**TASK 7**

**I/ Apache JMeter**

1. **JMeter là gì? Tại sao phải dử dụng**

**Đặt vấn đề**: Giả sử bạn cần kiểm tra năng suất của 1 trang web với 100 người dùng. Việc bố trí 100 máy PC và truy cập internet cùng một lúc => không khả thi (nếu yêu cầu là 1000 người dùng => càng không khả thi). Do đó, JMeter ra đời để mô phỏng hành vi của người dùng thực và kiểm tra hiệu suất/tải trang web.

Apache JMeterTM là ứng dụng Java mã nguồn mở, được thiết kế để tải thử nghiệm hành vi chức năng và đo lường hiệu suất. Có thể dùng JMeter để phân tích và đo lường hiệu suất của ứng dụng web (nghĩa là thử nghiệm 1 ứng dụng web với tải nặng, lưu lượng truy cập nhiều người dùng đồng thời) hoặc nhiều dịch vụ khác. Ngày nay, còn được dùng để kiểm tra chức năng, kiểm tra database server, ...

**Ưu điểm**

* JMeter hoàn toàn miễn phí, cho phép nhà phát triển sử dụng mã nguồn để phát triển
* GUI thân thiện => cực kỳ dễ sử dụng => tránh mất thời gian
* JMeter là ứng dụng máy tính để bàn Java thuần túy 100% => có thể chạy trên nhiều nền tảng
* Khung đa luồng đầy đủ: Cho phép đồng thời lấy mẫu các chức năng khác nhau bằng một nhóm luồng riêng biệt
* Trực quan hóa kết quả kiểm tra: Kết quả kiểm tra có thể được hiển thị ở định dạng khác như biểu đồ, bảng, cây và tệp nhật ký
* Dễ dàng cài đặt: Chỉ cần copy và chạy file \*.bat để chạy JMeter. Không cần cài đặt.
* Có khả năng mở rộng cao: Có thể viết bài kiểm tra của riêng bạn. JMeter cũng hỗ trợ các plugin trực quan cho phép bạn mở rộng thử nghiệm của mình
* JMeter hỗ trợ nhiều chiến lược thử nghiệm như Kiểm tra Tải, Thử nghiệm phân phối và Thử nghiệm chức năng.
* JMeter có thể mô phỏng nhiều người dùng với các luồng đồng thời, tạo tải nặng cho ứng dụng web đang được thử nghiệm
* Hỗ trợ đa giao thức, ví dụ như: HTTP, JDBC, LDAP, SOAP, JMS và FTP
* Ghi và phát lại – Kỷ lục hoạt động của người dùng trên trình duyệt và mô phỏng chúng trong ứng dụng web bằng cách sử dụng JMeter
* Kiểm tra tập lệnh: Jmeter có thể được tích hợp với Bean Shell & Selenium để thử nghiệm tự động.

1. **Cài đặt Jmeter**

* Cài đặt java và tải jdk
* Tải Apache JMeter (tệp binaries có thể dưới dạng zip, tgz) tại đây:

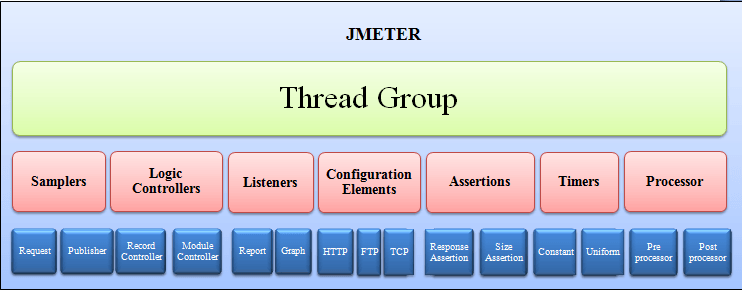
https://jmeter.apache.org/download\_jmeter.cgi

* Giải nén tệp vừa tải => thành công
* Chạy JMeter:
* Chế độ GUI (tiêu tốn tài nguyên): Chạy tệp \bin\jmeter.bat
* Chế độ server: Chạy tệp \bin\jmeter-server.bat
* Chế độ dòng lệnh: Trỏ cmd đến thư mục \bin rồi nhập lệnh sau

