EqualsBuilder builder = new EqualsBuilder();
   return builder.append( 123, 456 )
                 .isEquals() ? 1 : 0;
}
```

```
JDK: 1.8.0_77 (25.73-b03)

EqualsBuilderScenario: 12 runs, 5000 ms each

#0: ~= 0.00 b/call (352344 bytes/5311315968 calls)
#1: ~= 0.00 b/call (48 bytes/7383114752 calls)
#2: ~= 0.00 b/call (48 bytes/98100893696 calls)
#3: ~= 0.00 b/call (48 bytes/98947292160 calls)
#4: ~= 0.00 b/call (48 bytes/98130951168 calls)
...
```

№2.2: EqualsBuilder

```
long equalsWithBuilder() {
    final EqualsBuilder builder = new EqualsBuilder();
    return builder.append( "123","456" )
                  .isEquals() ? 1 : 0;
}
```

bytes/call = ?

№2.2: EqualsBuilder

```
long equalsWithBuilder() {
   final EqualsBuilder builder = new EqualsBuilder();
   return builder.append( "123","456" )
                 .isEquals() ? 1 : 0;
}
```

JDK: 1.8.0_77 (25.77-b03)

12 runs, 5000 ms each

```
#0: ~= 16.00 b/call (3935910552 bytes/245989376 calls)
#1: ~= 16.00 b/call (4909482032 bytes/306842624 calls)
#2: ~= 16.00 b/call (5425479728 bytes/339092480 calls)
#3: ~= 16.00 b/call (4982505520 bytes/311406592 calls)
#4: ~= 16.00 b/call (5179621424 bytes/323726336 calls)
```

№2.2: EqualsBuilder

```
long equalsWithBuilder() {
   final EqualsBuilder builder = new EqualsBuilder();
   return builder.append( "123","456" )
                 .isEquals() ? 1 : 0;
}
```

JDK: 1.8.0_77 (25.77-b03)

-XX:+PrintInlining -XX:+PrintCompilation

```
...
 @ 7   ...EqualsBuilderScenario::run (28 bytes)  inline (hot)
   @ 4   o.a.c.l.b.EqualsBuilder::<init> (10 bytes)   inline (hot)
   @ 4   o.a.c.l.b.EqualsBuilder::<init> (10 bytes)   inline (hot)
     @ 1   java.lang.Object::<init> (1 bytes)   inline (hot)
     @ 13   o.a.c.l.b.EqualsBuilder::append (327 bytes)  hot method too big
     @ 16   o.a.c.l.b.EqualsBuilder::isEquals (5 bytes)  accessor
...
```

№2.2: Большой-горячий

“hot method too big”:

- Метод горячий
- Размер его *байткода* больше, чем `-XX:FreqInlineSize` (=325)

№2.2: Большой...

```java
public EqualsBuilder append(Object lhs, Object rhs) {
    if (!isEquals) {
        return this;
    }
    if (lhs == rhs) {
        return this;
    }
    if (lhs == null || rhs == null) {
        this.setEquals(false);
        return this;
    }
    Class<?> lhsClass = lhs.getClass();
    if (!lhsClass.isArray()) {
        // The simple case, not an array, just test the element
        isEquals = lhs.equals(rhs);
    } else if (lhs.getClass() != rhs.getClass()) {
        // Here when we compare different dimensions, for example: a boolean[][] to a boolean[]
        this.setEquals(false);
    }
    // 'Switch' on type of array, to dispatch to the correct handler
    // This handles multi dimensional arrays of the same depth
    else if (lhs instanceof long[]) {
        append((long[]) lhs, (long[]) rhs);
    } else if (lhs instanceof int[]) {
        append((int[]) lhs, (int[]) rhs);
    } else if (lhs instanceof short[]) {
        append((short[]) lhs, (short[]) rhs);
    } else if (lhs instanceof char[]) {
        append((char[]) lhs, (char[]) rhs);
    } else if (lhs instanceof byte[]) {
        append((byte[]) lhs, (byte[]) rhs);
    } else if (lhs instanceof double[]) {
        append((double[]) lhs, (double[]) rhs);
    } else if (lhs instanceof float[]) {
        append((float[]) lhs, (float[]) rhs);
    } else if (lhs instanceof boolean[]) {
        append((boolean[]) lhs, (boolean[]) rhs);
    } else {
        // Not an array of primitives
        append((Object[]) lhs, (Object[]) rhs);
    }
    return this;
}
```

№2.2: Большой...

