Trong đoạn mã trên, EncryptorConfig là một lớp cấu hình của Spring. Dưới đây là giải thích chi tiết từng phần của đoạn mã:

**Phần 1: Khai báo và khởi tạo các hằng số**

java

Copier

private static final String JASYPT\_ALGORITHM = "PBEWithMD5AndDES";

private static final String JASYPT\_PASSWORD\_SALT = getJasyptPassword();

* JASYPT\_ALGORITHM là một hằng số định nghĩa thuật toán mã hóa sử dụng trong việc mã hóa và giải mã chuỗi.
* JASYPT\_PASSWORD\_SALT là một hằng số định nghĩa giá trị muối (salt) được sử dụng trong quá trình mã hóa và giải mã chuỗi. Giá trị này được lấy từ phương thức getJasyptPassword().

**Phần 2: Phương thức jasyptStringEncryptor()**

java

Copier

@Bean(name = "jasyptStringEncryptor")

public static StringEncryptor jasyptStringEncryptor() {

StandardPBEStringEncryptor configurationEncryptor = new StandardPBEStringEncryptor();

configurationEncryptor.setConfig(environmentVariablesConfiguration());

return configurationEncryptor;

}

* Phương thức jasyptStringEncryptor() được đánh dấu bởi @Bean và trả về một đối tượng StringEncryptor. Đối tượng này được sử dụng để mã hóa và giải mã chuỗi.
* Một đối tượng StandardPBEStringEncryptor được khởi tạo và gán cho biến configurationEncryptor.
* Phương thức setConfig(environmentVariablesConfiguration()) được gọi để đặt cấu hình cho configurationEncryptor sử dụng phương thức environmentVariablesConfiguration().
* Cuối cùng, configurationEncryptor được trả về từ phương thức.

**Phần 3: Phương thức getJasyptPassword()**

java

Copier

private static String getJasyptPassword() {

byte[] bytesFromFile = ResourceUtils.readBytesFromResource(

"encrypted.jasypt\_password");

byte[] saltKey = bytesFromFile;

if (Profiles.isActiveProfile(Profiles.PRD) || Profiles.isActiveProfile(Profiles.STG)) {

AWSKMS kmsClient = getAwskms();

ByteBuffer ciphertextBlob = ByteBuffer.wrap(bytesFromFile);

DecryptResult decryptResult =

kmsClient.decrypt(new DecryptRequest().withCiphertextBlob(ciphertextBlob));

saltKey = decryptResult.getPlaintext().array();

}

return new String(saltKey, StandardCharsets.UTF\_8);

}

* Phương thức getJasyptPassword() được sử dụng để lấy giá trị của JASYPT\_PASSWORD\_SALT.
* Đầu tiên, phương thức ResourceUtils.readBytesFromResource("encrypted.jasypt\_password") được gọi để đọc nội dung từ tệp "encrypted.jasypt\_password" và lưu trữ vào mảng bytesFromFile.
* Sau đó, mảng saltKey được gán bằng bytesFromFile.
* Nếu môi trường hoạt động là Profiles.PRD hoặc Profiles.STG, thì một đối tượng AWSKMS được tạo bằng cách gọi phương thức getAwskms().
* Một đối tượng ByteBuffer được tạo bằng cách gói bytesFromFile vào đó.
* Một yêu cầu giải mã DecryptRequest được tạo và gọi phương thức kmsClient.decrypt() để giải mã ciphertextBlob.
* Kết quả giải mã được lấy từ decryptResult.getPlaintext().array() và gán cho saltKey.
* Cuối cùng, một đối tượng String được tạo từ saltKey bằng cách sử dụng mã hóa chuỗi StandardCharsets.UTF\_8.

