Hibernate Validator 校验框架

一、 为什么使用校验框架

参数验证是一个常见的问题，例如验证用户输入的密码是否为空、邮箱是否合法等。但是无论是前端还是后台，都需对用户输入进行验证，以此来保证系统数据的正确性。对于 web 来说，有些人可能理所当然的想在前端验证就行了，但这样是非常错误的做法，前台的验证一般是通过 JavaScript，js 代码是可以被禁用和篡改的，所以相对后台检验而言，安全性会低一些。前端代码对于用户来说是透明的，稍微有点技术的人就可以绕过这个验证，直接提交数据到后台。无论是前端网页提交的接口，还是提供给外部的接口，参数验证随处可见，也是必不可少的。总之，一切用户的输入都是不可信的。

基于这样的常识，在后端的代码开发中，参数校验是一个永远也绕不开的话题。然而编写参数校验代码的过程是一个技术含量不高，及其耗时，繁琐的过程。比如极大多数的参数校验代码就是：判断一下用户名是否已经存在、用户名长度是否满足要求、用户填写的邮箱是否合法。年龄参数是不是一个整数等等。为了减少重复代码的开发，许多有技术积淀的公司，一般都会提供一套或多套特有的参数校验工具类。以此减少代码量。这种做法在项目中非常普遍，有了这些工具类库，程序员在编写业务代码的过程中，工作效率可以大大提升。

二、 校验框架 Hibernate Validator

1. 简单介绍

Hibernate Validator，The Bean Validation reference implementation. Express validation rules in a standardized way using annotation-based constraints and benefit from transparent integration with a wide variety of frameworks. 反正就是通过注解进行校验

2. 配置环境

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.2.Final</version>
</dependency>
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>javax.el</artifactId>
    <version>2.2.6</version>
</dependency>
```

3. 基本注解了解

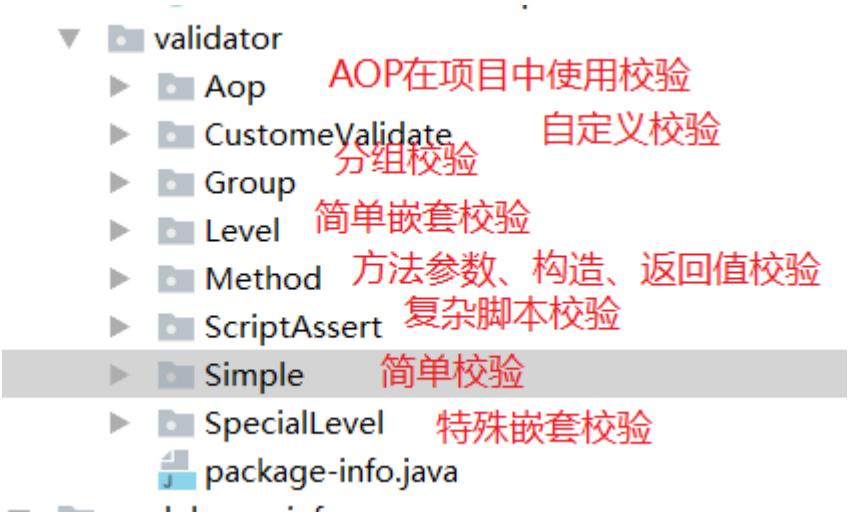
```
@Null 被注释的元素必须为 null
@NotNull 被注释的元素必须不为 null
@AssertTrue 被注释的元素必须为 true
@AssertFalse 被注释的元素必须为 false
@Min(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@Max(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@DecimalMin(value) 被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@DecimalMax(value) 被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@Size(max, min) 被注释的元素的大小必须在指定的范围内
@Digits (integer, fraction) 被注释的元素必须是一个数字，其值必须在可接受的范围内
@Past 被注释的元素必须是一个过去的日期
@Future 被注释的元素必须是一个将来的日期
@Pattern(value) 被注释的元素必须符合指定的正则表达式
表 2. Hibernate Validator 附加的 constraint
@Email 被注释的元素必须是电子邮箱地址
@Length 被注释的字符串的大小必须在指定的范围内
@NotEmpty 被注释的字符串的必须非空
@Range 被注释的元素必须在合适的范围内
```

[校验框架](https://blog.csdn.net/liuchuanhong1/article/details/52042294)
[参考文档](http://jinnianshilongnian.iteye.com/blog/1990081)
[参考文档](http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#preface)

三、 Hibernate Validator 实践

注意：有些注解必须是当前属性非空的情况下才起作用，一定要定义非空校验注解哦！

包层次结构



打印错误信息