```
public EqualsBuilder append(Object lhs, Object rhs) {
    if (!isEquals) {
        return this;
    }
    if (lhs == rhs) {
        return this;
    }
    if (lhs == null || rhs == null) {
        this.setEquals(false);
        return this;
    }
    Class<?> lhsClass = lhs.getClass();
    if (!lhsClass.isArray()) {
        // The simple case, not an array, just test the element
        isEquals = lhs.equals(rhs);
    } else if (lhs.getClass() != rhs.getClass()) {
        // Here when we compare different dimensions, for example: a boolean[][] to a boolean[]
        t
    }
    // 'Su
    // Th
    else i
        ap
    } else
        ap
    } else
        ap
    } else
        ap
    } else
        ap
    } else
        ap
    } else
        ap
    } else
        ap
    } else
        appe
    } else {
        // Not an array of primitives
        append((Object[]) lhs, (Object[]) rhs);
    }
    return this;
}
```

“Плюнь тому в глаза, кто скажет, что можно обнять необъятное!”

Козьма Прутков

№2.2: EqualsBuilder

```
long equalsWithBuilder() {
   final EqualsBuilder builder = new EqualsBuilder();
   return builder.append( "123","456" )
                 .isEquals() ? 1 : 0;
}
```

JDK: 1.8.0_77 (25.77-b03)

-XX:+PrintInlining -XX:+PrintCompilation -XX:FreqInlineSize=328

```
...
 @ 7   ...EqualsBuilderScenario::run (28 bytes)  inline (hot)
   @ 4   o.a.c.l.b.EqualsBuilder::<init> (10 bytes)   inline (hot)
     @ 1   java.lang.Object::<init> (1 bytes)   inline (hot)
   @ 13   o.a.c.l.b.EqualsBuilder::append (327 bytes)   inline (hot)
     @ 32   java.lang.Object::getClass (0 bytes)   (intrinsic)
     @ 37   java.lang.Class::isArray (0 bytes)   (intrinsic)
     @ 46   java.lang.String::equals (81 bytes)   (intrinsic)
     @ 16   o.a.c.l.b.EqualsBuilder::isEquals (5 bytes)   accessor
...
```

№2.2: EqualsBuilder

```
long equalsWithBuilder() {
   final EqualsBuilder builder = new EqualsBuilder();
   return builder.append( "123","456" )
                 .isEquals() ? 1 : 0;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

12 runs, 5000 ms each -XX:FreqInlineSize=328

#0: ~= 0.00 b/call (103840 bytes/1757181952 calls)
#1: ~= 0.00 b/call (48 bytes/2064528384 calls)
#2: ~= 0.00 b/call (48 bytes/65456977920 calls)
#3: ~= 0.00 b/call (48 bytes/65208904704 calls)
#4: ~= 0.00 b/call (48 bytes/66550593536 calls)
...
```

№2: Мораль

- Инлайнинг — лучший друг адаптивных рантаймов
- А краткость — его подруга

№3: Multi-values return

```
long multiValueReturn() {
   final Pair<String, String> tuple = tuple( pool );
   return tuple.getLeft().length()
          + tuple.getRight().length();
}

Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   return ImmutablePair.of( value1, value2 );
}
```

№3: Multi-values return

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   return ImmutablePair.of( value1, value2 );
}
```

```
...
JDK: 1.8.0_77 (25.77-b03)

IMMUTABLE_PAIR: 12 runs, 5000 ms each

#0: ~= 0.00 b/call (734008 bytes/211959808 calls)
#1: ~= 0.00 b/call (48 bytes/271724544 calls)
#2: ~= 0.00 b/call (48 bytes/259143680 calls)
#3: ~= 0.00 b/call (48 bytes/252703744 calls)
#4: ~= 0.00 b/call (48 bytes/265946112 calls)
...
```

№3.1: value or null

```
long multiValueReturn() {
   final Pair<String, String> tuple = tuple( pool );
   if( tuple != null ) {
      return tuple.getLeft().length()
           + tuple.getRight().length();
   } else {
      return 0;
   }
}

Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return ImmutablePair.of( value1, value2 );
   } else {
      return null;
   }
}
```

bytes/call = ?

№3.1: value or null

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return ImmutablePair.of( value1, value2 );
   } else {
      return null;
   }
}
```

JDK: 1.8.0_77 (25.77-b03)

```
#0: ~= 0.14 b/call (35882008 bytes/255149056 calls)
#1: ~= 0.14 b/call (40698192 bytes/289409024 calls)
#2: ~= 0.14 b/call (40074096 bytes/284971008 calls)
#3: ~= 0.14 b/call (39294912 bytes/279430144 calls)
#4: ~= 0.14 b/call (39939456 bytes/284013568 calls)
```

№3.2: Mixed types?

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return MutablePair.of( value1, value2 );
   } else {
      return ImmutablePair.of( value1, value2 );
   }
}
```

bytes/call = ?

№3.2: Mixed types?