**Phần 4: Phương thức getAwskms()**

java

Copier

private static AWSKMS getAwskms() {

AWSKMSClientBuilder awskmsClientBuilder = AWSKMSClientBuilder.standard();

if (Profiles.isActiveProfile(Profiles.STG)) {

awskmsClientBuilder = awskmsClientBuilder.withRegion(Regions.AP\_SOUTHEAST\_1);

}

return awskmsClientBuilder.build();

}

* Phương thức getAwskms() được sử dụng để tạo và trả về một đối tượng AWSKMS.
* Một đối tượng AWSKMSClientBuilder được khởi tạo bằng cách gọi AWSKMSClientBuilder.standard().
* Nếu môi trường hoạt động là Profiles.STG, thì phương thức withRegion(Regions.AP\_SOUTHEAST\_1) được gọi để đặt khu vực AWS là "AP\_SOUTHEAST\_1".
* Cuối cùng, phương thức build() được gọi trên awskmsClientBuilder để tạo và trả về đối tượng AWSKMS.

**Phần 5: Phương thức environmentVariablesConfiguration()**

java

Copier

public static PBEConfig environmentVariablesConfiguration() {

EnvironmentStringPBEConfig config = new EnvironmentStringPBEConfig();

config.setAlgorithm(JASYPT\_ALGORITHM);

config.setPassword(JASYPT\_PASSWORD\_SALT);

return config;

}

* Phương thức environmentVariablesConfiguration() được sử dụng để tạo và cấu hình một đối tượng PBEConfig.
* Một đối tượng EnvironmentStringPBEConfig được khởi tạo và gán cho biến config.
* Phương thức setAlgorithm(JASYPT\_ALGORITHM) được gọi để đặt thuật toán mã hóa của config thành giá trị của JASYPT\_ALGORITHM.
* Phương thức setPassword(JASYPT\_PASSWORD\_SALT) được gọi để đặt giá trị muối của config thành giá trị của JASYPT\_PASSWORD\_SALT.
* Cuối cùng, đối tượng config được trả về từ phương thức.

Tổng quan, mã trên cung cấp cấu hình cho việc mã hóa và giải mã chuỗi sử dụng thư viện Jasypt. Nó sử dụng một giá trị muối (salt) được lưu trữ trong tệp "encrypted.jasypt\_password" và có thể được giải mã bằng cách sử dụng dịch vụ AWS KMS nếu môi trường hoạt động là Profiles.PRD hoặc Profiles.STG.

//////////////////////////////////////////////////////////////////////

Đây là một lớp cấu hình trong ứng dụng Java Spring Boot. Lớp này được đánh dấu bằng @Configuration để chỉ ra rằng nó là một cấu hình bean.

Lớp FeatureApiProxyConfig có một số phương thức và lớp bên trong để cấu hình và khởi tạo các thông tin cần thiết cho việc proxy API.

Các thành phần chính của lớp này là:

1. private static final String FEATURE\_API\_PROXY\_CONFIG\_JSON = "feature-api-proxy-config.json";: Đây là tên của tệp cấu hình được sử dụng để đọc thông tin proxy API. Tệp này được giả định là có định dạng JSON và được đặt trong thư mục lớp.
2. private final FeatureApiProxyInfo featureApiProxyInfo;: Đây là một bean FeatureApiProxyInfo được khởi tạo trong constructor và được sử dụng trong ứng dụng.
3. private static final StringEncryptor CONFIG\_ENCRYPTOR = EncryptorConfig.jasyptStringEncryptor();: Đây là một đối tượng StringEncryptor để mã hóa và giải mã các giá trị cấu hình. Nó được khởi tạo bằng cách sử dụng EncryptorConfig.jasyptStringEncryptor(), một phương thức tĩnh từ một lớp khác.
4. public FeatureApiProxyConfig(@Lazy FeatureApiProxyInfo featureApiProxyInfo): Đây là constructor của lớp FeatureApiProxyConfig. Nó nhận một tham số featureApiProxyInfo được chú thích bằng @Lazy, chỉ định rằng bean này sẽ được khởi tạo theo cách lười biếng.
5. @Bean: Đây là một phương thức được chú thích bằng @Bean, nó tạo ra một bean có kiểu FeatureApiProxyInfo. Phương thức này đọc thông tin proxy API từ tệp cấu hình JSON và trả về một đối tượng FeatureApiProxyInfo được khởi tạo.
6. Lớp FeatureApiProxyInfo: Đây là một lớp trong lớp FeatureApiProxyConfig, đại diện cho các thông tin cấu hình và khởi tạo API proxy.
   * FeatureApiProxySetting: Đây là một lớp lưu trữ thông tin cấu hình chung cho proxy API, bao gồm portalApiEndpoint và danh sách apiList chứa các thông tin cấu hình cho từng API.
   * FeatureApiInfo: Đây là lớp lưu trữ thông tin chi tiết cho mỗi API trong apiList. Nó bao gồm tên, tiêu đề, tiền tố URI proxy, thông tin Swagger, địa chỉ endpoint của API mục tiêu, thông tin xác thực và danh sách tiêu đề được phép.
   * SwaggerInfo: Đây là lớp lưu trữ thông tin Swagger cho mỗi API, bao gồm phiên bản và URL API Docs.
   * ApiAuthorization: Đây là lớp lưu trữ thông tin xác thực cho mỗi API, bao gồm loại xác thực (header, SAHeader hoặc queryParam), khóa, giá trị và vai trò. Lớp này cũng có một phương thức setValue để giải mã giá trị nếu nó được mã hóa bằng cách sử dụng CONFIG\_ENCRYPTOR.