```
public static void pringValidateStr(Set<ConstraintViolation<Object>> set2) {
    for (ConstraintViolation<Object> constraintViolation : set2){
        log.info("错误：" + constraintViolation.getMessage());
        log.info("字段："+constraintViolation.getPropertyPath().toString());

    }
}
```

1. SimpleValidator 简单校验一个 Java 类中的字段

```java
@Slf4j
public class SimpleValidator {

    @NotNull(message = "id不能为空!")
    @Min(value = 1, message = "Id只能大于等于1，小于等于10")
    @Max(value = 10, message = "Id只能大于等于1，小于等于10")
    private Integer id;

    @NotNull(message = "姓名不能为空!")
    @Size(min = 2, max = 6, message = "姓名长度必须在{min}和{max}之间")
    @Pattern(regexp = "[\u4e00-\u9fa5]+", message = "名称只能输入是中文字符")
    private String userName;

    @NotBlank(message = "不能为空字符串")
    @Size(min = 6, max = 12, message = "密码长度必须在{min}和{max}之间")
    private String passWord;


    @NotNull(message = "邮件不能为空!")
    @Email(message = "邮件格式不正确")
    private String email;

    public Integer getId() { return id; }

    public SimpleValidator setId(Integer id) {
        this.id = id;
        return this;
    }
```

```java
/**
 * 简单校验测试类（快速失败模式-一旦发现错误就返回）
 *
 * @author: wangji
 * @date: 2018/07/11 10:01
 */
@Slf4j
public class SimpleTest {

    private   Validator validator;

    private SimpleValidator simpleValidator;

    @Before
    public void before(){
        //region 设置校验工厂
        ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.
                .configure()
                .failFast( false ) //快速失败模式
                .buildValidatorFactory();
         this.validator = validatorFactory.getValidator();
        //endregion
        //region 设置校验实体默认值
         this.simpleValidator = new SimpleValidator();
        simpleValidator.setEmail("983433479@qq.com")
                .setId(5)
                .setPassWord("password")
                .setUserName("汪吉");
        //endregion
    }


    @Test
    public  void testMin() {
        this.simpleValidator.setId(-1);
        pringValidateStr(validator.validate(this.simpleValidator));
    }

    @Test
    public  void testMax() {
        this.simpleValidator.setId(20);
        pringValidateStr(validator.validate(this.simpleValidator));
    }
```

测试错误

```
46
47      @Test
48      public  void testMin() {
49          this.simpleValidator.setId(-1);
50          pringValidateStr(validator.validate(this.simpleValidator));
51      }
52
```

1 test passed – 382ms

```
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 15:34:03,969  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 15:34:04,302  INFO [SimpleTest.java:119] : 错误：Id只能大于等于1，小于等于10
2018-07-11 15:34:04,302  INFO [SimpleTest.java:120] : 字段：id

Process finished with exit code 0
```

2. 针对某个 Class 的属性校验

和基本的处理校验一样的，当前类和 Simple Validator 在一个测试类中

```
/**
 * 针对某个Object 进行校验
 */
@Test
public  void testValidate() {
    this.simpleValidator.setPassWord(null);
    pringValidateStr(validator.validate(this.simpleValidator));
}

/**
 * 针对某个class上的字段进行校验
 */
@Test
public  void testValidateValue() {
    Set<ConstraintViolation<SimpleValidator>> constraintViolations = validator.validateValue(SimpleValidator.class, propertyName: "userName", value: "中");
    pringValidateStrSimpleValidator(constraintViolations);
}

/**
 * 针对某个实例的属性进行校验
 */
@Test
public  void testValidateProperty() {
    this.simpleValidator.setUserName("中");
    Set<ConstraintViolation<SimpleValidator>> constraintViolations = validator.validateProperty(simpleValidator, propertyName: "userName");;
    pringValidateStrSimpleValidator(constraintViolations);
}
```

3. 嵌套校验

测试类，嵌套校验 List<Address> 、Address、List<String>

```
public class SimpleLevelValidator {

    @Valid
    @NotNull(message = "地址参数不能为空")
    private Address1 address1;


    /**
     * @NotEmpty Asserts that the annotated string, collection, map or array is not {@code null} or empty.
     */
    @NotEmpty(message = "参数List数据不能为空")
    private List<String> messages;


    @Valid
    @NotEmpty(message = "参数List数据不能为空")
    private List<Address1> address1s;
```

嵌套信息类 Address1

```
*/
public class Address1 {

    @NotBlank(message = "addressName 不能为空")
    private String  addressName;

    public String getAddressName() { return addressName; }

    public Address1 setAddressName(String addressName) {
        this.addressName = addressName;
        return this;
    }

    @Override
    public String toString() {
        return new ToStringBuilder( object: this)
                .append("addressName", addressName)
                .toString();
    }
}
```

测试校验

