<http://www.baeldung.com/exception-handling-for-rest-with-spring>

<https://gist.github.com/matsev/4519323#file-globalcontrollerexceptionhandler-java>

http://www.baeldung.com/spring-tutorial

***@ControllerAdvice* (Spring 3.2 And Above)**

**Spring 3.2** brings support for a global *@ExceptionHandler*with the new *@ControllerAdvice* annotation. This enables a mechanism that breaks away from the older MVC model and makes use of *ResponseEntity* along with the type safety and flexibility of *@ExceptionHandler*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @ControllerAdvice  public class RestResponseEntityExceptionHandler extends ResponseEntityExceptionHandler {        @ExceptionHandler(value = { IllegalArgumentException.class, IllegalStateException.class })      protected ResponseEntity<Object> handleConflict(RuntimeException ex, WebRequest request) {          String bodyOfResponse = "This should be application specific";          return handleExceptionInternal(ex, bodyOfResponse,            new HttpHeaders(), HttpStatus.CONFLICT, request);      }  } |

The new annotation allows the multiple scattered *@ExceptionHandler* from before to be consolidated into a **single, global error handling component**.

The actual mechanism is extremely simple but also very flexible:

* it allows full control over the body of the response as well as the status code
* it allows mapping of several exceptions to the same method, to be handled together
* it makes good use of the newer RESTful *ResposeEntity* response

One thing to keep in mind here is to **match the exceptions declared with *@ExceptionHandler* with the exception used as the argument of the method**. If these don’t match, the compiler will not complain – no reason it should, and Spring will not complain either.

However, when the exception is actually thrown at runtime, **the exception resolving mechanism will fail with**:

|  |  |
| --- | --- |
| 1  2 | java.lang.IllegalStateException: No suitable resolver for argument [0] [type=...]  HandlerMethod details: ... |

## ****Handle the Access Denied in Spring Security****

The Access Denied occurs when **an authenticated user** tries to access resources that he doesn’t have enough authorities to access.

### **5.1. MVC – Custom Error Page**

First, let’s look at the MVC style of the solution and see how to customize an error page for Access Denied:

The**XML configuration**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <http>      <intercept-url pattern="/admin/\*" access="hasAnyRole('ROLE\_ADMIN')"/>      ...      <access-denied-handler error-page="/my-error-page" />  </http> |

And the **Java configuration**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Override  protected void configure(HttpSecurity http) throws Exception {      http.authorizeRequests()          .antMatchers("/admin/\*").hasAnyRole("ROLE\_ADMIN")          ...          .and()          .exceptionHandling().accessDeniedPage("/my-error-page");  } |

When users try to access a resource without having enough authorities, they will be redirected to “/my-error-page“.

### **5.2. Custom**AccessDeniedHandler

Next, let’s see how to write our custom AccessDeniedHandler:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @Component  public class CustomAccessDeniedHandler implements AccessDeniedHandler {        @Override      public void handle        (HttpServletRequest request, HttpServletResponse response, AccessDeniedException ex)        throws IOException, ServletException {          response.sendRedirect("/my-error-page");      }  } |

And now let’s configure it using **XML Configuration**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <http>      <intercept-url pattern="/admin/\*" access="hasAnyRole('ROLE\_ADMIN')"/>      ...      <access-denied-handler ref="customAccessDeniedHandler" />  </http> |

Or using Java Configuration:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Autowired  private CustomAccessDeniedHandler accessDeniedHandler;    @Override  protected void configure(HttpSecurity http) throws Exception {      http.authorizeRequests()          .antMatchers("/admin/\*").hasAnyRole("ROLE\_ADMIN")          ...          .and()          .exceptionHandling().accessDeniedHandler(accessDeniedHandler)  } |

Note how – in our CustomAccessDeniedHandler, we can customize the response as we wish by redirecting or display a custom error message.

### **5.3. REST and Method Level Security**

Finally, let’s see how to handle method level security @PreAuthorize, @PostAuthorize, and @Secure Access Denied.

We’ll, of course, use the global exception handling mechanism that we discussed earlier to handle the new AccessDeniedException as well:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @ControllerAdvice  public class RestResponseEntityExceptionHandler extends ResponseEntityExceptionHandler {        @ExceptionHandler({ AccessDeniedException.class })      public ResponseEntity<Object> handleAccessDeniedException(Exception ex, WebRequest request) {          return new ResponseEntity<Object>(            "Access denied message here", new HttpHeaders(), HttpStatus.FORBIDDEN);      }        ...  } |

## @ControllerAdvice

The [@ControllerAdvice](http://static.springsource.org/spring/docs/3.2.x/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html) is a new [TYPE](http://docs.oracle.com/javase/7/docs/api/java/lang/annotation/ElementType.html?is-external=true#TYPE) annotation that was added as part of the release. A class annotated with it will act as a global helper class for all controllers. In other words, any local, controller specific @ExceptionHandler method that is moved from the @Controller class to a class annotated with @ControllerAdvice, will be applicable for the entire application.