Trong constructor của FeatureApiProxyInfo, các giá trị cấu hình được kiểm tra và sắp xếp theo tiền tố URI proxy trước khi được lưu trữ trong apiInfoMap.

Cuối cùng, lớp FeatureApiProxyConfig giúp cung cấp các thông tin cấu hình và khởi tạo cho việc proxy API trong ứngdụng.

//////////////////////////////////////////////////////////

Đoạn mã trên là một lớp cấu hình trong ứng dụng Java Spring Boot, được đánh dấu bằng @Configuration.

Lớp MessageSenderConfig định nghĩa các bean để cấu hình và khởi tạo các MessageSender cho việc gửi tin nhắn. Hai bean được định nghĩa trong lớp này là slackMessageSender và consoleMessageSender.

1. @Value("${slack.webhooks.url:}"): Đây là một annotation @Value để tiêm giá trị của thuộc tính slack.webhooks.url từ các tài nguyên cấu hình. Nếu giá trị không được tìm thấy, slackWebhooksUrl sẽ được gán giá trị mặc định là một chuỗi rỗng.
2. @Autowired: Đây là một annotation để tiêm vào một bean Environment, cho phép truy cập các thông tin môi trường và cấu hình của ứng dụng.
3. @Bean(name = "slackMessageSender"): Đây là một phương thức được chú thích bằng @Bean và đặt tên là "slackMessageSender". Phương thức này trả về một bean của kiểu MessageSender. Bean này chỉ được tạo ra nếu thuộc tính message.channel có giá trị "slack" trong tệp cấu hình. Nếu không có giá trị này, bean sẽ không được tạo ra.
4. @Bean(name = "consoleMessageSender"): Đây là một phương thức được chú thích bằng @Bean và đặt tên là "consoleMessageSender". Phương thức này trả về một bean của kiểu MessageSender. Bean này chỉ được tạo ra nếu thuộc tính message.channel có giá trị "console" trong tệp cấu hình. Nếu không có giá trị này, bean sẽ được tạo ra mặc định.
5. new SlackMessageSender(slackWebhooksUrl, environment.getActiveProfiles()[0]): Đây là việc khởi tạo một đối tượng SlackMessageSender và truyền vào hai tham số là slackWebhooksUrl và environment.getActiveProfiles()[0]. Đối tượng SlackMessageSender được sử dụng để gửi tin nhắn đến Slack. Tham số đầu tiên là URL webhook của Slack, được lấy từ giá trị thuộc tính slack.webhooks.url. Tham số thứ hai là tên của profile đang hoạt động trong môi trường ứng dụng.
6. new ConsoleMessageSender(): Đây là việc khởi tạo một đối tượng ConsoleMessageSender. Đối tượng ConsoleMessageSender được sử dụng để gửi tin nhắn đến console, không cần bất kỳ cấu hình nào.

