Bean Validation

Bean Validation trata-se de uma aplicação baseada em **Hibernate Validator**, traz uma seria de **annotations** que servem para validação de dados.

Validadores Nativos:

@**AssertFalse :**O valor do campo ou propriedade deve ser falso.

@**AssertTrue :**O valor do campo ou propriedade deve ser verdadeiro.

@**DecimalMax :**O valor do campo ou propriedade deve ser um número decimal **menor ou igual** ao informado no atributo **value** da anotação.

@**DecimalMin :**O valor do campo ou propriedade deve ser um número decimal **maior ou igual** ao informado no atributo **value** da anotação.

@**Digits :**O valor do campo ou propriedade deve ser um número com a quantidade de dígitos menor ou igual à quantidade informada na anotação.

@**Future :**O valor do campo ou propriedade deve ser uma data futura em relação à atual.

@**Max :**O valor do campo ou propriedade deve ser menor ou igual ao especificado no atributo **value** da anotação.

@**Min :** O valor do campo ou propriedade deve ser maior ou igual ao especificado no atributo **value** da anotação.

@**NotNull :** O valor do campo ou propriedade não pode ser nulo.

@**Null :** O valor do campo ou propriedade deve ser nulo.

@**Past :** O valor do campo ou propriedade deve ser uma data passada em relação à atual.

@**Pattern :** O valor do campo ou propriedade deve obedecer à expressão regular informada no atributo **regexp** da anotação.

@**Size :** O valor do campo ou propriedade deve obedecer aos limites informados na anotação através dos atributos **min** e **max**. Aplica-se a Strings, Collections e arrays.

Validadores personalizados:

Exemplo:

```
@Target({FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy =
StringAsBigDecimalValidator.class)
public @interface StringAsBigDecimalValid {

    String message() default "Valor inválido";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    String value() default "";
}
```

@Target({FIELD}): A anotação apenas pode ser utilizada em fields, ou seja, em variáveis globais da classe;
@Retention(RUNTIME): Essa validação é apenas em tempo de execução;
@Constraint(validatedBy = StringAsBigDecimalValidator.class): Define que a classe StringAsBigDecimalValidator será responsável pela implementação da validação.

Os atributos message, groups, payload e value são os padrões das anotações de Bean Validation, onde tem os seguintes objetivos:

message: A mensagem emitida em caso de erro na validação;
groups: Para utilização de grupo de validações de Bean Validation;
payloads: Para configurar o grau do erro de validação;
value: É a propriedade que recebe o valor do atributo inserido.

Após criado a anotação para invocar a validação, é preciso implementar regra de validação, que é feita em uma classe concreta StringAsBigDecimalValidator que foi referenciada na anotação @Constraint.

A classe de validação implementa ConstraintValidator e precisa sobrescrever o método isValid, que o método que vai considerar se o valor recebido através do parâmetro value é valido ou não.

```
import com.google.common.base.Strings;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class StringAsBigDecimalValidator implements
ConstraintValidator<StringAsBigDecimalValid, String> {

    private String value;

    @Override
    public void initialize(StringAsBigDecimalValid
constraintAnnotation) {
        this.value = constraintAnnotation.value();
    }

    @Override
    public boolean isValid(String value,
ConstraintValidatorContext constraintValidatorContext) {
        if(Strings.isNullOrEmpty(value)) {
            return true;
        }

        try {
            new BigDecimal(value);
            return true;
        } catch (NumberFormatException nfex) {
            return false;
        }
    }
}
```

No final apenas acrescentamos a anotação na variável:

```
 @StringAsBigDecimalValid(message = "Valor inválido")
```

Grupos de validação

Se não for especificado nada o grupo padrão é o **default**, para especificar um grupo, usamos uma classe qualquer, geralmente criada apenas para representa-lo

Integração JPA e Bean Validation

Integrar JPA e Bean Validation é fácil basta que haja um **Validation Provider**, que é uma implementação do Bean Validation no ambiente, assim a JPA automaticamente passará a usa-la.

```
<persistence-unit name="default" transaction-type="RESOURCE_LOCAL">
<validation-mode>NONE< /validation-mode>
<properties>
...
</properties>
</persistence-unit>
```

Na tag validation-mode, os valores possíveis são:

NONE: desabilita a integração entre as especificações;
CALLBACK: explicitamente liga a integração, lançando uma exceção caso um provedor não esteja disponível. Pode ser interessante para assegurar que a aplicação não irá executar sem as validações previstas;
AUTO: habilita a validação se o provider for encontrado.

A JPA por padrão valida o grupo Default ao inserir e ao realizar a atualização dos registros. Podemos, porém, alterar esse comportamento para validar apenas o grupo “ValidacaoMinima”. Vamos adicionar ao arquivo persistence.xml as seguintes propriedades:

```
<property name="javax.persistence.validation.group.pre-persist"
value="facesmotors.validation.groups.ValidacaoMinima" />

<property name="javax.persistence.validation.group.pre-update"
value="facesmotors.validation.groups.ValidacaoMinima" />
```

Integração JSF e Bean Validation

A integração Bean Validation e JSF também é automática desde que uma implementação esteja disponível no ambiente da aplicação.

Desde que não desabilitemos a integração, não precisamos fazer nada para usar as anotações como validadores do JSF. De toda forma, temos a tag f:validateBean que, apesar de parecer, não serve para habilitar a validação do JSF via Bean Validation. Usamos essa tag para desabilitar a validação ou então especificar um grupo diferente de validação.

Exemplo:

```
<f:validateBean
validationGroups="facesmotors.validation.groups.ValidacaoMinima">
<h:panelGrid columns="2">
<h:inputText value="#{auto.anoFabricacao}" />
</h:panelGrid>
</f:validateBean>
```