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return MutablePair.of( value1, value2 );
   } else {
      return ImmutablePair.of( value1, value2 );
   }
}
```

```
JDK: 1.8.0_77 (25.77-b03)

#0: ~= 24.00 b/call (5870765704 bytes/244615168 calls)
#1: ~= 24.00 b/call (6249136176 bytes/260380672 calls)
#2: ~= 24.00 b/call (6540607536 bytes/272525312 calls)
#3: ~= 24.00 b/call (6369288240 bytes/265387008 calls)
#4: ~= 24.00 b/call (6376931376 bytes/265705472 calls)
```

№3.2: Mixed types??

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return ImmutablePair.of( value1, value2 );
   } else {
      return ImmutablePair.of( value1, value2 );
   }
}
```

bytes/call = ?

№3.2: Mixed types???

```
Pair<String, String> tuple( final StringsPool pool ) {
   final String value1 = pool.next();
   final String value2 = pool.next();
   if( value1.length() < value2.length() ) {
      return ImmutablePair.of( value1, value2 );
   } else {
      return ImmutablePair.of( value1, value2 );
   }
}
```

```
JDK: 1.8.0_77 (25.77-b03)

#0: ~= 24.00 b/call (2626585544 bytes/109441024 calls) 638263472
#1: ~= 24.00 b/call (3921592368 bytes/163399680 calls) 952952040
#2: ~= 24.00 b/call (3933438000 bytes/163893248 calls) 955830544
#3: ~= 24.00 b/call (3904241712 bytes/162676736 calls) 948735808
#4: ~= 24.00 b/call (3928522800 bytes/163688448 calls) 954636144
```

№3.2: Merge points!

```
long multiValueReturn_Inlined() {
   final Pair<String, String> tuple;

   final String value1 = pool.next();
   final String value2 = pool.next();

   if( value1.length() < value2.length() ) {
      tuple = ImmutablePair.of( value1, value2 );
   } else {
      tuple = ImmutablePair.of( value1, value2 );
   }

   return tuple.left.length()
          + tuple.right.length();
}
```

№4: Iterators

```
private Iterable<Integer> iterable = ...;

public long run() {
   return iterate( iterable );
}

long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += i;
   }
   return sum;
}
```

№4.1: ArrayList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

ARRAY_LIST[SIZE:16]: 12 runs, 5000 ms each

#0: ~= 0.01 b/call (1447392 bytes/165978112 calls)
#1: ~= 0.00 b/call (48 bytes/185303040 calls)
#2: ~= 0.00 b/call (48 bytes/141979648 calls)
#3: ~= 0.00 b/call (48 bytes/182721536 calls)
#4: ~= 0.00 b/call (48 bytes/192190464 calls)
...
```

№4.2: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

LINKED_LIST[SIZE:16]: 12 runs, 5000 ms each

#0: ~= 0.00 b/call (390656 bytes/151512064 calls)
#1: ~= 0.00 b/call (48 bytes/162306048 calls)
#2: ~= 0.00 b/call (48 bytes/159425536 calls)
#3: ~= 0.00 b/call (48 bytes/161737728 calls)
#4: ~= 0.00 b/call (48 bytes/161583104 calls)
...
```

№4: *.iterator

	1.8.0_77	1.7.0_80
ArrayList	+	+
LinkedList	+	+/−
Arrays.asList	+	+
HashSet	+	+/−
THashSet	+	+/−

№4.3: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.7.0_80 (24.80-b11)

LINKED_LIST[SIZE:2]: 12 runs, 5000 ms each

#0: ~= 0.00 b/call (173368 bytes/222623744 calls)
#1: ~= 0.00 b/call (48 bytes/162306048 calls)
#2: ~= 0.00 b/call (48 bytes/159425536 calls)
#3: ~= 0.00 b/call (48 bytes/161737728 calls)
#4: ~= 0.00 b/call (48 bytes/161583104 calls)

...
```

№4.3: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

JDK: **1.7.0_80** (24.80-b11)

LINKED_LIST[SIZE:65]: 12 runs, 5000 ms each

bytes/call = ?

№4.3: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.7.0_80 (24.80-b11)

LINKED_LIST[SIZE:65]: 12 runs, 5000 ms each

#0: ~= 31.91 b/call (334821496 bytes/10492928 calls)
#1: ~= 32.00 b/call (536805424 bytes/16776192 calls)
#2: ~= 32.00 b/call (481230896 bytes/15038464 calls)
#3: ~= 31.99 b/call (539885616 bytes/16875520 calls)
#4: ~= 32.00 b/call (550010928 bytes/17188864 calls)

...
```

