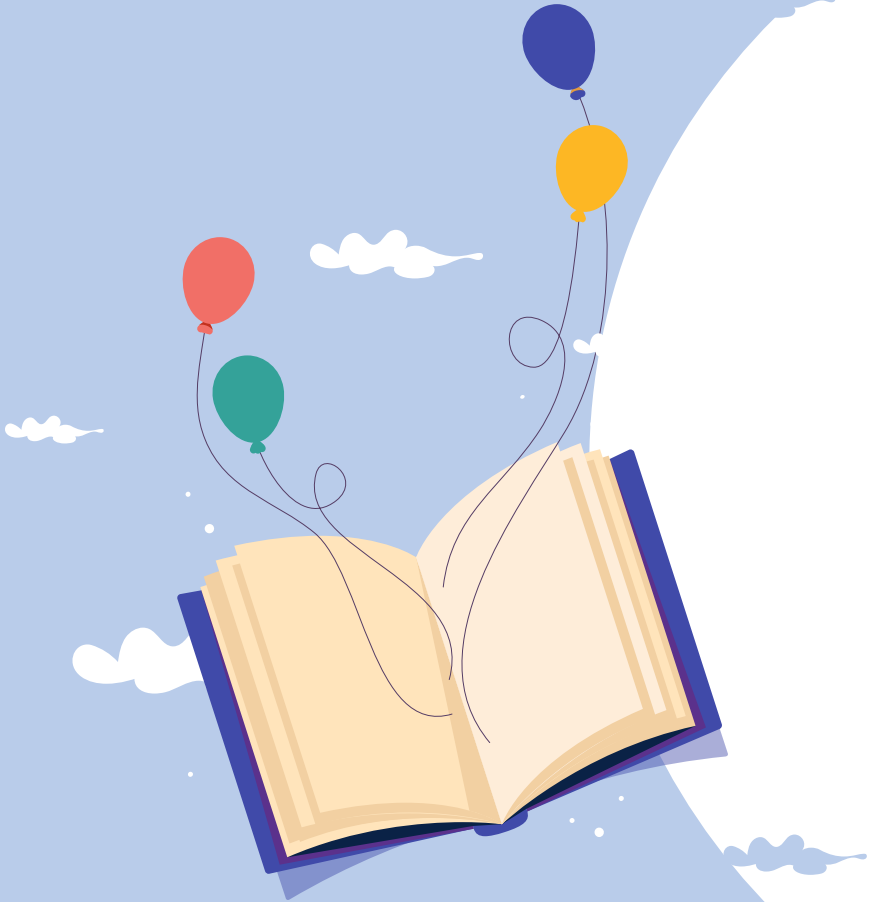
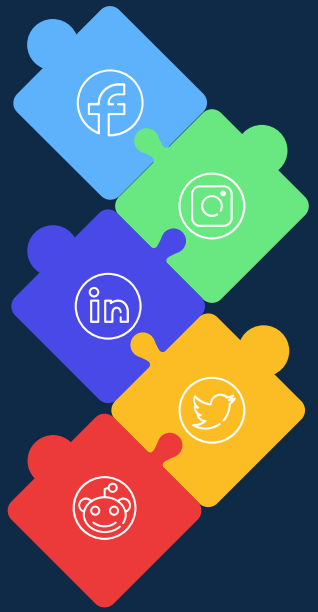


JAVA Spring Beginner

For coding starter

INDEX

- Exception process
- Log4j theory
- MessageSource
- Validation



EXCEPTION

- 예외 처리는 일반적으로 프로그램이 처리되는 동안 특정한 문제가 일어났을 때 처리를 중단하고 다른 처리를 하는 것을 의미(오류 처리)
- 즉, 프로그램을 실행할 때 발생할 수 있는 예외 상황에 대비한 코드를 작성하여 프로그램이 비정상으로 종료되는 것을 막는 것.
- 정상적인 실행 상태를 유지하는 예외 처리가 필요.
- 자바 구문에 어긋난 코드 때문에 발생하는 오류, 컴파일 할 때 발생하는 문법 오류, 프로그램을 실행할 때 상황에 따라 발생하는 실행 오류 등 웹 애플리케이션에서는 모든 유형의 오류가 발생할 수 있으므로 예외 처리는 안전 측면에서 필수.

