O Java vêm se modificando bastante ao longo dos anos afim de atender novas demandas e necessidades que nós desenvolvedores encontramos.

O Project Loom é mais uma dessas mudanças de paradigmas mais desejadas pelos profissionais da área. Em essência, o objetivo principal do Project Loom é oferecer suporte a um modelo de simultaneidade leve e de alto rendimento em Java.

Antes de entrar mais profundamente nessa questão é preciso entender primeiro como uma Thread funciona.

Imagine uma thread como uma sequencia de comandos sendo executados em um programa ou processo. Perceba que executar o mesmo programa duas vezes não é criar mais threads e sim criar dois processos do mesmo programa. Threads rodam concorrentemente em um mesmo processo. Processos executam concorrentemente em um sistema operacional.

entanto, como o Java usa os threads do [kernel](https://www.baeldung.com/cs/os-kernel) do [sistema operacional](https://www.baeldung.com/cs/os-kernel) para a implementação, ele falha em atender aos requisitos atuais de simultaneidade. Existem dois problemas principais em particular:

* As threads  não podem corresponder à escala da unidade de simultaneidade do domínio. Por exemplo, os aplicativos geralmente permitem até milhões de transações, usuários ou sessões. No entanto, o número de threads com suporte pelo kernel é muito menor. Portanto, uma Thread para cada usuário, transação ou sessão geralmente não é viável
* A maioria das aplicações simultâneas precisam de alguma sincronização entre threads para cada solicitação. Por conta disso, uma troca de contexto custosa acontece entre as threads do sistema operacional.

Por exemplo, considere o encadeamento de uma aplicação que executa alguma ação nas solicitações e, em seguida, passa os dados para outro encadeamento para processamento posterior. Aqui, seria melhor agendar essas duas threads na mesma CPU. Mas, como o escalonador é agnóstico em relação ao thread que solicita a CPU, isso é impossível de garantir.

O Project Loom propõe resolver isso por  meio de threads “user-mode” que contam com a implementação de continuações e “schedulers” em tempo de execução Java em vez da implementação do sistema operacional .

FIBERS: Nos protótipos recentes do OpenJDK, uma nova classe chamada  Fiber é introduzida na biblioteca junto com a  classe Thread .

Como a biblioteca planejada para Fibers é semelhante a Thread , a implementação do usuário também deve permanecer semelhante. No entanto, existe uma diferença principal: O Fiber  iria  condensar qualquer tarefa em d uma continuação do user-mode interno. Isso permitiria que a tarefa fosse suspensa e retomada no tempo de execução Java em vez do kernel.

Uma continuação (ou co-rotina) é uma sequência de instruções que pode parar e ser retomada pelo chamador em um estágio adiante.

Cada continuação tem um ponto de entrada e um ponto de rendimento. O ponto de rendimento é onde ele foi suspenso. Sempre que o “caller” retoma a continuação, o controle retorna ao último ponto de rendimento.

É importante notar **que essa suspensão/retomada agora ocorre no tempo de execução da linguagem, e não no sistema operacional** . Portanto, evita a dispendiosa troca de contexto entre os encadeamentos do kernel.

Semelhante as Threads, o Project Loom visa oferecer suporte a “fibers” aninhados. Uma vez que os “fibers” dependem de continuações internamente, ela também deve oferecer suporte a continuações aninhadas.

Para entender isso melhor, considere uma classe  “Continuation”  que permite o aninhamento:

Continuation cont1 = **new** Continuation(() -> {

Continuation cont2 = **new** Continuation(() -> {

//do something

suspend(SCOPE\_CONT\_2);

suspend(SCOPE\_CONT\_1);

});

});

Conforme mostrado acima, a continuação aninhada pode suspender a si mesma ou a qualquer uma das continuações que estejam envolvidas, passando uma variável de escopo . Por esse motivo, eles são conhecidos como “Continuations” com escopo .

Nome: Elvis Moises Melo de Oliveira / Matricula 202002410931

Fontes: <https://cr.openjdk.java.net/~rpressler/loom/Loom-Proposal.html>

<https://www.youtube.com/watch?v=-CPWbB-Pn14>

<https://www.baeldung.com/openjdk-project-loom>