Seam Validation Module
(TECHNOLOGY PREVIEW)

Reference Guide

3.1.0.Beta3

by Gunnar Morling

Introduction

The Seam Validation module aims at integrating *Hibernate Validator* [http://validator.hibernate.org/], the reference implementation for the Bean Validation API (*JSR 303* [http://jcp.org/en/jsr/detail?id=303]), with CDI (*JSR 299* [http://jcp.org/en/jsr/detail?id=299]).

This integration falls into two main areas:

- Enhanced dependency injection services for validators, validator factories and constraint validators
- Automatic validation of method parameters and return values based on Hibernate Validator's method validation feature

Note

The Seam Validation module is based on version 4.2 or later of Hibernate Validator. As of March 2011 Hibernate Validator 4.2 is still in the works and no final release exists yet.

This means that - though unlikely - also changes to the API of the Seam Validation module might become necessary.

The Seam Validation module is therefore released as a technology preview with the Seam 3 release train, with a final version following soon. Nevertheless you should give it a try already today and see what the Seam Validation module and especially the automatic method validation feature can do for you. Please refer to the *module home page* [http://seamframework.org/Seam3/ValidationModule] for any news on Seam Validation.

The remainder of this reference guide covers the following topics:

- *Installation* of Seam Validation
- *Dependency injection* services for Hibernate Validator
- Automatic *method validation*

Installation

This chapter describes the steps required to getting started with the Seam Validation Module.

2.1. Prerequisites

Not very much is needed in order to use the Seam Validation Module. Just be sure to run on JDK 5 or later, as the Bean Validation API and therefore this Seam module are heavily based on Java annotations.

2.2. Maven setup

The recommended way for setting up Seam Validation is using *Apache Maven* [http://maven.apache.org/]. The Seam Validation Module artifacts are deployed to the JBoss Maven repository. If not yet the case, therefore add this repository to your `settings.xml` file (typically in `~/.m2/settings.xml`) in order to download the dependencies from there:

Example 2.1. Setting up the JBoss Maven repository in settings.xml

```
...
<profiles>
   <profile>
      <repositories>
         <repository>
            <id>jboss-public</id>
            <url>http://repository.jboss.org/nexus/content/groups/public-jboss/</url>
            <releases>
               <enabled>true</enabled>
            </releases>
            <snapshots>
               <enabled>false</enabled>
            </snapshots>
         </repository>
      </repositories>
   </profile>
</profiles>

<activeProfiles>
   <activeProfile>jboss-public</activeProfile>
</activeProfiles>
...
```

General information on the JBoss Maven repository is available in the *JBoss community wiki* [http://community.jboss.org/wiki/MavenGettingStarted-Users], more information on Maven's `settings.xml` file can be found in the *settings reference* [???].

Having set up the repository you can add the Seam Validation Module as dependency to the `pom.xml` of your project. As most Seam modules the validation module is split into two parts, API and implementation. Generally you should be using only the types from the API within your application code. In order to avoid unintended imports from the implementation it is recommended to add the API as compile-time dependency, while the implementation should be added as runtime dependency only:

Example 2.2. Specifying the Seam Validation Module dependencies in pom.xml

```
...
<properties>
   <seam.validation.version>x.y.z</weld.version>
</properties>

...

<dependencies>
   ...
   <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>seam-validation-api</artifactId>
      <version>${seam.validation.version}</version>
      <scope>compile</scope>
   </dependency>
   <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>seam-validation</artifactId>
      <version>${seam.validation.version}</version>
      <scope>runtime</scope>
   </dependency>
   ...
</dependencies>
...
```

Note

Replace "x.y.z" in the properties block with the Seam Validation version you want to use.

2.3. Manual setup

In case you are not working with Maven or a comparable build management tool you can also add Seam Validation manually to you project.