EXCEPTION

Annotation Name	Explain
@ResponseStatus	예외 처리를 위한 가장 간단한 방법으로, 발생한 예외를 HTTP 상태 코드로 매핑시켜 응답하는 애너테이션
@ExceptionHandler	컨트롤러 안에 요청 처리 메서드에서 발생하는 오류나 예외를 직접 구체화하여 예외 처리를 위한 메서드에 선언하는 애너테이션
@ControllerAdvice	여러 컨트롤러에서 발생하는 오류의 공통점을 묶어 예외 처리를 위한 클래스에 선언하는 애너테이션

EXCEPTION

Response status code		Explain
400	BAD_REQUEST	일반적인 요청 실패에 사용
401	UNAUTHORIZED	클라이언트 인증에 문제가 있을 때 사용
403	FORBIDDEN	인증 상태에 상관없이 액세스를 금지할 때 사용
404	NOT_FOUND	요청 URI에 해당하는 리소스가 없을 때 사용
405	METHOD_NOT_ ALLOWED HTTP	메서드가 지원되지 않을 때 사용
406	NOT_ACCEPTABLE	요청된 리소스 미디어 타입을 제공하지 못할 때 사용
409	CONFLICT	리소스 상태에 위반되는 행위를 했을 때 사용
412	PRECONDITION_FAILED	조건부 연산을 지원할 때 사용
415	UNSUPPORTED_MEDIA_ TYPE	요청 페이로드에 있는 미디어 타입이 처리되지 못했을 때 사용
500	INTERNAL_SERVER_ ERROR	API가 잘못 작동할 때 사용

@ResponseStatus를 이용한 예외 처리

- @ResponseStatus는 웹 요청을 할 때 예외가 발생하면 지정된 HTTP 상태 코드를 웹 브라우저에 전달합니다. 이런 기능은 ResponseStatusExceptionResolver 클래스가 제공하며, 이 클래스는 디스패처 서블릿에 기본적으로 등록되어 있어 별도로 구성할 필요 없습니다.

```
// 예외 메서드에 사용할 때
@ResponseStatus(value=HttpStatus.상태 코드, reason="오류 설명")
@RequestMapping(...)
public String 메서드 이름() {
    ...
}
// 예외 클래스에 사용할 때
@ResponseStatus(value=HttpStatus.상태 코드, reason="오류 설명")
public class 클래스 이름 extends RuntimeException {
    ....
}
```

@ExceptionHandler를 이용한 컨트롤러 기반 예외 처리

- @ExceptionHandler는 웹 요청에 따라 컨트롤러의 요청 처리 메서드를 실행하는 동안 예외가 발생하면 이를 처리하기 위해 예외 처리 메서드에 사용

```
@ExceptionHandler(value={예외 클래스1.class[,예외 클래스2.class,...]})
public String 메서드 이름() {
    ...
}

@ExceptionHandler public String 메서드 이름(예외 클래스 exp) {
    ...
}
```

component	type	Explain
value	Class<? extends Throwable>[]	@ExceptionHandler가 선언된 메서드가 처리할 예외 클래스

@ControllerAdvice를 이용한 전역 예외 처리

- 스프링 MVC에서는 컨트롤러의 요청 처리 메서드에서 예외가 발생할 경우 같은 컨트롤러 안에서 예외 처리를 작성할 수 있다.
- 이런 예외 발생을 개별 컨트롤러에서 처리하지 않고 전체 애플리케이션에 한 번에 적용할 수 있는 @ControllerAdvice를 지원.
- @ControllerAdvice를 사용하면 하나의 컨트롤러가 아닌 여러 컨트롤러에서 발생하는 예외를 공통으로 처리 가능

```
@ControllerAdvice(basePackages={"기본 패키지 이름, ...})
public class 클래스 이름 {
    ...
}
```

