Caffeine Cache in Java Application

Srikanth Dannarapu · Follow
Published in Javarevisited · 5 min read · Jun 19, 2023

Caffeine is a high-performance, in-memory caching library developed by Ben Manes. It provides a fast and efficient implementation of a key-value cache, designed to optimize read and write operations.

Caffeine cache offers the following features:

1. Efficient memory usage: Caffeine intelligently manages memory by using various eviction strategies, such as LRU (Least Recently Used), LFU (Least Frequently Used), and more.
2. High-performance: Caffeine is designed to deliver fast read and write operations, making it suitable for use cases where performance is critical.
3. Concurrency support: Caffeine provides built-in support for concurrent access, allowing multiple threads to read and write to the cache simultaneously while maintaining data consistency.
4. Flexible configuration: Caffeine offers a wide range of configuration options, allowing you to customize cache size, eviction policies, expiration times, and more based on your specific needs.

Regarding Spring's support for Caffeine, starting from Spring Framework 4.3 and Spring Boot 1.4, Caffeine cache integration is available as a caching provider. Spring provides an abstraction layer for caching, allowing you to use various caching implementations, including Caffeine, with minimal configuration.

To use Caffeine cache in Spring, you need to include the Caffeine dependency in your project, as mentioned in the previous response. Then, you can configure and enable caching in your Spring Boot application using the `@EnableCaching` annotation. Spring's `@Cacheable` and `@CachePut` annotations can be used to cache and retrieve data from the Caffeine cache, respectively.

By integrating Caffeine cache with Spring, you can leverage its performance and memory management capabilities to improve the efficiency of your application's caching layer.

To implement Caffeine cache using `@Cacheable` and `@CachePut` annotations in Spring Boot, you need to follow these steps:

Step 1: Add Caffeine as a dependency Add the Caffeine cache dependency to your project. You can do this by adding the following dependency to your `pom.xml` if you're using Maven:

```
<dependency>
    <groupId>com.github.ben-manes.caffeine</groupId>
    <artifactId>caffeine</artifactId>
    <version>3.0.4</version>
</dependency>
```

Step 2: Enable caching in your Spring Boot application Enable caching in your Spring Boot application by adding the `@EnableCaching` annotation to your main application class. This annotation enables Spring's caching infrastructure to be aware of your cache annotations.

```
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@EnableCaching
public class YourApplication {
    public static void main(String[] args) {
        SpringApplication.run(YourApplication.class, args);
    }
}
```

Step 3: Define a cache configuration Create a configuration class to define the Caffeine cache manager bean. This class will specify the cache names and their configurations.

```
import org.springframework.cache.CacheManager;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.caffeine.CaffeineCacheManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.github.benmanes.caffeine.cache.Caffeine;

@Configuration
@EnableCaching
public class CacheConfig {
    @Bean
    public CacheManager cacheManager() {
        CaffeineCacheManager cacheManager = new CaffeineCacheManager("myCache");
        cacheManager.setCaffeine(caffeineCacheBuilder());
        return cacheManager;
    }

    Caffeine<Object, Object> caffeineCacheBuilder() {
        return Caffeine.newBuilder()
                .initialCapacity(100)
                .maximumSize(500)
                .expireAfterWrite(Duration.ofMinutes(10));
    }
}
```

Step 4: Use `@Cacheable` and `@CachePut` annotations Now you can use the `@Cacheable` and `@CachePut` annotations in your service or repository classes.

```
import org.springframework.cache.annotation.Cacheable;
import org.springframework.cache.annotation.CachePut;
import org.springframework.stereotype.Service;

@Service
public class MyService {

    @Cacheable("myCache") // Specify the cache name
    public String getCachedData(String key) {
        // This method will be cached
        // Implement your logic here to retrieve the data
        return "Data from cache";
    }

    @CachePut("myCache") // Specify the cache name
    public String updateCachedData(String key, String newData) {
        // This method will update the cache with new data
        // Implement your logic here to update the data
        return newData;
    }
}
```

In the above example, the `getCachedData` method will be cached using the specified cache name "myCache". Subsequent calls with the same key will retrieve the data from the cache if it exists.

The `updateCachedData` method will update the cache with the new data provided and return the updated data.

Remember to adjust the cache configurations in the `caffeineCacheBuilder` method according to your specific requirements.

That's it! You have now implemented Caffeine cache using `@Cacheable` and `@CachePut` annotations in Spring Boot.