```java
public class SimpleLevelValidatorTest {

    private Validator validator;

    private SimpleLevelValidator simpleLevelValidator;
    @Before
    public void before(){
        //region 设置校验工厂
        ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )
                .configure()
                .failFast( false ) //快速失败模式
                .buildValidatorFactory();
        this.validator = validatorFactory.getValidator();
        //endregion
        //region 设置校验实体默认值
        this.simpleLevelValidator = new SimpleLevelValidator();
        Address1 address1 = new Address1();
        address1.setAddressName("杭州");
        this.simpleLevelValidator.setAddress1(address1)
                .setMessages(Lists.newArrayList("生活不止眼前和苟且","还有诗和远方"))
                .setAddress1s(Lists.newArrayList(address1));
        //endregion
    }

    @Test
    public  void testLevelValid() {
        //region 第一层校验
        this.simpleLevelValidator.setAddress1(null);
        pringValidateStr(validator.validate(this.simpleLevelValidator));
        //endregion

        //region 第二层校验 Address1包装校验
        Address1 address1 = new Address1();
        address1.setAddressName("");
        this.simpleLevelValidator.setAddress1(address1);
        pringValidateStr(validator.validate(this.simpleLevelValidator));
        //endregion

        //region 第二层校验 List<Address1> 层级校验
        Address1 address2 = new Address1();
        address1.setAddressName("");
        this.simpleLevelValidator.setAddress1s(Lists.newArrayList(address2));
        pringValidateStr(validator.validate(this.simpleLevelValidator));
        //endregion
    }
```

测试结果：

```java
@Test
public  void testLevelValid() {
    //region 第一层校验
    this.simpleLevelValidator.setAddress1(null);
    pringValidateStr(validator.validate(this.simpleLevelValidator));
    //endregion

    //region 第二层校验 Address1包装校验
    Address1 address1 = new Address1();
    address1.setAddressName("");
    this.simpleLevelValidator.setAddress1(address1);
    pringValidateStr(validator.validate(this.simpleLevelValidator));
    //endregion

    //region 第二层校验 List<Address1> 层级校验
    Address1 address2 = new Address1();
    address1.setAddressName("");
    this.simpleLevelValidator.setAddress1s(Lists.newArrayList(address2));
    pringValidateStr(validator.validate(this.simpleLevelValidator));
    //endregion
}
```

1 test passed – 312ms

```
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 15:40:09,440  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 15:40:09,695  INFO [SimpleLevelValidatorTest.java:86] : 错误: 地址参数不能为空
2018-07-11 15:40:09,696  INFO [SimpleLevelValidatorTest.java:87] : 字段: address1
2018-07-11 15:40:09,699  INFO [SimpleLevelValidatorTest.java:86] : 错误: addressName 不能为空
2018-07-11 15:40:09,702  INFO [SimpleLevelValidatorTest.java:87] : 字段: address1.addressName
2018-07-11 15:40:09,712  INFO [SimpleLevelValidatorTest.java:86] : 错误: addressName 不能为空
2018-07-11 15:40:09,712  INFO [SimpleLevelValidatorTest.java:87] : 字段: address1s[0].addressName
2018-07-11 15:40:09,712  INFO [SimpleLevelValidatorTest.java:86] : 错误: addressName 不能为空
2018-07-11 15:40:09,713  INFO [SimpleLevelValidatorTest.java:87] : 字段: address1.addressName
```

4. 自定义校验

自定义校验处理的注解：字符串长度校验

```java
 */
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE, ElementType.PARAMETER,ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Constraint(validatedBy = CharLengthCustomeValidate.class)//处理的类   校验处理类，可以查看Hibernate-Validator源码 进行模仿
public @interface CharLength {
    String message() default "";

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

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

    int length() default 64;
}
```

校验处理实现类：

```java
*/
public class CharLengthCustomeValidate implements ConstraintValidator<CharLength, String> {

    private int length = 0;

    @Override
    public void initialize(CharLength constraintAnnotation) { this.length = constraintAnnotation.length(); }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context) {
        boolean result = true;