@ControllerAdvice를 이용한 전역 예외 처리

component	type	Explain
annotations	Class<? extends Annotation>[]	애너테이션의 배열
assignableTypes	Class<?>[]	클래스의 배열
basePackageClasses	Class<?>[]	@ControllerAdvice가 적용된 클래스가 지원할 컨트롤러를 선택할 수 있는 패키지를 지정
basePackages	String[]	기본 패키지의 배열
value	String[]	basePackages 속성의 별칭

Log4j 개요

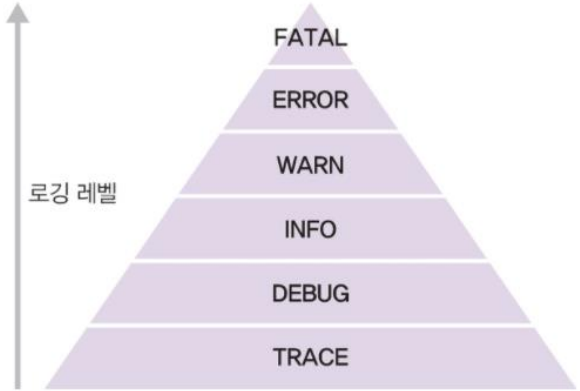
- 로깅 유틸리티 **Log4j**

- Log4j(Log for Java)는 아파치 소프트웨어 라이선스에 따라 배포되는 로깅 프레임워크로, 자바로 작성되어 있고 안정적이고 신속하며 유연
- 시스템 성능에 큰 영향을 미치지 않으면서 선택적인 로그를 남긴다거나 특정 파일에 로그를 기록할 수 있는 환경을 제공
- 웹 애플리케이션이 동작하는 중에 남긴 로그 기록은 애플리케이션이 운영되는 동안 정확한 상황과 상태 정보를 제공하고, 파일이나 DB에 남긴 로그 기록은 나중에 로그 결과를 분석하는 데 사용
- 로그 기록은 개발 중 문제가 발생했을 때 개발자가 자세한 상황을 파악할 수 있게 하므로 개발을 테스팅할 때 빠질 수 없는 요소
- 스프링 MVC에서 로깅 유틸리티 Log4j 기능을 사용하려면 의존성 라이브러리 정보를 등록해야 함

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
```

Log4j 로깅 레벨

Level	Explain
FATAL	조기 종료를 유발하는 심각한 오류가 발생한 상태를 나타냅니다.
ERROR	기타 런타임 오류 또는 예기치 않은 상태를 나타냅니다.
WARN	사용되지 않는 API 사용, API의 사용 빈도, 오류, 바람직하지 않거나 예기치 않은 런타임 상황의 경고성 메시지를 나타냅니다.
INFO	시작, 종료 같은 런타임 이벤트 메시지를 나타냅니다.
DEBUG	디버그 용도로 시스템 흐름에 대한 자세한 정보를 나타냅니다.
TRACE	가장 하위 로깅 레벨로, 모든 로그에 대한 상세한 정보를 나타냅니다



Appender 유형

Class	Explain
ConsoleAppender	콘솔에 로그 정보를 출력합니다.
FileAppender	파일에 로그 정보를 출력합니다.
RollingFileAppender	로그 크기가 지정한 용량 이상이 되면 다른 이름의 파일로 출력합니다.
DailyRollingFileAppender	하루 단위로 로그 정보를 파일에 출력합니다.
SMTPAppender	로그 메시지를 이메일로 보냅니다.
NTEventLogAppender	윈도의 이벤트 로그 시스템에 로그 정보를 기록합니다

Layout 구성

Class	Explain
DateLayout	로그 정보를 날짜 중심으로 간단하게 기록합니다.
HTMLLayout	로그 정보를 HTML 형식으로 기록합니다.
PatternLayout	로그 정보를 사용자 정의 패턴에 따라 기록합니다. • %c: 카테고리 출력 • %p: 로깅 레벨 출력 • %m: 로그 내용 • %d: 로깅 이벤트가 발생한 시간, yyyy-MM-dd, HH:mm:ss 등 시간 형식 사용 • %t: 로깅 이벤트를 발생한 스레드 이름 • %n: 개행 • %C: 클래스 이름 • %F: 로깅이 발생한 파일 이름 • %I: 로깅이 발생한 호출자 정보 • %L: 로깅이 발생한 코드 라인 수 • %M: 로깅이 발생한 메서드 이름 • %r: 애플리케이션 시작 이후부터 로깅이 발생한 시점의 시간 • %x: 로깅이 발생한 스레드와 관련된 NDC(Nested Diagnostic Context) • %X: 로깅이 발생한 스레드와 관련된 MDC(Mapped Diagnostic Context)
SimpleLayout	로그 정보를 '레벨-정보' 형식으로 간단하게 기록합니다.
XMLLayout	로그 정보를 XML 형식으로 기록합니다

Intercepter

- 인터셉터(intercepter)는 사용자가 URL을 요청하면 컨트롤러에 요청이 들어가기 전에, 즉 컨트롤러가 웹으로 응답하기 전에 가로채어 특정 작업을 처리하는 것을 의미
- 핸들러 인터셉터(handler Interceptor)라고도 함
- 컨트롤러에 들어오는 요청 HttpRequest와 컨트롤러가 응답하는 HttpResponse를 가로채는 역할
- 인터셉터는 디스패처 서블릿이 컨트롤러를 호출하기 전후에 요청과 응답을 가로채기 때문에 다음과 같은 특정 작업을 수행하는 데 사용

  - 시간별 동작 및 성능의 병목 지점을 검사(profiling)하는 기능
  - 응답 페이지를 출력하기 전에 서버에서 미리 데이터를 가져오는 기능(spooling)
  - 폼에서 제출(submit)이 중복으로 일어나는 것을 막는 기능
  - 요청이 처리되기 전에 파일을 업로드(multipart)하는 기능
  - 각 요청에 대한 상세한 내역을 기록(logging)하는 기능
  - 유효성을 검사(validation)하는 기능

Intercepter 등록

