Tool to Visualize Thread Pool

Han Yongjun[1], Kim Kyeonghyeon[2], Yoon Jaehwan[3], and Kin Junyoung[4]

[1]Sungkyunkwan University, Department of Civil Engineering, Computer Science Engineering
[2]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering
[3]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering
[4]Sungkyunkwan University, Department of Mathmatics, Computer Science Engineering

October 3, 2024

Abstract

Monitoring and visualizing the performance of multi-threaded applications is critical to understanding resource utilization and system behavior. Existing tools provide comprehensive solutions for capturing and analyzing system metrics, but often fail to provide detailed insights into individual thread performance, especially in terms of memory usage. This report presents the development of a new visualization tool designed to improve existing monitoring approaches. This graphical approach allows users to intuitively monitor real-time memory usage and thread status.

1 Introduction

In today's rapidly evolving technological landscape, the efficient management and monitoring of application performance have become paramount. With the increasing complexity of software architectures and the necessity for real-time data analysis, developers are seeking innovative solutions that provide actionable insights into their applications. In South Korea, Java holds a significant market share, and Spring Framework plays a dominant role in the backend development landscape. Approximately 60 percent of Java developers use Spring as the primary framework for their main applications ans Spring run upon Tomcat server which is based on Thread Pool Skill. This proposal outlines a cutting-edge tool designed to enhance the monitoring capabilities of Spring Boot applications through real-time thread pool visualization.

The tool leverages Spring Actuator, a powerful module that exposes operational information about the application, combined with D3.js, a robust JavaScript library for producing dynamic, interactive data visualizations. By integrating these technologies, the proposed solution enables developers to monitor critical metrics, such as memory usage and thread activity, in real-time. This not only facilitates a deeper understanding of application behavior but also aids in identifying performance bottlenecks and optimizing resource utilization.

2 Background

2.1 Thread Pool

Thread pool is a system that manags a collection of pre-created threads. Ir offers several advantages: 1) Reduces the overhead of creating new threads. 2) Prevents system overload by limiting the number of concurrently running threads. 3) Ensures efficient resource management and faster task processing

2.2 Spring Boot Actuator

The Spring Boot Actuator is production-ready feature to help monitor Spring Boot applications. It offers various endpoints that can be used to fetch internal information via HTTP requests. The endpoints are available only when they are set to be enabled and exposed. There are numerous built-in endpoints such as /health, /info, and/metrics. /metrics is the endpoint responsible for metrics including memory usage, CPU usage, HTTP request, and so on which is usually configured with Micrometer to support observability systems such as Elastic, Graphite, or Prometheus. Regarding thread pool monitoring, actuator provides built-in endpoints with metrics only at aggregate level such as maximum threads(tomcat.config.max), active threads(tomcat.busy), and idle threads(tomcat.idle). Fine-grained or per-thread monitoring requires customized endpoint.

Regarding thread pool monitoring, actuator provides built-in metrics only at aggregate level such as maximum threads(tomcat.config.max), active threads(tomcat.busy), and idle threads(tomcat.idle). Fine-grained or per-thread monitoring requires customized endpoint.

2.3 Dispatcher Servlet

The Dispatcher Servlet is a central component of the Spring MVC architecture, acting as the front controller that handles incoming client requests. When a client sends an HTTP request to the server, it is received by the Dispatcher Servlet, which is responsible for directing the request to the appropriate handler (controller) and returning the response back to the client.

2.4 Handler Interceptor

HandlerInterceptor is an interface that allows developers to intercept requests and responses during the processing lifecycle managed by the Dispatcher Servlet. It provides hooks for executing additional tasks at various points in the request handling process.

Key methods of Handler Interceptor.

preHandle(). This method is called before the request is processed. It can be used to validate requests or check authentication informtion. If this method returns false, the request handling is aborted.

postHandle(). This method is invoked after the handler is ecevuted but before the view is rendered. It can be used to midofy data or perform additional processing.

afterCompletion(). This method is called after the request handling and view rendering are complete. It is useful for cleaning up resources or logging.

