Advanced Bean Validation Concepts and Examples

This chapter describes how to create custom constraints, custom validator messages, and constraint groups using the Java API for JavaBeans Validation (Bean Validation).

The following topics are addressed here:

- Creating Custom Constraints
- Customizing Validator Messages
- Grouping Constraints

Creating Custom Constraints

Bean Validation defines annotations, interfaces, and classes to allow developers to create custom constraints.

Using the Built-In Constraints To Make a New Constraint

Bean Validation includes several built-in constraints that can be combined to create new, reusable constraints.

This can simplify constraint definitions by allowing developers to define a custom constraint made up of several built-in constraints that may then be applied to component attributes with a single annotation.

Example 47-1 The `@Email` Constraint

```
@Pattern.List({
@Pattern(regexp =
"[a-z0-9!#$%&'*+/=?^_`{|}~-]
+(?:\\."
+"[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*"
+
"@(?:[a-z0-9](?:[a-z0-9-]*
[a-z0-9])?\\.)+[a-z0-9]
(?:[a-z0-9-]*[a-z0-9])?")
})
@Constraint(validatedBy = {})
```

```
@Documented
@Target({ElementType.METHOD,
ElementType.FIELD,
ElementType.ANNOTATION_TYPE,
ElementType.CONSTRUCTOR,
ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface Email {
String message()
default "{invalid.email}";
```

```
Class<?>[] groups() default {};
Class<? extends Payload>[]
payload() default {};
@Target({ElementType.METHOD,
ElementType.FIELD,
ElementType.ANNOTATION_TYPE,
ElementType.CONSTRUCTOR,
ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Documented
```

```
@interface List
{ Email[] value(); }
}
```

This custom constraint can then be applied to an attribute.

```
...
@Email
protected String email;
...
```

Customizing Validator Messages

Bean Validation includes a resource bundle of default messages for the build-in constraints.

These messages can be customized, and localized for non-English speaking locales.

The `ValidationMessages` Resource Bundle

The `Validationmessages` resource bundle and the locale variants of this resource bundle contain strings that override the default validation messages.

The `ValidationMessages` resource bundle is typically a properties file, `ValidationMessages.properties`, in the default package of an application.

Localizing Validation Messages

Locale variants of `ValidationMessages.properties` are added by appending an underscore and the locale prefix.

For example, the Spanish locale variant resource bundle would be `ValidationMessages_es.properties`.

Grouping Constraints

Constraints may be added to one or more groups.

Constraint groups are used to create subsets of constraints, so only certain constraints will be validated for a particular object.

By default, all constraints are included in the `Default` constraint group.

Constraint groups are represented by interfaces.

```
public interface Employee {}
public interface Contractor {}
```

Constraint groups can inherit from other groups.

```
public interface Manager extends
Employee {}
```

When a constraint is added to an element, the constraint declares which groups that constraint belongs by specifying the class name of the group interface name in the `groups` element of the constraint.

```
@NotNull(groups=Employee.class)
Phone workPhone;
```

Multiple groups can be declared by surrounding the groups with angle brackets ({ and }) and separating the groups class names with commas.

```
@NotNull(groups=
{Employee.class,Contractor.class })
Phone workPhone;
```

If a group inherits from another group, validating that group results in validating all constraints declared as part of the supergroup.

For example, validating the `Manager` group results in the `workPhone` field being validated, because `Employee` is a super-interface of `Manager`.

Customizing Group Validation Order

By default, constraint groups are validated in no particular order.

There are some cases where some groups should be validated before others.

For example, in a particular class basic data should be validated before more advanced data.

To set the validation order for a group, add a `javax.validation.GroupSequence` annotation on the interface definition, listing the order in which the validation should occur.

```
@GroupSequence({Default.class,
ExpensiveValidationGroup.class})
public interface
FullValidationGroup {}
```

When validating `FullValidationGroup`, first the `Default` group is validated.

If all the data passes validation, then the `ExpensiveValidationGroup` group is validated.

If a constraint is part of part of both the `Default` and `ExpensiveValidationGroup` groups, the constraint is validated as part of the `Default` group, and will not be validated on the subsequent `ExpensiveValidationGroup` pass.