Hibernate Validator

Java Bean Validation

Design by: DieuNT1

Lesson Objectives

1. Understand what is Hibernate validator.
2. Understand the **basic steps** to create and use hibernate validator.
3. Able to use Hibernate basic and specific validation annotations.

Agenda

- Validation annotations
- Validating Ranges
- Validating Strings
- URL and HTML Validation
- Hibernate validation @Pattern

Section 01

VALIDATION ANNOTATIONS

Introduction

- Validating user input is a super common requirement in most applications.
- And the Java Bean Validation framework has become the de facto standard for handling this kind of logic.
- JSR 380 (Java Specification Request) is a specification of the Java API for bean validation, part of Jakarta EE and JavaSE. This ensures that the properties of a bean meet specific criteria, using annotations such as *@NotNull*, *@Min*, and *@Max*.

Dependencies

- The *validation-api* dependency:

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

- Validation API Reference Implementation

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.5.Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator-annotation-processor</artifactId>
    <version>6.1.5.Final</version>
</dependency>
```

Dependencies

Expression Language Dependencies

```
<dependency>
    <groupId>javax.el</groupId>
    <artifactId>javax.el-api</artifactId>
    <version>3.0.1-b06</version>
</dependency>

<dependency>
<groupId>org.glassfish.web</groupId>
    <artifactId>javax.el</artifactId>
    <version>2.2.4</version>
</dependency>
```

Hibernate Validator Annotations

ANNOTATION	DESCRIPTION
@Digits(integer=, fraction=)	Checks whether the annotated value is a number having up to integer digits and fraction fractional digits.
@Email	Checks whether the specified character sequence is a valid email address.
@Max(value=)	Checks whether the annotated value is less than or equal to the specified maximum.
@Min(value=)	Checks whether the annotated value is higher than or equal to the specified minimum
@NotBlank	Checks that the annotated character sequence is not null and the trimmed length is greater than 0.
@NotEmpty	Checks whether the annotated element is not null nor empty.
@Null	Checks that the annotated value is null

Hibernate Validator Annotations

ANNOTATION	DESCRIPTION
@NotNull	Checks that the annotated value is not null
@Pattern(regex=, flags=)	Checks if the annotated string matches the regular expression regex considering the given flag match
@Size(min=, max=)	Checks if the annotated element's size is between min and max (inclusive)
@Negative	Checks if the element is strictly negative. Zero values are considered invalid.
@NegativeOrZero	Checks if the element is negative or zero.
@Future	Checks whether the annotated date is in the future.
@FutureOrPresent	Checks whether the annotated date is in the present or in the future.
@PastOrPresent	Checks whether the annotated date is in the past or in the present.

Hibernate validator specific annotations

ANNOTATION	DESCRIPTION
@CreditCardNumber(ignoreNonDigitCharacters=)	Checks that the annotated character sequence passes the Luhn checksum test. Note, this validation aims to check for user mistakes, not credit card validity!
@Currency(value=)	Checks that the currency unit of the annotated javax.money.MonetaryAmount is part of the specified currency units.
@EAN	Checks that the annotated character sequence is a valid EAN barcode. The default is EAN-13.
@ISBN	Checks that the annotated character sequence is a valid ISBN.
@Length(min=, max=)	Validates that the annotated character sequence is between min and max included.
@Range(min=, max=)	Checks whether the annotated value lies between (inclusive) the specified minimum and maximum.
@UniqueElements	Checks that the annotated collection only contains unique elements.
@URL	Checks if the annotated character sequence is a valid URL according to RFC2396.

Example

- Create Model annotated with validation annotations

```
public class User {

    @NotNull(message = "Please enter id")
    private Long id;

    @Size(max = 20, min = 3, message = "{user.name.invalid}")
    @NotEmpty(message = "Please enter name")
    private String name;

    @Email(message = "{user.email.invalid}")
    @NotEmpty(message = "Please enter email")
    private String email;

    public User(Long id, String name, String email) {
        super();
        this.id = id;
        this.name = name;
        this.email = email;
    }

    //Setters and Getters
}
```

Example

❖ Message resource

- ✓ By default, all messages are resolved from `ValidationMessages.properties` file in classpath. If file does not exist, the message resolution has not happen.

ValidationMessages.properties

```
user.name.invalid=Invalid Username
user.email.invalid=Invalid Email
```

Example

Execute validation