Caching management endpoint

To enable the caching management endpoint in a Spring Boot application, you can follow these steps:

1. Add the necessary dependencies to your project's build file (e.g., Maven's `pom.xml` or Gradle's `build.gradle`):

For Maven:

```xml
<dependencies>
    <!-- Other dependencies -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

For Gradle:

```groovy
dependencies {
    // Other dependencies
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
}
```

The `spring-boot-starter-actuator` dependency provides the necessary classes and endpoints for managing various aspects of your Spring Boot application, including the cache management endpoint.

2. Configure the cache management endpoint in your `application.properties` or `application.yml` file:

For `application.properties`, add the following property:

```
management.endpoints.web.exposure.include=caches
```

For `application.yml`, add the following configuration:

```yaml
management:
  endpoints:
    web:
      exposure:
        include: caches
```

This configuration exposes the cache management endpoint (`/actuator/caches`) as part of the actuator endpoints.

3. Optionally, you can further customize the cache management endpoint by configuring additional properties. For example, you can set cache-specific properties or exclude certain caches from being exposed. Here's an example using `application.properties`:

```
management.endpoints.web.exposure.include=caches
management.endpoint.caches.cache-configs[0].cache-name=cache1
management.endpoint.caches.cache-configs[0].statistics-enabled=true
management.endpoint.caches.cache-configs[1].cache-name=cache2
management.endpoint.caches.cache-configs[1].statistics-enabled=false
```

In this example, `cache1` and `cache2` are specified as cache names, and their statistics are enabled and disabled, respectively.

4. Run your Spring Boot application, and the cache management endpoint should be available at the following URL:

