Yandex Cloud

Как готовить свой код к виртуальным потокам

Олег Естехин
Yandex Cloud
@OlegEstekhin

О чём не будет этот доклад

- Что такое виртуальные потоки

Project loom: Modern scalable concurrency for the Java platform

Alan Bateman

Joker 2020

https://www.youtube.com/watch?v=7GLVROqgQJY

Thread Wars: проект Loom наносит ответный удар

Иван Углянский

JPoint 2022

https://www.youtube.com/watch?v=kwS3OeoVCno

О чём не будет этот доклад

- Что такое виртуальные потоки
- Как виртуальные потоки устроены изнутри

The Challenges of Introducing Virtual Threads to the Java Platform - Project Loom

Alan Bateman

JVMLS 2023

https://youtu.be/WsCJYQDPrrE

Continuations - Under the Covers

Ron Pressler

JVMLS 2023

https://youtu.be/6nRS6UiN7X0

О чём не будет этот доклад

- Что такое виртуальные потоки
- Как виртуальные потоки устроены изнутри
- Как виртуальные потоки будут развиваться дальше

Virtual Thread’s Next Steps

Alan Bateman

FOSDEM 2024

https://fosdem.org/2024/schedule/event/fosdem-2024-3255-virtual-thread-s-next-steps/

О чём будет этот доклад

- Как хотя бы попробовать виртуальные потоки в своём проекте

О чём будет этот доклад

- Как хотя бы попробовать виртуальные потоки в своём проекте
- На что обратить внимание в процессе перехода на виртуальные потоки

Почему мне понадобился этот доклад

```
jcmd "${JAVA_PID?}" VM.info | grep "CPU: total"
CPU: total 12 (initial active 12) (6 cores per cpu, 2 threads per core)

jcmd "${JAVA_PID?}" Thread.print | grep tid | wc -l
3346
```

Не хватает цпу

Чем занимаются 3000 потоков на 12 ядрах?

