Hibernate Validator JSR303 Example Tutorial

APRIL 2, 2018 BY PANKAJ — 6 COMMENTS

Welcome to Hibernate Validator Example Tutorial. Data validation is integral part of any application. You will find data validation at presentation layer with the use of Javascript. Then at the server side code before processing the client data. Also data validation occurs before persisting it, to make sure it follows the correct format.

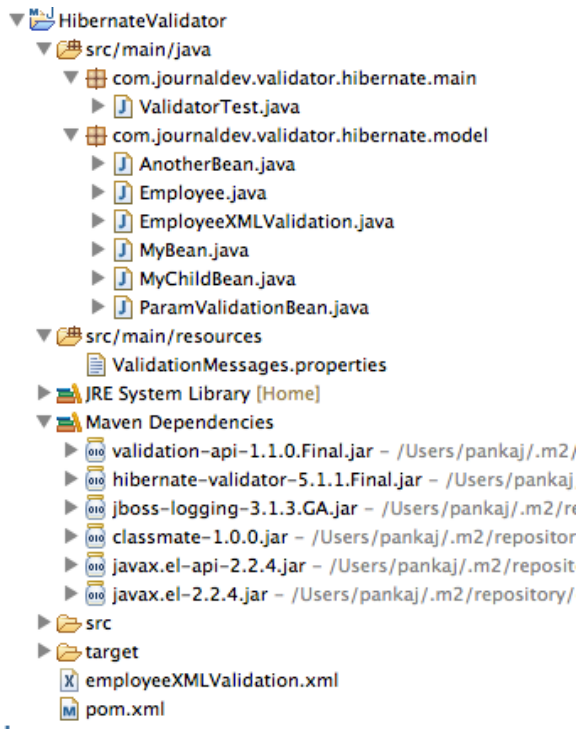
Table of Contents [hide]

Hibernate Validator

Validation is a cross cutting task, so we should try to keep it apart from our business logic. That's why JSR303 and JSR349 provides specification for validating a bean by using annotations. **Hibernate Validator** provides the reference implementation of both these bean validation specs.

It's very easy to use Hibernate Validator and best part is that we can easily extend it and create our own custom validation annotations. Today we will look into the hibernate validator in detail with examples. Finally we will have a test program to check out the validations.

I have created a sample project for all the hibernate validation example, below image shows the final project structure.



Hibernate Validator Maven Dependencies

We need to add JSR303 and Hibernate Validator dependencies to use them.

```
<dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>1.1.0.Final</version>
</dependency>
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
```

```
        <version>5.1.1.Final</version>
</dependency>
```

Hibernate Validator also requires an implementation of the Unified Expression Language (JSR 341) for evaluating dynamic expressions in constraint violation messages.

If your application is running in a servlet container such as JBoss, it's already provided. But if you are using it in a standalone application like my example project, you need to add them manually. Required dependencies are;

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

Check the image for all the maven dependencies in the project. Now we are ready to get started with hibernate validation examples.

Hibernate Validation Example

Let's create a simple class and add some validations to it.

Employee.java

```
package com.journaldev.validator.hibernate.model;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.CreditCardNumber;
import org.hibernate.validator.constraints.Email;

public class Employee {

        @Min(value=1, groups=EmpIdCheck.class)
```

```
    private int id;

    @NotNull(message="Name cannot be null")
    @Size(min=5, max=30)
    private String name;

    @Email
    private String email;

    @CreditCardNumber
```

We should avoid using implementation specific annotations for loose coupling. But hibernate validator provides some very good validation annotations such as `@Email` and `@CreditCardNumber`.

We can also provide custom error message to use with any validation. If there are no message defined then hibernate built-in error message will be used.

We can assign groups to any validation, this can be useful to validate a set of fields only. For example, if I just need to validate the Employee id field, I can use `EmpIdCheck` group. For this we need to define a class/interface.

`EmpIdCheck.java`

```
package com.journaldev.validator.hibernate.model;

public interface EmpIdCheck {
}
```

We will look it's usage in the test program later on.