```
// 모든 웹 요청 URL에 적용
<interceptors>
    <bean id=" " class="인터셉터 클래스(패키지 포함)"/>
</interceptors>


// 특정 웹 요청 URL에 적용
<interceptors>
    <interceptor>
        <mapping path="특정 패턴의 요청 URL"/>
        <beans:bean id=" " class="인터셉터 클래스(패키지 포함)"/>
    <interceptor>
</interceptors>
```

Intercepter 활용 로그 기록

로그를 기록하기 위해 HandlerInterceptor 인터페이스와 HandlerInterceptorAdapter 클래스를 알아봅니다.

HandlerInterceptor를 이용한 로그 기록

- HandlerInterceptor 인터페이스는 preHandle(), postHandle(), afterCompletion() 세 메서드를 가지고 있다.
- preHandle() 메서드는 웹 요청 URL이 컨트롤러에 들어가기 전에 호출되며, false로 반환하면 이후 내용은 실행하지 않는다.
- postHandle() 메서드는 웹 요청 URL을 컨트롤러가 처리한 후 호출되며, 컨트롤러에서 예외가 발생하면 더 이상 실행되지 않는다.
- afterCompletion() 메서드는 컨트롤러가 웹 요청을 처리하여 뷰에 응답 전송이 종료된 후 호출

HandlerInterceptor 인터페이스의 메서드 형식은 다음과 같습니다.