Не хватает памяти

3000 потоков * 1MB ~ 3GB на стеки потоков

Не хватает железа

VM

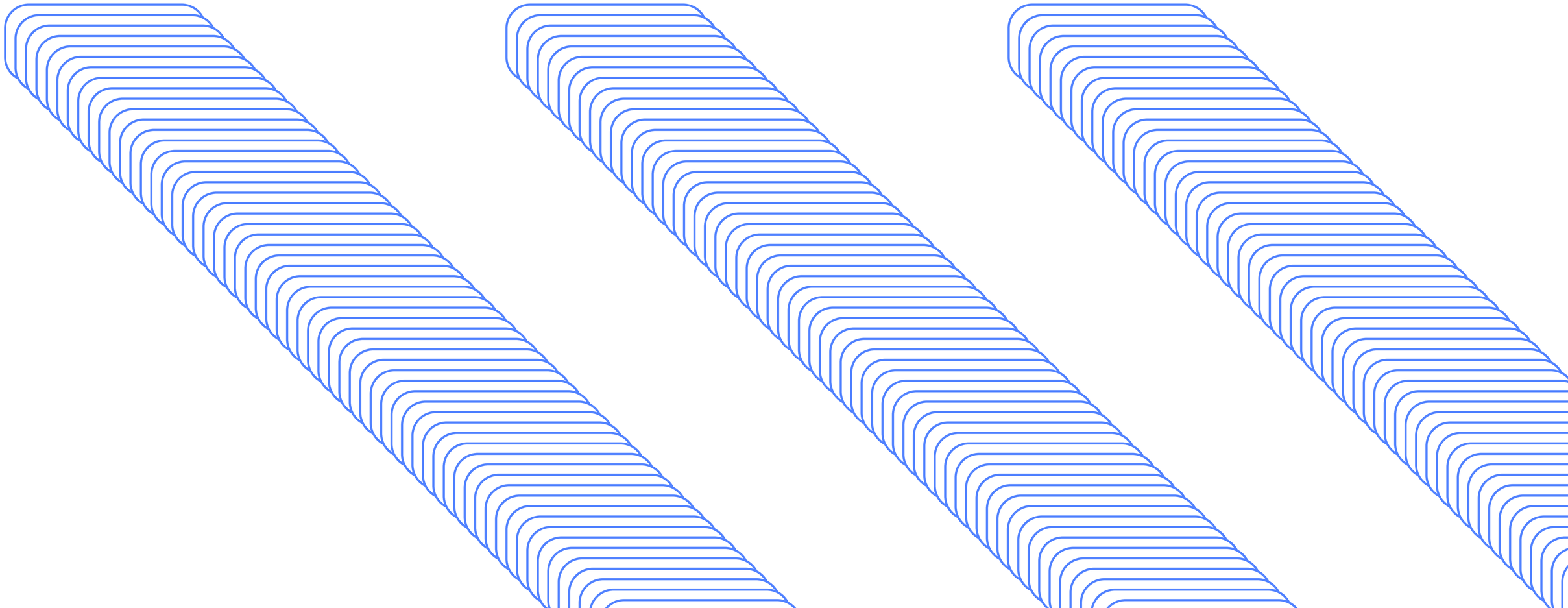
VM

VM

Не хватает железа

Не хватает железа

Не хватает железа



Не хватает цпу, памяти и железа для чего?

```
обработатьЗапрос() {
  var сущность = базаДанных.подождать()
  var дополнительныеДанные = клиентСмежногоСервиса.подождать()
  var обновлённаяСущность = сущность + дополнительныеДанные
  базаДанных.подождать(обновлённаяСущность)
  return обновлённаяСущность
}
```

Можно продолжать заливать железом

Можно продолжать заливать железом

Есть пределы толщине и количеству железа

Что делать

- Ничего
- Перейти на Async
- Перейти на Reactive
- Перейти на Kotlin
- Перейти на виртуальные потоки

Что делать

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего			
Async			
Reactive			
Kotlin			
Virtual Threads			

Ничего

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Не меняются*	Не меняется*
Async			
Reactive			
Kotlin			
Virtual Threads			

Ничего

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Растут	Деградирует
Async			
Reactive			
Kotlin			
Virtual Threads			

Перейти на Async

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Растут	Деградирует
Async	Средние	Средние	Задержки? Пропускная способность+
Reactive			
Kotlin			
Virtual Threads			

Перейти на Reactive

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Растут	Деградирует
Async	Средние	Средние	Задержки? Пропускная способность+
Reactive	Высокие	Высокие	Задержки+ Пропускная способность+
Kotlin			
Virtual Threads			

Перейти на Kotlin

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Растут	Деградирует
Async	Средние	Средние	Задержки? Пропускная способность+
Reactive	Высокие	Высокие	Задержки+ Пропускная способность+
Kotlin	Высокие	Не меняются?	Задержки? Пропускная способность+
Virtual Threads			

Перейти на виртуальные потоки

Что делать	Трудозатраты сейчас	Трудозатраты в будущем	Производительность
Ничего	Не меняются	Растут	Деградирует
Async	Средние	Средние	Задержки? Пропускная способность+
Reactive	Высокие	Высокие	Задержки+ Пропускная способность+
Kotlin	Высокие	Не меняются?	Задержки? Пропускная способность+
Virtual Threads	Не меняются	Не меняются?	Задержки? Пропускная способность+

Виртуальные потоки выглядят привлекательно

Что такое виртуальные потоки

- M:N Threading

Трудности перевода

- Mount – виртуальный поток ставится на платформенный
- Unmount – виртуальный поток снимается с платформенного
- Pinned – виртуальный поток мог бы сняться с платформенного, но не может

Что такое виртуальные потоки

- M:N
- JEP 444: Virtual Threads говорит, что «virtual threads are not cooperative»

Когда виртуальный поток снимается

- `LockSupport.park()`
- `Thread.sleep()`
- `Thread.yield()`

Виртуальные потоки лучше всего умеют ждать

Когда виртуальный поток пинится

- `synchronized`
- Нативный код на стеке

Когда виртуальный поток пинится в Java 21

- `synchronized`
- Нативный код на стеке

Когда виртуальный поток пинится в Java ~~21~~22

- `synchronized`
- Нативный код на стеке

Пин из-за synchronized

- Ссылки на мониторы хранятся в нативной реализации потока

Пин из-за synchronized

```
synchronized (monitor) {
    Thread.sleep(Duration.ofDays(1));
    System.out.println("How long was I asleep?");
}
```

Пин из-за synchronized

```
Thread.sleep(Duration.ofDays(1));
synchronized (monitor) {
    System.out.println("How long was I asleep?");
}
```

Пин из-за synchronized

- Quality of Implementation
- Ссылки на мониторы хранятся в нативной реализации потока
  - Можно копировать эти данные при unmount/mount