Just download the latest distribution file from *SourceForge* [http://sourceforge.net/projects/jboss/files/Seam/Validation/], un-zip it and add seam-validation.jar api as well as all JARs contained in the lib folder of the distribution to the classpath of your project.

Dependency Injection

The Seam Validation module provides enhanced support for dependency injection services related to bean validation. This support falls into two areas:

- Retrieval of `javax.validation.ValidatorFactory` and `javax.validation.Validator` via dependency injection in non-Java EE environments
- Dependency injection for constraint validators

3.1. Retrieving of validator factory and validators via dependency injection

As the Bean Validation API is part of Java EE 6 there is an out-of-the-box support for retrieving validator factories and validators instances via dependency injection in any Java EE 6 container.

The Seam Validation module provides the same service for non-Java EE environments such as for instance stand-alone web containers. Just annotate any field of type `javax.validation.ValidatorFactory` with `@Inject` to have the default validator factory injected:

Example 3.1. Injection of default validator factory

```
package com.mycompany;

import javax.inject.Inject;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;

public class MyBean {

   @Inject
   private ValidatorFactory validatorFactory;

   public void doSomething() {

      Validator validator = validatorFactory.getValidator();
      //...
   }
}
```

Note

The injected factory is the default validator factory returned by the Bean Validation bootstrapping mechanism. This factory can customized with help of the configuration file `META-INF/validation.xml`. The Hibernate Validator Reference Guide *describes in detail* [http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html/validator-xmlconfiguration.html] the available configuration options.

It is also possible to directly inject a validator created by the default validator factory:

Example 3.2. Injection of a validator from the default validator factory

```
package com.mycompany;

import java.util.Set;

import javax.inject.Inject;
import javax.validation.ConstraintViolation;
import javax.validation.Validator;

public class MyBean {

   @Inject
   private Validator validator;

   public void doSomething(Foo bar) {

      Set<ConstraintViolation<Foo>> constraintViolations = validator.validate(bar);
      //...
   }
}
```

3.2. Dependency injection for constraint validators

The Seam Validation module provides support for dependency injection within `javax.validation.ConstraintValidator` implementations. This is very useful if you need to access other CDI beans within you constraint validator such as business services etc. In order to make use of dependency injection within a constraint validator implementation it must be a valid bean type as described by the CDI specification, in particular it must be defined within a bean deployment archive.

Warning

Relying on dependency injection reduces portability of a validator implementation, i.e. it won't function properly without the Seam Validation module or a similar solution.

To make use of dependency injection in constraint validators you have to configure `org.jboss.seam.validation.InjectingConstraintValidatorFactory` as the constraint validator factory to be used by the bean validation provider. To do so create the file `META-INF/validation.xml` with the following contents:

Example 3.3. Configuration of InjectingConstraintValidatorFactory in META-INF/validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<validation-config
        xmlns="http://jboss.org/xml/ns/javax/validation/configuration"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration validation-configuration-1.0.xsd">

  <constraint-validator-factory>
    org.jboss.seam.validation.InjectingConstraintValidatorFactory
  </constraint-validator-factory>

</validation-config>
```

Having configured the constraint validator factory you can inject arbitrary CDI beans into you validator implementations. Listing *Example 3.4, "Dependency injection within ConstraintValidator implementation"* shows a `ConstraintValidator` implementation for the `@Past` constraint which uses an injected time service instead of relying on the JVM's current time to determine whether a given date is in the past or not.

Example 3.4. Dependency injection within ConstraintValidator implementation

```
package com.mycompany;

import java.util.Date;

import javax.inject.Inject;
```

```
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
import javax.validation.constraints.Past;

import com.mycompany.services.TimeService;

public class CustomPastValidator implements ConstraintValidator<Past, Date>
{

  @Inject
  private TimeService timeService;

