

Usando virtual threads para incrementar dramáticamente la paralelización de tu aplicación

JConf Perú 2023

Andrés Alcarraz

Sábado 2º de Diciembre de 2023



Acerca de mí

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Acerca de mí

- Uruguay 🇺🇾
- 24 años desarrollando en Java.
- @ alcarraz@gmail.com
- [linkedin.com/in/andresalcarraz](https://www.linkedin.com/in/andresalcarraz)
- [X/andresalcarraz](https://twitter.com/X/andresalcarraz)
- github.com/alcarraz
- [stackoverflow/3444205](https://stackoverflow.com/users/3444205)



Un poco de historia

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo.
[Ora04]



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
 - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
 - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
 - Scheduling.



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
 - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
 - Scheduling.
- Programación reactiva, nombre cool, pero lo cool termina ahí.



Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
 - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
 - Scheduling.
- Programación reactiva, nombre cool, pero lo cool termina ahí.
- Java 21: virtual threads, bloquear está bien de nuevo.



Un poco de historia

El problema con los threads nativos

- Costosos de crear.



Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.



Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.
- En llamadas bloqueantes, los recursos quedan bloqueados.



Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.
- En llamadas bloqueantes, los recursos quedan bloqueados.
- Veamos un ejemplo, ¡Coding time!.



Un poco de historia

La solución reactiva

Ejemplo tomado de José Paumard, que a su vez lo toma de Tomasz Nurkiewicz. [Pau23]

```
User user = userService.findUserByName(name);  
if (!repo.contains(user)) repo.save(user);  
  
var cart = cartService.loadCartFor(user);  
var total = cart.items().stream()  
    .mapToInt(Item::price)  
    .sum();  
  
var transactionId = paymentService.pay(user, total);  
emailService.send(user, cart, transactionId);
```



Un poco de historia

La solución reactiva

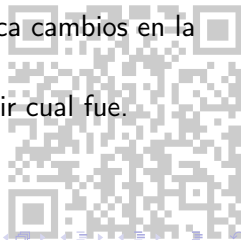
```
var future = supplyAsync(() -> userService.findUserByName(name))
    .thenCompose(
        user -> allof(
            supplyAsync(() -> !repo.contains(user))
                .thenAccept {
                    doesNotContain -> {
                        if (doesNotContain) repo.save(user);
                    }
                },
            supplyAsync(
                () -> cartservice.loadCartFor(user)
                    .thenApply(
                        cart ->
                            supplyAsync(() -> cart.items().stream().mapToInt(Item::price).sum())
                                .thenApply(
                                    total -> paymentService.pay(user, total))
                                        .thenAccept {
                                            transactionId -> emailService.send(user, cart, transactionId)
                                        }
                                )
                            )
                    )
            )
    )
)
```



Un poco de historia

La solución reactiva

- Operaciones escritas como lambdas
- El resultado de una es pasado a la siguiente.
- No se debe bloquear en las lambdas.
 - Cuando se está esperando por un recurso externo, retornar con un future.
- Adiós a la programación imperativa.
- La lógica de negocio se mezcla con la lógica del framework →complica cambios en la lógica de negocio.
- Si una lambda, genera un error en el framework, no es trivial descubrir cual fue.



Propiedades de los Virtual Threads

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.
- No se necesita hacer pool de ellos.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.
- No se necesita hacer pool de ellos.
- Son todos de tipo daemon.



Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.
- No se necesita hacer pool de ellos.
- Son todos de tipo daemon.
- Revisitemos el ejemplo anterior.



Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.



Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.



Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.
- Usar semáforos para controlar esto.



Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.
- Usar semáforos para controlar esto.
- Se puede crear un executor personalizado para esto.



Propiedades de los Virtual Threads

Como se usan

- Factory method en la clase Thread

```
Thread virtual = Thread.ofVirtual().start(  
    () -> System.out.println("Hola Mundo!"));
```

- Con executors:

```
ExecutorService service  
    = Executors.newVirtualThreadPerTaskExecutor();  
...  
service.execute(  
    () -> System.out.println("Hola Mundo!"));
```

- Por cierto, ExecutorService ahora implementa AutoCloseable. ¡Se puede utilizar en try-with-resources!



Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!



Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.



Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.
 - Operaciones en bases de datos.



Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.
 - Operaciones en bases de datos.
 - Llamadas a servicios REST.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
 - No deberíamos ejecutar código bloqueante en parallel streams.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
 - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
 - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.
 - No llamar a código nativo que pueda bloquearse.



Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
 - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.
 - No llamar a código nativo que pueda bloquearse.
 - No usar bloques `synchronized` para sincronizar cosas que pueden demorar.
Usar `ReentrantLock` en su lugar



Detalles de implementación

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

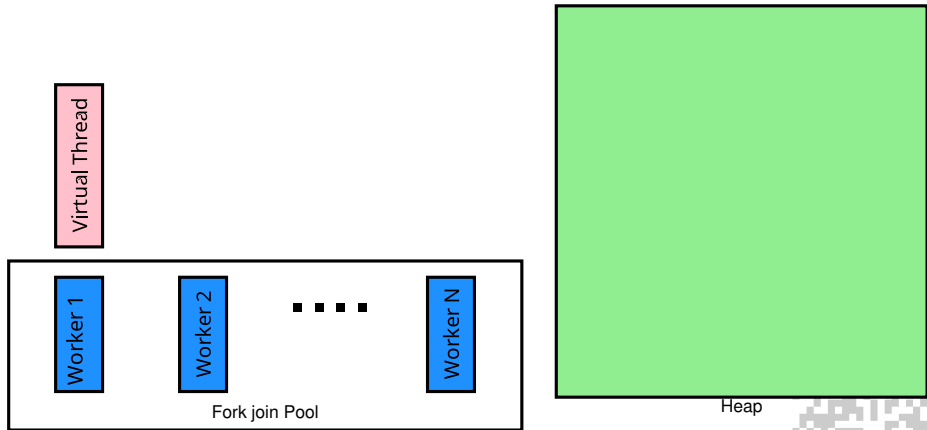
- Pull request
- Thread.*sleep*

5 Frameworks

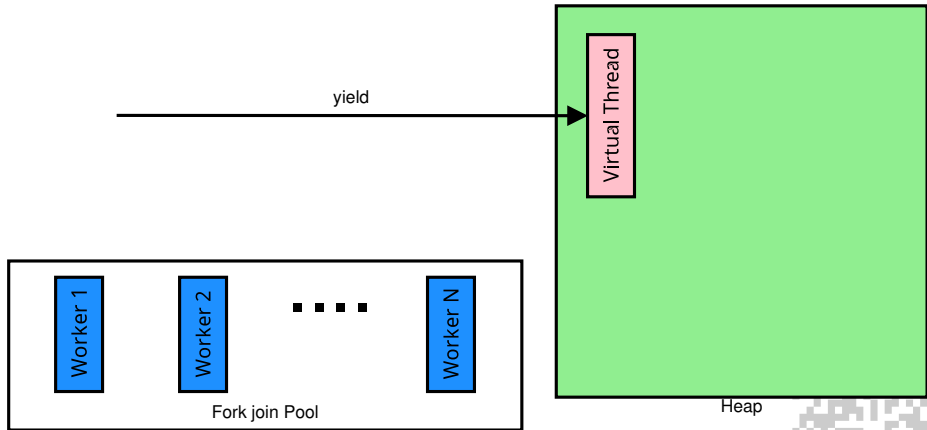
6 Más de project Loom



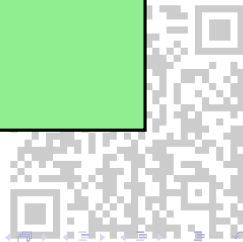
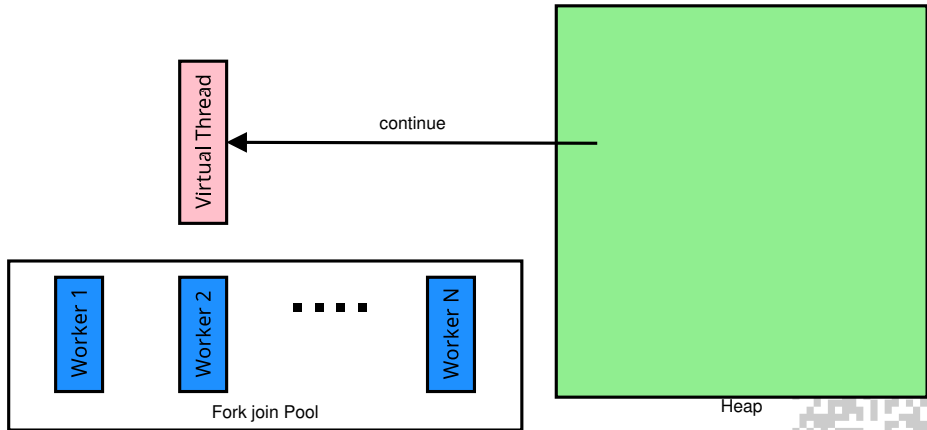
Detalles de implementación