        // 非空的情况下进行字符长度的校验、满足某些特定的可选择的需求
        if (StringUtils.isNotEmpty(value) && ValidatorUtils.getCharLength(value.trim()) > length) {
            result = false;
        }
        return result;
    }
}
```

使用：和基本的校验注解一样的，放在字段上即可；

5. 特殊嵌套校验

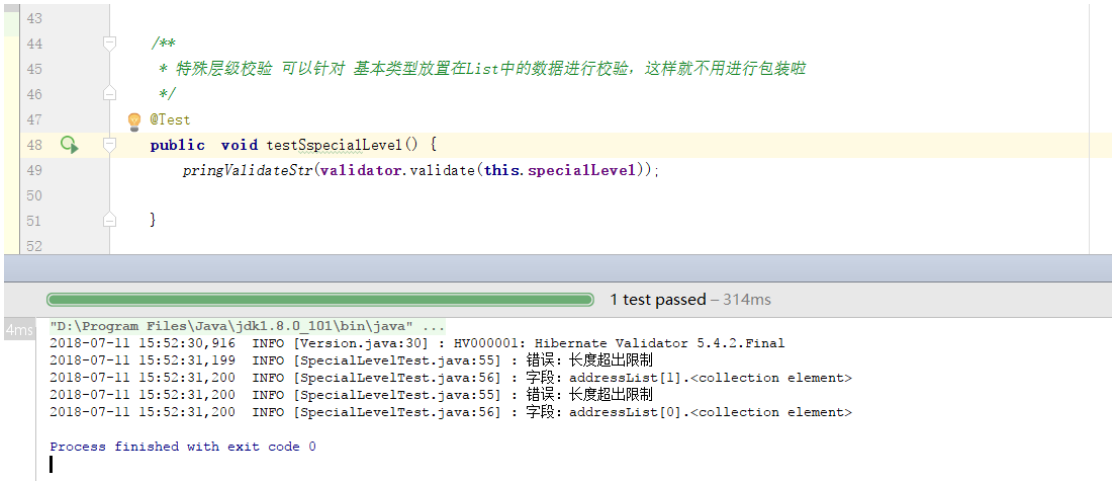
为何特殊啦 List<String> ，如果我想要校验 String 泛型参数的数据，如何处理？@NotEmpty 只能查看是否为空？根据 Hibernate 官方文档 Java1.8 以上版本支持的注解 ElementType.TYPE_USE 支持放在泛型的内部，这样就可以进行校验啦！之前自定义校验的时候，添加了这个注解在长度校验中，所以可以使用。

```java
/**
 * 特殊层级校验
 * <P>Hibernate Validator官方文档上说需要自定义校验且Java1.8以上版本支持的注解 ElementType.TYPE_USE，参考自定义校验</P>
 * <P>改变， ElementType.TYPE_USE支持放在泛型的内部，这样也是可以通过使用校验</P>

    @NotNull
    @Valid //深度校验
    private List<@CharLength String> addressList;
 *
 * @author: wangji
 * @date: 2018/07/11 11:07
 */
public class SpecialLevel {

    @Valid
    @NotEmpty(message = "数据不能为空")
    private List<@CharLength(message = "长度超出限制",length = 2) String> addressList;

    public List<String> getAddressList() { return addressList; }

    public SpecialLevel setAddressList(List<String> addressList) {
        this.addressList = addressList;
        return this;
    }

    @Override
    public String toString() {
        return new ToStringBuilder( object: this)
                .append("addressList", addressList)
                .toString();
    }
}
```

测试：

```java
@f4j
public class SpecialLevelTest {

    private Validator validator;

    private SpecialLevel specialLevel;
    @Before
    public void before(){
        //region 设置校验工厂
        ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )
                .configure()
                .failFast( false ) //快速失败模式
                .buildValidatorFactory();
        this.validator = validatorFactory.getValidator();
        //endregion
        //region 设置校验实体默认值
        this.specialLevel = new SpecialLevel();
        specialLevel.setAddressList(Lists.newArrayList("生活不止眼前和苟且","还有诗和远方","哦"));
        //endregion
    }


    /**
     * 特殊层级校验 可以针对 基本类型放置在List中的数据进行校验，这样就不用进行包装啦
     */
    @Test
    public  void testSspecialLevel() {
        pringValidateStr(validator.validate(this.specialLevel));

    }
```

测试结果：



6. 复杂的校验

比较复杂的校验需要根据当前字段的值进行处理，这个时候比较复杂，可以通过脚本处理进行简化；

```java
@ScriptAssert(lang = "javascript", script = "_this.checkParams(_this.first,_this.second)", message = "校验不通过，第一个比第二个大ya")
@Slf4j
public class ScriptAssertValidator {

    @NotNull
    private Integer first;

    @NotNull
    private Integer second;


    public boolean checkParams(Integer first, Integer second) {
        if (first != null && second != null) {

            if (first.compareTo(second) > 0) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        Validator validator = validatorFactory.getValidator();
        ScriptAssertValidator groupValidator = new ScriptAssertValidator();
        groupValidator.setFirst(15);
        groupValidator.setSecond(10);
        pringValidateStr(validator.validate(groupValidator));

    }
```

_this 当前对象，其实就是调用当前的方法，通过Js脚本进行执行

测试结果：

```
ScriptAssertValidator
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 15:56:35,007  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 15:56:35,820  INFO [ScriptAssertValidator.java:59] : 错误：校验不通过，第一个比第二个大ya
2018-07-11 15:56:35,821  INFO [ScriptAssertValidator.java:60] : 字段:
```

多个校验脚本处理效果一样@ScriptAssert.List