№4.3: LinkedList.iterator

```
JDK: 1.7.0_80 (25.77-b03), LINKED_LIST[SIZE:2] -XX:+PrintInlining

...

@ 5  ...::iterate (46 bytes)   inline (hot)
  @ 3  j.u.AbstractSequentialList::iterator (5 bytes) inline (hot)
   \-> TypeProfile (4243/4243 counts) = java/util/LinkedList
    @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
      @ 2 j.u.LinkedList::listIterator (15 bytes)  inline (hot)
        @ 2 j.u.LinkedList::checkPositionIndex (22 bytes) inline (hot)
          @ 2 j.u.LinkedList::isPositionIndex (18 bytes) inline (hot)
        @ 11 j.u.LinkedList$ListItr::<init> (52 bytes) inline (hot)
          @ 6   j.l.Object::<init> (1 bytes) inline (hot)
          @ 40  j.u.LinkedList::node (65 bytes) inline (hot)
  @ 12   j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
   \-> TypeProfile (12730/12730 counts) = java/util/LinkedList$ListItr
  @ 22   j.u.LinkedList$ListItr::next (56 bytes) inline (hot)
    @ 1   j.u.LinkedList$ListItr::checkForComodification (23 bytes) inline (hot)
    @ 5   j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
  @ 35   j.l.Integer::intValue (5 bytes) accessor
  @ 12   j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)

...
```

№4.3: LinkedList.iterator

```
JDK: 1.7.0_80 (25.77-b03), LINKED_LIST[SIZE:65] -XX:+PrintInlining

...

@ 5  ...::iterate (46 bytes)   inline (hot)
  @ 3  j.u.AbstractSequentialList::iterator (5 bytes)   inline (hot)
   \-> TypeProfile (419/419 counts) = java/util/LinkedList
    @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
      @ 2 j.u.LinkedList::listIterator (15 bytes) already compiled into a medium
method
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes)  inline (hot)
   \-> TypeProfile (27655/27655 counts) = java/util/LinkedList$ListItr
  @ 22  j.u.LinkedList$ListItr::next (56 bytes) inline (hot)
    @ 1  j.u.LinkedList$ListItr::checkForComodification (23 bytes) inline (hot)
    @ 5  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
  @ 35  j.l.Integer::intValue (5 bytes) accessor
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)

...
```

№4.3: LinkedList.iterator

```
JDK: 1.7.0_80 (25.77-b03), LINKED_LIST[SIZE:65] -XX:+PrintInlining

...

@ 3 j.u.AbstractSequentialList::iterator (5 bytes) inline (hot)
 \-> TypeProfile (104/104 counts) = java/util/LinkedList
  @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
    @ 2 j.u.LinkedList::listIterator (15 bytes) executed < MinInliningThreshold
times
@ 12 j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
 \-> TypeProfile (6846/6846 counts) = java/util/LinkedList$ListItr

...

@ 5  ...::iterate (46 bytes)   inline (hot)
  @ 3  j.u.AbstractSequentialList::iterator (5 bytes)   inline (hot)
   \-> TypeProfile (419/419 counts) = java/util/LinkedList
    @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
      @ 2 j.u.LinkedList::listIterator (15 bytes) already compiled into a medium
method
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes)  inline (hot)
   \-> TypeProfile (27655/27655 counts) = java/util/LinkedList$ListItr
  @ 22  j.u.LinkedList$ListItr::next (56 bytes) inline (hot)
    @ 1  j.u.LinkedList$ListItr::checkForComodification (23 bytes) inline (hot)
    @ 5  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
  @ 35  j.l.Integer::intValue (5 bytes) accessor
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
```

№4.3: InlineSmallCode

“already compiled into a big method”:

- *Метод уже скомпилирован*
- Размер его *машинного кода* больше, чем `-XX:InlineSmallCode`

“already compiled into a medium method”:

- Метод *не горячий*
- Метод уже скомпилирован
- Размер его *машинного кода* больше, чем `-XX:InlineSmallCode / 4`

(*) `-XX:InlineSmallCode = 1000 (1.7.0+), 2000 (1.8.0+)`

№4.3: LinkedList.iterator

```
JDK: 1.7.0_80 (25.77-b03), LINKED_LIST[SIZE:65] -XX:+PrintInlining

...

@ 3 j.u.Abstr                                                    t)
 \-> TypeProfile (104/104 counts) = java/util/LinkedList
  @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
    @ 2 j.u.LinkedList::listIterator (15 bytes) executed < MinInliningThreshold
times
@ 12 j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
 \-> TypeProfile (6846/6846 counts) = java/util/LinkedList$ListItr

...

@ 5  ...::iterate (40 bytes)   inline (hot)
  @ 3  j.u.AbstractSequentialList::iterator (5 bytes)   inline (hot)
   \-> TypeProfile (419/419 counts) = java/util/LinkedList
    @ 1 j.u.AbstractList::listIterator (6 bytes) inline (hot)
      @ 2 j.u.LinkedList::listIterator (15 bytes) already compiled into a medium
method
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes)  inline (hot)
   \-> TypeProfile (27655/27655 counts) = java/util/LinkedList$ListItr
  @ 22  j.u.LinkedList$ListItr::next (56 bytes) inline (hot)
    @ 1  j.u.LinkedList$ListItr::checkForComodification (23 bytes) inline (hot)
    @ 5  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
  @ 35  j.l.Integer::intValue (5 bytes) accessor
  @ 12  j.u.LinkedList$ListItr::hasNext (20 bytes) inline (hot)
```

