Bean Validation

Added in 5.2

Bean validation involves validating user input using Tapestry's built-in support for the JSR 303 Bean Validation API.

Tapestry has always provided a powerful non-JSR 303 validation mechanism (see Forms and Validation). Among other things this mechanism allows you to annotate your domain model classes with the @Validate annotation. However, this annotation is problematic if your domain model is used in non-Tapestry applications as well as in Tapestry applications. Your non-Tapestry application becomes dependent on *tapestry5-annotations* module. To make your domain model independent from Tapestry you can use the JSR 303: Bean Validation instead. This library provides integration between Tapestry and JSR-303.

Related Articles

- Bean Validation
- Forms and Validation

Configuration

The Tapestry's JSR 303 - Bean Validation Library is responsible for configuring and bootstrapping the Validator for you. In order to use this library you have to choose an implementation of the JSR-303 specification like Hibernate Validator 4.x. This library is not specific to any implementation of JSR-303 and will work with any implementation of your choice.

Bootstraping the Bean Validator

The BeanValidatorSource service is responsible for bootstrapping the Validator. You can contribute a BeanValidatorConfigurer to the configuration of this service in order to participate on the configuration of Validator.

```
@Contribute(BeanValidatorSource.class)
public static void provideBeanValidatorConfigurer(OrderedConfiguration<BeanValidatorConfigurer> configuration)
{
   configuration.add("MyConfigurer", new BeanValidatorConfigurer()
   {
      public void configure(javax.validation.Configuration<?> configuration)
      {
         configuration.ignoreXmlConfiguration();
      }
   });
}
```

Validation groups

In JSR-303 validation groups are used to define a subset of the constraints validated at a given time. If no validation group is specified the Default group is taken. By default, Tapestry passes only this group to Validator. You can tell Tapestry to pass more groups by contributing group classes into the configuration of the BeanValidatorSource service.

Usage

Validating Input Fields

Once you included this library and its dependencies into your web app, you may use the JSR-303 annotations to validate the user's input.

```
public class Login
{
   @NotNull
   @Size(max=10)
   @Pattern(regexp = "[a-zA-Z]*")
   @Property @Persist
   private String userName;

   @NotNull
   @Size(min=5, max=30)
   @Property @Persist
   private String password;

   void onSuccess()
   {
      // Login the user here
   }
}
```

You can even mix JSR-303 annotations and Tapestry's @Validate annotation.

```
public class Login
{
   @NotNull
   @Validate("maxlength=10")
   @Pattern(regexp = "[a-zA-Z]*")
   @Property @Persist
   private String userName;

   @NotNull
   @Validate("minlength=5,maxlength=30")
   @Property @Persist
   private String password;

   void onSuccess()
   {
      // Login the user here
   }
}
```

Next you have to pass the object to validate into the Form's *validate* parameter. In the following example the Form's fields are bound to the properties of the *Login* page. That's why we pass *this*, thus the page instance, to the *validate* parameter.

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
   <body>
      <t:form validate="this">

         <t:errors/>

         <p>
            <t:textfield t:id="userName"/>
         </p>

         <p>
            <t:textfield t:id="password"/>
         </p>

         <p>
            <input type="submit" value="Login"/>
         </p>
      <t:form>
   </body>
</html>
```

Since the *validate* parameter defaults to the container of the Form component, we could also remove *validate="this"* in the example above.

Validating Beans with BeanEditForm

If you use the BeanEditForm component it's even easier to validate your beans. The only thing you have to do is to annotate your beans with JSR-303 annotations. If you are migrating from Tapestry's built-in validation mechanism to JSR-303 Bean Validation, you don't have to change your template at all.

```
public class User
{
   @NotNull
   private String userName;

   @NotNull
   @Validate("minlength=10")
   private String password;

   ...
}
```

Client-side Validation

Unfortunately JSR-303 doesn't cover client-side validation, so web frameworks supporting this JSR need to come up with proprietary client-side solutions. Tapestry provides client-side validation for the following JSR-303 constraints:

JSR-303 constraint	Tapestry' JavaScript function
@Max	Tapestry.Validator.maxnumber
@Min	Tapestry.Validator.minnumber
@NotNull	Tapestry.Validator.notnull
@Null	Tapestry.Validator.isnull
@Pattern	Tapestry.Validator.pattern
@Size	Tapestry.Validator.size
@AssertTrue *(Since 5.4.5)*	
@AssertFalse *(Since 5.4.5)*	

Providing own client-side validators

Now let's see how to provide own client-side validation for JSR-303 constraints. Imagine you created the following constraint definition. The server-side implementation of the constraint is implemented by RangeValidator. I suppose you are familiar with JSR-303, so we won't explain how to implement RangeValidator.

```
@Documented
@Constraint(validatedBy = RangeValidator.class)
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
public @interface Range {
   long max() default Long.MAX_VALUE;

   long min() default Long.MIN_VALUE;

   String message() default "{com.acme.constraint.Range.message}";

   Class[] groups() default {};

   Class[] payload() default {};
}
```

To provide client-side validation of a constraint you have to add a JavaScript function to the built-in *Tapestry.Validator* JavaScript-object. The function should contain exactly three parameters:

1. Field being validated
2. Validation message
3. JSON object with values from the constraint annotation

Here is an example:

```
Tapestry.Validator.range = function(field, message, spec) {
   field.addValidator(function(value) {
      if (value < spec.min || value > spec.max) {
         throw message;
      }
   });
};
```

Now you have to tell Tapestry to call the function *Tapestry.Validator.range* when client-side validation of *@Range* should be executed. This is accomplished by a contribution to the ClientConstraintDescriptorSource service. The configuration of this service is a collection of ClientConstraintDescriptor. Each *ClientConstraintDescriptor* represents a client-side validation constraint. The constructor of *ClientConstraintDescriptor* has three parameters:

1. Class of the constraint annotation.
2. Name of the JavaScript function.
3. The last parameter is a varargs. It is used to pass the attribute names of the constraint annotation to be passed (along with their values) to the JavaScript function as an JSON object.

The last step is to make the contribution, which links the *@Range* annotation with the JavaScript function *range*. The attributes *max* and *min* and their values are passed to the function.

```
@Contribute(ClientConstraintDescriptorSource.class)
public static void provideClientConstraintDescriptors(Configuration<ClientConstraintDescriptor> config) {

   config.add(new ClientConstraintDescriptor(Range.class, "range", "min", "max"));
}
```