```java
/**
 * List脚本
 *
 * @author: wangji
 * @date: 2018/07/11 13:57
 */
@ScriptAssert.List(value = {
        @ScriptAssert(lang = "javascript", script = "_this.checkParams(_this.first,_this.second)", message = "校验不通过，第一个比第二个大"),
        @ScriptAssert(lang = "javascript", script = "_this.bigThan(_this.first)", message = "第一个要大于20")
})
@Slf4j
public class ScriptAssertValidatorList {
    @NotNull
    private Integer first;

    @NotNull
    private Integer second;
```

7. 分组校验

何为分组校验，分组校验的意思就是，比如我第一次插入数据不需要校验主键，更新的时候需要校验主键，这种不同情况下校验的选择不同，所以需要分组校验；默认情况下分组为Default.class;

分组标识为自己随便定义一个接口就好啦，如下：

```java
public interface First {
}
```

分组校验 :由于默认情况下就使用 Default.class,所以如果不需要分组, 分组信息可以不填的！这里只有 NotNull 注解未分组， 所以就被默认分组包含；

```java
@Slf4j
public class GroupValidator {

    @NotNull(message = "姓名不能为空!")
    @Size(min = 2, max = 4, message = "姓名长度必须在{min}和{max}之间", groups = {First.class})
    @Pattern(regexp = "[\u4e00-\u9fa5]+", message = "名称只能输入是中文字符", groups = {First.class})
    private String userName;

    @NotNull(message = "密码不能为空!", groups = {Second.class})
    @Size(min = 6, max = 12, message = "密码长度必须在{min}和{max}之间", groups = {Second.class})
    private String passWord;

    public static void main(String[] args) {
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        Validator validator = validatorFactory.getValidator();
        GroupValidator groupValidator = new GroupValidator();
        pringValidateStr(validator.validate(groupValidator, Default.class));

        groupValidator.setUserName("ww");
        pringValidateStr(validator.validate(groupValidator, First.class));

        groupValidator.setPassWord("ww");
        pringValidateStr(validator.validate(groupValidator, Second.class));
    }
```

测试结果：

```
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 15:59:01,601  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 15:59:01,801  INFO [GroupValidator.java:53] : 错误: 姓名不能为空!
2018-07-11 15:59:01,802  INFO [GroupValidator.java:54] : 字段: userName
2018-07-11 15:59:01,806  INFO [GroupValidator.java:53] : 错误: 名称只能输入是中文字符
2018-07-11 15:59:01,806  INFO [GroupValidator.java:54] : 字段: userName
2018-07-11 15:59:01,811  INFO [GroupValidator.java:53] : 错误: 密码长度必须在6和12之间
2018-07-11 15:59:01,811  INFO [GroupValidator.java:54] : 字段: passWord

Process finished with exit code 0
```

8. 默认分组顺序校验

默认的分组支持顺序校验， 当第一个不成功， 就不继续校验其它的啦！项目中使用较少；

```java
/**
 * 校验分组校验顺序(represents the {@link Default} group for that class.),最后定义的class一定是当前类
 * <p> 通过@GroupSequence指定验证顺序,先验证First分组，如果有错误立即返回
 * 而不会验证Second分组，接着如果First分组验证通过了， 那么才去验证Second分组，最后指定GroupSequenceValidator.class（当前类）
 * 表示那些没有分组的在最后
 * </p>
 *
 * @author: wangji
 * @date: 2018/07/11 11:29
 */
@GroupSequence({First.class, Second.class, GroupSequenceValidator.class})
@Slf4j
public class GroupSequenceValidator {

    @NotNull(message = "姓名不能为空!")
    @Size(min = 2, max = 4, message = "姓名长度必须在{min}和{max}之间", groups = {First.class})
    @Pattern(regexp = "[\u4e00-\u9fa5]+", message = "名称只能输入是中文字符", groups = {First.class})
    private String userName;

    @NotNull(message = "密码不能为空!", groups = {Second.class})
    @Size(min = 6, max = 12, message = "密码长度必须在{min}和{max}之间", groups = {Second.class})
    private String passWord;

    public static void main(String[] args) {
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        Validator validator = validatorFactory.getValidator();
        GroupSequenceValidator groupValidator = new GroupSequenceValidator();
        groupValidator.setUserName("ww");
        groupValidator.setPassWord("ww");
        pringValidateStr(validator.validate(groupValidator, Default.class));

