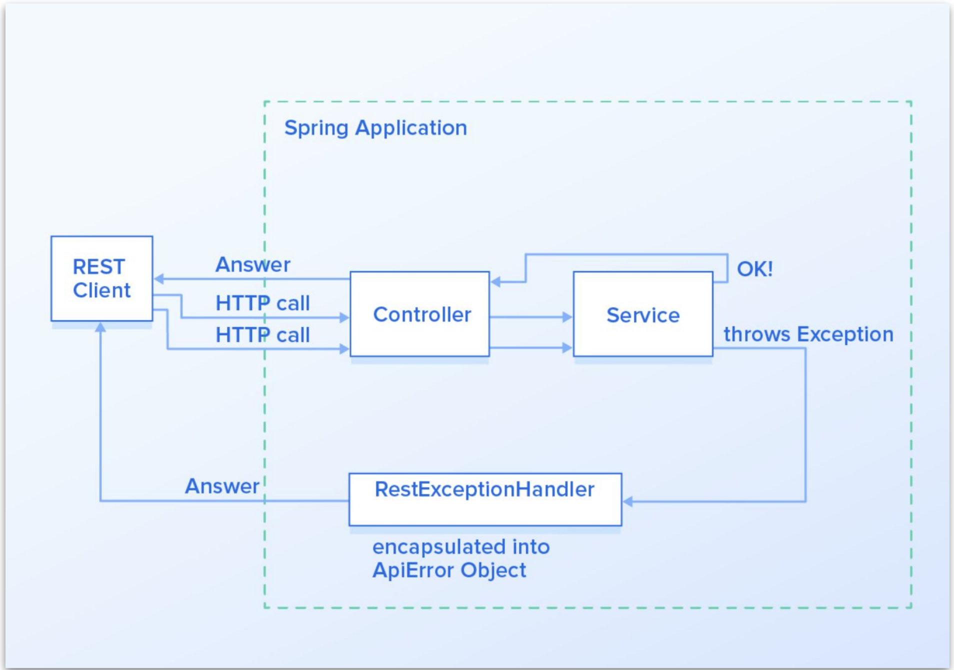
Exception Handling

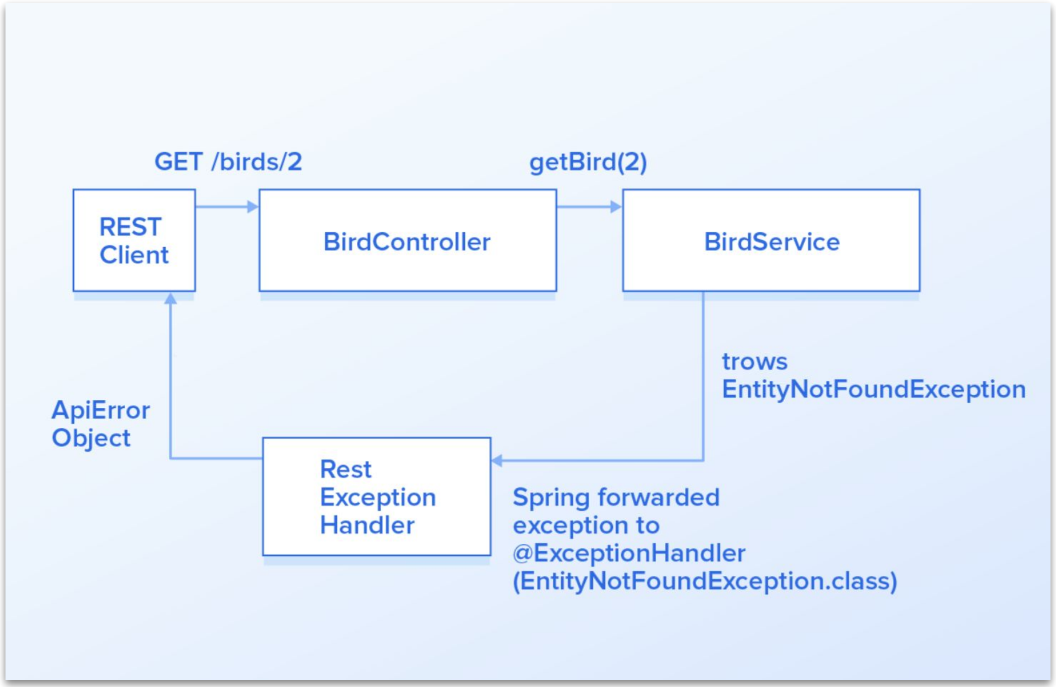
Content

Exception Handling in Spring



https://www.toptal.com/java/spring-boot-rest-api-error-handling

Exception Handling in Spring


GET /birds/2
getBird(2)
REST Client
BirdController
BirdService
trows EntityNotFoundException
ApiError Object
Rest Exception Handler
Spring forwarded exception to @ExceptionHandler (EntityNotFoundException.class)

https://www.toptal.com/java/spring-boot-rest-api-error-handling

Error Handling for REST

1. Controller level @ExceptionHandler
2. HandlerExceptionResolver
   a. DefaultHandlerExceptionResolver
   b. ResponseStatusExceptionResolver
3. @ControllerAdvice
4. ResponseStatusException (Spring 5 and Above)

Controller level @ExceptionHandler

```
@Controller
@RequestMapping(path = "/api/exception")
public class ExceptionController {

   @ExceptionHandler({NoSuchElementException.class, JsonMappingException.class})
   public ModelAndView handleException(Exception exception) {
      ModelAndView modelAndView = new ModelAndView();
      modelAndView.setViewName("custom-error");
      modelAndView.addObject("message", exception.getMessage());
      return modelAndView;
   }
}
```