```
public class TestHibernateValidator {
    public static void main(String[] args) {
        // Create ValidatorFactory which returns validator
        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

        // It validates bean instances
        Validator validator = factory.getValidator();

        UserDetail userDetail = new UserDetail("Tran", "Phuong", "Phuong",
                LocalDate.of(2000, 1, 1));
        User user = new User("", "admin", userDetail);

        // Validate bean
        Set<ConstraintViolation<User>> constraintViolations = validator.validate(user);

        // Show errors
        if (constraintViolations.size() > 0) {
            for (ConstraintViolation<User> violation : constraintViolations) {
                System.out.println(violation.getMessage());
            }
        } else {
            System.out.println("Valid Object");
        }
    }
}
```

Example

Result:

```
Please enter id

Invalid Email

Invalid Username
```

Section 02

VALIDATING RANGES

Validating Ranges

- **Package:** *org.hibernate.validator.constraints.Range*
  - Numeric and Monetary Ranges
  - Duration of Time

Validating Ranges

Numeric and Monetary Ranges

- The bean validation specification defines several constraints which we can enforce on numeric fields.
- Besides those, Hibernate Validator provides a handy annotation, *@Range*, that **acts as a combination of *@Min* and *@Max,*** matching a range inclusively:

```
@Range(min = 0, max = 100)
private BigDecimal percent;
```

※ Like *@Min* and *@Max*, *@Range* is applicable on fields of primitive number types and their wrappers; *BigInteger* and *BigDecimal*, *String* representations of the above, and, finally, *MonetaryValue* fields.

Validating Ranges

❖ Duration of Time

- In addition to standard JSR 380 annotations for values that represent points in time, Hibernate Validator includes constraints for **Durations** as well.
- So, we can enforce minimum and maximum durations on a property:

```
@DurationMin(days = 1, hours = 2)
@DurationMax(days = 2, hours = 1)
private Duration duration;
```

※ By default, **minimum and maximum values are inclusive.** That is, a value which is exactly the same as the minimum or the maximum will pass validation.

✓ **We can define the *inclusive* property to be false:**

```
@DurationMax(minutes = 30, inclusive = false)
```

Section 03

VALIDATING STRINGS
URL AND HTML VALIDATION

Validating Strings

String Length

- We can use two slightly different constraints to enforce that a string is of a certain length:
- Generally, we'll want to ensure a string's length in characters – the one we measure with the *length* method – is between a minimum and a maximum.

```
@Length(min = 1, max = 3)
private String someString;
```

- Due to the intricacies of Unicode, sometimes the length in characters and the length in code points differ.

```
@CodePointLength(min = 1, max = 3)
private String someString;
```

Validating Strings

❖ Checks on Strings of Digits

- Hibernate Validator includes several other constraints for strings of digits
- **@LuhnCheck:** Perform the check on a substring (*startIndex* and *endIndex*) and tell the constraint which digit is the checksum digit (※ with -1 meaning the last one in the checked substring)

```
@LuhnCheck(startIndex = 0, endIndex = Integer.MAX_VALUE, checkDigitIndex = -1)
private String someString;
```

- **@ISBN** - Checks that the annotated character sequence is a valid ISBN. The length of the number and the check digit are both verified.

```
@ISBN
private String someString; // ex: 978-161-729-045-9
```

Validating Strings

URL and HTML Validation

- The *@Url* constraint verifies that a string is a valid representation of a URL. Additionally, we can check that specific component of the URL has a certain value:

```
@URL(protocol = "https")
private String url;
```

- We can also verify that a property contains “safe” HTML code (for example, without script tags)

```
@SafeHtml
private String html;
```

Section 04

HIBERNATE VALIDATION @PATTERN

Hibernate validation @Pattern

- Check if the property match the **regular expression** given a match flag (see java.util.regex.Pattern)
- Annotation:

  @Pattern(regex="regexp", flag=)

  Or

  @Patterns({@Pattern(...)})
- Apply on:

  property (string)
- Example:

```
// a not null numeric string of 5 characters maximum
@Length(max=5)
@Pattern(regex="[0-9]+")
@NotNull
private someString;
```

Summary

- Validation annotations
- Validating Ranges
- Validating Strings
- URL and HTML Validation
- Hibernate validation @Pattern

Thank you