Tóm lại, lớp MessageSenderConfig cung cấp các bean để cấu hình và khởi tạo MessageSender cho việc gửi tin nhắn. Các bean này được tạo ra dựa trên giá trị của thuộc tính message.channel trong tệp cấu hình. Nếu giá trị là "slack", sẽ được tạo ra một bean SlackMessageSender với URL webhook của Slack và tên profile. Nếu giá trị là "console" hoặc không có giá trị, sẽ được tạo ra một bean ConsoleMessageSender.

/////////////////////////////////////////////////////////////

Đoạn mã trên là một lớp cấu hình trong ứng dụng Java Spring Boot, được đánh dấu bằng @Configuration. Nó cấu hình OpenAPI và Swagger UI cho ứng dụng.

1. @Profile({Profiles.LOCAL, Profiles.DEV, Profiles.STG}): Đây là một annotation @Profile để chỉ định rằng lớp cấu hình này chỉ nên được kích hoạt khi một trong các profile là "LOCAL", "DEV", hoặc "STG" được kích hoạt trong ứng dụng.
2. public static final String X\_SERVER\_AUTH = "X-Server-Auth";: Đây là một hằng số định nghĩa tên của Security Scheme sẽ được sử dụng trong OpenAPI.
3. @Autowired private FeatureApiProxyInfo featureApiProxy;: Đây là một trường được tiêm vào bean này để sử dụng thông tin cấu hình từ FeatureApiProxyInfo.
4. @Bean public OpenAPI openAPI(): Đây là một phương thức khởi tạo một bean OpenAPI, đại diện cho đối tượng OpenAPI trong cấu hình. Trong phương thức này, ta định nghĩa thông tin cơ bản cho OpenAPI bằng cách tạo một đối tượng Info với tiêu đề và mô tả tương ứng, sau đó tạo một đối tượng OpenAPI và thiết lập thông tin và các thành phần cần thiết. Trong trường hợp này, ta thiết lập một server cơ bản, định nghĩa một Security Scheme với tên X\_SERVER\_AUTH và thiết lập yêu cầu bảo mật cho OpenAPI.
5. @Primary @Bean public SwaggerUiConfigProperties swaggerUiConfigProperties(SwaggerUiConfigProperties swaggerUiConfigProperties): Đây là một phương thức khởi tạo một bean SwaggerUiConfigProperties, đại diện cho cấu hình của Swagger UI trong ứng dụng. Phương thức này nhận một đối tượng SwaggerUiConfigProperties như tham số và trả về nó sau khi cấu hình. Trong phương thức này, ta xử lý thông tin cấu hình cho các URL Swagger UI sẽ hiển thị. Ta lấy danh sách các API từ featureApiProxy, lọc ra các API có thông tin Swagger và tạo ra các đối tượng SwaggerUrl tương ứng. Sau đó, ta thêm một SwaggerUrl cho Portal chính vào danh sách. Cuối cùng, ta thiết lập danh sách URL Swagger và tên URL chính trong swaggerUiConfigProperties và trả về nó.

Tóm lại, lớp OpenApiConfig là một lớp cấu hình Spring Boot, được kích hoạt dựa trên các profile "LOCAL", "DEV", hoặc "STG". Nó cấu hình OpenAPI và Swagger UI trong ứng dụng, bao gồm các thông tin cơ bản, Security Scheme và các URL Swagger UI để hiển thị.

////////////////////////////////////////////////////////////////////////////

Đoạn mã trên là một lớp cấu hình trong ứng dụng Java Spring Boot, được đánh dấu bằng @Configuration. Nó cấu hình RestTemplate và RetryTemplate trong ứng dụng.