```
boolean preHandle(HttpServletRequest request,
                  HttpServletResponse response,
                  Object handler) throws Exception;
```

- *역할: 컨트롤러를 호출하기 이전에 핸들러 실행을 차단합니다.*
- *매개변수:*
  - *request: 현재 HTTP 요청*
  - *response: 현재 HTTP 응답*
  - *handler: 실행 핸들러 선택*

Intercepter 활용 로그 기록

*void **postHandle**(HttpServletRequest request,*
HttpServletResponse response,
Object handler,
ModelAndView modelAndView) throws Exception;

- *역할: 컨트롤러를 호출하여 처리한 후에 핸들러 실행을 차단합니다.*
- *매개변수:*
  - *request: 현재 HTTP 요청*
  - *response: 현재 HTTP 응답*
  - *handler: 시작된 비동기 실행 핸들러*
  - *modelAndView: ModelAndView로 반환된 핸들러*

*void **afterCompletion**(HttpServletRequest request,*
HttpServletResponse response,
Object handler,
Exception ex) throws Exception;

- *역할: 뷰에 최종 결과를 반환한 후에 핸들러 실행을 차단합니다.*
- *매개변수:*
  - *request: 현재 HTTP 요청*
  - *response: 현재 HTTP 응답*
  - *handler: 시작된 비동기 실행 핸들러*

Locale Process (다국어 처리)

다국어 처리는 웹 브라우저의 로케일(locale)에 따라 다양한 언어를 지원하는 서비스입니다. 즉, 다양한 언어와 지역적 차이에 따라 웹 애플리케이션의 코드를 수정할 필요 없이 웹 브라우저의 로케일을 기준으로 각 언어에 해당하는 메시지로 변경하는 서비스입니다.

TIPS 다국어 처리는 웹 브라우저의 로케일(locale)에 따라 다양한 언어를 지원하는 서비스입니다. 즉, 다양한 언어와 지역적 차이에 따라 웹 애플리케이션의 코드를 수정할 필요 없이 웹 브라우저의 로케일을 기준으로 각 언어에 해당하는 메시지로 변경하는 서비스

MessageSource

메시지 리소스 파일 작성

- 프로젝트에 지원 가능한 각 언어에 대해 메시지 리소스 파일(*.properties)을 생성하여 출력할 메시지를 작성
- 메시지 리소스 파일에는 key=value 쌍으로 구성되며, 여기에서 key는 뷰 페이지에서 메시지를 참조하는 데 사용
- 기본 언어의 메시지 리소스 파일을 '파일 이름.properties' 형태로 작성

다음은 기본 언어가 한국어고, 메시지 리소스 파일은 messages.properties로 작성한 예입니다. 여기에서 key는 'Person.form.Enter.message'고 value는 '당신의 정보를 입력하세요.'로 뷰 페이지에서 key를 사용하여 value에 설정된 메시지를 출력합니다.

messages.properties

Person.form.Enter.message = 당신의 정보를 입력하세요

Locale Process (다국어 처리)

File type	Explain
파일 이름.properties	시스템의 언어 및 지역에 맞는 리소스 파일이 없을 때 사용합니다.
파일 이름_ko_KR.properties	시스템 언어 코드가 한국어일 때 사용합니다.
파일 이름_en_EN.properties	시스템 언어 코드가 영어일 때 사용합니다.
파일 이름_en_UK.properties	시스템 언어 코드가 영어고 영국(국가 코드)일 때 사용합니다.
파일 이름_ja.properties	시스템 언어 코드가 일본어일 때 사용합니다.

※ 메시지 리소스 파일은 src/main/resources 폴더에 위치해야 함

LocaleResolver와 LocaleChangeInterceptor

다양한 언어로 작성된 메시지 리소스 파일을 읽어 웹 브라우저의 로케일에 따라 각 언어에 해당하는 메시지로 설정할 수 있는 LocaleResolver를 알아봅니다. 그리고 다양한 언어를 자유롭게 선택하여 변경할 수 있는 LocaleChangeInterceptor도 알아봅니다.

- **LocaleResolver 환경 설정**

스프링 MVC는 LocaleResolver로 웹 브라우저의 로케일을 추출해서 알맞은 언어를 선택하여 메시지를 출력합니다. 즉, 디스패처 서블릿은 웹 요청이 들어오면 LocaleResolver를 검색합니다. 로케일 객체가 검색된다면 이를 이용하여 로케일을 설정합니다.

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.LocaleResolver 구현체">
    <property name="defaultLocal" value="로케일 언어"/>
    ...
</bean>
```

LocaleResolver 구현체

Type	Explain
AcceptHeaderLocaleResolver	웹 브라우저에 설정된 기본 로케일 정보를 사용합니다. HTTP 요청의 accept-language 헤더에 지정된 기본 로케일을 사용합니다.
CookieLocaleResolver	쿠키를 이용한 로케일 정보를 사용합니다. 사용자 지정 로케일, 표준 시간대 정보를 브라우저 쿠키로 유지합니다.
SessionLocaleResolver	세션을 이용한 로케일 정보를 사용합니다. 사용자 세션에서 locale 속성을 사용하여 지정된 기본 로케일 또는 요청의 accept-header 로케일로 대체합니다.
FixedLocaleResolver	특정 로케일을 지정합니다. 항상 고정된 기본 로케일을 반환하고 선택적으로 시간대를 반환합니다.