Пин из-за нативного кода на стеке

- Вернулись из нативного кода в Java код

Пин из-за нативного кода на стеке

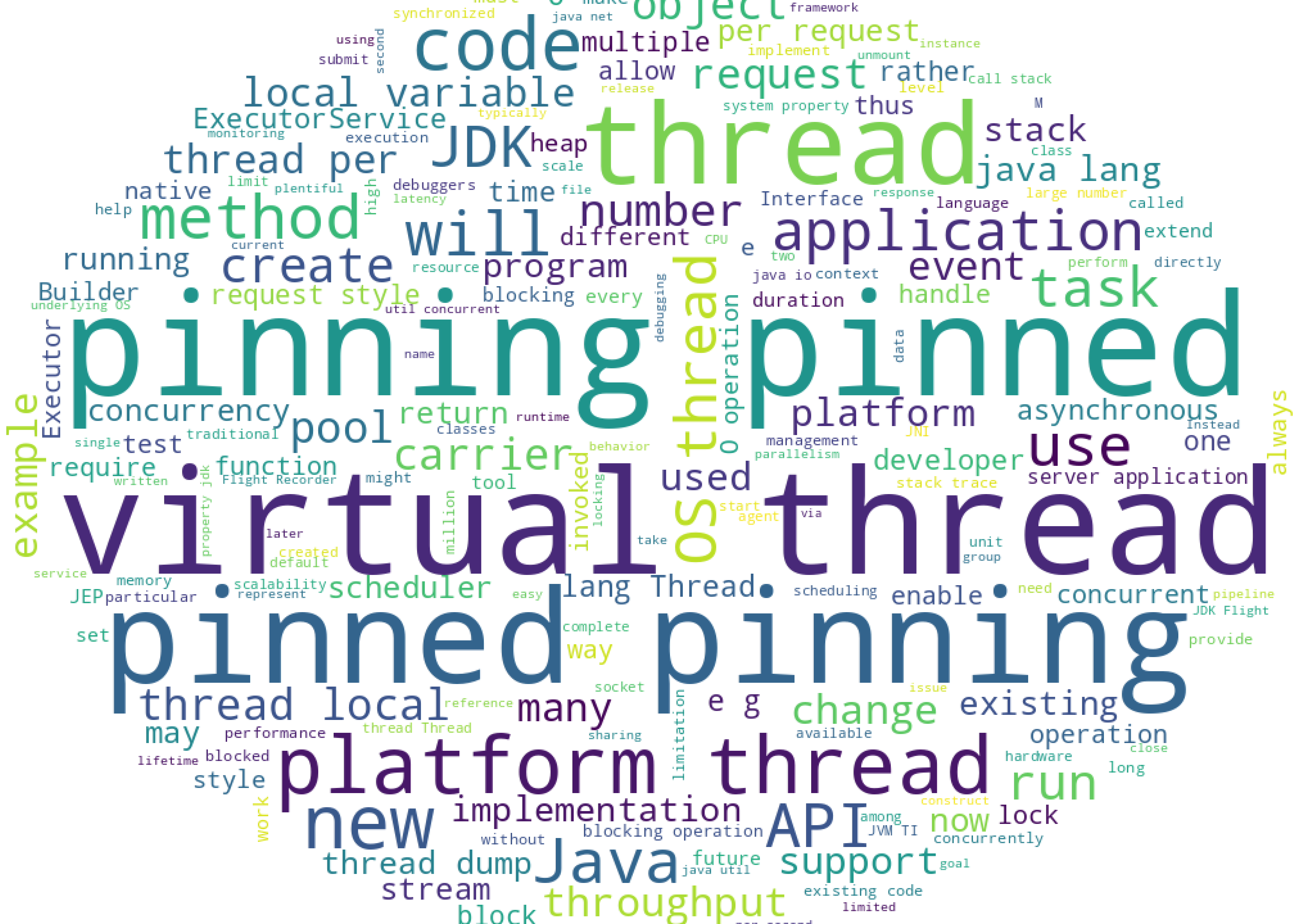
- Вернулись из нативного кода в Java код
  - Загрузка и инициализация классов

Начитались интернета и боимся запиниться

- А вдруг у нас коде есть `synchronized`?

Начитались интернета и боимся запиниться

- А вдруг у нас коде есть `synchronized`?
  - Если не у нас, то в зависимостях-то точно есть?


