Spring MVC2 15 [API 예외 처리]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.27

API 예외 처리는?

API 예외처리시에는 기존 HTML에서 처리하는 것과 다르게 JSON 데이터를 보내주는 오류 응답에 대한 스펙을 정하고 데이터를 내려줘야 한다.

API 오류 페이지도 서블릿 오류 부터 시작해서 확인.

서블릿 오류 - error가 was 전달, sendError 확인

```
@Slf4j
@RestController
public class ApiExceptionController {

    @GetMapping("/api/member/{id}")
    public MemberDto getMEmber(@PathVariable("id") String id){

        if(id.equals("ex")){
            throw new RuntimeException("잘못된 사용자");
        }

        return new MemberDto(id, "hello" + id);
    }

    @Data
    @AllArgsConstructor
    static class MemberDto{
        private String memberId;
        private String name;
    }
}
```

```
public class WebServerCustomizer implements
WebServerFactoryCustomizer<ConfigurableWebServerFactory>
```

이 설정되어 있어서 오류 컨트롤러로 보내 재전송된다면.

→ 다른 설정을 안하고 오류페이지(HTML)을 돌려준다.

원하는 결과는 JSON의 정보를 담아 보내는 것.

해당 **Controller** 설계

```
@RequestMapping(value = "/error-page/500", produces =
MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Map<String, Object>> errerPage500Api(
       HttpServletRequest request, HttpServletResponse response){

   log.info("API errorPage 500");

   String ERROR_EXCEPTION = "javax.servlet.error.exception";
   String ERROR_STATUS_CODE = "javax.servlet.error.status_code";


   Map<String, Object> result = new HashMap<>();
   Exception ex = (Exception) request.getAttribute(ERROR_EXCEPTION);

   result.put("status", request.getAttribute(ERROR_STATUS_CODE));
   result.put("message", ex.getMessage());

   Integer statusCode = (Integer)
request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE);

   return new ResponseEntity<>(result, HttpStatus.valueOf(statusCode));
}
```

HTTP Header에 Accept 를 확인해서 produces 값을 우선순위 해서 매핑한다.

Accept : application/json 일 시에.

API 예외 처리 **-** 스프링 부트

아무 설정을 안하고 스프링 부트 자체적으로 처리하는 것처럼 에러를 발생시키면

Accept형식에 따라서 HTML, JSON 데이터를 내려주게 된다.

BasicErrorController 에 설정 되어있는 내부처리.

```
@RequestMapping(produces = MediaType.TEXT_HTML_VALUE)
public ModelAndView errorHtml(HttpServletRequest request, HttpServletResponse
response) {
  HttpStatus status = getStatus(request);
  Map<String, Object> model = Collections
        .unmodifiableMap(getErrorAttributes(request,
                         getErrorAttributeOptions(request, MediaType.TEXT_HTML)));
  response.setStatus(status.value());

  ModelAndView modelAndView = resolveErrorView(request, response, status, model);
  return (modelAndView != null) ? modelAndView : new ModelAndView("error", model);
}
```

```
@RequestMapping
public ResponseEntity<Map<String, Object>> error(HttpServletRequest request) {
  HttpStatus status = getStatus(request);
  if (status == HttpStatus.NO_CONTENT) {
     return new ResponseEntity<>(status);
  }
  Map<String, Object> body = getErrorAttributes(request,
getErrorAttributeOptions(request, MediaType.ALL));
  return new ResponseEntity<>(body, status);
}
```

Accept : application/json 일 경우 확인

→ 이렇게 간단한 오류 처리는 기존에 정의 되어있는 것을 사용하면 된다.

HTTP 페이지 **- BasicErrorController** 가 편리하지만 **API** 경우 매번 다른 스펙에 맞는 **JSON**을 보내줘야 할 수 있어야 하기 때문에 **@ExceptionHandler**를 사용하는 것이 편리하다**.**

HandlerExceptionResolver

만약 WAS 까지 올라간 에러(500) 을 상황에 따라 다른 코드를 변경하기 위해서 ?

예) IllegalArgumentException → 잘못된 인자 전달 (사용자 , 400) ⇒ WAS (500) 으로 표출되는 것을 400으로 제대로 표현하고 싶어질 때.

