Spring Framework 6

Beginner to Guru

Data Validation Overview

GIGO

GARBAGE IN = GARBAGE OUT

What is Validation?

- Validation is a process of making assertions against data to ensure data integrity

- Is a value required? How long is a phone number?

- Is it a good date? What is the maximum length of a string?

- Some refer to data validation as defensive programming

- Or a process of *trust but verify*

- Validation is an important step, but easily overlooked

When to Validate?

- Validate data early and often!

- Validation should occur with every exchange

- User input data should be validated in the UI with rich user feedback

- RESTful API data should be validated early in the controller, before the service layer

- Data should be validated before persistence to the database

- Database constraints will also enforce data validation

- Best to validate early - Handling persistence errors is ugly

Java Bean Validation

- Java Bean Validation is a Java API standard

- Provides for a standard way of performing validation and handling errors

- Much more graceful than custom code blocks of if… then… throw Exception

- Bean Validation is an API, like JPA or JDBC you need an implementation

- Fun Fact - Gunnar Morling, founder of MapStruct is the spec lead for the Bean Validation
  API and contributor of the Hibernate Implementation of the Bean Validation API

JSR 303 - Java Bean Validation

- JSR 303 Introduced Java Bean Validation (Version 1.0)

  - Set of annotations used to validate Java Bean properties

- Approved on November 16th, 2009.

- Part of JEE v6 and above

- JSR 303 Supported by Spring since version 3

- Primary focus was to define annotations for data validation

  - Largely field level properties

JSR 349 Bean Validation 1.1

- JSR 349 - Java Bean Validation 1.1 released on April 10th, 2013

  - JEE v7, Spring Framework 4

- Builds upon 1.0 specification

- Expanded to method level validation

  - To validate input parameters

- Includes dependency injection for bean validation components

JSR 380 - Bean Validation 2.0

- Approved August 2017

- Added to Spring Framework 5.0 RC2

- Available in Spring Boot 2.0.0 +

- Uses Hibernate Validator 6.0 + (Implementation of Bean Validation 2.0)

- Primary goal of Bean Validation 2.0 is Java 8 language features

- Added ~11 new built in validation annotations

Jakarta Bean Validation 3.0

- Released July of 2020

- Name changed from Bean Validation to **Jakarta** Bean Validation

- Only change from 2.0 to 3.0 is the API package changes

  - 2.0 - javax.validation

  - 3.0 - jakarta.validaton

- Used in Spring Framework 6.x+

- Hibernate Validator 7.x+ is the implementation

Built In Constraint Definitions

- **@Null** - Checks value is null

- **@NotNull** - Checks value is not null

- **@AssertTrue** - Value is true

- **@AssertFalse** - Value is false

- **@Min** - Number is equal or higher

- **@Max** - Number is equal or less

Built In Constraint Definitions

- **@DecimalMin** - Value is larger

- **@DecimalMax** - Value is less than

- **@Negative** - Value is less than zero. Zero invalid.

- **@NegativeOrZero** - Value is zero or less than zero

- **@Positive** - Value is greater than zero. Zero invalid.

- **@PositiveOrZero** - Value is zero or greater than zero.

- **@Size** - checks if string or collection is between a min and max

Built In Constraint Definitions

- **@Digits** - check for integer digits and fraction digits

- **@Past** - Checks if date is in past

- **@PastOrPresent** - Checks if date is in past or present

- **@Future** - Checks if date is in future

- **@FutureOrPresent** - Checks if date is present or in future

- **@Pattern** - checks against RegEx pattern

Built In Constraint Definitions

- **@NotEmpty** - Checks if value is not null nor empty (whitespace characters or empty collection)

- **@NonBlank** - Checks string is not null or not whitespace characters

- **@Email** - Checks if string value is an email address

Hiberate Validator Constraints

- **@ScriptAssert** - Class level annotation, checks class against script

- **@CreditCardNumber** - Verifies value is a credit card number

- **@Currency** - Valid currency amount

- **@DurationMax** - Duration less than given value

- **@DurationMin** - Duration greater than given value

- **@EAN** - Valid EAN barcode

- **@ISBN** - Valid ISBN value

Hiberate Validator Constraints

- **@Length** - String length between given min and max

- **@CodePointLength** - Validates that code point length of the annotated character sequence is between min and max included.

- **@LuhnCheck** - Luhn check sum

- **@Mod10Check** - Mod 10 check sum

- **@Mod11Check** - Mod 11 check sum

Hiberate Validator Constraints

- **@Range** - checks if number is between given min and max (inclusive)

- **@SafeHtml** - Checks for safe HTML

- **@UniqueElements** - Checks if collection has unique elements

- **@Url** - checks for valid URL

Hiberate Validator Constraints

- **@CNPJ** - Brazilian Corporate Tax Payer Registry Number

- **@CPF** - Brazilian Individual Taxpayer Registry Number

- **@TituloEleitoral** - Brazilian voter ID

- **@NIP** - Polish VAR ID

- **@PESEL** - Polish National Validation Number

- **@REGON** - Polish Taxpayer ID

2022

Validation and Spring Framework

- Spring Framework has robust support for bean validation

- Validation support can be used in controllers, and services, and other Spring managed components

- Spring MVC will return a 400 Bad Request Error for validation failures

- Spring Data JPA with throw an exception for JPA constraint violations

Spring Boot and Validation

- Spring Boot will auto-configure validation when the validation implementation is found on classpath

  - If API is only on classpath (with no implementation) you can use the annotations, BUT validation will **NOT** occur

- Prior to Spring Boot 2.3, validation was included in starter dependencies

  - After Spring Boot 2.3, you must include the Spring Boot validation starter

SPRING FRAMEWORK GURU