LocaleResolver와 LocaleChangeInterceptor

LocaleChangeInterceptor를 이용한 로케일 변경

- LocaleChangeInterceptor 클래스를 사용하면 로케일을 변경하는 별도의 컨트롤러 클래스를 구현할 필요 없이 메시지를 해당 언어로 변경 가능
- 즉, 웹 요청의 매개변수를 사용하여 손쉽게 로케일을 바꿀 수 있다.
- LocaleChangeInterceptor 클래스는 HandlerInterceptor로 <interceptors> 요소에 등록만 하면 디스패처 서블릿이 컨트롤러에 접근할 때 응답을 가로채서 LocaleChangeInterceptor를 적용

Validation

유효성 검사의 유형

폼 데이터 값에 대한 유효성 검사는 JSR-380 Validation(Bean Validation)이나 스프링이 제공하는 Validator 인터페이스를 사용하여 구현

• JSR-380 Validation(Java Bean Validation 2.0) 방식: 웹 애플리케이션을 구성하는 특정 도메인 클래스의 멤버 변수, 즉 필드에 대한 유효성 검사 제약 사항(constraints) 애너테이션을 선언하여 해당 값이 올바른지 검증하는 방식

• Validator 인터페이스의 구현체 방식: 웹 애플리케이션을 구성하는 특정 도메인 클래스의 멤버 변수에는 제약 사항 애너테이션을 선언하지 않습니다. 그 대신 스프링에서 제공하는 Validator 인터페이스로 구현하고 이를 Validator 인스턴스로 사용하여 해당 속성 값의 유효성 검사를 수행합니다. 스프링 Validator 인터페이스는 애플리케이션의 모든 계층에서 유효성 검증을 위해 사용할 수 있음

@Valid

@Valid를 이용한 유효성 검사

스프링 MVC에서는 사용자가 폼 페이지에서 입력한 데이터의 유효성을 검사하기 위해 코드를 작성할 필요 없이 간단한 방법으로 @Valid 애너테이션을 제공

pom.xml 파일에 의존 라이브러리 등록하기

@Valid를 이용하여 폼 데이터 값에 대한 유효성 검사를 하려면 pom.xml 파일에 validation-api.jar과 hibernate-validator.jar 의존 라이브러리를 등록해야 함

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>7.0.4.Final</version>
</dependency>
```

@Valid

@Valid를 이용하면 컨트롤러 내 요청 처리 메서드의 매개변수에 전달되는 폼 데이터 값에 대한 유효성 검사를 실행

```
@PostMapping("/...")
 public String 메서드 이름(@Valid 매개변수, ..., BindingResult result) {
     if (result.hasErrors()) {
         // 오류 메시지 저장
     }
     return "뷰 이름";
}
```

뷰 페이지에 오류 메시지 출력하기

유효성을 검사하여 발생한 오류 메시지를 JSP 뷰 페이지에 쉽게 출력하려면 폼 태그 라이브러리 중 <form:errors> 태그를 사용하면 됩니다.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
...
<form:errors path="커맨드 객체의 멤버 변수 이름 ">
```

JSR-380 Validation

JSR-380 애너테이션을 이용한 유효성 검사는 웹 애플리케이션을 구성하는 특정 도메인 클래스의 프로퍼티(멤버 변수), 즉 필드에 대해 유효성 검사의 제약 사항 애너테이션을 선언하여 해당 멤버 변수 값이 올바른지 검사하는 것으로, Bean Validation 2.0이라고도 함

JSR-380 애너테이션 선언

- JSR-380 애너테이션은 Hibernate Validator가 제공하는 애너테이션을 그대로 따름
- JSR-380도 유효성 검사가 필요한 도메인 클래스의 프로퍼티, 즉 필드에 제약 사항을 설정 가능.
- 형식

```
public class 클래스 이름 {
    @JSR-380 제약 사항 애너테이션(속성[, message="오류 메시지 또는 [오류 코드]"])
    private String 멤버 변수;
    ...
    // Setter()와 Getter() 메서드 구현 생략
}
```