synchronized
java net
object
framework
code
multiple
per request
implement
instance
using
submit
second
unmount
allow
request
rather
call stack
release
level
local variable
system property
thus
M
typically
ExecutorService
thread
monitoring
execution
stack
JDK
heap
class
thread per
scale
java lang
limit
plentiful
native
time
file
Interface
response
large number
high
debuggers
latency
language
called
help
method
will
number
application
extend
current
different
CPU
e
running
create
program
two
perform
directly
resource
event
java io
context
task
Builder
request style
blocking
every
duration
handle
underlying OS
util concurrent
debugging
pinning
pinned
Executor
name
data
os thread
operation
concurrency
platform
asynchronous
example
pool
return
runtime
Instead
classes
use
JNI
single
test
traditional
management
one
always
carrier
behavior
parallelism
require
function
developer
used
written
jdk
Flight Recorder
might
server application
tool
invoked
stack trace
virtual thread
locking
million
start
agent
via
property
later
take
unit
created
default
group
service
memory
scalability
scheduler
lang Thread
scheduling
enable
need
concurrent
pipeline
JEP
particular
represent
easy
JDK Flight
set
complete
provide
way
pinned pinning
socket
issue
thread local
reference
many
e g
change
existing
may
thread Thread
limitation
available
operation
performance
sharing
close
lifetime
blocked
hardware
platform thread
long
style
run
construct
work
implementation
API
among
now
lock
new
without
blocking operation
JVM TI
concurrently
Java
future
thread dump
support
goal
java util
stream
existing code
throughput
limited
block

Что делать

- Перейти на Java 21+
- Перейти на виртуальные потоки
- Использовать `Lock` вместо `synchronized`
- Минимизировать использование `ThreadLocal`
- Использовать семафоры для ограничения доступа к ресурсам
- Задуматься о Structured Concurrency

Перейти на Java 21+

- Обновить JDK/JRE до 21+
- Обновить зависимости

Перейти на виртуальные потоки

```
var executorService = Executors.newVirtualThreadPerTaskExecutor();
```

Перейти на виртуальные потоки

```
var executorService1 = Executors.newThreadPerTaskExecutor(Thread
    .ofVirtual()
    .name("virtual-first-", 0)
    .factory()
);
var executorService2 = Executors.newThreadPerTaskExecutor(Thread
    .ofVirtual()
    .name("virtual-second-", 0)
    .factory()
);
```

Перейти на виртуальные потоки

```
spring.threads.virtual.enabled=True

quarkus.virtual-threads.enabled=True // default

reactor.schedulers.defaultBoundedElasticOnVirtualThreads=True
```

Перейти на виртуальные потоки

- Сейчас шедулер один на всех
- Сейчас настройки шедулера через свойства

Использовать `Lock` вместо `synchronized`

Использовать `Lock` вместо `synchronized`

- JEP 444: Virtual Threads говорит, что
  «no need to replace synchronized blocks and methods that are used infrequently»

Использовать Lock вместо synchronized

```
var monitor = new Object();

synchronized (monitor) {
}
```

Использовать Lock вместо synchronized

```
var lock = new ReentrantLock();

lock.lock();
try {
} finally {
    lock.unlock();
}
```

Использовать Lock вместо synchronized

```
class AutoCloseableReentrantLock
    extends ReentrantLock
    implements AutoCloseable
```

```
class AutoCloseableReentrantLock
    extends ReentrantLock
    implements AutoCloseable
{
    public AutoCloseableReentrantLock obtain() {
        lock();
        return this;
    }
    @Override
    public void close() {
        unlock();
    }
}
```

Использовать Lock вместо synchronized

```
var lock = new AutoCloseableReentrantLock();

try (var _ = lock.obtain()) {
}
```

Иногда нельзя просто так взять и использовать `Lock` вместо `synchronized`

- Когда `synchronized` часть публичного API
- Когда монитор дешевле `Lock`

Когда `synchronized` часть публичного API

Когда `synchronized` часть публичного API

- `StringBuffer`

Когда `synchronized` часть публичного API

- `StringBuffer`
- `Vector, Hashtable`

Когда `synchronized` часть публичного API

- `StringBuffer`
- `Vector, Hashtable`
- `InputStream, OutputStream` и их подклассы

Showing **1,133 changed files** with **95,870 additions** and **8,270 deletions**.