스프링에서는

스프링 MVC는 핸들러 밖으로 예외가 던져진 경우 예외를 해결하고 동작을 새로 정의할 방법을 제공 → **HandlerExceptionResolver**

Custom HandlerExceptionResolver 설계 후 등록

```
@Slf4j
public class MyHandlerExceptionResolver implements HandlerExceptionResolver {
   @Override
   public ModelAndView resolveException(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {

       try {
           if( ex instanceof IllegalArgumentException){
               log.info("IllegalArgumentException resolver to 400");
               response.sendError(HttpServletResponse.SC_BAD_REQUEST,
ex.getMessage()); ⇒ sendError로 exception을 덮어준다. 그 후 정상 흐름처럼 변경
               return new ModelAndView();
           }
       } catch (IOException e) {
           log.error("resolver ex", e);
       }

       return null;  //null ==> 기존 에러를 가지고 진행
```

```
    }
}
```

등록

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void extendHandlerExceptionResolvers(List<HandlerExceptionResolver>
resolvers) {
        resolvers.add(new MyHandlerExceptionResolver());
    }
```

만약 **?**

빈 깡통의 ModelAndView() 를 반환 하면 정상처럼 동작하게 되는 것. 만약 View나 Model을 지정하면 View를 바로 렌더링 해준다.

null을 반환 하게 되면다면 ? 다음 ExceptionResolver 를 찾고 없다면 기존 발생한 예외를 던진다 (WAS 이상 진행시 - 500 처리)

활용법 **- ExceptionResolver**

1. 예외 상태 코드 변환 : 오류 코드를 변경해서 처리하게 동작시킴
2. 뷰 템플릿 처리 : 새로운 View 렌더링 시키는 것 가능
3. API 응답 처리 : response.getWriter().println("hello") 처럼 바로 응답 가능하고 그것을 JSON으로 처리하는 것도 가능하다.

HandlerExceptionResolver 활용

예외가 발생한 것을 **resolver**에서 마무리 시킬 수 있다**.**

→ 직접 구현해보기 **Json** 반환

```
@Slf4j
public class UserHandlerExceptionResolver implements HandlerExceptionResolver {

    private final ObjectMapper objectMapper = new ObjectMapper();

    @Override
```

```
   public ModelAndView resolveException(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) {

       try{
           if( ex instanceof UserException){
               log.info("UserException resolver to 400");
               String acceptHeader = request.getHeader("accept");
               response.setStatus(HttpServletResponse.SC_BAD_REQUEST);

               if("application/json".equals(acceptHeader)){
                   Map<String, Object> errorResult = new HashMap<>();
                   errorResult.put("ex", ex.getClass());
                   errorResult.put("message", ex.getMessage());

                   //json--> 문자로
                   String result = objectMapper.writeValueAsString(errorResult);

                   response.setContentType("application/json");
                   response.setCharacterEncoding("utf-8");
                   response.getWriter().write(result);

                   return new ModelAndView();
               }else{
                   //TEXT/HTML
                   return new ModelAndView("error/500");
               }
           }

       }catch (IOException e){
           log.error("resolver ex", e);
       }

       //null ==> 기존 에러를 가지고 진행
       return null;
   }
}
```

Accept : application/json 인 경우

response를 설정해서 ModelAndView를 넘겨준다.

스프링의 **ExceptionResolver**

스프링 부트는 ExceptionResolver는 아래 순서로 등록(실행)한다

- ExceptionHandlerExceptionResolver → 중요! 아래에서 따로 설명
- ResponseStatusExceptionResolver
- DefaultHandlerExceptionResolver

ResponseStatusExceptionResolver

```
@GetMapping("/api/response-statues-ex1")
public String responseStatusEx1(){
   throw new BadRequestException();
}
```

요청이 들어왔을 때 새로 생성한 Exception에 대해

```
@ResponseStatus(code = HttpStatus.BAD_REQUEST, reason = "잘못된 요청 오류")
public class BadRequestException extends RuntimeException{
}
```

@ResponseStatus에 의해 상태 코드가 변경되고 동작한다.

→ ResponseStatusExceptionResolver가 동작해서 상태코드를 변경해준다.

내부에서 sendError를 호출해준다.

```
@ResponseStatus(code = HttpStatus.BAD_REQUEST, reason = "error.bad")
public class BadRequestException extends RuntimeException{
}
```

reason을 메시지 소스를 확인해서 등록된 메시지로 사용할 수 있게 해준다**.**

messages.properties 생성 후 설정하면

```
error.bad=잘못된 요청 오류입니다. 메시지 properties 사용
```

만약 커스텀에러가 아닌 경우에는**?**

```
@GetMapping("/api/response-statues-ex2")
public String responseStatusEx2(){
   throw new ResponseStatusException(HttpStatus.NOT_FOUND, "error.bad", new
IllegalArgumentException());
}
```

ResponseStatusException 발생시키는 방법을 사용한다. ⇒ 더 알아볼 필요 있음

DefaultHandlerExceptionResolver

→ 스프링 내부에서 발생하는 스프링 예외를 해결한다

만약 ? 파라미터 바인딩 타입이 잘맞지않을때 에러가 발생되고 WAS 까지 올라가고 서버 잘못이라는 500가 응답되는데 DefaultHandlerExceptionResolver가 500 을 400대로 변경해준다.

예