    }
```

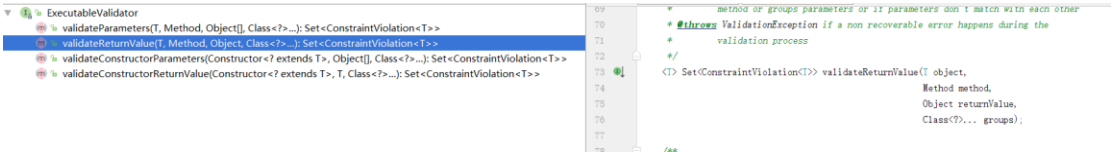
测试结果：

```
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 16:05:51,502  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 16:05:51,807  INFO [GroupSequenceValidator.java:50] : 错误: 密码长度必须在6和12之间
2018-07-11 16:05:51,808  INFO [GroupSequenceValidator.java:51] : 字段: passWord
```

9. 校验方法参数、构造函数、方法返回值

校验这些数据是通过 AOP 处理 Hibernate Validator 的基础，通过这些方法能够更好的理解通过 AOP 处理实现的细节；上面的部分基本上处理的数据都是在基本的 Object 数据实例的校验；

ExecutableValidator：Validates parameters and return values of methods and constructors.



需要校验的方法：

```java
/**
 * 校验构造函数、函数参数、返回值等，这里是处理AOP的基础
 * 【参考文档】(https://diamondfsd.com/article/78fa12cd-b530-4a90-b438-13d5a0c4e26c/)
 *
 * @author: wangji
 * @date: 2018/07/11 14:23
 */
@Slf4j
public class MethodValidator {

    public MethodValidator() {
    }

    /**
     * 校验构造函数
     *
     * @param message
     */
    public MethodValidator(@NotNull(message = "构造不能为空") String message) { log.info("Constructor通过校验"); }

    /**
     * 校验返回值
     *
     * @return
     */
    public
    @NotBlank(message = "不能为空的字符串")
    String validateReturn() { return ""; }


    public void testLevel1(Address1 address1, @NotBlank String name) {

    }
```

从上到下的处理方法参数、返回值、构造函数

```java
    public static void main(String[] args) throws Exception {
        ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
        Validator validator = factory.getValidator();
        /**
         * 获取校验方法参数的校验器
         */
        ExecutableValidator validatorMethodParam = validator.forExecutables();  得到处理方法参数的校验实现类
        MethodValidator methodValidator = new MethodValidator();

        //region 校验方法参数只能校验方法上的注解哦，不包含Address内部的注解
        Method method = methodValidator.getClass().getMethod( name: "testLevel", Address1.class, String.class);
        Address1 address1 = new Address1();
        Object[] paramObjects = new Object[]{address1, ""};
        pringValidateStr(validatorMethodParam.validateParameters(methodValidator, method, paramObjects));
        //endregion


        //region 校验方法的返回值，同样也是不包含内部注解
        Method validateReturnMethiod = methodValidator.getClass().getMethod( name: "validateReturn");
        Object returnValue = ReflectionUtils.invokeMethod(validateReturnMethiod, methodValidator);
        pringValidateStr(validatorMethodParam.validateReturnValue(methodValidator, validateReturnMethiod, returnValue));
        //endregion


        Constructor<?> constructor = MethodValidator.class.getConstructor(String.class);
        Object[] constructorsParams = new Object[]{null};
        pringValidateStr(validatorMethodParam.validateConstructorParameters(constructor, constructorsParams));

    }                                                                    可以添加分组哦，不适用就是默认的
```

测试结果：

```
"D:\Program Files\Java\jdk1.8.0_101\bin\java" ...
2018-07-11 16:20:06,661  INFO [Version.java:30] : HV000001: Hibernate Validator 5.4.2.Final
2018-07-11 16:20:06,990  INFO [MethodValidator.java:89] : 错误：不能为空
2018-07-11 16:20:06,991  INFO [MethodValidator.java:90] : 字段：testLevel.arg1
2018-07-11 16:20:07,002  INFO [MethodValidator.java:89] : 错误：不能为空的字符串
2018-07-11 16:20:07,002  INFO [MethodValidator.java:90] : 字段：validateReturn.<return value>
2018-07-11 16:20:07,005  INFO [MethodValidator.java:89] : 错误：构造不能为空
2018-07-11 16:20:07,006  INFO [MethodValidator.java:90] : 字段：MethodValidator.arg0
```

四、 AOP 实现思路

1. 基本思路梳理

我们通过注解能够获取到当前调用方法的参数值，在执行方法调用之前进行参数校验，然后通过 Hibernate Validator 对于方法参数校验进行处理，这里处理应该保护当前的校验的分组信息获取（通过注解获取到哦）、对于方法校验还需变量所以的方法参数进行 Class 中注解信息校验，能够处理好这些数据信息，然后通过抛出错误的异常信息，全局异常进行捕获处理即可完成；

2. 实践实现 AOP 校验

a) 定义校验的注解（包含一个数据分组信息）