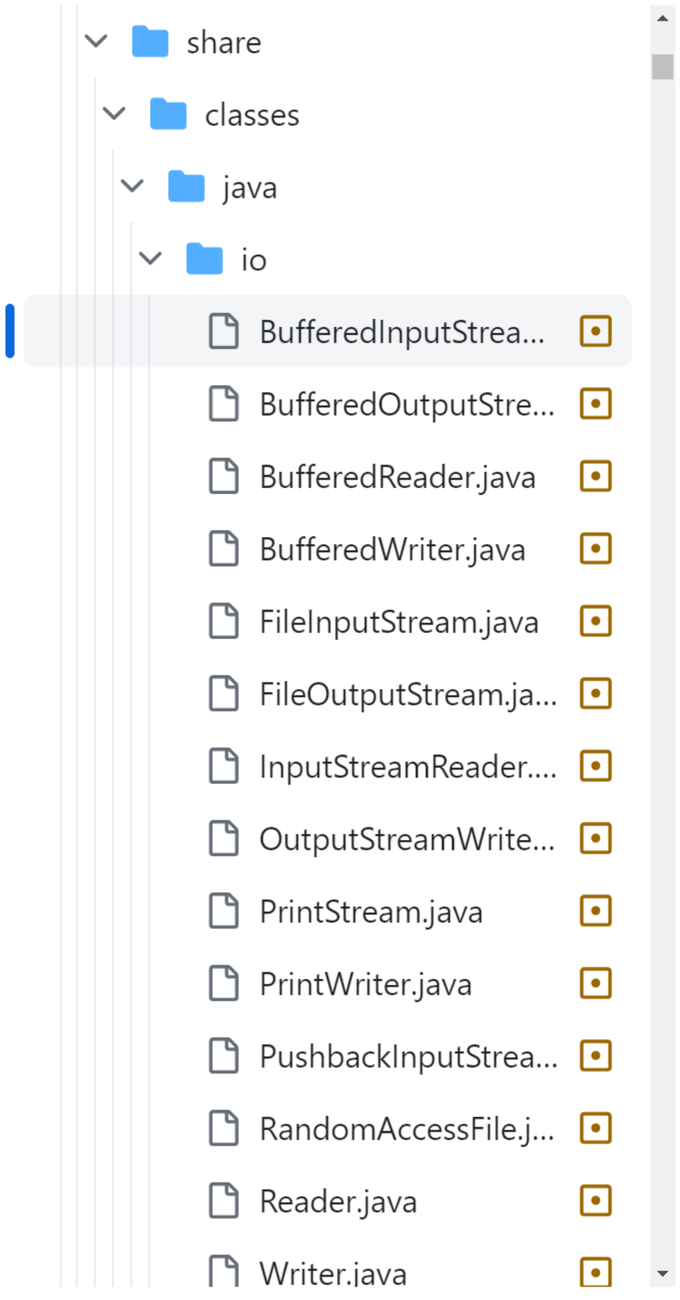
Whitespace | Ignore whitespace | Split | Unified

- share
  - classes
    - java
      - io
        - BufferedInputStrea...
        - BufferedOutputStre...
        - BufferedReader.java
        - BufferedWriter.java
        - FileInputStream.java
        - FileOutputStream.ja...
        - InputStreamReader....
        - OutputStreamWrite...
        - PrintStream.java
        - PrintWriter.java
        - PushbackInputStrea...
        - RandomAccessFile.j...
        - Reader.java
        - Writer.java

```
@@ -25,7 +25,7 @@
25  25
26  26  package java.io;
27  27
    28  + import jdk.internal.misc.InternalLock;
28  29  import jdk.internal.misc.Unsafe;
29  30  import jdk.internal.util.ArraysSupport;
30  31
@@ -63,6 +64,9 @@ public class BufferedInputStream extends FilterInputStream {
63  64      private static final long BUF_OFFSET
64  65              = U.objectFieldOffset(BufferedInputStream.class, "buf");
65  66
    67  +   // initialized to null when BufferedInputStream is sub-classed
    68  +   private final InternalLock lock;
    69  +
66  70      /**
67  71       * The internal buffer array where the data is stored. When necessary,
68  72       * it may be replaced by another array of
@@ -179,32 +183,39 @@
179 183       */
180 184      public BufferedInputStream(InputStream in) {
181 185          this(in, DEFAULT_BUFFER_SIZE);
182 186      }
183 187
184 188      /**
185 189       * Creates a {@code BufferedInputStream}
186 190       * with the specified buffer size,
```

Showing **1,133 changed files** with **95,870 additions** and **8,270 deletions**.

