Spring에서 Handler Interceptor 개념 및 구현해보기

2020-01-10 goodGid Spring

> 이 글의 코드 및 정보들은 강의를 들으며 정리한 내용을 토대로 작성하였습니다.

Concept

- Interceptor 2가지 역할을 해준다.

1. Handler에 요청을 전달하기 전/후로 **추가적인 작업**이 가능하다.
2. View 렌더링이 된 후 클라이언트에게 Reponse를 전달하기 전에 **추가적인 작업**이 가능하다.

Interceptor vs Filter

> 영향 범위 : Filter > Interceptor

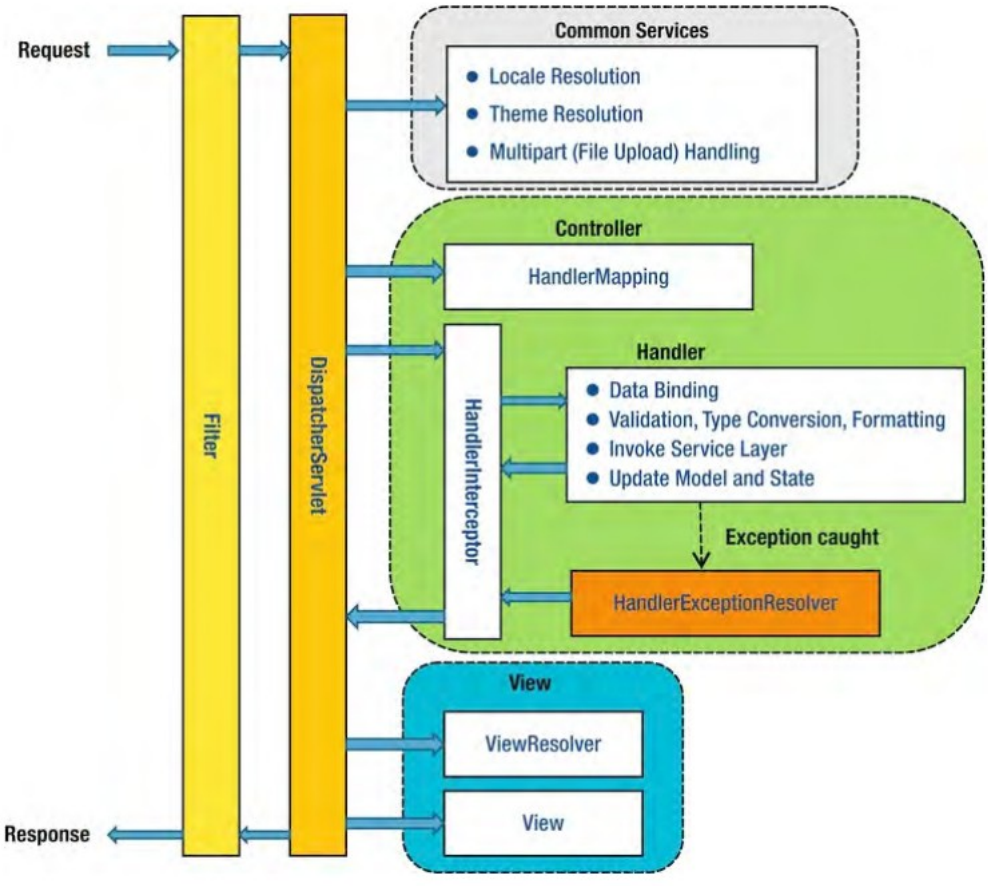

Figure 17-3. Spring MVC request life cycle

Index

정답은 아니지만

필자가 생각하기에

DispactcherServlet를 기준으로

Spring과 관련된 작업이 아니라면

(= Web에 대한 전반적인 작업)

Filter에서 구현하는게 적합하다고 생각한다.

Q&A

> Questioon

- 만약 XSS 요청에 대한

  Validate 작업을 진행한다면

  Interceptor와 Filter 중

  어느 곳에 Validate를 진행할 것인가?

> Answer

- 특정 Handler에 대한 전처리 작업이라기 보다는

  Web 전반적인 부분에 대한 Validation이기 때문에 Filter에서 진행한다.

Interceptor 동작 순서

- Interceptor의 Process에 대해 알아보자.

```
1. preHandler
2. 요청 처리
3. postHandler
4. View 렌더링
5. afterCompletion
```

> Step 1

- 요청을 처리할 Handler에 가기전에

  Interceptor의 preHandelr() 메소드가 호출된다.

> Step 2

- 요청을 처리할 수 있는 Handler가

  요청을 처리하고 retrun을 하게 되면

  Interceptor의 postHandler() 메소드가 호출된다.

> Step 3

- View 렌더링이 끝나면

  Interceptor의 afterCompletion() 메소드가 호출된다.

> Step 5

- Interceptor의 afterCompletion() 메소드에

  정의된 모든 작업이 끝나면 클라이언트에게 최종적으로 Response를 전달한다.