1. @Bean public RestTemplate restTemplate(): Đây là một phương thức khởi tạo một bean RestTemplate, đại diện cho một client HTTP trong ứng dụng. Trong phương thức này, ta tạo một đối tượng MappingJackson2HttpMessageConverter để xử lý các tin nhắn HTTP với định dạng JSON. Ta thiết lập loại phương tiện hỗ trợ là text/plain. Sau đó, ta tạo một đối tượng StringHttpMessageConverter để xử lý các tin nhắn HTTP dạng chuỗi với mã hóa UTF-8.

Tiếp theo, ta tạo một đối tượng RestTemplate và thiết lập HttpComponentsClientHttpRequestFactory làmphương thức gửi yêu cầu HTTP mặc định cho RestTemplate. Tiếp theo, ta lấy danh sách các HttpMessageConverter của RestTemplate và thêm textHttpMessageConverter vào cuối danh sách và stringHttpMessageConverter vào đầu danh sách. Cuối cùng, ta đặt lại danh sách HttpMessageConverter cho RestTemplate và trả về nó.

1. @Bean public RetryTemplate retryTemplate(): Đây là một phương thức khởi tạo một bean RetryTemplate, đại diện cho cấu hình cho việc thử lại trong ứng dụng. Trong phương thức này, ta tạo một đối tượng RetryTemplate và cấu hình nó. Đầu tiên, ta tạo một FixedBackOffPolicy với khoảng thời gian chờ là 2 giây. Sau đó, ta tạo một SimpleRetryPolicy với số lần thử lại tối đa là 5. Cuối cùng, ta thiết lập BackOffPolicy và RetryPolicy cho RetryTemplate và trả về nó.

Tóm lại, lớp RestTemplateConfig là một lớp cấu hình Spring Boot, cung cấp cấu hình cho RestTemplate và RetryTemplate trong ứng dụng. RestTemplate được cấu hình để hỗ trợ tin nhắn HTTP với định dạng JSON và chuỗi, cùng với một HttpComponentsClientHttpRequestFactory làm phương thức gửi yêu cầu mặc định. RetryTemplate được cấu hình để thực hiện thử lại yêu cầu với một khoảng thời gian chờ và số lần thử lại tùy chỉnh.

///////////////////////////////////////////////////////////////

Đoạn mã trên là một lớp cấu hình trong ứng dụng Java Spring Boot, được đánh dấu bằng @Configuration. Nó cấu hình ObjectMapper và ModelMapper trong ứng dụng.

1. @Bean public ObjectMapper objectMapper(): Đây là một phương thức khởi tạo một bean ObjectMapper, đại diện cho một đối tượng dùng để chuyển đổi giữa JSON và Java objects. Trong phương thức này, ta tạo một đối tượng ObjectMapper mới và đăng ký một module JavaTimeModule. Module này cung cấp hỗ trợ cho các lớp thời gian và ngày tháng trong Java 8+ khi chuyển đổi sang JSON và ngược lại. Sau đó, ta trả về đối tượng ObjectMapper.
2. @Bean public ModelMapper modelMapper(): Đây là một phương thức khởi tạo một bean ModelMapper, đại diện cho một thư viện chuyển đổi giữa các đối tượng Java. Trong phương thức này, ta tạo một đối tượng ModelMapper mới. Ta cũng tạo một TypeMap để định nghĩa quy tắc chuyển đổi giữa ExperimentDetailVo và Experiment. Sau đó, ta sử dụng phương thức skip để bỏ qua các trường trong Experiment khi thực hiện chuyển đổi. Cụ thể, các trường testers, rules, variants, siLogs, và measures trong Experiment sẽ không được sao chép từ ExperimentDetailVo. Tiếp theo, ta đặt chiến lược phù hợp (MatchingStrategies.STRICT) và điều kiện thuộc tính (Conditions.isNotNull()) cho ModelMapper. Cuối cùng, ta trả về đối tượng ModelMapper.