```java
@Target({ElementType.METHOD, ElementType.TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface EgValidate {
    Class<?> [] groups() default {Default.class};
}
```

b) 定义 AOP 拦截特定的注解

放在类上面拦截所有的方法、放在方法上只拦截当前方法

```java
@Component
@Aspect
public class EgMethodValidationAspect {

    /**
     * 获取校验的实体的信息
     */
    private static final Validator VALIDATOR = Validation.byProvider(HibernateValidator.class)
            .configure()
            .failFast(true)
            //快速失败模式开启，当检测到有一项失败立即停止
            .buildValidatorFactory().getValidator();

    /**
     * 注解在方法类上或者方法上有效
     */
    @Pointcut("@within(com.hikvision.energy.validator.EgValidate)||@annotation(com.hikvision.energy.validator.EgValidate) ")
    public void pointcut() {
    }
```

c) 在方法执行之前进行拦截处理

这个阶段是重点处理阶段，需要获取方法参数、当前执行方法、执行方法注解、方法参数注解、方法参数等等；还需要获取当前类中是否定义类级别的分组信息、方法上分组信息，如果方法上有分组信息就使用方法上的，按照优先级使用分组信息；

```java
     */
    @Before("pointcut()")
    public void before(JoinPoint point) throws NoSuchMethodException, SecurityException {

        //  获得切入目标对象
        Object target = point.getThis();
        // 获得切入方法参数
        Object[] args = point.getArgs();
        // 获得切入的方法
        Method method = ((MethodSignature) point.getSignature()).getMethod();
        Annotation[] classAnnotations = target.getClass().getAnnotations();   类注解
        Annotation[] methodAnnotations = method.getAnnotations();  方法注解
        Annotation[][] parameterAnnotations = method.getParameterAnnotations();   方法参数注解
        if (parameterAnnotations != null && parameterAnnotations.length > 0) {
            //如果方法参数有基本的注解，就进行Hibernate VALIDATOR 基本的参数校验
            validMethodParams(target, method, args);
        }                                   如果方法参数上有注解，就校验基本的方法参数，不对参数进行类级别校验
        // 判断参数中是否含有EgValidate注解，进行特殊分组，Bean级别的参数校验
        int i = 0;
        Set<Integer> idSet = Sets.newHashSet(3);
        //排查掉已经在参数中校验过的参数不适用类或者方法上的校验参数在次进行校验
        if (args != null && args.length > 0 && parameterAnnotations != null && parameterAnnotations.length > 0) {
            for (Object arg : args) {
                if (arg != null && parameterAnnotations[i] != null) {
                    for (Annotation parameterAnnotation : parameterAnnotations[i]) {
                        if (parameterAnnotation instanceof EgValidate) {方法参数上Class级别含有特殊的分组信息
                            if (!ClassUtils.isPrimitiveOrWrapper(arg.getClass())) {
                                validBeanParam(arg, ((EgValidate) parameterAnnotation).groups());
                                idSet.add(i);
                            }
                        }
                    }
                    i++;
                }
            }
        }
```

基本的方法参数校验处理完成啦，就开始处理 Class 级别的参数校验（非基本数据类型）,如果方法参数上有@EgValidator 可以获取到分组信息就立即进行 Class 级别数据校验，否则下面进行通过获取方法上注解、类上注解，以优先级为原则，选取默认的分组信息处理，最后进行类级别数据校验；

```
        }
        // 如果没有异常继续校验当前的每一个非基本类型的参数
        EgValidate egValidate = null;
        if (methodAnnotations != null) {
            //方法上是否有校验参数
            egValidate = AnnotationUtils.findAnnotation(method, EgValidate.class);
        }
        if (egValidate == null && classAnnotations != null) {
            // 类上是否含有
            egValidate = AnnotationUtils.findAnnotation(target.getClass(), EgValidate.class);
        }
        // 如果在类或者方法上加了验证注解 ， 则对所有非基本类型的参数对象进行验证,不管参数对象有没有加注解，使用方法上的分组
        if (egValidate != null && args != null && args.length > 0) {
            i = 0;
            for (Object arg : args) {
                if (arg != null && !ClassUtils.isPrimitiveOrWrapper(arg.getClass()) && !idSet.contains(i)) {
                    validBeanParam(arg, egValidate.groups());
                }
                i++;
            }
        }
    }
```

然后检测到有异常就抛出去就行啦！