Detalles de implementación



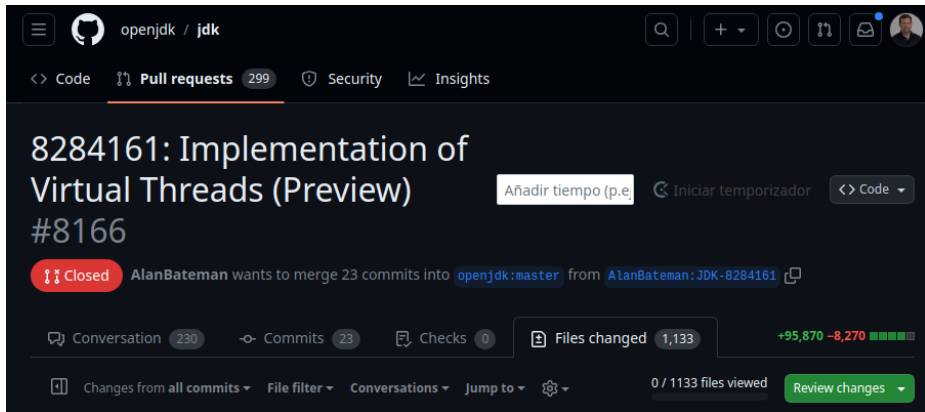
Detalles de implementación



Detalles de implementación

Pull request

Se refactorizó todo el código bloqueante de la JDK, para llamar a `Continuations.yield()`



The screenshot shows a GitHub pull request interface. At the top, the repository is 'openjdk / jdk'. The pull request title is '8284161: Implementation of Virtual Threads (Preview)' with a preview icon and a 'Code' dropdown. Below the title is the PR number '#8166'. A red 'Closed' badge is visible. The description states 'AlanBateman wants to merge 23 commits into openjdk:master from AlanBateman:JDK-8284161'. The 'Files changed' section shows 1,133 files changed, with a green bar indicating +95,870 and -8,270 lines. The bottom of the interface shows navigation options like 'Conversation', 'Commits', 'Checks', and 'Files changed', along with a 'Review changes' button.

openjdk / jdk

<> Code Pull requests 299 Security Insights

8284161: Implementation of Virtual Threads (Preview)

#8166

Closed AlanBateman wants to merge 23 commits into `openjdk:master` from `AlanBateman:JDK-8284161`

Conversation 230 Commits 23 Checks 0 Files changed 1,133 +95,870 -8,270

Changes from all commits File filter Conversations Jump to 0 / 1133 files viewed Review changes

Detalles de implementación

Thread.*sleep*

```
public static void sleep(long millis) throws InterruptedException {  
    if (millis < 0) {  
        throw new IllegalArgumentException("timeout value is negative");  
    }  
    long nanos = MILLISECONDS.toNanos(millis);  
    ThreadSleepEvent event = beforeSleep(nanos);  
    try {  
        if (currentThread() instanceof VirtualThread vthread) {  
            vthread.sleepNanos(nanos);  
        } else {  
            sleep0(nanos);  
        }  
    } finally {  
        afterSleep(event);  
    }  
}
```



Frameworks

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Probablemente no tengas que escribir código para hacer uso de los virtual threads:

- Spring: `spring.threads.virtual.enabled=true`
- Quarkus: `@RunOnVirtualThread`



Más de project Loom

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Más de project Loom

- Structured Concurrency (Preview) [JEP 453](#)
- Scoped Values (Preview) [JEP 446](#) (second preview en Java 22: [JEP 464](#))



Ejemplo

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom



Coding time!



Preguntas?

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom

Preguntas?



Referencias

1 Acerca de mí

2 Un poco de historia

- Historia de la multitarea en Java
- El problema con los threads nativos
- La solución reactiva

3 Propiedades de los Virtual Threads

- Ventajas
- Desventajas
- Como se usan
- Que hacer
- Que no hacer

4 Detalles de implementación

- Pull request
- Thread.*sleep*

5 Frameworks

6 Más de project Loom

Referencias

- [Jep] *JEP 444: Virtual Threads*. 2023. URL: <https://openjdk.org/jeps/444> (visitado 27-11-2023).
- [Ora04] Oracle. *Concurrency Utilities Overview*. 2004. URL: <https://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/overview.html> (visitado 27-11-2023).
- [Pau23] José Paumard. *Java 21 new feature: Virtual Threads, RoadTo21*. 2023. URL: <https://youtu.be/5E0LU85EnTI?si=pX3tL808G2AsT0wK> (visitado 01-12-2023).