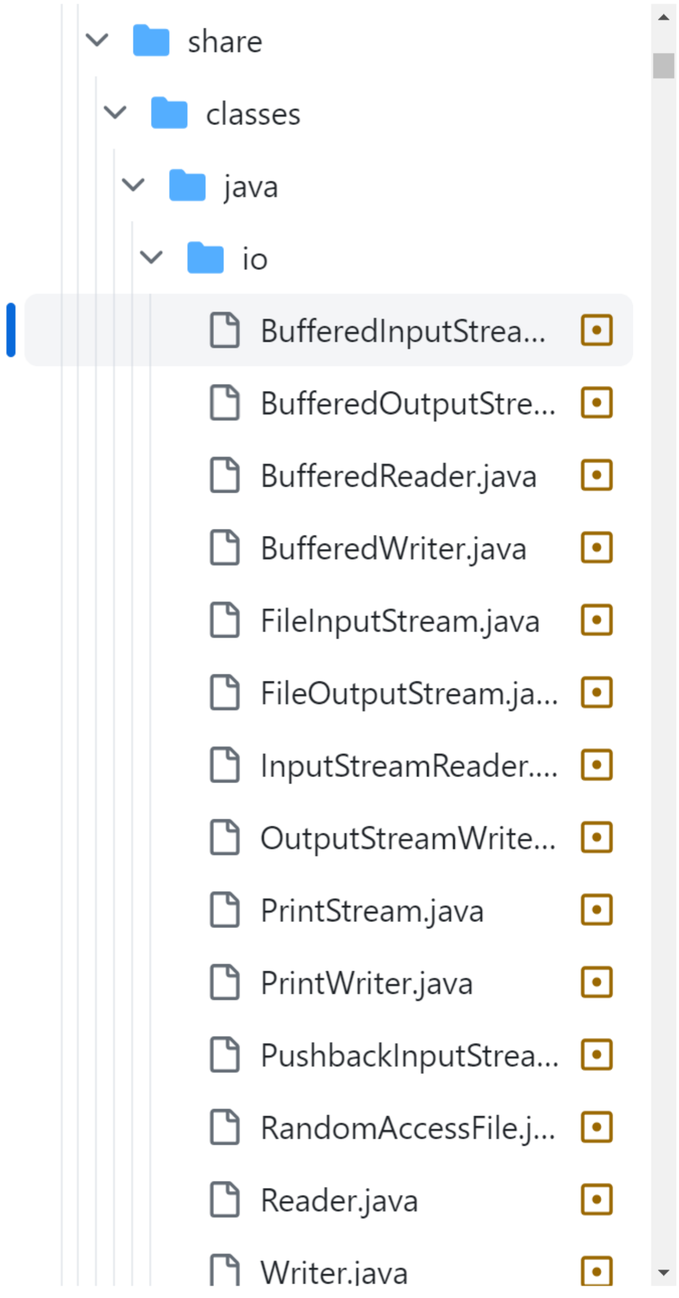
Whitespace | Ignore whitespace | Split | Unified




```
190  194          * is created and stored in {@code buf}.
191  195          *
192  196          * @param   in     the underlying input stream.
193  197          * @param   size   the buffer size.
194  198          * @throws  IllegalArgumentException if {@code size <= 0}.
195  199          */
196  200         public BufferedInputStream(InputStream in, int size) {
197  201             super(in);
198  202             if (size <= 0) {
199  203                 throw new IllegalArgumentException("Buffer size <= 0");
200  204             }
201  205             buf = new byte[size];
     206 +
     207 +           // use monitors when BufferedInputStream is sub-classed
     208 +           if (getClass() == BufferedInputStream.class) {
     209 +               lock = InternalLock.newLockOrNull();
     210 +           } else {
     211 +               lock = null;
     212 +           }
202  213         }
203  214
204  215         /**
205  216          * Fills the buffer with more data, taking into account
206  217          * shuffling and other tricks for dealing with marks.
207      -        * Assumes that it is being called by a synchronized method.
     218 +        * Assumes that it is being called by a locked method.
208  219          * This method also assumes that all data has already been read in,
209  220          * hence pos > count.
```

Showing **1,133 changed files** with **95,870 additions** and **8,270 deletions**.

Whitespace | Ignore whitespace | Split | Unified




```
255  266          *             stream is reached.
256  267          * @throws     IOException  if this input stream has been closed by
257  268          *                          invoking its {@link #close()} method,
258  269          *                          or an I/O error occurs.
259  270          * @see        java.io.FilterInputStream#in
260  271          */
261       -       public synchronized int read() throws IOException {
         272   +   public int read() throws IOException {
         273   +       if (lock != null) {
         274   +           lock.lock();
         275   +           try {
         276   +               return implRead();
         277   +           } finally {
         278   +               lock.unlock();
         279   +           }
         280   +       } else {
         281   +           synchronized (this) {
         282   +               return implRead();
         283   +           }
         284   +       }
         285   +   }
         286   +
         287   +   private int implRead() throws IOException {
262  288              if (pos >= count) {
263  289                  fill();
264  290                  if (pos >= count)
265  291                      return -1;
266  292              }
```

Когда `synchronized` часть публичного API

- Есть способы убрать `synchronized` и сохранить совместимость
- Вряд ли это вам понадобится, если вы не разработчик JDK

