Tối ưu và bảo mật web

Setup redis

porn.xml file

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

propoti file

# Cấu hình Redis connection

spring.redis.host=localhost

spring.redis.port=6379

spring.redis.password=your-password # Nếu Redis có password

spring.redis.database=0 # Chỉ định database Redis cần sử dụng

# Cấu hình TTL cho cache

spring.cache.redis.time-to-live=10m # Thời gian tồn tại của cache (10 phút)

spring.cache.redis.cache-null-values=false # Không lưu các giá trị null

setting trong class (setup cache)

@Configuration

@EnableCaching

@Bean

public CacheManager cacheManager(RedisConnectionFactory connectionFactory) {

RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()

.entryTtl(Duration.ofMinutes(10)) // TTL cho từng cache key

.disableCachingNullValues();

return RedisCacheManager.builder(connectionFactory)

.cacheDefaults(config)

.build();

}

Mã hóa thông tin

Lưu cache cho các truy vấn

cách dùng

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

// Caching kết quả trả về từ phương thức

@Cacheable(value = "products", key = "#categoryId")

public List<Product> getProductsByCategory(Long categoryId) {

return productRepository.findByCategoryId(categoryId);

}

}

@Cacheable: Spring sẽ lưu kết quả trả về từ phương thức vào cache. Khi có yêu cầu sau với categoryId tương tự, hệ thống sẽ lấy trực tiếp dữ liệu từ cache thay vì thực hiện truy vấn cơ sở dữ liệu.

value = "products": Là tên của cache, bạn có thể sử dụng tên này để truy cập dữ liệu đã được cache.

key = "#categoryId": Là khóa duy nhất để xác định dữ liệu trong cache. Khi categoryId thay đổi, dữ liệu cache cũng sẽ thay đổi.

Sử dụng Cache Evict (Xóa cache khi cần): Để đảm bảo tính nhất quán của dữ liệu, đôi khi bạn cần xóa cache khi dữ liệu bị thay đổi (ví dụ: khi sản phẩm mới được thêm vào).

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

@CacheEvict(value = "products", key = "#categoryId")

public void addProduct(Product product, Long categoryId) {

productRepository.save(product);

}

}

Sử dụng Cache Put (Cập nhật cache): Để cập nhật dữ liệu trong cache mà không cần phải xóa cache, bạn có thể sử dụng @CachePut.

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

@CachePut(value = "products", key = "#product.id")

public Product updateProduct(Product product) {

return productRepository.save(product);

}

}

Setup CORD

Phân quyền

Phân trang cho web

Dùng bất đồng bộ

Setup

@Configuration

@EnableAsync

public class AsyncConfig {

@Bean

public Executor taskExecutor() {

return new ThreadPoolTaskExecutor();

}

}

Sử dụng

@RestController

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/user/{id}")

public CompletableFuture<ResponseEntity<UserDTO>> getUser(@PathVariable Long id) {

// Gọi hàm bất đồng bộ để lấy dữ liệu người dùng

CompletableFuture<UserDTO> userDTOFuture = userService.getUserByIdAsync(id);

// Sau khi lấy xong dữ liệu người dùng, trả về dữ liệu cho frontend

return userDTOFuture.thenApply(userDTO -> {

// Bạn có thể xử lý thêm dữ liệu nếu cần

return ResponseEntity.ok(userDTO); // Trả về dữ liệu thực tế

});

}

}

@Service

public class UserService {

@Async

public CompletableFuture<UserDTO> getUserByIdAsync(Long id) {

// Giả lập truy vấn dữ liệu người dùng từ database

UserDTO userDTO = new UserDTO();

userDTO.setId(id);

userDTO.setName("User" + id);

// Trả về CompletableFuture đã hoàn thành với dữ liệu

return CompletableFuture.completedFuture(userDTO);

}

}

Giới hạng lượng truy cập với redis và token người dùng:

// Giới hạn số lượng yêu cầu cho người dùng

String userId = jwtTokenUtil.extractUserIdFromToken(token);

long requestCount = redisTemplate.opsForValue().increment("user\_rate\_limit\_" + userId, 1);

// Nếu số lượng yêu cầu vượt quá giới hạn (ví dụ: 200 yêu cầu trong 10 phút)

if (requestCount > 200) {

// Trả về mã lỗi 429: Too Many Requests

throw new TooManyRequestsException("Rate limit exceeded");

}

// Đặt thời gian hết hạn cho key Redis (10 phút)

redisTemplate.expire("user\_rate\_limit\_" + userId, 10, TimeUnit.MINUTES);

Gửi mail

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-mail</artifactId>

</dependency>

# Cấu hình server email

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=your-email@gmail.com

spring.mail.password=your-email-password

spring.mail.protocol=smtp

spring.mail.smtp.auth=true

spring.mail.smtp.starttls.enable=true

spring.mail.smtp.ssl.trust=smtp.gmail.com

spring.mail.default-encoding=UTF-8

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.stereotype.Service;

import javax.mail.internet.MimeMessage;

import javax.mail.MessagingException;

@Service

public class EmailService {

@Autowired

private JavaMailSender mailSender;

public void sendHtmlEmail(String to, String subject) throws MessagingException {

// Tạo đối tượng MimeMessage

MimeMessage message = mailSender.createMimeMessage();

// Sử dụng MimeMessageHelper để thiết lập thông tin cho email

MimeMessageHelper helper = new MimeMessageHelper(message, true);

helper.setTo(to);

helper.setSubject(subject);

helper.setFrom("your-email@gmail.com"); // Địa chỉ email người gửi

// HTML email với style inline

String htmlContent = "<html><body>"

+ "<h1 style='color: #4CAF50;'>Chào bạn!</h1>"

+ "<p style='font-size: 16px;'>Cảm ơn bạn đã sử dụng dịch vụ của chúng tôi.</p>"

+ "<p style='font-size: 16px;'>Đây là một email thử nghiệm được gửi từ ứng dụng Spring Boot với nội dung <b>HTML</b>.</p>"

+ "<table style='border: 1px solid #ccc; border-collapse: collapse;'>"

+ "<tr><th style='padding: 10px; text-align: left;'>Sản phẩm</th><th style='padding: 10px;'>Giá</th></tr>"

+ "<tr><td style='padding: 10px;'>Sản phẩm A</td><td style='padding: 10px;'>$10</td></tr>"

+ "<tr><td style='padding: 10px;'>Sản phẩm B</td><td style='padding: 10px;'>$20</td></tr>"

+ "</table>"

+ "</body></html>";

// Đặt nội dung email ở dạng HTML

helper.setText(htmlContent, true);

// Gửi email

mailSender.send(message);

}

}

gửi mail đính kèm file

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.stereotype.Service;

import javax.mail.internet.MimeMessage;

import java.io.File;

@Service

public class EmailService {

private final JavaMailSender mailSender;

public EmailService(JavaMailSender mailSender) {

this.mailSender = mailSender;

}

public void sendEmailWithAttachment(String to, String subject, String text, String filePath) throws Exception {

MimeMessage message = mailSender.createMimeMessage();

MimeMessageHelper helper = new MimeMessageHelper(message, true);

helper.setTo(to);

helper.setSubject(subject);

helper.setText(text, true); // true để email hỗ trợ HTML content

helper.setFrom("your-email@gmail.com");

// Đính kèm tệp

File file = new File(filePath);

helper.addAttachment(file.getName(), file);

mailSender.send(message);

}

}