64_技术挑战：如何解决创建订单时间格式参数错误的问题？

1、开篇

上节课作为本周的第一节课，专注于订单模块的讲解，主要讲解了订单的功能，以及创建订单的业务流程。本节课会和大家讲讲，如何解决创建订单时间格式参数错误的问题。今天课程的内容包括以下几个部分：

- 数据验证问题
- Spring 中的自定义验证

2、数据验证问题

上节课描述的订单功能中包括一些订单的输入信息，例如：“家教开始时间”、“家教结束时间”。我们需要对这些数据进行判断，从而保证输入的数据是符合标准的。由于“家教开始时间”、“家教结束时间”是两个重要的订单参数，只有知道开始和结束时间我们才能计算出教学的天数，从而才能知道订单的总价。因此这两个数据项是必填的，如果为空在系统中是不允许的。其次，这两个时间的数据格式需要满足一般时间的格式要求，否则系统是无法解析它们的。那么问题就转化为两个时间类型是否输入以及输入的格式是否正确，两个判断问题了。在传统方法中我们会在 Controller 的程序入口中对传入的两个变量进行“If”判断，按照上述逻辑对数据进行过滤，将有问题的数据进行提示，保证数据的正确性。这样做法的缺点也很明显，那就是在代码中保留大量的 “If-else” 语句导致整个代码难以维护，代码的可读性也降低了。

3、Spring 中的自定义验证

考虑到以上“If-else”的诸多不便在 Spring 中提供了很多自建的 annotation 帮助开发者进行数据验证，例如如下注释：

- @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

  不过如果遇到一些特殊情况例如判断手机号是否正确，日期是否符合规格就需要我们自己编写验证注释来完成客制化的验证功能。

  下面通过一个例子来看看 Spring 是如何实现自定义 Validate 的。假设有一个 User 实体，里面包含了 Phone 字段，需要实现一个对 Phone（手机号）的验证，判断手机号是否输入，并且看看手机号是否是“1”开头为 11 位的数字。

  如图 1 所示，建立一个 IsMobile 的接口，在该接口之上有几个注释，其中 @Target 注解表明自定义注解能够用来标记字段和方法，也就是说 IsMobile 这个注释可以用来修饰字段和方法。@Retention（RUNTIME）代表注解可以保留到运行时。

  @Constraint 表明注解是用来作为校验约束的。其中的 validatedBy 属性，指向了一个类，就是这个类中提供了约束检查相关的逻辑，后面会实现这个类。在 IsMobile 方法体中还定义了默认的 message 作为验证错误之后的信息。

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = {IsMobileValidator.class})
public @interface IsMobile {
    //校验错误的默认信息
    String message() default "手机号码格式有问题";
    //是否强制校验
    boolean isRequired() default false;
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

图 1 用于验证的注释（annotation）

定义完了验证的注释，再来定义验证的具体内容也就是，@Constraint 对应的具体验证约束。如图 2 所示，这里就是 IsMobileValidator 具体的验证类，该类实现了 ConstraintValidator，并且 Override 了 isValid 方法，在该方法中首先判断是否 required，这个值是在该类初始化的时候被传入的。也就是判断验证的字段是否必填，然后在通过 isMobile 方法验证电话是否符合验证的要求。其中 IsMobile 方法调用了正则表达式，该表达式判断了以 1 开头的 11 位数字，如果满足这个条件就说明是手机号码，表示验证通过。isValid 整个方法如果验证通过就会返回 true 否者会返回 false。

```java
public class IsMobileValidator implements ConstraintValidator<IsMobile, String> {
    private boolean required = false;
    private static final Pattern mobile_pattern = Pattern.compile("1\\d{10}");
    //工具方法，判断是否是手机号
    public static boolean isMobile(String src) {
        if (StringUtils.isEmpty(src)) {
            return false;
        }
        Matcher m = mobile_pattern.matcher(src);
        return m.matches();
    }
    @Override
    public void initialize(IsMobile constraintAnnotation) {
        required = constraintAnnotation.isRequired();
    }
    @Override
    public boolean isValid(String phone, ConstraintValidatorContext constraintValidatorContext) {
        //是否为手机号的实现
        if (required) {
            return isMobile(phone);
        } else {
            if (StringUtils.isEmpty(phone)) {
                return true;
            } else {
                return isMobile(phone);
            }
        }
    }
}
```

图 2 具体验证类 IsMobileValidator

定义完验证注释和对应的验证类之后，我们回到实体类将需要验证的字段打上注释。如图 3 所示，在 User 类中在 phone 字段上面打上@IsMobile 的注释，当对该字段赋值的时候，就会对其进行验证，运行 IsMobileValidator 中定义的验证代码，如果没有通过验证就会返回“手机号码格式有问题”的提示。

```java
@Data
public class User {
    @NotNull
    @Size(min=2, max=30,message = "请检查名字的长度是否有问题")
    private String name;

    @NotNull
    @Min(18)
    private Integer age;

    //这里是新添加的注解
    @IsMobile
    private String phone;
}
```

图 3 对实体类中的 phone 字段进行验证

4、总结

本节课延续上节课订单业务流程的内容，思考了在创建订单的时候需要对教学时间进行验证，为了提高代码的可读性会使用 Spring 的注释验证方式。不过针对我们的验证要求会使用到 Spring 的自定义验证，通过一个手机号验证的方式让大家快速地了解该方式的运作原理。

下节课进入码实战，带大家通过 Spring 自定义注解方式将时间参数进行统一校验。下期见，拜拜。