Rotating Index Load Balancer Pre-Production Specification

Lani Wagner
lani.wagner@students.fhnw.ch

2022-03-17 10:29:24

Abstract

This document contains the pre-production specificatino written for an implementation of a rotating index load balancer to reduce the I/O load for a service that is tasked with handling a large amount of requests, so that the I/O bottleneck is greatly reduced. This specification was written for the FHNW module Engineering Writing (engw).

Contents

1 General

the result of this specification will be a library. this document documents the features this library must provide. these specification is purposefully *not* programming-language-specific to allow the implementation in any language that best suits the existing infrastructure. The terminology in this document will be leaning on the terms most often used in while working with the java programming-language, such as type definitions. When working in another language the appropriate equivalent must be used in consideration of applicability and performance.

2 Design

The library shall provide an instantiable Singleton[3] class named `RotationBalancer<T>` with the following arguments. The required arguments must be implemented in the final product, but the passing of the arguments for the creation of the object is not required. The type `T` or `<T>` is a generic type that is set for every analysis when that request is made.

Required	**Type**	**Name**	**Content**
Yes	`List<Path>`	`folders`	The folders that should be scanned.
No	`List<Path>`	`excludedFolders`	Subfolders of `folders`[1] that don't need to be scanned.
No	`Predicate<Path>`	`isFileRelevant`	Executable function to verify whether the file is relevant[2] for the analysis.

Table 1: Library class arguments.

The instance shall only scan for files as long as active requests are being processed. This means that the scan should not start as soon as the class is instantiated. If the last request has been completed the balancer shall go into a sleeping state and woken up when a new request arrives.

Every request to the API must be sent to an existing instance. That means instantiating the object before starting any analysis is critical. Once the object exists the user must be able to call the API with the following arguments and receive a response of type `List<T>`.

Required	**Type**	**Name**	**Content**
Yes	`Function<Path, T>`	`analyzer`	The analysis function applied to every relevant[2] file.
No	`List<Path>`	`excludedFolders`	Subfolders of `folders`[1] that don't need to be scanned.
No	`Predicate<Path>`	`isFileRelevant`	Executable function to verify whether the file is relevant[2] for the analysis.

Table 2: Library analysis call arguments.

[1] This must be verified when the class instance is created.

[2] This f.e. allows for the exclusion of irrelevant files that may be stored in the same location. If no function is passed every file must be treated as relevant.

3 Functional and Non-Functional Requirements

3.1 Handling Unpredictability

3.2 Logging

3.3 Implementation and Testing

3.4 Documentation

4 Glossary

Term	Definition
Singleton	A class that can only exist once in memory. Trying to create a new instance of the class should return the existing instance.

Table 3: Glossary definitions.

List of Figures

List of Tables