Spring Framework Caching

Spring Framework Caching

- Spring Framework provides caching in a Spring Application, transparently. In Spring, the **cache abstraction** is a mechanism that allows consistent use of various caching methods with minimal impact on the code.

Cache Abstraction

- The cache abstraction mechanism applies to Java **methods**. The main objective of using cache abstraction is to **reduce** the number of executions based on the information present in the cache. It applies to expensive methods such as **CPU** or **IO bound.**
- Every time, when a method invokes, the abstraction applies a cache behavior to the method. It checks whether the method has already been executed for the given argument or not.
  - If yes, the cached result is returned without executing the actual method.
  - If no, first, the method executes, and the result is cached and returned to the user.
- Note: This approach works only for the methods that are guaranteed to return the same result for a given input. It does not matter how many times the method executes.

Caching

- The developers take care of two things while working with cache abstractions.
- **Cache Declaration:** It identifies the methods that need to be cached.
- **Cache Configuration:** The backing cache where the data is stored and read from.

dasturlash.uz

Caching

- Caching is a part of temporary memory (RAM). It lies between the application and persistence database. It stores the recently used data that reduces the number of database hits as much as possible. In other words, caching is to store data for future reference.

Why should we use the cache?

- The primary reason for using cache is to make data access faster and less expensive. When the highly requested resource is requested multiple times, it is often beneficial for the developer to cache resources so that it can give responses quickly. Using cache in an application enhances the performance of the application. Data access from memory is always faster in comparison to fetching data from the database. It reduces both monetary cost and opportunity cost.

What data should be cached?

- The data that do not change frequently.
- The frequently used read query in which results does not change in each call, at least for a period.
-

Types of Caching

- There are **four** types of caching are as follows:
  - In-memory Caching
  - Database Caching
  - Web server Caching
  - CDN Caching
-

In-memory Caching

- In-memory caching increases the performance of the application. It is the area that is frequently used. Memcached and **Redis** are examples of in-memory caching. It stores key-value between application and database. Redis is an **in-memory, distributed,** and advanced caching tool that allows backup and restore facility. We can manage cache in distributed clusters, also.

Spring Boot Cache Annotations

- @EnableCaching
- It is a class-level annotation. We can enable caching in the Spring Boot application by using the annotation **@EnableCaching.** It is defined in **org.springframework.cache.annotation** package. It is used together with **@Configuration** class.

Spring Boot Cache Annotations

- @CacheConfig
- It is a class-level annotation that provides a common cache-related setting. It tells the Spring where to store cache for the class. When we annotate a class with the annotation, it provides a set of default settings for any cache operation defined in that class. Using the annotation, we need not to declare things multiple times.

@Cacheable

- It is a method level annotation. It defines a cache for a method's return value. The Spring Framework manages the requests and responses of the method to the cache that is specified in the annotation attribute. The @Cacheable annotation contains more options. For example, we can provide a **cache name** by using the **value** or **cacheNames** attribute.
- We can also specify the **key** attribute of the annotation that uniquely identifies each entry in the cache. If we do not specify the key, Spring uses the default mechanism to create the key.

```
@Cacheable(value = "student", key = "#id")
   public StudentEntity get(Integer id) {
       return studentRepository.findById(id).orElse(null);
   }
```

Links

- https://www.javatpoint.com/spring-boot-caching
- https://howtodoinjava.com/spring-boot2/spring-boot-cache-example/