ORACLE

Java8
Advanced Stream Techniques

Сергей Куксенко
sergey.kuksenko@oracle.com, @kuksenk0

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.

Stream Design

Stream design

I like to look at this as having chosen a design center that recognizes that sequential is a degenerate case of parallel, rather than treating parallel as the "weird bonus mode". I realize that this choice was controversial and definitely caused some compromises, but eventually people will have to start to unlearn their sequential biases, and there's no time like the present.

(c) Brian Goetz

http://mail.openjdk.java.net/pipermail/lambda-dev/2014-February/011870.html

Ordered/Unodered

forEach

```
collection.forEach(Consumer<T> action);
```

VS

```
stream.forEach(Consumer<T> action);
```


forEach

```
iterable.forEach(Consumer<T> action);
```

VS

```
stream.forEach(Consumer<T> action);
```


Iterable.forEach

```
/**
  * ...  Unless otherwise specified by the implementing
  * class, actions are performed in the order of
  * iteration (if an iteration order is specified).
  * ...
  */
       iterable.forEach(Consumer<T> action);
```


Stream.forEach

```
/** ...
  * The behavior of this operation is explicitly
  * nondeterministic. For parallel stream pipelines,
  * this operation does not guarantee to respect the
  * encounter order of the stream, as doing so would
  * sacrifice the benefit of parallelism.
  * If the action accesses shared state, it is
  * responsible for providing the required
  * synchronization.
  * ...
  */
      stream.forEach(Consumer<T> action);
```


Stream.forEachOrdered

```
/** ...
  * This operation processes the elements one at
  * a time, in encounter order if one exists.
  * Performing the action for one element happens-before
  * performing the action for subsequent elements,
  * but for any given element, the action may be
  * performed in whatever thread the library chooses.
  * ...
  */
      stream.forEachOrdered(Consumer<T> action);
```


Demo0

Demo0

```
List<Long> list;

public List<Long> oldSchool() {
    List<Long> l = new ArrayList<>();
    for (Long v : list) {
        if ((v & 0xff) == 0) {
            l.add(v);
        }
    }
    return l;
}
```


Demo0

Sequential/Ordered

```
list.stream()
    .filter(x -> (x & 0xff) == 0)
    .collect(Collectors.toList());
```


Demo0

Sequential/Unordered

```
list.stream()
    .unordered()
    .filter(x -> (x & 0xff) == 0)
    .collect(Collectors.toList());
```


Demo0

Parallel/Ordered

```
list.parallelStream()
    .filter(x -> (x & 0xff) == 0)
    .collect(Collectors.toList());
```


Demo0

Parallel/Unordered

```
list.parallelStream()
    .unordered()
    .filter(x -> (x & 0xff) == 0)
    .collect(Collectors.toList());
```


Results

list == range from 0 to 10000000;

oldSchool	13
Sequential/Ordered	10
Sequential/Unordered	10
Parallel/Ordered	20
Parallel/Unordered	26

throughput, ops/sec

Spliterator или что у Stream'а под капотом

Spliterator

```
interface Spliterator<T> {
    ...

    long  estimateSize(); // Long.MAX_VALUE if unknown

    boolean tryAdvance(Consumer<T> action);

    Spliterator<T>  trySplit();

    int characteristics();

    ...
}
```


Характеристки Stream'a (Spliterator'a)

ORDERED
DISTINCT
SORTED
SIZED
SUBSIZED
NONNULL
IMMUTABLE
CONCURRENT

Demo1

Demo1

Как получить сумму четных чисел Фибоначчи не превышающих 4000000 [1] ?

[1] http://projecteuler.net

Demo1

Как получить сумму четных чисел Фибоначчи не превышающих N?

Demo1 prequel

- Получить Фибоначчи Stream
- Сложить первые 4096 элементов

Demo1 prequel results

sum of limit(4096)

	'no load'	
OldSchool	849	
Generator/Sequential	804	
Iterator/Sequential	760	
Iterate/Sequential	662	
Iterator/Parallel	219	
Iterate/Parallel	223	

throughput, ops/sec

Demo1 prequel results

sum of limit(4096)

	'no load'	'heavy load'
OldSchool	849	55
Generator/Sequential	804	53
Iterator/Sequential	760	53
Iterate/Sequential	662	54
Iterator/Parallel	219	105
Iterate/Parallel	223	106

throughput, ops/sec

Demo1 results

$$N = 4 * 10^{2048}$$

	'no load'	
OldSchool	239	
Iterator/Sequential	225	
Iterate/Sequential	216	
Iterator/Parallel	208	
Iterate/Parallel	209	
throughput, ops/sec		

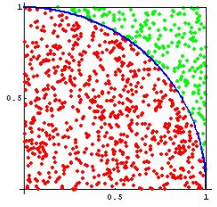
Demo1 results

$$N = 4 * 10^{2048}$$

	'no load'	'heavy load'
OldSchool	239	56
Iterator/Sequential	225	55
Iterate/Sequential	216	54
Iterator/Parallel	208	72
Iterate/Parallel	209	72
throughput, ops/sec		

Demo2

MonteCarlo



$$\pi = 4 \times \frac{M}{N}$$

N - брошено

M - попало

MonteCarlo results

OldSchool	14
ZipBoxed/Sequential	126
ZipDouble/Sequential	23
ZipDouble/Parallel	20
ZipUnsafe/Sequential	24
ZipUnsafe/Parallel	9
ZipPaired/Sequential	22
ZipPaired/Parallel	8
time, secs/op	

Leibniz

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Leibniz results

OldSchool	1175
Stream/Sequential	1507
Stream/Parallel	600
time, ms/op	

Thank you!

Q & A ?