Hibernate Validator Custom Error Messages

We can define our custom error messages too, all we need is to have `ValidationMessages.properties` file in the classpath.

`ValidationMessages.properties`

```
#Hibernate Bug for @CreditCardNumber Workaround -
https://hibernate.atlassian.net/browse/HV-881
org.hibernate.validator.constraints.LuhnCheck.message=The check digit for
${validatedValue} is invalid, Luhn Modulo 10 checksum failed.

org.hibernate.validator.constraints.Email.message=Invalid email address
```

These property files also support localization, in that case we need to keep file names like ValidationMessages_tr_TR.properties

The property name for message is fully classified annotation name with message at the end, you can easily figure out from above examples.

XML Based Constraints Validation

Sometimes we want validation on third party classes, then we can't use annotations with them. In this situation, xml configuration based validation comes handy. For example, let's say we have a class without any validation annotations like below.

EmployeeXMLValidation.java

```
package com.journaldev.validator.hibernate.model;

public class EmployeeXMLValidation {

	private int id;

	private String name;

	private String email;

	private String creditCardNumber;

	//default no-args constructor
	public EmployeeXMLValidation(){}

	public EmployeeXMLValidation(int id, String name, String email, String ccNum)
{
		this.id=id;
		this.name=name;
		this.email=email;
		this.creditCardNumber=ccNum;
	}
```

A simple example for above bean hibernate validation could be like below.

employeeXMLValidation.xml

```
<constraint-mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/mapping
validation-mapping-1.1.xsd"
        xmlns="http://jboss.org/xml/ns/javax/validation/mapping" version="1.1">

        <default-package>com.journaldev.validator.hibernate.model</default-package>

        <bean class="EmployeeXMLValidation" ignore-annotations="true">
                <field name="id">
                        <constraint annotation="javax.validation.constraints.Min">
                                <element name="value">1</element>
                        </constraint>
                </field>
                <field name="name">
                        <constraint annotation="javax.validation.constraints.NotNull"
/>
                        <constraint annotation="javax.validation.constraints.Size">
                                <element name="min">5</element>
                                <element name="max">30</element>
                        </constraint>
                </field>
                <field name="email">
```

default-package is used to define the base package, so that we can skip long package names.

ignore-annotations is used to tell Hibernate validator to ignore any annotations present in the class for validation purpose, only perform validations as configured in the xml file.

We can have multiple bean validations in a single file, we need to load this file to validator factory configuration, an example of this will be given later on.

Property level constraints

We can apply constraints on the getter methods too, we should not apply it on setter method. Also we should avoid applying constraints on both fields and getter method because then it will be validated twice.

MyBean.java

```
package com.journaldev.validator.hibernate.model;

import javax.validation.constraints.NotNull;

public class MyBean {

        private String name;
```

```
        @NotNull
        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
}
```

Hibernate Validation with Inheritance

Bean validations are inherited, so if we are validating object of child class then any validations in parent class will also be executed.

`MyChildBean.java`

```
package com.journaldev.validator.hibernate.model;

import javax.validation.constraints.NotNull;

public class MyChildBean extends MyBean {

        private String data;

        @NotNull
        public String getData() {
                return data;
        }

        public void setData(String data) {
                this.data = data;
        }
}
```

If we will validate instance of `MyChildBean`, MyBean name field will also be validated.

Validation with Composition using @Valid annotation

We can use `@Valid` annotation incase of composition, so that it's validations are executed.

`AnotherBean.java`

```
package com.journaldev.validator.hibernate.model;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;

public class AnotherBean {

	@NotNull
	@Valid
	private MyChildBean childBean;

	public MyChildBean getChildBean() {
		return childBean;
	}

	public void setChildBean(MyChildBean childBean) {
		this.childBean = childBean;
	}
}
```

Now if we validate `AnotherBean` instance, `MyChildBean` object will also be validated.

Method Parameter Hibernate Validation

We can define constraints for method parameters too, a simple example is given below.