|  |  |
| --- | --- |
|  | @ControllerAdvice |
|  | class GlobalControllerExceptionHandler { |
|  |  |
|  | @ExceptionHandler |
|  | @ResponseStatus(HttpStatus.BAD\_REQUEST) |
|  | @ResponseBody |
|  | ErrorMessage handleException(SomeException ex) { |
|  | ErrorMessage errorMessage = createErrorMessage(ex); |
|  | return errorMessage; |
|  | } |
|  |  |
|  | @ExceptionHandler |
|  | @ResponseStatus(HttpStatus.GONE) |
|  | @ResponseBody |
|  | ErrorMessage handleException(OtherException ex) { |
|  | ErrorMessage errorMessage = createErrorMessage(ex); |
|  | return errorMessage; |
|  | } |
|  | } |

The previous posts contained error handling examples of some Spring MVC exceptions, namely the MethodArgumentNotValidException, the HttpMediaTypeNotSupportedException and the HttpMessageNotReadableException. A corresponding Spring 3.2 based implementation can be found in the [GlobalControllerExceptionHandler](https://gist.github.com/4519323#file-globalcontrollerexceptionhandler-java), or continue reading for yet another implementation.

## ResponseEntityExceptionHandler

The above example works well, but it can be hard to identify the Spring MVC specific exceptions to implement a common error response handling strategy for them. One way of overcoming this problem is to extend the [ResponseEntityExceptionHandler](http://static.springsource.org/spring/docs/3.2.x/javadoc-api/org/springframework/web/servlet/mvc/method/annotation/ResponseEntityExceptionHandler.html)class, which was also added to the Spring 3.2 release. Similarly to the [DefaultHandlerExceptionResolver](http://static.springsource.org/spring/docs/3.2.x/javadoc-api/org/springframework/web/servlet/mvc/support/DefaultHandlerExceptionResolver.html), it provides methods for handling the exceptions, but it allows the developer to specify [ResponseEntity](http://static.springsource.org/spring/docs/3.2.x/javadoc-api/org/springframework/http/ResponseEntity.html)s as return values (as opposed to the [ModelAndView](http://static.springsource.org/spring/docs/3.2.x/javadoc-api/org/springframework/web/servlet/ModelAndView.html)s that are returned by the methods in the DefaultHandlerExceptionResolver). The implementation is still straight forward, create a class, annotate it with the @ControllerAdvice, extend the ResponseEntityExceptionHandler class and override the methods with the exception types that you are interested in:

|  |  |
| --- | --- |
|  | @ControllerAdvice |
|  | public class CustomResponseEntityExceptionHandler extends ResponseEntityExceptionHandler { |
|  |  |
|  | @Override |
|  | protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) { |
|  | List<FieldError> fieldErrors = ex.getBindingResult().getFieldErrors(); |
|  | List<ObjectError> globalErrors = ex.getBindingResult().getGlobalErrors(); |
|  | List<String> errors = new ArrayList<>(fieldErrors.size() + globalErrors.size()); |
|  | String error; |
|  | for (FieldError fieldError : fieldErrors) { |
|  | error = fieldError.getField() + ", " + fieldError.getDefaultMessage(); |
|  | errors.add(error); |
|  | } |
|  | for (ObjectError objectError : globalErrors) { |
|  | error = objectError.getObjectName() + ", " + objectError.getDefaultMessage(); |
|  | errors.add(error); |
|  | } |
|  | ErrorMessage errorMessage = new ErrorMessage(errors); |
|  | return new ResponseEntity(errorMessage, headers, status); |
|  | } |
|  |  |
|  | @Override |
|  | protected ResponseEntity<Object> handleHttpMediaTypeNotSupported(HttpMediaTypeNotSupportedException ex, HttpHeaders headers, HttpStatus status, WebRequest request) { |
|  | String unsupported = "Unsupported content type: " + ex.getContentType(); |
|  | String supported = "Supported content types: " + MediaType.toString(ex.getSupportedMediaTypes()); |
|  | ErrorMessage errorMessage = new ErrorMessage(unsupported, supported); |
|  | return new ResponseEntity(errorMessage, headers, status); |
|  | } |
|  |  |
|  | @Override |
|  | protected ResponseEntity<Object> handleHttpMessageNotReadable(HttpMessageNotReadableException ex, HttpHeaders headers, HttpStatus status, WebRequest request) { |
|  | Throwable mostSpecificCause = ex.getMostSpecificCause(); |
|  | ErrorMessage errorMessage; |
|  | if (mostSpecificCause != null) { |
|  | String exceptionName = mostSpecificCause.getClass().getName(); |
|  | String message = mostSpecificCause.getMessage(); |
|  | errorMessage = new ErrorMessage(exceptionName, message); |
|  | } else { |
|  | errorMessage = new ErrorMessage(ex.getMessage()); |
|  | } |
|  | return new ResponseEntity(errorMessage, headers, status); |
|  | } |
|  | } |