Tóm lại, lớp RootConfig là một lớp cấu hình Spring Boot, cung cấp cấu hình cho ObjectMapper và ModelMapper trong ứng dụng. ObjectMapper được cấu hình để hỗ trợ chuyển đổi JSON và Java objects, đặc biệt là các lớp thời gian và ngày tháng trong Java 8+. ModelMapper được cấu hình để chuyển đổi giữa các đối tượng Java, đồng thời loại bỏ các trường không cần thiết trong quá trình chuyển đổi.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Đoạn mã trên là một lớp DateUtils trong ứng dụng Java, được sử dụng để thực hiện các phép tính và chuyển đổi liên quan đến ngày tháng.

1. public static Date toFirstDayOfMonth(YearMonth yearMonth): Phương thức này nhận một đối tượng YearMonth và trả về Date đại diện cho ngày đầu tiên của tháng tương ứng. Nó sử dụng phương thức atDay(1) của yearMonth để lấy ngày đầu tiên của tháng, sau đó chuyển đổi nó thành Date bằng cách sử dụng atStartOfDay(ZoneId.systemDefault()).toInstant().
2. public static YearMonth toYearMonth(Date date): Phương thức này nhận một đối tượng Date và trả về YearMonth tương ứng. Nó sử dụng Calendar để lấy thông tin về năm và tháng từ date, sau đó tạo một đối tượng YearMonth từ thông tin đó.
3. public static Date parse(String dateString): Phương thức này nhận một chuỗi đại diện cho ngày tháng (theo định dạng chuẩn ISO yyyy-MM-dd) và trả về Date tương ứng. Nó sử dụng LocalDate.parse() để chuyển đổi chuỗi thành LocalDate, sau đó chuyển đổi nó thành Date bằng atStartOfDay(ZoneId.systemDefault()).toInstant().
4. public static boolean isLastDayOfMonth(Date date): Phương thức này kiểm tra xem date có phải là ngày cuối cùng của tháng hay không. Nó sử dụng Calendar để lấy thông tin về ngày trong tháng và so sánh nó với calendar.getActualMaximum(Calendar.DATE) (ngày tối đa trong tháng). Nếu hai giá trị này bằng nhau, tức là date là ngày cuối cùng của tháng.
5. public static String format(Date date, SimpleDateFormat dateFormat): Phương thức này nhận một Date và một đối tượng SimpleDateFormat (định dạng ngày tháng) tùy chọn và trả về chuỗi đại diện cho ngày tháng được định dạng. Nếu dateFormat là null, một đối tượng SimpleDateFormat với định dạng mặc định "yyyy-MM-dd" được sử dụng.
6. public static Date addDate(Date date, int amount): Phương thức này nhận một Date và một số nguyên amount, và trả về Date tương ứng sau khi thêm amount ngày vào date. Nó sử dụng Calendar để thực hiện phép tính này.
7. public static Date addMonth(Date date, int amount): Phương thức này nhận một Date và một số nguyên amount, và trả về Date tương ứng sau khi thêm amount tháng vào date. Nó cũng sử dụng Calendar để thực hiện phép tính này.
8. public static String toCron(Date date): Phương thức này nhận một Date và trả về một chuỗi biểu diễn date theo định dạng cron (thường được sử dụng để lập lịch). Nó sử dụng Calendar để lấy thông tin về giây, phút, giờ, ngày trong tháng, tháng và năm từ date, sau đó sử dụng String.format() để tạo chuỗi biểu diễn cron.

Tóm lại, lớp DateUtils cung cấp các phương thức tiện ích để làm việc với ngày tháng trong Java, bGiúp bạn có thể thực hiện các phép tính và chuyển đổi ngày tháng, bao gồm việc lấy ngày đầu tháng, chuyển đổi giữa YearMonth và Date, phân tích chuỗi ngày tháng, kiểm tra xem một ngày có phải là ngày cuối tháng, định dạng ngày tháng thành chuỗi, thêm ngày và tháng vào một ngày cụ thể, và chuyển đổi ngày thành chuỗi biểu diễn cron.

////////////////////////////////

Đoạn mã trên là một lớp ExcelUtils trong ứng dụng Java, được sử dụng để hỗ trợ xuất dữ liệu vào file Excel và tạo các kiểu dữ liệu cụ thể cho các ô trong file Excel.

