Implement caching in Java microservices using Redis Cluster

S Sina Salmani · Follow
3 min read · Jul 25, 2022

23

When it comes to boosting your application's speed, every millisecond matters. New research from Google has found that 53% of mobile website visitors will leave if a webpage doesn't load within three seconds.

· · ·

Why do we need cache for microservices?

Cache reduces latency and increases performance and scalability in microservices.

The scenario

Consider these microservices:

- Add a Product
- Fetch All Products
- Delete a Product by Id

Implement these services and use three annotations **@Cacheable** **@CacheEvict** ***@CachePut*** *to use Redis cache in inserting, fetching, and updating data. At the first call of fetch service, all products were retrieved from DB but for the next call, use cache. In case of deleting a product, it must remove from the cache, and when adding a new product, it should add to the cache.*

Where to start?

Install Redis on your system; use this for mac OS:

```
$ brew install redis
```

In case you want to install Redis using the docker container, use this link: https://www.atlantic.net/dedicated-server-hosting/how-to-install-redis-server-using-docker-container/

Start Redis server:

```
$ redis-server
```

Implementation

Step 1: Init spring boot and add this dependency for Redis connection:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

Spring provides Data Redis lib for easy configuration and access to Redis from Spring; besides, using Lettuce driver help to develop reactive API.

Step 2: Create a Product entity with @RedisHash

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@RedisHash("pro")
public class Product implements Serializable {

    @Id
    private int id;
    private String name;
    private long price;
}
```

Redis hashes are an implementation of the hash table or hash map data structure. Hash tables map unique keys to values.

Step 3: Create Redis configuration class

```
@Configuration
public class RedisConfig {

    //Creating Connection with Redis
    @Bean
    public RedisConnectionFactory redisConnectionFactory() {
        return new LettuceConnectionFactory();
    }

    //Creating RedisTemplate for Entity 'Product'
    @Bean
    public RedisTemplate<Integer, Product> redisTemplate(){
        RedisTemplate<Integer, Product> redisTemplate = new
RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory());
        return redisTemplate;
    }
}
```

Step 4: Create Product Repo

> *Based on Spring documentation @EnableRedisRepositories Annotation to activate Redis repositories. If no base package is configured through either `value()`, `basePackages()` or `basePackageClasses()` it will trigger scanning of the package of annotated class.*

```
@EnableRedisRepositories
public interface ProductRepo extends CrudRepository<Product,Integer>
{}
```

Step 5: Create a service layer

```
@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductRepo productRepo;

    public Product addProduct(Product product){
        return productRepo.save(product);
    }

    @Override
    public List<Product> getAllProduct() {

        System.out.println("HI");
        return (List<Product>) productRepo.findAll();
    }

    @Override
    public int deleteProductById(int id) {
        productRepo.deleteById(id);
        return 1;
    }
}
```

Step 6: Create a controller layer

```
@RestController
@RequestMapping("/api")
public class ProductController {

    @Autowired
    private ProductService productService;

    @PostMapping("/v1.1/add")
    @CachePut(value="product")
    public Product addProduct(@RequestBody Product product){

        return productService.addProduct(product);
    }

    @GetMapping("/v1.1/getAllProducts")
    @Cacheable(value="Product")
    public List<Product> getProduct(){

        return productService.getAllProduct();
    }

    @GetMapping("/v1.1/del/{id}")
    @CacheEvict(value = "Product",key = "id")
    public int deleteProduct(@PathVariable int id){

        return productService.deleteProductById(id);
    }
}
```

Step 7: Add @EnableCaching above the main Spring Boot application class

This annotation triggers a post-processor that inspects every Spring bean for the presence of caching annotations on public methods.

Step 8: Add this to application.properties