Multi Interceptor

- 2개 이상의 Interceptor가

  등록이 되어있는 상황에서

  각 Interceptor가 어떻게 동작하는지 알아보자.

```
1-1. preHandler 1
1-2. preHandler 2
2.   요청 처리
3-1. postHandler 2
3-2. postHandler 1
4.   View 렌더링
5-1. afterCompletion 2
5-2. afterCompletion 1
```

- 전체적인 Process는

  Interceptor가 1개일 경우와 동일하지만

  preHandler / postHandelr / afterCompletion 메소드의 호출이 뒤섞이게 된다.
- 만약 preHandler가 먼저 호출되면

  postHandelr / afterCompletion 메소드는 나중에 호출된다.

Example Code

Interceptor

- preHandler / postHandelr / afterCompletion

  각 메소드마다 Parameter가 다르다.

> GreetingInterceptor

```
public class GreetingInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response,
```

```
    @Override
    public void postHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler,
        ModelAndView modelAndView) throws Exception {
            /***
            * View를 렌더링하기 전에
            * postHandle 메소드가 호출된다.
            * 그렇기 때문에
            * modelAndView 정보를 알 수 있다.
            * 특정 View에 modelAndView 값을 수정해야할 필요가 있다면
            * postHandle 메소드에서 작업이 이뤄지면 된다.
            */
        System.out.println("[1] GreetingInterceptor postHandle");
    }

    @Override
    public void afterCompletion(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler,
        Exception ex) throws Exception {
            /***
            * 클라이언트에게
            * 최종적으로 Response를 전달하기 전에
            * 호출되는 afterCompletion 메소드에는
            * Exception 정보가 담겨온다.
            * 그렇기 때문에
            * afterCompletion 메소드에서는
            * response의 값을 Control 하거나
            * Exception 값에 따른 핸들링을 하면 된다.
            */
        System.out.println("[1] GreetingInterceptor afterCompletion");
    }
}
```

> GoodByeInterceptor

```
public class GoodByeInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler) throws Exception {
        System.out.println("[2] GoodByeInterceptor preHandle");
        return true;
    }

    @Override
    public void postHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler,
        ModelAndView modelAndView) throws Exception {
        System.out.println("[2] GoodByeInterceptor postHandle");
```

```
        HttpServletResponse response,
        Object handler,
        Exception ex) throws Exception {
        System.out.println("[2] GoodByeInterceptor afterCompletion");
    }
}
```

Controller

> SimpleController

```
@RestController
public class SimpleController {

    @GetMapping("/hello")
    public String hello(){
        return "hello";
    }
}
```

Test Code

> SimpleControllerTest

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
class SimpleControllerTest {

    @Autowired
    MockMvc mockMvc;

    @Test
    public void hello() throws Exception {
        this.mockMvc.perform(MockMvcRequestBuilders.get("/hello"))
                .andDo(print())
                .andExpect(content().string("hello"));
    }
}
```

Config

> WebConfig

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
```

```
        /***
         * ## Output
         * [1] GreetingInterceptor preHandle
         * [2] GoodByeInterceptor preHandle
         * [2] GoodByeInterceptor postHandle
         * [1] GreetingInterceptor postHandle
         * [2] GoodByeInterceptor afterCompletion
         * [1] GreetingInterceptor afterCompletion
         */

        /***
         * 2. Order 설정
         * order의 값이 낮을수록
         * 우선 순위가 높아진다.
         * = 먼저 호출된다.
         */
        registry.addInterceptor(new GreetingInterceptor()).order(0);
        registry.addInterceptor(new GoodByeInterceptor()).order(-1);

        /**
         * ## Output
         * [2] GoodByeInterceptor preHandle
         * [1] GreetingInterceptor preHandle
         * [1] GreetingInterceptor postHandle
         * [2] GoodByeInterceptor postHandle
         * [1] GreetingInterceptor afterCompletion
         * [2] GoodByeInterceptor afterCompletion
         */

        /***
         * 3. Pattern 명시
         */
        registry.addInterceptor(new GreetingInterceptor()).order(0);
        registry.addInterceptor(new GoodByeInterceptor())
                .addPathPatterns("/hi")
                .order(-1);
        /***
         * ## Output
         * [1] GreetingInterceptor preHandle
         * [1] GreetingInterceptor postHandle
         * [1] GreetingInterceptor afterCompletion
         */

    }
}
```

- Test Code를 실행시키면

  Interceptor가 실제로

  학습한대로 동작하는지 확인할 수 있다.

Summary

Multi Interceptor 상황 속에서

각 Interceptor가 어떤 순서로 동작하는지에 대해 알아봤다.

- 또한 preHandler / postHandelr / afterCompletion

  각 메소드마다 Parameter가 다른 이유에 대해서도 알아봤다.

- 이 글에서 다룬 Interceptor에 개념만 숙지하고 있으면

  Interceptor를 사용하는데 있어 충분하지 않을까 생각한다.

참고

- 스프링 웹 MVC
- SPRING MVC REQUEST LIFE CYCLE

Back : Spring Boot에서 WebMvcAutoConfiguration 클래스가 하는 역할 :: Converter / Formatter

Next : Spring의 Resource Handler 알아보기

Comments