JSR-380 Validation

도메인 클래스의 프로퍼티에 적용할 수 있는 JSR-380 애너테이션으로, JSR-380 Bean Validation에 내장된 제약 사항 애너테이션

Type	Explain	-
@AssertFalse	프로퍼티 값이 거짓(false)인지 검사합니다.	
@AssertTrue	프로퍼티 값이 참(true)인지 검사합니다.	
@DecimalMax	프로퍼티 값이 가질 수 있는 최대 실수 값을 검사합니다.	• value: 값 • inclusive: true/false
@DecimalMin	프로퍼티 값이 가질 수 있는 최소 실수 값을 검사합니다.	• value: 값 • inclusive: true/false
@Digits	프로퍼티가 가질 수 있는 지정된 범위 (정수 부분의 자릿수와 소수 부분의 자릿수)를 검사합니다.	• integer: 정수의 자릿수 • fraction: 소수의 자릿수
@Future	프로퍼티 값이 미래 날짜(현재일 이후)인지 검사합니다.	
@Max	프로퍼티 값이 가질 수 있는 최대 길이를 검사합니다.	value: 값
@Min	프로퍼티 값이 가질 수 있는 최소 길이를 검사합니다.	value: 값
@NotNull	프로퍼티 값이 Null이 아닌지 검사합니다.	
@Null	프로퍼티 값이 Null인지 검사합니다.	
@Past	프로퍼티 값이 과거 날짜(현재일 이전)인지 검사합니다.	
@Pattern	프로퍼티 값이 정의된 정규 표현식에 일치하는지 검사합니다. regexp: 정규 표현식	
@Size	프로퍼티 값이 가질 수 있는 최대, 최소 길이를 검사합니다.	• min: 최소 길이 • max: 최대 길이
@Valid	객체에 대해 유효성 검사를 합니다.	

JSR-380 Validation

JSR-380의 기본 메시지 사용하기

Annotation	Basic message
@AssertFalse	반드시 거짓(false)이어야 합니다.
@AssertTrue	반드시 참(true)이어야 합니다.
@DecimalMax	반드시 {value}보다 같거나 작아야 합니다.
@DecimalMin	반드시 {value}보다 같거나 커야 합니다.
@Digits	숫자 값이 허용 범위를 벗어납니다(허용 범위: <{integer} 자리>.<{fraction} 자리>).
@Future	반드시 미래 날짜이어야 합니다.
@Max	반드시 {value}보다 같거나 작아야 합니다.
@Min	반드시 {value}보다 같거나 커야 합니다.
@NotNull	반드시 값이 있어야 합니다.
@Null	반드시 값이 없어야 합니다.
@Past	반드시 과거 날짜이어야 합니다.
@Pattern	정규 표현식 "{regexp}" 패턴과 일치해야 합니다.
@Size	반드시 최솟값 {min}과(와) 최댓값 {max} 사이의 크기이어야 합니다.

Custom constraints annotation validation

JSR-380 제약 사항의 애너테이션으로 유효성 검사를 할 때는 중복 여부를 체크할 수 없다는 점에서 한계가 있습니다. 이런 문제를 해결하도록 JSR-380 Bean Validation API는 사용자 정의 제약 사항(custom constraints)을 선언할 수 있는 인터페이스를 제공합니다.

- 사용자 정의 애너테이션을 이용한 유효성 검사는 사용자 정의 애너테이션을 만들어 사용하는 것을 제외하고는 앞서 배운 JSR-380 애너테이션을 이용한 처리 과정과 같음
- 사용자 정의 애너테이션인 @MemberId를 만드는 방법

Product.java

```
public class Member {
    @MemberId // 사용자 정의 애너테이션 선언
    private String memberId;

    // Setter()와 Getter() 메서드
}
```

