Bean Validation BVAL-208

Support groups translation during cascaded validations

Details

Type:	Improvement	Status:	Resolved
Priority:	Major	Resolution:	Fixed
Affects Version/s:	1.0 final	Fix Version/s:	1.1.0.Beta1 (public draft 1)
Component/s:	None		
Labels:	None		

Description

Cascaded validation via @*Valid* should allow to transform validated groups when traversing from one object to the other. Something like:

```
public class User {
    @Valid
    @ConvertGroup.List( {
        @ConvertGroup(from=Default.class, to=BasicPostal.class),
        @ConvertGroup(from=Complete.class, to=FullPostal.class)
    } )
    Set<Address> getAddresses() { ... }
}
```

```
public class User {
    @Valid
    @ConvertGroup(
        from={Default.class, Complete.class},
        to={BasicPostal.class, FullPostal.class}
    )
    Set<Address> getAddresses() { ... }
}
```

Issue Links

is followed up by	HV-638 Support groups translation during cascaded validations

Activity

All Comments Work Log History Activity Commits

Emmanuel Bernard added a comment - 07/May/10 9:03 AM
There are valid reasons to get groups on @Valid but I did not get yours. Can you explain that a bit further?

Marc Schipperheyn added a comment - 07/May/10 9:35 AM
Hi, yes. In order not to have a huge amount of valeue objects, I use groups to select which attributes to include in validation.
With the @Valid option in Spring MVC, we can validate these Value Objects. It would be great to be able to select based on groups to limit the amount of Value Objects in the build.

Emmanuel Bernard added a comment - 10/May/10 3:21 AM
OK I see.

Gary Fleming added a comment - 18/May/10 10:20 AM
I've hit into this a couple of times, so would like to see it added. Additionally, there are two issues for this in the Spring JIRA, suggesting a custom annotation. I'd rather see this added to the spec, than have the functionality elsewhere:
http://jira.springframework.org/browse/SPR-6373
http://jira.springframework.org/browse/SPR-7062

Arthur Ronald F D Garcia added a comment - 24/Aug/10 1:35 PM - edited
Here goes some examples

Spring MVC 3.0, for instance, supports JSR-303 by using @Valid as ElementType.PARAMETER

```
public void doSomething(@Valid Command command, BindingResult result) throws Exception {}
```

But if JSR-303 supports the concept of groups, how can i supply which group i want to test. Because of that, i need to implement a Spring Validator interface instead

```
public class CommandValidator implements Validator {
        public boolean supports(Class clazz) {}

        public void validate(Object command, Errors errors) {
            Set<ConstraintViolation<Command>> constraintViolationSet = ValidatorUtil.getValidator().validate(((
            // post-proccessing validation
        }
    }
```

So instead of (Not supported)

```
public void doSomething(@Valid(groups=SomeUseCase.class) Command command, BindingResult result) throws Exceptio
```

I need to use (old-style)

```
private CommandValidator validator = new CommandValidator();

    public void doSomething(Command command, BindingResult result) throws Exception {
        validator.validate(command, result);
    }
```

There is more As @Valid annotation fullfil nested properties, how can i supply which group of the nested property i want to test ??? As @Valid does not support groups attribute, i just can use @Valid when using default group. Otherwise, i need to remove it

```
public class Command {
        @Valid
        public Children getChildren() { return this.children; }
    }
```

best regards

Arthur Ronald F D Garcia

Arthur Ronald F D Garcia added a comment - 22/Oct/10 11:45 AM

The JSR-303 javax.validation.Valid API is clear

Mark an association as cascaded. The associated object will be validated by cascade.

When Spring uses @Valid AS A MARKER to validate its command objects, it corrupts its purpose. Spring should instead create an specific annotation in which you can supply the target group (s). My vote has been removed.

Arthur Ronald F D Garcia added a comment - 22/Oct/10 12:11 PM

If @Valid annotation supports groups attribute, javax.validation API may have INCONSISTENT behavior because the validated group MUST BE PROPAGATED to its referenced objects annotated with @Valid annotation.

Hardy Ferentschik added a comment - 08/Sep/11 10:44 AM

I agree that the provided examples are not in the spirit of BV. So I am wondering which use case Emmanuel had in mind (which makes sense)?

Emmanuel Bernard added a comment - 15/Sep/11 2:47 AM - edited

I had two use cases in mind:

- define the group to validate in a method level annotation (parameter etc)
- let a user redefine the group to validate when cascaded (useful when aggregating object graphs)

Both use cases are well described by Arthur in comment-38181

Note that @Valid left alone would still fulfill the default specification interpretation. But one could envision a "NAT" style transformation from one group to another when a @Valid barrier is crossed.

```
@Valid(before={Simple.class, Heavy.class}, after={Easy.class, Default.class})
```

When Simple.class is validated in the mail object, Easy.class is validated on the cascaded object.
When Heavy.class is validated in the mail object, Default.class is validated on the cascaded object.