```
    /**
     * @param obj    参数中的Bean类型参数
     * @param groups 分组信息
     * @desction: 进行参数中的Bean校验
     * @author: wangji
     * @date: 2018/3/13 10:10
     */
    private void validBeanParam(Object obj, Class<?>... groups) {
        Set<ConstraintViolation<Object>> validResult = VALIDATOR.validate(obj, groups);
        throwConstraintViolationException(validResult);
    }


    /**
     * @param obj    当前的实例
     * @param method 实例的方法
     * @param params 参数
     * @desction: 对于Hibernate 基本校验Bean放在参数中的情况的校验 【例如 User getUserInfoById(@NotNull(message = "不能为空") Integer id);】
     * @author: wangji
     * @date: 2018/3/13 10:11
     */
    private void validMethodParams(Object obj, Method method, Object[] params) {
        ExecutableValidator validatorParam = VALIDATOR.forExecutables();
        Set<ConstraintViolation<Object>> validResult = validatorParam.validateParameters(obj, method, params);
        throwConstraintViolationException(validResult);
    }

    /**
     * @param validResult
     * @desction: 判断校验的结果是否存在异常（存在异常，通过全局异常统一处理）
     * @author: wangji
     * @date: 2018/3/13 10:09
     */
    private void throwConstraintViolationException(Set<ConstraintViolation<Object>> validResult) {
        if (!validResult.isEmpty()) {
            throw new ConstraintViolationException(validResult);
        }
    }
}
```

3. 实践

a) 定义一个 Class

```
public class User implements Serializable {

    @NotBlank(message = "用户名不能为空",groups = {ValidationDP1.class})
    private String name;

    @NotNull(message="地址不能为空")
    private String address;
    .....
}
```

b) 不使用校验分组

```
/**
 * 不使用校验分组，使用默认的分组Default.class(Bean中没有编写校验分组的信息)
 *
 * @param user
 * @return
 * @throws Exception
 */
@ResponseBody
@RequestMapping(value = "/test0")
@EgValidate
public AjaxResponse testValidator0(User user) throws Exception {
    AjaxResponse response = new AjaxResponse();
    response.setResult(true);
    response.setResultMsg("校验通过");
    return response;
}
```

c) 使用参数上校验分组

```
/**
 * 使用参数上的分组校验
 *
 * @param user
 * @return
 * @throws Exception
 */
@ResponseBody
@RequestMapping(value = "/test2")
@EgValidate
public AjaxResponse testValidator2(@EgValidate(groups = {ValidationDP1.class, Default.class}) User user) throws Exception {
    AjaxResponse response = new AjaxResponse();
    response.setResult(true);
    response.setResultMsg("校验通过");
    return response;
}
```

d) 使用参数上分组，还有基本的注解（这种情况基本不常见，Service 层校验可能会有这种奇葩）

```
/**
 * 先校验基本的Hibernate validator 注解，通过后使用参数上的分组校验Bean
 *
 * @param user
 * @return
 * @throws Exception
 */
@ResponseBody
@RequestMapping(value = "/test3")
@EgValidate
public AjaxResponse testValidator3(@EgValidate(groups = {ValidationDP1.class, Default.class}) User user, @NotNull(message = "不能为空") Integer id) throws Exception {
    AjaxResponse response = new AjaxResponse();
    response.setResult(true);
    response.setResultMsg("校验通过");
    return response;
}
```

e) 正常情况下不会出现这么复杂，所以这里实现的时候没有把方法上基本类型的校验分组进行处理；一般情况下分组都是在一个类中使用，所以这里没有进行处理；处理的情况比较复杂实用性不是很强！

```
<T> Set<ConstraintViolation<T>> validateParameters(T object,
                                                   Method method,
                                                   Object[] parameterValues,
                                                   Class<?>... groups);
```

基本上正常情况下处理都是这样的，分组都没有定义！直接处理 ok！

```
    @RequestMapping(value = "/save/operationNote/{remoteGuidanceId}", method = RequestMethod.POST)
    @ResponseBody
    @EgValidate
    public Object addRemoteGuidance(@RequestBody @NotNull(message = WORK_PARAMETE_RERROR) OperationNotesParam param
            , @PathVariable Long remoteGuidanceId, AjaxResponse result) {
        remoteGuidanceService.saveOrUpdateOperationNote(param, remoteGuidanceId);
        result.setResultMsg(I18nUtil.getMessage(I18nConst.COMMON_SAVE_SUCCESS));
        return result.getData();
    }
}
```

五、 参考文档

[Hibernate-validator 校验架](https://blog.csdn.net/liuchuanhong1/article/details/52042294)
[记录 Hibernate Validator 实践](https://blog.csdn.net/u012881904/article/details/79538895)
[官方文档](http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#preface)
[学习记录文档](http://jinnianshilongnian.iteye.com/blog/1990081)