```
http://localhost:8080/actuator/caches
```

Note: The actual base URL may vary based on your application's configuration and port number.

You can now access the cache management endpoint to monitor and manage your caches. Remember to secure the actuator endpoints in a production environment to prevent unauthorized access.

here is a complete example of using caffeine cache

GitHub - srikanthdannarapu/spring-boot-caffeine-cache

Contribute to srikanthdannarapu/spring-boot-caffeine-cache development by creating an account on GitHub.

github.com

Below diagram that illustrates the flow of the above github example

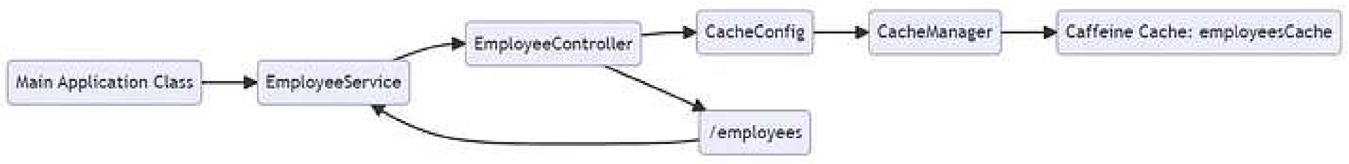


Diagram illustrates the flow of control and data in the caching example using Spring Boot and Caffeine cache

- The main application class (`Main Application Class`) initializes the Spring Boot application.
- The `EmployeeService` (`EmployeeService`) is a service class responsible for fetching employees. It is called by the `EmployeeController`.

- The `EmployeeController` (`EmployeeController`) is a REST controller that exposes the `/employees` endpoint for fetching employees.
- The `CacheConfig` (`CacheConfig`) is a configuration class that sets up the cache manager (`CacheManager`) for the application.
- The cache manager (`CacheManager`) is responsible for managing the caching infrastructure. It is configured to use Caffeine cache.
- The Caffeine cache (`Caffeine Cache: employeesCache`) is the cache named "employeesCache" that stores the response of the `getEmployees()` method in the `EmployeeService`.
- When a request is made to the `/employees` endpoint, it goes through the `EmployeeController`, which invokes the `getEmployees()` method in the `EmployeeService`. The service method checks the cache for the response before making an actual API call.
- If the response is found in the cache, it is returned directly. Otherwise, the service method fetches the employees from the source and stores the result in the cache for subsequent requests.

. . .

Thanks, before you go:

- 👏 Please clap for the story and follow the author 👉
- Please share your questions or insights in the comments section below. Let's help each other and become better Java developers.
- Let's connect on LinkedIn

Java · Spring Boot · Cache · Caffeine Cache · Performance

Published in Javarevisited

33K Followers · Last published 1 day ago

A humble place to learn Java and Programming better.

Follow

Written by Srikanth Dannarapu

636 Followers · 5 Following

Senior Java Developer

Follow

Responses (2)

What are your thoughts?

Respond

Mahesh Srinivas
over 1 year ago (edited)

Will Caffeine run in a distributed system where microservices have more than one instance running in kubernetes?

1 Hide replies Reply

Mukesh Kumar Gupta
22 days ago

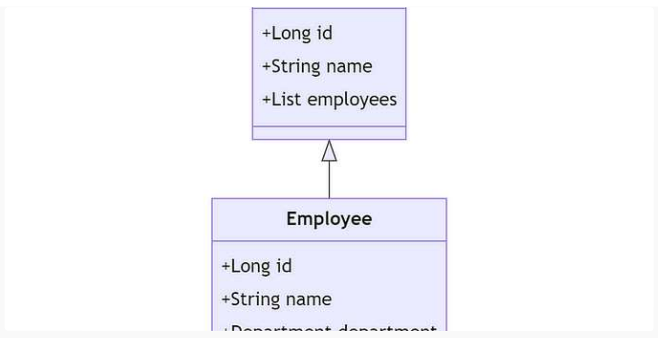
No it is alternative to loadingCache guava lib. It is using container memory (jvm) for cache.

Reply

Thteen
19 days ago

Thanks! On a related note, the integration of EchoAPI with JavaScript projects enhances my workflow by providing instant feedback on API calls.

Reply



In Javarevisited by Rahul Soni

How to Prevent Duplicate Requests in REST APIs and Why Spring Say...

Handling duplicate requests in a REST API is essential, especially for actions that create,...

Nov 11, 2024 · 226 · 2

In Javarevisited by Srikanth Dannarapu

Spring Data JPA findById Anti-Pattern

The Spring Data JPA findById method is a common anti-pattern that can lead to...

Mar 9, 2023 · 330 · 3

See all from Srikanth Dannarapu

See all from Javarevisited

Recommended from Medium

In DevOps.dev by Nithidol Vacharotayan

Resilience4j Circuit Breaker with Spring Boot

When building microservices, resilience is critical. Spring Boot and Resilience4j provid...

Dec 6, 2024 · 303 · 1

In Stackademic by Vijay SRJ

How to Implement SAGA Design Pattern in Spring Boot?

Contents

Nov 28, 2024 · 190 · 6

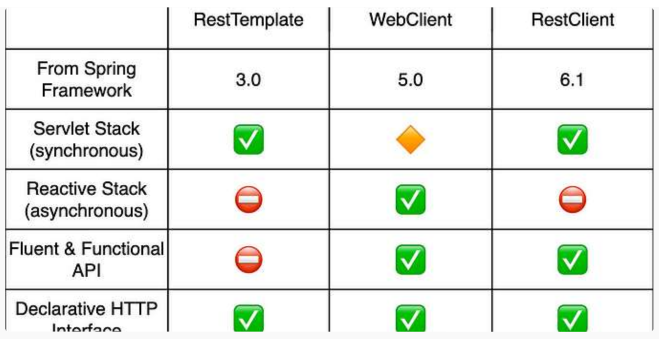
Lists

A Guide to Choosing, Planning, and Achieving...
13 stories · 2270 saves

General Coding Knowledge
20 stories · 1849 saves

data science and AI
40 stories · 310 saves

Staff picks
793 stories · 1553 saves



	RestTemplate	WebClient	RestClient
From Spring Framework	3.0	5.0	6.1
Servlet Stack (synchronous)	✅	🔶	✅
Reactive Stack (asynchronous)	⛔	✅	⛔
Fluent & Functional API	⛔	✅	✅
Declarative HTTP Interface	✅	✅	✅

Saeed Zarinfam

RestClient vs. WebClient vs RestTemplate

Using the suitable library to call REST API in Spring Boot

Jan 9, 2024 438 3

Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

Oct 26, 2024 8.5K 135

Full Stack Developer

Spring doesn't recommend @Autowired anymore???

There are 3 ways we can inject dependencies and Field Injection is not recommended...

Jul 19, 2024 423 11

Erick Zanetti

Comparison of Caching Tools in Java with Spring: Ehcache vs. Red...

Comparison of Ehcache, Redis, and Caffeine in Spring apps: Ehcache for ease, Redis for...

Aug 27, 2024 10 1

See more recommendations