```
spring.cache.redis.time-to-live=60000
```

Spring provides a comprehensive abstraction for common caching scenarios. If we want to set Time To Live of a cache, the configuration of the chosen cache provider must be tuned. You can also implement multiple TTL caches in Spring boot.

Step 9: add swagger to the project.

- in pom.xml

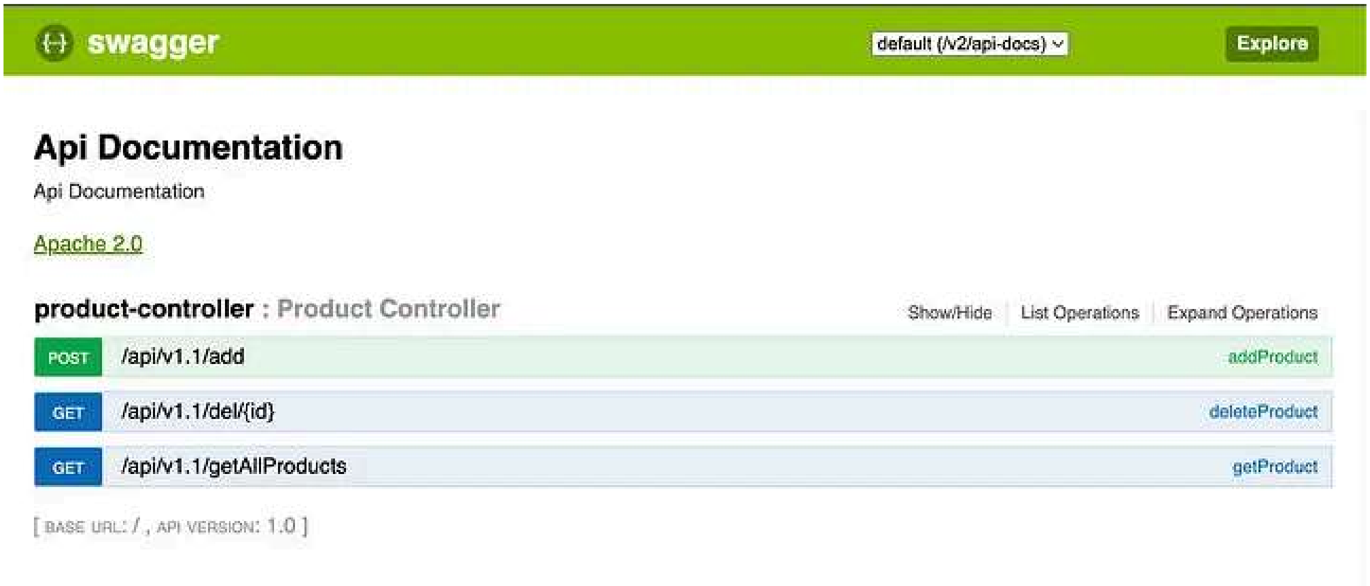
```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

- Add this annotation above the main Spring Boot application class

```
@EnableSwagger2
```

