**https://blog.csdn.net/motui/article/details/79012846**

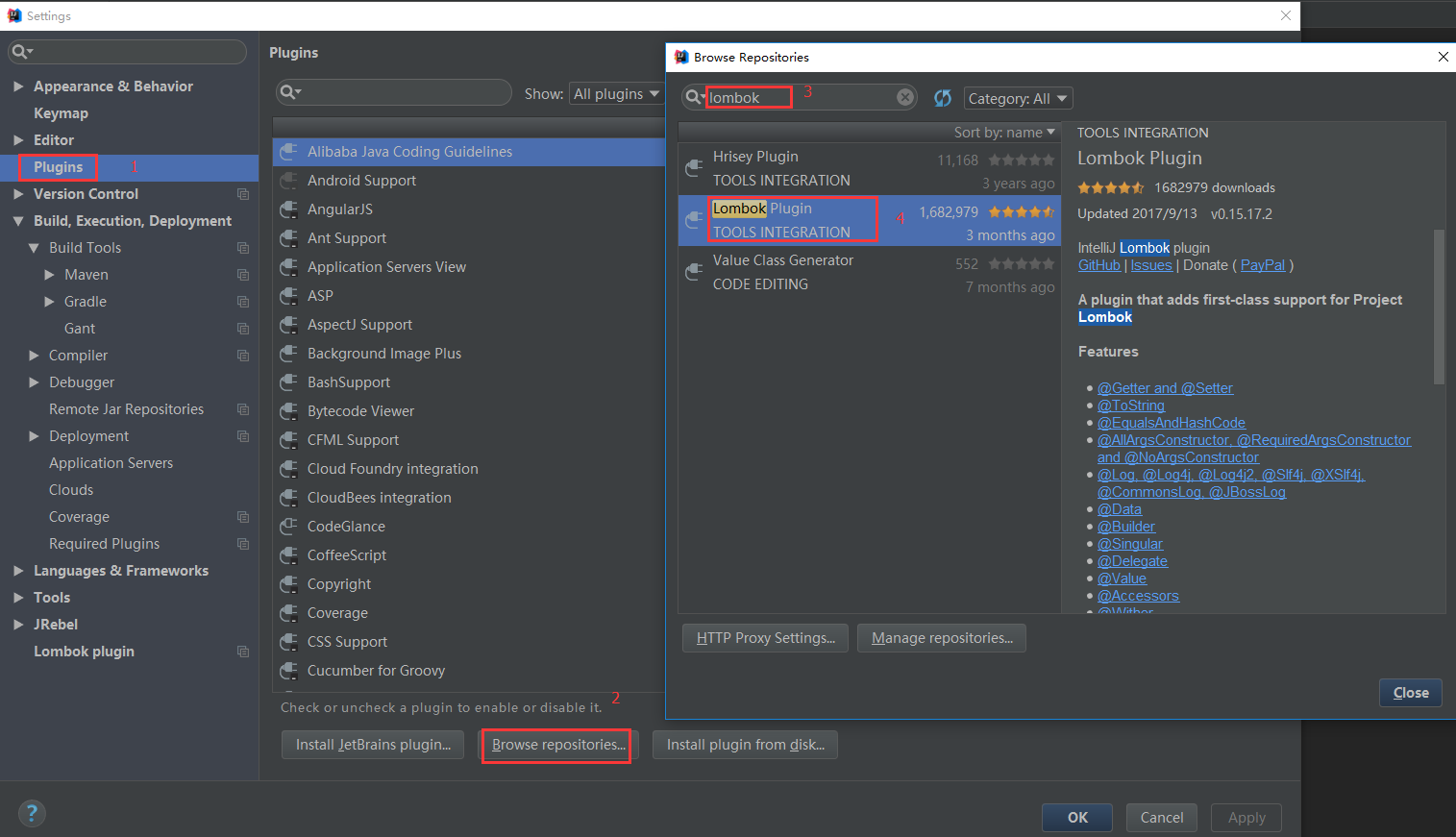
**Lombok使用**

**介绍**

在项目中使用Lombok可以减少很多重复代码的书写。比如说getter/setter/toString等方法的编写。

**IDEA中的安装**

打开IDEA的Setting –> 选择Plugins选项 –> 选择Browse repositories –> 搜索lombok –> 点击安装 –> 安装完成重启IDEA –> 安装成功



**引入依赖**

在项目中添加Lombok依赖jar，在pom文件中添加如下部分。(不清楚版本可以在[Maven仓库](http://mvnrepository.com/)中搜索)

<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.18</version>

<scope>provided</scope>

</dependency>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

**使用**

在对应的类或者方法上使用对应注解即可。例如： 

**Lombok有哪些注解**

* @Setter
* @Getter
* @Data
* @Log(这是一个泛型注解，具体有很多种形式)
* @AllArgsConstructor
* @NoArgsConstructor
* @EqualsAndHashCode
* @NonNull
* @Cleanup
* @ToString
* @RequiredArgsConstructor
* @Value
* @SneakyThrows
* @Synchronized

**注解详情**

**log**

注解在 **类** 上。有如下可选择可用：

//@CommonsLog

private static final org.apache.commons.logging.Log log = org.apache.commons.logging.LogFactory.getLog(LogExample.class);

//@JBossLog

private static final org.jboss.logging.Logger log = org.jboss.logging.Logger.getLogger(LogExample.class);

//@Log

private static final java.util.logging.Logger log = java.util.logging.Logger.getLogger(LogExample.class.getName());

//@Log4j

private static final org.apache.log4j.Logger log = org.apache.log4j.Logger.getLogger(LogExample.class);

//@Log4j2

private static final org.apache.logging.log4j.Logger log = org.apache.logging.log4j.LogManager.getLogger(LogExample.class);

//@Slf4j

private static final org.slf4j.Logger log = org.slf4j.LoggerFactory.getLogger(LogExample.class);

//@XSlf4j

private static final org.slf4j.ext.XLogger log = org.slf4j.ext.XLoggerFactory.getXLogger(LogExample.class);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

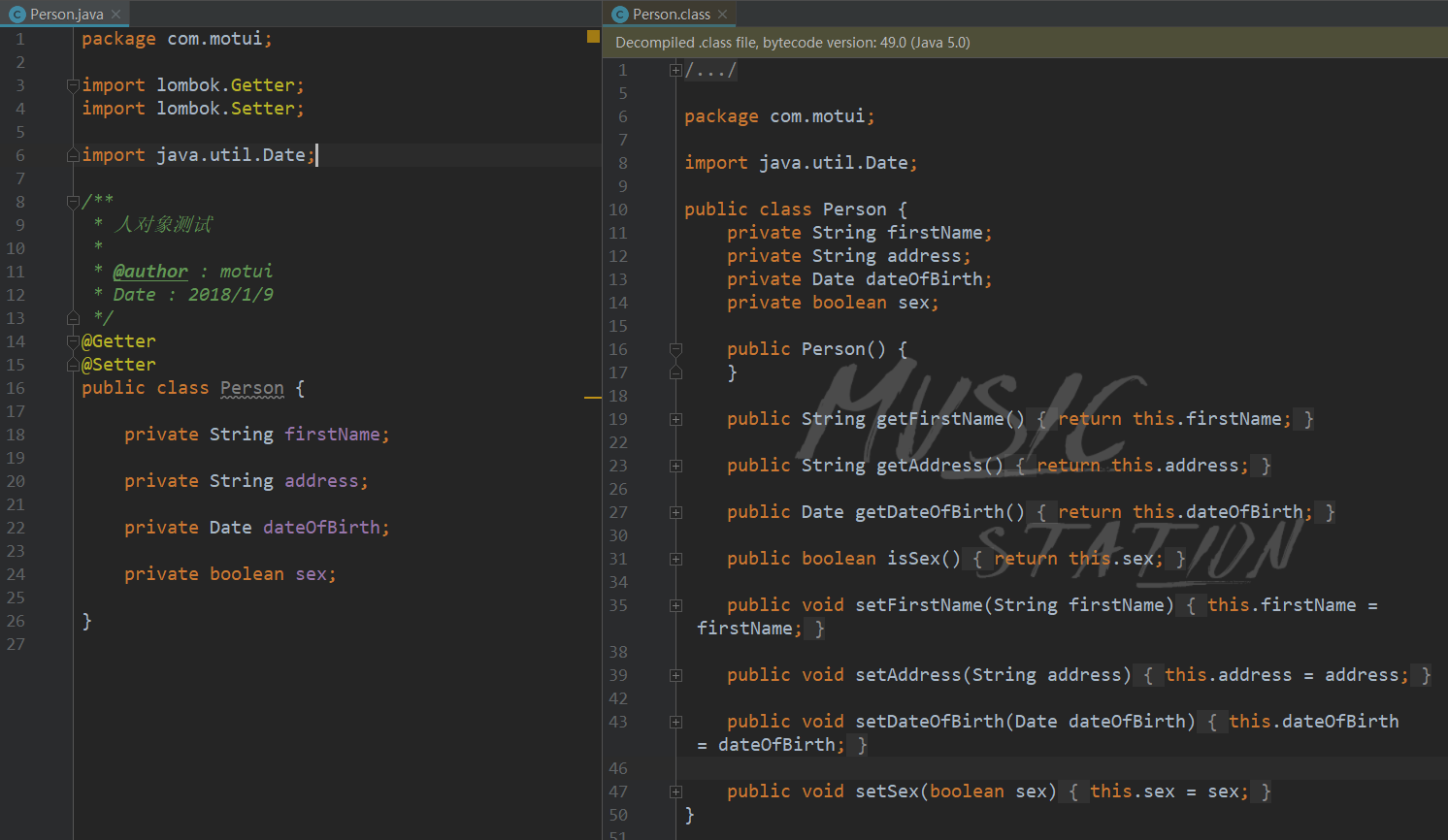
默认情况下，记录器的主题（或名称）将是使用注释进行@Log注释的类的类名称。这可以通过指定topic参数来定制。例如：@XSlf4j(topic="reporting")。