1. public static void export(String fileName, XSSFWorkbook workbook, HttpServletResponse response): Phương thức này nhận tên tệp, đối tượng XSSFWorkbook (đại diện cho workbook Excel) và đối tượng HttpServletResponse (để gửi workbook xuống khách hàng) và xuất workbook vào file Excel. Nó thiết lập các thông số cần thiết cho phản hồi HTTP, bao gồm đặt tiêu đề "Content-Disposition" để xác định tên tệp, đặt kiểu nội dung thành "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" để chỉ định loại file Excel và ghi workbook vào đầu ra phản hồi. Sau đó, nó đóng workbook và xả bộ đệm của phản hồi.
2. public static CellStyle createHeadCellStyle(XSSFWorkbook workbook): Phương thức này tạo và trả về một đối tượng CellStyle để định dạng phong cách cho các ô tiêu đề trong workbook. Nó thiết lập căn giữa ngang và căn giữa dọc, cho phép các dòng văn bản quá dài được xuống dòng, và thiết lập phông chữ in đậm với màu chữ đen.
3. public static CellStyle createTextCellStyle(XSSFWorkbook workbook): Phương thức này tạo và trả về một đối tượng CellStyle để định dạng phong cách cho các ô chứa văn bản trong workbook. Nó thiết lập căn trái ngang và căn giữa dọc, cho phép các dòng văn bản quá dài được xuống dòng.
4. public static CellStyle createNumberCellStyle(XSSFWorkbook workbook): Phương thức này tạo và trả về một đối tượng CellStyle để định dạng phong cách cho các ô chứa số trong workbook. Nó thiết lập căn phải ngang và căn giữa dọc, và định dạng dữ liệu số thành "#,##0" để hiển thị số nguyên và có dấu phẩy ngăn cách.
5. public static CellStyle createFloatCellStyle(XSSFWorkbook workbook): Phương thức này tạo và trả về một đối tượng CellStyle để định dạng phong cách cho các ô chứa số dấu phẩy động trong workbook. Nó thiết lập căn phải ngang và căn giữa dọc, và định dạng dữ liệu số thành "0.00" để hiển thị số dấu phẩy động với hai chữ số thập phân.
6. public static CellStyle createDoubleCellStyle(XSSFWorkbook workbook): Phương thức này tạo và trả về một đối tượng CellStyle để định dạng phong cách cho các ô chứa số double trong workbook. Nó thiết lập căn phải ngang và căn giữa dọc, và định dạng dữ liệu số thành "0.00000" để hiển thị số double với năm chữ số thập phân.

Tóm lại, lớp ExcelUtils cung cấp các phương thức tiện ích để hỗ trợ việc xuất dữ liệu vào file Excel và tạo các kiểu dữ liệu cụ thể cho các ô trong file Excel, bao gồm định dạng phong cách cho các ô tiêu đề, văn bản, số nguyên, số dấu phẩy động và số double.

/////////////////////////////////////////////////////////

Lớp LogUtils trong đoạn mã trên là một lớp tiện ích trong ứng dụng Java, được sử dụng để tạo chuỗi log dựa trên các đối tượng Collection hoặc Map. Nó cung cấp các phương thức để tạo chuỗi log từ danh sách các phần tử hoặc bản đồ.

