Spring MVC2 14 [예외 처리]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.27

서블릿 예외 처리

서블릿은 2가지 방식으로 예외처리를 한다

- Exception(예외 발생)
- response.sendError(HTTP 상태코드, 오류 메시지

Exception (예외)

- 문제가 생기면 쓰레드를 상위 로 올라가다가 최종적으로 올라가게 되면 스레드가 종료된다.

이러한 문제가 웹 애플리케이션에서

웹 애플리케이션은 사용자 별로 스레드가 할당되고 서블릿 컨테이너 안에서 실행됨

만약 컨트롤러에서 발생한 예외가 WAS 까지 전달 되면 어떻게 될까?

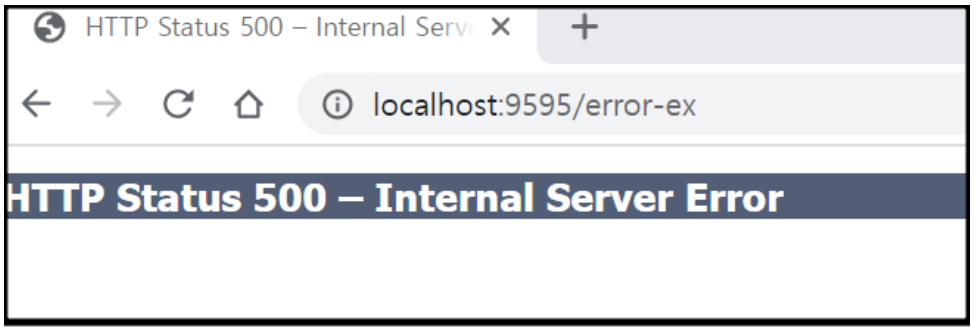
```
컨트롤러(예외 발생) → 인터셉터→  서블릿 → 필터 → WAS
```

was까지 올라가면 서버에서 처리할수 없구나 생각해서

500 에러를 내주고 페이지를 뿌린다.

```
@Controller
public class ServletExController {

   @GetMapping("/error-ex")
   public void errorEx(){
       throw new RuntimeException("예외 발생!");
   }
}
```



response.sendError() 로 직접 발생

```
@GetMapping("/error-404")
public void error404(HttpServletResponse response) throws IOException {
   response.sendError(404, "404 오류!");
}
@GetMapping("/error-500")
public void error500(HttpServletResponse response) throws IOException {
   response.sendError(500);
}
```

→ WAS에서 response의 저장된 sendError를 확인하고 오류페이지를 보여주게 된다.

오류 페이지 등록

과거 **web.xml**에서 오류 페이지 등록

```
50
51          <error-page>
52                  <error-code>404</error-code>
53                  <location>/error404</location>
54          </error-page>
55
56          <error-page>
57                  <error-code>405</error-code>
58                  <location>/error405</location>
59          </error-page>
60
61          <error-page>
62                  <error-code>500</error-code>
63                  <location>/error500</location>
64          </error-page>
65
66  </web-app>
```

https://github.com/Lece619/CampMarkett/blob/main/src/main/webapp/WEB-INF/web.xml

스프링 부트에서 등록

```
@Component
public class WebServerCustomizer implements
WebServerFactoryCustomizer<ConfigurableWebServerFactory> {

   @Override
   public void customize(ConfigurableWebServerFactory factory) {

       ErrorPage errorPage404 = new ErrorPage(HttpStatus.NOT_FOUND,
"/error-page/404");
       ErrorPage errorPage500 = new ErrorPage(HttpStatus.INTERNAL_SERVER_ERROR,
"/error-page/500");
       ErrorPage errorPageEx = new ErrorPage(RuntimeException.class,
"/error-page/500");

       factory.addErrorPages(errorPage404, errorPage500, errorPageEx);
   }
}
```

해당 에러가 발생시 WAS 지정한 URL을 다시 요청한다.

```
@Slf4j
@Controller
public class ErrorPageController {

   @RequestMapping("/error-page/404")
   public String errorPage404(HttpServletRequest request, HttpServletResponse
response){
       log.info("errorPage 404");
       return  "error-page/404";
   }

   @RequestMapping("/error-page/500")
   public String errorPage500(HttpServletRequest request, HttpServletResponse
response){
       log.info("errorPage 500");
       return  "error-page/500";
   }
}
```

요청한 **url** 매핑된 컨트롤러에 따른 오류 페이지가 보일것**.**

서블릿 예외 처리, 오류 페이지 작동 원리

WAS 까지 Exception이 올라거나 WAS가 sendError 가 존재 하면 WAS가 오류관련된 오류 페이지가 있는지 확인하고 다시 HTTP요청이 온것처럼 오류 페이지를 요청한다.

흐름.

```
컨트롤러(예외 발생) → 인터셉터→  서블릿 → 필터 → WAS
→ 오류페이지 요청(WAS) → 필터 → 서블릿 → 인터셉터 → 컨트롤러 → 오류 View
```

오류 정보를 추가해서 **WAS**가 보내준다.

```
log.info("ERROR_EXCEPTION={}",
request.getAttribute("javax.servlet.error.exception"));
log.info("ERROR_EXCEPTION_TYPE={}",
request.getAttribute("javax.servlet.error.exception_type"));
log.info("ERROR_MESSAGE={}",
request.getAttribute("javax.servlet.error.message"));
log.info("ERROR_REQUEST_URI={}",
request.getAttribute("javax.servlet.error.request_uri"));
log.info("ERROR_SERVLET_NAME={}",
request.getAttribute("javax.servlet.error.servlet_name"));
log.info("ERROR_STATUS_CODE={}",
request.getAttribute("javax.servlet.error.status_code"));

log.info("dispatchType={}", request.getDispatcherType());
```

상수로 정의된 정보를 확인 가능하다.