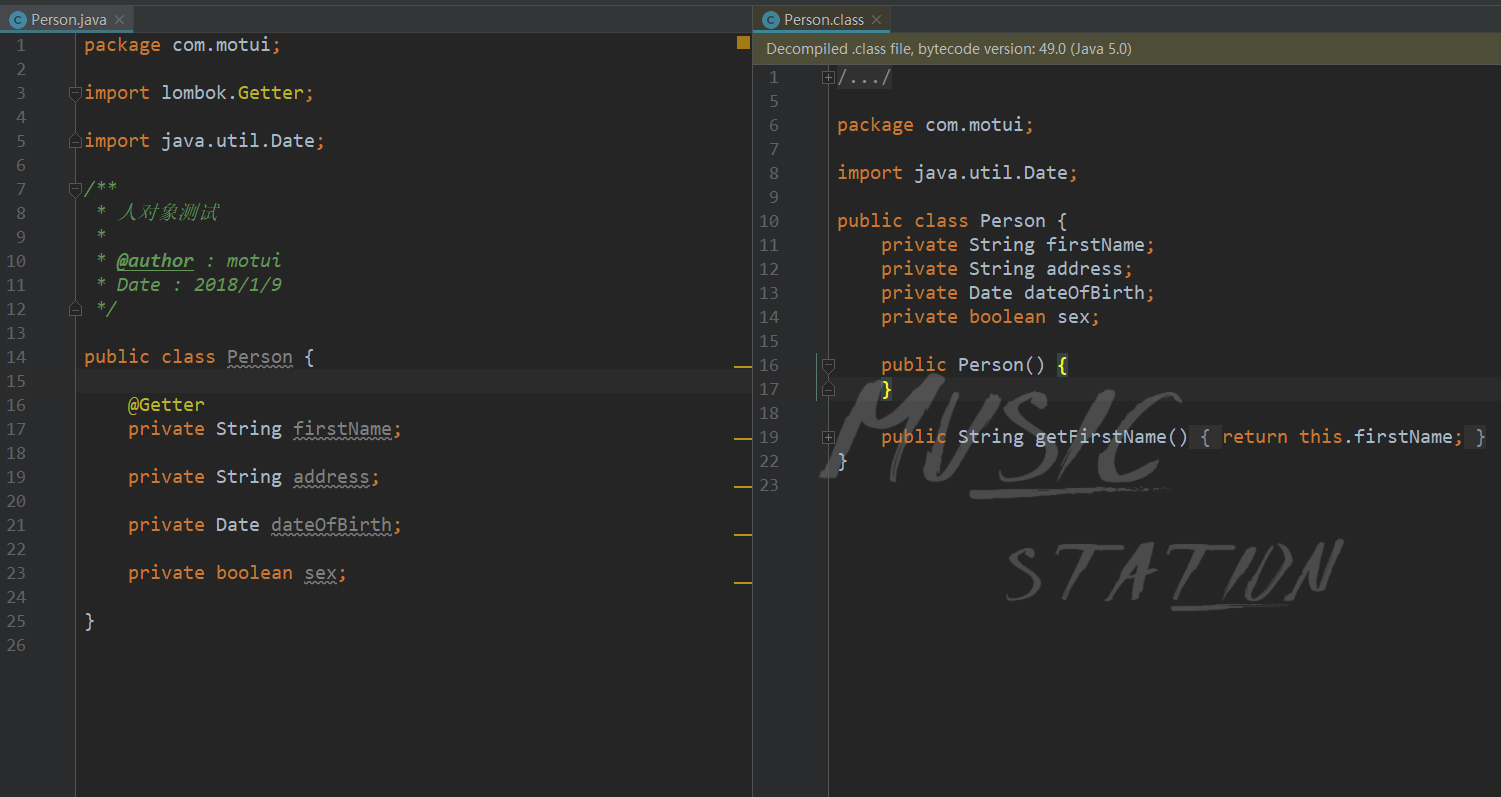
该类型注解可以满足不同的日志系统的日志使用，Lombok提供了一些自定义配置项可以参看官方说明文档。

[Log官方介绍](https://projectlombok.org/features/log)

**@Getter和@Setter**

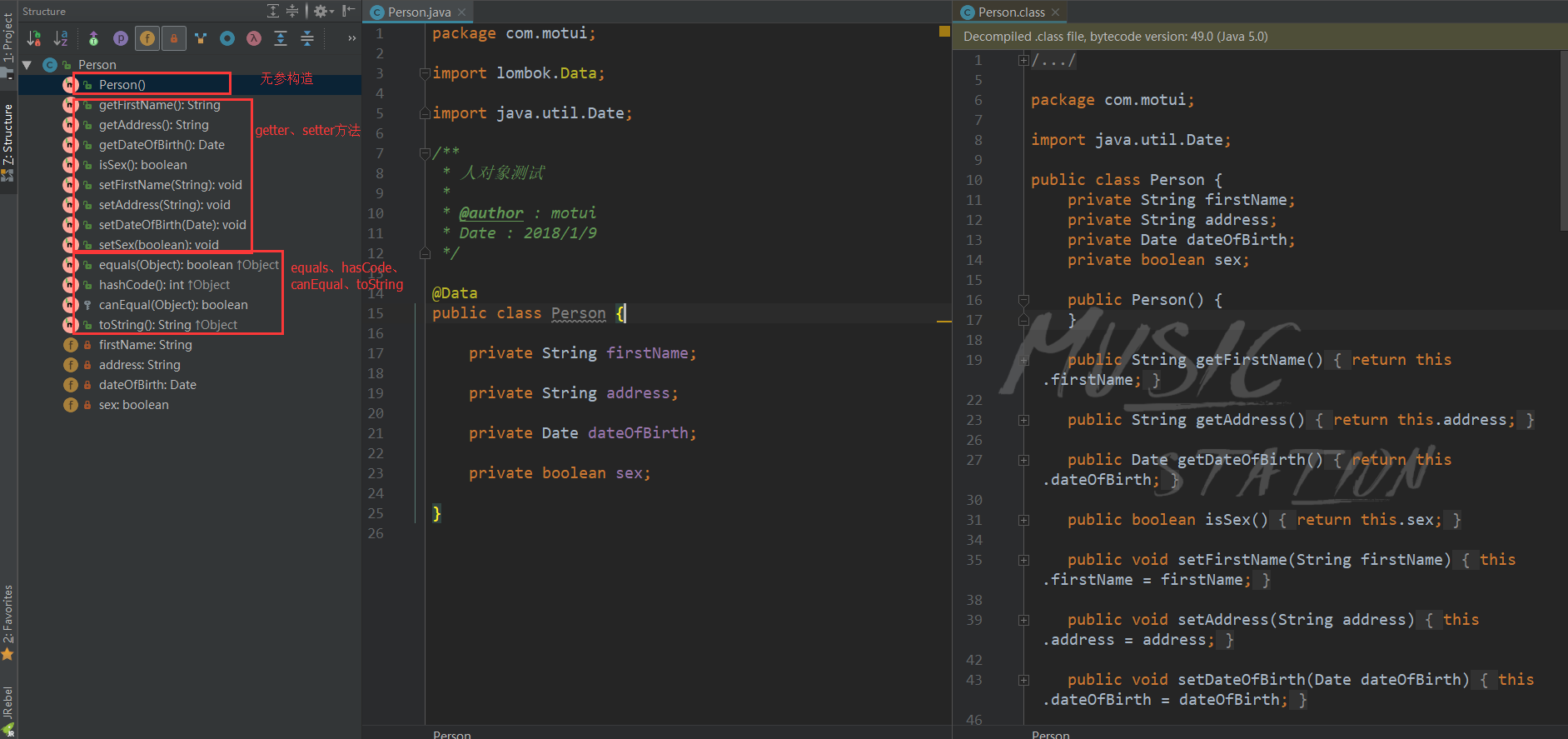
该注解使用在**类或者属性**上，该注解可以使用在类上也可以使用在属性上。生成的getter遵循布尔属性的约定。例如：boolean类型的sex,getter方法为isSex而不是getSex