*jmeter -n -t testPlan.jmx - l log.jtl -H 127.0.0.1 -P 8000*

* Dựa trên yêu cầu, bạn có thể cần một hoặc nhiều gói bổ sung tùy chọn

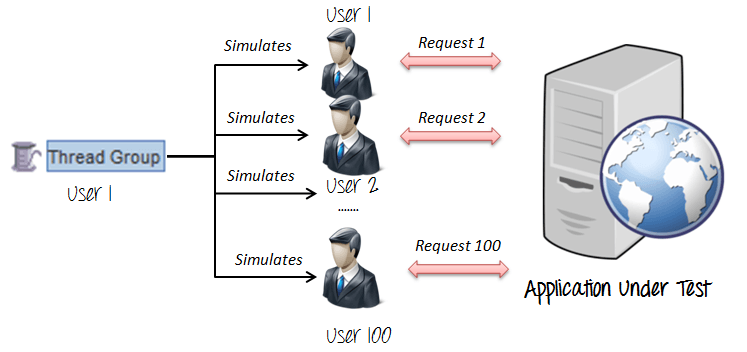
1. **Các thành phần của JMeter**



* Thread group

Là một tập hợp các thread. Mỗi thread đại diện cho một người dùng đang sử dụng ứng dụng đang được thử nghiệm và mô phỏng một yêu cầu thực của người dùng tới máy chủ. Các điều khiển cho một nhóm luồng cho phép bạn đặt số lượng luồng cho mỗi nhóm.

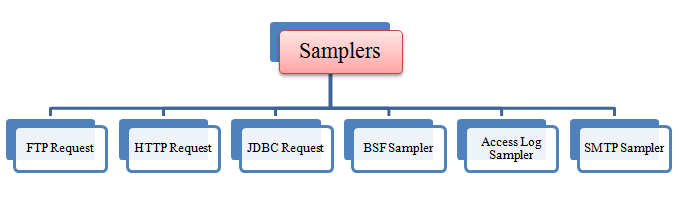
Ví dụ: nếu bạn đặt số lượng threads là 100; JMeter sẽ tạo và mô phỏng 100 yêu cầu của người dùng đến máy chủ đang được thử nghiệm



* Samplers

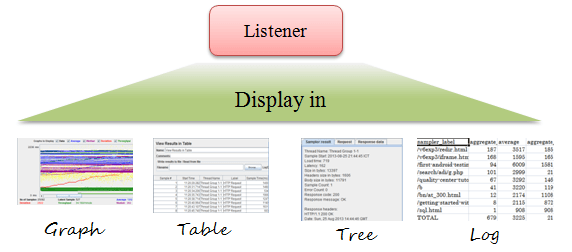
Làm sao để thread group biết loại giao thức nào mà nó cần thực hiện? => samplers

Yêu cầu của người dùng có thể là FTP request, HTTP request, JDBC request, ...



* FPT request
* HTTP request: Cho phép gửi HTTP/HTTPS request đến máy chủ web
* JDBC request: Cho phép thực thi cơ sở dữ liệu kiểm tra năng suất. Nó gửi JDBC request (truy vấn SQL) tới cơ sở dữ liệu.
* Listener

Cho phép hiển thị kết quả thực hiện kiểm thử. Có thể hiển thị kết quả ở định dạng khác như cây, bảng, biểu đồ hoặc tệp nhật ký



# **II/ Hai kịch bản**

1. **API cộng tiền**

**Lỗi:**

* Nhiều request cộng tiền vào 1 user nhưng số tổng số tiền thu được vị sai lệch

**Giải pháp:**

* Đảm bảo rằng chỉ có một request cộng tiền được thực hiện tại một thời điểm => Sử dụng mutex để đồng bộ hóa các request, tránh xung đột (Sử dụng Lock)
* Thay vì thực hiện thao tác cộng tiền trong source code rồi update vào database thì chuyển thao tác cộng tiền đó sang cho database xử lý rồi update (nghĩa là thao tác cộng được thực hiện đồng thời trong câu lệnh UPDATE)

