Tìm hiểu benchmark 0.5, test thử xem tốc độ

1. **Giải thuật Fork / Join**

Trong lĩnh vực tính toán song song, mô hình xử lí Fork-Join là 1 phương pháp để xây dựng và thực thi các chương trình 1 cách song song đồng thời.

Quá trình xử lí được chia thành các nhánh song song tại các điểm được chỉ định trong chương trình (Fork) và sau đó thì “hợp” (Join) lại tại 1 điểm và lại tái thực hiện quá trình này 1 cách tuần tự tiếp tục . Các vùng subtask song song kể trên có thể lại tiến hành chia ra (Fork) tiếp ( kiểu đệ quy ) cho tới khi các subtask con của chúng trở nên đủ nhỏ để xử lí theo kiểu bất đồng bộ .

Fork-Join có thể được coi là 1 design pattern kiểu song song.

A picture containing object

Description automatically generated

Vd như hình minh họa trên: Chương trình chạy trên 1 master Thread và để xử lí nó người ta phân nó thành 3 Task chạy song song với nhau. Trong đó mỗi task lại được chia nhỏ thành các task con (subtask) để xử lí. Các điểm chỉ định để “chia” (Fork) và “hợp” (Join) được mô tả bằng các dấu chấm đen.

Trong JAVA nói riêng, thì triển khai mô hình giải thuật này thì từ Java 7 đã thêm vào Fork/Join Framework. Nó cung cấp các công cụ giúp tăng tốc xử lý song song bằng cách cố gắng sử dụng tất cả các lõi bộ xử lý có sẵn, được thực hiện thông qua cách tiếp cận phân chia (fork) và gộp (join) task. Mục đích là để sử dụng tất cả các khả năng xử lý để nâng cao hiệu suất cho các ứng dụng.

Để cung cấp thực hiện xử lý song song hiệu quả, các fork / join Framework sử dụng hồ chứa (pool) các Thread được gọi là ForkJoinPool.

ForkJoinPool tương tự như Java ExecutorService nhưng với một sự khác biệt. ForkJoinPool phân chia các tác vụ cho các luồng thực thi trong Thread Pool. Framework Fork/ Join sử dụng thuật toán work-stealing. Các luồng sẽ thực thi công việc của mình trên một bộ xử lý riêng biệt (thread/ processor), khi làm hết việc của mình, nó lấy bớt (steal) các tác vụ từ các luồng khác đang bận rộn. Việc lấy bớt tác vụ từ luồng khác được gọi là “Work Stealing”

Hiểu 1 cách nôm na, work stealing là cơ chế giúp scheduler ( có thể là trên ngôn ngữ, hoặc OS) có thể thực hiện việc tạo thên M thread mới hoạt động mượt mà trên N core, với M có thể lớn hơn N rất nhiều.

Ý tưởng của work-stealing scheduler đóng vai trò như 1 quản lí phân cho mỗi một core sẽ có một queue những task phải làm. Mỗi task đó bao gồm một list các instructions phải thực hiện một cách tuần tự. Khi một processor làm hết việc của mình, nó sẽ nhìn ngó sang các processor xung quanh, xem có gì cần làm không và “steal” công việc từ đó.

Một mô hình khác với work stealing là work sharing, tức là mỗi task sẽ quyết fix là sẽ được thực hiện trên processor nào.

1. **LocalDate/Time trong Java 8 so với Java 7**

So với các phương thức liên quan đến thời gian trong java 7 ( Date & Calender ) thì 2 phương thức LocalDate/Time của java 8 có các ưu điểm hơn như sau:

+ Mặc định Date trong java 7 thì hiện thì thời gian hiển thì bao gồm cả ngày giờ năm tháng ngày. Trong khi java 8 thì tách biệt rõ rang nó với việc LocalDate thì chỉ hiện thị ngày tháng năm còn Time thì đảm nhiệm thời gian giờ phút giây, ngoài ra chúng có 1 dạng kết hợp với nhau là LocalDateTime để hiển thị đầy đủ được tất cả thông tin giống như Date làm được.

+ Date trong java 7 thì không có TimeZone cũng như không có format (phải format thông qua một phương thức khác vd: SimpleDateFormat() ) rất phức tạp, trong khi đó java 8 thì việc này dễ dàng hơn nhiều.

+ Việc thao tác với giá trị thời gian như cộng thêm, trừ bớt thì Date trong java 7 không hề có và chỉ có từ java 8 trở đi => việc xử lí thông tin ngày tháng trở nên thuận tiện hơn rất nhiều. Ngoài ra khả năng truyền vào chuỗi chứa thời gian String với phương thức parse cũng là 1 tiện ích mà java 8 có mà java 7 không có.

// Lấy thông tin về ngày tháng hiện tại ( đã gồm cả TimeZone )

final LocalDate today = LocalDate.now();

// Hàm thao tác cộng thời gian vd : ngày

LocalDate tomorrow = LocalDate.now().plusDays(1);

// Lấy thông tin về giờ phút giây hiện tại ( đã gồm cả TimeZone )

final LocalTime now = LocalTime.now();

// Lấy thông tin về thời gian tổng hợp

final LocalDateTime datetime = LocalDateTime.now();

// Tạo biến time chứa thông tin thời gian đã được format sẵn dễ dàng hơn

final LocalDateTime time = LocalDateTime.of( 2014, Month.APRIL, 16, 0, 0, 0 );

// Tạo ZoneId và triển khai dễ hiểu hơn so với java 7

ZoneId losAngles = ZoneId.of("America/Los\_Angeles");

LocalDateTime dateTime = LocalDateTime.of(2016, 05, 25, 12, 0);

ZonedDateTime losAnglesDateTime = ZonedDateTime.of(dateTime, losAngles);