在使用该注解时，会默认生成一个无参构造。和对应的getterhe setter方法   


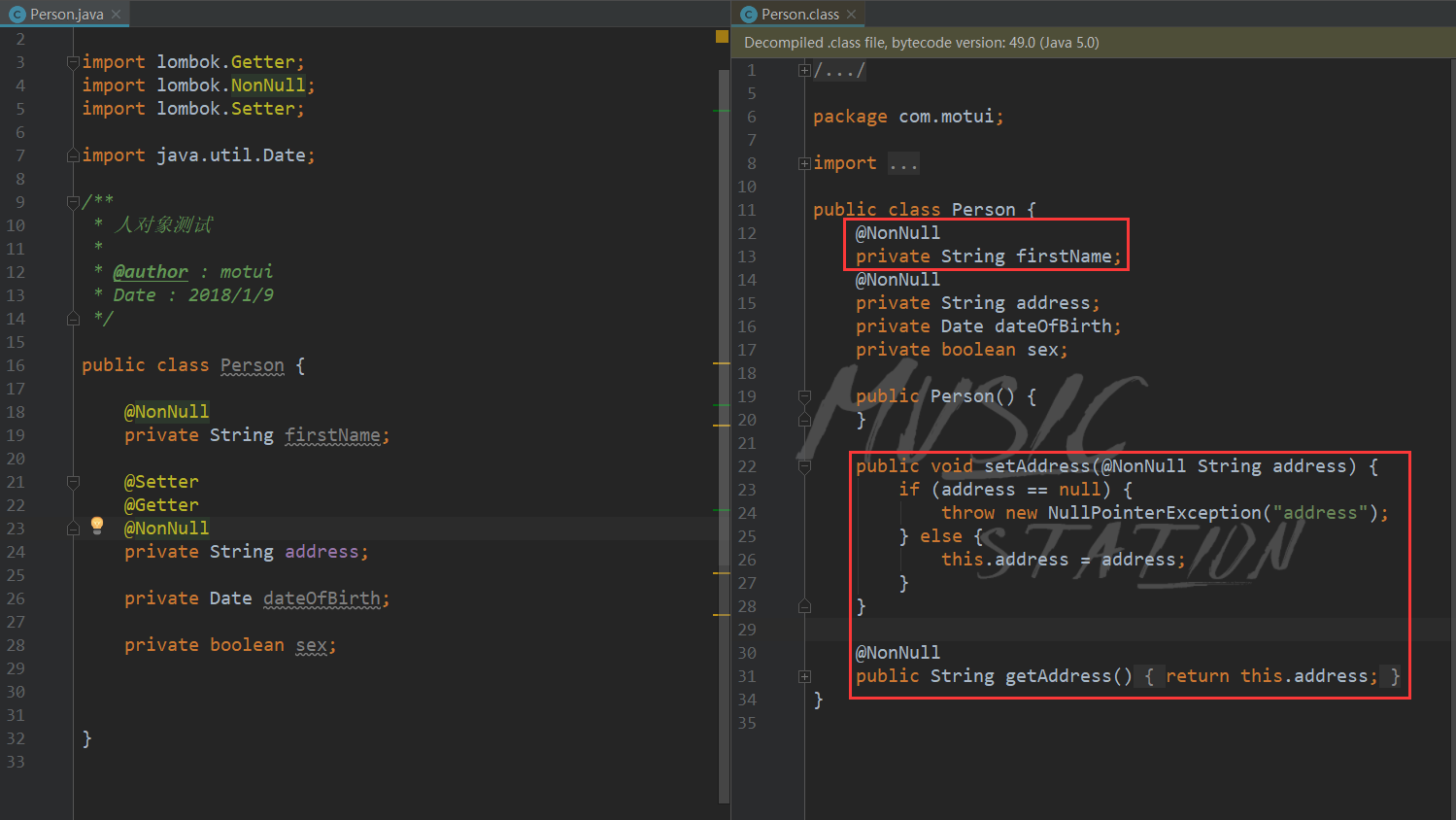
该注解也可以使用在单个属性上，会默认生成一个无参构造：   


**@Data**

该注解使用在**类**上，该注解会提供getter、setter、equals、canEqual、hashCode、toString方法。

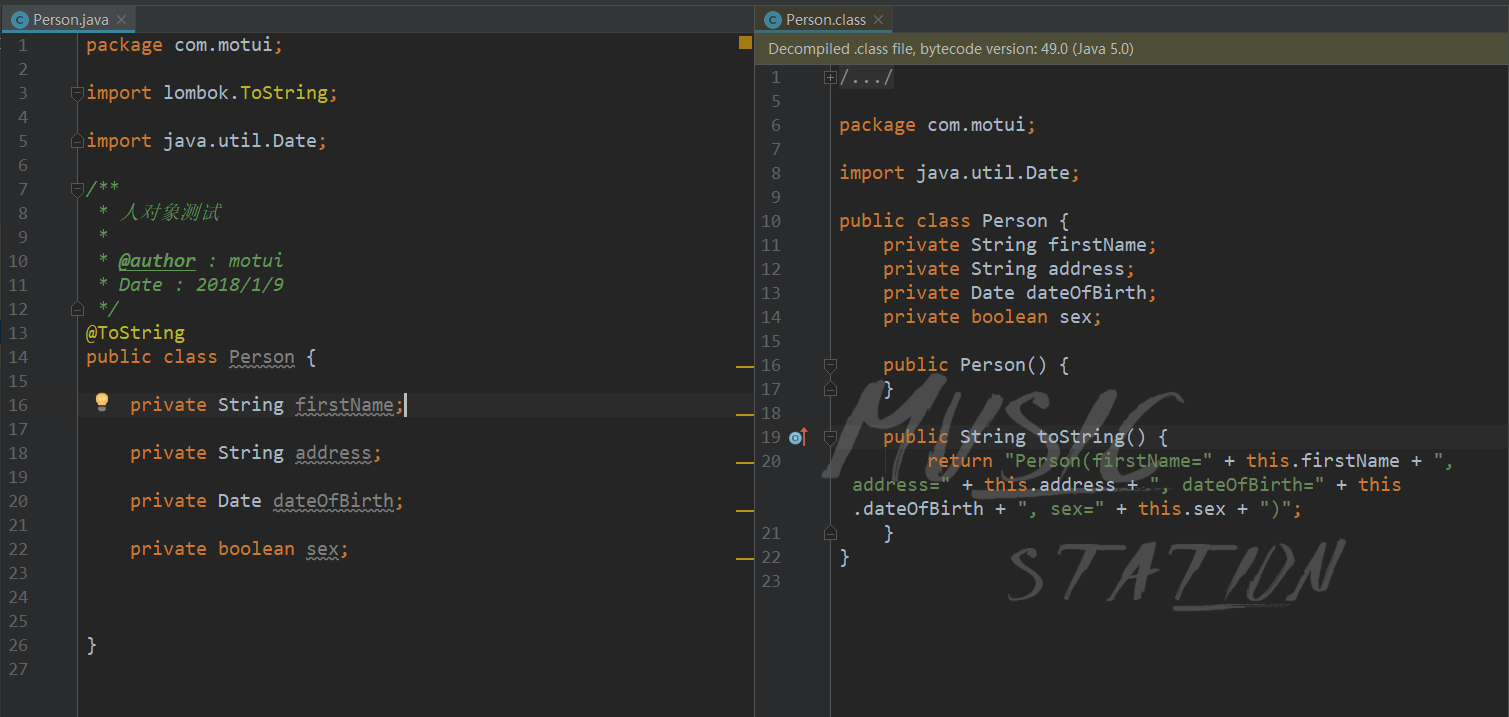
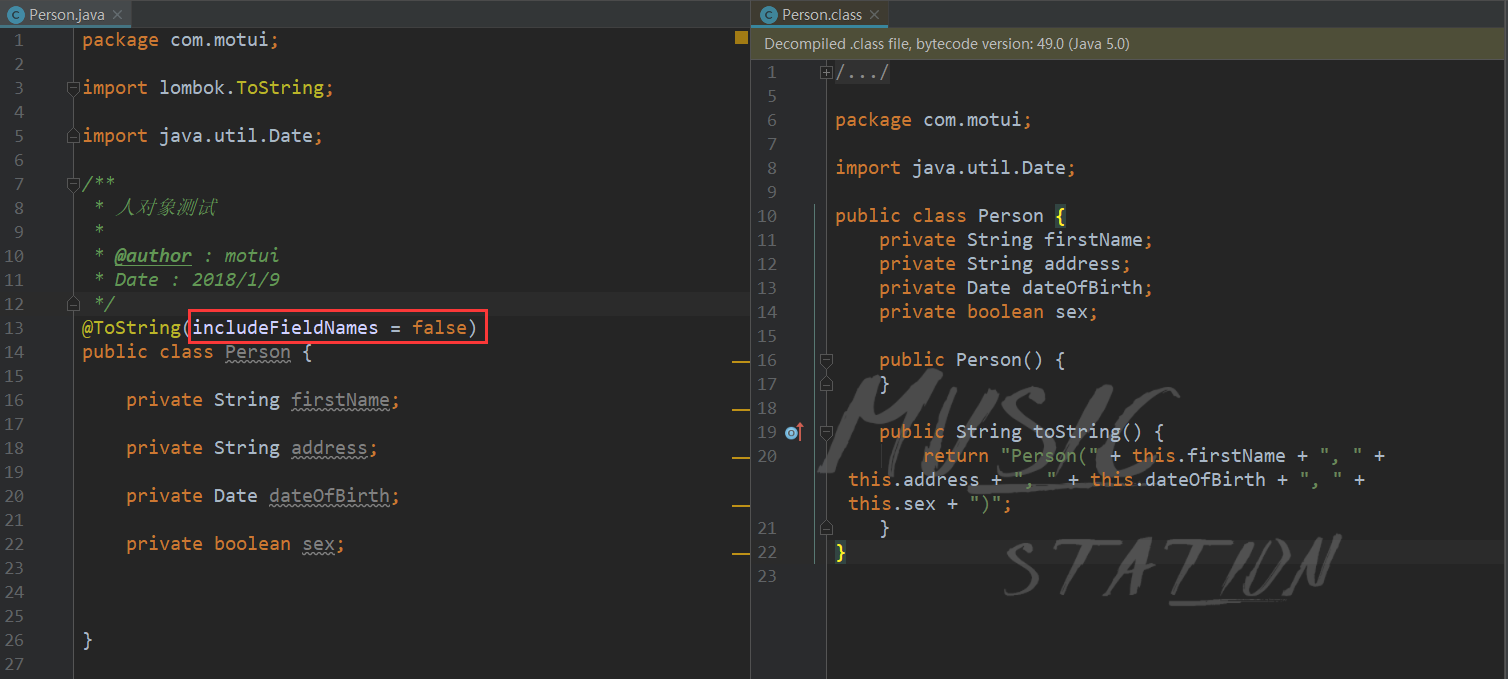
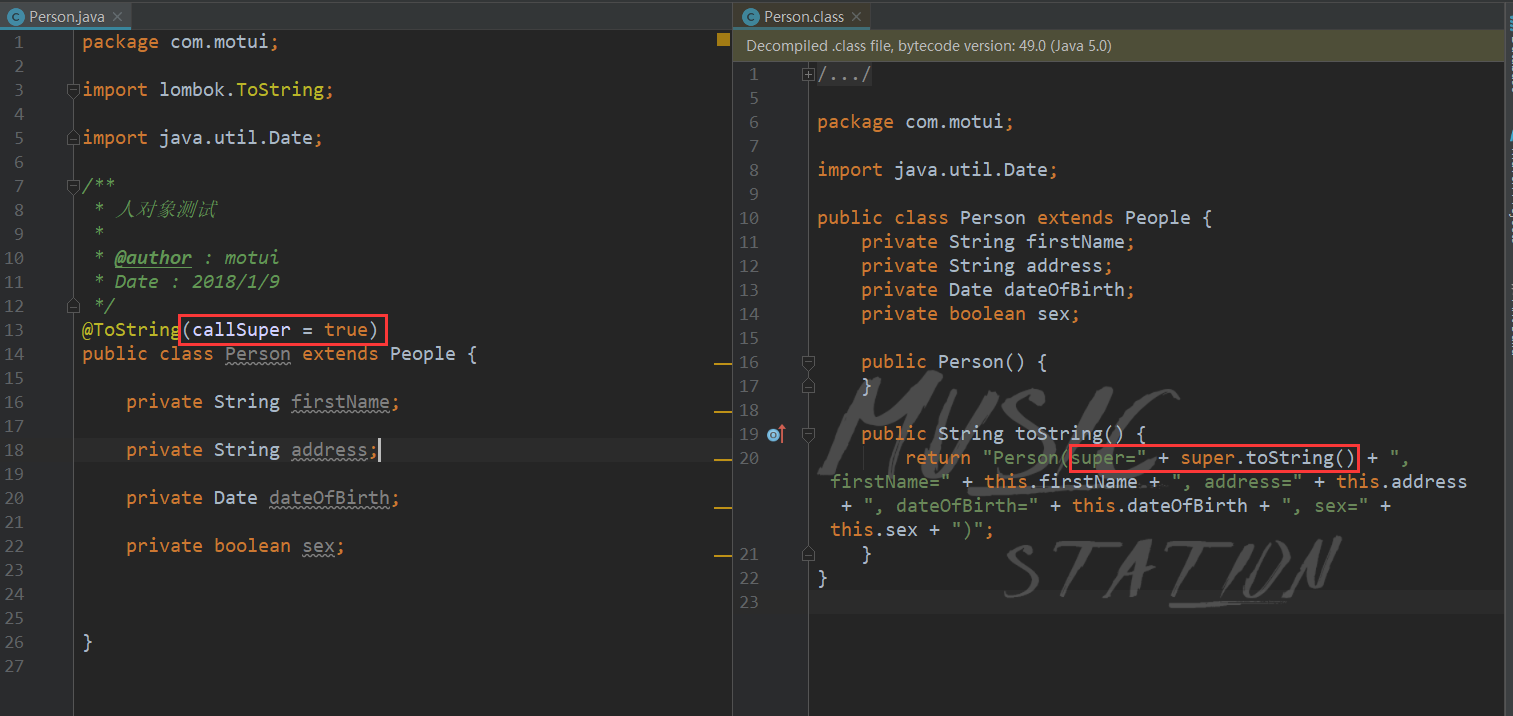


**@NonNull**

该注解使用在**属性**上，该注解用于属的非空检查，当放在setter方法的字段上，将生成一个空检查，如果为空，则抛出NullPointerException。   
该注解会默认是生成一个无参构造。   


**@toString**

该注解使用在**类**上，该注解默认生成任何非讲台字段以名称-值的形式输出。   
1、如果需要可以通过注释参数includeFieldNames来控制输出中是否包含的属性名称。   
2、可以通过exclude参数中包含字段名称，可以从生成的方法中排除特定字段。   
3、可以通过callSuper参数控制父类的输出。