A couple of problems to resolve:

- what about circular dependencies involving different groups "for a while" (ie Default on the nested object validates back the root object).

- what about group sequences. Should direct groups be translated, or should inner groups be too? What about inconsistencies in this case (ie the translated group sequence violating group sequence rules).

Note that I had in mind to reuse @Valid but it might be better to have one (or even two) additional annotations for these notions (ie the cascading, the group translation, the method-levle group definition.

Hardy Ferentschik added a comment - 15/Sep/11 3:07 AM

> Note that I had in mind to reuse @Valid but it might be better to have one (or even two) additional annotations for these notions (ie the cascading, the group translation, the method-levle group definition.

+1 for new annotations. I think just reusing *@Valid* will cause more problems than it solves.

Jan Vos added a comment - 03/Apr/12 1:09 AM

Our use case for the grouping is the following. If the bean is in a DRAFT state, nothing should be validated, if the bean is in a SENT state, validation should be executed. This works for all fields of the bean itself with the grouping in place, but everything marked with @Valid is always executed.

Hardy Ferentschik added a comment - 03/Apr/12 8:54 AM - edited

> - define the group to validate in a method level annotation (parameter etc)

I am not quite sure this is needed. If you have an actual constraint (*NotNull*, *Size*, etc) you can specify the groups. If you want to validate a whole object you add *@Valid* to the parameter and the actual constraints (with their specified) groups are getting validated. I guess what I am saying here is that a group belongs to an actual constraint not to a marker (*@Valid*) which basically says all constraints of the anntated element must be valid.

```
public interface GroupA {
}
```

```
public interface GroupB {
}
```

```
public class Command {
    @NotNull(groups = GroupA.class)
    String name;
}
```

```
public class Executor {
    public void doSomething(@Valid(groups=GroupB.class) Command command) {
        // ...
    }
}
```

```
// something like this to handle method validation
public class ValidationInvocationHandler {
    //...

    public Object invoke(Object object, Method method, Object[] args) throws Throwable {
        Set<MethodConstraintViolation<Object>> constraintViolations;
        // ...
        constraintViolations = validator.validateAllParameters( object, method, args, GroupB.class );

        if ( !constraintViolations.isEmpty() ) {
            throw new MethodConstraintViolationException( constraintViolations );
        }

        Object result = method.invoke( wrapped, args );

        constraintViolations = validator.validateReturnValue( wrapped, method, result, groups );

        if ( !constraintViolations.isEmpty() ) {
            throw new MethodConstraintViolationException( constraintViolations );
        }

        return result;
}
```

Is a newly created *Command* instance with a *null* name valid in this case?

Emmanuel Bernard added a comment - 03/Apr/12 9:05 AM

Well we are considering it already for @MethodValidated. Are you saying you want to undo that?

Hardy Ferentschik added a comment - 03/Apr/12 10:10 AM

I think *@MethodValidated* is a different beast. I lets *@Valid* and constraint declarations intact. I guess I am most worried about modifying

@*Valid*.

Emmanuel Bernard added a comment - 03/Apr/12 11:23 AM
I agree with this concern. I think that if we bring this group translation idea to fruition, it is going to be a dedicated annotation.
@ConverGroup(from=Foo.clas, to=Bar.class)

Hardy Ferentschik added a comment - 03/Apr/12 11:26 AM

> I agree with this concern. I think that if we bring this group translation idea to fruition, it is going to be a dedicated annotation.
> @ConverGroup(from=Foo.clas, to=Bar.class)

+1

val added a comment - 15/Aug/12 7:27 PM
+1 For this feature, I have run up against this problem multiple times already.

The simplest usecase I can come up with is a wizard like flow building up a credit card payment details bean, which may have two addresses shipping and billing. Step one you want to validate the shipping address, step two you need to validate the billing address.

@*Entity*
public class PaymentDetails{

@*Embeded*
@*Valid*(groups={IContactForm.class})
Address address;

@*Embedded*
@*Valid*(groups={IBillingForm.class})
@*AttributeOverrides*({@AttributeOverride(name="address1", column=@Column(name="billing_address1")), ..., ..., })
Address billingAddress;

}

Hardy Ferentschik added a comment - 30/Aug/12 8:11 AM
Some thoughts on group sequence resolution (correct me from wrong):

- the group execution order defined by the groups and groups sequences passed to the *Validator#validate* can still be resolved statically (as it is done now). There is no need to know anything about group translations or default group redefinitions
- a group transformation via *@ConvertGroup* (or whatever we are going to name this annotation) behaves similar to a redefined default group sequence. Effectively when the group translation is encountered a 'sub' validate call needs to be executed with the current bean as validated object and the translated groups as group parameters

People

Assignee:	Emmanuel Bernard
Reporter:	Marc Schipperheyn
Participants:	Arthur Ronald F D G… , (6)

Vote (4) Watch (7)

Dates

Created:	06/May/10 6:50 AM
Updated:	09/Nov/12 5:22 AM
Resolved:	18/Oct/12 10:15 AM