1. private static final int LOG\_LIST\_ITEM\_COUNT = 10;: Đây là một hằng số được định nghĩa là một số nguyên, với giá trị là 10. Hằng số này được sử dụng để xác định số lượng phần tử tối đa trong một danh sách hoặc bản đồ khi tạo chuỗi log.
2. public static String log(Collection<?> list): Phương thức này nhận một đối tượng Collection và tạo chuỗi log từ các phần tử trong danh sách. Mặc định, nó sử dụng giá trị của LOG\_LIST\_ITEM\_COUNT làm giới hạn số lượng phần tử trong danh sách.
3. public static String log(Collection<?> list, int count): Phương thức này nhận một đối tượng Collection và một số nguyên count và tạo chuỗi log từ các phần tử trong danh sách. Tham số count xác định số lượng tối đa của các phần tử sẽ được hiển thị trong chuỗi log.
4. public static String log(Map<?, ?> list): Phương thức này nhận một đối tượng Map và tạo chuỗi log từ các cặp key-value trong bản đồ. Mặc định, nó sử dụng giá trị của LOG\_LIST\_ITEM\_COUNT làm giới hạn số lượng cặp key-value trong bản đồ.
5. public static String log(Map<?, ?> list, int count): Phương thức này nhận một đối tượng Map và một số nguyên count và tạo chuỗi log từ các cặp key-value trong bản đồ. Tham số count xác định số lượng tối đa của các cặp key-value sẽ được hiển thị trong chuỗi log.

Các phương thức log trên kiểm tra nếu danh sách hoặc bản đồ list là null hoặc kích thước của nó nhỏ hơn hoặc bằng count, thì trả về chuỗi rỗng kèm theo danh sách hoặc bản đồ đó. Nếu không, nó trả về chuỗi "size : " kèm theo kích thước của danh sách hoặc bản đồ. Điều này giúp giới hạn kích thước của chuỗi log, đặc biệt khi danh sách hoặc bản đồ rất lớn, nhằm tránh việc log quá nhiều dữ liệu và làm chậm hệ thống.

///////////////////////////////////////////////////////

Lớp PageUtils trong đoạn mã trên là một lớp tiện ích trong ứng dụng Java, được sử dụng để chuyển đổi một danh sách các phần tử thành một đối tượng Page. Nó cung cấp các phương thức để thực hiện việc này.

1. public static <T> Page<T> convertToPage(@NonNull List<T> content, @NonNull Pageable pageable): Phương thức này nhận một danh sách các phần tử content và một đối tượng Pageable để xác định các thông số phân trang. Nó chuyển đổi danh sách content thành một đối tượng Page<T> với các phần tử trên trang được xác định bởi pageable.
2. public static <T> Page<T> convertToPage(@NonNull List<T> content, @NonNull Pageable pageable, long total): Phương thức này nhận một danh sách các phần tử content, một đốiPhương thức này nhận danh sách content là một danh sách các phần tử và đối tượng Pageable để xác định các thông số phân trang. Nó chuyển đổi danh sách content thành một đối tượng Page<T> với các phần tử trên trang được xác định bởi pageable.

Để thực hiện việc chuyển đổi, phương thức tính toán các chỉ số from và to để xác định phần danh sách content sẽ được lấy ra để tạo thành trang. Chúng được tính dựa trên thông tin trong đối tượng Pageable và kích thước của danh sách content.

* Chuyển đổi pageable.getOffset() thành một số nguyên from bằng cách chuyển đổi giá trị offset thành int.
* Kiểm tra nếu from lớn hơn kích thước của danh sách content. Nếu đúng, có nghĩa là chỉ số bắt đầu lớn hơn số lượng phần tử trong danh sách. Trong trường hợp này, trả về một trang trống mới với Collections.emptyList() và pageable ban đầu, cùng với total để xác định tổng số phần tử có sẵn.
* Chuyển đổi from + pageable.getPageSize() thành một số nguyên to bằng cách chuyển đổi tổng của from và pageSize thành int.
* Kiểm tra nếu to lớn hơn kích thước của danh sách content. Nếu đúng, chỉnh to thành bằng kích thước của danh sách content, để đảm bảo không vượt quá số lượng phần tử trong danh sách.
* Trích xuất một phần danh sách từ from đến to bằng cách sử dụng content.subList(from, to).
* Trả về một đối tượng PageImpl mới với phần danh sách đã trích xuất, pageable ban đầu và total để xác định tổng số phần tử có sẵn.

Tóm lại, lớp PageUtils cung cấp các phương thức để chuyển đổi một danh sách các phần tử thành một đối tượng Page, dựa trên thông tin phân trang được xác định bởi đối tượng Pageable.