* includeFieldNames是否包含属性名称：   
  
* exclude 排除指定字段   
  
* callSuper输出父类属性   
  

注意：父类也要有toString方法，不然打印的是对象内存地址

//父类无toString方法

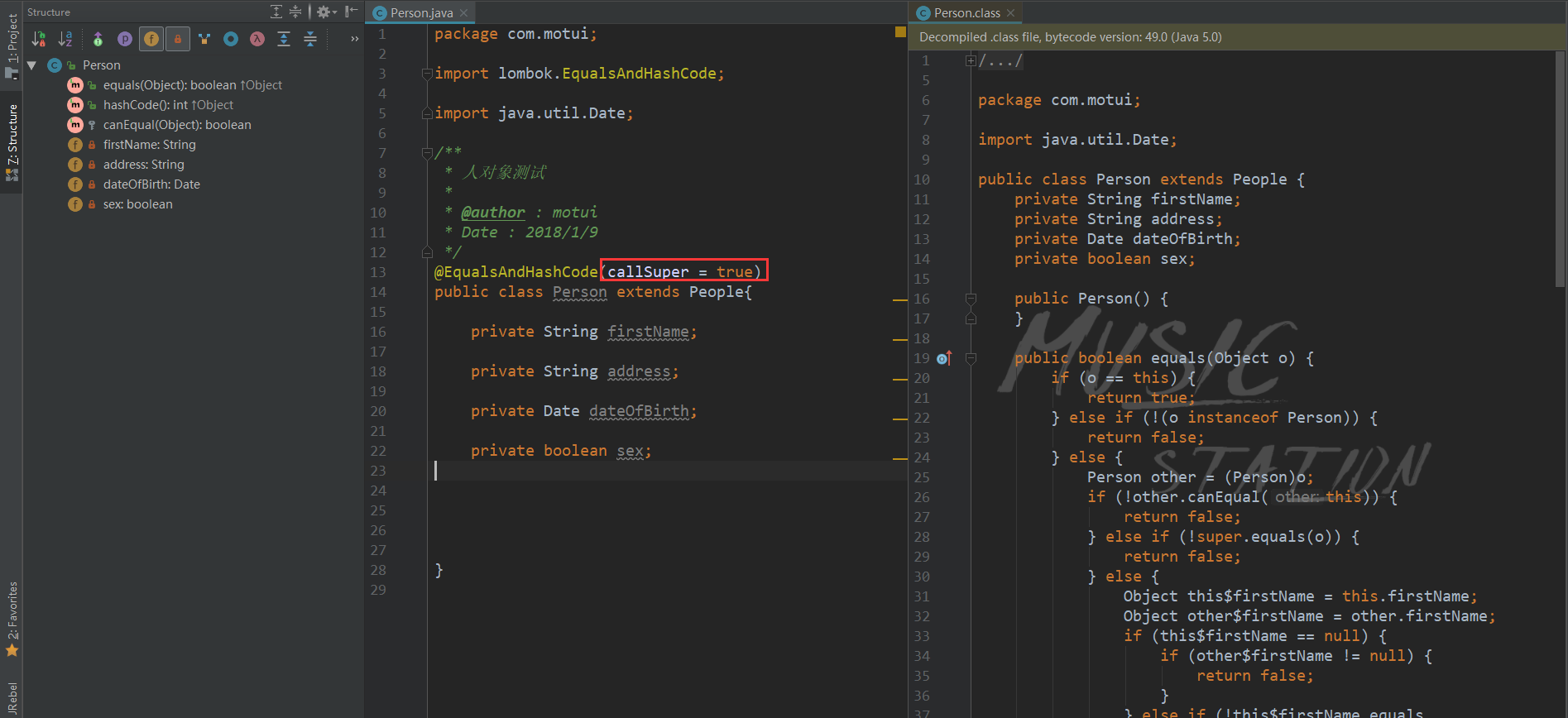
Person(super=com.motui.Person@3abfe836, firstName=motui, address=北京, dateOfBirth=Tue Jan 09 11:49:05 CST 2018, sex=true)

//父类有toString方法

Person(super=People(id=111), firstName=motui, address=北京, dateOfBirth=Tue Jan 09 11:50:11 CST 2018, sex=true)

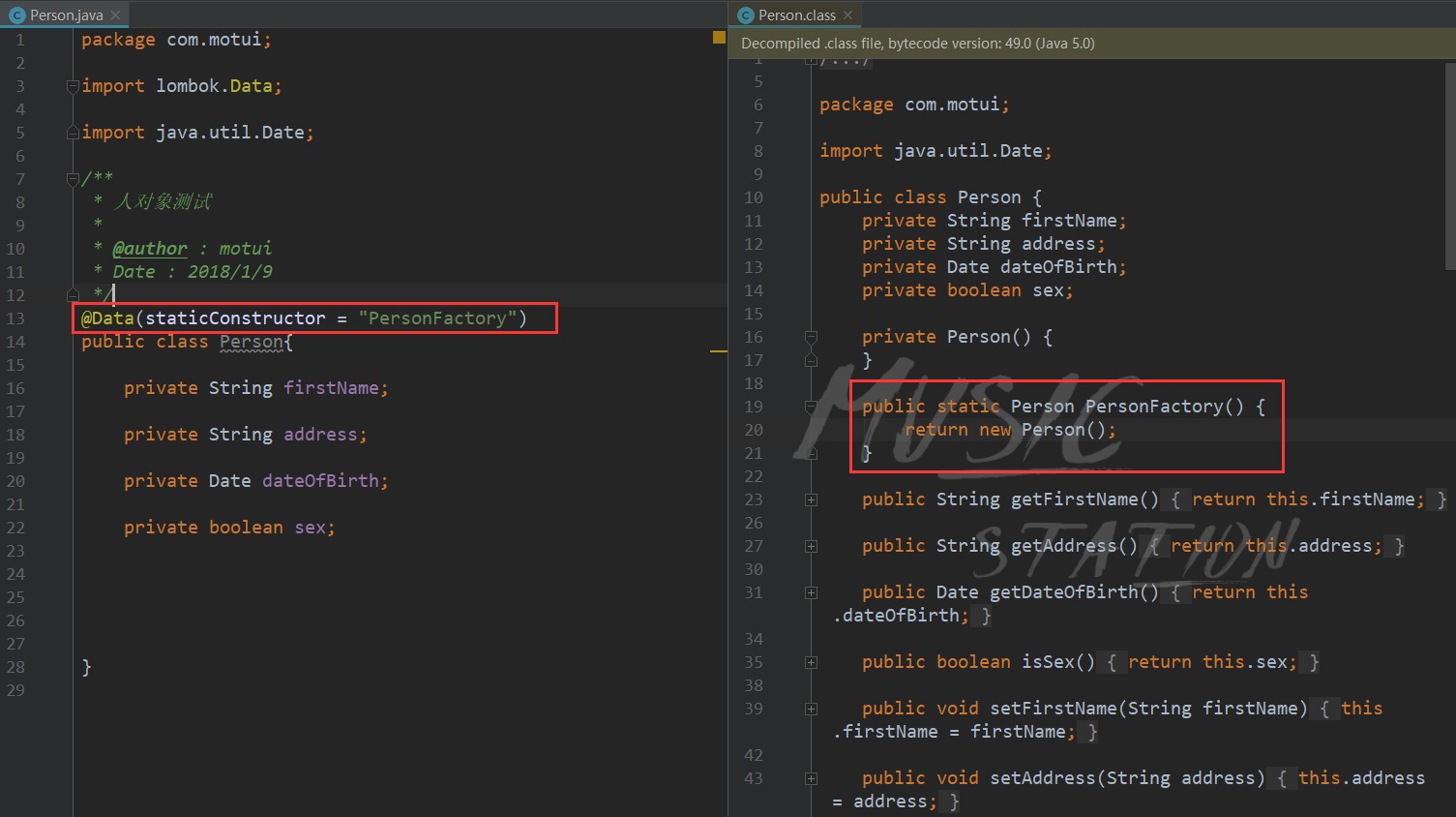
* 1
* 2
* 3
* 4

**@EqualsAndHashCode**

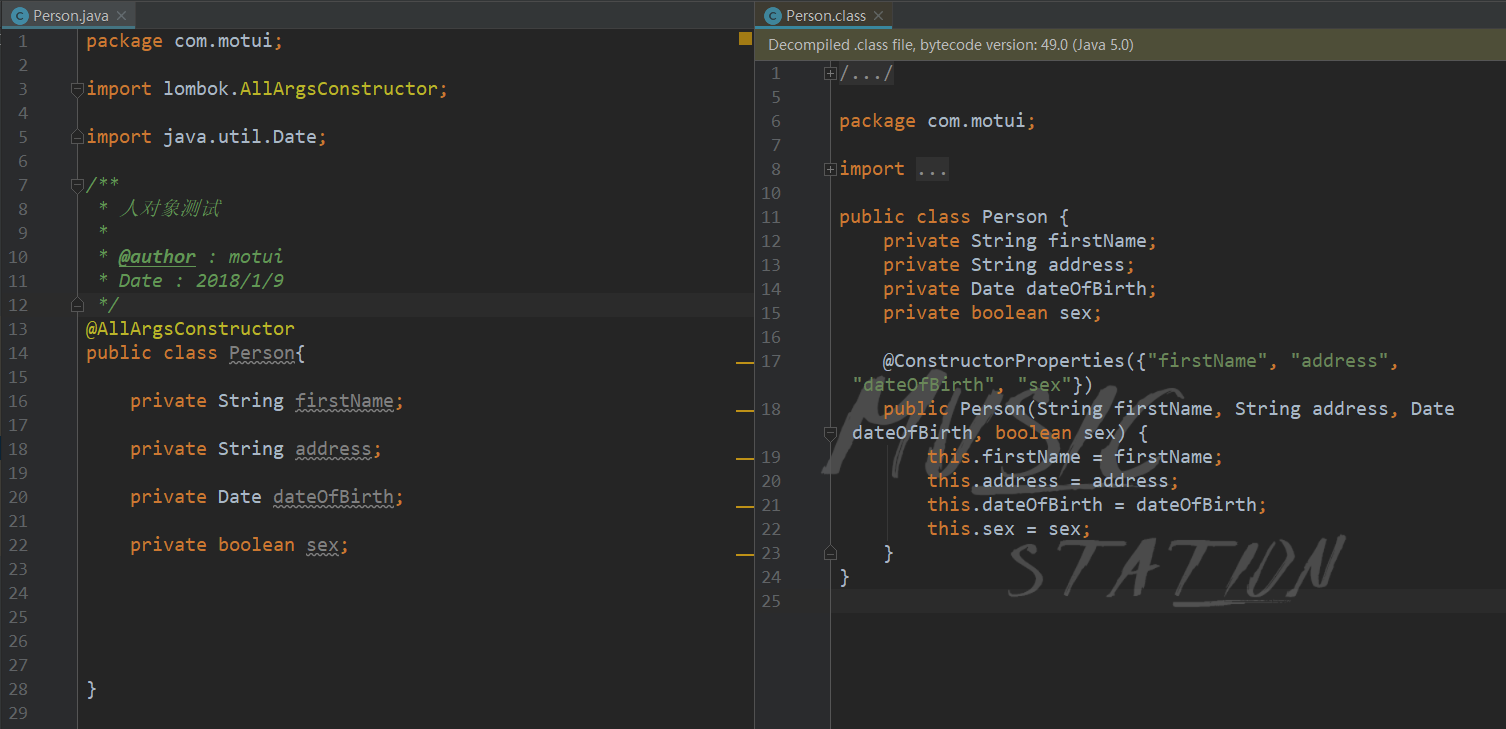
该注解使用在**类**上，该注解在类级别注释会同时生成equals和hashCode。   
注意继承关系的时候该注解的使用。详细介绍参照[官方介绍](http://jnb.ociweb.com/jnb/jnbJan2010.html)   
  
存在继承关系需要设置callSuper参数为true。

**@Data**

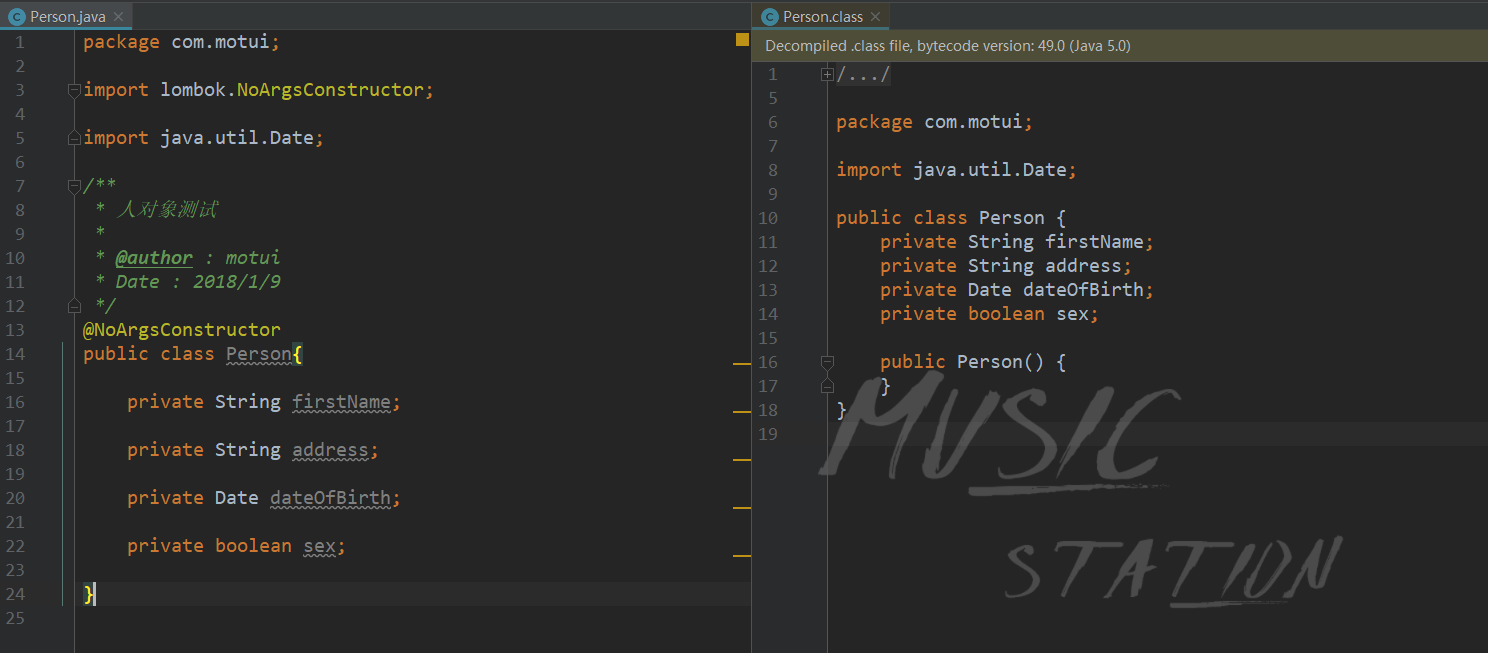
该注解使用在**类**上，该注解是最常用的注解，它结合了@ToString，@EqualsAndHashCode， @Getter和@Setter。本质上使用@Data注解，类默认@ToString和@EqualsAndHashCode以及每个字段都有@Setter和@getter。该注解也会生成一个公共构造函数，可以将任何@NonNull和final字段作为参数。

虽然@Data注解非常有用，但是它没有与其他注解相同的控制粒度。@Data提供了一个可以生成静态工厂的单一参数，将staticConstructor参数设置为所需要的名称，Lombok自动生成的构造函数设置为私有，并提供公开的给定名称的静态工厂方法。   


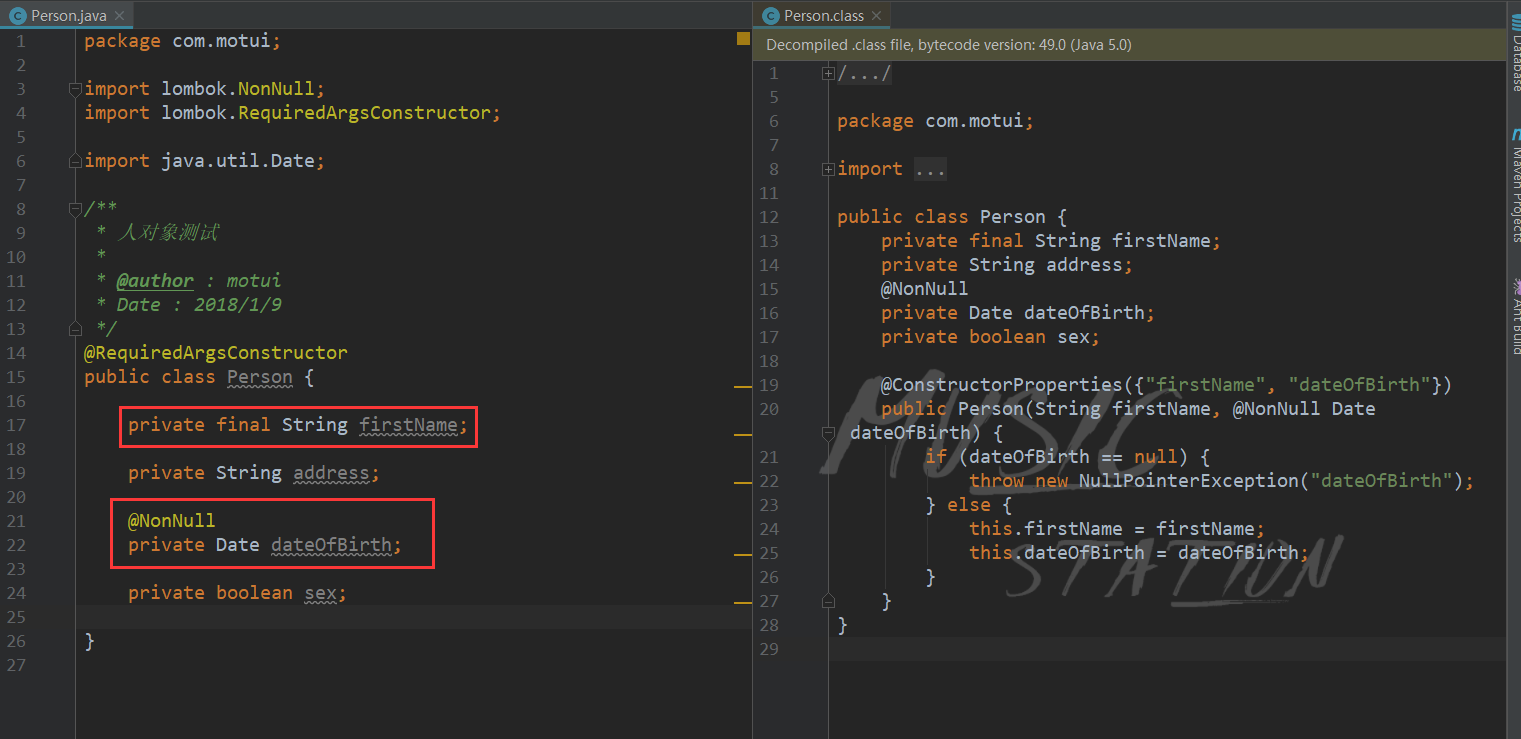
**@AllArgsConstructor**

该注解使用在**类**上，该注解提供一个全参数的构造方法，默认不提供无参构造。   


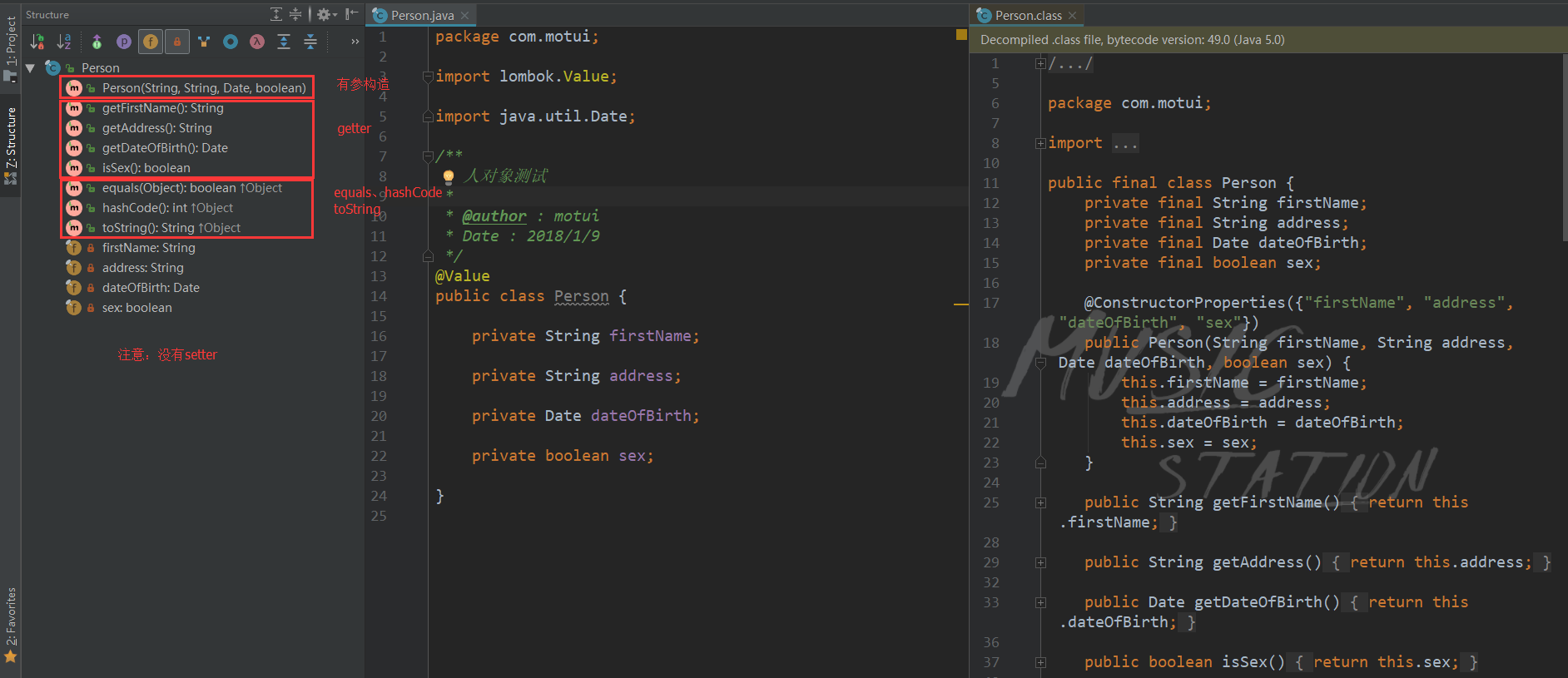
**@NoArgsConstructor**

该注解使用在**类**上，该注解提供一个无参构造   


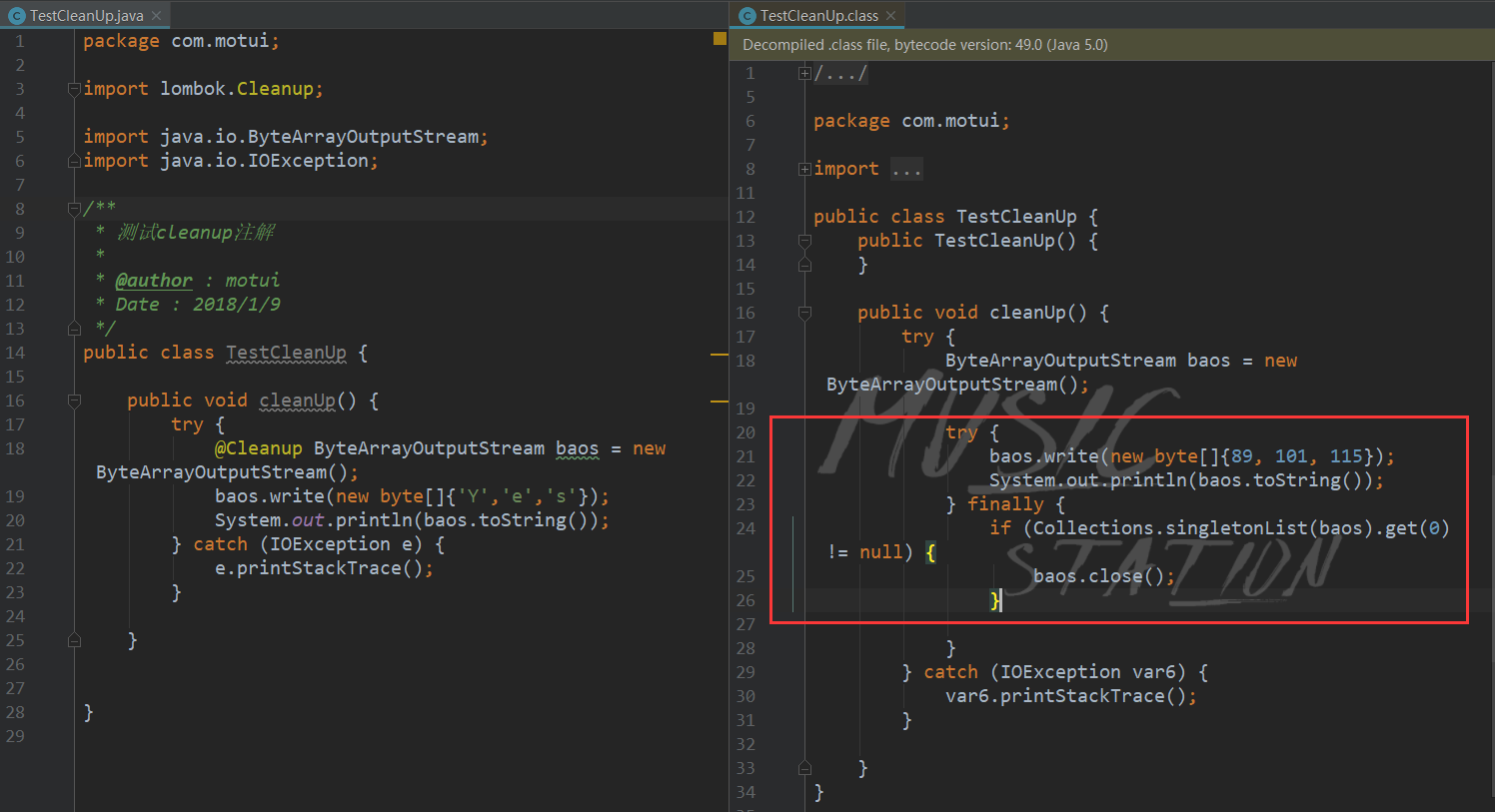
**@RequiredArgsConstructor**

该注解使用在**类**上，使用类中所有带有 @NonNull 注解的或者带有 final 修饰的成员变量生成对应的构造方法。   


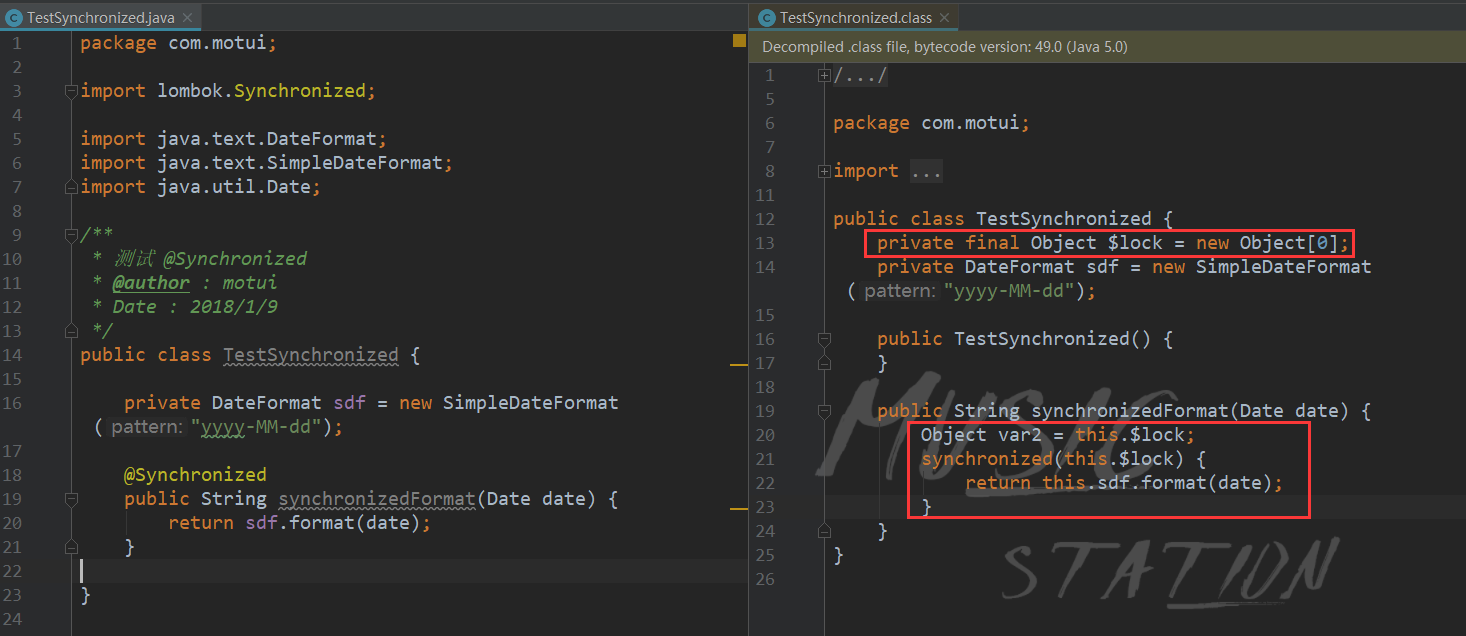
**@Value**

这个注解用在 类 上，会生成含所有参数的构造方法，get 方法，此外还提供了equals、hashCode、toString 方法。   
**注意：没有setter**   


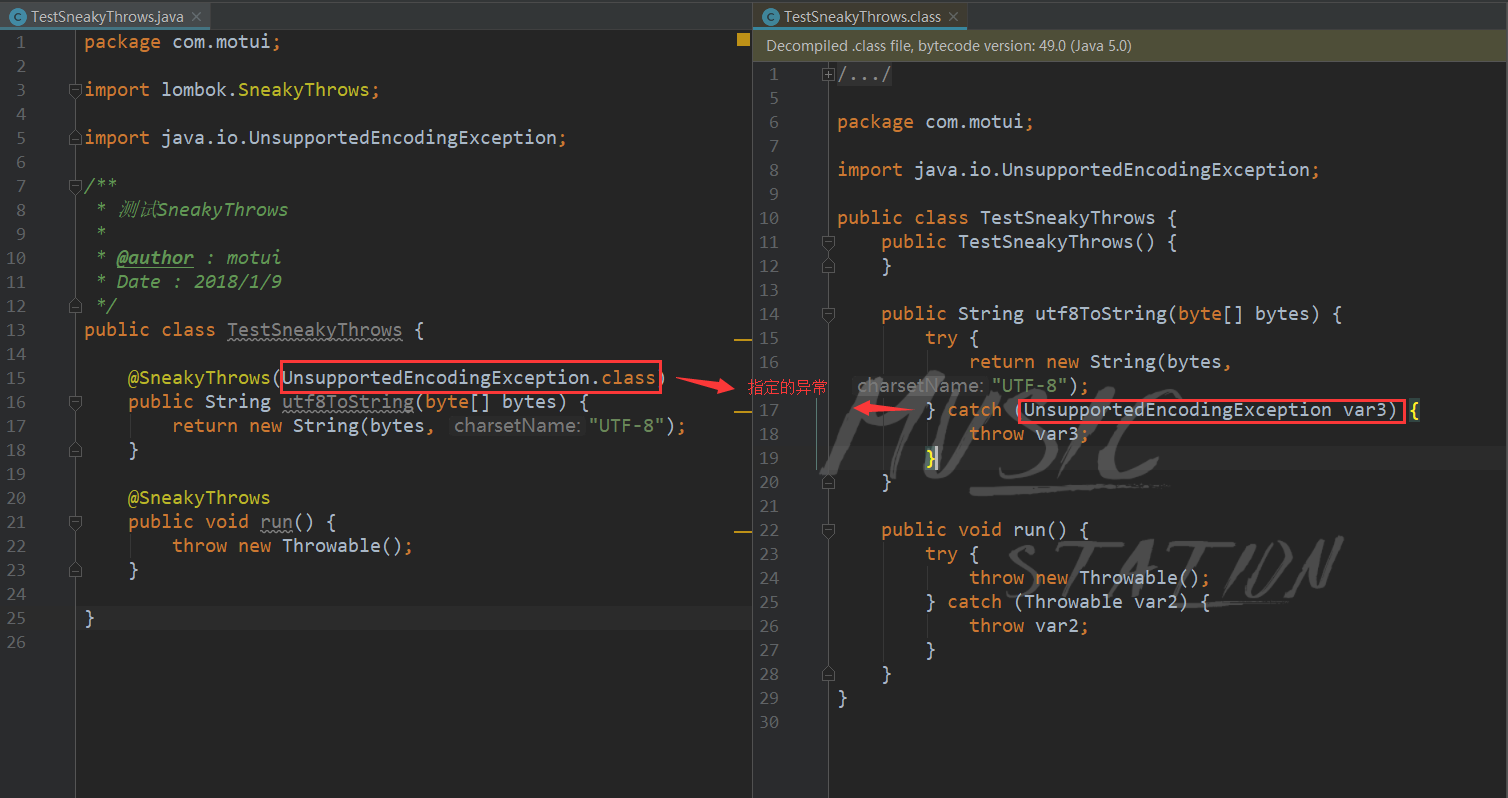
**@Cleanup**

该注解使用在**属性**前，该注解是用来保证分配的资源被释放。在本地变量上使用该注解，任何后续代码都将封装在try/finally中，确保当前作用于中的资源被释放。默认@Cleanup清理的方法为close，可以使用value指定不同的方法名称。   


**@Synchronized**

该注解使用在**类或者实例方法**上，Synchronized在一个方法上，使用关键字可能会导致结果和想要的结果不同，因为多线程情况下会出现异常情况。Synchronized   
关键字将在this示例方法情况下锁定当前对象，或者class讲台方法的对象上多锁定。这可能会导致死锁现象。一般情况下建议锁定一个专门用于此目的的独立锁，而不是允许公共对象进行锁定。该注解也是为了达到该目的。   


**@SneakyThrows**

该注解使用在**方法**上，这个注解用在 方法 上，可以将方法中的代码用 try-catch 语句包裹起来，捕获异常并在 catch 中用 Lombok.sneakyThrow(e) 把异常抛出，可以使用 @SneakyThrows(Exception.class) 的形式指定抛出哪种异常。该注解需要谨慎使用。详情参看[官方介绍](https://projectlombok.org/features/SneakyThrows)   


**结语**

Lombok的目的是减少代码的重复编写，并提供比较好的解决方案。当然也存在一些争议性的注解，可以根据实际场景进项使用。Lombok有一些实验注解可以在官网查看。