- In properties file

```
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```



Api Documentation

Api Documentation

Apache 2.0

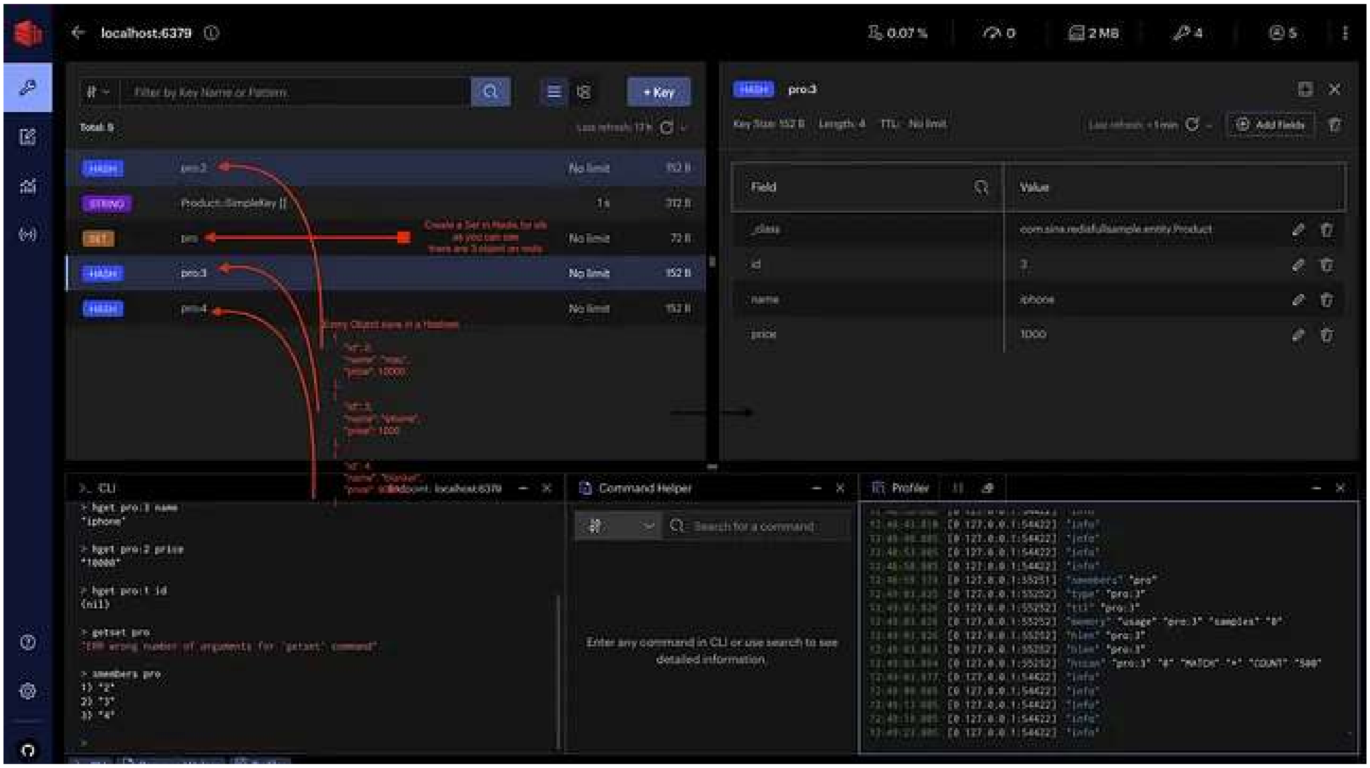
product-controller : Product Controller

POST	/api/v1.1/add	addProduct
GET	/api/v1.1/del/{id}	deleteProduct
GET	/api/v1.1/getAllProducts	getProduct

[BASE URL: / , API VERSION: 1.0]

Result in Redis

Install RedisInsight to profile the process and see Hash and Set that initialized when using cache. Adding three objects create a Set(the name set in @RedisHash). And per object, generate a Hash(pro:2,pro:3,pro:4 name of hash set: key)


localhost:6379
0.07 %
0
2 MB
4
5
Filter by Key Name or Pattern
+ Key
Total: 5
HASH pro:2 No limit 152 B
STRING Product::SimpleKey [] 1 s 312 B
SET pro No limit 72 B
HASH pro:3 No limit 152 B
HASH pro:4 No limit 152 B
HASH pro:3
Key Size: 152 B Length: 4 TTL: No limit
Add Fields
CLI
Command Helper
Search for a command
Profiler

Create a Set in Redis for ids
as you can see
there are 3 object on redis

Every Object save in a Hashes

```
{
"id":2,
"name":"mac",
"price":10000
}
{
"id":3,
"name":"iphone",
"price":1000
}
{
"id":4,
"name":"blanket",
"price"
```

Field	Value
_class	com.sina.redisfullsample.entity.Product
id	3
name	iphone
price	1000

Endpoint: localhost:6379

```
> hget pro:3 name
"iphone"

> hget pro:2 price
"10000"

> hget pro:1 id
(nil)

> getset pro
"ERR wrong number of arguments for 'getset' command"

> smembers pro
1) "2"
2) "3"
3) "4"