-XX:MinInliningThreshold = 250

-XX:InlineSmallCode = 1000

№4.3: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.7.0_80 (24.80-b11) -XX:MinInliningThreshold=100

LINKED_LIST[SIZE:65]: 12 runs, 5000 ms each

#0: ~= 0.00 b/call (14064 bytes/17743872 calls) 35771645952
#1: ~= 0.00 b/call (48 bytes/24135680 calls) 48657530880
#2: ~= 0.00 b/call (48 bytes/23661568 calls) 47701721088
#3: ~= 0.00 b/call (48 bytes/23972864 calls) 48329293824
#4: ~= 0.00 b/call (48 bytes/24044544 calls) 48473800704
...
```

№4.3: LinkedList.iterator

```
long iterate( final Iterable<Integer> iterable ) {
   long sum = 0;
   for( final Integer n : iterable ) {
      sum += n;
   }
   return sum;
}
```

```
JDK: 1.7.0_80 (24.80-b11) -XX:InlineSmallCode=2000

LINKED_LIST[SIZE:65]: 12 runs, 5000 ms each

#0: ~= 0.04 b/call (918264 bytes/22031360 calls) 44415221760
#1: ~= 0.00 b/call (48 bytes/24242176 calls) 48872226816
#2: ~= 0.00 b/call (48 bytes/24487936 calls) 49367678976
#3: ~= 0.00 b/call (48 bytes/24306688 calls) 49002283008
#4: ~= 0.00 b/call (48 bytes/24253440 calls) 48894935040
...
```

№4: Мораль

- JVM первой свежести лучше, чем не первой свежести
- `-XX:+PrintInlining` — верный друг
- Тестируйте на реальных данных
- `ArrayList` опять обставил `LinkedList`!

№4.4: Arrays.asList()

```
private final Integer[] array = new Integer[SIZE];
...
long arrayWithIndex() {
   final List<Integer> list = Arrays.asList( array );
   long sum = 0;
   for( int i = 0; i < list.size(); i++ ) {
      final Integer n = list.get( i );
      sum += n.intValue();
   }
   return sum;
}
```

bytes/call = ?

№4.4: Arrays.asList()

```
long arrayWithIndex() {
   final List<Integer> list = Arrays.asList( array );
   long sum = 0;
   for( int i = 0; i < list.size(); i++ ) {
      final Integer n = list.get( i );
      sum += n.intValue();
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

WITH_INDEX[SIZE:16]: 12 runs, 3000 ms each

#0: ~= 0.00 b/call (268584 bytes/210130944 calls)
#1: ~= 0.00 b/call (48 bytes/254804992 calls)
#2: ~= 0.00 b/call (48 bytes/223383552 calls)
#3: ~= 0.00 b/call (48 bytes/252419072 calls)
#4: ~= 0.00 b/call (48 bytes/254733312 calls)
```

№4.4: .asList().iterator()

```
private final Integer[] array = new Integer[SIZE];
...
long arrayWithIterator() {
   final List<Integer> list = Arrays.asList( array );
   long sum = 0;
   for( final Integer n : list ) {
      sum += n.intValue();
   }
   return sum;
}
```

bytes/call = ?

№4.4: .asList().iterator()

```
long arrayWithIterator() {
   final List<Integer> list = Arrays.asList( array );
   long sum = 0;
   for( final Integer n : list ) {
      sum += n.intValue();
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

WITH_ITERATOR[SIZE:16]: 12 runs, 3000 ms each

#0: ~= 24.05 b/call (4410704296 bytes/183425024 calls)
#1: ~= 24.00 b/call (5220499248 bytes/217509888 calls)
#2: ~= 24.00 b/call (5583102000 bytes/232629248 calls)
#3: ~= 24.00 b/call (5351530544 bytes/222979072 calls)
#4: ~= 24.00 b/call (5391245136 bytes/224625664 calls)
...
```

№4.4: .asList().iterator()

