Spring MVC2 16 [타입 컨버터, 포맷터]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.31

타입 컨버터

숫자 → 문자 문자 → 숫자등 타입을 변경시킬일은 굉장히 많음

스프링은 자동으로 받는 데이터 타입(문자) 여도 원하는 타입으로 컨버팅 해준다.

새로운 타입도 컨버터 인터페이스로 새로 만들 수 있다.

컨버터 인터페이스 생성

[문자← → 숫자] 컨버터 만들어보기

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.core.convert.converter.Converter;

@Slf4j
public class StringToIntegerConverter implements Converter<String, Integer> {

   @Override
   public Integer convert(String source) {
       log.info("convert source={}", source);
       return Integer.valueOf(source);
   }
}
```

💥Integer.valueOf , Integer.parseInt 차이 → 리턴형이 클래스 형인지 기본형인지 차이

```
@Slf4j
public class IntegerToStringConverter implements Converter<Integer, String> {
   @Override
   public String convert(Integer source) {
```

```
        log.info("convert source={}", source);
        return String.valueOf(source);
    }
}
```

문자를 → 정해진 객체로 변경시키는 컨버터

```
@Slf4j
public class StringToIpPortConverter implements Converter<String, IpPort> {
    @Override
    public IpPort convert(String source) {
        log.info("convert source={}", source);
        //"127.0.0.1:8080"
        String[] split = source.split(":");
        String ip = split[0];
        int port = Integer.parseInt(split[1]);
        return new IpPort(ip, port);
    }
}
```

생성한 컨버터 등록하기

- 컨버전 서비스로만 가져다 쓰면 된다. ConversionService

여러 컨버터

Converter → 기본 타입 컨버터

ConverterFactory → 전체 클래스 계층 구조가 필요할 때

GenericConverter → 정교한 구현, 대상 필드의 애노테이션 정보 사용가능

ConditionalGenericConverter → 특정 조건이 참인 경우 사용

스프링 ConversionService

컨버전 서비스 인터페이스는 컨버팅이 가능한지, 커버팅 실행 두개 기능이 있는 인터페이스이다.

```
@Test
void conversionService(){
   //등록
   DefaultConversionService conversionService = new DefaultConversionService();
   conversionService.addConverter(new StringToIntegerConverter());
   conversionService.addConverter(new IntegerToStringConverter());
   conversionService.addConverter(new StringToIpPortConverter());
   conversionService.addConverter(new IpPortToStringConverter());

   //사용
   Integer result = conversionService.convert("10", Integer.class);
   System.out.println("result = " + result);
}
```

```
[main] INFO
hello.typeconverter.controller.converter.StringToIntegerConverter - convert
source=10
result = 10
```

→ 어떤 컨버터가 사용되었는지 나온다.

각각 테스트

```
@Test
void conversionService(){
   //등록
   DefaultConversionService conversionService = new DefaultConversionService();
   conversionService.addConverter(new StringToIntegerConverter());
   conversionService.addConverter(new IntegerToStringConverter());
   conversionService.addConverter(new StringToIpPortConverter());
   conversionService.addConverter(new IpPortToStringConverter());

   //사용
   Integer result = conversionService.convert("10", Integer.class);
   assertThat(conversionService.convert("10", Integer.class)).isEqualTo(10);
   assertThat(conversionService.convert(10, String.class)).isEqualTo("10");
   assertThat(conversionService.convert("127.0.0.1:8080", IpPort.class))
```

```
            .isEqualTo(new IpPort("127.0.0.1", 8080));
    assertThat(conversionService.convert(new IpPort("127.0.0.1", 8080),
String.class))
            .isEqualTo("127.0.0.1:8080");
}
```

-각각 사용된 컨버터

```
18:09:35.306 [main] INFO
hello.typeconverter.controller.converter.StringToIntegerConverter - convert
source=10
18:09:35.353 [main] INFO
hello.typeconverter.controller.converter.IntegerToStringConverter - convert
source=10
18:09:35.353 [main] INFO
hello.typeconverter.controller.converter.StringToIpPortConverter - convert
source=127.0.0.1:8080
18:09:35.353 [main] INFO
hello.typeconverter.controller.converter.IpPortToStringConverter - convert
source=IpPort(ip=127.0.0.1, port=8080)
```

타입컨버터 들은 구체적인 어떤 컨버터가 사용되는지 몰라도 되고 컨버전 서비스 인터페이스만 의존해서 사용하면 된다.

DefaultConversionService는 ConversionService사용에 초점, CoverterRegistry 컨버터 등록에 초점에 나눠져있고 자신이 사용하지 않는 메서드에 의존하지 않게 되어 있다.

→ SOLID 의 ISP 인터페이스 분리원칙을 잘 따르게 된다.

스프링에 등록

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addFormatters(FormatterRegistry registry) {
        registry.addConverter(new StringToIpPortConverter());
        registry.addConverter(new StringToIntegerConverter());
        registry.addConverter(new IntegerToStringConverter());
        registry.addConverter(new IpPortToStringConverter());
    }
```

```
}
```

컨버터를 추가하면 추가한 컨버터가 우선 순위를 갖는다.