>
```

Enter any command in CLI or use search to see detailed information

```
12:48:43.816 [0 127.0.0.1:54422] "info"
12:48:48.885 [0 127.0.0.1:54422] "info"
12:48:53.885 [0 127.0.0.1:54422] "info"
12:48:58.885 [0 127.0.0.1:54422] "info"
12:48:59.773 [0 127.0.0.1:55251] "smembers" "pro"
12:49:01.825 [0 127.0.0.1:55252] "type" "pro:3"
12:49:01.826 [0 127.0.0.1:55252] "ttl" "pro:3"
12:49:01.828 [0 127.0.0.1:55252] "memory" "usage" "pro:3" "samples" "0"
12:49:01.826 [0 127.0.0.1:55252] "hlen" "pro:3"
12:49:01.863 [0 127.0.0.1:55252] "hlen" "pro:3"
12:49:01.864 [0 127.0.0.1:55252] "hscan" "pro:3" "0" "MATCH" "*" "COUNT" "500"
12:49:01.877 [0 127.0.0.1:54422] "info"
12:49:08.885 [0 127.0.0.1:54422] "info"
12:49:13.885 [0 127.0.0.1:54422] "info"
12:49:18.885 [0 127.0.0.1:54422] "info"
12:49:23.885 [0 127.0.0.1:54422] "info"
```

Redis Spring Boot Cache Microservices Spring Data

Written by Sina Salmani

17 Followers · 3 Following

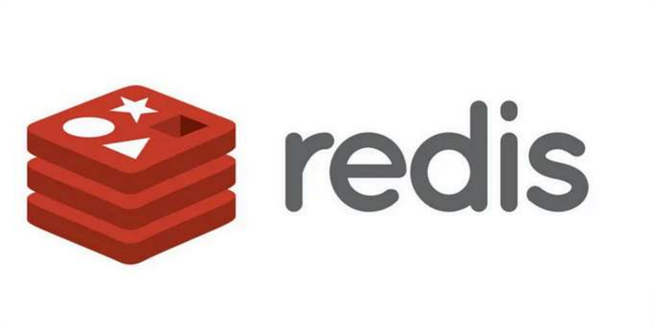
Follow

No responses yet

What are your thoughts?

Respond

Recommended from Medium



Tharindu Dulshan

Optimizing Spring Boot Applications with Redis Caching

In my previous blog, I explained what is Caching and different methods of caching...

Sep 20 •••

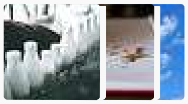
Technical Life

Solving Redis Connection Issues in a Spring Boot Application

Troubleshooted Redis connection issues in Spring Boot

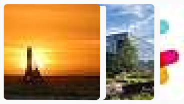
✦ Jun 24 •••

Lists



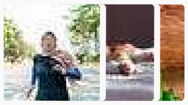
Staff picks

780 stories · 1492 saves



Stories to Help You Level-Up at Work

19 stories · 891 saves



Self-Improvement 101

20 stories · 3126 saves

Productivity 101

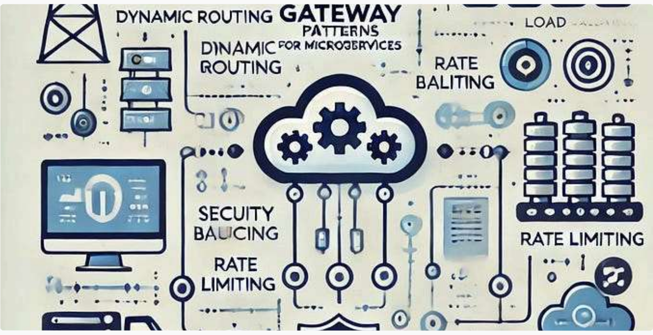
20 stories · 2641 saves

In Stackademic by Mpavani

Microservices tricky interview questions must know

1.You have two microservices, Order Service and Payment Service. When an order is...

✦ Oct 30 •••

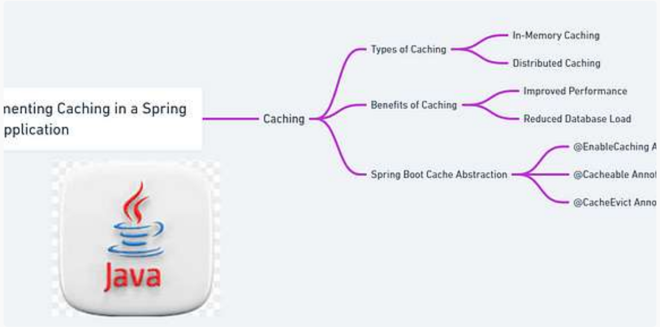


Balian's techologies and innovation lab

API Gateway Patterns with Spring Cloud Gateway: Routing, Load...

In microservices architectures, an API Gateway acts as the single entry point for all...

✦ Nov 12 •••



Sanjay Singh

Implementing Caching in a Spring Boot Application: A Complete…

Overview

✦ Oct 11

Jobin J

Redis Caching with Java using Jedis

Redis caching is a technique that uses Redis, an open-source, in-memory data structure…

Aug 27

See more recommendations