Когда монитор дешевле Lock

- Монитор «бесплатный», по крайней мере пока им не пользуются

```
java -jar jol-cli.jar footprint java.util.concurrent.locks.ReentrantLock
# VM mode: 64 bits
# Compressed references (oops): 3-bit shift
# Compressed class pointers: 0-bit shift and 0x78FD03000000 base
# Object alignment: 8 bytes

Instantiated the sample instance via default constructor.

java.util.concurrent.locks.ReentrantLock@50675690d footprint:
     COUNT       AVG       SUM   DESCRIPTION
         1        16        16   java.util.concurrent.locks.ReentrantLock
         1        32        32   java.util.concurrent.locks.ReentrantLock$NonfairSync
         2                  48   (total)
```

Когда монитор дешевле `Lock`

- Монитор «бесплатный», по крайней мере пока им не пользуются
- `ReentrantLock` +48 байт, даже если им не пользоваться

ConcurrentHashMap

* Other update operations (insert,
* delete, and replace) require locks. We do not want to waste
* the space required to associate a distinct lock object with
* each bin, so instead use the first node of a bin list itself as
* a lock. Locking support for these locks relies on builtin
* "synchronized" monitors.

Когда монитор дешевле Lock

- На мониторах можно сэкономить память
- Вряд ли это вам понадобится, если вы не разработчик JDK

Использовать `Lock` вместо `synchronized`

- А может просто подождать поддержки `synchronized`?

Минимизировать использование `ThreadLocal`

Минимизировать использование `ThreadLocal`

- `ThreadLocal` как Scoped Value
- `ThreadLocal` как кэш

ThreadLocal как Scoped Value

```
var threadLocal = new ThreadLocal<>();

threadLocal.set("value");
try {
} finally {
    threadLocal.remove();
}
```

`ThreadLocal` как кэш

- `ThreadLocal` как кэш не имеет смысла для виртуальных потоков

`ThreadLocal` как кэш

- `ThreadLocal` как кэш не имеет смысла для виртуальных потоков
  - Либо мусорим
  - Либо переходим на кэши объектов
    (которые не используют `ThreadLocal`)

ThreadLocal как кэш

```
var factory = new JsonFactoryBuilder()
    .configure(
        JsonFactory.Feature.USE_THREAD_LOCAL_FOR_BUFFER_RECYCLING,
        false
    )
    .build();

var mapper = new ObjectMapper(factory);
```

Минимизировать использование `ThreadLocal`

- `ThreadLocal` как Scoped Value – ждать продолжение JEP 464: Scoped Values
- `ThreadLocal` как кэш – избавляться

Ограничивать параллельный доступ к ресурсам

Ограничивать параллельный доступ к ресурсам

- Максимальный размер пула потоков это неявный ограничитель параллельного доступа к лимитированным ресурсам

Ограничивать параллельный доступ к ресурсам

- Максимальный размер пула потоков это неявный ограничитель параллельного доступа к лимитированным ресурсам
- Максимальный размер очереди задач для пула потоков это неявный backpressure для источника задач

Использовать Semaphore

```
var semaphore = new Semaphore(42);

semaphore.acquireUninterruptibly();
try {
} finally {
    semaphore.release();
}
```

Использовать Semaphore

```
class AutoCloseableSemaphore
    extends Semaphore
    implements AutoCloseable
```

```
class AutoCloseableSemaphore
    extends Semaphore
    implements AutoCloseable
{
    public AutoCloseableSemaphore obtain() {
        acquireUninterruptibly();
        return this;
    }
    @Override
    public void close() {
        release();
    }
}
```

Использовать Semaphore

```
var semaphore = new AutoCloseableSemaphore(42);

try (var _ = semaphore.obtain()) {
}
```

Что делать с backpressure?

- Обратная связь для источника задач для виртуального пула отсутствует!

Ограничивать параллельный доступ к ресурсам

- ~~А может просто подождать~~
- Ждать нельзя!

Structured Programming

```
var lock = new ReentrantLock();

lock.lock();
try {
} finally {
    lock.unlock();
}
```

Structured Programming

```
var lock = new AutoCloseableReentrantLock();

try (var _ = lock.obtain()) {
}
```

Когда ход выполнения программы разветвляется, все ветви должны сойтись обратно в определённой точке

JEP 462: Structured Concurrency (Second Preview)