→ @RequestParam 은 ArgumentResolver 인 RequestParamArgumentResolver에서ConversionService를 사용해서 타입을 변환한다.

뷰 템플릿에 컨버터 적용하기

객체→ 문자로 변환해서 표출 시키기

```
@GetMapping("/converter-view")
public String converterView(Model model){
   model.addAttribute("number", 10000);
   model.addAttribute("ipPort",new IpPort("127.0.0.1",8080));
   return "converter-view";
}
```

${ipPort} 와 ${{ipPort}} 의 차이

```
<ul>
   <li>${number}: <span th:text="${number}" ></span></li>
   <li>${{number}}: <span th:text="${{number}}" ></span></li>
   <li>${ipPort}: <span th:text="${ipPort}" ></span></li>
   <li>${{ipPort}}: <span th:text="${{ipPort}}" ></span></li>
</ul>
```

- ${number}: 10000
- ${{number}}: 10000
- ${ipPort}: IpPort(ip=127.0.0.1, port=8080)
- ${{ipPort}}: 127.0.0.1:8080

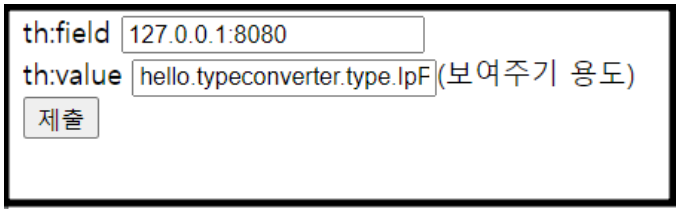
결과

→ ${} ⇒ toString

→ ${{}} ⇒ 컨버팅 후 toString [컨버터 서비스 실행]

th:field 는 자동 컨버터 적용, **th:value**는 컨버터 자동 적용**x**

```
th:field <input type="text" th:field="*{ipPort}"><br/>
th:value <input type="text" th:value="*{ipPort}">(보여주기 용도)<br/>
```



포맷터 **Formatter**

포맷팅되어있는 문자를 객체로 바꾸거나

특정한 객체를 정해진 포맷에 맞춰진 문자로 변환할 때 포맷터를 사용한다.

Fomatter = 문자에 특화된 컨버터 + 현지화

print → 객체를 문자로

parse → 문자를 객체로

```
@Slf4j
public class MyNumberFormatter implements Formatter<Number> {
   @Override
   public Number parse(String text, Locale locale) throws ParseException {
       log.info("text={}, locale={}", text, locale);
       //"1,000" -> 1000
       NumberFormat format = NumberFormat.getInstance(locale);
       return format.parse(text);
   }
   @Override
   public String print(Number object, Locale locale) {
       log.info("object={}, locale={}", object, locale);
       return NumberFormat.getInstance(locale).format(object);
```

```
    }
}
```

FormattingConversionService

사용하면 기본적인 포맷터가 적용되어있는 컨버터를 쓸수 있다.

```
@Test
void formattingConversionService(){
    DefaultFormattingConversionService conversionService = new
DefaultFormattingConversionService();
    //컨버터 서비스
    conversionService.addConverter(new StringToIpPortConverter());
    conversionService.addConverter(new IpPortToStringConverter());

    //포맷터 등록
    conversionService.addFormatter(new MyNumberFormatter());
    //컨버터 사용
    IpPort ipPort = conversionService.convert("127.0.0.1:8080", IpPort.class);
    assertThat(ipPort).isEqualTo(new IpPort("127.0.0.1", 8080));

    //포맷터 사용
    assertThat(conversionService.convert(1000, String.class)).isEqualTo("1,000");
    assertThat(conversionService.convert("1,000", Long.class)).isEqualTo(1000L);
}
```

포맷터 추가

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addFormatters(FormatterRegistry registry) {
//        registry.addConverter(new StringToIntegerConverter()); 우선순위 때문 주석
//        registry.addConverter(new IntegerToStringConverter()); 우선순위 때문 주석
        registry.addConverter(new StringToIpPortConverter());
        registry.addConverter(new IpPortToStringConverter());

        // 포맷터 추가
        registry.addFormatter(new MyNumberFormatter());
    }
}
```

이제 1000→ 1,000 포맷팅 된다.

파라미터를 전달 할 때도

1,000 를 전달 하고 Integer로 파라미터를 받게 되어도 Integer 10000 값으로 제대로 받아진다.

스프링이 제공하는 기본 포맷터

-애노테이션 기반으로 원하는 형식을 사용하는 포맷터 제공

@NumberFormat

@DateTimeFormat

두개를 제공한다. 숫자, 날짜

```
@Data
static class Form{
   @NumberFormat(pattern = "###,###")
   private Integer number;

   @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss")
   private LocalDateTime localDateTime;
}
```

주의

메시지 컨버터 (HttpMessageConverter) 에는 컨버전 서비스가 적용되지 않는다.

객체를 JSON으로 변환 할 때 메시지 컨버터 사용하면서 사용하게 되는데

@ResponseBody 등에서는 Jackson이 JSON을 포맷팅하는 방식을 사용하기 때문에

객체 모양이 아닌 다른모양의 포맷팅이 적용안될 수 있다 . (당황 할 수 있다)