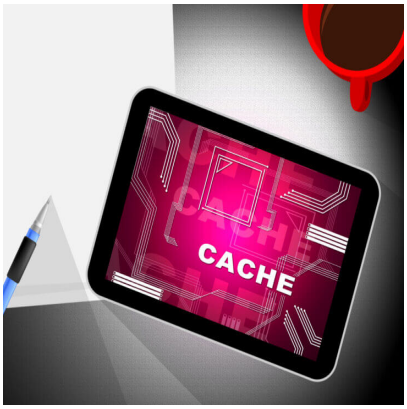

Improve Performance by Simple Cache Mechanism in SpringBoot Application

Description

Mobile Cloud Computing â?? Overview, Challenges and the Future

- Harish Nankani

Introduction â?? The Problem

For any project, there are database calls. Sometimes the database calls are done from the loops due to the requirements in responding to UI. Such loops can be repetitive, causing a performance hit by calling a database multiple times.

This blog can help solve this performance issue by using simple Cache implementation without using any additional libraries or frameworks.

Problem with Code

Consider DB calls are to find the object by Id, and if such call is made within for loop, then the code looks like this:

```
List productsList = productRepo.search(keyword, pageable);
ResponseDto response = new ResponseDto();
for (Product product : productsList) {
dto.setProduct(product);
Company company = companyRepo.getById(product.getCompanyID());
dto.setCompanyName(company.getName());
}
```

This illustration code is only to understand the concept and not the real code. It shows the result in UI â?? for the company name of each product, the DB call is made to get the company details within the for loop. If the products are huge, it will definitely impact the performance.

Basic Cache Implementation

Consider a simple cache class as CacheUtil.

```
public class CacheUtil {
private static Map<String, Company> companyMap = new HashMap<String, Company>();
public static void setCompanyMap(String id, Company company) {
this.companyMap.put(id, company);
}
public static Company getCompanyById(String id) {
this.companyMap.get(id);
}
public static void clear() {
this.companyMap.clear();
}
}
```


The above code uses a static map to ensure that the cache is available for all requests. It provides the company object by reference id.

How to use CacheUtil?

There is a small twist in using this cache. The strategy is to make repo implementation custom.

```
public interface CompanyRepoBasic extends JPARepository<String, Company> {
}

public interface CompanyRepoCustom {
public Company getCompanyById(String id);
}

public interface CompanyRepo extends CompanyRepoBasic, CompanyRepoCustom {
}

public class CompanyRepoImpl implements CompanyRepoCustom {
@Autowired
private CompanyRepoBasic companyRepoBasic;

public Company getCompanyById(String id) {
Company company = CacheUtil.getCompanyById(id);
if (company == null) {
company = companyRepoBasic.getById(id);
CacheUtil.put(id, company);
```

```
}
return company;
}
}
```

Final Call

A slight modification is to be made in the for loop to make it work. As a custom repo is created with a different method name getCompanyById, the method call companyRepo.getById used in the for loop should be changed to companyRepo.getCompanyById, and thatâ??s it.

```
for(Product product : productsList) {
dto.setProduct(product);
Company company = companyRepo.getCompanyById(product.getCompanyID());
dto.setCompanyName(company.getName());
}
```


How it works

CompanyRepo implementation includes both basic repo calls and custom repo calls. getCompanyById is the custom method that checks for the company from the cache. If the cache does not include the company of such id, then it calls the DB using basic repo and puts it into the cache.

So considering there are 100 products of the same company, then the for loop will not hit the DB 100 times with this cache implementation. It will hit DB once and then all the other 99 times; it will get the company object from the cache.

Enhancements

- Whenever the company object is saved or updated, the cache should be updated with the latest company object. This will always provide the latest company data for any request.
- Keep an expiry or clear the cache after some duration. Add one scheduler that runs every such configured duration to clear the cache.
- Multiple objects can be cached, and such implementation needs the modification of the above code (example shows only the company object). Use the map of maps or different maps for different objects.
- Make some property in application.properties or environment variables to set the number of objects that can be cached. For example, 1000 companies can be cached. If more than 1000 is being stored, keep the strategy to remove the oldest company.

The bottom line

We generally know that when it comes to offering the best experience to end-users, application performance is critical. Caching aids in this by acting as a bridge between the server and the end-user, delivering data on-demand real-time. So the more features added to the cache implementation, it becomes a custom cache library. Although caching may appear to be an afterthought in small applications, it is critical in complex ones.

Category

1. Atmecs-Blog

Tags

1. Application performance
2. Cache
3. JPA Cache
4. Performance
5. Simple Cache
6. Spring Boot Performance

Date Created
July 19, 2022
Author
admin