



Bean Validation

Java EE Grundlagen

Inhalte dieses Kapitels

Allgemeines

Validator API

Constraints

Fehlermeldungen

Integration mit anderen Java-EE-Technologien

Allgemeines [1|2]

Motivation

- Validierung als Aufgabe in allen Schichten einer Anwendung
- Implementierung von Validierungslogik in allen Schichten notwendig
 - Aufwand
 - Inkonsistenz
- Validierungslogik wird oft zum *Domain Model* gepackt

→ Standard über alle Schichten benötigt

Allgemeines [2|2]

Bean Validation (JSR 303)

- Metadaten und API für Validierung
- Metadaten über Annotationen und/oder XML
- Verfügbarkeit in
 - *Web Tier*
 - *Persistence Tier*
 - Java-SE-Umgebungen (z.B. Swing-Anwendungen)
- Erweiterung zum allgemeinen JavaBeans-Standard
- Definition von Validierungsregeln (*Constraints*)
- Überprüfung von Validierungsregeln durch *Constraint Validators*


Validator API

■ Aufruf der Validierung

```
// Object to validate
Address address = new Address();
// ...
Validator v = Validation.buildDefaultValidatorFactory().getValidator();
Set<ConstraintViolation<Address>> violations = v.validate(address);
for(ConstraintViolation<Address> violation : violations) {
    // ...
}
```

T extends java.lang.Object : Class

«Java Interface»

 **ConstraintViolation**

- `getMessage() : String`
- `getMessageTemplate() : String`
- `getRootBean() : T`
- `getRootBeanClass() : Class<T>`
- `getLeafBean() : Object`
- `getPropertyPath() : Path`
- `getInvalidValue() : Object`
- `getConstraintDescriptor() : ConstraintDescriptor<?>`

Constraints [1|7]

Field / Property Constraints

- `@AssertTrue`, `@AssertFalse`
- `@Null`, `@NotNull`
- `@Min`, `@Max`, `@DecimalMin`, `@DecimalMax`
- `@Digits`
- `@Size`
- `@Future`, `@Past`
- `@Pattern`
- Erstellung eigener Constraints

Object Constraints

- Validierung des Objekts in Gesamtheit
- Erstellung eigener Constraints

Constraints [2/7]

Beispiel

```
public class Person {  
  
    @NotNull  
    private String name;  
    @DecimalMin("0.5")  
    private double size;  
    @Past  
    private Date birthDate;  
  
    @AssertTrue  
    public boolean isAdult() {  
        // ...  
    }  
  
}
```

Constraints [3|7]

Payload

- Generischer Typ zur Übergabe von Metadaten an Auswerter der Fehlernachrichten

```
public interface Optional extends Payload {}
```

```
@DecimalMin(value = "0.5", payload = Optional.class)  
private double size;
```



```
for(ConstraintViolation<Person> violation : violations) {  
    if(violation.getConstraintDescriptor().getPayload().contains(Optional.class)) {  
        // ...  
    };  
}
```


Constraints [4|7]

Groups

- Logische Gruppierung mehrerer *Constraints*
- Aufruf der Validierung mit Angabe der *Groups*

```
public interface Calculations {}
```

```
@AssertTrue(groups=Calculations.class)
```

```
public boolean isAdult() {
```

```
    // ...
```

```
    return true;
```

```
}
```



```
Validator v = Validation.buildDefaultValidatorFactory().getValidator();
```

```
Set<ConstraintViolation<Person>> violations = v.validate(person, Calculations.class);
```

Constraints [5|7]

Custom Constraints

- Definition einer Annotation sowie zugehörige Validatoren
- Reservierte Felder: message, groups, payload

```
@Constraint(validatedBy = { ZipCodeValidator.class }) ← Validator Class
@Target(ElementType.TYPE) ← Object Constraint
@Retention(RetentionPolicy.RUNTIME)
public @interface ValidZipCode {

    String message() default "{error.zip}";

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

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

}
```

```
@ValidZipCode
public class Address {
    // ...
}
```

Constraints [6|7]

Custom Constraints

```
public class ZipCodeValidator implements ConstraintValidator<ValidZipCode, Address> {

    @Override
    public void initialize(ValidZipCode ann) {
        // initialization
    }

    @Override
    public boolean isValid(Address address,
        ConstraintValidatorContext context) {
        // compare zip code and city
        return false;
    }

}
```

Constraints [7|7]

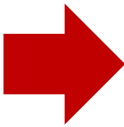
Composite Constraints

- Constraints auf Basis bestehender Constraints

```
@NotNull
@Pattern(regexp = "[A-Z].*")
@ReportAsSingleViolation
@Constraint(validatedBy = {})
@Target({ ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface CityName {

    String message() default "{invalid.cityname}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};

}
```



```
@CityName
private String city;
```

Fehlermeldungen

- Attribut message aller Constraints
- Ersetzung von {...}
 - Name eines Attributs des Constraints

```
@Size(min=3, max=5, message="Wert muss zwischen {min} und {max} Zeichen lang sein.")  
private String zip;
```

- Key aus ResourceBundle ValidationMessages

```
@ValidZipCode(message="{zip.code}")  
public class Address {  
    // ...  
}
```

Integration mit anderen Java-EE-Technologien

Bean Validation & JPA

- Automatische Validierung beim Datenbankzugriff
 - DDL-Schema-Generation (z.B. `@Column(nullable=false)` → `@NotNull`)
- Bisher nicht spezifiziert, aber angeregt

Bean Validation & Servlets

- Validator ist im Servlet verfügbar (`@Inject`)
- Kein Automatismus

Bean Validation & JSF

- Automatische Validierung aller Eingabekomponenten zugeordneten Feldern von JSF Managed Beans in Phase *Process Validation*