**Валидация данных при помощи Bean Validation API**

Для валидации данных в Java EE существует [Bean Validation](https://beanvalidation.org/" \t "_blank). Первая версия данного набора API была специфицирована в [JSR-303](https://www.jcp.org/en/jsr/detail?id=303) и опубликована как часть Java EE 6. Текущая версия — 2.0, является частью Java EE 8 и описана в [JSR-380](https://www.jcp.org/en/jsr/detail?id=380). Эталонной реализацией Bean Validation является [Hibernate Validator](http://hibernate.org/validator/" \t "_blank). Bean Validation может использоваться не только в классических приложениях на основе Java EE, но и в приложениях на основе Spring, и даже в приложениях, не имеющих отношения к Java EE.

**Зависимости**

Для использования API Bean Validation потребуется зависимость **validation-api** из **javax.validation**:

pom.xml

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>javax.validation</groupId>      <artifactId>validation-api</artifactId>      <version>2.0.1.Final</version>  </dependency> |

Если вы разрабатываете проект на чистом Java EE, то каких-либо других зависимостей вам не понадобится, более того можно использовать соответствующую версию **javaee-api**, так как API Bean Validation включены в неё.

В противном случае вам понадобятся ещё две зависимости: реализация Bean Validation и EL (Expression Language, язык выражений). В качестве реализации Bean Validation логичнее всего использовать эталонную реализацию — Hibernate Validator, а в качестве реализации EL можно использовать реализацию от Glassfish:

pom.xml

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <dependency>      <groupId>org.hibernate.validator</groupId>      <artifactId>hibernate-validator</artifactId>      <version>6.0.10.Final</version>  </dependency>  <dependency>      <groupId>org.glassfish</groupId>      <artifactId>javax.el</artifactId>      <version>3.0.1-b10</version>  </dependency> |

Теперь можно приступать к изучению ограничений/правил.

**Получение валидатора**

Самый простой способ получения валидатора — создать фабрику валидаторов по умолчанию и запросить у неё валидатор.

ValidationFactoryExample.java

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class ValidationFactoryExample {        public static void main(String[] args) {          ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();          Validator validator = validatorFactory.getValidator();          // ...      }  } |

В дальнейшем полученный объект типа **Validator** можно использовать для валидации объектов, аргументов, передаваемых в методы и значений, возвращаемых методами.

**Ограничения**

Правила валидации в Bean Validation задаются при помощи ограничений (constraints), аннотаций, расположенных в пакете **javax.validation.constraints**. Ограничения могут применяться к свойствам классов, аргументам методов и конструкторов, их возвращаемым значениям, а так же к типам обобщений.

Стандартный набор ограничений включает в себя наиболее часто используемые и универсальные. Кроме того, Bean Validation позволяет разработчикам добавлять собственные ограничения.

**Числовые ограничения**

**@DecimalMax**

Применима к переменным типов **BigDecimal**, **BigInteger**, **CharSequence**, **byte**, **short**, **int**, **long**и их классов-обёрток. Значение должно быть меньше, либо равно указанному значению, либо быть **null**для непримитивов.

**@DecimalMin**

Аналогична @DecimalMax, значение переменной должно быть числом и быть больше, либо равной указанной значению, либо быть null для непримитивов.

**@Digits**

Количество символов слева от запятой должно быть меньше, либо равным **integer**, а справа — меньше, либо равным **fraction,** **null**является валидным значением. Применима к **BigDecimal**, **BigInteger**, **CharSequence**, **byte**, **short**, **int**, **long**и их классам-обёртам.

**@Max**

Значение должно быть меньше, либо равно указанному значению, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**@Min**

Значение должно быть больше, либо равно указанному значению, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**@Negative**

Значение должно быть отрицательным, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**@NegativeOrZero**

Значение должно быть отрицательным, равняться 0, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**@Positive**

Значение должно быть положительным, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**@PositiveOrZero**

Значение должно быть отрицательным, равняться 0, либо быть **null**. Применима к переменным типов **BigDecimal**, **BigInteger**, **byte**, **short**, **int**, **long**и их классов-обёрток.