Ví dụ:

Thay vì sử dụng:

*UPDATE user SET money = ? WHERE id = ?*

Thì sử dụng:

*UPDATE user SET money = money + ? WHERE id = ?*

1. **API chuyển tiền**

**Lỗi:**

* Khi gửi 300 request chuyển tiền từ userA cho userB cùng lúc, với số tiền của userA không đủ cho 300 request. Nhưng kết quả là userB vẫn nhận được đầy đủ số tiền và số tiền userA còn lại là 1 số âm => lỗi
* Khi gửi 2 request userA gửi cho tiền cho userB và userB gửi tiền cho userA cùng lúc, số tiền của 2 user nhận được là không đúng => lỗi

**Giải pháp:**

Đảm bảo rằng chỉ có một request chuyển tiền được thực hiện tại một thời điểm => Sử dụng mutex để đồng bộ hóa các request, tránh xung đột (Sử dụng Lock)

|  |  |
| --- | --- |
| Synchronized | ReentrantLock |
| Chỉ dùng cho cấp độ phương thức hoặc khối mã | Tùy ý |
| Không thể thoát khỏi khóa một cách tường minh, chỉ khi khối synchronized kết thúc. | Có thể thoát khỏi khóa một cách tường minh bằng cách gọi unlock(). |
| Sử dụng chiến thuật NonfairSync | ReentrantLock có 2 chiến thuật:  - FairSync: đảm bảo thứ tự, luồng nào chờ lâu nhất thì sẽ lấy được lock  - NonfairSync (default): không đảm bảo thứ tự, bất cứ luồng nào đang chờ đều có thể lấy được lock. |
| Hiệu suất tốt hơn cho các trường hợp đơn giản | Hiệu suất tốt hơn cho các trường hợp phức tạp như quản lý đợi, ưu tiên, v.v. |
| Có thể gây ra deadlock, hiệu suất thấp khi 2 hay nhiều luồng đang chờ đợi nhau để giải phóng các khóa (lock) mà chúng đang nắm giữ dẫn đến việc các luồng bị treo vô thời hạn và không thể tiếp tục thực thi | Cho phép cùng một luồng giữ khóa nhiều lần và đờc tùy chỉnh thời gian chờ => tránh bị deadlock |

**Sử dụng ConcurrentHashMap**

* ConcurrentHashMap cho phép nhiều luồng có thể truy cập, sửa đổi map cùng một lúc mà không ảnh hưởng đến tính nhất quán
* Mỗi 1 request sẽ có 1 lock duy nhất, không dùng chung lock giống như ReentrantLock
* Các request không phải chờ đợi khóa của nhau mà chỉ đợi khóa của chính nó => giảm thời gian chờ đợi
* Nếu lock bị 1 thread khác nắm giữ, thread khác phải chờ đợi cho đến khi giải phóng lock
* Đảm bảo 1 thread truy cập 1 key tại 1 thời điểm

Pessimistic lock: Sử dụng khi có nhiều luồng cùng truy cập và gây xung đột dữ liệu rất cao nhưng vẫn đảm bảo tính nhất quán dữ liệu

Optimistic lock: Sử dụng khi xung đột dữ liệu thấp, chấp nhận việc giao dịch bị lặp lại hoặc bị hủy bỏ và không yêu cầu tính nhất quán dữ liệu ở mức độ cao.

Trong bài toán này, em sẽ sử dụng pessimistic locking của database, bằng cách thực hiện khóa bản ghi được truy vấn, ngăn chặn các transaction khác truy cập và thay đổi bản ghi đó cho đến khi transaction hiện tại được commit.

Về việc áp dụng cho số lượng lớn user:

* Với trường hợp các request chuyển tiền độc lập giữa các user (user A -> user B, user C -> user D, ...): Hiển nhiên đây là các bản ghi độc lập và có thể xử lý đồng thời
* Với trường hợp các request chuyển tiền không độc lập (user A -> user B, user A -> user C): Rõ ràng với permisstic lock đã khóa bản ghi lại và ngăn chặn các truy cập khác. Request nào vào trước thì được xử lý trước.