```
JDK: 1.8.0_77 (25.77-b03) WITH_ITERATOR[SIZE:16] -XX:+PrintInlining

@ 42   ...::arrayWithIterator (54 bytes)  inline (hot)
  @ 4 j.u.Arrays::asList (9 bytes)  inline (hot)
    @ 5 j.u.Arrays$ArrayList::<init> (16 bytes)  inline (hot)
      @ 1 j.u.AbstractList::<init> (10 bytes)  inline (hot)
        @ 1 j.u.AbstractCollection::<init> (5 bytes)  inline (hot)
          @ 1 java.lang.Object::<init> (1 bytes)  inline (hot)
      @ 6 j.u.Objects::requireNonNull (14 bytes)  inline (hot)
  @ 11 j.u.AbstractList::iterator (10 bytes) inline (hot)
   \-> TypeProfile (7304/7304 counts) = java/util/Arrays$ArrayList
    @ 6 j.u.AbstractList$Itr::<init> (6 bytes) inline (hot)
      @ 2 j.u.AbstractList$Itr::<init> (31 bytes)  inline (hot)
        @ 6 j.l.Object::<init> (1 bytes)   inline (hot)
  @ 20   j.u.AbstractList$Itr::hasNext (20 bytes) inline (hot)
   \-> TypeProfile (124168/124168 counts) = java/util/AbstractList$Itr
    @ 8 j.u.Arrays$ArrayList::size (6 bytes) inline (hot)
     \-> TypeProfile (127097/127097 counts) = java/util/Arrays$ArrayList
   @ 1 j.u.AbstractList$Itr::checkForComodification (23 bytes) inline (hot)
    @ 14 j.u.Arrays$ArrayList::get (7 bytes)   inline (hot)
     \-> TypeProfile (11316/11316 counts) = java/util/Arrays$ArrayList
  @ 43 j.l.Integer::intValue (5 bytes)   accessor
  @ 20 j.u.AbstractList$Itr::hasNext (20 bytes)   inline (hot)
```

№4.4: .asList().iterator()

```
JDK: 1.8.0_77 (25.77-b03) WITH_ITERATOR[SIZE:16] -XX:+PrintInlining

@ 42   ...::arrayWithIterator (54 bytes)  inline (hot)
  @ 4 j.u.Arrays::asList (9 bytes)  inline (hot)
    @ 5 j.u.Arrays$ArrayList::<init> (16 bytes)  inline (hot)
      @ 1 j.u.AbstractList::<init> (10 bytes)  inline (hot)
        @ 1 j.u.AbstractCollection::<init> (5 bytes)  inline (hot)
          @ 1 java.lang.Object::<init> (1 bytes)  inline (hot)
      @ 6 j.u.Objects::requireNonNull (14 bytes)  inline (hot)
  @ 11 j.u.AbstractList::iterator (10 bytes) inline (hot)
   \-> TypeProfile (7304/7304 counts) = java/util/Arrays$ArrayList
    @ 6 j.u.AbstractList$Itr::<init> (6 bytes) inline (hot)
      @ 2 j.u.AbstractList$Itr::<init> (31 bytes)  inline (hot)
        @ 6 j.l.Object::<init> (1 bytes)   inline (hot)
  @ 20   j.u.AbstractList$Itr::hasNext (20 bytes) inline (hot)
   \-> TypeProfile (124168/124168 counts) = java/util/AbstractList$Itr
    @ 8 j.u.Arrays$ArrayList::size (6 bytes) inline (hot)
     \-> TypeProfile (127097/127097 counts) = java/util/Arrays$ArrayList
   @ 1 j.u.AbstractList$Itr::checkForComodification (23 bytes) inline (hot)
    @ 14 j.u.Arrays$ArrayList::get (7 bytes)   inline (hot)
     \-> TypeProfile (11316/11316 counts) = java/util/Arrays$ArrayList
  @ 43 j.l.Integer::intValue (5 bytes)   accessor
  @ 20 j.u.AbstractList$Itr::hasNext (20 bytes)   inline (hot)
```

№4.4: .asList().iterator()

```
private final Integer[] array = new Integer[SIZE];
...
long arrayWithDetachedIterator() {
   final List<Integer> list = ArrayListEx( array );
   long sum = 0;
   for( final Integer n : list ) {
      sum += n.intValue();
   }
   return sum;
}

public class ArrayListEx<E> ... {
   ...
   public Iterator<E> iterator() {
      return new ArrayIterator<>( array );
   }
}

public class ArrayIterator<E> implements Iterator<E> {
   ...
}
```

№4.4: ArrayIterator

```
private final Integer[] array = new Integer[SIZE];
...
long arrayWithDetachedIterator() {
   final List<Integer> list = ArrayListEx( array );
   long sum = 0;
   for( final Integer n : list ) {
      sum += n.intValue();
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

WITH_DETACHED_ITERATOR[SIZE:16]: 12 runs, 3000 ms each

#0: ~= 0.00 b/call (512472 bytes/170881024 calls) 20505722880
#1: ~= 0.00 b/call (48 bytes/215118848 calls) 25814261760
#2: ~= 0.00 b/call (48 bytes/220208128 calls) 26424975360
#3: ~= 0.00 b/call (48 bytes/221283328 calls) 26553999360
#4: ~= 0.00 b/call (48 bytes/216147968 calls) 25937756160
...
```

№4.4: Collections.*