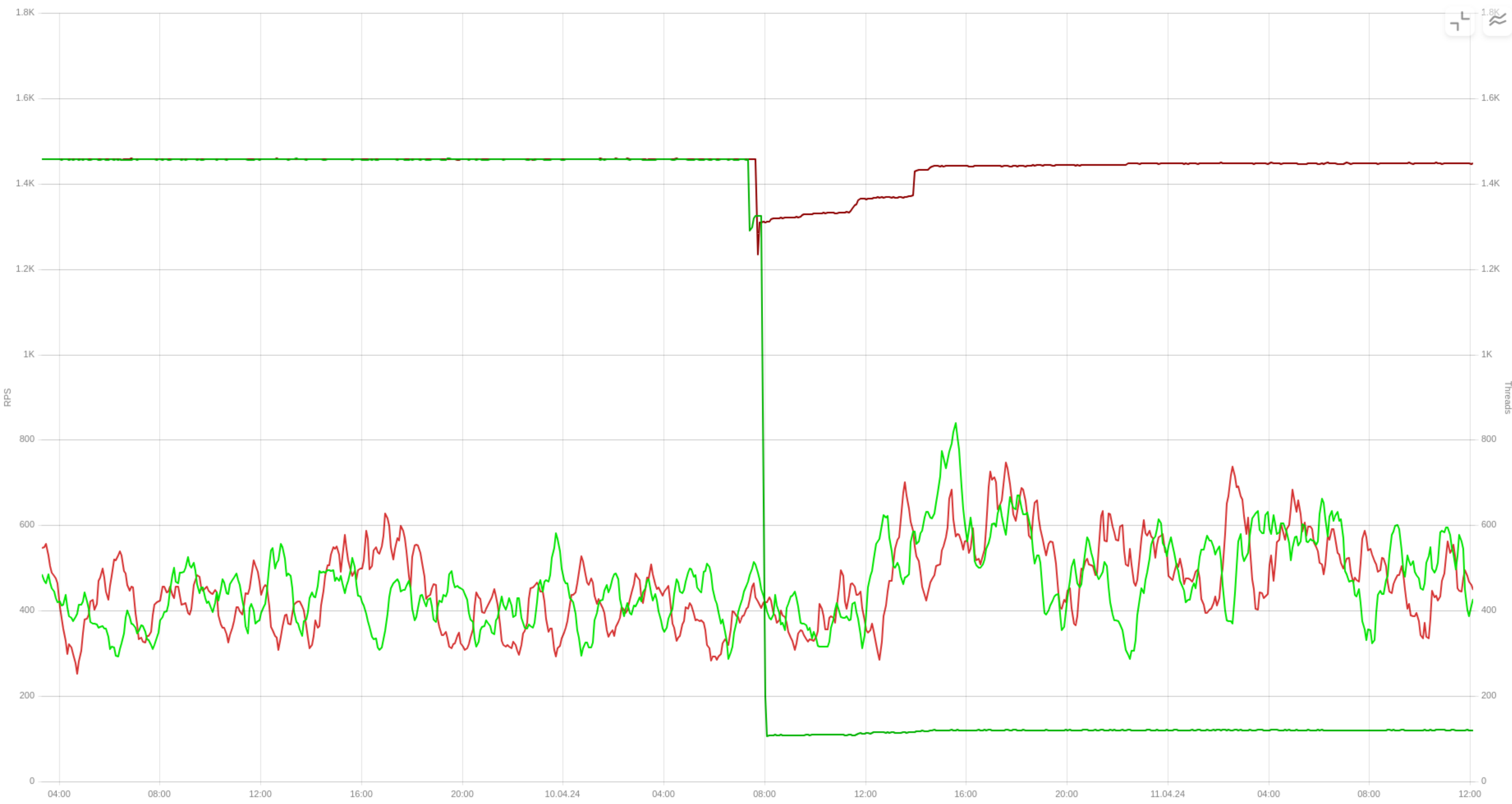
```
try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
    var subtask1 = scope.fork(() -> "Hello, ");
    var subtask2 = scope.fork(() -> "World!");

    scope.join();

    System.out.println(subtask1.get() + subtask2.get());
}
```

Structured Concurrency

- Ждать продолжения JEP 462: Structured Concurrency

Что мы сделали

- Обновили Java и зависимости
- Заменили часть пулов на виртуальные
- Не стали заменять `synchronized`, пока
- Не стали добавлять дополнительных семафоров, пока


1.8K
1.6K
1.4K
1.2K
1K
800
600
400
200
0
RPS
Threads
04:00
08:00
12:00
16:00
20:00
10.04.24
04:00
08:00
12:00
16:00
20:00
11.04.24
04:00
08:00
12:00

Что мы получили

- Проверили, что ничего не сломалось

Что мы получили

- Проверили, что ничего не сломалось
- Проверили, что ключевые метрики не стали хуже
- Убедились, что виртуальные потоки подходят для наших задач

Что мы будем делать дальше

- Разбираться с тестами производительности для увеличения пропускной способности
- Разбираться с темой ограничителей скорости для ограничения пропускной способности

Ждать или не ждать,
вот в чём вопрос…

Спасибо!

Олег Естехин
Yandex Cloud
@OlegEstekhin