3 Problem Statement

Monitoring and visualizing the performance of multi-threaded appli- cations is critical to understanding resource utilization and system be- havior. Existing tools provide comprehensive solutions for capturing and analyzing system metrics, but often fail to provide detailed insights into individual thread performance, especially in terms of memory usage. This report presents the development of a new visualization tool designed to improve existing monitoring approaches. This graphical approach allows users to intuitively monitor real-time memory usage and thread status.

4 Related Work

We analyzed 3 popular tools to monitor backend server and there drawbacks and comparison with our solutions are below. Be aware that drawbacks of these tools are beacuse of various kinds of matrics. On the other hands, we concentrated on each threads in thread pool.

4.1 Prometheus

Overall thread flow is known, but lack of detailed thread level metrics, such as below

Prometheus	Our solution
No memory usage of each thread each thread.	We provide memory usage of
Must be associated with other visualization programs.	There is no need for external programs.

Table 1: Comparison with Prometheus

4.2 Data Dog

Data Dog	Our Solution
Heavily priced plans for large scale	Free

Table 2: Comparison with Data Dog

4.3 Graphite

Graphite	Our Solution
Constraints in real-time monitoring and alerting. and alerting related functions	Serve data as a real-time
Steep Learning Curve for Queries	Doesnt need any learnings.

Table 3: Comparison with Graphite

5 Proposed Solution

Our solution aims to address the complexity and interpretability challenges of existing monitoring and visualization tools like Grafana and Prometheus. While these tools offer a wide range of data, the information can sometimes be overwhelming and difficult to interpret. The key differentiator of our solution compared to existing ones is its focus on simplifying complex data for easier interpretation. This ensures that not only developers but also non-experts can easily understand the state of the system.

5.1 Environment

The tool is designed to be easily integrated into existing Spring applications. It offers customizable settings for data collection intervals, thread pool monitoring scope, and visualization parameters.

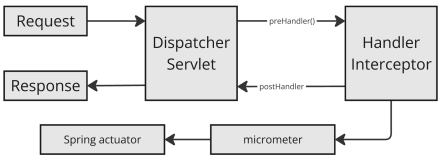

Figure 1: Data Process Flow

5.2 Data Collection

This implementation outlines the creation of a tool that collects real-time HTTP request and response data, tracks memory usage of the threads handling these request, and visualize the data in a Spring Boot application. By utilizing HandlerInterceptor, Spring Actuator, and Micrometer, the solution provides real-time insights into thread behavior and memory usage in response to specific HTTP requests. We will override key methods of Handler Interceptor to get information about request, memory usage of each threads and response data. This utilized methods are deployed as library and any Spring project owner can use it.

5.3 Visualization

Each thread is visualized as a circle in a graphical interface. The size and color of the circles represent key metrics, such as memory usage and thread status. This visual representation provides a quick, at-a-glance understanding of the thread pool's health.

6 Planning in Detail

6.1 Subtask

Our project contains two parts largely. One is data collection part and the other is web page part. In data collection part, we will implement the methods using Spring Framework. The roles are distributed into two parts which all members will be working together. For web page, we will use thymeleaf. Thymeleaf is a modern server-side Java template engine for both web and standalone environments. With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development.

6.2 Role Distribution

Name	Role
Han Yongjun	Data Collection, visualization
Kim Kyeonghyeon	Data Collection, visualization
Yoon Jaehjwan	Data Collection, visualization
Kim Junyoung	Data Collection, visualization

Table 4: Role Assignment

6.3 Development Plan

The project is planned to 7 parts: develop data collecting methods, deployed as a library, develop web application, UI/UX Design, and testing.

Weeks	2 3	4 5	6 7	8 9	10 11	12 13	14 15
Define Problem	O O	O					
Tech analysis and Study		O O	O				
Implement data collecting methods			O O				
deploy as library				O .			
develop web application				. O	O O	O	
UI/UX Design						O O	
Testing							O

Table 5: Weekly Schedule

7 References

7.1 Official Spring web site - https://docs.spring.io/spring-framework/reference, https://docs.spring.io/spring-boot/reference/actuator

7.2 Official Prometheus web site - https://prometheus.io

7.3 Official Datadog website - https://www.datadoghq.com

7.4 https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/