Java Spring & AWS
Sep 7, 2021 - Oct 8, 2021
Monday to Friday
9:30 AM ET - 4:30 PM ET

Agenda: cache memory

- Spring boot cache memory

- Caching is a mechanism to enhance the performance of a system. It is a temporary memory that lies between the application and the persistent database. Cache memory stores recently used data items in order to reduce the number of database hits as much as possible.

Why we need caching?

- Caching of frequently used data in application is a very popular technique to increase performance of application. With caching, we store such frequently accessed data in memory to avoid hitting the costly backends every time when user requests the data. Data access from memory is always faster in comparison to fetching from storage like database, file system or other service calls.

- This is mostly opinionated decision about the type of data which should reside in cache and go through *cache lifecycle*. It varies in different scenario and requirement on how much time we can tolerate stale data.
- So caching candidates will vary on each project, still those are few examples of caching –
- List of products available in an eCommerce store
- Any Master data which is not frequently changed
- Any frequently used database read query, where result does not change in each call at least for a specific period.

Types of cache

- **In-memory caching**

This is the most frequently used area where caching is used extensively to increase performance of the application. In-memory caches such as Memcached and Radis are key-value stores between your application and your data storage. Since the data is held in RAM, it is much faster than typical databases where data is stored on disk.

RAM is more limited than disk, so cache invalidation algorithms such as least recently used (LRU) can help invalidate 'cold' entries and keep 'hot' data in RAM. Memcached is in-momory caching where Redis is more advanced which allows us to backup and restore facility as well as it is distributed caching tool where we can manage caching in distributed clusters.

- **Database caching:**

Your database usually includes some level of caching in a default configuration, optimized for a generic use case. Tweaking these settings for specific usage patterns can further boost performance. One popular in this area is first level cache of Hibernate or any ORM frameworks.

- **CDN caching**

- **Caches can be located on the client side (OS or browser), server side, or in a distinct cache layer.**

- **Web server caching:**

Reverse proxies and caches such as Varnish can serve static and dynamic content directly. Web servers can also cache requests, returning responses without having to contact application servers. In today's API age, this option is a viable if we want to cache API responses in web server level.

Spring boot cache annotations

- Spring framework provides **cache abstraction api** for different cache providers. The usage of the API is very simple, yet very powerful.

- @EnableCaching

- It enables Spring's annotation-driven cache management capability. In spring boot project, we need to add it to the boot application class annotated with @SpringBootApplication. Spring provides one concurrent hashmap as default cache, but we can override CacheManager to register external cache providers as well easily.

- **@Cacheable**

- It is used on the method level to let spring know that the response of the method are cacheable. Spring manages the request/response of this method to the cache specified in annotation attribute. For example, @Cacheable ("cache-name1", "cache-name2").

- @Cacheable(value="books", key="#isbn")
- public Book findStoryBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)
-
- @Cacheable(value="books", key="#isbn.rawNumber")
- public Book findStoryBook (ISBN isbn, boolean checkWarehouse, boolean includeUsed)
-
- @Cacheable(value="books", key="T(classType).hash(#isbn)")
- public Book findStoryBook (ISBN isbn, boolean checkWarehouse, boolean includeUsed)

- **@CachePut**

- Sometimes we need to manipulate the cacheing manually to put (update) cache before method call. This will allow us to update the cache and will also allow the method to be executed. The method will always be executed and its result placed into the cache (according to the @CachePut options).

- **@CacheEvict**

- It is used when we need to evict (remove) the cache previously loaded of master data. When **CacheEvict** annotated methods will be executed, it will clear the cache.

- **@Caching**

- This annotation is required when we need both CachePut and CacheEvict at the same time.

- **How to register a cache engine with spring boot**
- **Spring boot provides integration with following cache providers. Spring boot does the auto configuration with default options if those are present in class path and we have enabled cache by @EnableCaching in the spring boot application.**
    - **JCache (JSR-107) (EhCache 3, Hazelcast, Infinispan, and others)**
    - **EhCache 2.x**
    - **Hazelcast**
    - **Infinispan**
    - **Couchbase**
    - **Redis**
    - **Caffeine**
    - **Simple cache**

- **We can override specific cache behaviors in Spring boot by overriding the cache provider specific settings – for example-**
    - **spring.cache.infinispan.config=infinispan.xml**

- **Caffeine is a high performance Java 8 based caching library providing a near optimal hit rate. It provides an in-memory cache very similar to the Google Guava API. Spring Boot Cache starters auto-configured a CaffeineCacheManager if it finds the Caffeine in the classpath. The Spring Framework provides support for transparently adding Caching to an application.**

- **<dependency>**
- **<groupId>org.springframework.boot</groupId>**
- **<artifactId>spring-boot-starter-cache</artifactId>**
- **</dependency>**

- **<dependency>**
- **<groupId>com.github.ben-manes.caffeine</groupId>**
- **<artifactId>caffeine</artifactId>**
- **<version>2.7.0</version>**
- **</dependency>**

- **Service Setup:Let's create a simple Customer Service which will return customer information from the underlying system. We will add the Spring framework caching abstraction on this layer using Caffeine Cache. Let's look at our service class**


```
public interface CustomerService {
    Customer getCustomer(final Long customerID);
}
// Implementation
@Service
@CacheConfig(cacheNames = {"customer"})
public class DefaultCustomerService implements CustomerService {
    private static final Logger LOG = LoggerFactory.getLogger(DefaultCustomerService.class);
    @Cacheable
    @Override
    public Customer getCustomer(Long customerID) {
        LOG.info("Trying to get customer information for id {} ",customerID);
        return getCustomerData(customerID);
    }
    private Customer getCustomerData(final Long id){
        Customer customer = new Customer(id, "testemail@test.com", "Test Customer");
        return  customer;
    }
}
```

Caffeine Cache Configuration

Spring Boot provide several options to configure Caffeine cache on startup. We have the option to configure these properties either through configuration file (application.properties or yml) or programmatically. Let's see how to configure Caffeine cache using application.properties file:

```
spring.cache.cache-names=ccustomer
spring.cache.caffeine.spec=maximumSize=500,expireAfterAccess=600s
```

The spring.cache.cache-names property creates customer caches. The Caffeine spec define the cache maximum size as 500 and a time to live of 10 minutes.

Caffeine Java Configuration: If you like, we can also configure Caffeine cache using Java configuration. Let's see how the Java configuration look like:

```
@Configuration
public class CaffeineCacheConfig {

 @Bean
 public CacheManager cacheManager() {
  CaffeineCacheManager cacheManager = new CaffeineCacheManager("customer");
  cacheManager.setCaffeine(caffeineCacheBuilder());
  return cacheManager;
 }
 Caffeine < Object, Object > caffeineCacheBuilder() {
  return Caffeine.newBuilder()
   .initialCapacity(100)
   .maximumSize(500)
   .expireAfterAccess(10, TimeUnit.MINUTES)
   .weakKeys()
   .recordStats();
 }
}
```

Any queries?