  @Override
  public void initialize(Past constraintAnnotation)
  {
  }

  @Override
  public boolean isValid(Date value, ConstraintValidatorContext context)
  {

    if (value == null)
    {
      return true;
    }

    return value.before(timeService.getCurrentTime());
  }

}
```

Note

If you want to redefine the constraint validators for built-in constraints such as `@Past` these validator implementations have to be registered with a custom constraint mapping. More information can be found in the *Hibernate Validator Reference Guide* [http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html/validator-xmlconfiguration.html#d0e2024].

Method Validation

Hibernate Validator provides several advanced validation features and related functionality which go beyond what is defined by JSR 303 ("Bean Validation API"). One of these additional features is a facility for the validation of method parameters and return values. With that API a style of program design known as "Programming by Contract" can be implemented using the concepts defined by the Bean Validation API.

This means that any Bean Validation constraints can be used to describe

- any preconditions that must be met before a method may legally be invoked (by annotating method parameters with constraints) and
- any postconditions that are guaranteed after a method invocation returns (by annotating methods)

To give an example listing *Example 4.1, "Exemplary repository with constraint annotations"* shows a fictional repository class which retrieves customer objects for a given name. Constraint annotations are used here to express the following pre-/postconditions:

- The value for the name parameter may not be null and must be at least three characters long
- The method may never return null and each Customer object contained in the returned set is valid with respect to all constraints it hosts

Example 4.1. Exemplary repository with constraint annotations

```
@AutoValidating
public class CustomerRepository {

    @NotNull @Valid Set<Customer> findCustomersByName(@NotNull @Size(min=3) String name);

}
```

Hibernate Validator itself provides only an API for validating method parameters and return values, but it does not trigger this validation itself.

This is where Seam Validation comes into play. Seam Validation provides a so called business method interceptor which intercepts client invocations of a method and performs a validation of the method arguments before as well as a validation of the return value after the actual method invocation.

To control for which types such a validation shall be performed, Seam Validation provides an interceptor binding, `@AutoValidating`. If this annotation is declared on a given type an automatic validation of each invocation of any this type's methods will be performed.

If either during the parameter or the return value validation at least one constraint violation is detected (e.g. because `findCustomersByName()` from listing *Example 4.1, "Exemplary repository with constraint annotations"* was invoked with a String only two characters long), a `MethodConstraintViolationException` is thrown. That way it is ensured that all parameter constraints are fulfilled when the call flow comes to the method implementation (so it is not necessary to perform any parameter null checks manually for instance) and all return value constraints are fulfilled when the call flow returns to the caller of the method.

The exception thrown by Seam Validation (which would typically be written to a log file) gives a clear overview what went wrong during method invocation:

Example 4.2. Output of MethodConstraintViolationException

```
org.hibernate.validator.MethodConstraintViolationException: 1 constraint violation(s) occurred
 during method invocation.
Method: public java.lang.Set
 com.mycompany.service.CustomerRepository.findCustomersByName(java.lang.String)
Argument values: [B]
Constraint violations:
  (1) Kind: PARAMETER
      parameter index: 0
      message: size must be between 3 and 2147483647
         root bean: com.mycompany.service.org$jboss$weld$bean-flat-ManagedBean-class_com
$mycompany$service$$CustomerRepository_$$_WeldSubclass@3f72c47b
      property path: CustomerRepository#findCustomersByName(arg0)
                                                                    constraint:

 min=3, max=2147483647, payload=[], groups=[])
```

To make use of Seam Validation's validation interceptor it has to be registered in your component's beans.xml descriptor as shown in listing *Example 4.3, "Registering the validation interceptor in beans.xml"*:

Example 4.3. Registering the validation interceptor in beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
beans_1_0.xsd">

  <interceptors>
    <class>org.jboss.seam.validation.ValidationInterceptor</class>
```

```
  </interceptors>
</beans>
```

It is recommended that you consult the Hibernate Validator *reference guide* [http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html/] to learn more about the method validation feature in general or for instance the rules that apply for constraining methods in inheritance hierarchies in particular.