Day 43 ~ Springboot3.1.x | 3分钟学会 |Spring Boot 缓存最佳实践

原创 编程疏影 路条编程 2024年04月15日 08:01 北京

在这个充满挑战和收获的60天学习之旅中，你将迅速提升成为一名全栈工程师。专注于Spring Boot框架，我们将深入研究高级特性，从项目初始化到微服务架构，再到性能优化和持续集成部署。无论你是初学者还是有一定经验的开发者，这个专题都将带你穿越从零到全面掌握Spring Boot的学习曲线。

继上一篇文章深入探讨Spring Boot的缓存整合后，今天我们将提供一些关于缓存使用的最佳实践。这些最佳实践将帮助我们更好地利用缓存来提升应用的性能。

缓存与数据库同步更新

一般来讲，数据库和缓存的同步更新主要通过两种策略： `Cache Aside` 和 `Read/Write Through` 。

Cache Aside模式

Cache Aside是最常用的一种缓存模式。流程如下：

- 读取数据：先从缓存读取数据，没有的话从数据库中读取，然后写入缓存；
- 更新数据：先把数据存入数据库，成功后使缓存失效。

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RedisTemplate redisTemplate;

    public User getUser(Long id) {
        // 查询缓存
        User cachedUser = (User)redisTempla
        if (cachedUser != null) {
              // 缓存命中
              return cachedUser;
        }
        // 缓存未命中，查询数据库并写入缓存
        User user = userRepository.findByI
        if (user != null) {
            redisTemplate.opsForValue().se
        }
        return user;
    }

    public User updateUser(User user) {
        // 操作数据库
        userRepository.insertOrUpdate(user
        // 使缓存失效
        redisTemplate.delete("user:" + use
    }
}
```

Read/Write Through模式

Read-Through模式是指从缓存读取数据，如果未命中再从数据源读取。Write-Through模式是指对缓存的每一个写操作都要写入到数据源。

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Cacheable(value = "users", key = "#i
    public User getUser(Long id) {
         // 查询数据库
        return userRepository.findById(id
    }

    @CachePut(value = "users", key = "#use
    public User updateUser(User user) {
        // 操作数据库
        return userRepository.insertOrUpda
    }
}
```

在上述代码中， `getUser` 方法会先查询缓存，如果缓存中没有，则查询数据库并将结果存到缓存，当

下次同样的请求来时，就可以直接从缓存读取。 `updateUser` 方法则是将修改内容同时写入数据库和缓存。

处理缓存击穿和雪崩

缓存击穿和雪崩是两种常见的缓存问题。缓存击穿是指一个查询在缓存和数据库中都查不到结果的情况，而缓存雪崩则是指在缓存失效时，大量的请求同时涌向数据库。

防止缓存击穿，我们可以在查询不到结果时，将一个空结果存入缓存，并设置一个较短的过期时间。

防止缓存雪崩，我们可以给缓存项设置一个随机的过期时间，从而避免大量缓存项同时过期。

缓存击穿

缓存击穿是指对一些持续高并发访问的热点数据，当这些数据的缓存失效的一刹那，所有的请求都直接访问数据库，造成数据库压力瞬间增大。处理这种问题，我们可以使用互斥锁或者采用空值缓存。

举个例子，当我们查询用户信息时，如果缓存击穿发生，我们可以让第一个请求去数据库查询数据，其他的请求等待，当第一个请求完成查询后，将结果放入缓存，然后释放锁，其他等待的请求此时可以直接从缓存中获取数据。

以下是利用Spring的同步锁@CacheLock来解决缓存击穿的代码示例：

```
@Service
public class UserService {

    @CacheLock
    @Cacheable(value = "users", key = "#i

    public User getUser(Long id) {
        return userRepository.findById(id
    }
}
```

缓存雪崩

缓存雪崩是指在某一时间点，大量的缓存同时失效，此时大量的请求直接打到数据库，造成数据库压力瞬间增大。我们可以通过在缓存失效时间上添加随机数来避免这个问题。

以下是添加随机失效时间的代码示例：

```
@Configuration
@EnableCaching

public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        Random random = new Random();

        Caffeine<Object, Object> caffeine(
                .expireAfterAccess(5 + ra
                .initialCapacity(100)
                .maximumSize(500);

        return new CaffeineCacheManager("
            setCaffeine(caffeineCacheBuil
        }};
    }
}
```

在上述代码中，我们对每个缓存项设置了5至10分钟

之间的随机失效时间，这样就可以避免大量的缓存同时失效。

但需要注意的是，这些仅仅是处理缓存问题的一种思路和方案，真正的解决方案需要根据业务情况和问题的实际表现进行调整和优化。

总的来说，为了更好地使用Spring Boot的缓存，我们需要了解缓存的工作机制，并利用Spring Boot提供的注解来有效地管理缓存。同时，我们也需要注意防止缓存击穿和雪崩等常见的缓存问题。

希望这篇文章能帮你理解并更有效地使用Spring Boot的缓存，并希望你能喜欢这个缓存最佳实践的教程。

今天就讲到这里，如果有问题需要咨询，大家可以直接留言或扫下方二维码来知识星球找我。也可以添加 happyzjp 微信受邀加入学习社群，我们会尽力为你解答。

作者：路条编程（转载请获本公众号授权，并注明作者与出处）

60天修炼 43

60天修炼 · 目录

上一篇 · Day 42 ~ Springboot3.1.x | 3分钟学...