Lombok

简介

Lombok 是一个可以通过简单的注解形式(@Data、@Setter、@Getter)来帮助我们简化消除一些必须有但是很臃肿的 Java 代码的工具。通过使用对应的注解，可以在编译源码的时候生成对应的方法。

官方地址: https://projectlombok.org/

Github 地址：https://github.com/rzwitserloot/lombok

IDEA 使用

在非 Maven 工程将 lombok Jar 导入类加载路径或在 Maven 工程通过 pom 加入依赖后，使用 lombok 不起作用，需先安装插件。

步骤 1. File --> Settings --> Plugins 搜索 lombok，右侧点击 install

步骤 2. File --> Settings --> Build、 Execution、 Deployment --> Compiler --> Annotation Processors，勾选 Enable annotation processing

pom 引入 lombok 依赖

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.16.14</version>
</dependency>
```

注解

@Getter and @Setter

可以用@Getter 和@Setter 注释任何字段，让 lombok 自动生成默认的 getter/setter 方法。默认生成的方法是 public 的，通过设置 AccessLevel 来修改方法修饰符。

```
@Getter(value =  AccessLevel.PROTECTED)
```

@ToString

生成 toString()方法，默认按顺序(逗号分隔)打印类名称以及每个字段。
可以通过**@ToString(exclude=”id”)**或@ToString(exclude={”id”，”name”})过滤字段。
可以通过**@ToString(callSuper = true)**让其调用父类的 toString()方法。

Example1	@ToString(exclude = {"id","name"},callSuper = true)

生成方法	public String toString() {
	return "People(super=" + super.toString() + ", age=" + this.getAge() + ")";
	}

Example2	@ToString(callSuper = false)

生成方法	public String toString() {
	return "People(id=" + this.getId() + ", name=" + this.getName() +
	", age=" + this.getAge() + ")";
	}

@EqualsAndHashCode(不常用)

生成 hashCode()和 equals()方法，默认使用所有非静态、非 transient 字段。可以通过在可选的 exclude 参数中来排除更多字段，或通过 parameter 参数中准确指定希望使用哪些字段。

@NoArgsConstructor

生成一个无参构造方法。当类中有 final 字段没有被初始化时，编译器会报错。此时可用@NoArgsConstructor(force=true)为没有初始化的 final 字段设置默认值 0/false/null；