`ParamValidationBean.java`

```
package com.journaldev.validator.hibernate.model;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class ParamValidationBean {

	private String name;

	//using @NotNull at constructor rather than at field
	public ParamValidationBean(@NotNull String name){
		this.name = name;
	}
```

```
	public void printData(@NotNull @Size(min=5) String data){
		System.out.println("Data is::"+data);
	}

	public String getName() {
		return name;
	}
```

Hibernate Validator Example Test Program

We have seen a lot of validation scenarios, here is the test program to show the process to validate them.

ValidatorTest.java

```
package com.journaldev.validator.hibernate.main;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.lang.reflect.Method;
import java.util.Set;

import javax.validation.Configuration;
import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;
import javax.validation.executable.ExecutableValidator;

import org.hibernate.validator.HibernateValidator;

import com.journaldev.validator.hibernate.model.AnotherBean;
import com.journaldev.validator.hibernate.model.EmpIdCheck;
import com.journaldev.validator.hibernate.model.Employee;
import com.journaldev.validator.hibernate.model.EmployeeXMLValidation;
import com.journaldev.validator.hibernate.model.MyChildBean;
```

When we run above hibernate validator example program, we get following output.

```
Jul 25, 2014 3:51:56 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.1.Final

Simple field level validation example
```

```
{javax.validation.constraints.Size.message}::name::size must be between 5 and 30
{org.hibernate.validator.constraints.CreditCardNumber.message}::creditCardNumber::inval
 credit card number
{org.hibernate.validator.constraints.Email.message}::email::Invalid email address

XML Based validation example
{org.hibernate.validator.constraints.Email.message}::email::email::Invalid email
address
{javax.validation.constraints.Size.message}::Name::name::size must be between 5 and
30
{org.hibernate.validator.constraints.CreditCardNumber.message}::123::creditCardNumber::
 credit card number

Validation Group example
{javax.validation.constraints.Min.message}::id::must be greater than or equal to 1

Validation with inheritance example
```

Important points from above hibernate validator test program are:

1. **Validator** instance is thread safe, so we can cache it and reuse it.
2. If there are multiple JSR303 implementation present, then we can get the Hibernate Validator instance using `Validation.byProvider()` method.
3. Notice the use of validation of a group, it's validating only the fields that are part of the group.
4. **ExecutableValidator** is used to validate the parameters of a method.
5. `ExecutableValidator` provide methods to validate constructor parameters and return values too, these methods are `validateReturnValue()`, `validateConstructorParameters()` and `validateConstructorReturnValue()`

Hibernate Validator Custom Validation and Spring Framework integration

We can create our own custom validator too, please read Spring Hibernate Validator Example for better understanding of this feature.

Hibernate Validator Summary

Hibernate Validator and JSR303 together provides base of cross cutting bean validation implementation, if used properly with Aspect Oriented Programming, we can separate all our bean validations from business logic. You can download the final project from below link and explore more.

Download Hibernate Validator Project

References:

Hibernate Validator

JSR 303

« PREVIOUS
Struts2 Hibernate Integration Example Tutorial

NEXT »
Hibernate Interview Questions and Answers

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: HIBERNATE

Comments

Suresh says
MAY 18, 2015 AT 10:14 AM
Nice tutorial thanks……
Reply

hoaibao says
MAY 5, 2015 AT 10:14 AM
good tutorial , thanks you very much
Reply

Manik Jain says
APRIL 14, 2015 AT 11:24 PM
Is there any way to track the return of validator.validate(bean) method for Custom Validators?
Plz let me know.
Reply

Nilanjan Sarkar says
JANUARY 7, 2015 AT 4:35 AM
The xml validation is not working. Could you please tell me what could be the possible reason
Reply

Reddisekhar Reddy says
NOVEMBER 4, 2014 AT 2:49 AM
its interesting and helpful one, in which version the validations included in hibernate?
Reply

Rahul says
OCTOBER 6, 2014 AT 6:30 AM
Pankaj, Nice tutorial. Thanks.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

HIBERNATE FRAMEWORK

Hibernate Tutorial

- Hibernate Example
- Hibernate SessionFactory
- Hibernate Session get load
- Hibernate Session save

RECOMMENDED TUTORIALS

- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial