23 Spring Core参数校验之JSR-303_JSR-349注解

更新时间：2020-08-06 13:40:13

> 学习知识要善于思考，思考，再思考。 —— 爱因斯

背景

参数校验是我们程序开发中必不可少的过程。用户在前端要校验参数的合法性，当数据到了后端，为了防止恶意操作，保持程序的健壮性，后端同样需要对数据进行校验。当我们多个地方需要校验时，我们就需要在每一个地方调用校验程序,导致代码很冗余，且不美观。那么如何优雅的对参数进行校验呢？

实现 Validator 接口

Validator 接口校验是 Spring 的校验机制。可以用它来验证自己定义的实体对象。注意：这个接口是上下文无关的，也就是说不仅可以在 Web 层验证对象，也可以在 DAO 层，或者其它任意层。（参照上篇）

JSR 349/303/380 注解方式

Bean Validation 为 JavaBean 验证定义了相应的元数据模型和 API。缺省的元数据是 Java Annotations，通过使用 XML 可以对原有的元数据信息进行覆盖和扩展。在应用程序中，通过使用 Bean Validation 或是你自己定义的 constraint，例如 `@NotNull`，`@Max`，`@ZipCode`， 就可以确保数据模型（JavaBean）的正确性。constraint 可以附加到字段，getter 方法，类或者接口上面。对于一些特定的需求，用户可以很容易的开发定制化的 constraint。Bean Validation 是一个运行时的数据验证框架，在验证之后验证的错误信息会被马上返回。常见的注解有：

- **@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。

JSR 349/303/380 参数校验实例

本次我们以实现 JSR 349/303/380 注解方式为例进行验证，并了解其背后的原理。

要验证的 Java Bean，加上验证注解：

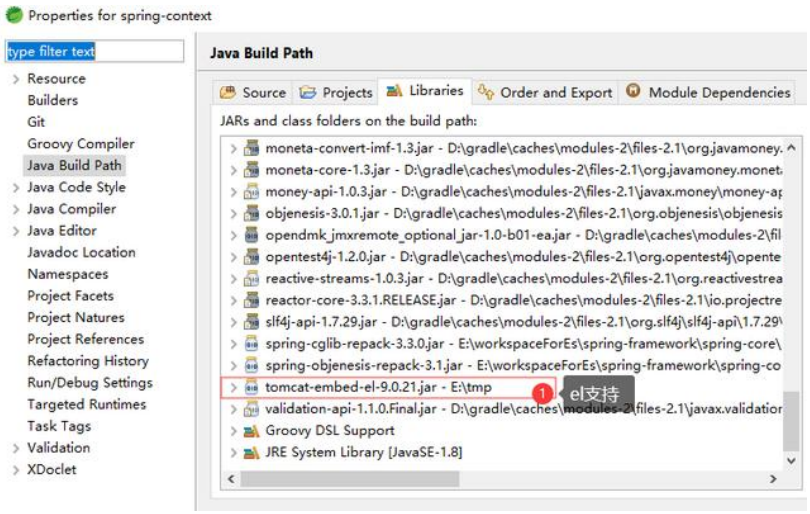
```
package com.davidwang456.test;

import javax.validation.constraints.DecimalMin;
import javax.validation.constraints.NotNull;
import java.math.BigDecimal;
import java.util.Date;

import lombok.Data;

@Data
public class Order {
      @NotNull(message = "日期不能为空")
      private Date date;

      @NotNull(message = "价格不能为空")
      @DecimalMin(value = "0", inclusive = false, message = "价格无效")
      private BigDecimal price;
}
```

要执行的验证逻辑：

```
package com.davidwang456.test;
import javax.validation.ConstraintViolation;
import javax.validation.Validator;

import org.springframework.beans.factory.annotation.Autowired;

import java.util.Locale;
import java.util.Set;

public class ClientBean {
  @Autowired
  private Order order;

  @Autowired
  Validator validator;

  public void processOrder () {
      if (validateOrder()) {
          System.out.println("processing " + order);
      }
  }

  private boolean validateOrder () {
      Locale.setDefault(Locale.US);
      Set<ConstraintViolation<Order>> c = validator.validate(order);
      if (c.size() > 0) {
          System.out.println("Order validation errors:");
          c.stream().map(v -> v.getMessage())
           .forEach(System.out::println);
          return false;
      }
      return true;
  }
}
```

Spring 的 Validator 工厂：

注意：ValidatorFactory 需要容器 javax.el 表达式的支持，如 Tomcat，Jetty，GlassFish 等,本文使用的是 Tomcat 的支持。



否则会报错：

> HV000183: Unable to load 'javax.el.ExpressionFactory'. Check that you have the EL dependencies on the classpath, or use ParameterMessageInterpolator instead

```
package com.davidwang456.test;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
//import org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;

import java.util.Date;

import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.ValidatorFactory;

@Configuration
public class Config {

    @Bean
    public ClientBean clientBean () {
        return new ClientBean();
    }

    @Bean
    public Order order () {
        Order order = new Order();
        //order.setPrice(BigDecimal.TEN);
        order.setDate(new Date(System.currentTimeMillis() - 60000));
        return order;
    }

    @Bean
    public Validator validatorFactory () {
       // return new LocalValidatorFactoryBean();
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        return validatorFactory.getValidator();
    }
}
```

测试类：

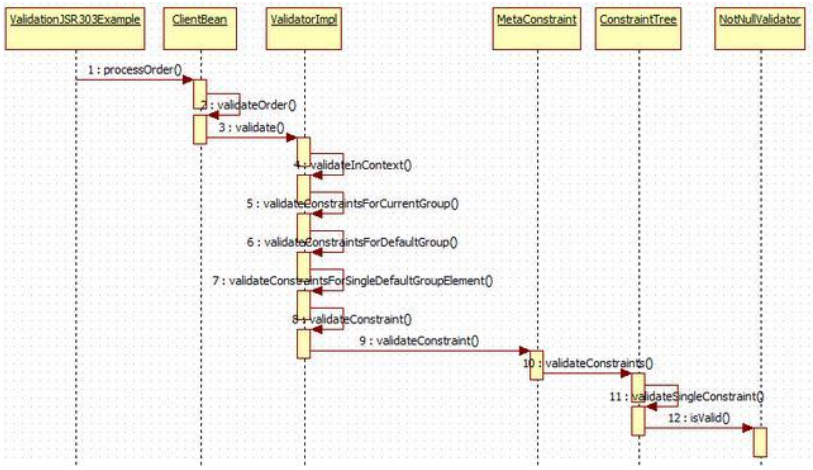
```
package com.davidwang456.test;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class ValidationJSR303Example {
    public static void main (String[] args) {
        AnnotationConfigApplicationContext context =
                            new AnnotationConfigApplicationContext(
                                    Config.class);

        ClientBean bean = context.getBean(ClientBean.class);
        bean.processOrder();
    }
}
```

执行结果：

> 日期不能为空
>
> 价格不能为空

工作原理

debug 源代码，可以看出整个的工作原理如下：



其中，比较重要的部分代码是调用不同的 Validator 实现类（Hibernate 自带的 Validator 实现类），ConstraintTree.java#validateSingleConstraint 见下图：

```
/**
 * @return an {@link Optional#empty()} if there is no violation or a corresponding {@link ConstraintValidatorContextImpl}
 *       otherwise.
 */
protected final <V> Optional<ConstraintValidatorContextImpl> validateSingleConstraint(
        ValueContext<?, ?> valueContext,
        ConstraintValidatorContextImpl constraintValidatorContext,
        ConstraintValidator<A, V> validator) {
    boolean isValid;
    try {
        @SuppressWarnings("unchecked")
        V validatedValue = (V) valueContext.getCurrentValidatedValue();
        isValid = validator.isValid( validatedValue, constraintValidatorContext );
    }
    catch (RuntimeException e) {
        if ( e instanceof ConstraintDeclarationException ) {
            throw e;
        }
        throw LOG.getExceptionDuringIsValidCallException( e );
    }
    if ( !isValid ) {
        //We do not add these violations yet, since we don't know how they are
        //going to influence the final boolean evaluation
        return Optional.of( constraintValidatorContext );
    }
    return Optional.empty();
}
```

重点

总结

Bean Validation 为 JavaBean 验证定义了相应的元数据模型和 API，Bean Validation 是一个运行时的数据验证框架，在验证之后验证的错误信息会被马上返回。

Hibernate Validator 是 Bean Validation 的参考实现 。 Hibernate Validator 提供了 JSR 303 规范中所有内置 constraint 的实现，除此之外还有一些附加的 constraint。

Hibernate 内部将 Validator 组成 Tree 形结构，通过可到达树来遍历 Validator。

}

22 Spring Core参数校验之 validator

24 Spring Core vs Spring MVC参数校验