```
1  package com.domain;
2
3  import ...
7
8  @NoArgsConstructor(force = true)
9  public class People{
10     private Long id;
11     private String name;
12     private Integer age;
13     private final boolean tag;
14 }
```

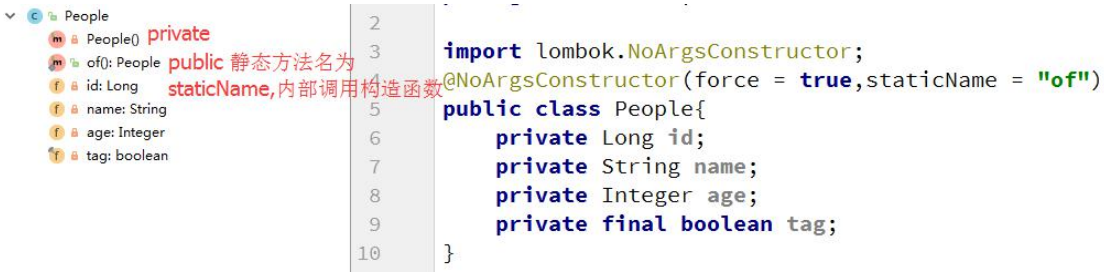
@RequiredArgsConstructor

@RequiredArgsConstructor 会生成构造方法（可能带参数也可能不带参数），如果带参数，这参数只能是以 final 修饰的未经初始化的字段，或者是以@NonNull 注解的未经初始化的字段

@AllArgsConstructor

生成一个全参数的构造方法。

@NoArgsConstructor @RequiredArgsConstructor @AllArgsConstructor 均有 staticName 属性，用法如下：



@Data

@Data =@ToString + @EqualsAndHashCode + @Getter + @Setter + @RequiredArgsConstructor

@Accessors(不常用)

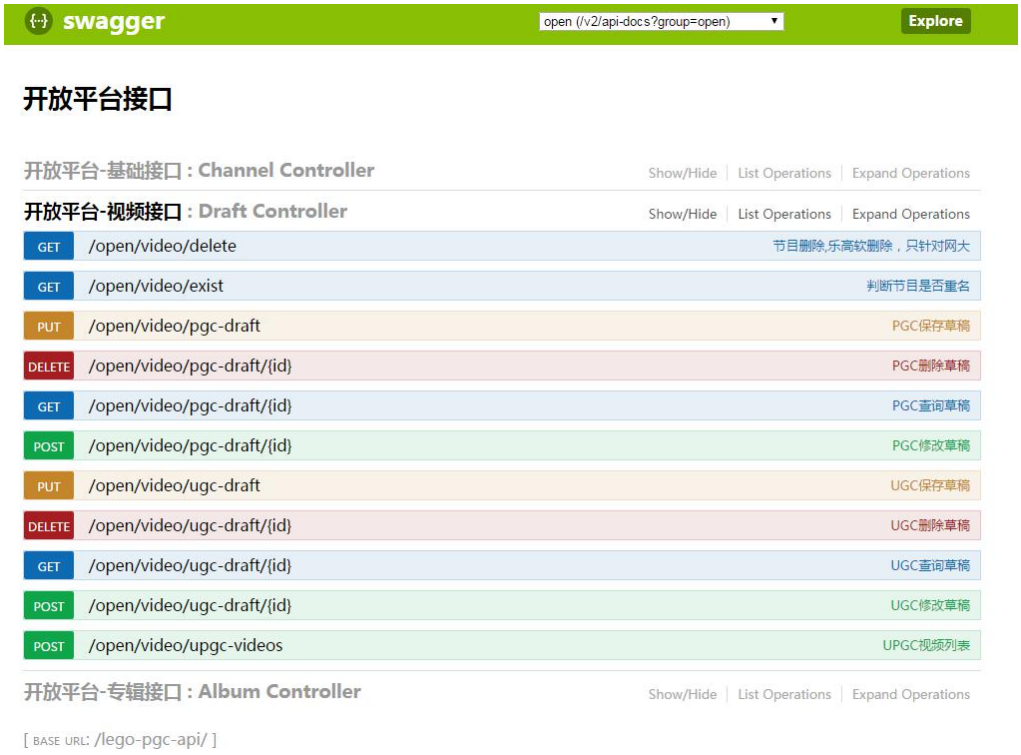
主要用于控制生成的 set、get 方法。

- ✧ fluent boolean 值，默认为 false。此字段主要为控制生成的 getter 和 setter 方法前面是否带 get/set
- ✧ chain boolean 值，默认 false。如果设置为 true，setter 返回的是此对象，方便链式调用方法
- ✧ prefix 设置前缀 例如：@Accessors(prefix = "abc") private String abcAge 当生成 get/set 方法时，会把此前缀去掉

Swagger

简介

Swagger 是一款 RESTFUL 接口的文档在线自动生成+功能测试功能软件，用于生成、描述、调用和可视化 RESTFUL 风格的 Web 服务。总体目标是使客户端和文件系统作为服务器以同样的速度来更新，文件的方法、参数和模型紧密集成到服务器端的代码，允许 API 来始终保持同步。Swagger 也提供了强大的页面测试功能来调试每个 RESTful API。



Spring Boot + Swagger2

采用未封装的 springfox-swagger2 和 springfox-swagger-ui 集成 Swagger2，参考 https://www.jianshu.com/p/136d7d816eca。

采用封装的 starter 进行集成：如下

1. pom.xml 中加入依赖

```
<dependency>
    <groupId>com.didispace</groupId>
    <artifactId>spring-boot-starter-swagger</artifactId>
    <version>1.1.0.RELEASE</version>
</dependency>
```

2. 在应用主类中增加@EnableSwagger2Doc,默认情况下就能产生所有当前 Spring MVC 加载的请求映射文档

参数设置

下列选项只是在 swagger-ui.html 界面上显示相关信息。

- swagger.title=标题
- swagger.description=描述
- swagger.version=版本
- swagger.license=许可证
- swagger.licenseUrl=许可证 URL
- swagger.termsOfServiceUrl=服务条款 URL
- swagger.contact.name=维护人
- swagger.contact.url=维护人 URL
- swagger.contact.email=维护人 email
- swagger.base-package=swagger 扫描的基础包，默认：全扫描
- swagger.base-path=需要处理的基础 URL 规则，默认：/**
- swagger.exclude-path=需要排除的 URL 规则，默认：空

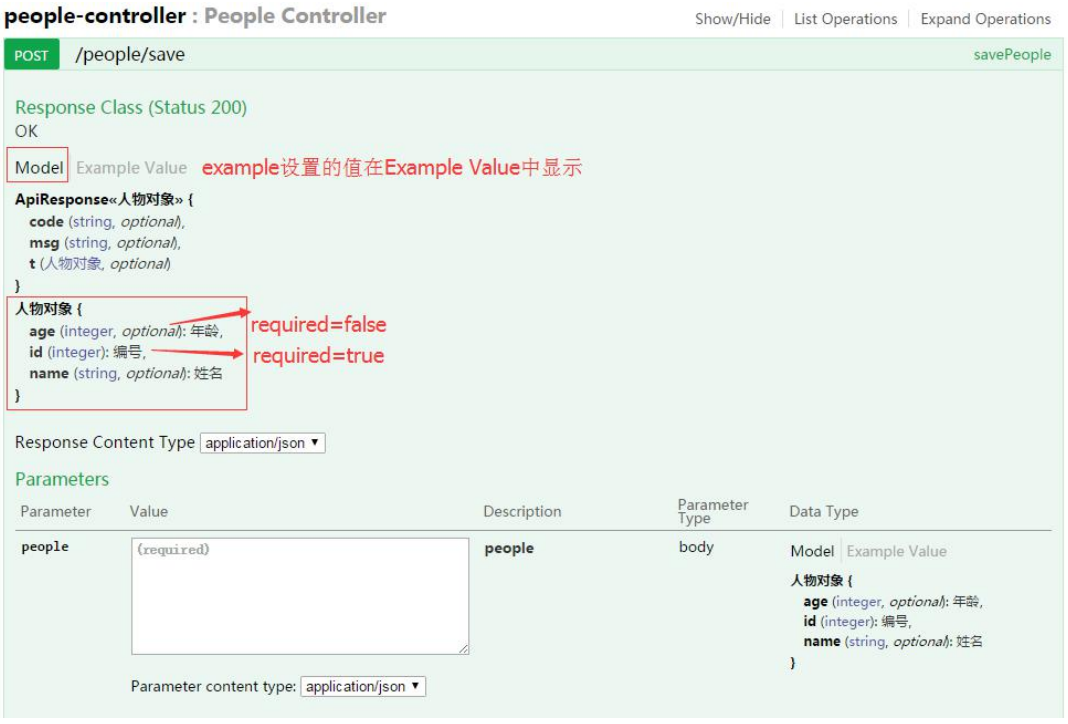
注解用法

@ApiModel & @ApiModelProperty

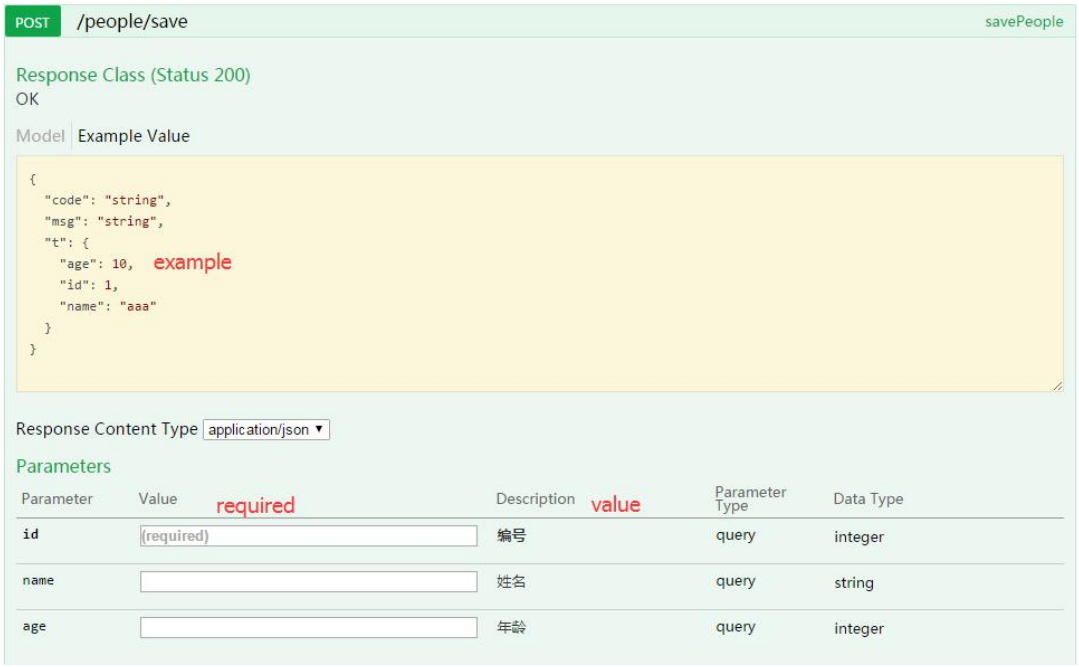
主要用于标注 Bean 对象，当 Bean 对象作为入参以及返回值时进行限制、字段说明和举例等，配合@ApiModelProperty 使用。@ApiModel 是对类对象进行描述，@ApiModelProperty 用于描述类对象的成员变量等。


```java
@ApiModel(value = "人物对象")
public class People {
    @ApiModelProperty(value = "编号",required = true,example = "1")
    private Integer id;
    @ApiModelProperty(value = "姓名",required = false,example = "aaa")
    private String name;
    @ApiModelProperty(value = "年龄",required = false,example = "10")
    private Integer age;
```

ContentType:application/json

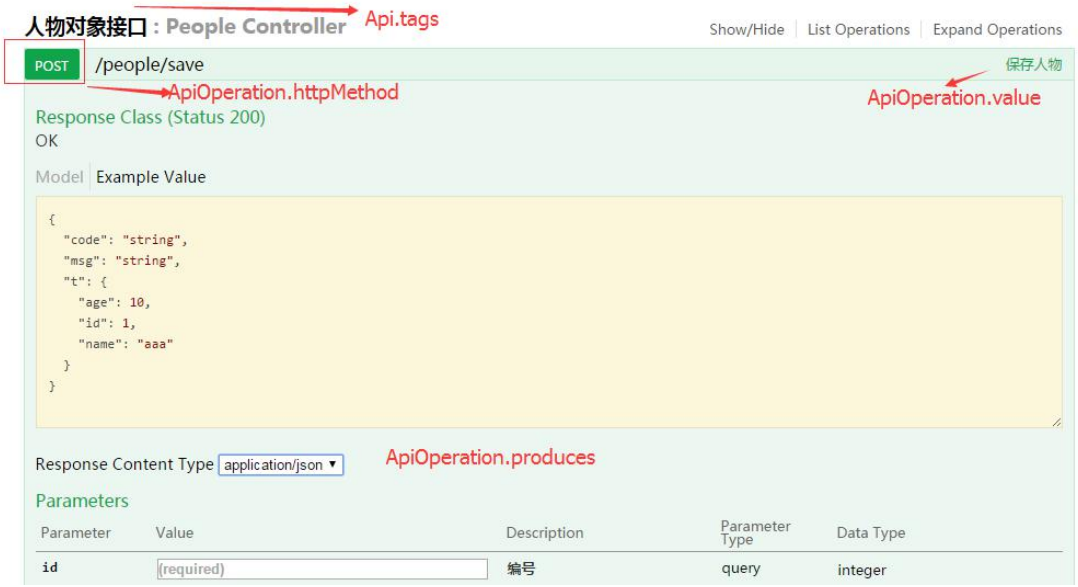


ContentType:application/x-www-form-urlencoded



@Api & @ApiOperation

Api 用在类上，说明该类的作用。可以标记一个 Controller 类做为 swagger 文档资源，ApiOperation：用在方法上，说明方法的作用，每一个 url 资源的定义。



@ApiParam

用于描述单独入参或几个相互独立入参，格式：
@ApiParam(required = “是否必须参数”, name = “参数名称”, value = “参数具体描述”

其他

此外还有 @ApiImplicitParams 、 @ApiImplicitParam @ApiResponse 等用法，参考 https://www.jianshu.com/p/12f4394462d5

Hibernate Validator

简介

JSR-303 是 Java EE6 的一项子规范，叫做 Bean Validation。Bean Validation 为 Java Bean 验证定义了相应的元数据模型和 API，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，提供了 JSR-303 规范中所有内置的 constraint 的实现，除此之外还有一些附加的 constraint。

SpringMVC SpringBoot 默认支持 Hibernate Validator，无需配置。

注解类型

注解	作用
AssertTrue	布尔值为真
AssertFalse	布尔值为假
Null	引用为空
NotNull	引用不为空
NotEmpty	字符串引用和值都不是空
Min	数字的最小值
Max	数字的最大值
Past	日期必须是过去
Future	日期必须是未来
Pattern	字符串必须匹配正则表达式
Valid	递归验证引用
Size	验证字符串是否在Size范围内
Email	验证字符串是否是一个有效的电子邮箱
URL	字符串是否是一个有效的URL

Email、URL 两个注解是 Hibernate Validator 自定义的，假如使用其他的 Bean Validation 实现，可能没有这两个注解。

Example

1. Bean 标注限制注解

```
public class People {
    @NotNull(message = "id 不能为空")
    private Integer id;
    @Size(min = 3,max = 5)
    private String name;
    @Max(value = 10)
    private Integer age;
}
```

2. @Valid 标注要检验的 Bean，后面紧跟 BindingResult 可以避免直接抛出异常返回信息，可判断 BindingResult 中是否含有 error 分别处理。

```
public ApiResponse<People> savePeople(@RequestBody @Valid People people,BindingResult result)
```

自定义限制注解

1. 定义注解

```
@Target({ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy=MoneyValidator.class)
public @interface Money {
    String message() default"不是金额形式";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

2. 实现 javax.validation.ConstraintValidator 接口

```
public class MoneyValidator implements ConstraintValidator<Money, Double>
```