```
for( final Integer n : Collections.singleton( item ) ) {
   ...
}

for( final Integer n : Collections.singletonList( item ) ) {
   ...
}

for( final Integer n : Collections.emptySet() ) {
   ...
}

for( final Integer n : Collections.emptyList() ) {
   ...
}
```

№4.4: Collections.*

	1.8.0_77	1.7.0_80
emptyList()	+	+
emptySet()	+	+
singleton()	+	+
singletonList()	+	+
Arrays.asList()	+/−	+/−

№4.4: Выводы

- Итерация по оберткам из Collections.* скаляризуется
- …Кроме Arrays.asList()
- Вложенные объекты не скаляризуются (в том числе inner classes)
- `-XX:+PrintInlining` продолжает помогать в беде

№5: Constant size arrays

```
static final int SIZE = ...;
long constantSizeArrayWithLoops() {
   final Integer[] array = new Integer[SIZE];
   for( int i = 0; i < array.length; i++ ) {
      array[i] = 1;
   }
   long sum = 0;
   for( final Integer n : array ) {
      sum += n.intValue();
   }
   return sum;
}
```

№5.1: Variable index

```
long constantSizeArrayWithLoops() {
   final Integer[] array = new Integer[SIZE]; // SIZE = 1
   for( int i = 0; i < array.length; i++ ) {
      array[i] = 1;
   }
   long sum = 0;
   for( final Integer n : array ) {
      sum += n.intValue();
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

[SIZE=1]: 12 runs, 3000 ms each

#0: ~= 0.00 b/call (447448 bytes/27396207616 calls)
#1: ~= 0.00 b/call (48 bytes/32654928896 calls)
#2: ~= 0.00 b/call (48 bytes/31433586688 calls)
#3: ~= 0.00 b/call (48 bytes/32107270144 calls)
#4: ~= 0.00 b/call (48 bytes/32341484544 calls)
...
```

№5.1: Variable index

```
long constantSizeArrayWithLoops() {
   final Integer[] array = new Integer[SIZE]; // SIZE = 2
   for( int i = 0; i < array.length; i++ ) {
      array[i] = 1;
   }
   long sum = 0;
   for( final Integer n : array ) {
      sum += n.intValue();
   }
   return sum;
}
```

```
JDK: 1.8.0_77 (25.77-b03)

[SIZE=2]: 12 runs, 3000 ms each

#0: ~= 24.00 b/call (12577014808 bytes/524042240 calls)
#1: ~= 24.00 b/call (16702414896 bytes/695933952 calls)
#2: ~= 24.00 b/call (16792485936 bytes/699686912 calls)
#3: ~= 24.00 b/call (16740753456 bytes/697531392 calls)
#4: ~= 24.00 b/call (15752945712 bytes/656372736 calls)
...
```

№5.2: Constant index

```
long constantSizeArrayUnrolled() {
   final Integer[] array = new Integer[SIZE]; // SIZE = 2

   array[0] = array[1] = 1;

   return array[0].intValue()
         + array[1].intValue();
}
```

bytes/call = ?

№5.2: Constant index

```
long constantSizeArrayUnrolled() {
   final Integer[] array = new Integer[SIZE]; // SIZE = 2

   array[0] = array[1] = 1;

   return array[0].intValue()
         + array[1].intValue();
}
```

JDK: 1.8.0_77 (25.77-b03)

[**SIZE=2**]: 12 runs, 3000 ms each

```
#0: ~= 0.00 b/call (288792 bytes/4796071936 calls)
#1: ~= 0.00 b/call (48 bytes/3981371392 calls)
#2: ~= 0.00 b/call (48 bytes/32599294976 calls)
#3: ~= 0.00 b/call (48 bytes/32402492416 calls)
#4: ~= 0.00 b/call (48 bytes/30604343296 calls)
```

№5.2: Object arrays

	LOOP (variable indexes)	UNROLLED (constant indexes)
0	+	+
1	+	+
2..64	---	+++
65	-	-

№5.2: Object arrays

	LOOP (variable indexes)	UNROLLED (constant indexes)
0		
1		
2..64	---	+++
65	-	-

"…В JVM для всего есть свой ключик…"

`-XX:EliminateAllocationArraySizeLimit = 64`

№5.3: Primitive arrays

```
static final int SIZE = ...;
long constantSizeArrayWithLoops() {
   final int[] array = new int[SIZE]; //byte, short, double?
   for( int i = 0; i < array.length; i++ ) {
      array[i] = 1;
   }
   long sum = 0;
   for( final int n : array ) {
      sum += n;
   }
   return sum;
}
```

№5.3: Primitive arrays

	LOOP (variable indexes)	UNROLLED (constant indexes)
0	+	+
1	+	+
2..64	---	+++
65	-	-

№5.4: Preconditions

```
static final double P = ...;

void checkArgument1Arg() {
   final boolean expression = falseWithProbability( P );
   final String arg1 = ...;
   checkArgument( expression, "1-arg(%s) message", arg1 );
}
```