**Ограничения даты и времени**

**@Future**

Значение переменной должно быть будущим временем. Применима к **Date**, **Calendar**и многим типам из пакета **java.time**.

**@FutureOrPresent**

Значение переменной должно быть будущим либо настоящим временем. Применима к **Date**, **Calendar**и многим типам из пакета **java.time**.

**@Past**

Значение переменной должно быть прошедшим временем. Применима к **Date**, **Calendar**и многим типам из пакета **java.time**.

**@PastOrPresent**

Значение переменной должно быть прошедшим либо настоящим временем. Применима к **Date**, **Calendar**и многим типам из пакета **java.time**.

**Строчные ограничения**

**@Email**

Значение должно быть адресом электронной почты; применима к **CharSequence**. Поведение зависит от конкретной реализации.

**@NotBlank**

Значение типа **CharSequence** не должно быть **null**, пустым или состоять из одних лишь пробельных символов.

**@Pattern**

Значение типа **CharSequence** должно соответствовать указанному регулярному выражению.

**Булевые ограничения**

**@AssertFalse**

Аннотация применима к переменным типов **boolean**и **Boolean**, значение которых должно быть **false**, либо **null**.

**@AssertTrue**

Противоположность @AssertFalse, значение должно быть **true** или **null**.

Универсальные

**@NotEmpty**

Значение типов **CharSequence**, **Collection**, **Map**или массив не должно быть **null**и должно содержать хотя бы 1 элемент.

**@NotNull**

Значение не должно быть **null**.

**@Null**

Значение должно быть **null**.

**@Size**

Размер значения типов **CharSequence, Collection, Map**или массива должен быть в указанном диапазоне — между **min**и **max.**

**Валидация свойств**

Для валидации свойств классов достаточно одну или несколько из вышеперечисленных аннотаций применить к нужному свойству, например:

Event.java

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Event {    *@NotNull*  *@PastOrPresent*      private Date dateCreated;  } |

При валидации объекта типа Event будет произведена проверка, что свойство dateCreated содержит дату, значение которой — текущее или прошедшее время.

Пример валидации:

ValidationFactoryExample.java

Default



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class ValidationFactoryExample {        public static void main(String[] args) {          ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();          Validator validator = validatorFactory.getValidator();            Set<ConstraintViolation<Event>> violations = validator.validate(new Event());      }  } |

Метод validate валидатора вернёт непустой список нарушений (ConstraintViolation), если свойства объекта нарушают правила валидации.

**Валидация аргументов и возвращаемых значений методов**

Для валидации аргументов и возвращаемых значений методов (в том числе и конструкторов) можно использовать **ExecutableValidator**, введённый в Bean Validation 1.1.

Допустим у нас есть класс **EventHandler**:

EventHandler.java

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class EventHandler {        public EventHandler(*@NotNull* EventRepository repository) {          // ..      }    *@NotNull*      public Optional<*@NotBlank* String> handle(*@NotNull* *@Valid* Event event) {          // ..      }  } |

Валидация аргументов конструктора и метода handle выполняется следующим образом:

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // constructor args validation  Constructor<EventHandler> constructor = EventHandler.class.getConstructor(String.class);  Set<ConstraintViolation<EventHandler>> constructorViolations = executableValidator          .validateConstructorParameters(constructor, new Object[]{"name"});    EventHandler eventHandler = new EventHandler(repository);  Event event = new Event();    // method args validation  Method method = eventHandler.getClass()          .getMethod("handle", Event.class);    Set<ConstraintViolation<EventHandler>> argsViolations = executableValidator          .validateParameters(eventHandler, method, new Object[]{event});    // returned value validation  String id = eventHandler.handle(event);    Set<ConstraintViolation<EventHandler>> returnViolations = executableValidator          .validateReturnValue(eventHandler, method, id); |

**Полезные ссылки**

* [Документация Bean Validation](https://beanvalidation.org/2.0/spec/)
* [Документация Hibernate Validator](https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/)