필터 - 서블릿 예외 처리

흐름 에서 확인 해보면 필터나 인터셉트가 다시 요청될 수 있다 (비 효율적)

```
컨트롤러(예외 발생) → 인터셉터→  서블릿 → 필터 → WAS
→ 오류페이지 요청(WAS) → 필터 → 서블릿 → 인터셉터 → 컨트롤러 → 오류 View
```

이를 방지하기 위해 필터는 **DispatcherType**이라는 옵션을 제공한다.

고객이 처음 요청하게 되면 → dispatcherType = REQUEST

에러가 발생하면 dispatcherType = ERROR

서블릿이나 다른 서블릿이나 JSP를 호출 → dispatcherType = FORWORD

서블릿에서 다른 서블릿을 포함시킬때 → dispatcherType = INCLUDE

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

   @Bean
   public FilterRegistrationBean logFilter(){

       FilterRegistrationBean<Filter> filterRegistrationBean = new
FilterRegistrationBean<>();
       filterRegistrationBean.setFilter(new LogFilter());
       filterRegistrationBean.setOrder(1);
       filterRegistrationBean.addUrlPatterns("/*");
       filterRegistrationBean.setDispatcherTypes(DispatcherType.REQUEST,
DispatcherType.ERROR);

       return filterRegistrationBean;
   }
}
```

필터에 DispatcherType을 설정해 줄수 있다.

필터가 호출 될때 REQUEST , 필터를 나갈때 RESPONSE log를 찍어주게되면

```
.exception.filter.LogFilter         : REQUEST [297e7d33-ac03-4659-b68f-72fe518b92e7][REQUEST][/error-500123]
.exception.filter.LogFilter         : RESPONSE [297e7d33-ac03-4659-b68f-72fe518b92e7][REQUEST][/error-500123]
.exception.filter.LogFilter         : REQUEST [ff086107-f073-45dc-83ae-8c84d2b113f5][ERROR][/error-page/404]
eption.servlet.ErrorPageController  : errorPage 404
eption.servlet.ErrorPageController  : ERROR_EXCEPTION=null
eption.servlet.ErrorPageController  : ERROR_EXCEPTION_TYPE=null
eption.servlet.ErrorPageController  : ERROR_MESSAGE=
eption.servlet.ErrorPageController  : ERROR_REQUEST_URI=/error-500123
eption.servlet.ErrorPageController  : ERROR_SERVLET_NAME=dispatcherServlet
eption.servlet.ErrorPageController  : ERROR_STATUS_CODE=404
eption.servlet.ErrorPageController  : dispatchType=ERROR
.exception.filter.LogFilter         : RESPONSE [ff086107-f073-45dc-83ae-8c84d2b113f5][ERROR][/error-page/404]
```

오류 페이지 호출시 로그

`setDispatcherTypes 은 Default 는 REQUEST다.`

서블릿 예외 처리 - 인터셉터

로그를 찍는 인터셉터를 주고 설정한다

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
   registry.addInterceptor(new LogInterceptor())
           .order(1)
           .addPathPatterns("/**")
           .excludePathPatterns("/css/**", "*.ico", "/error", "/error-page/**");
   //excludePathPatterns 를 가지고 오류 페이지경로를 넣어준다
}
```

스프링 부트 - 오류 페이지 등록

기존 - WebServerCustomizer 예외 종류에 따라 ErrorPage종류 ErrorPageController 생성등

필요한 작업이 많다.

스프링 부트는 기본 오류 페이지 **/error**를 기본 페이지로 등록해준다**.**

또한 **error**를 매핑하는 **BasicErrorController** 라는 컨트롤러를 자동 등록해준다**.**

뷰 선택 우선순위 - BasicErrorController 의 처리 순서

[템플릿] - [정적리소스] - 상세 오류 코드

1. 뷰 템플릿

   resources/templates/error/500.html

   resources/templates/error/5xx.html

2. 정적 리소스(static , public)

   resources/static/error/400.html

resources/static/error/404.html

resources/static/error/4xx.html

3. 적용 대상이 없을 때 뷰 이름(error)

   resources/templates/error.html

BasicErrorController

```
<li>오류 정보</li>
<ul>
   <li th:text="|timestamp: ${timestamp}|"></li>
   <li th:text="|path: ${path}|"></li>
   <li th:text="|status: ${status}|"></li>
   <li th:text="|message: ${message}|"></li>
   <li th:text="|error: ${error}|"></li>
   <li th:text="|exception: ${exception}|"></li>
   <li th:text="|errors: ${errors}|"></li>
   <li th:text="|trace: ${trace}|"></li>
</ul>
```

- 오류 정보
  - timestamp: Thu Oct 27 09:54:09 KST 2022
  - path: /error-ex
  - status: 500
  - message: null
  - error: Internal Server Error
  - exception: null
  - errors: null
  - trace: null

application.properties에서 Model에 담길 에러를 정할 수 있다.

default가 표시가 되지 않는 이유는 보안상 문제가 생길 수 있기 때문이다**.**

server.error.include-exception=true

이런식이지만 오류에 관한 정보를 error페이지까지 넘기지 않고 서버에서 로그로 처리해서 확인해햐 한다!

server.error.whitelabel.enabled=true → 기본

- 오류 처리 화면을 못 찾을 시, 스프링 whitelabel 오류 페이지 적용

server.error.path=/error → 기본

- 오류 페이지 경로

변경 확장은 → ErrorController , BasicErrorController 상속받아 기능을 추가하면 된다.