[Bài đọc] Annotated Controllers

Ứng dụng có thể sử dụng các lớp @Controller để xử lý các thông tin được gửi từ client. Các lớp có thể định nghĩa các phương thức ở phần tiếp theo như @MessageMapping, @SubscribeMapping và @ExceptionHandler

 @MessageMapping

Annotation @MessageMapping có thể sử dụng trong phương thức để định tuyến các thông tin (messages) dựa trên đích đến của nó. Annotation này sử dụng ở mức phương thức  cũng như ở mức loại (type). Ở mức loại (type) @MessageMapping sử dụng để chia sẻ ánh xạ thông qua tất cả phương thức trong controller.

Các  ánh xạ mặc định của đích mong muốn là kiểu “Ant-style”, các mẫu đường dẫn như "/foo\*", "/foo/\*\*". Các mẫu bao gồm hỗ trợ các biến như  "/foo/{id}", biến này có thể tham chiếu với các đối số của phương thức @DestinationVariable.

|  |
| --- |
| Ứng dụng có thể chọn chuyển sang quy ước kiểu dấu chấm “**.**”. Cụ thể xem links sau [Dot as Separator](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#websocket-stomp-destination-separator). |

Phương thức @MessageMapping có các đối số linh hoạt như sau:

|  |  |
| --- | --- |
| **Đối số của phương thức** | **Mô tả** |
| Message | Cho truy cập message đầy đủ. |
| MessageHeaders | Cho truy cập header trong Message. |
| MessageHeaderAccessor, SimpMessageHeaderAccessor, StompHeaderAccessor | Để truy cập tới header thông qua các phương thức truy cập đã nhập. |
| @Payload | Để truy cập tới payload của message, chuyển đổi (ví dụ: từ JSON) thông qua một MessageConverter đã cấu hình sẵn.  Annotation này không bắt buộc sử dụng vì nó được gán mặc định nếu không có đối số nào phù hợp.  Đối số Payload có thể sử dụng cùng với @javax.validation.Valid hoặc Spring’s @Validated để được kiểm tra hợp lệ tự động. |
| @Header | Để truy cập tới giá trị header cụ thể cùng với chuyển đổi kiểu bằng cách sử dụng một org.springframework.core.convert.converter.Converter nếu cần thiết. |
| @Headers | Để truy cập tới toàn bộ headers trong message. Đối số này phải được gán cho java.util.Map. |
| @DestinationVariable | Để truy cập tới biến temp được trích xuất từ các message đích. Giá trị sẽ bị chuyển đổi thành đối số khai báo trong phương thức nếu cần thiết. |
| java.security.Principal | Phản ánh người dùng đăng nhập vào thời điểm WebSocket HTTP handshake. |

Khi một phương thức @MessageMapping trả về một giá trị thì mặc định giá trị tuần tự trở thành một payload (dữ liệu vận chuyển ) thông qua một cấu hình MessageConverter,sau đó gửi dưới dạng Message tới “brokerChannel” từ broadcast tới người đăng ký. Điểm đến của message gửi đi giống với message đến nhưng được đặt trước bởi "/topic".

Bạn có thể sử dụng annotation phương thức  @SendTo để tùy chỉnh đích gửi payload. . @SendTo cũng có thể sử dụng ở mức lớp để chua sẽ một đích mặc định để gửi messages đến. @SendToUser là một biến thể chỉ để để gửi messages  tới người dùng được liên kết với một message. Có thể xem links sau User Destinations

Giá trị trả về từ một phương thức @MessageMapping có thể được bao bọc bởi  ListenableFuture,CompletableFuture, hoặc CompletionStage để payload không đồng bộ.

Để thay việc trả về một payload từ một phương thức @MessageMapping bạn có cũng có thể sử dụng SimpMessagingTemplate để gửi messages.  SimpMessagingTemplate cũng là cách trả giá trị được xử lý bao phủ. Cụ thể xem links sau :[Send Messages](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#websocket-stomp-handle-send).

@SubscribeMapping

Annotation @SubscribeMapping được sử dụng kết hợp với  @MessageMapping trong việc hạn chế ánh xạ tới các messages đăng ký. Trong kịch bản này thì annotation @MessageMapping chỉnh định đích còn @SubscribeMapping chỉ quan tâm đến các messages đăng ký.

Một phương thức @SubscribeMapping thường không có sự khác biệt với bất kì phương thức @MessageMapping về ánh xạ và đối số đầu vào. Ví dụ bạn có thể kết hợp chúng với một @MessageMapping mức kiểu (type-level)để hiển thị tiền tố được chia xẻ, và bạn cũng có thể sử dụng cùng đối số phương thức ([method arguments](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#websocket-stomp-message-mapping)) với bất kỳ phương thức @MessageMapping nào.

Sự khác biệt chính giữa  @SubscribeMapping là giá trị trả về của phương thức là được tuần tự hóa dưới một payload  và được gửi đi, chứ không phải là "brokerChannel" nhưng đến  "clientOutboundChannel",  trả lời trực tiếp tới client là cách hiệu quả hơn là thông qua trung gian. Điều này rất hữu dụng trong việc trả lời một lần rồi thôi, tức là trả lời trả lời tin nhắn rồi không bao giờ giữ lại đăng ký. Một kịch bản phổ biến để sử dụng

Một phương thức  @SubscribeMapping cũng có thể chú thích bởi @SendTo trong trường hợp trả về giá trị được gửi đến "brokerChannel" với đích được chỉ định rõ ràng.

@MessageExceptionHandler

Một ứng dụng có thể sử dụng phương thức @MessageExceptionHandler để xử lý các ngoại lệ từ phương thức @MessageMapping. Ngoại lệ có thể được định nghĩa trong chính nó hoặc thông qua đối số của phương thức nếu bạn cần truy cập đến thể hiện ngoại lệ.

@Controller  
**public class**MyController {  
  
*// ...*@MessageExceptionHandler  
**public**ApplicationError handleException(MyException exception) {  
*// ...***return**appError;  
}  
}

Phương thức @MessageExceptionHandler hỗ trợ linh hoạt method signatures và hỗ trợ cùng kiểu đối số của phương thức và trả giá trị như các phương thức [@MessageMapping](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#websocket-stomp-message-mapping).

Các phương thức @MessageExceptionHandler thường được áp dụng trong lớp @Controller (hoặc phân cấp lớp) mà chúng được khai báo.Nếu bạn muốn các phương thức như vậy áp dụng toàn cục hoặc trên các controller, bạn có thể định nghĩa chúng trong một lớp được chú thích bằng @ControllerAdvice. Để so sánh các hỗ trợ tương tự trong Spring MVC xem links sau: [similar support](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-controller-advice)

Last modified: Tuesday, 22 January 2019, 11:38 AM