Custom constraints annotation validation

사용자 정의 애너테이션은 JSR-380처럼 도메인 클래스의 멤버 변수에 선언할 수 있는 제약 사항입니다. 또한 내장된 제약 사항이 아니므로 사용자가 직접 생성하여 사용할 수 있습니다. 사용자 정의 애너테이션을 생성하는 형식은 다음과 같습니다.

```
@Constraint(validatedBy=유효성 검사 클래스.class)
@Target(속성)
@Retention(속성)
@Documented public @interface 사용자 정의 애너테이션 이름 {
    String message() default "출력할 오류 메시지";
    Class<?>[] groups() default {};
    Class<?>[] payload() default {};
}
```

Custom constraints annotation validation

- 사용자 정의 애너테이션의 필수 속성

Contribute	Explain
message	유효성 검사에서 오류가 발생하면 반환되는 기본 메시지입니다.
groups	특정 유효성 검사를 그룹으로 설정합니다.
payload	사용자가 추가한 정보를 전달하는 값입니다.

- @Retention의 속성

Contribute	Explain
Source	소스 코드까지만 유지합니다. 즉, 컴파일하면 해당 애너테이션 정보는 사라집니다.
Class	컴파일한 .class 파일에 유지합니다. 즉, 런타임을 할 때 클래스를 메모리로 읽어 오면 해당 정보는 사라집니다.
Runtime	런타임을 할 때도 .class 파일에 유지합니다. 사용자 정의 애너테이션을 만들 때 주로 사용합니다.

Custom constraints annotation validation

- @Target의 속성

Contribute	Explain
TYPE	class, interface, enum
FIELD	클래스의 멤버 변수
METHOD	메서드
PARAMETER	메서드 인자
CONSTRUCTOR	생성자
LOCAL_VARIABLE	로컬 변수
ANNOTATION_TYPE	애너테이션 타입에만 적용
PACKAGE	패키지
TYPE_PARAMETER	제네릭 타입 변수(예: MyClass<T>)
TYPE_USE	어떤 타입에도 적용(예: extends, implements, 객체 생성할 때 등)

Custom constraints annotation validation

- **ConstraintValidator 인터페이스의 구현체 생성**
- 사용자 정의 애너테이션의 유효성 검사 클래스는 javax.validation.ConstraintValidator 인터페이스의 구현체를 생성
- 해당 구현체를 생성하려면 initialize()와 isValid() 메서드를 구현해야 함.

- ConstraintValidator 인터페이스의 메서드

Contribute	Explain
void initialize(A constraintAnnotation)	사용자 정의 애너테이션과 관련 정보를 읽어 초기화합니다. 이때 A는 사용자 정의 제약 사항을 설정합니다.
boolean isValid(T value, ConstraintValidatorContext context)	유효성 검사 로직을 수행합니다. value는 유효성 검사를 위한 도메인 클래스의 변수 값이고, context는 제약 사항을 평가하는 컨텍스트입니다.

Validator 인터페이스로 유효성 검사

스프링에서 제공하는 Validator 인터페이스를 사용하여 도메인 클래스의 속성 값이 올바른지 유효성을 검사하는 방법을 구분하여 알아봅니다.

1. 도메인 클래스의 속성에 애너테이션을 선언하지 않고 스프링에서 제공하는 Validator 인터페이스의 구현체를 생성하고, 이를 이용하여 속성 값의 유효성을 검사하는 방법
2. 스프링의 Validator 인터페이스와 JSR-380 Bean Validation과 연동하여 유효성을 검사하는 방법

- Validator 인터페이스의 메서드

Method	Explain
boolean supports(Class<?> clazz)	주어진 객체(class)에 대해 유효성 검사를 수행할 수 있는지 검사할 목적으로 사용됩니다.
void validate(Object target, Errors errors)	주어진 객체(target)에 대해 유효성 검사를 수행하고 오류가 발생하면 주어진 Errors 타입의 errors 객체에 오류 관련 정보를 저장합니다.

- Errors 객체의 주요 메서드

Method	Explain
void rejectValue(String field, String errorCode, String defaultMessage)	설정된 field가 유효성 검사를 할 때 오류를 발생시키면 설정된 errorCode와 함께 거부합니다.
void reject(String errorCode, String defaultMessage)	유효성 검사를 할 때 오류가 발생하면 설정된 errorCode를 사용하여 도메인 객체에 대한 전역 오류로 사용합니다.