```
    ...  (com.google.common.base.Preconditions)  ...

public static void checkArgument(
    boolean expression,
    String errorMessageTemplate,
    Object... errorMessageArgs) {
  if ( !expression ) {
    throw new IllegalArgumentException(
       format(errorMessageTemplate, errorMessageArgs)
    );
  }
}
```

№5.4: Preconditions

```
static final double P = ...;

void checkArgument1Arg() {
   final boolean expression = falseWithProbability( P );
   final String arg1 = ...;
   checkArgument( expression, "1-arg(%s) message", arg1 );
}
```

bytes/call = f(P) ?

№5.4: Preconditions

```
static final double P = ...;

void checkArgument1Arg() {
   final boolean expression = falseWithProbability( P );
   final String arg1 = ...;
   checkArgument( expression, "1-arg(%s) message", arg1 );
}
```

JDK: 1.8.0_77 (24.80-b11)

failed probability = 1.0E-7

```
#0: ~= 22.72 b/call (3664639248 bytes/161320960 calls)
#1: ~= 23.97 b/call (4874072424 bytes/203309056 calls)
#2: ~= 24.00 b/call (5123080656 bytes/213460992 calls)
#3: ~= 24.00 b/call (5030451304 bytes/209601536 calls)
#4: ~= 24.00 b/call (5220374640 bytes/217515008 calls)
...
```

№5.4: Preconditions

```
static final double P = ...;

void checkArgument1Arg() {
   final boolean expression = falseWithProbability( P );
   final String arg1 = ...;
   checkArgument( expression, "1-arg(%s) message", arg1 );
}
```

```
JDK: 1.8.0_77 (24.80-b11)

failed probability = 1.0E-9

#0: ~= 0.02 b/call (6606064 bytes/297244672 calls)
#1: ~= 0.00 b/call (196464 bytes/254901248 calls)
#2: ~= 0.00 b/call (25432 bytes/300526592 calls)
#3: ~= 3.97 b/call (1101971992 bytes/277329920 calls)
#4: ~= 24.00 b/call (5573836848 bytes/232243200 calls)
...
```

№5.4: Preconditions

```
static final double P = ...;

void checkArgument1Arg() {
   final boolean expression = falseWithProbability( P );
   final String arg1 = ...;
   checkArgument( expression, "1-arg(%s) message", arg1 );
}
```

```
JDK: 1.8.0_77 (24.80-b11)

failed probability = 1.0E-11

#0: ~= 0.02 b/call (5851024 bytes/286074880 calls)
#1: ~= 0.00 b/call (196464 bytes/246331392 calls)
...
#9: ~= 0.00 b/call (48 bytes/324088832 calls)
#10: ~= 0.00 b/call (48 bytes/364259328 calls)
...
```

Итого — хорошо

- Скаляризуется многое из того, что хотелось бы
- Что не скаляризуется — можно понять и простить
- В свежих JVM

...расслабляться нельзя:

- Скаляризация иногда *хрупкая* и/или *нестабильная*
- Особенно в несвежих JVM
- Нужно понимать ограничения
- *Важные* сценарии нужно *тестировать*

Нормально делай — нормально будет

- Свежая JVM
- Короткие методы (проверяем инлайнинг)
- Меньше *реального* полиморфизма
- Одна ссылка указывает на один объект
- Не использовать `null`
- Не использовать вложенные объекты
- *Важные* сценарии нужно *тестировать*

Ссылки

1. "*Escape Analysis in the Context of Dynamic Compilation and Deoptimization*" Thomas Kotzmann
2. "*Escape Analysis for Java*" Jong-Deok Choi
3. "*Partial Escape Analysis and Scalar Replacement for Java*" Lukas Stadler
4. hg.openjdk.java.net/jdk9/dev/hotspot/file/68e9c20cfb88/src/share/vm/opto/escape.cpp
5. github.com/cheremin/scalarization
6. github.com/AdoptOpenJDK/jitwatch

Контакты

cheremin@gmail.com

ruslan.cheremin@db.com

cheremin.blogspot.com

Q/A

Disclaimer

Данный материал не является предложением или предоставлением какой-либо услуги. Данный материал предназначен исключительно для информационных и иллюстративных целей и не предназначен для распространения в рекламных целях. Любой анализ третьих сторон не предполагает какого-либо одобрения или рекомендации. Мнения, выраженные в данном материале, являются актуальными на текущий момент, появляются только в этом материале и могут быть изменены без предварительного уведомления.

Эта информация предоставляется с пониманием того, что в отношении материала, предоставленного здесь, вы будете принимать самостоятельное решение в отношении любых действий в связи с настоящим материалом, и это решение является основанным на вашем собственном суждении, и что вы способны понять и оценить последствия этих действий. ООО "Дойче Банк Техцентр" не несет никакой ответственности за любые убытки любого рода, относящихся к этому материалу.