HandlerExceptionResolver

DefaultHandlerExceptionResolver

1. This resolver was introduced in Spring 3.0, and it's enabled by default in the *DispatcherServlet*.
2. It's used to resolve standard Spring exceptions to their corresponding HTTP Status Codes, namely Client error – *4xx* and Server error – *5xx* status codes

http://localhost:8080/default-handler-exception-resolver/

https://github.com/nursultanturdaliev/money-transfer-app/commit/de5e8722a4bdd40f19b5336579846d63dad833d4

HandlerExceptionResolver: DefaultHandlerExceptionResolver

Exception	HTTP Status Code
`BindException`	400 (Bad Request)
`ConversionNotSupportedException`	500 (Internal Server Error)
`HttpMediaTypeNotAcceptableException`	406 (Not Acceptable)
`HttpMediaTypeNotSupportedException`	415 (Unsupported Media Type)
`HttpMessageNotReadableException`	400 (Bad Request)
`HttpMessageNotWritableException`	500 (Internal Server Error)
`HttpRequestMethodNotSupportedException`	405 (Method Not Allowed)
`MethodArgumentNotValidException`	400 (Bad Request)
`MissingServletRequestParameterException`	400 (Bad Request)
`MissingServletRequestPartException`	400 (Bad Request)
`NoSuchRequestHandlingMethodException`	404 (Not Found)
`TypeMismatchException`	400 (Bad Request)

HandlerExceptionResolver: ResponseStatusExceptionResolver

- Its main responsibility is to use the @ResponseStatus annotation available on custom exceptions and to map these exceptions to HTTP status codes.

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.CONFLICT)
public class UserResourceNotFoundException extends Exception {
  public UserResourceNotFoundException() {
     super();
  }
}
```

http://localhost:8080/response-status-exception-resolver/
https://github.com/nursultanturdaliev/money-transfer-app/commit/fdcebd27240cc673d7b4cb51a432f481633bdcbf

@ControllerAdvice @ExceptionHandler

```
@ControllerAdvice
class RestResponseEntityExceptionHandler
      extends ResponseEntityExceptionHandler {

   @ExceptionHandler(value = {RecordConflictException.class})
   public final ResponseEntity<ErrorResponse> handleUserNotFoundException(RecordConflictException ex) {
      ArrayList<String> details = new ArrayList<String>();
      details.add(ex.getLocalizedMessage());
      ErrorResponse error = new ErrorResponse(MoneyTransferAppApplication.RECORD_CONFLICT, details);
      return new ResponseEntity<>(error, HttpStatus.CONFLICT);
   }
}
```

http://localhost:8080/controller-advice-exception-handler/

https://github.com/nursultanturdaliev/money-transfer-app/commit/ac409512d9026d2ce1f1fde89250e2f0a4527aeb

@ControllerAdvice Advantages

- Full control over the body of the response as well as the status code
- Mapping of several exceptions to the same method, to be handled together, and
- It makes good use of the newer RESTful *ResponseEntity* response

ResponseStatusException (Spring 5 and Above)

```
@Controller
public class ResponseStatusExceptionController {

  @GetMapping(path = "/response-status-exception")
  public ResponseEntity<String> fetchAll() {
     throw new ResponseStatusException(
           HttpStatus.BANDWIDTH_LIMIT_EXCEEDED,
           "Custom bandwidth limit exceeded reason message");
  }
}
```

http://localhost:8080/response-status-exception

https://github.com/nursultanturdaliev/money-transfer-app/commit/a0269a68b4b746d87cc78fb4efbce181a66a458b

ResponseStatusException Benefits

- Excellent for prototyping: We can implement a basic solution quite fast
- One type, multiple status codes: One exception type can lead to multiple different responses. **This reduces tight coupling compared to the *@ExceptionHandler***
- We won't have to create as many custom exception classes
- **More control over exception handling** since the exceptions can be created programmatically

ResponseStatusException tradeoffs

- There's no unified way of exception handling: It's more difficult to enforce some application-wide conventions, as opposed to *@ControllerAdvice* which provides a global approach
- Code duplication: We may find ourselves replicating code in multiple controllers

Spring Boot Default Exception Handling

- Sending request with `Accept = text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3` returns html white label response
- Sending api request responds with JSON response body

```
{
   "timestamp": "2019-10-15T20:38:36.010+0000",
   "status": 404,
   "error": "Not Found",
   "message": "No message available",
   "path": "/asdf"
}
```

Spring Boot Configuring

- *server.error.whitelabel.enabled:* can be used to disable the Whitelabel Error Page and rely on the servlet container to provide an HTML error message
- *server.error.include-stacktrace:* with an *always* value, it includes the stacktrace in both the HTML and the JSON default response

Note

We can implement a *@ControllerAdvice* globally, but also *ResponseStatusException*s locally. However, we need to be careful: If the same exception can be handled in multiple ways, we may notice some surprising behavior. A possible convention is to handle one specific kind of exception always in one way.

Exercises

1. Remove not found responses from user endpoints and handle using exception handling
2. Handle JsonMappingException using @ExceptionHandler on user update endpoint
3. Handle EmptyResultDataAccessException using @ControllerAdvice

References

- https://www.baeldung.com/exception-handling-for-rest-with-spring
- https://www.baeldung.com/spring-boot-logging
- https://howtodoinjava.com/spring-core/spring-exceptionhandler-annotation/
- https://www.baeldung.com/spring-boot-custom-error-page
- https://howtodoinjava.com/spring-core/spring-exceptionhandler-annotation/
- https://www.baeldung.com/spring-response-status-exception
- https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc
- https://www.toptal.com/java/spring-boot-rest-api-error-handling