```
@GetMapping("/api/default-handler-ex")
public String defaultException(@RequestParam Integer data){
   return "ok";
}
```

/api/default-handler-ex?data=12q 호출시

500이 아닌 400 BadRequest로 변경해준다.

DefaultHandlerExceptionResolver 가 작동해서 변경해준다.

HandlerExceptionResolver 을 직접 설계하는 것은 복잡하고 API 응답에는 ModelAndView를 반환하기에 response에 직접 작성하는 노가다성으로 코드를 작성하기 때문에 불편하다

그래서 스프링은 @ExceptionHandler 를 제공해서 혁신적으로 예외 처리 기능을 제공한다.

@ExceptionHandler

→ HTML 화면 오류는 BasicErrorController 를 사용하는게 편리하다.

API 예외 처리의 어려운 점

HandlerExceptionResolver – ModelAndView 를 반환하기 때문에 굉장히 불편

특정 컨트롤러에서만 발생하는 예외를 잡기 불편하다.

[상품 컨트롤러 , 주문 컨트롤러 에서 동일한 에러발생시 - 서로 다른 예외처리를 해야할 때 불편]

ExceptionHandlerExceptionResolver를 작동

@ExceptionHandler 를 컨트롤러에 적용시킨다.

```
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
@ExceptionHandler(IllegalArgumentException.class)
public ErrorResult illegalExHandler(IllegalArgumentException e){
   log.error("[exceptionHandler] ex", e);
   return new ErrorResult("BAD", e.getMessage());
}

@GetMapping("/api2/members/{id}")
public MemberDto getMEmber(@PathVariable("id") String id){
   if(id.equals("bad")){
       throw new IllegalArgumentException("잘못된 입력");
   }
```

→@ExceptionHandler(해당Exception.class)

Exception이 발생하면 해당 컨트롤러에서 해당 애노테이션이 걸린 핸들러를 찾고

바로 정상흐름으로 바꿔서 리턴해준다. 서블릿 컨테이너까지 올라가지 않고 정상적으로 호출해주는 것.

```
@ExceptionHandler → 파라미터와 같을땐 생략이 가능하다.
public ResponseEntity<ErrorResult> userExHandler(UserException e){
   log.error("[exceptionHandler] ex", e);
   ErrorResult errorResult = new ErrorResult("USER-EX", e.getMessage());
```

```
    return new ResponseEntity<>(errorResult, HttpStatus.BAD_REQUEST);
}
```

모든 **Exception**을 전부 받아주는 **Handler**

```
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
@ExceptionHandler
public ErrorResult exHandler(Exception e){
    log.error("[exceptionHandler] ex", e);
    return new ErrorResult("EX", "내부 오류");
}
```

→ 실수로 놓치거나 공통처리가 필요한 예외는 여기에서 잡아준다.

ExceptionHandler의 예외 처리 방법

컨트롤러 안에서 발생한 Exception에 대해서 지정한 예외와 그 자식 클래스를 모두 잡아서 처리해 준다. 우선 순위는 항상 자세한 것이 우선순위를 갖는다.

@ExceptionHanlder({ExceptionA e, ExceptionB e}) 형식으로 한개 이상의 예외도 한꺼번에 처리가가능하다.

메서드가 많은 파라미터를 받을 수 있다. (컨트롤러와 유사)

https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#mvc-annexceptionhandler-args

@ControllerAdvice

예외처리와 정상코드가 하나에 몰렸있는 문제를 해결해 주는 애노테이션

기존 예외처리(@ExceptionHandler) 가 포함 되어있는 컨트롤러에서

```
@Slf4j
@RestControllerAdvice
public class ExControllerAdvice {

    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
```

```
    @ExceptionHandler(IllegalArgumentException.class)
    public ErrorResult illegalExHandler(IllegalArgumentException e){
        log.error("[exceptionHandler] ex", e);
        return new ErrorResult("BAD", e.getMessage());
    }
}
```

RestControllerAdvice를 분리해서 따로 생성.

실행해도 똑같이 동작한다.

대상을 적용하기

대상을 적용안하면 globel로 지정된다.

```
@ControllerAdvice(annotations = RestController.class)
public class ExampleAdvice1 {}
→ 적용할 애노테이션 설정

@ControllerAdvice("org.example.controllers")
public class ExampleAdvice2 {}
→ 적용할 특정 패키지 지정
@ControllerAdvice(assignableTypes = {ControllerInterface.class,
        AbstractController.class})
public class ExampleAdvice3 {}
→ 적용할 클래스 지정
```

AOP와 동작방식이 유사 advice라는 표현이 aop에서 가져온것.